

# 250W motor inverter reference design with GaN IPM DRV7308



## Description

This reference design illustrates a 250W high efficiency motor inverter without heat sink based on GaN IPM DRV7308, also demo a low standby power design with UCC28911. This reference design implements sensorless FOC motor control with 3-phase PMSM with FAST™ or eSMO observer. This reference design supports both the C2000™ MCU and MSPM0 microcontroller daughter-board on the same motherboard. The hardware, GUI software and firmware are available and ready-to-use to help to accelerate development time to market.

## Resources

[TIDA-010273](#)

[DRV7308, TMS320F2800137](#)

[MSPM0G1507, UCC28911](#)

[TLV9062, ISO6721, TLV74033](#)

[C2000WARE-MOTORCONTROL-SDK](#)

[MSPM0-SDK](#)

Design Folder

Product Folder

Product Folder

Product Folder

Tool Folder

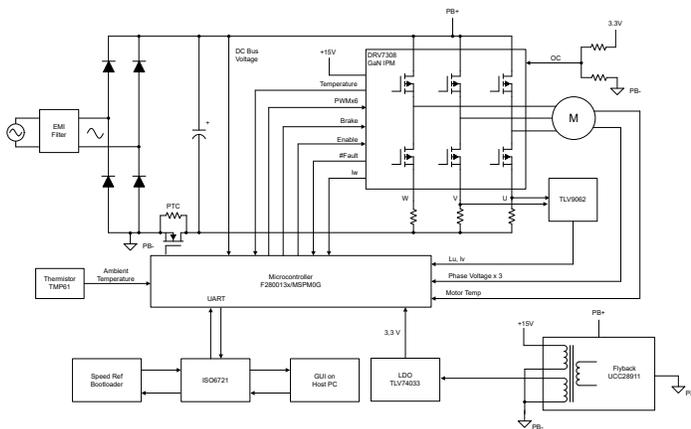
Tool Folder

## Features

- Peak efficiency > 99%
- No heat sink needed for up to 250W inverter stage, 15kHz switching frequency
- Wide operating voltage input range: 165 to 265 VAC, 50|60Hz
- 80mm × 55mm compact board size
- Low standby power design
- Modular design with either C2000 or MSPM0 controller daughter-board on the same power motherboard
- Sensorless Field Oriented Control (FOC) motor control, supports both FAST and eSMO observer, torque compensation, and automatic field weakening control
- User-friendly graphical user interface (GUI) to control, identify, and monitor the motor

## Applications

- [Refrigerator and freezer](#)
- [Heat pump](#)
- [Air conditioner indoor unit](#)
- [Cooker hood](#)
- [Washer and dryer](#)
- [Dishwasher](#)
- [Residential and living fan](#)
- [Mixer blender food processor](#)
- [Air conditioner outdoor unit](#)



## 1 System Description

Motor control for major appliances or similar applications today must meet a growing list of demands on lower cost, smaller size and higher energy efficiency. Magnet Synchronous Motors (PMSM) are becoming increasingly popular in major appliance applications.

This reference design provides a single 250W inverter mother board with either the TMS320F2800137 and MSPM0G1507 daughter-card for control, making this reference design convenient for users to evaluate both the C2000 and MSPM0 series microcontroller on the same hardware platform.

Software supports both FAST and eSMO observer, so performance of the two devices can be compared. A user-friendly GUI also helps identify the motor, as well as tune the control parameters, thus accelerating the development time.

### 1.1 Terminology

<b>PMSM</b>	Permanent Magnet Synchronous Motor
<b>BLDC</b>	Brushless Direct Current
<b>BEMF</b>	Back Electromotive Force
<b>PWM</b>	Pulse Width Modulation
<b>FET, MOSFET</b>	Metal Oxide Semiconductor Field Effect Transistor
<b>IGBT</b>	Insulated Gate Bipolar Transistor
<b>RMS</b>	Root Mean Square
<b>MTPA</b>	Maximum Torque Per Ampere
<b>FWC</b>	Field Weakening Control
<b>FOC</b>	Field Oriented Control
<b>HVAC</b>	Heating, Ventilation, and Air Conditioning
<b>ESMO</b>	Enhanced Sliding-Mode Observer
<b>PLL</b>	Phase Locked Loop
<b>FAST</b>	Flux, Angle, Speed and Torque observer

## 1.2 Key System Specifications

The TIDA-010273 specifications are listed in [Table 1-1](#).

**Table 1-1. Key System Specifications**

Parameters	TEST CONDITIONS	MIN	NOM	MAX	UNIT
SYSTEM INPUT CHARACTERISTICS					
Input Voltage ( $V_{INAC}$ )	–	165	230	265	VAC
Input Frequency ( $f_{LINE}$ )	–	47	50	63	Hz
No Load Standby Power ( $P_{NL}$ )	$V_{INAC} = 230V, I_{out} = 0A$	–	0.12	–	W
Input Current ( $I_{IN}$ )	$V_{INAC} = 230V, I_{out} = I_{MAX}$	–	3	–	A
MOTOR INVERTER CHARACTERISTICS					
PWM switching frequency ( $f_{SW}$ )	–	–	15	30	kHz
Rated output power ( $P_{OUT}$ )	$V_{INAC} = nom$	–	250	–	W
Output current ( $I_{RMS}$ )	$V_{INAC} = nom$	–	3	–	A
Inverter efficiency ( $\eta$ )	$V_{INAC} = nom, P_{OUT} = nom$	–	–	99	%
Motor electrical frequency (f)	$V_{INAC} = min\ to\ max$	20	200	500	Hz
Fault protections	Overcurrent, stall with recovery, undervoltage, overvoltage				
Drive control method and features	Sensorless-FOC with three or single shunt resistors for current sensing				
SYSTEM CHARACTERISTICS					
Built-in auxiliary power supply	$V_{INAC} = min\ to\ max$	15V $\pm$ 10%, 100mA , 3.3V $\pm$ 10%, 300mA			
Operating ambient	Open frame	–10	25	55	$^{\circ}C$
Board size	Length $\times$ width $\times$ height	85mm $\times$ 55mm $\times$ 60mm			mm <sup>3</sup>

### WARNING

TI intends this reference design to be operated in a lab environment only and does not consider the device to be a finished product for general consumer use.

TI Intends this reference design to be used only by qualified engineers and technicians familiar with risks associated with handling high-voltage electrical and mechanical components, systems, and subsystems.

**High voltage!** There are accessible high voltages present on the board. The board operates at voltages and currents that can cause shock, fire, or injury if not properly handled or applied. Use the equipment with necessary caution and appropriate safeguards to avoid injuring yourself or damaging property.

**Hot surface!** Contact can cause burns. **Do not touch!** Some components can reach high temperatures  $> 55^{\circ}C$  when the board is powered on. The user must not touch the board at any point during operation or immediately after operating, as high temperatures can be present.

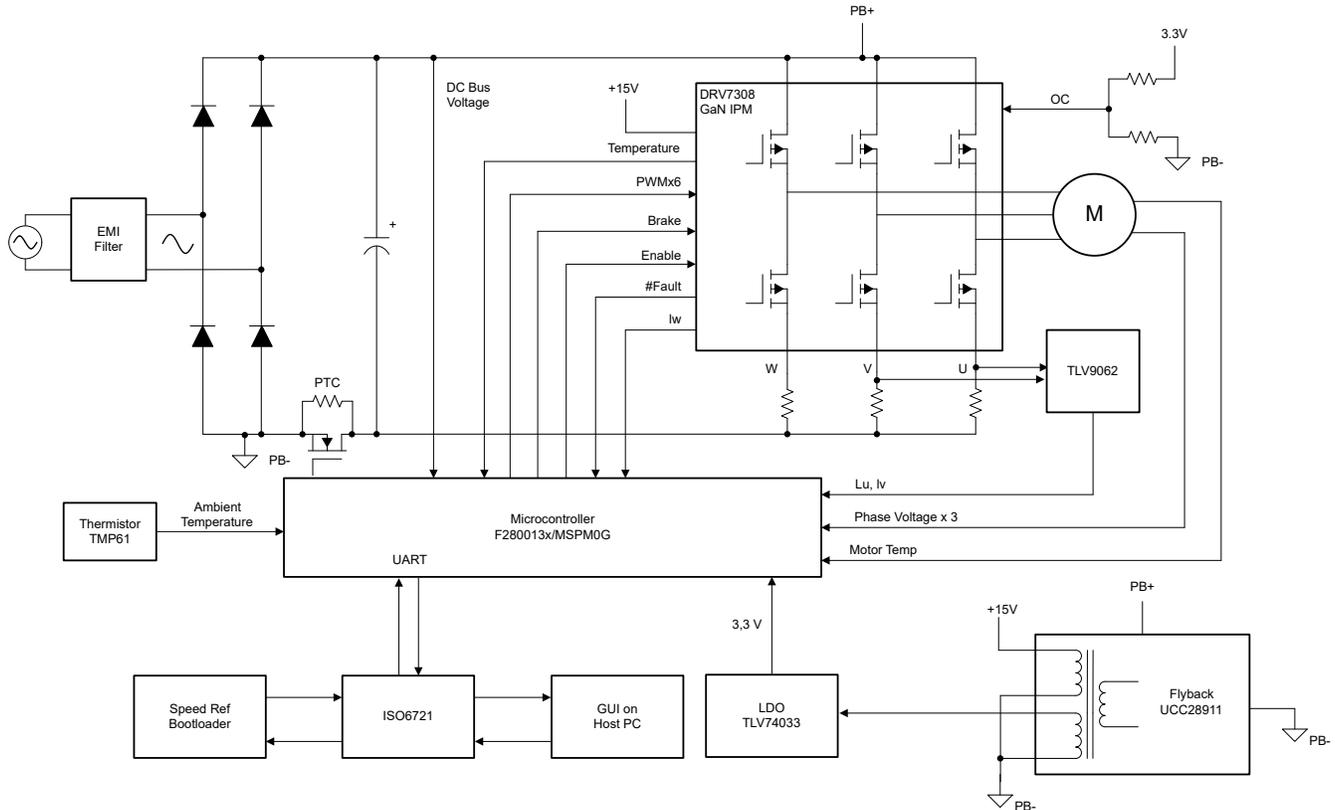
### CAUTION

Do not leave the design powered when unattended.

## 2 System Overview

### 2.1 Block Diagram

Figure 2-1 shows the block diagram of this reference design.



**Figure 2-1. TIDA-010273 250W Motor Inverter Block Diagram**

The entire system is represented in seven blocks:

- EMI filter
- Bridge rectifier
- DRV7308 GaN IPM inverter
- Auxiliary power supply
- TMS320F2800137 daughter card
- MSPM0G1507 daughter card
- Isolated UART port

### 2.2 Design Considerations

The design supports either the C2000 or MSPM0 controller for a single motor control. Highly noise-immune current and voltage sensing designs are necessary for precise motor drives. The following details the sensing and drive circuit that are used on this design. The hardware design files are available under the C2000Ware Motor Control SDK Install directory at `<install_location>\solutions\tida_010273_GaNInv\hardware`.

### 2.3 Highlighted Products

The following highlighted products are used in this reference design. Key features for selecting the devices for this reference design are revealed in the following sections. Find more details of the highlighted devices in respective product data sheets.

### 2.3.1 TMS320F2800137

The [TMS320F280013x](#) is a member of the C2000™ real-time microcontroller family of scalable, ultra-low latency devices designed for efficiency in power electronics applications. The real-time control subsystem is based on TI's 32-bit C28x digital-signal processor (DSP) core, which provides 120MHz of signal-processing performance for floating- or fixed-point code running from either on-chip flash or SRAM. The C28x CPU is further boosted by the Trigonometric Math Unit (TMU) and Cyclical Redundancy Check (VCRC) extended instruction sets, speeding up common algorithms key to real-time control systems. High-performance analog blocks are integrated on the F280013x real-time microcontroller (MCU) and are closely coupled with the processing and PWM units to provide exceptional real-time signal chain performance. Fourteen PWM channels, all supporting frequency-independent resolution modes, enable control of various power stages from a 3-phase inverter to advanced multilevel power topologies. Interfacing is supported through various industry-standard communication ports (such as SPI, three SCI|URAT, I2C, and CAN) and offers multiple pin-MUXing options for excellent signal placement.

### 2.3.2 MSPM0G1507

The [MSPM0G150x](#) microcontrollers (MCUs) are part of the highly-integrated, ultra-low-power, 32-bit MCU family from mixed-signal processing (MSP) based on the enhanced Arm® Cortex®-M0+ 32-bit core platform operating at up to 80MHz frequency. These cost-optimized MCUs offer high-performance analog peripheral integration, support extended temperature ranges from -40°C to 125°C, and operate with supply voltages ranging from 1.62V to 3.6V. The MSPM0G150x devices provide up to 128KB embedded Flash program memory with built-in error correction code (ECC) and up to 32KB SRAM with hardware parity option. The devices also incorporate a memory protection unit, seven-channel DMA, math accelerator, and a variety of high-performance analog peripherals such as two 12-bit 4MSPS ADCs, configurable internal shared voltage reference, one 12-bit 1MSPS DAC, three high-speed comparators with built-in reference digital-to-analog converters (DACs), two zero-drift, zero-crossover op amps with programmable gain, and one general-purpose amplifier. These devices also offer intelligent digital peripherals such as three 16-bit advanced control timers, three 16-bit general purpose timers, one 24-bit high-resolution timer, two windowed-watchdog timers, and one RTC with alarm and calendar mode. These devices provide data integrity and encryption peripherals and enhanced communication interfaces (four UARTs, two I2Cs, two serial-peripheral interfaces (SPIs)).

### 2.3.3 DRV7308

The [DRV7308](#) is a three phase motor driver IPM that consists of 205mΩ, 650V e-mode Gallium- Nitride (GaN) for driving three-phase BLDC motors up to 450V DC rails. These applications include field oriented control (FOC), sinusoidal current control, and trapezoidal current control of BLDC motors. The device integrates the pre-drivers for all the GaNFETs with slew rate control of phase node voltages. The device helps to achieve more than 99% efficiency for a 3-phase modulated FOC driven 250W motor drive application in a QFN 12mm x 12mm package, eliminating the need of heat sink. The device enables minimum dead time and hence ultra quiet operation even without dead time compensation.

The integrated bootstrap rectifier with current limit eliminates the need for an external bootstrap diode with related reverse recovery inefficiency and also eliminates the external current limit resistor. The DRV7308 brings out all the three low side source pins of the GaN FETs to support 3- or 2- or 1-shunt current sensing. The device integrates a 12MHz, 15V/μs operational amplifier for single shunt current sensing in FOC and trapezoidal control of BLDC motors.

### 2.3.4 UCC28911

The [UCC28911](#) are high-voltage flyback switchers that provide output voltage and current regulation without the use of an optical coupler. Both devices incorporate a 700V power FET and a controller that process operating information from the flyback auxiliary winding and power FET to provide a precise output voltage and current control. The integrated high-voltage current source for startup that is switched off during device operation, and the controller current consumption is dynamically adjusted with load. Both enable the very low stand-by power consumption.

Control algorithms in the UCC28911, combining switching frequency and peak primary current modulation, allow operating efficiencies to meet or exceed applicable standards. Discontinuous conduction mode (DCM) with valley switching is used to reduce switching losses. Built-in protection features help to keep secondary and

primary component stress levels in check across the operating range. The frequency jitter helps to reduce EMI filter cost.

### 2.3.5 TLV9062

The [TLV9062](#) is a dual-low-voltage (1.8V to 5.5V) operational amplifier (op amp) with rail-to-rail input- and output-swing capabilities. This device is a highly cost-effective design for applications where low-voltage operation, a small footprint, and high capacitive load drive are required. Although the capacitive load drive of the TLV906x is 100pF, the resistive open-loop output impedance makes stabilizing with higher capacitive loads simpler. The TLV906xS devices include a shutdown mode that allow the amplifiers to switch into standby mode with typical current consumption less than 1µA. The TLV906xS family helps simplify system design, because the family is unity-gain stable, integrates the RFI and EMI rejection filter, and provides no phase reversal in overdrive condition.

### 2.3.6 TLV74033

The [TLV740P](#) low-dropout (LDO) linear regulator is a low quiescent current LDO with excellent line and load transient performance designed for power-sensitive applications. This device provides a typical accuracy of 1%.

The TLV740P also provides inrush current control during device power up and enabling. The TLV740P limits the input current to the defined current limit to avoid large currents from flowing from the input power source. This functionality is especially important in battery-operated devices.

### 2.3.7 ISO6721B

The [ISO672xB](#) devices are high-performance, dualchannel digital isolators designed for cost sensitive applications requiring up to 3000 VRMS (D package) isolation ratings per UL 1577. These devices are also certified by VDE, TUV, CSA, and CQC. The ISO672xB devices provide high electromagnetic immunity and low emissions at low power consumption, while isolating CMOS or LVCMOS digital I/Os. Each isolation channel has a logic input and output buffer separated by TI's double capacitive silicon dioxide (SiO<sub>2</sub>) insulation barrier. The ISO6720B device has 2 isolation channels with both channels in the same direction. The ISO6721B device has 2 isolation channels with 1 channel in each direction.

Used in conjunction with isolated power supplies, these devices help prevent noise currents on data buses, such as UART, SPI, RS-485, RS-232, and CAN from damaging sensitive circuitry.

### 2.3.8 TMP6131

The [TMP61x](#) linear thermistor offers linearity and consistent sensitivity across temperature to enable simple and accurate methods for temperature conversion. The low power consumption and a small thermal mass of the device minimizes the impact of self-heating.

With built-in fail-safe behaviors at high temperatures and powerful immunity to environmental variation, these devices are designed for a long lifetime of high performance. The small size of the TMP6 series also allows for close placement to heat sources and quick response times.

## 2.4 System Design Theory

The main focus of this reference design is single motor control for refrigerator or similar appliance applications.

### 2.4.1 Hardware Design

A typical motor control board has several blocks: auxiliary power supply, inverter, current and voltage sensing, protection circuit, and a microcontroller. These design concepts are explained in this section.

#### 2.4.1.1 Modular Design

This reference design has two daughterboards and one motherboard for a modular design. One daughterboard has the MSPM0G1507 microcontroller and a two-phase current-amplifying circuit based on the two internal high-end op amps. Another daughterboard has the TMS320F2800137 microcontroller and single TLV9062 devices as phase-current amplifiers. The motherboard holds all the power devices including AC filters, rectifier, and the Intelligent Power Module (IPM). [Figure 2-2](#) and [Figure 2-3](#) show the motherboard and daughterboards.

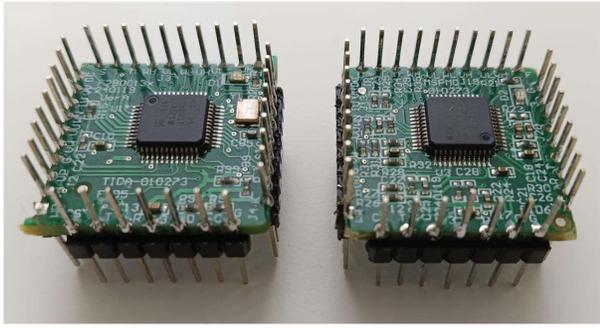


Figure 2-2. TIDA-010273 Daughterboards

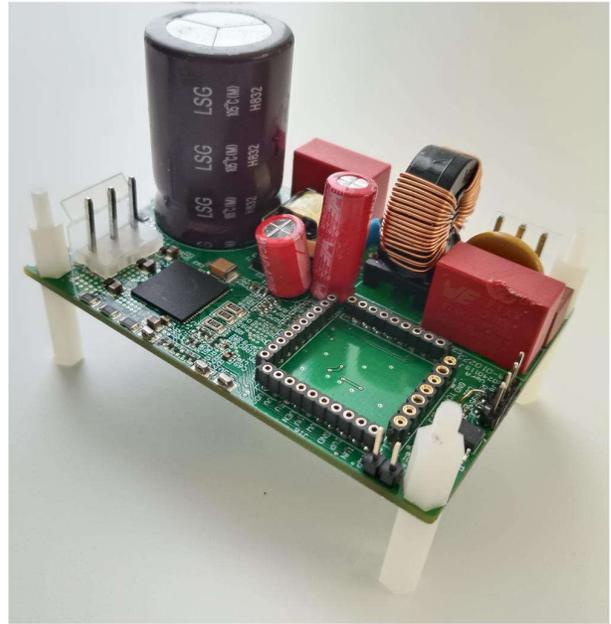


Figure 2-3. TIDA-010273 Motherboard

#### 2.4.1.2 Auxiliary Flyback Power Supply

A non-isolated UCC28911-based flyback supply provides auxiliary power supply for this reference design to delivery up to 100mA for 15VDC and 300mA for 5V. 5V is used to create 3.3V for microcontroller and related circuit by an additional TLV74033 regulator. Figure 2-4 shows the UCC28911 flyback power supply circuit.

Place R40, C35, R37, R41 and C38 closely to UCC28911, C38 is not needed normally but can be needed if R37 and R41 is too far away from UCC28911.

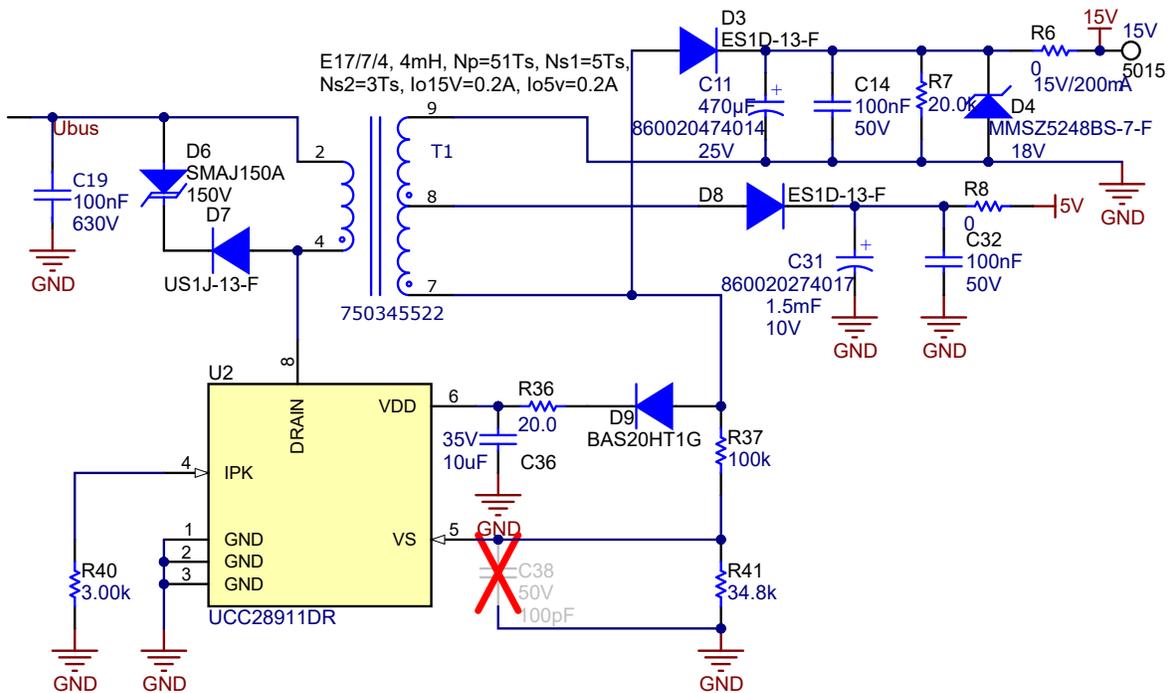


Figure 2-4. Auxiliary Flyback Power Supply Circuit

### 2.4.1.3 DC Link Voltage Sensing

The DC voltage sensing circuit is used to convert the rectified voltage signal into a low-voltage signal implemented by a low-cost resistor network as shown in Figure 2-5. The DC bus voltage can also be used to estimate AC input voltage. To achieve low standby power, the circuit is designed with a MOSFET Q3 to cut off the current on these resistors network.

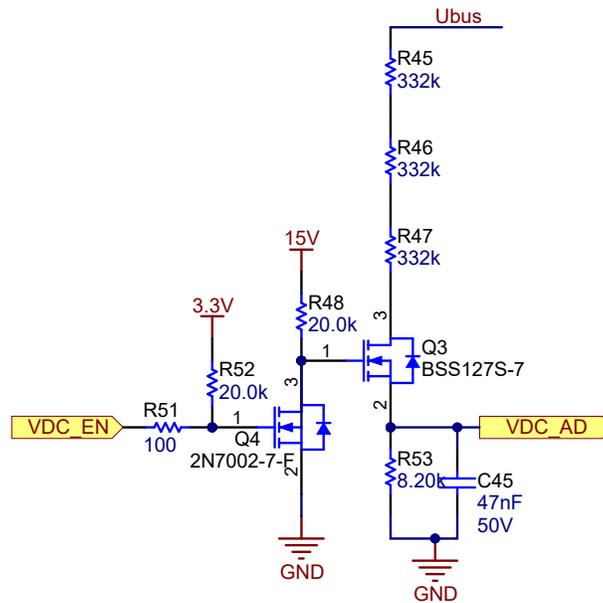


Figure 2-5. DC Bus Voltage Sensing Circuit

### 2.4.1.4 Inrush Current Protection

The reference board has a diode bridge in series with the electrolytic capacitor C67, there is a large inrush current to charge C67 when powered on. A inrush current protection circuit is designed to limit this inrush current. When board power is on, the capacitor current is limited by RT1, a 50ohm PTC resistor. After successful power-on, bias power supply provides 3.3V, then the microcontroller starts to work outputting a low level signal to pull down base of Q2 after a 2 second delay, then Q2 is turned off to turn on Q1 to provide a low resistance path for current of C67. Drian to source break down voltage of Q1 must be high enough since voltage drop on RT1 is very high, see Figure 3-40 for this voltage. And delay time can be set in software.

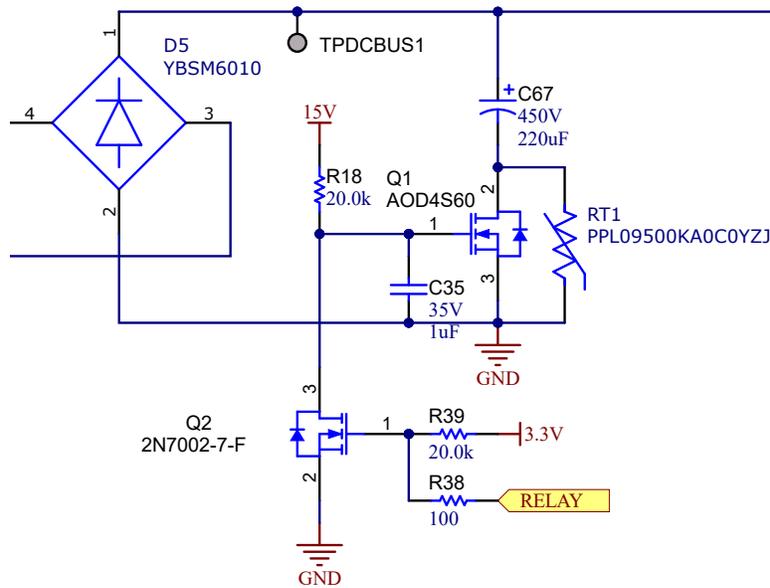


Figure 2-6. Inrush Current Protection Circuit

#### 2.4.1.5 Motor Phase Voltage Sensing

C2000 software for the TIDA-010273 supports both enhanced Slide Mode Observer (eSMO) and Flux, Angle, Speed, and Torque (FAST™) observer. The FAST observer can improve low-speed performance and reduce speed tolerance; however, FAST needs 3 motor phase voltage sensing in addition to phase current sensing. Section 2.4.2.4.2 has a detailed explanation of motor phase voltage sensing design.

#### 2.4.1.6 Motor Phase Current Sensing

The MS320F2800137 daughterboard is designed to support 1-3 phase current sensing, while the MSPM0G1507 daughterboard supports 1-2 phase current sensing. Section 2.4.2.4.1 has the details for motor phase current sensing design.

#### 2.4.1.7 Over Current Protection of DRV7308

DRV7308 provides overcurrent protection (OCP) reference voltage setting for the internal phase comparators. Figure 2-7 shows OCP reference voltage setting, then this reference voltage is used to compare to voltages on shunt resistor(R83). The exact overcurrent protection current can be calculated by below equations.

$$V_- = \frac{3.3 \text{ V}}{R_{76} + R_{77}} \times R_{77} = \frac{3.3 \text{ V}}{20 \text{ k} + 3 \text{ k}} \times 3 \text{ k} = 0.4304 \text{ V} \quad (1)$$

$$V_+ = R_{83} \times I_{\text{ocp}} = 0.1 \times I_{\text{ocp}} \quad (2)$$

$$I_{\text{ocp}} = \frac{V_-}{R_{83}} = \frac{0.4304}{0.1} = 4.304 \text{ A} \quad (3)$$

Both MS320F2800137 and MSPM01507 daughterboards can be triggered by this OCP circuit.

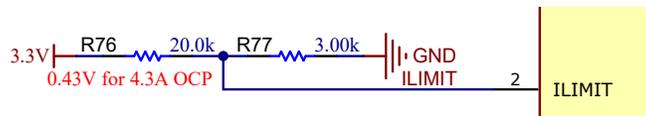


Figure 2-7. Over Current Protection Circuit

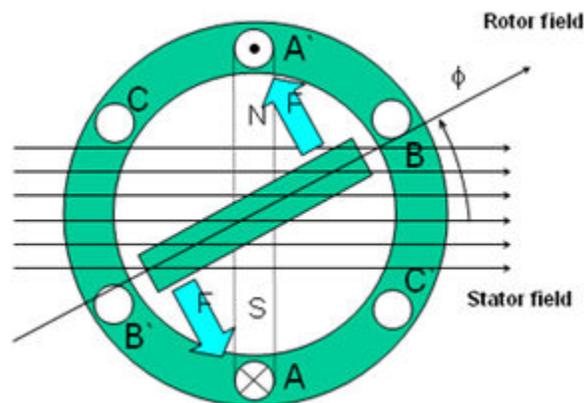
### 2.4.1.8 Internal Overcurrent Protection for TMS320F2800F137

The TMS320F2800F137 has internal window comparators (CMPSS) which can be configured to monitor three phases current, there is no any software or interrupt delay for internal comparators to trigger overcurrent to stop PWM, this quickly protects external IPM or power devices.

### 2.4.2 Three-Phase PMSM Drive

Permanent Magnet Synchronous motor (PMSM) has a wound stator, a permanent magnet rotor assembly, and internal or external devices to sense rotor position. The sensing devices provide position feedback for adjusting frequency and amplitude of stator voltage reference properly to maintain rotation of the magnet assembly. The combination of an inner permanent magnet rotor and outer windings offers the advantages of low rotor inertia, efficient heat dissipation, and reduction of the motor size.

- 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. When energized, the stator windings create a rotating electromagnetic field. To control the rotating magnetic field, the stator currents must be controlled.
- The actual structure of the rotor varies depending on the power range and rated speed of the machine. Permanent magnets are an excellent choice 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 torque. Since the stator is firmly mounted to the frame, and the rotor is free to rotate, the rotor rotates, producing a useful mechanical output as shown in [Figure 2-8](#).
- 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 fine-tuning is needed after closing the speed loop using a sensorless algorithm to draw the 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 experiences rapidly alternating positive and negative torque. This results in less than excellent 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 stops rotating at the synchronous frequency, and responds 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  $90^\circ$  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.



**Figure 2-8. Interaction Between the Rotating Stator Flux and the Rotor Flux Produces Torque**

### 2.4.2.1 Field-Oriented Control of PM Synchronous Motor

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, advanced control strategies can be implemented, 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 2-9. 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 exceptional 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.

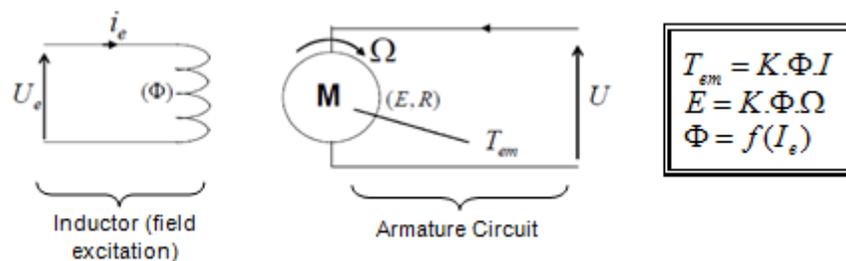


Figure 2-9. Flux and Torque are Independently Controlled in DC Motor Model

The goal of the FOC (also called vector control) on synchronous and asynchronous machines is to be able to separately control the torque-producing and magnetizing flux components. FOC control allows decoupling of the torque and of 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, you must 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 the 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 the vector cross product of the two existing magnetic fields as in Equation 4 .

$$\tau_{em} = \vec{B}_{stator} \times \vec{B}_{rotor} \quad (4)$$

This expression shows that the torque is maximum if stator and rotor magnetic fields are *orthogonal* meaning to maintain the load at 90 degrees. If this condition can be provided all the time and if the flux can be oriented correctly, the torque ripple is reduced and a better dynamic response is provided. However, the constraint is to know the rotor position: this can be achieved with a position sensor such as incremental encoder. For low-cost applications 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 FOC 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 coordinate (d and q coordinates) time invariant system. These projections lead to a structure similar to that of a DC machine control. FOC machines need two constants as input references: the torque component (aligned with the q coordinate) and the flux component (aligned with d coordinate). As FOC 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 5](#).

$$\tau_{em} \propto \psi_R \times i_{sq} \quad (5)$$

By maintaining the amplitude of the rotor flux ( $\psi_R$ ) at a fixed value, a linear relationship between torque and torque component ( $i_{sq}$ ) is obtained. Therefore, the torque can be controlled by controlling the torque component of the stator current vector.

#### 2.4.2.1.1 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 6](#).

$$\bar{i}_s = i_a + \alpha i_b + \alpha^2 i_c \quad (6)$$

where

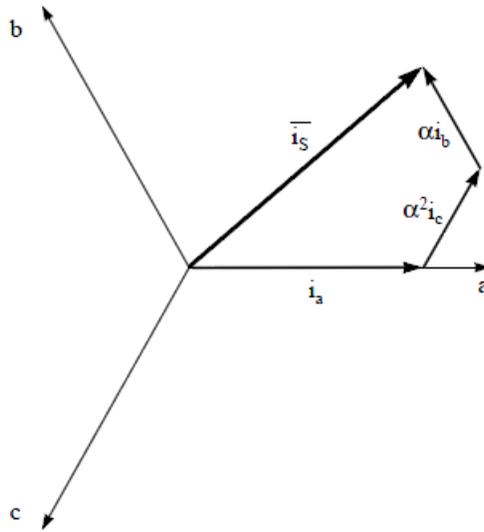
$$\alpha = e^{j\frac{2}{3}\pi} \quad (7)$$

and

$$\alpha^2 = e^{j\frac{4}{3}\pi} \quad (8)$$

represent the spatial operators

[Figure 2-10](#) shows the stator current complex space vector.



**Figure 2-10. Stator Current Space Vector and Component in (a, b, c) Frame**

where a, b, and c are the three-phase system axes.

This current space vector depicts the three-phase sinusoidal system which still needs to be transformed into a two time invariant co-ordinate system. This transformation can be split into two steps:

$$(a, b) \Rightarrow (\alpha, \beta) \tag{9}$$

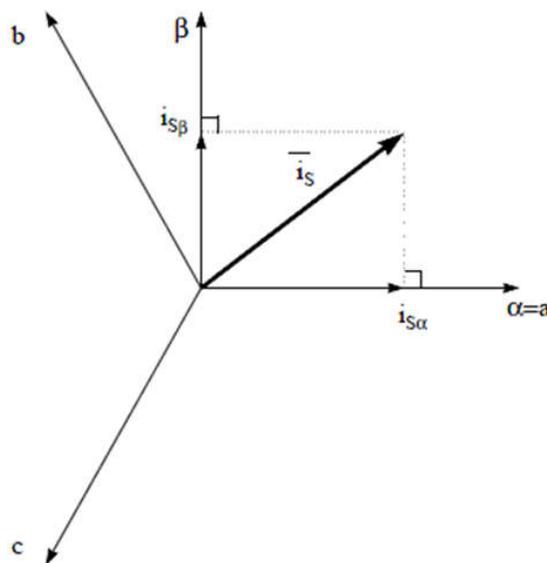
- (Clarke transformation) which outputs a 2-coordinate time-variant system

$$(\alpha, \beta) \Rightarrow (d, q) \tag{10}$$

- (Park transformation) which outputs a 2-coordinate time-invariant system

**2.4.2.1.1.1 (a, b) → (α, β) 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 *alpha* are in the same direction yields the vector diagram shown in [Figure 2-11](#).



**Figure 2-11. Stator Current Space Vector in the Stationary Reference Frame**

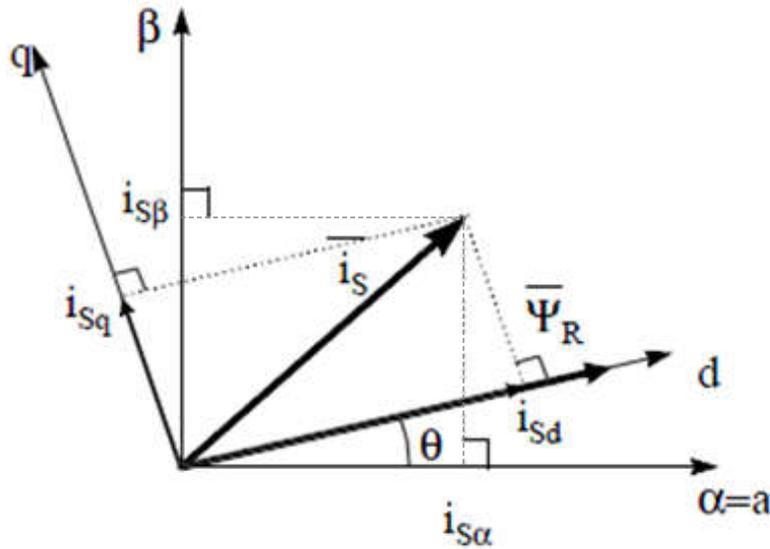
The projection that modifies the 3-phase system into the  $(\alpha, \beta)$  2-dimension orthogonal system is presented in [Equation 11](#).

$$\begin{aligned} i_{s\alpha} &= i_a \\ i_{s\beta} &= \frac{1}{\sqrt{3}}i_a + \frac{2}{\sqrt{3}}i_b \end{aligned} \quad (11)$$

The two phase  $(\alpha, \beta)$  currents are still dependent on time and speed.

#### 2.4.2.1.1.2 $(\alpha, \beta) \rightarrow (d, q)$ Park Transformation

This is the most important transformation in the FOC. In fact, this projection modifies a 2-phase orthogonal system  $(\alpha, \beta)$  in the  $(d, q)$  rotating reference frame. Considering the  $d$  axis aligned with the rotor flux, [Figure 2-12](#) shows the relationship for the current vector from the two reference frame.



**Figure 2-12. Stator Current Space Vector in the d,q Rotating Reference Frame**

The flux and torque components of the current vector are determined by [Equation 12](#).

$$\begin{aligned} i_{sd} &= i_{s\alpha}\cos(\theta) + i_{s\beta}\sin(\theta) \\ i_{sq} &= -i_{s\alpha}\sin(\theta) + i_{s\beta}\cos(\theta) \end{aligned} \quad (12)$$

where

- $\theta$  is the rotor flux position

These components depend on the current vector  $(\alpha, \beta)$  components and on the rotor flux position; if the right rotor flux position is known 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.

#### 2.4.2.1.2 Basic Scheme of FOC for AC Motor

[Figure 2-13](#) summarizes the basic scheme of torque control with FOC.

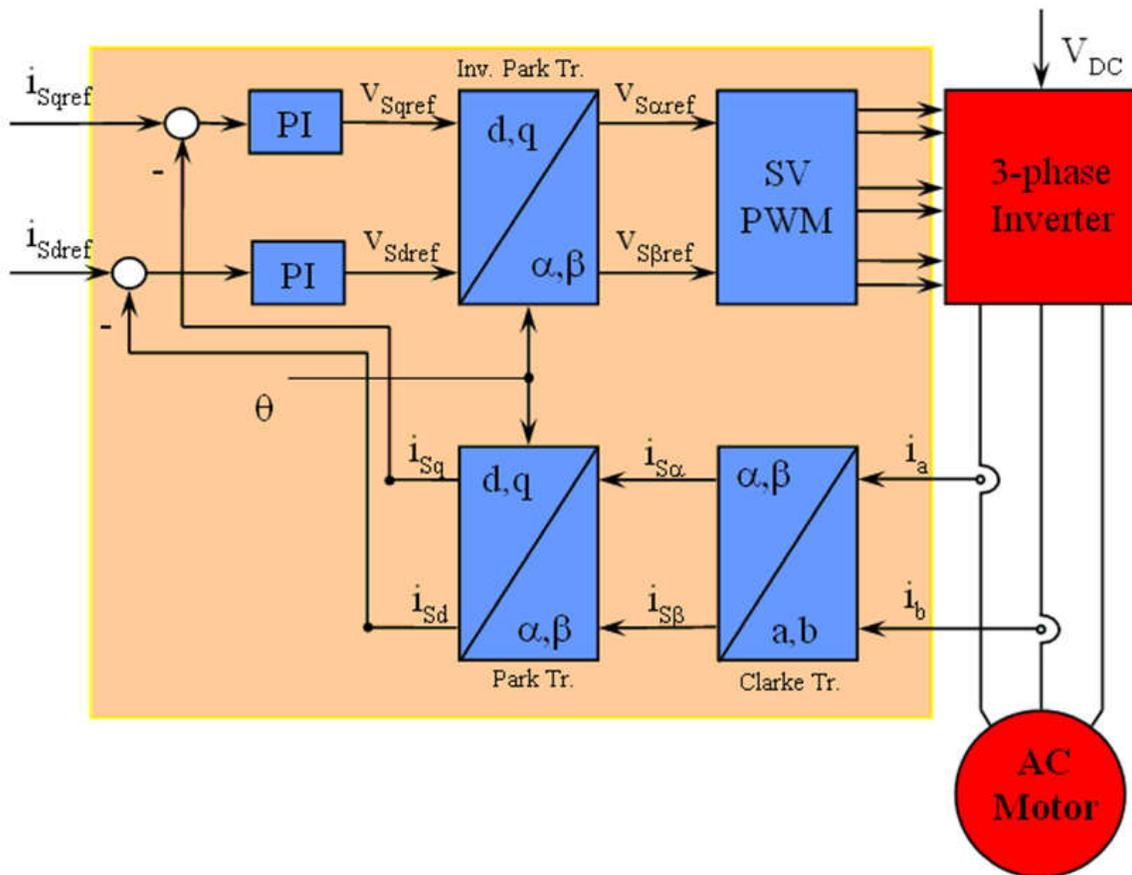
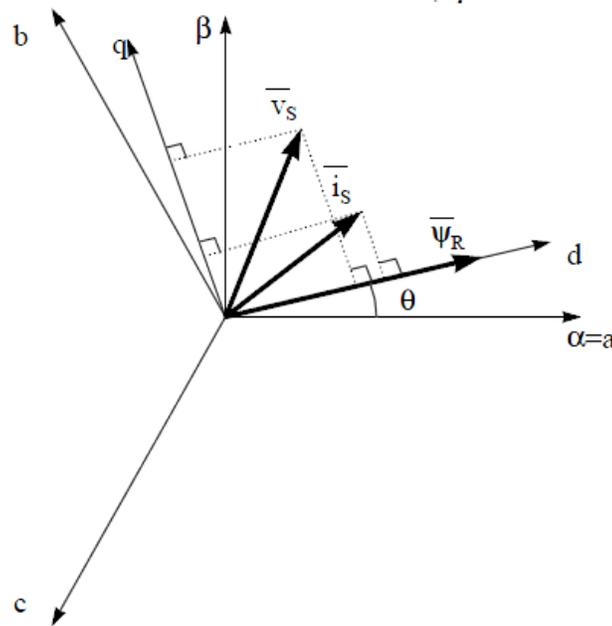


Figure 2-13. Basic Scheme of FOC for AC Motor

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: the structure 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, set  $i_{sdref}$  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  $i_{sqref}$  can be the output of the speed regulator when a speed FOC is used. The outputs of the current regulators are  $V_{sdref}$  and  $V_{sqref}$ ; these outputs are applied to the inverse Park transformation. The outputs of this projection are  $V_{saref}$  and  $V_{sbrref}$  which are the components of the stator vector voltage in the  $(\alpha, \beta)$  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).

#### 2.4.2.1.3 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 the  $d$ -axis and  $i_{sd}$  and  $i_{sq}$  are incorrect flux and torque components of the stator current. Figure 2-14 shows the  $(a, b, c)$ ,  $(\alpha, \beta)$  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 2-14. Current, Voltage and Rotor Flux Space Vectors in the (d, q) Rotating Reference Frame**

The measure of the rotor flux position is different when considering the synchronous or asynchronous motor:

- In the synchronous machine the rotor speed is equal to the rotor flux speed. Then  $\theta$  (rotor flux position) is directly measured by the 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 a particular method is needed to calculate  $\theta$ . 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 FOC for the PMSM drive allows the motor torque to 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 is 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 2-15 shows the overall block diagram of sensorless FOC of fan PMSM using eSMO with flying start in this reference design.

Figure 2-16 shows the overall block diagram of sensorless FOC of compressor PMSM using eSMO with field weakening control (FWC) and maximum torque per ampere (MTPA) in this reference design.

Figure 2-17 shows the overall block diagram of sensorless FOC of fan PMSM using FAST with flying start in this reference design.

Figure 2-18 shows the overall block diagram of sensorless FOC of compressor PMSM using FAST with field weakening control (FWC) and maximum torque per ampere (MTPA) in this reference design.

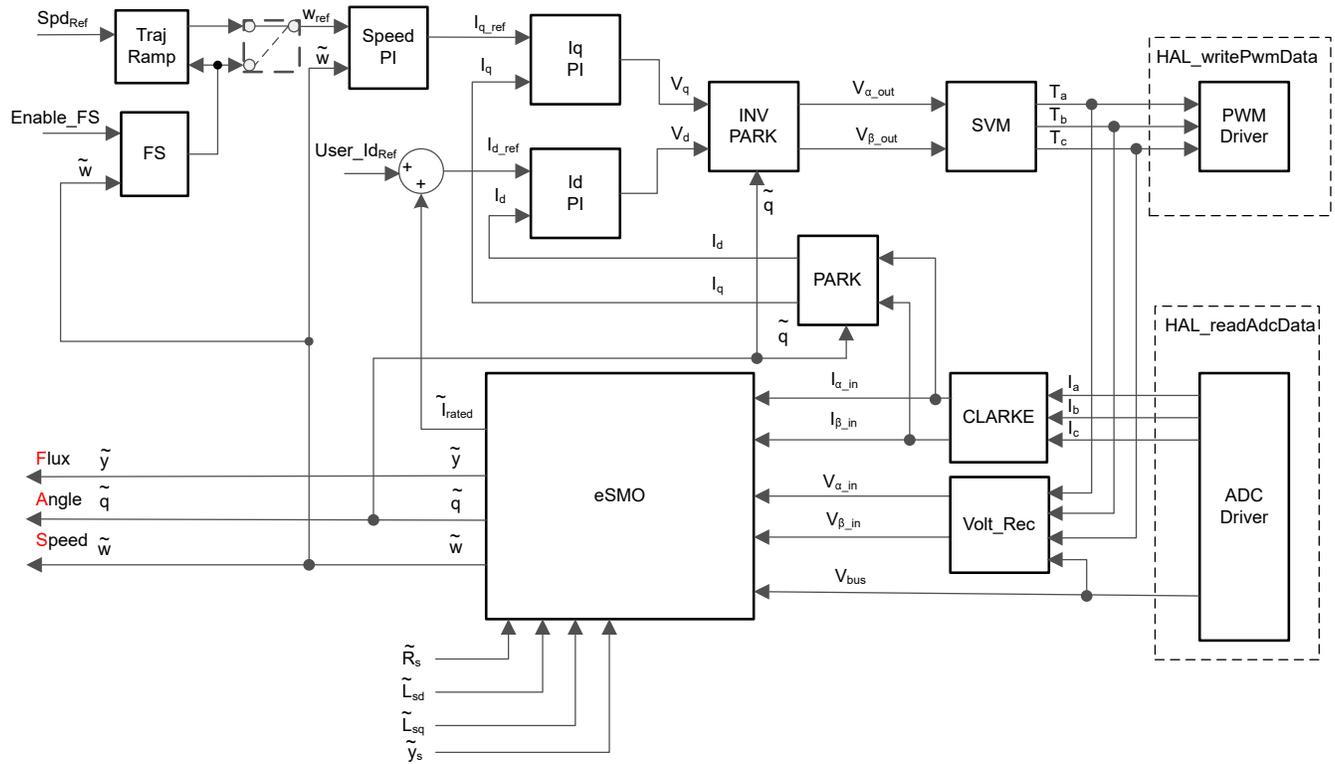


Figure 2-15. Sensorless FOC of PMSM Using eSMO With Flying Start (FS)

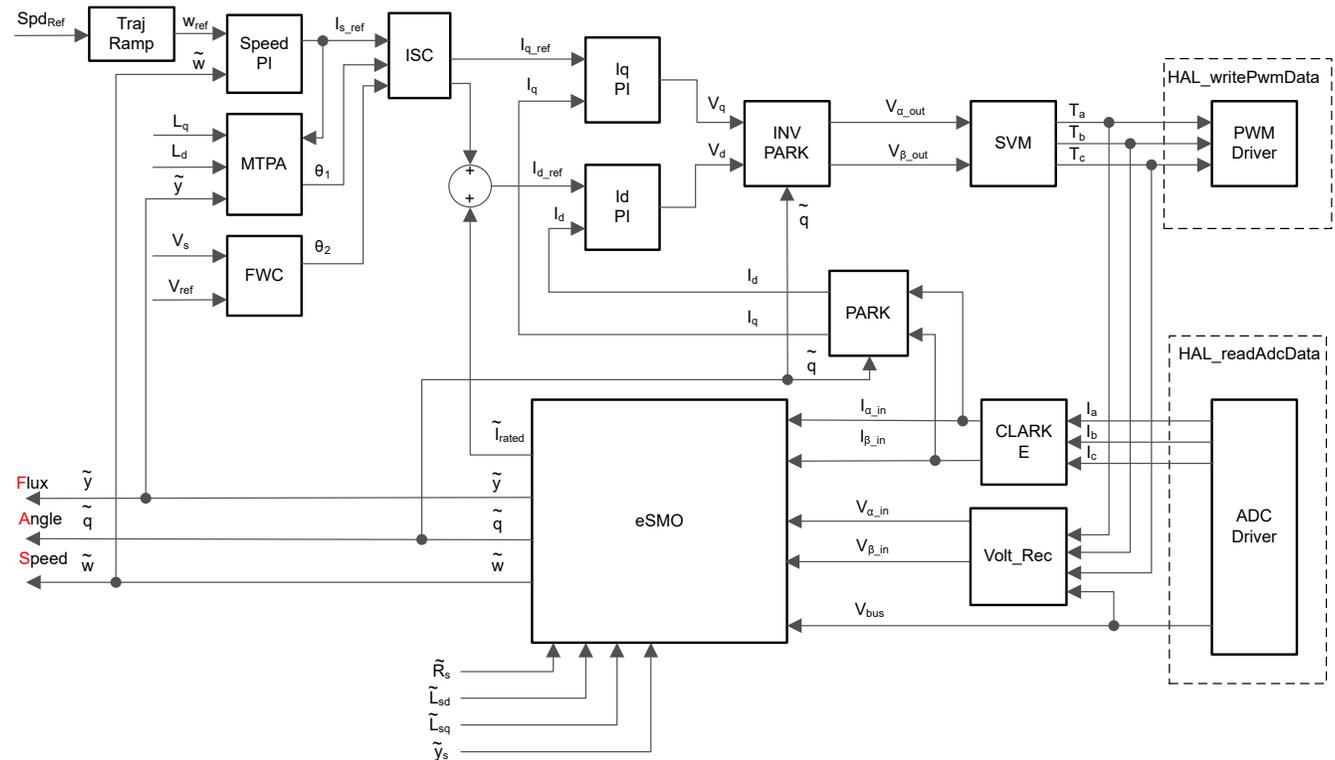


Figure 2-16. Sensorless FOC of PMSM Using eSMO With FWC and MTPA

System Overview

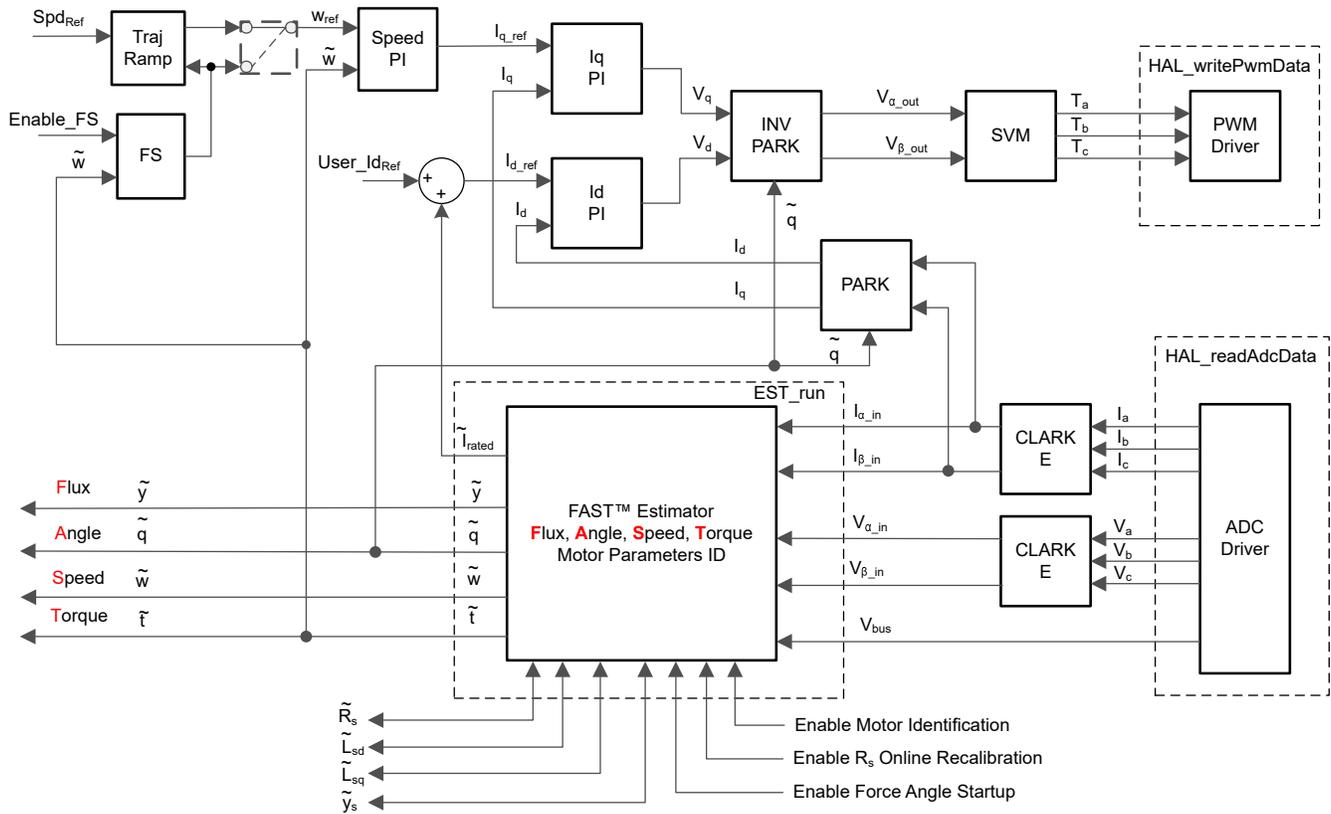


Figure 2-17. Sensorless FOC of PMSM Using FAST With Flying Start (FS)

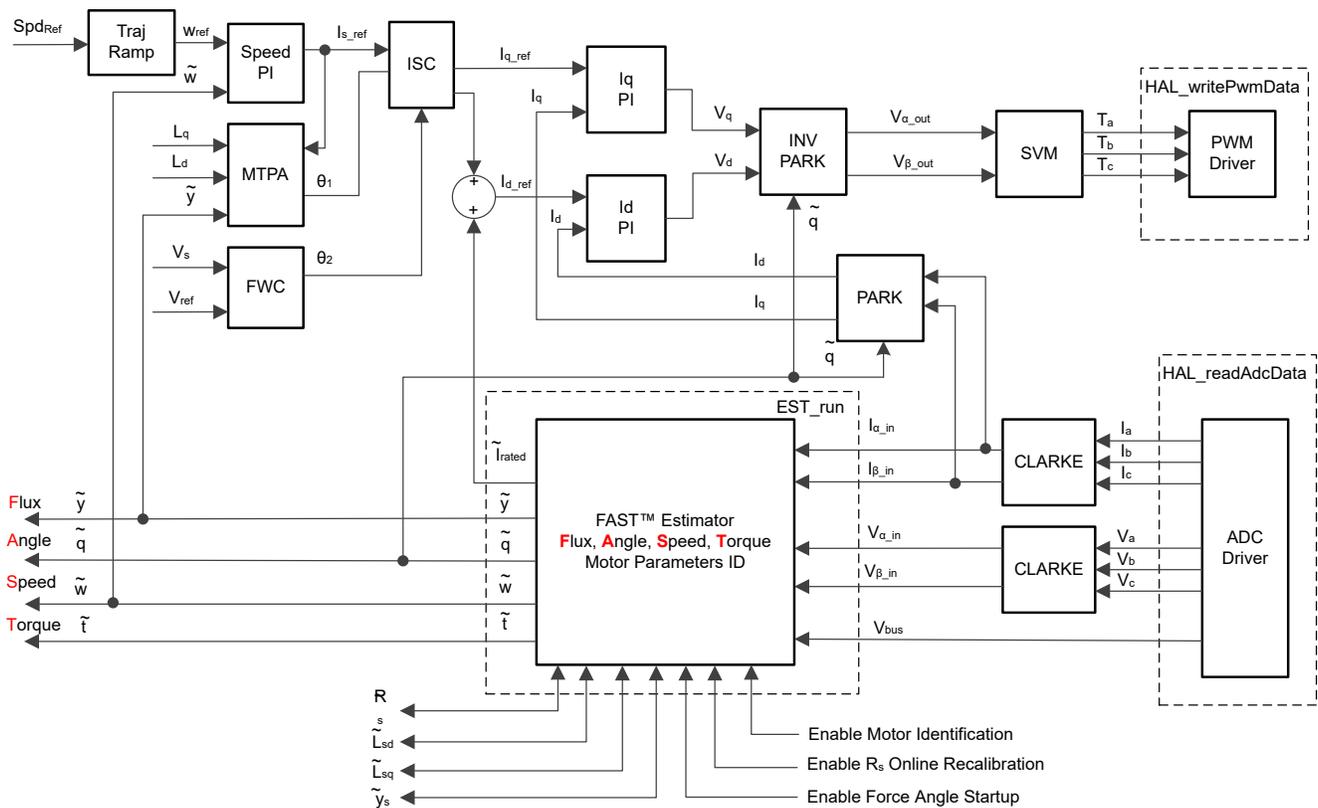


Figure 2-18. Sensorless FOC of PMSM Using FAST With FWC and MTPA

### 2.4.2.2 Sensorless Control of PM Synchronous Motor

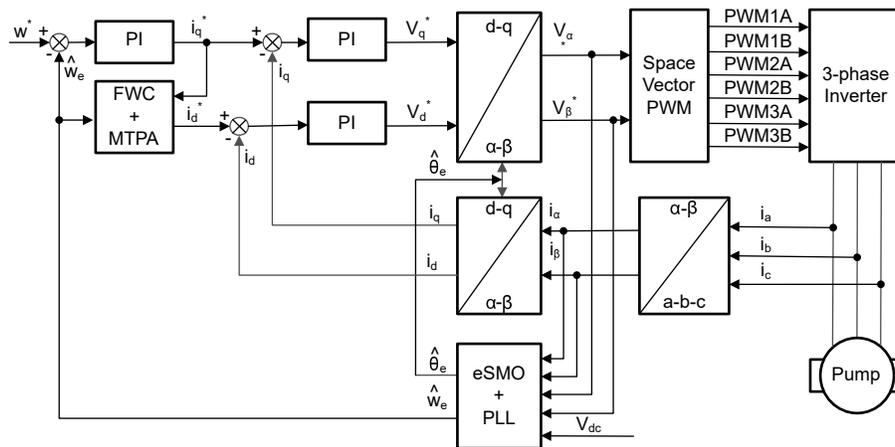
In home appliance applications, using a mechanical sensor increases cost, size, and reliability problems. To overcome these problems, sensorless control methods are implemented. Several estimation methods are used to get the rotor speed and position information without mechanical position sensor. The sliding mode observer (SMO) is commonly utilized due to the various attractive features including reliability, desired performance, and robustness against system parameter variations.

#### 2.4.2.2.1 Enhanced Sliding Mode Observer With Phase-Locked Loop

A model-based method is used to achieve position sensorless control of the IPMSM drive system when the motor runs at middle or high speed. The model method 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. The advantages include fast response, strong robustness, and insensitivity to both parameter changes and disturbances.

##### 2.4.2.2.1.1 Mathematical Model and FOC Structure of an IPMSM

The sensorless FOC structure for an IPMSM is illustrated in Figure 2-19. 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 2-19. Sensorless FOC Structure of an IPMSM 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  $\alpha$ - $\beta$  stationary reference frame can be obtained. The relationship among these reference frames are illustrated in Equation 13. The dynamic model of a generic PMSM can be written in the d-q rotor reference frame as:

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} R_s + pL_d & -\omega_e L_q \\ \omega_e L_d & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} 0 \\ \omega_e \lambda_{pm} \end{bmatrix} \quad (13)$$

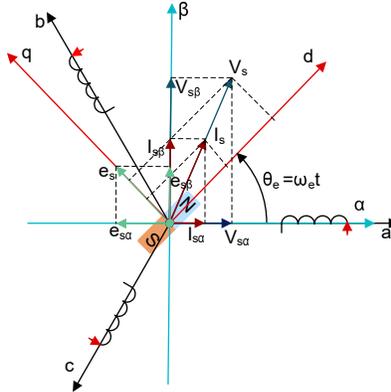
where

- $v_d$  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 Equation 14

$$\frac{d}{dt} \quad (14)$$

- $\lambda_{pm}$  is the flux linkage generated by the permanent magnets

- $R_s$  is the resistance of the stator windings
- $\omega_e$  is the electrical angular velocity of the rotor



**Figure 2-20. Definitions of Coordinate Reference Frames for PMSM Modeling**

By using the inverse Park transformation as shown in [Figure 2-20](#), the dynamics of the PMSM can be modeled in the  $\alpha$ - $\beta$  stationary reference frame as shown in [Equation 15](#):

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} R_s + pL_d & \omega_e(L_d - L_q) \\ -\omega_e(L_d - L_q) & R_s + pL_q \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} \quad (15)$$

where

- $e_\alpha$  and  $e_\beta$  are components of extended electromotive force (EEMF) in the  $\alpha$ - $\beta$  axis and can be defined as shown in [Equation 16](#):

$$\begin{bmatrix} e_\alpha \\ e_\beta \end{bmatrix} = (\lambda_{pm} + (L_d - L_q)i_d)\omega_e \begin{bmatrix} -\sin(\theta_e) \\ \cos(\theta_e) \end{bmatrix} \quad (16)$$

According to [Equation 15](#) and [Equation 16](#), 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 [Equation 15](#) as a state equation using the stator current as a state variable:

$$\begin{bmatrix} \dot{i}_\alpha \\ \dot{i}_\beta \end{bmatrix} = \frac{1}{L_d} \begin{bmatrix} -R_s & -\omega_e(L_d - L_q) \\ \omega_e(L_d - L_q) & -R_s \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{1}{L_d} \begin{bmatrix} V_\alpha - e_\alpha \\ V_\beta - e_\beta \end{bmatrix} \quad (17)$$

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

$$S(x) = \begin{bmatrix} \hat{i}_\alpha - i_\alpha \\ \hat{i}_\beta - i_\beta \end{bmatrix} = \begin{bmatrix} \tilde{i}_\alpha \\ \tilde{i}_\beta \end{bmatrix} \quad (18)$$

where

- [Equation 19](#) and [Equation 20](#) are the estimated currents

$$\hat{i}_\alpha \quad (19)$$

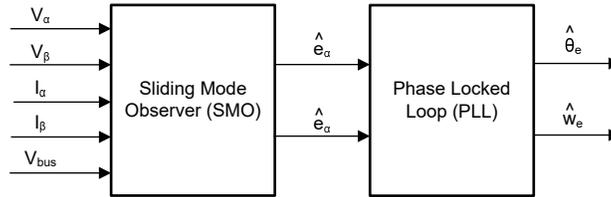
$$\hat{i}_\beta \quad (20)$$

- the superscript  $\hat{\phantom{x}}$  indicates the estimated value

- the superscript  $\sim$  indicates the variable error which refers to the difference between the observed value and the actual measurement value

**2.4.2.2.1.2 Design of ESMO for the IPMSM**

Figure 2-21 shows the conventional PLL integrated into the SMO.



**Figure 2-21. Block Diagram of eSMO With PLL for a PMSM**

The traditional reduced-order sliding-mode observer is constructed, with the mathematical model shown in Equation 21 and the block diagram shown in Figure 2-22.

$$\begin{bmatrix} \dot{\hat{i}}_{\alpha} \\ \dot{\hat{i}}_{\beta} \end{bmatrix} = \frac{1}{L_d} \begin{bmatrix} -R_s & -\hat{\omega}_e(L_d - L_q) \\ \hat{\omega}_e(L_d - L_q) & -R_s \end{bmatrix} \begin{bmatrix} \hat{i}_{\alpha} \\ \hat{i}_{\beta} \end{bmatrix} + \frac{1}{L_d} \begin{bmatrix} V_{\alpha} - \hat{e}_{\alpha} + z_{\alpha} \\ V_{\beta} - \hat{e}_{\beta} + z_{\beta} \end{bmatrix} \quad (21)$$

where

- $z_{\alpha}$  and  $z_{\beta}$  are sliding-mode feedback components and are defined as shown in Equation 22:

$$\begin{bmatrix} z_{\alpha} \\ z_{\beta} \end{bmatrix} = \begin{bmatrix} k_{\alpha} \text{sign}(\hat{i}_{\alpha} - i_{\alpha}) \\ k_{\beta} \text{sign}(\hat{i}_{\beta} - i_{\beta}) \end{bmatrix} \quad (22)$$

where

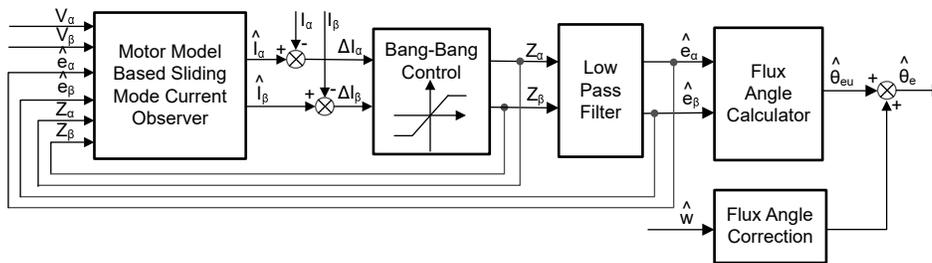
- $k_{\alpha}$  and  $k_{\beta}$  are the constant sliding-mode gain designed by Lyapunov stability analysis

If  $k_{\alpha}$  and  $k_{\beta}$  are positive and significant enough to provide the stable operation of the SMO, then  $k_{\alpha}$  and  $k_{\beta}$  are large enough to hold

$$k_{\alpha} > \max(|e_{\alpha}|) \quad (23)$$

and

$$k_{\beta} > \max(|e_{\beta}|) \quad (24)$$



**Figure 2-22. Block Diagram of Traditional Sliding-Mode Observer**

The estimated value of EEMF in  $\alpha$ - $\beta$  axes (Equation 25, Equation 26) can be obtained by low-pass filter from the discontinuous switching signals  $z_{\alpha}$  and  $z_{\beta}$ :

$$\hat{e}_{\alpha} \quad (25)$$

$$\hat{e}_{\beta} \quad (26)$$

$$\begin{bmatrix} \hat{e}_\alpha \\ \hat{e}_\beta \end{bmatrix} = \frac{\omega_c}{s + \omega_c} \begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} \quad (27)$$

where

$$\omega_c = 2\pi f_c \quad (28)$$

is the cutoff angular frequency of the LPF, which is usually selected according to the fundamental frequency of the stator current

Therefore, the rotor position can be directly calculated from arc-tangent the back EMF, as [Equation 29](#) defines:

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) \quad (29)$$

Low-pass filters remove the high-frequency term of the sliding-mode function, which results in phase delay. The delay can be compensated by the relationship between the cut-off frequency  $\omega_c$  and back EMF frequency  $\omega_e$ , which is defined as shown in [Equation 30](#):

$$\Delta\theta_e = -\tan^{-1}\left(\frac{\omega_e}{\omega_c}\right) \quad (30)$$

Then the estimated rotor position by using SMO method is found with [Equation 31](#):

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) + \Delta\theta_e \quad (31)$$

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 21](#) in  $\alpha$ - $\beta$  coordinates is given by [Equation 32](#) as:

$$\begin{bmatrix} \hat{i}_\alpha(n+1) \\ \hat{i}_\beta(n+1) \end{bmatrix} = \begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} \begin{bmatrix} \hat{i}_\alpha(n) \\ \hat{i}_\beta(n) \end{bmatrix} + \begin{bmatrix} G_\alpha \\ G_\beta \end{bmatrix} \begin{bmatrix} v_\alpha^*(n) - \hat{e}_\alpha(n) + z_\alpha(n) \\ v_\beta^*(n) - \hat{e}_\beta(n) + z_\beta(n) \end{bmatrix} \quad (32)$$

where

- the matrix [F] and [G] are given by [Equation 33](#) and [Equation 34](#) as:

$$\begin{bmatrix} F_\alpha \\ F_\beta \end{bmatrix} = \begin{bmatrix} e^{-\frac{R_s}{L_d}} \\ e^{-\frac{R_s}{L_q}} \end{bmatrix} \quad (33)$$

$$\begin{bmatrix} G_\alpha \\ G_\beta \end{bmatrix} = \frac{1}{R_s} \begin{bmatrix} 1 - e^{-\frac{R_s}{L_d}} \\ 1 - e^{-\frac{R_s}{L_q}} \end{bmatrix} \quad (34)$$

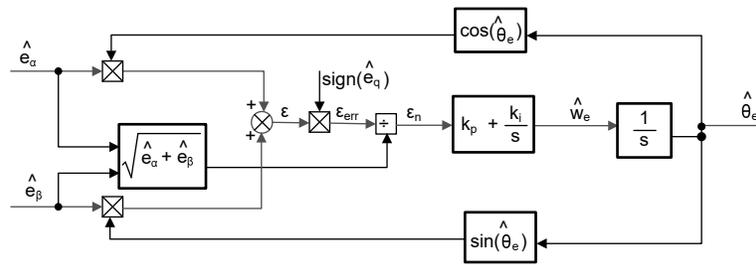
The time-discrete form of [Equation 27](#) is given by [Equation 35](#) as:

$$\begin{bmatrix} \hat{e}_\alpha(n+1) \\ \hat{e}_\beta(n+1) \end{bmatrix} = \begin{bmatrix} \hat{e}_\alpha(n) \\ \hat{e}_\beta(n) \end{bmatrix} + 2\pi f_c \begin{bmatrix} z_\alpha(n) - \hat{e}_\alpha(n) \\ z_\beta(n) - \hat{e}_\beta(n) \end{bmatrix} \quad (35)$$

#### 2.4.2.2.1.3 Rotor Position and Speed Estimation With PLL

With the arc tangent method, 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 IPMSM. [Section 2.4.2.2.1.2](#) illustrates the PLL

structure used with SMO. The back-EMF estimations Equation 25 and Equation 26 can be used with a PLL model to estimate the motor angular velocity and position as shown in Figure 2-23.



**Figure 2-23. Block Diagram of Phase-Locked Loop Position Tracker**

Since Equation 36, Equation 37, and Equation 38, the position error can be defined as Equation 39:

$$e_\alpha = E \cos(\theta_e) \quad (36)$$

$$e_\beta = E \sin(\theta_e) \quad (37)$$

$$E = \omega_e \lambda_{pm} \quad (38)$$

$$\epsilon = \hat{e}_\beta \cos(\hat{\theta}_e) - \hat{e}_\alpha \sin(\hat{\theta}_e) = E \sin(\theta_e) \cos(\hat{\theta}_e) - E \cos(\theta_e) \sin(\hat{\theta}_e) = E \sin(\theta_e - \hat{\theta}_e) \quad (39)$$

where  $E$  is the magnitude of the EEMF, which is proportional to the motor speed  $\omega_e$ .

When Equation 40, then Equation 39 can be simplified as Equation 41.

$$(\theta_e - \hat{\theta}_e) < \frac{\pi}{2} \quad (40)$$

$$\epsilon = E(\theta_e - \hat{\theta}_e) \quad (41)$$

Further, the position error after the normalization of the EEMF can be obtained (Equation 42):

$$\epsilon_n = \theta_e - \hat{\theta}_e \quad (42)$$

According to the analysis, the simplified block diagram of the quadrature phase-locked loop position tracker can be obtained as shown in Figure 2-24. The closed-loop transfer functions of the PLL can be expressed as Equation 43:

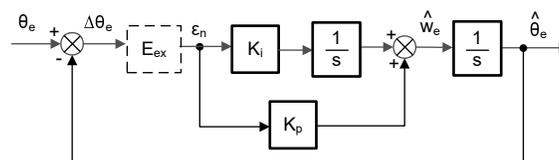
$$\frac{\hat{\theta}_e}{\theta_e} = \frac{k_p s + k_i}{s^2 + k_p s + k_i} = \frac{2\xi\omega_n s + \omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \quad (43)$$

where

- $k_p$  and  $k_i$  are the proportional and the integral gains of the standard PI regulator

The natural frequency  $\omega_n$  and the damping ratio  $\xi$  are given in Equation 44:

$$k_p = 2\xi\omega_n, \quad k_i = \omega_n^2 \quad (44)$$



**Figure 2-24. Simplified Block Diagram of Phase-Locked Loop Position Tracker**

### 2.4.2.3 Field Weakening (FW) and Maximum Torque Per Ampere (MTPA) Control

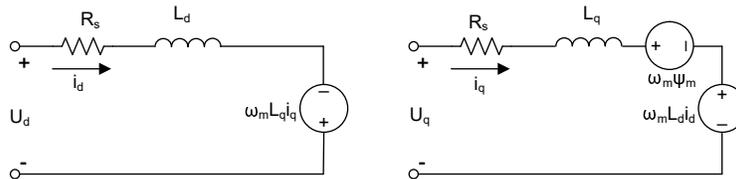
Permanent magnet synchronous motor (PMSM) is widely used in home appliance applications due to the high power density, high efficiency, and wide speed range. The PMSM includes two major types: the surface-mounted PMSM (SPM), and the interior PMSM (IPM). SPM motors are easier to control due to the linear relationship between the torque and q-axis current. However, the IPMSM has electromagnetic and reluctance torques due to a large saliency ratio. The total torque is non-linear with respect to the rotor angle. As a result, the MTPA technique can be used for IPM motors to optimize torque generation in the constant torque region. The aim of the field-weakening control is to optimize to reach the highest power and efficiency of a PMSM drive. Field-weakening control can enable a motor operation over the base speed, expanding the operating limits to reach speeds higher than rated speed and allow exceptional control across the entire speed and voltage range.

The voltage equations of the mathematical model of an IPMSM can be described in d-q coordinates as shown in Equation 45 and Equation 46.

$$v_d = L_d \frac{di_d}{dt} + R_s i_d - p \omega_m L_q i_q \quad (45)$$

$$v_q = L_q \frac{di_q}{dt} + R_s i_q + p \omega_m L_d i_d + p \omega_m \psi_m \quad (46)$$

Figure 2-25 shows the dynamic equivalent circuit of an IPM synchronous motor.



**Figure 2-25. Equivalent Circuit of an IPM Synchronous Motor**

The total electromagnetic torque generated by the IPMSM can be expressed as Equation 47 that the produced torque is composed of two distinct terms. The first term corresponds to the mutual reaction torque occurring between torque current  $i_q$  and the permanent magnet  $\psi_m$ , while the second term corresponds to the reluctance torque due to the differences in d-axis and q-axis inductance.

$$T_e = \frac{3}{2} p [\psi_m i_q + (L_d - L_q) i_d i_q] \quad (47)$$

In most applications, IPMSM drives have speed and torque constraints, mainly due to inverter or motor rating currents and available DC link voltage limitations respectively. These constraints can be expressed with the mathematical equations Equation 48 and Equation 49.

$$I_a = \sqrt{i_d^2 + i_q^2} \leq I_{max} \quad (48)$$

$$V_a = \sqrt{v_d^2 + v_q^2} \leq V_{max} \quad (49)$$

where  $V_{max}$  and  $I_{max}$  are the maximum allowable voltage and current of the inverter or motor.

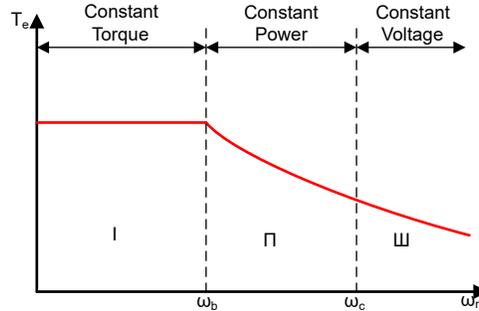
In a two-level three-phase Voltage Source Inverter (VSI) fed machine, the maximum achievable phase voltage is limited by the DC link voltage and the PWM strategy. The maximum voltage is limited to the value as shown in Equation 50 if Space Vector Modulation (SVPWM) is adopted.

$$\sqrt{v_d^2 + v_q^2} \leq v_{max} = \frac{v_{dc}}{\sqrt{3}} \quad (50)$$

Usually the stator resistance  $R_s$  is negligible at high speed operation and the derivative of the currents is zero in steady state, thus Equation 51 is obtained as shown.

$$\sqrt{L_d^2 \left( i_d + \frac{\psi_{pm}}{L_d} \right)^2 + L_q^2 i_q^2} \leq \frac{V_{max}}{\omega_m} \quad (51)$$

The current limitation of Equation 48 produces a circle of radius  $I_{max}$  in the d-q plane, and the voltage limitation of Equation 50 produces an ellipse whose radius  $V_{max}$  decreases as speed increases. The resultant d-q plane current vector must be controlled to obey the current and voltage constraints simultaneously. According to these constraints, three operation regions for the IPMSM can be distinguished as shown in Figure 2-26.



**Figure 2-26. IPMSM Control Operation Regions**

1. Constant Torque Region: MTPA can be implemented in this operation region to provide maximum torque generation.
2. Constant Power Region: Field-weakening control must be employed and the torque capacity is reduced as the current constraint is reached.
3. Constant Voltage Region: In this operation region, deep field-weakening control keeps a constant stator voltage to maximize the torque generation.

In the constant torque region, according to Equation 47, the total torque of an IPMSM includes the electromagnetic torque from the magnet flux linkage and the reluctance torque from the saliency between  $L_d$  and  $L_q$ . The electromagnetic torque is proportional to the q-axis current  $i_q$ , and the reluctance torque is proportional to the multiplication of the d-axis current  $i_d$ , the q-axis current  $i_q$ , and the difference between  $L_d$  and  $L_q$ .

Conventional vector control systems of SPM motors only utilizes electromagnetic torque by setting the commanded  $i_d$  to zero for non-field-weakening modes. But while the IPMSM utilizes the reluctance torque of the motor, the designer must also control the d-axis current. The aim of the MTPA control is to calculate the reference currents  $i_d$  and  $i_q$  to maximize the ratio between produced electromagnetic torque and reluctance torque. The relationship between  $i_d$  and  $i_q$ , and the vectorial sum of the stator current  $I_s$  is shown in the following equations.

$$I_s = \sqrt{i_d^2 + i_q^2} \quad (52)$$

$$I_d = I_s \cos \beta \quad (53)$$

$$I_q = I_s \sin \beta \quad (54)$$

where

- $\beta$  is the stator current angle in the synchronous (d-q) reference frame

Equation 47 can be expressed as Equation 55 where  $I_s$  substituted for  $i_d$  and  $i_q$ .

Equation 55 shows that motor torque depends on the angle of the stator current vector:

$$T_e = \frac{3}{2} p I_s \sin \beta \left[ \psi_m + (L_d - L_q) I_s \cos \beta \right] \quad (55)$$

This equation shows the maximum efficiency point can be calculated when the motor torque differential is equal to zero. The MTPA point can be found when this differential, Equation 56 is zero as given in Equation 57.

$$\frac{dT_e}{d\beta} \quad (56)$$

$$\frac{dT_e}{d\beta} = \frac{3}{2}p \left[ \psi_m I_s \cos \beta + (L_d - L_q) I_s^2 \cos 2\beta \right] = 0 \quad (57)$$

Following this equation, the current angle of the MTPA control can be derived as in [Equation 58](#).

$$\beta_{mtpa} = \cos^{-1} \frac{-\psi_m + \sqrt{\psi_m^2 + 8 \times (L_d - L_q)^2 \times I_s^2}}{4 \times (L_d - L_q) \times I_s} \quad (58)$$

Thus, the effective d-axis and q-axis reference currents can be expressed by [Equation 59](#) and [Equation 60](#) using the current angle of the MTPA control.

$$I_d = I_s \times \cos \beta_{mtpa} \quad (59)$$

$$I_q = I_s \times \sin \beta_{mtpa} \quad (60)$$

However, as shown in [Equation 58](#), the angle of the MTPA control,  $\beta_{mtpa}$  is related to d-axis and q-axis inductance. This means that the variation of inductance impedes the ability to find the exceptional MTPA point. To improve the efficiency of a motor drive, estimate the d-axis and q-axis inductance online, but the parameters  $L_d$  and  $L_q$  are not easily measured online and are influenced by saturation effects. A robust Look-Up Table (LUT) method provides controllability under electrical parameter variations. Usually, to simplify the mathematical model, the coupling effect between d-axis and q-axis inductance can be neglected. Thus, assume that  $L_d$  changes with  $i_d$  only, and  $L_q$  changes with  $i_q$  only. Consequently, d- and q-axis inductance can be modeled as a function of the d-q currents respectively, as shown in [Equation 61](#) and [Equation 62](#).

$$L_d = f_1(i_d, i_q) = f_1(i_d) \quad (61)$$

$$L_q = f_2(i_q, i_d) = f_2(i_q) \quad (62)$$

Reduce the ISR calculation burden by simplifying [Equation 58](#). The motor-parameter-based constant,  $K_{mtpa}$  is expressed instead as [Equation 64](#), where  $K_{mtpa}$  is computed in the background loop using the updated  $L_d$  and  $L_q$ .

$$K_{mtpa} = \frac{\psi_m}{4 \times (L_q - L_d)} = 0.25 \times \frac{\psi_m}{(L_q - L_d)} \quad (63)$$

$$\beta_{mtpa} = \cos^{-1} \left( K_{mtpa} \div I_s - \sqrt{(K_{mtpa} \div I_s)^2 + 0.5} \right) \quad (64)$$

A second intermediate variable,  $G_{mtpa}$  described in [Equation 65](#), is defined to further simplify the calculation. Using  $G_{mtpa}$ , the angle of the MTPA control,  $\beta_{mtpa}$  can be calculated as [Equation 66](#). These two calculations are performed in the ISR to achieve a real current angle  $\beta_{mtpa}$ .

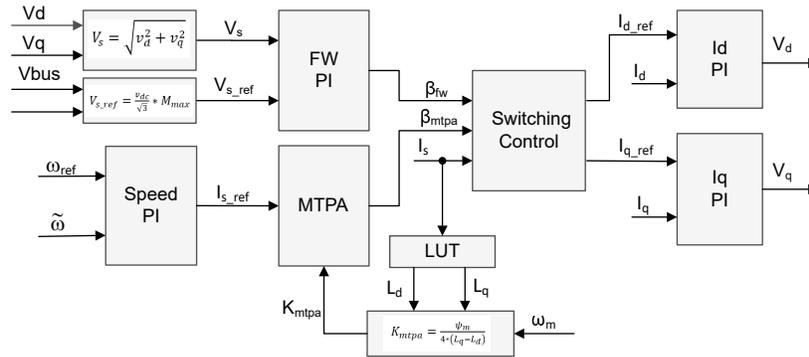
$$G_{mtpa} = K_{mtpa} \div I_s \quad (65)$$

$$\beta_{mtpa} = \cos^{-1} \left( G_{mtpa} - \sqrt{G_{mtpa}^2 + 0.5} \right) \quad (66)$$

In all cases, the magnetic flux can be weakened to extend the achievable speed range by acting on the direct axis current  $i_d$ . As a consequence of entering this constant power operating region, field-weakening control is chosen instead of the MTPA control used in constant power and voltage regions. Since the maximum inverter voltage is limited, PMSM motors cannot operate in such speed regions where the back-electromotive force, almost proportional to the permanent magnet field and motor speed, is higher than the maximum output voltage of the inverter. The direct control of magnet flux is not an option in PM motors. However, the air gap flux can be weakened by the demagnetizing effect due to the d-axis armature reaction by adding a negative  $i_d$ . Considering

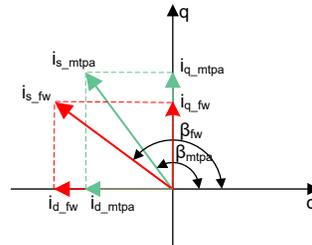
the voltage and current constraints, the armature current and the terminal voltage are limited as Equation 48 and Equation 49. The inverter input voltage (DC-Link voltage) variation limits the maximum output of the motor. Furthermore, the maximum fundamental motor voltage also depends on the PWM method used. In Equation 51, the IPMSM has two factors: one is a permanent magnet value and the other is made by inductance and current of flux.

Figure 2-27 shows the typical control structure is used to implement field weakening.  $\beta_{fw}$  is the output of the field-weakening (FW) PI controller and generates the reference  $i_d$  and  $i_q$ . Before the voltage magnitude reaches the limit, the input of the PI controller of FW is always positive and therefore the output is always saturated at 0.



**Figure 2-27. Block Diagram of Field-Weakening and Maximum Torque per Ampere Control**

Figure 2-16 and Figure 2-18 show the implementation of FAST or eSMO based FOC block diagram. The block diagrams provide an overview of the functions and variables of the FOC system. There are two control modules in the motor drive FOC system: one is MTPA control and the other one is field-weakening control. These two modules generate current angle  $\beta_{mtpa}$  and  $\beta_{fw}$ , respectively, based on input parameters as shown in Figure 2-28.



**Figure 2-28. Current Phasor Diagram of an IPMSM During FW and MTPA**

The switching control module is used to determine angle of application, and then calculate the reference  $i_d$  and  $i_q$  as shown in Equation 53 and Equation 54. The current angle is chosen as in the following: Equation 67 and Equation 68.

$$\beta = \beta_{fw} \text{ if } \beta_{fw} > \beta_{mtpa} \tag{67}$$

$$\beta = \beta_{mtpa} \text{ if } \beta_{fw} < \beta_{mtpa} \tag{68}$$

The flow chart in Figure 2-29 shows the steps required to run InstaSPIN™-FOC with FW and MPTA in the main loop and interrupt.

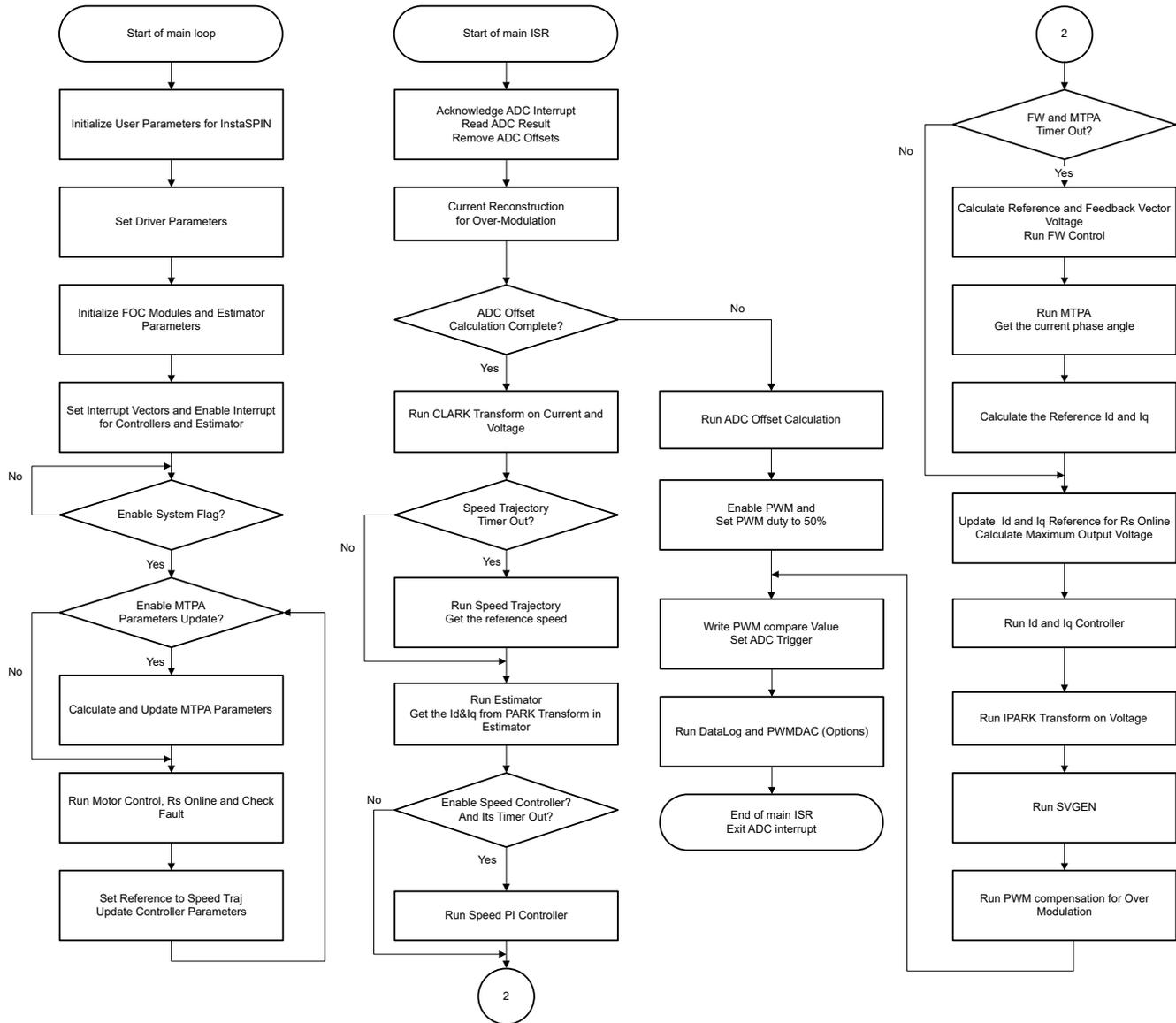


Figure 2-29. Flow Chart for an InstaSPIN-FOC Project With FW and MTPA

#### 2.4.2.4 Hardware Prerequisites for Motor Drive

The algorithm for controlling the motor makes use of sampled measurements of the motor conditions, including dc bus power supply voltage, the voltage on each motor phase, the current of each motor phase. There are a few hardware-dependent parameters that need to be set correctly to identify the motor properly and run the motor effectively using Field Oriented Control (FOC). The following sections show how to calculate the current scale value, voltage scale value, and voltage filter pole for motor control with FAST or eSMO.

##### 2.4.2.4.1 Motor Current Feedback

Two techniques are supported to measure phase currents of the motor.

- Three-shunt current sensing
- Single-shunt current sensing

Either one of these two current sensing techniques can be selected in the build configuration of the project. The *Flash\_MtrlInv\_3SC* build configuration supports three-shunt current sensing method, the *Flash\_MtrlInv\_1SC* supports single-shunt current sensing method as described in [Section 3.3.2](#).

### 2.4.2.4.1.1 Three-Shunt Current Sensing

The current through the motor is sampled by microcontroller as part of the motor control algorithm during every one PWM cycle. TMS320F2800137 daughter board supports 1-3 shunt current sensing, MSPM0G1507 daughterboard supports 1-2 shunt current sensing. To measure bidirectional currents of the motor phase, that is, positive and negative currents, this bidirectional currents requires a offset reference voltage.

Figure 2-30 shows how the motor current is represented as a voltage signal, with filtering, amplification, and offset to the center of the ADC input range for TMS320F2800137 daughterboard. This circuit is used for each phase of the 3-phase PMSM of compressor and fan. The transfer function of this circuit is given by Equation 69.

$$V_{OUT} = V_{OFFSET} + (I_{IN} \times R_{SHUNT} \times G_i) \tag{69}$$

where

- $R_{shunt} = 0.1\Omega$
- $V_{offset} = 1.65V$

The calculated resistance values lead to the sensing circuit shown in Figure 2-35,  $G_i$  is given by Equation 70.

$$G_i = \frac{R_{fb}}{R_{in}} = \frac{R20}{R17} = \frac{10\text{ k}\Omega}{2\text{ k}\Omega} = 5 \tag{70}$$

The maximum peak-to-peak current measurable by the microcontroller is given by Equation 71.

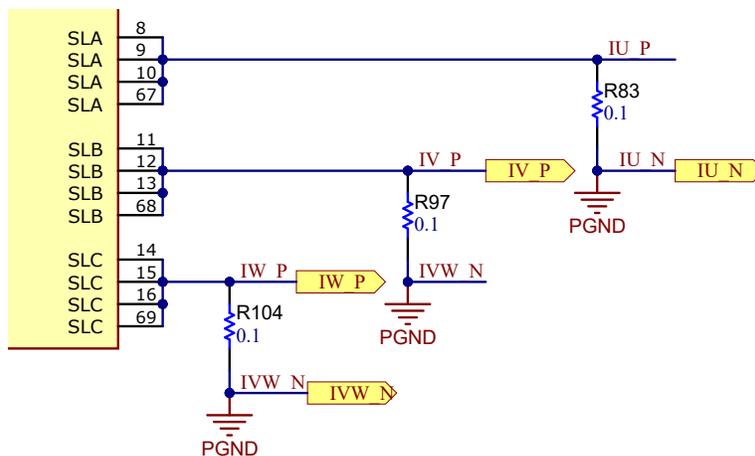
$$I_{scale\_max} = \frac{V_{ADC\_max}}{R_{SHUNT} \times G_i} = \frac{3.3}{0.1 \times 5} = 6.6\text{ A} \tag{71}$$

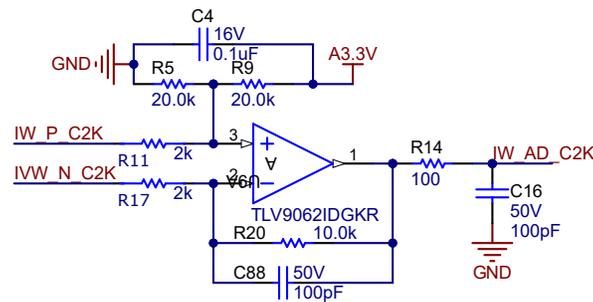
This has the peak-to-peak value of  $\pm 3.3A$ . The following code snippet shows how this is defined for compressor motor in user\_mtr1.h file:

```
///  
#define USER_M1_ADC_FULL_SCALE_CURRENT_A (6.6f)
```

Correct polarity of the current feedback is also important so that the microcontroller has an accurate current measurement. In this hardware board configuration, the negative pin of the shunt resistor, which is connected to ground, is also connected to the inverting pin of the operational amplifier. The highlighted sign is required to be configured to have the correct polarity for the current feedback in software as shown in the following code snippet in user.mtr1.h:

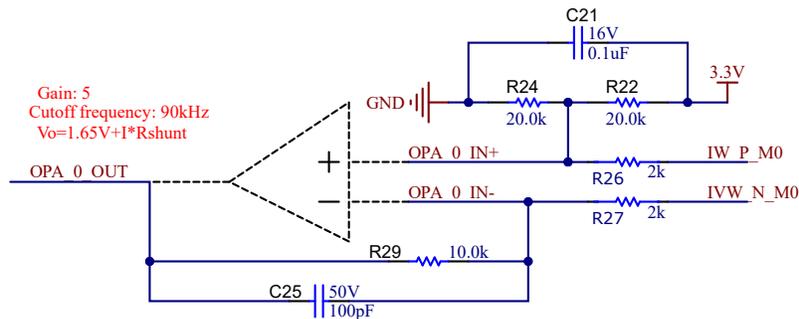
```
// define the sign of current feedback based on hardware board  
#define USER_M1_SIGN_CURRENT_SF (1.0f)
```





**Figure 2-30. Three-Shunt Current Sensing Circuit for TMS320F2800137**

On the MSPM0 daughterboard, two shunt current sensing are implemented with the two high-end internal amplifiers to save system cost. The amplifier gain is also 5, and the cutoff frequency is 90kHz. [Figure 2-31](#) shows the two-shunt current sensing circuit for the MSPM0G1507 daughterboard.



**Figure 2-31. MSPM0G1507 Two-Shunt Current-Sensing Circuit**

#### 2.4.2.4.1.2 Single-Shunt Current Sensing

The single-shunt current-sensing technique measures the DC-link bus current, with knowledge of the power FET switching states and reconstructs the three-phase current of the motor. The detailed description of the single shunt technique is described in the [Sensorless-FOC for PMSM With Single DC-Link Shunt](#) application note.

On this reference board, implement the single-shunt current-sensing technique by removing two shunts and shorting the connection of the U, V, W ground of the power module as shown in [Figure 2-32](#).

1. On the motherboard, remove current shunt resistors R97, and R104, keep only shunt resistor R83 to sense the DC-Link current.
2. On TMS320F2800137 daughterboard, remove U9 and related parts and also remove C61 to increase bandwidth of internal amplifier of DRV7308 for single-shunt sampling.
3. On the MSPM0G1507 daughterboard, remove components related with internal OPA of MSPM0G1507 to increase the bandwidth for single-shunt sampling.
4. Use a thick wire to connect the NU, NV, and NW pins together.

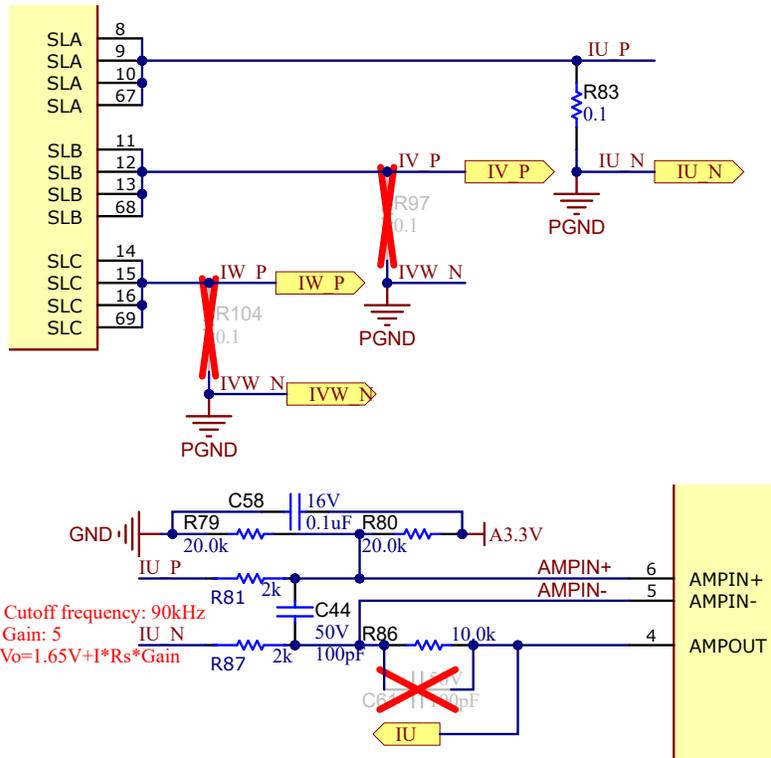


Figure 2-32. Single-Shunt Current-Sensing Circuit for TMS320F2800137

By default, the board has three shunt resistors, Figure 2-33 shows the layout of the shunt resistors. To run with a single-shunt resistor, remove R97 and R104 while keeping R83, solder IW\_P, IV\_P and IU\_P (pin 2 of R83, R97, and R104) together, then all three phase currents flow through only R83.

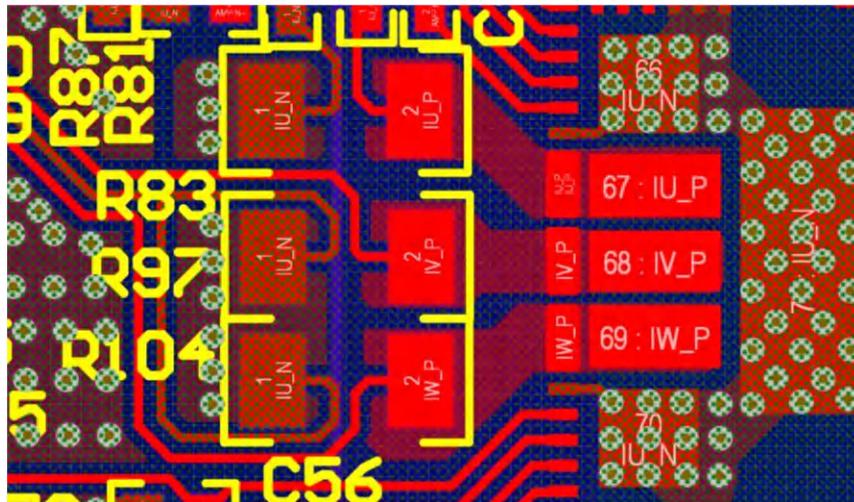
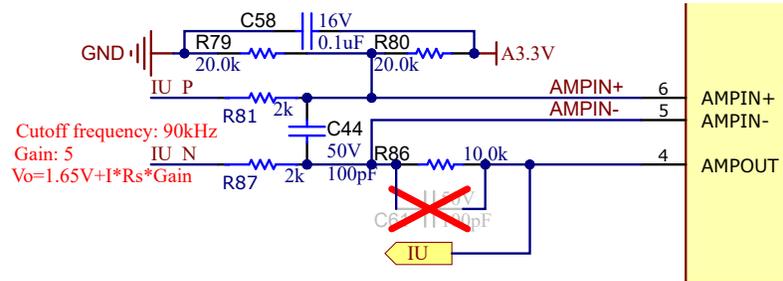


Figure 2-33. Shunt Resistors Layout

The DC-Link current is a unidirectional signal, so the DC current offset can be set to a minimum or maximum value to improve the ADC sampling range for the DC-Link current as Figure 2-34 shows. For both daughterboards, change R79 from 20kΩ to 2.2kΩ/1% resistor for the reference voltage to have 0.327V offset for DC current sensing.



**Figure 2-34. DC Offset Reference for Single Shunt of TMS320F2800137 and MSPM0 Daughterboard**

The transfer function of this current sampling circuit and the calculation for single shunt are the same as the three shunts.

#### 2.4.2.4.2 Motor Voltage Feedback

Voltage feedback is needed in the FAST estimator to allow the best performance at the widest speed range, the phase voltages are measured directly from the motor phases instead of a software estimate. The eSMO relies on software estimation values to represent the voltage phases without using the motor phase voltage-sensing circuit. This software value (USER\_ADC\_FULL\_SCALE\_VOLTAGE\_V) depends on the circuit that senses the voltage feedback from the motor phases. Figure 2-35 shows how the motor voltage is filtered and scaled for the ADC input range using a voltage feedback circuit based on resistor dividers. The similar circuit is used to measure all three of both compressor and fan motors, and dc bus.

The maximum phase voltage feedback measurable by the microcontroller in this reference design can be calculated as given in Equation 72, considering the maximum voltage for the ADC input is 3.3V.

$$V_{FS} = V_{ADC\_FS} \times G_V = 3.3 \text{ V} \times 122.46 = 404.13 \text{ V} \quad (72)$$

where

- $G_V$  is attenuation factor,  $G_V$  is calculated with Equation 73

$$G_V = \frac{(R_{62} + R_{67} + R_{70} + R_{74})}{R_{74}} = \frac{(332 \text{ k}\Omega + 332 \text{ k}\Omega + 332 \text{ k}\Omega + 8.2 \text{ k}\Omega)}{8.2 \text{ k}\Omega} = 122.46 \quad (73)$$

With that voltage feedback circuit, the following setting is done in user\_mtr1.h:

```
/*! \brief Defines the maximum voltage at the AD converter
#define USER_M1_ADC_FULL_SCALE_VOLTAGE_V (404.1292683f)
```

The voltage filter pole is needed by the FAST estimator to allow an accurate detection of the voltage feedback. Make the filter low enough to filter out the PWM signals, and at the same time allow a high-speed voltage feedback signal to pass through the filter. As a general guideline, a cutoff frequency of a few hundred Hz is enough to filter out a PWM frequency of 5 to 20kHz. Change the hardware filter only when ultra-high-speed motors are run, which generate phase-voltage frequencies in the order of a few kHz.

In this reference design the filter pole setting can be calculated with Equation 76:

$$f_{\text{filter\_pole}} = \frac{1}{(2 \times \pi \times R_{\text{Parallel}} \times C)} = 405.15 \text{ Hz} \quad (74)$$

where

$$C = 47 \text{ nF} \quad (75)$$

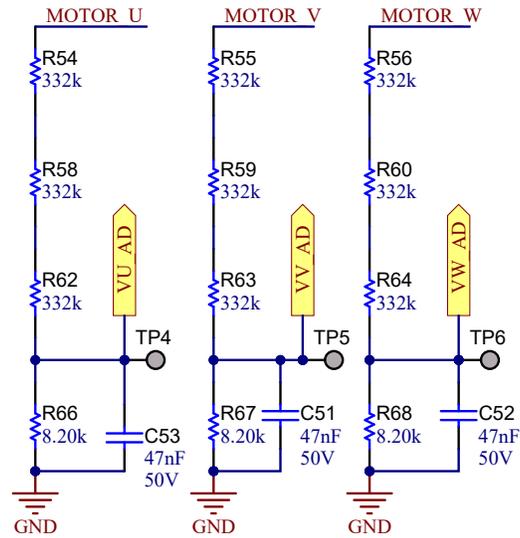
and

$$R_{\text{Parallel}} = \frac{((332 \text{ k}\Omega + 332 \text{ k}\Omega + 332 \text{ k}\Omega) \times 8.2 \text{ k}\Omega)}{(332 \text{ k}\Omega + 332 \text{ k}\Omega + 332 \text{ k}\Omega) + 8.2 \text{ k}\Omega} = 8.133 \text{ k}\Omega \quad (76)$$

The following code example shows how this is defined in user\_mtr1.h:

```

//! \brief Defines the analog voltage filter pole location, Hz
#define USER_M1_VOLTAGE_FILTER_POLE_HZ (416.3602877f)
  
```



**Figure 2-35. Motor Voltage Sensing Circuit**

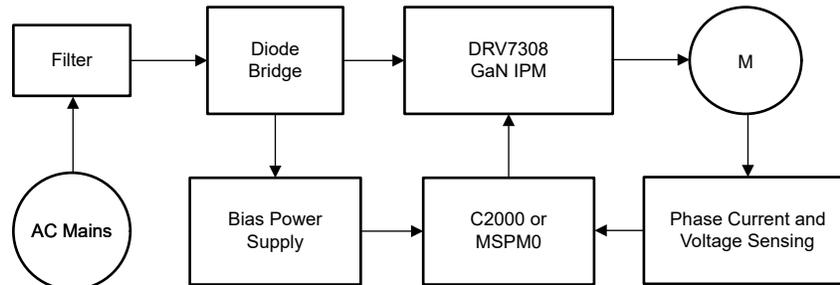
## 3 Hardware, Software, Testing Requirements, and Test Results

### 3.1 Getting Started Hardware

This section details the necessary equipments, test setups, and procedure instructions for the reference design board and software testing and validation.

#### 3.1.1 Hardware Board Overview

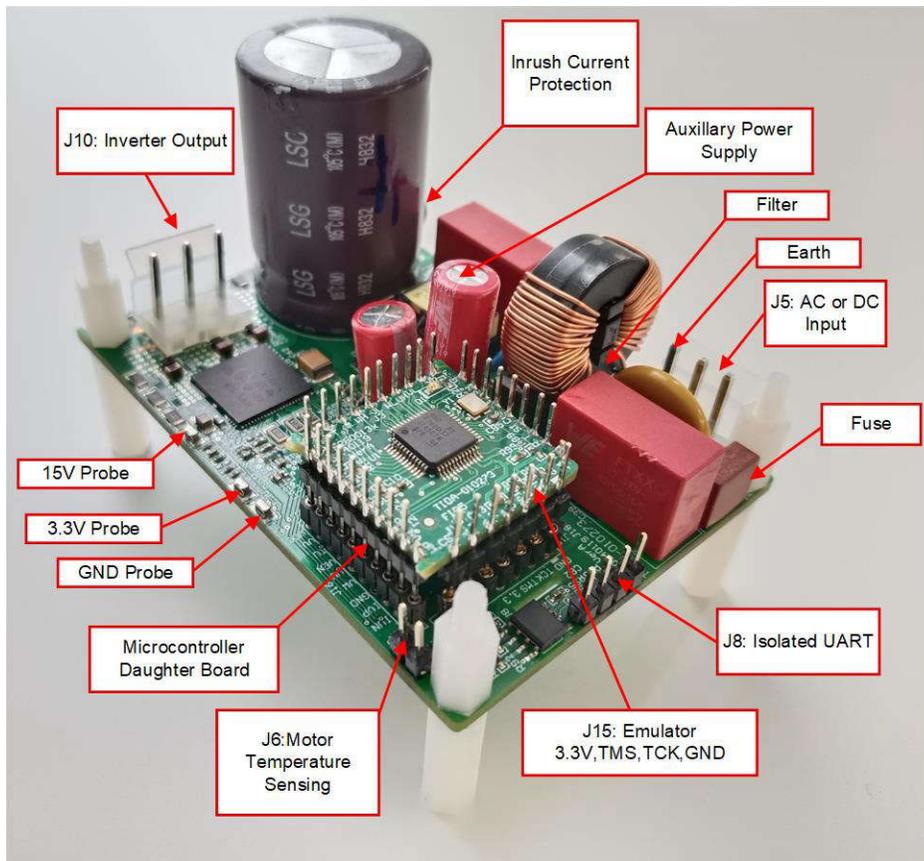
Figure 3-1 shows an overview of a typical motor inverter system.



**Figure 3-1. Hardware Board Block Diagram of TIDA-010273**

The reference board has functional groups that enable a complete motor drive system. The following is a list of the blocks (and functions) on the board, Figure 3-2 shows the top view of the board and different blocks of the TIDA-010273 board.

- Power line input filter
- 3-phase inverter
  - Up to 250W 3-phase inverter supports PMSM or IPM
  - 15kHz switching frequency
  - 1-3 shunts current sensing
- Control
  - Single TMS320F2800137 or MSPM0G1507 series MCU in 48-pin LQFP package
  - Amplify and input filters for the analog signals
- Interface for external motor temperature sensing
- Isolated UART port
- Auxiliary power supply
  - Onboard power supply +3.3V, and +15V



**Figure 3-2. TIDA-010273 Reference Design Board Layout**

TI recommends taking the following precautions when using the board:

**WARNING**

- Do not touch any part of the board or components connected to the board when the board is energized.
- Use the AC Mains or wall power supply to power the kit. TI recommends an isolation AC source.
- Do not touch any part of the board, the kit or the assembly when energized. (Though the power module heat sink is isolated from the board, high-voltage switching generates some capacitive coupled voltages over the heat sink body.)
- Control Ground can be hot.

**3.1.2 Test Conditions**

Observe the following for testing the reference design software:

- For input, the power supply source must range from 120V to 265V AC if using the AC source, or must range from 160V to 400V DC if using the DC power supply. Set the input current limit of input AC source to 3A or DC power supply to 1.5A, but start with a lower current limit during initial board bring up.
- For the output, use a 3-phase PMSM with dynamo meters.

**3.1.3 Test Equipment Required for Board Validation**

The designer must use the following equipment for board validation:

- Isolated AC or DC source
- Three-phase power analyzer
- Digital oscilloscope
- Multimeters
- DC power supply

- 250W, 3-phase PM synchronous motors
- Dynamo meters
- Three-phase power analyzer
- USB-to-UART adaptor

### 3.2 Getting Started GUI

Source code for this reference design is provided so the designer can debug firmware directly, as explained in [Section 3.3](#). However, software debugging needs more time. To speed up development, a UART based GUI software is provided to help quickly tune parameters on any customized application. This section introduces how to debug and tune the motor control parameters with the GUI software.

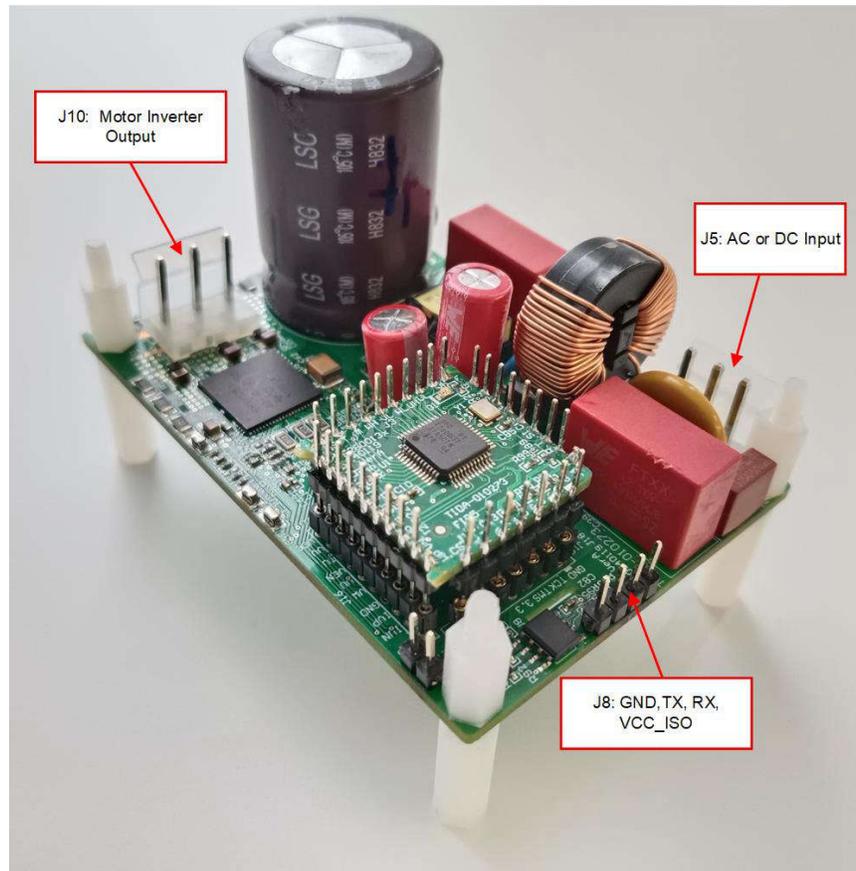
For now, only the C2000 daughter-board firmware supports this GUI.

When connecting the host PC to this reference board by UART at J8, since on board UART is an isolated UART port, UART needs to be supplied by external power supply.

#### 3.2.1 Test Setup

The GUI software needs only a UART connection between the host PC and the reference design board. [Figure 3-3](#) shows the hardware connection for testing with the GUI. Setup the hardware using the following steps:

1. Connect GND, TX, RX, and 3.3V or 5V to VCC\_ISO at J8 to the host PC through a USB-to-UART adapter.
2. Connect motor wires to J10.
3. Connect the multimeter, oscilloscope probes, and other measurement equipment to probe or analyze various signals and parameters
4. Power on the board with a DC bus power, AC power supply or AC mains power to the board at J5.
  - a. The maximum output of the DC power supply is 380VDC.
  - b. The maximum output of the AC power supply is 265VAC, 50/60Hz.
  - c. AC main power is 220VAC, 50/60Hz.



**Figure 3-3. Hardware Connection for Testing With GUI Software**

**WARNING**

The ground planes of both the power domains can be the same or different depending on the hardware configuration. Meet proper isolation requirements before connecting any test equipment with the board for personal safety and to prevent damaging equipment. Review the GND connections before powering the board. An isolator is required if measurement equipment is connected to the board.

### 3.2.2 Overview of GUI Software

The GUI software can be run on a Microsoft® Windows® based system. The GUI has 6 tabs: Control Window, Debug Windows, Control Parameters, Motor Parameters, System Parameters, and Communication Setting, as shown in Figure 3-4. The Analysis Windows is not available in this version GUI. Those tabs provides multiple functions, such as motor control, identification, control parameters tuning, virtual oscilloscope, read and write MCU flash, and so forth.

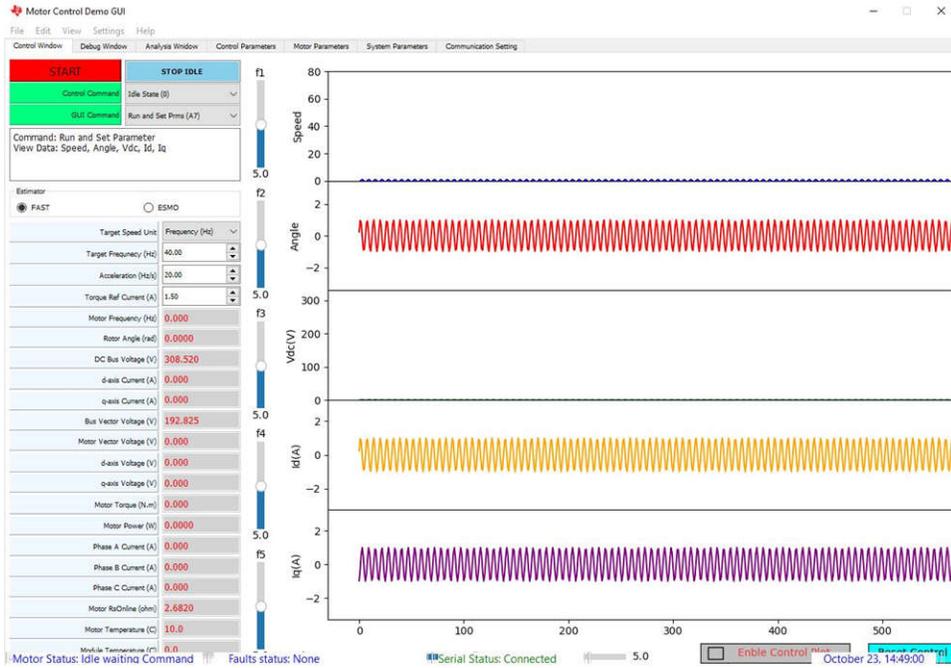
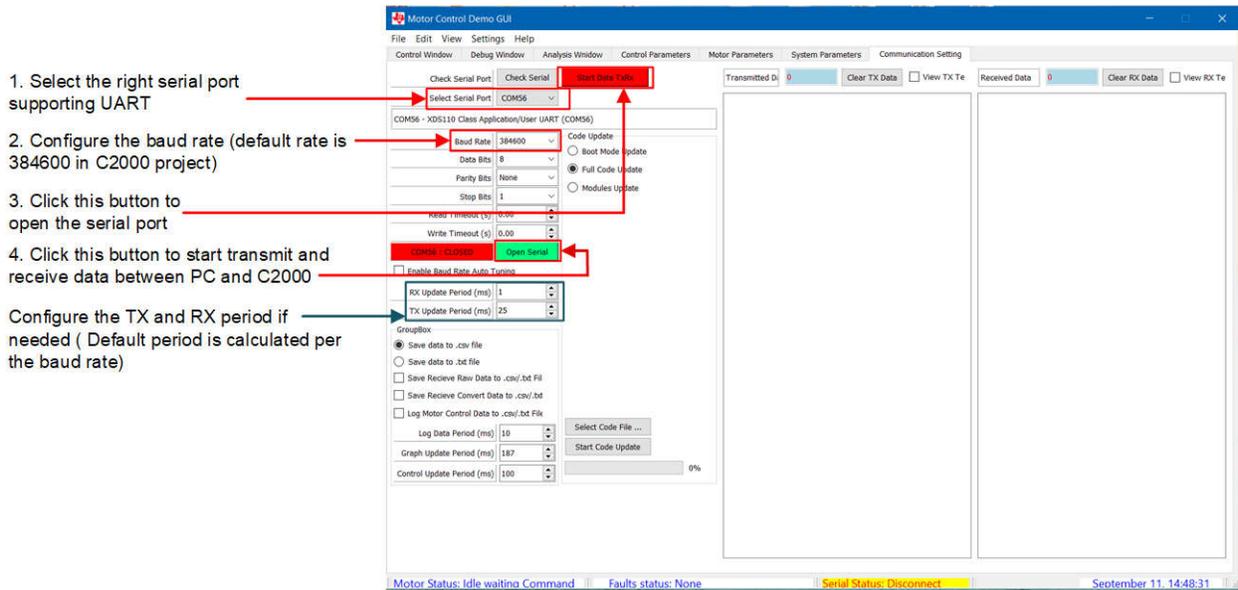


Figure 3-4. GUI Software

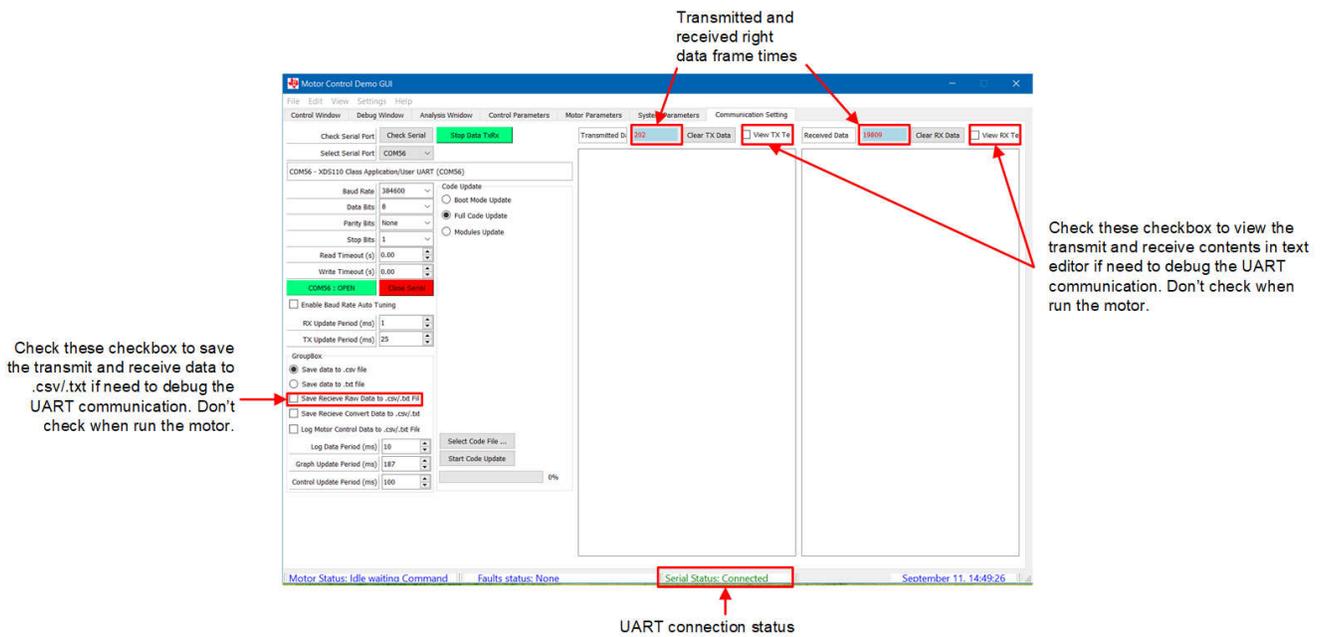
### 3.2.3 Setup Serial Port

Run the GUI software on the host PC. Wait for GUI window to pop up. Figure 3-5 shows the steps to connect the host PC to this reference design board via UART. The default baud rate is 384600bps. The user must change the baud rate setting on both GUI and C2000 if the user wants to use the different UART speed.



**Figure 3-5. Setup Serial Port**

A successful connection and communication can be verified by the number of *Transmitted Data* and *Received Data*. Figure 3-6 shows that both numbers increase continuously with a successful connection. Those transmitted and received data can be viewed by checking *View Tx Text* and *View Rx Text*; however, do not check these features to view when the motor is running.



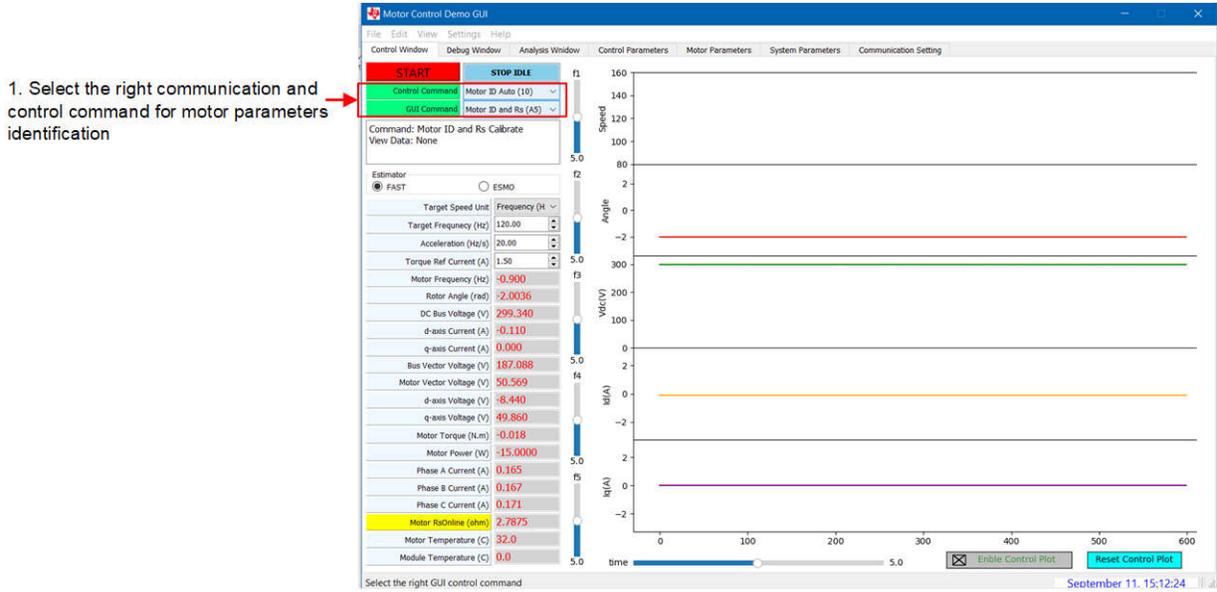
**Figure 3-6. Successful Communication Status**

### 3.2.4 Motor Identification

Implementing the correct motor parameters is critical for the firmware to control the motor successfully. The parameters include Stator resistance, Stator inductance, flux, and so forth, and those parameters have a default value for a default motor inside the firmware.

For a different PMSM motor, those parameters are usually found from the specification; however, if those parameters cannot be found, the GUI software can identify those parameters.

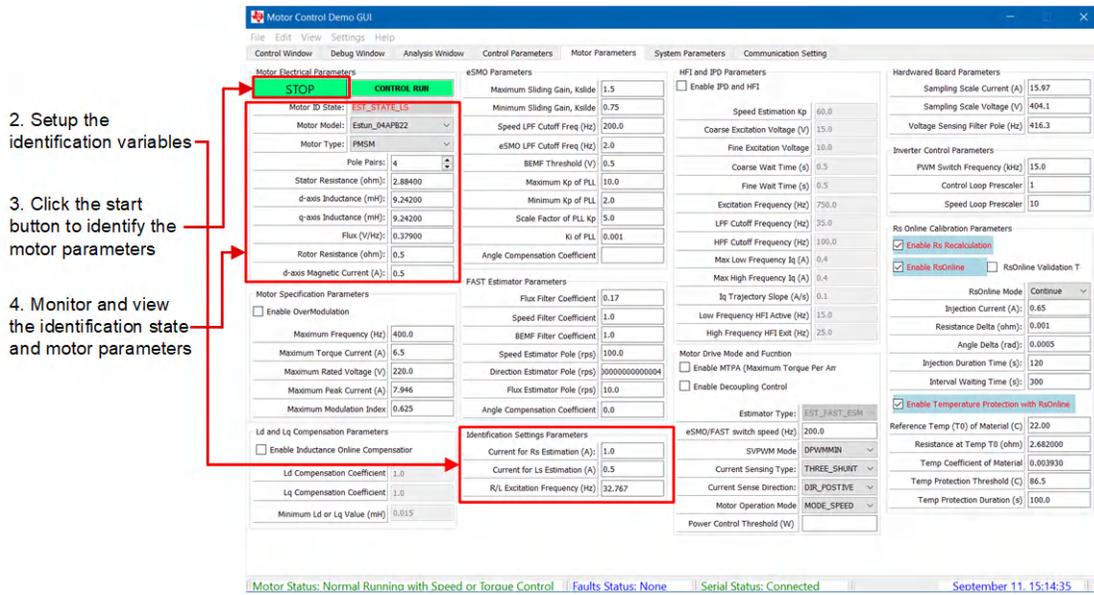
First, choose the motor identification command in the *Control Window* tab as shown in Figure 3-7.



**Figure 3-7. Motor Identification Command**

Next, select the *Motor Parameters* tab. The motor identification applies current to the motor to estimate motor parameters, those identification parameters, such as current for stator resistor estimation, current for stator inductor estimation and *R/L Excitation Frequency (Hz)* can be changed or left on the default settings.

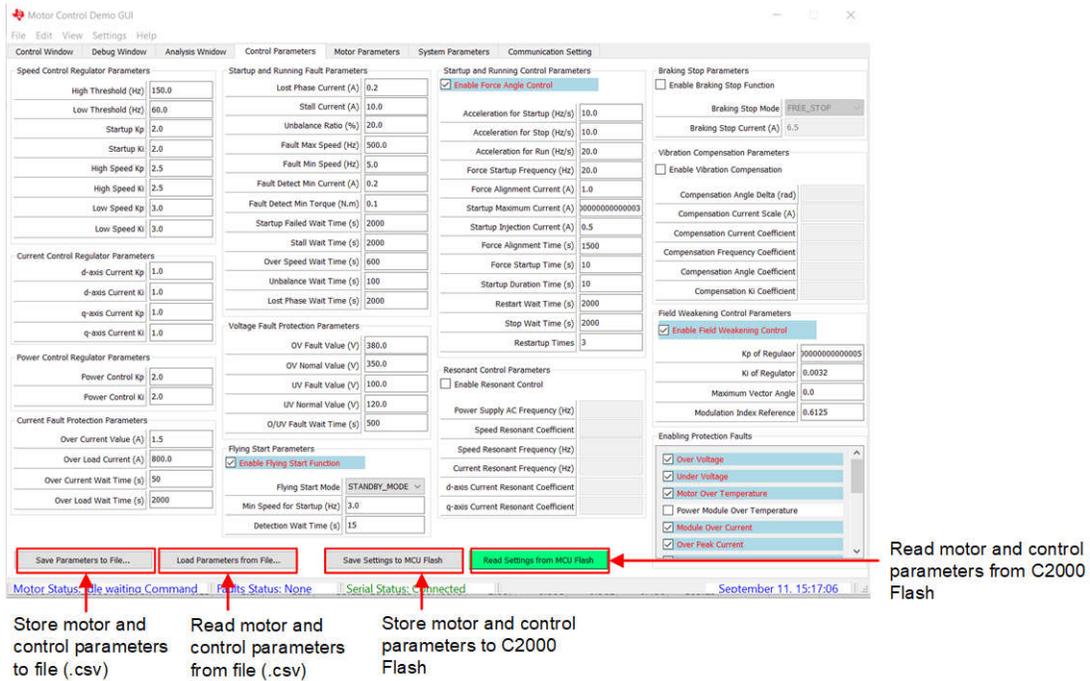
Click the *START* button to start motor identification. An audible noise is heard, and the motor spins at low speed during identification. Monitor the identification status and motor parameters, the overall identification time is about 2 minutes. Figure 3-8 shows the steps to start motor identification.



**Figure 3-8. Start Motor Identification**

After identification is complete, the Motor Pairs, Stator resistance, Stator inductance, flux, and so forth, must be written to MCU flash to make sure those parameters are saved inside the MCU. Select the *Control Parameters* tab, choose to store motor and control parameters to MCU Flash, or save them to a file. To verify writing is successful, click the *Read Settings from MCU Flash* button, then select the *Motor Parameters* tab to make sure

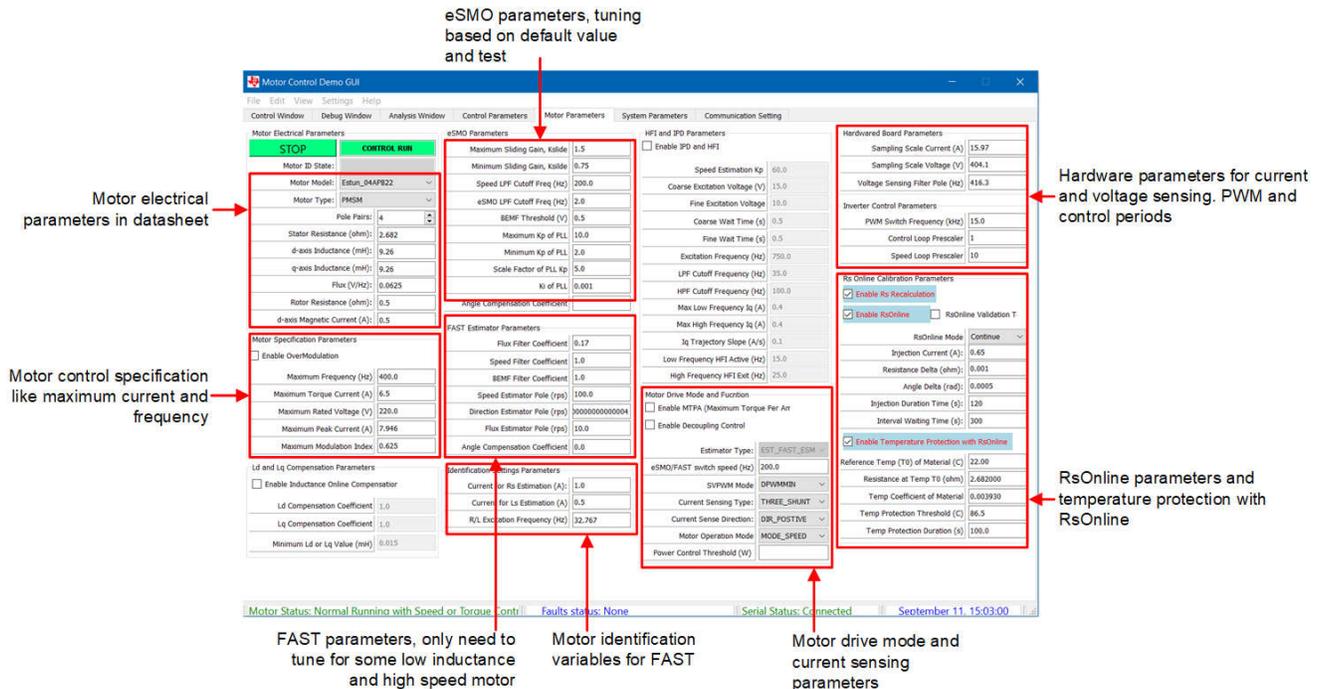
motor parameters are the same as what were written before. Figure 3-9 shows the locations of the buttons mentioned in this paragraph.



**Figure 3-9. Store Motor Identification Result**

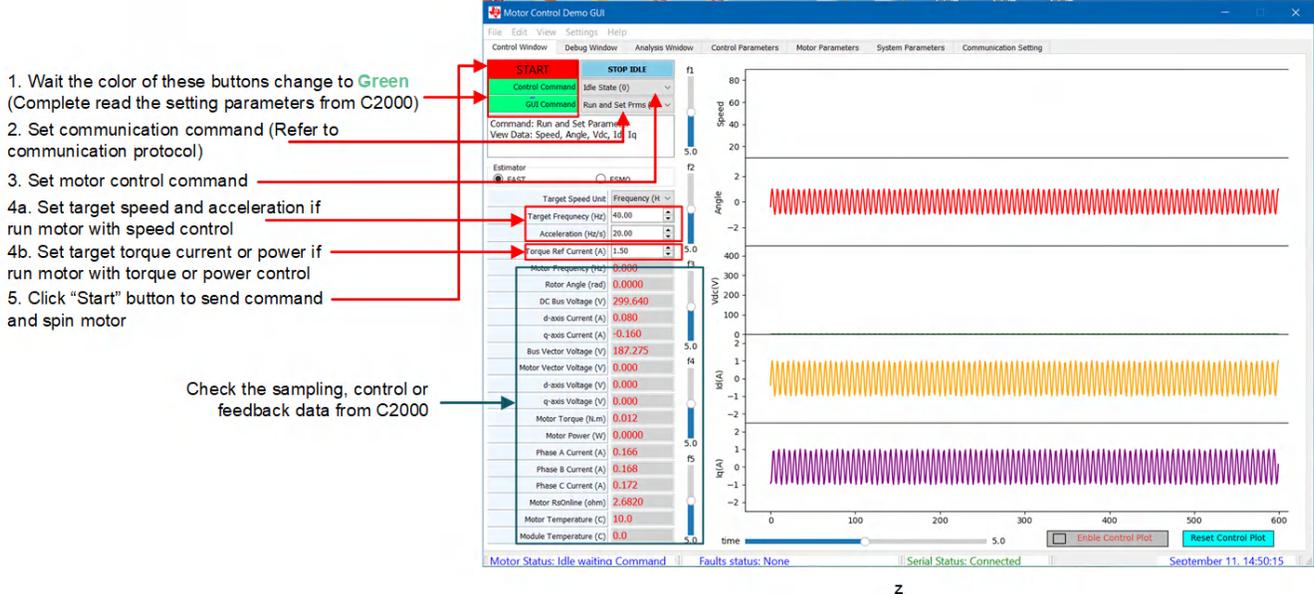
**3.2.5 Spin Motor**

Figure 3-10 shows the *Motor Parameters* tab indicating the motor and control parameters. Make sure the motor electrical parameters are correct.



**Figure 3-10. Motor Drive Parameters**

On the *Control Window* tab, if the DC BUS voltage is high enough (> 230 VDC), the *Control Command* and *GUI Command* button are in green, the motor spin commands are effective. Follow the steps in [Figure 3-11](#) to spin the motor.



**Figure 3-11. Steps to Spin Motor**

### 3.2.6 Motor Fault Status

Faults can be found while the motor is spinning, especially when motor parameters are incorrect or the motor control parameters are not well tuned. Watch the *Control Window* carefully, and pay attention to any faults reported as shown in [Figure 3-12](#).

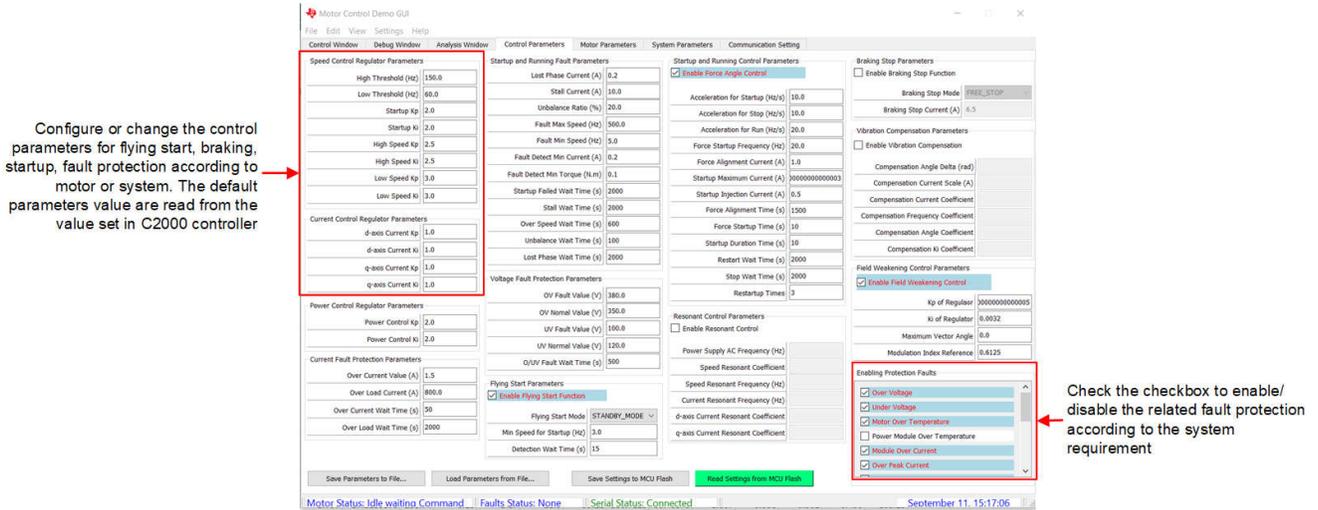


**Figure 3-12. Fault Status Monitor**

Faults detection can be enabled or disabled as shown in Figure 3-13.

**CAUTION**

Disabling some faults can cause the board can break. Do not disable fault protections without full knowledge of acceptable settings.



Configure or change the gain of PI regulators for speed, current, and power control. The values are coefficient, the final gains are these coefficient multiply the setting value in C2000 controller. The setting gains are calculated per the motor electrical parameters.

**Figure 3-13. Motor Control Parameters and Faults Detection Disable**

### 3.2.7 Tune Control Parameters

On the *Debug Window* tab, the PI regulators can be tuned for speed, current, and power control as shown in Figure 3-14. The values are coefficient, the final gains are that these coefficients multiply the setting value in the C2000 controller. The setting gains are calculated per the motor electrical parameters. As Figure 3-9 shows, those tuned values can be written to MCU flash.

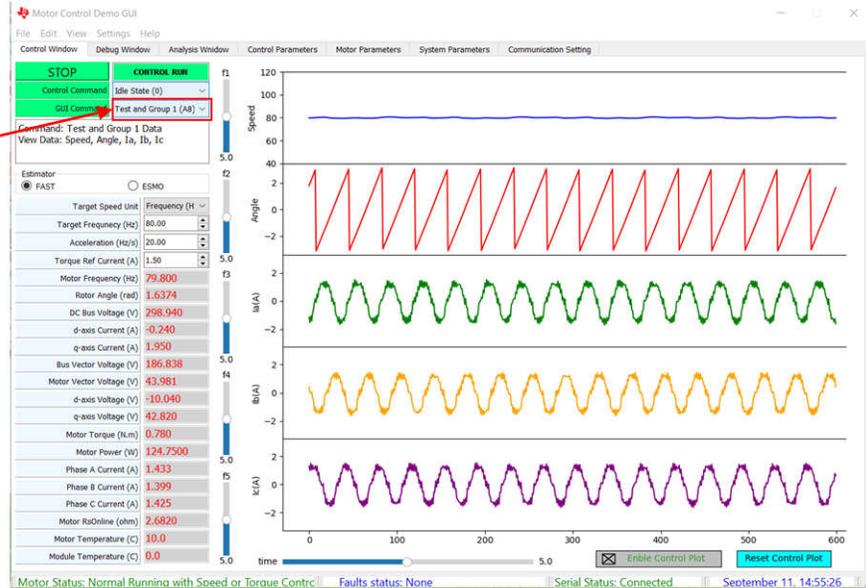


**Figure 3-14. Tune Control Parameters**

### 3.2.8 Virtual Oscilloscope

The GUI software has a virtual oscilloscope function, and can show the waveform of angle, phase current, and phase voltage. Figure 3-15 shows how to configure the command to show the rotor angle and phase current.

Using "Test and Group 1 (A8)" communication command to view the feedback speed, rotor angle and three-phase current (Ia, Ib, and Ic). The target speed and acceleration can't be changed in this mode.

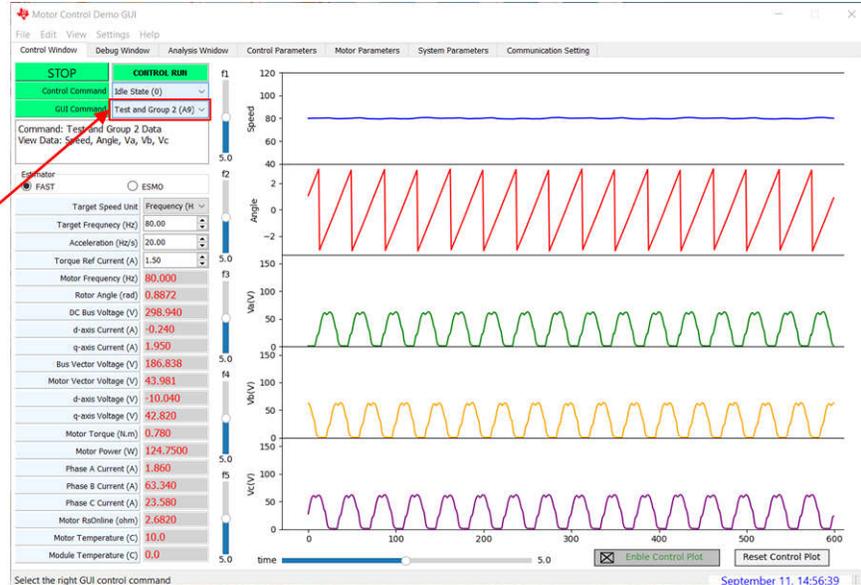


The graph quality is dependent on the maximum baud rate supporting by the hardware board. Much higher baud rate much better graph view

Figure 3-15. Virtual Oscilloscope for Rotor Angle and Phase Current

Figure 3-16 illustrates the command *Test and Group 2 (A9)* and shows the rotor angle and phase voltages.

Using "Test and Group 2 (A9)" communication command to view the feedback speed, rotor angle and three-phase voltage (Va, Vb, and Vc). The target speed and acceleration can't be changed in this mode.

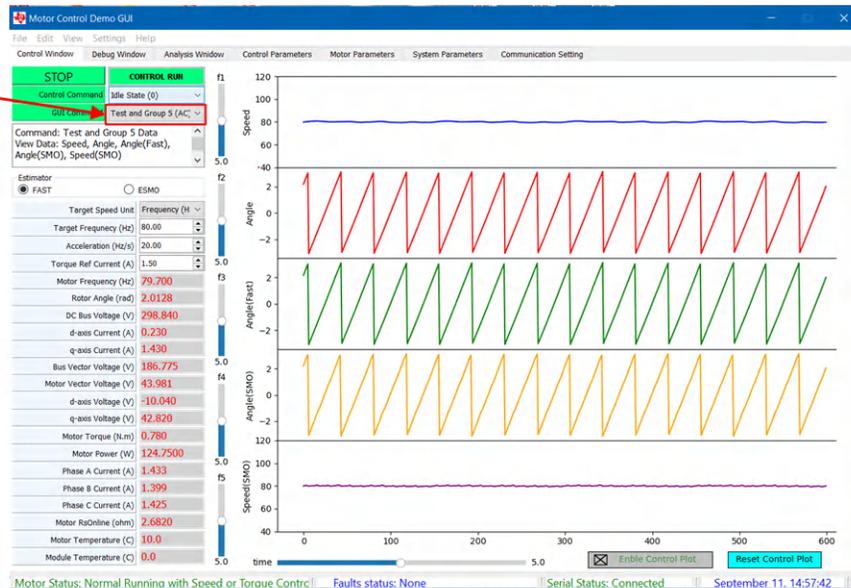


The graph quality is dependent on the maximum baud rate supporting by the hardware board. Much higher baud rate much better graph view

Figure 3-16. Virtual Oscilloscope for Rotor Angle and Phase Voltages

Command *Test and Group 5 (AC)* can show the rotor angle of FAST and EMO as shown in Figure 3-17.

Using "Test and Group 5 (AC)" communication command to view the rotor angle and speed from FAST or eSMO. The target speed and acceleration can't be changed in this mode.



The graph quality is dependent on the maximum baud rate supporting by the hardware board. Much higher baud rate much better graph view

Figure 3-17. Virtual Oscilloscope for Rotor Angle of FAST and eSMO

### 3.3 Getting Started C2000 Firmware

Download and install [C2000WARE-MOTORCONTROL-SDK v5.03.00.00](#) or newer software from the link provided by TI. Install this Motor Control SDK software in the default folder. The software project then resides inside the C2000Ware Motor Control SDK folder at <install\_location>\solutions\tida\_010273\_GaNIInv\. Follow these steps to build and run this code with different incremental builds.

#### 3.3.1 Download and Install Software Required for Board Test

1. Download and install Code Composer Studio™ IDE from the [Code Composer Studio \(CCS\) Integrated Development Environment \(IDE\)](#) tools folder. Version 12.5 or newer is recommended.
2. Install C2000WARE-MOTORCONTROL-SDK in one of two ways:
  - Download the software through the [C2000Ware MotorControl SDK](#) tools folder
  - Go to CCS and under *View* → *Resource Explorer*. Under the TI Resource Explorer, go to *Software* → *C2000Ware\_MotorControl\_SDK*, and click the install button.
3. Once installation is complete, close CCS, and create a new workspace for importing the project.

#### Note

This reference design supports SysConfig for configuring device pins and initializing device peripherals in an easy-to-use graphical interface. This feature is just for a reference in the current release. The designer can download [SysConfig](#), and refer to [C2000 SysConfig Software Guide](#) to implement the SysConfig to migrate the reference design to the board for configuring the device.

### 3.3.2 Opening Project Inside CCS

The projectspec file for F280013x based reference design is in the directory below<install\_location>\solutions\tida\_010273\_GaNIInv\f280013x\ccs\motor\_control

Import the project within CCS and select the right build configurations by right-clicking on project name as shown in Figure 3-18. Select the right build configuration for the HVAC reference design. The *Flash\_MtrInv\_3SC* build configuration supports the three-shunt current sensing method, *Flash\_MtrInv\_1SC* supports the single-shunt current sensing method.

Configure the project to select the supporting functions in the project by clicking *Project* → *Import CCS Projects*, and browse to <install\_location>\solutions\tida\_010273\_GaNIInv\f280013x\ccs\motor\_control for F280013x based reference design, and then right-click on the imported project name, click *Properties* command to set the pre-define symbols for the project as shown in Figure 3-19.

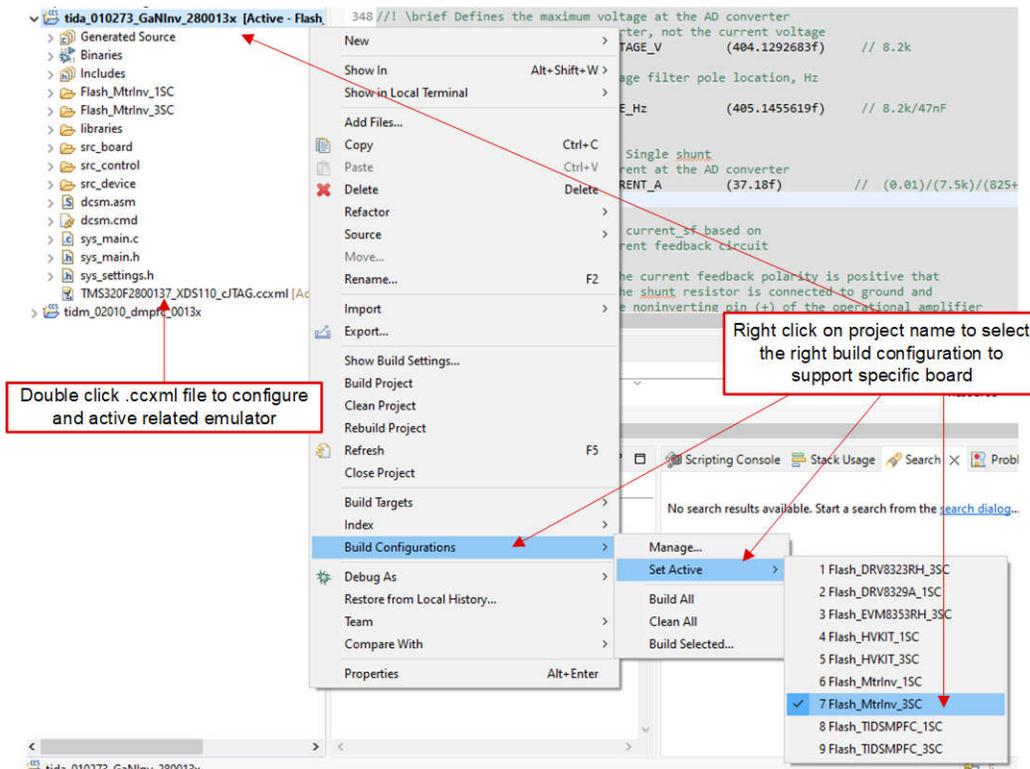


Figure 3-18. Select the Correct Build Configuration

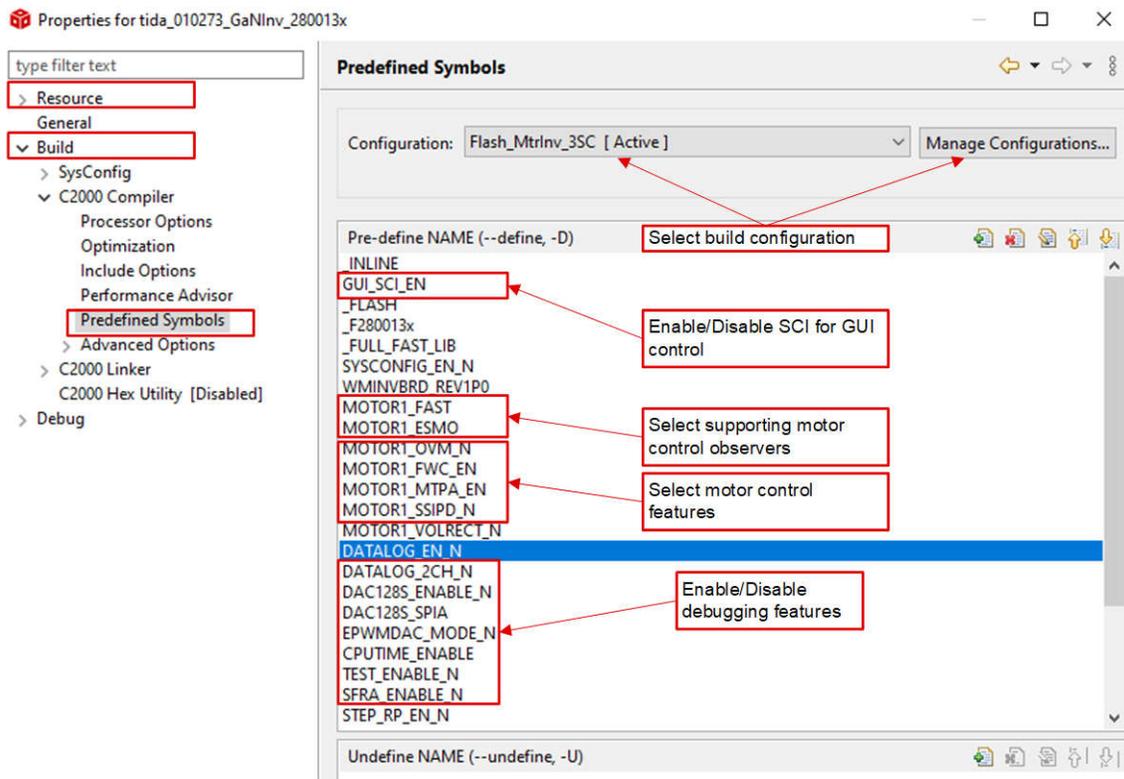


Figure 3-19. Select the Correct Predefined Symbols in Project Properties

### 3.3.3 Project Structure

Once the project is imported, the project explore appears inside CCS as shown in Figure 3-20. The device peripherals configuration is based on C2000Ware driverlib. The users only needs to change the codes and definitions in *hal.c* and *hal.h*.

The folder *src\_control* includes *hal.c* and *user\_mtr1.c*, in which users can change codes and definitions.

The folder *src\_board* includes board drivers for this hardware board.

The folder *src\_control* includes motor drive files that call motor control core algorithm functions within the interrupt service routines and background tasks.

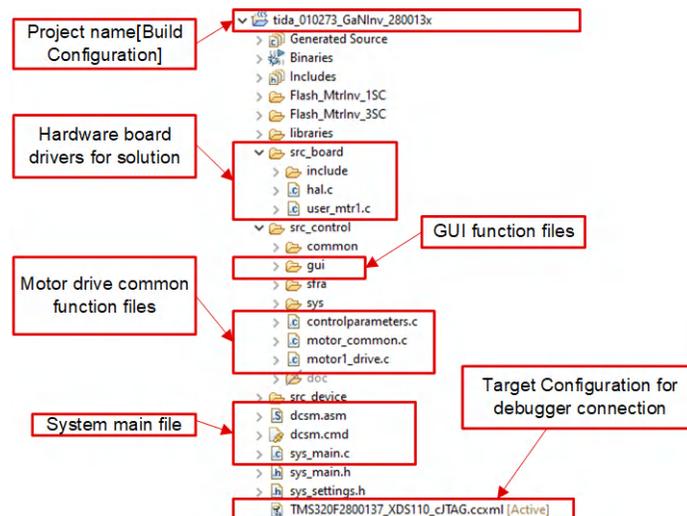
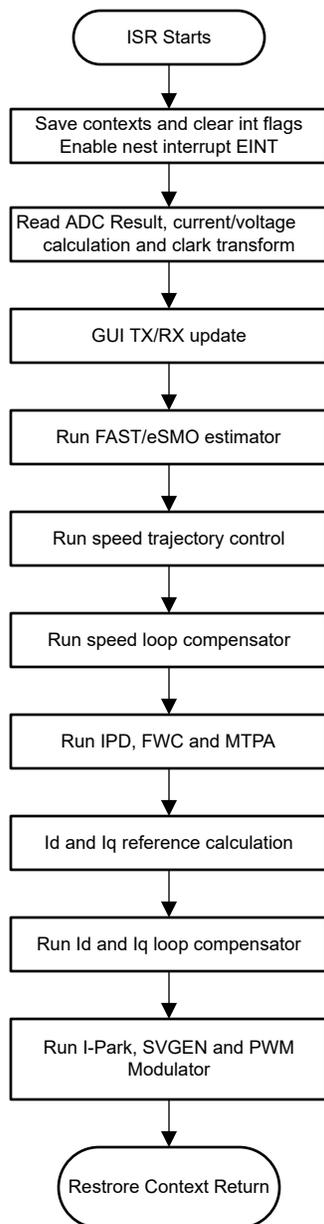


Figure 3-20. TIDA-010273 Project Explorer View

Figure 3-21 shows the project software flow diagram of ISR for motor control, a main loop for motor control parameters update in background loop.



**Figure 3-21. Firmware Project Flow Diagram**

The project consists of a motor control interrupt service routine, which are called every PWM cycle. A few background tasks are called in a loop forever in main() and can be used to run slow tasks for which absolute timing accuracy is not required, motor control parameters update, and so on. A CPU timer is used to trigger slow background tasks.

*motor1CtrlISR* is reserved for calling the motor drive control algorithms to spin the motor that is periodically triggered at USER\_M1\_ISR\_FREQ\_Hz.

To simplify the system, the bring up and design of the software for this reference design is organized in four labs with incremental builds (DMC\_BUILDLEVEL), which makes learning and getting familiar with the board and software easier. This approach is also good for debugging and testing boards. Table 3-1 lists the detailed incremental build options. To select a particular build option, select the corresponding BUILDLEVEL option in

*sys\_settings.h*. Once the build option is selected, compile the project by selecting the *rebuild all* compiler option. [Section 3.3.4](#) provides more details to run each of the build level options.

**Table 3-1. Incremental Build Options**

OPERATION	BUILD OPTION	DESCRIPTION
MOTOR DRIVE	DMC_LEVEL_1	50% PWM duty, verify ADC offset calibration, PWM output and phase shift
	DMC_LEVEL_2	Open-loop v/f control to check current and voltage sensing signals for motor
	DMC_LEVEL_3	Closed current loop to check the hardware settings
	DMC_LEVEL_4	Motor parameters identify and run with InstaSPIN-FOC or eSMO

### 3.3.4 Test Procedure

#### WARNING

There are **high voltages** present on the board. To safely evaluate this board, use an appropriate isolated and current limited power source. Before power is applied to the board, an appropriate resistive or electronic load must be connected at the output. Do not handle the unit when power is applied. Only use appropriately rated equipment and follow proper isolation and safety practices.

#### CAUTION

Use caution when connecting scopes and other test equipment to the board because the AC rectifier generates the DC-output voltage, which has a **HOT Ground** floating from the protective earth ground. Isolation transformers must be used when connecting grounded equipment to the kit.

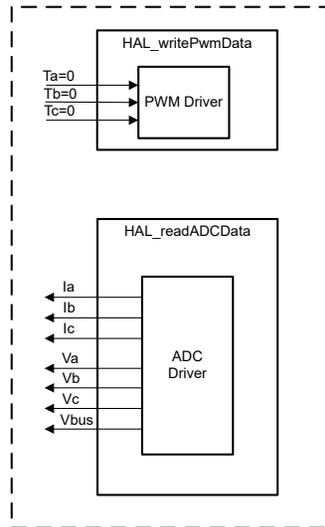
#### 3.3.4.1 Build Level 1: CPU and Board Setup

Objectives learned in this build level:

- Evaluate the open-loop operation of the system
- Use the HAL object to set up the MCU controller and initialize the inverter
- Verify the PWM and ADC driver modules
- Become familiar with the operation of CCS

Because this system is running with open-loop control, the ADC measured values are only used for instrumentation purposes in this build level. Only bias power supply for MCU controller and gate drivers is used in this build level. The high-voltage AC and DC power supply are not implemented on the inverter.

In this build level, the board is executed in open-loop fashion with a fixed duty cycle. The duty cycles are set to 50% for the motor. This build level verifies the sensing of feedback values from the power stage and also operation of the PWM gate driver and makes sure there are no hardware issues. Additionally calibration of input and output voltage sensing can be performed in this build level. The software flow for this build level is shown in [Figure 3-22](#).

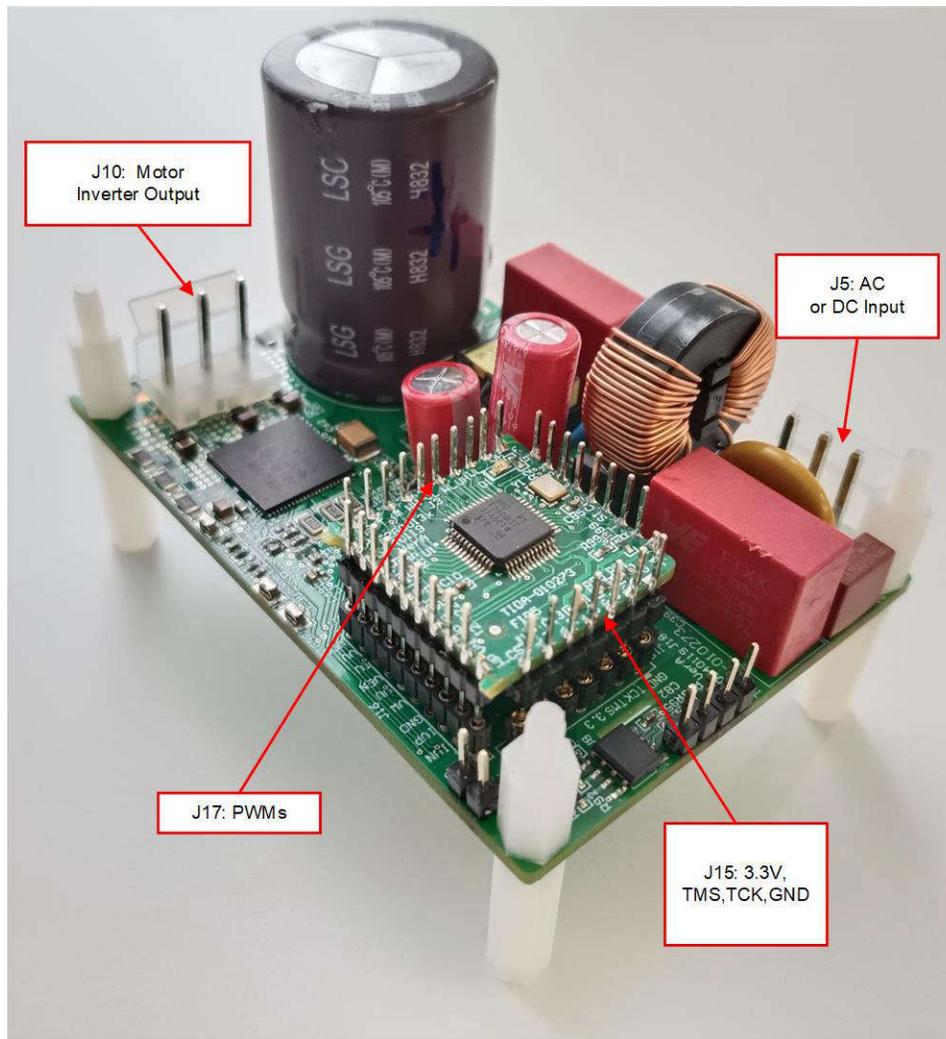


**Figure 3-22. Control Software Block Diagram: Build Level 1 – Offset Validation**

#### 3.3.4.1.1 Start CCS and Open Project

To start CCS and open the project, complete the following steps:

1. Connect the emulator at J15.
2. Connect AC or DC power supply to J5 as shown in [Figure 3-23](#).
3. Open CCSv12.5 (or newer). A project contains all the files and build options needed to generate an executable output file (.out), which can be run on the C2000 controller-based hardware. On the menu bar, click *Project* → *Import CCS Projects*. Below *Select search-directory*:, browse to the C2000Ware Motor Control SDK folder and select <install\_location>\solutions\tida\_010273\_GaNInv . Click *Finish* to import the related project into CCS. This project invokes all the necessary tools (compiler, assembler, linker) to build the project.
4. In the project window on the left, click the plus sign (+) to the left of Project. An example project window is shown in [Figure 3-20](#).



**Figure 3-23. Connect the External AC or DC Power Source to Verify the Hardware**

#### 3.3.4.1.2 Build and Load Project

To build and load the project, complete the following steps:

1. Right-click on the project name, click the *Properties* command, move to pre-defined symbols to change `GUI_SCI_EN` to `GUI_SCI_N` to disable the SCI function for the GUI as shown in [Figure 3-19](#).

#### Note

The SCI function can be enabled again to allow GUI control through SCI, as described in [Section 3.2](#).

2. Open the `sys_settings.h` file, set `DMC_BUILDLEVEL` to `DMC_LEVEL_1`.
3. If another build option was built previously, right click on the project name and click on *Clean Project*, and then click on *Build Project*. Watch the tools run in the build window. The project builds successfully.
4. In the *Project Explorer* make sure the correct target configuration file is set as Active as shown in [Figure 3-20](#).
5. Turn on the AC or DC power supply to apply 120VAC or 160 VDC to J5, to create the +15 V and 3.3 V for

the DRV7308 and microcontroller. Click on the Debug button  or click *Run* → *Debug*. The build level 1 code can be compiled and loaded on the C2000 device. Notice the CCS Debug icon in the upper right-hand corner, indicating that the user is now in the *Debug Perspective* view. The program can be stopped at the start of `main()`.

### 3.3.4.1.3 Setup Debug Environment Windows

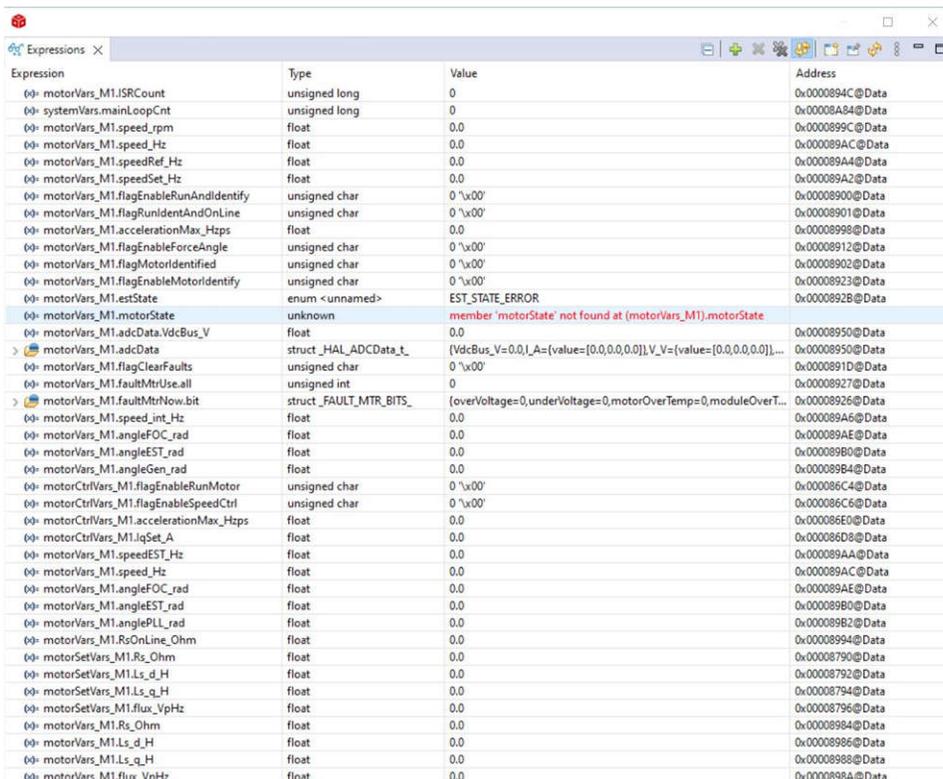
To watch local and global variables while debugging code is a standard debug practice. There are various methods for doing this in CCS, such as memory views and watch views. Additionally, CCS has the ability to make time (and frequency) domain plots. This ability allows the user to view waveforms using the graph tool.

1. Click **View** → **Expressions** on the menu bar to open an **Expressions** watch window. Move the mouse to the **Expressions** window to view the variables being used in the project. Add variables to the **Expressions** window as shown in **Figure 3-24**. The window uses the number format associated with variables during declaration, and **Figure 3-24** shows an example of the **Expressions** window. Select a desired number format for the variable by right clicking on expressions and choosing.
2. Alternately, a group of variables can be imported into the **Expressions** window by right clicking within the **Expressions** window and clicking **Import**, and browse to the directory of the project at `<install_location>\solutions\tida_010273_GaNInv\src_control\common\debug` and pick **BuildLevel1.txt** and click the **OK** button to import the variables shown in **Figure 3-24**.

#### Note

Some of the variables have not been initialized at this point in the main code and can contain some useless values.

3. The structure variables `motorVars_M1[]` have references to most variables that are related to controlling the motor. By expanding this variable, you can see and edit all the variables, as needed.
4. Click on the **Continuous Refresh** button  in the expressions window. This enables the window to run with real-time mode. By clicking the down arrow in this **Expressions** window, you can select **Customize Continuous Refresh Interval** and edit the refresh rate of the expressions window. Choosing too fast an interval can affect performance.



Expression	Type	Value	Address
motorVars_M1.ISRCount	unsigned long	0	0x0000894C@Data
systemVars.mainLoopCnt	unsigned long	0	0x00008A84@Data
motorVars_M1.speed_rpm	float	0.0	0x0000899C@Data
motorVars_M1.speed_Hz	float	0.0	0x000089AC@Data
motorVars_M1.speedRef_Hz	float	0.0	0x000089A4@Data
motorVars_M1.speedSet_Hz	float	0.0	0x000089A2@Data
motorVars_M1.flagEnableRunAndIdentify	unsigned char	0 '\x00'	0x00008900@Data
motorVars_M1.flagRunIdentAndOnLine	unsigned char	0 '\x00'	0x00008901@Data
motorVars_M1.accelerationMax_Hzps	float	0.0	0x00008998@Data
motorVars_M1.flagEnableForceAngle	unsigned char	0 '\x00'	0x00008912@Data
motorVars_M1.flagMotorIdentified	unsigned char	0 '\x00'	0x00008902@Data
motorVars_M1.flagEnableMotorIdentify	unsigned char	0 '\x00'	0x00008923@Data
motorVars_M1.estState	enum <unnamed>	EST_STATE_ERROR	0x00008928@Data
motorVars_M1.motorState	unknown	member 'motorState' not found at (motorVars_M1).motorState	
motorVars_M1.adcData.VdcBus_V	float	0.0	0x00008950@Data
motorVars_M1.adcData	struct _HAL_ADCData_t	{VdcBus_V=0.0,La=[value={0.0,0.0,0.0}],V_V={value={0.0,0.0,0.0}}...	0x00008950@Data
motorVars_M1.flagClearFaults	unsigned char	0 '\x00'	0x0000891D@Data
motorVars_M1.faultMtrUse.all	unsigned int	0	0x00008927@Data
motorVars_M1.faultMtrNow.bit	struct_FAULT_MTR_BITS_	{overVoltage=0,underVoltage=0,motorOverTemp=0,moduleOverT...	0x00008926@Data
motorVars_M1.speed_int_Hz	float	0.0	0x000089A6@Data
motorVars_M1.angleFOC_rad	float	0.0	0x000089AE@Data
motorVars_M1.angleEST_rad	float	0.0	0x000089B0@Data
motorVars_M1.angleGen_rad	float	0.0	0x00008984@Data
motorCtrlVars_M1.flagEnableRunMotor	unsigned char	0 '\x00'	0x000086C4@Data
motorCtrlVars_M1.flagEnableSpeedCtrl	unsigned char	0 '\x00'	0x000086C6@Data
motorCtrlVars_M1.accelerationMax_Hzps	float	0.0	0x000086E0@Data
motorCtrlVars_M1.lqSet_A	float	0.0	0x000086D8@Data
motorVars_M1.speedEST_Hz	float	0.0	0x000089AA@Data
motorVars_M1.speed_Hz	float	0.0	0x000089AC@Data
motorVars_M1.angleFOC_rad	float	0.0	0x000089AE@Data
motorVars_M1.angleEST_rad	float	0.0	0x000089B0@Data
motorVars_M1.anglePLL_rad	float	0.0	0x000089B2@Data
motorVars_M1.RsOnLine_Ohm	float	0.0	0x00008994@Data
motorSetVars_M1.Rs_Ohm	float	0.0	0x00008790@Data
motorSetVars_M1.Ls_d_H	float	0.0	0x00008792@Data
motorSetVars_M1.Ls_q_H	float	0.0	0x00008794@Data
motorSetVars_M1.flux_VpHz	float	0.0	0x00008796@Data
motorVars_M1.Rs_Ohm	float	0.0	0x00008984@Data
motorVars_M1.Ls_d_H	float	0.0	0x00008986@Data
motorVars_M1.Ls_q_H	float	0.0	0x00008988@Data
motorVars_M1.flux_VpHz	float	0.0	0x0000898A@Data

**Figure 3-24. Build Level 1: Expressions Watch Window at Reset**

#### 3.3.4.1.4 Run the Code

To run the code, complete the following steps:

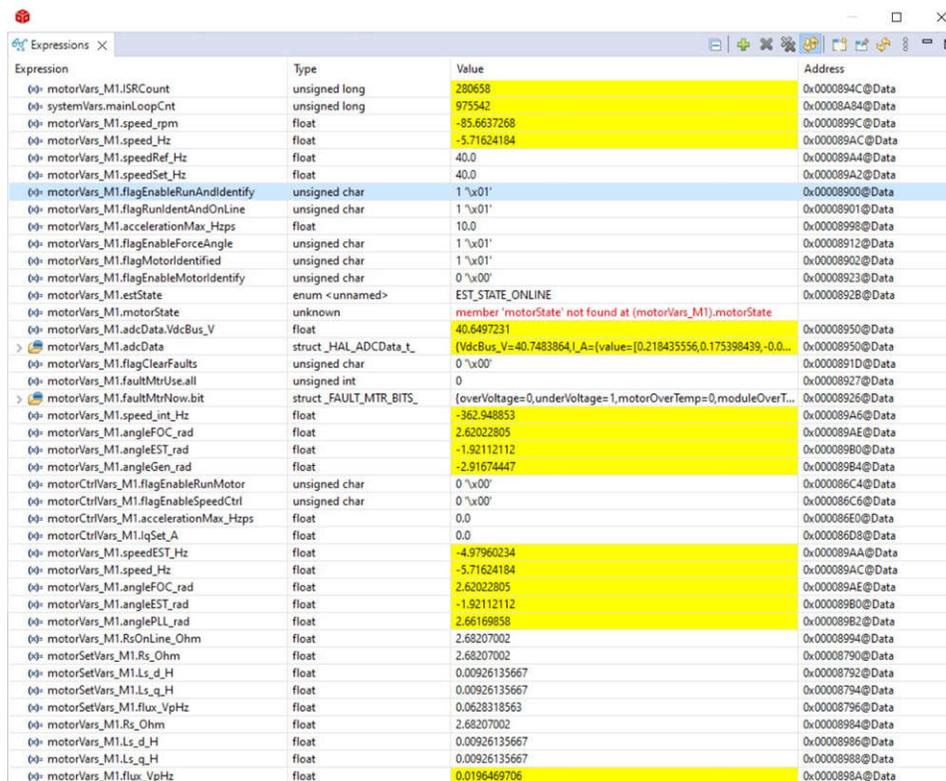
1. Run the project by clicking the button , or click **Run** → **Resume** in the **Debug** tab.

2. In the *Expressions* window, set the variables *motorVars\_M1.flagEnableRunAndIdentify* to "1" after *systemVars.flagEnableSystem* was automatically set to "1" in the watch window.
3. The project can now run, and the values in the graphs and *Expressions* window can continuously update as shown in [Figure 3-25](#) while using this project. The windows can be resized according to user preference.
4. In the watch view, the variables *motorVars\_M1.flagRunIdentAndOnLine* can be set to "1" automatically. The *ISRCount* is increasing continuously.
5. Check calibration offsets of the motor, the offset value of the motor phase current sensing can be equal to approximately half of the scale current of ADC as shown in [Figure 3-25](#).
6. Probe the PWM output for motor drive control with an oscilloscope at J15 as shown in [Figure 3-23](#). All of the PWM duty are set to 50% in this build level, the PWM output waveforms are as shown in [Figure 3-26](#). The PWM switching frequency of *motor\_1* is 15 kHz.
7. The controller can now be halted, and the debug connection terminated. Fully halt the controller by first clicking the *Halt* button  on the toolbar or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on  or clicking *Run* → *Reset*.
8. Erase the code in the controller for the next build level by clicking *Tools* → *On-Chip Flash*, and click *Erase Flash* in the *On-Chip Flash* tab (make sure that all of the flash banks are checked) as shown in [Figure 3-27](#). This operation erases all of the program code stored in flash. (This step is optional, the user can ignore this step to load the new program code in next build level.)

### Note

Do not click *Cancel*, turn off the power of the board, or disconnect the emulator when erasing flash.

9. Close the CCS debug session by clicking the *Terminate Debug Session* button  or clicking *Run* → *Terminate*.



Expression	Type	Value	Address
motorVars_M1.ISRCount	unsigned long	280658	0x0000894C@Data
systemVars.mainLoopCnt	unsigned long	975542	0x00008A84@Data
motorVars_M1.speed_rpm	float	-85.6637268	0x0000895C@Data
motorVars_M1.speed_Hz	float	-5.71624184	0x000089AC@Data
motorVars_M1.speedRef_Hz	float	40.0	0x000089A4@Data
motorVars_M1.speedSet_Hz	float	40.0	0x000089A2@Data
motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\u001'	0x00008900@Data
motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\u001'	0x00008901@Data
motorVars_M1.accelerationMax_Hzps	float	10.0	0x00008998@Data
motorVars_M1.flagEnableForceAngle	unsigned char	1 '\u001'	0x00008912@Data
motorVars_M1.flagMotorIdentified	unsigned char	1 '\u001'	0x00008902@Data
motorVars_M1.flagEnableMotorIdentify	unsigned char	0 '\u000'	0x00008923@Data
motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	0x0000892B@Data
motorVars_M1.motorState	unknown	member 'motorState' not found at (motorVars_M1).motorState	
motorVars_M1.adcData.VdcBus_V	float	40.6497231	0x00008950@Data
motorVars_M1.adcData	struct _HAL_ADCData_t_	{VdcBus_V=40.7483864, I_A=[value=[0.218435556,0.175398439,-0.0...	0x00008950@Data
motorVars_M1.flagClearFaults	unsigned char	0 '\u000'	0x0000891D@Data
motorVars_M1.faultMtrUse.all	unsigned int	0	0x00008927@Data
motorVars_M1.faultMtrNow.bit	struct _FAULT_MTR_BITS_	{overVoltage=0, underVoltage=1, motorOverTemp=0, moduleOverT...	0x00008926@Data
motorVars_M1.speed_int_Hz	float	-362.948853	0x000089A6@Data
motorVars_M1.angleFOC_rad	float	2.62022805	0x000089AE@Data
motorVars_M1.angleEST_rad	float	-1.92112112	0x000089B0@Data
motorVars_M1.angleGen_rad	float	-2.91674447	0x000089B4@Data
motorCtrlVars_M1.flagEnableRunMotor	unsigned char	0 '\u000'	0x000086C4@Data
motorCtrlVars_M1.flagEnableSpeedCtrl	unsigned char	0 '\u000'	0x000086C6@Data
motorCtrlVars_M1.accelerationMax_Hzps	float	0.0	0x000086E0@Data
motorCtrlVars_M1.IqSet_A	float	0.0	0x000086D8@Data
motorVars_M1.speedEST_Hz	float	-4.97960234	0x000089AA@Data
motorVars_M1.speed_Hz	float	-5.71624184	0x000089AC@Data
motorVars_M1.angleFOC_rad	float	2.62022805	0x000089AE@Data
motorVars_M1.angleEST_rad	float	-1.92112112	0x000089B0@Data
motorVars_M1.anglePLL_rad	float	2.66169858	0x000089B2@Data
motorVars_M1.RsOnLine_Ohm	float	2.68207002	0x00008994@Data
motorSetVars_M1.Rs_Ohm	float	2.68207002	0x00008790@Data
motorSetVars_M1.Ls_d_H	float	0.00926135667	0x00008792@Data
motorSetVars_M1.Ls_q_H	float	0.00926135667	0x00008794@Data
motorSetVars_M1.flux_VpHz	float	0.0628318563	0x00008796@Data
motorVars_M1.Rs_Ohm	float	2.68207002	0x00008984@Data
motorVars_M1.Ls_d_H	float	0.00926135667	0x00008986@Data
motorVars_M1.Ls_q_H	float	0.00926135667	0x00008988@Data
motorVars_M1.flux_VpHz	float	0.0196469706	0x0000898A@Data

**Figure 3-25. Build Level 1: Expressions Window at Run Time**

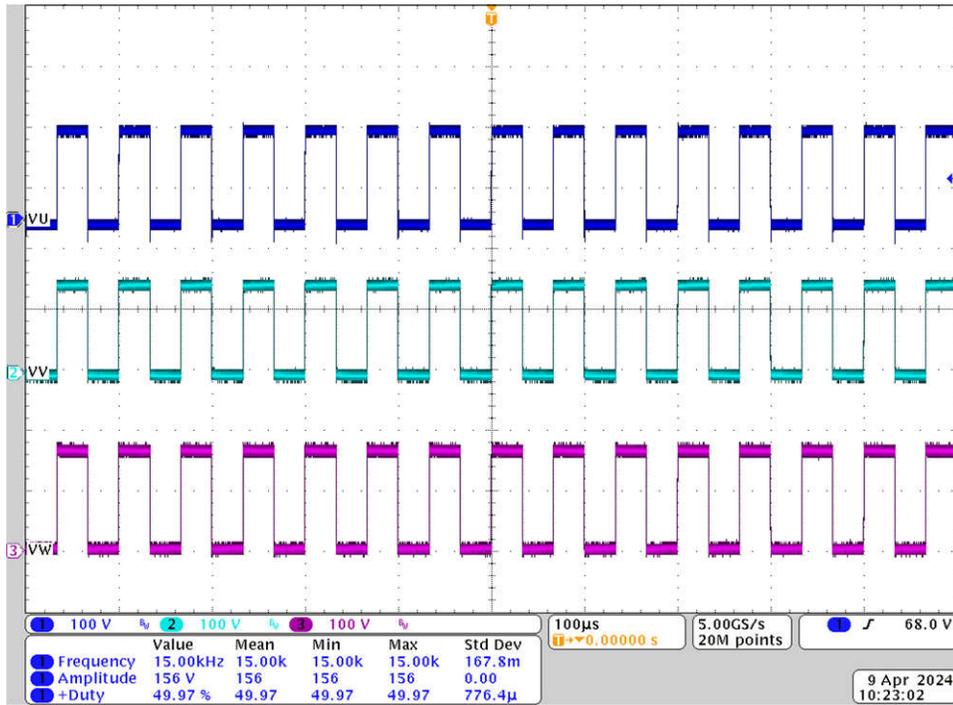


Figure 3-26. Build Level 1: MCU PWM Output and IPM Output

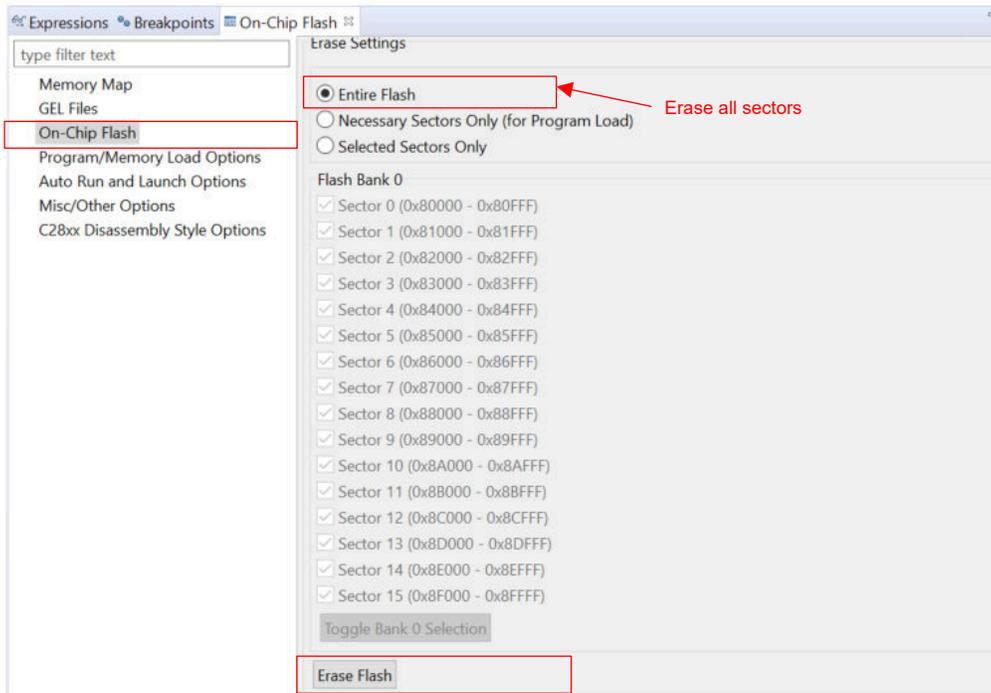


Figure 3-27. Build Level 1: Erase Program Code in Flash for Next Build Level

### 3.3.4.2 Build Level 2: Open-Loop Check With ADC Feedback

Objectives learned in this build level:

- Implements a simple scalar v/f control of motor to drive motor for validating current and voltage sensing circuit, and IPM circuit.
- Test InstaSPIN-FOC FAST or eSMO modules for motor control.

This system is running with open-loop control, the ADC measured values are only used for instrumentation purposes in this build level. The high-voltage DC power supply is implemented on the inverter, the bias power supply for the MCU controller and IPM is provided by the auxiliary power-supply module. Figure 3-28 shows the software flow for this build level.

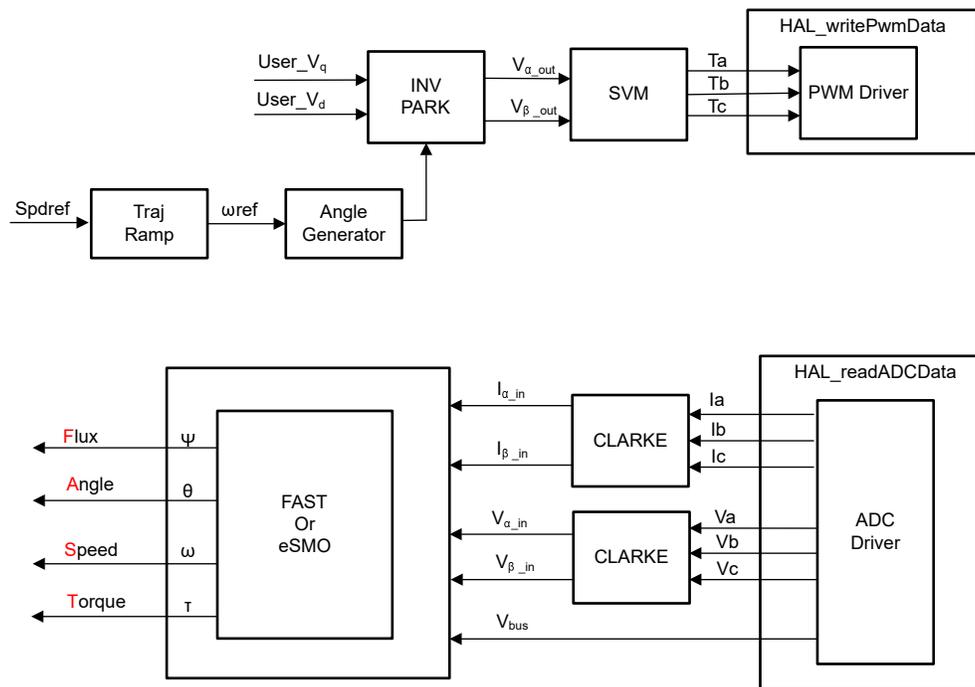
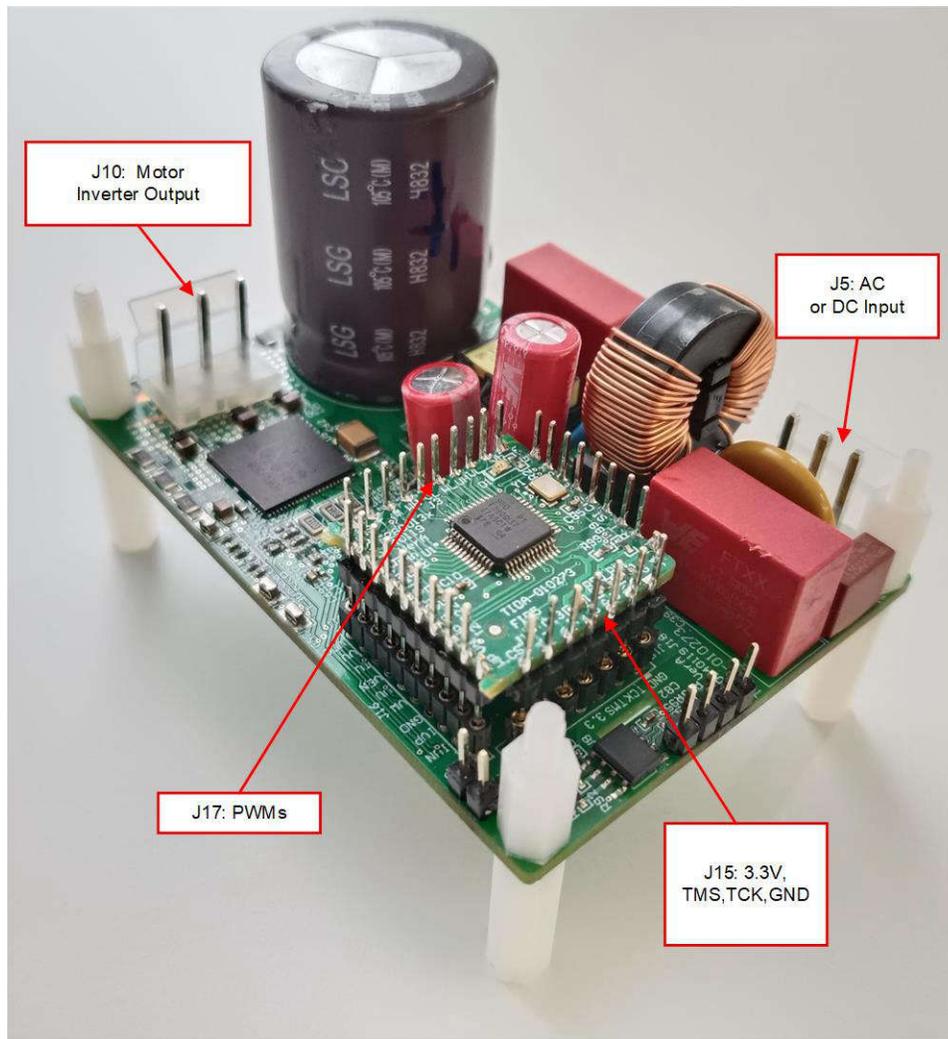


Figure 3-28. Control Software Block Diagram: Build Level 2 – Open-Loop Control

#### 3.3.4.2.1 Start CCS and Open Project

To start CCS and open the project, complete the following steps:

1. Connect an isolated AC power source capable of providing universal input up to 300 W to the input terminals (connector J5) of the reference board as shown in Figure 3-23. Set the power supply current limit to 2A. Do not turn the power supply on at this time.
2. Connect a motor to J10.
3. Follow Steps 2 and 3 of Section 3.3.4.1.1 to open the project.



**Figure 3-29. Connect the External AC or DC Power Supply to Verify the Hardware**

#### 3.3.4.2.2 Build and Load Project

To build and load the project, complete the following steps:

1. Set `DMC_BUILDLEVEL` to `DMC_LEVEL_2`.
2. Follow Steps 2 to 4 of [Section 3.3.4.1.2](#) to build the project and load code into controller.

#### 3.3.4.2.3 Setup Debug Environment Windows

Follow Steps 1 to 4 of [Section 3.3.4.1.3](#) to import the variables into the *Expressions* window by picking *BuildLevel2.txt*. The *Expressions* window appears as shown in [Figure 3-30](#).

#### 3.3.4.2.4 Run the Code

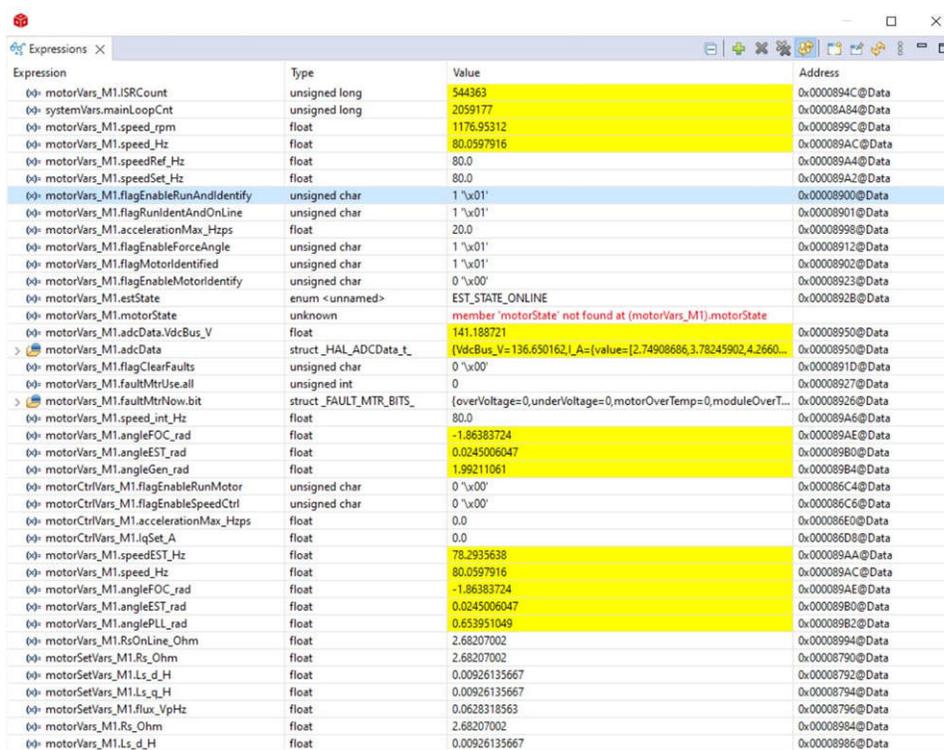
To run the code, complete the following steps:

1. Set the AC power source output to 0V, turn on the AC power source, slowly increase the output voltage from 0V to 220-VAC.
2. Run the project by clicking on the  button, or click *Run* → *Resume* in the *Debug* tab. The motor fault flags `motorVars_M1.faultMtrUse.all` need to be equal to "0", if not, the user must check the current and voltage sensing circuit as described in [Section 3.3.4.1](#).
3. To verify the current and voltage-sensing circuit of the inverter for the motor, set the variable `motorVars_M1.flagEnableRunAndIdentify` to "1" in the *Expressions* window as shown in [Figure 3-30](#). The

motor\_1 needs to run with v/f open loop, tune the v/f profile parameters in *user\_mtr1.h* as below according to the specification of the motor if the motor does not spin smoothly.

```
#define USER_MOTOR1_FREQ_LOW_HZ      (10.0f)      // Hz
#define USER_MOTOR1_FREQ_HIGH_HZ    (275.0f)     // Hz
#define USER_MOTOR1_VOLT_MIN_V      (10.0f)      // volt
#define USER_MOTOR1_VOLT_MAX_V     (200.0f)     // volt
```

4. The motor now spins with a setting speed in the variable *motorVars\_M1.speedRef\_Hz*, check the value of *motorVars\_M1.speed\_Hz* in the *Expressions* window. The value needs to be very close, as shown in [Figure 3-30](#).
5. Connect the oscilloscope voltage and current probes to watch the motor phase voltage and current as shown in [Figure 3-31](#).
6. Verify the overcurrent fault protection by decreasing the value of the variable *motorVars\_M1.overCurrent\_A*, the overcurrent protection is implemented by the CMPSS modules. The overcurrent fault is triggered if the *motorVars\_M1.overCurrent\_A* is set to a value less than the actual current, the PWM output is disabled, the *motorVars\_M1.flagEnableRunAndIdentify* is cleared to "0", and the *motorVars\_M1.faultMtrUse.all* is set to "0x10".
7. The controller can now be halted, and the debug connection terminated. Fully halting the controller by first clicking the *Halt* button on the toolbar  or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on  or clicking *Run* → *Reset*.
8. Close the CCS debug session by clicking on *Terminate Debug Session*  or clicking *Run* → *Terminate*.



Expression	Type	Value	Address
motorVars_M1.ISRCnt	unsigned long	544363	0x0000894C@Data
systemVars.mainLoopCnt	unsigned long	2059177	0x00008A84@Data
motorVars_M1.speed_rpm	float	1176.95312	0x0000899C@Data
motorVars_M1.speed_Hz	float	80.0597916	0x000089AC@Data
motorVars_M1.speedRef_Hz	float	80.0	0x000089A4@Data
motorVars_M1.speedSet_Hz	float	80.0	0x000089A2@Data
motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\x01'	0x00008900@Data
motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\x01'	0x00008901@Data
motorVars_M1.accelerationMax_Hzps	float	20.0	0x00008998@Data
motorVars_M1.flagEnableForceAngle	unsigned char	1 '\x01'	0x00008912@Data
motorVars_M1.flagMotorIdentified	unsigned char	1 '\x01'	0x00008902@Data
motorVars_M1.flagEnableMotorIdentify	unsigned char	0 '\x00'	0x00008923@Data
motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	0x0000892B@Data
motorVars_M1.motorState	unknown	member 'motorState' not found at (motorVars_M1).motorState	
motorVars_M1.adcData.VdcBus_V	float	141.188721	0x00008950@Data
motorVars_M1.adcData	struct _HAL_ADCData_t	{VdcBus_V=136.650162, I_A={values:[2.74908686,3.78245902,4.2660...}	0x00008950@Data
motorVars_M1.flagClearFaults	unsigned char	0 '\x00'	0x0000891D@Data
motorVars_M1.faultMtrUse.all	unsigned int	0	0x00008927@Data
motorVars_M1.faultMtrNow.bit	struct _FAULT_MTR_BITS_	{overVoltage=0,underVoltage=0,motorOverTemp=0,moduleOverT...	0x00008926@Data
motorVars_M1.speed_int_Hz	float	80.0	0x000089A6@Data
motorVars_M1.angleFOC_rad	float	-1.86383724	0x000089AE@Data
motorVars_M1.angleEST_rad	float	0.0245006047	0x000089B0@Data
motorVars_M1.angleGen_rad	float	1.99211061	0x000089B4@Data
motorCtrlVars_M1.flagEnableRunMotor	unsigned char	0 '\x00'	0x000086C4@Data
motorCtrlVars_M1.flagEnableSpeedCtrl	unsigned char	0 '\x00'	0x000086C6@Data
motorCtrlVars_M1.accelerationMax_Hzps	float	0.0	0x000086E0@Data
motorCtrlVars_M1.lqSet_A	float	0.0	0x000086D8@Data
motorVars_M1.speedEST_Hz	float	78.2935638	0x000089AA@Data
motorVars_M1.speed_Hz	float	80.0597916	0x000089AC@Data
motorVars_M1.angleFOC_rad	float	-1.86383724	0x000089AE@Data
motorVars_M1.angleEST_rad	float	0.0245006047	0x000089B0@Data
motorVars_M1.anglePLL_rad	float	0.653951049	0x000089B2@Data
motorVars_M1.RsOnLine_Ohm	float	2.68207002	0x00008994@Data
motorSetVars_M1.Rs_Ohm	float	2.68207002	0x00008790@Data
motorSetVars_M1.Ls_d_H	float	0.00926135667	0x00008792@Data
motorSetVars_M1.Ls_q_H	float	0.00926135667	0x00008794@Data
motorSetVars_M1.flux_VpHz	float	0.0628318563	0x00008796@Data
motorVars_M1.Rs_Ohm	float	2.68207002	0x00008984@Data
motorVars_M1.Ls_d_H	float	0.00926135667	0x00008986@Data

**Figure 3-30. Build Level 2: Expressions Window at Run Time**

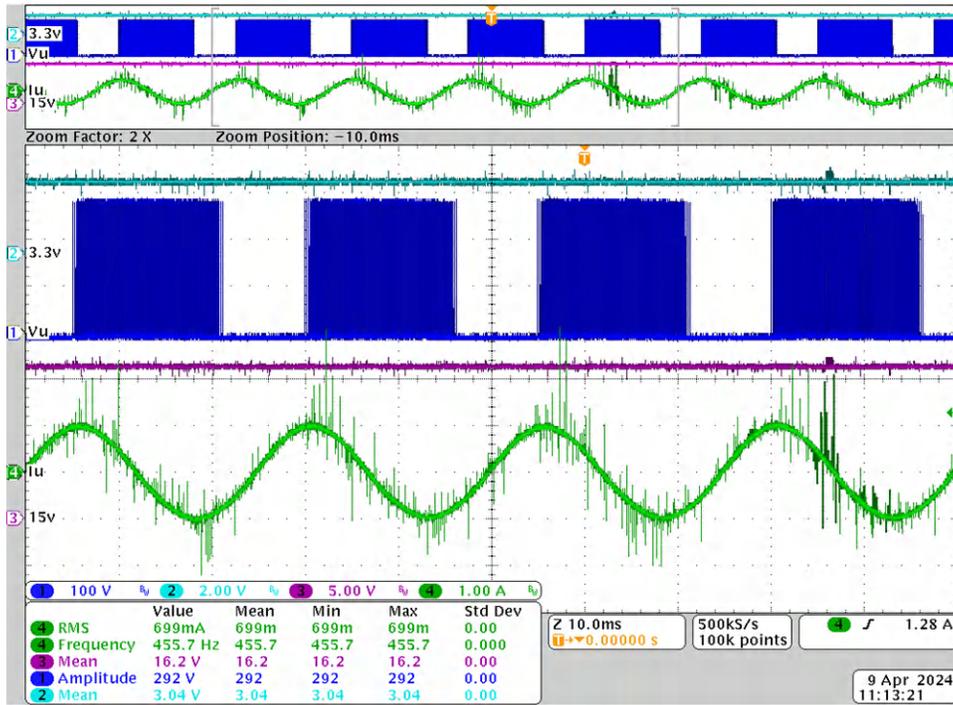


Figure 3-31. Build Level 2: Motor Phase Voltage and Current

### 3.3.4.3 Build Level 3: Closed Current Loop Check

Objectives learned in this build level:

- Evaluate the closed current loop of motor operation.

In this build level, the motor is controlled using i/f control that the rotor angle is generated from ramp generator module. Figure 3-32 shows the software flow for this build level.

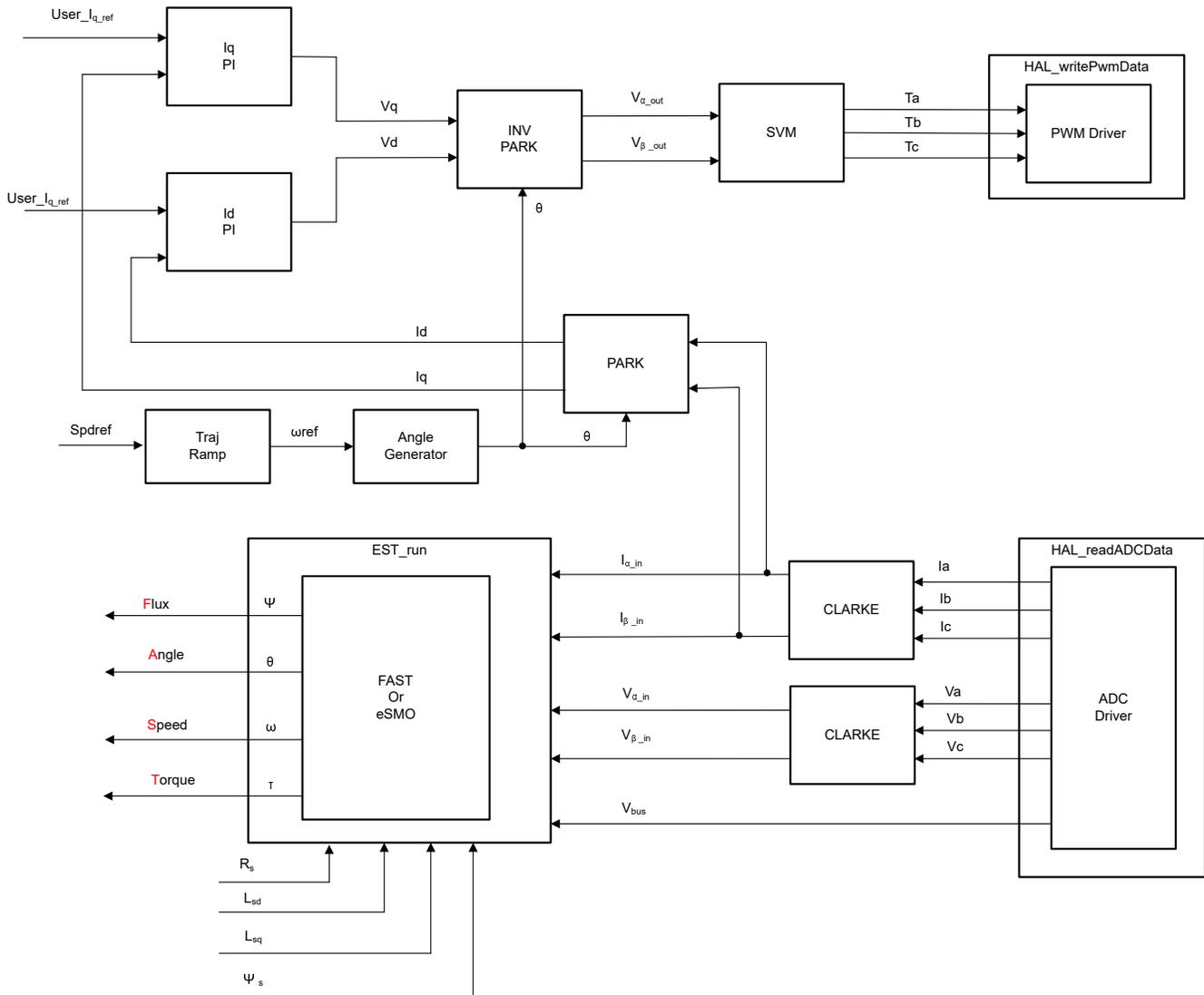


Figure 3-32. Control Software Block Diagram: Build Level 3 – Current Close-Loop Control

#### 3.3.4.3.1 Start CCS and Open Project

To start CCS and open the project, complete the following steps:

1. Connect a programmable, isolated AC power supply capable of providing universal AC input up to 300W to the input terminals (connector J5) of the reference board as shown in Figure 3-29. Set the power supply current limit to 3A and the output frequency to 50/60Hz. Do not turn the power supply on at this time.
2. Connect a motor (compressor) to J10.
3. Follow Steps 2 and 3 of Section 3.3.4.1.1 to open the project.

#### 3.3.4.3.2 Build and Load Project

To build and load the project, complete the following steps:

1. Open the sys\_settings.h file, set DMC\_BUILDLEVEL to DMC\_LEVEL\_3.

- Follow Steps 2 to 4 of [Section 3.3.4.1.2](#) to build the project and load code into controller.

### 3.3.4.3.3 Setup Debug Environment Windows

Follow Steps 1 to 4 of [Section 3.3.4.1.3](#) to import the variables into the *Expressions* window by picking *BuildLevel3.txt*. The *Expressions* window appears as shown in [Figure 3-30](#).

### 3.3.4.3.4 Run the Code

To run the code, complete the following steps:

- Set the AC source output to 0V at 50/60 Hz, turn on the AC power supply, slowly increase the input voltage from 0V to 220V AC.
- Run the project by clicking the  button, or click *Run* → *Resume* in the *Debug* tab. Set *systemVars.flagEnableSystem* to "1" after a fixed time, that means the offsets calibration has been done and the power relay for inrush is turned on. The motor fault flags for *motorVars\_M1.faultMtrUse.all* need to equal to "0", if the values do not, check the current and voltage sensing circuit as described in [Section 3.3.4.1](#).
- To verify current closed-loop control for motor, set the variable *motorVars\_M1.flagEnableRunAndIdentify* to "1" in the *Expressions* window as shown in [Figure 3-33](#). The motor needs to run with a closed-loop control using the angle from the angle generator at a setting speed in the variable *motorVars\_M1.speedRef\_Hz*, check the value of *motorVarsM1.speed\_Hz* in *Expressions* window, both variables value need to be very close.
- The motor current *I<sub>q</sub>* can be set and changed with *motorVars\_M1.Idq\_Set\_A.value[1]*
- Connect oscilloscope probes to IPM output to watch the motor phase voltage and current as shown in [Figure 3-34](#). Change the *Idq\_set\_A[0].value[1]* in the *Expressions* window, the motor phase current needs to be increasing accordingly.
- If the motor cannot run with current-closed loop and appears to experience an overcurrent fault, check if the sign of *adcData[0].current\_sf* and the value of *userParams[0].current\_sf* are set correctly according to the hardware board.
- The controller can now be halted before setting the *motorVars\_M1.flagEnableRunAndIdentify* to "0", and the debug connection terminated. Fully halting the controller by first clicking the *Halt* button  on the toolbar or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on  or clicking *Run* → *Reset*.
- Close CCS debug session by clicking on *Terminate Debug Session*  or clicking *Run* → *Terminate*.

Expression	Type	Value	Address
motorVars_M1.ISRCCount	unsigned long	1284134	0x0000894C@Data
systemVars.mainLoopCnt	unsigned long	5307166	0x00008A84@Data
motorVars_M1.speed_rpm	float	592.226624	0x0000899C@Data
motorVars_M1.speed_Hz	float	40.6853447	0x000089AC@Data
motorVars_M1.speedRef_Hz	float	40.0	0x000089A4@Data
motorVars_M1.speedSet_Hz	float	40.0	0x000089A2@Data
motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\u0001'	0x00008900@Data
motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\u0001'	0x00008901@Data
motorVars_M1.accelerationMax_Hzps	float	20.0	0x00008998@Data
motorVars_M1.flagMotorIdentified	unsigned char	1 '\u0001'	0x00008912@Data
motorVars_M1.flagEnableMotorIdentify	unsigned char	0 '\u0000'	0x00008902@Data
motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	0x0000892B@Data
motorVars_M1.controlStatus	enum <unnamed>	MOTOR_CTRL_RUN	0x0000892A@Data
motorVars_M1.adcData.VdcBus_V	float	308.918152	0x00008950@Data
motorVars_M1.adcData	struct _HAL_ADCData_t	{VdcBus_V=308.7208251_A=(value=[-1.88685691,-0.674133778,0.91...	0x00008950@Data
motorVars_M1.flagClearFaults	unsigned char	0 '\u0000'	0x0000891D@Data
motorVars_M1.faultMtrUseAll	unsigned int	0	0x00008927@Data
motorVars_M1.faultMtrNow.bit	struct _FAULT_MTR_BITS_	{overVoltage=0,underVoltage=0,motorOverTemp=0,moduleOverT...	0x00008926@Data
motorVars_M1.speed_int_Hz	float	40.0	0x000089A6@Data
motorVars_M1.angleFOC_rad	float	2.30622911	0x000089AE@Data
motorVars_M1.angleEST_rad	float	-1.95582139	0x000089B0@Data
motorVars_M1.angleGen_rad	float	-2.92251492	0x000089B4@Data
motorCtrlVars_M1.flagEnableRunMotor	unsigned char	0 '\u0000'	0x000086C4@Data
motorCtrlVars_M1.flagEnableSpeedCtrl	unsigned char	0 '\u0000'	0x000086C6@Data
motorCtrlVars_M1.accelerationMax_Hzps	float	0.0	0x000086E0@Data
motorCtrlVars_M1.lqSet_A	float	0.0	0x000086D8@Data
motorVars_M1.speedEST_Hz	float	40.7923813	0x000089AA@Data
motorVars_M1.speed_Hz	float	40.6853447	0x000089AC@Data
motorVars_M1.angleFOC_rad	float	2.30622911	0x000089AE@Data
motorVars_M1.angleEST_rad	float	-1.95582139	0x000089B0@Data
motorVars_M1.anglePLL_rad	float	1.01841605	0x000089B2@Data
motorVars_M1.RsOnLine_Ohm	float	2.68207002	0x00008994@Data
motorSetVars_M1.Rs_Ohm	float	2.68207002	0x00008790@Data
motorSetVars_M1.Ls_d_H	float	0.00926135667	0x00008792@Data
motorSetVars_M1.Ls_q_H	float	0.00926135667	0x00008794@Data
motorSetVars_M1.flux_VpHz	float	0.0628318563	0x00008796@Data
motorVars_M1.Rs_Ohm	float	2.68207002	0x00008984@Data
motorVars_M1.Ls_d_H	float	0.00926135667	0x00008986@Data
motorVars_M1.Ls_q_H	float	0.00926135667	0x00008988@Data
motorVars_M1.flux_VpHz	float	0.378431171	0x0000898A@Data
motorVars_M1.flux_Wb	float	0.0603588633	0x0000898C@Data
angleGen_M1	struct _ANGLE_GEN_Obj_	{freq_Hz=40.0,angleDeltaFactor=0.000418879034,angleDelta_rad=...	0x0000861C@Data
motorVars_M1.Idq_set_A.value[0]	float	0.0	0x00008A2A@Data
motorVars_M1.Idq_set_A.value[1]	float	2.0	0x00008A2C@Data

Figure 3-33. Build Level 3: Expressions Window at Run Time

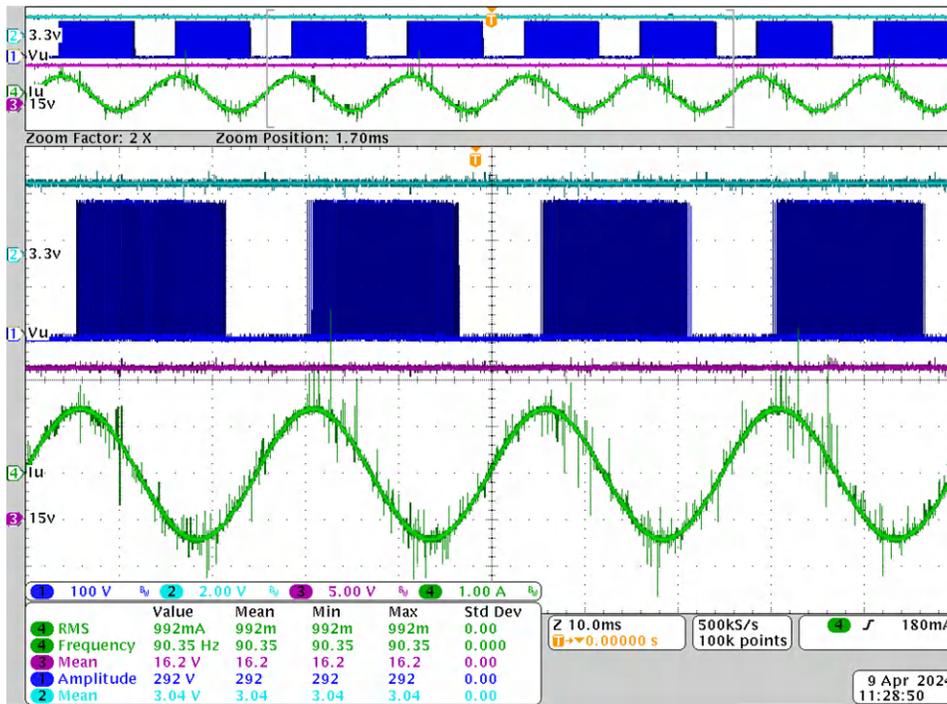


Figure 3-34. Build Level 3: Motor Current Under 1-Arms I<sub>Q</sub> Setting



- The required motor parameters must be recorded in the header files (*user\_mtr1.h*) as shown in the following example codes. If the motor parameters are not well known, the motor identification can be used to achieve the motor parameters if the FAST estimator is implemented in the reference design.

```
#define USER_MOTOR1_Rs_Ohm          (4.5f)
#define USER_MOTOR1_Ls_d_H          (0.0196f)
#define USER_MOTOR1_Ls_q_H          (0.0196)
#define USER_MOTOR1_RATED_FLUX_VpHz (0.441f)
```

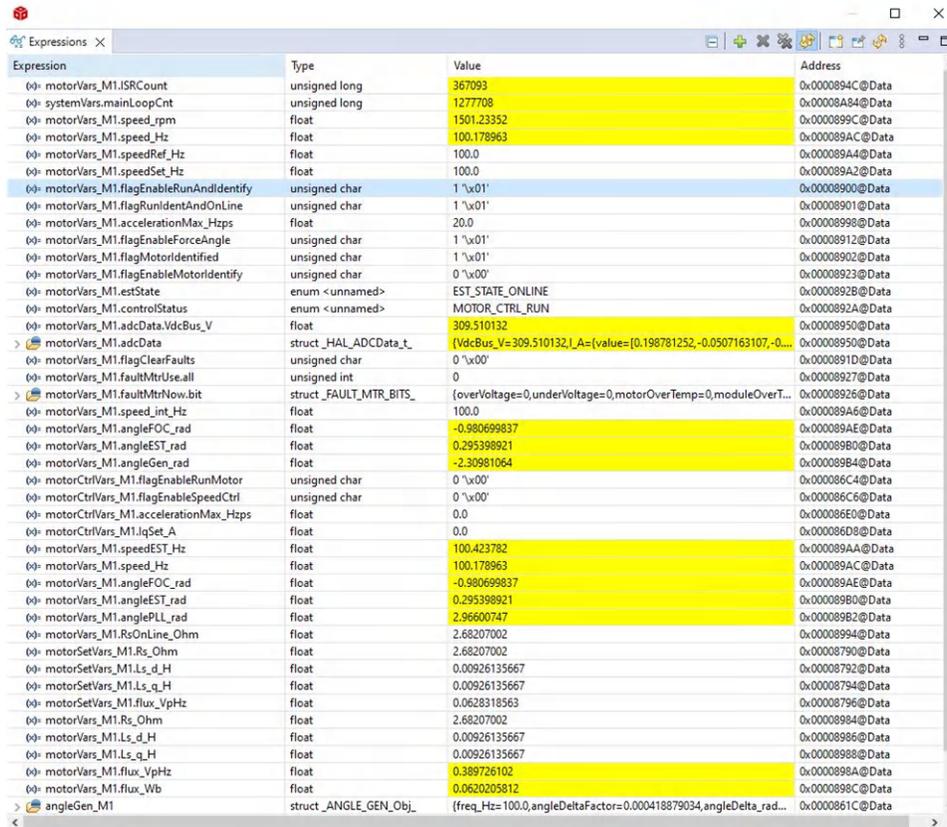
- Change the *userParams\_M1.flag\_bypassMotorId* value to "false" to enable the motor identification as the following example code for motor.

```
// true->enable identification, false->disable identification
userParams[MTR_1].flag_bypassMotorId = false;
```

- Set the right identification variables value in the *user\_mtr1.h* according to the specification of the motor.

```
#define USER_MOTOR1_RES_EST_CURRENT_A    (1.0f)    // A - 10~30% of rated current of the
motor
#define USER_MOTOR1_IND_EST_CURRENT_A    (-1.0f)   // A - 10~30% of rated current of the
motor, just enough to enable rotation
#define USER_MOTOR1_MAX_CURRENT_A       (2.0f)    // A - 30~150% of rated current of the
motor
#define USER_MOTOR1_FLUX_EXC_FREQ_HZ     (40.0f)   // Hz - 10~30% of rated frequency of
the motor
```

- Rebuild the project and load the code into the controller, run the project by clicking on the  button, or click *Run* → *Resume* in the *Debug* tab. The *systemVars.flagEnableSystem* needs to be set to "1" after a fixed time, that means the offsets calibration have been done and the power relay for inrush is turned on. The motor fault flags *motorVars\_M1.faultMtrUse.all* need to be equal to "0", if not, check the current and voltage sensing circuit as described in [Section 3.3.4.1](#).
- Set the variable *motorVars\_M1.flagEnableRunAndIdentify* to "1" in the *Expressions* window as shown in [Figure 3-36](#), the motor identification can be executed, the whole process takes about 150 s. Once *motorVars\_M1.flagEnableRunAndIdentify* is equal to "0", the motor parameters have been identified. Record the watch window values with the newly-defined motor parameters in *user\_mtr1.h* as follows:
  - USER\_MOTOR1\_Rs* = *motorVars\_M1.Rs\_Ohm*'s value
  - USER\_MOTOR1\_Ls\_d* = *motorVars\_M1.Ls\_d\_H*'s value
  - USER\_MOTOR1\_Ls\_q* = *motorVars\_M1.Ls\_q\_H*'s value
  - USER\_MOTOR\_RATED\_FLUX* = *motorVars\_M1.flux\_VpHz*'s value
- Set both *userParams\_M1.flag\_bypassMotorId* to "true" after successfully identify the motors parameters, rebuild the project and load the code into the controller.
  - Set the variables *motorVars\_M1.flagEnableRunAndIdentify* equal to "1" again for starting to run the motor.
  - Set the variables *motorVars\_M1.speedRef\_Hz* to a different value and watch how the motor shaft speed follows.
  - To change the acceleration, enter a different acceleration value for the variables *motorVars\_M1.accelerationMax\_Hzps* and *motorVars\_M1.accelerationMax\_Hzps*.
- The controller can now be halted before setting the *motorVars\_M1.flagEnableRunAndIdentify* to "0", and the debug connection terminated. Fully halting the controller by first clicking the *Halt* button  on the toolbar or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on  or clicking *Run* → *Reset*.
- Close the CCS debug session by clicking on *Terminate Debug Session*  or clicking *Run* → *Terminate*.



Expression	Type	Value	Address
motorVars_M1.ISRCCount	unsigned long	367093	0x0000894C@Data
systemVars.mainLoopCnt	unsigned long	1277708	0x00008A84@Data
motorVars_M1.speed_rpm	float	1501.23352	0x0000899C@Data
motorVars_M1.speed_Hz	float	100.178963	0x000089AC@Data
motorVars_M1.speedRef_Hz	float	100.0	0x000089A4@Data
motorVars_M1.speedSet_Hz	float	100.0	0x000089A2@Data
motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\x01'	0x00008900@Data
motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\x01'	0x00008901@Data
motorVars_M1.accelerationMax_Hzps	float	20.0	0x00008998@Data
motorVars_M1.flagEnableForceAngle	unsigned char	1 '\x01'	0x00008912@Data
motorVars_M1.flagMotorIdentified	unsigned char	1 '\x01'	0x00008902@Data
motorVars_M1.flagEnableMotorIdentify	unsigned char	0 '\x00'	0x00008923@Data
motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	0x00008928@Data
motorVars_M1.controlStatus	enum <unnamed>	MOTOR_CTRL_RUN	0x0000892A@Data
motorVars_M1.adcData.VdcBus_V	float	309.510132	0x00008950@Data
motorVars_M1.adcData	struct_HAL_ADCData_t	{VdcBus_V=309.510132,I_A=(value=[0.198781252,-0.0507163107,-0...	0x00008950@Data
motorVars_M1.flagClearFaults	unsigned char	0 '\x00'	0x0000891D@Data
motorVars_M1.faultMtrUseAll	unsigned int	0	0x00008927@Data
motorVars_M1.faultMtrNow.bit	struct_FAULT_MTR_BITS_	{overVoltage=0,underVoltage=0,motorOverTemp=0,moduleOverT...	0x00008926@Data
motorVars_M1.speed_int_Hz	float	100.0	0x000089A6@Data
motorVars_M1.angleFOC_rad	float	-0.980699837	0x000089AE@Data
motorVars_M1.angleEST_rad	float	0.295398921	0x000089B0@Data
motorVars_M1.angleGen_rad	float	-2.30981064	0x000089B4@Data
motorCtrlVars_M1.flagEnableRunMotor	unsigned char	0 '\x00'	0x000086C4@Data
motorCtrlVars_M1.flagEnableSpeedCtrl	unsigned char	0 '\x00'	0x000086C6@Data
motorCtrlVars_M1.accelerationMax_Hzps	float	0.0	0x000086E0@Data
motorCtrlVars_M1.lqSet_A	float	0.0	0x000086D8@Data
motorVars_M1.speedEST_Hz	float	100.423782	0x000089AA@Data
motorVars_M1.speed_Hz	float	100.178963	0x000089AC@Data
motorVars_M1.angleFOC_rad	float	-0.980699837	0x000089AE@Data
motorVars_M1.angleEST_rad	float	0.295398921	0x000089B0@Data
motorVars_M1.anglePLL_rad	float	2.96600747	0x000089B2@Data
motorVars_M1.RsOnLine_Ohm	float	2.68207002	0x00008994@Data
motorSetVars_M1.Rs_Ohm	float	2.68207002	0x00008790@Data
motorSetVars_M1.Ls_d_H	float	0.00926135667	0x00008792@Data
motorSetVars_M1.Ls_q_H	float	0.00926135667	0x00008794@Data
motorSetVars_M1.flux_VpHz	float	0.0628318563	0x00008796@Data
motorVars_M1.Rs_Ohm	float	2.68207002	0x00008994@Data
motorVars_M1.Ls_d_H	float	0.00926135667	0x00008996@Data
motorVars_M1.Ls_q_H	float	0.00926135667	0x00008998@Data
motorVars_M1.flux_VpHz	float	0.389726102	0x0000899A@Data
motorVars_M1.flux_Wb	float	0.0620205812	0x0000899C@Data
angleGen_M1	struct_ANGLE_GEN_Obj_	{freq_Hz=100.0,angleDeltaFactor=0.000418879034,angleDelta_rad...	0x0000861C@Data

Figure 3-36. Build Level 4: Expressions Window at Run Time

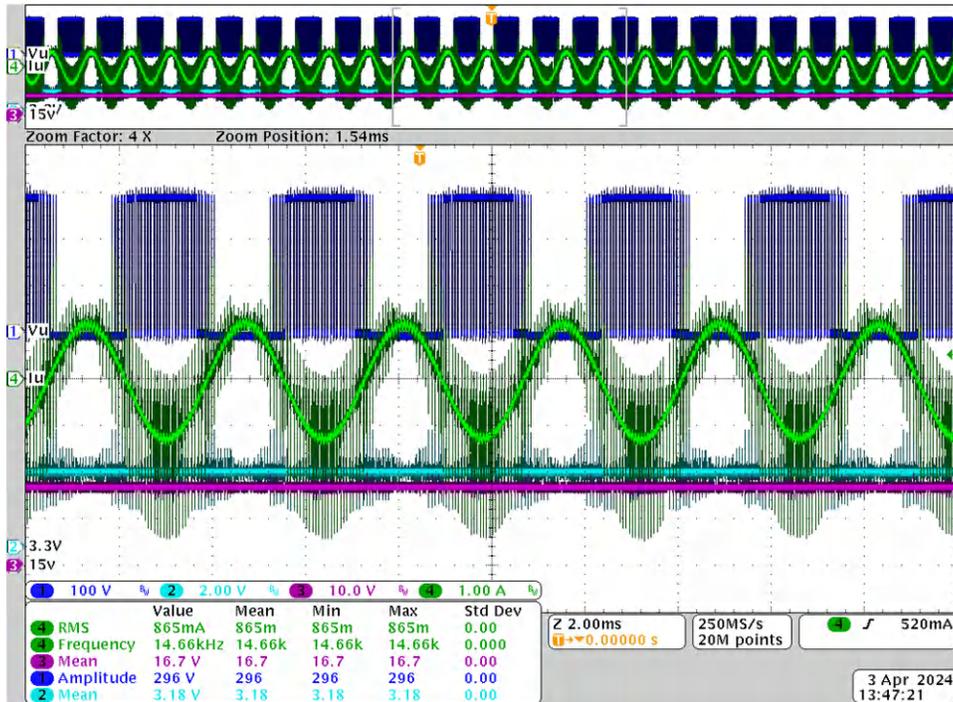


Figure 3-37. Build Level 4: Rotor Angle, Phase Current of Motor

### 3.3.4.4.5 Tuning Motor Drive FOC Parameters

The sliding mode current observer consists of a model-based current observer and a bang-bang control generator driven by error between estimated motor currents and actual motor currents. The F and G parameters are calculated based on the motor parameters  $R_s$ , and  $L_s$  as described in [Section 2.4.2.2.1.2](#). The observer gain  $k$  for bang-bang control, the cutoff frequency for LPF, and the  $K_p$  and  $K_i$  for PLL angle tracker must be tuned according to the testing state, and try to get the best parameters. The user can run the FAST estimator and eSMO in parallel to validate the angle from the eSMO for tuning the parameters. The initial parameters are defined in the user-mtr1.h files.

```
// Only for eSMO
#define USER_MOTOR1_KSLIDE_MAX      (1.50f)
#define USER_MOTOR1_KSLIDE_MIN      (0.75f)

#define USER_MOTOR1_PLL_KP_MAX      (10.0f)
#define USER_MOTOR1_PLL_KP_MIN      (2.0f)
#define USER_MOTOR1_PLL_KP_SF      (5.0f)

#define USER_MOTOR1_BEMF_THRESHOLD  (0.5f)
#define USER_MOTOR1_BEMF_KSLF_FC_HZ (2.0f)
#define USER_MOTOR1_THETA_OFFSET_SF (1.0f)
#define USER_MOTOR1_SPEED_LPF_FC_HZ (200.0f)
```

The speed and current PI regulator gains are calculated according to the motor parameters, the user can tune these gains online to optimize the control performance of the system.

- Adding the motorVars[0].Kp\_spd, motorVars[0].Ki\_spd, motorVars[0].Kp\_lq, motorVars[0].Ki\_lq, motorVars[0].Kp\_Id, and motorVars[0].Ki\_Id to the *Expressions* window in CCS Debug Perspective. Change the PI gains for the compressor motor drive and record the values.

### 3.3.4.4.6 Tuning Field Weakening and MTPA Control Parameters

The FWC and MTPA functions are added and called in the motor drive ISR to calculate current angle, and then compute the reference currents of the d-axis and q-axis.

1. Adding the pre-define symbols *MOTOR1\_FWC* and *MOTOR1\_MTPA* in the build configuration of the project as described in [Section 3.3.2](#) for enabling the FWC and MTPA, respectively.
2. In the *user\_mtr1.h* file, make sure the motor parameters are known and correctly set. In *mtpa.h*, make sure the tables are set properly for and calculations are set according to the specification of the motor.
3. Add the variables VsRef\_pu, Kp\_fwc, and Ki\_fwc to the *Expressions* window in CCS Debug Perspective, and tune these parameters to achieve the expected performance for the field weakening control according to the motor and the system.
4. After tuning and fixing these variables, record the watch window values with the newly-defined parameters in *user\_mtr1.h* file.

USER\_M1\_FWC\_VREF = VsRef\_pu's value. The factor of the reference voltage for Field Weakening Control.

USER\_M1\_FWC\_KP = Kp\_fwc's value. The Kp gain of PI regulator for Field Weakening Control

USER\_M1\_FWC\_KI = Ki\_fwc's value. The Ki gain of PI regulator for Field Weakening Control

5. MTPA control parameters are calculated according to the motor parameters,  $L_d$ ,  $L_q$ , and  $\psi_m$ , so there are not any additional parameters to be tuned online.

### 3.3.4.4.7 Tuning Current Sensing Parameters

Accurate current sensing is important to estimate the rotor angle and speed, and also have the best dynamic motor control. The current sensing parameters must match the hardware by setting the following related parameters:

- Dead-band time, the rising edge delay time must be greater than (high-side turn on time) + (low side turn-off time) of the power module, and the falling edge delay time must be greater than (high-side turn-off time) +

(low-side turn-on time) of the power module as shown in the following setting for a power module used in the reference design.

```

//! \brief Defines the PWM deadband falling edge delay count (system clocks)
#define MTR1_PWM_DBFED_CNT (uint16_t)(2.5f * 120.0f) // 2.5us, (>2.0us)

//! \brief Defines the PWM deadband rising edge delay count (system clocks)
#define MTR1_PWM_DBRED_CNT (uint16_t)(2.5f * 120.0f) // 2.50us, (>2.0us)
  
```

- Minimum duration of pulse width PWM, specifies to be greater than (Hardware delay time + Dead band time + Ringing duration + ADC sampling time).

```

//! \brief Defines the minimum duration, Clock cycle
#define USER_M1_DCLINKSS_MIN_DURATION (450U)
  
```

- Sample and hold delay time, specifies the time delay from PWM output to ADC sample time for current sensing. The delay time is dependent on the hardware and includes the propagation delay of the gate driver circuit and turn on and turn off delay of the power FET, and is less than or equal to (Minimum duration – ADC sampling time).

```

//! \brief Defines the sample delay, Clock cycle
#define USER_M1_DCLINKSS_SAMPLE_DELAY (430U)
  
```

### 3.4 Test Results

The following sections show the test data from characterizing the design. The test results are divided in multiple sections that cover the steady-state performance and data, functional performance waveforms, and transient performance waveforms of the fan and compressor motor.

#### 3.4.1 Fast and clean Rising/Falling Edge

As a IPM with GaN, DRV7308 has very high switching speed, fast switching speed will reduce power loss of IPM, clean switching edge can improve EMI performance. Figure 3-38 shows a 40nS rising edge from 0 to 310VDC.

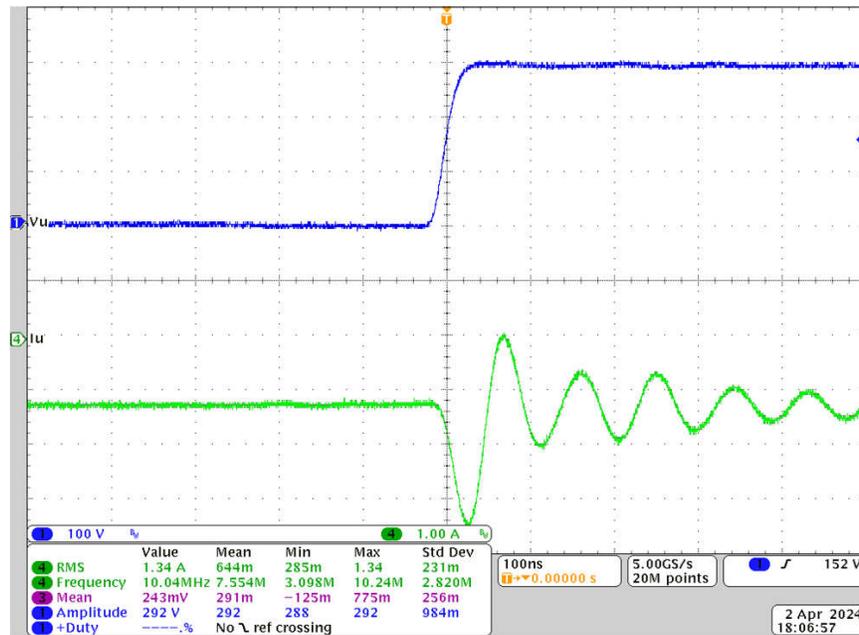


Figure 3-38. Fast and Clean Rising Edge

Figure 3-39 shows a 40nS falling edge from 310V to 0VDC.

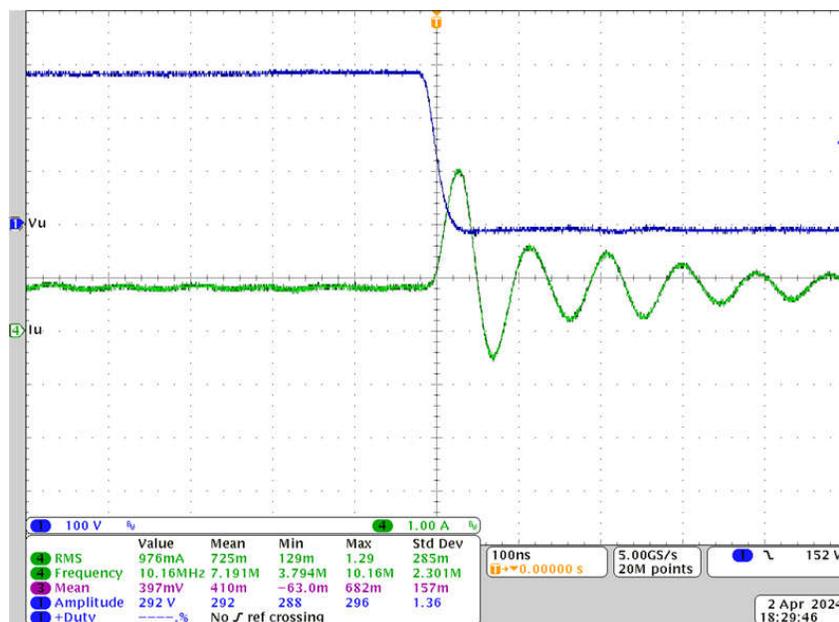


Figure 3-39. Fast and Clean Falling Edge

### 3.4.2 Inrush Current Protection

Figure 3-40 shows waveform of inrush protection circuit when power on under 275VAC input voltage. Charging current of electrolytic capacitor C67 flow through PTC RT1, and produce a high voltage drop on both RT1 and Q1(these are in parallel), test shows that there is no current flow thought Q1 during powering on. Software can turn on Q1 to short RT1 to provide a low impedance current path after 2-3 seconds delay.

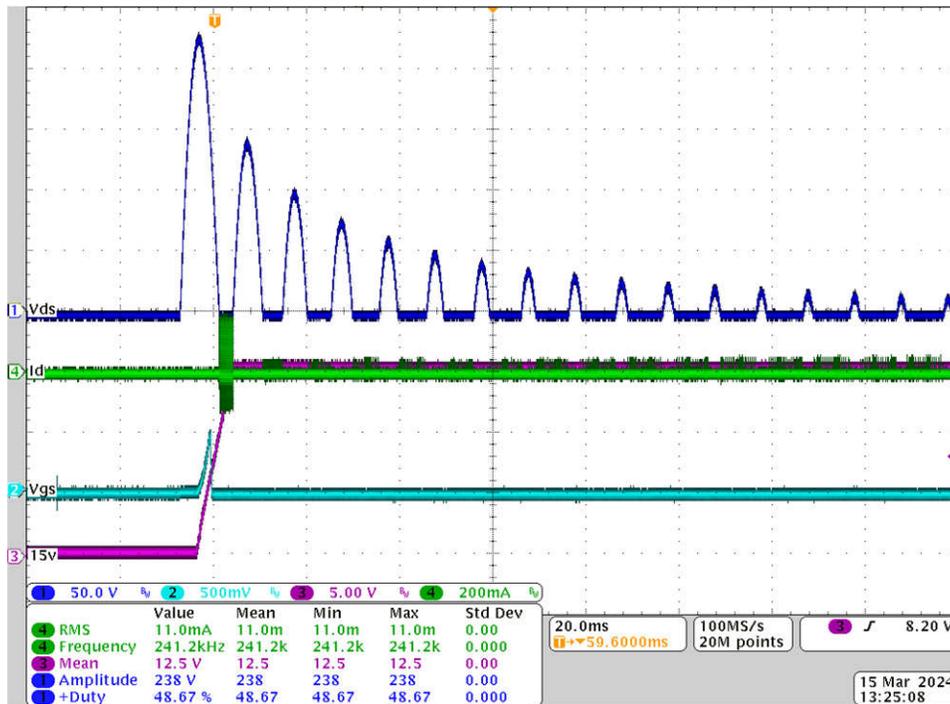


Figure 3-40. Inrush Current Protection Under 275VAC

### 3.4.3 Thermal performance under 300VDC

Figure 3-41 shows the waveform at 3480 RPM (290Hz) with 250W SC225 fan motor load. The waveform includes the following display:

- CH1 (Blue): phase U voltage
- CH2 (Light Blue): 3.3V voltage
- CH3 (Purple): 15V voltage
- CH4 (Green): Phase U current

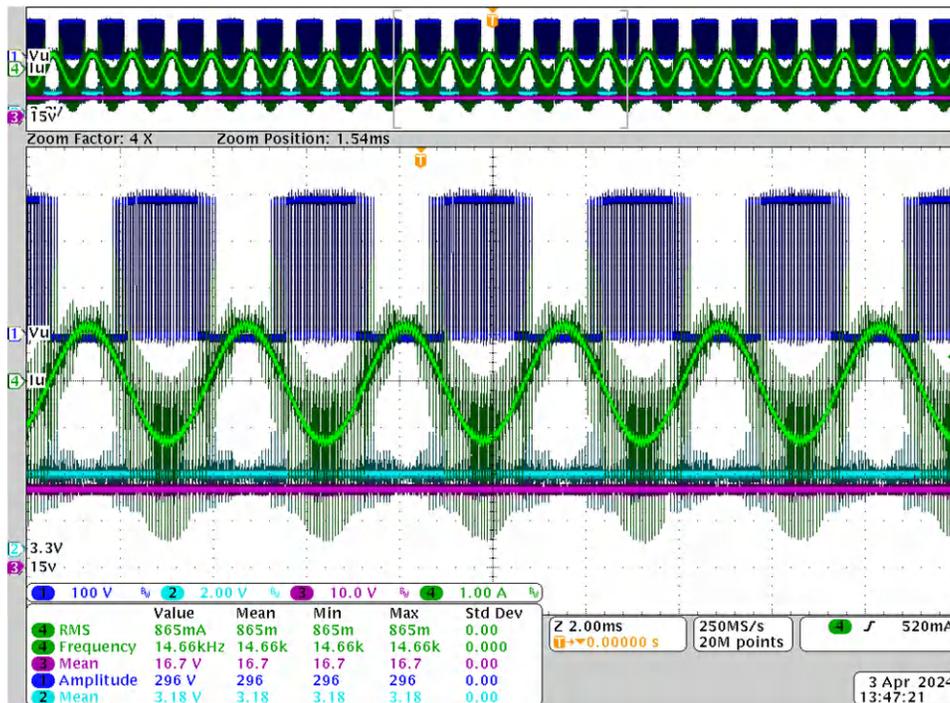


Figure 3-41. Phase Current and Voltage Waveforms of Motor at 250W, 290Hz

Figure 3-42 shows the 41C temperature rising for DRV7308 under 250 W, 3480 RPM (290 Hz) and 25C ambient temperature. Although GaN offers high efficiency and low power loss, but there is still about 2.5W power loss on GaN at 250W power level, and since there is no heatsink, DRV7308 can only be cooled down by copper pours, so temperature rising is related to the size of copper pure. Make sure provide enough copper pours for DRV7308.

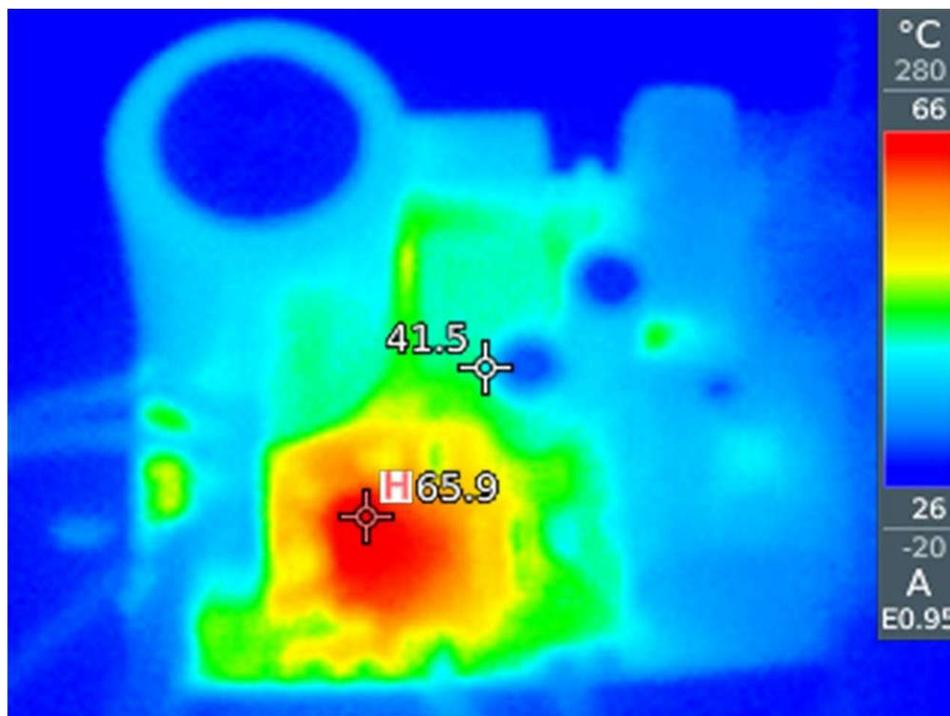


Figure 3-42. Thermal Permanence Under 300VDC, 250W, 290Hz

### 3.4.4 Thermal performance under 220VAC

Figure 3-43 shows a 50C temperature rising for DRV7308 under 220VAC, 250W, 3480RPM (290Hz) and 25C ambient temperature. This temperature rising is 9C higher than test under 300VDC power source as shown in Section 3.4.3. The reason is if board is powered with AC source, ripple current increases temperature of C67, and C67 is very close to DRV7308 (<5mm) for this compact reference design board, so high temperature rising of C67 finally increases temperature of DRV7308, hence if board area is allowed, place electrolytic capacitor relatively far away from DRV7308.

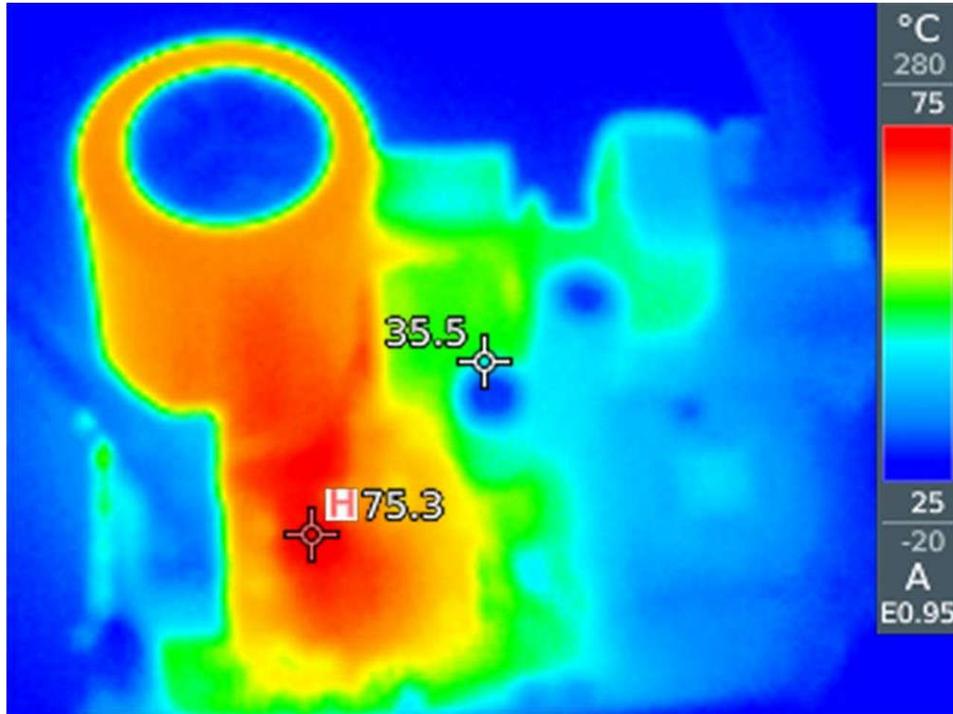


Figure 3-43. Thermal Permanence Under 220VAC, 250W, 290Hz

### 3.4.5 Overcurrent Protection by Internal CMPSS

As explained in Section 2.4.1.8, the internal CMPSS can be configured for overcurrent protection. Figure 3-44 shows the internal overcurrent protection waveform, is triggered by internal CMPSS, since both IPM\_FALUT and IPM\_CIN are not triggered. Overcurrent can be set with the following codes.

```
objSets->maxPeakCurrent_A = USER_M1_ADC_FULL_SCALE_CURRENT_A * 0.4975f;
```

- CH4 (Green): Current of phase U

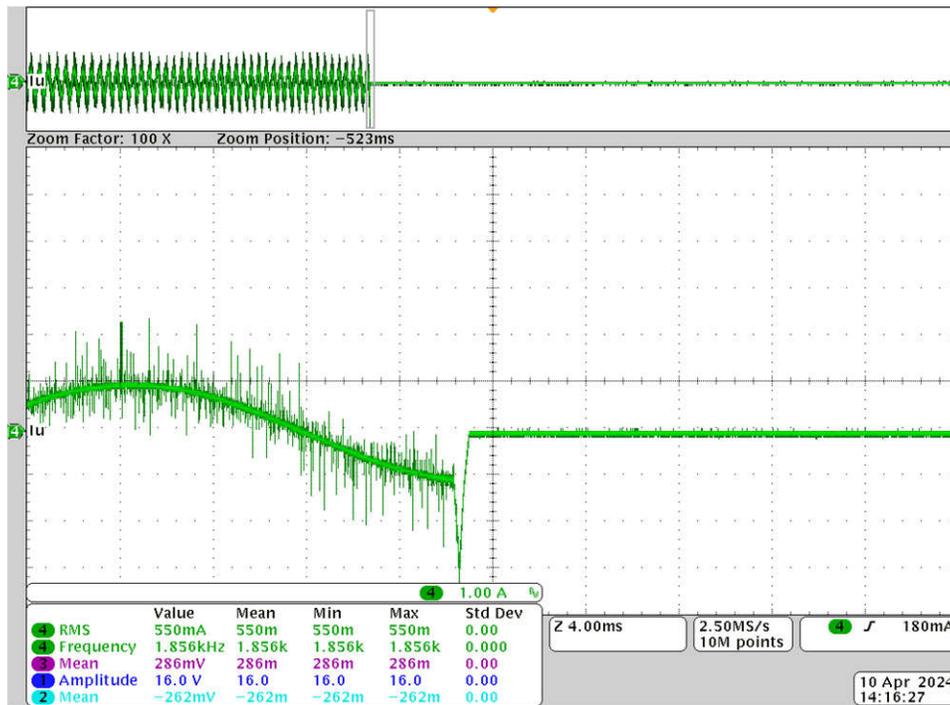


Figure 3-44. Overcurrent Protection by Internal CMPSS

### 3.4.6 IPM Efficiency with External Bias Supply under 300VDC

To measure IPM efficiency with external bias supply, remove R72 on bottom side to disable onboard bias supply, and measure input current at J5 and input voltage across C19. Probe IPM output voltage and current at J10. Then apply 15V and 3.3V to corresponding probes, finally apply DC input voltage to J5 starting from 50VDC, and slowly increase input voltage to 300VDC. Connect UART at J8 to laptop through USB-to-UART adaptor, then send command to run motor on GUI software on laptop. [Figure 3-45](#) shows the test setup.

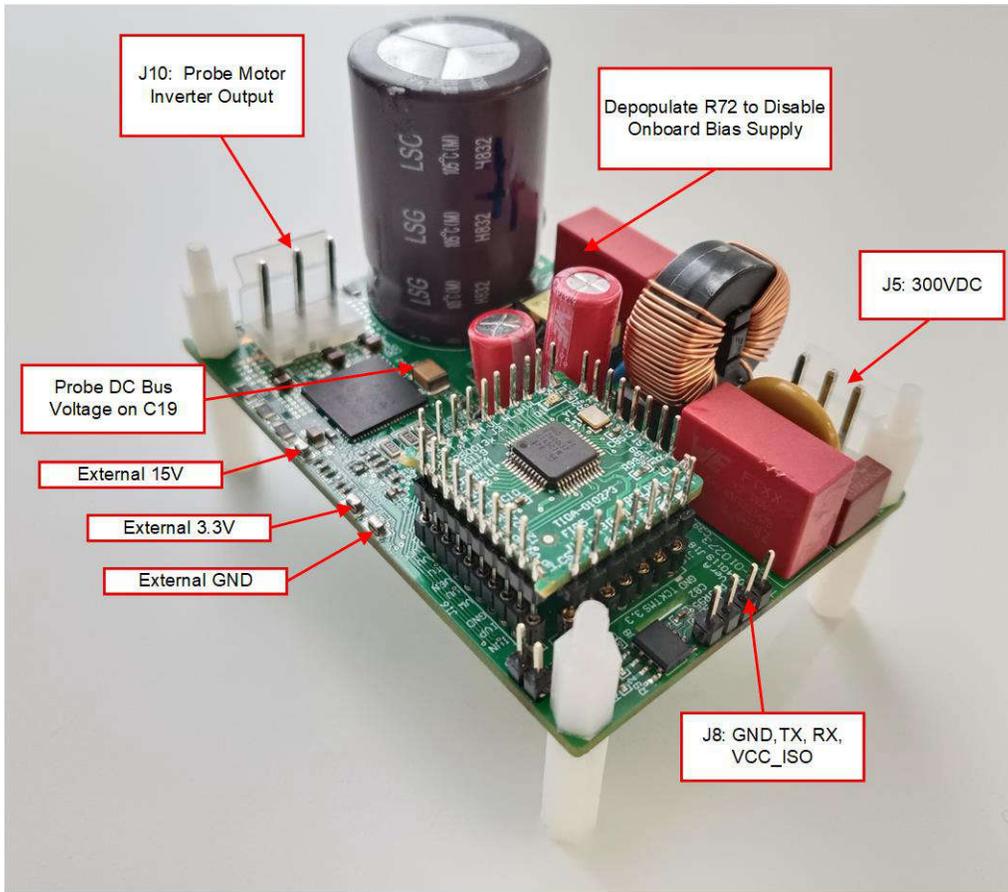


Figure 3-45. IPM Efficiency Test Setup with External Bias Supply

Figure 3-46 shows the tested IPM efficiency.

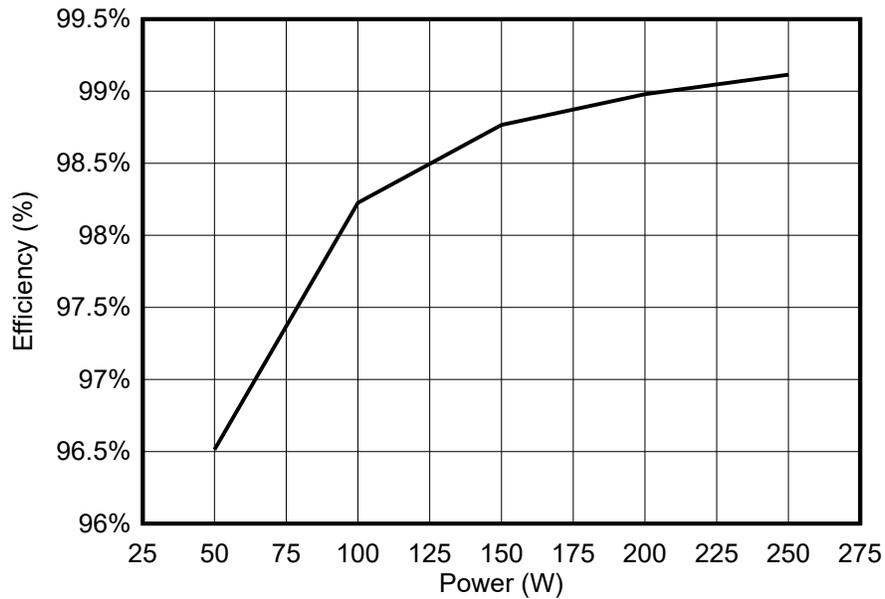
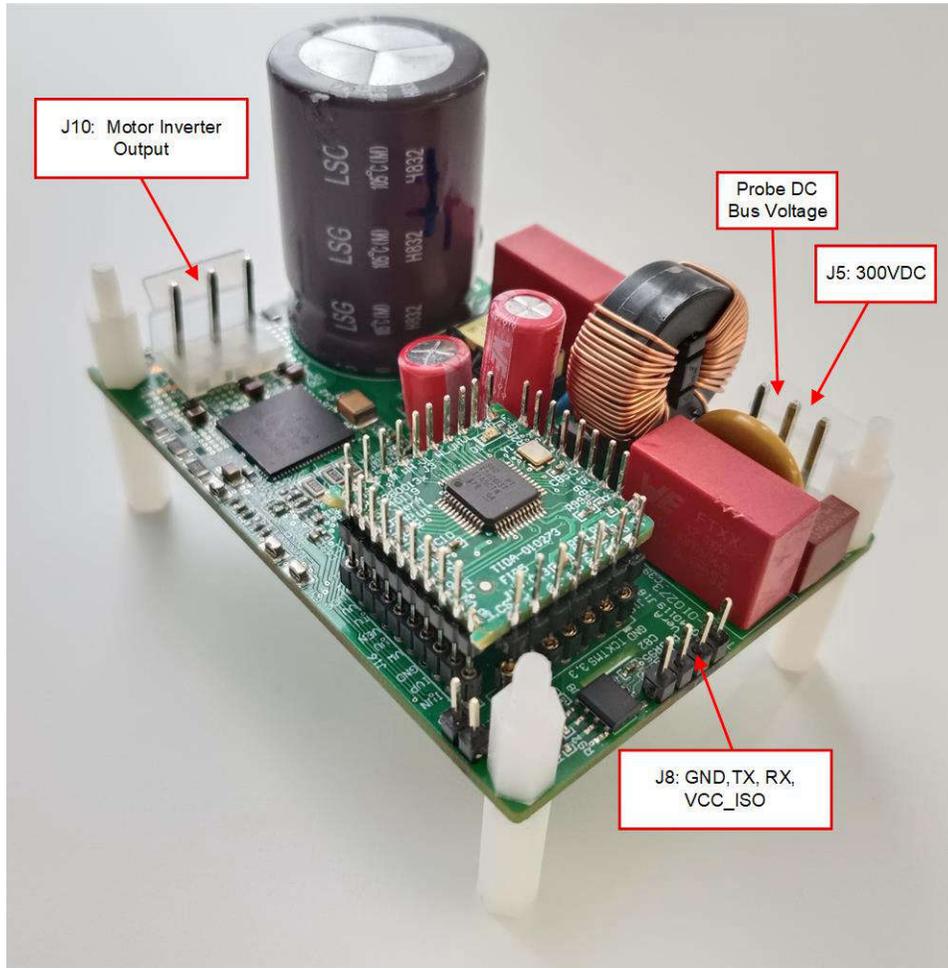


Figure 3-46. IPM Efficiency With External Bias Supply Under 300VDC

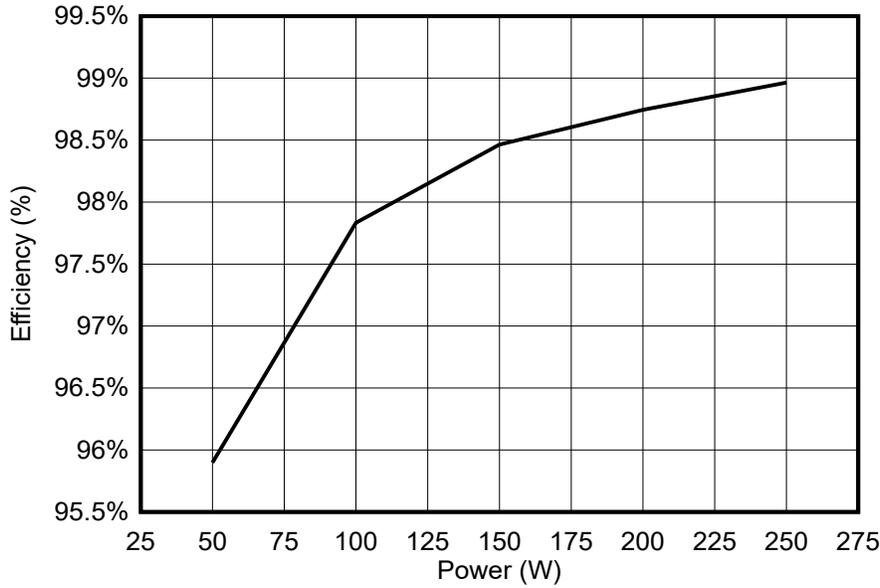
### 3.4.7 Board Efficiency with Onboard Bias Supply under 300VDC

To test board efficiency, measure input current and voltage at J5, probe IPM output voltage and current at J10. Then apply 300VDC input voltage to J5 directly, also connect UART at J8 to laptop through USB-to-UART adaptor, and send command to run motor on GUI software on laptop. [Figure 3-47](#) shows the test setup.



**Figure 3-47. Board Efficiency Test Setup With Onboard Bias Supply**

[Figure 3-48](#) shows the tested board efficiency with onboard bias supply under 300VDC.

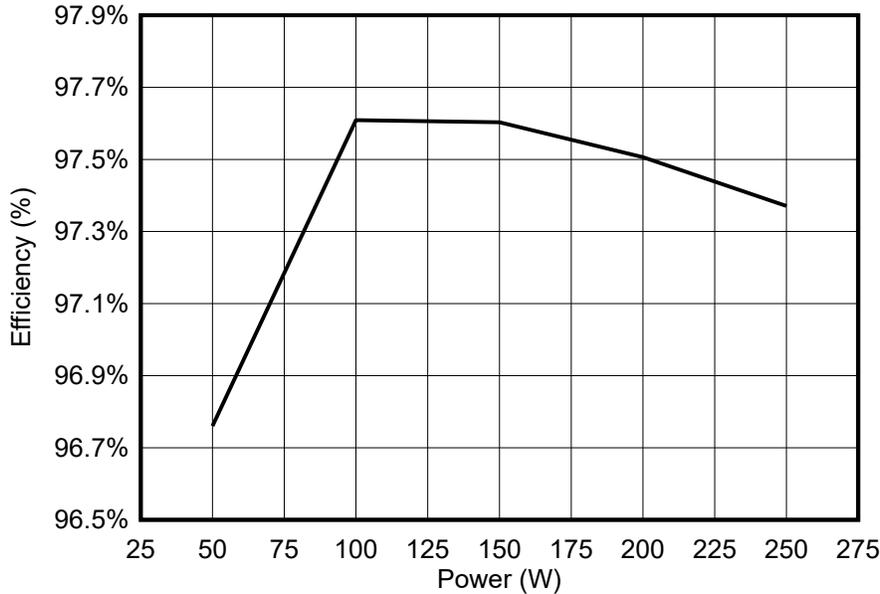


**Figure 3-48. Board Efficiency With Onboard Bias Supply Under 300VDC**

**3.4.8 Board Efficiency with External Bias Supply under 220VAC**

Test setup is similar to [Figure 3-45](#), but apply 220VAC input voltage to J5. [Figure 3-49](#) shows the efficiency test result.

Efficiency under 220VAC is lower than 300VDC input because the power loss of filter and ripple current of electrolytic capacitor.



**Figure 3-49. Board Efficiency With External Bias Supply Under 220VAC**

**3.4.9 Board Efficiency with Onboard Bias Supply under 220VAC**

Test setup is similar to [Figure 3-47](#), but apply 220VAC input voltage to J5. [Figure 3-50](#) shows the test result.

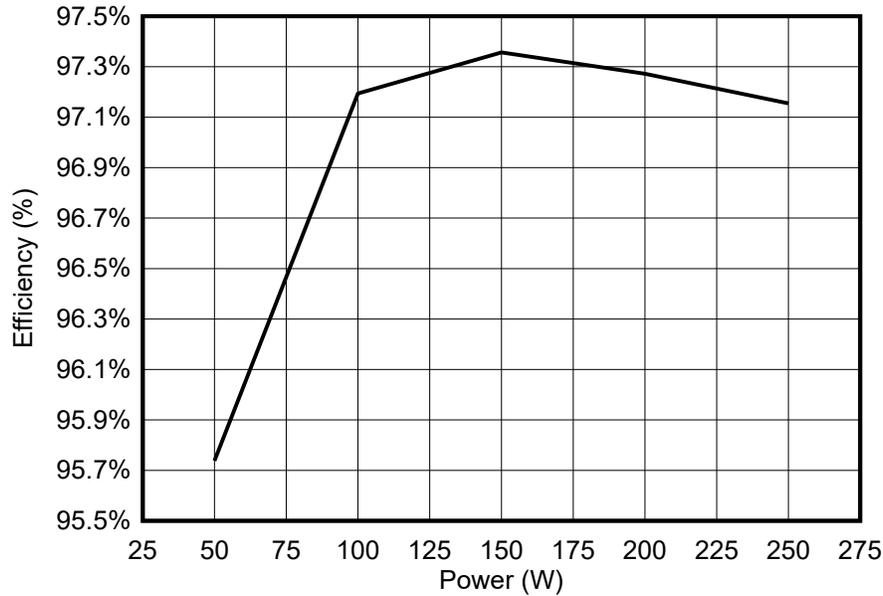


Figure 3-50. Board Efficiency with Onboard Bias Supply Under 220VAC

### 3.4.10 iTHD Test of Motor Phase Current

Apply DC300V input voltage to J5, run SC225 fan motor with GUI, and test iTHD of phase current, Figure 3-51 shows the test result. Result shows that 15kHz high switching frequency and 0.2uS dead time can produce motor phase current with very low iTHD, and can improve EMI, reduce motor noise and increase motor efficiency.

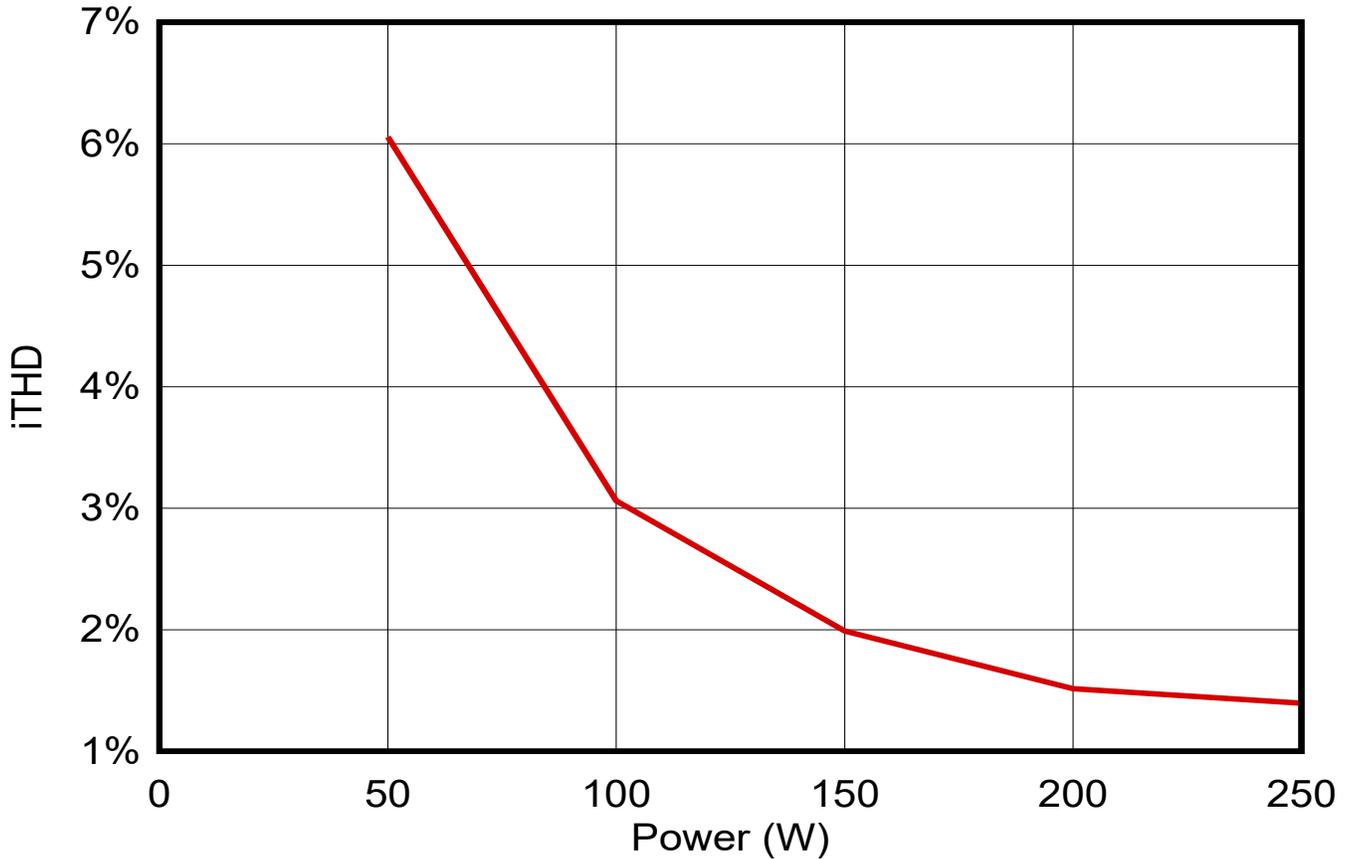


Figure 3-51. iTHD of Motor Phase Current

### 3.4.11 Standby Power Test

Figure 3-52 shows the standby power under 220VAC input, with halted F2800137 microcontroller, this can help to meet increasingly stringent low system standby power requirement, and make the system more green by energy saving.



Figure 3-52. Standby Power

## 3.5 Migrate Firmware to a New Hardware Board

If the designer wants to migrate the reference design to a hardware board, change the motor related PWM, CMPSS, ADC peripherals configuration, hardware parameters, and motor parameters, accordingly in the *hal.c*, *hal.h*, and *user\_mtr1.h* files as described in the following sections.

### 3.5.1 Configure the PWM, CMPSS, and ADC Modules

The application parameters to control the motor are written as `#define` configuring the PWM, CMPSS, and ADC modules base address in *hal.h* according to the hardware. The PWM, CMPSS, and ADC of the compressor motor defines are shown in the following codes.

Configure PWM and CMPSS base address for motor drive:

```
// EPWM
#define MTR1_PWM_U_BASE      EPWM2_BASE
#define MTR1_PWM_V_BASE      EPWM3_BASE
#define MTR1_PWM_W_BASE      EPWM4_BASE

// CMPSS->Iu/Iv/Iw
#define MTR1_CMPSS_U_BASE    CMPSSLITE3_BASE
#define MTR1_CMPSS_V_BASE    CMPSSLITE2_BASE
#define MTR1_CMPSS_W_BASE    CMPSSLITE4_BASE
```

Configure ADC base address and channels for motor drive:

```
// Three shunts
// Using ADCA/ADCC for current sensing
#define MTR1_ADC_TRIGGER_SOC    ADC_TRIGGER_EPWM2_SOCA // EPWM2_SOCA

#define MTR1_ADC_I_SAMPLEWINDOW    14
#define MTR1_ADC_V_SAMPLEWINDOW    20

#define MTR1_IW_ADC_BASE    ADCC_BASE // ADCC-A7/C3*/CMP4 -SOC1
#define MTR1_IV_ADC_BASE    ADCA_BASE // ADCA-A4*/C14/CMP2 -SOC2
#define MTR1_IU_ADC_BASE    ADCA_BASE // ADCA-A0*/C15/CMP3 -SOC1
```

```

#define MTR1_IW_ADCRES_BASE    ADCCRESULT_BASE        // ADCC-A7/C3*
#define MTR1_IV_ADCRES_BASE    ADCARESLT_BASE         // ADCA-A4*/C14
#define MTR1_IU_ADCRES_BASE    ADCARESLT_BASE         // ADCA-A0*/C15

#define MTR1_IW_ADC_CH_NUM     ADC_CH_ADCIN3         // ADCC-A7/C3*
#define MTR1_IV_ADC_CH_NUM     ADC_CH_ADCIN4         // ADCA-A4*/C14
#define MTR1_IU_ADC_CH_NUM     ADC_CH_ADCIN0         // ADCA-A0*/C15

#define MTR1_IW_ADC_SOC_NUM     ADC_SOC_NUMBER1      // ADCC-A7/C3*   -SOC1-PPB1
#define MTR1_IV_ADC_SOC_NUM     ADC_SOC_NUMBER2      // ADCA-A4*/C14   -SOC2-PPB2
#define MTR1_IU_ADC_SOC_NUM     ADC_SOC_NUMBER1      // ADCA-A0*/C15   -SOC1-PPB1

#define MTR1_IW_ADC_PPB_NUM     ADC_PPB_NUMBER1      // ADCC-A7/C3*   -SOC1-PPB1
#define MTR1_IV_ADC_PPB_NUM     ADC_PPB_NUMBER2      // ADCA-A4*/C14   -SOC2-PPB2
#define MTR1_IU_ADC_PPB_NUM     ADC_PPB_NUMBER1      // ADCA-A0*/C15   -SOC1-PPB1
    
```

Configure peripheral interrupt for motor drive control:

```

// Interrupt
#define MTR1_PWM_INT_BASE      MTR1_PWM_U_BASE        // EPWM1
#define MTR1_ADC_INT_BASE     ADCC_BASE             // ADCC-A11/C0*-SOC4
#define MTR1_ADC_INT_NUM      ADC_INT_NUMBER1       // ADCC_INT1   -SOC4
#define MTR1_ADC_INT_SOC      ADC_SOC_NUMBER4       // ADCC_INT1   -SOC4

#define MTR1_PIE_INT_NUM      INT_ADCC1             // ADCC_INT1   -SOC4
#define MTR1_CPU_INT_NUM      INTERRUPT_CPU_INT1    // ADCC_INT1-CPU_INT1
#define MTR1_INT_ACK_GROUP    INTERRUPT_ACK_GROUP1  // ADCC_INT1-CPU_INT1
    
```

Configure the connections between the ADC pin and CMPSS modules in *hal.h* based on the hardware, the details refer to the Table, Analog Pins, and Internal Connections in the [TMS320F280013x Real-Time Microcontrollers Technical Reference Manual](#)

```

// CMPSS->Iu/Iv/Iw
#define MTR1_CMPSS_U_BASE      CMPSSLITE3_BASE
#define MTR1_CMPSS_V_BASE      CMPSSLITE2_BASE
#define MTR1_CMPSS_W_BASE      CMPSSLITE4_BASE

#define MTR1_IU_CMPHP_SEL      ASYSCTL_CMPHPMUX_SELECT_3 // CMPSS3H-A0*/C15
#define MTR1_IU_CMPLP_SEL      ASYSCTL_CMPLPMUX_SELECT_3 // CMPSS3L-A0*/C15

#define MTR1_IV_CMPHP_SEL      ASYSCTL_CMPHPMUX_SELECT_2 // CMPSS2H-A4*/C14
#define MTR1_IV_CMPLP_SEL      ASYSCTL_CMPLPMUX_SELECT_2 // CMPSS2L-A4*/C14

#define MTR1_IW_CMPHP_SEL      ASYSCTL_CMPHPMUX_SELECT_4 // CMPSS4H-A7/C3*
#define MTR1_IW_CMPLP_SEL      ASYSCTL_CMPLPMUX_SELECT_4 // CMPSS4L-A7/C3*

#define MTR1_IU_CMPHP_MUX      2                       // CMPSS4H-A7/C3*
#define MTR1_IU_CMPLP_MUX      2                       // CMPSS4L-A7/C3*

#define MTR1_IV_CMPHP_MUX      0                       // CMPSS2H-A4*/C14
#define MTR1_IV_CMPLP_MUX      0                       // CMPSS2L-A4*/C14

#define MTR1_IW_CMPHP_MUX      1                       // CMPSS3H-A0*/C15
#define MTR1_IW_CMPLP_MUX      1                       // CMPSS3L-A0*/C15
    
```

Configure the trip signals from CMPSS to be passed to EPWM and GPIO output in *hal.h* based on the hardware, the details refer to Table, ePWM X-BAR MUX Configuration Table and Table, OUTPUT X-BAR MUX Configuration Table in [TMS320F280013x Real-Time Microcontrollers Technical Reference Manual](#).

```

// XBAR-EPWM
#define MTR1_XBAR_TRIP_ADDR_L   XBAR_O_TRIP8MUX0T015CFG
#define MTR1_XBAR_TRIP_ADDR_H   XBAR_O_TRIP8MUX16T031CFG

#define MTR1_XBAR_INPUT1        XBAR_INPUT1
#define MTR1_TZ_OSHT1           EPWM_TZ_SIGNAL_OSHT1

#define MTR1_XBAR_TRIP          XBAR_TRIP8
#define MTR1_DCTRIPIN           EPWM_DC_COMBINATIONAL_TRIPIN8y

// XBAR-EPWM->Iu/Iv/Iw
#define MTR1_IU_XBAR_EPWM_MUX    XBAR_EPWM_MUX04_CMPSS3_CTRIPH_OR_L // CMPSS3-HP&LP, A0*/C15
#define MTR1_IV_XBAR_EPWM_MUX    XBAR_EPWM_MUX02_CMPSS2_CTRIPH_OR_L // CMPSS2-HP&LP, A4*/C14
#define MTR1_IW_XBAR_EPWM_MUX    XBAR_EPWM_MUX06_CMPSS4_CTRIPH_OR_L // CMPSS4-HP&LP, A7/C3*
    
```

```
#define MTR1_IU_XBAR_MUX      XBAR_MUX04      // CMPSS3-HP&LP, A0*/C15
#define MTR1_IV_XBAR_MUX      XBAR_MUX02      // CMPSS2-HP&LP, A4*/C14
#define MTR1_IW_XBAR_MUX      XBAR_MUX06      // CMPSS4-HP&LP, A7/C3*
```

The related ADC channels are used for motor-current sensing which pins are internally connected to the Comparator Subsystem (CMPSS), configure the CMPSS registers in the HAL\_setupCMPSSs() function in the *hal.c* file as shown in the following codes. Three CMPSS modules are used to implement positive and negative overcurrent protection of U-phase, V-phase, and W-phase of the motor.

```
void HAL_setupCMPSSMTR(HAL_MTR_Handle handle)
{
    HAL_MTR_Obj *obj = (HAL_MTR_Obj *)handle;

    #if defined(DMCPFC_REV3P2) || defined(DMCPFC_REV3P1)
    #if !defined(MOTOR1_DCLINKSS) || !defined(MOTOR2_DCLINKSS)
    uint16_t cmpsaDACH;
    #endif // !(MOTOR1_DCLINKSS || MOTOR2_DCLINKSS)
    uint16_t cmpsaDACL;
    #else // !MOTOR1_DCLINKSS, Three-shunt
    cmpsaDACH = MTR1_CMPSS_DACH_VALUE;
    cmpsaDACL = MTR1_CMPSS_DACL_VALUE;

    ASysCtl_selectCMPHPMux(MTR1_IU_CMPHP_SEL, MTR1_IU_CMPHP_MUX);

    ASysCtl_selectCMPHPMux(MTR1_IV_CMPHP_SEL, MTR1_IV_CMPHP_MUX);

    ASysCtl_selectCMPLPMux(MTR1_IW_CMPLP_SEL, MTR1_IW_CMPLP_MUX);
    #endif
    return;
} // end of HAL_setupCMPSSs() function
```

The CMPSS-generated signals go to the X-Bar, where signals can be combined in different and unique fashions to flag unique trip events from multiple sources including external TZ signal from IPM #Fault to implement the fault protection. The faults include the overcurrent signals from the CMPSS and the fault indicator output from the power module. Configure the XBAR registers in HAL\_setupMtrFaults() function in the *hal.c* file as shown in the following codes.

```
void HAL_setupMtrFaults(HAL_MTR_Handle handle)
{
    HAL_MTR_Obj *obj = (HAL_MTR_Obj *)handle;
    uint16_t cnt;

    // Configure TRIP 7 to OR the High and Low trips from both
    // comparator 5, 3 & 1, clear everything first
    EALLOW;
    HWREG(XBAR_EPWM_CFG_REG_BASE + MTR1_XBAR_TRIP_ADDR_L) = 0;
    HWREG(XBAR_EPWM_CFG_REG_BASE + MTR1_XBAR_TRIP_ADDR_H) = 0;
    EDIS;

    ...
    // What do we want the OST/CBC events to do?
    // TZA events can force EPWMxA
    // TZB events can force EPWMxB
    EPWM_setTripZoneAction(obj->pwmHandle[cnt],
                           EPWM_TZ_ACTION_EVENT_TZA,
                           EPWM_TZ_ACTION_LOW);

    EPWM_setTripZoneAction(obj->pwmHandle[cnt],
                           EPWM_TZ_ACTION_EVENT_TZB,
                           EPWM_TZ_ACTION_LOW);

    ...
    // Clear any spurious fault
    EPWM_clearTripZoneFlag(obj->pwmHandle[0], HAL_TZFLAG_INTERRUPT_ALL);
    EPWM_clearTripZoneFlag(obj->pwmHandle[1], HAL_TZFLAG_INTERRUPT_ALL);
    EPWM_clearTripZoneFlag(obj->pwmHandle[2], HAL_TZFLAG_INTERRUPT_ALL);

    return;
}
```

Configure the GPIOs based on the hardware in HAL\_setupGPIOs() in the *hal.c* file as shown in the following codes.

```
void HAL_setupGPIOs(HAL_Handle handle)
{
    ...
    // GPIO2->EPWM2A->M1_UH
    GPIO_setPinConfig(GPIO_2_EPWM2_A);
    GPIO_writePin(2, 0);
    GPIO_setDirectionMode(2, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(2, GPIO_PIN_TYPE_STD);

    // GPIO3->EPWM2B->M1_UL
    GPIO_setPinConfig(GPIO_3_EPWM2_B);
    GPIO_writePin(3, 0);
    GPIO_setDirectionMode(3, GPIO_DIR_MODE_OUT);
    GPIO_setPadConfig(3, GPIO_PIN_TYPE_STD);
    ...
    return;
} // end of HAL_setupGPIOs() function
```

The configuration codes need to be changed in HAL\_enableMtrPWM() and HAL\_clearMtrFaultStatus() in the *hal.h* file as below marked in **bold** according to the used CMPSS for motor control.

```
static inline void HAL_enableMtrPWM(HAL_MTR_Handle handle)
{
    HAL_MTR_Obj *obj = (HAL_MTR_Obj *)handle;

    obj->flagEnablePWM = true;

    #if defined(DMCPFC_REV3P2) || defined(DMCPFC_REV3P1)
    if(obj->motorNum == MTR_1)
    {
    #if defined(MOTOR1_DCLINKSS)
        // Clear any comparator digital filter output latch
        CMPSS_clearFilterLatchLow(obj->cmpssHandle[0]);
    #else // !MOTOR1_DCLINKSS
        // Clear any comparator digital filter output latch
        CMPSS_clearFilterLatchHigh(obj->cmpssHandle[0]);

        CMPSS_clearFilterLatchHigh(obj->cmpssHandle[1]);

        CMPSS_clearFilterLatchLow(obj->cmpssHandle[2]);
    }
    return;
} // end of HAL_enableMtrPWM() function
```

```
static inline void HAL_clearMtrFaultStatus(HAL_MTR_Handle handle)
{
    HAL_MTR_Obj *obj = (HAL_MTR_Obj *)handle;

    #if defined(HVMTRPFC_REV1P1) || defined(WMINVBRD_REV1P0) || defined(TIDSMPCF_REV3P2)
    // Clear any comparator digital filter output latch
    CMPSS_clearFilterLatchHigh(obj->cmpssHandle[0]);
    CMPSS_clearFilterLatchLow(obj->cmpssHandle[0]);

    CMPSS_clearFilterLatchHigh(obj->cmpssHandle[1]);
    CMPSS_clearFilterLatchLow(obj->cmpssHandle[1]);

    CMPSS_clearFilterLatchHigh(obj->cmpssHandle[2]);
    CMPSS_clearFilterLatchLow(obj->cmpssHandle[2]);
    }
    return;
} // end of HAL_clearMtrFaultStatus() function
```

### 3.5.2 Setup Hardware Board Parameters

The *user\_mtr1.h* file is where all user parameters are stored for motor control. The maximum phase current and phase voltage at the input to the AD converter, these values are hardware-dependent and need to be based on the current and voltage sensing and scaling to the ADC input. The number of current sensors and voltage (phase) sensors used are defined in *user\_mtr1.h* that are hardware dependent.

All of the configurable parameters are defined in the *user\_mtr1.h* file. These parameters can be calculated using the *Motor\_Drive\_Parameters\_Calculation.xlsx* Microsoft® Excel® spreadsheet. This file is included with the TIDA-010273 archive file at the folder `..\solutions\tida_010273_GaNIInv\docs` to calculate these values and copy these parameters marked **bold** to *user\_mtr1.h* as shown in the following codes.

```

///! \brief Defines the maximum voltage at the AD converter
/// Full scale voltage of AD converter, not the current voltage
#define USER_M1_ADC_FULL_SCALE_VOLTAGE_V          (404.1292683f)

///! \brief Defines the analog voltage filter pole location, Hz
///!
#define USER_M1_VOLTAGE_FILTER_POLE_HZ          (416.3602877f)

///! \brief Defines the maximum current at the AD converter
#define USER_M1_ADC_FULL_SCALE_CURRENT_A        (6.6f)

```

### 3.5.3 Configure Faults Protection Parameters

Fault management is implemented in this system that includes overcurrent, overvoltage, undervoltage, stall, overload, start-up failed. The faults protection parameters are defined in *user\_mtr1.h* as shown in the following codes, which are hardware board, motors, and system dependent.

```

///! \brief motor over current threshold
#define USER_MOTOR1_OVER_CURRENT_A              (3.0f)           // A

///! \brief motor lost phase current threshold
#define USER_M1_LOST_PHASE_CURRENT_A            (0.02f)

///! \brief motor unbalance ratio percent threshold
#define USER_M1_UNBALANCE_RATIO                 (0.2f)

```

```

///! \brief DC bus over voltage threshold
#define USER_M1_OVER_VOLTAGE_FAULT_V            (380.0f)

///! \brief DC bus over voltage threshold
#define USER_M1_OVER_VOLTAGE_NORM_V            (350.0f)

///! \brief DC bus under voltage threshold
#define USER_M1_UNDER_VOLTAGE_FAULT_V          (100.0f)

```

### 3.5.4 Setup Motor Electrical Parameters

The parameters provided in *user\_mtr1.h* for PMSM and BLDC motors are listed as shown in the following codes. The motor parameters can be identified if the FAST technique is implemented on this motor or by getting the parameters from the motor data sheet.

```

#define USER_MOTOR1_TYPE                        MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS              (5)
#define USER_MOTOR1_Rr_Ohm                      (0.0f)
#define USER_MOTOR1_Rs_Ohm                      (4.5f)
#define USER_MOTOR1_Ls_d_H                      (0.0196f)
#define USER_MOTOR1_Ls_q_H                      (0.0196f)
#define USER_MOTOR1_RATED_FLUX_VpHz            (0.441f)

```

## 3.6 Getting Started MSPM0 Firmware

Contact the local TI sales representative for firmware for the MSPM0G1507 daughterboard.

## 4 Design and Documentation Support

### 4.1 Design Files

#### 4.1.1 Schematics

To download the schematics, see the design files at [TIDA-010273](#).

#### 4.1.2 Bill of Materials

To download the bill of materials (BOM), see the design files at [TIDA-010273](#).

#### 4.1.3 PCB Layout Recommendations

This reference design is implemented using a PCB with two-layer, 2oz copper with the bottom-side SMD component placement to save cost and board area. There are several important aspects to remember while designing the PCB. In the following bullets, system-level placement and layout of each block is explained.

- Separate components (or singles) into high and low voltage, high and low current, high and low independence groups. Place and route low-voltage and high-impedance components and signals together, such as microcontroller-related signals and the input side of IPM. Use a whole copper pour to provide an integrated GND plane for these area. AC input, filter, and rectifiers and IPM output sides are high-voltage, high-current and low-impedance parts and signals, route them with wider traces or copper pours to provide high-current paths, and also separate them from above low voltage and high-impedance signals to reduce interference.
- Components in the high power path are kept on the outer edges of the PCB with the minimum distance possible. The microcontroller is placed at the center considering the optimum distance from all the power blocks that need to be controlled. Pin assignment is set to minimize the control signal or feedback signal trace distance and the crossing between analog and digital signals.
- AC Line Protection and EMI Filter
  - AC line protection components are closely placed within the minimum distance to the connection path. Earth connection guarding is provided around protection and EMI filter circuit.
- Motor Drive
  - For the high ripple requirement, the motor drive is placed as close as possible to the film capacitor and DC bus capacitor bank.
  - The low-side shunt resistor method with four-wire sensing is implemented for current sensing. A differential pair with impedance matching resistor is used to connect the sensing signal from the shunt resistors to the op-amp circuit. Shunt resistors are placed near the module with an immediate ground copper plane connection.
- Auxiliary Power Supply
  - The GND of the auxiliary power supply connects the DC bus capacitor bank directly and independently to separate low current from high current and high frequency GND trace for the inverter.

#### 4.1.4 Altium Project

To download the Altium project files, see the design files at [TIDA-010273](#).

#### 4.1.5 Gerber Files

To download the Gerber files, see the design files at [TIDA-010273](#).

### 4.2 Software Files

[CCSTUDIO](#)

Code Composer Studio Integrated Development Environment

[C2000WARE-MOTORCONTROL-SDK](#)

TIDA-010273 hardware-specific software design files

### 4.3 Documentation Support

1. Texas Instruments, [TMS320F280013x Microcontrollers](#) data sheet
2. Texas Instruments, [TMS320F280013x Real-Time Microcontrollers](#) technical reference manual
3. Texas Instruments, [InstaSPIN-FOC and InstaSPIN-MOTION](#) user's guide
4. Texas Instruments, [Motor Control SDK Universal Project and Lab](#) user's guide
5. Texas Instruments, [C2000™ Software Frequency Response Analyzer \(SFRA\) Library and Compensation Designer](#) user's guide
6. Texas Instruments, [Sensorless-FOC for PMSM With Single DC-Link Shunt](#) application note
7. Texas Instruments, [C2000 SysConfig](#) application note

### 4.4 Support Resources

[TI E2E™ support forums](#) are an engineer's go-to source for fast, verified answers and design help — straight from the experts. Search existing answers or ask your own question to get the quick design help you need.

Linked content is provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's [Terms of Use](#).

### 4.5 Trademarks

InstaSPIN™, Code Composer Studio™, and TI E2E™ are trademarks of Texas Instruments.

Arm® and Cortex® are registered trademarks of Arm Limited.

Microsoft®, Windows®, and Excel® are registered trademarks of Microsoft Corporation.

All trademarks are the property of their respective owners.

## 5 About the Author

**HELY ZHANG** is a System Application Engineer at Texas Instruments, where he is responsible for developing home appliance related power delivery and motor inverters. Hely earned his masters degree from Anhui University of Science and Technology with Power electronics in 2002, and worked in SolarEdge and General Electric before joining TI.

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 will 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](#) or other applicable terms available either on [ti.com](https://www.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.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2024, Texas Instruments Incorporated