User's Guide

MSPM0 FOC Motor Control Solution

Sal Ye; Vishnu, Ajith

ABSTRACT

This user's guide introduces the overview of MSPM0 FOC Motor Control Solution. The guide also provides step-by-step guidance to set up an MSPM0 LaunchPad with supported DRV hardware board in CCS IDE environment, the reference tuning flow to spin and tune motors, and migration guide to apply the the project in a custom board.

Table of Contents

1 Introduction ... 3
2 Motor Control Theory ... 5
2.1 BLDC Motor Fundamentals ... 5
2.2 Mathematical Model and FOC Structure ... 6
2.3 Sensorless Field Oriented Control ... 7
3 MSP FOC System ... 17
3.1 Design Source ... 17
3.2 FOC Feature Overview ... 18
3.3 FOC Benchmark ... 19
4 MSP FOC Hardware ... 20
4.1 PWM Pin Configurations ... 20
4.2 ADC Pin Configurations ... 21
4.3 Fault Pin Configurations ... 22
4.4 Hall GPIO Pin Configurations ... 23
4.5 GPIO Pin Configurations ... 23
4.6 SPI Pin Configurations ... 23
4.7 UART Pin Configurations ... 23
4.8 External Connections for Evaluation Boards ... 24
5 MSP FOC Software ... 25
5.1 Project Structure ... 25
5.2 Software Overview ... 27
5.3 Register Map (Sensorless FOC) ... 30
6 Quick Start Guide ... 53
6.1 CCS IDE ... 53
6.2 GUI ... 55
7 Motor Tuning Guide ... 57
7.1 Hardware Board Parameter ... 57
7.2 Motor Parameter ... 59
7.3 Control Loop Parameter ... 61
7.4 Hall Angle Table ... 62
7.5 Spin the Motor (LVBLDC) ... 65
7.6 Spin the Motor with Hall Sensor ... 68
7.7 Tune the Motor (LVBLDC) ... 71
7.8 Overwrite User Input Register Table ... 90
8 Hardware Migration Guide ... 91
8.1 Hardware Layer Overview ... 91
8.2 Gate Driver Module ... 91
8.3 MCU Peripheral Configuration ... 98
8.4 Verification for Customized Board ... 115
9 Frequently Asked Questions (FAQs) ... 118
9.1 MSPM0 Failed to Connect ... 118

9.2 Spin the Motor in Hardcode....118
9.3 Reduce 1x ADC Pin for Simultaneously Sampling....118
9.4 Tune Real-time Control Parameter....118
9.5 Track Real-time Variable....119
10 Summary....123
11 References....124
12 Revision History....124

Trademarks

Code Composer Studio™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

Brushless Direct Current (BLDC) motors have gained widespread adoption across diverse applications due to their versatility in handling variable loads, constant loads, and precision positioning requirements. These motors are extensively utilized in:

- Industrial control systems
- Automotive applications
- Aviation systems
- Automation equipment
- Healthcare devices
- Various other specialized applications

Field-oriented Control (FOC) is a most common method for high performance BLDC motor control. Figure 1-1 shows a block diagram for FOC implementation utilizing two current sensors demonstrates a comprehensive control architecture that includes:

- **Dual current sensor feedback**
- **Park transformation** (converting three-phase to rotating reference frame)
- **Clarke transformation** (converting three-phase to stationary reference frame)
- **Inverse Park transformation** (converting stationary reference back to three-phase frame)
- **Space Vector PWM Generation** (SVGEN) for optimized switching

This design employs an incremental development approach to showcase a complete FOC implementation, providing a systematic framework that integrates all essential control components for motor control performance and efficiency.

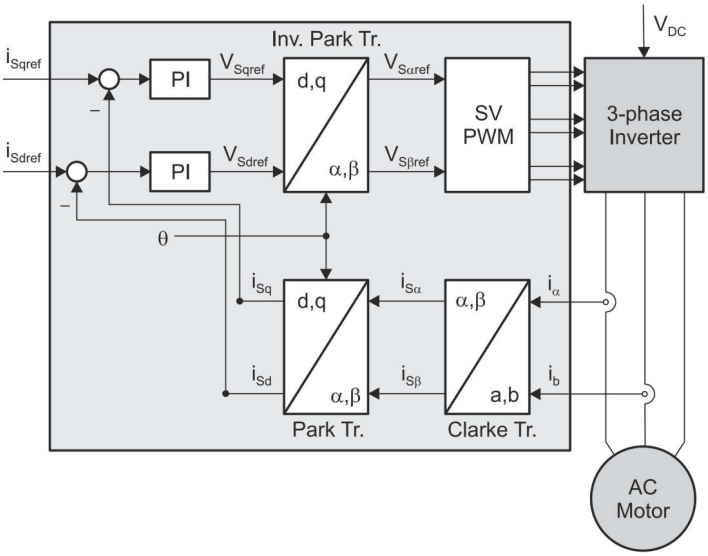


Figure 1-1. Sensorless Field-Oriented Control Using Two Current Sensors

Knowledge of rotor flux position serves as the fundamental cornerstone of the FOC algorithm, making accurate position feedback essential for effective motor control. Encoders offer the most straightforward approach to obtaining precise rotor position information, delivering high accuracy that directly translates to FOC performance. However, this accuracy comes at a higher system cost due to the additional hardware requirements.

Sensorless FOC presents a compelling alternative by eliminating the need for physical position sensors, resulting in significantly lower system costs. The absence of sensors also makes sensorless systems inherently more resistant to electromagnetic interference and capable of operating reliably in harsh environmental conditions where traditional sensors can fail. The trade-off for these advantages is increased algorithmic complexity, as sophisticated observer techniques must be implemented to accurately estimate rotor position and maintain high-performance motor control.

Hall sensor-based FOC represents a balanced compromise between cost-effectiveness and performance capabilities. This methodology delivers enhanced low-speed control robustness while experiencing some performance trade-offs at higher operating speeds when compared to sensorless implementations.

MSPM0 FOC Motor Control provides a middleware package to implement sensorless and hall-sensored FOC and to allow users to spin BLDC motors in 30 minutes or less using small-scale, simplified MSPM0 firmware examples with common motor driver designs.

The MSP FOC Motor Control Project requires:

- Code Composer Studio™ v12.8.0 or newer
- TI ARM CLANG Compilerv3.2.2 LTS or newer
- MSPM0 SDK v2.05.00.05 or newer

2 Motor Control Theory

2.1 BLDC Motor Fundamentals

A Brushless Direct Current (BLDC) motor comprises three essential elements: a wound stator, a permanent magnet rotor assembly, and rotor position sensing devices that can be integrated internally or mounted externally. These position sensors deliver real-time feedback that allows for precise adjustment of both the frequency and amplitude of the stator voltage reference, providing smooth torque generation and continuous rotor rotation.

The motor's architecture, featuring a permanent magnet rotor core surrounded by external stator windings, provides multiple performance advantages:

- **Low rotor inertia** for improved dynamic response
- **Great heat dissipation** through efficient thermal pathways
- **Compact design** enabling reduced overall motor dimensions

BLDC motors can exhibit either trapezoidal or sinusoidal Back Electromotive Force (BEMF) characteristics. This document specifically addresses motor control implementation for BLDC motors with sinusoidal BEMF waveforms (1). Follow the motor control principles below:

- Synchronous motor construction: Permanent magnets are rigidly fixed to the rotating axis to create a constant rotor flux. This rotor flux usually has a constant magnitude. The stator windings when energized create a rotating electromagnetic field. To control the rotating magnetic field, it is necessary to control the stator currents.
- The actual structure of the rotor varies depending on the power range and rated speed of the machine. Permanent magnets are suitable for synchronous machines ranging up-to a few Kilowatts. For higher power ratings the rotor usually consists of windings in which a DC current circulates. The mechanical structure of the rotor is designed for number of poles desired, and the desired flux gradients desired.
- The interaction between the stator and rotor fluxes produces a torque. Since the stator is firmly mounted to the frame, and the rotor is free to rotate, the rotor will rotate, producing a useful mechanical output as shown in Figure 2-1.
- The angle between the rotor magnetic field and stator field must be carefully controlled to produce maximum torque and achieve high electromechanical conversion efficiency. For this purpose a fine tuning is needed after closing the speed loop using sensorless algorithm to draw minimum amount of current under the same speed and torque conditions.
- The rotating stator field must rotate at the same frequency as the rotor permanent magnetic field; otherwise the rotor will experience rapidly alternating positive and negative torque. This will result in less than optimal torque production, and excessive mechanical vibration, noise, and mechanical stresses on the machine parts. In addition, if the rotor inertia prevents the rotor from being able to respond to these oscillations, the rotor will stop rotating at the synchronous frequency, and respond to the average torque as seen by the stationary rotor: Zero. This means that the machine experiences a phenomenon known as *pull-out*. This is also the reason why the synchronous machine is not self starting.
- The angle between the rotor field and the stator field must be equal to 90ºC to obtain the highest mutual torque production. This synchronization requires knowing the rotor position to generate the right stator field.
- The stator magnetic field can be made to have any direction and magnitude by combining the contribution of different stator phases to produce the resulting stator flux.

1. Includes library overview, software setup, hardware setup, and more. BLDC motors with a sinusoidal BEMF waveform are also commonly referred to as Permanent Magnet Synchronous (PMSM) motors.

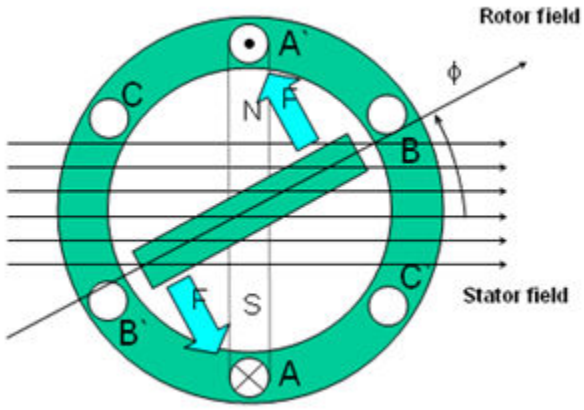


Figure 2-1. The Interaction Between the Rotating Stator Flux, and the Rotor Flux Produces a Torque

2.2 Mathematical Model and FOC Structure

The FOC structure for a PMSM is illustrated in Figure 2-2. In this system, the eSMO is used for achieving the sensorless control an IPMSM system, and the eSMO model is designed by utilizing the back EMF model together with a PLL model for estimating the rotor position and speed.

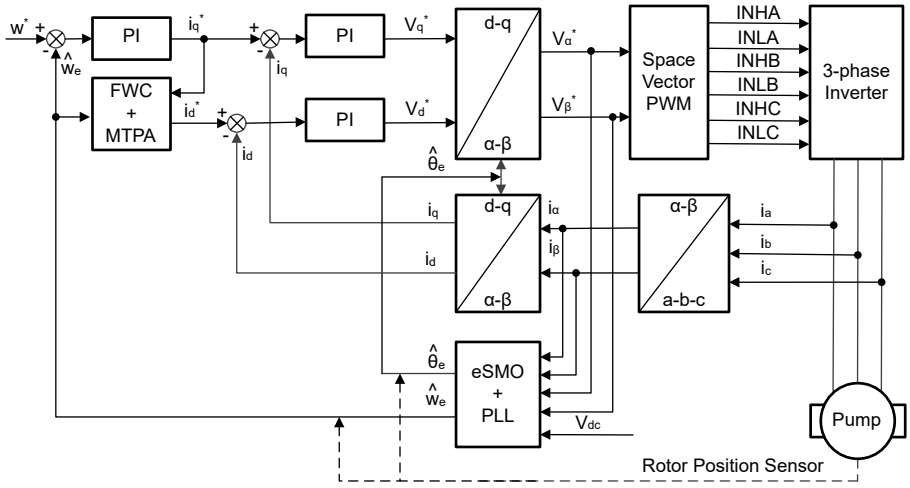


Figure 2-2. Sensorless FOC Structure of an PMSM System

An IPMSM consists of a three-phase stator winding (a, b, c axes), and permanent magnets (PM) rotor for excitation. The motor is controlled by a standard three-phase inverter. An IPMSM can be modeled by using phase a-b-c quantities. Through proper coordinate transformations, the dynamic PMSM models in the d-q rotor reference frame and the α-β stationary reference frame can be obtained. The relationship among these reference frames are illustrated in the following equation. The dynamic model of a generic PMSM can be written in the d-q rotor reference frame as:

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} R_s + pL_d & -\omega_e L_q \\ \omega_e L_d & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_e \lambda_{pm} \end{bmatrix} \quad (1)$$

Where v_d and v_q are the q-axis and d-axis stator terminal voltages, respectively; i_d and i_q are the d-axis and q-axis stator currents, respectively; L_d and L_q are the q-axis and d-axis inductances, respectively, *p* is the derivative operator, a short notation of $\frac{d}{dt}$; λ_{pm} is the flux linkage generated by the permanent magnets, R_s is the resistance of the stator windings; and ω_e is the electrical angular velocity of the rotor.

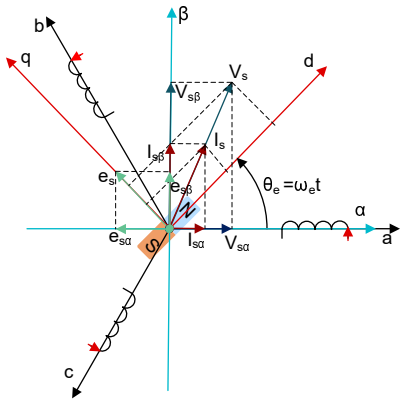


Figure 2-3. Definitions of Coordinate Reference Frames for PMSM Modeling

By using the inverse Park transformation as shown above, the dynamics of the PMSM can be modeled in the α-β stationary reference frame as:

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} R_s + pL_d & \omega_e\left(L_d - L_q\right) \\ -\omega_e\left(L_d - L_q\right) & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} \tag{2}$$

Where the e_a and $e_β$ are components of extended electromotive force (EEMF) in the α-β axis and can be defined as:

$$\begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} = \left(\lambda_{pm} + \left(L_d - L_q\right) i_d\right) \omega_e \begin{bmatrix} -\sin(\theta_e) \\ \cos(\theta_e) \end{bmatrix} \tag{3}$$

According to these equations, the rotor position information can be decoupled from the inductance matrix by means of the equivalent transformation and the introduction of the EEMF concept, so that the EEMF is the only term that contains the rotor pole position information. And then the EEMF phase information can be directly used to realize the rotor position observation. Rewrite the IPMSM voltage equation below as a state equation using the stator current as a state variable:

$$\begin{bmatrix} \dot{i}_\alpha \\ \dot{i}_\beta \end{bmatrix} = \frac{1}{L_d} \begin{bmatrix} -R_s & -\omega_e\left(L_d - L_q\right) \\ \omega_e\left(L_d - L_q\right) & -R_s \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{1}{L_d} \begin{bmatrix} V_\alpha - e_\alpha \\ V_\beta - e_\beta \end{bmatrix} \tag{4}$$

Since the stator current is the only physical quantity that can be directly measured, the sliding surface is selected on the stator current path:

$$S(x) = \begin{bmatrix} \hat{i}_\alpha - i_\alpha \\ \hat{i}_\beta - i_\beta \end{bmatrix} = \begin{bmatrix} \tilde{i}_\alpha \\ \tilde{i}_\beta \end{bmatrix} \tag{5}$$

where $\hat{i}_\alpha$ and $\hat{i}_\beta$ are the estimated currents, the superscript (**^**) indicates the estimated value, the superscript (**~**) indicates the variable error which refers to the difference between the observed value and the actual measurement value.

2.3 Sensorless Field Oriented Control

In applications similar to home appliances, mechanical sensor adds to cost, reliability and maintenance. In general, Sensorless based rotor position estimation methods are employed to efficiently drive the motor in applications where ultralow speed operation is not a requirement.

To detect the rotor position in sensorless methods, BEMF of the motor is estimated through various methods and there by the rotor speed and angle are approximated. The sliding mode observer (SMO) is commonly utilized

due to its various attractive features including reliability, desired performance, and robustness against system parameter variations.

The Finite Difference BEMF Estimation (FD-BEMF) method is also applied to observe the rotor position (Universal FOC only). The FD-BEMF is simple equation based BEMF estimation without sliding mode controller and filter for BEMF, this eliminates the Kslide tuning and filter tuing but BEMF is prone to noise and can create stability issues.

Estimated BEMF from both of the methods is used for rotor position tracking using a PLL method, are shown in Figure 2-4.

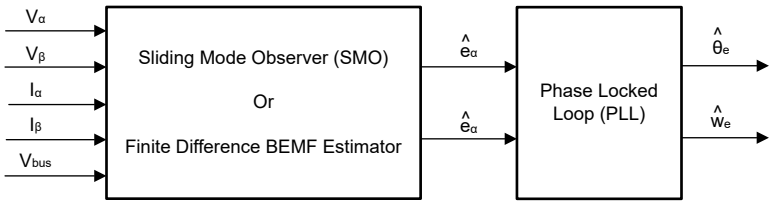


Figure 2-4. Rotor Angle Observer Block Diagram

2.3.1 FOC Fundamentals

To achieve better dynamic performance, a more complex control scheme needs to be applied, to control the PM motor. With the mathematical processing power offered by the microcontrollers, we can implement advanced control strategies, which use mathematical transformations to decouple the torque generation and the magnetization functions in PM motors. Such de-coupled torque and magnetization control is commonly called rotor flux oriented control, or simply Field Oriented Control (FOC).

In a direct current (DC) Motor, the excitation for the stator and rotor is independently controlled, the produced torque and the flux can be independently tuned as shown in Figure 2-5. The strength of the field excitation (for example, the magnitude of the field excitation current) sets the value of the flux. The current through the rotor windings determines how much torque is produced. The commutator on the rotor plays an interesting part in the torque production. The commutator is in contact with the brushes, and the mechanical construction is designed to switch into the circuit the windings that are mechanically aligned to produce the maximum torque. This arrangement then means that the torque production of the machine is fairly near optimal all the time. The key point here is that the windings are managed to keep the flux produced by the rotor windings orthogonal to the stator field.

To achieve better dynamic performance, a more complex control scheme needs to be applied, to control the PM motor. With the mathematical processing power offered by the microcontrollers, we can implement advanced control strategies, which use mathematical transformations to decouple the torque generation and the magnetization functions in PM motors. Such de-coupled torque and magnetization control is commonly called rotor flux oriented control, or simply Field Oriented Control (FOC).

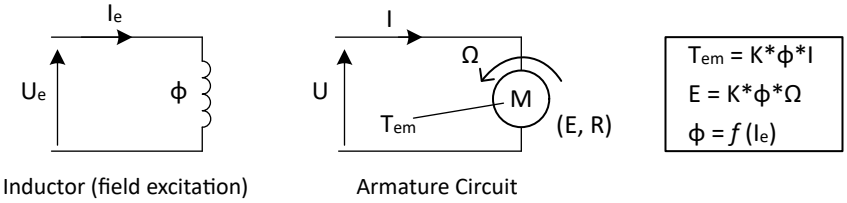


Figure 2-5. Flux and Torque are Independently Controlled in DC Motor Model

The goal of the FOC (also called vector control) on synchronous and asynchronous machine is to be able to separately control the torque producing and magnetizing flux components. FOC control will allow us to decouple the torque and the magnetizing flux components of stator current. With decoupled control of the magnetization, the torque producing component of the stator flux can now be thought of as independent torque control. To decouple the torque and flux, it is necessary to engage several mathematical transforms, and this is where the microcontrollers add the most value. The processing capability provided by the microcontrollers enables

these mathematical transformations to be carried out very quickly. This in turn implies that the entire algorithm controlling the motor can be executed at a fast rate, enabling higher dynamic performance. In addition to the decoupling, a dynamic model of the motor is now used for the computation of many quantities such as rotor flux angle and rotor speed. This means that their effect is accounted for, and the overall quality of control is better.

According to the electromagnetic laws, the torque produced in the synchronous machine is equal to vector cross product of the two existing magnetic fields as Equation 6.

$$\tau_{em} = \vec{B}_{stator} \times \vec{B}_{rotor} \tag{6}$$

This expression shows that the torque is maximum if stator and rotor magnetic fields are orthogonal meaning if we are to maintain the load at 90°. If we are able to ensure this condition all the time, if we are able to orient the flux correctly, we reduce the torque ripple and we ensure a better dynamic response. However, the constraint is to know the rotor position: this can be achieved with a position sensor such as incremental encoder. For low-cost application where the rotor is not accessible, different rotor position observer strategies are applied to get rid of position sensor.

In brief, the goal is to maintain the rotor and stator flux in quadrature: the goal is to align the stator flux with the q axis of the rotor flux, for example, orthogonal to the rotor flux. To do this, the stator current component in quadrature with the rotor flux is controlled to generate the commanded torque, and the direct component is set to zero. The direct component of the stator current can be used in some cases for field weakening, which has the effect of opposing the rotor flux, and reducing the back-emf, which allows for operation at higher speeds.

The Field Orientated Control consists of controlling the stator currents represented by a vector. This control is based on projections which transform a three phase time and speed dependent system into a two co-ordinate (d and q co-ordinates) time invariant system. These projections lead to a structure similar to that of a DC machine control. Field orientated controlled machines need two constants as input references: the torque component (aligned with the q co-ordinate) and the flux component (aligned with d co-ordinate). As Field Orientated Control is simply based on projections, the control structure handles instantaneous electrical quantities. This makes the control accurate in every working operation (steady state and transient) and independent of the limited bandwidth mathematical model. The FOC thus solves the classic scheme problems, in the following ways:

- The ease of reaching constant reference (torque component and flux component of the stator current)
- The ease of applying direct torque control because in the (d, q) reference frame the expression of the torque is defined in Equation 7.

$$\tau_{em} \propto \psi_R \times i_{sq} \tag{7}$$

By maintaining the amplitude of the rotor flux (ψ_R) at a fixed value we have a linear relationship between torque and torque component (i_{Sq}). We can then control the torque by controlling the torque component of stator current vector.

Space Vector Definition and Projection

The 3-phase voltages, currents and fluxes of AC-motors can be analyzed in terms of complex space vectors. With regard to the currents, the space vector can be defined as follows. Assuming that i_a, i_b, i_c are the instantaneous currents in the stator phases, then the complex stator current vector is defined in Equation 8.

$$\bar{i}_s = i_a + \alpha i_b + \alpha^2 i_c \tag{8}$$

where $\alpha = e^{j\frac{2}{3}\pi}$ and $\alpha^2 = e^{j\frac{4}{3}\pi}$ represent the spatial operators.

Figure 2-6 shows the stator current complex space vector.

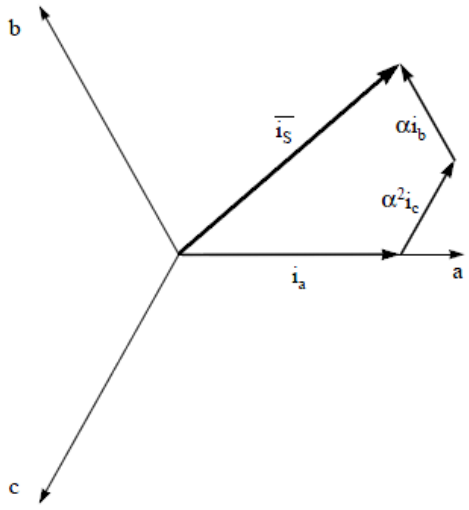


Figure 2-6. Stator Current Space Vector and its Component in (a,b,c) Frame

Where (a,b,c) are the three phase system axes. This current space vector depicts the three phase sinusoidal system. It still needs to be transformed into a two time invariant co-ordinate system. This transformation can be split into two steps:

- $(a,\ b) \Rightarrow (\alpha, \beta)$ (Clarke transformation) which outputs a 2-coordinate time-variant system
- $(\alpha, \beta) \Rightarrow (d,\ q)$ (Park transformation) which outputs a 2-coordinate time-invariant system

The $(a,\ b) \Rightarrow (\alpha, \beta)$ Clarke Transformation

The space vector can be reported in another reference frame with only two orthogonal axis called (α, β). Assuming that the axis a and the axis α*lpha* are in the same direction we have the following vector diagram as shown in Figure 2-7.

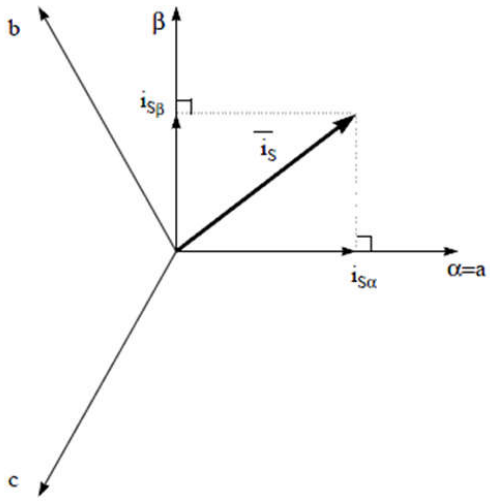


Figure 2-7. Stator Current Space Vector in the Stationary Reference Frame

The projection that modifies the 3-phase system into the (α, β) 2-dimension orthogonal system is presented in Equation 9.

$$i_{s\alpha} = i_a$$
$$i_{s\beta} = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b \quad (9)$$

The two phase (α, β) currents are still depends on time and speed.

The $(\alpha, \beta) \Rightarrow (d,\ q)$ Park Transformation

This is the most important transformation in the FOC. In fact, this projection modifies a 2-phase orthogonal system (α, β) in the (d, q) rotating reference frame. If we consider the d axis aligned with the rotor flux, Figure 2-8 shows the relationship for the current vector from the two reference frame.

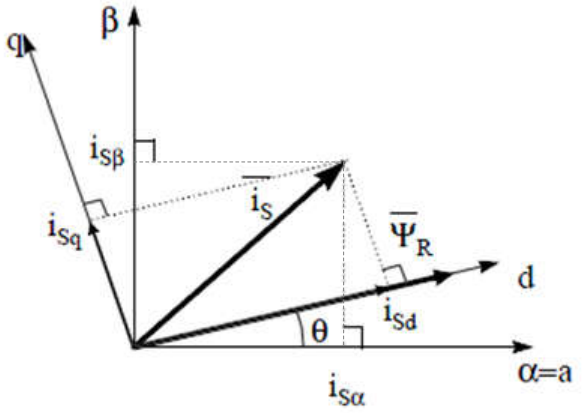


Figure 2-8. Stator Current Space Vector in The d,q Rotating Reference Frame

The flux and torque components of the current vector are determined by Equation 10.

$$\begin{aligned} i_{sd} &= i_{s\alpha}\cos(\theta) + i_{s\beta}\sin(\theta) \\ i_{sq} &= -i_{s\alpha}\sin(\theta) + i_{s\beta}\cos(\theta) \end{aligned} \tag{10}$$

where θ is the rotor flux position

These components depend on the current vector (α, β) components and on the rotor flux position; if we know the right rotor flux position then, by this projection, the d,q component becomes a constant. Two phase currents now turn into dc quantity (time-invariant). At this point the torque control becomes easier where constant i_{sd} (flux component) and i_{sq} (torque component) current components controlled independently.

The Basic Scheme of FOC for AC Motor

Figure 2-9 summarizes the basic scheme of torque control with FOC:

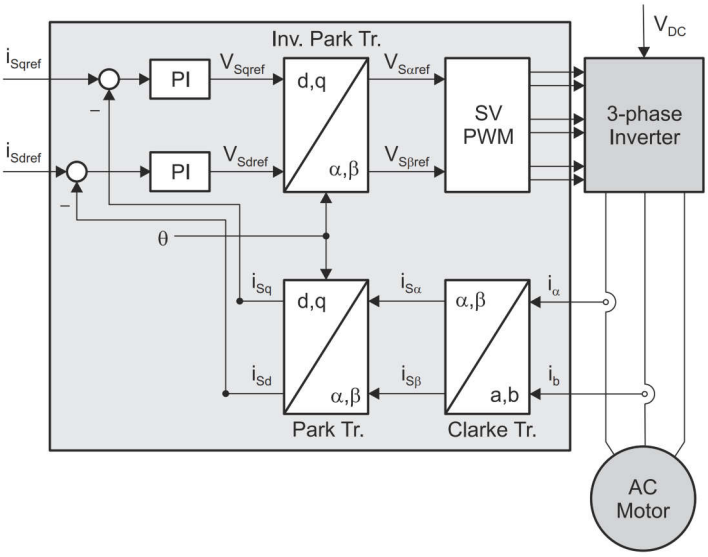


Figure 2-9. Basic Scheme of FOC for AC Motor

Two motor phase currents are measured. These measurements feed the Clarke transformation module. The outputs of this projection are designated $i_{s\alpha}$ and $i_{s\beta}$. These two components of the current are the inputs of the Park transformation that gives the current in the d,q rotating reference frame. The i_{sd} and i_{sq} components are compared to the references i_{sdref} (the flux reference component) and i_{sqref} (the torque reference component). At this point, this control structure shows an interesting advantage: it can be used to control either synchronous or induction machines by simply changing the flux reference and obtaining rotor flux position. As in synchronous permanent magnet a motor, the rotor flux is fixed determined by the magnets; there is no need to create one. Hence, when controlling a PMSM, i_{sdref} should be set to zero. As an AC induction motor needs a rotor flux creation to operate, the flux reference must not be zero. This conveniently solves one of the major drawbacks of the *classic* control structures: the portability from asynchronous to synchronous drives. The torque command i_{sqref} could be the output of the speed regulator when we use a speed FOC. The outputs of the current regulators are Vsdref and V_{sqref}; they are applied to the inverse Park transformation.

The outputs of this projection are $V_{s\alpha ref}$ and $V_{s\beta ref}$ which are the components of the stator vector voltage in the (α, β) stationary orthogonal reference frame. These are the inputs of the Space Vector PWM. The outputs of this block are the signals that drive the inverter. Note that both Park and inverse Park transformations need the rotor flux position. Obtaining this rotor flux position depends on the AC machine type (synchronous or asynchronous machine).

Rotor Flux Position

Knowledge of the rotor flux position is the core of the FOC. In fact if there is an error in this variable the rotor flux is not aligned with d-axis and i_{sd} and i_{sq} are incorrect flux and torque components of the stator current. Figure 2-10 shows the (a, b, c), (α, β) and (*d, q*) reference frames, and the correct position of the rotor flux, the stator current and stator voltage space vector that rotates with d,q reference at synchronous speed.

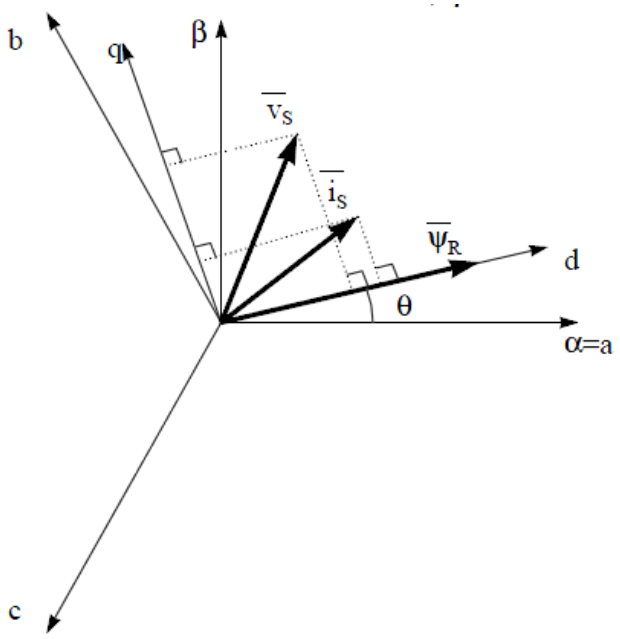


Figure 2-10. Current, Voltage and Rotor Flux Space Vectors in the (d, q) Rotating Reference Frame

The measure of the rotor flux position is different if we consider synchronous or asynchronous motor:

- In the synchronous machine the rotor speed is equal to the rotor flux speed. Then θ (rotor flux position) is directly measured by position sensor or by integration of rotor speed.
- In the asynchronous machine the rotor speed is not equal to the rotor flux speed (there is a slip speed), then it needs a particular method to calculate θ. The basic method is the use of the current model which needs two equations of the motor model in *d, q* reference frame.

Theoretically, the field oriented control for the PMSM drive allows the motor torque be controlled independently with the flux like DC motor operation. In other words, the torque and flux are decoupled from each other. The rotor position is required for variable transformation from stationary reference frame to synchronously rotating reference frame. As a result of this transformation (so called Park transformation), q-axis current will be controlling torque while d-axis current is forced to zero. Therefore, the key module of this system is the estimation of rotor position using enhance Sliding-Mode Observer (eSMO) or FAST estimator.

Figure 2-11 shows the overall block diagram of sensorless FOC of a PMSM motor using eSMO with field weakening control (FWC) and maximum torque per ampere (MTPA) in this document.

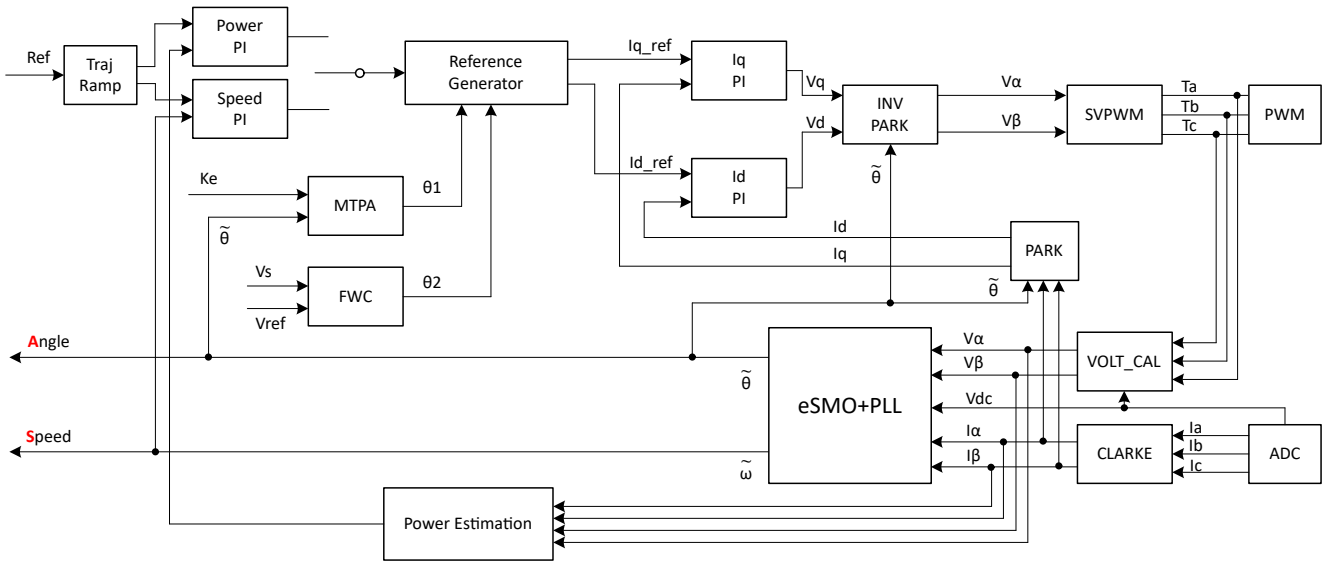


Figure 2-11. Sensorless FOC of PMSM Using eSMO With FWC and MTPA

2.3.2 Enhanced Sliding Mode Observer

Model-based BEMF estimation methods are used to achieve position sensorless control of the IPMSM drive system when the motor runs at middle or high speeds. The model methods estimates the rotor position by the back-EMF or the flux linkage model. The sliding mode observer is an observer-design method based on sliding mode control. The structure of the system is not fixed but purposefully changed according to the current state of the system, forcing the system to move according to the predetermined sliding mode trajectory. Advantages include fast response, strong robustness, and insensitivity to both parameter changes and disturbances.

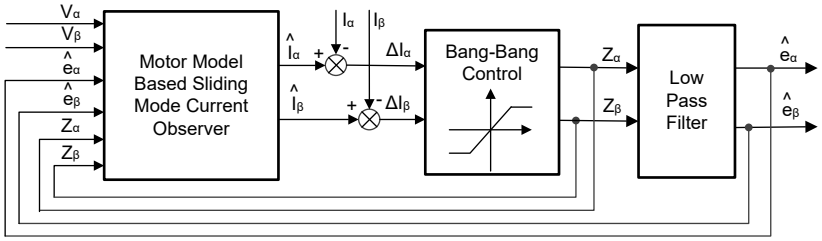


Figure 2-12. Block Diagram of Traditional Sliding Mode Observer

The traditional reduced-order sliding mode observer is constructed, which mathematical model is shown in Figure 2-4, the block diagram is shown in Figure 2-12.

$$\begin{bmatrix} \dot{\hat{i}}_\alpha \\ \dot{\hat{i}}_\beta \end{bmatrix} = \frac{1}{L_d}\begin{bmatrix} -R_s & -\hat{\omega}_e\left(L_d - L_q\right) \\ \hat{\omega}_e\left(L_d - L_q\right) & -R_s \end{bmatrix}\begin{bmatrix} \hat{i}_\alpha \\ \hat{i}_\beta \end{bmatrix} + \frac{1}{L_d}\begin{bmatrix} V_\alpha - \hat{e}_\alpha + z_\alpha \\ V_\beta - \hat{e}_\beta + z_\beta \end{bmatrix} \quad (11)$$

where z_α and z_β are sliding mode feedback components and are defined as:

$$\begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} = \begin{bmatrix} k_\alpha sign\left(\hat{i}_\alpha - i_\alpha\right) \\ k_\beta sign\left(\hat{i}_\beta - i_\beta\right) \end{bmatrix} \tag{12}$$

Where k_α and k_β are the constant sliding mode gain designed by Lyapunov stability analysis. If k_α and k_β are positive and significant enough to guarantee the stable operation of the SMO, the k_α and k_β should be large enough to hold $k_\alpha > \max(|e_\alpha|)$ and $k_\beta > \max\left(|e_\beta|\right)$.

The estimated value of EEMF in α-β axes ($\hat{e}_\alpha$, $\hat{e}_\beta$) can be obtained by low-pass filter from the discontinuous switching signals z_α and z_α :

$$\begin{bmatrix} \hat{e}_\alpha \\ \hat{e}_\beta \end{bmatrix} = \frac{\omega_c}{s + \omega_c} \begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} \tag{13}$$

Where $\omega_c = 2\pi f_c$ is the cutoff angular frequency of the LPF, which is usually selected according to the fundamental frequency of the stator current.

In a digital control application, a time discrete equation of the SMO is needed. The Euler method is the appropriate way to transform to a time discrete observer. The time discrete system matrix of Equation 11 in α-β coordinates is given by Equation 14 as:

$$\begin{bmatrix} \hat{i}_\alpha(n+1) \\ \hat{i}_\beta(n+1) \end{bmatrix} = \begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} \begin{bmatrix} \hat{i}_\alpha(n) \\ \hat{i}_\beta(n) \end{bmatrix} + \begin{bmatrix} G_\alpha \\ G_\beta \end{bmatrix} \begin{bmatrix} V_\alpha^*(n) - \hat{e}_\alpha(n) + z_\alpha(n) \\ V_\beta^*(n) - \hat{e}_\beta(n) + z_\beta(n) \end{bmatrix} \tag{14}$$

Where the matrix $[F]$ and $[G]$ are given by Equation 15 and Equation 16 as:

$$\begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} = \begin{bmatrix} e^{-\frac{R_s}{L_d}} \\ e^{-\frac{R_s}{L_q}} \end{bmatrix} \tag{15}$$

$$\begin{bmatrix} G_\alpha \\ G_\beta \end{bmatrix} = \frac{1}{R_s} \begin{bmatrix} 1 - e^{-\frac{R_s}{L_d}} \\ 1 - e^{-\frac{R_s}{L_q}} \end{bmatrix} \tag{16}$$

The time discrete form of Equation 13 is given by Equation 17 as:

$$\begin{bmatrix} \hat{e}_\alpha(n+1) \\ \hat{e}_\beta(n+1) \end{bmatrix} = \begin{bmatrix} \hat{e}_\alpha(n) \\ \hat{e}_\beta(n) \end{bmatrix} + 2\pi f_c \begin{bmatrix} z_\alpha(n) - \hat{e}_\alpha(n) \\ z_\beta(n) - \hat{e}_\beta(n) \end{bmatrix} \tag{17}$$

2.3.3 Finite Difference BEMF Estimator

Finite Difference BEMF Estimator could be applied in cases where Kslide (k_α and k_β) tuning is not desired or the noise disturbances in the system are minimal. In FD-BEMF method, the BEMF is derived based on stator reference frame voltage equations as shown in Equation 18.

$$\begin{bmatrix} \mathrm{e}\,\hat{\alpha} \\ \mathrm{e}\,\hat{\beta} \end{bmatrix} = \begin{bmatrix} \mathrm{v}\,\hat{\alpha} \\ \mathrm{v}\,\hat{\beta} \end{bmatrix} - \begin{bmatrix} \mathrm{r_s} & 0 \\ 0 & \mathrm{r_s} \end{bmatrix} \begin{bmatrix} \mathrm{i}\,\hat{\alpha} \\ \mathrm{i}\,\hat{\beta} \end{bmatrix} - \frac{\mathrm{d}}{\mathrm{dt}} \left(\begin{bmatrix} \mathrm{L_0} - \mathrm{L_1}\cos(2\theta) & -\mathrm{L_1}\sin(2\theta) \\ -\mathrm{L_1}\sin(2\theta) & \mathrm{L_0} + \mathrm{L_1}\cos(2\theta) \end{bmatrix} \begin{bmatrix} \mathrm{i}\,\hat{\alpha} \\ \mathrm{i}\,\hat{\beta} \end{bmatrix} \right) \tag{18}$$

where , $\mathrm{L_0} = \left(\frac{\mathrm{L_d} + \mathrm{L_q}}{2}\right)$, $\mathrm{L_1} = \left(\frac{\mathrm{L_d} - \mathrm{L_q}}{2}\right)$;

If the motor doesn't have any saliency, like in a surface mounted PMSM motor, the L_1 value is set to zero.

2.3.4 Rotor Position and Speed Estimation

Arc Tangent Estimator

The Arc Tangent Estimator used with SMO is illustrated in Figure 2-4.

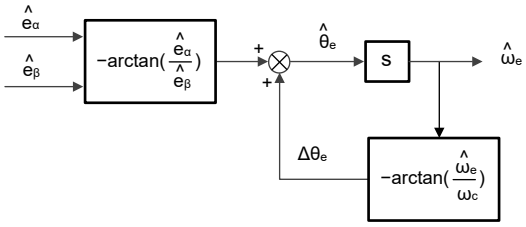


Figure 2-13. Block Diagram of Arc Tangent Estimation Position Tracker

In conventional SMO based rotor position estimators, the rotor flux angle is determined based on the arc tangent of estimated stationary co-ordinate BEMF values as in Equation 19:

$$\theta \hat{e} = -\tan^{-1}\left(\frac{e\hat{\alpha}}{e\hat{\beta}}\right) \quad (19)$$

A low pass filter removes the high-frequency term of the sliding mode function, which leads to occur phase delay resulting. It can be compensated by the relationship between the cut-off frequency ω_c and back EMF frequency ω_e , which is defined as:

$$\Delta\theta_e = -\tan^{-1}\left(\frac{\omega_e}{\omega_c}\right) \quad (20)$$

And then the estimated rotor position by using SMO method is:

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) + \Delta\theta_e \quad (21)$$

Phase Lock Loop (PLL) Tracker

With ARC Tangent Estimation, the accuracy of the position and velocity estimations are affected due to the existence of noise and harmonic components. To eliminate this issue, the PLL model can be used for velocity and position estimations in the sensorless control structure of the PMSM. The PLL structure used with SMO is illustrated in Figure 2-4.

The Back-EMF (BEMF) estimations $\hat{e}_\alpha$ and $\hat{e}_\beta$ can be used with a PLL model to estimate the motor angular velocity and position as shown in Figure 2-14.

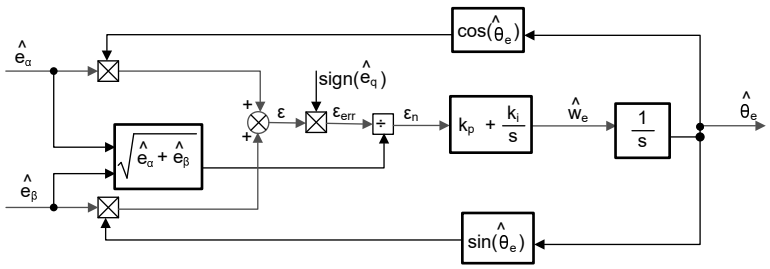


Figure 2-14. Block Diagram of Phase Locked Loop Position Tracker

Since $e_\alpha = E\cos(\theta_e)$, $e_\beta = E\sin(\theta_e)$ and $E = \omega_e\lambda_{pm}$, the position error can be defined as:

$$\varepsilon = \hat{e}_\beta \cos\left(\hat{\theta}_e\right) - \hat{e}_\alpha \sin\left(\hat{\theta}_e\right) = E\sin(\theta_e)\cos\left(\hat{\theta}_e\right) - E\cos(\theta_e)\sin\left(\hat{\theta}_e\right) = E\sin\left(\theta_e - \hat{\theta}_e\right) \quad (22)$$

Where E is the magnitude of the BEMF, which is proportional to the motor speed ω_e . When $\left(\theta_e - \hat{\theta}_e\right) < \frac{\pi}{2}$, the Equation 22 could be simplified as

$$\varepsilon = E\left(\theta_e - \hat{\theta}_e\right) \quad (23)$$

Further, the position error after the normalization of the BEMF can be obtained:

$$\varepsilon_n = \theta_e - \hat{\theta}_e \quad (24)$$

According to the analysis, the simplified block diagram of the quadrature phaselocked loop position tracker can be obtained as shown in Figure 2-15.

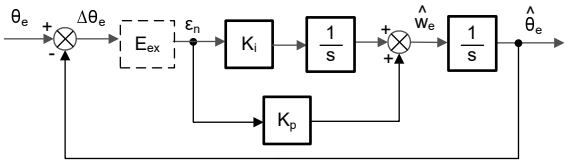


Figure 2-15. Simplified Block Diagram of Phase Locked Loop Position Tracker

The closed-loop transfer functions of the PLL can be expressed as Equation 25:

$$\frac{\hat{\theta}_e}{\theta_e} = \frac{k_p s + k_i}{s^2 + k_p s + k_i} = \frac{2\xi\omega_n s + \omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \quad (25)$$

where the k_p and k_i are the proportional and the integral gains of the standard PI regulator, its natural frequency ω_n and the damping ratio ξ is given:

$$k_p = 2\xi\omega_n, \qquad k_i = \omega_n^2 \quad (26)$$

3 MSP FOC System

The following figure shows the FOC motor control system overall block diagram. It mainly includes three parts: motor drive, motor control, and sensing part. The motor drive part consists of gate driver and MOSFET and is directly connected to the motor (BLDC/PMSM). The motor gets the power from the voltage bus by the motor drive part. The motor control part runs the control algorithm with specific requirements and generates the 6x PWM signals to gate driver. The sensing part is the bridge of the motor drive part and motor control part. The sensing part normally includes current sensing, voltage sensing, and might have rotor position sensing. The function of the sensing part is to get necessary electrical variable real-time value for control algorithm. This topology is designed for FOC applications that are low-cost and small form factors, such as pumps, fans, blowers, and small appliances.

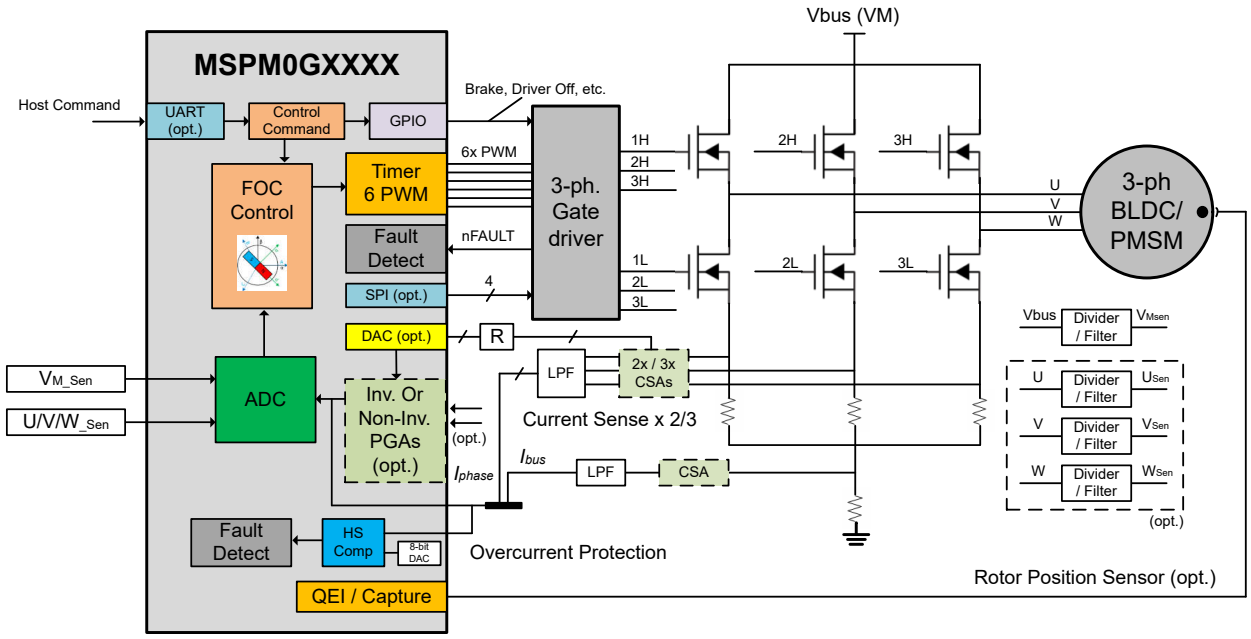


Figure 3-1. FOC Motor Control System

MSPM0GX50X series device provides a variety of high-performance analog peripherals such as two 12-bit 4-Msps ADCs (support simultaneously sampling mode) to accurately sense the motor phase voltage, bus voltage, phase current quickly, one 12-bit 1-Msps DAC which can customized the reference voltage for the signal process circuit, three high speed comparators with built-in reference DACs which can perform hardware overcurrent or over voltage protection, two zero-drift zero-crossover op-amps with programmable and one general-purpose amplifier to amplify the small input signal.

In lower voltage FOC applications, many gate driver or motor driver devices integrate up to three current sense amplifiers with programmable gain, which offloads analog requirements from the MSPM0GX50X device. MSPM0GX10X devices without analog integration are available in packages as small as VSSOP-20 and 24-VQFN to reduce system size and provide low cost. And MSPM0GX51X device provides higher memory and larger package options.

3.1 Design Source

The following table shows the overall FOC motor control design resource. Select available hardware solutions on TI.COM to run simple code examples with LaunchPad or onboard MSPM0 (TIDA reference design). Spin the motor with easy-to-use GUIs from the TI Gallery, or the CCSUDIO IDE tool.

Table 3-1. MSPM0 Motor Control Resource Summary

Algorithm Type	MSPM0G LaunchPad™ Kit	Motor Driver Hardware	GUI Link	SDK Project
Sensorless / Universal FOC	LP-MSPM0G3507	DRV8316REVM	MSPM0 Sensorless FOC GUI AND MSPM0 Universal FOC GUI	Example Project
	LP-MSPM0G3519			Example Project
	LP-MSPM0G3507	BOOSTXL-DRV8323RS		Example Project
	LP-MSPM0G3519			Example Project
	LP-MSPM0G3507	DRV8329AEVM		Example Project
	LP-MSPM0G3519			Example Project
	MSPM0G1507SPTR	TIDA-010250		Example Project
Sensored (Hall) FOC	LP-MSPM0G3507	DRV8316REVM	MSPM0 Sensored FOC GUI	Example Project
	LP-MSPM0G3519			Example Project
	MSPM0G1507SRHBR	TIDA-010251		Example Project

3.2 FOC Feature Overview

The following table shows the overall features supported by MSP SDK FOC Control Algorithm.

Table 3-2. FOC Control Support Feature

Support Feature		FOC Algorithm		
		Sensorless FOC	Universal FOC	Sensored FOC
Library Type	Fully open source for user's customization		√	√
	FOC algorithm is provided with static library	√		
Estimator	eSMO + PLL	√	√	
	FD-BEMF + PLL		√	
	Hall (supports calibration)			√
Startup	Align startup	√	√	
	Slow first cycle startup	√	√	√
	IPD startup	√	√	
	ISD startup	√	√	
Open loop	Speed & Current control	√	√	√
	Auto handoff	√		
	Auto torque current ramp down	√		
Closed loop	Speed closed loop	√	√	√
	Power closed loop	√	√	√
	Current closed loop	√	√	√
	PWM open loop	√	√	√
Stop	Coast (Hi-Z)	√	√	√
	Active spin down	√	√	√
	Brake	√	√	√
Fault Handling	Motor protection / Board protection	√	√	√
Deadtime Compensation	Compensation for PWM deadband time	√		√
OVM	Overmodulation on SVPWM generation	√	√	√
MTPA	Maximum torque per ampere control algorithm	√	√	√
FWC	Field weakening control algorithm	√	√	√

Table 3-2. FOC Control Support Feature (continued)

Support Feature		FOC Algorithm		
		Sensorless FOC	Universal FOC	Sensored FOC
Shunt Resistor Support	Single / Dual / Three shunt sampling mode	√	√	√

3.3 FOC Benchmark

The following table shows the benchmark for MSP SDK FOC Control Algorithm.

Table 3-3. FOC Control Benchmark

FOC Algorithm	Product	CPU / CLOCK	PWM Freq.	FOC Rate	FOC Time	CPU Bandwidth
Sensorless FOC	MSPM0G3507(1)	M0+ MathACL / 80MHz	20kHz	10kHz	60.8us	60.8%
	MSPM0G3107(2)	M0+ **Without** MathACL / 80MHz	20kHz	10kHz	79.9us	79.9%
	MSPM0C1106(2)	M0+ **Without** MathACL / 32MHz	15kHz	5kHz	199us	99.5%
Sensored FOC	MSPM0G3507(1)	M0+ MathACL / 80MHz	16kHz	16kHz	43.9us	70.3%
	MSPM0C1106(2)	M0+ **Without** MathACL / 32MHz	15kHz	5kHz	144us	72.0%

(1) The benchmark is tested with SDK FOC solution.
(2) The benchmark is estimated with SDK FOC solution.

4 MSP FOC Hardware

Table 4-1 shows the supported MSPM0 LaunchPad kits and EVMs for 3-phase sensorless FOC motor control. Refer to Section 8 when developing your custom hardware design.

Table 4-1. Supported Hardware for FOC using MSPM0

Motor Driver Hardware	Hardware User's Guide	Current Sense Amplifiers	SPI Driver Support	Motor Voltage Range (Rec.)	Motor Power (Rec.)
BOOSTXL-DRV8323RS	EVM User's Guide	3	Yes	6-60V	< 1000W
DRV8316REVM	EVM User's Guide	3	Yes	4.5-40V	< 75W
DRV8329AEVM	EVM User's Guide	1	No	4.5-60V	< 1000W
TIDA-010250	TIDA Design Guide	2 (in M0)	No	Max 265V AC	< 1000W
TIDA-010251	TIDA Design Guide	2 (in M0)	No	5-21V	< 600W

For the jumper configurations for the LaunchPad kit and EVM. Follow the guide in Table 4-2 to connect the chosen hardware evaluation board with MSPM0 LaunchPad.

Table 4-2. Hardware Connection Guide for LaunchPad and EVM Boards

FOC Algorithm	SDK User's Guide(1)
Sensorless FOC	Sensorless FOC SDK User's Guide
Universal FOC	Universal FOC SDK User's Guide
Sensored FOC	Sensored FOC SDK User's Guide

(1) Includes library overview, software setup, hardware setup, and more.

The following sections introduce the hardware design for LP-MSPM0G3507. The design of LP-MSPM0G3519 or others are the same with PINMUX little different due to BoosterPack PIN assignment difference.

4.1 PWM Pin Configurations

The default PWM output pin configurations for LP-MSPM0G3507 are shown in the following table. The required connections are six PWM output signals that send commutation patterns to gate drive for sensorless FOC motor control. Select TIMA instance of MSPM0 for FOC motor control, which includes the complimentary PWM outputs with deadband, fault handling with smaller than 40ns response time, and repeat counters for configuring FOC loop rates.

Table 4-3. Pin Configurations for PWM Outputs

MSPM0 Pin	Pin Function	DRV Pin Connection	DRV Function
PB4	TIMA0_C2, TIMA0 channel 2 output pin	INHA	Phase A high side PWM input
PB1	TIMA0_C2N, TIMA0 channel 2 complimentary output pin	INLA	Phase A low side PWM input
PA28	TIMA0_C3, TIMA0 channel 3 output pin	INHB	Phase B high side PWM input
PA31	TIMA0_C3N, TIMA0 channel 3 complimentary output pin	INLB	Phase B low side PWM input
PB20	TIMA0_C1, TIMA0 channel 1 output pin	INHC	Phase C high side PWM input
PB13	TIMA0_C1N, TIMA0 channel 1 complimentary output pin	INLC	Phase C low side PWM input
PA0	TIMA0_C0, TIMA0 channel 0, no output. It is used to trigger ADC to start current sampling by hardware		

TIMA0 is the preferred timer for motor control because it can directly provide three complimentary pairs of PWM outputs from the same timer counter (such as TIMA0_C1 and TIMA0_C1N). Additionally, any TIMA0 or TIMA1 output pair can be used with cross-trigger function to provide the synced three pairs PWM output. See Section 8.3.1 if required to modify the PWM output channel or pins.

4.2 ADC Pin Configurations

ADC0 and ADC1 are two simultaneous-sampling 4Msps analog-to-digital converters that are used to measure phase currents and voltages. ADC0 and ADC1 support measure phase currents simultaneously and bus voltage sequentially depending on the rotor angle under normal motor run conditions.

See Section 8.3.2 if required to modify the ADC channels.

4.2.1 DC Bus Voltage

The default pin configurations of LP-MSPM0G3507 for ADC voltages are shown in the following table.

Table 4-4. ADC Pin Configurations for DC Bus

MSPM0 Pin	MSPM0 Function	DRV8323 Pin Connection	DRV8323 Function
A0_2	ADC 0, Channel 2 Input	VSENVM	DC bus voltage output

The sensed voltages are realized using a resistor divider with an optional bypass filtering cap as shown in the following figure. Size the resistors so any motor voltage transients do not exceed the maximum voltage of the ADC inputs.

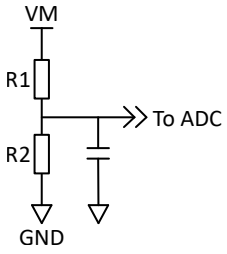


Figure 4-1. DC Bus Voltage Divider

4.2.2 Motor Phase Voltage

Phase voltage sensing is needed only if **initial speed detection** (ISD) feature is required. The default pin configurations of LP-MSPM0G3507 for ADC voltages are shown in the following table.

Table 4-5. ADC Pin Configurations for Motor Phase Voltage

MSPM0 Pin	MSPM0 Function	DRV8323 Pin Connection	DRV8323 Function
A1_6	ADC 1, channel 6 input	VSENA	Phase A sensed voltage output
A0_7	ADC 0, channel 7 input	VSENB	Phase B sensed voltage output
A1_5	ADC 1, channel 5 input	VSENC	Phase C sensed voltage output

4.2.3 Motor Phase Current

Depending on the current sampling method, the required connections are several (1~3) ADC inputs connected to the CSA outputs from the motor driver, external CSAs, or MCU built-in OPAs. An optional low-pass RC filter can be placed in series from the CSA outputs to the ADC inputs to filter out any high-frequency noise from the switching output signals for proper ADC sampling as shown in the following figure.

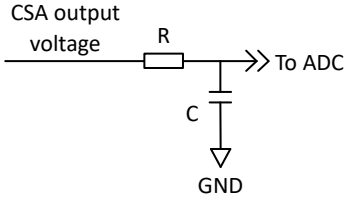


Figure 4-2. CSA Output Filter

Choose a filtering frequency that it at least 10 times the PWM switching frequency if used to detect motor current.

4.2.3.1 Single Shunt Current Sensing

The default ADC pin configurations of LP-MSPM0G3507 for single shunt current sensing in DRV8329 are shown in the following table. In single shunt current sensing, either of ADC0/ADC1 is used to sample the same shunt current (bus current) at two different instances in a single PWM cycle to estimate the three phase currents.

Table 4-6. ADC Pin Configurations for Single Shunt Sensing

MSPM0 Pin	MSPM0 Function	DRV8329 Pin Connection	DRV8329 Function
A1_2	ADC 1, Channel 2 Input	ISENA	DC bus current sense

4.2.3.2 Dual or Three Shunt Current Sensing

The default ADC pin configurations of LP-MSPM0G3507 for three phase current sensing are shown in the following table. The required connections are three ADC inputs connected to the three CSA outputs from the motor driver or external CSAs.

Table 4-7. ADC Pin Configurations for Three Shunt Current Sensing

MSPM0 Pin	MSPM0 Function	DRV8323 Pin Connection	DRV8323 Function
A1_2	ADC 1, Channel 2 Input	ISENA	Phase A current sense output
A0_3	ADC 0, Channel 3 Input	ISENB	Phase B current sense output
A1_3	ADC 1, Channel 3 Input	ISENC	Phase C current sense output

To mitigate the ADC conversion delay impact on FOC algorithm, recommend using two different ADC instances for three phase current sensing.

4.2.3.3 Three Shunt Current Sensing with Simultaneous Sampling

To get motor phase current real-time value preciously, user can enable simultaneous sampling mode, which requires connect phase B current input to two different ADC instances both, and connect phase A and phase C to different ADC instances separately. The following table shows the default ADC pin configurations of LP-MSPM0G3507 for three shunt motor current simultaneous sampling.

Table 4-8. ADC Pin Configurations with Simultaneous Sampling

MSPM0 Pin	MSPM0 Function	DRV8316 Pin Connection	DRV8316 Function
A1_13	ADC 1, Channel 13 Input	ISENA	Phase A current sense output
A1_14	ADC 1, Channel 14 Input	ISENB	Phase B current sense output
A0_12	ADC 0, Channel 12 input	ISENB	Phase B current sense output
A0_2	ADC 0, Channel 2 Input	ISENC	Phase C current sense output

In the run time, the algorithm selects the two phases with larger duty cycle to be sampled simultaneously and then calculates the last phase current value based on the sampling result.

4.3 Fault Pin Configurations

Typically, a motor driver drives an active-low open-drain fault pin (nFAULT) when there is a detected fault in the system. Examples are MOSFET overcurrent, gate drive, or power supply-related faults connections in the driver.

MSPM0 MCUs can detect the fault input with dedicated hardware paths to directly disable PWM output and provide low latency and response times compared to GPIO interrupt solution. See Section 4.3 if required to modify the fault pin.

The following table shows the default pin configurations for fault detection.

Table 4-9. Fault Pin Configurations

MSPM0 Pin	MSPM0 Function	DRV8323 Pin Connection	DRV8323 Function
TIMA_FAL2	TIMA fault handling input 2	nFAULT	Open-drain, active-low fault pin

4.4 Hall GPIO Pin Configurations

Sensored FOC requires Hall Sensor signal from BLDC / PMSM motor to operate the motor in closed loop speed control and drive the motor efficiently. Typically, the hall signals need external pull up to drive the HIGH output. The three Hall Sensor input signals are to be fed through general-purpose input/output (GPIO) inputs and Events are generated based on the changes in GPIO levels to a specific timer. A timer capture event is to be triggered based on any of the three GPIO input level changes. The 32-bit timer, TIMG12, is used to capture the timer capture event. See Section 8.3.4 if required to modify the Hall input pin.

Note

All the three GPIO pins must be connected to the same port and segment for Event triggering.

The following table summarizes the GPIO pin configurations with MSPM0 functionality on LP-MSPM0G3507.

Table 4-10. Hall GPIO Pin Configurations

MSPM0 Pin	MSPM0 Function	DRV8316 Pin Connection	DRV8316 Function
PA10	GPIO Port A Input PIN 10	HALLA	Hall Phase A signal output
PA11	GPIO Port A Input PIN 11	HALLB	Hall Phase B signal output
PA12	GPIO Port A Input PIN 12	HALLC	Hall Phase C signal output

4.5 GPIO Pin Configurations

The MSPM0 provides multiple GPIO output functions that can be used for motor driver-specific operations controlled by logic-level pins. Examples of motor driver functions include:

- Enable pin (ENABLE) / active-low sleep mode control (nSLEEP)
- Active high gate driver shutoff (DRVOFF)
- Active-high CSA Calibration (CAL)
- Active-high brake (BRAKE) / active-low brake (nBRAKE)
- Direction pin (DIR)

See the specific motor driver device's data sheet and the EVM user guide for GPIO configurable pins. Refer to Section 4.5 if required to modify the GPIO pin.

4.6 SPI Pin Configurations

The default pin configurations for SPI connections are shown in the following table. Some motor drivers include an optional SPI that is used for configuring control registers and reading status registers for fault diagnosis. Through SPI communication, users can:

- Configure gate drive source/sink current strength
- Configure CSA output behavior
- Run diagnostics
- Read fault bits when the fault pin has been detected as active low
- Clear fault status bits once the fault condition is removed

Table 4-11. SPI Pin Configurations

MSPM0 Pin	MSPM0 Function	DRV Pin Connection	DRV Function
SPIx_CSy	SPI chip select (y = 0,1,2,3)	nSCS	SPI chip select
SPIx_SCK	SPI bus clock	SCLK	SPI bus clock
SPIx_POCI	SPI peripheral out controller in	SDO	SPI data out
SPIx_PICO	SPI peripheral in controller out	SDI	SPI data in

4.7 UART Pin Configurations

UART can be used to receive commands to configure, spin, and control the motor. The commands are sent from a host MCU or GUI and can optionally be used for advanced protocols such as LIN communication.

Note

Use UART instance 3 (UART3_RX, UART3_TX) along with DMA for MSPM0GX50X and MSPM0GX10X. Details refer to DMA_ERR_01 in the Errata.

Table 4-12 shows the default pin configurations for UART connections to PC. Table 4-13 shows the MSPM0 PINMUX of LP-MSPM0G3507 for UART communication.

Table 4-12. UART Pin Configurations

MSPM0 Pin	MSPM0 Function	J101 Connection	XDS110 Function
UART3_TX	UART transmit	TXD>>	Backchannel UART receive
UART3_RX	UART receive	RXD<<	Backchannel UART transmit

Table 4-13. LP-MSPM0G3507 PINMUX for UART Communication

Gate Driver EVM Board	UART TX	UART RX
DRV8316REVM	PA26	PA13
BOOSTXL-DRV8323RS	PB12	
DRV8329AEVM	PB12	

4.8 External Connections for Evaluation Boards

Follow the steps below when connecting an MSPM0 LaunchPad to a DRV EVM board:

1. Connect the three motor phase terminals to the driver board (phases A, B, and C). If the motor has a center tap connection, leave these wires unconnected.
2. Connect the three Hall sensor inputs to the DRV Hall Interface header only if deploy Sensored FOC solution.
   a. If the DRV EVM board has no hall interface, users can directly connect the Hall sensor inputs to LaunchPad and enable pull-up resistor of MCU Hall input pins.
3. Make the inter-device connections from the MSPM0 LaunchPad kit to the DRV83XX EVM board by mating the EVM to the LaunchPad kit or using jumper wires. For hardware user guide connection details, see Sensorless FOC EVM Board Connection Guide and Sensored FOC EVM Board Connection Guide.
4. Connect a micro-USB cable from the MSPM0 LaunchPad kit to the PC.
5. Supply a voltage compliant with the Power Supply Voltage (VM) range. For recommended voltage range, see the board-specific user's guide or DRV-specific data sheet.

5 MSP FOC Software

The FOC software for MSPM0 MCUs is provided inside MSPM0-SDK and example projects are available for evaluation with Code Composer Studio™ (CCS) IDE.

The following table shows the software and documentation supported for FOC control in TI Resource Explorer.

Table 5-1. Software Support for FOC Control

FOC Algorithm	SDK Code Examples
Sensorless FOC	Sensorless FOC Examples
Universal FOC	Universal FOC Examples
Sensored FOC	Sensored FOC Examples

The following table shows the software version used by this document. For the higher FOC algorithm version, there might be few upgrade features, see the latest specific tuning guide for each FOC algorithm linked in Table 5-5.

Table 5-2. Used Software Version

FOC Algorithm	SDK Version	Algorithm Version
Sensorless FOC	2.08.00.03	2.04.01
Universal FOC	2.08.00.03	1.01.01
Sensored FOC	2.08.00.03	1.02.01

5.1 Project Structure

Figure 5-1 shows the CCS project structure for the Universal FOC project.

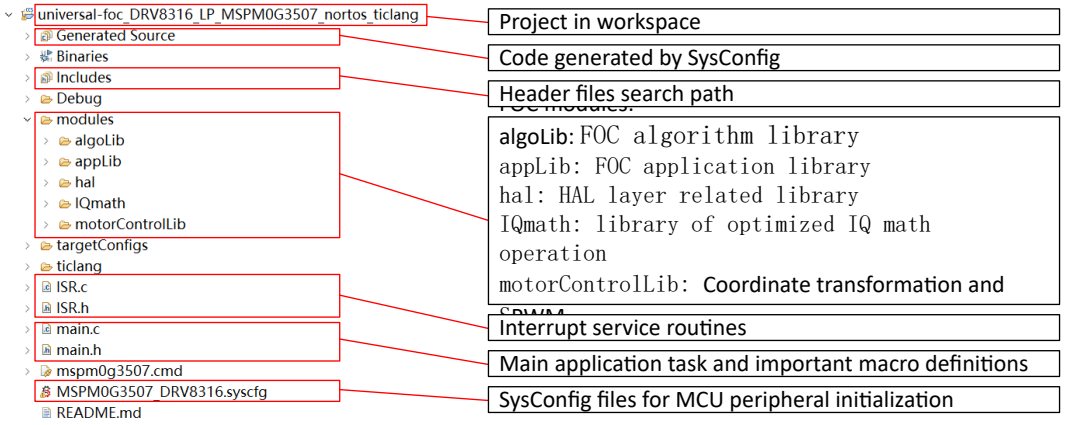


Figure 5-1. Project Structure Overview

The detailed project structure for each FOC module is shown below.

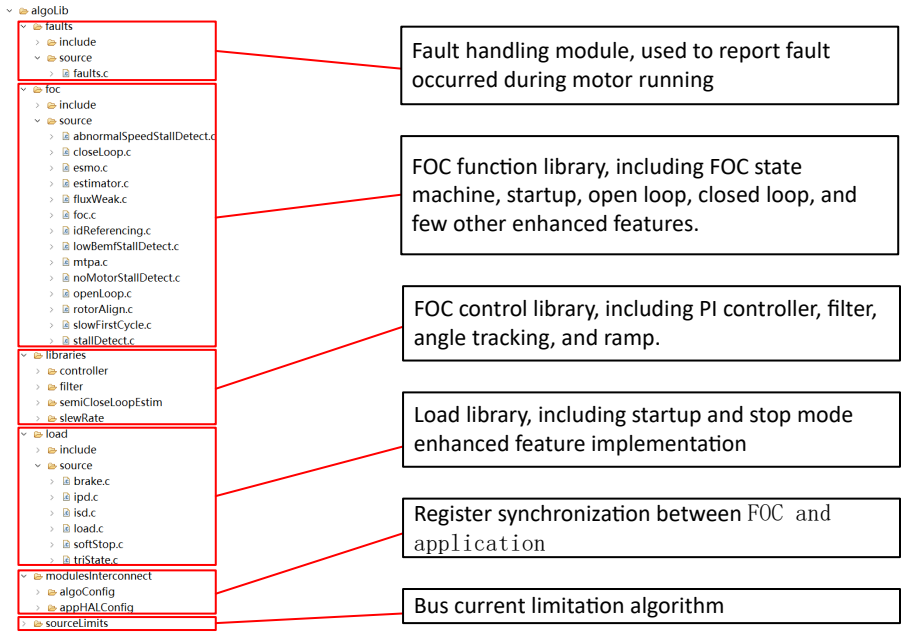


Figure 5-2. Project Structure Overview: algoLib

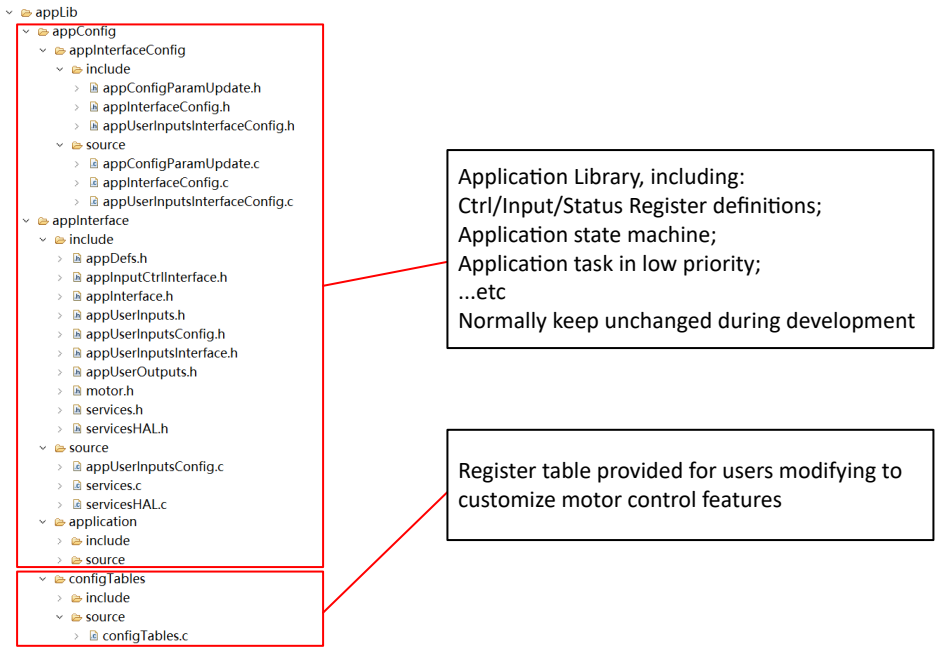


Figure 5-3. Project Structure Overview: appLib

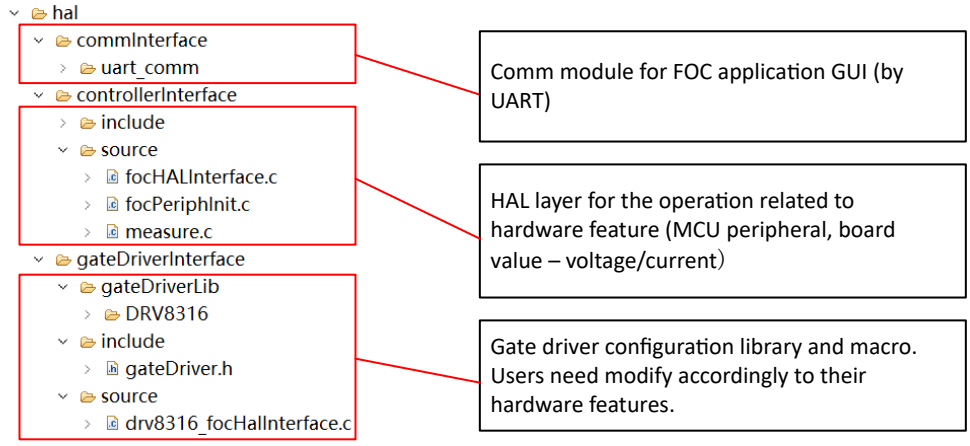


Figure 5-4. Project Structure Overview: hal

Sensorless FOC project shares a similar project structure while it has the static FOC library. Sensored FOC project shares a similar project structure with hall estimator instead.

5.2 Software Overview

Field-Oriented Control (FOC) Algorithm is made of three main layers: Application Layer, HAL Layer, and MSPM0 Driverlib Layer, shown in the following figure.

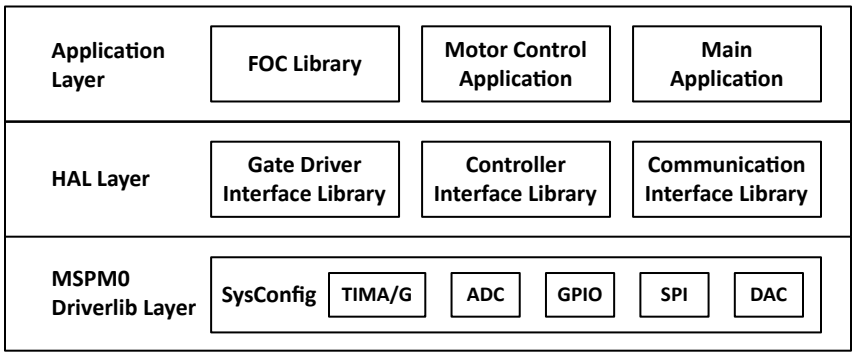


Figure 5-5. FOC Algorithm Architecture

5.2.1 Application Layer

Application Layer is a software layer that does specific functions like motor control. If any application performs any hardware specific actions, it is done through the HAL Layer APIs. The Application Layer contains three parts: FOC Library, Motor Control Application, and Main Application.

5.2.1.1 FOC Library

FOC library is a library that contains generic motor control algorithms for 3-phase FOC motor control. The supported FOC library features are summarized in Table 3-2.

Note

Sensorless FOC Library is a static library that does not support modification. Use Universal FOC Library instead if required to customize the FOC motor control source code or state machine.

5.2.1.2 Motor Control Application

Motor Control Application takes care of register value conversions from users' inputs and feeds to the FOC Library. The Application runs in the lower priority interrupt service routine (ISR) by a periodically 1ms timer, and main tasks are described below:

- Motor Control Application State Transitions
- Check for HV_IDE Faults (nFAULT input)
- Monitor the DC Bus Voltage
- MTPA & Field-weak function reference d-axis current setting
- Update the speed, power, torque input and run the input ramp
- Update User Status Register

5.2.1.3 Main Application

Main application contains the user code which initializes the application modules and periodically calls the configuration updates in while(1) loop. The following figure shows the main application code flow block diagram.

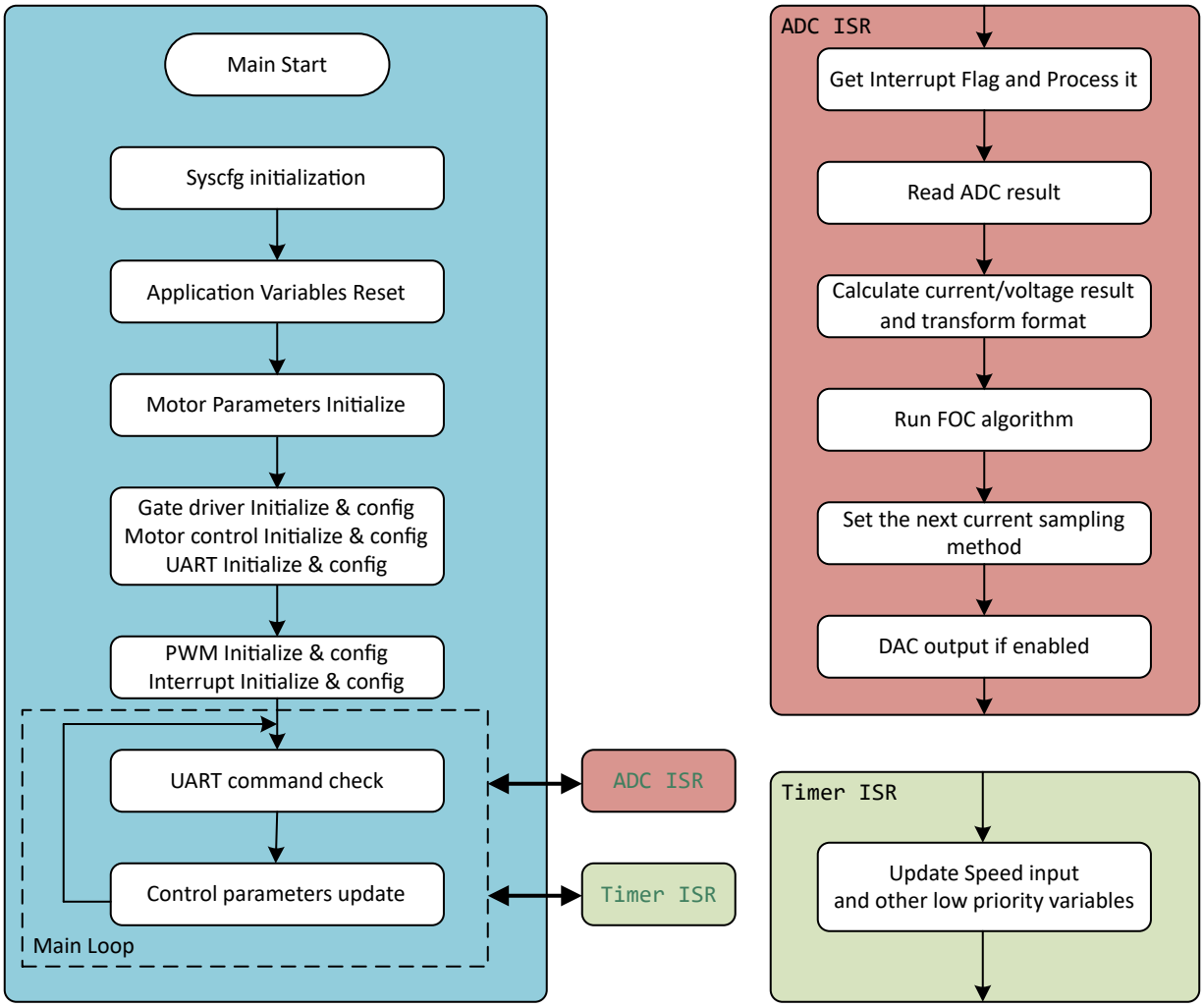


Figure 5-6. Main Application Flow

5.2.2 HAL Layer

The Hardware Abstraction Layer (HAL) creates an abstraction layer that provides APIs to configure different pins and peripherals. The goal of using HAL is to abstract all device specific configurations which simplify porting of the library to various hardware by minimizing the updates needed to other components. The HAL is meant

to abstract only the required pins or peripherals required for the application while still having flexibility and scalability for porting to other MSPM0 MCUs or motor drivers.

The following figure shows the hardware configuration relationship. The HAL uses the macro definitions generated by TI SysConfig. After users modify the peripheral instance or PINMUX in SysConfig tool, the HAL automatically follows the updated definition without the user having to change any code in the HAL layer.

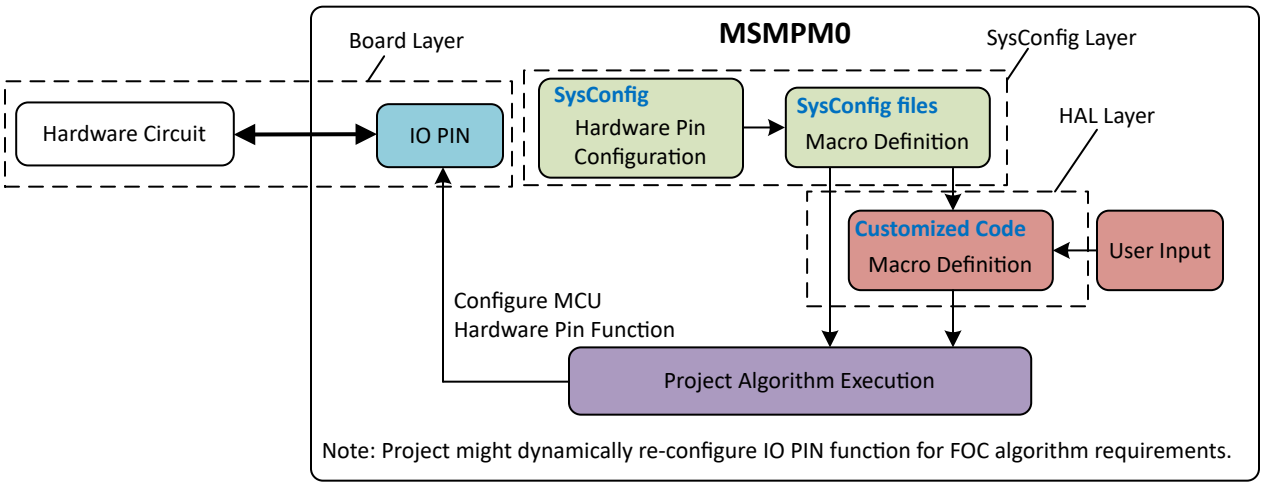


Figure 5-7. Hardware Configuration Relationship

5.2.2.1 Gate Driver Interface

Gate Driver Interface provides the APIs for reading the current and voltage channels. Additionally, if the DRV is used, the interface configures the CSA gain and other gate driver register configurations via SPI communication. Gate Driver Interface includes three parts:

- Gate Driver Library: has the open-source APIs to configure the gate driver registers through SPI if the gate driver is a SPI based
- Gate Driver HAL Interface: handles the configurations of ADC channels to convert phase voltage inputs and phase current inputs from gate driver
- Gate Driver Macro Definition: the ADC memory indexes used for reading the current and voltages are defined in the file (gateDriver.h)

5.2.2.2 Current Sensing Circuit

FOC algorithm supports single, dual and three shunt based current sensing methods for motor control. Depending on the gate driver EVM board hardware feature, the different sampling method is applied. The following table shows the details of different sensing methods.

Table 5-3. Current Sensing Method

Gate Driver EVM Board	Current Sensing Method	Invert / Non-invert Current Sensing Type
DRV8316REVM	Three Shunt	Invert Current Sensing
BOOSTXL-DRV8323RS	Three Shunt	Invert Current Sensing
DRV8329AEVM	Single Shunt	Non-invert Current Sensing
TIDA-010250	Two Shunt	Invert Current Sensing
TIDA-010251	Single Shunt	Non-invert Current Sensing

In three shunt based current sensing method, FOC algorithm additionally support simultaneous sampling method. The simultaneous sampling method dynamically modifies the two sampling channels in the given space sector, so that these two channels, with larger PWM duty cycle, can be sampled with different ADC instances (ADC0 and ADC1) at the same time.

In dual shunt based current sensing method, FOC algorithm calculates the third phase current based on the sampled two phase currents. It is a balance solution on the cost and current sensing accuracy. TI recommend

using two different ADC instances to sample two phase currents, so that the currents can be sampled at the same time.

In single shunt based current sensing method, the three phase currents are estimated based on sampled DC bus current from two instances. Unlike in dual or three phase current sensing, where the currents are sampled at center of PWM, the DC bus currents in 1-shunt method is sampled twice at various phase switching points within one PWM cycle. To realize this, the PWM are dynamically shifted in the FOC algorithm to enable sampling at different voltage vectors in the given space sector. Enough margin is created for current sampling in the non-overlapping regions.

5.2.2.3 Hardware Interface

In the application code, several GPIOs are used for a hardware interface to control the motor:

- Brake input: Pin input for applying brake
- Direction pin: Pin input for selecting direction
- NFAULT: Output pin, set low if any fault is detected
- nSleep: Output pin, sets the gate driver sleep input
- Hall sensor input (only apply for sensored FOC): Pin inputs for three phase hall signals

5.2.2.4 Communication Interface

Communication Interface helps to access the memory of the device using communication peripherals like UART, I2C etc. Users can read or write to the FOC application registers through the communication peripheral to control the motor externally.

FOC projects implement UART communication. Refer to UART Communication User Guide for more details on communication protocol.

5.2.3 MSPM0 Driverlib Layer

MSPM0 Driverlib is a set of fully functional APIs used to configure, control, and manipulate the hardware peripherals of the MSPM0 platform. Please refer to the SDK driverlib documentationfor more information.

5.3 Register Map (Sensorless FOC)

Register map contains set of three register structures for Setting the Motor Control Tuning Parameters, Monitoring the Motor Status variables and Setting the Real time Control parameters using User Input registers, User Status registers and User Control Registers, respectively, shown in Table 5-4.

Table 5-4. Register Overview

Register	Type	Function	Address
pUserCtrlRegs	USER_CTRL_INTERFACE_T	A set of user configurable parameters to control the motor in real time	0x20200400
pUserInputRegs	USER_INPUT_INTERFACE_T	A set of configurable registers to tune the motor performance in real time for various motor control features	0x20200000
pUserStatusRegs	USER_STATUS_INTERFACE_T	A set of consolidated variables available for users to read the motor status and analyze the control performance	0x20200430

Real-time control of the FOC registers can be carried out in two ways:

- Import the structures into the expression window of CCS during the code debug
- Read/Write the parameters over UART or GUI

The following sections describe registers and the variables associated with these structures for Sensorless FOC. FOC algorithm set the default value for register parameters to tune the motor. While the default values can be updated in source code by setting appropriate values as per application needs in the file (configTables.c).

For Sensored FOC and Universal FOC, they share a similar register structure but has difference on the supported features. See the specific tunning guide of each FOC algorithm for the detail and register difference, as shown in Table 5-5.

Table 5-5. FOC Tuning Guide

FOC Algorithm	FOC Tuning Guide
Sensorless FOC	MSPM0 Sensorless FOC Tuning Guide
Universal FOC	MSPM0 Universal FOC Tuning User's Guide
Sensored FOC	MSPM0 Sensored FOC Tuning Guide

Note

This document integrate register map of Sensorless FOC v2.04.01. See the register map update with a newer algorithm version in FOC algorithm's specific tuning guide.

5.3.1 User Control Registers (Base Address = 0x20200400h)

These sets of registers can be modified in the application code using pointer variable ***pUserCtrlRegs*** . Table 5-6 shows the structure of the User Control Registers of Sensorless FOC Algorithm.

Table 5-6. User Control Registers

Offset	Acronym	Register Name
0h	SPEED_CTRL	Speed Control Register
4h	ALGO_DEBUG_CTRL1	Algorithm Debug Control 1 register
8h	ALGO_DEBUG_CTRL2	Algorithm Debug Control 2 register
Ch	ALGO_DEBUG_CTRL3	Algorithm Debug Control 3 register
10h	DAC_CTRL	DAC Configuration and Control register

Complex bit access types are encoded to fit into small table cells as shown in Table 5-7 .

Table 5-7. Register Configuration Access Type Codes

Access Type	Code	Description
Read Type		
R	R	Read
Write Type		
W	W	Write
Reset or Default Value		
-n		Value after reset or the default value

5.3.1.1 Speed Control Register (Offset = 0h) [Reset = 00000000h]

Table 5-8 shows the register to control Motor Speed.

Table 5-8. SPEED_CTRL Register Field Descriptions

Bit	Field	Type	Reset	Description
31 - 15	RESERVED	R	0h	Reserved
14-0	SPEED_CTRL	W	000000000000000b	Target Motor Speed/Torque value % of speed or Torque command × 32768

5.3.1.2 Algo Debug Control 1 Register (Offset = 4h) [Reset = 00000000h]

Table 5-9 shows the register to control Algorithm debug functions.

Table 5-9. Algorithm Debug Control 1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31	CLEAR_FAULT	W	0b	Bit to clear set controller and Gate Driver Faults. Bit is automatically reset. 1h = Clear Fault Command.
30-0	Reserved	R	0000000 0000b	Reserved

5.3.1.3 Algo Debug Control 2 Register (Offset = 8h) [Reset = 00000000h]

Table 5-10 shows the register to control Algorithm Debug functions.

Table 5-10. Algorithm Debug Control 2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31	RESERVED	R	0h	Reserved
30	UPDATE_SYS_PARAMETERS	W	1h	Dynamically updates System parameters every 200mS like PI gains of Speed/Torque/Flux etc to tune for required performance 0b = Dynamic System updates Disabled 1b = Dynamic System updates Enabled
29	HALL_CALIB_ENABLE	W	0b	Use to enable Automatic Hall Calibration. This bit is reset automatically once calibration is completed. 0h = Hall Calibration Disabled/Completed 1h = Enable Hall Calibration
28	UPDATE_CONFIGS	R	0b	When the configurations are updated by the algorithm, this bit is reset. User can set this bit after giving the tuning command and wait for this bit to reset before starting the speed command.
27	STATUS_UPDATE_ENABLE	W	0b	This bit enables the continuous update of user Status variables in real time.
26	CURRENT_LOOP_DIS	W	0b	Use to control the FORCE_VD_CURRENT_LOOP_DIS and FORCE_VQ_CURRENT_LOOP_DIS. If CURRENT_LOOP_DIS = 1b, current loop and speed loop are disabled 0h = Enable Current Loop 1h = Disable Current Loop
25-16	FORCE_VD_CURRENT_LOOP_DIS	W-IQ(9)	0h	Sets Vd_ref in IQ(9) PU when current loop and speed loop are disabled If CURRENT_LOOP_DIS = 1b, then Vd is controlled using FORCE_VD_CURRENT_LOOP_DIS Vd_ref = (FORCE_VD_CURRENT_LOOP_DIS / 500) if FORCE_VD_CURRENT_LOOP_DIS < 500 - (FORCE_VD_CURRENT_LOOP_DIS - 512) / 500 if FORCE_VD_CURRENT_LOOP_DIS > 512 Valid values: 0 to 500 and 512 to 1000
15-6	FORCE_VQ_CURRENT_LOOP_DIS	W-IQ(9)	0h	Sets Vq_ref in IQ(9) PU when current loop speed loop are disabled If CURRENT_LOOP_DIS = 1b, then Vq is controlled using FORCE_VQ_CURRENT_LOOP_DIS Vq_ref = (FORCE_VQ_CURRENT_LOOP_DIS / 500) if FORCE_VQ_CURRENT_LOOP_DIS < 500 - (FORCE_VQ_CURRENT_LOOP_DIS - 512) / 500 if FORCE_VQ_CURRENT_LOOP_DIS > 512 Valid values: 0 to 500 and 512 to 1000
5-0	RESERVED	R	0h	Reserved

5.3.1.4 Algo Debug Control 3 Register (Offset = Ch) [Reset = 00000000h]

Table 5-11 shows the register to control Algorithm Debug 3 functions.

Table 5-11. Algorithm Debug Control 3 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-10	RESERVED	R	0h	

Table 5-11. Algorithm Debug Control 3 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
9-0	FLUX_MODE_REF	W-IQ(9)	0h	Sets Id_ref in IQ(9) PU when flux of the motor along D-axis is to be controlled Positive Id Control: (FLUX_MODE_REF/ 511) , if FLUX_MODE_REF < 512 Negative Id Control : (FLUX_MODE_REF - 512) / 511 if FLUX_MODE_REF > 512 Valid values are 0 to 511 and 512 to 1000-

5.3.1.5 DAC Configuration Register (Offset = 10h) [Reset = 00000000h]

DAC control registers defines configurations for monitoring the Real Time Algorithm and Hardware Register data on scope using the 12-bit DAC available on MSPM0G. For a detailed example on how to monitor an algorithm variable using DAC, see *Real Time Variable Tracking*.

Table 5-12. DAC Configuration Registers

Variables	Type	Reset	Description
DAC_EN	Unsigned Short (RW)	0h	0h = Disable DAC 1h = Enable DAC
DAC_SHIFT	short (RW)	0h	+ve value specifies the number of left bit shifts before loading the value to 12bit DAC register. -ve value specifies the number of right bit shifts before loading the value to 12bit DAC register. DAC Shift is used for monitoring unsigned integer values and Registers
DAC_SCALING_FACTOR	int(RW)	0x00000000h	Non zero scaling factor is used for numbers represented in IQ format to be monitored in DAC. To monitor the Global IQ(27) format variables DAC scaling factor of _IQ(1.0) is used. To represent other IQx format variables, set DAC scaling factor to IQx/IQGlobal.
DACOUT_ADDRESS	unsigned int(RW)	0x00000000h	Defines the address of 32 bit variable that is to be monitored through DAC.

5.3.2 User Input Registers (Base Address = 0x20200000h)

These sets of registers can be modified in the application code using pointer variable ***pUserInputRegs*** . Table 5-13 shows the structure of the User Input Registers of Sensorless FOC Algorithm.

Table 5-13. User Control Registers

Offset	Acronym	Register Name
0h	SYSTEM_PARAMETERS	System Parameters
38h	ISD_CFG	Initial Speed Detection Configuration
3Ch	RVS_DRV_CONFIG	Reverse Drive Configuration
40h	MOTOR_STARTUP1	Motor Startup 1 Configuration
44h	MOTOR_STARTUP2	Motor Startup 2 Configuration
48h	CLOSELOOP1	Close Loop1 Configuration
4Ch	CLOSELOOP2	Close Loop 2 Configuration
50h	FIELD_CTRL	Flux Control Configuration
54h	FAULT_CONFIG1	Fault Configuration 1
58h	FAULT_CONFIG2	Fault Configuration 2
5Ch	MISC_ALGO_CONFIG	Miscellaneous Algorithm Configuration
60h	PIN_CONFIGURATION	Pin Configuration
64h	PERI_CONFIG1	Peripheral Configuration 1

Complex bit access types are encoded to fit into small table cells as shown below.

Access Type	Code	Description
Read Type		
R	R	Read
Write Type		
W	W	Write
Reset or Default Value		

Access Type	Code	Description
-n		Value after reset or the default value

5.3.2.1 SYSTEM_PARAMETERS (Offset = 0h)

These tables show a set of basic system configuration parameters essential for motor control system functionality.

Table 5-14. Motor Resistance Configuration Registers (Offset = 0h)

Bit	Field	Type	Reset	Description
31-0	MTR_RESISTANCE	R/W	0000h	Motor Resistance in milliohms

Table 5-15. Motor Inductance Configuration (Offset = 4h)

Bit	Field	Type	Reset	Description
31-0	MTR_INDUCTANCE	R/W	0000h	Motor Inductance in microhenry. For Salient pole motors (Lq + Ld)/2

Table 5-16. Motor Saliency Configuration (Offset = 8h)

Bit	Field	Type	Reset	Description
31-0	MTR_SALIENCY	R/W	0.0(Float)	Saliency of Motor (Lq-Ld)/(Lq+Ld) in float.

Table 5-17. Motor BEMF Constant Configuration (Offset = Ch)

Bit	Field	Type	Reset	Description
31-0	MTR_BEMF_CONSTANT	R/W	0000h	Motor BEMF constant in mV/Hz × 10.

Table 5-18. Base Voltage Configuration (Offset = 10h)

Bit	Field	Type	Reset	Description
31-0	VOLTAGE_BASE	R/W	0.0(Float)	Base voltage of the board calculated based on the voltage divider as (3.3V × voltage divider ratio / √3) in volts. 3.3V is the full-scale value of the ADC.

Table 5-19. Base Current Configuration (Offset = 14h)

Bit	Field	Type	Reset	Description
31-0	CURRENT_BASE	R/W	0.0(Float)	Base current of the board calculated based on the CSA gain in as (1.65V / CSA Gain in volts/amp) in amps. 1.65V is the reference mid point voltage of the ADC for bidirectional current sensing. If the CSA gain is in V/V , multiply with current sense resistor value in ohms to compute CSA gain in volts/amp

Table 5-20. Motor Max Speed Configuration (Offset = 18h)

Bit	Field	Type	Reset	Description
31-0	MOTOR_MAX_SPEED	R/W	0.0(Float)	Rated motor speed in Hz from the data sheet

Table 5-21. Motor Max Power Configuration (Offset = 1Ch)

Bit	Field	Type	Reset	Description
31-0	MOTOR_MAX_POWER	R/W	0.0(Float)	Maximum Power rating of the Motor. **Note** FOC Algorithm computes the PU power for controlling the Input Power in closed loop. **Note** The PU Power is defined as MOTOR_MAX_POWER/ √3 * VOLTAGE_BASE * CURRENT_BASE

Table 5-22. Speed Loop Proportional Gain (Offset = 20h)

Bit	Field	Type	Reset	Description
31-0	SPEED_POWER_LOOP_KP	R/W	0.0(Float)	Proportional gain for the closed loop speed control / Closed Loop Power Control in float

Table 5-23. Speed Loop Integral Gain (Offset = 24h)

Bit	Field	Type	Reset	Description
31-0	SPEED_POWER_LOOP_KI	R/W	0.0(Float)	Integral gain for the closed loop speed control/ Closed Loop Power Control in float

Table 5-24. Torque Loop Proportional Gain (Offset = 28h)

Bit	Field	Type	Reset	Description
31-0	CURR_LOOP_KP	R/W	0.0(Float)	Proportional gain for the closed loop torque control in float

Table 5-25. Torque Loop Integral Gain (Offset = 2Ch)

Bit	Field	Type	Reset	Description
31-0	CURR_LOOP_KI	R/W	0.0(Float)	Integral gain for the closed loop torque control in float

Table 5-26. Flux weakening Controller Proportional Gain (Offset = 30h)

Bit	Field	Type	Reset	Description
31-0	FLUX_WEAK_KP	R/W	0.0(Float)	Proportional gain for the Flux weakening control in float

Table 5-27. Flux Weakening Controller Integral Gain (Offset = 34h)

Bit	Field	Type	Reset	Description
31-0	FLUX_WEAK_KI	R/W	0.0(Float)	Integral gain for the Flux weakening control in float

Table 5-28. Motor Pole Pairs (Offset = 38h)

Bit	Field	Type	Reset	Description
31-0	POLE_PAIRS	R/W	0(Int)	Total Pole pairs of Motor or Number of Poles / 2 .

5.3.2.2 MOTOR_STARTUP1 Register (Offset = 3Ch) [Reset = 00000000h]

Table 5-29 shows the register to configure motor startup settings1.

Table 5-29. MOTOR_STARTUP1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-30	RESERVED	R	00b	Reserved
29-26	CURR_RAMP_RATE	R/W	0h	Initial current ramp rate during Hall Calibration till the Maximum current is reached. 0h = 0.1A/s 1h = 1A/s 2h = 5A/s 3h = 10A/s 4h = 15A/s 5h = 25A/s 6h = 50A/s 7h = 100A/s 8h = 150A/s 9h = 200A/s Ah = 250A/s Bh = 500A/s Ch = 1000A/s Dh = 2000A/s Eh = 5000A/s Fh = No Limit A/s

Table 5-29. MOTOR_STARTUP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
25-22	CALIB_RUN_TIME	R/W	0h	Time spent while calibrating Hall for each CALIBRATION_ANGLE_STEP(defined in hallCalib.h) 0h = 10ms 1h = 50ms 2h = 100ms 3h = 200ms 4h = 300ms 5h = 400ms 6h = 500ms 7h = 750ms 8h = 1s 9h = 1.5s Ah = 2s Bh = 3s Ch = 4s Dh = 5s Eh = 7.5s Fh = 10s
21-17	CALIB_CURRENT_ILIMIT	R/W	00h	Current limit during Hall Calibration in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%

Table 5-29. MOTOR_STARTUP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
16-13	CALIB_ALIGN_TIME	R	000b	Time taken for initial rotor alignment to Zero Degrees before starting Hall Calibration. 0h = 10ms 1h = 50ms 2h = 100ms 3h = 200ms 4h = 300ms 5h = 400ms 6h = 500ms 7h = 750ms 8h = 1s 9h = 1.5s Ah = 2s Bh = 3s Dh = 5s Eh = 7.5s Fh = 10s
12-2	RESERVED	R	0h	Reserved
1	OL_ILIMIT_CONFIG	R/W	0b	Open loop current limit configuration 0h = Open loop current limit defined by OL_ILIMIT 1h = Open loop current limit defined by ILIMIT
0	RESERVED	R	0b	Reserved

5.3.2.3 MOTOR_STARTUP2 Register (Offset = 40h) [Reset = 00000000h]

Table 5-30 shows the register to configure motor startup settings2.

Table 5-30. MOTOR_STARTUP2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-27	OL_ILIMIT	R/W	0h	Open loop current limit in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%
26-23	OL_ACC_A1	R/W	0h	Open loop acceleration coefficient A1 0h = 0.01Hz/s 1h = 0.05Hz/s 2h = 1Hz/s 3h = 2.5Hz/s 4h = 5Hz/s 5h = 10Hz/s 6h = 25Hz/s 7h = 50Hz/s 8h = 75Hz/s 9h = 100Hz/s Ah = 250Hz/s Bh = 500Hz/s Ch = 750Hz/s Dh = 1000Hz/s Eh = 5000Hz/s Fh = 10000Hz/s

Table 5-30. MOTOR_STARTUP2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
22-19	OL_ACC_A2	R/W	0h	Open loop acceleration coefficient A2 0h = 0.0Hz/s2 1h = 0.05Hz/s2 2h = 1Hz/s2 3h = 2.5Hz/s2 4h = 5Hz/s2 5h = 10Hz/s2 6h = 25Hz/s2 7h = 50Hz/s2 8h = 75Hz/s2 9h = 100Hz/s2 Ah = 250Hz/s2 Bh = 500Hz/s2 Ch = 750Hz/s2 Dh = 1000Hz/s2 Eh = 5000Hz/s2 Fh = 10000Hz/s2
18	RESERVED	R/W	0h	Reserved
17-13	OL_MAX_SPEED	R/W	0h	Maximum operational Electrical Frequency during Forced Commutation for First electrical Cycle (% of MAX_SPEED) 0h = 1% 1h = 2% 2h = 3% 3h = 4% 4h = 5% 5h = 6% 6h = 7% 7h = 8% 8h = 9% 9h = 10% Ah = 11% Bh = 12% Ch = 13% Dh = 14% Eh = 15% Fh = 16% 10h = 17% 11h = 18% 12h = 19% 13h = 20% 14h = 22.5% 15h = 25% 16h = 27.5% 17h = 30% 18h = 32.5% 19h = 35% 1Ah = 37.5% 1Bh = 40% 1Ch = 42.5% 1Dh = 45% 1Eh = 47.5% 1Fh = 50%
12-8	Reserved	R	0h	Reserved

Table 5-30. MOTOR_STARTUP2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7-4	OL_FIRST_CYC_FREQ	R/W	0h	Frequency of first cycle in Open loop (% of MAX_SPEED) 0h = 1% 1h = 2% 2h = 3% 3h = 5% 4h = 7.5% 5h = 10% 6h = 12.5% 7h = 15% 8h = 17.5% 9h = 20% Ah = 25% Bh = 30% Ch = 35% Dh = 40% Eh = 45% Fh = 50%
3	FIRST_CYCLE_FREQ_SEL	R/W	0h	First cycle frequency in open loop 0h = Defined by OL_FIRST_CYC_FREQ 1h = 0Hz
2-0	RESERVED	R	0h	Reserved

5.3.2.4 CLOSED_LOOP1 Register (Offset = 44h) [Reset = 00000000h]

Table 5-31 shows the register to configure close loop settings1.

Table 5-31. CLOSED_LOOP1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-30	RESERVED	R/W	0h	Reserved
29-28	CONTROL_MODE	R/W	0h	FOC Closed loop Mode of operation 0h = Closed Loop Speed Control 1h = Closed Loop Power Control 2h = Closed Loop Torque Control 3h = Voltage Control mode
27	HIGH_FREQ_FOC_EN	R/W	0b	Enable /Disable High FOC Sampling rate. Higher the Sampling rate, lower the CPU bandwidth available for other tasks. 0h = High Frequency FOC Enable.(Max FOC Frequency 16Khz) 1h = High Frequency FOC Disable(Max FOC Frequency 8Khz)

Table 5-31. CLOSED_LOOP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
26-22	ILIMIT	R/W	0h	Current limit in Closed loop Torque Mode and Closed loop Speed control in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%
21-20	MTR_STOP	R/W	00b	Motor stop method 0h = Hi-z 1h = Active spin down 2h = Braking 3h = Reserved
19	OVERMODULATION_ENABLE	R/W	0b	Overmodulation enable 0h = Disable Over Modulation 1h = Enable Over Modulation

Table 5-31. CLOSED_LOOP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
18-14	CL_ACC	R/W	0h	Closed loop acceleration 0h = 0.5Hz/s 1h = 1Hz/s 2h = 2.5Hz/s 3h = 5Hz/s 4h = 7.5Hz/s 5h = 10Hz/s 6h = 20Hz/s 7h = 40Hz/s 8h = 60Hz/s 9h = 80Hz/s Ah = 100Hz/s Bh = 200Hz/s Ch = 300Hz/s Dh = 400Hz/s Eh = 500Hz/s Fh = 600Hz/s 10h = 700Hz/s 11h = 800Hz/s 12h = 900Hz/s 13h = 1000Hz/s 14h = 2000Hz/s 15h = 4000Hz/s 16h = 6000Hz/s 17h = 8000Hz/s 18h = 10000Hz/s 19h = 20000Hz/s 1Ah = 30000Hz/s 1Bh = 40000Hz/s 1Ch = 50000Hz/s 1Dh = 60000Hz/s 1Eh = 70000Hz/s 1Fh = No limit
13	CL_DEC_CONFIG	R/W	0h	Closed loop deceleration configuration 0h = Closed loop deceleration defined by CL_DEC 1h = Closed loop deceleration defined by CL_ACC

Table 5-31. CLOSED_LOOP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
12-8	CL_DEC	R/W	0h	Closed loop deceleration. This register is used only if AVS is disabled and CL_DEC_CONFIG is set to '0' 0h = 0.5Hz/s 1h = 1Hz/s 2h = 2.5Hz/s 3h = 5Hz/s 4h = 7.5Hz/s 5h = 10Hz/s 6h = 20Hz/s 7h = 40Hz/s 8h = 60Hz/s 9h = 80Hz/s Ah = 100Hz/s Bh = 200Hz/s Ch = 300Hz/s Dh = 400Hz/s Eh = 500Hz/s Fh = 600Hz/s 10h = 700Hz/s 11h = 800Hz/s 12h = 900Hz/s 13h = 1000Hz/s 14h = 2000Hz/s 15h = 4000Hz/s 16h = 6000Hz/s 17h = 8000Hz/s 18h = 10000Hz/s 19h = 20000Hz/s 1Ah = 30000Hz/s 1Bh = 40000Hz/s 1Ch = 50000Hz/s 1Dh = 60000Hz/s 1Eh = 70000Hz/s 1Fh = No limit
7-8	PWM_FREQ_OUT	R/W	0h	Output PWM switching frequency 0h = 5kHz 1h = 10kHz 2h = 16kHz 3h = 20kHz 4h = 25kHz 5h = 32kHz 6h = 40kHz 7h = 48kHz 8h = 50kHz 9h = 64kHz Ah = 80kHz Bh = N/A Ch = N/A Dh = N/A Eh = N/A Fh = N/A
14	PWM_MODE	R/W	0b	PWM modulation 0h = Continuous Space Vector Modulation 1h = Discontinuous Space Vector Modulation

Table 5-31. CLOSED_LOOP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
3	AVS_EN	R/W	0b	AVS enable 0h = Disable 1h = Enable
2	DEADTIME_COMP_EN	R/W	0b	Dead-time compensation enable 0h = Disable 1h = Enable
1	RESERVED	R/W	0b	Reserved

5.3.2.5 CLOSED_LOOP2 Register (Offset = 48h) [Reset = 00000000h]

Table 5-32 shows the register to configure close loop settings2.

Table 5-32. CLOSED_LOOP2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-28	ACT_SPIN_THR	R/W	0h	Speed threshold for active spin down (% of MAX_SPEED) 0h = 100% 1h = 90% 2h = 80% 3h = 70% 4h = 60% 5h = 50% 6h = 45% 7h = 40% 8h = 35% 9h = 30% Ah = 25% Bh = 20% Ch = 15% Dh = 10% Eh = 5% Fh = 2.5%
27-24.	BRAKE_SPEED_ THRESHOLD	R/W	0h	Speed threshold for BRAKE pin and motor stop options (Low Side Braking or align braking) (% of MAX_SPEED) 0h = 100% 1h = 90% 2h = 80% 3h = 70% 4h = 60% 5h = 50% 6h = 45% 7h = 40% 8h = 35% 9h = 30% Ah = 25% Bh = 20% Ch = 15% Dh = 10% Eh = 5% Fh = 2.5%

Table 5-32. CLOSED_LOOP2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
23-19	BRK_CURR_THR	R/W	0h	Brake current limit in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%
18-14	LEAD_ANGLE	R/W	0h	Lead Angle in degrees applied in Voltage Control Mode 0 - 15 = 1 * Bit Value 15 - 31 = 2 *(Bit Value -15) + 15
13 - 0	RESERVED	R	0h	Reserved

5.3.2.6 FIELD_CTRL Register (Offset = 4Ch) [Reset = 00000000h]

Register to configure Flux Control settings

Table 5-33. FIELD_CTRL register bit Descriptions

Bit	Field	Type	Reset	Description
31-7	Reserved	R	0h	Reserved
6	MTPA_EN	R/W	0b	Enable/Disable Maximum Torque Per Ampere Control (MTPA) 0h = Disable MTPA 1h = Enable MTPA
5-4	FLUX_WEAK_REF	R/W	00b	Modulation Index Reference to be tracked in Flux Weakening mode 0h = 70% 1h = 80% 2h = 90% 3h = 95%

Table 5-33. FIELD_CTRL register bit Descriptions (continued)

Bit	Field	Type	Reset	Description
3-1	FLUX_WEAK_CURR_RATIO	R/W	000b	Max value of Flux Weakening Current Reference as % of ILIMIT 0h = Only Circular Limit in Place 1h = 80% 2h = 70% 3h = 60% 4h = 50% 5h = 40% 6h = 30% 7h = 20%
0	FLUX_WEAK_EN	R/W	0b	Enable/Disable Flux Weakening Control (MTPA) 0h = Disable Flux Weakening 1h = Enable Flux Weakening

5.3.2.7 FAULT_CONFIG1 Register (Offset = 50h) [Reset = 00000000h]

Register to configure fault settings1

Table 5-34. FAULT_CONFIG1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-6	RESERVED	R/W	0h	Reserved
5-2	LCK_RETRY	R/W	0h	Lock detection retry time 0h = 100ms 1h = 500ms 2h = 1s 3h = 2s 4h = 3s 5h = 4s 6h = 5s 7h = 6s 8h = 7s 9h = 8s Ah = 9s Bh = 10s Ch = 11s Dh = 12s Eh = 13s Fh = 14s
1-0	MTR_LCK_MODE	R/W	00b	Motor Lock Mode 0h = Motor lock detection causes latched fault; nFAULT active; 1h = Fault automatically cleared after LCK_RETRY time 2h = Motor lock in report only mode 3h = Motor lock detection is disabled

5.3.2.8 FAULT_CONFIG2 Register (Offset = 54h) [Reset = 00000000h]

Table 5-35 shows the register to configure fault settings2.

Table 5-35. FAULT_CONFIG2 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-27	RESERVED	R/W	0h	Reserved
26	ABN_SPEED_LOCK_EN	R/W	0b	Lock 1 : Abnormal speed Lock 0h = Disable 1h = Enable
25	HALL_INVALID_LOCK_EN	R/W	0b	Lock 2 : Invalid Hall Input Lock 0h = Disable 1h = Enable

Table 5-35. FAULT_CONFIG2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
24	NO_MOTOR_LOCK_EN	R/W	0b	Lock 3 : No motor Lock enable 0h = Disable 1h = Enable
23-21	LOCK_ABN_SPEED	R/W	000b	Abnormal speed lock threshold (% of MAX_SPEED) 0h = 130% 1h = 140% 2h = 150% 3h = 160% 4h = 170% 5h = 180% 6h = 190% 7h = 200%
20-18	RESERVED	R	0b	Reserved
17-13	NO_MTR_THR	R/W	00000b	No Motor current limit in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%
12-8	RESERVED	R/W	0h	Reserved.

Table 5-35. FAULT_CONFIG2 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7-5	MIN_VM_MOTOR	R/W	000b	Minimum voltage for running motor in % of BASE_VOLTAGE 0h = No Limit 1h = 5% 2h = 10% 3h = 12% 4h = 15% 5h = 18% 6h = 20% 7h = 25%
4	MIN_VM_MODE	R/W	0b	Undervoltage fault mode 0h = Latch on Undervoltage 1h = Automatic clear if voltage in bounds
3-1	MAX_VM_MOTOR	R/W	000b	Maximum voltage for running motor in % of BASE_VOLTAGE 0h = 60% 1h = 65% 2h = 70% 3h = 75% 4h = 80% 5h = 85% 6h = 90% 7h = Max Voltage
0	MAX_VM_MODE	R/W	0b	Overvoltage fault mode 0h = Latch on Overvoltage 1h = Automatic clear if voltage in bounds

5.3.2.9 MISC_ALGO Register (Offset = 58h) [Reset = 00000000h]

Table 5-36 shows the register to multiple miscellaneous Algorithm Configuration.

Table 5-36. MISC_ALGO Register Field Descriptions

Bit	Field	Type	Reset	Description
31-10	RESERVED	R/W	0h	Reserved
9-6	CL_SLOW_ACC	R/W	0h	Close loop acceleration when estimator is not yet fully aligned 0h = 0.1Hz/s 1h = 1Hz/s 2h = 2Hz/s 3h = 3Hz/s 4h = 5Hz/s 5h = 10Hz/s 6h = 20Hz/s 7h = 30Hz/s 8h = 40Hz/s 9h = 50Hz/s Ah = 100Hz/s Bh = 200Hz/s Ch = 500Hz/s Dh = 750Hz/s Eh = 1000Hz/s Fh = 2000Hz/s
5-2	RESERVED	R	0b	Reserved

Table 5-36. MISC_ALGO Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
1-0	BRAKE_CURRENT_PERSIST	R/W	00b	Persistence time for current below threshold during brake 0h = 50ms 1h = 100ms 2h = 250ms 3h = 500ms

5.3.2.10 PIN_CONFIG Register (Offset = 5Ch) [Reset = 00000000h]

Table 5-37 shows the register to configure hardware pins.

Table 5-37. PIN_CONFIG Register Field Descriptions

Bit	Field	Type	Reset	Description
31-20	RESERVED	R/W	0h	Reserved
19	VDC_FILT_DIS	R/W	0b	Vdc Filter Disable 0h = Enabled 1h = Disabled
18-3	RESERVED	R/W	0h	Reserved
2	BRAKE_PIN_MODE	R/W	0b	Brake pin mode 0h = Low side Brake 1h = Align Brake
1-0	BRAKE_INPUT	R/W	00b	Brake pin override 0h = Hardware Pin BRAKE 1h = Override pin and brake / align according to BRAKE_PIN_MODE 2h = Override pin and do not brake / align 3h = Hardware Pin BRAKE

5.3.2.11 PERI_CONFIG Register (Offset = 60h) [Reset = 00000000h]

Table 5-38 show the register to peripheral.

Table 5-38. PERI_CONFIG1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-15	RESERVED	R	0h	Reserved
14-9	MCU_DEAD_TIME	R/W	0h	Dead time applied between the High Side and Low side switches = 50ns × MCU_DEAD_TIME

Table 5-38. PERI_CONFIG1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
8-4	BUS_CURRENT_LIMIT	R/W	00000b	Bus Current Limit in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%
3	BUS_CURRENT_LIMIT_ENABLE	R/W	0b	Bus current limit enable 0h = Disable 1h = Enable
2-1	DIR_INPUT	R/W	00b	DIR pin override 0h = Hardware Pin DIR 1h = Override DIR pin with clockwise rotation OUTA-OUTB-OUTC 2h = Override DIR pin with counter clockwise rotation OUTA-OUTC-OUTB 3h = Hardware Pin DIR
0	DIR_CHANGE_MODE	R/W	0b	Response to change of DIR pin status 0h = Follow motor stop options and ISD routine on detecting DIR change 1h = Change the direction through Reverse Drive while continuously driving the motor

5.3.3 User Status Registers (Base Address = 0x20200430h)

These sets of registers can be modified in the application code using pointer variable ***pUserStatusRegs*** . Table 4-29 shows the structure and definitions of the User Status Registers of Sensorless FOC Algorithm. The user status register updates every 1ms in a periodical timer interrupt.

Table 5-39. User Status Registers

Variables	Reset Value	Description
SYSTEM_FAULT_STATUS	NO_FAULT	Defines the status of motor faults. **MOTOR_STALL**: Indicates motor lock faults - abnormal BEMF, no motor, or abnormal speed **VOLTAGE_OUT_OF_BOUNDS**: Indicates undervoltage or overvoltage **LOAD_STALL**: Indicates IPD fault. HARDWARE_OVER_CURRENT: Indicates DC bus current limit fault **HARDWARE_OVER_CURRENT**: Indicates overcurrent in the DC bus **HV_DIE**: Indicates gate driver fault if applicable (nFAULT input pin)
MOTOR_STATE	MOTOR_IDLE	Defines the state of Current Motor Running Status **MOTOR_IDLE**: Motor Idle State **MOTOR_ISD**: Motor in Initial Speed Detection state **MOTOR_TRISTATE**: Motor in Tristate or Hi-Z mode **MOTOR_BRAKE_ON_START**: Motor Brake during Start up **MOTOR_IPD**: Motor in Initial Position Detection state **MOTOR_SLOW_FIRST_CYCLE**: Motor in Slow First Cycle start state **MOTOR_ALIGN**: Motor in Align start state **MOTOR_OPEN_LOOP**: Motor in Open Loop ramp up state. **MOTOR_CLOSE_LOOP_UNALIGNED**: Motor in Closed Loop run state with Angle unaligned **MOTOR_CLOSE_LOOP_ALIGNED**: Motor in Closed Loop run state with aligned angle **MOTOR_SOFT_STOP**: Motor in Stop state **MOTOR_BRAKE_ON_STOP**: Motor in Brake stop state **MOTOR_FAULT**: Motor in Motor Fault state
V_DQ_FILT	IQ27(0)	Indicates the filtered Vd and Vq applied to the motor. Output of current PI controllers.
I_DQ_PI	IQ27(0)	Indicates the Kp and Ki values of current PI controllers.
PI_SPEED	IQ27(0)	Indicates the reference and feedback values of speed PI controller set by FOC algorithm in PU.
PI_POWER	IQ27(0)	Indicates the reference and feedback values of Power PI controller set by FOC algorithm in PU.
PI_ID	IQ27(0)	Indicates the reference and feedback values of direct current PI controller set by FOC algorithm in PU.
PI_IQ	IQ27(0)	Indicates the reference and feedback values of quadrature current PI controller set by FOC algorithm in PU.
IPD_IDENTIFIED_SECTOR	0b	Indicates the IPD identified nearest rotor state.
ESTIMATED_SPEED	IQ27(0)	Indicates the motor speed in PU estimated by the FOC observer algorithm.
DC_BUS_VOLTAGE	IQ27(0)	Indicates the DC bus voltage vector value in PU
TORQUE_LIMIT	IQ27(0)	Indicates the quadrature current controller saturation limit set by FOC. This value is based on the limit set in ClosedLoop1 configuration.
GATE_DRIVER_FAULT_STATUS	0x00000000h	Defines the Index of Gate Driver Specific faults as defined in gateDriverLib.
CONTROLLER_FAULT_STATUS	0x00000000h	Defines the Index of FOC Control Algorithm Specific Faults as defined in main.h
APP_VERSION	0x00000000h	Defines the Version number of Application Firmware

6 Quick Start Guide

TI provides LaunchPad™ development kits to evaluate MSPM0 Arm Cortex-M0+ microcontrollers and evaluation modules (EVMs) to evaluate the DRV83xx family of brushless-DC motor drivers. These evaluation boards are available on TI.COM and can be used as a system evaluation platform for Sensorless, Universal, or Sensored FOC motor control. Refer to the following table for the hardware connection with chosen DRV83xx EVM board.

Table 6-1. Hardware Connection Guide for LaunchPad and EVM Boards

FOC Algorithm	SDK User's Guide(1)
Sensorless FOC	Sensorless FOC SDK User's Guide
Universal FOC	Universal FOC SDK User's Guide
Sensored FOC	Sensored FOC SDK User's Guide

(1) Includes library overview, software setup, hardware setup, and more.

6.1 CCS IDE

6.1.1 Project Setup

Follow these steps below to set up the debug environment in CCS.

1. Go to Import CCS Projects.

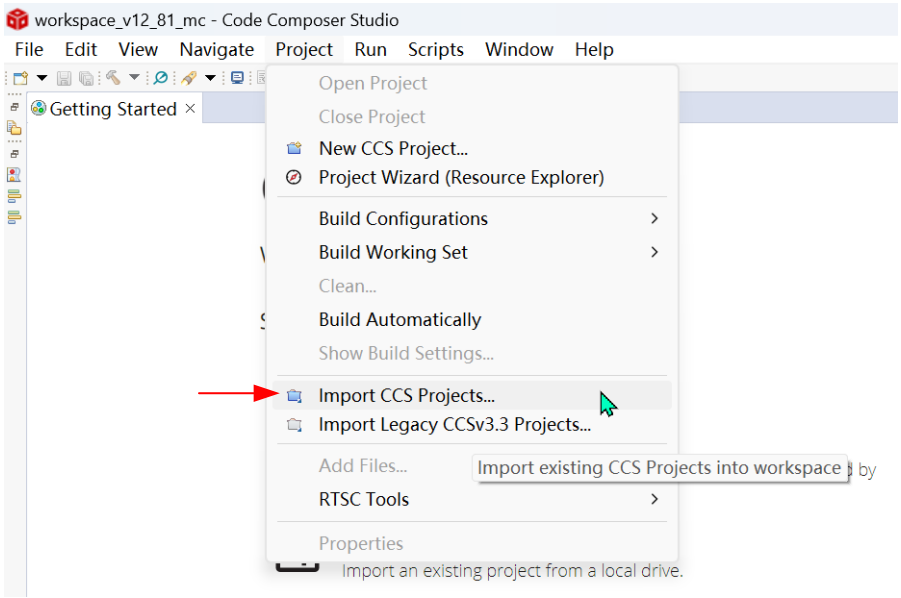


Figure 6-1. Import CCS Projects

2. Click Browse and Navigate to the address below:

```
C:\ti\mspm0_sdk_<SDK_Version>\examples\nortos\[LAUNCHPAD]\motor_control_xxxx_xxxx_foc
```

Figure 6-2 shows the flow:

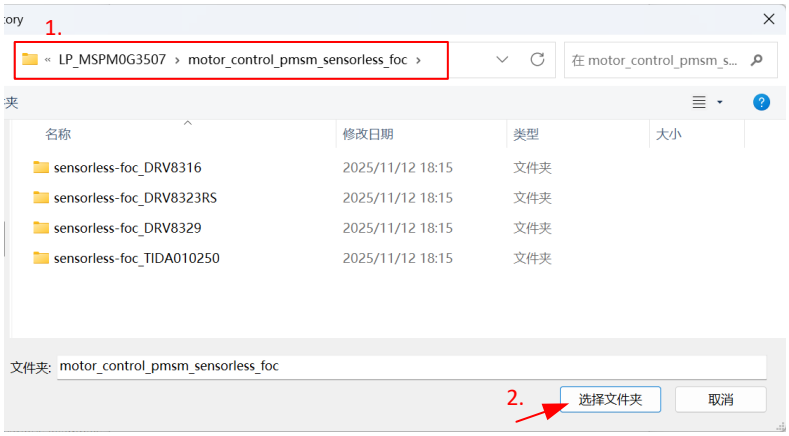


Figure 6-2. Navigate to SDK Projects

3. Click Select Folder. Check the desired hardware board and click Finish to import the project into your workspace.

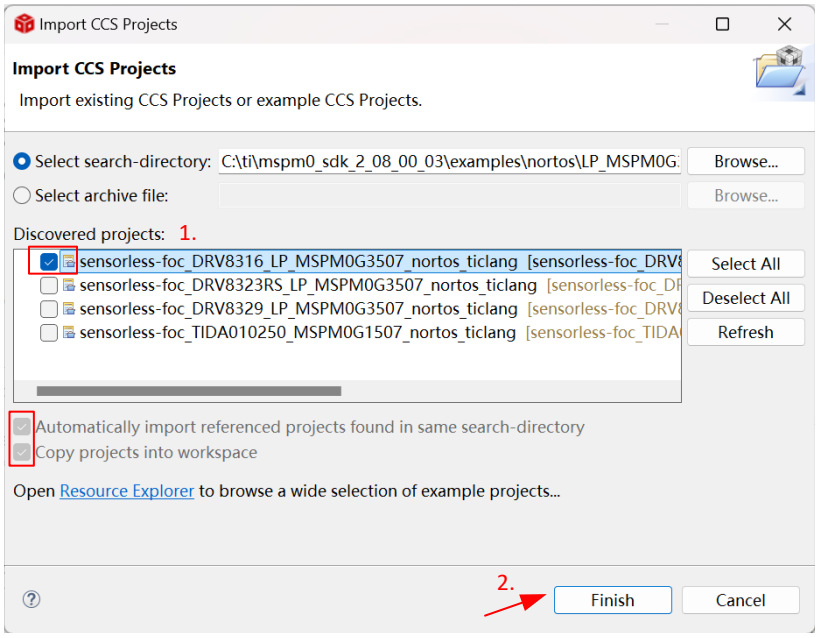


Figure 6-3. Select and Import the SDK Project

4. The project is shown in Project Explorer and ready to debug.

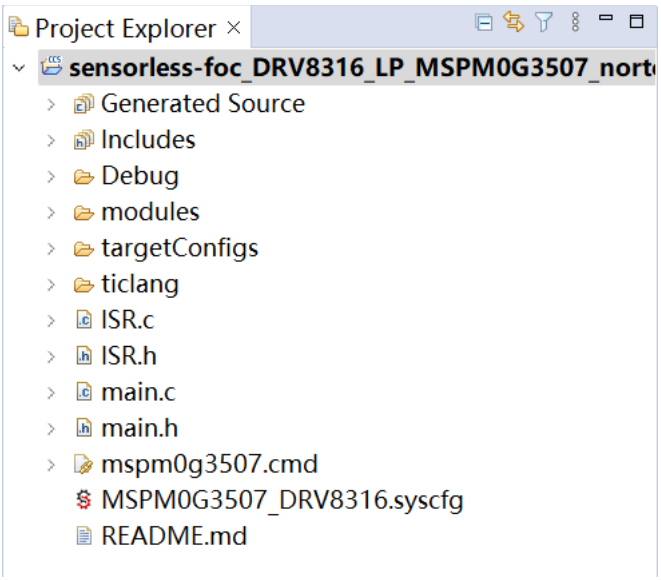


Figure 6-4. Example Project Overview

6.1.2 Project Debug

Follow the steps below to debug the FOC project.

1. Connect the hardware and turn on the power supply. There should be no more than 50mA on the power supply.
2. Click on the Build button: . Project should build with no errors.
3. Click on the Debug button: .
4. Open the Expressions window and add the following global structures pointers: ***pUserCtrlRegs*** / ***pUserInputRegs*** / ***pUserStatusRegs*** .

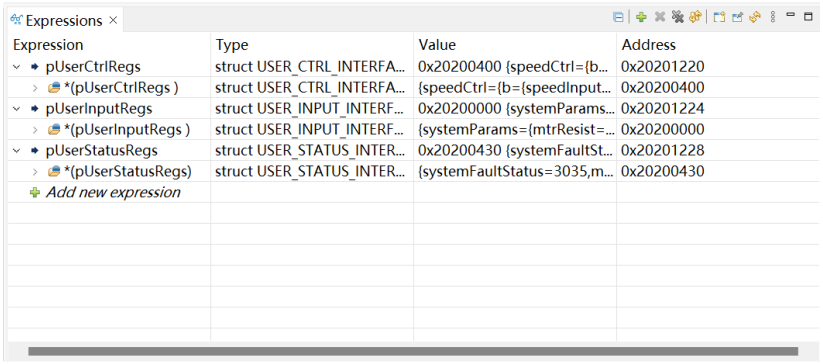


Figure 6-5. Expressions Window

5. Enable “Continuous Refresh” in the Expressions window: .
6. Press Play button : to start the foc application.

6.2 GUI

For GUI Quick Start Guide, refer to the FOC Tuning Guide in the Table 6-2 below.

Table 6-2. FOC Tuning Guide

FOC Algorithm	FOC Tuning Guide(1)
Sensorless FOC	Sensorless FOC SDK User’s Guide
Universal FOC	Universal FOC SDK User’s Guide

Table 6-2. FOC Tuning Guide (continued)

FOC Algorithm	FOC Tuning Guide[1]
Sensored FOC	Sensored FOC SDK User's Guide

(1) Includes GUI setup, basic and advanced function tuning, and more.

7 Motor Tuning Guide

This section provides a comprehensive reference workflow designed to guide newcomers through the MSPM0 FOC project implementation process. This step-by-step guidance ensures that users can efficiently navigate the initial setup, motor spinning procedures, and subsequent tuning processes required to achieve optimal motor control performance.

7.1 Hardware Board Parameter

The hardware board parameters contain the ADC sensing range for current and voltage. The FOC algorithm uses the per unit value to standardize measurements and calculations. Thus, users should set correct board parameters according to the ADC sampling hardware circuit.

The following table shows the variable structure of the hardware board parameters.

Table 7-1. Hardware Board Parameter

Variable	Description
pUserInputRegs-> systemParams.voltageBase	Base voltage of the board calculated as the maximum measurable voltage (in volts) detailed in equation: MAX_DC_VOLTAGE / SQRT(3)
pUserInputRegs-> systemParams.currentBase	Base current of the board calculated based on the CSA gain in amps

Note

Incorrect board parameters result in inaccurate power, voltage and current control loops, and degraded FOC control efficiency.

7.1.1 Base Voltage (V)

Max Voltage represents the maximum measurable bus voltage and phase voltages in the motor control system. See the Figure 7-1 for hardware configuration of the voltage divider scaling ratio.

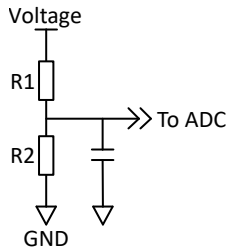


Figure 7-1. ADC Voltage Divider for Voltage Detection

Users can compute the system MaxVoltage based on the voltage scaling resistor divider bridge values R1 and R2 and the Full-Scale ADC Voltage (FSV) of 3.3V as shown in Equation 27:

$$\text{Max Voltage}=\text{FSV} / \text{Voltage Divider Scaling Ratio} = 3.3\text{V} \times (\text{R1}+\text{R2}) / \text{R2} \quad (27)$$

Base Voltage represents the maximum voltage in FOC control system, and it is the voltage base value in the per unit system:

$$\text{Base Voltage} = \text{Max Voltage} / \text{SQRT}(3) \quad (28)$$

For example, in a system with resistor divider scaling ratio of 1/20 from DC supply voltage to ADC input, the maximum measurable system voltage by the ADC is 3.3V / (1/20) = 66V, and the Base Voltage is 38.1V.

Note

Use the same hardware voltage divider for bus voltage and motor phase voltage detection.

7.1.2 Base Current (A)

Base Current represents the maximum measurable motor phase current in the motor control system.

Users can compute the system Base Current based on the current sense amplifier gain (CSAGAIN) in volts/amp and the Full-Scale ADC Voltage (FSV) as shown in Equation 29. There is a factor of 2 considered to support bidirectional current sensing with 1.65V as the zero-current offset, as shown in Figure 7-2.

$$\text{Base Current} = (\text{FSV} / 2) / \text{CSAGAIN [V/A]} \tag{29}$$

For example, in a system with CSAGAIN = 0.15V/A, the base current or maximum measurable system current by the ADC is 3.3V / (2 × 0.15V/A) = 11A.

If the system uses a current sense resistor (R_{SENSE}) with CSAGAIN units mentioned in volt/volt (V/V), the CSA gain in volts/amp can be computed using Equation 30.

$$\text{CSAGAIN [V/A]} = \text{RSENSE} \times \text{CSAGAIN [V/V]} \tag{30}$$

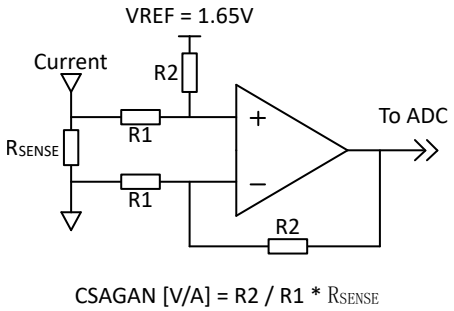


Figure 7-2. Bidirectional Current Sensing Circuit

If the system uses a unidirectional current sensing circuit, commonly used in the single shunt current sensing method, the zero-current offset is removed, as shown in Figure 7-3. OffsetVoltage is introduced and represents the common mode voltage of amplifier used to obtain maximum resolution for current measurement using Equation 31.

$$\text{Base Current} = (\text{FSV - OffsetVoltage}) / \text{CSAGAIN [V/A]} \tag{31}$$

For example, in the DRV8329 system with OffsetVoltage = V_{CSAREF}/8 and CSAGAIN = 0.4V/A, the base current or maximum measurable system current by the ADC is (3.3V – 3.3V/8) / (0.4V/A) = 7.22A.

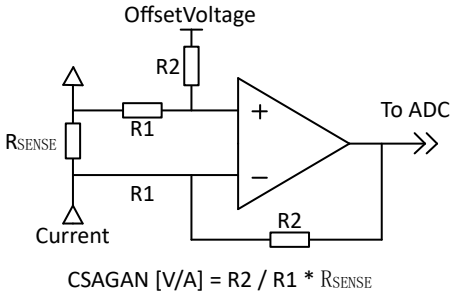


Figure 7-3. Unidirectional Current Sensing Circuit

7.2 Motor Parameter

The motor parameters contain the resistance, inductance, pole pairs, BEMF, and max speed. The following table shows the variable structure of the motor parameters.

Table 7-2. Motor Parameter

Variable	Description
pUserInputRegs ->systemParams.mtrResist	Motor phase resistance in milliohms
pUserInputRegs ->systemParams.mtrInductance	Motor inductance in microhenry. For salient pole motors: (Lq + Ld)/2
pUserInputRegs ->systemParams.mtrSaliency	Saliency of the motor (Lq-Ld)/(Lq+Ld) in float
pUserInputRegs ->systemParams.mtrBEMFConst	Motor BEMF constant in mV/Hz × 10
pUserInputRegs ->systemParams.maxMotorSpeed	Rated motor electrical speed in Hz from the motor data sheet
pUserInputRegs ->systemParams.maxMotorPower	Rated motor electrical speed in Hz from the motor data sheet
pUserInputRegs ->systemParams.polePairs	Apply for **Sensored FOC** only, represent total pole pairs of the motor or number of poles / 2

7.2.1 Motor Phase Resistance (mΩ)

Users can get the motor phase resistance from the motor data sheet. If the motor does not have a data sheet, then measure the phase-to-phase resistance across any two phases using a digital multimeter and calculate the phase resistance by dividing the phase-to-phase resistance by 2 as shown in Equation 32.

$$\text{Phase resistance} = \text{Measured Phase-to-Phase Resistance} \times 0.5 \tag{32}$$

This measurement is valid for both star wound and delta wound motors. In the project, we equate it to a star wound motor and apply its motor-related parameters to participate in the control algorithm.

7.2.2 Motor Phase Inductance (μH)

Users can calculate the motor phase inductance and saliency from the motor data sheet, as shown in Equation 33 and Equation 34.

$$\text{Motor Inductance} = (\text{Lq} + \text{Ld}) \,/\, 2 \tag{33}$$

$$\text{Motor Saliency} = (\text{Lq} - \text{Ld}) \,/\, (\text{Lq} + \text{Ld}) \tag{34}$$

If the motor does not have a data sheet, to know the motor inductance, measure the phase-to-phase inductance at 1kHz across any two phases using an LCR meter. Calculate the phase inductance by dividing the phase-to-phase inductance by 2, as shown in Equation 35. The saliency can be set as 0.

$$\text{Phase Inductance} = \text{Measured Phase-to-Phase Inductance} \times 0.5 \tag{35}$$

7.2.3 Saliency of IPMSM Motor

Saliency of an IPMSM motor is a measure of variation in Inductance between the quadrature axis and direct rotor axis. For FOC algorithm this value is to be given as Equation 34 in float variable.

The simplest method to deduce the Ld and Lq values is by measuring the inductance across any two phases and vary the rotor position slowly for one full rotation. The maximum measured inductance value can be noted as Lq, and the minimum measured inductance value can be noted as Ld.

7.2.4 Motor Pole Pairs

Users can get the motor pole pairs from the motor data sheet. The pole pairs parameter is widely used to calculate electrical parameters, including BEMF and max motor speed. For Sensored FOC solution, the pole pairs parameter is the key feature to execute hall sensor calibration.

If the motor does not have a data sheet, follow the steps below:

1. Use a lab power supply and make sure the current limit is set to less than the motor rated current. Do not turn on the supply.
2. Connect V+ of the supply to phase A and V- of the supply to phase B of the motor. Any 2 of the 3 phases can be chosen at random if the phases are not labeled.
3. Turn on supply, The rotor settles at one position by injecting current.
4. Manually rotate the rotor until rotor snaps to another settle position. The rotor settles down at various positions around one mechanical cycle.
5. Count the number of settle-down positions for one fully mechanical cycle, which is the number of pole pairs. Multiplying by two calculates the number of poles.

Be careful of gearing systems within a motor. The gearing ratio determines how many rotor revolutions correlate to the shaft mechanical revolution.

7.2.5 Motor BEMF Constant (mV/Hz)

Using the motor's data sheet, the user can input the motor's BEMF constant ***Ke*** in mV/Hz and program mtrBEMFConst as ***Ke*** × 10. For example, if a motor's BEMF constant is 40mV/Hz, then user should set mtrBEMFConst = 400.

Sometimes the motor data sheet clarifies motor's BEMF constant with a different unit. Use Equation 36 and Equation 37 to convert it to mV/Hz.

$$\text{BEMF Constant [mV/Hz]} = \text{BEMF Constant [mV/rpm]} \times 60 \,/\, \text{Pole Pairs} \tag{36}$$

$$\text{BEMF Constant [mV/Hz]} = \text{BEMF Constant [mN·m/A]} \times 2\pi \,/\, \text{Pole Pairs} \tag{37}$$

If the motor does not have a data sheet, measure the voltage across any two phases of the motor using an oscilloscope by manually spinning the motor. A sinusoidal or trapezoidal voltage appears on the oscilloscope. Measure the peak voltage ***Ep*** in milli-volts and period ***Tp*** in seconds. The following figure shows the test motor waveform captured by a scope. Then, users can calculate BEMF constant ***Ke*** as shown in Equation 38.

$$\text{BEMF Constant Ke [mV/Hz]} = \text{Ep} \times \text{Tp} \,/\, \text{SQRT}(3) \tag{38}$$

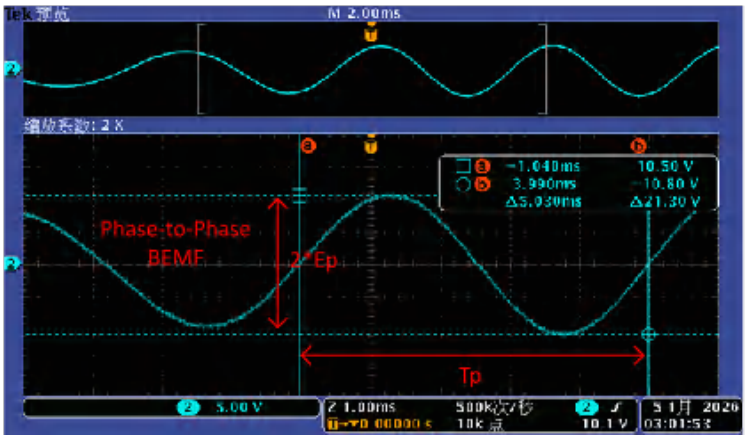


Figure 7-4. Motor Phase-to-Phase BEMF

7.2.6 Maximum Motor Electrical Speed (Hz)

Using the motor's data sheet, the user can input the maximum motor electrical speed in Hz. If this data is not available, the user can convert the motor mechanical speed in rpm to motor electrical speed in Hz using Equation 39.

$$\text{Max Motor Speed [Hz]} = \text{Pole Pairs} \times \text{Mechanical Speed [rpm]} \,/\, 60 \tag{39}$$

7.2.7 Maximum Motor Power (W)

Users need to input the Maximum Power Rating of the motor when Closed loop Power Control is required. To determine the Maximum Rated Power of the motor, see the Motor data sheet and compute the product of Rated Motor Voltage in Volts and Rated Motor Current in Amps and feed the value to MOTOR_MAX_POWER in the System parameters.

7.3 Control Loop Parameter

The FOC algorithm implements Proportional-Integral (PI) control regulator for closed loop control, including current loop and speed loop. The power loop replaces the speed loop if enabled by users.

The PI regulator contains two parameters, Kp and Ki, for user's tuning. Use default PI parameters to spin the motor and tune it during the run-time to improve control performance.

7.3.1 Speed / Power Loop

The FOC Algorithm uses an integrated Speed/Power Control Loop that helps maintain a constant Speed/Power over varying operating conditions, as shown in Table 7-3.

Table 7-3. Speed / Power Loop PI Parameter

Variable	Description
pUserInputRegs ->systemParams.speedLoopKp	Proportional gain for the closed loop speed / power control in float
pUserInputRegs ->systemParams.speedLoopKi	Integral gain for the closed loop speed / power control in float

The output of the speed loop / power Loop is used to generate the current reference for torque (current loop) control. When output of the speed loop / Power Loop saturates, the integrator is disabled to prevent integral wind-up. The tuning of speed loop Kp and Ki is experimental. Table 7-4 shows general guidelines for changing PI parameters.

Table 7-4. Steady State and Transient Response Trends with PI Gain Coefficients Increase

Parameter	Rise Time	Overshoot	Settling Time	Stead State Error	Stability
Kp	Decreases	Increases	Small Change	Decreases	Degrades
Ki	Decreases	Increases	Increases	Eliminates	Degrades

7.3.2 Current Loop

FOC algorithm has two PI controllers, one each for Id and Iq to control flux and torque separately. Kp and Ki coefficients are the same for both PI controllers and are configured in Table 7-5. The outputs of the current control loops are used to generate voltage signals Vd and Vq to be applied to the motor. The outputs of the current loops are clamped to supply voltage VM. Id current PI loop is executed first and output of Id current PI loop Vd is checked for saturation. When the output of the current loop saturates, the integration is disabled to prevent integral wind-up.

Table 7-5. Current Loop PI Parameter

Variable	Description
pUserInputRegs ->systemParams.currLoopKp	Proportional gain users set for the closed loop torque control in float
pUserInputRegs ->systemParams.currLoopKi	Integral gain users set for the closed loop torque control in float

The FOC application supports automatically calculating the current loop PI parameters based on the motor parameters (resistance and inductance). By default, the current loop bandwidth is set as 0.03 times the FOC loop frequency in the algorithm. For example, if users set FOC loop as 10kHz, then the default current loop bandwidth is 300Hz.

Set the variables described in Table 7-5 as 0 to enable the PI parameters auto calculation. Users can then get the computed PI parameters in Table 7-6.

Table 7-6. Auto Calculated PI Parameters of Current Loop

Variable	Description
pUserStatusRegs ->currentPI.Kp	Read Only. Proportional / Integral gain for current loop. The value is passed from variable: ***pUserInputRegs*** ->systemParams.currLoopKp / currLoopKp (if either has non-zero value). The value is generated by FOC algorithm if is ***pUserInputRegs*** ->systemParams. currLoopKp / currLoopKp are both zero.
pUserStatusRegs ->currentPI.Ki	

After getting auto calculated PI parameters shown in Table 7-6, users can set the calculated parameters value as the initial value of the PI parameters shown in Table 7-5. Users can then do manual parameters tuning based on the actual control performance.

Note

There is a scaling factor relationship between the current loop PI parameters set by the user in ***pUserInputRegs*** and the actual PI parameters used in the PI controller calculations, as the motor speed and current are based on PU values.

7.4 Hall Angle Table

In **sensored FOC**, Hall signals are used to detect the rotor position information and drive the motor efficiently. FOC application requires three digital Hall Inputs placed 60 degrees electrically connected to the GPIO's as inputs feeding the rotor position information.

User needs to populate the Hall Angle Table in IQ27 (PU) appropriately for a given Hall Sequence with reference to motor phase connections, as shown in the following table.

Table 7-7. Hall Angle Table

File	Constant Variable	Description
ISR.c	***forwardHallIndexLUT[MAX_HALL_INDEX]***	Hall angle table used for forward rotation.
	reverseHallIndexLUT[MAX_HALL_INDEX]	Hall angle table used for reverse rotation.

In general, the Hall Pin sequence with reference to the Motor Phase connection is introduced in the motor's data sheet. The Hall Pin sequence is significant and the driving angle for a given phase depends on the sequence of connections. However, the Hall sensor placements can be erroneous, and the electrical displacement can be less than or greater than 60 degrees. Any error in the hall placement results in torque ripples and non-sinusoidal current. It is essential for users to do Hall sensor calibration to correct the hall placement error for improved current waveform in the motor.

7.4.1 Hall Calibration

The hall angle table values shown in Table 7-7 can be auto generated with **Hall Calibration Routine** as described below:

1. Motor_Align: Aligning the motor to a known rotor angle is the first step in the calibration sequence. In this routine, the motor aligns to the angle (CALIBRATION_ALIGN_ANGLE macro) for a duration (pUserInputRegs->mtrStartUp1.b.calibAlignTime). Sufficient calibAlignTime is to be configured such that the motor aligns and stops movement before the calibration is initiated.
2. Motor_Calib_Run: Once Motor_Align is successfully completed, the rotor micro steps with an angle specified as in the macro (CALIBRATION_ANGLE_STEP) for a duration (pUserInputRegs->mtrStartUp1.b.calibRunTime). Motor rotates for one complete mechanical cycle in both forward and reverse direction to compute the average Hall angle.
3. Motor_Calib_Complete: Once Motor_Calib_Run is successfully completed, rotor angle for each Hall State transitions for both Forward and Reverse directions are generated in the Hall angle tables as g_pMC_App->hallAngleTableForward and g_pMC_App->hallAngleTableReverse.

The following table shows the related variables used for Hall calibration.

Table 7-8. Hall Calibration Variables

Variable / Macro	Description
pUserInputRegs ->mtrStartUp1.b.calibCurrLimit	Current limit during Hall calibration in % of base current.
pUserInputRegs ->mtrStartUp1.b.calibAlignTime	Time taken for initial rotor alignment to CALIBRATION_ALIGN_ANGLE.
pUserInputRegs ->mtrStartUp1.b.calibRunTime	Time spent while calibrating Hall for each CALIBRATION_ANGLE_STEP.
pUserInputRegs ->mtrStartUp1.b.currRampRate	Initial current ramp rate during Motor_Align till the maximum current is reached.
pUserCtrlRegs ->algoDebugCtrl2.b.hallCaliEnable	Use to enable Automatic Hall Calibration.
pUserCtrlRegs ->speedCtrl.b.speedInput	Target Motor Speed/Torque value in % of Speed or Torque command × 32768.
g_pMC_App->hallAngleTableForward	Hall angle table used for forward rotation. The value is passed from ***forwardHallIndexLUT*** if not perform Hall calibration.
g_pMC_App->hallAngleTableReverse	Hall angle table used for reverse rotation. The value is passed from ***reverseHallIndexLUT*** if not perform Hall calibration.
CALIBRATION_ALIGN_ANGLE	Initial rotor alignment defined in ***hallCalib.h*** file.
CALIBRATION_ANGLE_STEP	Calibration angle step size defined in ***hallCalib.h*** file.

7.4.2 Register Table

This section introduces the register table for Hall Calibration.

MOTOR_STARTUP1 Register (Sensored FOC)

Table 7-9 shows the register to configure motor startup settings1.

Table 7-9. MOTOR_STARTUP1 Register Field Descriptions

Bit	Field	Type	Reset	Description
31-30	RESERVED	R	00b	Reserved
29-26	CURR_RAMP_RATE	R/W	0h	Initial current ramp rate during Hall Calibration till the Maximum current is reached. 0h = 0.1A/s 1h = 1A/s 2h = 5A/s 3h = 10A/s 4h = 15A/s 5h = 25A/s 6h = 50A/s 7h = 100A/s 8h = 150A/s 9h = 200A/s Ah = 250A/s Bh = 500A/s Ch = 1000A/s Dh = 2000A/s Eh = 5000A/s Fh = No Limit A/s

Table 7-9. MOTOR_STARTUP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
25-22	CALIB_RUN_TIME	R/W	0h	Time spent while calibrating Hall for each CALIBRATION_ANGLE_STEP(defined in hallCalib.h) 0h = 10ms 1h = 50ms 2h = 100ms 3h = 200ms 4h = 300ms 5h = 400ms 6h = 500ms 7h = 750ms 8h = 1s 9h = 1.5s Ah = 2s Bh = 3s Ch = 4s Dh = 5s Eh = 7.5s Fh = 10s
21-17	CALIB_CURRENT_ILIMIT	R/W	00h	Current limit during Hall Calibration in % of CURRENT_BASE 0h = 7.5% 1h = 8.0% 2h = 8.5% 3h = 9.0% 4h = 9.5% 5h = 10% 6h = 11% 7h = 12% 8h = 13% 9h = 14% Ah = 15% Bh = 16% Ch = 17% Dh = 18% Eh = 20% Fh = 22.5% 10h = 25% 11h = 27.5% 12h = 30% 13h = 35% 14h = 40% 15h = 45% 16h = 50% 17h = 55% 18h = 60% 19h = 70% 1Ah = 75% 1Bh = 80% 1Ch = 85% 1Dh = 90% 1Eh = 95% 1Fh = 100%

Table 7-9. MOTOR_STARTUP1 Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
16-13	CALIB_ALIGN_TIME	R	000b	Time taken for initial rotor alignment to Zero Degrees before starting Hall Calibration. 0h = 10ms 1h = 50ms 2h = 100ms 3h = 200ms 4h = 300ms 5h = 400ms 6h = 500ms 7h = 750ms 8h = 1s 9h = 1.5s Ah = 2s Bh = 3s Dh = 5s Eh = 7.5s Fh = 10s
12-2	RESERVED	R	0h	Reserved
1	OL_ILIMIT_CONFIG	R/W	0b	Open loop current limit configuration 0h = Open loop current limit defined by OL_ILIMIT 1h = Open loop current limit defined by ILIMIT
0	RESERVED	R	0b	Reserved

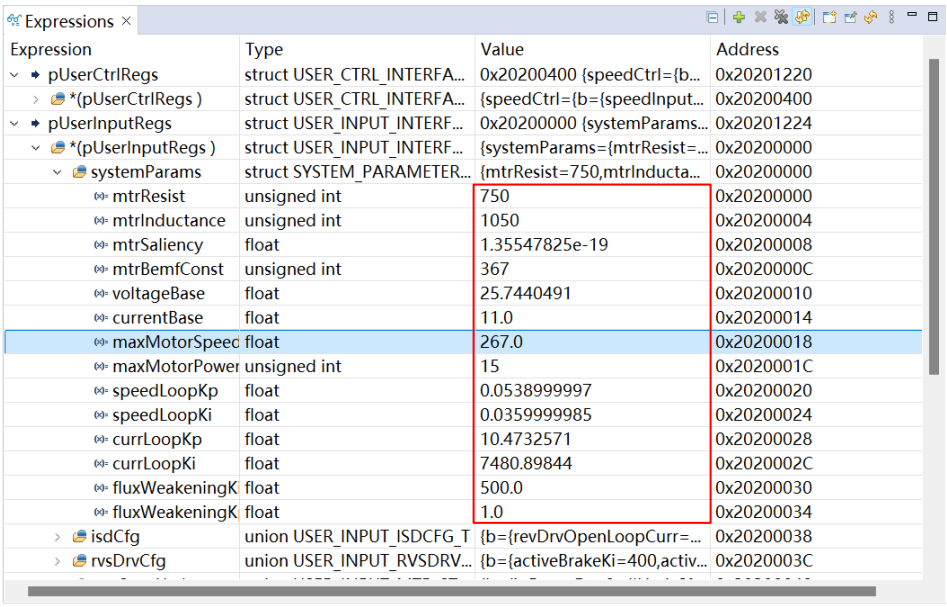
7.5 Spin the Motor (LVBLDC)

This section gives an example of how to spin the LVBLDC motor with DRV8316REVM. Users can find the related parameters in the product page or refer to the following table.

Table 7-10. LVBLDC Parameter

Part Number	Rated				Line to Line		Torque Constant	BEMF	Inertia	Pole Pairs
	Voltage	Speed	Torque	Power	Resistance	Inductance				
DN42040S24-026	VDC	RPM	mNm	W	Ohm	mH	mNm/A	V/Krpm	g·cm2	/
	24	4000	62.5	26	1.50	2.10	35.00	2.45	24	4

Refer to Section 6.1 to run the **sensorless-foc_DRV8316** project. In Debug mode, users can dynamically overwrite the FOC control registers in Expression window. Figure 7-5 shows the variable result overwritten to match the motor parameters.



Expressions

Expression	Type	Value	Address
pUserCtrlRegs	struct USER_CTRL_INTERFA...	0x20200400 {speedCtrl={b...	0x20201220
*(pUserCtrlRegs)	struct USER_CTRL_INTERFA...	{speedCtrl={b={speedInput...	0x20200400
pUserInputRegs	struct USER_INPUT_INTERF...	0x20200000 {systemParams...	0x20201224
*(pUserInputRegs)	struct USER_INPUT_INTERF...	{systemParams={mtrResist=...	0x20200000
systemParams	struct SYSTEM_PARAMETER...	{mtrResist=750,mtrInducta...	0x20200000
mtrResist	unsigned int	750	0x20200000
mtrInductance	unsigned int	1050	0x20200004
mtrSaliency	float	1.35547825e-19	0x20200008
mtrBemfConst	unsigned int	367	0x2020000C
voltageBase	float	25.7440491	0x20200010
currentBase	float	11.0	0x20200014
maxMotorSpeed	float	267.0	0x20200018
maxMotorPower	unsigned int	15	0x2020001C
speedLoopKp	float	0.0538999997	0x20200020
speedLoopKi	float	0.0359999985	0x20200024
currLoopKp	float	10.4732571	0x20200028
currLoopKi	float	7480.89844	0x2020002C
fluxWeakeningK	float	500.0	0x20200030
fluxWeakeningK	float	1.0	0x20200034
isdCfg	union USER_INPUT_ISDCFG_T	{b={revDrvOpenLoopCurr=...	0x20200038
rvsDrvCfg	union USER_INPUT_RVSDRV...	{b={activeBrakeKi=400,activ...	0x2020003C

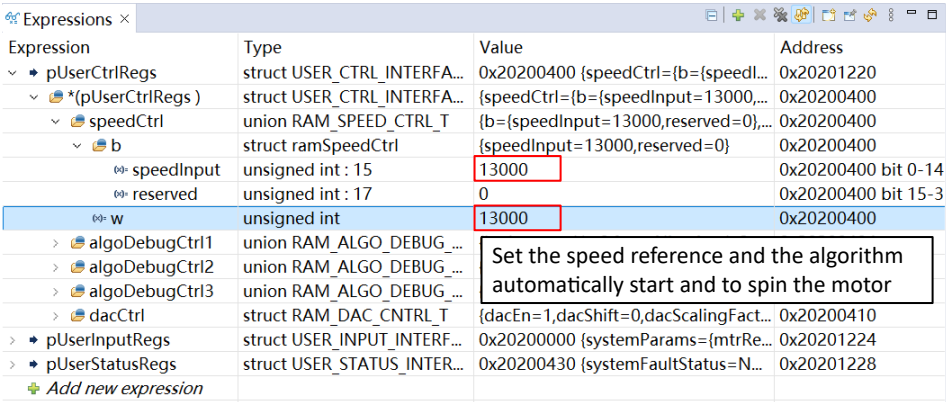
Figure 7-5. Dynamically Overwrite the FOC Control Registers in Expression Window

Motor parameters can be also hardcoded directly into the source code to avoid the tedious process of repeatedly modifying the same parameters, as shown in Figure 7-6.

```
105    focPeriphInit();  /* Does foc application specific Peripheral configurations */
106
107    /* Configure the Motor Params */
108    pUserInputRegs->systemParams.mtrResist        = 750;
109    pUserInputRegs->systemParams.mtrInductance    = 1050;
110    pUserInputRegs->systemParams.mtrBemfConst     = 367;
111    pUserInputRegs->systemParams.maxMotorSpeed = 266.7;
112
113    /* Configure the Current Loop Params - copy from auto calculated */
114    pUserInputRegs->systemParams.currLoopKp     = 10.4732571;
115    pUserInputRegs->systemParams.currLoopKi     = 7480.89844;
116
117    while (1)
118    {
```

Figure 7-6. Hardcode the Motor Parameters into The Source Code

Then, users can set value of speedInput to start FOC control and spin the motor, as shown in Figure 7-7.



Expressions

Expression	Type	Value	Address
pUserCtrlRegs	struct USER_CTRL_INTERFA...	0x20200400 {speedCtrl={b={speedI...	0x20201220
*(pUserCtrlRegs)	struct USER_CTRL_INTERFA...	{speedCtrl={b={speedInput=13000,...	0x20200400
speedCtrl	union RAM_SPEED_CTRL_T	{b={speedInput=13000,reserved=0},...	0x20200400
b	struct ramSpeedCtrl	{speedInput=13000,reserved=0}	0x20200400
speedInput	unsigned int : 15	13000	0x20200400 bit 0-14
reserved	unsigned int : 17	0	0x20200400 bit 15-3
w	unsigned int	13000	0x20200400
algoDebugCtrl1	union RAM_ALGO_DEBUG_...		
algoDebugCtrl2	union RAM_ALGO_DEBUG_...		
algoDebugCtrl3	union RAM_ALGO_DEBUG_...		
dacCtrl	struct RAM_DAC_CNTRL_T	{dacEn=1,dacShift=0,dacScalingFact...	0x20200410
pUserInputRegs	struct USER_INPUT_INTERF...	0x20200000 {systemParams={mtrRe...	0x20201224
pUserStatusRegs	struct USER_STATUS_INTER...	0x20200430 {systemFaultStatus=N...	0x20201228
Add new expression			

Set the speed reference and the algorithm automatically start and to spin the motor

Figure 7-7. Set Value of speedInput to Spin the Motor

Users can change the value of ***pUserCtrlRegs*** ->speedCtrl.w or ***pUserCtrlRegs*** ->speedCtrl.b.speedInput to change the speed in real time. The motor real-time status can be observed in ***pUserStatusReg s*** , as shown

in Figure 7-8. For example, ***pUserStatusRegs*** ->piSpeed.feedback, or ***pUserStatusRegs*** ->estimatedSpeed represents the motor real-time speed.

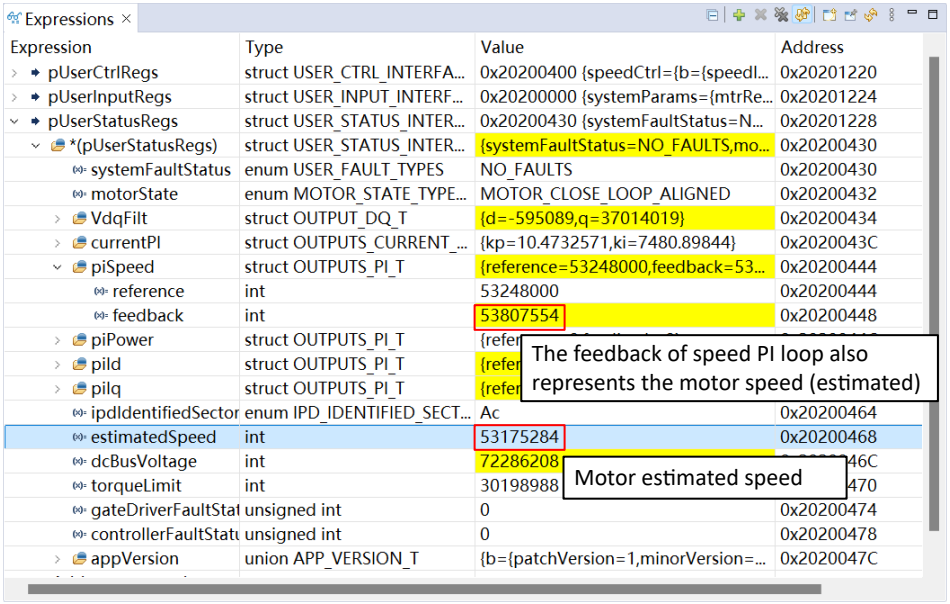


Figure 7-8. Monitor Motor Status in Expressions Window

The FOC algorithm uses fixed point data to improve code efficiency. MSPM0 driverlib provides the IQmath library for users to easily integrate the fix point data to simulate float point data. In CCS Expressions window, there is a straightforward way to display the IQ type value, as shown in Figure 7-9.

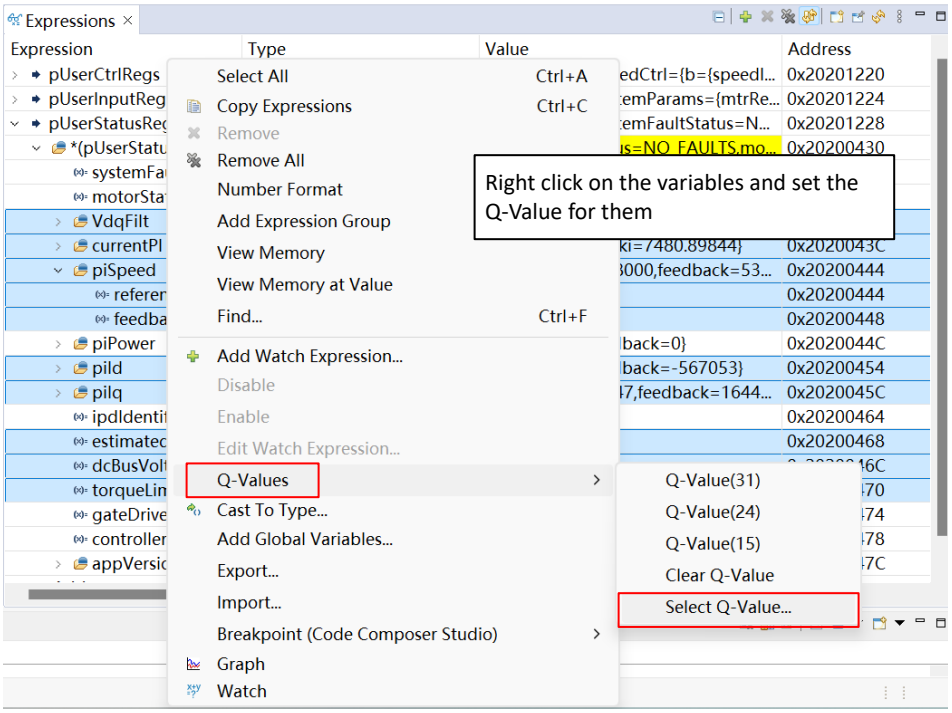


Figure 7-9. Set Variables in IQ Type

The motor speed input variable is set in IQ15 type. Other variables in registers are set in IQ27 type if apply for PU type. Figure 7-10 shows the variables displayed in IQ27 type.

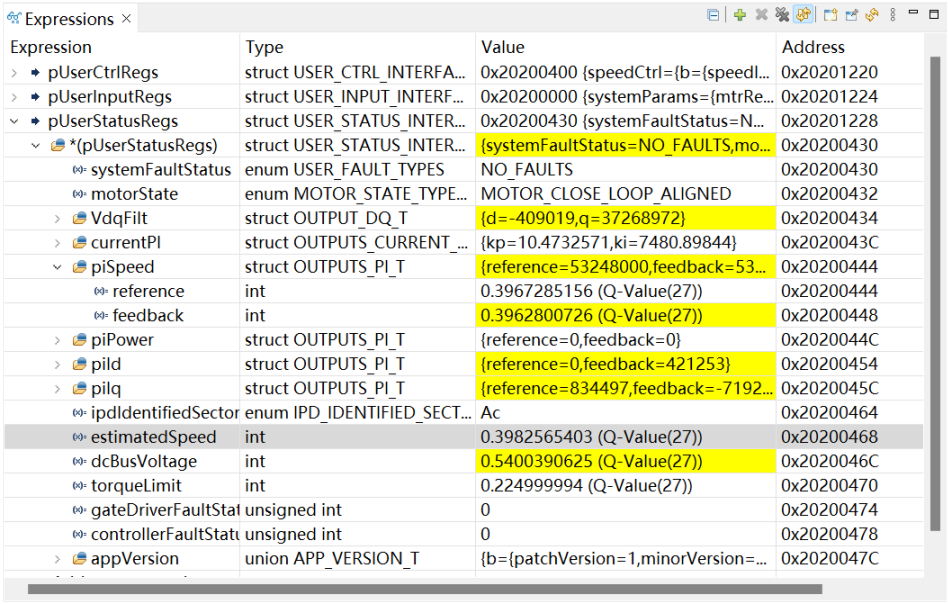


Figure 7-10. Display Variables in IQ27 Type

7.6 Spin the Motor with Hall Sensor

The section uses a standard BLDC motor (DT4260) with hall sensor to demonstrate Sensored FOC solution. The following table shows the parameters of the motor.

Table 7-11. DT4260 Motor Parameter

Part Number	Rated			Line to Line		Torque Constant	BEMF	Inertia	Pole Pairs
	Voltage	Speed	Torque	Resistance	Inductance				
DT4260-24-055-04H	VDC	rpm	mNm	Ohm	mH	mNm/A	mV/Hz	g·cm2	/
	24	4000	125	0.8	1.2	35.50	35.7	48	4

Follow the steps below to spin the motor with Sensored FOC solution:

1. Refer to Sensored FOC SDK User's Guide for hardware connection between MSPM0 Launchpad and DRV8316AEVM.
2. Refer to Section 6.1.1 to import the CSS project:

   hall_sensored-foc_DRV8316_LP_MSPM0G3507_nortos_ticlang or

   hall_sensored-foc_DRV8316_LP_MSPM0G3519_nortos_ticlang
3. Overwrite the motor parameters according to motor datasheet (Table 7-11), or manual measurement (Section 7.2).
4. Set the Hall calibration parameters according to Section 7.4.1.
5. Set the Hall Sensor Calibration Enable bit in Algo Debug Control 2 Register.
6. Add expressions below:
   - g_pMC_App->foc.hallCalibObj.calibState
   - g_pMC_App->hallAngleTableForward
   - g_pMC_App->hallAngleTableReverse
7. Set the **speedInput** to non-zero value to start the Hall calibration.
8. After Hall calibration completed, set the **speedInput** to zero to reset the FOC state machine.
   - Users should update the hardcoded Hall Angle Table in ISR.c with the newly calibrated hall angle values (See Table 7-7).
9. Set the **speedInput** to non-zero value to spin the motor.

Note

Hall calibration is required to spin a new BLDC motor with hall interface.

Figure 7-11 shows the hardcode to set the motor parameters (DT4260) and Hall calibration parameters.

```
105    focPeriphInit();  /* Does foc application specific Peripheral configurations */
106
107    /* Configure the Motor Params */
108    pUserInputRegs->systemParams.mtrResist        = 400;    //mOhm
109    pUserInputRegs->systemParams.mtrInductance    = 600;    //uH
110    pUserInputRegs->systemParams.mtrBemfConst     = 357;    //0.1 mV/Hz
111    pUserInputRegs->systemParams.polePairs     = 4;         // Sensored foc only
112    pUserInputRegs->systemParams.maxMotorSpeed = 266.7;
113
114    /* When set as 0, the algorithm will automatically calculate current loop parameters */
115    pUserInputRegs->systemParams.currLoopKp     = 0;
116    pUserInputRegs->systemParams.currLoopKi     = 0;
117    /* Configure the Current Loop Params - copy from auto calculated */
118//    pUserInputRegs->systemParams.currLoopKp     = 5.9847188;
119//    pUserInputRegs->systemParams.currLoopKi     = 3989.8125;
120
121    /* COnfigure HAL Table Calibration */
122    pUserInputRegs->mtrStartUp1.b.calibCurrLimit = 0x0; // 7.5%*base_current, Id
123    pUserInputRegs->mtrStartUp1.b.currRampRate = 0x1;   // 1 A/s8
124    pUserInputRegs->mtrStartUp1.b.calibAlignTime = 0x6; // 500ms for 0 degree
125    pUserInputRegs->mtrStartUp1.b.calibRunTime = 0x2;   // 100ms for each step
126    /*
127     * Note:
128     * Set macro definition CALIBRATION_ALIGN_ANGLE in hallCalib.h for initial position alignment
129     * Set macro definition CALIBRATION_ANGLE_STEP in hallCalib.h to use different step
130     */
131    while (1)
132    {
133        if(gdReadTestEn)
134        {
135            regData =  gateDriverRegRead(regAddr);
136        }
137        UART_checkForCommand(pUART);
138
139        updateConfigs();
140    }
141}
```

Figure 7-11. Hardcode for Hall Parameter Configuration

Figure 7-12 and Figure 7-13 show the detailed flow for Hall calibration execution in CCS *Expressions* Window.

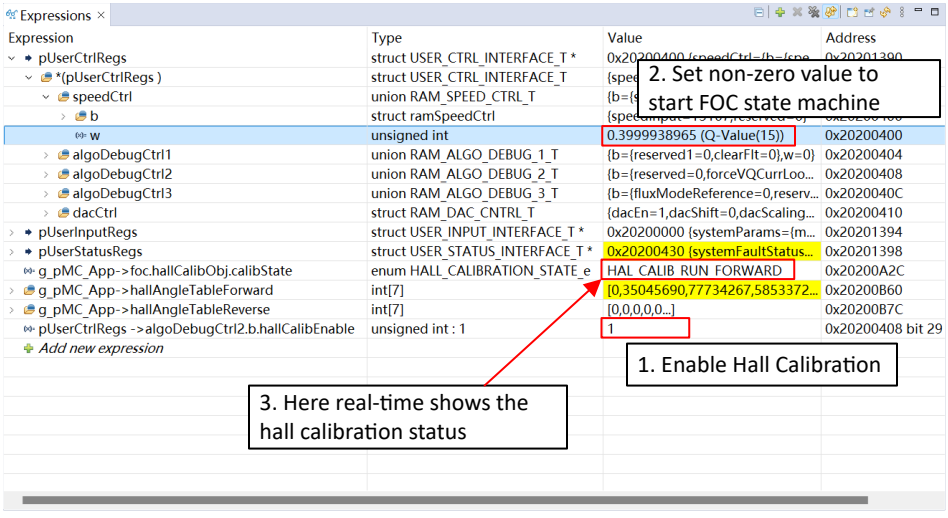


Figure 7-12. Hall Calibration Run

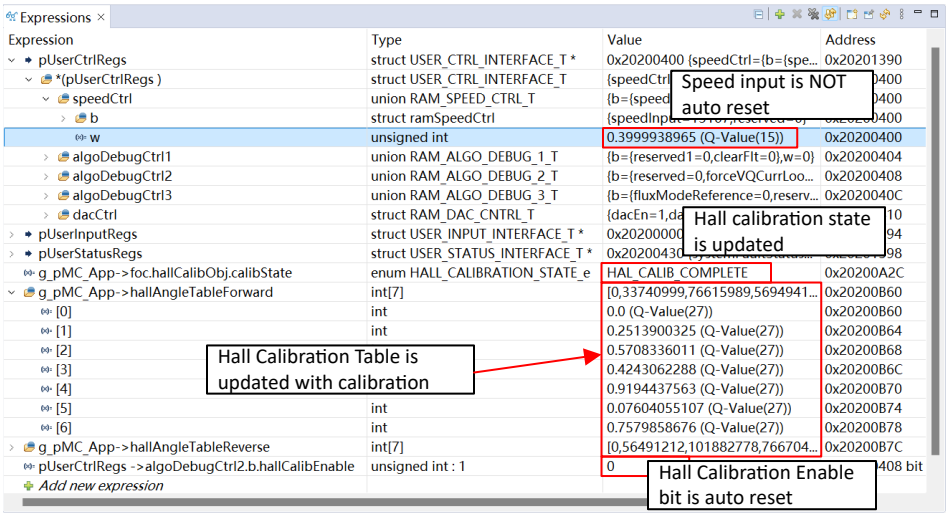


Figure 7-13. Hall Calibration Done

Users need manually set **speedInput** to zero to reset the FOC state machine. Then, users set a target speed in IQ15 format to **speedInput** register to spin the motor, as shown in Figure 7-14.

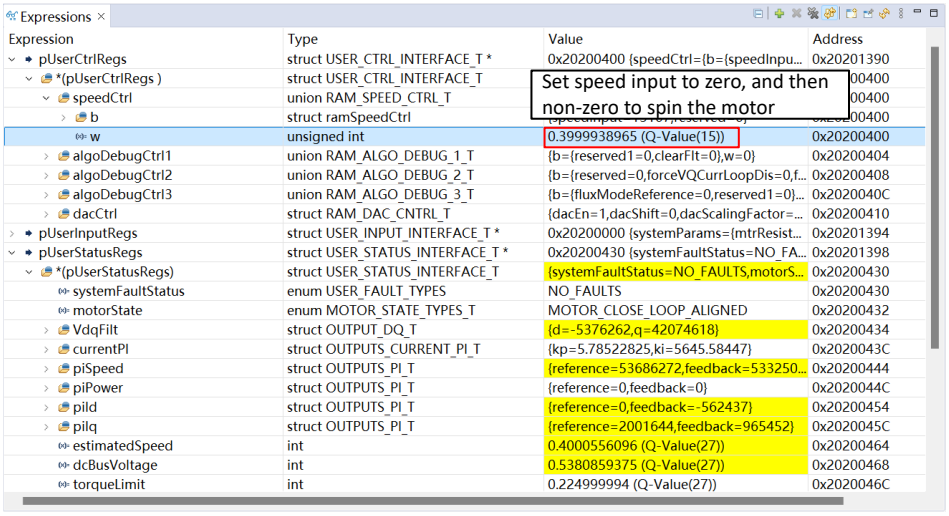


Figure 7-14. Spin The Motor After Hall Calibration Done

After Hall Calibration done, users should overwrite the Hall angle table in ISR.c file, as shown in Figure 7-14. Any changes to the hall signal wiring require re-execution of the complete calibration flow.

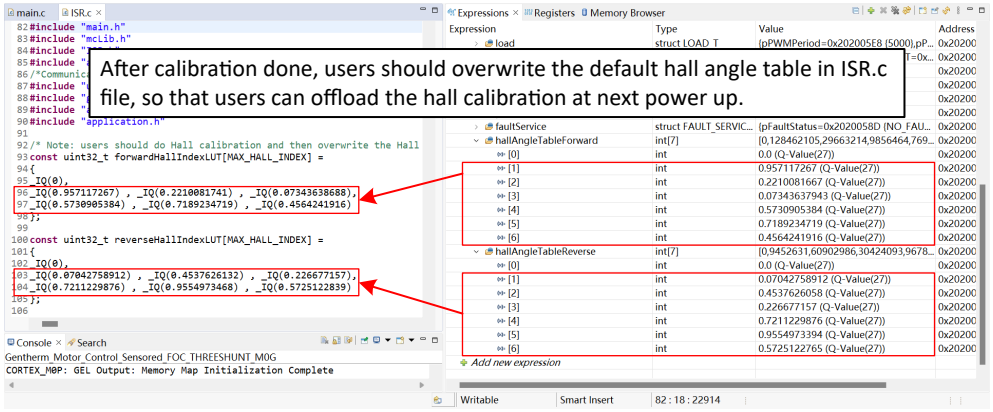


Figure 7-15. Overwrite Hall Angle Table

Hall calibration requires additional register modifications and takes time to complete. For hardcode to automatically implement the Hall calibration, refers to the code below:

```
/* Start Calibration when first connected to the motor */
__BKPT(0);    /* For debug */

pUserCtrlRegs->algoDebugCtrl2.b.hallCalibEnable = 0x1;

pUserCtrlRegs->algoDebugCtrl2.b.updateConfigs = 0x1;
while(pUserCtrlRegs->algoDebugCtrl2.b.updateConfigs){
    updateConfigs();    /* Polling until all register updated */
}

/* Start Calibration */
pUserCtrlRegs->speedCtrl.b.speedInput = 10000;
while (g_pMC_App->foc.hallCalibObj.calibState != HAL_CALIB_COMPLETE) {
    updateConfigs();
    /* Polling until calibration done */
}
/* Reset motor control state machine */
pUserCtrlRegs->speedCtrl.b.speedInput = 0;
pUserCtrlRegs->algoDebugCtrl2.b.hallCalibEnable = 0x0;

pUserCtrlRegs->algoDebugCtrl2.b.updateConfigs = 0x1;
while(pUserCtrlRegs->algoDebugCtrl2.b.updateConfigs){
    updateConfigs();
}

__BKPT(0);    /* For debug */
/* Calibration done */
```

7.7 Tune the Motor (LVBLDC)

The following figure shows the one phase current waveform when spinning the motor under the default tuning parameters. Users can separately set the configurations of each mode to implement specific application requirements.

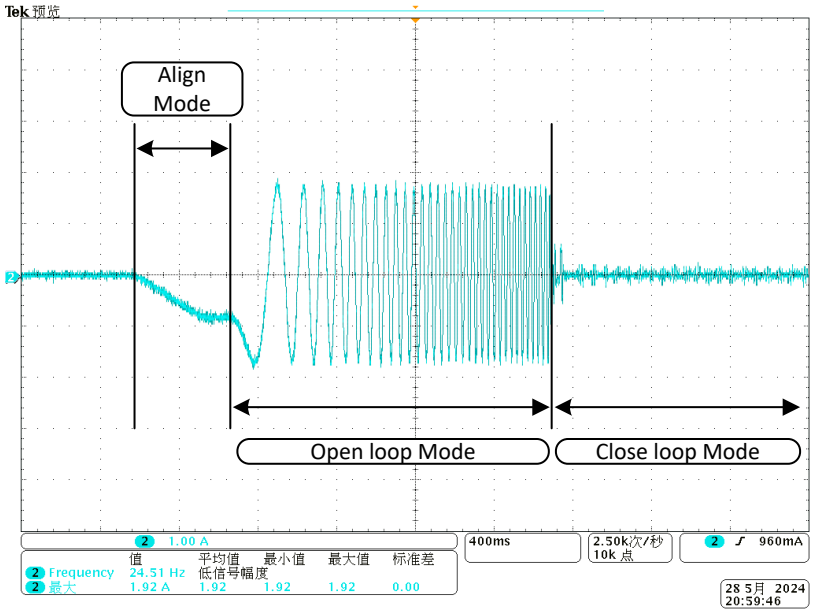


Figure 7-16. Motor Start-up Phase Current Waveform

Note

This section presents the tuning guide for Sensorless FOC. The same procedures also apply to Universal FOC and Hall Sensored FOC where these features are available.

7.7.1 Basic Tuning

The firmware has default settings to easy start. Users can overwrite the registers to tune the motor for application requirements. This section introduces basic tuning function.

7.7.1.1 Startup Mode

FOC algorithm supports different startup modes, including align mode, double align mode, IPD mode, and SFC mode. The firmware sets align mode as default startup mode. Users can set ***pUserInputRegs*** ->mtrStartUp1.b.mtrStartUpOption as the different value to select other startup modes.

The startup mode does not apply for Sensored FOC algorithm, as Sensored FOC algorithm directly get the motor rotor initial position from Hall sensor (in 60 degrees resolution).

7.7.1.1.1 Align Mode

Align method is a basic start method of sensorless FOC control algorithm. It is implemented to obtain an accurate initial position of the motor rotor. Set mtrStartUpOption as 0h to select align mode.

Follow the steps below to tune the align mode parameters:

1. Using pUserInputRegs->mtrStartUp2.b.alignAngle to set the motor rotor angle alignment
2. Using pUserInputRegs->mtrStartUp1.b.alignTime to set the duration of align mode
3. Using pUserInputRegs->mtrStartUp1.b.alignOrSlowCurrLimit to set maximum align current
4. Using pUserInputRegs->mtrStartUp1.b.alignSlowRampRate to set different current ramp up rate before reaching maximum align current

7.7.1.1.1.1 Force Align Mode in Current Loop

For debug, FOC algorithm provides the option that forces the motor control state machine to stay in align mode to tune the align mode parameter and verify the performance efficiently.

Set ***pUserCtrlRegs*** ->algoDebugCrtl1.b.forceAlignEn as 1b to enable force align mode. Set speedInput as non-zero value to spin the motor. The following figure shows the current waveform in force align mode.

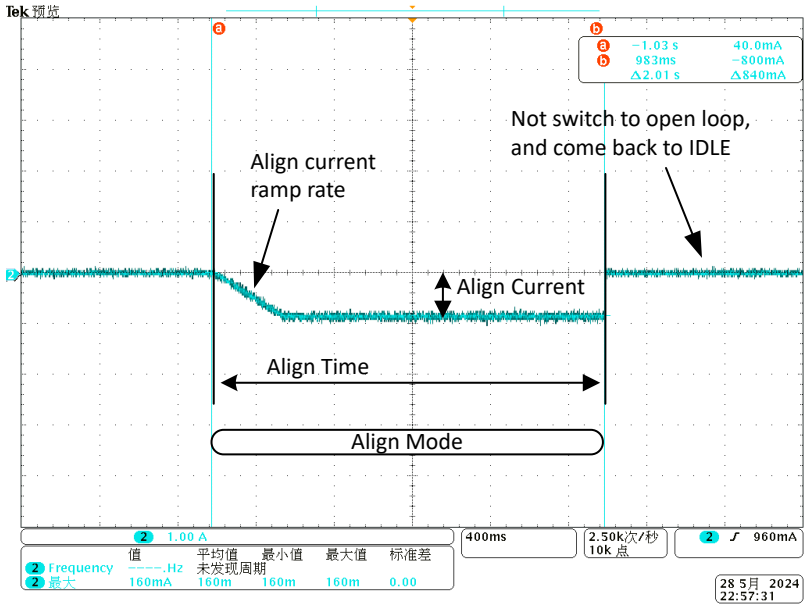


Figure 7-17. Force Align Mode in Current Loop

In the align mode, q-axis current align method is applied by default. The current closed loop is performed and the three phase currents will be one positive and two negatives.

7.7.1.1.1.2 Force Align Mode in PWM Loop

For debug, FOC algorithm provides the option that forces the motor control state machine to stay in align mode and disable the current closed loop.

When disable current closed loop, the FOC algorithm directly control the PWM duty cycle to three phase. Users can easily check the PWM functionality and validate the hardware circuit connection as no control loop is enabled at this time.

Set ***pUserCtrlRegs*** ->algoDebugCrtl2.b.currLoopDis to disable current closed loop. Then users can overwrite ***pUserCtrlRegs*** ->algoDebugCrtl2.b.forceVDCurrLoopDis and forceVQCurrLoopDis to set different SVPWM outputs, as shown in Figure 7-18.

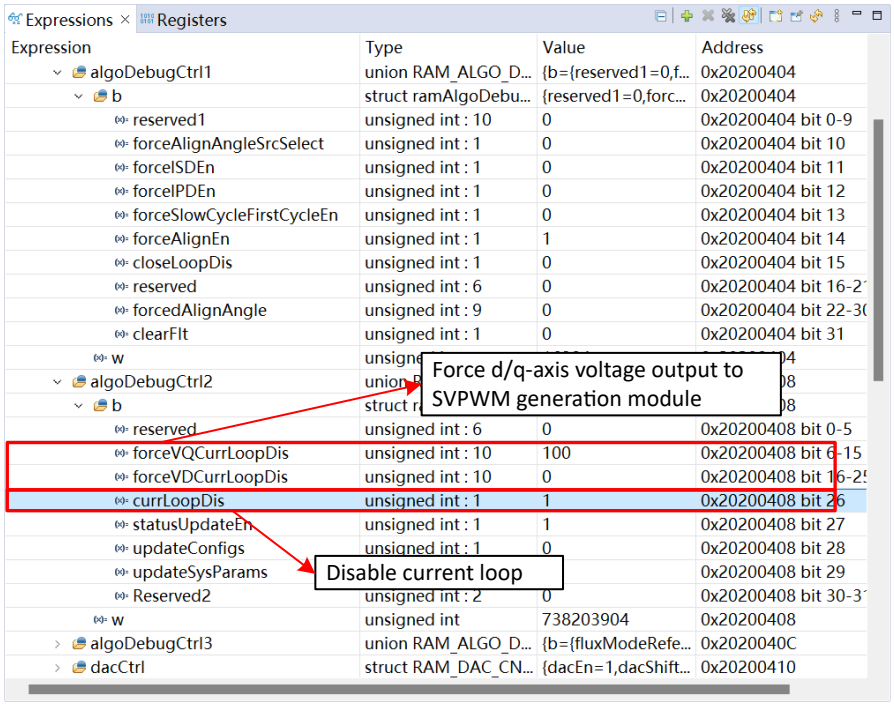


Figure 7-18. Register Setting in CCS Debug Window

Figure 7-19 shows the PWM output waveform in force align mode.

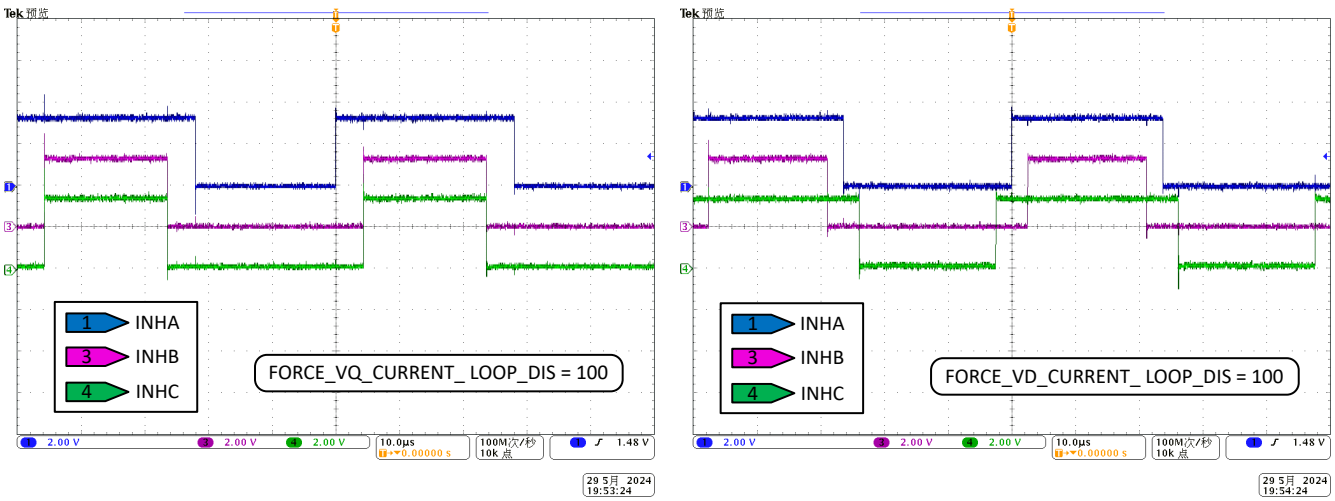


Figure 7-19. Force Align Mode in PWM Loop

Note

TI recommends disconnecting the motor when testing the PWM loop, because a small duty cycle may output a high current in motor phase. If connected motor, users should be careful to increase the PWM cycle from zero.

7.7.1.1.2 Double Align Mode

Double align method avoids the marginally stability state that occurs during motor alignment when the motor rotor position differs from the alignment angle by 180 electrical degrees.

FOC application does not support double align mode, setting mtrStartUpOption to 1h also represents align mode.

7.7.1.1.3 Initial Position Detection (IPD) Mode

Align or double align method may result in the motor spinning in the reverse direction before starting open loop acceleration. IPD can be used in such applications where reverse rotation of the motor is unacceptable. IPD does not wait for the motor to align with the commutation and therefore can allow for a faster motor start-up sequence. IPD works well when the inductance of the motor varies as a function of position. IPD works by pulsing current into the motor and hence can generate acoustics which must be considered when determining the best start-up method for a particular application.

IPD algorithm utilizes the ADC current sensing path to identify the phase current rise times to detect the rotor position. The window comparators of ADC continuously monitor the phase current against a pre-set IPD threshold current limit to create a current pulse. A Timer is used to capture rising times of these various pulses across different sectors and compared against detecting the rotor position.

The Timer and ADC configurations are updated during the IPD initialization based on the SysConfig configuration. The algorithm configures the required WCOMP settings based on the selected current sensing method (Section 4.2.3) and current sensing type (Section 7.1.2).

Note

During IPD pulse time, the rest of the algorithm interrupts are halted to continuously monitor the ADC current at very high sampling rates. The normal interrupt operations resume once the IPD operation is complete.

Set mtrStartUpOption as 2h to select IPD mode for startup. Follow the sequence below to configure the IPD parameters:

1. Select the IPD current threshold [pUserInputRegs->mtrStartUp1.b.ipdCurrThresh]. IPD current threshold is selected based on the inductance saturation point of the motor. A higher current has better chance to accurately detect the initial position. However, higher current can result in rotor movement, vibration, and noise. Start with 50% of the rated current of the motor. If the motor startup is unsuccessful, increase the threshold until the motor starts successfully. Do not set the IPD current threshold higher than the rated current of the motor.
2. Select IPD clock value [pUserInputRegs->mtrStartUp1.b.ipdClkFreq]. IPD clock defines how fast the IPD pulses are applied. Motors with higher time constants and higher current thresholds need longer time to decay the current before next IPD pulse is excited. So, set the clock at a slower time as starting value and increase till the IPD TIME OUT fault is not triggered. A slower clock makes the IPD noise softer but lasts longer, so set the clock such that IPD fault is not triggered and Noise is reasonably acceptable.

   Note

   The device triggers the IPD timeout fault IPD_FAULT_CLOCK_TIMEOUT (controllerRawFaultStatus) for motors with very high time constant, or if the motor is not connected. If this fault is triggered, make sure that the motor is connected to the device. If the fault still persists, decrease the IPD clock frequency (ipdClkFreq).
3. Select IPD Advance Angle [pUserInputRegs->mtrStartUp1.b.ipdAdvAngle]. Start with 90° for maximum startup torque. If there is sudden jerk observed during startup, reduce the angle to 60° or 30° for a smoother startup.
4. Select number of times IPD is executed [pUserInputRegs->mtrStartUp1.b.ipdRepeat]. Increased IPD execution times improve IPD accuracy but also increase time consumption. Users can start with 1 time.

Set ***pUserCtrlRegs*** ->algoDebugCrtl1.b.forceIPDEn as 1b to enable force IPD mode. It is useful for users to debug IPD performance and tune the parameters.

7.7.1.1.3.1 High Resolution IPD

FOC application supports High Resolution IPD functions by setting ***pUserInputRegs*** ->miscAlgo.b.ipdHiResolEn to 1b (only available in Sensorless FOC). The following table shows the difference between IPD and High Resolution IPD.

Table 7-12. ISD Feature Comparison

Algorithm	Description
IPD	Basic ISD function supports detecting the initial motor rotor angle with an accuracy of within 30 electrical degrees
High Resolution IPD	High Resolution IPD function supports detecting the initial motor rotor angle with an accuracy of within 10 electrical degrees

7.7.1.1.4 Slow First Cyle (SFC) Mode

Set mtrStartUpOption as 3h to select slow first cycle mode. In slow first cycle start-up, the FOC algorithm starts motor commutation at a frequency defined by ***pUserInputRegs*** ->mtrStartUp2.b.slowFirstCycFreq. The frequency configured is used only for first cycle, and then the motor commutation follows acceleration profile configured by open loop parameters, as shown in the following figure. The slow first cycle frequency has to be configured to be slow enough to allow motor to synchronize with the commutation sequence. This mode is useful when fast startup is desired as it significantly reduces the align time.

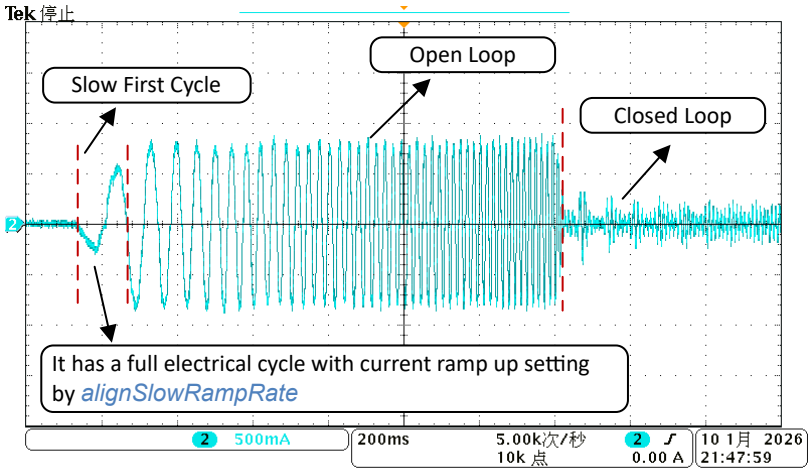


Figure 7-20. Phase Current in Slow First Cyle Mode

Set ***pUserCtrlRegs*** ->algoDebugCrtl1.b.forceSlowCycleFirstCycleEn to 1b to enable force slow first cycle mode for debug usage.

7.7.1.2 Open Loop Mode

Upon completing the motor position initialization with either align, double align, IPD or slow first cycle, the FOC algorithm begins to accelerate the motor in open loop. During open loop, the speed increases with a fixed current limit. In open loop, the control PI loops for Iq and Id actively control the currents. The angle during open loop is provided from the ramp generator.

Follow the steps below to tune the open loop parameters:

1. Set the pUserInputRegs->mtrStartUp2.b.olILimit for open loop current limit setting.

   Note

   Configuring the current limit to a value higher than motor stall current overheats or damages the motor.

2. Set the pUserInputRegs->mtrStartUp2.b.olAcc1 and pUserInputRegs->mtrStartUp2.b.olAcc2 for acceleration and Jerk parameters of the motor to determine the motor acceleration time (equals to open loop time), see Equation 40.

3. Set the pUserInputRegs->mtrStartUp2.b.olClHandOffThr for the switch speed from open loop mode to closed loop mode. Increase the current limit (olILimit) if the motor cannot reach the switch speed.

$$\text{Speed}(t) = \text{olAcc1} \times t + \text{olAcc2} \times t^2 \quad (40)$$

Users have to set a sufficient switch speed for the back-EMF observer to estimate the angle and speed of the motor. 15~25% of rated speed should work in majority of motors and get stable switch performance.

7.7.1.2.1 Auto Handoff

Users can enable auto handoff if the motor has sufficient BEMF for the observer by setting ***pUserInputRegs*** ->mtrStartUp2.b.autoHandOffEn as 1b.

The BEMF threshold (mV) for auto handoff can be configured by using ***pUserInputRegs*** ->miscAlgo.b.autoHandoffMinBemf.

7.7.1.2.2 Force Open Loop Mode

For debug, FOC algorithm provides the option that forces the motor control state machine to stay in open loop mode to tune the open loop mode parameter and verify the performance efficiently. In Force Open Loop Mode, users can directly observe the motor phase current response with tuning the current loop control parameters (currLoopKp and currLoopKi).

Setting ***pUserCtrlRegs*** ->algoDebugCrtl1.b.closeLoopDis as 1b to enable Force Open Loop Mode. The following figure shows the motor phase current waveform in Force Open Loop Mode.

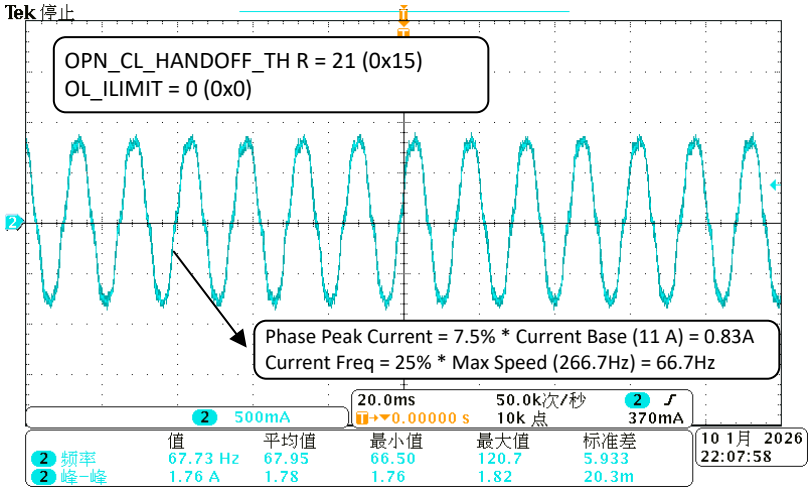


Figure 7-21. Phase Current in Force Open Loop Mode

7.7.1.3 Transition From Open Loop to Closed Loop

Once the motor has reached a sufficient speed for the back-EMF observer to estimate the angle and speed of the motor, the FOC algorithm transitions into closed loop mode.

To have smooth transition and avoid speed transients, the theta_error (open loop angle – estimation angle) is decreased linearly after transition. The ramp rate of theta_error reduction can be configured using ***pUserInputRegs*** ->mtrStartUp2.b.thetaErrRampRate.

If the current limit (olILimit) set during the open loop is high and if it is not reduced before transition to closed loop, the motor speed may momentarily rise to higher values than speed reference (speedInput) after transition into closed loop. To avoid such speed variations, configure the ***pUserInputRegs*** ->mtrStartUp2.b.iqRampEn to 1b, so that iq_ref decreases prior to transition into closed loop, as shown in Figure 7-22. However, if the speed reference (speedInput) is more than two times the open loop to closed loop hand off speed (olClHandOffThr), then iq_ref is not decreased independent of the iqRampEn setting, to enable faster motor acceleration.

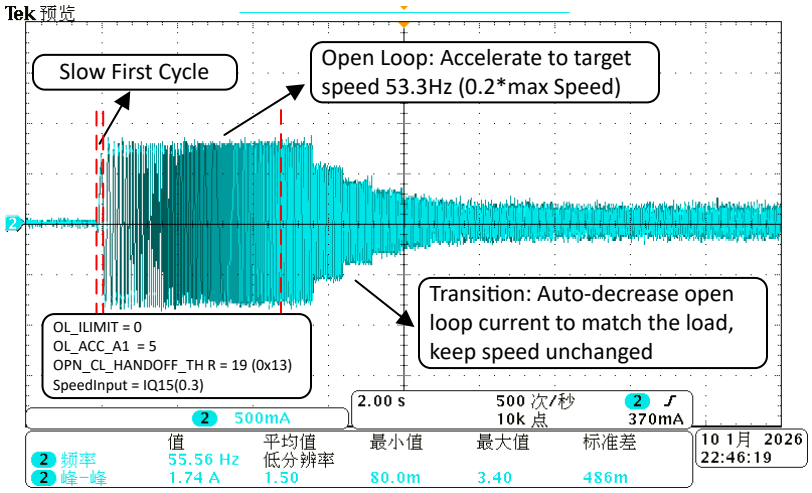
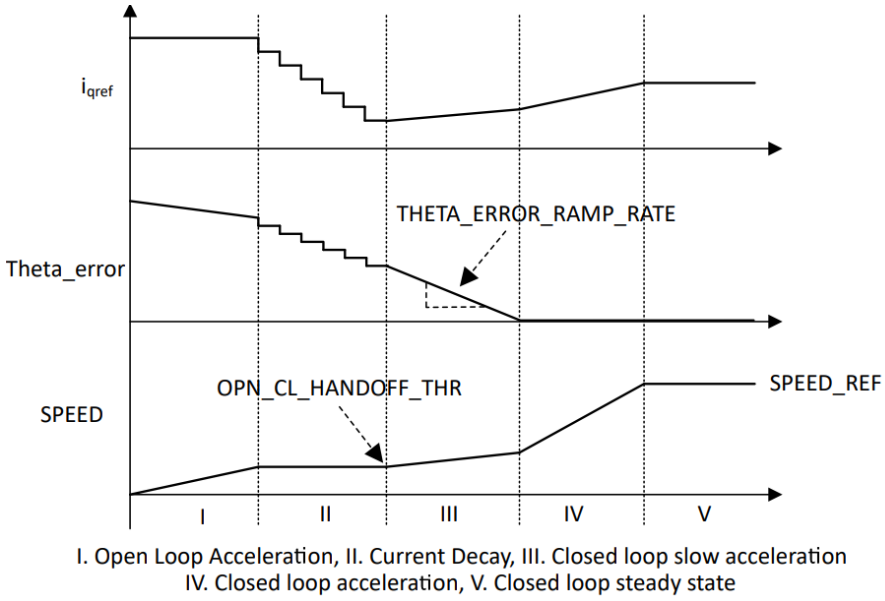


Figure 7-22. Phase Current with IqRamp Enabled

After handoff to closed loop at a sufficient speed, there could be still some theta error, as the estimators may not be fully aligned. A slow acceleration can be used after the open loop to closed loop transition, ensuring that the theta error reduces to zero. Slow acceleration can be configured using ***pUserInputRegs*** ->miscAlgo.b.clSlowAcc.

Figure 7-23 shows the control sequence in open to closed loop transition.



I. Open Loop Acceleration, II. Current Decay, III. Closed loop slow acceleration
IV. Closed loop acceleration, V. Closed loop steady state

Figure 7-23. Control Sequence in Open to Closed Loop Transition

7.7.1.4 Closed Loop Mode

In closed loop operation, the motor angle and electrical speed are estimated using the back-EMF observer. The speed and current regulation are achieved using PI control loop. To improve efficiency, the direct axis current is set to zero (Id_ref = 0), which will ensure that stator and rotor field are orthogonal (90 degrees out of phase) to each other.

To spin motor stably in closed loop mode, users should keep a sufficient speed for the back-EMF observer to estimate the angle and speed of the motor. Recommend keeping larger than 15% of the rated motor speed for closed loop operation.

7.7.1.4.1 Tune Control Parameter

The integrated speed (or power) control loop helps maintain a constant speed (or power) over varying operating conditions. The Kp and Ki coefficients are configured through speedLoopKp and speedLoopKi. The output of the speed loop is limited to implement a current limit. The current limit is set by configuring iLimit.

Follow the steps below to tune motors in closed loop:

1. Set the pUserInputRegs->closeLoop1.b.iLimit for closed loop maximum current limit setting. Unlike open loop, the q-axis current in closed loop depends on the motor load. Increase the current limit if motor speed is smaller than reference speed.
2. Set the pUserInputRegs->closeLoop1.b.clAcc and pUserInputRegs->closeLoop1.b.clDec for acceleration and deceleration slew rate parameters of the motor. This allows for a linear change in speed reference input even when there is a step change in speed reference (speedInput) and helps prevent sudden changes in the torque applied to the motor which could result in acoustic noise.

7.7.1.4.2 Tune PI Parameter

To tune the Kp and Ki values for speed loop:

1. Configure the motor to spin continuously in open loop by setting closeLoopDis to 1b.
2. Disable the automatic handoff by setting autoHandOffEn to 0b if enabled by user application code.
3. Set the closed loop hand off threshold to around 50% of maximum speed using olClHandOffThr.
4. Set the iqRampEn bit to 1b to enable iq_ref decreases prior to transition into closed loop
5. The current reference gradually decreases and settles down to the lowest possible Iqref to run at the given threshold speed.
6. Speed loop Kp (speedLoopKp) is calculated using Equation 41.

$$\text{Speed Loop Kp} = \text{Current Reference at olClHandOffThr (Amps)} \,/\, \text{olClHandOffThr (Hz)} \tag{41}$$

7. Speed loop Ki (speedLoopKi) is calculated using Equation 42.

$$\text{Speed Loop Ki} = \text{Speed Loop Kp} \times 0.1 \tag{42}$$

8. Enable closed loop by clearing the closedloopDis to 0b.

Note

The tuning of speed loop Kp and Ki is experimental. If the above recommendation does not work, manually tune speed loop Kp and Ki until the desired results are achieved.

7.7.1.5 Stop Mode

Set speedInput to 0 to generate a stop command. The FOC algorithm provides different options for stopping the motor which can be configured by ***pUserInputRegs*** ->closeLoop1.b.mtrStopOption.

7.7.1.5.1 Coast (Hi-Z) Mode

Coast (Hi-Z) mode is configured by setting mtrStopOption to 0h.

When motor stop command is received, the FOC application will transition into a high impedance (Hi-Z) state by turning off all MOSFETs. When the FOC application transitions from driving the motor into a Hi-Z state, the inductive current in the motor windings continues to flow and the energy returns to the power supply through the body diodes in the MOSFET output stage (see example in).

The following figure shows an example of how the Hi-Z mode works. In this example, current is applied to the motor through the high-side phase-A MOSFET (HSA), high-side phase-B MOSFET(HSB) and returned through the low-side phase-C MOSFET (LSC).

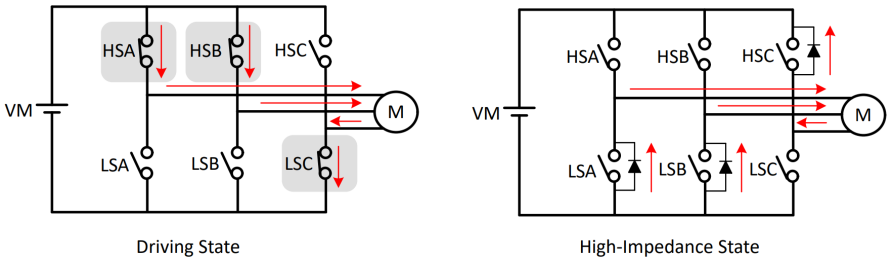


Figure 7-24. Coast (Hi-Z) Mode

In Coast mode, a reverse current is injected into the DC bus and might result in DC bus voltage overshoot.

7.7.1.5.2 Active Spin Down Mode

Active spin down mode is configured by setting mtrStopOption to 1h. When a motor stop command is received, the FOC application reduces motor speed reference to ***pUserInputRegs*** ->closeLoop2.b.actSpinThr and then transitions to Hi-Z state by turning all MOSFETs OFF.

The advantage of this mode is that by reducing motor speed reference, the motor is decelerated to lower speed thereby reducing the phase currents before entering Hi-Z. Now, when the motor transitions into Hi-Z state, the energy transfer to the power supply is reduced.

Active spin down can be used as a motor stop option in applications where fast stop is not required but some amount of inductive energy going back to power supply is acceptable. If there is voltage overshoot seen on the power supply, decrease the ACT_SPIN_THR till the voltage overshoot reaches acceptable limit. Note that threshold actSpinThr needs to configured high enough for FOC application to not lose synchronization in closed loop with the motor.

7.7.1.5.3 Braking Mode

Braking mode is configured by setting mtrStopOption to 2h. Follow the instructions below to set braking mode:

1. A software configuration is applied to enable braking mode by setting pUserInputRegs->pinCfg.b.brakeInput as 2h. FOC application enters the braking mode with a stop command in this setting.
2. Setting brakeInput as 1h to directly enter brake state without a stop command. FOC application stay in brake state till brakeInput is set to 2h. If the speedInput is non-zero value when brakeInput recover to 1h, the motor starts to spin immediately.
3. A hardware input pin (BRAKE) is configured to enable braking mode by setting brakeInput as 0h or 3h. As long as the BRAKE pin is driven HIGH externally, the motor stays in brake state.

Before entering brake state, FOC application decreases output speed to value defined by ***pUserInputRegs*** ->closeLoop2.b.brkSpeedThr. If the motor speed is below brkSpeedThr prior to receiving stop command, then the FOC application transitions directly into the brake state.

Brake state can be further configured to either low side braking (Low-Side Braking) or align brake (Align Braking) through ***pUserInputRegs*** ->pinCfg.b.brakePinMode.

7.7.1.5.3.1 Low-Side Braking

Set brakePinMode as 0b to select the low-side braking mode. The following figure shows the circuit state for low-side brake braking state.

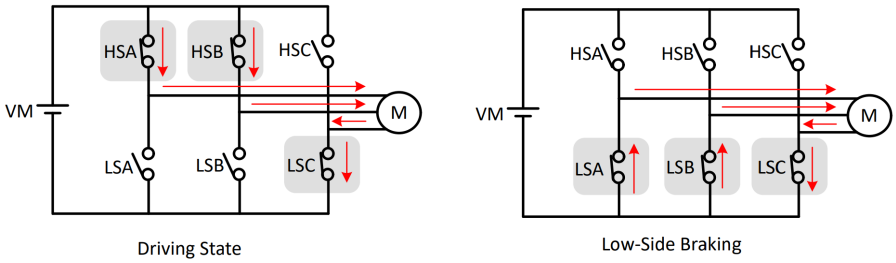


Figure 7-25. Low-Side Braking

7.7.1.5.3.2 Align Braking

Align braking mode is configured by setting brakePinMode to 1b. The FOC application can also enter align brake state through the BRAKE pin. In this mode, the FOC application aligns the motor by injecting a DC current through a particular phase pattern for a certain time configured by ***pUserInputRegs*** ->miscAlgo.b.brkCurrPersist.

The phase pattern during align is generated based on the angle at which align needs to be performed and the align angle can be configured through ***pUserInputRegs*** ->mtrStartUp2.b.alignAngle. The current limit threshold during align braking is configured through ***pUserInputRegs*** ->mtrStartUp1.b.alignOrSlowCurrLimit. The following figure shows the phase current waveform in align braking mode.

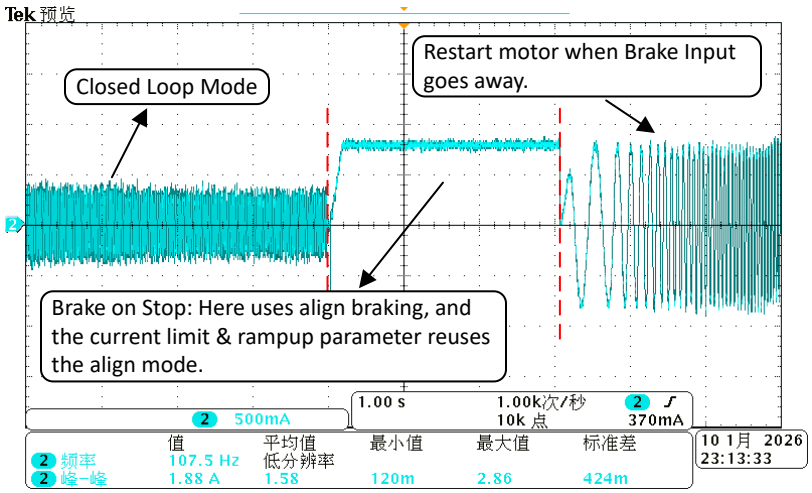


Figure 7-26. Align Braking

7.7.1.6 Fault Handling

The fault status is registered in ***pUserStatusRegs*** (see Section 5.3.3). The following table summarizes faults that can be triggered based on the fault configuration.

Table 7-13. Fault Report Summary

System Fault Status	FOC Fault Status	Description
NO_FAULTS	NO_FAULTS (0x00000000)	No fault occurred.
MOTOR_STALL	ABN_SPEED_FAULT_INDEX	The motor estimated speed is over the Abnormal Speed Lock threshold (% of max speed)
	ABN_BEMF_FAULT_INDEX	The estimated BEMF voltage drops below the programmed Abnormal BEMF Lock threshold (% of expected BEMF)
	NO_MOTOR_FAULT_INDEX	The motor phase current is below the No Motor Lock threshold % of base current.

Table 7-13. Fault Report Summary (continued)

System Fault Status	FOC Fault Status	Description
VOLTAGE_OUT_OF_BOUNDS	OVER_VOLTAGE_FAULT_INDEX	The DC bus voltage is over the programmed Maximum Voltage in % of base voltage.
	UNDER_VOLTAGE_FAULT_INDEX	The DC bus voltage drops below the programmed Minimum Voltage in % of base voltage.
LOAD_STALL	IPD_CLOCK_TIMEOUT_FAULT_INDEX	IPD timeout fault occurs. Time overflows before current ramp-up to IPD threshold current.
	IPD_DECAY_TIME_FAULT_INDEX	IPD decay time fault occurs. Complete decay of current not finished before next IPD pulse.
HARDWARE_OVER_CURRENT	BUS_CURRENT_LIMIT_INDEX	Not applied.
HV_DIE	HV_DIE_FAULT_INDEX	The fault input pin (PWM fault pin) is LOW. There is an external fault event.
NA	HW_LOCK_ILIMIT_FAULT_INDEX	Not applied.

7.7.1.6.1 MOTOR_STALL

MOTOR_STALL fault includes no motor fault, abnormal speed fault, and abnormal BEMF fault. Using ***pUserStatusRegs*** ->controllerFaultStatus to determine which fault is triggered under MOTOR_STALL fault.

FOC application provides various lock and retry methods when trigger MOTOR_STALL by using ***pUserInputRegs*** ->faultCfg1.b.mtrLckMode and ***pUserInputRegs*** ->faultCfg1.b.lockRetry. Refer to Section 5.3.2.7 for detailed instructions.

7.7.1.6.1.1 ABN_SPEED_FAULT

FOC application monitors the speed continuously and at any time the speed exceeds Abnormal Speed Lock threshold (% of max speed) defined in ***pUserInputRegs*** ->faultCfg2.b.lockAbnSpeed.

Setting ***pUserInputRegs*** ->faultCfg2.b.lock1En as 1b to enable abnormal speed protection.

7.7.1.6.1.2 ABN_BEMF_FAULT

Setting ***pUserInputRegs*** ->faultCfg2.b.lock2En as 1b to enable abnormal BEMF protection. This fault is triggered when the estimated BEMF voltage drops below the Abnormal BEMF threshold percentage defined in ***pUserInputRegs*** ->faultCfg2.b.abnBemfThr.

For example, if the estimated or measured Ke is 5 mV/Hz and the registered Abnormal BEMF threshold is 40%, this fault is triggered when the estimated Ke drops below 2 mV/Hz.

There are two typical cases which trigger the ABN_BEMF_FAULT:

1. The registered Ke is inaccurate. Refer to Section 7.2.5 to calculate or obtain accurate Ke.
2. Estimated BEMF voltage drops when the motor speed drops. Motor speed can drop due to load dynamics (sudden change in load). For applications with load dynamics, the speed is expected to drop and recover back. For such applications, the recommended value of 10% is to be set for the Abnormal BEMF threshold to avoid triggering this fault.

Note

Abnormal BEMF protection also monitors the result of transition stage from open loop mode to closed loop mode and reports ABN_BEMF_FAULT if transition fails.

7.7.1.6.1.3 NO_MOTOR_FAULT

Setting ***pUserInputRegs*** ->faultCfg2.b.lock3En as 1b to enable no motor protection. This fault is triggered when the phase current is below the No Motor Lock threshold % of base current defined in ***pUserInputRegs*** ->faultCfg2.b.noMtrThr.

Follow the steps below if NO_MOTOR_FAULT occurs:

1. Make sure the motor phases are well connected to the terminals.
2. If the fault persists, increase the No Motor Lock current threshold by using noMtrThr.

7.7.1.6.2 VOLTAGE_OUT_OF_BOUNDS

In applications where the power supply fluctuates, users need to specify the minimum and maximum power supply voltage range. During an undervoltage condition, the motor operates in overmodulation region to achieve the target speed leading to current distortion, inefficiency or noise. During an overvoltage condition, the MOSFETs and motor are stressed with continued operation in high voltage.

Using ***pUserInputRegs*** ->faultCfg2.b.minVmMtr and minVmMtr to set proper DC bus voltage range. And ***pUserStatusRegs*** ->controllerFaultStatus is used to determine that undervoltage or overvoltage is triggered.

FOC application provides an undervoltage recovery mode [***pUserInputRegs*** ->faultCfg2.b.minVmMode] and an overvoltage recovery mode [***pUserInputRegs*** ->faultCfg2.b.maxVmMode]. Undervoltage or overvoltage recovery mode can be configured to either automatically clear Undervoltage by setting minVmMode or minVmMode as 1b, or latch on the fault by setting minVmMode or minVmMode as 0b.

7.7.1.6.3 LOAD_STALL

FOC application uses a 16-bit timer running at 80MHz to estimate the time during the current ramp up and ramp down during IPD, when the motor start-up is configured as IPD (Section 7.7.1.1.3).

During IPD, the algorithm checks for a successful current ramp-up to ipdCurrThresh. If the IPD timer overflows (current does not reach ipdCurrThresh), then the IPD_CLOCK_TIMEOUT_FAULT gets triggered. If IPD_CLOCK_TIMEOUT_FAULT occurs, using ***pUserInputRegs*** ->miscAlgo.b.ipdMaxOverflow to set longer timeout period or using ***pUserInputRegs*** ->mtrStartUp1.b.ipdClkFreq to set lower IPD frequency.

Similarly, the algorithm checks for a successful current decay to zero during IPD current ramp down. IPD gives incorrect results if the next IPD pulse is commanded before the complete decay of current due to present IPD pulse. If the IPD timer overflows (current does not ramp down to zero), then the IPD_DECAY_TIME_FAULT gets triggered. Set lower ipdClkFreq to if IPD_DECAY_TIME_FAULT occurs.

7.7.1.6.4 HARDWARE_OVER_CURRENT

This feature is not applied in current FOC algorithm version.

7.7.1.6.5 HV_DIE

FOC application enables the PWM fault pin protection to allow the hardware to react (disable PWM) quickly to the external fault and leave the output signals in a safe state.

When the PWM fault input pin is LOW, the MCU hardware disable PWM output (set LOW) without software participation first, and then FOC applications trigger HV_DIE fault and process the protection logic in state machine.

7.7.1.7 Motor Spin Direction

The direction of the motor spinning is set by using ***pUserInputRegs*** ->periphCfg1.b.dirInput. FOC application provides two methods below to set the direction:

1. A software configuration is applied by setting dirInput as 1h (clockwise) or 2h (counterclockwise).
   - When dirInput is set as 1h, FOC application drives the clockwise rotation: OUTA-OUTB-OUTC
   - When dirInput is set as 2h, FOC application drives the counterclockwise rotation: OUTA-OUTC-OUTB
2. A hardware input pin (DIR) is configured to set the motor spinning direction by setting dirInput as 0h or 3h.
   - DIR input HIGH, FOC application drives the clockwise rotation: OUTA-OUTB-OUTC
   - DIR input LOW, FOC application drives the counterclockwise rotation: OUTA-OUTC-OUTB

Using ***pUserInputRegs*** ->periphCfg1.b.dirChangeMode to determine the FOC application response to the change of motor spinning direction:

- When dirChangeMode is set as 0b, FOC application follows motor stop options and ISD routine on motor direction change.

- When dirChangeMode is set as 1b, FOC application changes the direction through Reverse Drive (Section 7.7.2.7.2) while continuously driving the motor.

7.7.1.8 PWM Configuration

7.7.1.8.1 PWM Frequency

Using ***pUserInputRegs*** ->closeLoop1.b.pwmFreqOut to set different PWM frequency for motor control. By default, the firmware setting the PWM frequency as 16kHz.

7.7.1.8.2 PWM Deadband Time

The hardware circuit for FOC application required protection against any cross conduction of the MOSFETs. The high-side and low-side MOSFETs are carefully controlled to avoid any shoot-through events by inserting a deadband time.

Using ***pUserInputRegs*** ->periphCfg1.b.mcuDeadTimeto set different PWM deadband times.

Set Larger Deadband Time

Currently there is provision to set the deadband time as a 6-bit configurable value in ***pUserInputRegs*** ->periphCfg1 register. So, the register provides a Max of 3.2uS. In cases where the larger deadband time is needed, users can update this register to increase the size of MCU_DEAD_TIME to higher number of bits in ***appInputCtrlInterface.h*** file to enable larger deadband time.

Figure 7-27 shows an example to configure deadband time to 8-bit.

```
458 /*! @brief userInputPeriCfg1 structure */
459 typedef struct
460 {
461     uint32_t
462     /*! Response to change of DIR pin status */
463     dirChangeMode:              1,
464     /*! DIR pin override */
465     dirInput:                   2,
466     /*! Bus current limit enable */
467     busCurrLimitEnable:         1,
468     /*! Bus current limit */
469     busCurrLimit:               5,
470     /*! deadtime for PWM outputs */
471     mcuDeadTime:                8,
472     /*! Reserved */
473     reserved:                   15;
474 }userInputPeriCfg1;
```

Figure 7-27. Set the Deadband Time Register to 8-bit

7.7.1.9 FOC Loop Frequency

The FOC algorithm is periodically executed in the interrupt routine to update the rotor angle to extract the maximum efficiency from the motor. This FOC rate can be configured by user based on the application bandwidth requirements.

Setting ***pUserInputRegs*** ->closeLoop1.b.highFreqFOCEn to 0b to get the maximum FOC execution rate of 16kHz. Setting highFreqFOCEn to 1b reduces the maximum FOC execution rate by 2.

Note

FOC routine can only be executed at a multiple of PWM frequency, Hence, the maximum achievable FOC rate of 16kHz is applicable for PWM frequencies with multiple of 16 (for example, 16kHz, 32kHz, 48kHz). For PWM frequencies of 20KHz, 40kHz, and so on, the maximum FOC rate is limited to 10kHz (20KHz/2, 40KHz/4, and so on).

7.7.1.10 Hardcode for Basic Tuning

The following figure shows the hardcode to tune LVBLDC Motor.

```
    focPeriphInit();  /* Does foc application specific Peripheral configurations */

    /* Configure the Motor Params */
#ifdef  DT42040
    /* DT42040 */
    pUserInputRegs->systemParams.mtrResist        = 750;
    pUserInputRegs->systemParams.mtrInductance    = 1050;
    pUserInputRegs->systemParams.mtrBemfConst     = 367;
    pUserInputRegs->systemParams.maxMotorSpeed = 266.7;
#elif defined (DT4260)
    /* DT4260 */
    pUserInputRegs->systemParams.mtrResist        = 400;
    pUserInputRegs->systemParams.mtrInductance    = 600;
    pUserInputRegs->systemParams.mtrBemfConst     = 357;
    pUserInputRegs->systemParams.maxMotorSpeed = 266.7;
#endif

    /* When set as 0, the algorithm will automatically calculate current loop parameters */
    pUserInputRegs->systemParams.currLoopKp     = 0;
    pUserInputRegs->systemParams.currLoopKi     = 0;
    /* Configure the Current Loop Params - copy from auto calculated */
//    pUserInputRegs->systemParams.currLoopKp     = 5.9847188;
//    pUserInputRegs->systemParams.currLoopKi     = 3989.8125;

    /* StartUp Parameter */
    pUserInputRegs->mtrStartUp1.b.alignOrSlowCurrLimit = 0x0;   //7.5%*base_current
    pUserInputRegs->mtrStartUp1.b.alignTime = 0x6;  /*500ms*/
    pUserInputRegs->mtrStartUp2.b.olILimit = 0x0;       //7.5%*base_current
    pUserInputRegs->mtrStartUp2.b.olAcc1 = 0x5;         // 10Hz/s
    pUserInputRegs->mtrStartUp2.b.olClHandOffThr = 0x15;    //25%*base_speed

    /* Close Loop Parameter */
    pUserInputRegs->closeLoop1.b.controlMode = 0x0; // Closed Loop Speed Control
    pUserInputRegs->closeLoop1.b.iLimit = 0xE;      // 20%*base_current limitation
    pUserInputRegs->closeLoop1.b.clAcc  = 0x5;      // 10Hz/s
    pUserInputRegs->closeLoop1.b.clDec  = 0x5;      // 10Hz/s
//    pUserInputRegs->closeLoop1.b.pwmFreqOut = 0x3;  // Default is 16kHz

    /* Fault Handling */
    pUserInputRegs->faultCfg2.b.lockAbnSpeed    = 0;    // 130% max speed lock
    pUserInputRegs->faultCfg2.b.lock1En         = 0;    // Not enabled
    pUserInputRegs->faultCfg2.b.noMtrThr        = 0;    // 7.5% no motor threshold
    pUserInputRegs->faultCfg2.b.lock3En         = 0;    // Not enabled
    pUserInputRegs->faultCfg2.b.abnBemfThr      = 2;    // 50% Ke threshold
    pUserInputRegs->faultCfg2.b.lock2En         = 0;    // Not enabled

    pUserInputRegs->faultCfg2.b.maxVmMtr        = 0x7;  // No max Vm limitation
    pUserInputRegs->faultCfg2.b.minVmMtr        = 0x0;  // No min Vm limitation

    pUserInputRegs->periphCfg1.b.busCurrLimit   = 0xE;  // 20%*base_current limitation
    pUserInputRegs->periphCfg1.b.busCurrLimitEnable = 0x0;  // Not enabled

    pUserInputRegs->faultCfg1.b.mtrLckMode      = 0;    // Latch up the fault

    while (1)
    {
        if(gdReadTestEn)
        {
            regData =  gateDriverRegRead(regAddr);
        }
        UART_checkForCommand(pUART);

        updateConfigs();
    }

}
```

Figure 7-28. Hardcode for Basic Tuning

7.7.2 Advanced Tuning

This section helps users tune the advanced features of FOC application.

7.7.2.1 Control Mode Setting

FOC application can be controlled in below four modes using the variable ***pUserInputRegs*** ->closeLoop1.b.controlMode. The reference input for Speed/Power/Torque/Voltage is configured through the speedInput register.

7.7.2.1.1 Closed Loop Speed Control Mode

Speed closed loop control is a widely used control method. In speed control mode, the speed of the motor (Electrical speed, by Hz) is controlled using a closed loop PI control according to the input reference set as speedInput value in Speed Control Register (P.U value in IQ15 format). The P.U speed is computed as the ACTUAL_MOTOR_SPEED / MAX_MOTOR_SPEED value configured in systemParams.

Example: For maxMotorSpeed set to 100Hz, setting reference Input in speedInput to 0x3FFFh (0.5 P.U in IQ15) sets the motor real electrical speed to 50Hz.

By default, FOC algorithm uses speed closed loop control with controlMode set as 0h.

7.7.2.1.2 Closed Loop Power Control Mode

In power control mode, the input **electrical** power of the motor (Watts) is controlled using a closed loop PI control according to the input reference set as speedInput value in Speed Control Register (P.U value in IQ15 format). The P.U power is computed as the ACTUAL_MOTOR_POWER / MAX_MOTOR_POWER value configured in systemParams.

Example: For maxMotorPower set to 100Watts, setting reference Input in speedInput to 0x3FFFh (0.5 P.U in IQ15) operates the Motor at a constant **electrical** power of 50W.

FOC algorithm uses power closed loop control by setting controlMode as 1h.

7.7.2.1.3 Closed Loop Torque Control Mode

In torque (current) control mode, the torque component current Iq of the motor (in Amps) is controlled using a closed loop PI control according to the input reference set as speedInput value in Speed Control Register (P.U value in IQ15 format). The P.U torque component of q-axis current is computed as the TORQUE_CURRENT / CURRENT_BASE value configured in systemParams.

Example: For currentBase set to 10 Amps, setting reference Input in speedInput to 0x3FFFh (0.5 P.U in IQ15) operates the Motor at a constant q-axis current of 5A.

Note

When motor operates in torque mode, an appropriate load must be connected to the motor.

FOC algorithm uses torque closed loop control by setting controlMode as 2h.

7.7.2.1.4 Open Loop Voltage Control Mode

In voltage control mode, the speed and motor d-q axis current is not under controlled. FOC algorithm directly output the d-q axis voltage to SVPWM module. The modulation index of the motor is controlled according to the input reference set as speedInput value in Speed Control Register (P.U value in IQ15 format).

Example: Setting reference Input in speedInput to 0x3FFFh (0.5 P.U in IQ15) operates the motor with a constant modulation Index of 0.5.

FOC algorithm uses voltage open loop control by setting controlMode as 3h.

7.7.2.1.4.1 Lead Angle Control

In voltage control mode, Lead Angle can be adjusted to derive the best efficiency from the motor for a given speed. Using ***pUserInputRegs*** ->closeLoop2.b.leadAngle to set Lead Angle.

For a given Lead Angle (θ), Applied Voltages Vq and Vd are defined as:

$$Vq = MODULATION\_INDEX \times Cos\theta \quad (43)$$

$$Vq = MODULATION\_INDEX \times Sin\theta \quad (44)$$

7.7.2.2 Maximum Torque Per Ampere (MTPA) Control

MTPA (Maximum Torque Per Ampere) control within Field-Oriented Control (FOC) is an advanced strategy for salient pole motors. MTPA optimizes current injections to produce the most torque with the least input current to maximize motor control efficiency. Users can configure the saliency of the motor as non-zero value as detailed in saliency parameter description (Section 7.2.3).

This feature can be enabled by setting ***pUserInputRegs*** ->fieldCtrl.b.mtpaEnable to 1b.

7.7.2.3 Field Weakening Control (FWC)

Field weakening control expands the motor's speed range by reducing the intensity of the motor's excitation magnetic field. Especially after exceeding the rated speed, by injecting a current component opposite to the motor d-axis magnetic field, the total d-axis magnetic field is weakened, allowing the motor to continue accelerating under voltage limitations.

Note

During flux weakening operation, the motor cannot deliver the rated torque. The torque limit Iq is automatically adjusted based on the circular motor current limit defined by ILIMIT = SQRT(Id^2 + Iq^2).

This feature can be enabled by setting ***pUserInputRegs*** ->fieldCtrl.b.fluxWeakeningEn to 1b.

Follow the steps below to tune the field weakening control method:

1. Setting pUserInputRegs->fieldCtrl.b.fluxWeakeningEn to 1b to enable flux weakening
2. Using pUserInputRegs->fieldCtrl.b.fluxWeakCurrRatio to determine the maximum weakening d-axis current. Setting a larger weakening d-axis current has the risk of demagnetization in motor permanent magnets.
3. Using pUserInputRegs->fieldCtrl.b.fluxWeakeningReference to determine the maximum modulation index beyond which the field weakening is enabled. Setting a smaller speed threshold results in lower efficiency.

After field weakening control enabled, a separate PI controller applies for Id reference setting. Using ***pUserInputRegs*** ->systemParams.fluxWeakeningKp and ***pUserInputRegs*** ->systemParams.fluxWeakeningKi to tune the field weakening control.

7.7.2.4 Deadtime Compensation

Dead time is inserted between the switching instants of high-side and low-side MOSFET in a half bridge leg to avoid shoot-through condition.

Due to dead time insertion, the expected voltage and applied voltage at the phase node differ based on the phase current direction. The phase node voltage distortion introduces undesired distortion in the phase current causing audible noise. The distortion in current waveform due to dead time appear as sixth harmonic of fundamental frequency in the d-q axis reference frame.

The FOC algorithm integrates a proprietary dead time compensation using a resonant controller to control the sixth harmonic component in phase current to zero, ensuring that the current distortion due to dead time is alleviated. The resonant controller is employed in both iq and id control paths. The dead time compensation can be enabled or disabled by configuring ***pUserInputRegs*** ->closeLoop1.b.deadTimeCompEn.

7.7.2.5 PWM Generation Mode

The FOC algorithm supports two different modulation schemes, namely, continuous and discontinuous space vector PWM modulation schemes.

In continuous PWM modulation, all the three phases switch all the time as per the defined switching frequency. In discontinuous PWM modulation, one of the phases is clamped to ground for 120° electrical period, and the other two phases are pulse width modulated. The modulation scheme is configured using ***pUserInputRegs*** ->closeLoop1.b.pwmMode.

The following figure shows the modulated average phase voltages for different modulation schemes.

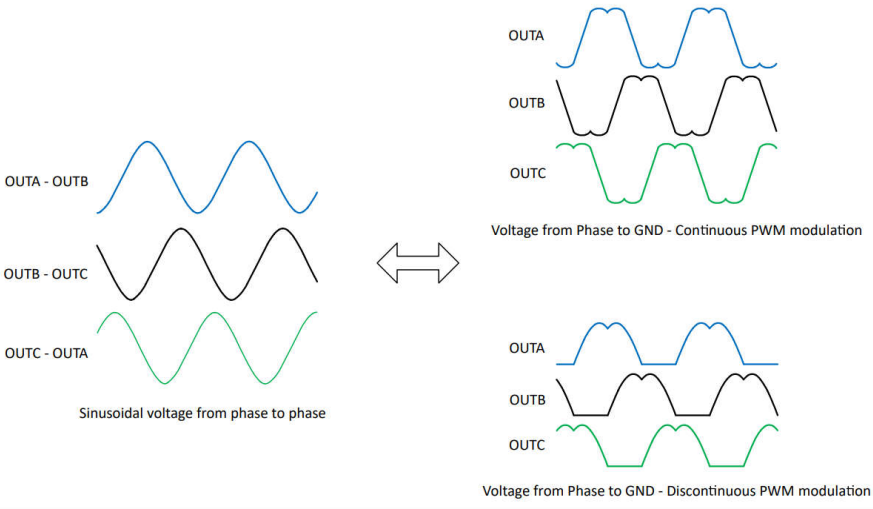


Figure 7-29. Continuous and Discontinuous PWM Modulation Phase Voltages

Continuous modulation helps in reducing current ripple for motors having low inductance, but it results in higher switching losses because all three phases are switching. Discontinuous modulation has lower switching losses due to only two phases switching at a time, but higher current ripple.

7.7.2.6 Overmodulation Mode

FOC algorithm provides an overmodulation option to operate the motor at a higher speed at the same VM voltage by increasing the applied fundamental phase voltage by suitably modifying the applied PWM pattern - the higher fundamental phase voltage is accompanied by an increase in higher order harmonics.

This feature can be enabled by setting ***pUserInputRegs*** ->closeLoop1.b.overModEnable to 1b.

7.7.2.7 Initial Speed Detection (ISD) Mode

The ISD function is used to identify the initial condition of the motor and is enabled by setting ***pUserInputRegs*** ->isdCfg.b.isdEn to 1b. The initial speed, position and direction is determined by sampling the phase voltage through the MSPM0 ADC. This function is useful for fan applications.

If the function is disabled, the FOC algorithm does not perform the initial speed detect function and proceeds to check if the brake routine [***pUserInputRegs*** ->isdCfg.b.brakeEn] is enabled.

7.7.2.7.1 Motor Resynchronization

The motor resynchronization function works when the ISD and resynchronization functions are both enabled and the device determines that the initial state of the motor is spinning in the forward direction (same direction as the commanded direction). The speed and position information measured during ISD are used to initialize the drive state, which can transition directly into closed loop (or open loop if motor speed is not sufficient for closed loop operation) state without needing to stop the motor.

Setting ***pUserInputRegs*** ->isdCfg.b.resyncEn as 1b to enable motor resynchronization. If motor resynchronization is disabled, the firmware proceeds to check if the motor coast (Hi-Z) routine [***pUserInputRegs*** ->isdCfg.b.hiZEn] is enabled.

7.7.2.7.2 Reverse Drive

The FOC algorithm uses the reverse drive function to change the direction of the motor rotation when ***pUserInputRegs*** ->isdCfg.b.rvsDrEn and isdEn are both set to 1b and the ISD determines the motor spin direction to be opposite to that of the commanded direction. This function is only valid in **Sensorless FOC**.

Reverse drive includes synchronizing with the motor speed in the reverse direction, reverse decelerating the motor through zero speed, changing direction, and accelerating in open loop in forward (or commanded) direction until the device transitions into closed loop in forward direction.

FOC algorithm provides the option of using the forward direction parameters or a separate set of reverse drive parameters by configuring ***pUserInputRegs*** ->rvsDrvCfg.b.revDrvConfig.

Follow these recommendations if the motor fails to resynchronize in reverse direction:

1. Increase the reverse deceleration speed threshold to transition to open loop [pUserInputRegs->isdCfg.b.revDrvHandoffThr]
2. Enable Open loop reverse drive configuration (revDrvConfig)
3. Increase the Reverse Drive Open Loop Current Reference [pUserInputRegs->isdCfg.b.revDrvOpenLoopCurr]
4. Decrease open loop acceleration coefficient A1 and A2 during reverse drive [pUserInputRegs->rvsDrvCfg.b.revDrvOpenLoopAccelA1 and revDrvOpenLoopAccelA2]

The following figure shows the motor speed curve under the reverse drive transition.

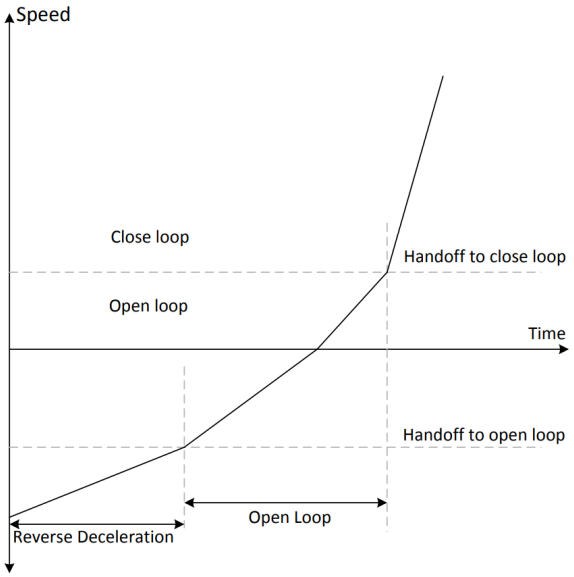


Figure 7-30. Motor Speed Curve Under the Reverse Drive Transition

7.7.2.7.3 Fast ISD

FOC application supports Fast ISD functions by setting ***pUserInputRegs*** ->miscAlgo.b.fastIsdEnableto 1b (only available in **Sensorless FOC**). The following table shows the difference between ISD and Fast ISD.

Table 7-14. ISD Feature Comparison

Algorithm	Description
ISD	Basic ISD function supports resync the motor position within 1 electrical cycle.
Fast ISD	Fast ISD function supports quick resync the motor position within 60 electrical degrees.

7.7.2.8 Anti-Voltage Surge

When a motor is driven, energy is transferred from the power supply into the motor. Some of this energy is stored in the form of inductive and mechanical energy. If the speed command suddenly drops such that the BEMF voltage generated by the motor is greater than the voltage that is applied to the motor, then the mechanical energy of the motor is returned to the power supply and the VM voltage surges.

The AVS feature works to prevent this voltage surge on VM and can be enabled by ***pUserInputRegs*** ->closeLoop1.b.avsEn. When AVS is disabled, the motor speed deceleration rate is configured through ***pUserInputRegs*** ->closeLoop1.b.clDec.

7.8 Overwrite User Input Register Table

In FOC application, users implement functions based on the register configuration, and the values of register represent different physical meanings, as shown in Register Macro (Section 5.3.2). FOC application provides partial register maps in a format of const array in the " ***configTables.c*** " file, shown in Figure 7-31. Users can overwrite the values in array to set their customization register table for FOC application.

```
#include "configTables.h"

/* ISD Config Tables*/
/*! @brief Table for forward and reverse drive */
const uint16_t tbl_fwRevDrv_pMil[16]    = {50,100,150,200,250,300,350,400,450,500,550,600,700,800,900,1000};

/*! @brief Table for HiZ brake time */
const uint16_t tbl_hiZ_brk_Time_ms[16] = {10,50,100,200,300,400,500,750,1000,2000,3000,4000,5000,7500,10000,15000};

/*! @brief Table for Stall detection threshold */
const _iq tbl_StatDetectThr_pu[8] = {_IQ(0.00104),_IQ(0.003125),_IQ(0.0041),_IQ(0.0104),_IQ(0.0208),_IQ(0.03125),_IQ(0.0416),_IQ(0.0625)};

/*Closed Loop PWM Frequency Table */
const uint16_t tbl1_clPWMFreqKHz[16] = {5,10,16,20,25,32,40,48,50,64,80,80,80,80,80,80};

/* Open Loop Speed and Acceleration Slew Rates */
/*! @brief Table 1 for open loop acceleration */
const uint16_t tbl1_olAccA1A2_centiHzPerSec[8] = {1, 5, 100, 250, 500, 1000, 2500, 5000};
/*! @brief Table 2 for open loop acceleration */
const uint16_t tbl2_olAccA1A2_HzPerSec[8] = {75, 100, 250, 500, 750, 1000, 5000, 10000};

/* Close Loop Speed and Acceleration Slew Rates */
/*! @brief Table 1 for close loop acceleration and deceleration */
const uint16_t tbl1_clDecClAcc_deciHzPerSec[16] = {5,10,25,50,75,100,200,400,600,800,1000,2000,3000,4000,5000,6000};
/*! @brief Table 2 for close loop acceleration and deceleration */
const uint16_t tbl2_clDecClAcc_HzPerSec[14] = {700,800,900,1000,2000,4000,6000,8000,10000,20000,30000,40000,50000,60000};

/* Close Loop 2 */
/*! @brief Table for brake speed threshold and active spin threshold */
const uint16_t tbl_brkDutyActSPinThr_pMil[16] = {1000,900,800,700,600,500,450,400,350,300,250,200,150,100,50,25};

/* Motor StartUp1 */
/*! @brief Table for ipd clock frequency */
const uint16_t tbl_ipdClkFreq_Hz[8] = {50,100,250,500,1000,2000,5000,10000};

/*! @brief Table for align time */
const uint16_t tbl_alignTime_msec[16] ={10,50,100,200,300,400,500,750,1000,1500,2000,3000,\
                                        4000,5000,7500,10000};

/*! @brief Table for align and slow cycle start ramp rate */
const uint16_t tbl_alignSlowRampRate[16] = {1,10,50,100,150,250,500,1000,1500,2000,2500,5000,10000,20000,50000,0};

/* Motor Start Up2 */
/*! @brief Table for theta error ramp rate */
const uint16_t tbl_thetaErrRampRate_mili[8]  = {10,50,100,150,200,500,1000,2000};

/*! @brief Table for align angle */
const uint16_t tbl_alignAngle[32] = {0,10,20,30,45,60,70,80,90,110,120,135,150,160,170,180,190, \
                                     210,225,240,250,260,270,280,290,315,330,340,350,350,350,350};

/*! @brief Table for slow first cycle frequency */
const uint16_t tbl_slowFirstCycFreqPerMil[16]  = {10,20,30,50,75,100,125,150,175,200,250,300,350,400,450,500};

/* Fault Config 1 and Config2 */
/*! @brief Table for abnormal speed lock */
const uint16_t tbl_lckAbnormalSpeed_pMil[8] = {1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000};

/*! @brief Table for abnormal backemf threshold */
const uint16_t tbl_AbnormalBEMFThr_pMil[8] = {10,20,30,50,80,100,120,150};

/* Internal algo 1 Params*/

/*! @brief Table for isd stop time and isd run time */
const uint16_t tbl_isdStopTime_msec[4]     = {1,5,50,100};

/*! @brief Table for isd timeout */
const uint16_t tbl_isdRunTime_msec[4]     = {100,250,500,1000};

/*! @brief Table for autohandoff minimum backemf */
const uint16_t tbl_autoHandOffMinBemf_mV[8]   = {0,50,100,250,500,1000,1250,1500};

/*! @brief Table for persistent brake current */
const uint16_t tbl_brakeCurrPersist_msec[4]   = {50,100,250,500};

/* Internal algo 2 Params*/
/*! @brief Table for close loop alow acceleration and deceleration */
const uint16_t tbl_clSlowAcc_deci[16]     = {1,10,20,30,50,100,200,300,400,500,1000,2000,5000,7500,10000,20000};

/*! @brief Table for current */
const _iq tbl_pu[32] = {_IQ(0.075),_IQ(0.080),_IQ(0.085),_IQ(0.090),_IQ(0.095),_IQ(0.100),_IQ(0.110),_IQ(0.120),   \
                        _IQ(0.130),_IQ(0.140),_IQ(0.150),_IQ(0.160),_IQ(0.170),_IQ(0.180),_IQ(0.200),_IQ(0.225),   \
                        _IQ(0.250),_IQ(0.275),_IQ(0.300),_IQ(0.350),_IQ(0.400),_IQ(0.450),_IQ(0.500),_IQ(0.550),   \
                        _IQ(0.600),_IQ(0.700),_IQ(0.750),_IQ(0.800),_IQ(0.850),_IQ(0.900),_IQ(0.950),_IQ(1.000)};

/*! @brief Table for minimum VM */
const _iq tbl_minVm_pMil[8] = {_IQ(0.0),_IQ(0.05),_IQ(0.10),_IQ(0.12),_IQ(0.15),_IQ(0.18),_IQ(0.20),_IQ(0.25)};

/*! @brief Table for maximum VM */
const _iq tbl_maxVm_pMil[8] = {_IQ(0.60),_IQ(0.65),_IQ(0.70),_IQ(0.75),_IQ(0.80),_IQ(0.85),_IQ(0.90),_IQ(1.0)};

/*! @brief Table for Modulation Limit Beyond which Flux weakening is enabled */
const _iq tbl_mSqrRef[4] = {_IQ(0.49), _IQ(0.64),_IQ(0.81), _IQ(0.902)};

/* Misc algo 1 Params*/
/*! @brief Table for IPD maximum overflow */
const uint8_t tbl_ipdMaxOverflow[4] = {5, 10, 20, 40};
```

Figure 7-31. Config Tables File Overview

For example, if users want to set align time to 150ms for application requirements, while the default register map only provides the options for 100ms or 200ms. Then users can position the tbl_alignTime_msec in ***configTables.c*** and overwrite the tbl_alignTime_msec[3] from 200 to 150. Such that the align time is now 150ms with setting ***pUserInputRegs*** ->mtrStartUp1.b.alignTime to 3h.

8 Hardware Migration Guide

This section helps users to migrate the SDK example project to their own customer board.

8.1 Hardware Layer Overview

The following figure shows the hardware configuration relationships for FOC project. The hardware layer contains three parts: Board layer, SysConfig layer and HAL layer.

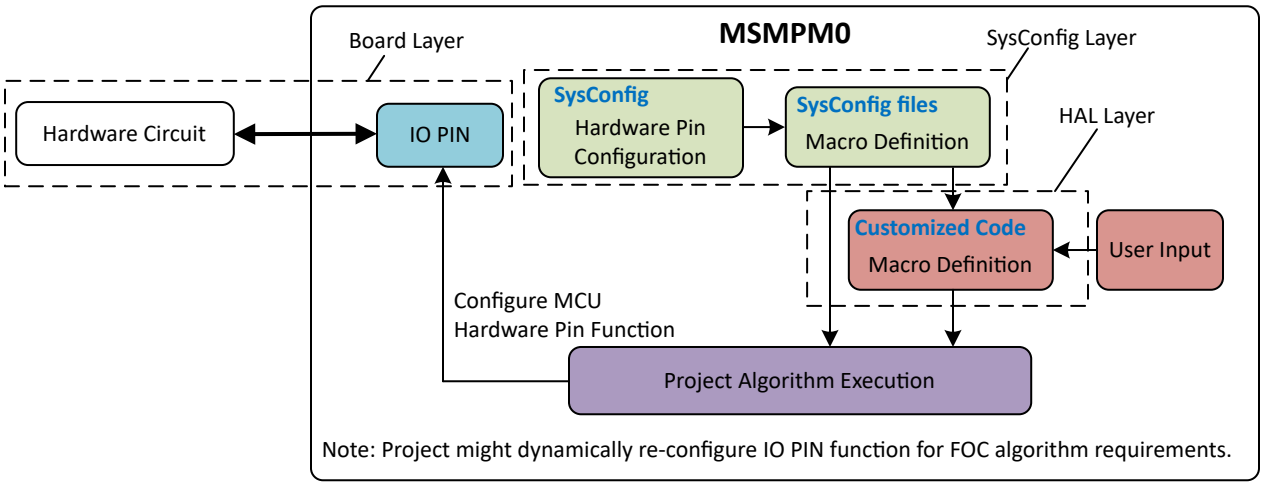


Figure 8-1. Hardware Configuration Relationship

The Board layer is the layer that users' connection for the MCU with other components in hardware circuit. For FOC applications, it is mainly for the connection to the used gate driver device and ADC sampling channels.

The SysConfig layer sets the MCU peripheral initialization features in the . ***syscfg*** file. The IO PIN connected to the external circuit needs to be set as the appropriate peripheral function to run FOC application sucessfully. The SDK project provides several peripheral default configurations to fit the DRV EVM Board or TIDA Reference Board. Users need to configure the . ***syscfg*** file to manually adapt the hardware circuit. The SysConfig tool will automatically generate MCU peripheral initialization file, including macro definitions mapped to MCU hardware peripherals.

The second HAL layer is the Customized Macro Definition in the SDK FOC project. Lots of FOC functions call customized macro definitions to determine the implementation of the algorithm. Therefore, users need manually to manage these customized macro definitions in header files to adapt the hardware circuit and SysConfig generated macro definition.

Follow the steps below to migrate the hardware configurations for customized board:

1. Check the MSPM0 IO pin used for each module on the customized board.
2. Modify SysConfig configurations to fit the hardware design.
3. Modify HAL layer macro definitions to fit the hardware design.

8.2 Gate Driver Module

FOC application uses a pre-defined symbol to determine which gate driver board is used to configure the board parameters and HAL layer properly, as shown in the following figure.

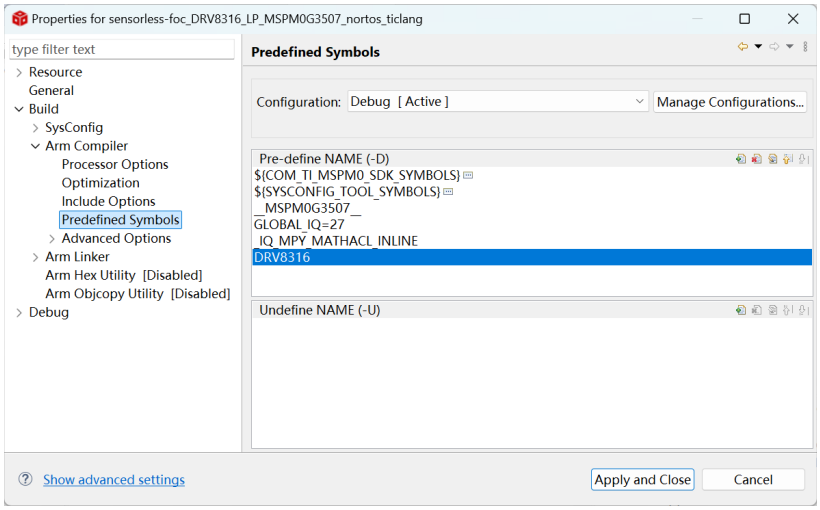


Figure 8-2. DRV8316 Pre-define Symbol in Project Properties

The default gate driver interface is configured for the EVM hardware board. Users can overwrite the relevant configuration set for DRV8316 to their own board configuration.

While recommend using **CUSTOM** symbol for users' specific hardware design if use another gate driver for better code management. Follow the steps below to configure the Custom Gate Driver Module with this approach.

8.2.1 Select Reference Projects

MSPM0 SDK provides various FOC example projects for different hardware boards. Refer to the following table to select the preferred example project to do the migration for your own hardware board.

Table 8-1. Project Recommendation for Migration

FOC Type	Customized Hard Board		Project Recommendation	Migration Effort
	Gate Driver	Current Sensing Type		
Sensorless / Universal FOC	DRV8323	Single Shunt (1)	sensorless-foc_DRV8323RS	Porting single shunt configuration.
			sensorless-foc_DRV8329	Porting gate driver configuration.
		Dual or Three Shunt	sensorless-foc_DRV8323RS	No significant effort.
	DRV8316	Single Shunt (1)	sensorless-foc_DRV8316	Porting single shunt configuration.
			sensorless-foc_DRV8329	Porting gate driver configuration.
		Dual or Three Shunt	sensorless-foc_DRV8316	No significant effort.
	Others	Single Shunt	sensorless-foc_DRV8329	No significant effort.
		Dual or Three Shunt	sensorless-foc_TIDA010250	No significant effort.

Table 8-1. Project Recommendation for Migration (continued)

FOC Type	Customized Hard Board		Project Recommendation	Migration Effort
	Gate Driver	Current Sensing Type		
Sensored FOC	DRV8316	Single Shunt (1)	hall_sensored-foc_DRV8316	Porting single shunt configuration.
			hall_sensored-foc_TIDA010251	Porting gate driver configuration.
		Dual or Three Shunt	hall_sensored-foc_DRV8316	No significant effort.
	Others	Single Shunt	hall_sensored-foc_TIDA010251	No significant effort.
		Dual or Three Shunt	hall_sensored-foc_DRV8316	No significant effort.

1. Recommend to start migration with the example project has the single shunt configuration if users' hardware board has single shunt circuit.

The following sections take the TIDA010250 as an example to introduce the flow for a customized gate driver module migration.

8.2.2 Modify Pre-defined Symbols

In the project properties, modify the default Pre-defined Symbols: TIDA010250->CUSTOM.

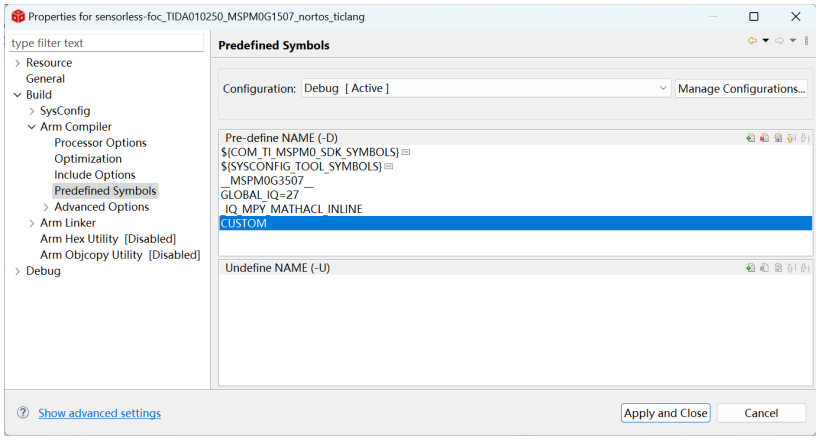


Figure 8-3. Modify Pre-define Symbol in Project Properties

8.2.3 Add Custom Source Files

By default, FOC project has integrated SPI communication for gate drivers in FOC library, users are owned to maintain the source code for gate driver communication if use customized gate drivers. To help users quickly remove the default SPI communication, FOC projects provide custom source files in SDK folder. Users could manually add source files into the customized project with **CUSTOM** pre-defined symbol.

8.2.3.1 Gate Driver Comm Folder

The preset custom source files for Gate Driver Communication are in the following SDK path:

```
...\ti\mspm0_sdk_<SDK_Version>\source\ti\motor_control_pmsm_foc\common_modules\hal\LP_MSPM0Gx5xx\gateDriverInterface\gateDriverLib
```

Figure 8-4 shows the CUSTOM Gate Driver Library folder. The library deletes default codes for SPI communications for gate drivers in FOC application, so that users can add their own gate driver communication codes (Section 8.2.4) in their customized files.

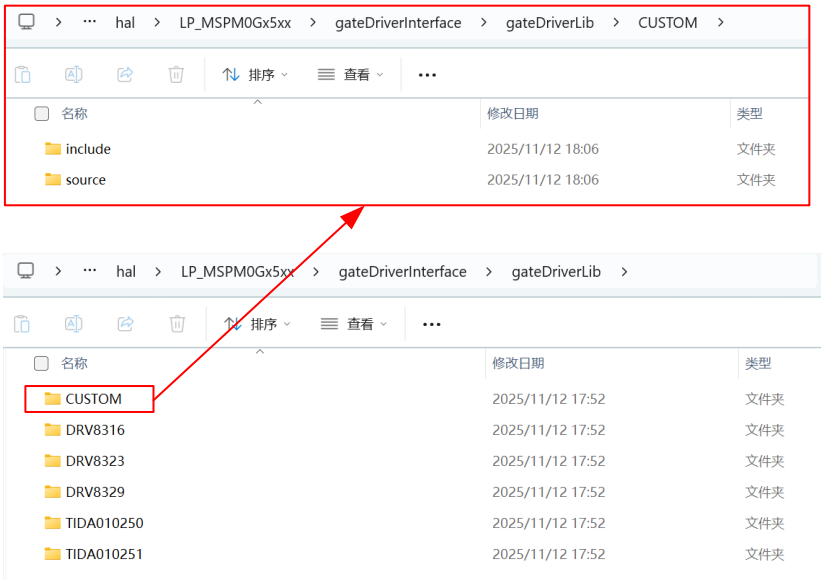


Figure 8-4. CUSTOM Gate Driver Library

Add the CUSTOM folder (copy and paste) into the example FOC project as shown in Figure 8-5. The default Gate Driver folder (TIDA010250) can be deleted (optional).

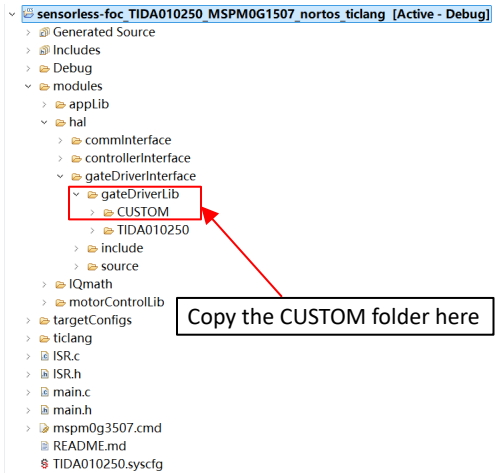


Figure 8-5. Add CUSTOM Folder into FOC Project

Add the CUSTOM folder path int the Include Options in Project Properties below:

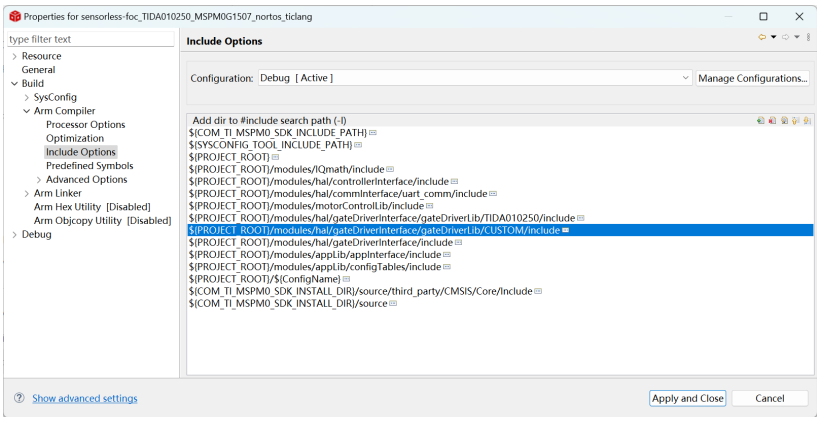


Figure 8-6. Add CUSTOM Folder in Include Options

8.2.3.2 HAL Layer File

MSPM0 SDK provides preset custom source files for HAL layer are in the following path:

```
...\ti\mspm0_sdk_<SDK_Version>\source\ti\motor_control_pmsm_foc\common_modules\hal\LP_MSPM0Gx5xx\gateDriverInterface\[LAUNCHPAD]\source
```

Figure 8-7 shows the CUSTOM HAL layer file.

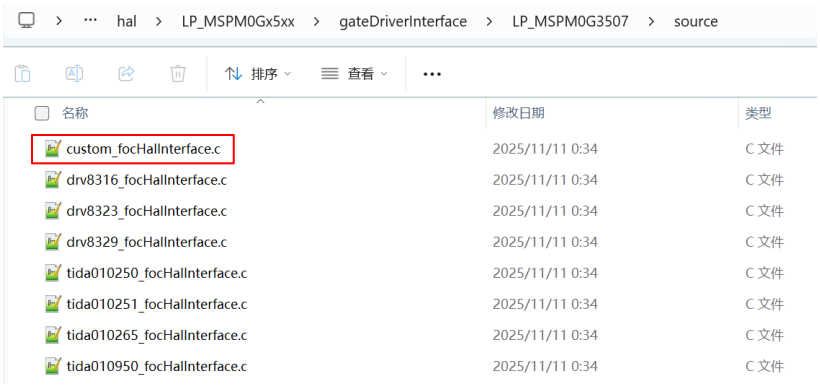


Figure 8-7. CUSTOM HAL Layer File

Add the ***custom_focHallInterface.c*** file into the ***gateDriverLib*** folder of the FOC project below:

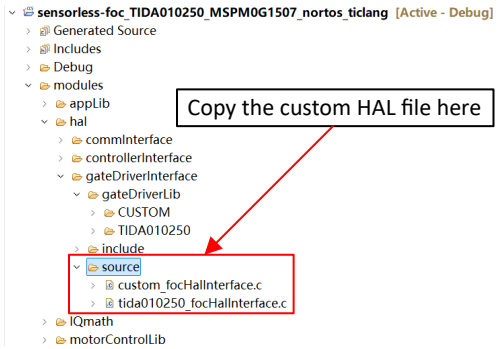


Figure 8-8. Add the Custom HAL File into FOC Project

8.2.4 Add Custom Comm Interface

Gate drivers support different Comm interfaces to configurate, including resistor based, SPI based, IIC based, and others. It is complex for users to integrate a new Comm interface into FOC HAL library and debug. An easier

way is introduced to disable default SPI based communication (DRV8316 or DRV8323 FOC project), so that the FOC algorithm does not use any MCU pins or peripherals to do communication.

Follow the steps in Section 8.2.3.1 to use CUSTOM communication interface. Then the gate driver communication APIs is set to empty and will not impact normal FOC applications, as shown in Figure 8-9. Users can also manually modify the source code as empty for the gate driver communication interface and then add their own gate driver files out of FOC application to suit their own hardware board.

Figure 8-9. Custom Gate Driver File

Note

Directly deleting APIs shown in the above figure results in compiler error as Sensorless FOC Library calls these APIs in a static library.

8.2.5 Overwrite Default Macro Definitions

8.2.5.1 main.h File

In this SDK version, the ***main.h*** misses the CUSTOM macro for user customization. Add the related code below (copy from other macro, such as TIDA-010250, and overwrite the value with CUSTOM):

Manually add CUSTOM parts

```
217
218 #elif defined CUSTOM
219
220 /*! @brief TIDA010250 has inverting isense */
221 #define _INVERT_ISEN
222 /*! @brief IPD feature is enabled */
223 #define __IPD_ENABLE
224 /*! @brief TIDA010250 propagation delay */
225 #define DRIVER_PROPAGATION_DELAY_nS                500
226 /*! @brief TIDA010250 minimum on time (rise time + settling time) */
227 #define DRIVER_MIN_ON_TIME_nS                      8000
228 /*! @brief DC voltage base value */
229 #define DC_VOLTAGE_BASE                             448.0
230 /*! @brief Full scale readable current used as current base value,
231 calculated using (FULL Scale Voltage(3.3)/2* CSA Gain) */
232 #define FULL_SCALE_CURRENT_BASE                     8.25
233 /*! @brief Current shunt configuration */
234 #define __CURRENT_THREE_SHUNT_DYNAMIC
235 /*! @brief Enable dynamic current shunt changing */
236 #define DYNAMIC_CURRENT_SHUNT_CONFIG_EN             FALSE//TRUE
237
238 #endif
```

Figure 8-10. Add the Custom Macro into main.h File

Users should take care of the macro definitions modification based on the hardware board features, shown in Table 8-2.

Table 8-2. Hardware Macro Definitions for FOC Application

Hardware Feature	Macro	Description
Current Sensing Type	_INVERT_ISEN	The board has an inverting or non-inverting current sensing hardware circuit. See Section 8.3.2.1.
	_NONINVERT_ISEN	
Current Sensing Method	__CURRENT_XX_SHUNT	Current sensing method includes Single Shunt, Dual Shunt and Three Shunt. See Section 8.3.2.2.
Board Parameters	DC_VOLTAGE_BASE	Maximum measurable bus voltage. See Section 7.1.1.
	FULL_SCALE_CURRENT_BASE	Base current. See Section 7.1.2.
	DRIVER_PROPAGATION_DELAY_nS	Defines the Time delay in ns between the Input PWM logic edge fed to the gate driver and actual Gate Driver output.
	DRIVER_MIN_ON_TIME_nS	Defines the combined rise time and settling time of the current sense amplified output.
IPD Function	__IPD_ENABLE	Enable IPD function for FOC application. The IPD configuration register should be set properly, See Section 7.7.1.1.3.

8.2.5.1.1 Delay Component in Current Sensing Path

The following figure shows the delay components in the current measurement path.

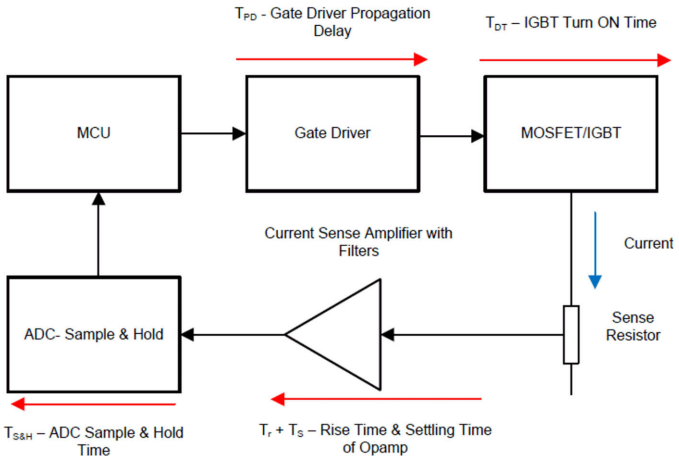


Figure 8-11. Delay in the Current Sensing Loop

FOC application uses DRIVER_PROPAGATION_DELAY to define the time delay in ns between the Input PWM logic edge fed to the gate driver and actual gate driver Output PWM. This delay impacts the Current Sense sampling instance on the actual gate driver output and has to be fed to the algorithm for accurate Current Sensing. Users can get the propagation delay from the used gate driver's data sheet or measure the time with hardware.

FOC application uses DRIVER_MIN_ON_TIME to define the combined rise time and settling time of the current sense amplified output. This value has to be captured independently for a full-scale change in voltage across the current shunt. For accurate current sense reading, the current sense amplifier output to be settled before the current signal is captured. Refer to Equation 45 to calculate the DRIVER_MIN_ON_TIME.

$$\text{DRIVER\_MIN\_ON\_TIME} = \text{CSASettlingTime} + \text{CASRiseTime} \quad (45)$$

Note

DRIVER_MIN_ON_TIME limits the maximum FOC output PWM duty cycle to reserve sufficient time for CSA to establish the stable current signal to feed ADC sampling channel.

8.2.5.2 gateDriver.h File

The ***gateDriver.h*** file defines the HAL layer macros. It is essential for users to overwrite the macros according to users' hardware board circuit.

- For PWM related macro definition overwriting, refer to Section 8.3.1.
- For ADC related macro definition overwriting, refer to Section 8.3.2.

8.3 MCU Peripheral Configuration

8.3.1 PWM Module

Using TIMA0 of MSPM0 to enable three pairs of complementary PWM output and deadband time insertion for FOC application. There have four output channels of TIMA0, and users can select any three of them for FOC application.

When the hardware circuit is finalized, users should pay attention to the correspondence between the motor phase sequence and PWM channel setting. The default mapping relationship for **sensorless-foc_DRV8316_LP_MSPM0G3507** project is shown in the following table.

Table 8-3. PWM Mapping

Board Layer		HAL Layer		Descriptions
	SysConfig Layer			
Board Macro	**IO PIN**	**PWM Channel**	**HAL Layer Macro**	
INHA	PB4	TIMA0_C2	FOC_PWMA0_U_IDX	Phase A PWM output.
INHB	PA28	TIMA0_C3	FOC_PWMA0_V_IDX	Phase B PWM output.
INHC	PB20	TIMA0_C1	FOC_PWMA0_W_IDX	Phase C PWM output.
FAULT	PA27	Fault Pin 2	NOT USED	Gate driver output fault pin.
NA	NA	TIMA0_C0	FOC_PWMA0_ADC_TRIG_IDX	Trigger channel from PWM to ADC.

The HAL Layer Macro always keeps the same mapping relationship to the Board Macro, and the SysConfig layer should be modified accordingly.

Note

In current FOC applications, the unused PWM channel is configured as ADC trigger channel, but the PIN output function is enabled in the project. Users can disable it manually in the main loop code, or use the non-output channel (CC4 or CC5), to get rid of this PWM PIN output impact.

8.3.1.1 Different Pin Used for PWM Output

For the case that users only change different PWM output pins in hardware while the mapping relationship keeps the same, users can just modify the PINMUX selection of the TIMA0output channel in SysConfig tools, as shown in the following figure.

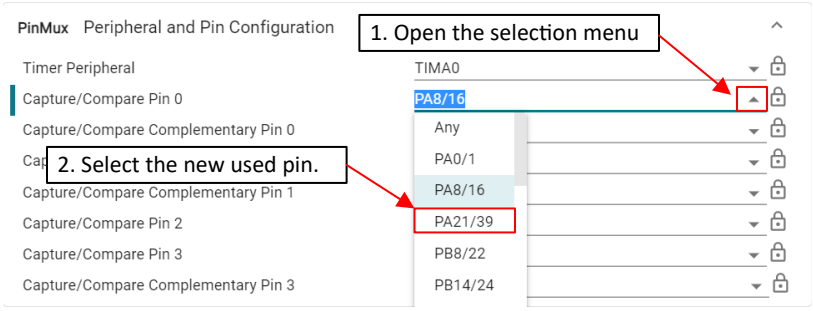


Figure 8-12. Modify PWM Output Pin in SysConfig

8.3.1.2 Different Pin Used for PWM Fault Input

For the case that users change different PWM fault pins in hardware, users can just modify the PINMUX selection of the TIMA0fault pin in SysConfig tools, as shown in Figure 8-13.

If the hardware circuit does not support FAULT protection, unselect 'Enable Fault Handler' to disable it. Incorrect FAULT signal (floating, or low pulse) might triger HV_DIE fault when spin the motor.

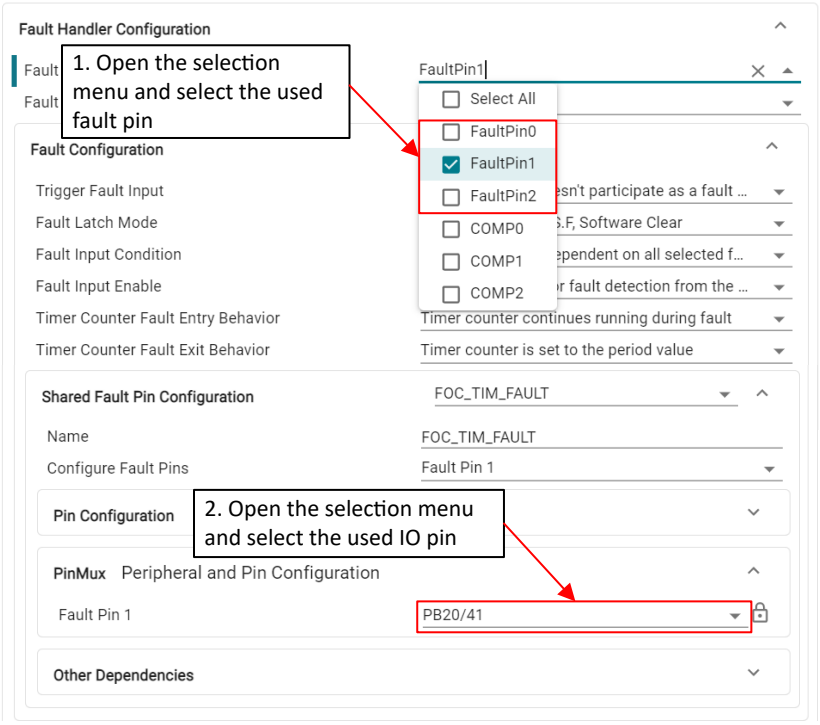


Figure 8-13. Modify PWM Fault Pin in SysConfig

8.3.1.3 Different Mapping to PWM Output Channel

For the case that the PWM output channel mapping in hardware is different, users should first refer to Section 8.3.1.1 to modify the IO PIN used in SysConfig. Then position to ***gateDriver.h*** file to overwrite the HAL layer macro.

For example, if users have a new mapping table below:

Table 8-4. Custom PWM Mapping

Board Macro	IO PIN	PWM Channel	HAL Layer Macro
INHA	TBD (Set in SysConfig)	TIMA0_C0	FOC_PWMA0_U_IDX
INHB	TBD (Set in SysConfig)	TIMA0_C1	FOC_PWMA0_V_IDX
INHC	TBD (Set in SysConfig)	TIMA0_C2	FOC_PWMA0_W_IDX
NA	NA	TIMA0_C3	FOC_PWMA0_ADC_TRIG_IDX

The corresponding modifications in ***gateDriver.h*** file are shown in the following figure.

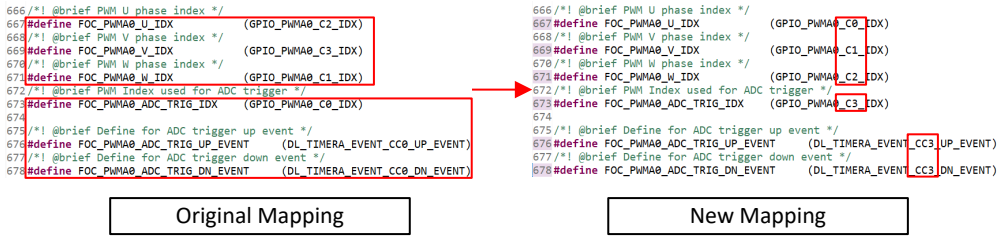


Figure 8-14. Modify HAL Layer Macro

The remaining unused PWM output channel is always used to trigger ADC sampling in dual or three shunt current sensing method. And the ADC trigger EVENT should be set as the event of the trigger PWM channel.

For single shunt current sensing method, FOC application uses the secondary TIMA1 to trigger the ADC, so that no trigger channel modification is required.

8.3.2 ADC Module

When migrating ADC module to customized setting, usersshould pay attention to the mapping relationship as well as the current sensing method used in hardware circuit.

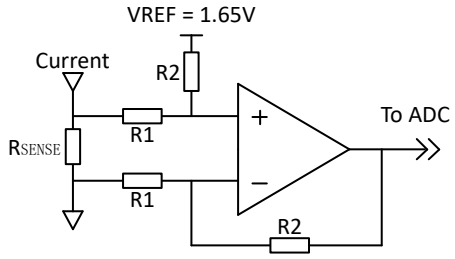
8.3.2.1 Current Sensing Type

Users should overwrite the macro definitions in the project according to the hardware sensing circuit type in ***main.h*** file, as shown in the following table.

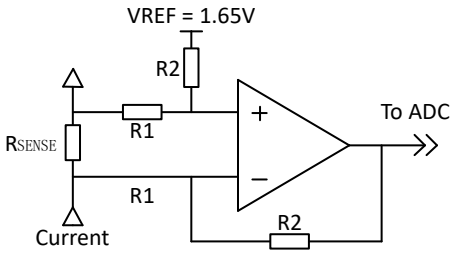
Table 8-5. Current Sensing Type

File	Macro Definition	Description
main.h	_NONINVERT_ISEN	Hardware board uses non-inverting current sensing circuit
	_INVERT_ISEN	Hardware board uses inverting current sensing circuit

Figure 8-15 shows the typical bidirectional current sensing circuit block diagram for users' reference.



a. Non-inverting Current Sensing Circuit



b. Inverting Current Sensing Circuit

Figure 8-15. Current Sensing Type

8.3.2.2 Current Sensing Method

The SDK FOC example can be configured for various shunt configuration options such as Single Shunt, Dual Shunt and Three Shunt. Based on the HW design the appropriate shunt configuration has to be selected for proper operation of algorithm. FOC application supports simultaneously sampling the two phases at a given instance to optimize the current sampling time. By default, in all shunt configurations, both the ADC instances are used to utilize the simultaneous sampling feature.

8.3.2.2.1 Three Shunt Configuration

The following table shows the macro for three shunt configurations.

Table 8-6. Macro for Three Shunt Sensing

File	Macro Definition	Description
main.h	__CURRENT_THREE_SHUNT_AB_C	Select this configuration if A and B phases are sensed through ADC0 and C phase is sensed through ADC1.
	__CURRENT_THREE_SHUNT_A_BC	Select this configuration if A phase is sensed through ADC0 and B, C phases are sensed through ADC1.

8.3.2.2.2 Three Shunt Configuration with Simultaneously Sampling

Users can also route one of the Phases say 'B' to both the ADC0 and ADC1 instance and the other two phases to two different ADC instances.

For example, Phase A is routed to ADC0, and Phase C is routed to ADC1. Phase B is routed to both ADC0 and ADC1 instances. Then algorithm can dynamically switch to the two samples which gives the best current sampling time based on the given sector if enabled simultaneously sampling. Refer to Section 9.3 if users want to reduce 1x ADC pin for simultaneously sampling function.

Under simultaneously sampling mode, FOC application supports shifting the current sensing estimation dynamically to the two phases for maximizing the modulation index. As in a balanced three phase Motor, any one of the three-phase currents can be estimated with the other two-phase currents in Equation 46.

$$I_a + I_b + I_c = 0 \tag{46}$$

Based on the operational sector the two phases with lowest modulation index are selected for current measurement and third phase with highest modulation index is estimated using the other two-phase currents. This method helps in extending the modulation index to higher limits with continuous SVM operation.

The following table shows the example to enable simultaneously sampling.

Table 8-7. Macro for Simultaneously Sampling

File	Macro Definition	Description
main.h	__CURRENT_THREE_SHUNT_DYNAMIC	Add macro to the file.
	DYNAMIC_CURRENT_SHUNT_CONFIG_EN	Add macro to the file and set it to TRUE.

8.3.2.2.3 Dual Shunt Configuration

The following table shows the macro for dual shunt configurations.

Table 8-8. Macro for Three Shunt Sensing

File	Macro Definition	Description
main.h	__CURRENT_TWO_SHUNT_A_B	Select this configuration if only two shunt sense across phase A and B are available current sampling where A is channeled to ADC 0 and B to ADC1.
	__CURRENT_TWO_SHUNT_B_C	Select this configuration if only two shunt sense across phase B and C are available current sampling where B is channeled to ADC 0 and C to ADC1.
	__CURRENT_TWO_SHUNT_A_C	Select this configuration if only two shunt sense across phase A and B are available current sampling and A is channeled to ADC 0 and C to ADC1.

If user has a different combination of phases routed to the ADC 0 and ADC1 instances than the default connections in SDK. Appropriate changes are required to be made, see Section 8.3.2.4.1.2.

8.3.2.2.4 Single Shunt Configuration

Using **__CURRENT_SINGLE_SHUNT** for single shunt configurations.

Single shunt configuration requires users to add additional TIMA1 instance, enable cross trigger function with TIMA0. Recommend porting configurations from SDK example project with single shunt configuration as shown in Table 8-1 and following the instructions in Section 8.3.2.4.1.3.

8.3.2.3 CSA Offset Scaling Factor

FOC application converts the sampled currents through ADC into PU system based on the maximum current that can be sensed through the ADC. This depends on the CSA offset introduced from the amplifier.

Typically for bipolar current sense measurement, full scale value of ADC 3.3V / 2 = 1.65V is given as offset. For applications where the current sensing is always unipolar, offset values are set at less than 0.5V to use the maximum full scale ADC output for positive current measurement and small margin is left for negative current measurement.

FOC application requires this scaling to be specified for appropriate functionality. As the ADC 12-bit value is converted to PU value, if the offset is set as 0: Then the scaling factor to be set as _IQ(1). If the CSA offset in HW is set as 1.65V (3.3V / 2) for bipolar current sense measurement the scaling factor to be set as _IQ(2). For any arbitrary offset values, the scaling values to be specified as Equation 47:

$$\_IQ(CSA\_OFFSET[PU]) = \_IQ(3.3V/(3.3V - CSA\_OFFSET[V])) \quad (47)$$

FOC application provides a macro for users to change the preset CSA offset defined for single shunt current sensing method, as shown in the following table.

Table 8-9. Macro for CSA Offset

File	Macro Definition	Description
Drv8329_focHalInterface.c	DRV8329_CURRENT_SF_IQ	Define the CSA offset scaling factor for single shunt current sensing method.

For dual or three shunt current sensing method, FOC application hardcode the scaling factor to _IQ(2) in ***focHALInterface.c*** file by applying format conversion API (_IQ11toIQ).

8.3.2.4 Channel Mapping

Table 8-10 shows the default mapping relationship for **sensorless-foc_DRV8316_LP_MSPM0G3507** project.

Table 8-10. ADC Mapping

Board Layer		HAL Layer		SysConfig Configurations
	SysConfig Layer			
Board Connection	**ADC CHAN**	**ADC MEMRES**	**HAL Layer Macro**	
Phase A Current	A0_3	A0_MEM0	ADC0_CURRENT_U_CH	Initializes Memory 0/1 of ADC0/ADC1. The channel selection does not matter as the FOC application dynamically overwrites the channel settings according to HAL layer macro in runtime.
Phase B Current 1	A0_2	A0_MEM1	ADC0_CURRENT_V_CH	
Phase B Current 2	A1_2	A1_MEM0	ADC1_CURRENT_V_CH	
Phase C Current	A1_1	A1_MEM0	ADC1_CURRENT_W_CH	
Bus Voltage	A1_3	A1_MEM2	FOC_ADC_VOLT_DC_INST ADC_VOLT_DC_IDX	Assign the ADC1 channel 3 to Memory 2.
Phase A Voltage	A1_6	A1_MEM0	ADC_VOLTAGE_U_INST ADC_VOLTAGE_U_IDX ADC_VOLTAGE_U_CH	Not initialized in SysConfig and dynamically initialized in the FOC application if run ISD function.
Phase B Voltage	A0_7	A0_MEM0	ADC_VOLTAGE_V_INST ADC_VOLTAGE_V_IDX ADC_VOLTAGE_V_CH	
Phase C Voltage	A1_5	A1_MEM1	ADC_VOLTAGE_W_INST ADC_VOLTAGE_W_IDX ADC_VOLTAGE_W_CH	

Table 8-10. ADC Mapping (continued)

Board Layer		HAL Layer		SysConfig Configurations
	SysConfig Layer			
Board Connection	ADC CHAN	ADC MEMRES	HAL Layer Macro	
ADC Interrupt	NA	A0_MEM1	FOC_ADC_ISR_INST	Initialize ADC0 interrupt configuration with MEM1 Result Loaded.
			FOC_ADC_MEM_RES_LOAD	

The FOC application dynamically modifies ADC channel and ADC memory result register mapping for different modes. In ISD mode, FOC applications use ADC0 MEM0 and ADC1 MEM0/1 to store sampled phase voltages. In other mode, FOC application use ADC0 MEM0/1 and ADC1 MEM0/1 to store sampled phase currents. Table 8-11 shows the API used by FOC application in ***Drv8316_focHallInterface.c*** file.

Table 8-11. APIs to Modify ADC Mapping

FOC API	HAL Layer Macro	Description
HAL_GD_ConfigureVoltageChannels()	ADC_VOLTAGE_X_INST ADC_VOLTAGE_X_IDX ADC_VOLTAGE_X_CH	Map the phase voltage channel (X_CH) to certain ADC MEM (X_IDX) of ADC Instance (X_INST). X = U / V / W.
HAL_GD_ReadVoltages()	ADC_VOLTAGE_U_INST ADC_VOLTAGE_U_IDX	Read the phase voltage from the ADC MEM (X_IDX) of the ADC Instance (X_INST). X = U / V / W.
HAL_GD_ConfigureCurrentChannels()	ADC0_CURRENT_X_CH	Map the phase current channel (X_CH) to the **hardcoded** ADC instance and ADC Memory. X = U / V / W.
HAL_GD_ReadCurrents()	NA	Read the phase current from the hardcoded ADC Memory of ADC Instance.
HAL_GD_ReadDCVBusVoltage()	FOC_ADC_VOLT_DC_INST ADC_VOLT_DC_IDX	Read the bus voltage from the pre-defined ADC memory of ADC instance.

The following sections introduce the flow if user's hardware does not follow the default mapping of SDK project.

8.3.2.4.1 Phase Current Channels

The modifiable macros for phase current channels are the ADC channels, the remaining configurations (ADC Instance and ADC Index) are hardcoded in the ***HAL_GD_ConfigureCurrentChannels()*** API, as shown in Figure 8-16.

```
168 void HAL_GD_ConfigureCurrentChannels(CURRENT_SHUNT_TYPES currentShunt)
169 {
170     /* Configure the ADC channels based on the ADC pin configurations */
171     switch(currentShunt)
172     {
173           case CURRENT_THREE_SHUNT_DYNAMIC:
174           case CURRENT_THREE_SHUNT_AB_C:
175               HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
176               HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_1, ADC0_CURRENT_V_CH);
177               HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
178               break;
179           case CURRENT_THREE_SHUNT_A_BC:
180               HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
181               HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_V_CH);
182               HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_1, ADC1_CURRENT_W_CH);
183               break;
184
185           case CURRENT_TWO_SHUNT_A_B:
186               HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
187               HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_V_CH);
188               break;
189
190           case CURRENT_TWO_SHUNT_A_C:
191               HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
192               HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
193               break;
194
195           case CURRENT_TWO_SHUNT_B_C:
196               HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_V_CH);
197               HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
198               break;
199
200           case CURRENT_SINGLE_SHUNT:
201               break;
202           default:
203               break;
204     }
205 }
```

With different current shunt configurations, foc application set the ADC channels accordingly

Figure 8-16. Source Code for HAL_GD_ConfigureCurrentChannels

Figure 8-17 shows the ADC channel macros of phase currents (in gateDriver.h file).

```
680 /*! @brief Instance for ADC Interrupt */
681 #define FOC_ADC_ISR_INST        (ADC0_INST)
682 /*! @brief Memory Load Register for ADC Interrupt */
683 #define FOC_ADC_MEM_RES_LOAD    DL_ADC12_IIDX_MEM1_RESULT_LOADED
684
685 /*! @brief ADC instance for DC bus voltage */
686 #define FOC_ADC_VOLT_DC_INST    (ADC0_INST)
687 /*! @brief IDX for the DC bus voltage */
688 #define ADC_VOLT_DC_IDX         (ADC0_ADCMEM_3)
689
690 /*! @brief Channel for Phase U current */
691 #define ADC0_CURRENT_U_CH       (DL_ADC12_INPUT_CHAN_3)
692 /*! @brief Channel for Phase V current */
693 #define ADC0_CURRENT_V_CH       (DL_ADC12_INPUT_CHAN_1)
694 /*! @brief Channel for Phase V current */
695 #define ADC1_CURRENT_V_CH       (DL_ADC12_INPUT_CHAN_2)
696 /*! @brief Channel for Phase W current */
697 #define ADC1_CURRENT_W_CH       (DL_ADC12_INPUT_CHAN_1)
```

Define the ADC interrupt trigger source. Use the last ADC MEM for phase current as the trigger source

Define the ADC channel used by users' hardware circuit. Modify the CHAN accordingly.

Figure 8-17. Phase Current ADC Channel Macros

Depending on the current sensing method of the hardware design, users should overwrite source code of the ***HAL_GD_ConfigureCurrentChannels()*** API and ADC channel macros for phase currents in ***gateDriver.h*** file to sample phase current correctly.

8.3.2.4.1.1 Three Shunt Configuration

Figure 8-18 shows the three or dual shunt configuration signal chain. Recommend using two ADC instances to sample and convert three phase currents for MSPM0G series.

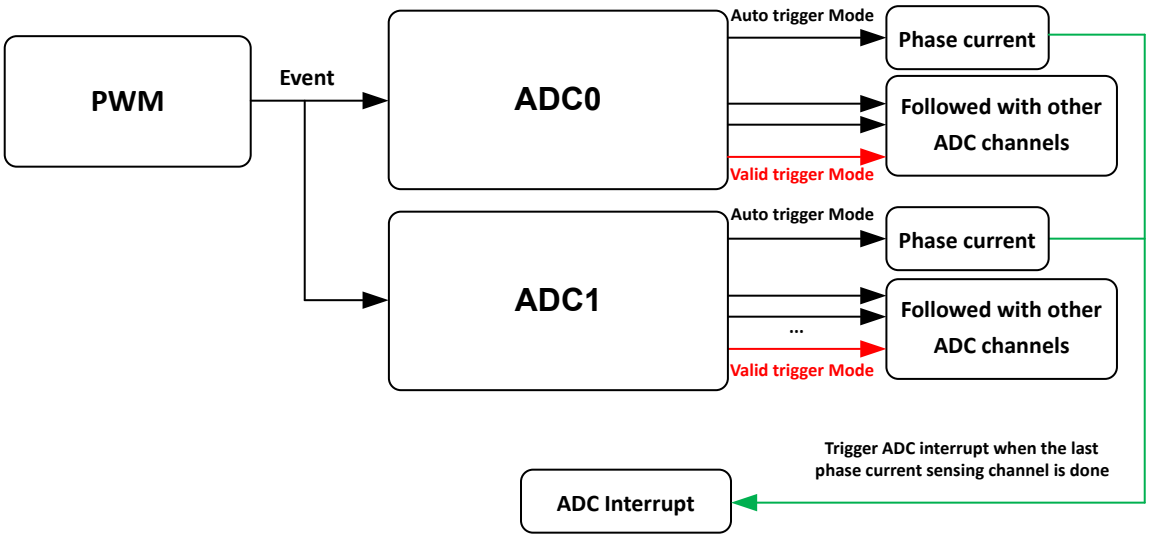


Figure 8-18. Peripheral Connection in Three or Dual Shunt Configuration

Follow the steps below to modify phase current channels according to the hardware circuit.

1. Add the corresponding macro definition (Section 8.3.2.2) into main.h file
2. Modify the ADC channels in ***gateDriver.h*** file according to your hardware design
3. If Simultaneously Sampling is not enabled, users can set the ADC0_CURRENT_V_CH and ADC1_CURRENT_V_CH as the same
4. Select the last ADC MEM Index for phase current sampling as the interrupt trigger source. For example, if users set all three phase currents to the ADC0, then FOC application uses three ADC MEM Index (MEM0, MEM1, MEM2), so that the FOC_ADC_MEM_RES_LOAD should be set as ADC MEM2
5. Add macro FOC_ISR_ADC1 in ***gateDriver.h*** file if the ADC interrupt is used with ADC1 instance
6. Initialize ADC Memory accordingly and set the ADC interrupt source in SysConfig. The ADC channel setting is optional as FOC application calls HAL_GD_ ConfigureCurrentChannels() to re-configure it in run-time
7. Overwrite the source code in HAL_GD_ConfigureCurrentChannels() API to meet the ADC instance and ADC Memory Index configuration implemented in step 2 ~ 6, as the ADC instance setting and memory read selection is hardcoded
8. Overwrite the source code in HAL_GD_HAL_GD_ReadCurrents() API to meet the ADC instance and ADC Memory Index configuration implemented in step 7

Figure 8-19 shows an example of modifying all phase current channels mapping to ADC1. Example ADC channels setting refers to below:

- Phase U current channel: ADC1.1 -> ADC MEM0
- Phase V current channel: ADC1.2 -> ADC MEM1
- Phase U current channel: ADC1.3 -> ADC MEM2 -> Trigger ADC1 Interrupt

Note

Example only. Not recommend using one ADC instance for all three current sampling.

Step 1: Add the current shunt configuration

```
#elif defined CUSTOM

/*! @brief TIDA010250 has inverting isense */
#define _INVERT_ISEN
/*! @brief IPD feature is enabled */
#define __IPD_ENABLE
/*! @brief TIDA010250 propagation delay */
#define DRIVER_PROPAGATION_DELAY_nS                500
/*! @brief TIDA010250 minimum on time (rise time + settling time) */
#define DRIVER_MIN_ON_TIME_nS                      8000
/*! @brief DC voltage base value */
#define DC_VOLTAGE_BASE                             448.0
/*! @brief Full scale readable current used as current base value,
calculated using (FULL Scale Voltage(3.3)/2* CSA Gain) */
#define FULL_SCALE_CURRENT_BASE                     8.25
/*! @brief Current shunt configuration */
#define __CURRENT_THREE_SHUNT_DYNAMIC
/*! @brief Enable dynamic current shunt changing */
#define DYNAMIC_CURRENT_SHUNT_CONFIG_EN             FALSE//TRUE

#endif
```

Step 2 & 3: Set ADC Current Channel

```
/*! @brief Channel for Phase U current */
#define ADC0_CURRENT_U_CH         (DL_ADC12_INPUT_CHAN_1)
/*! @brief Channel for Phase V current */
#define ADC0_CURRENT_V_CH         (DL_ADC12_INPUT_CHAN_2)
/*! @brief Channel for Phase V current */
#define ADC1_CURRENT_V_CH         (DL_ADC12_INPUT_CHAN_2)
/*! @brief Channel for Phase W current */
#define ADC1_CURRENT_W_CH         (DL_ADC12_INPUT_CHAN_3)
```

This example uses all ADC1 channels.
And simultaneously sampling is not enabled.

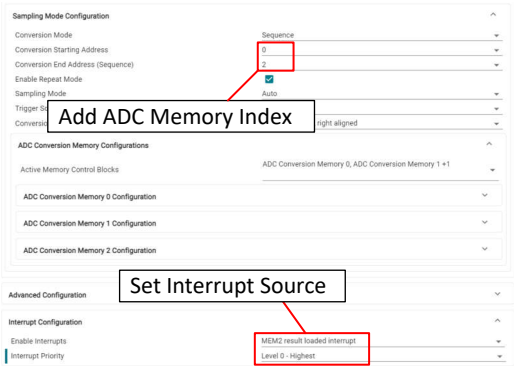
Step 4 & 5: Set ADC Interrupt Source

```
/*! @brief Instance for ADC Interrupt */
#define FOC_ADC_ISR_INST          (ADC1_INST)
/*! @brief Memory Load Register for ADC Interrupt */
#define FOC_ADC_MEM_RES_LOAD       DL_ADC12_IIDX_MEM2_RESULT_LOADED

#define FOC_ISR_ADC1
```

This example uses all ADC1 channels, so the ADC MEM is increased to 2 (0,1,2)
Due to ADC1 is used with larger Memory Index for phase current smapling, there add FOC_ISR_ADC1 macro for FOC application

Step 6: Configure SysConfig



Step 7: HAL_GD_ConfigureCurrentChannels()

```
void HAL_GD_ConfigureCurrentChannels(CURRENT_SHUNT_TYPES currentShunt)
{
    /* Configure the ADC channels based on the ADC pin configurations */
    switch(currentShunt)
    {
        case CURRENT_THREE_SHUNT_DYNAMIC:
        case CURRENT_THREE_SHUNT_AB_C:
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC0_IDX_1, ADC0_CURRENT_V_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_2, ADC1_CURRENT_W_CH);
            break;
        case CURRENT_THREE_SHUNT_A_BC:
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_1, ADC1_CURRENT_V_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_2, ADC1_CURRENT_W_CH);
            break;

        case CURRENT_TWO_SHUNT_A_B:
            HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_V_CH);
            break;

        case CURRENT_TWO_SHUNT_A_C:
            HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_U_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
            break;

        case CURRENT_TWO_SHUNT_B_C:
            HAL_setADCIdxChannel(FOC_CURR_ADC0_INST, ADC0_IDX_0, ADC0_CURRENT_V_CH);
            HAL_setADCIdxChannel(FOC_CURR_ADC1_INST, ADC1_IDX_0, ADC1_CURRENT_W_CH);
            break;

        case CURRENT_SINGLE_SHUNT:
            break;
```

This example uses all ADC1 channels, so the ADC MEM is increased to 2 (0,1,2). The ADCX_INST and ADC_IDX should be both overwritten according to the hardware circuit. Users only requires to modify the used THREE_SHUNT_XX related branch. Here modify both and is optional.

Step 8: HAL_GD_ReadCurrents()

```
void HAL_GD_ReadCurrents(HAL_MEASURE_CURRENT_T *pCurrent)
{
    int32_t adc0Idx0,adc0Idx1,adc1Idx0,adc1Idx1;

    MC_ABC_T* iabcRaw = &pCurrent->iabcRaw;

    CURRENT_SHUNT_TYPES currentShunt = pCurrent->currentShunt;

    //ADC1-MEM0
    adc0Idx0 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC0_IDX_0);
    //ADC1-MEM1
    adc0Idx1 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC0_IDX_1);
    adc1Idx0 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC1_IDX_1);
    //ADC1-MEM2
    adc1Idx1 = DL_ADC12_getMemResult(FOC_CURR_ADC1_INST, FOC_CURR_ADC1_IDX_2);

    switch(pCurrent->currentShunt)
    {
        case CURRENT_THREE_SHUNT_DYNAMIC:
        case CURRENT_THREE_SHUNT_AB_C:
            iabcRaw->a = adc0Idx0;   //ADC1-MEM0
            iabcRaw->b = adc0Idx1;   //ADC1-MEM1
            iabcRaw->c = adc1Idx1;   //ADC1-MEM2
        break;

        case CURRENT_THREE_SHUNT_A_BC:
            iabcRaw->a = adc0Idx0;   //ADC1-MEM0
            iabcRaw->b = adc1Idx0;   //ADC1-MEM1
            iabcRaw->c = adc1Idx1;   //ADC1-MEM2
        break;

        case CURRENT_TWO_SHUNT_A_B:
            iabcRaw->a = adc0Idx0;
            iabcRaw->b = adc1Idx0;
        break;

        case CURRENT_TWO_SHUNT_A_C:
            break;
        default:
```

This example uses all ADC1 channels, so ADC MEM read back code should be modified according to Step 7.
The Phase U is read from ADC1 – MEM0 (iabcRaw->a)
The Phase V is read from ADC1 – MEM1 (iabcRaw->b)
The Phase W is read from ADC1 – MEM0 (iabcRaw->c)

Figure 8-19. Modify ADC Current Channel in Three Shunt Configuration

8.3.2.4.1.2 Dual Shunt Configuration

Refer to the Three Shunt Configuration Flow (Section 8.3.2.2.1) for dual shunt configuration, with the minus difference below:

- For Step 1, users can select dual shunt configuration macro from the available CURRENT_TWO_SHUNT_X_Y (X, Y = A, B, C).
- For Step 7 and Step 8, only the used CURRENT_TWO_SHUNT_X_Y (Section 8.3.2.2.3) branch requires to be overwritten, others can remain unchanged.

8.3.2.4.1.3 Single Shunt Configuration

Figure 8-20 shows the single shunt configuration signal chain.

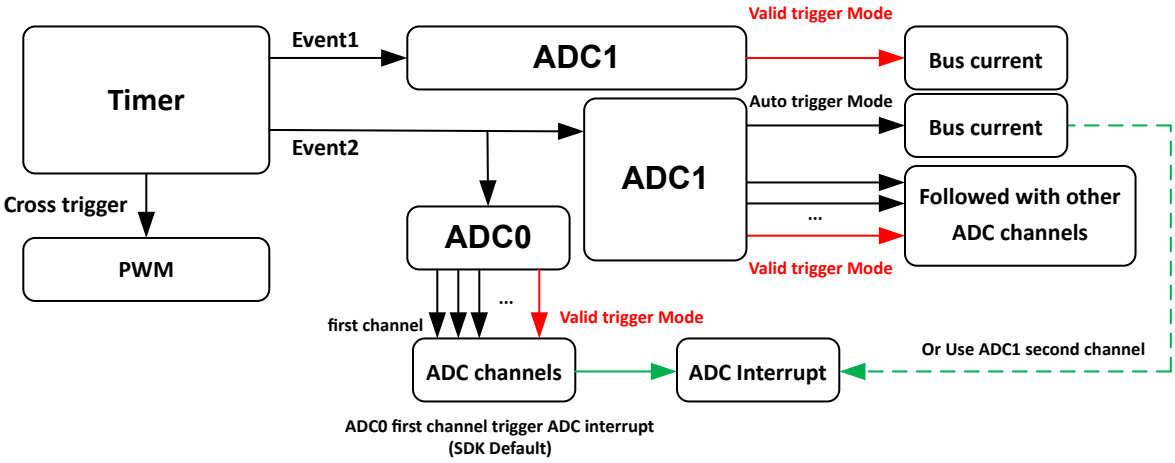


Figure 8-20. Peripheral Connection in Single Shunt Configuration

Follow the steps below to modify phase current channels according to the hardware circuit.

1. Add the corresponding macro definition __CURRENT_SINGLE_SHUNT into main.h file.
2. Modify the ADC channels in gateDriver.h file according to your hardware design. Single shunt requires one ADC channel and two ADC MEM Index as it samples twice in one PWM period.
3. Select ADC MEM0 Index of the other ADC instance as the interrupt trigger source. For example, if users use ADC1 for bus current sampling, then use ADC0 instance for interrupt trigger.
   a. If ADC1 interrupt is used, add macro FOC_ISR_ADC1 into gateDriver.h file.
4. Initialize ADC Memory accordingly and set the ADC interrupt source in SysConfig. The ADC channel setting is optional as FOC application calls HAL_GD_ ConfigureCurrentChannels() to re-configure it.
   a. If ADC1 interrupt is used, switch Event Trigger for ADC0 and ADC1 in TIMA1.

Figure 8-21 shows an example of modifying ADC current channel mapping in Single Shunt Configuration.

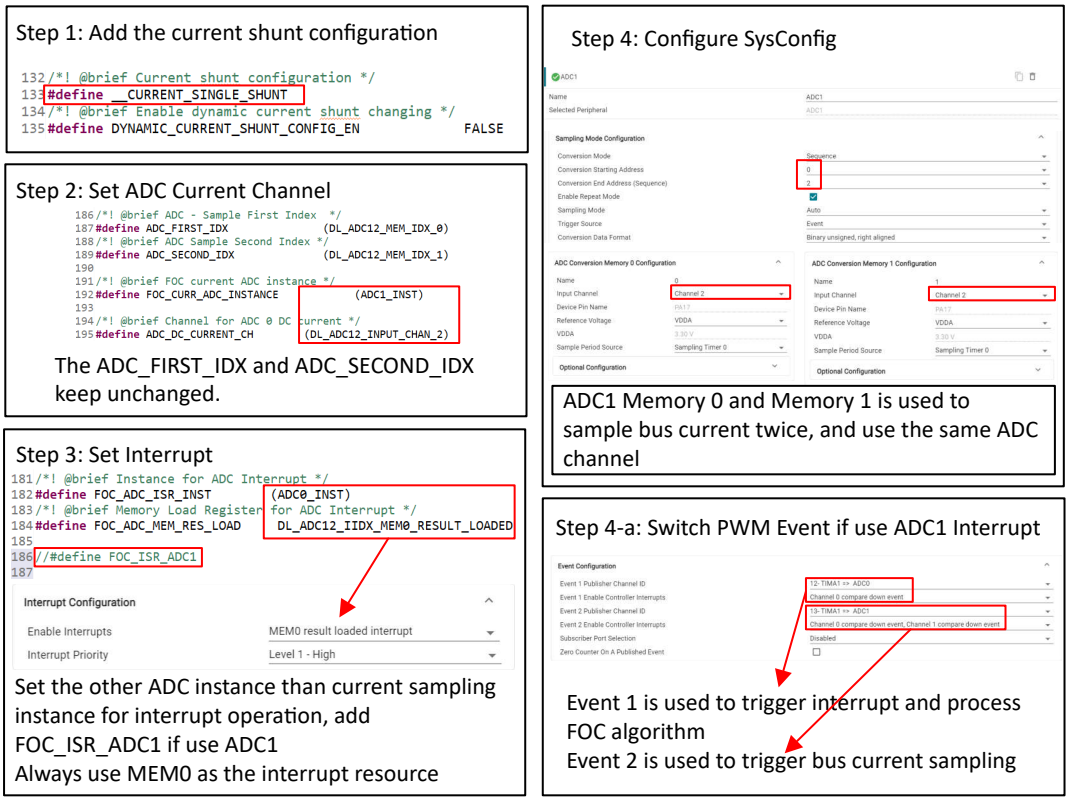


Figure 8-21. Modify ADC Current Channel in Single Shunt Configuration

Note

Migration for single shunt is not restricted to one certain pattern. Users should take care of the interrupt setting, including setting proper event trigger in Timer, and setting proper trigger mode and interrupt trigger in ADC to meet the requirements defined in Figure 8-20.

8.3.2.4.2 Bus Voltage Channel

Follow the steps below to modify bus voltage channel according to the hardware circuit.

1. Assign the bus voltage channel to the desired ADC Instance and ADC Memory in SysConfig
2. Overwrite the HAL Layer Macro in gateDriver.h file accordingly

The following figure shows an example to modify voltage channel mapping: assign bus voltage channel (ADC0_4 defined in new hardware circuit) to ADC0 Memory 3.

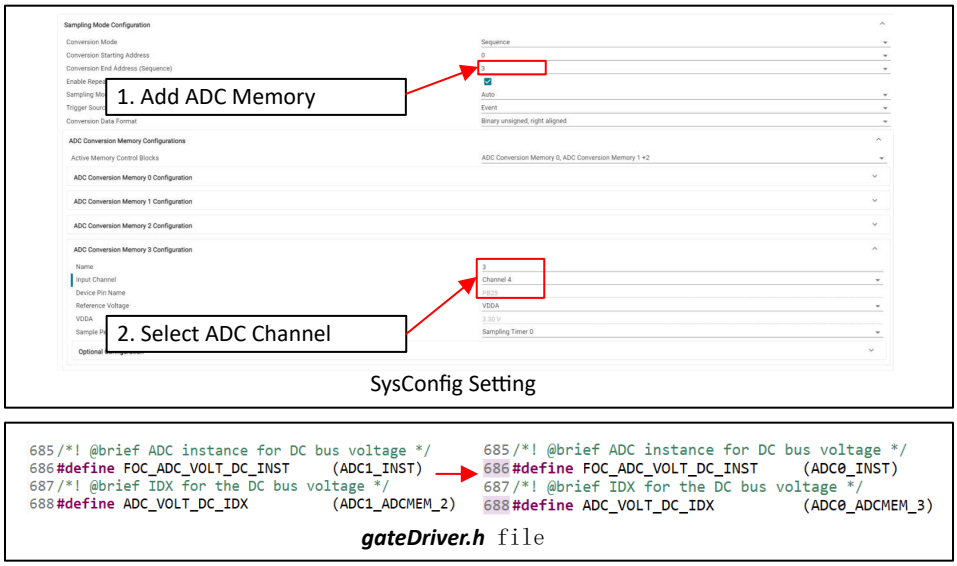


Figure 8-22. Modify Bus Voltage Channel

TI recommends using ADC0 or ADC1 MEM2 for bus voltage sampling (after phase current sampling) and keeping other ADC channel followed by bus voltage channel.

8.3.2.4.3 Phase Voltage Channels

Follow the steps below to modify phase voltage channels according to the hardware circuit.

1. Make sure the SysConfig has initialized the ADC Memory Index (ADCx_ADCMEM_y). There is no need to be set with corresponding hardware voltage ADC channel, as FOC application calls HAL_GD_ConfigureVoltageChannels() to re-configure it.
2. Overwrite the HAL Layer Macro in gateDriver.h file according to hardware circuit.

Figure 8-23 shows an example of modifying phase voltage channel mapping.

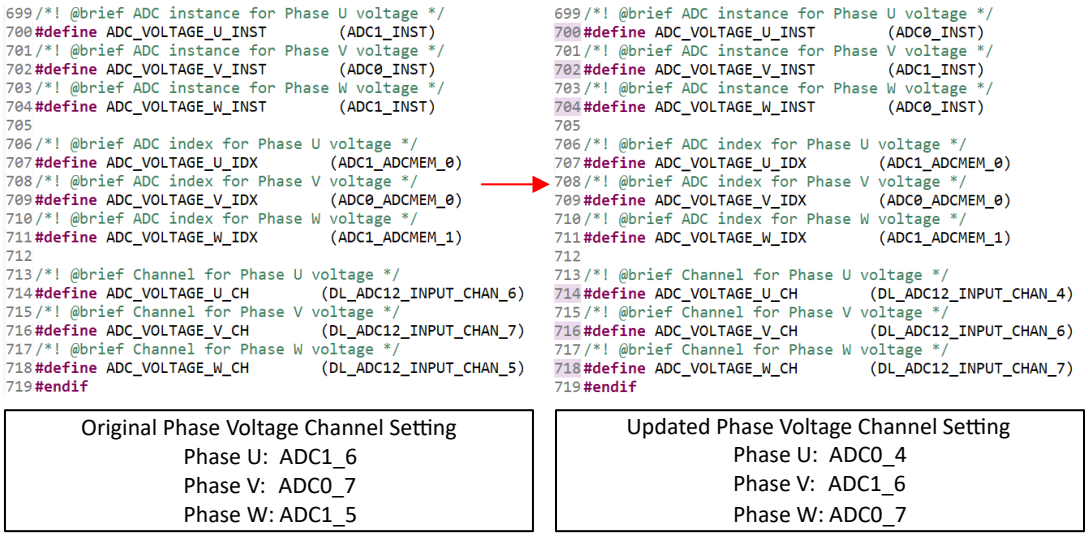


Figure 8-23. Modify Phase Voltage Channel

The ADC Memory Index can be left unchanged unless all three index is loaded to the same ADC Instance.

Note

If the Initial Speed Detection (ISD) feature is not applied, the phase voltage channel is unused in foc application.

8.3.2.5 Trigger Mode

FOC application start the ADC conversion by a hardware event generated by PWM module to avoid software delay. The ADC repeat mode is used with valid trigger mode to implement periodically conversion in every PWM cycle. The following table introduces the difference between auto step and valid trigger mode.

Table 8-12. ADC Trigger Mode Behavior

ADC Trigger Mode	Description
Auto step to next conversion	When ADC finishes current memory conversion, it automatically steps to next memory and starts conversion.
Valid trigger to next conversion	When ADC finishes current memory conversion, it waits for another trigger signal to start conversion at next memory. If no trigger signal occurs, ADC stays at current ADC memory and with no operation.

Follow the steps below to set proper trigger mode of the ADC channels for FOC application.

8.3.2.5.1 Three or Dual Shunt Configuration

For three or dual shunt configurations, FOC application converts phase currents one time in one PWM period. The ADC instance used for phase current sampling must set and only set the last ADC memory Index with Valid Trigger mode. So that in every PWM period cycle, PWM module generates an event to ADC module when low side MOSFET all ON, and this event triggers ADC to sample and convert phase currents. The following figure shows the SysConfig setting of ***sensorless-foc_DRV8316*** project.

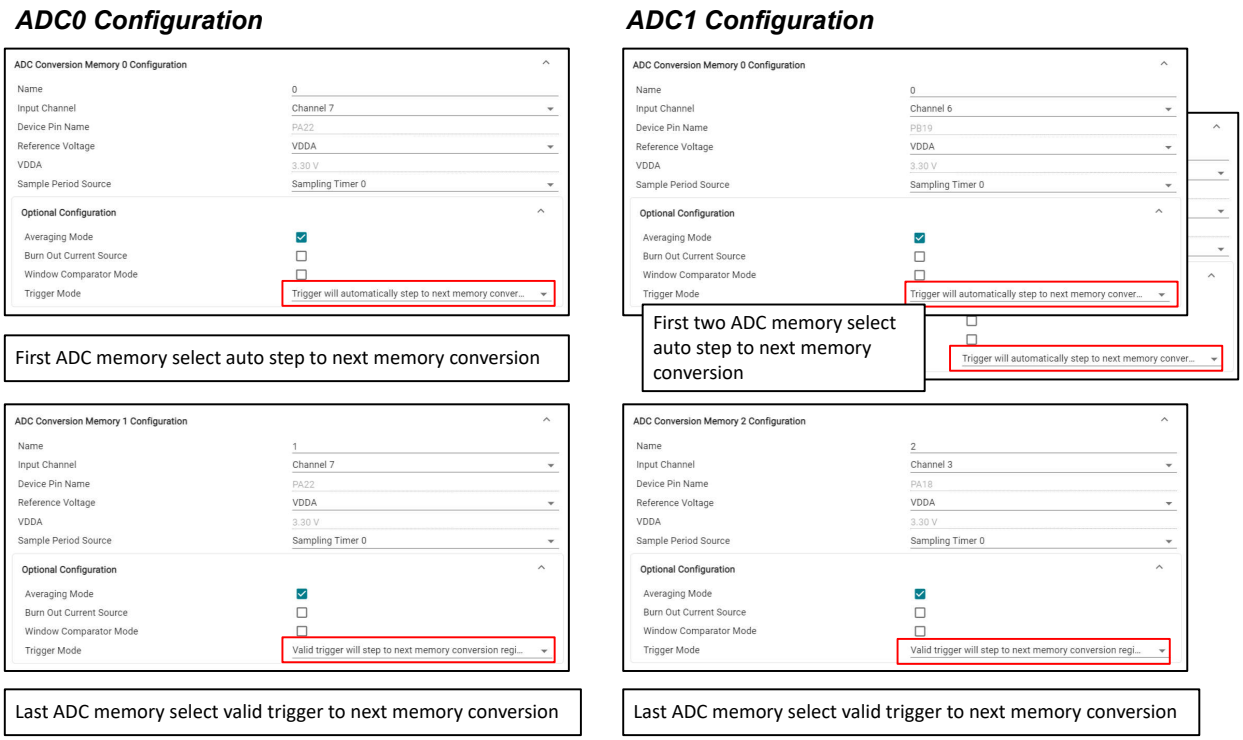


Figure 8-24. ADC Trigger Mode Setting in Three or Dual Shunt Configuration

8.3.2.5.2 Single Shunt Configuration

For single shunt configuration, FOC application converts phase currents two times in one PWM period and triggers the interrupt at the second time. The ADC instance used for phase current sampling must set and only set the **first** and **last** ADC memory Index with Valid Trigger mode. So that in every PWM period cycle, PWM module generates two events to ADC module according to space vector state, and events trigger ADC to sample and convert bus current in sequence. The following figure shows the SysConfig setting of ***sensorless-foc_DRV8329*** project.

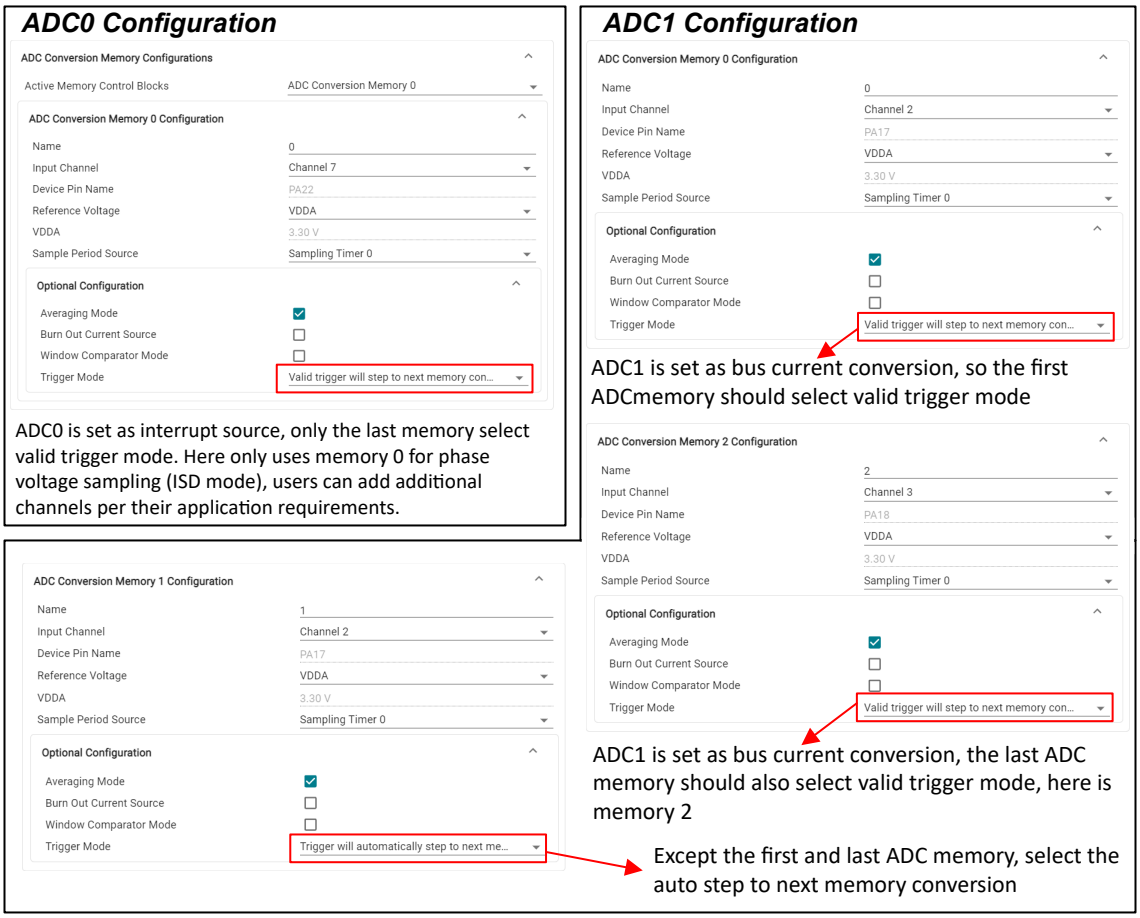


Figure 8-25. ADC Trigger Mode Setting in Single Shunt Configuration

Note

If using the ISD function, the two phase voltage channels should be configured to an ADC instance that does not sample the bus current; otherwise, it will not be possible to sample all three phase voltages simultaneously.

8.3.3 GPIO Pin

The GPIO used in FOC project mainly includes below parts:

- HALL_GPIO_IN: Hall input signal, detailed in Section 8.3.4
- FOC_GPIO_IN: to obtain direction and braking command input
- FOC_GPIO_OUT: to control the gate driver power or indicate FOC fault
- TST: test pins, not applied for FOC algorithm

Refer to Section 8.3.4 if users want to reserve the GPIO function and assign new PINMUX. If users want to remove the GPIO pins, follow the steps below:

1. Delete the GPIO Pin in SysConfig

2. Remove the related API functions which use the related GPIO macros in ***focHALInterface.c*** file, shown in the following figure.

```
927 void HAL_ClearNFault()
928 {
929 //    DL_GPIO_clearPins(FOC_GPIO_NFAULT_PORT, FOC_GPIO_NFAULT_PIN);
930 }
931
932 void HAL_SetNFault()
933 {
934 //    DL_GPIO_setPins(FOC_GPIO_NFAULT_PORT, FOC_GPIO_NFAULT_PIN);
935 }

1003 _Bool HAL_getDirPinStatus(void)
1004 {
1005     return 0; //DL_GPIO_readPins(FOC_GPIO_DIR_PORT, FOC_GPIO_DIR_PIN)?
1006               //          HAL_GPIO_HIGH : HAL_GPIO_LOW;
1007 }
1008
1009 _Bool HAL_getBrakePinStatus(void)
1010 {
1011     return 0; //DL_GPIO_readPins(FOC_GPIO_BRAKE_PORT, FOC_GPIO_BRAKE_PIN)?
1012               //          HAL_GPIO_HIGH : HAL_GPIO_LOW;
1013 }
```

Figure 8-26. Remove Preset GPIO Function

8.3.4 HALL Module

In Sensored FOC only, the three Hall Sensor input signals are to be fed through general-purpose input/output (GPIO) inputs and Events are generated based on the changes in GPIO levels to a specific timer (capture mode). Using SysConfig to overwrite the HALL pins according to your hardware design.

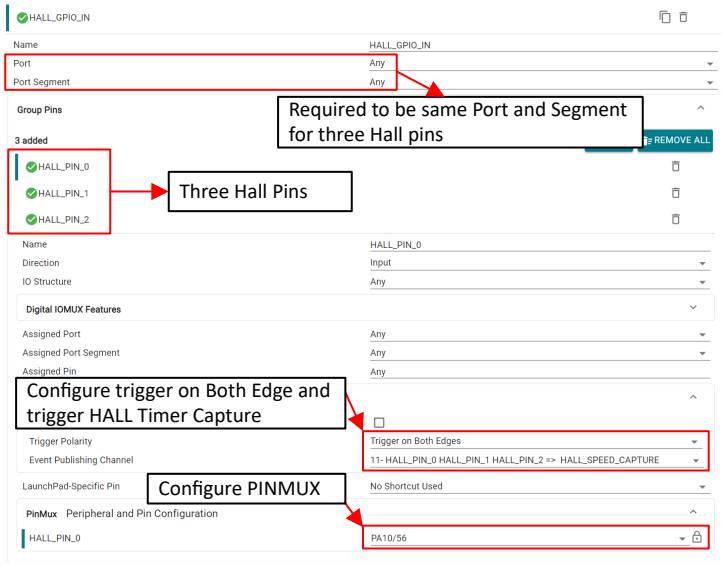


Figure 8-27. Modify HALL Pin

8.3.5 UART Module

The FOC project uses UART to communicate with GUI to support motor tuning without CCS IDE. The DMA is used to improve UART transmission efficiency and not impacted by FOC ISR.

Follow the steps below to remove the UART module:

1. Delete the UART module and CRC module in SysConfig
2. Exclude the uart_comm.c file from build
3. Remove the related API functions which use the related UART or DMA macros in main.c and ISR.c files

Figure 8-28 shows example of removing UART module. Then users can then add UART module back and apply their own communication protocol.

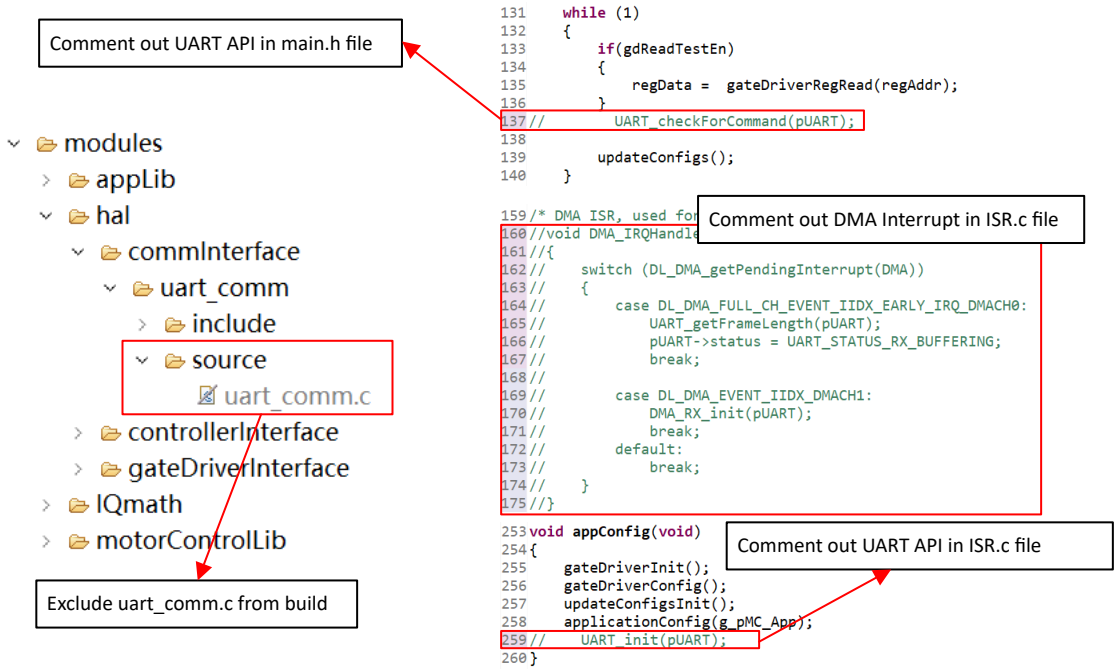


Figure 8-28. Remove UART Module

8.3.6 DAC12 Module

FOC application enables DAC12 module to track the variable in real-time running at MCU pin by default. Follow the steps below to remove the DAC12 module:

1. Delete the DAC12 module in SysConfig.
2. Comment out the code of DAC output generation in FOC_ADC_ISR (located in ***ISR.c*** file).

```
//    if(pUserCtrlRegs->dacCtrl.dacEn != 0)
//    {
//        if(pUserCtrlRegs->dacCtrl.dacScalingFactor != 0)
//        {
//            dacWriteData = _IQmpy_mathacl(
//                        *((uint32_t *)pUserCtrlRegs->dacCtrl.dacOutAddr),
//                         pUserCtrlRegs->dacCtrl.dacScalingFactor);
//            /* Adding offset value to half of reference voltage */
//            dacWriteData = _IQtoIQ12((dacWriteData >> 1) + _IQ(0.5));
//        }
//        else
//        {
//            dacWriteData = *((uint32_t *)pUserCtrlRegs->dacCtrl.dacOutAddr);
//            if((pUserCtrlRegs->dacCtrl.dacShift)>=0)
//            {
//                dacWriteData <<= pUserCtrlRegs->dacCtrl.dacShift;
//            }
//            else
//            {
//                dacWriteData >>= (-1*pUserCtrlRegs->dacCtrl.dacShift);
//            }
//        }
//        DL_DAC12_output12(DAC0, dacWriteData);
//    }
}
```

Figure 8-29. Remove DAC12 Module

8.3.7 IPD Module (Capture Timer)

A timer is enabled in capture mode for IPD pulse time counting. If the IPD function is not used, follow the steps below to delete the CAPTURE module.

1. Delete the CAPTURE module in SysConfig.

2. Comment out the macro __IPD_ENABLE in main.h file.
3. Comment out the IPD_ADC_init() API (located in focHALInterface.c file).

main.h

```
159 /*! @brief DRV8316 has inverting isense */
160 #define _INVERT_ISEN
161 /*! @brief IPD feature is enabled */
162 //#define __IPD_ENABLE
163 /*! @brief DRV8323 propagation delay */
164 #define DRV8323_PROPAGATION_DELAY_nS                100
165 /*! @brief DRV8316 minimum on time (rise time + settling time) */
```

focHALInterface.c

```
1021 static void IPD_ADC_init(ADC12_Regs *adc12Inst, uint32_t chansel)
1022 {
1023 //    DL_ADC12_setClockConfig(adc12Inst, (DL_ADC12_ClockConfig *) &gADC_IPDClockConfig);
1024 //    DL_ADC12_initSingleSample(adc12Inst,
1025 //        DL_ADC12_REPEAT_MODE_ENABLED, DL_ADC12_SAMPLING_SOURCE_AUTO, DL_ADC12_TRIG_SRC_SOFTWARE,
1026 //        DL_ADC12_SAMP_CONV_RES_12_BIT, DL_ADC12_SAMP_CONV_DATA_FORMAT_UNSIGNED);
1027 //    DL_ADC12_configConversionMem(adc12Inst, DL_ADC12_MEM_IDX_0,
1028 //        chansel, DL_ADC12_REFERENCE_VOLTAGE_VDDA, DL_ADC12_SAMPLE_TIMER_SOURCE_SCOMP0, DL_ADC12_AVER
1029 //        DL_ADC12_BURN_OUT_SOURCE_DISABLED, DL_ADC12_TRIGGER_MODE_AUTO_NEXT, DL_ADC12_WINDOWS_COMP_MO
1030 //    DL_ADC12_setPowerDownMode(adc12Inst,DL_ADC12_POWER_DOWN_MODE_MANUAL);
1031 //    DL_ADC12_setSampleTime0(adc12Inst,2);
1032 //    DL_ADC12_setPublisherChanID(adc12Inst,FOC_IPD_EVENT_CH);
1033 //#ifdef _NONINVERT_ISEN
1034 //    DL_ADC12_enableEvent(adc12Inst,DL_ADC12_EVENT_WINDOW_COMP_HIGH);
1035 //#else
1036 //    DL_ADC12_enableEvent(adc12Inst,DL_ADC12_EVENT_WINDOW_COMP_LOW);
1037 //#endif
1038 //    DL_ADC12_enableConversions(adc12Inst);
1039 }
```

Figure 8-30. Remove IPD Module

8.4 Verification for Customized Board

Once the FOC project has been successfully migrated to match your hardware configuration, follow these steps to verify software functionality.

ADC Interrupt Verification

The ADC interrupt serves as the critical task for the FOC application, responsible for reading motor phase currents and executing the FOC algorithm. Users can set a breakpoint (or GPIO toggle) in FOC_ADC_ISR() to verify that the ADC interrupt trigger frequency aligns with the expected FOC loop frequency.

In IDLE state, ADC memory index registers are periodically updated with conversion results that reflect the hardware circuit's baseline conditions. These values establish proper reference readings prior to motor operation. For instance, applying 1.65V to the ADC phase current input channel yields an ADC memory index register value of approximately 2047.

Check the following configurations if the ADC interrupt is not triggered:

- Add FOC_ISR_ADC1 macro if use ADC1 interrupt
- Verify the proper PWM/Timer trigger event is set for ADC conversion
- Verify one proper ADC interrupt trigger event is set for ADC module

Check the following configurations if the ADC interrupt occurs at an unexpected frequency:

- Verify proper ADC trigger mode is set for each ADC memory index
- Verify one proper ADC interrupt trigger event is set for ADC module
- Sensorless/Universal FOC supports max 10kHz ADC interrupt and Sensored FOC supports max 16kHz ADC interrupt

Check the following configurations if the ADC conversion value is unexpected:

- Verify motor phase A/B/C current input signal (to ADC channel) is correct
- Verify the proper PWM/Timer trigger event is set for ADC conversion

PWM Output and Phase Current Input Mapping

The FOC application uses U/V/W to represent motor phases A/B/C respectively. Although there are no restrictions on maintaining the default relationships, users must ensure proper alignment between PWM outputs and phase current inputs.

For example, if your hardware circuit assigns:

- FOC_PWMA0_U_IDX -> Motor Phase B
- FOC_PWMA0_V_IDX -> Motor Phase A

Then you must configure the motor phase current accordingly:

- Motor Phase B ADC Input Channel -> ADCx_CURRENT_U_CH
- Motor Phase A ADC Input Channel -> ADCx_CURRENT_V_CH

This ensures the PWM-to-Current relationship remains consistent. If mistached, the current close loop function fails when running FOC application.

Gate Driver Output Verification

Follow the steps below to verify gate driver output:

- Disconnect the motor
- Use a scope to observe the three-phase PWM output
- Configure the FOC application to run in Force Align Mode with PWM Loop (Section 7.7.1.1.1.2)
- Set a non-zero value to FORCE_VQ_CURRENT_ LOOP_DIS or FORCE_VQ_CURRENT_ LOOP_DIS
- Set a non-zero value to speedInput to start FOC statemachine
- Validate that three-phase PWM outputs follow the SVPWM pattern (Figure 7-19)
- Validate that the three PWM outputs follow the defined sequence with motor phases

Current Sensing Circuit Verification

Follow the steps below to verify motor phase current sensing circuit:

- Connect the motor
- Configure the DC voltage source output and set the current limit to an appropriate value, typically below the motor's rated current specification
- Use a current probe to monitor one motor phase current and a voltage probe to monitor this motor phase current signal at the ADC input channel
- Configure the FOC application to run in Force Align Mode with Current Loop (Section 7.7.1.1.1.1)
- Set appropriate align time and align current parameters
- Set a non-zero value to speedInput to start FOC statemachine
- Validate that the motor phase current follows the Align Mode pattern (Figure 7-17)
- Validate that the motor phase current amplitude match the align mode parameters

Check below configurations if the align current verification fails:

- Check motor phase shunt resistance is proper for ADC sampling
- Check the motor phase current signal at ADC input channel (scaled) match the current probe signal
- Check PWM Output and Phase Current Input Mapping
- Check Current Sensing Type and Current Sensing Method
- Check ADC Channel Mapping with used shunt configurations
- Check current loop PI parameters (See Section 7.3.2)

Current Control Loop Verification

Follow the steps below to verify current control loop:

- Validate the current sensing circuit works fine
- Use a current probe to monitor one motor phase current and a voltage probe to monitor this motor phase current signal at the ADC input channel

- Configure the FOC application to run in Force Open Loop (Section 7.7.1.2.2)
- Set appropriate align mode parameters (typically set align current to 0h for unloaded motors)
- Set proper open loop current and acceleration rate parameters (typically set open loop current to 0h for unloaded motors)
- Set handoff threshold to 20-30% of motor rated speed
- Configure current control loop parameters, recommend to start with auto calculated parameters (Section 7.3.2)
- Set a non-zero value to speedInput to start FOC statemachine
- Validate that the motor phase current follows the Open Loop Mode pattern (Figure 7-21) and is a clean sine wave
- Validate that the motor phase current amplitude and frequency match the open loop parameters

Check the following configurations if the open loop verification fails:

- Check the motor phase current signal at ADC input channel (scaled) match the current probe signal
- Check current loop PI parameters (See Section 7.3.2)
- Refer to Track Real-time Variable to monitor the ADC raw data and verify whether the raw ADC data matches a sine waveform.

Note

Once hardware functionality is verified through these steps, refer to Tune the LVBLDC Motor to begin comprehensive motor tuning with all available features.

9 Frequently Asked Questions (FAQs)

9.1 MSPM0 Failed to Connect

The DRV8316 FOC project generates POR reset if MCU failed to communicate with DRV8316, which might cause MCU in a periodically reset scenario and break SWD connection. Connect DRV8316 EVM board to voltage source with a suitable range to fix it.

If this does not help or the issue occurs on other DRV EVM boards, follow the steps below to recover:

1. Press S1 button (PA18) and then repower the MSPM0 LaunchPad.
2. Download the new firmware to MSPM0 device.

9.2 Spin the Motor in Hardcode

FOC application provides a status bit [***pUserCtrlRegs*** ->algoDebugCrtl2.b.updateConfigs] that gives user the status of configuration updates. This bit is reset every 200mS when the tuning configurations are updated by algorithm and motor is not spinning. To make sure that User Configurations are reflected in algorithm, users can set this bit after required tuning configurations are made and wait for this status bit to reset before giving the speed command.

Below shows the reference code flow:

```
  ... /* Set tuning parameter */
  pUserCtrlRegs->algoDebugCtrl2.b.updateConfigs = 0x1;
  /* After all tuning parameters updated to FOC algorithm, this bit reset to zero */
  while(pUserCtrlRegs->algoDebugCtrl2.b.updateConfigs){
    updateConfigs();
}
  ... /* Set speedInput as non-zero value to spin motor */
```

9.3 Reduce 1x ADC Pin for Simultaneously Sampling

In FOC applications, the simultaneous sampling feature in three shunt current sensing method is enabled with four ADC channels.

For MSPM0GX50X series, there are two unique ADC channels that are mapped to ADC0 and ADC1 instance both as shown below. Select one of these channels for phase B current sampling.

CHANNEL[0:7]	SIGNAL NAME[2]		CHANNEL[8:15]	SIGNAL NAME[1] [2]	
	ADC0	ADC1		ADC0	ADC1
0	A0_0	A1_0 / DAC_OUT[4]	8	A1_7[3]	A0_7[3]
1	A0_1	A1_1	9	-	-
2	A0_2	A1_2	10	-	-
3	A0_3	A1_3	11	*Temperature Sensor*	-
4	A0_4	A1_4	12	A0_12	*Temperature Sensor*
5	A0_5	A1_5	13	*OPA0 output*	*OPA1 output*
6	A0_6	A1_6	14	*GPAMP output*	*GPAMP output*
7	A0_7	A1_7	15	*Supply/Battery Monitor*	*Supply/Battery Monitor*

Figure 9-1. MSPM0GX50X ADC Channel Mapping Table

Refer to device's specific data sheet to determine whether this dual mapping ADC feature is enabled.

9.4 Tune Real-time Control Parameter

Setting ***pUserCtrlRegs*** ->algoDebugCrtl2.b.updateSysParams as 1b to enable dynamically update of System Parameters every 200ms, such as PI gains of Speed/Torque/Current loops to tune for required performance. FOC application by default enabled parameter dynamically updating method for users debug activity.

9.5 Track Real-time Variable

9.5.1 DAC12 Output

32-bit algorithm variables can be output in real time from the MCU through the DAC12 module. DAC12 output is enabled by setting ***pUserCtrlRegs*** ->dacCtrl.dacEn as 1b. The DAC12 in MSPM0 is a 12-bit DAC module, thus a scaling needs to be applied before output. Follow the steps below to configure DAC12 output:

1. For variables in global IQ format (IQ27), refer to Equation 48. Setting the pUserCtrlRegs->dacCtrl.dacScalingFactor to IQ(1.0) enables DAC12 output to represent a data of IQ(1.0) to IQ(-1.0) in between 0V and 3.3V. To represent the data exceeding the value IQ(1.0) use higher dacScalingFactor. For example, to represent data from IQ(2.0) to IQ(-2.0), set the dacScalingFactor to IQ(0.5).

$$DAC\_OUTPUT[V]=(VARIABLE\_VALUE[IQ\,format]\times SCALING\_FACTOR+1)\times 1.65V \quad (48)$$

2. For output of any IQ format other than IQ27, users can shift the variable left or right to bring the data into a 12-bit range before output. This mode is selected by setting dacScalingFactor to 0.
   a. If variable value is less than a 12-bit value, set pUserCtrlRegs->dacCtrl.dacShift to positive, the DAC output follows Equation 49.
   b. If variable value is greater than a 12-bit value, set dacShift to negative, the DAC output follows Equation 50.

$$DAC\_OUTPUT[V]=(VARIABLE\_VALUE[IQ\,format]<<DAC\_SHIFT)\times 3.3V \quad (49)$$

$$DAC\_OUTPUT[V]=(VARIABLE\_VALUE[IQ\,format]>>DAC\_SHIFT)\times 3.3V \quad (50)$$

Note

Settings dacEn as 1b feeds the variable output to the DAC registers, but user needs to enable the DAC peripheral in TI SysConfig for the DAC peripheral to function. Also make sure the DAC output pin is not loaded by any other peripheral or DRV board.

Table 9-1 lists the FOC motor control variable addresses for real-time tracking, with updates occurring every FOC loop. In contrast, variables in **pUserStatusRegs** update every 1ms and therefore cannot display real-time changes.

Table 9-1. Address Table for DAC Monitoring

Parameter	Variable Address / Name			IQ Type
	Sensorless FOC	Universal FOC	Sensored FOC	
A phase current	g_pMotorInputs->current.iabc.a			IQ27
B phase current	g_pMotorInputs->current.iabc.b			IQ27
C phase current	g_pMotorInputs->current.iabc.c			IQ27
A phase current raw ADC value	g_pMotorInputs->current.iabcRaw.a			IQ11
B phase current raw ADC value	g_pMotorInputs->current.iabcRaw.b			IQ11
C phase current raw ADC value	g_pMotorInputs->current.iabcRaw.c			IQ11
A phase voltage	g_pMotorInputs->voltage.vabc.a			IQ27
B phase voltage	g_pMotorInputs->voltage.vabc.b			IQ27
C phase voltage	g_pMotorInputs->voltage.vabc.c			IQ27
A phase voltage raw ADC value	g_pMotorInputs->voltage.vabcRaw.a			IQ12
B phase voltage raw ADC value	g_pMotorInputs->voltage.vabcRaw.b			IQ12
C phase voltage raw ADC value	g_pMotorInputs->voltage.vabcRaw.c			IQ12
D axis current	0x20200760	g_pMC_App->foc.idq.d		IQ27
Q axis current	0x20200764	g_pMC_App->foc.idq.q		IQ27
D axis voltage	0x20200768	g_pMC_App->foc.vdq.d		IQ27
Q axis voltage	0x2020076C	g_pMC_App->foc.vdq.q		IQ27

Table 9-1. Address Table for DAC Monitoring (continued)

Parameter	Variable Address / Name			IQ Type
	Sensorless FOC	Universal FOC	Sensored FOC	
D axis Filtered BEMF	0x20200BEC	g_pMC_App->angleTrackingPLLEstim.EdqFilt.d	NA	IQ27
Q axis Filtered BEMF	0x20200BF0	g_pMC_App->angleTrackingPLLEstim.EdqFilt.q	NA	IQ27
Estimated motor velocity filtered	0x20200C0C	g_pMC_App->angleTrackingPLLEstim.velocityFilt	g_pMC_App->foc.hallObj.hallEstimVelocityFilt	IQ27
Estimated rotor angle	0x20200C14	g_pMC_App->angleTrackingPLLEstim.fluxAngle	g_pMC_App->foc.hallObj.hallEstimFluxAngle	IQ27
Power Feedback	0x20200940	g_pMC_App->foc.closeLoop.PowerFeedback		IQ27
SVM output duty A phase	0x20200730	g_pMC_App->foc.svm.Dabc.a		IQ0
SVM output duty B phase	0x20200734	g_pMC_App->foc.svm.Dabc.b		IQ0
SVM output duty C phase	0x20200738	g_pMC_App->foc.svm.Dabc.c		IQ0

Note

Variable address might vary in different FOC algorithm versions. Refer to latest MSPM0 Sensorless FOC Tuning Guide for variable address of newer Sensorless FOC algorithm versions.

Figure 9-2 and Figure 9-3 show an example to output estimated rotor angle variable at DAC12 output pin.

```
161    /* DAC12 output */
162    pUserCtrlRegs->dacCtrl.dacEn              = 1;
163    pUserCtrlRegs->dacCtrl.dacScalingFactor   = _IQ(2.0);     //
164    pUserCtrlRegs->dacCtrl.dacOutAddr         = 0x20200C14;   // Rotor angle
```

Figure 9-2. Register Setting for DAC12 Module

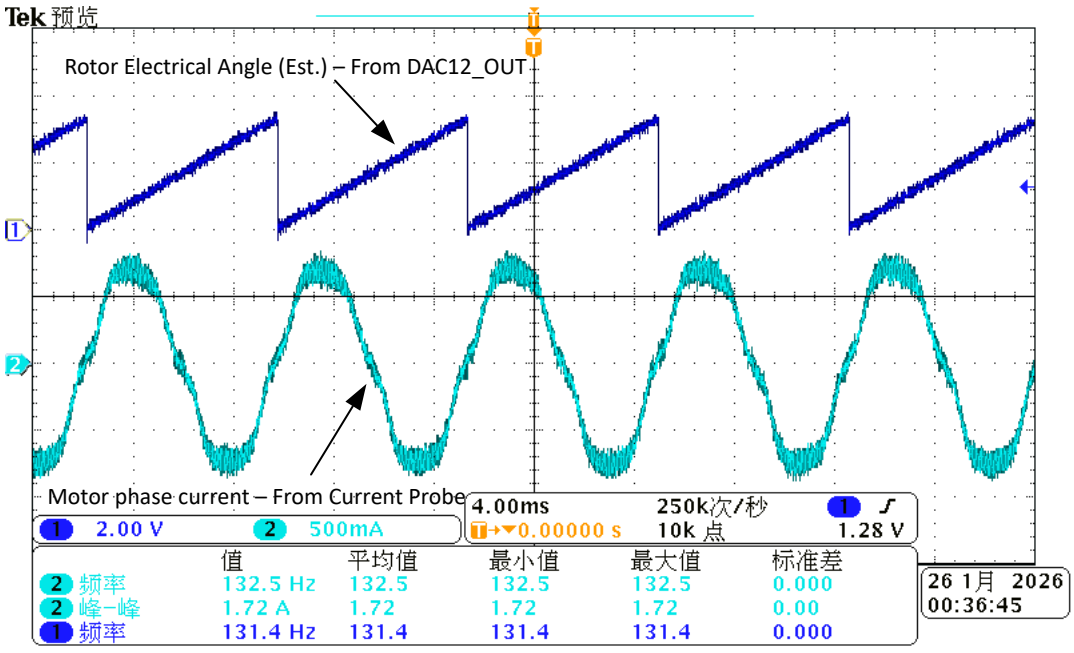


Figure 9-3. Output Estimated Rotor Angle by DAC12 Module

Note

For estimated rotor angle, IQ(1.0) represents 32 electrical angle cycles to avoid the overflow in algorithm.

9.5.2 J-Scope Tool

Using J-Scope to track multiple variables simultaneously in run-time. Follow the steps below:

1. Assign the required variables (detailed in Table 8-1) to preset global variables for monitoring.
   a. Add attributes for the preset variables: __attribute ((used))
2. Open the J-Scope and Import the .out (or .elf) file of FOC project.
3. Add the preset variables in Step1 to J-Scope observation panel.
4. Run the FOC project and track the variables in real-time.
5. There requires a UART (or other communication interfaces) to dynamically control the motor, or a auto spin mechanism in application code.

Figure 9-4 and Figure 9-5 show an example to monitor multiple variables simultaneously in run-time.

Global Vairable

```
177 volatile __attribute ((used)) int32_t gPhaseCurrentA = 0;
178 volatile __attribute ((used)) int32_t gPhaseCurrentB = 0;
179 volatile __attribute ((used)) int32_t gPhaseCurrentC = 0;
180 volatile __attribute ((used)) int32_t gEstimatedMotorSpeed = 0;
181 volatile __attribute ((used)) int32_t gEstimatedRotorAngle = 0;
```

ISR.c

```
void FOC_ADC_ISR(void)
225     /* Update variable for J-scope observation in @FOC_ADC_ISR() */
226     gPhaseCurrentA = *((int32_t*)(0x20200614));
227     gPhaseCurrentB = *((int32_t*)(0x20200618));
228     gPhaseCurrentC = *((int32_t*)(0x2020061C));
229
230     gEstimatedMotorSpeed = *((int32_t*)(0x20200C0C));
231     gEstimatedRotorAngle = *((int32_t*)(0x20200C14));
232 }
```

Figure 9-4. Global Variable Setting for J-Scope Tracking

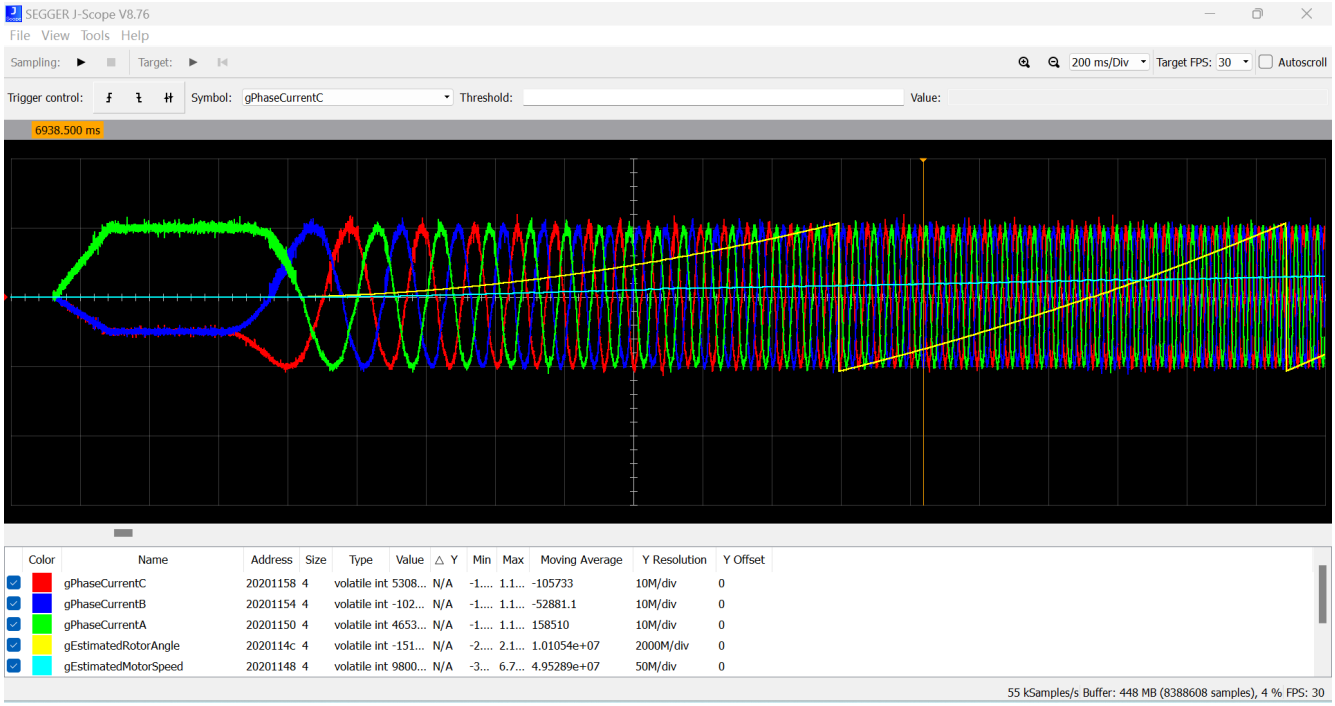


Figure 9-5. Monitor Multiple Variables in J-Scope

10 Summary

This document provides users with a comprehensive guide to MSPM0 FOC motor control solutions, covering architecture, hardware, and software components. It includes a detailed reference workflow with step-by-step instructions and practical examples for setting up debug environments, tuning motors, and migrating hardware configurations.

11 References

- Sensorless FOC SDK User's Guide
- Universal FOC SDK User's Guide
- Sensored FOC SDK User's Guide
- Sensorless FOC EVM Board Connection Guide
- Sensored FOC EVM Board Connection Guide
- UART Communication User Guide
- MSPM0 Sensorless FOC Tuning Guide
- MSPM0 Universal FOC Tuning User's Guide
- MSPM0 Sensored FOC Tuning Guide
- Sensorless-FOC for PMSM With Single DC-Link Shunt
- Sensorless Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors

12 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

DATE	REVISION	NOTES
March 2026	*	Initial Release

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025