PMSM FOC motor control software using XMC™

XMC1300/XMC1400 and XMC4400

About this document

Scope and purpose

This document describes the implementation of Permanent Magnet Synchronous Motors (PMSM) Field Oriented Control (FOC), motor control software for a 3-phase motor using the Infineon XMC1302, XMC1402, XMC1404 or XMC4400 microcontroller.

Intended audience

This document is intended for customers who would like a configurable system for FOC control with sensorless feedback using the XMC™ series microcontroller.

Referenced documents

[1] XMC1300 AB-Step Reference Manual, XMC1000 family

[2] XMC1400 AA-Step Reference Manual, XMC1000 family

[3] XMC4400 Reference Manual

Table of contents

1 Introduction

The intention of this software is to offer functionality to drive Permanent Magnet Synchronous Motors (PMSM) in sensorless or sensor modes. It contains all the common modules necessary for the modes as generic drives, and provides a high level of configurability and modularity to address different segments.

Field Oriented Control (FOC) is a method of motor control to generate three phase sinusoidal signals which can easily be controlled in frequency and amplitude in order to minimize the current, which in turn means to maximize the efficiency. The basic idea is to transform three phase signals into two rotor-fix signals and vice-versa.

Feedback on rotor position and rotor speed is required in FOC motor control. The feedback can come from sensorless FOC or from FOC with sensors.

- Sensorless FOC derives the rotor position and rotor speed based on motor modeling, the voltage applied to the motor phases, and the current in the three motor phases.
- FOC with sensors determines the rotor position and rotor speed from rotor sensor(s), such as Hall sensors or an encoder.

Feedback on the phase currents can be measured in the motor phase, in the leg shunt or DC-Link shunt at the low-side MOSFET. In this software, phase current sensing is expected from the leg shunt or DC-Link shunt.

In the next figure we see the typical block diagram for the PMSM FOC motor control, where single shunt and three shunt low-side current sensing are supported.

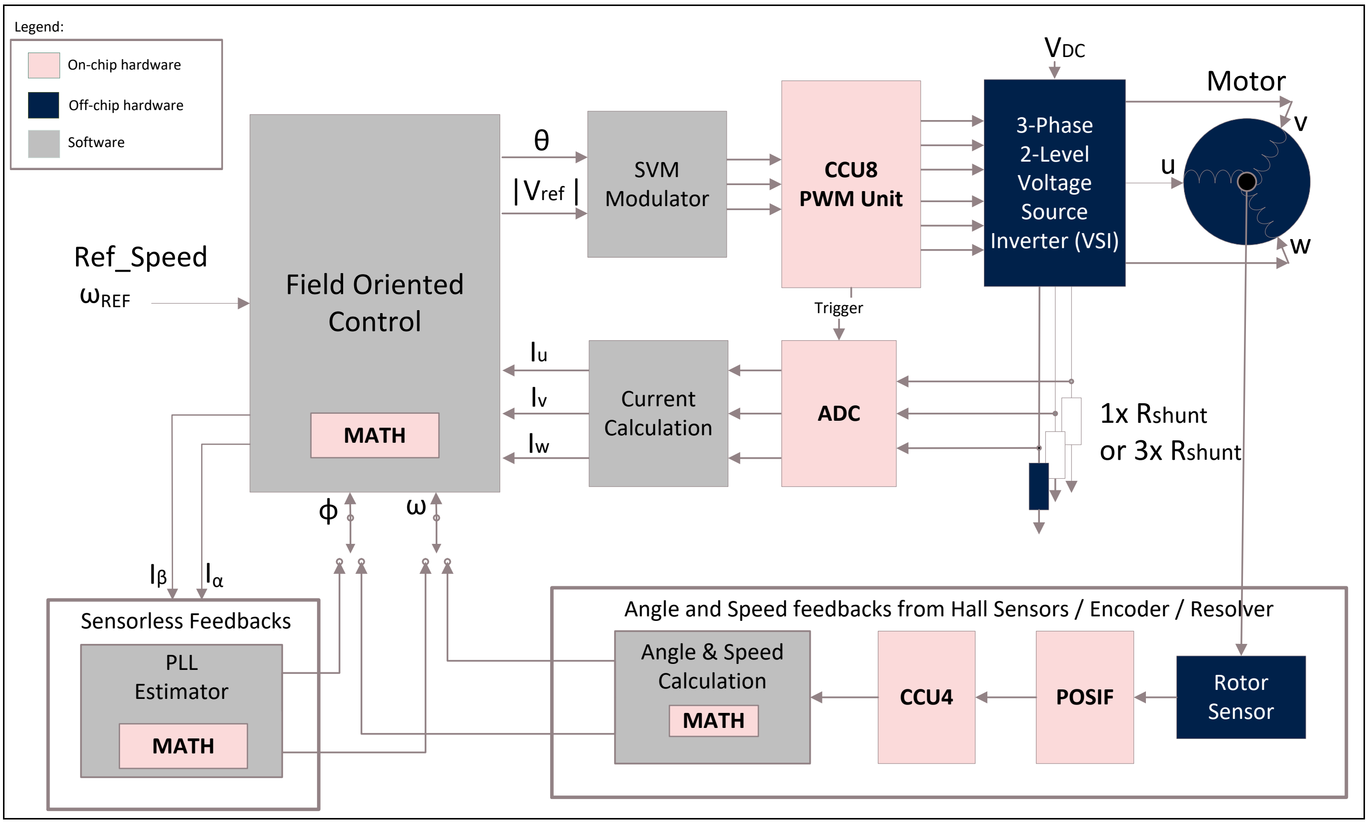


Figure 1 Block diagram of PMSM FOC motor control

1.1 Key features

Multiple Infineon innovations and unique features are included in the sensorless PMSM FOC software, such as:

- Optimized FOC
  - No Inverse Park Transform
  - Lowest cost by eliminating external Op-Amp
- SVM with Pseudo Zero Vectors (PZV), for single shunt current sensing
- MET (Maximum Efficiency Tracking) for smooth transition from V/f open-loop to FOC closed-loop
- PLL Estimator, the sensorless feedback mechanism which requires only one motor parameter, stator inductance L, for rotor speed and position feedback

The key features supported are listed in the following table:

Table 1 Key Software features supported

Feature	
Math Control Blocks	Clarke Transformation
	Park Transformation
	Id and Iq current flux/torque PI controller
	Speed PI controller
	Cartesian to Polar Transformation
	Ramp Function
Control Scheme	Speed control
	Torque control
	Vq control
Space Vector Modulation	5-segment SVM
	7-segment SVM
	Pseudo Zero Vector SVM; 4-segment SVM
Low Side Current Sensing	Leg shunt: 3/2 support, over-modulation and 7-segment SVM
	DC link single shunt: Support PZV and 4-segment SVM, no over-modulation
Start-up Algorithm	Direct FOC start-up
Protection	Phase over-current protection
	DC link under voltage & over-voltage Protection
Device Feature	ADC on-chip gain for current sensing
	ADC synchronous conversion: motor phase current sensing
Control Feature	Motor control state machine
	DC-bus voltage clamping during fast braking
	Motor stop - brake
Rotor Speed and Angle Calculation	Sensorless PLL Estimator using HW CORDIC
Others	S-curve Ramp generator / Linear Ramp generator
	PI anti-windup for Speed control

Feature	
	dq-axis decoupling

1.2 Abbreviations and acronyms

Table 2 Abbreviations and acronyms used in this document

Term	Definition
API	Application Programming Interface
BOM	Bill of Material
CCU8	Capture Compare Unit 8
CPU	Central Processing Unit
FOC	Field Oriented Control
GPIO	General Purpose Input / Output
IP	Intellectual Property
ISR	Interrupt Service Routine
LLD	Low Level Driver
MCU	Microcontroller Unit
MET	Maximum Efficiency Tracking
PI	Proportional Integral Controller
PLL	Phase Locked Loop
PMSM	Permanent Magnet Synchronous Motors
PWM	Pulse Width Modulation
PZV	Pseudo Zero Vector
SRAM	Static Random Access Memory
SVM	Space Vector Modulation
UART	Universal Asynchronous Receiver Transmitter
VADC	Versatile Analog-to-Digital Converter
XMC	XMC™ MCU family based on ARM®
XMC1000	XMC™ MCU family based on ARM® Cortex®-M0 core
XMC1300	XMC™ MCU series with 32MHz Core and 64MHz Peripheral frequency
XMC1302	XMC™ MCU product with specific feature set. E.g. Cordic
XMC1400	XMC™ MCU series with 48MHz Core and 96MHz Peripheral frequency
XMC1402	XMC™ MCU product with specific feature set. E.g. Cordic
XMC1404	XMC™ MCU product with specific feature set. E.g. Cordic and CAN
XMC4400	XMC™ MCU product with specific feature set. E.g. FPU

1.3 XMC™ resource allocation

The XMC1302, XMC1402, XMC1404 and XMC4400 microcontroller are all ideal for PMSM FOC motor control systems. They have dedicated motor control peripherals, POSIF, CCU8, ADC, and CCU4. In XMC1300/XMC1400, the CORDIC coprocessor is used for mathematical calculations, whereas a floating point unit (FPU) is used in XMC4400. In this PMSM FOC motor control software, the hardware peripherals used are listed in the table that follows.

Note: The default resource allocations are for the Infineon XMC1000/XMC4400 Motor Control Application Kit (order numbers KIT_XMC1X_AK_MOTOR_001 and KIT_XMC4400_DC_V1 respectively).

Table 3 XMC™ Peripherals used for sensorless PMSM FOC with three shunt current sensing

XMC™ peripherals	Usage	Default resource allocation in XMC1300/XMC1400	Default resource allocation In XMC4400
CCU80	PWM Generation for Phase U	CCU80 slice 0	CCU80 slice 0
	PWM Generation for Phase V	CCU80 slice 1	CCU80 slice 1
	PWM Generation for Phase W	CCU80 slice 2	CCU80 slice 3
	Timer for ADC Trigger	CCU80 slice 3	CCU80 slice 2
VADC	Phase U Current sensing	G0 channel 4[1] G1 channel 3[1]	G0 channel 1
	Phase V Current sensing	G0 channel 3[1] G1 channel 2[1]	G1 channel 7
	Phase W Current sensing	G0 channel 2[1] G1 channel 4[1]	G2 channel 1
	Alias Channels for three shunt synchronous conversion	G0 channel 0 G0 channel 1 G1 channel 0 G1 channel 1	G0 channel 0 G1 channel 0 G2 channel 0
	DC link Voltage sensing	G1 channel 5	G1 channel 1
	DC link average Current sensing	G1 channel 6	G1 channel 0 (not implemented)
	Potentiometer for speed change	G1 channel 7	G1 channel 5
MATH		CORDIC	FPU
Nested Vectored Interrupt Controller (NVIC)	PWM Period Match Interrupt	CCU80.SR0	CCU80.SR0
	CTrap Interrupt	CCU80.SR1	CCU80.SR3
	ADC Interrupt for single shunt sensing	VADC0.G1SR1[2]	Not available

[1] The same input pin must connect to both the ADC group channels to perform synchronous sampling. Refer to **chapter 3.2.1** for details.
[2] If DC link current sensing uses the VADC Group0 channels, then the VDC0.G0SR1 interrupt node is used.

1.4 XMC™ hardware modules inter-connectivity

The XMC1000 and XMC4400 families have comprehensive hardware inter-connectivity.

The figure below shows the interconnections between XMC1302 hardware peripheral modules. This is valid for XMC1402 and XMC1404 also.

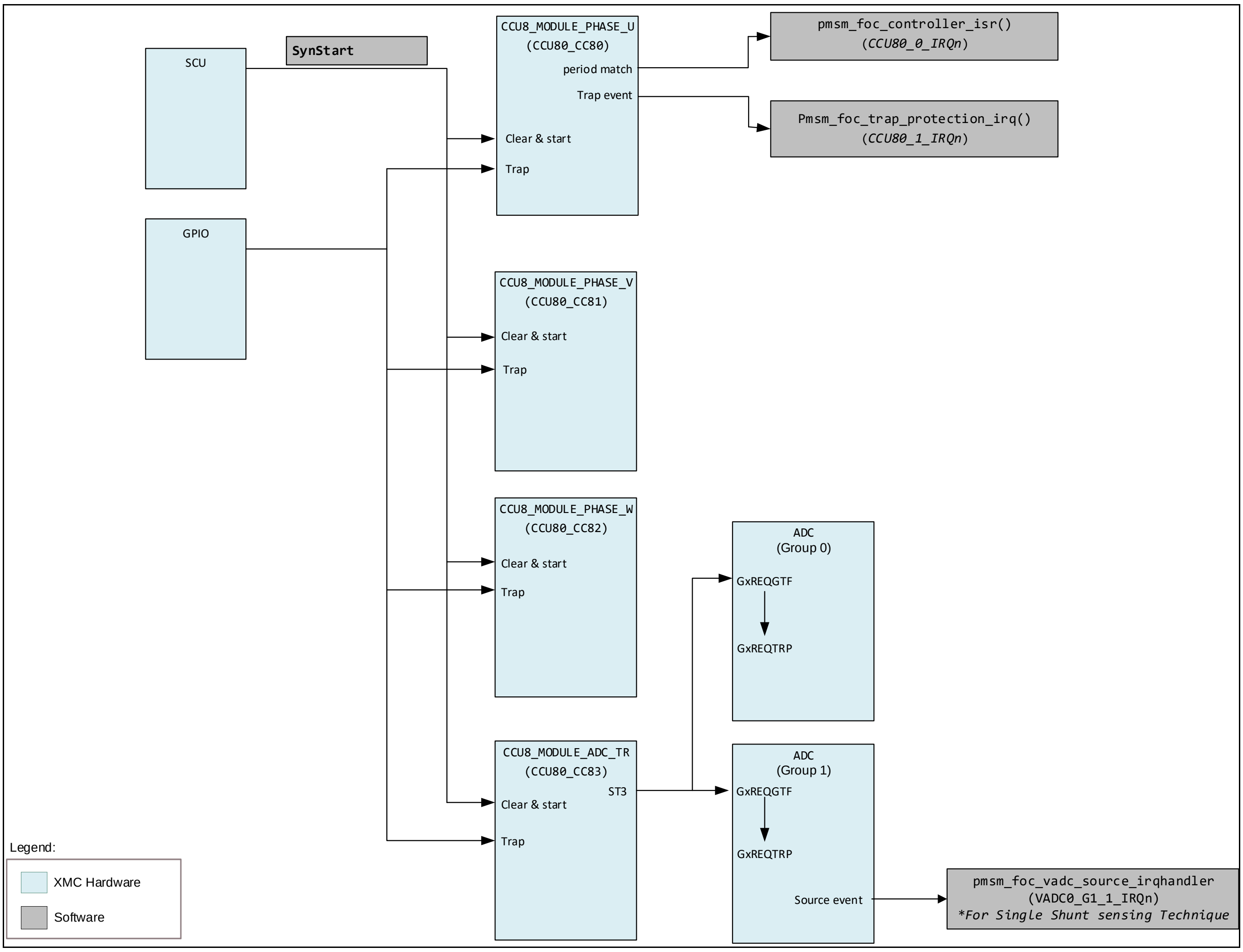


Figure 2 XMC1302 hardware interconnection

Note:

1. *The CCU8 slice timers are started synchronously with the sync start signal from the SCU (System Control Unit).*
2. *The VADC conversion is triggered by the CCU8 Slice 3 compare match status signal.*

Figure 3 shows the interconnections between the XMC4400 hardware peripheral modules.

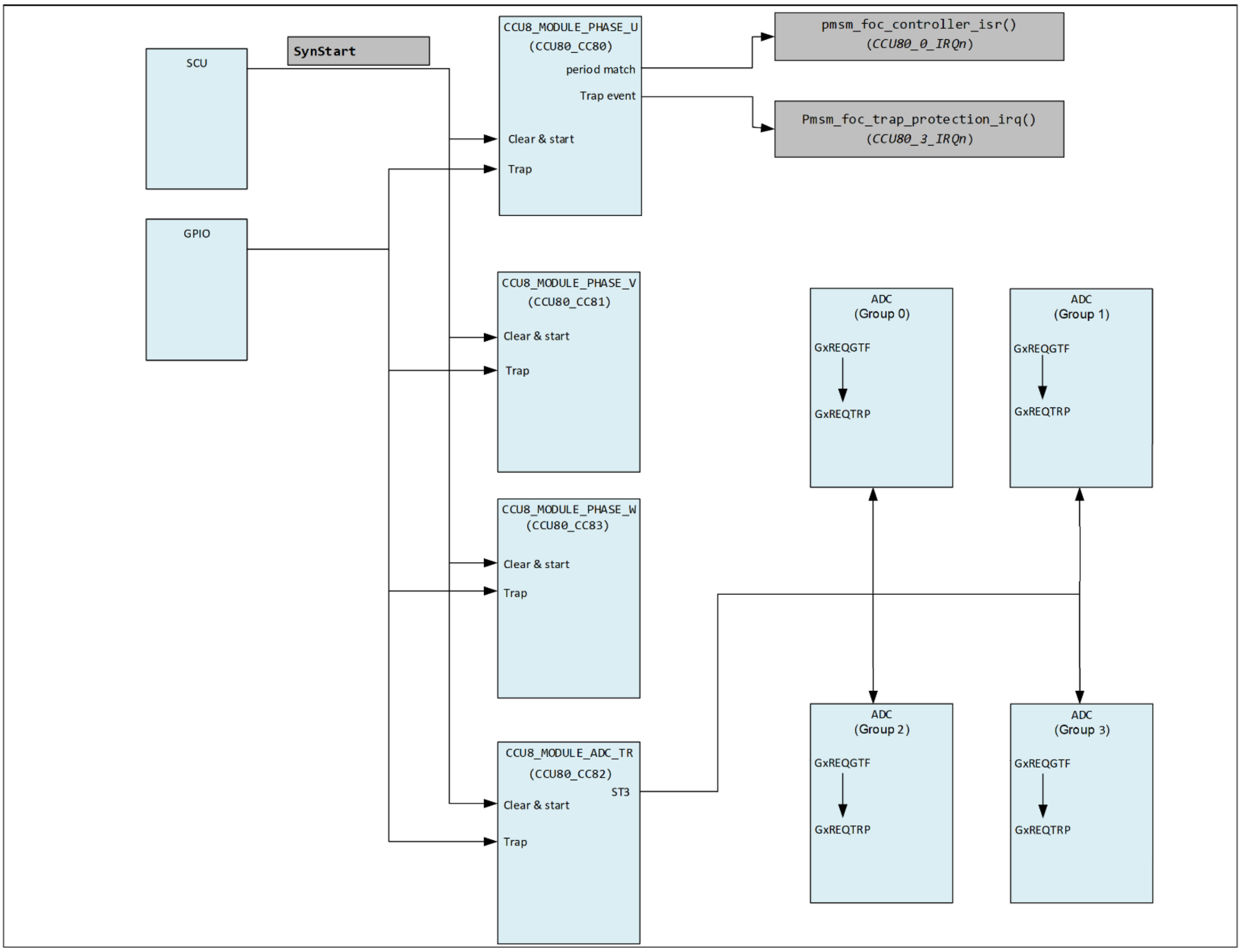


Figure 3 XMC4400 hardware interconnection

Note:

3. *The CCU8 slice timers are started synchronously with the sync start signal from the SCU (System Control Unit).*
4. *The VADC conversion is triggered by the CCU8 Slice 2 compare match status signal.*

1.5 Execution time and memory usage

In the XMC1000 and the XMC4000 families, access to the Static Random Access Memory (SRAM) requires no wait state. The major parts of the software are executed from the SRAM. The Interrupt Service Routines (ISRs), all the mathematical blocks of the FOC algorithm, the SVM, and the motor phase current sensing and calculation are executed in the SRAM. This improves the performance as the execution time to run the FOC algorithm is reduced by approximately 30%.

*Note: Please refer to **chapter 9** for the list of APIs running in SRAM.*

Breakdown of the memory usage and CPU time-utilization are provided in the following table based on the default settings for the Infineon XMC1000 Motor Control Application Kit (XMC1302), the XMC1400 Boot Kit (XMC1404), and the XMC4400 Motor Control Application Kit.

- Control Scheme
  - Open-Loop to FOC Closed Loop Speed Control
- Current Sensing Technique
  - Three shunt synchronous ADC conversion

Table 4 CPU utilization and memory usage for three shunt current sensing with XMC1300 and XMC1400

PWM frequency	20 kHz – Interrupt Service Routine runs every 50 µs		
DAVE™ 4 GCC compiler optimization level	Optimized most (-O3)		
MCU	**XMC1300**	**XMC1400**	**XMC4400**
CPU utilization	31 µs (62%)	21 µs (42%)	9.7 µs (19.4%)
Flash code size (bytes)	10416	10822	24790
SRAM code size (bytes)	2376	2376	3060
SRAM data size (bytes)	348	352	176

1.6 Software overview

The PMSM FOC motor control software is developed based on a well-defined layered approach.

The layered architecture is designed in such a way as to separate the modules into groups. This allows different modules in a given layer to be easily replaced without affecting the performance in other modules and the structure of the complete system.

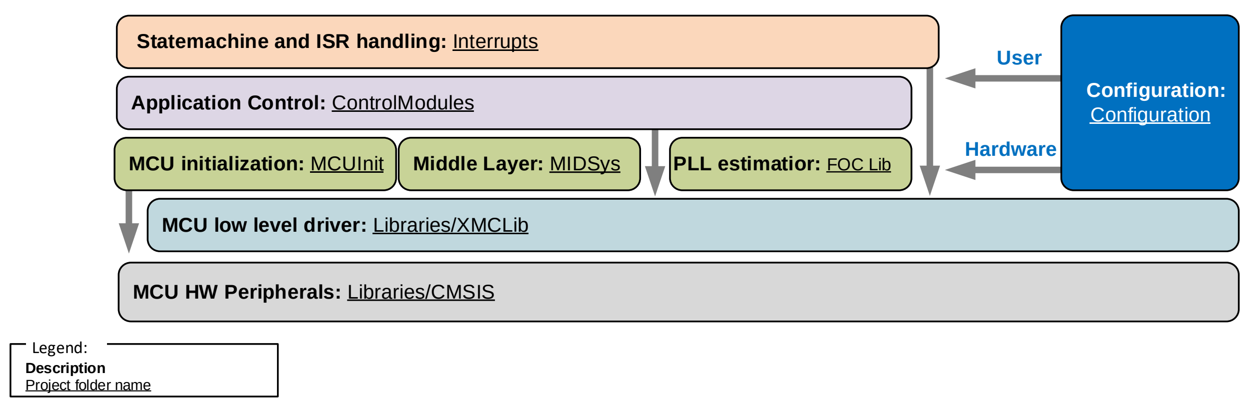


Figure 4 PMSM FOC software overview; layered structure

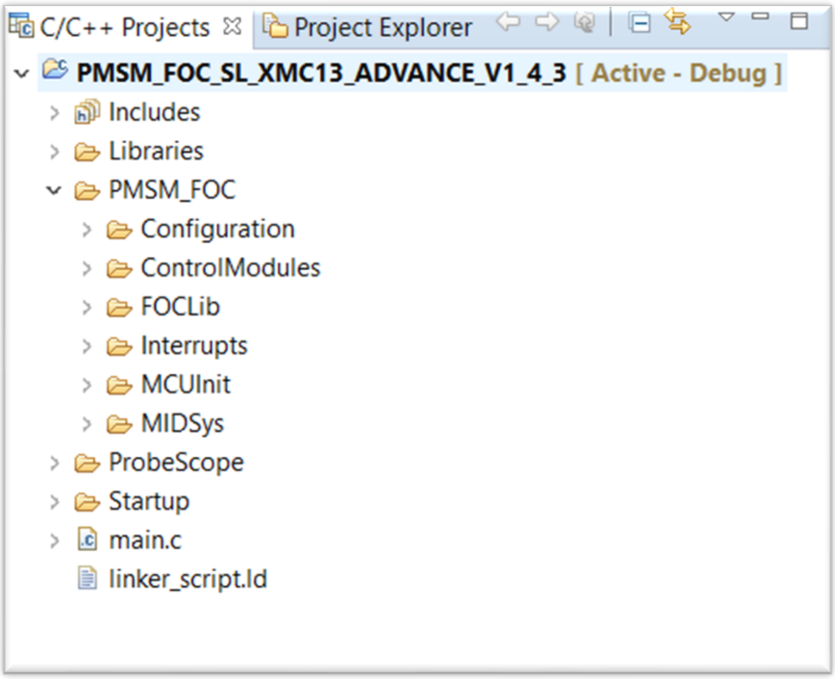


Figure 5 Project folder structure

State machine and ISR handling: Interrupts

This layer consists of a CCU8 trap interrupt handling function, a CCU8 period match ISR function, and a VADC ISR function for shunt current sensing. All files are stored in the 'Interrupts' folder.

Application Control: Control Modules

This layer consists of FOC SW control modules. This includes the Clarke Transform, Park Transform, Cartesian to Polar, current reconstruction, PI controller, Open Loop, and Ramping for example.

All the routines mentioned are called from the CCU80 period match Interrupt Service Routines.

All files for this layer can be found in the folder 'ControlModules'.

Configuration: Configuration

The configuration is divided. The user configuration effects the general behavior of the software. The file pmsm_foc_user_config.h can be modified. Pre-defined hardware kits are available.

The hardware configuration allows for more detailed adaptation to the customer hardware.

This layer is divided into:

- Controller Card
- Inverter Card
- Motors

The specific associated *.h files can be found in the associated folders.

The static configuration and required scaling are accessible with the following files:

- pmsm_foc_const.h
- pmsm_foc_macro.h
- pmsm_foc_variable_scaling.h

Note: You should not change these configuration and definition files.

All configurations can be found in the folder 'Configuration'.

PLL estimator: FOCLib

Infineon patented IP, and PLL Estimator, is provided as a compiled .a library file.

The file can be found in the folder 'FOCLib'.

Middle Layer: MIDSys

This layer provides routines for PWM generation, ADC measurements, and angle and speed information to the FOC control module layer. The main purpose of this layer is to give flexibility to add or remove a sensor feedback module into the FOC software. For example, when using Hall sensors, you can add in files in this layer to provide position and feedback from the Hall sensors without making huge changes to the layers on top. This layer also provides a mathematical library for XMC4400.

All files for this layer can be found in the folder 'MIDSys'.

MCU Initialization: MCUInit

This layer controls the initialization of all MCU peripherals. It contains XMCLib data structure initialization and peripheral initialization functions. This layer closely interacts with XMCLib and the MIDSys layer to configure each peripheral.

All files for this layer can be found in the folder 'MCUInit'.

MCU low level driver: Libraries/XMCLib

MCU hardware Peripherals: Libraries/CMSIS

This layer is the hardware abstraction layer to the MCU peripherals.

All files for this layer can be found in the folder 'Libraries'.

1.7 Limitations of use for PMSM FOC software

For this application note the current software version used is PMSM FOC software v1.5.x.

At the time of release of this example software, the following limitations in usage apply:

- Only a single motor drive is supported.
  - Dual motor control support is not available
- Position and speed feedback from Hall sensors/encoders is not supported
- This software is developed in DAVE™ version 4. It is not tested on other IDE (Integrated Development Environment) platforms
- The following are not currently documented:
  - Scaling for pmsm_foc_set_motor_target_torque().
  - Scaling for pmsm_foc_set_motor_target_voltage().
- No T_{min} available for current measurement.
- No support for driver delay. This value is used to shift the ADC trigger to compensate the driver IC delay. For example, with this shift it is ensured to start the 3 shunt measurement in the middle center of the PWM pattern.
- Over-Current Protection is through a DC link shunt. This protects the inverter but current between phases of the motor is not measured.
- PT1 filter is not documented.
- No catch-free running implementation
- UART DEBUG is available only in XMC1300/XMC1400, and it is not tested.
- SETTING_TARGET_SPEED options BY_POT_ONLY, and BY_UART_ONLY are not tested.
- In XMC4400, single shunt sensing is not implemented.

2 PMSM FOC sensorless software components

The intention of this PMSM FOC motor control software is to offer functionality to drive the PMSM motors with sensors or sensorless modules. The current version supports sensorless modules.

The PMSM FOC software provides a high level of configurability and modularity to address different motor control applications.

Five types of control scheme are supported:

- Speed Control Transition FOC Startup
- Speed Control Direct FOC Startup
- Torque Control Direct FOC Startup
- Vq Control Direct FOC Startup
- Open-Loop Voltage Control

The major components of the PMSM FOC software are shown in the following diagram. Each of the modules is described and referenced.

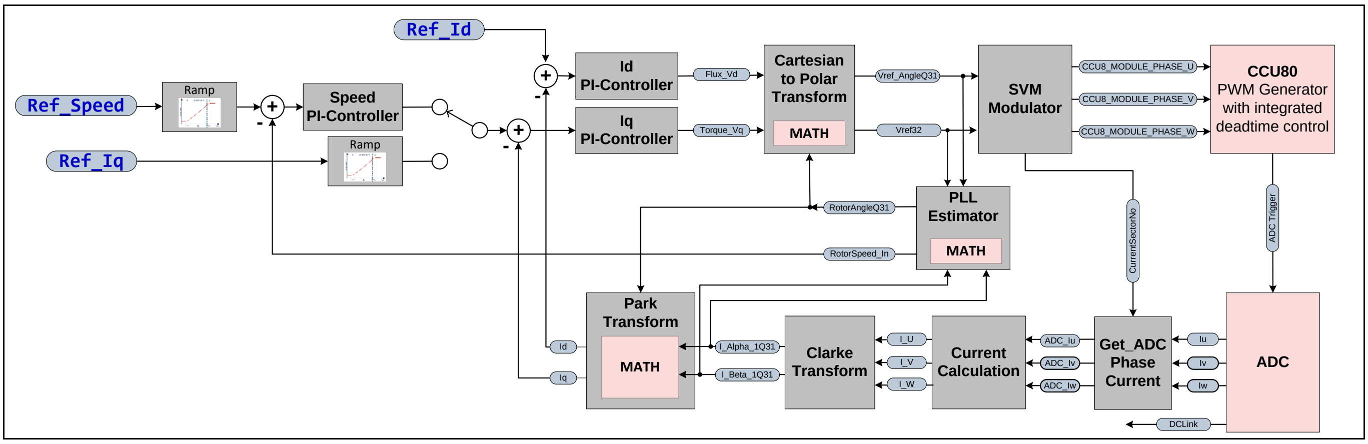


Figure 6 PMSM FOC block diagram

2.1 Motor start / speed change / motor stop operations control

The motor is started with the start command.

The target speed can be changed between:

- USER_SPEED_LOW_LIMIT_RPM
- USER_SPEED_HIGH_LIMIT_RPM

One option for controlling is a potentiometer.

The relationship between the ADC data and the motor target speed for speed control scheme is shown in the figure below.

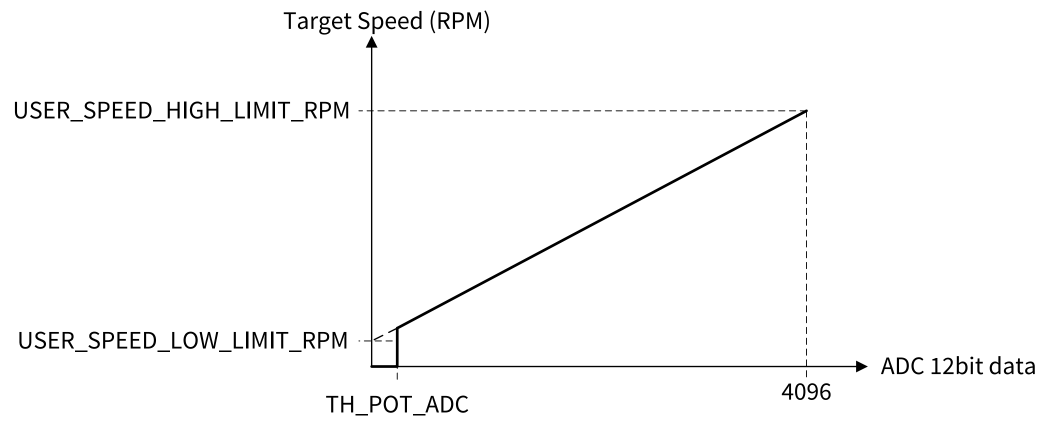


Figure 7 Potentiometer ADC value versus speed in speed control scheme

From the state FOC CLOSED LOOP, three options are available to prevent the motor from running:

- Set target speed below EXIT_FOC_SPEED
- Call the break function pmsm_foc_motor_break()
- Call the stop function pmsm_foc_motor_stop()

Set target speed below EXIT_FOC_SPEED

If the motor speed is below 10% of the maximum speed the state is changed to MOTOR_HOLD.

A 50% PWM ON/OFF is applied to all phases. The motor is held.

Call the break function pmsm_foc_motor_break()

The software does not accept any target speed and ramps down the motor to the limit FOC_EXIT_SPEED. After crossing this limit the state is changed to MOTOR_STOP. The inverter is disabled and the output set to tristate. The result is an uncontrolled freewheeling. Because of the ramp-down the motor speed should be very low.

Call the stop function pmsm_foc_motor_stop()

The state is immediately changed to MOTOR STOP. The inverter is disabled and the output set to tristate. The result is an uncontrolled freewheeling. Depending on the current motor speed and the friction, the motor should run in a freewheeling state.

2.2 Ramp generator

PMSM FOC motor control software provides two types of ramp functions:

- Linear curve
- S-curve

An input parameter is ramped from an initial value to an end value. The ramp generator input is connected to a user set value or to an analog input, depending on your configuration.

The ramp-up rate in the linear region is defined as USER_SPEED_RAMPUP_RPM_PER_S in the user configuration file, pmsm_foc_motor_XXXX.h (refer to **chapter 7.2.3.2**). The ramp generator function is called every PWM frequency cycle.

In the S-curve ramp generator function, the initial ramp-up rate is half of the defined ramp-up rate. The ramp rate is slowly increased to the defined value. This generates the first s-curve.

The second S-curve starts when the speed is SPEED_TH_2ND_S from the Ref_Speed.

The constant SPEED_TH_2ND_S is defined in the pmsm_foc_interface.c file.

The S-curve ramp generator is only used in the speed control.

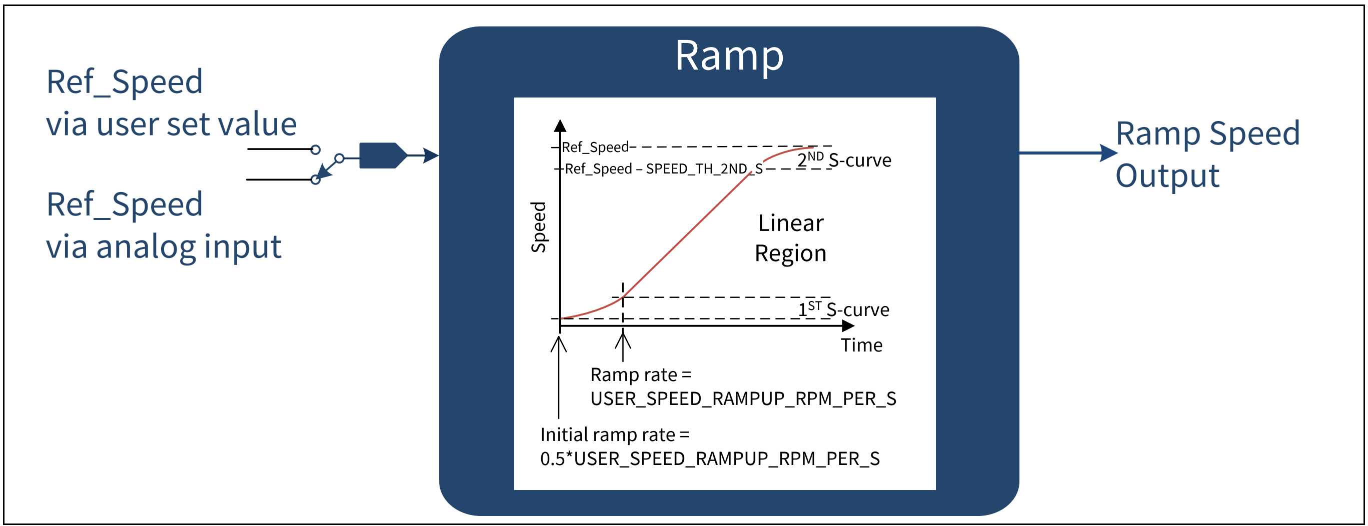


Figure 8 S-curve ramp generator

In both ramp generator functions, the DC link voltage is monitored during ramp-down operation. It stops the speed/torque ramp-down if the DC link voltage is over the user configured voltage limit. This is to avoid an over-voltage condition during the fast braking. This voltage limit, VDC_MAX_LIMIT, is defined in the header file pmsm_foc_variables_scaling.h.

The ramp output is the reference signal to the control scheme. For the linear ramp generator, depending on the control scheme selected, the ramp output can be speed, torque current, or torque Vq.

2.3 Control schemes

In this software block the control schemes for the 3-phase PMSM FOC motor can be:

- Open loop voltage control
- Speed control
- Torque control
- Vq control

2.3.1 Open loop voltage control

In an open loop voltage control, a reference voltage (Vref) is used to cause the power inverter to generate a given voltage at the motor. The mechanical load influences the speed and the current of the PMSM motor.

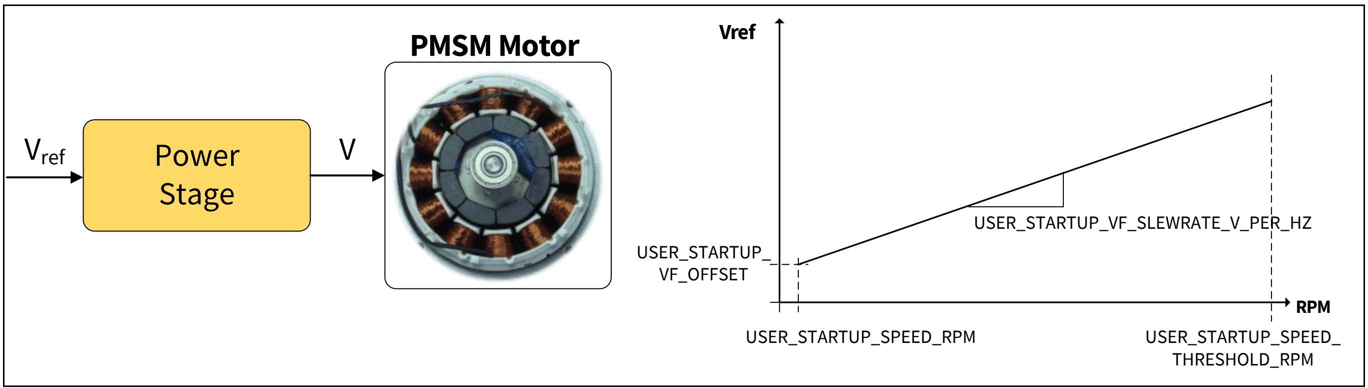


Figure 9 Open loop voltage control

2.3.2 Speed control

A speed control scheme is a closed loop control. This scheme uses a cascaded speed and currents control structure. This is due to the change response requirement for a speed control loop which is much slower than the one for current loop.

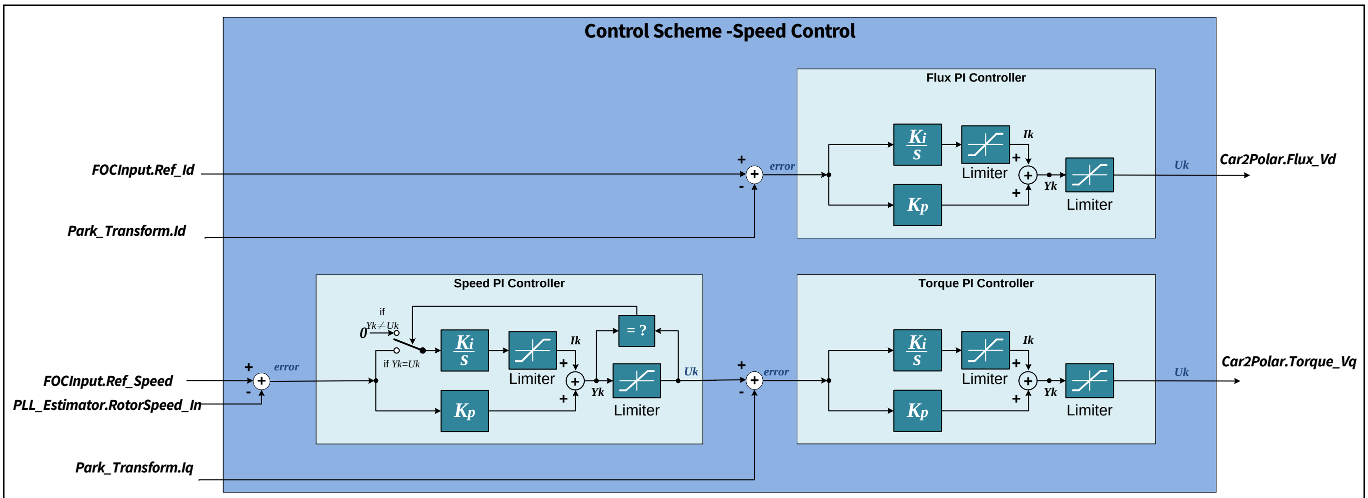


Figure 10 Speed control

Direct FOC startup and transition startup (open loop to closed loop) modes are supported in speed control.

The speed PI controller supports integral anti-windup. The integral output is held stable when either PI output or integral output reaches its limit.

The output of the speed PI is used as the reference for the torque PI controller.

Transition mode – 3 steps startup with Maximum Efficiency Tracker (MET)

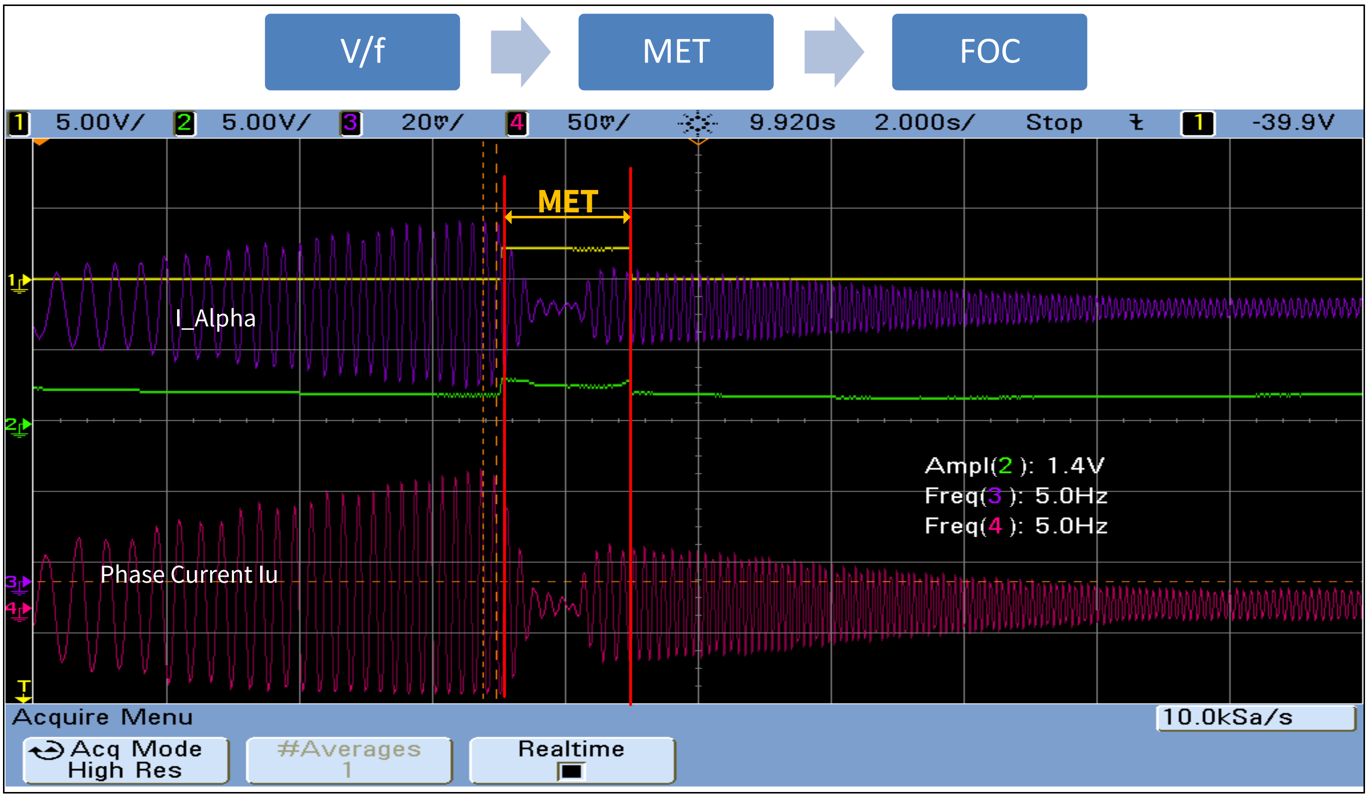


Figure 11 3 steps motor startup mechanism

The three steps are:

1. The motor starts in V/f open loop control state and ramp-up to a user defined startup speed.
2. Sensorless MET closed-loop control state takes over. This state is added to ensure the stator flux is perpendicular to the rotor flux in a smooth and controlled way.
3. The state machine switches to FOC_CLOSED_LOOP state and ramps up the motor speed to the user defined target speed.

Advantages:

- High energy efficiency MET and FOC closed-loop
- Smooth transitions for all the three steps
- V/f open-loop -> MET closed loop at low motor speed, therefore low startup power

2.3.3 Torque control direct startup

PMSM FOC motor control software provides direct startup torque control. The control loop consists of the d-axis (Flux) and q-axis (Torque) PI controllers. The motor torque is maintained at torque reference value (Iq_ref). Any change in the load will cause the speed of the motor to change but the torque remains constant.

This control scheme is useful in applications where direct current control is important, such as e-bike or battery-operated devices for example.

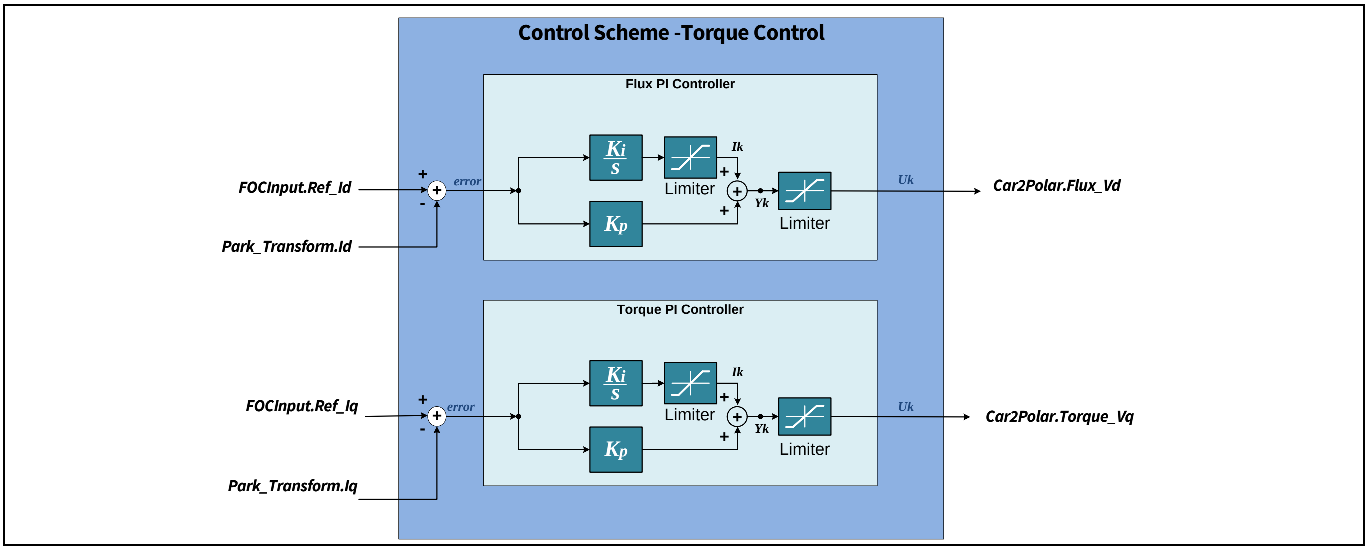


Figure 12 Torque control scheme

2.3.4 Vq control direct startup

The Vq control is used when a fast response is required and varying speed is not a concern.

The speed PI control loop and torque PI control loop are disabled.

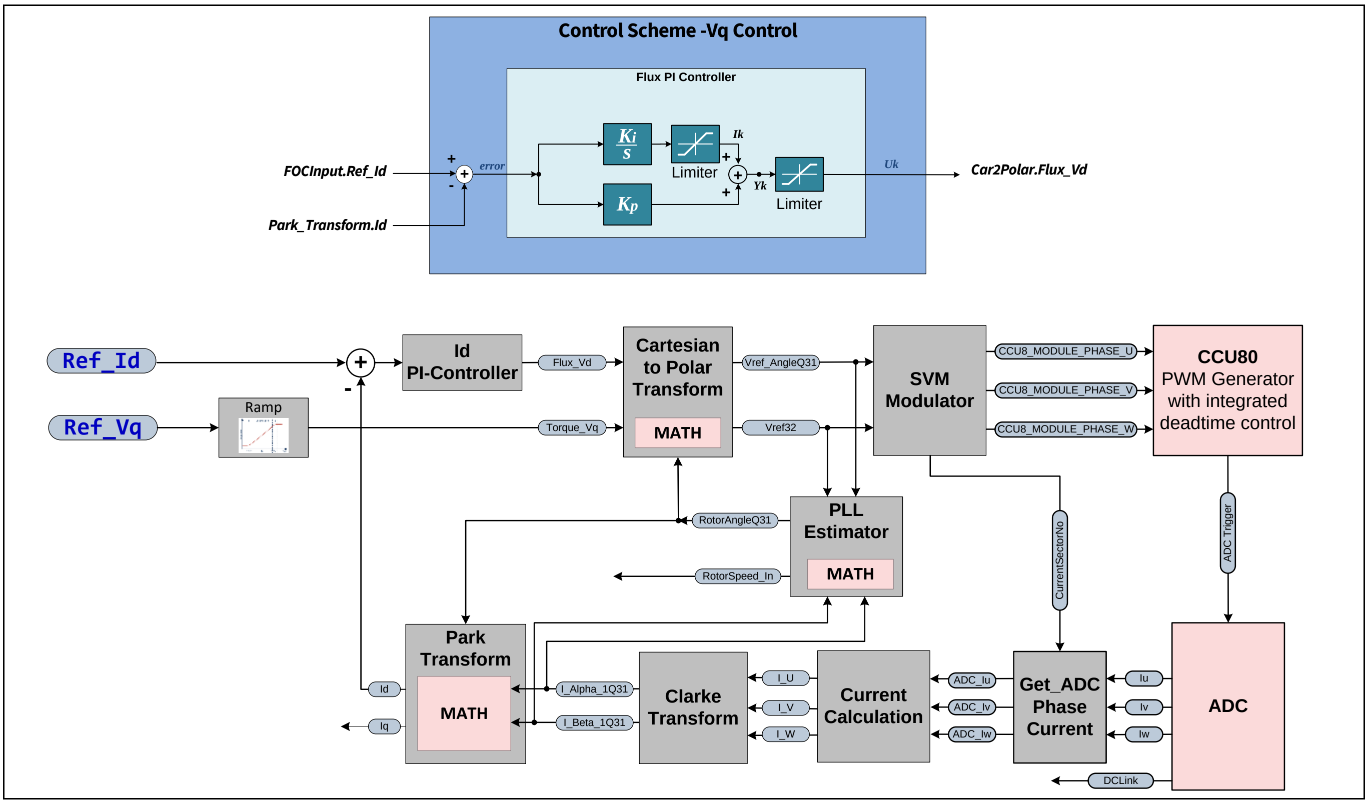


Figure 13 Vq control scheme

2.3.5 D-axis and Q-axis decoupling

The control of I_d and I_q currents are not independent from one another. The I_d current has an effect on the I_q current and vice-versa.

This coupling effect acts as a disturbance which becomes prominent during transient conditions at high speed. To correct for this coupling effect, feed-forward decoupling is applied to each axis to remove the disturbance.

Torque voltage, $V_q = V_q - \omega L_q I_q$
Flux voltage, $V_d = V_d + \omega L_d I_d$

Assuming the torque inductance and the flux inductance are equal, ω represents the estimated speed from the PLL estimator output.

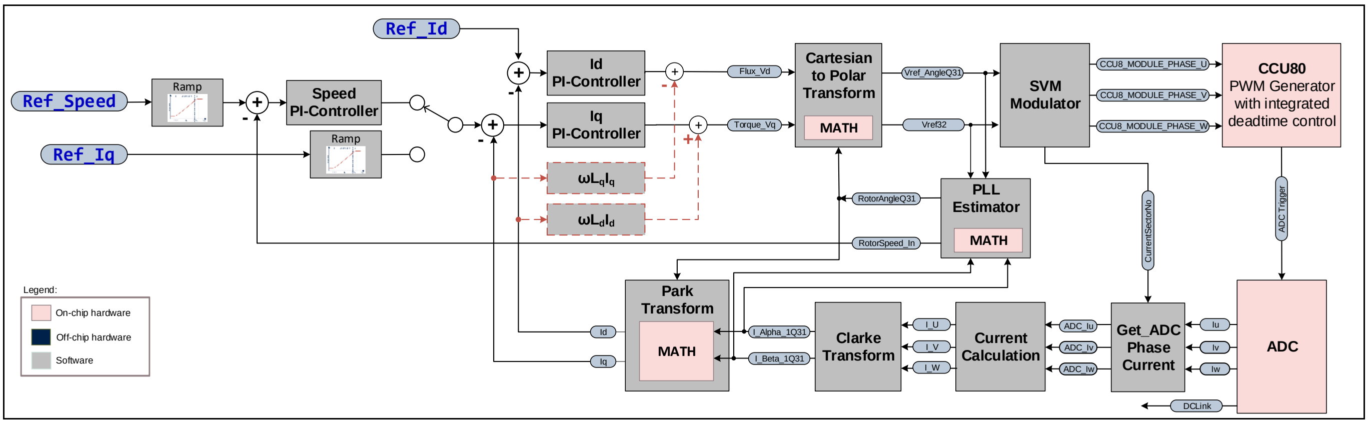


Figure 14 FOC control with dq decoupling

2.4 Cartesian to Polar transform

Using the outputs from the torque and flux PI controllers in XMC1300/XMC1400, the Cartesian to Polar transform is calculated with the hardware CORDIC coprocessor in circular vectoring mode.

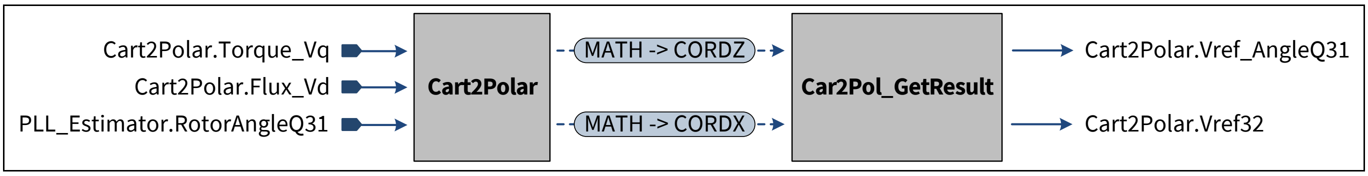


Figure 15 Cartesian to Polar transform

Table 5 XMC1000 CORDIC settings for Cartesian to Polar transform

Parameters	Settings
CORDIC Control Mode	Circular Vectoring Mode
K	≈ 1.646760258121
Magnitude Prescaler, MPS	2
CORDX	Cart2Polar.Flux_Vd
CORDY	Cart2Polar.Torque_Vq
CORDZ	PLL_Estimator.RotorAngleQ31

According equations:

$$CORDX = K * \left|V_{ref\,32}\right| / MPS = K * \sqrt{V_{Flux\_Vd}{}^2 + V_{Torque\_Vq}{}^2} / MPS,$$

$$\left|V_{ref\,32}\right| = CORDX * MPS / K = CORDX * 1.2145$$

$$V_{ref\_AngleQ31}, \theta = CORDZ = RotorAngleQ31 + \arctan\left(\frac{V_{Torque\_Vq}}{V_{Fluz\_Vd}}\right)$$

To improve the execution the function is divided into two, for parallel processing; the start CORDIC function and the read CORDIC result function. While the CORDIC coprocessor is making the calculation, the CPU can execute other software functions such as the PI controller for example, and read the CORDIC result later. Instead, in XMC4400, the calculations are done immediately in FPU (not in parallel) calling fpu_cart2polar()

function implemented in fpu_math2 library. The results from the Cartesian to Polar transform are fed into the Space Vector Modulation (SVM) module.

2.5 Space Vector Modulation (SVM)

Space Vector Modulation (SVM) transforms the stator voltage vectors into Pulse Width Modulation (PWM) signals (compare match values).

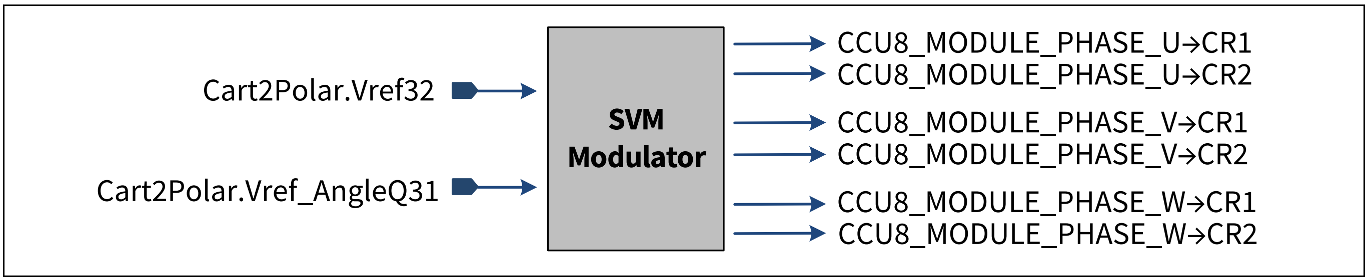


Figure 16 PWM SVM function

PMSM FOC motor control software supports different modes of SVM:

- 7-segment SVM
- 5-segment SVM
- SVM with Pseudo Zero Vector
- 4-segment SVM
- Over-modulation SVM

Each CCU80 timer slice controls an inverter phase with complementary outputs. Dead-time is inserted to prevent a DC link voltage short-circuit. The dead-time value is configured by the user in the header file; pmsm_foc_invertercard_parameter.h

The timer counting scheme used in the CCU80 is asymmetrical edge aligned mode. This is to have the same CCU80 settings for 7-segment SVM, 4-segments SVM, and Pseudo Zero Vector PWM. With the asymmetric mode, there is more flexibility for sampling shunt currents via the ADC.

The default initial settings of the CCU80 module are for the Infineon Motor Control Application Kit XMC1300 KIT_XMC1X_AK_MOTOR_001, and XMC4400 KIT_XMC4400_DC_V1.

Table 6 CCU80 default initial settings for 3-phase SVM generation

Parameters	Settings for XMC1300/XMC1400 and XMC4400
Timer Counting Mode	Edge aligned mode
Shadow Transfer on Clear	Enabled
Prescaler mode	Normal
Passive level	Low
Asymmetric PWM	Enable
Output selector for CCU80.OUTy0	Connected to inverted CC8yST1
Output selector for CCU80.OUTy1	Connected to CC8yST1
Dead time clock control	Time slice clock frequency, f_{tclk}
Dead time value	USER_DEAD_TIME_US*USER_PCLK_FREQ_MHZ (750 nsec)

Parameters	Settings for XMC1300/XMC1400 and XMC4400
Phase U Slice Period Match Interrupt Event	Enabled
Trap Interrupt Event	Enabled

2.5.1 7-segment SVM

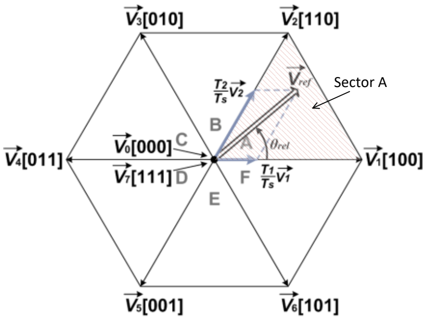


Figure 17 7-segment SVM

Using the voltage space vector in sector A as an example; the following equations are used to calculate PWM on-time of the SVM.

$$\vec{V}_{ref} = \frac{T_0}{T_S}\vec{V}_0 + \frac{T_1}{T_S}\vec{V}_1 + \frac{T_2}{T_S}\vec{V}_2$$

$$T_S = T_0 + T_1 + T_2$$

$$T_1 = \frac{\sqrt{3}|V_{ref}|T_S}{V_{DC}} sin\left(\frac{\pi}{3} - \theta_{rel}\right)$$

$$T_2 = \frac{\sqrt{3}|V_{ref}|T_S}{V_{DC}} sin(\theta_{rel})$$

$$T_0 = T_S - T_1 - T_2$$

Where:

T_S Sampling period

$\vec{V}_0$ Zero vector

$\vec{V}_1$ $\vec{V}_2$ Active vectors

T_0 Time of zero vector(s) is applied. The zero vector(s) is $\vec{V}_0[000]$,$\vec{V}_7[111]$ or both

T_1 Time of active vector $\vec{V}_1$ is applied within one sampling period

T_2 Time of active vector $\vec{V}_2$ is applied within one sampling period

T_{DC} Inverter DC link voltage

θ_{rel} Relative angle between Vref and V1 ($0 \leq \theta_{rel} \leq \frac{\pi}{3}$)

For example, in SVM sector A, the PWM on time for phase U is PWM period minus $T_0/2$, phase V is PWM period minus $(T_0/2 + T_1)$ and phase W is $T_0/2$.

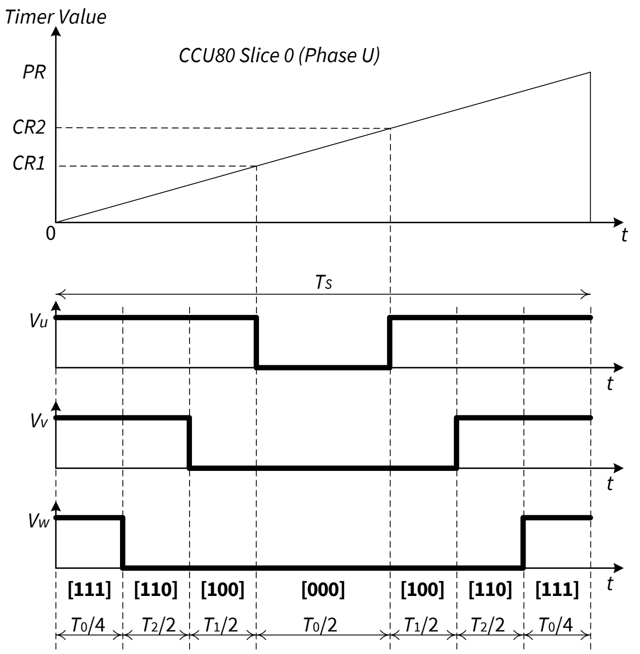


Figure 18 **7-segment SVM timing diagram in SVM sector A**

2.5.2 5-segment SVM

The 5-segment SVM uses the same equations as in 7-segment SVM to calculate the T0, T1, and T2 timing. In 5-segment SVM, the zero vector is only$\vec{V}_0[000]$. Unlike in 7-segment SVM, the zero vectors are $\vec{V}_0[000]$ and$\vec{V}_7[111]$.

For example, in SVM sector A, the PWM on time for phase U is PWM period minus T_0, phase V is PWM period minus $(T_0 + T_1)$ and phase W is zero.

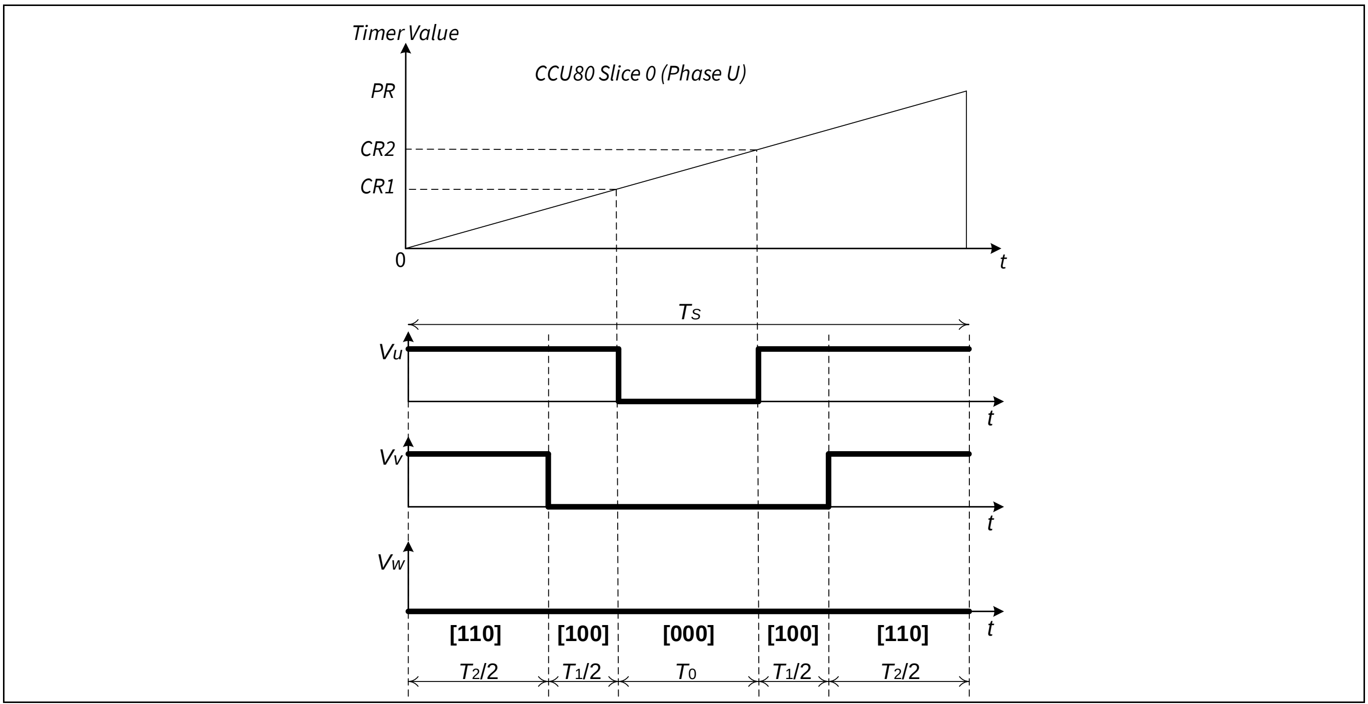


Figure 19 **5-segment SVM timing diagram in SVM sector A**

2.5.3 Pseudo Zero Vector (PZV)

In single-shunt current reconstruction, the current through one of the phase can be sensed across the shunt resistor during each active vector. However, under certain conditions, for example at sector crossovers or when the length of the vector is low, the duration of one or both active vectors ($T_1 < T_{min}$ or $T_2 < T_{min}$) is too small to guarantee reliable sampling of the phase currents. These conditions are shaded in the space vector diagram as shown in the **Figure 20**.

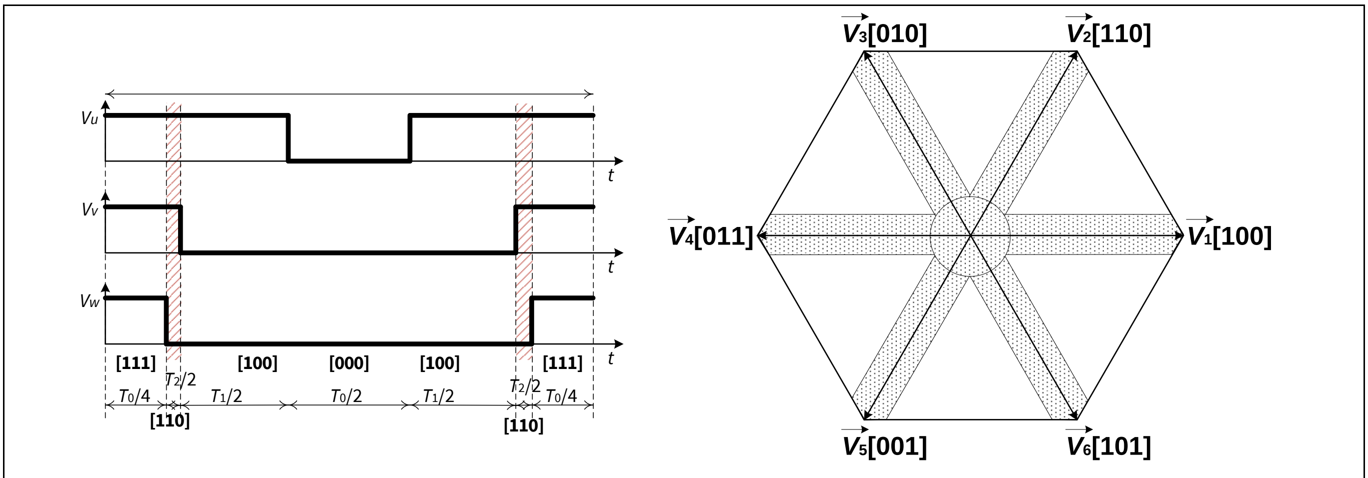
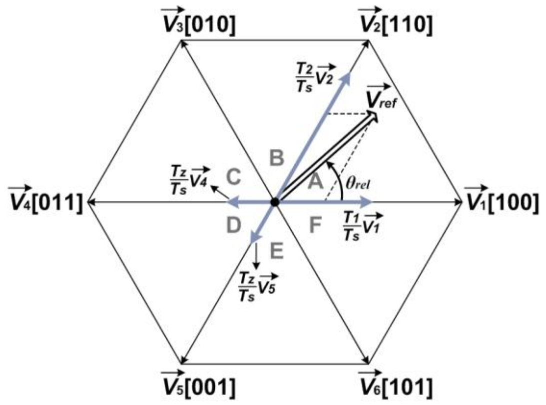


Figure 20 **7-segment SVM - single shunt sensing**

In order to resolve this, Pseudo Zero Vector (PZV) is used in these conditions for single-shunt current sensing. The Pseudo Zero Vector time T_Z is adjusted to ensure adequate ADC sampling time for the phase currents sensing.



$$T_1 = \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \sin\left(\frac{\pi}{3} - \theta_{rel}\right) + T_{\min} \qquad (T_Z = T_{\min})$$

$$T_2 = \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \sin(\theta_{rel}) + T_{\min}$$

$$T_1 + T_2 = \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \cdot \sin\left(\frac{\pi}{3} + \theta_{rel}\right) + 2T_{\min}$$

$$T_0 = T_s - T_1 - T_2 - 2T_{\min} = T_s - \frac{\sqrt{3}\cdot T_s \cdot |V_{ref}|}{V_{DC}} \cdot \sin\left(\frac{\pi}{3} + \theta_{rel}\right) - 4T_{\min}$$

Note: $T_{\min}$ is PWM dead time + driver delay + ADC sampling time

Figure 21 Pseudo Zero Vector

The figure above shows the equations to calculate the T_1 and T_2 timing.

Figure 22 shows the timing diagram and the SVM diagram of the Pseudo Zero Vector.

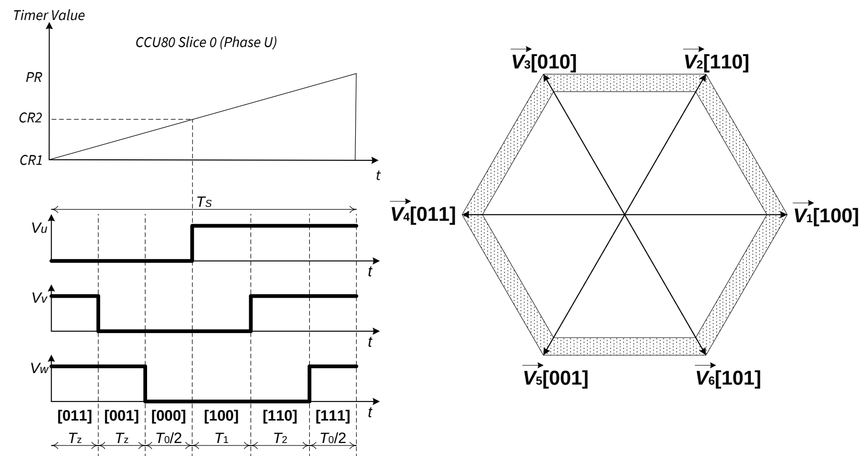


Figure 22 Pseudo Zero Vector timing diagram in sector A

2.5.4 4-segment SVM

This SVM pattern is also used in the single shunt current sensing technique. Pseudo Zero Vector is useful in certain conditions, at sector crossovers, or when the length of the vector is low. However, if PZV is used throughout, the motor might not be able to spin up to its maximum target speed due to the limitation in the voltage amplitude; the higher the T_Z value, the lower the motor speed that it can reach. This is the shaded area in the SVM diagram in **Figure 22**. To resolve this condition, 4-segment SVM is used.

In the PMSM_FOC software, a transition between PZV and 4-segment SVM PWM generation is implemented to resolve this issue. When T_1 and T_2 are greater than T_{min} and the motor speed is more than 75% of the maximum motor speed, 4-segment SVM is used. In this way, maximum target speed can be achieved.

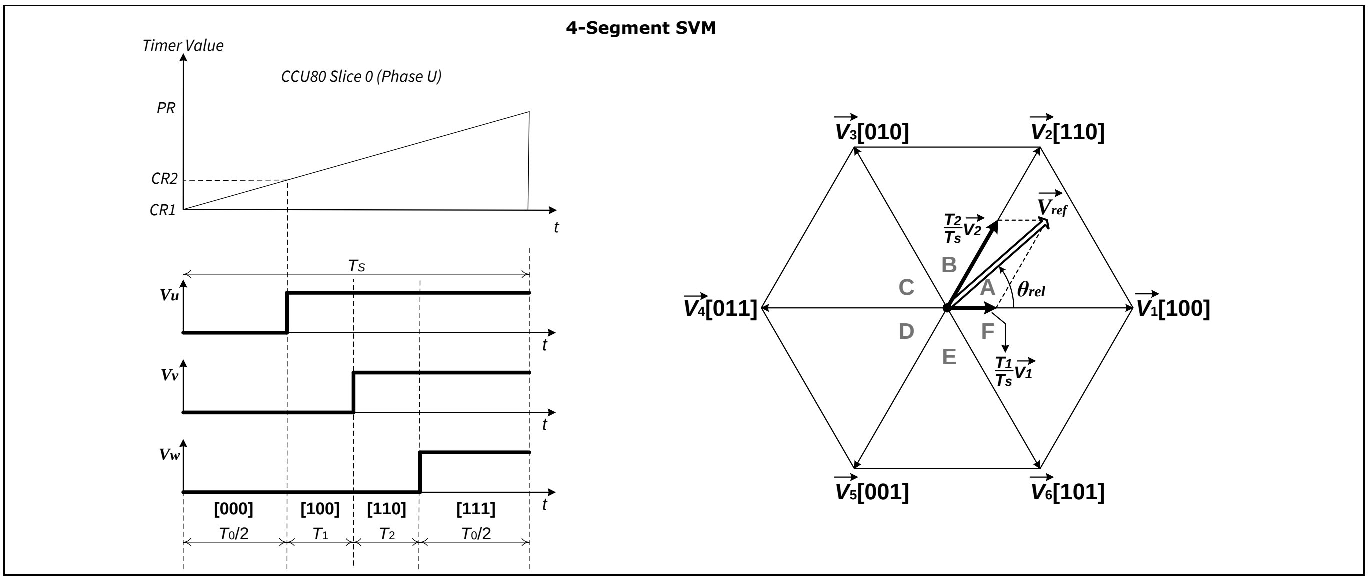


Figure 23 4-segment SVM

2.5.5 Over-modulation SVM

For sinusoidal commutation, V_{ref} has to be smaller than 86% of the maximum V_{ref}.

For non-sinusoidal commutation, the SVM can have a higher V_{ref} amplitude. This technique is referred to as over-modulation of SVM.

Figure 24 shows the area (shaded in red) where the over-modulation technique is used.

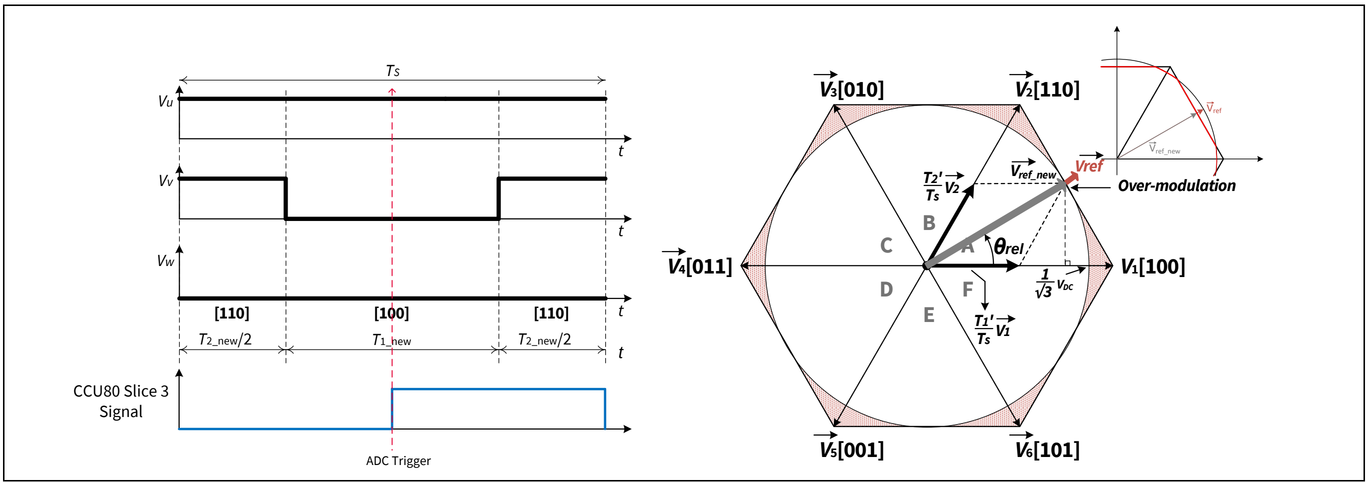


Figure 24 Over-modulation

In the PMSM FOC software, the over-modulation is implemented by reducing the amplitude of V_{ref} to the V_{ref_new}. When T_1 plus T_2 is more than PWM period, T_S, the vector is reduced to follow the edge of the hexagon. This is done by reducing the T_1 and T_2 timing proportionally.

T_1 and T_2 are re-calculated as:

$$T_{1\_new} = T_1 \times \frac{T_S}{T_1 + T_2}$$

$$T_{2\_new} = T_S - T_{1\_new}$$

In over-modulation, the time for zero vector is reduced to zero. The new timing diagram of SVM sector A is shown in the left diagram in Figure 24. From the diagram, it shows that only 2-phase currents, I_v and I_w, can be measured.

When the motor is running at high-speed, over-modulation is used to maximize DC bus utilization. The drawback of over-modulation is that the output voltage is not sinusoidal, and it contains high-order harmonics which cause acoustic noise.

2.6 DC link voltage

The DC link voltage is measured via the voltage divider on the power inverter board. The measured value is scaled to 2^12.

The voltage divider ratio value is defined in the pmsm_foc_invertercard_parameter.h (refer to **chapter 7.2.2**).

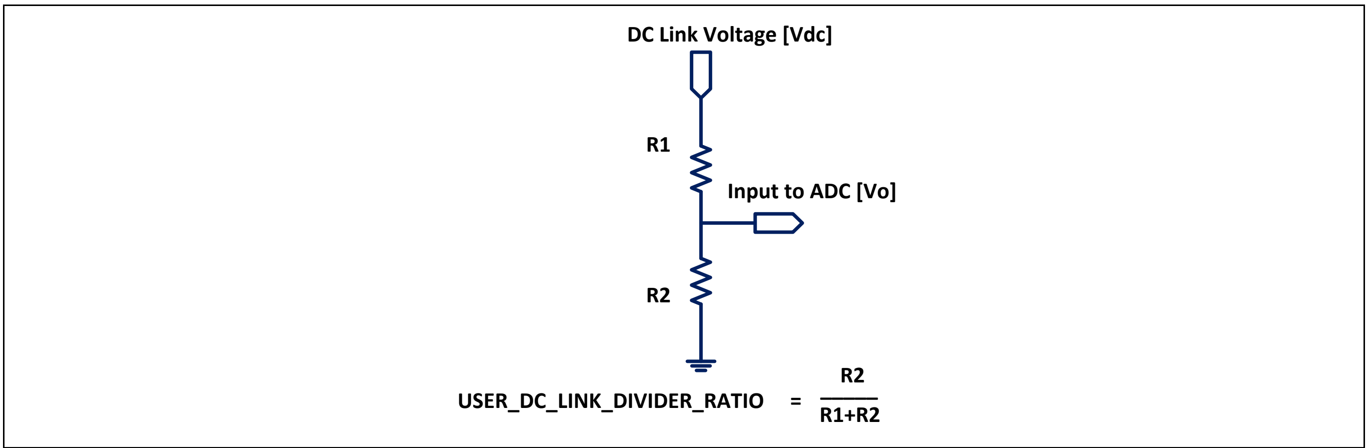


Figure 25 DC link voltage divider

2.7 Clarke transform

In this module the phase currents ($I_{I\_u}, I_{I\_v}, I_{I\_w}$) from the current sensing module, are transformed into currents I_Alpha and I_Beta on the 2-phase orthogonal reference frame.

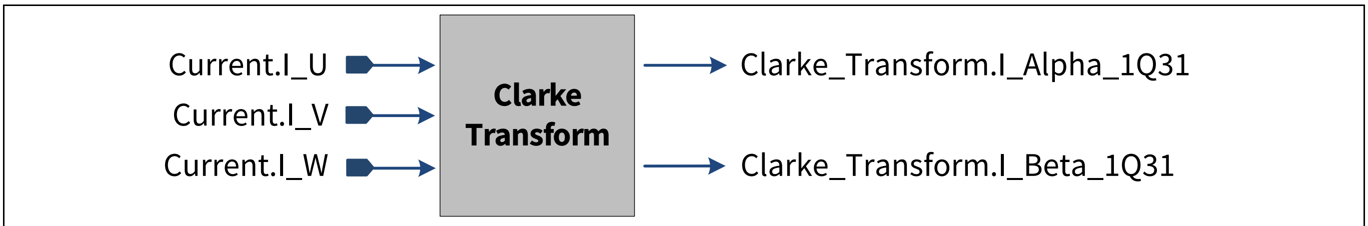


Figure 26 Clarke transform

Equations for Clarke transform:

$$I_\alpha = I_{I\_u}$$

$$I_\beta = \frac{1}{\sqrt{3}} \cdot I_{I\_u} + \frac{2}{\sqrt{3}} \cdot I_{I\_v} = \frac{1}{\sqrt{3}} \cdot \left(I_{I\_u} + 2 \cdot I_{I\_v}\right)$$

Where:

$$I_{I\_u} + I_{I\_v} + I_{I\_w} = 0$$

Scaling factor of the Current.I_U, I_V and I_W are based on the current scaling (see **chapter 2.10**). In XMC1300/XMC1400 the outputs of the Clarke Transform are shifted left by 14 bits (CORDIC_SHIFT) due to the code optimization for XMC CORDIC hardware module.

2.8 Park transform

In the Park transform, the currents I_Alpha and I_Beta are resolved to a rotating orthogonal frame with rotor angle.

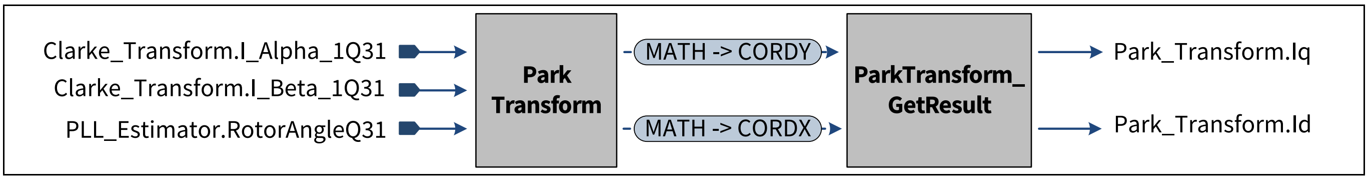


Figure 27 Park transform

In XMC1300/XMC1400, the Park transform is calculated by the CORDIC coprocessor.

$$I_q = \left(-I_\alpha \cdot \sin(RotorAngle) + I_\beta \cdot \cos(RotorAngle)\right) * CORDIC\_GAIN$$

$$I_d = \left(I_\alpha \cdot \cos(RotorAngle) + I_\beta \cdot \sin(RotorAngle)\right) * CORDIC\_GAIN$$

Note:: *CORDIC_GAIN = K/MPS*

The input PLL_Estimator.RotorAngleQ31 is shifted left by 14 bits due to code optimized for XMC CORDIC hardware module (refer to XMC1300 or XMC1400 reference manual [1]).

Scaling factor of I_q and I_d are based on the current scaling (**see chapter 2.10**).

Table 7 XMC1000 CORDIC settings for Park transform

Parameters	Settings
CORDIC Control Mode	Circular Rotating Mode
K	≈ 1.646760258121
Magnitude Prescaler, MPS	2
CORDX	Clarke_Transform.I_Beta_1Q31
CORDY	Clarke_Transform.I_Alpha_1Q31
CORDZ	PLL_Estimator.RotorAngleQ31

In XMC4400, the Park transform is calculated by the FPU using fpu_park_q31() function implemented in fpu_math2 library.

2.9 Protection

The PMSM FOC motor control software supports the following protection schemes:

- CCU80 CTrap Function
- Over-Current Protection
- Over/Under Voltage Protection

CCU80 CTrap Function

Trap function of the CCU8 module provides hardware overload condition protection. The CTrap input pin is connected to the fault pin of the gate driver. Once the gate driver detects a fault, the CTrap pin is set to active state and the PWM outputs are set to PASSIVE level. The CTrap interrupt is triggered. In the Interrupt Service Routine, the gate driver is disabled and the motor state machine is set to TRAP_PROTECTION state.

Over-Current Protection (only in XMC1300/XMC1400)

The average current flow through DC link shunt resistor is sampled every cycle of PWM. This value is read to detect an over-current condition. Once this condition occurs, the reference motor speed is scaled down by a factor till the current is within the limit (USER_IDC_MAXCURRENT_A) defined in the user configuration file.

The variable FOCInput.overcurrent_factor, is used to update motor speed using this equation:

$$FOCInput.Ref\_Speed = \frac{Motor.Ref_{Speed} * \ FOCInput.overcurrent\_factor}{4096}$$

FOCInput.overcurrent_factor is reduced if the average DC link current is above the limit.

The nominal value of the FOCInput.overcurrent_factor is 4096. This value is increased back to nominal when the average DC link current is within the limit. In XMC4400, this feature is not supported.

Over/Under Voltage Protection

The DC link voltage is continuously converted. If the DC link voltage is less than or greater than specific user limits, an interrupt is generated and the function pmsm_foc_over_under_voltage_isr () is called. The gate driver is disabled to stop driving the motor. The CCU8 timer is still running and the motor state machine is changed to DC_LINK_UNDER_VOLTAGE or DC_LINK_OVER_VOLTAGE state.

The voltage protection check is not done during motor ramping and open loop condition.

2.10 Scaling

PMSM FOC software uses integers to represent real-world floating-point variables, such as angle, current, and voltage. To provide the best resolution, the software represents the Physical Value depending on the Target Value.

For example, the phase current is represented by 0 - 100% of the Target Value, where the Target Value is the maximum current that can be measured by the current sensing circuit.

The following equation shows the conversion of Physical Value to the Norm Value represented in the software.

$$Norm\ Value = \frac{Physical\ Value \ * \ 2^N}{Target\ Value}$$

Table 8 Scaling used in PMSM FOC software

Parameter	Scaling		Range	
	Target value [Unit]	N	[%]	[hex]
SVM Amplitude (V_{ref})	$N_{\_Vref\_(SVM)}$ [Volts]	15	0% to 100%	0x0 to 0x7FFF
Current I_U, I_V, I_W, I_q, I_d	$N_{\_I\_(\alpha,\beta)}$ [A]	15	-100% to 100%	0x8001 to 0x7FFF
Angle of rotor position, Angle of space vector	360° [degree]	16 + USER_RES_INC	0 to 360°	0x0 to 0xFFFF (for 16+ USER_RES_INC bit)
Speed	N_{max_speed} [degree/second]	$\log_2(max\_speed\_integer)$	0% to 100%	0x0 to max_speed_interger

In the following sections, the calculations of the Target Values are described.

Scaling for SVM voltage

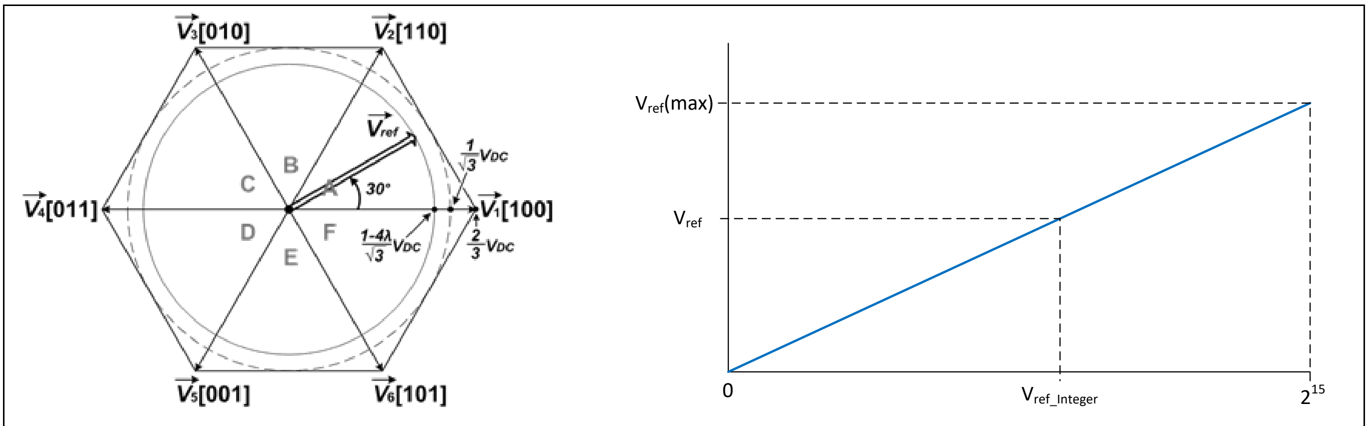


Figure 28 SVM voltage scaling

$$V_{ref} = \frac{N_{\_Vref\_SVM}}{2^{15}} \cdot V_{ref\_Integer}$$

$N_{\_Vref\_SVM}$ is the maximum reference voltage of SVM

$$N_{\_Vref\_SVM} = \frac{1-4\lambda}{\sqrt{3}} V_{DC}$$

$\lambda = T_Z/T_S$ is the pseudo zero vector ratio, λ = 0 for standard SVM

V_{DC} is inverter DC link voltage, USER_VDC_LINK_V

Example:

USER_VDC_LINK_V is 24.0f, λ = 0

⇨ $N_{\_Vref\_(SMV)}$ = 13.86 V

To represent V_{ref} = 0.5 V, the software integer, $V_{ref\_integer}$ is 1182.

Scaling for phase current

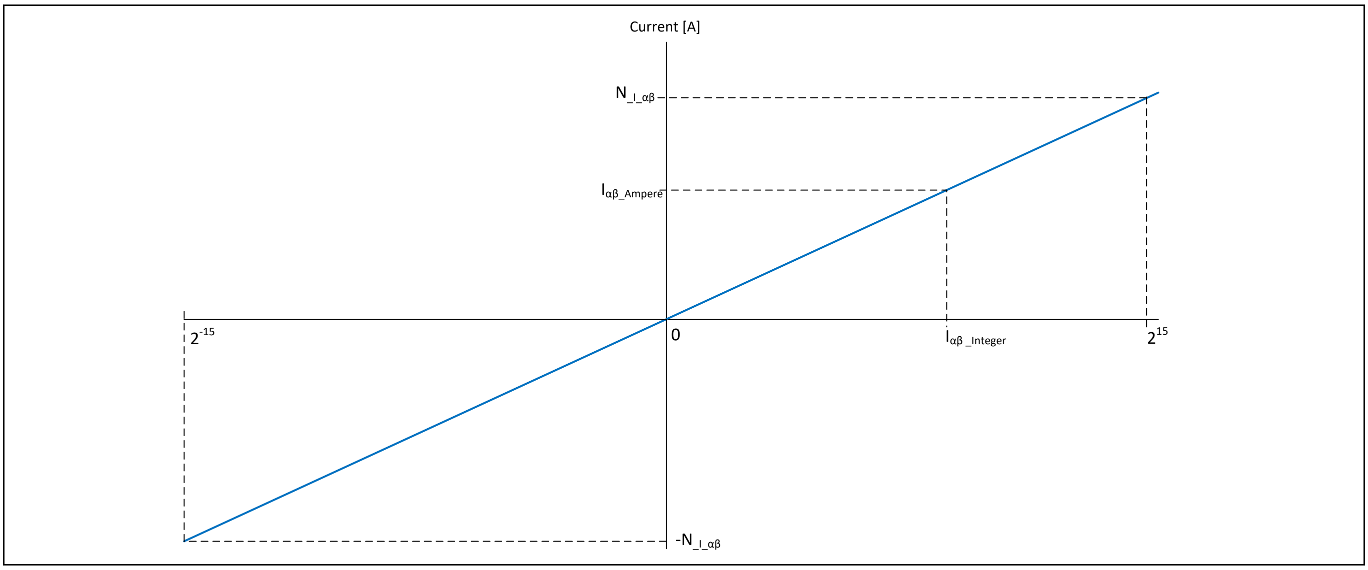


Figure 29 Phase current scaling

The Target Value of the current is the maximum current that can be measured by the current sensing circuit:

$$N_{I\_(\alpha,\beta)} = \frac{V_{AREF}/2}{R_{shunt} \times G_{OpAmp}}$$

If internal ADC gain is used, G_{OpAmp} is replaced with the ADC gain factor setting.

Scaling for angle and speed

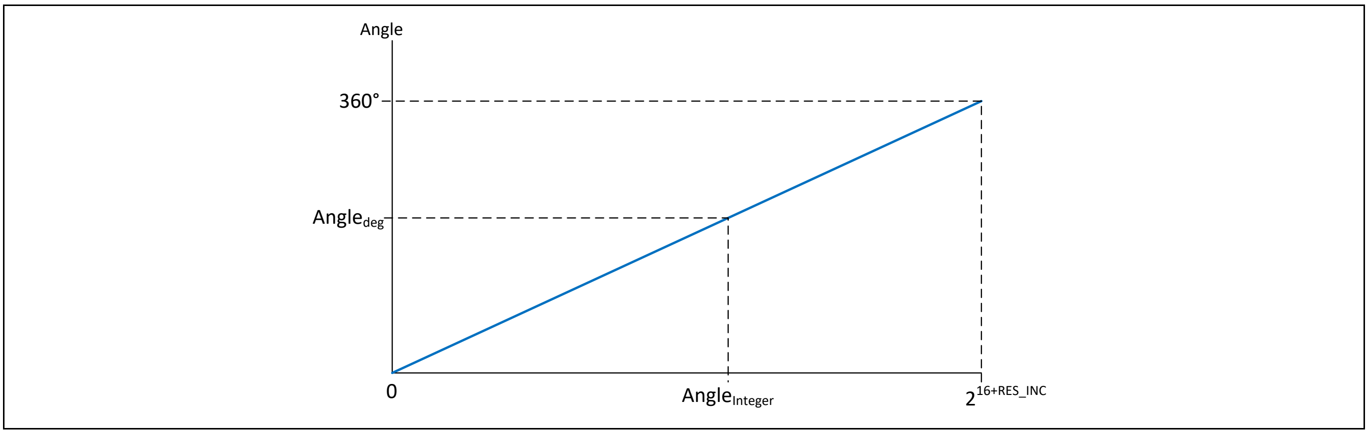


Figure 30 Angle scaling

In the PMSM_FOC software, it uses 16-bit (or 16 + USER_RES_INC bits where USER_RES_INC: 0~8) integers to represent angles of 0° to 360°. The angle scaling equation is:

$$Angle_{deg} = \frac{360°}{2^{16+RES\_INC}} \cdot Angle_{integer}$$

Following the angle scaling, the speed scaling is:

$$\omega_{degree/second} = \frac{N_{max\_speed}}{2^{N}} \cdot \omega_{integer}$$

Where:

$\omega_{integer}$ is the angle increase/decrease every CCU8 PWM cycle (i.e. integer speed)

Target Value for speed is:

$$N_{max\_speed} = \frac{USER\_SPEED\_HIGH\_LIMIT\_RPM * USER\_MOTOR\_POLE\_PAIR}{60} * 360°\ degree/seconds$$

$$max\_speed\_integer = \frac{N_{max\_speed} * 2^{16+USER\_RES\_INC}}{60 * f_{CCU8\_PWM}}$$

$$N = \log_2(max\_speed\_integer)$$

Note: If speed control is not done in every CCU8 PWM cycle, the scaling indicated above needs to be adjusted accordingly based on the speed control rate.

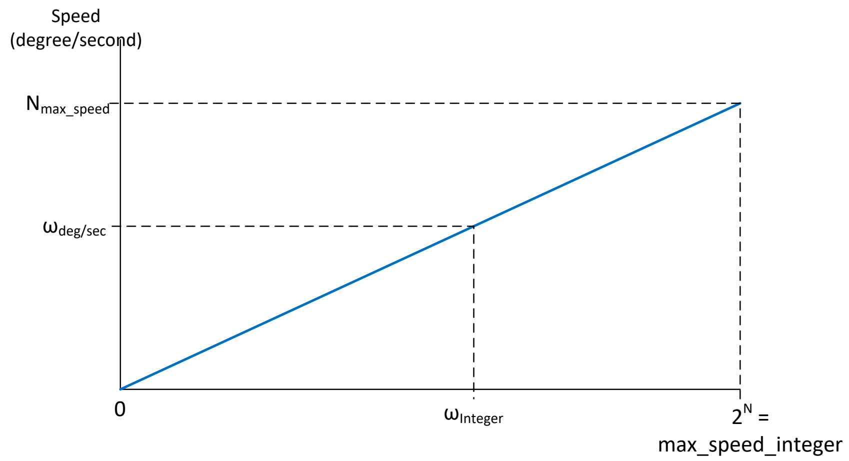


Figure 31 Speed scaling

Example:

- Motor maximum speed, USER_SPEED_HIGH_LIMIT_RPM = 10,000 rpm
- Motor pole pairs, USER_MOTOR_POLE_PAIR = 4
- CCU8 PWM Frequency = 25 KHz
- USER_RES_INC = 3

⇨ $N_{max\_speed} = \frac{(10000\ rpm*4)*360°}{60} = 240{,}000\ degree/second$

⇨ $max\_speed\_integer = \frac{240{,}000 * 2^{16+3}}{360 * 25{,}000} = 13{,}981$

⇨ $N = \log_2(max\_speed\_integer) = \log_2(13{,}981) = 13.77$

To represent speed 2,000 rpm which is 48,000 (degree/second), the software integer, $\omega_{integer}$= 2,796

2.11 Determination of flux and torque current PI gains

To calculate the initial values of the PI gains of the torque and flux current control, it is necessary to know the electrical parameters of the motor. For the SPMSM motor, the torque inductance and the flux inductance are considered equal.

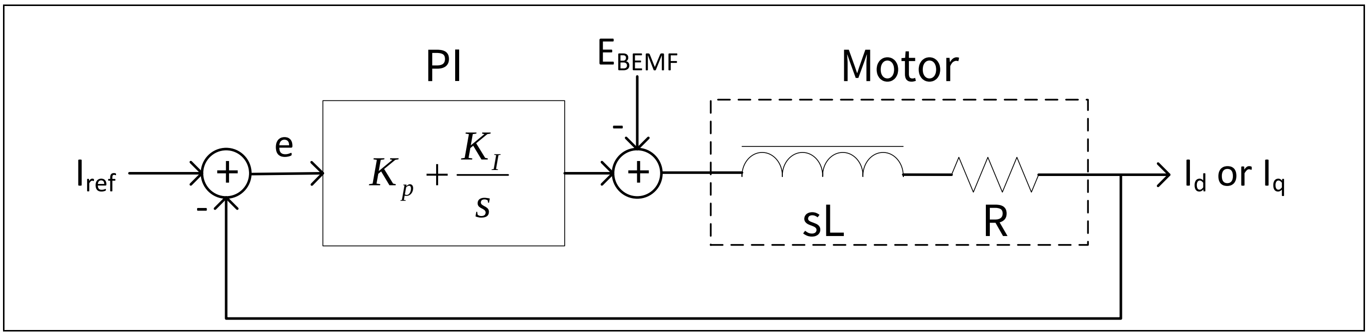


Figure 32 Torque / Flux current control loop

The calculation of the PI gains is made by using the pole-zero cancellation technique as illustrated. By having $K_P/K_I = L/R$, the controller zero will cancel the motor pole. With this the transfer function of the control loop is a first order LPF with time constant, T_c. In addition, the proportional gain calculation is based on motor inductance and the integral gain is on the motor resistance.

At constant motor speed the Back-EMF of the motor is near constant. Therefore it is negligible in the frequency domain. The figure shows the simplified diagram after pole-zero cancellation.

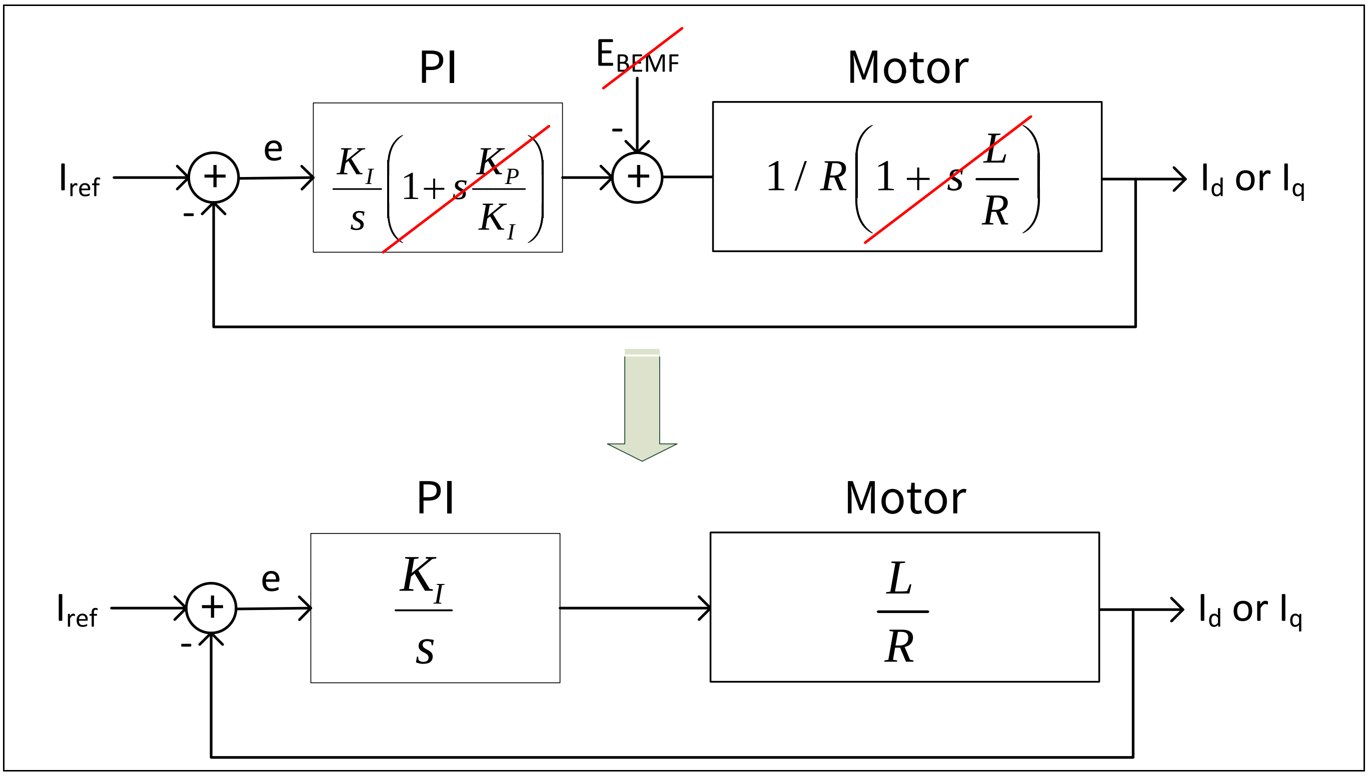


Figure 33 Simplified current control loop due to pole-zero cancellation

As $K_P/L = K_I/R = \omega_c$, the PI controller gains are:

Proportional Gain $K_P = \omega_c L$

Integral Gain $K_I = \omega_c R$

Where:

- ω_c is the cutoff frequency of the first order LPF.
- L is the motor inductance
- R is the motor resistance.

In the digital controller implementation, the integral part is a digital accumulator. Therefore the K_I gain has to include a scaling factor for the sampling time T_S, which is the PWM frequency.

Revised formula:

Proportional Gain $K_P = \omega_c L \times A$

Integral Gain $K_I = \omega_c R \times T_S = RT_S K_P/L$

Where:

- A is the XMC hardware optimize scaling factor.

Based on the past experience, set the cutoff frequency to three times of the maximum electrical motor speed to obtain a good tradeoff between dynamic response and sensitivity to the measurement noise.

3 Current sensing and calculation

This module is used to measure motor phase currents using the VADC peripheral.

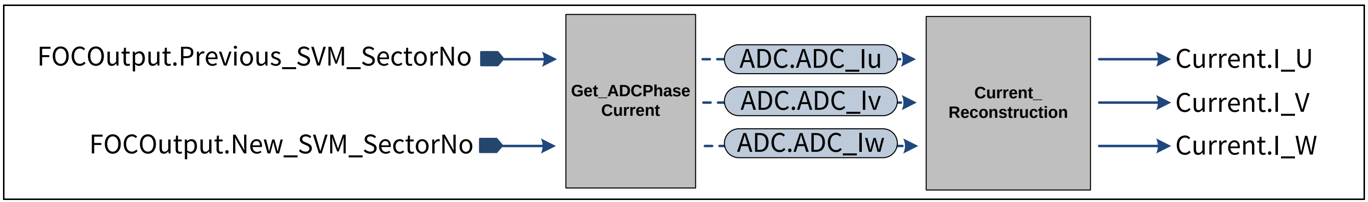


Figure 34 Current sensing and calculation functions

Two techniques to measure phase currents

- Single shunt current sensing (only in XMC1300/XMC1400)
- Three shunt current sensing

You can select the option of the current sensing technique in the user configuration file.

The phase currents measurements are synchronized with the PWM SVM pattern generation. The fourth slice of the CCU80 module, slice 3 (slice 2 in XMC4400), is used to trigger the ADC conversions. Initial settings of the CCU80 and VADC modules for different current sensing techniques are listed in their respective sub-chapters, **chapter 3.1** and **chapter 3.2**.

The figure below shows the timing diagram of the three shunt current sensing technique using synchronous ADC conversion.

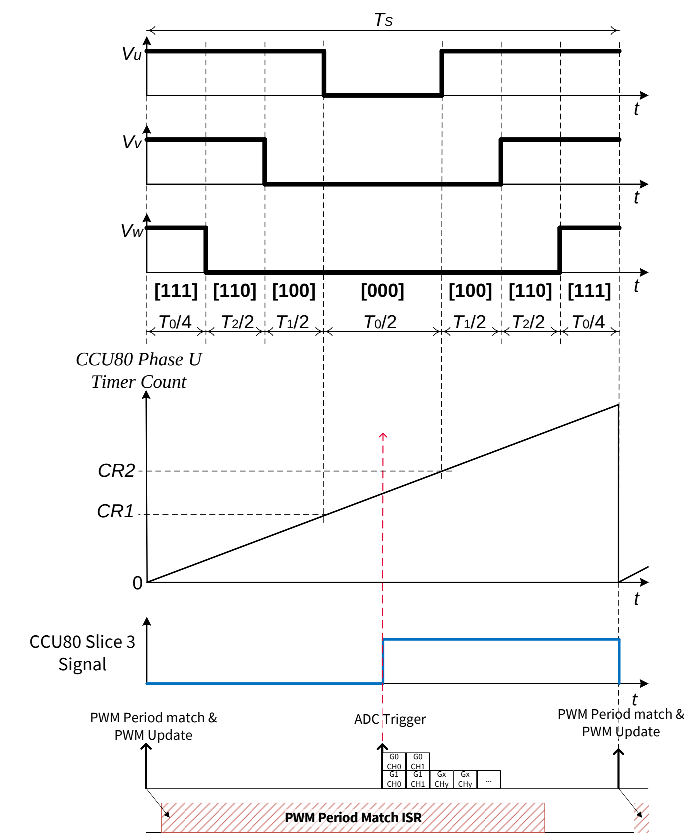


Figure 35 Three shunt current sensing timing diagram using synchronous conversion ADC

Internal ADC Gain Feature

In default applications an OP-amp is used to amplify the voltage drop above the current shunt to combine low power losses and high ADC accuracy. This method is supported by the XMC PMSM FOC motor control software.

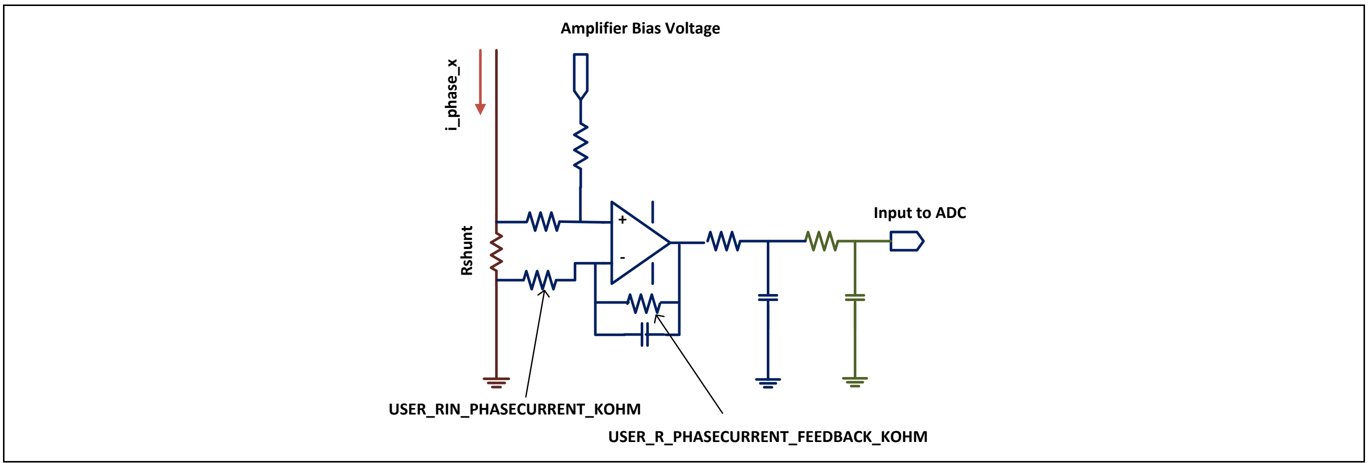
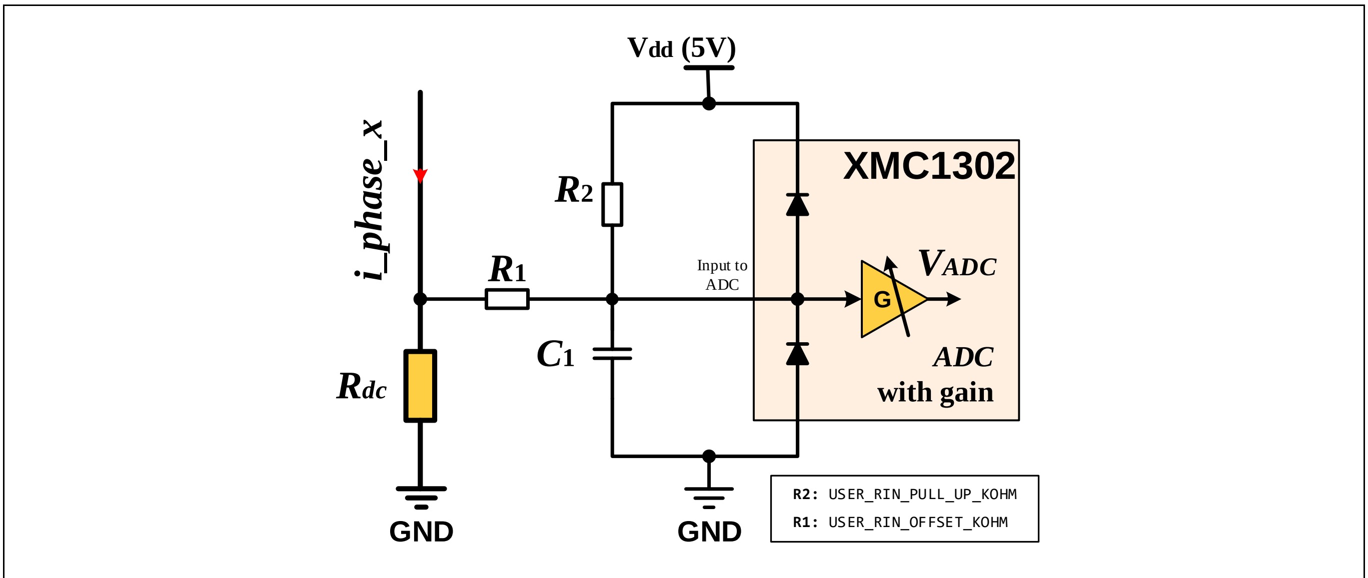


Figure 36 External phase current amplifier

Additionally, the XMC supports an analog gain stage for the ADC (VADC). With this feature an external fast op-amp is not required for the phase current signals. This leads to cost saving in the BOM of the PCB boards.



$$V_{ADC} \approx \frac{GR_1}{R_1 + R_2} V_{dd} + \frac{R_2}{R_1 + R_2} G \cdot R_{dc} \cdot I_{dc}$$

Figure 37 Phase current amplifier with On-chip gain

3.1 Single shunt current sensing (only in XMC1300/XMC1400)

The single shunt current measurement technique measures the power supply current and, with knowledge of the switching states, recreates the three phase current of the motors.

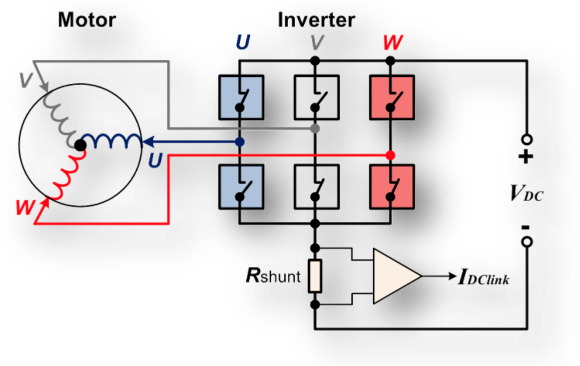


Figure 38 Single shunt sensing technique

The Pseudo Zero Vector (PZV) SVM is used to ensure enough time is given for single shunt current sensing.

Figure 39 shows the direction of the voltage space vector and what current can be measured in that state. The CCU80 slice 3 is used as a timer to automatically trigger the ADC conversion at specific time as shown in Figure 39. The ADC conversions are triggered at both the rising and falling edges of the CCU80 slice 3 signal (slice 2 in XMC4400). When 4–segment SVM is used, the ADC conversion trigger points are also changed, refer to Figure 40.

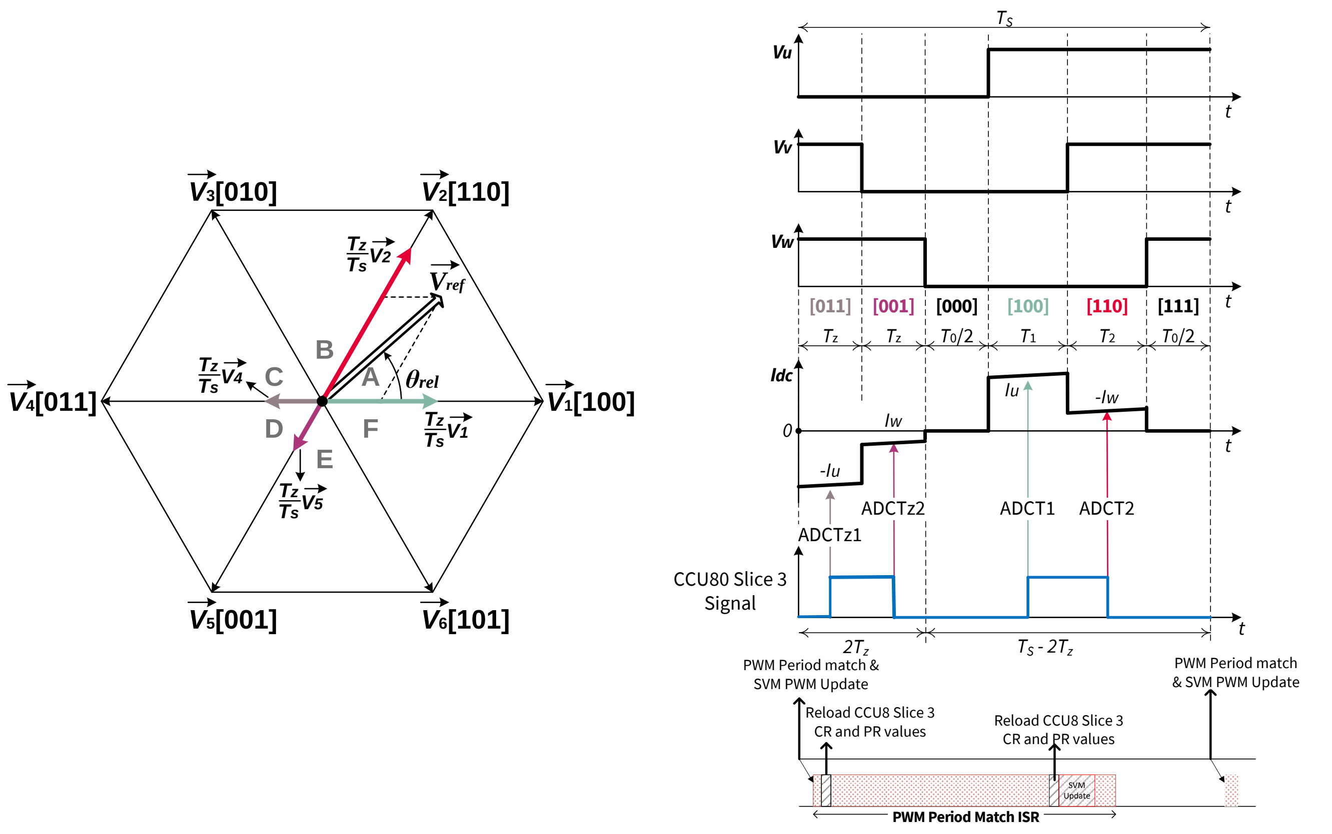


Figure 39 Single shunt – 3-phase current sensing in sector A

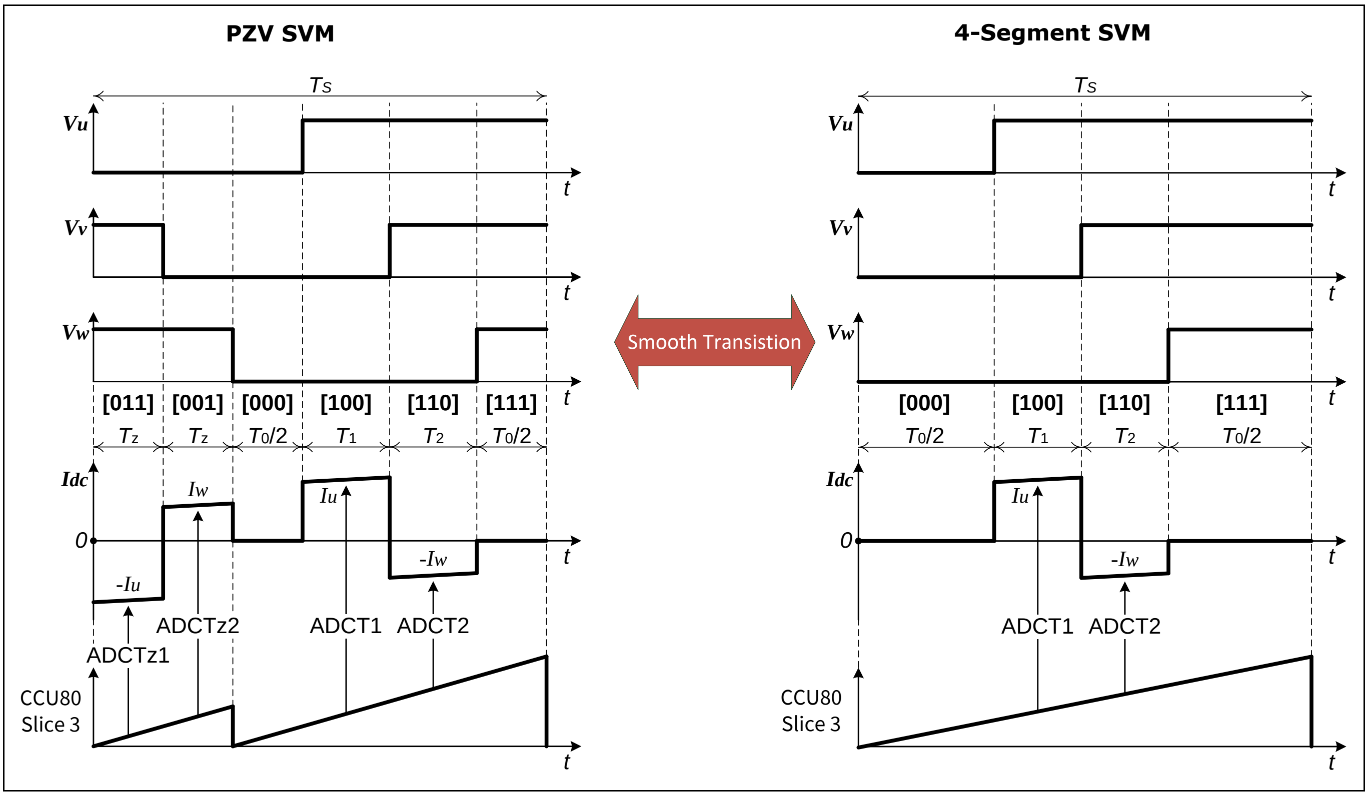


Figure 40 Smooth transition from PZV to 4-segment SVM in sector A

The VADC source interrupt event is enabled and its service routine performs the following tasks:

- Read the results of the ADC conversion
- Generate the phase current values and scale the values to 2^{15}

The measured 12-bit current value is scaled to 1Q15 format.

In PZV:

- Current -> $I\_U = (I_{ADCT1} - I_{ADCTz1}) * 2^3$
- Current -> $I\_W = (I_{ADCT2} - I_{ADCTz2}) * 2^3$

In 4-segment SVM:

- Current -> $I\_U = \left(I_{ADCT1} - I_{ADC\_Bias}\right) * 2^4$
- Current -> $I\_W = \left(I_{ADCT2} - I_{ADC\_Bias}\right) * 2^4$

The two tables below show the initial settings of the VADC and CCU80 slice 3 for single shunt current sensing technique.

Table 9 VADC initial settings for single shunt

Parameters	Settings
Request Source for Single Shunt	Queue
Request Source for other Channels	Background Scan
FIFO for Single Shunt	2-Stage Buffer
Source Interrupt	Enabled
ADC Conversion Trigger Signal	CCU80.ST3A (through gating select input)
ADC Conversion Trigger Edge	Both Rising and Falling Edges

Table 10 CCU80 slice 3 initial setting for single shunt

Parameters	Settings
Timer Counting Mode	Edged Aligned
Single Shot Mode	Enabled
Period Register	$2 * T_Z$
Compare Register Channel 1, CR1	$T_Z * 0.85$
Compare Register Channel 2, CR2	$T_Z + T_Z * 0.85$

3.2 Three shunt current sensing

The three shunt current measurement technique is more robust compared with single shunt sensing. Using this technique in XMC1300/XMC1400, we can select two out of the three phase currents for the current reconstruction calculation. In the XMC4400, we use all the three phases.

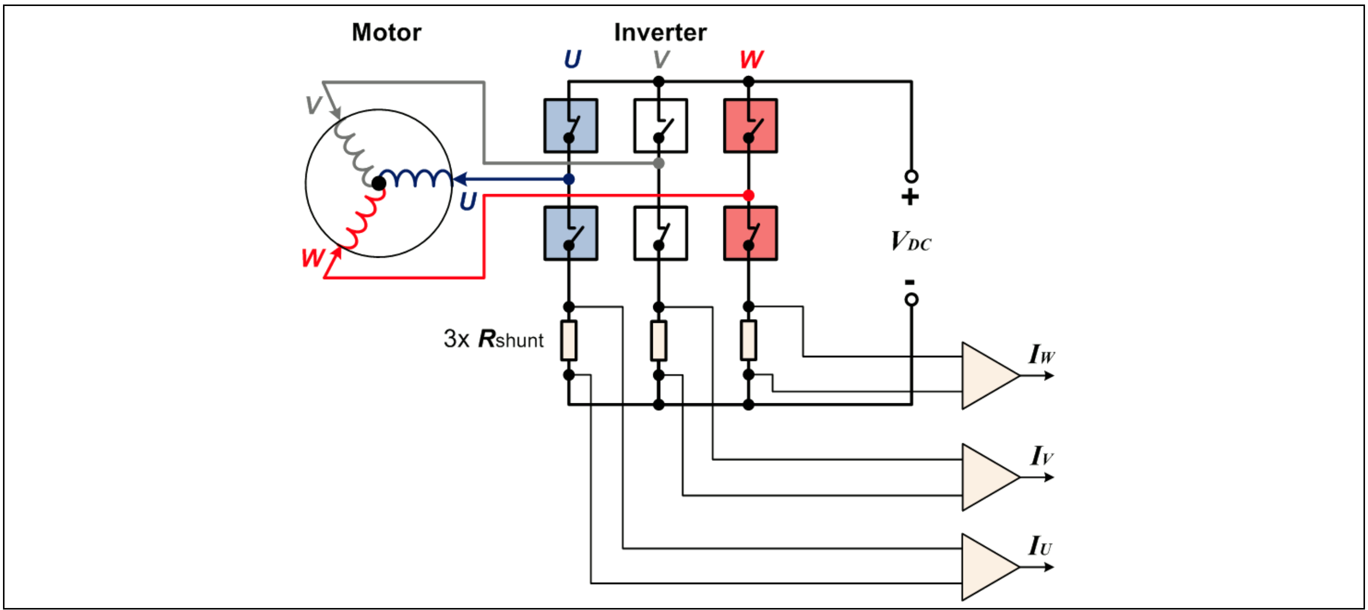


Figure 41 Three shunt sensing technique

For three shunt current sensing, the ADC conversion trigger is set at half of the PWM cycle where all the low-side switches are on, refer to Figure 42. The current will always flow through the shunt resistor when the low-side switch is on and high-side switch is off.

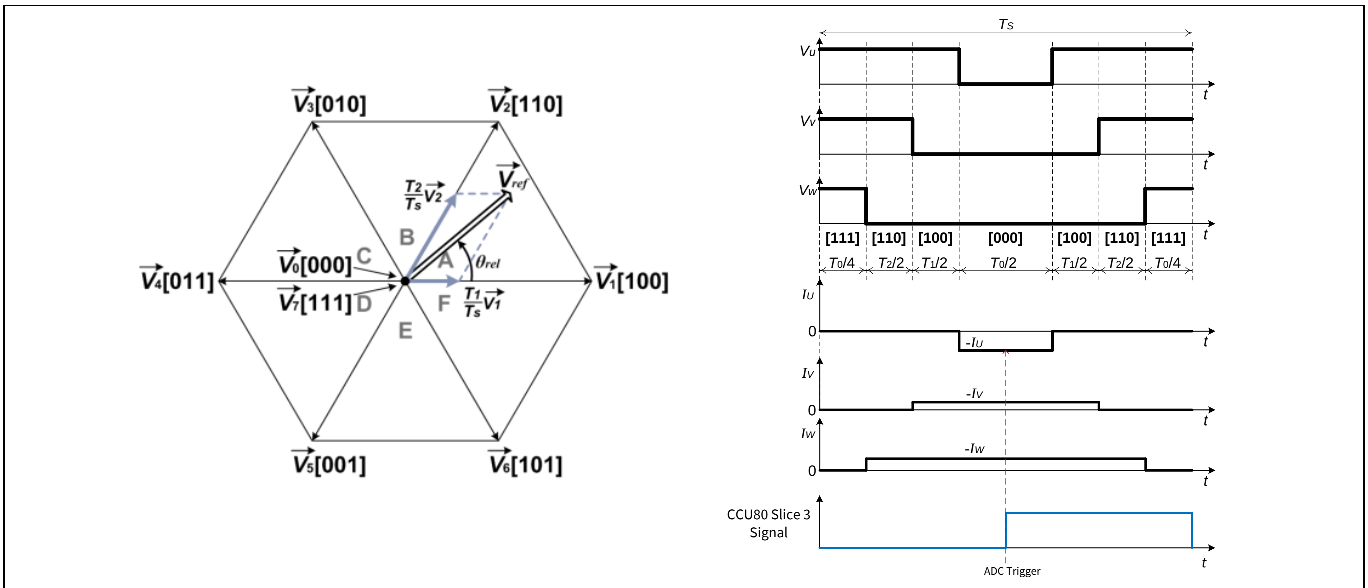


Figure 42 Three shunt, 3-phase current sensing in sector A

In the current calculation function, the measured 12-bit current value is scaled to 1Q15 format.

- Current -> $I\_U = (ADC\_Bias\_I_u - I_{ADC\_Iu}) * 2^3$
- Current -> $I\_V = (ADC\_Bias\_I_v - I_{ADC\_Iv}) * 2^3$
- Current -> $I\_W = (ADC\_Bias\_I_w - I_{ADC\_Iw}) * 2^3$

The initial settings of the VADC module and CCU80 slice 3 are detailed in the following tables.

Table 11 VADC initial settings for three shunt

Parameters	Settings
Request Source for Three Shunt	Queue
Request Source for Other Channels	Background Scan
FIFO	Disabled
Source Interrupt	Disabled
ADC Conversion Trigger Signal	CCU80.ST3A (through gating select input)
ADC Conversion Trigger Edge	Rising Edge

Table 12 CCU80 slice 3 initial setting for three shunt

Parameters	Settings
Timer Counting Mode	Edged Aligned
Single Shot Mode	Disabled
Period Register	Same as Period Register value for 3-phase PWM
Compare Register Channel 1	Half of Period Register value
Compare Register Channel 2	Compare Register Channel 1 value + 1

3.2.1 Asynchronous theory (only in XMC1300/XMC1400)

The term asynchronous conversion is related to the independency of the Groups. This mode is an easy implementation and the benefit is a free Group 1, no reload of the ADC, and easy to understand.

In the default configuration all three ADC inputs are sampled one after the other, and all three currents are measured one after each other. This mode does not require a reload of the ADC and is especially suitable if three ADCs are available (Not true for XMC1000). You can manually distribute the channels to the Groups. The main drawback is that the time to measure is up to double the time of an advanced implementation. Due to the required long current measurement window it is recommended to use this implementation only for demonstration purposes. It’s not implemented in the XMC4400 family.

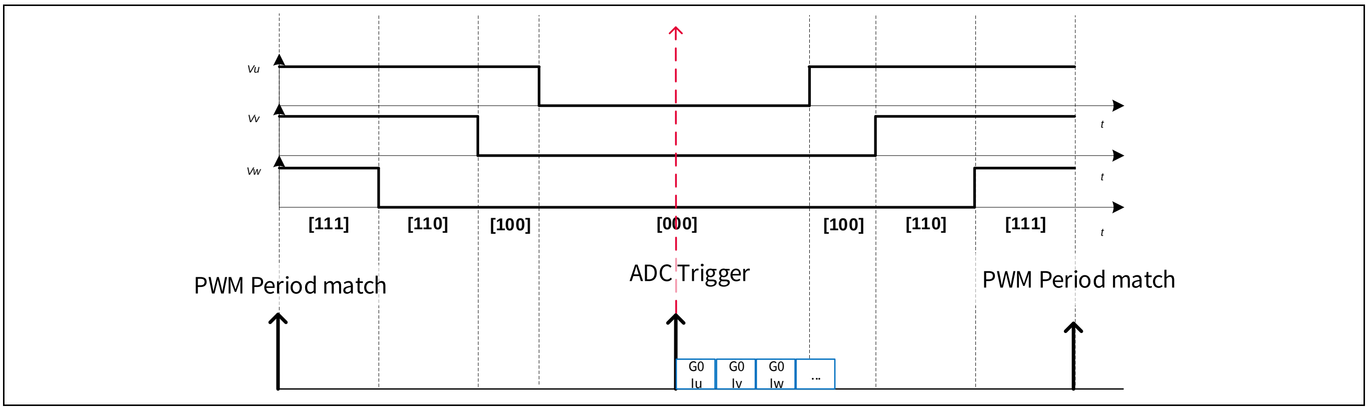


Figure 43 Asynchronous conversion sequence

3.2.2 Synchronous theory

In the XMC1300 and XMC1400 families, this implementation uses the three following hardware features:

The first feature allows synchronized sampling and sequential conversion of two shunt currents. This improves the accuracy and reduces the minimum measurement window. Both VADC Sample and Hold units are used for this feature to measure two currents at the same time (for example phase U and V). This method is not impacted if another measurement runs in the background. This gives rise to the implementation name ‘synchronous conversion’.

The second hardware feature improves the measurement for large amplitudes. In three phase leg shunt current measurement the current is measured in the middle where all high-side switches are off. This measurement window decreases with rising amplitudes, general higher torque. Which phase has a small measurement window depends on the sector (see **Figure 44**).

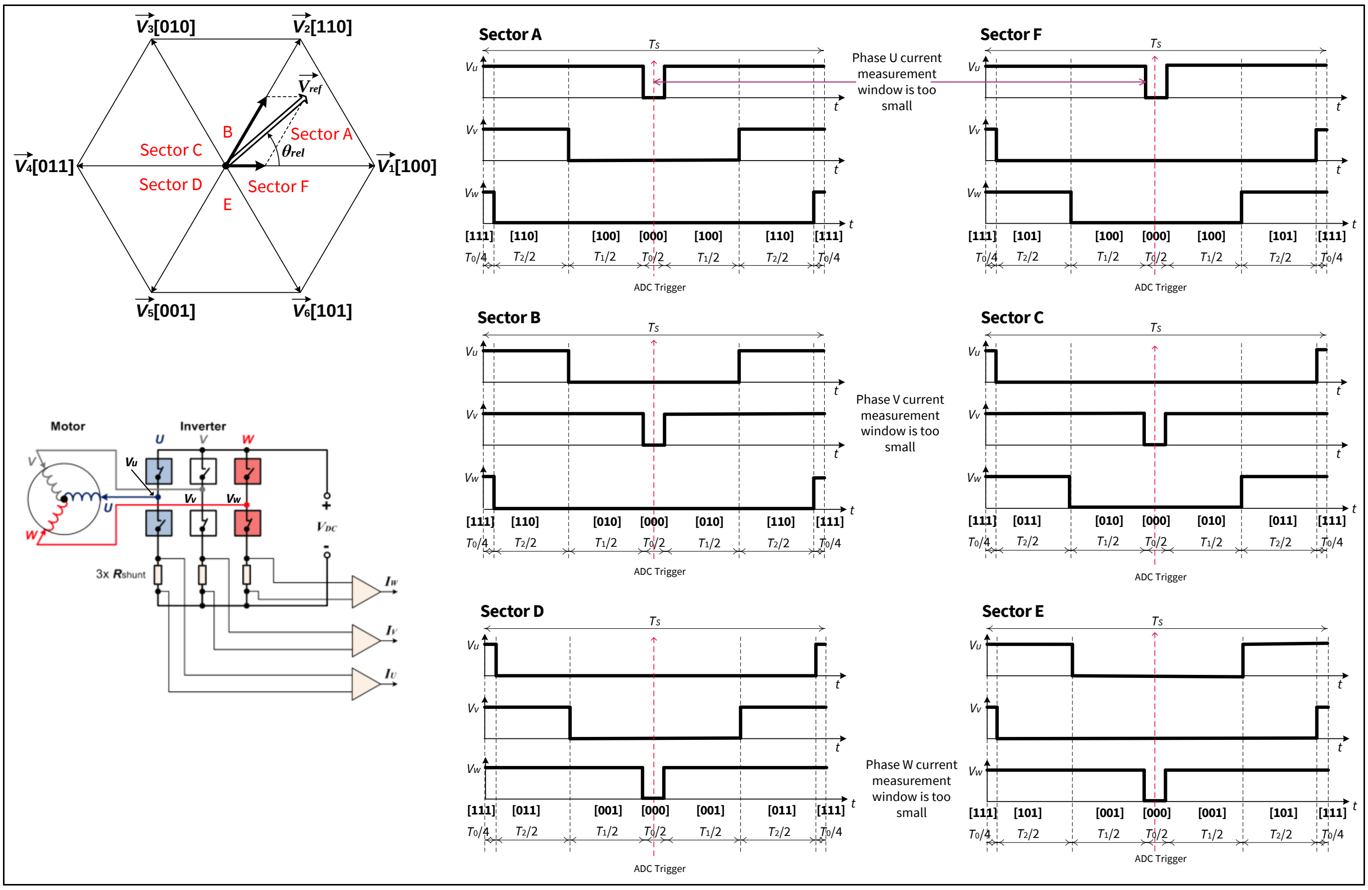


Figure 44 SVM sectors at high motor torque

The software changes the sequence of measurement depending on the sector. Therefore, it can measure the two non-critical phase currents. For example, for sectors A and F it can assign I_V and I_W ADC channels.

The following table shows the synchronous measured phases per sector.

Table 13 Phase measurement per SVM sectors

SVM sectors	Shunt current measured
Sector A and F	Phase W
	Phase V
Sector B and C	Phase U
	Phase W
Sector D and E	Phase V
	Phase U

The second hardware feature is the alias feature of the ADC, which allows for fast changing of the sequence, so even large amplitudes can be measured.

Consequently, the software discards the measurement of the third phase if the measurement window is smaller than the minimum measurement time T_{min}. It is then switching from 3 leg shunt measurement to 2 leg shunt measurement. In three areas the minimum measurement window fall below T_{min} at two phases. The Figure 45 shows which area can be measured with 3 or 2 shunt measurement.

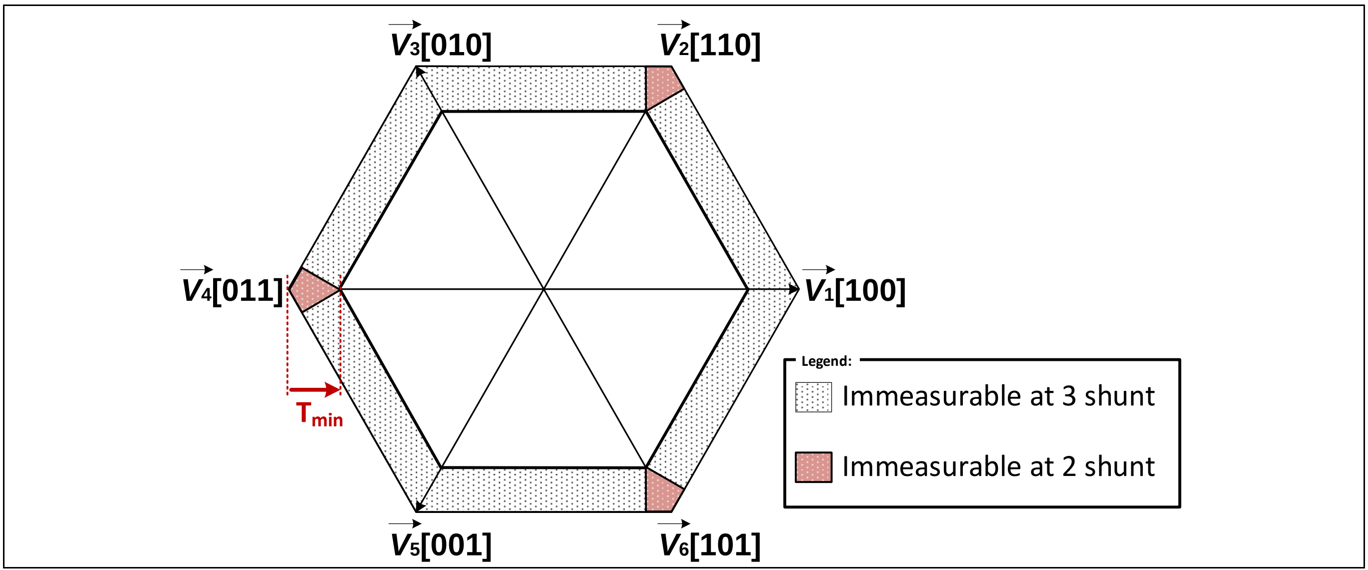


Figure 45 SVM 2/3 leg shunt immeasurable areas

The switching of the parallel sampled phases requires that all three inputs (I_U, I_V, I_W) are available for both groups (G0 and G1). Normally this would double the pin consumption. The third hardware feature of the XMC1300 and XMC1400 family avoid this doubling by overlapping group channels. Up to four pins are accessible from both groups.

In the XMC4400 family, four VADCs are present, so all phases can be measured in a synchronized way. Each phase is connected to a different ADC, and each channel is aliased to channel 0.

3.2.3 Synchronous Implementation

Using the alias feature in the ADC module, we can assign different ADC input channels to be converted in parallel. In XMC1300/XMC1400, we can measure the two most non-critical phase currents for all the SVM sectors. For sectors A and F, we assign I_V and I_W ADC channels. Whereas in the XMC4400 family, we can measure all three phases unconditionally.

The following table shows the synchronous measured phases per sector for XMC1300/XMC1400.

Table 14 Aliasing settings for SVM sectors for XMC1300/XMC1400

SVM sectors	Shunt current measured	Alias channels for CH0
Sector A and F	Phase W (Pin P2.9)	Group 0 Channel 2
	Phase V (Pin P2.10)	Group 1 Channel 2
Sector B and C	Phase U (Pin P2.11)	Group 0 Channel 4
	Phase W (Pin P2.9)	Group 1 Channel 4
Sector D and E	Phase V (Pin P2.10)	Group 0 Channel 3
	Phase U (Pin P2.11)	Group 1 Channel 3

The implementation for all sectors are shown in the following figures. The CH0 of the master (G0) and the slave (G1) are measured synchronously. After the conversion the CH1 of the master (G0) and the slave (G1) is measured.

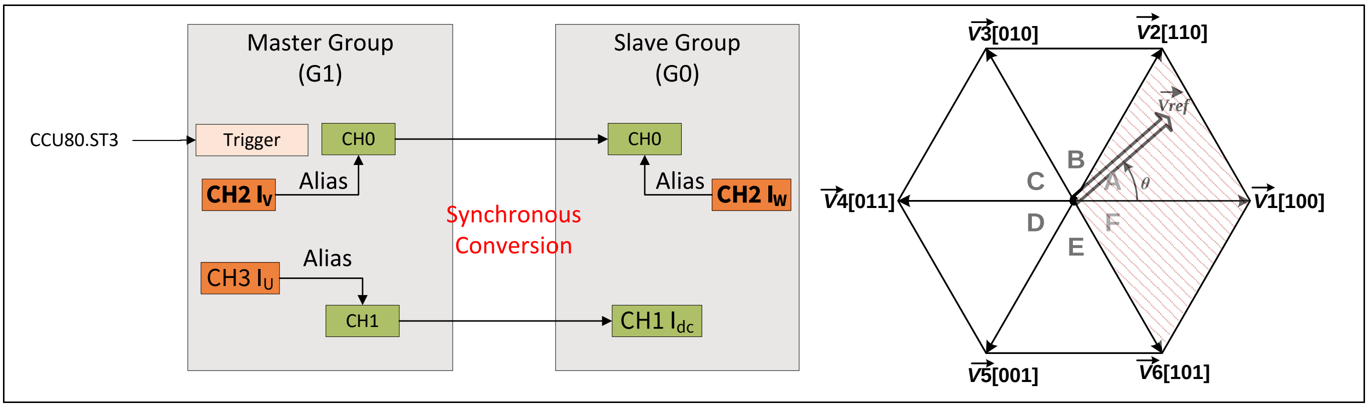


Figure 46 Synchronous conversion using alias feature in XMC1300 and XMC1400 – sectors A and F

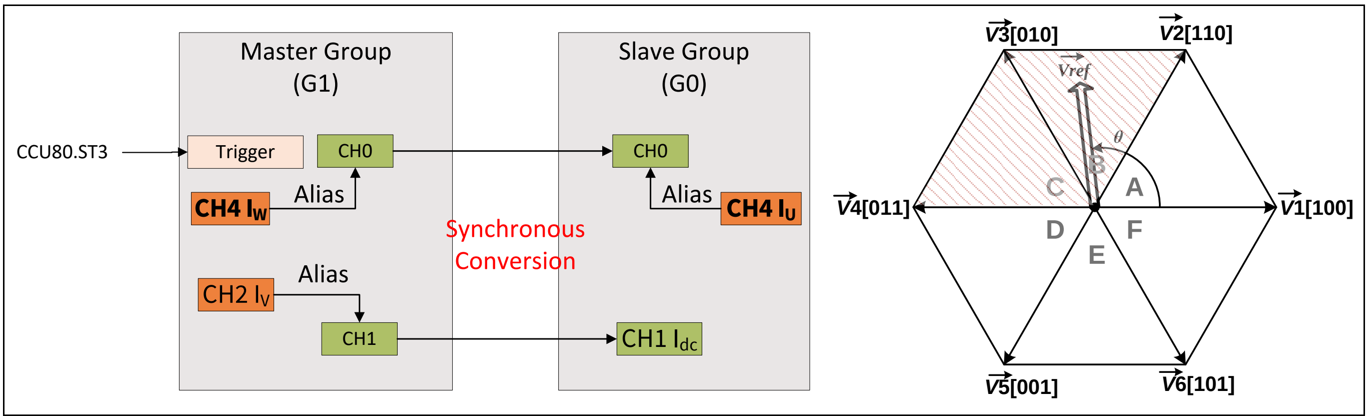


Figure 47 Synchronous conversion using alias feature in XMC1300 and XMC1400 – sectors B and C

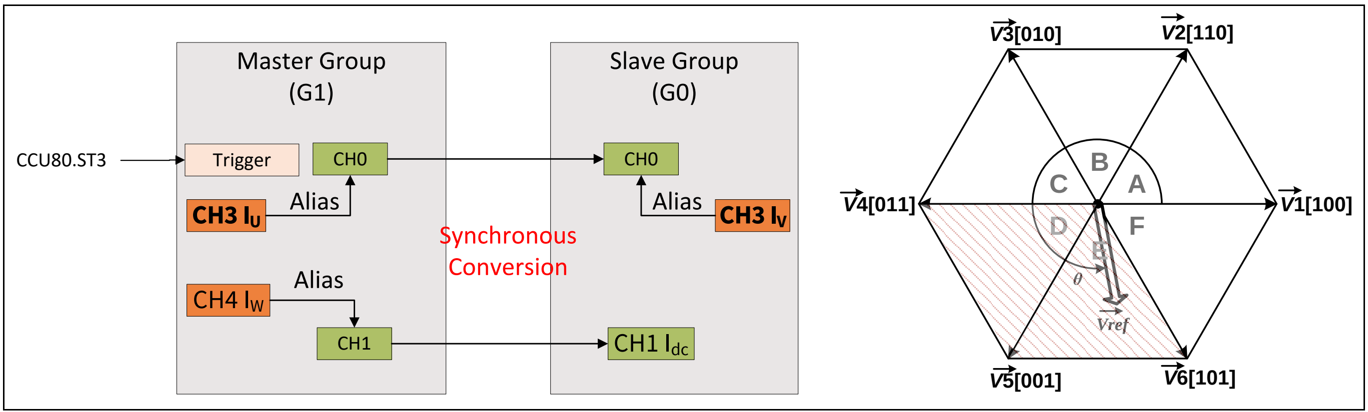


Figure 48 Synchronous conversion using alias feature in XMC1300 and XMC1400 – sectors D and E

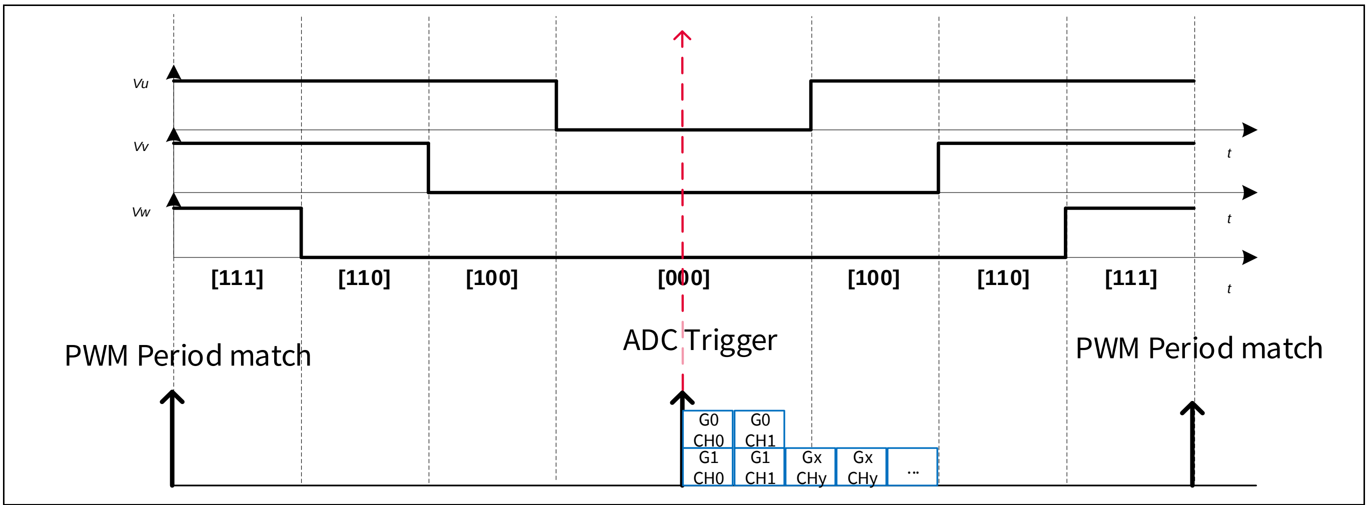


Figure 49 Synchronous conversion sequence in XMC1300/XMC1400

In XMC4400, all the three phases are sensed synchronously at the trigger point.

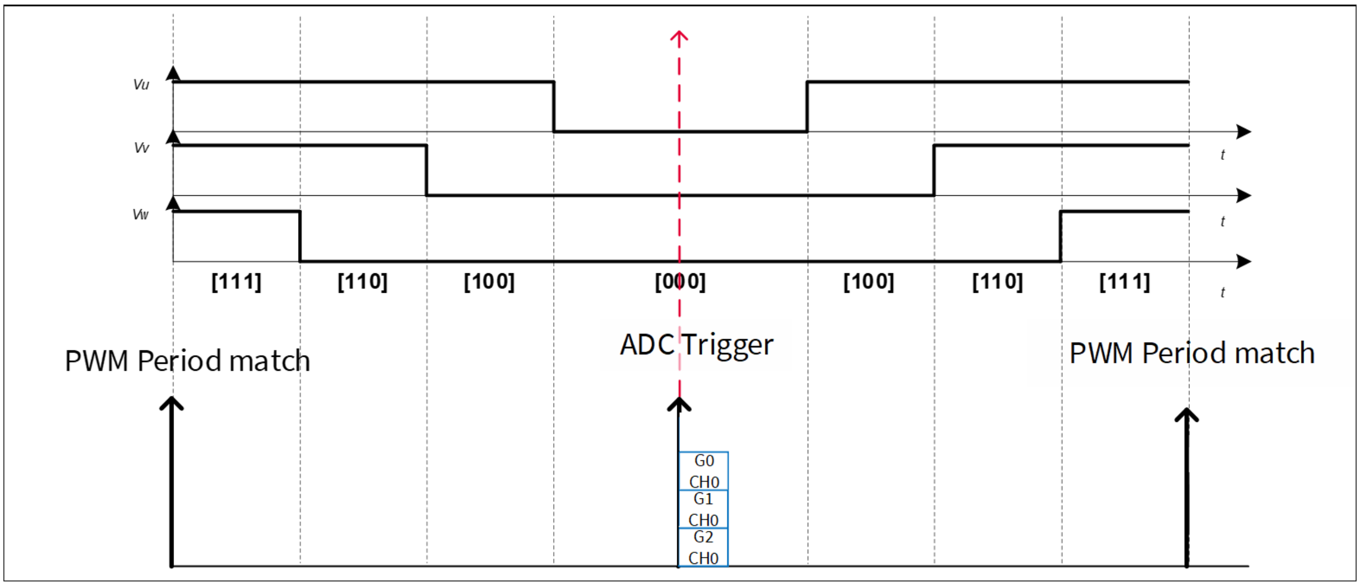


Figure 50 Synchronous conversion sequence in the XMC4400 family

4 Motor speed and position feedback in sensorless FOC control

The rotor speed and position feedback of the motor are determined in the PLL Estimator software library. This library contains the Infineon patented IP and is provided as a compiled libPLL_Estimator.a file. The following are the list of APIs provided in the library.

Note: It is important that these APIs are called in the exact order indicated.

1. PLL_Imag(int32_t Vref_AngleQ31, int32_t I_Alpha_1Q31, int32_t I_Beta_1Q31)
2. PLL_Imag_GetResult(PLL_EstimatorType* const HandlePtr)
3. PLL_Vref(int32_t Delta_IV, uint32_t Vref32, int32_t PLL_UK, int32_t Phase_L, PLL_EstimatorType* const HandlePtr)
4. PLL_Vref_GetResult(PLL_EstimatorType* const HandlePtr)
5. PLL_GetPosSpd(PLL_EstimatorType* const HandlePtr)

Below is a brief description of each API and the required parameters.

Table 15 PLL_Imag() function

Name	PLL_Imag(int32_t Vref_AngleQ31, int32_t I_Alpha_1Q31, int32_t I_Beta_1Q31)	
Description	In XMC1300/XMC1400. this function starts the first CORDIC calculation of the sensorless estimator. In XMC4400, this functions makes the first FPU calculation and loads the result in PLL_Estimator structure.	
Input Parameters	Vref_AngleQ31	Angle of voltage space vector
	I_Alpha_1Q31	Alpha coordinate of current space vector
	I_Beta_1Q31	Beta coordinate of current space vector
	HandlePtr	Pointer to the structure of PLL_Estimator
Return (only in XMC4400)	Current_I_Mag	Current magnitude
	Delta_IV	To be provided as input parameter for the second CORDIC calculation

Table 16 PLL_Imag_GetResult() function

Name	PLL_Imag_GetResult(PLL_EstimatorType* const HandlePtr)	
Description	In XMC1300/XMC1400, this function reads out the results of the first CORDIC calculation of the sensorless estimator. In XMC4400, this function is not called.	
Input Parameters	HandlePtr	Pointer to the structure of PLL_Estimator
Return (only in XMC1300/XMC1400)	Current_I_Mag	Current magnitude
	Delta_IV	To be provided as input parameter for the second CORDIC calculation

Table 17 PLL_Vref() function

Name	PLL_Vref(int32_t Delta_IV, uint32_t Vref32, int32_t PLL_UK, int32_t Phase_L, PLL_EstimatorType* const HandlePtr)

Description	In XMC1300/XMC1400, this function starts the second CORDIC calculation of the sensorless estimator. In XMC4400, this functions does the second FPU calculation and loads the result in PLL_Estimator pointer.	
Input Parameters	Delta_IV	Result of the first CORDIC calculation of the sensorless estimator
	Vref32	SVM voltage magnitude of last PWM cycle
	PLL_Uk	PLL Estimator PI controller output
	Phase_L	Phase inductance of motor stator winding
	HandlePtr	Pointer to the structure of PLL_Estimator
Return(in XMC1300/XMC1400)	Current_I_Mag	Updated current magnitude
Return(in XMC4400)	VrefxSinDelta	It is used for PLL_Estimator

Table 18 PLL_Vref_GetResult() function

Name	PLL_Vref_GetResult(PLL_EstimatorType* const HandlePtr)	
Description	In XMC1300/XMC1400, this function reads the results of the second CORDIC calculation of the sensorless estimator. In XMC4400, this function is not called.	
Input Parameters	HandlePtr	Pointer to the structure of PLL_Estimator
Return (only in XMC1300/XMC1400)	VrefxSinDelta	It is used for PLL_Estimator

Table 19 PLL_GetPosSpd() function

Name	PLL_GetPosSpd(PLL_EstimatorType* const HandlePtr)	
Description	This function is to calculate and read the rotor position and rotor speed from the sensorless estimator.	
Input Parameters	HandlePtr	Pointer to the structure of PLL_Estimator
Return	RotorAngleQ31	Estimated rotor position
	RotorSpeed_In	Estimated rotor speed

5 Interrupts

Interrupt events and priorities in the PMSM FOC software are listed in the following table:

Table 20 Interrupt priorities

Interrupt events	Priorities	Comment
CTrap	0 (Highest priority)	Fault detection
ADC Queue Source	1	Only for single shunt sensing(that is not implemented in XMC4400). It is disabled for three shunt sensing.
PWM Period Match (Phase U)	2	State machine execution.
Secondary Loop	3	APIs for low priority and low cycle frequency.
Over/Under Voltage Protection	1	Event happens only if DC bus is outside Voltage range

5.1 PWM period match interrupt

The PMSM_FOC state machine is executed in Phase U PWM frequency period match Interrupt Service Routine. The Interrupt Service Routine consists of a state machine (see **chapter 6**).

An example of the flow of the PWM period match interrupt is shown in Figure 51. This example shows the flow of the FOC direct startup control scheme. The current sensing technique chosen is three shunt synchronous conversion.

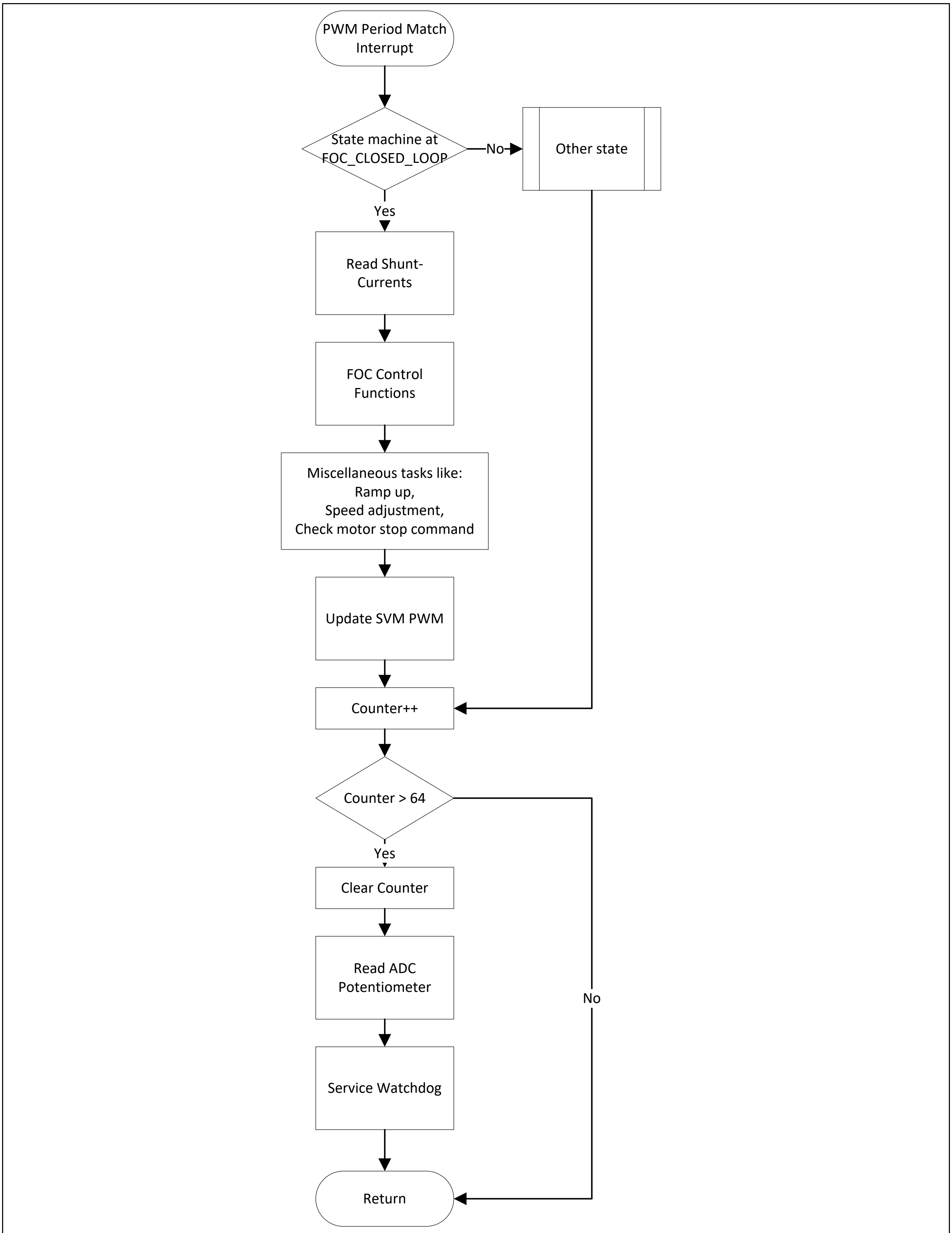


Figure 51 PWM period match Interrupt Service Routine flowchart

5.2 Ctrap interrupt

When a TRAP condition is detected at the selected input pin (P0.12), the CCU80 outputs are set to passive level and Trap_Protection_INT() is executed. In the Interrupt Service Routine, the gate driver is disabled and the motor state is set to TRAP_PROTECTION. This ISR is executed with the highest priority, level 0.

5.3 ADC source interrupt (only in XMC1300/XMC1400)

This interrupt is only enabled for single shunt current sensing. It is triggered at the end of ADC conversion. In the ISR, the ADC results are read.

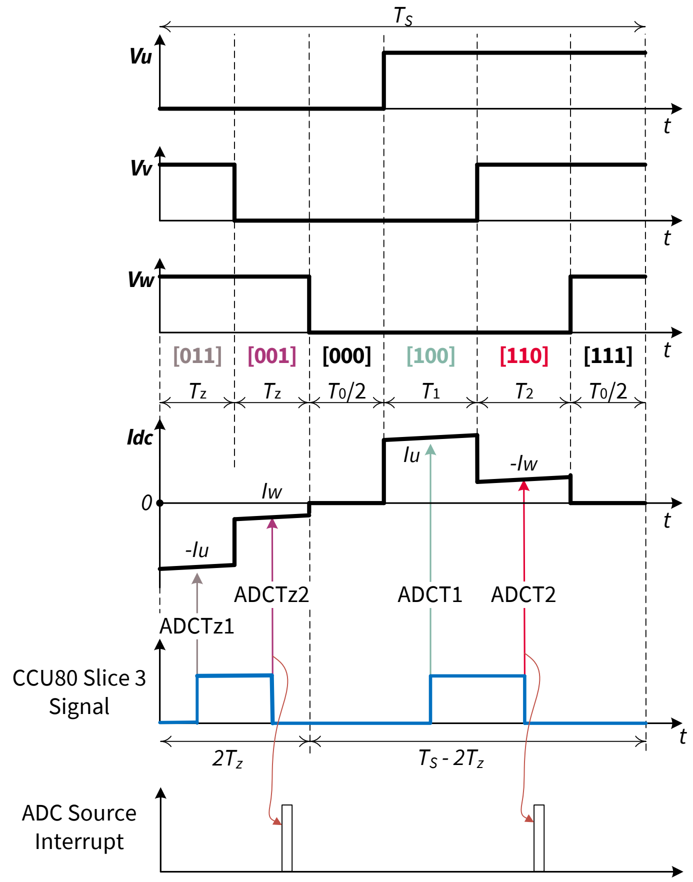


Figure 52 ADC source interrupt timing diagram

5.4 Secondary loop interrupt

This loop is used for slower tasks such as communication or a watchdog service. The trigger is generated from an independent timer. This means it is not mandatory that the frequency is synchronous with the PWM period match interrupt. For deterministic reasons its frequency is intended to be a fraction of USER_CCU8_PWM_FREQ_HZ (default 20 kHz).

5.5 Over/Under voltage protection

This interrupt is triggered from ADC only when the DC bus is outside of the voltage range.

6 Motor state machine

The PMSM FOC software has an internal state machine:

MOTOR_IDLE

This is the first state entered after power-on or software reset. In this state the inverter is disabled and it reads the bias voltage of the ADC pins that are connected to the motor phase currents. The state machine and Timer are started. Exit from this state occurs when the motor start command is received.

EN_INVERTER_BOOTSTRAP

In this state the inverter is enabled and the bootstrap capacitors are charged for a defined period. It reads the bias voltage of the ADC pins that are connected to the motor phase currents.

Exit from this state occurs after the bootstrap time.

PRE-POSITIONING

This state is only for Direct FOC Startup control schemes. In this state the rotor is aligned to a known position to get the maximum starting torque. The amplitude input to the SVM function is gradually increased to a defined value USER_STARTUP_VF_OFFSET_V, for a specific time USER_ROTOR_PREPOSITION_TIME_MS. These macros are defined in the pmsm_foc_motor_XXXX.h file (see **chapter 7.2.3.2**).

VF_OPENLOOP_RAMPUP

In this state the motor starts in V/F open loop control mode.

Exit from this state occurs when the motor speed reaches the startup threshold speed defined in the macro USER_STARTUP_SPEED_THRESHOLD_RPM.

MET_FOC

This state enables a smooth transition from open loop to closed loop with maximum energy efficiency.

FOC_CLOSED_LOOP

In this state the motor is running in FOC mode. The FOC functions are executed.

FOC_CLOSED_LOOP_BREAK

This function is the same as the FOC_CLOSED_LOOP expect that no target values are accepted. The target value is ramping down in an S-curve.

After crossing the FOC_EXIT_SPEED the state is changed to MOTOR_STOP.

MOTOR_HOLD

This state is entered when the motor speed is below 10% of the maximum speed. The motor is set to hold with a 50% ON and 50% OFF PWM. A Motor break command in this state will lead directly to a MOTOR_STOP.

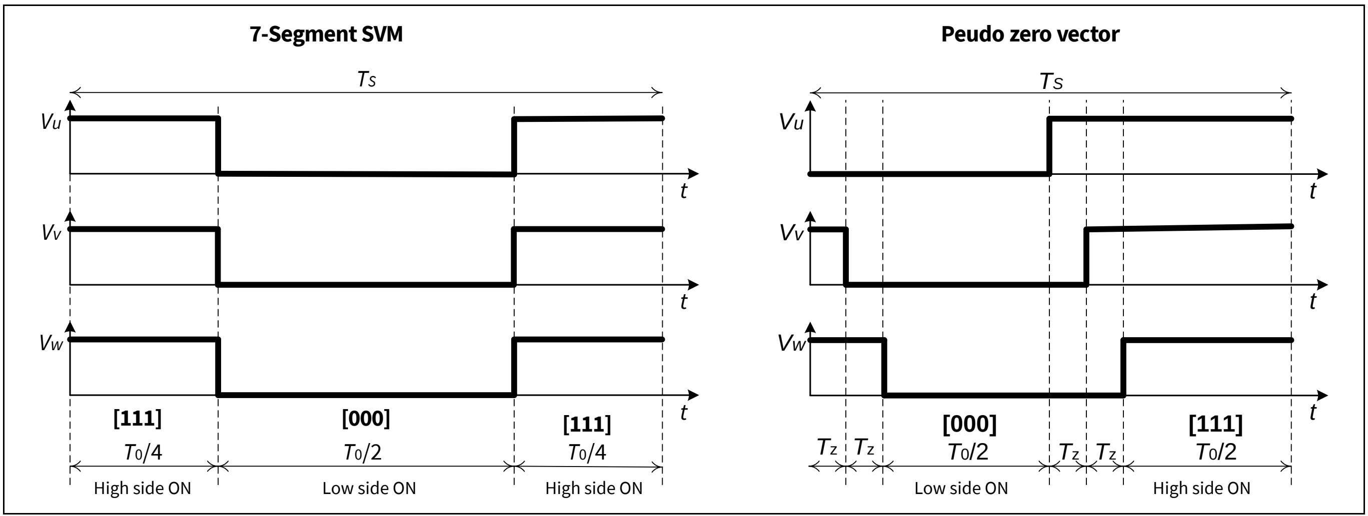


Figure 53 SVM outputs in motor brake condition

MOTOR_STOP

This state is entered from all states with the stop command (and other state related conditions). The motor output is set to tristate and the inverter is disabled. This leads to an uncontrolled freewheeling of the motor. The state exits to the idle state after processing.

TRAP_PROTECTION

This state is entered if Ctrap is triggered.

To exit this state, set the target value below the MOTOR_HOLD_THRESHOLD and set the break or stop command.

DCLINK_UNDER_VOLTAGE

This state is entered when the DC link voltage is below the limits set by the user. The gate driver is disabled and the motor will be in free running.

Only the motor stop or motor brake command will exit this state.

DCLINK_OVER_VOLTAGE

This state is entered when the DC link voltage is above the limits set by the user. The gate driver is disabled, and the motor will be free-running.

Only the motor stop or motor brake command will exit this state.

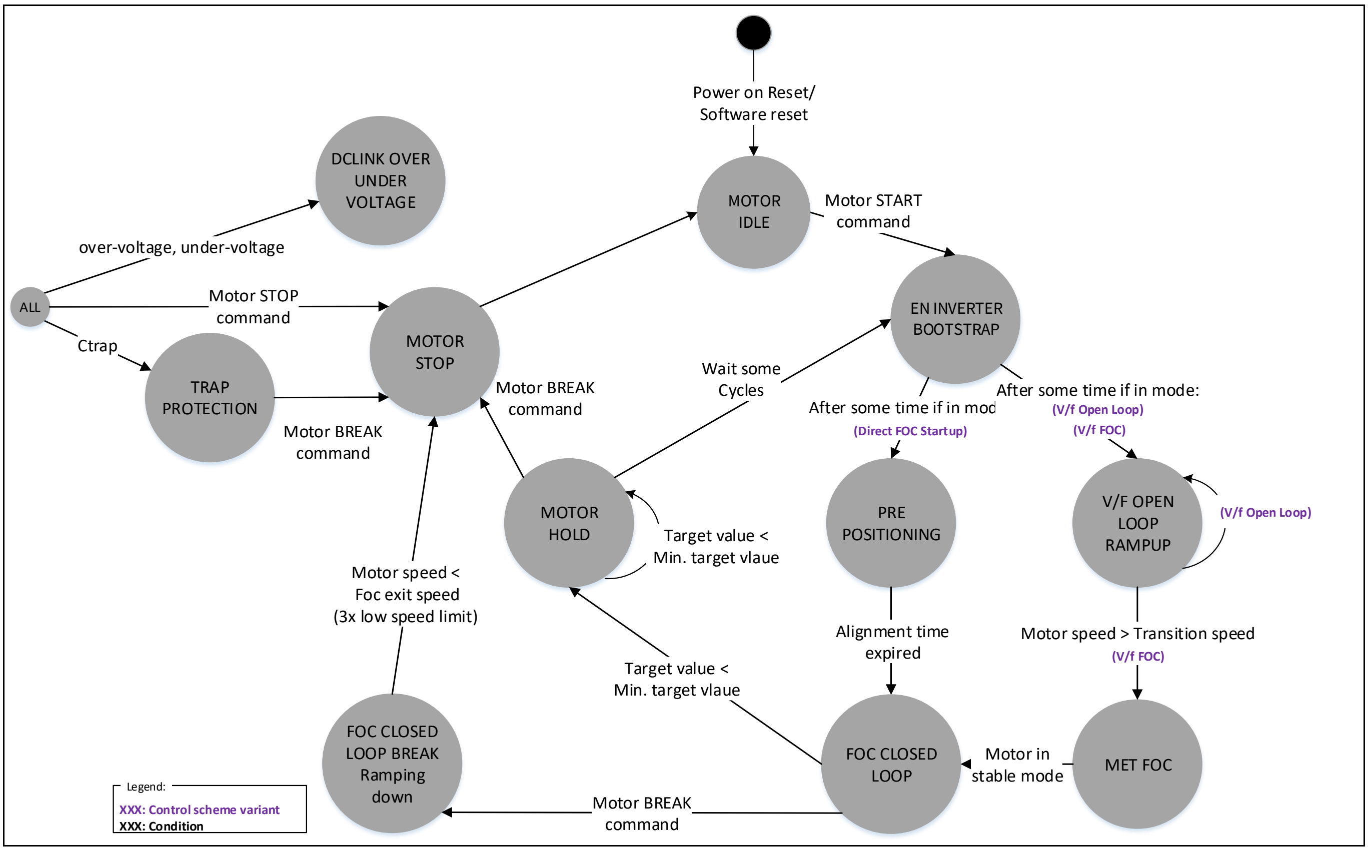


Figure 54 PMSM FOC state machine

For different control schemes, the flow of the state machine is different.

The following schemes are available:

- Direct FOC Startup
  - SPEED_CONTROLLED_DIRECT_FOC
  - TORQUE_CONTROLLED_DIRECT_FOC
  - VQ_CONTROLLED_DIRECT_FOC
- V/f FOC
  - SPEED_CONTROLLED_VF_MET_FOC
- V/f Open Loop
  - SPEED_CONTROLLED_VF_ONLY

7 Configuration

The default configuration of the parameters in the PMSM FOC software is set based on the XMC1000 Motor Control Application Kit. The configuration is split up in to 3 levels:

- User configuration
  - Allows fast access to the general configuration such as the FOC scheme and selection of one of the pre-configured, purchasable hardware boards.
- Hardware configuration
  - Allows for specialization of the controller card, inverter card, and motors. With this configuration you can adapt the hardware configuration to your application and board. Configurations such as pinout and maximum speed can be found here.
- FOC configuration
  - Allows you to change basic values and constant calculations. One example is the over-voltage definition of 120% of the DC link voltage. Most of this configurations are essential, calculated, and should not be changed by you, which is why they are not described in this document.

7.1 User Configuration

The user configuration allows for fast access to the general configuration such as the FOC scheme and selection of one of the pre-configured, purchasable hardware boards. All configuration options are available in the file PMSM_FOC\Configuration\file pmsm_foc_user_config.h.

The default settings are for the "Maxon motor 267121" used in the Infineon XMC1302 Motor Control Application Kit, KIT_XMC1X_AK_MOTOR_001.

7.1.1 General

PMSM FOC hardware Kit

Various configuration options are available for the software. The configurations are mainly independent from each other. The hardware consists of:

- Controller_Card (MCUCARD_TYPE)
- Inverter_Card (INVERTERCARD_TYPE)
- Motor (MOTOR_TYPE)

In this document a combination of all three is referred to as a:

- hardware kit (PMSM_FOC_HARDWARE_KIT)

Many purchasable boards are pre-defined. Additionally it is possible to exchange parts of the hardware board. To support these options you can select a custom hardware kit and adapt the MCUCARD_TYPE, INVERTERCARD_TYPE, and MOTOR_TYPE, and the associated paths.

```
#define PMSM_FOC_HARDWARE_KIT                    KIT_XMC1X_AK_MOTOR_001
```

- Select a pre-defined hardware kit.

Options:

- `KIT_XMC1X_AK_MOTOR_001` – Infineon XMC1000 Motor Control Application Kit
- `KIT_XMC750WATT_MC_AK_V1` – XMC 750Watt Motor Control Application Kit

- `KIT_XMC14_BOOT_001` – XMC1404 CPU card for KIT_XMC1X_AK_MOTOR_001
- `KIT_XMC750WATT_MC_AK_V1` – with XMC1404 version
- `KIT_MOTOR_DC_250W_24V`
- `IFX_MADK_EVAL_M1_05F310 Kit`
- `IFX_MADK_EVAL_M1_05_65D_V1`
- `IFX_MADK_EVAL_M1_CM610N3`
- `CUSTOM_KIT` – User defined motor control system.
- `KIT_XMC4400_EXAGON`
- `KIT_XMC4400_ISOLATED`

Current sensing

```
#define CURRENT_SENSING
        USER_THREE_SHUNT_SYNC_CONV
```

- Define the current sensing technique used.

Options:

- `USER_SINGLE_SHUNT_CONV` – Single shunt current sensing technique with Pseudo Zero Vector PWM generation, refer to **chapter 3.1** (not implemented in XMC4400)
- `USER_THREE_SHUNT_ASSYNC_CONV` – Three shunt current sensing technique with ADC standard conversion, refer to **chapter 3.2** (not implemented in XMC4400)
- `USER_THREE_SHUNT_SYNC_CONV` – Three shunt current sensing technique with ADC synchronous conversion, refer to **chapter 3.2.1**

FOC control schematic

```
#define MY_FOC_CONTROL_SCHEME                    SPEED_CONTROLLED_DIRECT_FOC
```

- Define the FOC control scheme.

Options:

- `SPEED_CONTROLLED_DIRECT_FOC` – Direct FOC start-up using speed control, refer to **chapter 2.3.2**
- `SPEED_CONTROLLED_VF_ONLY` – Open loop speed control, refer to **chapter 2.3.1**
- `SPEED_CONTROLLED_VF_MET_FOC` – Open loop start-up to MET to closed loop FOC speed control, refer to **chapter 2.3.2**
- `TORQUE_CONTROLLED_DIRECT_FOC` – Direct FOC start-up using torque control, refer to **chapter 2.3.3**
- `VQ_CONTROLLED_DIRECT_FOC` – Direct FOC start-up using voltage torque control, refer to **chapter 2.3.4**

Input selection for target values

```
#define SETTING_TARGET_SPEED                          SET_TARGET_SPEED
```

- Define the concept of how a target value is provided.

Note: *Only one option out of the following options is available at the same time. Additionally, depending on the MY_FOC_CONTROL_SCHEME, the input is stored as Target_Speed, Target_Torque, or Target_Voltage.*

Options:

- `SET_TARGET_SPEED` – An API function is available to store the target value. Additionally µC/Probe can be used for update.
- `BY_POT_ONLY` – The potentiometer is used to control motor operation via the ADC pin.
- `BY_UART_ONLY` – Reference speed is set via UART communication. (not available in XMC4400)

SVM switching schematic

```
#define SVM_SWITCHING_SCHEME                           STANDARD_SVM_7_SEGMENT
```

- Define the SVM switching scheme.

Options:

- `STANDARD_SVM_7_SEGMENT` – 7-segment switching mode (see **chapter 2.5.1**)
- `STANDARD_SVM_5_SEGMENT` – 5-segment switching mode (see **chapter 2.5.2**)

µC/Probe GUI selection

```
#define uCPROBE_GUI_OSCILLOSCOPE                       ENABLED
```

- Enable or disable the firmware support for Micrium µC/Probe GUI and oscilloscope tool. If enabled it is called in the pmsm_foc_controlloop_isr().

Protections and limitations

```
#define VDC_UNDER_OVERVOLTAGE_PROTECTION               ENABLED
```

- Enable or disable the DC link voltage protection.

```
#define VDC_UNDER_OVERVOLTAGE_PERCENTAGE                (20U)
```

- Set limit check +-20% for over-voltage and under-voltage.

```
#define OVERCURRENT_PROTECTION                         ENABLED
```

- Enable or disable DC link current protection. (not implemented in XMC4400)

```
#define USER_IDC_MAXCURRENT_A                          (10.0f)
```

- This setting is the maximum DC link current limit. This limit is checked if the over-current protection feature is enabled. Once this limit is hit, the reference speed is reduced (see over-current protection in **chapter 2.9**). (not implemented in XMC4400)

```
#define VDC_MAX_LIMIT                                  ((VADC_DCLINK * 19U)>>4)
```

- Set the maximum DC link voltage limit for ramp-down operation. The default setting is 18.7% more than nominal DC link voltage. You should change this limit according to your hardware design.

```
#define WATCH_DOG_TIMER                         ENABLED
```

- Enable or disable the watchdog timer feature.

7.1.2 Custom Kit configuration

Controller_Card

```
#define MCUCARD_TYPE                    defined by PMSM_FOC_HARDWARE_KIT
#define MCUCARD_TYPE_PATH               defined by PMSM_FOC_HARDWARE_KIT
```

- In the default configuration this is defined by the hardware board. To configure a custom MCU card or combination, the hardware board CUSTOM_KIT should be used. Only the hardware board combinations are tested.

Options for MCUCARD_TYPE:

- `CUSTOM_MCU`
- `EVAL_M1_1302`
- `KIT_XMC13_BOOT_001`
- `KIT_XMC1300_DC_V1`
- `BOOTKIT_XMC1400_V1`
- `KIT_XMC1400_DC_V1`
- `KIT_XMC4400_DC_V1`
- `KIT_XMC4400_EE1_001`

Note: It is necessary to adapt the MCUCARD_TYPE_PATH according to the selection.

Inverter_Card

```
#define INVERTERCARD_TYPE               defined by PMSM_FOC_HARDWARE_KIT
#define INVERTERCARD_TYPE_PATH          defined by PMSM_FOC_HARDWARE_KIT
```

- In the default configuration this is defined by the hardware board. To configure a custom inverter card or combination, the hardware board CUSTOM_KIT should be used. Only the hardware board combinations are tested.
- `CUSTOM_INVERTER`
- `EVAL_M1_05_65A`
- `EVAL_M1_05F310`
- `EVAL_M1_CM610N3`
- `KIT_MOTOR_DC_250W_24V`
- `PMSM_LV15W`
- `POWERINVERTER_750W`
- `KIT_XMC4X_MOT_GPDLV_001`

Note: It is necessary to adapt the INVERTERCARD_TYPE_PATH according to the selection.

Motor

```
#define MOTOR_TYPE                      defined by PMSM_FOC_HARDWARE_KIT
#define MOTOR_TYPE_PATH                 defined by PMSM_FOC_HARDWARE_KIT
```

- In the default configuration this is defined by the hardware board. To configure a custom motor or combination the hardware board CUSTOM_KIT should be used. Only the hardware board combinations are tested.

- `CUSTOM_MOTOR`
- `MAXON_MOTOR_267121`
- `NANOTEC_MOTOR_DB42S03`

Note: It is necessary to adapt the INVERTERCARD_TYPE_PATH according to the selection.

7.1.3 Advanced user configuration

ADC specific configurations

```
#define ADC_STARTUP_CALIBRATION                          DISABLED
```

- Enable or disable the ADC startup calibration feature. Please enable this feature if using XMC1302AA step (not implemented in XMC4400).

```
#define ADC_ALTERNATE_REFERENCE                          DISABLED
```

- Enable or disable the ADC alternative reference feature. If enabled, the channel 0 is used as the reference (not implemented in XMC4400).

```
#define ADC_ALTERNATE_REF_PHASEUVW                       DISABLED
```

- This value disables or enables the alternate reference for phase UVW. This option is only available if alternate reference is enabled (not implemented in XMC4400).

```
#define ADC_ALTERNATE_REF_SINGLESHUNT                    DISABLED
```

- This value disables or enables the alternate reference for phase single shunt. This option is only available if alternate reference is enabled (not implemented in XMC4400).

```
#define MOTOR_HOLD_THRESHOLD
        Defined by MY_FOC_CONTROL_SCHEME
```

- This value defines the minimum valid digital value. This value is required due to the physical behavior of potentiometers and analog-to-digital conversion. All values below this value are assumed to be zero or off.

```
#define FOC_EXIT_SPEED                                   (SPEED_LOW_LIMIT * 3)
```

- This is the limit until the motor is ramping down before changing the state from FOC_CLOSED_LOOP_BREAKING to MOTOR_STOP.

```
#define SPEED_LOW_LIMIT                                  SPEED_LOW_LIMIT_RPM
#define SPEED_LOW_LIMIT_RPM                              calculated
```

- This value is used as the minimum allowed speed. It is calculated from USER_SPEED_LOW_LIMIT_RPM but scaled for MCU calculation.

Secondary Loop callback

```
#define PMSM_FOC_SECONDARYLOOP_CALLBACK                  DISABLED
```

- Enable or disable a callback in the secondary loop. If enabled the function needs to be created by the user. The callback function is by default executed in flash to reduce SRAM consumption. For fast execution you have to manually add the SRAM code attribute.

```
#define USER_SECONDARY_LOOP_FREQ_HZ                      (1000U)
```

- This value defines the target secondary loop frequency in Hz. This loop is used for slower tasks like communication or a watchdog service. This frequency is intended to be a fraction of USER_CCU8_PWM_FREQ_HZ (default 20 kHz)

FOC control

```
#define DQ_DECOUPLING                                   ENABLED
```

- Enable or disable the dq decoupling feature.

7.1.4 Torque control specific

These configurations are only available if:

MY_FOC_CONTROL_SCHME is set to TRQUE_CONTROLLED_DIRECT_FOC

```
#define USER_IQ_CURRENT_ALLOWED_A                       (2.0f)
```

- Set the high limit of the reference torque current in amperes.

```
#define USER_IQ_REF_LOW_LIMIT                           (0U)
```

- Set the low limit of the reference torque current.

```
#define USER_IQ_REF_HIGH_LIMIT  (32768* USER_IQ_CURRENT_ALLOWED_A/I_MAX_A)
```

- Scale the high limit of the reference torque current in 1Q15 format.

```
#define USER_IQ_RAMPUP                                  (10U)
```

- Torque current ramp-up steps used in linear ramp generator.
- 1 step is (1/32768) * I_MAX_A where I_MAX_A is equal to (5.0 V/(USER_R_SHUNT_OHM * OP_GAIN_FACTOR)) / 2.

```
#define USER_IQ_RAMPDOWN                                (10U)
```

- Torque current ramp-down steps used in linear ramp generator.
- 1 step is (1/32768) * I_MAX_A where I_MAX_A is equal to (5.0 V/(USER_R_SHUNT_OHM * OP_GAIN_FACTOR)) / 2.

```
#define USER_IQ_RAMP_SLEWRATE                           (50U)
```

- Define the frequency to ramp-up/ramp-down I_q.
- In every USER_IQ_RAMP_SLEWRATE *PWM cycles, ramp-up I_q by USER_IQ_RAMPUP steps, or ramp-down I_q by USER_IQ_RAMPDOWN step.

7.1.5 VQ control specific (V/f)

```
#define USER_VQ_VOLTAGE_ALLOWED_V                          (10U)
```

- Set the limit of the torque voltage in volts. This value must be less than VREF_MAX_V.

```
#define USER_VQ_REF_LOW_LIMIT                              (0U)
```

- Set the low limit of the reference torque voltage.

```
#define USER_VQ_REF_HIGH_LIMIT
        (32768* USER_VQ_VOLTAGE_ALLOWED_V/VREF_MAX_V)
```

- Scale the limit of the reference torque voltage to 1Q15 format.
- VREF_MAX_V is defined as DC link voltage divided by square root of 3.

```
#define USER_VQ_RAMPUP                                     (2U)
```

- Define the number of steps to ramp-up the torque voltage in the linear ramp generator.

```
#define USER_VQ_RAMPDOWN                                   (2U)
```

- Define the number of steps to ramp-down the torque voltage in the linear ramp generator.

```
#define USER_VQ_SLEWRATE                                   (10U)
```

- Define the frequency to ramp-up/ramp-down V_q. In every USER_VQ_RAMP_SLEWRATE * PWM cycles, ramp-up V_q by USER_VQ_RAMPUP steps, or ramp-down V_q by USER_VQ_RAMPDOWN step.

7.1.6 MET specific

```
#define USER_MET_THRESHOLD_HIGH                            (64U)
```

- Define the high threshold limit for speed hysteresis control in MET motor state.

```
#define USER_MET_THRESHOLD_LOW                             (16U)
```

- Define the low threshold limit for speed hysteresis control in MET motor state.

```
#define USER_MET_LPF                                       (2U)
```

- Low pass filter factor for the MET threshold calculation, Y[n] = Y[n-1] + (X[n] – Y[n-1]) >> USER_MET_LPF

7.2 Hardware configuration

The detailed hardware configuration allows specialization of the controller card, inverter card, and motors. With this configuration you can adapt the hardware configuration to your application and board. Configurations such as pinout and maximum speed can be found here.

The configuration is separated in to three types:

- PMSM_FOC\Configuration\Controller_Card
- PMSM_FOC\Configuration\Inverter_Card
- PMSM_FOC\Configuration\Motors

7.2.1 Controller Card

Table 21 GPIO configuration

Define	XMC1300/XMC1400 value	XMC4400 value	Notes
#define TRAP_PIN	P0_12	P0_7	External CCU80 Ctrap input pin assignment
#define INVERTER_EN_PIN	P0_11	P0_12	This pin connects to the enable pin of the inverter switch/gate drivers.
#define PHASE_U_HS_PIN	P0_0	P0_5	
#define PHASE_U_LS_PIN	P0_1	P0_2	
#define PHASE_V_HS_PIN	P0_7	P0_4	
#define PHASE_V_LS_PIN	P0_6	P0_1	
#define PHASE_W_HS_PIN	P0_8	P0_6	
#define PHASE_W_LS_PIN	P0_9	P0_11	

Table 22 XMC Alternate output pin register setting

Define	XMC1300/XMC1400 value	XMC4400 value
#define PHASE_U_HS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3
#define PHASE_U_LS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3
#define PHASE_V_HS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3
#define PHASE_V_LS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3
#define PHASE_W_HS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3
#define PHASE_W_LS_ALT_SELECT	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5	XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3

- XMC alternate output pin register value setting. CCU80 PWM output is selected.
- Please refer to XMC1300 or XMC1400 reference manual [1] for the alternate output setting.

Table 23 Primary Loop and SVM Resources (CCU8)

Define	XMC1300/XMC1400 value	Notes
#define CCU8_MODULE	CCU80	CCU80
#define CCU8_MODULE_PHASE_U	CCU80_CC80	CCU80_CC80
#define CCU8_MODULE_PHASE_V	CCU80_CC81	CCU80_CC81
#define CCU8_MODULE_PHASE_W	CCU80_CC82	CCU80_CC83
#define CCU8_MODULE_ADC_TR	CCU80_CC83	CCU80_CC82
#define CCU8_MODULE_PRESCALER_VALUE	(0U)	(0U)

- Assign the XMC CCU8 module for SVM generation. Multiple modules are available, with the actual amount dependent on the device.
- Assign the timer slice of the CCU8 module for phase U, V, W PWM generation and current trigger.
- Assign the CCU8 module timer slice for the ADC conversion trigger (TR). The pre-scaler effects the resolution. Changing the frequency should re-calculate this value. The lowest possible value should always be taken.

Secondary loop resources (CCU4)

```
#define SECONDARY_LOOP_MODULE                              CCU40
#define SECONDARY_LOOP_SLICE                               CCU40_CC42
#define SECONDARY_LOOP_MODULE                              (0U)
```

- Necessary only in XMC4400.

```
#define SECONDARY_LOOP_SLICE_NUM                           (2U)
#define SECONDARY_LOOP_SLICE_SHADOW_TRANS_ENABLE_Msk
      XMC_CCU4_SHADOW_TRANSFER_SLICE_2
#define SECONDARY_LOOP_SLICE_PRESCALER                     (2U)
```

- Assign the CCU4 module timer slice for the secondary loop. It is mandatory that slice name, slice number, and slice shadow transfer enable mask match. The pre-scaler effects the resolution. Changing the frequency should re-calculate this value.

ADC resources for three shunt synchronous conversion for XMC1300/XMC1400 (VADC)

```
#define VADC_IU_G1_CHANNEL                                 (3U)         // P2.11
#define VADC_IU_G0_CHANNEL                                 (4U)         // P2.11
#define VADC_IV_G1_CHANNEL                                 (2U)         // P2.10
#define VADC_IV_G0_CHANNEL                                 (3U)         // P2.10
#define VADC_IW_G1_CHANNEL                                 (4U)         // P2.9
#define VADC_IW_G0_CHANNEL                                 (2U)         // P2.9
```

- The channel number is equivalent to a pin. In the default configuration channel IU, IV, and IW from G0 and G1 are connected to the same pin, reducing the pin consumption.

Note: The connection between group channel number and pin is visible in the reference manual.

ADC resources for three shunt asynchronous conversion for XMC1300/XMC1400 (VADC)

```
#define VADC_IU_GROUP                                     VADC_G1
#define VADC_IU_GROUP_NO                                  (1U)
#define VADC_IV_GROUP                                     VADC_G1
#define VADC_IV_GROUP_NO                                  (1U)
#define VADC_IW_GROUP                                     VADC_G1
#define VADC_IW_GROUP_NO                                  (1U)
```

- This configuration selects which ADC group is used for measurement. It is mandatory that the Group name and number match. Additionally, in the default the group for IU, IV, and IW needs to be the same. If you prefer to use two groups, it is necessary to have one XMC_VADC_QUEU_ENTRY with .external_trigger = true per group.

```
#define VADC_IU_CHANNEL                                   (3U)        // P2.11
#define VADC_IU_RESULT_REG                                (3U)
#define VADC_IV_CHANNEL                                   (2U)        // P2.10
#define VADC_IV_RESULT_REG                                (2U)
#define VADC_IW_CHANNEL                                   (4U)        // P2.9
#define VADC_IW_RESULT_REG                                (4U)
```

- The channel number is equivalent to a pin. The connection between group channel number and pin is visible in the reference manual. The only restriction in selecting the result register is that every register needs a unique number.

ADC resources for three shunt synchronous conversion for XMC4400 (VADC)

```
#define VADC_IU_GROUP                                     VADC_G0
#define VADC_IU_GROUP_NO                                  (0U)
#define VADC_IV_GROUP                                     VADC_G1
#define VADC_IV_GROUP_NO                                  (1U)
#define VADC_IW_GROUP                                     VADC_G2
#define VADC_IW_GROUP_NO                                  (2U)
#define VADC_IU_CHANNEL                                   (1U)
#define VADC_IU_RESULT_REG                                (15U)
#define VADC_IV_CHANNEL                                   (7U)
#define VADC_IV_RESULT_REG                                (3U)
#define VADC_IW_CHANNEL                                   (1U)
#define VADC_IW_RESULT_REG                                (0U)
```

ADC resources for single shunt conversion (VADC) (only for XMC1300/XMC1400)

```
#define VADC_ISS_GROUP                                    VADC_G1
#define VADC_ISS_GROUP_NO                                 (1U)
#define VADC_ISS_CHANNEL                                  (1U)        // P2.7
#define VADC_ISS_RESULT_REG                               (15U)
```

- This configuration selects which ADC group is used for measurement. It is mandatory that the Group name and number match. The channel number is equivalent to a pin. The connection between group channel number and pin is visible in the reference manual. There are no other restrictions regarding the group, channel, or result register selection.

ADC resources for DC link voltage measurement (VADC)

```
#define VADC_VDC_GROUP                                      VADC_G1
#define VADC_VDC_GROUP_NO                                   (1U)
#define VADC_VDC_CHANNEL                                    (5U)    //(1U) in XMC4400
#define VADC_VDC_RESULT_REG                                 (5U)    //(4U) in XMC4400
```

- This configuration selects which ADC group is used for measurement. It is mandatory that the Group name and number match. The channel number is equivalent to a pin. The connection between group channel number and pin is visible in the reference manual. There are no other restrictions regarding the group, channel, or result register selection.

ADC resources for average DC link voltage measurement (VADC) (only in XMC1300/XMC1400)

```
#define VADC_IDC_GROUP                                      VADC_G0
#define VADC_IDC_GROUP_NO                                   (0U)
#define VADC_IDC_CHANNEL                                    (6U)        // P2.1
#define VADC_IDC_RESULT_REG                                 (6U)
```

- This configuration selects which ADC group is used for measurement. It is mandatory that the Group name and number match. The channel number is equivalent to a pin. The connection between group channel number and pin is visible in the reference manual. There are no other restrictions regarding the group, channel, or result register selection.

ADC resources for Potentiometer measurement (VADC)

```
#define VADC_POT_GROUP                                      VADC_G1
#define VADC_POT_GROUP_NO                                   (1U)
#define VADC_POT_CHANNEL                                    (7U)    //(5U) in XMC4400
#define VADC_POT_RESULT_REG                                 (7U)    //(14U) in XMC4400
```

- This configuration selects which ADC group is used for measurement. It is mandatory that the Group name and number match. The channel number is equivalent to a pin. The connection between group channel number and pin is visible in the reference manual. There are no other restrictions regarding the group, channel, or result register selection.

UART pin configuration (UART) (only in XMC1300/XMC1400)

```
#define UART_ENABLE                                         USIC0_CH1_P1_2_P1_3
```

- If SETTING_TARGET_SPEED is set to BY_UART_ONLY the input output pins can be configured.

Options:

- USIC0_CH0_P1_4_P1_5 – UART channel 0 used to receive commands to control motor operations. Port pins P1.4 and P1.5 are configured to receive and transmit data. ADC potentiometer result is discarded.

– USIC0_CH1_P1_2_P1_3 – UART channel 1 used to receive commands to control motor operations. Port pins P1.2 and P1.5 are configured to transmit and receive data. ADC potentiometer result is discarded.

Debug PWM (CCU4) (only in XMC1300/XMC1400)

```
#define DEBUG_PWM_0_ENABLE                                    (0U)
#define DEBUG_PWM_1_ENABLE                                    (0U)
```

– Up to two debug PWMs are available. This value enables instance 0 and/or instance 1 of the debug PWM.

Options:

– `0` - Disabled
– `1` – Enabled

```
#define DEBUG_PWM_CCU4_MODULE                                 CCU40
#define DEBUG_PWM_PERIOD_CNTS                                 (400U)
#define DEBUG_PWM_50_PERCENT_DC_CNTS
       ((uint16_t)(DEBUG_PWM_PERIOD_CNTS >> 1))
```

– This configurations defines which module is used, the Frequency via Period cnts and 50% dc.

```
#define REVERSE_CRS_OR_0                                      (- Tmp_CRS)
```

– This value defines how negative values are interpreted.

Options:

– `0` - zero
– `- Tmp_CRS` - absolute value

```
#define DEBUG_PWM_0_SLICE                                     CCU40_CC40
#define DEBUG_PWM_0_SLICE_NUM                                 (0U)
#define DEBUG_PWM_0_SLICE_SHADOW_TRANS_ENABLE_Msk
       XMC_CCU4_SHADOW_TRANSFER_SLICE_0
```

– Assign CCU4 module timer slice for debugPWM 0. It is mandatory that the slice name, slice number, and slice shadow transfer enable mask all match. The pre-scaler is by default 0 to support the highest frequency.

```
#define DEBUG_PWM_0_PORT                                      XMC_GPIO_PORT1
#define DEBUG_PWM_0_PIN                                       (0U)
#define DEBUG_PWM_0_ALT_OUT
       XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT2
```

– This configures the output of the PWM for debug instance 0. It is mandatory that an associated Port-Pin, alternate output, and CCU4 slice are all chosen. This information can be found in the reference manual.

```
#define DEBUG_PWM_1_SLICE                                     CCU40_CC41
#define DEBUG_PWM_1_SLICE_NUM                                 (1U)
#define DEBUG_PWM_1_SLICE_SHADOW_TRANS_ENABLE_Msk
       XMC_CCU4_SHADOW_TRANSFER_SLICE_1
```

- Assign CCU4 module timer slice for debugPWM 1. It is mandatory that the slice name, slice number, and slice shadow transfer enable mask all match. The pre-scaler is per default 0 to support the highest frequency.

```
#define DEBUG_PWM_1_PORT                                                XMC_GPIO_PORT0
#define DEBUG_PWM_1_PIN                                                 (4U)
#define DEBUG_PWM_1_ALT_OUT
      XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT4
```

- This configures the output of the PWM for debug instance 1. It is mandatory that an associated Port-Pin, alternate output and CCU4 slice is chosen. This information can be found in the reference manual.

7.2.2 Inverter board configuration for current and voltage sensing

General

```
#define USER_CCU8_PWM_FREQ_HZ                           (20000U)
```

- This macro defines the PWM frequency in Hz. This is the fastest loop and the control loop. The main tasks of the FOC are done in this loop or fractions of it. The PMSM FOC software can support up to 25 kHz (with a maximum 30 kHz in some cases).

```
#define USER_MAX_ADC_VDD_V                              (5.0f)
```

- This value defines the maximum input of the XMC ADC in volts. The XMC1000 family has a wide input range. Common is 5V and 3.3V. The input value is in float. Most physical scaling's are linked to this define.

Supply voltage

```
#define USER_VDC_LINK_V                                 (24.0f)
```

- This macro defines the nominal DC voltage in volts, used in the motor inverter board. It is used for scaling and limit check +-20% (default) for over-voltage and under-voltage.

```
#define USER_DC_LINK_DIVIDER_RATIO                      (5.1f/(5.1f+47.0f))
//In XMC4400 (5.6f)/(5.6f+56.0f)
```

- See **chapter 2.6**. The DC link voltage divider ratio is R2/(R1+R2).
- This value influences the scaling of the ADC results.

Output, Bridge Driver and B6 Bridge

```
#define USER_DEAD_TIME_US                               (0.75f)
```

- This macro defines the dead-time in microseconds. This value has to be defined according to the switches and bridge drivers. A too small value leads to a short cut. A high value reduces the maximum voltage that can be applied. In default settings the same dead-time is applied to the rising and falling edge. If not compensated for, the dead-time adds a constant error.

```
#define USER_BOOTSTRAP_PRECHARGE_TIME_MS                (20U)
```

- This is the initial bootstrap capacitor pre-charging time in milliseconds. Depending on the driver and driver circuit this time needs to be adapted. Some drivers have implemented a bootstrap charge pump and do not require a bootstrap. If the time is to short the high-side switches will not turn on for the first cycles. There is no drawback if the time is too long except that the time between start request and start is delayed by the bootstrap time.

```
#define CCU8_INPUT_TRAP_LEVEL
        XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_LOW
```

- Define the CCU8 input trap signal logic level according to the gate driver fault signal active level.

Options:

- `XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_LOW`
- `XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_HIGH`

```
#define GATE_DRIVER_INPUT_LOGIC                          PASSIVE_LOW
```

- This macro defines the logic level of the gate driver. Out of this definition the MOTOR_COASTING_xx and MOTOR_RUN_xx configuration are defined.

Options:

- PASSIVE_HIGH
- PASSIVE_LOW

```
#define INVERTER_ENABLE_PIN                              (1U)
```

- This macro defines the logic of the inverter pin.

Options:

- 1 = Active high
- 0 = Active low

Current measurement

```
#define INTERNAL_OP_GAIN                                 DISABLED
```

- The XMC VADC has an internal gain. If this is used, no external OP is required. This configuration enables or disables the internal ADC gain. The current measurement configuration changes based on this configuration.

```
#define USER_R_SHUNT_OHM                                 (0.05f)
```

- Current shunt resistance in ohm. Value in DC link measurement or value per phase in leg shunt measurement. This value is used to calculate I_MAX which is later used to limit the reference current in torque control mode and other FOC functions such as angle estimation.

```
#define USER_DC_SHUNT_OHM                                (0.05f)
```

- DC link current shunt resistance in ohms.
- This value is used for over-current protection with a DC link shunt.

Current measurement: internal OP enabled

```
#define OP_GAIN_FACTOR                                   (3U)
```

- If internal ADC channel gain factor is enabled, the definition of the OP_GAIN_FACTOR is according to the XMC built-in gain factor: 1, 3, 6, and 12.
- It is mandatory that only hardware available gain factors are used.

```
#define USER_RIN_OFFSET_KOHM                             (2.0f)
```

- Offset resistor in kilo Ω as a floating number.
- The purpose is explained in **Figure 37** Phase current amplifier with On-chip gain.

```
#define USER_RIN_PULL_UP_KOHM                            (10.0f)
```

- Offset resistor in kilo Ω as a floating number.
- The purpose is explained in **Figure 37** Phase current amplifier with On-chip gain.

Current measurement: internal OP disabled

```
#define USER_RIN_PHASECURRENT_KOHM                    (1.0f)
```

- This is the RIN resistor value in the phase current amplifier (see **Figure 36)**.
- The unit is in kilo Ω.

```
#define USER_R_PHASECURRENT_FEEDBACK KOHM             (16.4f)
```

- This is the feedback resistor value in the phase current amplifier.
- The unit is in kilo Ω.
- With the USER_RIN_PHASECURRENT_KOHM, the gain of the current amplifier is calculated.
- See **Figure 36**.

```
#define USER_RIN_DCCURRENT_KOHM                       (10.0f)
```

- This is the RIN resistor value in the DC link current amplifier.
- The unit is in kilo Ω.
- See **Figure 36**.

```
#define USER_R_DCCURRENT_FEEDBACK KOHM                (75.0f)
```

- This is the feedback resistor value in the DC link current amplifier.
- The unit is in kilo Ω.
- With the USER_RIN_DCCURRENT_KOHM, the gain of the DC current amplifier is calculated.
- See **Figure 36.**

7.2.3 Motor specific configuration

7.2.3.1 Motor parameter

```
#define USER_MOTOR_R_PER_PHASE_OHM                 (6.8f)
```

- Define the motor phase to neutral resistance in ohms.

```
#define USER_MOTOR_L_PER_PHASE_uH                  (3865.0f)
```

- Define the motor phase to neutral stator inductance in micro henry.
- For IPMSM (Interior Permanent Magnet Synchronous Motor) brushless DC motor, q-axis inductance (Lq) of one motor phase is used.

```
#define USER_MOTOR_POLE_PAIR                       (4U)
```

- Number of pole pairs in the motor, used to calculate the electrical RPM of the rotor.

```
#define USER_SPEED_HIGH_LIMIT_RPM                  (4530.0f)
```

- This value is used as the maximum allowed target speed. Additional control parameters are calculated from this value. The motor Nominal speed should be used.

```
#define USER_SPEED_LOW_LIMIT_RPM             (USER_SPEED_HIGH_LIMIT_RPM/30)
```

- This value is used as minimum allowed target speed. In sensor-less motor control it is mandatory to measure a phase current. At low torque (usually at low speed) it is not possible to provide a sufficient motor control. The minimum speed is application dependent. A default 30% of the high speed limit is configured.

```
#define USER_SPEED_RAMPUP_RPM_PER_S                (500U)
#define USER_SPEED_RAMPDOWN_RPM_PER_S              (500U)
```

- To ensure smooth control a ramp generator is implemented between target input and PI controller. This configuration defines the maximum ramp-up and ramp-down in RPM/sec.

```
#define PWM_THRESHOLD_USEC                         (2U)
```

- Minimum threshold for current measurement in 3 leg shunt measurement mode. If this value is exceeded, the 2 leg shunt measurement is used. This threshold includes the current ringing and ADC measurement time.

```
#define USER_STARUP_SPEED_RPM                      (0U)
```

- Define the initial speed for V/f open loop in RPM.

```
#define USER_STARTUP_SPEED_THRESHOLD_RPM           (500U)
```

- Define the threshold speed to transit from open loop control to closed loop control.

```
#define USER_STARTUP_VF_OFFSET_V                   (1.0f)
```

- V/f open loop control startup voltage offset.

```
#define USER_STARTUP_VF_SLEWRATE_V_PER_HZ          (0.1f)
```

- V/f open loop control startup slew rate in volts per Hz.

```
#define USER_ROTOR_PREPOSITION_TIME_MS                (100)
```

- Time in milliseconds, for motor pre-positioning.

7.2.3.2 PI settings

The default PI settings of each motors are stored in "..\PMSM_FOC\Configuration\Motors". For the "Maxon motor 267121" used in the Infineon XMC1000 Motor Control Application Kit, KIT_XMC1X_AK_MOTOR_001, the file pmsm_foc_motor_MAXON_MOTOR_267121.h contains related configurations.

PI settings for the speed controller

```
#define PI_SPEED_KP                                   (1U << 15)
```

- Configure proportional gain for speed control block.
- Minimum value is 1.
- Maximum is 32767.

```
#define PI_SPEED_KI                                   (3U)
```

- Configure integral gain for speed control block.
- Minimum value is 0.
- Maximum is 32767.

```
#define PI_SPEED_SCALE_KPKI                           (10U + USER_RES_INC)
```

- Configure scaling factor of K_P and K_I.
- K_P and K_I are scaled-up by 2^ PI_SPEED_SCALE_KPKI.
- The USER_RES_INC is a constant defined in pmsm_foc_const.h file.
- The maximum value of PI_SPEED_SCALEKPKI is 15.
- The minimum value is USER_RES_INC.

```
#define PI_SPEED_IK_LIMIT_MIN                         (-(((1<<15)*3)>>2))
```

- Configure minimum output value of the integral buffer.
- Minimum value is -32768

```
#define PI_SPEED_IK_LIMIT_MAX                         (((1<<15)*3)>>2)
```

- Configure maximum output value of the integral buffer.
- Maximum value is 32767

```
#define PI_SPEED_UK_LIMIT_MIN                         (16U)
```

- Configure minimum output value of the speed PI control block.
- Minimum value is -32768.

```
#define PI_SPEED_UK_LIMIT_MAX                         (32767U)
```

- Configure maximum output value of the speed PI control block.
- Maximum value is 32767.

```
#define USER_RES_INC                                        (3U)
```

- This define increase the calculation resolution for the angle. It should never exceed 8U and PI_SPEED_SCALE_KPKI should not exceed 15U.

PI settings for the torque controller

```
#define PI_TORQUE_KP                                        (USER_DEFAULT_IQID_KP)
```

- Configure proportional gain for torque control block.
- The gain value is determined by the equation $K_P = \omega_c L \times A$ (see **chapter 2.11**).

```
#define PI_TORQUE_KI
        (USER_DEFAULT_IQID_KI)
```

- Configure integral gain for torque control block.
- The gain value is determined by the equation $K_I = RT_S K_P / L$ (see **chapter 2.11**).

```
#define PI_TORQUE_SCALE_KPKI                                (SCALING_CURRENT_KPKI)
```

- Configure scaling factor of K_P and K_I.
- The K_P and K_I are scaled-up by 2^ PI_TORQUE_SCALE_KPKI.

```
#define PI_TORQUE_IK_LIMIT_MIN                              (-32768)
```

- Configure minimum output value of the integral buffer.
- Minimum value is -32768.

```
#define PI_TORQUE_IK_LIMIT_MAX                              (32767)
```

- Configure maximum output value of the integral buffer.
- Maximum value is 32767.

```
#define PI_TORQUE_UK_LIMIT_MIN                              (-32768)
```

- Configure minimum output value of the torque PI control block.
- Minimum value is -32768.

```
#define PI_TORQUE_UK_LIMIT_MAX                              (32767)
```

- Configure maximum output value of the torque PI control block.
- Maximum value is 32767.

PI settings for the flux controller

```
#define PI_FLUX_KP                                          (USER_DEFAULT_IQID_KP)
```

- Configure proportional gain for flux control block.
- The gain value is determined by the equation $K_P = \omega_c L \times A$ (see **chapter 2.11**).

```
#define PI_FLUX_KI                                   (USER_DEFAULT_IQID_KI >> 0)
```

- Configure integral gain for flux control block.
- The gain value is determined by the equation $K_I = RT_S K_P / L$ (see **chapter 2.11**).

```
#define PI_FLUX_SCALE_KPKI                    (SCALING_CURRENT_KPKI + 0)
```

- Configure scaling factor of K_P and K_I.
- The K_P and K_I are scaled-up by 2^ PI_FLUX_SCALE_KPKI.

```
#define PI_FLUX_IK_LIMIT_MIN                          (-32768)
```

- Configure minimum output value of the integral buffer.
- Minimum value is -32768.

```
#define PI_FLUX_IK_LIMIT_MAX                          (32767)
```

- Configure maximum output value of the integral buffer.
- Maximum value is 32767.

```
#define PI_FLUX_UK_LIMIT_MIN                          (-32768)
```

- Configure minimum output value of the flux PI control block.
- Minimum value is -32768.

```
#define PI_FLUX_UK_LIMIT_MAX                          (32767)
```

- Configure maximum output value of the flux PI control block.
- Maximum value is 32767.

PI settings for the PLL Estimator controller

```
#define PI_PLL_KP                                     (1U << 8)
```

- Configure proportional gain for PLL Estimator control block.
- Minimum value is 1.
- Maximum is 32767.

```
#define PI_PLL_KI                                     (1U << 6)
```

- Configure integral gain for PLL Estimator control block.
- Minimum value is 0.
- Maximum is 32767.

```
#define PI_PLL_SCALE_KPKI                             (19U - USER_RES_INC)
```

- Configure scaling factor of K_P and K_I.
- The K_P and K_I are scaled-up by 2^ PI_PLL_SCALE_KPKI.
- The USER_RES_INC is a constant defined in pmsm_foc_feature_config.h file.
- The minimum value of PI_PLL_SCALEKPKI is 15.

```
#define PI_PLL_IK_LIMIT_MIN               (-(1 << (30 - PI_PLL_SCALE_KPKI)))
```

- Configure minimum output value of the integral buffer.
- Minimum value is -32768.

```
#define PI_PLL_IK_LIMIT_MAX                    (1 << (30 - PI_PLL_SCALE_KPKI))
```

- Configure maximum output value of the integral buffer.
- Maximum value is 32767.

```
#define PI_PLL_UK_LIMIT_MIN                    (SPEED_LOW_LIMIT >> 4)
```

- Configure minimum output value of the PLL Estimator PI control block.
- Minimum value is -32768.

```
#define PI_PLL_UK_LIMIT_MAX            (SPEED_HIGH_LIMIT + SPEED_LOW_LIMIT)
```

- Configure maximum output value of the PLL Estimator PI control block.
- Maximum value is 32767.

8 PMSM FOC software data structure

8.1 FOC control module input data structure

This data structure defines the input variables used in the FOC functions.

Table 24 FOCInput data structure

Category (Structure)	Variable name	Data type / Range	Description	Remark
FOCInput	Phase_L	signed 32-bit	Motor inductance per phase	Motor parameters Initialize once only
	Phase_R	signed 32-bit	Motor resistance per phase	
	Phase_L_Scale	16-bit / 8 – 18	Scaling for inductance; Real inductance in SW = Phase_L/($2^{Phase\_L\_Scale}$)	
	CCU8_Period	16-bit	CCU8 period. CCU8 PWM 1KHz to 90KHz	Initialize once only
	Res_Inc	16-bit / 0 – 8	Resolution increase, SW uses (16+Res_Inc) bit to represent 360°	
	LPF_N_BEMF	16-bit / 0 – 8	LPF factor for BEMF	
	Threshold	32-bit	BEMF threshold for transitions to FOC	
	Threshold_LOW	16-bit / 0 – 512	LOW BEMF threshold, for transitions to FOC	
	Threshold_HIGH	16-bit / 0 – 512	HIGH BEMF threshold, for transitions to FOC	
	Flag_State	16-bit / 0 or 1	0: Motor to run in FOC 1: Always in transition state	
	BEMF1	signed 32-bit	BEMF of sensorless estimator	
	BEMF2	signed 32-bit	BEMF of sensorless estimator	
	overcurrent_factor	16-bit / 0-4096	Used to reduce motor speed when over-current is detected	
	Ref_Speed	signed 32-bit	Motor reference speed for Speed PI controller	
	Vq_Flag	16-bit / 0 or 1	0: FOC Vq from Iq PI controller 1: Vq from external	
	Vq	signed 32-bit / 0 – 32767	Vq input from external, e.g.: handler of e-bike	
	Ref_Id	signed 32-bit	Id reference for Id PI controller	Default = 0
	Ref_Iq	signed 32-bit	Reference of Iq PI controller from external	
	Iq_PI_Flag	16-bit / 0 or 1	0: Reference of Iq PI controller from speed PI output 1: Reference of Iq PI controller = Ref_Iq from external	

8.2 FOC control module output data structure

This data structure defines the output variables in the FOC functions.

Table 25 FOCOutput data structure

Category (Structure)	Variable name	Data type / Range	Description
FOCOutput	Rotor_PositionQ31	signed 32-bit	Rotor angle feedback, from the sensorless estimator
	Speed_By_Estimator	signed 32-bit	Rotor speed feedback, from the sensorless estimator
	Previous_SVM_SectorNo	16-bit / 0 – 5	SVM sector number in previous PWM cycle
	New_SVM_SectorNo	16-bit / 0 – 5	SVM sector number in current PWM cycle

8.3 FOC control module data type

The table below shows the data structures of the variables used in the FOC control functions.

Table 26 Data structures used in FOC functions

Category (Structure)	Variable name	Data type / Range	Description
Current	I_U	signed 16-bit / 1Q15 data format	Current of Iu current sensing
	I_V	signed 16-bit / 1Q15 data format	Current of Iv current sensing
	I_W	signed 16-bit / 1Q15 data format	Current of Iw current sensing
Clarke_Transform	I_Alpha_1Q31	signed 16-bit / 1Q15 data format	Current I_Alpha, output of Clarke Transform
	I_Beta_1Q31	signed 16-bit / 1Q15 data format	Current I_Beta, output of Clarke Transform
Park_Transform	Id	signed 16-bit / 1Q15 data format	Current Id, output of Park Transform
	Iq	signed 16-bit / 1Q15 data format	Current Iq, output of Park Transform
Car2Polar	Flux_Vd	signed 32-bit	Voltage Vd, output of PI Flux controller
	Torque_Vq	signed 32-bit	Voltage Vq, output of PI Torque controller
	Vref32	32-bit	SVM voltage magnitude of last PWM cycle
	Vref_AngleQ31	signed 32-bit	SVM voltage angle of last PWM cycle
	SVM_Vref16	16-bit	SVM voltage magnitude in open loop control
	SVM_Angle16	16-bit	SVM angle in open loop control

8.4 SVM module data structure

This data structure defines the variables used in the SVM module.

Table 27 Data structure used in SVM module

Category (Structure)	Variable name	Data type / Range	Description	Remark
SVM	CurrentSectorNo	16-bit / 0 – 5	SVM new sector number, 0 to 5 represent sector A to F	
	PreviousSectorNo	16-bit / 0 – 5	To keep track of sector number in last PWM cycle	
	Flag_3or2_ADC	16-bit / 0 or 0xBB	To indicate using 2 or 3 ADC results of phase currents for current construction 0: USE ALL ADC 0xBB: USE 2 ADC	For three shunt current sensing technique
	SVM_Flag	16-bit / 0 or 0xAD	To indicate using SVM with PZV or standard 4-segment SVM 0: PZV 0xAD: Standard 4-seg SVM	For single shunt current sensing technique

8.5 Get current software module data structure

This data structure defines the variables used in the Get Current module where the phase current ADC results are read.

Table 28 Data structure used in get current module

Category (Structure)	Variable Name	Data Type / Range	Description	Remark
ADC	ADC_Iu	16-bit / 0 – 4095	Store phase U current ADC result	Three shunt current sensing technique
	ADC_Iv	16-bit / 0 – 4095	Store phase V current ADC result	
	ADC_Iw	16-bit / 0 – 4095	Store phase W current ADC result	
	ADC_Bias_Iu	16-bit / 0 – 4095	Bias of ADC_Iu	
	ADC_Bias_Iv	16-bit / 0 – 4095	Bias of ADC_Iv	
	ADC_Bias_Iw	16-bit / 0 – 4095	Bias of ADC_Iw	
	ADCTrig_Point	16-bit / 0 – CCU8 Slice 3 period value	VADC trigger position; This is compare match value for CCU8 slice 3 timer	
	ADC_Bias	16-bit / 0 – 4095	Bias of single shunt current input	

Category (Structure)	Variable Name	Data Type / Range	Description	Remark
	ADC_ResultTz1	16-bit / 0 – 4095	ADC value of the first phase current ADC sampling	Single shunt current sensing technique
	ADC_ResultTz2	16-bit / 0 – 4095	ADC value of the second phase current ADC sampling	
	ADC_Result3	16-bit / 0 – 4095	ADC value of the third phase current ADC sampling	
	ADC_Result4	16-bit / 0 – 4095	ADC value of the fourth phase current ADC sampling	
	ADC_Result1	signed 32-bit	Two most critical Phase current results for current reconstruction	
	ADC_Result2	signed 32-bit		
	ADC3Trig_Point	16-bit / 0 to ADC4Trig_Point	VADC trigger position; This is compare match value for CCU8 slice 3 timer	
	ADC4Trig_Point	16-bit / ADC3Trig_Point+1 to CCU8 Slice3 period value	VADC trigger position; This is compare match value for CCU8 slice 3 timer	
	Result_Flag	16-bit / 0 – 2	To indicate ADC result is for first CCU8 slice 3 cycle or second CCU8 slice 3 cycle or for standard 4-segment SVM: 0: First cycle, PZV 1: Second cycle, PZV 2: Standard 4-seg SVM	
	ADC_POT	0 – 4095	Store ADC result of potentiometer	
	ADC_DCLink	0 – 4095	Store ADC result of DC link voltage	
	ADC_IDCLink	0 – 4095	Store ADC result of average DC link current	

9 PMSM FOC software API functions

In this chapter the PMSM FOC software API functions are documented. The APIs are grouped into User Functions and Controller APIs.

To improve the performance and to reduce the CPU loading, most of the time critical APIs are executed in the SRAM. The table below shows the list of APIs that are executed in SRAM.

Table 29 List of APIs

Type		API function name	Execute in SRAM	Execute in Flash
User Functions		pmsm_foc_motor_start()		x
		pmsm_foc_motor_stop()		x
		pmsm_foc_motor_brake ()		x
		pmsm_foc_set_motor_target_speed ()		x
		pmsm_foc_set_motor_target_torque ()		x
		pmsm_foc_set_motor_target_voltage ()		x
		pmsm_foc_get_motor_speed ()		x
Interrupt service routine		pmsm_foc_controlloop_isr()	x	
		pmsm_foc_secondaryloop_isr() pmsm_foc_over_under_voltage_isr()	x	x
Controlloop ISR	Startup	pmsm_foc_bootstrap_charge()		x
		pmsm_foc_enable_inverter()		x
		pmsm_foc_disable_inverter()		x
		pmsm_foc_directfocrotor_pre_positioning()		x
		pmsm_foc_vf_foc_openloop_rampup()		x
		pmsm_foc_vf_smooth_transition_to_foc()		x
		pmsm_foc_misc_works_of_met()		x
	InOut handling	pmsm_foc_get_IDCLink_current()	x	
		pmsm_foc_linear_ramp_generator()	x	
		pmsm_foc_linear_torque_ramp_generator()	x	
		pmsm_foc_Linear_vq_ramp_generator()	x	
		pmsm_foc_scurve_ramp_generator()	x	
		pmsm_foc_svpwm_update()	x	
		pmsm_foc_adc34_triggersetting()	x	
		pmsm_foc_adctz12_triggersetting()	x	
		pmsm_foc_get_adcphasecurrent()	x	
	General	pmsm_foc_error_handling()		x
		pmsm_foc_motor_hold()		x
		pmsm_foc_misc_works_of_foc()	x	
	Controller	pmsm_foc_speed_controller()	x	
		pmsm_foc_torque_controller()	x	
		pmsm_foc_vq_controller()	x	
Secondaryloop ISR		pmsm_foc_misc_works_of_irq()	x	
		XMC_WDT_Service()	x	
		pmsm_foc_secondaryloop_callback()		x
Over_under_voltage_isr		pmsm_foc_disable_inverter()		x

9.1 User Functions

User functions are intended to be called by external users. They are the interface to other applications.

pmsm_foc_motor_start ()

This API is called to start the motor.

If the Motor.State is TRAP_PROTECTION the trap is cleared and the MCU is reinitialized.

Table 30 pmsm_foc_motor_start()

Input Parameters	-	
Return	-	
Updated Variables	Motor.State	EN_INVERTER_BOOTSTRAP

pmsm_foc_motor_stop ()

This API is called to disable the inverter and set all output pins in tristate. This will lead to an uncontrolled freewheeling. For a controlled ramp-down you should use the pmsm_foc_set_motor_target_speed/torque.

- If the Motor.State is TRAP_PROTECTION the trap is cleared and the MCU is re-initialized.

Table 31 pmsm_foc_motor_stop()

Input Parameters	-	
Return	-	
Updated Variables	Motor.State	MOTOR_STOP

pmsm_foc_motor_brake ()

This API is called to ramp-down the motor.

Table 32 pmsm_foc_motor_stop()

Input Parameters	-	
Return	-	
Updated Variables	Motor.State	FOC_CLOSED_LOOP_BRAKE if state isn’t TRAP_PROTECTION

pmsm_foc_set_motor_target_speed ()

This API is called to set the target speed. The scaling is application specific and the input parameter is limited by SPEED_HIGH_LIMIT and SPEED_LOW_LIMIT. To stop the motor pmsm_foc_motor_stop() function should be used. The MCU will approach the reference speed to the target speed using the ramp generator.

This function is only available if SETTING_TARGET_SPEED is configured with SET_TARGET_SPEED and MY_FOC_CONTROL_SCHEME is SPEED_CONTROLLED_VF_MET_FOC or SPEED_CONTROLLED_DIRECT_FOC.

Table 33 pmsm_foc_set_motor_target_speed ()

Input Parameters	(uint32)motor_target_speed	
Return	-	
Updated Variables	Motor.Target_Speed	

pmsm_foc_set_motor_target_torque ()

This API is called to set the target speed. The scaling is application specific and the input parameter is limited by USER_IQ_REF_HIGH_LIMIT and USER_IQ_REF_LOW_LIMIT. To stop the motor pmsm_foc_motor_stop() function should be used. The MCU will approach the reference speed to the target speed using the ramp generator.

This function is only available if SETTING_TARGET_SPEED is configured with SET_TARGET_SPEED and MY_FOC_CONTROL_SCHEME is TORQUE_CONTROLLED_DIRECT_FOC.

Table 34 pmsm_foc_set_motor_target_torque ()

Input Parameters	(uint32) motor_target_torque	
Return	-	
Updated Variables	Motor. Target_Torque	

pmsm_foc_set_motor_target_voltage ()

This API is called to set the target speed. The scaling is application specific and the input parameter is limited by USER_VQ_REF_HIGH_LIMIT and USER_VQ_REF_LOW_LIMIT. To stop the motor pmsm_foc_motor_stop() function should be used. The MCU will approach the reference speed to the target speed using the ramp generator.

This function is only available if SETTING_TARGET_SPEED is configured with SET_TARGET_SPEED and MY_FOC_CONTROL_SCHEME is VQ_CONTROLLED_DIRECT_FOC.

Table 35 pmsm_foc_set_motor_target_voltage ()

Input Parameters	(uint32) motor_target_voltage	
Return	-	
Updated Variables	Motor.Target_Voltage	

pmsm_foc_get_motor_speed ()

This API returns the motor speed in RPM based on Motor.Speed.

Table 36 pmsm_foc_get_motor_speed ()

Input Parameters	-	
Return	Speed in RPM	
Updated Variables	-	

pmsm_foc_get_Vdc_link ()

This API returns the voltage value of DC bus.

Table 37 pmsm_foc_get_Vdc_link ()

Input Parameters	-	
Return	Voltage in V	
Updated Variables	-	

9.2 Controlloop ISR

9.2.1 Startup

The following list of APIs are executed for the direct FOC startup.

pmsm_foc_bootstrap_charge ()

The function of this API is to perform a motor brake and charge the gate driver bootstrap capacitor. At the same time the motor phase currents bias are read and updated.

This API is called in the EN_INVERTER_BOOTSTRAP motor state.

Table 38 pmsm_foc_bootstrap_charge ()

Input parameters	None	-
Return	None	-
Updated variables	ADC.ADC_Bias_Iu	ADC bias value for phase U current
	ADC.ADC_Bias_Iv	ADC bias value for phase V current
	ADC.ADC_Bias_Iw	ADC bias value for phase W current
	Motor.State	Updated motor state to PRE_POSITIONING once motor start command is received
	Motor.Transition_Status	Motor status flag for transition; Change this flag to MOTOR_TRANSITION once motor start command is received

pmsm_foc_enable_inverter ()

This function enables the inverter pin and connects the PWM GPIOs to the PWM signal.

Table 39 pmsm_foc_enable_inverter ()

Input Parameters	None	-
Return	None	-
Updated Variables	None	-

pmsm_foc_disalbe_inverter ()

This function disables the inverter pin and sets the PWM GPIOs to tristate.

Table 40 pmsm_foc_disable_inverter ()

Input Parameters	None	-
Return	None	-
Updated Variables	None	-

pmsm_foc_directfocrotor_pre_positioning ()

This API is to set the initial rotor position/alignment. It is called in the PRE_POSITIONING motor state. It initializes the PI controller variables and FOCInput, and changes the motor state to FOC_CLOSED_LOOP.

Table 41 pmsm_foc_directfocrotor_pre_positioning ()

Input parameters	None	-
Return	None	-
Updated variables	Current.I_u, Current.I_v, Current.I_w	Current reconstruct function is called and I_u, I_v and I_w updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	I_Alpha and I_Beta updated
	Car2Polar.Vref	Increase Vref value gradually for SVM
	Motor.State	Updated motor state to FOC_CLOSED_LOOP once preposition timer expired
	FOCInput	Initialized FOCInput variables before entering FOC_CLOSED_LOOP
	PI_Speed PI_Torque PI_Flux PI_PLL	Initialized PI controllers variables before entering FOC_CLOSED_LOOP

pmsm_foc_vf_foc_openloop_rampup ()

This API is called in the VFOPENLOOP_RAMP_UP motor state. It sets the initial rotor position/alignment and then ramps-up the motor in open loop. Once the motor speed your defined VF_TRANSITION_SPEED, the motor state is changed to MET_FOC.

Table 42 pmsm_foc_vf_foc_openloop_rampup()

Input Parameters	None	-
Return	None	-
Updated Variables	Current.I_u, Current.I_v, Current.I_w	Current reconstruct function is called and I_u, I_v and I_w updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	I_alpha and I_Beta updated
	Car2Polar.Vref	Increase Vref value gradually for SVM
	Motor.State	Updated motor state to MET_FOC once motor speed ramp-up to a predefined value
	FOCInput	Initialized FOCInput variables before entering MET_FOC
	PLL_Estimator	Called PLL_Estimator API and update variables before entering MET_FOC

pmsm_foc_vf_smooth_transition_to_foc ()

This API is called in the MET_FOC motor state. It is for MET control strategy. Once the stator flux is perpendicular to the rotor flux, return the status as MOTOR_STABLE.

Table 43 pmsm_foc_vf_smooth_transition_to_foc ()

Input Parameters	None	-
Return	Motor. Transition_Status	Motor mode status MOTOR_STABLE: MET is done, can switch to next state MOTOR_TRANSITION: MET function not done
Updated Variables	Current.I_u, Current.I_v, Current.I_w	Current reconstruct function is called and I_u, I_v and I_w updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	Clarke transform function is called and I_alpha and I_Beta updated
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar Vref and Vref_AngleQ31 updated
	FOCInput	FOCInput variables updated
	PLL_Estimator	PLL_Estimator APIs are called and variables are updated

pmsm_foc_misc_works_of_met ()

This routine checks for a motor stop command while waiting for MET state to be finished.

Once the MET is complete (Motor.Mode_Flag == MOTOR_STABLE), it switches the motor state to FOC_CLOSED_LOOP.

Table 44 pmsm_foc_misc_works_of_met()

Input Parameters	Motor.Mode_Flag	Motor mode status
Return	None	-
Updated Variables	Motor.State	Updated motor state to FOC_CLOSED_LOOP once Motor.Mode_Flag == MOTOR_STABLE
	FOCInput	Initialized FOCInput variables before entering FOC_CLOSED_LOOP
	PI_Speed PI_Torque PI_Flux PI_PLL	Initialized PI controllers variables before entering FOC_CLOSED_LOOP
	PLL_Estimator.RotorAngleQ31	Initialize rotor angle for the first FOC PWM cycle before entering FOC_CLOSED_LOOP

9.2.2 InOut handling

pmsm_foc_get_IDCLink_current ()

This functions updates the DC link value in a scaled version.

Table 45 pmsm_foc_get_IDCLink_current ()

Input Parameters	-	
Return	-	
Updated Variables	ADC.ADC_IDCLink	

pmsm_foc_get_adcphasecurrent ()

This API is only used in the three shunt current sensing technique. It reads the ADC results of the 3-phase currents.

If synchronous conversion is used, the VADC alias channels settings are also updated according to the SVM sector (only in XMC1300/XMC1400).

Table 46 pmsm_foc_get_adcphasecurrent ()

Input Parameters	SVM.PreviousSectorNo	SVM sector number in previous PWM cycle
	SVM.CurrentSectorNo	Current SVM sector number
Return	ADC.ADC_Iu	ADC Phase U current result
	ADC.ADC_Iv	ADC Phase V current result
	ADC.ADC_Iw	ADC Phase W current result
Updated Variables	-	

pmsm_foc_linear_ramp_generator ()

In this routine, the linear ramp generator for speed control is implemented.

Table 47 pmsm_foc_linear_ramp_generator()

Input Parameters	set_val	Target speed value
	rampup_rate	Speed ramp-up rate
	rampdown_rate	Speed ramp-down rate
	speedrampstep	Number of steps changed every (rampup_rate or rampdown_rate) x PWM cycles
Return	Motor.Ref_speed	Motor reference speed for PI speed controller
Updated Variables	Motor.Ramp_Up_Rate	Motor speed ramp-up rate
	Motor.Ramp_Counter	Ramp counter

pmsm_foc_linear_torque_ramp_generator ()

In this routine the linear ramp generator for torque control is implemented.

Table 48 pmsm_foc_linear_torque_ramp_generator()

Input Parameters	current_set	Target torque value
	inc_step	Torque ramp-up rate
	dec_step	Torque ramp-down rate
Return	FOCInput.Ref_Iq	Motor reference torque for PI torque controller
Updated Variables	-	

pmsm_foc_linear_vq_ramp_generator ()

In this routine the linear ramp generator for Vq control is implemented.

Table 49 pmsm_foc_linear_vq_ramp_generator()

Input Parameters	set_val	Target Vq value
	inc_step	Vq ramp-up rate
	dec_step	Vq ramp-down rate
Return	FOCInput.Vq	Reference Vq for Cartesian to Polar transformation
Updated Variables	-	

pmsm_foc_scurve_ramp_generator ()

In this routine the S-curve ramp generator for speed control is implemented.

Table 50 pmsm_foc_scurve_ramp_generator()

Input Parameters	set_val	Target speed value
	rampup_rate	Speed ramp-up rate
	rampdown_rate	Speed ramp-down rate
	speedrampstep	Number of steps changed every (rampup_rate or rampdown_rate) x PWM cycles
Return	Motor.Ref_speed	Motor reference speed for PI speed controller
Updated Variables	Motor.Ramp_Up_Rate	Motor speed ramp-up rate
	Motor.Ramp_Dn_Rate	Motor speed ramp-down rate
	Motor.Ramp_Counter	Ramp counter

pmsm_foc_svpwm_update ()

In this API the SVM algorithm is executed and 3-phase PWM duty cycles are updated.

Table 51 pmsm_foc_svpwm_update ()

Input Parameters	Amplitude	Amplitude of voltage space vector
	Angle	Angle of voltage space vector
Return	-	
Updated Variables	SVM.PreviousSectorNo	Backup current SVM sector number
	SVM.CurrentSectorNo	Update current SVM sector number

pmsm_foc_adctz12_triggersetting ()

This API is only called in the single shunt current sensing technique. It is to set the first two trigger points for the ADC conversion. The settings of the trigger points are the pre-calculated constants in the configuration file.

Table 52 pmsm_foc_adctz12_triggersetting()

Input Parameters	-	
Return	-	
Updated Variables	-	

pmsm_foc_adc34_triggersetting ()

This API is only called in the single shunt current sensing technique. This API is to set the third and fourth trigger points for the ADC conversion.

Table 53 pmsm_foc_adc34_triggersetting()

Input Parameters	ADC.ADC3Trig_Point	Third trigger point setting
	ADC.ADC4Trig_Point	Fourth trigger point setting
Return	-	
Updated Variables	-	

9.2.3 General

pmsm_foc_misc_works_of_foc ()

In this routine, if the motor is ramping-up in speed/torque/Vq, the reference speed or reference torque or Vq, is updated based on the ramping rate.

Table 54 pmsm_foc_misc_works_of_foc()

Input Parameters	Motor.Mode_Flag	Motor mode status
Return	None	-
Updated Variables	Motor.State	Update motor state to MOTOR_HOLD if motor stop command received
	Motor.Ref_Speed	Updated if SPEED_CONTROLLED_DIRECT_FOC or SPEED_CONTROLLED_VF_MET_FOC control scheme is selected
	FOCInput.Ref_Iq	Updated if TORQUE_CONTROLLED_DIRECT_FOC control scheme is selected
	FOCInput.Vq	Updated if VQ_CONTROLLED_DIRECT_FOC control scheme is selected

pmsm_foc_error_handling ()

This function handles the return from a TRAP state. You can enter a reaction for a TRAP. One option is to re-initialize the motor control after a period of time. In the default there is no automated return from a TRAP state.

Table 55 pmsm_foc_error_handling ()

Input Parameters		
Return		
Updated Variables	Motor.State	MOTOR_HOLD

pmsm_foc_Clear_trap ()

This function clears the hardware trap state and clears the Motor state. It is called by the motor stop and motor brake in case of a TRAP.

Table 56 pmsm_foc_Clear_trap ()

Input Parameters		
Return		
Updated Variables	Motor.CCU8_Trap_Status	0x00
	Motor.State	MOTOR_IDLE

pmsm_foc_motor_hold ()

This function is used for startup and active freewheeling. In active freewheeling the speed is reduced to zero but the PWM pattern is still updated.

Table 57 pmsm_foc_motor_hold ()

Input Parameters		
Return		
Updated Variables		

9.2.4 Controller

pmsm_foc_speed_controller ()

This API is called if SPEED_CONTROLLED_DIRECT_FOC or SPEED_CONTROLLED_VF_MET_FOC control scheme are selected. It executes the FOC speed control algorithm and FOC calculations. It is called in the FOC_CLOSED_LOOP motor state.

With the MATH coprocessor (in XMC1302, XMC1402 and XMC1404) there is a timing gain of a few microseconds by interleaving the CPU calculation with CORDIC computations (see the flowchart in **Figure 55**).

Table 58 pmsm_foc_speed_controller()

Input Parameters	ADC.ADC_Bias_Iu	ADC bias value for phase U current
	ADC.ADC_Bias_Iv	ADC bias value for phase V current
	ADC.ADC_Bias_Iw	ADC bias value for phase W current
	FOCInput.Ref_Speed	Reference speed value
	SVM.CurrentSectorNo	SVM current sector number
Return	Motor.Speed	Current motor speed from PLL Estimator
Updated Variables	Current.I_u, Current.I_v, Current.I_w	Current reconstruct function is called and I_u, I_v and I_w updated
	Park_Transform.Iq, Park_Transform.Id	Park transform function is called and Iq and Id updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	Clarke transform function is called and I_alpha and I_Beta updated
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar function is called and Vref and Vref_AngleQ31 updated
	PI_Speed PI_Torque PI_Flux PI_PLL	PI controller functions are called and the PI outputs updated
	PLL_Estimator	PLL Estimator APIs are called and the variables updated
	FOCOuput.Speed_by_Estimator	Estimated rotor speed from PLL Estimator
	FOCOutput.Rotor_PositionQ31	Estimated rotor position from PLL Estimator

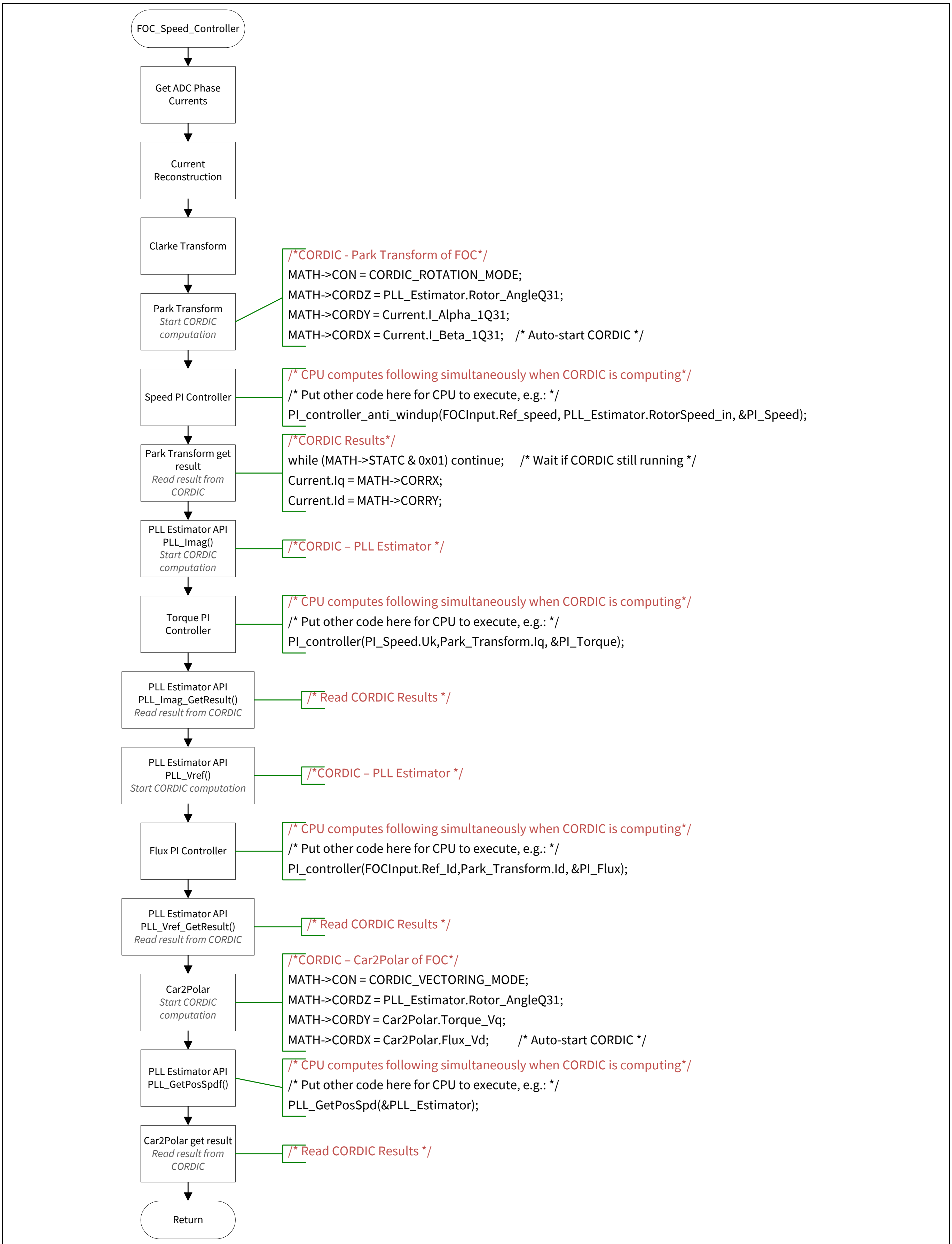


Figure 55 FOC speed control flowchart

pmsm_foc_torque_controller ()

This API is called if the TORQUE_CONTROLLED_DIRECT_FOC control scheme is selected. It executes the FOC torque control and its calculations. It is called in the FOC_CLOSED_LOOP motor state.

Table 59 pmsm_foc_torque_controller ()

Input Parameters	ADC.ADC_Bias_Iu	ADC bias value for phase U current
	ADC.ADC_Bias_Iv	ADC bias value for phase V current
	ADC.ADC_Bias_Iw	ADC bias value for phase W current
	FOCInput.Ref.Iq	Reference torque value
	SVM.CurrentSectorNo	SVM current sector number
Return	Motor.Speed	Current motor speed from PLL Estimator
Updated Variables	Current.I_u, Current.I_v, Current.I_w	Current reconstruct function is called and I_u, I_v and I_w updated
	Park_Transform.Iq, Park_Transform.Id	Park transform function is called and Iq and Id updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	Clarke transform function is called and I_Alpha and I_Beta updated
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar function is called and Vref and Vref_AngleQ31 updated
	PI_Torque PI_Flux PI_PLL	PI controller functions are called and the PI outputs updated
	PLL_Estimator	PLL Estimator APIs are called and the variables updated
	FOCOuput.Speed_by_Estimator	Current motor speed from PLL Estimator
	FOCOutput.Rotor_PositionQ31	Current motor position from PLL Estimator

pmsm_foc_vq_controller (void)

This API is called if the VQ_CONTROLLED_DIRECT_FOC control scheme is selected. It executes the FOC VQ control and FOC calculations. It is called in the FOC_CLOSED_LOOP motor state.

Table 60 pmsm_foc_vq_controller(void)

Input Parameters	ADC.ADC_Bias_Iu	ADC bias value for phase U current
	ADC.ADC_Bias_Iv	ADC bias value for phase V current
	ADC.ADC_Bias_Iw	ADC bias value for phase W current
	FOCInput.Vq	Reference Vq value
	SVM.CurrentSectorNo	SVM current sector number
Return	Motor.Speed	Current motor speed from PLL Estimator
Updated Variables	Current.I_u, Current.I_v, Current.I_w	Current reconstruct function is called and I_u, I_v and I_w updated
	Park_Transform.Iq, Park_Transform.Id	Park transform function is called and Iq and Id updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	Clarke transform function is called and I_alpha and I_Beta updated
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar function is called and Vref and Vref_AngleQ31 updated
	PI_Flux PI_PLL	PI controller functions are called and the PI outputs updated
	PLL_Estimator	PLL Estimator APIs are called and the variables updated
	FOCOuput.Speed_by_Estimator	Current motor speed from PLL Estimator
	FOCOutput.Rotor_PositionQ31	Current motor position from PLL Estimator

9.3 Secondaryloop ISR

pmsm_foc_misc_works_of_irq ()

This API is called in the secondary loop with a default 1 kHz frequency. It will read the motor speed you have set and will scale it up (see **chapter 2.10** for scaling).

Table 61 pmsm_foc_misc_works_of_irq()

Input Parameters	None	-
Return	None	-
Updated Variables	Motor.Target_Speed	If SETTING_TARGET_SPEED is set to BY_POT_ONLY or BY_UART_ONLY the value is updated. Which value is updated depends on MY_FOC_CONTROL_SCHEME setting.
	Motor.Target_Torque	
	Motor.Target_Voltage	

pmsm_foc_secondaryloop_callback ()

This API is only available if PMSM_FOC_SECONDARYLOOP_CALLBACK is ENABLED. You need to define this function. The secondary loop is placed in the flash to support large functions. To reduce execution time the function can be placed in RAM manually.

Table 62 pmsm_foc_misc_works_of_irq()

Input Parameters	None	-
Return	None	-
Updated Variables	User defined	

9.4 FPU library

All the mathematical functions the field oriented control needs to work are implemented in the FPU library fpu_math2. In particular it also implements the sine, cosine, arctangent, magnitude, and the park transform.

9.4.1 Theory

A good choice to implement the sine, cosine and arctangent is using the lookup table(LUT). The sine, cosine and park transform functions are imported from the CMSIS library with the relative LUT (for support look at CMSIS support manual). For arctangent, it's impossible to use only look up table because of its infinite and not periodic domain, so the library uses a mix of LUT and math approximation. For the arctangent LUT, 300 variable step samples are used in the domain from 0 to a certain value (in library set to 5.02).

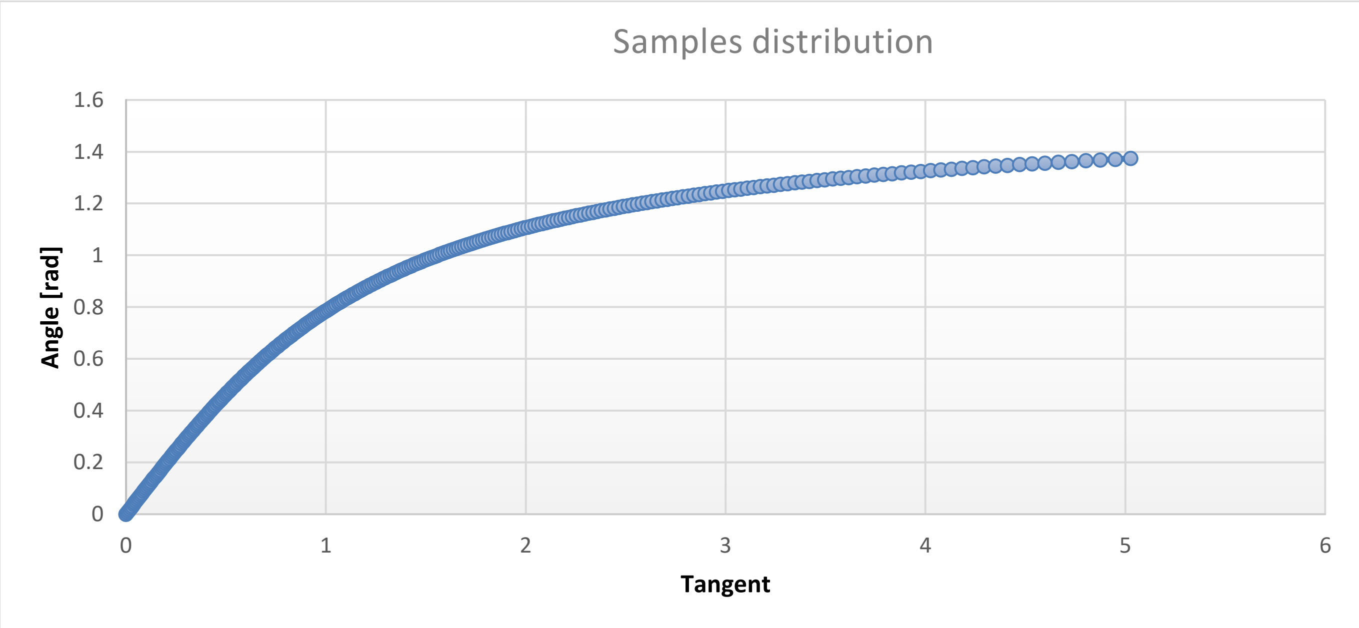


Figure 56 Sample distribution

The mathematical map to convert tangent value to the sample vector index is in relation with the second derivative of the arctangent function, properly scaled and adapted:

$$y = \frac{\left(\frac{x}{AT\_X\_SCALING} + AT\_X\_OFFSET\right)^2}{1 + \left(\frac{x}{AT\_X\_SCALING} + AT\_X\_OFFSET\right)^2} \cdot AT\_Y\_SCALING - AT\_Y\_OFFSET$$

Where x is the tangent value and y is the index vector. The best values for the parameters are defined in the library.
Once the float index value is found, it is used for linear interpolation to find the angle value. The variable step is used for mapping the function with more points where it is more curved. Because of the concavity of the arctangent, the linear interpolation would always return an underestimated value. A constant is added to all the samples to compensate for this phenomenon (`AT_CC_ERROR`).

For tangent larger than last LUT element, library uses the following math approximation:

$$angle = -\frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \frac{\pi}{2}$$

Where x is the tangent and the result is in radiant.

The precision of the function arctangent is about 2E-6 radiants (0.00012 degrees, 20 bit in q31). Library support is also an error correction that brings the precision to 5E-7 radiants (0.00003 degrees, 22 bit in q31). It's mandatory to call fpu_tangent_lookup_table_generation() before using fpu_cart2polar() function.

9.4.2 Library API

fpu_sin_q31 ()

This function calculates the sine of the given angle, both in q31.

Table 63 fpu_sin_q31 ()

Input parameters	Angle	Q31 format
Return	Sin(Angle)	Q31 format
Updated variables	None	

fpu_cos_q31 ()

This function calculates the cosine of the given angle, both in q31.

Table 64 fpu_cos_q31 ()

Input parameters	Angle	Q31 format
Return	Cos(Angle)	Q31 format
Updated variables	None	

fpu_park_q31 ()

This function calculates the park transform for given I_alpha, I_beta, and sine/cosine of rotor angle.

Table 65 fpu_park_q31 ()

Input parameters	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	Y and X components of any physical quantity, in this case, Clarke transform Currents.
	Angle_sine Angle_cosine	The sine and cosine of the rotation angle, in this case, rotor angle.
Return	Park_Transform.Iq, Park_Transform.Id	Y and X rotated components, in this case, Id and Iq.
Updated variables	None	

fpu_cart2polar ()

This function calculates the polar coordinates starting from the cartesian coordinates of any physical quantity, then it adds the offset angle to the polar one. Before using, it's mandatory to call fpu_tangent_lookup_table_generation().

Table 66 fpu_cart2polar ()

Input parameters	Car2Polar.Flux_Vd Car2Polar.Torque_Vq	Y and X components of any physical quantity, in this case, flux and torque components.
	Rotor Angle	An angle added to the polar one. In this case, rotor angle.
Return	Car2Polar.Vref32 Car2Polar.Vref_AngleQ31	Polar components of the physical quantity.
Updated variables	None	

fpu_tangent_lookup_table_generation()

This function calculates the samples for arctangent lookup table.

Table 67 fpu_cart2polar ()

Input parameters	None	
Return	None	
Updated variables	Arctangent lookup table	

Value for arctangent

```
#define    AT_X_SCALING          2
#define    AT_X_OFFSET           0.675f
#define    AT_Y_SCALING          500.467269f
#define    AT_Y_OFFSET           156.6511975f
#define    AT_ERROR_CORRECTION   1
```
— Enable error correction
```
#define    AT_LAST_LUT_TANGENT   5.02719008f
```
— Last element of the LUT table
```
#define    FPU_PI                3.1415926535f
#define    FPU_HALF_PI           1.57079632679f

#define    AT_CC_ERROR           0.000000261111f
```
— Offset of table values

10 Resources

- Infineon XMC1000 Motor Control Application Kit document: XMC1000 Motor Control Application Kit.
- Examples can be found at XMC1000.
- AP32370 - XMC1000 - PMSM FOC motor control software using XMC™.
- Infineon XMC1000 Motor Control Application Kit document: XMC4400 motor drive card.

11 Revision history

Document version	Date	Description of changes
1.5	2018-12-31	Updated this document according to the PMSM_FOC software version 1.5.x.
		Chapter 1.6: New structure of Software overview.
		Chapter 3.2: Improved description of 3 shunt measurement and clear separation of Synchronous and Asynchronous conversion.
		Chapter 7: New structure for configuration. Adapt to v1.5.x
		Chapter 9: New user functions.
		XMC1400 added.
		Kits added.
1.6	2023-06-15	Added XMC4400 support.

Trademarks of Infineon Technologies AG
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-06-15

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference: AP32370