NXP Semiconductors
Application Note

Document Number: AN5237
Rev. 2, 09/2016

Sensorless PMSM Field-Oriented Control on Kinetis KV and KE

By: Josef Tkadlec

1. Introduction

This application note describes the implementation of the sensorless Motor Control Reference Solution Package (MCRSP) software for a 3-phase Permanent Magnet Synchronous Motor (PMSM), including a motor parameters identification algorithm, running on 32-bit Kinetis V and E series MCUs. The sensorless control software itself and the PMSM control theory in general is described in *Sensorless PMSM Field-Oriented Control* (document DRM148). The NXP Freedom board (FRDM-MC-LVPMSM), Tower System modular development platform module (TWR-MC-LV3PH), and High-Voltage Platform power stages (HVP-MC3PH) are used as hardware platforms for the PMSM control reference solution. The hardware-dependent part of the sensorless control software is addressed as well, including detailed peripheral setup and the Motor Control Peripheral Drivers (MCDRV). 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 FreeMASTER run-time debugging tool. These tools represent a simple and user-friendly way of motor parameter identification, algorithm tuning, software control, debugging, and diagnostics.

Contents

2. Development Platforms

There are these three standard NXP power stages:

- FRDM-MC-LVPMSM
- TWR-MC-LV3PH
- HVP-MC3PH

2.1. FRDM-MC-LVPMSM

This evaluation board (in a shield form factor) effectively turns a Freedom development board into a complete motor-control reference design, compatible with the existing Freedom development boards—FRDM-KV31F, FRDM-KV10Z, and FRDM-KE15Z. The Freedom motor-control headers are compatible with Arduino™ R3 pin layout.

The FRDM-MC-LVPMSM board has a power-supply input voltage of 24–48 V DC with a reverse polarity protection circuitry. An auxiliary power supply of 5.5 V DC is available to provide power to the FRDM MCU boards. The output current reaches up to 5 A RMS. The inverter is realized by the 3-phase bridge inverter (six MOSFETs) and the 3-phase MOSFET gate driver. Analog quantities (such as 3-phase motor currents, DC-Bus voltage, and DC-Bus current) are measured on this board. There is also an interface for speed/position sensors (Encoder Hall). The block diagram of a complete Freedom motor-control development kit is shown in this figure:

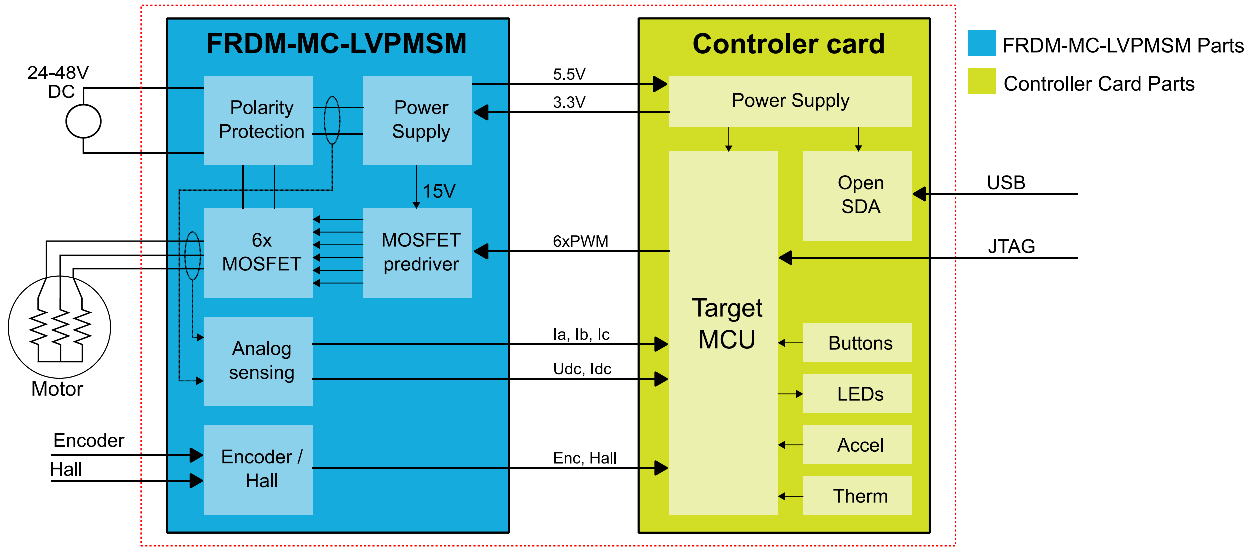


Figure 1. Freedom motor-control development platform block diagram

The FRDM-MC-LVPMSM board does not require a complicated setup and there is only one way to connect this shield board to the Freedom MCU board. See the user's guide for your version of MCRSP (document PMSMCRSPUG). For more information about the Freedom development platform, visit www.nxp.com/freedom.

2.2. TWR-MC-LV3PH

This module effectively turns a Tower System development module into a complete motor-control reference design, compatible with the existing Tower System Kinetis V and E modules. This module provides all necessary feedback signals to drive PMSM and BLDC motors. The TWR-MC-LV3PH module has the power-supply input voltage of 12–24 V DC, extendable up to 50 V DC, with reverse polarity protection circuitry. An auxiliary power supply of 5 V DC and 3.3 V DC provides power supply for the Tower System MCU modules. The output current reaches up to 5 A RMS. The inverter is realized by the 3-phase bridge inverter (six MOSFETs) and the 3-phase MOSFET gate driver. Analog quantities (such as 3-phase motor currents, 3-phase motor back-EMF voltage, DC-bus voltage, and DC-bus current) are measured on this board. There is an interface for speed/position sensors (Encoder Hall) and a connector for a braking resistor. There is also a user LED, a power-on LED, and six PWM LED diodes for diagnostics. The block diagram of a complete Tower System motor-control development kit is shown in this figure:

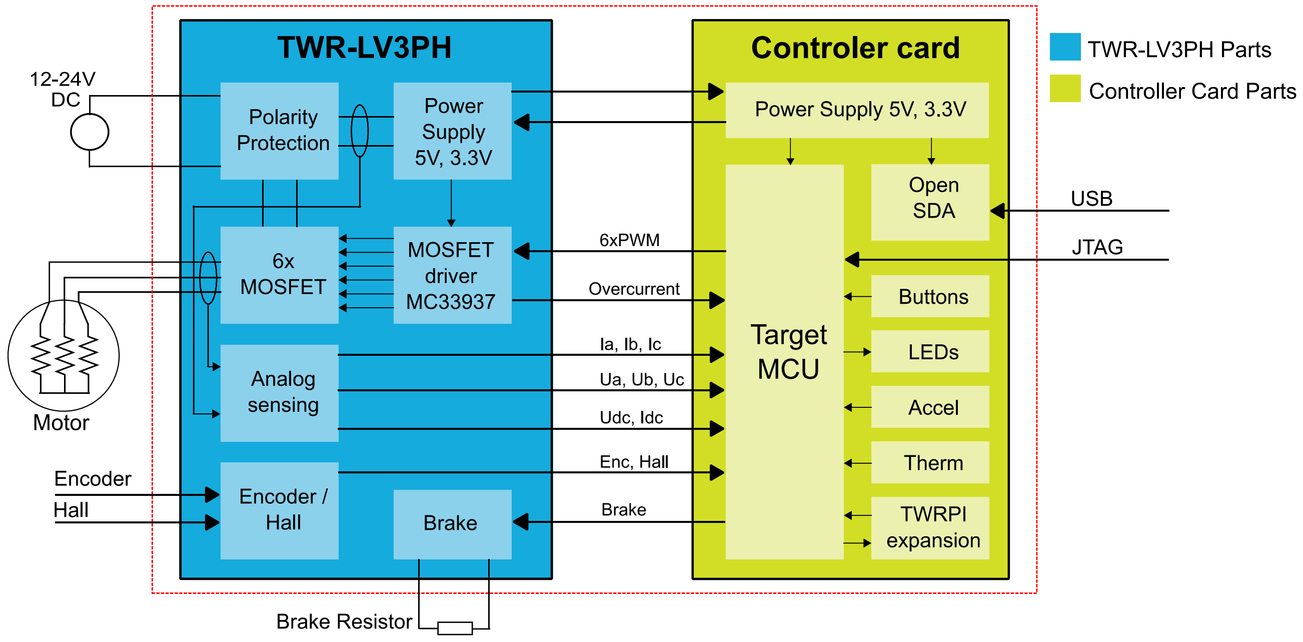


Figure 2. Tower System motor-control development platform block diagram

The TWR-MC-LV3PH module does not require a complicated setup. Keep in mind that the side with the white stripe must be connected to the primary (white) elevator. See the user's guides for the TWR-MC-LV3PH (document TWRMCLV3PHUG) and for your version of MCRSP (document PMSMCRSPUG). For more information about the Tower System visit www.nxp.com/tower.

2.3. HVP-MC3PH

The 3-phase High-Voltage Development Platform (HVP) is a 115/230 V, 1 kW power stage that is an integral part of the embedded motion-control series of development tools. It is supplied in the HVP-MC3PH kit. Combined with the HVP daughter board, it provides a ready-made software development platform for more than one-horsepower motors. The block diagram of a complete high-voltage motor-control development kit is shown in the following figure.

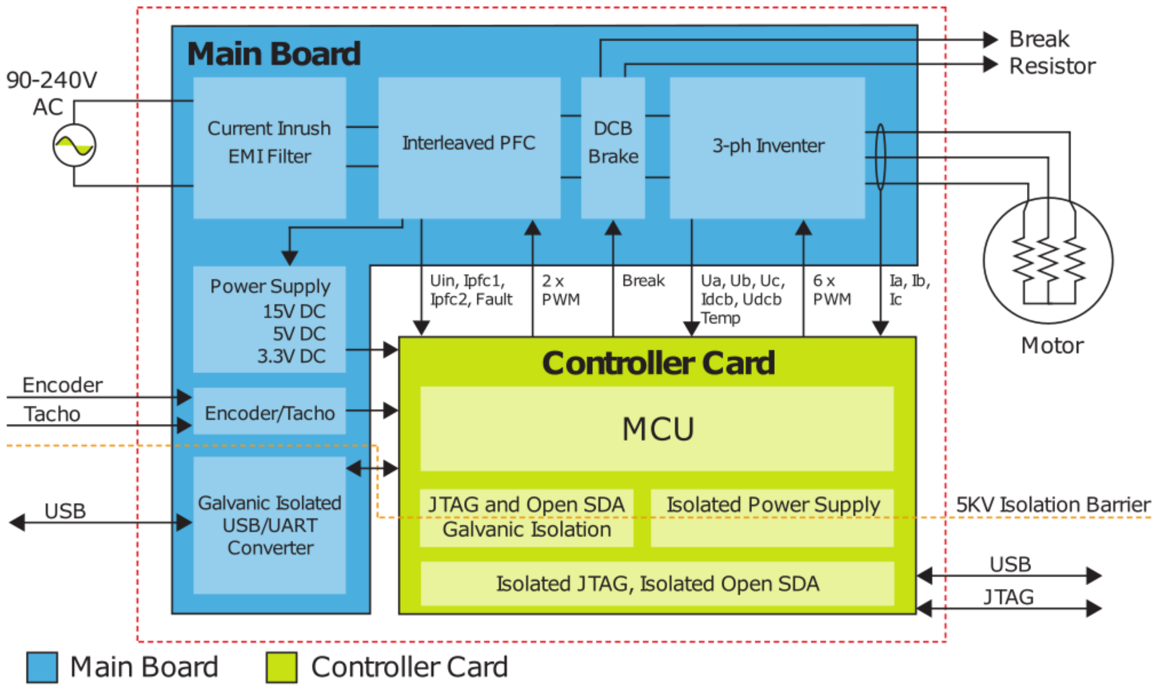


Figure 3. High-Voltage Development Platform block diagram

The HVP-MC3PH power stage does not require any complicated setup and there is only one way to connect a daughter board to the HVP. See the user’s guides for the HVP power stage (document HVPMC3PHUG) and for your version of MCRSP (document PMSMCRSPUG).

NOTE

Due to high voltage, the HVP platform can represent a safety risk when not handled properly. For more information about the High-Voltage Development Platform, visit www.nxp.com/hvp.

3. MCU Features and Peripheral Settings

The peripherals used for motor control vary among different Kinetis V MCUs. The following sections describe the peripheral settings and application timing for each MCU. There are also differences among the MC platforms for different MCUs. These differences are summarized in tables.

3.1. KV1x family

The KV10Z and KV11Z MCU families are highly scalable members of the Kinetis V series and provide a cost-competitive motor-control solution. Built upon the ARM® Cortex®-M0 core running at 75 MHz with up to 128 KB of flash and up to 16 KB of RAM, the MCUs deliver a platform that enables the customers to build a scalable solution portfolio. The additional features include dual 16-bit ADCs sampling at up to 1.2 MS/s in 12-bit mode and 20 channels of flexible motor-control timers (PWMs) across six independent time bases. For more information, see *KV11F Sub-Family Reference Manual* (document KV11P64M75RM).

3.1.1. Hardware timing and synchronization

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

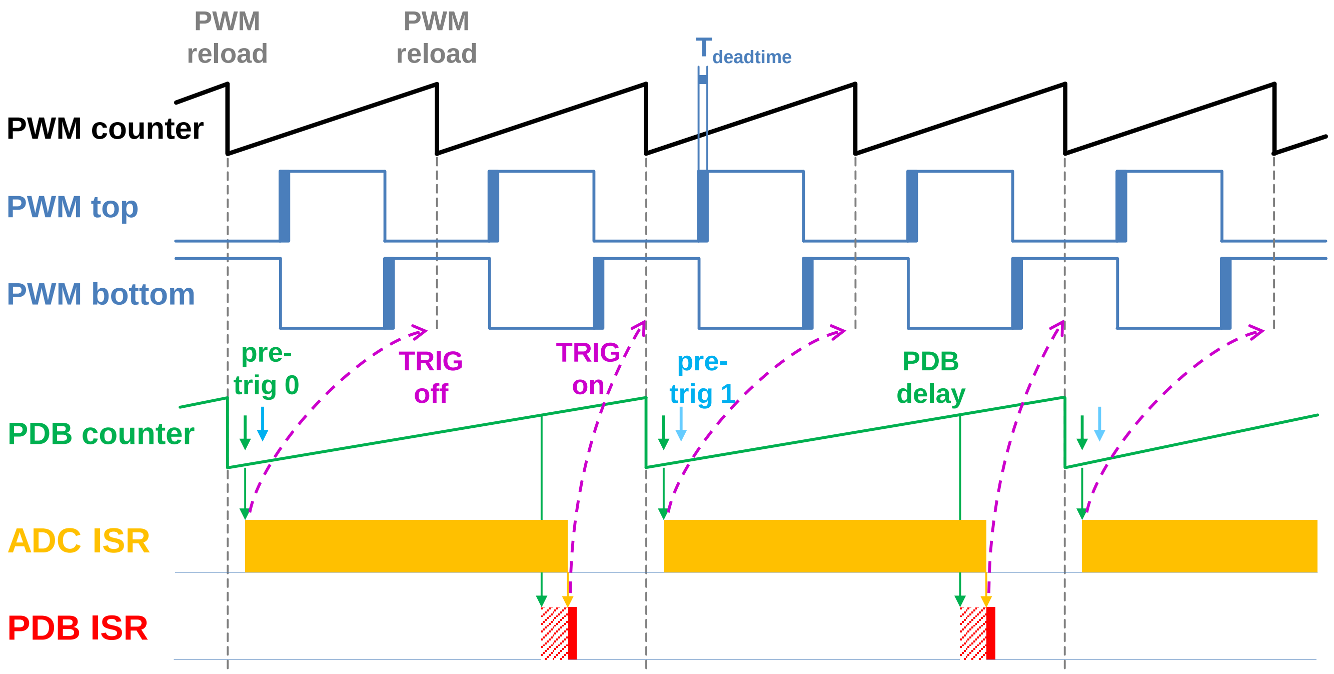


Figure 4. Hardware timing and synchronization on KV11Z and KV10Z

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized at the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated on the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates the first pre-trigger for the first ADC (phase current) sample with a delay of approximately $T_{deadtime}$ / 2. This delay ensures correct current sampling at duty cycles close to 100 %.
- When the conversion of the first ADC sample (phase current) is completed, the **ADC ISR** is entered. At first, the next FTM_TRIG is disabled (**TRIG off**). This ensures that the **PDB counter** is not reset at the next **PWM reload**. The FOC is then calculated.
- The **PDB ISR** is called in the middle of the next PWM period (**PDB delay**). This interrupt enables the FTM_TRIG (**TRIG on**) at the next **PWM reload**. The **PDB ISR** has lower priority than the **ADC ISR**. The **PDB delay** length determines the ratio between the PWM and FOC frequencies.
- The PDB uses the back-to-back mode to automatically generate the **pre-trig 1** (for DC-bus voltage measurement) immediately after the first conversion is completed.

3.1.2. Peripheral settings

The peripherals used for motor control are described in this section. On KV10Z and KV11Z, a 6-channel FlexTimer (FTM) is used for 6-channel PWM generation, and two 16-bit SAR ADCs are used for phase currents and DC-bus voltage measurement. The FTM and ADC are synchronized by the Programmable Delay Block (PDB). One channel from another independent FTM is used for slow loop interrupt generation.

3.1.2.1. PWM generation—FTM0

- The FTM is clocked from the 75 MHz System clock.
- Only six channels are used, the other two are masked in the OUTMASK register.
- The channels (0+1, 2+3, and 4+5) are combined into pairs, with each pair running in a complementary mode.
- The Fault mode is enabled for each combined pair 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 as a time needed for the FTM to count from CNTIN to MOD. By default, CNTIN = -MODULO / 2 = -3750 and MOD = MODULO / 2 - 1 = 3749. Because the FTM is clocked from the 75 MHz System clock, it takes 0.0001 s (10 kHz).
- Inserting dead time is enabled for each combined pair. The dead time length is calculated as System clock 75 MHz × $T_{deadtime}$. The dead time varies among platforms.
- The FTM generates a trigger to PDB at counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.1.2.2. Analog sensing—ADC0, ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 25 MHz Bus clock divided by 2 = 12.5 MHz.
- For ADC calibration purposes, the ADC clock is set to 3.125 MHz. Continuous conversion and averaging with 32 samples are enabled in the SC3 register. After the calibration is done, the SC register is filled with its default values and the clock is set back to 12.5 MHz.
- Both ADCs are triggered from the PDB pre-triggers.
- An interrupt that serves the FOC fast-loop algorithm is generated after the first conversion is completed.

3.1.2.3. PWM and ADC synchronization—PDB0

- Unlike FTM, the PDB is clocked from the Bus clock, which is 3× slower than the System clock (used for FTM). Therefore the modulo value at PDB is divided by 3.
- The PDB is triggered from the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated 0.5 × $T_{deadtime}$ after the FTM0_TRIG.

- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC.
- The PDB Delay interrupt is enabled. This interrupt is generated when the PDB_IDLY is reached. This interrupt enables the FTM_TRIG (see Figure 4).
- The PDB Sequence Error and PDB Delay interrupts share a common interrupt vector. Which event generates the interrupt is determined at the beginning of the interrupt according to the ERR flag.

3.1.2.4. Over-current detection at FRDM platform—CMP1

- The plus input to the CMP is taken from the analog pin.
- The minus input to the CMP is taken from the 6-bit DAC0 reference. The DAC reference is set to 3.197 V ($62 / 64 \times V_{DD}$) which corresponds to 7.73 A (in 8.25 A scale).
- The CMP filter is enabled and four consecutive samples must match.

3.1.2.5. Slow-loop interrupt generation—FTM2

- The slow loop is usually 10× slower than the fast loop. Therefore the FTM2 is clocked from the System clock / 16 to keep its modulo value reasonably low.
- The FTM counts from CNTIN = 0 to SPEED_MODULO.
- The interrupt is enabled and generated at the counter reload that serves the slow loop.

3.1.2.6. Communication with MC33937 MOSFET driver—SPI

- The SPI runs in the master mode.
- The SPI chip select 1 signal is active in logic high.
- The baud rate is 3.12 MHz.

3.1.3. Peripheral settings differences among platforms

There are differences in peripheral settings among different platforms. This table summarizes these differences:

Table 1. KV10 and KV11 platform differences

Peripheral	Feature	Platform		
		FRDM	**Tower System**	**HVP**
FTM0	PWM polarity	high sides active high low sides active high	high sides active low low sides active high	high sides active high low sides active high
	Fault source	FLT0, CMP1 out	FLT1, input pin	FLT0, input pin
	Fault polarity	Active high	Active high	Active low
	Dead time	0.5 µs	0.5 µs	1.5 µs
SPI	Driver on SPI	No	Yes	No
PDB	Pre-trigger 0 delay	0.25 µs	0.25 µs	0.75 µs

3.1.4. CPU load and memory usage

The following information apply to the *demo* application built using IAR® Embedded Workbench® IDE in release configuration (maximum optimization for speed). Table 4 shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.1 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). 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) frequency. In this case, it applies to the fast loop of 10 kHz and the slow loop of 1 kHz. The total CPU load is calculated according to these equations:

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

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

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

where:

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

Table 2. KV10 and KV11 CPU load and memory usage

—	MKV10Z	MKV11Z
CPU load [%]	73.5	59.3
Flash usage [B]	24 232	24 288
RAM usage [B]	3 853	3 857

3.2. KV3x family

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

3.2.1. Hardware timing and synchronization

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

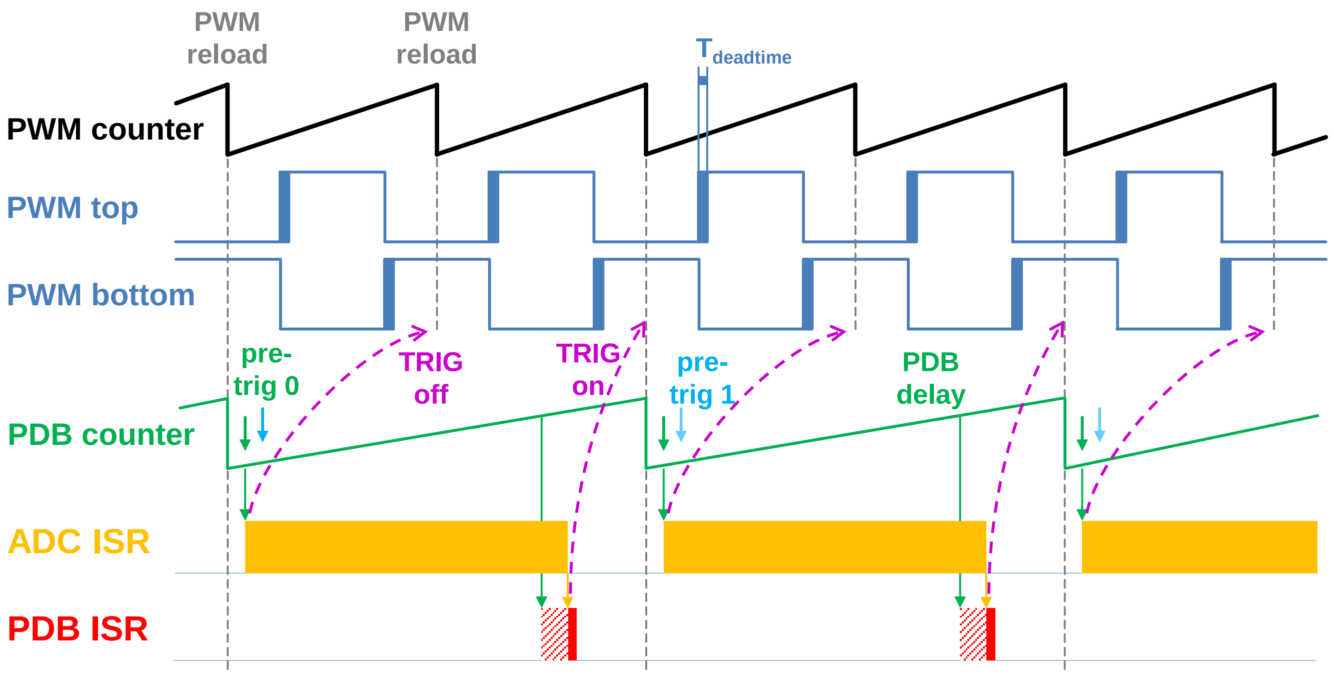


Figure 5. Hardware timing and synchronization on KV31F

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized on the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated at the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates a first pre-trigger for the first ADC (phase current) sample with a delay of approximately $T_{deadtime}$ / 2. This delay ensures correct current sampling at duty cycles close to 100 %.
- When the conversion of the first ADC sample (phase current) is completed, the **ADC ISR** is entered. At first, the next FTM_TRIG is disabled (**TRIG off**). This ensures that the **PDB counter** does not reset at the next **PWM reload**. The FOC is then calculated.
- In the middle of the next PWM period (**PDB delay**) the **PDB ISR** is called. This interrupt enables the FTM_TRIG (**TRIG on**) at the next **PWM reload**. The **PDB ISR** has lower priority than the **ADC ISR**. The **PDB delay** length determines the ratio between the PWM and FOC frequencies.
- The PDB uses back-to-back mode to automatically generate the **pre-trig 1** (to measure the DC-bus voltage) immediately after the first conversion is completed.

3.2.2. Peripheral settings

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

3.2.2.1. PWM generation—FTM0

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

3.2.2.2. Analog sensing—ADC0 and ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 48 MHz IRC48 clock, divided by 2 = 24 MHz.
- For ADC calibration purposes, the ADC clock is set to 6 MHz. Continuous the conversion and averaging with 32 samples are enabled in the SC3 register. After the calibration is done, the SC register is filled with its default values and the clock is set back to 24 MHz.
- Both ADCs are triggered by the PDB pre-triggers.
- There is an interrupt that serves the FOC fast-loop algorithm, and it is generated after the first conversion is completed.

3.2.2.3. PWM and ADC synchronization—PDB0

- Like the FTM, the PDB is clocked from the 60 MHz Bus clock.
- The PDB is triggered by the FTM0_TRIG.
- The pre-trigger 0 at each channel is generated $0.5 \times T_{deadtime}$ after the FTM0_TRIG.
- The pre-trigger 1 at each channel is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Sequence Error interrupt is enabled. This interrupt is generated when a certain result register is not read and the same pre-trigger occurs at the ADC.
- The PDB Delay interrupt is enabled. This interrupt is generated when the PDB_IDLY is reached. This interrupt enables the FTM_TRIG (Figure 5).
- The PDB Sequence Error and PDB Delay interrupts both share a common interrupt vector. Which event generated the interrupt is determined at the beginning of the interrupt according to the ERR flag.

3.2.2.4. FRDM platform over-current detection—CMP1

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

3.2.2.5. Slow-loop interrupt generation—FTM2

- The slow loop is usually 10× slower than the fast loop. Therefore the FTM2 is clocked from the System clock / 16 to keep its modulo value reasonably low.
- The FTM counts from CNTIN = 0 to MOD = SPEED_MODULO.
- The interrupt that serves the slow loop is enabled and generated at the reload.

3.2.2.6. Communication with MC33937 MOSFET driver—SPI

- The SPI runs in the master mode.
- The SPI chip-select 1 signal is active in logic high.
- The baud rate is 3.12 MHz.

3.2.3. Peripheral settings differences among platforms

There are differences in peripheral settings among different platforms. This table summarizes these differences:

Table 3. KV31 platform differences

Peripheral	Feature	Platform		
		Freedom	Tower System	HVP
FTM0	PWM polarity	high sides active high low sides active high	high sides active low low sides active high	high sides active high low sides active high
	Fault source	FLT1, CMP1 out	GPIO pin checked in SW, no HW connection to FTM.	FAULT 0, input pin
	Fault polarity	Active high	Active high	Active low
	Dead time	0.5 µs	0.5 µs	1.5 µs
SPI	Driver on SPI	No	Yes	No
PDB	Pre-trigger 0 delay	0.25 µs	0.25 µs	0.75 µs

3.2.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. Table 4 shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.1 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the *SysTick* timer. The CPU load depends on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 10 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated according to Eq. 3.

Table 4. KV31 CPU load and memory usage

—	MKV31F
CPU load [%]	29,4
Flash usage [B]	24 896
RAM usage [B]	3 797

3.3. KV4x family

The KV46F family of Kinetis MCUs is a high-performance solution built upon the ARM Cortex-M4 core running at 168 MHz with floating-point unit and up to 256 KB of flash and 32 KB of RAM. It is targeted mainly at motor-control applications. Advanced peripherals, such as high-resolution Pulse-Width Modulation (PWM) modules with a total of 30 PWM channels and dual 12-bit Analog-to-Digital Converters (ADCs) make these devices ideal for high-end motor-control applications. For more information, see *KV4x Reference Manual* (document KV4XP100M150RM).

3.3.1. Hardware timing and synchronization

A correct and precise timing is crucial in motor-control applications. The motor-control peripherals handle the timing and synchronization on the hardware layer. You can set the PWM frequency as a multiple of the ADC interrupt (FOC calculation) frequency, in this case $FOC_{freq} = PWM_{freq} / 2$. The timing diagram is shown in this figure:

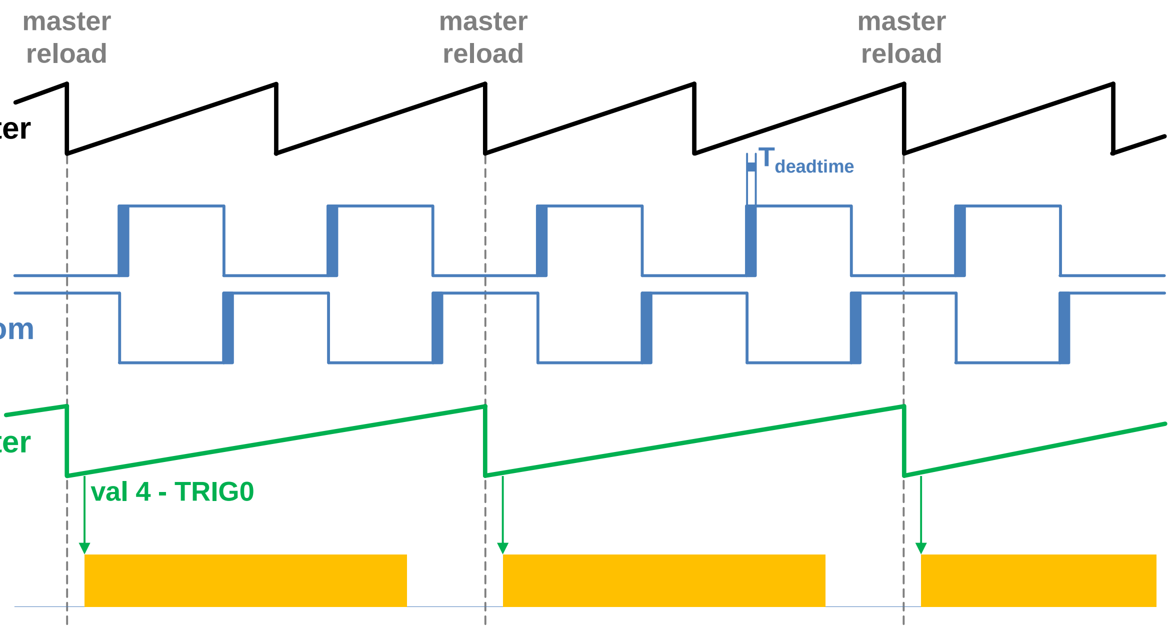


Figure 6. Hardware timing and synchronization on KV46F

- The top signal (**SM0 counter**) shows the eFlexPWM counter. The dead time is emphasized in the **PWM top** and **PWM bottom** signals. The **SM0** submodule generates the master reload at every second opportunity.
- The **SM3 counter** runs with a **SM0 counter** / 2 frequency, and its reload is synchronized with the master reload.
- The **SM3** generates a trigger (val4 – TRIG0) for the ADC scan with a delay of approximately $\mathbf{T}_{\mathbf{deadtime}}$ / 2. This delay ensures correct current sampling at duty cycles close to 100 %.
- When the ADC scan completes, the **ADC ISR** is entered. The FOC calculation is made in this interrupt.

3.3.2. Peripheral settings

Only the peripherals used for motor control are described in this section. On KV46F, three submodules from the enhanced FlexPWM (eFlexPWM) are used to generate a 6-channel PWM, and two 12-bit cyclic ADCs are used to measure the phase currents and DC-bus voltage. The eFlexPWM and ADC are synchronized using the fourth eFlexPWM submodule. One channel from the independent FTM is also used to generate the slow-loop interrupt.

3.3.2.1. PWM generation—PWMA

- The eFlexPWM is clocked from the 74 MHz Fast Peripheral clock.
- Six channels from three submodules are used to generate a 3-phase PWM. Submodule 0 generates a master reload event every nth opportunity, depending on user-defined `M1_FOC_FREQ_VS_PWM_FREQ`. Submodules 1, 2, and 3 are reloaded when the master reload occurs.
- Submodules 1 and 2 are clocked from submodule 0.
- The counters at submodules 1 and 2 are synchronized with the master sync signal from submodule 0. The counter at submodule 3 is synchronized with the master reload signal from submodule 0.
- Submodule 3 is used for synchronization with the ADC. The clock for submodule 3 is divided by 2 (37 MHz). The Val 4 register generates the output trigger $T_{deadtime}$ / 2 after the PWM reload.
- The fault mode is enabled for channels A and B at submodules 0, 1, and 2 with automatic fault clearing. The PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero). The PWM fault input pin and its polarity vary among platforms.
- The PWM period (frequency) is the time the counter needs to count from INIT to VAL1. By default INIT = -MODULO / 2 = -3700 and VAL1 = MODULO / 2 - 1 = 3699. The eFlexPWM clock runs at 74 MHz, so the frequency is 0.0001 s (10 kHz).
- Dead time insertion is enabled. The dead time length is calculated as the Fast Peripheral clock 74 MHz × $T_{deadtime}$. The dead time varies among platforms.

3.3.2.2. Analog sensing—ADC12

- The ADC12 wrapper contains two independent ADCs. The ADCs operate as 12-bit, single-ended converters. ADC12 operates in a triggered parallel mode (ADC0 and ADC1 convert SAMPLE0+SAMPLE8 and SAMPLE1+SAMPLE9 simultaneously). The ADC scan is triggered by the SYNC0 signal.
- The clock source for ADC12 is the 74 MHz Fast Peripheral clock divided by 3 = 24.6 MHz.
- Only SAMPLE0, SAMPLE1, SAMPLE8, and SAMPLE9 are enabled.
- The end-of-scan interrupt that serves the FOC fast loop algorithm is generated after the entire scan is completed.

3.3.2.3. Peripheral interconnections—XBARA

- The PWMA_TRG0 output trigger generated by submodule 3 is connected to the ADC_SYNC0 input.
- The over-current pin input signal is connected to the PWMA fault input that varies among platforms.

3.3.2.4. Slow loop interrupt generation—FTM1

- The slow loop is usually 10× slower than the fast loop. Therefore the FTM2 is clocked from the System clock / 16 to keep its modulo value reasonably low.
- The FTM counts from CNTIN = 0 to MOD = SPEED_MODULO.
- The interrupt enabled and generated at the reload serves the slow loop.

3.3.2.5. Communication with MC33937 MOSFET driver—SPI

- The SPI runs in the master mode.
- The SPI chip select 1 signal is active in logic high.
- The baud rate is 3.12 MHz.

3.3.3. Peripheral settings differences among platforms

There are some differences in peripheral settings among different platforms. This table summarizes those differences:

Table 5. KV46 platform differences

Peripheral	Feature	Platform	
		Tower System	HVP
PWMA	PWM polarity	high sides active low low sides active high	high sides active high low sides active high
	Fault source	FAULT 0, input pin	FAULT 1, input pin
	Fault polarity	Active high	Active low
	Dead time	0.5 μs	1.5 μs
	SM3_VAL4	10 dec (delay 0.27 μs)	28 dec (delay 0.76 μs)
SPI	Driver on SPI	Yes	No

3.3.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. Table 6 shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.1 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the SysTick timer. The CPU load depends on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 10 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated according to Eq. 3.

Table 6. KV46 CPU load and memory usage

—	MKV46F
CPU load [%]	20.6
Flash usage [B]	24 772
RAM usage [B]	3 753

3.4. KV5x family

The KV58F family of Kinetis MCUs is a high-performance solution built upon the ARM Cortex-M7 core running at 220 MHz with floating-point unit and up to 1 MB of flash and 64 KB of RAM. The advanced peripherals, such as high-resolution Pulse-Width Modulation (PWM) modules with a total of 42 PWM channels and four 12-bit high-speed Analog-to-Digital Converters (ADCs) with a sampling rate of 5 MSPS, make these devices ideal for high-end multi-motor control applications. For more information, see *KV5x Reference Manual* (document KV5XP144M220RM).

3.4.1. Hardware timing and synchronization

A correct and precise timing is crucial in motor-control applications. Therefore, the motor-control dedicated peripherals handle the timing and synchronization on the hardware layer. You can set the PWM frequency as a multiple of the ADC interrupt (FOC calculation) frequency, in this case $FOC_{freq} = PWM_{freq} / 2$. The timing diagram is shown in this figure:

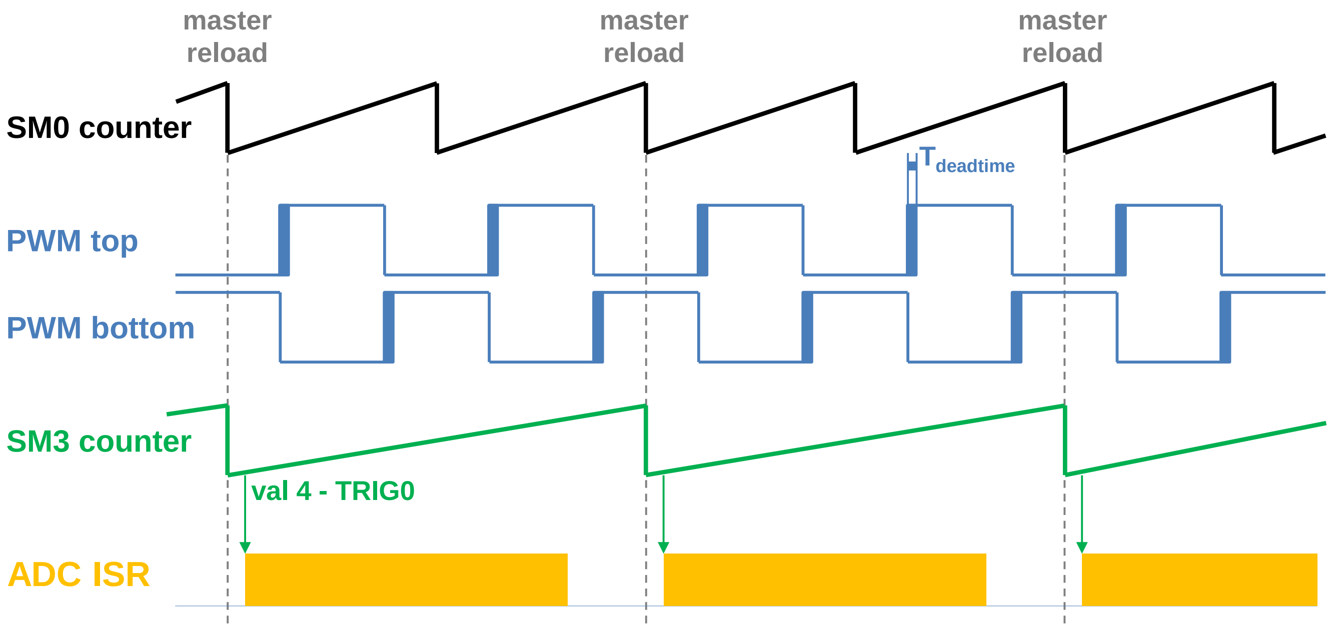


Figure 7. Hardware timing and synchronization on KV58F

- The top signal (**SM0 counter**) shows the eFlexPWM counter. The dead time is emphasized in the **PWM top** and **PWM bottom** signals. The **SM0** submodule generates the master reload every second opportunity.
- The **SM3 counter** runs with a frequency of **SM0 counter** / 2 and its reload is synchronized with master reload.
- The **SM3** generates a trigger (val4 – TRIG0) for the ADC scan with a delay of approximately $T_{deadtime}$ / 2. This delay ensures correct current sampling at duty cycles close to 100 %.
- When the ADC scan is completed, the **ADC ISR** is entered. The FOC calculation is done in this interrupt.

3.4.2. Peripheral settings

Only the peripherals used for motor control are described in this section. On KV46F, three submodules from the enhanced FlexPWM (eFlexPWM) are used to generate a 6-channel PWM, and two 12-bit high-speed ADCs are used to measure the phase currents and DC-bus voltage. The eFlexPWM and HSADC are synchronized via the fourth eFlexPWM submodule. One channel from an independent FTM is used to generate the slow-loop interrupt.

3.4.2.1. PWM generation—PWMA

- eFlexPWM is clocked from the 100 MHz Fast Peripheral clock.
- Six channels from three submodules are used to generate a 3-phase PWM. Submodule 0 generates the master reload event every n^{th} opportunity, depending on user-defined `M1_FOC_FREQ_VS_PWM_FREQ`. Submodules 1, 2, and 3 are reloaded when the master reload occurs.
- Submodules 1 and 2 are clocked from submodule 0.
- The counters at submodules 1 and 2 are synchronized with the master sync signal from submodule 0. The counter at submodule 3 is synchronized with the master reload signal from submodule 0.
- Submodule 3 is used for synchronization with the ADC. The clock for submodule 3 is divided by 2 (50 MHz). The Val 4 register generates the output trigger $T_{deadtime} / 2$ after the PWM reload.
- The fault mode is enabled for channels A and B at submodules 0, 1, and 2 with automatic fault clearing (PWM outputs are re-enabled the first PWM reload after the fault input returns to zero). The PWM fault input pin and its polarity vary among platforms.
- The PWM period (frequency) is determined as a time for the counter to count from INIT to VAL1. By default $INIT = -MODULO / 2 = -5000$ and $VAL1 = MODULO / 2 - 1 = 4999$. The eFlexPWM clock is 100 MHz, so the PWM period is 0.0001 s (10 kHz).
- The dead time insertion is enabled. The dead time length is calculated as the Fast peripheral clock $100\ \text{MHz} \times T_{deadtime}$. The dead time varies among the platforms.

3.4.2.2. Analog sensing—ADC12

- The HSADC wrappers are similar to the cyclic ADC12 wrapper at KV46. There are two wrappers—HSADC0 and HSADC1. The HSADC0A and HSADC1A are used for MC analog sensing.
- The clock source for HSADC0A and HSADC1A is the 100 MHz Fast Peripheral clock divided by 4 = 25 MHz.
- The ADCs operate as 12-bit, single-ended converters. ADC12 operates in a triggered sequential mode (HSADC0A converts SAMPLE0 and SAMPLE1, and HSADC1A also converts SAMPLE0 and SAMPLE1). Each HSADC scan is triggered by the SYNC0 generated by the eFlexPWM.
- Only SAMPLE0 and SAMPLE1 are enabled at each ADC.

- The end-of-scan interrupt that serves the FOC fast-loop algorithm is generated after the entire scan (SAMPLE0, SAMPLE1) is completed by HSADC0.

3.4.2.3. Peripheral interconnections—XBARA

- The PWM0_OUT_TRG30 output trigger generated by submodule 3 is connected to the HSADC0A_SYNC and HSADC1A_SYNC inputs.
- The over-current pin input signal is connected to the PWM0_FAULT0 fault input.

3.4.2.4. Slow-loop interrupt generation—FTM2

- The slow loop is usually 10× slower than the fast loop. Therefore the FTM2 is clocked from the System clock / 16 to keep its modulo value reasonably low.
- The FTM counts from CNTIN = 0 to MOD = SPEED_MODULO.
- The interrupt that serves the slow loop is enabled and generated at reload.

3.4.2.5. Communication with MC33937 MOSFET driver—SPI

- SPI runs in a master mode.
- The SPI chip select 1 signal is active in logic high.
- The baud rate is 3.12 MHz.

3.4.3. Peripheral settings differences among platforms

There are differences in peripheral settings among different platforms. This table summarizes those differences:

Table 7. KV58 platform differences

Peripheral	Feature	Platform
		Tower System
PWMA	PWM polarity	high sides active low low sides active high
	Fault source	FAULT 0, input pin
	Fault polarity	Active high
	Dead time	0.5 µs
	SM3_VAL4	13 dec (delay 0.26 µs)
SPI	Driver on SPI	Yes

3.4.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. Table 8 shows the memory usage and CPU load. The memory usage is calculated from the linker .map file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.1 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the SysTick timer. The CPU load depends on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 10 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated according to Eq. 3.

Table 8. KV58 CPU load and memory usage

—	MKV58F
CPU load [%]	28.7
Flash usage [B]	25 917
RAM usage [B]	3 793

3.5. KE1xZ family

The KE15Z is a part of Kinetis E series of ARM Cortex-M0+ MCUs. The Kinetis E series family is a product portfolio with an enhanced ESD/EFT performance for cost-sensitive, high-reliability applications used in the environments with high electrical noise.

Built upon the ARM Cortex-M0+ core running at 72 MHz with up to 256 KB of flash and 32 KB of RAM, it delivers a platform that enables you to build a scalable solution portfolio. For more information, see *KE1xZ Sub-Family Reference Manual* (document KE1xZP100M72SF0RM).

3.5.1. Hardware timing and synchronization

A correct and precise timing is crucial in motor-control applications. The motor-control dedicated peripherals handle the timing and synchronization on the hardware layer. The timing diagram is shown in this figure:

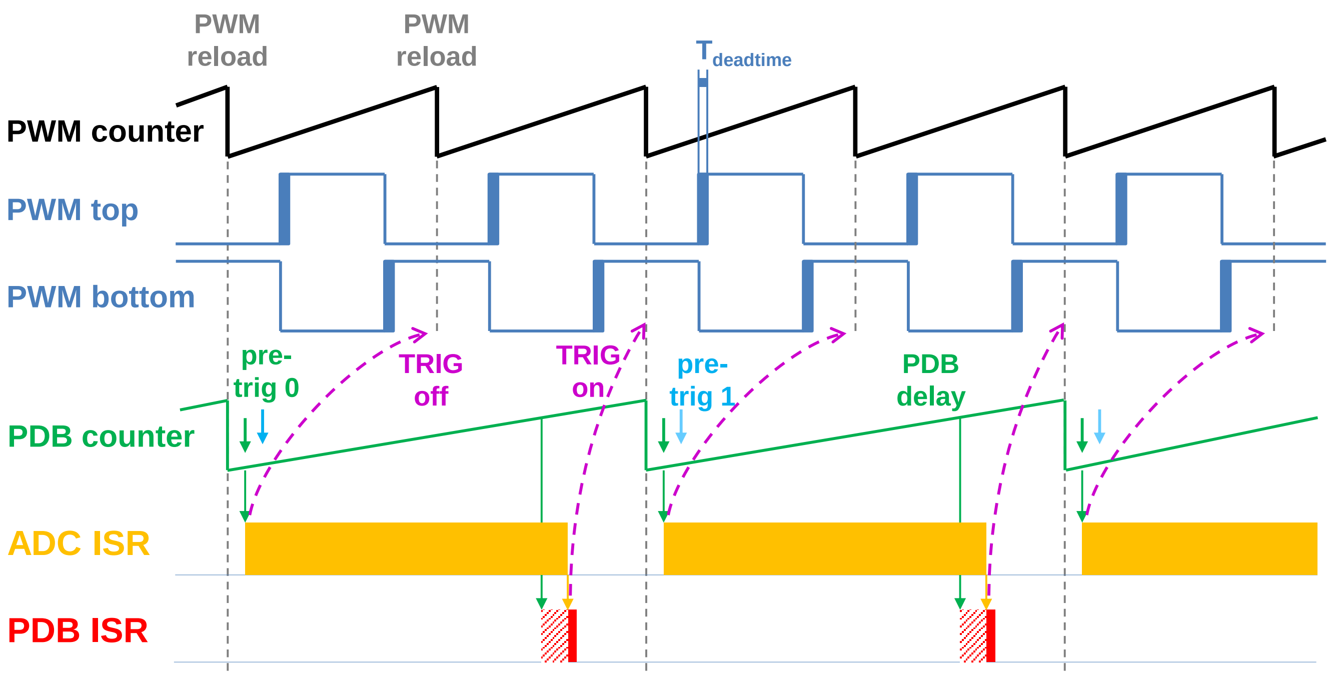


Figure 8. Hardware timing and synchronization on KV11Z and KV10Z

3.5.2. Peripheral settings

This section describes only the peripherals used for motor control. The KE15Z uses a 6-channel FlexTimer (FTM) to generate a 6-channel PWM and two 12-bit SAR ADCs to measure the back-EMF voltage, DC-bus current, and DC-bus voltage. The FTM and ADC are synchronized via the Programmable Delay Block (PDB). One channel from another independent FTM is used for the slow-loop interrupt generation.

3.5.2.1. PWM generation—FTM0

- The FTM is clocked from the 72-MHz System clock.
- Only six channels are used, the other two are masked in the OUTMASK register.
- Channels 0+1, 2+3, and 4+5 are combined in pairs and running in a complementary mode.
- The PWM period (frequency) is determined as a time for the FTM to count from CNTIN to MOD. By default, CNTIN = -MODULO / 2 = -3600 and MOD = MODULO / 2 - 1 = 3599. The FTM is clocked from the 72-MHz System clock, so the PWM period is 0.0001 s (10 kHz).
- The dead time insertion is enabled for each combined pair. The dead time length is calculated as System clock 72 MHz × $T_{deadtime}$. The dead time length is 0.5 µs.
- The FTM generates a trigger for the PDB on the counter initialization.

3.5.2.2. Analog sensing—ADC0 and ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 24-MHz Bus clock divided by 2 = 12 MHz.
- For the ADC calibration purposes, the ADC clock is set to 3 MHz. The continuous conversion and averaging with 32 samples are enabled in the SC3 register. After the calibration is done, the SC register is filled with its default values and the clock is set back to 12 MHz.
- Both ADCs are triggered by the PDB pre-triggers.
- An interrupt that serves for the fast-loop algorithm calculation is generated when the first conversion is completed.

3.5.2.3. PWM and ADC synchronization—PDB0

- Like the FTM, the PDB is clocked from the 72-MHz System clock.
- The PDB is triggered by the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated $0.5 \times T_{\text{deadtime}}$ after the FTM0_TRIG.
- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Delay interrupt is enabled. This interrupt is generated when the PDB_IDLY is reached. This interrupt enables the FTM_TRIG (see Figure 8).
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC. For more information about the PDB error sequence handling, see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document AN4822).

3.5.2.4. Slow-loop interrupt generation—FTM2

- The slow loop is usually 10× (or more) slower than the fast loop. Therefore, the FTM2 is clocked from the System clock $/$ 16 to keep its modulo value reasonably low.
- The FTM counts from $\text{CNTIN} = 0$ to $\text{MOD} = \text{SPEED\_MODULO}$.
- The interrupt that serves the slow loop is enabled and generated at the reload.

3.5.3. CPU load and memory usage

The following information apply to the demonstration application built using the IAR Embedded Workbench IDE. Table 9 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) and 3.1 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the SysTick timer. The CPU load depends on the fast-loop (FOC calculation) and slow-loop (speed-loop) frequencies. In this case, it applies to the fast-loop frequency of 10 kHz and the slow-loop frequency of 1 kHz. The total CPU load is calculated according to Eq. 3.

Table 9. KE15Z CPU load and memory usage

—	MKE15Z
CPU load [%]	58,2
Flash usage [B]	20 665
RAM usage [B]	3 727

3.6. KE1xF family

The KE18F is a part of Kinetis E series of ARM Cortex-M4 MCUs. This device is a 32-bit Kinetis MCU based on the ARM Cortex-M4 processor. It is an extension of the existing Kinetis E series MCU family with an enhanced CPU performance and additional memories and peripherals. This sub-family provides up to 168 MHz CPU performance, 512 KB flash, and 64 KB SRAM.

For more information, see *KE1xZ Sub-Family Reference Manual* (document KE1xFP100M168SF0RM).

3.6.1. Hardware timing and synchronization

A correct and precise timing is crucial in motor-control applications. The motor-control-dedicated peripherals handle the timing and synchronization on the hardware layer. The timing diagram is shown in this figure:

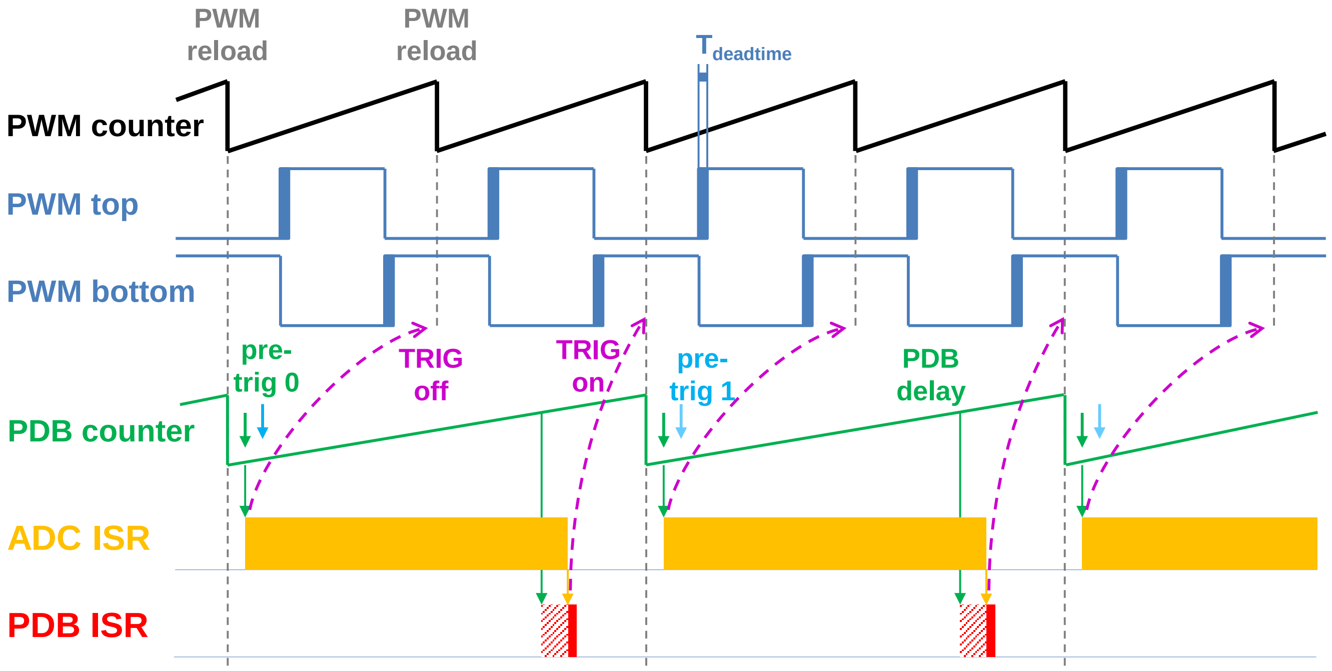


Figure 9. Hardware timing and synchronization on KV11Z and KV10Z

3.6.2. Peripheral settings

This section describes only the peripherals used for motor control. The KE18F uses a 6-channel FlexTimer (FTM) to generate a 6-channel PWM, and two 12-bit SAR ADCs to measure the back-EMF voltage, DC-bus current, and DC-bus voltage. The FTM and ADC are synchronized via the

Programmable Delay Block (PDB). One channel from another independent FTM is used for the slow-loop interrupt generation.

3.6.2.1. PWM generation—FTM0

- The FTM is clocked from the 168-MHz System clock.
- Only six channels are used, the other two are masked in the OUTMASK register.
- Channels 0+1, 2+3, and 4+5 are combined in pairs and running in a complementary mode.
- The PWM period (frequency) is determined as a time for the FTM to count from CNTIN to MOD. By default, CNTIN = -MODULO / 2 = -8400 and MOD = MODULO / 2 - 1 = 8399. The FTM is clocked from the 168-MHz System clock, so it takes 0.0001 s (10 kHz).
- The dead time insertion is enabled for each combined pair. The dead time length is calculated as System clock 168 MHz × $T_{deadtime}$. The dead time length is 0.5 μs.
- The FTM generates a trigger for the PDBs on the counter initialization.

3.6.2.2. Analog sensing—ADC0 and ADC2

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 84-MHz Bus clock divided by 2 = 42 MHz.
- For the ADC calibration purposes, the ADC clock is set to 10.5 MHz. The continuous conversion and averaging with 32 samples are enabled in the SC3 register. After the calibration is done, the SC register is filled with its default values and the clock is set back to 42 MHz.
- Both ADCs are triggered by the PDB pre-triggers.
- The interrupt that serves for the fast-loop algorithm calculation is generated when the first conversion is completed.

3.6.2.3. PWM and ADC synchronization—PDB0, PDB2

- Like the FTM, the PDB is clocked from the 168-MHz System clock.
- The PDB is triggered by the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated 0.5 × $T_{deadtime}$ after the FTM0_TRIG.
- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Delay interrupt is enabled. This interrupt is generated when the PDB_IDLY is reached. This interrupt enables the FTM_TRIG (see Figure 9).
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC. For more information about the PDB error sequence handling, see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document AN4822).

3.6.2.4. Slow-loop interrupt generation—FTM2

- The slow loop is usually 10× (or more) slower than the fast loop. Therefore, the FTM2 is clocked from the System clock / 16 to keep its modulo value reasonably low.
- The FTM counts from CNTIN = 0 to MOD = SPEED_MODULO.
- The interrupt that serves the slow loop is enabled and generated at the reload.

3.6.2.5. Communication with MC33937 MOSFET driver—LPSPI

- The SPI runs in the master mode.
- The SPI chip select 2 signal is active in the logic high.
- The baud rate is 0.5 MHz.

3.6.3. CPU load and memory usage

The following information apply to the demonstration application built using the IAR Embedded Workbench IDE. Table 10 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) and 3.1 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the SysTick timer. The CPU load depends on the fast-loop (FOC calculation) and slow-loop (speed-loop) frequencies. In this case, it applies to the fast-loop frequency of 10 kHz and the slow-loop frequency of 1 kHz. The total CPU load is calculated according to Eq. 3.

Table 10. KE18F CPU load and memory usage

—	MKE18F
CPU load [%]	13,9
Flash usage [B]	23 247
RAM usage [B]	3 785

4. Motor Control Peripheral Drivers

Motor Control Peripheral Drivers (MCDRV) represent a simple way of peripheral initialization and access for the purposes of 3-phase ACIM or PMSM control. The features provided by the MCDRV library are 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 PWM generation using the Space Vector Modulation (SVM) technique are described in *Sensorless PMSM Field-Oriented Control* (document DRM148).

The MCDRV are divided into two parts:

- The first part is the peripheral initialization module, consisting of *mcdrv_<platform>-<device>.c* and *mcdrv_<platform>-<device>.h* files, which are unique for each supported device. The header file includes all MCDRV setup options including the ADC channel assignment. The source file contains the functions to initialize all peripherals used for motor control. This module is described in Section 4.2, "Motor Control Peripheral Drivers API".
- The second part consists of the peripheral driver library modules for each supported periphery.

Generally, all ADC and PWM periphery drivers share the same API within their class. This enables the higher-level code to be platform-independent, as the peripheral driver function calls were replaced by universally named macros. The list of supported peripherals and the API of their drivers is described in Section 4.2, "Motor Control Peripheral Drivers API".

4.1. Motor Control Peripheral Drivers initialization

The MCDRV initialization module consists of a set of MCU peripheral-initialization functions, as well as all the definitions that you can specify. The functions are contained in device-specific *mcdrv_<platform><device>.c* source and *mcdrv_<platform><device>.h* header files. Out of all functions in the MCDRV initialization module, call the *MCDRV_Init_M1()* function during MCU startup and before calling any other MCDRV functions. All peripherals used by a given device for motor-control purposes are initialized within this function.

The *mcdrv_<platform><device>.h* header files offer several macros that you can define:

- M1_MCDRV_ADC—this macro specifies the ADC periphery used. If you select an unsupported periphery, the preprocessor error is issued.
- M1_MCDRV_PWM3PH—this macro specifies the PWM periphery used. If you select an unsupported periphery, the preprocessor error is issued.
- M1_PWM_FREQ—PWM frequency, for example, 10 kHz.
- M1_FOC_FREQ_VS_PWM_FREQ—enables you to call the fast loop interrupt every 1st, 2nd, 3rd, and nth PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast-loop interrupt length (running out of CPU performance).
- M1_PWM_PAIR_PH[A..C]—these macros enable 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[0,1]_PH_[A..C]—these macros serve to assign the ADC channels for phase-current measurement. The general rule is that at least one of the phase currents must be measurable on both ADC converters and the remaining two phase currents must be measurable on different ADC converters. The reason for this is that the selection of the phase-current pair to measure depends on the current SVM sector. When this rule is broken, preprocessor error is issued. For more information about the 3-phase current measurement, see *Sensorless PMSM Field-Oriented Control* (document DRM148). The unassigned ADC channels are set to the MCDRV_CHAN_OFF value.
- M1_ADC[0,1]_UDCB and M1_ADC[0,1]_AUX—use these defines to select the ADC channel to measure the DC-bus voltage and one user-defined auxiliary quantity, which is not used directly for motor control (the IPM temperature is measured by default). The rule for the ADC channel assignment is that the DC-bus voltage and the auxiliary quantity must be measurable on different ADC converters. If this rule is broken, preprocessor error is issued.

4.2. Motor Control Peripheral Drivers API

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

The available API for the ADC MC drivers is:

- *bool_t M1_MCDRV_ADC_PERIPH_INIT*()—by default, this function is called during the ADC peripheral initialization procedure invoked by the *MCDRV_Init_M1*() function, and it must 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 the proper ADC channels for the next 3-phase current measurement based on the SVM sector. This function always returns *true*.
- *bool_t M1_MCDRV_CURR_3PH_CALIB_INIT*(*MCDRV_ADC_T**)—this function initializes the phase-current channel-offset measurement. This function always returns *true*.
- *bool_t M1_MCDRV_CURR_3PH_CALIB*(*MCDRV_ADC_T**)—this function reads the current information from the unpowered phases of a standstill motor and filters them using moving average filters. The goal is to obtain the value of the measurement offset. The length of the window for moving 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*.

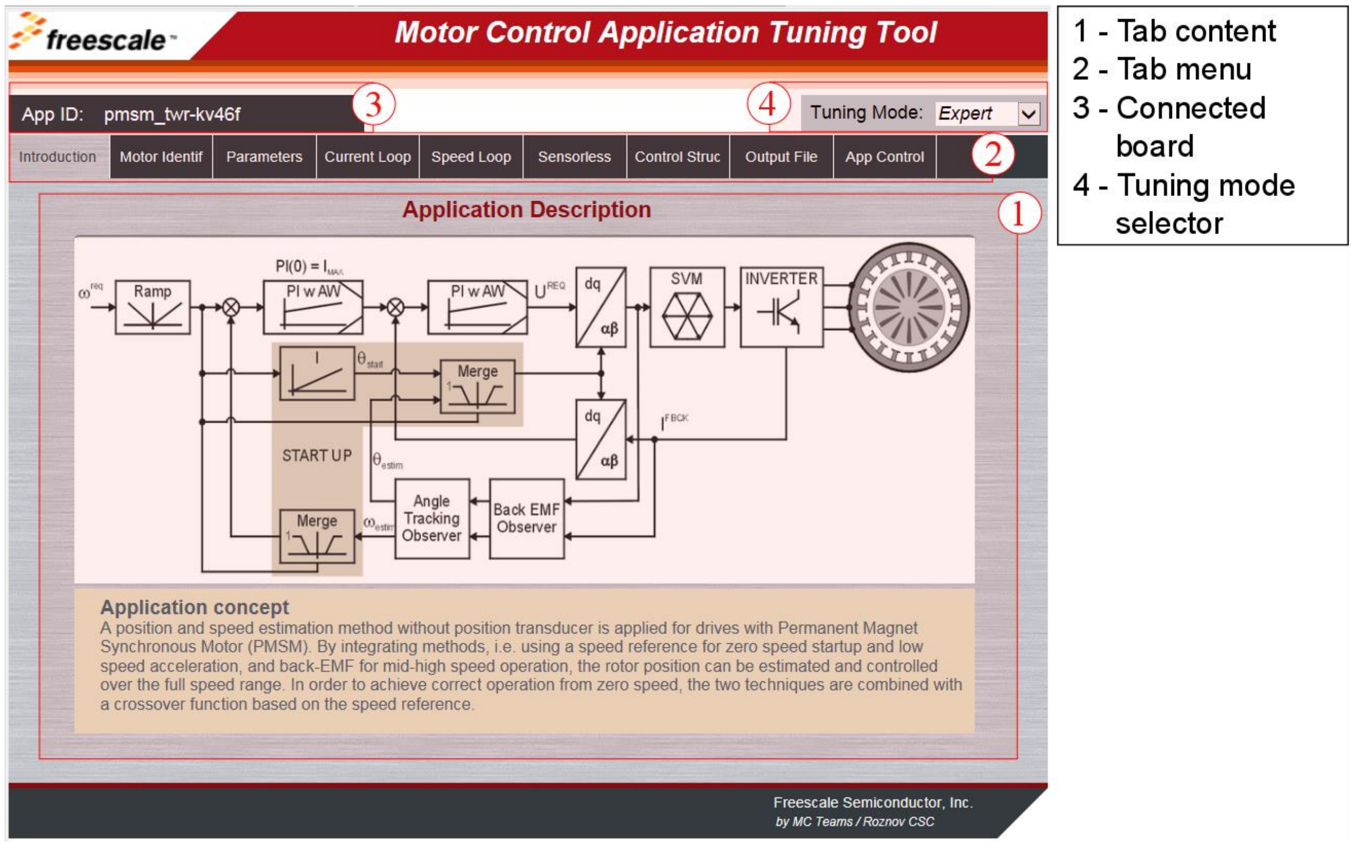
The API for the PWM MC drivers is:

- *bool_t M1_MCDRV_PWM_PERIPH_INIT* (*M1_MCDRV_PWM_T**)—this function is called by default during the PWM periphery-initialization procedure invoked by the *MCDRV_Init_M1*() function. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_SET*(*M1_MCDRV_PWM_T**)—this function updates the PWM phase duty cycles based on the required values stored in the M1_MCDRV_PWMIO_DUTY variable. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_EN*(*M1_MCDRV_PWM_T**)—calling this function enables all PWM channels. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_DIS* (*M1_MCDRV_PWM_T**)—calling this function disables all PWM channels. This function always returns *true*.
- *bool_t M1_MCDRV_PWM3PH_FAULT_GET*(*M1_MCDRV_PWM_T**)—this function returns the state of the over-current fault flags and automatically clears the flags (if set). This function returns *true* when the over-current event occurs, otherwise, it returns *false*.

5. Tuning and Controlling the Application

This section provides information about the tools and recommended procedures for controlling the sensorless PMSM Field-Oriented Control (FOC) application. The application contains the embedded-side driver of 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 algorithm tuning. Besides the target-side driver, FreeMASTER requires installing the PC application as well. For more information, visit www.nxp.com/freemaster.

Control and tune the PMSM sensorless FOC application easily using Motor Control Application Tuning (MCAT) page for PMSM. The MCAT for PMSM is a user-friendly modular page, which runs within FreeMASTER. To launch it, execute the *.pmp* file located next to your project. See the user’s guide for your version of MCRSP_PMSM for more information (document MCRSPPMSMUG). Figure 10 shows the MCAT for PMSM welcome page. The tool consists of a tab menu (point one), tuning mode selector (point two), and the workspace (point three). Each tab represents a submodule, which enables tuning or controlling different aspects of the application. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the variable watch window are predefined in the FreeMASTER project file to further simplify motor parameter tuning and debugging. The Basic and Expert tuning modes are available. Selecting the Expert mode grants you the access to modify all parameters and fields available in MCAT. The Basic mode is intended for inexperienced users. When FreeMASTER is not connected to the target, the “App ID” line shows “offline”. When the communication with the target MCU (with correct software) is established, the “App ID” line displays the MCU and platform, and all stored parameters for the given MCU are loaded.



Application concept

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

Figure 10. MCAT layout

In the default configuration, these tabs are available:

- “Introduction”—welcome page with the PMSM sensorless FOC diagram and a short description of the application.
- “Motor Identif”—PMSM semi-automated parameter-measurement control page. The PMSM parameter identification is described in detail later on in this document.
- “Parameters”—this page enables you to modify the motor parameters, the specification of hardware and application scales, and fault limits.
- “Current Loop”—current-loop PI controller gains and output limits.
- “Speed Loop”—this tab contains fields to specify the speed controller proportional and integral gains as well as the output limits and parameters of the speed ramp.
- “Sensorless”—this page enables you to tune the parameters of the BEMF observer, tracking observer, and open-loop startup.
- “Control Struc”—the application control page enables you to select and control the PMSM using different techniques (scalar—Volt/Hertz control, voltage FOC, current FOC, and speed FOC). The application state is also shown in this tab.
- “Output file”—this tab enables the user to view all calculated constants that are required by the PMSM sensorless FOC application. It also enables you to generate the *m1_pmsm_appconfig.h* file, which is then used to preset all application parameters permanently at project rebuild.

- “Control page”—this tab contains graphical elements such as speed gauges, DC-bus voltage measurement bar, and variety of switches that enable simple, quick, and user-friendly application control. Here you can control the fault-clearing and demo mode, which sets various predefined required speeds over time.

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

The following sections provide simple instructions for identifying the parameters of a connected PMSM, and tuning the application appropriately.

5.1. PMSM parameter identification

Because the model-based control methods of 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. To implement the FOC algorithms, you must know the values of stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and BEMF constant K_e.

5.1.1. Power stage characterization

Each inverter introduces 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 11. The power stage characterization is a part of MCAT, which can be controlled using the “Motor Identif” tab. To perform the characterization, connect the motor with a known stator resistance R_s, and set this value in the “Calib Rs” field. Then specify the “Calibration Range”, which is the range of the stator current i_s, in which the measurement of U_{error} is to be performed. Start the characterization by clicking the “Calibrate” button. The characterization gradually performs 65 i_{sd} current steps (from $i_s = -I_{s,calib}$ to $i_s = I_{s,calib}$), with each step lasting 300 ms. The whole process then 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. Perform the following R_s measurement with a maximum current $I_{s,calib}$. It is recommended to use a motor with a low R_s for characterization purposes.

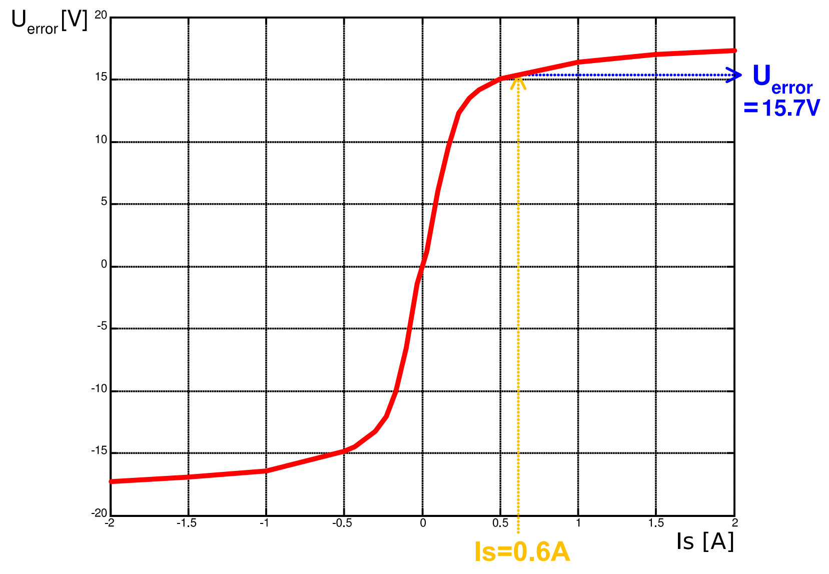


Figure 11. Example power stage characteristic

The power stage characterization is necessary only when you use your own hardware. When using NXP power stages with the application, omit the characterization process. The acquired characterization data is saved to a file, so it is necessary to do it only once for a given hardware.

5.1.2. Stator resistance measurement

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

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

5.1.3. Stator inductance

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

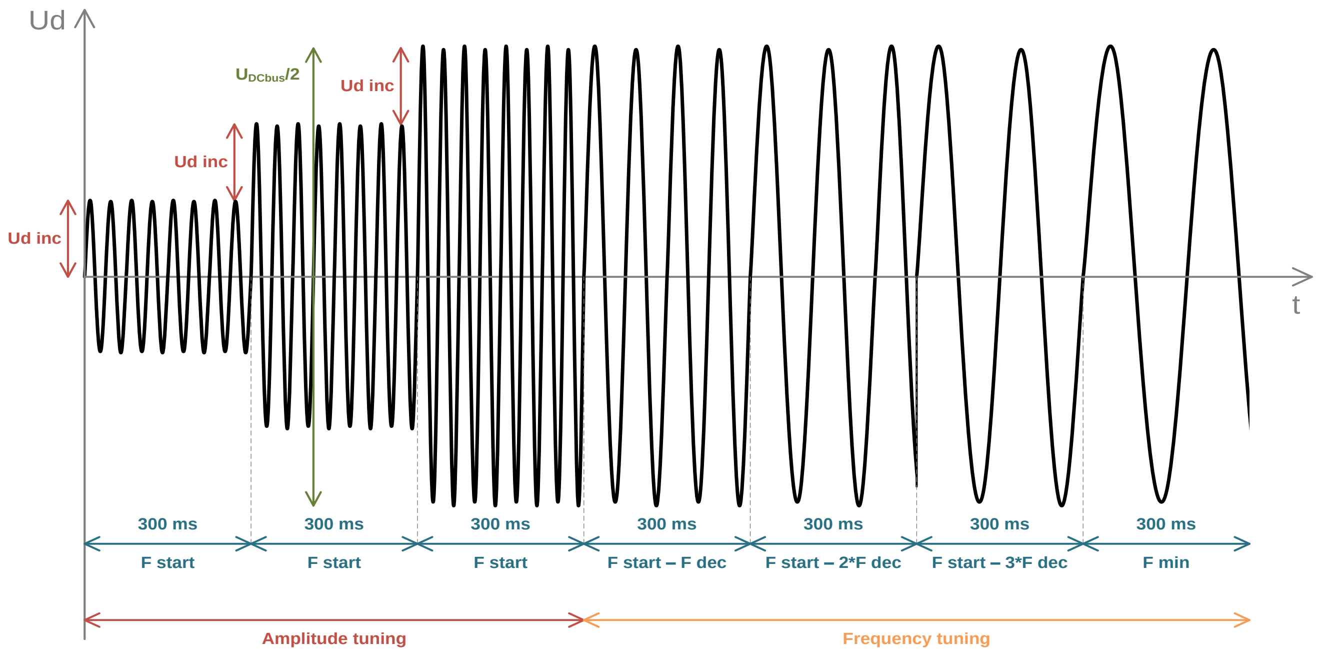


Figure 12. Tuning Ls measurement signal

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.

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

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

Eq. 7 $$L_s = \frac{X_{Ls}}{2\pi f}\ [\mathrm{H}]$$

The direct inductance (L_d) and quadrature inductance (L_q) measurements are done in the same way as the L_S measurement. Before the L_d and L_q measurements take place, DC current is applied to the D-axis, which aligns the rotor. For L_d measurement, the sinusoidal voltage is applied in the D-axis, and for 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, MCAT calculates the current controllers and BEMF-observer constants from the previously measured R_s, L_d, and L_q. To measure the K_e, the motor must be spinning. The I_d is controlled using I_s *DC*, and the electrical open-loop position is generated by integrating the required speed, 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 as follows:

Eq. 8 $$K_e = \frac{U_{BEMF}}{\omega_{el}}\ [\frac{V.s}{rad}]$$

While K_e is being measured, look at the motor to determine whether it is spinning properly. If the motor is not spinning properly, perform these steps:

- Ensure that the number of pole-pairs (*pp*) is correct. The required speed for the K_e measurement is also calculated from *pp*, so an incorrect *pp* causes an incorrect K_e.
- Increase the value of I_s *DC* to produce a higher torque when spinning during the open loop.
- Decrease the value of *N nom* to decrease the required speed for the K_e measurement.

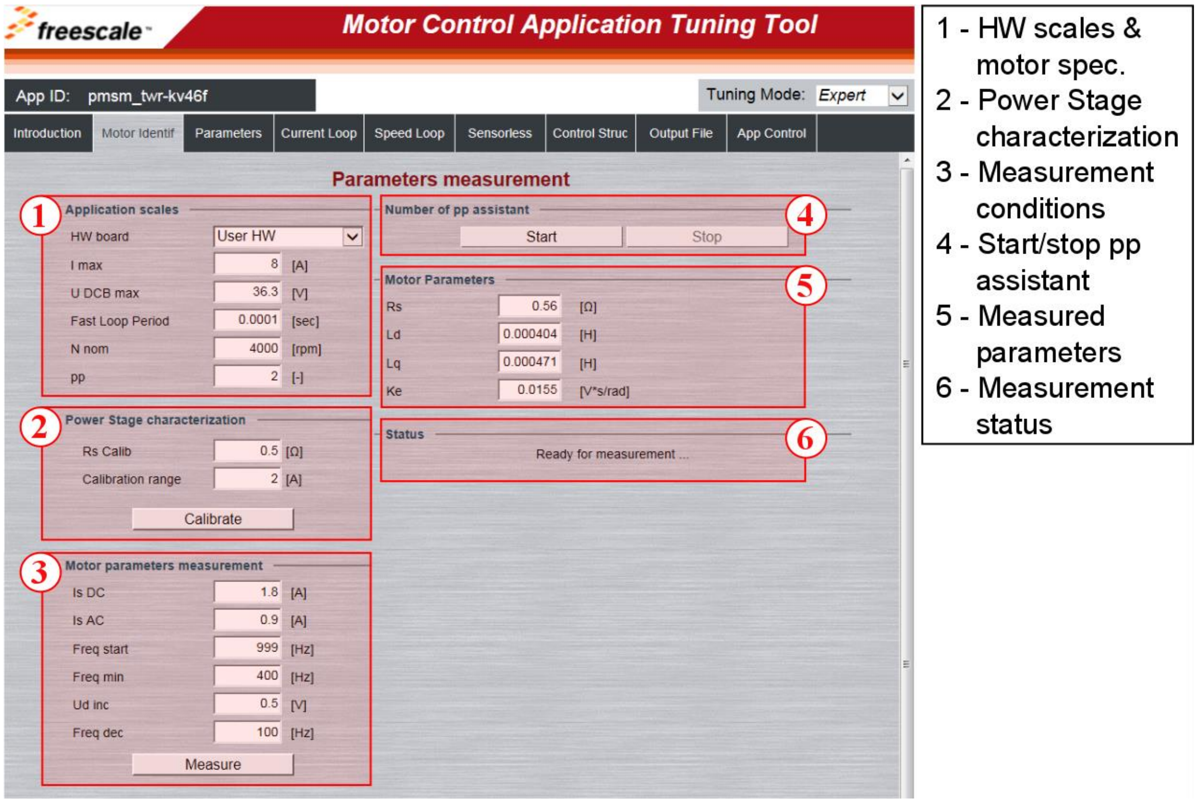
5.1.5. Number of pole-pairs assistant

The number of pole-pairs cannot be measured without a position sensor, however, there is a simple assistant to determine the number of *pp*. The number of pole-pairs assistant performs one electrical revolution and stops for a few seconds, and then repeats it. Because the *pp* value is a ratio between the electrical and mechanical speeds, it is 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 which affects the number of stops occurs during the first revolution. During the *pp* measurement, the current loop is enabled, and current I_d is controlled to I_s *DC*. The electrical position is generated by integrating the open-loop speed. If the rotor does not move after starting the number of pole-pairs assistant, stop the assistant, increase I_s *DC*, and restart the assistant.

5.1.6. PMSM electrical parameter measurement process

Control and set up the motor identification process using the MCAT “Motor Identif” tab, which is shown in Figure 13. To measure your own motor, follow these steps (shown in Figure 14):

- Select your hardware board. You can select the standard NXP hardware or use your own. If you use your own hardware, specify its scales (*I max*, *U DCB max*, *Fast Loop Period*).
- If you don’t know the number of motor pole-pairs, use the number of pole-pairs assistant described in Section 5.1.5, “Number of pole-pairs 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 “Basic” or “Expert” mode) and start the measurement by pressing the “Measure” button. You can observe which parameter is being measured in the “Status” bar.



Motor Control Application Tuning Tool

App ID: pmsm_twr-kv46f

Tuning Mode: Expert

Parameters measurement

Application scales

HW board	User HW	
I max	8	[A]
U DCB max	36.3	[V]
Fast Loop Period	0.0001	[sec]
N nom	4000	[rpm]
pp	2	[-]

Power Stage characterization

Rs Calib	0.5	[Ω]
Calibration range	2	[A]

Motor parameters measurement

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

Motor Parameters

Rs	0.56	[Ω]
Ld	0.000404	[H]
Lq	0.000471	[H]
Ke	0.0155	[V*s/rad]

Status: Ready for measurement ...

1 - HW scales & motor spec.
2 - Power Stage characterization
3 - Measurement conditions
4 - Start/stop pp assistant
5 - Measured parameters
6 - Measurement status

Figure 13. PMSM identification tab

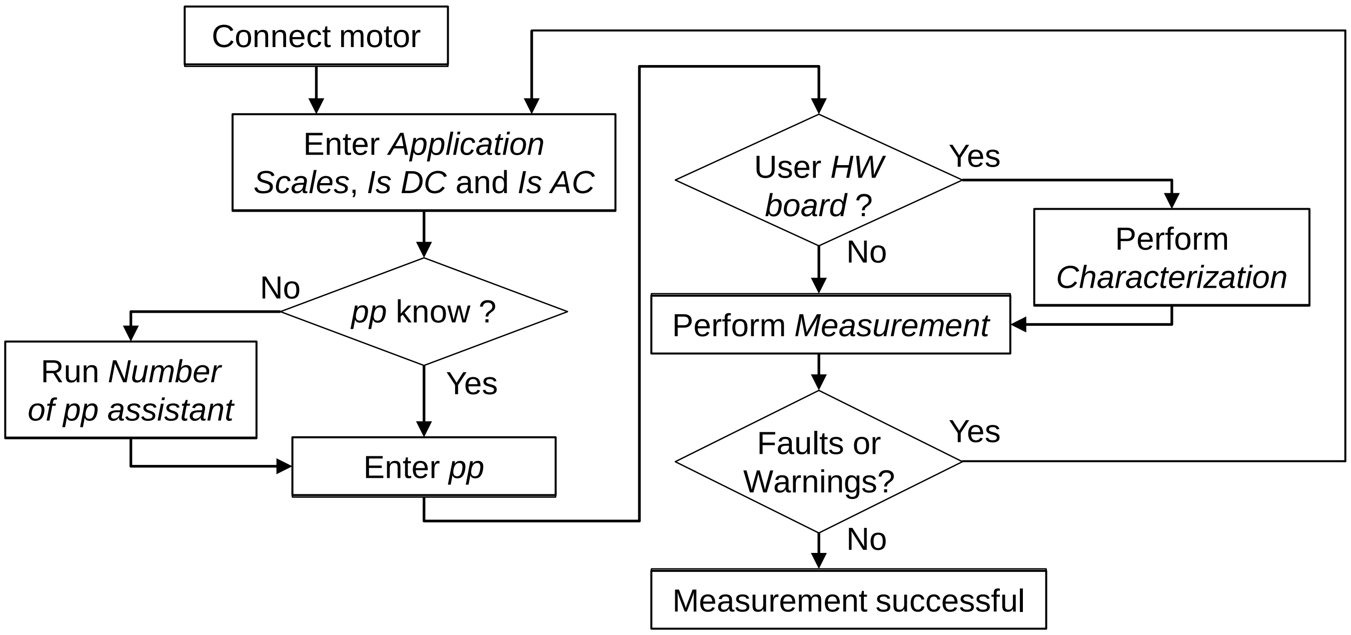


Figure 14. Measurement process diagram

Faults and warnings may occur during the measurement. Do not confuse these faults with application faults such as over-current, under-voltage, and others. The list of these faults and their description and possible troubleshooting is shown in this table:

Table 11. Measurement faults and warnings

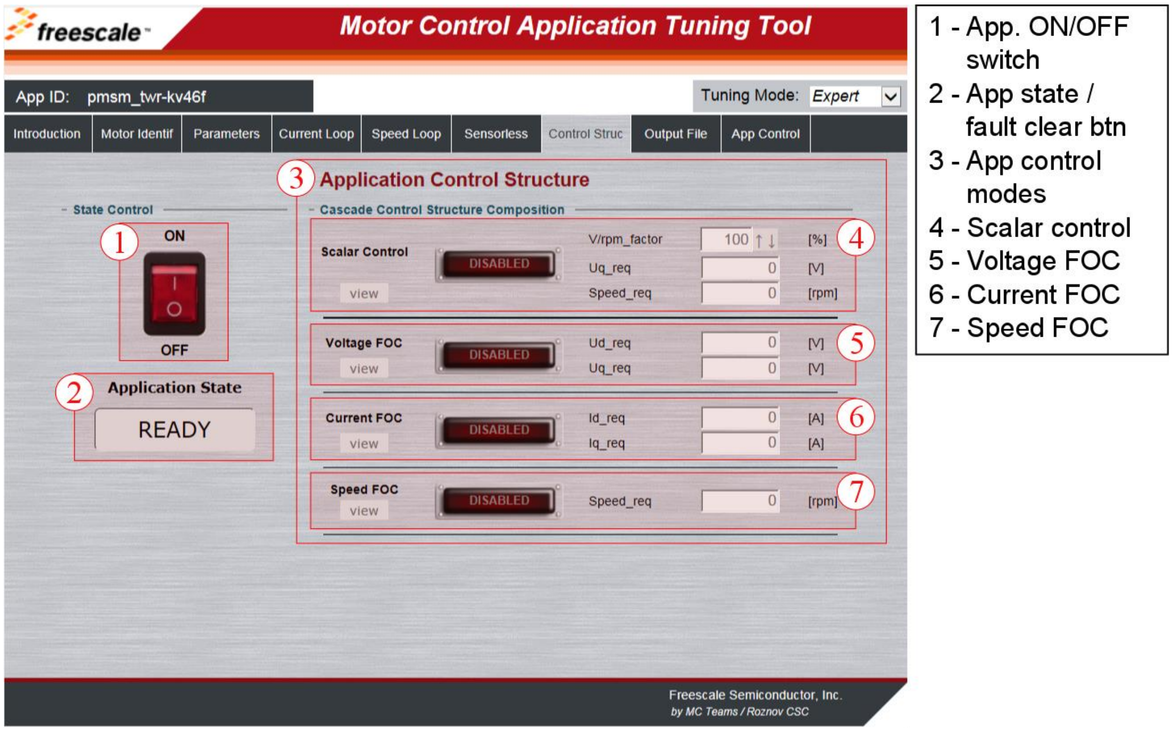
Fault no.	Fault description	Fault reason	Troubleshooting
01	Motor not connected	I_s > 50 mA cannot be reached with the available DC-bus voltage.	Check that the motor is connected.
02	R_s too high for calibration	Calibration *I* cannot be reached with the available DC-bus voltage.	Use a motor with a lower R_s for power stage characterization.
03	Current measurement I_s *DC* not reached	User-defined I_s *DC* is not reached, so the measurement is 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.
04	Current amplitude measurement I_s *AC* not reached	User-defined I_s *AC* is not reached, so the measurement is 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.
05	Wrong characteristic data	Characteristic data that is used for voltage correction does not correspond to the actual power stage.	Select “User HW” and perform the calibration.

5.2. PMSM sensorless application control and tuning using MCAT

Use FreeMASTER enabled with the MCAT page to control and tune the PMSM sensorless FOC application easily. The MCAT for PMSM submodule tabs are described here.

5.2.1. Application control using MCAT

Control the application using the “Control Struc” tab (shown in Figure 15). The “State Control” area on the left-hand side of the screen shows the current application state and enables turning the main application switch on or off (turning the running application off disables all PWM outputs).
The “Cascade Control Structure Composition” area is on the right-hand side of the screen.
Choose between the scalar and FOC control using the appropriate buttons. Enable the selected parts of the FOC cascade structure by selecting “Voltage FOC”, “Current FOC”, or “Speed FOC”. This is useful for application tuning and debugging.



1 - App. ON/OFF switch
2 - App state / fault clear btn
3 - App control modes
4 - Scalar control
5 - Voltage FOC
6 - Current FOC
7 - Speed FOC

Figure 15. MCAT for PMSM control page

The scalar control diagram is in Figure 16. It is the simplest type of a motor-control technique. Keep the ratio between the magnitude of the stator voltage and the frequency (frequency information is hidden in the “Speed_req” value) at the nominal ratio. This control method is sometimes called Volt per Hertz or V/Hz. Pay attention when entering the required voltage and speed in the Expert tuning mode. The ratio stays constant in the Basic mode and only the speed is required. The position-estimation BEMF observer and the tracking observer algorithms are running in the background even if the estimated position information is not directly used. See *Sensorless PMSM Field-Oriented Control* (document DRM148) for more information. This is useful for BEMF observer tuning.

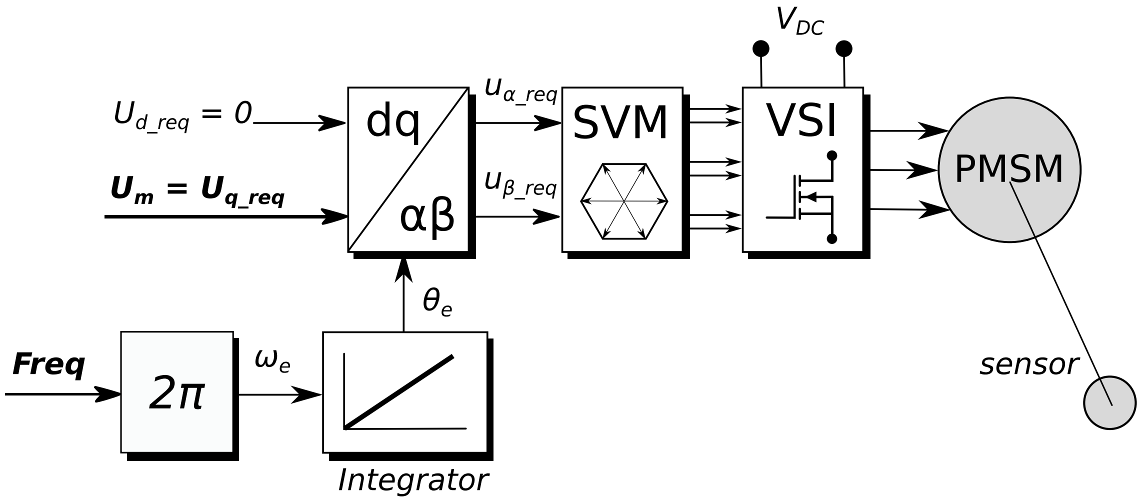


Figure 16. Scalar control mode

The block diagram of the voltage FOC is shown in the following figure. As opposed to scalar control, the position feedback is closed using the BEMF observer, and the stator voltage magnitude is not dependent on motor speed. Specify the *d*-axis and *q*-axis stator voltages using the “Ud_req” and “Uq_req” fields. This control method is useful for the BEMF observer functionality check.

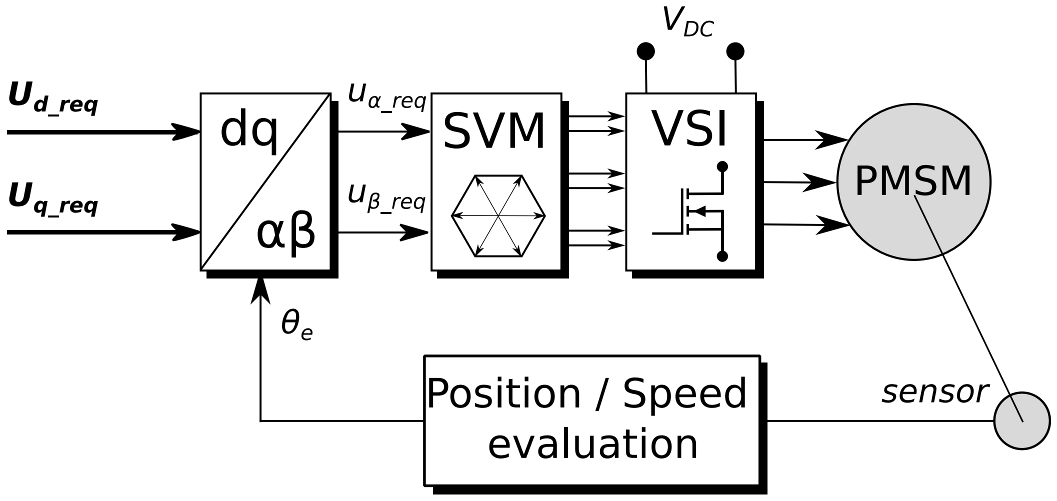


Figure 17. Voltage FOC control mode

The current FOC (or torque control) requires transforming the rotor position feedback (as well as the currents) into the d-q reference frame. The reference variables “Id_req” and “Iq_req” are available for motor control (see the following figure). The *d*-axis current component $i_{\text{sd\_req}}$ is responsible for the rotor-flux control, while the *q*-axis current component of the current $i_{\text{sq\_req}}$ generates torque, and the motor runs when it is applied. When changing the polarity of the current $i_{\text{sq\_req}}$, the motor changes the rotation direction. When tuning the BEMF observer correctly, tune the current PI controllers using the current FOC control structure.

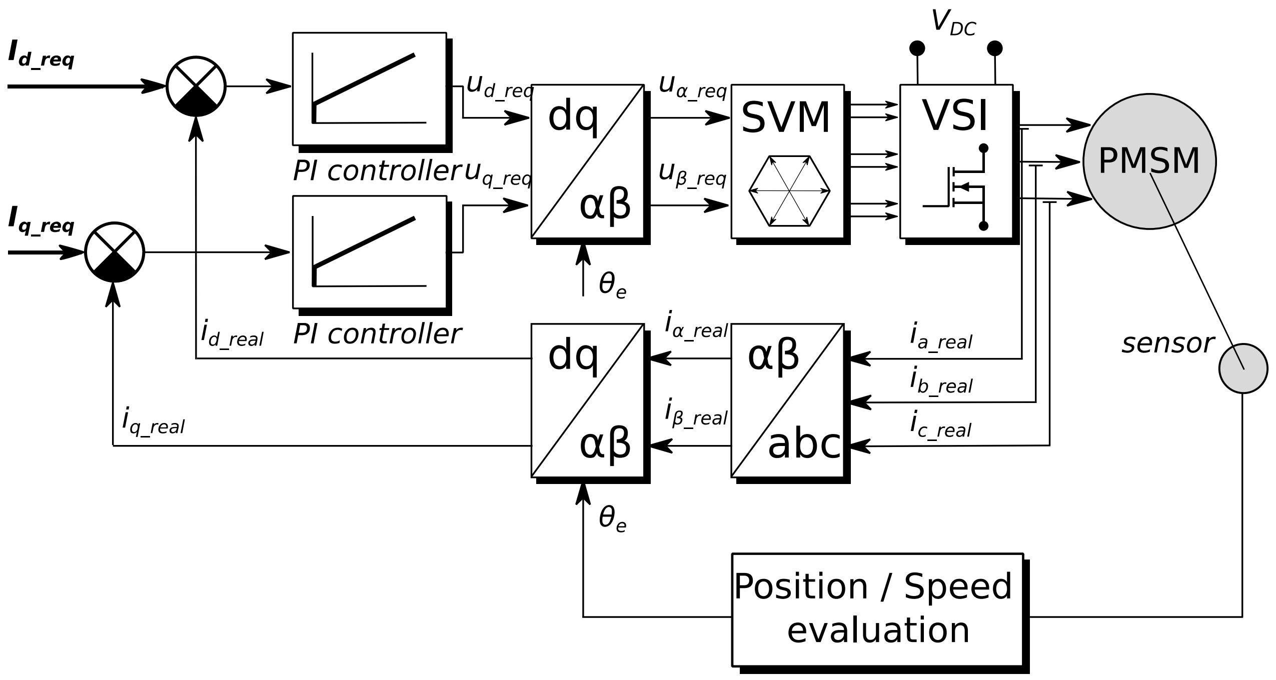


Figure 18. Current (torque) control mode

Activate the speed PMSM sensorless FOC (whose diagram is shown in the following figure) by enabling the “Speed FOC” control structure. Enter the required speed into the “Speed_req” field. The *d*-axis current reference is kept at 0 during the entire FOC operation. This control scheme is used for the speed PI controller design, which is the final stage of the PMSM sensorless application tuning.

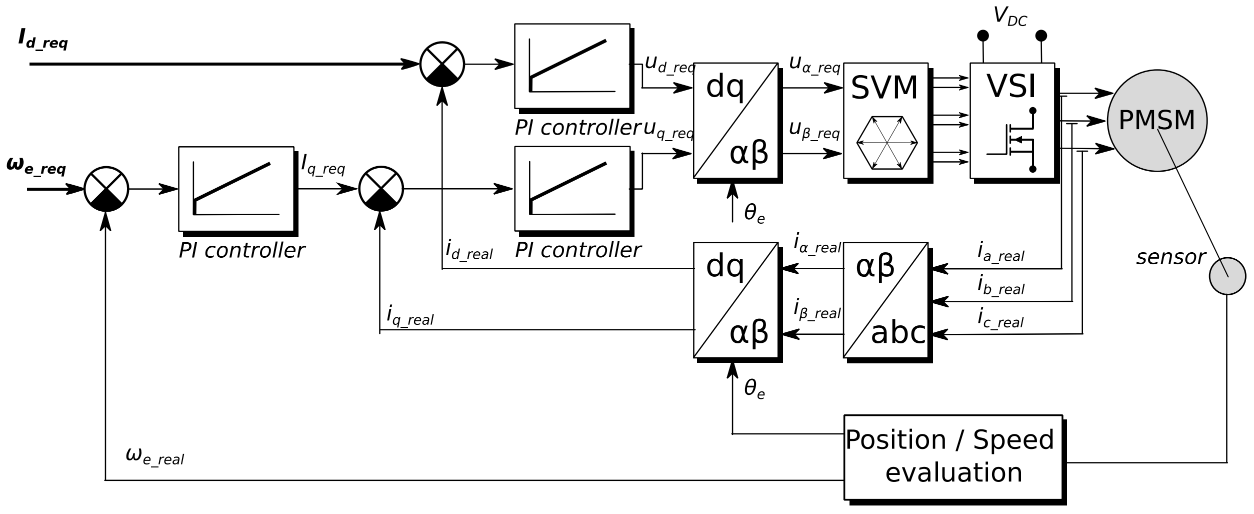


Figure 19. Speed FOC control mode

5.2.2. PMSM sensorless application tuning using MCAT

This section provides a guide for running your motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any issues during the tuning process. The state diagram in the following figure shows a typical PMSM sensorless control tuning process. The tuning phases are described in the following sections.

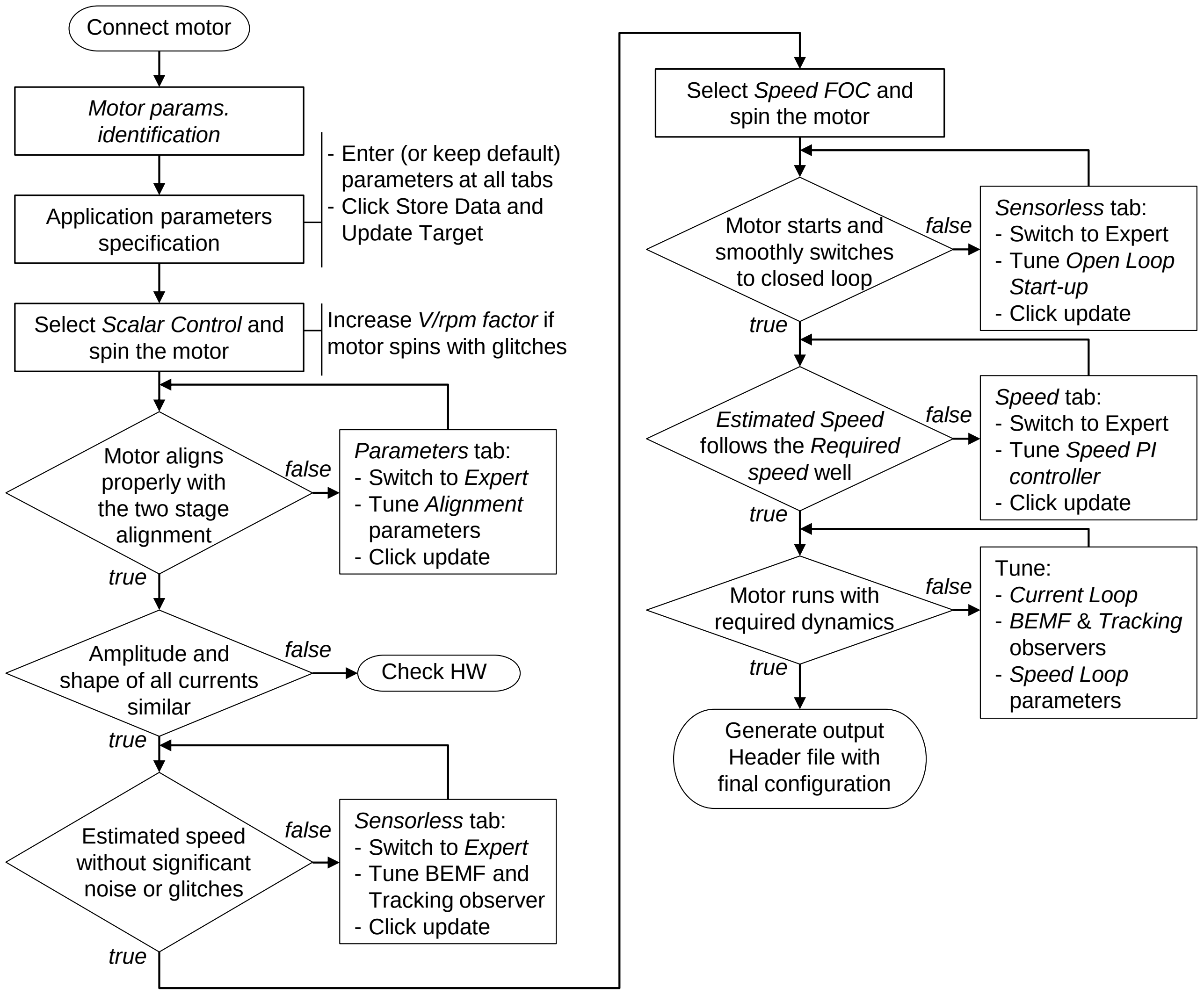

Connect motor
Motor params. identification
Application parameters specification
- Enter (or keep default) parameters at all tabs
- Click Store Data and Update Target
Select Scalar Control and spin the motor
Increase V/rpm factor if motor spins with glitches
Motor aligns properly with the two stage alignment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

Figure 20. Running a new PMSM

5.2.3. Initial configuration setting and update

1. Open the PMSM sensorless control application FreeMASTER project containing the dedicated MCAT plug-in module.
2. Select the “Basic” mode—recommended for users who are not experienced in motor-control theory. The number of required input parameters is reduced.
3. Select the “Parameters” tab.
4. Leave the measured motor parameters as they are, or specify the parameters manually. Obtain the motor parameters from the motor data sheet or using the PMSM parameters measurement procedure described in *PMSM electrical parameters measurement* (document AN4680).
   All parameters provided in the following table are accessible in both the Basic and the Expert modes. The motor inertia *J* expresses the overall system inertia that is very often difficult to obtain. Obtain the additional methods to identify the drive inertia from other resources, for example, from IEEE. The *J* parameter is used to calculate the speed controller constant. You can also use manual controller tuning to calculate this constant.

Table 12. MCAT motor parameters

Parameter	Units	Description	Typical range
pp	—	Motor pole-pairs	1–10
Rs	[Ω]	One-phase stator resistance	0.3–50
Ld	[H]	One-phase direct inductance	0.00001–0.1
Lq	[H]	One-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—modifying these two fields is not required when using a reference to the standard power stage board. These scales specify the maximum measurable current and voltage analog quantities.
6. Check the fault limits—these fields are not accessible in the “Basic” mode and they are calculated using the motor parameters and hardware scales. See this table:

Table 13. Fault limits

Parameter	Units	Description	Typical range
U DCB trip	[V]	Voltage value when the external braking resistor switch is turned on	U DCB Over~U DCB max
U DCB under	[V]	Trigger value when the under-voltage fault is detected	0~U DCB Over
U DCB over	[V]	Trigger value when the over-voltage fault is detected	U DCB Under~U max
N over	[rpm]	Trigger value when the over-speed fault is detected	N nom~N max
N min	[rpm]	Minimal actual speed value for the sensorless control	(0.05~0.2) * N max
E block	[V]	Bemf voltage threshold for blocked rotor detection. If the Bemf voltage drops down under this threshold, the blocked rotor fault sets on (see Figure 19).	0.1 * E max

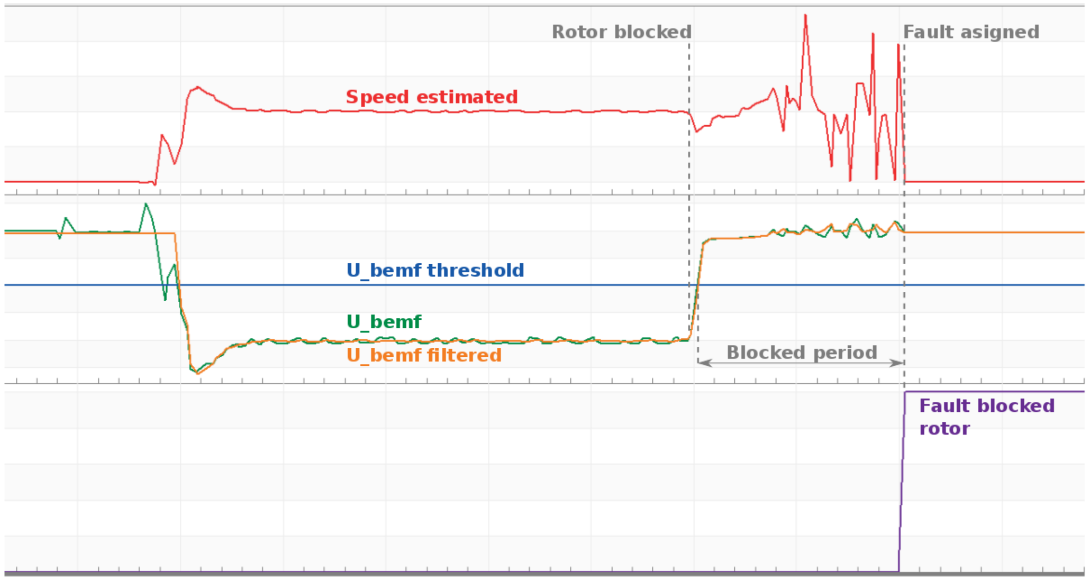


Figure 21. Blocked rotor detection

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 14. Application scales

Parameter	Units	Description	Typical range
N max	[rpm]	Speed scale	>1.1 * N nom
E max	[V]	BEMF scale	Ke * N max
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 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 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 speed value in the "Speed_req" field (e.g., 500 rpm in the "Scalar Control" section). The motor starts running (see the following figure).

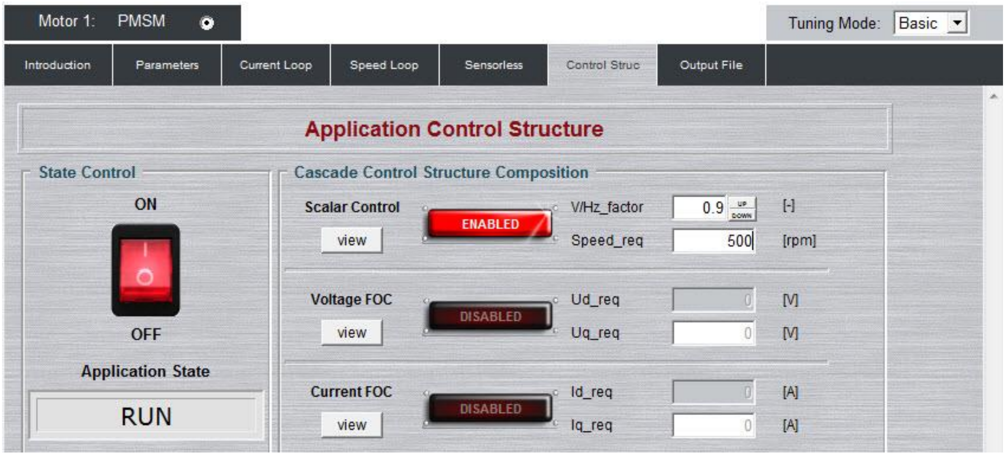


Figure 22. MCAT scalar control

4. Select the "Phase Currents" recorder from FreeMASTER project tree "Scalar and Voltage Control".
5. Find the optimal ratio for the V/Hz profile by changing the V/Hz factor directly or using the UP/DOWN buttons. The shape of the motor currents must be close to a sinusoidal shape:

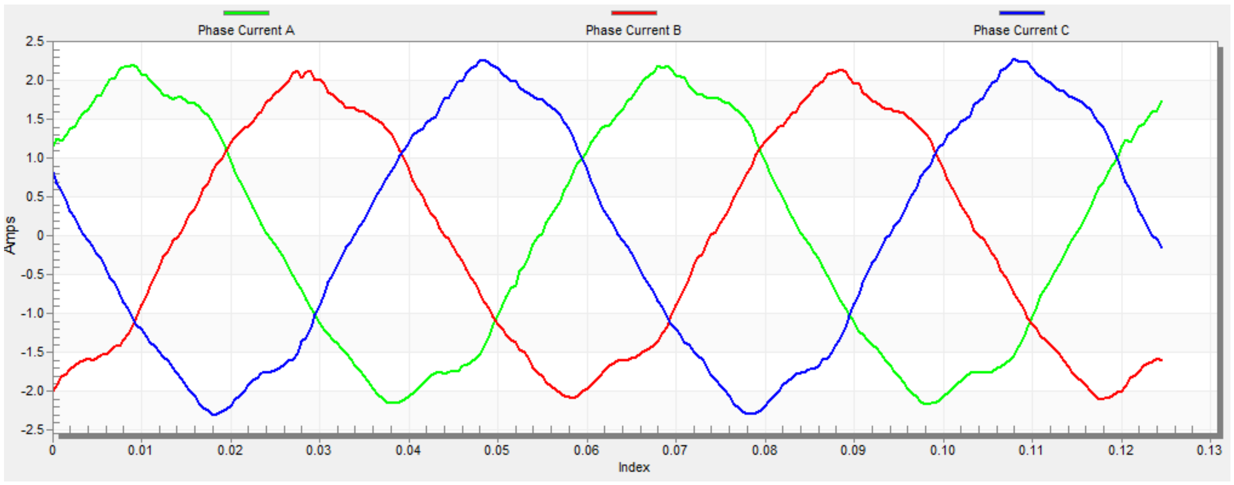


Figure 23. Phase currents

6. Select the "Position" recorder to check the observer functionality. The difference between the "Position Electrical Scalar" and the "Position Estimated" must be minimal (see the following figure) 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).

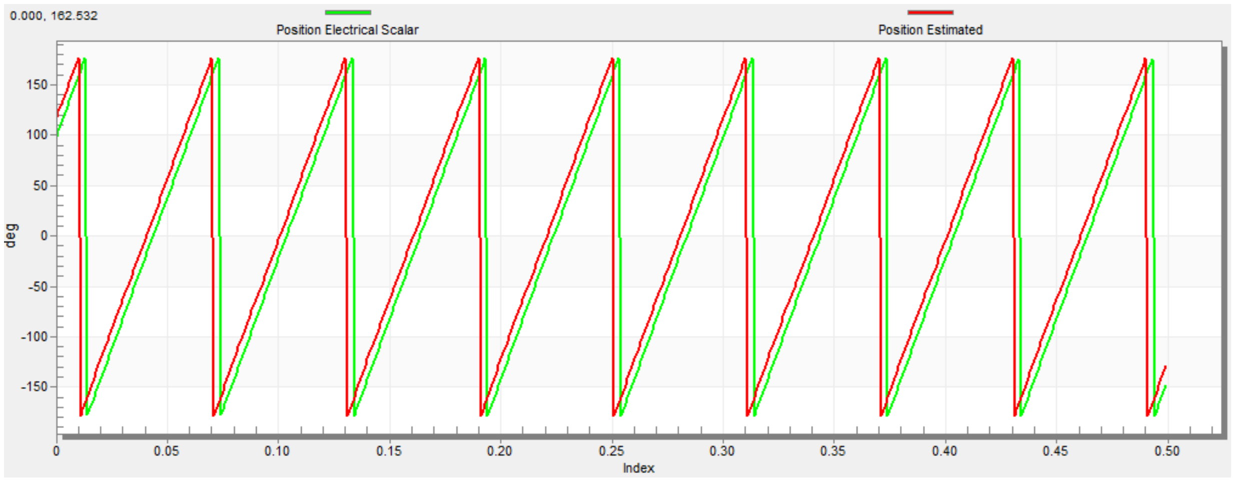


Figure 24. Generated and estimated positions

7. If an opposite speed direction is required, set a negative value in the "Speed_req" field.
8. A proper observer functionality and 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. Turn the main application switch on and put a non-zero value into the "Uq_req" field. The FOC algorithm uses the estimated position to run the motor.

5.2.5. Alignment tuning

The alignment procedure sets the rotor to an accurate initial position and enables applying 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 others). The aim of the alignment is to have the rotor in a stable position (without oscillations) before the startup.

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

5.2.6. Current loop tuning

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, tune the bandwidth and attenuation parameters.

1. Switch the tuning mode to “Expert”.
2. Set the required loop bandwidth and attenuation and click the “Update Target” button in the “Current Loop” tab. The tuning loop bandwidth parameter defines the speed of the loop response, whilst the tuning loop attenuation parameter defines the actual quantity-overshoot magnitude.
3. Select the “Current Controller Id” recorder.
4. Select the “Control Structure” tab, switch to the “Current FOC”, set the “Iq_req” to a very low value (e.g., 0.01), and set the required step to “Id_req”. The control-loop response is shown in the recorder (see Figure 6).
5. 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 “Id” current is long:

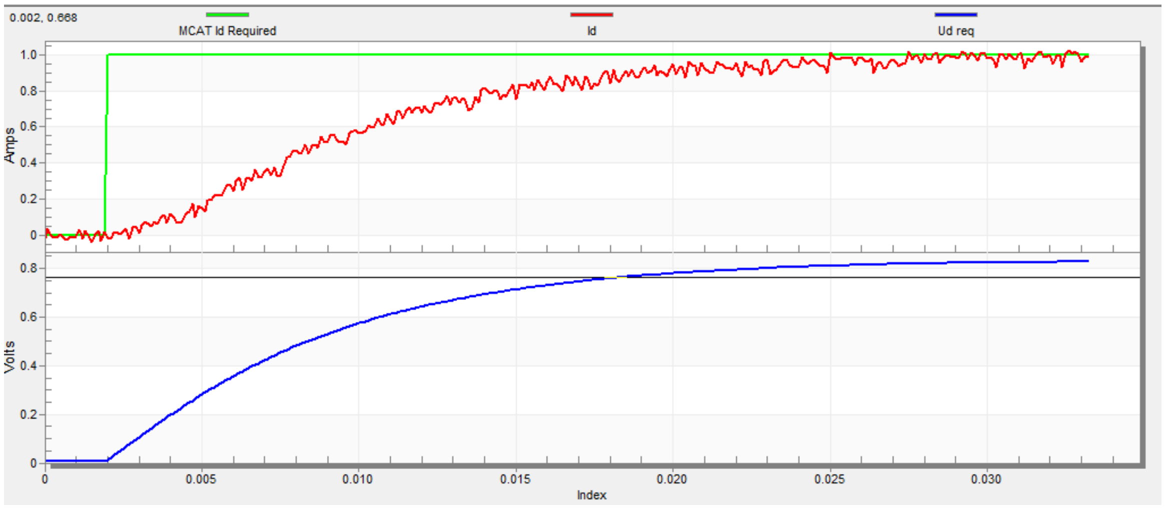


Figure 25. Slow step response of Id current controller

– The loop bandwidth (400 Hz) is optimal and the response time of the “Id” current is sufficient:

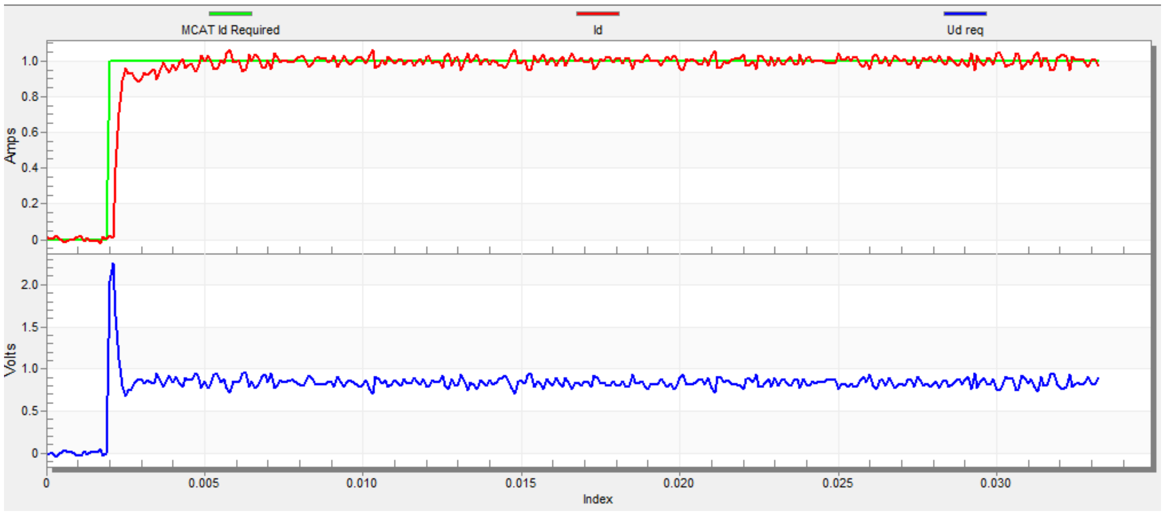


Figure 26. Optimal step response of Id current controller

– The loop bandwidth is high (700 Hz) and the response time of the “Id” current is very fast, but it contains oscillation and overshoot:

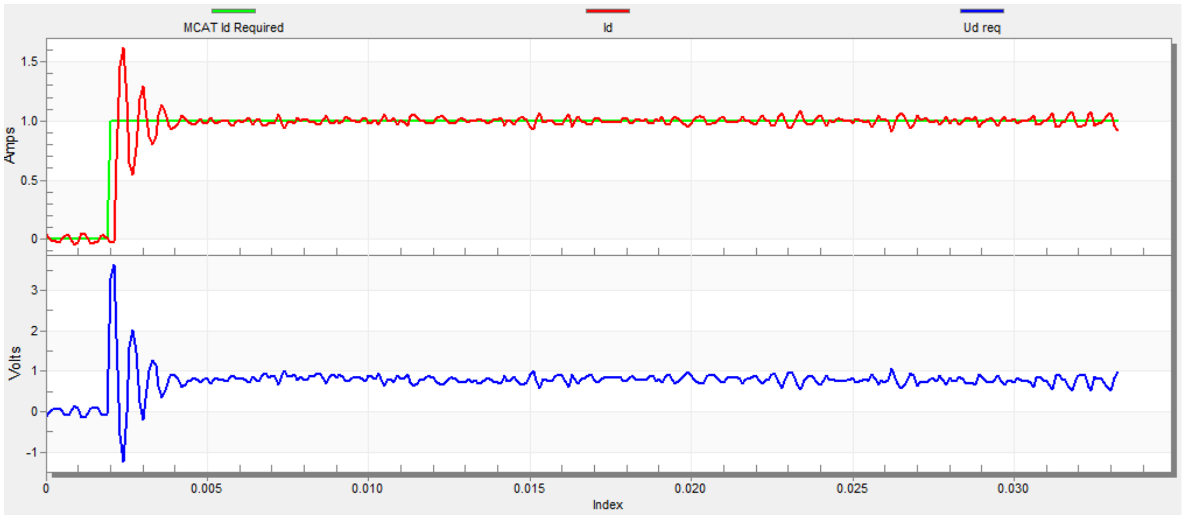


Figure 27. Fast step response of Id current controller

5.2.7. Actual speed filter

The estimated speed from the BEMF observer is fed into the speed PI controller through the IIR filter. Modify the filter cut-off frequency in the Expert mode in the “Speed Loop” tab. The speed loop sample time is typically several milliseconds, so the actual speed filter cut-off frequency mostly ranges from 5 Hz to 100 Hz.

Track the filter output in the “Speed” scope. Write the modified filter cut-off frequency value to the MCU by clicking the “Update Target” button.

5.2.8. Speed ramp tuning

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

Increments are common for the scalar and speed control. The increment fields are located in the "Speed Loop" tab and they are accessible in both tuning modes. Clicking the "Update Target" button writes the changes to the MCU. An example speed profile is shown in the following figure. The ramp down increment is set to 500rpm/sec, while the up increment is set to 3,000 rpm/sec.

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

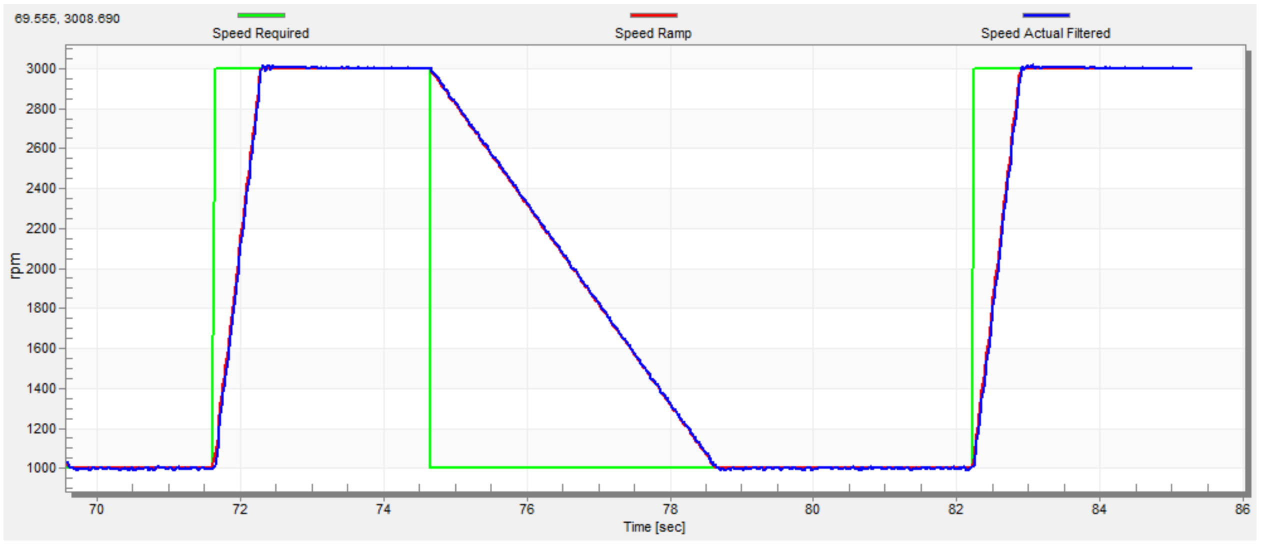


Figure 28. Speed profile

5.2.9. Open-loop startup

Tune the start-up process by a set of parameters located in the "Sensorless" tab. You can access two of them (ramp increment and current) in both tuning modes. The start-up tuning can be processed in all control modes, except for the scalar control. Set the optimal values to achieve a proper motor startup. An example start-up state of low-dynamic drives (fans, pumps) is shown in Figure 29.

1. Select the "Startup" recorder from the FreeMASTER project tree.
2. Set the start-up ramp increment 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 you can set it to 15 % of the nominal current.
4. Set the required merging speed—the threshold when the open-loop and estimated-position merging starts, mostly set in the range of 5 %~10 % of the nominal speed.
5. Set the merging coefficient—the position-merging process duration, where 100 % corresponds to a half of the electrical revolution. The higher the value is, the faster the merge is done. Values

close to 1 % are set for drives where a high start-up torque and a smooth transition between the open loop and the closed loop are required.

6. Click the “Update Target” button to write the changes to the MCU.
7. Switch to the “Control Structure” tab, and enable “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 run, 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 %.

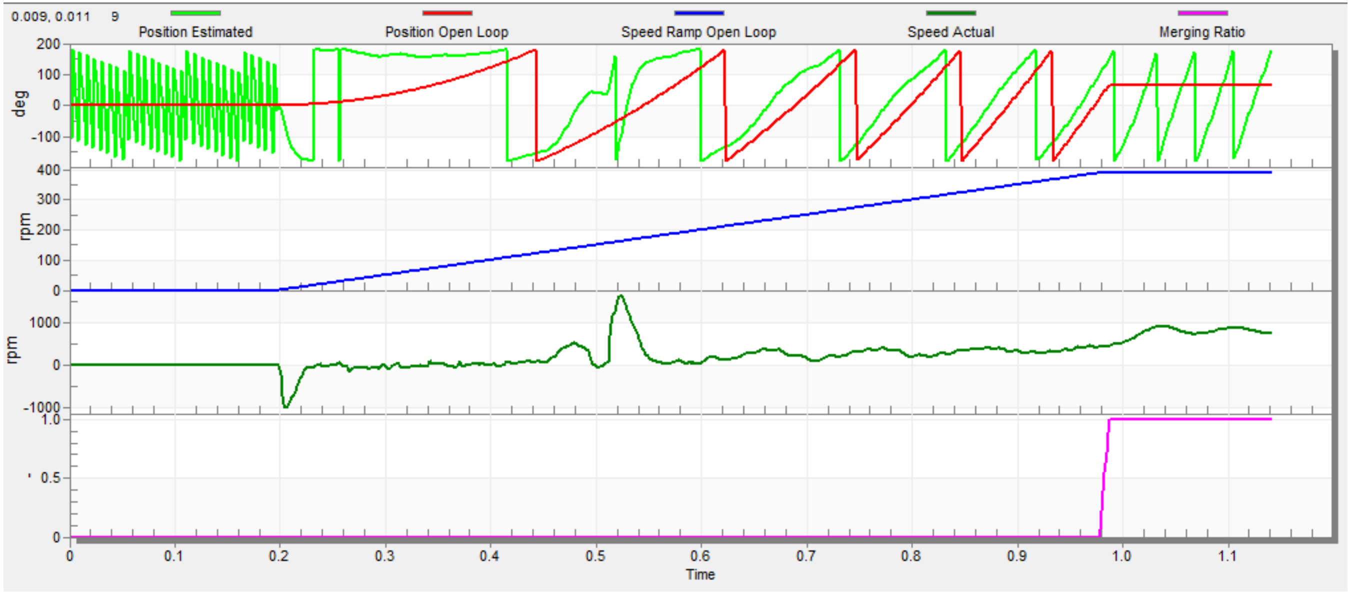


Figure 29. Motor startup

5.2.10. BEMF observer tuning

The BEMF observer and tracking observer parameters are fully calculated in the Basic mode using the motor parameters and no action is required in this mode. If the calculated loop parameters do not correspond to the optimal response, tune the bandwidth and attenuation parameters.

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 from 10 Hz to 20 Hz for most low-dynamic drives (fans, pumps).
5. Click the “Update Target” button to write 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. If these mechanical constants are available, tune the PI controller constants using the loop bandwidth and attenuation. The values of the motor and load inertias and frictions are very often unknown and it is quite difficult to obtain them. Therefore, manual tuning of the P and I portions of the speed controllers is available to obtain the required speed response (see the example response in Figure 10). There are dozens of approaches for tuning the PI controller constants. These steps provide an example of setting and tuning the speed PI controller for a PM synchronous motor.

1. Select “Speed Controller” from the FreeMASTER project tree.
2. Select the “Speed loop” tab.
3. Check “Manual Constant Tuning”—that is, the bandwidth and attenuation fields are disabled, and “SL_Kp” and “SL_Ki” are enabled.
4. Tune the proportional gain:
   - Set the “SL_Ki” integral gain to zero.
   - Set the speed ramp to 1000 rpm/sec (or higher).
   - Switch to the “Control Structure” tab and run the motor at a convenient speed (about 30 % of the nominal speed).
   - Set the step in the required speed to 40 % of *N nom*.
   - Switch back to the “Speed Loop” tab.
   - Keep tuning the proportional gain “SL_Kp” until the system responds properly to the required value (without oscillations or excessive overshoot):
     - If “SL_Kp” is set low, then the system response is slow.
     - If the “SL_Kp” is set high, then the system response is tighter.
     - If the “SL_Ki” is set to zero, then the system may not achieve the required speed.
   - Click the “Update Target” button to write the changes to the MCU.
5. Tune the integral gain:
   - Increase the “SL_Ki” slowly to minimize the difference between the required and actual speeds to zero.
   - Adjust the “SL_Ki” so that you do not see any oscillation or large overshoot of the actual speed value while applying the required speed step.
   - Click the “Update Target” button to write the changes to the MCU.

6. Keep tuning the loop bandwidth and attenuation until you achieve the required response. The waveform examples with correct and incorrect settings of the current 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”:

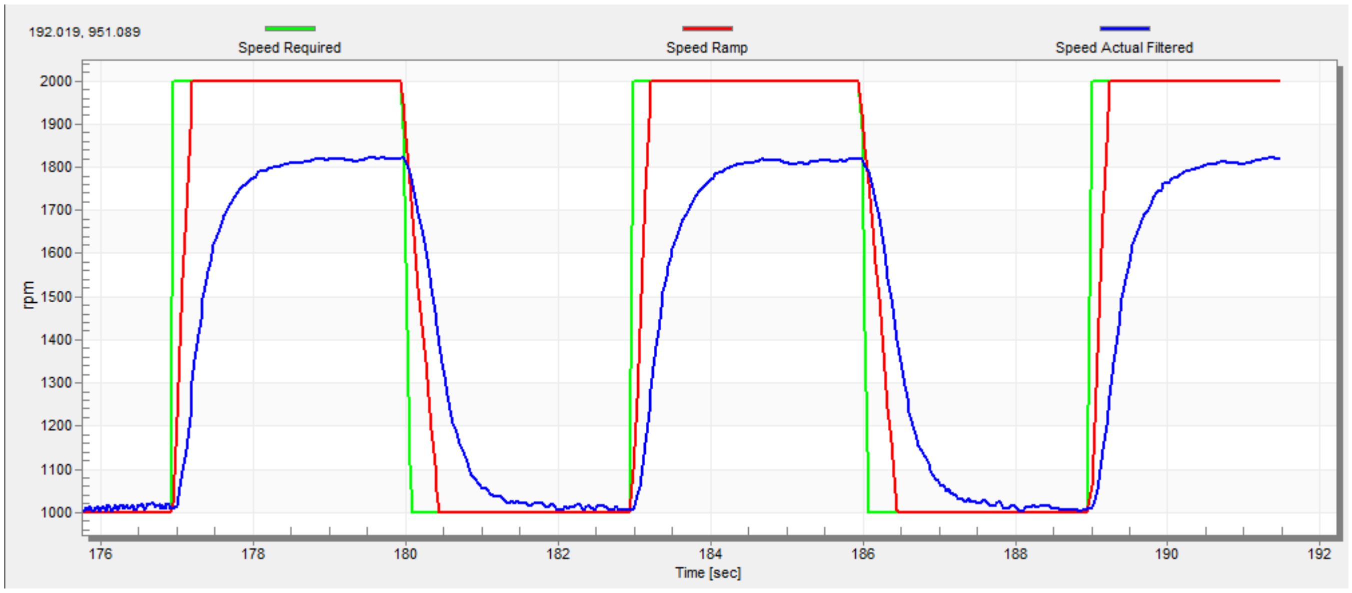


Figure 30. Speed controller response—“SL_Ki” is low, “Speed Ramp” not achieved

   - The “SL_Kp” value is low, “Speed Actual Filtered” greatly overshoots, and the long settling time is not wanted:

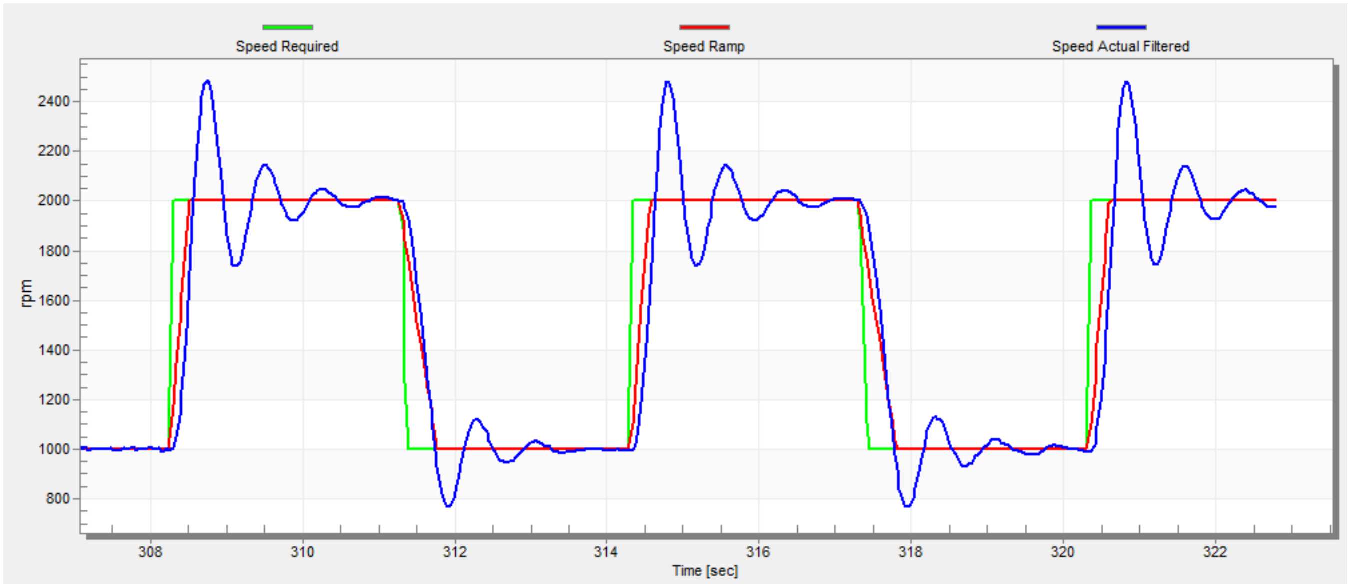


Figure 31. Speed controller response—“SL_Kp” is low, “Speed Actual Filtered” greatly overshoots

– Speed loop response has a small overshoot, and the “Speed Actual Filtered” settling time is sufficient. Such response is considered optimal:

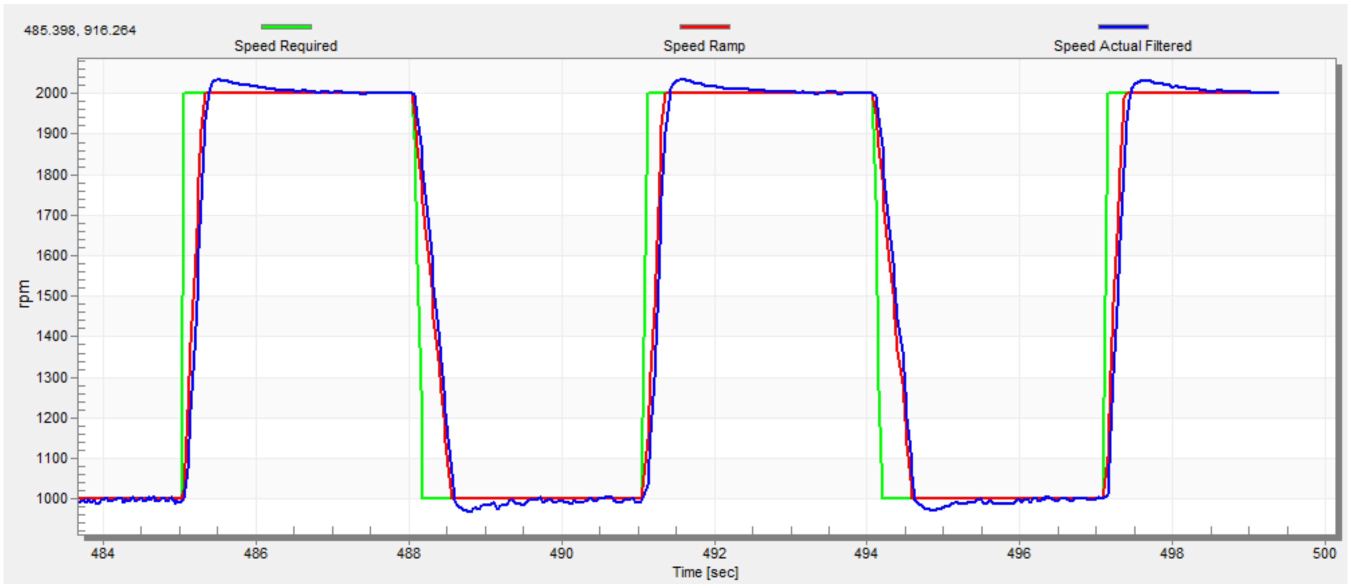


Figure 32. Speed controller response—speed loop response with small overshoot

5.2.12. Generating MCAT output file

When you successfully finish tuning the application and want to store all calculated parameters to the embedded application, navigate to the “Output File” tab. View the list of all definitions generated by MCAT. Clicking the “Generate Configuration File” button overwrites the old version of the *m1_pmsm_appconfig.h* file, which contains these definitions. Provide a correct path to the file for a proper generation of the motor parameter file. To change the path, navigate the cursor to the right corner of the MCAT screen, and a symbol with a screw driver and a wrench appears. When clicking this symbol, the “Application Settings Page” appears. Modify the path to the *m1_pmsm_appconfig.h* file in the “Project Path Selection” area.

6. Conclusion

This application note describes the implementation of a sensorless Field-Oriented Control of the 3-phase PMSM using 32-bit Kinetis V series devices and the High-Voltage Platform, Tower System, and Freedom development platforms. The hardware-dependent part of the sensorless control software (including a detailed peripheral setup), Motor Control Peripheral Drivers (MCDRV), and application timing are described in Section 3, “MCU Features and Peripheral Settings”. The motor parameters identification theory and the identification algorithms are described in Section 5.1, “PMSM parameter identification”. The last part of the document describes the user interface represented by Motor Control Application Tuning (MCAT) tool, based on FreeMASTER communication interface.

7. Acronyms and Abbreviations

Table 15. Acronyms and abbreviations

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

8. References

These references are available on nxp.com:

- *Sensorless PMSM Field-Oriented Control* (document DRM148)
- *Kinetis KV11: 75 MHz Cortex-M0+ 64/128 KB Flash (32-64 pin)* (document KV11P64M75RM)
- *KV31F Sub-Family Reference Manual* (document KV31P100M120SF7RM)
- *KV4x Reference Manual* (document KV4XP100M168RM)
- *KV5x Sub-Family Reference Manual* (document KV5XP144M220RM)
- *NXP High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG)
- *HVP-KV31F120M User's Guide* (document HVPKV31F120MUG)
- *Using FlexTimer in ACIM/PMSM Motor Control Applications* (document AN3729)
- *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document AN4822)
- *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642)

9. Revision History

This table summarizes the changes done to this document since the initial release:

Table 16. Revision history

Revision number	Date	Substantive changes
0	02/2016	Initial release.
1	06/2016	Updated FSLESL 4.1 to RTCESL 4.3. Added blocked rotor threshold detection. Bug fixes.
2	09/2016	Added KE1xZ and KE1xF MCUs.

Document Number: AN5237
Rev. 2
09/2016