NXP Semiconductors
Application Note

Document Number: AN12214
Rev. 0, 12/2018

PMSM Field-Oriented Control on MIMXRT10xx EVK

1 Introduction

This application note describes the implementation of the sensor and sensorless speed and position motor control software for 3-phase Permanent Magnet Synchronous Motors (PMSM), including the motor parameters identification algorithm, on the MIMXRT10xx Evaluation Kit (EVK) based on the NXP i.MX RT10xx processor.

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

The last part of this document introduces and explains the user interface represented by the Motor Control Application Tuning (MCAT) page based on the FreeMASTER runtime debugging tool. These tools represent a simple and user-friendly way of the motor parameters identification, algorithm tuning, software control, debugging, and diagnostics.

Contents

1 Introduction 1
2 Development platform 2
3 MCU features and peripheral settings 2
3.1 i.MX RT1050 EVK/i.MX RT1060 EVK 2
3.2 i.MX RT1020 EVK 7
3.3 CPU load and memory usage 12
4 Motor control peripheral initialization 14
5 Tuning and controlling the application 16
5.1 PMSM parameters identification 18
5.2 PMSM sensor/sensorless application control and tuning using MCAT 23
6 Conclusion 40
7 Acronyms and abbreviations 40
8 References 41

2 Development platform

The FRDM-MC-LVPMSM development platform has the power supply input voltage of 24-48 VDC with the reverse polarity protection circuitry. The auxiliary power supply of 5.5 VDC is created to supply the FRDM MCU boards. The output current is up to 5 A RMS. The inverter itself is realized by the 3-phase bridge inverter (6-MOSFETs) and the 3-phase MOSFET gate driver. The analog quantities (such as the 3-phase motor currents, DC-bus voltage, and DC-bus current) are sensed on this board. There is also an interface for speed and position sensors (Encoder, Hall). The block diagram of a complete NXP Freedom motor control development kit is shown in Figure 1.

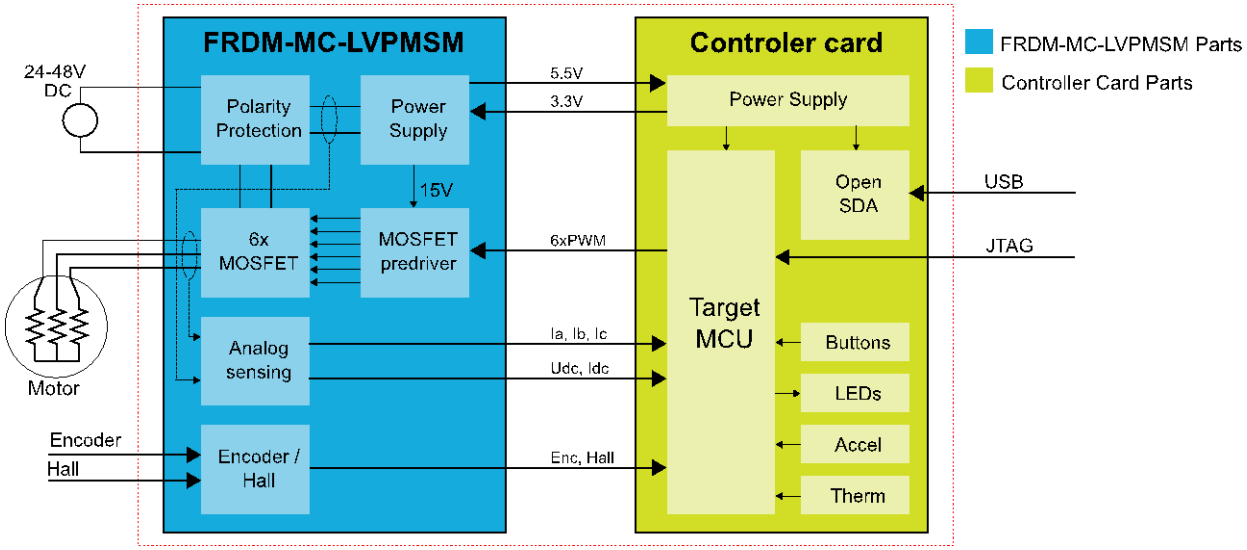

Figure 1. Motor control development platform block diagram

The FRDM-MC-LVPMSM does not require a complicated setup. It is only needed to connect all signals from the i.MX RT10xx EVK to the FRDM-MC-LVPMSM by wires. For more details, see the user's guide. For more information about the NXP Freedom development platform, see www.nxp.com/freedom.

3 MCU features and peripheral settings

This chapter describes the peripheral settings and application timing. The i.MX RT10xx is a new processor family featuring NXP's advanced implementation of the ARM® Cortex®-M7 core which operates at speeds of up to 600 MHz in i.MXRT1050/1060 and up to 500 MHz in i.MXRT1020.

3.1 i.MX RT1050 EVK/i.MX RT1060 EVK

The MIMXRT1050 EVK and MIMXRT1060 EVK boards are platforms designed to showcase the most commonly used features of the i.MX RT10XX processors in small, low-cost packages. The MIMXRT1050 and MIMXRT1060 EVK boards are entry-level development boards, which help you to become familiar with the processors before investing a large amount of resources in more specific designs. The EVK boards provide various types of memory, especially the 64-Mbit Quad SPI flash and 512-Mbit Hyper flash.

3.1.1 Hardware timing and synchronization

Correct and precise timing is crucial for motor control applications. Therefore, the motor control-dedicated peripherals take care of the timing and synchronization on the hardware layer. In addition, it is possible to set the PWM frequency as a multiple of the ADC interrupt (ADC ISR) frequency where the FOC algorithm is calculated. In this case, the PWM frequency is equal to the FOC frequency. The timing diagram is shown in Figure 2.

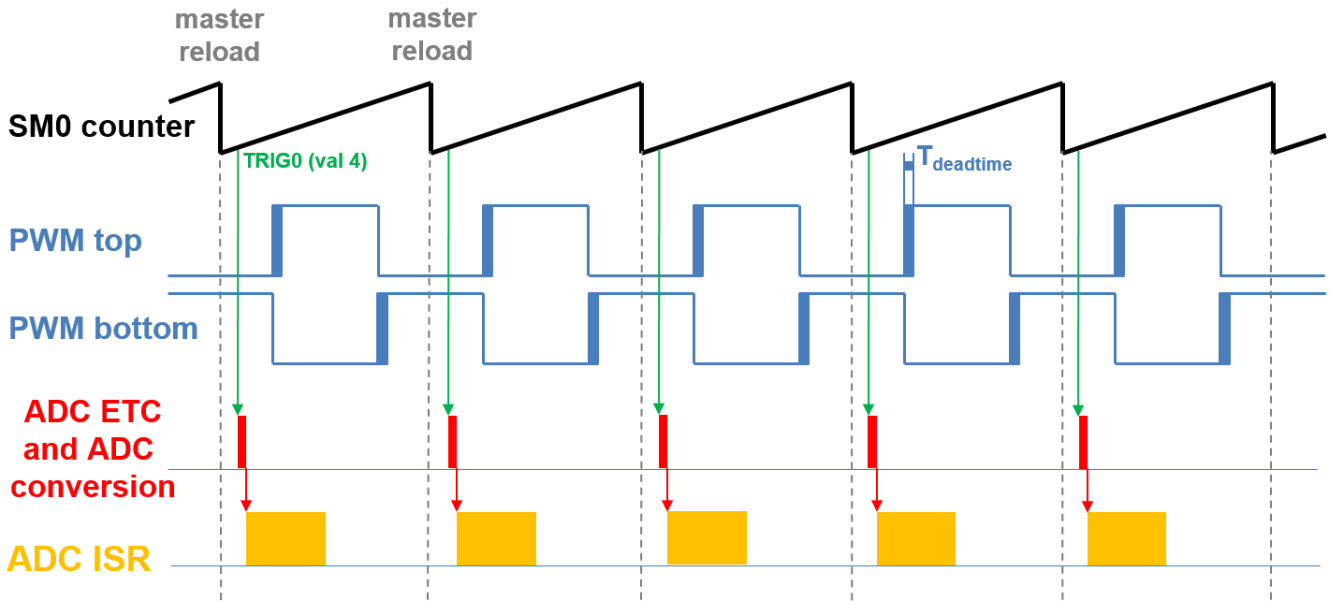

Figure 2. Hardware timing and synchronization on i.MX RT1050/1060

- The top signal shows the eFlexPWM counter (SM0 counter). The dead time is emphasized at the PWM top and PWM bottom signals. The SM0 submodule generates the master reload at every opportunity.
- The SM0 generates trigger 0 (when the counter counts to a value equal to the TRIG4 value) for the ADC_ETC (ADC External Trigger Control) with a delay of approximately $T_{deatime}/2$. This delay ensures correct current sampling at the duty cycles close to 100 %.
- ADC_ETC starts the ADC conversion.
- When the ADC conversion is completed, the ADC ISR (ADC interrupt) is entered. The FOC calculation is done in this interrupt.

3.1.2 Peripheral settings

This section describes the peripherals used for the motor control. On i.MX RT1050/1060, there are three submodules from the enhanced FlexPWM (eFlexPWM) used for 6-channel PWM generation and two 12-bit ADCs for the phase currents and DC-bus voltage measurement. The eFlexPWM and ADC are synchronized via submodule 0 from the eFlexPWM. The following settings are in the *mcdrv_evkbimxrt1050.c* (*mcdrv_evkimxrt1060.c*) and *board.c* files and in their header files.

3.1.2.1 Clock Control Module (CCM)

The CCM generates and controls the clocks of various modules in the design and manages the low-power modes. This module uses the available clock sources to generate the clock roots.

The clock sources used in the motor control application are:

- PLL2, also called System PLL, with a frequency of 528 MHz.
- PLL3, also called USB1 PLL with a frequency of 480 MHz.

The ARM clock core works at a frequency of 528 MHz and the clock source is PLL2. For this setting, the following registers are set: CBCMR[PRE_PERIPH_CLK_SEL], CBCDR[PERIPH_CLK_SEL], and CBCDR[AHB_PODF] in *clock_config.c*. The ADC, XBAR, and PWM are clocked from the IPG_CLK_ROOT output which has a frequency of 132 MHz. The CBCDR[IPG_PODF] register must be set for this setting. The IPG_CLK_ROOT is sourced from the AHB_CLK_ROOT. The LPUART is sourced from the PLL3 at a frequency of 480 MHz divided by 6.

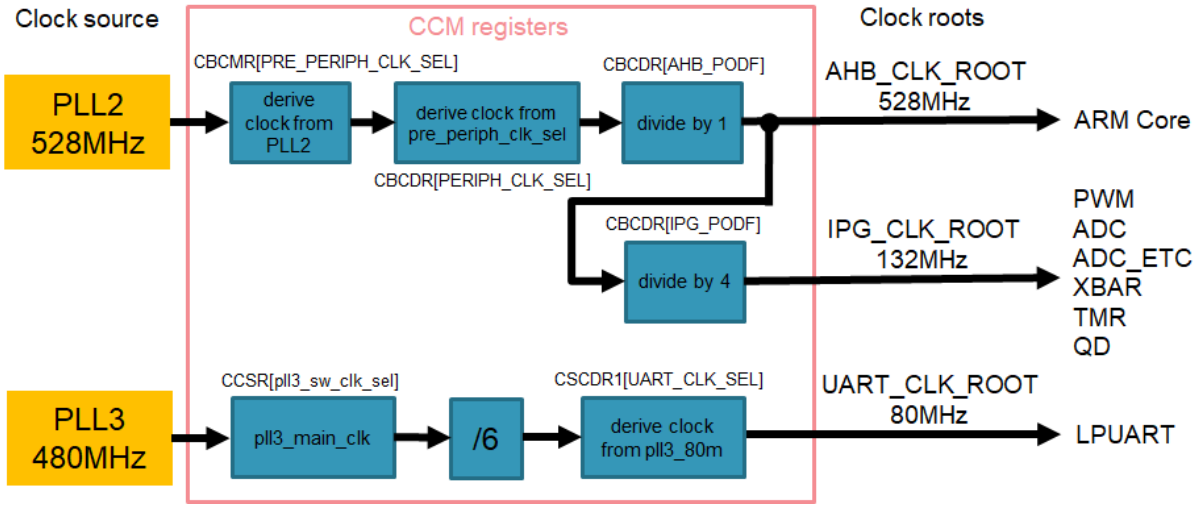

Figure 3. i.MX RT1050/1060 clock source for motor control peripherals

The clock sources for the peripherals used for the motor control are listed in Table 1.

Table 1. i.MX RT1050/1060 clock source for motor control peripherals

—	Clock source	Clock root	Clock root frequency
Arm core	PLL2	AHB_CLK_ROOT	528 MHz
PWM	PLL2	IPG_CLK_ROOT	132 MHz
ADCs	PLL2	IPG_CLK_ROOT	132 MHz
ADC_ETC	PLL2	IPG_CLK_ROOT	132 MHz
XBAR	PLL2	IPG_CLK_ROOT	132 MHz
TMR	PLL2	IPG_CLK_ROOT	132 MHz
QD	PLL2	IPG_CLK_ROOT	132 MHz
LPUART	PLL3	UART_CLK_ROOT	80 MHz

For more details, see the i.MX RT1050 or i.MX RT1060 reference manual.

3.1.2.2 PWM generation—PWM1

- The eFlexPWM is clocked from the 132-MHz IPG_CLK_ROOT.
- Six channels from three submodules are used for the 3-phase PWM generation. Submodule 0 generates the master reload at event every n^{th} opportunity, depending on the user-defined macro M1_FOC_FREQ_VS_PWM_FREQ.
- Submodules 1 and 3 get their clocks from submodule 0.
- The counters at submodules 1 and 3 are synchronized with the master reload signal from submodule 0 (submodule 2 is not used).
- Submodule 0 is used for synchronization with ADC_ETC. The submodule generates the output trigger after the PWM reload, when the counter counts to VAL4.
- Fault mode is enabled for channels A and B at submodules 0, 1, and 3 with automatic fault clearing (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 it takes the counter to count from INIT to VAL1. By default, INIT = -MODULO/2 = -6600 and VAL1 = MODULO/2 -1 = 6599. The eFlexPWM clock is 132 MHz so it takes 0.0001 s (10 kHz).
- Dead time insertion is enabled. The dead time length is defined by the user in the M1_PWM_DEADTIME macro.

3.1.2.3 ADC External Trigger Control—ADC_ETC

The ADC_ETC module enables multiple users to share the ADC modules in the Time Division Multiplexing (TDM) way. The external triggers can be brought from the Cross BAR (XBAR) or other sources. The ADC scan is started via ADC_ETC.

- Both ADCs have set their own trigger chains.
- The trigger chain length is set to 2. The back-to-back ADC trigger mode is enabled.
- The SyncMode is on. In the SyncMode, ADC1 and ADC2 are controlled by the same trigger source. The trigger source is the PWM submodule 0.

3.1.2.4 Analog sensing—ADC1 and ADC2

ADC1 and ADC2 are used for the MC analog sensing of currents and DC-bus voltage.

- The clock frequency for ADC1 and ADC2 is 66 MHz. It is taken from IPG_CLK_ROOT and divided by 2.
- The ADCs operate as 10-bit with the single-ended conversion and hardware trigger selected. The ADCs are triggered from ADC_ETC by the trigger generated by the eFlexPWM.
- The conversion complete interrupt is enabled and serves the FOC fast loop algorithm generated after the last scan is completed by ADC1.

3.1.2.5 Quadrature Decoder module—QD

The QD module is used to sense the position and speed from the encoder sensor.

- The conversion complete interrupt (which serves the FOC fast loop algorithm generated after the last scan is completed at ADC1) is enabled.
- The direction of counting is set by the user in the M1_POSPE_ENC_DIRECTION macro.
- The modulo counting and the modulus counting roll-over/under to increment/decrement revolution counter are enabled.

3.1.2.6 Peripheral interconnection for i.MX RT1050 —XBARA1

The crossbar is used to interconnect the trigger from the PWM to the ADC_ETC and to connect the encoder (connected to GPIO) to the QD.

- The FLEXPWM1_PWM1_OUT_TRIG0_1 output trigger (generated by submodule 0) is connected to ADC_ETC_XBAR0_TRIG0.
- The encoder signal Phase A - IOMUX_XBAR_INOUT14 output is assigned to ENC1_PHASE_A_INPUT (GPIO_AD_B0_00 is configured as XBAR1_INOUT14 in *pinmux.c*).
- The encoder signal Phase B - IOMUX_XBAR_INOUT15 output is assigned to ENC1_PHASE_B_INPUT (GPIO_AD_B0_01 is configured as XBAR1_INOUT15 in *pinmux.c*).

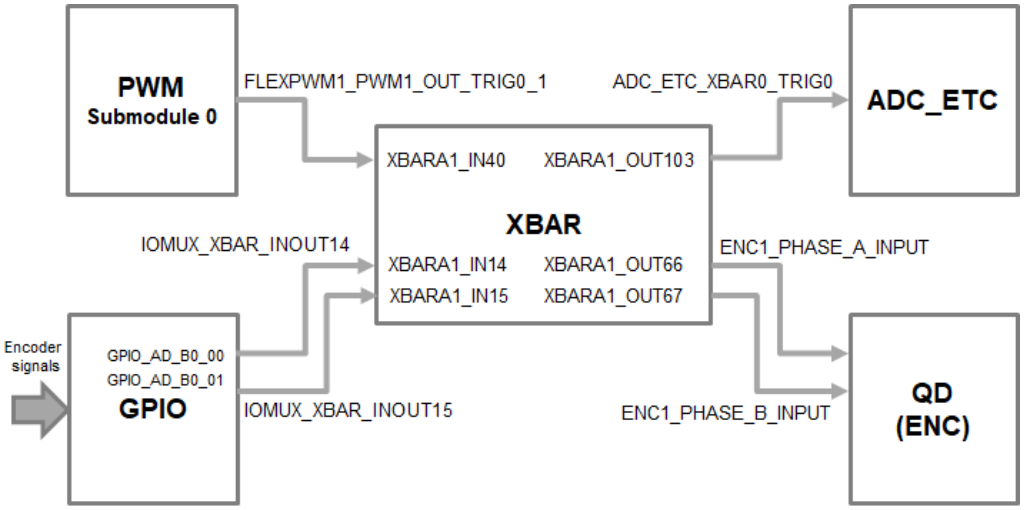

Figure 4. Crossbar interconnection for i.MX RT1050

3.1.2.7 Peripheral interconnection for i.MX RT1060 —XBARA1

The crossbar is used to interconnect the trigger from the PWM to the ADC_ETC and to connect the encoder (connected to GPIO) to the QD.

- The FLEXPWM1_PWM1_OUT_TRIG0_1 output trigger (generated by submodule 0) is connected to ADC_ETC_XBAR0_TRIG0.
- The encoder signal Phase A - IOMUX_XBAR_INOUT16 output is assigned to ENC1_PHASE_A_INPUT (GPIO_AD_B0_02 is configured as XBAR1_INOUT16 in *pinmux.c*).
- The encoder signal Phase B - IOMUX_XBAR_INOUT17 output is assigned to ENC1_PHASE_B_INPUT (GPIO_AD_B0_03 is configured as XBAR1_INOUT17 in *pinmux.c*).

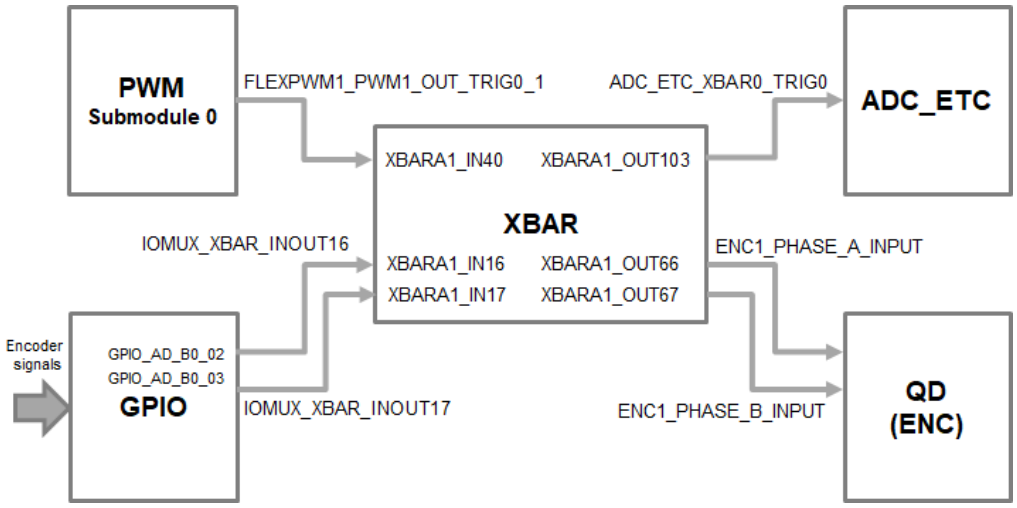

Figure 5. Crossbar interconnection for i.MX RT1060

3.1.2.8 Slow loop interrupt generation—TMR1

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

- QuadTimer TMR1 is clocked from IPG CLK ROOT divided by 16, so the clock frequency of TMR1 is 8.25 MHz.
- The slow loop is usually ten times slower than the fast loop. Therefore, the interrupt is generated after the counter counts from CNTR0 = 0 to COMP1 = IPG CLK ROOT / (16U * Speed Loop Freq). The speed loop frequency is set in the M1_SPEED_LOOP_FREQ macro and equals 1000 Hz.
- An interrupt (which serves the slow loop period) is enabled and generated at the reload event.

3.1.2.9 FreeMASTER communication—LPUART0

LPUART0 (Low-Power Universal Asynchronous Receiver and Transmitter) is used for the FreeMASTER communication between the MCU board and the PC.

- The baud rate is set to 115200 bit/s.
- The receiver and transmitter are both enabled.
- The other settings are set to default.

3.2 i.MX RT1020 EVK

The i.MX RT1020 EVK is a 2-layer low-cost through-hole USB-powered PCB. At its heart lies the i.MX RT1020 crossover processor in a LQFP144 package.

The i.MX RT1020 expands the i.MX RT crossover processor families by providing high-performance feature set in low-cost LQFP packages, further simplifying board design and layout for customers. The i.MX RT1020 contains the Arm Cortex-M7 core. This core operates at speeds of up to 500 MHz to provide high CPU performance and best real-time response.

3.2.1 Hardware timing and synchronization

Correct and precise timing is crucial for motor control applications. Therefore, the motor control-dedicated peripherals take care of the timing and synchronization on the hardware layer. In addition, it is possible to set the PWM frequency as a multiple of the ADC interrupt (ADC ISR) frequency where the FOC algorithm is calculated. In this case, the PWM frequency is equal to the FOC frequency. The timing diagram is shown in Figure 6.

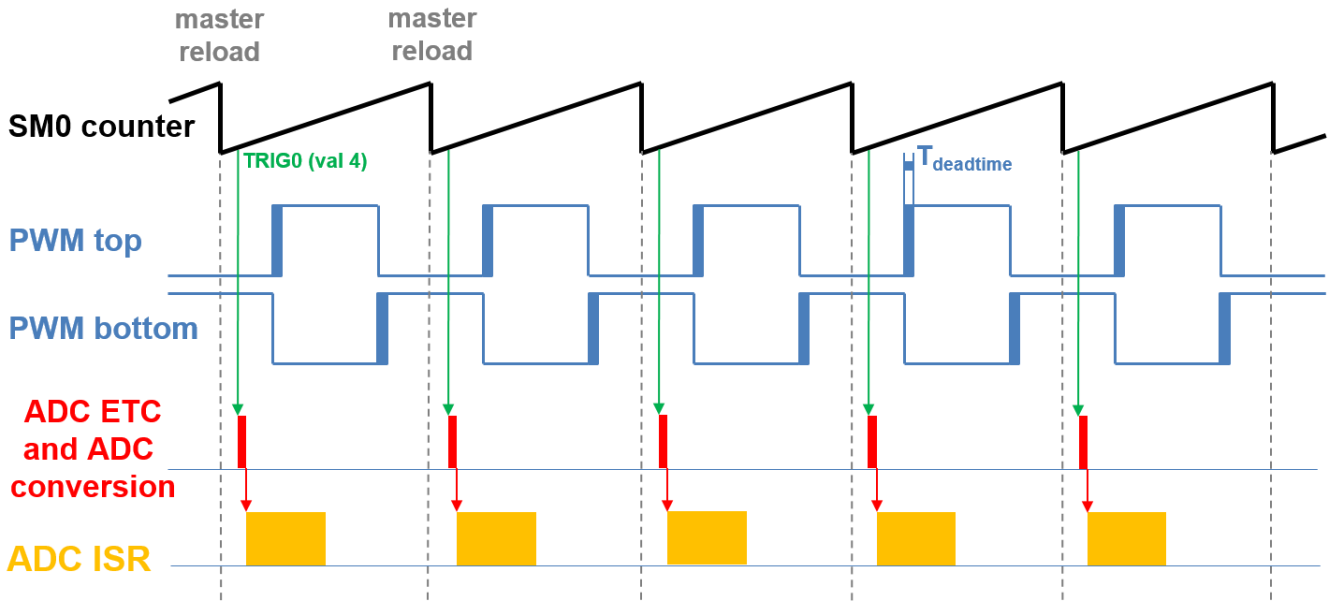

Figure 6. Hardware timing and synchronization on i.MX RT1020

- The top signal shows the eFlexPWM counter (SM0 counter). The dead time is emphasized at the PWM top and PWM bottom signals. The SM0 submodule generates the master reload at every opportunity.
- The SM0 generates trigger 0 (when the counter counts to a value equal to the TRIG4 value) for the ADC_ETC (ADC External Trigger Control) with a delay of approximately $T_{deatime}/2$. This delay ensures correct current sampling at the duty cycles close to 100 %.
- ADC_ETC starts the ADC conversion.
- When the ADC conversion is completed, the ADC ISR (ADC interrupt) is entered. The FOC calculation is done in this interrupt.

3.2.2 Peripheral settings

This section describes the peripherals used for the motor control. On i.MX RT1020, there are three submodules from the enhanced FlexPWM (eFlexPWM) used for 6-channel PWM generation and two 12-bit ADCs for the phase currents and DC-bus voltage measurement. The eFlexPWM and ADC are synchronized via submodule 0 from the eFlexPWM. The following settings are in the *mcdrv_imxrt1020.c* and *board.c* files and in their header files.

3.2.2.1 Clock Control Module (CCM)

The CCM generates and controls the clocks of various modules in the design and manages the low-power modes. This module uses the available clock sources to generate the clock roots.

The clock sources used in the motor control application are:

- PLL6, also called System PLL, with a frequency of 500 MHz.
- PLL3, also called USB1 PLL, with a frequency of 480 MHz.

The Arm clock core works at a frequency of 500 MHz and the clock source is PLL6. For this setting, these registers are set: CBCMR[PRE_PERIPH_CLK_SEL], CBCDR[PERIPH_CLK_SEL], and CBCDR[AHB_PODF] in *clock_config.c*. The ADC, XBAR, and PWM are clocked from the IPG_CLK_ROOT output which has a frequency of 125 MHz. The CBCDR[IPG_PODF] register must be set for this setting. The IPG_CLK_ROOT is sourced from the AHB_CLK_ROOT. The LPUART is sourced from the PLL3 at a frequency of 480 MHz divided by 6.

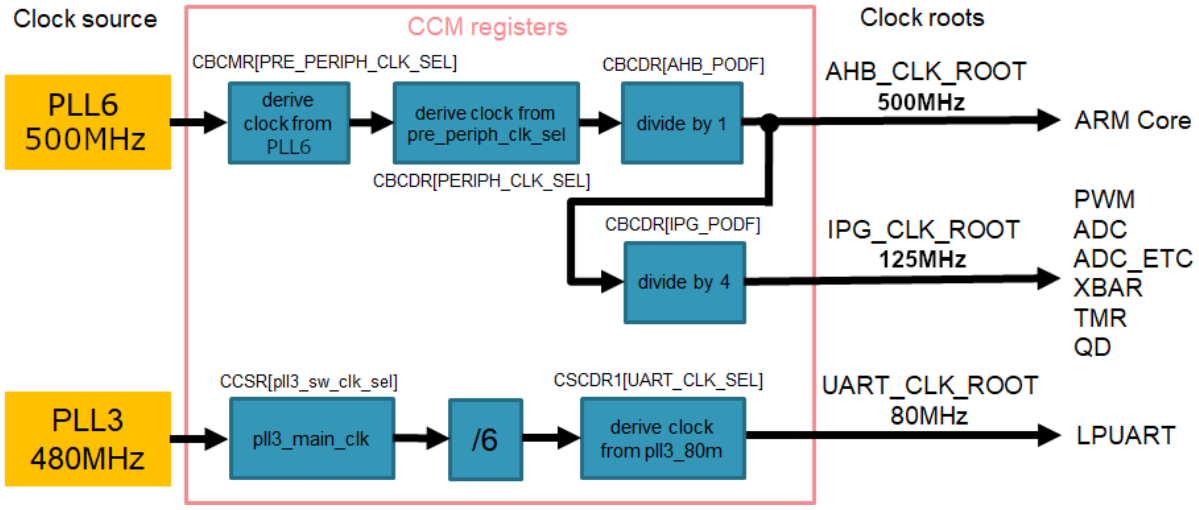

Figure 7. i.MX RT1020 clock source for motor control peripherals

The clock sources for the peripherals used for the motor control are listed in Table 2.

Table 2. i.MX RT1020 clock source for motor control peripherals

—	Clock source	Clock root	Clock root frequency
Arm core	PLL6	AHB_CLK_ROOT	500 MHz
PWM	PLL6	IPG_CLK_ROOT	125 MHz
ADCs	PLL6	IPG_CLK_ROOT	125 MHz
ADC_ETC	PLL6	IPG_CLK_ROOT	125 MHz
XBAR	PLL6	IPG_CLK_ROOT	125 MHz
TMR	PLL6	IPG_CLK_ROOT	125 MHz
QD	PLL6	IPG_CLK_ROOT	125 MHz
LPUART	PLL3	UART_CLK_ROOT	80 MHz

For more details, see the *i.MX RT1020 Processor Reference Manual* (document IMXRT1020RM).

3.2.2.2 PWM generation—PWM2

- The eFlexPWM is clocked from the 125-MHz IPG_CLK_ROOT.
- Six channels from three submodules are used for the 3-phase PWM generation. Submodule 0 generates the master reload at event every n^{th} opportunity, depending on the user-defined macro M1_FOC_FREQ_VS_PWM_FREQ.
- Submodules 1 and 3 get their clocks from submodule 0.
- The counters at submodules 1 and 3 are synchronized with the master reload signal from submodule 0 (submodule 2 is not used).
- Submodule 0 is used for synchronization with ADC_ETC. The submodule generates the output trigger after the PWM reload, when the counter counts to VAL4.
- Fault mode is enabled for channels A and B at submodules 0, 1, and 3 with automatic fault clearing (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 it takes the counter to count from INIT to VAL1. By default, INIT = -MODULO/2 = -6250 and VAL1 = MODULO/2 -1 = 6249. The eFlexPWM clock is 125 MHz so it takes 0.0001 s (10 kHz).
- Dead time insertion is enabled. The dead time length is defined by the user in the M1_PWM_DEADTIME macro.

3.2.2.3 ADC External Trigger Control—ADC_ETC

The ADC_ETC module enables multiple users to share the ADC modules in the Time Division Multiplexing (TDM) way. The external triggers can be brought from the Cross BAR (XBAR) or other sources. The ADC scan is started via ADC_ETC.

- Both ADCs have their own trigger chains.
- The trigger chain length is set to 2. The back-to-back ADC trigger mode is enabled.

- The SyncMode is on. In the SyncMode, ADC1 and ADC2 are controlled by the same trigger source. The trigger source is PWM submodule 0.

3.2.2.4 Analog sensing—ADC1 and ADC2

ADC1 and ADC2 are used for the MC analog sensing of the currents and DC-bus voltage.

- The clock frequency for ADC1 and ADC2 is 62.5 MHz. It is taken from IPG_CLK_ROOT and divided by 2.
- The ADCs operate as 10-bit with the single-ended conversion and hardware trigger selected. The ADCs are triggered from ADC_ETC by the trigger generated by eFlexPWM.
- The conversion complete interrupt is enabled and serves the FOC fast loop algorithm generated after the last scan is completed by ADC1.

3.2.2.5 Quadrature Decoder module—QD

The QD module is used to sense the position and speed from the encoder sensor.

- The conversion complete interrupt (which serves the FOC fast loop algorithm generated after the last scan is completed at ADC1) is enabled.
- The direction of counting is set by the user in the M1_POSPE_ENC_DIRECTION macro.
- The modulo counting and the modulus counting roll-over/under to increment/decrement revolution counter are enabled.

3.2.2.6 Peripheral interconnection—XBARA1

The crossbar is used to connect the trigger from PWM to ADC_ETC and to connect the encoder (connected to GPIO) to the QD module.

- The FLEXPWM2_PWM1_OUT_TRIG0_1 output trigger (generated by submodule 0) is connected to ADC_ETC_XBAR0_TRIG0.
- The encoder signal Phase A - IOMUX_XBAR_INOUT12 output is assigned to ENC1_PHASE_A_INPUT (GPIO_AD_B1_08 is configured as XBAR1_INOUT12 in *pinmux.c*).
- The encoder signal Phase B - IOMUX_XBAR_INOUT13 output is assigned to ENC1_PHASE_B_INPUT (GPIO_AD_B1_09 is configured as XBAR1_INOUT13 in *pinmux.c*).

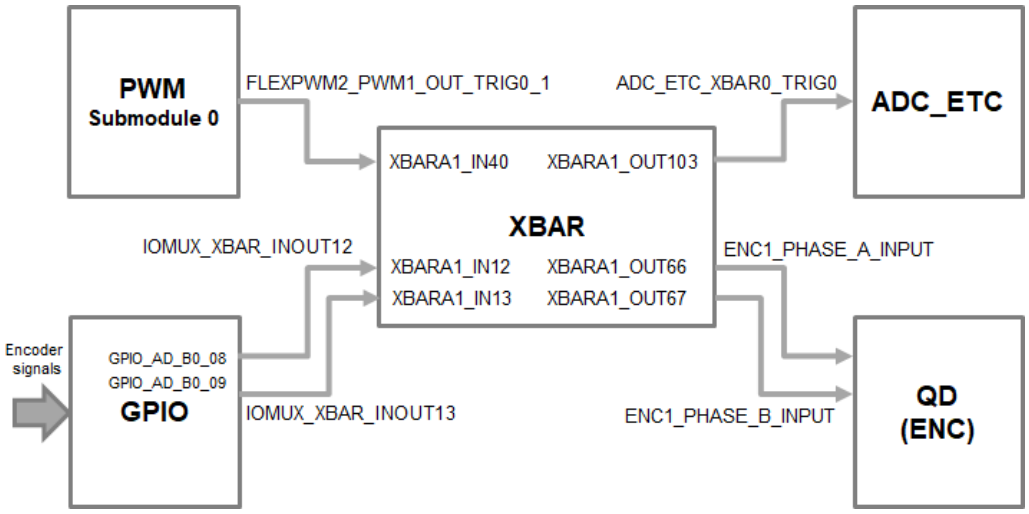

Figure 8. Crossbar interconnection for i.MX RT1020

3.2.2.7 Slow loop interrupt generation—TMR1

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

- QuadTimer TMR1 is clocked from IPG CLK ROOT divided by 16, so the clock frequency of TMR1 is 7.8125 MHz.
- The slow loop is usually ten times slower than the fast loop. Therefore, the interrupt is generated after the counter counts from CNTR0 = 0 to COMP1 = IPG CLK ROOT / (16U * Speed Loop Freq). The speed loop frequency is set in the M1_SPEED_LOOP_FREQ macro and equals 1000 Hz.
- An interrupt (which serves the slow loop period) is enabled and generated at the reload event.

3.2.2.8 FreeMASTER communication—LPUART0

LPUART0 (Low-Power Universal Asynchronous Receiver and Transmitter) is used for the FreeMASTER communication between the MCU board and the PC.

- The baud rate is set to 115200 bit/s.
- The receiver and transmitter are both enabled.
- The other settings are set to default.

3.3 CPU load and memory usage

The following information apply to the application built using the MCUXpresso IDE in the Debug RAM and Release FLASH configurations. Tables below shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including the 2-KB FreeMASTER recorder buffer allocated in RAM. The CPU load is measured using the SysTick timer. The CPU load is dependent on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 10 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated using following equations:

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

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

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

Where the used characters are:

CPU_{fast} —the CPU load taken by the fast loop (ADC ISR).

$cycles_{fast}$ —the number of cycles consumed by the fast loop.

f_{fast} —the frequency of the fast loop calculation (10 kHz).

f_{CPU} —the CPU frequency.

CPU_{slow} —the CPU load taken by the slow loop (TMR ISR).

$cycles_{slow}$ —the number of cycles consumed by the slow loop.

f_{slow} —the frequency of the slow loop calculation (1 kHz).

CPU_{total} —the total CPU load taken by the motor control.

Table 3. i.MX RT1050 CPU load and memory usage

	Debug RAM		Release FLASH	
Board FLASH	-	-	51440 B	Usage 0.08%
Board SRAM	-	-	-	-
SRAM_DTC	85572 B	Usage 65.29%	23076 B	Usage 17.61%
SRAM_ITC	-	-	10640 B	Usage 8.12%
SRAM_OC	-	-	-	-
	Speed Control	**Position Control**	**Speed Control**	**Position Control**
Maximum CPU load	7.31%	7.84%	7.25%	6.84%

Table 4. i.MX RT1060 CPU load and memory usage

	Debug RAM		Release FLASH	
Board FLASH	-	-	49028 B	Usage 0.58%
Board SRAM	-	-	-	-
SRAM_DTC	8 KB	Usage 6.25%	23060 B	Usage 17.59%
SRAM_ITC	-	-	9672 B	Usage 7.38%
SRAM_OC	83484 B	Usage 10.62%	-	-
	Speed Control	**Position Control**	**Speed Control**	**Position Control**
Maximum CPU load	7.31%	7.84%	7.25%	6.84%

Table 5. i.MX RT1020 CPU load and memory usage

	Debug RAM		Release FLASH	
Board FLASH	-	-	69376 B	Usage 0.83%
Board SRAM	-	-	-	-
SRAM_DTC	-	-	23080 B	Usage 35.22%
SRAM_ITC	-	-	-	-
SRAM_OC	88752 B	Usage 67.71%	-	-
	Speed Control	**Position Control**	**Speed Control**	**Position Control**
Maximum CPU load	7.90%	7.73%	7.70%	7.23%

4 Motor control peripheral initialization

The motor control peripherals are initialized by calling the *MCDRV_Init_M1()* function during MCU startup and before the peripherals are used. All initialization functions are in the *mcdrv_imxrt10xx.c* source file and the *mcdrv_imxrt10xx.h* header file. The definitions specified by the user are also in these files. The features provided by the functions are the 3-phase PWM generation and 3-phase current measurement, as well as the DC-bus voltage and auxiliary quantity measurement. The principles of both the 3-phase current measurement and the PWM generation using the Space Vector Modulation (SVM) technique are described in *Sensorless PMSM Field-Oriented Control* (document DRM148).

The *mcdrv_imxrt10xx.h* header file provides several macros, which can be defined by the user:

- *M1_MCDRV_ADC*—this macro specifies which ADC periphery is used. If you select an unsupported periphery, the preprocessor error is issued.
- *M1_MCDRV_PWM3PH*—this macro specifies which PWM periphery is used. If you select an unsupported periphery, the preprocessor error is issued.
- *M1_MCDRV_QD_ENC*—this macro specifies which QD periphery is used. If the user selects unsupported periphery, the preprocessor error is issued.
- *M1_PWM_FREQ*—the value of this definition sets the PWM frequency.
- *M1_FOC_FREQ_VS_PWM_FREQ*—enables you to call the fast loop interrupt at every first, second, third, or n^{th} PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast loop interrupt.
- *M1_SPEED_LOOP_FREQ* —the value of this definition sets the speed loop frequency (TMR1 interrupt).
- *M1_PWM_DEADTIME*—the value of the PWM dead time in nanoseconds.
- *M1_PWM_PAIR_PH[A..C]*—these macros enable a simple assignment of the physical motor phases to the PWM periphery channels (or submodules). Change the order of the motor phases this way.
- *M1_ADC[1,2]_PH_[A..C]*—these macros are used to assign the ADC channels for the phase current measurement. The general rule is that at least one of the phase currents must be measurable on both ADC converters and the two remaining phase currents must be measurable on different ADC converters. The reason for this is that the selection of the phase current pair to measure depends on the current SVM sector. If this rule is broken, a preprocessor error is issued. For more information

about the 3-phase current measurement, see *Sensorless PMSM Field-Oriented Control* (document DRM148).

- *M1_ADC[1,2]_UDCB*—this define is used to select the ADC channel for the measurement of the DC-bus voltage.

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

- The available APIs for the ADC are:
  - *mcdrv_adc_t*—MCDRV ADC structure data type.
  - *bool_t M1_MCDRV_ADC_PERIPH_INIT()*—this function is by default called during the ADC peripheral initialization procedure invoked by the *MCDRV_Init_M1()* function and should not be called again after the peripheral initialization is done.
  - *bool_t M1_MCDRV_CURR_3PH_CHAN_ASSIGN(mcdrv_adc_t*)*—calling this function assigns proper ADC channels for the next 3-phase current measurement based on the SVM sector. The function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB_INIT(mcdrv_adc_t*)*—this function initializes the phase-current channel-offset measurement. This function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB(mcdrv_adc_t*)*—this function reads the current information from the unpowered phases of a stand-still motor and filters them using moving average filters. The goal is to obtain the value of the measurement offset. The length of the window for moving the average filters is set to eight samples by default. This function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB_SET(mcdrv_adc_t*)*—this function asserts the phase-current measurement offset values to the internal registers. Call this function after a sufficient number of *M1_MCDRV_CURR_3PH_CALIB()* calls. This function always returns true.
  - *bool_t M1_MCDRV_ADC_GET(mcdrv_adc_t*)*—this function reads and calculates the actual values of the 3-phase currents, DC-bus voltage, and auxiliary quantity. This function always returns true.
- The available APIs for the PWM are:
  - *mcdrv_pwma_pwm3ph_t*—MCDRV PWM structure data type.
  - *bool_t M1_MCDRV_PWM_PERIPH_INIT*—this function is by default called during the PWM periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
  - *bool_t M1_MCDRV_PWM3PH_SET(mcdrv_pwma_pwm3ph_t*)*—this function updates the PWM phase duty cycles based on the required values stored in the M1_MCDRV_PWMIO_DUTY variable. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_EN(mcdrv_pwma_pwm3ph_t*)*—calling this function enables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_DIS (mcdrv_pwma_pwm3ph_t*)*—calling this function disables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwma_pwm3ph_t*)*—this function returns the state of the over-current fault flags and automatically clears the flags (if set). This function returns true when an over-current event occurs. Otherwise, it returns false.
- The available API for Quadrature encoder are:
  - *mcdrv_qd_enc_t*—MCDRV QD structure data type.

- *bool_t M1_MCDRV_QD_PERIPH_INIT()*—this function is by default called during the QD periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
- *bool_t M1_MCDRV_QD_GET(mcdrv_qd_enc_t*)*—this function returns the actual position and speed. This function always returns true.
- *bool_t M1_MCDRV_QD_SET_DIRECTION(mcdrv_qd_enc_t*)*—this function sets the direction of the quadrature encoder. This function always returns true.
- *bool_t M1_MCDRV_QD_CLEAR(mcdrv_qd_enc_t*)*—this function clears the internal variables and decoder counter. This function always returns true.

5 Tuning and controlling the application

This section provides information about the tools and recommended procedures for controlling the sensor/sensorless PMSM Field-Oriented Control (FOC) application. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool 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. For more information, see www.nxp.com/freemaster.

The PMSM sensor/sensorless FOC application can be easily controlled and tuned using the Motor Control Application Tuning (MCAT) plug-in for PMSM. The MCAT for PMSM is a user-friendly modular page, which runs within FreeMASTER. To launch it, simply execute the *.pmp* file located next to your project. The tool consists of the tab menu, tuning mode selector, and workspace shown in Figure 9. Each tab from the tab menu represents one sub-module which enables you to tune or control different aspects of the application. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the project tree are predefined in the FreeMASTER project file to further simplify the motor parameter tuning and debugging. The “Basic” and “Expert” tuning modes are available. Selecting the “Expert” mode grants you the access to modify all parameters and fields available in the MCAT.

The “Basic” mode is recommended for inexperienced users. When the FreeMASTER is not connected to the target, the “App ID” line shows “offline”. When the communication with the target MCU using a correct software is established, the “App ID” line displays the board name “pmsm_evk_imxrt10xx” and all stored parameters for the given MCU are loaded.

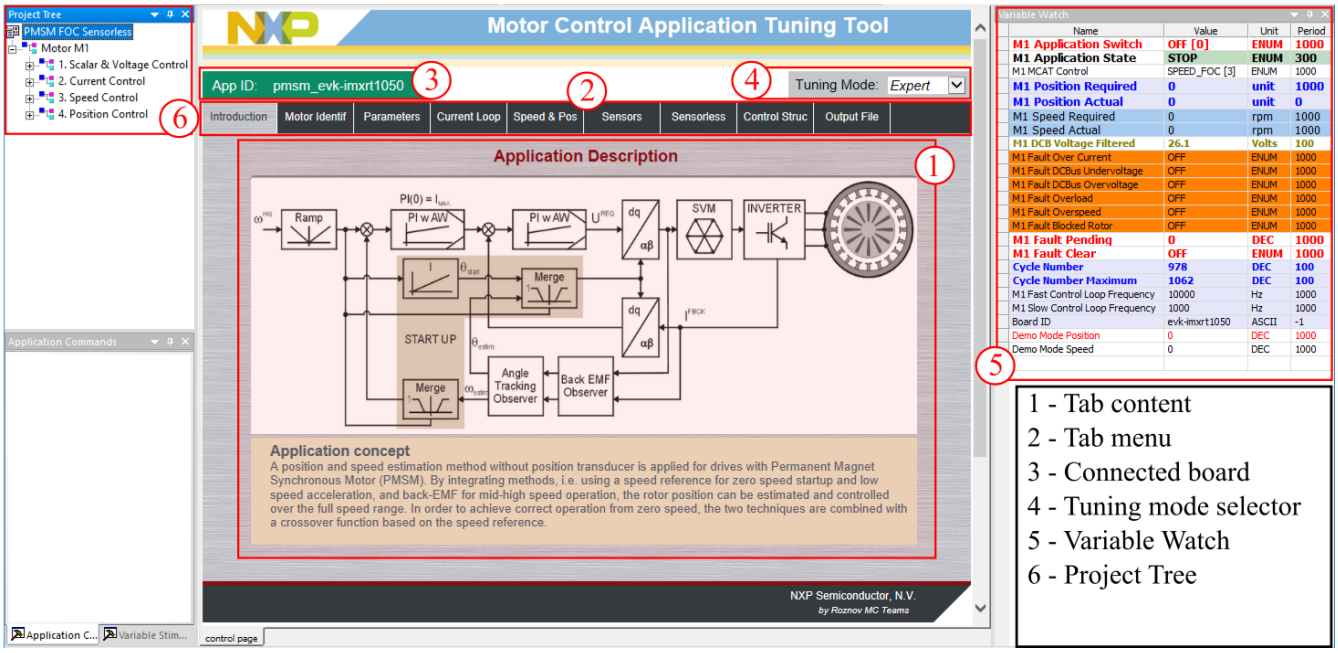

Figure 9. MCAT layout

In the default configuration, the following tabs are available:

- "Introduction"—welcome page with the PMSM sensor/sensorless FOC diagram and a short description of the application.
- "Motor Identif"—PMSM semi-automated parameter measurement control page. The PMSM parameter identification is more closely described further on in this document.
- "Parameters"—this page enables you to modify the motor parameters, specification of hardware and application scales, alignment, and fault limits.
- "Current Loop"—current loop PI controller gains and output limits.
- "Speed & Pos"—this tab contains fields for the specification of speed controller proportional and integral gains, as well as the output limits and parameters of the speed ramp. The position proportional controller constant is also set here.
- "Sensors"—this page contains the encoder parameters and the position observer parameters.
- "Sensorless"—this page enables you to tune the parameters of the BEMF observer, tracking observer, and open loop startup.
- "Control Struc"—this application control page enables you to select and control the PMSM using different techniques (scalar—Volt/Hertz control, voltage FOC, current FOC, speed FOC, and position FOC). The application state is also shown on this tab.
- "Output file"—this tab shows all the calculated constants that are required by the PMSM sensor/sensorless FOC application. It is also possible to generate the *m1_acim_appconfig.h* file, which is then used to preset all application parameters permanently at the project rebuild.
- "App page"—this tab contains the graphical elements like the speed gauge, DC-bus voltage measurement bar, and variety of switches which enable a simple, quick, and user-friendly application control. The fault clearing and the demo mode (which sets various predefined required speeds and positions over time) can be also controlled from here.

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

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

5.1 PMSM parameters identification

Because the model-based control methods of the PMSM drives are the most effective and usable, obtaining an accurate model of a motor is an important part of the drive design and control. For the implemented FOC algorithms, it is necessary to know the value of the stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and BEMF constant K_e.

5.1.1 Power stage characterization

Each inverter introduces the total error voltage U_{error} which is caused by the dead time, current clamping effect, and transistor voltage drop. The total error voltage U_{error} depends on the phase current i_s and this dependency is measured during the power stage characterization process. An example of the inverter error characteristic is shown in Figure 10. The power stage characterization is a part of the MCAT and can be controlled from the “Motor Identif” tab. To perform the characterization, connect the motor with the known stator resistance R_s and set this value in the “Calib Rs” field. Then specify the “Calibration Range”, which is the range of the stator current i_s, in which the measurement of U_{error} is performed. Start the characterization by pressing the “Calibrate” button. The characterization gradually performs 65 i_{sd} current steps (from $i_s = -I_{s,calib}$ to $i_s = I_{s,calib}$) with each taking 300 ms, so be aware that the process takes about 20 seconds and the motor must withstand this load. The acquired characterization data is saved to a file and used later for the phase voltage correction during the R_s measurement process. The following R_s measurement can be done with maximum current $I_{s,calib}$. It is recommended to use a motor with a low R_s for characterization purposes.

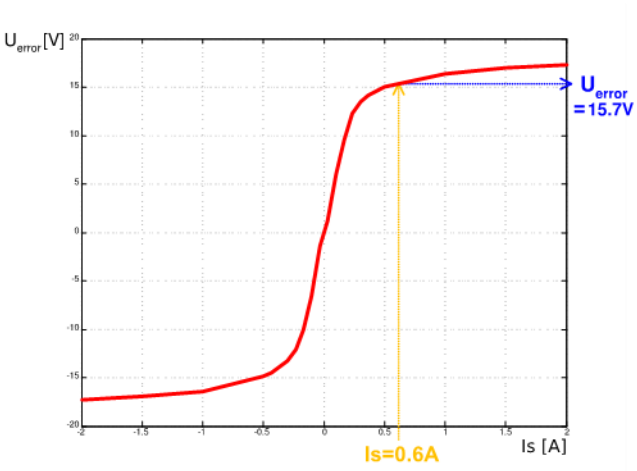

Figure 10. Example power stage characteristic

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

5.1.2 Stator resistance measurement

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

$$R_s = \frac{U_{DC} - U_{error}}{I_{phN}} [\Omega] \quad \textit{Eq. 4}$$

5.1.3 Stator inductance

For the stator inductance L_S identification purposes, a sinusoidal measurement voltage is applied to the motor. During the L_S measurement, the voltage control is enabled. The frequency and amplitude of the sinusoidal voltage are obtained before the actual measurement, during the tuning process. The tuning process begins with a 0-V amplitude and the *F start* frequency, which are applied to the motor. The amplitude is gradually increased by *Ud inc* up to a half of the DC-bus voltage (DCbus/2) until *Id ampl* is reached. If the *Id ampl* is not reached even with the DCbus/2 and *F start*, the frequency of the measuring signal is gradually decreased by *F dec* down to *F min* again until *Id ampl* is reached. If *Id ampl* is still not reached, the measurement continues with DCbus/2 and *F min*. The tuning process is shown in Figure 11.

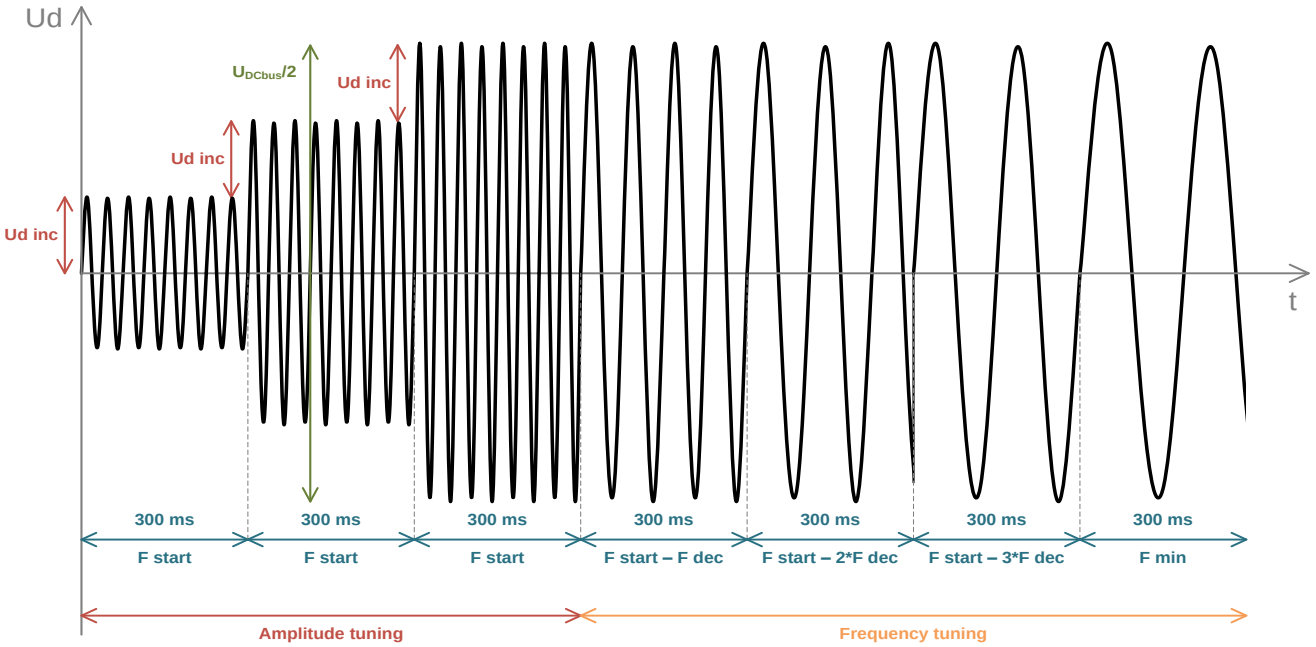

Figure 11. Tuning Ls measuring signal

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

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

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

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

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

5.1.4 BEMF constant measurement

Before the actual BEMF constant (K_e) measurement, the MCAT tool calculates the current controllers and BEMF observer constants from the previously measured R_s, L_d, and L_q. To measure K_e, the motor must spin. I_d is controlled through the "Id meas" and the electrical open-loop position is generated by integrating the required speed, which is derived from N_{nom}. When the motor reaches the required speed, the BEMF voltages obtained by the BEMF observer are filtered and K_e is calculated:

$$K_e = \frac{U_{BEMF}}{\omega_{el}}\ [\Omega] \qquad \textit{Eq. 8}$$

When K_e is being measured, the user must visually check to determine whether the motor is spinning properly. If the motor is not spinning properly, perform the following steps:

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

5.1.5 Number of pole-pair assistant

The number of pole-pairs cannot be measured without a position sensor. However, there is a simple assistant to determine the number of pole-pairs (*pp*). The number of the *pp* assistant performs one electrical revolution, stops for a few seconds, and then repeats it. Because the *pp* value is the ratio between the electrical and mechanical speeds, it can be determined as the number of stops per one mechanical revolution. It is recommended not to count the stops during the first mechanical revolution because the alignment occurs during the first revolution and affects the number of stops. During the *pp* measurement, the current loop is enabled and the I_d current is controlled to $I_{d\ meas}$. The electrical position is generated by integrating the open-loop speed. If the rotor does not move after the start of the number of *pp* assistant, stop the assistant, increase $I_{d\ meas}$, and restart the assistant.

5.1.6 Mechanical parameters measurement

The moment of inertia J and the viscous friction B can be identified using Eq. 12 during the speed acceleration test, with the known generated torque T and the loading torque T_{load}.

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

The ω_m character in the equation is the mechanical speed. The mechanical parameter identification software uses the torque profile, as shown in Figure 12. The loading torque is (for simplicity reasons) said to be 0 during the whole measurement. Only the friction and the motor-generated torque are

considered. During the first phase of measurement, the constant torque T_{meas} is applied and the motor accelerates to 50 % of its nominal speed in time t_1. These integrals are calculated during the period from t_0 (the speed estimation is accurate enough) to t_1:

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

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

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

The final mechanical parameter estimation can be calculated by integrating Eq. 12:

$$\omega_{\mathrm{m}}(t_1) = \frac{1}{J} T_{int} - B\omega_{\mathrm{int}} + \omega_{\mathrm{m}}(t_0) \; [rad/s] \quad \textit{Eq. 12}$$

The moment of inertia is:

$$J = \frac{\tau_{\mathrm{m}} T_{int}}{\tau_{\mathrm{m}}[\omega_{\mathrm{m}}(t_1) - \omega_{\mathrm{m}}(t_0)] + \omega_{\mathrm{int}}} \; [kgm^2] \quad \textit{Eq. 13}$$

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

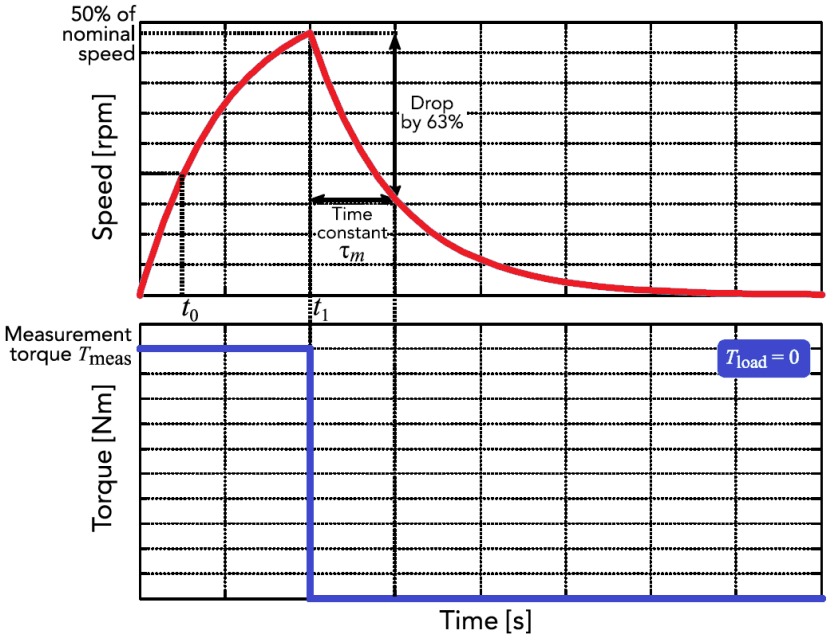

Figure 12. PMSM identification tab

5.1.7 PMSM electrical and mechanical parameters measurement process

The motor identification process can be controlled and set up in the MCAT "Motor Identif" tab, which is shown in Figure 13. To measure your own motor, follow these steps:

- Select your hardware board. Choose between the standard NXP hardware or use your own. If you use your own hardware, specify its scales (“I max” and “U DCB max” in the “Parameters” menu tab).
- If you don’t know the number of motor’s pole-pairs, use the number of pole-pair assistant described in Section 5.1.5, “Number of pole-pair assistant”.
- If you use your own hardware for the first time, perform the power stage characterization described in Section 5.1.1, “Power stage characterization”.
- Enter the motor measurement parameters (depending on the “Basic/Expert” mode) and start the measurement by pressing the “Measure electrical” or “Measure mechanical” buttons. You can observe which parameter is being measured in the “Status” bar.

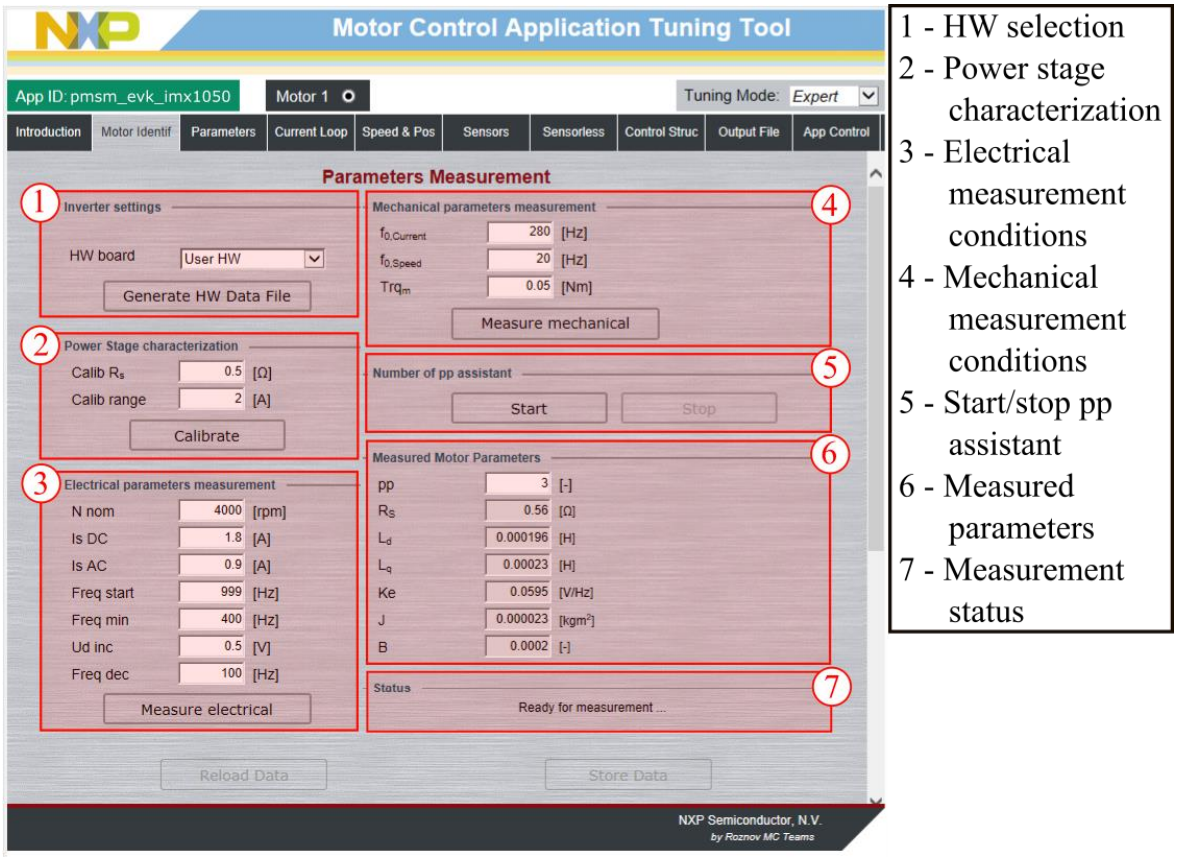

Figure 13. PMSM identification tab

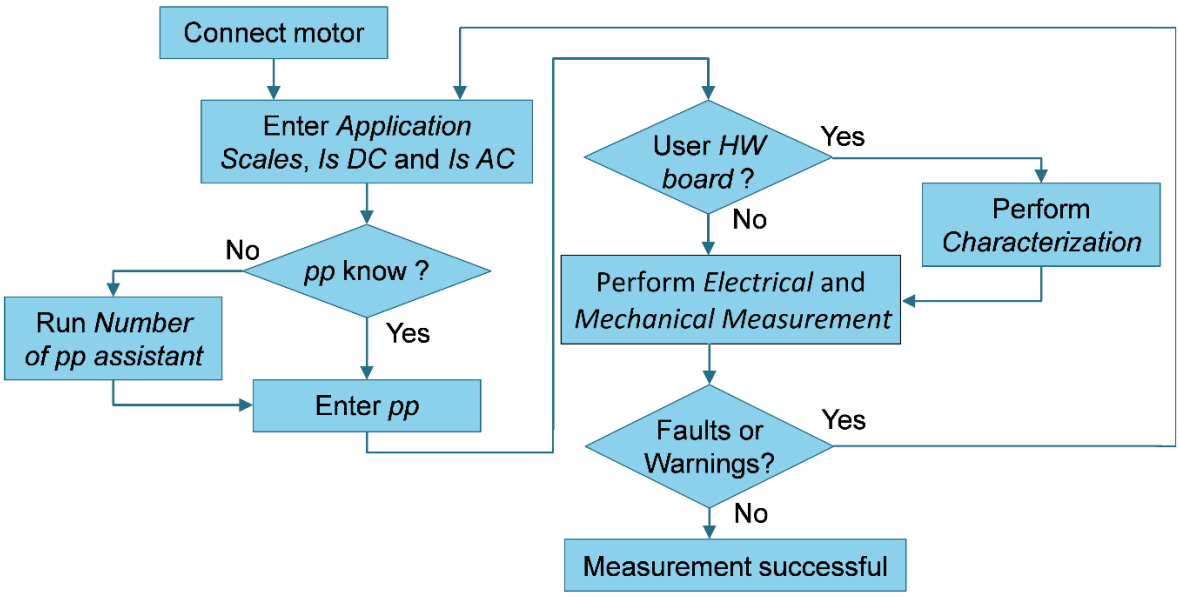

Figure 14. Measurement process diagram

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

Table 6. Measurement faults and warnings

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

5.2 PMSM sensor/sensorless application control and tuning using MCAT

The FreeMASTER (enabled with the MCAT page) can be used to completely control and easily tune the ACIM sensorless FOC application. The MCAT for PMSM submodule tabs (listed in Section 5, "Tuning and controlling the application") are described more closely here.

5.2.1 Application control using MCAT

The application can be controlled through the “Control Struc” tab, which is shown in Figure 15. The state control area on the left side of the screen shows the current application state and enables you to turn the main application switch on or off (turning a running application off disables all PWM outputs). The “Cascade Control Structure” area is placed on the right-hand side of the screen. Here you can choose between the scalar control and FOC control using the appropriate buttons. The selected parts of the FOC cascade structure can be enabled by choosing the “Voltage FOC”, “Current FOC”, “Speed FOC” (sensor/sensorless), or “Position FOC” (sensor). This is useful for application tuning and debugging.

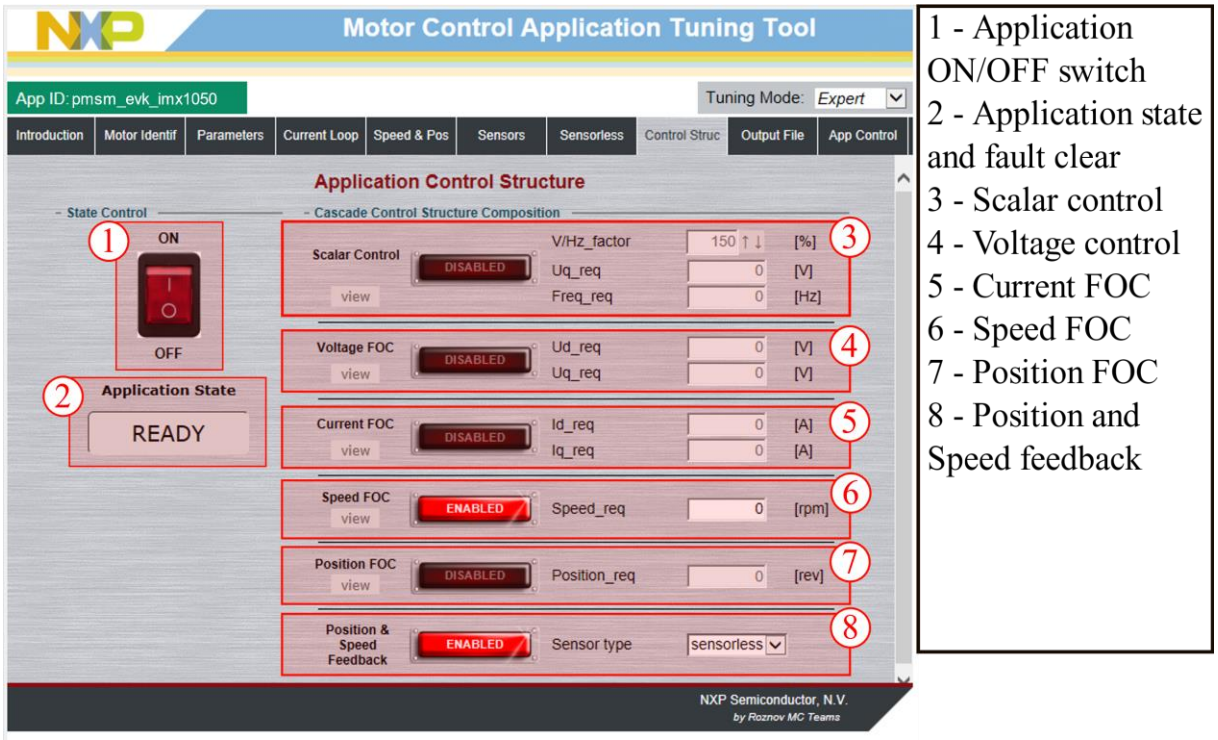

Figure 15. MCAT for PMSM control page

The scalar control diagram is shown in Figure 16. 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 ratio. Hence, the control method is sometimes called Volt per Hertz or V/Hz. Pay attention when entering the required voltage and frequency in the “Expert” tuning mode. The ratio is kept constant in the “Basic” mode and the only required inputs are voltage and frequency. The position estimation BEMF observer and tracking observer algorithms (see *Sensorless PMSM Field-Oriented Control* (document DRM148) for more information) run on the background, even if the estimated position information is not directly used. This is useful for the BEMF observer tuning.

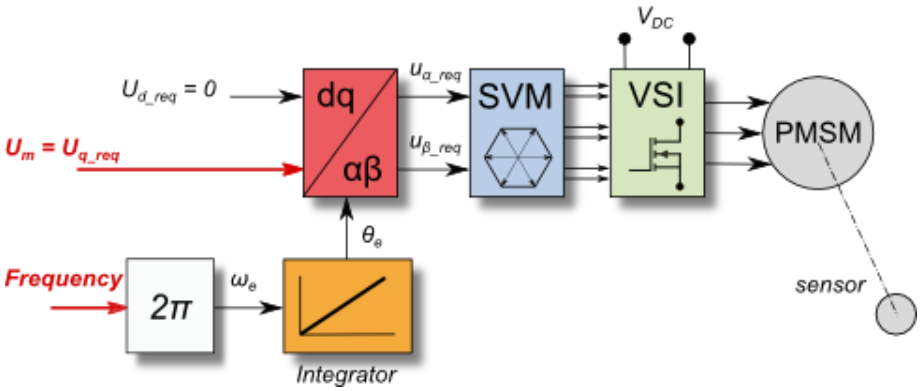

Figure 16. Scalar control modes

The block diagram of the voltage FOC is 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 using the "Ud_req" and "Uq_req" fields. This control method is useful for the BEMF observer functionality check.

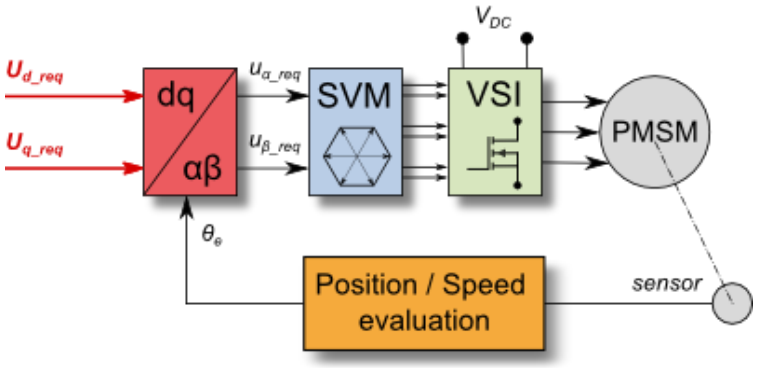

Figure 17. Voltage FOC control mode

The current FOC (or torque) control requires the rotor position feedback and the currents transformed into a d-q reference frame. There are two reference variables ("Id_req" and "Iq_req") available for the motor control, as shown in the block diagram in Figure 18. The *d*-axis current component i_{sd_req} is responsible for the rotor flux control. The *q*-axis current component of the current i_{sq_req} generates a torque and by its application, the motor starts running. By changing the polarity of the current i_{sq_req}, the motor changes the direction of rotation. Supposing that the BEMF observer is tuned correctly, the current PI controllers can be tuned using the current FOC control structure.

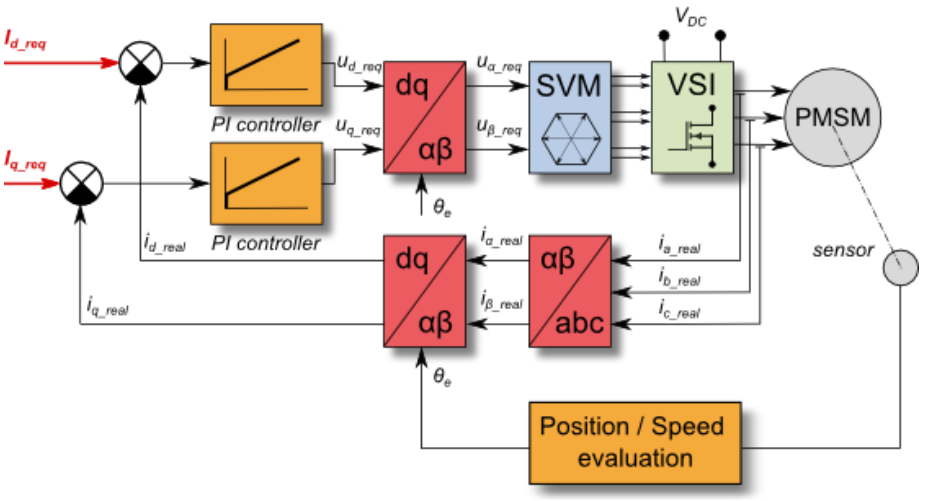

Figure 18. Current (torque) control mode

The speed PMSM sensor/sensorless FOC (its diagram is shown in Figure 19) is activated by enabling the speed FOC control structure. Enter the required speed into the "Speed_req" field. The *d*-axis current reference is held at 0 during the entire FOC operation.

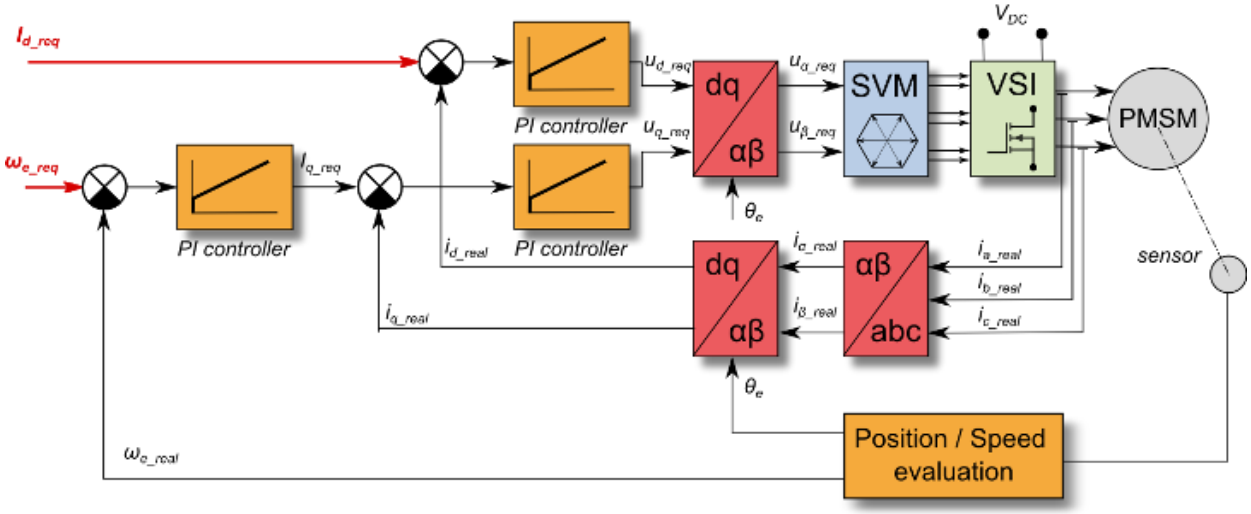

Figure 19. Speed FOC control mode

The position PMSM sensor FOC is shown in Figure 20. The position control using the P controller can be tuned in the "Speed & Pos" 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 needed to set a soft speed ramp (in the "Speed & Pos" menu tab) because the voltage on the DC-bus can rise when braking the quickly spinning shaft. It may cause the overvoltage fault.

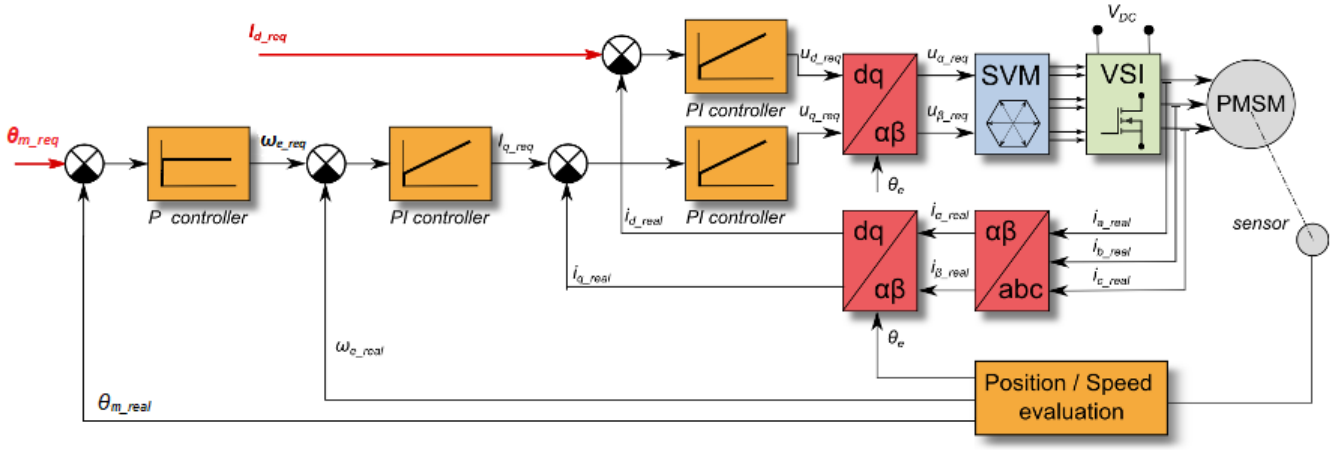

Figure 20. Position FOC control mode

5.2.2 PMSM sensor/sensorless application tuning using MCAT

This section provides a guide on how to run your motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any possible issues during the tuning process. The state diagram shown in Figure 21 depicts a typical PMSM sensor/sensorless control tuning process. Each tuning phase is described in more detail in the following sections.

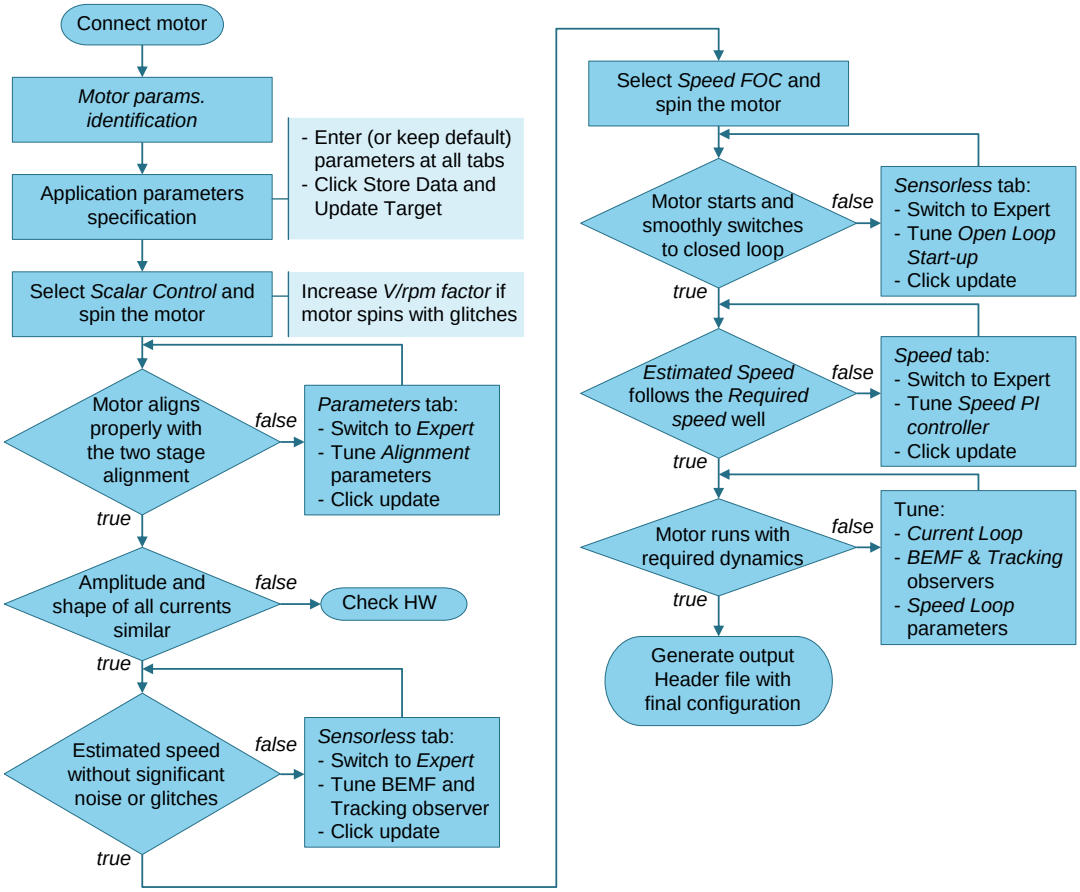

Figure 21. Running a new PMSM

5.2.3 Initial configuration setting and update

1. Open the PMSM control application FreeMASTER project containing the dedicated MCAT plug-in module.
2. Select the “Basic” mode (recommended for users that are not experienced in the motor control theory). The number of required input parameters is reduced.
3. Select the “Parameters” tab.
4. Leave the measured motor parameters or specify the parameters manually. The motor parameters can be obtained from the motor data sheet or using the PMSM parameters measurement procedure described in *PMSM Electrical Parameters Measurement* (document AN4680). All parameters provided in Table 7 are accessible in both the “Basic” and “Expert” modes. The motor inertia J expresses the overall system inertia and can be obtained using a mechanical measurement. The J parameter is used to calculate the speed controller constant. However, the manual controller tuning can also be used to calculate this constant.

Table 7. MCAT motor parameters

Parameter	Units	Description	Typical range
pp	[-]	Motor pole pairs	1 – 10
Rs	[Ω]	1-phase stator resistance	0.3 – 50
Ld	[H]	1-phase direct inductance	0.00001 – 0.1
Lq	[H]	1-phase quadrature inductance	0.00001 – 0.1
Ke	[V.sec/rad]	BEMF constant	0.001 – 1
J	[kg.m2]	System inertia	0.000001 – 1
Iph nom	[A]	Motor nominal phase current	0.5 – 8
Uph nom	[V]	Motor nominal phase voltage	10 – 300
N nom	[rpm]	Motor nominal speed	1000 – 2000

5. 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.
6. Check the fault limits—these fields are not accessible in the “Basic” mode and are calculated using the motor parameters and hardware scales (see Table 8).

Table 8. Fault limits

Parameter	Units	Description	Typical range
U DCB trip	[V]	Voltage value at which the external braking resistor switch turns on	U DCB Over ~ U DCB max
U DCB under	[V]	Trigger value at which the undervoltage fault is detected	0 ~ U DCB Over
U DCB over	[V]	Trigger value at which the overvoltage fault is detected	U DCB Under ~ U max
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

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

Table 9. Application scales

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

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

calculated using the motor parameters and hardware scales. The parameters express the required voltage value applied to the motor during the rotor alignment and its duration.

9. Click the “Store Data” button to save the modified parameters into the inner file.

5.2.4 Control structure modes

1. Select the scalar control by clicking the “DISABLED” button in the “Scalar Control” section. The button color changes to red and the text changes to “ENABLED”.
2. Turn the application switch on. The application state changes to “RUN”.
3. Set the required frequency value in the “Freq_req” field; for example, 15 Hz in the “Scalar Control” section. The motor starts running (see Figure 22).

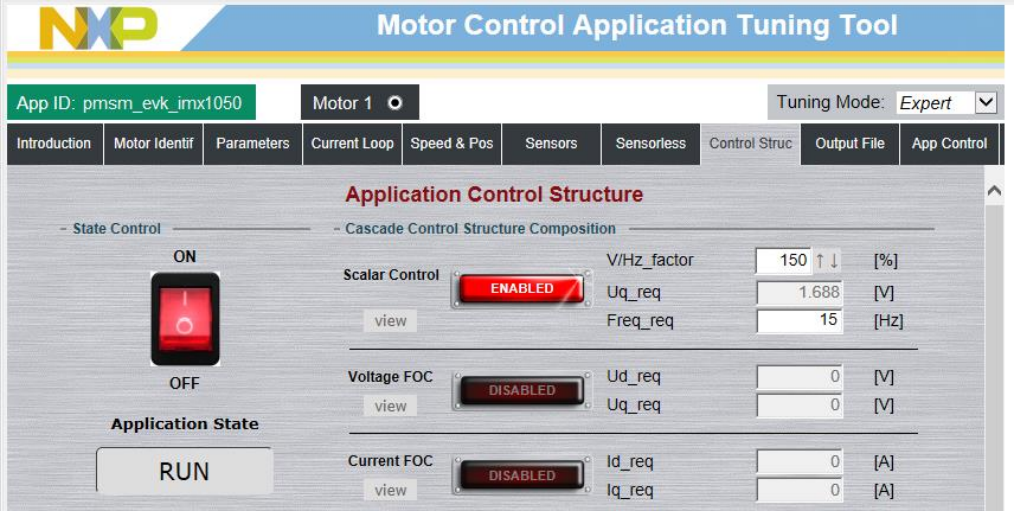

Figure 22. MCAT scalar control

4. Select the “Phase Currents” recorder from the “Scalar & Voltage Control” FreeMASTER project tree.
5. The optimal ratio for the V/Hz profile can be found by changing the V/Hz factor directly or using the “UP/DOWN” buttons. The shape of the motor currents should be close to a sinusoidal shape (see Figure 23).

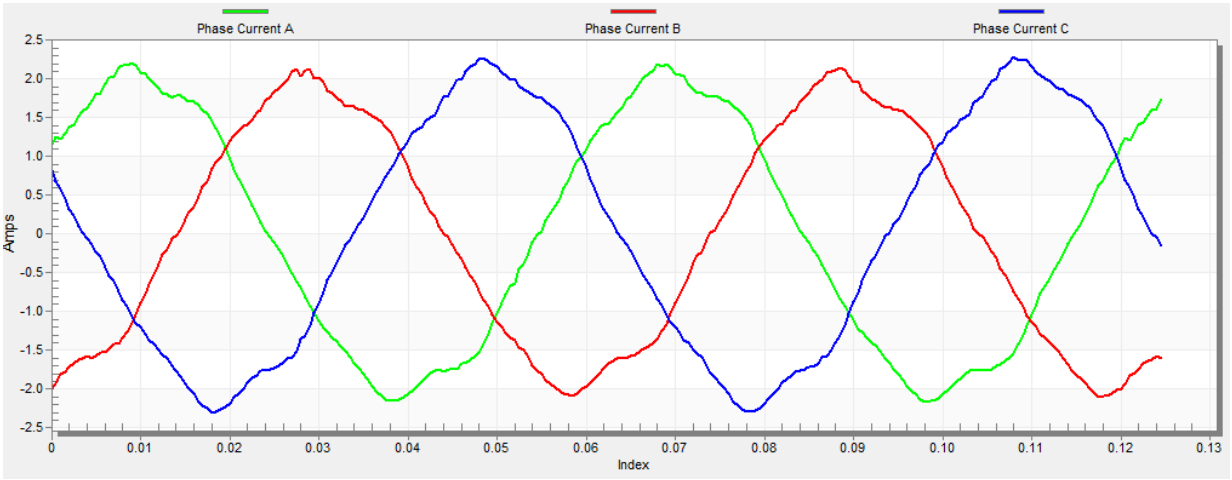

Figure 23. Phase currents

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 24) 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.

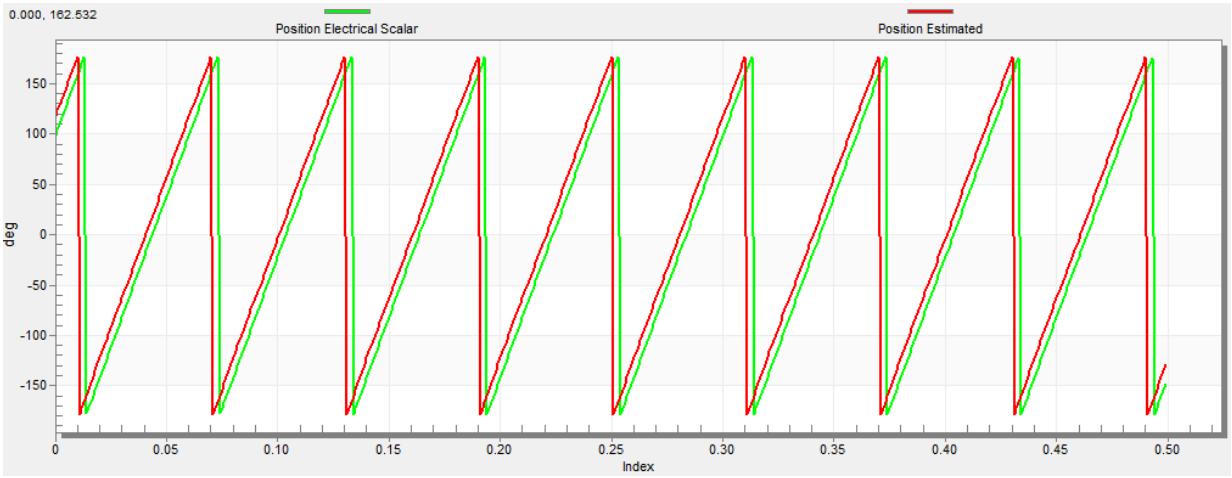

Figure 24. Generated and estimated positions

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

5.2.5 Encoder sensor setting

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

1. Select scalar control by clicking the DISABLED button in the “Scalar Control” section of the “Control Struct” tab. The button color changes to red, and the text to “ENABLED”.
2. Turn the application switch on. The application state changes to “RUN”.
3. Set the required frequency value in the “Freq_req” field; for example, 10 Hz in the “Scalar control” section. The motor starts running.
4. Check the encoder direction. Select the “Encoder Direction Scope” from the “Scalar & Voltage Control” project tree. If the encoder direction is right, the estimated speed is equal to the measured mechanical speed. If the measured mechanical speed is opposite to the estimated speed, the direction in the “Sensor” tab must be changed from 0 to 1 (or vice versa).

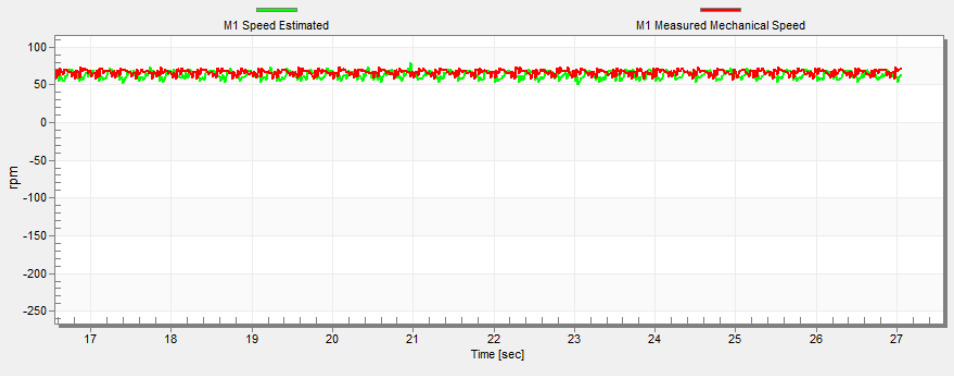

Figure 25. Encoder direction – right direction

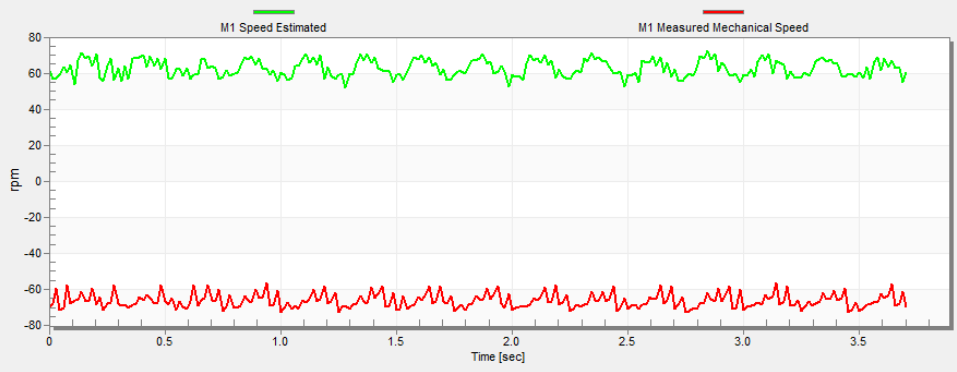

Figure 26. Encoder direction – wrong direction

5.2.6 Alignment tuning

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

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

5.2.7 Current loop tuning

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

1. Switch the tuning mode to “Expert”.
2. Lock the motor shaft.
3. Set the required loop bandwidth and attenuation and click the “Update Target” button in the “Current Loop” tab. The tuning loop bandwidth parameter defines how fast the loop response is whilst the tuning loop attenuation parameter defines the actual quantity overshoot magnitude.

4. Select the “Current Controller Id” recorder.
5. Select the “Control Structure” tab, switch to “Current FOC”, set the “Iq_req” field to a very low value (for example 0.01), and set the required step in “Id_req”. The control loop response is shown in the recorder (see Figure 10).
6. Tune the loop bandwidth and attenuation until you achieve the required response. The example waveforms show the correct and incorrect settings of the current loop parameters:
   - The loop bandwidth is low (110 Hz) and the settling time of the I_d current is long (see Figure 27).

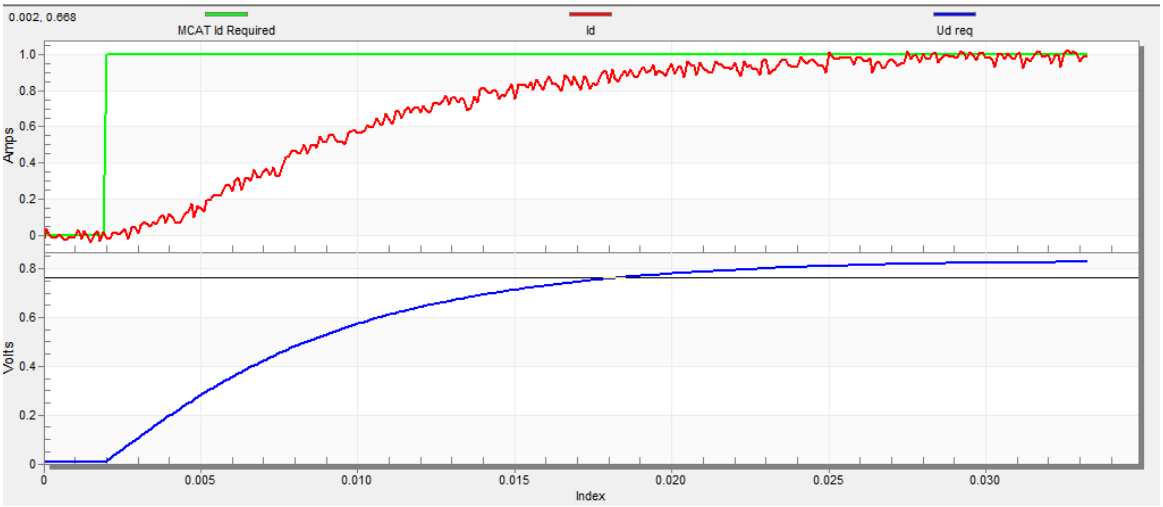

Figure 27. Slow step response of the Id current controller

   - The loop bandwidth (400 Hz) is optimal and the response time of the I_d current is sufficient (see Figure 28).

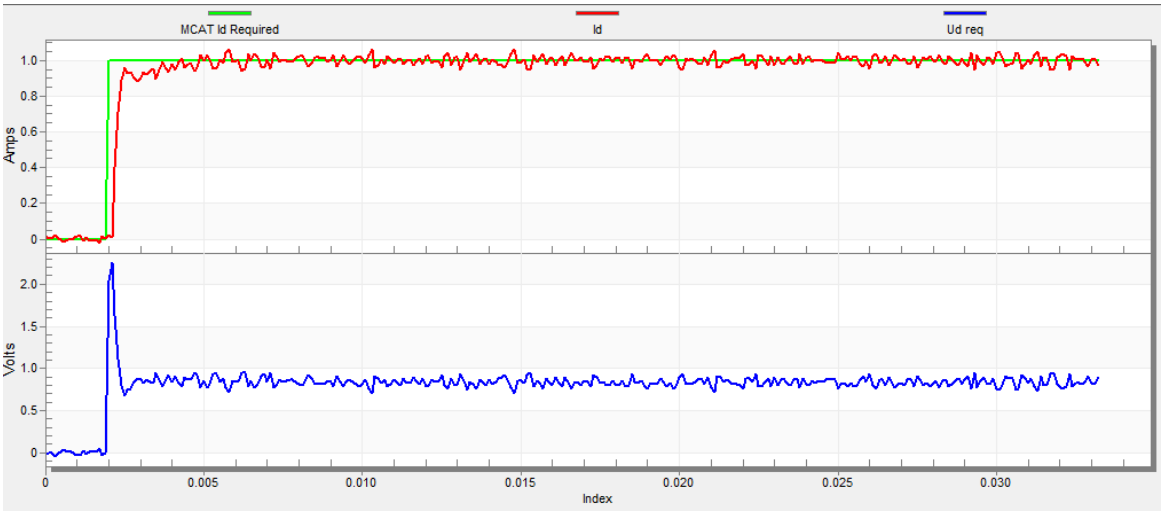

Figure 28. Optimal step response of the Id current controller

   - The loop bandwidth is high (700 Hz) and the response time of the I_d current is very fast, but with oscillation and overshoot (see Figure 29).

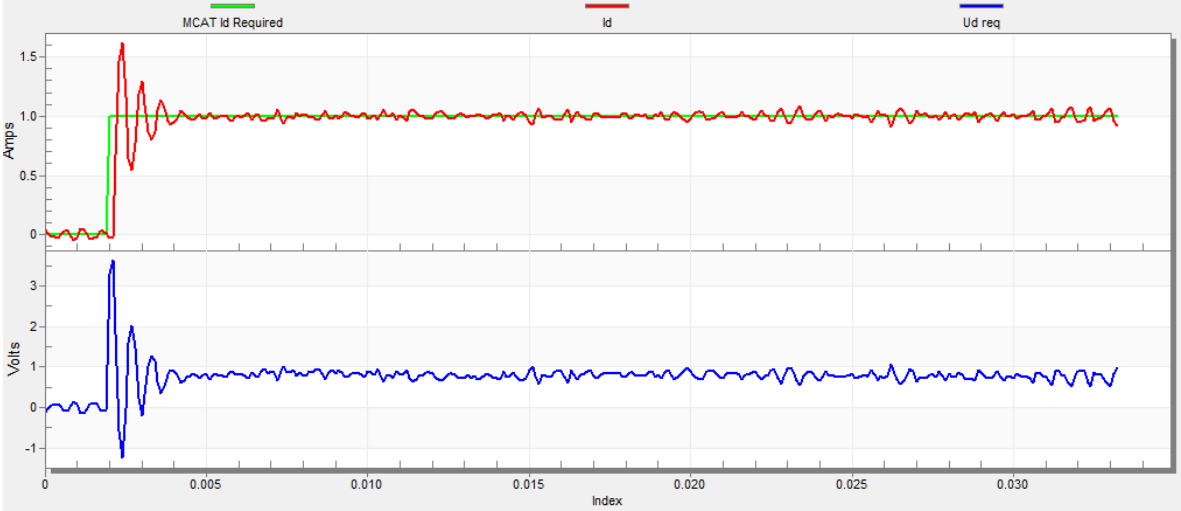

Figure 29. Fast step response of the Id current controller

5.2.8 Speed ramp tuning

The speed command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) 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 the overvoltage fault during deceleration. In the “Speed” scope, you can see whether the “Speed Actual Filtered” waveform shape equals the “Speed Ramp” profile.

The increments are common for the scalar and speed control. The increment fields are in the “Speed & Pos” tab and accessible in both tuning modes. Clicking the “Update Target” button applies the changes to the MCU. An example speed profile is shown in Figure 30. The ramp increment down is set to 500 rpm/sec, while the increment up is set to 3,000 rpm/sec.

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

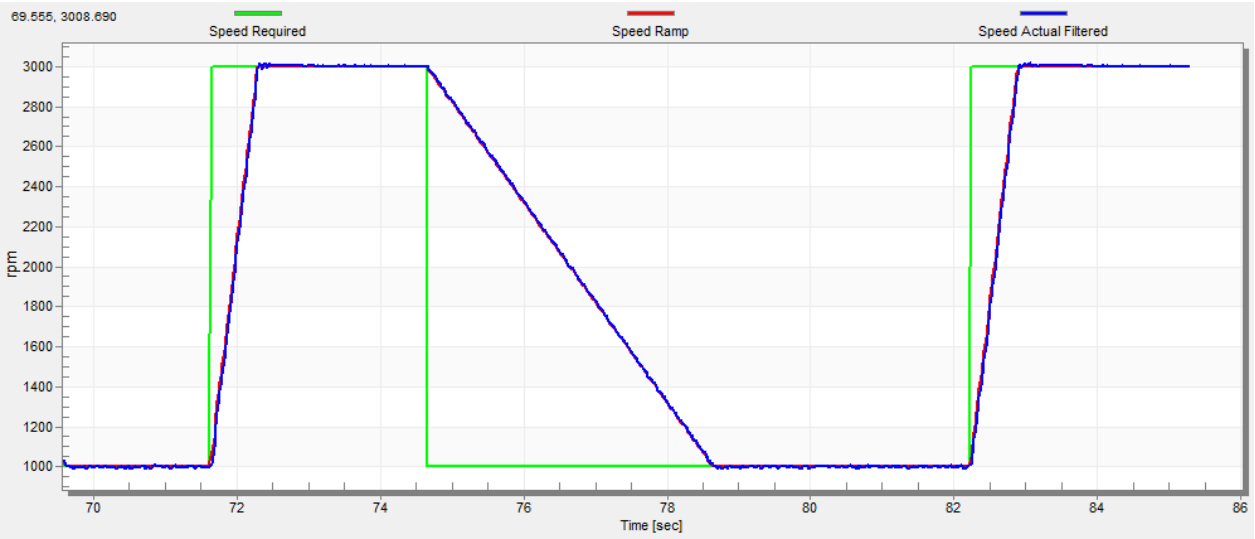

Figure 30. Speed profile

5.2.9 Open loop startup

The start-up process can be tuned by a set of parameters located in the “Sensorless” tab. Two of them (ramp increment and current) are accessible in both tuning modes. The start-up tuning can be processed in all control modes besides the scalar control. Setting the optimal values results in a proper motor startup. An example start-up state of low dynamic drives (fans, pumps) is shown in Figure 31.

1. Select the “Startup” recorder from the FreeMASTER project tree.
2. Set the start-up ramp increment typically to a higher value than the speed loop ramp increment.
3. Set the start-up current according to the required start-up torque. For drives such as fans or pumps, the start-up torque is not very high and can be set to 15 % of the nominal current.
4. 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.
5. Set the merging coefficient—in the position merging process duration, 100 % corresponds to a half of an electrical revolution. The higher the value, the faster the merge is done. The values close to 1 % are set for the drives where a high start-up torque and a smooth transition between the open loop and the closed loop are required.
6. Click the “Update Target” button to apply the changes to the MCU.
7. Switch to the “Control Structure” tab, and enable the “Speed FOC”.
8. Set the required speed higher than the merging speed.
9. Check the start-up response in the recorder.
10. Tune the start-up parameters until you achieve an optimal response.
11. If the rotor does not start running, increase the start-up current.
12. If the merging process fails (the rotor is stuck or stopped), decrease the start-up ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

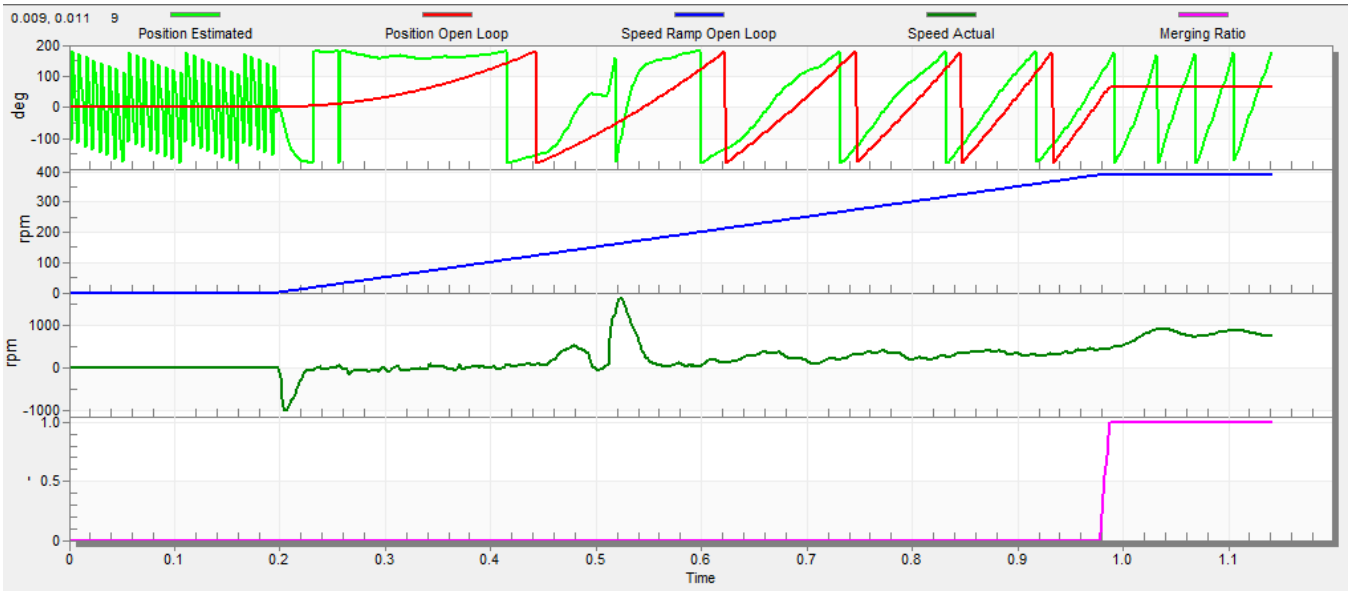

Figure 31. Motor startup

5.2.10 BEMF observer tuning

In the “Basic” mode, the parameters of the BEMF observer and the tracking observer are fully calculated using the motor parameters and no action is required in this mode. If the calculated loop parameters do not correspond to the optimal response, the bandwidth and attenuation parameters can be tuned.

1. Switch the tuning mode to “Expert”.
2. Select the “Observer” recorder from the FreeMASTER project tree.
3. Set the required bandwidth and attenuation of the BEMF observer—the bandwidth is typically set to a value close to the current loop bandwidth.
4. Set the required bandwidth and attenuation of the tracking observer—the bandwidth is typically set in the range of 10 – 20 Hz for most low-dynamic drives (fans, pumps).
5. Click the “Update Target” button to apply the changes to the MCU.
6. Check the observer response in the recorder.

5.2.11 Speed PI controller tuning

The motor speed control loop is a first-order function with a mechanical time constant that depends on the motor inertia and friction. Obtaining these mechanical constants using the PMSM electrical and mechanical parameter measurement process is described in Section 5.1.6, “Mechanical parameters measurement” and Section 5.1.7, “PMSM electrical and mechanical parameters measurement process”. If these mechanical constants are available, the PI controller constants can be tuned using the loop bandwidth and attenuation. Otherwise, the manual tuning of the P and the I portions of the speed controllers is available to obtain the required speed response (see the example response in Figure 32). There are dozens of approaches available to tune the PI controller constants. The following steps provide one of these examples to set and tune the speed PI controller for a PM synchronous motor:

1. Select the “Speed Controller” option from the FreeMASTER project tree.
2. Select the “Speed & Pos” tab.
3. Check the “Manual Constant Tuning” option—that is, the “Bandwidth” and “Attenuation” fields are disabled and the “SL_Kp” and “SL_Ki” fields are enabled.
4. Tune the proportional gain:
    a) Set the “SL_Ki” integral gain to zero.
    b) Set the speed ramp to 1000 rpm/sec (or higher).
    c) Switch to the “Control Structure” tab and run the motor at a convenient speed (about 30 % of the nominal speed).
    d) Set a step in the required speed to 40 % of N_{nom}.
    e) Switch back to the “Speed loop” tab.
    f) Adjust the proportional gain “SL_Kp” until the system responds to the required value properly and without any oscillations or excessive overshoot:
        - If the “SL_Kp” field is set low, then the system response is slow.
        - If the “SL_Kp” field is set high, then the system response is tighter.
        - When the “SL_Ki” field equals 0, then the system most probably does not achieve the required speed.

g) Click the “Update Target” button to apply the changes to the MCU.

5. Tune the integral gain:
   a) Increase the “SL_Ki” field slowly to minimize the difference between the required and actual speeds to 0.
   b) 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.
   c) Click the “Update Target” button to apply the changes to the MCU.
6. Tune the loop bandwidth and attenuation until the required response is received. The example waveforms with the correct and incorrect settings of the speed loop parameters are shown in the following figures:
   – The “SL_Ki” value is low, and the “Speed Actual Filtered” does not achieve the “Speed Ramp” (see Figure 32).

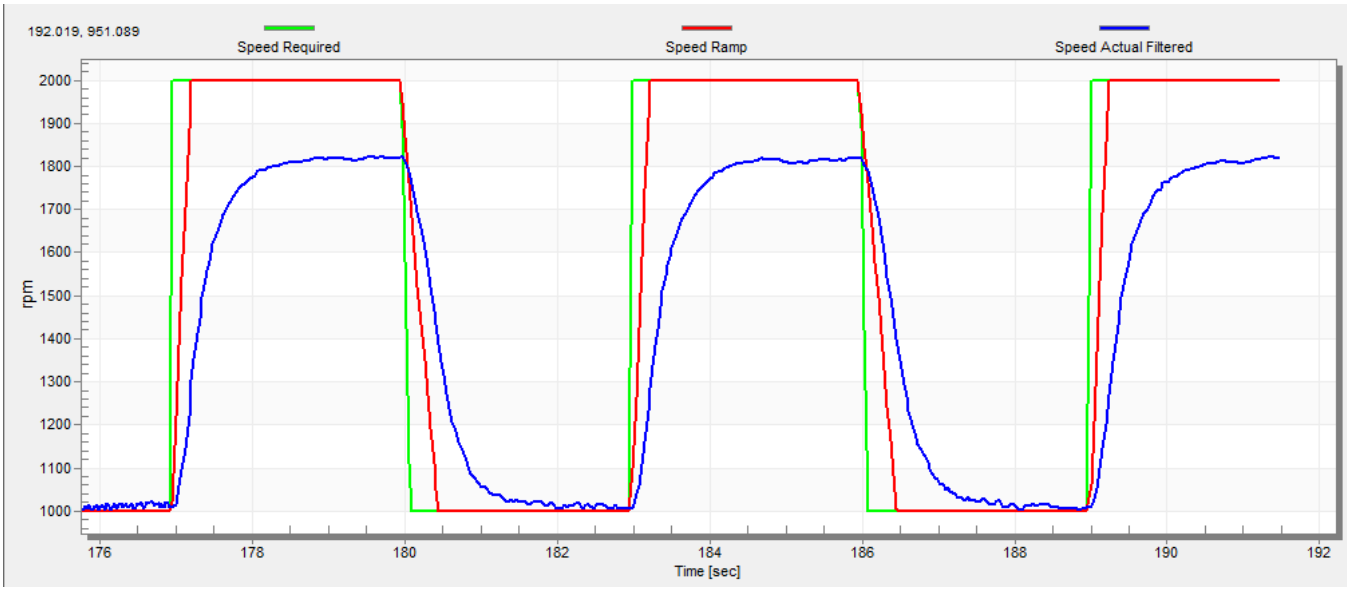

Figure 32. Speed Controller Response—SL_Ki value is low, Speed Ramp is not achieved

   – The “SL_Kp” value is low, the “Speed Actual Filtered” greatly overshoots, and the long settling time is unwanted (see Figure 33).

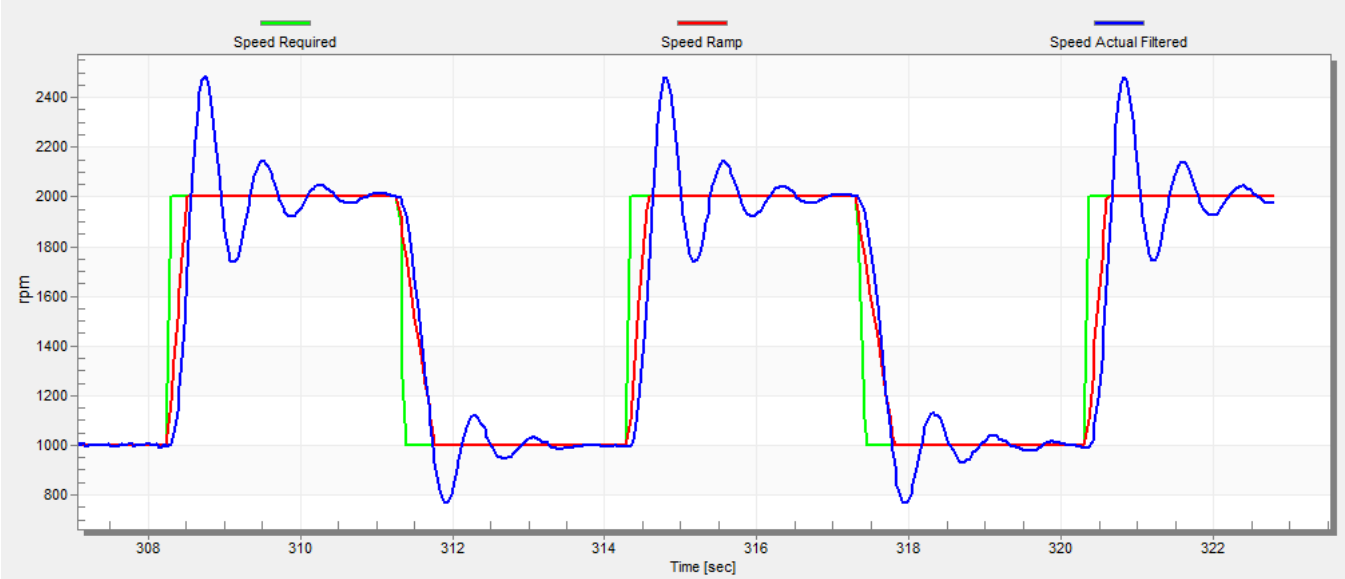

Figure 33. Speed Controller Response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

– The speed loop response has a small overshoot, and the “Speed Actual Filtered” settling time is sufficient. Such response can be considered optimal (see Figure 34).

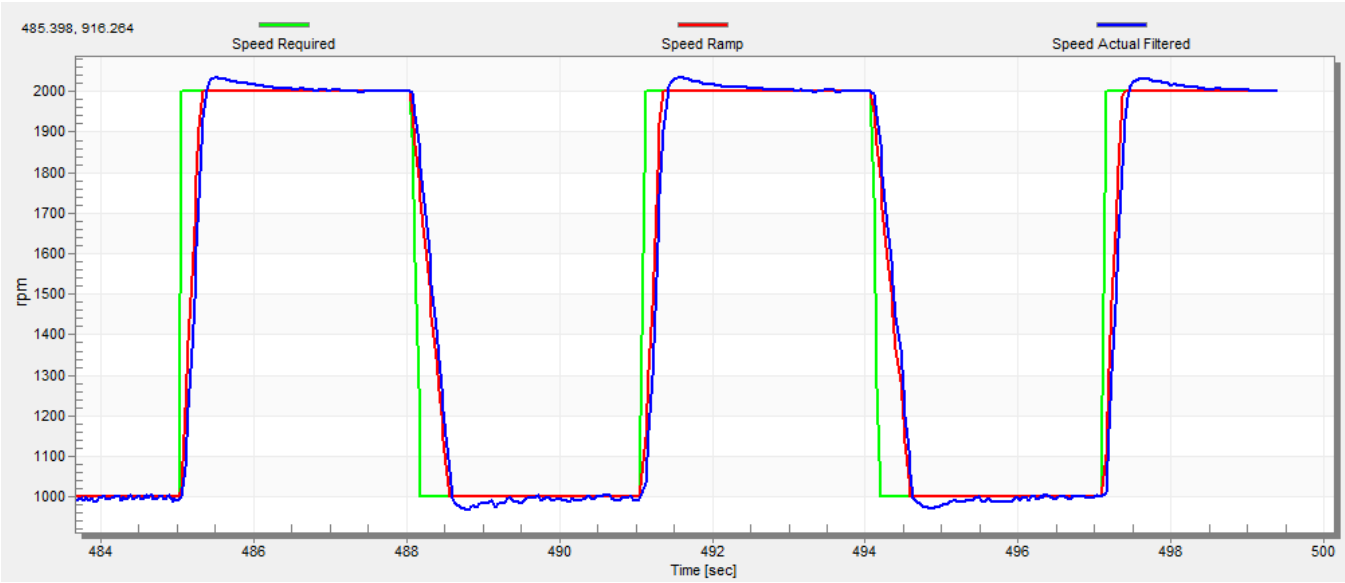

Figure 34. Speed Controller Response—speed loop response with a small overshoot

5.2.12 Position P controller tuning

The position control loop can be tuned using the proportional gain “PL_Kp” in the “Speed & Pos” menu tab. It is a proportional controller, which can be used for unpretending position control systems. The key for the the optimal position response is a proper value of the controller which simply multiplies the error by the proportional gain (Kp) to get the controller output. The predefined base value for a TG drive motor is described in the user’s guide, but, for diffrent motors, it can be manually changed. 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” from the FreeMASTER project tree.
2. Select the “Speed & Pos” tab.
3. Tune the proportional gain in the position P controller constant:
    a) Set a small value of “PL_Kp” (M1 Position Loop Kp Gain) and click the “Update Target” button.
    b) Switch to “Control structure”, select the position control and set the required position (for example) to 10 revolutions.
    c) Select the “Position Controller” and watch the actual position response.
4. 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.

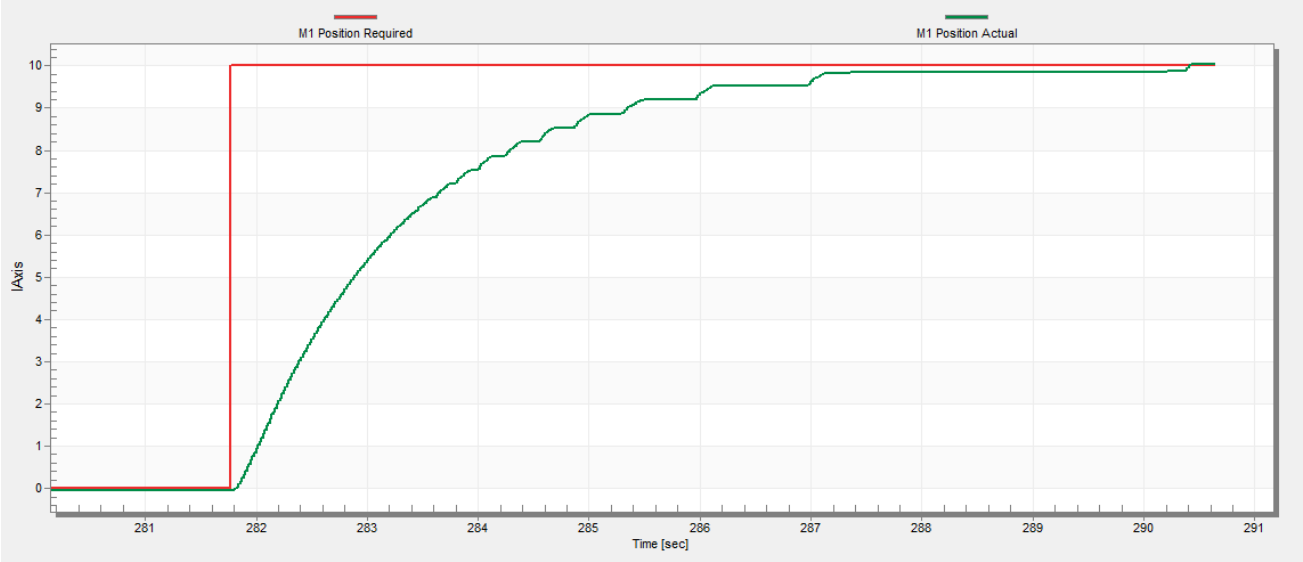

Figure 35. Position Controller Response—PL_Kp value is low, the actual position response is very slow

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

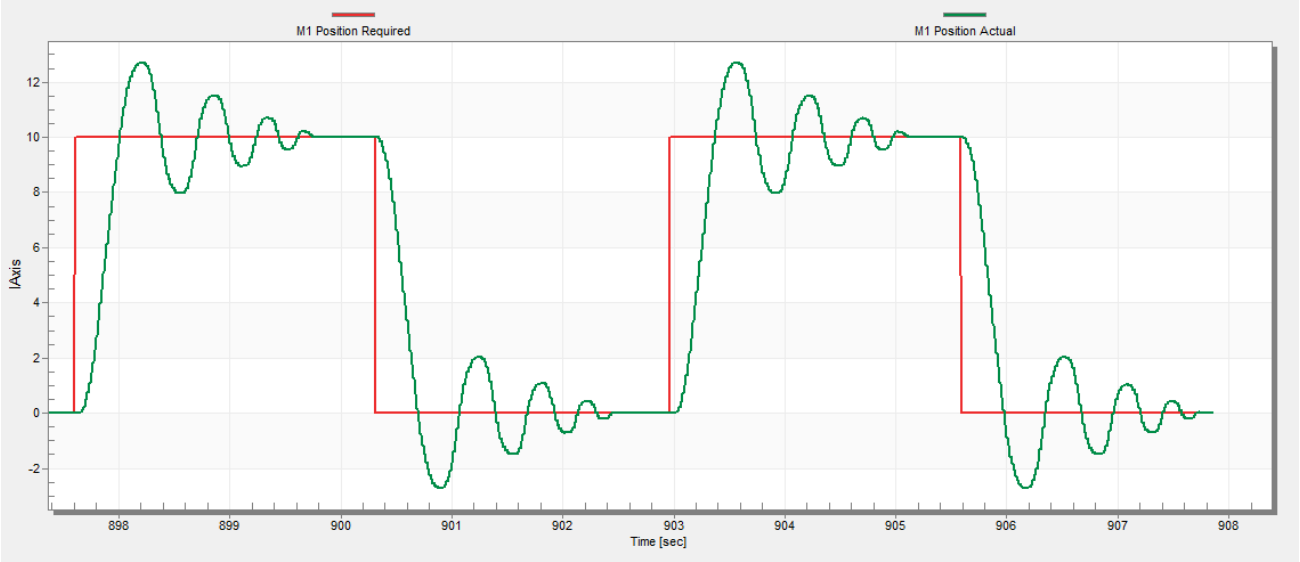

Figure 36. Position Controller Response—PL_Kp value is too high the actual position overshoots

The “PL_Kp” value is optimal and the actual position response is optimal.

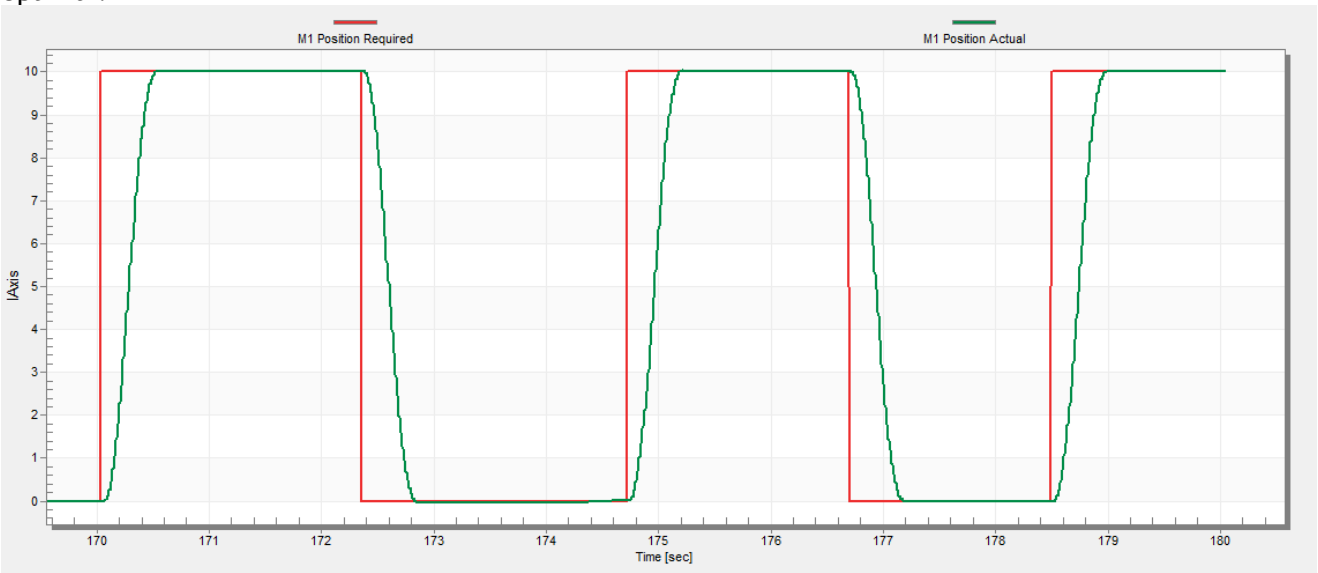

Figure 37. Position Controller Response—the actual position response is good

5.2.13 MCAT output file generation

When you successfully finish tuning the application and want to store all calculated parameters to an embedded application, navigate to the “Output File” tab. The list of all definitions generated by the MCAT can be viewed there. Clicking the “Generate Configuration File” button overwrites the old version of the *m1_pmsm_appconfig.h* file, which contains these definitions. Note that for the right function of the motor parameter file generation, a correct path to the file must be provided. To change the path, navigate the mouse cursor to the right-hand corner of the MCAT screen, until the symbol with

a screw driver and a wrench appears. When you click this symbol, the “Application Settings Page” appears. In the “Project Path Selection” area, you can modify the path to *m1_pmsm_appconfig.h*.

6 Conclusion

This application note describes the implementation of the sensor and sensorless Field-Oriented Control of a 3-phase PMSM on the NXP MIMXRT10xx EVK with the FRDM-MC-LVPMSM NXP Freedom Development Platform. The hardware-dependent part of the control software (including the detailed peripheral setup, motor control peripheral initialization, and application timing) is described in Section 3, “MCU features and peripheral settings”. The motor parameters identification theory and the identification algorithms themselves are described in Section 5.1, “PMSM parameters identification”. The last part of the document introduces and explains the user interface represented by the Motor Control Application Tuning (MCAT) tool based on the FreeMASTER communication interface.

7 Acronyms and abbreviations

Table 10. Acronyms and abbreviations

Acronym	Meaning
ADC	Analog-to-Digital Converter
ACIM	Asynchronous Induction Motor
ADC_ETC	ADC External Trigger Control
AN	Application Note
CCM	Clock Controller Module
CPU	Central Processing Unit
DC	Direct Current
DRM	Design Reference Manual
ENC	Encoder
FOC	Field-Oriented Control
GPIO	General-Purpose Input/Output
LPUART	Universal Asynchronous Receiver/Transmitter
MCAT	Motor Control Application Tuning tool
MCDRV	Motor Control Peripheral Drivers
MCU	Microcontroller
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

8 References

These references are available on www.nxp.com:

1. *Sensorless PMSM Field-Oriented Control* (document DRM148).
2. *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642).
3. *Sensorless PMSM Field-Oriented Control on Kinetis KV* (document AN5237).
4. *PMSM Field-Oriented Control on MIMXRT1050 EVK* (document AN12169).
5. *i.MX RT1050 Processor Reference Manual* (document IMXRT1050RM).
6. *i.MX RT1020 Processor Reference Manual* (document IMXRT1020RM).
7. *i.MX RT1060 Processor Reference Manual* (document IMXRT1060RM).

How to Reach Us:

Home Page:
www.nxp.com

Web Support:
www.nxp.com/support

Document Number: AN12214
Rev. 0
12/2018