

High-Voltage HEV and EV HVAC eCompressor Motor Control Reference Design



Description

The TIDM-02012 is a High-Voltage, 5 kW reference design built for hybrid electric vehicle (HEV) and electric vehicle (EV) eCompressor applications controlled by a mid-performance C2000™ F28003x real-time MCU. The reference design is designed to evaluate with both 400 V and 800 V DC-bus, covering the market-trending of higher battery voltage. A controlCARD-based design enables users to evaluate multiple MCU options, and is scalable to support other devices within the C2000™ portfolio including future roadmap devices to meet growing cybersecurity, functional safety, and other automotive market needs.

Resources

[TIDM-02012](#)

Design Folder

[TMS320F280039C](#)

Product Folder

[UCC21530-Q1, OPA607-Q1](#)

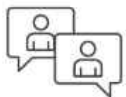
Product Folder

[LM25184-Q1, TCAN1044A-Q1](#)

Product Folder

[C2000WARE-MOTORCONTROL-SDK](#)

Tool Folder



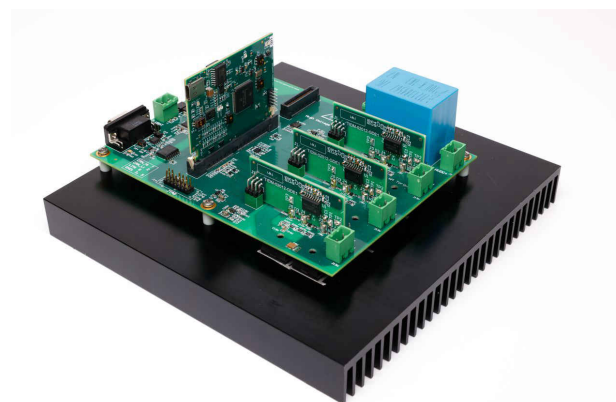
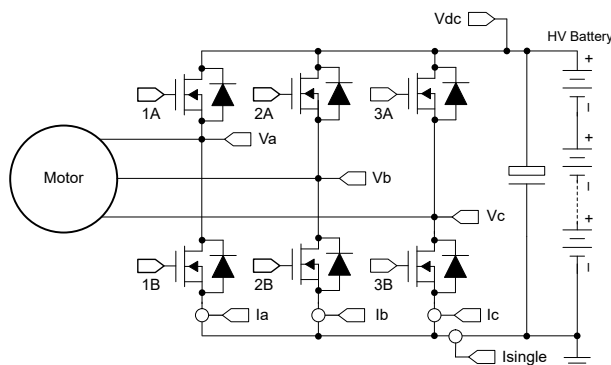
[Ask our TI E2E™ support experts](#)

Features

- Sensorless FOC using the InstaSPIN FAST observer running on a cost-optimized C2000 real-time microcontroller.
- Hardware and software available with this reference design are tested and ready-to-use to help accelerate development time to market.
- Incremental software builds to verify different software modules step by step and Optional software features for supporting field-weakening control, MTPA, over modulation, and vibration compensation.
- Wide speed range operation starting from 5 electric Hz.
- CAN FD and LIN interface support, motor over-current protection using on-chip comparators.

Applications

- [Automotive HVAC compressor](#)



1 System Description

For decades, the internal combustion engine (ICE) has run the car as well as the heating and cooling systems. As the automotive industry electrifies and transitions to hybrid electric vehicles (HEVs) with small combustion engines or fully electric vehicles (EVs) with no engine at all, the heating, ventilation and air-conditioning (HVAC) systems use a PMSM motor to drive the eCompressor from high-voltage power. The eCompressor can be used to support cooling (air-conditioning), heating (heat pump), and managing thermal of other systems in the powertrain such as the battery pack and traction drive.

Today's eCompressor in HEV/EV must meet a growing list of demands on low cost, smaller size, less vibration and noise, higher power level and higher energy efficiency. This reference design demonstrates controlling eCompressor motor using field-oriented control (FOC) without a position sensor. The overall system helps users to reduce the number of critical components in the bill of materials, improve efficiency, reduce vibration and noise, and save development time. This reference design is based on TMS320F28003x real-time controller series and is scalable with future MCU in the portfolio with the controlCARD-based form factor.

Common communication interfaces in automotive including CAN FD and LIN are supported to ease customer evaluation process. Both multi-shunt and single-shunt current sensing modes are supported for customers with different design goals to use this design for evaluation. Isolated gate driver is chosen to enable support of 400V and 800V DC bus voltage, as the trend of the industry is moving toward higher voltage level.

WARNING

TI intends this reference design to be operated in a lab environment only and does not consider the reference design 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.

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 with key TI components highlighted.

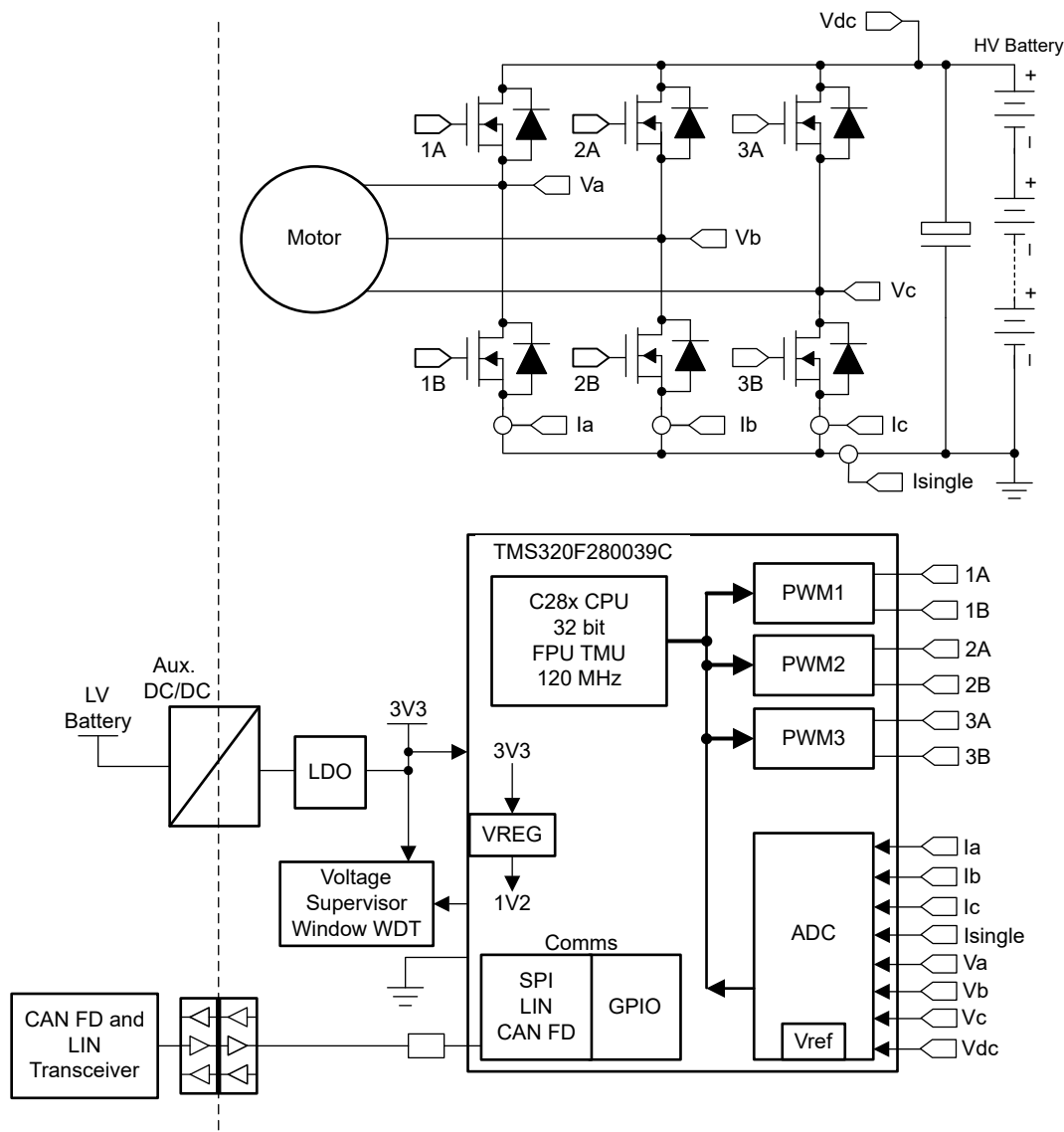


Figure 2-1. TIDM-02012 eCompressor Block Diagram

2.2 Design Considerations

The design drives a HEV and EV eCompressor with mid-performance C2000™ F28003x real-time MCU. Highly noise-immune current and voltage sensing processes are necessary for precise motor drive. The following detail the sensing and drive circuit that are used on this design. The hardware design files are available at [TIDM-02012](#).

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 their respective product data sheet.

2.3.1 TMS320F280039C

The [TMS320F28003x](#) is a member of the C2000 real-time microcontroller family of scalable, ultra-low latency devices designed for efficiency in power electronics, including but not limited to: high power density, high switching frequencies, and supporting the use of GaN and SiC technologies. The [Essential Guide for Developing with C2000™ Real-Time Microcontroller](#) application note is based on TI's 32-bit C28x DSP core, which provides 120 MHz 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 Floating-Point Unit (FPU), Trigonometric Math Unit (TMU), and VCRC (Cyclical Redundancy Check) extended instruction sets, speeding up common algorithms key to real-time control systems. The CLA is an independent 32-bit floating-point math accelerator that executes in parallel with the C28x CPU, allowing significant offloading of common tasks.

The F28003x supports up to 384KB (192KW) of flash memory divided into three 128KB (64KW) banks, which enable programming and execution in parallel. Up to 69KB (34.5KW) of on-chip SRAM is also available to supplement the flash memory.

High-performance analog blocks are integrated on the F28003x real-time microcontroller (MCU) and are closely coupled with the processing and PWM units to provide optimal real-time signal chain performance. Sixteen PWM channels, all supporting frequency-independent resolution modes, enable control of various power stages from a 3-phase inverter to power factor correction and advanced multi-level power topologies.

2.3.2 UCC21530-Q1

The [UCC21530-Q1](#) is an isolated dual-channel gate driver with 4-A source and 6-A sink peak current. The gate driver is designed to drive IGBTs, Si MOSFETs, and SiC MOSFETs up to 5-MHz with best-in-class propagation delay and pulse-width distortion.

The input side is isolated from the two output drivers by a 5.7-kV_{RMS} reinforced isolation barrier, with a minimum of 100-V/ns common-mode transient immunity (CMTI). Internal functional isolation between the two secondary-side drivers allows a working voltage of up to 1850 V.

This driver can be configured as two low-side drivers, two high-side drivers, or a half-bridge driver with programmable dead time (DT). The EN pin pulled low shuts down both outputs simultaneously and allows for normal operation when left open or pulled high. As a fail-safe measure, primary-side logic failures force both outputs low.

2.3.3 OPA607-Q1

The [OPA607-Q1](#) device is a decompensated, minimum gain of 6 V/V stable, general-purpose CMOS operational amplifier LM25184-Q1 with low noise of 3.8 nV/√Hz and a GBW of 50 MHz. The low voltage offset drift (dVOS/dT) and wide bandwidth of the OPAx607-Q1 devices make them attractive for low cost general-purpose applications like low side current sensing and TIA (transimpedance amplifier). The high-impedance CMOS inputs make the OPAx607-Q1 devices ideal amplifiers to interface with sensors with high output impedance (for example, piezoelectric transducers).

2.3.4 LM25184-Q1

The [LM25184-Q1](#) is a primary-side regulated (PSR) flyback converter with high efficiency over a wide input voltage range of 4.5 V to 42 V. The isolated output voltage is sampled from the primary-side flyback voltage. The high level of integration results in a simple, reliable, and high-density design with only one component crossing the isolation barrier. Boundary conduction mode (BCM) switching enables a compact magnetic solution and better than ±1.5% load and line regulation performance. An integrated 65-V power MOSFET provides output power up to 15 W with enhanced headroom for line transients.

The LM25184-Q1 simplifies the implementation of isolated DC/DC supplies with optional features to optimize performance for the target end equipment. The output voltage is set by one resistor, while an optional resistor improves output voltage accuracy by negating the thermal coefficient of the flyback diode voltage drop. Additional features include an internally-fixed or externally-programmable soft-start, precision enable input with hysteresis for adjustable line UVLO, hiccup-mode overload protection, and thermal shutdown protection with automatic recovery.

2.3.5 TCAN1044A-Q1

The [TCAN1044A-Q1](#) is a high speed controller area network (CAN) transceiver that meets the physical layer requirements of the ISO 11898-2:2016 high-speed CAN specification. The transceivers have certified electromagnetic compatibility (EMC) operation making it an ideal choice for classical CAN and CAN FD networks up to 5 megabits per second (Mbps). Up to 8 Mbps operation in simpler networks is possible with these devices. The TCAN1044AV-Q1 includes internal logic level translation through the V_{IO} pin to allow for interfacing the transceiver I/O's directly to 1.8-V, 2.5-V, 3.3-V, or 5-V logic levels. The transceiver supports a low-power standby mode and wake over CAN which is compliant to the ISO 11898-2:2016 defined wake-up pattern (WUP).

2.4 System Design Theory

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

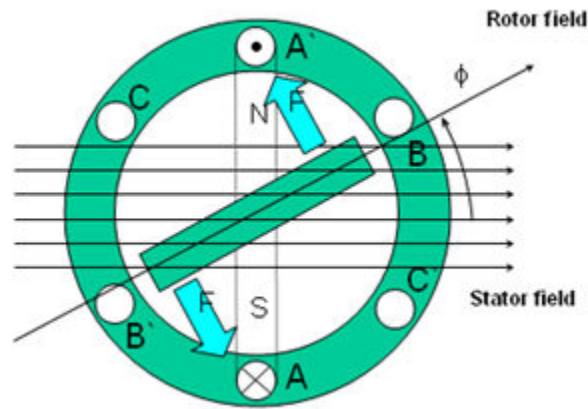


Figure 2-2. The Interaction Between the Rotating Stator Flux, and the Rotor Flux Produces a Torque

2.4.2 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, we can implement advanced control strategies, which use mathematical transformations to decouple the torque generation and the magnetization functions in PM motors. Such de-coupled torque and magnetization control is commonly called rotor flux oriented control, or simply Field Oriented Control (FOC).

In a direct current (DC) Motor, the excitation for the stator and rotor is independently controlled, the produced torque and the flux can be independently tuned as shown in Figure 2-3. The strength of the field excitation (for example, the magnitude of the field excitation current) sets the value of the flux. The current through the rotor windings determines how much torque is produced. The commutator on the rotor plays an interesting part in the torque production. The commutator is in contact with the brushes, and the mechanical construction is designed to switch into the circuit the windings that are mechanically aligned to produce the maximum torque. This arrangement then means that the torque production of the machine is fairly near optimal all the time. The key point here is that the windings are managed to keep the flux produced by the rotor windings orthogonal to the stator field.

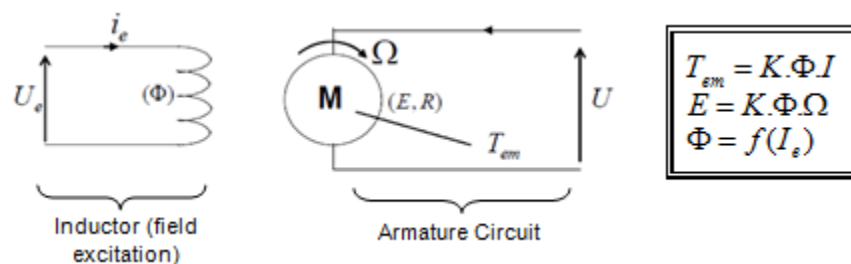


Figure 2-3. Flux and Torque are Independently Controlled in DC Motor Model

The goal of the FOC (also called vector control) on synchronous and asynchronous machine is to be able to separately control the torque producing and magnetizing flux components. FOC control will allow us to decouple the torque and the magnetizing flux components of stator current. With decoupled control of the magnetization, the torque producing component of the stator flux can now be thought of as independent torque control. To decouple the torque and flux, it is necessary to engage several mathematical transforms, and this is where the microcontrollers add the most value. The processing capability provided by the microcontrollers enables these mathematical transformations to be carried out very quickly. This in turn implies that the entire algorithm controlling the motor can be executed at a fast rate, enabling higher dynamic performance. In addition to the decoupling, a dynamic model of the motor is now used for the computation of many quantities such as rotor flux angle and rotor speed. This means that their effect is accounted for, and the overall quality of control is better.

According to the electromagnetic laws, the torque produced in the synchronous machine is equal to vector cross product of the two existing magnetic fields as Equation 1.

$$\tau_{em} = \vec{B}_{stator} \times \vec{B}_{rotor} \quad (1)$$

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

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

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

- The ease of reaching constant reference (torque component and flux component of the stator current)
- The ease of applying direct torque control because in the (d, q) reference frame the expression of the torque is defined in [Equation 2](#).

$$\tau_{em} \propto \psi_R \times i_{sq} \quad (2)$$

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

Space Vector Definition and Projection

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

$$\vec{i}_s = i_a + \alpha i_b + \alpha^2 i_c \quad (3)$$

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

[Figure 2-4](#) shows the stator current complex space vector.

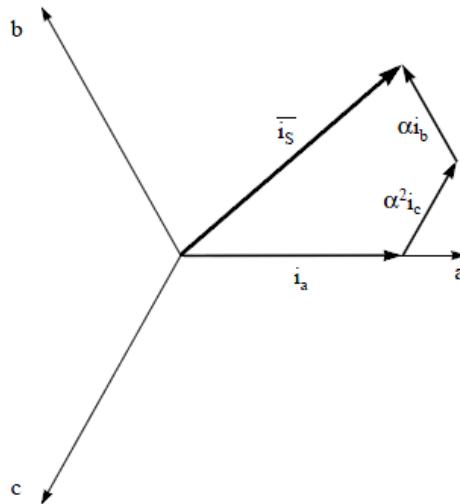


Figure 2-4. Stator Current Space Vector and its Component in (a,b,c) Frame

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

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

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

The space vector can be reported in another reference frame with only two orthogonal axis called (α, β) . Assuming that the axis a and the axis α are in the same direction we have the following vector diagram as shown in Figure 2-5.

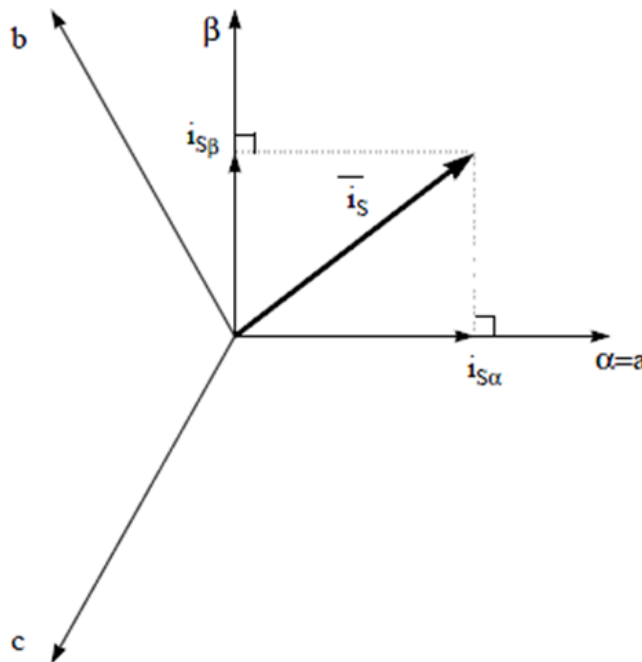


Figure 2-5. Stator Current Space Vector in the Stationary Reference Frame

The projection that modifies the 3-phase system into the (α, β) 2-dimension orthogonal system is presented in Equation 4.

$$\begin{aligned} i_{s\alpha} &= i_a \\ i_{s\beta} &= \frac{1}{\sqrt{3}}i_a + \frac{2}{\sqrt{3}}i_b \end{aligned} \quad (4)$$

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

The (α, β) $\Rightarrow$ (d, q) Park Transformation

This is the most important transformation in the FOC. In fact, this projection modifies a 2-phase orthogonal system (α , β) in the (d , q) rotating reference frame. If we consider the d axis aligned with the rotor flux, [Figure 2-6](#) shows the relationship for the current vector from the two reference frame.

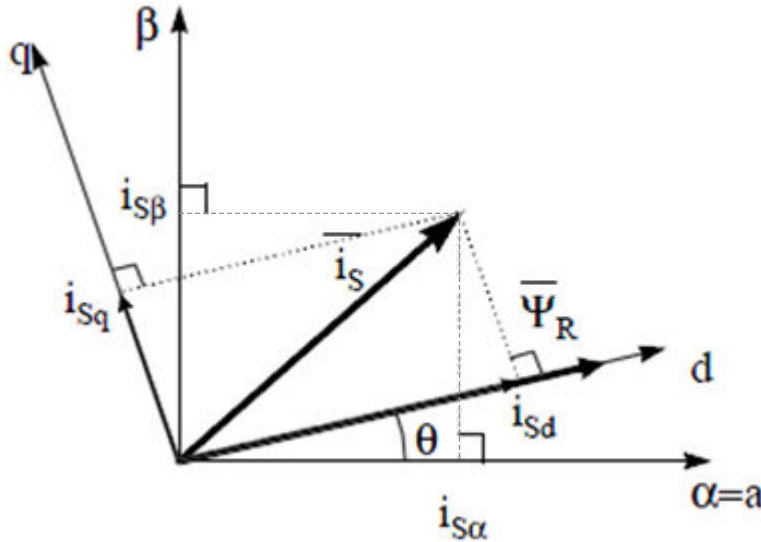


Figure 2-6. Stator Current Space Vector in The d,q Rotating Reference Frame

The flux and torque components of the current vector are determined by [Equation 5](#).

$$\begin{aligned} i_{sd} &= i_{s\alpha}\cos(\theta) + i_{s\beta}\sin(\theta) \\ i_{sq} &= -i_{s\alpha}\sin(\theta) + i_{s\beta}\cos(\theta) \end{aligned} \quad (5)$$

where θ is the rotor flux position

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

The Basic Scheme of FOC for AC Motor

[Figure 2-7](#) summarizes the basic scheme of torque control with FOC:

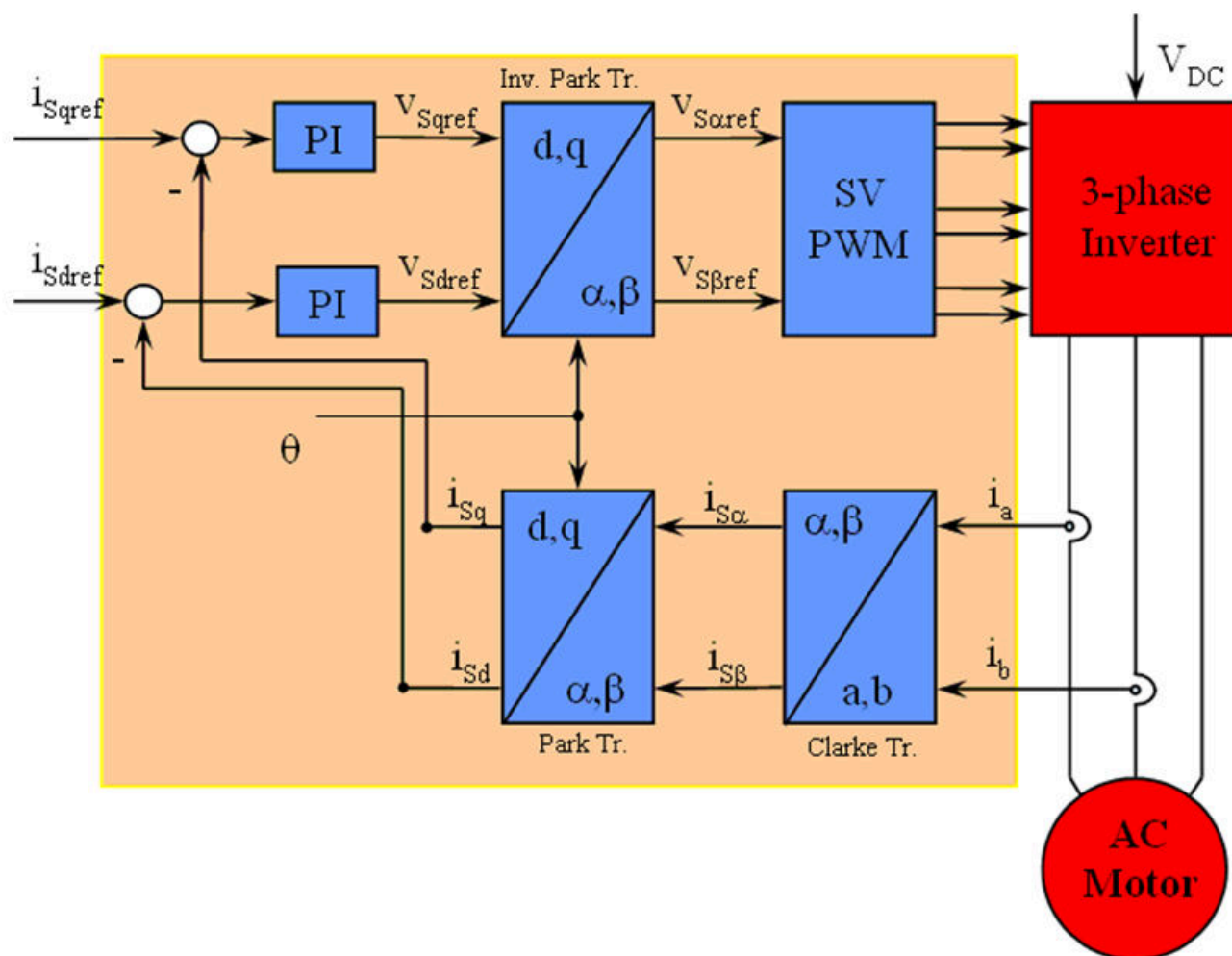


Figure 2-7. Basic Scheme of FOC for AC Motor

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

Rotor Flux Position

Knowledge of the rotor flux position is the core of the FOC. In fact if there is an error in this variable the rotor flux is not aligned with d -axis and i_{sd} and i_{sq} are incorrect flux and torque components of the stator current. [Figure 2-8](#) shows the (a, b, c) , (α, β) and (d, q) reference frames, and the correct position of the rotor flux, the stator current and stator voltage space vector that rotates with d,q reference at synchronous speed.

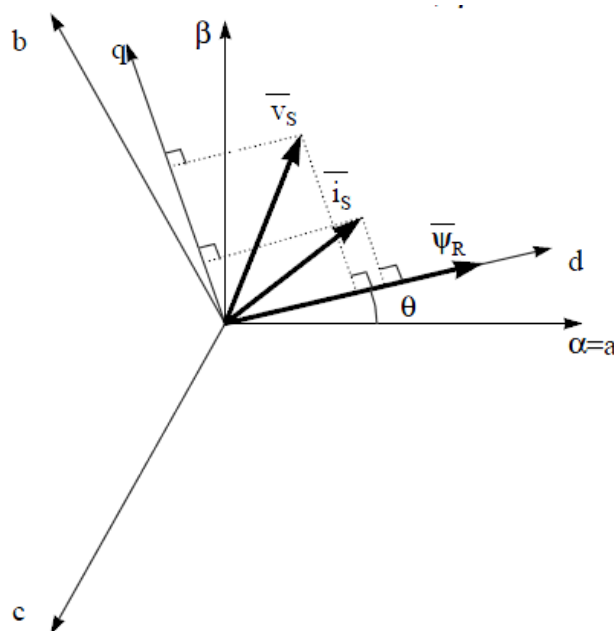


Figure 2-8. Current, Voltage and Rotor Flux Space Vectors in the (d, q) Rotating Reference Frame

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

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

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

Figure 2-9 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.

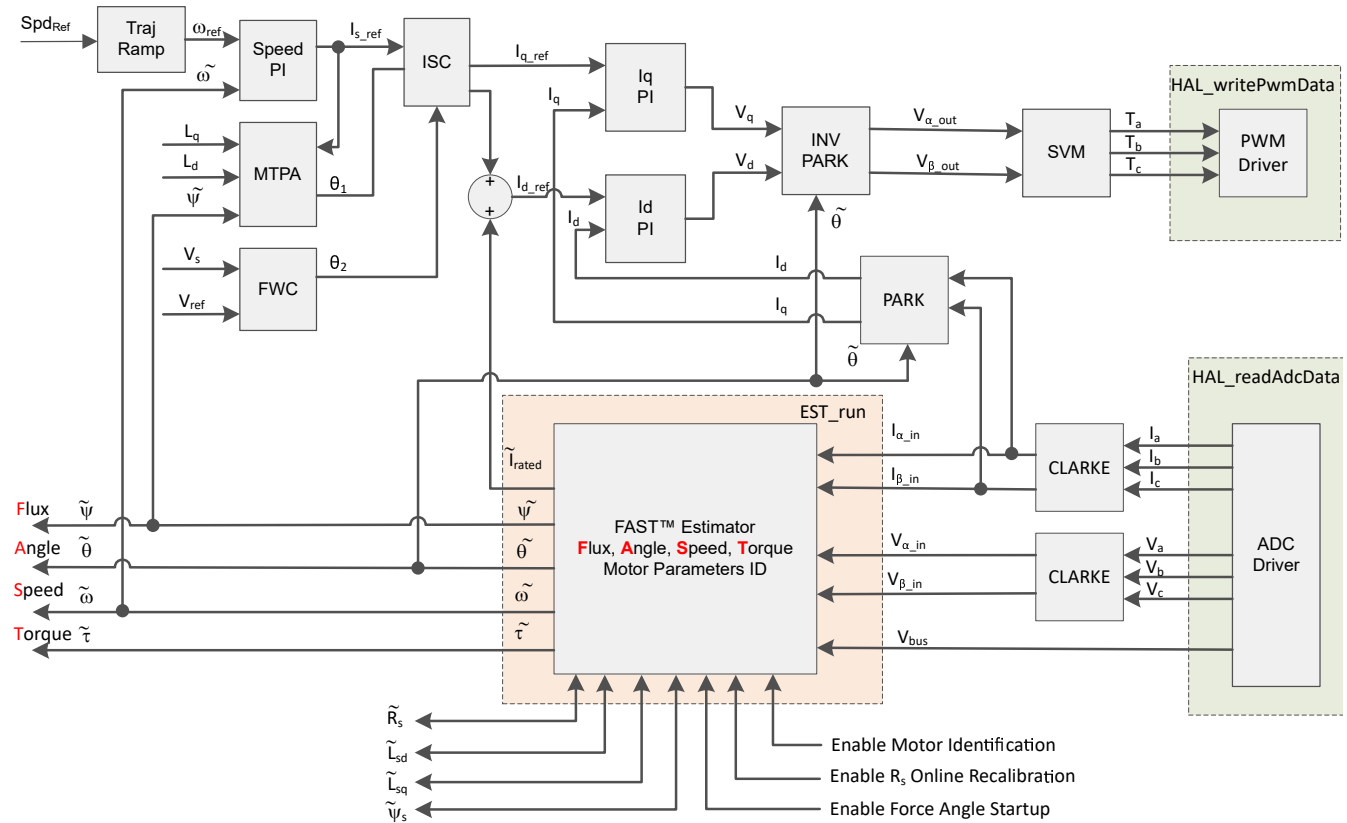


Figure 2-9. Sensorless FOC of Compressor PMSM using FAST with FWC and MTPA

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

Permanent magnet synchronous motor (PMSM) is widely used in home appliance applications due to its 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 their 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 its base speed, expanding its operating limits to reach speeds higher than rated speed and allow optimal 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 6 and Equation 7.

$$v_d = L_d \frac{di_d}{dt} + R_s i_d - p \omega_m L_q i_q \quad (6)$$

$$v_q = L_q \frac{di_q}{dt} + R_s i_q + p \omega_m L_d i_d + p \omega_m \psi_m \quad (7)$$

The dynamic equivalent circuit of an IPM synchronous motor is shown in Figure 2-10.

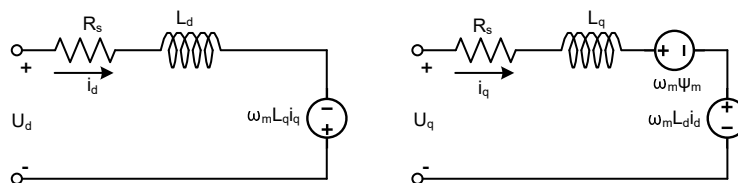


Figure 2-10. Equivalent Circuit of an IPM Synchronous Motor

The total electromagnetic torque generated by the IPMSM can be expressed as Equation 8 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 ψ_m , while the second term corresponds to the reluctance torque due to the differences in d-axis and q-axis inductance.

$$T_e = \frac{3}{2}p[\psi_m i_q + (L_d - L_q)i_d i_q] \quad (8)$$

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 9 and Equation 10.

$$I_a = \sqrt{i_d^2 + i_q^2} \leq I_{max} \quad (9)$$

$$V_a = \sqrt{v_d^2 + v_q^2} \leq V_{max} \quad (10)$$

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 11 if Space Vector Modulation (SVPWM) is adopted.

$$\sqrt{v_d^2 + v_q^2} \leq v_{max} = \frac{v_{dc}}{\sqrt{3}} \quad (11)$$

Usually the stator resistance R_s is negligible at high speed operation and the derivate of the currents is zero in steady state, thus Equation 12 is obtained as shown.

$$\sqrt{L_d^2 \left(i_d + \frac{\psi_{pm}}{L_d}\right)^2 + L_q^2 i_q^2} \leq \frac{V_{max}}{\omega_m} \quad (12)$$

The current limitation of Equation 11 produces a circle of radius I_{max} in the d-q plane, and the voltage limitation of Equation 12 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 2-11.

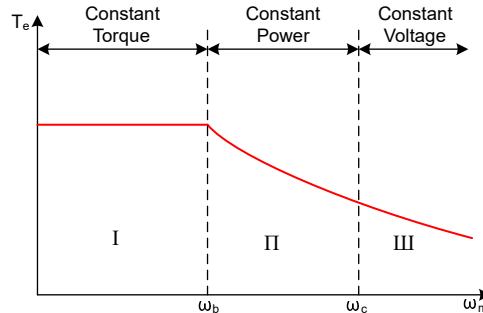


Figure 2-11. IPMSM Control Operation Regions

1. Constant Torque Region: MTPA can be implemented in this operation region to ensure 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 8, 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 a SPM motors only utilizes electromagnetic torque by setting the commanded i_d to zero for non-field weakening modes. But an IPMSM will utilize the reluctance torque of the motor, d-axis current should be controlled as well. 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.

$$I_s = \sqrt{i_d^2 + i_q^2} \quad (13)$$

$$I_d = I_s \cos \beta \quad (14)$$

$$I_q = I_s \sin \beta \quad (15)$$

Where β is the stator current angle in the synchronous (d-q) reference frame. Equation 8 can be expressed as Equation 16 where I_s substituted for i_d and i_q .

Equation 16 shows that motor torque depends on the angle of the stator current vector; as such

$$T_e = \frac{3}{2} p I_s \sin \beta [\psi_m + (L_d - L_q) I_s \cos \beta] \quad (16)$$

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, $\frac{dT_e}{d\beta}$ is zero as given in Equation 17.

$$\frac{dT_e}{d\beta} = \frac{3}{2} p [\psi_m I_s \cos \beta + (L_d - L_q) I_s^2 \cos 2\beta] = 0 \quad (17)$$

Following, the current angle of the MTPA control can be derived as in Equation 18.

$$\beta_{mtpa} = \cos^{-1} \frac{-\psi_m + \sqrt{\psi_m^2 + 8(L_d - L_q)^2 I_s^2}}{4(L_d - L_q) I_s} \quad (18)$$

Thus, the effective d-axis and q-axis reference currents can be expressed by Equation 19 and Equation 20 using the current angle of the MTPA control.

$$I_d = I_s \cos \beta_{mtpa} \quad (19)$$

$$I_q = I_s \sin \beta_{mtpa} \quad (20)$$

However, as shown in Equation 18, the angle of the MTPA control, β_{mtpa} is related to d-axis and q-axis inductance. This means that the variation of inductance will impede the ability to find the optimal MTPA point. To improve the efficiency of a motor drive, the d-axis and q-axis inductance should be estimated 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 ensures 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, assumes 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 their d-q currents respectively, as shown in Equation 21 and Equation 22.

$$L_d = f_1(i_d, i_q) = f_1(i_d) \quad (21)$$

$$L_q = f_2(i_q, i_d) = f_2(i_q) \quad (22)$$

To reduce the ISR calculation burden by simplifying Equation 18. The motor-parameter-based constant, β_{mtpa} is expressed instead as Equation 24, where K_{mtpa} is computed in the background loop using the updated L_d and L_q .

$$K_{mtpa} = \frac{\psi_m}{4*(L_q - L_d)} = 0.25 * \frac{\psi_m}{(L_q - L_d)} \quad (23)$$

$$\beta_{mtpa} = \cos^{-1} \left(K_{mtpa} / I_s - \sqrt{(K_{mtpa} / I_s)^2 + 0.5} \right) \quad (24)$$

A second intermediate variable, G_{mtpa} described in Equation 25, is defined to further simplify the calculation. Using G_{mtpa} , the angle of the MTPA control, β_{mtpa} can be calculated as Equation 26. These two calculations are performed in the ISR to achieve a real current angle β_{mtpa} .

$$G_{mtpa} = K_{mtpa} / I_s \quad (25)$$

$$\beta_{mtpa} = \cos^{-1} \left(G_{mtpa} - \sqrt{G_{mtpa}^2 + 0.5} \right) \quad (26)$$

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 9 and Equation 10. 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 12, the IPMSM has two factors: one is a permanent magnet value and the other is made by inductance and current of flux.

Figure 2-12 shows the typical control structure is used to implement field weakening. β_{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 its limit, the input of the PI controller of FW is always positive and therefore the output is always saturated at 0.

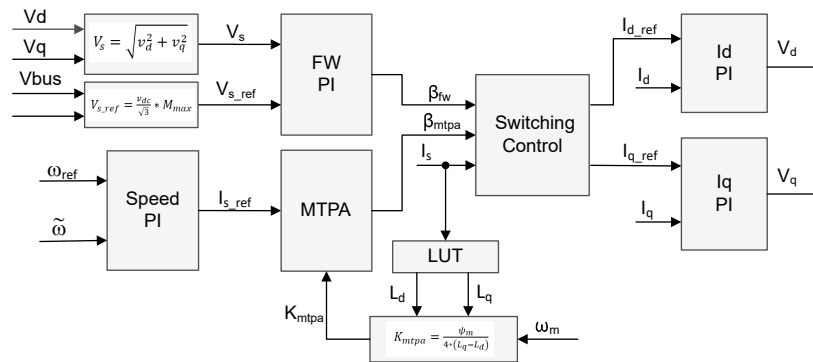


Figure 2-12. Block Diagram of Field-Weakening and Maximum Torque per Ampere Control

Sensorless FOC of Compressor PMSM using FAST with FWC and MTPA shows the implementation of FAST-based FOC block diagram. The block diagrams provide an overview of the FOC system's functions and variables. 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 β_{mtpa} and β_{fw} respectively based on input parameters as show in Figure 2-13.

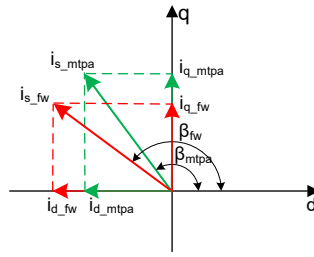


Figure 2-13. Current Phasor Diagram of an IPMSM During FW and MTPA

The switching control module is used to decide which angle should be applied, and then calculate the reference i_d and i_q as shown in Equation 14 and Equation 15. The current angle is chosen as following Equation 27 and Equation 28.

$$\beta = \beta_{fw} \text{ if } \beta_{fw} > \beta_{mtpa} \quad (27)$$

$$\beta = \beta_{mtpa} \text{ if } \beta_{fw} < \beta_{mtpa} \quad (28)$$

Figure 2-14 is the flowchart that shows the steps required to run InstaSPIN-FOC with FW and MPTA in the main loop and interrupt.

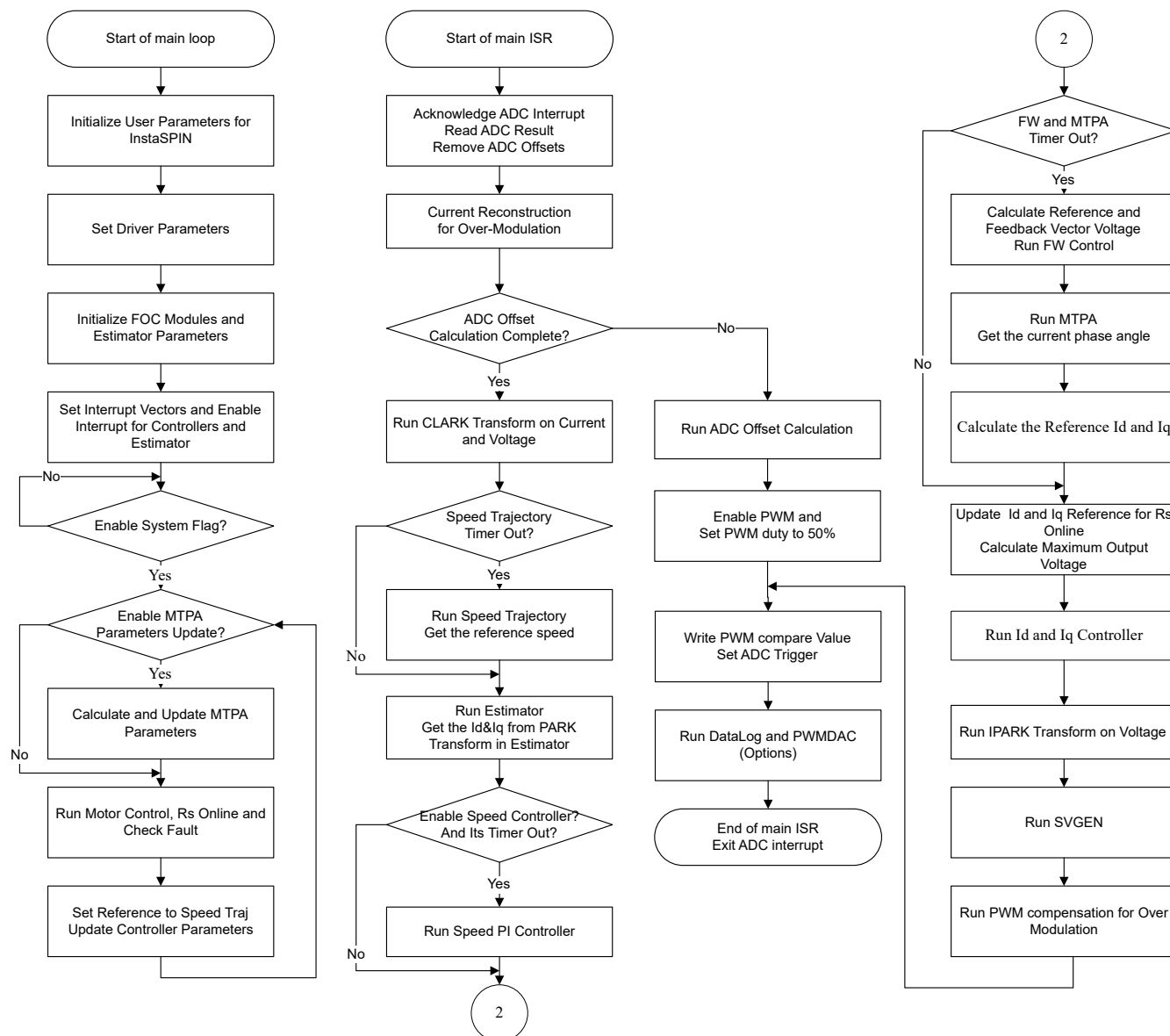


Figure 2-14. Flowchart for an InstaSPIN-FOC Project with FW and MTPA

2.4.4 Compressor Drive with Automatic Vibration Compensation

Vibration and noise can become a problem in air conditioning compressors applications since they cause an undesirable end user experience, as well as mechanical failures due to stress. The compressor applications contain pulsating loads, which is dependent on the mechanical angle as shown in [Figure 2-15](#), can cause motor vibration and audible noise. There are several causes of vibration and noise. The main cause is the vibration produced by the load characteristic. A new dynamic and adaptive compensation method will also be covered, showing the details on how it operates and the minimal tuning required.

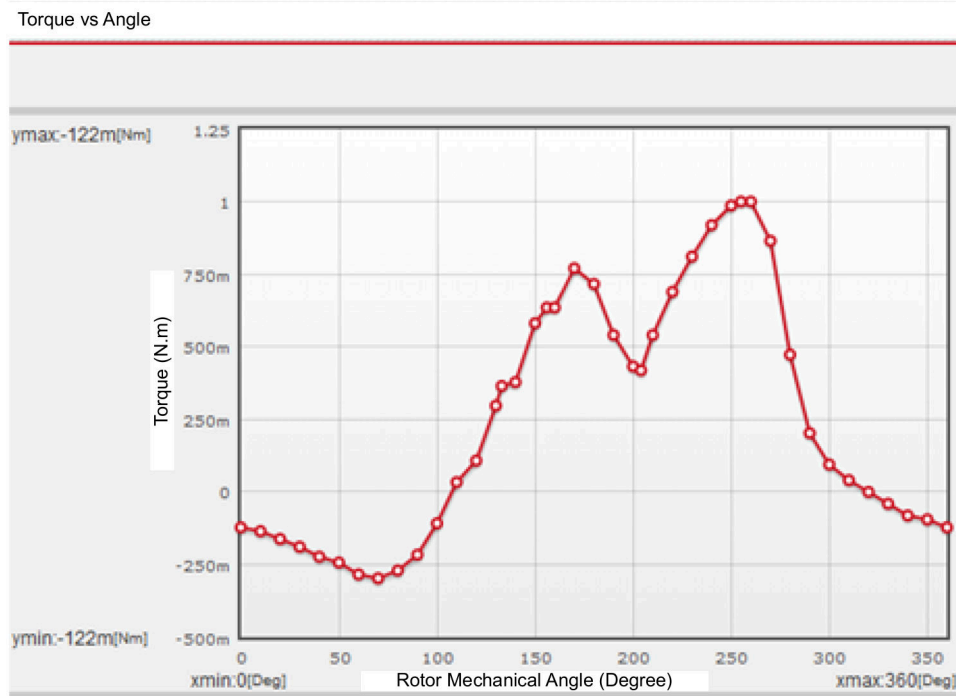


Figure 2-15. Waveform of Load Torque

The vibration compensation algorithm learns the load profile as the motor runs, and as the speed controller tries to correct for these load changes, and once the load is learned, the algorithm is used to extract load information relative to the mechanical angle, and uses that information as a feed forward in the speed controller. As shown in Figure 2-16, a new block, called *Dynamic Vibration Compensation* is added to the FOC system, is used to learn the torque load, to allow adding a feedforward term to the speed controller, in the form of a summing point to the output generated by the speed controller.

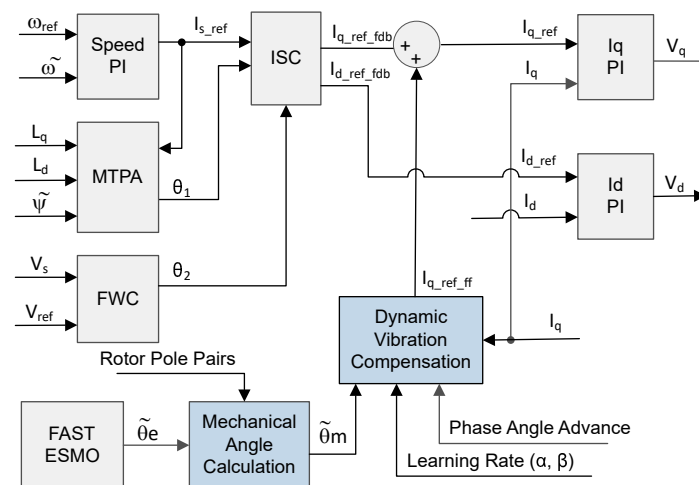


Figure 2-16. Block Diagram of FOC based Motor Drive with Vibration Compensation

This algorithm requires four main blocks to be able to work:

1. A speed controller with feed forward input
2. A table to hold a learning curve
3. A way of extracting a specific member of that table based on an index
4. Calculation of an index to update the learning curve and an advanced index to extract a value from that table

The algorithm is able to dynamically learn a load profile based on two inputs:

1. Electrical angle information. Starting from the lower left as shown in [Figure 2-16](#), the first input that is required by the vibration compensation module is the mechanical angle. This is calculated based on the electrical angle and the number of pole pairs. It is not required for the mechanical angle to be synchronized with the electrical angle. For example, the zero of the mechanical angle, physically, does not need to be the zero of the electrical angle. This is because the vibration compensation module will learn the load according to the mechanical angle provided, independently of what the mechanical angle is compared to the physical position of the shaft.
2. A measured current value, in the case of FOC, I_q , which is responsible for the torque of the motor.

Then the vibration compensation module is implemented. The module needs four parameters in this implementation.

1. Phase Advance. which is how the learned load will be accessed with respect to the mechanical angle. If zero phase advance, the loaded value on I_{qRef_ff} will correspond to the provided mechanical angle. If phase advance is 10, the loaded value on I_{qRef_ff} will correspond to the mechanical angle plus 10 mechanical degrees (in a scale from 0 to 360 degrees).
2. Learning Rate. This parameter is a value from 0 to 1, which is essentially how fast (less noise immune) or how slow (more noise immune) the load learning happens
3. Points. This is the number of points of the compensation table to be used for the learning curve.
4. Pole Pairs. This is the number of poles in the motor.

Then the summation point in between the speed controller and the I_q controller. This is where the output of the vibration compensation module is used, to help the speed controller with this term. This technique is also known as feedforward, since the load is known in advance, according to the mechanical angle provided.

Once the load has been learned by the vibration compensation module, the speed controller will correct for transients in load change, that don't relate to the natural mechanical load vs. mechanical angle, which is already compensated by the vibration compensation module. To illustrate how the vibration compensation module helps, let's take a look at the following plot, where we show the output of the speed controller with vibration compensation disabled. It is obvious that the speed controller gains need to be high to track the load changes as the motor spins every cycle.

Tuning the learning rate

The learning rate can be adjusted depending on two factors. One is how quickly the user wants to learn the curve, and the second consideration is how noisy is the input to the learning curve. The second consideration is important, because the noise is not only coming from the current sensing method itself, but minor mechanical perturbations in the system, that are not periodic, and that we would like to filter out and not have it in our compensation table. If the learning rate is too low, the learning time could be too long for a specific application, so a trade-off needs to be made.

Tuning the phase angle

In a discrete system there are several delays when it comes to outputting a voltage to the motor, and also delays related to sensing currents. For example, when implementing an FOC system in a processor, usually the output voltage goes through a Pulse Width Modulator (PWM) which has delays. This phase advance parameter allows fine tuning of that delay, by providing a number in units of table positions, from the learned table, so that the appropriate output can be applied to the speed controller's feed forward input. The easiest way to tune this parameter is by looking at the speed variation after applying dynamic compensation, and tuning the value so that a minimum amount of speed variation is achieved.

2.4.5 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 the eCompressor motor control with FAST observer.

2.4.5.1 Motor Current Feedback

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

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

The current version of software only includes support for three-shunt current sensing method. The support for single-shunt current sensing method is planned in a future release.

2.4.5.1.1 Current Sensing with Three-Shunt

The current through the motor is sampled by the microcontroller as part of the motor control algorithm during every PWM cycle. To measure bidirectional currents of the motor phase, that is, positive and negative currents, the circuits below require a reference voltage of 1.65-V. This offset reference voltage is created by a voltage follower as shown in Figure 2-17. This 1.65-V reference voltage is used to both three-shunt and single-shunt AC voltage feedback sensing circuit.

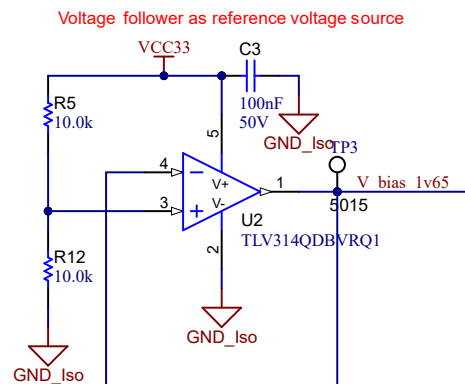


Figure 2-17. 1.65-V Reference from 3.3-V Input Circuit

Figure 2-18 shows how the motor current is represented as a voltage signal, with filtering, amplification, and offset to the center of the ADC input range. 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 29.

$$V_{OUT} = V_{OFFSET} + (I_{IN} \times R_{SHUNT} \times G_i) \quad (29)$$

Where $R_{shunt}=0.005(\text{OHM})$ and $V_{offset}=1.65(\text{V})$

The calculated resistance values lead to the sensing circuit shown in Figure 2-18, the G_i is given by Equation 30.

$$G_i = \frac{R_{fb}}{R_{in}} = \frac{R_{29}}{R_{31}} = \frac{10K}{1K} = 10 \quad (30)$$

The maximum peak to peak current measurable by the microcontroller is given by Equation 31.

$$I_{scale_max} = \frac{V_{ADC_max}}{R_{shunt} \times G_i} = \frac{3.3}{0.005 \times 10} = 66A \quad (31)$$

which is the peak to peak value of $\pm 33A$. The following code snippet shows how this is defined for compressor motor in user_mtr1.h file:

```

//! \brief Defines the maximum current at the AD converter
//!
#define USER_M1_ADC_FULL_SCALE_CURRENT_A      (66.00f)

```

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 non-inverting pin of the operational amplifier.

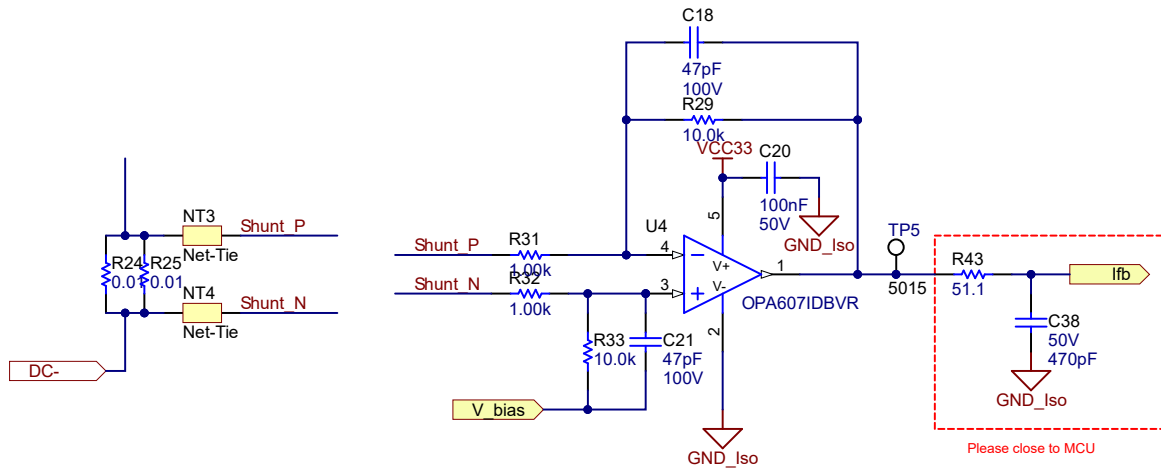


Figure 2-18. Motor Current Sensing Circuit with Three-Shunt

2.4.5.1.2 Current Sensing with Single-Shunt

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

On this reference board, to implement single shunt current sensing technique by having a separate shunt resistor and amplifier circuit as shown in [Figure 2-19](#).

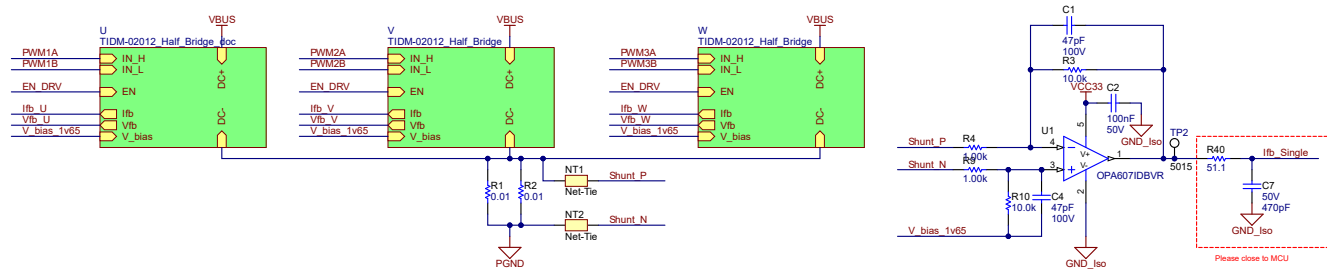


Figure 2-19. Motor Current Sensing Circuit with Single-Shunt

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

2.4.5.2 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. This software value (USER_ADC_FULL_SCALE_VOLTAGE_V) depends on the circuit that senses the voltage feedback from the motor phases. [Figure 2-20](#) 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 both compressor motor and DC bus.

The maximum phase voltage feedback measurable by the microcontroller in this reference design can be calculated as given in [Equation 32](#), considering the maximum voltage for the ADC input is 3.3 V.

$$V_{FS} = V_{ADC_FS} \times G_V = 3.3V \times 293.955 = 970.05V \quad (32)$$

Where G_V is attenuation factor can be calculated from

$$G_V = \frac{(R26 + R27 + R28 + R30)}{R30} = \frac{(499K + 499K + 499K + 5.11K)}{5.11K} = 293.955 \quad (33)$$

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 (970.05f)
```

The voltage filter pole is needed by the FAST estimator to allow an accurate detection of the voltage feedback. The filter should be 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 20 kHz. The hardware filter should only be changed when ultra-high-speed motors are run, which generate phase-voltage frequencies in the order of a few kHz.

The filter pole setting can be calculated as the following Equation 34 in this reference design:

$$f_{filter_pole} = \frac{1}{(2 \times \pi \times R_{Parallel} \times C)} = 664.94 \text{ Hz} \quad (34)$$

where,

$$C = 47\text{nF}$$

$$R_{Parallel} = \left(\frac{(499K + 499K + 499K) \times 5.11K}{(499K + 499K + 499K) + 5.11K} \right) = 5.0926k\Omega$$

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 (664.94f)
```

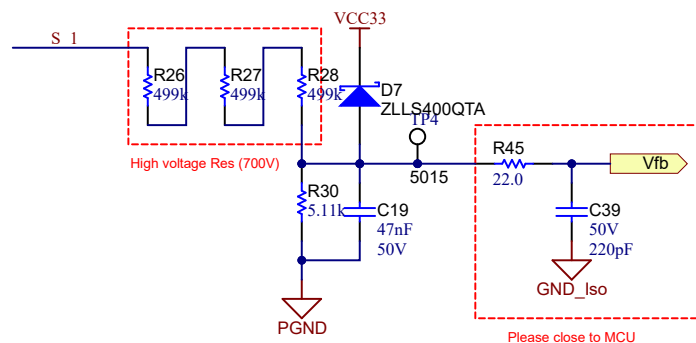


Figure 2-20. Motor Voltage Sensing Circuit

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Hardware Requirements

This section details the hardware and explains the different sections on the board and how to set them up for the experiments as outlined in this design guide.

3.1.1 Hardware Board Overview

Figure 3-1 shows an overview of a typical eCompressor drive system.

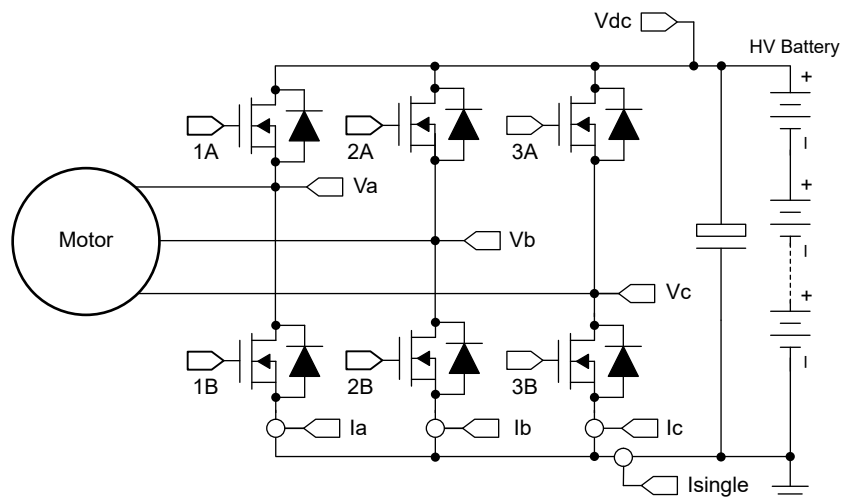


Figure 3-1. Hardware Board Block Diagram of TIDM-02012

The motor control board has functional groups that enable a complete motor drive system. The following is a list of the blocks on the board and their functions, [Figure 3-2](#) shows the top view of the board and different blocks of the TIDM-02012 PCB.

- DC bus input
 - DC bus input connector
 - 10 uF film capacitor rated for 1.3 kV
- 3-phase inverter for eCompressor motor
 - Up to 5 kW 3-phase inverter supports PMSM or IPM
 - 15 kHz switching frequency
 - 3-shunt / single shunt for current sensing
 - Amplify and input filters for the analog signals
- Control
 - TMS320F28039C MCU controlCARD
 - 100 MHz 32-bit CPU with FPU and TMU
- Auxiliary power supply
 - On-board power supply to generate isolated +6 V, and +16 V. The +6V is then driven to +5V and +3.3V via LDO to power the controlCARD and control circuits.

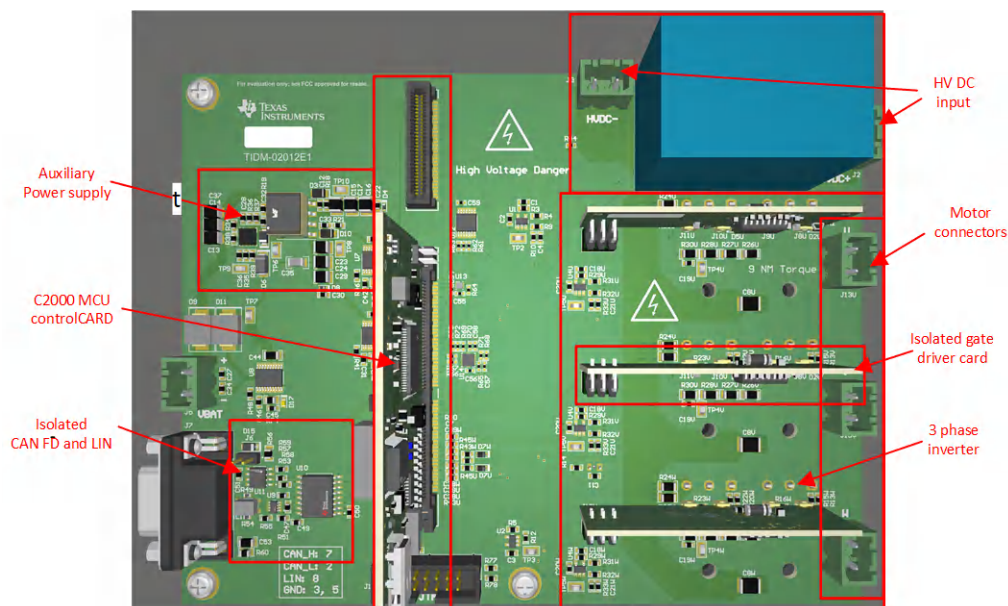


Figure 3-2. The Layout of eCompressor Reference Design Board

Note

Although the TMS320F280039C controlCARD only takes one 120-pin HSEC connector, we reserved the 180-pin HSEC connector for compatibility with future devices.

TI recommends taking the following precautions when using the board:

- Do not touch any part of the board or components connected to the board when the board is energized.
- Use the AC Mains/wall power supply to power the kit. TI recommends an isolation AC source.
- Do not touch any part of the board, the kit or its 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

For the user to test the reference design software

- For input, the power supply source support up to 800 V if using the DC power supply. Set the input current limit of input DC power supply to 25 A, but start with a lower current limit during initial board bring-up
- For output, a 3-phase PMSM with dynamo meters

3.1.3 Test Equipment Required for Board Validation

- DC power supply
- Digital oscilloscope
- Multimeters
- 3-phase PM synchronous motors
- Dynamo meters
- Three-phase power analyzer

3.2 Test Setup**3.2.1 Hardware Setup**

Figure 3-2 shows the position of these blocks and the connectors on the board. Setting up the hardware following the steps below.

1. Connect a USB cable to TMS320F280039C controlCARD for JTAG connection.
2. Connect the compressor motor wires to the terminals J13U, J13V and J13W.
3. Apply 12V auxiliary DC power supply to terminal J5.
4. Apply a DC bus power supply to terminal J2 and J3. The maximum output of the DC power supply is 800 VDC.
5. Connect multimeter, oscilloscope probes, and other measurement equipment to probe or analyze various signals and parameters as desired. Only use appropriately rated equipment and follow proper isolation and safety practices.

Note

Add ferrite beads on JTAG signals and USB cable, if the external emulator has connectivity issues while testing. And make the connection lines as short as possible.

WARNING

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

3.2.2 Software Setup

Download and install [Code Composer Studio \(CCS\)](#) which will be needed to build and run the project. CCS version 12.0.0 or newer is required for this project. Find more details about CCS installation and use in the [CCS User's Guide](#).

The software for this reference design is provided as part of the [C2000Ware MotorControl SDK v4.01.00.00](#) or newer. Once you have downloaded and installed it, you can browse the folder for this design by going to <SDK install location>\solutions\tidm_02012_ecompressor\.

3.2.2.1 Code Composer Studio Project

To import the reference project in CCS, click **Project** → **Import CCS Projects**, and browse to <SDK install location>\solutions\tidm_02012_ecompressor\<device>\ccs and click **Select Folder**. Select the project called *tidm_02012_ecompressor_<device>* and click **Finish**. The project should now be visible in the **Project Explorer** pane in CCS.

The folder *src_foc* includes the typical FOC modules, including transforms, PID functions, and estimators. The folder *src_lib* includes the InstaSPIN-FOC library and related header files. These modules are independent of the specific device and board and are used across several other solutions in the SDK.

The folder *src_control* includes motor drive control files that call motor control core algorithm functions within the interrupt service routines and background tasks. The folder *src_sys* includes some files reserved for other system features such as drivers for CAN communication. You can add your own code for system control, communication, and so forth. These modules are specific to this reference design project but are independent of the device and board.

Board-specific, motor-specific and device-specific files are in the folder *src_board*. These files consist of device-specific drivers to run the solution. If you want to migrate the project to your own board or other devices, you only need to make changes to the files *hal.c*, *hal.syscfg*, *hal.h*, and *user_mtr1.h* based on the pin assignments and features of your device or board.

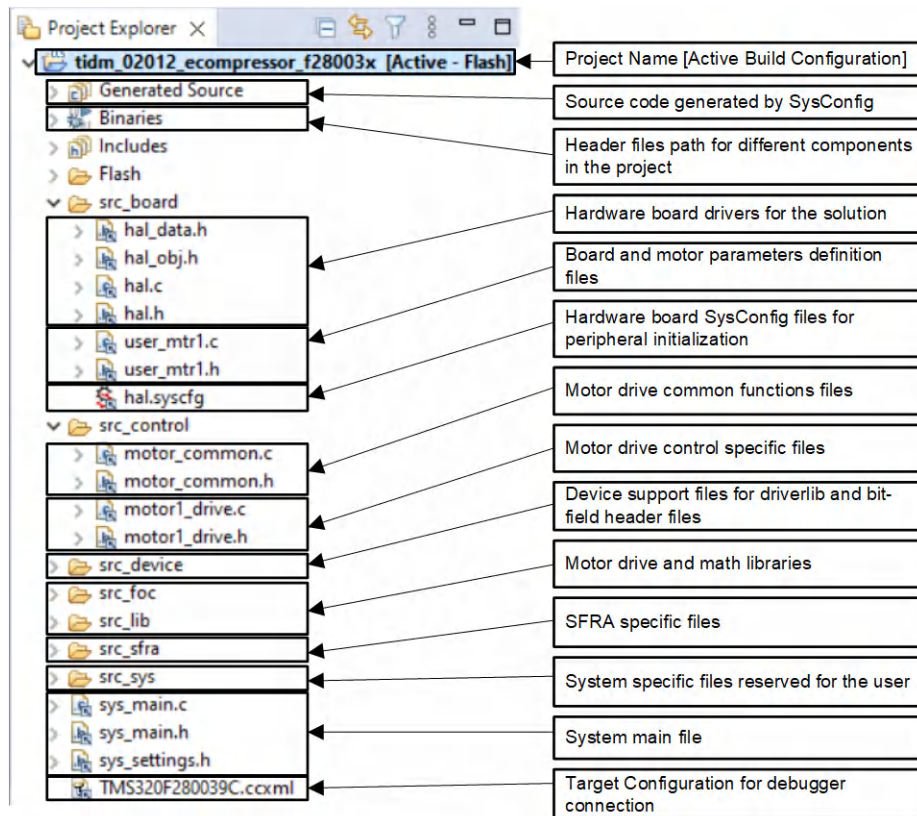


Figure 3-3. Project Explorer View of the Reference Project

There are several optional features in the software that can be enabled and disabled using predefined symbols in the project properties. They are as follows:

- **MOTOR1_OVM** to enable overmodulation
- **MOTOR1_FWC** to enable field-weakening control (FWC)
- **MOTOR1_MTPA** to enable maximum torque per ampere (MTPA) mode
- **MOTOR1_VIBCOMPA** or **MOTOR1_VIBCOMPT** to enable vibration compensation

To view and edit the predefined symbols, right-click on your project and select **Properties**. Then go to the **Predefined Symbols** section of the **C2000 Compiler** options as shown in Figure 3-4. By default the features listed above are disabled by the "_N" appended to the symbol name. Edit the symbol to remove the "_N" to enable the feature.

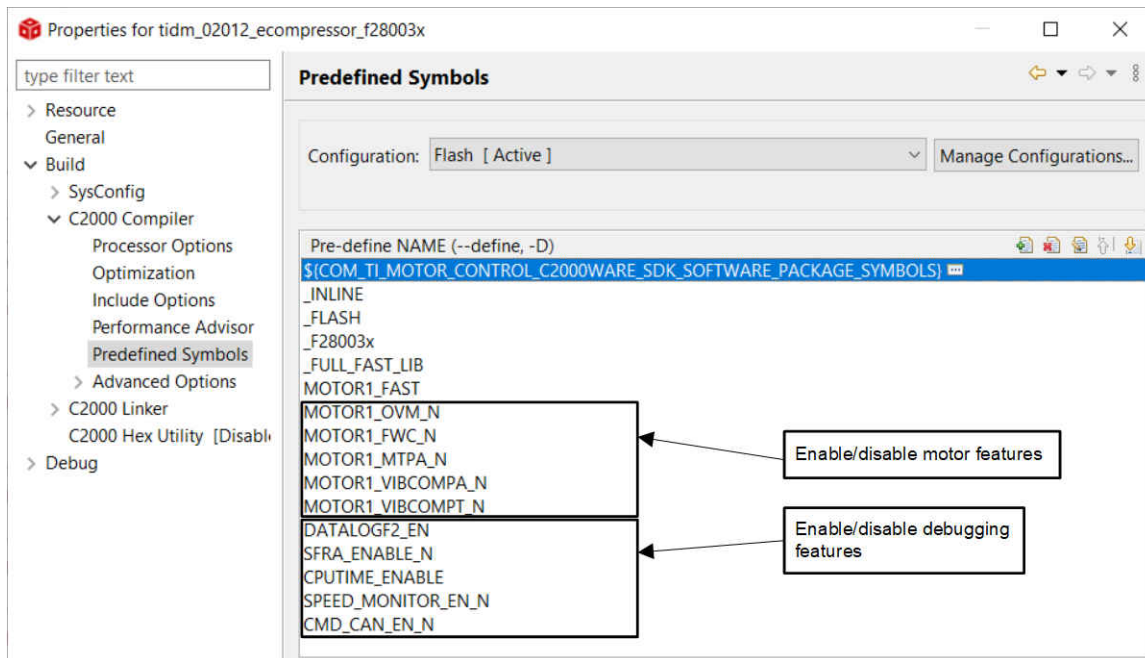


Figure 3-4. Selecting Predefined Symbols in the Project Properties

3.2.2.2 Software Structure

The general structure of the project is shown in Figure 3-5. The device peripheral configuration is based on C2000Ware Driverlib and is partially generated using SysConfig. You only need to change the *hal.c*, *hal.syscfg*, and *hal.h* files and the parameters in *user_mtr1.h* if you want to migrate the reference design software to your own board or a different device.

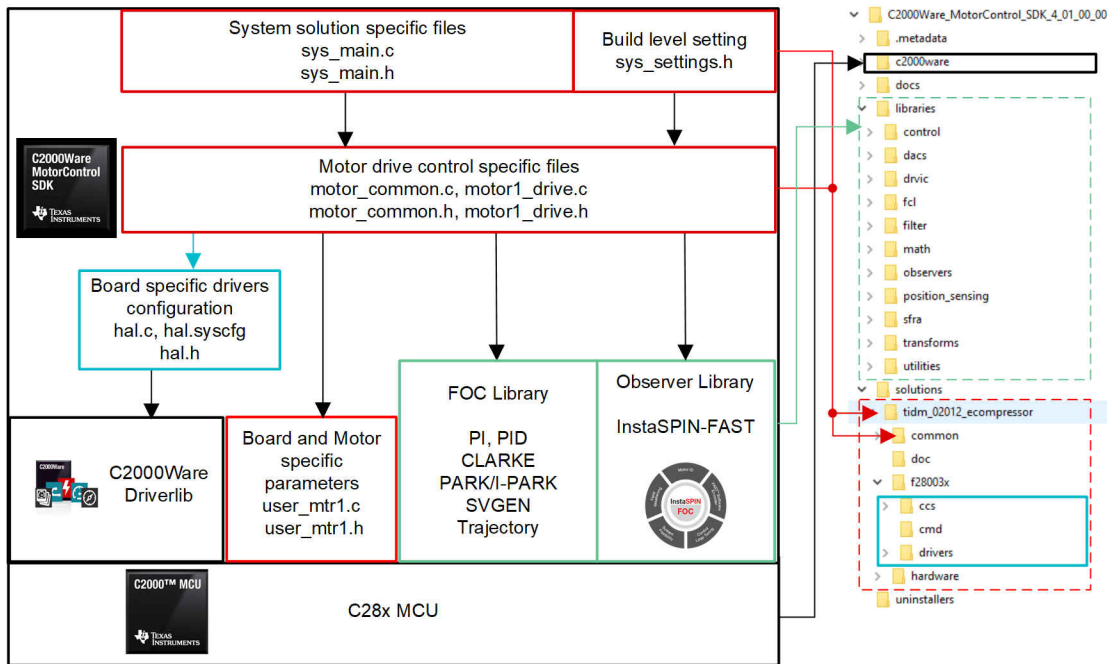


Figure 3-5. Project Structure Overview

Figure 3-6 shows the project software flow diagram of the firmware that includes one ISR for real time motor control, a main loop for motor control parameters update in background loop. The ISR will be triggered by ADC End of Conversion (EOC).

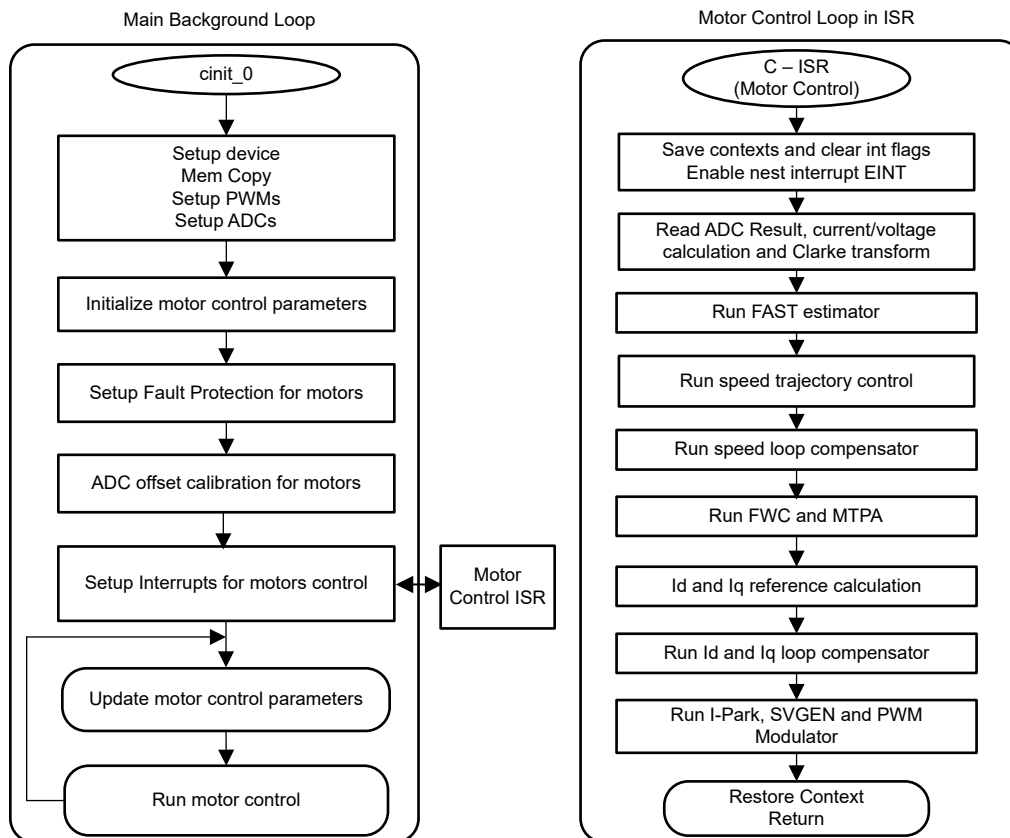


Figure 3-6. Flowcharts of Background Software and Motor Control ISR

To simplify the system bring up and design, the software is organized into four incremental builds described in [Table 3-1](#), which makes learning and getting familiar with the board and software easier. This approach is also helpful for debugging and testing boards. To select a particular build option, you will edit the *DMC_BUILDLEVEL* option in *sys_settings.h* file and rebuild the project. This process is described in the detail in the next section.

Table 3-1. Incremental Build Options

DMC_BUILDLEVEL Value	Description
DMC_LEVEL_1	50% duty, offset calibration and verify phase shift
DMC_LEVEL_2	Open loop control to check sensing signals
DMC_LEVEL_3	Closed current loop to check the hardware settings
DMC_LEVEL_4	Parameters identification and run with InstaSPIN-FOC

3.3 Test Procedure

The system software is gradually built up until the final system can be confidently operated. To select a particular build option, select the corresponding *DMC_BUILDLEVEL* option in *sys_settings.h*. Once the build option is selected, compile the project by right-clicking on the project name and clicking **Rebuild Project**.

```
//=====
// Incremental Build options for System check-out
//=====
#define DMC_LEVEL_1    1 // 50% duty, offset calibration and verify phase shift
#define DMC_LEVEL_2    2 // Open loop control to check sensing signals
#define DMC_LEVEL_3    3 // Closed current loop to check the hardware settings
#define DMC_LEVEL_4    4 // Parameters identification and run with InstaSPIN-FOC

#define DMC_BUILDLEVEL DMC_LEVEL_4
```

3.3.1 Level 1 Incremental Build

Objectives of this build level:

- Use the HAL to initialize the peripherals of the MCU.
- Verify the PWM and ADC driver modules.
- Verify the ADC Offset validation.
- Become familiar with the operation of CCS.

In this build level, the board is executed in open loop mode with a fixed duty cycle. The duty cycles are set to 50%. This build level verifies the sensing of feedback values from the power stage and also operation of the PWM gate driver. Additionally, calibration of input and output voltage sensing can be performed in this build level. During this process the motor must remain disconnected. The software block diagram of this build level is shown in [Figure 3-7](#).

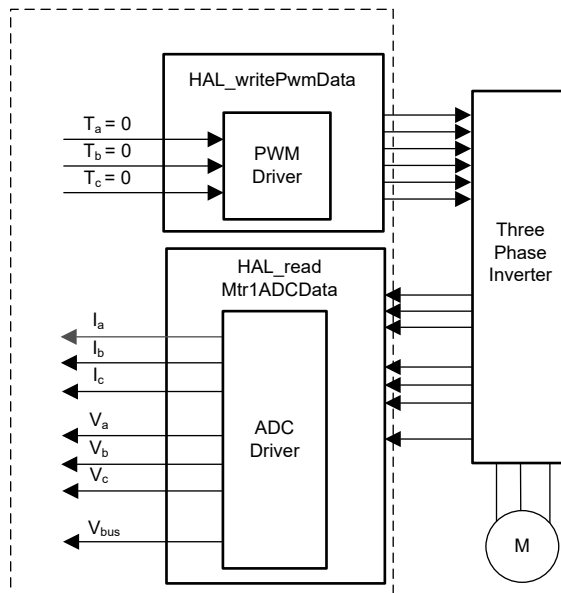


Figure 3-7. Build Level 1 Software Block Diagram

3.3.1.1 Project Setup

Import the project into CCS and select the appropriate build configuration. Open the *sys_settings.h* file and set the `#define` for *DMC_BUILDLEVEL* to *DMC_LEVEL_1*. This configures the project to run the first incremental build. Right-click on the project in the **Project Explorer** and select **Rebuild Project**. Confirm that the **Console** pane shows that the project built without any errors.

On successful completion of the build, with the *tidm_02012_ecompressor* project selected, go to **Run** → **Debug** or click the **Debug** button on the tool bar. By default the project will launch a debug session using the *TMS320F280039C.ccxml* file in the project. *TMS320F280039C.ccxml* is configured to use the Texas Instruments XDS110 USB Debug Probe on board the TMDSCNCD280039C controlCARD.

After clicking **Debug**, CCS will automatically connect to the target, load the output file into the device, and change to the **CCS Debug** perspective. The program should be halted at the start of *main()*.

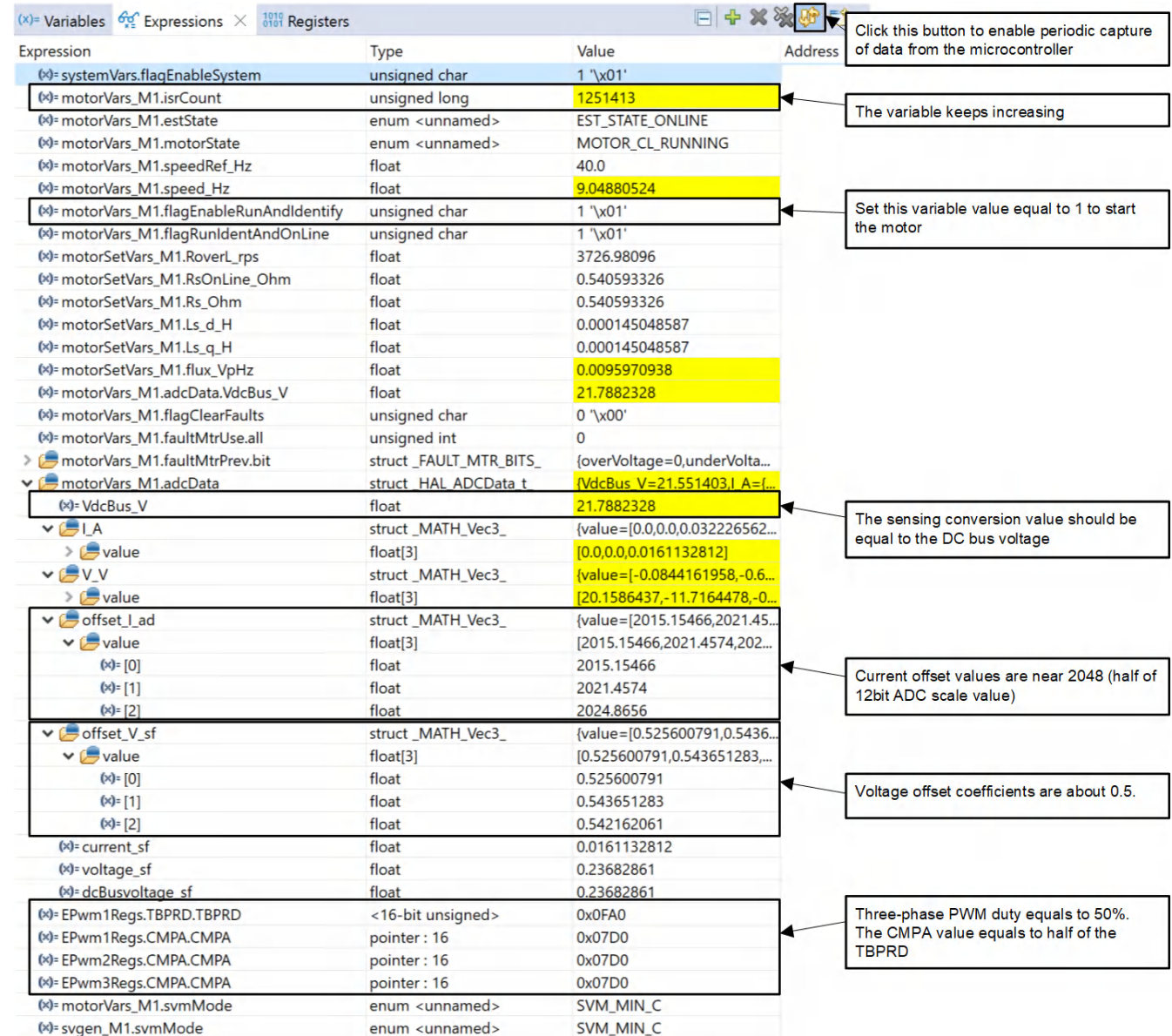
If the **Expressions** pane is not already open, click **View** → **Expressions** in the CCS menu bar. You can either add variables manually or you can import a recommended list of variables associated with this build level by right-clicking within the **Expressions** window, selecting **Import...**, and finding the file `<SDK install location>\solutions\tidm_02012_ecompressor\common\debug\tidm_02012_level1.txt`. Click **OK** and the window will be populated with variables as shown in Figure 3-8.

Click the **Continuous Refresh** button in the Expressions window tool bar to tell CCS to update the data continuously at a rate defined in the CCS debug preferences.

3.3.1.2 Running the Application

Run the code by going to **Run** → **Resume** or clicking the **Resume** button in the tool bar. The project should now run and you should see the variables in the Expressions window updating and the D2 LED blinking on the controlCARD. Check the following to confirm the application and hardware set up are working:

- Set the variable *motorVars_M1.flagEnableRunAndIdentify* to 1 after you see *systemVars.flagEnableSystem* was automatically set to 1. Your variables should appear to similar to what is shown in Figure 3-8.
- The variable *motorVars_M1.flagRunIdentAndOnLine* should be set to 1 automatically if no faults are detected. The *motorVars_M1.isrCount* should be increasing continuously.
- Check the calibration offsets of the motor inverter board. The offset values of the motor phase current sensing values should be equal to approximately half of the scale current of ADC.
- Probe the PWM output for motor drive control with an oscilloscope. The three PWMs' duty are set to 50% in this build level. The PWM switching frequency should be as defined for *USER_M1_PWM_FREQ_kHz* in *user_mtr1.h*.



Expression	Type	Value	Address
(x)= systemVars.flagEnableSystem	unsigned char	1 '\x01'	
(x)= motorVars_M1.isrCount	unsigned long	1251413	
(x)= motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	
(x)= motorVars_M1.motorState	enum <unnamed>	MOTOR_CL_RUNNING	
(x)= motorVars_M1.speedRef_Hz	float	40.0	
(x)= motorVars_M1.speed_Hz	float	9.04880524	
(x)= motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\x01'	
(x)= motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\x01'	
(x)= motorSetVars_M1.RoverL_rps	float	3726.98096	
(x)= motorSetVars_M1.RsOnLine_Ohm	float	0.540593326	
(x)= motorSetVars_M1.Rs_Ohm	float	0.540593326	
(x)= motorSetVars_M1.Ls_d_H	float	0.000145048587	
(x)= motorSetVars_M1.Ls_q_H	float	0.000145048587	
(x)= motorSetVars_M1.flux_VpHz	float	0.0095970938	
(x)= motorVars_M1.adcData.VdcBus_V	float	21.7882328	
(x)= motorVars_M1.flagClearFaults	unsigned char	0 '\x00'	
(x)= motorVars_M1.faultMtrUse.all	unsigned int	0	
> motorVars_M1.faultMtrPrev.bit	struct _FAULT_MTR_BITS_	{overVoltage=0,underVolta...	
▼ motorVars_M1.adcData	struct HAL_ADCData_t	{VdcBus V=21.551403, A={...	
(x)= VdcBus_V	float	21.7882328	
▼ I_A	struct _MATH_Vec3_	{value=[0.0,0.0,0.032226562...	
> value	float[3]	[0.0,0.0,0.0161132812]	
▼ V_V	struct _MATH_Vec3_	{value=[-0.0844161958,-0.6...	
> value	float[3]	[20.1586437,-11.7164478,-0...	
▼ offset_I_ad	struct _MATH_Vec3_	{value=[2015.15466,2021.45...	
▼ value	float[3]	[2015.15466,2021.4574,202...	
(x)= [0]	float	2015.15466	
(x)= [1]	float	2021.4574	
(x)= [2]	float	2024.8656	
▼ offset_V_sf	struct _MATH_Vec3_	{value=[0.525600791,0.5436...	
▼ value	float[3]	[0.525600791,0.543651283,...	
(x)= [0]	float	0.525600791	
(x)= [1]	float	0.543651283	
(x)= [2]	float	0.542162061	
(x)= current_sf	float	0.0161132812	
(x)= voltage_sf	float	0.23682861	
(x)= dcBusvoltage_sf	float	0.23682861	
(x)= EPwm1Regs.TBPRD.TBPRD	<16-bit unsigned>	0x0FA0	
(x)= EPwm1Regs.CMPA.CMPA	pointer : 16	0x07D0	
(x)= EPwm2Regs.CMPA.CMPA	pointer : 16	0x07D0	
(x)= EPwm3Regs.CMPA.CMPA	pointer : 16	0x07D0	
(x)= motorVars_M1.svmMode	enum <unnamed>	SVM_MIN_C	
(x)= svgen_M1.svmMode	enum <unnamed>	SVM_MIN_C	

Figure 3-8. Build Level 1 Variables in the Expressions View

You can halt the CPU by first clicking the **Suspend** button on the toolbar or by selecting **Target** → **Suspend**. To run the application from the start again, reset the controller by clicking on the **CPU Reset** tool bar button or clicking **Run** → **Reset** → **CPU Reset** and then clicking on the **Restart** button or **Run** → **Restart**. You can close the CCS debug session by clicking the **Terminate** button or by clicking **Run** → **Terminate**. This will halt the program and disconnect CCS from the controller.

Note that it is not necessary to terminate the debug session each time you change the code. Instead you can go to **Run** → **Load** → **Load Program...** (or **Reload Program...** if it is the same file). CCS will also automatically prompt you to ask if you want to reload your executable if it detects you have rebuilt it.

3.3.2 Level 2 Incremental Build

Objectives of this build level:

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

Figure 3-9.

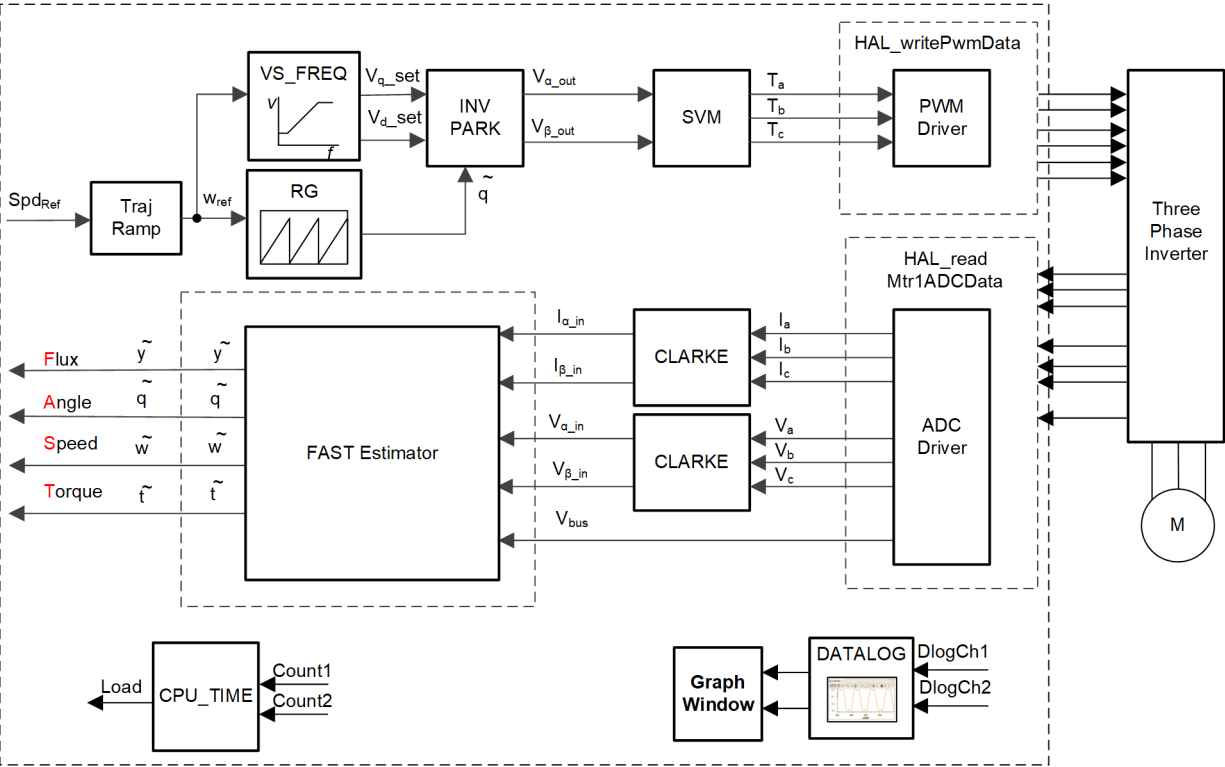


Figure 3-9. Build Level 2 Software Block Diagram

3.3.2.1 Project Setup

Follow the steps described in [Section 3.3.1.1](#) to change `DMC_BUILDLEVEL` to `DMC_LEVEL_2` and rebuild and load the project. You should also change the contents of the **Expressions** view to the variables listed in `tidm_02012_level2.txt`.

3.3.2.2 Running the Application

Run the code by going to **Run** → **Resume** or clicking the **Resume** button in the tool bar. Check the following:

- `systemVars.flagEnableSystem` needs to be set to 1 after a fixed time, meaning the offset calibration has been done. The fault flags, `motorVars_M1.faultMtrUse.all`, should be equal to 0. Set the variable `motorVars_M1.flagEnableRunAndIdentify` to 1.
- The motor will 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 doesn't spin smoothly.

```
#define USER_MOTOR1_FREQ_LOW_Hz    (10.0f)    // Hz
#define USER_MOTOR1_FREQ_HIGH_Hz   (200.0f)    // Hz
#define USER_MOTOR1_VOLT_MIN_V      (10.0f)    // Volt
#define USER_MOTOR1_VOLT_MAX_V      (200.0f)    // Volt
```

- After that, the motor spins with a setting speed in the variable motorVars_M1.speedRef_Hz, check the value of motorVars_M1.speed_Hz in Expressions window, the values of these two variables should be very close as shown in [Figure 3-10](#).
- Use the DATALOG module with the CCS **Graph** utility to check the current sensing signals. Go to **Tools** → **Graph** → **Dual Time** to launch the tool and use the **Import** button to navigate to the file <SDK install location>\solutions\tidm_02012_ecompressor\common\debug\motor_datalog_fp2.graphP
rop. Click **OK** and the graphed current readings should be displayed as shown in [Figure 3-11](#). The code

below shows how the variables to be logged are configured in `sys_main.h`. For more details about about the DATALOG module, see the [Motor Control SDK Universal Project and Lab User's Guide](#).

```
// set datalog parameters
datalogObj->iptr[0] = &motorVars_M1.adcData.I_A.value[0];
datalogObj->iptr[1] = &motorVars_M1.adcData.I_A.value[1];
```

- Verify the over current fault protection by decreasing the value of the variable `motorVars_M1.overCurrent_A`, the over current protection is implemented by the CMPSS modules. The over current fault will be triggered if the `motorVars_M1.overCurrent_A` is set to a value less than the motor phase current actual value. When the fault occurs the PWM output will be disabled, the `motorVars_M1.flagEnableRunAndIdentify` will be cleared to 0, and the `motorVars_M1.faultMtrUse.all` will be set to 0x10 (16) as shown in [Figure 3-10](#).
- Set the variable `motorVars_M1.flagEnableRunAndIdentify` to 0 to stop the motor. Terminate the debug session and power off the power supply to the inverter board.

Expression	Type	Value	
(x)= systemVars.flagEnableSystem	unsigned char	1 '\x01'	
(x)= motorVars_M1.isrCount	unsigned long	4926708	
(x)= motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	
(x)= motorVars_M1.motorState	enum <unnamed>	MOTOR_CL_RUNNING	
(x)= motorSetVars_M1.RoverL_rps	float	3726.98096	
(x)= motorSetVars_M1.RsOnLine_Ohm	float	0.540593326	
(x)= motorSetVars_M1.Rs_Ohm	float	0.540593326	
(x)= motorSetVars_M1.Ls_d_H	float	0.000145048587	
(x)= motorSetVars_M1.Ls_q_H	float	0.000145048587	
(x)= motorSetVars_M1.flux_VpHz	float	0.0542857125	Set target speed value (Hz) to this variable
(x)= motorVars_M1.speedRef_Hz	float	15.0	
(x)= motorVars_M1.speed_Hz	float	15.1370792	Check if the estimation speed (Hz) is equal/ close to the setting target speed (Hz)
(x)= motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\x01'	Set this variable value equal to 1 to start the motor
(x)= motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\x01'	
(x)= motorVars_M1.flagClearFaults	unsigned char	0 '\x00'	
(x)= motorVars_M1.faultMtrUse.all	unsigned int	0	Means the inverter/controller has fault when run the motor if the variable value is not zero
> motorVars_M1.faultMtrPrev.bit	struct_FAULT_MTR_BITS_	{overVoltage=0,underVolta...	
(x)= motorSetVars_M1.dacCMPValH	unsigned int	2451	
(x)= motorSetVars_M1.dacCMPValL	unsigned int	1645	
(x)= motorSetVars_M1.overCurrent_A	float	6.5	The threshold value of the over current protection
(x)= motorVars_M1.speedEST_Hz	float	14.0038528	
(x)= motorVars_M1.angleFOC_rad	float	-0.912926078	
(x)= motorVars_M1.angleEST_rad	float	0.720553875	
(x)= motorVars_M1.accelerationMax_Hzps	float	20.0	
(x)= motorVars_M1.accelerationStart_Hzps	float	10.0	
(x)= motorVars_M1.torque_Nm	float	-0.130972117	
(x)= motorVars_M1.adcData.VdcBus_V	float	21.7882328	
(x)= motorVars_M1.Vdq_out_V.value[0]	float	2.70000005	
(x)= motorVars_M1.Vdq_out_V.value[1]	float	4.42536497	
> motorVars_M1.Irms_A	float[3]	{3.48463702,3.61578441,3.4...	
▼ motorVars_M1.adcData	struct_HAL_ADCData_t_	{VdcBus_V=21.551403,I_A=(...	
(x)= VdcBus_V	float	22.051403	
> I_A	struct_MATH_Vec3_	{value=[-4.51171875,-0.338...	
> V_V	struct_MATH_Vec3_	{value=[-2.8962326,-3.5340...	
> offset_I_ad	struct_MATH_Vec3_	{value=[2015.04102,2021.40...	
> offset_V_sf	struct_MATH_Vec3_	{value=[0.54594183,0.56457...	
(x)= current_sf	float	0.0161132812	
(x)= voltage_sf	float	0.23682861	
(x)= dcBusvoltage_sf	float	0.23682861	
> Cmpss1Regs.COMPSTS	Register	0x0000	One or more of these registers values will be non-zero if there is a fault
> Cmpss2Regs.COMPSTS	Register	0x0000	
> Cmpss3Regs.COMPSTS	Register	0x0000	
> EPwm1Regs.TBPRD	Register	0x0FA0	
(x)= EPwm1Regs.CMPA.CMPA	pointer : 16	0x0933	

Figure 3-10. Build Level 2 Variables in the Expressions View

The Graph tool can be used to display the DATALOG buffers.

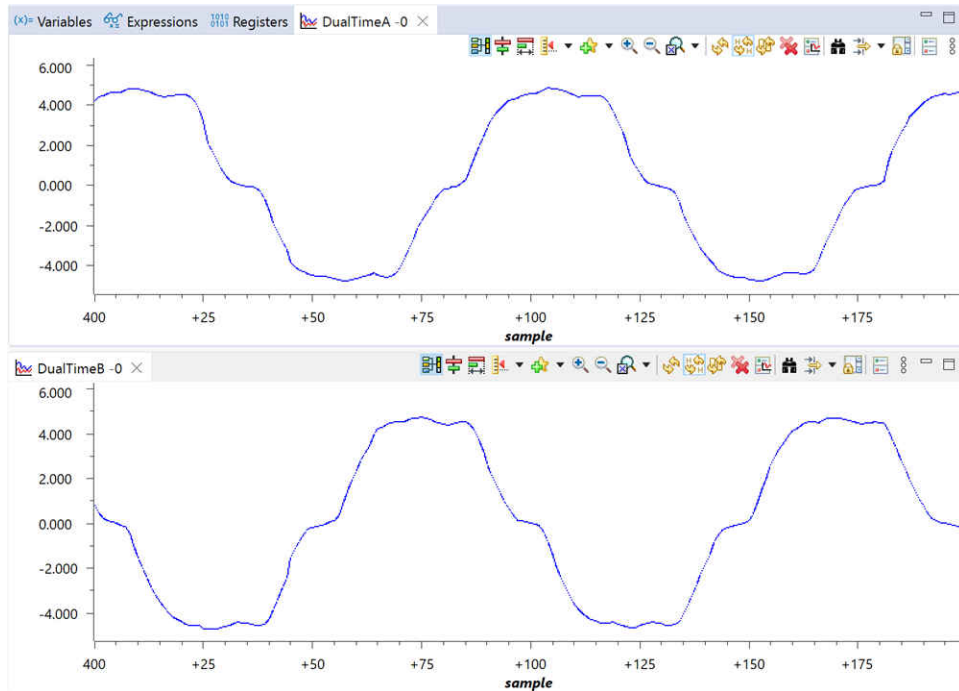


Figure 3-11. Build Level 2 Data Logged Current Readings in the CCS Graph Tool

The over current fault will trip the PWM output and also be reflected in the fault flag variables.

motorVars_M1.flagClearFaults	unsigned char	0 '\x00'	
motorVars_M1.faultMtrUse.all	unsigned int	16	
motorVars_M1.faultMtrPrev.bit	struct_FAULT_MTR_BITS_	{overVoltage=0,underV	This value will be non-zero if there is an over-current fault
overVoltage	unsigned int : 1	0	
underVoltage	unsigned int : 1	0	
motorOverTemp	unsigned int : 1	0	
moduleOverTemp	unsigned int : 1	0	
moduleOverCurrent	unsigned int : 1	1	Over-current flag is set
overPeakCurrent	unsigned int : 1	0	
overLoad	unsigned int : 1	0	
motorLostPhase	unsigned int : 1	0	
currentUnbalance	unsigned int : 1	0	
motorStall	unsigned int : 1	0	
startupFailed	unsigned int : 1	0	
overSpeed	unsigned int : 1	0	
reserve12	unsigned int : 1	0	
reserve13	unsigned int : 1	0	
currentOffset	unsigned int : 1	0	
voltageOffset	unsigned int : 1	0	
motorSetVars_M1.dacCMPValH	unsigned int	2110	
motorSetVars_M1.dacCMPValL	unsigned int	1986	
motorSetVars_M1.overCurrent_A	float	1.0	Adjust the current threshold value to verify the over current function

Figure 3-12. Build Level 2 Over Current Protection Check in the Expressions View

3.3.3 Level 3 Incremental Build

Objectives of this build level:

- Evaluate the closed current loop operation of the motor.
- Verify the current sensing parameters settings.

In this build level, the motor is controlled by using i/f control that the rotor angle is generated from ramp generator module. The software flow for this build level is shown in [Figure 3-13](#).

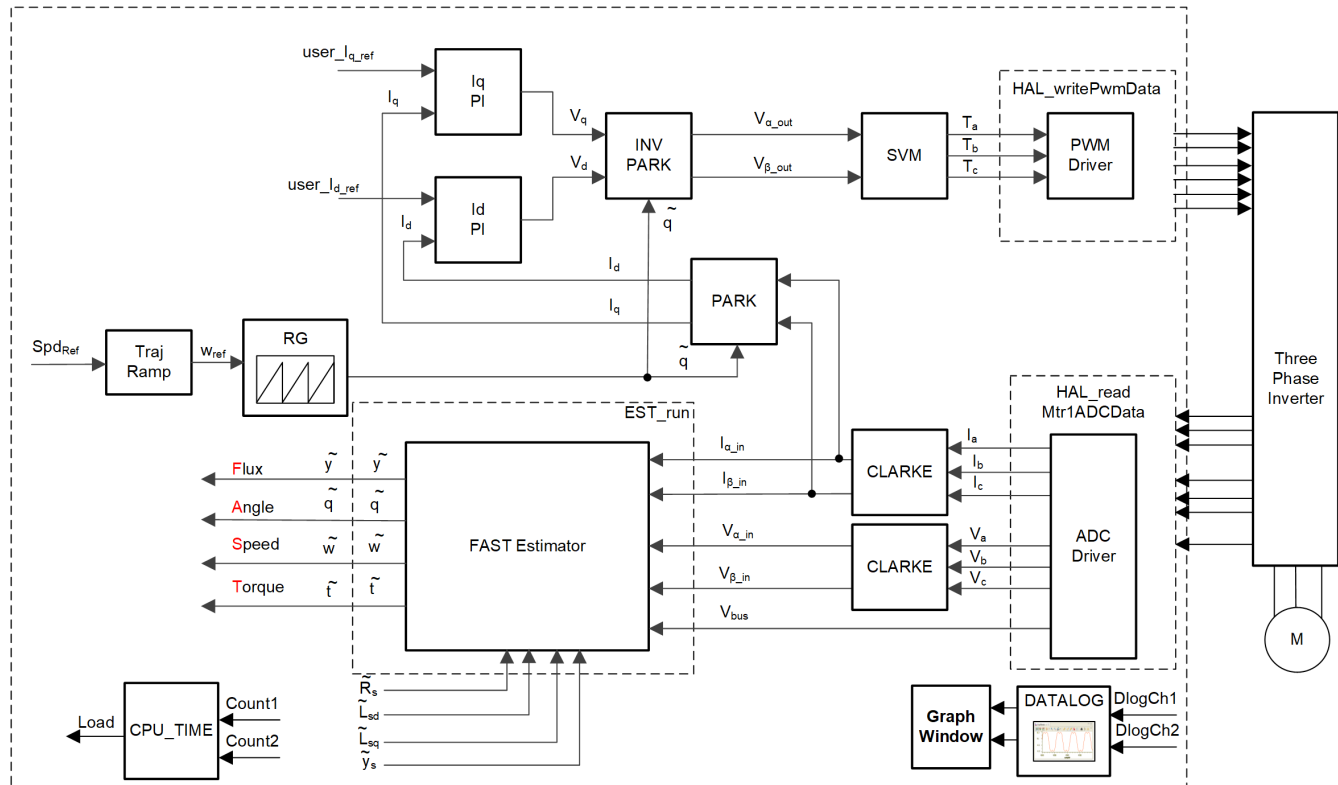


Figure 3-13. Build Level 3 Software Block Diagram

3.3.3.1 Project Setup

Follow the steps described in [Section 3.3.1.1](#) to change `DMC_BUILDLEVEL` to `DMC_LEVEL_3` and rebuild and load the project. You should also change the contents of the **Expressions** view to the variables listed in `tidm_02012_level3.txt`.

3.3.3.2 Running the Application

Run the code by going to **Run** → **Resume** or clicking the **Resume** button in the tool bar. Check the following:

- `systemVars.flagEnableSystem` should be set to 1 after a fixed time, meaning the offset calibration has been done. The fault flags, `motorVars_M1.faultMtrUse.all`, should be equal to 0.
- To run the motor with current closed-loop control, set the variable `motorVars_M1.flagEnableRunAndIdentify` to 1 in the **Expressions** window. The motor will run with 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 `motorVars_M1.speed_Hz` to confirm that the two values are very close.
- Change the value of `Idq_set_A.value[1]` in the Expressions window to set the reference torque current. The motor phase current will be increasing with the same percentage accordingly.
- Set the variable `motorVars_M1.flagEnableRunAndIdentify` to 0 to stop the motor. Terminate the debug session and power off the power supply to the inverter board.

Registers		
Expression	Type	Value
(x)= systemVars.flagEnableSystem	unsigned char	1 '\x01'
(x)= motorVars_M1.isrCount	unsigned long	1443642
(x)= motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE
(x)= motorVars_M1.motorState	enum <unnamed>	MOTOR_CTRL_RUN
(x)= motorSetVars_M1.RoverL_rps	float	3726.98096
(x)= motorSetVars_M1.RsOnLine_Ohm	float	0.540593326
(x)= motorSetVars_M1.Rs_Ohm	float	0.540593326
(x)= motorSetVars_M1.Ls_d_H	float	0.000145048587
(x)= motorSetVars_M1.Ls_q_H	float	0.000145048587
(x)= motorSetVars_M1.flux_VpHz	float	0.0419800468
(x)= motorVars_M1.adcData_VdcBus_V	float	21.7882328
(x)= motorVars_M1.speedRef_Hz	float	40.0
(x)= motorVars_M1.speed_Hz	float	40.1356735
(x)= motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 '\x01'
(x)= motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 '\x01'
(x)= motorVars_M1.flagEnableMotorIdentify	unsigned char	0 '\x00'
(x)= motorVars_M1.flagEnableForceAngle	unsigned char	1 '\x01'
(x)= motorVars_M1.enableSpeedCtrl	unsigned char	1 '\x01'
(x)= motorVars_M1.speedEST_Hz	float	40.2982712
(x)= motorVars_M1.angleFOC_rad	float	-0.232513309
(x)= motorVars_M1.angleEST_rad	float	1.94626439
(x)= motorVars_M1.accelerationMax_Hzps	float	20.0
(x)= motorVars_M1.accelerationStart_Hzps	float	10.0
(x)= motorVars_M1.flagClearFaults	unsigned char	0 '\x00'
(x)= motorVars_M1.faultMtrUse.all	unsigned int	0
> motorVars_M1.faultMtrPrev.bit	struct _FAULT_MTR_BITS_	{overVoltage=0,underVolta...
(x)= motorSetVars_M1.dacCMPValH	unsigned int	2451
(x)= motorSetVars_M1.dacCMPValL	unsigned int	1645
(x)= motorSetVars_M1.overCurrent_A	float	6.5
(x)= motorVars_M1.startCurrent_A	float	3.5
(x)= motorVars_M1.maxCurrent_A	float	6.0
(x)= motorVars_M1.Idq_set_A.value[1]	float	3.5
(x)= motorVars_M1.IdqRef_A.value[0]	float	0.0
(x)= motorVars_M1.IdqRef_A.value[1]	float	3.5
> motorVars_M1.Irms_A	float[3]	[2.49551034,2.5000093,2.48...
(x)= motorSetVars_M1.Kp_Id	float	0.364546865
(x)= motorSetVars_M1.Ki_Id	float	0.062116351
(x)= motorSetVars_M1.Kp_Iq	float	0.364546865
(x)= motorSetVars_M1.Ki_Iq	float	0.062116351
(x)= motorSetVars_M1.Kp_spd	float	0.0489085019
(x)= motorSetVars_M1.Ki_spd	float	0.0167551599

Figure 3-14. Build Level 3 Variables in the Expressions View

3.3.4 Level 4 Incremental Build

Objectives of this build level:

- Evaluate motor identification with FAST estimators.
- Evaluate the complete motor drive with FAST-based sensorless-FOC.
- Evaluate the additional features, such as field weakening control, MTPA, and torque compensation.

In this build level, the outer speed loop is closed with the inner current loop for motor that the rotor angle is from FAST. The software flow for this build level is shown in [Figure 3-15](#).

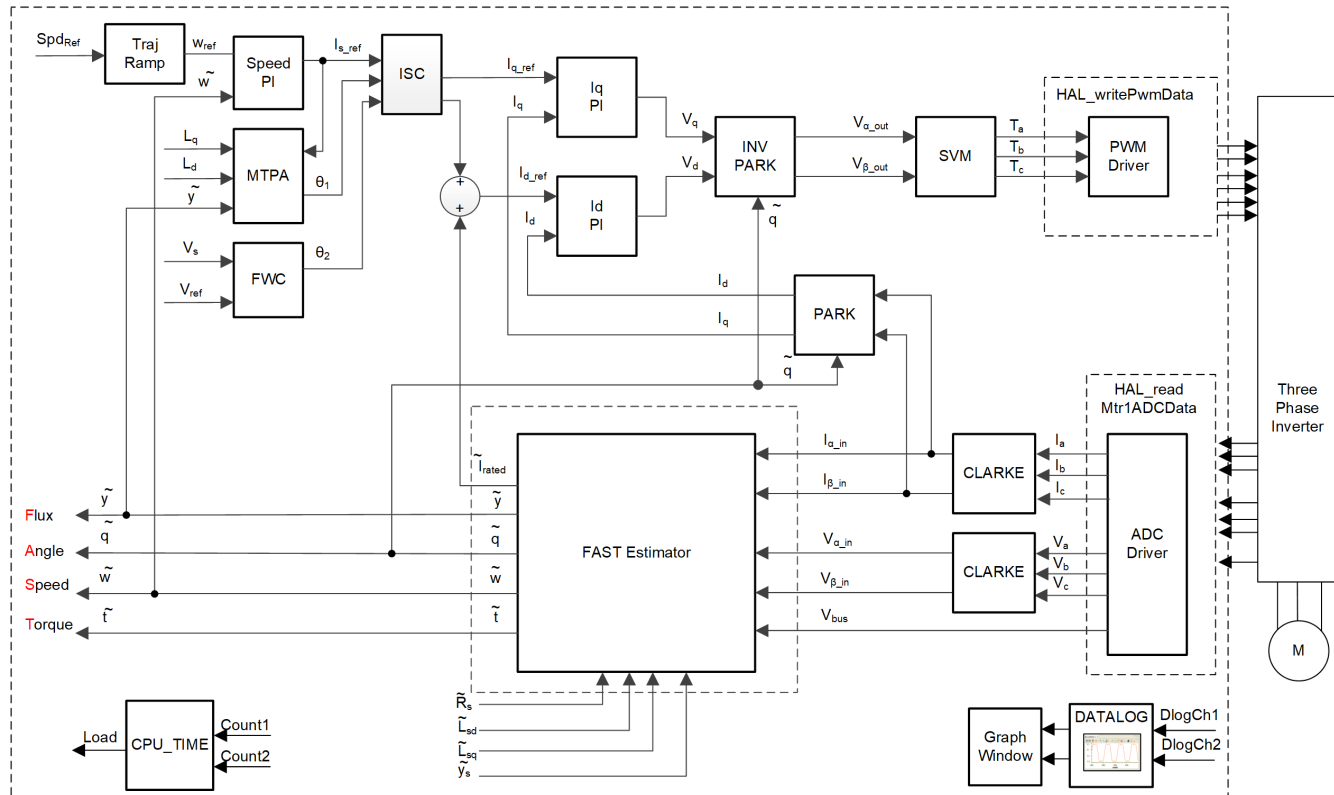


Figure 3-15. Build Level 4 Software Block Diagram

3.3.4.1 Project Setup

Follow the steps described in [Section 3.3.1.1](#) to change *DMC_BUILDLEVEL* to *DMC_LEVEL_4*. There are several additional edits that should be made for this build level:

- To run motor identification, you should also change the *MOTOR_IDENT* definition in *sys_settings.h* to 1 and rebuild and load the project.
- The required motor parameters must be defined in the header files *user_mtr1.h* as shown in the following example codes. If the motor parameters are not well known by the user, the motor identification can be used to achieve the motor parameters if the FAST estimator is implemented in the example lab.

```
#define USER_MOTOR1_Rs_Ohm           (0.540593326f)
#define USER_MOTOR1_Ls_d_H          (0.000145048587f)
#define USER_MOTOR1_Ls_q_H          (0.000145048587f)
#define USER_MOTOR1_RATED_FLUX_VpHz (0.038f)
```

- Set the right identification values in the *user_mtr1.h* according to the motor specification.

```
#define USER_MOTOR1_RES_EST_CURRENT_A (1.5f) // 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 (6.0f) // 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 and load the project. You should also change the contents of the **Expressions** view to the variables listed in *tidm_02012_level4.txt*.

3.3.4.2 Running the Application

Run the code by going to **Run** → **Resume** or clicking the **Resume** button in the tool bar.

systemVars.flagEnableSystem should be set to 1 after a fixed time, meaning the offset calibration has been done. The fault flags, *motorVars_M1.faultMtrUse.all*, should be equal to 0.

Set the variable *motorVars_M1.flagEnableRunAndIdentify* to 1. The motor identification will be executed; the whole process will take about 150 seconds. Once *motorVars_M1.flagEnableRunAndIdentify* is equal to 0 and the

motor stops, the motor parameters have been identified. Copy the variable values in the Expressions window to replace the defined motor parameters in *user_mtr1.h* as follows:

- `USER_MOTOR1_Rs_Ohm` = `motorSetVars_M1.Rs_Ohm`'s value
- `USER_MOTOR1_Ls_d_H` = `motorSetVars_M1.Ls_d_H`'s value
- `USER_MOTOR1_Ls_q_H` = `motorSetVars_M1.Ls_q_H`'s value
- `USER_MOTOR1_RATED_FLUX_VpHz` = `motorSetVars_M1.flux_VpHz`'s value

Change `MOTOR_IDENT` back to 0 to disable identification after successfully identifying the motor parameters. Rebuild and reload the application.

Once the motor parameters identification completes or are set correctly in the *user_mtr1.h* file, start the motor by setting `motorVars_M1.flagEnableRunAndIdentify` equal to 1 again. Check the following:

- Set the target speed value using the variable `motorVars_M1.speedRef_Hz` and watch how the motor shaft speed follows the set speed. To change the acceleration, enter a different value into the variable `motorVars_M1.accelerationMax_Hzps`.
- You can use the CCS Graph tool to monitor variables as described in [Section 3.3.2](#). By default for this build level, the motor angle and current waveforms are logged.
- The default proportional gain (Kp) and integral gain (Ki) for the current controllers of the FOC system are calculated in the function `setupControllers()`. After `setupControllers()` is called, the global variables `motorSetVars_M1.Kp_Id`, `motorSetVars_M1.Ki_Id`, `motorSetVars_M1.Kp_Iq`, and `motorSetVars_M1.Ki_Iq` are initialized with the newly calculated Kp and Ki gains. Tune the Kp and Ki value of these four variables in **Expressions** window as shown in [Figure 3-16](#) for the current controllers to achieve the expected current control bandwidth and response. The Kp gain creates a zero that cancels the pole of the motor's stator and can easily be calculated. The Ki gain adjusts the bandwidth of the current controller-motor system. When a speed controlled system is needed for a certain damping, the Kp gain of the current controller will relate to the time constant of the speed controlled system.
- The default proportional gain (Kp) and integral gain (Ki) for the speed controllers of the FOC system are also calculated in the function `setupControllers()`. After `setupControllers()` is called, the global variables `motorSetVars_M1.Kp_spd` and `motorSetVars_M1.Ki_spd` are initialized with the newly calculated Kp and Ki gains. Tune the Kp and Ki value of these two variables in **Expressions** window as shown in [Figure 3-16](#) for the speed controllers to achieve the expected current control bandwidth and response. Tuning the speed controller has more unknowns than when tuning a current controller. The default calculated Kp and Ki are just serve as a starting point starting point.

Expression	Type	Value	
(*)= systemVars.flagEnableSystem	unsigned char	1 "\x01"	
(*)= motorVars_M1.isrCount	unsigned long	1194180	
(*)= motorVars_M1.speed_Hz	float	41.3085709	Estimation feedback speed (Hz)
(*)= motorVars_M1.speedRef_Hz	float	40.0	Set target speed value (Hz) to this variable
(*)= motorVars_M1.flagEnableRunAndIdentify	unsigned char	1 "\x01"	Set this variable value equal to 1 to start the motor
(*)= motorVars_M1.flagRunIdentAndOnLine	unsigned char	1 "\x01"	
(*)= motorVars_M1.accelerationMax_Hzps	float	20.0	
(*)= motorVars_M1.accelerationStart_Hzps	float	10.0	
(*)= motorVars_M1.flagEnableForceAngle	unsigned char	1 "\x01"	
(*)= motorVars_M1.flagMotorIdentified	unsigned char	1 "\x01"	
(*)= motorVars_M1.flagEnableMotorIdentify	unsigned char	0 "\x00"	
(*)= motorVars_M1.estState	enum <unnamed>	EST_STATE_ONLINE	Estimator state
(*)= motorVars_M1.motorState	enum <unnamed>	MOTOR_CTRL_RUN	Motor operation state
(*)= motorVars_M1.svmMode	enum <unnamed>	SVM_MIN_C	
(*)= motorVars_M1.adcData.VdcBus_V	float	21.7882328	
(*)= motorSetVars_M1.Kp_Id	float	0.364546865	
(*)= motorSetVars_M1.Ki_Id	float	0.062116351	
(*)= motorSetVars_M1.Kp_Iq	float	0.364546865	
(*)= motorSetVars_M1.Ki_Iq	float	0.062116351	
(*)= motorSetVars_M1.Kp_spd	float	0.0489085019	
(*)= motorSetVars_M1.Ki_spd	float	0.0167551599	Tune these Kp or Ki of current and speed regulators to achieve the required response
(*)= motorVars_M1.flagClearFaults	unsigned char	0 "\x00"	
(*)= motorVars_M1.faultMtrUse.all	unsigned int	0	
(*)= motorVars_M1.faultMtrNow.all	unsigned int	128	
> (*)= motorVars_M1.faultMtrPrev.bit	struct _FAULT_MTR_BITS_	{overVoltage=0,underVolta...	
(*)= motorSetVars_M1.dacCMPValH	unsigned int	2451	
(*)= motorSetVars_M1.dacCMPValL	unsigned int	1645	
(*)= motorSetVars_M1.overCurrent_A	float	6.5	
(*)= motorSetVars_M1.RoverL_rps	float	3726.98096	
(*)= motorSetVars_M1.RsOnLine_Ohm	float	0.540593326	
(*)= motorSetVars_M1.Rs_Ohm	float	0.540593326	
(*)= motorSetVars_M1.Ls_d_H	float	0.000145048587	
(*)= motorSetVars_M1.Ls_q_H	float	0.000145048587	
(*)= motorSetVars_M1.flux_VpHz	float	0.0402238332	Setting/Identified motor electrical parameters
(*)= motorVars_M1.speedEST_Hz	float	38.8036995	
(*)= motorVars_M1.speedPLL_Hz	float	0.0	
(*)= motorVars_M1.angleFOC_rad	float	2.35817194	
(*)= motorVars_M1.angleEST_rad	float	2.69316673	
(*)= motorVars_M1.anglePLL_rad	float	0.0	
(*)= userParams_M1.maxVsMag_V	float	12.5500212	
(*)= userParams_M1.maxVsMag_pu	float	0.575999975	
(*)= motorVars_M1.Vs_V	float	2.97489977	
(*)= motorVars_M1.VsRef_V	float	225.792007	
(*)= motorVars_M1.VsRef_pu	float	0.564480007	

Figure 3-16. Build level 4 variables in the Expressions view

3.3.4.3 Tuning Field Weakening and MTPA Control

The FWC and MTPA functions can optionally be called in the motor drive ISR to calculate the current angle and then compute the reference currents of d-axis and q-axis. Add the predefined symbols *MOTOR1_FWC* and *MOTOR1_MTPA* in the compiler settings of the project as described in [Section 3.2.2.1](#) to enable the FWC and MTPA respectively.

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 according to the specification of the motor.

Add the variables *motorVars_M1.VsRef_pu*, *motorSetVars_M1.Kp_fw*, and *motorSetVars_M1.Ki_fw* to **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 system.

After you've tuned and fixed these variables value, record the Expressions window values with the newly defined parameters in *user_mtr1.h* file:

- `USER_M1_FWC_VREF` = *motorVars_M1.VsRef_pu*'s value, the reference voltage for FWC
- `USER_M1_FWC_KP` = *motorSetVars_M1.Kp_fwc*'s value, the Kp gain of PI regulator for FWC
- `USER_M1_FWC_KI` = *motorSetVars_M1.Ki_fwc*'s value, the Ki gain of PI regulator for FWC

MTPA control parameters are calculated according to the motor parameters, L_d , L_q and ψ_m , so there are no additional parameters that need to be tuned online.

3.3.4.4 Tuning Vibration Compensation

The automatic vibration compensation function can optionally be called in the motor drive ISR to calculate feedforward torque current. Add the predefined symbol `MOTOR1_VIBCOMPA` in the compiler settings of the project as described in [Section 3.2.2.1](#) to enable vibration compensation.

Add the variables *motorVars_M1.vibCompAlpha*, *motorVars_M1.vibCompGain*, and *motorVars_M1.vibCompIndexDelta* to **Expressions** window in CCS Debug perspective, and tuning these parameters to achieve the expectation performance for the vibration compensation according to the compressor and air-conditioner system.

- *motorVars_M1.vibCompAlpha* is used as the learning speed. The higher this value (with a maximum of 1.0) the slower it learns the algorithm. A high value is desirable though, since it provides noise immunity.
- *motorVars_M1.vibCompIndexDelta* is to advance the output waveform into the future by a little bit so that the resulting current will be very close to the desired value by the time the mechanical angle reaches that point. A typical value of 10 is recommended, but ultimately needs to be fine-tuned by the user.
- *motorVars_M1.vibCompGain* is the gain factor of the feedforward torque reference torque current value (with a maximum of 1.0).

Change speed reference (*motorVars_M1.speedRef_Hz*) and speed controller gains (*motorSetVars_M1.Kp_spd* and *motorSetVars_M1.Ki_spd*). This step is used to take the motor and load to where the motor vibrates due to the pulsating load. For vibration compensation to work better, increase the values of the speed controller gains. Make sure the speed controller is still stable though.

Adding the predefined symbol `SPEED_MONITOR_EN` in build configuration of the project for enabling the motor running speed vibration. Now enable the vibration compensation output by setting this flag, *motorVars_M1.vibCompFlagEnable* = 1. Then let it run for a 5 to 10 seconds, and then get the new speed variation by setting this bit: *motorVars_M1.flagClearRecord* = 1. Record that the speed variation is about 2Hz.

If the vibration was not reduced, try increasing the speed controller gains. Also try increasing the learning speed of the vibration compensation algorithm by decreasing the value of *motorVars_M1.vibCompAlpha* in decrements of 0.02, so try: 0.99, 0.97, 0.95, etc. each time you change *motorVars_M1.vibCompAlpha*, let it run for a few seconds and get a reading of the speed variation by resetting that calculation: *motorVars_M1.flagClearRecord* = 1.

After tuned and fixed these variables value, record the watch window values with the newly defined parameters in *user_mtr1.h* file.

- `USER_MOTOR1_VIBCOMPA_ALPHA` = *motorVars_M1.vibCompAlpha*'s value, the learning rate of the vibration compensation module from 0.0 to 1.0
- `USER_MOTOR1_VIBCOMPA_GAIN` = *motorVars_M1.vibCompGain*'s value, the gain of the vibration compensation module from 0.0 to 1.0
- `USER_MOTOR1_VIBCOMPA_INDEX_DELTA` = *motorVars_M1.vibCompIndexDelta*'s value, the phase advance of the vibration compensation module from 0 to 360

Controlling motor current based on compressor torque vs angle is an alternate technique to counter the speed ripple variations. Adding the predefined symbol `MOTOR1_VIBCOMPT` in the compiler settings of the project to enable this vibration compensation method.

Depending upon the rolling piston angle, an additional torque current component can be either added or subtracted from the speed PI controller output. The current needs to be added during compression stage and subtracted during exhaustion (where it aids the movement of piston leading to increase in speed) and its magnitude can be computed empirically to match the torque profile of the compressor. Algorithm has split

the 360 mechanical deg into 3 sector and compensation current can be added separately through variables *motorVars_M1.vibCompAlpha0*, 120 and 240. With rough tuning of the compensation, the speed ripple is reduced from 200Hz to under 100Hz @ 1200rpm. Further reduction in speed ripple is possible, when tuning matches the torque profile to the closest. Usually vibration compensation is enabled for compressor speeds between 1200 - 2000rpm (100Hz), post which its impact tends to be smaller.

3.3.4.5 CAN FD Command Interface

A CAN FD interface for sending commands and receiving debug information can optionally be added to the project. This interface not only demonstrates the use of the MCAN module on the C2000 device, but is helpful for debugging the project in a situation where a JTAG-based debug probe connection is inconvenient.

You will need to connect the CAN FD interface on the reference design board to an external device with which the reference design application will be communicating. For example, you could use a CAN FD-to-USB adapter to send and receive frames from a PC. You could also use another C2000 development board with a CAN FD transceiver. The TMS320F280039C LaunchPad™ development kit (LAUNCHXL-F280039C) has an on-board CAN transceiver and is a useful tool for this purpose. A CAN FD communication utility application for the LAUNCHXL-F280039C has been provided as part of the reference design software folder. This will be the method demonstrated in this section.

Note

To connect to multiple devices in CCS at the same time, extra steps may be required, especially in the case where the two boards use the same debug probe type. See the article [Debugging with Multiple XDS Debug Probes](#) for help in setting up your debug environment. It will likely be easiest to use two instance of CCS—one for the main application and one for the CAN communication utility.

Following the steps described in [Section 3.2.2.1](#), import the project called *tidm_02012_cancom_util_<device>*. Build the project, connect to the LaunchPad, and load the application. In the **Expressions** view, add the variables *speedSet_Hz*, *flagEnableCmd*, *flagCmdRun*, and *canComVars*, or import the *tidm_02012_cancom_util.txt* file.

Add support for the CAN command interface in the main *tidm_02012_ecompressor* project by adding the predefined symbol *CMD_CAN_EN* in the compiler settings of the project as described in [Section 3.2.2.1](#). Build and load the project and add the *canComVars* and *motorVars_M1.cmdCAN* variables to your **Expressions** view or import the set of variables found in *tidm_02012_can.txt*.

Run both applications. If the CAN FD communication is working you should be able to see the *canComVars.txMsgCount* and *canComVars.rxMsgCount* incrementing in both devices' debug sessions. In the LaunchPad CAN utility debug session, try updating *speedSet_Hz*. You should see the update reflected in *motorVars_M1.speedRef_Hz* in the main application. If it's not updated, check that there aren't any motor fault flags set.

To send a command to run the motor, set *flagCmdRun* in the LaunchPad debug session. This will set *motorVars_M1.flagEnableRunAndIdentify* in the main application. In the LaunchPad debug session, you should see the *canComVars* updated with status information sent from the main application as shown in [Figure 3-17](#).

Expression	Type	Value	
(x)= speedSet_Hz	float	40.0	Speed setting (in Hz) to send
(x)= flagEnableCmd	unsigned char	1 '\x01'	
(x)= flagCmdRun	unsigned char	1 '\x01'	Set to 1 to send command to run the motor
(x)= canComVars.motorStateRx	enum <unnamed>	MOTOR_STOP_IDLE	
(x)= canComVars.speedRx_Hz	float	42.0	Status information received from the main application
(x)= canComVars.lqRx_A	float	1.12	
(x)= canComVars.rxMsgCount	unsigned int	33155	
(x)= canComVars.txMsgCount	unsigned int	33158	For debug—if CAN FD interface is working, MsgCounts should be increasing continuously
(x)= canComVars.errorFlag	unsigned int	0	

Figure 3-17. CAN FD Command Utility Variables in the Expressions View

3.4 Test Results

The eCompressor motor drive is tested with 3 phase PMSM motor with 400 V DC bus input. The labs 1-4 are tested and the motor can run smoothly with FAST sensorless FOC control.

The CAN FD communication with another TMS320F280039C (LaunchPAD) is tested, including remote control and data monitoring functionality.

3.4.1 MCU CPU Load, Memory, and Peripheral Usage

Table 3-2 shows the CPU cycles used and the CPU loading when running the reference project on the F280039C with a 120 MHz CPU clock. These numbers are based on build level 4 and use the project's default settings regarding which functions are run from RAM (such as the main ISR) and which functions are run from Flash.

Table 3-2. CPU Loading

CPU=120MHz	Maximum CPU Cycles For ISR	Maximum CPU Utilization [%]	Maximum MIPS Used [MIPS]
CPU Utilization (15-kHz ISR)	2079	25.99	31.185

Table 3-3 shows how much memory is needed to run the application on the microcontroller. This memory footprint is based on the default project settings. Adding additional features (such as MTPA or vibration compensation) or removing features (such as switching from *fast_full_lib.lib* to *fast_simple_lib.lib* to remove motor identification) results in some change to the memory footprint. As shown, a significant part of the memory is still available for performing other tasks.

Table 3-3. Memory Usage

Type	Used Memory on F280039C	Available Memory on F280039C	F280039C Memory Utilization
FLASH	41.7 KB	384 KB	10.9%
RAM	15.3 KB	69 KB	22.2%

Table 3-4 lists the peripherals used by this reference design.

Table 3-4. F28003x Peripheral Usage

Module	Purpose
ADCA, ADCB, ADCC	Three-phase PWM (total of 6 PWM channels)
EPWM1, EPWM2, EPWM3	Motor current and voltage sensing (total of 7 ADC channels)
CMPSS1, CMPSS2, CMPSS3	Three-phase overcurrent fault protection
EPWMXBAR TRIP7	CMPSS output to the EPWM trip for overcurrent protection
MCANA	Communication
CPU Timer 0	Virtual timer for motor and system control in background loop
GPIOs	One for controlCARD LED D2, One for DISABLE_FET_SUPPLY signal

4 Design and Documentation Support

4.1 Design Files

4.1.1 Schematics

To download the schematics, see the design files at [TIDM-02012](#).

4.1.2 BOM

To download the bill of materials (BOM), see the design files at [TIDM-02012](#).

4.2 Tools and Software

Tools

[TMDSCNCD280039C](#) The TMS320F280039C evaluation module controlCARD is a low-cost evaluation and development board for the series of F28003x devices. It is used as a daughter card for this design connected to the reference board through a standard 180-pin controlCARD HSEC interface.

[LAUNCHXL-F280039C](#) LAUNCHXL-F280039C is a low-cost development board for F28003x devices. As part of the vast TI MCU LaunchPad™ ecosystem, it is also cross-compatible with a broad range of plug-in modules. In this reference design, it is a helpful CAN FD debug tool since it has an on-board CAN transceiver.

Software

[C2000WARE-MOTORCONTROL-SDK](#) The MotorControl SDK is a set of software, tools, and documentation designed to C2000 real-time controller based motor control system development time. The software for this reference design can be found in v4.01.00.00 or newer.

[CCSSTUDIO](#) The Code Composer Studio Integrated Development Environment is required for building and running the reference design software project. CCS v12.00.00 or newer is recommended for this project.

4.3 Documentation Support

1. Texas Instruments, [TMS320F28003x Real-Time Microcontrollers](#) Technical Reference Manual
2. Texas Instruments, [TMS320F28003x Real-Time Microcontrollers](#) data sheet
3. Texas Instruments, [TMS320F280039C controlCARD Information Guide](#)
4. Texas Instruments, [InstaSPIN-FOC™ and InstaSPIN-MOTION™ User's Guide](#)
5. Texas Instruments, [Sensorless-FOC for PMSM With Single DC-Link Shunt](#)
6. Texas Instruments, [Motor Control SDK Universal Project and Lab User's Guide](#)

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

C2000™, TI E2E™, FAST™, LaunchPad™, and are trademarks of Texas Instruments.
All trademarks are the property of their respective owners.

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](#) 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 © 2022, Texas Instruments Incorporated