

Sensorless BLDC Control on Kinetis KV and KE

By: Martin Sebest

1. Introduction

This application note describes the implementation of the sensorless Motor Control Reference Solution Package (MCRSP) software for a 3-phase Brushless DC motor (BLDC), running on 32-bit Kinetis V and E series MCUs. The sensorless control software itself and the BLDC control theory in general is described in *3-Phase BLDC Sensorless Motor Control Application* (document [DRM144](#)). The NXP Freedom board (FRDM-MC-LVPMSM), Tower System modular development platform module (TWR-MC-LV3PH), and High-Voltage Platform power stages (HVP-MC3PH) are used as hardware platforms for the BLDC control reference solution.

The hardware-dependent part of the sensorless control software is addressed as well, including detailed peripheral setup and the Motor Control Peripheral Drivers (MCDRV). The last part of this document introduces and explains the user interface represented by the Motor Control Application Tuning (MCAT) tool page based on FreeMASTER run-time debugging tool. These tools represent a simple and user-friendly way of algorithm tuning, software control, debugging, and diagnostics.

Contents

1.	Introduction	1
2.	Development Platforms	2
2.1.	FRDM-MC-LVBLDC	2
2.2.	TWR-MC-LV3PH	3
2.3.	HVP-MC3PH	3
3.	MCU Features and Peripheral Settings	4
3.1.	KV1x family	4
3.2.	KV3x family	8
3.3.	KV4x family	12
3.4.	KV5x family	15
3.5.	KE1xZ family	18
3.6.	KE1xF family	22
4.	Motor Control Peripheral Drivers	25
4.1.	Motor Control Peripheral Drivers initialization	25
4.2.	Motor Control Peripheral Drivers API	26
5.	Tuning and Controlling the Application	27
5.1.	BLDC sensorless application control and tuning using MCAT tool	29
6.	Conclusion	41
7.	Acronyms and Abbreviations	41
8.	References	41
9.	Revision History	42

2. Development Platforms

There are these three standard NXP power stages:

- FRDM-MC-LVBLDC
- TWR-MC-LV3PH
- HVP-MC3PH

2.1. FRDM-MC-LVBLDC

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

The FRDM-MC-LVBLDC board has a power-supply input voltage of 9–15 V DC with a reverse polarity protection circuitry. An auxiliary power supply of 5.5 V DC is available to provide power to the FRDM MCU boards. The output current reaches up to 5 A RMS. The inverter is realized by the 3-phase bridge inverter (six MOSFETs) and the 3-phase MOSFET gate driver. Analog quantities (such as 3-phase motor back-EMF voltage, DC-Bus voltage, and DC-Bus current) are measured on this board. There is also an interface for speed/position sensors (Encoder Hall). The block diagram of a complete Freedom motor-control development kit is shown in this figure:

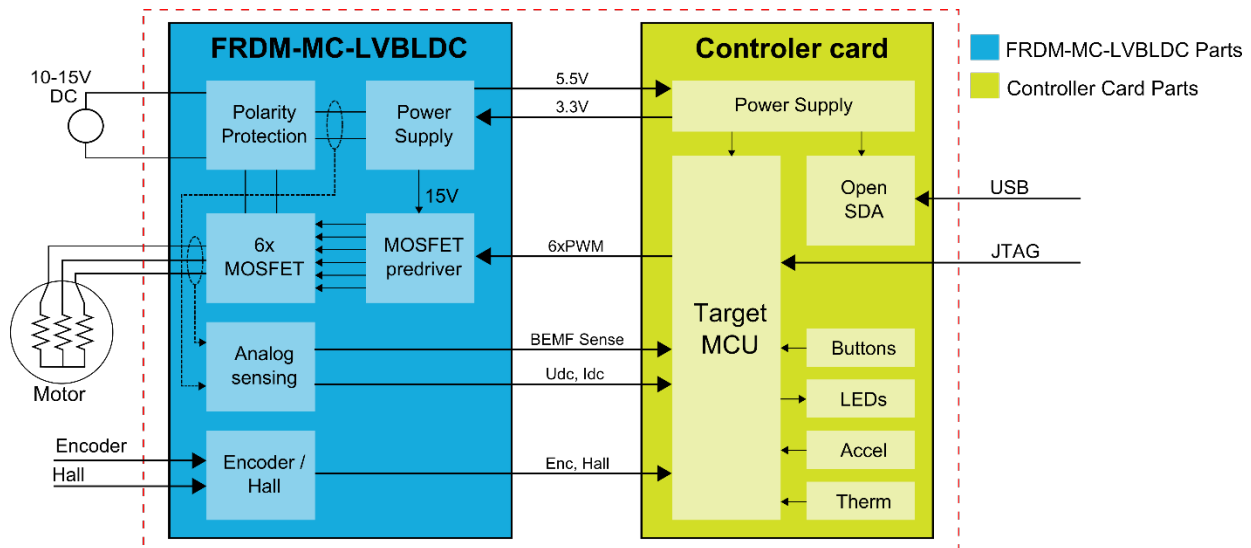


Figure 1. Freedom motor-control development platform block diagram

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

2.2. TWR-MC-LV3PH

This module effectively turns a Tower System development module into a complete motor-control reference design, compatible with the existing Tower System Kinetis V and Kinetis E modules.

This module provides all necessary feedback signals to drive PMSM and BLDC motors.

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

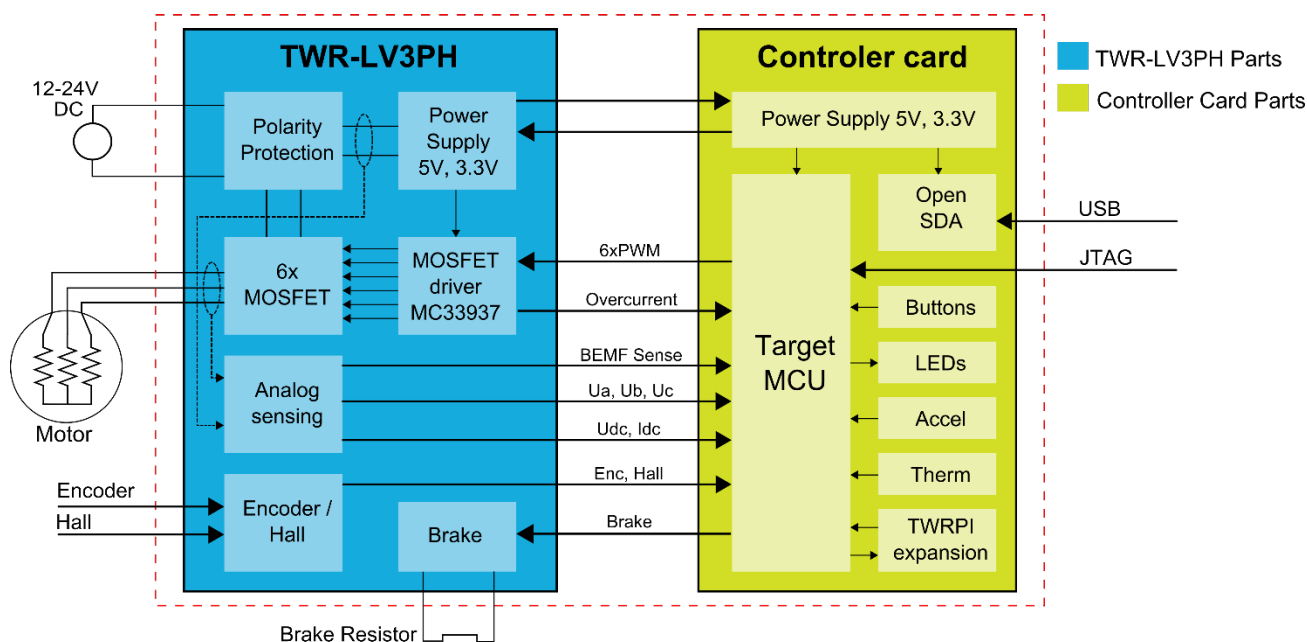


Figure 2. Tower System motor-control development platform block diagram

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

2.3. HVP-MC3PH

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

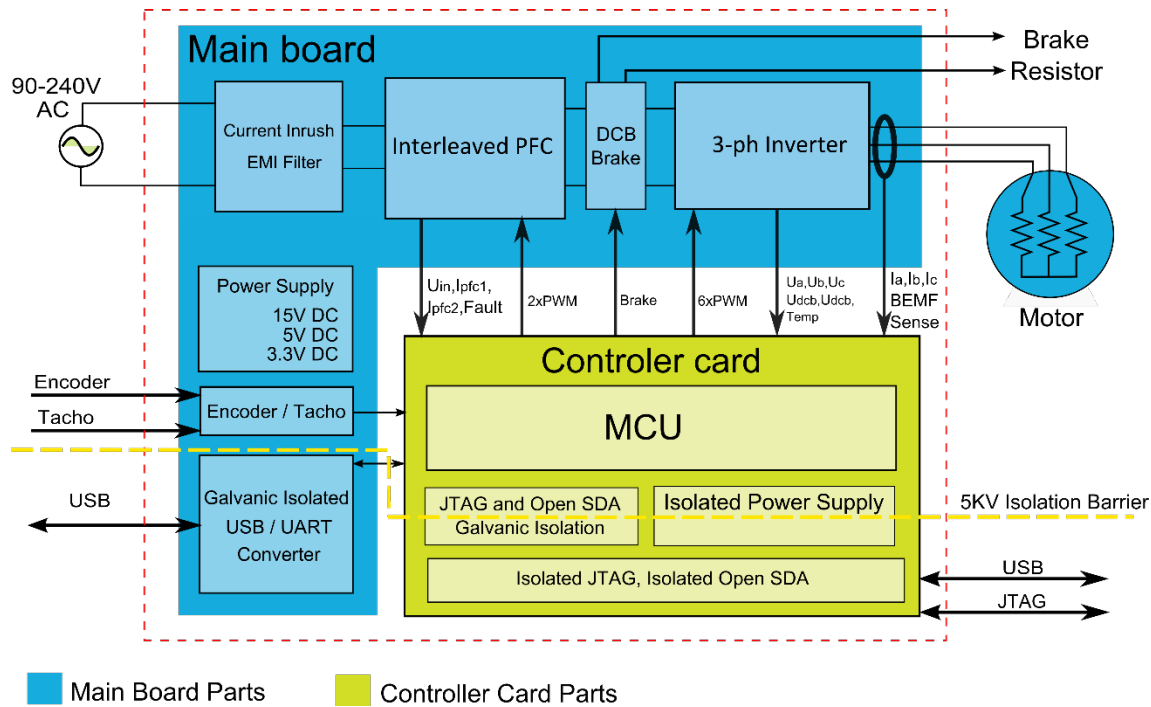


Figure 3. High-Voltage Development Platform block diagram

The HVP-MC3PH power stage does not require any complicated setup and there is only one way to connect a daughter board to the HVP. See the user's guides for the HVP power stage (document [HVP MC3PHUG](#)) and for your version of MCRSP (document [BLDC MCRSPUG](#)).

NOTE

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

3. MCU Features and Peripheral Settings

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

3.1. KV1x family

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

3.1.1. Hardware timing and synchronization

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

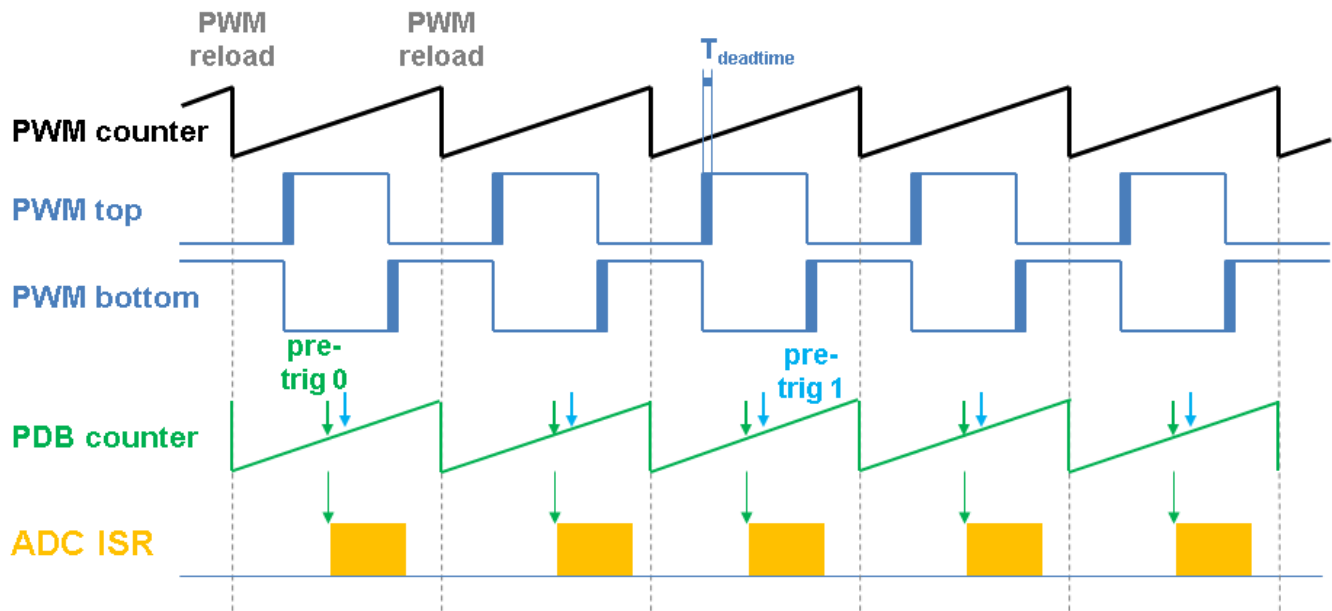


Figure 4. Hardware timing and synchronization on KV11Z and KV10Z

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized at the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated on the **PWM reload** event, which triggers the PDB (resets the **PDB counter**).
- The PDB generates the first pre-trigger for the first ADC sample with a delay of approximately $T_{\text{PWM}} / 2$. This delay ensures correct DC-bus current sampling.
- When the conversion of the first ADC sample is completed, the **ADC ISR** is entered and fast loop control function is calculated.
- The PDB uses the back-to-back mode to automatically generate the **pre-trig 1** (for next quantity measurement) immediately after the first conversion is completed.

3.1.2. Peripheral settings

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

3.1.2.1. PWM generation—FTM0

- The FTM is clocked from the 75-MHz System clock.

- Only six channels are used, the other two are masked in the OUTMASK register.
- The channels (0+1, 2+3, and 4+5) are combined into pairs, with each pair running in a complementary mode.
- The Fault mode is enabled for each combined pair with automatic fault clearing (the PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as a time needed for the FTM to count from CNTIN to MOD. By default, $CNTIN = -MODULO / 2 = -1875$ and $MOD = MODULO / 2 - 1 = 1874$. Because the FTM is clocked from the 75 MHz System clock, it takes 0.00005 s (20 kHz).
- Inserting dead time is enabled for each combined pair. The dead time length is calculated as $System\ clock\ 75\ MHz \times T_{deadtime}$. The dead time varies among platforms.
- The FTM generates a trigger to PDB at counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.1.2.2. Analog sensing—ADC0, ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 25 MHz Bus clock divided by 1 = 25 MHz.
- For ADC calibration purposes, the ADC clock is set to 3.125 MHz. Continuous conversion and averaging with 32 samples are enabled in the SC3 register. After the calibration is done, the SC register is filled with its default values and the clock is set back to 12.5 MHz.
- Both ADCs are triggered from the PDB pre-triggers.
- An interrupt that serves for fast-loop algorithm calculation is generated after the first conversion is completed.

3.1.2.3. PWM and ADC synchronization—PDB0

- Unlike FTM, the PDB is clocked from the Bus clock, which is 3× slower than the System clock (used for FTM). Therefore the modulo value at PDB is divided by 3.
- The PDB is triggered from the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated in the middle of one PWM period. This is setup in PDBx_CHnDLY register. $PDBx_CHnDLY = PWM_MODULO / 2$, but the PDB is clocked from Bus clock, which is 3x slower than System clock (used for FTM), therefore $PDBx_CHnDLY = PWM_MODULO / 6$.
- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC. For more information about PDB error sequence handling see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document [AN4822](#)).

3.1.2.4. Time event, forced commutation control—FTM1

- FTM1 is used for forced commutation control.
- FTM1 is set as free running counter and clocked from the System clock. Timer clock prescaler is setup to 128.
- The FTM1 counts from 0 to 0xffff.
- The output compare interrupt is enabled and generated when counter equals to value register.
- Value register is periodically updated in fast control loop function and FTM1 output compare interrupt is not invoked until error in BLDC commutation process appears.
- If error in BLDC commutation process appears, the forced commutation is performed in FTM1 interrupt service routine.

3.1.2.5. Slow-loop interrupt generation—FTM2

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

3.1.2.6. Communication with MC33937 MOSFET driver—SPI

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

3.1.3. Peripheral settings differences among platforms

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

Table 1. KV10 and KV11 platform differences

Peripheral	Feature	Platform		
		FRDM	Tower System	HVP
FTM0	PWM polarity	high sides active high low sides active high	high sides active low low sides active high	high sides active high low sides active high
	Dead time	0.84 μs	0.84 μs	0.84 μs
SPI	Driver on SPI	No	Yes	No
PDB	Pre-trigger 0 delay	0.1 ms	0.1 ms	0.1 ms

3.1.4. CPU load and memory usage

The following information apply to the application built using IAR® Embedded Workbench® IDE. [Table 4](#) shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.8 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 and slow loop (speed loop) frequency. In this case, it applies to the fast loop of 20 kHz and the slow loop of 1 kHz. The total CPU load is calculated according to these equations:

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

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

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

where:

CPU_{fast}	—CPU time consumed by the fast loop
$cycles_{fast}$	—number of cycles consumed by the fast loop
f_{fast}	—frequency of the fast loop calculation (20 kHz)
f_{CPU}	—CPU frequency
CPU_{slow}	—CPU time consumed by the slow loop
$cycles_{slow}$	—number of cycles consumed by the slow loop
f_{slow}	—frequency of the slow loop calculation (1 kHz)
CPU_{total}	—total CPU load consumed by the motor control

Table 2. KV10 and KV11 CPU load and memory usage

—	KV10	KV11
CPU clock [MHz]	75	75
Fast Control Loop [cycles] (%)	1364 (36.4 %)	1171 (31.2 %)
Slow Control Loop [cycles] (%)	2686 (3.6 %)	2217 (2.9 %)
Total CPU load [%]	40.0 %	34.1 %
Flash usage [B]	15 678	15 856
RAM usage [B]	3 393	3 401

3.2. KV3x family

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

3.2.1. Hardware timing and synchronization

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

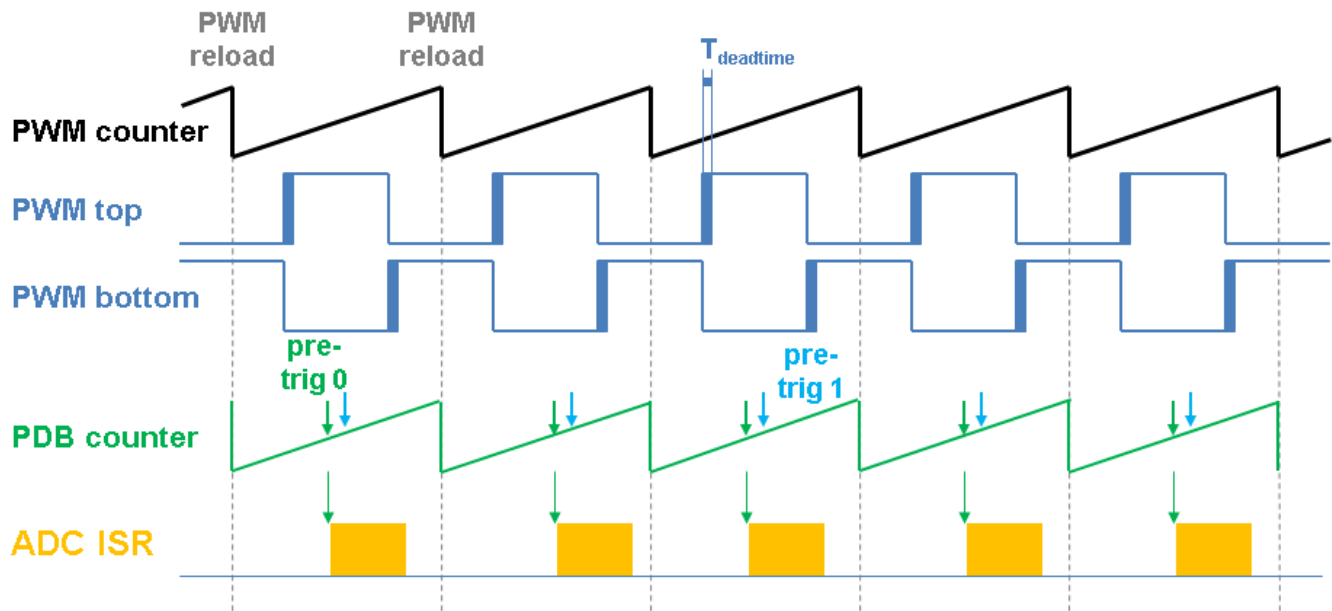


Figure 5. Hardware timing and synchronization on KV31F

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized at the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated on the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates the first pre-trigger for the first ADC sample with a delay of approximately $T_{PWM} / 2$. This delay ensures correct DC-bus current sampling.
- When the conversion of the first ADC sample is completed, the **ADC ISR** is entered and fast loop control function is calculated.
- The PDB uses the back-to-back mode to automatically generate the **pre-trig 1** (for next quantity measurement) immediately after the first conversion is completed.

3.2.2. Peripheral settings

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

3.2.2.1. PWM generation—FTM0

- The FTM is clocked from the 60-MHz Bus clock.
- Only six channels are used, the other two are masked in the OUTMASK register.

- Channels 0+1, 2+3, and 4+5 are combined in pairs and they are running in a complementary mode.
- The Fault mode is enabled for each combined pair with automatic fault clearing (PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as a time for the FTM to count from CNTIN to MOD. By default, CNTIN = -MODULO / 2 = -1500 and MOD = MODULO / 2 - 1 = 1499. The FTM is clocked from the 60 MHz Bus clock, so it takes 0.00005 s (20 kHz).
- Dead time insertion is enabled for each combined pair. The dead time length is calculated as System clock 60 MHz × T_{deadtime}. The dead time varies among platforms.
- The FTM generates a trigger for the PDB on counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.2.2.2. Analog sensing—ADC0 and ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 60 MHz Bus clock, divided by 4 = 15 MHz. The same ADC clock setup is used for ADC calibration purposes.
- Both ADCs are triggered by the PDB pre-triggers.
- An interrupt that serves for fast-loop algorithm calculation is generated when the first conversion is completed.

3.2.2.3. PWM and ADC synchronization—PDB0

- Like the FTM, the PDB is clocked from the 60 MHz Bus clock.
- The PDB is triggered by the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated in the middle of one PWM period. This is setup in PDBx_CHnDLY register. PDBx_CHnDLY = PWM_MODULO / 2.
- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC. For more information about PDB error sequence handling see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document [AN4822](#)).

3.2.2.4. Time event, forced commutation control—FTM1

- FTM1 is used for forced commutation control.
- FTM1 is set as free running counter and clocked from the Bus clock. Timer clock prescaler is setup to 128.
- The FTM1 counts from 0 to 0xffff.
- The output compare interrupt is enabled and generated when counter equals to value register.
- Value register is periodically updated in fast control loop function and FTM1 output compare

interrupt is not invoked until error in BLDC commutation process appears.

- If error in BLDC commutation process appears, the forced commutation is performed in FTM1 interrupt service routine.

3.2.2.5. Slow-loop interrupt generation—FTM2

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

3.2.2.6. Communication with MC33937 MOSFET driver—SPI

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

3.2.3. Peripheral settings differences among platforms

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

Table 3. KV31 platform differences

Peripheral	Feature	Platform		
		Freedom	Tower System	HVP
FTM0	PWM polarity	high sides active high low sides active high	high sides active low low sides active high	high sides active high low sides active high
	Dead time	1.05 μs	1.05 μs	1.05 μs
SPI	Driver on SPI	No	Yes	No
PDB	Pre-trigger 0 delay	0.1 ms	0.1 ms	0.1 ms

3.2.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. [Table 4](#) shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.8 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the *SysTick* timer. The CPU load depends on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 20 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated according to [Eq. 3](#).

Table 4. KV31 CPU load and memory usage

—	KV31
CPU clock [MHz]	120
Fast Control Loop [cycles] (%)	1189 (19.8 %)
Slow Control Loop [cycles] (%)	2105 (1.8 %)
Total CPU load [%]	21.6 %
Flash usage [B]	15 904
RAM usage [B]	3 409

3.3. KV4x family

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

3.3.1. Hardware timing and synchronization

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

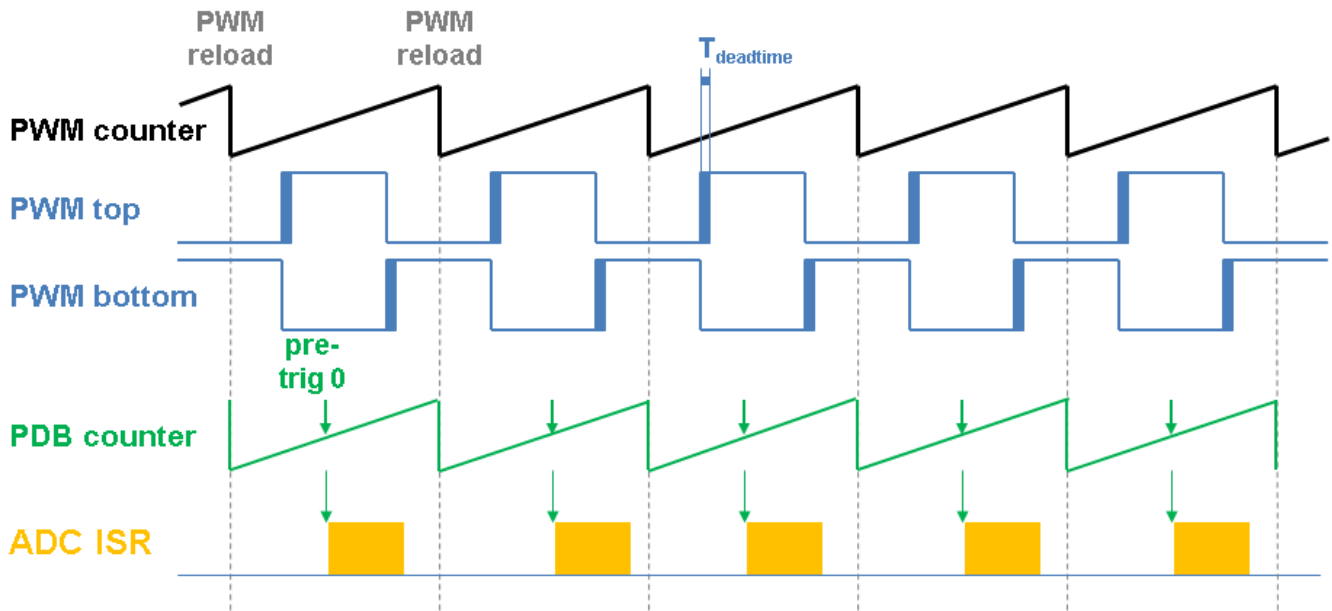


Figure 6. Hardware timing and synchronization on KV46F

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized at the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated on the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates the pre-trigger for the ADCA and ADCB (parallel sampling) conversion with

a delay of approximately $T_{PWM} / 2$. This delay ensures correct DC-bus current sampling.

- When the conversion of the all ADC samples is completed, the **ADC ISR** is entered and fast loop control function is calculated.

3.3.2. Peripheral settings

This section describes only the peripherals used for motor control. KV46F uses a 6-channel FlexTimer (FTM) to generate a 6-channel PWM, and two 12-bit cyclic SAR ADCs to measure the back-EMF voltage, DC-bus current, and DC-bus voltage. The FTM and ADC are synchronized via Programmable Delay Block (PDB). One channel from another independent FTM is used for slow-loop interrupt generation.

3.3.2.1. PWM generation—FTM0

- The FTM is clocked from the Fast Peripheral Clock 74 MHz Bus clock.
- Only six channels are used, the other two are masked in the OUTMASK register.
- Channels 0+1, 2+3, and 4+5 are combined in pairs and they are running in a complementary mode.
- The Fault mode is enabled for each combined pair with automatic fault clearing (PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as a time for the FTM to count from CNTIN to MOD. By default, $CNTIN = -MODULO / 2 = -1850$ and $MOD = MODULO / 2 - 1 = 1849$. The FTM is clocked from the 74-MHz Fast Peripheral Clock, so it takes 0.00005 s (20 kHz).
- Dead time insertion is enabled for each combined pair. The dead time length is calculated as $System\ clock\ 74\ MHz \times T_{deadtime}$. The dead time varies among platforms.
- The FTM generates a trigger for the PDB on counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.3.2.2. Analog sensing—ADC12

- The ADC12 wrapper contains two independent ADCs. The ADCs operate as 12-bit, single-ended converters. ADC12 operates in a triggered parallel mode (ADC0 and ADC1 convert $SAMPLE0 + SAMPLE8$ and $SAMPLE1 + SAMPLE9$ simultaneously). The ADC scan is triggered by the SYNC0 signal.
- The clock source for ADC12 is the 74-MHz Fast Peripheral clock divided by 6 = 12.3 MHz.
- Only $SAMPLE0$, $SAMPLE1$, $SAMPLE8$, and $SAMPLE9$ are enabled.
- The end-of-scan interrupt serves for fast-loop algorithm calculation is generated when the all conversions are completed.

3.3.2.3. PWM and ADC synchronization—PDB0

- Like the FTM, the PDB is clocked from the 74-MHz Fast Peripheral clock.
- The PDB is triggered by the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated in the middle of one PWM period. This is setup in PDBx_CHnDLY register. $PDBx_CHnDLY = PWM_MODULO / 2$.
- 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 PDB error sequence handling see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document [AN4822](#)).

3.3.2.4. Time event, forced commutation control—FTM1

- FTM1 is used for forced commutation control.
- FTM1 is set as free running counter and clocked from the Fast Peripheral clock. Timer clock prescaler is setup to 128.
- The FTM1 counts from 0 to 0xffff.
- The output compare interrupt is enabled and generated when counter equals to value register.
- Value register is periodically updated in fast control loop function and FTM1 output compare interrupt is not invoked until error in BLDC commutation process appears.
- If error in BLDC commutation process appears, the forced commutation is performed in FTM1 interrupt service routine.

3.3.2.5. Slow loop interrupt generation—FTM3

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

3.3.2.6. Communication with MC33937 MOSFET driver—SPI

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

3.3.3. Peripheral settings differences among platforms

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

Table 5. KV46 platform differences

Peripheral	Feature	Platform	
		Tower System	HVP
FTM0	PWM polarity	high sides active low low sides active high	high sides active high low sides active high
	Dead time	0.8 μ s	0.8 μ s
SPI	Driver on SPI	Yes	No
PDB	Pre-trigger 0 delay	0.1 ms	0.1 ms

3.3.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. [Table 6](#) shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.8 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the SysTick timer. The CPU load depends on the fast loop (current loop) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 20 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated according to [Eq. 3](#).

Table 6. KV46 CPU load and memory usage

—	KV46
CPU clock [MHz]	148
Fast Control Loop [cycles] (%)	826 (11.2 %)
Slow Control Loop [cycles] (%)	1543 (1.0 %)
Total CPU load [%]	12.2 %
Flash usage [B]	15 884
RAM usage [B]	3 361

3.4. KV5x family

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

3.4.1. Hardware timing and synchronization

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. The timing diagram is shown in this figure:

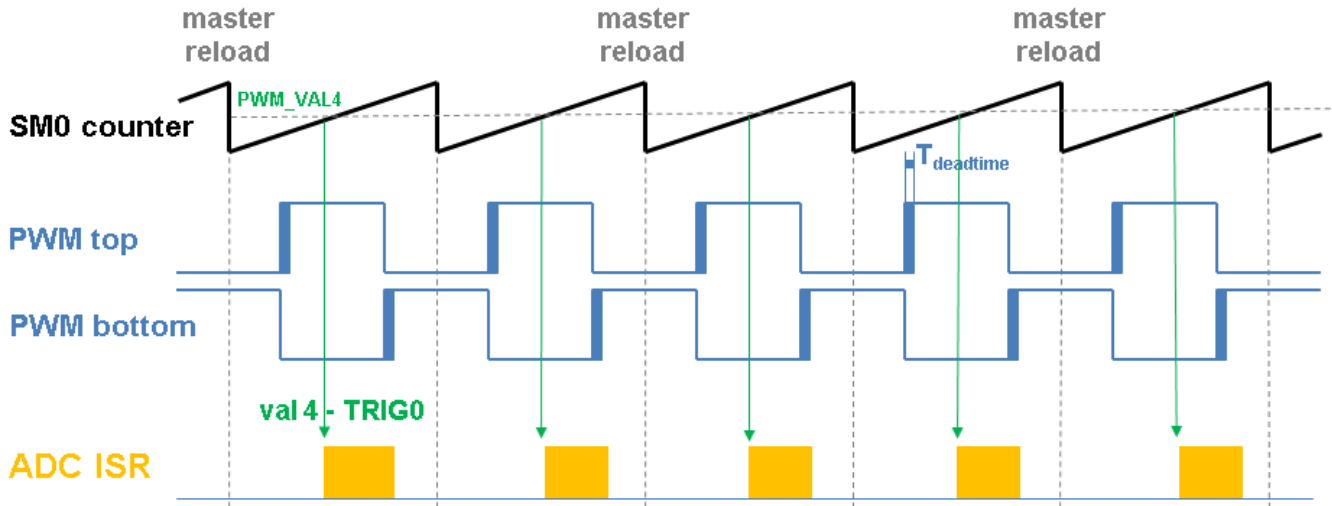


Figure 7. Hardware timing and synchronization on KV58F

- The top signal (**SM0 counter**) shows the eFlexPWM counter. The dead time is emphasized in the **PWM top** and **PWM bottom** signals.
- The **SM0** generates a trigger (**val4 – TRIG0**) for the ADC scan with a delay of approximately $T_{PWM} / 2$. This delay ensures correct DC-bus current sampling at all duty cycles.
- When the ADC scan is completed, the **ADC ISR** is entered. The fast loop calculation is done in this interrupt.

3.4.2. Peripheral settings

Only the peripherals used for motor control are described in this section. On KV58F, 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 DC-bus voltage, DC-bus current and back-EMF voltage. The eFlexPWM and HSADC are synchronized via the first eFlexPWM submodule and XBAR periphery. One channel from an independent FTM is used to generate the slow-loop interrupt.

3.4.2.1. PWM generation—PWMA

- eFlexPWM is clocked from the 100-MHz Fast Peripheral clock.
- Six channels from three submodules are used to generate a 3-phase PWM. Submodule 0 generates the master reload event every n^{th} opportunity. 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.
- Submodule 0 is used for synchronization with the ADC. The Val 4 register generates the output trigger $T_{PWM} / 2$ 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 $INIT = -MODULO / 2 = -2500$ and $VAL1 = MODULO / 2 - 1 = 2499$. The eFlexPWM clock is 100 MHz, so the PWM period is 0.00005 s (20 kHz).
- The dead time insertion is enabled. The dead time length is calculated as the Fast peripheral clock $100 \text{ MHz} \times T_{\text{deadtime}}$. The dead time varies among the platforms.

3.4.2.2. Analog sensing—ADC12

- The HSADC wrappers are similar to the cyclic ADC12 wrapper at KV46. There are two wrappers—HSADC0 and HSADC1. The HSADC0A and HSADC1A are used for MC analog sensing.
- The clock source for HSADC0A and HSADC1A is the 100 MHz Fast Peripheral clock divided by 2 = 50 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.
- All samples are enabled at each ADC.
- The end-of-scan interrupt serves for fast-loop algorithm calculation is generated when the all conversions are completed.

3.4.2.3. Peripheral interconnections—XBARA

- The PWM0_OUT_TRG20 output trigger generated by submodule 0 is connected to the HSADC0A_SYNC and HSADC1A_SYNC inputs.

3.4.2.4. Slow-loop interrupt generation—FTM2

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

3.4.2.5. Communication with MC33937 MOSFET driver—SPI

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

3.4.3. Peripheral settings differences among platforms

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

Table 7. KV58 platform differences

Peripheral	Feature	Platform	
		Tower System	HVP
FTM0	PWM polarity	high sides active low low sides active high	high sides active high low sides active high
	Dead time	0.8 μ s	0.8 μ s
SPI	Driver on SPI	Yes	No
PDB	Pre-trigger 0 delay	0.1 ms	0.1 ms

3.4.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. [Table 8](#) shows the memory usage and CPU load. The memory usage is calculated from the linker .map file, including 2 KB FreeMASTER recorder buffer (allocated in RAM) and 3.8 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the SysTick timer. The CPU load depends on the fast loop (FOC calculation) and slow loop (speed loop) frequencies. In this case, it applies to the fast loop frequency of 20 kHz and the slow loop frequency of 1 kHz. The total CPU load is calculated according to [Eq. 3](#).

Table 8. KV58 CPU load and memory usage

—	KV58
CPU clock [MHz]	237.5
Fast Control Loop [cycles] (%)	564 (4.8 %)
Slow Control Loop [cycles] (%)	1012 (0.4 %)
Total CPU load [%]	5.2 %
Flash usage [B]	15 892
RAM usage [B]	3 389

3.5. KE1xZ family

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

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

3.5.1. Hardware timing and synchronization

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

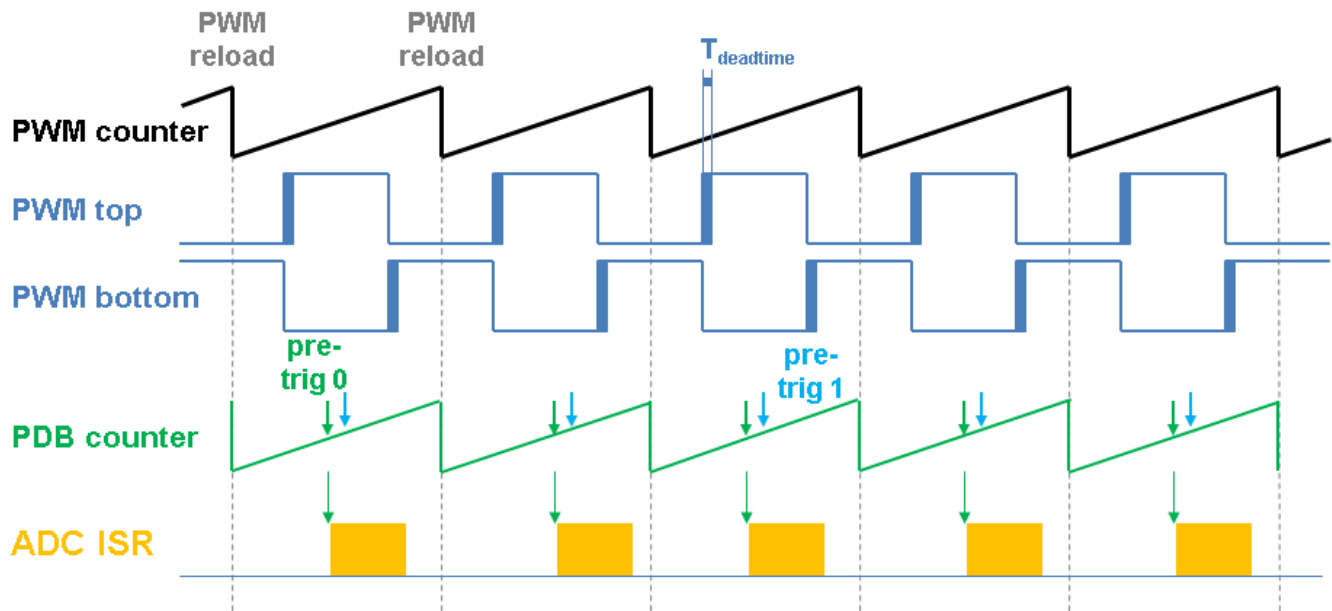


Figure 8. Hardware timing and synchronization on KE15Z

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized at the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated on the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates the first pre-trigger for the first ADC sample with a delay of approximately $T_{\text{PWM}} / 2$. This delay ensures a correct DC-bus current sampling.
- When the conversion of the first ADC sample is completed, the **ADC ISR** is entered and the fast-loop control function is calculated.
- The PDB uses the back-to-back mode to automatically generate the **pre-trig 1** (for the next quantity measurement) immediately after the first conversion is completed.

3.5.2. Peripheral settings

This section describes only the peripherals used for motor control. The KE1xZ uses a 6-channel FlexTimer (FTM) to generate a 6-channel PWM, and two 12-bit SAR ADCs to measure the back-EMF voltage, DC-bus current, and DC-bus voltage. The FTM and ADC are synchronized via the Programmable Delay Block (PDB). One channel from another independent FTM is used for the slow-loop interrupt generation.

3.5.2.1. PWM generation—FTM0

- The FTM is clocked from the 72-MHz System clock.

- Only six channels are used, the other two are masked in the OUTMASK register.
- Channels 0+1, 2+3, and 4+5 are combined in pairs and run in a complementary mode.
- The Fault mode is enabled for each combined pair with an automatic fault clearing (PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as a time for the FTM to count from CNTIN to MOD. By default, $CNTIN = -MODULO / 2 = -1800$ and $MOD = MODULO / 2 - 1 = 1799$. The FTM is clocked from the 72-MHz System clock, so it takes 0.00005 s (20 kHz).
- The dead time insertion is enabled for each combined pair. The dead time length is calculated as $System\ clock\ 72\ MHz \times T_{deadtime}$. The dead time varies among platforms.
- The FTM generates a trigger for the PDB on the counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.5.2.2. Analog sensing—ADC0 and ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 8-MHz System oscillator (SOSC), divided by 2 = 4 MHz. For calibration purposes, the ADC clock is 1 MHz.
- Both ADCs are triggered by the PDB pre-triggers.
- The interrupt that serves for the fast-loop algorithm calculation is generated when the first conversion is completed.

3.5.2.3. PWM and ADC synchronization—PDB0

- Like the FTM, the PDB is clocked from the 72-MHz System clock.
- The PDB is triggered by the FTM0_TRIG.
- At each channel, the pre-trigger 0 is generated in the middle of one PWM period. This is set up in the PDBx_CHnDLY register. $PDBx_CHnDLY = PWM_MODULO / 2$.
- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC. For more information about PDB error sequence handling, see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document [AN4822](#)).

3.5.2.4. Time event, forced commutation control—FTM1

- The FTM1 is used for the forced commutation control.
- The FTM1 is set as a free-running counter and clocked from the System clock. The Timer clock pre-scaler is set to 128.
- The FTM1 counts from 0 to 0xffff.
- The output-compare interrupt is enabled and generated when the counter equals the value register.

- The value register is periodically updated in the fast control loop function and the FTM1 output compare interrupt is not invoked until an error in the BLDC commutation process appears.
- If an error in the BLDC commutation process appears, the forced commutation is performed in the FTM1 interrupt service routine.

3.5.2.5. Slow-loop interrupt generation—FTM2

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

3.5.3. Peripheral settings differences among platforms

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

Table 9. KE1xZ platform differences

Peripheral	Feature	Platform
		Freedom
FTM0	PWM polarity	high sides active high low sides active high
	Dead time	0.875 μ s
SPI	Driver on SPI	No
PDB	Pre-trigger 0 delay	0.1 ms

3.5.4. CPU load and memory usage

The following information apply to the demonstration application built using the IAR Embedded Workbench IDE. The following table shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including the 2 KB FreeMASTER recorder buffer (allocated in RAM) and the 3.8 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the *SysTick* timer. The CPU load depends on the fast-loop (FOC calculation) and slow-loop (speed-loop) frequencies. In this case, it applies to the fast-loop frequency of 20 kHz and the slow-loop frequency of 1 kHz. The total CPU load is calculated according to [Eq. 3](#).

Table 10. KE1xZ CPU load and memory usage

—	KE15Z
CPU clock [MHz]	72
Fast Control Loop [cycles] (%)	1298 (34.6 %)
Slow Control Loop [cycles] (%)	2563 (3.6 %)
Total CPU load [%]	38.2 %
Flash usage [B]	16 194
RAM usage [B]	3 397

3.6. KE1xF family

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

For more information, see *KE1xZ Sub-Family Reference Manual* (document [KE1xZFP100M168SF0RM](#)).

3.6.1. Hardware timing and synchronization

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

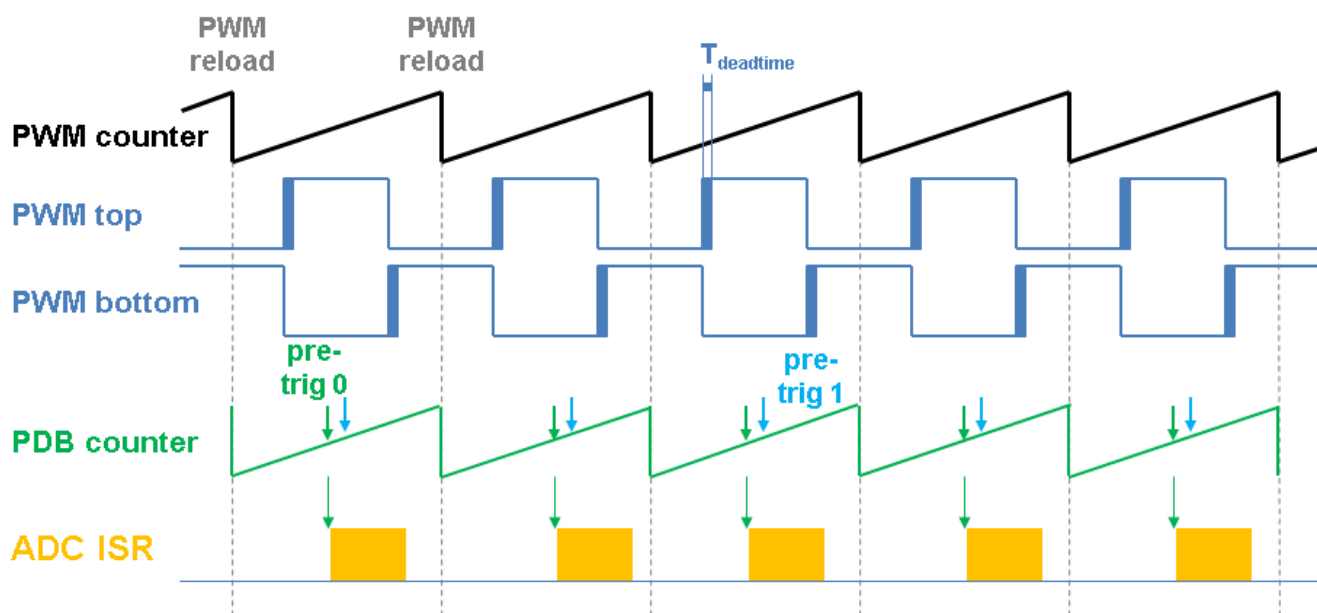


Figure 9. Hardware timing and synchronization on KE18F

- The top signal (**PWM counter**) shows the FTM counter reloads. The dead time is emphasized at the **PWM top** and **PWM bottom** signals. The FTM_TRIG is generated on the **PWM reload**, which triggers the PDB (resets the **PDB counter**).
- The PDB generates the first pre-trigger for the first ADC sample with a delay of approximately $T_{PWM} / 2$. This delay ensures a correct DC-bus current sampling.
- When the conversion of the first ADC sample is completed, the **ADC ISR** is entered and the fast-loop control function is calculated.
- The PDB uses the back-to-back mode to automatically generate the **pre-trig 1** (for the next quantity measurement) immediately after the first conversion is completed.

3.6.2. Peripheral settings

This section describes only the peripherals used for motor control. The KE1xF uses a 6-channel FlexTimer (FTM) to generate a 6-channel PWM, and two 12-bit SAR ADCs to measure the back-EMF voltage, DC-bus current, and DC-bus voltage. The FTM and ADC are synchronized via the Programmable Delay Block (PDB). One channel from another independent FTM is used for the slow-loop interrupt generation.

3.6.2.1. PWM generation—FTM0

- The FTM is clocked from the 168-MHz System clock.
- Only six channels are used, the other two are masked in the OUTMASK register.
- Channels 0+1, 2+3, and 4+5 are combined in pairs and run in a complementary mode.
- The Fault mode is enabled for each combined pair with an automatic fault clearing (PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero).
- The PWM period (frequency) is determined as a time for the FTM to count from CNTIN to MOD. By default, $CNTIN = -MODULO / 2 = -4200$ and $MOD = MODULO / 2 - 1 = 4199$. The FTM is clocked from the 168-MHz System clock, so it takes 0.00005 s (20 kHz).
- The dead time insertion is enabled for each combined pair. The dead time length is calculated as $System\ clock\ 168\ MHz \times T_{deadtime}$. The dead time varies among platforms.
- The FTM generates a trigger for the PDB on the counter initialization.
- The FTM fault input is enabled, but its polarity and source vary among platforms.

3.6.2.2. Analog sensing—ADC0 and ADC1

- The ADCs operate as 12-bit, single-ended converters.
- The clock source for both ADCs is the 8-MHz System Oscillator (SOSC) clock, divided by 1 = 8 MHz. For calibration purposes, the ADC clock is 1 MHz.
- Both ADCs are triggered by the PDB pre-triggers.
- An interrupt that serves for the fast-loop algorithm calculation is generated when the first conversion is completed.

3.6.2.3. PWM and ADC synchronization—PDB0

- 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 in the middle of one PWM period. This is set up in the PDBx_CHnDLY register. $PDBx_CHnDLY = PWM_MODULO / 2$.
- At each channel, the pre-trigger 1 is generated immediately after the first conversion is completed using the back-to-back mode.
- The PDB Sequence Error interrupt is enabled. This interrupt is generated if a certain result register is not read and the same pre-trigger occurs at this ADC. For more information about the

PDB error sequence handling, see *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document [AN4822](#)).

3.6.2.4. Time event, forced commutation control—FTM1

- The FTM1 is used for the forced commutation control.
- The FTM1 is set as a free-running counter and clocked from the System clock. The timer clock pre-scaler is set up to 128.
- The FTM1 counts from 0 to 0xffff.
- The output compare interrupt is enabled and generated when the counter equals the value register.
- The value register is periodically updated in the fast control loop function and the FTM1 output compare interrupt is not invoked until an error in the BLDC commutation process appears.
- If an error in the BLDC commutation process appears, the forced commutation is performed in the FTM1 interrupt service routine.

3.6.2.5. Slow-loop interrupt generation—FTM2

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

3.6.2.6. 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 0.5 MHz.

3.6.3. Peripheral settings differences among platforms

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

Table 11. KE18F 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
	Dead time	0.37 μ s	0.37 μ s
LPSPi	Driver on LPSPi	Yes	No
PDB	Pre-trigger 0 delay	0.1 ms	0.1 ms

3.6.4. CPU load and memory usage

The following information apply to the demonstration application built using IAR Embedded Workbench IDE. The following table shows the memory usage and CPU load. The memory usage is calculated from the linker *.map* file, including the 2 KB FreeMASTER recorder buffer (allocated in RAM) and the 3.8 KB FreeMASTER TSA (Target-Side Addressing) table (allocated in flash). The CPU load is measured using the *SysTick* timer. The CPU load depends on the fast-loop (FOC calculation) and slow-loop (speed-loop) frequencies. In this case, it applies to the fast-loop frequency of 20 kHz and the slow-loop frequency of 1 kHz. The total CPU load is calculated according to [Eq. 3](#).

Table 12. KE1xF CPU load and memory usage

—	KE18F
CPU clock [MHz]	168
Fast Control Loop [cycles] (%)	769 (9.2 %)
Slow Control Loop [cycles] (%)	1216 (0.7 %)
Total CPU load [%]	9.9 %
Flash usage [B]	16 056
RAM usage [B]	3 409

4. Motor Control Peripheral Drivers

Motor Control Peripheral Drivers (MCDRV) represent a simple way of peripheral initialization and access for the purposes of 3-phase ACIM, PMSM or BLDC control. The features provided by the MCDRV library are 3-phase PWM generation and 3-phase current measurement, as well as the DC-bus voltage, back-EMF voltage and auxiliary quantity measurement. The principle of BLDC sensorless application based on integration of back-EMF voltage is described in *3-Phase Sensorless BLDC Motor Control Application* (document [DRM144](#)).

The MCDRV are divided into two parts:

- The first part is the peripheral initialization module, consisting of *mcdrv_<platform>-<device>.c* and *mcdrv_<platform>-<device>.h* files, which are unique for each supported device. The header file includes all MCDRV setup options including the ADC channel assignment. The source file contains the functions to initialize all peripherals used for motor control. This module is described in [Section 4.2, “Motor Control Peripheral Drivers API”](#).
- The second part consists of the peripheral driver library modules for each supported periphery. Generally, all ADC and PWM periphery drivers share the same API within their class. This enables the higher-level code to be platform-independent, as the peripheral driver function calls were replaced by universally named macros. The list of supported peripherals and the API of their drivers is described in [Section 4.2, “Motor Control Peripheral Drivers API”](#).

4.1. Motor Control Peripheral Drivers initialization

The MCDRV initialization module consists of a set of MCU peripheral-initialization functions, as well as all the definitions that you can specify. The functions are contained in device-specific *mcdrv_<platform>-<device>.c* source and *mcdrv_<platform>-<device>.h* header files. Out of all functions in the MCDRV initialization module, call the *MCDRV_Init_MI()* function during MCU

startup and before calling any other MCDRV functions. All peripherals used by a given device for motor-control purposes are initialized within this function.

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

- `M1_MCDRV_ADC`—this macro specifies the ADC peripheral used. If you select an unsupported peripheral, the preprocessor error is issued.
- `M1_MCDRV_PWM3PH`—this macro specifies the PWM peripheral used. If you select an unsupported peripheral, the preprocessor error is issued.
- `M1_PWM_PAIR_PH[A..C]`—these macros enable simple assignment of the physical motor phases to the PWM peripheral channels or submodules. Change the order of the motor phases this way.
- `MCDRV_CHAN_OFF`—use this macro to specify the unassigned ADC channel. It is recommended not to change this number.
- `M1_ADC[0,1]_PH[A..C]`—these macros serve to assign the ADC channels for back-EMF voltage measurement. The general rule is that the only one ADC module can be assigned to sense required back-EMF voltage. When this rule is broken, preprocessor error is issued. The unassigned ADC channels are set to the `MCDRV_CHAN_OFF` value.
- `M1_ADC[0,1]_UDCB`, `M1_ADC[0,1]_IDCB` and `M1_ADC[0,1]_AUX`—use these defines to select the ADC channel to measure the DC-bus voltage, DC-bus current and one user-defined auxiliary quantity, which is not used directly for motor control (the IPM temperature is measured by default).
- `M1_SET_PTR_U_DC_BUS`—this define must contain the name of the 16-bit fractional variable where you want to store the DC-bus voltage measurements.
- `M1_SET_PTR_I_DC_BUS`—this define must contain the name of the 16-bit fractional variable where you want to store the DC-bus current measurements.
- `M1_SET_PTR_BEMF_VOLT`—this define must contain the name of the 16-bit fractional variable where you want to store the back-EMF voltage measurements.
- `M1_SET_PTR_AUX_CHAN`—this define stores the name of the 16-bit fractional variable, in which you want to store the auxiliary quantity measured values.

4.2. Motor Control Peripheral Drivers API

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

The available API for the ADC MC drivers is:

- `M1_MCDRV_ADC_T`—MCDRV ADC structure data type.
- `bool_t M1_MCDRV_ADC_PERIPH_INIT()`—by default, this function is called during the ADC peripheral initialization procedure invoked by the `MCDRV_Init_MI()` function, and it must not be called again after the peripheral initialization is done.
- `bool_t M1_MCDRV_ADC_ASSIGN_BEMF(MCDRV_ADC_T*)`—calling this function assigns the proper ADC channels for the next back-EMF voltage measurement based on the actual commutation sector. This function always returns `true`.

- *bool_t MI_MCDRV_CURR_CALIB_INIT(MCDRV_ADC_T*)*—this function initializes the DC-bus current channel-offset measurement. This function always returns *true*.
- *bool_t MI_MCDRV_CURR_CALIB(MCDRV_ADC_T*)*—this function reads the current information from the unpowered phase of a standstill motor and filters them using moving average filter. The goal is to obtain the value of the measurement offset. The length of the window for moving average filters is set to eight samples by default. This function always returns *true*.
- *bool_t MI_MCDRV_CURR_CALIB_SET(MCDRV_ADC_T*)*—this function asserts the DC-bus current measurement offset value to the internal register. Call this function after a sufficient number of *MI_MCDRV_CURR_3PH_CALIB()* calls. This function always returns *true*.
- *bool_t MI_MCDRV_ADC_GET(MCDRV_ADC_T*)*—this function reads and calculates the actual values of the back-EMF voltage, DC-bus voltage, DC-bus current and auxiliary quantity. The obtained measurements are stored in the *M1_SET_PTR_I_DC_BUS*, *M1_SET_PTR_U_DC_BUS*, *M1_SET_PTR_BEMF_VOLT* and *M1_SET_PTR_AUX_CHAN* variables. This function always returns *true*.

The API for the PWM MC drivers is:

- *MI_MCDRV_PWM_3PH_T*—MCDRV PWM structure data type.
- *bool_t MI_MCDRV_PWM_PERIPH_INIT(MCDRV_FTM_PWM3PH_T*)*—this function is called by default during the PWM periphery-initialization procedure invoked by the *MCDRV_Init_MI()* function. This function always returns *true*.
- *bool_t MI_MCDRV_PWM3PH_SET_DUTY(MCDRV_FTM_PWM3PH_T*, int16_t i16InpDuty)*—this function updates the PWM duty cycles based on the required value stored in the *f16DutyCycle* variable. This function always returns *true*.
- *bool_t MI_MCDRV_PWM3PH_SET_PWM_OUTPUT(MCDRV_FTM_PWM3PH_T*, int16_t i16Sector)*—calling this function enables the required PWM channels and disables non-required PWM channels according to actual commutation sector.

The API for the CMT MC drivers is:

- *MI_MCDRV_TMR_CMT_T*—MCDRV CMT structure data type.
- *bool_t MI_MCDRV_TMR_CMT_PERIPH_INIT(MCDRV_FTM_CMT_T*)*—this function is called by default during the FTM periphery-initialization procedure invoked by the *MCDRV_Init_MI()* function. This function always returns *true*.
- *bool_t MI_MCDRV_TMR_CMT_SET(MCDRV_FTM_CMT_T*)*—this function updates the FTM value register. This function always returns *true*.
- *bool_t MI_MCDRV_TMR_CMT_GET(MCDRV_FTM_CMT_T*)*—this function reads the actual values of FTM value and counter register. This function always returns *true*.

5. Tuning and Controlling the Application

This section provides information about the tools and recommended procedures for controlling the sensorless BLDC motor control application. The application contains the embedded side driver of FreeMASTER real-time debug monitor and data visualization tool for communication with the PC. It supports non-intrusive monitoring as well as the modification of target variables in real time, which is

very useful for algorithm tuning. Besides the target-side driver, FreeMASTER requires installing the PC application as well. For more information, visit www.nxp.com/freemaster.

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

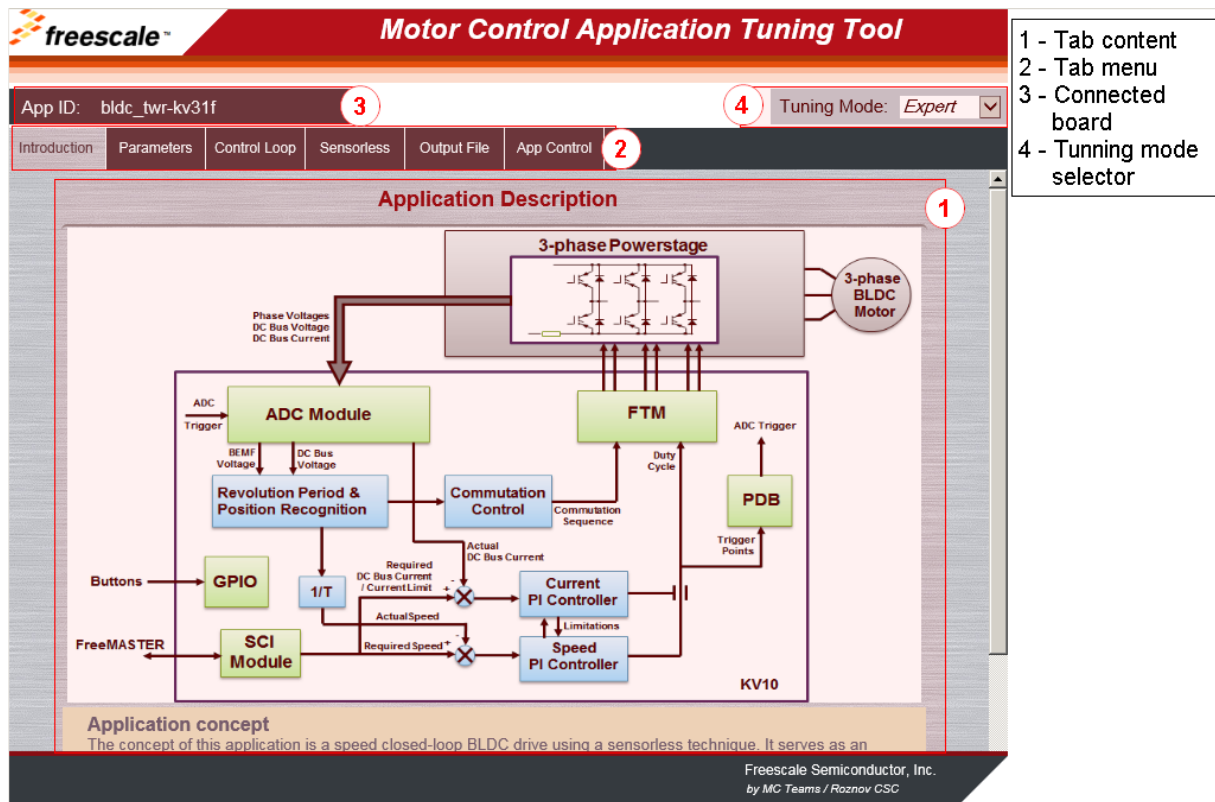


Figure 10. MCAT tool layout

In the default configuration, these tabs are available:

- “Introduction”—welcome page with the BLDC sensorless application diagram and a short description of the application.
- “Parameters”—this page enables you to modify the motor parameters, the specification of hardware and application scales, and fault limits.

- “Control Loop”—this tab enables you to modify speed and current (torque)-loop PI controller gains and output limits and to modify speed ramp parameters.
- “Sensorless”—this page enables you to tune the important parameters for sensorless BLDC application like integration threshold, minimal speed and open loop startup parameters.
- “Output file”—this tab enables the user to view all calculated constants that are required by the BLDC sensorless application. It also enables you to generate the *ml_bldc_appconfig.h* file, which is then used to preset all application parameters permanently at project rebuild.
- “App Control”—this tab contains graphical elements such as speed gauges, DC-bus voltage and DC-bus current measurement bar, and variety of switches that enable simple, quick, and user-friendly application control. Here you can control the fault-clearing and demo mode, which sets various predefined required speeds over time.

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

The following sections describe the individual MCAT tabs more deeply.

5.1. BLDC sensorless application control and tuning using MCAT tool

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

5.1.1. Application parameters setup using MCAT tool

The screenshot displays the 'Parameters' tab of the 'Motor Control Application Tuning Tool'. The interface is organized into several sections for parameter configuration:

- Motor Parameters:**
 - pp: 2 [-]
 - iph nom: 1.67 [A]
 - Uph nom: 24 [V]
 - N nom: 4000 [rpm]
- Fault Limits:**
 - U DCB trip: 29 [V]
 - U DCB under: 14.5 [V]
 - U DCB over: 29 [V]
 - N over: 4180 [rpm]
 - N min: 200 [rpm]
- Hardware Scales:**
 - I max: 8 [A]
 - U DCB max: 36.3 [V]
- Application Scales:**
 - N max: 4400 [rpm]
 - ke: 0.02865 [V sec/rad]
- Alignment:**
 - Align current: 1.34 [A]
 - Align duration: 1 [sec]

At the bottom of the tool, there are three buttons: 'Update Target', 'Reload Data', and 'Store Data'. The Freescale logo and 'Motor Control Application Tuning Tool' title are visible at the top. The App ID is 'bldc_twr-kv31f' and the Tuning Mode is set to 'Basic'.

Figure 11. Parameters tab

In basic tuning mode the motor parameters and hardware scales can be modified only. Other application parameters like Fault limits, application scales and alignment constants are calculated from motor parameters and hardware scales.

More detailed information about individual parameters are provided in following table:

Table 13. MCAT tool parameters tab description

Motor Parameters			
Parameter	Units	Description	Typical range
pp	—	Motor pole-pairs	1–10
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	1000–5000
Hardware Scales			
Parameter	Units	Description	
I _{max}	[A]	Current scale	
U DCB max	[V]	Voltage scale	
Fault Limits			
Parameter	Units	Description	Typical value
U DCB trip	[V]	Trip DC-BUS voltage for braking into resistor	0.8 * U DCB max
U DCB under	[V]	DC Bus under voltage fault trigger	0.4 * U DCB max
U DCB over	[V]	DC Bus over voltage fault trigger	0.8 * U DCB max
N over	[rpm]	Over speed fault trigger	0.95 * N max
N min	[rpm]	Minimal motor speed	0.05 * N nom
Application Scales			
Parameter	Units	Description	Typical range
N max	[rpm]	Motor mechanical speed scale	1.1 * N nom
Ke	[V.sec/rad]	Back EMF constant, calculated as $(U_{nom} * 60) / (2 * \pi * pp * N_{nom})$	0.00001 – 0.1
Alignment			
Parameter	Units	Description	Typical range
Align current	[A]	Align current	0.1 – Iph nom
Align duration	[sec]	Align duration	0.2 - 10

There are three buttons, at the bottom of parameters tab. The **Update Target** button serves to write the changes into MCU. The **Reload Data** button serves to reload parameters (default values) from inner file. The **Store Data** serves to save the parameters from parameters tab into inner file. Reload data and store data button are non-active by default until some change in parameters is performed.

5.1.2. Application Control loops setup using MCAT tool

The screenshot displays the 'Motor Control Application Tuning Tool' interface. At the top, the 'freescaler' logo is on the left, and the title 'Motor Control Application Tuning Tool' is in the center. Below the title, the 'App ID' is 'bldc_twr-kv31f' and the 'Tuning Mode' is 'Basic'. A navigation bar contains tabs for 'Introduction', 'Parameters', 'Control Loop', 'Sensorless', 'Output File', and 'App Control'. The 'Control Loop' tab is active, showing the 'BLDC Control Loop' configuration. The parameters are organized into several sections:

- Loop Sample Time:** Sample time (0.001 [sec]), PWM freq (20000 [Hz]).
- PI controller type:** Parallel.
- Control Loop Output Limits:** Output limit high (90 [%]), Output limit low (0 [%]).
- Speed Ramp:** Inc Up (2000 [rpm/sec]), Inc Down (2000 [rpm/sec]).
- Speed PI Controller Constants:** Speed Loop Kp (0.00003), Speed Loop Ki (0.03).
- Torque PI Controller Constants:** Torque Loop Kp (0.0178), Torque Loop Ki (17.8).

At the bottom of the interface, there are three buttons: 'Update Target', 'Reload Data', and 'Store Data'. The footer text reads 'Freescale Semiconductor, Inc. by MC Teams / Roznov CSC'.

Figure 12. Control Loop Tab

In basic tuning mode the Speed Ramp parameters can be modified only. Other control loop parameters like control loop output limits and PI controllers constants can be accessed in expert mode only.

More detailed information about individual control loop parameters are provided in following table:

Table 14. MCAT tool control loop tab description

Loop Sample Time			
Parameter	Units	Description	Typical range
Sample time	[sec]	Speed control loop sampling period	1 – 10 ms
PWM freq	[Hz]	PWM frequency	4000 - 20000
Control Loop Output Limits			
Parameter	Units	Description	Typical value
Output limit high	[%]	High limit of BLDC control loop output in percentage of duty cycle	90
Output limit low	[%]	Low limit of BLDC control loop output in percentage of duty cycle	0
Speed Ramp			
Parameter	Units	Description	Typical range
Inc Up	[rpm/sec]	Speed Ramp up increment	5 – 3000
Inc Down	[rpm/sec]	Speed Ramp down decrement	5 - 3000
Speed PI Controller Constants			
Parameter	Units	Description	Typical value
Speed Loop Kp	□	Speed Controller proportional constant in time domain	0.00003
Speed Loop Ki	□	Speed Controller integration constant in time domain	0.03
Torque PI Controller Constants			
Parameter	Units	Description	Typical range
Torque Loop Kp	□	Torque Controller proportional constant in time domain	0.0178
Torque Loop Ki	□	Torque Controller integration constant in time domain	17.8

There are three buttons, at the bottom of control loop tab. The **Update Target** button serves to write the changes into MCU. The **Reload Data** button serves to reload parameters (default values) from inner file. The **Store Data** serves to save the parameters from parameters tab into inner file. Reload data and store data button are non-active by default until some change in parameters is performed.

5.1.3. BLDC Sensorless application setup using MCAT tool

The screenshot displays the 'Sensorless Control' tab in the MCAT tool. The 'App ID' is 'bldc_twr-kv31f' and the 'Tuning Mode' is 'Basic'. The interface is divided into several sections:

- Sensorless Parameters:**
 - Timer freq: 468750 [Hz]
 - Speed min: 250 [rpm]
 - Freewheel time: 1 [sec]
- Open Loop Start-up Parameters:**
 - OL speed lim: 360 [rpm]
 - Cmt count: 2 [#]
 - 1st cmt period: 0.05 [sec]
- Commutation Parameters:**
 - Time off: 22 [%]
 - Integ thr corr.: 100 [%]
- Sensorless Control Constants:**
 - Cmt_period_min: 533
 - Cmt_period_start: 23438
 - Speed_scale: 3196
 - Start_acceleration: 0.27777778
 - Integration_threshold: 67707

At the bottom, there are three buttons: 'Update Target', 'Reload Data', and 'Store Data'. The footer indicates 'Freescale Semiconductor, Inc. by MC Teams / Roznov CSC'.

Figure 13. Control Loop Tab

In basic tuning mode the integration threshold correction constant can be modified only. Other sensorless parameters like start-up parameters can be accessed in expert mode only.

More detailed information about individual sensorless control parameters are provided in following table:

Table 15. MCAT tool sensorless control tab description

Sensorless Parameters			
Parameter	Units	Description	Typical range
Timer freq	[Hz]	Forced commutation timer frequency, calculated as Timer input clock / Prescale factor (128)	
Speed min	[rpm]	Minimal speed for sensorless control	0.05 * Nnom
Freewheel time	[sec]	Motor Freewheel period from any motor speed	0.5 - 5
Open Loop Start-up Parameters			
Parameter	Units	Description	Typical range
OL sped lim	[rpm]	Open loop start-up minimal speed to switch to close loop control	0.1 – 0.3 Nnom
Cmt count	[#]	Open loop start-up commutation number	2 - 8
1 st cmt period	[sec]	First commutation period	0.05
Commutation Parameters			
Parameter	Units	Description	Typical range
Time off	[%]	Time off after commutation	15 - 25
Integ thr corr	[%]	Integration threshold correction constant	10 - 200

There are three buttons, at the bottom of control loop tab. The **Update Target** button serves to write the changes into MCU. The **Reload Data** button serves to reload parameters (default values) from inner file. The **Store Data** button serves to save the parameters from parameters tab into inner file. Reload data and store data button are non-active by default until some change in parameters is performed.

According to entered parameters the following sensorless control constants are calculated:

Table 16. MCAT tool sensorless control calculated constants

Sensorless Control Constants			
Parameter	Units	Description	Calculation
Cmt_period_min	□	Number of timer interrupt counts for maximal speed	$\frac{Timer_{freq} * 10}{N_{max} * pp}$
Cmt_period_start	□	Number of timer interrupt counts for start-up speed	$Timer_{freq} * Cmt_{period}$
Speed_scale	□	Speed scale constant for actual speed calculation	$\frac{Timer_{freq} * pp}{N_{max} * pp}$
Start_acceleration	□	Start-up acceleration constant	$\left(\frac{1}{60 * (N_{OLspeed\ lim} * pp * 6 * Cmt_{per})} \right)^{CMT_{count} - 1}$
Integration_threshold	□	Integration threshold constant	$\frac{PWM_{freq} * \pi * k_e * 32768}{U_{max} * 24}$

5.1.4. BLDC sensorless application tuning using MCAT tool

This section provides a guide for running your motor in several steps. It is highly recommended to go through all the steps carefully to eliminate any issues during the tuning process. The state diagram in the

following figure shows a typical BLDC sensorless control tuning process. The tuning phases are described in the following sections.

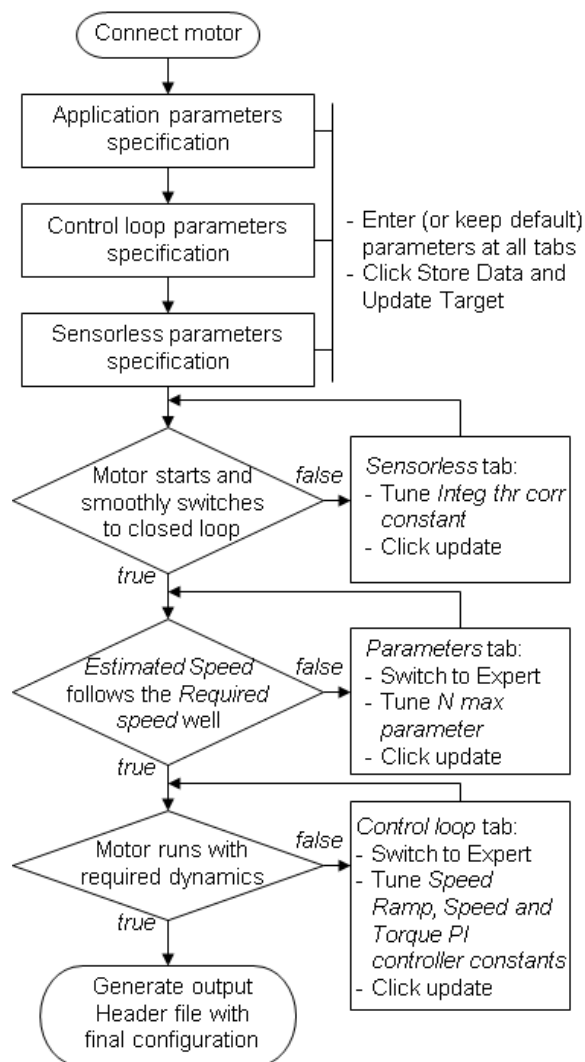


Figure 14. Running a new BLDC

5.1.5. Initial configuration setting and update

1. Open the BLDC sensorless control application FreeMASTER project containing the dedicated MCAT plug-in module.
2. Select the “Basic” mode—recommended for users who are not experienced in motor-control theory. The number of required input parameters is reduced.
3. Select the “Parameters” tab.
4. Leave the motor parameters and as they are if you are using Linux motor. If you use other motor, set correct motor parameters.
5. Set the hardware scales—modifying these two fields is not required when using a reference to the standard power stage board. These scales specify the maximum measurable current and voltage analog quantities.

6. Check the fault limits—these fields are not accessible in the “Basic” mode and they are calculated using the motor parameters and hardware scales. See this table:

Table 17. Fault limits

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

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

Table 18. Application scales

Parameter	Units	Description	Typical range
N max	[rpm]	Speed scale	>1.1 * N nom
Ke	[V.sec/rad]	Back EMF constant, calculated as $(U_{nom} * 60) / (2 * \pi * f_{pp} * N_{nom})$	0.00001 – 0.1

8. Check the alignment parameters—these fields are not accessible in the Basic mode and are calculated using the motor parameters and hardware scales. The parameters express the required phase current value applied to the motor winding during rotor alignment and its duration.
9. Click the “Store Data” button to save the modified parameters into the inner file.
10. Select the “Control Loop” tab.
11. Set the speed ramp parameters, according to your required dynamics.
12. Select the “App Control” tab.
13. Click on Run button and set your required speed on speed slider.

5.1.6. Alignment tuning

The alignment procedure sets the rotor to an accurate initial position and enables applying a full start-up torque to the motor. The rotor-alignment parameters are available for editing in the Expert mode. A correct initial position is needed mainly for high start-up loads (compressors, washers, and others). The aim of the alignment is to have the rotor in a stable position (without oscillations) before the startup.

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

5.1.7. Open-loop startup tuning

Tune the start-up process by a set of parameters located in the “Sensorless” tab. Set the optimal values to achieve a proper motor startup. An example start-up state of low-dynamic drives (fans, pumps) is shown in Figure 15.

1. Select the “Sensorless” tab and switch to expert mode.
2. Set the open loop speed limit parameter (OL speed lim). It is the minimal speed to switch to close loop control.
3. Set the start-up commutation count parameter. It is the number of open loop start-up commutations to be performed before switching to close loop control.
4. Set the first commutation period (1st cmt period). It is the time duration between zero speed and first commutation. This parameter is responsible for acceleration during start-up.
5. Click the “Update Target” button to write the changes to the MCU.
6. Switch to the “App Control” tab, and try to run the motor.

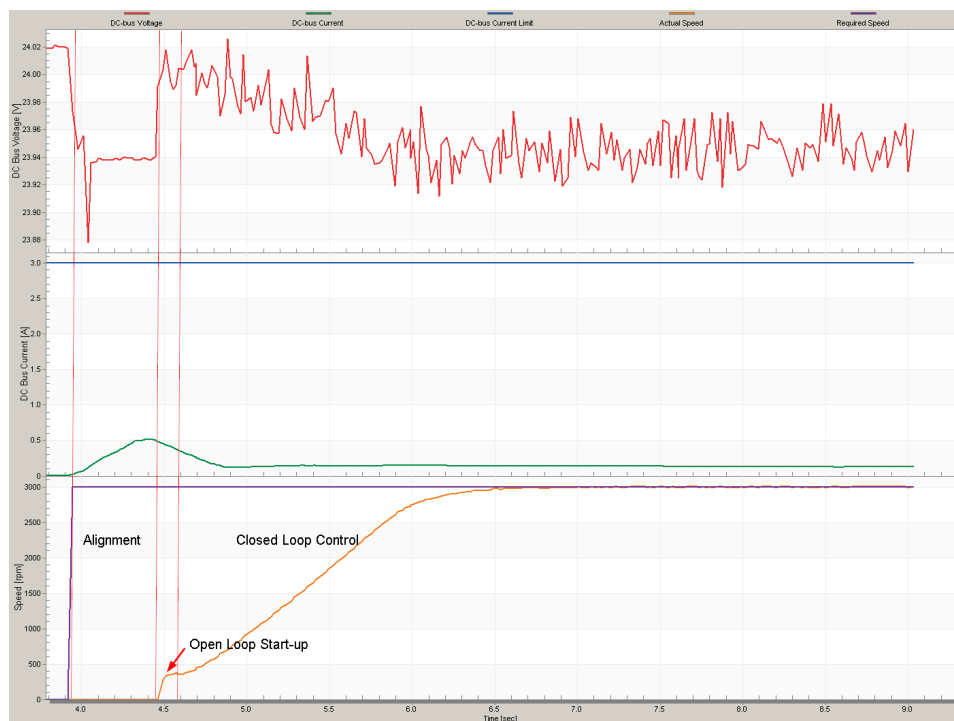


Figure 15. Motor open-loop startup

5.1.8. Speed ramp tuning

The “Speed” command is applied to the speed controller through a speed ramp. The ramp function contains two increments (up and down) that express motor acceleration and deceleration per second. If the increments are very high, they can cause an over-current fault during acceleration and an over-voltage fault during deceleration.

The increment fields are located in the “Control Loop” tab and they are accessible in both tuning modes. Clicking the “Update Target” button writes the changes to the MCU. An example speed profile is shown

in the following figure. The ramp down increment is set to 2000rpm/sec, while the up increment is set to 500 rpm/sec.

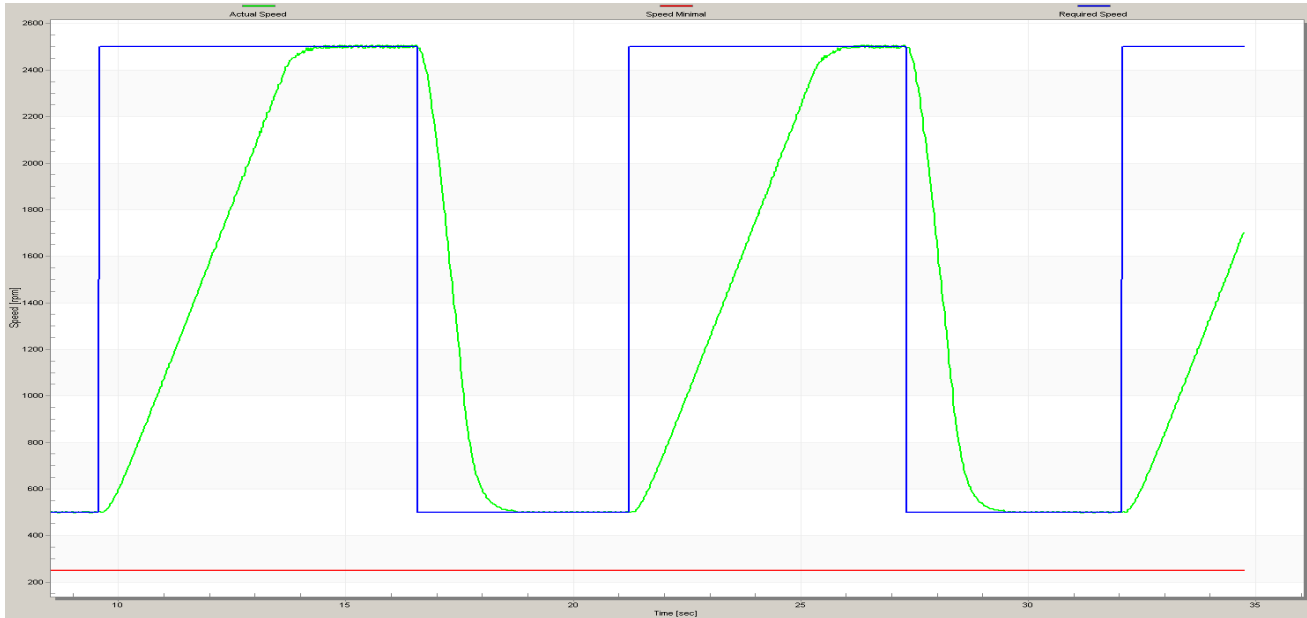


Figure 16. Speed ramp

5.1.9. Current and Speed PI controllers tuning

The current and speed PI controllers constants are adjusted to the same values in fractional format by default. The different scale for current and speed PI controller constants is the reason why they are different in s-domain and in “Control loop” tab. The conversion relationship between controller constants in fractional and s-domain is the following:

Current PI controller constants:

$$K_{I\ frac} = K_{I\ s} * T_s * \frac{I_{max}}{U_{max}} \Rightarrow K_{I\ s} = \frac{K_{I\ f} * U_{max}}{T_s * I_{max}}$$

$$K_{P\ frac} = K_{P\ s} * \frac{I_{max}}{U_{max}} \Rightarrow K_{P\ s} = \frac{K_{P\ f} * U_{max}}{I_{max}}$$

Speed PI controller constants:

$$K_{I\ frac} = K_{I\ s} * T_s * \frac{N_{max}}{I_{max}} \Rightarrow K_{I\ s} = \frac{K_{I\ f} * I_{max}}{T_s * N_{max}}$$

$$K_{P\ frac} = K_{P\ s} * \frac{N_{max}}{I_{max}} \Rightarrow K_{P\ s} = \frac{K_{P\ f} * I_{max}}{N_{max}}$$

Where T_s is control loop sampling period.

Generally, it is not recommended to change the default PI controller constants. The current PI controller output limits the speed PI controller output to not exceed the nominal phase current of motor.

If it is required by final application to tune the PI controller constants it is possible to do in MCAT tool in Control Loop tab. Follow these instructions:

1. Select the Speed scope from the FreeMASTER Project Tree.
2. Select the Control loop tab, switch to expert mode.
3. Tune the proportional gain:
 - Set the speed loop Ki (SL_Ki) integral gain to zero.
 - Set the speed ramp to 1000 rpm/sec or higher.
 - Switch to the App Control tab and run the motor at a convenient speed, about 30% of nominal speed.
 - Set a step in the required speed to 40% of N nom.
 - Switch back to the Control loop tab.
 - Adjust the proportional gain Speed loop Kp (SL_Kp) until the system responds properly to the required value without oscillations or excessive overshoot:
 - If SL_Kp is set low, then the system respond will be slow.
 - If SL_Kp is set high, then the system respond will be tighter.
 - When $SL_Ki = 0$, then the system will probably not achieve the required speed.
 - Click the Update Target button to apply the changes to the MCU.
4. Tune the integral gain:
 - Increase slowly SL_Ki to minimize the difference between the required and actual speeds to zero.
 - Adjust SL_Ki in such a way as to 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.

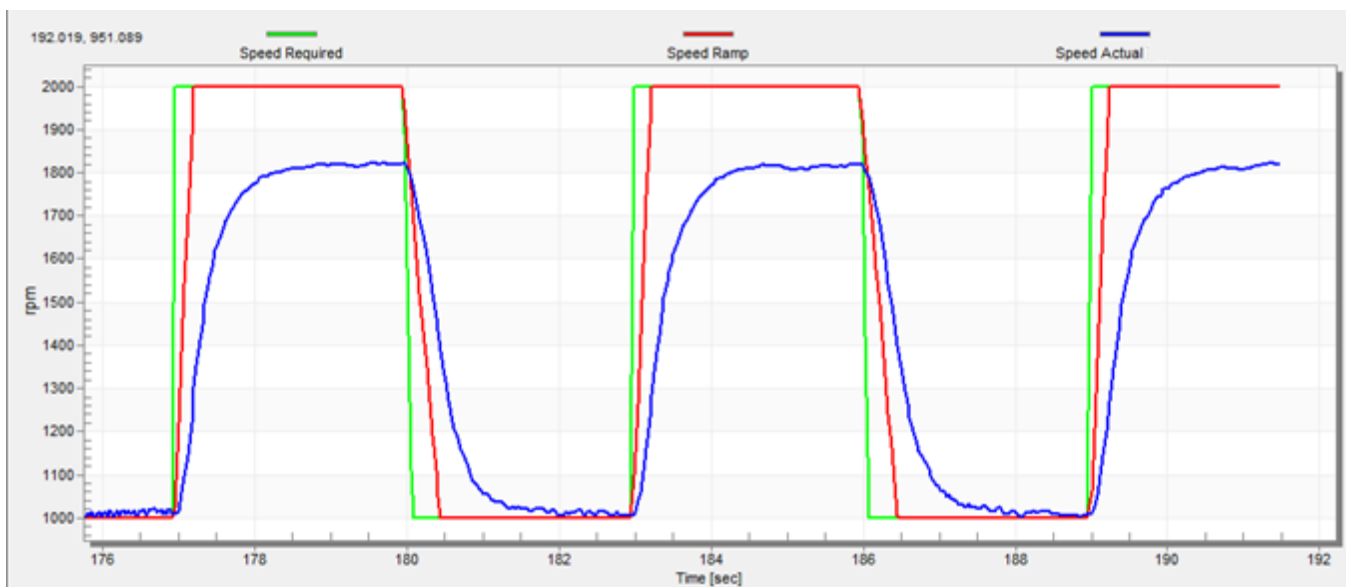


Figure 17. Speed controller response— SL_Ki value is low, Speed Ramp not achieved

- SL_Ki value is high, and *Speed Actual* greatly overshoots and the long settling time is unwanted, see this figure:

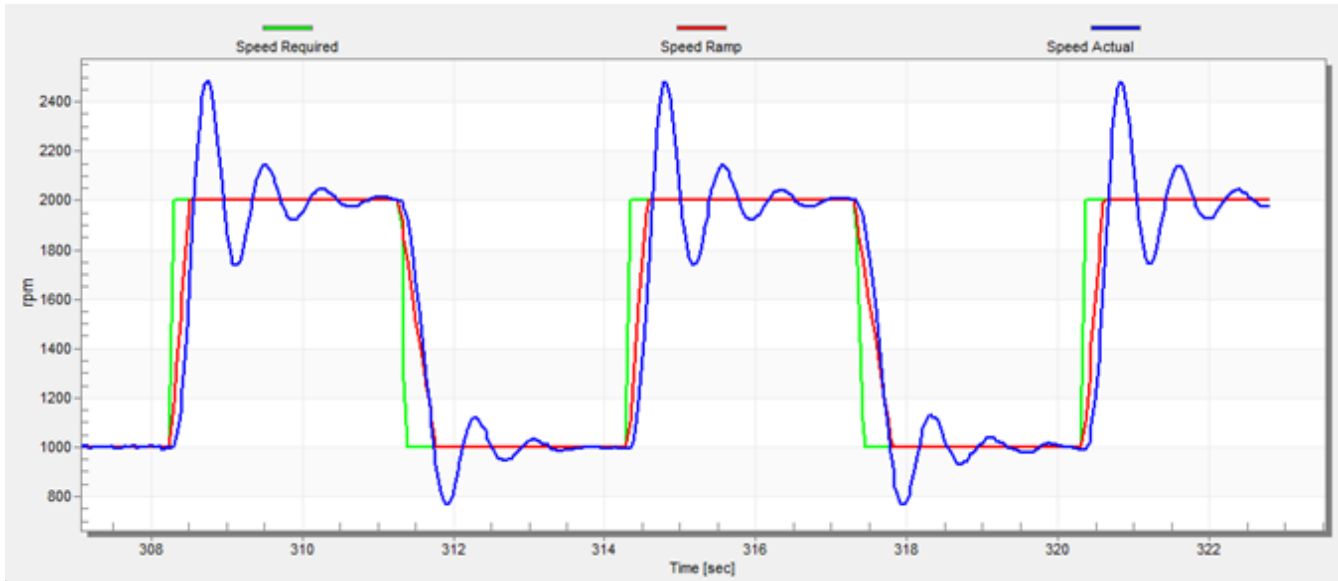


Figure 18. Speed Controller Response— SL_Kp value is low, *Speed Actual Filtered* greatly overshoots

- Speed loop response is with a small overshoot, and the *Speed Actual* settling time is sufficient. Such a response can be considered as an optimal, see this figure:

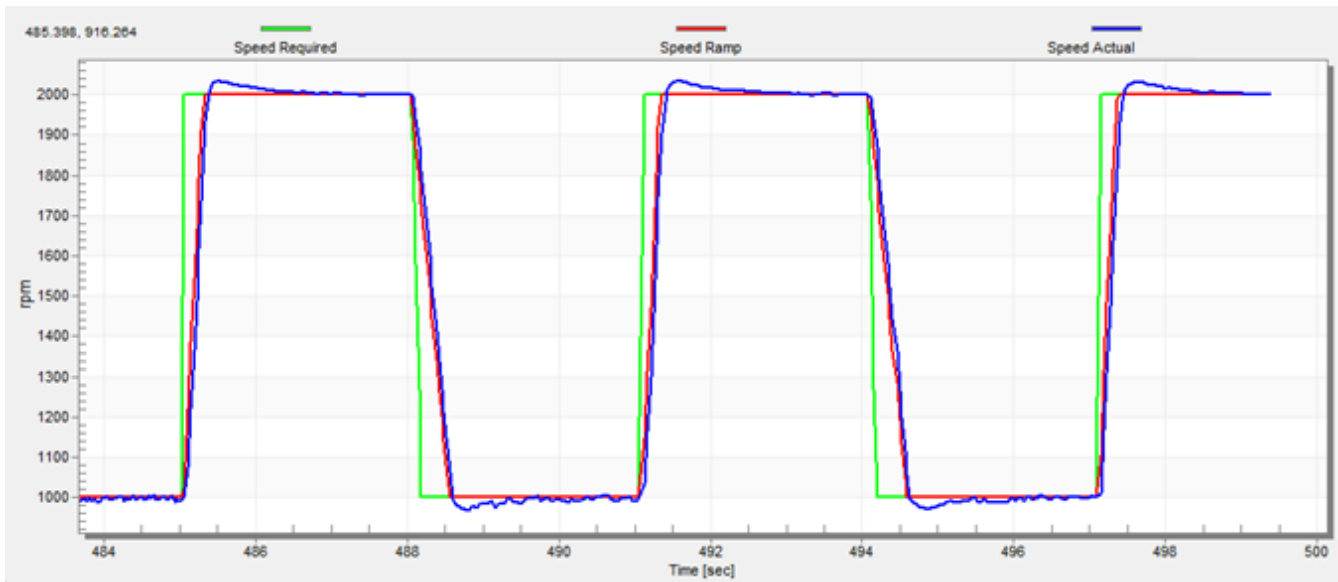


Figure 19. Speed Controller Response—Speed loop response with small overshoot

5.1.10. Generating MCAT tool output file

When you successfully finish tuning the application and you want to store all calculated parameters to the embedded application, navigate to the “Output File” tab. View the list of all definitions generated by MCAT tool. Clicking the “Generate Configuration File” button overwrites the old version of the *ml_bldc_appconfig.h* file, which contains these definitions. Provide a correct path to the file for a proper

generation of the motor parameter file. To change the path, navigate the cursor to the right corner of the MCAT tool screen, and a symbol with a screw driver and a wrench appears. When you click this symbol, the “Application Settings Page” appears. Modify the path to the *m1_blcdc_appconfig.h* file in the “Project Path Selection” area.

6. Conclusion

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

7. Acronyms and Abbreviations

Table 19. Acronyms and abbreviations

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

8. References

These references are available on nxp.com:

- *Three-phase BLDC sensorless motor control application* (document [DRM144](#))
- *KV11 Sub-Family Reference Manual* (document [KV31P64M75RM](#))
- *KV31F Sub-Family Reference Manual* (document [KV31P100M120SF7RM](#))

- *KV4x Reference Manual* (document [KV4XP100M168RM](#))
- *KV5x Sub-Family Reference Manual* (document [KV5XP144M220RM](#))
- *NXP High-Voltage Motor Control Platform User's Guide* (document [HVPKC3PHUG](#))
- *HVP-KV31F120M User's Guide* (document [HVPKV31F120MUG](#))
- *Using FlexTimer in ACIM/PMSM Motor Control Applications* (document [AN3729](#))
- *Tips and Tricks Using PDB in Motor Control Applications on Kinetis* (document [AN4822](#))
- *Motor Control Application Tuning (MCAT) Tool for 3-Phase PMSM* (document [AN4642](#))
- *BLDC Sensorless Algorithm Tuning* (document [AN4597](#))
- *Tuning Three-Phase BLDC Motor Sensorless Control Application Using the MKV10x* (document [AN4870](#))

9. Revision History

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

Table 20. Revision history

Revision number	Date	Substantive changes
0	02/2016	Initial release.
1	09/2016	Added KE1xZ and KE1xF MCUs.
2	10/2016	CPU load and memory usage tables extended and updated.

How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address:

nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, and Kinetis are trademarks of NXP B.V.

Tower and Freedom are trademarks of NXP B.V. ARM, the ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. IAR is a trademark and IAR Embedded Workbench is a registered trademark of IAR Systems AB. All other product or service names are the property of their respective owners. All rights reserved.

© 2016 NXP B.V.

Document Number: AN5263

Rev. 2

10/2016

