BLDCRT1170B

MCUXpresso SDK Six Step Control of 3-Phase BLDC Motors

Rev. 0 — 20 February 2024 **User guide**

Document information

Information	Content
Keywords	BLDCRT1170B , BLDC, Six step control, MCAT, Motor control, Sensorless control, Speed control
Abstract	This user guide describes the implementation of the motor-control software for 3-phase Brushless DC Motors.

1 Introduction

SDK motor control example user guide describes the sensorless implementation of the motor-control software for 3-phase Brushless DC (BLDC) motor using following NXP platforms:

- i.MX RT1170-EVKB (MIMXRT1170-EVK)
- Freedom Development Platform for Low-Voltage, 3-Phase BLDC Motor Control (FRDM-MC-LVBLDC)

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

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

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

Example type	Supported motor	Possible control methods in SDK example	
		Sensorless Speed FOC	Sensored Speed FOC
bldc	Linix 45ZWN24-40 (default motor)	✓	N/A

SDK motor control example description:

- **bldc** - bldc example uses fraction arithmetic, the example contains sensorless speed control. Default motor configuration is tuned for the Linix 45ZWN24-40 motor.

The SDK motor control example contains several additional features:

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

The control software and the BLDC control theory, in general, are described in *3-Phase BLDC Sensorless Motor Control Application* (document DRM144).

2 Hardware setup

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

2.1 Linix 45ZWN24-40 motor

The Linix 45ZWN24-40 motor is a low-voltage 3-phase permanent-magnet motor with hall sensor used in BLDC applications. The motor parameters are listed in Table 2.

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 BLDC sensorless application, only the power input wires are needed.

2.2 FRDM-MC-LVBLDC

The FRDM-MC-LVBLDC low-voltage evaluation board (in a shield form factor) effectively turns the Freedom development platform or an evaluation board into a complete motor-control reference design. It is compatible with existing NXP Freedom development boards and evaluation boards. The Freedom motor-control headers are compatible with the Arduino R3 pin layout.

The FRDM-MC-LVBLDC board has a power supply input voltage of 12 VDC and does not require any hardware configuration or jumper settings. It contains no jumpers.

Figure 2. Motor-control development platform block diagram

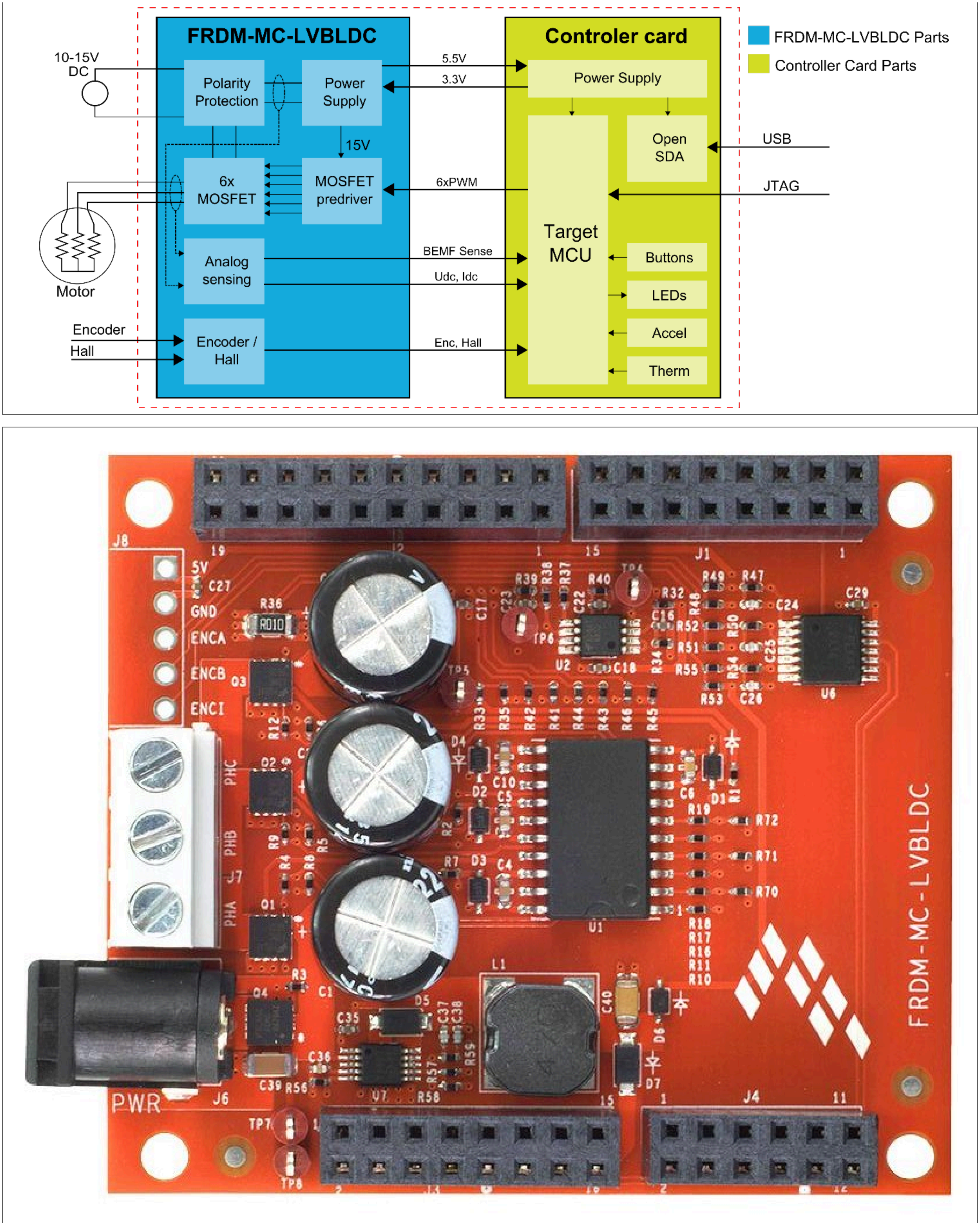


Figure 3. FRDM-MC-LVBLDC

The FRDM-MC-LVBLDC board does not require a complicated setup. For more information about the Freedom development platform, see www.nxp.com.

2.3 i.MX RT1170-EVKB

The i.MX RT1170-EVKB provides a high-performance solution in a highly integrated board. It consists of a 6-layer PCB with through hole design for better EMC performance at a low cost, and it includes key components and interfaces. The dual-core i.MX RT1170 runs on the Cortex-M7 at 1 GHz and Arm Cortex-M4 at 400 MHz, while providing best-in-class security.

Table 3. MIMXRT1170-EVKB jumper settings

Jumper	Setting	Jumper	Setting	Jumper	Setting
JP6	1-2	J53	1-2	J90	1-2
JP7	1-2	J56	2-3	J91	1-2
J14	1-2	J67	1-2	J93	1-2
J19	1-2	J68	1-2	J97	1-2
J23	1-2	J69	1-2	J98	1-2
J28	1-2	J71	1-2	J99	1-2
J38	7-8	J73	1-2	J100	1-2
J41	1-2	J79	1-2		
J49	1-2	J80	1-2		

All others jumpers are open.

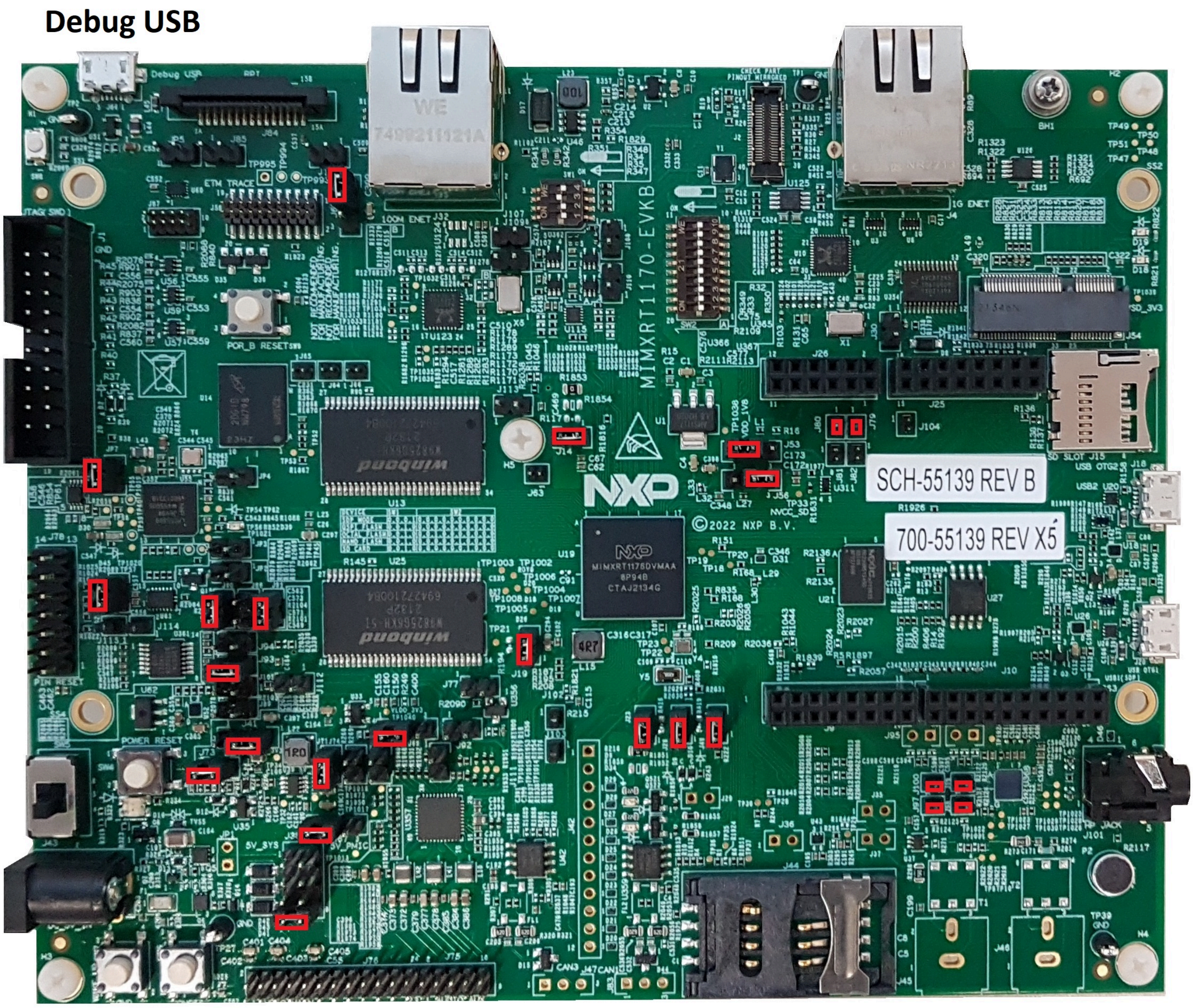


Figure 4. MIMXRT1170-EVKB board with highlighted jumper settings

The motor-control application requires removing and soldering some zero resistors for a correct connection. Remove and solder zero resistors according to Table 4.

Table 4. Add and remove resistors

Add resistors		Remove resistors	
R1841	R1845	R188	R412
R1842	R1846	R193	R1814
R1843	R1847		
R1844			

For locate resistors on the board see schematic and layout on board web page.

2.3.1 Hardware assembling

1. Connect the FRDM-MC-LVBLDC shield on top of the MIMXRT1170-EVK board.

2. Connect the 3-phase motor wires to the screw terminals (J7) on the Freedom BLDC power stage.
3. Plug the USB cable from the USB host to the Debug USB connector J86 on the EVK board.
4. Plug the 12-V DC power supply to the DC power connector on the Freedom BLDC power stage.

Note: *For a correct current measurement it is necessary to connect pin J2-9 to pin J1-16 on BLDC low-voltage platform. (On i.MX RT1170-EVKB it is pin J10-9 (GPIO_AD_30) to pin J9-16(GPIO_AD_14))*

3 Processors features and peripheral settings

This chapter describes the peripheral settings and application timing.

3.1 i.MX RT1170

The i.MX RT1170 crossover MCUs are setting speed records at 1 GHz. This ground-breaking family combines superior computing power and multiple media capabilities with ease of use and real-time functionality. The i.MX RT1170 MCU offers support over a wide temperature range and is qualified for consumer, industrial, and automotive markets.

For more information, see i.MX RT1170 Crossover MCU Family web pages.

3.1.1 RT1170 - Hardware timing and synchronization

Correct and precise timing is crucial for motor-control applications. Therefore, the motor-control-dedicated peripherals take care of the timing and synchronization on the hardware layer. In addition, you can set the PWM frequencies as a multiple of the ADC interrupt (ADC ISR) frequency.

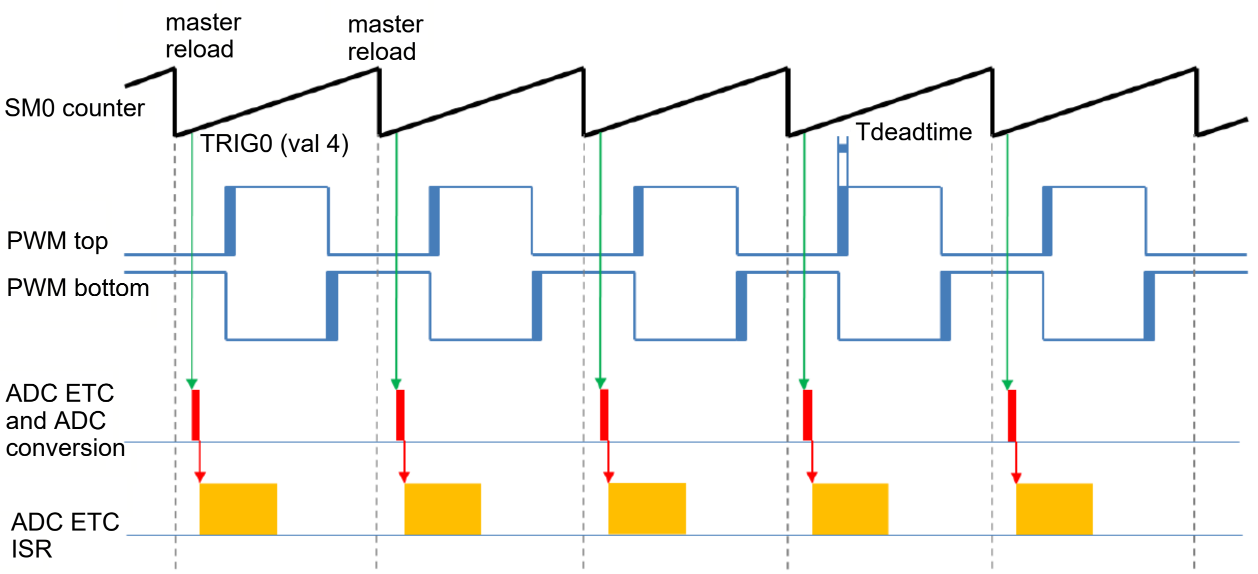


Figure 5. Hardware timing and synchronization on i.MX RT1170

- The top signal shows the eFlexPWM counter (SM0 counter). The dead time is emphasized at the PWM top and PWM bottom signals. The SM0 submodule generates the master reload at every opportunity.
- The SM0 generates trigger 0 (when the counter counts to a value equal to the VAL4) for the ADC_ETC (ADC External Trigger Control) with a delay of $T_{deatime}/2$. This delay ensures correct current sampling at the duty cycles close to 100 %.
- ADC_ETC starts the ADC conversion.
- When the ADC conversion is completed, the ADC_ETC ISR (ADC_ETC interrupt) is entered. The FOC calculation is done in this interrupt.

3.1.2 RT1170 - Peripheral settings

This section describes the peripherals used for the motor control. On i.MX RT1170, three submodules from the enhanced FlexPWM (eFlexPWM) are used for 6-channel PWM generation and two 12-bit ADCs for the phase currents and DC-bus voltage measurement. The eFlexPWM and ADC are synchronized via submodule 0 from the eFlexPWM. The following settings are located in the `mc_periph_init.c` and `peripherals.c` files and their header files.

3.1.2.1 PWM generation - PWM1

- Six channels from three submodules are used for the 3-phase PWM generation. Submodule 0 generates the master reload at event every n^{th} opportunity, depending on the user-defined macro `M1_FOC_FREQ_VS_PWM_FREQ`.
- Submodules 1 and 2 get their clocks from submodule 0.
- The counters at submodules 1 and 2 are synchronized with the master reload signal from submodule 0.
- Submodule 0 is used for synchronization with `ADC_ETC`. The submodule generates the output trigger after the PWM reload, when the counter counts to VAL4.
- Fault mode is enabled for channels A and B at submodules 0, 1, and 2 with automatic fault clearing.
  Note: *The PWM outputs are re-enabled at the first PWM reload after the fault input returns to zero.*
- The PWM period (frequency) is determined by how long the counter takes to count from INIT to VAL1. By default, INIT = -MODULO/2 and VAL1 = MODULO/2 -1.
- Dead time insertion is enabled. Define the dead time length in the `M1_PWM_DEADTIME` macro.

3.1.2.2 ADC external trigger control - ADC_ETC

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

- Both ADCs have set their own trigger chains.
- The trigger chain length is set to 2. The back-to-back ADC trigger mode is enabled.
- The SyncMode is on. In the SyncMode, ADC1 and ADC2 are controlled by the same trigger source. The trigger source is the PWM submodule 0.
- After both ADCs conversion is completed, `ADC_ETC` interrupt is enabled and serves the fast-loop algorithm.

3.1.2.3 Analog sensing - ADC1 and ADC2

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

- The ADCs operate as 12-bit with the single-ended conversion and hardware trigger selected. The ADCs are triggered from ADC_ETC by the trigger generated by the eFlexPWM.

3.1.2.4 Time event, forced commutation control - PWM1

The PWM1 submodule 3 is used for forced commutation control.

- PWM1 submodule 3 is set as free running counter and get their clocks from submodule 0. Submodule 0's clock prescaler is setup to 128.
- The PWM1 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 PWM1 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 PWM1 output compare interrupt service routine.

3.1.2.5 Peripheral interconnection for - XBARA1

The crossbar is used to interconnect the trigger from the PWM to the `ADC_ETC`.

- The `FLEXPWM2_PWM1_OUT_TRIG0_1` output trigger (generated by submodule 0) is connected to `ADC_ETC_XBAR0_TRIG0`.
- The encoder signal Phase A and Phase B are configured in `pinmux.c`.

3.1.2.6 Slow-loop interrupt generation - TMR1

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

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

3.1.2.7 FreeMASTER communication - LPUART1

Low-Power Universal Asynchronous Receiver and Transmitter (LPUART1) is used for the FreeMASTER communication between the MCU board and the PC.

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

3.2 CPU load and memory usage

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

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

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

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

Where:

CPU_{fast} = the CPU load taken by the fast loop

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

f_{fast} = the frequency of the fast-loop calculation

f_{CPU} = CPU frequency

CPU_{slow} = the CPU load taken by the slow loop

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

f_{slow} = the frequency of the slow-loop calculation

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

Table 5. Maximum CPU load (fast loop)

		debug configuration
Device	**Example**	**Speed Control**
i.MX RT1170-EVKB	bldc	34,3%

CPU load measured without defined `RAM_RELOCATION` macro. Measured CPU load and memory usage applies to the application built using IAR IDE.

Note: *The maximum CPU load is depending on executing functions from RAM or flash memory. Executing functions can be speeding up in `RTCESL_cfg.h` header file by using macro `RAM_RELOCATION`.*

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

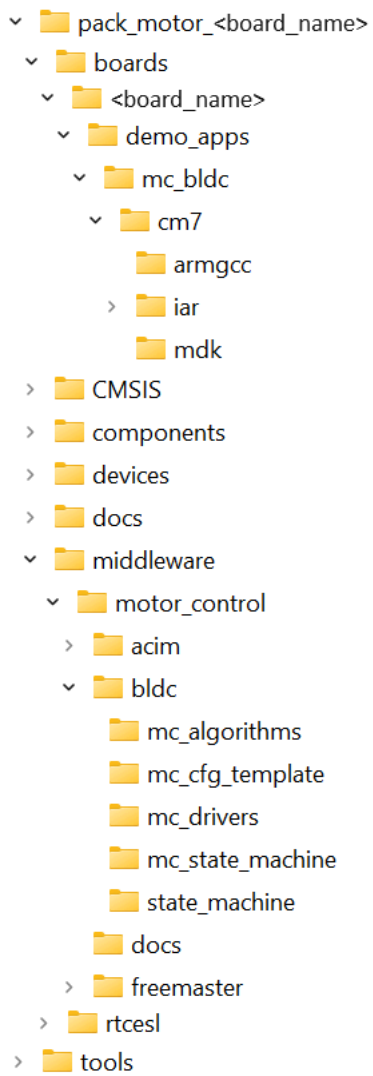
4 Project file and IDE workspace structure

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

4.1 BLDC project structure

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

Figure 6. Directory tree



The main project folder `pack_motor_<board_name>\boards\<board_name>\demo_apps\mc_bldc\<core>` contains the following folders and files:

- `iar`: for the IAR Embedded Workbench IDE.

- `armgcc`: for the GNU Arm IDE.
- `mdk`: for the uVision Keil IDE.
- `m1_bldc_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.c and .h`: contains the CPU clock setup functions.
- `mc_periph_init.c`: contains the motor-control driver peripherals initialization functions that are specific for the board and MCU used.
- `mc_periph_init.h`: header file for `mc_periph_init.c`. This file contains the macros for changing the PWM period and 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.c and .h`: port configuration files. Generate these files in the pin tool.
- `peripherals.c and .h`: MCUXpresso Config Tool configuration files.

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

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

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

- `mc_algorithms`: contains the main control algorithms used to control commutation control and speed control loop.
- `mc_cfg_template`: contains templates for MCUXpresso Config Tool components.
- `mc_drivers`: contains the source and header files used to initialize and run motor-control applications.
- `mc_state_machine`: contains the software routines that are executed when the application is in a particular state or state transition.
- `state_machine`: contains the state machine functions for the FAULT, INITIALIZATION, STOP, and RUN states.

5 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 `mc_periph_init.c` source file and the `mc_periph_init.h` header file. The definitions specified by the user are also in these files. The features provided by the functions are the 3-phase PWM generation and BEMF voltage measurement, as well as the DC-bus voltage, current and auxiliary quantity measurement.

The `mc_periph_init.h` header file provides the following macros defined by the user:

- `M1_MCDRV_ADC_PERIPH_INIT`: this macro calls ADC peripheral initialization.
- `M1_MCDRV_PWM_PERIPH_INIT`: this macro calls PWM peripheral initialization.
- `M1_MCDRV_CMP_INIT`: this macro calls comparator peripheral initialization.
- `M1_MCDRV_TMR_CMT_PERIPH_INIT`: this macro calls PWM peripheral initialization of commutation timer event0.
- `M1_PWM_FREQ`: the value of this definition sets the PWM frequency.
- `M1_FOC_FREQ_VS_PWM_FREQ`: enables you to call the fast-loop interrupt at every first, second, third, or n^{th} PWM reload. This is convenient when the PWM frequency must be higher than the maximal fast-loop interrupt.
- `M1_SPEED_LOOP_FREQ`: the value of this definition sets the speed loop frequency (TMR1 interrupt).
- `M1_PWM_DEADTIME`: the value of the PWM dead time in nanoseconds.
- `M1_FAULT_NUM`: the value of the Over Current Fault detection.
- `M1_ADC[1,2]_PH_[A..C]_CHNL`: 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. For more information about the back-EMF voltage measurement, see Three-phase BLDC sensorless motor control application (document DRM144)
- `M1_ADC[1,2]_PH_[A..C]_SIDE`: this define is used to select the ADC channel side for the BEMF voltage measurement.
- `M1_ADC[1,2]_UDCB_CHNL`: this define is used to select the ADC channel for the measurement of the DC-bus voltage.
- `M1_ADC[1,2]_UDCB_SIDE`: this define is used to select the ADC channel side for the measurement of the DC-bus voltage.
- `M1_ADC[1,2]_IDCB_CHNL`: this define is used to select the ADC channel for the measurement of the DC-bus current.
- `M1_ADC[1,2]_IDCB_SIDE`: this define is used to select the ADC channel side for the measurement of the DC-bus current.

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

- The available APIs for the ADC are:
  - `mcdrv_adcetc_t`: MCDRV ADC structure data type.
  - `void 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.
  - `void M1_MCDRV_ADC_ASSIGN_BEMF(mcdrv_adcetc_t*)`: calling this function assigns proper ADC channels for the next BEMF voltage measurement based on the commutation sector.
  - `void M1_MCDRV_CURR_CALIB_INIT(mcdrv_adcetc_t*)`: this function initializes the phase-current channel-offset measurement.
  - `void M1_MCDRV_CURR_CALIB(mcdrv_adcetc_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.

- `void M1_MCDRV_CURR_CALIB_SET(mcdrv_adcetc_t*)`: this function asserts the current measurement offset values to the internal registers. Call this function after a sufficient number of `M1_MCDRV_CURR_3PH_CALIB()` calls.
- `void M1_MCDRV_ADC_GET(mcdrv_adcetc_t*)`: this function reads and calculates the actual values of the BEMF volatage, DC-bus voltage, DC-bus current, and auxiliary quantity.

- The available APIs for the PWM are:
  - `mcdrv_pwm3ph_pwma_t`: MCDRV PWM structure data type.
  - `void M1_MCDRV_PWM_PERIPH_INIT`: this function is by default called during the PWM periphery initialization procedure invoked by the `MCDRV_Init_M1()` function.
  - `void M1_MCDRV_PWM3PH_SET_DUTY(mcdrv_pwma_pwm3ph_t*)`: this function updates the PWM phase duty cycles.
  - `void M1_MCDRV_PWM3PH_SET_PWM_OUTPUT(mcdrv_pwma_pwm3ph_t*)`: this function disables one PWM channel for BEMF measurement, set one and invert its signal generation for the other phase.
  - `bool_t M1_MCDRV_PWM3PH_FLT_GET(mcdrv_pwma_pwm3ph_t*)`: this function returns the state of the overcurrent fault flags and automatically clears the flags (if set). This function returns true when an overcurrent event occurs. Otherwise, it returns false.
- The available APIs for the asynchronous time event functions are:
  - `mcdrv_cmt_pwma_t`: MCDRV CMT_PWMA structure data type.
  - `void M1_MCDRV_TMR_CMT_PERIPH_INIT()`: this function is by default called during the CMT_PWMA Timer initialization procedure invoked by the `MCDRV_Init_M1()` function.
  - `void M1_MCDRV_TMR_CMT_GET(mcdrv_cmt_pwma_t*)`: this function read and returns the actual values of PWMA counter and value register.
  - `void M1_MCDRV_TMR_CMT_SET(mcdrv_cmt_pwma_t*, uint16_t ui16TimeNew)`: this function update PWMA value register.
- The available APIs for the comparator are:
  - `void M1_MCDRV_CMP_INIT()`: this macro calls Comparator peripheral initialization.

Note: *Not all macros are available for every motor control example type.*

6 User interface

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

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

The application can be controlled using the following two interfaces:

- The user button on the development board (controlling the demo mode):
  - MIMXRT1170-EVKB - SW7
- Remote control using FreeMASTER (Following chapter):
  - Setting a variable in the FreeMASTER Variable Watch. See chapter Section 7.4

7 Remote control using FreeMASTER

This section provides information about the tools and recommended procedures to control the sensorless Brushless DC (BLDC) application using FreeMASTER. The application contains the embedded-side driver of the FreeMASTER real-time debug monitor and data visualization tool for communication with the PC. It supports non-intrusive monitoring, as well as the modification of target variables in real time, which is very useful for the algorithm tuning. Besides the target-side driver, the FreeMASTER tool requires the installation of the PC application as well. You can download the latest version of FreeMASTER at www.nxp.com/freemaster. To run the FreeMASTER application including the MCAT tool, double-click the **`bldc.pmpx`** file located in the `middleware\motor_control\freemaster` folder. The FreeMASTER application starts and the environment is created automatically, as defined in the *.pmpx file.

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

7.1 Establishing FreeMASTER communication

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

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

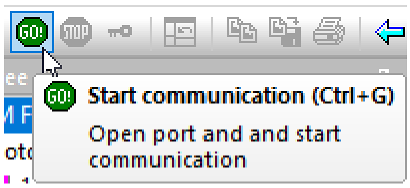


Figure 7. Green “GO” button placed in top left-hand corner

4. If the communication is established successfully, the FreeMASTER communication status in the bottom right-hand corner changes from "Not connected" to "RS-232 UART Communication; COMxx; speed=115200". Otherwise, the FreeMASTER warning pop-up window appears.

Figure 8. FreeMASTER—communication is established successfully

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

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

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

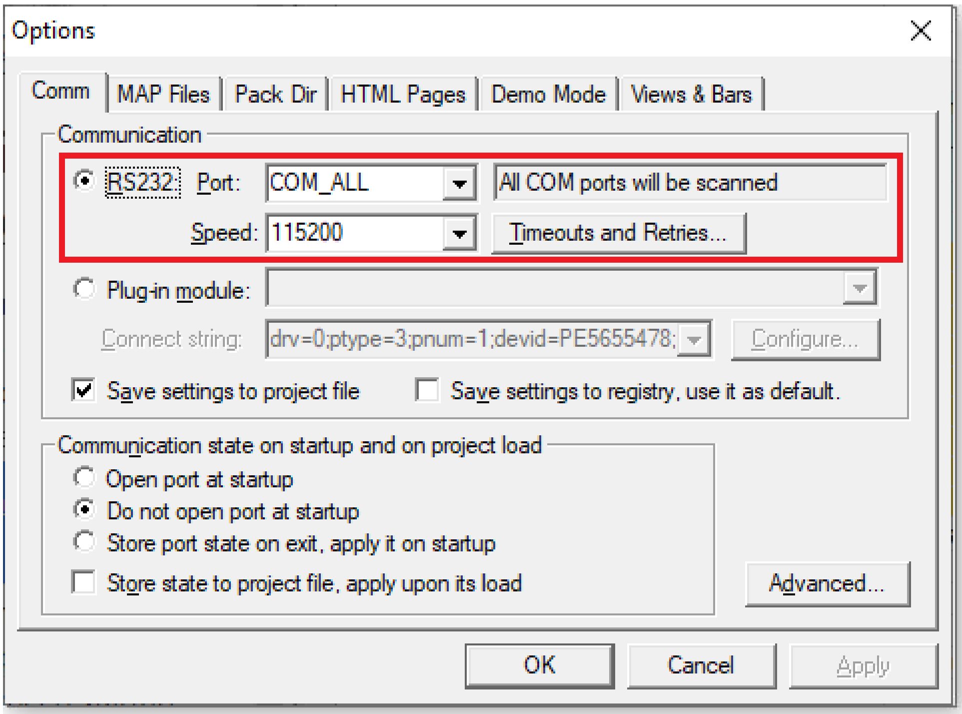


Figure 9. FreeMASTER communication setup window

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

7.2 TSA replacement with ELF file

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

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

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

To modify the example code, follow the steps below:

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

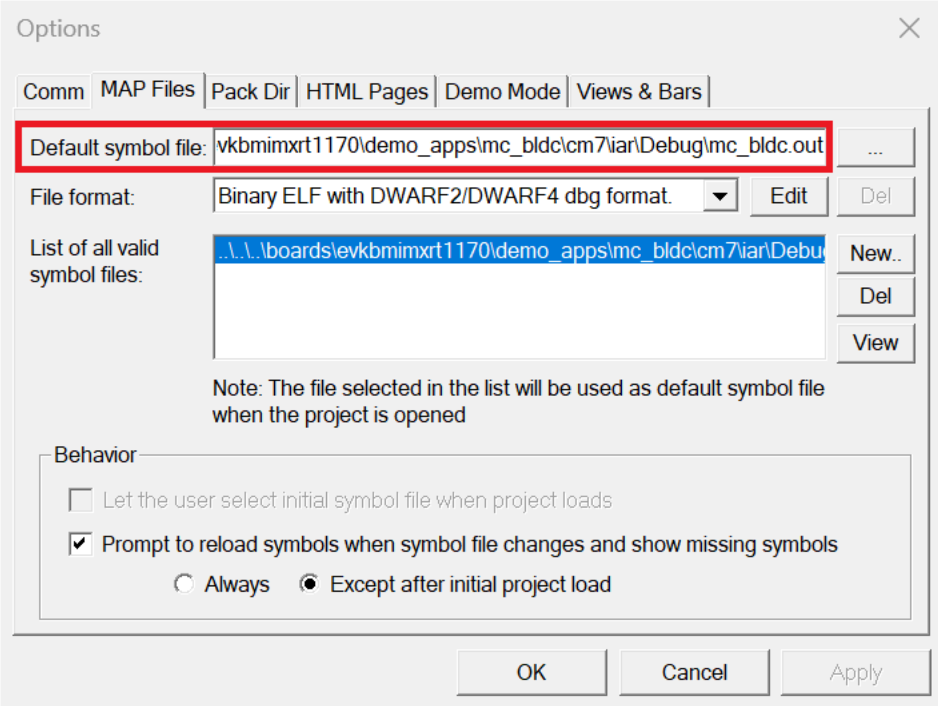


Figure 10. Default symbol file

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

For more information, check FreeMASTER User Guide.

7.3 Motor Control Aplication Tuning interface (MCAT)

The BLDC sensorless application can be easily controlled and tuned using the Motor Control Application Tuning (MCAT) plug-in for BLDC. The MCAT for BLDC is a user-friendly page, which runs within the FreeMASTER. The tool consists of the tab menu and workspace as shown in Figure 11. Each tab from the tab menu (4) represents one submodule which enables tuning or controlling different application aspects. Besides the MCAT page for BLDC, several scopes, recorders, and variables in the project tree (5) are predefined in the FreeMASTER project file to further the motor parameter tuning and debugging simplify.

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

There are three action buttons in MCAT (3):

- **Load data** - MCAT input fields (for example, motor parameters) are loaded from `mX_bldc_appconfig.h` file (JSON formatted comments). Only existing `mX_bldc_appconfig.h` files can be selected for loading. Loaded `mX_bldc_appcofig.h` file is displayed in grey field (7).
- **Save data** - MCAT input fields (JSON formatted comments) and output macros are saved to `mX_bldc_appconfig.h` file. Up to 9 files (`m1-9_bldc_appconfig.h`) can be selected. A pop-up window with the user motor ID and description appears when a different `mX_bldc_appcofig.h` file is selected. The motor ID and description are also saved in `mX_bldc_appcofig.h` as a JSON comment. The embedded code includes `m1_bldc_appcofig.h` only at single motor control application. Therefore, saving to higher indexed `mX_bldc_appconfig.h` files has no effect at the compilation stage.
- **Update target** - writes the MCAT calculated tuning parameters to FreeMASTER Variables, which effectively updates the values on target MCU. These tuning parameters are updated in MCU's RAM. To write these

tuning parameters to MCU's flash memory, `m1_bldc_appcofig.h` must be saved, code recompiled, and downloaded to MCU.

Note: *Path to `mX_bldc_appcofig.h` file also composed from Board ID value. Therefore, FreeMASTER must be connected to the target, and Board ID value read prior using Save/Load buttons.*

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

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

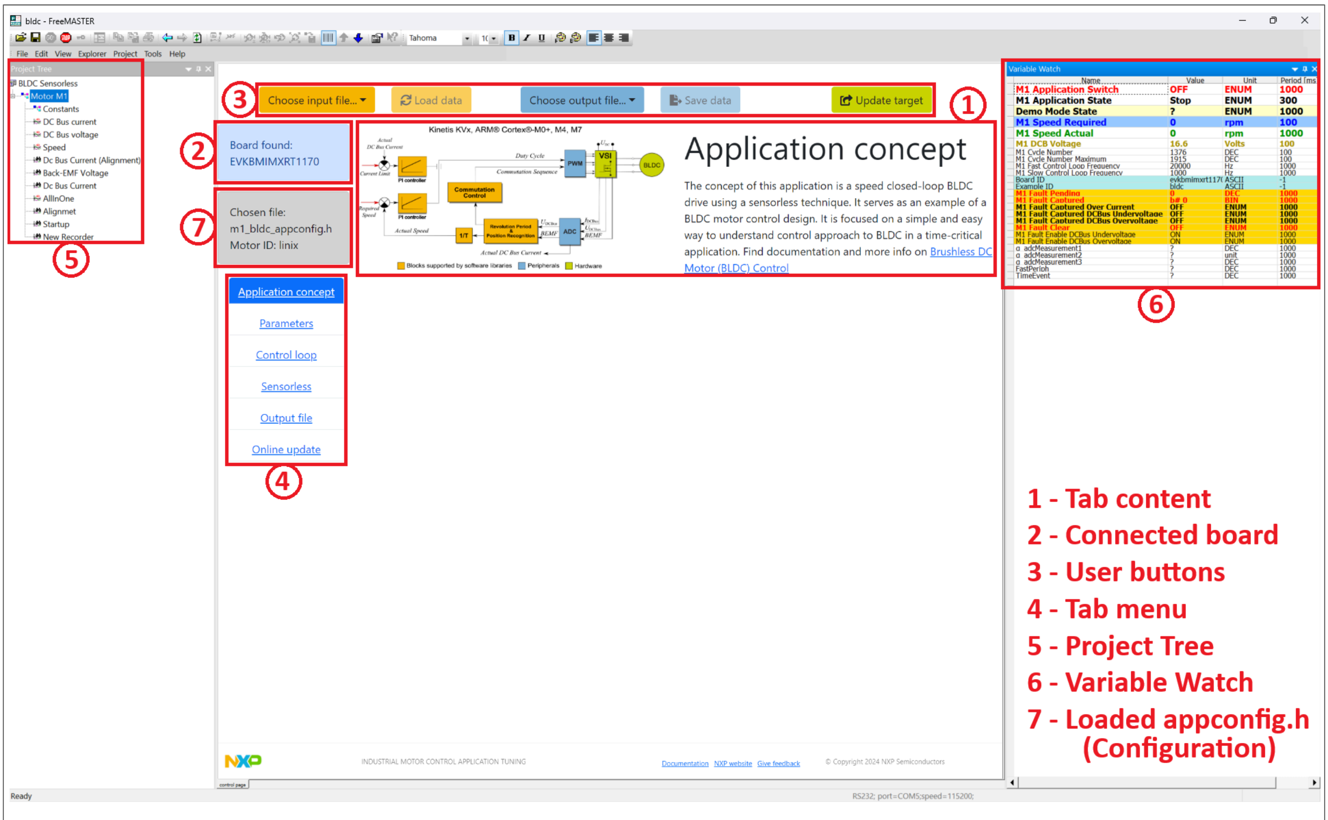


Figure 11. FreeMASTER + MCAT layout

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

- Application concept: 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 shows all the calculated constants that are required by the BLDC sensorless application. It is also possible to generate the `m1_bldc_appconfig.h` file, which is then used to preset all application parameters permanently at the project rebuild.
- Online update : this tab shows actual values of variables on target and new calculated values, which can be used to update the target variables.

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

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

7.3.1 MCAT tabs description

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

Table 6. Constants used in equations

Constant	Value	Unit
IIRxCoefsScaleType	8	-
CtrlLOOP_Ts	0.001	-
IDC_limit	3	-
pi	3.1416	-

7.3.1.1 Application concept

This tab is a welcome page with the BLDC sensorless diagram and a short description of the application.

7.3.1.2 Parameters

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

Table 7. Parameters tab inputs

MCAT group	MCAT name	Equation name	Description	Unit
Motor parameters	PP	parametersPP	Motor number of pole-pairs. Obtain from motor manufacturer.	-
	N nom	parametersNnom	Nominal motor speed. Obtain from motor manufacturer.	[rpm]
Hardware scales	I max	parametersImax	Current sensing HW scale. Keep as-is in case of standard NXP HW or recalculate according to own schematic.	[A]
	U DCB max	parametersUdcbMax	DCBus voltage sensing HW scale. Keep as-is in case of standard NXP HW or recalculate according to own schematic.	[V]
Fault limits	U DCB under	parametersUdcbUnder	DCBus under voltage fault threshold	[V]
	U DCB over	parametersUdcbOver	DCBus over voltage fault threshold	[V]
	N over	parametersNover	Over speed fault threshold	[rpm]
	N min	parametersNmin	Minimal closed loop speed. When the required speed ramps down under this threshold, the motor control state machine	[rpm]

Table 7. Parameters tab inputs*...continued*

MCAT group	MCAT name	Equation name	Description	Unit
			goes to freewheel state where top and bottom transistors are turned off and motor speeds down freely. Applies only for sensorless operation.	
Application scales	U DCB IIR F0	parametersUdcbIrrF0	Cut-off frequency of DCBus IIR filter	[Hz]
	I DCB IIR F0	parametersIdcbIrrF0	Cut-off frequency of DCBus IIR filter	[Hz]
	Calibration duration	parametersCalibDuration	ADC (phase current offset) calibration duration. Done every time transitioning from STOP to RUN.	[sec]
	Fault duration	parametersFaultDuration	After fault condition disappears, wait defined time to clear pending faults bitfield and transition to STOP state.	[sec]
	N max	parametersNmax	Application speed scale. Keep about 10 % margin above *N over*.	[rpm]
	Ke	parametersKe	Motor electrical constant. Obtain from motor manufacturer or use the Ke identification and then fill manually.	[V.sec/rad]
Alignment	Align voltage	parametersAlignVoltage	Motor alignment voltage.	[V]
	Align current	parametersAlignCurrent	Motor alignment current.	[A]
	Align duration	parametersAlignDuration	Motor alignment duration.	[sec]

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

- `M1_ALIGN_CURRENT` = parametersAlignCurrent / parametersImax
- `M1_ALIGN_DURATION` = parametersAlignDuration / 0.001
- `M1_CALIB_DURATION` = parametersCalibDuration / constants.CtrlLOOP_Ts
- `M1_FAULT_DURATION` = parametersFaultDuration / constants.CtrlLOOP_Ts
- `M1_I_DCB_LIMIT` = constants.IDC_limit / parametersImax
- `M1_U_DCB_UNDERVOLTAGE` = parametersUdcbUnder / parametersUdcbMax
- `M1_U_DCB_OVERVOLTAGE` = parametersUdcbOver / parametersUdcbMax
- `M1_N_NOM` = parametersNnom / parametersNmax
- `M1_N_MIN` = parametersNmin / parametersNmax
- `M1_UDCB_IIR_B0` = 4 * ((2 * Math.PI * UDCB_IIR_cutoff_freq / controlLoopPwmFreq) / (2 + (2 * Math.PI * UDCB_IIR_cutoff_freq / controlLoopPwmFreq))) / constants.IIRxCoefsScaleType
- `M1_UDCB_IIR_B1` = 4 * ((2 * Math.PI * UDCB_IIR_cutoff_freq / controlLoopPwmFreq) / (2 + (2 * Math.PI * UDCB_IIR_cutoff_freq / controlLoopPwmFreq))) / constants.IIRxCoefsScaleType
- `M1_UDCB_IIR_A1` = (-4) * ((2 * Math.PI * UDCB_IIR_cutoff_freq / controlLoopPwmFreq - 2) / (2 + (2 * Math.PI * UDCB_IIR_cutoff_freq / controlLoopPwmFreq))) / constants.IIRxCoefsScaleType
- `M1_IDCB_IIR_B0` = 4 * ((2 * Math.PI * IDCB_IIR_cutoff_freq / controlLoopPwmFreq) / (2 + (2 * Math.PI * IDCB_IIR_cutoff_freq / controlLoopPwmFreq))) / constants.IIRxCoefsScaleType

- `M1_IDCB_IIR_B1` = 4 * ((2 * Math.PI * IDCB_IIR_cutoff_freq / controlLoopPwmFreq) / (2 + (2 * Math.PI * IDCB_IIR_cutoff_freq / controlLoopPwmFreq))) / constants.IIRxCoefsScaleType
- `M1_IDCB_IIR_A1` = (-4) * ((2 * Math.PI * IDCB_IIR_cutoff_freq / controlLoopPwmFreq - 2) / (2 + (2 * Math.PI * IDCB_IIR_cutoff_freq / controlLoopPwmFreq))) / constants.IIRxCoefsScaleType

7.3.1.3 Control loop

This tab enables modification of speed ramp parameters and control loop parameters like control loop output limits and PI controllers. MCAT group and MCAT name help to locate the parameter in MCAT layout. Equation name represents the input parameter in equations bellow.

Table 8. Control loop tab input

MCAT group	MCAT name	Equation name	Description	Unit
Loop sample time	Sample time	controlLoopSampleTime	Slow control loop period. This disabled value is read from target via FreeMASTER because application timing is set in embedded code by peripherals setting. This value is accessible only if target is not connected and value cannot be obtained from target.	[sec]
	PWM freq	controlLoopPwmFreq	PWM frequency	[Hz]
Control loop output limits	Upper limit	controlLoopLimitHigh	Maximal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]
	Lower limit	controlLoopLimitLow	Minimal required Q-axis current (Speed controller's output). Q-axis current limitation equals to motor torque limitation.	[A]
Speed ramp	Inc up	controlLoopIncUp	Required speed maximal acceleration	[rpm/sec]
	Inc down	controlLoopIncDown	Required speed maximal acceleration	[rpm/sec]
Speed PI controller constants	Speed Loop Kp	controlLoopSLKp	Speed Controller Proportional constant in time domain	-
	Speed Loop Ki	controlLoopSLKi	Speed Controller Integration constant in time domain	-
Torque PI controller constants	Torque Loop Kp	controlLoopTLKp	Torque Controller Proportional constant in time domain	-
	Torque Loop Ki	controlLoopTLKi	Torque Controller Integration constant in time domain	-

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

- parametersWmax = 2 * Math.PI * parametersPP * parametersNmax / 60
- `M1_SPEED_LOOP_KP_GAIN` = controlLoopSLKp * parametersWmax / parametersImax
- `M1_SPEED_LOOP_KI_GAIN` = controlLoopSLKi * parametersWmax / parametersImax * constants.CtrlLOOP_Ts

- `M1_SPEED_RAMP_UP` = controlLoopIncUp / 60 * parametersPP * 2 * Math.PI /parametersWmax * constants.CtrlLOOP_Ts
- `M1_SPEED_RAMP_DOWN` = controlLoopIncDown / 60 * parametersPP * 2 * Math.PI /parametersWmax * constants.CtrlLOOP_Ts
- `M1_CTRL_LOOP_LIM_HIGH`= controlLoopLimitHigh / 100
- `M1_CTRL_LOOP_LIM_LOW` = controlLoopLimitLow / 100
- `M1_TORQUE_LOOP_KP_GAIN` = controlLoopTLKp * parametersImax / parametersUdcbMax
- `M1_TORQUE_LOOP_KI_GAIN` = controlLoopTLKi * parametersImax / parametersUdcbMax * constants.CtrlLOOP_Ts

7.3.1.4 Sensorless

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

Table 9. Sensorless tab input

MCAT group	MCAT name	Equation name	Description	Unit
Sensorless parameters	Timer freq	sensorlessTimerFreq	Forced commutation timer frequency, calculated as Timer input clock / Prescale factor (128)	[Hz]
	Freewheel duration	sensorlessFreewheelTime	Motor Freewheel period from any motor speed	[sec]
Open loop start-up parameters	OL speed lim	sensorlessOLspeedLim	Open Loop Start-up minimal speed to switch to close-loop control of nominal speed	[rpm]
	Cmt count	sensorlessCmtCount	Open Loop Start-up commutation number of nominal speed.	[#]
	1st cmt period	sensorlessCmtPeriod	First commutation period	[sec]
Commutation parameters	Time off	sensorlessTimeOff	Time off after commutation	[%]
	Integ thr corr.	sensorlessIntegThrCorr	Integration threshold correction constant with range	[%]

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

- `M1_FREEWHEEL_T_LONG` = sensorlessFreewheelTime / 0.001
- `M1_FREEWHEEL_T_SHORT` = sensorlessFreewheelTime / 2 / 0.001
- `M1_N_START_TRH` = sensorlessOLspeedLim / parametersNmax
- `M1_STARTUP_CMT_PER` = sensorlessCmtPeriod * sensorlessTimerFreq
- `M1_CMT_T_OFF` = sensorlessTimeOff / 100
- `M1_SPEED_SCALE_CONST` = sensorlessTimerFreq * 60 / (parametersNmax * parametersPP)
- `M1_CMT_PER_MIN` = sensorlessTimerFreq / (parametersNmax * parametersPP / 10)
- `M1_START_CMT_ACCELER` = Math.pow((60 / (sensorlessOLspeedLim * parametersPP * 6) / sensorlessCmtPeriod), (1 / (sensorlessCmtCount - 1)))
- temp1 = 1/4 * parametersKe * (2 * Math.PI * parametersPP * parametersNnom / 60)
- temp2 = 1/2 / (parametersNnom * parametersPP * 6 / 60) * controlLoopPwmFreq * temp1
- `M1_INTEG_TRH` = Math.round(temp2 / parametersUdcbMax * 32768 * sensorlessIntegThrCorr / 100)

7.4 Motor Control Modes - How to run motor

In the "Project Tree", you can choose Speed control using the appropriate FreeMASTER tabs. To turn on or off the application, use "M1 Application Switch" variable. Set/clear "M1 Application Switch" variable also enables/ disables all PWM channels.

Before motor starts, several conditios have to be completed:

1. Connected power supply to the inverter with the correct voltage value.
2. No pending fault. Check variable "M1 Fault Pending" in "Motor M1" project tree subblock. If there is some value, first remove the cause of the fault, or disable fault checking. (for example in variable "M1 Fault Enable Undervoltage")

7.4.1 Speed control

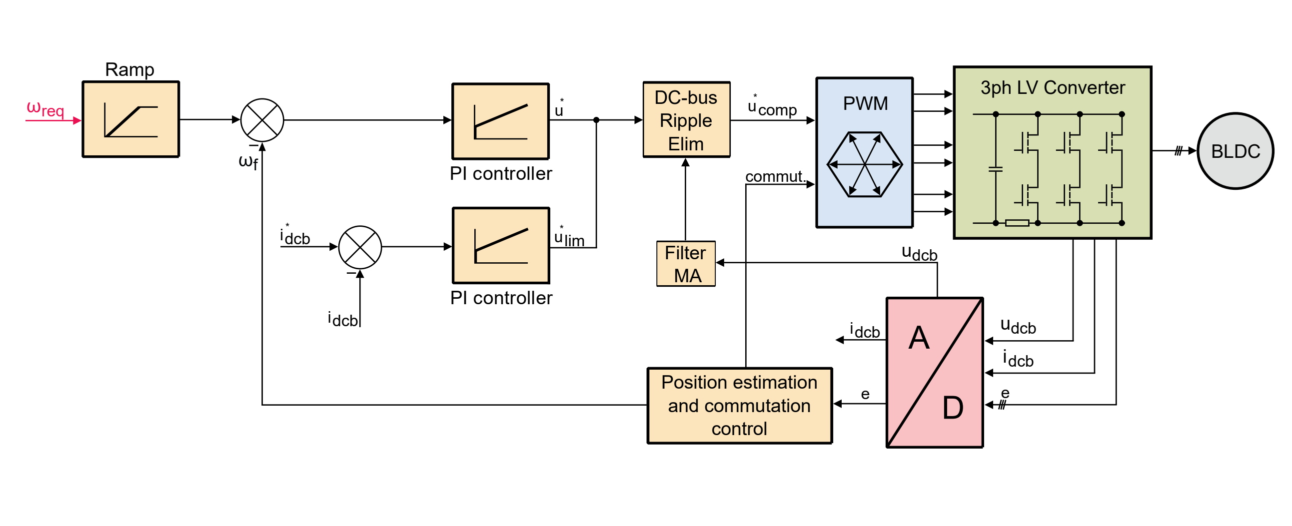


Figure 12. Control of Brushless DC motor

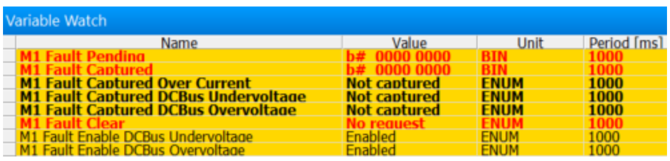
For run motor in speed control, follow these steps:

1. Switch project tree subblock on "Speed".
2. In variable "M1 Speed Required" set the required speed. (i.e. 1000rpm). The motor automatically starts spinning.
3. Observe motor speed, Back-EMF Voltage and other graphs predefined in subblock scopes and recorders.

7.5 Faults explanation

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

Figure 13. Faults in variable watch located in "Motor M1" subblock



Variable Watch

Name	Value	Unit	Period [ms]
M1 Fault Pending	b# 0000 0000	BIN	1000
M1 Fault Captured	b# 0000 0000	BIN	1000
M1 Fault Captured Over Current	Not captured	ENUM	1000
M1 Fault Captured DCBus Undervoltage	Not captured	ENUM	1000
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM	1000
M1 Fault Clear	No request	ENUM	1000
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM	1000
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM	1000

7.5.1 Variable "M1 Fault Pending"

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

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

Figure 14. Undervoltage fault is indicated (pending)

Variable Watch

Name	Value	Unit	Period [ms]
M1 Fault Pending	b# 0000 0010	BIN	1000
M1 Fault Captured	b# 0000 0010	BIN	1000
M1 Fault Captured Over Current	Not captured	ENUM	1000
M1 Fault Captured DCBus Undervoltage	Captured	ENUM	1000
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM	1000
M1 Fault Clear	No request	ENUM	1000
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM	1000
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM	1000

7.5.2 Variable "M1 Fault Captured"

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

Variable Watch

Name	Value	Unit	Period [ms]
M1 Fault Pending	b# 0000 0000	BIN	1000
M1 Fault Captured	b# 0000 0010	BIN	1000
M1 Fault Captured Over Current	Not captured	ENUM	1000
M1 Fault Captured DCBus Undervoltage	Captured	ENUM	1000
M1 Fault Captured DCBus Overvoltage	Not captured	ENUM	1000
M1 Fault Clear	No request	ENUM	1000
M1 Fault Enable DCBus Undervoltage	Enabled	ENUM	1000
M1 Fault Enable DCBus Overvoltage	Enabled	ENUM	1000

Figure 15. Undervoltage fault is captured

7.5.3 Variable "M1 Fault Enable"

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

Note: *The overcurrent fault cannot be disabled.*

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

7.6 Initial motor parameters and harware configuration

Motor control examples contain two or more configuration files: `m1_bldc_appconfig.h`, `m2_bldc_appconfig.h`, and so on. Each contains constants tuned for the selected motor (Linix 45ZWN24-40 or Teknic M-2310P for the Freedom development platform and Mige 60CST-MO1330 for the High-voltage platform). The initial motor parameters and the hardware configuration (inverter) are to MCAT loaded from `m1_bldc_appconfig.h` configuration file. There are tree ways to change motor configuration corresponding to the connected motor.

1. The first way is rename the configuration file:
   - In the project example folder, find configuration file to be used.
   - Rename this configuration file to `m1_bldc_appconfig.h`.
   - Rebuild project and load the code to the MCU.
2. The second way is to change motor configuration, as described in Section 7.3.
3. The last way is change motor and hardware parameters manually:
   - Open the BLDC control application FreeMASTER project containing the dedicated MCAT plug-in module.
   - Select the "Parameters" tab.
   - Specify the parameters manually. All parameters provided in Table 10 are accessible.

Table 10. MCAT 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

   - Set the hardware scales—the modification of these two fields is not required when a reference to the standard power stage board is used. These scales express the maximum measurable current and voltage analog quantities.
   - Check the fault limits—these fields are calculated using the motor parameters and hardware scales (see Table 11).

Table 11. Fault limits

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

   - Check the application scales—these fields are calculated using the motor parameters and hardware scales (see Table 12).

Table 12. 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 (Unom * 60) / (2*π*pp*Nnom)	0.0001-1

- Check the alignment parameters—these fields are calculated using the motor parameters and hardware scales. The parameters express the required voltage value applied to the motor during the rotor alignment and its duration.
- **Table 13. Alignment parameters**

Parameter	Units	Description	Typical range
Align current	[A]	Align current	0.1 - Iph nom
Align duration	[sec]	Align duration	0.2-10

7.7 Control parameters tuning

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 tuning phases are described in the following sections.

7.7.1 Alignment tuning

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

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

7.7.2 Open loop start-up 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 16).

1. Select the “Sensorless” tab in the FreeMASTER project tree.
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 (1stcmt period). It is the time duration between zero speed and first commutation. This parameter is responsible for acceleration during start-up.

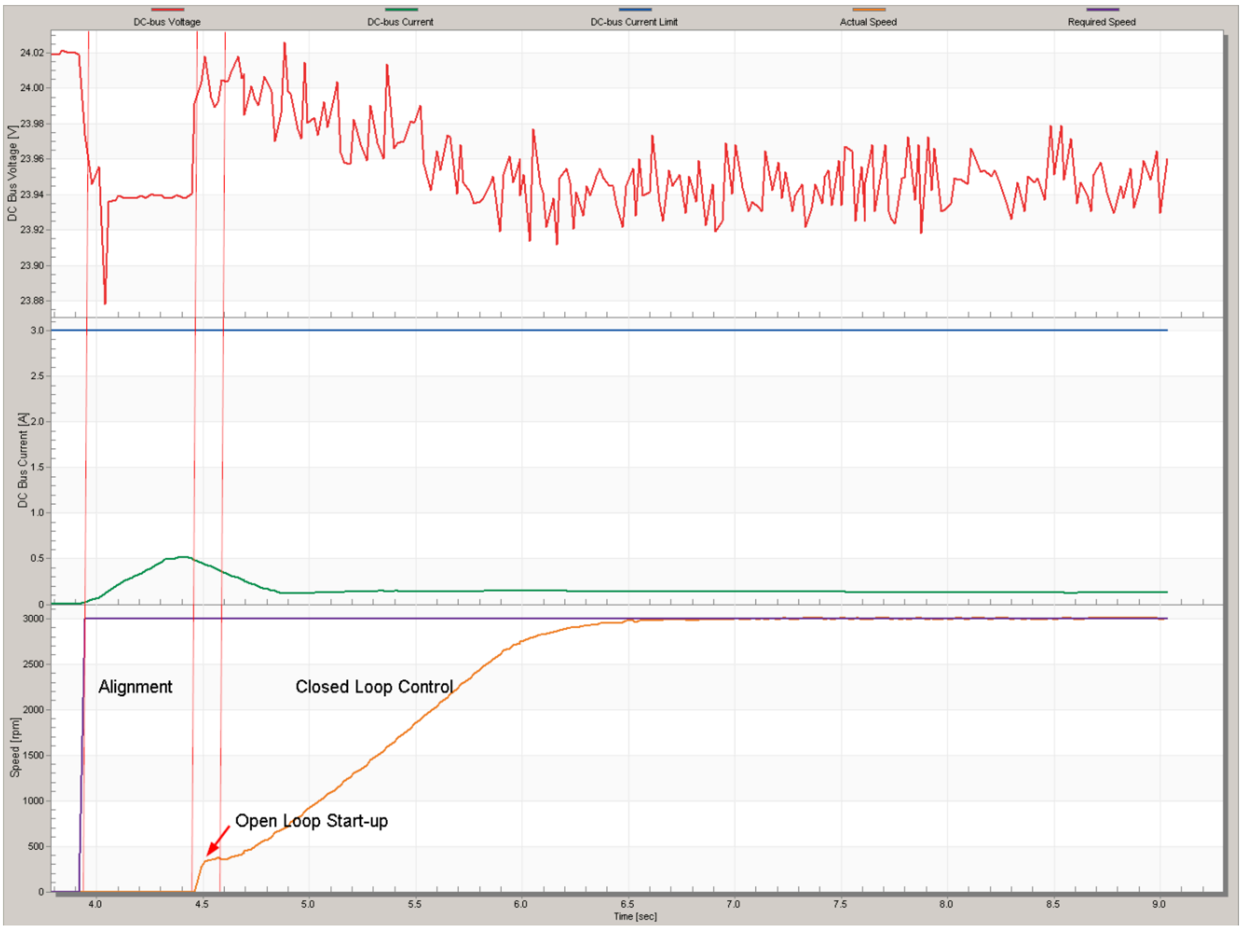
5. Click the “Update target” button to write the changes to the MCU.
6. Select “AllInOne” oscilloscope in project tree and turn the application on. Observe the waveform response in the oscilloscope:
   - 

Figure 16. Motor Open-Loop Startup

7.7.3 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 overvoltage 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 Figure 17). The ramp down increment is set to 2000rpm/sec, while the up increment is set to 500 rpm/sec.

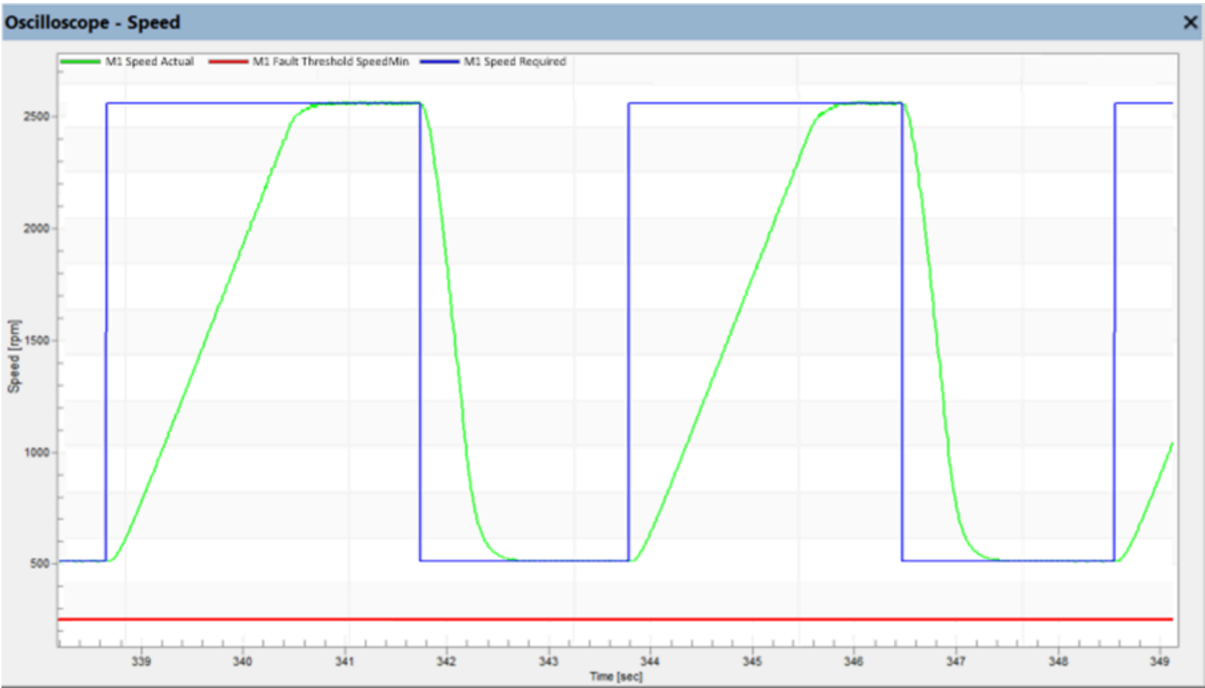


Figure 17. Speed ramp

7.7.4 Current and Speed PI controller 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:

$$K_{Ifrac} = K_{Is} * T_s * \frac{I_{max}}{U_{max}} => K_{Is} = \frac{K_{Ifrac} * U_{max}}{T_s * I_{max}} \quad (4)$$

$$K_{Pfrac} = K_{Ps} * \frac{I_{max}}{U_{max}} => K_{Ps} = \frac{K_{Pfrac} * U_{max}}{I_{max}} \quad (5)$$

Speed PI controller constants:

$$K_{Ifrac} = K_{Is} * T_s * \frac{N_{max}}{I_{max}} => K_{Is} = \frac{K_{Ifrac} * I_{max}}{T_s * N_{max}} \quad (6)$$

$$K_{Pfrac} = K_{Ps} * \frac{N_{max}}{I_{max}} => K_{Ps} = \frac{K_{Pfrac} * I_{max}}{N_{max}} \quad (7)$$

Where:

K_P = PI controller proportional constant

K_I = PI controller integration constant

T_S = Current/Speed loop sample time period

I_{max} = Phase current measurement scale

U_{max} = DCBus voltage measurement scale

N_{max} = Mechanical speed measurement maximum limit

$K_{X\ frac}$ = Fraction coefficient

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.

1. Select the *"Speed"* option from the FreeMASTER project tree.
2. Select the *"Control loop"* tab.
3. Tune the proportional gain:
   - Set the *"Speed Loop Ki"* integral gain to 0.
   - Set the speed ramp to 1000 rpm/sec (or higher).
   - Run the motor at a convenient speed (about 30 % of the nominal speed).
   - Set a step in the required speed to 40 % of N_{nom}.
   - Adjust the proportional gain *"Speed Loop Kp"* until the system responds to the required value properly and without any oscillations or excessive overshoot:
     - If the *"Speed Loop Kp"* field is set low, the system response is slow.
     - If the *"Speed Loop Kp"* field is set high, the system response is tighter.
     - When the *"Speed Loop Ki"* field is 0, the system most probably does not achieve the required speed.
     - To apply the changes to the MCU, click the *"Update Target"* button.
4. Tune the integral gain:
   - Increase the *"Speed Loop Ki"* field slowly to minimize the difference between the required and actual speeds to 0.
   - Adjust the *"Speed Loop Ki"* field such that you do not see any oscillation or large overshoot of the actual speed value while the required speed step is applied.
   - To apply the changes to the MCU, click the *"Update target"* button.
5. The example waveforms with the correct and incorrect settings of the speed loop parameters are shown in the following figures:
   - The *"Speed Loop Ki"* value is low and the *"Speed Actual Filtered"* does not achieve the *"Speed Ramp"*.

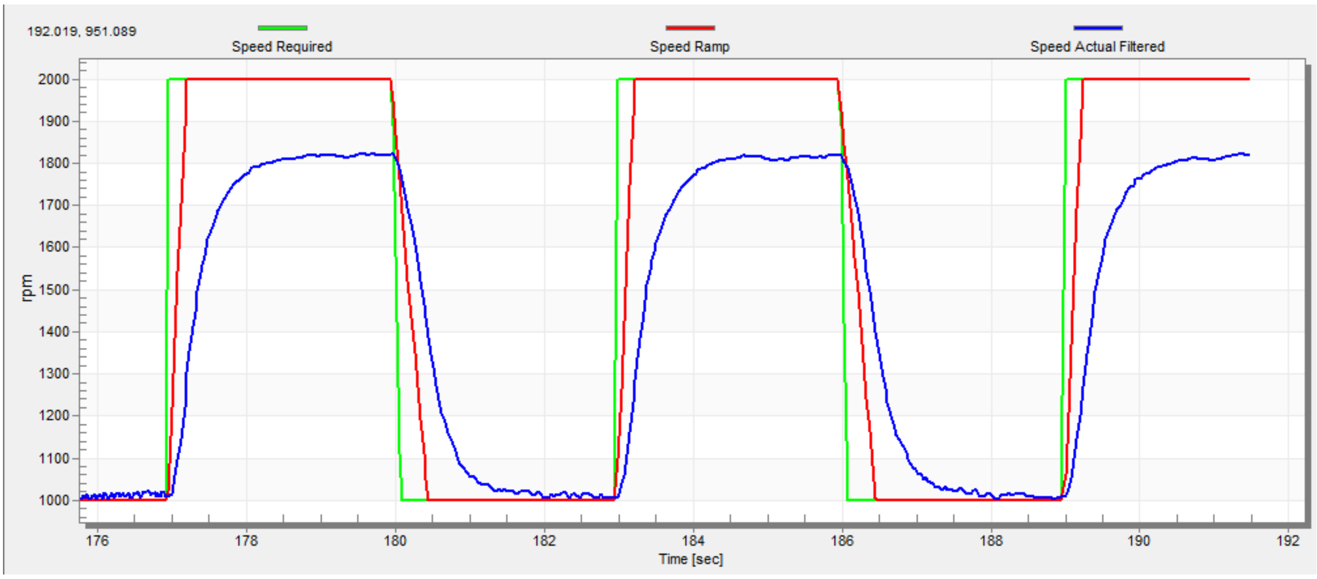


Figure 18. Speed controller response—SL_Ki value is low, Speed Ramp is not achieved

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

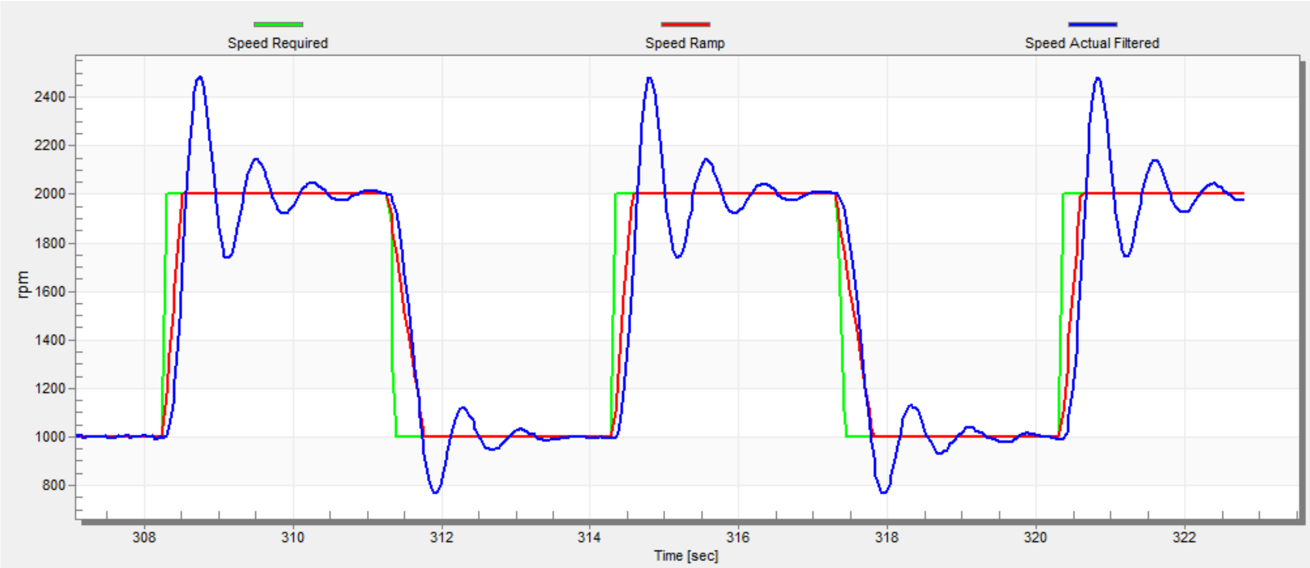


Figure 19. Speed controller response—SL_Kp value is low, Speed Actual Filtered greatly overshoots

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

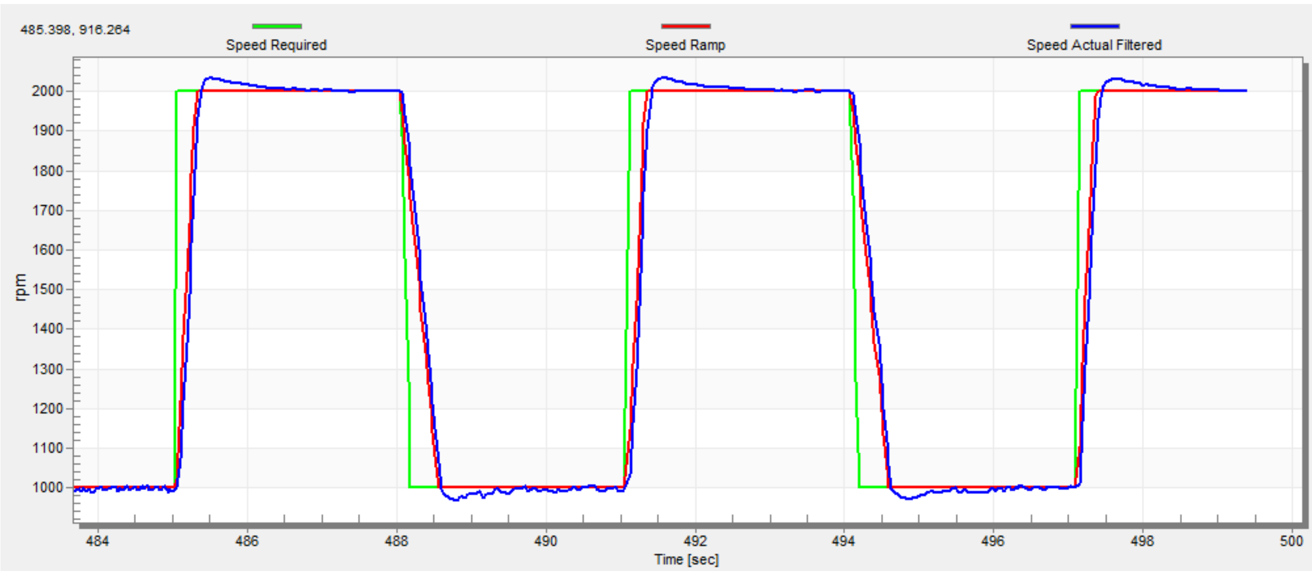


Figure 20. Speed controller response—speed loop response with a small overshoot

8 Conclusion

This application note describes the implementation of the sensorless 3-phase BLDC motor control. The motor control software is implemented on NXP MIMXRT1170-EVKB board with the FRDM-MC-LVBLDC NXP Freedom development platform. The hardware-dependent part of the control software is described in Section 2. The motor-control application timing, and the peripheral initialization are described in Section 3. The motor user interface and remote control using FreeMASTER are described in Section 6.

9 Acronyms and abbreviations

Table 14 lists the acronyms and abbreviations used in this document.

Table 14. Acronyms and abbreviations

Acronym	Meaning
ADC	Analog-to-Digital Converter
ADC_ETC	ADC External Trigger Control
AN	Application Note
BLDC	Brushless DC motor
CPU	Central Processing Unit
DC	Direct Current
DRM	Design Reference Manual
GPIO	General-Purpose Input/Output
MCAT	Motor Control Application Tuning tool
MCDRV	Motor Control Peripheral Drivers
MCU	Microcontroller
PI	Proportional Integral controller
PWM	Pulse-Width Modulation
TMR	Quad Timer
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
XBAR	Inter-Peripheral Crossbar Switch

10 References

These references are available on www.nxp.com:

- *Three-phase BLDC sensorless motor control application* (document DRM144)
- *BLDC Sensorless Algorithm Tuning* (document AN4597)

11 Useful links

- MCUXpresso SDK for Motor Control www.nxp.com/sdkmotorcontrol
- Motor Control Application Tuning (MCAT) Tool
- i.MX RT Crossover MCUs
- FRDM-MC-BLDC Freedome Development Platform
- MCUXpresso IDE - Importing MCUXpresso SDK
- MCUXpresso Config Tool
- MCUXpresso SDK Builder (SDK examples in several IDEs)
- Model-Based Design Toolbox (MBDT)

12 Revision history

Section 12 summarizes the changes done to the document since the initial release.

Table 15. Revision history

Revision number	Date	Substantive changes
0	Feb 2024	Initial release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Tables

Figures

Contents

Date of release: 20 February 2024
Document identifier: BLDCRT1170B