Dual Sensorless PMSM Field-Oriented Control With Power Factor Correction on MC56F84789 DSC

Design Reference Manual

Devices Supported:
MC56F84789

Document Number: DRM139
Rev. 0
05/2013

Contents

Section number **Title** **Page**

Chapter 1
Introduction

1.1 Introduction

This document describes the design of a dual 3-phase PMSM vector control drive without position sensor and PFC on single controller. The design is targeted for consumer and industrial applications. This cost-effective, high-efficiency, low-noise, variable-power advanced refrigeration system solution benefits from Freescale's MC56F84789 digital signal controller (DSC) device dedicated for motor control.

1.2 Application features and components

The system is designed to drive two 3-phase PMSMs and PFC. Following are the application features of the system:

- Sensorless 3-phase PMSM speed vector (FOC) control of a compressor
- Sensorless 3-phase PMSM speed vector (FOC) control of a fan
- Interleaved PFC control
- Motor current sensing with three current sensors
- PFC current sensing with one current sensor
- DC bus voltage sensing
- Input voltage sensing
- Temperature sensing at three points
- Based on Freescale's MC56F84789 digital signal controller
- Running on the 3-in-1 power stage with an MC56F84789 daughter board
- Input voltage 90–245 V, 40–70 Hz
- Remote RS-232 control

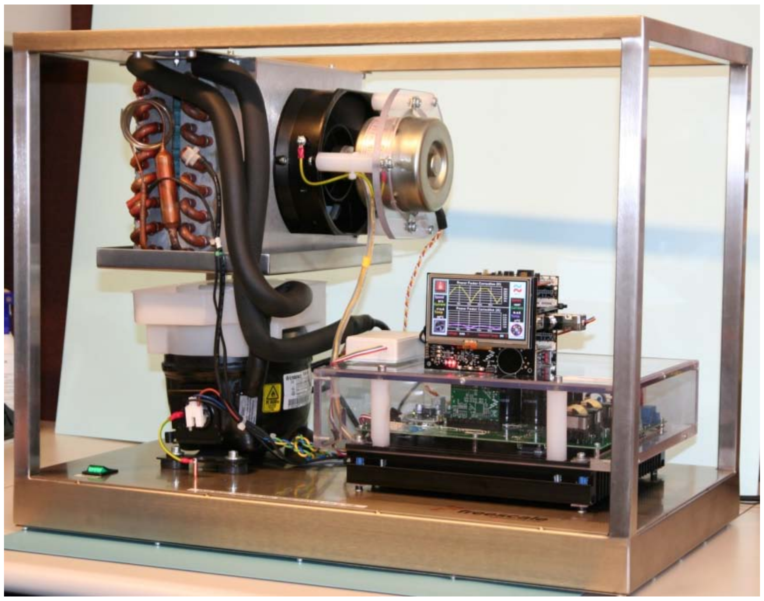


Figure 1. Application system demo

Main application components available for customers are:

- Software—written in C-code using some library algorithms from Freescale's Embedded Software Libraries (FSLESL)
- Hardware—based on Freescale's 3-in-1 power stage
- Documentation—See the References section.

1.3 Sinusoidal PMSM applications overview

Sinusoidal PMSMs are more and more popular for new drives, replacing brushed DC, universal, and other motors in a wide application area. The reason is a better reliability (no brushes), better efficiency, lower acoustic noise, and also other benefits of electronic control. A disadvantage of PM synchronous motor drives might be the need for a more sophisticated electronic circuit. But nowadays, most applications need an electronic speed or torque regulation and other features with the necessity of electronic control. Once a system with electronic control is used, it just takes a small system cost increase to implement more advanced drives like the sinusoidal PM synchronous motor with digitally-controlled switching inverter and DC-bus circuit. It is only necessary to have a cost-effective controller with a good calculation performance. One of them is the Freescale MC56F84789.

The PMSM also has advantages when compared to an AC induction motor. Because a PM synchronous motor achieves higher efficiency by generating the rotor magnetic flux with rotor magnets, it is used in white goods (such as refrigerators, washing machines, dishwashers), pumps, compressors, fans, and other appliances that require a high reliability and efficiency.

The 3-phase synchronous motors with permanent magnets come in two most popular variants: the sinusoidal PMSM and the trapezoidal BLDC motor. The sinusoidal PM synchronous motor is very similar to the trapezoidal BLDC (Electronically-Commuted) motor. There are two main differences:

- Motor construction
  - The shape of BEMF inducted voltage — sinusoidal (PM synchronous motor) versus trapezoidal (BLDC) motor.
- Control — shape of the control voltage
  - 3-phase sinusoidal (all three phases connected at one time) versus rectangular six-step commutation (one phase is non-conducting at any time).

Generally, the performance of sinusoidal PMSM is better due to constant torque, while the trapezoidal BLDC motor can be more easily controlled. The trapezoidal BLDC motors are used mainly for historical reasons. It was easier to create a six-step commutation and estimate the rotor position with simpler algorithms, since one phase is non-conducting. The sinusoidal PM synchronous motors require more sophisticated control, but provide some benefits such as smoother torque, lower acoustic noise, and so on. As shown in this document, the MC56F84789 provides all the necessary functionality for dual 3-phase sinusoidal PM synchronous motor vector control.
Using Freescale MC56F84789 gives a valid reason to replace the trapezoidal BLDC (electronically-commuted) motors with the 3-phase sinusoidal PMSM with almost no system cost increase. The application described here is a speed vector control. The speed-control algorithms can be sorted into following two general groups.

- Scalar control: The constant Volt per Hertz control is a very popular technique representing scalar control.
- Vector- or field-oriented control (FOC): The field-oriented techniques bring overall improvements to drive performance over scalar control such as higher efficiency, full torque control, decoupled control of flux and torque, improved dynamics, and so on.

For the PM synchronous motor control, it is necessary to know the exact rotor position. This application uses special algorithms to estimate the speed and position instead of using a physical position sensor. This design reference manual describes the basic motor theory, the system design concept, hardware implementation, and the software design, including the FreeMASTER software visualization tool.

1.4 Freescale DSC advantages and features

The Freescale's MC56F84789 is well-suited for digital motor control, combining the calculation capability with the features of the MCU on a single chip. These controllers offer many dedicated peripherals such as pulse width modulation (PWM) modules, analog-to-digital converters (ADC), timers, direct memory access (DMA), cross bar units (XBAR), communication peripherals (SCI, SPI, I^{2}C), and onboard Flash and RAM.

MC56F84789 provides the following blocks:

- Core operating at a frequency of 100 MHz
  - Single-cycle 32 × 32-bit multiplier-accumulator (MAC) with 32- or 64-bit result with an optional 32-bit parallel move

- o Single-cycle 16 x 16-bit multiplier-accumulator (MAC) with 16- or 32-bit result with up to two optional 16-bit parallel moves
  - o Four 36-bit accumulators, including extension bits

- Internal Flash 256 KB
- Internal RAM 32 KB
- COP
- Interrupt Controller
- System Integration Module
- 8-channel high-resolution Pulse Width Modulator
- 8-channel Pulse Width Modulator
- Timers
- Two high-speed 12-bit ADCs with up to 16 channels
- One 16-channel 16-bit SAR ADC
- One 12-bit DAC
- Four analog comparators with DAC
- Serial Interface: I2C, UART, SPI, CAN
- 4-channel DMA
- 2x intermodule Crossbar Switch (XBAR)
- And/Or/Invert module
- Ultra low-power operation
- Cyclic Redundancy Check Generator

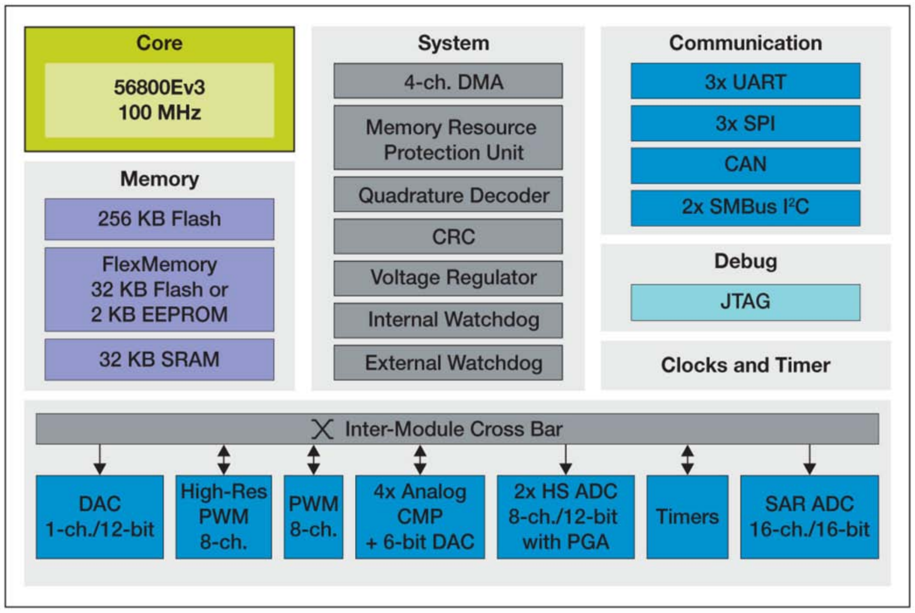


Figure 2. MC56F84789 block diagram

The PMSM vector control and PFC benefit greatly from the PWM modules and ADC. The PWM offers flexibility in its configuration, enabling efficient 3-phase motor or PFC control. The PWM module is capable of generating asymmetric PWM duty cycles in a center-aligned configuration.

The PWM block has the following features:

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- Fractional delay for enhanced resolution of the PWM period and edge placement
- Dithering to simulate enhanced resolution when fine edge placement is not available
- PWM outputs that can operate as complementary pairs or independent channels
- Ability to accept signed numbers for PWM generation
- Independent control of both the edges of each PWM output
- Support for synchronization to external hardware or other PWM
- Double-buffered PWM registers
    - Integral reload rates from 1 to 16
    - Half cycle reload capability
- Multiple output trigger events can be generated per PWM cycle via hardware.
- Support for double switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs.
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values.
- Individual software control for each PWM output
- All outputs can be programmed to change simultaneously via a FORCE_OUT event.
- PWM_X pin can optionally output a third PWM signal from each submodule.
- Channels not used for PWM generation can be used for buffered output compare functions.
- Channels not used for PWM generation can be used for input capture functions.
- Enhanced dual edge capture functionality
- The option to supply the source for each complementary PWM signal pair from any of the following:
    - External digital pin
    - Internal timer channel
    - Crossbar module outputs
    - External ADC input, taking into account values set in ADC high and low limit registers

The 12-bit ADC module has the following features:

- 12-bit resolution
- Designed for maximum ADC clock frequency of 20 MHz with 50 ns period
- Sampling rate up to 6.67 million samples per second
- Single conversion time of 8.5 ADC clock cycles (8.5×50 ns = 425 ns)
- Additional conversion time of 6 ADC clock cycles (6×50 ns = 300 ns)

- Eight conversions in 26.5 ADC clock cycles (26.5 × 50 ns = 1.325 μs) using parallel mode
- Can be synchronized to other peripherals that are connected to an internal Inter-Peripheral Crossbar module, such as the PWM, through the SYNC0/1 input signal
- Sequentially scans and stores up to 16 measurements
- Scans and stores up to eight measurements each on two ADC converters operating simultaneously and in parallel
- Scans and stores up to eight measurements each on two ADC converters operating asynchronously to each other in parallel. A scan can pause and await new SYNC input prior to continuing.
- Gains the input signal by x1, x2, or x4
- Optional interrupts at the end of scan if an out-of-range limit is exceeded or there is a zero crossing
- Optional DMA function to transfer conversion data at the end of a scan or when a sample is ready to be read
- Optional sample correction by subtracting a pre-programmed offset value
- Signed or unsigned result
- Single-ended or differential inputs
- PWM outputs with hysteresis for three of the analog inputs

The 16-bit ADC module has the following features:

- Linear successive approximation algorithm with up to 16-bit resolution
- Up to 24 single-ended external analog inputs
- Output modes:
  - Single-ended 16-bit, 12-bit, 10-bit, and 8-bit modes
- Output in right-justified unsigned format for single-ended
- Single or continuous conversion, that is, automatic return to idle after single conversion
- Configurable sample time and conversion speed/power
- Conversion complete/hardware average complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in low-power modes for lower noise
- Asynchronous clock source for lower noise operation with option to output the clock
- Selectable hardware conversion trigger with hardware channel select
- Automatic compare with interrupt for less-than, greater-than or equal-to, within range, or out-of-range, programmable value
- Temperature sensor
- Hardware average function
- Selectable voltage reference: external or alternate
- Self-calibration mode

The application uses the ADC block synchronized to the PWM pulses. This configuration allows a simultaneous conversion of the required analog values for the inverter currents and DC-bus voltage within the required time.

1.5 References

1. *DSP56800E and DSP56800EX Reference Manual*, DSP56800ERM, Rev. 3, available on **freescale.com**.

2. *MC56F847xx Reference Manual*, MC56F847XXRM, Rev. 1, available on **freescale.com**.

3. *AN4583: MC56F84789 Peripherals Synchronization for Interleaved PFC Control*, available on **freescale.com**.

4. *AN4608: Use of PWM and ADC on MC56F84789 to Drive Dual PMS Motor FOC*, available on **freescale.com**

5. *FreeMASTER Software Users Manual*, **freescale.com/FreeMaster**, by Freescale Semiconductor, Inc.

6. *Freescale's Embedded Software Libraries*, **freescale.com/fslesl**, by Freescale Semiconductor, Inc.

7. *Freescale's Motor Control*, **freescale.com/motorcontrol**, by Freescale Semiconductor, Inc.

For a current list of documentation, visit **freescale.com**.

1.6 Acronyms and abbreviations

Table 1. Acronyms

Term	Meaning
AC	Alternating current
ADC	Analogue-to-digital converter
API	Application interface
FSLESL	Freescale's Embedded Software Libraries
BEMF	Back-electromotive force
BLDC	Brushless DC motor
CCW	Counter-clockwise direction
COP	Computer operating properly (watchdog timer)
CW	Clockwise direction
DAC	Digital-to-analogue converter
DC	Direct current
DMA	Direct memory access module

DRM	Design reference manual
DT	Dead time: a short time that must be inserted between the turning off of one transistor in the inverter half bridge and turning on of the complementary transistor, due to the limited switching speed of the transistors
GPIO	General-purpose input/output
HSCMP	High speed comparators module
I/O	Input/output interfaces between a computer system and the external world—a CPU reads an input to sense the level of an external signal and writes to an output to change the level of an external signal.
ISR	Interrupt Service Routine
I^2C (IIC)	Inter-integrated module
LED	Light-emitting diode
PFC	Power factor correction
DSC	Digital signal controller
PDB	Programmable delay block module
PLL	Phase-locked loop: a clock generator circuit in which a voltage-controlled oscillator produces an oscillation that is synchronized to a reference signal.
PWM	Pulse-width modulation
RPM	Revolutions per minute
XBAR	Crossbar unit
SCI	Serial communication interface module: a module that supports asynchronous communication
SPI	Serial peripheral interface module

1.7 Glossary of symbols

Table 2. Glossary

Term	Definition
d,q	Rotational orthogonal coordinate system
g_ω	Adaptive speed scheme gain
u_γ, u_δ	Alpha/Beta BEMF observer error

i_{sa}, i_{sb}, i_{sc}	Stator currents of the a, b, and c phases
$i_{sd,q}$, $i_s(d,q)$	Stator currents in the d, q coordinate system
i'_Q	First derivative of i_Q current
$i_{s(d,q)*}$, $i_{\gamma,\delta}$	Stator currents in estimated d, q coordinate system
$i_{s\alpha,\beta}$, $i_s(\alpha,\beta)$	Stator currents in α, β coordinate system
$\hat{i}_{sg}$	Stator current space vector in general reference frame
i_{sx}, i_{sy}	Stator current space vector components in general reference frame
$\hat{i}_{rg}$	Rotor current space vector in general reference frame
i_{rx}, i_{ry}	Rotor current space vector components in general reference frame
J	Mechanical inertia
K_M	Motor constant
k_e	BEMF constant
L_s	Stator-phase inductance
L_{sd}, L_D	Stator-phase inductance d axis
L_{sq}, L_Q	Stator-phase inductance q axis
p	Number of poles per phase
R_s	Stator-phase resistance
s	Derivative operator
T_L	Load torque
$u_{S\alpha,\beta}$,$u_{S(\alpha,\beta)}$	Stator voltages in α, β coordinate system
$u_{Sd,q}$, $u_{S(d,q)}$	Stator voltages in d, q coordinate system
a,b	Stator orthogonal coordinate system

$\Psi_{S\alpha,\beta}$	Stator magnetic fluxes in α, β coordinate system
$\Psi_{Sd,q}$	Stator magnetic fluxes in d, q coordinate system
Ψ_M	Rotor magnetic flux
θ_r	Rotor position angle in α, β coordinate system
ω, ω_r, ω_{el}/ ω_F	Electrical rotor angular speed / fields angular speed

1.8 Revision history

Revision	Date	Topics	Substantial changes
0	**05/2013**	**—**	**Initial release**

Chapter 2
Control Theory

2.1 Three-phase PM synchronous motor (PMSM)

The PMSM is a rotating electric machine with a classic 3-phase stator like that of an induction motor; the rotor has surface-mounted permanent magnets (see Figure 3).

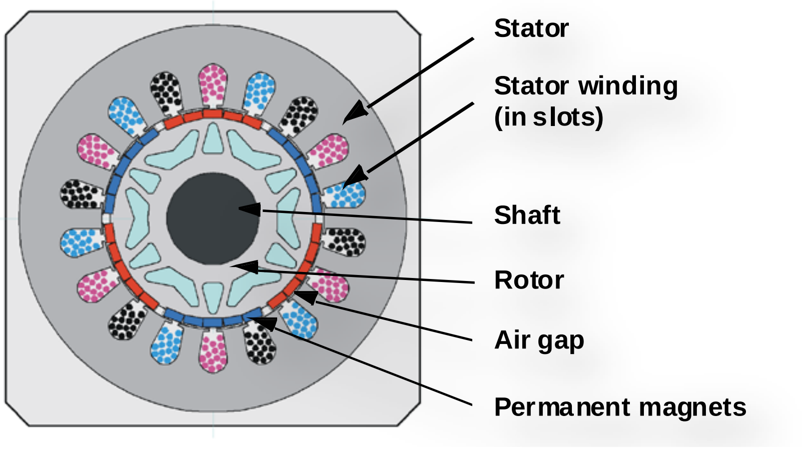


Figure 3. PM synchronous motor cross-section

In this aspect, the PMSM is equivalent to an induction motor, where the air gap magnetic field is produced by a permanent magnet, so the rotor magnetic field is constant. PM synchronous motors offer a number of advantages in designing modern motion control systems. The use of a permanent magnet to generate substantial air gap magnetic flux makes it possible to design highly efficient PM motors.

2.2 Mathematical description of PM synchronous motors

There are a number of PMSM models. The model used for vector control design can be obtained by utilizing space-vector theory. The 3-phase motor quantities (such as voltages, currents, magnetic flux, etc.) are expressed in terms of complex space vectors. Such a model is valid for any instantaneous variation of voltage and current, and adequately describes the performance of the machine under both steady-state and transient operations. Complex space vectors can be described using only two orthogonal axes. Considering the motor as a 2-phase machine motor model reduces the number of equations and simplifies the control design.

2.2.1 Space vector definitions

Assume i_{sa} , i_{sb} , and i_{sc} are the instantaneous balanced 3-phase stator currents:

$$i_{sa} + i_{sb} + i_{sc} = 0$$

Equation 1

Then the stator current space vector can be defined as follows:

$$\overline{i_s} = k(i_{sa} + a i_{sb} + a^2 i_{sc})$$

Equation 2

where a and a^2 are the spatial operators $a = e^{j2\pi/3}$, $a^2 = e^{j4\pi/3}$, and k is the transformation constant and is chosen $k=2/3$. Figure 4 shows the stator current space vector projection.

The space vector defined by Equation 2 can be expressed utilizing the two-axis theory. The real part of the space vector is equal to the instantaneous value of the direct-axis stator current component, $i_{s\alpha}$, and whose imaginary part is equal to the quadrature-axis stator current component, $i_{s\beta}$. Thus, the stator current space vector in the stationary reference frame attached to the stator can be expressed as:

$$\overline{i_s} = i_{s\alpha} + j i_{s\beta}$$

Equation 3

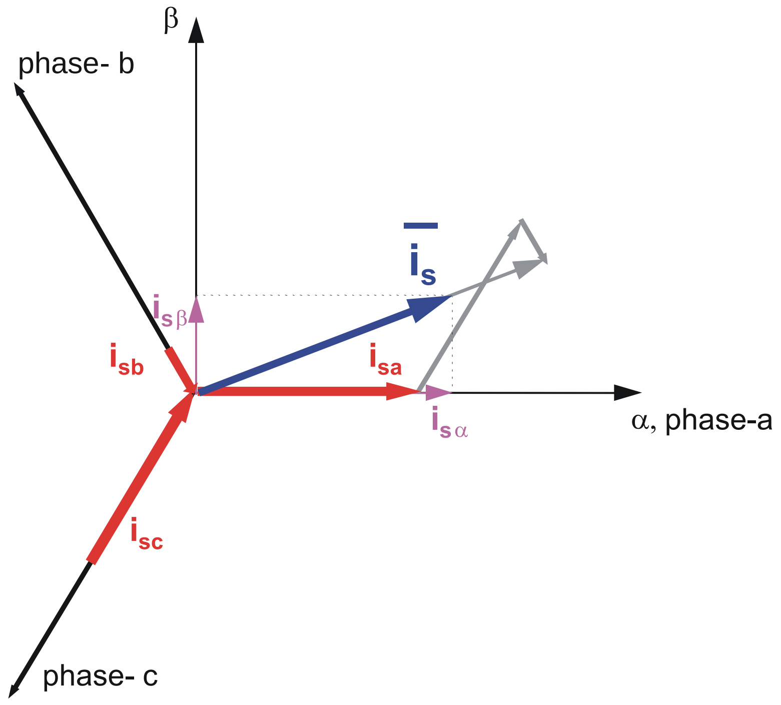


Figure 4. Stator current space vector and its projection

In symmetrical 3-phase machines, the direct and quadrature axis stator currents $i_{s\alpha}$, $i_{s\beta}$ are fictitious quadrature-phase (2-phase) current components, which are related to the actual 3-phase stator currents as follows:

$$i_{s\alpha} = k\left(i_{sa} - \frac{1}{2}i_{sb} - \frac{1}{2}i_{sc}\right)$$

Equation 4

$$i_{s\beta} = k\frac{\sqrt{3}}{2}(i_{sb} - i_{sc})$$

Equation 5

where $k=2/3$ is a transformation constant so that the final equation is:

$$i_{s\beta} = \frac{1}{\sqrt{3}}(i_{sb} - i_{sc})$$

Equation 6

The space vectors of other motor quantities (such as voltages, currents, magnetic fluxes) can be defined in the same way as the stator current space vector.

2.2.2 PM synchronous motor Model

For the description of a PM synchronous motor, the symmetrical three-phase smooth-air-gap machine with sinusoidally-distributed windings is considered. The voltage equations of stator in the instantaneous form can then be expressed as:

$$u_{SA} = R_S i_{SA} + \frac{d}{dt}\psi_{SA}$$

Equation 7

$$u_{SB} = R_S i_{SB} + \frac{d}{dt}\psi_{SB}$$

Equation 8

$$u_{SC} = R_S i_{SC} + \frac{d}{dt}\psi_{SC}$$

Equation 9

where u_{SA}, u_{SB} and u_{SC} are the instantaneous values of stator voltages, i_{SA}, i_{SB} and i_{SC} are the instantaneous values of stator currents, and ψ_{SA}, ψ_{SB}, ψ_{SC} are instantaneous values of stator flux linkages, in phase SA, SB and SC. Due to the large number of equations in the instantaneous form, (Equation 7, Equation 8, and Equation 9), it is more practical to rewrite the instantaneous equations using two axis theory (Clarke transformation). The PM synchronous motor can be expressed as:

$$u_{S\alpha} = R_S i_{S\alpha} + \frac{d}{dt}\psi_{S\alpha}$$

Equation 10

$$u_{S\beta} = R_S i_{S\beta} + \frac{d}{dt}\psi_{S\beta}$$

Equation 11

$$\psi_{S\alpha} = L_{S\alpha} i_{S\alpha} + \psi_M \cos\Theta_r$$

Equation 12

$$\psi_{S\beta} = L_{S\beta} i_{S\beta} + \psi_M \sin\Theta_r$$

Equation 13

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p\left(\psi_{S\alpha} i_{S\beta} - \psi_{S\beta} i_{S\alpha}\right) - T_L\right]$$

Equation 14

For glossary of symbols, see Table 2.

The equations Equation 10 through Equation 14 represent the model of a PM synchronous motor in the stationary frame α, β fixed to the stator.

Besides the stationary reference frame attached to the stator, motor model voltage space vector equations can be formulated in a general reference frame, which rotates at a general speed ω_g. If a general reference frame is used, with direct and quadrature axes x,y rotating at a general instantaneous speed $\omega_g = d\theta_g/dt$, as shown in **Error! Reference source not found.**, where θ_g is the angle between the direct axis of the stationary reference frame (α) attached to the stator and the real axis (x) of the general reference frame, then Equation 15 defines the stator current space vector in general reference frame:

$$\overline{i_{sg}} = \overline{i_s} e^{-j\Theta_g} = i_{sx} + j i_{sy}$$

Equation 15

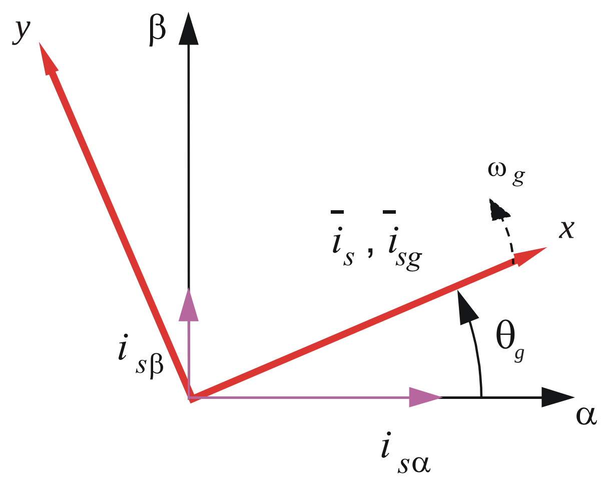


Figure 5. Stator current space vector and its projection

The stator voltage and flux-linkage space vectors can be similarly obtained in the general reference frame.

Similar considerations hold for the space vectors of the rotor voltages, currents and flux linkages. The real axis (rα) of the reference frame attached to the rotor is displaced from the direct axis of the stator reference frame by the rotor angle θ_r. Since it can be seen that the angle between the real axis (x) of the general reference frame and the real axis of the reference frame rotating with the rotor (rα) is θ_g-θ_r, in the general reference frame, the space vector of the rotor currents can be expressed as:

$$\overline{\iota_{rg}} = \overline{\iota_r} e^{-j(\Theta_g - \theta_r)} = i_{rx} + j i_{ry}$$

Equation 16

where $\overline{\iota_r}$ is the space vector of the rotor current in the rotor reference frame.

The space vectors of the rotor voltages and rotor flux linkages in the general reference frame can be similarly expressed.

The motor model voltage equations in the general reference frame can be expressed by utilizing introduced transformations of the motor quantities from one reference frame to the general reference frame. The PM synchronous motor model is often used in vector control algorithms. The aim of vector control is to implement control schemes which produce high dynamic performance and are similar to those used to control DC machines. To achieve this, the reference frames may be aligned with the stator flux-linkage space vector, the rotor flux-linkage space vector or the magnetizing space vector. The most popular reference frame is the reference frame attached to the rotor flux linkage space vector, with direct axis (d) and quadrature axis (q).

After transformation into d, q co-ordinates, the motor model is as follows:

$$u_{Sd} = R_S i_{Sd} + \frac{d}{dt}\psi_{Sd} - \omega_F \psi_{Sq}$$

Equation 17

$$u_{Sq} = R_S i_{Sq} + \frac{d}{dt}\psi_{Sq} + \omega_F \psi_{Sd}$$

Equation 18

$$\psi_{Sd} = L_S i_{Sd} + \psi_M$$

Equation 19

$$\psi_{Sq} = L_S i_{Sq}$$

Equation 20

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p\left(\psi_{Sd} i_{Sd} - \psi_{Sq} i_{Sq}\right) - T_L\right]$$

Equation 21

Considering that below base speed i_{sd}=0, Equation 21 can be reduced to the following form:

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p\left(\psi_M i_{Sq}\right) - T_L\right]$$

Equation 22

From Equation 22, it can be seen that the torque is dependent and can be directly controlled by the current i_{sq} only.

2.3 Vector control of PM synchronous motor

2.3.1 Fundamental principle of vector control

High-performance motor control is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, fast accelerations and decelerations. To achieve such control, vector control techniques are used for three-phase AC motors. The vector control techniques are usually also referred to field-oriented control (FOC). The basic idea of the FOC algorithm is to decompose a stator current into a magnetic field-generating part and a torque-generating part. Both components can be controlled separately after decomposition. The structure of the motor controller is then as simple as that for a separately excited DC motor.

Figure 6 shows the basic structure of the vector control algorithm for the PM synchronous motor. To perform vector control, it is necessary to follow these steps:

1. Measure the motor quantities (phase voltages and currents).

2. Transform them into the 2-phase system (α,β) using Clarke transformation.
3. Calculate the rotor flux space-vector magnitude and position angle.
4. Transform stator currents into the d, q reference frame using Park transformation.
5. The stator current torque (i_{sq}) and flux (i_{sd}) producing components are separately controlled.
6. The output stator voltage space vector is calculated using the decoupling block.
7. The stator voltage space vector is transformed by an inverse Park transformation back from the d, q reference frame into the 2-phase system fixed with the stator.
8. Using space vector modulation, the output 3-phase voltage is generated.

To be able to decompose currents into torque and flux producing components (i_{sd}, i_{sq}), the position of the motor magnetizing flux must be known. This requires accurate rotor position and velocity information to be sensed. Incremental encoders or resolvers attached to the rotor are naturally used as position transducers for vector control drives. In some applications, the use of speed/position sensors is not desirable either. Then, the aim is not to measure the speed/position directly, but to employ some indirect techniques to estimate the rotor position instead. Algorithms which do not employ speed sensors, are called “sensorless control”.

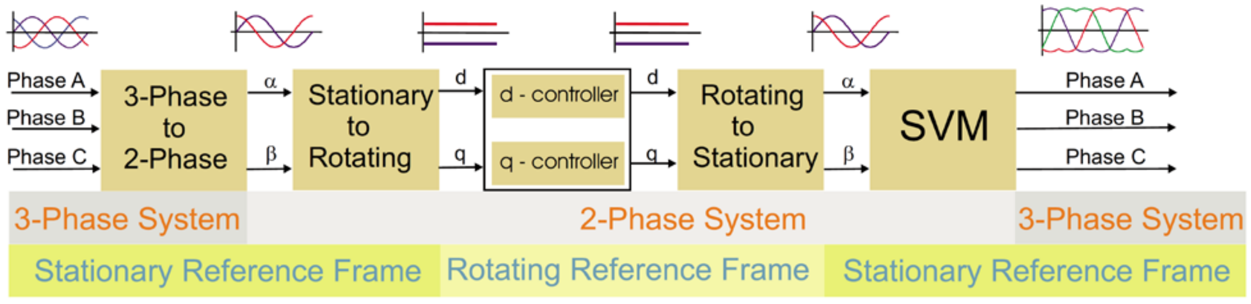


Figure 6. Vector control transformations

2.3.2 Description of the vector control algorithm

The overview block diagram of the implemented control algorithm is illustrated in Figure 7. Similarly, as with other vector control oriented techniques, it is able to control the field and torque of the motor separately. The aim of control is to regulate the motor speed. The speed command value is set by high level control. The algorithm is executed in two control loops. The fast inner control loop is executed with a 100 μs period. The slow outer control loop is executed with a period of one millisecond.

To achieve the goal of the PM synchronous motor control, the algorithm utilizes feedback signals. The essential feedback signals are: three-phase stator current and the stator voltage. For the stator voltage, the regulator output is used. For correct operation, the presented control structure requires a position and speed sensor on the motor shaft or an advanced algorithm to estimate the position and speed.

The fast control loop executes the following two independent current control loops.

- **Direct-axis current (i_{sd}) PI controllers**: The direct-axis current (i_{sd}) is used to control the rotor magnetizing flux.
- **Quadrature-axis current (i_{sq}) PI controllers:** The quadrature-axis current corresponds to the motor torque.

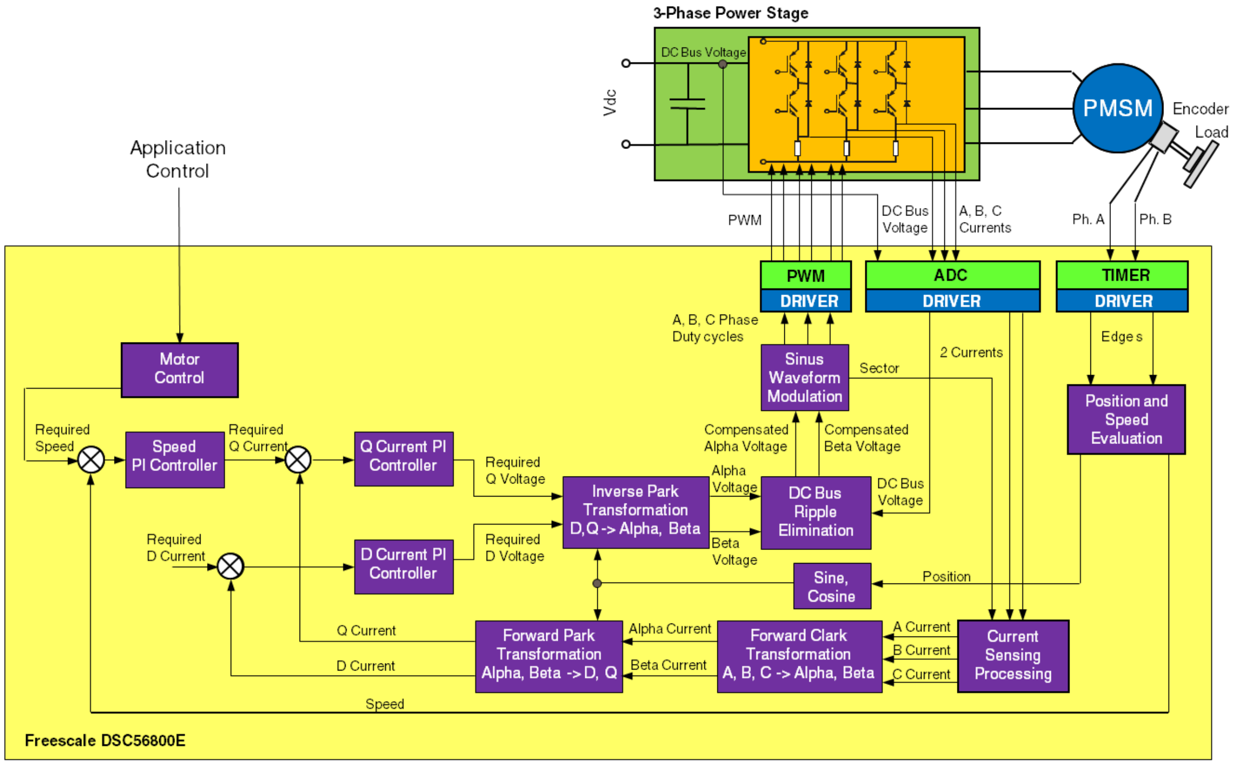


Figure 7. PMSM vector control algorithm overview

The current PI controllers' outputs are summed with the corresponding d- and q-axis components of the decoupling stator voltage to obtain the desired space-vector for the stator voltage, which is applied to the motor. The fast control loop executes all the following necessary tasks to be able to achieve an independent control of the stator current components.

- Three-Phase Current Reconstruction
- Forward Clark Transformation
- Forward and Backward Park Transformations
- DC-Bus Voltage Ripple Elimination
- Space Vector Modulation (SVM)

The slow control loop executes the speed controller and lower priority control tasks. The PI speed controller output sets a reference for the torque producing quadrature axis component of the stator current (i_{sq}).

2.3.3 Space vector modulation

Space vector modulation (SVM) can directly transform the stator voltage vectors from the two-phase α,β-coordinate system into pulse width modulation (PWM) signals (duty cycle values).

The standard technique of output voltage generation uses an inverse Clarke transformation to obtain 3-phase values. Using the phase voltage values, the duty cycles needed to control the power stage switches are then calculated. Although this technique gives good results, space vector modulation is more straightforward (valid only for transformation from the α,β-coordinate system).

The basic principle of the standard space vector modulation technique can be explained with the help of the power stage schematic diagram depicted in Figure 8. Regarding the 3-phase power stage configuration, as shown in Figure 8, eight possible switching states (vectors) are feasible. They are given by combinations of the corresponding power switches. A graphical representation of all combinations is the hexagon shown in Figure 9. There are six non-zero vectors, U_0, U_{60}, U_{120}, U_{180}, U_{240}, U_{300}, and two zero vectors, O_{000} and O_{111}, defined in α,β coordinates.

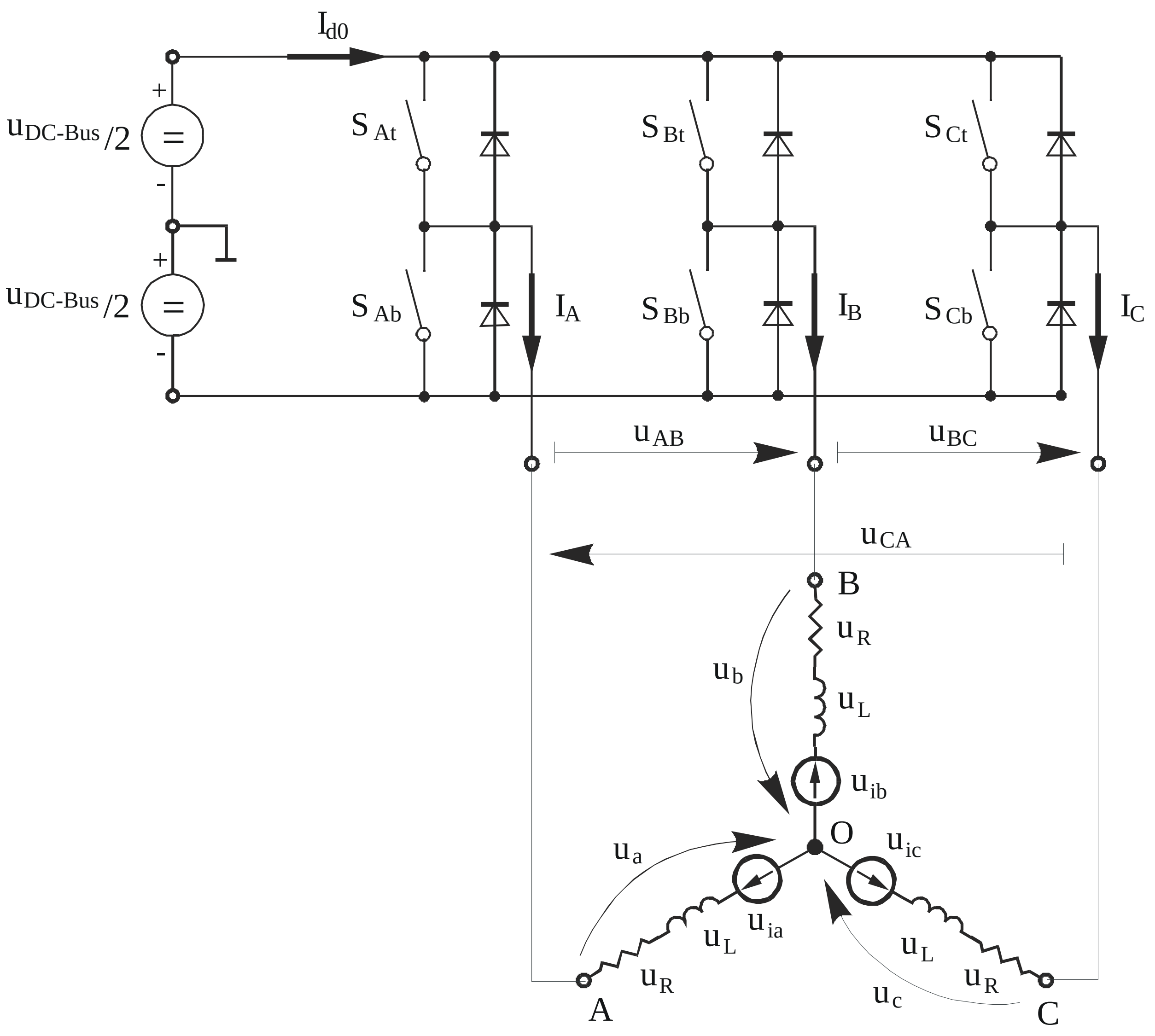


Figure 8. Power stage schematic diagram

The combination of ON/OFF states in the power stage switches for each voltage vector is coded in Figure 9 by the three-digit number in parenthesis. Each digit represents one phase. For each phase, a value of one means that the upper switch is ON and the bottom switch is OFF. A value of zero means that the upper switch is OFF and the bottom switch is ON. These states, together with the resulting instantaneous output line-to-line voltages, phase voltages and voltage vectors, are listed in Table 3.

Table 3. Switching patterns and resulting instantaneous

a	b	c	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	$2U_{DC\text{-}Bus}/3$	$-U_{DC\text{-}Bus}/3$	$-U_{DC\text{-}Bus}/3$	$U_{DC\text{-}Bus}$	0	$-U_{DC\text{-}Bus}$	U_0
1	1	0	$U_{DC\text{-}Bus}/3$	$U_{DC\text{-}Bus}/3$	$-2U_{DC\text{-}Bus}/3$	0	$U_{DC\text{-}Bus}$	$-U_{DC\text{-}Bus}$	U_{60}
0	1	0	$-U_{DC\text{-}Bus}/3$	$2U_{DC\text{-}Bus}/3$	$-U_{DC\text{-}Bus}/3$	$-U_{DC\text{-}Bus}$	$U_{DC\text{-}Bus}$	0	U_{120}
0	1	1	$-2U_{DC\text{-}Bus}/3$	$U_{DC\text{-}Bus}/3$	$U_{DC\text{-}Bus}/3$	$-U_{DC\text{-}Bus}$	0	$U_{DC\text{-}Bus}$	U_{240}
0	0	1	$-U_{DC\text{-}Bus}/3$	$-U_{DC\text{-}Bus}/3$	$2U_{DC\text{-}Bus}/3$	0	$-U_{DC\text{-}Bus}$	$U_{DC\text{-}Bus}$	U_{300}
1	0	1	$U_{DC\text{-}Bus}/3$	$-2U_{DC\text{-}Bus}/3$	$U_{DC\text{-}Bus}/3$	$U_{DC\text{-}Bus}$	$-U_{DC\text{-}Bus}$	0	U_{360}
1	1	1	0	0	0	0	0	0	O_{111}

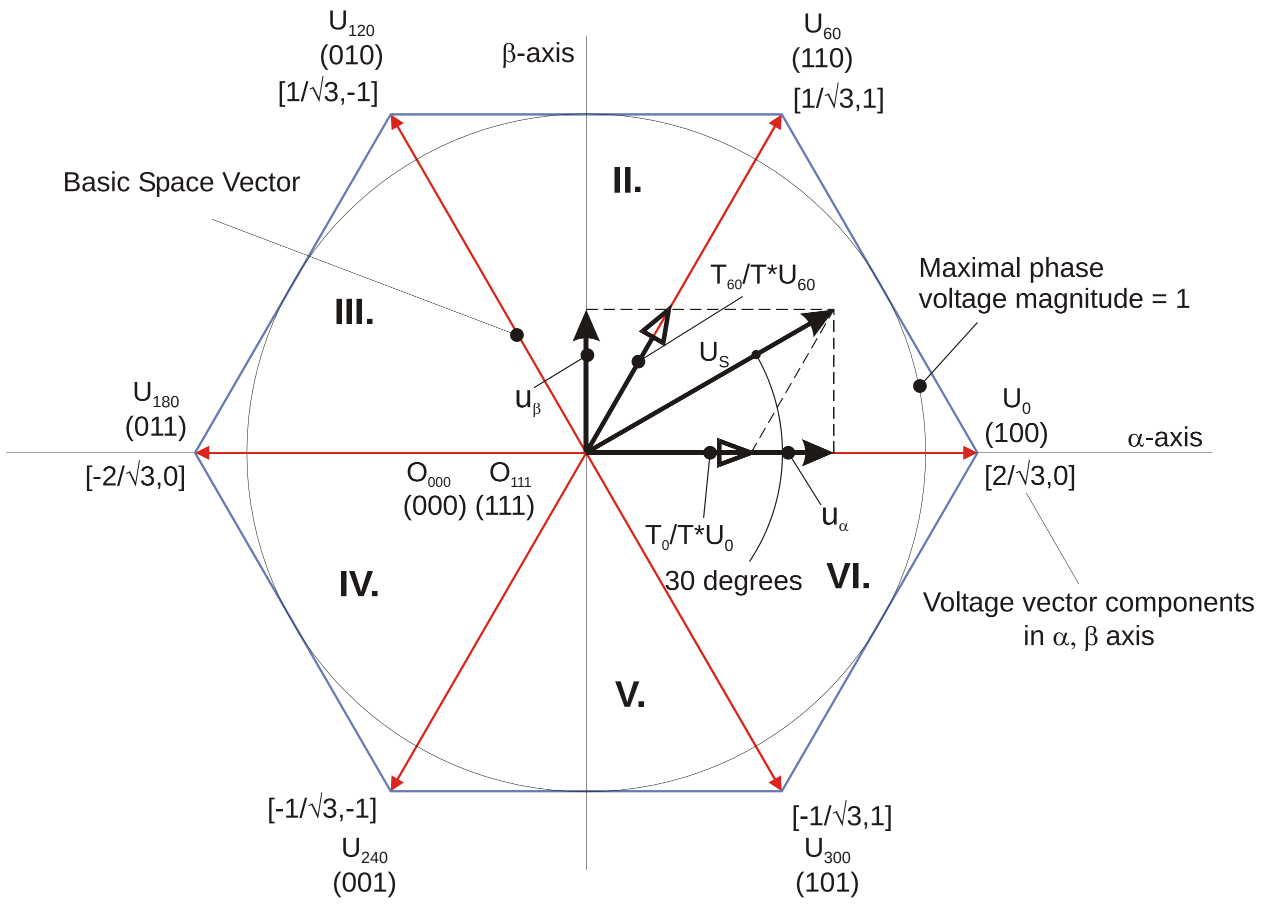


Figure 9. Basic space vectors and voltage vector projection

SVM is a technique used as a direct bridge between vector control (voltage space vector) and PWM.

The SVM technique consists of the following steps:

1. Sector identification
2. Space voltage vector decomposition into directions of sector base vectors U_x, $U_{x\pm60}$
3. PWM duty cycle calculation

The principle of SVM is the application of the voltage vectors U_{XXX} and O_{XXX} for certain instances in such a way that the "mean vector" of the PWM period T_{PWM} is equal to the desired voltage vector.

This method gives the greatest variability in arranging the zero and non-zero vectors during the PWM period. One can arrange these vectors to lower switching losses; another might want to reach a different result, such as centre-aligned PWM, edge-aligned PWM, minimal switching, etc.

For the chosen SVM, the following rule is defined:

- The desired space voltage vector is created only by applying the sector base vectors: the non-zero vectors on the sector side, (U_x, $U_{x\pm60}$) and the zero vectors (O_{000} or O_{111}).

The following expressions define the principle of the SVM:

$$T_{PWM} \cdot U_{S[\alpha,\beta]} = T_1 \cdot U_X + T_2 \cdot U_{X\pm60} + T_0 \cdot (O_{000} \vee O_{111})$$

Equation 23

$$T_{PWM} = T_1 + T_2 + T_0$$

Equation 24

In order to solve the time periods T_0, T_1, and T_2, it is necessary to decompose the space voltage vector $U_{S[\alpha,\beta]}$ into directions of the sector base vectors U_x, $U_{x\pm60}$. Equation 23 splits into Equation 25 and Equation 26.

$$T_{PWM} \cdot U_{SX} = T_1 \cdot U_X$$

Equation 25

$$T_{PWM} \cdot U_{S(X\pm60)} = T_2 \cdot U_{X\pm60}$$

Equation 26

By solving this set of equations, you can obtain the necessary duration for the application of the sector base vectors U_x, $U_{x\pm60}$ during the PWM period T_{PWM} to produce the right stator voltages.

$$T_1 = \frac{|U_{SX}|}{|U_X|} T_{PWM} \text{ for vector } U_X$$

Equation 27

$$T_2 = \frac{|U_{SX}|}{|U_{X\pm60}|} T_{PWM} \text{ for vector } U_{X\pm60}$$

Equation 28

$$T_0 = T_{PWM} - (T_1 + T_2) \text{ either for vector } O_{000} \text{ or } O_{111}$$

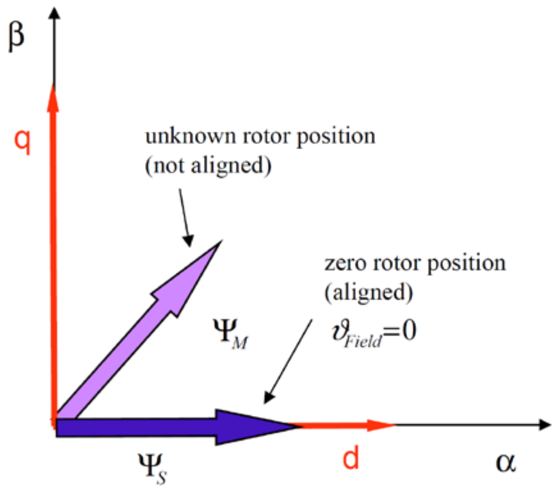
Equation 29

2.3.4 Position sensor elimination

The first stage of the proposed overall control structure is alignment algorithm of rotor PM to set an accurate initial position. This allows applying a full startup torque to the motor. In the second stage, the field-oriented control is in open-loop mode, in order 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, the rotor speed and position calculation is based on the estimation of a back-EMF in the stationary reference frame using a Luenberger type of observer.

2.3.5 Motor position alignment

In the presented design, the quadrature encoder is used as a motor position and speed sensor. Since the quadrature encoder does not give the absolute position, the exact position of the rotor before the motor is started, must be known. One possible and easily implantable method is the rotor alignment to a predefined position. The motor is powered by a selected static voltage pattern (usually the zero position in the sinewave table) and the rotor aligns to the predefined position. The alignment is done only once during the first motor start. Figure 10 shows the motor alignment. Before the constant current vector is applied to the stator, the rotor position is not known. After some stabilization period, the rotor flux must be aligned to the stator flux. In practice, this is true when the external load torque is low enough as compared to the torque produced by the alignment vector.



β
q
unknown rotor position (not aligned)
zero rotor position (aligned)
Ψ_M
$\vartheta_{Field}=0$
Ψ_S
d
α

Figure 10. Rotor alignment stabilization: PMSM starting mode

2.3.6 Open-loop startup

After identifying the initial rotor position, the field-oriented control is used in the open-loop mode (only in case of sensorless control). The current set-point is determined by the speed controller, which generates the torque reference current i_{Qref} and the proportional integral controller of speed control loop is initialized to maximum allowable current. The angular speed feedback ω_{FBCK} is kept at zero level during the open loop operation and the vector transformations are fed by a time varying reference position signal derived by integrating the speed ramp reference. This strategy moves the motor up to the speed values where the observer provides sufficiently accurate speed and position estimates.

The implementation of the open-loop startup is described in Motor open-loop startup.

2.3.7 Back-EMF observer

When the PMSM reaches a minimum operating speed, a minimum measurable level of back-EMF is generated by the rotor permanent magnets. The back-EMF observer then gradually transitions into the closed-loop mode. The feedback loops are then controlled by the estimated angle and estimated speed signals from the back EMF observer.

This estimation method for the position and angular speed is based on the motor mathematical model with an extended electromotive force function. This extended back-EMF model includes both position information from the conventionally defined back-EMF and the stator inductance. This allows to extract the rotor position and velocity information by estimating the extended back-EMF only.

$$\begin{bmatrix} u_\gamma \\ u_\delta \end{bmatrix} = \begin{bmatrix} R_S + sL_D & -\omega_r L_Q \\ \omega_r L_Q & R_S + sL_D \end{bmatrix} \begin{bmatrix} i_\gamma \\ i_\delta \end{bmatrix} + \left[\Delta L \cdot \left(\omega_{el} i_D - i'_Q\right) + k_e \omega_e\right] \cdot \begin{bmatrix} -\sin\theta_{error} \\ \cos\theta_{error} \end{bmatrix}$$

Equation 30

The observer is applied to PMSM motor with an estimator model excluding the extended back-EMF term. Then extended back-EMF term can be estimated using the observer as depicted in Figure 11, which utilizes a simple observer of PMSM stator current. The back-EMF observer presented here is realized within rotating reference frame (dq). The estimator of dq-axis consists of the stator current observer based on RL motor circuit with estimated motor parameters. This current observer is fed by the sum of the actual applied motor voltage, cross-coupled rotational term, which corresponds to the motor saliency (L_d-L_q), and compensator corrective output. The observer provides back-EMF signals as disturbance because back-EMF is not included in the observer model.

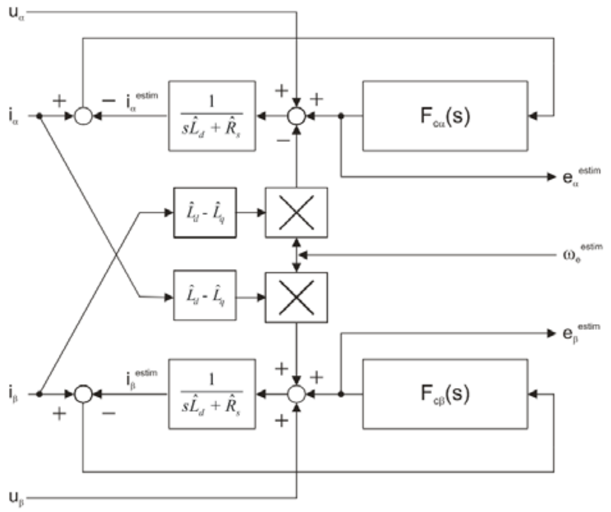


Figure 11. Luenberger type stator current observer acting as state filter for back-EMF

2.3.8 Speed and position extraction

To get the speed and position information from the position error, another algorithm is used—tracking observer. This algorithm adopts the phase-locked loop mechanism. It requires a single input argument as phase error. Such phase tracking observer, with standard PI controller used as the loop compensator, is depicted in Figure 12.

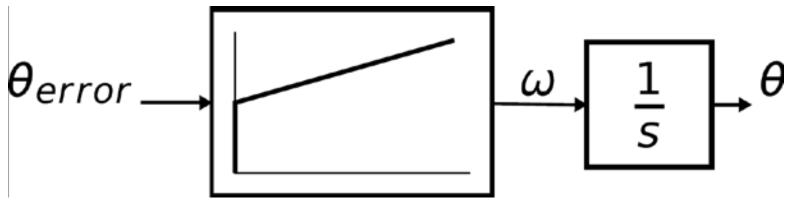


Figure 12. Block diagram of proposed PLL scheme for position estimation

2.4 PFC average current control mode theory

The control structure is divided into two loops: an inner current control and an outer voltage control loop. The outer voltage control loop is implemented via software in the microcontroller and keeps a constant voltage on the DC bus. The voltage control loop utilizes a PI controller, and the output defines the amplitude required for the PFC current. The PFC control algorithm provides a sinusoidal input current without phase shift to the input voltage through dedicated PFC hardware controlled by the DSC.

The hardware incorporates an input bridge rectifier, PFC inductances (L_1 and L_2), PFC diodes (D_1 and D_2), and PFC switches (Q_1 and Q_2) (see Figure 13). These analogue quantities sensed are rectified input voltage, PFC current, and DC-bus voltage. The input current is controlled using the PFC switch to achieve the desired input current and the desired level of the DC-bus voltage.

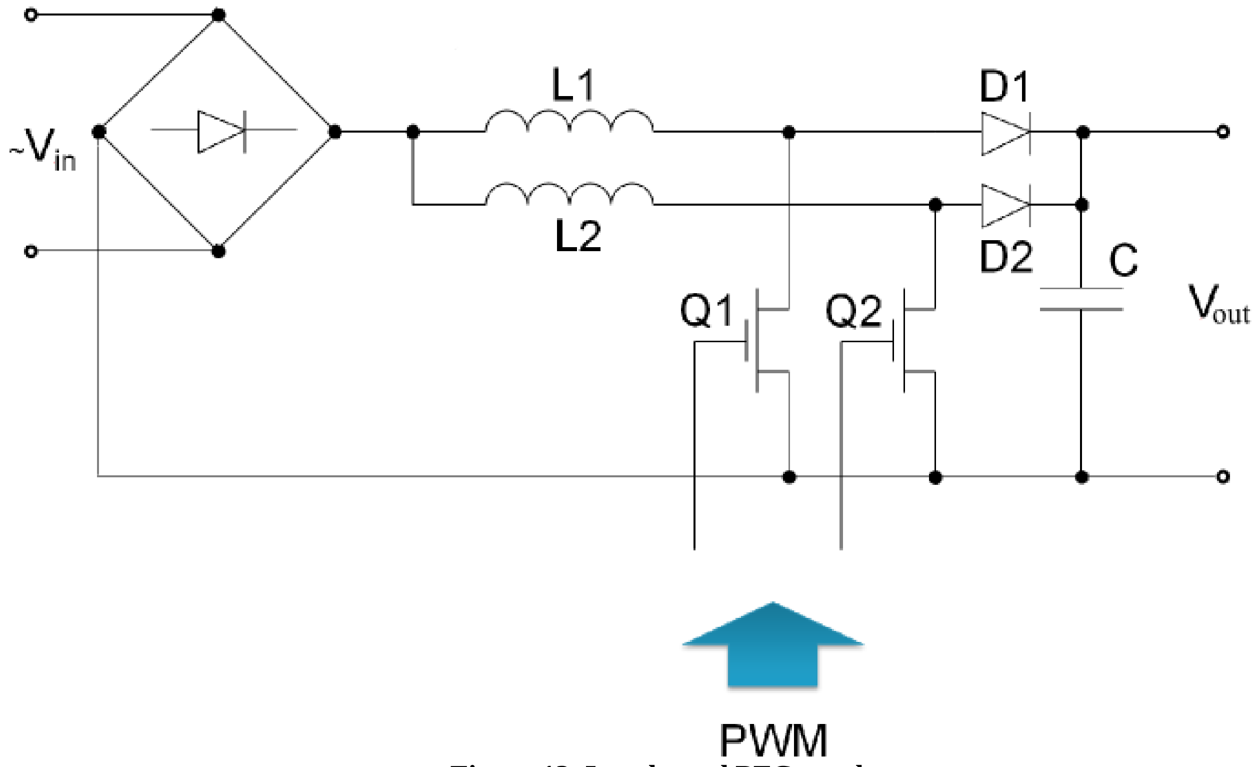
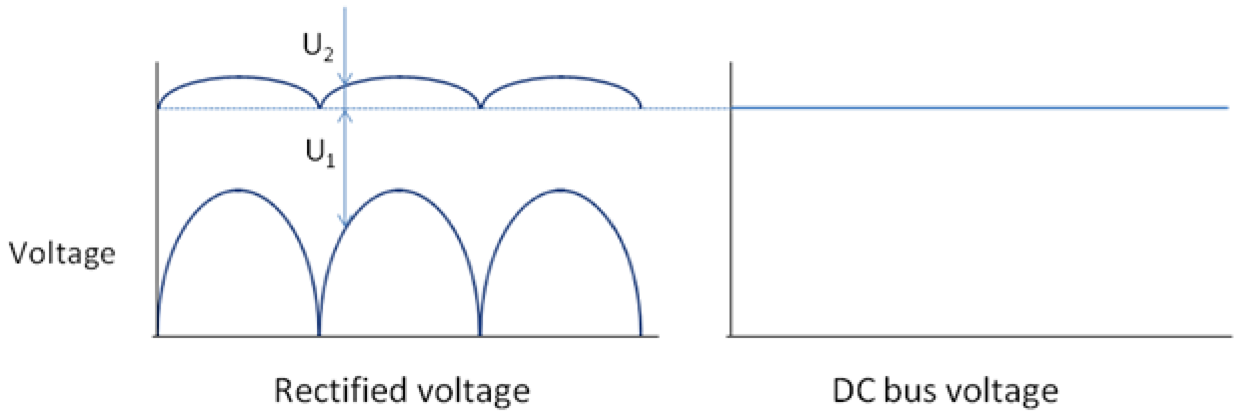


Figure 13. Interleaved PFC topology

The inner current loop is implemented via software as with the outer loop, and employs the PI controller to maintain the sinusoidal input current by directly controlling the PFC transistor. The input to the PI controller is the difference between the current reference and the actual current. The sinusoidal waveform of the required current is maintained by multiplying the required current reference by the generated sine signal that is in phase with the rectified input voltage. The current PI controller's output generates a signal corresponding to the needed voltage on the PFC inductor to make the required current flow to the DC-bus capacitor. Thus, this voltage will depend on the immediate input voltage value and the voltage drop on the parasitic resistance of the PFC components (see Figure 14). The final duty cycle of the corresponding voltage on the inductor is then given its division by the input voltage.



Voltage on the inductor = $U_1 + U_2$

U_1 – minimum voltage to get to the DC bus level

U_2 – voltage drop on the PFC components' resistance

Figure 14. PFC voltage

2.4.1 Interleaved PFC

The PFC energy transfer is not continuous; it is done in determined cycles. This happens due to the charging and discharging of inductor with the PWM frequency. If the load is heavy, the DC-bus capacity may not be sufficient to keep the voltage more or less constant and the voltage will have ripples with respect to the charging and discharging times on the inductor. To reduce this effect, the interleaved conception of PFC has been used.

The interleaved PFC means that the PFC components such as the inductors, diodes, and switches, are doubled. Then, the switches are switched in counter phase, that is, with a shift of 180° to each other. In this case, the output voltage ripple will be lowered because the voltage at the L1 inductor's charging phase is backed up by the L2 inductor's discharging phase and vice versa.

From one point of view, this strategy seems to be a nice solution. But, the interleaved PFC concept is used only for high-power applications because of the following reasons.

- As all the components are doubled, costs on components will be higher.
- More space on the PCB is required especially for the additional inductor and an additional control signal is required.

Chapter 3
System Concept

3.1 PFC dual motor control system specification

The concept of the PFC dual motor control system is displayed in Figure 15. It represents the typical application requirements for the air-conditioning where one motor controls the compressor and the other controls the fan.

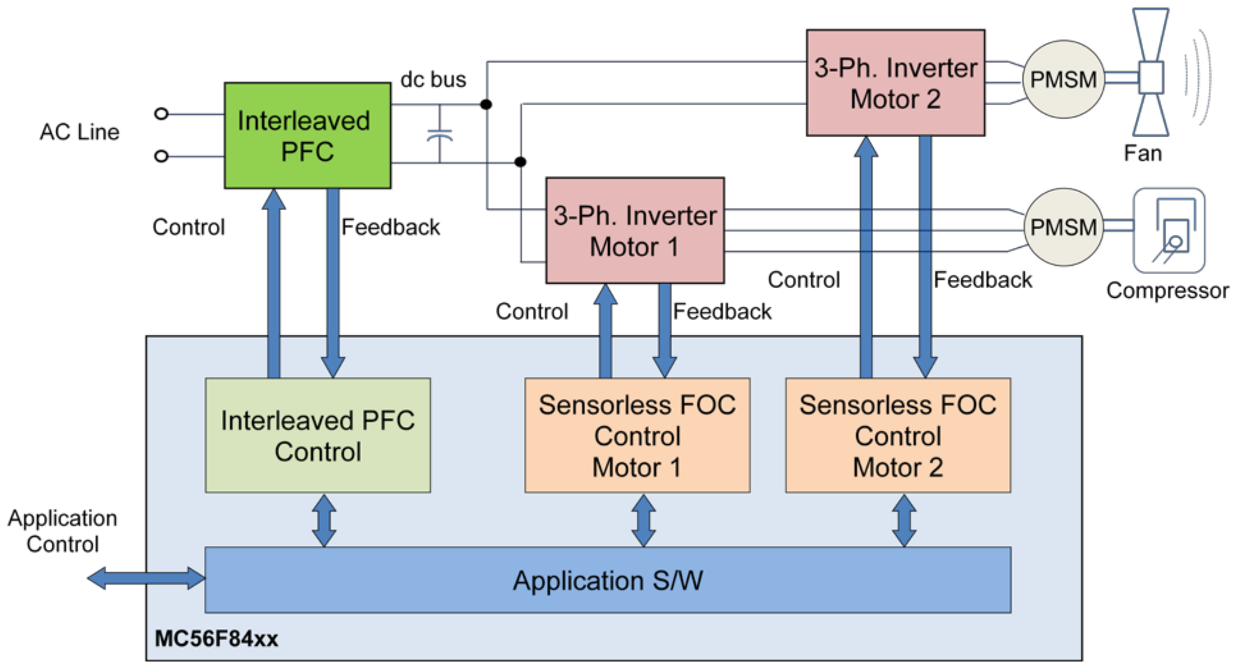


Figure 15. PFC dual motor control concept

The application consists of the field-oriented control of two sensorless permanent magnet synchronous motors and the control of the power factor correction. The specifications for each component are the following:

- Motor 1— compressor drive
    - Sensorless PMSM FOC with field-weakening
    - Alignment and open-loop startup
    - Reconstruction of 3-phase motor current from two shunt resistors
    - 10 kHz PWM
    - 10 kHz fast loop frequency
    - 1 kHz slow loop frequency
    - Single direction of rotation
    - 400 W, 3 A peak current

- Two pole pairs
  - Mechanical speed range 900 to 5000 RPM
  - Electrical speed 1800 to 10000 RPM (30 to 166 Hz)
  - Speed ramp 2500 RPM/s
- Motor 2—fan drive
  - Sensorless PMSM FOC with field-weakening
  - Alignment and open-loop startup
  - Reconstruction of 3-phase motor current from two shunt resistors
  - 10 kHz PWM
  - 10 kHz fast loop frequency
  - 1 kHz slow loop frequency
  - Single direction of rotation
  - 30 W, 0.3 A peak current
  - Four pole pairs
  - Mechanical speed range 500 to 2000 RPM
  - Electrical speed 2000 to 8000 RPM (33 to 133 Hz)
  - Speed ramp 1000 RPM/s
- PFC
  - 2-MOSFET Interleaved mode
  - Current measurement on a single shunt resistor
  - PWM frequency 80 kHz
  - 20 kHz fast loop frequency
  - 500 Hz slow loop frequency
  - 12 A peak current
  - Output voltage ramp 300 V/s
- Miscellaneous application features
  - Measurement of temperatures at three different points (hot side, cold side, ambient)
  - RS-232 communication with the remote touch graphic LCD control based on Kinetis K70 Tower system
  - Fault protection—overcurrent, overvoltage, undervoltage, over-temperature, input frequency range check

3.2 Application description

The application incorporates the following hardware components:

- High-voltage PFC dual motor power stage
- 3-phase permanent magnet compressor
- 3-phase permanent magnet fan
- MC56F84789 daughter board
- Tower system based on:
  - Kinetis K70 Tower System Module (TWR-K70F120M)
  - Tower System Graphical LCD module with RGB interface (TWR-LCD-RGB)
  - Serial (USB, Ethernet, CAN, RS232/485) Tower System Module (TWR-SER)
  - Primary and secondary elevators (TWR-ELEV)

- 5 V power supply for the tower system

The MC56F84789, populated on the MC56F84789 daughter board, executes the control algorithms. In response to the user interface that sends the commands to the MC56F84789 DSC over the RS232 communication and the local feedback signals, it generates PWM signals to drive two sensorless PMS motors and the interleaved PFC.

3.2.1 Motor control process

Both the motors use the same software structure to be driven. The only difference between the two motors is in the different set of parameters used and the speed control. For the compressor, the PID controller is used, while for the fan, the PI controller is sufficient.

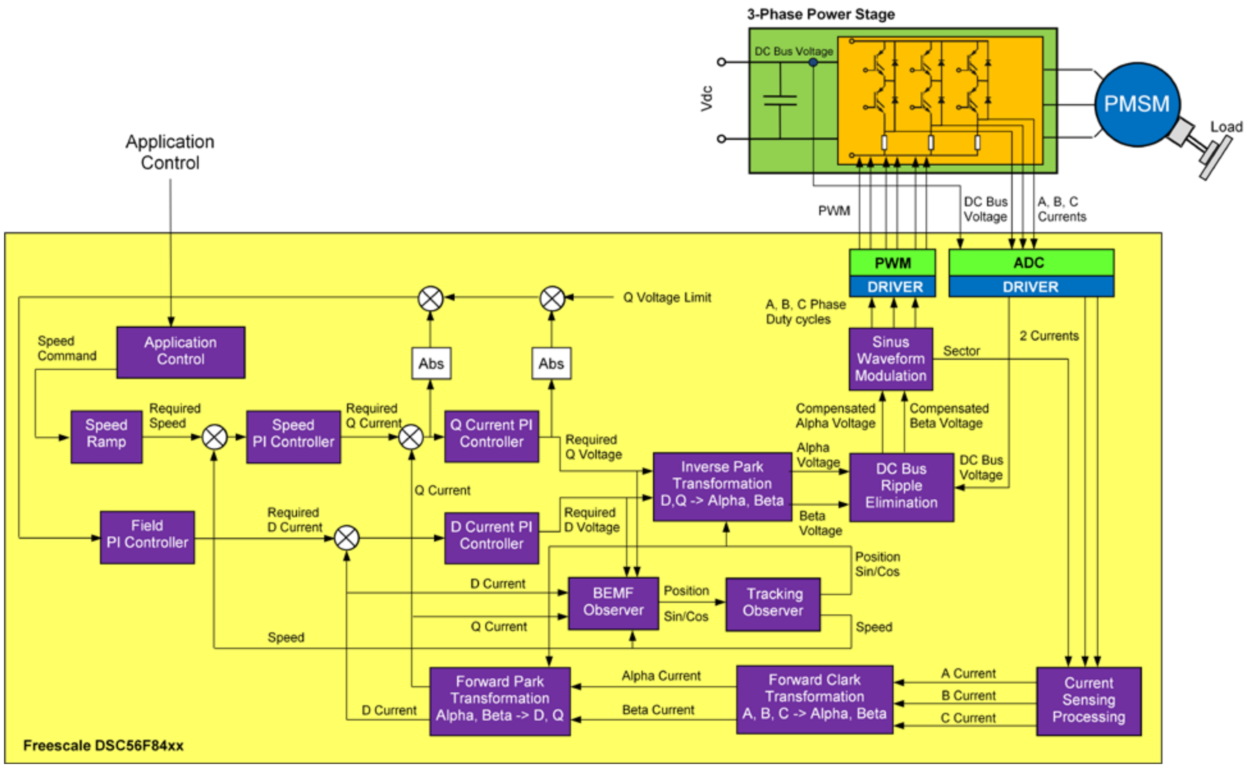


Figure 16. Sensorless PMSM field-oriented control algorithm blocks

Figure 16 shows the FOC algorithm blocks of a PMSM. The state of the ON/OFF switch variable of each motor control piece of application is periodically scanned. The speed command that is entered from the high-level application layer is processed by means of the speed-ramp algorithm. The comparison between the actual speed command, obtained from the ramp algorithm output, and the measured speed generates a speed error. The speed error is input to the speed PI(D) controller, generating a new desired level of reference for the torque-producing component of the stator current.

The DC-bus voltage and phase currents are sampled with ADC. The ADC sampling is started by the PWM module trigger signals. A digital filter is applied to the sampled values of the DC-bus voltage. The phase currents are used unfiltered. The 3-phase motor current is reconstructed from two samples taken from the inverter's shunt resistors. The reconstructed 3-phase current is then transformed into space vectors and used by the FOC algorithm.

The rotor position and speed are given by the BEMF observer and the Tracking Observer. The BEMF observer calculates the error of the angle using the d/q voltage, d/q current, and the speed. The error angle of the BEMF is supplied into the Tracking Observer, which returns the position and speed. Based on measured feedback signals, the FOC algorithm performs a vector-control technique, as described in Description of the vector control algorithm.

The motors are also able to operate in the speed and torque ranges where the DC-bus voltage is not high enough over the motor's BEMF. This condition is given by the difference of the Q voltage reserve (difference between the Q voltage limit and the applied Q voltage), and the Q current error. If the difference is negative, the Field PI Controller starts reducing the required D current; the field-weakening algorithm executes to reduce the permanent magnet magnetic flux and lower the actual BEMF. Having lowered the BEMF, current can be delivered to the motor phase from the DC-bus again. The speed increases but the capability of the maximum torque is proportionally reduced. This method of field weakening has been developed and later patented by Freescale (WO/2009/138821).

Two independent current PI controllers are executed to achieve the desired behavior of the motor. Output from the FOC is a stator-voltage space vector, which is transformed by means of Space Vector Modulation into PWM signals. The 3-phase stator voltage is generated by means of a 3-phase voltage source inverter and applied to the motor, which is connected to the power stage terminals.

3.2.2 PFC control process

The PFC uses similar software structure approach as the motor control piece of software. The FSLESL algorithms used for the motor control are also used for the PFC control. See Figure 17.

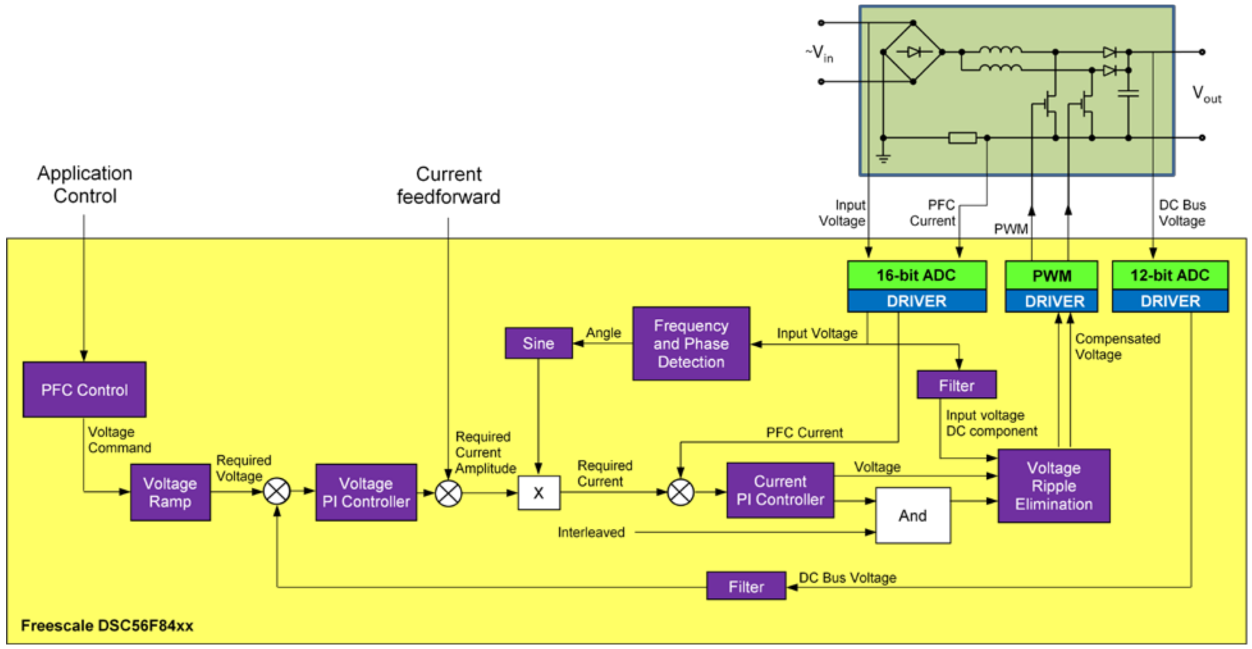


Figure 17. PFC control algorithm blocks

The state of the ON/OFF switch variable of the PFC control piece of application is periodically scanned. The voltage command that is entered from the high-level application layer is processed by means of the voltage-ramp algorithm. The comparison between the voltage command, obtained from the ramp algorithm output, and the measured DC-bus voltage is a voltage error. The voltage error is input to the voltage PI controller, generating a new desired level of reference for the current amplitude.

The input (rectified) voltage is measured. An algorithm to find its phase and frequency is applied on the measured voltage. The output from this algorithm is the angle in phase with the input voltage. The sine of this angle is calculated and multiplied with the current reference amplitude (output from the voltage PI controller). The comparison between this product and the measured current is the current error. The current error is the input to the current PI controller, generating a new desired level of voltage on the PFC inductors.

The desired voltage is fed to the voltage ripple elimination algorithm. This algorithm uses the input voltage DC component to calculate the duty cycles for the MOSFET's from the desired voltage.

3.2.3 Application State Machine Blocks of Control

The entire application consists of several state machines. Each of them has a unique functionality. The motors and PFC have their own state machines. These state machines are slaves to the main application state machine. The application state machine controls and monitors the state machines of the motors and PFC. Also, there is an independent communication state machine which receives and answers the commands from the UART. See Figure 18.

The communication state machine receives only the following commands:

- **Set:** This command sets the application switch on or off. The compressor requested speed is included in this command. The answer to this command is the status of the application like the state machine switches status, speed of the motors and current, temperatures, voltages, faults.
- **Get M1 current phases:** This command starts recording the phase currents of motor 1 into the buffer and when finished, the answer to this command will be the buffer content.
- **Get M2 current phases:** This command starts recording the phase currents of motor 2 into the buffer and when finished, the answer to this command will be the buffer content.
- **Get PFC voltage and current:** This command starts recording the input voltage and current of PFC into the buffer and when finished, the answer to this command will be the buffer content.

The application state machine receives the Set command to switch the application on. The application state machine is switched into the Run state where the PFC state machine is commanded to switch on with the application of the voltage command.

When the application state machine receives the Set command with the requested compressor speed higher than the minimum speed value, the application state machine commands the motor 1 state machine to be switched on and sends the requested speed command. The compressor runs.

The application state machine monitors three temperatures of the system:

- Hot-side—temperature of the condenser
- Cold-side—temperature of the evaporator
- Ambient—temperature of the air flowing out of the fan

When the hot-side temperature reaches certain value, the application state machine commands the motor 2 state machine to be switched on and sends the requested fan speed command. This speed is proportional to the hot-side temperature. When the temperature gets low to the acceptable temperature, the fan is completely stopped and the motor 2 state machine is commanded to be switched off.

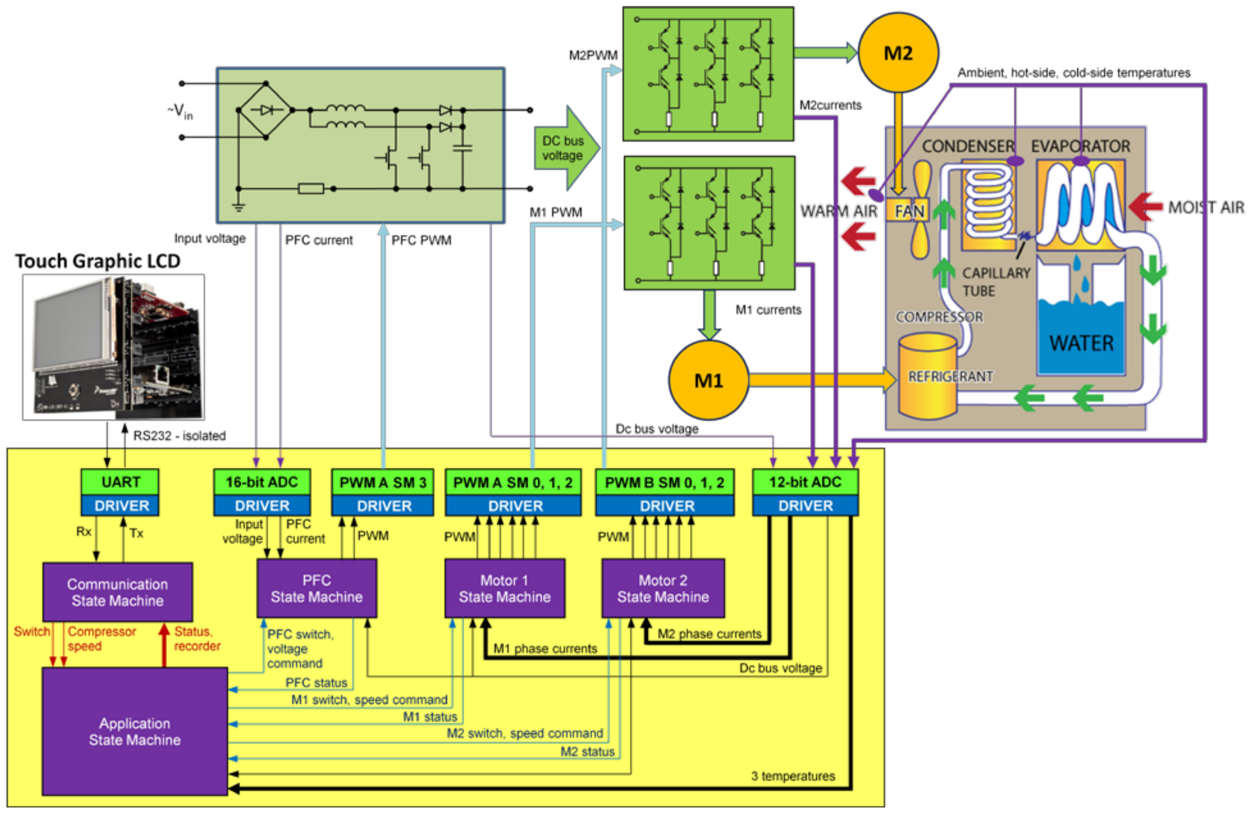


Figure 18. Application state machine blocks of control

When the application is switched off or receives a fault, the system is turned off in a sequence. First, the compressor (motor 1) is turned off. The fan (motor 2) runs until the hot-side temperature is acceptable and then it is turned off. When both motors are turned off, the PFC is turned off. When PFC is turned off, the application state machine is switched into the Stop or Fault mode.

Chapter 4
Hardware

4.1 Power stage purpose

This power stage board in conjunction with the daughter card controller board is intended to control one or two 3-phase electromotors simultaneously. The power limit is 1500 W for the main motor 1 and 500 W for the second motor 2. The power board is equipped with the interleaved power factor corrector (PFC) stage, so the power line supply voltage range is from 110 V/60 Hz to 230 V/50 Hz. The maximum power is restricted to 1 kW for the input voltage 110 V AC. The board is equipped with the non isolated USB interface for the communication purpose to PC.

The whole 3-in-1 power stage is shown in Figure 19.

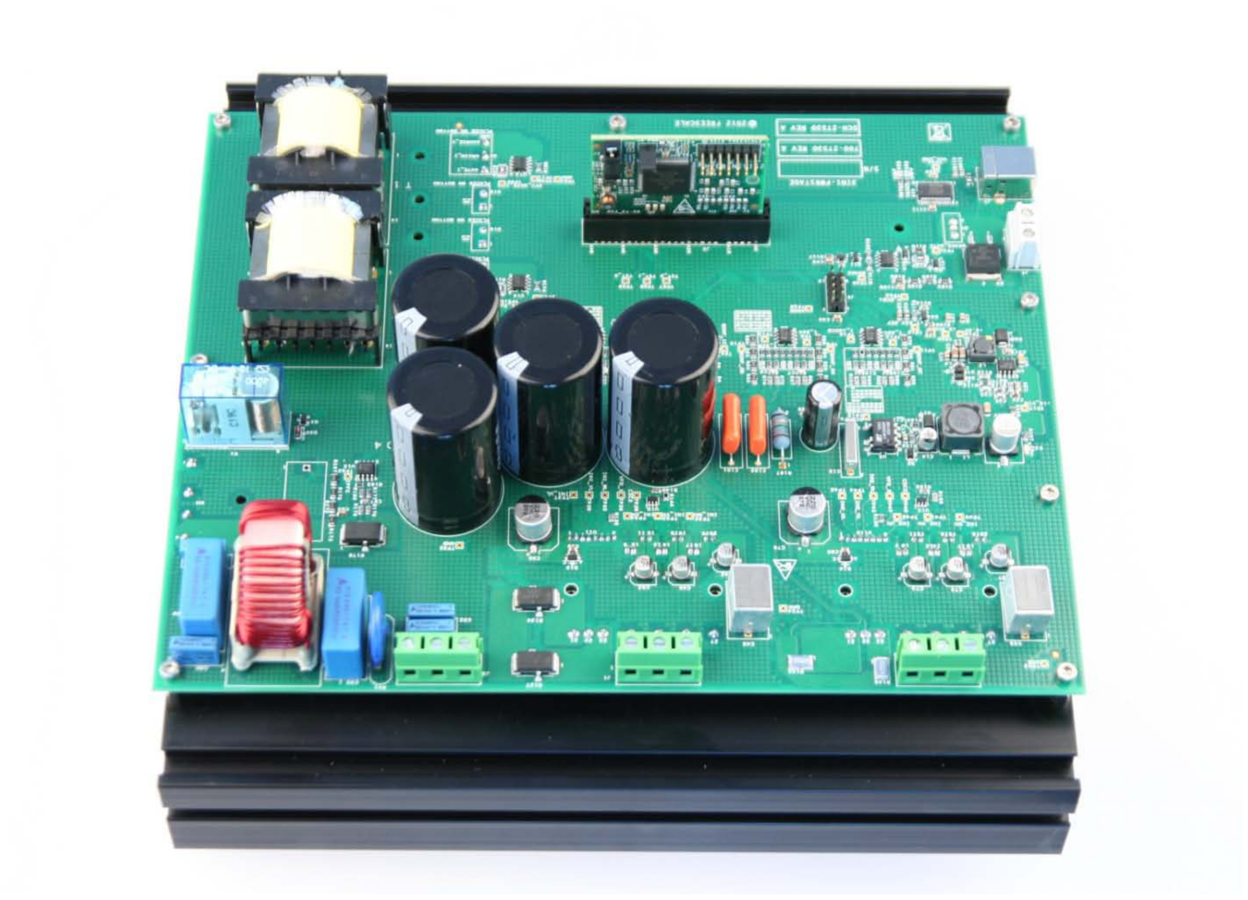


Figure 19. 3-in-1 power stage board

4.2 Board specification

Following are the specifications of the 3-in-1 power stage board:

- Input voltage range 85-265 V AC, 50/60 Hz;
- Interleaved PFC on input
- Two 3-phase inverters (1500 W + 500 W); each of them accommodates
    - IGBT inverter bridges with overcurrent protection
    - 3-phase motor current sensing
    - DC-bus current sensing
    - Back-EMF voltage sensing
- DC-bus voltage sensing
- Galvanic isolated SCI / USB interface
- User LEDs
- DC-Brake switch with terminals for brake resistor
- PCI Express interface connector for the control board
- On-board power supply+3.3 V, +3.3 VA, and +15 V

Following are the specifications of 3-in-1 controller board:

- DSC MC56F84789 in 100-pin LQFP package
- PCI Express edge connector compatible with power stage board
- JTAG interface
- Low power on-board power supply +3.3 V
- Power-on LED

4.3 Hardware description

The block diagram of the 3-in-1 power stage board is shown in Figure 20.

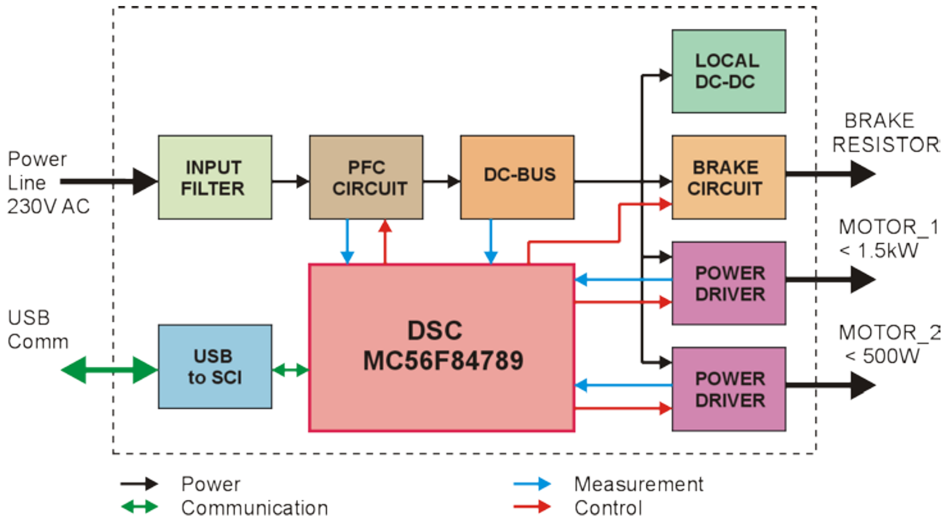


Figure 20. 3-in-1 power stage board block diagram

The heart of the power board is the control daughter card, equipped with the digital signal controller (DSC) MC56F84789.

4.4 Power line input filter

The power line input filter provides the attenuation of the high-frequency noise generated by the PFC stage. The structure of this standard common mode filter is shown in Figure 21. The cut-off frequency of this filter is 2.9 kHz.

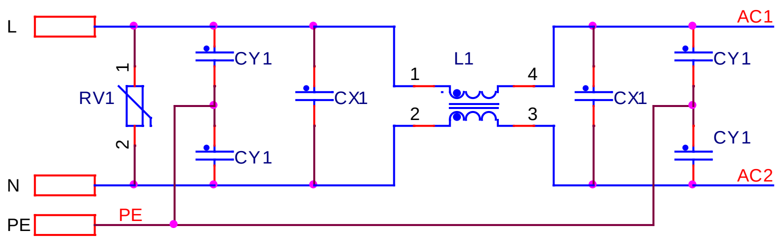


Figure 21. Power line input filter

4.5 Interleaved PFC stage

The whole power board is designed for the overall maximum power of 2 kW. Thus, the PFC stage is used in the interleaved configuration. Each inductor circuit can transfer up to 1 kW of power. The basic structure is shown in Figure 22.

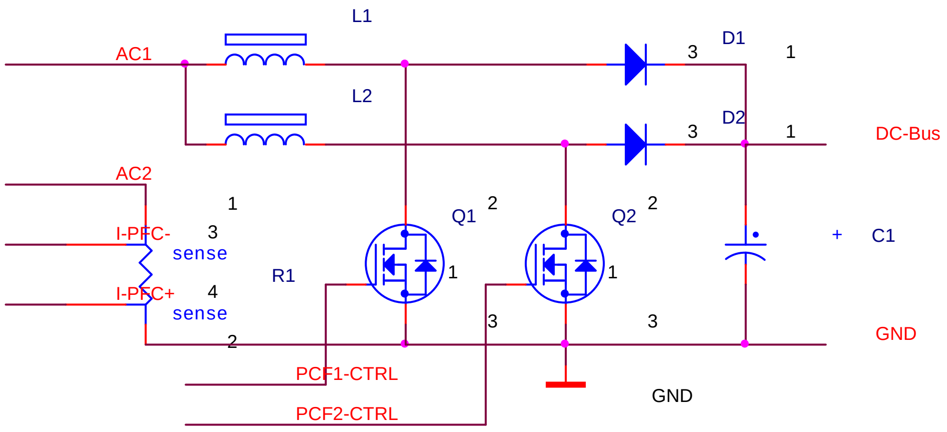


Figure 22. Interleaved PFC

The PFC stage is controlled by the DSC. The controller measures the currents in the both PFC coils and the DC-bus voltage. It maintains the stable level of the DC-bus voltage and the sine shape of the input current from the power line. The power level of the control signals is changed by the separate MOSFET drivers. The MOSFET drivers are powered from the onboard power supply +15 V DC. The PFC currents are sensed by the common sensing resistor R1. The voltage drop on this resistor is amplified by the standard operational amplifier and connected to the analog input of the DSC.

4.6 Local DC-to-DC power supply

The DC-bus capacitor is the main energy storage for the both the power drivers. It provides the power for the succession on board power supplies: +15 V for the power drivers and +3.3 V DC for the digital and analog sections of the control circuitry. The reference voltage +1.65 V is derived from the +3.3 VA line. The structure of this local power supply is given in Figure 23.

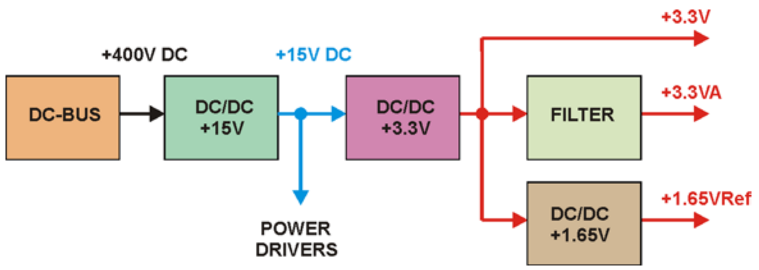


Figure 23. Local DC-DC power supply

4.7 DC-bus brake and protection circuit

The brake circuit is intended for the DC-bus overvoltage protection. The circuit consists of the braking power resistor, switching device (IGBT or MOSFET), and the control circuit. The switching device can be controlled by the control DSC and independently by the hardware comparator. The hardware comparator is set to maximum voltage level in the range of 405–410 V DC. The functional schematic is shown in Figure 24.

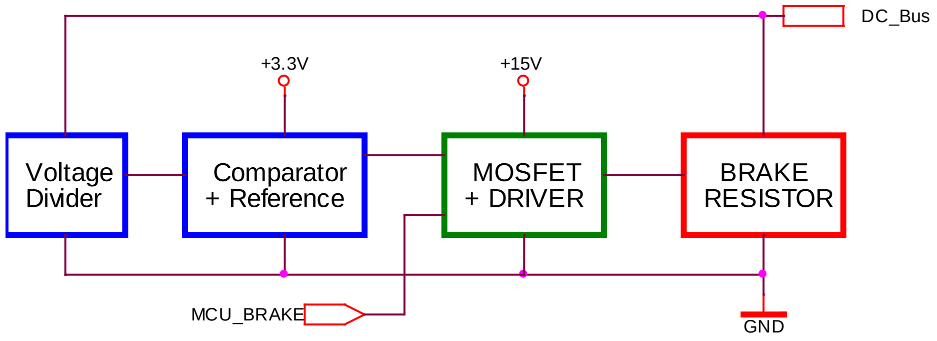


Figure 24. DC-bus brake & protection circuit

4.8 3-phase power driver

The power driver provides the 3-phase power for the motor. It contains the power IGBTs in the 3-phase half-bridge configuration, the gate drivers for the each IGBT, the boost diodes for the each phase, and the undervoltage lockout and overcurrent protection circuits. The power module with main control, power supply, and output signals is shown in Figure 25.

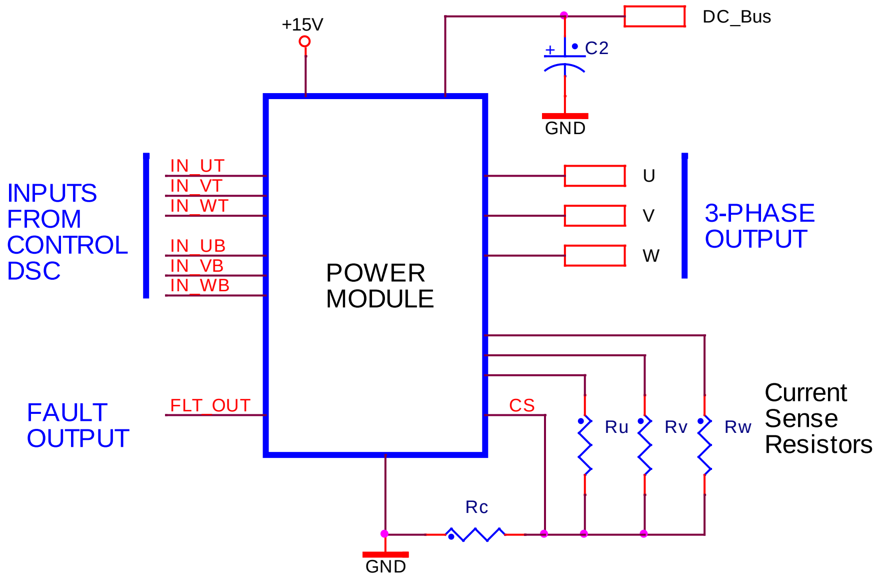


Figure 25. Power module connection

The DC-bus connection provides the main power to the module. The low-voltage +15 V line provides the power for the internal MOSFET drivers and other analog and digital circuits inside the module. The inputs are the PWM control signals from DSC in the +3.3 V level. These inputs are internally connected to the MOSFETs' gate drivers. The top and bottom of each MOSFET can be independently controlled by the DSC. The 3-phase output is intended for the motor connection. The phase currents and the whole DC-bus current are sensed by the current sense resistors in topology shown in Figure 25. The voltage from the resistor Rc is used by the internal circuit for the overcurrent protection. If the voltage on the resistor R_C reaches the 0.5 V level, the internal circuit switches OFF all MOSFETs and sends the FAULT signal to control the DSC. Then the DSC stops the generation of PWM control signals. This configuration maintains the fastest response to the overcurrent events.

The power modules for high-power (1500 W) and the low-power (500 W) motor have the same hardware configuration. The modules differ only in the maximum output current.

4.9 Analog measurement

The measurement amplifiers provide the amplification of the low-voltage level signals from the current sensing resistors. They are powered by the filtered +3.3VA analog voltage line. The standard circuit is used for the +1.65V voltage reference. The main structure of the measurement amplifier is shown in Figure 26 .

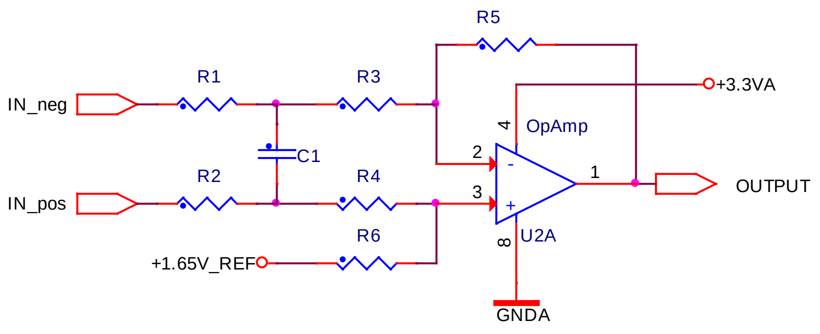


Figure 26. Analog amplifier

The input resistors R1 and R2 with the small capacitance C1 maintain the filtering of the high-frequency noise. The resistors R5 and R6 together with R1, R2, R3, and R4 define the amplification factor of the whole analog amplifier. The output is connected to the ADC input of the DSC through the input RC filter.

The voltages are measured by the ADC pins on the voltage dividers where the maximum voltage corresponds to 3.3 V and the minimum voltage (0 V) corresponds to 0 V.

4.10 SCI-to-USB interface

The SCI interface on the control DSC is intended for debugging or as a communication channel to external device. The USB port is chosen as it provides an easy connection to the PC. It is built on the standard FTDI chip powered from the USB port. For the connection to the PC, it must be used with the USB isolator because the onboard SCI/USB interface is not isolated from the power line.

4.11 Control daughter card

The control daughter card is the heart of the whole board. It is equipped by the Freescale DSC MC56F84789 in 100-pin LQFP package. The block schematic of this board is shown in Figure 27.

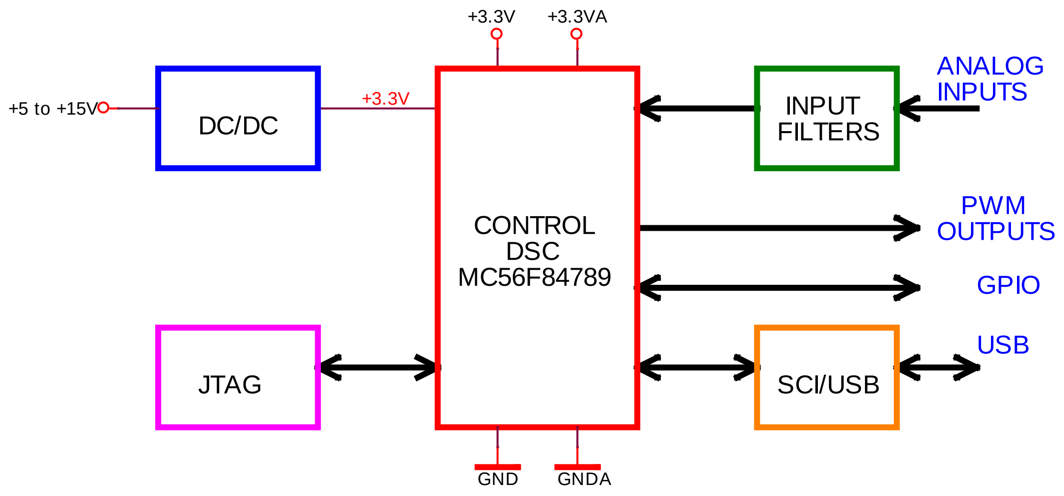


Figure 27. Control daughter card

The daughter card and the main power board are connected through the PCI Express 98-pin connector. All the pins excluding JTAG pins of the DSC are available on the PCI connector. The daughter control board is powered by the main power board. While it is used independently, it can be powered by the small low-power onboard DC-DC buck converter.

The input filters for the analog signals are simply RC low-pass filters. It improves the ADC measurement quality of the DSC.

The output PWM signals are connected to the MOSFET drivers. The GPIO pins are also available for general use.

The SCI communication port of the DSC is used for the FreeMASTER tool running on the PC. For a simple connection, it is transferred to the USB port. It can be used for a connection to the any other external device.

Note: Be aware that the JTAG and USB ports must be externally isolated from the application.

Chapter 5
Software Design

This application has been written under CodeWarrior 10.2 (later converted to 10.3) in simple C project with the inclusion of Freescale's Embedded Software Libraries (FSLESL). In this application development, no configuration or speed development tool has been used. These libraries can be downloaded from **freescale.com/fslesl** and contains necessary algorithms, math functions, observers, filters, and so on, used in the application. See *AN4586: Inclusion of DSC Freescale Embedded Software Libraries in CodeWarrior 10.2*, available on **freescale.com** to know about the inclusion of the libraries into the project.

This chapter describes the software design of the PFC dual sensorless PMSM vector control application. First, the numerical scaling in fixed-point fractional arithmetic of the controller is discussed. Then, specific issues such as speed and current sensing are explained. Finally, the control software implementation is described.

5.1 Fractional numbers representation

This application uses a fractional representation for most of the quantities. The fractional arithmetic is supported by the Freescale DSCs. The application and the algorithms library benefits from this fact.

The N-bit signed fractional format is represented using the 1.[N-1] format (1 sign bit, N-1 fractional bits). Signed fractional numbers (SF) lie in the following range:

$$-1.0 \le SF \le +1.0 - 2^{-[N-1]}$$

For words and long-word signed fractions, the most negative number that can be represented is –1.0, whose internal representation is 0x8000 and 0x80000000, respectively. The most positive word is 0x7FFF or $1.0 - 2^{-15}$, and the most positive long-word is 0x7FFFFFFF or $1.0 - 2^{-31}$.

5.2 Scaling of analog quantities

The following equation shows the relationship between a real and a fractional representation:

$$\text{Fractional Value} = \frac{\text{Real Value}}{\text{Real Quatity Range}}$$

Equation 31

Where:

- Fractional Value = Fractional representation of quantities [–]

- Real Value = Real quantity in physical units [..]
- Real Quantity Range = Maximum defined quantity value used for scaling in physical units [..]

A few examples of how the quantities are scaled are provided in the following subsections.

5.2.1 Voltage scale

The voltage is generally measured on the voltage resistor divider by the ADC. So, the maximum voltage scale is proportional to the maximum ADC input voltage range. In this application, the maximum voltage is 472 V. The example below shows how the fractional voltage variable is used:

Voltage scale: $V_{max} = 472$ V
Measured voltage: $V_{measured} = 352$ V

$$(\mathrm{Frac16})\mathrm{voltage\_variable} = \frac{V_{\mathrm{measured}}}{V_{\mathrm{max}}} = \frac{352\mathrm{V}}{472\mathrm{V}} = 0.7458$$

Equation 32

This 16-bit fractional variable is internally stored as a 16-bit integer variable:

$$(\mathrm{Int16})\mathrm{voltage\_variable} = (\mathrm{Frac16})\mathrm{voltage\_variable} \cdot 2^{15} = 0.7458 \,.\, 2^{15} = 24438$$

Equation 33

shows Equation 32 and Equation 33.

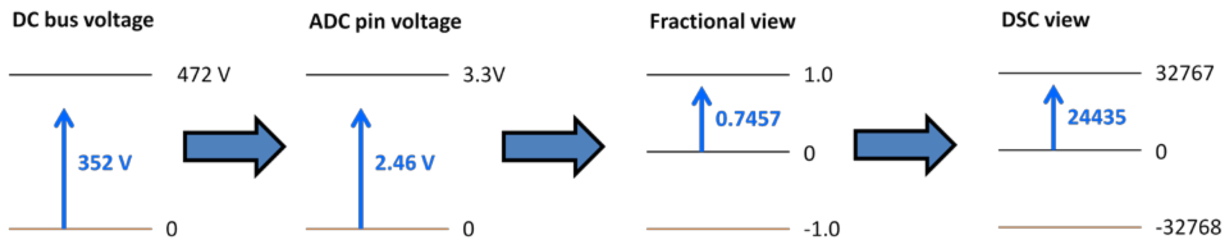


Figure 28. Voltage measurement

5.2.2 Current scale

The current is generally measured as voltage drop on the shunt resistors which is amplified by an operational amplifier. If both the positive and negative currents are needed, the amplified signal has an offset. The offset is generally the half of the ADC range. Then the maximum current scale is proportional to the half of maximum ADC input voltage range, see Figure 29. The manipulation with the current variable is similar to the voltage variable manipulation.

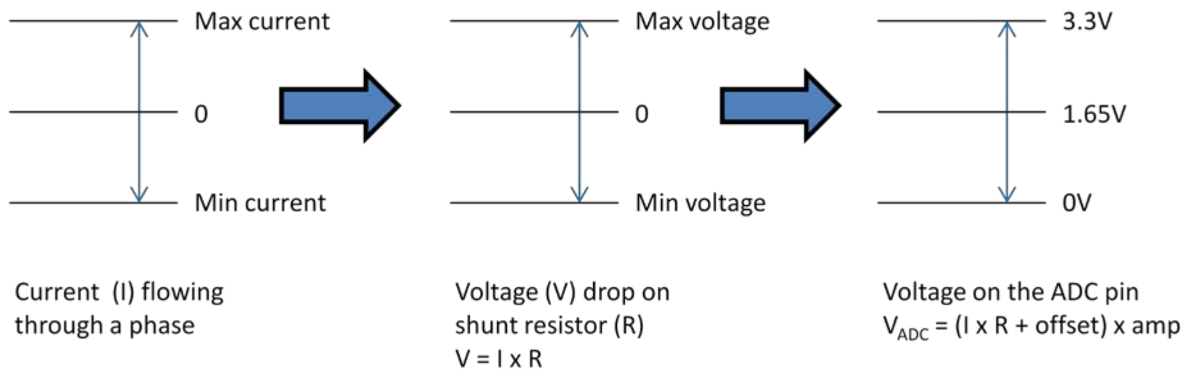

Max current
0
Min current
Max voltage
0
Min voltage
3.3V
1.65V
0V
Current (I) flowing through a phase
Voltage (V) drop on shunt resistor (R)
V = I x R
Voltage on the ADC pin
V_{ADC} = (I x R + offset) x amp

Figure 29. Current measurement

5.2.3 Angle scale

The angles, such as rotor position, are represented as 16-bit signed fractional values in the range <–1, 1), which corresponds to the angle in the range <–π, π). In a 16-bit signed integer value, the angle is represented as follows:

$$-\pi = 0x8000$$

Equation 34

$$\pi = 0x7FFF$$

Equation 35

5.3 Application overview

The PFC dual motor control application (3-in-1 application) controls three active objects on a single digital signal controller (DSC): two permanent magnet synchronous motors (PMSM) and the power factor correction circuitry (PFC). They all run on a single DSC having only one core. Thus the control algorithms must be properly synchronized.

5.4 Motor control algorithms synchronization

First of all, the two motor control algorithms must be synchronized. The PWM modules have the same frequency of 10 kHz in this application. The MC56F84789 DSC has a great calculation capacity as it runs at 100 MHz; therefore it is enough to lag the algorithms of the second motor by only 90 degrees in time. The purpose of the 90-degree lag is to sequence the current consumption by the motor inverters from the DC-bus capacitor so as to lower the probability of the same time active energy consumption.

To generate the 90-degree lag is a quite simple process. The PWM of motor 1 is started and when it reaches 90 degrees, an interrupt is generated where the PWM of motor 2 is started.

The algorithms of both the motors are calculated when the ADC measures their currents. So as the current measurement is synchronized to the PWM signals, the algorithms' routine call of both the motors also has the lag of 90 degrees.

Figure 30 shows synchronization of the PWM signals. The user can observe the bottom MOSFET PWM signals of the motors, the points where the desired quantities are measured, and the points where the algorithms are calculated. The detail description of how to configure and synchronize the peripherals to properly drive two PMS motors is delivered in *AN4608: Use of PWM and ADC on MC56F84789 to Drive Dual PMS Motor FOC*, available on **freescale.com**.

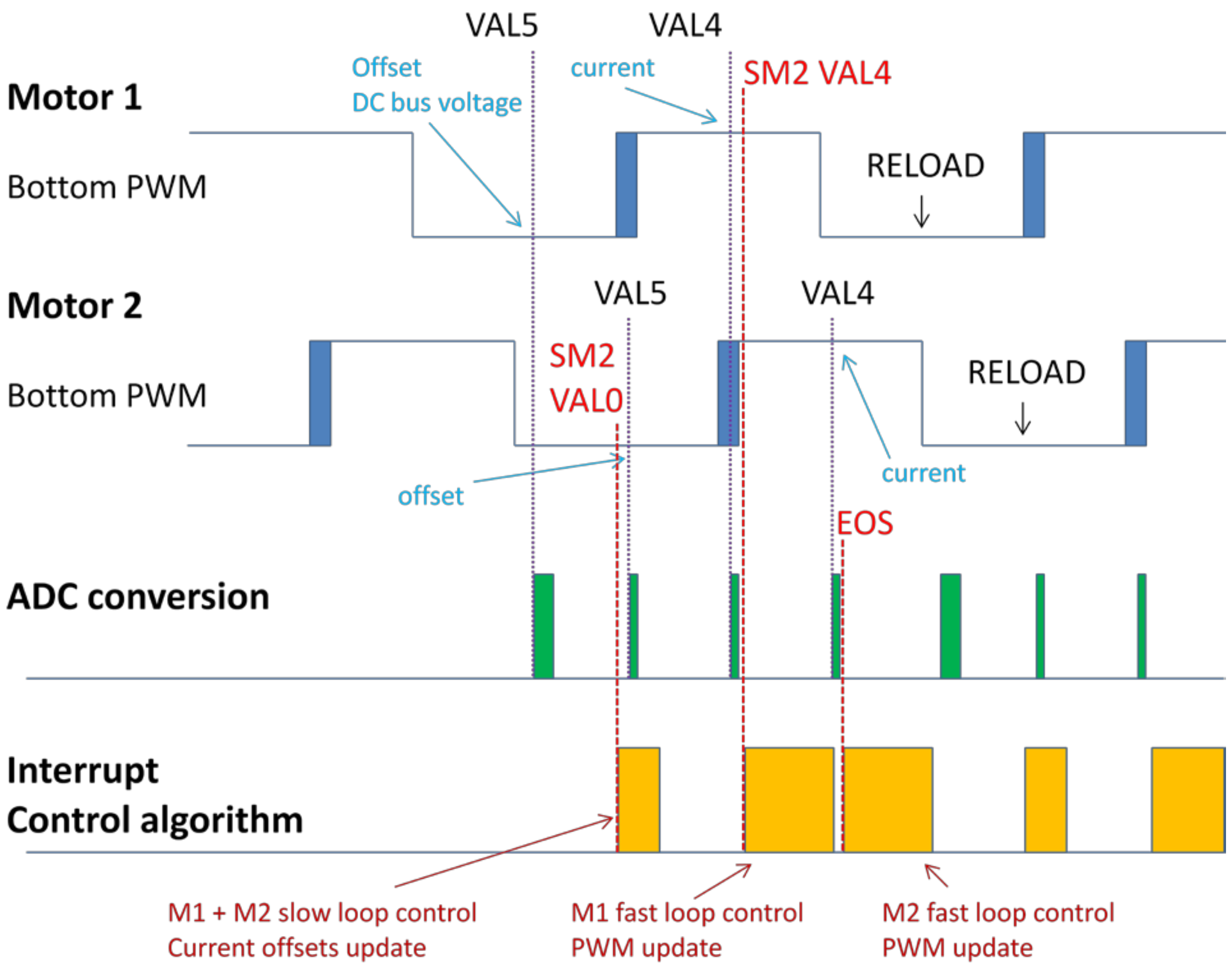


Figure 30. Algorithms synchronization of two PMSMs

5.5 PFC control algorithms synchronization

When the motors are synchronized, the PFC PWM has to be synchronized with the two motors' PWMs. The PWM frequency is 80 kHz (8 times higher than the motors' PWM frequency) but the PFC control algorithm is called every fourth cycle, that is, at 20 kHz (2 times more often than the motor control algorithm).

From Figure 30, it is visible that there are some gaps where the PFC control algorithm can be put in order not to interfere with the motor control algorithms too much. One gap is before the M1 fast loop calculation and the other, after the M2 fast loop calculation. The PFC algorithm has higher priority than the motor control algorithm. Therefore, in case of overlay, the motor control algorithm is interrupted or must wait the termination of the PFC control algorithm.

The synchronization is made as simply as the synchronization of the motor PWMs. The PWM interrupt of motor 1 is programmed to the point which is lagged 15 μs after the PWM half cycle (reload) point. This delay has been found experimentally when observing the position of the motor control and PFC algorithms on the scope. Figure 31 shows the PFC control algorithm synchronization to the PFC PWM signals.

The detailed description of how to configure and synchronize the peripherals to properly drive the PFC can be found in *AN4583:MC56F84789 Peripherals Synchronization for Interleaved PFC Control* available on **freescale.com.**

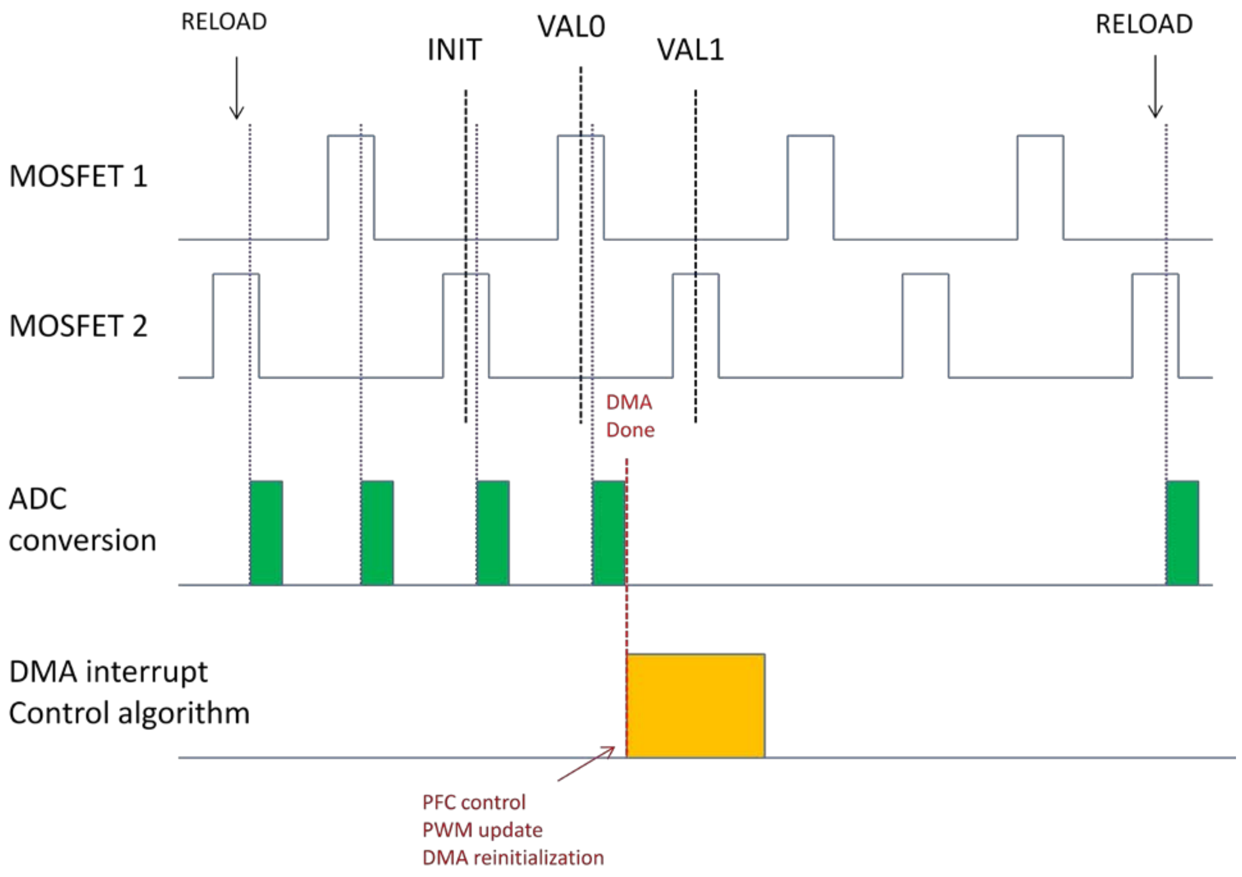


Figure 31. PFC control algorithm synchronization

5.6 Algorithms call

The application is interrupt-driven, running in real-time. There are several periodic-interrupt service routines executing the major application control tasks.

The **DMA channel 0 done** interrupt service routine is executed when the DMA channel 0 finishes the predefined number of transfers, see Figure 31.

- The DMA transfers the results from the 16-bit ADC C module to the buffer. This ADC is synchronized to the PFC PWM, that is, PWM A submodule 3.
- This interrupt has the highest priority (no. 2) and is called with a 50 µs period to calculate the PFC control algorithm. The PFC state machine logic is processed in this interrupt too. See Figure 32.

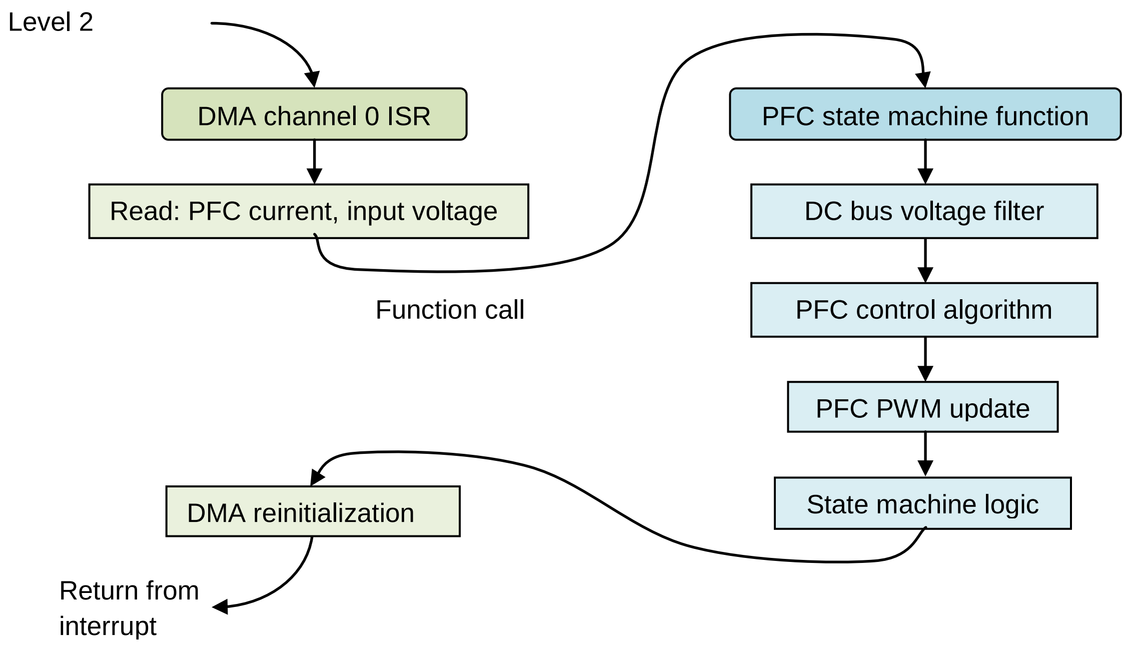


Figure 32. DMA channel 0 ISR

The **PWM A submodule 2 compare** interrupt service routine is executed when the submodule reaches the VAL4 (slightly after the current measurement point for motor 1).

- This interrupt has the priority 1. When this interrupt is entered, the phase currents of motor 1 are ready to be read and the PMSM FOC algorithm is calculated.
- If the ADC end-of-scan flag is generated before leaving this interrupt, it means the phase currents of motor 2 are ready, the interrupt is not left and the routine reads the measured currents of motor 2 and calculates the PMSM FOC algorithm. In this case, the ADC end-of-scan interrupt is cleared. The purpose of this is to save time of entering and leaving another interrupt. Therefore, both motors are calculated in one interrupt service routine. See Figure 33.

The **ADC end-of-scan interrupt** service routine is executed when the 12-bit ADC finishes the conversion of phase currents of motor 2.

- This interrupt has the priority 1. When this interrupt is entered, the phase currents of motor 2 are ready to be read and the PMSM algorithm is calculated.
- This interrupt is entered if the PWM A submodule 2 interrupt service routine terminates before the ADC conversion is complete. This happens when the motor 1 is stopped; that is, its algorithm calculation is not long. See Figure 34.

The **PWM B sub-module 0** interrupt service routine is executed when the submodule reaches the VAL0 (slightly before the current offset measurement point for motor 1).

- This interrupt has the priority 1. When this interrupt is entered, the DC-bus voltage, the offset of both motors’ current channels, and the three temperatures are read.
- The offsets are filtered and updated before the next phase current measurement. The motor 1 and 2 slow loop algorithms are calculated here.
- The motor 1 and 2 and the application state machine logic is processed in this interrupt.
- The data for the communication are recorded here and the commands from the communication state machine are applied here too. See Figure 35.

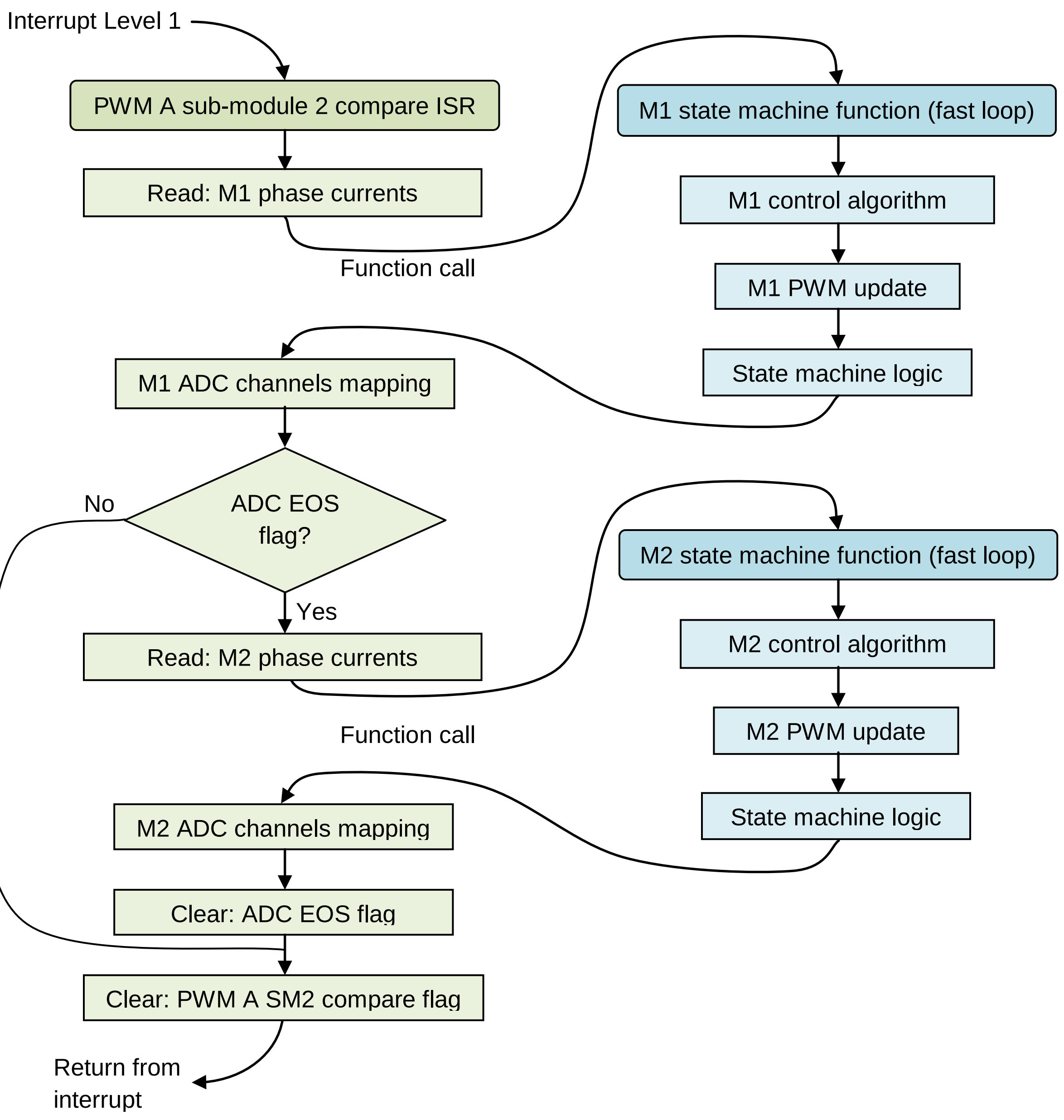


Figure 33. PWM A submodule 2 compare ISR

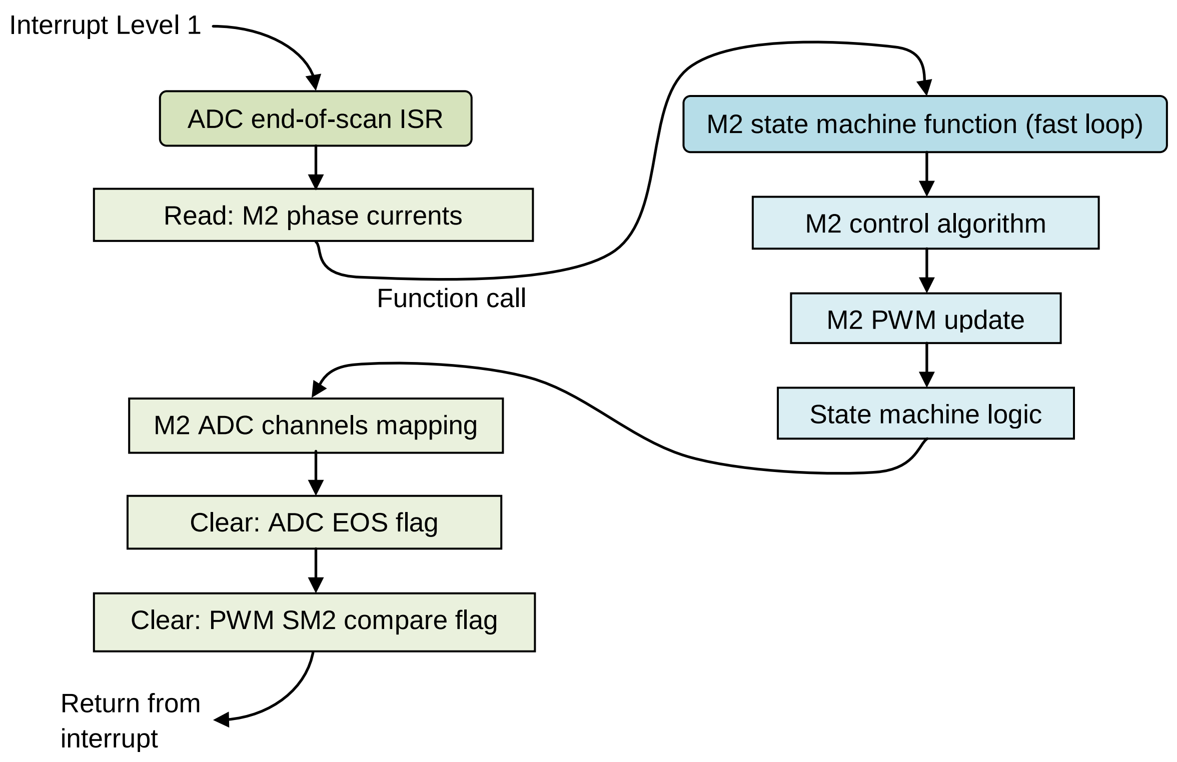


Figure 34. ADC end-of-scan ISR

The **PWM A submodule 1** interrupt service routine is executed when the submodule reaches the VAL5 value.

- At this point, the PWM B module of motor 2 is started, and this interrupt source is disabled.
- Next time, this interrupt is entered when the submodule reaches the VAL4 value.
- At this point, the PFC PWM A submodule 3 is started, and this interrupt source is disabled. The priority is 2 and this interrupt serves to synchronize the peripherals after reset.

The **PWM B submodule 0** interrupt service routine is executed when the submodule reaches the VAL4 value.

- At this point, the ADC triggers are enabled. It is necessary to get the correct sequence of measurement. This interrupt source is disabled.
- The priority is 2 and this interrupt serves to synchronization of peripherals after reset.

The **PWM A submodule 3** interrupt service routine is executed when the submodule reaches the VAL1 value.

- At this point, the Timer A3 mode is reconfigured. See *AN4583*: *MC56F84789 Peripherals Synchronization for Interleaved PFC Control*, available on **freescale.com**.
- The priority is 2 and this interrupt serves to synchronization of peripherals after reset.

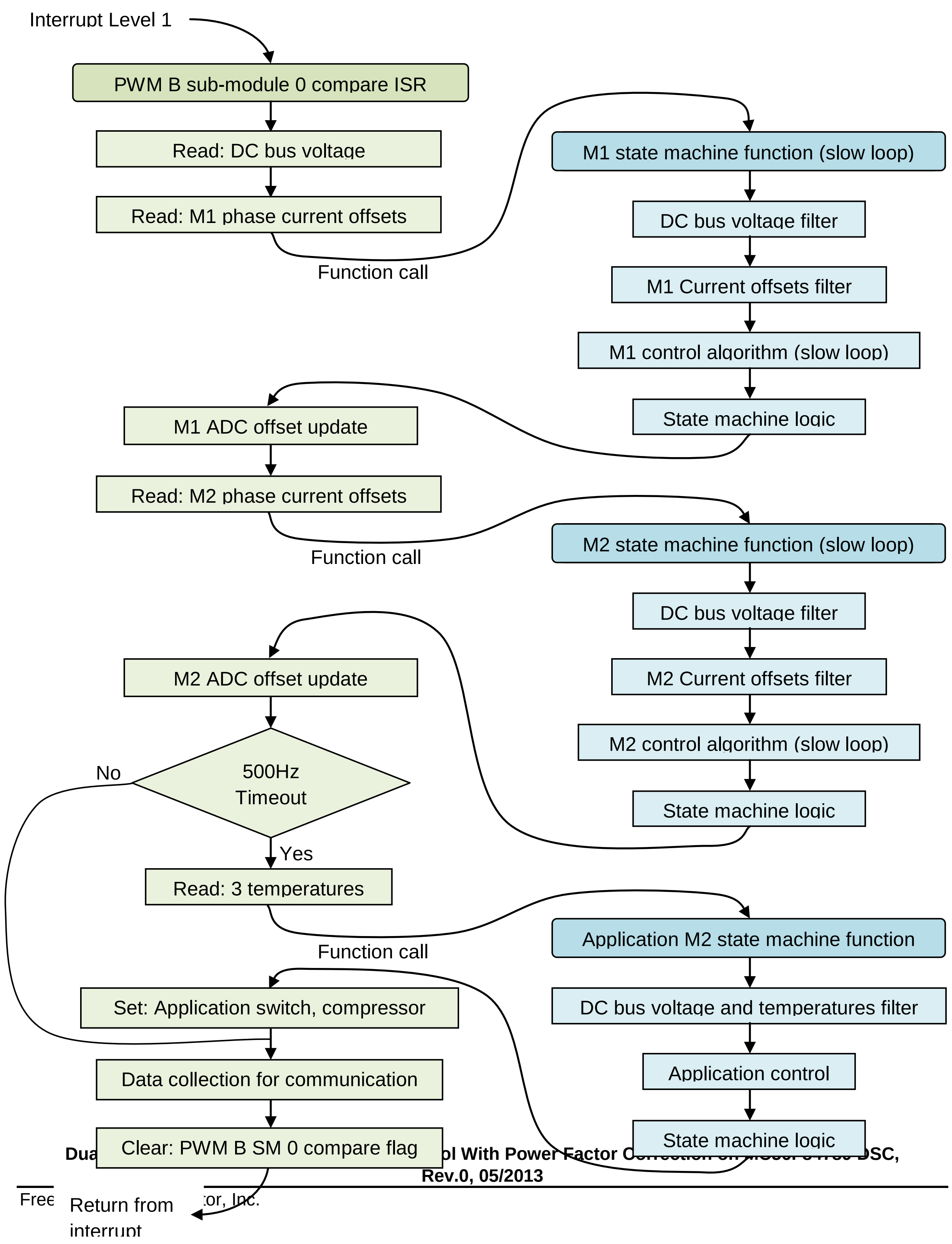

Interrupt Level 1
PWM B sub-module 0 compare ISR
Read: DC bus voltage
Read: M1 phase current offsets
Function call
M1 state machine function (slow loop)
DC bus voltage filter
M1 Current offsets filter
M1 control algorithm (slow loop)
State machine logic
M1 ADC offset update
Read: M2 phase current offsets
Function call
M2 state machine function (slow loop)
DC bus voltage filter
M2 Current offsets filter
M2 control algorithm (slow loop)
State machine logic
M2 ADC offset update
No
500Hz Timeout
Yes
Read: 3 temperatures
Function call
Application M2 state machine function
DC bus voltage and temperatures filter
Application control
State machine logic
Set: Application switch, compressor
Data collection for communication
Clear: PWM B SM 0 compare flag
Return from interrupt

Figure 35. PWM B submodule 0 ISR

The **DMA channel 3 done** interrupt service routine is executed when the DMA channel 3 finishes the predefined number of transfers from the buffer to the UART data register.

- This interrupt priority is 0 (the lowest) and serves to clear the transmission-in-progress flag. So this interrupt arrives at the end of data buffer transmission.
- If the UART Tx interrupt is used, this interrupt is not used.

The **SCI0 Tx** interrupt service routine is executed when the UART data register is empty.

- This interrupt priority is 0 (the lowest) and serves to send the data buffer to UART. At the last byte, it clears the transmission-in-progress flag.
- If the UART uses the DMA channel, this interrupt is not used.

The **background** loop is executed in the main application.

- It handles non-critical timing tasks, such as the communication state machine which reads the data received from the UART and prepares data to be sent to the UART.
- The command to switch on/off the application and set the compressor speed is created as the output of the communication state machine function. See Figure 36.

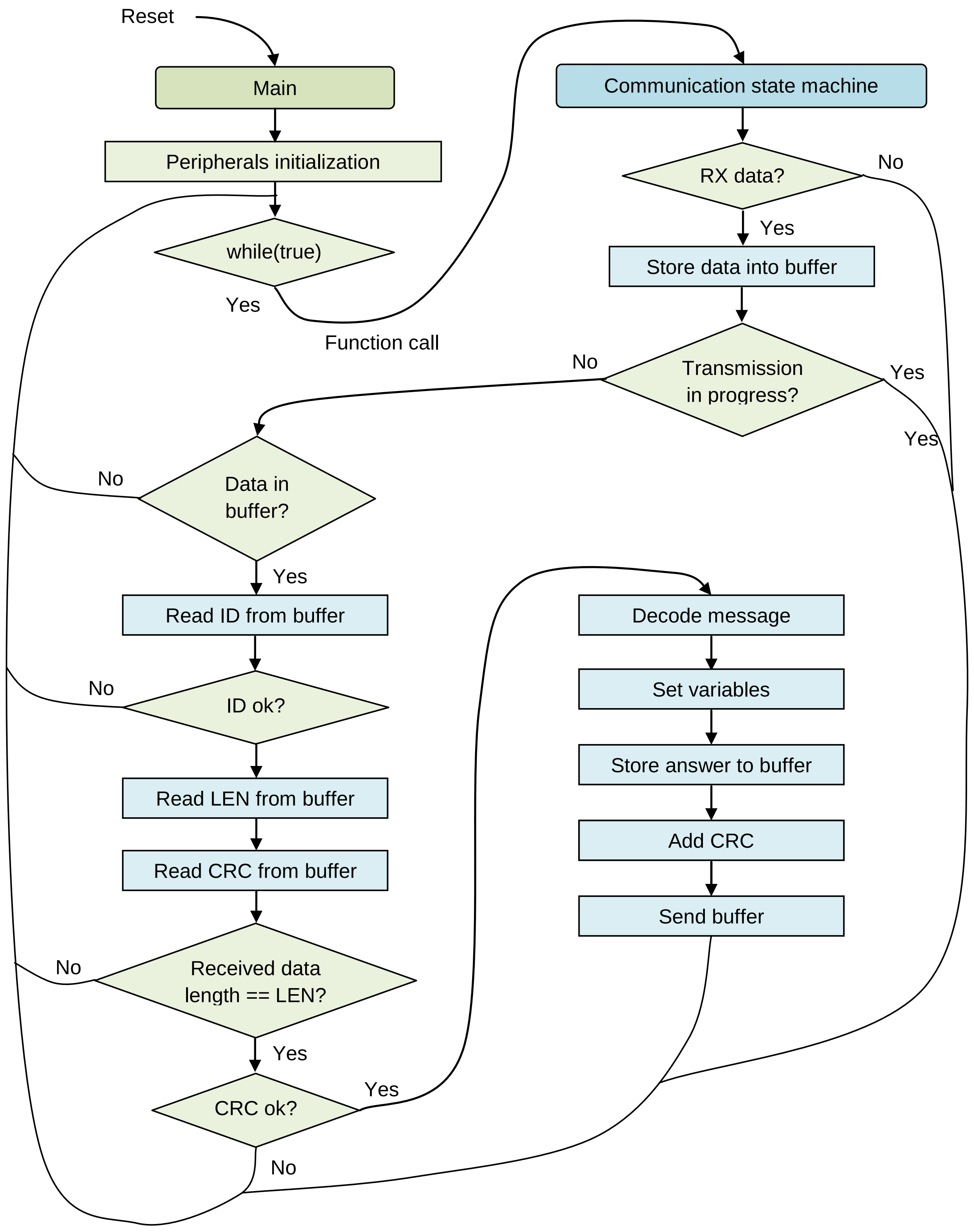


Figure 36. Background loop and communication

As mentioned above, the interrupt service routines call the state machine functions. The state machine function depends on the particular state. In dependence on this state, the particular state machine function is called which will take care of the correct state machine logic and control algorithm call. The state machine is described in the following sections.

5.7 Main state machine

The state machine structure has been unified for all components of the application and consists of four main states. See Figure 37. The states are the following:

- **Fault**—system faced a fault condition
- **Init**—variables initialization
- **Stop**—system is initialized and waiting for the Run command.
- **Run**—system is running; can be stopped by the Stop command.

There are transition functions between these state functions:

- **Init** > **Stop**—initialization has been done, the system is entering the Stop state.
- **Stop** > **Run**—the Run command has been applied, the system is entering the Run state if the Run command has been acknowledged.
- **Run** > **Stop**—the Stop command has been applied, the system is entering the Stop state if the Stop command has been acknowledged.
- **Fault** > **Init**—fault flag has been cleared, the system is entering the Init state.
- **Init, Stop, Run** > **Fault**—a fault condition has occurred, the system is entering the Fault state.

The state machine structure uses the following flags to switch between the states:

- **SM_CTRL_INIT_DONE** —when this flag is set, the system goes from the Init to the Stop state.
- **SM_CTRL_FAULT**—when this flag is set the system goes from any state to the Fault state.
- **SM_CTRL_FAULT_CLEAR** —when this flag is set, the system goes from the Fault state to the Init state.
- **SM_CTRL_START**—this flag informs the system that there is a command to go from the Stop state to the Run state. The transition function is called; nevertheless the action must be acknowledged. The reason is that sometimes it can take time before the system gets ready to be switched on.
- **SM_CTRL_RUN_ACK**—this flag acknowledges that the system can proceed from the Stop state to the Run state.

- **SM_CTRL_STOP**—this flag informs the system that there is a command to go from the Run state to the Stop state. The transition function is called nevertheless the action must be acknowledged. The reason is the system must be properly turned off which can take time.
- **SM_CTRL_STOP_ACK**—this flag acknowledges that the system can proceed from the Run state to the Stop state.

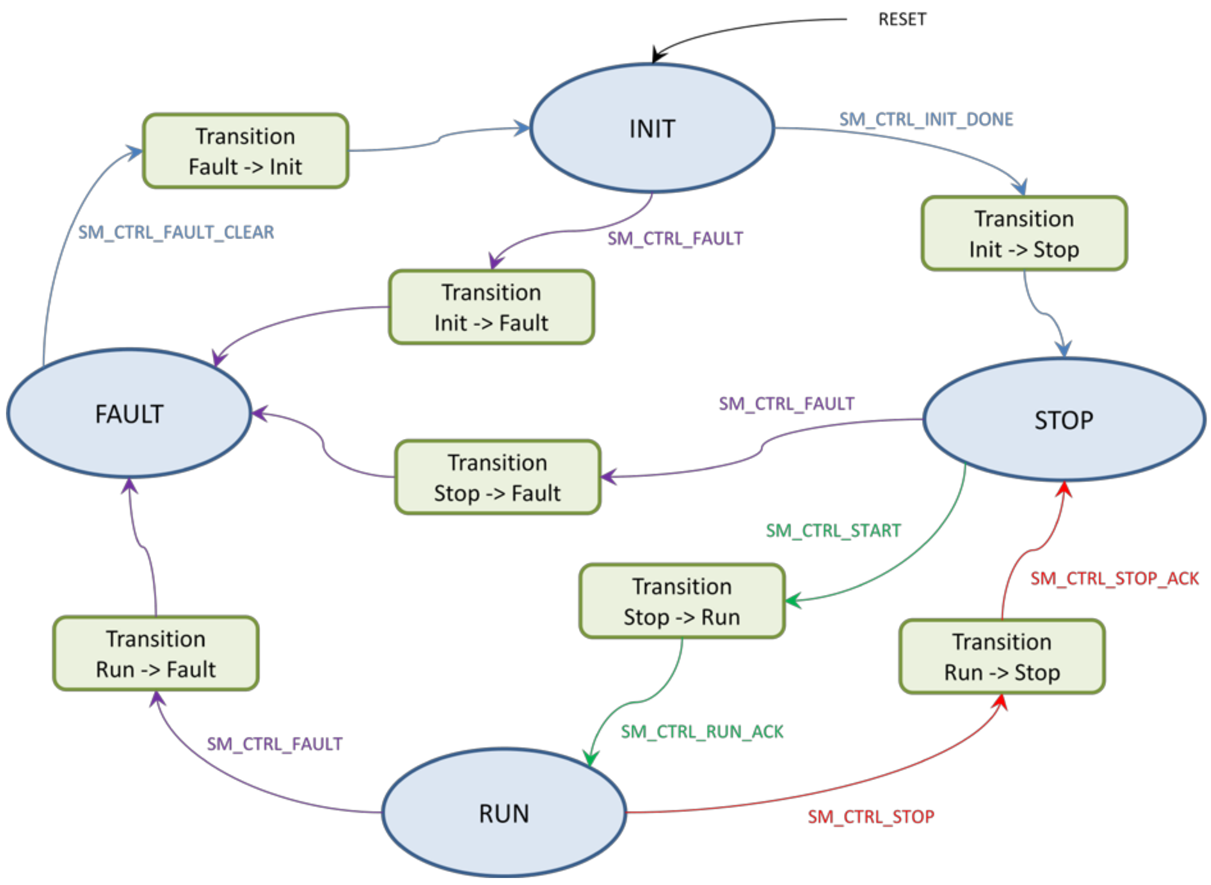


Figure 37. Main state machine diagram

The implementation of this structure of state machine is done in the state_machine.c .h files. The following code lines define the state machine structure:

```
/* State machine control structure */
typedef struct
{
    __pmem SM_APP_STATE_FCN_T const*      psState; /* State functions */
    __pmem SM_APP_TRANS_FCN_T const*      psTrans; /* Transition functions */
    SM_APP_CTRL                           uiCtrl;  /* Control flags */
    SM_APP_STATE_T                        eState; /* State */
} SM_APP_CTRL_T;
```

The four variables used in the code are defined as follows.

- *psState*—pointer to the user state machine functions. The particular state machine function from this table is called when the state machine is in a specific state.
- *psTrans*—pointer to the user transient functions. The particular transient function is called when the system goes from one state to another.
- *uiCtrl*—this variable is used to control the state machine behavior using the above mentioned flags.
- *eState*—this variable determines the actual state of the state machine

The user state machine functions are defined in the following structure:

```
/* User state machine functions structure */
typedef struct
{
      PFCN_VOID_VOID    Fault;
      PFCN_VOID_VOID    Init;
      PFCN_VOID_VOID    Stop;
      PFCN_VOID_VOID    Run;
} SM_APP_STATE_FCN_T;
```

The user transient state machine functions are defined in the following structure:

```
/* User state-transition functions structure*/
typedef struct
{
      PFCN_VOID_VOID    FaultInit;
      PFCN_VOID_VOID    InitFault;
      PFCN_VOID_VOID    InitStop;
      PFCN_VOID_VOID    StopFault;
      PFCN_VOID_VOID    StopInit;
      PFCN_VOID_VOID    StopRun;
      PFCN_VOID_VOID    RunFault;
      PFCN_VOID_VOID    RunStop;
} SM_APP_TRANS_FCN_T;
```

The control flag’s variable has the following definitions:

```
typedef unsigned short SM_APP_CTRL;

/* State machine control command flags */
#define SM_CTRL_NONE          0x0
#define SM_CTRL_FAULT         0x1
#define SM_CTRL_FAULT_CLEAR   0x2
#define SM_CTRL_INIT_DONE     0x4
#define SM_CTRL_STOP          0x8
#define SM_CTRL_START         0x10
#define SM_CTRL_STOP_ACK      0x20
#define SM_CTRL_RUN_ACK       0x40
```

The state identification variable has the following definitions:

```
/* Application state identification enum */
typedef enum {
    FAULT           = 0,
    INIT            = 1,
    STOP            = 2,
    RUN             = 3,
} SM_APP_STATE_T;
```

The state machine must be periodically called from the code using the following inline function. This function input is the pointer to the above-described state machine structure. This structure is declared and initialized in the code where the state machine is called:

```
/* State machine function */
extern inline void SM_StateMachine(SM_APP_CTRL_T *sAppCtrl)
{
      gSM_STATE_TABLE[sAppCtrl -> eState](sAppCtrl);
}
```

The particular example how to initialize and use the state machine structure will be shown in the state machine application for the motors, PFC, and the application.

5.8 Motor state machine

The motor state machines are based on the main state machine structure. The Run state sub-states have been added on top of the main structure to control the motors properly. Following is the description of the main states' user functions.

- **Fault**—the system faces a fault condition and waits until the fault flags are cleared. In case of motor 1's startup fail or overload, a pressure relaxation time (4 min) is defined before the system retries to start the motor again. It is due to the high pressure in the system where the motor can't start. The DC-bus voltage is measured and filtered.
- **Init**—variables initialization
- **Stop**—the system is initialized and is waiting for the Run command. The PWM output is disabled. The DC-bus voltage is measured and filtered.
- **Run**—the system is running and can be stopped by the Stop command. The Run sub-state functions are called from here.

There are transition functions between these state functions:

- **Init** > **Stop**—nothing is processed in this function.
- **Stop** > **Run**:
    - The duty cycle is initialized to 50 %
    - PWM output is enabled.

    - The current ADC channels are initialized.
    - The Calib sub-state is set as the initial Run sub-state.
- **Run > Stop**—The Stop command has been applied and the system is entering the Stop state if the Stop command has been acknowledged. The system does not go directly to Stop if the system is in certain Run sub-states.
- **Fault > Init**—nothing is processed in this function
- **Init, Stop > Fault**:
    - The PWM output is disabled.
- **Run > Fault**:
    - Certain current and voltage variables are zeroed.
    - The PWM output is disabled.
    - In case of motor 1's startup fail or overload, the pressure relaxation time is initialized.

The Run sub-states are called when the state machine is in the Run state. The Run sub-state functions are the following:

- **Calib**: The current channels ADC offset calibration.
    - After the time expires, the system is switched to Ready.
    - The DC-bus voltage is measured and filtered.
    - The PWM is set to 50% and its output is enabled.
- **Ready**:
    - The PWM is set to 50 % and its output is enabled.
    - The current is measured and the ADC channels are set up.
    - Certain variables are initialized.
- **Align**:
    - The current is measured and the ADC channels are set up.
    - The rotor alignment algorithm is called and the PWM is updated.
    - After the alignment time expires, the system is switched to Startup.
    - The DC-bus voltage is measured and filtered.
    - The current channels offset is measured and filtered.
- **Startup**:
    - The current is measured and the ADC channels are set up.
    - The BEMF observer algorithm is called to estimate the speed and position.
    - The FOC algorithm is called and the PWM is updated.
    - If the startup is successful, the system is switched into Spin, otherwise to Freewheel.
    - The DC-bus voltage is measured and filtered.
    - The current channels offset is measured and filtered.
    - The open-loop startup algorithm is called.
    - The estimated speed is filtered (in case of motor 1).
- **Spin**:
    - The current is measured and the ADC channels are set up.
    - The BEMF observer algorithm is called to estimate the speed and position.

    - The FOC algorithm is called and the PWM is updated.
    - The motor spins.
    - The DC-bus voltage is measured and filtered.
    - The current channels offset is measured and filtered.
    - The estimated speed is filtered (in case of Motor 1).
    - The speed ramp, field-weakening and the speed PI(D) controller algorithms are called.
    - The speed command and speed overload condition are evaluated.
- **Freewheel**:
    - The PWM output is disabled and the module is set to 50 %.
    - The current is measured and the ADC channels are set up. The DC-bus voltage is measured and filtered.
    - The current channels offset is measured and filtered.
    - The system waits in this sub-state for certain time which is generated due to rotor inertia, it means to wait until the rotor stops itself.
    - Then the system evaluates the conditions and proceeds into one of these sub-states: Align or Ready. In case of several faulty starts, the system goes into the Fault state.

The Run sub-states also have the transition functions that are called in between the sub-states' transition. The sub-state transition functions are the following:

- **Calib** > **Ready**—calibration done, entering the Ready state.
- **Ready** > **Align**—non-zero speed command; entering the Align state.
    - Certain variables are initialized (voltage, speed, position)
    - The startup counter is set to 1.
    - The alignment time is set up.
- **Align** > **Ready**—zero speed command; entering the Ready state.
    - Certain voltage and current variables are zeroed.
    - The PWM is set to 50 %.
- **Align** > **Startup**—alignment done; entering the Startup state.
    - The filters and control variables are initialized.
    - The PWM is set to 50 %.
- **Startup** > **Spin**—startup successful; entering the Spin state.
- **Startup** > **Freewheel**—startup failed; entering the Freewheel state.
    - Certain variables are initialized (voltage, speed, position),
    - The startup counter is set to 1.
    - The freewheel time is set up.
- **Spin** > **Freewheel**—zero speed command; entering the Freewheel state.
    - Certain variables are initialized (voltage, speed, position),
    - The startup counter is set to 1.
    - The freewheel time is set up.
- **Freewheel** > **Ready**—zero-speed command; entering the Ready state.
    - The PWM output is enabled.
- **Freewheel** > **Align**—non-zero speed command; entering the Align state.
    - The PWM output is enabled.
    - Certain variables are initialized (voltage, speed, position).

o The alignment time is set up.

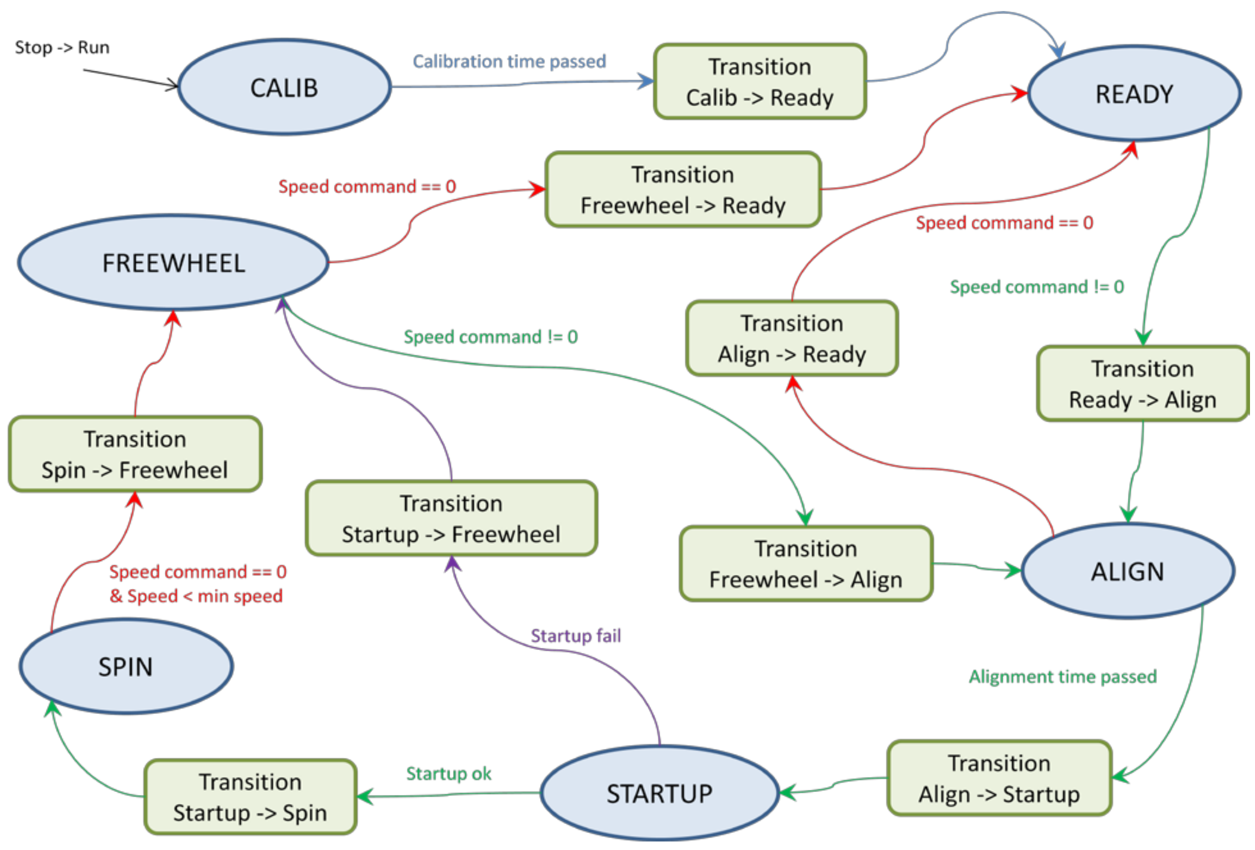

Stop -> Run
CALIB
Calibration time passed
Transition Calib -> Ready
READY
Speed command == 0
Transition Freewheel -> Ready
FREEWHEEL
Speed command == 0
Speed command != 0
Speed command != 0
Transition Align -> Ready
Transition Ready -> Align
Transition Spin -> Freewheel
Transition Startup -> Freewheel
Transition Freewheel -> Align
ALIGN
Speed command == 0 & Speed < min speed
SPIN
Startup fail
Alignment time passed
Transition Startup -> Spin
Startup ok
STARTUP
Transition Align -> Startup

Figure 38. Motor Run sub-state diagram

The implementation of this structure of motor state machine is done in the *M1_statemachine.c .h* and *M2_statemachine.c .h* files. The main state-machine structure of motor 1 is the following:

The main states' user function prototypes:

```
static void M1_StateFault(void);
static void M1_StateInit(void);
static void M1_StateStop(void);
static void M1_StateRun(void);
```

The main states' user transient function prototypes:

```
static void M1_TransFaultInit(void);
static void M1_TransInitFault(void);
static void M1_TransInitStop(void);
static void M1_TransStopFault(void);
static void M1_TransStopInit(void);
static void M1_TransStopRun(void);
static void M1_TransRunFault(void);
static void M1_TransRunStop(void);
```

The main states functions table initialization:

```
/* State machine functions field (in pmem) */
__pmem static const SM_APP_STATE_FCN_T msSTATE = {M1_StateFault,
M1_StateInit, M1_StateStop, M1_StateRun};
```

The main state transient functions table initialization:

```
/* State-transition functions field (in pmem) */
__pmem static const SM_APP_TRANS_FCN_T msTRANS = {M1_TransFaultInit,
M1_TransInitFault, M1_TransInitStop, M1_TransStopFault, M1_TransStopInit,
M1_TransStopRun, M1_TransRunFault, M1_TransRunStop};
```

Finally, the main state machine structure initialization:

```
/* State machine structure declaration and initialization */
SM_APP_CTRL_T gsM1_Ctrl =
{
      /* gsM1_Ctrl.psState, User state functions  */
      &msSTATE,

      /* gsM1_Ctrl.psTrans, User state-transition functions */
      &msTRANS,

      /* gsM1_Ctrl.uiCtrl, Deafult no control command */
      SM_CTRL_NONE,

      /* gsM1_Ctrl.eState, Default state after reset */
      INIT
};
```

Similarly, the Run sub-state machine is declared. Thus, the Run sub-state identification variable has the following definitions:

```
typedef enum {
    CALIB     = 0,
    READY     = 1,
    ALIGN     = 2,
    STARTUP   = 3,
    SPIN      = 4,
    FREEWHEEL = 5,
} M1_RUN_SUBSTATE_T;         /* Run sub-states */
```

For the Run sub-states, two sets of user functions are defined. One set is called when the fast loop control is calculated and the other is called when the slow loop control is calculated. The user function prototypes of the Run substates are as follows.

```
static void M1_StateRunCalib (void);
static void M1_StateRunReady(void);
static void M1_StateRunAlign(void);
static void M1_StateRunStartup(void);
static void M1_StateRunSpin(void);
static void M1_StateRunFreewheel(void);
static void M1_StateRunCalibSlow(void);
static void M1_StateRunReadySlow(void);
static void M1_StateRunAlignSlow(void);
static void M1_StateRunStartupSlow(void);
static void M1_StateRunSpinSlow(void);
static void M1_StateRunFreewheelSlow(void);
```

The Run sub-states' user transient function prototypes are given below.

```
static void M1_TransRunCalibReady(void);
static void M1_TransRunReadyAlign(void);
static void M1_TransRunAlignStartup(void);
static void M1_TransRunAlignReady(void);
static void M1_TransRunStartupSpin(void);
static void M1_TransRunStartupFreewheel(void);
static void M1_TransRunSpinFreewheel(void);
static void M1_TransRunFreewheelAlign(void);
static void M1_TransRunFreewheelReady(void);
```

The Run sub-states functions table initialization:

```
/* Sub-state machine functions field (in pmem) */
__pmem static const PFCN_VOID_VOID mM1_STATE_RUN_TABLE[6][2] =
{
      {M1_StateRunCalib, M1_StateRunCalibSlow},
      {M1_StateRunReady, M1_StateRunReadySlow},
      {M1_StateRunAlign, M1_StateRunAlignSlow},
      {M1_StateRunStartup, M1_StateRunStartupSlow},
      {M1_StateRunSpin, M1_StateRunSpinSlow},
      {M1_StateRunFreewheel, M1_StateRunFreewheelSlow}
};
```

The state machine is called from the interrupt service routine as mentioned in Algorithms call. There are two interrupts to call the state machine function—one for the fast loop control, and one for the slow loop control. The code syntax used to call the fast loop control state machine is given below.

```
/* Fast loop calculation */
geM1_StateRunLoop = FAST;

/* StateMachine call */
SM_StateMachine(&gsM1_Ctrl);
```

Similarly, the slow loop control state machine is called in the following way:

```
/* Slow loop calculation */
geM1_StateRunLoop = SLOW;

/* StateMachine call */
SM_StateMachine(&gsM1_Ctrl);
```

Inside the user Run state function, the sub-state functions are called as follows:

```
/* Run sub-state function */
mM1_STATE_RUN_TABLE[meM1_StateRun][geM1_StateRunLoop]();
```

where the first variable identifies the Run sub-state, and the second, the fast or slow loop.

For motor 2 the code is written in the same way. The only difference is that the variables used M2 instead of M1 in their names.

5.9 PFC state machine

The PFC state machine is based on the main state machine structure. The Run state sub-states have been added on top of the main structure to control the PFC properly. Following is the description of the main states' user functions.

- **Fault**—the system faces a fault condition and waits until the fault flags are cleared.
    - The PWM output is disabled.
    - The DC-bus voltage and the input voltage are measured and filtered.
- **Init**—variables initialization
- **Stop**—the system is initialized and waits for the Run command.
    - The PWM output is disabled.
    - The DC-bus voltage is measured and filtered.
    - The DC-bus voltage and the input voltage are measured and filtered.
    - The threshold voltage to detect the phase of the input voltage is calculated from the input voltage DC component.
- **Run**—the system is running and can be stopped by the Stop command.
    - The DC-bus voltage and the input voltage are measured and filtered.
    - The phase detection algorithm and the Run sub-state functions are called from here.

There are transition functions between these state functions:

- **Init** > **Stop**—nothing is processed in this function
- **Stop** > **Run:**
    - The duty cycle is initialized to 0 %;
    - The PWM output is enabled.
    - The control variables are initialized.
    - The calibration time is set up.
    - The PFC_Calib sub-state is set as the initial Run sub-state.
- **Run** > **Stop**:
    - The Stop command has been applied,
    - The system is entering the Stop state after acknowledged.
- **Fault** > **Init**—nothing is processed in this function.

- **Init, Stop -> Fault:**
  - The PWM output is disabled.

- **Run > Fault**:
  - The PWM output is disabled.
  - The PWM is set to 0.

The Run sub-states are called when the state machine is in the Run state. The Run sub-state functions are the following:

- **Calib** —the application is getting the phase of the power line voltage. If the input voltage frequency is within the permitted range, the system goes to Ready. Otherwise a fault flag is generated.
- **Ready**—PFC is in phase with the input line voltage and waits for the voltage command.
- **Run**—PFC is running. The fast (current) loop and slow (voltage) loop control algorithms are calculated. The voltage ramp algorithm is calculated.

The Run sub-states have also the transition functions that are called in between the sub-states' transition. The sub-state transition functions are the following:

- **Calib** > **Ready**—phasing done, entering the Ready state
- **Ready** > **Run**—non-zero voltage command; entering the Run state. Certain control variables are initialized.
- **Run** > **Ready**—zero voltage command; entering the Ready state. The PWM is set to 0. The required current and voltage variables are set to 0.

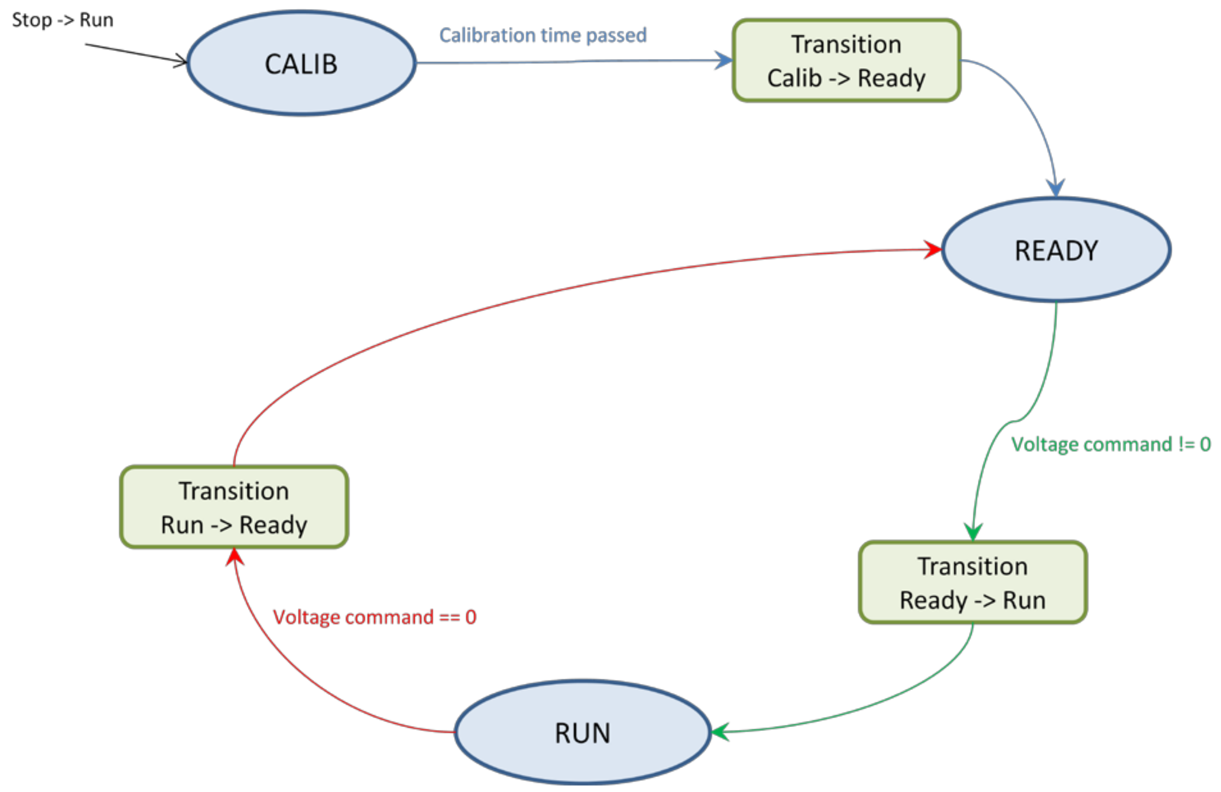


Figure 39. PFC Run sub-state diagram

The implementation of this structure of PFC state machine is done in the PFC_statemachine.c .h files. The main PFC's state-machine structure is the following:

The user function prototypes of the main states are given below.

```
static void PFC_StateFault(void);
static void PFC_StateInit(void);
static void PFC_StateStop(void);
static void PFC_StateRun(void);
```

The main states' user transient function prototypes are given below.

```
static void PFC_TransFaultInit(void);
static void PFC_TransInitFault(void);
static void PFC_TransInitStop(void);
static void PFC_TransStopFault(void);
static void PFC_TransStopInit(void);
static void PFC_TransStopRun(void);
static void PFC_TransRunFault(void);
static void PFC_TransRunStop(void);
```

The table initialization of main states functions can be done using the following code:

```
/* State machine functions field (in pmem) */
__pmem static const SM_APP_STATE_FCN_T msSTATE = {PFC_StateFault,
PFC_StateInit, PFC_StateStop, PFC_StateRun};
```

The table initialization of the main state transient functions can be done using the following code:

```
/* State-transition functions field (in pmem) */
__pmem static const SM_APP_TRANS_FCN_T msTRANS = {PFC_TransFaultInit,
PFC_TransInitFault, PFC_TransInitStop, PFC_TransStopFault, PFC_TransStopInit,
PFC_TransStopRun, PFC_TransRunFault, PFC_TransRunStop};
```

Finally, the main state machine structure initialization can be done using the following code:

```
/* State machine structure declaration and initialization */
SM_APP_CTRL_T gsPFC_Ctrl =
{
      /* gsPFC_Ctrl.psState, User state functions  */
      &msSTATE,

      /* gsPFC_Ctrl.psTrans, User state-transition functions */
      &msTRANS,

      /* gsPFC_Ctrl.uiCtrl, Default no control command */
      SM_CTRL_NONE,

      /* gsPFC_Ctrl.eState, Default state after reset */
      INIT
};
```

Similarly, the Run sub-state machine is declared. Thus, the Run sub-state identification variable has the following definitions:

```
typedef enum {
    PFC_CALIB = 0,
    PFC_READY = 1,
    PFC_RUN   = 2
} PFC_RUN_SUBSTATE_T;         /* Run sub-states */
```

The Run sub-state function prototypes are given below.

```
static void PFC_StateRunCalib(void);
static void PFC_StateRunReady(void);
static void PFC_StateRunRun(void);
```

The user transient function prototypes of Run sub-states are given below:

```
static void PFC_TransRunCalibReady(void);
static void PFC_TransRunReadyRun(void);
static void PFC_TransRunRunReady(void);
```

The table initialization of Run sub-states functions can be done using the following code:

```
/* Sub-state machine functions field (in pmem) */
__pmem static const PFCN_VOID_VOID mPFC_STATE_RUN_TABLE[6] =
{
```

```
            PFC_StateRunCalib,
            PFC_StateRunReady,
            PFC_StateRunRun
};
```

The state machine is called from the interrupt service routine as mentioned in Algorithms call. The way how to call the state machine is given below:

```
/* State machine call */
SM_StateMachine(&gsPFC_Ctrl);
```

5.10 Application state machine

The application state machine is also based on the main state machine structure. This part of the application is the master state machine that controls the PFC and motor state machines. It only has the four states:

- **Fault**—the system faces a fault condition and waits until the fault condition is cleared.
  - temperatures and DC-bus voltage measured and filtered
  - fault flags checked
  - motors and PFC forced to Stop
  - Relay off
  - LED flashing quickly
- **Init**—variables initialization
- **Stop**—system initialized; waiting for the Run command
  - temperatures and DC-bus voltage measured and filtered
  - fault flags checked
  - motors forced to Stop
  - Relay off
  - LED flashing slowly
- **Run**—system is running and can be stopped by the Stop command
  - PFC voltage command set; PFC running
  - Motor 1 and 2 run if the conditions are accomplished
  - temperatures and DC-bus voltage measured and filtered
  - fault flags checked
  - Relay on
  - LED flashing the “Smoke on the Water” main riff

There are transition functions between these state functions:

- **Init** > **Stop**—LED flashing program initialization
- **Stop** > **Run**—if DC-bus voltage is in 1 % proximity from the PFC voltage command, the Run command is acknowledged and the application moves to Run.

    - LED flash program is initialized.
- **Run** > **Stop**—if the PFC is not running, the Stop command is acknowledged and the application moves to Stop.
    - The relay is turned off.
    - LED flash program is initialized.
- **Fault** > **Init**—nothing is processed in this function.
- **Init, Stop** > **Fault**—motor 1 pressure relaxation time is set to 0.
    - LED flash program is initialized.
- **Run** -> **Fault**—if motor 1 is running, the pressure relaxation time is initialized.
    - The relay is turned off.
    - LED flash program is initialized.

The procedure when the application goes from the Stop to Run mode can be seen in Figure 40. When the Run command is applied, the PFC is turned on, and the PFC voltage command is applied. As soon as the PFC output voltage (DC bus voltage) reaches the desired voltage level, the application enters the Run mode.

The procedure when the application goes from the Run to Stop mode can be seen in Figure 40. It is more complex. If motor 1 is running, it is stopped with a down-ramp. Then the high-side temperature is checked. If the temperature is higher than 35 °C, the application waits until the temperature gets to the safe level. If the temperature is less than 35 °C, the motor 2 is stopped. Then the PFC is turned off. If the PFC state machine confirms the PFC is not operating, the application enters the Stop mode.

The procedure when the application goes from the Run to Fault mode can be seen in Figure 41. It uses similar approach as the procedure when it goes from Run to Stop.

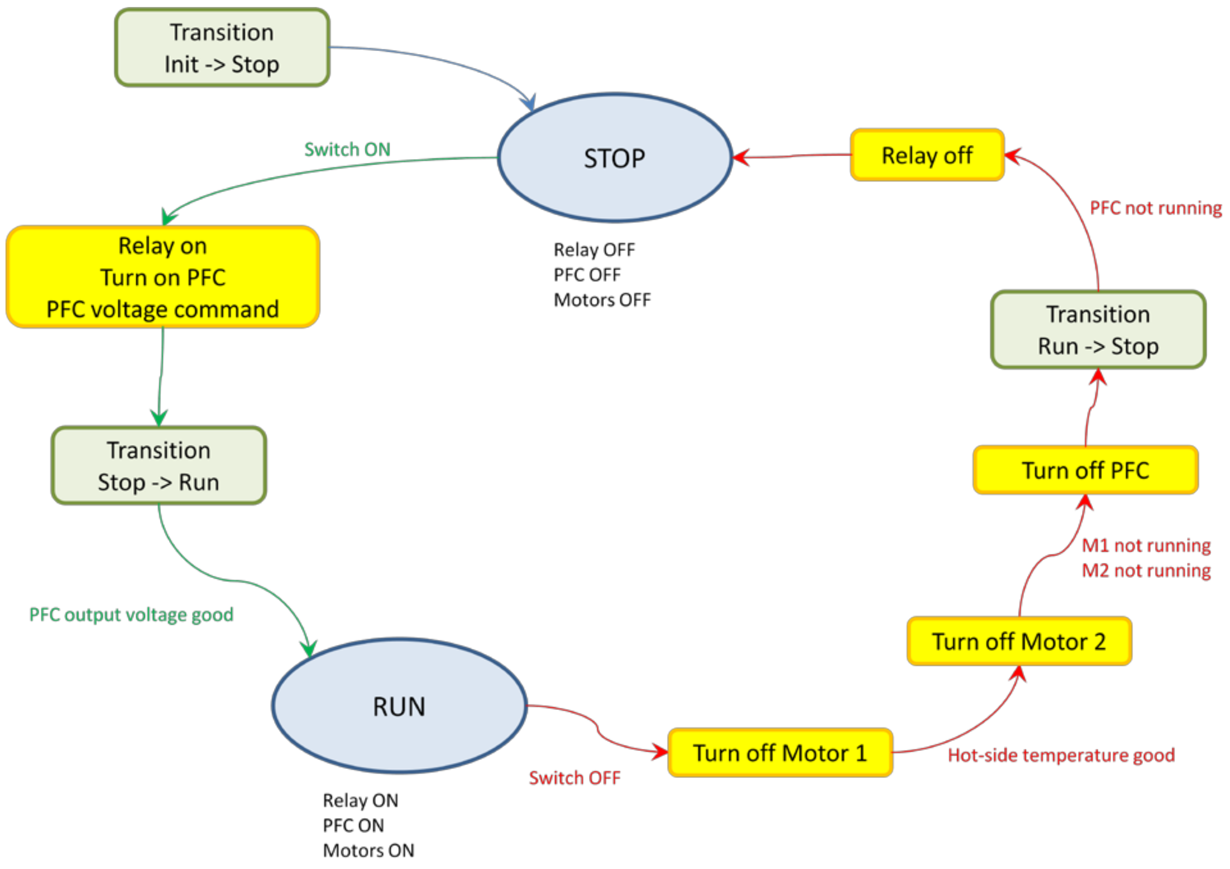


Figure 40. Application Stop/Run transition

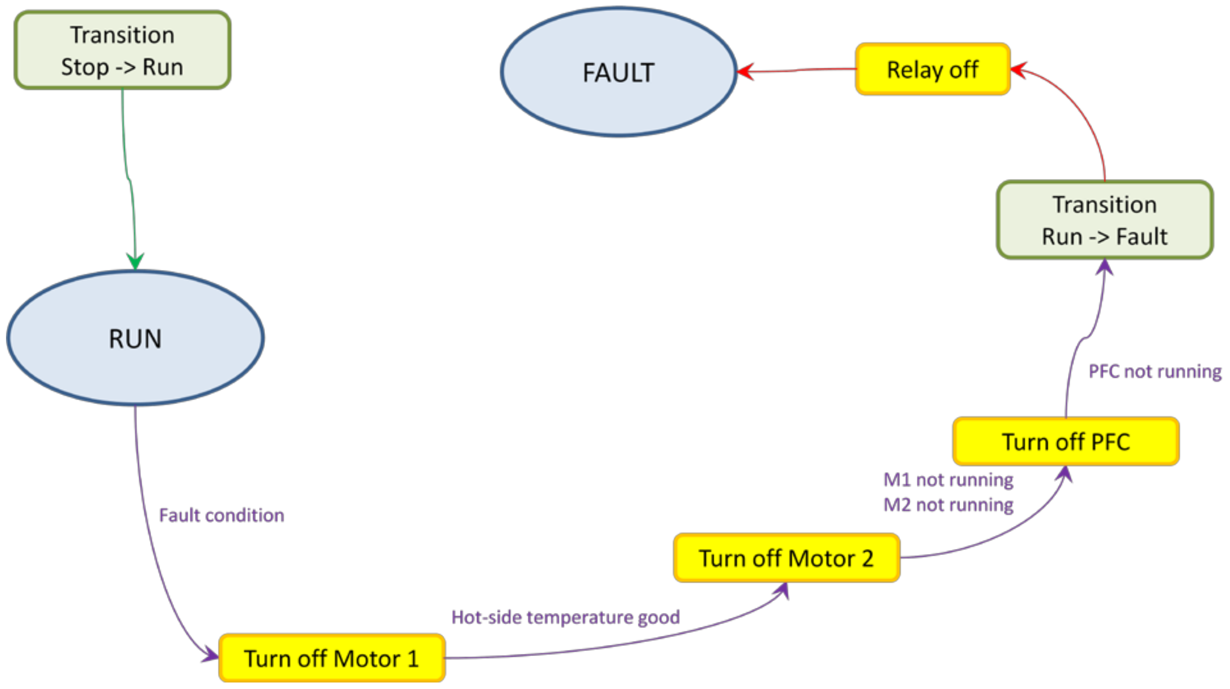


Figure 41. Application Run/Fault transition

The implementation of this structure uses the *APP_statemachine.c .h* files. The main application's state-machine structure is the following:

The user function prototypes of the main states are given below:

```
static void APP_StateFault(void);
static void APP_StateInit(void);
static void APP_StateStop(void);
static void APP_StateRun(void);
```

The user transient function prototypes of the main states are as follows:

```
static void APP_TransFaultInit(void);
static void APP_TransInitFault(void);
static void APP_TransInitStop(void);
static void APP_TransStopFault(void);
static void APP_TransStopInit(void);
static void APP_TransStopRun(void);
static void APP_TransRunFault(void);
static void APP_TransRunStop(void);
```

The main states functions table initialization:

```
/* State machine functions field (in pmem) */
__pmem static const SM_APP_STATE_FCN_T msSTATE = {APP_StateFault,
APP_StateInit, APP_StateStop, APP_StateRun};
```

The main state transient functions table initialization can be done as follows:

```
/* State-transition functions field (in pmem) */
__pmem static const SM_APP_TRANS_FCN_T msTRANS = {APP_TransFaultInit,
APP_TransInitFault, APP_TransInitStop, APP_TransStopFault, APP_TransStopInit,
APP_TransStopRun, APP_TransRunFault, APP_TransRunStop};
```

Finally, the main state machine structure initialization can be done using the following code:

```
/* State machine structure declaration and initialization */
SM_APP_CTRL_T gsAPP_Ctrl =
{
      /* gsAPP_Ctrl.psState, User state functions  */
      &msSTATE,

      /* gsAPP_Ctrl.psTrans, User state-transition functions */
      &msTRANS,

      /* gsAPP_Ctrl.uiCtrl, Default no control command */
      SM_CTRL_NONE,

      /* gsAPP_Ctrl.eState, Default state after reset */
      INIT
};
```

5.11 Sensorless PMS motor control

The application controls two PMSMs in sensorless mode. As the main principle of the control technique is the same for both the motors, the application uses only one set of routines for both the motors. The inputs to these routines are the structures of particular motors. This approach saves the necessary program flash memory in the application.

The following sections are dedicated to the motor control algorithm pieces.

5.11.1 Field-oriented control

The field-oriented control (FOC alias vector control) theory is described in Vector control of PM synchronous motor. The following description belongs to the FOC code implementation.

The FOC has been optimized into one function which has one input/output pointer to a structure. The prototype of the function is the following:

```
void MCSTRUC_FocPMSMCurrentCtrl(MCSTRUC_FOC_PMSM_T *psFocPMSM)
```

The structure referred by the input/output structure pointer is defined as follows:

```
 typedef struct
{
GFLIB_CONTROLLER_PI_P_PARAMS_T sIdPiParams; /* Id PI controller parameters */
GFLIB_CONTROLLER_PI_P_PARAMS_T sIqPiParams; /* Iq PI controller parameters */
GDFLIB_FILTER_IIR1_T    sIdZcFilter; /* D current zero-cancellation filter */
GDFLIB_FILTER_IIR1_T    sIqZcFilter; /* Q current zero-cancellation filter */
GDFLIB_FILTER_IIR1_T    sUDcBusFilter; /* Dc bus voltage filter */
MCLIB_3_COOR_SYST_T     sIABC; /* Measured 3-phase current */
MCLIB_2_COOR_SYST_ALPHA_BETA_T sIAlBe;    /* Alpha/Beta current */
MCLIB_2_COOR_SYST_D_Q_T        sIDQ;      /* DQ current */
MCLIB_2_COOR_SYST_D_Q_T        sIDQReq;   /* DQ required current */
MCLIB_2_COOR_SYST_D_Q_T        sIDQReqZc; /* DQ required current after zero-
cancelation */
MCLIB_2_COOR_SYST_D_Q_T        sIDQError;  /* DQ current error */
MCLIB_3_COOR_SYST_T            sDutyABC;   /* Applied duty cycles ABC */
MCLIB_2_COOR_SYST_ALPHA_BETA_T sUAlBeReq;  /* Required Alpha/Beta voltage */
MCLIB_2_COOR_SYST_ALPHA_BETA_T sUAlBeComp; /* Compensated to DC bus
Alpha/Beta volate */
MCLIB_2_COOR_SYST_D_Q_T        sUDQReq;         /* Required DQ voltage */
MCLIB_2_COOR_SYST_D_Q_T        sUDQController;  /* Required DQ voltage */
MCLIB_ANGLE_T      sAnglePosEl;     /* Electrical position sin/cos (at the
moment of PWM current reading) */
MCLIB_ANGLE_T      sAnglePosElUpdate; /* Compensated electrical position
sin/cos (at the moment of PWM update) */
MCSTRUC_ALIGNMENT_T sAlignment; /* Alignment structure params */
UWord16     uw16SectorSVM; /* SVM sector */
Frac16      f16DutyCycleLimit; /* Max. allowable dutycycle in frac */
Frac16      f16UDcBus;          /* DC bus voltage */
Frac16      f16UDcBusFilt;      /* Filtered DC bus voltage */
Int16       i16IdPiSatFlag;     /* Id PI controller saturation flag */
Int16       i16IqPiSatFlag;     /* Iq PI controller saturation flag */
bool        bOpenLoop;          /* Current control loop is open */
bool        bUseMaxBus;        /* Calculate the max. possible DQ current
controllers' output limits based on dc bus voltage */
bool        bUseZc;            /* User zero-cancellation filter */
} MCSTRUC_FOC_PMSM_T;
```

So this structure contains all the necessary variables or sub-structures for the field-oriented control algorithm implementation. The types used in this structure are defined in Freescale's Embedded Software Libraries (FSLESL). Here is a description of the used items in this application:

- D and Q current PI controllers—serves to control the D and Q current
- D and Q zero-cancellation—zero-cancellation is not used by this application
- Dc bus voltage 1st order IIR filter —serves to filter the DC-bus voltage
- A, B, C currents—measured 3-phase current; input to the algorithm
- Alpha, beta currents —currents transformed into the alpha/beta frame
- D, Q currents—currents transformed into the D/Q frame
- Required D, Q currents—required currents in the D/Q frame; input to the algorithm
- Required D, Q currents after zero-cancellation—not used by this application
- D, Q current error—error (difference) between the required and measured D/Q currents
- A, B, C duty cycles—3-phase duty cycles; output from the algorithm
- Required alpha, beta voltages—required voltages in the alpha/beta frame
- Compensated required alpha, beta voltages—the previous item recalculated on the actual level of the DC-bus voltage
- Required D, Q voltage—required voltages in the alpha/beta frame; outputs from the PI controllers
- Required D, Q voltage output—required voltages in the alpha/beta frame at the outputs from the current PI controllers
- Angle when current measured—electrical rotor angle (sine, cosine) at the moment where the ADC measured the currents
- Angle when PWM updated—electrical rotor angle (sine, cosine) at the moment where the PWM is updated
- Alignment—this sub-structure contains items used at the alignment; its detail description is in the chapter dedicated to the alignment.
- SVM sector—sector information; output from the SVM algorithm
- Duty cycle limit—this variable limits the maximum value of the A, B, C duty cycles
- DC-bus voltage—measured DC-bus voltage
- Filtered DC-bus voltage—filtered value of the previous item
- D current saturation flag—saturation flag for the D current PI controller
- Q current saturation flag—saturation flag for the Q current PI controller
- Open loop control—determines if the current control is open; otherwise closed
- Maximum DC-bus voltage use—this determines if the current PI controllers' outputs are limited according to the DC-bus voltage

- Zero-cancellation use—determines if the zero-cancellation is active; turned off in this application

This routine calculates the field-oriented control. The inputs to this routine include the 3-phase current, DC-bus voltage, the electrical position, the required D and Q currents, and the logical switches (open-loop control, maximum DC-bus voltage use, zero-cancellation). The output of this routine is the 3-phase duty cycle, SVM sector, and the saturation flags of PI controllers. The PI controllers and filters have structures which must be initialized prior to this routine use.

This routine is called in the fast loop state machine and its process flow chart can be seen in Figure 42. The function uses the algorithms from Freescale’s Embedded Software Libraries (FSLESL).

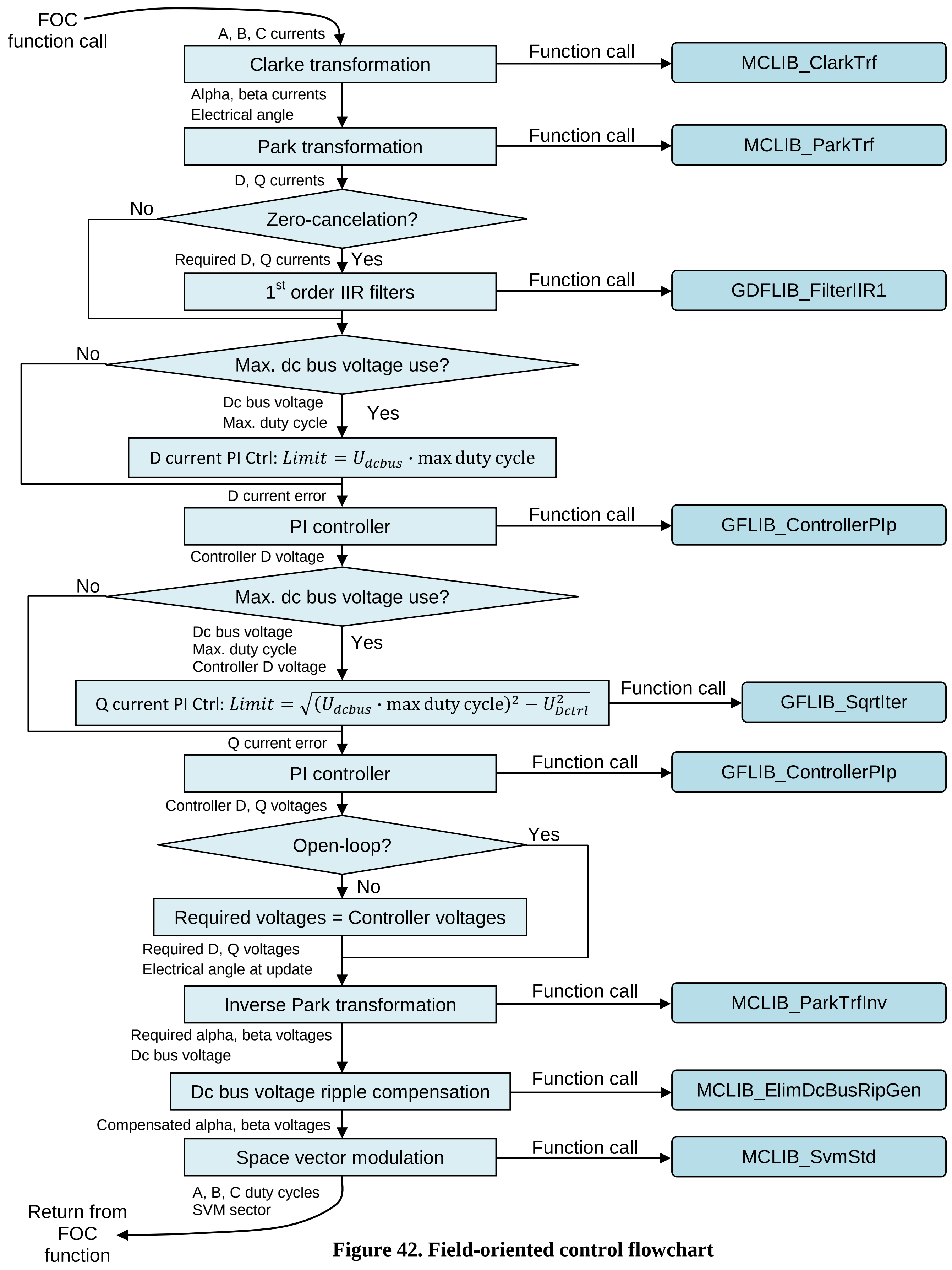


Figure 42. Field-oriented control flowchart

5.11.2 Position and speed estimation

This application uses the BEMF observer in the D/Q reference frame. Similarly as the FOC algorithm, the position and speed estimation has been optimized into one function which has one input/output pointer to a structure. The prototype of the function is the following:

```
void MCSTRUC_PMSMPositionObsDQ(MCSTRUC_FOC_PMSM_T *psFocPMSM,
MCSTRUC_BEMF_OBS_DQ_T *psObserver, MCSTRUC_POS_SPEED_EST_T *psPosition)
```

The function uses the FOC structure which is described in Field-oriented control. There are two additional structures referred by the input/output structure pointers. Their definitions follow:

```
typedef struct
{
ACLIB_BEMF_OBSRV_DQ_T         sBemfObsrv; /* BEMF observer in DQ */
ACLIB_TRACK_OBSRV_T           sTo;        /* Tracking observer */
} MCSTRUC_BEMF_OBS_DQ_T;

typedef struct
{
MCLIB_ANGLE_T     sAnglePosEl;       /* Electrical position sin/cos (at the
moment of PWM current reading) */
MCLIB_ANGLE_T     sAnglePosElUpdate; /* Compensated electrical position
sin/cos (at the moment of PWM update) */
GDFLIB_FILTER_IIR1_T    sSpeedEstFilter;  /* Estimated speed filter */
MCSTRUC_EST_STARTUP_T   sStartUp;   /* Start-up structure */
Frac16      f16PositionEl;          /* Fractional electrical position */
Frac16      f16SpeedEstimated;      /* Speed by BEMF and ATO */
Frac16      f16SpeedEstimatedFilt;  /* Speed by BEMF and ATO filtered */
bool        bObserver;              /* Observer turn on/off */
bool        bStartUp;               /* Start-up mode */
bool        bOpenLoop;              /* Position estimation loop is open */
bool        bStartUpFail;           /* Start-up fail flag */
} MCSTRUC_POS_SPEED_EST_T;
```

The former structure contains the necessary structures to calculate the BEMF observer in the D/Q frame and the tracking observer. The latter structure holds the speed and position variables and structures. Their description is the following:

- Angle when current measured—electrical rotor angle (sine, cosine) at the moment where the ADC measured the currents
- Angle when PWM updated—electrical rotor angle (sine, cosine) at the moment where the PWM is updated
- Estimated speed first order IIR filter—serves to filter the estimated speed
- Startup structure—this contains the parameters to control the open-loop start-up; it will be described in Motor open-loop startup.
- Electrical position—electrical position of the rotor angle
- Estimated speed—the estimated speed output from the tracking observer
- Filtered estimated speed—filtered estimated speed; generally the high-frequency noise is removed from the estimated speed
- Observer switch—this habilitates the use of the observer output

- Startup flag—identifies if the system is in the open-loop startup
- Startup fail flag—identifies that the startup has not been successful

This routine calculates the BEMF observer in the D/Q frame and the tracking observer. The input to the function is: the 3-phase current, required D/Q voltages, and the speed from the previous step. These parameters are necessary for the calculation. Then there are the conditional switches and flags that manage the behavior of the function, that is, to determine working at the open-loop startup and/or at the normal running. The output of this routine is the electrical position, the sine/cosine angle of the estimated position, and the estimated speed. The observers and filters have structures which must be initialized prior to this routine use.

This routine is called in the fast loop state machine prior the FOC routine. Its process flow chart can be seen in Figure 43. The function uses the algorithms from Freescale's Embedded Software Libraries (FSLESL).

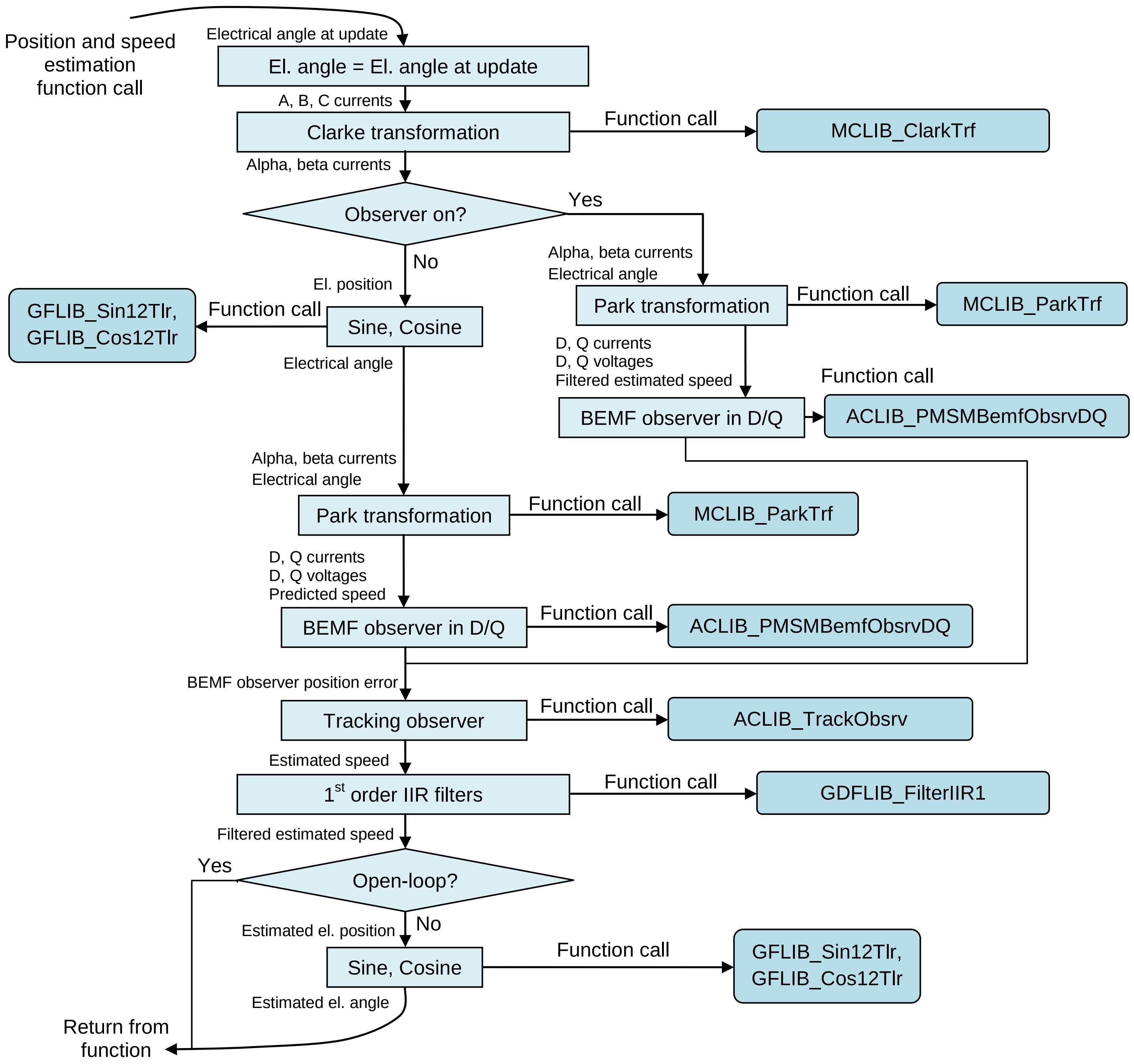


Figure 43. Position and speed estimation flowchart

5.11.3 Rotor alignment

This application uses the rotor alignment before the motor is started. It means the rotor is forced to a known position. The alignment can be any of the following.

- Rotating: this is the case of the compressor (motor 1).
- Non-rotating: the case of the fan (motor2).

Similarly as the previous algorithms, the alignment has been optimized into one function which has one input/output pointer to a structure. The prototype of the function is the following:

```
void MCSTRUC_AlignmentPMSM(MCSTRUC_FOC_PMSM_T *psFocPMSM)
```

The function uses the FOC structure which is described in Field-oriented control. In this structure, there's a sub-structure that is dedicated to the alignment. Its definition follows:

```
typedef struct
{
Frac32      f32Position;            /* Position of field at alignment */
Frac32      f32U;                   /* D voltage at alignment */
Frac32      f32Speed;               /* Speed of field at alignment */
Frac32      f32UStep;               /* D voltage ramp at alignment */
Frac16      f16IMax;                /* Max D current at alignment */
Frac16      f16UMax;                /* Max D voltage at alignment */
UWord16     uw16TimeAlignment;      /* Alignment time duration */
} MCSTRUC_ALIGNMENT_T;
```

The structure contains the necessary variables to perform the rotor alignment. As the alignment routine is called in the fast loop frequency, some variables are extended into 32-bit format. Then the upper 16 bits represent the value before the decimal point. The lower 16 bits represent the number after the decimals. The structure description follows:

- Electrical position—holds the electrical position of the field
- Voltage—the required voltage applied in the D axis
- Electrical speed—the speed of the field at the alignment; keep it zero if the rotation is not desirable
- Voltage step—the voltage ramp increment used to ramp the required D voltage
- Maximum current—limit of the measured D current at the alignment
- Maximum voltage—limit of the applied D voltage at the alignment
- Duration—defines the duration of the alignment

The routine rotates the position with the defined alignment speed. The D voltage is increased with the voltage step until the maximum voltage or until the measured current reaches its limit value. Then the D voltage is maintained constant. The electrical position and voltage must be pre-initialized prior to this algorithm use. At the end, the user must take care of setting the D voltage to zero and continue the startup from the last known electrical position from this algorithm. The alignment algorithm flow chart is displayed by Figure 44.

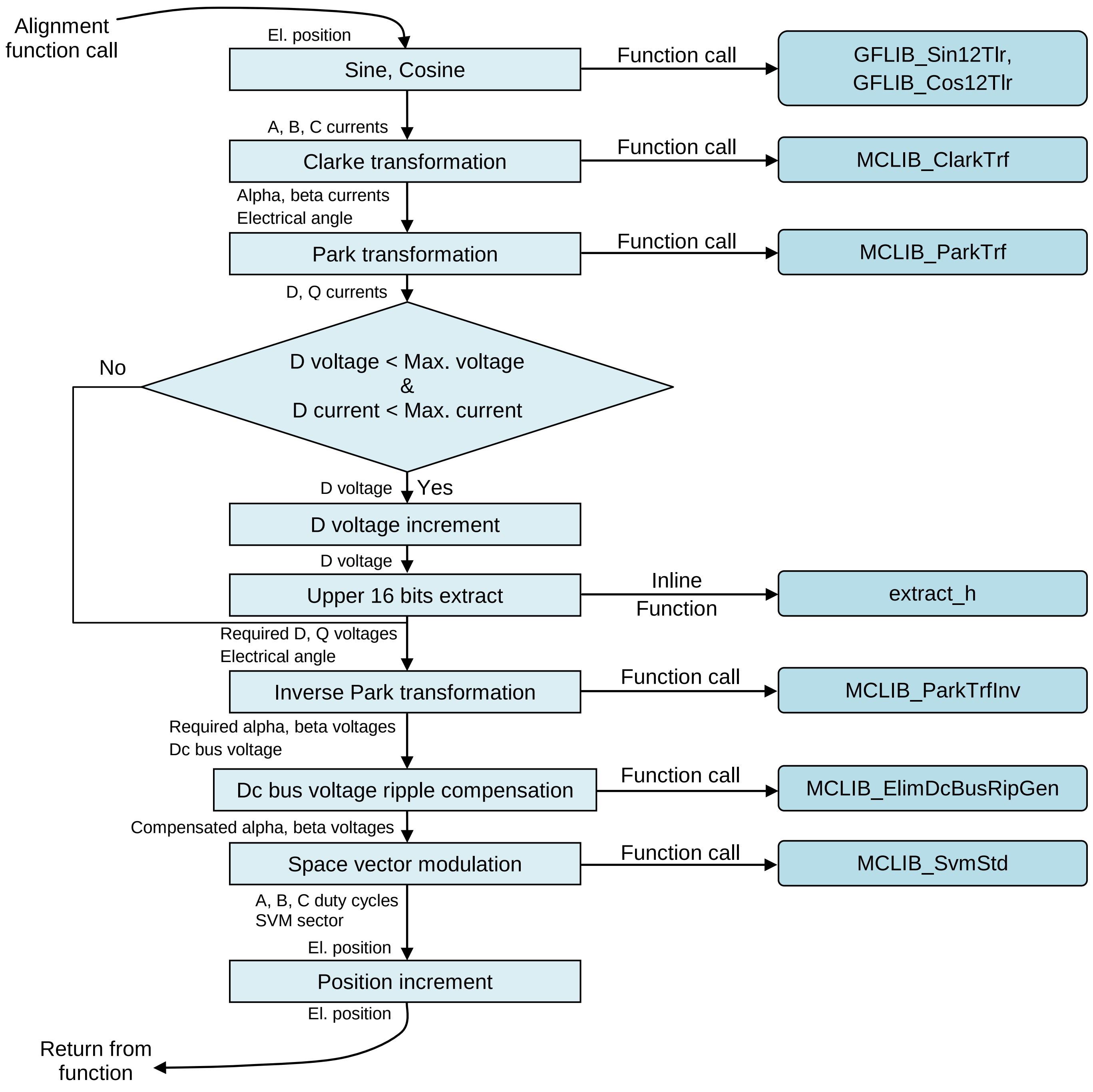


Figure 44. Rotor alignment flowchart

5.11.4 Motor open-loop startup

As the BEMF observer does not give reliable feedback at very low speeds, the motor needs to be started to certain speed in the open-loop mode. The way of start-up expects the similar startup conditions for each startup. The method consists of generated rotating field with the Q current profile that will spin the rotor according to the generated speed.

Similar to the previous algorithms, the open-loop startup has been optimized into one function which has one input/output pointer to a structure. The prototype of the function is the following:

```
void MCSTRUC_PMSMOpenLoopStartUp(MCSTRUC_FOC_PMSM_T *psFocPMSM,
MCSTRUC_POS_SPEED_EST_T *psPosition, MCSTRUC_SPEED_T *psSpeed)
```

The function uses the FOC and position/speed estimation structures which are described in the Field-oriented control and Position and speed estimation. There is an additional structure referred by the input/output structure pointers. Its definition follows:

```
typedef struct
{
GDFLIB_FILTER_IIR1_T sSpeedFilter;        /* Speed filter */
GFLIB_CONTROLLER_PI_P_PARAMS_T  sSpeedPiParams; /* Speed PI controller
parameters */
GFLIB_DYNRAMP16_T sSpeedRampParams; /* Speed ramp parameters */
Frac16 f16Speed;        /* Speed */
Frac16 f16SpeedFilt;    /* Speed filtered */
Frac16 f16SpeedError;   /* Speed error */
Frac16 f16SpeedRamp;    /* Required speed (ramp output) */
Frac16 f16SpeedReq;     /* Required speed (ramp input) */
Frac16 f16SpeedCmd;     /* Speed command (entered by user or master layer) */
Int16  i16SpeedPiSatFlag;     /* Speed PI controeller saturation flag */
bool   bOpenLoop;             /* Speed control loop is open */
} MCSTRUC_SPEED_T;
```

The structure contains the necessary variables to perform speed control loop. It is also used in the open-loop start-up because when the system is switched from the speed open-loop mode to the speed closed-loop mode, certain variables must be initialized to avoid speed dropouts. The structure description is as follows:

- Speed first order IIR filter—serves to filter the speed; not used for motor 2
- Speed PI controller structure—serves to control the speed
- Speed ramp structure—serves to generate the speed ramp
- Speed—speed of the motor
- Filtered speed—filtered speed of the motor; not used for motor 2
- Speed error—error between the required and measured speed
- Ramped speed—speed ramp algorithm output
- Required speed—speed input to the ramp algorithm
- Speed command—speed requested by the user or higher layer of the application
- Speed PI controller saturation flag—determines that the PI controller is at saturation
- Open-loop flag—the speed control loop is open.

Another structure that must be described is within the position/speed estimation structure. This structure serves for the open-loop startup:

```
typedef struct
{
ACLIB_INTEGRATOR_T sAccIntegrator;  /* Acceleration integrator structure */
ACLIB_INTEGRATOR_T sSpeedIntegrator; /* Speed integrator structure */
Frac16 f16IStartUp;           /* Current, ramped at start-up */
Frac16 f16IStartUpRamp;       /* Current ramp at start-up */
Frac16 f16IStartUpMax;         /* Max. current at start-up */
Frac16 f16IStartUpPullOut;     /* Pull-out current */
Frac16 f16IStartUpSpin;        /* Start-up current value for spinning */
Frac16 f16IStartUpSpinInit;    /* First attempt's current value for spinning 
for f16IStartUpSpin */
Frac16 f16IStartUpRamp1; /* Current ramp to pull-out current */
Frac16 f16IStartUpRamp2; /* Current ramp from the max current spike on */
Frac16 f16IStartUpBoostUp; /* f16StartUpCurrent incrementation if start-up 
fails */
UWord16 uw16CounterIStartUpMax; /* Counter of the max current spike duration 
*/
UWord16 uw16TimeIStartUpMax;   /* Max current spike duration */

Frac16 f16PositionPredicted;   /* Position generated in open-loop */
Frac16 f16SpeedPredicted;      /* Speed generated open-loop */
Frac16 f16SpeedStartUpMax;     /* Max speed reached at start-up */
Frac16 f16PositionDifferenceMax; /* Max position difference of the generated 
and estimated value */
Frac16 f16Acceleration;        /* Speed acceleration at start-up */
Frac16 f16AccelerationInit;    /* Speed acceleration at start-up, 1st attempts 
init value */
Frac16 f16AccelerationBoostDown; /* Speed acceleration decrementation at 
start-up fail */
Frac16 f16SpeedEstimWrong; /* Start-up failed speed estimation threshold */
Frac16 f16SpeedObserverOn; /* Observer-on speed */

Frac16 f16SpeedCatchUp; /* Catch-up speed */
Frac16 f16RatioCatchUpStep; /* Catch-up ratio step */
Frac16 f16RatioCatchUpOk; /* Catch-up ratio ok, min ration to be reached */
Frac16 f16RatioCatchUp;         /* Catch-up ratio */
UWord16 uw16CounterCatchUp;     /* Catch-up counter */
UWord16 uw16CounterCatchUpMax; /* Max value of the catch-up counter */

UWord16 uw16AttemptStartUp;     /* Start-up attempt counter */
UWord16 uw16AttemptStartUpMax; /* Max permitted no. of start-up attempts */

UWord16 uw16TimeStartUpFreeWheel; /* Free-wheel duration if start-up fails */

UWord16 uw16CounterStartUpOpenLoop; /* Start-up speed open-loop running 
counter */
UWord16 uw16TimeStartUpOpenLoop; /* Start-up speed open-loop running time */

Frac16 f16UdStartUpLimit;      /* Start-up D voltage limit */
Frac16 f16UqStartUpLimit;      /* Start-up Q voltage limit */
} MCSTRUC_EST_STARTUP_T;
```

The structure contains the necessary variables to perform the open-loop startup. The startup procedure is depicted in Figure 45. The structure description follows:

- Acceleration integration structure—serves to integrate the constant acceleration resulting the speed in the correct speed scale
- Speed integration structure—serves to integrate the speed resulting the position in the correct position scale
- Startup current—Q current applied to start the motor; the red curve
- Current ramp—the ramp increment which is used to ramp the current

- Maximum current—the maximum current that is applied at the startup
- Pullout current—current value that generates such a torque to pull out the rotor from the stand still
- Spin current—current which is added to the pullout current after the maximum current application point
- Init spin current—the spin current value of the first startup attempt; it is copied to the spin current variable
- Current ramp 1—current ramp increment used from the zero point to the pullout current point
- Current ramp 2—current ramp increment used from the end of the maximum current point
- Current boost-up—coefficient to increase the spin current after the unsuccessful startup
- Maximum current time counter—counter variable to count the time of the maximum current application
- Maximum current time—duration of the maximum current application
- Predicted position—generated position to spin the rotor (integrated predicted speed)
- Predicted speed—generated speed to spin the rotor (integrated acceleration)
- Maximum speed—maximum defined speed for the generated speed at startup
- Maximum position difference—maximum difference between the estimated and predicted position to allow switching to the observer closed-loop mode
- Acceleration—acceleration used at the open-loop startup to generate the predicted speed and position
- Init acceleration—this is the acceleration value of the first startup attempt; it is copied to the acceleration variable
- Acceleration boost-down—to decrease the acceleration after the unsuccessful startup
- Wrong estimated speed—speed threshold that defines the maximum real speed after the start-up. Sometimes the system thinks it is running but it is not; the observer generates malign position and speed information.
- Catch-up speed—where to start to compare the estimated and predicted speed
- Catch-up ratio step—increment of the catch-up ratio to merge the predicted and estimated position and speed
- Catch-up ok ratio—the minimum ratio that must be reached before the system can be switched to use the estimated feedback values
- Catch-up ratio—the ratio that determines how much the generated and how much estimated position and speed information will be used as the merging output
- Catch-up counter—counter used to count the time during the catch-up merging process
- Catch-up counter's maximum—the maximum value of the catch-up counter
- Startup attempts' counter—counter of the startup attempts
- Startup attempts' counter's maximum—the maximum value of the startup attempts. If reached, the startup fail flag is generated
- Startup freewheel time—duration applied due to the rotor's inertia after a failed startup before the next startup is tried
- Startup open-loop counter —counter to count time from estimator's loop closing to the speed controller's loop closing. During this time, the system runs without speed control.

- Startup open-loop time—duration of the speed open-loop running after the observer's loop is closed
- Startup D voltage limit—D voltage limit at the startup
- Startup Q voltage limit—Q voltage limit at the startup

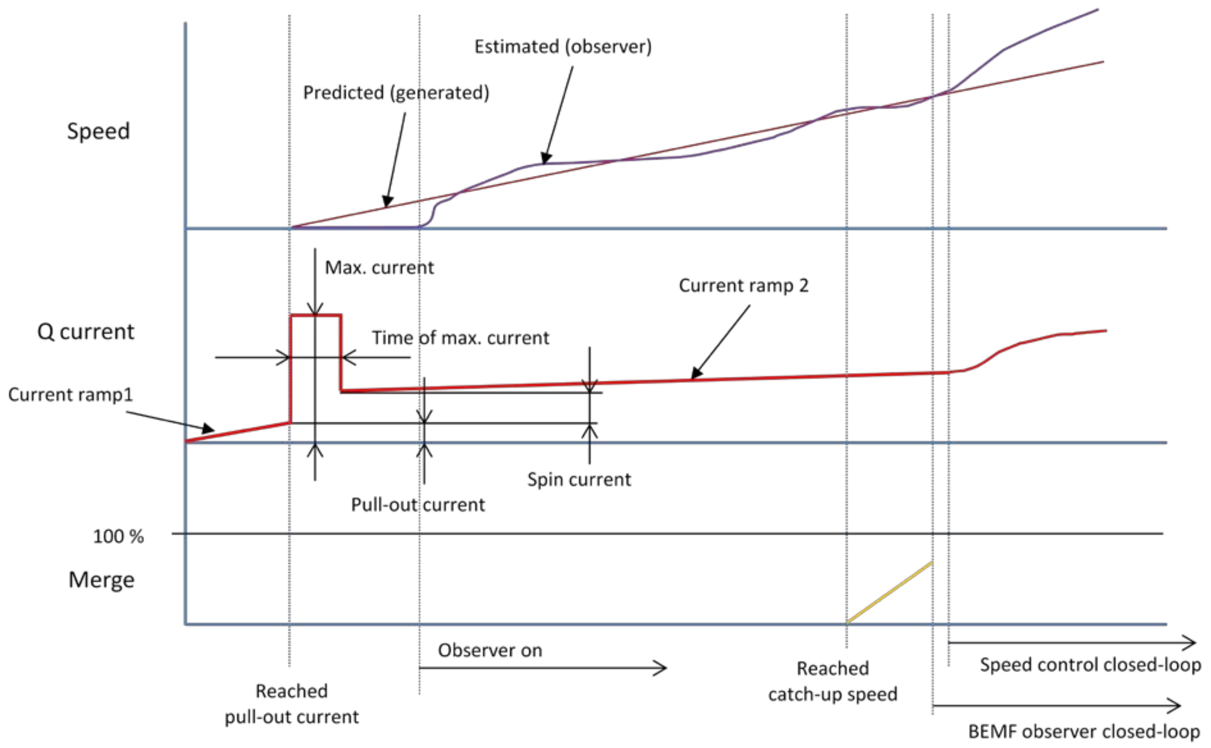


Figure 45. Motor open-loop startup

The startup procedure is quite complex process which comes from applying torque and expected the speed response accordingly. Therefore, the torque is proportional to the Q current that can be seen in Figure 45. Its generation consists of the following segments:

- From zero to the pullout current
- Maximum current peak generation
- Spin current

The acceleration and speed are integrated from the point where the pullout current is reached. At this segment, the current is ramped with the current ramp 1 increment. At this point, the maximum startup current is applied for a very short time just to pull out the rotor. After this short period of time, the value of current is the sum of pullout current and the spin current. From this point, the current is ramped with the current ramp 2 increment.

From the pullout current point, the speed is ramped with the constant acceleration. When the speed reaches the observer-on speed, the BEMF observer is turned on but its feedback is not used for the control. When the catch-up speed is reached, the catch-up ratio is incremented with the catch-up ratio step. This ratio determines how much of the estimated position and speed information is merged with the generated (predicted) ones. At the beginning, it is 0 % for the estimated and 100 % for the predicted. Their sum is always 1.

If the position difference between the predicted and estimated position is lower than the maximum permitted position difference in the range of the catch-up ok ratio to its maximum value, the system is switched into the observer closed-loop. Otherwise the startup fail flag is generated.

If the system closed the observer loop, the system runs without the speed control for a while. Then the speed loop is closed and the motor is successfully started. During the speed-open loop running, the system may think it is running but the rotor is stopped. This is recognized by estimating the speed which is much higher than the maximum startup speed. If this happens, the startup fail flag is generated.

There is a defined maximum number of startup attempts which is incremented. If the maximum number of increments is reached, the startup fail fault is generated.

5.11.5 Speed loop of motor 1

The speed loop is calculated in the Run > Spin sub-state, that is, in the *M1_StateRunSpinSlow* function. The slow loop frequency is divided to reach 1 kHz of control. The compressor uses the PID controller for the speed control, so the structure needs to be expanded along the MCSTRUC_SPEED_T structure (described in Motor open-loop startup). The structure used is given by the following code block.

```
typedef struct

{
GDFLIB_FILTER_IIR1_T    sSpeedFilterD;    /* Speed filter for the derivative
component */
GFLIB_CONTROLLER_PID_P_PARAMS_T sSpeedPidParams;      /* Speed PI controller
parameters */
Frac16 f16SpeedFiltD;         /* Speed filtered for the derivative component
*/
Frac16 f16SpeedErrorD;        /* Speed error for the derivative component */
Frac16 f16SpeedErrorDK_1;     /* Speed error for the derivative component
from previous step */
} MCSTRUC_SPEED_PID_T;
```

The structure description follows:

- Speed filter for the derivative input—this is the filter that is used to filter the speed for the derivative error input of the PID controller.
- Speed PID controller—speed PID controller structure used to control motor 1.
- Speed filtered for the derivative input—filtered speed used for the derivative input error of the PID controller
- Speed error for the derivative input—speed error used for the derivative input of the PID controller

- Speed error from the previous step—previous step speed error

First of all, the field-weakening algorithm is calculated. This will set up the required D current and will set up the Q current limit according to the following equation:

$$I_{Qlim} = \sqrt{I_{max}^2 - I_D^2}$$

Equation 36

The speed control process uses two speed variables—one is for the PI input of the PID controller and the other is for the D input of the PID controller. As the speed swing by the compressor principle is caused by the compression/expansion torque, the speed which is controlled by the PID controller is its integral. To be able to reduce the speed swing, the derivative of the speed must be controlled. So the D component of the PID controller controls this effect. The integral of the torque, the speed is 90 degrees lagged after the torque so the P component will not compensate it. And the I component will control the information with the lag of 180 degrees due to the integration. It means, it will support the torque swing instead of reducing it. Therefore only the error for the D component can have the signal with the speed swing information. The P and I components must have the signal without the swing to control the speed properly. It means filtered value.

The estimated speed is slightly filtered. As the D component gains the noise, the filter for the D component speed information must filter out the high-frequency noise. The filter for the PI components must filter out the noise and the speed swing.

Note: It must be taken into account that the filter generates transportation delay; therefore the filters cannot be strong, otherwise the control process will not be efficient.

So, the speed control uses two speeds, one for the PI components and one for the D component of the PID controller. The one for the PI components is also used for the saturation flag condition and then for the dynamic ramp calculation. Then the speed errors for the PI and D inputs are calculated and fed into the PID controller which will calculate the required Q current.

5.11.6 Speed loop of motor 2

The speed loop is calculated in the Run > Spin sub-state, that is, in the *M2_StateRunSpinSlow* function. The slow loop frequency is divided to reach 1 kHz of control. This motor uses only the PI controller as the fan's torque does not change with its position.

The speed is not filtered anymore; it takes the slightly filtered speed from the observer algorithm for the ramp and PI controller. As for the compressor, the field-weakening algorithm is calculated first which will set up the required D current and the limit for the speed PI controller. Then the ramp and PI controller are calculated which will give the required Q current.

5.12 PFC control

The application controls the PFC along two PMS motors. The PFC control uses similar principle of structural code as the motor control portion. The inputs to the routines are the particular PFC structures.

The following subsections are dedicated to the PFC control algorithm pieces.

5.12.1 Phase detection

To run the PFC, it is necessary to detect the input voltage phase and its frequency. There is an algorithm that was developed to do this job. The algorithm has been optimized into one function which has one input/output pointer to a structure. The prototype of the function is the following:

```
void PFCSTURC_PhaseDetect(PFCSTRUC_PHASE_DETECT_T *psPhaseDetect)
```

The function uses the phase detection structure. Its description follows:

```
typedef struct
{
GDFLIB_FILTER_MA32_T sFilterPeriod; /* Period filter */
Frac16 f16Freq; /* Frequency */
Frac16 f16Phase; /* Electrical phase of the voltage */
Frac16 f16Sine; /* Sine generation */
Frac16 f16UInput; /* Rectified voltage */
Word16 w16TimeDetectFallingEdge; /* Time where the start detection of the
falling edge */
Word16 w16TimeDetectRisingEdge; /* Time where the start detection of the
rising edge */
Word16 w16TimeFallingEdge; /* Falling edge time */
Word16 w16TimeRisingEdge; /* Rising edge time */
Word16 w16DelayAfterFallingEdge; /* Time to wait after the falling edge
detection before the rising edge detection is turned on */
Word16 w16DelayAfterRisingEdge; /* Time to wait after the falling edge
detection before the rising edge detection is turned on */
Word16 w16TimeZeroAngle; /* Zero angle time */
Word16 w16TimeZeroAngleNext; /* Zero angle time for next cycle*/
Word16 w16Time; /* Actual time */
Word16 w16Period; /* Period duration */
Word16 w16PeriodFilt;   /* Filtered period duration */
Frac16 f16UThresholdEdge; /* Rectified voltage edge threshold */
bool bFallingEdge; /* Looking for threshold when falling */
bool bRisingEdge; /* Looking for threshold when rising */
} PFCSTRUC_PHASE_DETECT_T;
```

The structure contains the necessary variables to perform the phase and frequency detection. The phase detection principle is depicted in Figure 46. The structure description follows:

- Period moving average filter—serves to filter the detected period of the input rectified sine voltage
- Frequency—detected rectified input voltage frequency
- Phase—phase of the rectified input voltage

- Sine—sine of the detected phase
- Input voltage—rectified input voltage
- Falling-edge detection time—time where to start detection of the threshold of the falling-edge
- Rising-edge detection time—time where to start detection of the threshold of the rising-edge
- Zero angle time—time where the zero angle was detected
- Next zero angle time—next predicted zero angle time
- Time—actual time
- Period – detected period
- Filtered period—filtered detected period
- Threshold voltage—voltage to detect on the rising or falling edge of the input voltage
- Falling edge —switch that determines the system is looking for the falling-edge threshold
- Rising edge—switch that determines the system is looking for the rising-edge threshold

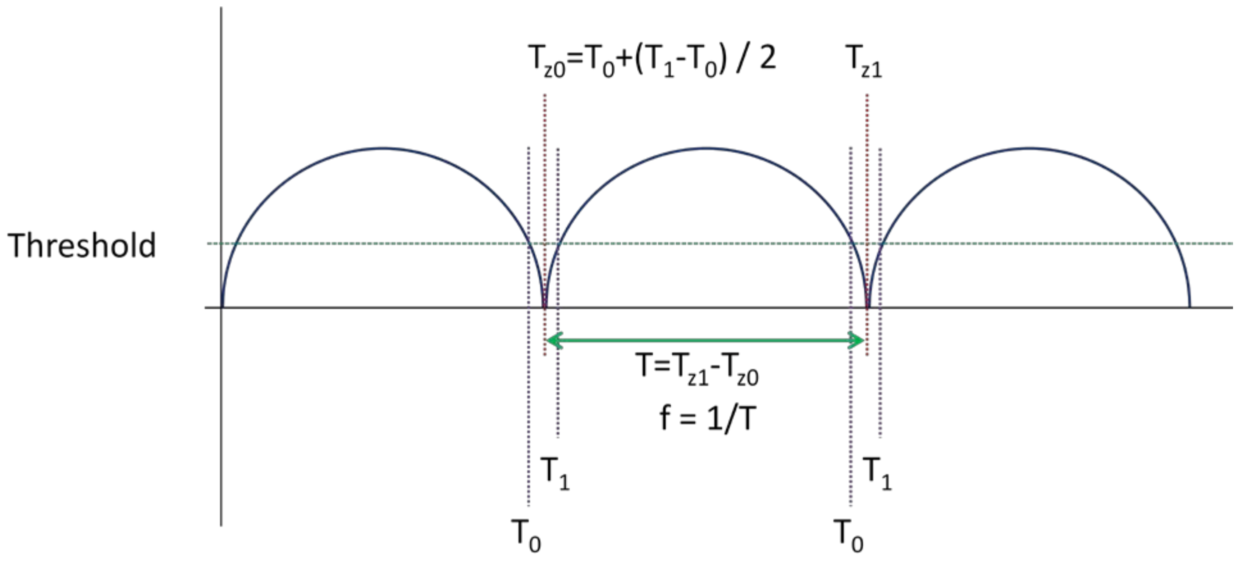


Figure 46. Input voltage phase detection

The phase detection principle is visible in Figure 46. The blue curve is the rectified input voltage that is an input to the algorithm. The input voltage value is compared to the voltage threshold. The threshold value is set to the half of the average value of the signal. The threshold value is limit to minimum value.
To find the zero angle, the system looks for the times where the input voltage passes through the threshold. First, it is set to look for the threshold on the falling edge. If it is found, its time is saved (T_0). Then the system is switched to look for the threshold on the rising-edge. If it is found, its time is saved (T_1). Therefore the zero angle (T_{z0}) is in the middle between these two times. After the threshold detection, there is always a window where the detection is disabled due to noise on the input voltage.
The period (T) is calculated from the difference between the two zero angle times. Therefore, the frequency (f) is calculated as 1 / T. The period is filtered before the frequency calculation.

Then the frequency is used to increment the angle of that phase. This angle is always set to zero at the next zero angle time. This angle is used to calculate its sine value which is the output from this algorithm. This routine is called in the state machine Run state function that runs at 20 kHz.

5.12.2 PFC current control

The PFC current control algorithm has been optimized into one function which has one input/output pointer to a structure. The prototype of the function is the following:

```
void PFCSTURC_CurrentCtrl(PFCSTRUC_CURRENT_T *psCurrentCtrl)
```

The structure referred by the input/output structure pointer is defined as follows:

```
typedef struct
{
GDFLIB_FILTER_MA32_T sUInputFilter; /* Input voltage DC component filter */
GFLIB_CONTROLLER_PI_P_PARAMS_T sIPiParams; /* Current PI controller
parameters */
MCLIB_2_COOR_SYST_T sDutyAB;  /* Applied duty cycles AB */
Frac16 f16AnglePhaseEl;       /* Electrical phase sin */
Frac16 f16UInputDC;           /* Rectified voltage - dc component */
Frac16 f16I;                  /* Current through PFC */
Frac16 f16IRefReq;            /* Required current reference */
Frac16 f16IReq;               /* Required current sine */
Frac16 f16IError;             /* Current error */
Frac16 f16IRefReqMin;         /* Min value of the required current reference
to be applied */
Frac16 f16UController;        /* Required voltage from the controller */
Frac16 f16UComp;              /* Compensated to input voltage */
Frac16 f16UReq;               /* Required voltage */
Int16 i16IPiSatFlag;          /* Current PI controller saturation flag */
Frac16 f16DutyCycleLimit;     /* Max. allowable duty cycle in frac */
bool bUseMaxBus;   /* Calculate the max. possible current controllers' output
limits based on voltage */
bool bOpenLoop;    /* Current control loop is open */
bool bInterleaved; /* If 2 MOSFET interleaved PFC, otherwise single MOSFET */
} PFCSTRUC_CURRENT_T;
```

The structure contains the necessary variables to perform the current control of PFC. The structure description follows:

- Input voltage moving average filter—heavy filter to get the DC component from the rectified input voltage
- Current PI controller —PI controller used to control the PFC current
- PFC duty cycles—two-MOSFET duty cycle, the output
- Sine of the angle—sine of the angle in phase with the power line voltage
- Input voltage DC component —DC component of the rectified input voltage
- Current—measured PFC current
- Required current reference—amplitude of the required current
- Required current—required current shape in phase with the power-line voltage

- Current error—difference between the required and measured current
- Controller output—required voltage output from the PI controller
- Compensated required voltage—voltage compensated to the input voltage DC component
- Required voltage—required voltage on the PFC inductors, copied controller output
- PI controller saturation flag—saturation flag of the current PI controller
- Duty cycle limit—limitation of the PWM duty cycle
- Maximum DC bus voltage use—this determines if the current PI controller's output is limited according to the input voltage DC component
- Open loop control—determines if the current control is open; otherwise closed
- Interleaved option—determines if there are two MOSFETs used; otherwise one MOSFET used.

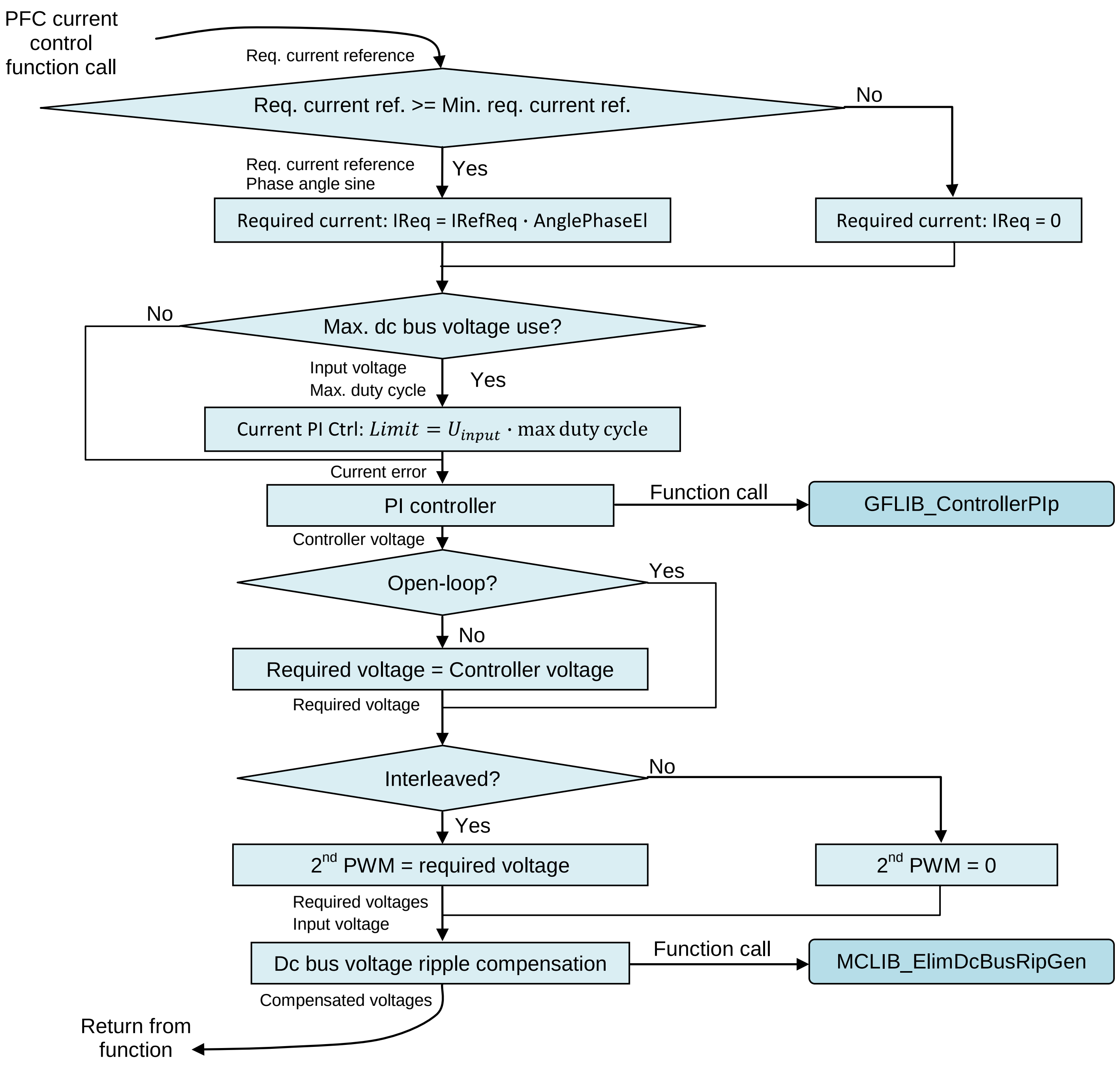


Figure 47. PFC current control flowchart

This routine calculates the PFC current control. Its inputs are the PFC current, input voltage, power line voltage angle sine, the required current, and the logical switches (open-loop control, maximum input voltage use). The output of this routine is the 2-MOSFET duty cycle and the PI controller’s saturation flag. The PI controller and the filter have structures which must be initialized prior to this routine use.

This routine is called in the state machine Run state’s Run sub-state that runs at 20 kHz and its process flowchart can be seen in Figure 47. The function uses the algorithms from Freescale’s Embedded Software Libraries (FSLESL).

5.12.3 PFC voltage control

The voltage loop is calculated in the Run > Run sub-state, that is, in the *PFC_StateRunRun* function. The slow loop frequency is divided to reach 500 Hz of control.

The slightly filtered DC-bus voltage information is used to control the PFC’s required current reference. The required voltage is ramped using the ramp algorithm and then the error between the ramped voltage and the DC-bus voltage is fed to the PI controller. The PI controller’s output is the required current reference. This current reference is copied into the current control structure only at the instant of the zero angle; it means that it will be applied for the next half sine period. The motor 1 current is fed forward, that is, it is recalculated to the PFC side and is added to the required current reference.

The structure that is used for the voltage control is the following:

```
typedef struct
{
GDFLIB_FILTER_IIR1_T sUDcBusFilter; /* Dc bus voltage filter */
GFLIB_CONTROLLER_PI_P_PARAMS_T sUDcBusPiParams; /* Dc bus voltage PI
controller parameters */
GFLIB_DYNRAMP16_T sUDcBusRampParams; /* Dc bus voltage ramp parameters */
Frac16 f16UDcBus;       /* DC bus voltage */
Frac16 f16UDcBusFilt;   /* Filtered DC bus voltage */
Frac16 f16UDcBusError;  /* DC bus voltage error */
Frac16 f16UDcBusRamp;   /* Required DC bus voltage (ramp output) */
Frac16 f16UDcBusReq;    /* Required DC bus voltage (ramp input) */
Frac16 f16UDcBusCmd;    /* DC bus voltage command (entered by user or master
layer) */
Frac16 f16UDcBusStep;   /* DC bus voltage step with respect to the input
voltage peak */
Frac16 f16IRefController; /* Required current from PI controller */
Frac16 f16IRefReq;        /* Required current reference */
Int16 i16UDcBusPiSatFlag; /* Dc bus voltage PI controller saturation flag */
bool bOpenLoop;           /* Dc bus voltage control loop is open */
bool bUDcBusAdaptive;     /* Dc bus voltage control is adaptive with respect
to the input voltage */
} PFCSTRUC_VOLTAGE_T;
```

The structure contains the necessary variables to perform the voltage control of PFC. The structure description is given below:

- DC-bus voltage filter—the first order IIR filter to filter the DC-bus voltage
- Voltage PI controller—PI controller used to control the DC-bus voltage
- DC-bus voltage—measured DC-bus voltage

- Filtered DC-bus voltage—filtered value of the measured DC-bus voltage
- DC-bus voltage error—difference between the required (ramped) voltage and the measured DC bus voltage
- Ramped required DC-bus voltage—the required DC-bus voltage that is ramped (output from the ramp algorithm)
- Required DC-bus voltage—required DC-bus voltage (input to the ramp algorithm)
- DC-bus voltage command—commanded DC-bus voltage from the higher layer
- DC-bus voltage step—determines the voltage step above the input voltage peak which is used to set up the required DC-bus voltage in the adaptive output mode
- Controller output—required current reference output from the PI controller
- Required current reference—required current reference, copied controller output
- PI controller saturation flag—saturation flag of the voltage PI controller
- Open loop control—determines if the current control is open; otherwise closed
- Adaptive output mode—if turned on, the required DC-bus voltage is set according to the input voltage peak + the DC-bus voltage step; otherwise the required DC-bus voltage is taken from the DC-bus voltage command.

5.12.4 Motor 1 current feed-forward

The PFC controls its current amount according to the measured DC-bus voltage. The current requirement depends on the DC-bus voltage change with respect to the required DC-bus voltage. If the motor 1 (that has high power demand at the start-up) is turned on, it takes energy from the DC-bus capacitor and the DC-bus voltages gets down. Generally, the PFC voltage controller will increase the required PFC current reference on how the voltage is changed. But the voltage change is the integral of the taken current; therefore the voltage PI controller action is delayed.

To change the PFC required current reference immediately when the current is needed and reduce the DC-bus voltage swing, the current of motor 1 is fed forward to the PFC required current reference.

The current of motor 1 must be recalculated to the PFC conditions. The power equation is taken in consideration in this case. Simply said, if the voltage on the DC-bus capacitor is constant, the power consumption of motor 2 is very low in comparison to that of motor 1. Thus, the power delivered from the PFC is equal to the power consumed by motor 1.

$$P_{PFC} = P_{M1}$$

Equation 37

The PFC power is given by the product of the effective (RMS) values of the PFC current and the input voltage:

$$P_{PFC} = U_{INef} \cdot I_{PFCef}$$

Equation 38

The motor 1 power consumption is given by the voltage and current on the motor. In this case, the alpha/beta frame is used:

$$P_{M1} = \frac{3}{2}\left(U_{M1\alpha} \cdot I_{M1\alpha} + U_{M1\beta} \cdot I_{M1\beta}\right)$$

Equation 39

The currents and voltages have certain scales. The voltage scale is the same for the PFC and motor 1's scale is the DC-bus voltage scale divided by the square root of 3 due to used SVM. The current scales are different. Putting them into the equations and comparing them, the following formula can be reached:

$$U_{INef} \cdot u_{sc} \cdot I_{PFCef} \cdot i_{PFCsc} = \frac{3}{2}\left(U_{M1\alpha} \cdot \frac{u_{sc}}{\sqrt{3}} \cdot I_{M1\alpha} \cdot i_{M1sc} + U_{M1\beta} \cdot \frac{u_{sc}}{\sqrt{3}} \cdot I_{M1\beta} \cdot i_{M1sc}\right)$$

Equation 40

This equation can be simplified:

$$U_{INef} \cdot I_{PFCef} \cdot i_{PFCsc} = \frac{3}{2}\left(U_{M1\alpha} \cdot I_{M1\alpha} + U_{M1\beta} \cdot I_{M1\beta}\right) \cdot \frac{i_{M1sc}}{\sqrt{3}}$$

Equation 41

This application uses the average value of the input voltage and the PFC current's amplitude, therefore the relation is the following:

$$U_{INef} = \frac{\pi}{2\sqrt{2}} \cdot U_{INavg}$$

Equation 42

$$I_{PFCef} = \frac{1}{\sqrt{2}} \cdot I_{PFCmax}$$

Equation 43

Using these two equations, Equation 41 can be written in this form:

$$\frac{\pi}{2\sqrt{2}} \cdot U_{INavg} \cdot \frac{1}{\sqrt{2}} \cdot I_{PFCmax} \cdot i_{PFCsc} = \frac{3}{2}\left(U_{M1\alpha} \cdot I_{M1\alpha} + U_{M1\beta} \cdot I_{M1\beta}\right) \cdot \frac{i_{M1sc}}{\sqrt{3}}$$

Equation 44

Therefore to express the PFC current's amplitude, the equation is the following:

$$I_{PFCmax} = \frac{3}{2} \cdot \frac{\left(U_{M1\alpha} \cdot I_{M1\alpha} + U_{M1\beta} \cdot I_{M1\beta}\right)}{U_{INavg}} \cdot \frac{4}{\pi\sqrt{3}} \cdot \frac{i_{M1sc}}{i_{PFCsc}}$$

Equation 45

5.12.5 PFC input voltage compensation

From Figure 48, it can be observed that the PFC current sensing shunt resistor moves the ground of the V_{in} signal. This means that the input voltage scale depends on the PFC current that flows through the shunt resistor. Therefore, it is necessary to compensate the measured voltage by the measured PFC current value.

The input voltage measurement circuit can be seen in Figure 48. The equation for the voltage on the divider that is measured by the ADC can be stated as follows:

$$u = U\frac{R_1 + R_2}{R_2} - i \cdot R_3$$

Equation 46

$$U = \frac{R_1 + R_2}{R_2} \cdot u + i \cdot R_3 \cdot \frac{R_1 + R_2}{R_2}$$

Equation 47

Using the scale quantities, the equation will be rewritten in this way:

$$U_{scaled} = u_{scaled} + i_{scaled} \cdot R_3 \cdot \frac{R_1 + R_2}{R_2} \cdot \frac{U_{max}}{I_{max}}$$

Equation 48

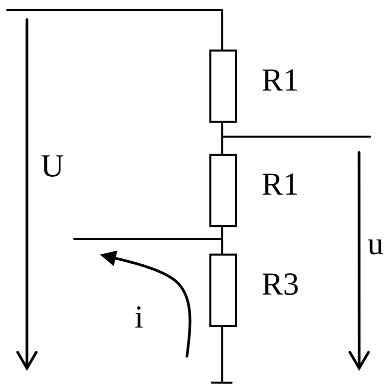


Figure 48. Input voltage measurement

5.13 Interface function

The interface functions are used for communication between the state machines. These functions are called to control and monitor the motors and PFC.

5.13.1 Switch control functions

These functions control the switch of the motors, PFC, and the application. The parameter is the boolean value determining the state of the switch; it means ON (true) or OFF (false).

```
void M1_SetAppSwitch(bool bValue)
void M2_SetAppSwitch(bool bValue)
void PFC_SetAppSwitch(bool bValue)
void APP_SetAppSwitch(bool bValue)
```

The first three functions are controlled by the application state machine while the last one is called from the user or higher layer. The last one is the entry point.

To read the status of the switch, the following functions are used. The state of the switch is returned as the boolean value.

```
bool M1_GetAppSwitch(void)
bool M2_GetAppSwitch(void)
bool PFC_GetAppSwitch(void)
bool APP_GetAppSwitch(void)
```

5.13.2 Command functions

These functions command the speed of the motors and voltage of the PFC output. The parameter is the Frac16.

```
void M1_SetSpeed(Frac16 f16SpeedCmd)
void M2_SetSpeed(Frac16 f16SpeedCmd)
void PFC_SetVoltage(Frac16 f16VoltageCmd)
void APP_SetSpeed(Frac16 f16SpeedCmd)
```

The first two functions set the speed of the particular motors. The third function sets the voltage command of the PCF. They are controlled from the application state machine. The last one controls the speed of the compressor and it is called from the user or higher layer. It is an entry point to set the speed for the compressor.

The inverse functions to them are used to monitor the speed or voltage. Their return is the Frac16 value.

```
Frac16 M1_GetSpeed(void)
Frac16 M2_GetSpeed(void)
Frac16 PFC_GetVoltage(void)
```

5.13.3 State monitor functions

These functions are used to identify if the motors or PFC are running. They return the boolean value answering the question.

```
bool M1_IsRunning(void)
bool M2_IsRunning(void)
bool PFC_IsRunning(void)
bool APP_IsRunning(void)
```

The following function is to determine that the DC bus voltage is already stabilized at the desired value.

```
bool PFC_IsVoltageGood(void)
```

5.14 Application parameters

The application parameters to control the motors, PFC, and the application are written as *#defines*. The following list represents the parameters:

5.15 Motor 1 parameters

```
#define M1_FAST_CONTROL_LOOP_FREQ   10000.0 /* Current loop frequency [Hz] */
#define M1_SLOW_CONTROL_LOOP_FREQ   1000.0 /* Speed loop frequency [Hz] */
#define M1_POLE_PAIRS   2 /* Number of motor pole pairs */

#define M1_V_DCB_SCALE  472.2 /* MAX measurable DCB votlage [V]*/
#define M1_V_FOC_SCALE  272.6 /* V_DCB / SQRT(3) [V] */
#define M1_I_SCALE      2.50 /* MAX measurable current [A] */
#define M1_SPEED_SCALE  5000.0 /* MAX measurable speed [RPM] */

#define M1_I_MAX        2.4   /* Max current [A] */
#define M1_SPEED_MIN    700.0 /* MIN applicable speed [RPM] */
#define M1_SPEED_MAX    4800.0 /* Max application speed [RPM] */
#define M1_SPEED_FW_ON  2500.0 /* Speed when the fieldweakening is allowed
[RPM] */
#define M1_OVERVOLT_LIMIT 410.0 /* Over-voltage threshold [V] */
#define M1_UNDERVOLT_LIMIT 220.0 /* Under-voltage threshold [V] */
#define M1_PRESSURE_RELAX_DURATION  4.0 /* Pressure relax time [min]; time
when start-up fail before next retry */

/* Alignment */
#define M1_ALIGN_CURRENT  2.3   /* Alignment current [V] */
#define M1_ALIGN_VOLT 30.0  /* Alignment voltage (in case the current is not
controlled) [V] */
#define M1_ALIGN_VOLT_MAX  50.0  /* Max. voltage for alignment [V] */
#define M1_ALIGN_VOLT_RAMP 20.0  /* Alignment voltage ramp [V/s] */
#define M1_ALIGN_SPEED  12.0  /* Alignment speed [RPM] */

/* Open-loop start-up */
```

```
#define M1_START_UP_RAMP_MAX 1500.0 /* Max speed of start-up ramp prediction
[RPM/s] */
#define M1_START_UP_ACCELERATION 2500.0 /* Start-up acceleration [RPM/s] */
#define M1_OBSERVER_ON_SPEED  80.0 /* Speed when the observer is turned on
[RPM] */
#define M1_START_UP_CATCH_UP_SPEED  600.0 /* Speed when the observer feedback
is starting to be considered [RPM] */

#define M1_START_UP_CURRENT_RAMP    6.0 /* Start-up current ramp from zero to
pull-out instant [A/s]*/
#define M1_START_UP_CURRENT_RAMP2   0.5 /* Start-up current ramp after pull-
out instant [A/s] */
#define M1_START_UP_CURRENT   0.4 /* Start-up fixed current portion after
pull-out instant [A] */
#define M1_START_UP_CURRENT_MAX 2.2 /* Start-up current max value, applied to
pull out the rotor [A] */
#define M1_START_UP_CURRENT_MAX_TIME 0.005 /* Duration of the max current
value [s] */
#define M1_START_UP_CURRENT_PULL_OUT 0.20 /* Current when to apply the max
current to pull out the rotor [A] */
#define M1_START_UP_MAX_POSITION_DIFFERENCE 30.0 /* Start-up max permitted
position difference [deg] */
#define M1_START_UP_OPEN_LOOP_RUN 0.20 /* Time [s] to run the motor
sensorless before the speed controller is turned on */
#define M1_START_UP_LIP 2250.0 /* The peak point where the motor goes to
after start-up before it starts surfing [RPM] */
#define M1_START_UP_LIP_TIME 2.5 /* The time the lip is maintained for [s] */


#define M1_START_UP_ATTEMPTS  8 /* Number of start-up attempts before it goes
to fault */
#define M1_START_UP_RESTART_DELAY   5.0 /* Time it waits [min] before it
attempts motor start-up again */

#define M1_START_UP_WRONG_SPEED_ESTIM 2500.0 /* Invalid estimated speed
threshold at start-up [RPM] */

#define M1_START_UP_CATCH_UP_RATIO_STEP 0.01 /* Catch-up step [0 to 1] */
#define M1_START_UP_CATCH_UP_RATION_OK 0.5 /* When the catch-up allow
switching to closed loop [0 to 1] */

#define M1_START_UP_ACCELERATION_DOWN 0.01 /* Acceleration reduction when
start-up fails [0 to 1] */
#define M1_START_UP_CURRENT_UP 0.05 /* Current raise when start-up fails [0
to 1] */

#define M1_DURATION_TASK_DEFAULT 0
#define M1_DURATION_TASK_ALIGN 2.0 /* Duration of alignment [s] */
#define M1_DURATION_TASK_CALIB 1.0 /* Duration of current calibration [s] */
#define M1_DURATION_TASK_INTER_RUN 2.0 /* Duration between start-ups [s] */
#define M1_DURATION_TASK_FREE_WHEEL 5.0 /* Duration of freewheel [s] */
#define M1_DURATION_TASK_FAULT_RELEASE 2.0 /* Duration after fault clear [s]
*/

#define M1_DUTY_CYCLE_LIMIT 0.93    /* Max. allowable dutycycle [0 to 1] */
```

```
/* D current PI controller */
#define M1_PI_D_P_GAIN  0.5978 /* D current controller proportional gain */
#define M1_PI_D_I_GAIN  0.7315 /* D current controller integral gain */
#define M1_PI_D_P_GAIN_SHIFT 2 /* D current controller proportional gain
shift */
#define M1_PI_D_I_GAIN_SHIFT -1 /* D current controller integral gain shift
*/
#define M1_PI_D_I_START_UP_LIMIT 50.0 /* D current controller output limit
[V] */

/* Q current PI controller */
#define M1_PI_Q_P_GAIN  0.6370 /* Q current controller proportional gain */
#define M1_PI_Q_I_GAIN  0.7780 /* Q current controller integral gain */
#define M1_PI_Q_P_GAIN_SHIFT 2 /* Q current controller proportional gain
shift */
#define M1_PI_Q_I_GAIN_SHIFT -1 /* Q current controller integral gain shift
*/
#define M1_PI_Q_I_START_UP_LIMIT 90.0 /* Q current controller output limit
[V] */


/* Speed current PI controller */
#define M1_PID_SPEED_P_GAIN   0.7    /* Speed controller proportional gain */
#define M1_PID_SPEED_I_GAIN   0.02   /* Speed controller integral gain */
#define M1_PID_SPEED_D_GAIN   0.8    /* Speed controller integral gain */
#define M1_PID_SPEED_P_GAIN_SHIFT 2  /* Speed controller proportional gain
shift */
#define M1_PID_SPEED_I_GAIN_SHIFT 0 /* Speed controller integral gain shift
*/
#define M1_PID_SPEED_D_GAIN_SHIFT 6 /* Speed controller integral gain shift
*/
#define M1_PID_SPEED_OUTPUT_LIMIT 2.3 /* Speed controller output limit [A]*/


/* Field-weakening PI controller */
#define M1_PI_FW_P_GAIN 0.05 /* Field-weakening proportional gain */
#define M1_PI_FW_I_GAIN 0.03 /* Field-weakening integral gain */
#define M1_PI_FW_P_GAIN_SHIFT  0 /* Field-weakening proportional gain shift
*/
#define M1_PI_FW_I_GAIN_SHIFT 0 /* Field-weakening integral gain shift */
#define M1_PI_FW_OUTPUT_LIMIT 1.5 /* Output limit [A] */

/* Speed ramp */
#define M1_SPEED_RAMP 2500.0  /* Speed acceleration [RPM/s] */
#define M1_SPEED_RAMP_UNSAT 100.0   /* Speed decceleration [RPM/s] in case of
saturation to unsaturate the system */

#define M1_SPEED_RAPM_DOWN_SPEED 1600.0 /* speed threshold; it speed is lower
than this threshold, the RAMP_DOWN will be applied for down-ramp only
[RPM/s]; otherwise the SPEED_RAMP is used */
#define M1_SPEED_RAMP_DOWN 500.0 /* Speed acceleration [RPM/s] */

/* BEMF observer */
#define M1_OBSRV_I_SCALED        32306
#define M1_OBSRV_U_SCALED        5579
```

```
#define M1_OBSRV_E_SCALED         3794
#define M1_OBSRV_WI_SCALED        3597
#define M1_OBSRV_P_GAIN           27805
#define M1_OBSRV_P_GAIN_SHIFT     1
#define M1_OBSRV_I_GAIN           17621
#define M1_OBSRV_I_GAIN_SHIFT     -2

/* Tracking observer */
#define M1_TO_P_GAIN              26739
#define M1_TO_P_GAIN_SHIFT        -2
#define M1_TO_I_GAIN              25299
#define M1_TO_I_GAIN_SHIFT        -9
#define M1_TO_TH_SCALE            17476
#define M1_TO_TH_SHIFT            -4

/* Pole zero cancellation, not used in this application */
#define M1_ZC_D_B1                0.0
#define M1_ZC_D_B2                0.0
#define M1_ZC_D_A2                0.0
#define M1_ZC_Q_B1                0.0
#define M1_ZC_Q_B2                0.0
#define M1_ZC_Q_A2                0.0

/* Filters */
#define M1_FILTER_WINDOW_ADC_OFFSET 3 /* current calibration filter shift */

#define M1_FILTER_UDCBUS_B1    0.0305      /* DC bus voltage filter B1 */
#define M1_FILTER_UDCBUS_B2    0.0305      /* DC bus voltage filter B2 */
#define M1_FILTER_UDCBUS_A2    -0.9391     /* DC bus voltage filter A2 */

#define M1_FILTER_SPEED_EST_B1    0.0305   /* Estimated speed filter B1 */
#define M1_FILTER_SPEED_EST_B2    0.0305   /* Estimated speed filter B2 */
#define M1_FILTER_SPEED_EST_A2    -0.9391  /* Estimated speed filter A2 */

/* 8Hz */
#define M1_FILTER_SPEED_B1    0.0245       /* Speed filter B1 */
#define M1_FILTER_SPEED_B2    0.0245       /* Speed filter B2 */
#define M1_FILTER_SPEED_A2    -0.9510      /* Speed filter A2 */

/* 25Hz */
#define M1_FILTER_SPEED_D_B1 0.073         /* Speed filter for D comp B1 */
#define M1_FILTER_SPEED_D_B2 0.073         /* Speed filter for D comp B2 */
#define M1_FILTER_SPEED_D_A2 -0.8541       /* Speed filter for D comp A2 */

#define M1_FILTER_FW_B1  0.073      /* Fieldweakening filter B1 */
#define M1_FILTER_FW_B2  0.073      /* Fieldweakening filter B2 */
#define M1_FILTER_FW_A2  -0.8541    /* Fieldweakening filter A2 */
```

5.16 Motor 2 parameters

```
#define M2_FAST_CONTROL_LOOP_FREQ   10000.0 /* Current loop frequency [Hz] */
#define M2_SLOW_CONTROL_LOOP_FREQ   1000.0 /* Speed loop frequency [Hz] */
```

```
#define M2_POLE_PAIRS               4     /* Number of motor pole pairs */

#define M2_V_DCB_SCALE  472.2 /* MAX measurable DCB votlage [V] */
#define M2_V_FOC_SCALE  272.6 /* V_DCB / SQRT(3) [V] */
#define M2_I_SCALE      0.29 /* MAX measurable current [A] */
#define M2_SPEED_SCALE  3000.0 /* MAX measurable speed [RPM] */

#define M2_I_MAX        0.29  /* Max current [A] */
#define M2_SPEED_MIN    500.0 /* MIN applicable speed [RPM] */
#define M2_SPEED_MAX    2200.0 /* Max application speed [RPM] */
#define M2_SPEED_FW_ON  1700.0 /* Speed when the fieldweakening is allowed
[RPM] */
#define M2_OVERVOLT_LIMIT 410.0 /* Over-voltage threshold [V] */
#define M2_UNDERVOLT_LIMIT 180.0 /* Under-voltage threshold [V] */

/* Alignment */
#define M2_ALIGN_CURRENT   0.25 /* Alignment current [A] */
#define M2_ALIGN_VOLT      30.0 /* Alignment voltage (in case the current is
not controlled) [V]*/
#define M2_ALIGN_VOLT_MAX  25.0 /* Max. voltage for alignment [V] */
#define M2_ALIGN_VOLT_RAMP 20.0 /* Alignment voltage ramp [V/s] */
#define M2_ALIGN_SPEED     0.0  /* Alignment speed [RPM] */

/* Open-loop start-up */
#define M2_START_UP_RAMP_MAX 1200.0 /* Max speed of start-up ramp prediction
[RPM] */
#define M2_START_UP_ACCELERATION 1000.0 /* Start-up acceleration [RPM/s] */
#define M2_OBSERVER_ON_SPEED 80.0 /* Speed when the observer is turned on
[RPM] */
#define M2_START_UP_CATCH_UP_SPEED 200.0 /* Speed when the observer feedback
is starting to be considered [RPM] */
#define M2_START_UP_CURRENT_RAMP 2.0 /* Start-up current ramp from zero to
pull-out instant [A/s] */
#define M2_START_UP_CURRENT_RAMP2 0.05 /* Start-up current ramp after pull-
out instant [A/s] */
#define M2_START_UP_CURRENT 0.013 /* Start-up fixed current portion after
pull-out instant [A] */
#define M2_START_UP_CURRENT_MAX 0.18 /* Start-up current max value, applied
to pull out the rotor [A] */
#define M2_START_UP_CURRENT_MAX_TIME 0.003 /* Duration of the max current
value [s] */
#define M2_START_UP_CURRENT_PULL_OUT 0.008 /* Current when to apply the max
current to pull out the rotor [A] */
#define M2_START_UP_MAX_POSITION_DIFFERENCE 30.0 /* Start-up max permitted
position difference [deg] */
#define M2_START_UP_OPEN_LOOP_RUN 0.150 /* Time [s] to run the motor
sensorless before the speed controller is turned on */



#define M2_START_UP_ATTEMPTS  8 /* Number of start-up attempts before it goes
to fault */
#define M2_START_UP_RESTART_DELAY 8.0 /* Time it waits [min] before it
attempts motor start-up again */
```

```
#define M2_START_UP_WRONG_SPEED_ESTIM 1200.0 /* Invalid estimated speed
threshold at start-up [RPM] */

#define M2_START_UP_CATCH_UP_RATIO_STEP 0.01 /* Catch-up step [0 to 1]*/
#define M2_START_UP_CATCH_UP_RATION_OK 0.5 /* When the catch-up allow
switching to closed loop [0 to 1] */


#define M2_START_UP_ACCELERATION_DOWN 0.01 /* Acceleration reduction when
start-up fails [0 to 1] */
#define M2_START_UP_CURRENT_UP 0.05 /* Current raise when start-up fails [0
to 1] */

#define M2_DURATION_TASK_DEFAULT 0
#define M2_DURATION_TASK_ALIGN 2.5 /* Duration of alignment [s] */
#define M2_DURATION_TASK_CALIB 1.0 /* Duration of current calibration [s] */
#define M2_DURATION_TASK_INTER_RUN 2.0 /* Duration between start-ups [s] */
#define M2_DURATION_TASK_FREE_WHEEL 5.0 /* Duration of freewheel [s] */
#define M2_DURATION_TASK_FAULT_RELEASE 2.0 /* Duration after fault clear [s]
*/

#define M2_DUTY_CYCLE_LIMIT 0.93 /* Max. allowable dutycycle [0 to 1] */

/* D current PI controller */
#define M2_PI_D_P_GAIN  0.8 /* D current controller proportional gain */
#define M2_PI_D_I_GAIN  0.6 /* D current controller integral gain */
#define M2_PI_D_P_GAIN_SHIFT 0 /* D current controller proportional gain
shift */
#define M2_PI_D_I_GAIN_SHIFT -3 /* D current controller integral gain shift
*/
#define M2_PI_D_I_START_UP_LIMIT 30.0 /* D current controller output limit
[V] */

/* Q current PI controller */
#define M2_PI_Q_P_GAIN  0.85 /* Q current controller proportional gain */
#define M2_PI_Q_I_GAIN  0.7 /* Q current controller integral gain */
#define M2_PI_Q_P_GAIN_SHIFT 0 /* Q current controller proportional gain
shift */
#define M2_PI_Q_I_GAIN_SHIFT -2 /* Q current controller integral gain shift
*/
#define M2_PI_Q_I_START_UP_LIMIT 40.0 /* Q current controller output limit
[V] */

/* Speed current PI controller */
#define M2_PI_SPEED_P_GAIN 0.7   /* Speed controller proportional gain */
#define M2_PI_SPEED_I_GAIN 0.002 /* Speed controller integral gain */
#define M2_PI_SPEED_P_GAIN_SHIFT 2 /* Speed controller proportional gain
shift */
#define M2_PI_SPEED_I_GAIN_SHIFT 0 /* Speed controller integral gain shift */
#define M2_PI_SPEED_OUTPUT_LIMIT 0.25 /* Speed controller output limit in [A]
*/

/* Field-weakening PI controller */
#define M2_PI_FW_P_GAIN       0.05 /* Field-weakening proportional gain */
#define M2_PI_FW_I_GAIN       0.03 /* Field-weakening integral gain */
#define M2_PI_FW_P_GAIN_SHIFT 0 /* Field-weakening proportional gain shift */
#define M2_PI_FW_I_GAIN_SHIFT 0 /* Field-weakening integral gain shift */
```

```
#define M2_PI_FW_OUTPUT_LIMIT 0.20 /* Output limit [A] */

/* Speed ramp */
#define M2_SPEED_RAMP 1000    /* Speed acceleration [RPM/s] */
#define M2_SPEED_RAMP_UNSAT 500 /* Speed decceleration [RPM/s] in case of
saturation to unsaturate the system */

/* BEMF observer */
#define M2_OBSRV_I_SCALED     31964
#define M2_OBSRV_U_SCALED     16784
#define M2_OBSRV_E_SCALED     11391
#define M2_OBSRV_WI_SCALED    2311
#define M2_OBSRV_P_GAIN       17330
#define M2_OBSRV_P_GAIN_SHIFT 0
#define M2_OBSRV_I_GAIN       23231
#define M2_OBSRV_I_GAIN_SHIFT -4

/* Tracking observer */
#define M2_TO_P_GAIN          22283
#define M2_TO_P_GAIN_SHIFT    -2
#define M2_TO_I_GAIN          21083
#define M2_TO_I_GAIN_SHIFT    -9
#define M2_TO_TH_SCALE        20971
#define M2_TO_TH_SHIFT        -4

/* Pole zero cancellation, not used in this application */
#define M2_ZC_D_B1 0.0
#define M2_ZC_D_B2 0.0
#define M2_ZC_D_A2 0.0
#define M2_ZC_Q_B1 0.0
#define M2_ZC_Q_B2 0.0
#define M2_ZC_Q_A2 0.0

/* Filters */
#define M2_FILTER_WINDOW_ADC_OFFSET 3 /* current calibration filter shift */

#define M2_FILTER_UDCBUS_B1 0.0305        /* DC bus voltage filter B1 */
#define M2_FILTER_UDCBUS_B2 0.0305        /* DC bus voltage filter B2 */
#define M2_FILTER_UDCBUS_A2 -0.9391       /* DC bus voltage filter A2 */

#define M2_FILTER_SPEED_EST_B1 0.073      /* Estimated speed filter B1 */
#define M2_FILTER_SPEED_EST_B2 0.073      /* Estimated speed filter B2 */
#define M2_FILTER_SPEED_EST_A2 -0.8541    /* Estimated speed filter A2 */

#define M2_FILTER_FW_B1 0.073       /* Fieldweakening filter B1 */
#define M2_FILTER_FW_B2 0.073       /* Fieldweakening filter B2 */
#define M2_FILTER_FW_A2 -0.8541     /* Fieldweakening filter A2 */
```

5.17 PFC parameters

```
#define PFC_FAST_CONTROL_LOOP_FREQ  20000.0 /* Current loop frequency [V] */
#define PFC_SLOW_CONTROL_LOOP_FREQ  500.0 /* Speed loop frequency [V] */
#define PFC_INTERLEAVED             1 /* Keep defined if interleaved PFC */
```

```
#define PFC_V_DCB_SCALE       472.2 /* MAX measurable DCB votlage [V] */
#define PFC_V_IN_SCALE        472.2 /* MAX measurable input votlage [V] */
#define PFC_V_IN_I_PFC_COEF   14.3  /* coef used to remove current effect
from Vin */
#define PFC_I_SCALE           11.8 /* MAX measurable current [A]*/

#define PFC_FREQUENCY_SCALE
      (PFC_FAST_CONTROL_LOOP_FREQ / 2.0) /* MAX measurable frequency [Hz] */

#define PFC_VOLTAGE_MIN       280.0 /* MIN applicable voltage [V] */
#define PFC_VOLTAGE_MAX       385.0 /* Max application voltage [V] */
#define PFC_IN_OVERVOLT_LIMIT 270.0 /* Max input voltage dc component [V] */
#define PFC_IN_UNDERVOLT_LIMIT 75.0 /* Min input voltage dc component [V] */
#define PFC_OVERVOLT_LIMIT    415.0 /* Max dc bus voltage [V] */
#define PFC_UNDERVOLT_LIMIT   100.0 /* Min dc bus voltage [V] */
#define PFC_OVERFREQUENCY_LIMIT 70.0 /* Max freq on input voltage [Hz] */
#define PFC_UNDEFREQUENCY_LIMIT 40.0 /* Min freq on input voltage [Hz] */
#define PFC_VOLTAGE_GOOD_DEVIATION 0.01 /* Portion of output voltage to set
deviation when voltage is good [0 to 1] */

#define PFC_EDGE_THRESHOLD_VOLTAGE  85.0  /* Voltage to detect edges before
and after zero angle [V] */
#define PFC_OUT_STEP 40.0     /* Output voltage step added to the input
voltage peak for the dc bus voltage desired value [V] */

/* Current PI controller */
#define PFC_PI_I_P_GAIN 0.04327 /* Current controller proportional gain */
#define PFC_PI_I_I_GAIN 0.09833 /* Current controller integral gain */
#define PFC_PI_I_P_GAIN_SHIFT 0 /* Current controller proportional gain shift
*/
#define PFC_PI_I_I_GAIN_SHIFT 0 /* Current controller integral gain shift */
#define PFC_PI_I_START_UP_LIMIT 30.0 /* Current controller output limit [V]
*/

#define PFC_PI_UDCB_P_GAIN 0.68 /* Voltage controller proportional gain */
#define PFC_PI_UDCB_I_GAIN 0.35 /* Voltage controller integral gain */
#define PFC_PI_UDCB_P_GAIN_SHIFT 4 /* Voltage controller proportional gain
shift */
#define PFC_PI_UDCB_I_GAIN_SHIFT -3 /* Voltage controller integral gain shift
*/
#define PFC_PI_UDCB_I_LIMIT 7.5 /* Voltage controller output limit [A] */

/* Speed ramp */
#define PFC_SPEED_RAMP 300 /* Voltage ramp [V/s] */
#define PFC_SPEED_RAMP_UNSAT 50 /* Voltage ramp down [V/s] in case of
saturation to unsaturate the system */

/* Mult by 1.56 (average = peak * 2 / pi) */
#define PFC_UINDC_UDCB_MULT 0.78 /* Recalculation coef of input voltage DC
component to get the peak */
#define PFC_UINDC_UDCB_MULT_SHIFT 1 /* Recalculation shift of input voltage
DC component to get the peak */

#define PFC_PERIOD_FILTER_WINDOW 6 /* Period filter */
```

```
#define PFC_DELAY_AFTER_FALLING_EDGE 0.0004 /* Time to wait after falling
edge detection before looking for the rising edge [s] */
#define PFC_DELAY_AFTER_RISING_EDGE 0.0050 /* Time to wait after rising edge
detection before looking for the rising edge [s] */
#define PFC_I_REF_REQ_MIN 0.5 /* Min current to be applied on the PI
controller [A] */

#define PFC_FILTER_UDCBUS_B1 0.1367 /* DC bus voltage filter B1 */
#define PFC_FILTER_UDCBUS_B2 0.1367 /* DC bus voltage filter B2 */
#define PFC_FILTER_UDCBUS_A2 -0.7265 /* DC bus voltage filter A2 */

#define PFC_FILTER_UIN_DC_WINDOW 12 /* Input voltage DC component filter */
#define PFC_FILTER_IPFC_FEEDFORWARD_WINDOW 7 /* Pfc current Motor 1's current
feedforward */

#define PFC_DURATION_TASK_DEFAULT 0
#define PFC_DURATION_TASK_CALIB 2.0 /* Duration of frequency calibration [s]
*/
#define PFC_DURATION_TASK_FAULT_RELEASE 2.0 /* Duration after fault clear [s]
*/

#define PFC_DUTY_CYCLE_LIMIT  0.9 /* Duty cycle limit [0 to 1]*/
```

5.18 Application parameters

```
#define APP_FAST_CONTROL_LOOP_FREQ 500.0  /* Fast loop frequency [Hz]*/
#define APP_SLOW_CONTROL_LOOP_FREQ 20.0   /* Slow loop frequency [Hz] */

#define APP_V_DCB_SCALE 472.2 /* MAX measurable DCB voltage [V] */
#define APP_COMPRESSOR_SPEED_SCALE  M1_SPEED_SCALE
#define APP_FAN_SPEED_SCALE         M2_SPEED_SCALE
#define APP_TEMP_SCALE              128.0 /* Scale for temperatures [deg C]*/

#define APP_OVERVOLT_LIMIT    420.0 /* Max dc bus voltage [V] */
#define APP_UNDERVOLT_LIMIT   127.0 /* Min dc bus voltage [V] */
#define APP_OVERTEMP_LIMIT    57.0  /* Hot side max permitted temperature
[deg C] */

#define APP_COMPRESSOR_SPEED_MAX    4500.0 /* Max compressor speed [RPM] */
#define APP_COMPRESSOR_SPEED_MIN    1200.0 /* Min compressor speed [RPM] */

#define APP_FAN_SPEED_MAX     2000.0 /* Max fan speed [RPM] */
#define APP_FAN_SPEED_MIN     500.0 /* Min fan speed [RPM] */
#define APP_FAN_SPEED_ON      1000.0 /* Fan speed command where the
compressor is turned on [RPM] */
#define APP_COMPRESSOR_SPEED_FAN_ON M1_SPEED_MIN /* Min compressor speed to
allow fan's turning on [RPM] */

#define APP_TEMP_SPEED_MIN    30.0 /* Min fan speed temperature [deg C] */
#define APP_TEMP_SPEED_MAX    60.0 /* Max fan speed temperature [deg C] */

#define APP_HOT_SIDE_TEMP     35.0  /* Level (deg C) to be maintained during
refrigeration on hot side [deg C] */
```

```
#define APP_DURATION_TASK_DEFAULT         0
#define APP_DURATION_TASK_FAULT_RELEASE   2.0   /* Duration after fault clear
[s] */

#define APP_FILTER_UDCBUS_WINDOW    6 /* DC bus voltage filter shift */
#define APP_FILTER_HOT_TEMP_WINDOW  6 /* Hot side temperature filter shift */
#define APP_FILTER_COLD_TEMP_WINDOW 6 /* Cold side temperature filter shift
*/
#define APP_FILTER_AMBIENT_TEMP_WINDOW 6 /* Ambient side temperature filter
shift */
```

5.19 Communication parameters

```
#define COMMUNICATION
#define COMM_USE_TX_DMA
#define COMM_CONTROL_LOOP_FREQ      10000.0 /* Control loop frequency [Hz] */
#define COMM_BUFFER_SIZE 600   /* Phase current data buffer size [bytes]*/
#define COMM_FREQ       200.0 /* Communication values update frequency [Hz]
*/
#define COMM_RECORDER_POINTS  200 /* Phase current recorder points */

#define COMM_FILTER_IPFC_WINDOW 9   /* PFC current filter shift */
#define COMM_FILTER_IM1_WINDOW 9    /* Motor 1 current filter shift */
#define COMM_FILTER_IM2_WINDOW 9    /* Motor 2 current filter shift */
```

5.20 Microcontroller memory usage

Table 4 shows how much memory is needed to run the application on MC56F84789 DSC. A significant part of the microcontroller memory is still available for other tasks.

Table 4. Memory usage

Memory	Available MC56F84789	Used
FLASH	288 KB	24.7 KB
RAM	32 KB	2.7 KB

5.21 Peripherals usage

For the proper function of this application, the following peripherals (see Table 5) must be used. It is not allowed to use these peripherals for any other purpose.

Table 5. MC56F84789 peripheral usage

Module	Purpose

PWM A	• Motor 1 3-phase PWM • PFC PWM
PWM B	• Motor 2 3-phase PWM
ADC A and B	• Motor 1 and 2 currents • DC-bus voltage • 3 temperatures
ADC C	• PFC current • PFC input voltage
CMP C and D	• DC bus overvoltage fault • PFC overcurrent fault
Timer A3	• PFC's ADC synchronization to PWM
PDB A	• PFC's ADC synchronization to PWM
DMA channel 0 and 1	• PFC's ADC C serving
DMA channel 2	• UART Tx
UART 0	• Communication with the master control tower
Crossbar A and B	• PWM-to-ADC triggers • Pins-to-PWM faults • Modules synchronization

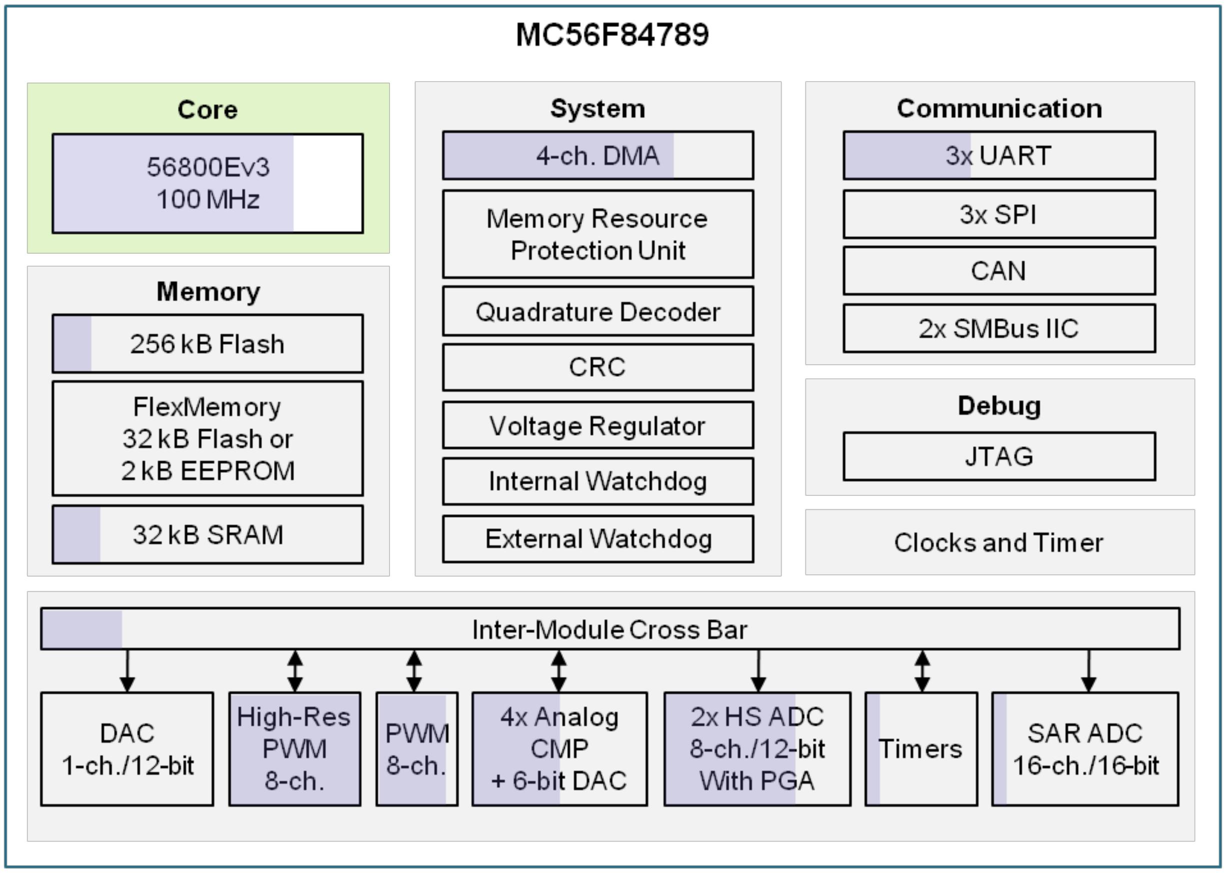


Figure 49. MC56F84789 peripherals explotation

5.22 Application timing on scope

The following scope screen shots (see Figure 50, Figure 51, Figure 52, and Figure 53) show the application timing where no motor runs, when one of the motors run and when both run.

- The first (yellow) channel shows the PFC PWM. It runs at 80 kHz.
- The second (cyan) channel is the motor 1 top channel PWM. This PWM is only visible when motor 1 runs and the PWM frequency is 10 kHz.
- The third (purple) channel is a signal from a GPIO pin which was set at the beginning of the PFC control interrupt and cleared at the end of that interrupt. One can observe the duration of the PFC routine calculation and its frequency of 20 kHz.
- The fourth (green) channel is a signal from a GPIO pin which was set when the motor 1 or 2 fast loop or both motor's slow loop calculation interrupt was entered and cleared at the end of the particular interrupt. One can observe the duration of the slow loop calculation of both motors, the fast loop calculation of motor 1 and the fast loop calculation of motor 2. If motor 1 runs, then the motor 2's fast loop is calculated in the same interrupt.

  Note: The PFC loop interrupt has higher priority therefore it interrupts the motors loop calculation. It means the observer has to subtract the time of the PFC calculation from the green signal.

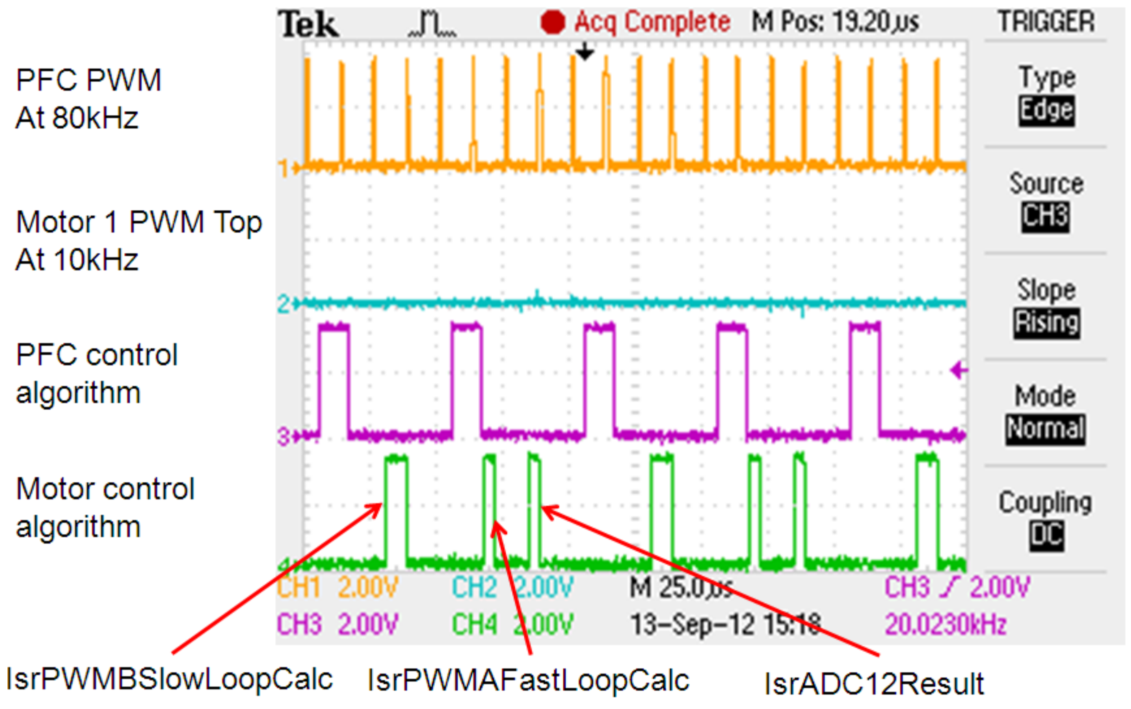


Figure 50. Application timing—PFC running, motors stopped

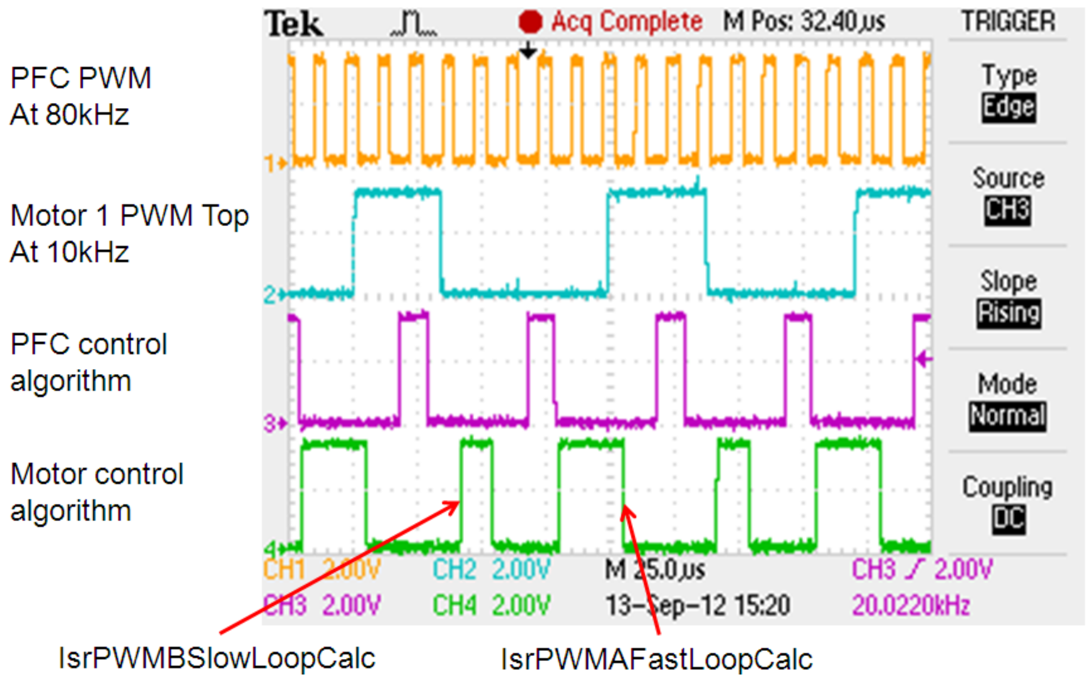


Figure 51. Application timing—PFC + motor 1 running, motor 2 stopped

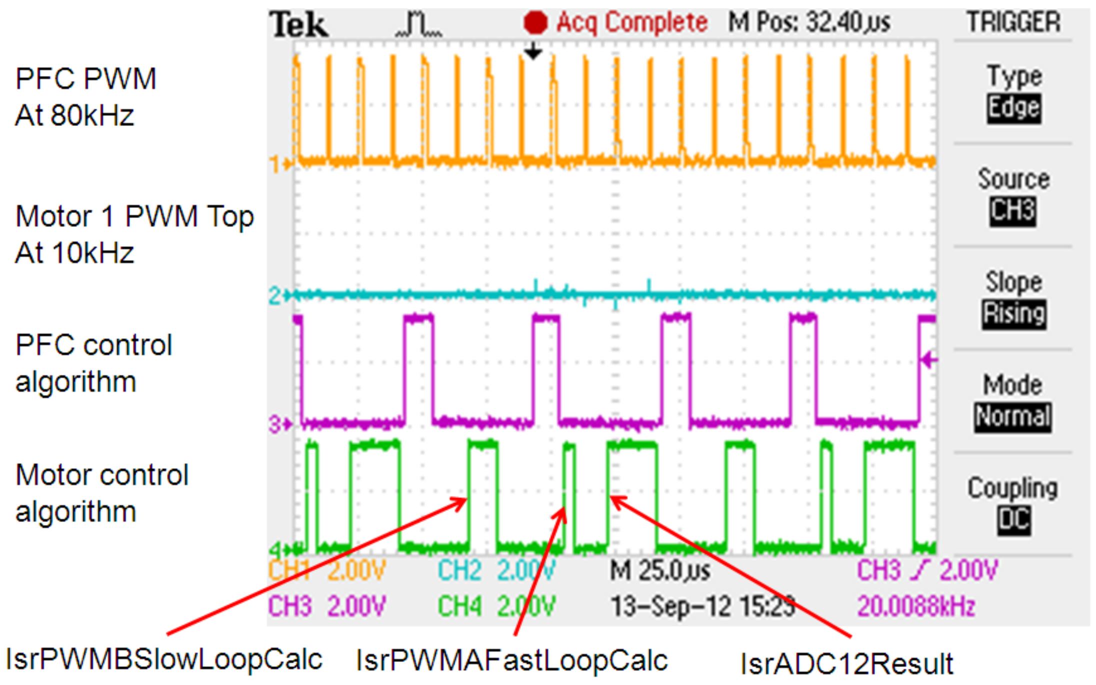


Figure 52. PFC + motor 2 running, motor 1 stopped

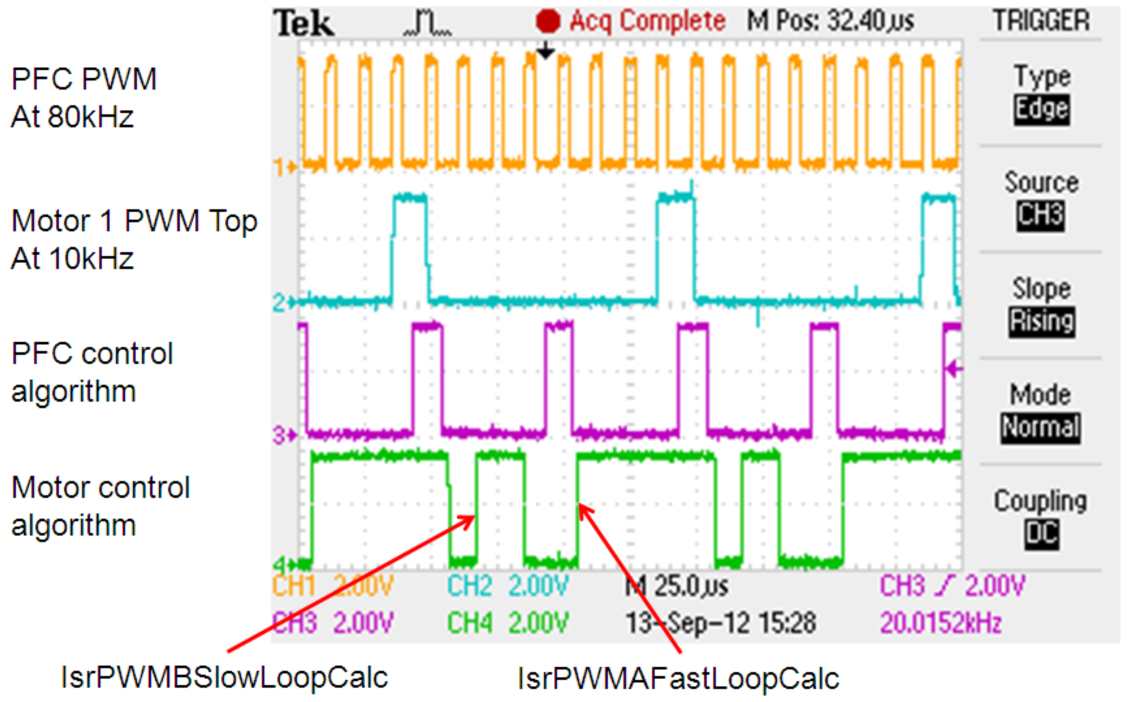


Figure 53. PFC + motor 1 + motor 2 running

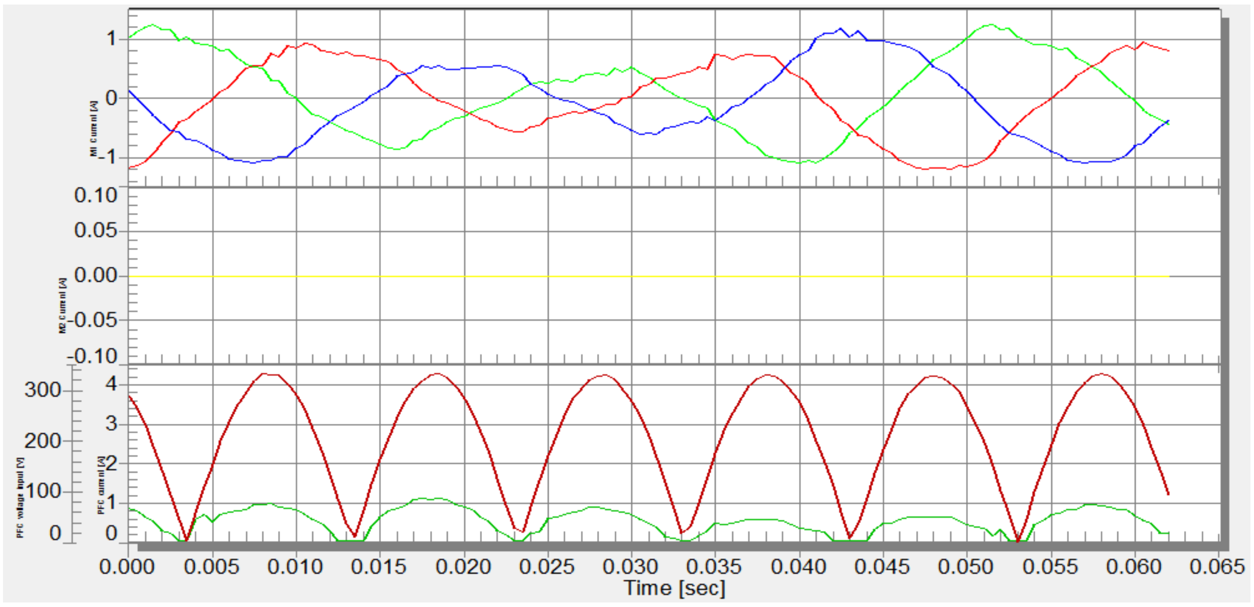
5.23 Results

Figure 54 and Figure 55 show the snapshots from FreeMASTER visualizing the motor 1 and 2 three-phase currents and the PFC's input voltage and current. The following two conditions were applied on the application:

- Input voltage 230 V and frequency 50 Hz
- Input voltage 110 V and frequency 60 Hz

The snapshots were taken for high and low load conditions where motor 1 was running at 4500 RPM (high load) or 1200 RPM (low load). The power consumption and current of motor 2 is very low.

The PFC current is maintained sinusoidal proportional to the motor 1 load. The input voltage is inversely proportional to the PFC current, that is, higher the input voltage, lower is the PFC current.



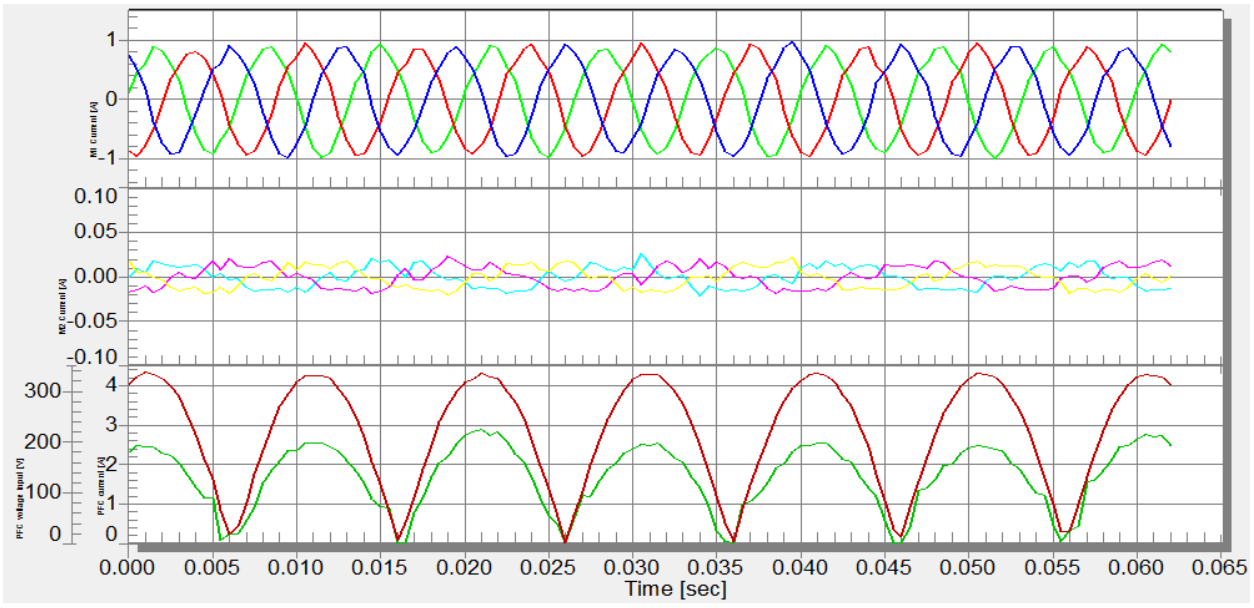
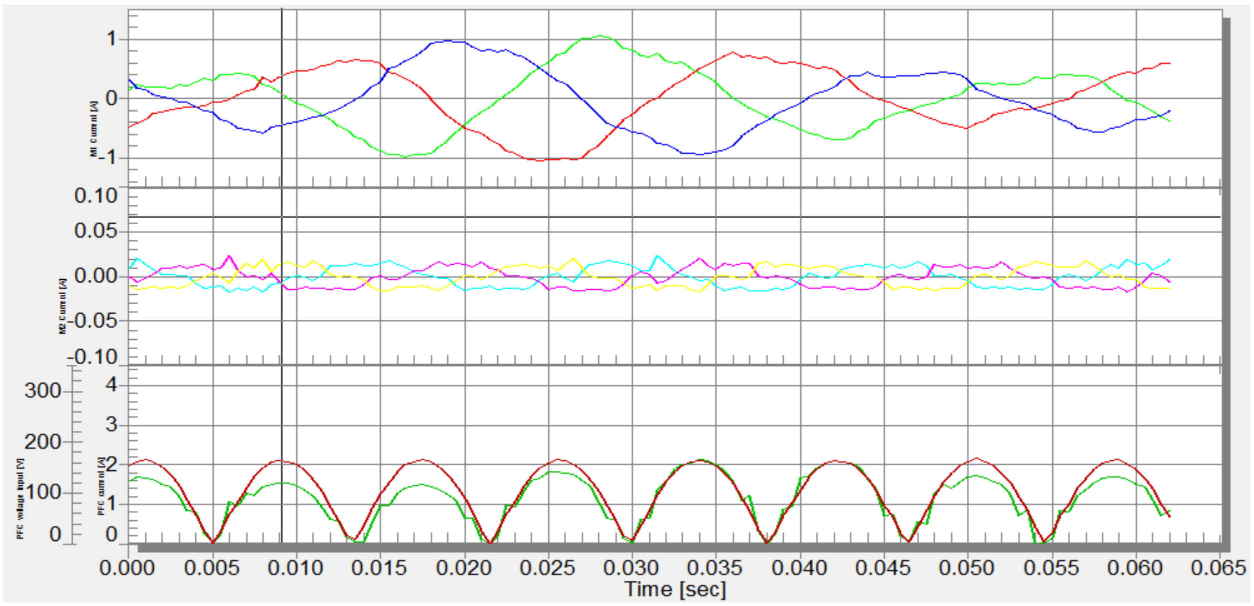


Figure 54. Motor 1 & 2 currents, PFC input voltage and current connected to 230 V/50 Hz



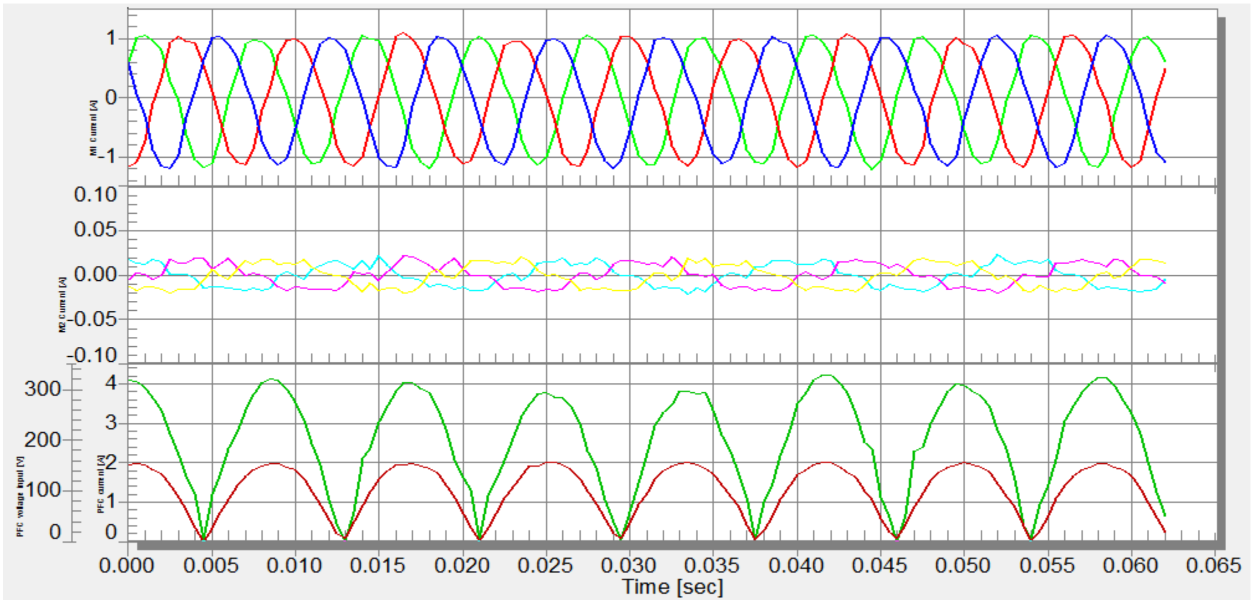


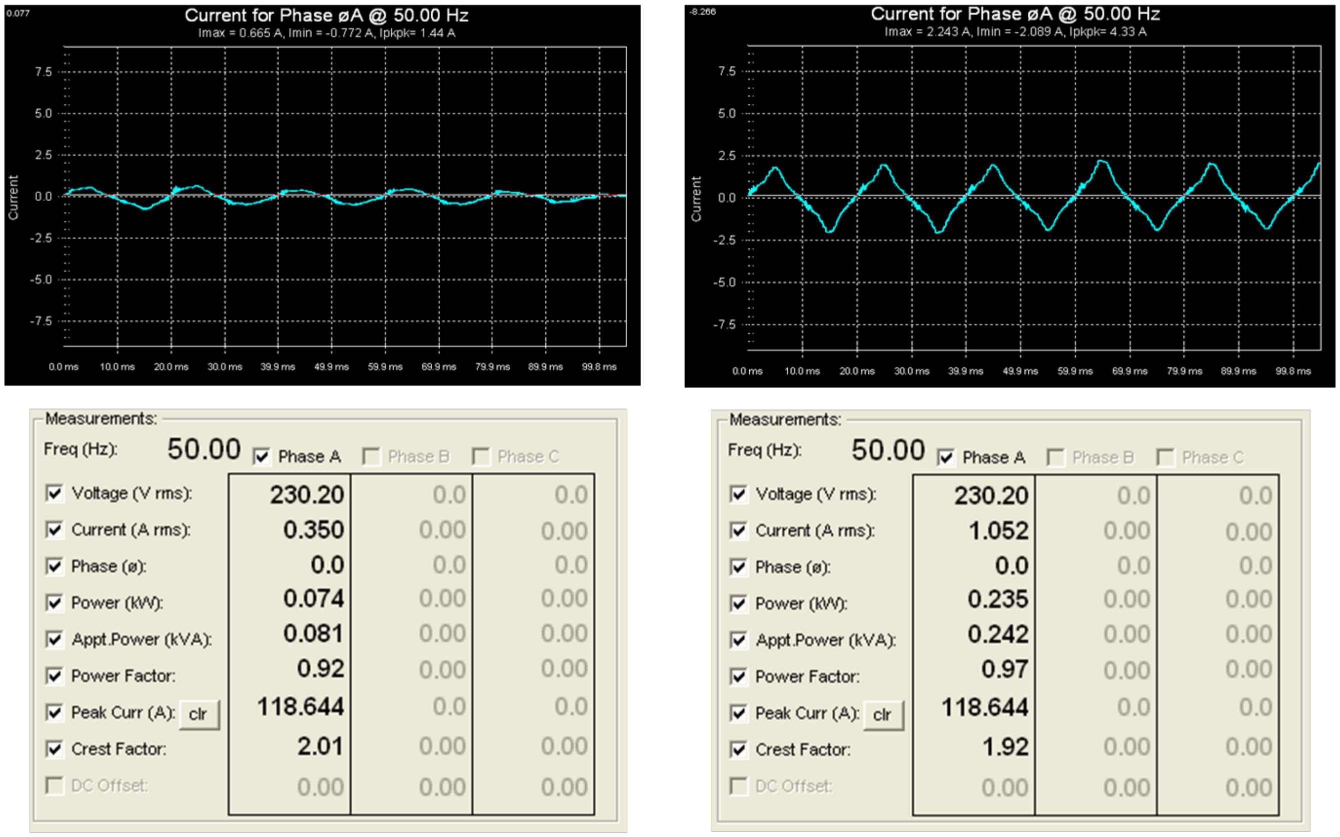
Figure 55. Motor 1 & 2 currents, PFC input voltage and current connected to 110 V / 60 Hz

The following figures show the measurement on the AC power supply. They were taken with two conditions:

- Input voltage 230 V and frequency 50 Hz (See Figure 56.)
- Input voltage 110 V and frequency 60 Hz (See Figure 57.)

With the 230 V condition, it can be observed that the current is not as sinusoidal as on the picture from FreeMASTER. This is caused by discontinuity of the PFC current. The current is measured at one point, so the PI controller controls what is measured. To fix this issue, the inductance of the PFC inductor or, the PFC switching frequency must be increased. For the condition of 110 V, where the current is higher and continuous, the current is maintained sinusoidal.

Finally, the power factor for the high load is maintained at 99 % for 110 V, and 97 % for 230 V. For the low load, it is a bit worse: 96 % for 110 V and 92 % for 220 V.


0.077
Current for Phase øA @ 50.00 Hz
Imax = 0.665 A, Imin = -0.772 A, Ipkpk= 1.44 A
Current
7.5
5.0
2.5
0.0
-2.5
-5.0
-7.5
0.0 ms
10.0 ms
20.0 ms
30.0 ms
39.9 ms
49.9 ms
59.9 ms
69.9 ms
79.9 ms
89.9 ms
99.8 ms
-8.266
Current for Phase øA @ 50.00 Hz
Imax = 2.243 A, Imin = -2.089 A, Ipkpk= 4.33 A

Measurements:

Freq (Hz): 50.00 ☑ Phase A ☐ Phase B ☐ Phase C

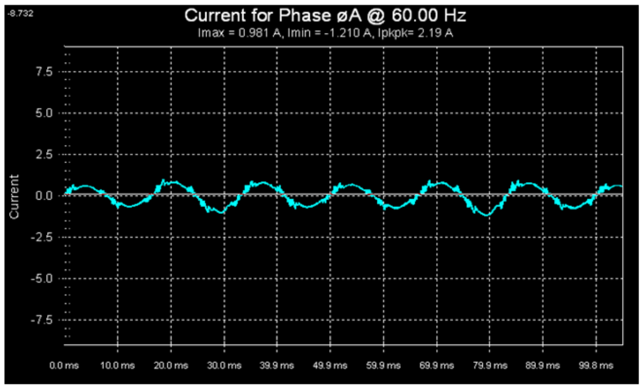
☑ Voltage (V rms):	230.20	0.0	0.0
☑ Current (A rms):	0.350	0.00	0.00
☑ Phase (ø):	0.0	0.0	0.0
☑ Power (kW):	0.074	0.00	0.00
☑ Appt.Power (kVA):	0.081	0.00	0.00
☑ Power Factor:	0.92	0.00	0.00
☑ Peak Curr (A): clr	118.644	0.0	0.0
☑ Crest Factor:	2.01	0.00	0.00
☐ DC Offset:	0.00	0.00	0.00

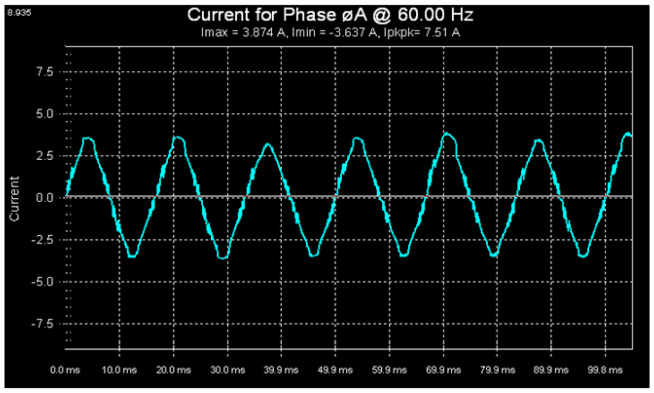
Measurements:

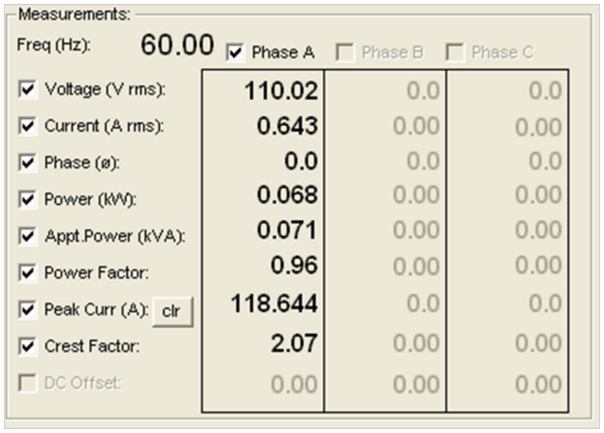
Freq (Hz): 50.00 ☑ Phase A ☐ Phase B ☐ Phase C

☑ Voltage (V rms):	230.20	0.0	0.0
☑ Current (A rms):	1.052	0.00	0.00
☑ Phase (ø):	0.0	0.0	0.0
☑ Power (kW):	0.235	0.00	0.00
☑ Appt.Power (kVA):	0.242	0.00	0.00
☑ Power Factor:	0.97	0.00	0.00
☑ Peak Curr (A): clr	118.644	0.0	0.0
☑ Crest Factor:	1.92	0.00	0.00
☐ DC Offset:	0.00	0.00	0.00

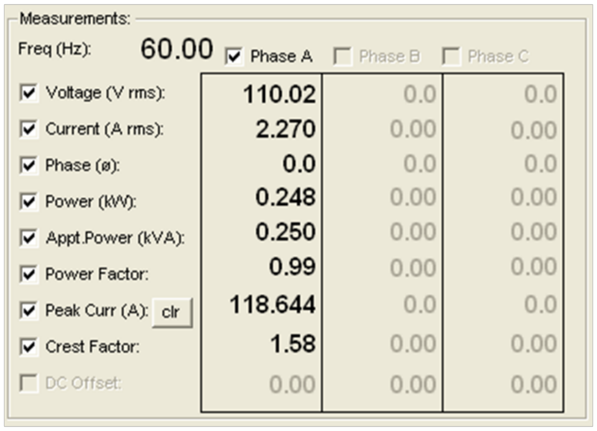
Figure 56. Power line data—230 V / 50 Hz






Measurements:

Freq (Hz): 60.00	✓ Phase A	Phase B	Phase C
✓ Voltage (V rms):	110.02	0.0	0.0
✓ Current (A rms):	0.643	0.00	0.00
✓ Phase (ø):	0.0	0.0	0.0
✓ Power (kW):	0.068	0.00	0.00
✓ Appt.Power (kVA):	0.071	0.00	0.00
✓ Power Factor:	0.96	0.00	0.00
✓ Peak Curr (A): clr	118.644	0.0	0.0
✓ Crest Factor:	2.07	0.00	0.00
DC Offset:	0.00	0.00	0.00


Measurements:

Freq (Hz): 60.00	✓ Phase A	Phase B	Phase C
✓ Voltage (V rms):	110.02	0.0	0.0
✓ Current (A rms):	2.270	0.00	0.00
✓ Phase (ø):	0.0	0.0	0.0
✓ Power (kW):	0.248	0.00	0.00
✓ Appt.Power (kVA):	0.250	0.00	0.00
✓ Power Factor:	0.99	0.00	0.00
✓ Peak Curr (A): clr	118.644	0.0	0.0
✓ Crest Factor:	1.58	0.00	0.00
DC Offset:	0.00	0.00	0.00

Figure 57. Power line data—110 V / 60 Hz

Document Number: DRM139
Rev.0, 05/2013