NXP Semiconductors

User Guide

Document identifier: PMSMSAFEUG

Rev. 0, 31 May 2022

Three-phase PMSM Pump Reference Safety Software Design User Guide

Contents

Chapter 1
Introduction

This user guide describes three-phase permanent magnet synchronous motor pump (PMSM) reference design optimized for HVP-MC3PH-LITE hardware equipped with Kinetis V series MCU. The aim of the reference design is to help customers develop motor control solutions with IEC60730 class B safety features intended for controlled heating systems, electric circulation pumps, service water installation, and other devices used in industrial. The supported control methods are listed in Table 1.

The document consists of several parts. The first part talks about hardware and its settings. The next part is about MCU and its peripheral settings. After this part follows the software part, describing safety implementation, state machine, and all control algorithms including current reconstruction and space vector modulation. Chapter user interface has part aimed for remote control using real-time debug monitor and data visualization tool FreeMASTER. The next chapter shows the project and IDE workspace structure. The end of the document talks about tuning of the software for using with customer motor.

The sensorless control software and the PMSM control theory in general are described in design reference manual DRM148 Sensorless PMSM Field-Oriented Control (FOC). The NXP IEC 60730 Class B Safety Library and Real-Time Control Embedded Software Motor Control and Power Conversion Libraries, also known as RTCESL, are used in the reference design. For more information, visit www.nxp.com/motorcontrol_pmsm.

Table 1. Supported control methods

		Supported control methods in SDK example					
Device	Default motor	Scalar	Voltage	Current FOC (Torque)	Sensorless Speed FOC	Sensored Speed FOC	Servo control (Position FOC)
HVP3PH_LITE	PMSM	✓	✓	✓	✓	X	X

1.1 Hardware setup

The *Three-phase PMSM Pump Reference Design* application was designed, implemented, and verified for the HVP3PH_LITE hardware development platform and the reference PMSM motor. Both are described in next sections.

1.2 Inverter HVP3PH_LITE

The HVP3PH_LITE is high-voltage platform intended for using with 3-phase motors roughly up to 60 Watts. Development platform board has the power supply input voltage of 230 VAC. The output current is up to 0.5 A RMS. The inverter itself is realized by advanced Motion System-on-module providing a fully-featured, high-performance inverter output stage for AC motors. These modules integrate optimized gate drive of the built-in MOSFETs (FRFET technology) to minimize EMI and losses, while also providing multiple on-module protection features including under-voltage lockouts and thermal monitoring.

The board is mounted with MKV10Z32VLF7 Kinetis KV10 MCU build on Arm® Cortex®-M0+ core running at 75 MHz with 32 kB Flash and 8 kB SRAM.

Board has integrated input for medium temperature sensor and PWM input intended for reading the external pump speed command.

The block diagram of this complete NXP motor-control development kit is shown in Figure 1. The top of the HVP3PH_LITE is shown in Figure 2.

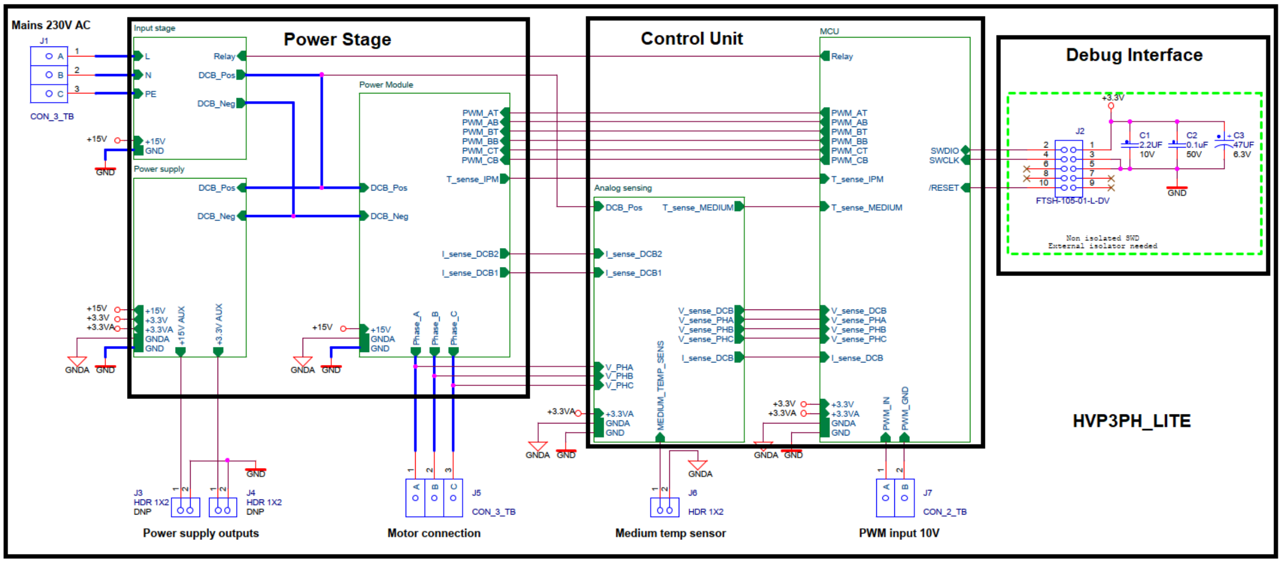


Figure 1. HVP3PH_LITE block diagram

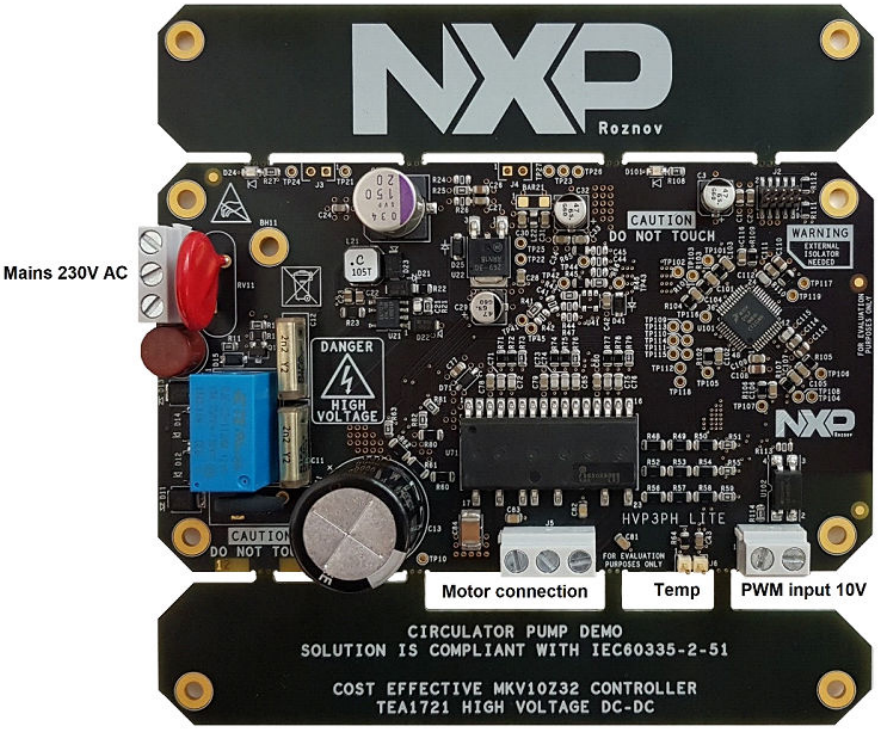


Figure 2. HVP3PH_LITE

1.2.1 Medium temperature sensor

Platinum temperature sensor PTFx102xxxx is used as a default temperature sensor of pump medium. The sensor is connected to connector J6.

Chapter 2
External command PWM input 10 V

PWM input is used for controlling speed of the circulation pump (i.e. the external command PWM input). The input signal is connected to J7. By default, the valid PWM signal has duty cycle from 10 % to 90 % with frequency in range from 200 Hz to 2 kHz.

2.1 Reference permanent magnet synchronous motor

The application is designed for using permanent magnet synchronous motor (PMSM). The application is tuned for motor with following parameters:

Table 2. Default PMSM motor parameters

Characteristic	Symbol	Value	Units
Rated voltage	V_t	230	V
Rated speed	-	1000	rpm
Rated power	P	52	W
Current RMS	I_{RMS}	0.45	A
Number of pole-pairs	pp	3	-

Parameters of the customer motor can be obtained by several ways. The two common ways are using parameters from the motor data sheet or manual measurement (see document AN4680). The next option is using MCAT what is part of all SDK motor control examples. MCAT is used for automated PMSM parameter identification. (see document AN4896)

2.2 Hardware assembling

- Connect the three-phase motor wires to the screw terminals J5 (Motor connection) on the HVP3PH_LITE.
- Connect the Medium temp sensor to J6 connector on the HVP3PH_LITE.

Connect external command PWM source to the screw terminals J7 (PWM input 10 V) on the HVP3PH_LITE. Make sure its duty cycle and frequency matches limits described in Section 2.1.2.

- Connect J-link debugger via external isolator to the SWD connector J2 on the HVP3PH_LITE.
- Plug the 230 V AC to the AC power connector J1 on the HVP3PH_LITE.

Chapter 3
MCU features and peripheral settings

This chapter describes the peripheral settings and application timing.

3.1 Kinetis KV1x family

The KV10Z MCU is highly scalable member of the Kinetis V series and provides 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).

3.2 Peripheral settings

In this chapter there are described peripherals settings. On KV10Z there is a 6-channel FlexTimer (FTM) used for 6-channel PWM generation and two 16-bit SAR ADCs for phase currents, DC-bus voltage, temperatures, and reference voltage measurement. The FTM and ADC are synchronized via Programmable Delay Block (PDB). There is also one channel from another independent FTM used for slow loop interrupt generation. Motor control peripheral settings are located in *m1_periph_init.c* and *m1_periph_init.h* file. External control peripheral settings are located in *app_periph_init.c* and *app_periph_init.h* file. The safety peripheral settings are located in *safety_periph_init.c* and *safety_periph_init.h*.

3.2.1 PWM generation - FTM0 (M1_PWM_PERIPH)

- FTM is clocked from 75 MHz System clock source
- Only 6six channels are used, the other two are masked in OUTMASK register.
- FTM counter is running in BDM mode.
- Channels 0+1, 2+3, 4+5 are combined in pairs running in complementary mode
- Fault mode is enabled at each combined pair with manual fault clearing.
- PWM frequency is defined in macro `M1_PWM_FREQ`, default frequency is 10 kHz.
- Dead-time insertion is enabled at each combined pair. Value of deadtime is defined in macro `M1_PWM_DEADTIME`.
- FTM generates trigger to PDB on counter initialization.
- FTM fault zero is connected to the M1_CMP_OT_PERIPH (over-temperature) and fault one is connected to M1_CMP_OC_PERIPH (over-current). Both FTM fault signals are active high.

3.2.2 Analog sensing – ADC1 (M1_ADC_PERIPH) and ADC0 (FS_ADC_PERIPH)

- ADCs operate as 12-bit, single-ended converters.
- ALTCLK clock source is used (by default set to ~18.67 MHz).
- ADCs are using HW triggers.
- DMA is enabled for M1_ADC_PERIPH.
- band gap voltage regulator is enabled.

3.2.3 PWM and ADC synchronization (M1_PDB_PERIPH)

- PDB is used ADC triggering and synchronization between M1_PWM_PERIPH and M1_ADC_PERIPH & FS_ADC_PERIPH.
- PDB is triggered from FTM0.

- There is a PDB Sequence Error interrupt enabled.

3.2.4 Over-current and over-temperature check – CMP1 (M1_CMP_OC_PERIPH) and CMP0 (M1_CMP_OT_PERIPH)

- DAC output value is set according to desired over-current/over-temperature threshold level.
- Reference voltage is VDD.
- Positive input to the CMP is taken from analog pin (M1_CMP_OT_PERIPH_IN_POS and M1_CMP_OC_PERIPH_IN_POS).
- Negative input is taken from internal 6-bit DAC reference (M1_CMP_OT_PERIPH_IN_NEG and M1_CMP_OC_PERIPH_IN_NEG).
- CMP filter is enabled, four consecutive samples must agree.

3.2.5 DMA for ADC results reading – DMA0 (M1_DMA_PERIPH)

- Error interrupts enabled.
- Enabled TRGCOCO triggering for M1_DMA_CHN_RSLT (result register transfer from M1_ADC_PERIPH).
- Enabled TRGDMAAB triggering for M1_DMA_CHN_ACHN (channel number transfer to M1_ADC_PERIPH).
- Enabled TRGDMA1 triggering for M1_DMA_CHN_DLY (PDB delay transfer to M1_PDB_PERIPH).
- Enabled TRGDMADBG for M1_DMA_CHN_DBG channel (M1_DMA_DEBUG_MASK mask transfer to M1_DMA_DEBUG_GPIO.PTOR upon completion of M1_DMA_CHN_DLY transfer).

3.2.6 Slow-loop interrupt generation – FTM2 (M1_TMR_PERIPH)

- FTM is clocked from 75 MHz System clock.
- FTM counter is running in BDM mode.
- Initialize modulo is set to frequency 1 kHz.
- FTM interrupt is enabled.

3.2.7 External control signal measurement – FTM1 (APP_EXTCMD_PERIPH)

- FTM is clocked from 75 MHz System clock.
- FTM counter is running in BDM mode.
- Initialized modulo is set to maximal value.
- Dual-edge, one-shot capture mode is set.
- The *pwm_in_mcu* signal ON-time measurement is selected.

3.3 Peripheral connection

The analog measurement, PWM generation, and fast-loop FL timing peripheral connection block diagram is shown in Figure 3 below. Following peripherals are used:

- M1_PWM_PERIPH – The three-phase PWM generator periphery. Responsible for TRGP synchronization trigger generation for M1_PDB_PERIPH upon the PWM timer reload.
- M1_PDB_PERIPH – Two-channel PDB delay timer and ADC conversion trigger TRGAM1 and TRGAFS generator. The M1_PDB_CHANNEL channel delay register is updated by M1_DMA_PERIPH and the TRGAM1 is generated N_{smpl}-times per T_{PWM} period. The FS_PDB_CHANNEL generates single TRGAFS trigger per T_{PWM} is updated by software in FL during $T_{\mathrm{TST\_UI\_MAX}}$ period.

- M1_ADC_PERIPH – The analog converter used for motor-control quantity sample acquisition. Triggered N_{smpl}-times per T_{PWM} period by M1_PDB_CHANNEL of M1_PDB_PERIPH. Each conversion completion generates TRGCOCO trigger, which start M1_DMA_CHN_RSLT transfer.
- FS_ADC_PERIPH – The analog converter used for safety comparison check FS.CMP. Triggered *once* per T_{PWM} period by FS_PDB_CHANNEL of M1_PDB_PERIPH. The conversion result is recovered at the start of FL during $T_{TST\_UI\_MAX}$ period.
- Three M1_DMA_PERIPH channels:
  - M1_DMA_CHN_RSLT – Transfer of ADC measurement result from RA register of M1_ADC_PERIPH register to the ADC result table M1_DMA_TAB_RSLT in non-safety part of RAM. Each transfer is triggered by TRGCOCO trigger and generates TRGDMA0 trigger upon completion.
  - M1_DMA_CHN_ACHN – Transfer of the next ADC channel value from M1_DMA_TAB_ACHN table in Flash to SC1A register of M1_ADC_PERIPH converter. Each transfer is triggered by TRGDMA0 trigger and generates TRGDMA1 trigger upon completion.
  - M1_DMA_CHN_DLY – Transfer of the next M1_PDB_CHANNEL delay register value and M1_PDB_CS_LDOK_MASK mask from PDB delay table M1_DMA_TAB_DLY to the DLY0 and SC registers of M1_PDB_PERIPH. Each transfer is triggered by TRGDMA1 trigger. Completion of major loop (transfer of the last sample in T_{PWM}) starts the fast loop interrupt routine (FL).

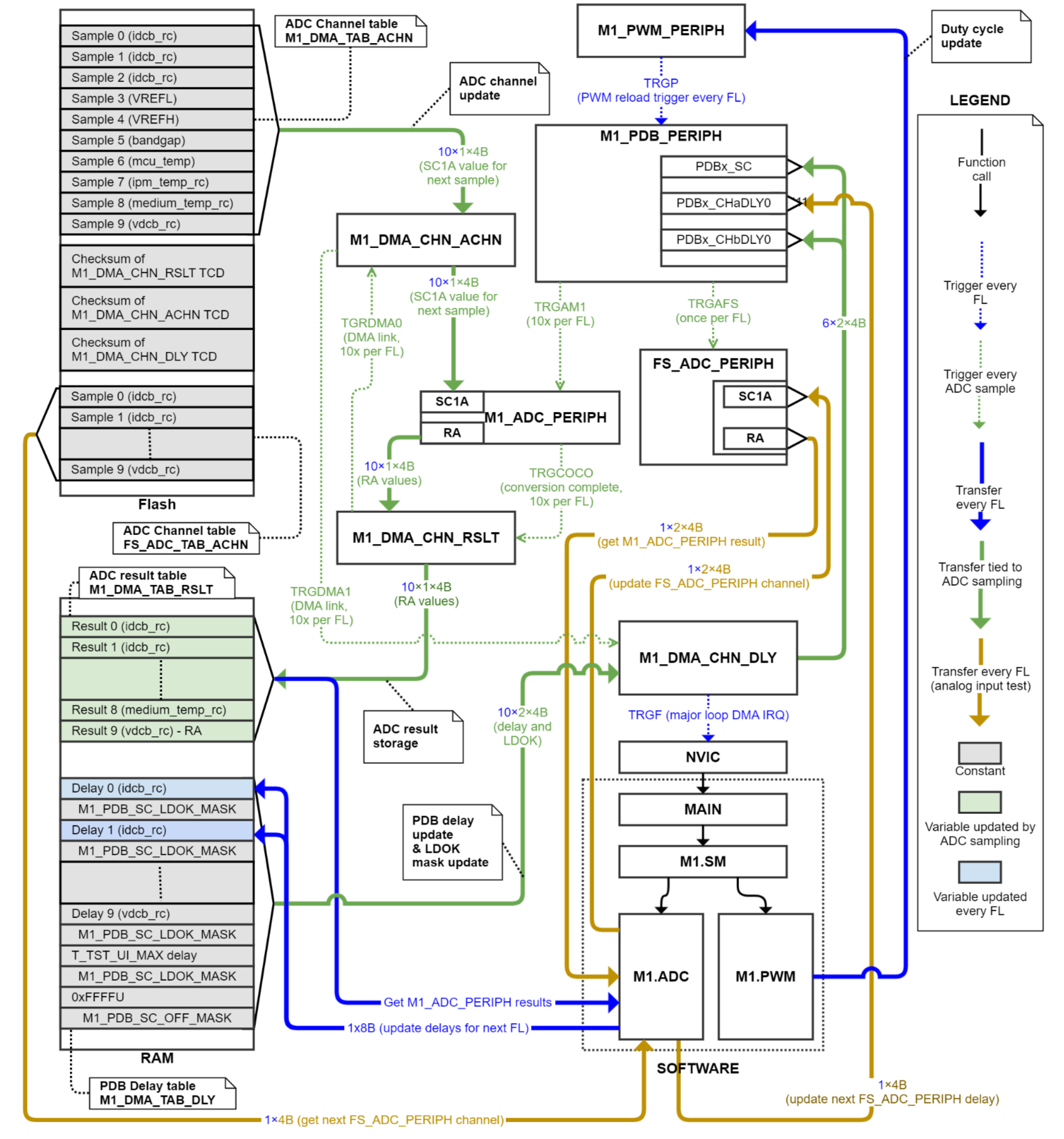


Figure 3. ADC measurement, PWM generation, and FL triggering chain

The slow-control loop (SL) is triggered by M1_TMR_PERIPH timer periphery (see Figure 4 below).

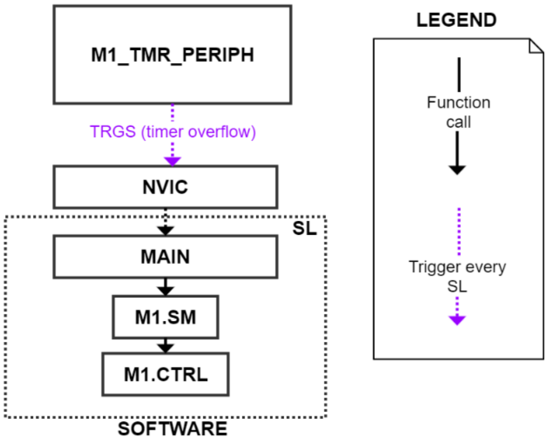


Figure 4. Slow-loop triggering chain

All implemented triggers are listed in Table 3 below.

Table 3. : Trigger signal description

Signal name	Description	Occurrence
TRGRST	The MCU restart trigger.	At startup
TRGF	Fast control loop (FL) synchronization trigger.	Once per $T_{s.}$
TRGS	Slow control loop (SL) synchronization trigger.	Once per $T_{s\text{-}slow}$.
TRGOC	Fast hardware over-current fault trigger from M1_CMP_OC_PERIPH.	Raised by hardware over-current event.
TRGOT	Fast hardware over-temperature fault trigger from M1_CMP_OT_PERIPH.	Raised by hardware over-temperature event.
TRGP	Trigger generated by M1_PWM_PERIPH periphery for analog conversion sequence synchronization.	Once per $T_{s.}$
TRGCOCO	The M1_ADC_PERIPH conversion completion trigger.	N_{smpl}-times per $T_{s.}$
TRGAM1	Trigger generated by M1_PDB_PERIPH periphery for triggering of individual M1_ADC_PERIPH samples.	N_{smpl}-times per $T_{s.}$
TRGAFS	Trigger generated by M1_PDB_PERIPH periphery for triggering of FS_ADC_PERIPH sample.	Once per $T_{s.}$
TRGDMA0	Trigger generated by M1_DMA_CHN_RSLT channel.	N_{smpl}-times per $T_{s.}$
TRGDMA1	Trigger generated by M1_DMA_CHN_ACHN channel.	N_{smpl}-times per $T_{s.}$

The clock distribution diagram is shown in Figure 5 below. There are three independent clock sources:

- CLOCK_LPO_FREQ – The 1 kHz clock generated by PMC periphery, used by WDOG periphery only.
- CLOCK_MCGIRCLK_FREQ – The 1 MHz clock generated by FAST_IRCLK in MCG periphery. Serves as independent clock source for LPTMR timer periphery to perform the FS.WDOG reset-capability and the FS.CLK clock tests.

- CLOCK_MCGOUTCLK_FREQ – The 74.7 MHz frequency used by all other MCU peripheries, memories, and CPU. Generated by Frequency Locked-Loop (FLL) using the SLOW_IRCLK source in MCG periphery. The clock is further divided in SIM periphery as follows:
  - — CLOCK_SYSTEM_FREQ – The undivided 74.7 MHz clock for core, M1_PWM_PERIPH, M1_DMA_PERIPH, M1_TMR_PERIPH, PORT, GPIO, and APP_EXTCMD_PERIPH peripheries.
  - — CLOCK_BUS_FEQ – The 24.9 MHz clock for Flash, M1_PDB_PERIPH, M1_CMP_OC_PERIPH, and M1_CMP_OV_PERIPH peripheries.
  - — CLOCK_ALT_ADC_FREQ – The 18.7 MHz clock for M1_ADC_PERIPH and FS_ADC_PERIPH analog converters.

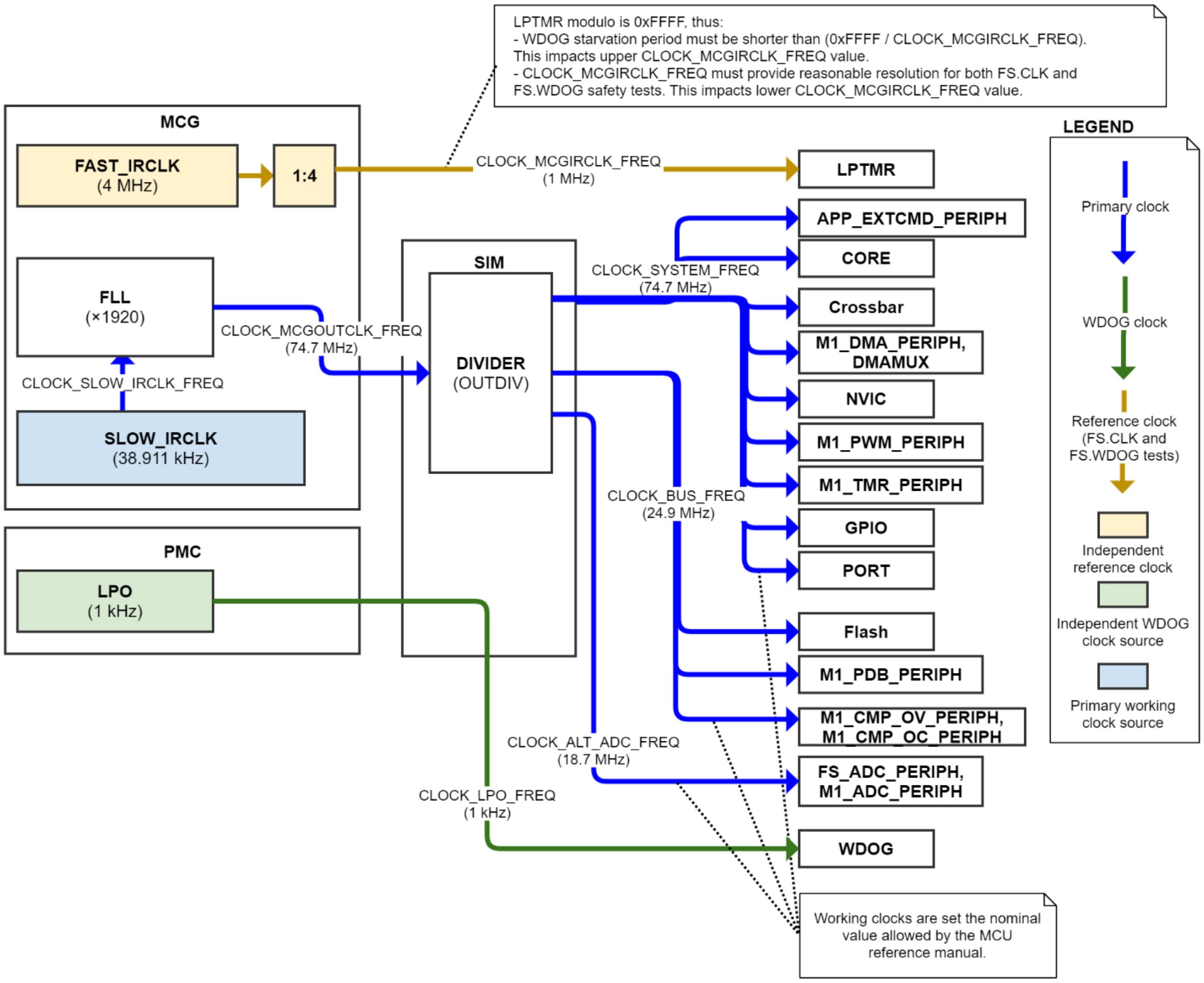


Figure 5. Clock distribution diagram

3.4 Hardware timing and synchronization

There are generally four timing sections in which the application is executed:

- *After-reset (AR)* – The initialization phase executed after the MCU reset. Started by the MCU restart trigger (TRGRST).
- *Background (BR)* – The lowest-priority execution cycle with non-fixed execution period.
- *Slow-loop (SL)* – The slow control loop with priority higher that BR. Execution started by slow control loop synchronization trigger (TRGS) every $T_{s\text{-}slow}$ = 1 ms, after the AR phase is completed.

- *Fast-loop (FL)* - The fast control loop with the highest priority (it is assumed to be uninterruptible). Execution started by fast control loop synchronization trigger (TRGF) every T_s = 100 μs, after the AR phase is completed.

Additionally, the following time periods are defined:

- T_{PWM} – Period of the generated PWM signals *pwm_at*, *pwm_bt*, *pwm_ct*, *pwm_ab*, *pwm_bb*, and *pwm_cb*. By default set to 100 μs.
- T_s – Period at which the fast-loop is executed. By default equal to the T_{PWM}.
- $T_{s\text{-}slow}$ – Period at which the slow-loop is executed. By default equal to 1 ms.
- T_{DLY1ST} – Minimal delay between M1_PWM_PERIPH reload (TRGP trigger event) and the first allowed M1_ADC_PERIPH conversion (end of $T_{TST\_UI\_MAX}$ period).
- $T_{DLYLAST}$ – Minimal delay between the last M1_ADC_PERIPH conversion in T_{PWM} period and M1_PWM_PERIPH reload (TRGP trigger event).
- T_{DLYNXT} – Minimal delay between two M1_ADC_PERIPH samples given by conversion time and transfer time of M1_DMA_CHN_RSLT, M1_DMA_CHN_ACHN, and M1_DMA_CHN_DLY channels.
- $T_{TST\_UI\_MAX}$ – Period during which all uninterruptible safety tests are sequentially executed (no sample can be taken by M1_ADC_PERIPH and no other interrupt can occur during this time). The completion of all necessary tasks is confirmed by software by correctly configuring M1_DMA_CHN_DLY channel, otherwise the sample is taken by M1_ADC_PERIPH at the end of $T_{TST\_UI\_MAX}$, resulting in M1_PDB_SC_OFF_MASK being applied (M1_PDB_PERIPH counter is stopped).
- T_{DT} – Dead-time defined as minimum time between falling and raising edge of complementary PWM signal. The delay is inserted by M1_PWM_PERIPHERY periphery to prevent DC-bus shoot through and the MOSFET heating.

The application timing diagram in Figure 6 shows the synchronization of the M1_PWM_PERIPH → M1_PDB_PERIPH → M1_ADC_PERIPH → M1_DMA_PERIPH and M1_PWM_PERIPH → M1_PDB_PERIPH → FS_ADC_PERIPH peripherals and execution of background, fast-loop, and slow loop.

To help with acquisition of large number of variably placed ADC samples, scalable DMA-enhanced M1.ADC driver is implemented. The benefits are a minimal CPU assistance, a large number of quantities in exact times can be acquired every T_{PWM} period and the second converter FS_ADC_PERIPH is free for safety compare checking FS.CMP. Total of N_{smpl} = 10 quantities are sampled by default:

1. The first *idcb_rc* current sample (sample position changes).
2. The second *idcb_rc* current sample (sample position changes).
3. The *idcb_rc* current offset measurement (during V_{111} voltage vector).
4. Voltage reference *VREFL*.
5. Voltage reference *VREFH*.
6. Internal band gap voltage reference.
7. Inverter temperature *ipm_temp_rc.*
8. Medium temperature *medium_temp_rc.*
9. MCU temperature *mcu_temp.*
10. DC-bus voltage *vdcb_rc*.

The M1_ADC_PERIPH measurement occurs as follows:

1. The M1_PDB_PERIPH counter is reloaded with every M1_PWM_PERIPH counter reload (TRGP). This ensures PWM-ADC synchronization.
2. When M1_PDB_PERIPH counter reaches the DLY0 value, M1_ADC_PERIPH measurement trigger is generated (TRGAM1).
3. Once the conversion completes, the M1_DMA_CHN_RSLT channel is triggered (TRGCOCO). The measurement result is transferred from RA register to the ADC result table M1_DMA_TAB_RSLT.

4. The M1_DMA_CHN_ACHN channel is triggered next (TRGDMA0). The channel number of the next analog sample is transferred to SC1A register of M1_ADC_PERIPH.
5. The M1_DMA_CHN_DLY channel is triggered next (TRGDMA1). The DLY0 is updated for next sample (to be ahead of the PDB counter current value).
6. The fast loop (FL) ISR is triggered on M1_DMA_PERIPH major loop complete trigger. The M1_PWM_PERIPH registers are loaded via LDOK by software at the start of fast-loop.
7. All the active-vector *idcb_rc* sample time updates and uninterruptible safety tests (FS.PC, FS.RAM, FS.WDOG refresh, FS.DMA, FS.CMP, and part of FS.CORE) are conducted during $T_{TST\_UI\_MAX}$ time.
8. The M1_PDB_PERIPH is by default configured to trigger M1_ADC_PERIPH sample at the end of $T_{TST\_UI\_MAX}$ period. The software must configure the M1_PDB_PERIPH before this sample conversion is triggered, otherwise the M1_PDB_SC_OFF_MASK is applied to SC register of M1_PDB_PERIPH, resulting PDB counter stop. This feature is implemented to make sure that proper timing is adhered by the application and the M1_PDB_TAB_DLY table is updated before the first sample is taken.

The safety FS_ADC_PERIPH measurement channel and time is sequentially changed to provide parallel measurement of all motor-control M1_ADC_PERIPH quantities (so analog compare check FS.CMP can be done). The FS_ADC_PERIPH measurement occurs as follows:

1. The M1_PDB_PERIPH counter is reloaded with every FTM counter reload (TRGP). This ensures PWM-ADC synchronization.
2. When M1_PDB_PERIPH counter reaches the DLY0 value, FS_ADC_PERIPH measurement trigger is generated (TRGAFS).
3. The latest sample is recovered and the next is configured during $T_{TST\_UI\_MAX}$ period at the start of fast-loop ISR (FL).

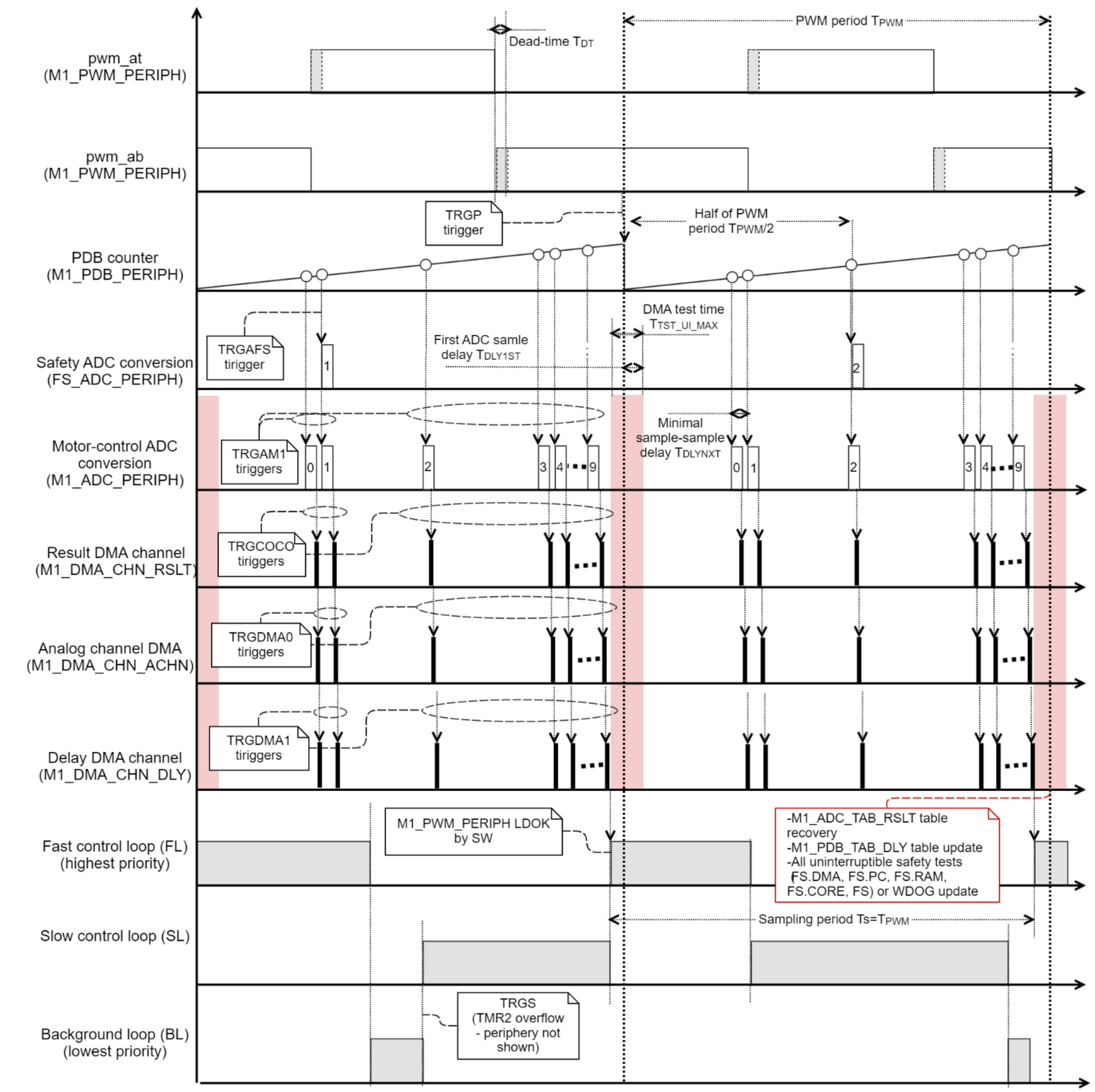


Figure 6. Application timing diagram

While it is desirable to use T_{PWM}~100 μs due to lower audible noise, it might not always be necessary to operate with such short fast loop period T_s (depending on motor electrical time constant) and therefore greatly lower the CPU load. The M1.ADC driver allows to use double rate $T_s = 2 \cdot T_{PWM}$. This is achieved by stopping the M1_PDB_PERIPH counter before the first *idcb_rc* current sample is taken and waiting for the next TRGP trigger. Principle of This feature is shown in figure below.

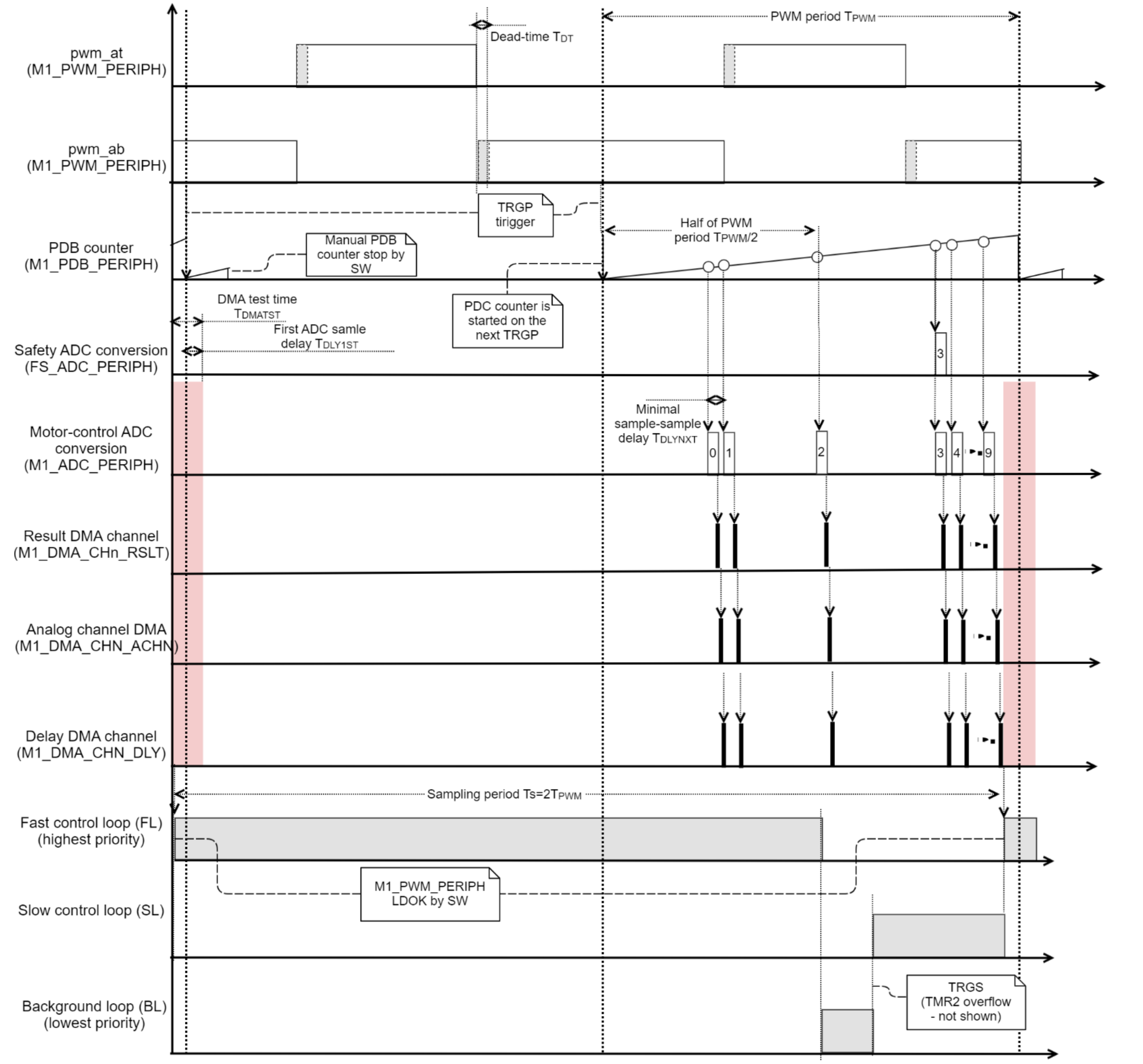


Figure 7. Application timing for double PWM vs FL rate

Chapter 4
Software description

This chapter describes architecture of the *mc_pmsm_safe* software. Additional documents like the requirement specification and software design document are normally NXP-internal only. The high-level block diagram of the PMSM pump software is shown in Figure 8 below. At high level, the software is organized in following blocks:

- *Startup* – Safe MCU memory initialization.
- *main* – The safe module with all ISRs, calling MCU after-reset (AR), background (BL), fast-loop (FL), slow-loop (SL) routine.
- *Application* – The non-safety user application tasks. One of goals is FOC setpoint and state commanding.
- *FOC* – The non-safety Field-Oriented Control routines (speed and current control loops, setpoint command).
- *MC state-machine* – The safe sensorless motor control part, including state-machine, estimators, and diagnostic routines.
- ADC+DMA – The safe DMA-based ADC driver designed for single-shunt current reconstruction.
- PWM – The safe three-phase shifted-PWM driver.
- FS – The functional safety routines module, containing a number of MCU core, memory, and peripheral self-tests.
- RTCESL 4.7 – Real Time Control Embedded Software Motor Control and Power Conversion Libraries (www.nxp.com/rtcesl).
- IEC60730B library 4.1 – Certified IEC60730 class B safety libraries (www.nxp.com/iec60730).
- FreeMASTER 3.0 – The FreeMASTER debugging interface (www.nxp.com/freemaster).

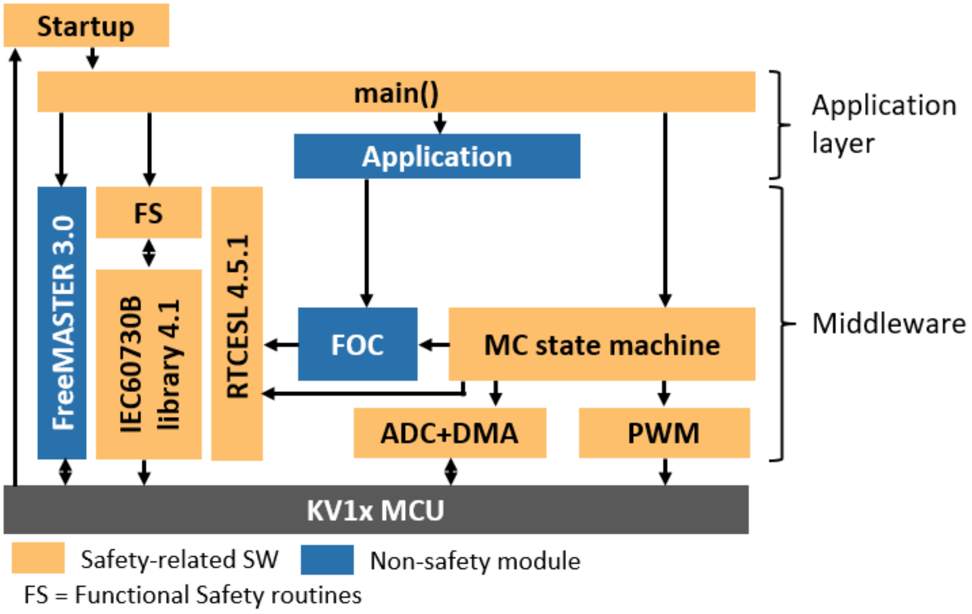


Figure 8. High-level system block diagram

4.1 Safety tests implemented using IEC60730 Class B library

Safety tests are implemented using IEC60730B libraries. Individual tests can be configurated and optional switched on or off in *safety_cfg.h* header file. The necessary macros for the safety example are defined in this file. The "switch macros", which enable the user to turn off the calling of the safety test, are defined in the beginning. More information can be found in document Kinetis CM0+ Safety Example or IEC60730_B_CM0_Library_UG_v4_1.

The safety mechanisms were selected and implemented on the basis of Failure Mode and Effect Analysis (FMEA), which is available in Appendix A.

4.1.1 Program counter register test (FS.PC)

The goal is test program counter register for stuck-at. The test is executed after-reset and during runtime (uninterruptible test period $T_{TST\_UI\_MAX}$).

The PC register test is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). The program counter cannot be tested using a simple pattern like other core register. The test instead executes small routines in two different memory addresses (addresses should be different in as many bits as possible to check for stuck-at). The PC test routine located in Flash at 0x410 is executed first. During the execution a small part for code is copied to 0x1FFFFBEE address (close invert value of 0x410) and executed as well. If the routines correctly set the PC Test Flag memory, the test passed. The RAM memory content at 0x1FFFFBEE is restored after the test.

NOTE

The program counter test cannot be interrupted.

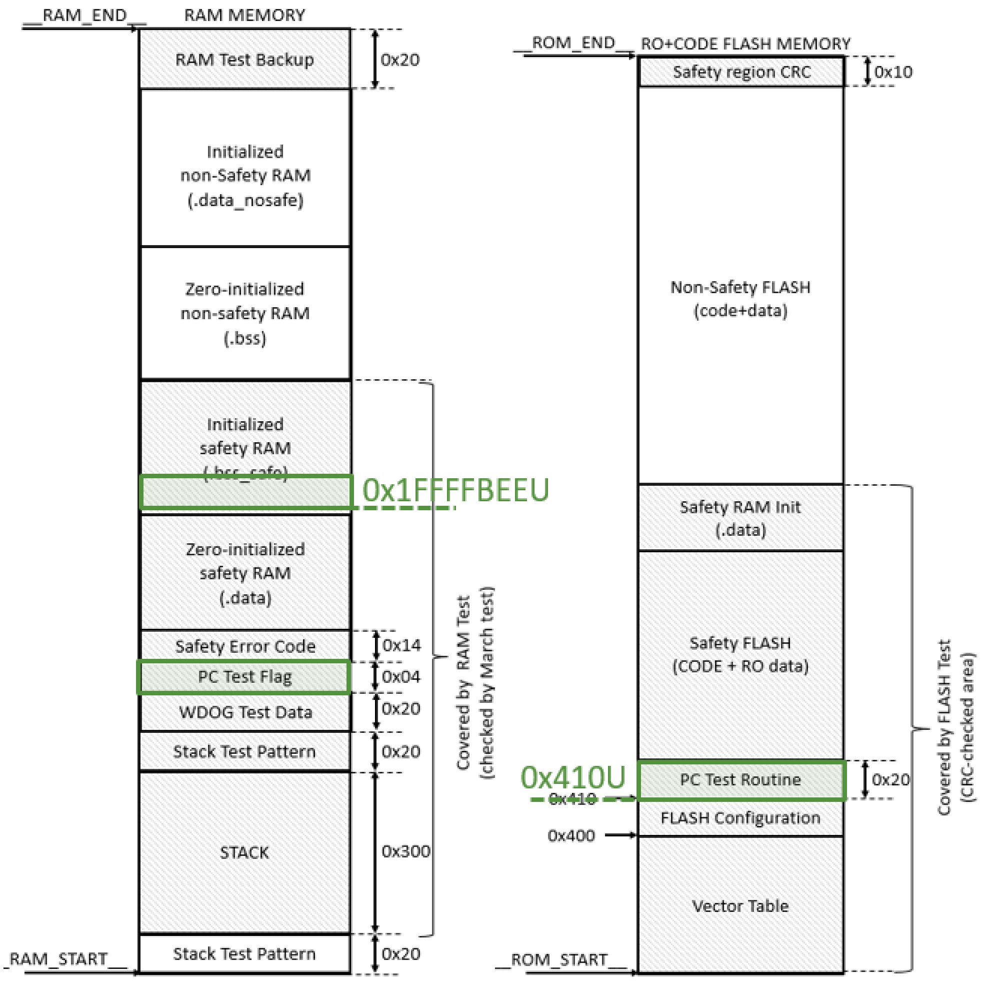


Figure 9. Program Counter test routine locations

4.1.2 Core register test (FS.CORE)

The goal is to test the CM0+ core registers for the stuck-at condition. The test is executed after-reset AR and during runtime (fast-loop FL and background BG).

The core registry is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). A pattern test is used to check for stuck-at faults. The test is split as follows:

- *Interruptible* - Register tested after-reset and at background: R0-R7, R12, LR, APSR, PSP, R8-R11, and CONTROL.
- *Uninterruptible* - Registers tested after-reset and in the fast-loop as uninterruptible test: PRIMASK and MSP.

4.1.3 Watchdog test (FS.WDOG)

The goal is check for program stall and to test the ability of WDOG to reset MCU in specified time. The test is executed after-reset (reset capability and starvation) and during runtime (starvation).

The WDOG test is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). The FS.WDOG test covers:

- *Reset capability test* - The after-reset test of WDOG ability to cause MCU reset in time. Two clocks are compared by this test:
  - — The CLOCK_MCGIRCLK_FREQ (sourced by FAST_IRCLK) is acting as an independent timer for LPTMR periphery.
  - — The CLOCK_LPO_FREQ (sourced by LPO 1kHz oscillator) acts as WDOG source clock.

The reset source is determined at the start of the test. If other-than-WDOG source is detected, the LPTMR counter is restarted and WDOG starvation is awaited in endless loop. If the restart was caused by WDOG, the last value of LPTMR counter (before WDOG-caused reset) is checked for valid limits.

- *Program stall check* - The WDOG must be periodically fed, otherwise its starvation causes MCU reset. The starvation period is by default set to 30ms. During runtime the WDOG update is conducted as one of uninterruptible test routines at the beginning of the fast control loop ($T_{TST\_UI\_MAX}$).

NOTE

Some debuggers do not allow the WDOG reset. Due to this, it is necessary to turn off the WDOG when debugging the application.

4.1.4 Interrupt handling test (FS.ISR)

The goal is check for correct interrupt execution rate and safely handle unexpected IRQs.

The test is executed during runtime (within fast-, slow- and background-loop).

Following system of counters is implemented to check for correct interrupt execution rate:

- Fast-loop counter – Incremented every fast-loop FL. Checked for limit violation and then cleared every slow-loop ISR. If the fast-loop vs slow-loop execution rate is incorrect a safety fault is triggered.
- Slow-loop counter – Incremented every slow-loop ISR. Checked for limit violation and then cleared every background execution. Because the slow loop and background loop are not synchronized, only upper limit is checked (violation means too long background loop execution).

All interrupt vectors have assigned ISR. If the unexpected IRQ is generated, the safety error handler routine is invoked.

4.1.5 Analog compare test (FS.CMP)

The goal is checking the PORT→AMUX→M1_ADC_PERIPH→M1_DMA_PERIPH analog measurement chain. The test is executed during runtime (within fast-loop).

The test is implemented within M1.ADC driver. The M1_ADC_PERIPH samples of all measured quantities are store in ADC result table M1_DMA_TAB_RSLT table each fast loop. The FS_ADC_PERIPH is configured to measure one analog quantity at the same time as M1_ADC_PERIPH (see Figure 10). The FS_ADC_PERIPH quantity is periodically changed every SL, so all quantities are eventually scanned. Difference between M1_ADC_PERIPH and FS_ADC_PERIPH results is accumulated in FL, until evaluation in SL occurs. If the accumulated difference between M1_ADC_PERIPH and FS_ADC_PERIPH results crosses maximal threshold, the safety error is entered.

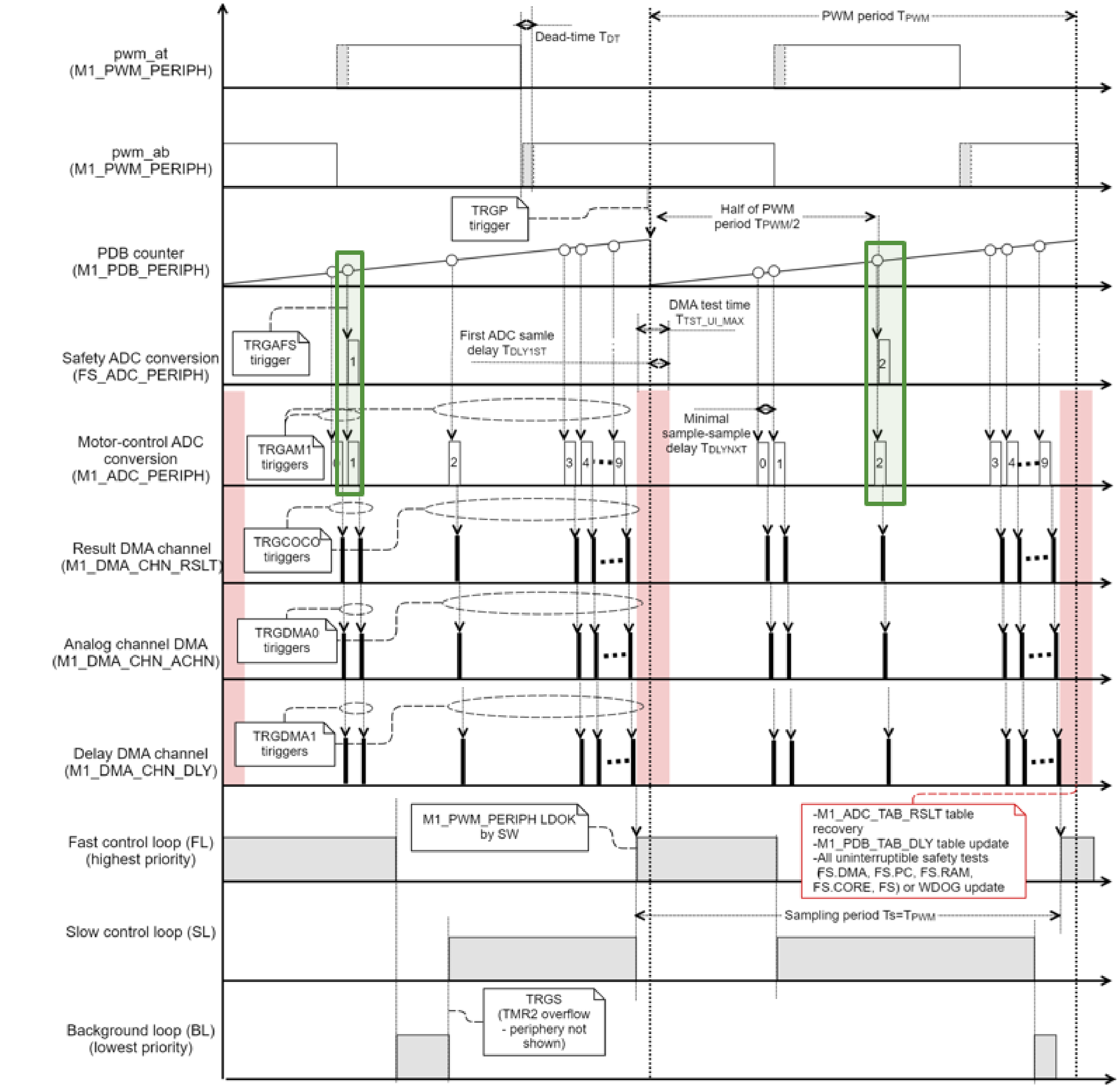


Figure 10. The analog compare check

4.1.6 Analog reference check (FS.REF)

The goal is check measured analog quantities for valid range.

The test is executed during runtime (within slow-loop SL).

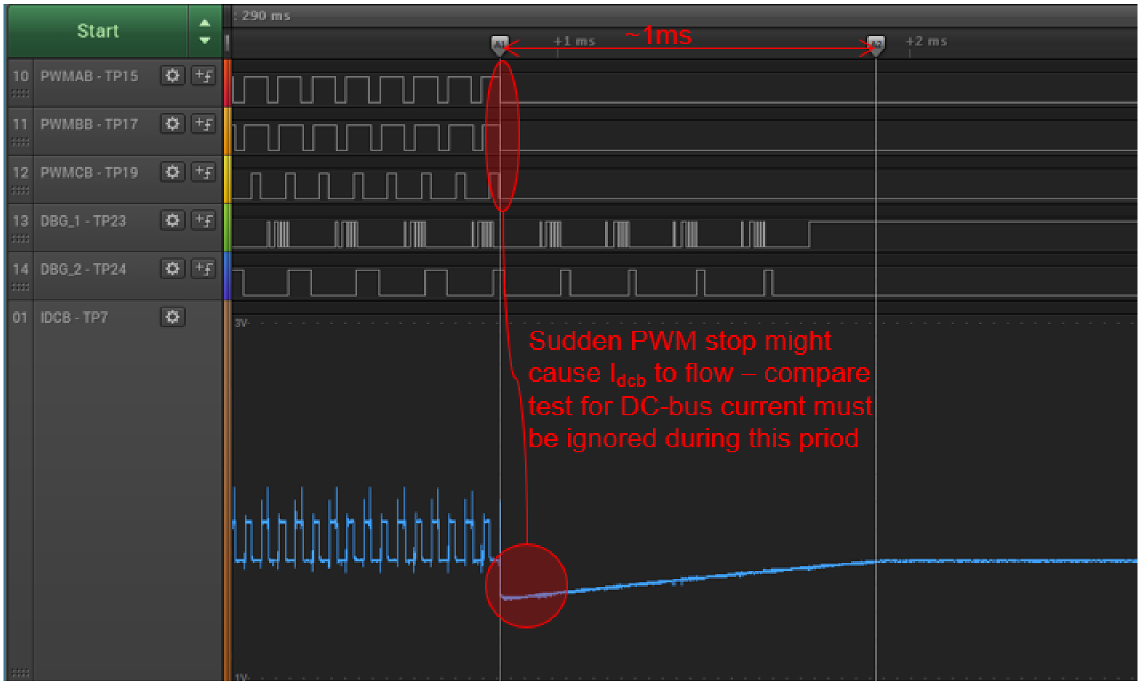
The quantities measured by M1.ADC driver, which are relevant for ADC reference test, are listed in Table 4. When any of these quantities are above expected maximum or below expected minimum, the safety fault is triggered.

Table 4. Quantities check by analog reference test

Quantity	Minimum	Maximum
idcb_rc current offset	1.60 V	1.70 V
VREFH	-	0.10 V
VREFL	3.20 V	-
medium_temp_rc	1.45 V	1.95 V
ipm_temp_rc	0.10 V	2.80 V
mcu_temp	0.50 V	0.80 V
band gap	0.95 V	1.05 V

NOTE

DC-bus current *idcb_rc* offset reference test can be violated upon sudden PWM stop, because the DC-bus current might continue to flow, depending on pre-stop conditions (see example in Figure 11). This phenomenon is considered to be safe, therefore, to prevent unnecessary fault trigger, the *idcb_rc* offset reference test violation is ignored roughly one millisecond after M1.PWM stop PWM output generation.



Sudden PWM stop might cause I_{dcb} to flow – compare test for DC-bus current must be ignored during this priod

Figure 11. Safe idcb_rc analog reference test violation

4.1.7 M1_DMA_PERIPH Safety Function (FS.DMA)

The goal is check for correct DMA behavior.

The test is executed during rutime as uninterruptible test (during $T_{TST\_UI\_MAX}$).

A number of tests were implemented to check the M1_DMA_PERIPH operation (see Figure 12):

- *TCD memory stuck-at test* - Stuck-at test of all M1_DMA_PERIPH TCDs read-write memory. A simple pattern test is used.
- *TCD checksum test* - Checksum test of all DMA TCDs. The TCD checksums are calculated during build a compared to value calculated during runtime.
- *M1_DMA_TAB_DLY checksum test* - The checksum check for constant part of M1_DMA_TAB_DLY.
- *M1_DMA_TAB_RSLT under-/over-flow test* - The M1_DMA_TAB_RSLT result table over- and under-flow check. Patterns around the result table are checked for change.

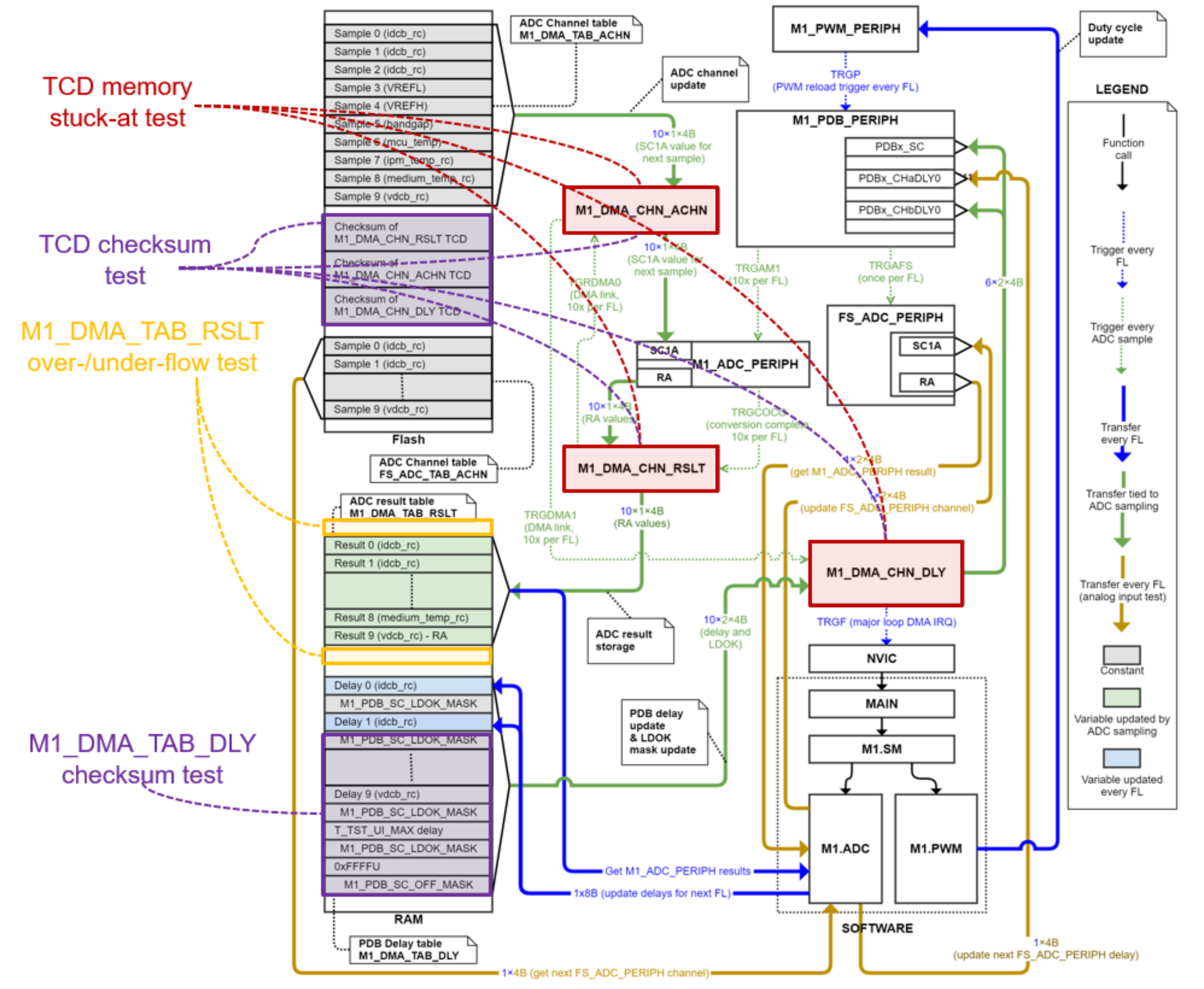

TCD memory stuck-at test
TCD checksum test
M1_DMA_TAB_RSLT over-/under-flow test
M1_DMA_TAB_DLY checksum test
Sample 0 (idcb_rc)
Sample 1 (idcb_rc)
Sample 2 (idcb_rc)
Sample 3 (VREFL)
Sample 4 (VREFH)
Sample 5 (bandgap)
Sample 6 (mcu_temp)
Sample 7 (ipm_temp_rc)
Sample 8 (medium_temp_rc)
Sample 9 (vdcb_rc)
Checksum of M1_DMA_CHN_RSLT TCD
Checksum of M1_DMA_CHN_ACHN TCD
Checksum of M1_DMA_CHN_DLY TCD
Sample 0 (idcb_rc)
Sample 1 (idcb_rc)
Sample 9 (vdcb_rc)
Flash
ADC Channel table FS_ADC_TAB_ACHN
ADC result table M1_DMA_TAB_RSLT
Result 0 (idcb_rc)
Result 1 (idcb_rc)
Result 8 (medium_temp_rc)
Result 9 (vdcb_rc) - RA
Delay 0 (idcb_rc)
M1_PDB_SC_LDOK_MASK
Delay 1 (idcb_rc)
M1_PDB_SC_LDOK_MASK
Delay 9 (vdcb_rc)
M1_PDB_SC_LDOK_MASK
T_TST_UI_MAX delay
M1_PDB_SC_LDOK_MASK
0xFFFFU
M1_PDB_SC_OFF_MASK
RAM
PDB Delay table M1_DMA_TAB_DLY
ADC Channel table M1_DMA_TAB_ACHN
ADC channel update
M1_PWM_PERIPH
TRGP (PWM reload trigger every FL)
M1_PDB_PERIPH
PDBx_SC
PDBx_CHaDLY0
PDBx_CHbDLY0
10×1×4B (SC1A value for next sample)
M1_DMA_CHN_ACHN
10×1×4B (SC1A value for next sample)
TGRDMA0 (DMA link, 10x per FL)
TRGAM1 (10x per FL)
TRGAFS (once per FL)
6×2×4B
FS_ADC_PERIPH
SC1A
RA
SC1A
RA
M1_ADC_PERIPH
10×1×4B (RA values)
TRGCOCO (conversion complete 10x per FL)
M1_DMA_CHN_RSLT
1×2×4B (get M1_ADC_PERIPH result)
1×2×4B (update FS_ADC_PERIPH channel)
TRGDMA1 (DMA link, 10x per FL)
10×1×4B (RA values)
M1_DMA_CHN_DLY
ADC result storage
10×2×4B (delay and LDOK)
TRGF (major loop DMA IRQ)
NVIC
MAIN
M1.SM
PDB delay update & LDOK mask update
M1.ADC
M1.PWM
Get M1_ADC_PERIPH results
1x8B (update delays for next FL)
SOFTWARE
1×4B (update next FS_ADC_PERIPH delay)
1×4B (get next FS_ADC_PERIPH channel)
Duty cycle update
LEGEND
Function call
Trigger every FL
Trigger every ADC sample
Transfer every FL
Transfer tied to ADC sampling
Transfer every FL (analog input test)
Constant
Variable updated by ADC sampling
Variable updated every FL

Figure 12. Implemented FS.DMA tests

4.1.8 Program flow check (FS.FLOW)

The goal is check for correct and complete order of application execution. The test is executed after-reset and in all safety relevant runtime loops.

The Control Flow Checking by Software Signatures (CFCSS) is utilized (see Figure 13 for principle). Following loops have their signature variables:

- *After-reset sequence*. Signatures are ordered as follows:

1. FS.INIT – WDOG initialization.
2. FS.START – MCU memory startup.
3. FS.INIT – Clock initialization.
4. FS.INIT – The PORT (MCU pin) initialization.
5. FS.INIT – The LPTMR initialization.
6. FS.FCN – Common safety routine initialization.
7. FS.WDOG – Test of WDOG reset capability.
8. FS.FLASH – Test of complete safety-related Flash memory.
9. FS.RAM – Test of complete safety-related RAM memory.
10. FS.PC – Test of program counter.
11. FS.CPU – Test of core registers.
12. MC.INIT – Initialization of M1_DMA_PERIPH.
13. MC.INIT – Initialization of M1_ADC_PERIPH and FS_ADC_PERIPH.
14. MC.INIT – Initialization of M1_PDB_PERIPH.
15. MC.INIT – Initialization of M1_TMR_PERIPH.
16. MC.INIT – Initialization of M1_PWM_PERIPH.
17. M1.ADC – Initialization of M1.ADC driver and FS.REF, FS.DMA, and FS.CMP analog tests.
18. M1.SM – Initialization of M1.SM state-machine.

- *Fast-loop sequence.* Signatures are ordered as follows:

1. FS.FCN – One uninterruptible test was completed.
2. M1.ADC – Analog measurements (phase currents and DC-bus voltage) were updated.
3. M1.SM - The fast-loop M1.DIAG fault diagnostic routines were completed. The set of routines depends on M1.SM state.
4. M1.SM – The control action from M1.CTRL was obtained **or** M1.SM state transition occurred.
5. M1.PWM – The PWM output driver update.
6. FS.ISR – The interrupt handling test.

- *Slow-loop sequence.* Signatures are ordered as follows:
  1. FS.CLK – The clock test measurement.
  2. FS.ISR – The interrupt handling test.
  3. FS.CMP – The analog compare test.
  4. M1.ADC – Analog measurements (temperatures and voltage references) were updated.
  5. M1.SM – The slow-loop M1.DIAG fault diagnostic routines were completed. The set of routines depends on M1.SM state.
  6. M1.SM – The M1.CTRl slow-loop update (providing M1.SM state to M1.CTRL and obtaining *M1SM_RequestStart/ M1SM_RequestStop* request from M1.CTRL).
- *Background-loop sequence.* Signatures are ordered as follows:
  1. FS.CORE – Interruptible test of core registers.
  2. FS.FLASH – Safety-related Flash runtime test.
  3. FS.CLK – The clock test evaluation.
  4. FS.STACK – The stack over-/under-flow check.

5. FS.ISR – The interrupt handling test.

- *Uninterruptible tests.* Executed sequentially (one test per fast-loop FL). Signatures are ordered as follows:
    1. FS.CORE – Uninterruptible core registers test.
    2. FS.PC – Program counter test.
    3. FS.RAM – Safety-relevant RAM memory runtime test.
    4. FS.WDOG – Feeding of watchdog.
    5. FS.DMA – M1_DMA_PERIPH TCD memory stuck-at test.
    6. FS.DMA – Check of M1_DMA_PERIPH TCDs checksum.
    7. FS.DMA – The M1_DMA_TAB_DLY table checksum.
    8. FS.DMA – The M1_DMA_TAB_RSLT table over-/under-flow check.

If the final signature at the end of the loop does not match the expected value, the safety fault is triggered.

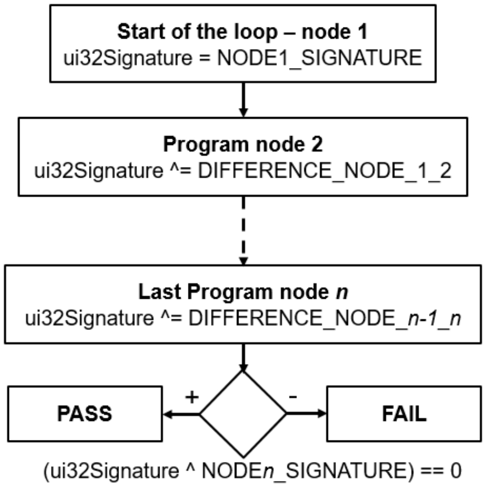

Start of the loop – node 1
ui32Signature = NODE1_SIGNATURE
Program node 2
ui32Signature ^= DIFFERENCE_NODE_1_2
Last Program node *n*
ui32Signature ^= DIFFERENCE_NODE_*n*-1_*n*
PASS
+
-
FAIL
(ui32Signature ^ NODE*n*_SIGNATURE) == 0

Figure 13. Control Flow Checking by Software Signatures (CFCSS) principle

4.1.9 External command check (M1.DIAG.EXTCMD)

The goal is checking the *pwm_in_mcu* PWM control signal to prevent unwanted M1.SM start. The test is executed in the fast-loop (edge counting) and slow-loop (edge count check).

A precise value of the control PWM duty cycle *D* is acquired via FTM periphery in double-capture mode. In order to prevent unwanted machine start a parallel control PORT-based PWM signal raising edge counter is implemented in the fast loop (see block diagram Figure 14). If the edge count rate does not correspond to allowed *pwm_in_mcu* frequency range (200 Hz < f_{ctrl} < 2 kHz by default), a FAULT state is entered by M1.SM.

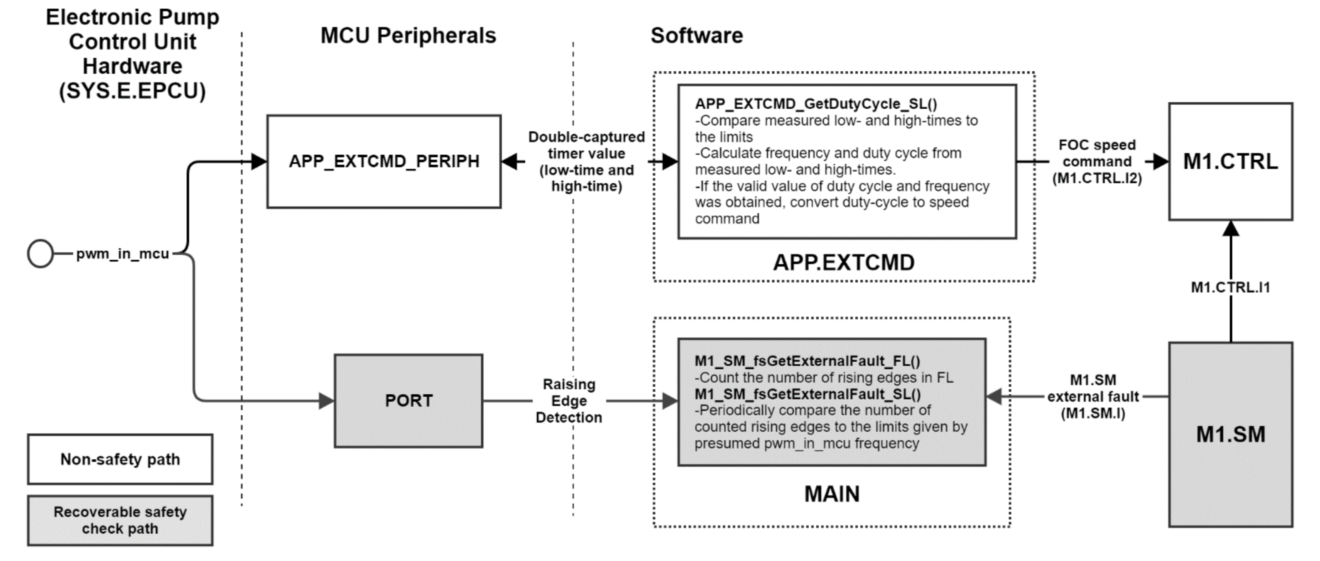

Electronic Pump Control Unit Hardware (SYS.E.EPCU)
MCU Peripherals
Software
APP_EXTCMD_PERIPH
Double-captured timer value (low-time and high-time)
APP_EXTCMD_GetDutyCycle_SL()
-Compare measured low- and high-times to the limits
-Calculate frequency and duty cycle from measured low- and high-times.
-If the valid value of duty cycle and frequency was obtained, convert duty-cycle to speed command
APP.EXTCMD
FOC speed command (M1.CTRL.I2)
M1.CTRL
M1.CTRL.I1
pwm_in_mcu
PORT
Raising Edge Detection
M1_SM_fsGetExternalFault_FL()
-Count the number of rising edges in FL
M1_SM_fsGetExternalFault_SL()
-Periodically compare the number of counted rising edges to the limits given by presumed pwm_in_mcu frequency
MAIN
M1.SM external fault (M1.SM.I)
M1.SM
Non-safety path
Recoverable safety check path

Figure 14. The M1.DIAG.EXTCMD safety mechanism block diagram

4.1.10 Safety flash test (FS.FLASH)

The goal is check safe part of Flash memory for content change.

The test is executed after-reset AR and during runtime in the background BG.

The Flash test is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). The CRC of safety-related Flash is calculated using CRC periphery and compared to CRC stored at the end of FLASH memory in *Safety region CRC* section (see Figure 15). The Flash test is executed:

- After-reset the entire safety-relevant Flash region is checked.
- During runtime the CRC for Flash is repeatedly calculated by 8x4B blocks.

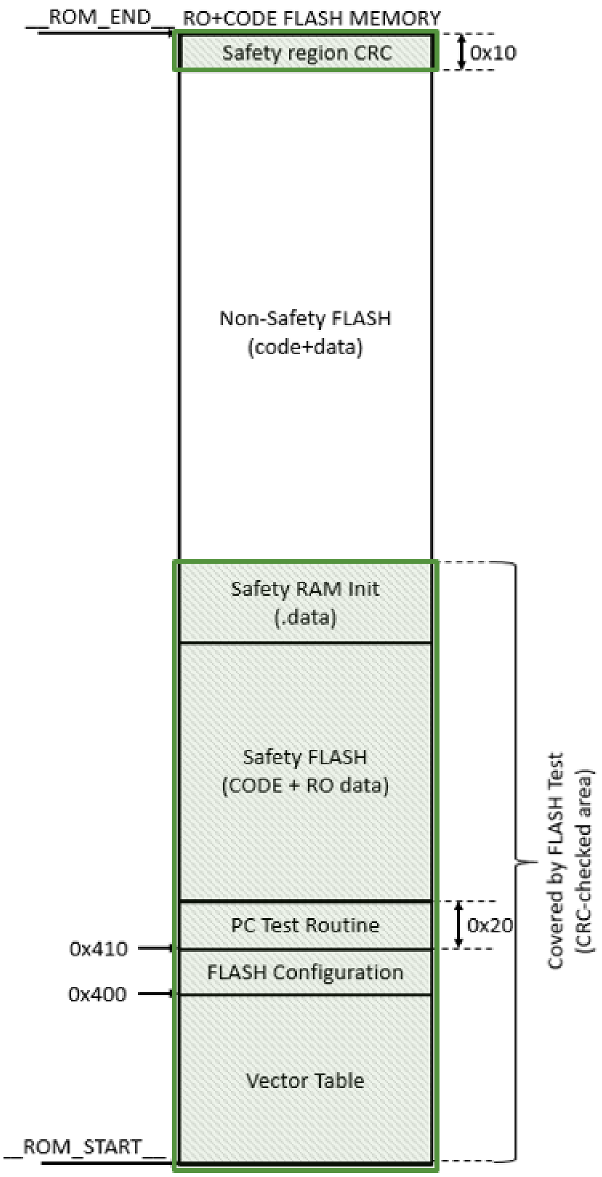


Figure 15. Tested Flash memory

The test can be turned off in the *safety_config.h* file.

The test consists of the following two parts:

- Post-build CRC calculation of the dedicated memory.
- Runtime CRC calculation and comparison with the post-build result.

The post-build calculation is different for each IDE:

In the IAR IDE, the CRC is calculated by the IDE directly using the linker (see Options->Build Action). The Flash test is fully integrated to the example project in the IAR IDE. It is necessary only to turn this test on in the *safety_config.h* file.

In the MCUXpresso IDE, the CRC is calculated by the *S-record* third-party tool. The user must do some additional steps. For more information, see post-build CRC in document Kinetis CM0+ Safety Example.

NOTE

When you debug your application with the Flash test turned on, be careful when using the breakpoint. The software breakpoint usually changes the CRC result and causes a safety error.

4.1.11 Safety RAM Test (FS.RAM)

The test checks the on-chip RAM for direct-coupling faults. The test is executed after-reset and during runtime in the fast-loop (uninterruptible test period $T_{TST\_UI\_MAX}$).

The march test is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). The test is destructive so the investigated RAM is copied to the *RAM Test Backup* section (see Figure 16) and restored once after the memory block test completes. The test is executed:

- After-reset test is conducted for entire safety-relevant RAM.
- The runtime test is executed for 3x4B memory blocks. However, the next block address is moved only by 1x4B. This way, the faults in neighboring RAM cell can be discovered.

NOTE

This test cannot be interrupted.

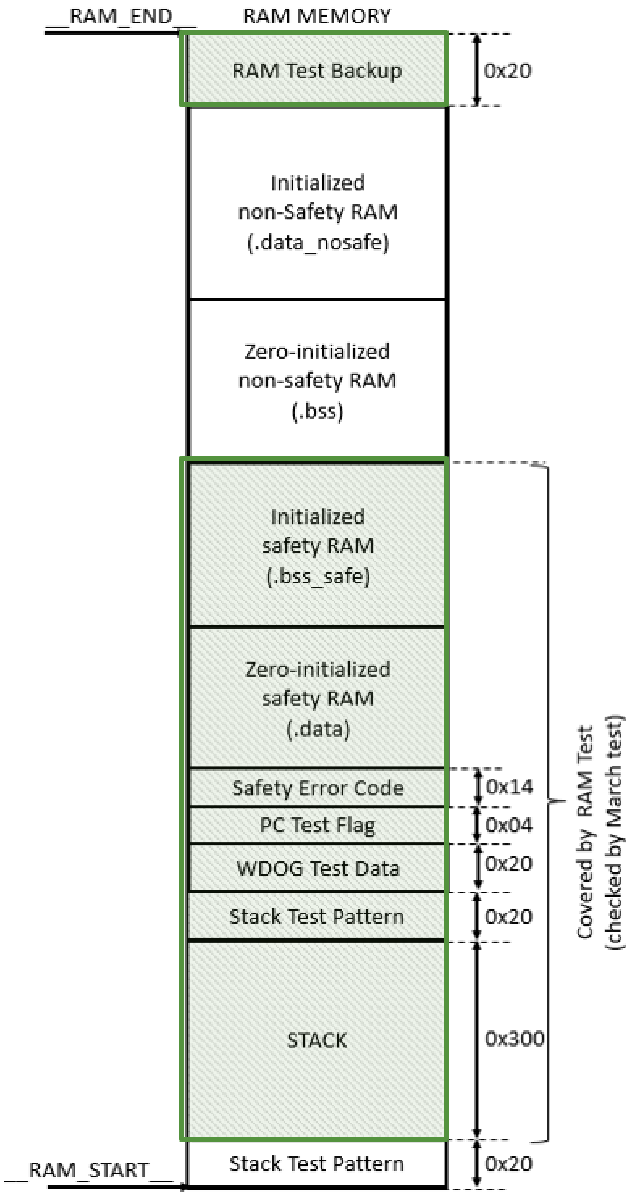


Figure 16. Tested RAM memory

4.1.12 Stack over-/under-flow test (FS.STACK)

The goal is check for the stack over- and under-flow. The test is executed during runtime in the background BG.

The march test is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). The stack RAM memory area is surrounded by specific memory patterns (see Figure 17). These patterns are checked for change during runtime.

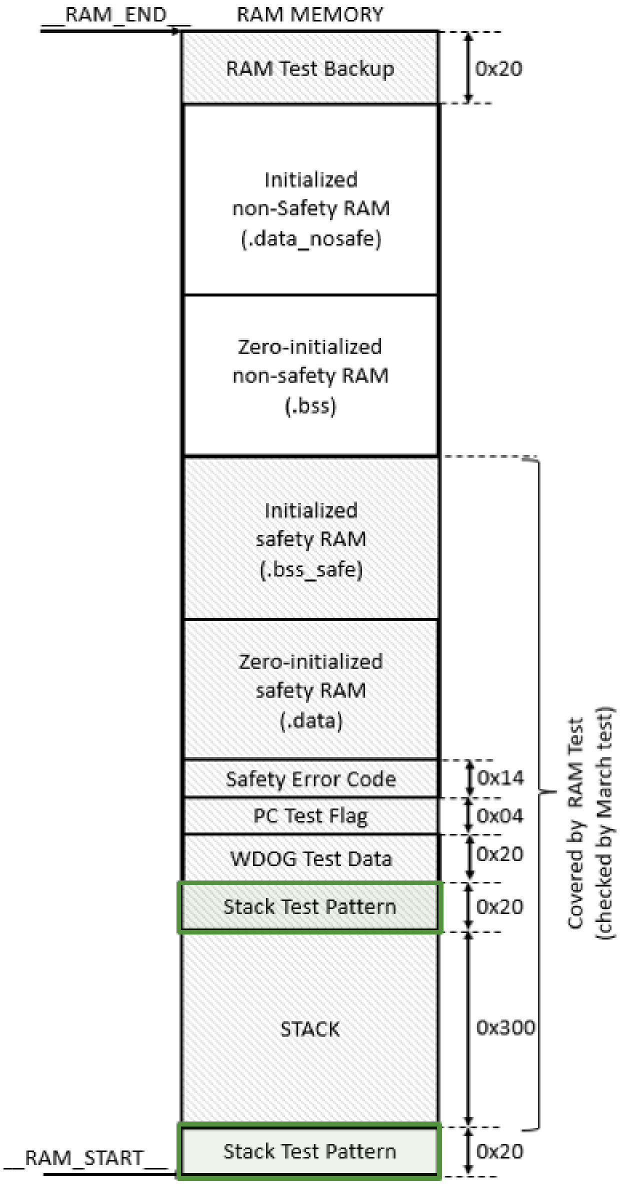


Figure 17. Stack test patterns

4.1.13 Clock test (FS.CLK)

The clock test procedure tests the oscillator frequency for the CPU core in the wrong frequency condition.

The goal is to test the CLOCK_MCGOUTCLK_FREQ frequency for drift.

The test is executed during runtime in slow-loop SL (measurement) and back-ground (evaluation).

The clock test is implemented using the certified IEC60730 class B safety library routines (see documentation at www.nxp.com/iec60730). Two clocks are compared by the clock test:

- The core and slow-loop (SL) timer is supplied by CLOCK_MCGOUTCLK_FREQ (sourced by SLOW_IRCLK oscillator).
- The LPTMR periphery is supplied by CLOCK_MCGIRCLK_FREQ (sourced by FAST_IRCLK oscillator).

The LPTMR counter value is stored and then restarted every SL (one millisecond by default). The stored counter value is then checked in the background for valid range.

4.2 Motor control faults

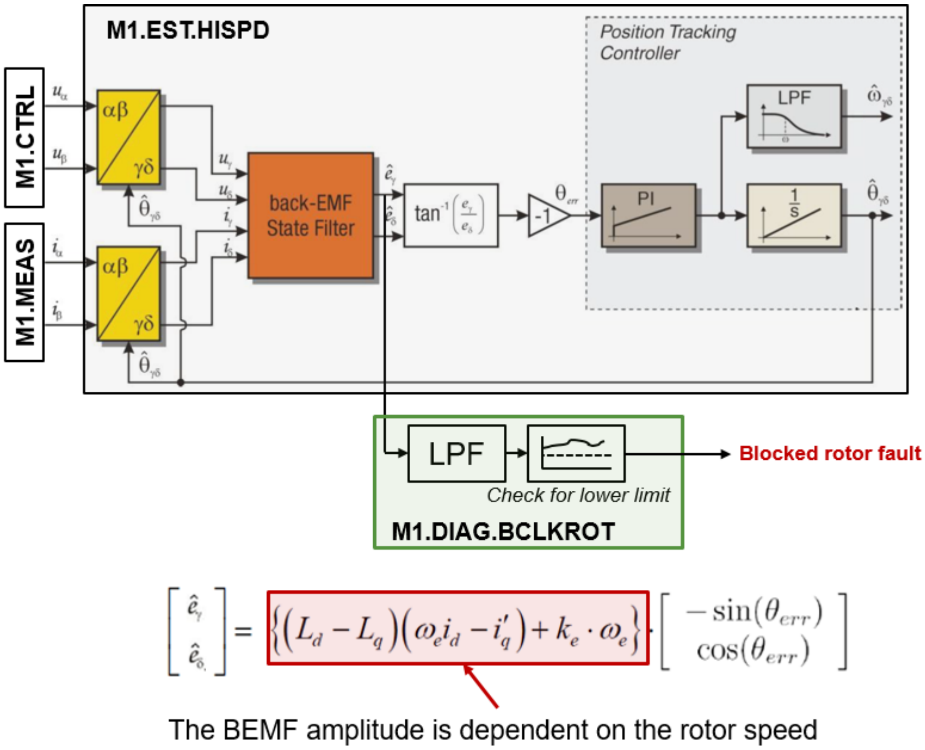
The motor control faults can be configured in *m1_pmsm_appconfig.h* file. The safety mechanisms were selected and implemented based on Failure Mode and Effect Analysis (FMEA), which is available in Appendix A.

4.2.1 Blocked rotor test (M1.DIAG.BCLKROT)

The goal is check for blocked-rotor condition.

The test is executed during HI_SPD state of M1.SM in the slow-loop SL.

The blocked rotor detection algorithm principle is shown in Figure 18 below. The blocked rotor condition is determined based on the BEMF observer (M1.EST.HISPD) estimated EMF voltage. If the rotor is spinning properly, then the estimated Q-axis BEMF voltage should be above minimal threshold value. Otherwise a fault condition is triggered.



$$\begin{bmatrix} \hat{e}_\gamma \\ \hat{e}_\delta \end{bmatrix} = \boxed{\left\{ \left(L_d - L_q\right)\left(\omega_e i_d - i'_q\right) + k_e \cdot \omega_e \right\}} \begin{bmatrix} -\sin(\theta_{err}) \\ \cos(\theta_{err}) \end{bmatrix}$$

The BEMF amplitude is dependent on the rotor speed

Figure 18. Blocked-rotor test algorithm

The fault check can be disabled using `M1_DIAG_BLOCK_ROTOR_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.2 Disconnected phase test (M1.DIAG.PHLOSS)

The goal is check for disconnected phase or malfunctioning MOSFET.

The test is executed during HI_SPD state in the slow-loop SL (evaluation) and fast-loop FL (measurement).

Permanently open MOSFET eventually causes over-current condition (see short states in Table 5 below). Disconnected phase or permanently closed MOSFET causes permanently zero *idcb_rc* sample in one or more SVM sectors. The algorithm detecting such condition is shown in Figure 19 below. Both *idcb_rc* samples for given SVM sector are compared against minimal threshold constant *idcb_rc_min*. If |*idcb_rc*| < *idcb_rc_min* condition persists for too long, the fault condition is activated.

Table 5. The *idcb_rc* signal during phase-loss or MOSFET malfunction

SVM Sector		1	2	3	4	5	6
State	Sample						
All phases OK	*idcb_rc* #1	IA	IB	IB	IC	IC	IA
	idcb_rc #2	-IC	-IC	-IA	-IA	-IB	-IB
Phase_A lost	*idcb_rc* #1	0	IBC	IBC	ICB	ICB	0
	idcb_rc #2	IBC	IBC	0	0	ICB	ICB
Phase_B lost	*idcb_rc* #1	IAC	0	0	ICA	ICA	IAC
	idcb_rc #2	IAC	IAC	ICA	ICA	0	0
Phase_C lost	*idcb_rc* #1	IAB	IBA	IBA	0	0	IAB
	idcb_rc #2	0	0	IBA	IBA	IAB	IAB
pwm_at permanently ON	*idcb_rc* #1	IA	short	short	short	short	IA
	idcb_rc #2	-IC	-IC	short	short	-IB	-IB
pwm_at permanently OFF	*idcb_rc* #1	0	IB	IB	IC	IC	0
	idcb_rc #2	IBC	IBC	-IA	-IA	ICB	ICB
pwm_ab permanently ON	*idcb_rc* #1	short	IB	IB	IC	IC	short
	idcb_rc #2	short	short	-IA	-IA	short	short
pwm_ab permanently OFF	*idcb_rc* #1	IA	IBC	IBC	ICB	ICB	IA
	idcb_rc #2	-IC	-IC	0	0	-IB	-IB

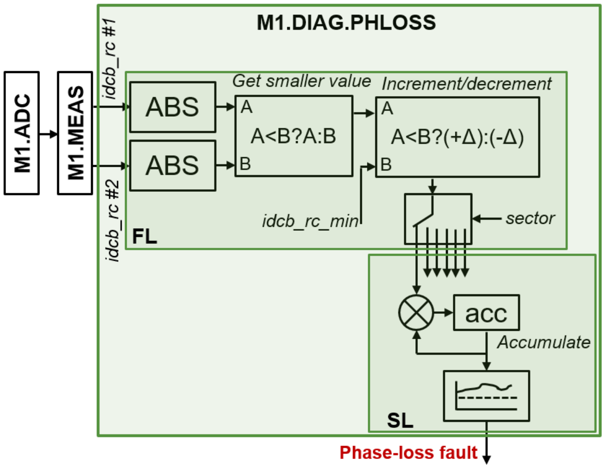


Figure 19. Phase-loss test algorithm block diagram

The fault check can be disabled using `M1_DIAG_PHLOSS_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.3 Rotor over-load test (M1.DIAG.LOAD)

The goal is check for motor over-load.

The test is executed during HI_SPD state in the slow-loop SL.

Block diagram of the over-load detection algorithm is shown in Figure 20 below. The fault condition is true when the maximal current (torque) is generated but the speed drops. Both following conditions must, therefore, be true at the same time:

- The rotor speed is below minimal value $\omega_e < \omega_{eOL}$.
- The actual stator current $I^2 = \left(i_\alpha^2 + i_\beta^2\right) > I_{OL}^2$ for a minimal time.

The threshold constant I_{OL}^2 should be configured to be close, but below the maximal current output value of speed controller. The speed threshold constant ω_{eOL} should be set above the under-speed fault threshold ω_{eUS}.

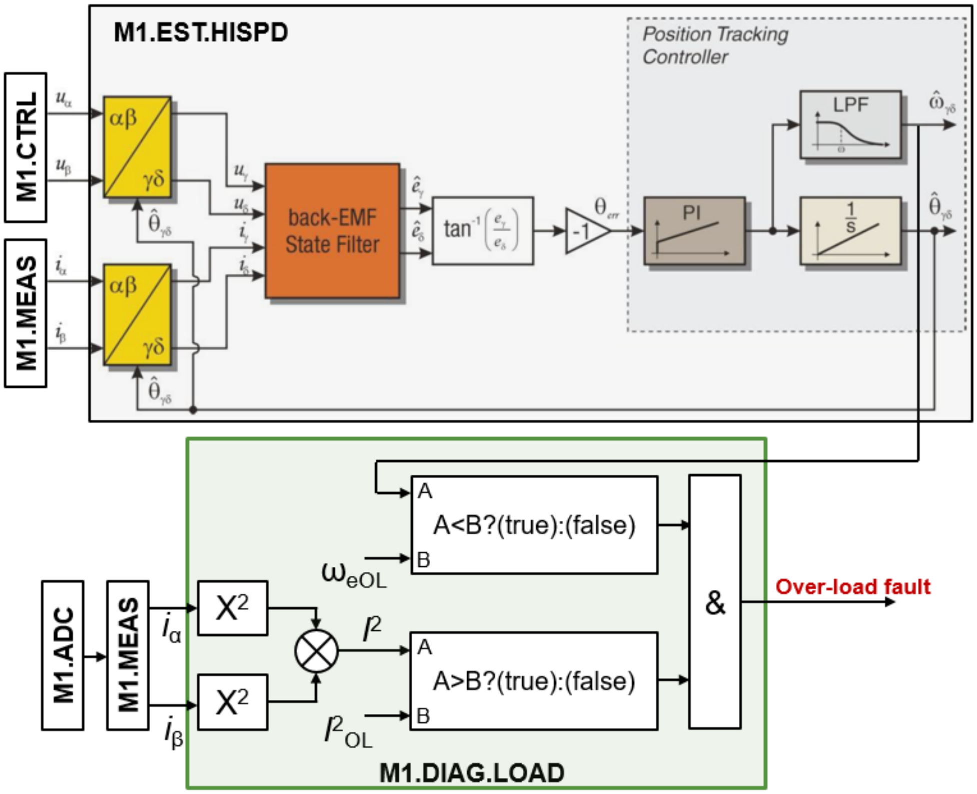


Figure 20. Over-load detection algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_LOAD_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.4 Over-temperature of medium (M1.DIAG.TMP_MED)

The goal is check T_{med} (quantity *medium_temp_rc* converted into Celsius degrees) for over-temperature.

The test is executed during all M1.SM states in the slow-loop SL.

The condition $T_{med} > T_{OTmed}$ is checked, where T_{OTmed} is the fault activation threshold constant (see block diagram in Figure 21).

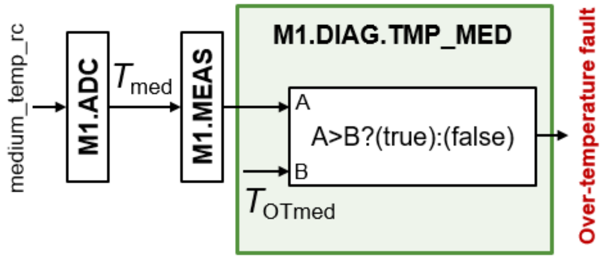


Figure 21. Medium over-temperature test algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_MED_TMP_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.5 Over-temperature of inverter (M1.DIAG.TMP_IPM)

The goal is check T_{IPM} (quantity *ipm_temp_rc* converted into Celsius degrees) for over-temperature.

The test is executed during all M1.SM states in the slow-loop SL.

The condition $T_{IPM} > T_{OTIPM}$ is checked, where T_{OTIPM} is the fault activation threshold constant (see block diagram in Figure 22).

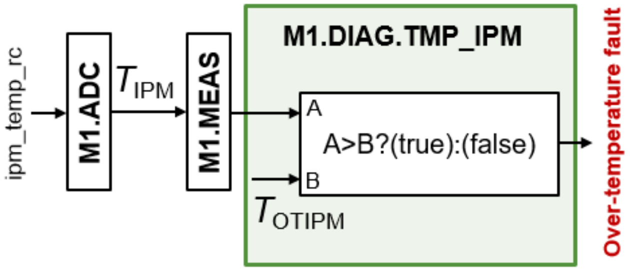


Figure 22. IPM over-temperature test algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_IPM_TMP_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.6 Over-temperature of MCU (M1.DIAG.TMP_MCU)

The goal is check T_{MCU} (quantity *mcu_temp* converted into Celsius degrees) for over-temperature.

The test is executed during all M1.SM states in the slow-loop SL.

The condition $T_{MCU} > T_{OTMCU}$ is checked, where T_{OTMCU} is the fault activation threshold constant (see block diagram in Figure 23).

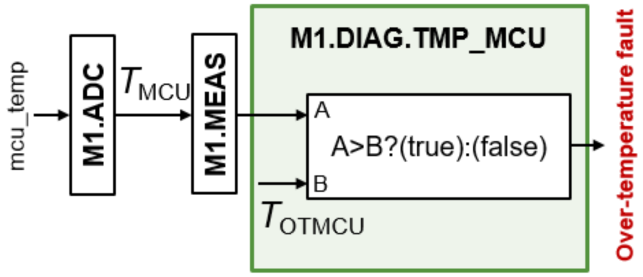


Figure 23. MCU over-temperature test algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_MCU_TMP_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.7 Under-/over-voltage test (M1.DIAG.UV_OV)

The goal is check DC-bus U_{dcb} voltage (converted vdcb_rc quantity) for over-/under-voltage.

The test is executed during all M1.SM states in the fast-loop FL.

The under-voltage condition $U_{dcb} < U_{dcbUV}$ and the over-voltage condition $U_{dcb} > U_{dcbOV}$ is checked, where U_{dcbUV} and U_{dcbOV} are fault activation threshold constants (see block diagram in Figure 24).

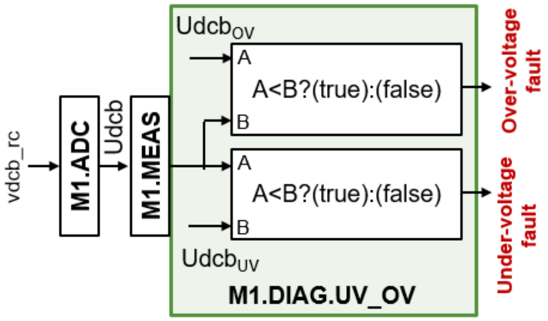


Figure 24. Under-/over-voltage test algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_MCU_TMP_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.8 Hardware over-current test (M1.DIAG.HWOC)

The goal is check DC-bus current *idcb_rc* for over-current.

The test is active at all times. Status is check by software in M1.SM during fast loop FL. The condition *idcb_rc* > *idcb_rc_thr* is constantly checked by M1_CMP_OC_PERIPH (see block diagram in Figure 25). If such condition is detected, the TRGOC trigger is generated and M1_PWM_PERIPH sets PWM outputs into inactive state without requiring software interaction. The status of M1.DIAG.HWOC is checked every FL in M1.SM. If the fault activation is detected, the M1.SM enters FAULT state.

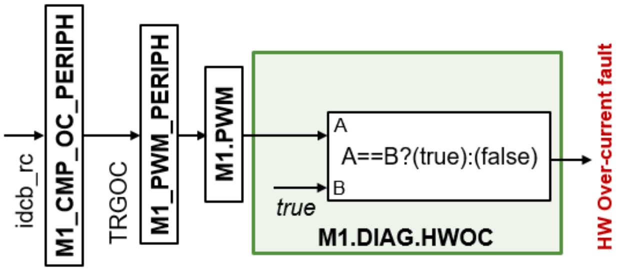


Figure 25. Hardware over-current test algorithm block diagram

4.2.9 Software over-current test (M1.DIAG.SWOC)

The goal is check squared stator current I^2 value for over-current.

The test is executed during all M1.SM states in the fast-loop FL.

The SW over-current fault protection checks condition $I^2 = i_\alpha^2 + i_\beta^2 > I_{swOC}^2$ (see block diagram in Figure 26). The SW over-current activation threshold constant I_{swOC} is generally set below the HW over-current threshold *idcb_rc_thr*.

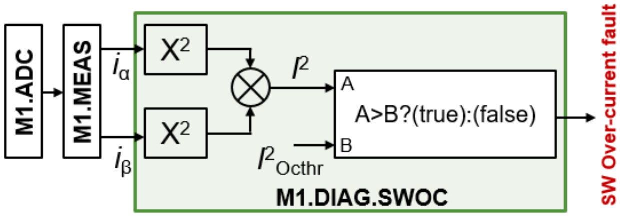


Figure 26. Software over-current test algorithm block diagram

4.2.10 Under-/over-power test (M1.DIAG.UP_OP)

The goal is check input power P for over-power and under-power (includes dry run) condition.

The test is executed during HI_SPD state in the slow loop SL.

The under-power protection checks condition $P = \frac{3}{2}(i_d u_d + i_q u_q) < P_{UP}$ and the overpower protection checks $P > P_{OP}$ (see block diagram in Figure 27). The thresholds $P_{UP} = f(\omega_e)$ and $P_{OP} = f(\omega_e)$ are speed-dependent and updated during runtime using two Look-Up Tables (LUT).

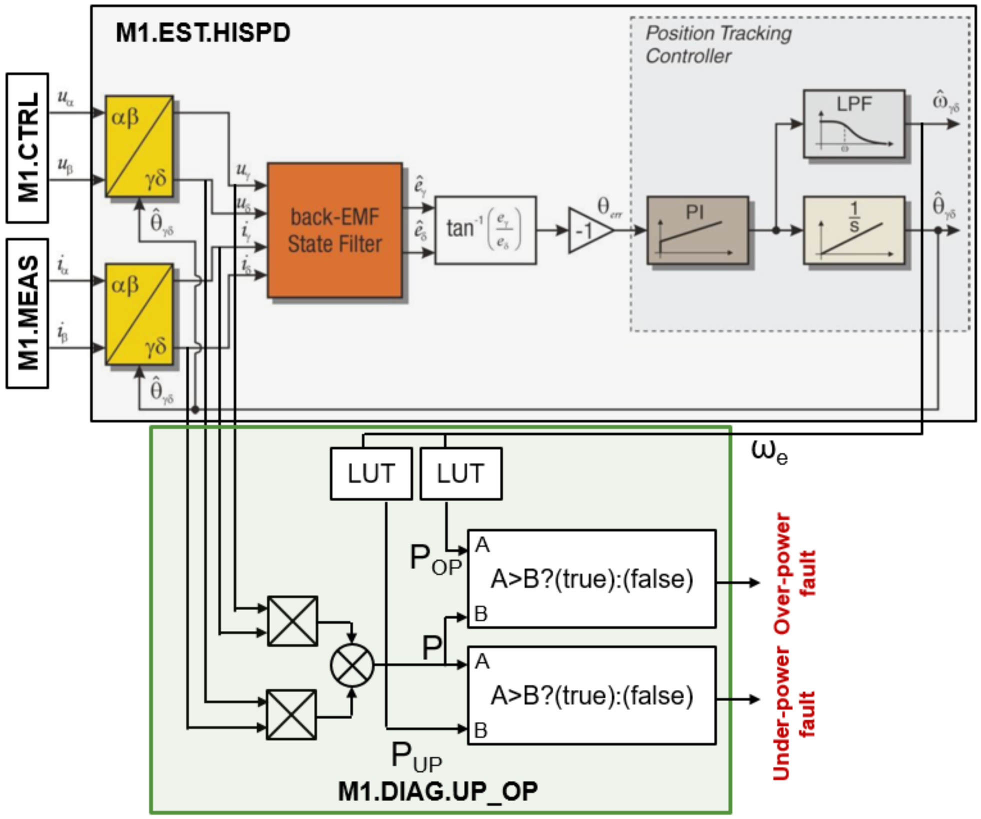


Figure 27. Over-/under-power test algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_POWER_ENABLE` and `M1_DIAG_UNDER_POWER_ENABLE` macros in *m1_pmsm_appconfig.h* file.

4.2.11 Under-/over-speed test (M1.DIAG.US_OS)

The goal is check estimated rotor speed ω_e for over-speed and under-speed condition.

The test is executed during HI_SPD state in the slow loop SL.

The under-speed protection checks condition $|\omega_e| < \omega_{eUS}$ and the over-speed protection checks condition $|\omega_e| > \omega_{eOS}$,

where ω_{eUS} and ω_{eUS} are fault activation threshold constants (see block diagram in Figure 28).

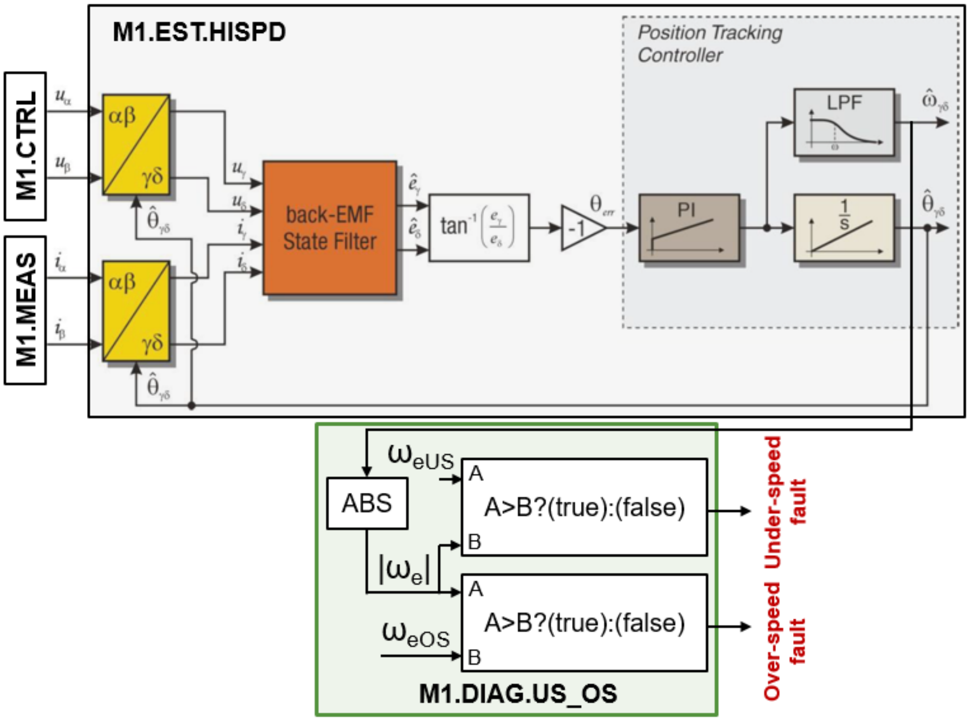


Figure 28. Over-/under-speed test algorithm block diagram

The fault check can be disabled using `M1_DIAG_OVER_SPEED_ENABLE` and `M1_DIAG_UNDER_SPEED_ENABLE` macros in *m1_pmsm_appconfig.h* file.

4.2.12 Stator resistance test (M1.DIAG.RES)

The goal is check estimated stator resistance for valid range.

The test is executed during HI_SPD and ALIGN state in the slow loop SL.

There are two checks executed, which both aim to detect stator resistance change:

1. In ALIGN M1.SM state: This stator resistance test checks for invalid conditions $I^2 < I^2_{AUC}$ and $I^2 > I^2_{AOC}$, where I^2_{AUC} and I^2_{AOC} are alignment current limits (see block diagram in Figure 29). The test is enabled only in the last quarter of the alignment duration $time < \frac{3}{4} T_{align}$ so the current has time to settle. The aim of this test is to check that the stator current measurement (including current measurement scale) is correct before the current controllers are engaged (see M1.CTRL.SPEED_CL control mode description).

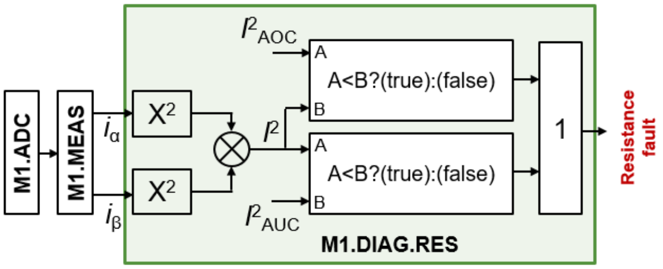


Figure 29. Stator resistance test algorithm during ALIGN block diagram

2. In HI_SPD M1.SM state: The stator resistance test checks for invalid conditions $R < R_{MIN}$ and $R > R_{MAX}$, where R_{MIN} and R_{MAX} are expected stator resistance limits (see block diagram in Figure 30). This test relies on the M1.EST.RES output and its goal is to monitor for changes in the estimated stator resistance during runtime.

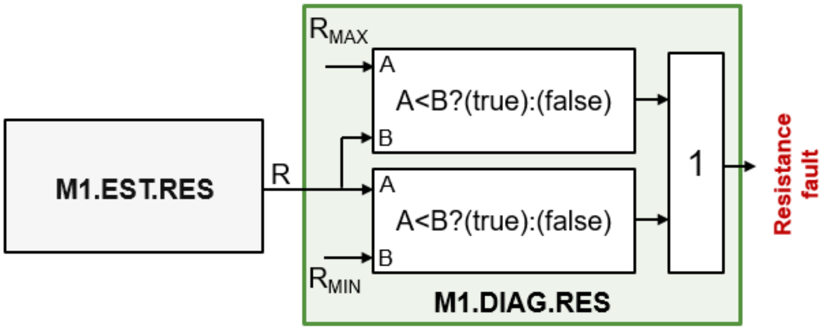


Figure 30. Stator resistance test algorithm during HI_SPD block diagram

The fault check can be disabled using `M1_DIAG_REST_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.2.13 Power stability test (M1.DIAG.PWRSTAB)

The goal is check for input power P steady state condition violation.

The test is executed during HI_SPD state in the slow-loop SL.

An ability of system to reach steady state in defined time is checked. A following algorithm (see timing diagram in Figure 31 and block diagram in Figure 32) is used:

1. Input power $P = \frac{3}{2}(i_d u_d + i_q u_q)$ is filtered by two levels of Low-Pass Filters (LPFs) $P_{filt1} = LPF\{P\}$ and $P_{filt2} = LPF\{P_{filt1}\}$. Setup of this filters affects sensitivity of the algorithm to fast input power deviations.
2. The difference between two different filtered powers is accumulated $\Delta P_{accum}(k+1) = \mathrm{SAT}\{\Delta P_{accum}(k) + \left[\left|P_{filt1}(k) - P_{filt2}(k)\right| - \Delta P_{limit}\right]\}$, where ΔP_{limit} is power difference limit constant and ΔP_{accum} accumulated power difference.

3. The accumulated power difference is then compared to fault threshold $\Delta P_{accum} > \Delta P_{UNSTAB}$.

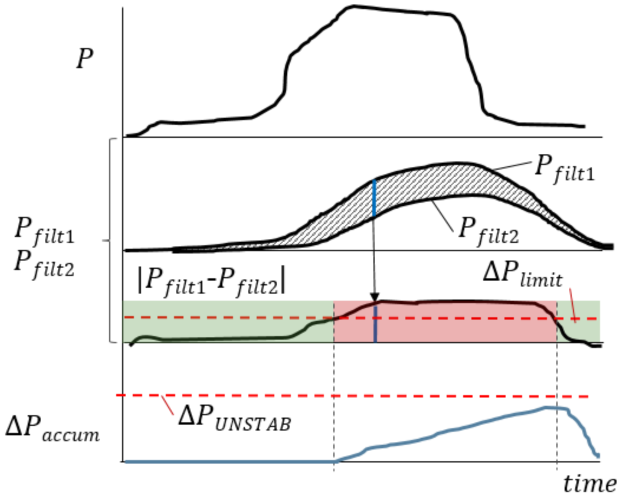


Figure 31. Power stability test algorithm time diagram

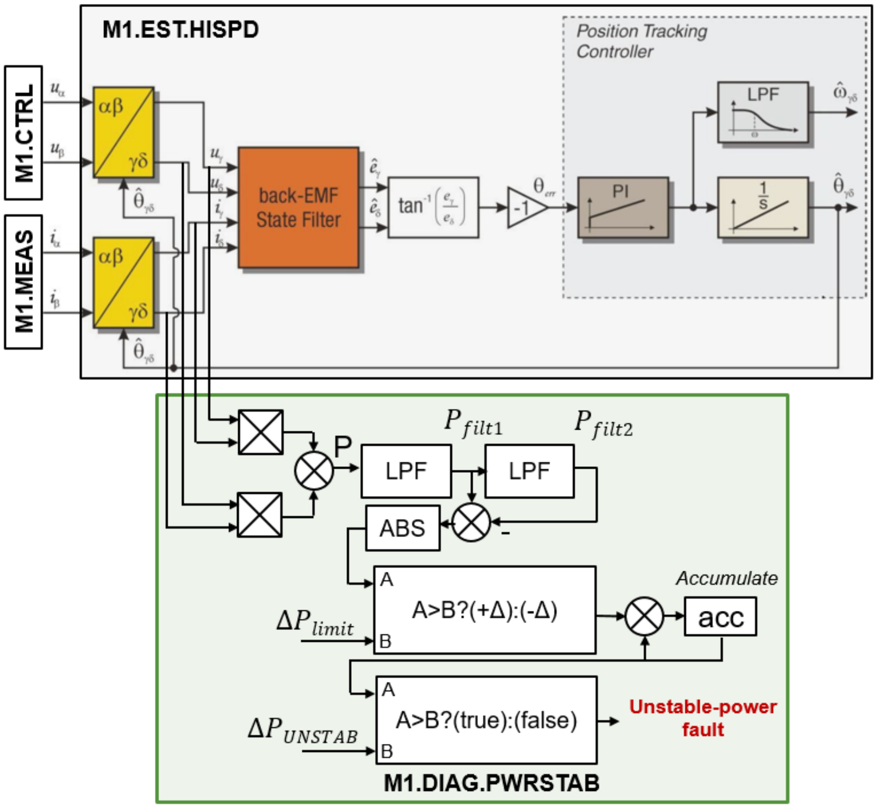


Figure 32. Power stability test algorithm block diagram

The fault check can be disabled using `M1_DIAG_UNSTAB_PWR_ENABLE` macro in *m1_pmsm_appconfig.h* file.

4.3 State machine

From motor-control point of view, the state machine is responsible for conducting following steps:

- Measure (M1.MEAS) – M1.ADC driver is called so all quantities (phase currents, DC-bus voltage, temperatures) are available. Safety relevant step.
- Estimate (M1.EST) – The stator resistance and the rotor position and speed estimation algorithms are called. The used algorithms differ based on the MC.SM state. Safety relevant step.
- Diagnostics (M1.DIAG) – Measured and estimated quantities are analyzed by various algorithms so unsafe conditions can be detected. The algorithm differs based on the MC.SM state. Safety relevant step.
- Control (M1.CTRL) – The control algorithm executed in M1.CTRL. The current and speed control loops are executed to obtain required stator voltage. This step is not safety relevant.
- Actuate (M1.ACT) – The DC-bus ripple and dead-time compensations are applied and the Space Vector Modulation (SVM) algorithm calculates the required phase duty cycles for M1.PWM driver.

These steps are, however, dependent on the actual motor state (namely rotor speed), and *M1SM_RequestStart*/ *M1SM_RequestStop* request issued by M1.CTRL module. The role of the M1.SM state-machine is, therefore, implemented right timing, machine start and stop, state transfer, and ensure that right algorithms are executed. The high-level M1.SM execution flowchart is shown in Figure 33. Following M.1SM states (described in more detail later in this section) were implemented:

- NO_INIT – Default M1.SM state prior initialization.
- FAULT – Entered when any M1.DIAG pending fault is detected by diagnostic algorithms. The STOP state is entered when no fault is observed for a configured time.
- STOP – The M1.SM is idle and the *M1SM_RequestStart* request form M1.CTRL is awaited.
- ALIGN – Rotor alignment state.
- LO_SPD – Open-loop startup state
- MI_SPD – The position open-loop to the position closed-loop transition.
- HI_SPD – High-speed state with closed-loop speed control.
- FREE – The freewheel state where no toque is applied for a constant time to allow the rotor to slow down.

All implemented M1.SM states are executed in both fast-loop FL and slow-loop SL. State transition is controlled in the fast-loop FL.

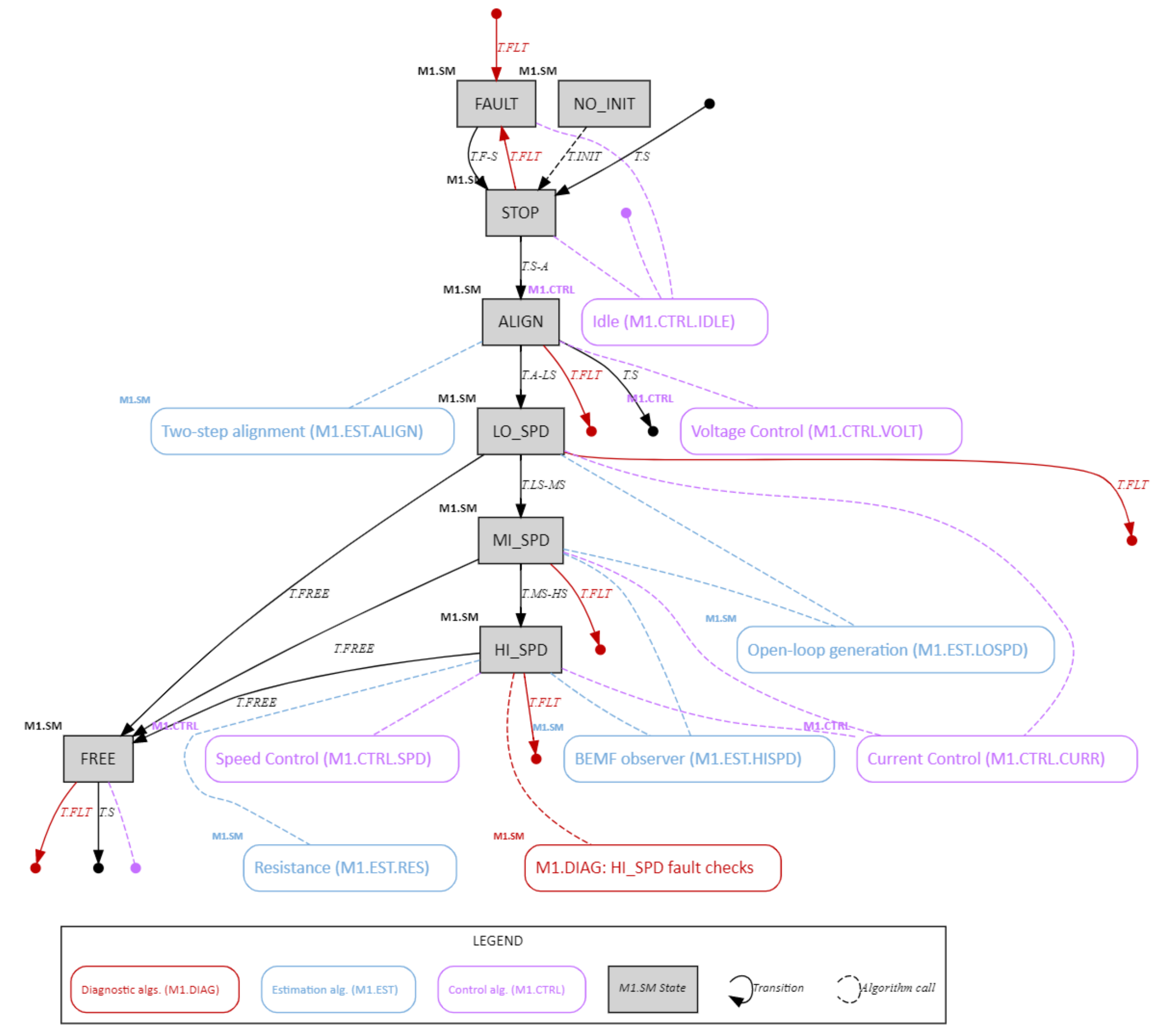


Figure 33. M1.SM state-machine flowchart

4.3.1 Uninitialized state (NO_INIT)

Goal: Enter safety error state when initialized M1.SM state-machine SL or FL routine is executed.

Execution: The state should never be executed.

Transitions:

- T.INIT – The STOP state is entered when the M1.SM state machine initialization routine is executed during AR.

Called M1.CTRL algorithm: none

Called M1.EST algorithm: none

Called M1.DIAG algorithm: none

Details: The default state prior to the M1.SM state-machine initialization. The M1.SM initialization is expected to be done by a separate function during AR phase, so normally the NO_INIT state should not occur, and its execution is considered to be safety error trigger.

4.3.2 Idle state (STOP)

Goal: Await *M1SM_RequestStart* request from M1.CTRL.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.FLT – The M1.DIAG fault was detected. The M1_PWM_PERIPH output is immediately disabled. All M1.SM internal state variables are cleared.
- T.S-A – The ALIGN state is entered when *M1SM_RequestStart* request is received from M1.CTRL.

Called M1.CTRL algorithm: M1.CTRL.IDLE

Called M1.EST algorithm: none

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD

Details: The portion relevant to this state is highlighted in Figure 34. This is the default idle state of M1.SM, during which the M1.CTRL.IDLE control subroutine is called to receive *M1SM_RequestStart* request (results in T.S-A transition). The M1_PWM_PERIPH output signals are disabled.

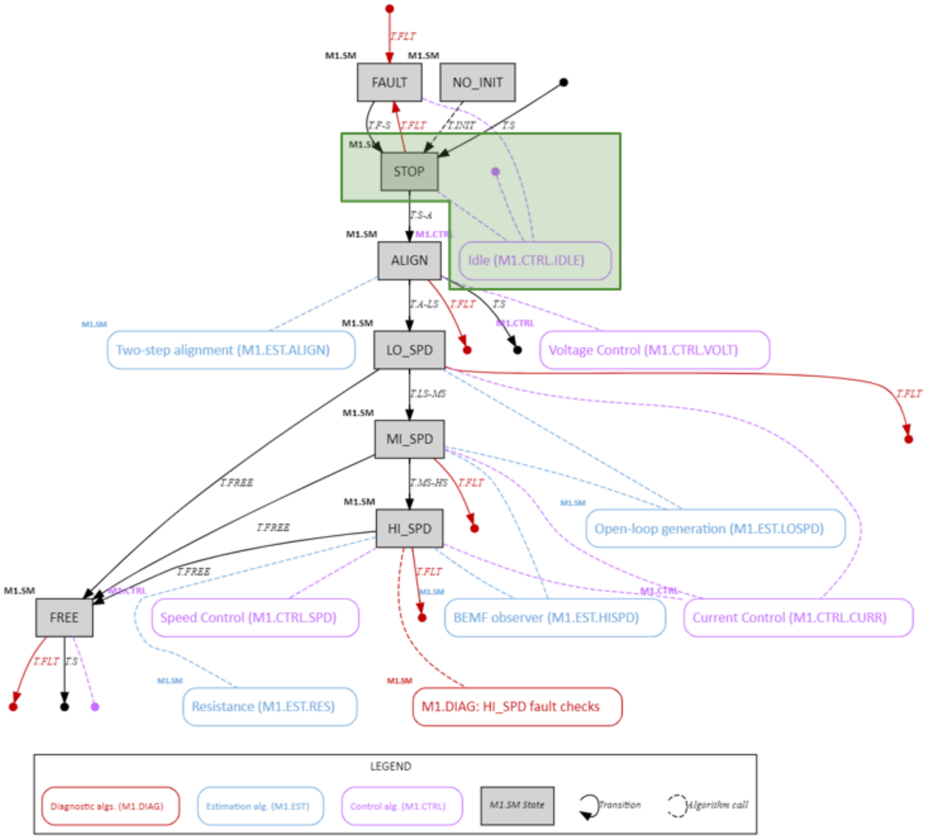


Figure 34. Highlighted M1.SM STOP state

4.3.3 Fault state (FAULT)

Goal: Await for pending recoverable fault condition to disappear.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.S – The STOP state is entered when fault recovery time T_{fault} passes after all pending M1.DIAG faults disappear. All M1.SM internal state variables are cleared.

Called M1.CTRL algorithm: M1.CTRL.IDLE

Called M1.EST algorithm: none

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD

Details: The portion relevant to this state is highlighted in Figure 35. The FAULT state is entered whenever a recoverable M1.DIAG fault is diagnosed in any M1.SM state. The state is left only when the fault condition disappears for at least T_{fault}. The M1_PWM_PERIPH output signals are disabled.

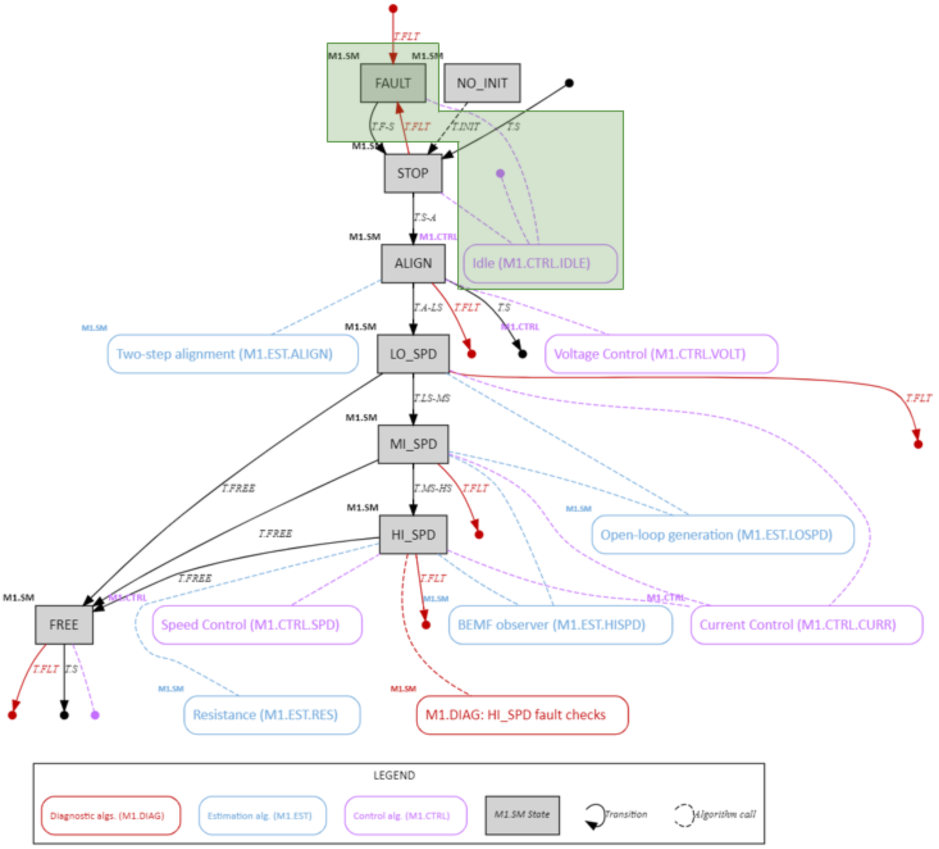


Figure 35. Highlighted M1.SM FAULT state

4.3.4 Alignment state (ALIGN)

Goal: Align rotor into a known position before the motor startup procedure can begin.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.S – The STOP state is entered when request *M1SM_RequestStop* is received from M1.CTRL. All M1.SM internal state variables are cleared.
- T.FLT – The M1.DIAG fault was detected. The M1_PWM_PERIPH output is immediately disabled. All M1.SM internal state variables are cleared.
- T.A-LS – The alignment procedure was completed after T_{align} passed.

Called M1.CTRL algorithm: M1.CTRL.VOLT

Called M1.EST algorithm: M1.EST.ALIGN

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD

Details: The portion relevant to this state is highlighted in Figure 36. Before any non-zero stator voltage is applied (M1.CTRL.VOLT output is applied), two steps are performed:

1. The PWM is enabled – The bootstrap circuit is charged using the M1.PWM.START algorithm.
2. The startup calibration – The phase current measurement offsets are measured in M1.ADC.CALIB algorithm so these can removed during further operation.

Once the PWM is reliably enabled and calibration is completed, the M1.CTRL.VOLT output is applied for T_{align} duration. The voltage vector position is given by M1.EST.ALIGN algorithm (see Section 4.4.2.1). The stator voltage and position during ALIGN state is shown in Figure 37.

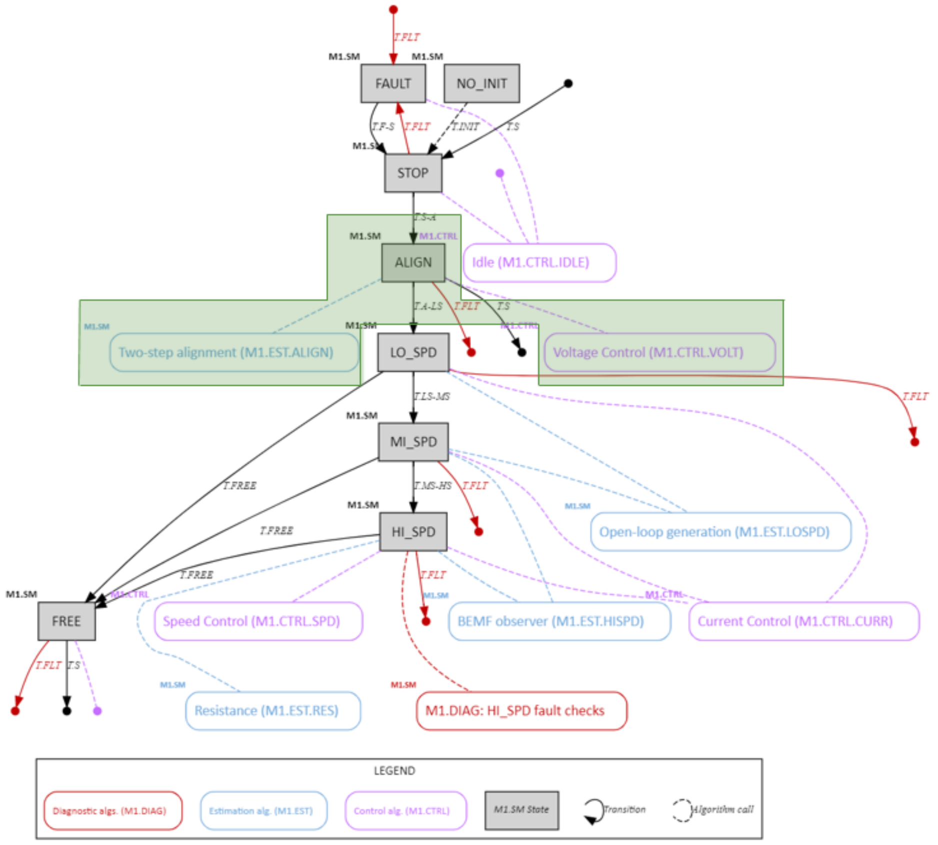


Figure 36. Highlighted M1.SM ALIGN state

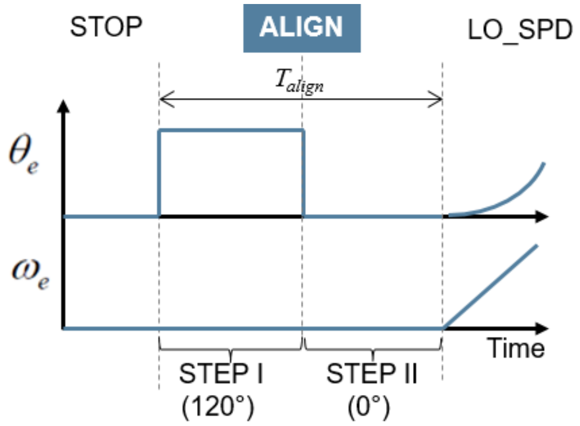


Figure 37. Stator voltage position and amplitude during ALIGN state

4.3.5 Low-speed state (LO_SPD)

Goal: Accelerate the rotor until a minimal M1.EST.HISPD speed is reached.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.FREE – The STOP state is entered when request *M1SM_RequestStop* is received from M1.CTRL.
- T.FLT – The M1.DIAG fault was detected. The M1_PWM_PERIPH output is immediately disabled. All M1.SM internal state variables are cleared.
- T.LS-MS – The minimal speed was reached by M1.EST.LOSPD algorithm and the LO_SPD state is entered.

Called M1.CTRL algorithm: M1.CTRL.CURR

Called M1.EST algorithm: M1.EST.LOSPD

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD

Details: The portion relevant to this state is highlighted in Figure 38. This is the sensorless rotor startup state, during which the stator current vector is rotating with constantly increasing speed (see Figure 39). The M1.CTRL.CURR control subroutine is called to maintain the stator current amplitude. Once the startup speed reaches the BEMF observer activation threshold ω_{elsthr}, the MI_SPD state is entered via T.LS-MS transition. Alternatively, if the *M1SM_RequestStart* request is received from M1.CTRL, the FREE state is entered via T.FREE transition.

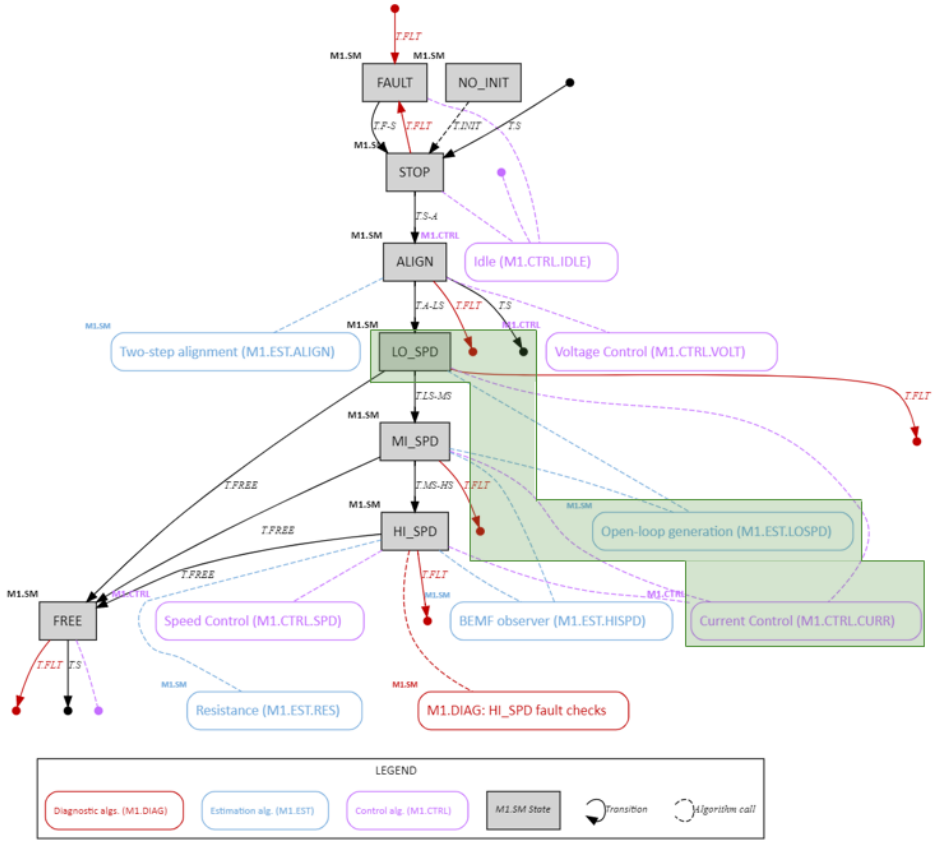


Figure 38. Highlighted M1.SM LO_SPD state

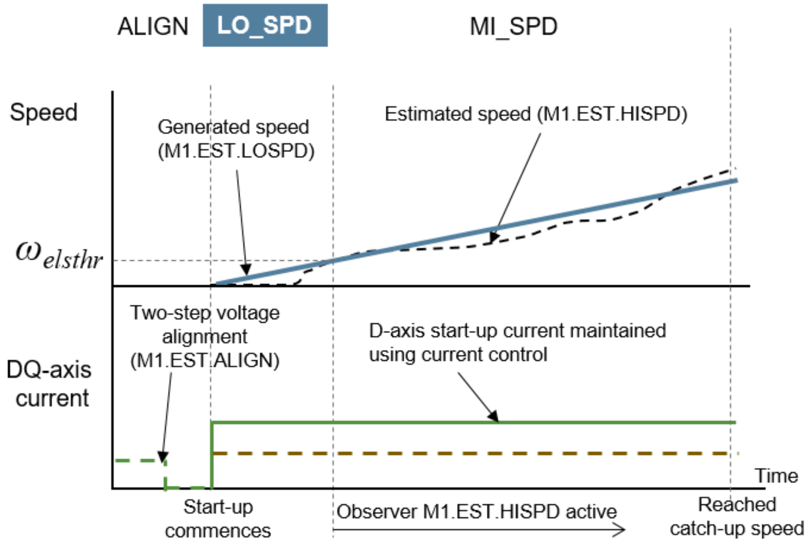


Figure 39. Stator current position, speed, and amplitude during LO_SPD state

4.3.6 Medium-speed state (MI_SPD)

Goal: Continue accelerating until M1.EST.HISPD speed estimate can be used to drive motor.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.FREE – The STOP state is entered when request *M1SM_RequestStop* is received from M1.CTRL.
- T.FLT – The M1.DIAG fault was detected. The M1_PWM_PERIPH output is immediately disabled. All M1.SM internal state variables are cleared.
- T.MS-HS – The speed, where the M1.EST.LOSPD estimates become reliable was reached.

Called M1.CTRL algorithm: M1.CTRL.CURR

Called M1.EST algorithm: M1.EST.LOSPD and M1.EST.HISPD

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD

Details: The portion relevant to this state is highlighted in Figure 40. The M1.CTRL.CURR control routine is called to perform the stator current control and the M1.EST.HISPD algorithm is running on the background (see Figure 41). Once the low-speed algorithm M1.EST.LOSPD speed reaches merge threshold ω_{emin}, the HI_SPD state is entered via T.MS-HS transition. Alternatively, if the *M1SM_RequestStart* request is received from M1.CTRL, the FREE state is entered via T.FREE transition.

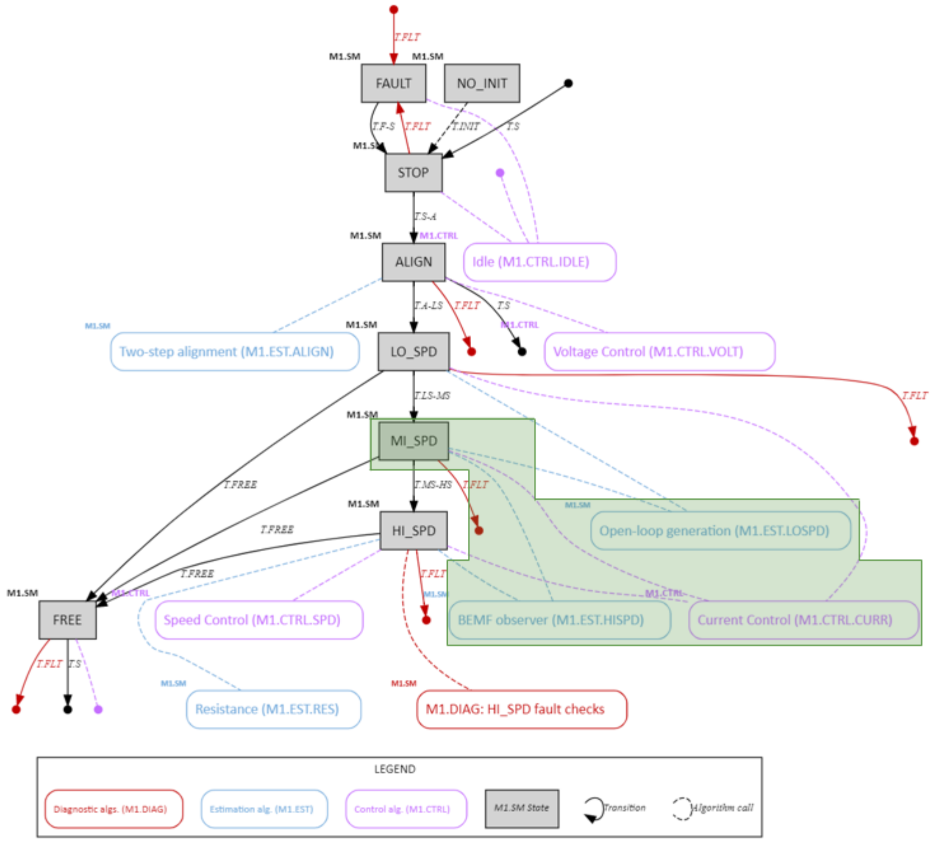


Figure 40. Highlighted M1.SM MI_SPD state

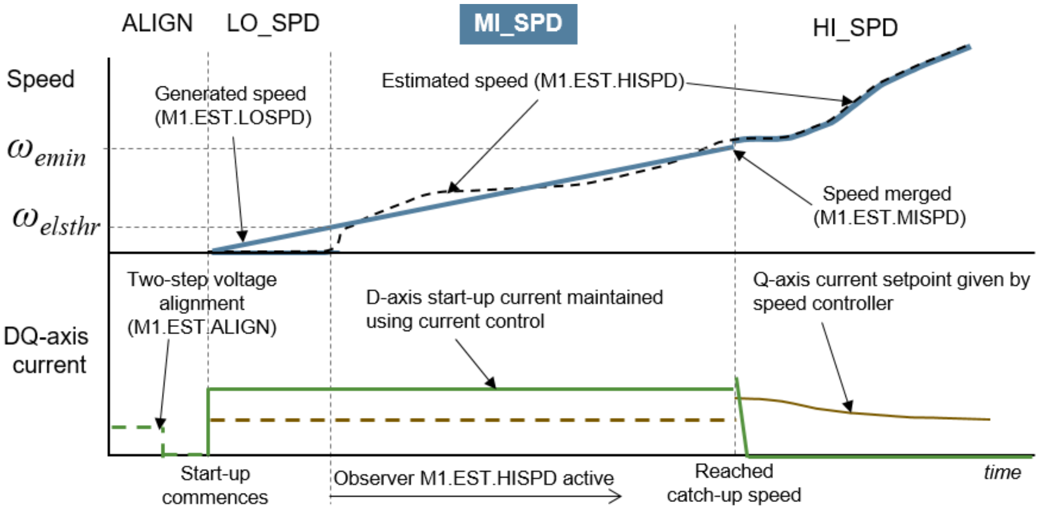


Figure 41. Figure Stator current position, speed, and amplitude during MI_SPD state

4.3.7 High-speed State (HI_SPD)

Goal: Normal motor operation.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.FREE – The STOP state is entered when request *M1SM_RequestStop* is received from M1.CTRL.
- T.FLT – The M1.DIAG fault was detected. The M1_PWM_PERIPH output is immediately disabled. All M1.SM internal state variables are cleared.

Called M1.CTRL algorithm: M1.CTRL.CURR and M1.CTRL.SPD

Called M1.EST algorithm: M1.EST.HISPD

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD, M1.DIAG.PWRSTAB, M1.DIAG.RES, M1.DIAG.UP_OP, M1.DIAG.US_OS, M1.DIAG.PHLOSS, M1.DIAG.LOAD, M1.DIAG.BLCKROT

Details: The portion relevant to this state is highlighted in Figure 42. The normal high-speed operation state, during which all M1.DIAG algorithms are active. The M1.CTRL.CURR and M1.CTRL.SPD control subroutines are called to perform the stator current and speed control. If the *M1SM_RequestStart* request is received from M1.CTRL, the FREE state is entered via T.FREE transition.

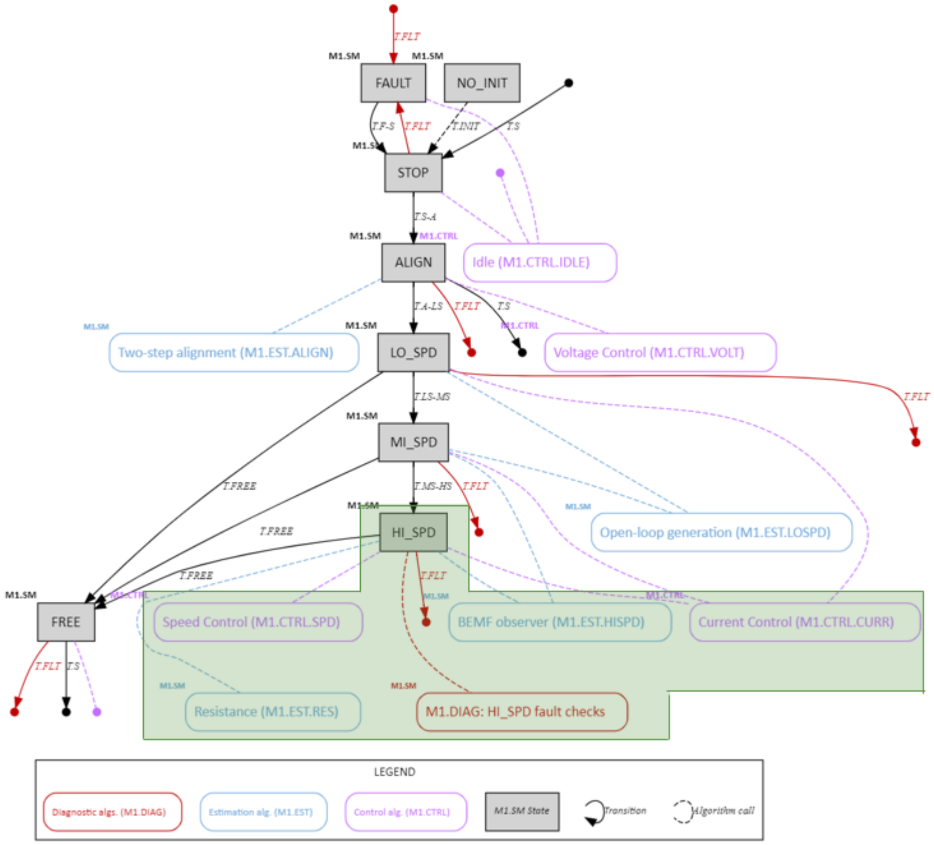


Figure 42. Highlighted M1.SM HI_SPD state

4.3.8 Freewheel state (FREE)

Goal: Await T_{free} period so the rotor can slow down before *M1SM_RequestStart* request can be received again.

Execution: Fast-loop FL and slow-loop SL.

Transitions:

- T.S – The STOP state is entered after T_{free} period passes. All M1.SM internal state variables are cleared.
- T.FLT – The M1.DIAG fault was detected. The M1_PWM_PERIPH output is immediately disabled. All M1.SM internal state variables are cleared.

Called M1.CTRL algorithm: none

Called M1.EST algorithm: none

Called M1.DIAG algorithm: M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM, M1.DIAG.TMP_MCU, M1.DIAG.UV_OV, M1.DIAG.SWOC, M1.DIAG.HWOC, M1.DIAG.EXTCMD

Details: The portion relevant to this state is highlighted in Figure 43. The PWM signals are disabled during this state. Once the T_{free} period after entering this state expires, the STOP state is entered via T.S transition.

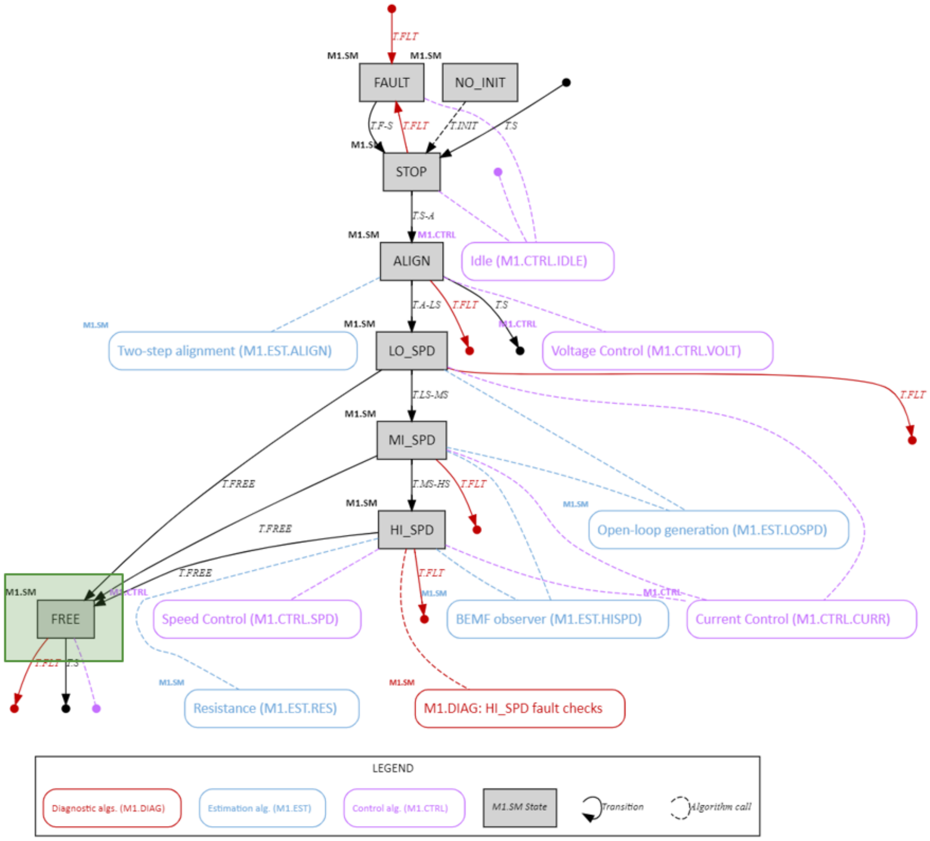


Figure 43. Highlighted M1.SM FREE state

4.4 Motor control software algorithms

This section describes notable individual motor-control algorithms. A detailed block diagram of implemented PMSM pump Field-Oriented Control algorithm is in Figure 44. Software blocks, which participate in the motor-control are:

- APP.EXTCMD – The APP_EXTCMD_PERIPH driver responsible for the external pump rotor speed command acquisition.
- APP – The application layer called by MAIN module. By default, the external speed command (or command received by FMSTR) and *M1SM_RequestStart* / *M1SM_RequestStop* command is passed to the M1.CTRL module via M1.CTRL.I2 API.
- M1.CTRL – The action control module, responsible for calculation of required stator voltage based on provided measurements, estimations, and required values (commands). It is called via:
  - M1.CTRL.I2 – Called by APP application layer to provide M1.SM state and to receive speed command (or current, voltage or frequency command from FMSTR).
  - M1.CTRL.I1 – Called by M1.SM to provide *M1SM_RequestStart* / *M1SM_RequestStop* request and required stator voltage (execute internal current and speed controllers).
- M1.SM – The safe motor-control state-machine. It is responsible for following steps:
  1. Measure (M1.MEAS) – M1.ADC driver is called so all quantities (phase currents, DC-bus voltage, temperatures) are available. Safety relevant step.
  2. Estimate (M1.EST) – The stator resistance and the rotor position and speed estimation algorithms are called. The used algorithms differ based on the MC.SM state. Safety relevant step.
  3. Diagnostics (M1.DIAG) – Measured and estimated quantities are analyzed by various algorithms so unsafe conditions can be detected. The algorithm differs based on the MC.SM state. Safety relevant step.

4. Control (M1.CTRL) – The control algorithm executed in M1.CTRL. The current and speed control loops are executed to obtain required stator voltage. On top of the default closed loop speed FOC control algorithm M1.CTRL.SPEED_CL, several other control modes are implemented to support development phase of the customers software. This step is not safety relevant.
5. Actuate (M1.ACT) – The DC-bus ripple and dead-time compensations are applied and Space Vector Modulation (SVM) algorithm calculates the required phase duty cycles for M1.PWM driver.

- M1.ADC – The M1_ADC_PERIPH and FS_ADC_PERIPH driver responsible for safe analog quantity sample acquisition. A single-shunt current reconstruction algorithm is implemented.
- M1.PWM – The M1_PWM_PERIH driver responsible for three-phase shifted-PWM generation.

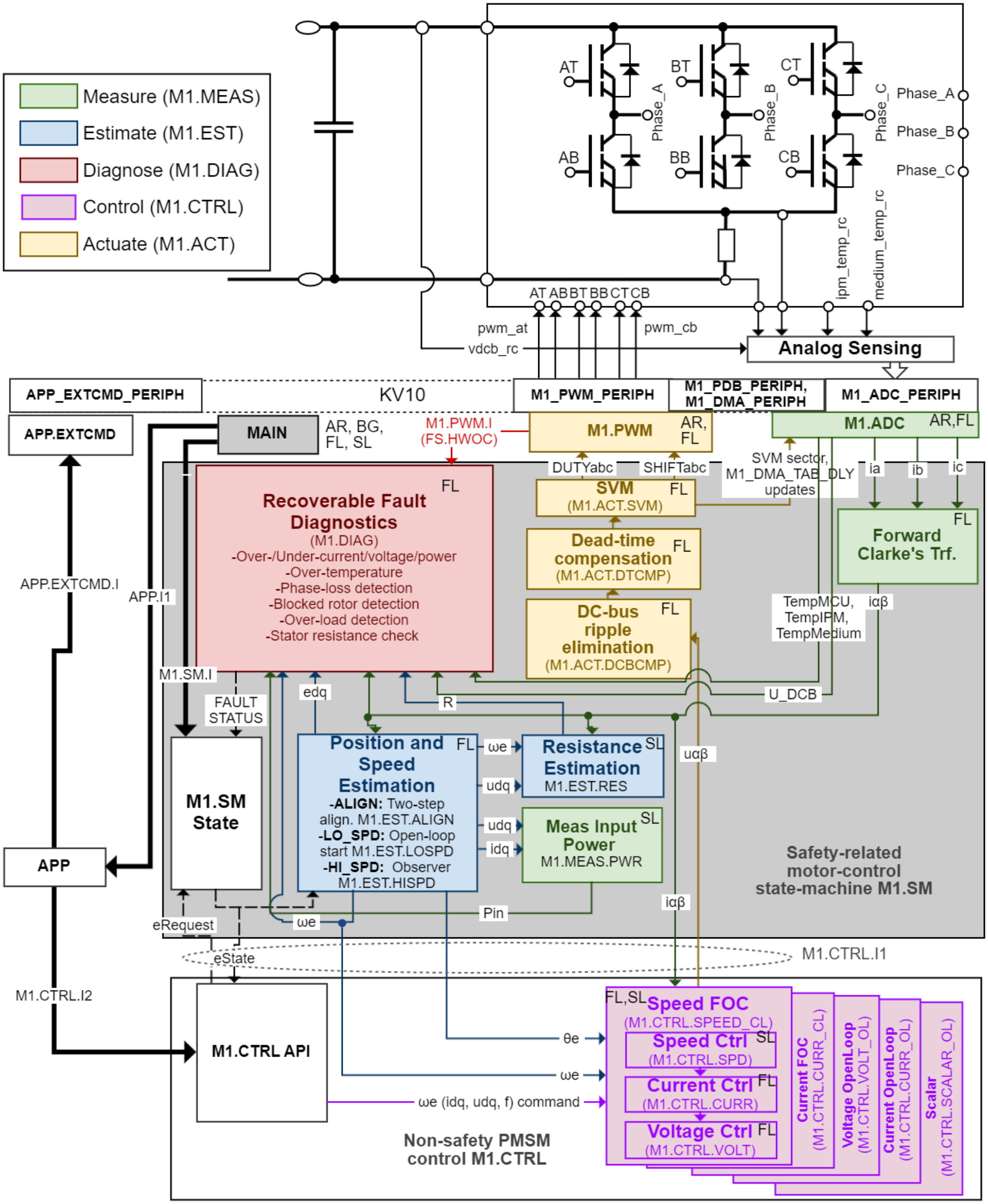


Figure 44. Sensorless PMSM field-oriented control algorithm blocks

4.4.1 Measurement algorithms (M1.ADC, M1.MEAS)

This section describes notable analog measurement algorithms of M1.ADC and M1.SM modules.

4.4.1.1 Single-shunt three-phase current reconstruction (M1.ADC)

Goal: Reconstruct three-phase currents based on a single-shunt voltage drop *idcb_rc* measurements.

Execution: In the fast loop FL during all M1.SM states.

Details: The M1.ADC peripheral connection and timing was already explained. This section explains the three-phase current reconstruction principle from a single-shunt current measurement. The shifted-PWM generation, which is closely tied to the phase current reconstruction, is explained in Section 4.4.5.1. The single-shunt phase-current reconstruction and shifted-PWM algorithms were chosen to achieve a minimal hardware cost.

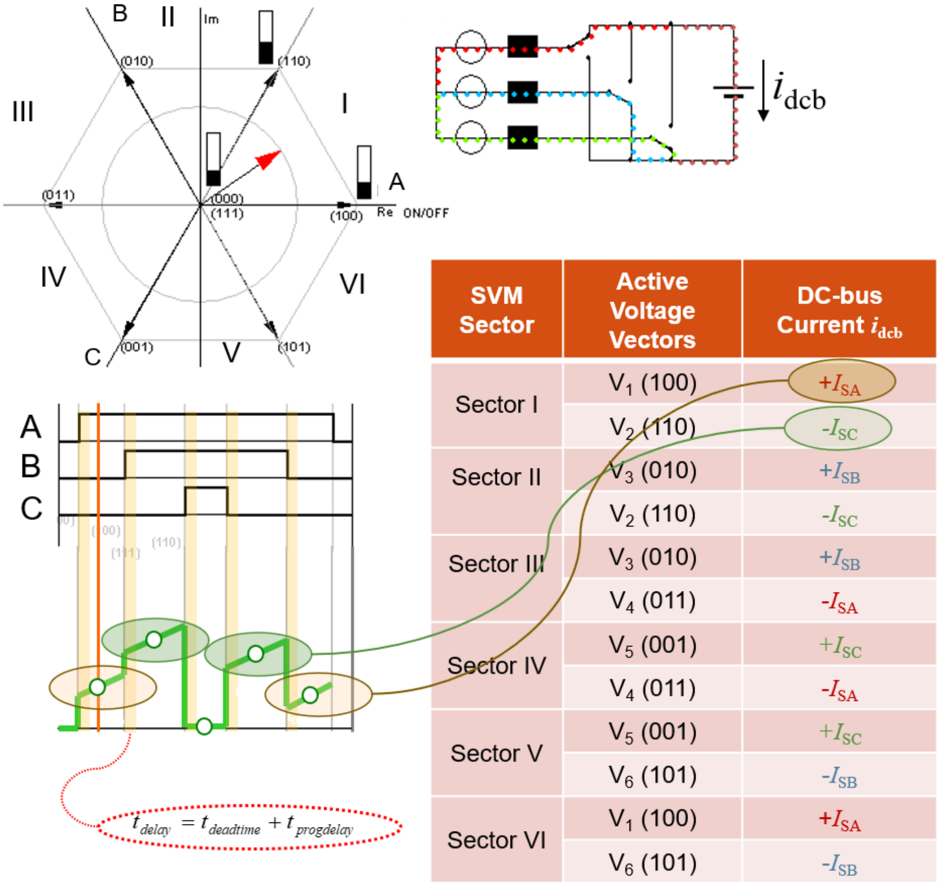
The M1.ADC driver was designed to allow acquisition of large number of variably placed ADC samples. Its benefits are minimal CPU assistance, a large number of quantities in exact times can be acquired every T_{PWM} period (including variably placed *idcb_rc* samples for phase current reconstruction), and the fact that the second converter FS_ADC_PERIPH is free for safety compare FS.CMP test. Total of N_{smpl} = 10 quantities are sampled:

1. The first *idcb_rc* current sample – Used for phase current reconstruction. Sample position changes.
2. The second *idcb_rc* current sample - Used for phase current reconstruction. Sample position changes.
3. The *idcb_rc* current offset measurement - Used for online phase current offset calibration (see Section 4.4.1.2). Sample is measured during V_{111} voltage vector.
4. Voltage reference VREFL – Used for FS.REF and FS.CMP tests.
5. Voltage reference VREFH – Used for FS.REF and FS.CMP tests.
6. Voltage reference band gap – Used for FS.REF and FS.CMP tests.
7. Inverter temperature *ipm_temp_rc* – Used for M1.DIAG.TEMP_IPM test.
8. Medium temperature *medium_temp_rc* – Used for M1.DIAG.TEMP_MED test.
9. MCU temperature *mcu_temp* – Used for M1.DIAG.TEMP_MCU test.
10. DC-bus voltage *vdcb_rc* – Used for control (DC-bus ripple compensation M1.ACT.DCBCMP) M1.DIAG.UV_OV tests.

As the block diagram in Figure 36 shows, the DC-bus current i_{dcb} can be measured via voltage drop on the shunt resistor R52 connecting GND and bottom inverter MOSFTEs. Whenever at least one bottom MOSFET is enabled, the voltage drop idcb_rc will increase or decrease from its default 1.65V value accordingly to the current flowing. When using standard Space Vector Modulation for PWM signal generation, four voltage vectors will be applied to motor phases each PWM period:

- Inactive voltage vector V_0 (000) – All bottom MOSFETs are enabled.
- Inactive voltage vector V_7 (111) – All top MOSFETs are enabled.
- Two active vectors - See table in Figure 45.

Two phase currents are, therefore, normally available as DC-bus current i_{dcb} (or idcb_rc voltage) during active voltage vectors each PWM period. Therefore, it is possible to reconstruct all phase currents by measuring **two different** idcb_rc samples of i_{dcb} per PWM period and calculating the third phase current using Kirchoff's law. The i_{dcb} is zero during inactive voltage vectors V_0 (000) and V_7 (111). This is used for online calibration of measurement offsets.



SVM Sector	Active Voltage Vectors	DC-bus Current i_{dcb}
Sector I	V_1 (100)	$+I_{SA}$
	V_2 (110)	$-I_{SC}$
Sector II	V_3 (010)	$+I_{SB}$
	V_2 (110)	$-I_{SC}$
Sector III	V_3 (010)	$+I_{SB}$
	V_4 (011)	$-I_{SA}$
Sector IV	V_5 (001)	$+I_{SC}$
	V_4 (011)	$-I_{SA}$
Sector V	V_5 (001)	$+I_{SC}$
	V_6 (101)	$-I_{SB}$
Sector VI	V_1 (100)	$+I_{SA}$
	V_6 (101)	$-I_{SB}$

Figure 45. Single-shunt phase current reconstruction

4.4.1.2 Phase-current measurement calibration (M1.ADC.CALIB)

Goal: Remove unwanted bias from phase current measurement.

Execution: In the fast loop FL. Runtime calibration at all times, the startup calibration at the beginning of ALIGN state.

Details: Two phase current offset compensation methods are implemented:

- **Runtime calibration:** The phase current offset is measured at V_7 (111) during each PWM period (third M1_ADC_PERIPH sample each PWM period – see Section 4.4.1.1) and subtracted from both idcb_rc samples (the first and the second M1_ADC_PERIPH sample each PWM period – see Section 4.4.1.1) during M1.ADC execution inf fast loop FL.

Startup calibration: Offsets are measured during ALIGN state of M1.SM for both idcb_rc samples (the first and the second M1_ADC_PERIPH sample – see Section 4.4.1.1) for all SVM sectors. A 50% duty cycles are applied to *Phase_A*, *Phase_B*, and *Phase_C* during this calibration so zero idcb_rc would be measured in ideal situation. This measured offsets are then subtracted from idcb_rc samples during runtime and remain constant during the rest of operation (LO_SPD, MI_SPD, and HI_SPD states of M1.SM). Correct offset to subtract is selected based on active SVM sector.

4.4.2 Estimations algorithms (M1.EST)

This section describes notable estimation algorithms of M1.SM module.

4.4.2.1 Alignment position generation (M1.EST.ALIGN)

Goal: Provide reliable initial electrical rotor position and speed.

Execution: In fast-loop FL during ALIGN state of M1.SM.

Details: The two-step alignment algorithm is used to set SYS.E.MOT rotor into known 0° position no matter the previous rotor position or speed and, therefore, ensure reliable motor startup. The stator voltage vector angle (electrical rotor position) is initially set to 120° position and changed after 0.5 T_{align} to 0° (see timing diagram in Figure 46). Unlike in case of single-step alignment, this prevents misalignment in cases, when the rotor is ~180° from the forced position (little torque would be generated then).

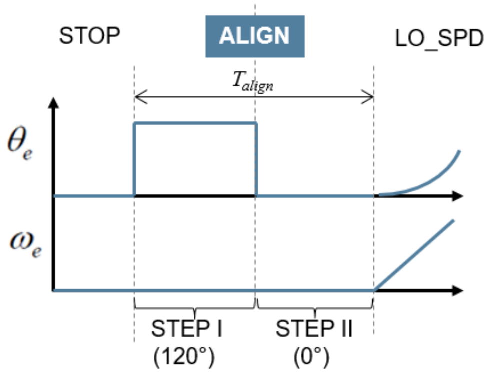


Figure 46. Two-step alignment algorithm

4.4.2.2 Open-loop position and speed generation (M1.EST.LOSPD)

Goal: Provide electrical rotor position and speed at low speeds.

Execution: In fast-loop FL during LO_SPD state of M1.SM.

Details: A constant acceleration is applied to open-loop speed. The open loop position is generated using integrator (see block diagram in Figure 47). The purpose of this algorithm is to accelerate the rotor until a minimal speed ω_{elsthr} is reached when position and speed observer M1.EST.HISPD can be started.

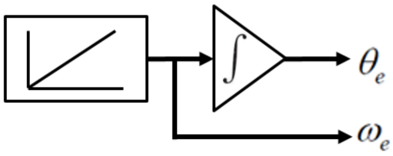


Figure 47. Open-loop position and speed generator

4.4.2.3 Position and speed merging (M1.EST.MISPD)

Goal: Merge M1.EST.LOSPD and M1.EST.HISPD position and speed estimations.

Execution: In fast-loop FL during MI_SPD state of M1.SM.

Details: The example of rotor speed ω_e during startup is shown in Figure 48. Both M1.EST.LOSPD and M1.EST.HISPD position and speed estimators are active during MI_SPD state of M1.SM, however only M1.EST.LOSPD position is used by subsequent algorithms. Once the ω_{emin} is reached, it is assumed that M1.EST.HISPD output is reliable and transition to HI_SPD state immediately occurs. Hence the merging is done by simply switching the used estimator.

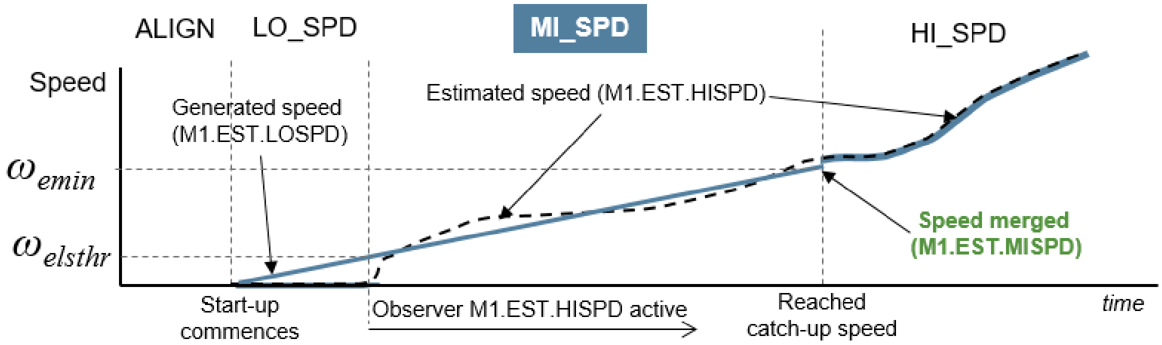


Figure 48. Position and speed merge

4.4.2.4 Position and speed observer (M1.EST.HISPD)

Goal: Provide electrical rotor position and speed at medium- to high-speeds.

Execution: In fast-loop FL during MI_SPD and HI_SPD state of M1.SM.

Details: The observer is implemented using the Advanced Motor-Control Library (AMCLIB) library routines of RTCESL (see documentation at www.nxp.com/rtcesl). Observer is designed in synchronous reference frame, i.e. all observer quantities are DC in steady state making the observer accuracy independent of rotor speed (see Figure 49).

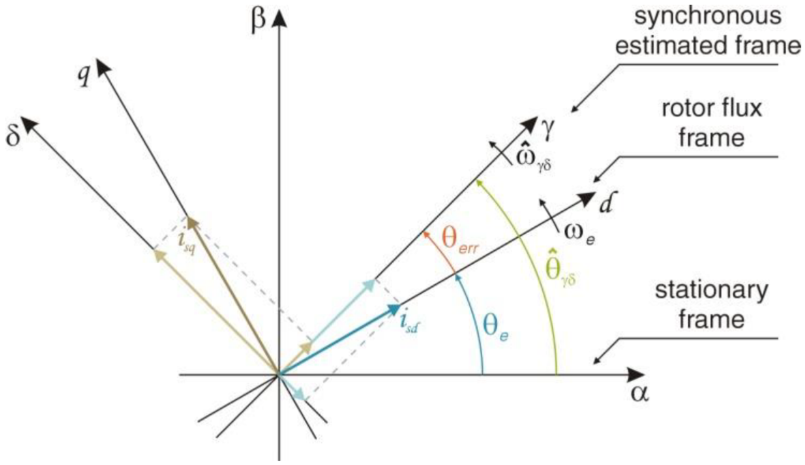
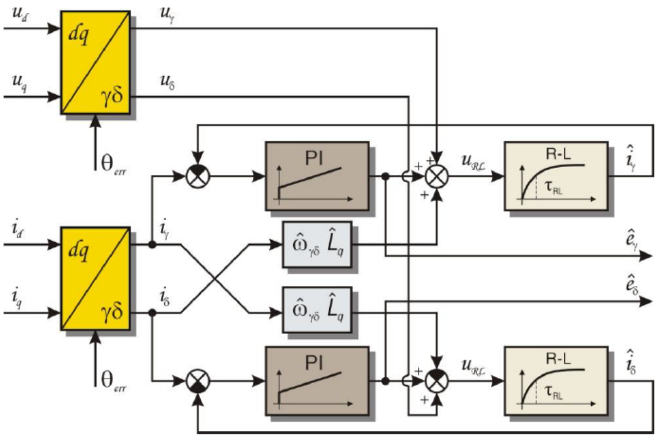


Figure 49. Synchronous rotor frame

The block diagram of BEMF observer is in Figure 50. Because back-EMF term is not modeled, observer actually acts as a back-EMF $e_{\gamma\delta}$ state filter. Saliency based back-EMF voltage is generated due to $L_d \neq L_q$.



$$\begin{bmatrix} u_\gamma \\ u_\delta \end{bmatrix} = \begin{bmatrix} R_s + sL_d & -\hat{\omega}_{\gamma\delta}L_q \\ \hat{\omega}_{\gamma\delta}L_q & R_s + sL_d \end{bmatrix} \begin{bmatrix} i_\gamma \\ i_\delta \end{bmatrix} + E_{\text{sal}} \begin{bmatrix} -\sin(\theta_{err}) \\ \cos(\theta_{err}) \end{bmatrix}$$

Figure 50. BEMF observer block diagram

The rotor position and speed can be obtained using the phase-locked-loop mechanism (see Figure 51).

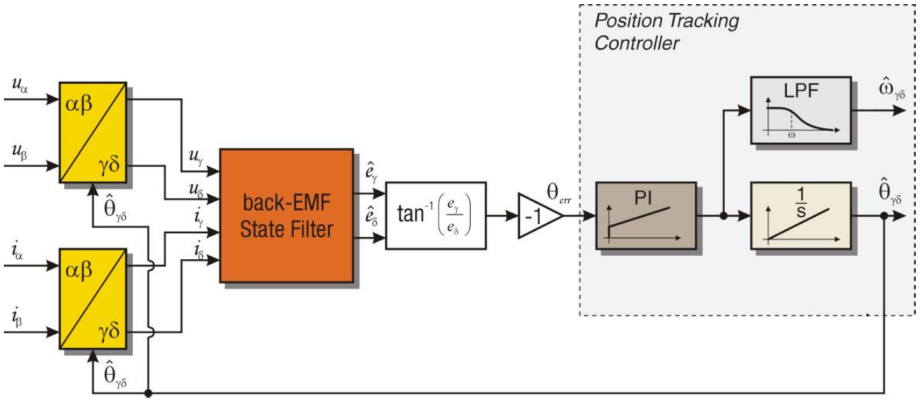


Figure 51. M1.EST.HISPD position and speed observer block diagram

4.4.2.5 Stator resistance estimation (M1.EST.REST)

Goal: Provide stator resistance estimation to support M1.DIAG.RES test.

Execution: In slow-loop SL during HI_SPD of M1.SM.

Details: Estimated stator resistance R can provide information on current plausibility as well as motor winding heating and DC-bus current measurement shunt resistance change. The BEMF-based Model Reference Adaptive System (MRAS) estimator is used (see block diagram in Figure 52). [5] A dead-time compensation algorithm M1.ACT.DTCMOP is necessary, otherwise the estimation will be highly inaccurate. Any other speed dependencies are compensated by the LUT_R resistance bias and LUT_ω speed gain constant compensation. These two tables are necessary to compensate for:

- Observer position estimation error.
- Current measurement distortion due to PWM shifting (see Section 4.4.5.1).
- Current and voltage scale errors

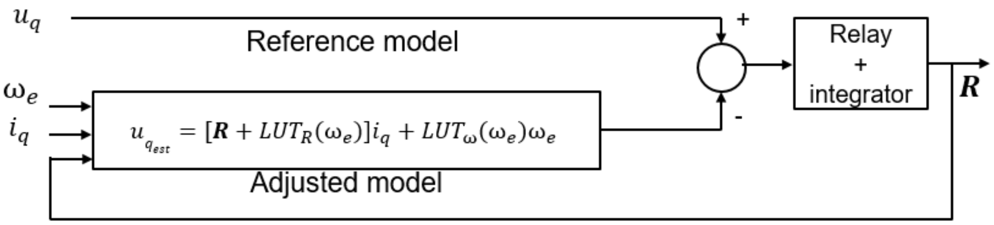

u_q
Reference model
+
Relay
+
integrator
ω_e
i_q
$u_{q_{est}} = [\boldsymbol{R} + LUT_R(\omega_e)]i_q + LUT_\omega(\omega_e)\omega_e$
-
$\boldsymbol{R}$
Adjusted model

Figure 52. Stator resistance estimation algorithm

4.4.3 Diagnostic algorithms

All diagnostic algorithms were described in Section 4.2.

4.4.4 Control algorithms

This section describes notable control algorithms of M1.CTRL module.

4.4.4.1 Idle control mode (M1.CTRL.IDLE)

Goal: Await valid command.

Execution:

- STOP – All M1_PWM_PERIPH outputs are disabled.

Command: none

Position feedback: none

Details: This is the only control mode, during which other control modes can be selected (M1.CTRL.SPEED_CL, M1.CTRL.CURR_CL,…). Once the valid command (for given selected control mode M1.CTRL.SPEED_CL, M1.CTRL.CURR_CL, …) is received, the *M1SM_RequestStart* request is generated for M1.SM. This control mode is also eventually entered when any other control mode is changed during runtime.

4.4.4.2 Closed-loop speed control mode (M1.CTRL.SPEED_CL)

Goal: Control rotor speed.

Execution:

- ALIGN – Voltage controller M1.CTRL.VOLT in fast loop FL.
- LO_SPD, MI_SPD, HI_SPD - Current controller M1.CTRL.CURR in fast loop FL.
- HI_SPD – Speed controller M1.CTRL.SPD in slow loop SL.

Command: Required rotor speed from external *pwm_in_mcu* signal (APP.EXTCMD driver) or from FreeMASTER.

Position feedback: Estimated position and speed from M1.SM is used (closed-loop mode).

Details: The algorithm block diagram is shown in Figure 53. Direct and quadrature axis currents are controlled separately by two PI controller (M1.CTRL.CURR), where direct axis current is kept zero and quadrature axis current setpoint is given by speed controller (M1.CTRL.SPD). The speed command *APP CMD Speed FOC* is filtered by ramp algorithm. Default option used during normal operation. If invalid command or control mode change is received by M1.CTRL when this control mode is actively running, the rotor speed is slowed to a minimal speed and the *M1SM_RequestStop* request is generated for M1.SM.

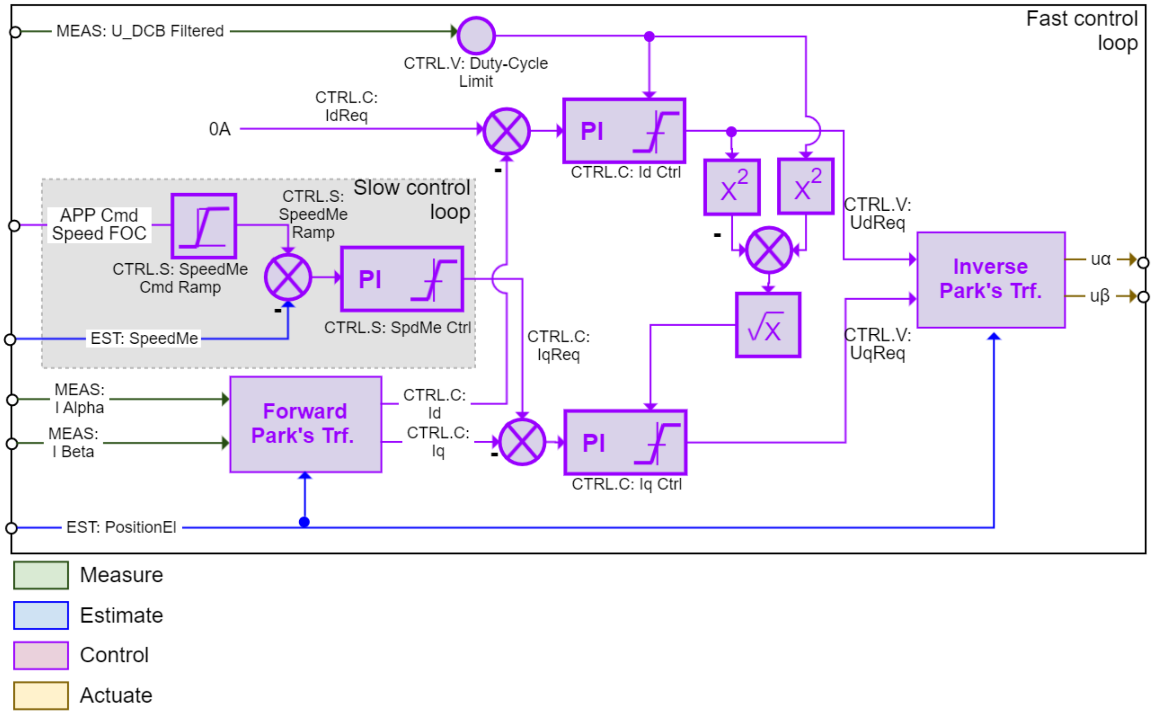


Figure 53. Speed FOC control mode (M1.CTRL.SPEED_CL)

4.4.4.3 Closed-loop current control mode (M1.CTRL.CURR_CL)

Goal: Control stator current (rotor torque).

Execution:

- ALIGN – Voltage controller M1.CTRL.VOLT in fast loop FL.
- LO_SPD, MI_SPD, HI_SPD - Current controller M1.CTRL.CURR in fast loop FL.

Commands: Required *dq*-axis currents from FreeMASTER (*APP Cmd Curr Id*, *APP Cmd Curr Iq*)

Position feedback: Estimated position and speed from M1.SM is used (closed-loop mode).

Details: The algorithm block diagram is shown in Figure 54. Direct and quadrature axis currents are controlled separately by two PI controller (M1.CTRL.CURR), where *APP Cmd Curr Id* and *APP Cmd Curr Iq* setpoints are directly set by FMSTR. Unlike in the case of speed closed loop FOC, the speed controller is not engaged in HI_SPD state (basically torque control). This torque-control mode is present to help with algorithm tuning process. If invalid command or control mode change is received by M1.CTRL when this control mode is actively running, the *M1SM_RequestStop* request is generated for M1.SM.

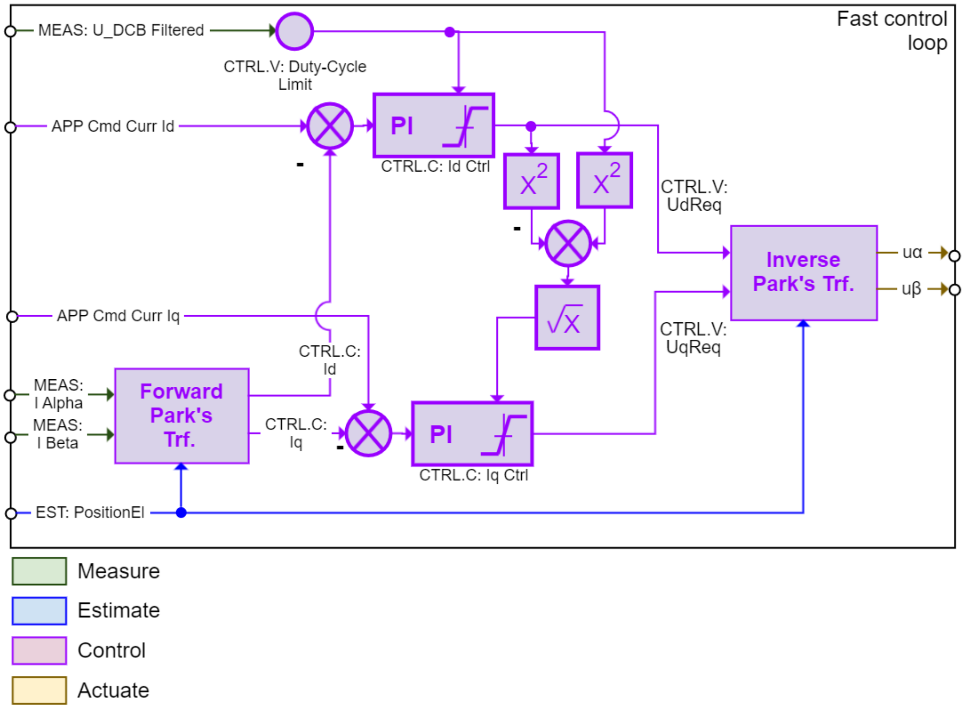


Figure 54. Closed-loop current control mode (M1.CTRL.CURR_CL)

4.4.4.4 Open-loop Current Control Mode (M1.CTRL.CURR_OL)

Goal: Control stator current vector.

Execution:

- ALIGN, LO_SPD, MI_SPD, HI_SPD - Current controller M1.CTRL.OL in fast loop FL.

Commands: Required *dq*-axis currents (*APP Cmd Curr Id, APP Cmd Curr Iq*), current vector position (*APP Cmd PosEl*) and frequency (*APP Cmd Freq*) from FreeMASTER (FMSTR).

Position feedback: Position for inverse Park is generated from local integrator (open-loop mode).

Details: The algorithm block diagram is shown in Figure 55. Direct and quadrature axis currents are controlled separately by two PI controller (M1.CTRL.OL), where *APP Cmd Curr Id* and *APP Cmd Curr Iq* setpoints are directly set by FMSTR. The current vector frequency command *APP CMD Freq* is filtered by ramp algorithm. The control mode is useful for current controller tuning. If invalid command or control mode change is received by M1.CTRL when this control mode is actively running, the *M1SM_RequestStop* request is generated for M1.SM.

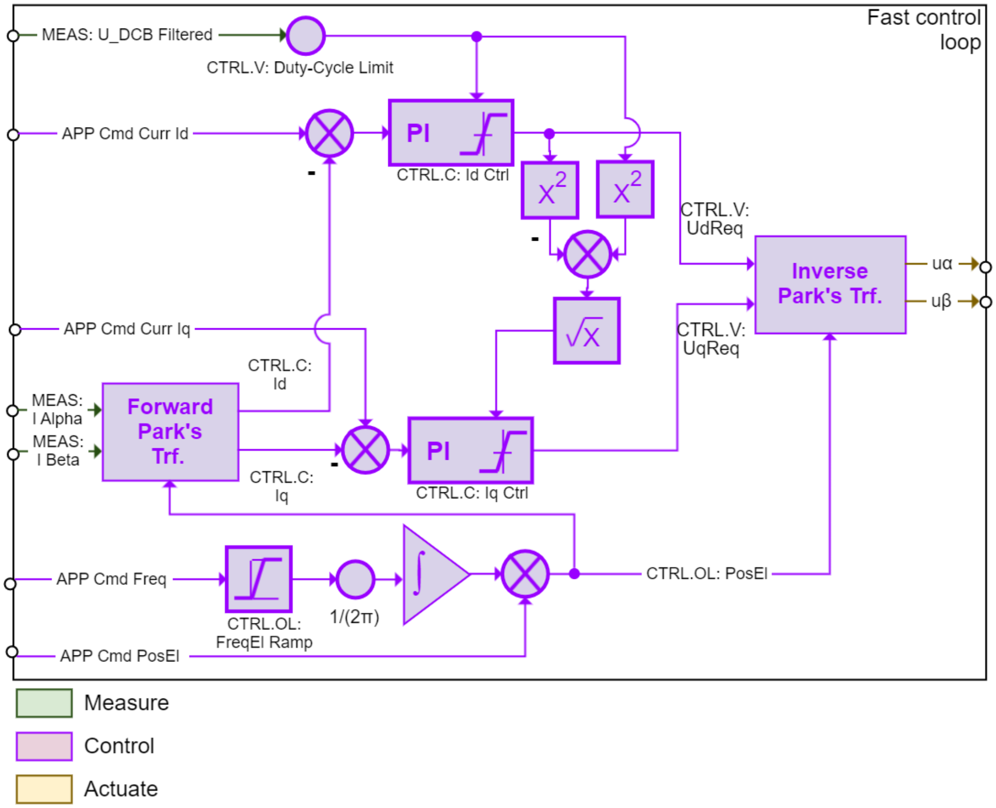


Figure 55. Open-loop current control mode (M1.CTRL.CURR_OL)

4.4.4.5 Scalar control mode (M1.CTRL.SCALAR_OL)

Goal: Control stator voltage frequency and amplitude.

Execution:

- ALIGN, LO_SPD, MI_SPD, HI_SPD -Voltage controller M1.CTRL.OL in fast loop FL.

Commands: Required frequency from FreeMASTER (*APP Cmd Freq*).

Position feedback: Position for inverse Park is generated from local integrator (open-loop mode).

Details: The algorithm block diagram is shown in Figure 56. The stator voltage amplitude is proportional to the required voltage vector frequency. The voltage vector frequency command *APP CMD Freq* is filtered by ramp algorithm. Useful for estimator tuning because the position and speed from M1.SM is still available (the M1.EST.HISPD is still running). If invalid command or control mode change is received by M1.CTRL when this control mode is actively running, the *M1SM_RequestStop* request is generated for M1.SM.

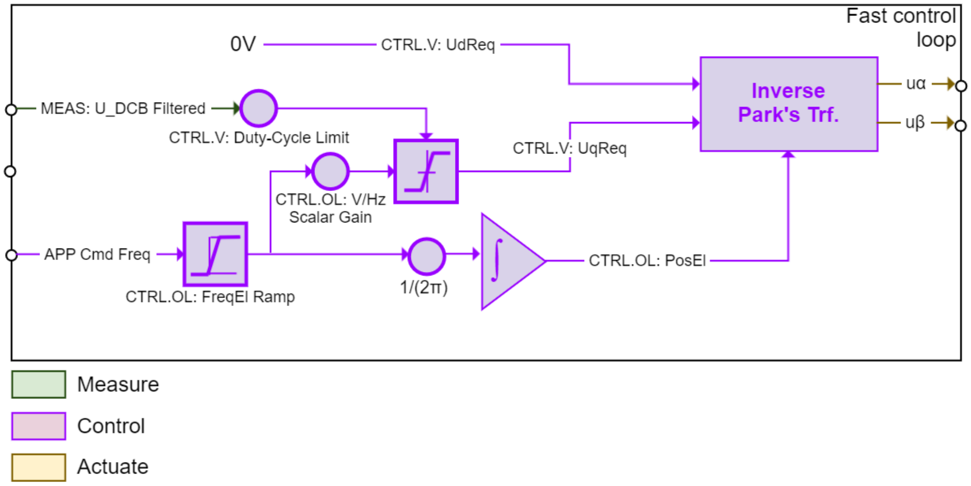


Figure 56. Scalar control mode (M1.CTRL.SCALAR_OL)

4.4.4.6 Closed-loop voltage control mode (M1.CTRL.VOLT_OL)

Goal: Control stator voltage vector.

Execution:

- ALIGN, LO_SPD, MI_SPD, HI_SPD -Voltage controller M1.CTRL.OL in fast loop FL.

Commands: Required *dq*-axis voltages (*APP Cmd Volt Ud*, *APP Cmd Volt Uq*), current vector position (*APP Cmd PosEl*) and frequency (*APP Cmd Freq*) from FreeMASTER (FMSTR).

Position feedback: Position for inverse Park is generated from local integrator (open-loop mode).

Details: The algorithm block diagram is shown in Figure 57. The voltage vector frequency command *APP CMD Freq* is filtered by ramp algorithm. Useful for HW debugging and tuning. If invalid command or control mode change is received by M1.CTRL when this control mode is actively running, the *M1SM_RequestStop* request is generated for M1.SM.

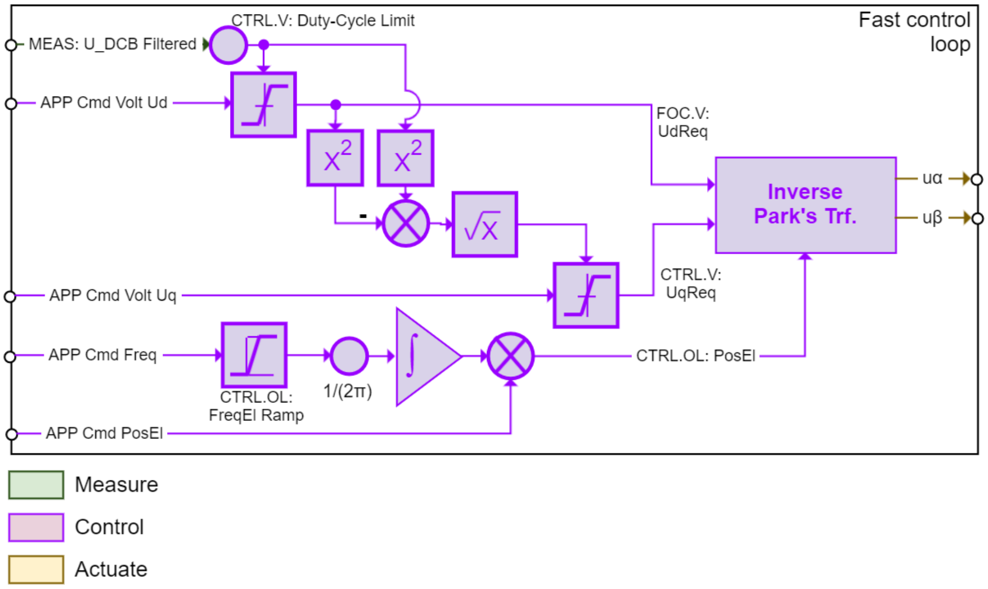


Figure 57. Open-loop voltage control mode (M1.CTRL.VOLT_OL)

4.4.5 Actuator algorithms

This section describes notable actuation algorithms of M1.SM and M1.PWM modules.

4.4.5.1 Space vector modulation (M1.ACT.SVM)

Goal: Generate updates for M1_DMA_TAB_DLY table and duty cycles and shifts for M1.PWM driver so single-shunt current reconstruction is possible.

Execution: In fast-loop FL during ALIGN, LO_SPD, MI_SPD, and HI_SPD of M1.SM.

Details: The SVM algorithm is based on Advanced Motor-Control Library (AMCLIB) library routine of RTCESL (see documentation at www.nxp.com/rtcesl), with the modification of providing additional outputs (PWM shifts and *idcb_rc* sample locations). The reason why a modified algorithm is necessary is the fact that *idcb_rc* samples cannot be taken when:

1. **Voltage vector is crossing SVM sector border.** Only one sample can be taken then (see Figure 59-1).
2. **Modulation index is low.** Sampling intervals are too short and none of current samples can be taken (see Figure 59-2).

There are many solutions to these problems available, but for this project the shifted-PWM method was used. It is based on modification (shifting) of the PWM ON/OFF times, while preserving duty cycles (applied stator voltage is the same).

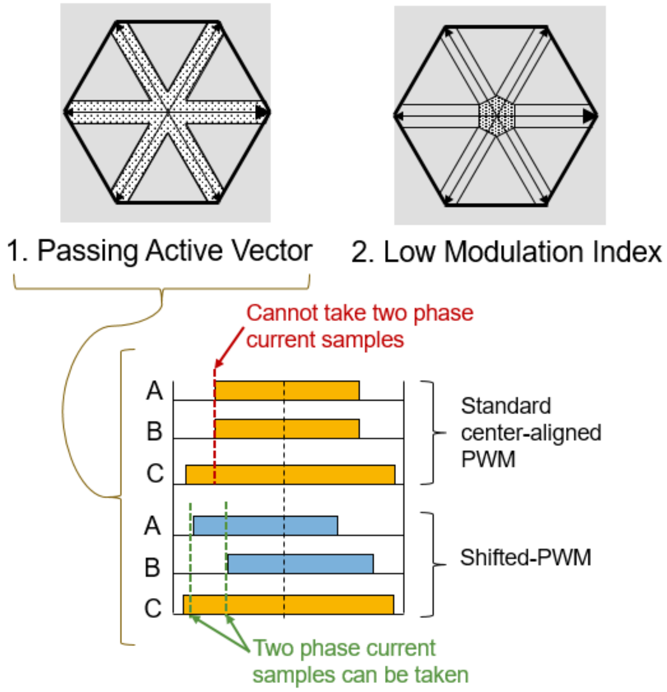

1. Passing Active Vector
2. Low Modulation Index
Cannot take two phase current samples
A
B
C
Standard center-aligned PWM
A
B
C
Shifted-PWM
Two phase current samples can be taken

Figure 58. Need for shifted PWM

Different shifting strategy is applied for both critical cases:

1. **Passing Active Vector:** See Figure 59-top for example of PWM shifting for this case. Generally, following steps are followed:
   - Freeze center edge
   - Move one critical edge
   - Used for higher modulation indexes
2. **Low modulation Indexes:** See Figure 59-bottom for example of PWM shifting for this case. Generally, following steps are followed:
   - Freeze center edge
   - Move both side edges in opposite direction
   - Used low modulation indexes

The right method is selected within M1.ACT.SVM algorithm and shifts are applied by M1.PWM driver.

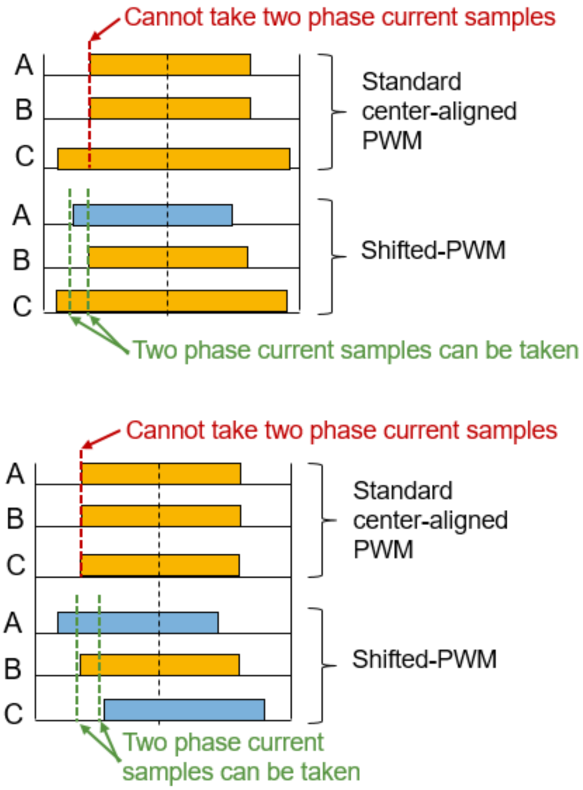

Cannot take two phase current samples
A
B
C
Standard center-aligned PWM
A
B
C
Shifted-PWM
Two phase current samples can be taken
Cannot take two phase current samples
A
B
C
Standard center-aligned PWM
A
B
C
Shifted-PWM
Two phase current samples can be taken

Figure 59. Cases for PWM shifting

4.4.5.2 DC-bus ripple compensation (M1.ACT.DCBCOMP)

Goal: Compensate required stator voltage for DC-bus voltage changes.

Execution: In fast-loop FL during ALIGN, LO_SPD, MI_SPD, and HI_SPD of M1.SM.

Details: The DC-bus compensation algorithm is implemented using the Advanced Motor-Control Library (AMCLIB) library routines of RTCESL (see documentation at www.nxp.com/rtcesl).

4.4.5.3 Dead-time compensation (M1.ACT.DTCOMP)

Goal: Compensate required stator voltage for inverter non-linearities.

Execution: In fast-loop FL during ALIGN, LO_SPD, MI_SPD, and HI_SPD of M1.SM.

Details: Each inverter introduces the total error voltage U_{error}, which is caused by the dead-time, current clamping effect, and transistor voltage drop. The actual inverter output voltage is, therefore, lower than the voltage required by the U_{error}. The error voltage amplitude U_{error} depends on the actual phase current I_{PH}. The example of the inverter error characteristic is shown in Figure 60, it can be seen, that it is not linear. The look-up table (LUT) compensation algorithm, which adds the U_{error} voltage to $U_{\alpha\beta}$ voltage vector is used (see block diagram in Figure 61).

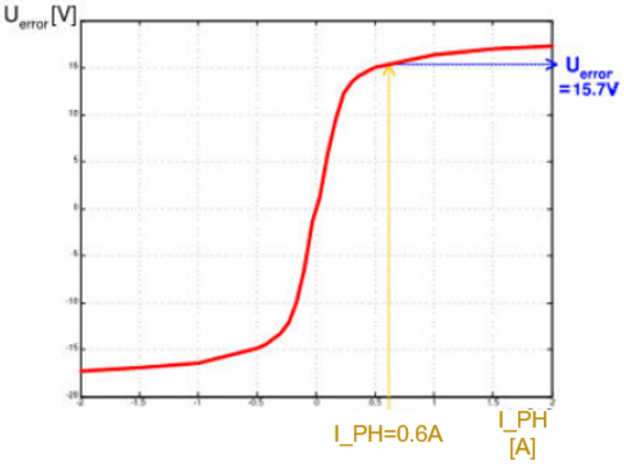


Figure 60. Dead-time voltage example

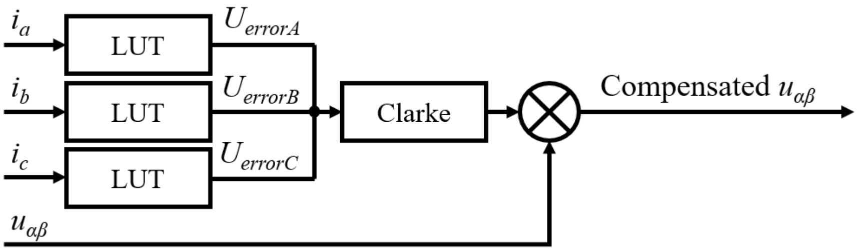


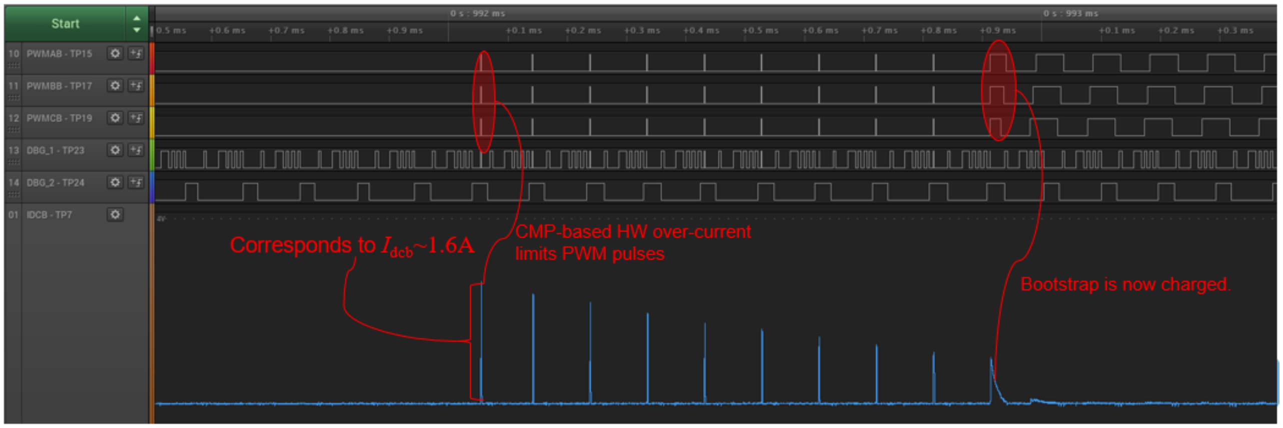
Figure 61. Dead-time compensation algorithm block diagram

4.4.5.4 PWM startup algorithm (M1.PWM.START)

Goal: Perform the IPM boot-strap circuit charging.

Execution: In fast-loop FL at the beginning of ALIGN state of M1.SM.

Details: The PWM enablement requires charging of the bootstrap circuit. This results in spikes on the DC-bus current *idcb_rc* (see example in Figure 62), which would, however, normally cause over-current condition. To perform safe PWM start even in case of actual over-current fault, the M1_PWM_PERIPH periphery is temporarily configured into *automatic fault* clearing mode and the M1_CMP_OC_PERIPH based HW over-current fault protection feature then disables the PWM during the first *idcb_rc* spikes (see algorithm flowchart in Figure 63).



Corresponds to I_{dcb}~1.6A

CMP-based HW over-current limits PWM pulses

Bootstrap is now charged.

Figure 62. PWM startup sequence example

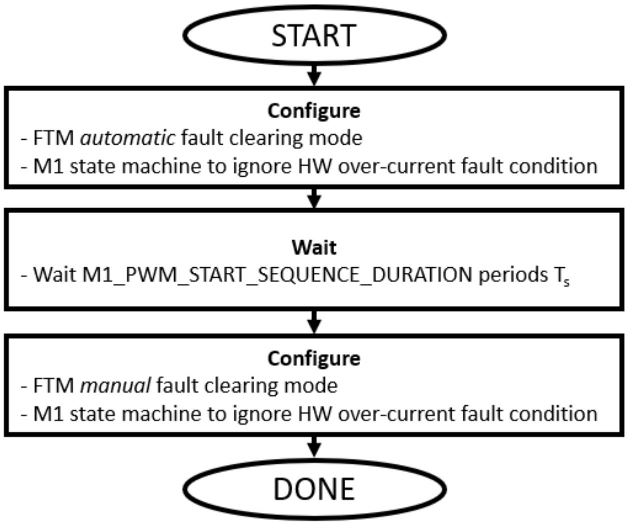

START
Configure
- FTM *automatic* fault clearing mode
- M1 state machine to ignore HW over-current fault condition
Wait
- Wait M1_PWM_START_SEQUENCE_DURATION periods T_s
Configure
- FTM *manual* fault clearing mode
- M1 state machine to ignore HW over-current fault condition
DONE

Figure 63. PWM startup sequence flowchart

4.4.6 Application algorithms (APP)

This section describes notable application algorithms of APP and APP.EXTCMD modules.

4.4.6.1 External command measurement (APP.EXTCMD)

Goal: Get the pump speed command from isolated *pwm_in_mcu* input signal.

Execution: In the fast-loop FL and slow-loop SL as part of APP application tasks.

Details: The speed command (passed as *APP CMD Speed FOC* to M1.CTRL.SPEED_CL algorithm in BG) is encoded in the *pwm_in_mcu* signal via duty cycle D (see conversion chart example in Figure 64). The signal is only valid when its frequency f_{ctrl} is in range 200 Hz to 2 kHz. The APP_EXTCMD_PERIPH FlexTimer periphery with double-capture feature is used to determine PWM signal frequency and duty cycle D. Only measured duty cycle in range $D_{min} < D < D_{max}$ is accepted and it corresponds to $\omega_{ereqmin} < APP\ CMD\ Speed\ FOC < \omega_{ereqmax}$. A hysteresis is added near the corner values D_{min}

and D_{max} to prevent repeated unwanted *M1SM_RequestStart*/*M1SM_RequestStop* requests. A parallel safety mechanism is implemented via PORT raising edge rate reading to perform plausibility check (see M1.DIAG.EXTCMD in Section 4.1.9).

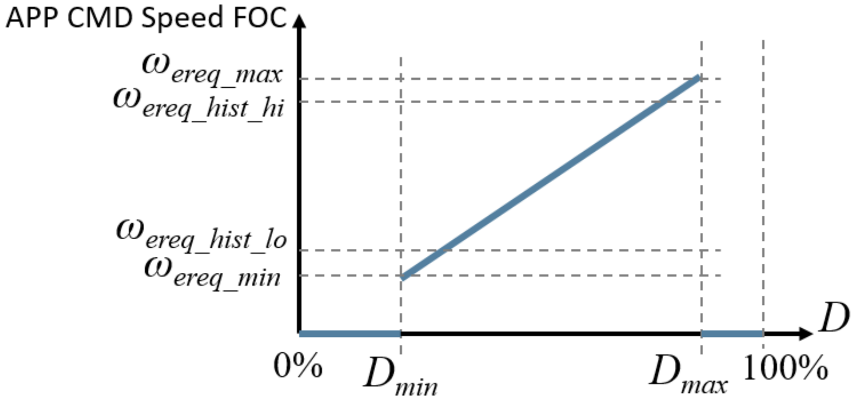


Figure 64. The APP.EXTCMD speed command conversion

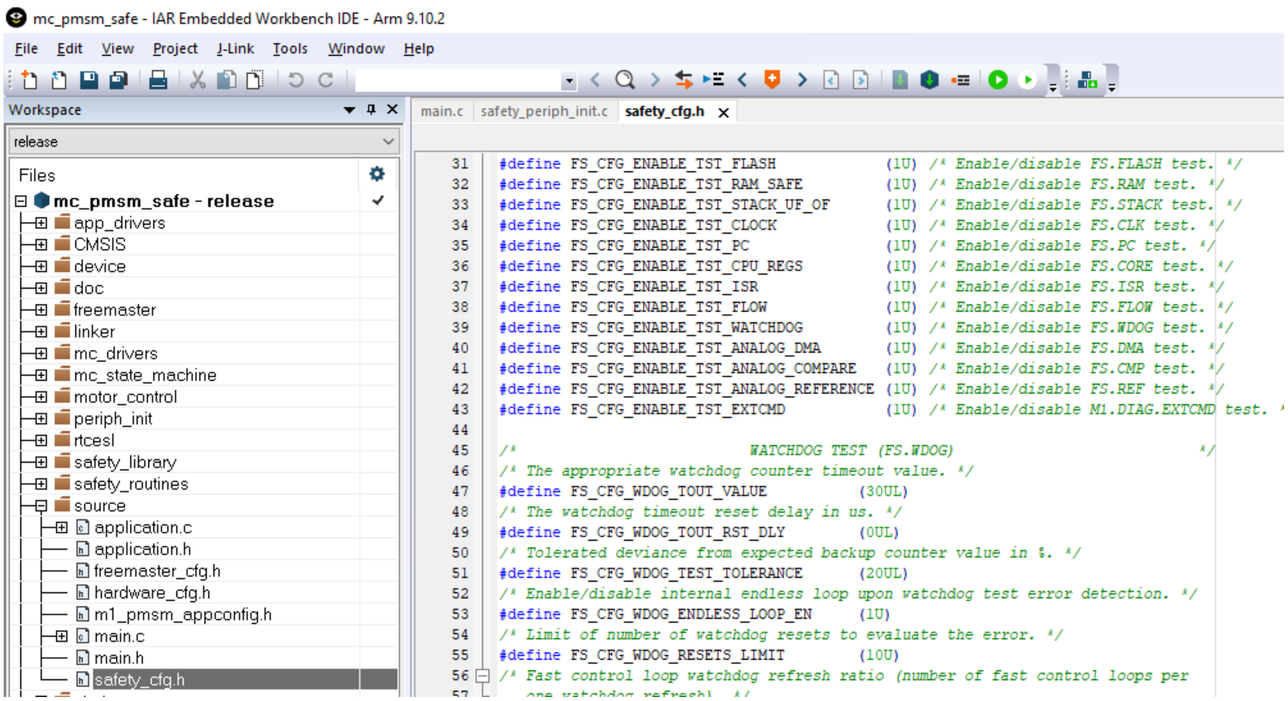
Chapter 5
Build and run application

This chapter provides a guide on how to build and run the *mc_pmsm_safe* example application using the supported IDEs. The *mc_pmsm_safe* example application was verified for the IAR Embedded Workbench IDE v9.10.2 and the MCUXpresso v11.4.0.

5.1 IAR Embedded Workbench IDE

When using IAR IDE, please follow these steps:

- Ensure that you done steps in chapter 2.3.
- Open IAR project file *mc_pmsm_safe.eww* in *\pack_pmsm_safe_hvpmc3phlite\iar\*
- Check the required safety tests settings in *source/safety_cfg.h*. It is recommended to disable flash test during debugging as the SW breakpoints corrupt the CRC calculation. Also it is recommended to disable clock test during debugging as LPTMR counter is still running. If you are not using external command PWM input, please disable M1.DIAG.EXTCMD test (using FS_CFG_ENABLE_TST_EXTCMD), otherwise the application will neter a fault state.



```
#define FS_CFG_ENABLE_TST_FLASH            (1U) /* Enable/disable FS.FLASH test. */
#define FS_CFG_ENABLE_TST_RAM_SAFE         (1U) /* Enable/disable FS.RAM test. */
#define FS_CFG_ENABLE_TST_STACK_UF_OF      (1U) /* Enable/disable FS.STACK test. */
#define FS_CFG_ENABLE_TST_CLOCK            (1U) /* Enable/disable FS.CLK test. */
#define FS_CFG_ENABLE_TST_PC               (1U) /* Enable/disable FS.PC test. */
#define FS_CFG_ENABLE_TST_CPU_REGS         (1U) /* Enable/disable FS.CORE test. */
#define FS_CFG_ENABLE_TST_ISR              (1U) /* Enable/disable FS.ISR test. */
#define FS_CFG_ENABLE_TST_FLOW             (1U) /* Enable/disable FS.FLOW test. */
#define FS_CFG_ENABLE_TST_WATCHDOG         (1U) /* Enable/disable FS.WDOG test. */
#define FS_CFG_ENABLE_TST_ANALOG_DMA       (1U) /* Enable/disable FS.DMA test. */
#define FS_CFG_ENABLE_TST_ANALOG_COMPARE   (1U) /* Enable/disable FS.CMP test. */
#define FS_CFG_ENABLE_TST_ANALOG_REFERENCE (1U) /* Enable/disable FS.REF test. */
#define FS_CFG_ENABLE_TST_EXTCMD           (1U) /* Enable/disable M1.DIAG.EXTCMD test. *

/*                          WATCHDOG TEST (FS.WDOG)                          */
/* The appropriate watchdog counter timeout value. */
#define FS_CFG_WDOG_TOUT_VALUE          (30UL)
/* The watchdog timeout reset delay in us. */
#define FS_CFG_WDOG_TOUT_RST_DLY        (0UL)
/* Tolerated deviance from expected backup counter value in %. */
#define FS_CFG_WDOG_TEST_TOLERANCE      (20UL)
/* Enable/disable internal endless loop upon watchdog test error detection. */
#define FS_CFG_WDOG_ENDLESS_LOOP_EN     (1U)
/* Limit of number of watchdog resets to evaluate the error. */
#define FS_CFG_WDOG_RESETS_LIMIT        (10U)
/* Fast control loop watchdog refresh ratio (number of fast control loops per
   one watchdog refresh). */
```

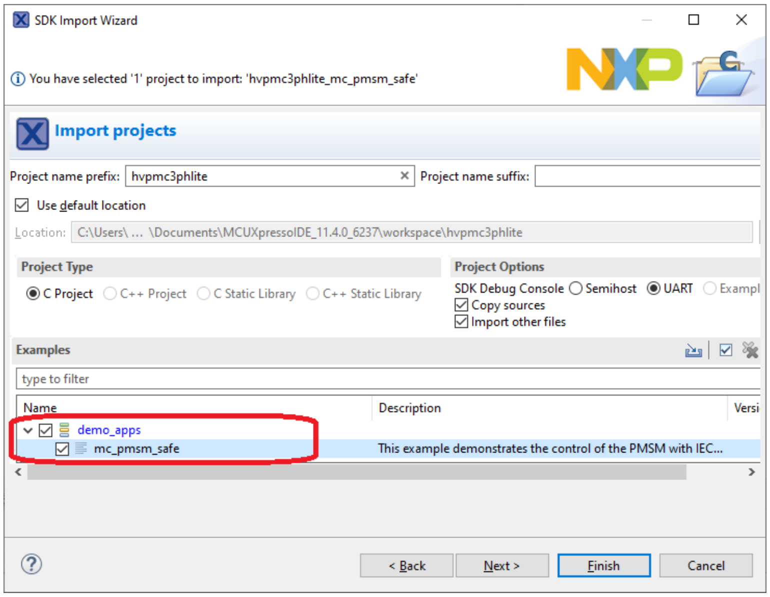
Figure 65. The safety test configuration file

- Click to download and debug button, or press “Ctrl + D”.
- After code is built and downloaded run the program in debugger by clicking to go button.
- If you don't want debug code, stop debugging by clicking stop button, od press “Ctrl+Shift+D”. Then reset the board. (Unplug and plug power source)
- Open *pmsm_safe.pmp* FreeMASTER project file and establish FreeMASTER communication according to following sections.

5.2 MCUXpresso IDE

When using MCUXpresso, please ensure that you done steps in Chapter 2.3, open MCUXpresso IDE, and switch to the main IDE. You will then be able to import the example by completing these steps:

- Plug *pack_pmsm_safe_hvpmc3phlite* package in Windows explorer and drop it in MCUXpresso Installed SDKs tab.
- Click to *Import SDK example(s)…* button located in left bottom corner.
- In SDK Import Wizard select *hvpmc3phlite* and click to *Next* button.
- Then select *demo_apps* and *mc_pmsm_safe* and click to *Finish* button.



SDK Import Wizard

You have selected '1' project to import: 'hvpmc3phlite_mc_pmsm_safe'

Import projects

Project name prefix: hvpmc3phlite   Project name suffix:

☑ Use default location

Location: C:\Users\ … \Documents\MCUXpressoIDE_11.4.0_6237\workspace\hvpmc3phlite

Project Type: ● C Project ○ C++ Project ○ C Static Library ○ C++ Static Library

Project Options: SDK Debug Console ○ Semihost ● UART ○ Exampl
☑ Copy sources
☑ Import other files

Examples

type to filter

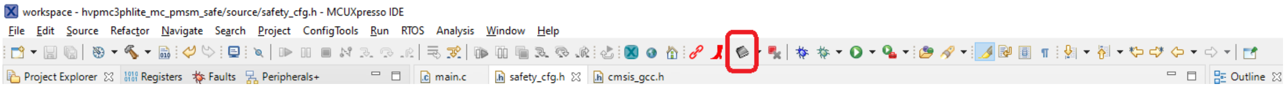
Name	Description	Versi
☑ demo_apps		
☑ mc_pmsm_safe	This example demonstrates the control of the PMSM with IEC…	

Figure 66. MCUXpresso IDE – SDK Import Wizard

- If the project is successfully imported, it will appear in Project Explorer.

Once the example was imported, you can build and run the application. To do so, please follow this guide:

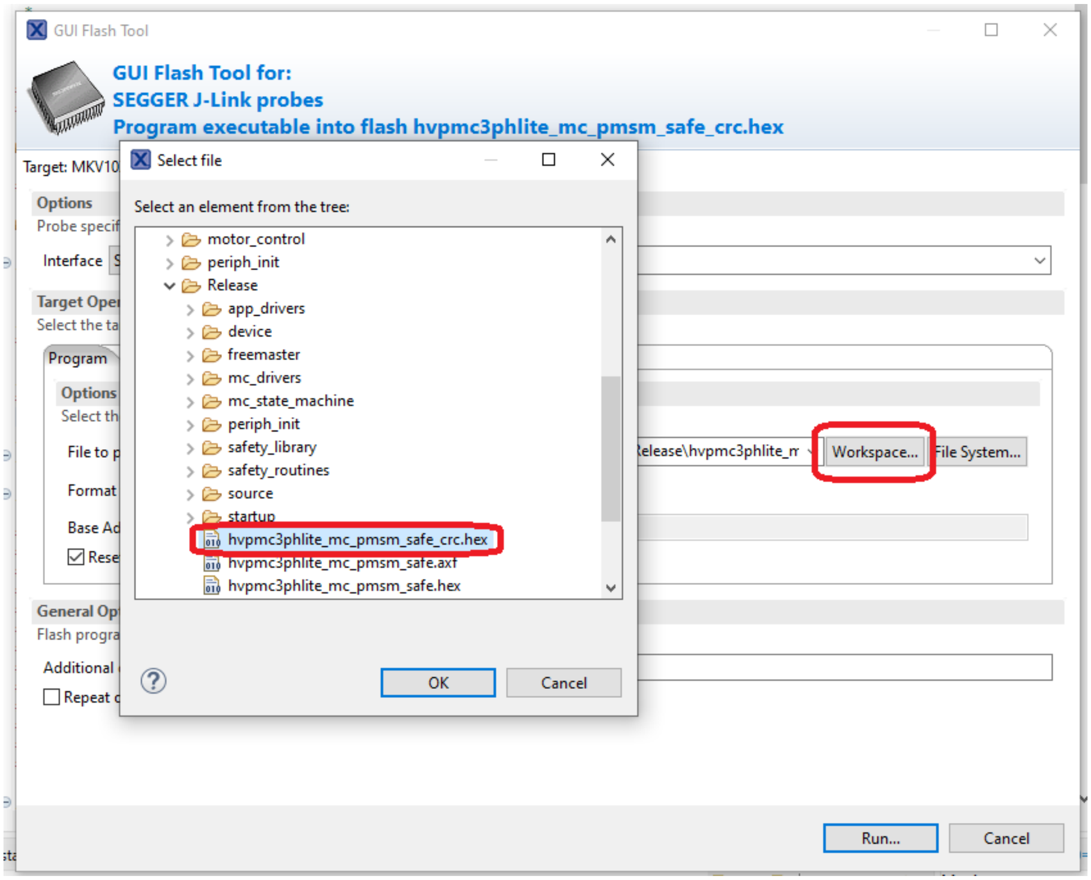
- Select build configuration to *release*, click to *project -> build configuration -> Set Active -> Release.*
- Check the required safety tests settings in *source/safety_cfg.h*. It is recommended to disable flash test during debugging as the SW breakpoints corrupt the CRC calculation. Also it is recommended to disable clock test during debugging because LPTMR counter is still running while in debug. If you are not using external command PWM input, please disable M1.DIAG.EXTCMD test (using FS_CFG_ENABLE_TST_EXTCMD), otherwise the application will enter a fault state.
- Click to *build* project button in left bottom corner.
- If the Flash test is disabled in *source/safety_cfg.h*, click to Debug button.
- If the Flash safety test is enabled, you have to download modified *hvpmc3phlite_mc_pmsm_safe* file with postfix "*_crc"* to the target (*hvpmc3phlite_mc_pmsm_safe_crc.hex*). Please, click to GUI Flash Tool to do so (see Figure 67). Then click the Workspace button and find *hvpmc3phlite_mc_pmsm_safe_crc.hex* in output folder (see Figure 68).



workspace - hvpmc3phlite_mc_pmsm_safe/source/safety_cfg.h - MCUXpresso IDE

File Edit Source Refactor Navigate Search Project ConfigTools Run RTOS Analysis Window Help
Project Explorer
Registers
Faults
Peripherals+
main.c
safety_cfg.h
cmsis_gcc.h
Outline

Figure 67. GUI Flash Tool



GUI Flash Tool

GUI Flash Tool for:
SEGGER J-Link probes
Program executable into flash hvpmc3phlite_mc_pmsm_safe_crc.hex

Select file

Select an element from the tree:

- motor_control
- periph_init
- Release
  - app_drivers
  - device
  - freemaster
  - mc_drivers
  - mc_state_machine
  - periph_init
  - safety_library
  - safety_routines
  - source
  - startup
  - hvpmc3phlite_mc_pmsm_safe_crc.hex
  - hvpmc3phlite_mc_pmsm_safe.axf
  - hvpmc3phlite_mc_pmsm_safe.hex

Figure 68. Modified hvpmc3phlite_mc_pmsm_safe.hex file

- Click the *run* button and wait while the code is being downloaded. Then reset the board.

For more information about importing projects to the MCUXpresso IDE, see the MCUXpresso IDE - Importing MCUXpresso SDK video (https://www.nxp.com/video/mcuxpresso-ide-importing-mcuxpresso-sdk:MCUXPRESSO-IDE-IMPORTING-SDK).

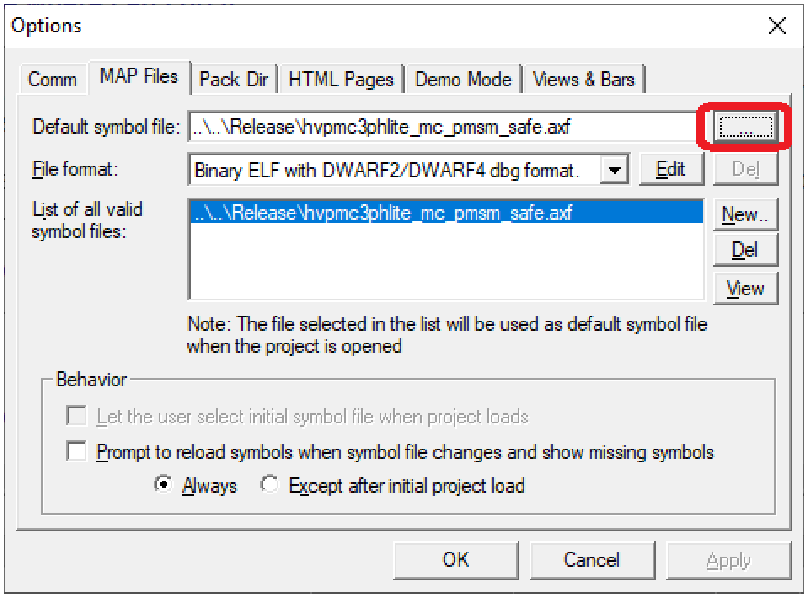
Chapter 6
Remote control using FreeMASTER

This section provides information about the tools and recommended procedures to control the PMSM 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 FreeMASTER 3.0 at www.nxp.com/freemaster. To run the FreeMASTER application, double-click the *pmsm_safe.pmp* file located directly in the *pack_pmsm_safe_hvpmc3phlite* package.

6.1 Establishing FreeMASTER communication

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

1. Download the project from your chosen IDE to the MCU and run it (see Section 5).
2. Open the FreeMASTER file *pmsm_safe.pmp*.
3. Insert right symbol file (located in output folder). Click to *Project -> Options -> MAP Files* and chose right output file (the file will differ depending on the IDE).



Default symbol file: ..\..\Release\hvpmc3phlite_mc_pmsm_safe.axf

File format: Binary ELF with DWARF2/DWARF4 dbg format.

List of all valid symbol files: ..\..\Release\hvpmc3phlite_mc_pmsm_safe.axf

Note: The file selected in the list will be used as default symbol file when the project is opened

Behavior

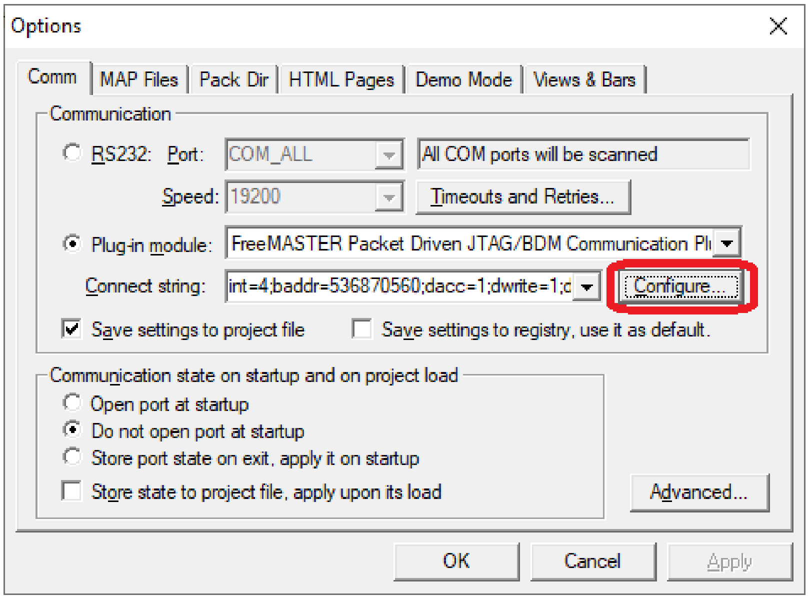
□ Let the user select initial symbol file when project loads

□ Prompt to reload symbols when symbol file changes and show missing symbols

● Always ○ Except after initial project load

Figure 69. Default symbol file in FreeMASTER Project - > Options -> MAP Files

4. Go to *Project -> Options -> Comm*, click to *Configure* button and then to *Search Address Now* button



Options

Comm | MAP Files | Pack Dir | HTML Pages | Demo Mode | Views & Bars

Communication

○ RS232: Port: COM_ALL — All COM ports will be scanned

Speed: 19200 — Timeouts and Retries...

● Plug-in module: FreeMASTER Packet Driven JTAG/BDM Communication Pl

Connect string: int=4;baddr=536870560;dacc=1;dwrite=1;c — Configure...

☑ Save settings to project file ☐ Save settings to registry, use it as default.

Communication state on startup and on project load

○ Open port at startup

● Do not open port at startup

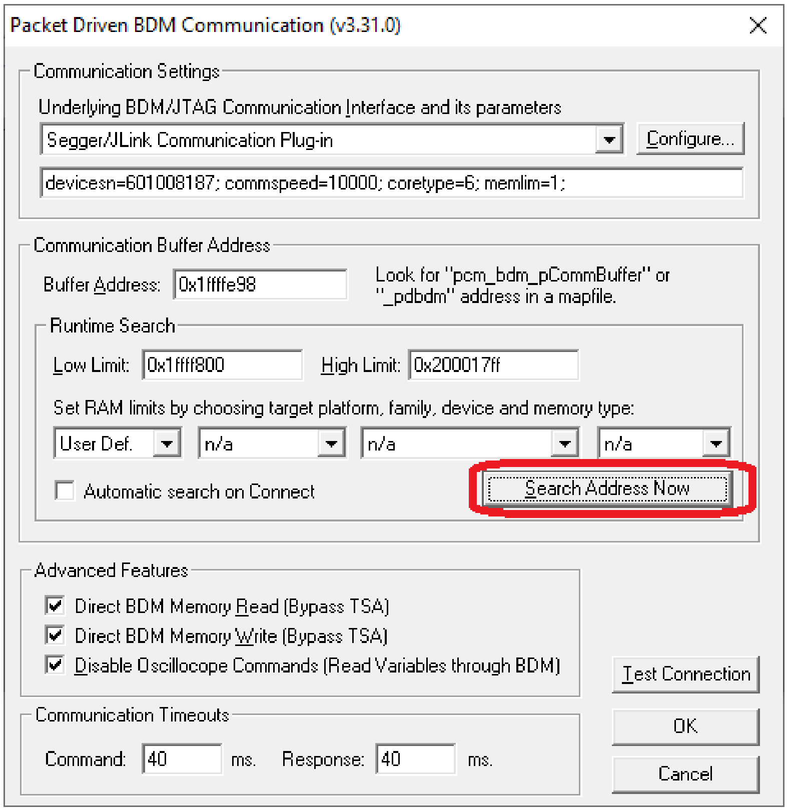
○ Store port state on exit, apply it on startup

☐ Store state to project file, apply upon its load

Advanced...

OK | Cancel | Apply

Figure 70. Configure button in FreeMASTER Project - > Options -> Comm



Packet Driven BDM Communication (v3.31.0)

Communication Settings

Underlying BDM/JTAG Communication Interface and its parameters

Segger/JLink Communication Plug-in — Configure...

devicesn=601008187; commspeed=10000; coretype=6; memlim=1;

Communication Buffer Address

Buffer Address: 0x1ffffe98 — Look for "pcm_bdm_pCommBuffer" or "_pdbdm" address in a mapfile.

Runtime Search

Low Limit: 0x1ffff800 High Limit: 0x200017ff

Set RAM limits by choosing target platform, family, device and memory type:

User Def. | n/a | n/a | n/a

☐ Automatic search on Connect — Search Address Now

Advanced Features

☑ Direct BDM Memory Read (Bypass TSA)

☑ Direct BDM Memory Write (Bypass TSA)

☑ Disable Oscillocope Commands (Read Variables through BDM)

Test Connection

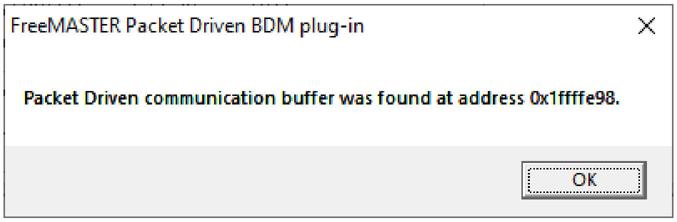
Communication Timeouts

Command: 40 ms. Response: 40 ms.

OK | Cancel

Figure 71. Search Address Now button in FreeMASTER Packet Driven Communication window

5. If the communication buffer address has been found, the following windows will appear



FreeMASTER Packet Driven BDM plug-in

Packet Driven communication buffer was found at address 0x1ffffe98.

Figure 72. Address has been successfully found

6. Then you can click to Test Connection button. Connection should be verified. Click to OK and close settings windows (OK).
7. Start communication by clicking to communication GO button or press “Ctrl + G”

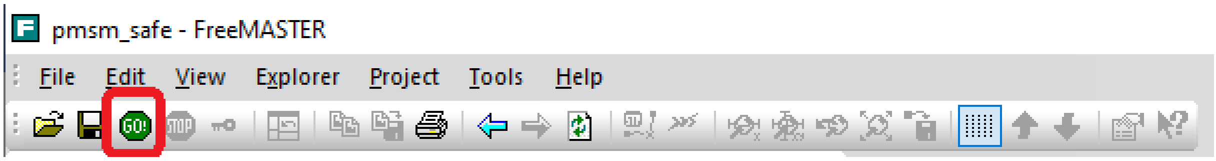
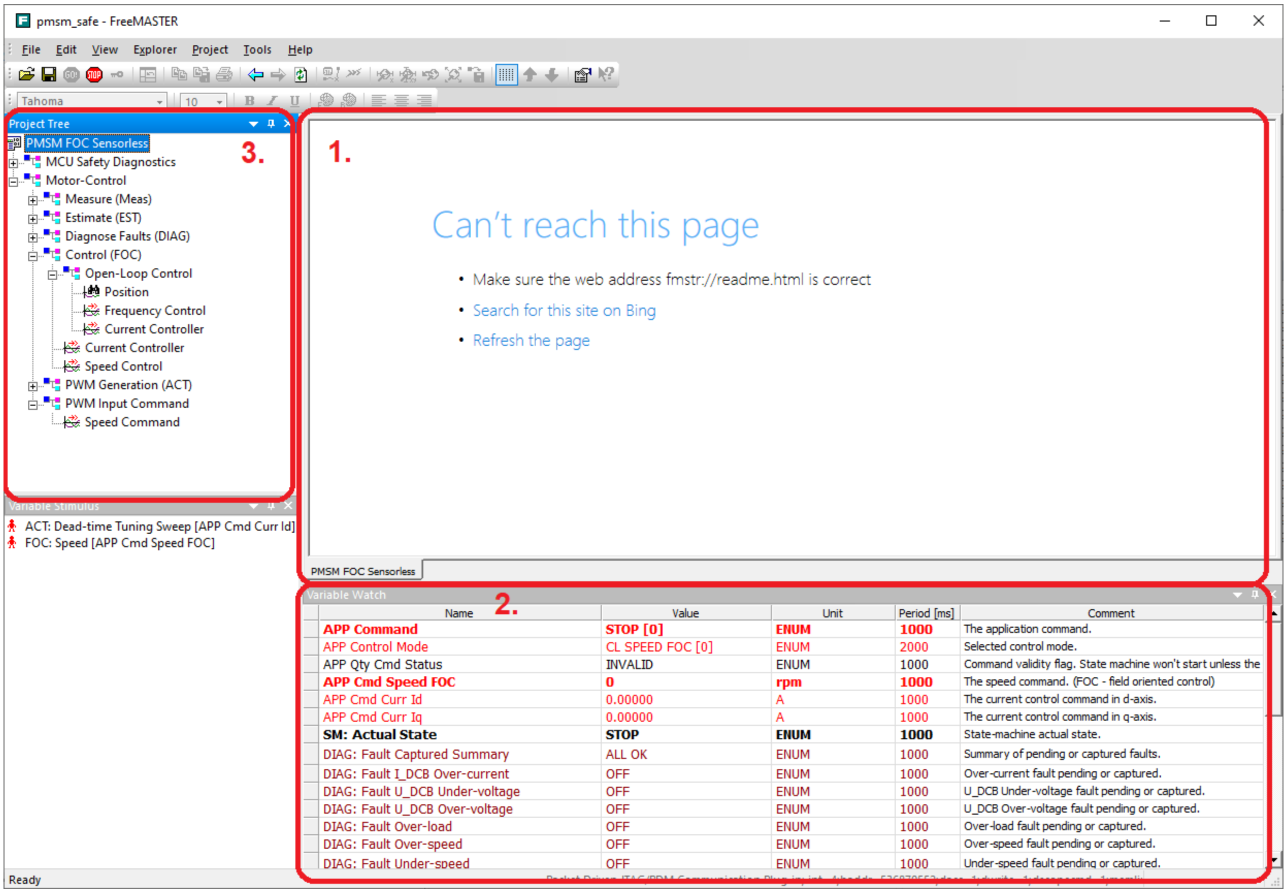


Figure 73. FreeMASTER Go button

If the communication is established successfully, the FreeMASTER communication status in the bottom right-hand corner changes from “*Not connected*” to “*Packet Driven JTAG/BDM Communication Plug-in…*”. Also you should see values of variables in Variable Watch instead question marks.

8. If you rebuild and download the new code to the target, turn the FreeMASTER application off and on.

6.2 FreeMASTER project file description



Can't reach this page

- Make sure the web address fmstr://readme.html is correct
- Search for this site on Bing
- Refresh the page

Name	Value	Unit	Period [ms]	Comment
APP Command	**STOP [0]**	**ENUM**	**1000**	The application command.
APP Control Mode	CL SPEED FOC [0]	ENUM	2000	Selected control mode.
APP Qty Cmd Status	INVALID	ENUM	1000	Command validity flag. State machine won't start unless the
APP Cmd Speed FOC	**0**	**rpm**	**1000**	The speed command. (FOC - field oriented control)
APP Cmd Curr Id	0.00000	A	1000	The current control command in d-axis.
APP Cmd Curr Iq	0.00000	A	1000	The current control command in q-axis.
SM: Actual State	**STOP**	**ENUM**	**1000**	State-machine actual state.
DIAG: Fault Captured Summary	ALL OK	ENUM	1000	Summary of pending or captured faults.
DIAG: Fault I_DCB Over-current	OFF	ENUM	1000	Over-current fault pending or captured.
DIAG: Fault U_DCB Under-voltage	OFF	ENUM	1000	U_DCB Under-voltage fault pending or captured.
DIAG: Fault U_DCB Over-voltage	OFF	ENUM	1000	U_DCB Over-voltage fault pending or captured.
DIAG: Fault Over-load	OFF	ENUM	1000	Over-load fault pending or captured.
DIAG: Fault Over-speed	OFF	ENUM	1000	Over-speed fault pending or captured.
DIAG: Fault Under-speed	OFF	ENUM	1000	Under-speed fault pending or captured.

Figure 74. Default FreeMASTER layout

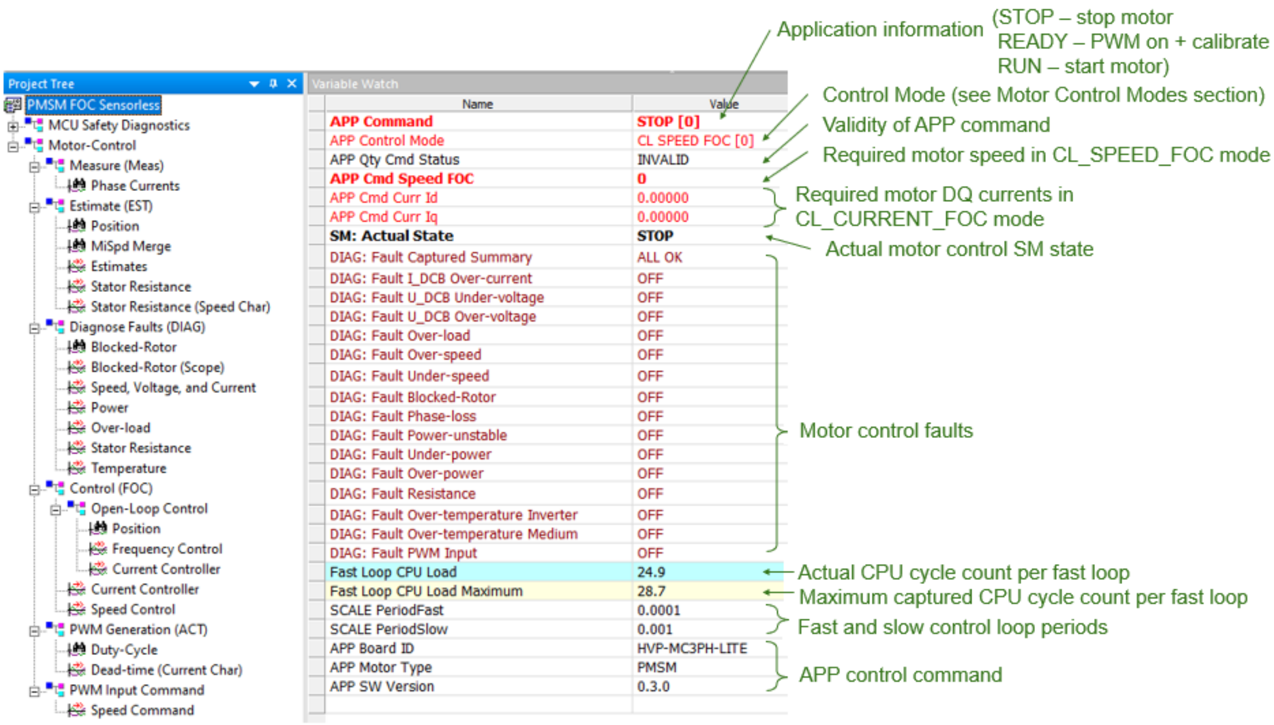
The FreeMASTER window consists of:

1. *Control Page* – The main page of the project, which displays selected scopes and recorders.
2. *Variable Watch* - Set of variables corresponding to the active subblock. See *Comment* for description of each variable. Some variables have read-only value, but some variables can be edited at runtime. To help the user with orientation, the following color-coding was used:
   - Bright red font - Writeable variables intended for control (e.g. the motor required speed, required currents,…).
   - Yellow background - Writeable variables intended for configuration (e.g. control fault thresholds, estimator parameters,…).
   - Blue background - variables are read only and serve as information to user.
   - Dark red font - Used for variables showing pending and captured fault.

Project Tree - Shows the *PMSM FOC Sensorless* FreeMASTER project organization into subblocks, scopes, and recorders. The subblock were created according to software division described in Section 4.3. Every subblock has several scopes and recorders with predefined variables corresponding to its type. Every subblock has also its own variable watch.

6.3 PMSM FOC sensorless project

Figure below shows the "Variable Watch" window content of the "PMSM FOC Sensorless" root item in the "Project Tree" window. Besides the set of basic control variables, it also contains application information. See figure below for description of notable variables.



- PMSM FOC Sensorless
  - MCU Safety Diagnostics
  - Motor-Control
    - Measure (Meas)
      - Phase Currents
    - Estimate (EST)
      - Position
      - MiSpd Merge
      - Estimates
      - Stator Resistance
      - Stator Resistance (Speed Char)
    - Diagnose Faults (DIAG)
      - Blocked-Rotor
      - Blocked-Rotor (Scope)
      - Speed, Voltage, and Current
      - Power
      - Over-load
      - Stator Resistance
      - Temperature
    - Control (FOC)
      - Open-Loop Control
        - Position
        - Frequency Control
        - Current Controller
      - Current Controller
      - Speed Control
    - PWM Generation (ACT)
      - Duty-Cycle
      - Dead-time (Current Char)
    - PWM Input Command
      - Speed Command

Name	Value
APP Command	**STOP [0]**
APP Control Mode	CL SPEED FOC [0]
APP Qty Cmd Status	INVALID
APP Cmd Speed FOC	**0**
APP Cmd Curr Id	0.00000
APP Cmd Curr Iq	0.00000
SM: Actual State	**STOP**
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
Fast Loop CPU Load	24.9
Fast Loop CPU Load Maximum	28.7
SCALE PeriodFast	0.0001
SCALE PeriodSlow	0.001
APP Board ID	HVP-MC3PH-LITE
APP Motor Type	PMSM
APP SW Version	0.3.0

Application information (STOP – stop motor
READY – PWM on + calibrate
RUN – start motor)

Control Mode (see Motor Control Modes section)

Validity of APP command

Required motor speed in CL_SPEED_FOC mode

Required motor DQ currents in
CL_CURRENT_FOC mode

Actual motor control SM state

Motor control faults

Actual CPU cycle count per fast loop

Maximum captured CPU cycle count per fast loop

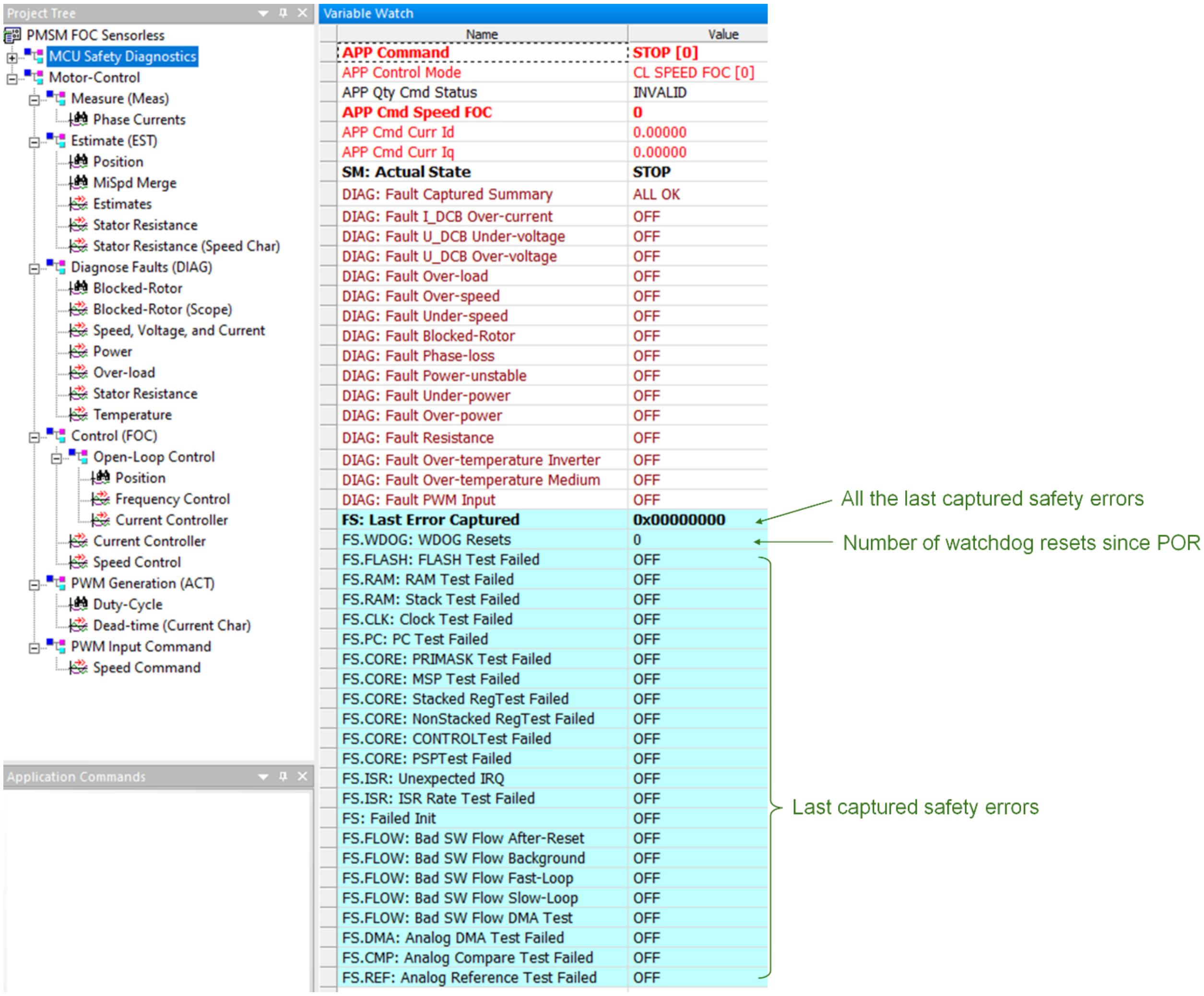
Fast and slow control loop periods

APP control command

Figure 75. FreeMASTER *PMSM FOC Sensorless* subblock

The *PMSM FOC Sensorless* project is organized into the following subblocks:

- *MCU Safety Diagnostics*: Variable watch contains status of actual safety tests. See more details about implemented safety algorithms in Section 4.1. See figure below for description of notable variables.



Project Tree

- PMSM FOC Sensorless
  - MCU Safety Diagnostics
  - Motor-Control
    - Measure (Meas)
      - Phase Currents
    - Estimate (EST)
      - Position
      - MiSpd Merge
      - Estimates
      - Stator Resistance
      - Stator Resistance (Speed Char)
    - Diagnose Faults (DIAG)
      - Blocked-Rotor
      - Blocked-Rotor (Scope)
      - Speed, Voltage, and Current
      - Power
      - Over-load
      - Stator Resistance
      - Temperature
    - Control (FOC)
      - Open-Loop Control
        - Position
        - Frequency Control
        - Current Controller
      - Current Controller
      - Speed Control
    - PWM Generation (ACT)
      - Duty-Cycle
      - Dead-time (Current Char)
    - PWM Input Command
      - Speed Command

Application Commands

Variable Watch

Name	Value
APP Command	STOP [0]
APP Control Mode	CL SPEED FOC [0]
APP Qty Cmd Status	INVALID
APP Cmd Speed FOC	0
APP Cmd Curr Id	0.00000
APP Cmd Curr Iq	0.00000
SM: Actual State	STOP
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
FS: Last Error Captured	0x00000000
FS.WDOG: WDOG Resets	0
FS.FLASH: FLASH Test Failed	OFF
FS.RAM: RAM Test Failed	OFF
FS.RAM: Stack Test Failed	OFF
FS.CLK: Clock Test Failed	OFF
FS.PC: PC Test Failed	OFF
FS.CORE: PRIMASK Test Failed	OFF
FS.CORE: MSP Test Failed	OFF
FS.CORE: Stacked RegTest Failed	OFF
FS.CORE: NonStacked RegTest Failed	OFF
FS.CORE: CONTROLTest Failed	OFF
FS.CORE: PSPTest Failed	OFF
FS.ISR: Unexpected IRQ	OFF
FS.ISR: ISR Rate Test Failed	OFF
FS: Failed Init	OFF
FS.FLOW: Bad SW Flow After-Reset	OFF
FS.FLOW: Bad SW Flow Background	OFF
FS.FLOW: Bad SW Flow Fast-Loop	OFF
FS.FLOW: Bad SW Flow Slow-Loop	OFF
FS.FLOW: Bad SW Flow DMA Test	OFF
FS.DMA: Analog DMA Test Failed	OFF
FS.CMP: Analog Compare Test Failed	OFF
FS.REF: Analog Reference Test Failed	OFF

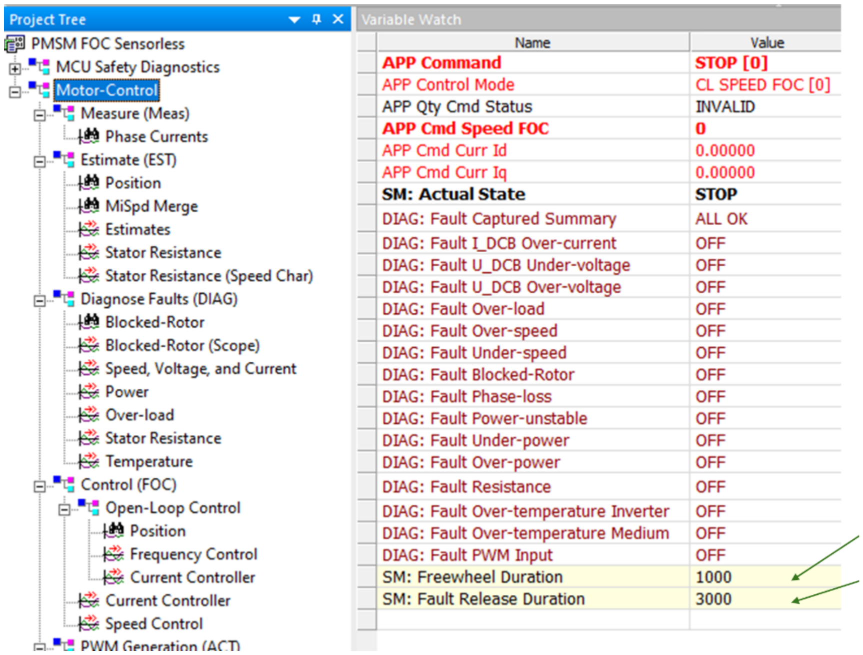
All the last captured safety errors

Number of watchdog resets since POR

Last captured safety errors

Figure 76. FreeMASTER MCU *Safety Diagnostics* subblock

- *Motor-Control*: This subblock groups motor-control-related scopes, recorders, and "Variable Watch" windows based on the affiliation to the measurement (*Meas*), estimation (*EST*), diagnostics (*DIAG*), control *(FOC*), and actuator (*ACT*) parts of the motor-control software (see Section 4.3). See figure below for description of notable variables.



Project Tree

- PMSM FOC Sensorless
  - MCU Safety Diagnostics
  - Motor-Control
    - Measure (Meas)
      - Phase Currents
    - Estimate (EST)
      - Position
      - MiSpd Merge
      - Estimates
      - Stator Resistance
      - Stator Resistance (Speed Char)
    - Diagnose Faults (DIAG)
      - Blocked-Rotor
      - Blocked-Rotor (Scope)
      - Speed, Voltage, and Current
      - Power
      - Over-load
      - Stator Resistance
      - Temperature
    - Control (FOC)
      - Open-Loop Control
        - Position
        - Frequency Control
        - Current Controller
      - Current Controller
      - Speed Control
    - PWM Generation (ACT)

Variable Watch

Name	Value
APP Command	STOP [0]
APP Control Mode	CL SPEED FOC [0]
APP Qty Cmd Status	INVALID
APP Cmd Speed FOC	0
APP Cmd Curr Id	0.00000
APP Cmd Curr Iq	0.00000
SM: Actual State	STOP
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
SM: Freewheel Duration	1000
SM: Fault Release Duration	3000

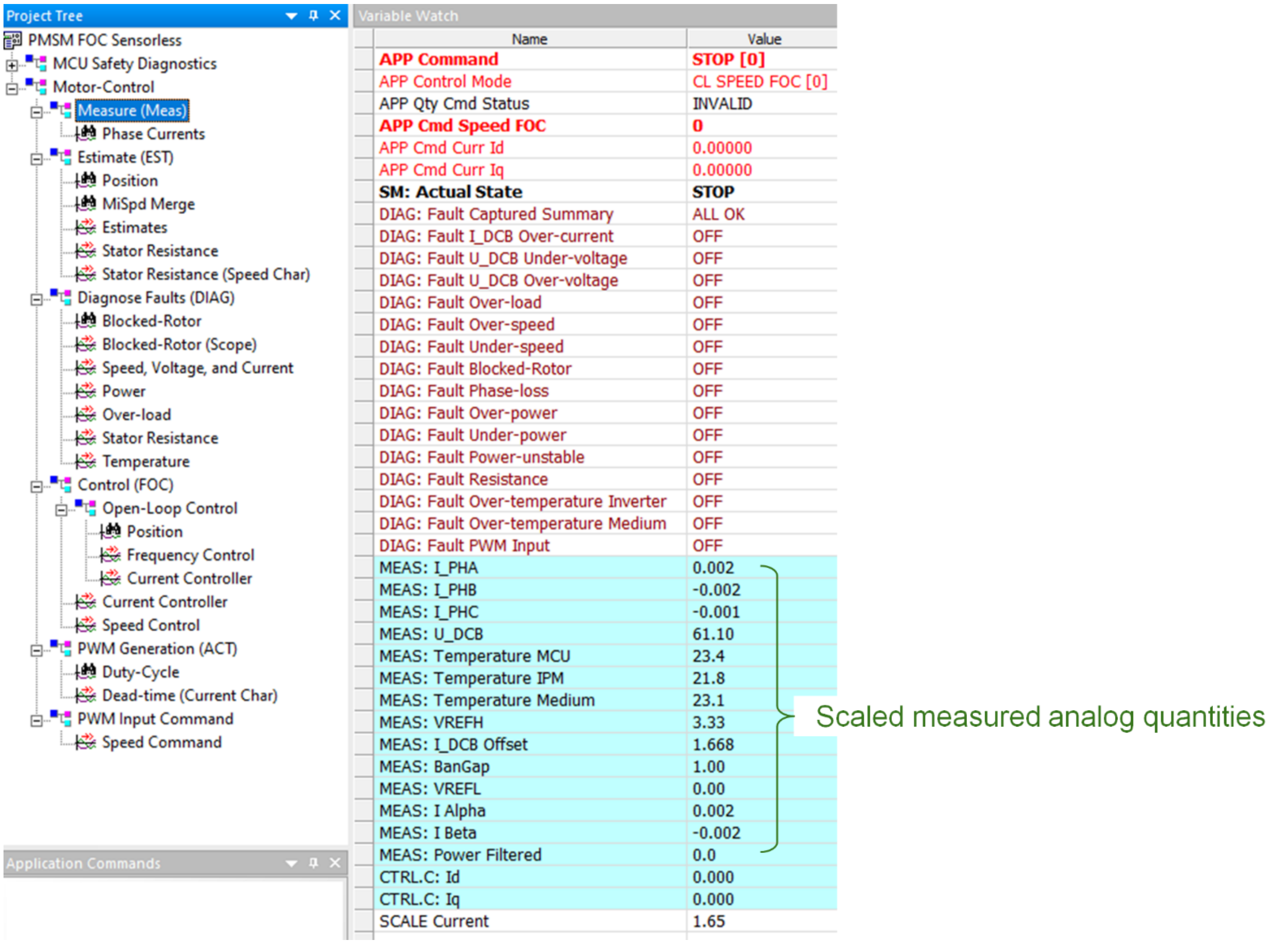
Rotor freewheel duration

Release time to STOP state after FAULT conditions disappear

Figure 77. FreeMASTER *Motor-Control* subblock

The *Motor-Control* subblock is further divided:

- ***Measure (Meas)*** - Variable watch contains all ADC measured variables like phase currents, voltages, temperatures, and many others. The measurement filters can be configured here as well. See figure below for description of notable variables.



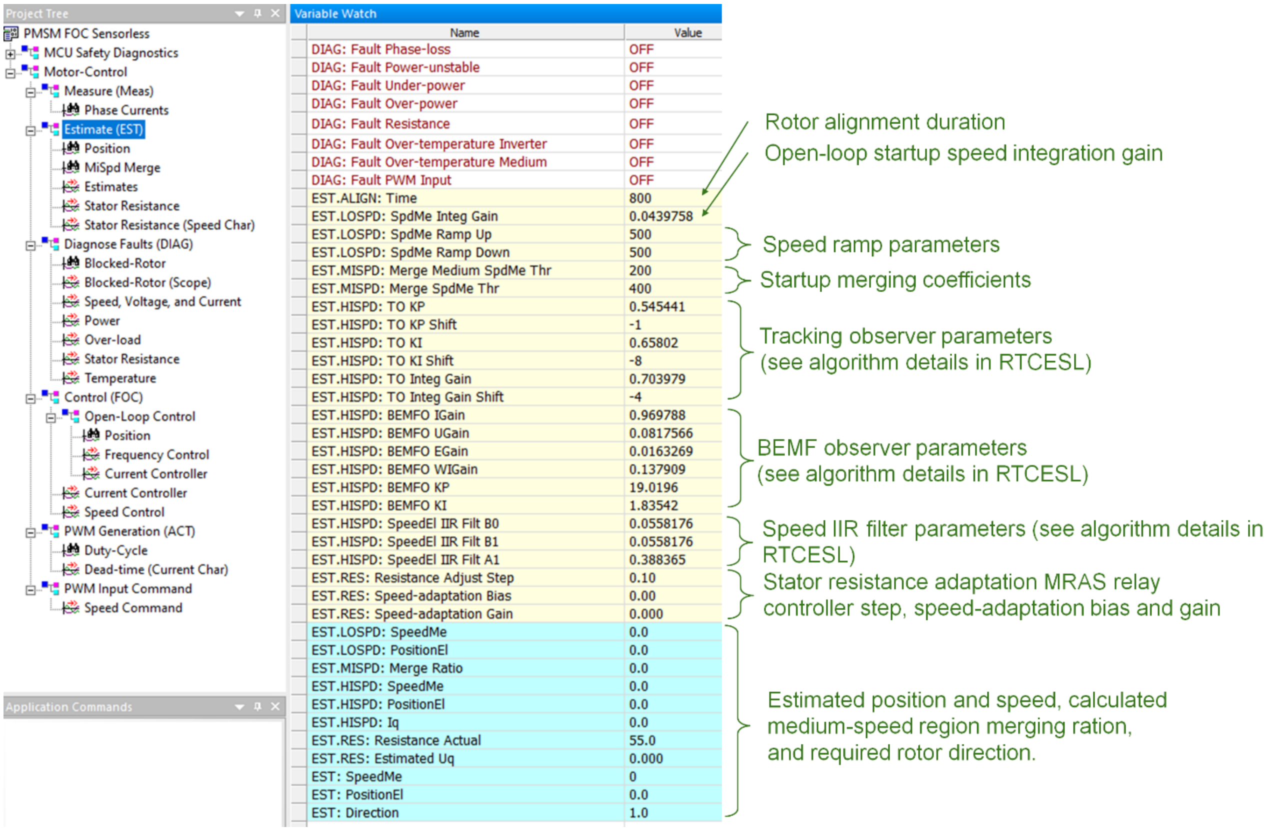
- PMSM FOC Sensorless
  - MCU Safety Diagnostics
  - Motor-Control
    - Measure (Meas)
      - Phase Currents
    - Estimate (EST)
      - Position
      - MiSpd Merge
      - Estimates
      - Stator Resistance
      - Stator Resistance (Speed Char)
    - Diagnose Faults (DIAG)
      - Blocked-Rotor
      - Blocked-Rotor (Scope)
      - Speed, Voltage, and Current
      - Power
      - Over-load
      - Stator Resistance
      - Temperature
    - Control (FOC)
      - Open-Loop Control
        - Position
        - Frequency Control
        - Current Controller
      - Current Controller
      - Speed Control
    - PWM Generation (ACT)
      - Duty-Cycle
      - Dead-time (Current Char)
    - PWM Input Command
      - Speed Command

Name	Value
APP Command	**STOP [0]**
APP Control Mode	CL SPEED FOC [0]
APP Qty Cmd Status	INVALID
APP Cmd Speed FOC	**0**
APP Cmd Curr Id	0.00000
APP Cmd Curr Iq	0.00000
SM: Actual State	**STOP**
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
MEAS: I_PHA	0.002
MEAS: I_PHB	-0.002
MEAS: I_PHC	-0.001
MEAS: U_DCB	61.10
MEAS: Temperature MCU	23.4
MEAS: Temperature IPM	21.8
MEAS: Temperature Medium	23.1
MEAS: VREFH	3.33
MEAS: I_DCB Offset	1.668
MEAS: BanGap	1.00
MEAS: VREFL	0.00
MEAS: I Alpha	0.002
MEAS: I Beta	-0.002
MEAS: Power Filtered	0.0
CTRL.C: Id	0.000
CTRL.C: Iq	0.000
SCALE Current	1.65

Scaled measured analog quantities

Figure 78. FreeMASTER *Measure* subblock

- *Estimate (EST)* – This variable watch contains variables needed for estimation of position and speed (mainly for the tracking observer and BEMF observer). Multiple algorithms are implemented (see Section 4.4.5), each used in the given ALIGN, LOSPD, MISPD, and HISPD state-machine states (matches the zero-, low-, medium-, and high-speed regions). See figure below for description of notable variables.



Name	Value
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
EST.ALIGN: Time	800
EST.LOSPD: SpdMe Integ Gain	0.0439758
EST.LOSPD: SpdMe Ramp Up	500
EST.LOSPD: SpdMe Ramp Down	500
EST.MISPD: Merge Medium SpdMe Thr	200
EST.MISPD: Merge SpdMe Thr	400
EST.HISPD: TO KP	0.545441
EST.HISPD: TO KP Shift	-1
EST.HISPD: TO KI	0.65802
EST.HISPD: TO KI Shift	-8
EST.HISPD: TO Integ Gain	0.703979
EST.HISPD: TO Integ Gain Shift	-4
EST.HISPD: BEMFO IGain	0.969788
EST.HISPD: BEMFO UGain	0.0817566
EST.HISPD: BEMFO EGain	0.0163269
EST.HISPD: BEMFO WIGain	0.137909
EST.HISPD: BEMFO KP	19.0196
EST.HISPD: BEMFO KI	1.83542
EST.HISPD: SpeedEl IIR Filt B0	0.0558176
EST.HISPD: SpeedEl IIR Filt B1	0.0558176
EST.HISPD: SpeedEl IIR Filt A1	0.388365
EST.RES: Resistance Adjust Step	0.10
EST.RES: Speed-adaptation Bias	0.00
EST.RES: Speed-adaptation Gain	0.000
EST.LOSPD: SpeedMe	0.0
EST.LOSPD: PositionEl	0.0
EST.MISPD: Merge Ratio	0.0
EST.HISPD: SpeedMe	0.0
EST.HISPD: PositionEl	0.0
EST.HISPD: Iq	0.0
EST.RES: Resistance Actual	55.0
EST.RES: Estimated Uq	0.000
EST: SpeedMe	0
EST: PositionEl	0.0
EST: Direction	1.0

Rotor alignment duration

Open-loop startup speed integration gain

Speed ramp parameters

Startup merging coefficients

Tracking observer parameters
(see algorithm details in RTCESL)

BEMF observer parameters
(see algorithm details in RTCESL)

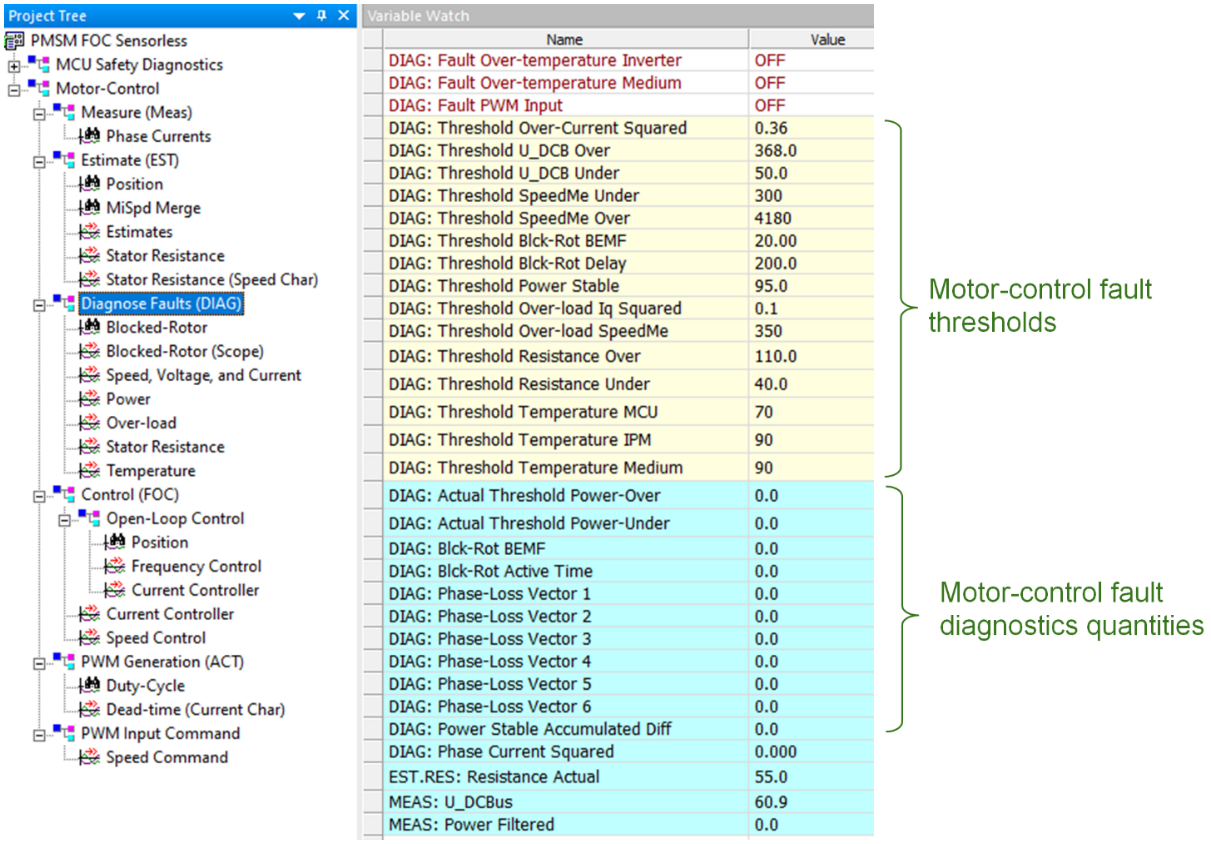
Speed IIR filter parameters (see algorithm details in RTCESL)

Stator resistance adaptation MRAS relay controller step, speed-adaptation bias and gain

Estimated position and speed, calculated medium-speed region merging ration, and required rotor direction.

Figure 79. FreeMASTER *Estimate* subblock

- *Diagnose Faults (DIAG)* - Variable watch consists of motor-control diagnostics and all fault thresholds (e.g. over-voltage, under-voltage, over-current,…). See more details about implemented motor control diagnostic algorithms in Section 4.2. See figure below for description of notable variables.



Project Tree

- PMSM FOC Sensorless
  - MCU Safety Diagnostics
  - Motor-Control
    - Measure (Meas)
      - Phase Currents
    - Estimate (EST)
      - Position
      - MiSpd Merge
      - Estimates
      - Stator Resistance
      - Stator Resistance (Speed Char)
    - Diagnose Faults (DIAG)
      - Blocked-Rotor
      - Blocked-Rotor (Scope)
      - Speed, Voltage, and Current
      - Power
      - Over-load
      - Stator Resistance
      - Temperature
    - Control (FOC)
      - Open-Loop Control
        - Position
        - Frequency Control
        - Current Controller
      - Current Controller
      - Speed Control
    - PWM Generation (ACT)
      - Duty-Cycle
      - Dead-time (Current Char)
    - PWM Input Command
      - Speed Command

Variable Watch

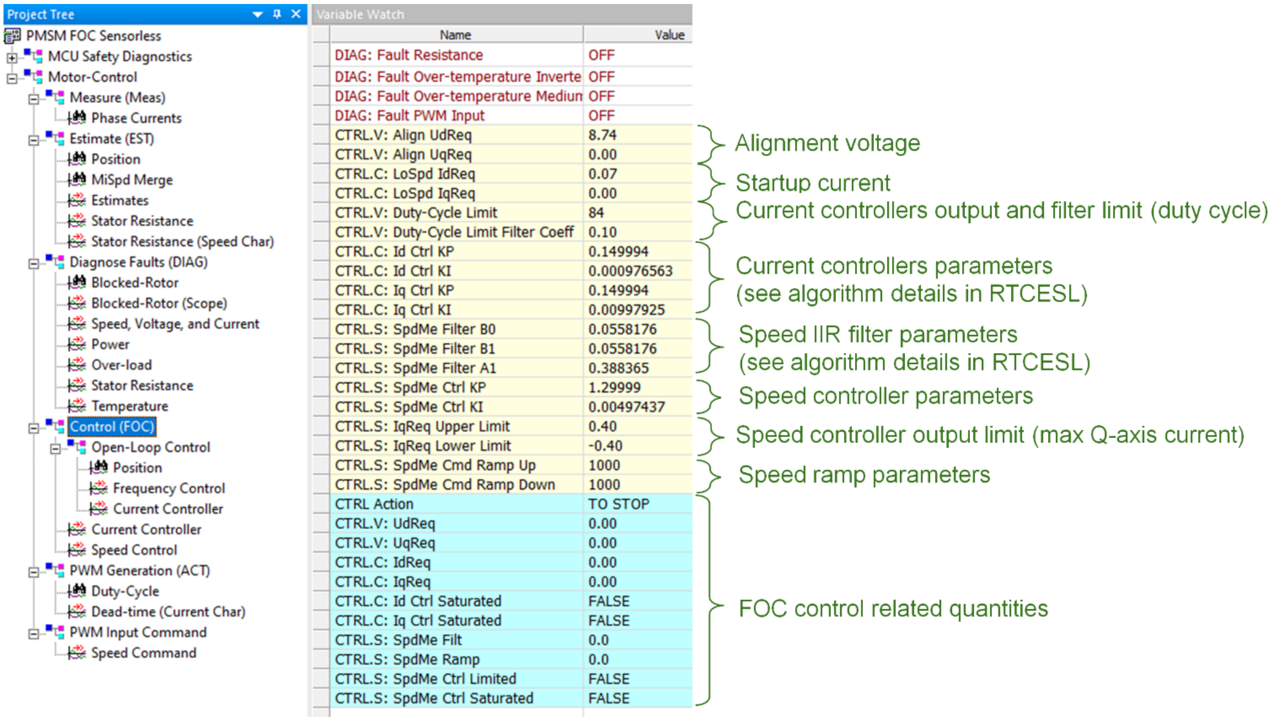
Name	Value
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
DIAG: Threshold Over-Current Squared	0.36
DIAG: Threshold U_DCB Over	368.0
DIAG: Threshold U_DCB Under	50.0
DIAG: Threshold SpeedMe Under	300
DIAG: Threshold SpeedMe Over	4180
DIAG: Threshold Blck-Rot BEMF	20.00
DIAG: Threshold Blck-Rot Delay	200.0
DIAG: Threshold Power Stable	95.0
DIAG: Threshold Over-load Iq Squared	0.1
DIAG: Threshold Over-load SpeedMe	350
DIAG: Threshold Resistance Over	110.0
DIAG: Threshold Resistance Under	40.0
DIAG: Threshold Temperature MCU	70
DIAG: Threshold Temperature IPM	90
DIAG: Threshold Temperature Medium	90
DIAG: Actual Threshold Power-Over	0.0
DIAG: Actual Threshold Power-Under	0.0
DIAG: Blck-Rot BEMF	0.0
DIAG: Blck-Rot Active Time	0.0
DIAG: Phase-Loss Vector 1	0.0
DIAG: Phase-Loss Vector 2	0.0
DIAG: Phase-Loss Vector 3	0.0
DIAG: Phase-Loss Vector 4	0.0
DIAG: Phase-Loss Vector 5	0.0
DIAG: Phase-Loss Vector 6	0.0
DIAG: Power Stable Accumulated Diff	0.0
DIAG: Phase Current Squared	0.000
EST.RES: Resistance Actual	55.0
MEAS: U_DCBus	60.9
MEAS: Power Filtered	0.0

Motor-control fault thresholds

Motor-control fault diagnostics quantities

Figure 80. FreeMASTER *Diagnose Faults* subblock

- *Control (FOC)* - Variable watch contains variables needed for running the motor using with implemented control techniques (see Section 4.4.4). Multiple algorithms are implemented and used based on the state-machine state and the control mode selected. See figure below for description of notable variables.



Name	Value
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverte	OFF
DIAG: Fault Over-temperature Mediun	OFF
DIAG: Fault PWM Input	OFF
CTRL.V: Align UdReq	8.74
CTRL.V: Align UqReq	0.00
CTRL.C: LoSpd IdReq	0.07
CTRL.C: LoSpd IqReq	0.00
CTRL.V: Duty-Cycle Limit	84
CTRL.V: Duty-Cycle Limit Filter Coeff	0.10
CTRL.C: Id Ctrl KP	0.149994
CTRL.C: Id Ctrl KI	0.000976563
CTRL.C: Iq Ctrl KP	0.149994
CTRL.C: Iq Ctrl KI	0.00997925
CTRL.S: SpdMe Filter B0	0.0558176
CTRL.S: SpdMe Filter B1	0.0558176
CTRL.S: SpdMe Filter A1	0.388365
CTRL.S: SpdMe Ctrl KP	1.29999
CTRL.S: SpdMe Ctrl KI	0.00497437
CTRL.S: IqReq Upper Limit	0.40
CTRL.S: IqReq Lower Limit	-0.40
CTRL.S: SpdMe Cmd Ramp Up	1000
CTRL.S: SpdMe Cmd Ramp Down	1000
CTRL Action	TO STOP
CTRL.V: UdReq	0.00
CTRL.V: UqReq	0.00
CTRL.C: IdReq	0.00
CTRL.C: IqReq	0.00
CTRL.C: Id Ctrl Saturated	FALSE
CTRL.C: Iq Ctrl Saturated	FALSE
CTRL.S: SpdMe Filt	0.0
CTRL.S: SpdMe Ramp	0.0
CTRL.S: SpdMe Ctrl Limited	FALSE
CTRL.S: SpdMe Ctrl Saturated	FALSE

Alignment voltage

Startup current

Current controllers output and filter limit (duty cycle)

Current controllers parameters
(see algorithm details in RTCESL)

Speed IIR filter parameters
(see algorithm details in RTCESL)

Speed controller parameters

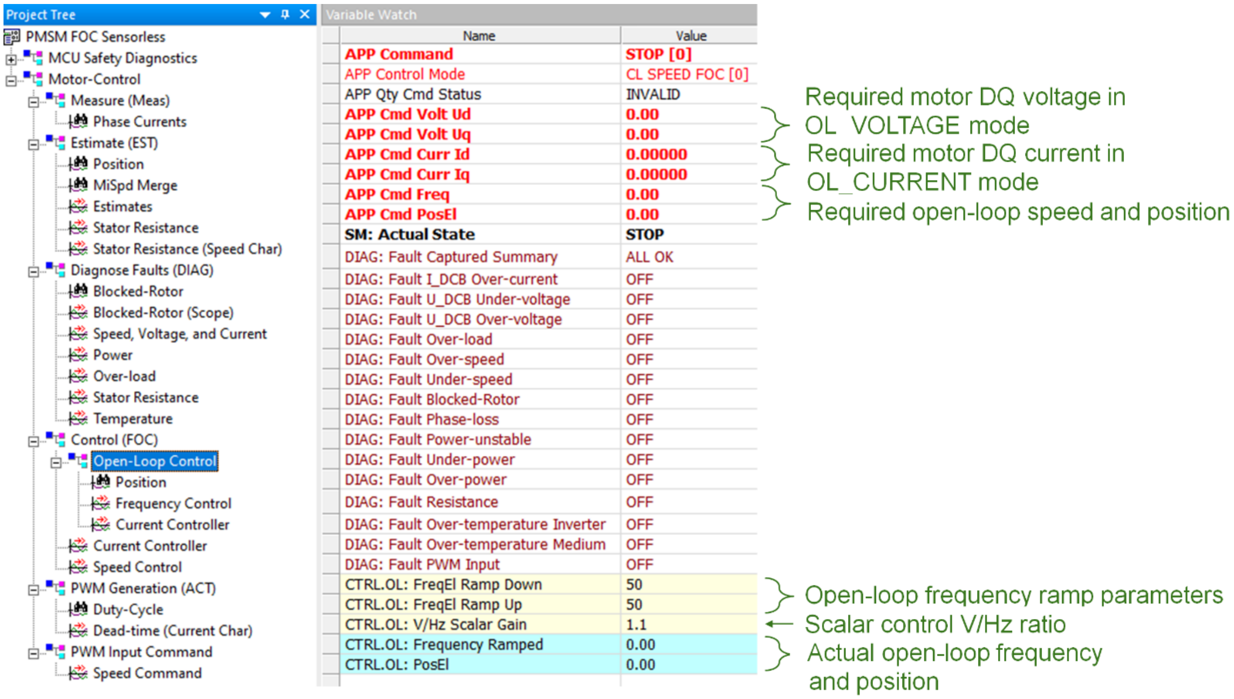
Speed controller output limit (max Q-axis current)

Speed ramp parameters

FOC control related quantities

Figure 81. FreeMASTER *Control (FOC)* subblock

- *Open-Loop Control* - Besides the default Field-Oriented Control (FOC), which allows for fully decoupled speed and torque control, other and simpler control modes are implemented as well (scalar control, open-loop current control, and so on) to allow for easier debugging and tuning. See Section 4.4.4 for more information. To control and configure these algorithms, see the Variable Watch window of the "Open-Loop Control" subblock. See figure below for description of notable variables.



Project Tree

- PMSM FOC Sensorless
  - MCU Safety Diagnostics
  - Motor-Control
    - Measure (Meas)
      - Phase Currents
    - Estimate (EST)
      - Position
      - MiSpd Merge
      - Estimates
      - Stator Resistance
      - Stator Resistance (Speed Char)
    - Diagnose Faults (DIAG)
      - Blocked-Rotor
      - Blocked-Rotor (Scope)
      - Speed, Voltage, and Current
      - Power
      - Over-load
      - Stator Resistance
      - Temperature
    - Control (FOC)
      - Open-Loop Control
        - Position
        - Frequency Control
        - Current Controller
      - Current Controller
      - Speed Control
    - PWM Generation (ACT)
      - Duty-Cycle
      - Dead-time (Current Char)
    - PWM Input Command
      - Speed Command

Variable Watch

Name	Value
APP Command	STOP [0]
APP Control Mode	CL SPEED FOC [0]
APP Qty Cmd Status	INVALID
APP Cmd Volt Ud	0.00
APP Cmd Volt Uq	0.00
APP Cmd Curr Id	0.00000
APP Cmd Curr Iq	0.00000
APP Cmd Freq	0.00
APP Cmd PosEl	0.00
SM: Actual State	STOP
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
CTRL.OL: FreqEl Ramp Down	50
CTRL.OL: FreqEl Ramp Up	50
CTRL.OL: V/Hz Scalar Gain	1.1
CTRL.OL: Frequency Ramped	0.00
CTRL.OL: PosEl	0.00

Required motor DQ voltage in OL VOLTAGE mode

Required motor DQ current in OL_CURRENT mode

Required open-loop speed and position

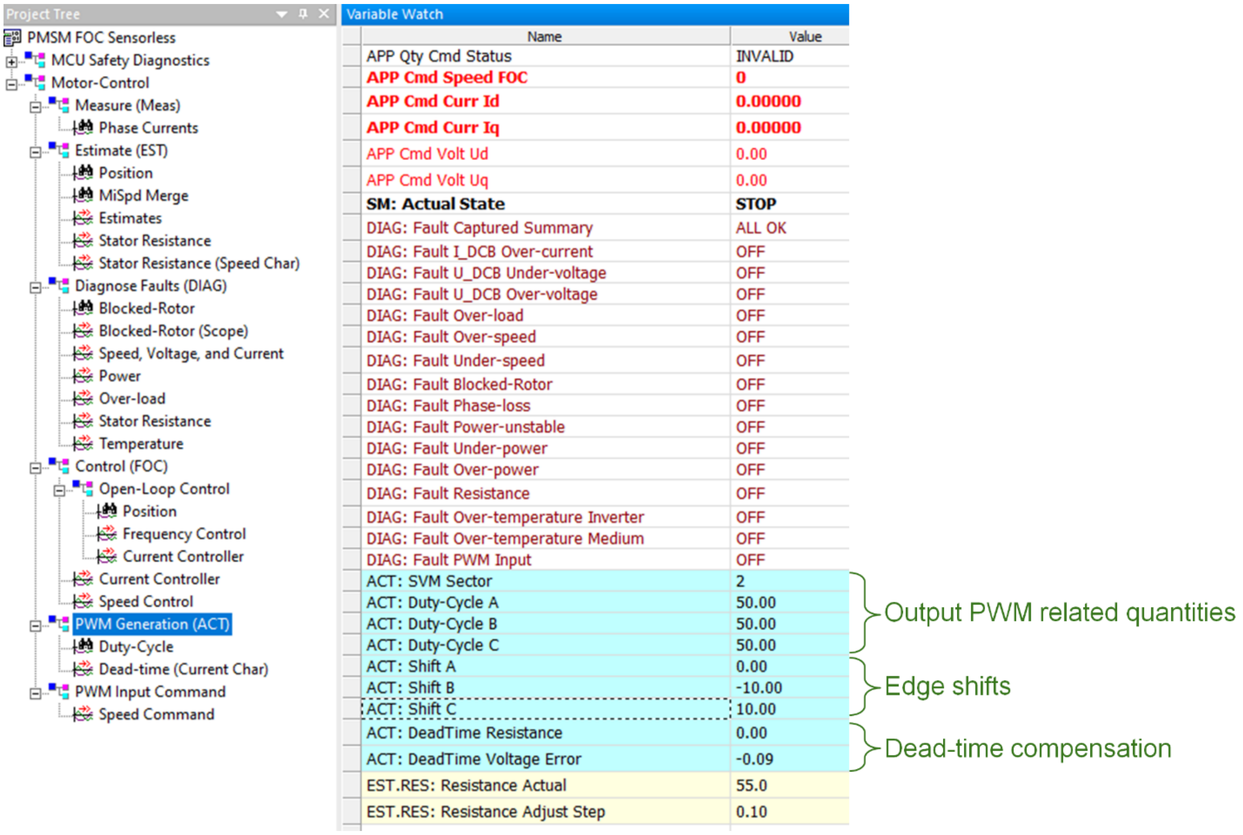
Open-loop frequency ramp parameters

Scalar control V/Hz ratio

Actual open-loop frequency and position

Figure 82. FreeMASTER *Open-Loop Control* subblock

- *PWM Generation (ACT)* – This subblock contains all variables related to the PWM generation (actuator). For more details about implemented algorithms, see Section 4.4.5. See figure below for description of notable variables.



Name	Value
APP Qty Cmd Status	INVALID
APP Cmd Speed FOC	**0**
APP Cmd Curr Id	**0.00000**
APP Cmd Curr Iq	**0.00000**
APP Cmd Volt Ud	0.00
APP Cmd Volt Uq	0.00
SM: Actual State	**STOP**
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
ACT: SVM Sector	2
ACT: Duty-Cycle A	50.00
ACT: Duty-Cycle B	50.00
ACT: Duty-Cycle C	50.00
ACT: Shift A	0.00
ACT: Shift B	-10.00
ACT: Shift C	10.00
ACT: DeadTime Resistance	0.00
ACT: DeadTime Voltage Error	-0.09
EST.RES: Resistance Actual	55.0
EST.RES: Resistance Adjust Step	0.10

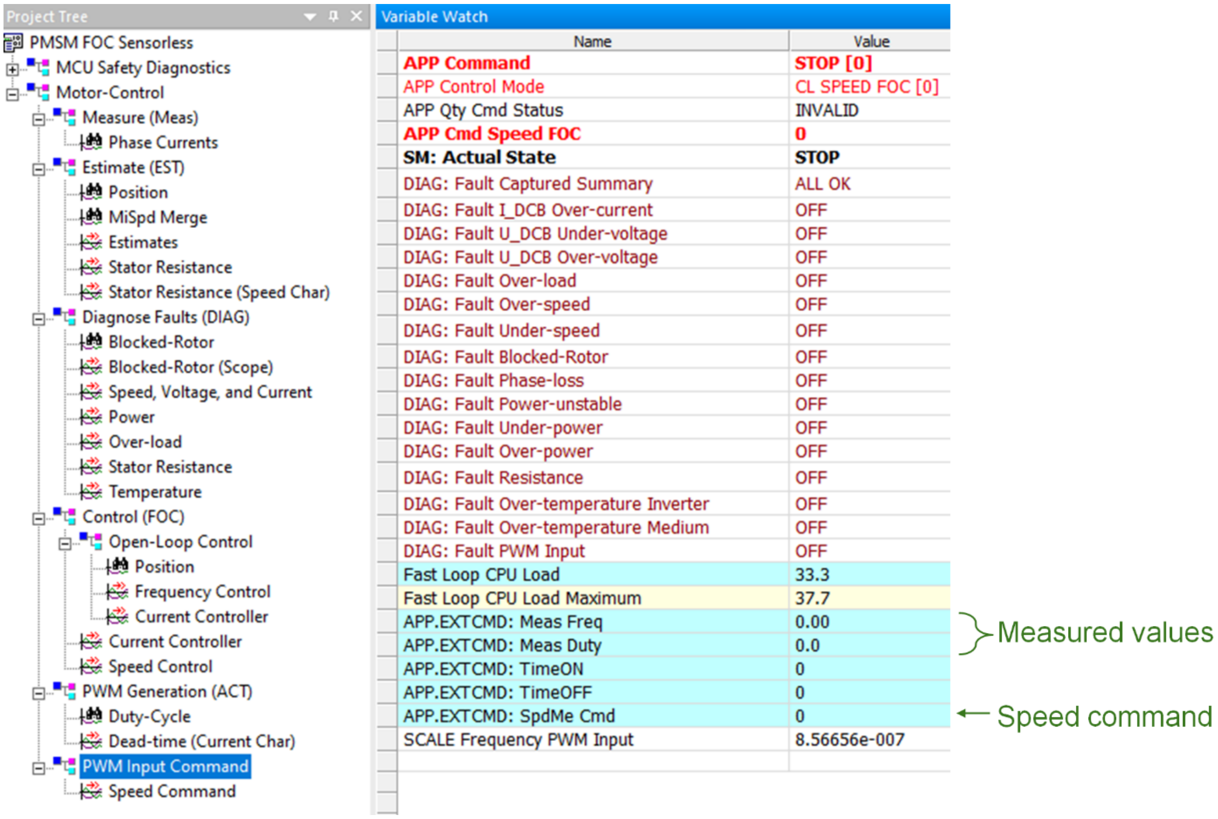
Output PWM related quantities

Edge shifts

Dead-time compensation

Figure 83. FreeMASTER *PWM Generation* subblock

- *PWM input command* – Variable watch contains FreeMASTER variables related to the measurement of external PWM command frequency and duty cycle. See more details in Section 4.4.6.1. See figure below for description of notable variables.



Name	Value
APP Command	STOP [0]
APP Control Mode	CL SPEED FOC [0]
APP Qty Cmd Status	INVALID
APP Cmd Speed FOC	0
SM: Actual State	STOP
DIAG: Fault Captured Summary	ALL OK
DIAG: Fault I_DCB Over-current	OFF
DIAG: Fault U_DCB Under-voltage	OFF
DIAG: Fault U_DCB Over-voltage	OFF
DIAG: Fault Over-load	OFF
DIAG: Fault Over-speed	OFF
DIAG: Fault Under-speed	OFF
DIAG: Fault Blocked-Rotor	OFF
DIAG: Fault Phase-loss	OFF
DIAG: Fault Power-unstable	OFF
DIAG: Fault Under-power	OFF
DIAG: Fault Over-power	OFF
DIAG: Fault Resistance	OFF
DIAG: Fault Over-temperature Inverter	OFF
DIAG: Fault Over-temperature Medium	OFF
DIAG: Fault PWM Input	OFF
Fast Loop CPU Load	33.3
Fast Loop CPU Load Maximum	37.7
APP.EXTCMD: Meas Freq	0.00
APP.EXTCMD: Meas Duty	0.0
APP.EXTCMD: TimeON	0
APP.EXTCMD: TimeOFF	0
APP.EXTCMD: SpdMe Cmd	0
SCALE Frequency PWM Input	8.56656e-007

Measured values

Speed command

Figure 84. FreeMASTER *PWM Input Command* subblock

6.4 Control application

Application is controlled by writing values to variables in variable watch of the FreeMASTER project. To run application in chosen control method follow next sections. To run the motor, no fault must be pending. To change fault settings, switch the FreeMASTER Project Tree to *Diagnose Faults* subblock and perform required changes. To disable functional safety faults rewrite enable/disable macros in *safety_cfg.h*. (e.g. use `FS_CFG_ENABLE_TST_FLASH` to disable flash test or `FS_CFG_ENABLE_TST_CLOCK` to disable clock test). To disable motor control fault checks, modify enable/disable macros in *m1_pmsm_appconfig.h*.

6.4.1 Run motor in open-loop scalar control

Scalar control is very basic form of motor control that is using non-vector approach scheme. Main equations of the motor in time domain are below:

$$u_d = R_s \cdot i_d + \frac{d}{dt}\psi_d - \omega_e \psi_q$$

$$u_q = R_s \cdot i_q + \frac{d}{dt}\psi_q + \omega_e \psi_d$$

In steady state regime, the flux linkage variation is zero, and for further simplification we are going to assume the stator winding resistance is neglectable. Taking into consideration these simplifications and the flux linkage equation then the equations become:

$$u_d = -\omega_e L_q i_q$$

$$u_q = \omega_e L_d i_d + \omega_e \psi_{PM}$$

At this point we can transform the electric speed in frequency and rewrite the equation as a ratio of V/F:

$$\frac{u_d}{f} = -(2\pi)L_q i_q$$

$$\frac{u_q}{f} = (2\pi)\cdot(L_d i_d + \psi_{PM})$$

In V/F scalar control method the frequency of the stator magnetic flux is set according with the desired synchronous rotor speed while the magnitude of the stator voltage is adjusted to keep the ratio between them constant. No control over the voltage or the current vector angles is utilized, hence the name scalar control.

The V/F ratio is calculated from the nominal values of the PMSM voltage and frequency parameters. By maintaining a constant V/F ratio between the amplitude and frequency of three-phase voltage waveforms, then the stator flux of the PMSM can be maintained relatively constant in steady state.

The V/F scalar control is the most common control strategy used for induction motor drives. In case of PMSM, the V/F scalar control is a good alternative in applications where good dynamic performance is not required (e.g. HVAC, fans, pumps or blowers). In such cases the V/F scalar control is performed without the need of a position/speed sensor.

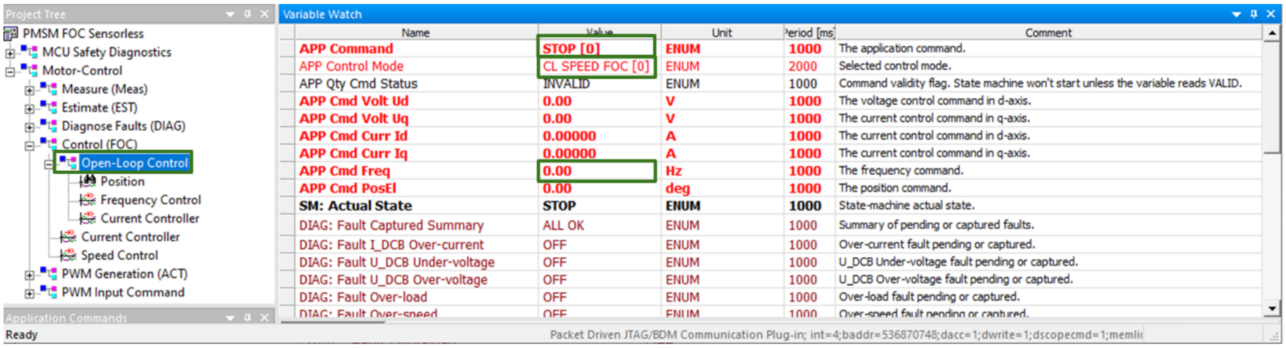
By using V/F scalar control there is no need for high capability CPU as in the case of FOC, but keep in mind that this kind of simplicity also comes with some disadvantages:

- Instability of the system after exceeding a certain applied frequency
- Low dynamic performance, which limits the use of this control method
- Poor fault protection against stall detection and over-currents

Open-loop V/F scalar control is used in applications where system dynamic response is not a concern. For such use cases, the frequency is determined based on the desired speed and the assumption that the rotor will ultimately follow the synchronous speed.

To run motor in open-loop scalar control follow these steps:

1. Switch FreeMASTER Project Tree to *Control (FOC) → Open-Loop Control*
2. In FreeMASTER Variable Watch select scalar control technique OL SCALAR in *APP Control Mode* variable.
3. Set required scalar control frequency to *App Cmd Freq* variable (e.g. example 25 Hz). If the value is valid, *APP Qty Cmd Status* variable shows VALID.
4. Select RUN in *APP Command* variable.
5. The motor should run in scalar control. You can observe electrical position, stator voltage, phase currents and next in the relevant scopes and recorders. If some fault is pending or captured, you can modified fault thresholds in *Diagnose Faults (DIAG)* subblock of Project Tree.



Name	Value	Unit	Period [ms]	Comment
APP Command	STOP [0]	ENUM	1000	The application command.
APP Control Mode	CL SPEED FOC [0]	ENUM	2000	Selected control mode.
APP Qty Cmd Status	INVALID	ENUM	1000	Command validity flag. State machine won't start unless the variable reads VALID.
APP Cmd Volt Ud	0.00	V	1000	The voltage control command in d-axis.
APP Cmd Volt Uq	0.00	V	1000	The current control command in q-axis.
APP Cmd Curr Id	0.00000	A	1000	The current control command in d-axis.
APP Cmd Curr Iq	0.00000	A	1000	The current control command in q-axis.
APP Cmd Freq	0.00	Hz	1000	The frequency command.
APP Cmd PosEl	0.00	deg	1000	The position command.
SM: Actual State	STOP	ENUM	1000	State-machine actual state.
DIAG: Fault Captured Summary	ALL OK	ENUM	1000	Summary of pending or captured faults.
DIAG: Fault I_DCB Over-current	OFF	ENUM	1000	Over-current fault pending or captured.
DIAG: Fault U_DCB Under-voltage	OFF	ENUM	1000	U_DCB Under-voltage fault pending or captured.
DIAG: Fault U_DCB Over-voltage	OFF	ENUM	1000	U_DCB Over-voltage fault pending or captured.
DIAG: Fault Over-load	OFF	ENUM	1000	Over-load fault pending or captured.
DIAG: Fault Over-speed	OFF	ENUM	1000	Over-speed fault pending or captured.

Ready

Packet Driven JTAG/BDM Communication Plug-in; int=4;baddr=536870748;dacc=1;dwrite=1;dscopecmd=1;memli

Figure 85. Scalar Control mode variables

6.4.2 Run motor in open-loop current control mode variables

The current controllers are engaged in this mode. Set the required amplitude of *dq*-axis currents, the current vector rotation frequency, and the position bias. The BEMF and Tracking observers are running in the background, so this control mode can be used for the parameter tuning of observers and current controllers. To enable the Open Loop Current control, perform the following steps:

1. Switch FreeMASTER Project Tree to *Control (FOC) → Open-Loop Control*
2. In FreeMASTER Variable Watch select scalar control technique OL CURRENT in *APP Control Mode* variable.
3. Set the required amplitude of the *dq*-axis currents (*APP Cmd Curr Id* and *APP Cmd Curr Iq* variables), the current vector open-loop frequency (*APP Cmd Freq* variable), and the position bias (*APP Cmd PosEl* variable).. If the value is valid, *APP Qty Cmd Status* variable shows VALID.
4. Select RUN in *APP Command* variable.

6.4.3 Run motor in open-loop voltage control

The current controllers are disabled in this mode and the stator voltage is controlled directly. Set the required amplitude of *dq*-axis voltages, the voltage vector rotation frequency, and the position bias. The BEMF and tracking observers are running in the background, so this control mode can be used for parameter tuning of observers and basic debugging for the PWM generation. To enable the Open Loop Voltage control, perform the following steps:

1. Switch FreeMASTER Project Tree to Control (FOC) → Open-Loop Control
2. In FreeMASTER Variable Watch select scalar control technique OL VOLTAGE in APP Control Mode variable.
3. Set the required amplitude of *dq*-axis voltages (*APP Cmd Volt Ud* and *APP Cmd Volt Uq* variables), the voltage vector open-loop frequency (*APP Cmd Freq* variable), and the position bias (*APP Cmd PosEl* variable). If the value is valid, *APP Qty Cmd Status* variable shows VALID.
4. Select RUN in *APP Command* variable.

6.4.4 Run motor in close-loop current (torque) FOC

High-performance motor control is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. To achieve such control, Field Oriented Control is used for PMSM motors. The FOC concept is based on an efficient torque control requirement, which is essential for achieving a high control dynamic. Analogous to standard DC machines, AC machines develop maximal torque when the armature current vector is perpendicular to the flux linkage vector. Therefore, if only the fundamental harmonic of stator magnetomotive force is considered, the torque T_e developed by an AC machine, in vector notation, is given by

$$T_e = \frac{3}{2} \cdot pp \cdot \overline{\psi_s} \times \bar{\imath}_s$$

where *pp* is the number of motor pole-pairs, i_s stator current vector and ψ_s represents vector of the

stator flux. Constant 3/2 indicates a non-power invariant transformation form.

In instances of DC machines, the requirement to have the rotor flux vector perpendicular to the stator

current vector is satisfied by the mechanical commutator. Because there is no such mechanical

commutator in PMSM, the functionality of the commutator has to be substituted electrically by enhanced current control. This reveal that stator current vector should be oriented in such a way that component necessary for magnetizing of the machine (flux component) shall be isolated from the torque producing component. This can be accomplished by decomposing the current vector into two components projected in the reference frame, often called the *dq* frame that rotates synchronously with the rotor. It has become a standard to position the *dq*-axis reference frame such that the *d*-axis is aligned with the position of the rotor lux vector, so that the current in the d-axis will alter the amplitude of the rotor flux linkage vector. The reference frame position must be updated so that the *d*-axis should be always aligned with the rotor flux axis.

Because the rotor flux axis is locked to the rotor position, when using PMSM machines, a mechanical position transducer or position observer can be utilized to measure the rotor position and the position of

the rotor flux axis. When the reference frame phase is set such that the d-axis is aligned with the rotor

flux axis, the current in the *q*-axis represents solely the torque producing current component.

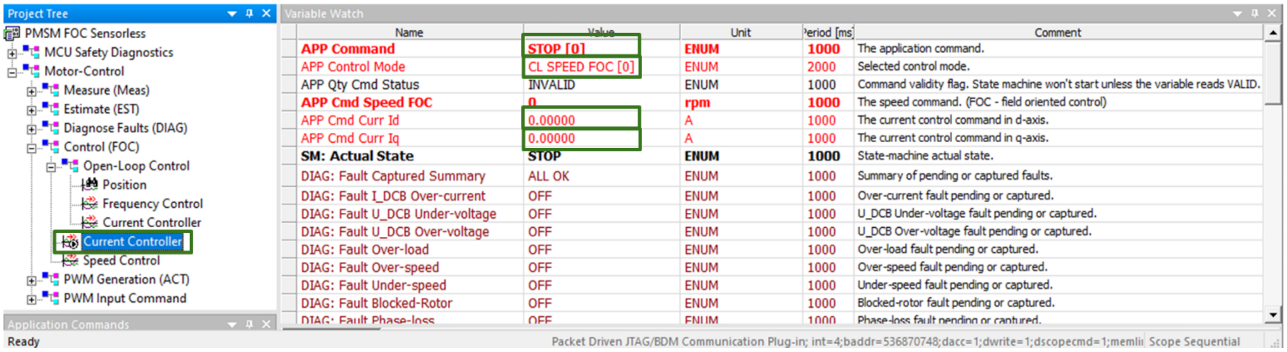
What further resulted from setting the reference frame speed to be synchronous with the rotor flux axis

speed is that both d-axis and *q*-axis current components are DC values. This implies utilization of simple

current controllers to control the demanded torque and magnetizing flux of the machine, therefore simplifying the control structure design. The torque control method is generic and may be applicable to all sort of means of transportation: trains, buses, bikes, scooters and modern day cars.

To run motor in close-loop current control follow these steps:

1. Switch FreeMASTER Project Tree to *Control (FOC).*
2. In FreeMASTER Variable Watch select current control technique CL CURRENT FOC in *APP Control Mode* variable.
3. Set required current in q-axis to *APP Cmd Curr Iq* variable (e.g. 0.05 A). If the value is valid, *APP Qty Cmd Status* variable shows VALID.
4. Select RUN in *APP Command* variable.
5. The motor should run in close-loop current control. You can observe electrical position, stator voltage, phase currents, speed and next in the relevant scopes and recorders. If some fault is pending or captured, you can modified fault thresholds in *Diagnose Faults (DIAG)* subblock of the Project Tree.



Name	Value	Unit	Period [ms]	Comment
APP Command	STOP [0]	ENUM	1000	The application command.
APP Control Mode	CL SPEED FOC [0]	ENUM	2000	Selected control mode.
APP Qty Cmd Status	INVALID	ENUM	1000	Command validity flag. State machine won't start unless the variable reads VALID.
APP Cmd Speed FOC	0	rpm	1000	The speed command. (FOC - field oriented control)
APP Cmd Curr Id	0.00000	A	1000	The current control command in d-axis.
APP Cmd Curr Iq	0.00000	A	1000	The current control command in q-axis.
SM: Actual State	STOP	ENUM	1000	State-machine actual state.
DIAG: Fault Captured Summary	ALL OK	ENUM	1000	Summary of pending or captured faults.
DIAG: Fault I_DCB Over-current	OFF	ENUM	1000	Over-current fault pending or captured.
DIAG: Fault U_DCB Under-voltage	OFF	ENUM	1000	U_DCB Under-voltage fault pending or captured.
DIAG: Fault U_DCB Over-voltage	OFF	ENUM	1000	U_DCB Over-voltage fault pending or captured.
DIAG: Fault Over-load	OFF	ENUM	1000	Over-load fault pending or captured.
DIAG: Fault Over-speed	OFF	ENUM	1000	Over-speed fault pending or captured.
DIAG: Fault Under-speed	OFF	ENUM	1000	Under-speed fault pending or captured.
DIAG: Fault Blocked-Rotor	OFF	ENUM	1000	Blocked-rotor fault pending or captured.
DIAG: Fault Phase-loss	OFF	ENUM	1000	Phase-loss fault pending or captured.

Ready

Packet Driven JTAG/BDM Communication Plug-in; int=4;baddr=536870748;dacc=1;dwrite=1;dscopecmd=1;memli Scope Sequential

Figure 86. Current FOC control mode variables

6.4.5 Run motor in sensorless speed FOC

To run motor in sensorless speed FOC we need add to the current (torque) control one more PI controller. Its output is required current. Input to controller is difference between required speed (set by user) and actual rotor speed. In sensorless mode is the actual rotor speed and position computed using back-EMF observer.

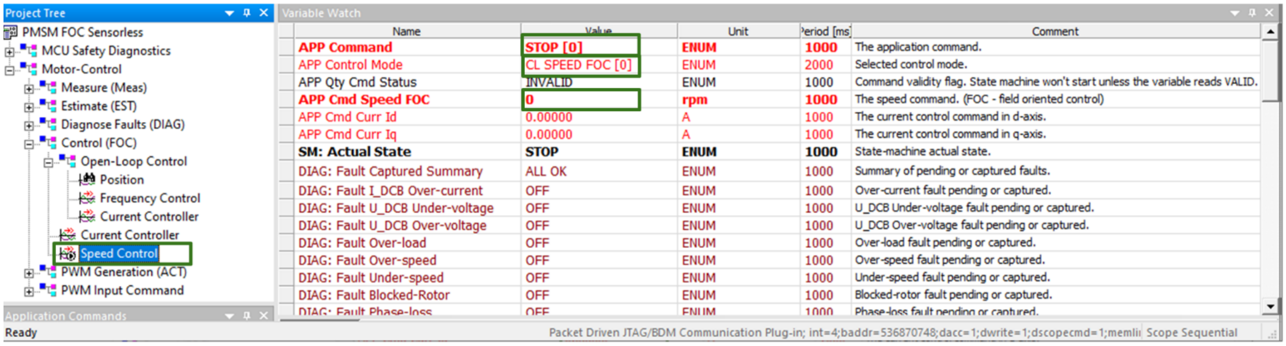
Back-EMF observer provides only relative position. To get absolute position, initial position must be known. Therefore application uses mechanical rotor alignment when the rotor is moved from unknown to known position applying DC voltage.

The alignment algorithm applies DC voltage to *d*-axis resulting full DC voltage applied to phase A and negative half of the DC voltage applied to phase B, C for a certain period. This will cause the rotor to move to "align" position, where stator and rotor fluxes are aligned. The rotor position in which the rotor stabilizes after applying DC voltage is set as zero position. Motor is ready to produce full startup torque once the rotor is properly aligned.

Application in sensorless mode must start with open loop start-up sequence to move the motor up to a speed value where the observer provides sufficiently accurate speed and position estimations. As soon as the observer provides appropriate estimates, application transits to closed-loop mode, when the rotor speed and position calculation is based on the estimation of a BEMF in the stationary reference frame. Back-EMF observer is as a part of the NXP's RTCESL (see documentation at www.nxp.com/rtcesl).

To run motor in sensorless speed FOC control follow these steps:

1. Switch FreeMASTER Project Tree to *Control (FOC).*
2. In FreeMASTER Variable Watch select speed FOC control technique CL SPEED FOC in *APP Control Mode* variable.
3. Set required speed to *APP Cmd Speed FOC* variable. For example 500rpm. If the value is valid, *APP Qty Cmd Status* variable shows VALID.
4. Select RUN in *APP Command* variable.
5. The motor should run in sensorless speed control. You can observe electrical position, stator voltage, phase currents, speed and next in the relevant scopes and recorders. If some fault is pending or captured, you can modified fault thresholds in *Diagnose Faults (DIAG)* Project Tree.



Name	Value	Unit	Period [ms]	Comment
APP Command	STOP [0]	ENUM	1000	The application command.
APP Control Mode	CL SPEED FOC [0]	ENUM	2000	Selected control mode.
APP Qty Cmd Status	INVALID	ENUM	1000	Command validity flag. State machine won't start unless the variable reads VALID.
APP Cmd Speed FOC	0	rpm	1000	The speed command. (FOC - field oriented control)
APP Cmd Curr Id	0.00000	A	1000	The current control command in d-axis.
APP Cmd Curr Iq	0.00000	A	1000	The current control command in q-axis.
SM: Actual State	STOP	ENUM	1000	State-machine actual state.
DIAG: Fault Captured Summary	ALL OK	ENUM	1000	Summary of pending or captured faults.
DIAG: Fault I_DCB Over-current	OFF	ENUM	1000	Over-current fault pending or captured.
DIAG: Fault U_DCB Under-voltage	OFF	ENUM	1000	U_DCB Under-voltage fault pending or captured.
DIAG: Fault U_DCB Over-voltage	OFF	ENUM	1000	U_DCB Over-voltage fault pending or captured.
DIAG: Fault Over-load	OFF	ENUM	1000	Over-load fault pending or captured.
DIAG: Fault Over-speed	OFF	ENUM	1000	Over-speed fault pending or captured.
DIAG: Fault Under-speed	OFF	ENUM	1000	Under-speed fault pending or captured.
DIAG: Fault Blocked-Rotor	OFF	ENUM	1000	Blocked-rotor fault pending or captured.
DIAG: Fault Phase-loss	OFF	ENUM	1000	Phase-loss fault pending or captured.

Ready

Packet Driven JTAG/BDM Communication Plug-in; int=4;baddr=536870748;dacc=1;dwrite=1;dscopecmd=1;memli Scope Sequential

Figure 87. Speed FOC control mode variables

6.4.5.1 Run motor in speed FOC using external PWM command

To run motor in sensorless speed FOC control using external PWM command follow these steps:

1. Switch FreeMASTER Project Tree to *Control (FOC).*
2. In FreeMASTER Variable Watch select speed control technique CL SPEED EXT in *APP Control Mode* variable.
3. Apply the control PWM to external command PWM input with duty cycle from 10 % to 90 % with base frequency from 200 Hz to 2 kHz.
4. Check PWM input command in *PWM Input Command* subblock of Project Tree in Speed Command scope. If the input frequency is valid, the *APP Qty Cmd Status* variable shows VALID.
5. Select RUN in *APP Command* variable.

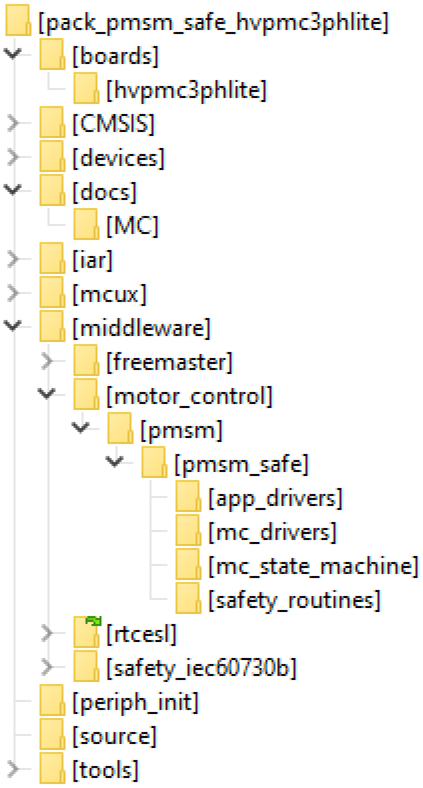
The motor should run in sensorless speed control. You can observe electrical position, stator voltage, phase currents, speed and next in the relevant scopes and recorders. If some fault is pending or captured, you can modified fault thresholds in *Diagnose Faults (DIAG)* in the Project Tree.

Chapter 7
Project files 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 in a logical manner. The folder structure used in the IDE is different from the structure of the PMSM package installation, but it uses the same files. The different organization is chosen due to a better manipulation with folders and files in workplaces and due to the possibility to add or remove files and directories. The *pack_pmsm_safe_hvpmc3phlite* package includes *mc_pmsm_safe* project, all available functions and routines, scalar and vector control of the motor and the FreeMASTER project file.

7.1 Directory structure

The directory tree of the *pack_pmsm_safe_hvpmc3phlite* package is shown in figure below.



```
[pack_pmsm_safe_hvpmc3phlite]
    [boards]
        [hvpmc3phlite]
    [CMSIS]
    [devices]
    [docs]
        [MC]
    [iar]
    [mcux]
    [middleware]
        [freemaster]
        [motor_control]
            [pmsm]
                [pmsm_safe]
                    [app_drivers]
                    [mc_drivers]
                    [mc_state_machine]
                    [safety_routines]
        [rtcesl]
        [safety_iec60730b]
    [periph_init]
    [source]
    [tools]
```

Figure 88. Directory structure

The package *pack_pmsm_safe_hvpmc3phlite* contains these folders and files:

- Folder *mcux* – folder containing project files for the MCUXpresso IDE.
- Folder *iar* – folder containing project files including *mc_pmsm_safe.eww* for the IAR Embedded Workbench IDE.
- Folder *middleware/motor_control/pmsm/pmsm_safe* – contains main PMSM motor-control functions and drivers:
    - Folder *app_drivers* contains the source and header files used to initialize and run motor application using external PWM command.
    - Folder *mc_drivers* contains the source and header files used to initialize and run motor-control applications.
    - Folder *mc_state_machine* contains the software routines that are executed when the application is in a particular state or state transition. In this folder are located also files with control methods.

  - Folder *safety_routines* contains the safety software routines.
- Folder *periph_init* consist from initialization files:
  - *app_periph_init.c and .h* – The files contains initialization of the FTM1 periphery for PWM input control signal measurement.
  - *m1_periph_init.c and .h* - The files contains initialization of motor control peripherals.
  - *safety_periph_init.c and .h* – The files contains clock and pins initialization.
- Folder *source* contains these files:
  - *m1_pmsm_appconfig.h* – contains the definitions of constants for the application control processes, parameters of the motor and regulators, and the constants for other vector-control-related algorithms.
  - *main.c and h* – contains the basic application initialization, subroutines for accessing the MCU peripherals, and interrupt service routines. The FreeMASTER communication is processed in the background infinite loop.
  - *application.c and h* – contains the definitions of control variables and structures.
  - *freemaster_cfg.h* – FreeMASTER configuration file
  - *hardware_cfg.h* – The hardware configuration file containing hardware setup like ISRs, clocks, and pin-muxing.
  - *safety_cfg.h* – The safety configuration file containing safety tests setup.

Chapter 8
Identifying parameters of user motor

Because the model-based control methods of the PMSM drives are the most effective and usable, obtaining an accurate model of a motor is an important part of the drive design and control. For the implemented FOC algorithms, it is necessary to know the value of the stator resistance R_s, direct inductance L_d, quadrature inductance L_q, and BEMF constant K_e. These motor parameters are used for computing PI controllers, Tracking observer and BEMF observer parameters. The parameters are stored in *m1_pmsm_appconfig.h*.

Identification of parameters of the user‘s motor is possible by several ways, which are described in the following sections.

8.1 Parameter identification using SDK example and MCAT

- The motor identification software is available just for floating point cores. For example, the NXP HVP-KV31F with M4F core and HVP-MC3PH high-voltage platform is a suitable platform. Download SDK example for HVP-KV31F (2.9.0).
- Follow Motor Identification chapter 9.4 in SDK example documentation (MCUXpresso SDK 3-Phase PMSM Control with IEC60730 Safety).
- Copy measured and computed parameters to *m1_pmsm_appconfig.h*.

NOTE

Is possible use obsolete motor identification using MCAT in SDK example for HVP-KV11Z (2.10.0). Follow SDK example documentation MCUXpresso SDK 3-Phase PMSM Control(KV) (rev3).

NOTE

Motor control application tool (MCAT) from SDK example is possible use also without board, that mean offline. This option is favorable if the motor parameters are known and we need just compute control parameters of PI controllers, observers and filters.

8.2 Parameter identification using manual measurement

First, follow instructions in document PMSM Electrical Parameters Measurement to perform the measurements. To compute and update the parameters in the software, use computing formulas described in Section 8.3.

8.3 Computing of control parameters

This approach assumes that the machine parameters were already obtained (e.g. from manufacturers documentation). To completely configure the motor control application, the user must first summarize all the configured parameters (see Section 8.3.1 for summary example) and then calculate correct values, which shall then be entered into configuration defines in *m1_pmsm_appconfig.h* (see Section 8.3.2 for the calculation script).

8.3.1 Example summary of configuration parameters

```
%Motor parameters

pp = 3; %[-] Motor number of pole-pairs

Rs = 55.94; %[Ohm] Stator phase resistance

Ld = 0.179701; %[H] Stator direct inductance

Lq = 0.184883; %[H] Stator quadrature inductance

J = 0.0000016; %[kg.m2] Drive inertia

Iph_nom = 0.45; %[A] Nominal motor current

Uph_nom = 250; %[V] Nominal motor voltage
```

```
N_nom = 4400; %[rpm] Nominal motor speed

%Hardware scales

I_max = 1.65; %[A] Current sensing HW scale

U_DCB_max = 433; %[V] DC-bus voltage sensing HW scale

%Fault limits

U_DCB_trip = 346.4; %[V] DC-bus braking resistor threshold

U_DCB_under = 173.2; %[V] DC-bus under voltage fault threshold

U_DCB_over = 346.4; %[V] DC-bus over voltage fault threshold

N_over = 4180; %[rpm] Over speed fault threshold

N_min = 400; %[rpm] Minimal closed loop speed

E_block = 7; %[V] Blocked rotor detection BEMF voltage level

Scalar_Uq_min = 4; %[V] Scalar control voltage bias

%Application Scales

N_max = 4400; %[rpm] Application speed scale

E_max = 50; %[V] FOC BEMF maximum limit

kt = 0.01217; %[Nm/A] Torque constant

%Alignment

Align_voltage = 6; %[V] Voltage applied on d-axis for mechanical rotor alignment

Align_duration = 0.8; %[sec] Time of rotor alignment

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Current Control Loop - CL %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Loop Parameters

CL_SampleTime = 0.0001; %[sec] Current control loop sampling period

CL_F0 = 280; %[Hz] Current control loop bandwidth

CL_Ksi = 1; %[-] Current control loop attenuation

%Current PI Controller Limits

CL_OutputLimit = 90; %[%] Limit of current loop in percentage of DC-bus voltage

%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Speed Control Loop - SL %

%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Loop Parameters

SL_SampleTime = 0.001; %[sec] Speed control loop sampling period

SL_F0 = 10; %[Hz] Speed control loop bandwidth

SL_Ksi = 1; %[-] Speed control loop attenuation

%Speed Ramp

SL_IncUp = 5000; %[rpm/sec] Speed ramp increment up

SL_IncDown = 5000; %[rpm/sec] Speed ramp increment down
```

```
%Actual Speed Filter
SL_CutOffFreq = 100; %[Hz] Cut-off frequency of IIR speed measurement filter
%Speed PI Controller Limits
SL_UpperLimit = 2; %[A] Upper limit of speed loop
SL_LowerLimit = -2; %[A] Lower limit of speed loop
% Speed PI Controller Constants (manual settings)
SL_Kp = 0.0008; %[-] Speed controller proportional constant in time domain
SL_Ki = 0.0009; %[-] Speed controller integration constant in time domain
Manual_Constant_Tuning = 0; %[-] Switch between manual or automatic speed constant tunning
%%%%%%%%%%%%%%%%%%%%%%
% Sensorless - SNSLS %
%%%%%%%%%%%%%%%%%%%%%%
%BEMF Observer Parameters
SNSLS_BemfObsrvF0 = 280; %[Hz] BEMF DQ observer bandwidth
SNSLS_BemfObsrvKsi = 1; %[-] BEMF DQ observer attenuation
%Tracking Observer Parameters
SNSLS_TrackObsrvF0 = 25; %[Hz] Tracking observer bandwidth
SNSLS_TrackObsrvKsi = 1; %[-] Tracking observer attenuation
%Open Loop Start-up Parameters
SNSLS_StartupRamp = 1500; %[rpm/sec] Open loop start-up ramp increment up
SNSLS_StartupCurrent = 0.2; %[A] Open loop start-up current
SNSLS_MergingSpeed = 500; %[rpm] Speed where algorithm switches from open to closed loop
SNSLS_MergingCoeff = 100; %[%] Position weight merging coefficient
%Base constants - do not modify
k_factor = 100; %Scalar factor
UdcbIIRf0 = 100;
IIRxCoefsScaleType = 8;
UmaxCoeff=1.732050807568877;
DiscMethodFactor = 2.0; %Trapezoidal
ERRmax = 1;
Wmax = 2*pi*pp*N_max/60;
E_block_per = 2000;
CALIB_T=0.2;
FAULT_T=3;
FREEWHEEL_T=1;
```

8.3.2 Macro calculation procedure

```
M1_MOTOR_PP = pp
```

M1_I_MAX = I_max

M1_U_DCB_MAX = U_DCB_max

M1_U_MAX = Uph_nom

M1_N_MAX = N_max

FREQ_MAX = N_max/60*pp

M1_E_MAX = E_max

M1_U_DCB_TRIP = U_DCB_trip/U_DCB_max

M1_U_DCB_UNDERVOLTAGE = U_DCB_under/U_DCB_max

M1_U_DCB_OVERVOLTAGE = U_DCB_over/U_DCB_max

M1_N_OVERSPEED = N_over/N_max

M1_N_MIN = N_min/N_max

M1_N_NOM = N_nom/N_max

M1_I_PH_NOM = Iph_nom/I_max

M1_UDCB_IIR_B0 = 4/IIRxCoefsScaleType * (2 * pi * UdcbIIRf0 * CL_SampleTime)

/ (2 + (2 * pi * UdcbIIRf0 * CL_SampleTime))

M1_UDCB_IIR_B1 = 4/IIRxCoefsScaleType * (2 * pi * UdcbIIRf0 * CL_SampleTime)

/ (2 + (2 * pi * UdcbIIRf0 * CL_SampleTime))

M1_UDCB_IIR_A1 = 4/IIRxCoefsScaleType * (-(2 * pi * UdcbIIRf0 * CL_SampleTime - 2)

/ (2 + (2 * pi * UdcbIIRf0 * CL_SampleTime)))

M1_ALIGN_DURATION = Align_duration/CL_SampleTime

M1_ALIGN_VOLTAGE = Align_voltage/Uph_nom

M1_FAULT_DURATION = FAULT_T/SL_SampleTime

M1_FREEWHEEL_DURATION = FREEWHEEL_T/SL_SampleTime

M1_E_BLOCK_TRH = E_block/E_max

M1_E_BLOCK_PER = E_block_per

M1_CLOOP_SAMPLE_TIME = CL_SampleTime

M1_CLOOP_LIMIT = CL_OutputLimit/100

M1_D_KP_GAIN = ((2 * CL_Ksi * 2 * pi * CL_F0 * Ld) - Rs) * I_max/Uph_nom

M1_D_KI_GAIN = ((2 * pi * CL_F0)^2 * Ld * CL_SampleTime / DiscMethodFactor)

* I_max/Uph_nom

M1_Q_KP_GAIN = ((2 * CL_Ksi * 2 * pi * CL_F0 * Lq) - Rs) * I_max/Uph_nom

M1_Q_KI_GAIN = ((2 * pi * CL_F0)^2 * Lq * CL_SampleTime / DiscMethodFactor)

* I_max/Uph_nom

M1_SLOOP_SAMPLE_TIME = SL_SampleTime

if(Manual_Constant_Tuning == 1)

M1_SPEED_PI_PROP_GAIN = SL_Kp * (2 * pi* pp * N_max)/(60 * I_max)

M1_SPEED_PI_INTEG_GAIN = SL_Ki * SL_SampleTime * (2 * pi* pp * N_max)

/(60 * I_max)

```
else
M1_SPEED_PI_PROP_GAIN = (2 * pi / 60)*(4 * SL_Ksi * pi * SL_F0 * J / kt)
* pp * N_max/I_max
M1_SPEED_PI_INTEG_GAIN= (2 * pi / 60)*((2 * pi * SL_F0)^2 * SL_SampleTime
* J / kt) * pp * N_max/I_max
end
M1_SPEED_LOOP_HIGH_LIMIT = SL_UpperLimit/I_max
M1_SPEED_LOOP_LOW_LIMIT = SL_LowerLimit/I_max
M1_SPEED_RAMP_UP = SL_IncUp*SL_SampleTime/N_max
M1_SPEED_RAMP_DOWN = SL_IncDown*SL_SampleTime/N_max
M1_SPEED_IIR_B0 = 4/IIRxCoefsScaleType * (2*pi*SL_CutOffFreq*SL_SampleTime)
/(2+(2*pi*SL_CutOffFreq*SL_SampleTime))
M1_SPEED_IIR_B1 = 4/IIRxCoefsScaleType * (2*pi*SL_CutOffFreq*SL_SampleTime)
/(2+(2*pi*SL_CutOffFreq*SL_SampleTime))
M1_SPEED_IIR_A1 = -4/IIRxCoefsScaleType * (2*pi*SL_CutOffFreq*SL_SampleTime
-2)/(2+(2*pi*SL_CutOffFreq*SL_SampleTime))
M1_I_SCALE = (Ld / (Ld + CL_SampleTime * Rs))
M1_U_SCALE = (CL_SampleTime / (Ld + CL_SampleTime * Rs)) * Uph_nom/I_max
M1_E_SCALE = (CL_SampleTime / (Ld + CL_SampleTime * Rs)) * E_max/I_max
M1_WI_SCALE = (2 * pi / 60)*(Lq * CL_SampleTime / (Ld + CL_SampleTime * Rs))
* pp * N_max
M1_BEMF_DQ_KP_GAIN = ((2 * SNSLS_BemfObsrvKsi * 2 * pi * SNSLS_BemfObsrvF0 *
Ld - Rs)) * I_max/E_max
M1_BEMF_DQ_KI_GAIN = (Ld * (2 * pi * SNSLS_BemfObsrvF0)^ 2 * CL_SampleTime)
* I_max/E_max
TO_Kps = 2*SNSLS_TrackObsrvKsi*2*pi*SNSLS_TrackObsrvF0*(ERRmax/Wmax);
TO_Kis = ((2*pi*SNSLS_TrackObsrvF0)^2)*CL_SampleTime*(ERRmax/Wmax);
TO_Kpz = TO_Kps;
TO_Kiz = TO_Kis*CL_SampleTime;
TO_Kpz_f = TO_Kpz*(ERRmax/Wmax);
TO_Kiz_f = TO_Kiz*(ERRmax/Wmax);
if(TO_Kpz_f<1)
M1_TO_KP_SHIFT = -ceil(log(abs(1/TO_Kpz_f))/log(2)-1)
else
M1_TO_KP_SHIFT = ceil(log(abs(TO_Kpz_f))/log(2))
end
M1_TO_KP_GAIN = round(TO_Kpz_f*2^(-M1_TO_KP_SHIFT)*1000000000000)/1000000000000
if(TO_Kiz_f<1)
```

```
M1_TO_KI_SHIFT = -ceil(log(abs(1/TO_Kiz_f))/log(2)-1)
else
M1_TO_KI_SHIFT = ceil(log(abs(TO_Kiz_f))/log(2))
end
M1_TO_KI_GAIN = round(TO_Kiz_f*2^(-M1_TO_KI_SHIFT)*1000000000000)/1000000000000
TO_Theta_f = CL_SampleTime*Wmax/pi;
if(TO_Theta_f<1)
M1_TO_THETA_SHIFT = -ceil(log(abs(1/TO_Theta_f))/log(2)-1)
else
M1_TO_THETA_SHIFT = ceil(log(abs(TO_Theta_f))/log(2))
end
M1_TO_THETA_GAIN = round(TO_Theta_f*2^(-M1_TO_THETA_SHIFT)
*1000000000000)/1000000000000
TO_W_IIR_cutoff_freq = 1 / (2 * SL_SampleTime) * 0.8;
TO_W_IIR_B0_fl = (2*pi*TO_W_IIR_cutoff_freq*CL_SampleTime)
/(2+(2*pi*TO_W_IIR_cutoff_freq*CL_SampleTime));
TO_W_IIR_B1_fl = (2*pi*TO_W_IIR_cutoff_freq*CL_SampleTime)/
(2+(2*pi*TO_W_IIR_cutoff_freq*CL_SampleTime));
TO_W_IIR_A1_fl = (2*pi*TO_W_IIR_cutoff_freq*CL_SampleTime-2)
/(2+(2*pi*TO_W_IIR_cutoff_freq*CL_SampleTime));
M1_TO_SPEED_IIR_B0 = 4.0*TO_W_IIR_B0_fl/IIRxCoefsScaleType
M1_TO_SPEED_IIR_B1 = 4.0*TO_W_IIR_B1_fl/IIRxCoefsScaleType
M1_TO_SPEED_IIR_A1 = -4.0*TO_W_IIR_A1_fl/IIRxCoefsScaleType
M1_OL_START_RAMP_INC = SNSLS_StartupRamp/60*pp*2*pi/Wmax*CL_SampleTime
M1_OL_START_I = SNSLS_StartupCurrent/I_max
M1_MERG_SPEED_TRH = (SNSLS_MergingSpeed / N_max)
M1_MERG_COEFF = ((SNSLS_MergingCoeff / 100) * (60 / (pp * SNSLS_MergingSpeed))
/ CL_SampleTime / 2 / 32768)
k_rate_gain = Uph_nom*k_factor/100/(N_nom*pp*2*pi/60);
k_rate_sc = k_rate_gain*Wmax/(U_DCB_max/UmaxCoeff);
if(k_rate_sc == 1.000000000000)
k_rate_sc = k_rate_sc + 0.0000001;
end
if(k_rate_sc >1)
SCALAR_VHZ_FACTOR_SHIFT = ceil(log(abs(k_rate_sc))/log(2))
else
SCALAR_VHZ_FACTOR_SHIFT = 0
end
```

M1_SCALAR_VHZ_FACTOR_GAIN = `Uph_nom*k_factor/100/(N_nom*pp/60)`

M1_SCALAR_INTEG_GAIN = `(2*pi*pp*N_max/60*CL_SampleTime)/pi`

M1_SCALAR_RAMP_UP = `SL_IncUp/60*pp*2*pi/Wmax*CL_SampleTime`

M1_SCALAR_RAMP_DOWN = `SL_IncDown/60*pp*2*pi/Wmax*CL_SampleTime`

Chapter 9 Acronyms

Table 6. Acronyms

Acronym	Meaning
ADC	ADC Analog-to-Digital Converter
AN	AN Application Note
AR	After-reset
BEMF	Back Electromotive Force
BR	Background
CCM	CCM Clock Controller Module
CFCSS	Control Flow Checking by Software Signatures
CPU	CPU Central Processing Unit
CRC	Cyclic Redundancy Check
DAC	Digital-to-Analog Converter
DC	DC Direct Current
DRM	DRM Design Reference Manual
FL	Fast Loop
FLL	Frequency Locked-Loop
FMEA	Failure Mode and Effect Analysis
FMSTR	FreeMASTER
FOC	FOC Field-Oriented Control
FS	Functional Safety
FTM	FlexTimer
GPIO	GPIO General-Purpose Input/Output
GUI	Graphical User Interface
HVAC	Heating, Ventilation, and Air Conditioning
HW	Hardware

Table continues on the next page...

Table 6. Acronyms (continued)

Acronym	Meaning
IDE	Integrated Development Environment
LPIT	LPIT Low-power Periodic Interrupt Timer
LPTMR	Low-Power Timer
LPUART	LPUART Low-power Universal Asynchronous
LUT	Look-up Table
MC	Motor Control
MCAT	MCAT Motor Control Application Tuning
MCAT	Motor Control Application Tool
MCDRV	MCDRV Motor Control Peripheral Drivers
MCU	MCU Microcontroller
PDB	PDB Programmable Delay Block
PDB	Programmable Delay timer
PI	PI Proportional Integral controller
PLL	PLL Phase-Locked Loop
PMSM	PMSM Permanent Magnet Synchronous
PWM	PWM Pulse-Width Modulation
SL	Slow Loop
SW	Software
TMR	TMR Quad Timer
USB	USB Universal Serial Bus
USB	Universal Serial Bus
V/F	Volte-per-Hertz (scalar) control
XBAR	XBAR Inter-Peripheral Crossbar Switch

Chapter 10
List of symbols

Table 7. List of symbols

Symbol	Unit	Description
B	Nm·s/rad	Viscous friction coefficient.
D_{abc}	%	Three-phase PWM duty cycles.
f	Hz	Frequency.
f_{PWM}	Hz	PWM frequency.
f_{ctrl}	Hz	The *pwm_in_mcu* signal frequency.
i_{abc}	A	Three-phase stator currents.
I_{dcb}	A	Inverter DC-bus current.
$i_{\alpha\beta}$	A	Stator current vector in the two-phase stator coordinate frame.
i_{α}	A	Stator current in the α-axis.
i_{β}	A	Stator current in the β-axis.
i_{dq}	A	Stator current in the two-phase rotating synchronous coordinate frame.
i_{d}	A	Stator current in the direct axis.
i_{q}	A	Stator current in the quadrature axis.
J	Kgm^2	Rotor moment of inertia.
k	-	Discrete sample number.
L_{d}	H	Direct axis inductance.
L_{q}	H	Quadrature axis inductance.
N_{e}	rpm	Electrical rotor speed.
N_{m}	rpm	Mechanical rotor speed.
N_{smpl}	-	Number of analog samples acquired per $\mathrm{T_s}$
p	-	Laplace operator.
P	W	Real inverter electrical input power.
P_{p}	-	Number of machine pole pairs.

Table continues on the next page...

Table 7. List of symbols (continued)

Symbol	Unit	Description
P_{out}	W	Output mechanical power.
R	Ω	Stator resistance.
t	s	Time.
T_{align}	s	The alignment duration.
T_{fault}	s	Fault/error recovery time.
T_{free}	s	The rotor deceleration wait time.
$T_{\mathrm{fs\_ctrl}}$	s	The software fault/error control response time.
$T_{\mathrm{fs\_det}}$	s	The software fault/error detection time.
T_{DT}	s	PWM dead-time.
T_{e}	Nm	Machine electrical output torque.
T_{l}	Nm	Load torque.
T_{s}	s	Fast (current) control loop sampling period.
$T_{\mathrm{s\text{-}slow}}$	s	Slow (speed) control loop sampling period.
U_{dcb}	V	Inverter DC-bus voltage.
$u_{\mathrm{\alpha\beta}}$	V	Stator voltage vector in the two-phase stator coordinate frame.
$u_{\mathrm{\alpha}}$	V	Stator voltage in the α-axis.
$u_{\mathrm{\beta}}$	V	Stator voltage in the β-axis.
u_{dq}	V	Stator voltage vector in the two-phase rotating synchronous frame.
u_{d}	V	Stator voltage in the direct axis
u_{q}	V	Stator voltage in the quadrature axis
$\Psi_{\alpha\beta}$	Wb	Stator flux vector in the two-phase stator coordinate frame
$\Psi_{\mathrm{\alpha}}$	Wb	Stator flux in the α-axis
$\Psi_{\mathrm{\beta}}$	Wb	Stator flux in the β-axis
Ψ_{dq}	Wb	Stator flux in the two-phase rotating synchronous coordinate frame
Ψ_{d}	Wb	Stator flux in the direct axis

Table continues on the next page...

Table 7. List of symbols (continued)

Symbol	Unit	Description
Ψ_q	Wb	Stator flux in the quadrature axis
Ψ_{PM}	Wb	Permanent magnet stator flux
ω_e	rad/s	Electrical rotor angular velocity
ω_m	rad/s	Mechanical rotor angular velocity

Chapter 11
Useful links

[1] *Sensorless PMSM Field-Oriented Control* (Design Reference Manual DRM148)

[2] MCUXpresso SDK for Motor Control

Chapter 12
Referenced documents

[1] *Requirement Specification and Risks (RS),* rev 1.0.0. Internal NXP document.

[2] *Motor-Control Pump Reference Design Confluence page*. Internal NXP document.

[3] *KV10 Sub-Family Reference Manual*, rev 7. (Document number KV10P48M75RM).

[4] *Kinetis CM0+ Safety Example, Rev 3.* (Document number IEC60730BKCM0EUG)

[5] S. S. Badini and V. Verma, "A New Stator Resistance Estimation Technique for Vector-Controlled PMSM Drive," in *IEEE Transactions on Industry Applications*, vol. 56, no. 6, pp. 6536-6545, Nov.-Dec. 2020, doi: 10.1109/TIA.2020.3025265.

Chapter 13
Revision history

Table 8. Revision history

Revision number	Date	Substantive changes
0	31 May 2022	Initial release.

Chapter 14
Failure mode and effect analysis

Table 9. Failure mode and effect analysis

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
IPM PWM generation	M1_PWM_PERIPH(15) → PWM_AT signal → FSB5060B	-M1_PWM_PERIPH output drives always LOW -PWM_AT signal is always LOW -Transistor impedance is always HIGH	SYS.E.MOT might overheat due to unpowered *Phase_A*	-M1.PWM issue -M1_PWM_PERIPH issue -PWM_AT signal stuck-at LOW -Transistor stuck OPEN -Boot-strap circuit problem	The phase-loss detection algorithm **M1.DIAG.PHLOSS** to be implemented (analysis of *idcb_rc* signal).
		-M1_PWM_PERIPH output drives always HIGH -PWM_AT signal is always HIGH -Transistor impedance is always LOW	FSB5060B might overheat due to short of DC-bus	-M1.PWM issue -M1_PWM_PERIPH issue -PWM_AT signal stuck-at HIGH -Transistor stuck SHORT	Over-current detection algorithms **M1.DIAG.HWOC+ M1.DIAG.SWOC** to be implemented (analysis of *idcb_rc* signal).
		-M1_PWM_PERIPH drives PWM_AT output HIGH when PWM_BT is driven HIGH too (insufficient deadtime T_{DT}).	FSB5060B might overheat	-M1.PWM issue -M1_PWM_PERIPH issue	The FSB5060B IPM over-temperature detection algorithm **M1.DIAG.TMP_IPM** to be implemented.
	M1_PWM_PERIPH(48) → PWM_AB signal → FSB5060B	-M1_PWM_PERIPH output drives always LOW -PWM_BT signal is always LOW -Transistor impedance is always HIGH	FSB5060B might overheat due to short of DC-bus	-M1.PWM issue -M1_PWM_PERIPH issue -PWM_BT signal stuck-at LOW -Transistor stuck OPEN	The phase-loss detection algorithm **M1.DIAG.PHLOSS** to be implemented (analysis of *idcb_rc* signal).
		-M1_PWM_PERIPH output drives always HIGH -PWM_AT signal is always HIGH	FSB5060B might overheat due to short of DC-bus	-M1.PWM issue -M1_PWM_PERIPH issue	Over-current detection algorithms **M1.DIAG.HWOC+ M1.DIAG.SWOC** to

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
		-Transistor impedance is always LOW		-PWM_BT signal stuck-at HIGH -Transistor stuck SHORT	be implemented (analysis of *idcb_rc* signal).
	M1_PWM_PERIPH(38) → PWM_BT signal → FSB5060B	See PWM_AT& PWM_AB analysis above.			
	M1_PWM_PERIPH(37) → PWM_BB signal → FSB5060B				
	M1_PWM_PERIPH(45) → PWM_CT signal → FSB5060B				
	M1_PWM_PERIPH(46) → PWM_CB signal → FSB5060B				
Input stage and DC-bus	F1→ D2,D3,D5,D6 → C6	DC-bus voltage is LOW	-Cannot generate PWM due to missing 15V -Cannot generate necessary stator voltage	-Fuse F1 his OPEN -D2,D3,D5, or D6 OPEN -LOW or OPEN DC-bus capacity C6	-Under-voltage diagnostics **M1.DIAG.UV_OV**
		DC-bus voltage is NEGATIVE	-C6 capacitor destroyed	-D2,D3,D5, or D6 SHORTED	Cannot be covered by SW.
		DC-bus voltage is HIGH	-D2,D3,D5,D6, or C6 capacitor destroyed when surge current occurs	-Varistor RV1 OPEN	Cannot be covered by SW.

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
			-C6 capacitor destroyed	-Motor SYS.E.MOT braking	-Over-voltage diagnostics M1.DIAG.UV_OV
		Input current is HIGH	-Fuse F1 becomes OPEN	-Varistor RV1 is SHORT	Not safety related
Relay control	GPIO(16) → RELAY	Inrush circuit impedance is always LOW	In-rush current	-Relay K1 stuck ON. -Signal REALY is stuck at HIGH. -Thermistor RT1 is shorted	Not safety related
		In-rush current limiting NTC thermistor RT1 won be bypassed	-Thermistor RT1 overheating. -Increased output impedance of DC-bus (wont trigger OC)	-Relay K1 stuck OFF. -Signal REALY is stuck at LOW.	Not safety related
		Inrush circuit impedance is always HIGH	DC-bus won’t be powered	-Thermistor RT1 has HIGH-Z.	Not safety related
Digital voltage source	+15V → +3.3V → VDD1(1),VDD2(22)	Voltage is LOW	The program execution might be corrupted.	-+3.3V power circuit supply issue -+3.3VA disconnected from MCU pins	-Internal MCU brown-out detector to be properly enabled.
		Voltage is HIGH	MCU might be damaged	-3V3 power circuit supply issue	Cannot be covered by SW.
		Voltage drifts	N/A	N/A	Not safety related
		Voltage raise is SLOW	The program execution might be corrupted.	-/RESET signal not LOW on power-up (C20 disconnected) -3V3 power circuit supply issue	-Internal MCU brown-out detector to be properly enabled.

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
Analog voltage source	+15V → +3.3VA → VREFH(10), VDDA(9)	Voltage is LOW	The program execution might be corrupted.	-+3.3V power circuit supply issue -+3.3VA disconnected from MCU pins	-Internal MCU brown-out detector to be properly enabled.
		Voltage is HIGH	MCU might be damaged	-+3.3V power circuit supply issue	Cannot be covered by SW.
		Voltage drifts	Analog measurements drift too and no longer math physical quantities	-All algorithms relying on analog measurements might not provide correct results	-A stable voltage band gap 1.0V reference to be measured and compared to expected value (FS.REF).
		Voltage raise is SLOW	N/A	N/A	Not safety related
15V power source	DCB_Pos → +15V	Voltage is LOW	IPM cannot generate PWM	-Capacitor C6 OPEN -15V branch current consumption HIGH	The phase-loss detection algorithm M1.DIAG.PHLOSS to be implemented (analysis of *idcb_rc* signal).
			IPM cannot generate PWM	-DC-bus voltage too LOW	-Under-voltage diagnostics M1.DIAG.UV_OV
		Voltage is HIGH	-3.3V digital source is destroyed	-15V IC U5 is shorted or malfunctioning	Cannot be covered by SW.
		Voltage drifts	N/A	N/A	Not safety related
		Voltage raise is SLOW	N/A	N/A	Not safety related
System Control Interface (SYS.I.CTRL)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)

Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
External command	*pwm_in_mcu*(28) → APP_EXTCMD_PERIPH	-The *pwm_in_mcu* is permanently HIGH -APP_EXTCMD_PERIPH reports high frequency f_{ctrl}	Inverter is not powered	Signal *pwm_in_mcu* stuck at HIGH (R44 LOW impedance) or APP_EXTCMD_PERIPH issue	Not safety related
		-The *pwm_in_mcu* is permanently LOW -APP_EXTCMD_PERIPH reports low/none frequency f_{ctrl}	Inverter is not powered	Signal *pwm_in_mcu* stuck at LOW (U3 output short to GND) or APP_EXTCMD_PERIPH issue	Not safety related
		-The APP_EXTCMD_PERIPH incorrectly reports valid *pwm_in_mcu* f_{ctrl} frequency	Inverter is powered when it should not (*Phase_A*, *Phase_B*, or *Phase_C* signals become live)	The APP_EXTCMD_PERIPH issue.	The **M1.DIAG.EXTCMD** safety mechanism to be implemented.

System Debug Interface (SYS.I.DBG)

Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
	SWCLK → SWD_CLK(17), SWDIO → SWD_DIO(20),	Signal unwanted HIGH or LOW	N/A	N/A	Not safety related
	/RESET → RESET(26),	Signal unwanted HIGH or LOW	N/A	N/A	Not safety related

Electronic Pump Control Unit (SYS.E.EPCU)

Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
Debug MCU pins	GPIO(41) → dbg_1 (TP19)	Signal unwanted HIGH or LOW			Not safety related

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
	GPIO(42) → dbg_2 (TP20)	Signal unwanted HIGH or LOW			Not safety related
	GPIO(18) → user_led	Signal unwanted HIGH or LOW	Incorrect LED signaling	Signal or GPIO stuck-at LO or HI	Not safety related
		Signal at HI-Z	LED wont enable	Disconnected MCU pin	Not safety related
	GPIO(14) → dacout (TP21)	Signal unwanted HIGH or LOW	N/A	N/A	Not safety related
Misc. MCU pins	(21) → NMI	-Signal unwanted LOW	MCU enters NMI ISR	Signal stuck-at LO	NMI ISR to be safely handled.
		-Signal at HI-Z	None		Not safety related
	GPIO(6, 7, 8, 24, 25, 29, 31, 32, 33, 36, 40, 43, 44) → Any GND	Signal unwanted strong output HIGH	MCU might overheat, causing incorrect program execution	-Unwanted write to GPIO caused by corrupted software execution.	-The MCU over-temperature detection algorithm **M1.DIAG.TMP_MCU** to be implemented. -See SW part of FMEA.
		Signal unwanted strong output LOW	N/A	N/A	Not safety related
Analog Interface	*ipm_temp_rc* → ADC0_SE8,ADC1_SE8	Voltage at pin does not match physical quantity value	IPM overheats, but program does not react	- FSB5060B thermal sensor damaged -*ipm_temp_rc* signal shorted to other signal -*ipm_temp_rc* signal disconnected from pin	-Measured signal range check **FS.REF**
	medium_temp_rc → ADC1_SE2	Voltage at pin does not match physical quantity value	Pump motor overheats, but program does not react	-Pump/medium thermal sensor damaged -*medium_temp_rc* signal shorted to other signal	-Measured signal range check **FS.REF** -Motor stator resistance check **M1.DIAG.RES**

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
				-*medium_temp_rc* signal disconnected from pin	
	vdcb_rc → ADC1_SE6	Voltage at pin does not match physical quantity value	-Generated PWM no-longer valid	-R3, R7, R9, or R13 resistance changed -*vdcb_rc* signal shorted to other signal -*vdcb_rc* signal disconnected from pin	-Measured signal range check **FS.REF** -Motor stator resistance check **M1.DIAG.RES** -Under and over-voltage range diagnostics **M1.DIAG.UV_OV** -Motor input power vs speed range check **M1.DIAG.UP_OP**
		-Voltage value above MCU maximum	MCU destroyed	-R3,R7,R9 shorted or bypassed	-MCU destroyed. Cannot be covered by SW.
	idcb_rc → ADC1_SE3, ADC0_SE11	Voltage at pin does not match physical quantity value	-Number of diagnostic algorithms not able to operate correctly -Unnecessarily large currents generated -Control algorithm instability	-R52 resistance changed or bypassed -*idcb_rc* signal shorted to other signal -*idcb_rc* signal disconnected from pin	-Measured signal offset range check **FS.REF** during zero-vector V7(111) -Motor stator resistance check **M1.DIAG.RES** -Over-current diagnostics **M1.DIAG.SWOC** -Motor input power vs speed range check **M1.DIAG.UP_OP** -Motor input-power stability check **M1.DIAG.PWR_STAB**
	vdcb_rc → CMP1_IN5,CMP0_IN5	Voltage at pin does not match physical quantity value	N/A	N/A	Not safety related

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
	idcb_rc → ADC1_SE3, ADC0_SE11	Voltage at pin does not match physical quantity value	-Fast hardware over-current FS.HWOC protection no longer operational (IPM might be damaged by fast current spikes)	-*idcb_rc* signal disconnected from pin	Software-based over-current protection **M1.DIAG.SWOC** to be implemented
Internal MCU Hardware					
Module	**Element**	**Failure mode**	**Failure effect**	**Root cause**	**SW safety mechanism**
Memories	Flash	-Incorrect safety-related data/ instructions are read/accessed	-Program cannot be safely executed	-Memory cell STUCK-AT -Address bus STUCK-AT -Data-bus STUCK-AT	-Flash test **FS.FLASH** to be implemented -Program flow check **FS.FLOW** to be implemented -Hard-fault to be safely handled (**FS.ISR**)
	RAM	-Incorrect safety-related data are read/accessed	-Program cannot be safely executed	-Memory cell STUCK-AT -Address bus STUCK-AT -Data-bus STUCK-AT	-RAM test **FS.RAM** to be implemented -Program flow check **FS.FLOW** to be implemented -Hard-fault to be safely handled (**FS.ISR**)
Clocks	Clocks (PMC,MCG→SIM)	-Clock generator provides unexpected clock value	-Timing fails (routines not executed in time) -Peripherals fail to operate	-Incorrect clock division -Incorrect clock multiplication	-Clock test **FS.CLK** to be implemented

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
			with high clocks	-Incorrect clock routing	-Interrupt rate test **FS.ISR** to be implemented
CPU	Program Counter register	-Incorrect instruction executed	-Program cannot be safely executed	-PC register STUCK-AT	-PC register test (**FS.PC**)
		-Unknown instruction executed	-Program enters hard-fault	-PC register STUCK-AT	-Hard-fault to be safely handled (**FS.ISR**)
	R8-R11, PRIMASK, CONTROL, R0-R7, R12, LR, APSR, MSP, and PSP registers	-CPU result/operation is not correct	-Program cannot be safely executed	-Register STUCK-AT	-Core register test (**FS.CORE**)
			-Program enters hard-fault	-Incorrect instruction	-Hard-fault to be safely handled (**FS.ISR**)
			-Unexpected IRQ executed	-Incorrect ISR processing	-Unexpected IRQ to be safely handled (**FS.ISR**)
Watchdog	WDOG	-Watchdog unable to cause reset	-Corrupted program continuous to run	-Internal WDOG issue (i.e. counter stuck-at)	-Watchdog after-reset Test (**FS.WDOG**)
Trigger chain	M1_PWM_PERIPH → M1_PDB_PERIPH → M1_ADC_PERIPH → M1_DMA_PERIPH	-The TRGF trigger not generated in time -Fast-loop (FL) ISR not executed	-All FL control and safety algorithms not timely executed	-M1_TMR_PERIPH issue (i.e. counter stuck-at)	-Interrupt rate test FS.ISR to be implemented -Watchdog starvation reset (**FS.WDOG**)
		-M1_ADC_PERIPH triggers TRGM1 not acquired at correct time	-Timing sensitive samples (*idcb_rc*) not acquired	-The M1_DMA_TAB_DLY table delays corrupted or not applied to M1_PDB_PERIPH	-The M1_DMA_TAB_DLY table checksum test (**FS.DMA**) -Measured signal offset range check **FS.REF** during zero-vector V7(111)

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
				-M1_PDB_PERIPH issue (i.e. counter stuck-at)	-Motor stator resistance check **M1.DIAG.RES** -Over-current diagnostics **M1.DIAG.SWOC** -Motor input power vs speed range check **M1.DIAG.UP_OP** -Motor input-power stability check **M1.DIAG.PWR_STAB**
	M1_TMR_PERIPH	-The TRGS trigger not generated in time -Slow-loop (SL) ISR not executed	-All SL control and safety algorithms not timely executed	-M1_TMR_PERIPH issue (i.e. counter stuck-at)	-Clock test **FS.CLK** to be implemented -Interrupt rate test **FS.ISR** to be implemented
Analog measurement	PORT → AMUX → M1_ADC_PERIPH → M1_DMA_PERIPH	-Incorrect analog measurement is stored in the M1_DMA_TAB_RSLT	-All diagnostic and safety mechanisms using analog measurements might fail -Incorrect M1 control action	-Incorrect PORT configuration -Incorrect quantity routed via analog multiplexer AMUX -Incorrect M1_ADC_PERIPH conversion result -Analog conversion result corrupted during DMA transfer	-Analog compare test (**FS.CMP**) -Flash test **FS.FLASH** to check M1_DMA_TAB_ACHN table
DMA	M1_DMA_PERIPH	-M1_DMA_PERIPH stores outside of M1_DMA_TAB_RSLT	-Safety related memory gets corrupted	-DMA TCD memory is corrupted (stuck-at) -Address bus stuck-at.	-TCD memory test using pattern (**FS.DMA**) -Checksum calculation of DMA TCDs during $T_{TST\_UI\_MAX}$ (**FS.DMA**)

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
					-DMA error IRQ to be safely handled (**FS.ISR**)
Software					
Element		**Failure mode**	**Failure effect**	**Root cause**	**SW safety mechanism**
Stack		Stack over-flow/under-flow	-Safety related memory gets corrupted	-Corrupted program behavior (i.e. too many function calls)	-Stack under/ overflow pattern check (**FS.STACK**) -Proper separation of safety-related RAM and FLASH memory (**FS.DESIGN**)
Program flow		Program stall	-Safety mechanisms not executed -Motor control action not updated (might cause over-current)	-Infinite loop cycle entered (i.e. incorrect instruction, data or address)	-Watchdog starvation -Hardware over-current diagnostics (**M1.DIAG.HWOC**)
		Incorrect execution order	-Safety mechanisms not executed -Motor control action not updated (might cause over-current)	-Function is skipped (i.e. incorrect instruction, data or address; corrupted non-safety program part)	-Program flow check (**FS.FLOW**) -Safety-related function might be called only by other safety-related function (**FS.DESIGN**) -Proper separation of safety-related RAM and FLASH memory (**FS.DESIGN**)
		Unexpected ISR invoked	-Program flow is corrupted	-Corrupted vector table -Incorrect instruction executed (i.e.	-Unknown ISRs to be safely handled (**FS.ISR**)

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
				corrupted non-safety program part)	-The Flash test **FS.FLASH** to cover vector table as well
		Peripheral or program module initialization fails	-Program flow is corrupted	-Periphery or software module (i.e. motor-control state machine M1.SM) not initialized -M1_ADC_PERIPH calibration failed	-Program flow check (**FS.FLOW**) -Unexpected SW paths (M1_ADC_PERIPH calibration result, initialized M1.SM state,…) to be safely handled (**FS.DESIGN**)
M1.CTRL		Unstable control action	-Some safety mechanisms might not operate properly -Motor might overheat	-Corrupted non-safety program part	-Motor input-power stability check **M1.DIAG.PWR_STAB** -Over-current diagnostics **M1.DIAG.SWOC** -Motor input power vs speed range check **M1.DIAG.UP_OP**
M1.SM		Incorrect speed or position information estimated	-Some safety mechanisms might not operate properly -Motor might overheat	-Motor parameters no longer match (motor issue)	-Motor stator resistance check **M1.DIAG.RES** -Over-current diagnostics **M1.DIAG.SWOC** -Motor input power vs speed range check **M1.DIAG.UP_OP** -Motor input-power stability check **M1.DIAG.PWR_STAB**
Electrical Pump Motor (SYS.E.MOT)					
Element		Failure mode	Failure effect	Root cause	SW safety mechanism

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
Rotor		Blocked rotor	Motor over-heats	Pump rotor blockage	-The blocked-rotor detection algorithm M1.DIAG.BLCKROT to be implemented
		Dry-run	Motor over-heats, excessive bearings stress	Pump medium missing	-Motor input power vs speed range check M1.DIAG.UP_OP (no medium will result in low motor input power)
		Over-speed	Motor damage by over-speed	Incorrect M1.CTRL action	-Motor range speed check M1.DIAG.US_OS
		Under-speed	Estimated rotor speed no longer reliable	Incorrect M1.CTRL action or external command set tool low	-Motor range speed check M1.DIAG.US_OS
		Over-load	Motor over-heats	Pump rotor blockage	-The over-load detection algorithm M1.DIAG.LOAD to be implemented
Stator		Over-voltage	Stator isolation damage	Excessive motor braking	-DC-bus voltage range check M1.DIAG.UV_OV
		Over-current	Stator winding damage	Inverter or M1.CTRL control action issue	-Over-current diagnostics M1.DIAG.SWOC and M1.DIAG.HWOC
		Over-power	Motor over-heats	-Pump medium issue -Damaged bearings -M1.CTRL control action issue	-Motor input power vs speed range check M1.DIAG.UP_OP (no medium will result in low motor input power)
		Over-temperature of motor windings	Motor over-heats	-Pump medium issue -Damaged bearings	-Medium and IPM temperature range to be checked

Table continues on the next page...

Table 9. Failure mode and effect analysis (continued)

Power Stage (SYS.E.PWRSTG)					
Module	Element	Failure mode	Failure effect	Root cause	SW safety mechanism
				-M1.CTRL control action issue	(M1.DIAG.TMP_MED, M1.DIAG.TMP_IPM)
		Disconnected phase	Motor over-heats	-Disconnected *Phase_A*, *Phase_B*, or *Phase_C* signal	-Phase-loss detection algorithm M1.DIAG.PHLOSS -Motor stator resistance check M1.DIAG.RES -Motor input power vs speed range check M1.DIAG.UP_OP -Motor input-power stability check M1.DIAG.PWR_STAB
		Windings shorted	Motor over-heats	Internal stator problem (i.e. windings short)	-Motor stator resistance check M1.DIAG.RES -Over-current diagnostics M1.DIAG.SWOC -Motor input power vs speed range check M1.DIAG.UP_OP -Motor input-power stability check M1.DIAG.PWR_STAB

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 http://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 data sheet 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.

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.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Airfast — is a trademark of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Cadence — the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide.

CodeWarrior — is a trademark of NXP B.V.

ColdFire — is a trademark of NXP B.V.

ColdFire+ — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

EdgeScale — is a trademark of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

FeliCa — is a trademark of Sony Corporation.

Freescale — is a trademark of NXP B.V.

HITAG — is a trademark of NXP B.V.

ICODE and I-CODE — are trademarks of NXP B.V.

Immersiv3D — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

Mantis — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

MOBILEGT — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

Processor Expert — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

SafeAssure — is a trademark of NXP B.V.

SafeAssure — logo is a trademark of NXP B.V.

StarCore — is a trademark of NXP B.V.

Synopsys — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

Tower — is a trademark of NXP B.V.

UCODE — is a trademark of NXP B.V.

VortiQa — is a trademark of NXP B.V.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 31 May 2022
Document identifier: PMSMSAFEUG