

PMSM FOC motor control using XMC™ MCUs

XMC1300/XMC1400 and XMC4400/XMC4800

About this document

Scope and purpose

This application note describes the implementation of permanent magnet synchronous motor (PMSM) field oriented control (FOC) motor control software for a 3-phase motor using the Infineon XMC1302, XMC1402, XMC1404, XMC4400, or XMC4800 microcontroller.

Intended audience

This document is intended for customers who want to design 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](#)
- [4] [XMC4800 Reference Manual](#)

Table of contents

About this document.....	1
Table of contents.....	2
1 Introduction	4
1.1 Key features.....	5
1.2 Abbreviations and acronyms	6
1.3 XMC™ resource allocation.....	7
1.4 Interconnectivity of XMC™ hardware modules	9
1.5 Execution time and memory usage	11
1.6 Software overview.....	12
1.7 Limitations of use for PMSM FOC software	14
2 PMSM FOC sensorless software components	15
2.1 Motor start/speed change/motor stop operations control.....	16
2.2 Ramp generator.....	17
2.3 Control schemes.....	18
2.3.1 Open-loop voltage control.....	18
2.3.2 Speed control	18
2.3.3 Torque control direct startup	20
2.3.4 Vq control direct startup.....	21
2.3.5 D-axis and Q-axis decoupling	21
2.4 Cartesian-to-Polar Transform.....	22
2.5 Space vector modulation (SVM)	23
2.5.1 7-segment SVM.....	24
2.5.2 5-segment SVM.....	25
2.5.3 Pseudo zero vector (PZV).....	26
2.5.4 4-segment SVM.....	27
2.5.5 Overmodulation SVM	29
2.6 DC link voltage.....	30
2.7 Clarke Transform.....	30
2.8 Park Transform.....	32
2.9 Protection.....	33
2.10 Scaling	33
2.11 Determination of flux and torque current PI gains	37
3 Current sensing and calculation.....	39
3.1 Single-shunt current sensing (only in XMC1300/XMC1400)	41
3.2 Three-shunt current sensing.....	44
3.2.1 Asynchronous conversion (only in XMC1300/XMC1400).....	46
3.2.2 Synchronous conversion	46
3.2.3 Synchronous measurement implementation.....	49
4 Motor speed and position feedback in sensorless FOC control	51
5 Interrupts.....	53
5.1 PWM period match interrupt	53
5.2 CTrap interrupt.....	55
5.3 ADC source interrupt (only in XMC1300/XMC1400)	55
5.4 Secondary loop interrupt.....	55
5.5 Overvoltage/undervoltage protection	55
6 Motor state machine.....	56

Table of contents

7	Configuration	59
7.1	User configuration.....	59
7.1.1	General	59
7.1.2	Custom kit configuration	62
7.1.3	Advanced user configuration.....	64
7.1.4	Torque control specific	65
7.1.5	VQ control specific (V/f).....	66
7.1.6	MET specific.....	66
7.2	Hardware configuration.....	67
7.2.1	Controller card	67
7.2.2	Inverter board configuration for current and voltage sensing.....	74
7.2.3	Motor-specific configuration	77
7.2.3.1	Motor parameters	77
7.2.3.2	PI settings.....	78
8	PMSM FOC software data structure	81
8.1	FOC control module input data structure	81
8.2	FOC control module output data structure.....	81
8.3	FOC control module data type.....	82
8.4	SVM module data structure	83
8.5	Get Current software module data structure.....	83
9	PMSM FOC software API functions.....	85
9.1	User functions	86
9.2	Control loop ISR	89
9.2.1	Startup.....	89
9.2.2	InOut handling	92
9.2.3	General	95
9.2.4	Controller	96
9.3	Secondaryloop ISR.....	99
9.4	FPU library	99
9.4.1	Library theory	99
9.4.2	Library API	101
	Resources	103
	Revision history.....	104
	Disclaimer.....	105

Introduction

1 Introduction

This software offers the functionality to drive a 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 that can easily be controlled in frequency and amplitude to minimize the current. This in turn maximizes the efficiency. The basic idea is to transform three phase signals into two rotor-fix signals and vice-versa.

Feedback on the 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 sensors, such as Hall sensors or an encoder.

Feedback on the phase currents can be measured in the motor phase, 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.

Figure 1 shows the typical block diagram for PMSM FOC motor control, where single-shunt and three-shunt low-side current sensing are supported.

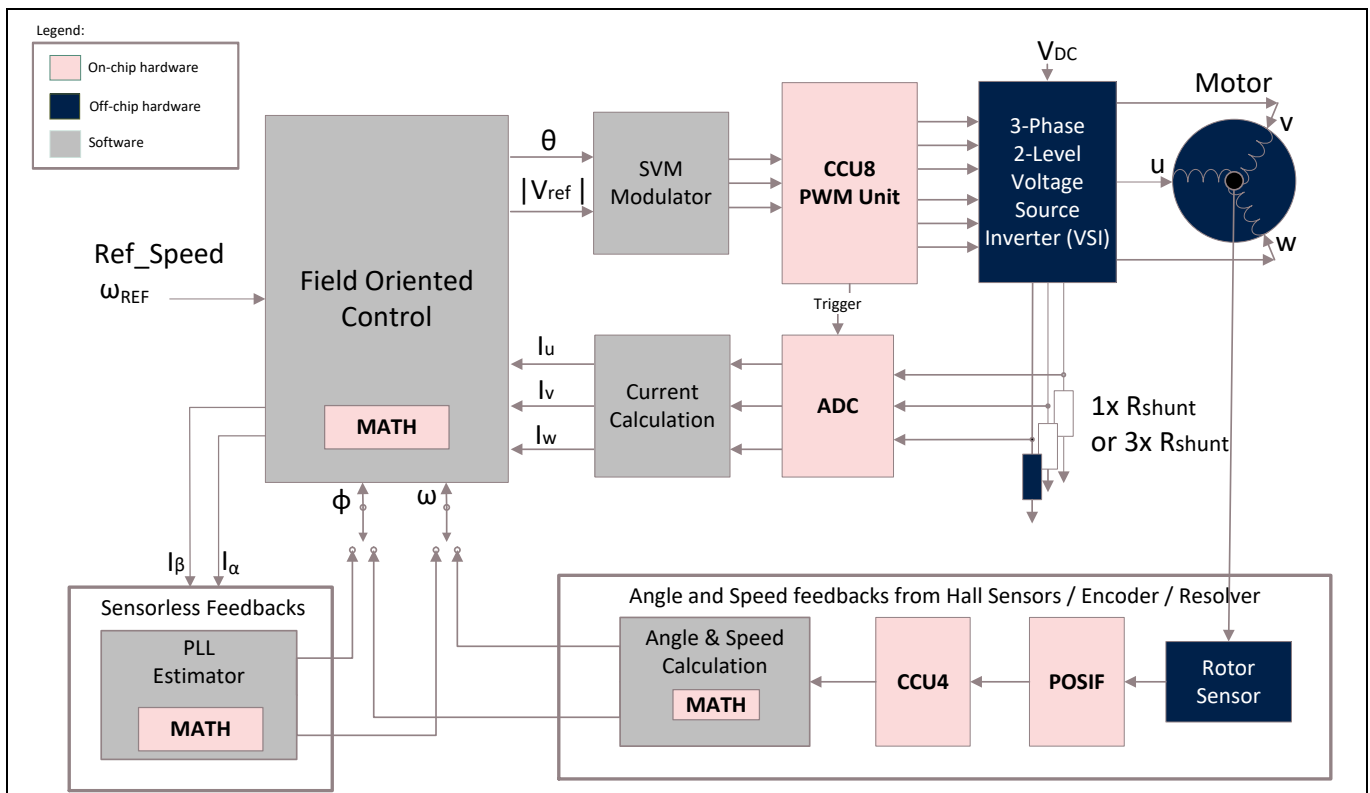


Figure 1 Block diagram of PMSM FOC motor control

Introduction

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 the external opamp
- Space vector modulation (SVM) with pseudo zero vectors (PZV) for single-shunt current sensing
- Maximum efficiency tracking (MET) 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, overmodulation and 7-segment SVM
	DC link single-shunt: Support PZV and 4-segment SVM, no overmodulation
Startup algorithm	Direct FOC startup
Protection	Phase overcurrent protection
	DC link undervoltage and overvoltage protection
Device features	ADC on-chip gain for current sensing
	ADC synchronous conversion: motor phase current sensing
Control features	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
	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 motor
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 32 MHz core and 64 MHz peripheral frequency
XMC1302	XMC™ MCU product with specific feature set. E.g., CORDIC
XMC1400	XMC™ MCU series with 48 MHz core and 96 MHz 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
XMC4800	XMC™ MCU series with 144 MHz core and peripheral frequency

Introduction

1.3 XMC™ resource allocation

The XMC1302, XMC1402, XMC1404, XMC4400, and XMC4800 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 and XMC4800. 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/XMC4800 Motor Control Application Kit (order numbers KIT_XMC1X_AK_MOTOR_001, KIT_XMC4400_DC_V1, EVAL-XMC4800PSOC6M5 respectively).

Table 3 XMC™ MCU peripherals used for sensorless PMSM FOC with three-shunt current sensing

XMC™ peripherals	Usage	Default resource allocation in XMC1300 or XMC1400	Default resource allocation in XMC4400	Default resource allocation in XMC4800
CCU80	PWM generation for Phase U	CCU80 slice 0	CCU80 slice 0	CCU81 slice 0
	PWM generation for Phase V	CCU80 slice 1	CCU80 slice 1	CCU81 slice 1
	PWM generation for Phase W	CCU80 slice 2	CCU80 slice 3	CCU81 slice 2
	Timer for ADC trigger	CCU80 slice 3	CCU80 slice 2	CCU81 slice 3
VADC	Phase U current sensing	G0 channel 4 ¹ G1 channel 3 ¹	G0 channel 1	G0 channel 0
	Phase V current sensing	G0 channel 3 ¹ G1 channel 2 ¹	G1 channel 7	G1 channel 0
	Phase W current sensing	G0 channel 2 ¹ G1 channel 4 ¹	G2 channel 1	G2 channel 4
	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	G0 channel 0 G1 channel 0 G1 channel 1
	DC link voltage sensing	G1 channel 5	G1 channel 1	G1 channel 2
	DC link average current sensing	G1 channel 6	G1 channel 0 (not implemented)	G0 channel 6 (not implemented)
	Potentiometer for speed change	G1 channel 7	G1 channel 5	G1 channel 3
MATH		CORDIC	FPU	FPU
Nested vectored interrupt	PWM period match interrupt	CCU80.SR0	CCU80.SR0	CCU81.SR0
	CTrap interrupt	CCU80.SR1	CCU80.SR3	CCU81.SR3

¹ The same input pin must connect to both the ADC group channels to perform synchronous sampling. Refer to Section 3.2.1 for details.

Introduction

XMC™ peripherals	Usage	Default resource allocation in XMC1300 or XMC1400	Default resource allocation in XMC4400	Default resource allocation in XMC4800
controller (NVIC)	ADC interrupt for single-shunt sensing	VADC0.G1SR1 ¹	Not available	Not available

¹ If DC link current sensing uses the VADC Group0 channels, then the VDC0.G0SR1 interrupt node is used.

Introduction

1.4 Interconnectivity of XMC™ hardware modules

The XMC1000, XMC4400, and XMC4800 families have comprehensive hardware interconnectivity.

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

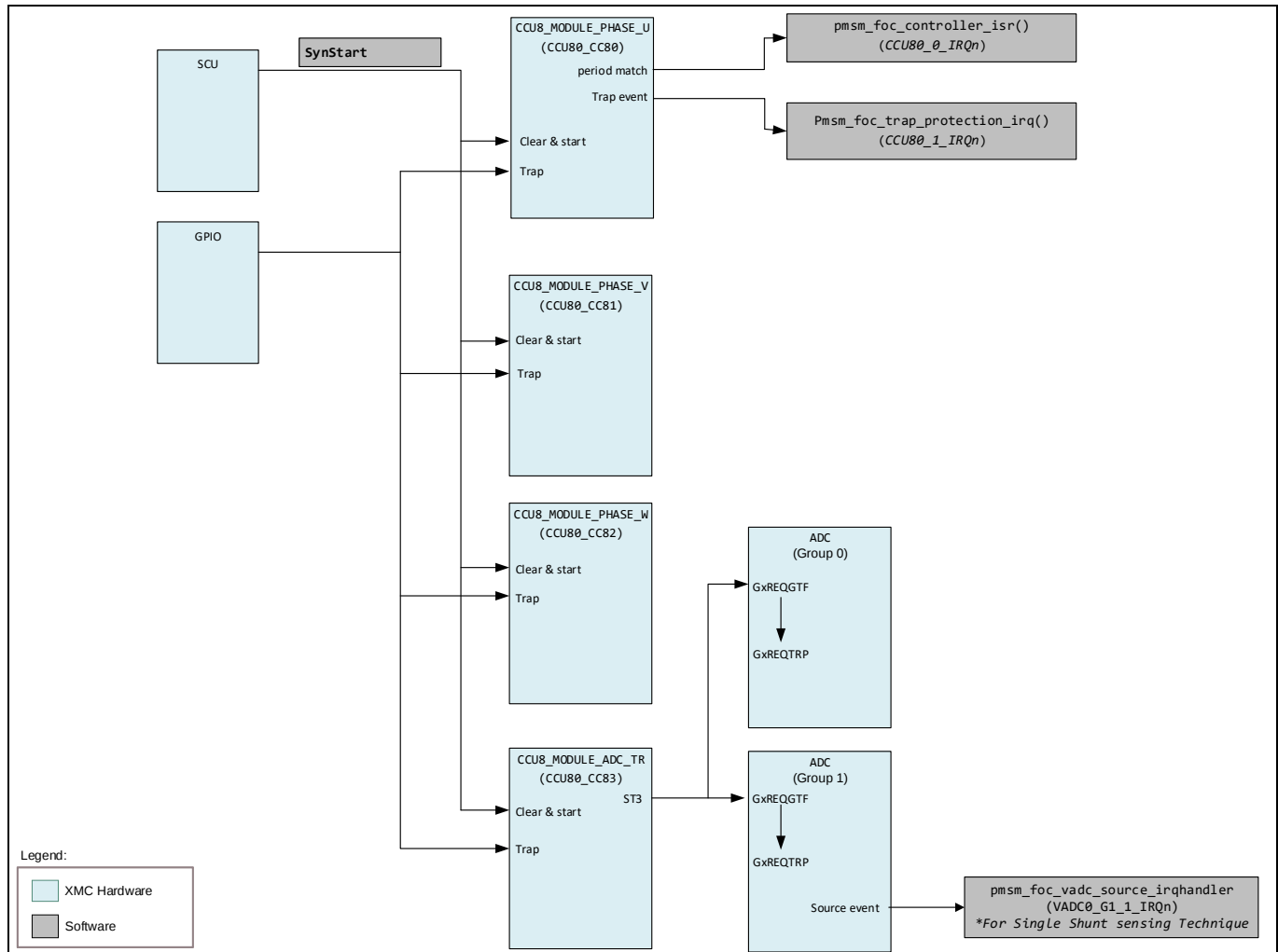


Figure 2 XMC1302 hardware interconnection

Note:

1. CCU8 slice timers are started synchronously with the sync start signal from the SCU (system control unit).
2. VADC conversion is triggered by the CCU8 Slice 3 compare match status signal.

Introduction

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

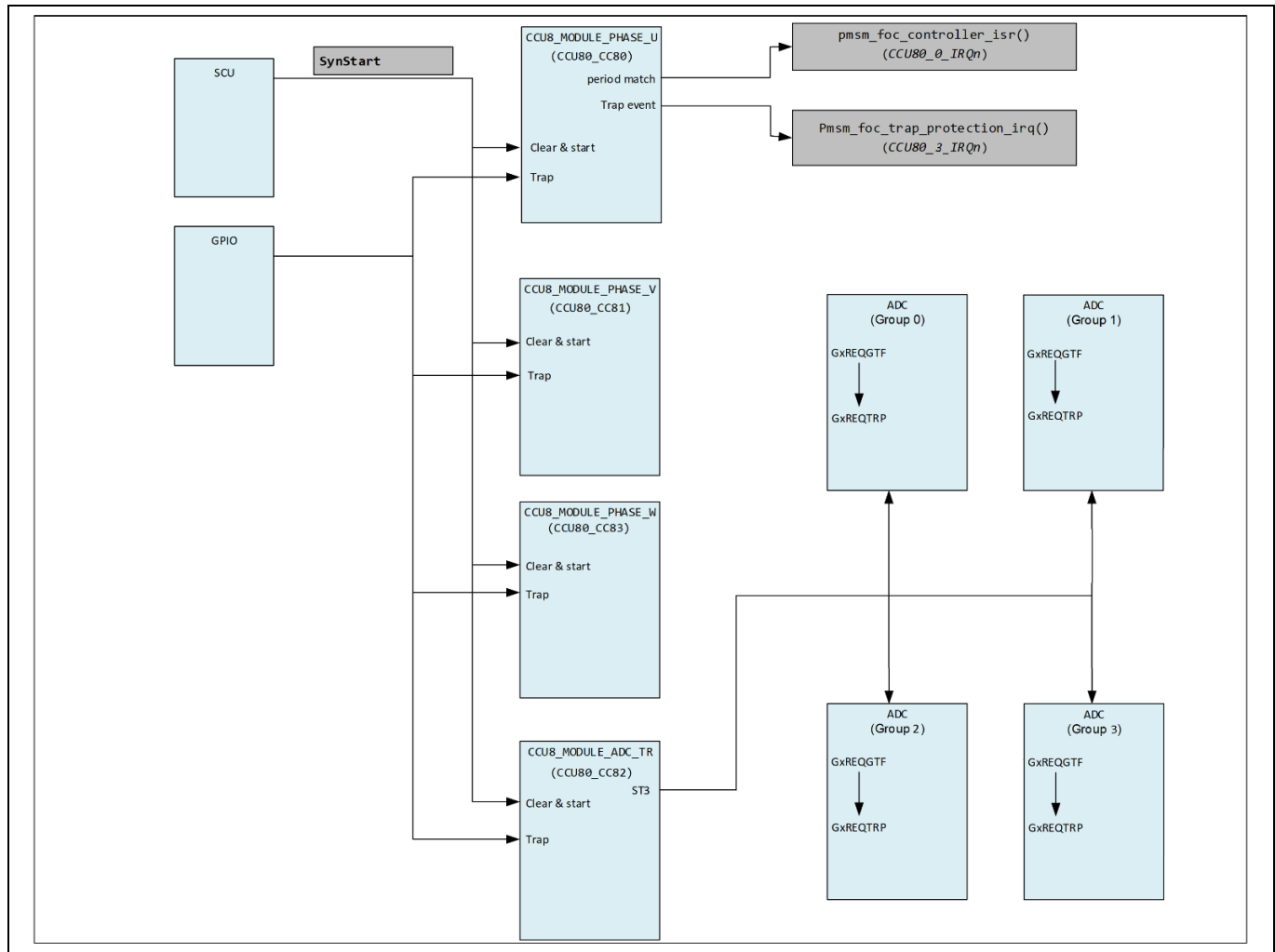


Figure 3 XMC4400 hardware interconnection

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.
5. XMC4800 uses CCU81 Slice 0, 1, 2, 3 for phase U, V, W, and ADC trigger respectively. CCU81_0 and CCU81_3 is used for controller and trap protection interrupts respectively.

1.5 Execution time and memory usage

In the XMC1000 and the XMC4000 families, access to the 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 because the execution time to run the FOC algorithm is reduced by approximately 30%.

Note: See Chapter 9 for the list of APIs running in the SRAM.

Breakdown of the memory usage and CPU time utilization is provided in the following table based on the default settings for the XMC1000 Motor Control Application Kit (XMC1302), XMC1400 Boot Kit (XMC1404), XMC4400 Motor Control Application Kit, and XMC4800PSOC6M5 board.

- 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, XMC1400, XMC4400, and XMC4800

PWM frequency	20 kHz – ISR runs every 50 µs			
DAVE™ 4 GCC compiler optimization level	Optimized most (-O3)			
MCU	XMC1300 @32 MHz	XMC1400 @48 MHz	XMC4400 @120 MHz	XMC4800 @144MHz
CPU utilization	31 µs (62%)	21 µs (42%)	12.5 µs (25%)	10.7 µsec (21.4%)
Flash code size (bytes)	10416	10822	24790	24262
SRAM code size (bytes)	2376	2376	3060	3060
SRAM data size (bytes)	348	352	176	176

Introduction

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.

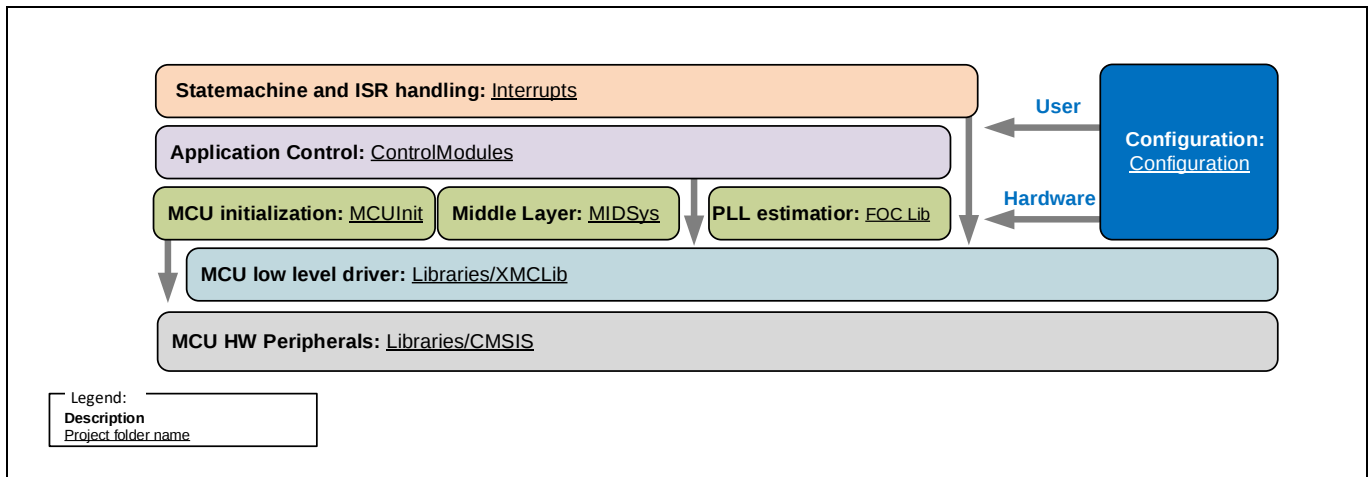


Figure 4 PMSM FOC software overview: layered structure

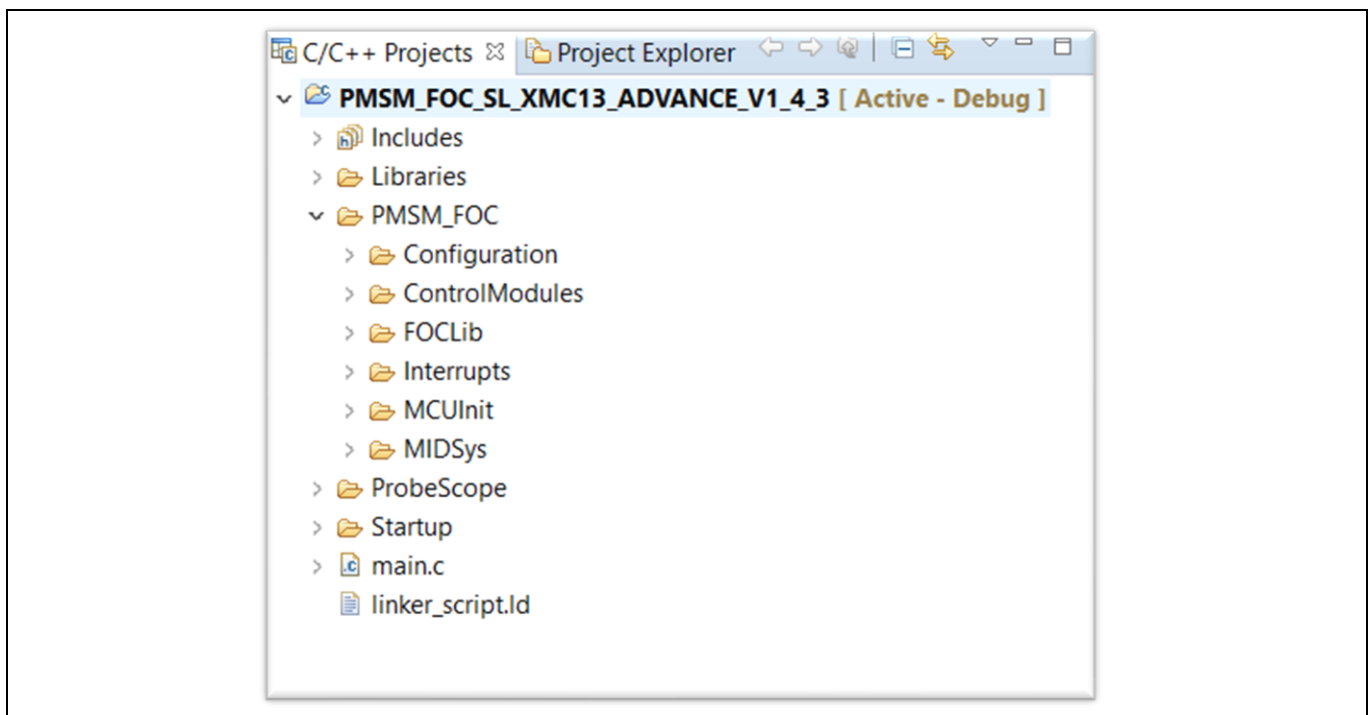


Figure 5 Project folder structure

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.

Introduction

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 ISRs. All files for this layer are in the *ControlModules* folder.

Configuration: Configuration

The user configuration affects the general behavior of the software. The file *pmsm_foc_user_config.h* can be modified. Predefined hardware kits are available. The hardware configuration allows for more detailed adaptation to the customer hardware.

This layer is divided into the following:

- Controller card
- Inverter card
- Motors

The specific associated *.h files are 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 are in the *Configuration* folder.

PLL Estimator: FOCLib

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

The file is in the *FOCLib* folder.

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 and XMC4800 MCUs.

All files for this layer are in the *MIDSys* folder.

MCU Initialization: MCUInit

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

All files for this layer are in the *MCUInit* folder.

Introduction

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 are in the *Libraries* folder.

1.7 Limitations of use for PMSM FOC software

This application note uses 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 platforms.
- The following are not currently documented:
 - Scaling for `pmsm_foc_set_motor_target_torque()`.
 - Scaling for `pmsm_foc_set_motor_target_voltage()`.
 - PT1 filter
- 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, this shift ensures that the 3 shunt measurement is stored in the middle center of the PWM pattern.
- Overcurrent protection is through a DC link shunt. This protects the inverter but the current between phases of the motor is not measured.
- No catch-free running implementation.
- UART debug is available only in XMC1300/XMC1400; it is not tested.
- SETTING_TARGET_SPEED options BY_POT_ONLY, and BY_UART_ONLY are not tested.
- In XMC4400 and XMC4800, single-shunt sensing is not implemented.

2 PMSM FOC sensorless software components

The PMSM FOC motor control software provides functionality to drive PMSM motors with sensors or sensorless modules. The current version supports sensorless modules. It 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.

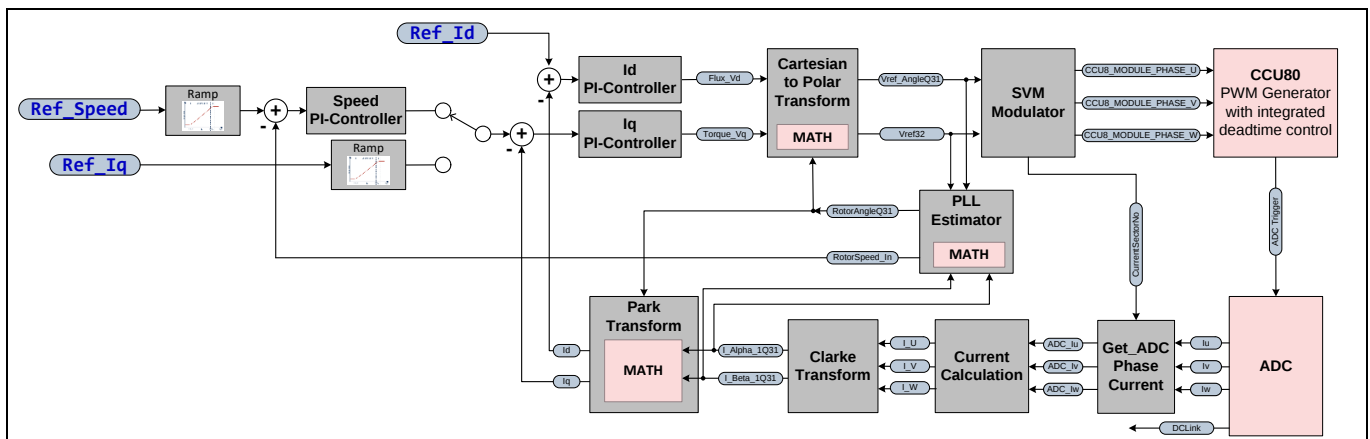


Figure 6 PMSM FOC block diagram

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 the following:

- USER_SPEED_LOW_LIMIT_RPM
- USER_SPEED_HIGH_LIMIT_RPM

The target speed can be controller using a potentiometer.

The relationship between the ADC data and motor target speed for the speed control scheme is as follows:

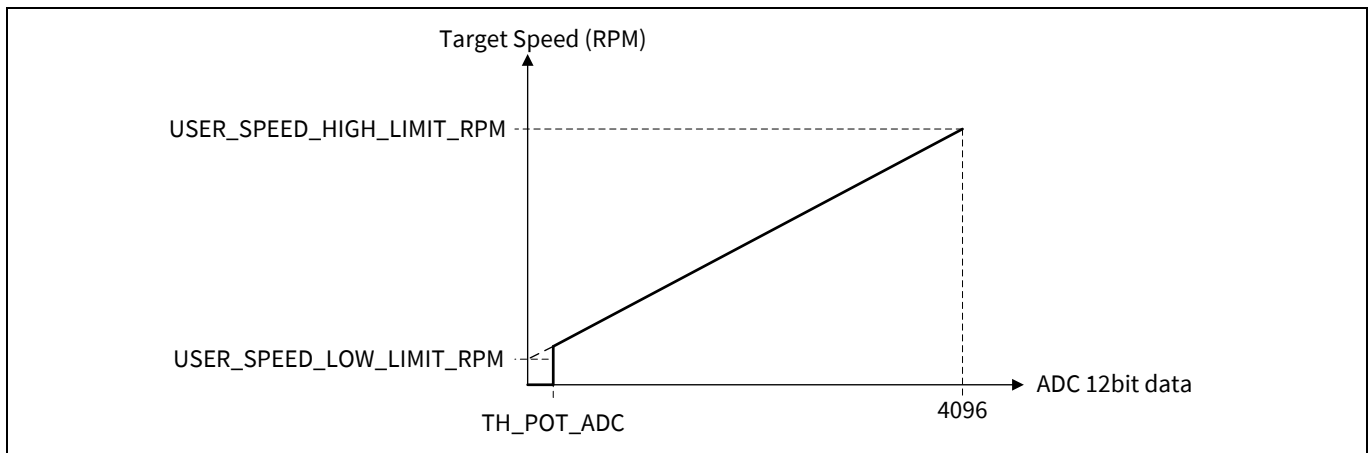


Figure 7 Potentiometer ADC value vs. speed in speed control scheme

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

- Set the 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, resulting in 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 Section 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 used only in speed control.

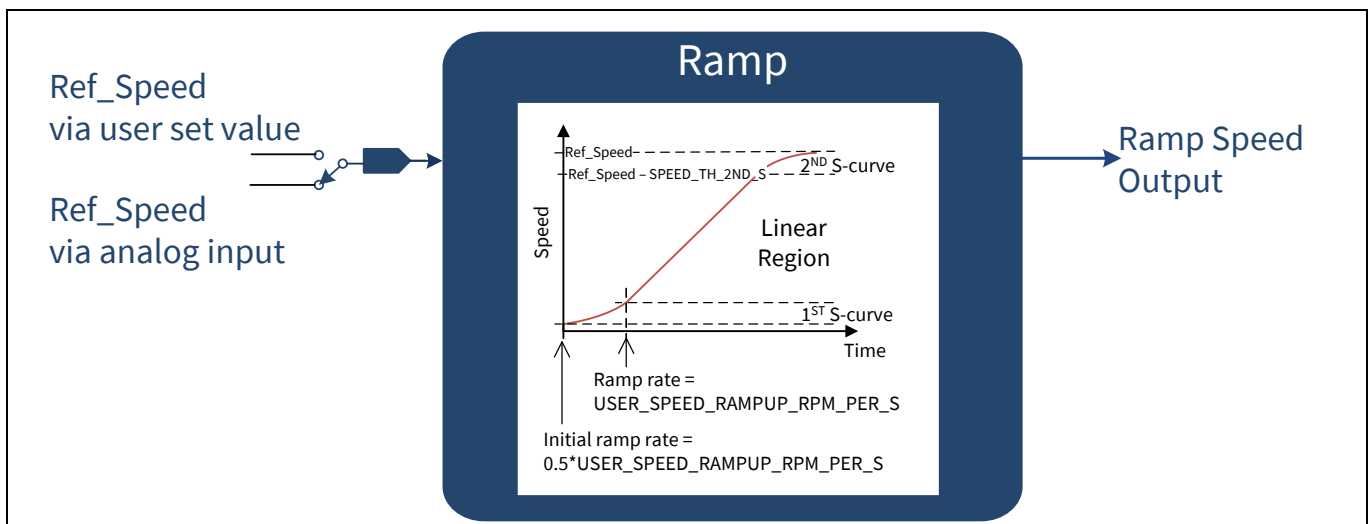


Figure 8 S-curve ramp generator

In both ramp generator functions, the DC link voltage is monitored during the 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 overvoltage condition during 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 V_q .

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 open-loop voltage control, a reference voltage (V_{ref}) is used to cause the power inverter to generate a given voltage at the motor. The mechanical load influences the speed and current of the PMSM motor.

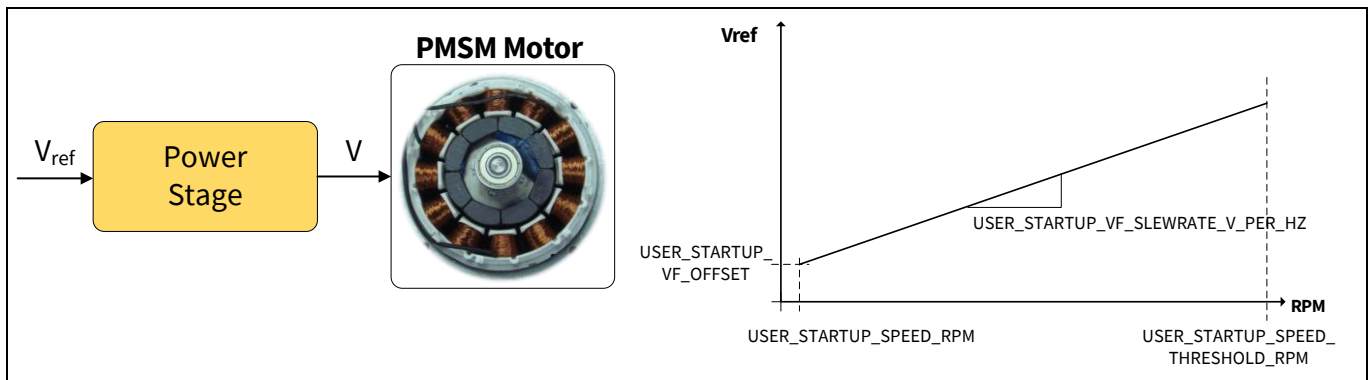


Figure 9 Open-loop voltage control

2.3.2 Speed control

A speed control scheme is a closed-loop control. This scheme uses a cascaded speed and current 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.

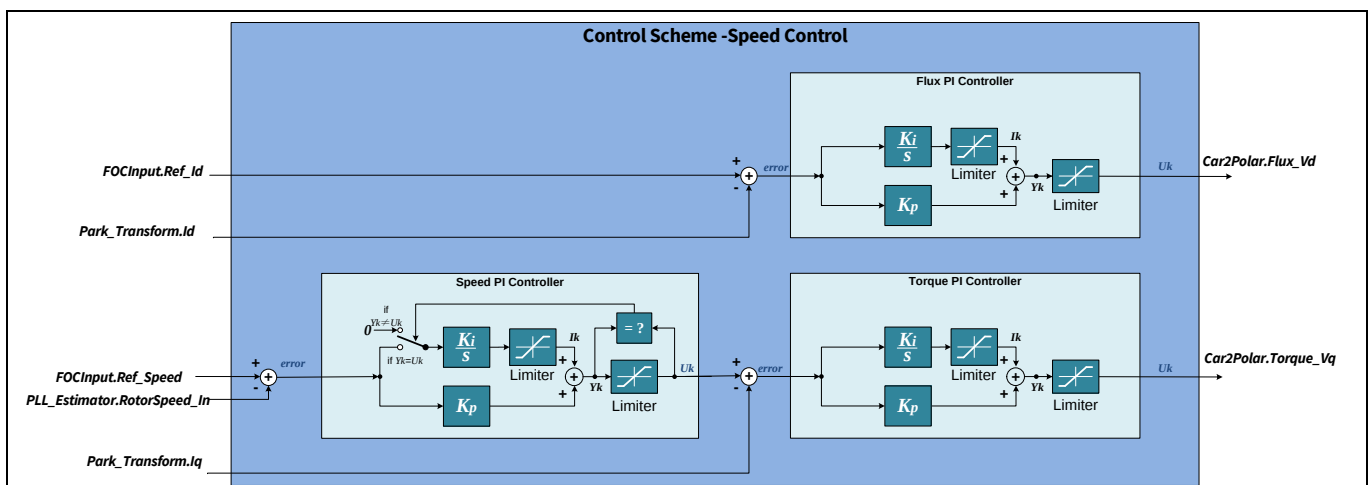


Figure 10 Speed control

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)

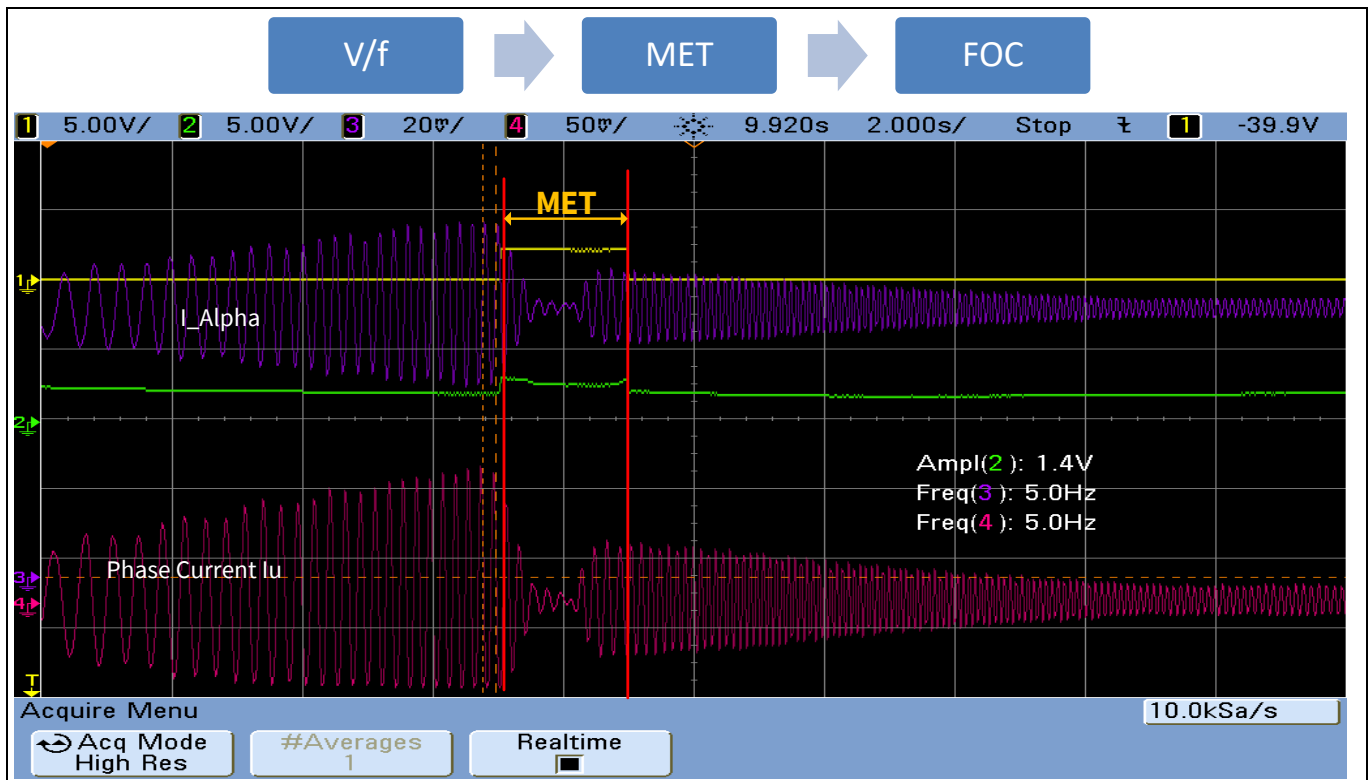


Figure 11 3 steps motor startup mechanism

The three steps are:

1. The motor starts in V/f open-loop control state and ramps up to a user-defined startup speed.
2. Sensorless MET closed-loop control state takes over. This state is added to ensure that 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 in MET and FOC closed-loop
- Smooth transitions for all 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 (I_{q_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.

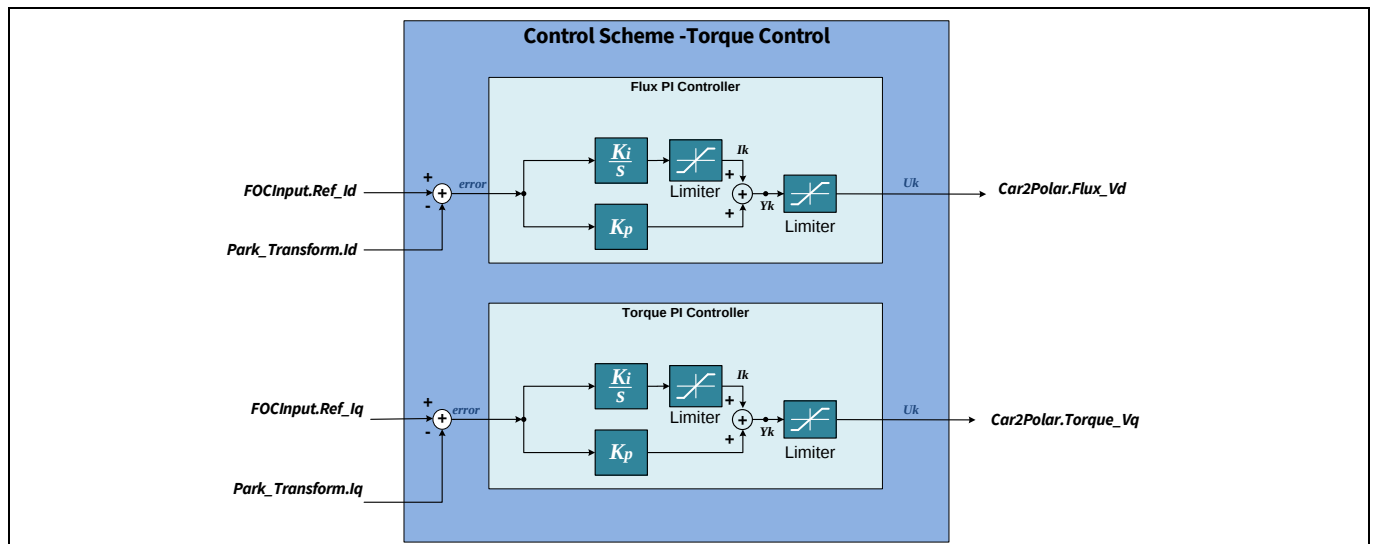


Figure 12 Torque control scheme

To improve the execution performance, 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, and read the CORDIC result later. Instead, in XMC4400/4800, the calculations are done immediately in FPU (not in parallel) calling the `fpu_cart2polar()` function implemented in the `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)

SVM transforms the stator voltage vectors into pulse width modulation (PWM) signals (compare match values).

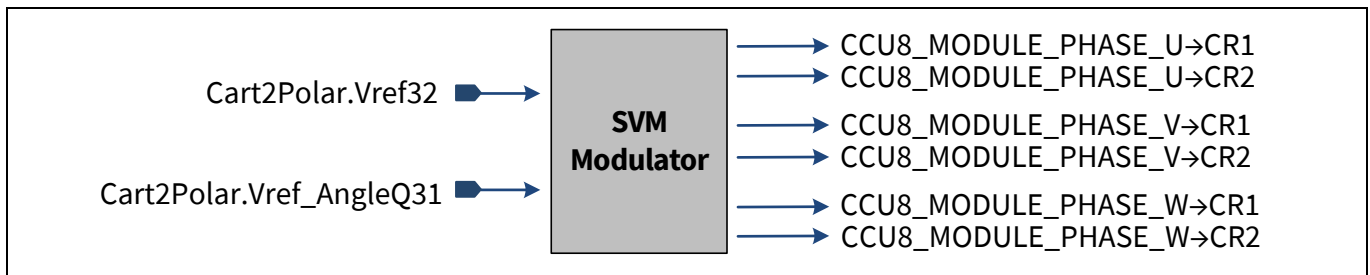


Figure 16 PWM SVM function

The 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 user-configurable in `pmsm_foc_invertercard_parameter.h`

The timer counting scheme used in 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 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, XMC4400 KIT_XMC4400_DC_V1, and XMC4800 XMC4800PSOC6M5 board.

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

Parameters	Settings for XMC1300/XMC1400 and XMC4400/XMC4800 ¹
Timer counting mode	Edge-aligned mode
Shadow transfer on clear	Enabled
Prescaler mode	Normal
Passive level	Low
Asymmetric PWM	Enable

¹ XMC4800 uses CCU81 Slice 0, 1, 2, 3 for phase U, V, W, and ADC trigger respectively. CCU81_0 and CCU81_3 is used for controller and trap protection interrupts respectively.

Parameters	Settings for XMC1300/XMC1400 and XMC4400/XMC4800 ¹
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 ns)
Phase U slice period match interrupt event	Enabled
Trap interrupt event	Enabled

2.5.1 7-segment SVM

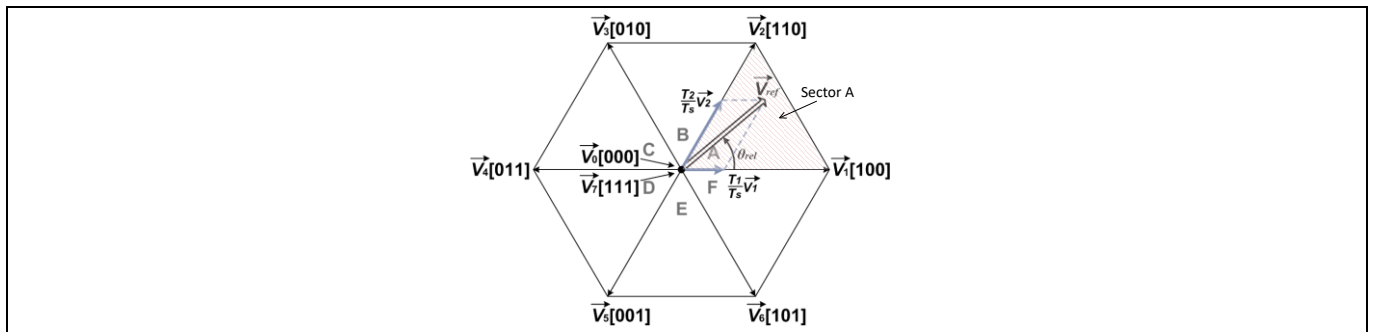


Figure 17 7-segment SVM

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

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

Equation 4

$$T_s = T_0 + T_1 + T_2$$

Equation 5

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

Equation 6

$$T_2 = \frac{\sqrt{3}|\vec{V}_{ref}|T_s}{V_{DC}} \sin(\theta_{rel})$$

Equation 7

$$T_0 = T_s - T_1 - T_2$$

Equation 8

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 $T_0/2$.

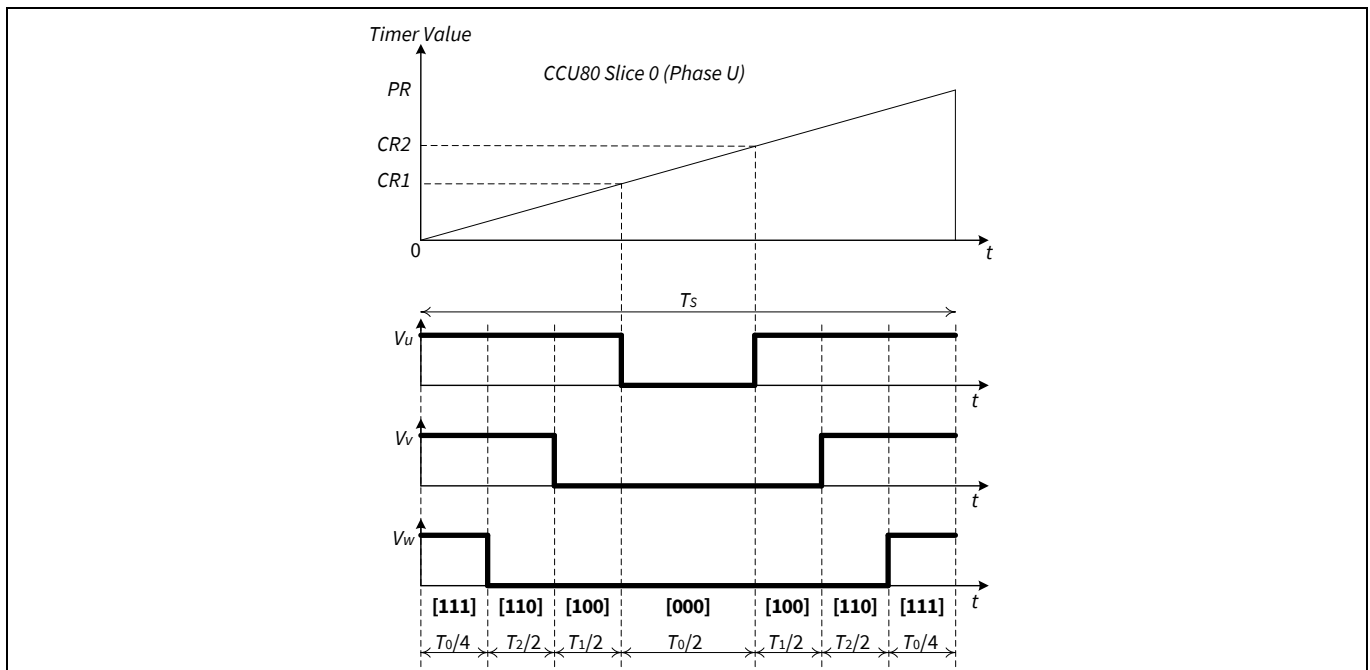


Figure 18 7-segment SVM timing diagram in SVM sector A

2.5.2 5-segment SVM

The 5-segment SVM uses the same equations as in 7-segment SVM to calculate the T_0 , T_1 , and T_2 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.

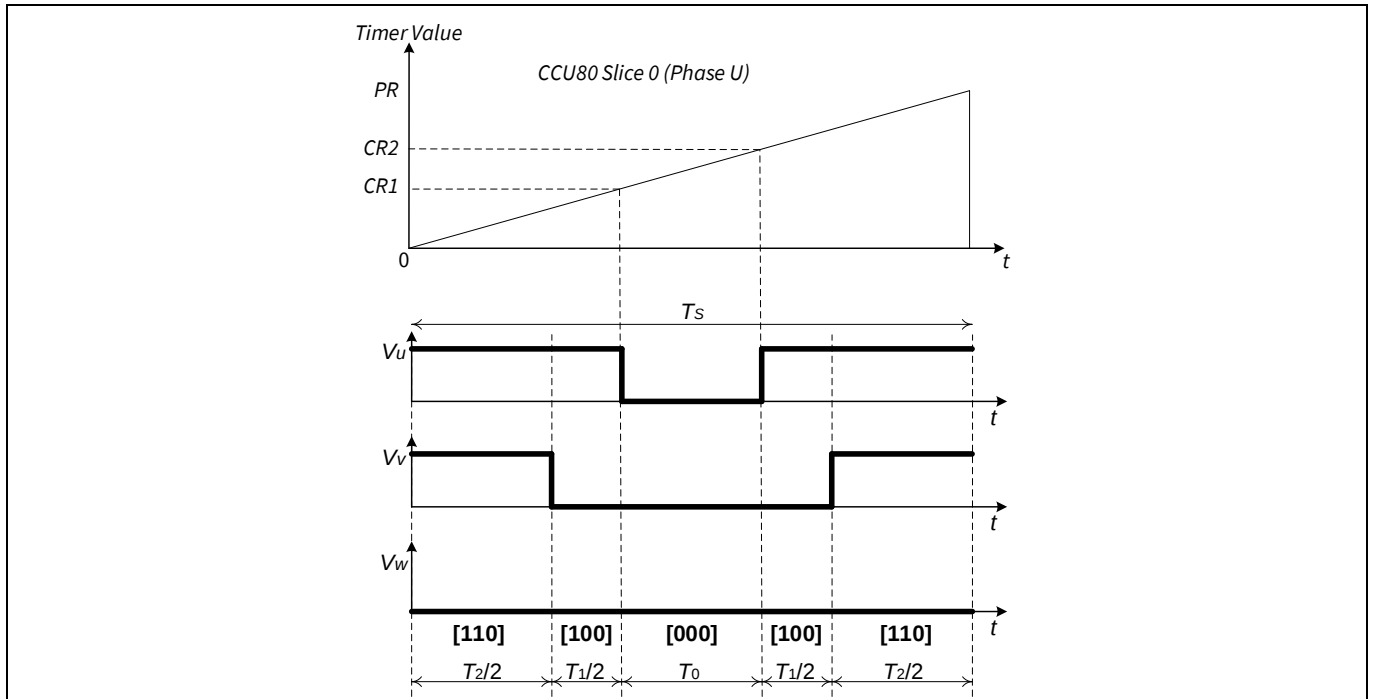


Figure 19 5-segment SVM timing diagram in SVM sector A

2.5.3 Pseudo zero vector (PZV)

In single-shunt current reconstruction, the current through one of the phases 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](#).

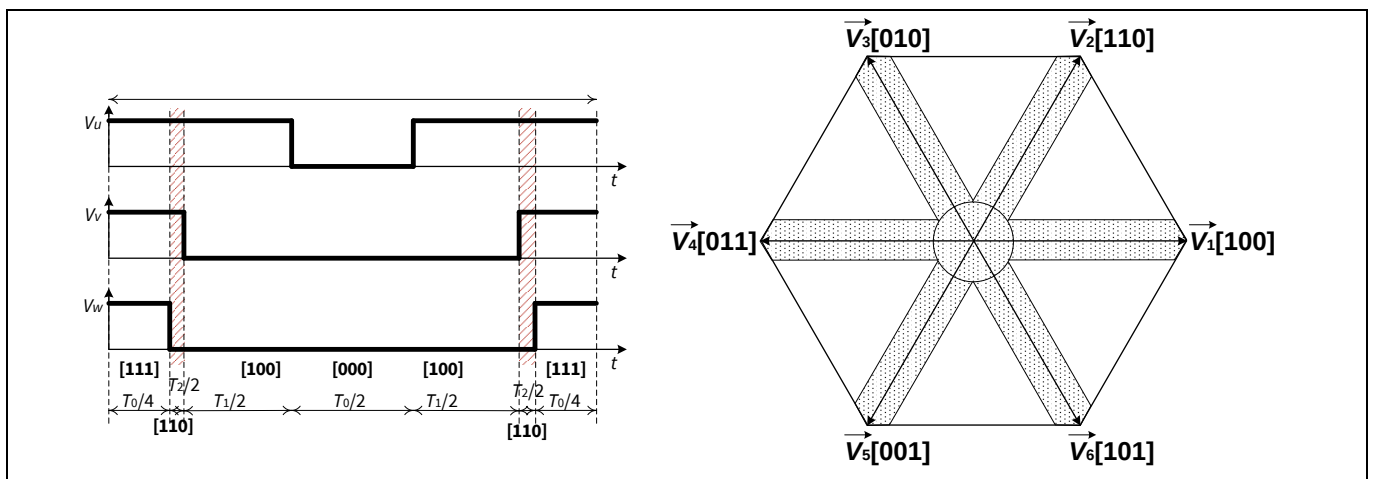


Figure 20 7-segment SVM – single-shunt sensing

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

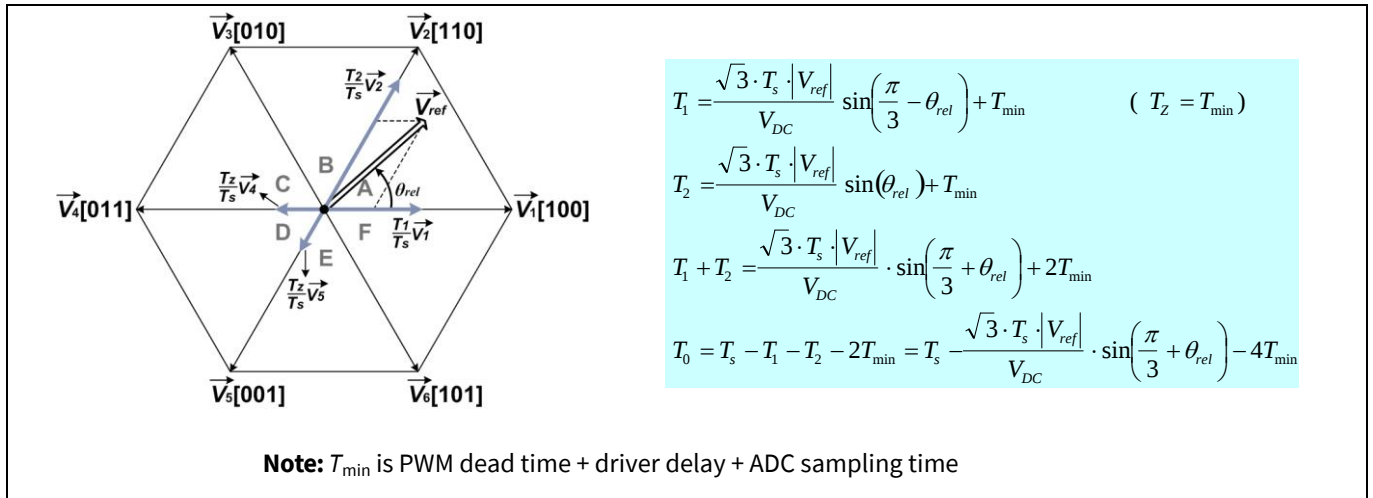


Figure 21 Pseudo zero vector

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 PZV.

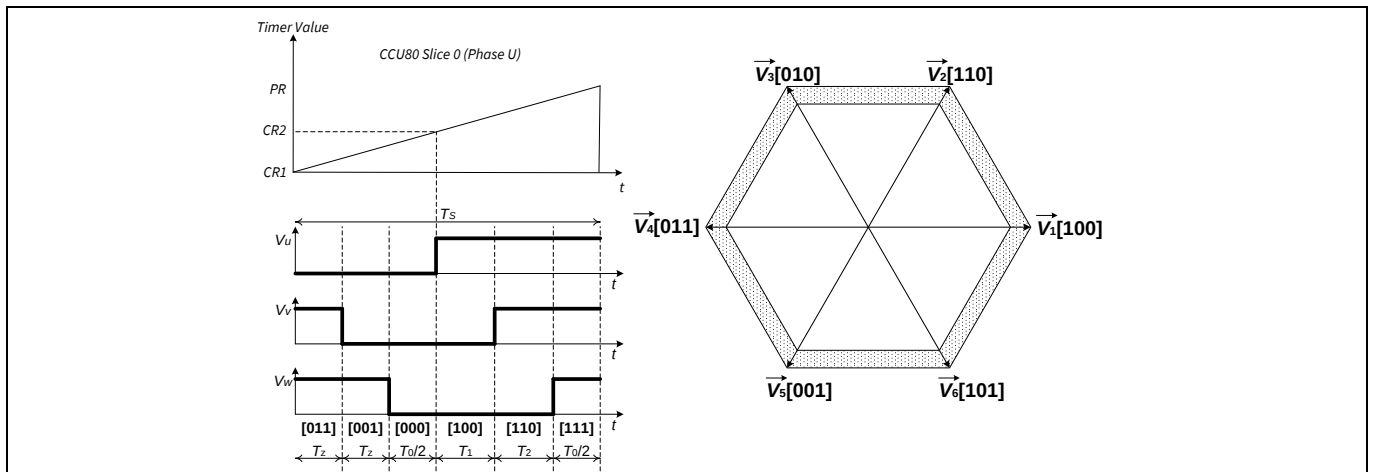


Figure 22 PZV timing diagram in sector A

2.5.4 4-segment SVM

This SVM pattern is also used in the single-shunt current sensing technique. PZV 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.

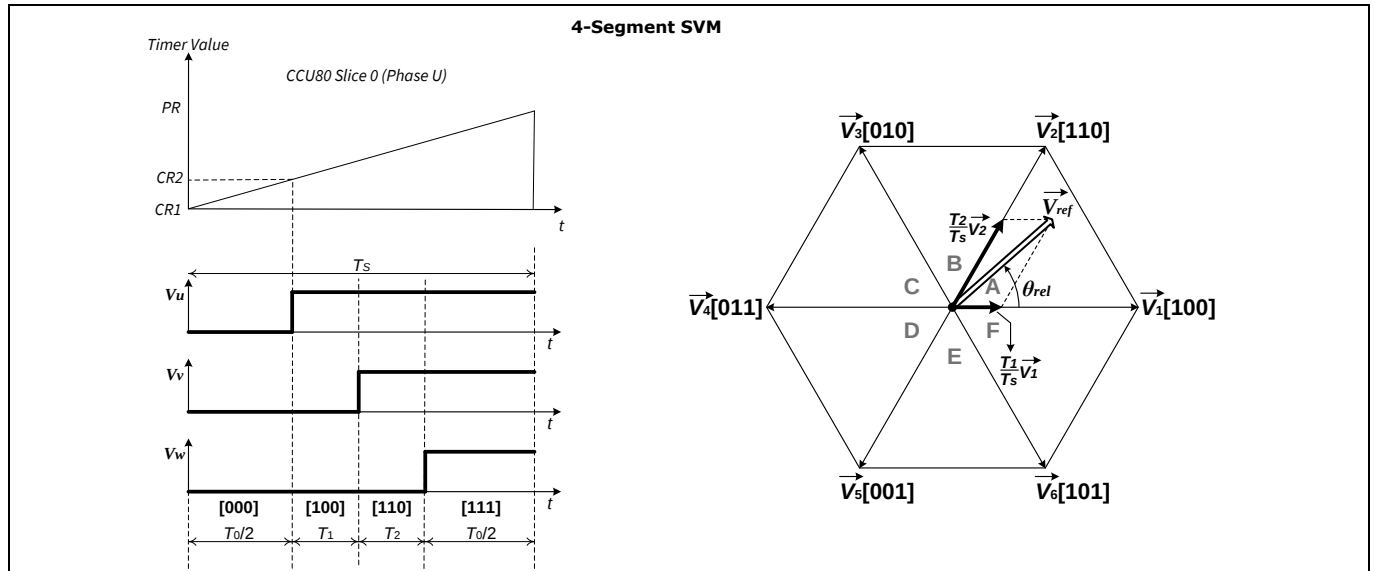


Figure 23 4-segment SVM

2.5.5 Overmodulation SVM

For sinusoidal commutation, V_{ref} must 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 “overmodulation” of SVM.

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

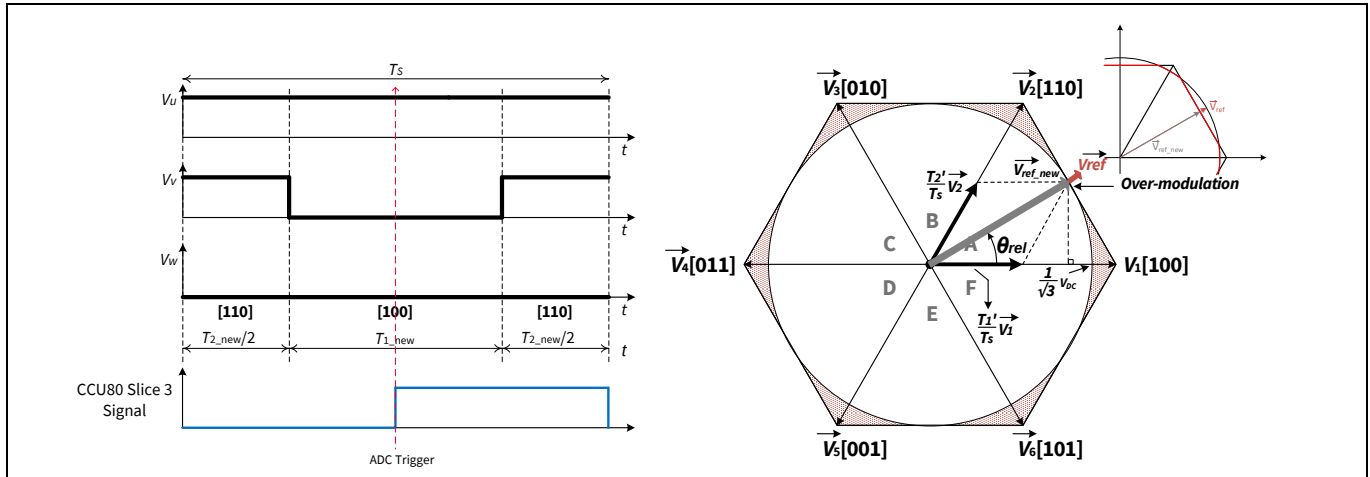


Figure 24 Overmodulation

In the PMSM FOC software, 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 the 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 recalculated as:

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

Equation 9

$$T_{2_new} = T_s - T_{1_new}$$

Equation 10

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 in Figure 24. From the diagram, it shows only two phases of the current, I_v and I_w , can be measured.

When the motor is running at high-speed, overmodulation is used to maximize DC bus utilization. The drawback of overmodulation 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 2¹².

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

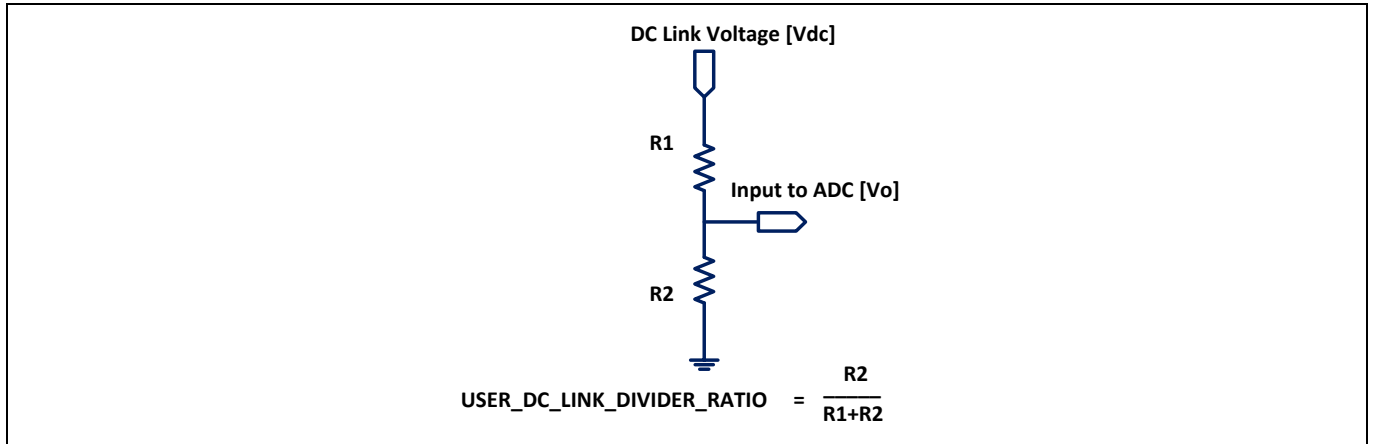


Figure 25 DC link voltage divider

2.7 Clarke Transform

In this module, the phase currents (I_{L_u} , I_{L_v} , I_{L_w}) from the current sensing module are transformed into currents I_{Alpha} and I_{Beta} on the 2-phase orthogonal reference frame.

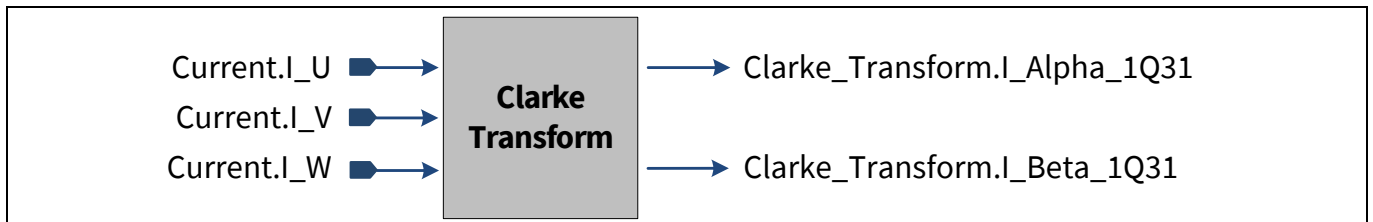


Figure 26 Clarke Transform

Equations for Clarke Transform:

$$I_{\alpha} = I_{L_u}$$

Equation 11

$$I_{\beta} = \frac{1}{\sqrt{3}} \cdot I_{L_u} + \frac{2}{\sqrt{3}} \cdot I_{L_v} = \frac{1}{\sqrt{3}} \cdot (I_{L_u} + 2 \cdot I_{L_v})$$

Equation 12

Where:

$$I_{I_u} + I_{I_v} + I_{I_w} = 0$$

Equation 13

The scaling factor of the currents I_U, I_V, and I_W are based on the current scaling (see Section [2.10](#)). In XMC1300/XMC1400, the outputs of the Clarke transform function are shifted left by 14 bits (CORDIC_SHIFT) due to the code optimization for the XMC™ CORDIC hardware module.

2.8 Park Transform

In Park Transform, the currents I_{α} and I_{β} are resolved to a rotating orthogonal frame with the rotor angle.

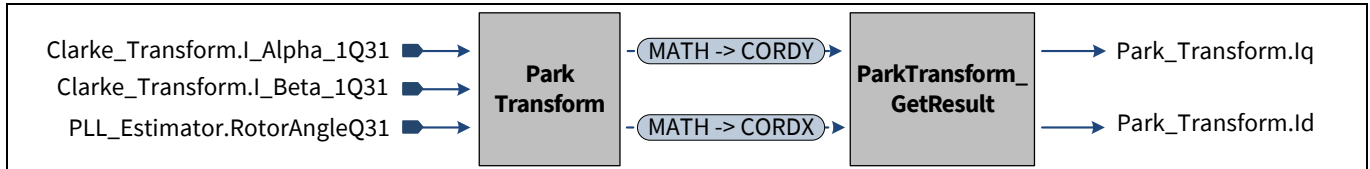


Figure 27 Park Transform

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

$$I_q = (-I_{\alpha} \cdot \sin(RotorAngle) + I_{\beta} \cdot \cos(RotorAngle)) * CORDIC_GAIN$$

Equation 14

$$I_d = (I_{\alpha} \cdot \cos(RotorAngle) + I_{\beta} \cdot \sin(RotorAngle)) * CORDIC_GAIN$$

Equation 15

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 (see XMC1300 or XMC1400 reference manual [1]).

Scaling factor of I_q and I_d are based on the current scaling (see Section 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	<code>Clarke_Transform.I_Beta_1Q31</code>
CORDY	<code>Clarke_Transform.I_Alpha_1Q31</code>
CORDZ	<code>PLL_Estimator.RotorAngleQ31</code>

In XMC4400/4800, the Park Transform function is calculated by the FPU using the `fpu_park_q31()` function implemented in the `fpu_math2` library.

2.9 Protection

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

- CCU80 CTrap function
- Overcurrent protection
- Overvoltage/undervoltage protection

CCU80 CTrap function

Trap function of the CCU8 module provides protection against hardware overload conditions. 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 ISR, the gate driver is disabled and the motor state machine is set to `TRAP_PROTECTION` state.

Overcurrent protection (only in XMC1300/XMC1400)

The average current flow through the DC link shunt resistor is sampled every cycle of the PWM. This value is read to detect an overcurrent condition. Once this condition occurs, the reference motor speed is scaled down by a factor until 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 the motor speed using the following equation:

$$FOCInput.Ref_Speed = \frac{Motor.RefSpeed * FOCInput.overcurrent_factor}{4096}$$

Equation 16

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

The nominal value of `FOCInput.overcurrent_factor` is 4096. This value is increased back to the nominal value when the average DC link current is within the limit. In XMC4400/4800, this feature is not supported.

Overvoltage/undervoltage 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 `pmsm_foc_over_under_voltage_isr()` function 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

The 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 the Physical Value to the Norm Value represented in the software.

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

Equation 17

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 $max_speed_integer$

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

Scaling for SVM voltage

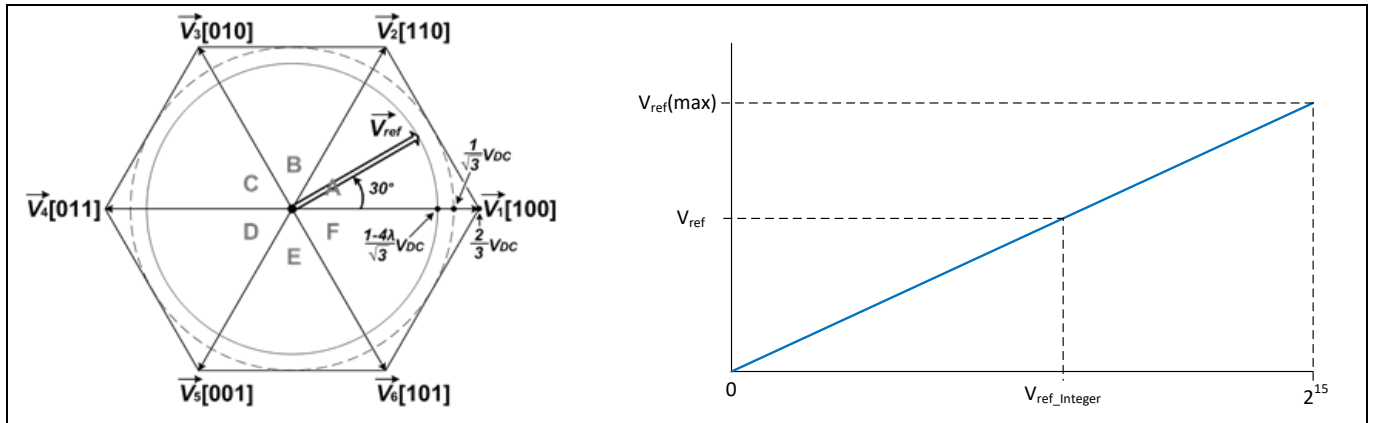


Figure 28 **SVM voltage scaling**

$$V_{ref} = \frac{N_{Vref_SVM}}{2^{15}} \cdot V_{ref_Integer}$$

Equation 18

N_{Vref_SVM} is the maximum reference voltage of SVM

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

Equation 19

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

V_{DC} is inverter DC link voltage, USER_VDC_LINK_V

Example:

USER_VDC_LINK_V is 24.0V, $\lambda = 0$

$$\Rightarrow N_{Vref_SMV} = 13.86 \text{ V}$$

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

Scaling for phase current

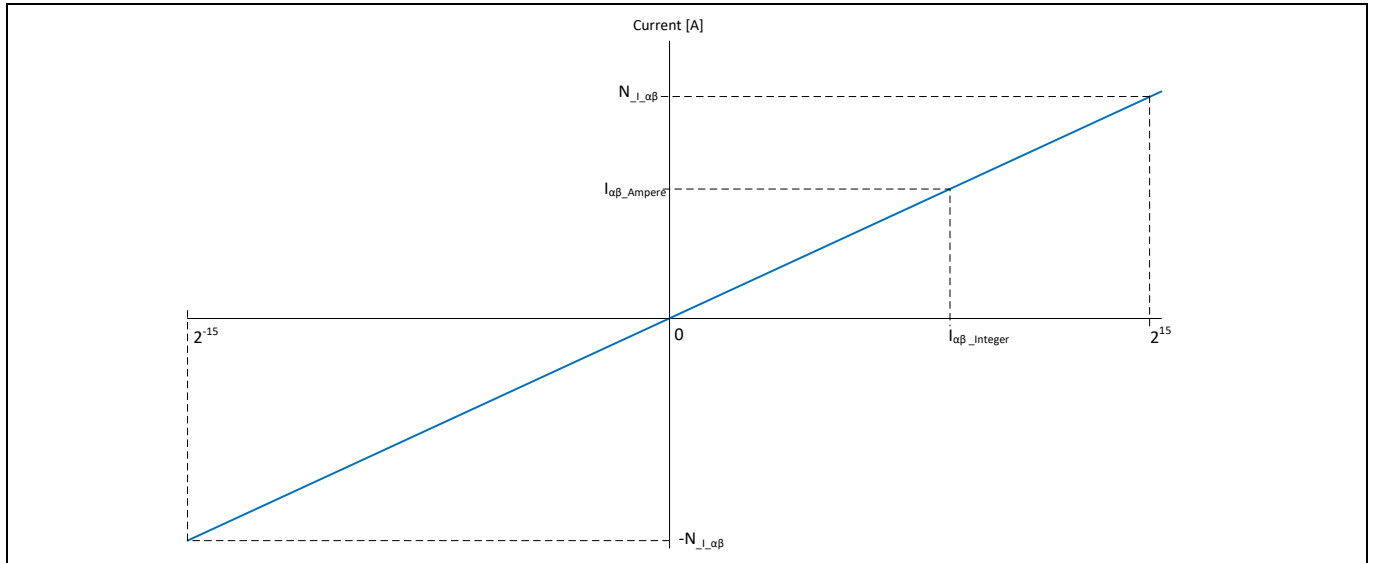


Figure 29 Phase current scaling

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

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

Equation 20

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

Scaling for angle and speed

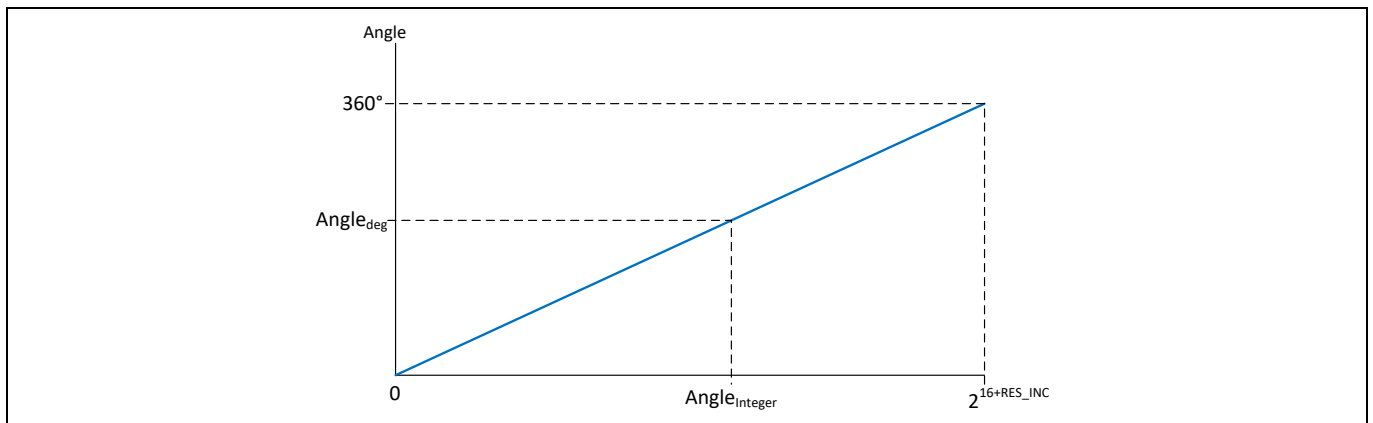


Figure 30 Angle scaling

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:

$$Angle_{deg} = \frac{360^\circ}{2^{16+RES_INC}} \cdot Angle_{integer}$$

Equation 21

Following the angle scaling, the speed scaling is:

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

Equation 22

Where:

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

Target Value for speed is:

$$N_{max_speed} = \frac{USER_SPEED_HIGH_LIMIT_RPM \cdot USER_MOTOR_POLE_PAIR}{60} \cdot 360^\circ \text{ degree/seconds}$$

Equation 23

$$max_speed_integer = \frac{N_{max_speed} \cdot 2^{16+USER_RES_INC}}{60 \cdot f_{CCU8_PWM}}$$

Equation 24

$$N = \log_2(max_speed_integer)$$

Equation 25

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.

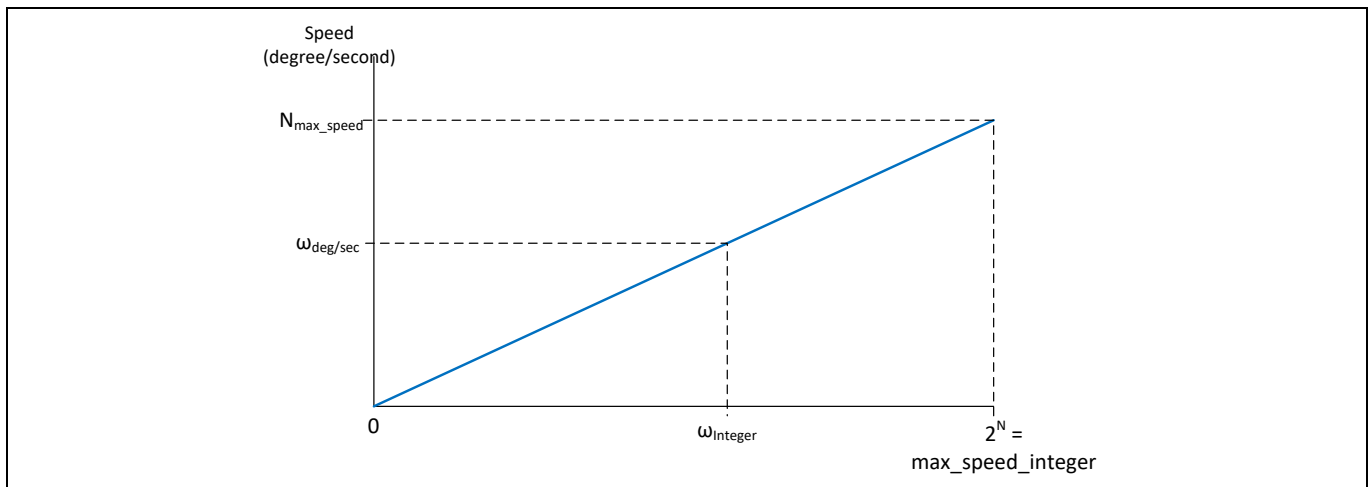


Figure 31 Speed scaling

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

$$N_{max_speed} = \frac{(10000 \text{ rpm} * 4) * 360^\circ}{60} = 240,000 \text{ degree/second}$$

Equation 26

$$max_speed_integer = \frac{240,000 * 2^{16+3}}{360 * 25,000} = 13,981$$

Equation 27

$$N = \log_2(max_speed_integer) = \log_2(13,981) = 13.77$$

Equation 28

To represent speed 2000 rpm which is 48,000 (degree/second), the software integer, $\omega_{integer} = 2796$

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.

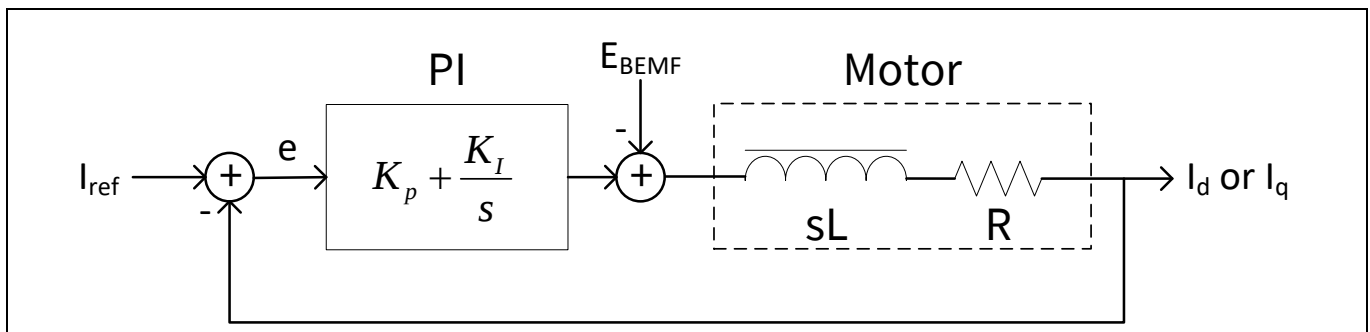


Figure 32 Torque/flux current control loop

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 the 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.

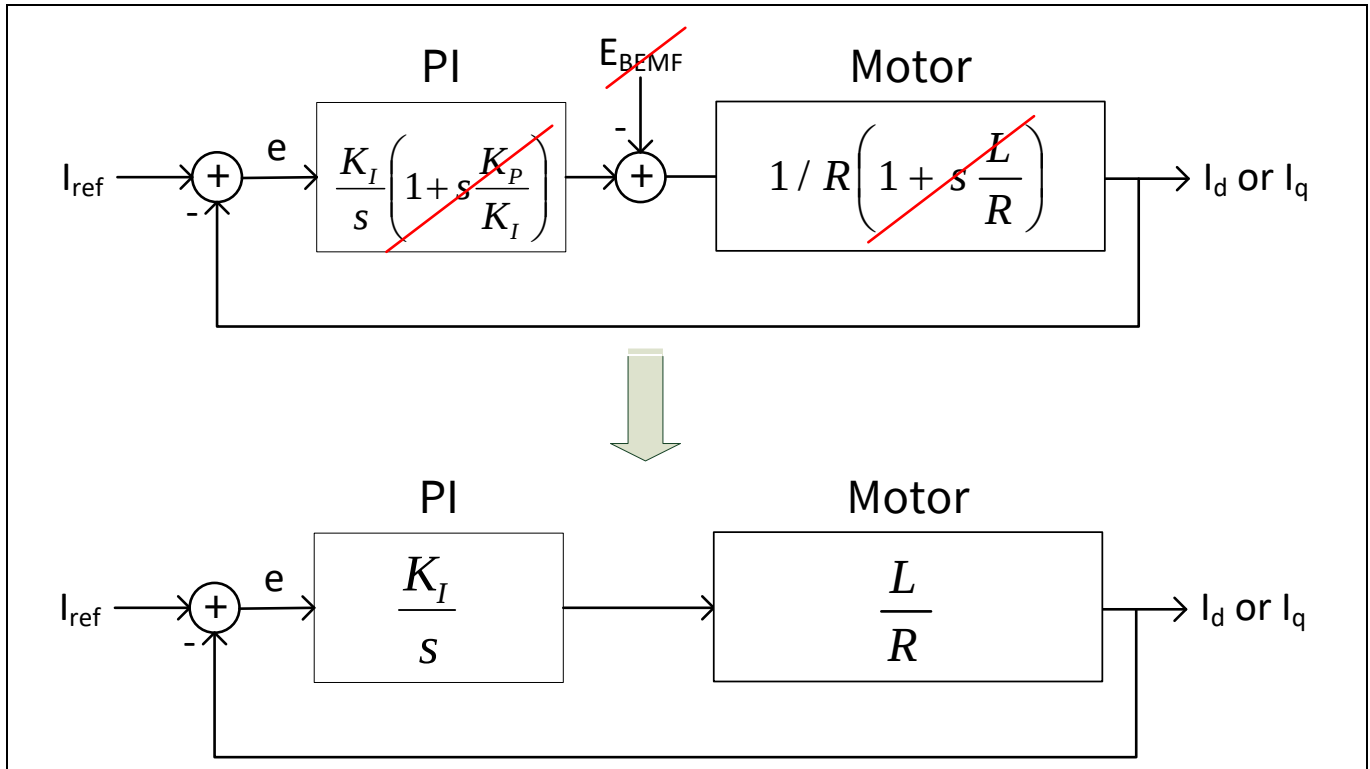


Figure 33 Simplified current control loop due to pole-zero cancellation

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-optimized 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.

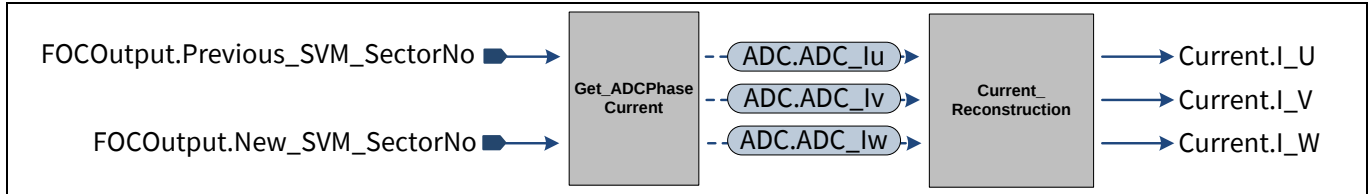


Figure 34 Current sensing and calculation functions

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 current 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 ADC conversions. Initial settings of the CCU80 and VADC modules for different current sensing techniques are listed in their respective sub-sections: Section 3.1 and Section 3.2.

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

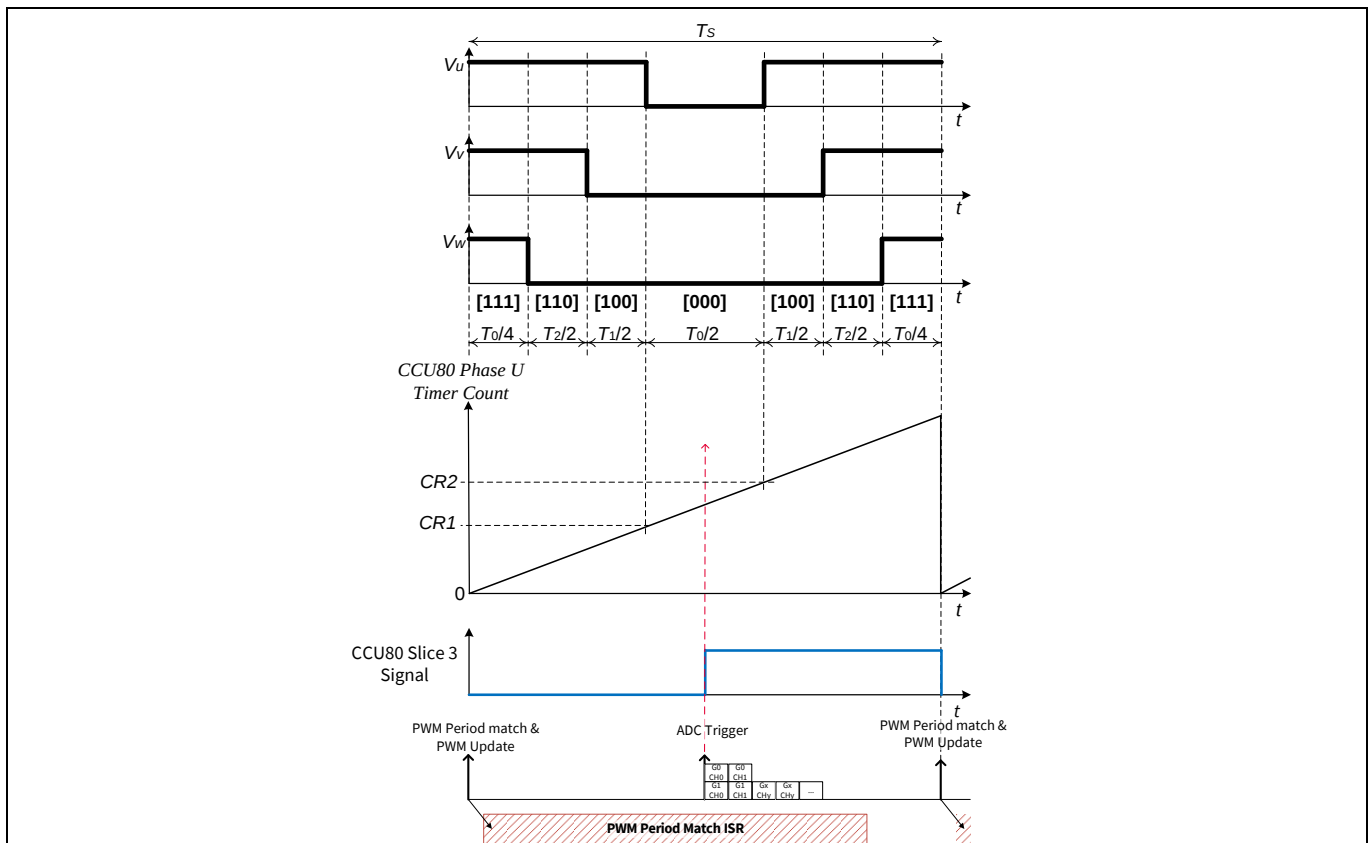


Figure 35 Three-shunt current sensing timing diagram using synchronous conversion ADC

Internal ADC gain feature

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

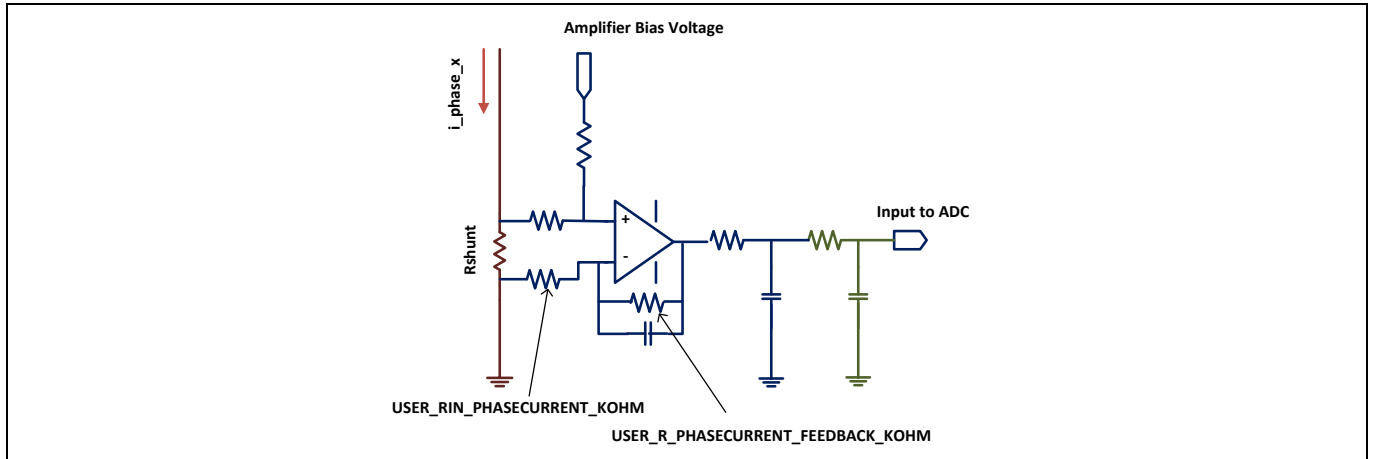


Figure 36 External phase current amplifier

Additionally, XMC™ MCU supports an analog gain stage for the ADC (VADC). With this feature, an external fast opamp is not required for phase current signals. This leads to cost saving in the BOM.

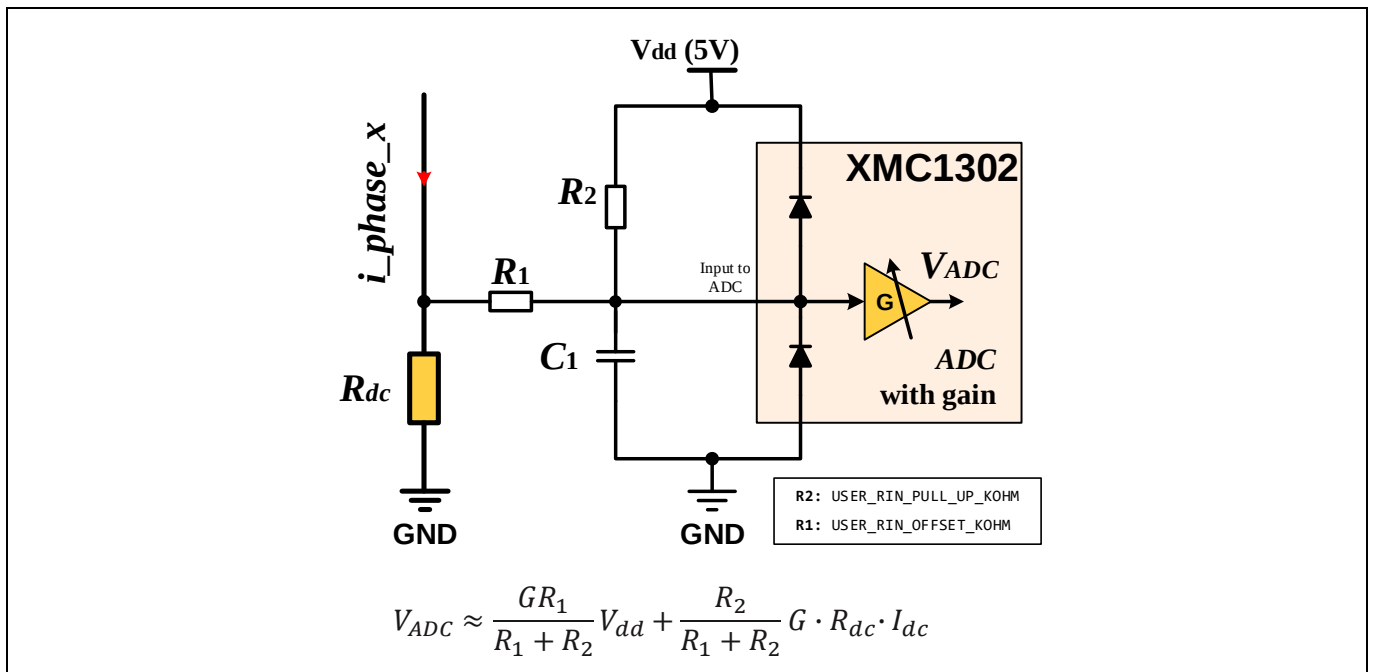


Figure 37 Phase current amplifier with on-chip gain

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 currents of the motors.

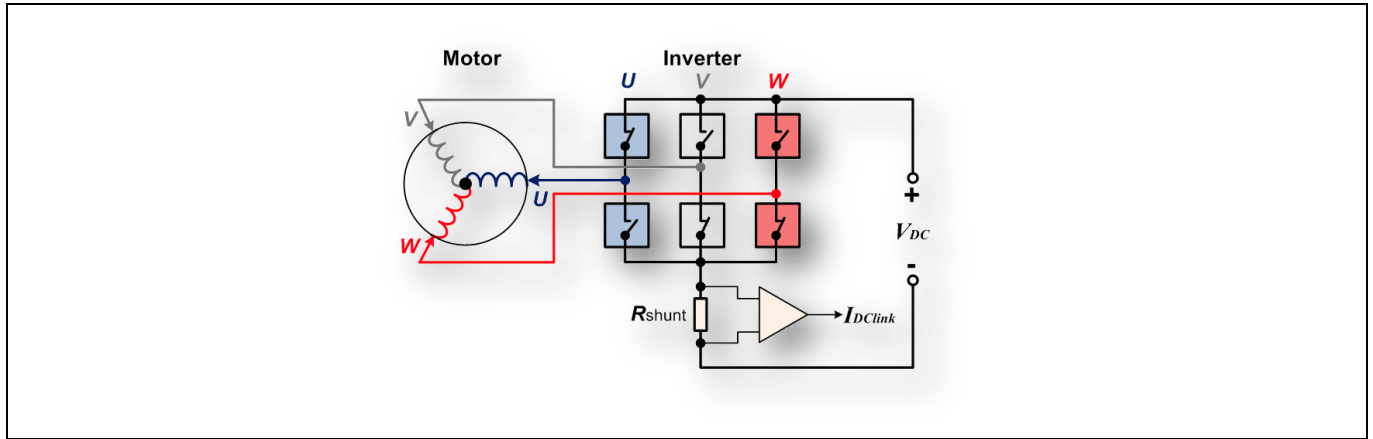


Figure 38 Single-shunt sensing technique

The pseudo zero vector (PZV) SVM is used to ensure that 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 a specific time as shown in Figure 39. ADC conversions are triggered at both the rising and falling edges of the CCU80 slice 3 signal (slice 2 in XMC4400). When a 4-segment SVM is used, the ADC conversion trigger points are also changed; refer to Figure 40.

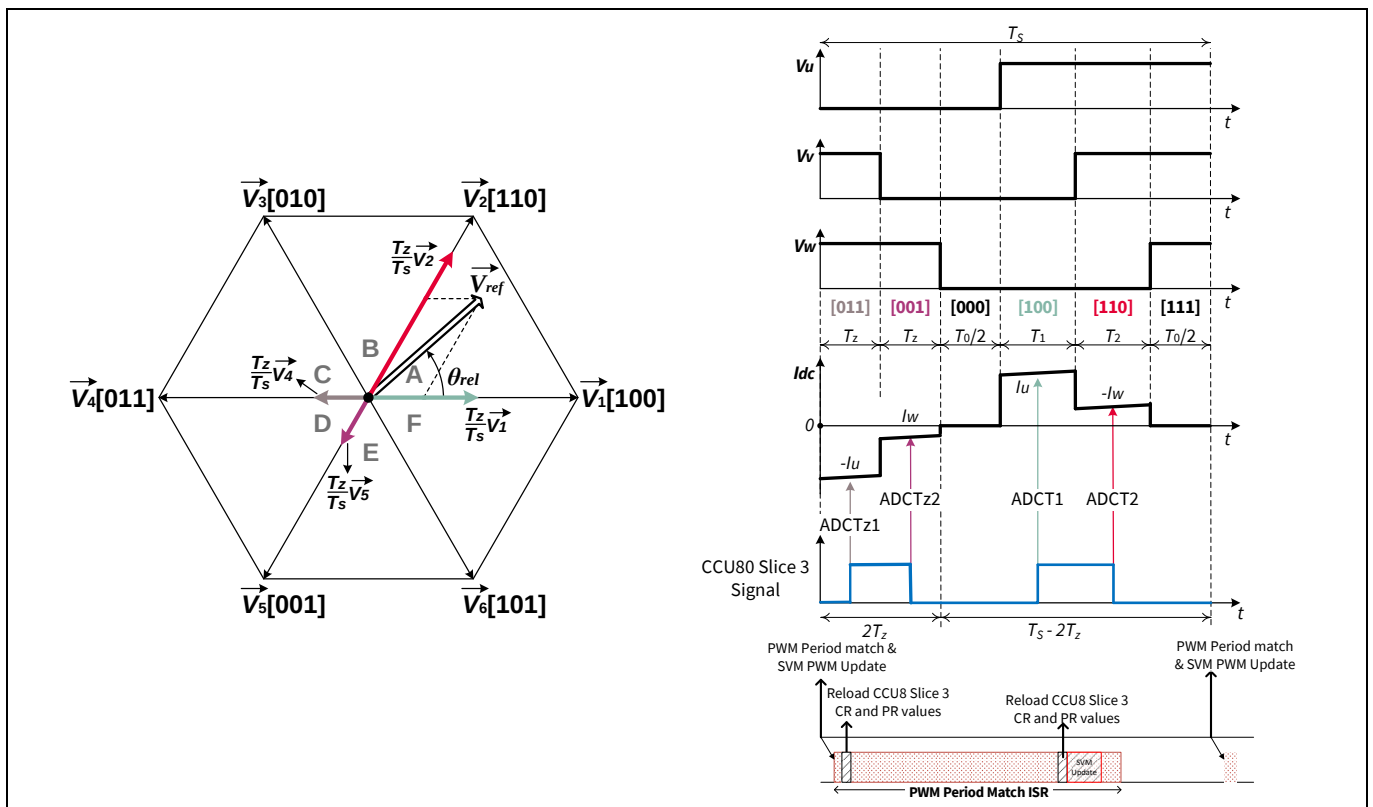


Figure 39 Single-shunt - 3-phase current sensing in sector A

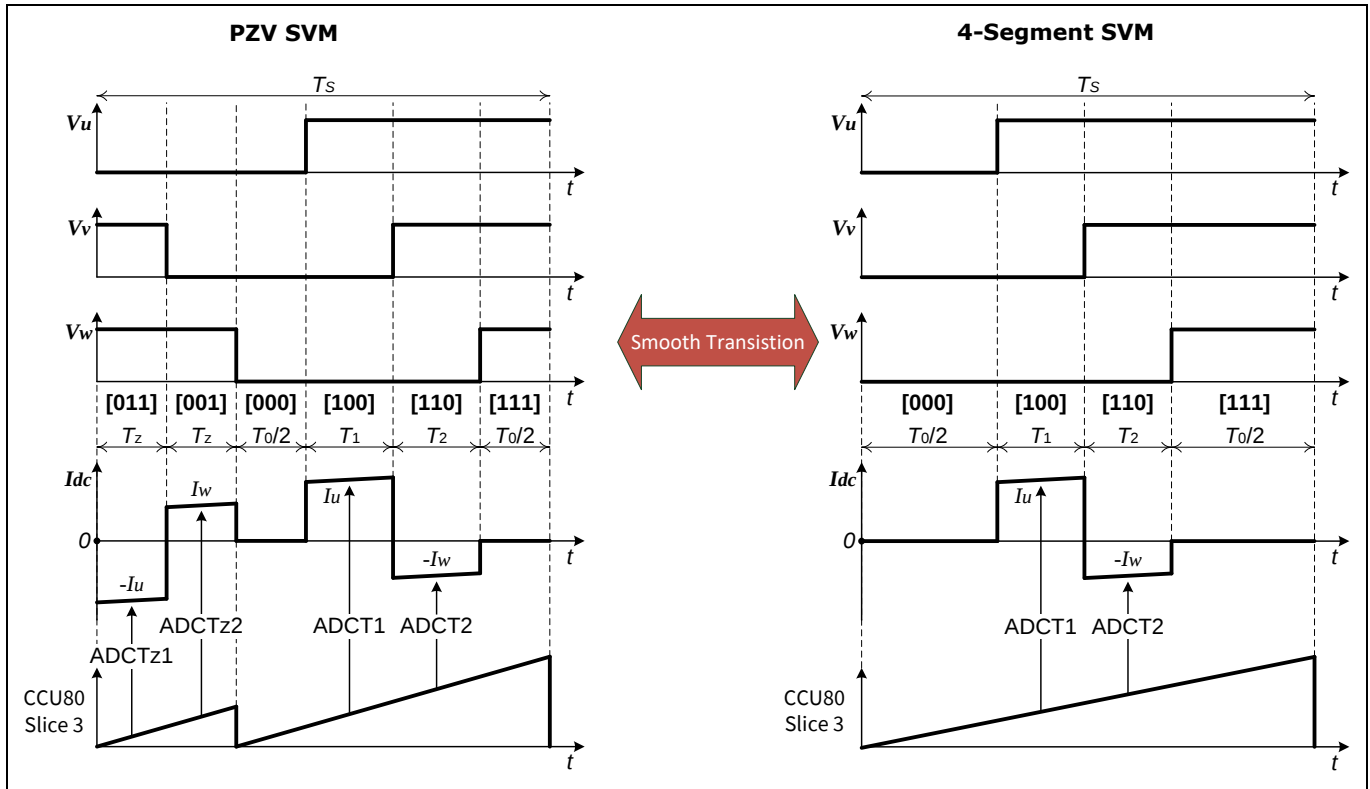


Figure 40 Smooth transition from PZV to 4-segment SVM in sector A

The VADC source interrupt event is enabled; 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 $\rightarrow I_U = (I_{ADCT1} - I_{ADCTz1}) * 2^3$
- Current $\rightarrow I_W = (I_{ADCT2} - I_{ADCTz2}) * 2^3$

In 4-segment SVM:

- Current $\rightarrow I_U = (I_{ADCT1} - I_{ADC_Bias}) * 2^4$
- Current $\rightarrow I_W = (I_{ADCT2} - I_{ADC_Bias}) * 2^4$

The two tables below show the initial settings of the VADC and CCU80 slice 3 for the 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, you can select two out of the three phase currents for the current reconstruction calculation. In the XMC4400/4800, all the three phases are used.

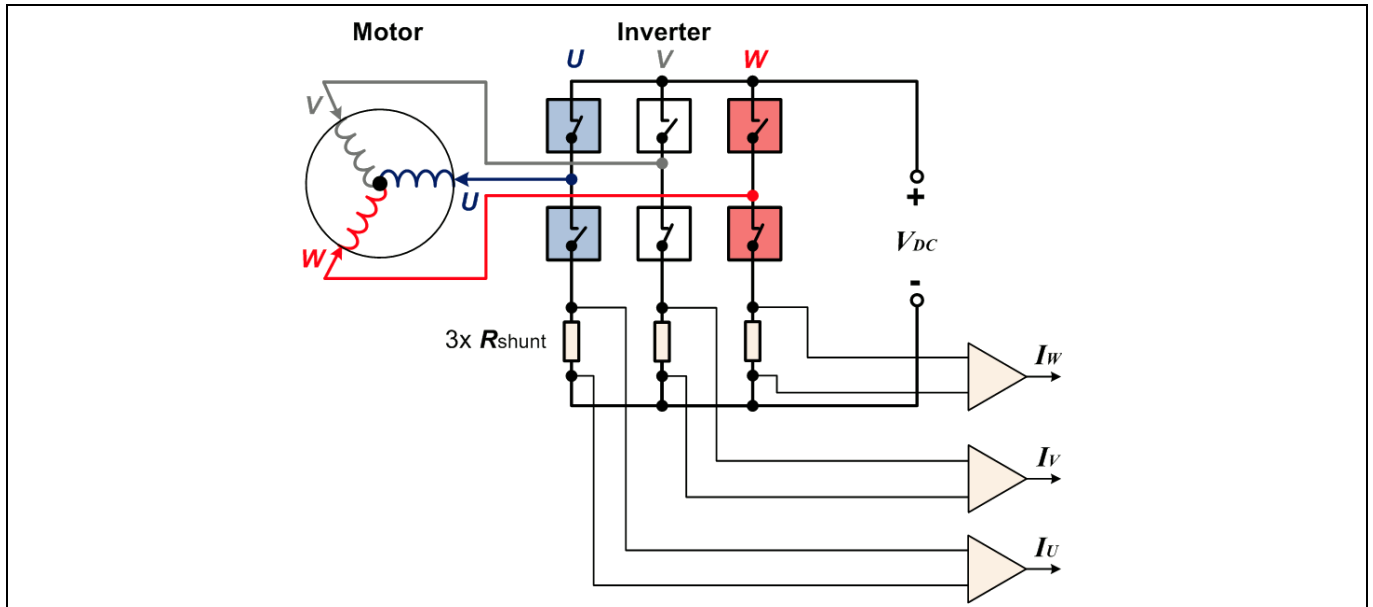


Figure 41 Three-shunt sensing technique

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; see [Figure 42](#). The current will always flow through the shunt resistor when the low-side switch is on and the high-side switch is off.

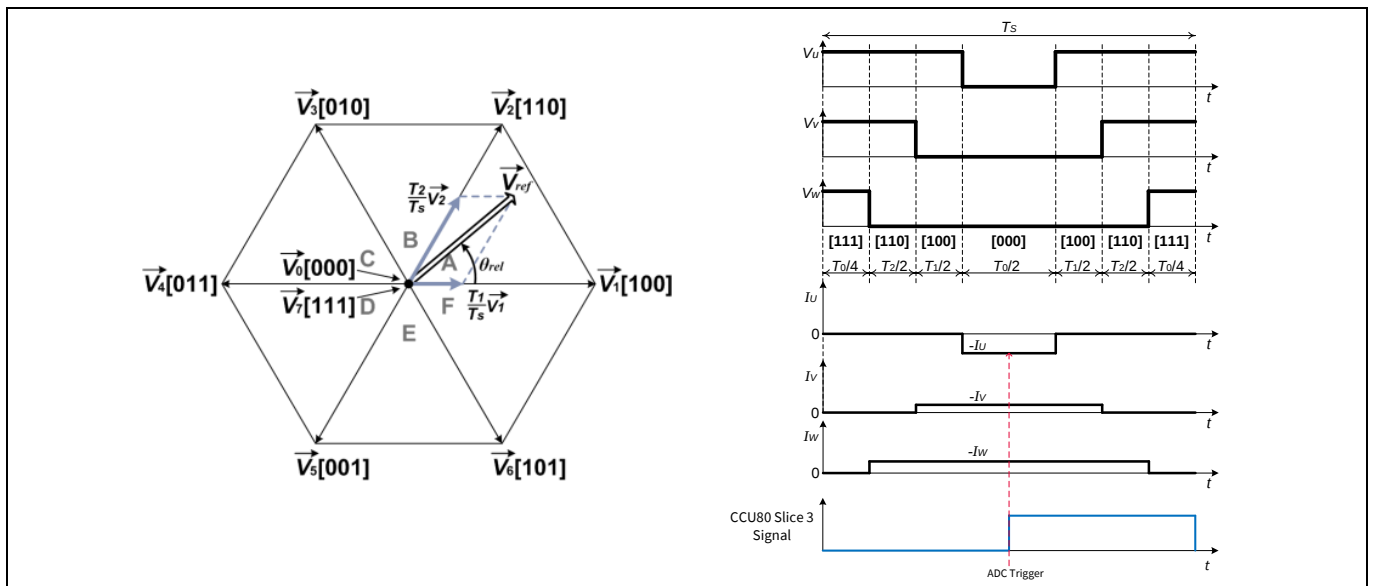


Figure 42 Three-shunt, 3-phase current sensing in sector A

Current sensing and calculation

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

- Current $\rightarrow I_U = (ADC_Bias_{I_u} - I_{ADC_Iu}) * 2^3$
- Current $\rightarrow I_V = (ADC_Bias_{I_v} - I_{ADC_Iv}) * 2^3$
- Current $\rightarrow I_W = (ADC_Bias_{I_w} - I_{ADC_Iw}) * 2^3$

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 sensing

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 sensing

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

Note:

1. XMC4800 uses CCU81.ST3A for ADC conversion trigger signal. It also uses CCU81 slice 3 for initial three-shunt setting.

3.2.1 Asynchronous conversion (only in XMC1300/XMC1400)

The term “asynchronous conversion” is related to the independence 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 applicable 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 is not implemented in the XMC4400/4800 family.

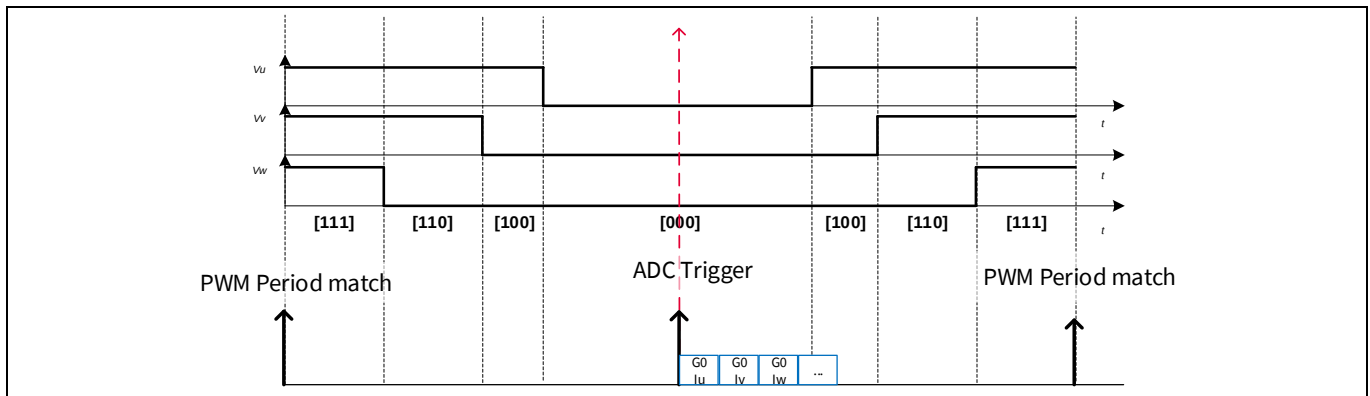


Figure 43 Asynchronous conversion sequence

3.2.2 Synchronous conversion

In XMC1300 and XMC1400 families, this implementation uses the following three 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, phases U and V). This method is not affected 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, and generates a higher torque. The determination of the phase with the small measurement window depends on the sector (see [Figure 44](#)).

PMSM FOC motor control using XMC™ MCUs

XMC1300/XMC1400 and XMC4400/XMC4800

Current sensing and calculation

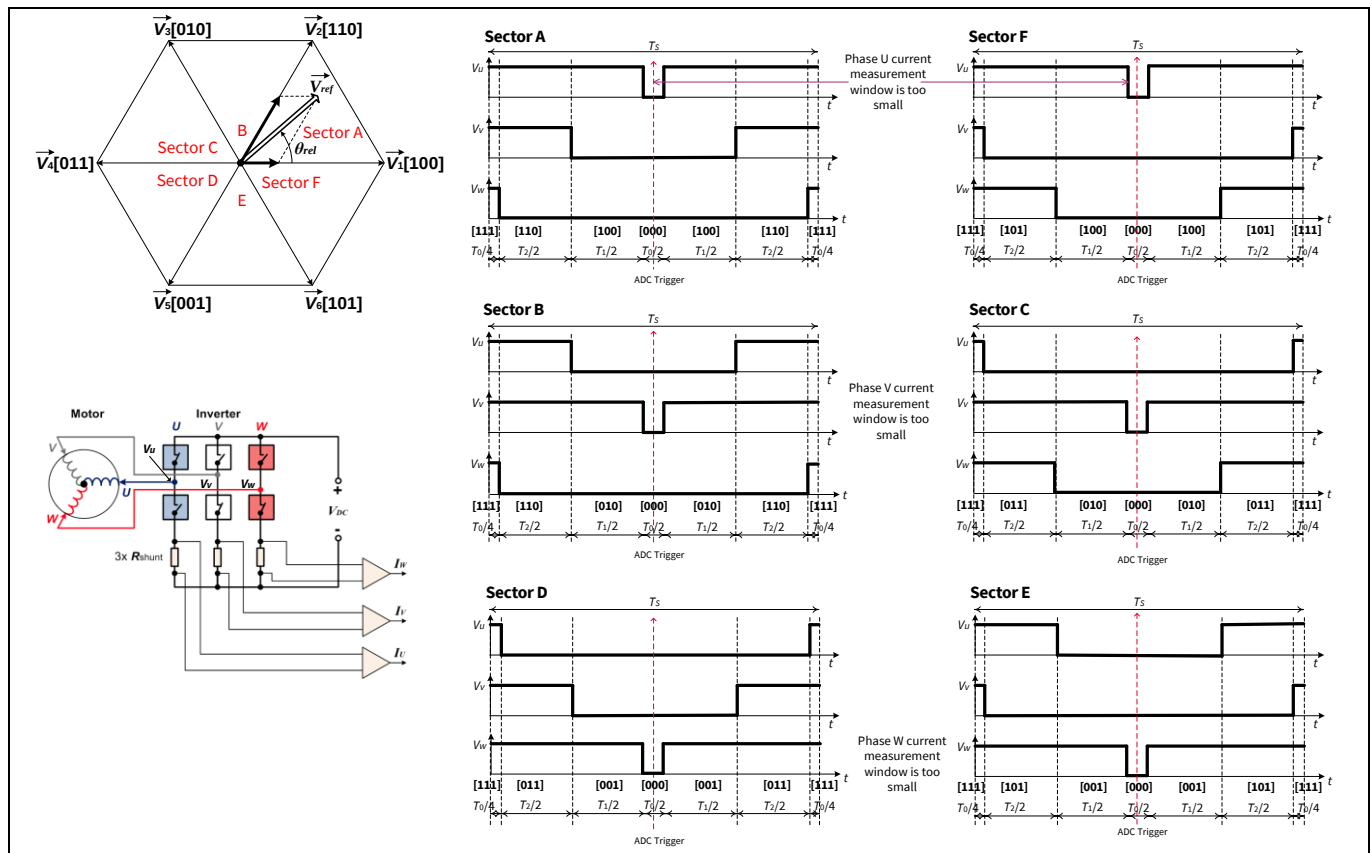


Figure 44 SVM sectors at high motor torque

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} . In such a case, the software switches from 3-leg shunt measurement to 2-leg shunt measurement. In three areas, the minimum measurement window falls below T_{min} at two phases. Figure 45 shows the area that can be measured with 3-leg or 2-leg shunt measurement.

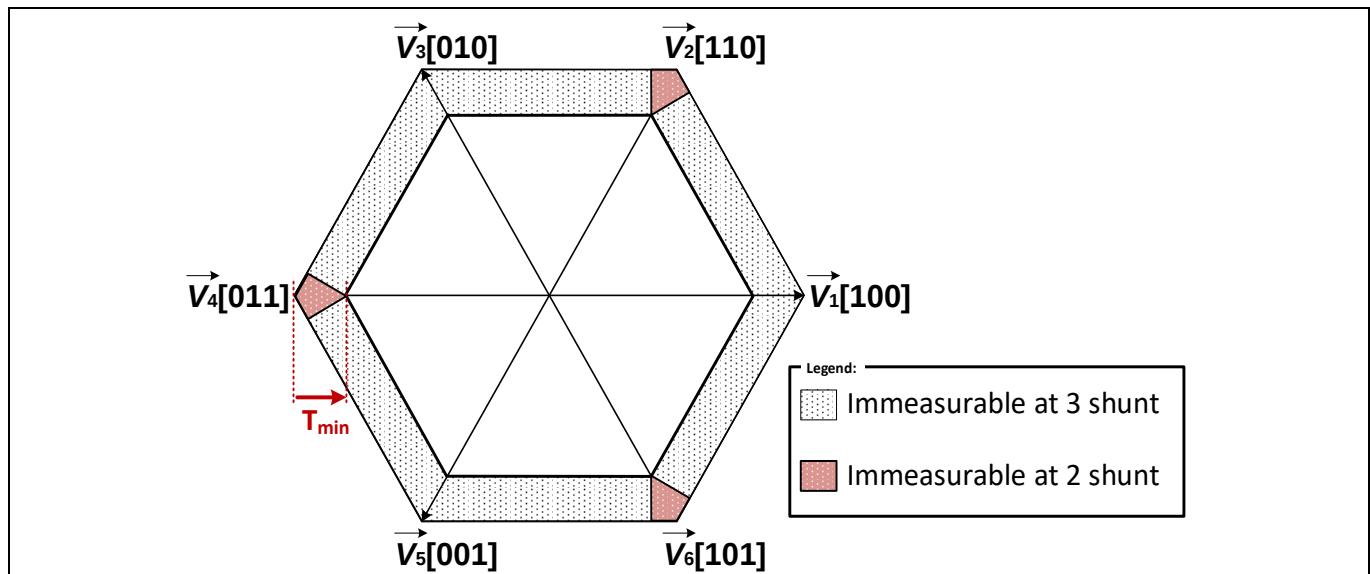


Figure 45 SVM 2/3 leg shunt immeasurable areas

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 avoids this doubling by overlapping group channels. Up to four pins are accessible from both groups.

In the XMC4400/4800 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 measurement implementation

Using the alias feature in the ADC module, you can assign different ADC input channels to be converted in parallel. In XMC1300/XMC1400, you can measure the two most non-critical phase currents for all the SVM sectors. For sectors A and F, you can assign I_V and I_W ADC channels, while in the XMC4400/4800 family, all three phases can be measured 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 is shown in the following figures. CH0 of the master (G0) and the slave (G1) are measured synchronously. After conversion, CH1 of the master (G0) and the slave (G1) are measured.

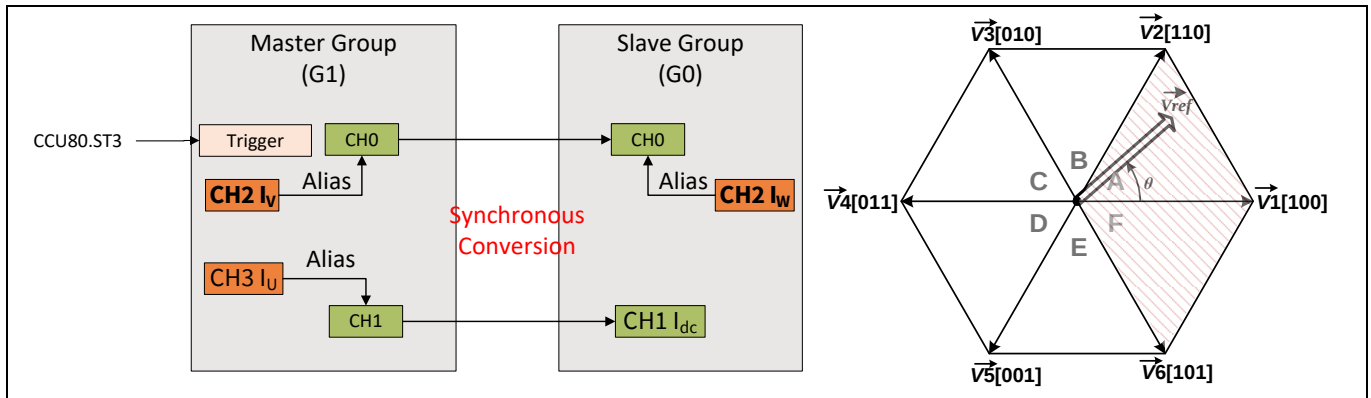


Figure 46 Synchronous conversion using alias feature in XMC1300 and XMC1400 – sectors A and F

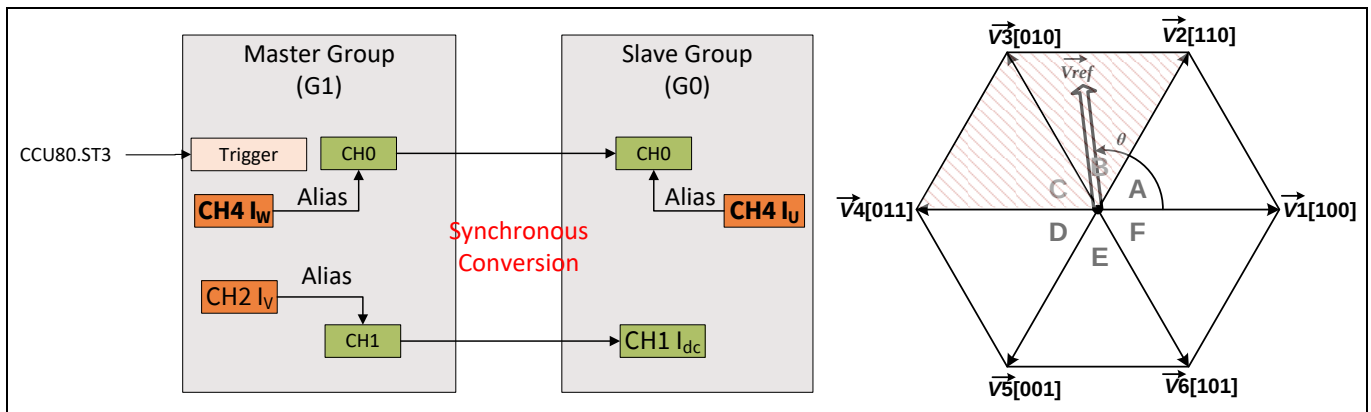


Figure 47 Synchronous conversion using alias feature in XMC1300 and XMC1400 – sectors B and C

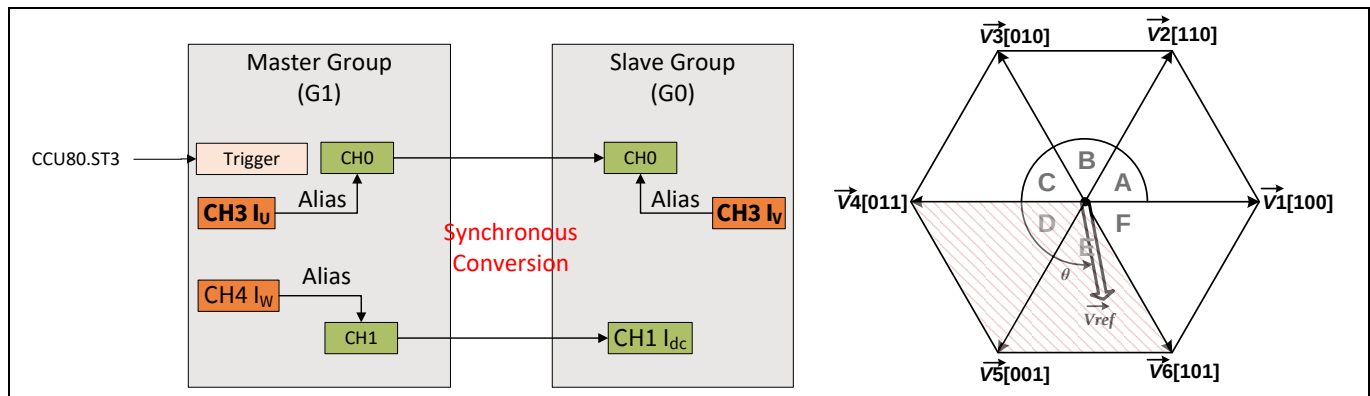


Figure 48 Synchronous conversion using alias feature in XMC1300 and XMC1400 – sectors D and E

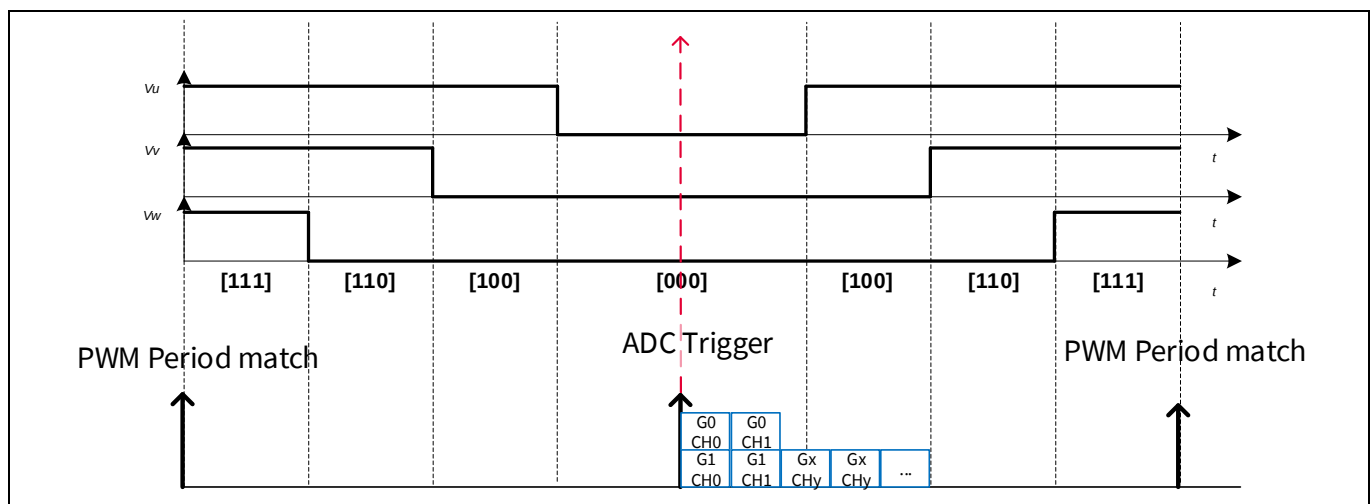


Figure 49 Synchronous conversion sequence in XMC1300/XMC1400

In XMC4400/4800, all the three phases are sensed synchronously at the trigger point.

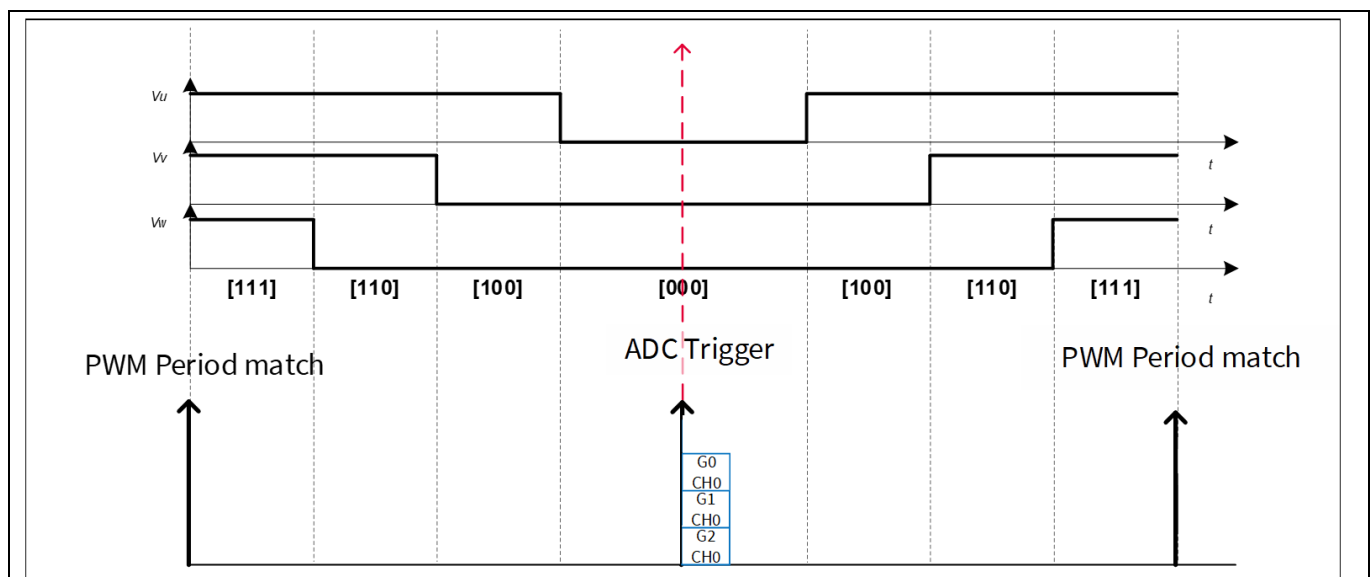


Figure 50 Synchronous conversion sequence in the XMC4400/4800 family

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 APIs are provided in the library.

Note: You must ensure 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)`

These APIs and the required parameters are described as follows:

Table 15 PLL_Imag() function

Name	<code>PLL_Imag(int32_t Vref_AngleQ31, int32_t I_Alpha_1Q31, int32_t I_Beta_1Q31)</code>	
Description	In XMC1300/XMC1400, this function starts the first CORDIC calculation of the sensorless estimator. In XMC4400/4800, this function makes the first FPU calculation and loads the result in the <code>PLL_Estimator</code> structure.	
Input parameters	<code>Vref_AngleQ31</code>	Angle of the voltage space vector
	<code>I_Alpha_1Q31</code>	Alpha coordinate of the current space vector
	<code>I_Beta_1Q31</code>	Beta coordinate of the current space vector
	<code>HandlePtr</code>	Pointer to the structure of <code>PLL_Estimator</code>
Return (only in XMC4400/4800)	<code>Current_I_Mag</code>	Current magnitude
	<code>Delta_IV</code>	To be provided as an input parameter for the second CORDIC calculation

Table 16 PLL_Imag_GetResult() function

Name	<code>PLL_Imag_GetResult(PLL_EstimatorType* const HandlePtr)</code>	
Description	In XMC1300/XMC1400, this function reads out the results of the first CORDIC calculation of the sensorless estimator. In XMC4400/4800, this function is not called.	
Input parameters	<code>HandlePtr</code>	Pointer to the structure of <code>PLL_Estimator</code>
Return (only in XMC1300/XMC1400)	<code>Current_I_Mag</code>	Current magnitude
	<code>Delta_IV</code>	To be provided as an input parameter for the second CORDIC calculation

Table 17 PLL_Vref() function

Name	<code>PLL_Vref(int32_t Delta_IV, uint32_t Vref32, int32_t PLL_UK, int32_t Phase_L, PLL_EstimatorType* const HandlePtr)</code>	
Description	In XMC1300/XMC1400, this function starts the second CORDIC calculation of the sensorless estimator. In XMC4400/4800, this function does the second FPU calculation and loads the result in the <code>PLL_Estimator</code> pointer.	

Input parameters	Delta_IV	Result of the first CORDIC calculation of the sensorless estimator
	Vref32	SVM voltage magnitude of the last PWM cycle
	PLL_Uk	PLL Estimator PI controller output
	Phase_L	Phase inductance of the motor stator winding
	HandlePtr	Pointer to the PLL_Estimator structure
Return (in XMC1300/XMC1400)	Current_I_Mag	Updated current magnitude
Return (in XMC4400/4800)	VrefxSinDelta	Used for the PLL_Estimator structure

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 XMC4400/4800, this function is not called.	
Input parameters	HandlePtr	Pointer to the PLL_Estimator structure
Return (only in XMC1300/XMC1400)	VrefxSinDelta	Used for the PLL_Estimator structure

Table 19 PLL_GetPosSpd() function

Name	PLL_GetPosSpd(PLL_EstimatorType* const HandlePtr)	
Description	Calculates and reads the rotor position and rotor speed from the sensorless estimator.	
Input parameters	HandlePtr	Pointer to the PLL_Estimator structure
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

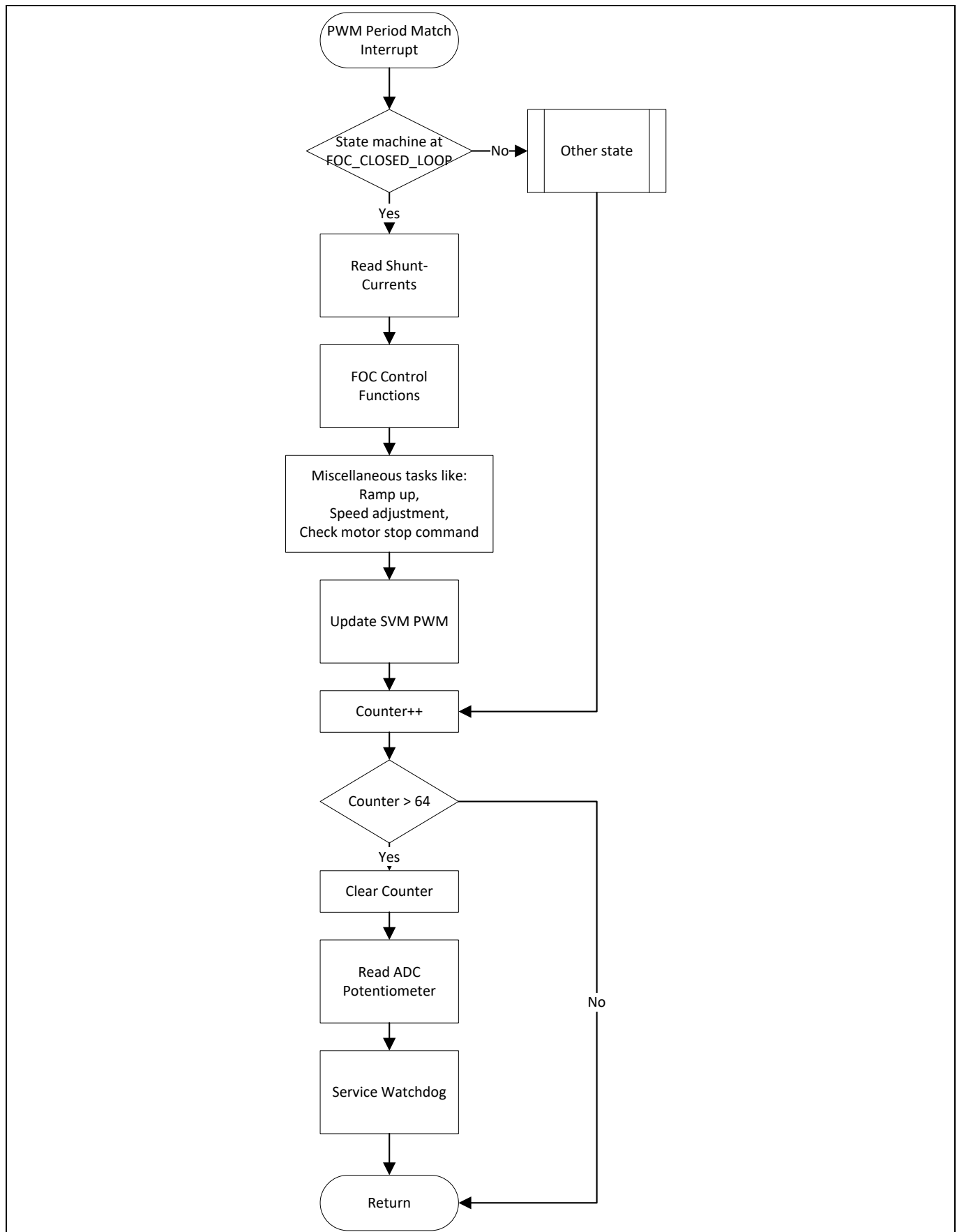
Interrupt events	Priorities	Comment
CTrap	0 (Highest priority)	Fault detection
ADC Queue Source	1	Only for single-shunt sensing (not implemented in XMC4400/4800). Disabled for three-shunt sensing.
PWM Period Match (Phase U)	2	State machine execution
Secondary Loop	3	APIs for low priority and low cycle frequency
Over/Under Voltage Protection	1	Event happens only if the DC bus is outside the voltage range

5.1 PWM period match interrupt

The PMSM_FOC state machine is executed in the Phase U PWM frequency period match ISR, which 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.

Interrupts

**Figure 51** PWM period match ISR flowchart

Interrupts

5.2 CTrap interrupt

When a trap condition is detected at the selected input pin (P0.12), CCU80 outputs are set to passive level and `Trap_Protection_INT()` is executed. In the ISR, 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 enabled only for single-shunt current sensing, and is triggered at the end of an ADC conversion. In the ISR, the ADC results are read.

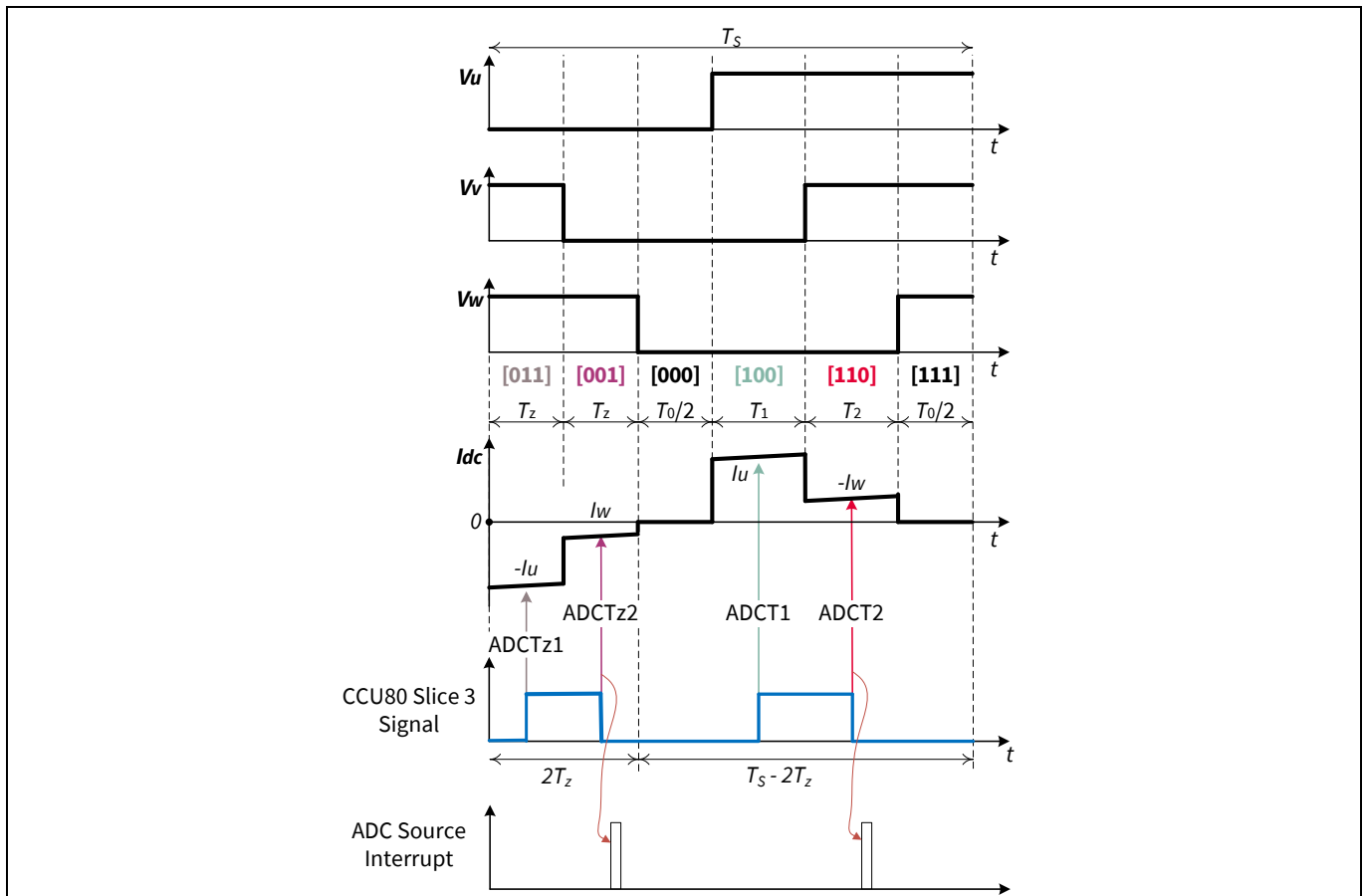


Figure 52 ADC source interrupt timing diagram

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 that it is not mandatory to have the frequency to be 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 is 20 kHz).

5.5 Overvoltage/undervoltage protection

This interrupt is triggered from the ADC only when the DC bus is outside 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; it reads the bias voltage of the ADC pins that are connected to motor phase currents. The state machine and the 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 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_XXX.h* file (see Section 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 `USER_STARTUP_SPEED_THRESHOLD_RPM` macro.

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 `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 brake command in this state will lead directly to a `MOTOR_STOP`.

Motor state machine

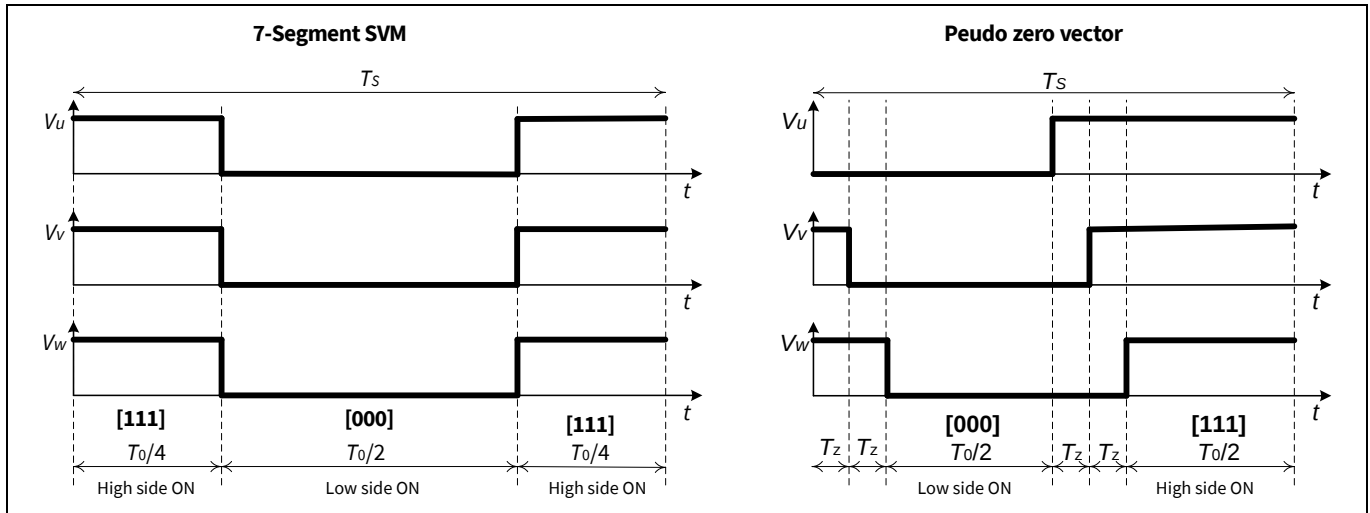


Figure 53 SVM outputs in motor brake condition

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 `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.

Motor state machine

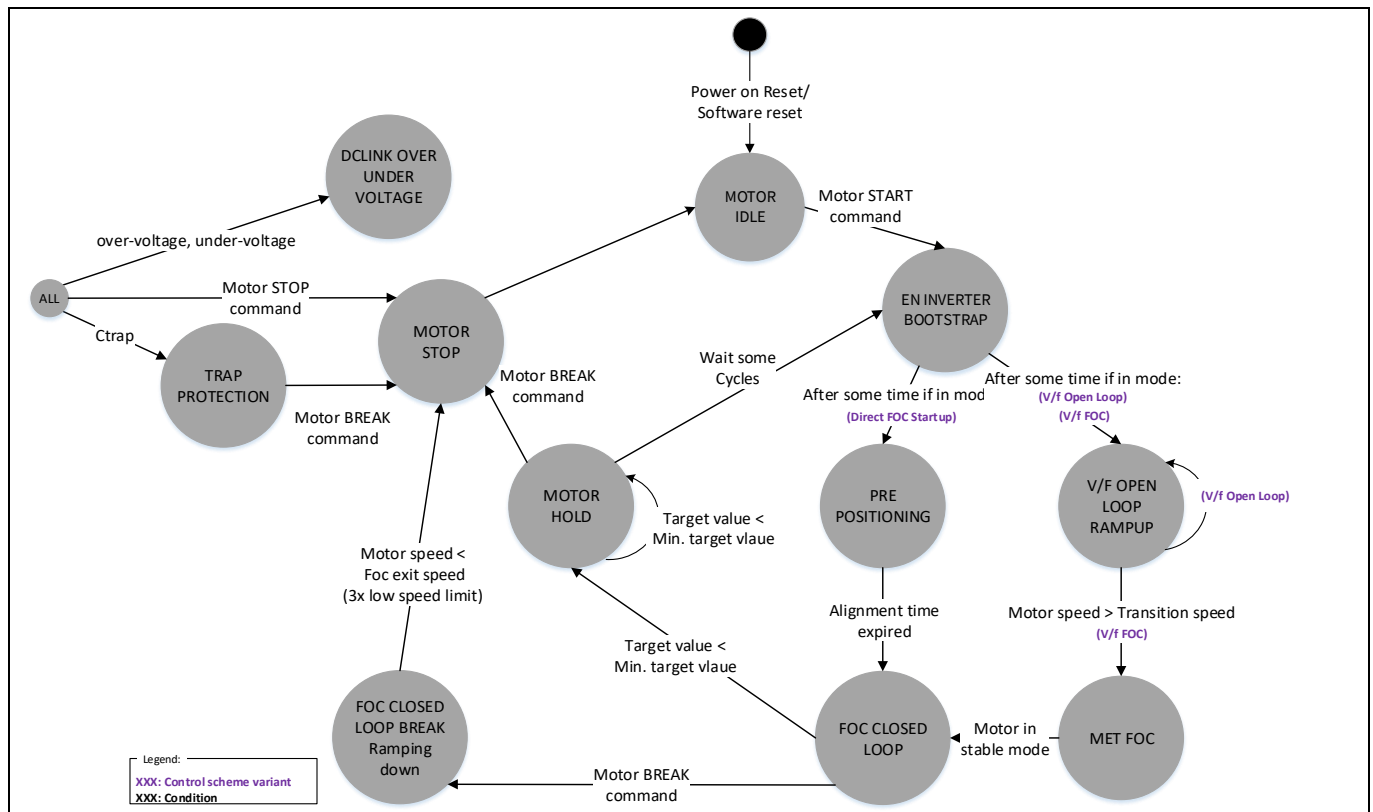


Figure 54 PMSM FOC state machine

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 three 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 overvoltage definition of 120% of the DC link voltage. Most of these configurations are essential, calculated, and should not be changed by you. Therefore, these settings 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 *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 the following:

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

This document refers to a combination of all three as the *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
```

- Selects a pre-defined hardware kit.

Options:

- KIT_XMC1X_AK_MOTOR_001 – Infineon XMC1000 Motor Control Application Kit
- KIT_XMC750WATT_MC_AK_V1 – XMC™ 750 Watt Motor Control Application Kit
- KIT_XMC14_BOOT_001 – XMC1404 CPU card for KIT_XMC1X_AK_MOTOR_001

Configuration

- 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 – XMC4400 CPU card for KIT_MOTOR_DC_250W_24V
- KIT_XMC4800PSOC_IOT – XMC4800 CPU card for KIT_MOTOR_DC_250W_24V

Current sensing

```
#define CURRENT_SENSING USER_THREE_SHUNT_SYNC_CONV
```

- Defines the current sensing technique used.

Options:

- USER_SINGLE_SHUNT_CONV – Single-shunt current sensing technique with pseudo zero vector PWM generation. See Section 3.1 (not implemented in XMC4400/4800)
- USER_THREE_SHUNT_ASSYNC_CONV – Three-shunt current sensing technique with ADC standard conversion. See Section 3.2 (not implemented in XMC4400/4800)
- USER_THREE_SHUNT_SYNC_CONV – Three-shunt current sensing technique with ADC synchronous conversion, refer to Section 3.2.1

FOC control schematic

```
#define MY_FOC_CONTROL_SCHEME SPEED_CONTROLLED_DIRECT_FOC
```

- Defines the FOC control scheme.

Options:

- SPEED_CONTROLLED_DIRECT_FOC – Direct FOC startup using speed control. See Section 2.3.2
- SPEED_CONTROLLED_VF_ONLY – Open-loop speed control. See Section 2.3.1
- SPEED_CONTROLLED_VF_MET_FOC – Open-loop startup to MET to closed-loop FOC speed control. See Section 2.3.2
- TORQUE_CONTROLLED_DIRECT_FOC – Direct FOC startup using torque control. See Section 2.3.3
- VQ_CONTROLLED_DIRECT_FOC – Direct FOC startup using voltage torque control. See Section 2.3.4

Input selection for target values

```
#define SETTING_TARGET_SPEED SET_TARGET_SPEED
```

- Defines 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:

Configuration

- `SET_TARGET_SPEED` – An API function is available to store the target value. Additionally, μ C/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 XMC4400/4800).

SVM switching schematic

```
#define SVM_SWITCHING_SCHEME STANDARD_SVM_7_SEGMENT
```

- Defines the SVM switching scheme.

Options:

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

μ C/Probe GUI selection

```
#define uCPROBE_GUI_OSCILLOSCOPE ENABLED
```

- Enables or disables the firmware support for Micrium μ C/Probe GUI and oscilloscope tool. If enabled, it is called in `pmsm_foc_controlloop_isr()`.

Protections and limitations

```
#define VDC_UNDER_OVERVOLTAGE_PROTECTION ENABLED
```

- Enables or disables DC link voltage protection.

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

- Sets limit check $\pm 20\%$ for overvoltage and undervoltage conditions.

```
#define OVERCURRENT_PROTECTION ENABLED
```

- Enables or disables DC link current protection (not implemented in XMC4400/4800).

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

- Maximum DC link current limit. This limit is checked if overcurrent protection is enabled. Once this limit is hit, the reference speed is reduced (see overcurrent protection in Section 2.9). This is not implemented in XMC4400/4800.

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

- Sets 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
```

- Enables or disables the watchdog timer feature.

Configuration

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`
- `KIT_XMC14_BOOT_001`
- `KIT_XMC1400_DC_V1`
- `KIT_XMC4400_DC_V1`
- `KIT_XMC4400_EE1_001`
- `KIT_XMC4800PSOC_IOT`

Note: You should adapt `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: You should adapt `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.

Configuration

- CUSTOM_MOTOR
- MAXON_MOTOR_267121
- NANOTEC_MOTOR_DB42S03

Note: You should adapt `INVERTERCARD_TYPE_PATH` according to the selection.

Configuration

7.1.3 Advanced user configuration

ADC-specific configurations

```
#define ADC_STARTUP_CALIBRATION
```

DISABLED

- Enables or disables the ADC startup calibration feature. Enable this feature if using XMC1302AA step (not implemented in XMC4400/4800).

```
#define ADC_ALTERNATE_REFERENCE
```

DISABLED

- Enables or disables the ADC alternative reference feature. If enabled, channel 0 is used as the reference (not implemented in XMC4400/4800).

```
#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 XMC4400/4800).

```
#define ADC_ALTERNATE_REF_SINGLESUNT
```

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 XMC4400/4800).

```
#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 ramps 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

- Enables or disables 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 must 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)

Configuration

FOC control

```
#define DQ_DECOUPLING
```

ENABLED

- Enables or disables the dq decoupling feature.

7.1.4 Torque control specific

These configurations are only available if:

MY_FOC_CONTROL_SCHEME is set to TRQVE_CONTROLLED_DIRECT_FOC

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

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

```
#define USER_IQ_REF_LOW_LIMIT
```

(0U)

- Sets the low limit of the reference torque current.

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

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

```
#define USER_IQ_RAMPUP
```

(10U)

- Torque current ramp-up steps used in the linear ramp generator.
- 1 step is $(1/32768) * I_{MAX_A}$ where I_{MAX_A} is equal to $(5.0 \text{ V} / (USER_R_SHUNT_OHM * OP_GAIN_FACTOR)) / 2$.

```
#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 $(5.0 \text{ V} / (USER_R_SHUNT_OHM * OP_GAIN_FACTOR)) / 2$.

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

- Defines 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.

Configuration

7.1.5 VQ control specific (V/f)

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

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

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

- Sets the low limit of the reference torque voltage.

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

- Scales 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)
```

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

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

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

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

- Defines the frequency to ramp up/ramp down V_q . In every `USER_VQ_RAMP_SLEWRATE * PWM` cycle, 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)
```

- Defines 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 MET threshold calculation, $Y[n] = Y[n-1] + (X[n] - Y[n-1]) >> \text{USER_MET_LPF}$

Configuration

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	XMC4800 value	Notes
#define TRAP_PIN	P0_12	P0_7	P5_0	External CCU80 CTrap input pin assignment
#define INVERTER_EN_PIN	P0_11	P0_12	P8_9	Connect to the enable pin of the inverter switch/gate drivers.
#define PHASE_U_HS_PIN	P0_0	P0_5	P1_15	–
#define PHASE_U_LS_PIN	P0_1	P0_2	P1_12	–
#define PHASE_V_HS_PIN	P0_7	P0_4	P1_14	–
#define PHASE_V_LS_PIN	P0_6	P0_1	P1_11	–
#define PHASE_W_HS_PIN	P0_8	P0_6	P1_13	–
#define PHASE_W_LS_PIN	P0_9	P0_11	P1_10	–

Configuration

Table 22 XMC™ alternate output pin register setting

Define	XMC1300/XMC1400 value	XMC4400/4800 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.
- See XMC1300 or XMC1400 reference manual [\[1\]](#) for alternate output setting.

Configuration

Table 23 Primary loop and SVM resources (CCU8)

Define	XMC1300/XMC1400 value	XMC4400 value	XMC4800 value
#define CCU8_MODULE	CCU80	CCU80	CCU81
#define CCU8_MODULE_PHASE_U	CCU80_CC80	CCU80_CC80	CCU81_CC80
#define CCU8_MODULE_PHASE_V	CCU80_CC81	CCU80_CC81	CCU81_CC81
#define CCU8_MODULE_PHASE_W	CCU80_CC82	CCU80_CC83	CCU81_CC82
#define CCU8_MODULE_ADC_TR	CCU80_CC83	CCU80_CC82	CCU81_CC83
#define CCU8_MODULE_PRESCALER_VALUE	(0U)	(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 prescaler affects the resolution. Changing the frequency should re-calculate this value. The lowest possible value should always be taken.

Table 24 Secondary loop resources (CCU4)

Define	XMC1300/XMC1400 value	XMC4400/4800 value
#define SECONDARY_LOOP_MODULE	CCU40	CCU41
#define SECONDARY_LOOP_SLICE	CCU40_CC42	CCU41_CC43
#define SECONDARY_LOOP_MODULE_NUM	0	1
#define SECONDARY_LOOP_SLICE_PRESCALER	(2U)	(1U)
#define SECONDARY_LOOP_SLICE_NUM	(2U)	(3U)
#define SECONDARY_LOOP_SLICE_SHADOW_TRANSFER_ENABLE_Msk	XMC_CCU4_SHADOW_TRANSFER_SLICE_2	XMC_CCU4_SHADOW_TRANSFER_SLICE_1

- Assign the CCU4 module timer slice for the secondary loop. The slice name, slice number, and slice shadow transfer enable mask must match. The prescaler affects 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.

Configuration

Note: See the reference manual for details of the connection between group channel number and pin.

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. The group name and number must match. Additionally, in the default the group for IU, IV, and IW must be the same. If you prefer to use two groups, it is necessary to have one XMC_VADC_QUEUE_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. See the reference manual for details of the connection between group channel number and pin. The only restriction in selecting the result register is that every register must have 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 three-shunt synchronous conversion for XMC4800 (VADC)

```
#define VADC_IU_GROUP          VADC_G0
#define VADC_IU_GROUP_NO      (0U)
#define VADC_IV_GROUP          VADC_G1
```

Configuration

```
#define VADC_IV_GROUP_NO (1U)
#define VADC_IW_GROUP VADC_G2
#define VADC_IW_GROUP_NO (2U)
#define VADC_IU_CHANNEL (0U)
#define VADC_IU_RESULT_REG (0U)
#define VADC_IV_CHANNEL (0U)
#define VADC_IV_RESULT_REG (1U)
#define VADC_IW_CHANNEL (4U)
#define VADC_IW_RESULT_REG (4U)
```

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 determines the ADC group that is used for measurement. The group name and number must match. The channel number is equivalent to a pin. See reference manual for details of the connection between group channel number and pin. 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, (2U) in XMC4800
#define VADC_VDC_RESULT_REG (5U) // (4U) in XMC4400, (2U) in XMC4800
```

- This configuration determines the ADC group that is used for measurement. The group name and number must match. The channel number is equivalent to a pin. See the reference manual for details of the connection between group channel number and pin. 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 determines the ADC group that is used for measurement. The group name and number must match. The channel number is equivalent to a pin. See the reference manual for details of the connection between group channel number and pin. 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, (3U) in XMC4800
```

Configuration

```
#define VADC_POT_RESULT_REG          (7U)    //(14U) in XMC4400 and XMC4800
```

- This configuration determines the ADC group that is used for measurement. The group name and number must match. The channel number is equivalent to a pin. See the reference manual for details of the connection between group channel number and pin. 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 configuration defines which module is used, the frequency via period counts 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. The slice name, slice number, and slice shadow transfer enable mask all must match. The prescaler is by default 0 to support the highest frequency.

```
#define DEBUG_PWM_0_PORT            XMC_GPIO_PORT1
```

```
#define DEBUG_PWM_0_PIN              (0U)
```

Configuration

```
#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. See the reference manual for more information.

```
#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. The slice name, slice number, and slice shadow transfer enable mask all must match. The prescaler 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. An associated port pin, alternate output, and CCU4 slice must be chosen. See the reference manual for more information.

Configuration

7.2.2 Inverter board configuration for current and voltage sensing

General

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

- Defines the PWM frequency in Hz. This is the fastest loop and the control loop. The main tasks of FOC are done in this loop or its fractions. 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)
```

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

Supply voltage

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

- Defines the nominal DC voltage in volts, used in the motor inverter board. It is used for scaling and limit check $\pm 20\%$ (default) for overvoltage and undervoltage conditions.

```
#define USER_DC_LINK_DIVIDER_RATIO (5.1f / (5.1f + 47.0f))
```

```
//In XMC4400 and XMC4800 (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 must be defined according to the switches and bridge drivers. A value that is too low leads to shorting of high-side and low-side switches while 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 must be adapted. Some drivers have implemented a bootstrap charge pump and do not require a bootstrap. If the time is too 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
```

- Defines 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

Configuration

```
#define GATE_DRIVER_INPUT_LOGIC PASSIVE_LOW
```

- 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)
```

- 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 opamp 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 overcurrent 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 `OP_GAIN_FACTOR` is done according to the XMC™ built-in gain factor: 1, 3, 6, and 12.
- Only hardware-available gain factors must be used.

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

- Offset resistor in kΩ as a floating number.
- The purpose is explained in [Figure 37](#).

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

- Offset resistor in kΩ as a floating number.
- The purpose is explained in [Figure 37](#).

Configuration

Current measurement: internal OP disabled

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

- RIN resistor value in the phase current amplifier in kΩ (see [Figure 36](#)).

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

- Feedback resistor value in the phase current amplifier in kΩ.
- With the USER_RIN_PHASECURRENT_KOHM, the gain of the current amplifier is calculated. See [Figure 36](#).

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

- RIN resistor value in the DC link current amplifier in kΩ. See [Figure 36](#).

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

- Feedback resistor value in the DC link current amplifier in kΩ. With 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 parameters

- ```
#define USER_MOTOR_R_PER_PHASE_OHM (6.8f)
```
- Defines the motor phase to neutral resistance in ohms.
- ```
#define USER_MOTOR_L_PER_PHASE_uH (3865.0f)
```
- Defines the motor phase to neutral stator inductance in micro μ H.
 - For IPMSM (interior permanent magnet synchronous motor) brushless DC motor, q-axis inductance (L_q) 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 the minimum allowed target speed. In sensorless motor control, it is mandatory to measure a phase current. At low torque (usually at low speed), it is not possible to provide 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 the 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 three-leg shunt measurement mode. If this value is exceeded, the two-leg shunt measurement is used. This threshold includes the current ringing and ADC measurement time.
- ```
#define USER_STARUP_SPEED_RPM (0U)
```
- Defines the initial speed for V/f open-loop in RPM.
- ```
#define USER_STARTUP_SPEED_THRESHOLD_RPM (500U)
```
- Defines 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.

Configuration

```
#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 motor 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 *pmsm_foc_motor_MAXON_MOTOR_267121.h* file contains related configurations.

PI settings for the speed controller

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

- Configures proportional gain for speed control block. Minimum 1; maximum 32767.

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

- Configures the integral gain for the speed control block. Minimum 0; maximum 32767.

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

- Configures the scaling factor of K_p and K_i . K_p and K_i are scaled up by $2^{PI_SPEED_SCALE_KPKI}$. $USER_RES_INC$ is a constant defined in the *pmsm_foc_const.h* file. The maximum value of $PI_SPEED_SCALE_KPKI$ is 15; minimum is $USER_RES_INC$.

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

- Configures the minimum output value of the integral buffer. Minimum –32768.

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

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

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

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

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

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

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

- This define increases 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)
```

- Configures the proportional gain for the torque control block. The gain value is determined by the equation $K_p = \omega_c L \times A$ (see Section 2.11).

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

- Configures the integral gain for the torque control block. The gain value is determined by the equation $K_i = RT_s K_p / L$ (see Section 2.11).

Configuration

```
#define PI_TORQUE_SCALE_KPKI (SCALING_CURRENT_KPKI)
- Configures the 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)
- Configures the minimum output value of the integral buffer. Minimum is -32768.

#define PI_TORQUE_IK_LIMIT_MAX (32767)
- Configures the maximum output value of the integral buffer. Maximum is 32767.

#define PI_TORQUE_UK_LIMIT_MIN (-32768)
- Configures the minimum output value of the torque PI control block. Minimum is -32768.

#define PI_TORQUE_UK_LIMIT_MAX (32767)
- Configures the maximum output value of the torque PI control block. Maximum is 32767.
```

PI settings for the flux controller

```
#define PI_FLUX_KP (USER_DEFAULT_IQID_KP)
- Configures the proportional gain for the flux control block. The gain value is determined by the equation  $K_p = \omega_c L \times A$  (see Section 2.11).

#define PI_FLUX_KI (USER_DEFAULT_IQID_KI >> 0)
- Configures the integral gain for the flux control block. The gain value is determined by the equation  $K_i = RT_S K_p / L$  (see Section 2.11).

#define PI_FLUX_SCALE_KPKI (SCALING_CURRENT_KPKI + 0)
- Configures the 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)
- Configures the minimum output value of the integral buffer. Minimum is -32768.

#define PI_FLUX_IK_LIMIT_MAX (32767)
- Configures the maximum output value of the integral buffer. Maximum is 32767.

#define PI_FLUX_UK_LIMIT_MIN (-32768)
- Configures the minimum output value of the flux PI control block. Minimum is -32768.

#define PI_FLUX_UK_LIMIT_MAX (32767)
- Configures the maximum output value of the flux PI control block. Maximum is 32767.
```

PI settings for the PLL Estimator controller

```
#define PI_PLL_KP (1U << 8)
- Configures the proportional gain for the PLL Estimator control block. Minimum is 1; maximum is 32767.

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

Configuration

- Configures the integral gain for the PLL Estimator control block. Minimum is 0; maximum is 32767.

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

- Configures the scaling factor of K_p and K_i . K_p and K_i are scaled up by $2^{\text{PI_PLL_SCALE_KPKI}}$.
- `USER_RES_INC` is a constant defined in *pmsm_foc_feature_config.h*. Minimum value of `PI_PLL_SCALEKPKI` is 15.

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

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

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

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

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

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

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

- Configures the maximum output value of the PLL Estimator PI control block. Maximum 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 25 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 only once.
	Phase_R	Signed 32-bit	Motor resistance per phase	
	Phase_L_Scale	16-bit/8–18	Scaling for inductance; Real inductance in SW = $\text{Phase_L} / (2^{\text{Phase_L_Scale}})$	
	CCU8_Period	16-bit	CCU8 period. CCU8 PWM 1 kHz to 90 kHz	Initialize only once.
	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 the sensorless estimator	
	BEMF2	Signed 32-bit	BEMF of the sensorless estimator	
	overcurrent_factor	16-bit/0–4096	Used to reduce the motor speed when overcurrent 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 the 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 the Iq PI controller from the speed PI output 1: Reference of the Iq PI controller = Ref_Iq from external	

8.2 FOC control module output data structure

This data structure defines the output variables in FOC functions.

Table 26 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 following table shows the data structures of the variables used in FOC control functions.

Table 27 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 the last PWM cycle
	Vref_AngleQ31	Signed 32-bit	SVM voltage angle of the 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 28 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 the sector A to F	
	PreviousSectorNo	16-bit/0–5	To keep track of the sector number in the 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 29 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 the compare match value for CCU8 slice 3 timer	
	ADC_Bias	16-bit/0–4095	Bias of single-shunt current input	Single-shunt current sensing technique
	ADC_ResultTz1	16-bit/0–4095	ADC value of the first phase current ADC sampling	
	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	

PMSM FOC software data structure

Category (Structure)	Variable Name	Data type/range	Description	Remark
	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 the 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 the compare match value for CCU8 slice 3 timer	
	Result_Flag	16-bit / 0 – 2	To indicate that the 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 the ADC result of the potentiometer	
	ADC_DCLink	0–4095	Store the ADC result of the DC link voltage	
	ADC_IDCLink	0–4095	Store the ADC result of the average DC link current	

9 PMSM FOC software API functions

This chapter describes the PMSM FOC software API functions. 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 30 List of APIs

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

Type		API function name	Execute in SRAM	Execute in flash
		<code>pmsm_foc_secondaryloop_callback()</code>		Yes
Over_under_voltage_isr		<code>pmsm_foc_disable_inverter()</code>		Yes

9.1 User functions

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

`pmsm_foc_motor_start ()`

This function is called to start the motor.

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

Table 31 `pmsm_foc_motor_start()`

Input parameters	–	–
Return	–	–
Updated variables	<code>Motor.State</code>	<code>EN_INVERTER_BOOTSTRAP</code>

`pmsm_foc_motor_stop ()`

This function 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 `pmsm_foc_set_motor_target_speed/torque`.

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

Table 32 `pmsm_foc_motor_stop()`

Input Parameters	–	–
Return	–	–
Updated Variables	<code>Motor.State</code>	<code>MOTOR_STOP</code>

`pmsm_foc_motor_brake ()`

This function is called to ramp down the motor.

Table 33 `pmsm_foc_motor_stop()`

Input parameters	–	–
Return	–	–
Updated variables	<code>Motor.State</code>	<code>FOC_CLOSED_LOOP_BRAKE</code> if state isn't <code>TRAP_PROTECTION</code>

`pmsm_foc_set_motor_target_speed()`

This function 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, use the `pmsm_foc_motor_stop()` function. The MCU will approach the reference speed to the target speed using the ramp generator.

PMSM FOC software API functions

This function is available only 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 34 **pmsm_foc_set_motor_target_speed ()**

Input parameters	(uint32) <code>motor_target_speed</code>	-
Return	-	-
Updated variables	<code>Motor.Target_Speed</code>	-

pmsm_foc_set_motor_target_torque ()

This function 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, use the `pmsm_foc_motor_stop()` function. 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 35 **pmsm_foc_set_motor_target_torque ()**

Input parameters	(uint32) <code>motor_target_torque</code>	-
Return	-	-
Updated variables	<code>Motor.Target_Torque</code>	-

pmsm_foc_set_motor_target_voltage ()

This function 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, use the `pmsm_foc_motor_stop()` function. The MCU will approach the reference speed to the target speed using the ramp generator.

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

Table 36 **pmsm_foc_set_motor_target_voltage ()**

Input parameters	(uint32) <code>motor_target_voltage</code>	-
Return	-	-
Updated variables	<code>Motor.Target_Voltage</code>	-

pmsm_foc_get_motor_speed ()

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

Table 37 **pmsm_foc_get_motor_speed ()**

Input parameters	-	-
Return	Speed in RPM	-
Updated variables	-	-

PMSM FOC motor control using XMC™ MCUs

XMC1300/XMC1400 and XMC4400/XMC4800

PMSM FOC software API functions

pmsm_foc_get_Vdc_link ()

This function returns the voltage value of DC bus.

Table 38 **pmsm_foc_get_Vdc_link ()**

Input parameters	–	–
Return	Voltage in V	–
Updated variables	–	–

9.2 Control loop ISR

9.2.1 Startup

The following functions are executed for direct FOC startup.

pmsm_foc_bootstrap_charge ()

This function performs a motor brake and charge the gate driver bootstrap capacitor. At the same time, the motor phase currents bias are read and updated.

This function is called in the `EN_INVERTER_BOOTSTRAP` motor state.

Table 39 **pmsm_foc_bootstrap_charge ()**

Input parameters	None	–
Return	None	–
Updated variables	<code>ADC.ADC_Bias_Iu</code>	ADC bias value for phase U current
	<code>ADC.ADC_Bias_Iv</code>	ADC bias value for phase V current
	<code>ADC.ADC_Bias_Iw</code>	ADC bias value for phase W current
	<code>Motor.State</code>	Updated motor state to <code>PRE_POSITIONING</code> once the motor start command is received
	<code>Motor.Transition_Status</code>	Motor status flag for transition. Change this flag to <code>MOTOR_TRANSITION</code> once the 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 40 **pmsm_foc_enable_inverter ()**

Input Parameters	None	–
Return	None	–
Updated Variables	None	–

pmsm_foc_disable_inverter ()

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

Table 41 **pmsm_foc_disable_inverter ()**

Input Parameters	None	–
Return	None	–
Updated Variables	None	–

pmsm_foc_directfocrotor_pre_positioning ()

This function sets the initial rotor position/alignment, and 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 42 **pmsm_foc_directfocrotor_pre_positioning ()**

Input parameters	None	–
Return	None	–
Updated variables	<code>Current.I_u</code> , <code>Current.I_v</code> , <code>Current.I_w</code>	Current reconstruct function is called and <code>I_u</code> , <code>I_v</code> , and <code>I_w</code> are updated
	<code>Clarke_Transform.I_Alpha</code> , <code>Clarke_Transform.I_Beta</code>	<code>I_Alpha</code> and <code>I_Beta</code> are updated
	<code>Car2Polar.Vref</code>	Increases the <code>Vref</code> value gradually for SVM
	<code>Motor.State</code>	Updates motor state to <code>FOC_CLOSED_LOOP</code> once the preposition timer expires
	<code>FOCInput</code>	Initializes <code>FOCInput</code> variables before entering <code>FOC_CLOSED_LOOP</code>
	<code>PI_Speed</code> <code>PI_Torque</code> <code>PI_Flux</code> <code>PI_PLL</code>	Initializes PI controller variables before entering <code>FOC_CLOSED_LOOP</code>

pmsm_foc_vf_foc_openloop_rampup ()

This function 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 mode. Once the motor speed reaches the defined `VF_TRANSITION_SPEED`, the motor state is changed to `MET_FOC`.

Table 43 **pmsm_foc_vf_foc_openloop_rampup()**

Input parameters	None	–
Return	None	–
Updated variables	<code>Current.I_u</code> , <code>Current.I_v</code> , <code>Current.I_w</code>	Current reconstruct function is called and <code>I_u</code> , <code>I_v</code> , and <code>I_w</code> are updated
	<code>Clarke_Transform.I_Alpha</code> , <code>Clarke_Transform.I_Beta</code>	<code>I_alpha</code> and <code>I_Beta</code> are updated
	<code>Car2Polar.Vref</code>	Increases the <code>Vref</code> value gradually for SVM
	<code>Motor.State</code>	Updates the motor state to <code>MET_FOC</code> once the motor speed ramps up to a predefined value
	<code>FOCInput</code>	Initializes <code>FOCInput</code> variables before entering <code>MET_FOC</code>
	<code>PLL_Estimator</code>	Calls the <code>PLL_Estimator</code> API and updates the variables before entering <code>MET_FOC</code>

pmsm_foc_vf_smooth_transition_to_foc ()

This function is called in the `MET_FOC` motor state, and is used for MET control strategy. Once the stator flux is perpendicular to the rotor flux, this returns the status as `MOTOR_STABLE`.

Table 44 **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 the 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 are updated
	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	Clarke Transform function is called and I_alpha and I_Beta are updated
	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar Vref and Vref_AngleQ31 are updated
	FOCInput	FOCInput variables are 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 the 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 45 **pmsm_foc_misc_works_of_met()**

Input parameters	Motor.Mode_Flag	Motor mode status
Return	None	–
Updated variables	Motor.State	Updates the motor state to FOC_CLOSED_LOOP once Motor.Mode_Flag == MOTOR_STABLE
	FOCInput	Initializes the FOCInput variables before entering FOC_CLOSED_LOOP
	PI_Speed PI_Torque PI_Flux PI_PLL	Initializes the PI controller variables before entering FOC_CLOSED_LOOP
	PLL_Estimator.RotorAngleQ31	Initializes the rotor angle for the first FOC PWM cycle before entering FOC_CLOSED_LOOP

9.2.2 InOut handling

pmsm_foc_get_IDCLink_current ()

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

Table 46 pmsm_foc_get_IDCLink_current ()

Input parameters	–	–
Return	–	–
Updated variables	ADC.ADC_IDCLink	–

pmsm_foc_get_adcphasecurrent ()

This function is used only 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 channel settings are also updated according to the SVM sector (only in XMC1300/XMC1400).

Table 47 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 ()

This function implements the linear ramp generator for speed control.

Table 48 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 ()

This function implements the linear ramp generator for torque control.

Table 49 **pmsm_foc_linear_torque_ramp_generator()**

Input parameters	<code>current_set</code>	Target torque value
	<code>inc_step</code>	Torque ramp-up rate
	<code>dec_step</code>	Torque ramp-down rate
Return	<code>FOCInput.Ref_Iq</code>	Motor reference torque for PI torque controller
Updated variables	–	–

pmsm_foc_linear_vq_ramp_generator ()

This function implements the linear ramp generator for Vq control.

Table 50 **pmsm_foc_linear_vq_ramp_generator()**

Input parameters	<code>set_val</code>	Target Vq value
	<code>inc_step</code>	Vq ramp-up rate
	<code>dec_step</code>	Vq ramp-down rate
Return	<code>FOCInput.Vq</code>	Reference Vq for Cartesian-to-polar transformation
Updated variables	–	–

pmsm_foc_scurve_ramp_generator ()

This function implements the S-curve ramp generator for speed control.

Table 51 **pmsm_foc_scurve_ramp_generator()**

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

pmsm_foc_svpwm_update ()

This function executes the SVM algorithm and updates the 3-phase PWM duty cycles.

Table 52 **pmsm_foc_svpwm_update ()**

Input parameters	Amplitude	Amplitude of the voltage space vector
	Angle	Angle of the 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 function is called only in the single-shunt current sensing technique. It is to set the first two trigger points for ADC conversion. The settings of the trigger points are the pre-calculated constants in the configuration file.

Table 53 **pmsm_foc_adctz12_triggersetting()**

Input parameters	–	–
Return	–	–
Updated variables	–	–

pmsm_foc_adc34_triggersetting ()

This function is called only in the single-shunt current sensing technique. This function sets the third and fourth trigger points for ADC conversion.

Table 54 **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 ()

This function ramps up the motor in speed/torque/Vq, and updates the reference speed or reference torque or Vq based on the ramping rate.

Table 55 pmsm_foc_misc_works_of_foc()

Input parameters	Motor.Mode_Flag	Motor mode status
Return	None	–
Updated variables	Motor.State	This variable is updated the motor state to MOTOR_HOLD if the motor stop command received
	Motor.Ref_Speed	This variable is updated if the SPEED_CONTROLLED_DIRECT_FOC or SPEED_CONTROLLED_VF_MET_FOC control scheme is selected
	FOCInput.Ref_Iq	This variable is updated if the TORQUE_CONTROLLED_DIRECT_FOC control scheme is selected
	FOCInput.Vq	This variable is updated if the 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 reinitialize motor control after a period of time. By default, there is no automated return from a trap state.

Table 56 pmsm_foc_error_handling ()

Input parameters	None	–
Return	None	–
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 57 pmsm_foc_Clear_trap ()

Input parameters	None	–
Return	None	–
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 58 **pmsm_foc_motor_hold ()**

Input parameters	None	–
Return	None	–
Updated variables	None	–

9.2.4 Controller

pmsm_foc_speed_controller ()

This function is called if `SPEED_CONTROLLED_DIRECT_FOC` or `SPEED_CONTROLLED_VF_MET_FOC` control scheme is 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 59 **pmsm_foc_speed_controller()**

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

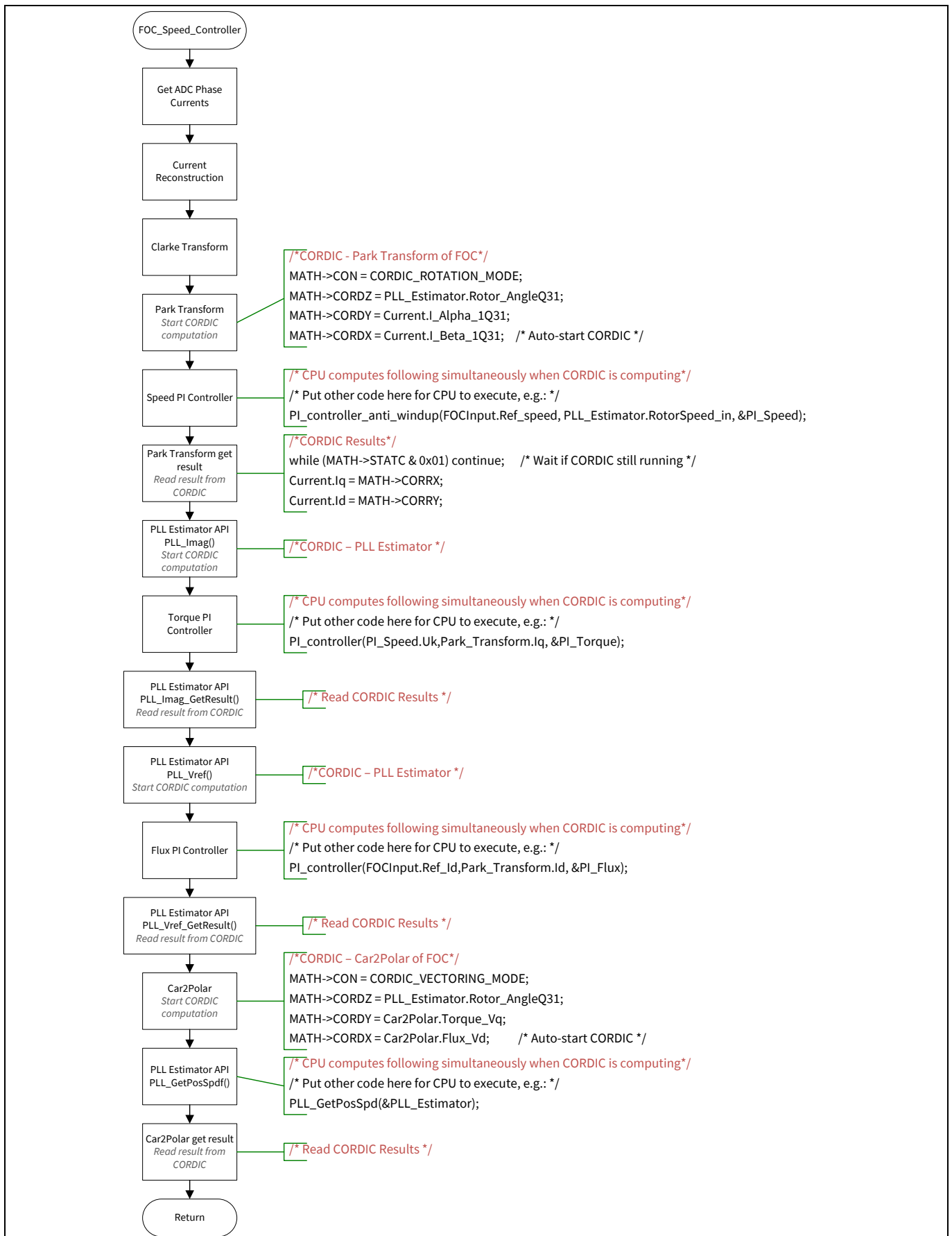


Figure 55 FOC speed control flowchart

pmsm_foc_torque_controller ()

This function 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 60 **pmsm_foc_torque_controller ()**

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

pmsm_foc_vq_controller (void)

This function 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 61 **pmsm_foc_vq_controller(void)**

Input parameters	<code>ADC.ADC_Bias_Iu</code>	ADC bias value for phase U current
	<code>ADC.ADC_Bias_Iv</code>	ADC bias value for phase V current
	<code>ADC.ADC_Bias_Iw</code>	ADC bias value for phase W current
	<code>FOCInput.Vq</code>	Reference Vq value
	<code>SVM.CurrentSectorNo</code>	SVM current sector number
Return	<code>Motor.Speed</code>	Current motor speed from PLL Estimator
Updated variables	<code>Current.I_u</code> , <code>Current.I_v</code> , <code>Current.I_w</code>	The current reconstruct function is called and <code>I_u</code> , <code>I_v</code> , and <code>I_w</code> are updated
	<code>Park_Transform.Iq</code> , <code>Park_Transform.Id</code>	Park Transform function is called and <code>Iq</code> and <code>Id</code> are updated
	<code>Clarke_Transform.I_Alpha</code> , <code>Clarke_Transform.I_Beta</code>	Clarke Transform function is called and <code>I_alpha</code> and <code>I_Beta</code> are updated

	Car2Polar.Vref, Car2Polar.Vref_AngleQ31	Car2Polar function is called and Vref and Vref_AngleQ31 are updated
	PI_Flux PI_PLL	PI controller functions are called and the PI outputs are updated
	PLL_Estimator	PLL Estimator APIs are called and the variables are updated
	FOCOuput.Speed_by_Estimator	Current motor speed from PLL Estimator
	FOCOuput.Rotor_PositionQ31	Current motor position from PLL Estimator

9.3 Secondaryloop ISR

pmsm_foc_misc_works_of_irq()

This function is called in the secondary loop with a default 1 kHz frequency. It reads the motor speed you have set and scales it up (see Chapter 2.10 for scaling).

Table 62 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. The value that is updated depends on the MY_FOC_CONTROL_SCHEME setting.
	Motor.Target_Torque	
	Motor.Target_Voltag	

pmsm_foc_secondaryloop_callback()

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

Table 63 pmsm_foc_misc_works_of_irq()

Input parameters	None	-
Return	None	-
Updated variables	User-defined	-

9.4 FPU library

All the mathematical functions that are required for field oriented control to work are implemented in the FPU library *fpu_math2*. In particular it also implements the sine, cosine, arctangent, magnitude, and Park Transform.

9.4.1 Library 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, see the [CMSIS support manual](#)). For arctangent, it's impossible to use only a lookup 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 the library, set to 5.02).

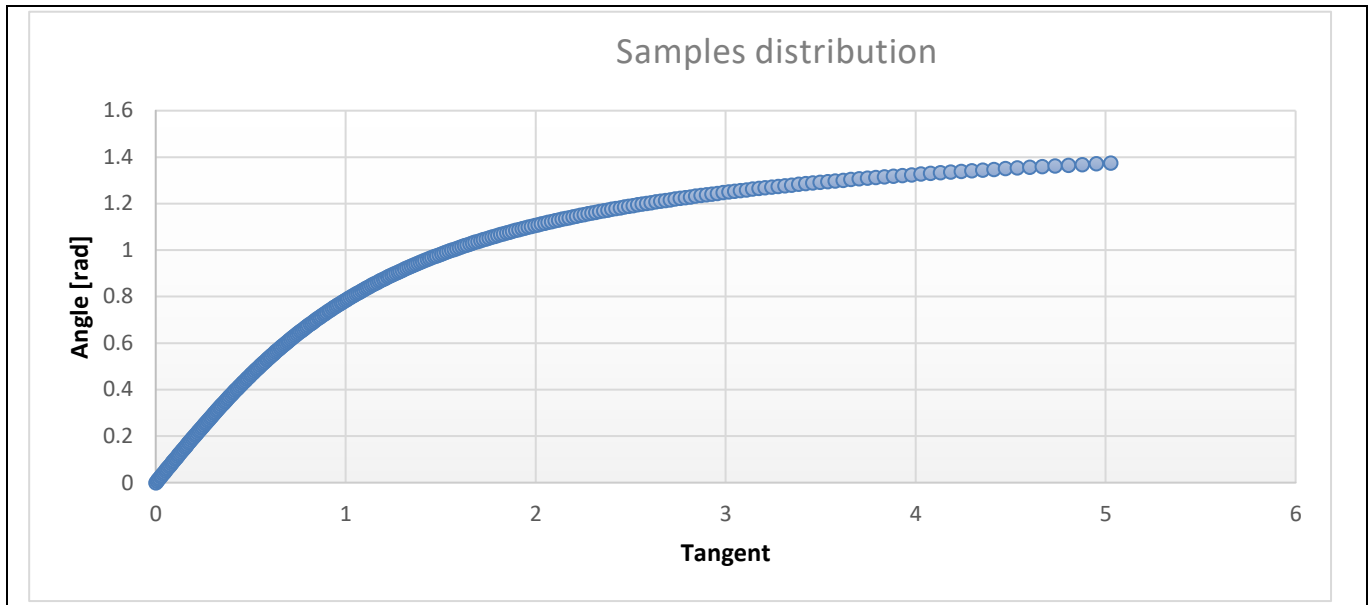


Figure 56 Sample distribution

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:

$$y = \frac{\left(\frac{x}{AT_X_SCALING + AT_X_OFFSET}\right)^2}{1 + \left(\frac{x}{AT_X_SCALING + AT_X_OFFSET}\right)^2} \cdot AT_Y_SCALING - AT_Y_OFFSET$$

Equation 29

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 the last LUT element, the library uses the following math approximation:

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

Equation 30

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

The precision of the function arctangent is about 2E-6 radians (0.00012 degrees, 20 bits in Q31 format). The library also supports an error correction mechanism that brings the precision to 5E-7 radians (0.00003 degrees, 22 bits in Q31). You must call the `fpu_tangent_lookup_table_generation()` function before using the `fpu_cart2polar()` function.

9.4.2 Library API

fpu_sin_q31 ()

This function calculates the sine of the given angle, both in Q31.

Table 64 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 65 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 the rotor angle.

Table 66 fpu_park_q31 ()

Input parameters	Clarke_Transform.I_Alpha, Clarke_Transform.I_Beta	The 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, the rotor angle.
Return	Park_Transform.Iq, Park_Transform.Id	The 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, and then adds the offset angle to the polar one. Before using, you must call the fpu_tangent_lookup_table_generation() function.

Table 67 fpu_cart2polar ()

Input parameters	Car2Polar.Flux_Vd Car2Polar.Torque_Vq	The Y and X components of any physical quantity. In this case, gtf flux and torque components
	Rotor Angle	An angle added to the polar one. In this case, the 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 68 **fpu_cart2polar ()**

Input parameters	None	–
Return	None	–
Updated variables	Arctangent lookup table	–

Value for actangent

```
#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
```

Resources

- Infineon XMC1000 Motor Control Application Kit document: [XMC1000 Motor Control Application Kit](#).
- XMC code examples can be found at Infineon's [GitHub](#) page.
- AP32370 - XMC1000/XMC4000 - PMSM FOC motor control software using XMC™.
- Infineon XMC1000/XMC4000 Motor Control Power Card document: [3-phase DC Motor Control Power Card](#).
- XMC4800 control board for MADKS with M5 connector: [EVAL-XMC4800PSOC6M5](#).

Revision history

Revision history

Document revision	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.
1.7	2024-10-16	Added XMC4800 support.

Trademarks of Infineon Technologies AG

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

Edition 2024-10-16

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2024 Infineon Technologies AG.

All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

AP32370

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.