NXP Semiconductors
User's Guide

Document identifier: 3PPMSMCKVUG
Rev. 0, 02/2020

MCUXpresso SDK 3-Phase PMSM Control (KV)

Contents

Chapter 1
Introduction

This user's guide describes the implementation of the sensorless motor-control software for a 3-phase Permanent Magnet Synchronous Motor (PMSM), including the motor parameters identification algorithm, on NXP 32-bit Kinetis V series MCUs. The sensorless control software itself and the PMSM control theory in general are described in DRM148: Sensorless PMSM Field-Oriented Control. The Freedom (FRDM-MC-LVPMSM), Tower (TWR-MC-LV3PH), and High Voltage (HVP-MC3PH) power stages are used as hardware platforms for the PMSM control reference solution. The hardware-dependent part of the sensorless control software, including a detailed peripheral setup and the Motor Control (MC) peripheral drivers, is described as well. The motor parameters identification theory and algorithms are described in this document. The last part of the document introduces and explains the user interface represented by the Motor Control Application Tuning (MCAT) page based on the FreeMASTER run-time debugging tool. These tools present simple and user-friendly way for motor parameter identification, algorithm tuning, software control, debugging, and diagnostics.

Chapter 2
Supported development boards for KV

There are development boards for the Kinetis KV and KE MCUs for motor-control applications. The development boards and supported MCUs are shown in Table 1. The Tower System modular development platform (TWR) and the Freedom development platform (FRDM) are targeted for low-voltage and low-power applications with the PMSM control type. The High-Voltage Platform (HVP) is designed to drive high-voltage (115/220 V) applications with up to 1 kW of power.

Table 1. Supported development boards

	Platform		
	FRDM	**TWR**	**HVP**
MCU / Power Stage	**FRDM-MC-LVPMSM**	**TWR-LV3PH**	**HVP-MC3PH**
KV11	FRDM-KV11Z	TWR-KV11Z75M	-
KV31	FRDM-KV31F	TWR-KV31F120	HVP-KV31F512
KV46	-	TWR-KV46F150M	HVP-KV46F150
KV58	-	TWR-KV58F220M	HVP-KV58F220M
KE15Z	FRDM-KE15Z	-	-
KE18F	-	TWR-KE18F	HVP-KE18F

Chapter 3
Motor control vs. SDK peripheral drivers

The motor-control examples use the MCUXpresso SDK peripheral drivers to configure the general peripherals (such as clocks, SPI, SIM, and ports). However, motor control requires critical application timing because most control algorithms run in a 100-us loop. To optimize the CPU load, most peripheral hardware features are implemented for the PWM signal generation, analog signal sampling, and synchronization between the PWM and ADC units.

The standard SDK peripheral drivers do not support the configuration and handling of all required features. The motor-control drivers are designed to configure the critical MC peripherals (eflexPWM, FTM, ADC, and PDB).

It is highly recommended not to modify the default configuration of the allocated MC peripherals due to a possible application-timing conflict. The particular *mcdrv_< board&MCU >.c* source file contains configuration functions of allocated peripherals.

Chapter 4
Hardware setup

The PMSM sensorless application runs on Tower, Freedom, or EVK development platforms with the 24-V Linix Motor in the default configuration. The HVP platform runs with the default configuration for the MIGE 60CST-MO1330 motor.

4.1 Linix 45ZWN24-40 motor

The Linix 45ZWN24-40 motor is a low-voltage 3-phase permanent-magnet motor with hall sensor used in PMSM applications. The motor parameters are listed in Table below.

Table 2. Linix 45ZWN24-40 motor parameters

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

Figure 1. Linix 45ZWN24-40 permanent magnet synchronous motor

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

4.2 MIGE 60CST-MO1330 motor

The MIGE 60CST-MO1330 motor (described in Table 3) is used by the PMSM sensorless application. You can also adapt the application to other motors, just by defining and changing the motor-related parameters. The motor is connected directly to the high-voltage development board via a flexible cable connected to the three-wire development board connector.

Table 3. MIGE 60CST-MO1330 motor parameters

Characteristic	Symbol	Value	Units
Rated Voltage	Vt	220	V
Rated speed	-	3000	RPM
Rated power	P	400	W
Number of pole-pairs	pp	4	-

Figure 2. MIGE 60CST-MO1330 motor

4.3 Running PMSM application on Tower System

To run the PMSM application on the Tower System, you need the following Tower modules:

- Tower board with a Kinetis V or E series MCU (TWR-KV11Z75M, TWR-KV31F120M, TWR-KV46F150M, TWR-KV58F220M, or TWR-KE18F).
- 3-phase low-voltage power module (TWR-MC-LV3PH) including the Linix motor.
- Tower elevator modules (TWR-ELEV).

You can order all Tower modules from www.nxp.com or from distributors to easily build the hardware platform for the target application.

4.3.1 TWR-MC-LV3PH Tower System module

This evaluation board effectively turns a Tower System development board into a complete motor-control reference design, compatible with the existing Tower System Kinetis V development boards. This board provides all necessary feedback signals for driving PMSM and BLDC motors. These signals enable a variety of algorithms to control 3-phase PMSM and BLDC motors. A high level of board protection (over-current, under-voltage, over-temperature) is provided by the MC33937 pre-driver.

The TWR-MC-LV3PH has the power supply input voltage from 12 to 24 VDC, extended up to 50 VDC with the reverse polarity protection circuitry. An auxiliary power supply with 5 and 3.3 VDC is created for Tower System MCU boards. The output current is up to 5 A RMS. The inverter itself is realized by the 3-phase bridge inverter (6-MOSFET's) and the 3-phase MOSFET gate driver. Analog quantities, such as the 3-phase motor currents, 3-phase motor back-EMF voltage, DC bus voltage, and DC bus current are sensed on this board. There is an interface for speed/position sensors (Encoder Hall) and a connector for a braking resistor. There are also user LED, power-on LED, and 6 PWM LED diodes for diagnostics. The block diagram of a complete Tower System motor control development kit is shown below.

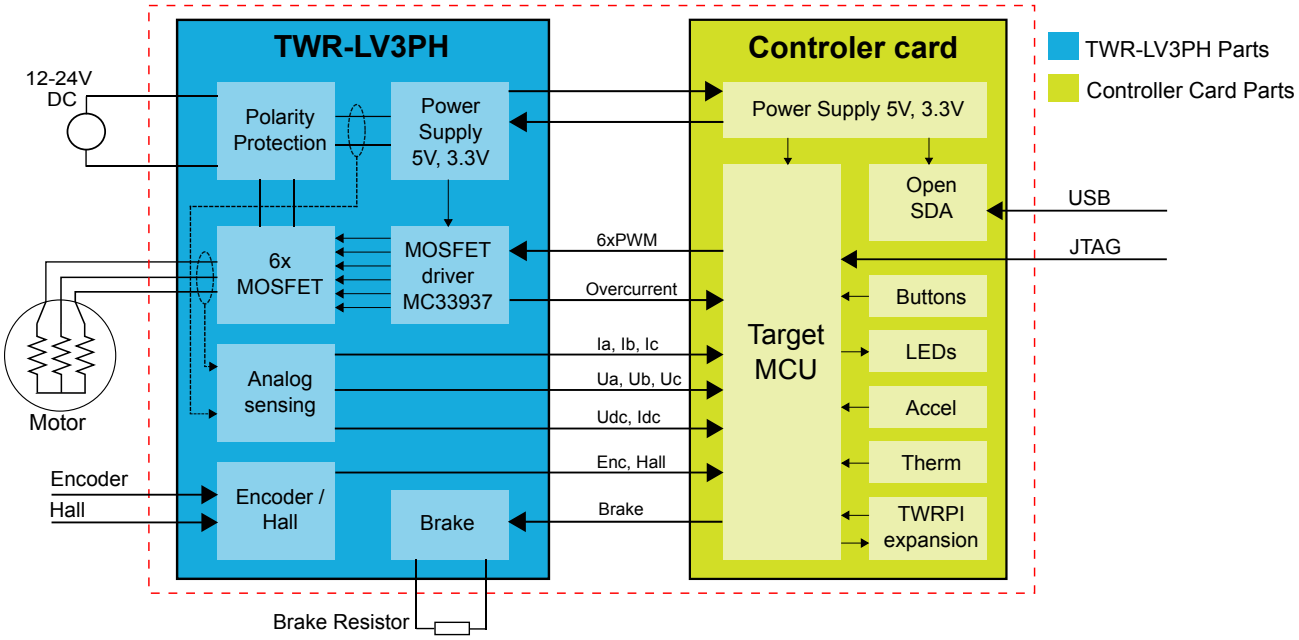

Figure 3. Tower motor control development platform block diagram

The TWR-MC-LV3PH does not require any complicated setup. Just keep in mind that the side with the white stripe must be connected to the primary (white) elevator and, before plugging the TWR-MC-LV3PH module into the Tower System development platform, ensure that the jumpers on your TWR-MC-LV3PH module are configured as follows:

Table 4. TWR-MC-LV3PH jumper settings

Jumper	Settings	Function
J2	1-2	Selects internal analog power supply.
J3	1-2	Selects internal analog power reference (GND).
J10	2-3	Selects I_SENSE_C.
J11	2-3	Selects I_SENSE_B.
J12	2-3	Selects I_SENSE_A.
J13	2-3	Selects I_SENSE_C.
J14	2-3	Selects I_SENSE_A.

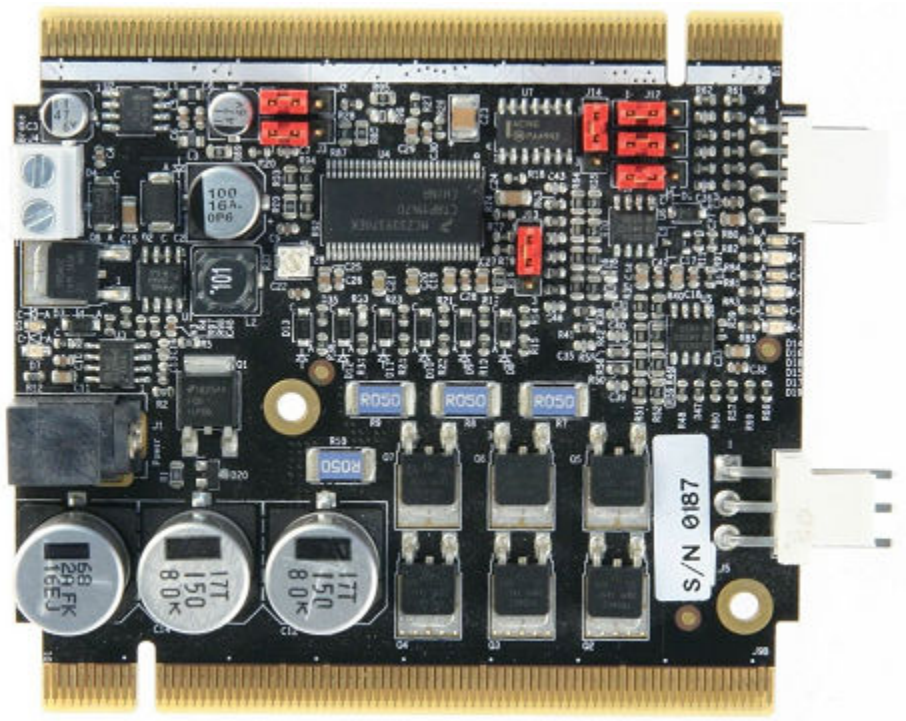

Figure 4. TWR-MC-LV3PH Tower System module

For more information about the Tower System development platform, see www.nxp.com.

4.3.2 TWR-KV11 Tower System module

The TWR-KV11Z75M is a development tool for the NXP Kinetis KV1x family of MCUs built around the Arm Cortex-M0+ core. This MCU has enough power for use in motor-control applications (such as PMSM or BLDC motors with simple or advanced control techniques). The MCU has a wide range of peripherals.

To begin, configure the jumpers on the TWR-KV11Z75M and TWR-MC-LV3PH Tower System modules properly. The following table lists the specific jumpers and their settings for the TWR-KV11Z75M Tower System module.

Table 5. TWR-KV11Z jumper settings

Jumper	Setting	Jumper	Setting	Jumper	Setting
J1	open	J9	open	J505	2-3
J2	open	J10	1-2	J506	2-3
J4	open	J11	open	J512	1-2
J5	1-2, 5-6, 7-8, 9-1	J12	1-2	J517, J518	J518-J517(2)
J6	open	J13	open	J519	1-2
J7	open	J14	1-2	J523	1-2
J8	open	J17	2-3	J524	open
-	-	-	-	J526	1-2

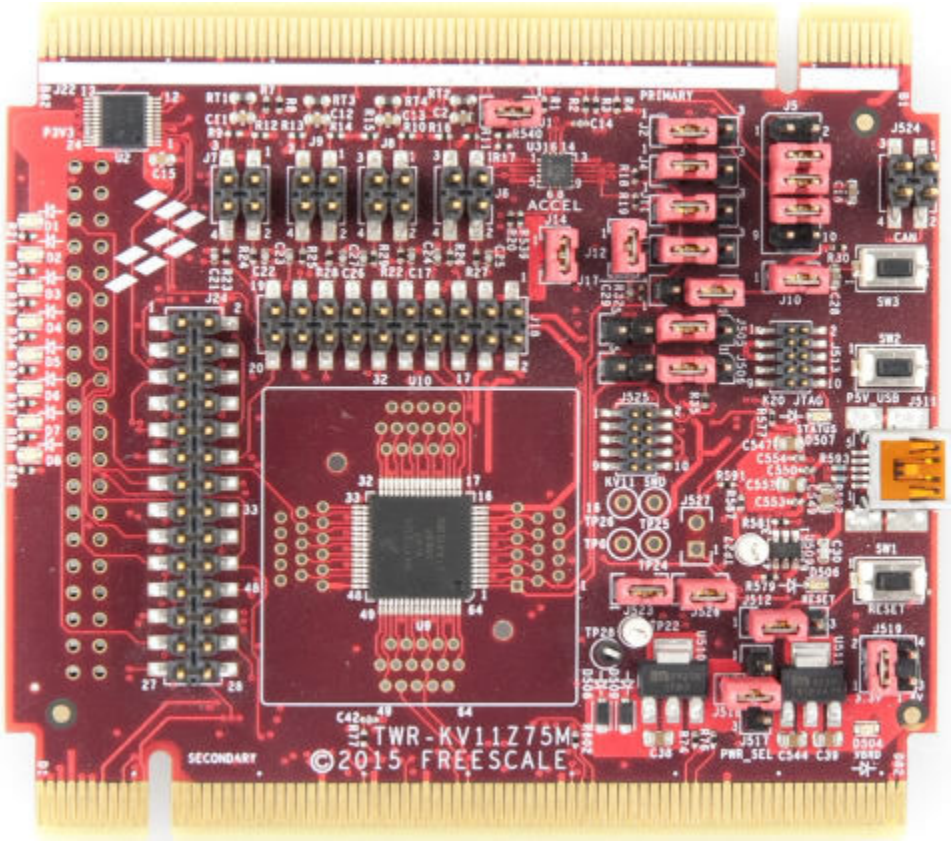

Figure 5. TWR-KV11Z Tower System module

4.3.3 TWR-KV31F Tower System module

The TWR-KV31F120M is a development tool for the NXP Kinetis KV3x family of MCUs built around the Arm Cortex-M4 core. This MCU has enough power for use in motor-control applications (such as PMSM or BLDC motors with simple or advanced control techniques). The MCU has a wide range of peripherals, lots of memory (depending on the model used), and a powerful core.

To begin, configure the jumpers on the TWR-KV31F120M and TWR-MC-LV3PH Tower System modules properly. The following table lists the specific jumpers and their settings for the TWR-KV31F120M Tower System module.

Table 6. TWR-KV31F Tower System module

Jumper	Setting	Jumper	Setting	Jumper	Setting
J1	1-2	J10	open	J17	1-2, 3-4, 5-6, 7-8
J3	1-2	J11	open	J20	open
J4	open	J12	open	J22	2-3
J5	1-3	J13	1-2	J29	2-3
J8	open	J14	1-2	J25	1-2
J9	open	J15	1-2	J26	2-3

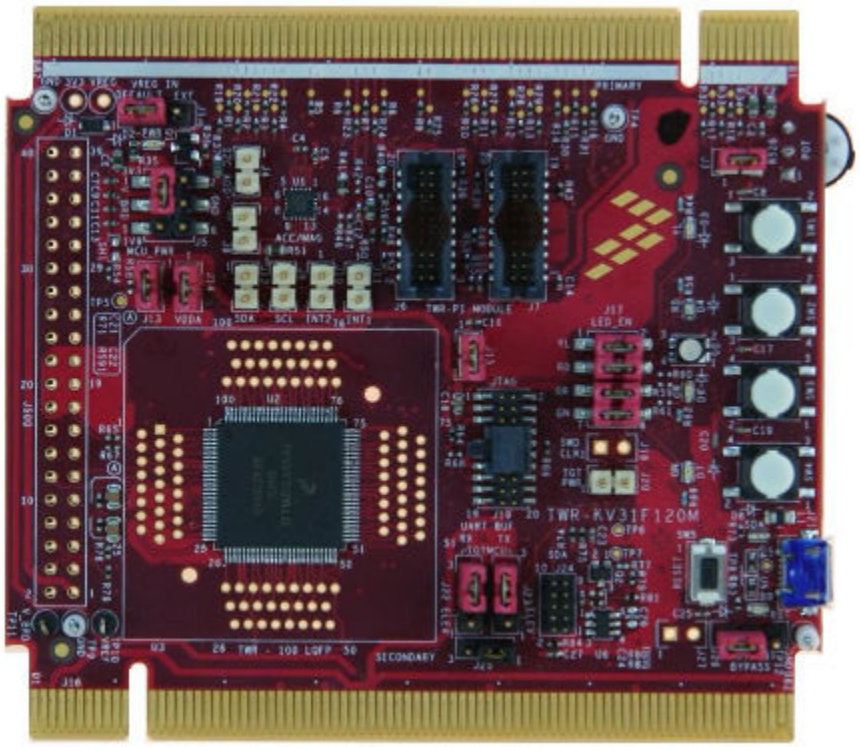

Figure 6. TWR-KV31F Tower System module

4.3.4 TWR-KV46F Tower System module

The TWR-KV46F150M is a development tool for the NXP Kinetis KV4x family of MCUs built around the Arm Cortex-M4 core. This MCU has enough power for use in motor-control applications (such as PMSM or BLDC motors with simple or advanced control techniques). The MCU has a wide range of motor-control peripherals, lots of memory (depending on the model used), and a powerful core.

To begin, configure the jumpers on the TWR-KV46F150M and TWR-MC-LV3PH Tower System modules properly. The following table lists the specific jumpers and their settings for the TWR-KV46F150M Tower System module.

Table 7. TWR-KV46F Tower System module

Jumper	Setting	Jumper	Setting	Jumper	Setting
J1	open	J16	open	J505	3-4
J2	open	J19	open	J506	3-4
J4	2-3	J20	1-2	J512	1-2
J5	open	J21	open	J514	2-3
J13	1-2, 3-4	J23	open	J517	2-3
J15	1-2	J25	1-2	J519	3-4

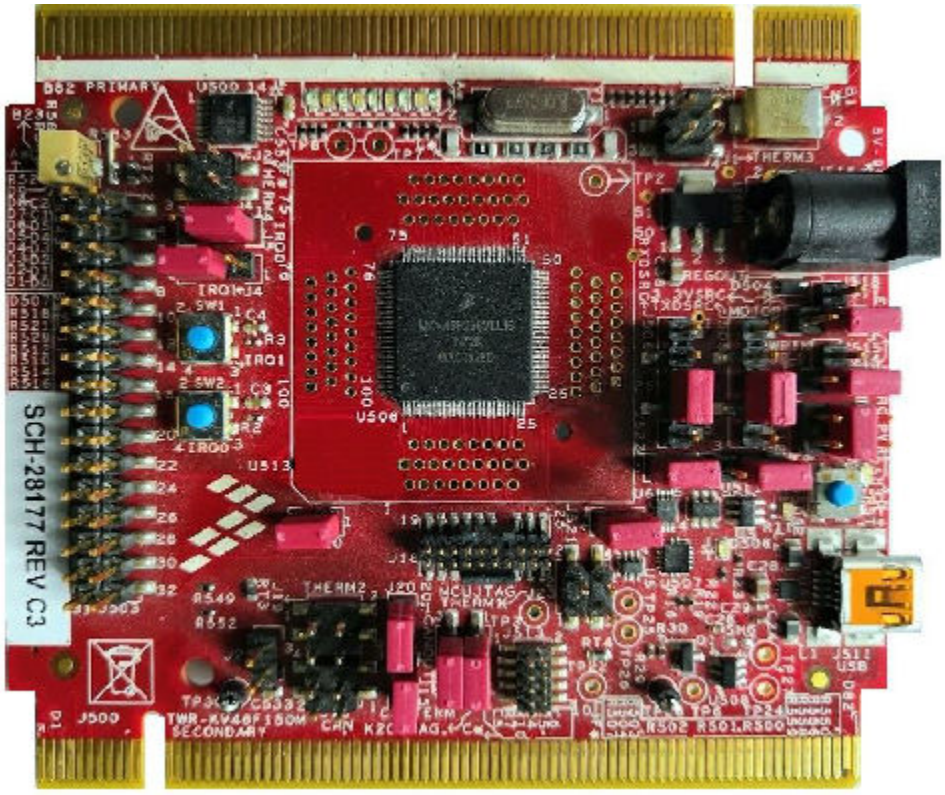

Figure 7. TWR-KV46F Tower System module

4.3.5 TWR-KV58F Tower System module

The TWR-KV58F220M is a development tool for the NXP Kinetis KV5x family of MCUs built around the Arm Cortex-M7 core. This MCU has enough power for use in multi-motor-control applications (such as PMSM or BLDC motors with simple or advanced control techniques). The MCU has a wide range of motor-control peripherals, lots of memory (depending on the model used), and a powerful core.

To begin, configure the jumpers on the TWR-KV58F220M and TWR-MC-LV3PH Tower System modules properly. The following table lists the specific jumpers and their settings for the TWR-KV58F220M Tower System module.

Table 8. TWR-KV58F Tower System module

Jumper	Setting	Jumper	Setting	Jumper	Setting
J1	1-2	J11	1-2	J23	2-3
J2	open	J12	1-2	J24	2-3
J3	1-2	J14	open	J25	2-3
J4	1-2	J17	open	J26	2-3
J7	1-2	J18	open	J28	1-2
J8	1-2	J19	open	J29	open
J9	open	J20	open	J30	1-2, 3-4, 5-6, 7-8
J10	1-2	J21	1-2	-	-

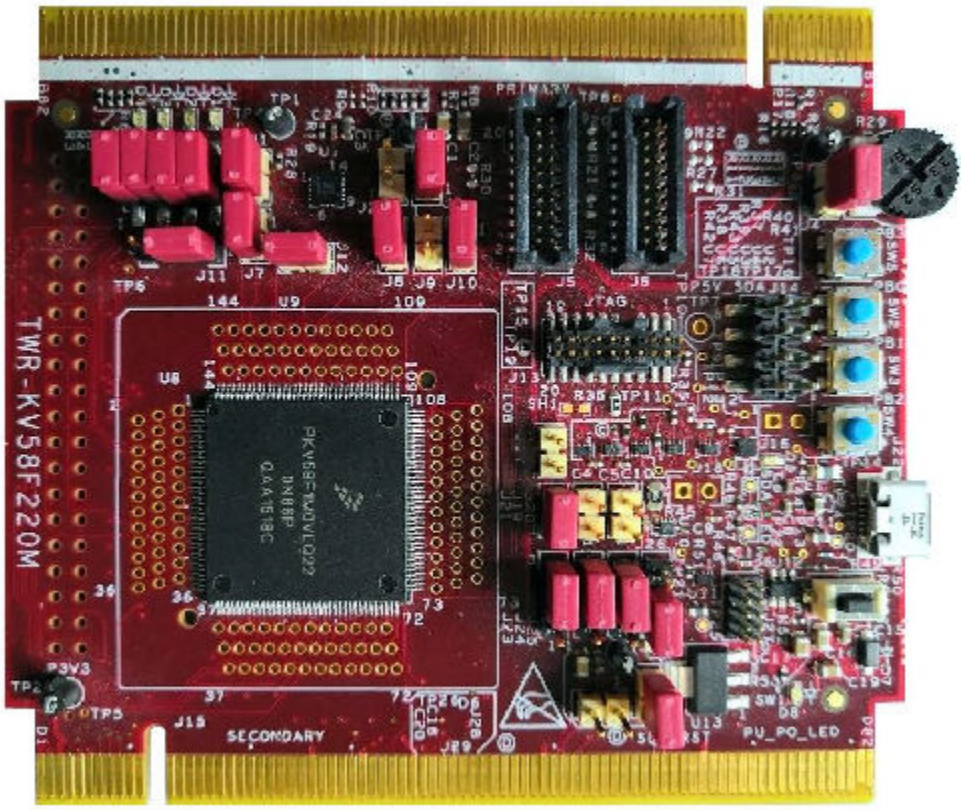

Figure 8. TWR-KV58F Tower System module

4.3.6 TWR-KE18F Tower System module

The TWR-KE18F is a development tool for the NXP Kinetis KE1xF family of MCUs built around the Arm Cortex-M4 core. This MCU has enough power for use in motor-control applications (such as PMSM or BLDC motors with simple or advanced control techniques). The MCU has a wide range of peripherals, lots of memory (depending on the model used), and a powerful core.

To begin, configure the jumpers on the TWR-KE18F and TWR-MC-LV3PH Tower System modules properly. The following table lists the specific jumpers and their settings for the TWR-KE18F Tower System module.

Table 9. TWR-KE18F Tower System module

Jumper	Setting	Jumper	Setting	Jumper	Setting
J3	2-3	J9	1-2	J18	1-2
J4	1-2	J11	1-2	J19	1-2
J5	2-3	J12	1-2	J21	open
J6	2-3	J13	1-2	J22	1-2
J7	2-3	J16	1-2	J23	open
J8	open	J17	1-2	J24	1-2

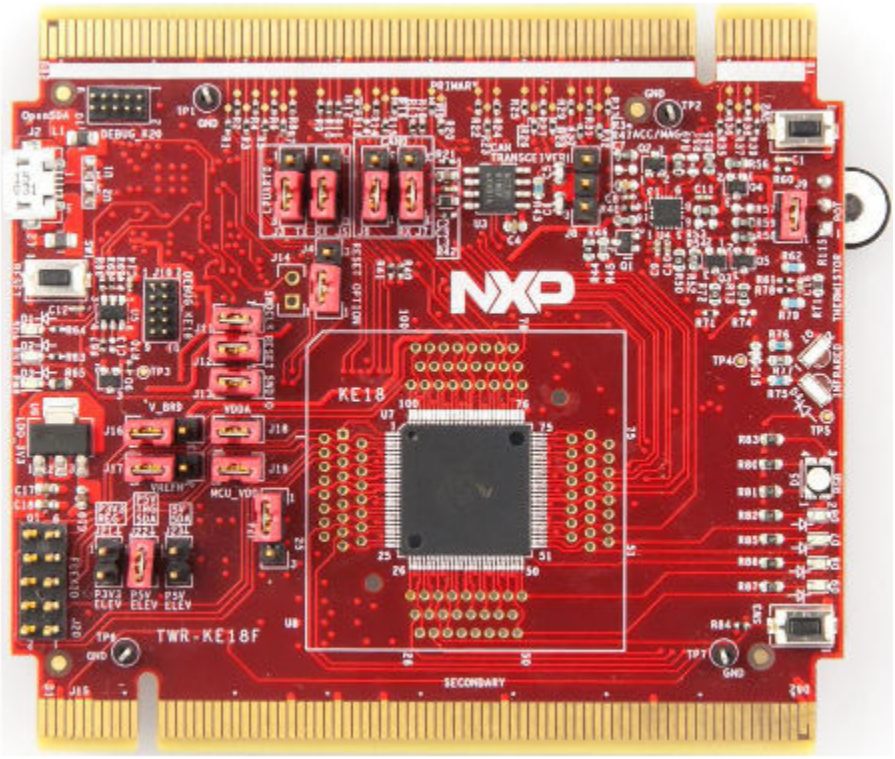

Figure 9. TWR-KE18F Tower System module

4.3.7 Tower System assembling

1. Insert the TWR-KVxxXxx MCU module and the TWR-MC-LV3PH peripheral module into the TWR-ELEV cards. Ensure that the primary sides of the modules (marked by a white stripe) are inserted into the primary elevator card (marked by white connectors).
2. After assembling the Tower System, connect the required cables as follows:
   - Connect the power input cable (3-wire connector) of the Linix motor to its corresponding connector (J5) on the TWR-MC-LV3PH motor-control driver board.
   - Plug the power supply cable attached to the TWR-MC-LV3PH system kit to the motor-control peripheral board TWR-MC-LV3PH (DC 24V).
   - Connect the TWR MCU module to any USB port on the host PC via a USB cable.

Figure 10. Assembled Tower System

4.4 Running PMSM application on Freedom development platform

To run the PMSM application using the NXP Freedom development platform, you need these Freedom boards:

- Freedom board with a Kinetis V series MCU (FRDM-KV11Z or FRDM-KV31F) or Freedom board with a Kinetis E series MCU (FRDM-KE15Z).
- 3-phase low-voltage power Freedom shield (FRDM-MC-LVPMSM) with included Linix motor.

You can order all Freedom modules from www.nxp.com or from distributors to easily build the hardware platform for the target application.

4.4.1 FRDM-MC-LVPMSM

This evaluation board, in a shield form factor, effectively turns a NXP Freedom development board into a complete motor control reference design, compatible with existing NXP Freedom development boards. Freedom motor control headers are compatible with Arduino ™ R3 pin layout.

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

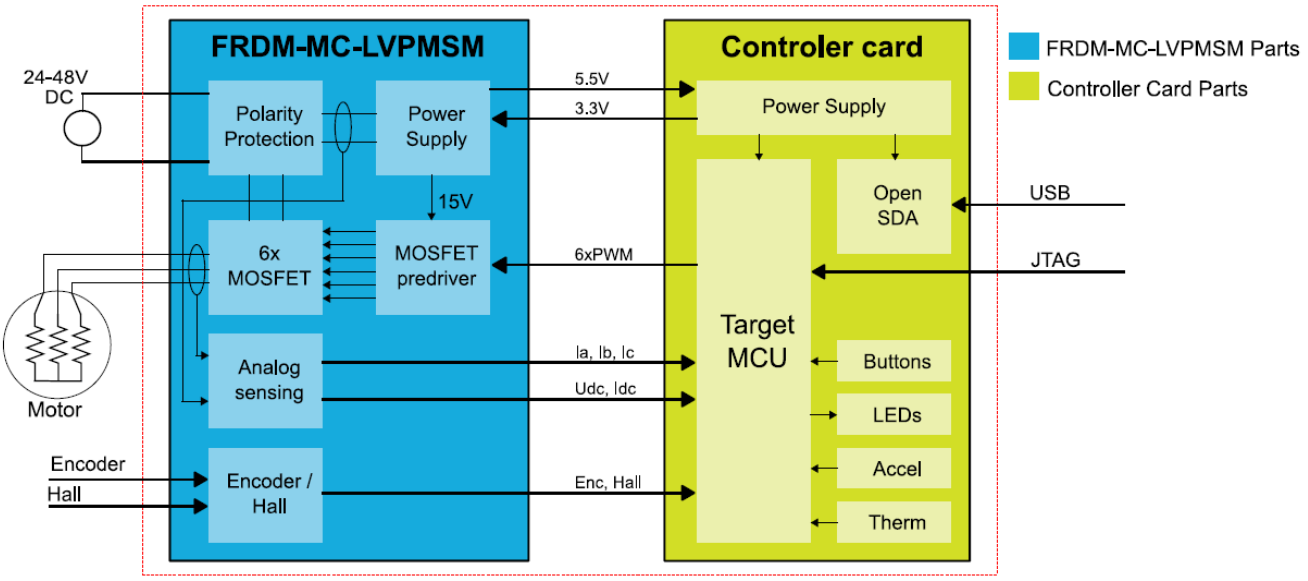

Figure 11. Motor-control development platform block diagram

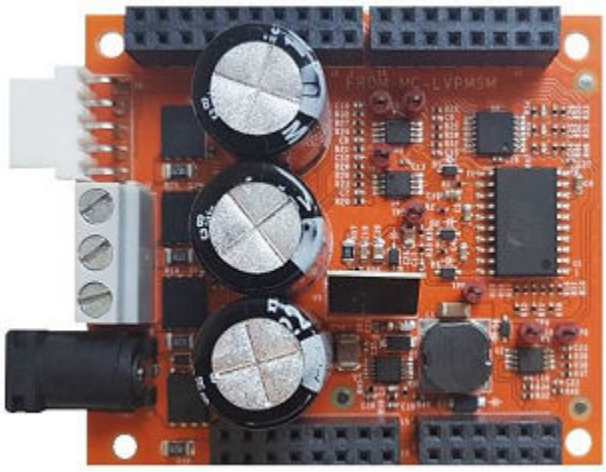

Figure 12. FRDM-MC-LVPMSM

The FRDM-MC-LVPMSM does not require any complicated setup. For more information about Freedom development platform see nxp.com.

4.4.2 FRDM-KV31F board

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

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

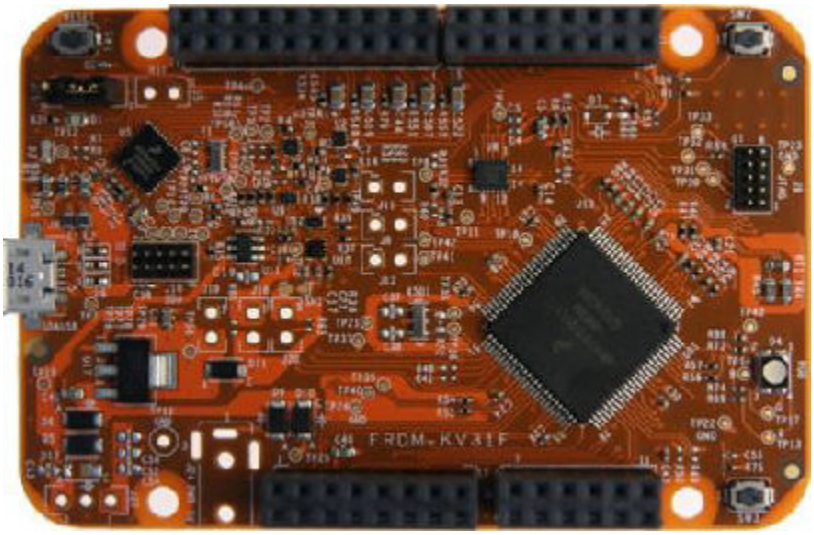

Figure 13. FRDM-KV31F Freedom development board

4.4.3 FRDM-KV11Z board

The FRDM-KV11Z board is a low-cost development tool for the Kinetis V series KV1x MCU family built upon the Arm Cortex-M0+ processor. The FRDM-KV11Z board hardware is form-factor compatible with the Arduino R3 pin layout, providing a broad range of expansion board options. The FRDM-KV11Z platform features OpenSDA, the open-source hardware embedded serial and debug adapter running an open-source bootloader.

To begin, configure the jumpers on the FRDM-KV11Z Freedom System module properly. Table 10 lists the specific jumpers and their settings for the FRDM-KV11Z Freedom System module.

Table 10. FRDM-K11Z jumper settings

Jumper	Setting
J10	1-2

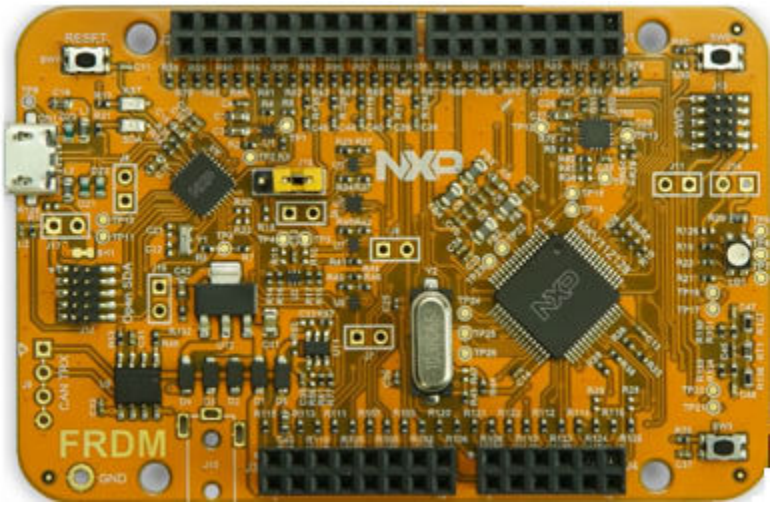

Figure 14. FRDM-KV11Z Freedom development board

4.4.4 FRDM-KE15Z board

The FRDM-KE15Z is a low-cost development tool for Kinetis KE1x family of MCUs built around the Arm Cortex-M0+ core. The FRDM-KE15Z hardware is form-factor compatible with the Arduino R3 pin layout, providing a broad range of expansion board options. The FRDM-KE15Z platform features OpenSDA, the NXP open-source hardware embedded serial and debug adapter running an open-source bootloader.

To begin, configure the jumpers on the FRDM-KE15Z Freedom System module properly. Table 11 lists the specific jumpers and their settings for the FRDM-KE15Z Freedom System module.

Table 11. FRDM-KE15Z jumper settings

Jumper	Setting	Jumper	Setting	Jumper	Setting
J7	1-2	J10	1-2	J15	2-3
J8	1-2	J14	1-2		

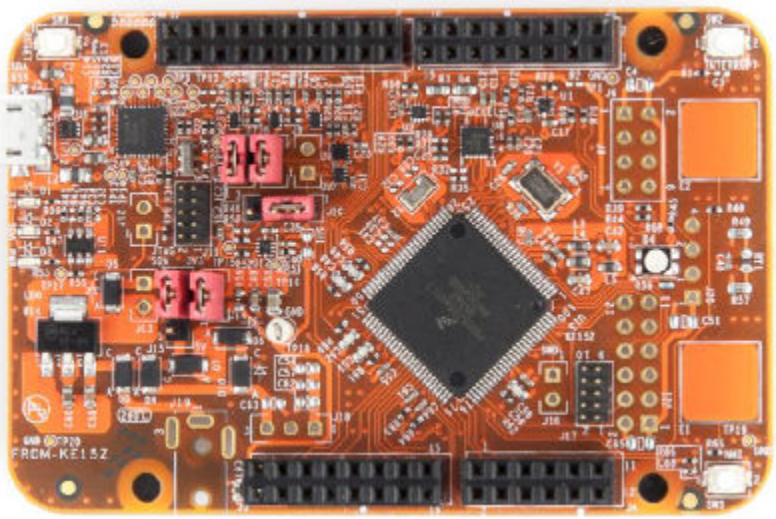

Figure 15. FRDM-KE15Z Freedom development board

4.4.5 Freedom system assembling

1. Connect the FRDM-MC-LVPMSM shield on top of the FRDM-KVxxx board (there is only one possible option).
2. Connect the Linix motor 3-phase wires to the screw terminals on the board.
3. Plug the USB cable from the USB host to the OpenSDA micro USB connector.
4. Plug the 24-V DC power supply to the DC power connector.

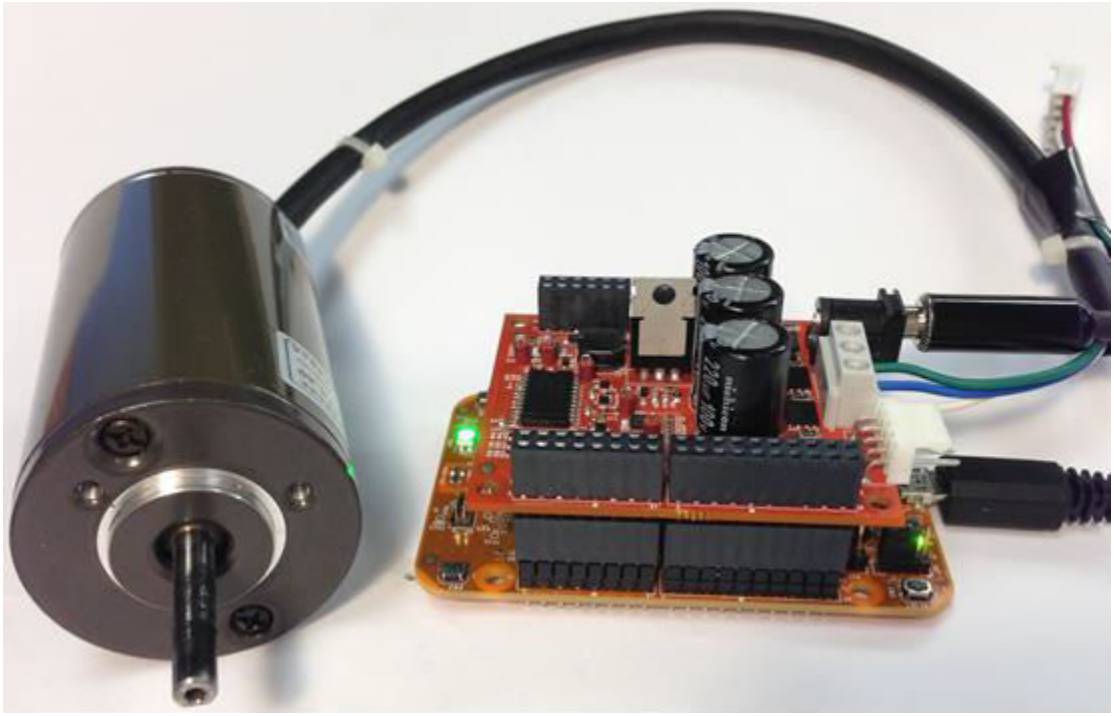

Figure 16. Assembled Freedom system

4.5 Running PMSM application on High-Voltage Platform

To run the PMSM application within the NXP High-Voltage Platform, you need these components:

- HVP daughter card with a Kinetis V or E series MCU (HVP-KV31F120M, HVP-KV46F150M, HVP-KV58F, or HVP-KE18F).
- High-Voltage Platform power stage (HVP-MC3PH) (motor not included).

You can order all the modules of the High-Voltage Platform from www.nxp.com or from distributors, and easily build the hardware platform for the target application.

4.5.1 HVP-MC3PH

The NXP High-Voltage Platform is an evaluation and development solution for Kinetis V and E series MCUs. This platform enables the development of 3-phase PMSM, BLDC, and ACIM motor-control and power-factor-correction solutions in a safe high-voltage environment. Platform is a 115/230-V, 1-kW power stage that is an integral part of the NXP embedded motion control series of development tools. It is supplied in the HVP-MC3PH kit. In combination with an HVP daughter card, it provides a ready-made software development platform for more than 1-horsepower motors. The block diagram of a complete high-voltage motor-control development kit is in Figure 17.

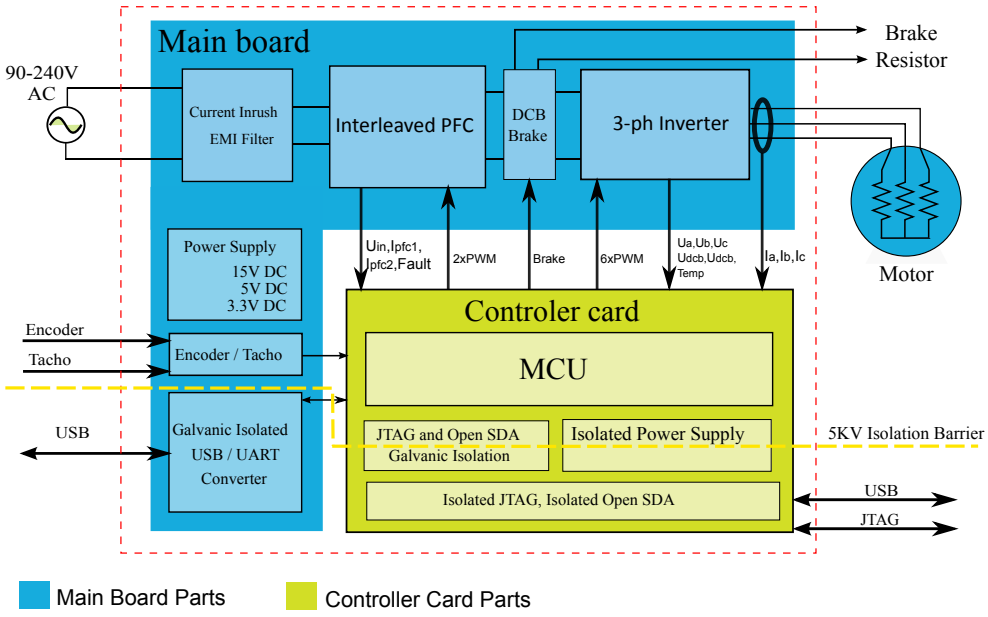

Figure 17. High-voltage motor-control development platform block diagram

The HVP-MC3PH power stage does not require a complicated setup and there is only one way to connect a daughter card to the HVP. The boards work in the default configuration, and you don't have to set any jumpers to run the attached application. It is strongly recommended to read the complete *High-Voltage Motor Control Platform User's Guide* (document HVPMC3PHUG). Note that due to high-voltage, the HVP platform may represent safety risk when not handled correctly. For more information about the NXP high-voltage motor-control development platform, see nxp.com.

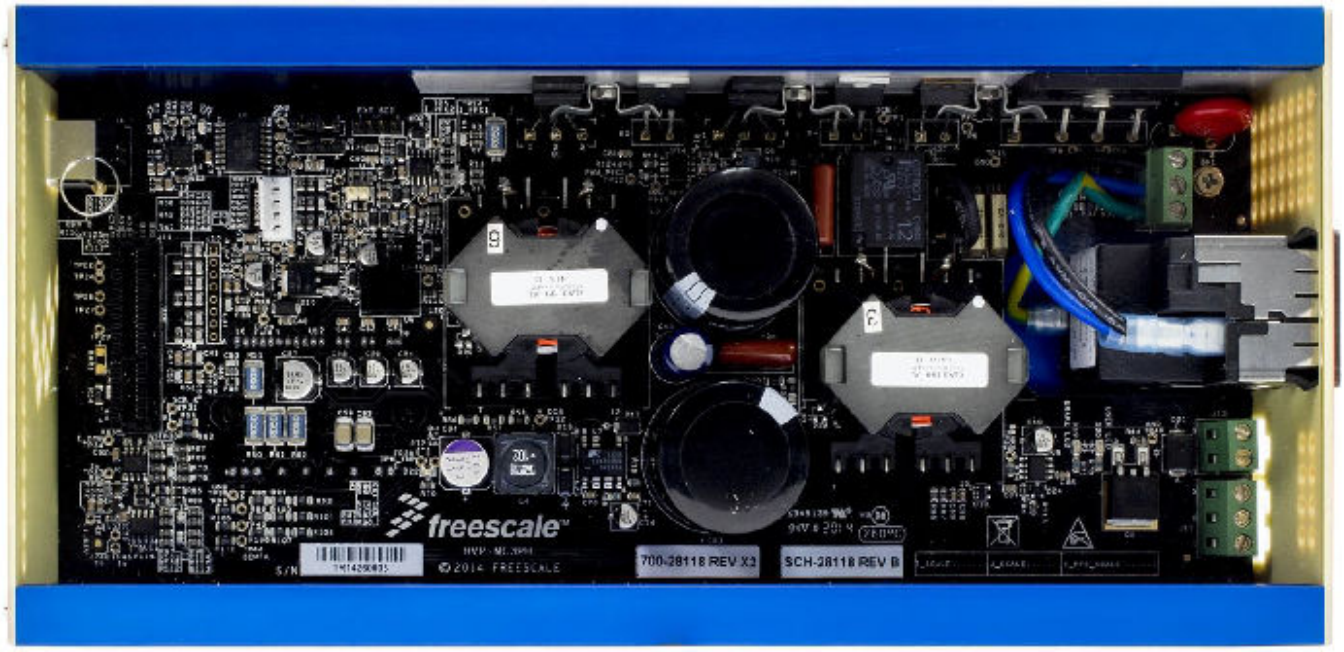

Figure 18. HVP-MC3PH high-voltage platform

4.5.2 HVP-KV11Z daughter card

The HVP-KV11Z MCU daughter card contains a Kinetis KV1x family MCU built around the Arm Cortex-M0+ core running at 75 MHz and containing up to 128 KB of flash memory. This daughter card is developed for use in motor-control applications, together with the High-Voltage Platform power stage. This daughter card features OpenSDA, the NXP open-source hardware embedded serial and debug adapter running an open-source bootloader.

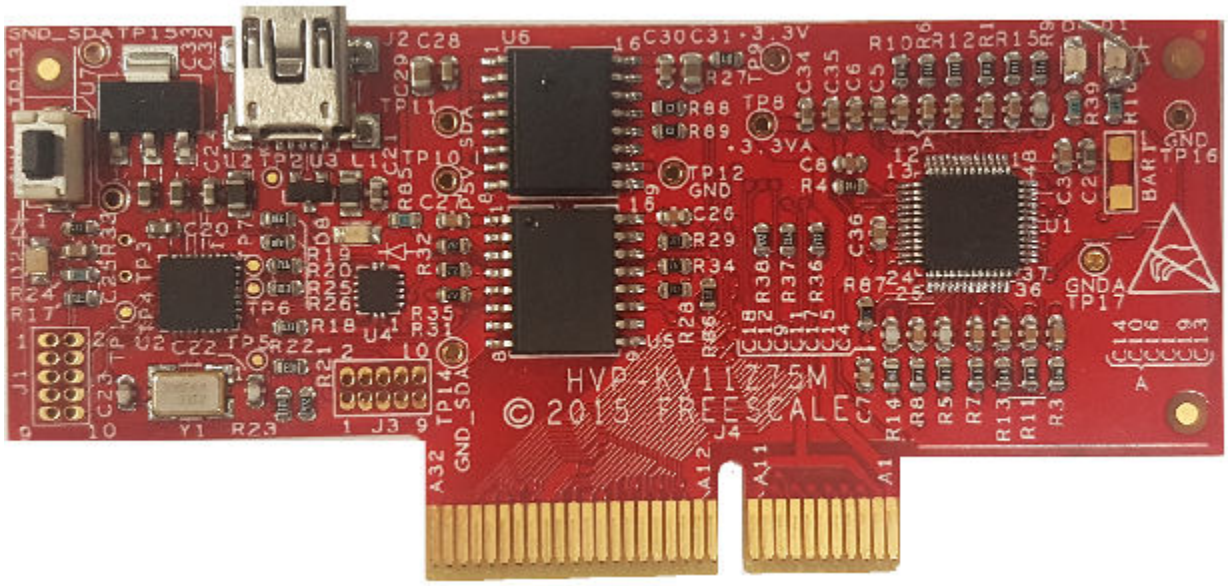

Figure 19. HVP-KV11Z daughter card

4.5.3 HVP-KV31F512M daughter card

The HVP-KV31F512 MCU daughter card contains a Kinetis KV3x family MCU built around the Arm Cortex-CM4 core with a floating-point unit, running at 120 MHz, and containing up to 512 KB of flash memory. This daughter card is developed for use in motor-control applications, together with the High-Voltage Platform power stage. This daughter card features OpenSDA, the NXP open-source hardware embedded serial and debug adapter running an open-source bootloader.

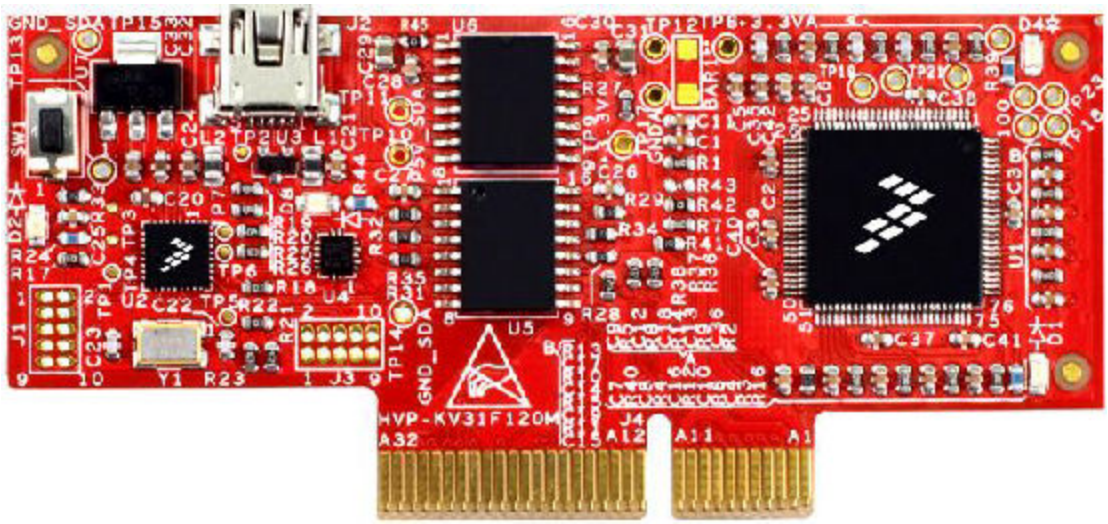

Figure 20. HVP-KV31F120M daughter card

4.5.4 HVP-KV46F150M daughter card

The HVP-KV46F150M MCU daughter card contains a Kinetis KV4x family MCU built around the Arm Cortex-CM4 core with a floating-point unit, running at 150 MHz, and containing up to 256 KB of flash memory. This daughter card is developed for use in motor-control applications, together with the High-Voltage Platform power stage. This daughter card features OpenSDA, the NXP open-source hardware embedded serial and debug adapter running an open-source bootloader.

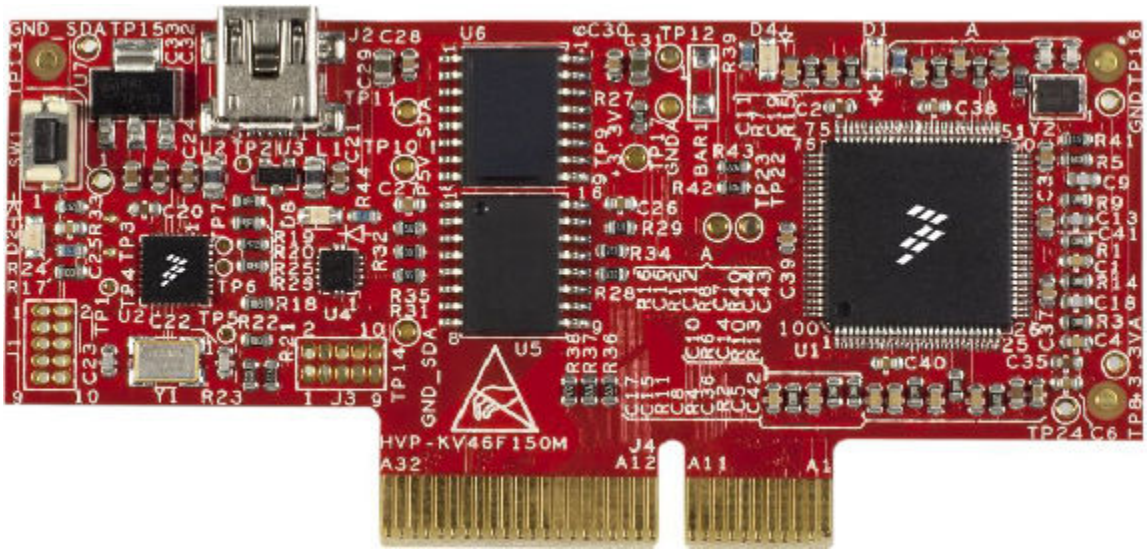

Figure 21. HVP-KV46F150M daughter card

4.5.5 HVP-KV58F daughter card

The HVP-KV58F MCU daughter card contains a Kinetis KV5x family MCU built around the Arm Cortex-CM7 core with a floating-point unit, running at 240 MHz, and containing up to 1 MB of flash memory. This daughter card is developed for use in motor-control and connectivity applications, together with the High-Voltage Platform power stage. This daughter card features OpenSDA, the NXP open-source hardware embedded serial and debug adapter running an open-source bootloader.

Figure 22. HVP-KV58F daughter card

4.5.6 HVP-KE18F daughter card

The HVP-KE18F MCU daughter card contains a Kinetis KE1xF family MCU built around the Arm Cortex-CM4 core with a floating-point unit, running at 168 MHz and containing up to 256 KB of flash memory. This daughter card is developed for use in motor-control applications, together with the High-Voltage Platform power stage. This daughter card features OpenSDA, the NXP open-source hardware embedded serial and debug adapter running an open-source bootloader.

Figure 23. HVP-KE18F development board

4.5.7 High-Voltage Platform assembling

1. Check whether the HVP-MC3PH main board is unplugged from the voltage source.
2. Insert the HVP-KVxxx daughter board to the HVP-MC3PH main board (there is only one possible option).
3. Connect the PMSM motor 3-phase wires into the screw terminals on the board.
4. Plug the USB cable from the USB host to the OpenSDA micro USB connector.
5. Plug a 230-V power supply to the power connector and switch it on.

Chapter 5
MCU features and peripheral settings

The peripherals used for motor control differ among different Kinetis V MCUs. The peripheral settings and application timings for each MCU are described in the following sections. The differences among MC platforms are at each MCU. These differences are summarized in tables.

5.1 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 on the Arm Cortex-M0 core running at 75 MHz with up to 128 KB of flash and up to 16 KB of RAM, it delivers a platform enabling customers to build a scalable solution portfolio. Additional features include dual 16-bit ADCs sampling at up to 1.2 MS/s in a 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).

Hardware timing and synchronization

Correct and precise timing is crucial for motor-control applications. Therefore, the motor-control-dedicated peripherals take care about the timing and synchronization on the hardware layer. It is also possible to set the PWM frequency as a multiple of the ADC interrupt (FOC calculation) frequency, in this case **FOCfreq = PWMfreq/2**. The timing diagram is in Figure 24.

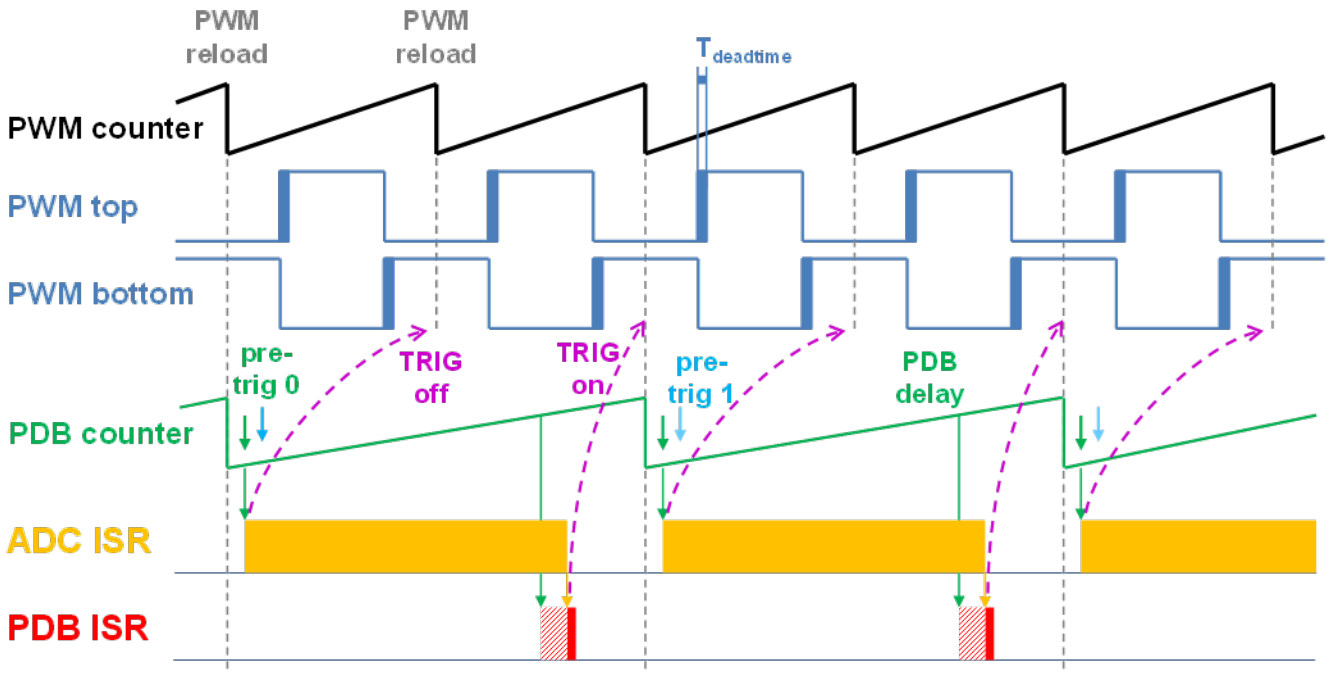

Figure 24. Hardware timing and synchronization on KV11Z

- The top signal (**PWM counter**) shows the FTM counter reloads. At the **PWM top** and **PWM bottom** signals, the dead time is emphasized. 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 approximately **Tdeatime/2** delay. 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. Firstly, the next **FTM_TRIG** is disabled (**TRIG off**). This ensures that the **PDB counter** does not reset at the next **PWM reload**. Then the FOC is calculated.

- In the middle of the next PWM period (**PDB delay**), the **PDB ISR** is called. This interrupt only enables the **FTM_TRIG** (**TRIG on**) in 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 the DC-bus voltage measurement) immediately after the first conversion is completed.

Peripheral settings

This chapter describes only the peripherals used for motor control. On KV11Z, a 6-channel FlexTimer (FTM) is used for 6-channel PWM generation and two 16-bit SAR ADCs are used for the phase currents and DC-bus voltage measurement. The FTM and ADC are synchronized via the Programmable Delay Block (PDB). There is also one channel from another independent FTM used for the slow loop interrupt generation.

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.
- Channels 0+1, 2+3, and 4+5 are combined in pairs running in a complementary mode (each).
- The fault mode is enabled at each combined pair with automatic fault clearing (PWM outputs are re-enabled the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as how long it takes the FTM to count from CNTIN to MOD. By default CNTIN = -MODULO/2 = -3750 and MOD = MODULO/2 -1 = 3749. The FTM is clocked from the System clock (75 MHz) so it takes 0.0001 s (10 KHz).
- Dead-time insertion is enabled at each combined pair. The dead-time length is calculated as System clock 75 MHz * Tdeadtime. The dead-time varies among platforms.
- The FTM generates a trigger to the PDB on counter initialization.
- The FTM fault input is enabled but its polarity and source varies among platforms.

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 3.125 MHz and continues the conversion and averaging with 32 samples 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.
- There is an interrupt that serves the FOC fast-loop algorithm generated after the first conversion is completed.

PWM and ADC synchronization – PDB0

- Unlike the FTM, the PDB is clocked from the Bus clock which is three times slower than the System clock (used for FTM). Therefore, the modulo value at PDB is divided by three.
- The PDB is triggered from the FTM0_TRIG.
- The pre-trigger 0 at each channel is generated 0.5 * $T_{deadtime}$ 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 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.
- The PDB Sequence Error and PDB Delay interrupts have a common interrupt vector. Which event generated the interrupt is determined at the beginning of the interrupt according to the ERR flag.

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*VDD) which corresponds to 7.73 A (for the 8.25 A scale).
- The CMP filter is enabled and four consecutive samples must agree.

Slow loop interrupt generation – FTM2

- The FTM2 is clocked from the System clock / 16, because the slow loop is usually ten times slower than the fast loop, so its modulo value can be kept reasonably low.
- The FTM counts from CNTIN = 0 to MOD = MODULO/16 * 10.
- An interrupt is enabled and generated at the counter reload and it serves the slow loop.

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.

Peripheral settings differences among platforms

There are some differences in peripheral settings among different platforms. Table 12 summarizes these differences.

Table 12. KV11 platform differences

Peripheral	Feature	Platform		
		FRDM	Tower	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 us	0.5 us	1.5 us
SPI	Driver on SPI	No	Yes	No
PDB	Pre-trigger 0 delay	0.25 us	0.25 us	0.75 us

CPU load and memory usage

The following information apply to the *demo* application built with IAR IDE. Table 13 shows the memory usage and the CPU load. The memory usage is calculated from the *.map* linker file, including the 2-KB FreeMASTER recorder buffer (allocated in RAM). The CPU load is measured using the *SysTick* timer. The CPU load is dependent on the fast-loop (FOC calculation) and slow-loop (speed loop) frequencies. In this case, it applies for the fast loop of 10 KHz and the slow loop of 1 KHz. The total CPU load is calculated according to the following equations.

$$CPU_{fast} = cycles_{fast} \frac{f_{fast}}{f_{CPU}} 100\ [\%]$$

$$CPU_{slow} = cycles_{slow} \frac{f_{slow}}{f_{CPU}} 100\ [\%]$$

$$CPU_{total} = CPU_{fast} + CPU_{slow}\ [\%]$$

Where:

CPU_{fast} - the CPU load taken by the fast loop.

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

f_{fast} - the frequency of the fast-loop calculation (10 KHz).

f_{CPU} - the CPU frequency.

CPU_{slow} - the CPU load taken by the slow loop.

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

f_{slow} - the frequency of the slow-loop calculation (1 KHz).

CPU_{total} - the total CPU load consumed by the motor control.

Table 13. KV11 CPU load and memory usage

	MKV11Z
CPU load [%]	59.3
Flash usage [B]	14 081
RAM usage [B]	3 091

5.2 KV3x family

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

Hardware timing and synchronization

Correct and precise timing is crucial for motor-control applications. Therefore, the motor-control-dedicated peripherals take care about the timing and synchronization on the hardware layer. In addition, it is possible set the PWM frequency as a multiple of the ADC interrupt (FOC calculation) frequency, in this case **FOCfreq = PWMfreq/2**. The timing diagram is shown in Figure 25.

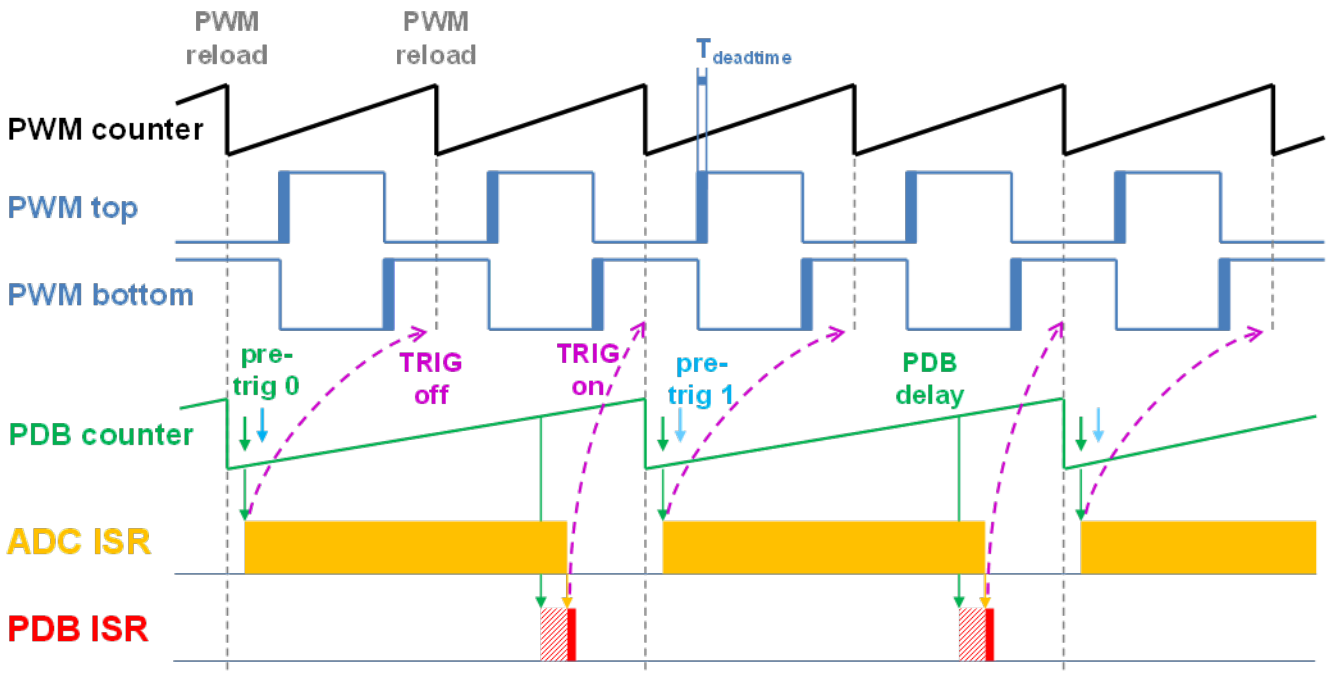

Figure 25. Hardware timing and synchronization on KV31F

- 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 a first pre-trigger for the first ADC (phase current) sample with approximate delay of **Tdeatime/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 no **PDB counter** resets 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 only enables the FTM_TRIG (**TRIG on**) in 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.

Peripheral settings

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

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 running in the complementary mode.
- The fault mode is enabled at each combined pair with the automatic fault clearing (PWM outputs are re-enabled the first PWM reload after the fault input returns to zero).

- The PWM period (frequency) is determined as how long it takes 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 System clock (60 MHz), so it takes 0.0001 s (10 kHz).
- The dead-time insertion is enabled at each combined pair. The dead-time length is calculated as the System clock 60 MHz * $T_{deadtime}$. The dead time varies among platforms.
- The FTM generates a trigger to the PDB on counter initialization.
- The FTM fault input is enabled, but its polarity and source varies among platforms.

Analog sensing – ADC0, 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 the ADC calibration purposes, the ADC clock is 6 MHz and continues the conversion and averaging with 32 samples 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 from the PDB pre-triggers.
- The interrupt that serves the FOC fast loop algorithm is generated after the first conversion is completed.

PWM and ADC synchronization – PDB0

- Like the FTM, the PDB is clocked from the 60-MHz Bus clock.
- The PDB is triggered from the FTM0_TRIG.
- The pre-trigger 0 at each channel is generated 0.5 * $T_{deadtime}$ 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 this ADC.
- The PDB Delay interrupt is enabled. This interrupt is generated when the PDB_IDLY is reached. This interrupt enables the FTM_TRIG.
- The PDB Sequence Error and PDB Delay interrupts have a common interrupt vector. Which event generated the interrupt is determined at the beginning of the interrupt according to the ERR flag.

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*VDD), which corresponds to 7.73 A (for the 8.25 A scale).
- The CMP filter is enabled and four consecutive samples must agree.

Slow loop interrupt generation – FTM2

- The FTM2 is clocked from the System clock / 16, because the slow loop is usually ten times slower than the fast loop and the modulo value can be kept reasonably low.
- The FTM counts from CNTIN = 0 to MOD = MODULO/16 * 10.
- An interrupt is enabled and generated at the reload and it serves the slow loop.

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.

Peripheral settings differences among platforms

There are some differences in the peripheral settings among different platforms. Table 14 summarizes these differences.

Table 14. KV31 platform differences

Peripheral	Feature	Platform		
		FRDM	Tower	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 software, no hardware connection FTM	FAULT 0, input pin
	Fault polarity	Active high	Active high	Active low
	Dead-time	0.5 us	0.5 us	1.5 us
SPI	Driver on SPI	No	Yes	No
PDB	Pre-trigger 0 delay	0.25 us	0.25 us	0.75 us

CPU load and memory usage

The following information apply for the *demo* application built with the IAR IDE. Table 15 shows the memory usage and the CPU load. The memory usage is calculated from the *.map* linker file, including the 2-KB FreeMASTER recorder buffer (allocated in RAM). The CPU load is measured using the *SysTick* timer. The CPU load is dependent on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies for the fast loop of 10 KHz and slow loop of 1 KHz. The total CPU load calculation is described in CPU load and memory usage.

Table 15. KV31 CPU load and memory usage

	MKV31F
CPU load [%]	29,4
Flash usage [B]	14 447
RAM usage [B]	3 087

5.3 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 a 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).

Hardware timing and synchronization

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 FOCfreq = PWMfreq / 2. The timing diagram is shown in Figure 26:

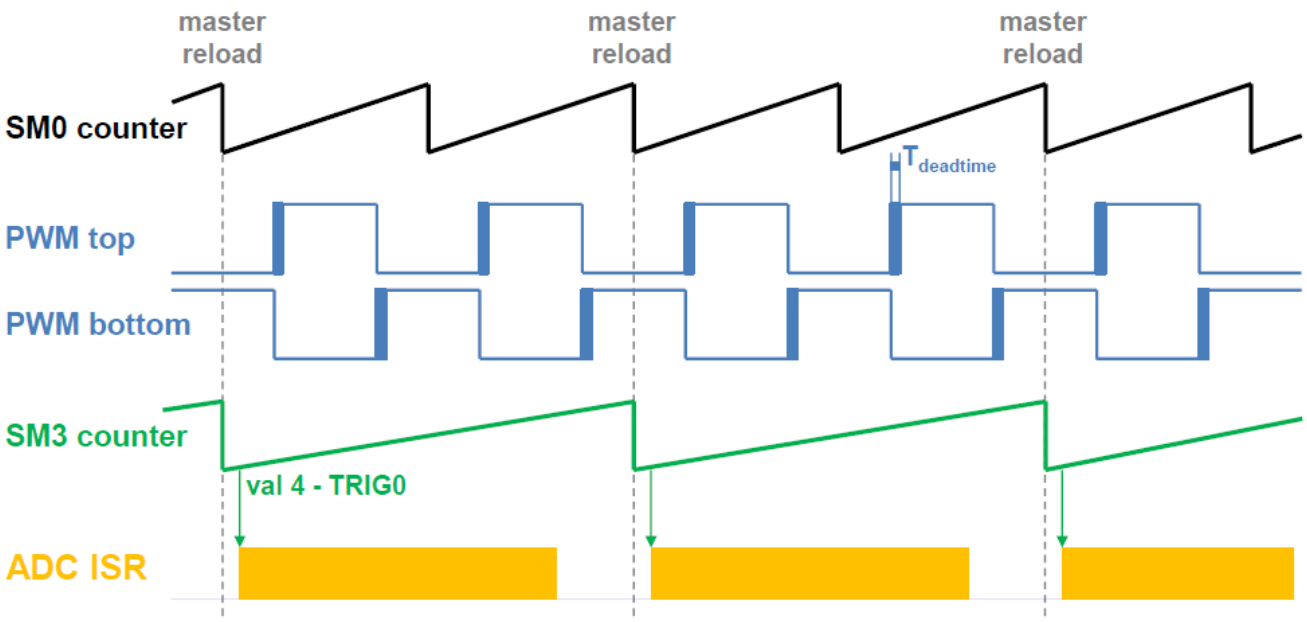

Figure 26. Hardware timing and synchronization on KV46F

- 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 $T_{deadtime}$ / 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.

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.

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 n^{th} opportunity, depending on the 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).
- The 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.

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.

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.

Slow loop interrupt generation—FTM1

- The slow loop is usually ten times 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.

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.

Peripheral settings differences among platforms

There are some differences in peripheral settings among different platforms. Table 16 summarizes those differences.

Table 16. KV46F platform differences

Peripheral	Feature	Platform	
		Tower System	HVP
FTM0	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 us	1.5 us
	SM3_VAL4	10 dec (delay 0.27 us)	28 dec (delay 0.76 us)
SPI	Driver on SPI	Yes	No

CPU load and memory usage

The following information apply for the *demo* application built with the IAR IDE. Table 17 shows the memory usage and CPU load. The memory usage is calculated from the *.map* linker file including the 2-KB FreeMASTER recorder buffer (allocated in RAM) and the 4.2-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) frequencies. In this case, it applies for the fast loop of 10 KHz and slow loop of 1 KHz. The total CPU load calculation is described in Table 17.

Table 17. KV46F CPU load and memory usage

	KV46F
CPU load [%]	15.3
Flash usage [B]	24 772
RAM usage [B]	3 753

5.4 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 a 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 the *KV5x Reference Manual* (document KV5XP144M220RM).

Hardware timing and synchronization

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 FOCfreq = PWMfreq / 2. The timing diagram is shown in Figure 27.

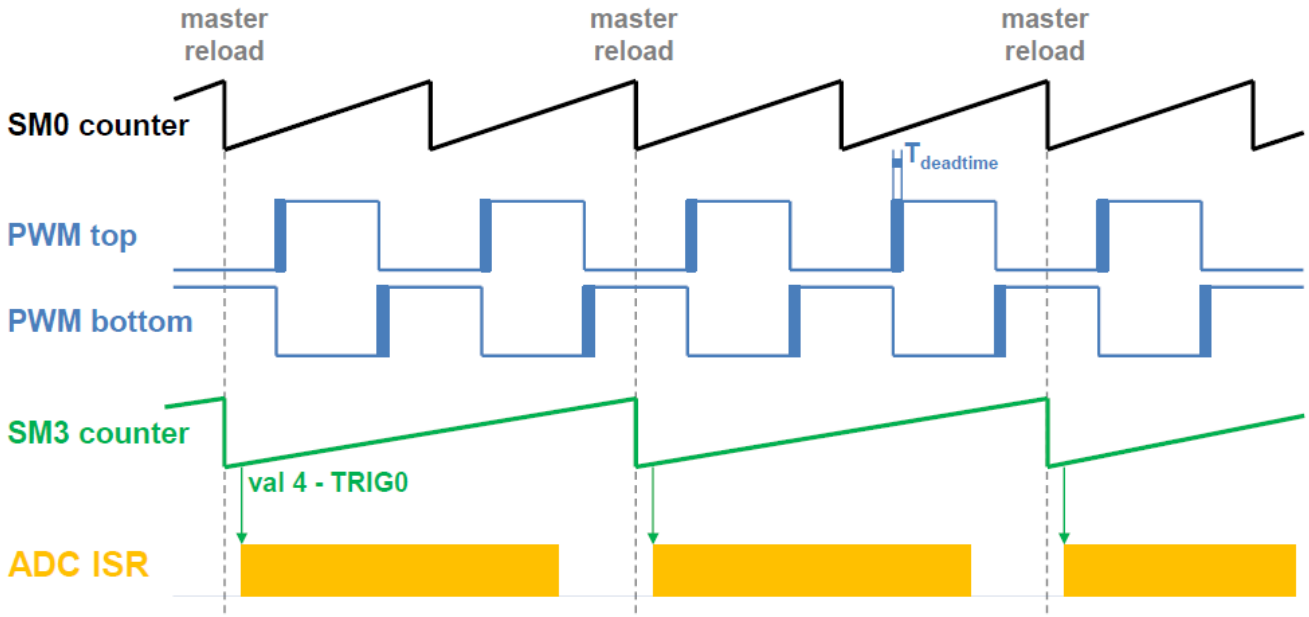

Figure 27. Hardware timing and synchronization on KV58F

- 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 the 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.

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.

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 the 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 MHz × $T_{deadtime}$. The dead time varies among the platforms.

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.

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.

Slow loop interrupt generation—FTM2

- The slow loop is usually ten times 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.

Communication with MC33937 MOSFET driver—SPI

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

Peripheral settings differences among platforms

There are some differences in peripheral settings among different platforms. Table 18 summarizes those differences.

Table 18. KV58F platform differences

Peripheral	Feature	Platform	
		Tower System	HVP
FTM0	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	

CPU load and memory usage

The following information apply for the *demo* application built with the IAR IDE. Table 18 shows the memory usage and CPU load. The memory usage is calculated from the *.map* linker file including the 2-KB FreeMASTER recorder buffer (allocated in RAM) and the 4.2-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) frequencies. In this case, it applies for the fast loop of 10 KHz and slow loop of 1 KHz. The total CPU load calculation is described in Table 19.

Table 19. KV58F CPU load and memory usage

CPU load [%]	7.3
Flash usage [B]	25 917
RAM usage [B]	3 793

5.5 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 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).

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 Figure 28.

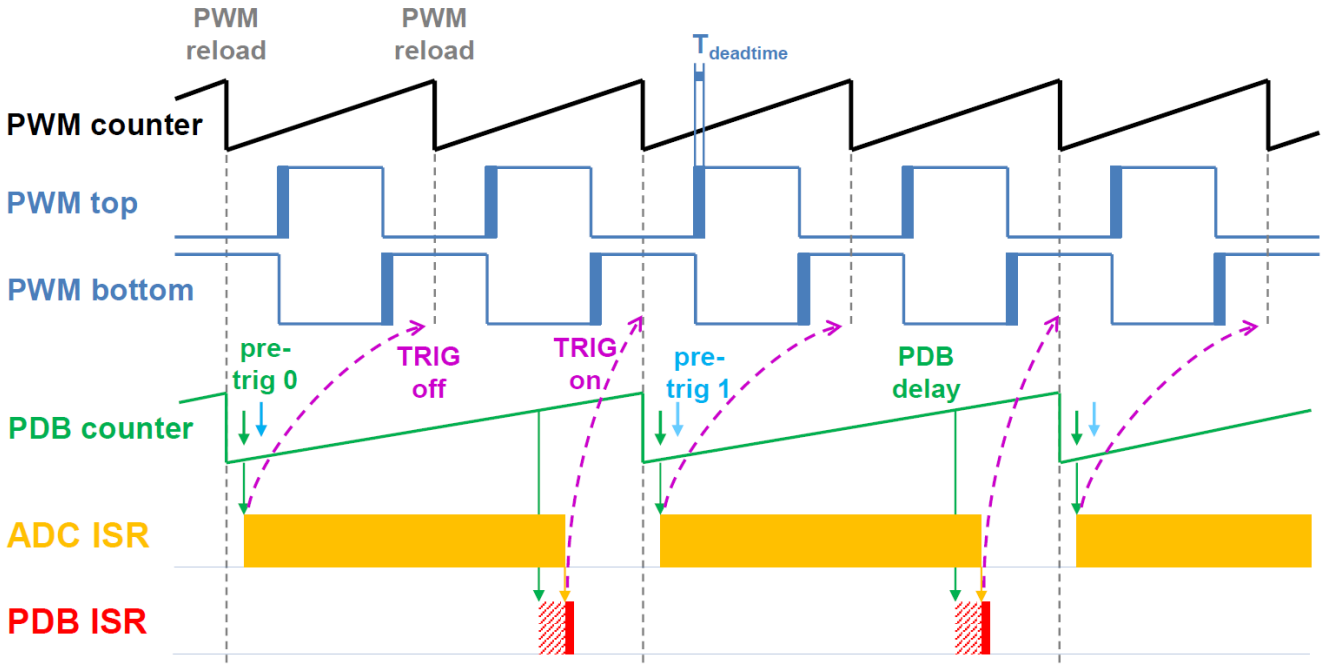

Figure 28. Hardware timing and synchronization on KE1xZ

Peripheral settings

This section describes only the peripherals used for motor control. 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.

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 × Tdeadtime. The dead time length is 0.5 µs.
- The FTM generates a trigger for the PDB on the counter initialization.

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.

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 × $T_{deadtime}$ 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 28).
- 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).

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.

CPU load and memory usage

The following information apply for the *demo* application built with IAR IDE. The Table 20 shows memory usage and CPU load. Memory usage is calculated from the *.map* linker file including the 2-KB FreeMASTER recorder buffer (allocated on RAM) and the 4.2-KB FreeMASTER Target-Side Addressing (TSA) 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) frequencies. In this case, it applies for the 10-kHz fast loop and 1-kHz slow loop. The total CPU load calculation is described in CPU load and memory usage.

Table 20. KE15Z CPU load and memory usage

	KE15Z
CPU load [%]	58.8
Flash usage [B]	20 665
RAM usage [B]	3 727

5.6 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 frequency, 512 KB of flash, and 64 KB of SRAM.

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

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 Figure 29:

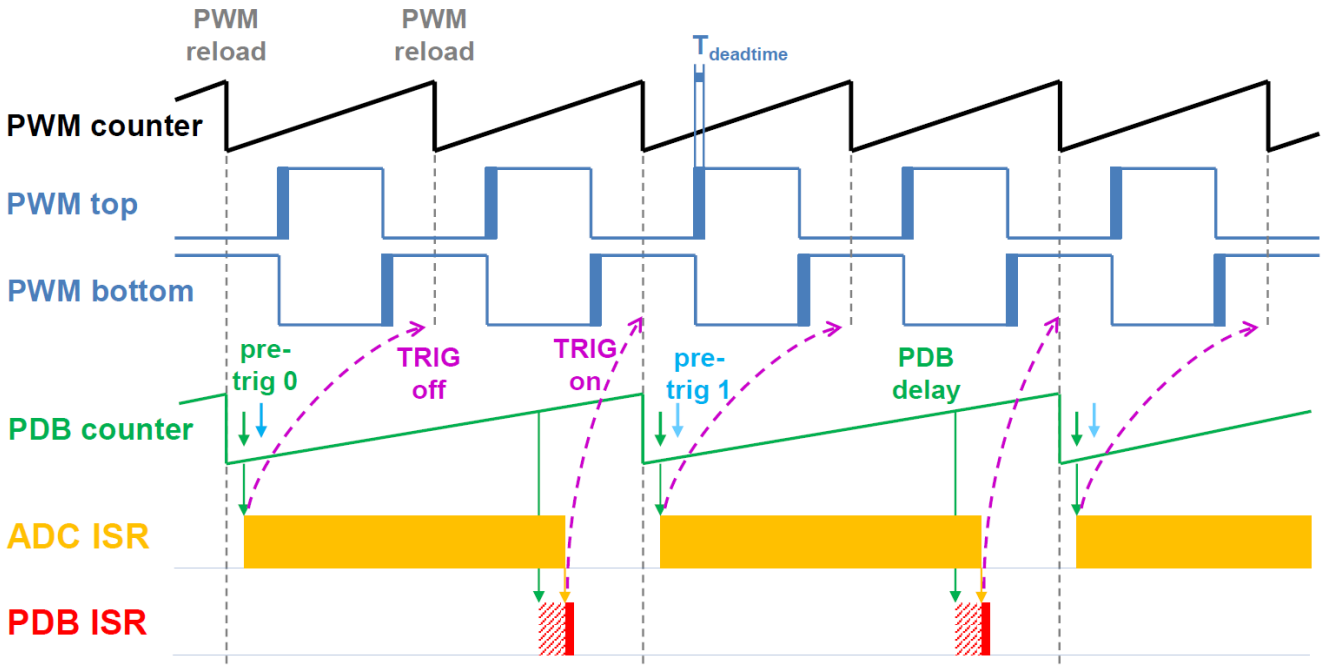

Figure 29. Hardware timing and synchronization on KE18F

Peripheral settings

This section describes only the peripherals used for motor control. 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.

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.

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.

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 29).
- 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).

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.

Communication with MC33937 MOSFET driver—SPI

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

CPU load and memory usage

The following information apply for the *demo* application built with IAR IDE. Table 21 shows the memory usage and CPU load. The memory usage is calculated from the *.map* linker file including the 2-KB FreeMASTER recorder buffer (allocated in RAM) and the 4.2-KB FreeMASTER Target-Side Addressing (TSA) 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) frequencies. In this case, it applies 10 KHz for the fast loop and 1 KHz for the slow loop. The total CPU load calculation is described in CPU load and memory usage.

Table 21. KE18F CPU load and memory usage

	KE18F
CPU load [%]	13.9
Flash usage [B]	23 247
RAM usage [B]	3 785

Chapter 6
Project file and IDE workspaces structure

All the necessary files are included in one package, which simplifies the distribution and decreases the size of the final package. The directory structure of this package is simple, easy to use, and organized in a logical manner. The folder structure used in the IDE is different from the structure of the PMSM package installation, but it uses the same files. The different organization is chosen due to a better manipulation with folders and files in workplaces and due to the possibility to add or remove files and directories. The "*pack_motor_board*" project includes all the available functions and routines, MID functions, scalar and vector control of the motor, FOC control, and FreeMASTER MCAT project. This project serves for development and testing purposes.

6.1 PMSM project structure

The directory tree of the PMSM project is shown in Figure 30.

Figure 30. Directory tree

The main project folder *pack_motor_xkxxx\boards\xkxxx\demo_apps\mc_pmsm\pmsm_snsless*contains these folders:

- *iar*—for the IAR Embedded Workbench IDE.
- *armgcc*—for the GNU Arm IDE.
- *mdk*—for the uVision Keil IDE.

The folder contains also these files:

- *m1_pmsm_appconfig.h*—contains the definitions of constants for the application control processes, parameters of the motor and regulators, and the constants for other vector-control-related algorithms. When you tailor the application for a different motor using the Motor Control Application Tuning (MCAT) tool, the tool generates this file at the end of the tuning process.
- *main.c*—contains the basic application initialization (enabling interrupts), subroutines for accessing the MCU peripherals, and interrupt service routines. The FreeMASTER communication is performed in the background infinite loop.
- *board.c*—contains the functions for the UART, GPIO, and SysTick initialization.
- *board.h*—contains the definitions of the board LEDs, buttons, UART instance used for FreeMASTER, and so on.
- *clock_config*—contains the CPU clock setup functions. These files are going to be generated by the clock tool in the future.
- *mcdrv.h*—this file ensures the abstraction of the mcdrv_lpcxpressoxx.h file inclusion.
- *mcdrv_xkxxx.c*—contains the motor-control driver peripherals initialization functions that are specific for the board and MCU used.
- *mcdrv_xkxxx.h*—header file for *mcdrv_lpcxpressoxx.c*. This file contains the macros for changing the PWM period and the ADC channels assigned to the phase currents and board voltage.
- *freemaster_cfg.h*—the FreeMASTER configuration file containing the FreeMASTER communication and features setup.
- *pin_mux*—port configuration files. It is recommended to generate these files in the pin tool.

This folder contains also the FreeMASTER project file *pmsm_float.pmp* (*pmsm_frac.pmp* for the fraction version of the MCU). Open this file in the FreeMASTER tool and use it to control the application.

The *pack_motor_xkxxx\middleware\motor_control\freemaster\pmsm_float* folder contains the auxiliary files for the MCAT tool.

(*pack_motor_xkxxx\middleware\motor_control\freemaster\pmsm_frac* for the fraction version of the MCU)

The *pack_motor_xkxxx\middleware\motor_control\pmsm\pmsm_float* folder contains these subfolders common to the other motor-control projects:

(*pack_motor_xkxxx\middleware\motor_control\pmsm\pmsm_frac* for the fraction version of the MCU)

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

Chapter 7
Tools

Install the FreeMASTER Run-Time Debugging Tool 3.0 and one of the following IDEs on your PC to run and control the PMSM application properly:

- IAR Embedded Workbench IDE v8.40.2 or higher.
- MCUXpresso v11.1.0.
- ARM-MDK - Keil µVision version 5.25.2.
- FreeMASTER Run-Time Debugging Tool 3.0.

7.1 Compiler warnings

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

The IAR Embedded Workbench IDE suppresses these warnings:

- Pa082—undefined behavior; the order of volatile accesses is not defined in this statement.
- Ta022—possible ROM access (<ptr>) from within a __ramfunc function.
- Ta023—call to a non __ramfunc function (somefunction) from within a __ramfunc function.

The Arm-MDK Keil µVision IDE suppresses these warnings:

- 66—the enumeration value is out of the “int” range.
- 1035—a single-precision operand is implicitly converted to a double-precision operand.
- 1296—the extended constant initializer is used.

By default, there are no other warnings shown during the compiling process.

Chapter 8
Motor-control peripheral initialization

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

The *mcdrv_xkxxx.h* header file provides several macros that can be defined by the user:

- *M1_MCDRV_ADC*—this macro specifies which ADC periphery is used. If you select an unsupported periphery, a preprocessor error is issued.
- *M1_MCDRV_PWM3PH*—this macro specifies which PWM periphery is used. If you select an unsupported periphery, the preprocessor error is issued.
- *M1_PWM_FREQ*—the value of this definition sets the PWM frequency.
- *M1_FOC_FREQ_VS_PWM_FREQ*—enables you to call the fast loop interrupt at every first, second, third, or n^{th} PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast loop interrupt.
- *M1_SPEED_LOOP_FREQ* —the value of this definition sets the speed loop frequency.
- *M1_PWM_DEADTIME*—the value of the PWM dead time in nanoseconds.
- *M1_PWM_PAIR_PH[A..C]*—these macros enable a simple assignment of the physical motor phases to the PWM periphery channels (or submodules). Change the order of the motor phases this way.
- *M1_ADC[1,2]_PH_[A..C]*—these macros are used to assign the ADC channels for the phase current measurement. The general rule is that at least one of the phase currents must be measurable on both ADC converters and the two remaining phase currents must be measurable on different ADC converters. The reason for this is that the selection of the phase current pair to measure depends on the current SVM sector. If this rule is broken, a preprocessor error is issued. For more information about the 3-phase current measurement, see *Sensorless PMSM Field-Oriented Control* (document DRM148).
- *M1_ADC[1,2]_UDCB*—this define is used to select the ADC channel for the measurement of the DC-bus voltage.

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

- The available APIs for the ADC are:
  - *mcdrv_adc_t*—MCDRV ADC structure data type.
  - *bool_t M1_MCDRV_ADC_PERIPH_INIT()*—this function is by default called during the ADC peripheral initialization procedure invoked by the *MCDRV_Init_M1()* function and should not be called again after the peripheral initialization is done.
  - *bool_t M1_MCDRV_CURR_3PH_CHAN_ASSIGN(mcdrv_adc_t*)*—calling this function assigns proper ADC channels for the next 3-phase current measurement based on the SVM sector. The function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB_INIT(mcdrv_adc_t*)*—this function initializes the phase-current channel-offset measurement. This function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB(mcdrv_adc_t*)*—this function reads the current information from the unpowered phases of a stand-still motor and filters them using moving average filters. The goal is to obtain the value of the measurement offset. The length of the window for moving the average filters is set to eight samples by default. This function always returns true.
  - *bool_t M1_MCDRV_CURR_3PH_CALIB_SET(mcdrv_adc_t*)*—this function asserts the phase-current measurement offset values to the internal registers. Call this function after a sufficient number of *M1_MCDRV_CURR_3PH_CALIB()* calls. This function always returns true.

  - bool_t *M1_MCDRV_ADC_GET(mcdrv_adc_t*)*—this function reads and calculates the actual values of the 3-phase currents, DC-bus voltage, and auxiliary quantity. This function always returns true.

- The available APIs for the PWM are:
  - *mcdrv_pwma_pwm3ph_t*—MCDRV PWM structure data type.
  - *bool_t M1_MCDRV_PWM_PERIPH_INIT()*—this function is by default called during the PWM periphery initialization procedure invoked by the *MCDRV_Init_M1()* function.
  - *bool_t M1_MCDRV_PWM3PH_SET(mcdrv_pwma_pwm3ph_t*)*—this function updates the PWM phase duty cycles. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_EN(mcdrv_pwma_pwm3ph_t*)*—calling this function enables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_DIS (mcdrv_pwma_pwm3ph_t*)*—calling this function disables all PWM channels. This function always returns true.
  - *bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwma_pwm3ph_t*)*—this function returns the state of the over-current fault flags and automatically clears the flags (if set). This function returns true when an over-current event occurs. Otherwise, it returns false.

Chapter 9
User interface

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

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

The application can be controlled the using these two interfaces:

- The button on the development board (controlling the demo mode):

Table 22. Control button assignment

Board	Control button	LED state indication
TWR-KV11Z75	SW2	D5
TWR-KV31F120	SW2	D5
TWR-KV46F150M	SW2	D5
TWR-KV58F220M	SW2	D5
TWR-KE18F	SW2	D5
FRDM-KV11Z	SW2	D4
FRDM-KV31F	SW2	D4
FRDM-KE15Z	SW2	D4
HVP-KV11Z	-	D20
HVP-KV31F	-	D20
HVP-KV46F	-	D20
HVP-KV58F	-	D20
HVP-KE18F	-	D20

- Remote control using FreeMASTER:
  - Using the Motor Control Application Tuning (MCAT) interface in the “Control Structure” tab or the “Application control” tab (controlling the demo mode).
  - Setting a variable in the FreeMASTER Variable Watch.

Chapter 10
Remote control using FreeMASTER

This section provides information about the tools and recommended procedures to control the sensor/sensorless PMSM Field-Oriented Control (FOC) application using FreeMASTER. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool for communication with the PC. It supports non-intrusive monitoring, as well as the modification of target variables in real time, which is very useful for algorithm tuning. Besides the target-side driver, the FreeMASTER tool requires the installation of the PC application as well. You can download FreeMASTER 3.0 at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the *pmsm_float.pmp* (for fraction version *pmsm_frac.pmp*) file located in the *pack_motor_xkxxx\boards\xkxxx\demo_apps\mc_pmsm\pmsm_snsless*. The FreeMASTER application starts and the environment is created automatically, as defined in the **.pmp* file.

10.1 Establishing FreeMASTER communication

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

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

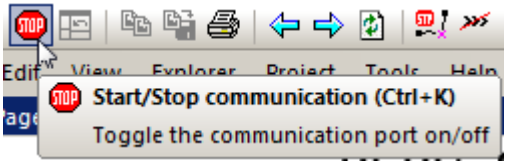

Figure 31. Red “STOP” button placed in top left-hand corner

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

Figure 32. FreeMASTER—communication is established successfully

5. Press *F5* to reload the MCAT html page and check the App ID.
6. Control the PMSM motor using the MCAT “Control structure” tab, the MCAT “Application demo control” tab, or by directly writing to a variable in a variable watch.
7. If you rebuild and download the new code to the target, turn the FreeMASTER application off and on.

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

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

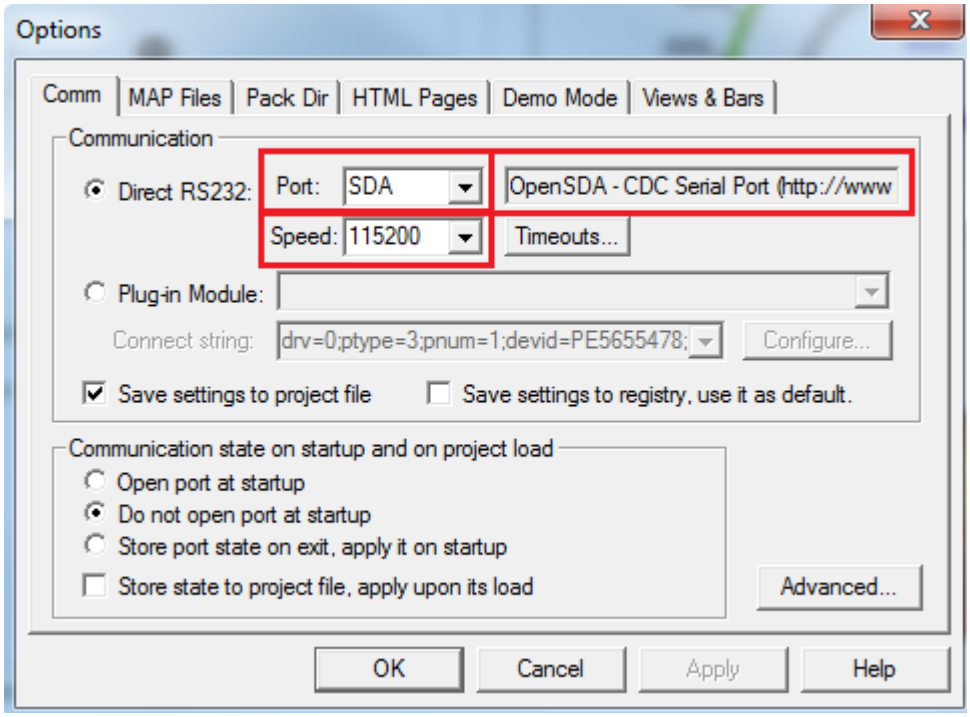

Figure 33. FreeMASTER communication setup window

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

Make sure to supply your development board from a sufficient energy source. Sometimes the PC USB port is not sufficient to supply the development board.

10.2 MCAT FreeMASTER interface (Motor Control Application Tuning)

The PMSM sensor/sensorless FOC application can be easily controlled and tuned using the Motor Control Application Tuning (MCAT) plug-in for PMSM. The MCAT for PMSM is a user-friendly modular page, which runs within FreeMASTER. The tool consists of the tab menu, tuning mode selector, and workspace shown in Figure 34. Each tab from the tab menu represents one sub-module which enables you to tune or control different aspects of the application. Besides the MCAT page for PMSM, several scopes, recorders, and variables in the project tree are predefined in the FreeMASTER project file to further simplify the motor parameter tuning and debugging. When the FreeMASTER is not connected to the target, the “App ID” line shows “offline”. When the communication with the target MCU is established using a correct software, the “App ID” line displays the board name “pmsm_used_board” and all stored parameters for the given MCU are loaded. If the connection is established and the board ID is not shown, press *F5* to reload the MCAT html page.

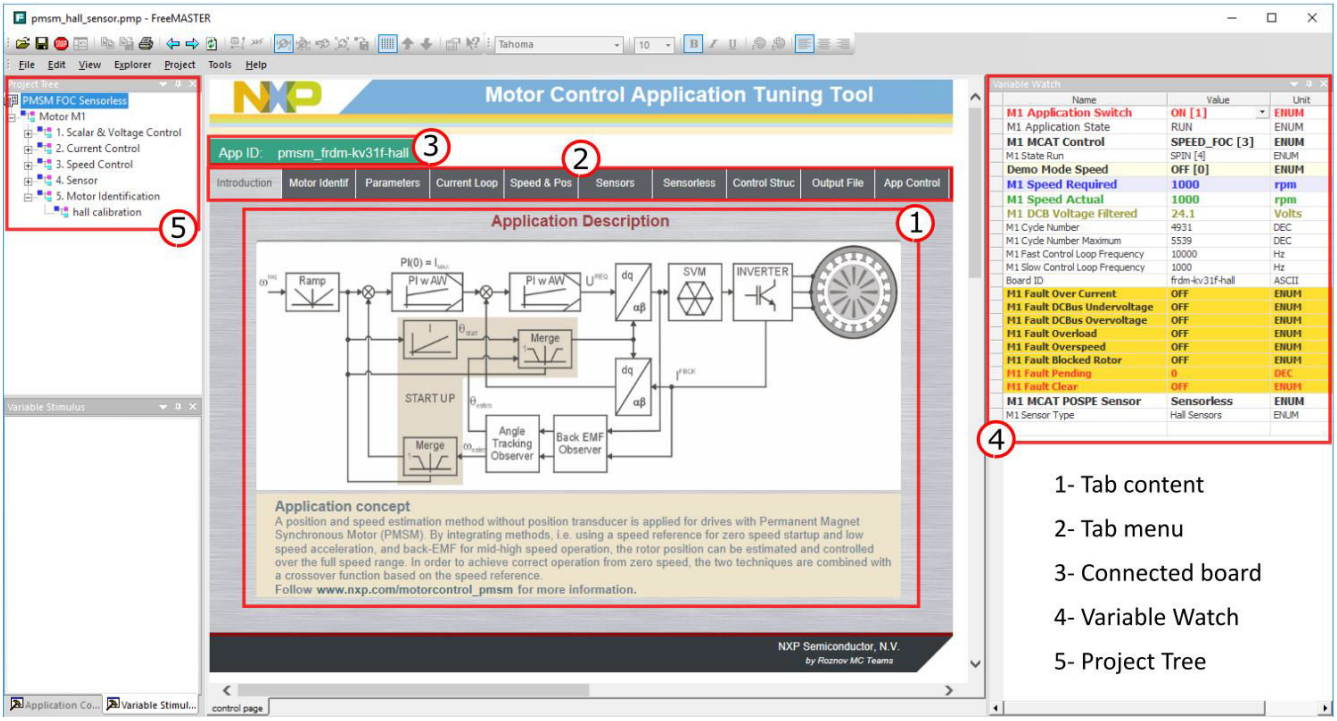

Figure 34. MCAT layout

In the default configuration, these tabs are available:

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

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

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

Control structure—“Control Struc” tab

The application can be controlled through the “Control Struc” tab, which is shown in Figure 35. The state control area on the left side of the screen shows the current application state and enables you to turn the main application switch on or off (turning a running application off disables all PWM outputs). The “Cascade Control Structure” area is placed in the right-hand side of the screen. Here you can choose between the scalar control and the FOC control using the appropriate buttons. The selected parts of the FOC cascade structure can be enabled by selecting “Voltage FOC”, “Current FOC”, and “Speed FOC” (sensor/sensorless). This is useful for application tuning and debugging.

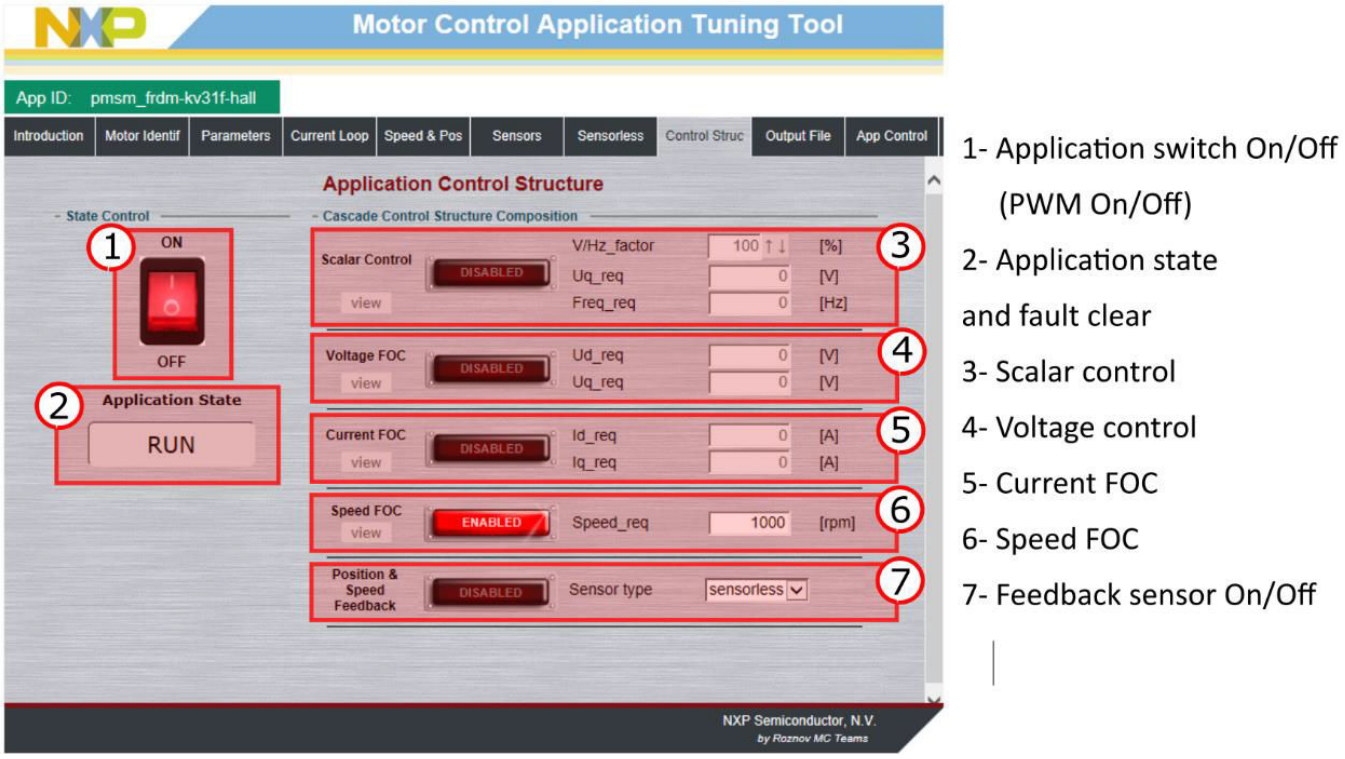

Figure 35. MCAT for PMSM control page

The scalar control diagram is shown in Figure 36. It is the simplest type of motor-control techniques. The ratio between the magnitude of the stator voltage and the frequency must be kept at the nominal value. Hence, the control method is sometimes called Volt per Hertz (or V/Hz). The position estimation BEMF observer and tracking observer algorithms (see Sensorless PMSM Field-Oriented Control (document DRM148) for more information) run in the background, even if the estimated position information is not directly used. This is useful for the BEMF observer tuning.

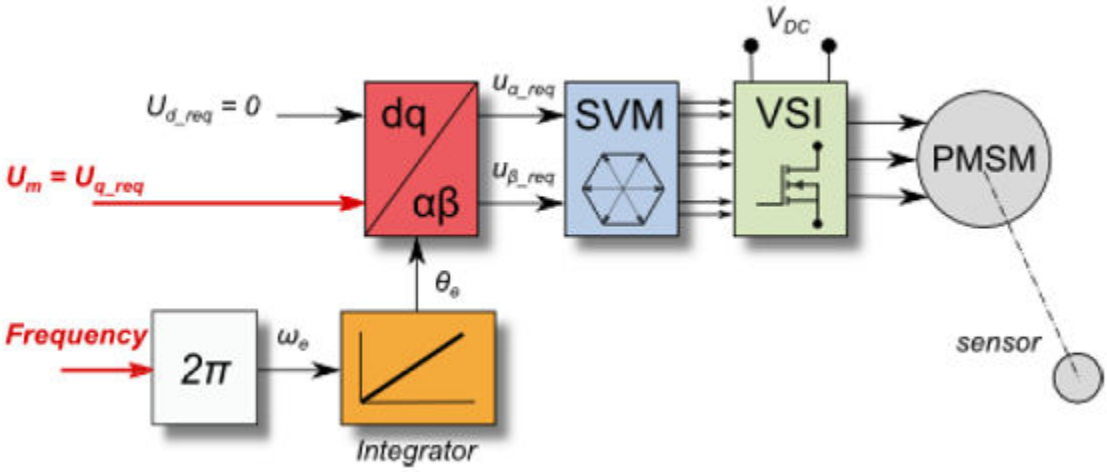

Figure 36. Scalar control mode

The block diagram of the voltage FOC is in Figure 37. Unlike the scalar control, the position feedback is closed using the BEMF observer and the stator voltage magnitude is not dependent on the motor speed. Both the d-axis and q-axis stator voltages can be specified in the “Ud_req” and “Uq_req” fields. This control method is useful for the BEMF observer functionality check.

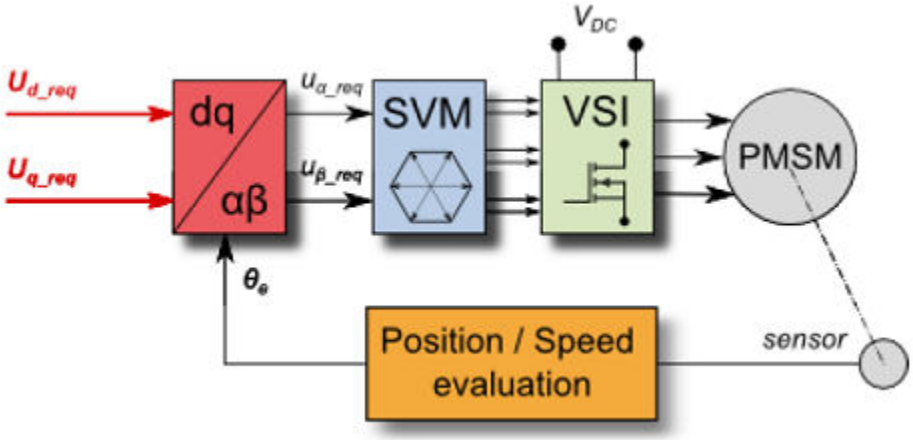

Figure 37. Voltage FOC control mode

The current FOC (or torque) control requires the rotor position feedback and the currents transformed into a d-q reference frame. There are two reference variables (“Id_req” and “Iq_req”) available for the motor control, as shown in the block diagram in Figure 38. The d-axis current component isd_req is responsible for the rotor flux control. The q-axis current component of the current isq_req generates torque and, by its application, the motor starts running. By changing the polarity of the current isq_req, the motor changes the direction of rotation. Supposing that the BEMF observer is tuned correctly, the current PI controllers can be tuned using the current FOC control structure.

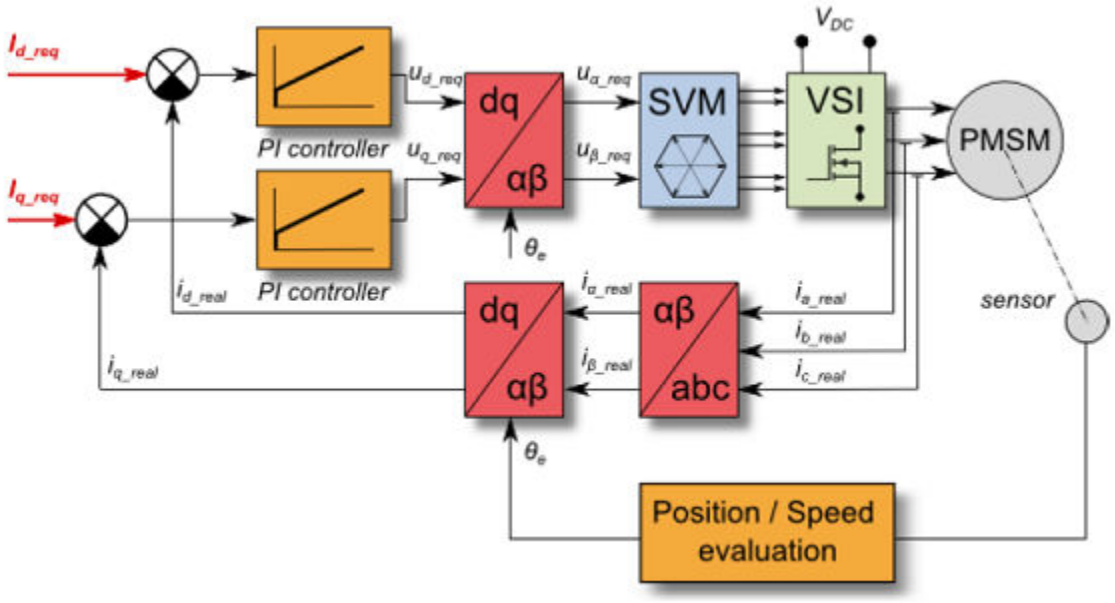

Figure 38. Current (torque) control mode

The speed PMSM sensor/sensorless FOC (its diagram is shown in Figure 39) is activated by enabling the speed FOC control structure. Enter the required speed into the “Speed_req” field. The d-axis current reference is held at 0 during the entire FOC operation.

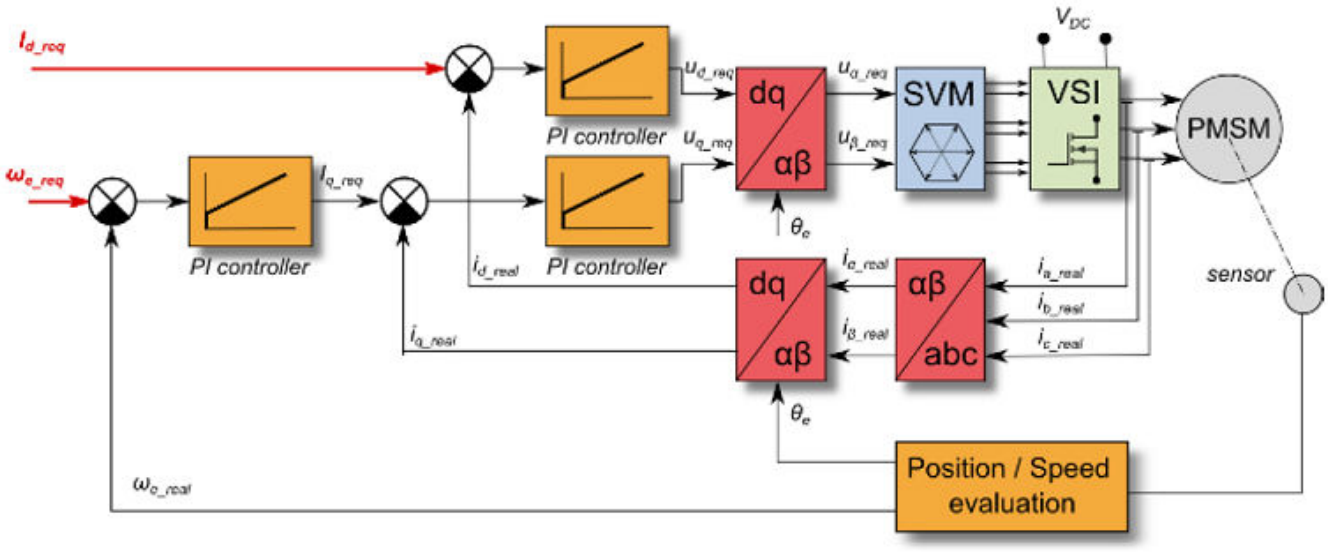

Figure 39. Speed FOC control mode

Application demo control—“App control” tab

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

- Speed gauge—shows the actual and required speeds.
- Required speed slider—sets up the required speed.
- DC-bus voltage—shows the actual DC-bus voltage.
- Current i_q—shows the actual torque-producing current.

- Current limitation—sets up the torque-producing current limit.
- Demo mode on/off button—turns the demonstration mode on/off.
- RUN/STOP PWM button—runs/stops the whole application (sets the PWM on and off).
- Notification—shows the notification about the actual application state (or faults).

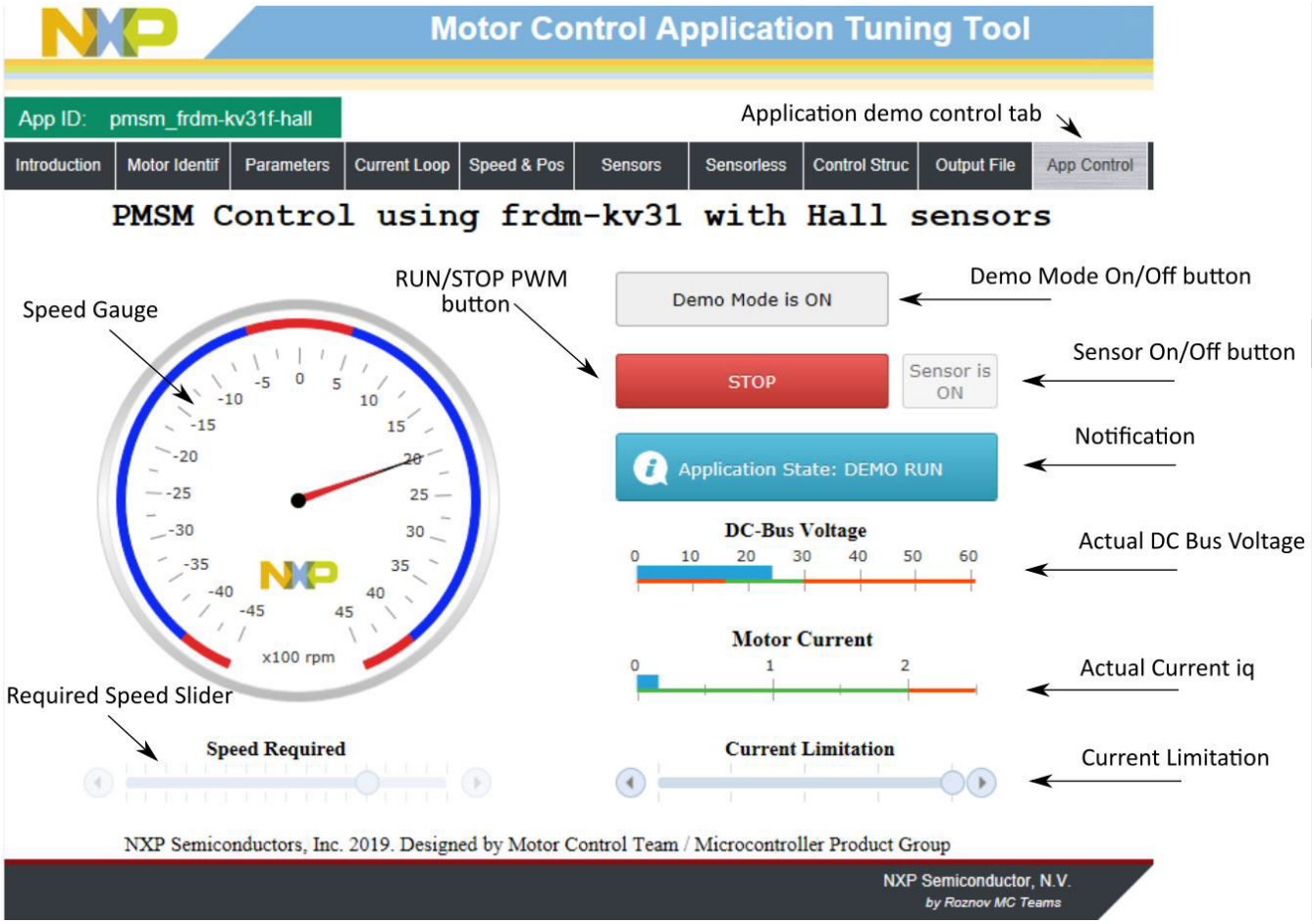

Figure 40. FreeMASTER control page

These are the basic instructions for controlling a motor:

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

Chapter 11
Identifying parameters of user motor using MCAT

This section provides a guide on how to run your own motor or tune the default motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any possible issues during the tuning process. The state diagram in Figure 41 shows a typical PMSM sensor/sensorless control tuning process.

Because the model-based control methods of the PMSM drives are the most effective and usable, obtaining an accurate model of a motor is an important part of the drive design and control. For the implemented FOC algorithms, it is necessary to know the value of the stator resistance Rs, direct inductance Ld, quadrature inductance Lq, and BEMF constant Ke. If your connected PMSM motor is not the default Teknic or Linix motor described in the previous sections, identify the parameters of your motor first. Each tuning phase is described in more detail in the following sections.

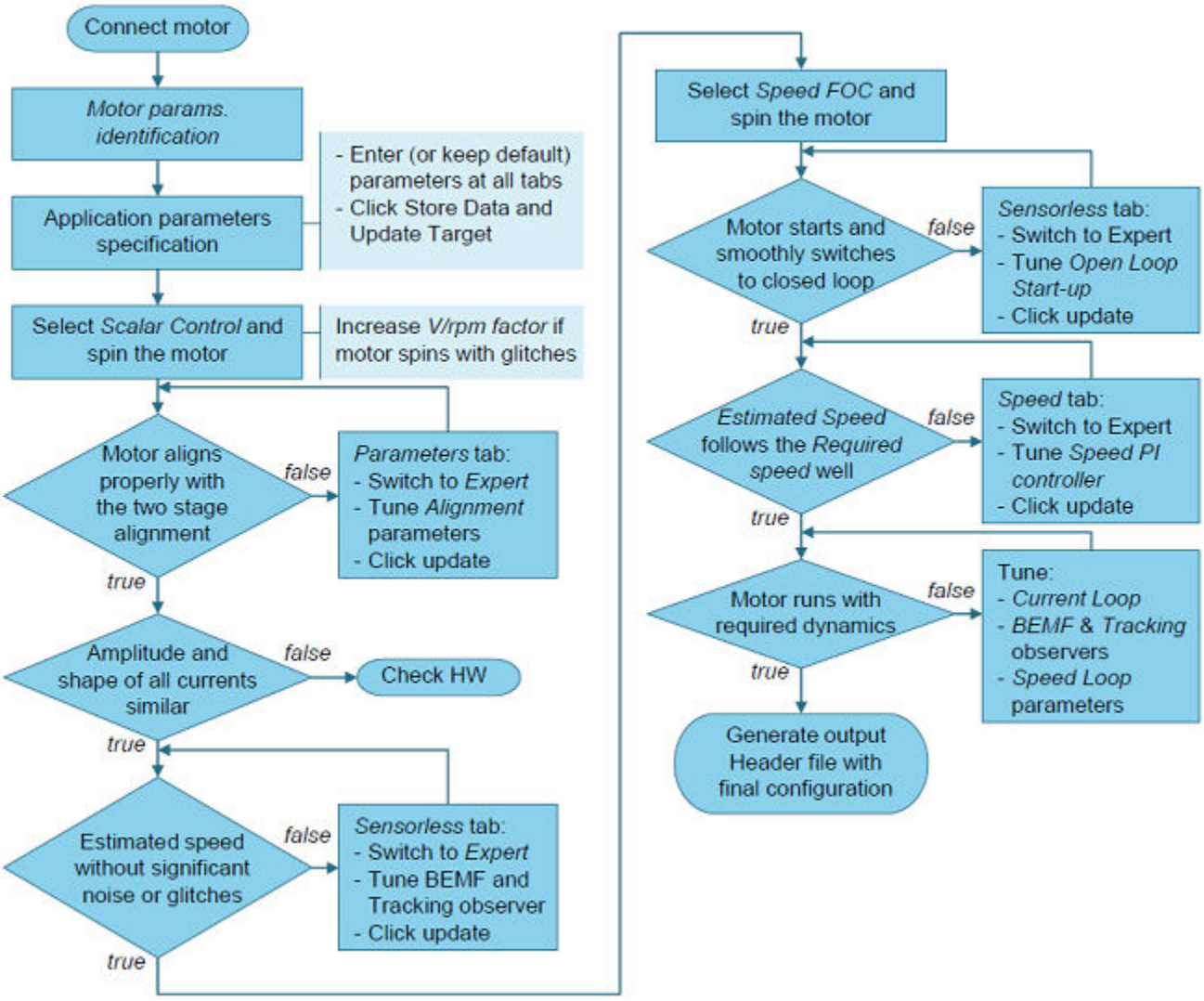

Figure 41. Running a new PMSM

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

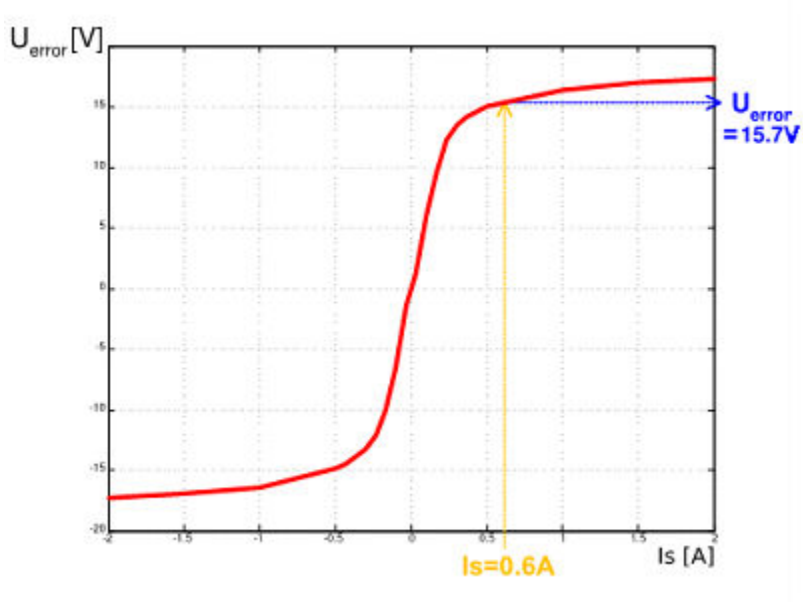

Figure 42. Example power stage characteristic

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

Stator resistance measurement

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

$$R_s = \frac{U_{DC} - U_{error}}{I_{phN}} \ [\Omega]$$

Stator inductance

For the stator inductance (L_s) identification purposes, a sinusoidal measurement voltage is applied to the motor. During the L_S measurement, the voltage control is enabled. The frequency and amplitude of the sinusoidal voltage are obtained before the actual measurement, during the tuning process. The tuning process begins with a 0-V amplitude and the *F start* frequency, which are applied to the motor. The amplitude is gradually increased by *Ud inc* up to a half of the DC-bus voltage (DCbus/2), until *Id ampl* is reached. If *Id ampl* is not reached even with the DCbus/2 and *F start*, the frequency of the measuring signal is gradually decreased by *F dec* down to *F min* again, until *Id ampl* is reached. If *Id ampl* is still not reached, the measurement continues with DCbus/2 and *F min*. The tuning process is shown in Figure 43.

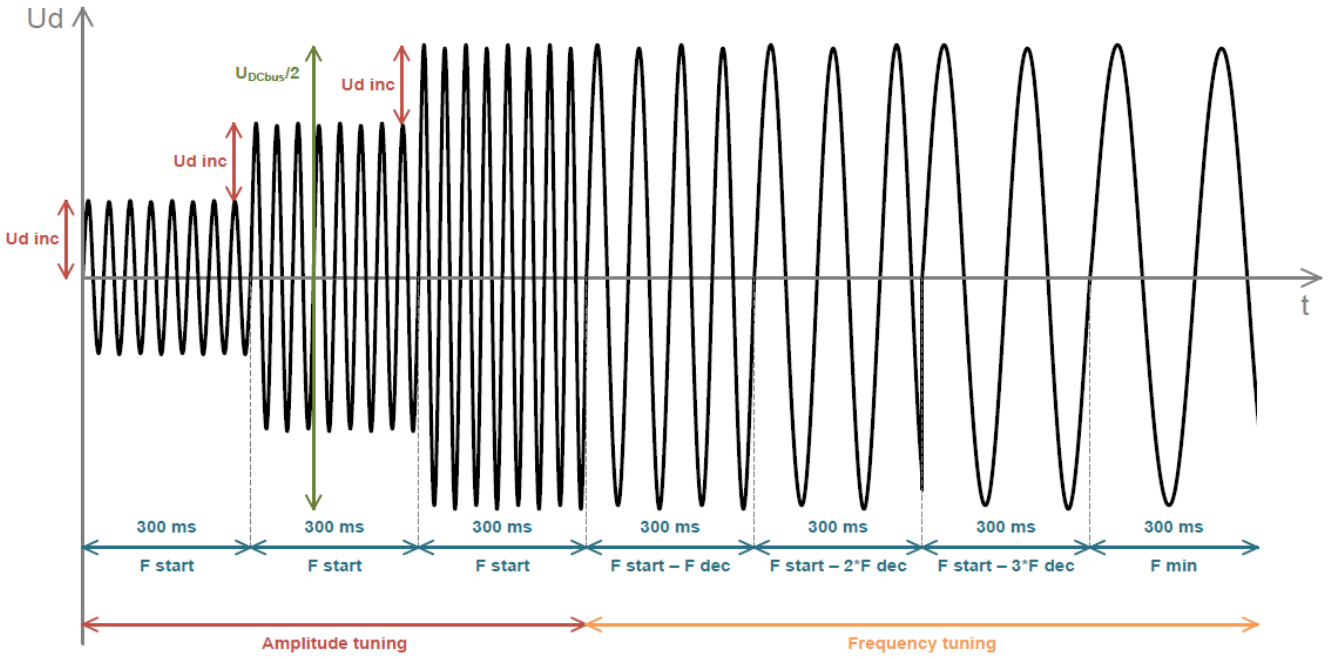

Figure 43. Tuning Ls measuring signal

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

$$Z_{RL} = \frac{U_d}{I_{d\ ampl}}\ [\Omega]$$

$$X_{Ls} = \sqrt{Z_{RL}^2 - R_s^2}\ [\Omega]$$

$$L_s = \frac{X_{Ls}}{2\pi f}\ [\Omega]$$

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

BEMF constant measurement

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

$$K_e = \frac{U_{BEMF}}{\omega_{el}}\ [\Omega]$$

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

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

Number of pole-pair assistant

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

Mechanical parameters measurement

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

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

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

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

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

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

The final mechanical parameter estimation can be calculated by integrating:

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

Te moment of inertia is:

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

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

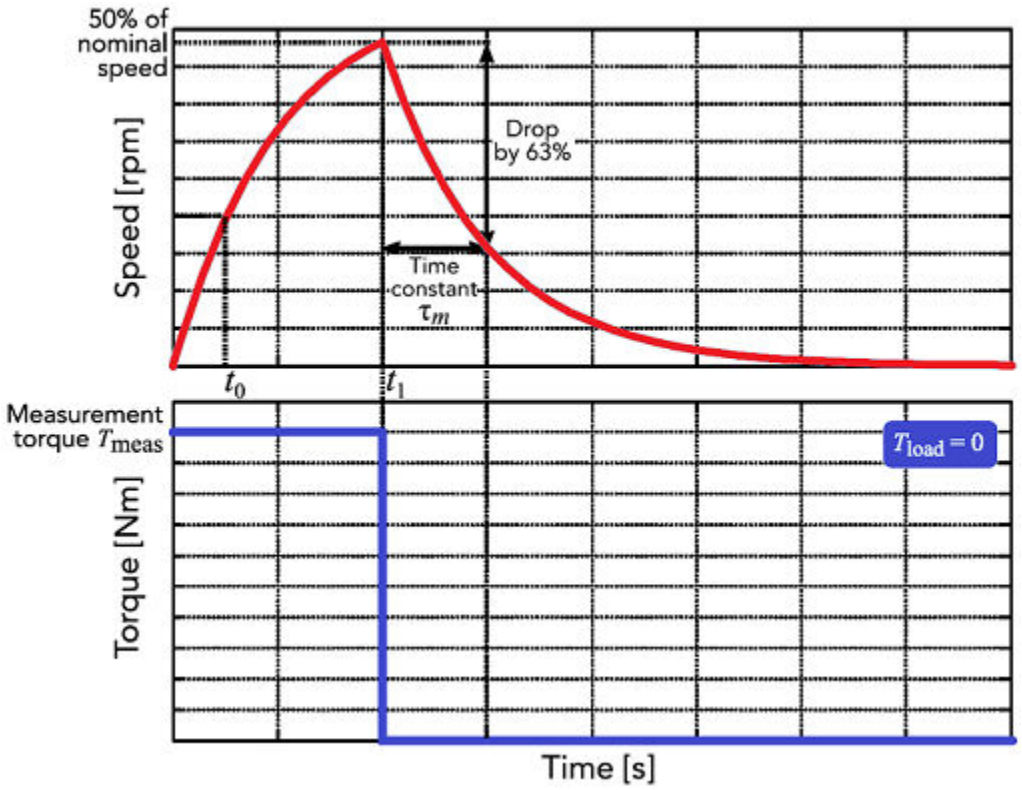

Figure 44. PMSM identification tab

11.1 PMSM electrical and mechanical parameters measurement process

The motor identification process can be controlled and set up in the MCAT “Motor Identif” tab, which is shown in Figure 45. To measure your own motor, follow these steps:

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

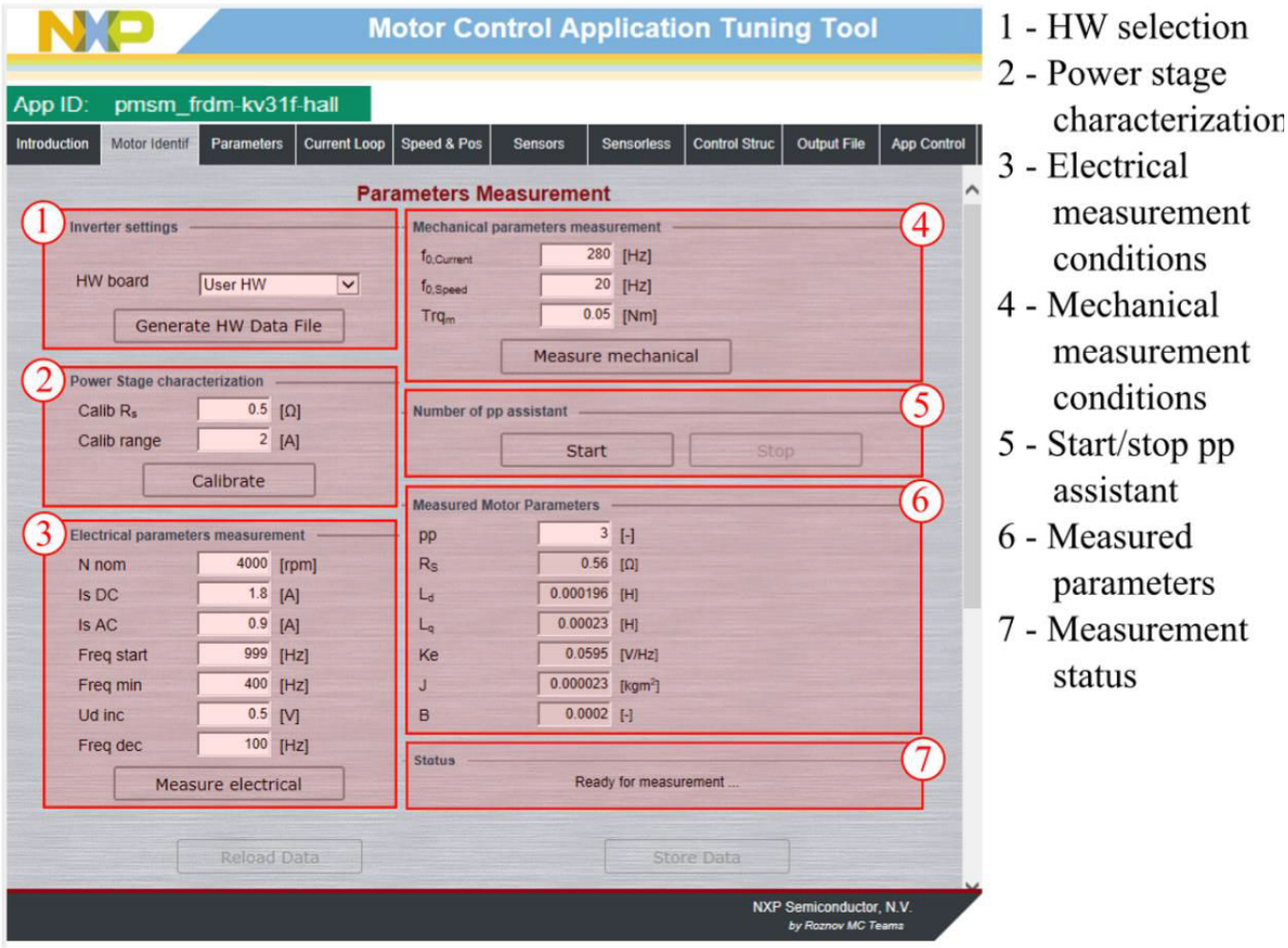

Figure 45. PMSM identification tab

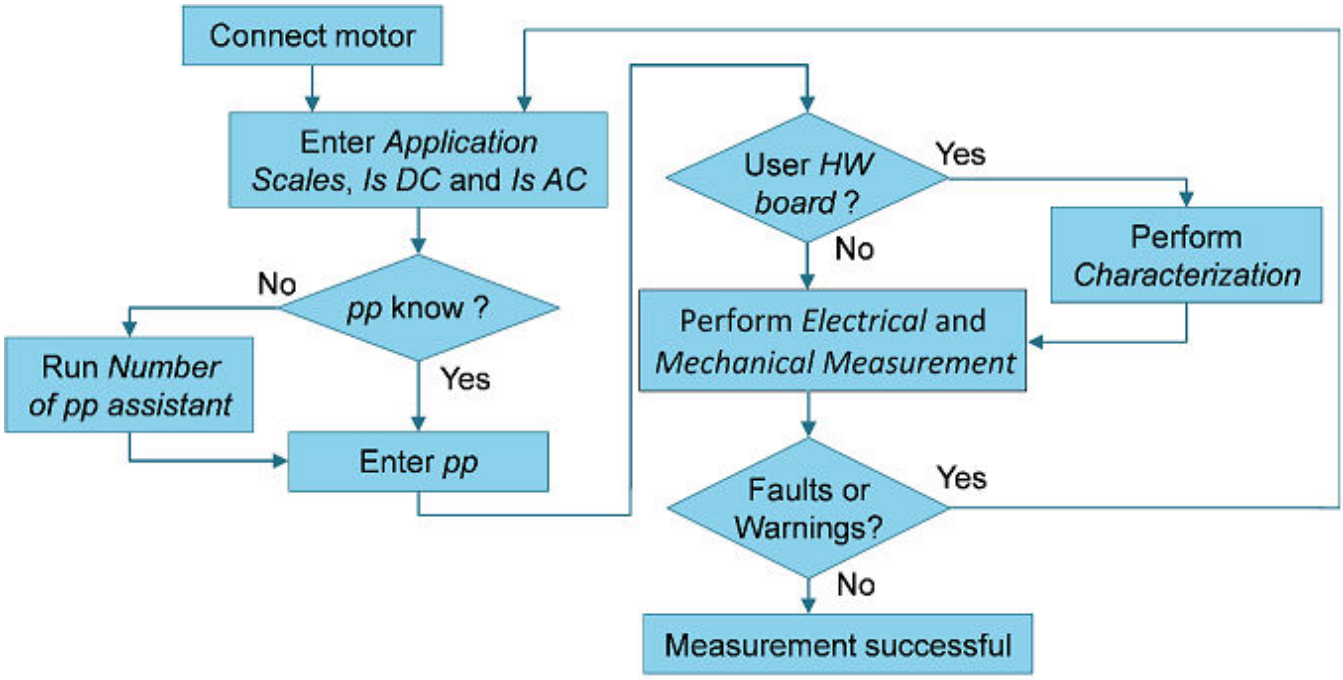

Figure 46. Measurement process diagram

During the measurement, faults and warnings may occur. Do not confuse these faults for the application faults, such as overcurrent, undervoltage, and so on. The list of these faults with their description and possible troubleshooting is shown in Table 23.

Table 23. Measurement faults and warnings

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

Table continues on the next page...

Table 23. Measurement faults and warnings (continued)

Fault no.	Fault description	Fault reason	Troubleshooting
6	Mechanical measurement timeout	The mechanical measurement takes too long.	Repeat the measurement process with a different setup.

11.2 Initial configuration setting and update

1. Open the PMSM control application FreeMASTER project containing the dedicated MCAT plug-in module.
2. Select the “Parameters” tab.
3. Leave the measured motor parameters or specify the parameters manually. The motor parameters can be obtained from the motor data sheet or using the PMSM parameters measurement procedure described in *PMSM Electrical Parameters Measurement* (document AN4680). All parameters provided in Table 24 are accessible. The motor inertia J expresses the overall system inertia and can be obtained using a mechanical measurement. The J parameter is used to calculate the speed controller constant. However, the manual controller tuning can also be used to calculate this constant.

Table 24. MCAT motor parameters

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

4. Set the hardware scales—the modification of these two fields is not required when a reference to the standard power stage board is used. These scales express the maximum measurable current and voltage analog quantities.
5. Check the fault limits—these fields are not accessible in the “Basic” mode and are calculated using the motor parameters and hardware scales (see Table 25).

Table 25. Fault limits

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

Table continues on the next page...

Table 25. Fault limits (continued)

Parameter	Units	Description	Typical range
N over	[rpm]	Trigger value at which the overspeed fault is detected	N nom ~ N max
N min	[rpm]	Minimal actual speed value for the sensorless control	(0.05~0.2) *N max

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

Table 26. Application scales

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

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

11.3 Control structure modes

1. Select the scalar control by clicking the “DISABLED” button in the “Scalar Control” section. The button color changes to red and the text changes to “ENABLED”.
2. Turn the application switch on. The application state changes to “RUN”.
3. Set the required frequency value in the “Freq_req” field; for example, 15 Hz in the “Scalar Control” section. The motor starts running (Figure 47).

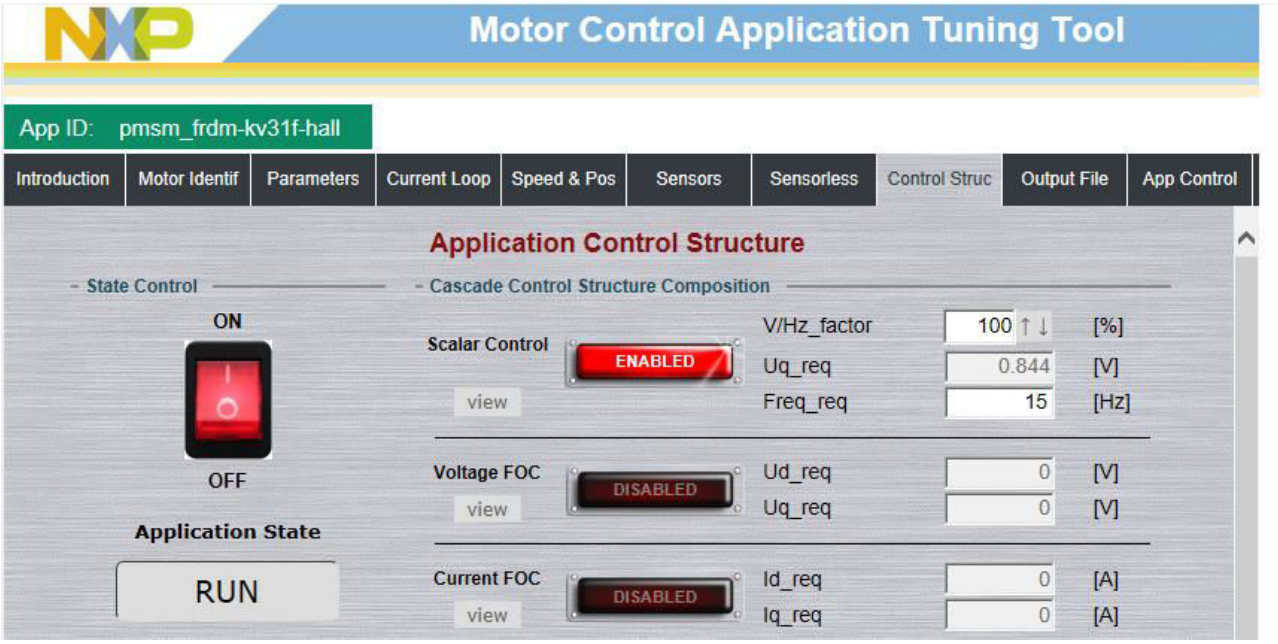

Figure 47. MCAT scalar control

4. Select the “Phase Currents” recorder from the “Scalar & Voltage Control” FreeMASTER project tree.
5. The optimal ratio for the V/Hz profile can be found by changing the V/Hz factor directly or using the “UP/DOWN” buttons. The shape of the motor currents should be close to a sinusoidal shape (Figure 48).

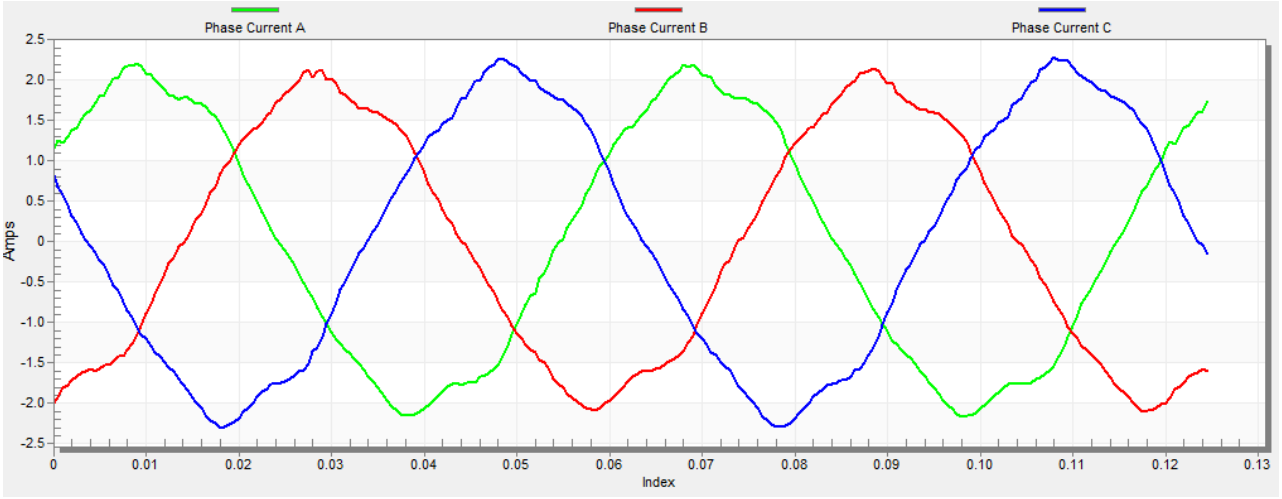

Figure 48. Phase currents

6. Select the “Position” recorder to check the observer functionality. The difference between the “Position Electrical Scalar” and the “Position Estimated” should be minimal (see Figure 49) 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.

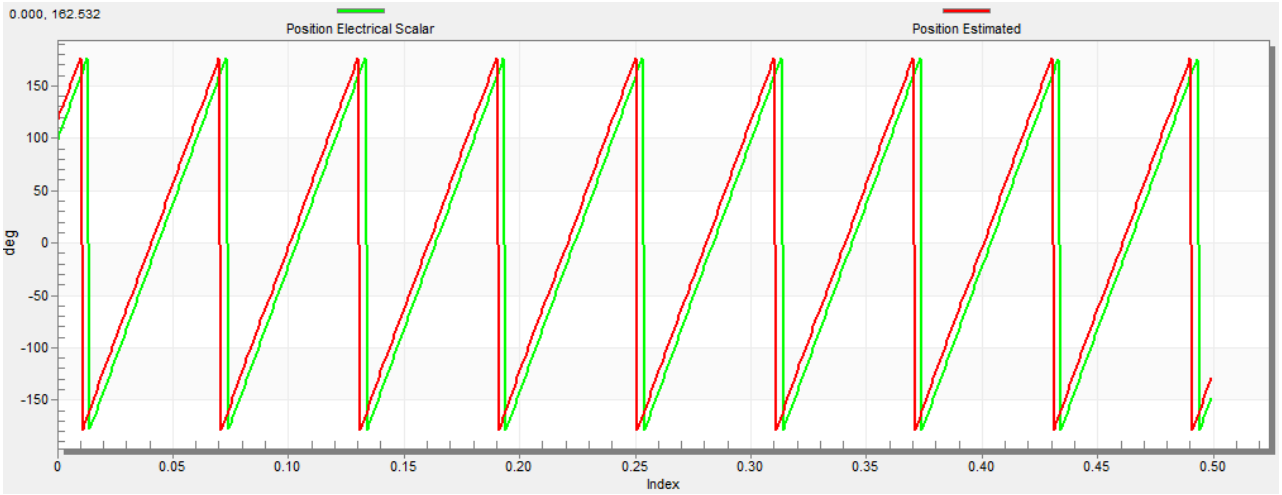

Figure 49. Generated and estimated positions

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

11.4 Alignment tuning

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

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

11.5 Current loop tuning

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

1. Lock the motor shaft.
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 how fast the loop response is 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 “Current FOC”, set the “Iq_req” field to a very low value (for example 0.01), and set the required step in “Id_req”. The control loop response is shown in the recorder.
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 50).

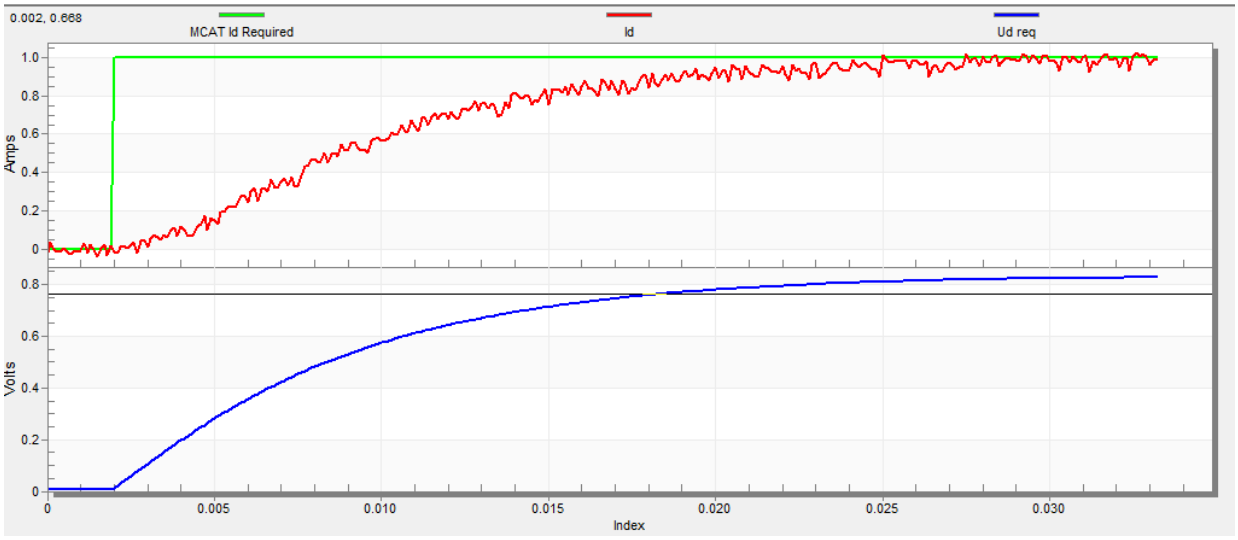

Figure 50. Slow step response of the Id current controller

   - The loop bandwidth (400 Hz) is optimal and the response time of the Id current is sufficient (see Figure 51).

Figure 51. Optimal step response of the Id current controller

- The loop bandwidth is high (700 Hz) and the response time of the Id current is very fast, but with oscillation and overshoot (see Figure 52).

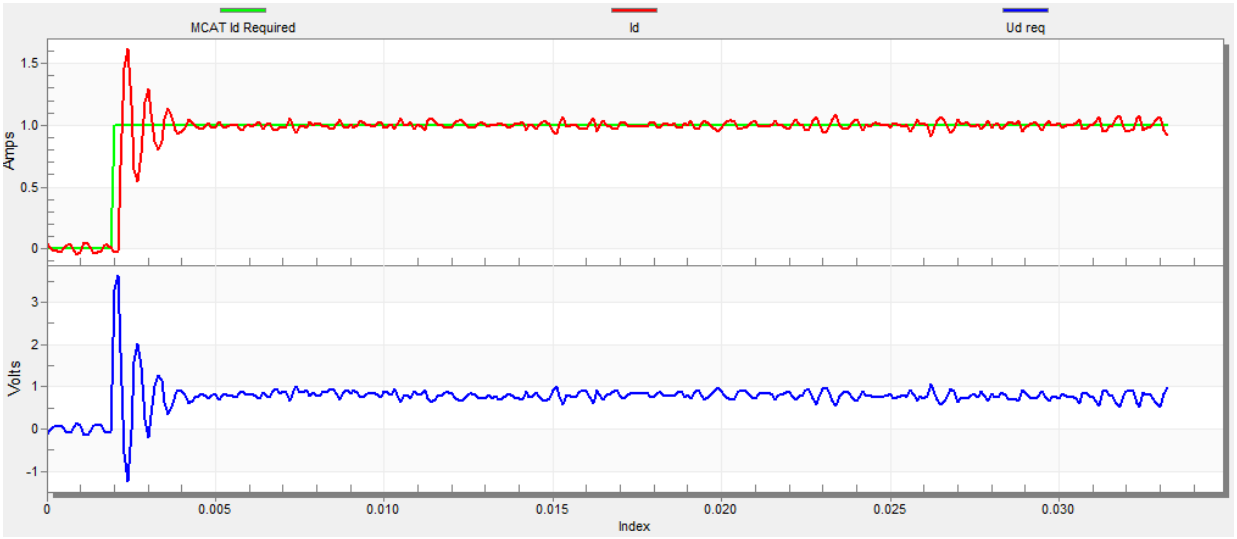

Figure 52. Fast step response of the Id current controller

11.6 Speed ramp tuning

1. The speed command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) which express the motor acceleration and deceleration per second. If the increments are very high, they can cause an overcurrent fault during acceleration and an overvoltage fault during deceleration. In the “Speed” scope, you can see whether the “Speed Actual Filtered” waveform shape equals the “Speed Ramp” profile.
2. The increments are common for the scalar and speed control. The increment fields are in the “Speed & Pos” tab and accessible in both tuning modes. Clicking the “Update Target” button applies the changes to the MCU. An example speed profile is shown in Figure 53. The ramp increment down is set to 500 rpm/sec and the increment up is set to 3000 rpm/sec.
3. The start-up ramp increment is in the “Sensorless” tab and its value is usually higher than that of the speed loop ramp.

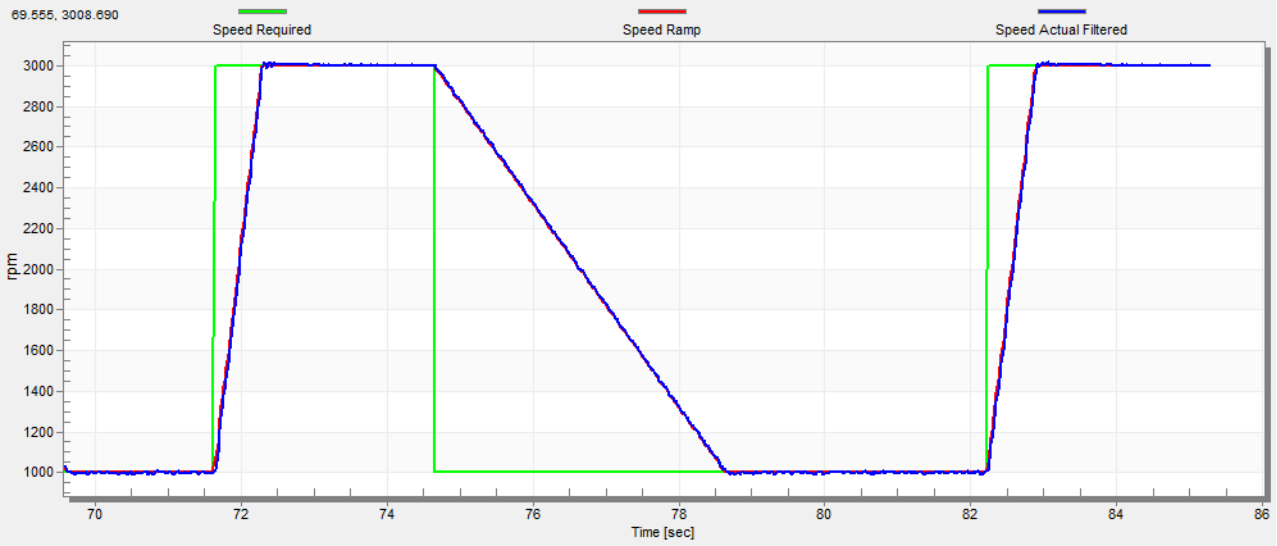

Figure 53. Speed profile

11.7 Open loop startup

1. The start-up process can be tuned by a set of parameters located in the “Sensorless” tab. Two of them (ramp increment and current) are accessible in both tuning modes. The start-up tuning can be processed in all control modes besides the scalar control. Setting the optimal values results in a proper motor startup. An example start-up state of low-dynamic drives (fans, pumps) is shown in Figure 54.
2. Select the “Startup” recorder from the FreeMASTER project tree.
3. Set the start-up ramp increment typically to a higher value than the speed-loop ramp increment.
4. Set the start-up current according to the required start-up torque. For drives such as fans or pumps, the start-up torque is not very high and can be set to 15 % of the nominal current.
5. Set the required merging speed—when the open-loop and estimated position merging starts, the threshold is mostly set in the range of 5 % ~ 10 % of the nominal speed.
6. Set the merging coefficient—in the position merging process duration, 100 % corresponds to a half of an electrical revolution. The higher the value, the faster the merge. Values close to 1 % are set for the drives where a high start-up torque and smooth transitions between the open loop and the closed loop are required.
7. Click the “Update Target” button to apply the changes to the MCU.
8. Switch to the “Control Structure” tab, and enable the “Speed FOC”.
9. Set the required speed higher than the merging speed.
10. Check the start-up response in the recorder.
11. Tune the start-up parameters until you achieve an optimal response.
12. If the rotor does not start running, increase the start-up current.
13. If the merging process fails (the rotor is stuck or stopped), decrease the start-up ramp increment, increase the merging speed, and set the merging coefficient to 5 %.

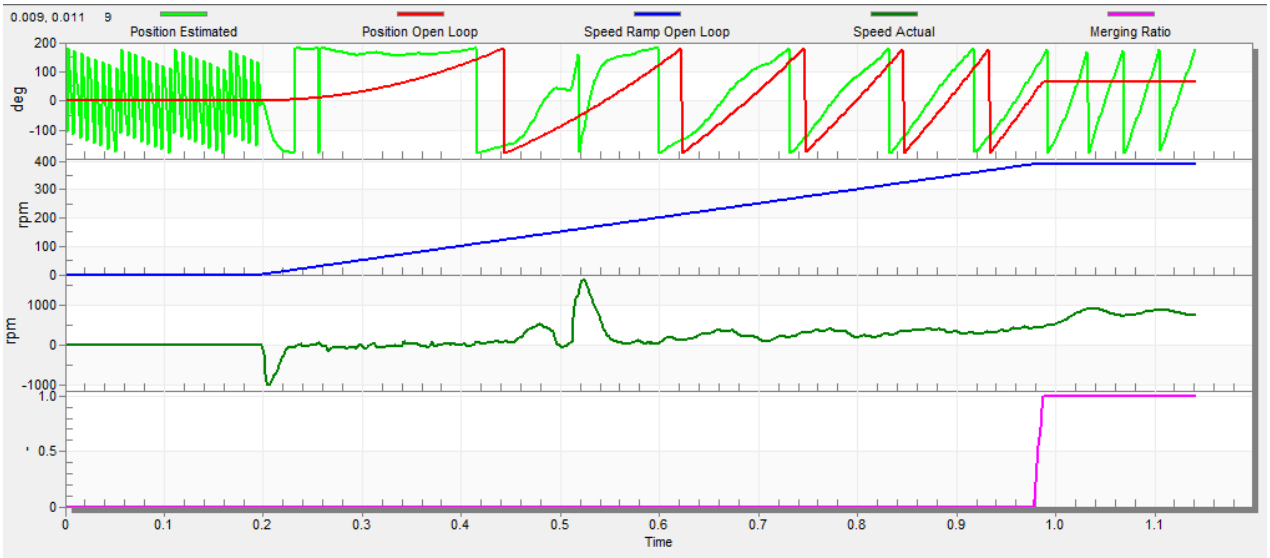

Figure 54. Motor startup

11.8 BEMF observer tuning

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

11.9 Speed PI controller tuning

The motor speed control loop is a first-order function with a mechanical time constant that depends on the motor inertia and friction. If the mechanical constant is available, the PI controller constants can be tuned using the loop bandwidth and attenuation. Otherwise, the manual tuning of the P and I portions of the speed controllers is available to obtain the required speed response (see the example response in Figure 55). There are dozens of approaches to tune the PI controller constants. The following steps provide an approach to set and tune the speed PI controller for a PM synchronous motor:

1. Select the “Speed Controller” option from the FreeMASTER project tree.
2. Select the “Speed & Pos” tab.
3. Check the “Manual Constant Tuning” option—that is, the “Bandwidth” and “Attenuation” fields are disabled and the “SL_Kp” and “SL_Ki” fields are enabled.
4. Tune the proportional gain:
   - Set the “SL_Ki” integral gain to 0.
   - Set the speed ramp to 1000 rpm/sec (or higher).
   - Switch to the “Control Structure” tab and run the motor at a convenient speed (about 30 % of the nominal speed).

- Set a step in the required speed to 40 % of N_{nom}.
- Switch back to the “Speed loop” tab.
- Adjust the proportional gain “SL_Kp” until the system responds to the required value properly and without any oscillations or excessive overshoot:
  - If the “SL_Kp” field is set low, the system response is slow.
  - If the “SL_Kp” field is set high, the system response is tighter.
  - When the “SL_Ki” field is 0, the system most probably does not achieve the required speed.
  - Click the “Update Target” button to apply the changes to the MCU.

5. Tune the integral gain:
   - Increase the “SL_Ki” field slowly to minimize the difference between the required and actual speeds to 0.
   - Adjust the “SL_Ki” field such that you do not see any oscillation or large overshoot of the actual speed value while the required speed step is applied.
   - Click the “Update Target” button to apply the changes to the MCU.
6. Tune the loop bandwidth and attenuation until the required response is received. The example waveforms with the correct and incorrect settings of the speed loop parameters are shown in the following figures:
   - The “SL_Ki” value is low and the “Speed Actual Filtered” does not achieve the “Speed Ramp” (see Figure 55).

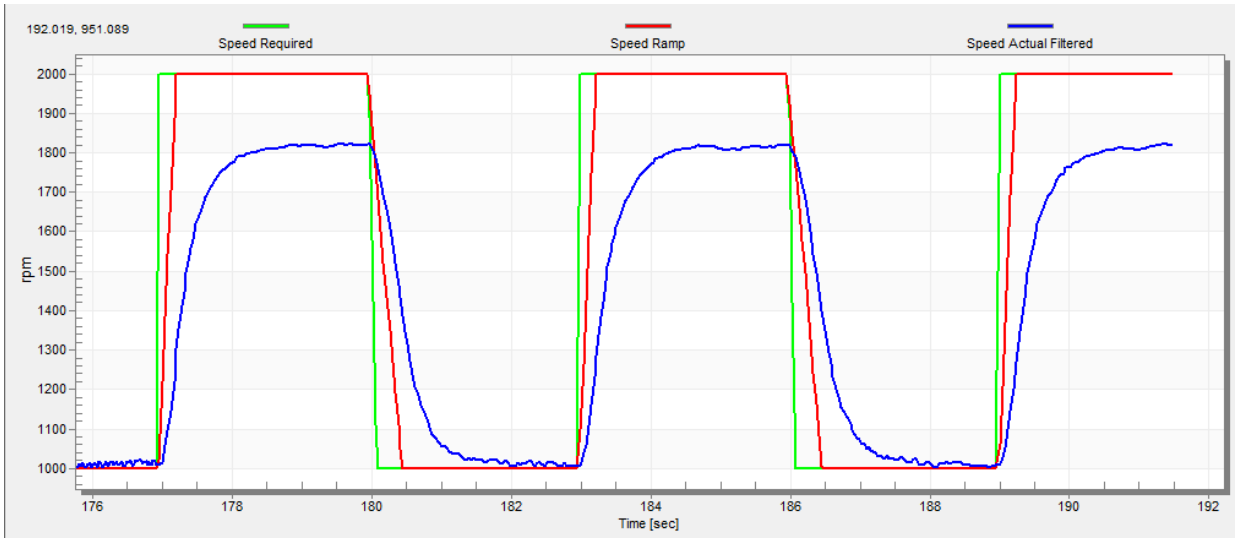

Figure 55. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved

   - The “SL_Kp” value is low, the “Speed Actual Filtered” greatly overshoots, and the long settling time is unwanted (see Figure 56).

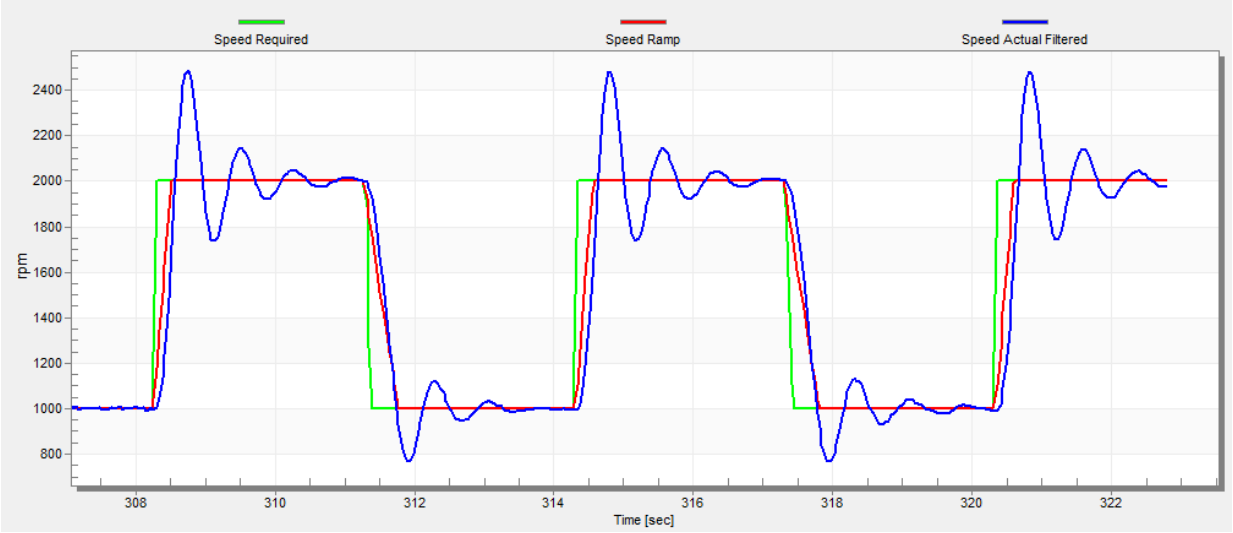

Figure 56. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

- The speed loop response has a small overshoot and the “Speed Actual Filtered” settling time is sufficient. Such response can be considered optimal (see Figure 57).

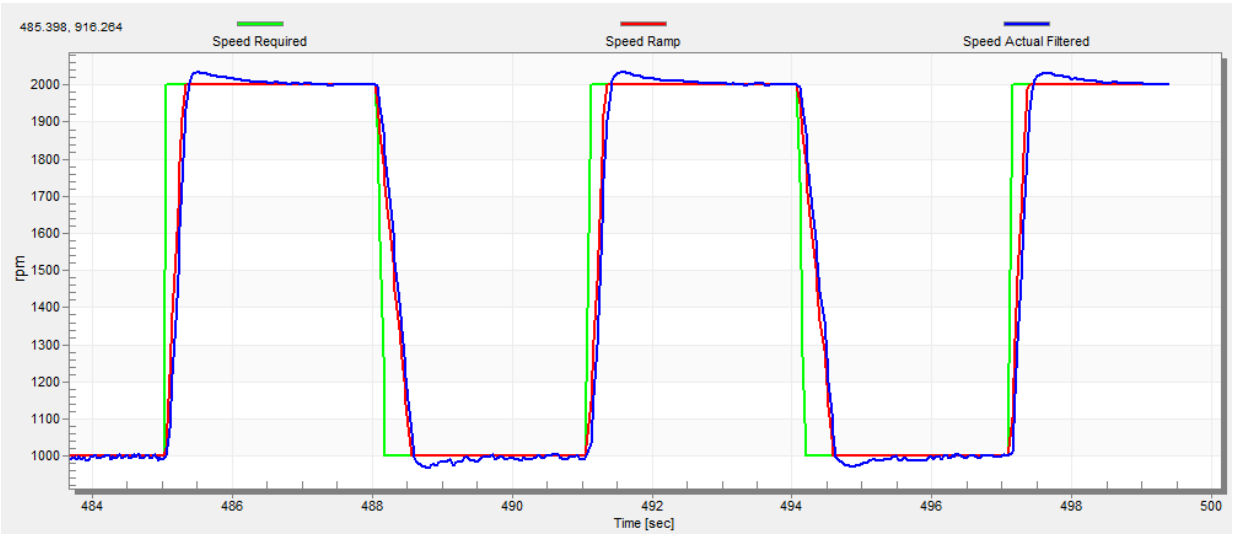

Figure 57. Speed controller response—speed loop response with a small overshoot

Chapter 12
Conclusion

This application note describes the implementation of a sensorless field-oriented control of the 3-phase PMSM using 32-bit Kinetis V and Kinetis E series devices and the High-Voltage Platform, Tower System, and Freedom development platforms. The hardware-dependent part of the control software is described in Hardware setup. The motor control application timing is described in MCU features and peripheral settings and the peripheral initialization in Motor-control peripheral initialization. The motor user interface and remote control using FreeMASTER are as follows. The motor parameters identification theory and the identification algorithms are described in Identifying parameters of user motor using MCAT.

Chapter 13
Acronyms and abbreviations

Table 27.

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

Chapter 14
References

These references are available on www.nxp.com:

1. *Sensorless PMSM Field-Oriented Control* (document DRM148).
2. *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document AN4642).
3. *Sensorless PMSM Field-Oriented Control on Kinetis KV* (document AN5237).
4. *PMSM Sensorless Application Package User's Guide* (document PMSMSAPUG)

Chapter 15
Useful links

1. PMSM Control Reference Design www.nxp.com/motorcontrol_pmsm
2. BLDC Control Reference Design www.nxp.com/motorcontrol_bldc
3. ACIM Control Reference Design www.nxp.com/motorcontrol_acim
4. FRDM-MC-PMSM Freedome Development Platform
5. TWR-MC-LV3PH Tower Development Platform
6. HVP-MC3PH High-Voltage Platform
7. MCUXpresso IDE - Importing MCUXpresso SDK
8. MCUXpresso SDK Builder (SDK examples in several IDEs) https://mcuxpresso.nxp.com/en/welcome

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 02/2020

Document identifier: 3PPMSMCKVUG

arm