Design Guide: TIDM-02018

Universal Motor Control Reference Design for AM263x Arm® Based MCU Devices

Description

This reference design offers a universal motor control design for TI's AM263x Arm® based MCUs. The design shows how to use AM263x MCUs for various types of FOC motor control techniques, such as sensorless (eSMO) and sensored (incremental encoder, Hall sensor). The design supports two main hardware setups: a low-voltage setup using AM263x LaunchPad™ with 3PHGANINV BoosterPack™ and a high-voltage setup using AM263x controlCARD™ with TMDSHVMTRINSPIN motor control kit. The documentation also covers instructions on migrating the design to a custom board and porting the project to new devices.

Resources

TIDM-02018	Design Folder
AM2634-Q1	Product Folder
AM263x MCU+ SDK	Tool Folder
AM263x Academy	Training Materials

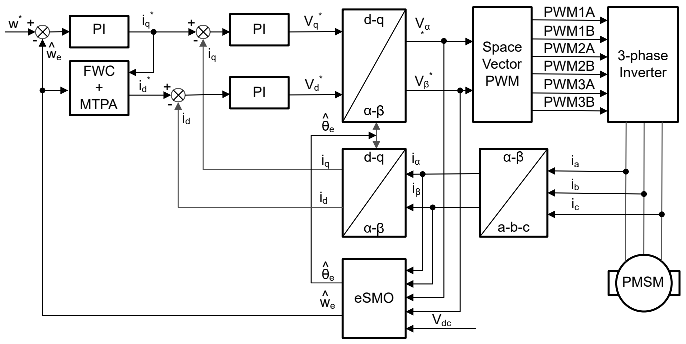
Ask our TI E2E™ support experts

Features

- Comprehensive software package, tools, and documentation with this reference design helps reduce the development time of motor control systems based on AM263x MCUs.
- Various FOC motor control methods: Sensorless (eSMO) and Sensored (Incremental Encoder, Hall sensor) control are supported.
- TI's system features and debug interfaces that are compatible with many three-phase inverter motor evaluation kits are included.
- Sysconfig-based project allows for easier migration between different devices and boards. This is enabled by a user-friendly graphical user interface that allows users to adjust pins, peripherals, software stacks, clock tree, and other elements, thus speeding up software development.

Applications

- HEV/EV Inverter and Motor Control
- Motor Drives
- AC Inverter and VF Drives
- AC Drive Control Module



1 System Description

The Universal Motor Control project described in this reference design is intended for you to not only experiment with various motor control algorithms but also to use as a reference for your own design. This single project with different *Build Configuration* options provides different FOC motor control techniques including sensorless (eSMO) and Sensored (Incremental Encoder, Hall sensor) together with TI's system features and debug interfaces. This project offers various features such as: data logging, software frequency response analyzer (SFRA), motor PI tune, field weakening, maximum torque per ampere (MPTA), CPU time computing, EPWM DAC mode, step response module, and phase adjustment. This reference design evaluates two hardware kits that can be ordered from TI.com:

- BOOSTXL-3PHGANINV + LVSERVOMTR + LP-AM263
- TMDSHVMTRINSPIN + HVPMSMMTR + TMDSCNCD263

1.1 Terminology

FOC	Field Oriented Control
eSMO	Enhanced Sliding-Mode Observer
PMSM	Permanent Magnet Synchronous Motor
EEMF	Extended Electromotive Force
PLL	Phase Locked Loop
IPMS	Interior PMSM
FW	Field Weakening
MTPA	Maximum Torque Per Ampere
SVPWM	Space Vector Modulation
PWM	Pulse Width Modulation
RPM	Revolutions Per Minute

1.2 Key System Specifications

- For information on the 48V three-phase inverter with shunt-based in-line motor phase current sensing evaluation module, see the BOOSTXL-3PHGANINV design files and technical documentation.
- For information on the high voltage motor control kit, refer to the TMDSHVMTRINSPIN.
- For information on the low voltage servo motor, encoder and wiring harness, refer to LVSERVOMTR data sheet.
- For information on the high voltage permanent magnet synchronous motor, see the HVPMSMMTR data sheet.

2 System Overview

2.1 Block Diagram

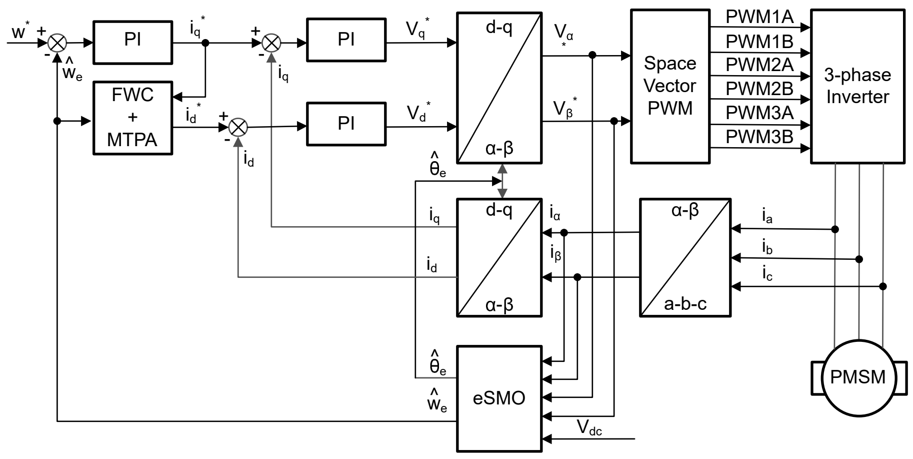


Figure 2-1. eSMO Based Sensorless FOC Block Diagram for PMSM Motor

2.2 Highlighted Products

2.2.1 AM263x Microcontrollers

The AM263x Arm® based Microcontrollers are built to meet the complex real-time processing needs of next generation industrial and automotive embedded products. The AM263x MCU family consists of multiple pin-to-pin compatible devices with up to four 400MHz Arm® Cortex®-R5F cores. The multiple Arm® cores can be optionally programmed to run in lock-step option for different functional safety configurations. The industrial communications subsystem (ICSS) enables integrated industrial Ethernet communications such as PROFINET IRT, TSN, or EtherCAT® (among many others), or for standard Ethernet connectivity or custom I/O interfacing. The AM263x family is designed for advanced motor control and digital power control applications with advanced analog modules.

2.2.1.1 TMDSCNCD263

The AM263x Control Card Evaluation Module (EVM) is an evaluation and development board for the Texas Instruments Sitara™ AM263x series of microcontrollers (MCUs). This EVM provides an easy way to start developing traction inverter designs on the AM263x MCUs with on-board emulation for programming and debugging as well as buttons and LED for a simple user interface. The control card also enables header-pin access to key for rapid prototyping.

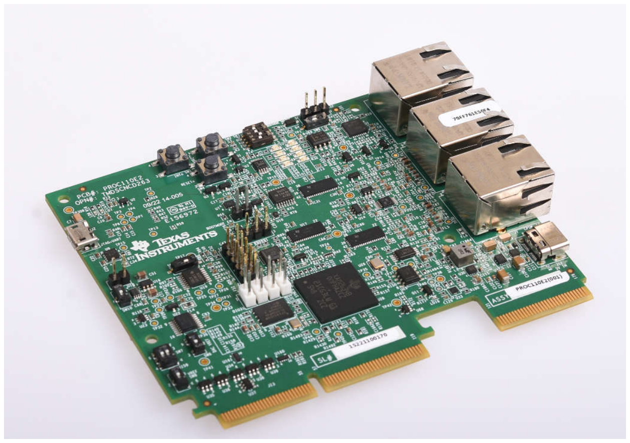


Figure 2-2. AM263x controlCARD™

2.2.1.2 LP-AM263

LP-AM263 is a cost-optimized development board for Sitara™ high-performance microcontrollers (MCUs) from the AM263x series. This board is an excellent choice for initial evaluation and prototyping as the board provides a standardized and easy-to-use platform to develop your next application.

LP-AM263 is equipped with a Sitara AM2634 processor, along with additional components, allowing the user to make use of various device interfaces, including industrial Ethernet (IE), standard Ethernet, fast serial interface (FSI) and others to easily create prototypes. AM2634 supports a variety of IE protocols, such as EtherCAT, EtherNet/IP and PROFINET®.

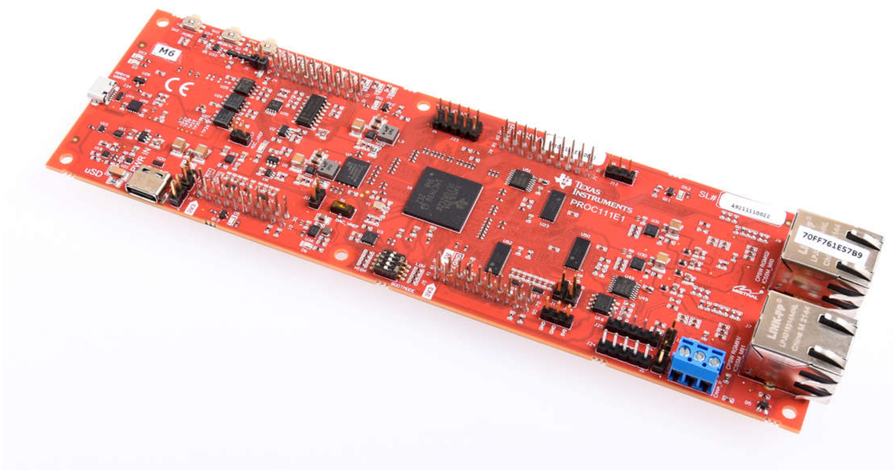


Figure 2-3. AM263x LaunchPad™

3 System Design Theory

3.1 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, 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 an excellent choice for synchronous machines ranging up-to a few Kilowatts. For higher power ratings the rotor usually consists of windings in which a DC current circulates. The mechanical structure of the rotor is designed for number of poles desired, and the desired flux gradients desired.
- The interaction between the stator and rotor fluxes produces a torque. Since the stator is firmly mounted to the frame, and the rotor is free to rotate, the rotor rotates, producing a useful mechanical output as shown in Figure 3-1.
- The angle between the rotor magnetic field and stator field must be carefully controlled to produce maximum torque and achieve high electromechanical conversion efficiency. For this purpose a fine tuning is needed after closing the speed loop using sensorless algorithm to draw minimum amount of current under the same speed and torque conditions.
- The rotating stator field must rotate at the same frequency as the rotor permanent magnetic field; otherwise the rotor experiences rapidly alternating positive and negative torque. This results 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 stops rotating at the synchronous frequency, and respond to the average torque as seen by the stationary rotor: Zero. This means that the machine experiences a phenomenon known as *pull-out*. This is also the reason why the synchronous machine is not self starting.
- The angle between the rotor field and the stator field must be equal to 90ºC to obtain the highest mutual torque production. This synchronization requires knowing the rotor position to generate the right stator field.
- The stator magnetic field can be made to have any direction and magnitude by combining the contribution of different stator phases to produce the resulting stator flux.

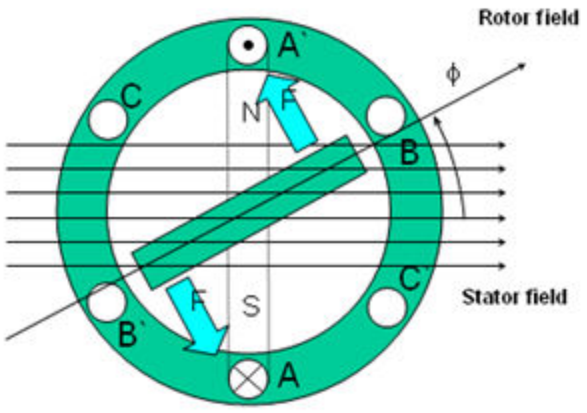


Figure 3-1. Interaction Between the Rotating Stator Flux and the Rotor Flux Produces a Torque

3.1.1 Mathematical Model and FOC Structure of PMSM

The FOC structure for a PMSM is illustrated in Figure 2-1. In this system, the eSMO is used for achieving the sensorless control an IPMSM system, and the eSMO model is designed by utilizing the back EMF model together with a PLL model for estimating the rotor position and speed.

An IPMSM consists of a three-phase stator winding (a, b, c axes), and permanent magnets (PM) rotor for excitation. The motor is controlled by a standard three-phase inverter. An IPMSM can be modeled by using phase a-b-c quantities. Through proper coordinate transformations, the dynamic PMSM models in the d-q rotor reference frame and the α-β stationary reference frame can be obtained. The relationship among these reference frames are illustrated in Equation 1. The dynamic model of a generic PMSM can be written in the d-q rotor reference frame as:

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

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

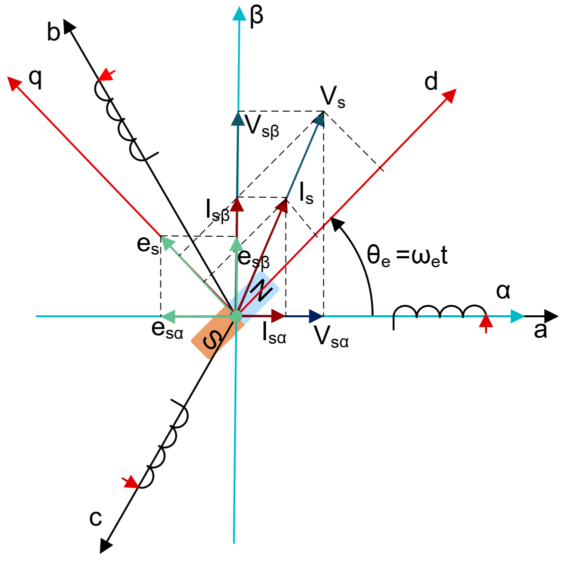


Figure 3-2. Definitions of Coordinate Reference Frames for PMSM Modeling

By using the inverse Park transformation as shown in Figure 3-2, the dynamics of the PMSM can be modeled in the α-β stationary reference frame as:

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

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

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

According to Equation 2 and Equation 3, the rotor position information can be decoupled from the inductance matrix by means of the equivalent transformation and the introduction of the EEMF concept, so that the EEMF is the only term that contains the rotor pole position information. And then the EEMF phase information can be directly used to realize the rotor position observation. Rewrite the IPMSM voltage Equation 4 as a state equation using the stator current as a state variable:

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \frac{1}{L_d}\begin{bmatrix} -R_S & -\omega_e(L_d - L_q) \\ \omega_e(L_d - L_q) & -R_S \end{bmatrix}\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{1}{L_d}\begin{bmatrix} V_\alpha - e_\alpha \\ V_\beta - e_\beta \end{bmatrix} \tag{4}$$

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

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

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

3.1.2 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 3-3. 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.

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 3-3. 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.

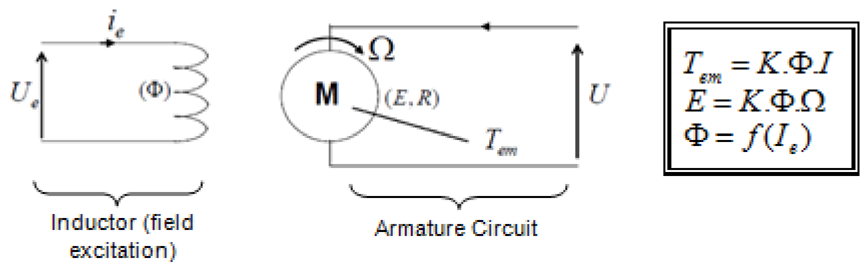


Figure 3-3. Flux and Torque are Independently Controlled in DC Motor Model

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 allows 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, 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 these effects are accounted for, and the overall quality of control is better.

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

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

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

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

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

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

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

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

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

The space vector can be reported in another reference frame with only two orthogonal axis called (α, β). Assuming that the axis a and the axis α*lpha* are in the same direction we have the following vector diagram as shown in Figure 3-4.

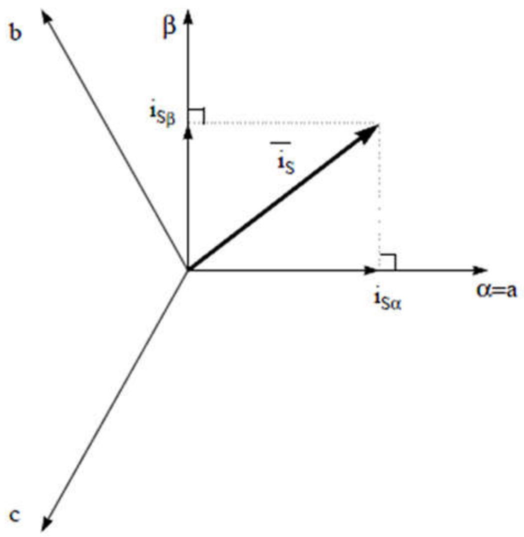


Figure 3-4. Stator Current Space Vector in the Stationary Reference Frame

The projection that modifies the 3-phase system into the (α, β) 2-dimension orthogonal system is presented in Equation 8.

$$i_{s\alpha} = i_a \qquad (8)$$
$$i_{s\beta} = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b$$

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

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

This is the most important transformation in the FOC. In fact, this projection modifies a 2-phase orthogonal system (α, β) in the (d, q) rotating reference frame. If we consider the d axis aligned with the rotor flux, Figure 3-5 shows the relationship for the current vector from the two reference frame.

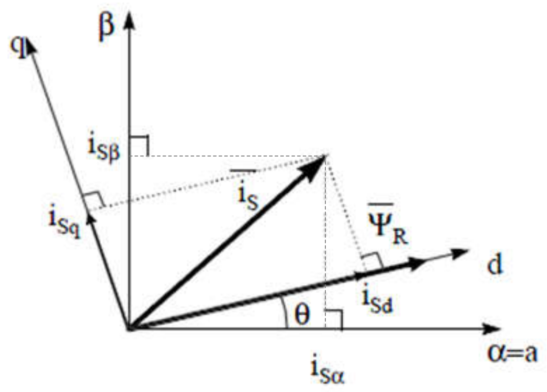


Figure 3-5. Stator Current Space Vector in The d,q Rotating Reference Frame

The flux and torque components of the current vector are determined by Equation 9.

$$i_{sd} = i_{s\alpha}\cos(\theta) + i_{s\beta}\sin(\theta) \qquad (9)$$
$$i_{sq} = -i_{s\alpha}\sin(\theta) + i_{s\beta}\cos(\theta)$$

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.

3.1.2.3 The Basic Scheme of FOC for AC Motor

Figure 3-6 summarizes the basic scheme of torque control with FOC:

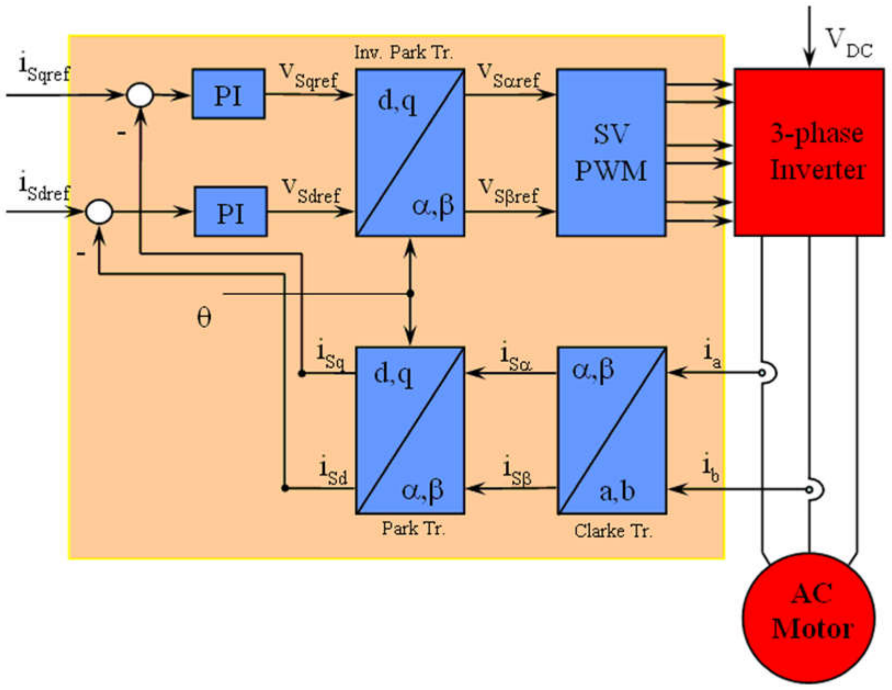


Figure 3-6. Basic Scheme of FOC for AC Motor

Two motor phase currents are measured. These measurements feed the Clarke transformation module. The outputs of this projection are designated $i_{sα}$ and $i_{sβ}$. These two components of the current are the inputs of the Park transformation that gives the current in the d,q rotating reference frame. The i_{sd} and i_{sq} components are compared to the references i_{sdref} (the flux reference component) and i_{sqref} (the torque reference component). At this point, this control structure shows an interesting advantage: the control structure can be used to control either synchronous or induction machines by simply changing the flux reference and obtaining rotor flux position. As in synchronous permanent magnet a motor, the rotor flux is fixed determined by the magnets; there is no need to create one. Hence, when controlling a PMSM, i_{sdref} is 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} can be the output of the speed regulator when we use a speed FOC. The outputs of the current regulators are Vsdref and V_{sqref}; these are applied to the inverse Park transformation. The outputs of this projection are $V_{sαref}$ and $V_{sβref}$ 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).

3.1.2.4 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 3-7 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.

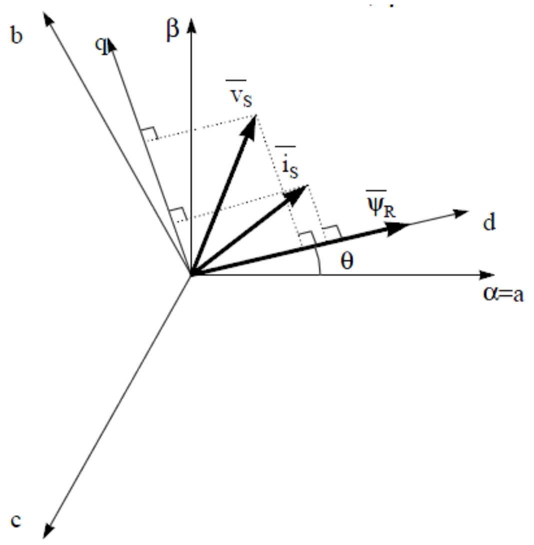


Figure 3-7. Current, Voltage and Rotor Flux Space Vectors in the (d, q) Rotating Reference Frame

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), needing 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 is controlling torque while d-axis current is forced to zero. Therefore, the key module of this system is the estimation of rotor position using enhance Sliding-Mode Observer (eSMO).

3.1.3 Sensorless Control of PM Synchronous Motor

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

3.1.3.1 Enhanced Sliding Mode Observer With Phase Locked Loop

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

3.1.3.1.1 Design of ESMO for PMSM

The conventional PLL integrated into the SMO is shown in Figure 3-8.

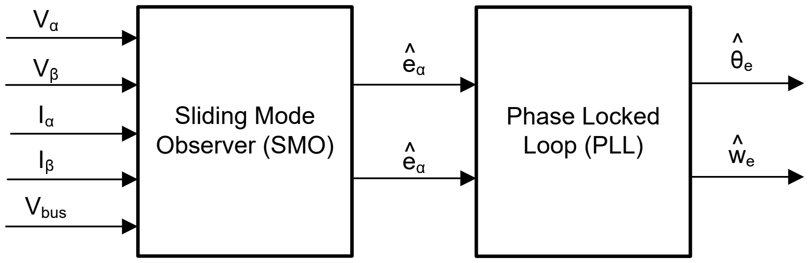


Figure 3-8. Block Diagram of eSMO with PLL for a PMSM

The traditional reduced-order sliding mode observer is constructed, which mathematical model is shown in Equation 10 and the block diagram is shown in Figure 3-9.

$$\begin{bmatrix} \dot{\hat{i}}_\alpha \\ \dot{\hat{i}}_\beta \end{bmatrix} = \frac{1}{L_d}\begin{bmatrix} -R_s & -\hat{\omega}_e(L_d - L_q) \\ \hat{\omega}_e(L_d - L_q) & -R_s \end{bmatrix}\begin{bmatrix} \hat{i}_\alpha \\ \hat{i}_\beta \end{bmatrix} + \frac{1}{L_d}\begin{bmatrix} V_\alpha - \hat{e}_\alpha + z_\alpha \\ V_\beta - \hat{e}_\beta + z_\beta \end{bmatrix} \tag{10}$$

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

$$\begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} = \begin{bmatrix} k_\alpha sign(\hat{i}_\alpha - i_\alpha) \\ k_\beta sign(\hat{i}_\beta - i_\beta) \end{bmatrix} \tag{11}$$

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

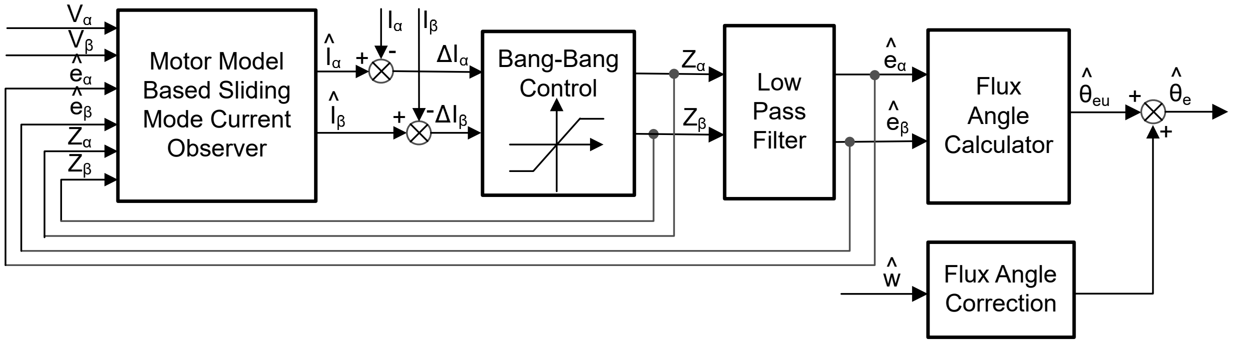


Figure 3-9. Block Diagram of Traditional Sliding Mode Observer

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

$$\begin{bmatrix} \hat{e}_\alpha \\ \hat{e}_\beta \end{bmatrix} = \frac{\omega_c}{s + \omega_c}\begin{bmatrix} z_\alpha \\ z_\beta \end{bmatrix} \tag{12}$$

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

Therefore, the rotor position can be directly calculated from arc-tangent the back EMF, defined as follow

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) \tag{13}$$

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

$$\Delta\theta_e = -\tan^{-1}\left(\frac{\omega_e}{\omega_c}\right) \tag{14}$$

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

$$\hat{\theta}_e = -\tan^{-1}\left(\frac{\hat{e}_\alpha}{\hat{e}_\beta}\right) + \Delta\theta_e \tag{15}$$

In a digital control application, a time discrete equation of the SMO is needed. The Euler method is the appropriate way to transform to a time discrete observer. The time discrete system matrix of Equation 10 in α-β coordinates is given by Equation 16 as:

$$\begin{bmatrix}\hat{i}_\alpha(n+1)\\ \hat{i}_\beta(n+1)\end{bmatrix} = \begin{bmatrix}F_\alpha\\ F_\beta\end{bmatrix}\begin{bmatrix}\hat{i}_\alpha(n)\\ \hat{i}_\beta(n)\end{bmatrix} + \begin{bmatrix}G_\alpha\\ G_\beta\end{bmatrix}\begin{bmatrix}V^*_\alpha(n) - \hat{e}_\alpha(n) + z_\alpha(n)\\ V^*_\beta(n) - \hat{e}_\beta(n) + z_\beta(n)\end{bmatrix} \tag{16}$$

Where the matrix $[F]$ and $[G]$ are given by Equation 17 and Equation 18 as:

$$\begin{bmatrix}F_\alpha\\ F_\beta\end{bmatrix} = \begin{bmatrix}e^{-\frac{R_S}{L_d}}\\ e^{-\frac{R_S}{L_q}}\end{bmatrix} \tag{17}$$

$$\begin{bmatrix}G_\alpha\\ G_\beta\end{bmatrix} = \frac{1}{R_S}\begin{bmatrix}1-e^{-\frac{R_S}{L_d}}\\ 1-e^{-\frac{R_S}{L_q}}\end{bmatrix} \tag{18}$$

The time discrete form of Equation 12 is given by Equation 19 as:

$$\begin{bmatrix}\hat{e}_\alpha(n+1)\\ \hat{e}_\beta(n+1)\end{bmatrix} = \begin{bmatrix}\hat{e}_\alpha(n)\\ \hat{e}_\beta(n)\end{bmatrix} + 2\pi f_c\begin{bmatrix}z_\alpha(n) - \hat{e}_\alpha(n)\\ z_\beta(n) - \hat{e}_\beta(n)\end{bmatrix} \tag{19}$$

3.1.3.1.2 Rotor Position and Speed Estimation with PLL

With the arc tangent method, the accuracy of the position and velocity estimations are affected due to the existence of noise and harmonic components. To eliminate this issue, the PLL model can be used for velocity and position estimations in the sensorless control structure of the IPMSM. The PLL structure used with SMO is illustrated in Section 3.1.3.1.1. The back-EMF estimations $\hat{e}_\alpha$ and $\hat{e}_\beta$ can be used with a PLL model to estimate the motor angular velocity and position as shown in Figure 3-10.

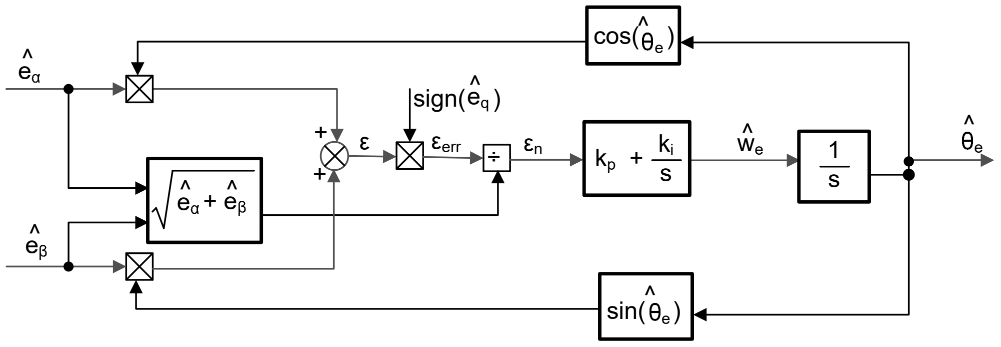


Figure 3-10. Block Diagram of Phase Locked Loop Position Tracker

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

$$\varepsilon = \hat{e}_\beta\cos\left(\hat{\theta}_e\right) - \hat{e}_\alpha\sin\left(\hat{\theta}_e\right) = E\sin(\theta_e)\cos\left(\hat{\theta}_e\right) - E\cos(\theta_e)\sin\left(\hat{\theta}_e\right) = E\sin\left(\theta_e - \hat{\theta}_e\right) \tag{20}$$

Where E is the magnitude of the EEMF, which is proportional to the motor speed ω_e . When $\left(\theta_e - \hat{\theta}_e\right) < \frac{\pi}{2}$, the Equation 20 can be simplified as

$$\varepsilon = E\left(\theta_e - \hat{\theta}_e\right) \tag{21}$$

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

$$\varepsilon_n = \theta_e - \hat{\theta}_e \tag{22}$$

According to the analysis, the simplified block diagram of the quadrature phaselocked loop position tracker can be obtained as shown in Figure 3-11. The closed-loop transfer functions of the PLL can be expressed as:

$$\frac{\hat{\theta}_e}{\theta_e} = \frac{k_p s + k_i}{s^2 + k_p s + k_i} = \frac{2\xi\omega_n s + \omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \tag{23}$$

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

$$k_p = 2\xi\omega_n, \qquad k_i = \omega_n^2 \tag{24}$$

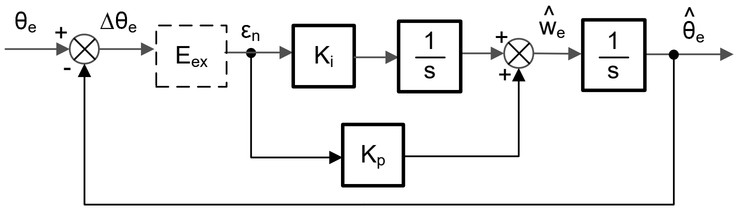


Figure 3-11. Simplified Block Diagram of Phase Locked Loop Position Tracker

3.1.4 Hardware Prerequisites for Motor Drive

The algorithm for controlling the motor makes use of sampled measurements of the motor conditions, including dc bus power supply voltage, the current of each motor phase. There are a few hardware dependent parameters such as current scale value, voltage scale value and voltage filter pole that need to be set correctly to identify the motor properly and run the motor effectively using Field Oriented Control (FOC).

3.1.5 Additional Control Features

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

Permanent magnet synchronous motor (PMSM) is widely used in home appliance applications due to the high power density, high efficiency, and wide speed range. The PMSM includes two major types: the surface-mounted PMSM (SPM), and the interior PMSM (IPM). SPM motors are easier to control due to the linear relationship between the torque and q-axis current. However, the IPMSM has electromagnetic and reluctance torques due to a large saliency ratio. The total torque is non-linear with respect to the rotor angle. As a result, the MTPA technique can be used for IPM motors to optimize torque generation in the constant torque region. The aim of the field weakening control is to optimize to reach the highest power and efficiency of a PMSM drive. Field weakening control can enable a motor operation over the base speed, expanding the operating limits to reach speeds higher than rated speed and allow 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 25 and Equation 26.

$$v_d = L_d\frac{di_d}{dt} + R_s i_d - p\omega_m L_q i_q \tag{25}$$

$$v_q = L_q\frac{di_q}{dt} + R_s i_q + p\omega_m L_d i_d + p\omega_m \psi_m \tag{26}$$

The dynamic equivalent circuit of an IPM synchronous motor is shown in Figure 3-12.

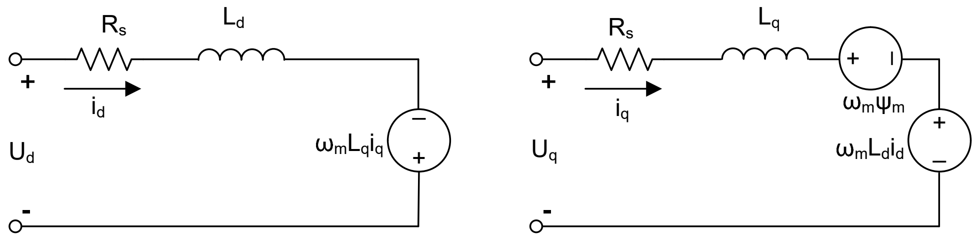


Figure 3-12. Equivalent Circuit of an IPM Synchronous Motor

The total electromagnetic torque generated by the IPMSM can be expressed as Equation 27 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.

$$T_e = \frac{3}{2}p\left[\psi_m i_q + \left(L_d - L_q\right) i_d i_q\right] \tag{27}$$

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 28 and Equation 29.

$$I_a = \sqrt{i_d^2 + i_q^2} \le I_{max} \tag{28}$$

$$V_a = \sqrt{v_d^2 + v_q^2} \le V_{max} \tag{29}$$

Where $\mathrm{V_{max}}$ and $\mathrm{I_{max}}$ 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 30 if Space Vector Modulation (SVPWM) is adopted.

$$\sqrt{v_d^2 + v_q^2} \le v_{max} = \frac{v_{dc}}{\sqrt{3}} \tag{30}$$

Usually the stator resistance R_s is negligible at high speed operation and the derivative of the currents is zero in steady state, thus Equation 31 is obtained as shown.

$$\sqrt{L_d^2\left(i_d + \frac{\psi pm}{L_d}\right)^2 + L_q^2 i_q^2} \le \frac{V_{max}}{\omega_m} \tag{31}$$

The current limitation of Equation 28 produces a circle of radius $\mathrm{I_{max}}$ in the d-q plane, and the voltage limitation of Equation 30 produces an ellipse whose radius $\mathrm{V_{max}}$ 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 3-13.

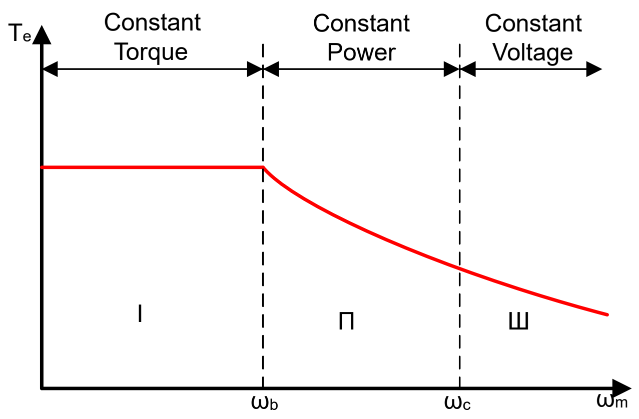


Figure 3-13. IPMSM Control Operation Regions

1. Constant Torque Region: MTPA can be implemented in this operation region to maintain 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 27, 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 utilizes the reluctance torque of the motor, d-axis current must 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 $\mathrm{I_s}$ is shown in the following equations.

$$I_s = \sqrt{i_d^2 + i_q^2} \tag{32}$$

$$I_d = I_s \cos\beta \tag{33}$$

$$I_q = I_s \sin\beta \tag{34}$$

Where β is the stator current angle in the synchronous (d-q) reference frame. Equation 27 can be expressed as Equation 35 where $\mathrm{I_s}$ substituted for i_d and i_q .

Equation 35 shows that motor torque depends on the angle of the stator current vector; as such

$$T_e = \frac{3}{2} p I_s \sin\beta \left[\psi_m + \left(L_d - L_q\right) I_s \cos\beta \right] \tag{35}$$

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, $\frac{\mathrm{dT_e}}{\mathrm{d\beta}}$ is zero as given in Equation 36.

$$\frac{dT_e}{d\beta} = \frac{3}{2} p \left[\psi_m I_s \cos\beta + \left(L_d - L_q\right) I_s^2 \cos 2\beta \right] = 0 \tag{36}$$

Following, the current angle of the MTPA control can be derived as in Equation 37.

$$\beta_{mtpa} = \cos^{-1}\frac{-\psi_m + \sqrt{\psi_m^2 + 8^*\left(L_d - L_q\right)^2 {}^* I_s^2}}{4^*\left(L_d - L_q\right)^* I_s} \tag{37}$$

Thus, the effective d-axis and q-axis reference currents can be expressed by Equation 38 and Equation 39 using the current angle of the MTPA control.

$$I_d = I_s{}^*\cos\beta_{mtpa} \tag{38}$$

$$I_q = I_s{}^*\sin\beta_{mtpa} \tag{39}$$

However, as shown in Equation 37, the angle of the MTPA control, β_{mtpa} is related to d-axis and q-axis inductance. This means that the variation of inductance impedes the ability to find the best MTPA point. To improve the efficiency of a motor drive, the d-axis and q-axis inductance must 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 maintains 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 the d-q currents respectively, as shown in Equation 40 and Equation 41.

$$L_d = f_1\left(i_d, i_q\right) = f_1\left(i_d\right) \tag{40}$$

$$L_q = f_2\left(i_q, i_d\right) = f_2\left(i_q\right) \tag{41}$$

To reduce the ISR calculation burden by simplifying Equation 37. The motor-parameter-based constant, K_{mtpa} is expressed instead as Equation 42, where K_{mtpa} is computed in the background loop using the updated L_d and L_q .

$$K_{mtpa} = \frac{\psi_m}{4^*\left(L_q - L_d\right)} = 0.25^*\frac{\psi_m}{\left(L_q - L_d\right)} \tag{42}$$

$$\beta_{mtpa} = cos^{-1}\left(K_{mtpa}/I_s - \sqrt{\left(K_{mtpa}/I_s\right)^2 + 0.5}\right) \tag{43}$$

A second intermediate variable, G_{mtpa} described in Equation 44, is defined to further simplify the calculation. Using G_{mtpa} , the angle of the MTPA control, β_{mtpa} can be calculated as Equation 45. These two calculations are performed in the ISR to achieve a real current angle β_{mtpa} .

$$G_{mtpa} = K_{mtpa}/I_s \tag{44}$$

$$\beta_{mtpa} = cos^{-1}\left(G_{mtpa} - \sqrt{G_{mtpa}^2 + 0.5}\right) \tag{45}$$

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 28 and Equation 29. 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 31, the IPMSM has two factors: one is a permanent magnet value and the other is made by inductance and current of flux.

Figure 3-14 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 the limit, the input of the PI controller of FW is always positive and therefore the output is always saturated at 0.

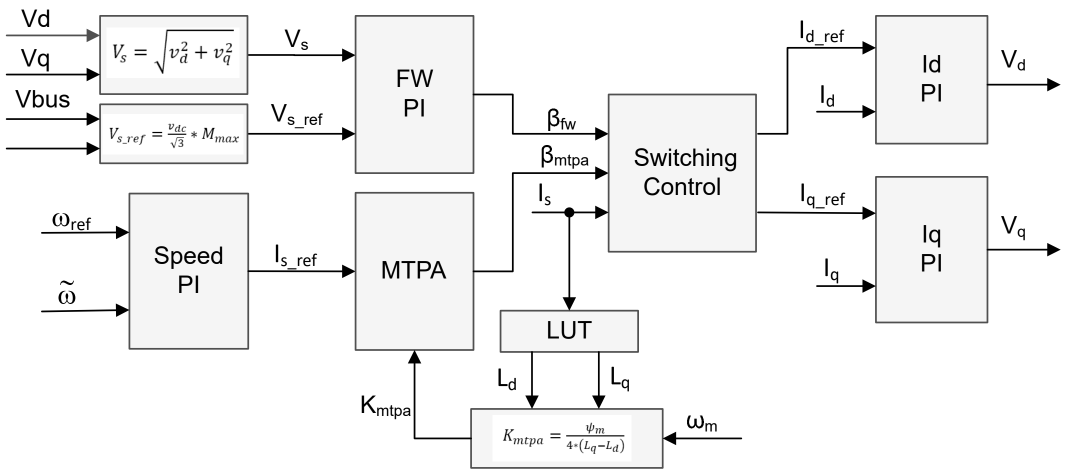


Figure 3-14. Block Diagram of Field-Weakening and Maximum Torque per Ampere Control

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 3-15.

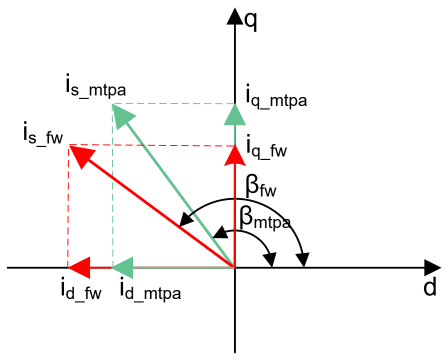


Figure 3-15. Current Phasor Diagram of an IPMSM During FW and MTPA

The switching control module is used to decide which angle can be applied, and then calculate the reference i_d and i_q as shown in Equation 33 and Equation 34. The current angle is chosen as following Equation 46 and Equation 47.

$$\beta = \beta_{fw}\, if\ \beta_{fw} > \beta_{mtpa} \tag{46}$$

$$\beta = \beta_{mpta}\, if\ \beta_{fw} < \beta_{mtpa} \tag{47}$$

Figure 3-16 shows the overall block diagram of sensorless FOC of PMSM using eSMO with field weakening control (FWC) and maximum torque per ampere (MTPA) in this reference design.

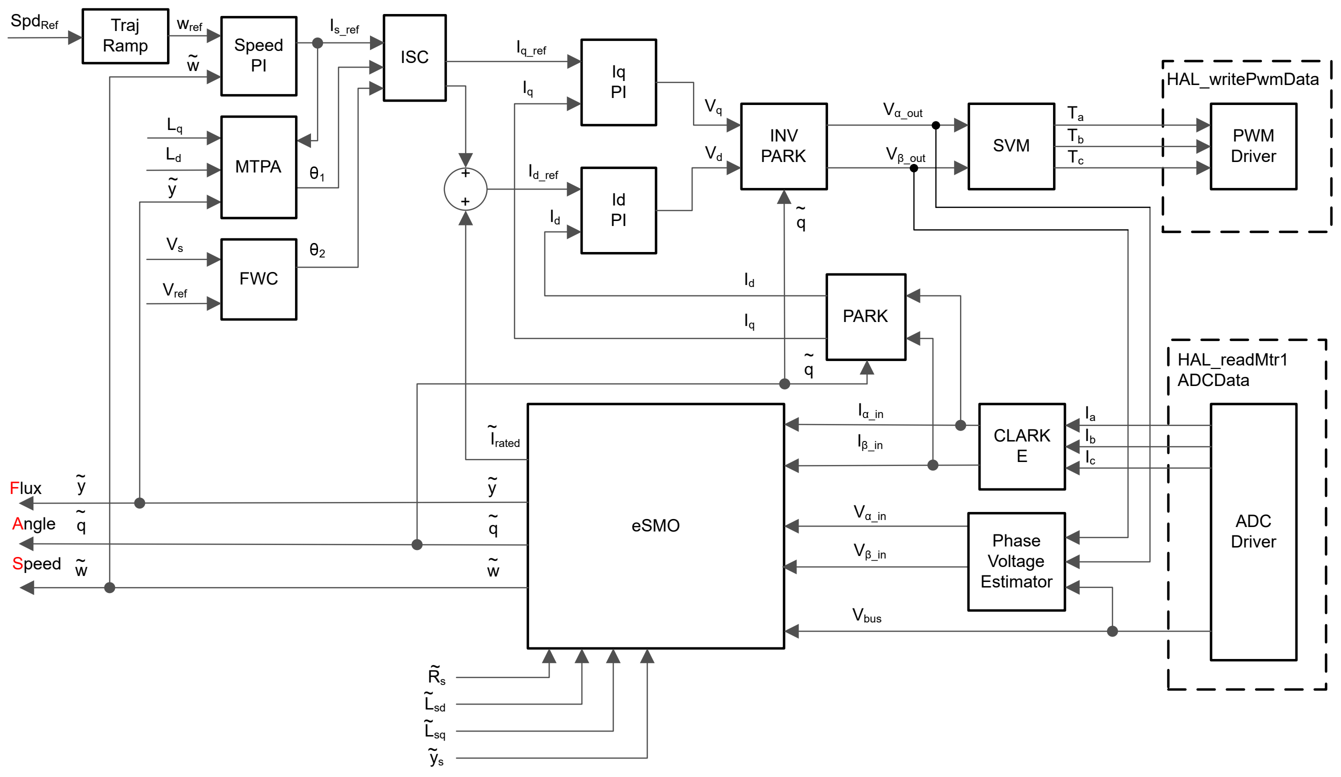


Figure 3-16. Sensorless FOC of PMSM using eSMO with FWC and MTPA

3.1.5.2 Flying Start

Flying start (FS) is a feature that allows the drive to determine the speed and direction of a spinning motor and begin the output voltage and frequency at that speed and direction. Without flying start, the drive begins the output at zero volts and zero speed and attempt to ramp to the commanded speed. If the inertia or direction of rotation of a load requires the motor to produce a large amount of torque, excess current can result and overcurrent trips can occur on the drive. These problems can be eliminated with flying start.

Flying start is the capacity to start control at any speed other than **ZERO**, which is an important function in air-condition application for fan drive.

When a motor is started in the normal mode, the control initially applies a frequency of 0Hz and ramps to the desired frequency. If the drive is started in this mode with the motor already spinning with non-zero frequency, large currents are generated. An over current trip can result if the current limiter cannot react quickly enough. Even if the current limiter is fast enough to prevent an over current trip, synchronization can take an unacceptable amount of time to occur and for the motor to reach the desired frequency. In addition, larger mechanical stress is placed on the application.

In flying start mode, the drive’s response to a start command is to synchronize with the motor’s speed (frequency and phase) and voltage. The motor then accelerates to the commanded frequency. This process prevents an over current trip and significantly reduces the time for the motor to reach the commanded frequency. Because the drive synchronizes with the motor at the rotating speed and ramps to the proper speed, little or no mechanical stress are present.

The flying start function implements an algorithm that searches for the rotor speed. The algorithm searches for a motor voltage that corresponds with the excitation current applied to the motor

When the motor is spinning, the speed and position information can be estimated from the BEMF voltages. Since the stator voltage is measured, the speed and position are easily obtained by switching the inverter. A zero torque current is applied to the motor and the generated current and stator voltage is measured, then FOC module uses these signals to estimate rotor position and speed.

The block diagram of FOC with flying start is shown in Figure 3-17, the flying start module outputs a flag to enable or disable speed close loop control. A zero reference torque current is set and the speed PI controller output is disabled while flying start is operating.

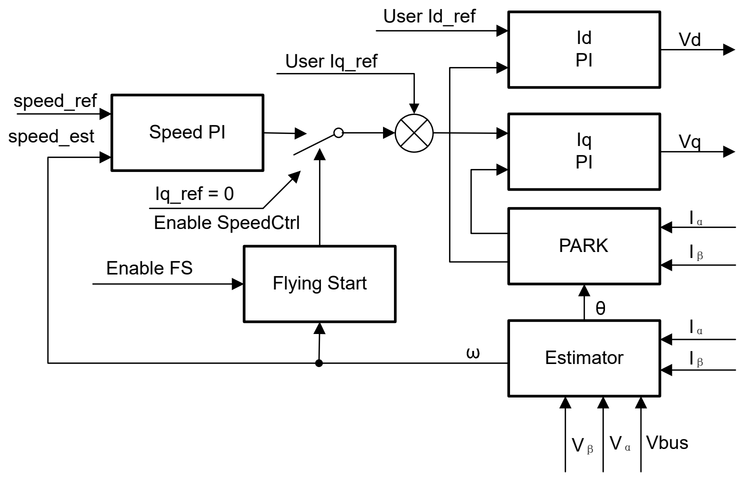


Figure 3-17. Flying Start Control Block Diagram

Figure 3-18 shows the overall block diagram of sensorless FOC of PMSM using eSMO with flying start in this reference design.

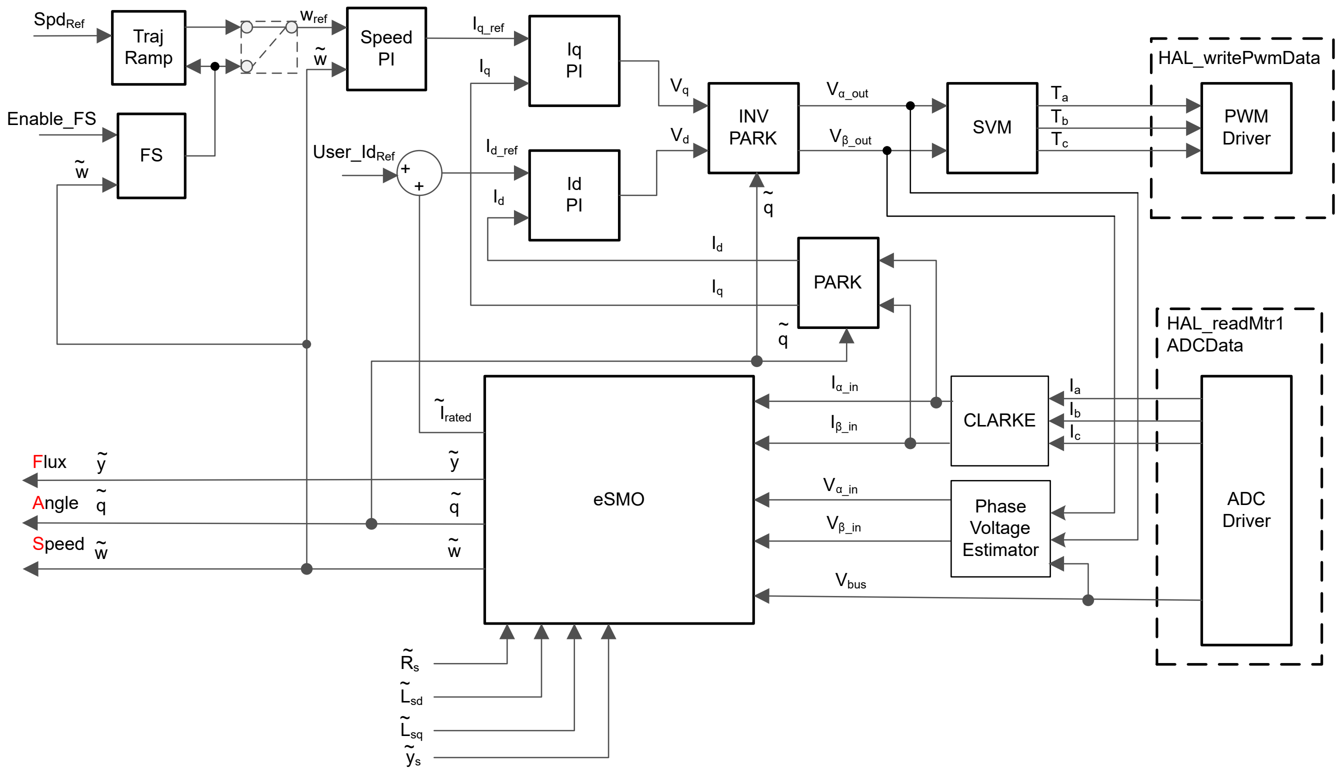


Figure 3-18. Sensorless FOC of PMSM using eSMO with Flying Start

As shown in Figure 3-19, the module routine disables speed close loop control, sets the reference Iq to zero, and enables the FOC module during starting run the motor. After the phase currents and voltages are measured, the routine runs FOC and the real motor speed can be estimated. The program re-enables speed closed loop control and sets the speed reference value after flying start is completed.

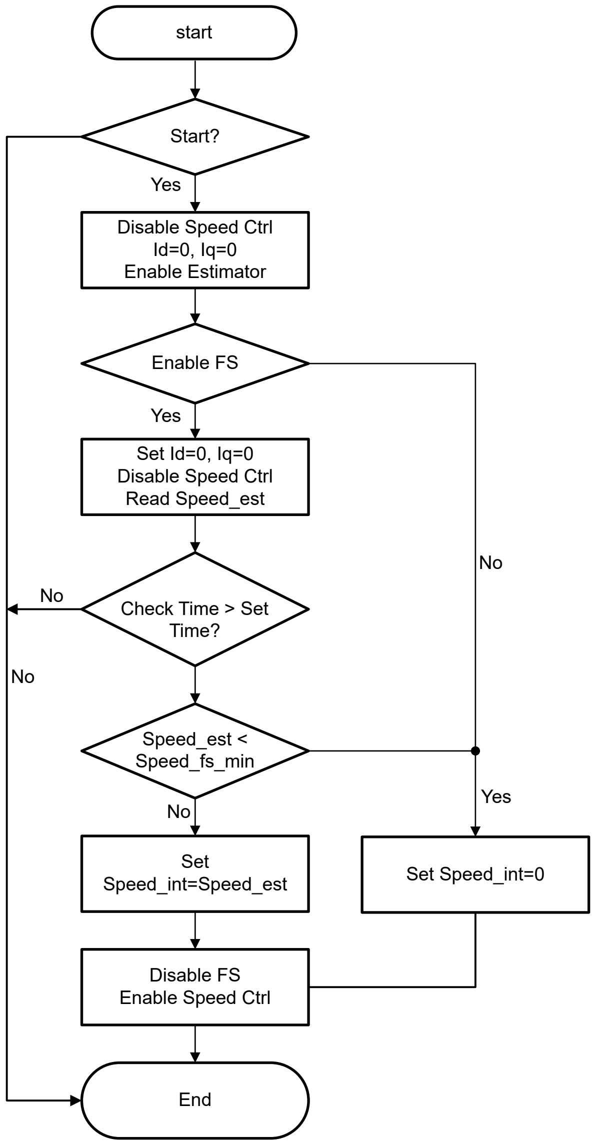


Figure 3-19. Flying Start Module Program Flowchart

4 Hardware, Software, Testing Requirements, and Test Results

4.1 Hardware Requirements

Table 4-1 lists the current evaluation kits that are supported for the universal motor control project.

Table 4-1. Motor Drive Evaluation Kits Supported by Universal Motor Control

Motor Drive Evaluation Board		TI MCU Evaluation Module	Current Sensing Topology	Rotor Position Sensing Method	Tested Motors
Part Number	Description				
BOOSTXL-3PHGAN INV	12-60V, 3.5A 3-ph GaN inverter	LP-AM263	Three shunt-based inline motor phase current sensing	eSMO observer based sensorless-FOC QEP encoder based sensored-FOC Hall sensors based sensored-FOC	LVSERVOMTR (Encoder and Hall Sensor are Embedded)
TMDSHVMTRINSPI N [(1)]	400V, 10A 3ph inverter	TMDSCNCD263 with TMDSADAP180TO100	Three low-side current shunt	eSMO observer based sensorless-FOC QEP encoder based sensored-FOC	HVPMSMMTR (Encoder is Embedded)

(1) If you want to run a low voltage motor like LVSERVOMTR with the high voltage kit, you need to populate a jumper on J1, J2, J3 and J4 to bypass the 820k resistors for phase and dc bus voltages sensing. Also, set the parameters in user_mtr1.h as shown in the following code. The recommendation is to not run a low voltage motor with high current and low inductance on the high voltage kit.

```
// Bypass the 820k resistor for low voltage motor on this kit
#define LV_JUMPER_EN             // Bypass the 820k resistor
```

If the project is set to use Encoder or Hall based sensored-FOC, maintain that the physical connections are connected in the correct order. If the motor, encoder, or hall wires are connected in the wrong order, the project does not function properly, potentially resulting in the motor being unable to spin. For the motor phase wires, verify that the motor phases are connected to the right phase on the inverter board. For the motors that are provided with the TI Motor Control Reference Kits, the correct phase connections are provided as shown in Table 4-2.

For the encoder, verify that A is connected to A, B to B, and I to I. For the Hall sensor, maintain that A is connected to A, B to B, and C to C. Often +5V dc and ground connections are required as well. If you are using Hall sensors or encoders that are different than the ones specifically listed in Table 4-2, please refer to the users manual for the Hall sensor or encoder you are using to verify that you properly connect the wires.

Make sure that for the setup and configuration of the ENC module that the number of slots per rotation for the encoder is provided. This allows the ENC module to correctly convert the encoder signal into an angle. The USER_MOTOR1_NUM_ENC_SLOTS constant that is defined in the *user_mtr1.h* file needs to be updated to the correct value for your encoder. If this value is not correct, the motor spins faster or slower depending on the value that was set. Note that this value is set to the number of slots on the encoder, not the resulting number of counts after figuring the quadrature accuracy.

Table 4-2. Motor Phase, Encoder, or Hall Sensors Connections for Reference Kits and Motors

		LVSERVOMTR	HVPMSMMTR
Motor Phase Lines	U	BLACK (16AWG)	RED
	V	RED (16AWG)	BLUE/BLACK
	W	WHITE (16AWG)	WHITE
Encoder	GND	BLACK (J4-1)	BLACK
	+5V	RED (J4-2)	RED
	I	BROWN (J4-3)	YELLOW
	B	ORANGE (J4-4)	GREEN
	A	BLUE (J4-1)	BLUE

Table 4-2. Motor Phase, Encoder, or Hall Sensors Connections for Reference Kits and Motors (continued)

		LVSERVOMTR	HVPMSMMTR
Hall Sensors	GND	BLACK (J10-1)	Not support for Hall sensor based sensored-FOC
	+5V	RED (J10-2)	
	A	GRAY-WHITE (J10-3)	
	B	GREEN-WHITE (J10-4)	
	C	GREEN (J10-5)	

Get started with TI Real-Time Control Microcontrollers (MCUs) to implement motor control.

- Step 1: Order the desired motor drive evaluation board, TI MCU evaluation module, and motor as shown in Table 4-1.
- Step 2: Download the latest version of MOTOR-CONTROL-SDK-AM263X.
- Step 3: Download the latest version of Code Composer Studio IDE.
- Step 4: Follow the instructions in technical documentation to setup the hardware and run the project described in the following sections.
- Step 5: For answers to any design questions that you have, you can search existing answers or ask your own question using the TI C2000 E2E design support forum.

4.2 Software Requirements

1. Download and install Code Composer Studio from the Code Composer Studio (CCS) Integrated Development Environment (IDE) tools folder. Version 12.6 or above is recommended. More details about CCS installation and implementation in CCS User's Guide.
2. Download and install MOTOR-CONTROL-SDK-AM263X software package from the link provided by TI, install this Motor Control SDK software in the default folder. MOTOR-CONTROL-SDK-AM263X can be installed in one of two ways:
   a. Download the software through the MOTOR-CONTROL-SDK-AM263X download folder.
   b. Go to CCS and under View → Resource Explorer. Under the TI Resource Explorer, go to Arm®-based microcontrollers → MOTOR CONTROL SDK for AM263x, and click on the install button.
3. Once the installation is complete, close CCS, and create a new workspace for importing the project. Follow the steps to build and run this project with different incremental builds as described in the following sections.

4.2.1 Importing and Configuring Project

This project is a universal motor control design that has support for TI EVM motor driver kits and can be used in conjunction with the AM263x MCU devices. The user can run different TI EVM kits by setting the build configurations and properties of the project. In the following sections, the LP-AM263 is used in combination with the BOOSTXL-3PHGANINV lab to show how to import and run the example lab on this kit.

1. Import the project within CCS by clicking "Project" ➔"Import CCS Projects...", and then click "Browse..." button to select search directory at:
   a. `<install_location>\examples\` to select the "universal_motorcontrol_lab" folder.
2. The project can be configured to run on two motor driver kits. You can select one of these kits by right-clicking on the imported project name and selecting the right build configuration (such as **3phGaN_3SC**) as shown in Figure 4-1.
3. Configure the project to select the supporting functions in the project by right-clicking on the imported project name, and then click the "Properties" command to set the pre-define symbols for the project as shown in Figure 4-2.
   a. A pre-define symbol is active or disabled by removing or adding the "_N" in the name. For example, field weakening control is enabled by removing the "_N" in "MOTOR1_FWC_N" to change to "MOTOR1_FWC", and field weakening control functions are disabled by changing the "MOTOR1_FWC" symbol name to "MOTOR1_FWC_N".
   b. Select the right supporting motor control algorithm based on the motor and hardware board by enabling the related pre-define symbol as described above. The supporting algorithms and related motors matrix are shown in Table 4-3.
   c. Select the right supporting functions by enabling the pre-define symbol/s as shown in Figure 4-2.
4. Select the right target configuration file (.ccxml) as shown in Figure 4-4 by right clicking on the file name to select "Set as Active Target Configuration" and "Set as Default Target Configuration" on the pop-up menu.
   a. AM263_LP.ccxml is for the LP-AM263 based hardware kit.
   b. AM263_CC.ccxml is for the TMDSCNCD263 based hardware kit.
5. Select or define the right motor model in the *user_mtr1.h* and *user_common.h* files. These files are located under the src_board folder located in the project explorer window. Uncomment the #define that corresponds with the motor being tested, and verify that the rest of the #define motors remain commented out. Make sure that the motor parameters in the code match with the specifications of the connecting motor.
6. Set up the hardware kit, connect the motor, encoder, and/or hall sensor to the kit as described in Section 4.3.

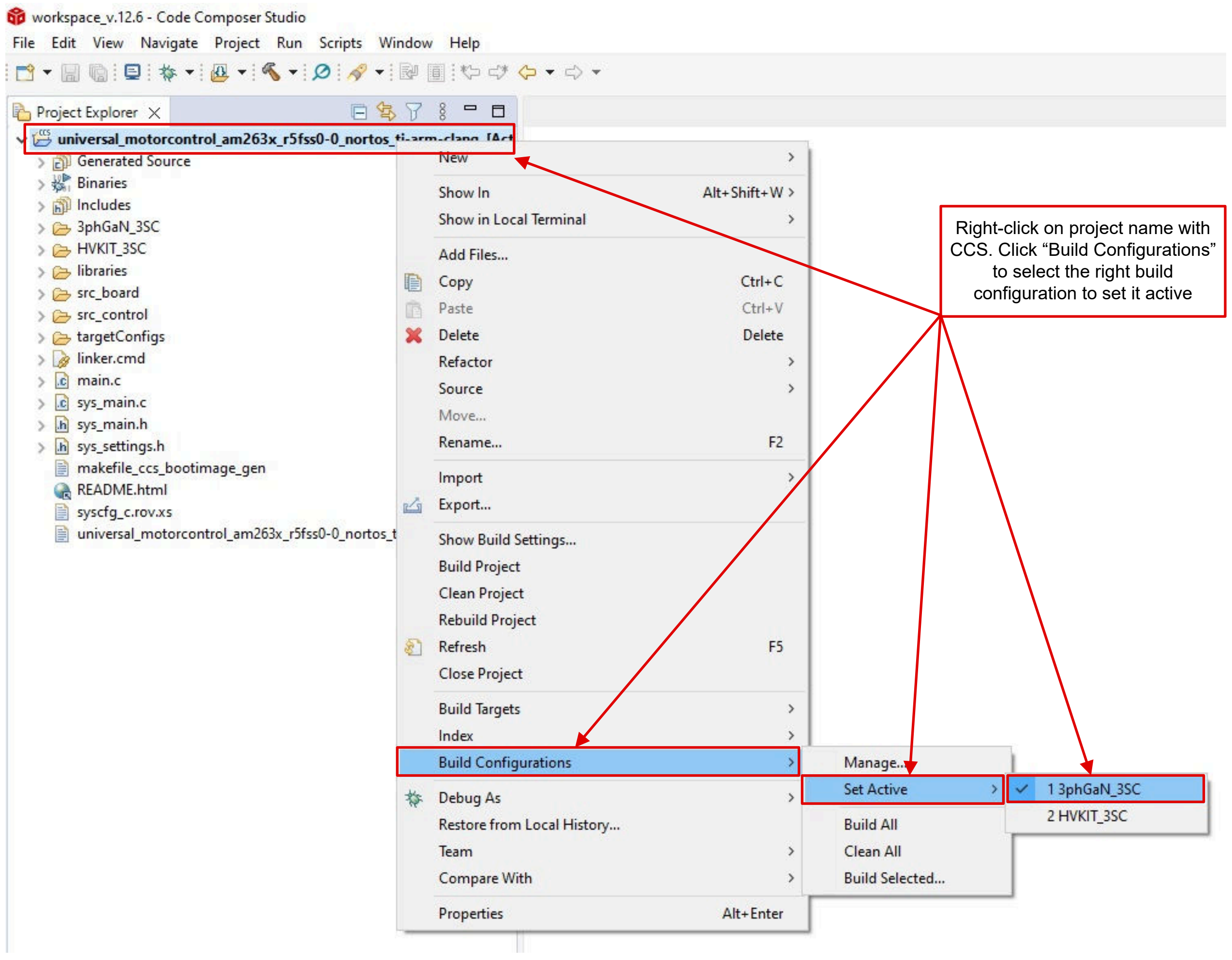


Figure 4-1. Select the Right Build Configurations within CCS

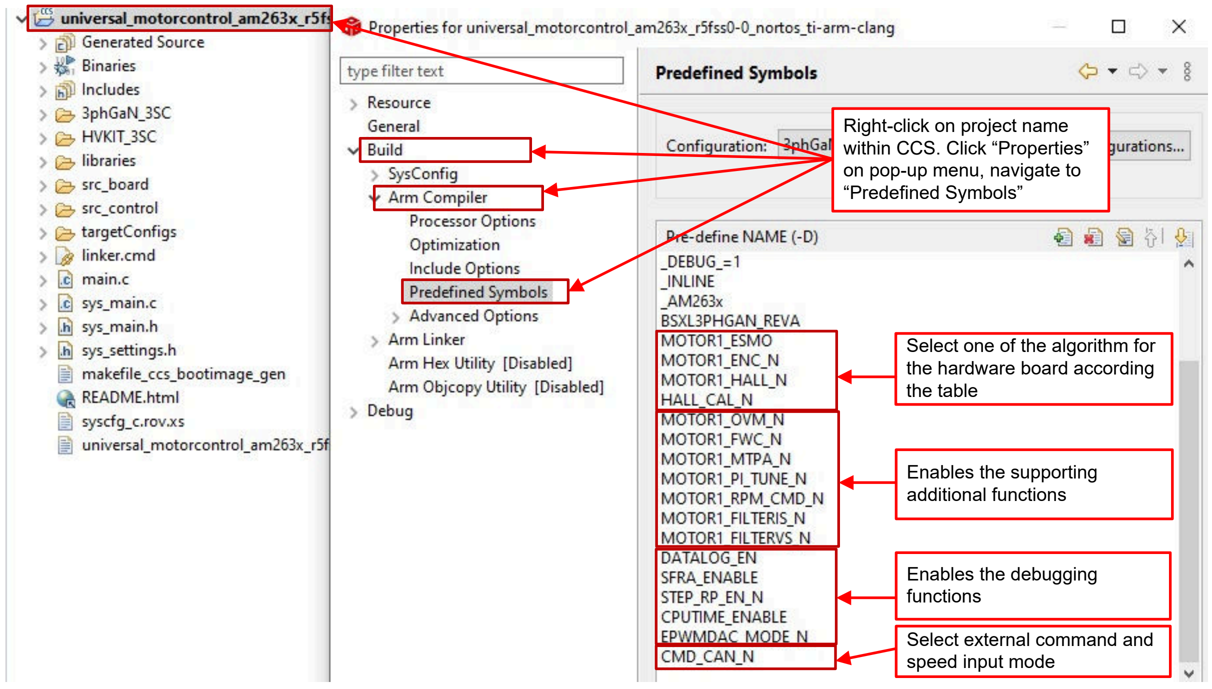


Figure 4-2. Select the Desired Pre-define Symbols in Project Properties

Table 4-3. The Supporting Algorithms, Functions and Motors Matrix in Universal Motor Control

Algorithms or Functions	Pre-Define Symbols	LaunchPad	controlCARD
		BOOSTXL-3PHGANINV	TMDSHVMTRINSPIN
eSMO based Sensorless FOC	MOTOR1_ESMO	✓, LVSERVOMTR	✓, HVPMSMMTR
QEP Encoder based Sensored FOC	MOTOR1_ENC	✓, LVSERVOMTR	✓, HVPMSMMTR
Hall Sensors based Sensored FOC	MOTOR1_HALL HALL_CAL	✓, LVSERVOMTR	✖
Datalog with Graph Tool	DATALOG_EN	✓	✓
PWMDAC	EPWMDAC_MODE	✖	✓
SFRA Tool	SFRA_ENABLE	✓	✓
Step Response with Graph Tool	STEP_RP_EN	✓	✓

4.2.2 Project Structure

The general structure of the project is shown in Figure 4-3. The device peripherals configuration is based on *TI SysConfig*. The user only needs to change the code and definitions in the *hal.c* and *hal.h* files, and the parameters in the *user_mtr1.h* file, if the user wants to migrate the reference design software to a custom board or to a different device.

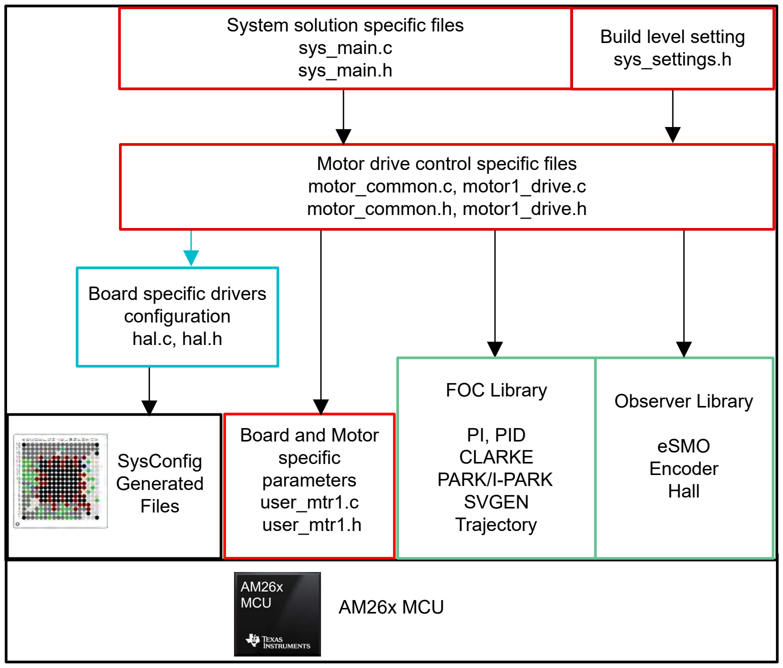

System solution specific files
sys_main.c
sys_main.h
Build level setting
sys_settings.h
Motor drive control specific files
motor_common.c, motor1_drive.c
motor_common.h, motor1_drive.h
Board specific drivers configuration
hal.c, hal.h
SysConfig Generated Files
Board and Motor specific parameters
user_mtr1.c
user_mtr1.h
FOC Library
PI, PID
CLARKE
PARK/I-PARK
SVGEN
Trajectory
Observer Library
eSMO
Encoder
Hall
AM26x MCU
AM26x MCU

Figure 4-3. Project Structure Overview

Once the project is imported into CCS, the project explorer appears inside CCS as shown in Figure 4-4.

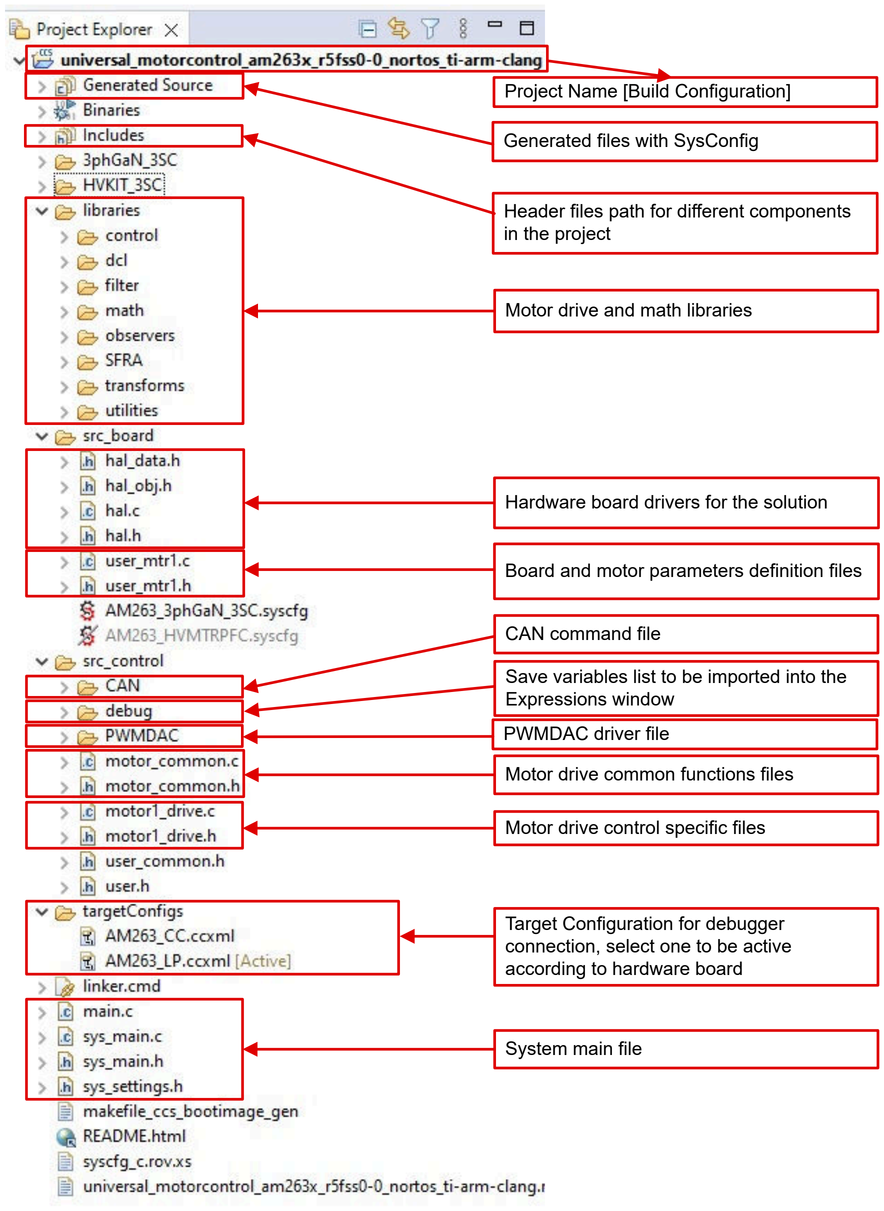
The *transforms* folder includes the typical FOC modules including Park, Clark, and inverse Park and and SVGEN that are part of the motor drive ISR and are independent of specific devices or boards.

The *libraries* folder includes the estimator library and other libraries that are not specific to any particular device or board.

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

The *src_sys* folder includes some files reserved for system control that are independent of specific devices or boards. The user can add code for system control, communication, and so forth.

Board-specific and motor-specific files are in the *src_board* folder. These files consist of device specific drivers to run the design. If the user wants to migrate the project for their own board or to other devices, the user only needs to make changes to the *hal.c, hal.h, xxx.syscfg* and *user_mtr1.h* files based on the usage of device peripherals for the board.



Project Explorer

- universal_motorcontrol_am263x_r5fss0-0_nortos_ti-arm-clang
  - Generated Source
  - Binaries
  - Includes
  - 3phGaN_3SC
  - HVKIT_3SC
  - libraries
    - control
    - dcl
    - filter
    - math
    - observers
    - SFRA
    - transforms
    - utilities
  - src_board
    - hal_data.h
    - hal_obj.h
    - hal.c
    - hal.h
    - user_mtr1.c
    - user_mtr1.h
    - AM263_3phGaN_3SC.syscfg
    - AM263_HVMTRPFC.syscfg
  - src_control
    - CAN
    - debug
    - PWMDAC
    - motor_common.c
    - motor_common.h
    - motor1_drive.c
    - motor1_drive.h
    - user_common.h
    - user.h
  - targetConfigs
    - AM263_CC.ccxml
    - AM263_LP.ccxml [Active]
  - linker.cmd
  - main.c
  - sys_main.c
  - sys_main.h
  - sys_settings.h
  - makefile_ccs_bootimage_gen
  - README.html
  - syscfg_c.rov.xs
  - universal_motorcontrol_am263x_r5fss0-0_nortos_ti-arm-clang.r

Project Name [Build Configuration]
Generated files with SysConfig
Header files path for different components in the project
Motor drive and math libraries
Hardware board drivers for the solution
Board and motor parameters definition files
CAN command file
Save variables list to be imported into the Expressions window
PWMDAC driver file
Motor drive common functions files
Motor drive control specific files
Target Configuration for debugger connection, select one to be active according to hardware board
System main file

Figure 4-4. Project Explorer View of the Universal Motor Control Project

4.2.3 Lab Software Overview

Figure 4-5 shows the project software flow diagram of the firmware that includes one ISR for real time motor control and the main loop for motor control parameters that updates in a background loop. The ISR is triggered by ADC End of Conversion (EOC).

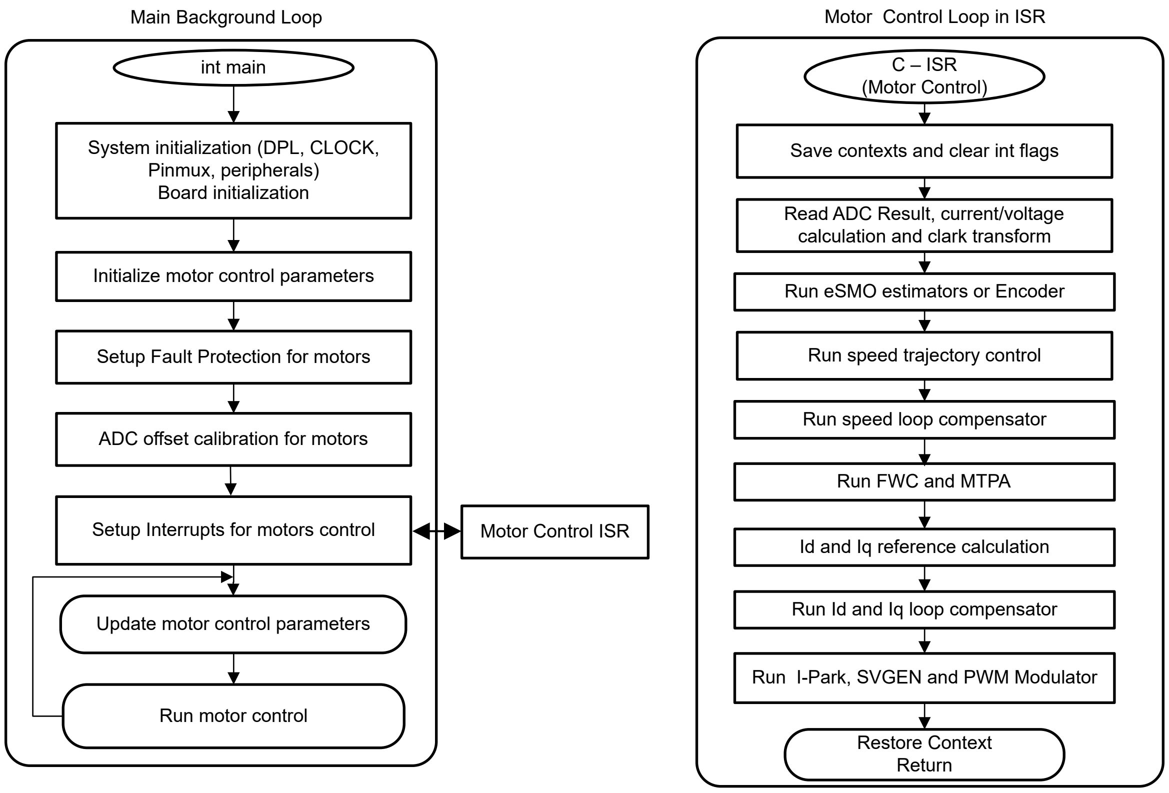

Main Background Loop
int main
System initialization (DPL, CLOCK, Pinmux, peripherals) Board initialization
Initialize motor control parameters
Setup Fault Protection for motors
ADC offset calibration for motors
Setup Interrupts for motors control
Motor Control ISR
Update motor control parameters
Run motor control
Motor Control Loop in ISR
C – ISR (Motor Control)
Save contexts and clear int flags
Read ADC Result, current/voltage calculation and clark transform
Run eSMO estimators or Encoder
Run speed trajectory control
Run speed loop compensator
Run FWC and MTPA
Id and Iq reference calculation
Run Id and Iq loop compensator
Run I-Park, SVGEN and PWM Modulator
Restore Context Return

Figure 4-5. Project Software Flowchart Diagram

To simplify the system bring up and design, the software is organized in four incremental builds, which makes learning and getting familiar with the board and software easier. This approach is also good for debugging and testing boards.

Table 4-4 lists the framework modules to be used in this project.

Table 4-4. Using Motor Control Modules in Project

Module Names	Explanation	Algorithm
ANGLE_GEN_run	Ramp angle generator for open-loop running	eSMO, ENC, HALL
CLARKE_run	Clarke transformation for current or voltage	eSMO, ENC, HALL
collectRMSData, calculateRMSData	Collect sampling values to calculate the RMS value of phase current and voltage	eSMO, ENC, HALL
DATALOG_update	Stores the real-time values into for displaying with graph tool	All Algorithms
ENC_run	Calculate rotor angle based on encoder	ENC
ESMO_run	Enhance Sliding Mode Observer (eSMO) for sensorless-FOC	eSMO
HAL_readMtr1ADCData	Returns ADC conversion values with floating-point format	All Algorithms
HAL_writePWMDACData	Converts software variables into the PWM signals	All Algorithms
HAL_writePWMData	PWM drives for motor	All Algorithms
HALL_run	Calculate rotor angle ans speed based on Hall sensors	HALL
IPARK_run	Inverse Park transformation	eSMO, ENC, HALL
PARK_run	Park Transformation	eSMO, ENC, HALL
PI_run	PI Regulators for current and speed	All Algorithms
PI_run_series	Runs the series form of the PI controller	SFRA, MPTA
SPDCALC_run	Speed Measurement based on the angle from encoder signal	ENC
SPDFR_run	Speed measurement based on the angle from observer	eSMO
SVGEN_runMin	Space Vector PWM with quadrature control	eSMO, ENC, HALL
TRAJ_run	Trajectory for setting speed reference	All Algorithms
VS_FREQ_run	Generate vector voltage with v/f profile for Vd and Vq calculation in level 2. This can be done manually based on specific motors.	eSMO, ENC, HALL

Table 4-5 summarizes the modules tested in each incremental system build.

Table 4-5. Motor Control Modules Used per Incremental Build

Software Module	DMC_LEVEL_1	DMC_LEVEL_2	DMC_LEVEL_3	DMC_LEVEL_4
	50% PWM duty, verify ADC offset calibration, PWM output, and phase shift	Open loop control to verify the motor current and voltage sensing signals	Closed current loop to validate the current sensing on the board and the current control with the PID	Closed-loop run with estimators/observers
HAL_readMtr1ADCData	√√	√	√	√
HAL_writePWMData	√√	√	√	√
ANGLE_GEN_run		√√	√	√(eSMO, ENC, HALL)*
VS_FREQ_run		√√		
CLARKE_run		√	√	√
TRAJ_run		√√	√	√√
ESMO_run		√(eSMO)*	√(eSMO)*	√√ (eSMO)*
SPDFR_run		√(eSMO)*	√(eSMO)*	√√ (eSMO)*
ENC_run		√(ENC)*	√(ENC)*	√√(ENC)*
SPDCALC_run		√(ENC)*	√(ENC)*	√√(ENC)*
HALL_run		√(HALL)*	√(HALL)*	√√(HALL)*
PARK_run		√	√	√
PI_run (Id)			√√	√
PI_run (Iq)			√√	√
PI_run (speed)				√√
IPARK_run		√√	√	√
SVGEN_runMin		√√	√	√
HAL_writePWMDACData		√**	√**	√**
DATALOG_update		√	√	√

1. √ means this module is used. √√ means this module is being tested.
2. √ (eSMO)* means this module is only used by eSMO. √ (ENC)* means this module is only used by ENC. √ (HALL)* means this module is only used by HALL.
3. √** means this module is supported by some hardware kit as shown in Table 4-1.

The universal project can use one of the FOC algorithms separately for motor control, or use two of the eSMO and encoder FOC algorithms simultaneously. The estimator in use can be switched smoothly on the fly if two algorithms are implemented in the project.

4.3 Test Setup

This section describes how to set up hardware boards for motor control when combining the motor driver evaluation board with the TI development tools. The following sections show the detailed operation procedure on different motor driver evaluation boards.

4.3.1 LP-AM263 Setup

LP-AM263 is a low-cost development board for the TI Arm®-based real-time microcontrollers. This LaunchPad kit offers extra pins for development and supports the connection of two BoosterPack™ plug-in modules.

- For more details about the LP-AM263 , see the LP-AM263 LaunchPad User's Guide.
- Make sure that the boot switch on the LP-AM263 are set as shown in Figure 4-6.
  - For **QSPI_D0 (SOP0)**, position the switch to the **LEFT** (Logic High).
  - For **QSPI_D1 (SOP1)**, position the switch to the **LEFT** (Logic High).
  - For **SPI0_CLK_pad (SOP2)**, position the switch to the **Right** (Logic Low).
  - For **SPI0_D0_pad (SOP3)**, position the switch to the **LEFT** (Logic High).
- Make sure that the DAC VREF Switch (S1) on the LP-AM263 is set on AM263x on-die LDO for correct CMPSS Operation.

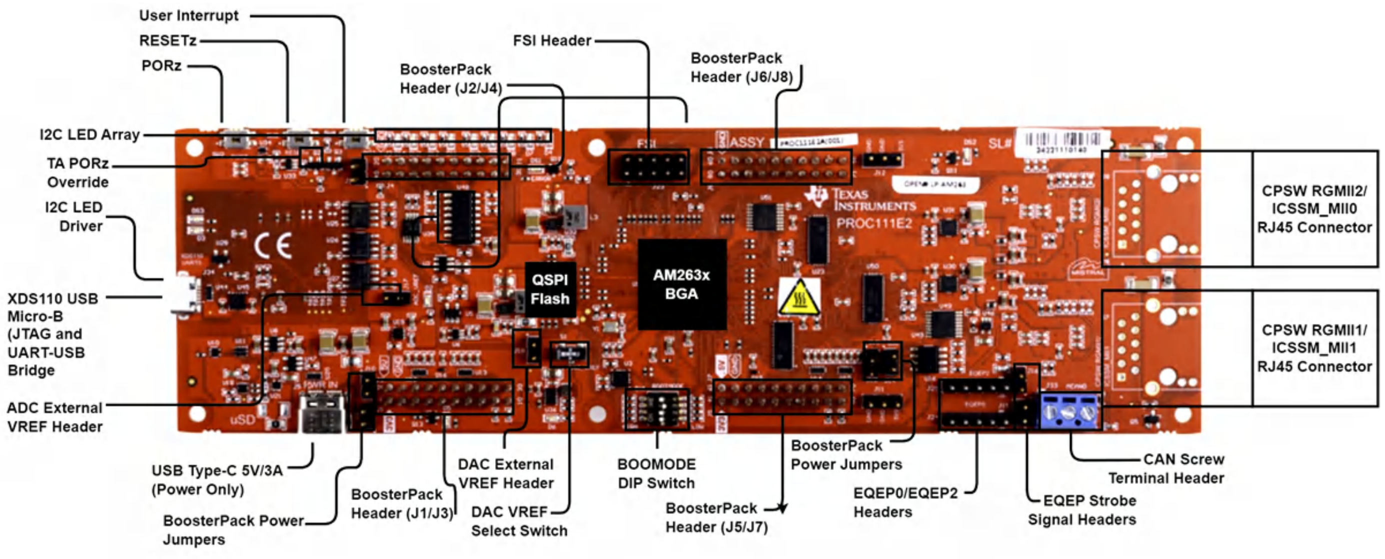

User Interrupt
RESETz
PORz
FSI Header
BoosterPack Header (J2/J4)
BoosterPack Header (J6/J8)
I2C LED Array
TA PORz Override
I2C LED Driver
XDS110 USB Micro-B (JTAG and UART-USB Bridge
ADC External VREF Header
QSPI Flash
AM263x BGA
CPSW RGMII2/ ICSSM_MII0 RJ45 Connector
CPSW RGMII1/ ICSSM_MII1 RJ45 Connector
USB Type-C 5V/3A (Power Only)
BoosterPack Power Jumpers
BoosterPack Header (J1/J3)
DAC External VREF Header
DAC VREF Select Switch
BOOMODE DIP Switch
BoosterPack Header (J5/J7)
BoosterPack Power Jumpers
EQEP0/EQEP2 Headers
EQEP Strobe Signal Headers
CAN Screw Terminal Header

Figure 4-6. LP-AM263 LaunchPad™ Board Overview and Switches Setting

4.3.2 BOOSTXL-3PHGANINV Setup

The BOOSTXL-3PHGANINV evaluation module features a 48V/10A three-phase GaN inverter with precision in-line shunt-based phase current sensing for accurate control of precision drives such as servo drives. The module also has an individual DC bus and three-phase voltage sensing, making this board for BLDC/PMSM control with TI LaunchPad™ development kits designed for use with the sensorless FOC algorithm.

- The hardware files and more details are available on the BOOSTXL-3PHGANINV page within ti.com.
- For more details about the BOOSTXL-3PHGANINV, see the corresponding User's Guide.
- Make sure that the following items are completed as described, and then connect the BOOSTXL-3PHGANINV to J6/J8 and J5/J7 of the LP-AM263 as shown in Figure 4-7.
- Connect the motor, encoder, and Hall sensors to the BOOSTXL-3PHGANINV and the LP-AM263 as described in Table 4-2 and shown in Figure 4-7.
- Connect a supply voltage 24V from a battery or a DC voltage source to the voltage supply pins. Follow the operation instructions in Section 4.4 to turn on the power source.

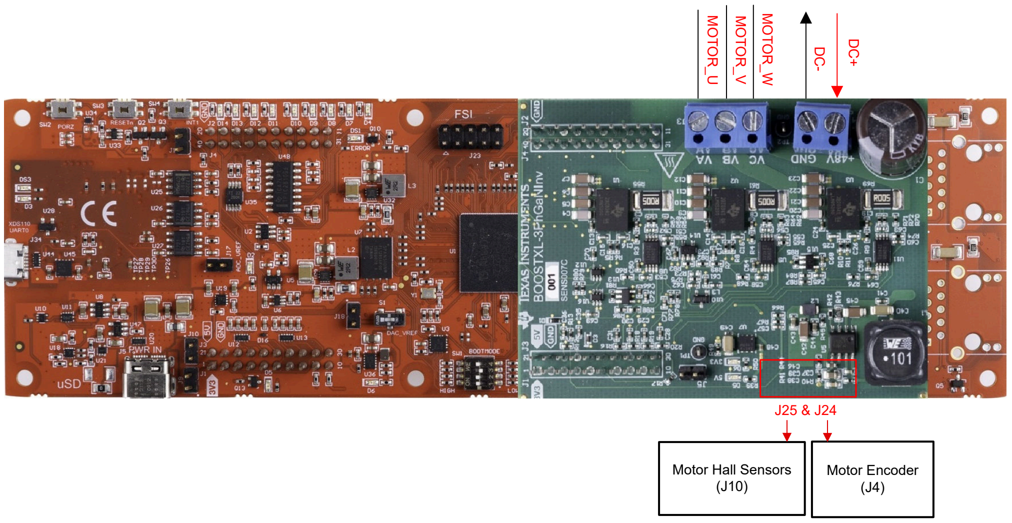


Figure 4-7. LP-AM263 Connected to the BOOSTXL-3PHGANINV

4.3.3 TMDSCNCD263 Setup

TMDSCNCD263 is an evaluation and development board for TI Arm® based MCU series of AM26x devices. TMDSCNCD263 comes with a HSEC180 (180-pin High Speed Edge Connector) and can be used on existing 100-Pin DIMM based TMDSHVMTRINSPIN with TMDSADAP180TO100 adapter

- For more details about on TMDSCNCD263 , see AM263x Sitara Control Card Hardware User's Guide.
- Make sure that boot switches on TMDSCNCD263 are set as described in Figure 4-8.
  - For SW3.1, SW3.2 and SW3.4 position switches to the **LEFT**, and SW3.2 to the**Right** for using the on-Card XDS110 emulator in Quad Read UART Fallback Mode
- Make sure that the DAC VREF Switch (SW6) on the TMDSCNCD263 is switched to **UP** to set reference voltage on AM263x on-die LDO for right CMPSS Operation.

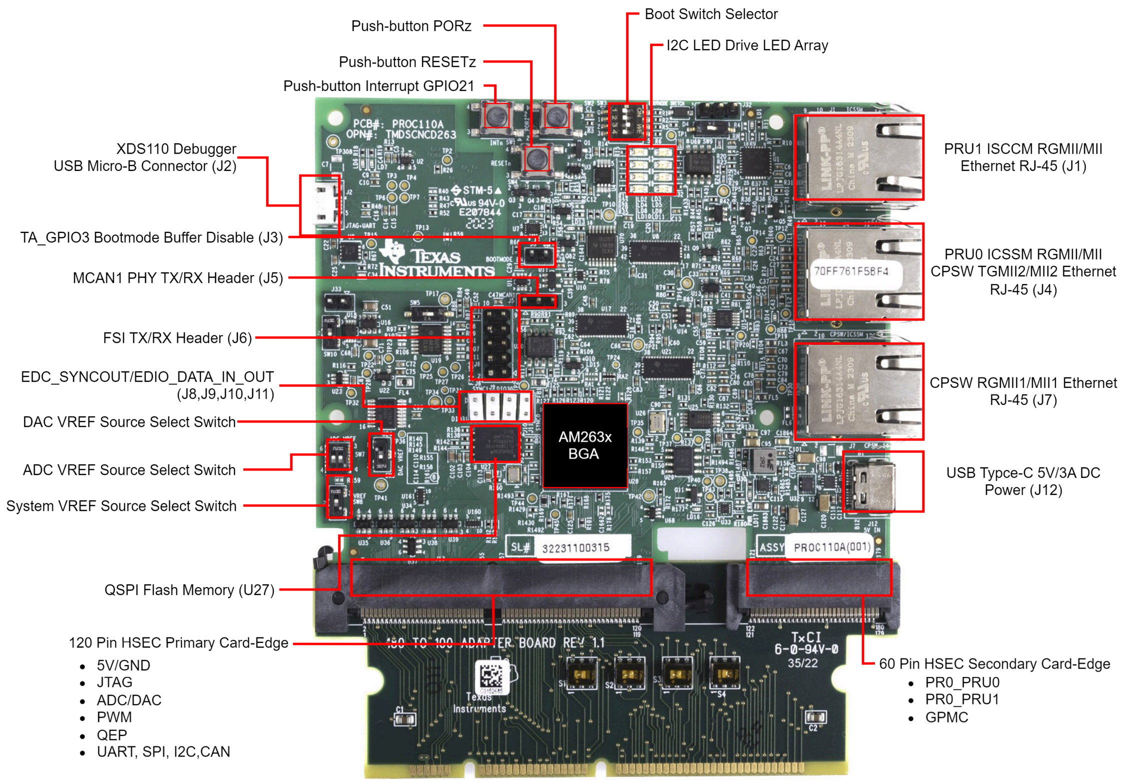

Push-button PORz
Push-button RESETz
Push-button Interrupt GPIO21
Boot Switch Selector
I2C LED Drive LED Array
XDS110 Debugger USB Micro-B Connector (J2)
TA_GPIO3 Bootmode Buffer Disable (J3)
MCAN1 PHY TX/RX Header (J5)
FSI TX/RX Header (J6)
EDC_SYNCOUT/EDIO_DATA_IN_OUT (J8,J9,J10,J11)
DAC VREF Source Select Switch
ADC VREF Source Select Switch
System VREF Source Select Switch
QSPI Flash Memory (U27)
120 Pin HSEC Primary Card-Edge
5V/GND
JTAG
ADC/DAC
PWM
QEP
UART, SPI, I2C,CAN
PRU1 ISCCM RGMII/MII Ethernet RJ-45 (J1)
PRU0 ICSSM RGMII/MII CPSW TGMII2/MII2 Ethernet RJ-45 (J4)
CPSW RGMII1/MII1 Ethernet RJ-45 (J7)
USB Typce-C 5V/3A DC Power (J12)
AM263x BGA
60 Pin HSEC Secondary Card-Edge
PR0_PRU0
PR0_PRU1
GPMC

Figure 4-8. TMDSCNCD263 controlCARD and Switches Setting

4.3.4 TMDSADAP180TO100 Setup

The TMDSADAP180TO100 adapter allows the use of 180-Pin TI controlCARDs with existing 100-Pin DIMM based evaluation tools. The TMDSCNCD263 controlCARD needs TMDSADAP180TO100 to be used on TMDSHVMTRINSPIN.

- Hardware files are located in the `<install_location>\boards\controlCARDs\TMDSADAP180TO100` folder of C2000Ware.
- Make sure that switches TMDSADAP180TO100 are set as described or shown in Figure 4-9.
  - The S1, S2 and S3 switches need to be positioned to the **RIGHT**, and S4 switch needs to be positioned to the **LEFT**.

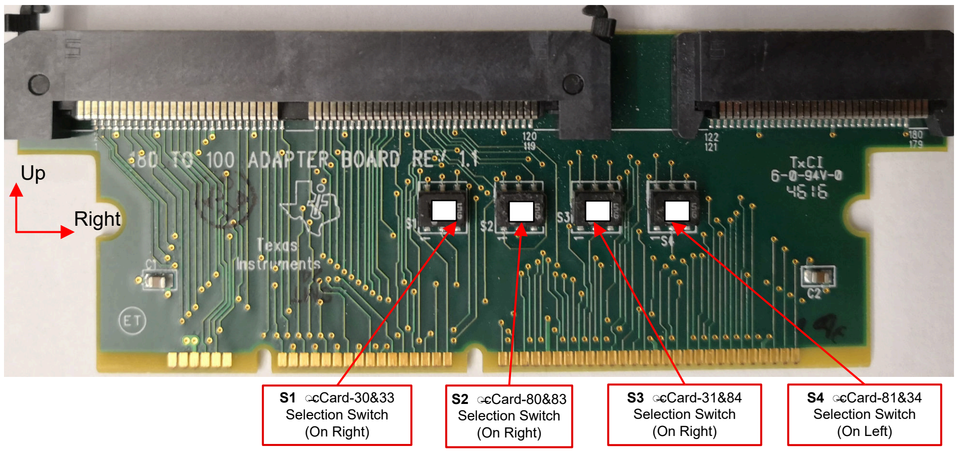

Up
Right
S1 cCard-30&33 Selection Switch (On Right)
S2 cCard-80&83 Selection Switch (On Right)
S3 cCard-31&84 Selection Switch (On Right)
S4 cCard-81&34 Selection Switch (On Left)

Figure 4-9. TMDSADAP180TO100 Adapter and Switches Setting

4.3.5 TMDSHVMTRINSPIN Setup

WARNING

- This EVM is meant to be operated in a lab environment only and is not considered by TI to be a finished end-product fit for general consumer use.
- This EVM must be used only by qualified engineers and technicians familiar with risks associated with handling high voltage electrical and mechanical components, systems and subsystems.
- This EVM operates at voltages and currents that can result in electrical shock, fire hazard and/or personal injury if not properly handled. Equipment must be used with necessary caution and appropriate safeguards must be employed to avoid personal injury or property damage.
- Always use caution when using the EVM electronics due to presence of high voltages. DC bus Capacitors remain charged for a long time after the mains supply is disconnected.
- The EVM can accept power from the AC Mains/wall power supply, only uses the live and neutral line from the wall supply, the protective earth is unconnected (floating). The power ground is floating from the protective earth ground, all of the ground planes are the same. Hence appropriate caution must be taken and proper isolation requirements must be met before connecting scopes and other test equipment to the board. Isolation transformers must be used when connecting grounded equipment to the EVM.
- The power stages on the board are individually rated. The user is responsible to make sure that these ratings (for example, voltage, current and power levels) are well understood and complied with, prior to connecting these power blocks together and energizing the board. When energized, the EVM or components connected to the EVM must not be touched.

TMDSHVMTRINSPIN is a DIMM100 controlCARD based motherboard evaluation module showcasing control of the most common types of high voltage, three-phase motors including AC induction (ACI), brushless DC (BLDC), and permanent magnet synchronous motors (PMSM). The High Voltage Motor Control Kit has individual DC bus and three-phase voltage sensing making this board for BLDC/PMSM control with TI controlCARD™ an excellent choice for use with the sensorless FOC algorithm.

- The hardware Files are in the `<install_location>\solutions\tmdshvmtrinspin\hardware` folder of C2000WARE-MOTORCONTROL-SDK.

This section explains the steps needed to run the TMDSHVMTRINSPIN with the software supplied through MotorControl SDK. The kit ships with the jumper and switch settings correctly positioned for connecting with the controlCARD. Make sure that these settings are valid on the board as described in the following, and then insert the controlCARD with the TMDSADAP180TO100 adapter into the TMDSHVMTRINSPIN board as shown in Figure 4-10.

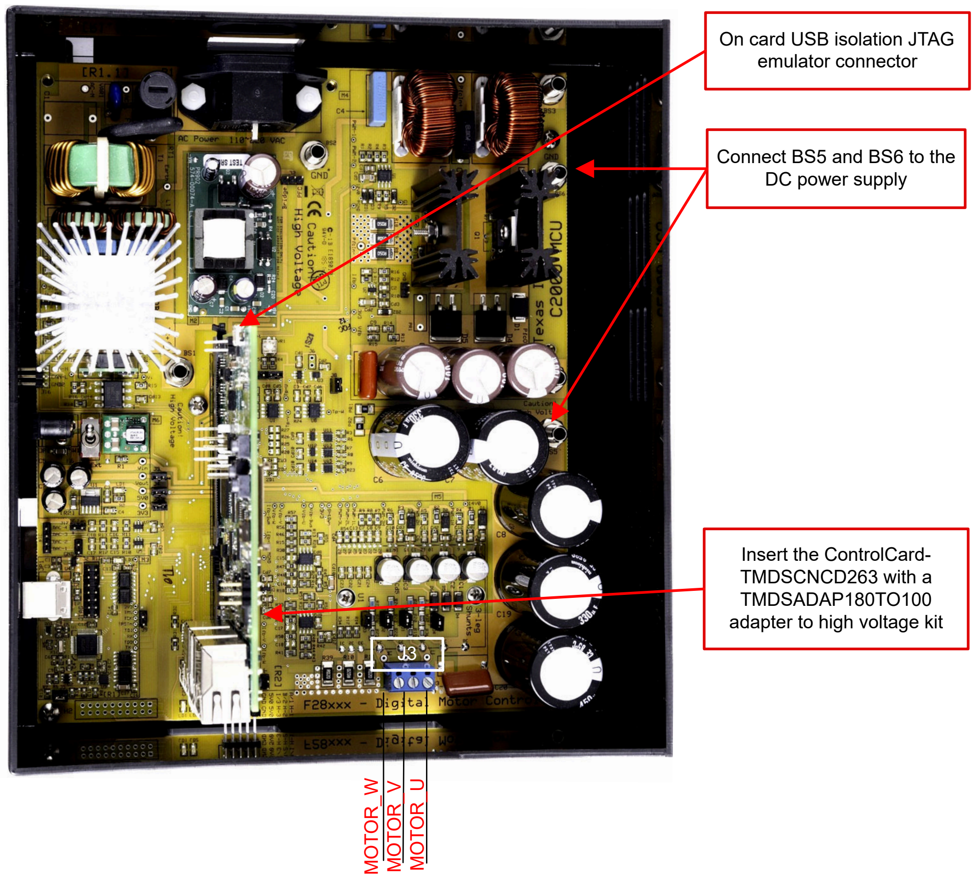

On card USB isolation JTAG emulator connector
Connect BS5 and BS6 to the DC power supply
Insert the ControlCard-TMDSCNCD263 with a TMDSADAP180TO100 adapter to high voltage kit
J3
MOTOR_W
MOTOR_V
MOTOR_U

Figure 4-10. TMDSHVMTRINSPIN Connected to the TMDSCNCD263 with TMDSADAP180TO100

- Make sure nothing is connected to the board, and no power is being supplied to the board.
- Insert the Control card with TMDSADAP180TO100 adapter into the [Main]-J1 controlCARD connector if not already populated.
- Make sure the following jumpers & connector settings are correctly implemented as shown in Figure 4-11.
  – [Main]-J3, J4, J5 and J8 are populated.
  – [Main]-J9 and [M3]-J5 are not populated for using a controlCARD with the onboard emulation to disable the XDS100 on HVKIT.
  – [Main]-J7 is populated between pins 2-3 (pins furthest from the DIMM 100 socket).
  – Make sure that the DC Fan shipped with the kit is connected to the DC Fan Jumper [Main]-J17 when operating the motor under load > 150W.
- Two options to get DC Bus power are as follows, recommend using the external 15V DC power supply.
  – [Main]-J2 is not populated if using the +15V from an external 15VDC power supply. Verify that [M6]-SW1 is in the “Off” position, connect 15V DC power supply to [M6]-JP1.
  – [Main]-J2 is populated with a jumper between bridge and the middle pin if using the +15V power supply from aux power supply module.
- Turn on [M6]-SW1. Now [M6]-LD1 turns on. Notice the control card LED lights up as well indicating the control card is receiving power from the board.
- Connect the motor, encoder, and Hall sensors to the kits as described in Table 4-2 and shown in Figure 4-11.

- Connect a supply voltage from an AC or a DC voltage source to the voltage supply pins. Power is applied when instructed to do so in Section 4.4, keep disconnected otherwise.

Table 4-6 shows the various connections available on the board. The location of these connections on the board are shown in Figure 4-11.

Table 4-6. Key Jumpers, Connectors Explanation

[Main]-P1	AC input connector (110V – 220VAC)
[Main]-TB3	Terminal Block to connect motor
[Main]-BS1	Banana Jack for Output from AC Rectifier
[Main]-BS2, BS6	Banana Jack for GND Connection
[Main]-BS3	Banana Jack for connecting an input voltage for the PFC stage, this is typically rectified AC voltage from the [Main]-BS1 connector.
[Main]-BS4	Banana Jack for connecting a load to the output from the PFC stage, When using PFC+Motor project the output of the PFC stage is connected to the input for the inverter bus for example [Main]-BS5
[Main]-BS5	Banana Jack for input of DC bus voltage for the inverter
[Main]-J2	Aux power supply module input voltage selection jumper, • When jumper connected to Bridge position the aux power supply module sources power from the AC rectifier bridge output. • When Jumper connected to PFC position the aux power supply module sources power from the output of the PFC stage.
[Main]-J3, J4, J5	Jumpers J3,J4 and J5 are used for sourcing 15V, 5V and 3.3V power respectively for the board from the 15V DC Power supply.
[Main]-J7	J7 is used to select the over current protection threshold source
[Main]-J8	J8 is used to enable/disable the IPM over current protection
[Main]-J9	JTAG TRSTn disconnect jumper, populating the jumper enables JTAG connection to the Microcontroller. The jumpers need to be unpopulated when no JTAG connection is required such as when booting from FLASH.
[Main]-J14	PWMDAC outputs: Gives voltage outputs that result from a PWM being attached to a first-order low-pass filter. Pins 1,2,3 and 4 are attached to low pass filtered PWM output pins respectively to observe system variables on an oscilloscope.
[Main]-J16	Isolated CAN bus connector
[Main]-J17	Connector to supply power to the DC fan (shipped with the board) that is attached to the IPM heat sink.
[Main]-H1	QEP connector: connects with a 0-5V QEP sensor to gather information on a motor's speed and position. CAP/Hall effect sensor connector: connects with a 0-5V sensor to gather information on a motor's speed and position.
[M1]-F1	Fuse for the AC input
[M3]-JP1	USB connection for on-board emulation
[M3]-J2	External JTAG interface: this connector gives access to the JTAG emulation pins. If external emulation is desired, place a jumper across [M3]-J5 and connect the emulator to the board. To power the emulation logic a USB connector still needs to be connected to [M3]-JP1.
[M3]-J5	On-board emulation disable jumper: Place a jumper here to disable the on-board emulator and give access to the external interface.

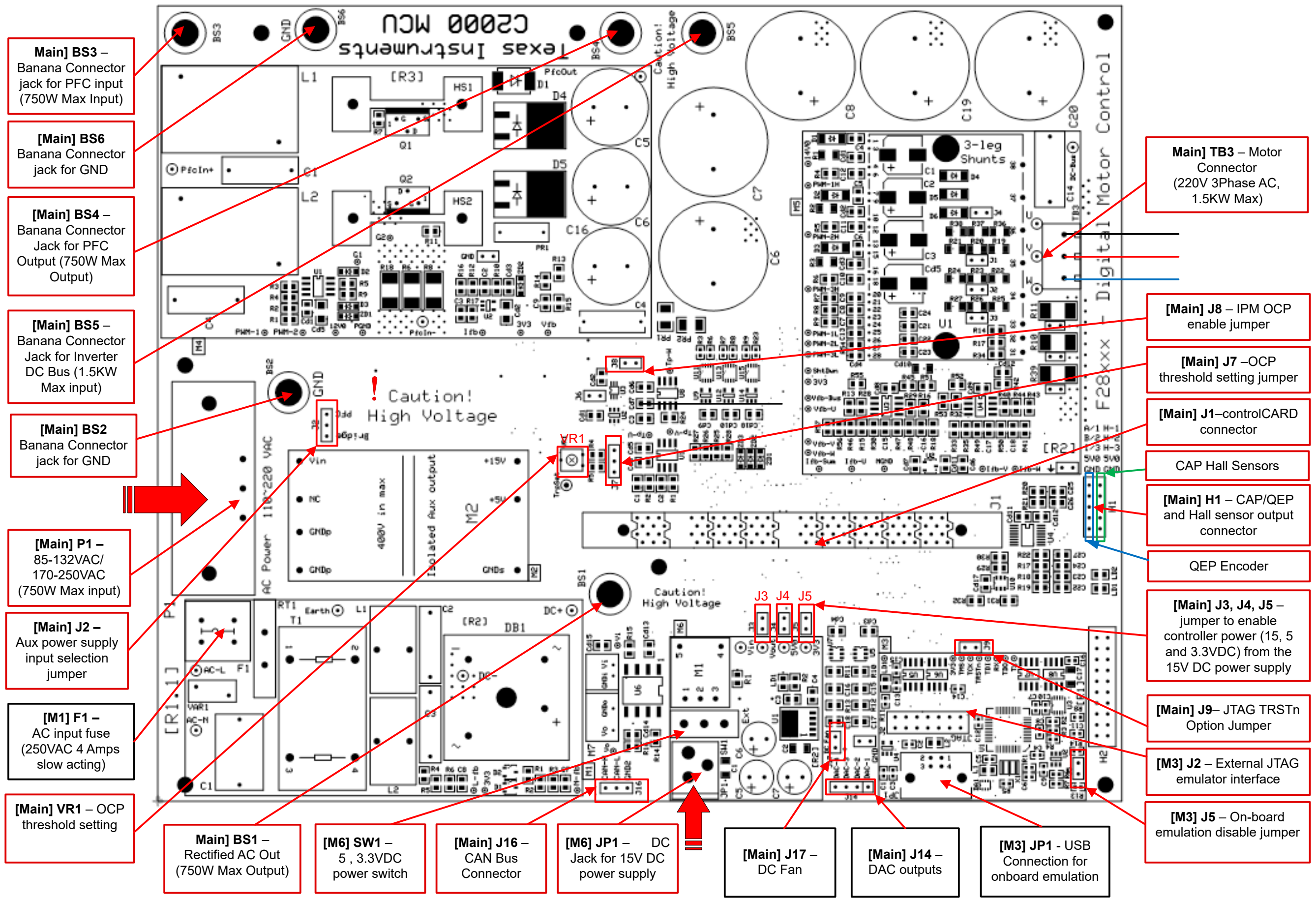

Main] BS3 – Banana Connector jack for PFC input (750W Max Input)
[Main] BS6 Banana Connector jack for GND
[Main] BS4 – Banana Connector Jack for PFC Output (750W Max Output)
[Main] BS5 – Banana Connector Jack for Inverter DC Bus (1.5KW Max input)
[Main] BS2 Banana Connector jack for GND
[Main] P1 – 85-132VAC/ 170-250VAC (750W Max input)
[Main] J2 – Aux power supply input selection jumper
[M1] F1 – AC input fuse (250VAC 4 Amps slow acting)
[Main] VR1 – OCP threshold setting
Main] BS1 – Rectified AC Out (750W Max Output)
[M6] SW1 – 5 , 3.3VDC power switch
[Main] J16 – CAN Bus Connector
[M6] JP1 – DC Jack for 15V DC power supply
[Main] J17 – DC Fan
[Main] J14 – DAC outputs
[M3] JP1 - USB Connection for onboard emulation
Main] TB3 – Motor Connector (220V 3Phase AC, 1.5KW Max)
[Main] J8 – IPM OCP enable jumper
[Main] J7 –OCP threshold setting jumper
[Main] J1–controlCARD connector
CAP Hall Sensors
[Main] H1 – CAP/QEP and Hall sensor output connector
QEP Encoder
[Main] J3, J4, J5 – jumper to enable controller power (15, 5 and 3.3VDC) from the 15V DC power supply
[Main] J9– JTAG TRSTn Option Jumper
[M3] J2 – External JTAG emulator interface
[M3] J5 – On-board emulation disable jumper
C2000 MCU
Texas Instruments
Digital Motor Control
F28xxx
Caution! High Voltage
VR1
J3 J4 J5
3-leg Shunts
AC Power 110~220 VAC
Isolated Aux output
400V in max

Figure 4-11. TMDSHVMTRINSPIN Kit Jumpers and Connectors Diagram

4.4 Test Results

The system is gradually tested and verified in multiple stages so that the final system can be confidently operated. To select a particular build option, change the value of the DMC_BUILDLEVEL define to the desired DMC_LEVEL_X option in the *sys_settings.h* file. After the build option is selected, compile the project by right-clicking on the project name and clicking *Rebuild Project*.

4.4.1 Level 1 Incremental Build

Objectives for this build level:

- Use the HAL object to initialize the peripherals of the MCU for the motor drive hardware.
- Verify the PWM and ADC driver modules
- Verify the ADC Offset validation
- Become familiar with the operation of CCS. More details about CCS can be found in the CCS User's Guide.

In this build level, the board is executed in open loop mode with a fixed PWM 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 and maintains there are no hardware issues. 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 4-12.

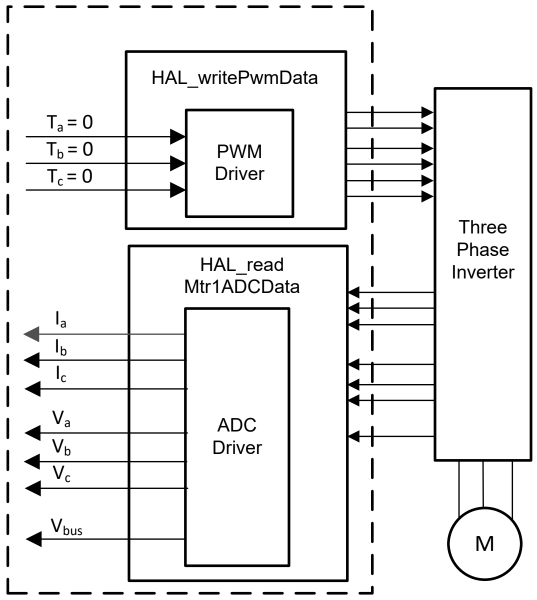


Figure 4-12. Build Level 1 Software Block Diagram - Offset Validation

4.4.1.1 Build and Load Project

1. Set up the motor driver hardware board and the TI LaunchPad or controlCARD as described in Section 4.3, except the motor does **NOT** need to be connected to the motor driver board in this build level.
2. Connect a USB cable from the computer to the onboard USB connector on the TI LaunchPad or controlCARD to enable isolation JTAG emulation to the MCU.
3. Power on the motor driver board by applying the appropriate voltage to the bus voltage input terminal as described in Section 4.3.
4. Import the universal motor control project into CCS and select the correct build configuration as described in Section 4.2.1. Open the *sys_settings.h* file and set DMC_BUILDLEVEL to DMC_LEVEL_1. This can make sure the project is configured to run the first incremental build.
5. In the Project Explorer window, make sure the correct target configuration file is set as Active by right-clicking on the desired target configuration file name and selecting *Set as Active Target Configuration*. The recommendation is to also set the desired target configuration file as default by right-clicking on the file name and selecting *Set as Default Target Configuration*. One reason for doing this is because there is no visible indicator to show which file is active, but if the file is set to default then the [default] indicator appears next to the file name in the project explorer window. Setting the file as default can also cause the file to be

used by default unless a different configuration file is specifically set to Active. You can also link a target configuration to a project in the workspace by going to *View* > *Target Configurations* and right-clicking on the target configuration name in the Target Configurations view and selecting *Link to Project*.

6. Right-click the project name and click *Rebuild Project*. Watch the Console window. Any errors in the project are displayed in the Console window.
7. On successful completion of the build, click the Debug button or click *Run* → *Debug*. The IDE now automatically connects to the target, load the output file into the device, and change to the Debug perspective. The CCS Debug icon appears in the upper right corner, indicating that the user is now in the Debug Perspective view. The program needs to be halted at the start of main().

4.4.1.2 Setup Debug Environment Windows

The standard debug practice is to watch local and global variables while debugging code. There are various methods for doing this in CCS, such as memory views and watch views. Additionally, CCS has the ability to create time (and frequency) domain plots. This ability allows the user to view waveforms using the graph tool. For information on how to set up and configure the graph tool, see Section 4.5.1. For information on setting up the Expressions window, see the following instructions.

1. Setup watch window: *Click View* → *Expressions* on the menu bar to open an Expressions watch window. Variables can be added to the Expressions window by clicking *Add new expression* within the Expressions window and entering the name of the variable and then pressing enter. The number format that the variable value is displayed in is based on the number format associated with the variable when the variable value was declared. You can change the desired number format for a particular variable by right clicking on the variable and navigating to *Number Format* and selecting the desired format.
2. Alternately, a group of variables can be imported into the Expressions window by right clicking within the Expressions window and clicking Import, browse to the directory of the project at `<workspace>\universal_motorcontrol_am263x_r5fss0-0_nortos_ti-arm-clang\src_control\debug\`, select the *universal_motor_control_level1.txt* file, and click OK to import the variables shown in Figure 4-13.

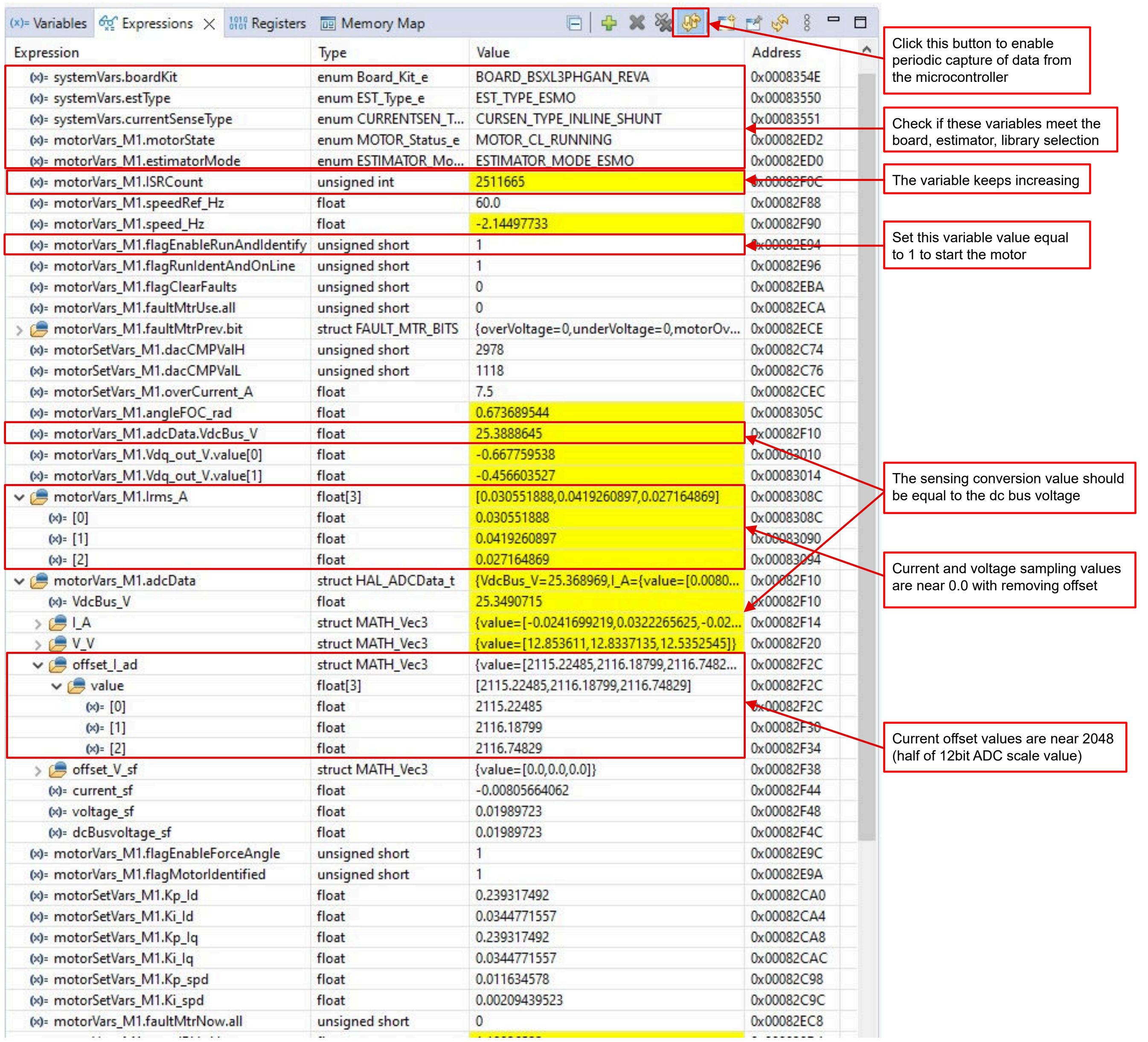
Note

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

3. Note: the structure of variables *motorVars_M1* has references to most variables that are related to controlling motor drive.
4. Click on the Continuous Refresh button in the top right corner of the Expressions Window tab to enable periodic capture of data from the Microcontroller. By clicking the *View Menu* button (the 3 dots in the upper right hand corner of the Expressions window),you can select *Continuous Refresh Interval* and edit the refresh rate of the Expressions window. Note that choosing too fast an interval can affect performance.

4.4.1.3 Run the Code

1. Disable the data cache by unchecking the *Data Cache Enabled* in the *Tools* > *ARM Advanced Features*.
2. Run the code by pressing on the Resume button, or click *Run* → *Resume* in the Debug tab.
3. The project now runs, and the values in the graph and watch window keep updating.
4. In the Expressions window, set the variables *motorVars_M1.flagEnableRunAndIdentify* to 1 after *systemVars.flagEnableSystem* was automatically set to 1 in the watch window.
5. The project now runs, and the values in the graphs and expressions window continuously update as shown in Figure 4-13 while using this project. You can re-size the windows according to your preference.
6. In the watch view, the variables *motorVars_M1.flagRunIdentAndOnLine* is set to 1 automatically if there are no faults. The *ISRCount* increases continuously.
7. Check the calibration offsets of the motor driver board. The offset values of the motor phase current sensing values are equal to approximately half of the ADC scale current as shown in Figure 4-13.
8. If using the graph tool, the variables shown in the graphs are the FOC angle and phase currents for phase u, v and w.
9. Expand and check the *MotorVars_M1.faultMtrPrev.bit* structure to maintain that there are no fault flags set.
10. Using an oscilloscope, probe the PWM outputs that are used for motor drive control. The duty cycles of the three PWMs are set to 50% in this build level. The expected PWM output waveforms are as shown in Figure 4-14. The PWM switching frequency are the same as the value that was set for the USER_M1_PWM_FREQ_kHz define in the *user_mtr1.h* file.
11. Set the *motorVars_M1.flagEnableRunAndIdentify* variable to 0 to disable the PWMs.
12. If any of the previous steps provide unexpected results, additional debug is necessary. A few things to check are:
    a. Make sure that the motor driver board being used is the same as the board selected in the build configurations.
    b. Make sure that the proper predefines are set.
    c. Make sure that the switches are configured properly on the Lunchpad/ControlCARD as described in Section 4.3.
13. Once the previous steps are complete, the controller can now be halted, and the debug connection terminated. Halt the controller by first clicking the Halt button on the toolbar or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on the button or clicking *Run* → *Reset*→CPU Reset.
14. Close the CCS debug session by clicking the Terminate Debug Session button or clicking *Run* → *Terminate.* This halts the program and disconnect Code Composer from the MCU.
15. Terminating the debug session each time the user changes or runs the code again is not necessary. Instead, the following procedure can be followed. After rebuilding the project, press the button or click *Run* → *Reset*→CPU Reset, and then press the Restart button or click *Run* → *Restart.* The project must be terminated if the target device or the configuration is changed, and before shutting down CCS.



Expression	Type	Value	Address
systemVars.boardKit	enum Board_Kit_e	BOARD_BSXL3PHGAN_REVA	0x0008354E
systemVars.estType	enum EST_Type_e	EST_TYPE_ESMO	0x00083550
systemVars.currentSenseType	enum CURRENTSEN_T...	CURSEN_TYPE_INLINE_SHUNT	0x00083551
motorVars_M1.motorState	enum MOTOR_Status_e	MOTOR_CL_RUNNING	0x00082ED2
motorVars_M1.estimatorMode	enum ESTIMATOR_Mo...	ESTIMATOR_MODE_ESMO	0x00082ED0
motorVars_M1.ISRCount	unsigned int	2511665	0x00082F0C
motorVars_M1.speedRef_Hz	float	60.0	0x00082F88
motorVars_M1.speed_Hz	float	-2.14497733	0x00082F90
motorVars_M1.flagEnableRunAndIdentify	unsigned short	1	0x00082E94
motorVars_M1.flagRunIdentAndOnLine	unsigned short	1	0x00082E96
motorVars_M1.flagClearFaults	unsigned short	0	0x00082EBA
motorVars_M1.faultMtrUse.all	unsigned short	0	0x00082ECA
motorVars_M1.faultMtrPrev.bit	struct FAULT_MTR_BITS	{overVoltage=0,underVoltage=0,motorOv...	0x00082ECE
motorSetVars_M1.dacCMPValH	unsigned short	2978	0x00082C74
motorSetVars_M1.dacCMPValL	unsigned short	1118	0x00082C76
motorSetVars_M1.overCurrent_A	float	7.5	0x00082CEC
motorVars_M1.angleFOC_rad	float	0.673689544	0x0008305C
motorVars_M1.adcData.VdcBus_V	float	25.3888645	0x00082F10
motorVars_M1.Vdq_out_V.value[0]	float	-0.667759538	0x00083010
motorVars_M1.Vdq_out_V.value[1]	float	-0.456603527	0x00083014
motorVars_M1.Irms_A	float[3]	[0.030551888,0.0419260897,0.027164869]	0x0008308C
[0]	float	0.030551888	0x0008308C
[1]	float	0.0419260897	0x00083090
[2]	float	0.027164869	0x00083094
motorVars_M1.adcData	struct HAL_ADCData_t	{VdcBus_V=25.368969,I_A={value=[0.0080...	0x00082F10
VdcBus_V	float	25.3490715	0x00082F10
I_A	struct MATH_Vec3	{value=[-0.0241699219,0.0322265625,-0.02...	0x00082F14
V_V	struct MATH_Vec3	{value=[12.853611,12.8337135,12.5352545]}	0x00082F20
offset_I_ad	struct MATH_Vec3	{value=[2115.22485,2116.18799,2116.7482...	0x00082F2C
value	float[3]	[2115.22485,2116.18799,2116.74829]	0x00082F2C
[0]	float	2115.22485	0x00082F2C
[1]	float	2116.18799	0x00082F30
[2]	float	2116.74829	0x00082F34
offset_V_sf	struct MATH_Vec3	{value=[0.0,0.0,0.0]}	0x00082F38
current_sf	float	-0.00805664062	0x00082F44
voltage_sf	float	0.01989723	0x00082F48
dcBusvoltage_sf	float	0.01989723	0x00082F4C
motorVars_M1.flagEnableForceAngle	unsigned short	1	0x00082E9C
motorVars_M1.flagMotorIdentified	unsigned short	1	0x00082E9A
motorSetVars_M1.Kp_Id	float	0.239317492	0x00082CA0
motorSetVars_M1.Ki_Id	float	0.0344771557	0x00082CA4
motorSetVars_M1.Kp_Iq	float	0.239317492	0x00082CA8
motorSetVars_M1.Ki_Iq	float	0.0344771557	0x00082CAC
motorSetVars_M1.Kp_spd	float	0.011634578	0x00082C98
motorSetVars_M1.Ki_spd	float	0.00209439523	0x00082C9C
motorVars_M1.faultMtrNow.all	unsigned short	0	0x00082EC8

Click this button to enable periodic capture of data from the microcontroller

Check if these variables meet the board, estimator, library selection

The variable keeps increasing

Set this variable value equal to 1 to start the motor

The sensing conversion value should be equal to the dc bus voltage

Current and voltage sampling values are near 0.0 with removing offset

Current offset values are near 2048 (half of 12bit ADC scale value)

Figure 4-13. Build Level 1: Variables in Expressions Window

The PWM with deadband outputs to the input of the gate drive are shown in Figure 4-14.

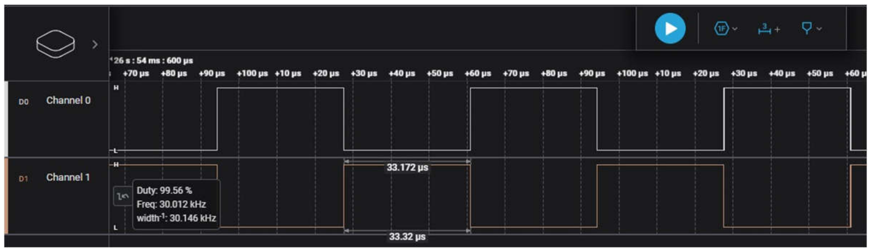


Figure 4-14. Build Level 1: PWM Output Waveforms

4.4.2 Level 2 Incremental Build

Objectives learned in this build level:

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

In this build level the system is running with open-loop control, so the ADC values are only used for verification and validation, the ADC values are not actually used in the control loop of the motor. The software flow for this build level is shown in Figure 4-15.

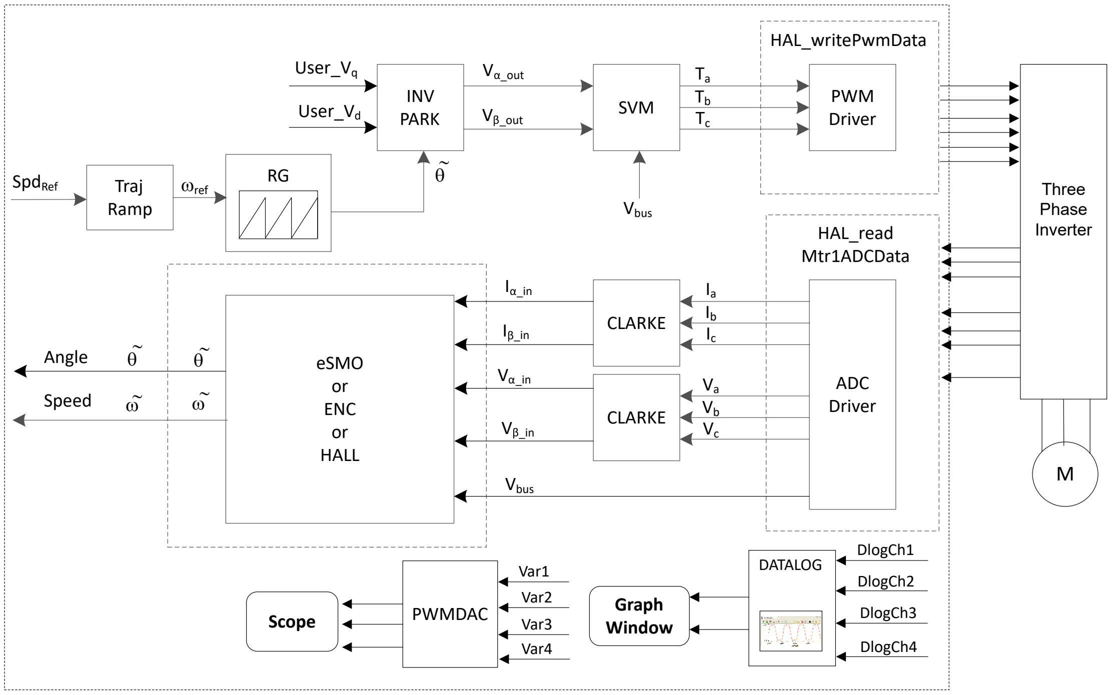


Figure 4-15. Build Level 2 Software Block Diagram - Open Loop Control

4.4.2.1 Build and Load Project

Connect the motor to the appropriate terminals on the motor driver evaluation board. Follow steps 2-7 of Section 4.4.1.1 to build and load the project. In step 4, set the DMC_BUILDLEVEL to DMC_LEVEL_2.

4.4.2.2 Setup Debug Environment Windows

Follow the steps in Section 4.4.1.2 to import the variables into the Expressions window. For build level 2, select the *universal_motor_control_level2.txt* file. The Expressions window appears as shown in Figure 4-16.

4.4.2.3 Run the Code

1. Power on the appropriate power supply, and gradually increase the output voltage of the power supply to get an appropriate DC-bus voltage.
2. If using the graph tool, level 2 uses the same graph configurations and parameters as lab 1 to monitor 2 of the phase currents.
3. Disable the data cache by unchecking the *Data Cache Enabled* in *Tools* > *ARM Advanced Features*.
4. Run the project by clicking on the Resume button, or click *Run* → *Resume* in the Debug tab. The *systemVars.flagEnableSystem* is set to *1* after a fixed time, that means the offsets calibration has completed. The fault flags, *motorVars_M1.faultMtrUse.all* is equal to *0.* If this is not the case, the user must double check the current and voltage sensing circuit in level 1 as described in Section 4.4.1.3. Also, if the *moduleOverCurrent* in *motorVars_M1.faultMtrPrev.bit* is 1, then *motorSetVars_M1.overCurrent_A* needs to be set to higher value to avoid initial high current fault.
5. To verify the current and voltage sensing circuits of the motor driver, set the variable *motorVars_M1.flagEnableRunAndIdentify* to *1* in the Expressions window as shown in Figure 4-16. The motor runs with voltage/frequency (v/f) open loop. If the motor does not spin smoothly, tune the v/f profile parameters in the *user_mtr1.h* file as shown in the following according to the specification of the motor. Note: modification of these parameters requires rebuilding the project. See step 15 of Section 4.4.1.3 for more information on rebuilding the project in debug mode.

```
#define USER_MOTOR1_FREQ_LOW_HZ           (5.0)          // Hz
#define USER_MOTOR1_FREQ_HIGH_HZ          (400.0)        // Hz
#define USER_MOTOR1_VOLT_MIN_V            (1.0)          // Volt
#define USER_MOTOR1_VOLT_MAX_V            (24.0)         // Volt
```

6. The *motorVars_M1.speedRef_Hz* variable is used to set the speed reference for the motor. Check the value of the *motorVars_M1.speed_Hz* variable in the Expressions window to maintain that the motor speed (*motorVars_M1*.speed_Hz) is close to the reference speed (*motorVars_M1.speedRef_Hz)* as shown in Figure 4-16.
7. In this build level, the current sensing, voltage sensing, rotor angle estimator, and generator need to be validated. This can be done using the PWMDAC in HV motor kit as described in Section 4.5.2. Additionally, the DATALOG module can be used to view these sensing waveforms. For more information on using the DATALOG to view the currents, voltages, and angle signals, see step 8.
8. If using the DATALOG module with the graph tool to check the current sensing signals, voltage sensing signals, and the angle outputs, follow the steps described in the following. For more info on the datalog module, see Section 4.5.1. Note:You must rebuild the project in between each of the following steps after modifying the code.
   a. To test the phase currents using the DATALOG module, the following code must be set up in the *sys_main.c* file. Note: this code is already configured by default for build level 2. The phase current sampling signals waveform displayed on the graph tool as shown in Figure 4-18.

   ```
   datalogObj->iptr[0] = (float32_t*) &motorVars_M1.adcData.I_A.value[0];
   datalogObj->iptr[1] = (float32_t*) &motorVars_M1.adcData.I_A.value[1];
   ```

   b. To test the phase voltage using the DATALOG module, the following code must be set up in the *sys_main.c* file. The phase voltage sampling signals waveform displayed on graph tool as shown in Figure 4-19.

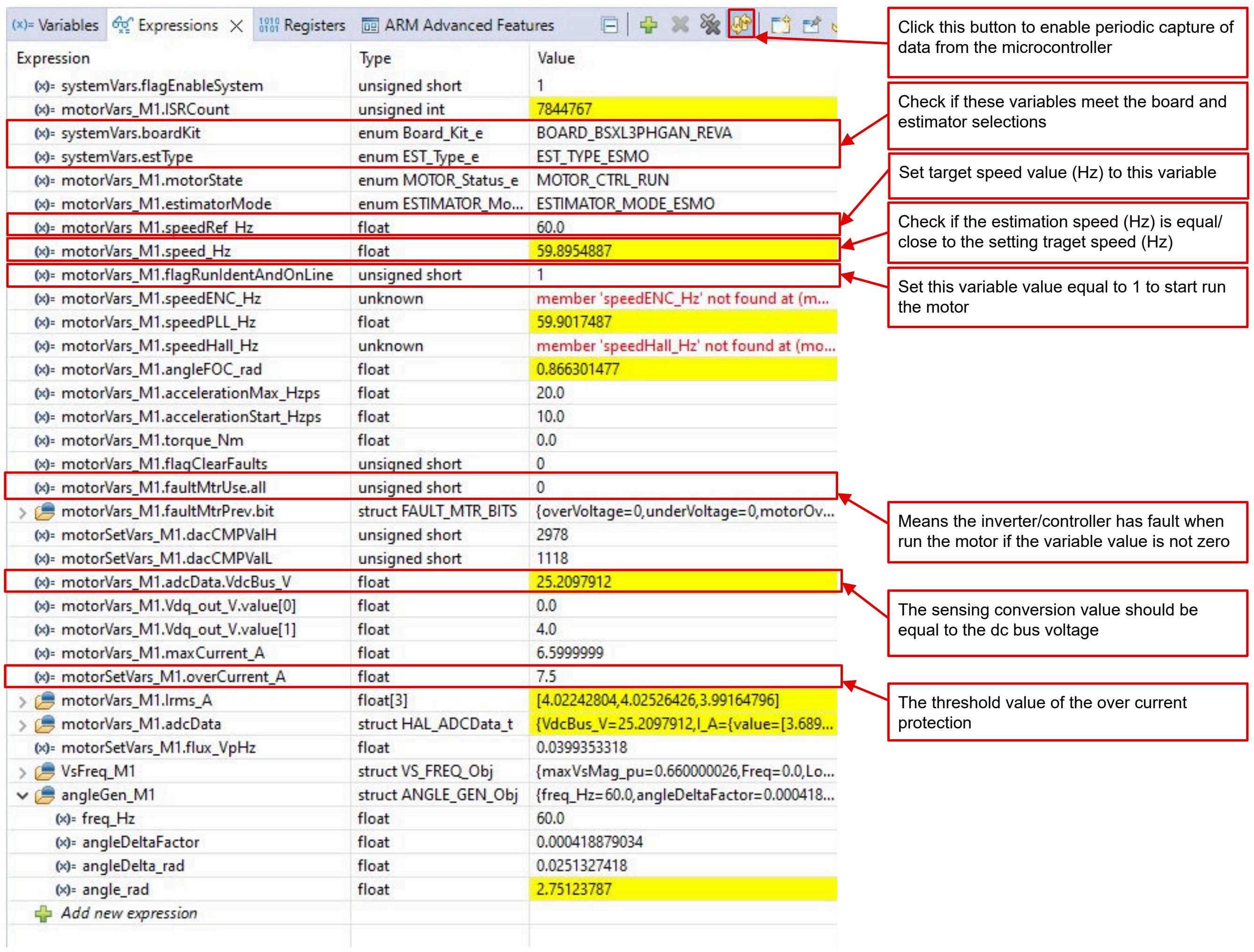
   ```
   datalogObj->iptr[2] = (float32_t*) &motorVars_M1.adcData.V_V.value[0];
   ```

   c. The angle from the force angle generator or estimator can be monitored on the graph tool as shown in Figure 4-21. Notice that the angle of the force angle generator is very similar as the estimated rotor angle of the eSMO estimator.

   ```
   datalogObj->iptr[3] = (float32_t*) &motorVars_M1.angleFOC_rad;
   ```

9. 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 triggers if the *motorVars_M1.overCurrent_A* is set to a value less than the motor phase current actual value, the PWM output is disabled, the *motorVars_M1.flagEnableRunAndIdentify* is cleared to *0,* as shown in Figure 4-17.
10. Set the variables *motorVars_M1.flagEnableRunAndIdentify* to 0 to stop run the motor.

11. Once complete, the controller can now be halted, and the debug connection terminated. Fully halting the controller by first clicking the Halt button on the toolbar Suspend or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on CPU Reset or clicking *Run* → *Reset*.
12. Close CCS debug session by clicking on Terminate Debug Session button or clicking *Run* → *Terminate*.
13. Power off the power supply to the inverter kit.



Expression	Type	Value
systemVars.flagEnableSystem	unsigned short	1
motorVars_M1.ISRCount	unsigned int	7844767
systemVars.boardKit	enum Board_Kit_e	BOARD_BSXL3PHGAN_REVA
systemVars.estType	enum EST_Type_e	EST_TYPE_ESMO
motorVars_M1.motorState	enum MOTOR_Status_e	MOTOR_CTRL_RUN
motorVars_M1.estimatorMode	enum ESTIMATOR_Mo...	ESTIMATOR_MODE_ESMO
motorVars_M1.speedRef_Hz	float	60.0
motorVars_M1.speed_Hz	float	59.8954887
motorVars_M1.flagRunIdentAndOnLine	unsigned short	1
motorVars_M1.speedENC_Hz	unknown	member 'speedENC_Hz' not found at (m...
motorVars_M1.speedPLL_Hz	float	59.9017487
motorVars_M1.speedHall_Hz	unknown	member 'speedHall_Hz' not found at (mo...
motorVars_M1.angleFOC_rad	float	0.866301477
motorVars_M1.accelerationMax_Hzps	float	20.0
motorVars_M1.accelerationStart_Hzps	float	10.0
motorVars_M1.torque_Nm	float	0.0
motorVars_M1.flagClearFaults	unsigned short	0
motorVars_M1.faultMtrUse.all	unsigned short	0
motorVars_M1.faultMtrPrev.bit	struct FAULT_MTR_BITS	{overVoltage=0,underVoltage=0,motorOv...
motorSetVars_M1.dacCMPValH	unsigned short	2978
motorSetVars_M1.dacCMPValL	unsigned short	1118
motorVars_M1.adcData.VdcBus_V	float	25.2097912
motorVars_M1.Vdq_out_V.value[0]	float	0.0
motorVars_M1.Vdq_out_V.value[1]	float	4.0
motorVars_M1.maxCurrent_A	float	6.5999999
motorSetVars_M1.overCurrent_A	float	7.5
motorVars_M1.Irms_A	float[3]	[4.02242804,4.02526426,3.99164796]
motorVars_M1.adcData	struct HAL_ADCData_t	{VdcBus_V=25.2097912,I_A={value=[3.689...
motorSetVars_M1.flux_VpHz	float	0.0399353318
VsFreq_M1	struct VS_FREQ_Obj	{maxVsMag_pu=0.660000026,Freq=0.0,Lo...
angleGen_M1	struct ANGLE_GEN_Obj	{freq_Hz=60.0,angleDeltaFactor=0.000418...
freq_Hz	float	60.0
angleDeltaFactor	float	0.000418879034
angleDelta_rad	float	0.0251327418
angle_rad	float	2.75123787
Add new expression		

Click this button to enable periodic capture of data from the microcontroller

Check if these variables meet the board and estimator selections

Set target speed value (Hz) to this variable

Check if the estimation speed (Hz) is equal/ close to the setting traget speed (Hz)

Set this variable value equal to 1 to start run the motor

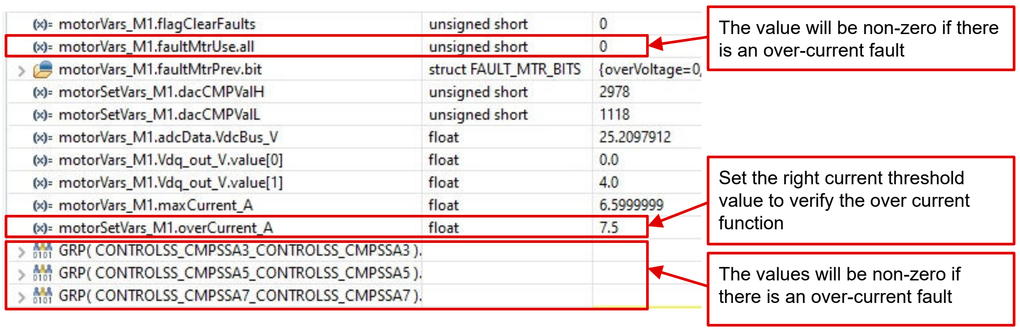
Means the inverter/controller has fault when run the motor if the variable value is not zero

The sensing conversion value should be equal to the dc bus voltage

The threshold value of the over current protection

Figure 4-16. Build Level 2: Variables in Expressions Window

Adjust the value of *motorVars_M1.overCurrent_A* in Expression window to trigger the over current fault as shown in Figure 4-17.



motorVars_M1.flagClearFaults	unsigned short	0
motorVars_M1.faultMtrUse.all	unsigned short	0
motorVars_M1.faultMtrPrev.bit	struct FAULT_MTR_BITS	{overVoltage=0,
motorSetVars_M1.dacCMPValH	unsigned short	2978
motorSetVars_M1.dacCMPValL	unsigned short	1118
motorVars_M1.adcData.VdcBus_V	float	25.2097912
motorVars_M1.Vdq_out_V.value[0]	float	0.0
motorVars_M1.Vdq_out_V.value[1]	float	4.0
motorVars_M1.maxCurrent_A	float	6.5999999
motorSetVars_M1.overCurrent_A	float	7.5
GRP(CONTROLSS_CMPSSA3_CONTROLSS_CMPSSA3).		
GRP(CONTROLSS_CMPSSA5_CONTROLSS_CMPSSA5).		
GRP(CONTROLSS_CMPSSA7_CONTROLSS_CMPSSA7).		

The value will be non-zero if there is an over-current fault

Set the right current threshold value to verify the over current function

The values will be non-zero if there is an over-current fault

Figure 4-17. Build Level 2: Current Protection Setting

Use Datalog with Graph Tool to monitor three phase sensing current of the motor as shown in Figure 4-18.

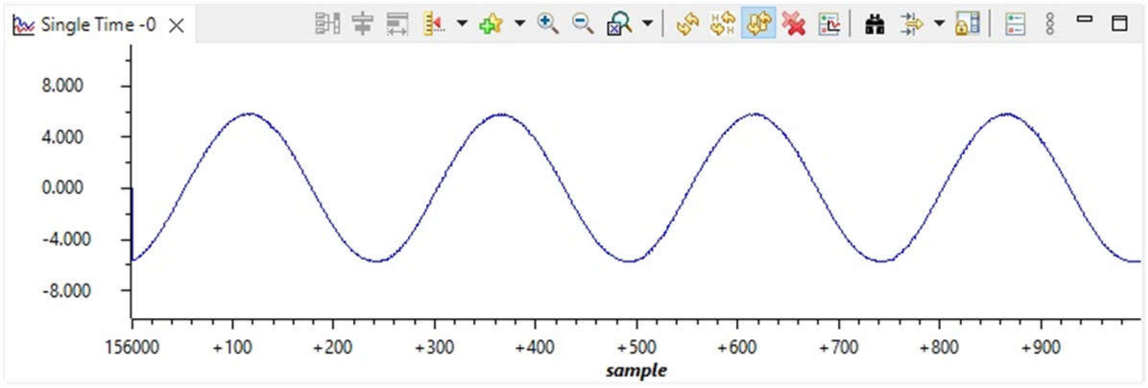


Figure 4-18. Build Level 2: Motor Phase Current Waveforms with Graph Tool

Use Datalog with Graph Tool to monitor three phase sensing voltage of the motor as shown in Figure 4-19. The SVM mode selected here is DPWM with minimum modulation.

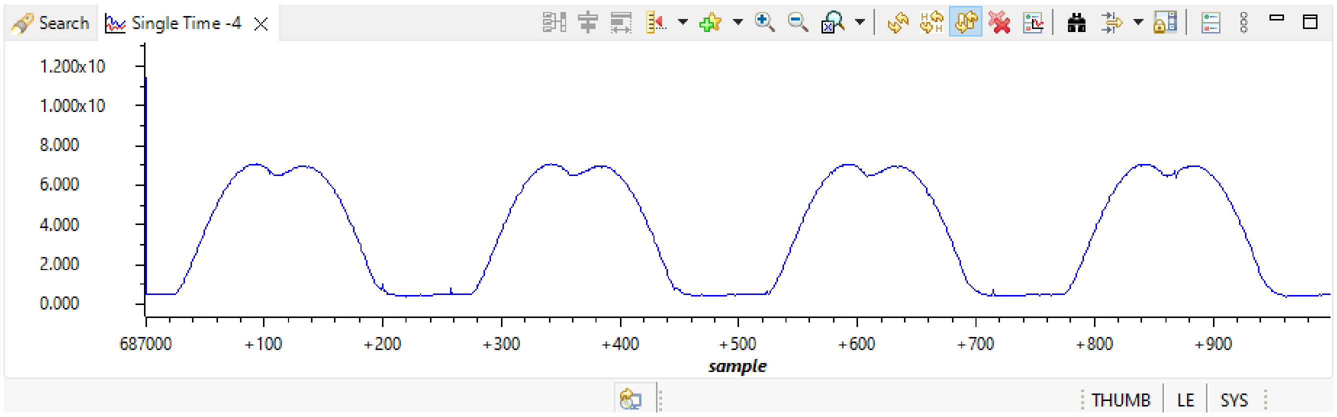


Figure 4-19. Build Level 2: Motor Phase Voltage Waveforms with Graph Tool - DPWM Minimum

Use Datalog with Graph Tool to monitor three phase sensing voltage of the motor with SVPWM Common modulation as shown in Figure 4-20. SVM type can be selected in *motor1_drive.c* file.

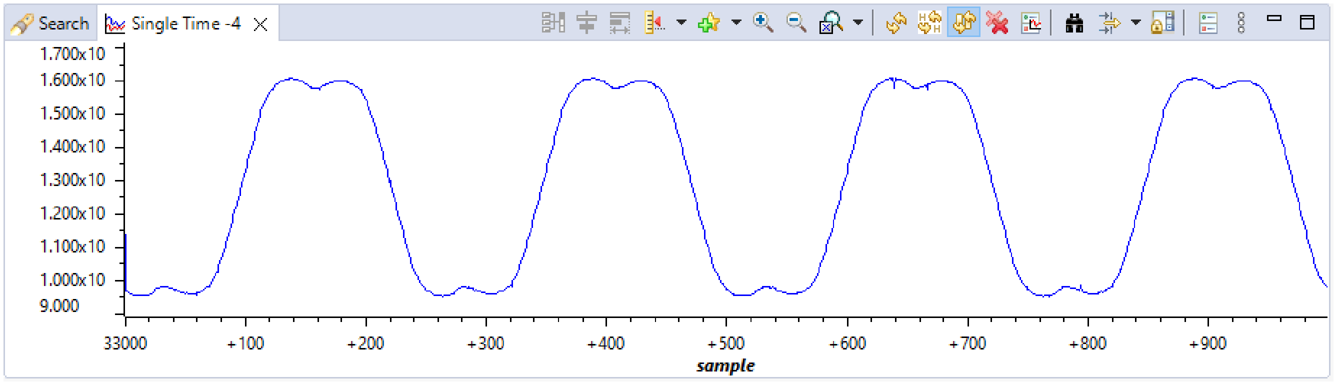


Figure 4-20. Build Level 2: Motor Phase Voltage Waveforms with Graph Tool - SVPWM Common

Use Datalog with Graph Tool to monitor rotor angle of the motor from the angle generator and angle from the eSMO estimator as shown in Figure 4-21.

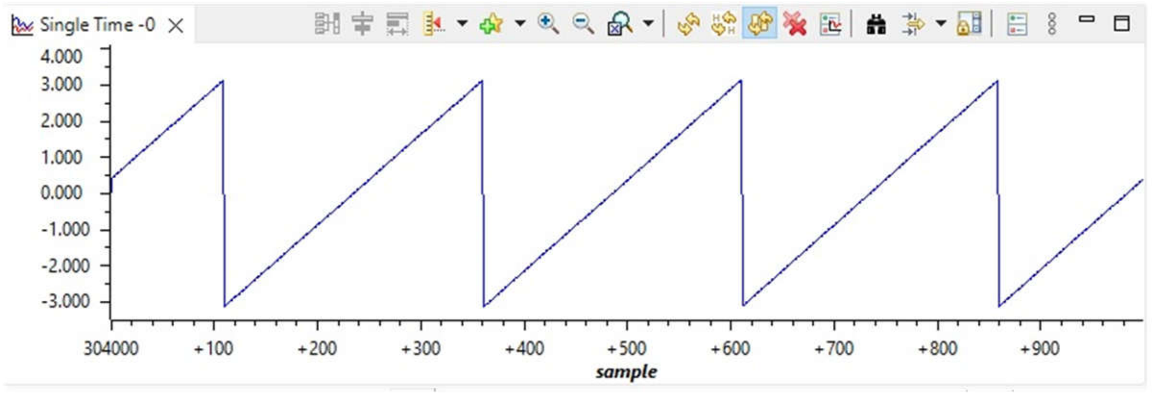


Figure 4-21. Build Level 2: Motor Rotor Angle Waveforms With Graph Tool

4.4.3 Level 3 Incremental Build

Objectives learned in 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 4-22.

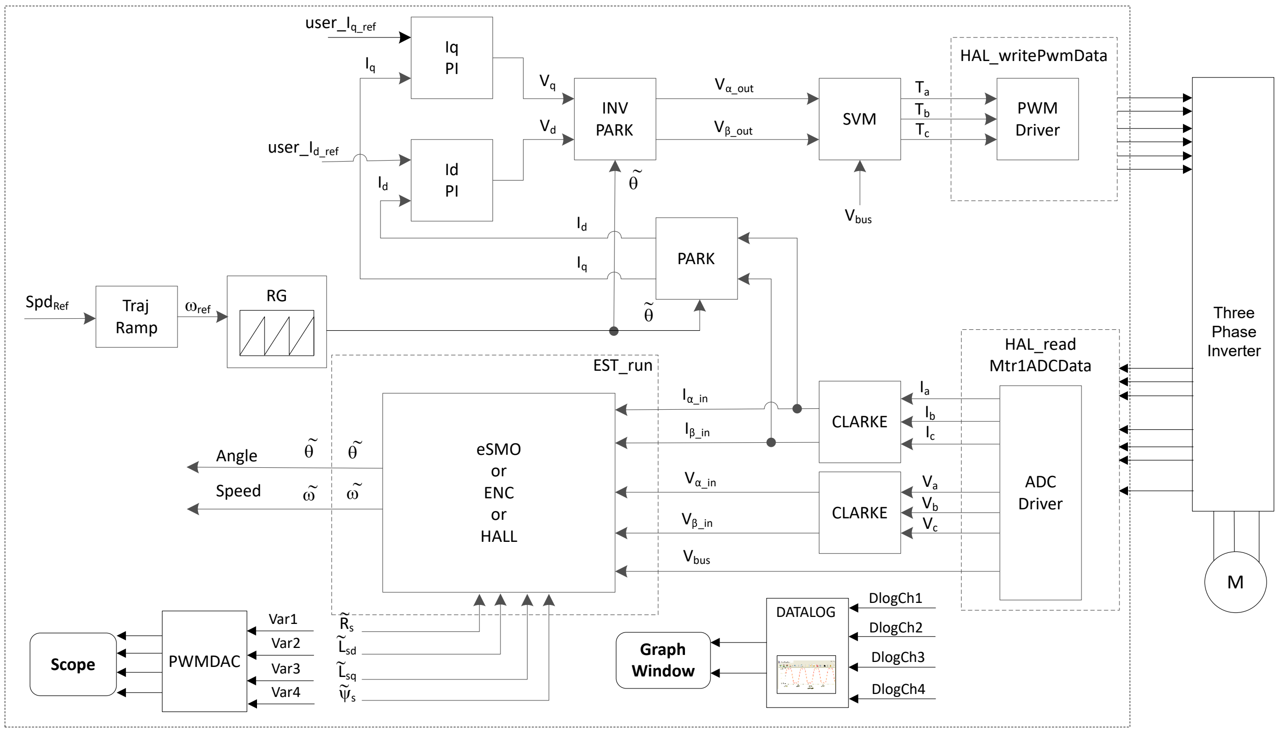


Figure 4-22. Build Level 3 Software Block Diagram - Current Close Loop Control

4.4.3.1 Build and Load Project

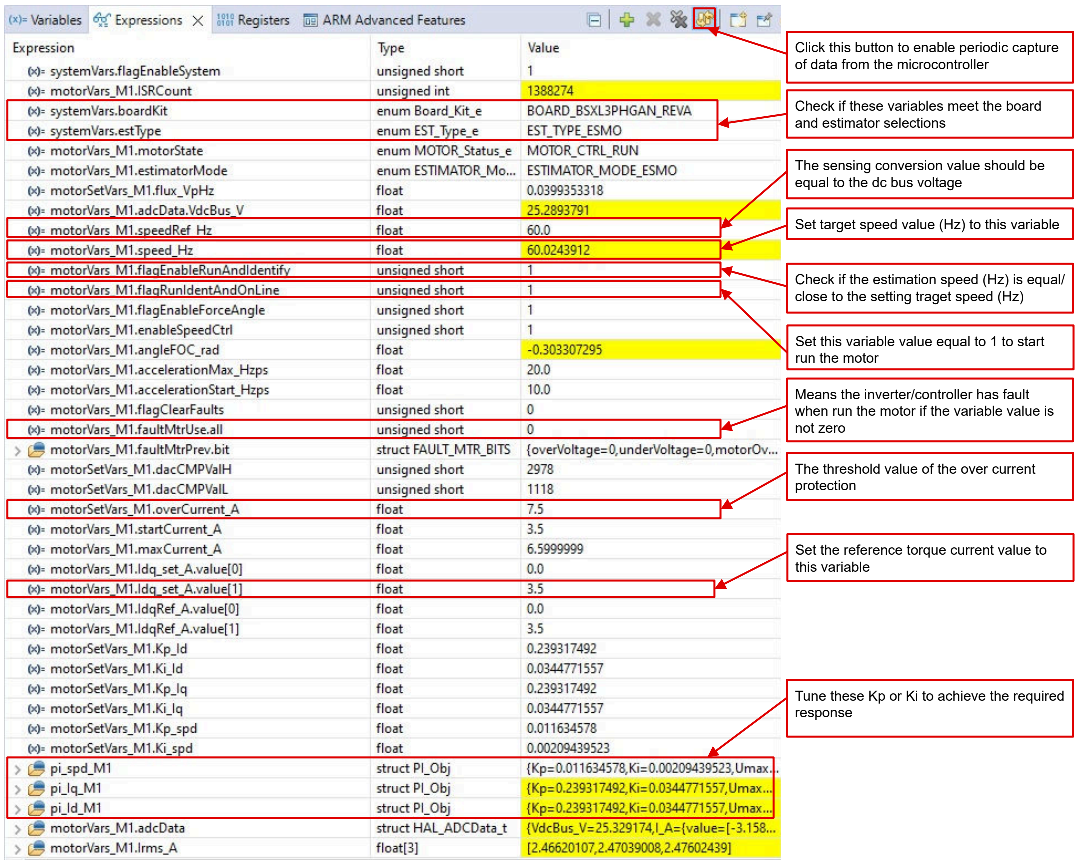
Connect the motor to the related terminals on the power inverter board. Follow the operation steps in Section 4.4.1.1 to build and load project by setting DMC_BUILDLEVEL to DMC_LEVEL_3 in the *sys_settings.h* file.

4.4.3.2 Setup Debug Environment Windows

Follow operation steps in Section 4.4.1.2 to import the variables into the Expressions window by picking *universal_motor_control_level3.txt*. The Expressions window appears as shown in Figure 4-23.

4.4.3.3 Run the Code

1. Power on the AC or DC power supply, gradually increase output voltage at power supply to get an appropriate DC-bus voltage.
2. Disable the data cache by unchecking the *Data Cache Enabled* in *Tools* > *ARM Advanced Features*.
3. Run the project by clicking on Resume button, or click *Run* → *Resume* in the Debug tab. The *systemVars.flagEnableSystem* must be set to *1* after a fixed time, that means the offsets calibration have been done. The fault flags *motorVars_M1.faultMtrUse.all* are equal to *0* , if not, the user have to check the current and voltage sensing circuit as described in Section 4.4.1.
4. To verify run the motor with current closed-loop control, set the variable *motorVars_M1.flagEnableRunAndIdentify* to *1* in the Expressions window as shown in Figure 4-23. The motor runs with a closed-loop control using the angle from the angle generator at a setting speed in the variable *motorVars_M1.speedRef_Hz.* Check the value of *motorVars_M1.speed_Hz* in Expressions window. The values of both variables are very close.
5. Connect oscilloscope probes to the EPWMDAC (for HV kit) outputs and motor phase line to probe the angles and current signals, and current. These waveforms on the oscilloscope appear as shown in Figure 4-24. Change the *motorVars_M1.Idq_set_A.value[1]* in the Expressions window to set the reference torque current, the motor phase current increases with the same percentage accordingly.
6. If the motor cannot run with current-closed loop control and an over current fault appears, check if the sign of *motorVars_M1.adcData.current_sf* and the value of *userParams_M1.current_sf* are set correctly according to the hardware board. The values of both variables are related to the definition constant USER_M1_ADC_FULL_SCALE_CURRENT_A in the *user_mtr1.h* file.
7. Set the variables *motorVars_M1.flagEnableRunAndIdentify* to 0 to stop run the motor.
8. Once complete, the controller can now be halted and the debug connection terminated. Fully halting the controller by first clicking the Suspend button on the toolbar or by clicking *Target* → *Halt*. Finally, reset the controller by clicking on CPU Reset button or clicking *Run* → *Reset*.
9. Close CCS debug session by clicking on Terminate Debug Session button or clicking *Run* → *Terminate*.



Expression	Type	Value
systemVars.flagEnableSystem	unsigned short	1
motorVars_M1.ISRCount	unsigned int	1388274
systemVars.boardKit	enum Board_Kit_e	BOARD_BSXL3PHGAN_REVA
systemVars.estType	enum EST_Type_e	EST_TYPE_ESMO
motorVars_M1.motorState	enum MOTOR_Status_e	MOTOR_CTRL_RUN
motorVars_M1.estimatorMode	enum ESTIMATOR_Mo...	ESTIMATOR_MODE_ESMO
motorSetVars_M1.flux_VpHz	float	0.0399353318
motorVars_M1.adcData.VdcBus_V	float	25.2893791
motorVars_M1.speedRef_Hz	float	60.0
motorVars_M1.speed_Hz	float	60.0243912
motorVars_M1.flagEnableRunAndIdentify	unsigned short	1
motorVars_M1.flagRunIdentAndOnLine	unsigned short	1
motorVars_M1.flagEnableForceAngle	unsigned short	1
motorVars_M1.enableSpeedCtrl	unsigned short	1
motorVars_M1.angleFOC_rad	float	-0.303307295
motorVars_M1.accelerationMax_Hzps	float	20.0
motorVars_M1.accelerationStart_Hzps	float	10.0
motorVars_M1.flagClearFaults	unsigned short	0
motorVars_M1.faultMtrUse.all	unsigned short	0
motorVars_M1.faultMtrPrev.bit	struct FAULT_MTR_BITS	{overVoltage=0,underVoltage=0,motorOv...
motorSetVars_M1.dacCMPValH	unsigned short	2978
motorSetVars_M1.dacCMPValL	unsigned short	1118
motorSetVars_M1.overCurrent_A	float	7.5
motorVars_M1.startCurrent_A	float	3.5
motorVars_M1.maxCurrent_A	float	6.5999999
motorVars_M1.Idq_set_A.value[0]	float	0.0
motorVars_M1.Idq_set_A.value[1]	float	3.5
motorVars_M1.IdqRef_A.value[0]	float	0.0
motorVars_M1.IdqRef_A.value[1]	float	3.5
motorSetVars_M1.Kp_Id	float	0.239317492
motorSetVars_M1.Ki_Id	float	0.0344771557
motorSetVars_M1.Kp_Iq	float	0.239317492
motorSetVars_M1.Ki_Iq	float	0.0344771557
motorSetVars_M1.Kp_spd	float	0.011634578
motorSetVars_M1.Ki_spd	float	0.00209439523
pi_spd_M1	struct PI_Obj	{Kp=0.011634578,Ki=0.00209439523,Umax...
pi_Iq_M1	struct PI_Obj	{Kp=0.239317492,Ki=0.0344771557,Umax...
pi_Id_M1	struct PI_Obj	{Kp=0.239317492,Ki=0.0344771557,Umax...
motorVars_M1.adcData	struct HAL_ADCData_t	{VdcBus_V=25.329174,I_A={value=[-3.158...
motorVars_M1.Irms_A	float[3]	[2.46620107,2.47039008,2.47602439]

Click this button to enable periodic capture of data from the microcontroller

Check if these variables meet the board and estimator selections

The sensing conversion value should be equal to the dc bus voltage

Set target speed value (Hz) to this variable

Check if the estimation speed (Hz) is equal/ close to the setting traget speed (Hz)

Set this variable value equal to 1 to start run the motor

Means the inverter/controller has fault when run the motor if the variable value is not zero

The threshold value of the over current protection

Set the reference torque current value to this variable

Tune these Kp or Ki to achieve the required response

Figure 4-23. Build Level 3: Variables in Expressions Window

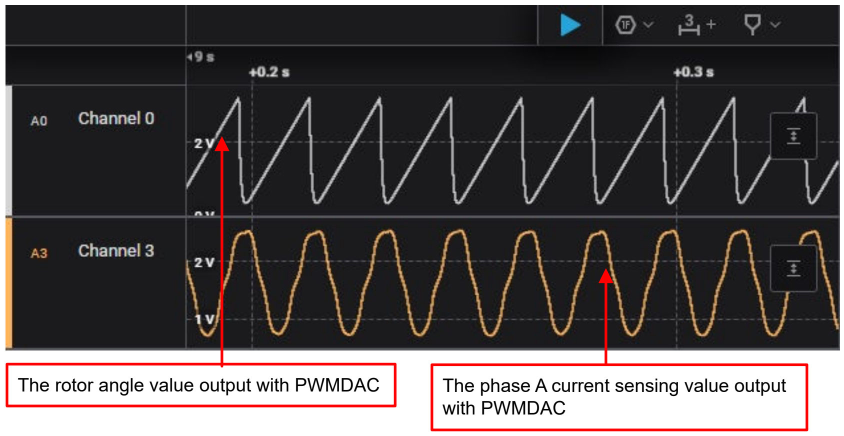

A0
Channel 0
2 V
0 V
A3
Channel 3
2 V
1 V
+0.2 s
+0.3 s
The rotor angle value output with PWMDAC
The phase A current sensing value output with PWMDAC

Figure 4-24. Build Level 3: Motor Rotor Angle and Phase Current Waveforms Monitoring by EPMWDAC

4.4.4 Level 4 Incremental Build

Objectives learned in this build level:

- Evaluate the complete motor drive with eSMO based sensorless-FOC, encoder based sensored-FOC or hall based sensored-FOC.
- Evaluate the additional features, such as field weakening control, flying start, MTPA, and braking.

In this build level, the outer speed loop is closed with the inner current loop for motor that the rotor angle is from eSMO, encoder or Hall sensors modules. The software flow for this build level is shown in Figure 4-25.

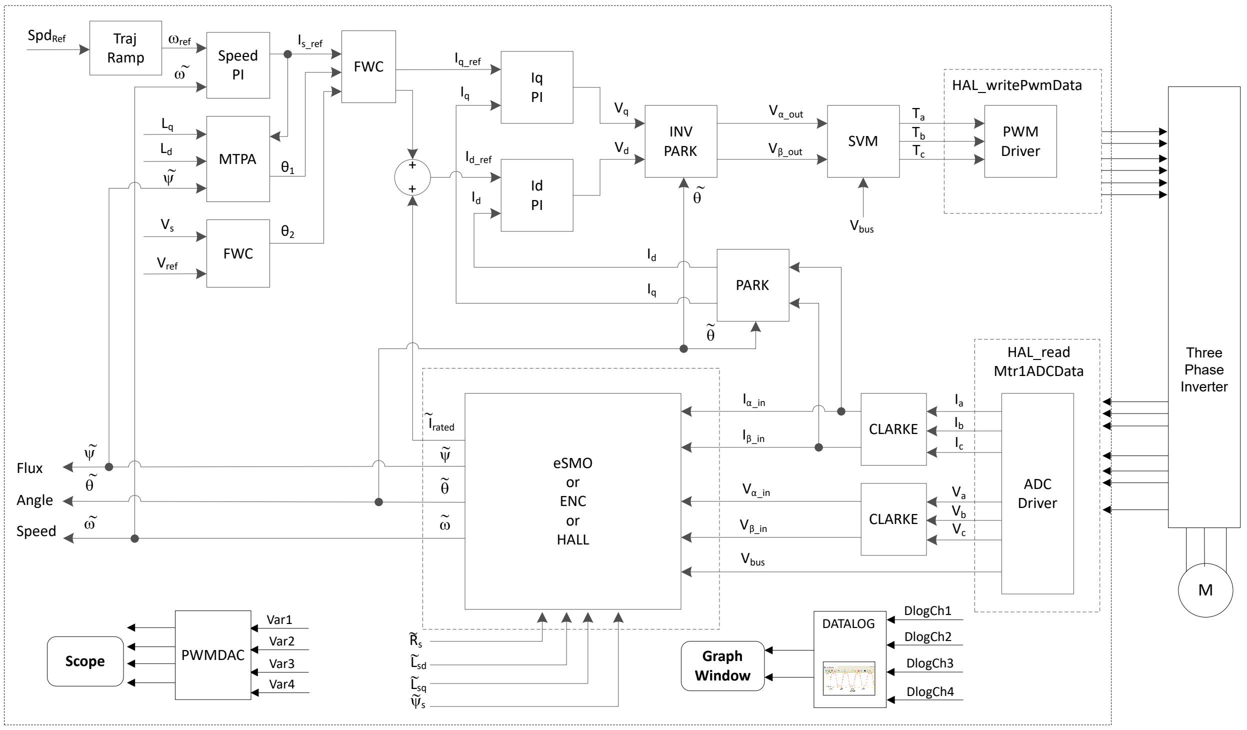


Figure 4-25. Build Level 4 Software Block Diagram - Speed and Current Close Loop Control

4.4.4.1 Build and Load Project

Connect the motor to the related terminals on the power inverter board. Follow the operation steps in Section 4.4.1.1 to build and load project by setting DMC_BUILDLEVEL to DMC_LEVEL_4 in the *sys_settings.h* file.

4.4.4.2 Setup Debug Environment Windows

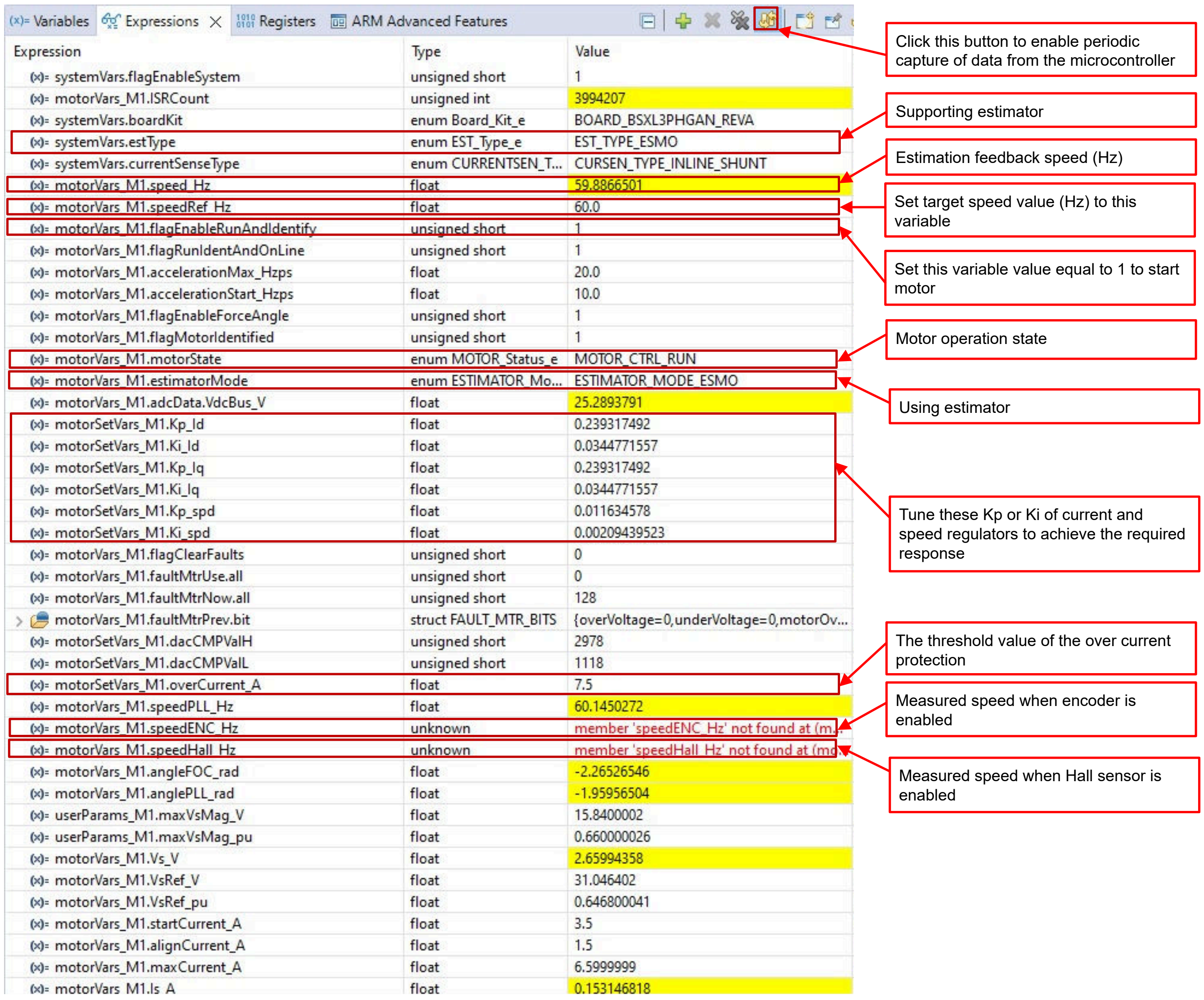
Follow operation steps in Section 4.4.1.2 to import the variables into the Expressions window by picking *universal_motor_control_level4.txt*. The Expressions window appears as shown in Figure 4-26.

4.4.4.3 Run the Code

1. Power on the AC or DC power supply, gradually increase output voltage at power supply to get an appropriate DC-bus voltage.
2. The required motor parameters must be defined in the *user_mtr1.h* header file as shown in the following example code.

```
#define USER_MOTOR1_TYPE MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS (4)
#define USER_MOTOR1_Rr_Ohm (NULL)
#define USER_MOTOR1_Rs_Ohm (0.38157931f)
#define USER_MOTOR1_Ls_d_H (0.000188295482f)
#define USER_MOTOR1_Ls_q_H (0.000188295482f)
#define USER_MOTOR1_RATED_FLUX_VpHz (0.0396642499f)
```

3. Build the project and load the code into the controller, disable the data cache by unchecking the "Data Cache Enabled" in Tools > ARM Advanced Features, run the project by clicking on Resume button, or click *Run → Resume* in the Debug tab. The *systemVars.flagEnableSystem* is set to *1* after a fixed time, that means the offsets calibration have been done and the power relay for inrush is turned on. The fault flags *motorVars_M1.faultMtrUse.all* is equal to *0* , if not, the user must check the current and voltage sensing circuit as described in Section 4.4.1.
4. Set the variable *motorVars_M1.flagEnableRunAndIdentify* to *1* in the Expressions window as shown in Figure 4-26.
5. After you start running the motor:
    a. Set the target speed value to the variable *motorVars_M1.speedRef_Hz* and watch how the motor shaft speed follow the setting speed.
    b. To change the acceleration, enter a different acceleration value for the variable *motorVars_M1.accelerationMax_Hzps*.
    c. Use PWMDAC module to display the monitoring variables as described in Section 4.5.2. The motor angle and current waveforms are shown in Figure 4-27.
6. 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 Watch Window as shown in Figure 4-26 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 relates to the time constant of the speed controlled system.
7. Set the variables *motorVars_M1.flagEnableRunAndIdentify* to *0* to stop run the motor.
8. Once complete, the controller can now be halted and the debug connection terminated. Fully halting the controller by first clicking the Suspend button on the toolbar or by clicking *Target → Halt*. Finally, reset the controller by clicking on CPU Reset button or clicking *Run → Reset*.
9. Close CCS debug session by clicking on Terminate Debug Session button or clicking *Run → Terminate*.



Expression	Type	Value
systemVars.flagEnableSystem	unsigned short	1
motorVars_M1.ISRCount	unsigned int	3994207
systemVars.boardKit	enum Board_Kit_e	BOARD_BSXL3PHGAN_REVA
systemVars.estType	enum EST_Type_e	EST_TYPE_ESMO
systemVars.currentSenseType	enum CURRENTSEN_T...	CURSEN_TYPE_INLINE_SHUNT
motorVars_M1.speed_Hz	float	59.8866501
motorVars_M1.speedRef_Hz	float	60.0
motorVars_M1.flagEnableRunAndIdentify	unsigned short	1
motorVars_M1.flagRunIdentAndOnLine	unsigned short	1
motorVars_M1.accelerationMax_Hzps	float	20.0
motorVars_M1.accelerationStart_Hzps	float	10.0
motorVars_M1.flagEnableForceAngle	unsigned short	1
motorVars_M1.flagMotorIdentified	unsigned short	1
motorVars_M1.motorState	enum MOTOR_Status_e	MOTOR_CTRL_RUN
motorVars_M1.estimatorMode	enum ESTIMATOR_Mo...	ESTIMATOR_MODE_ESMO
motorVars_M1.adcData.VdcBus_V	float	25.2893791
motorSetVars_M1.Kp_Id	float	0.239317492
motorSetVars_M1.Ki_Id	float	0.0344771557
motorSetVars_M1.Kp_Iq	float	0.239317492
motorSetVars_M1.Ki_Iq	float	0.0344771557
motorSetVars_M1.Kp_spd	float	0.011634578
motorSetVars_M1.Ki_spd	float	0.00209439523
motorVars_M1.flagClearFaults	unsigned short	0
motorVars_M1.faultMtrUse.all	unsigned short	0
motorVars_M1.faultMtrNow.all	unsigned short	128
motorVars_M1.faultMtrPrev.bit	struct FAULT_MTR_BITS	{overVoltage=0,underVoltage=0,motorOv...
motorSetVars_M1.dacCMPValH	unsigned short	2978
motorSetVars_M1.dacCMPValL	unsigned short	1118
motorSetVars_M1.overCurrent_A	float	7.5
motorVars_M1.speedPLL_Hz	float	60.1450272
motorVars_M1.speedENC_Hz	unknown	member 'speedENC_Hz' not found at (m...
motorVars_M1.speedHall_Hz	unknown	member 'speedHall_Hz' not found at (mo
motorVars_M1.angleFOC_rad	float	-2.26526546
motorVars_M1.anglePLL_rad	float	-1.95956504
userParams_M1.maxVsMag_V	float	15.8400002
userParams_M1.maxVsMag_pu	float	0.660000026
motorVars_M1.Vs_V	float	2.65994358
motorVars_M1.VsRef_V	float	31.046402
motorVars_M1.VsRef_pu	float	0.646800041
motorVars_M1.startCurrent_A	float	3.5
motorVars_M1.alignCurrent_A	float	1.5
motorVars_M1.maxCurrent_A	float	6.5999999
motorVars_M1.Is_A	float	0.153146818

Click this button to enable periodic capture of data from the microcontroller

Supporting estimator

Estimation feedback speed (Hz)

Set target speed value (Hz) to this variable

Set this variable value equal to 1 to start motor

Motor operation state

Using estimator

Tune these Kp or Ki of current and speed regulators to achieve the required response

The threshold value of the over current protection

Measured speed when encoder is enabled

Measured speed when Hall sensor is enabled

Figure 4-26. Build Level 4: Variables in Expressions Window

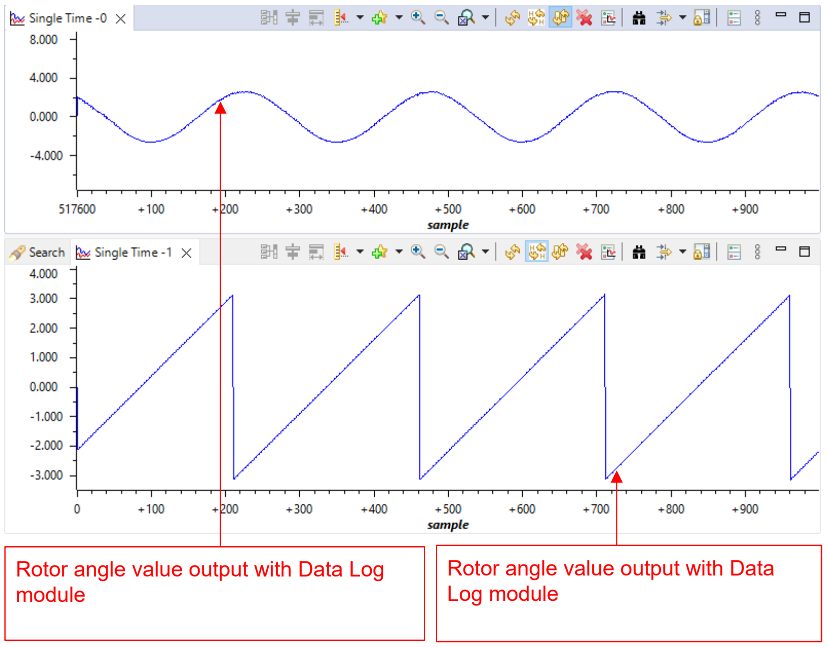


Figure 4-27. Build Level 4: Phase Current and Rotor Angle with eSMO Waveforms at Forward Move

As illustrated in Section 4.2.2, multiple FOC algorithms can be supported in the project. The user can use one (eSMO or Hall or Encoder) algorithm or two algorithms (eSMO + Encoder) for motor control in the project.

The user can implement eSMO and Encoder estimators in the project simultaneously by adding the pre-define name *MOTOR1_ESMO* and *MOTOR1_ENC* in project properties as described in Section 4.2.1. Rebuild, load and run the project as the operation steps above.

- The systemVars.estType value equals to EST_TYPE_ESMO_ENC that means eSMO and Encoder estimators are enabled in this project.
- The motorVars_M1.estimatorMode equals to ESTIMATOR_MODE_ESMO that means the eSMO estimator is using for sensorless-FOC, equals to ESTIMATOR_MODE_ENC that means the encoder estimator is using for sensored-FOC.
- The estimated rotor angles from eSMO and Encoder are shown in Figure 4-28. The motor is running with eSMO at forward rotation by setting motorVars_M1.speedRef_Hz to a positive value.
- The user can change the value to ESTIMATOR_MODE_ENC to select the Encoder estimator for sensored-FOC. And also the user can change the value to switch the using estimator on the fly.

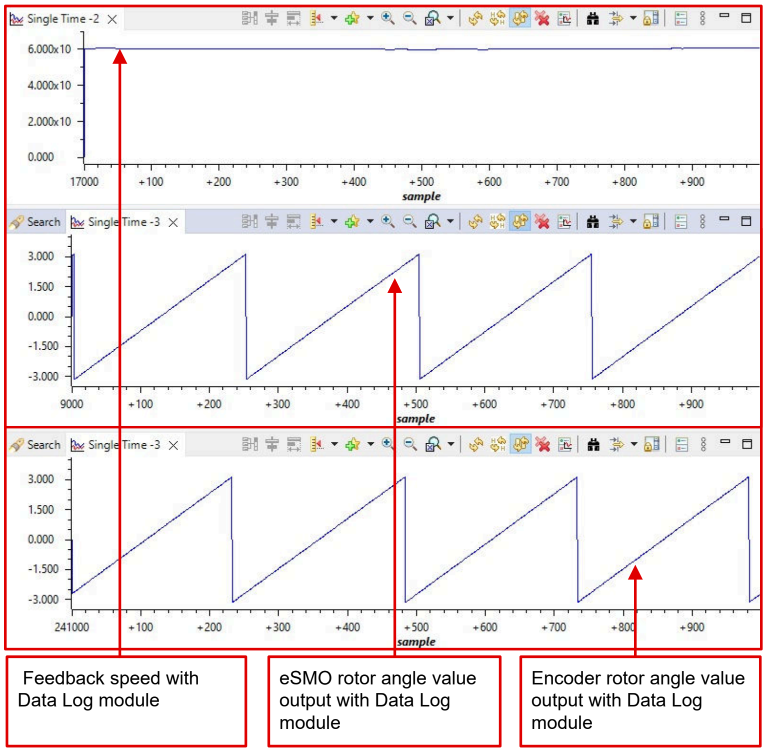


Figure 4-28. Build Level 4: Rotor Angle with eSMO and Encoder, Phase Current Waveforms at Forward Rotation

4.5 Adding Additional Functionality to Motor Control Project

4.5.1 Using DATALOG Function

The DATALOG module stores the real-time values of user selectable software variables (by default four variables) in the data RAM provided on the TI MCU as shown in Figure 4-29. The four variables are selected by configuring the module inputs to the address of the four variables. The starting addresses of the four RAM buffer locations are &((datalog).datalogBuff)[0], &((datalog).datalogBuff)[1], &((datalog).datalogBuff)[2] and &((datalog).datalogBuff)[3]. These Datalog buffers are large arrays that contain value-triggered data that can then be displayed to a graph. The datalog prescalar is configurable, which allows the data log function to only log one out of every prescalar samples. The number of data log buffers, buffer size and data type can be selected in the *datalog_input.h* file.

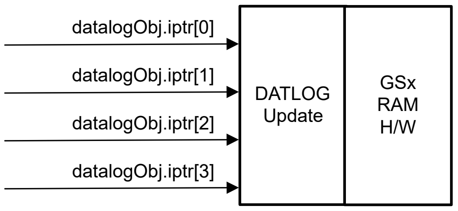


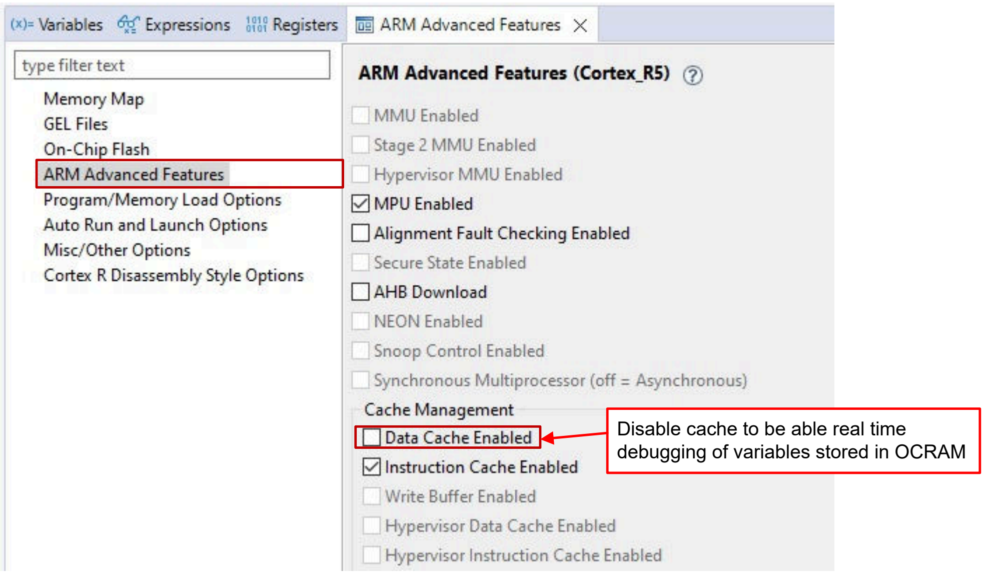
Figure 4-29. DATALOG Module Block Diagram

To enable the datalog functionality, the predefine symbol DATALOG_EN must be added in the project properties as shown in Figure 4-2.

The following code shows the declaration of one DATALOG object and handle. This code is located in the *datalog.c* file.

```
__attribute__ ((section("datalog_data"))) DATALOG_Obj datalog;
DATALOG_Handle datalogHandle;       //!< the handle for the Datalog object
```

This puts data log object in the *datalog_data* section of memory. This section can be either TCM or OCRAM. Generally, we recommend to use OCRAM as TCM has limited size and needed by time critical part of software. Disable Data Cache to be able logging data in OCRAM in CCS12.6. To disable the data cache, *Data Cache Enabled* must be unchecked in the *Tools > ARM Advanced Features* as shown in Figure 4-30.



ARM Advanced Features (Cortex_R5)

- [ ] MMU Enabled
- [ ] Stage 2 MMU Enabled
- [ ] Hypervisor MMU Enabled
- [x] MPU Enabled
- [ ] Alignment Fault Checking Enabled
- [ ] Secure State Enabled
- [ ] AHB Download
- [ ] NEON Enabled
- [ ] Snoop Control Enabled
- [ ] Synchronous Multiprocessor (off = Asynchronous)

Cache Management

- [ ] Data Cache Enabled
- [x] Instruction Cache Enabled
- [ ] Write Buffer Enabled
- [ ] Hypervisor Data Cache Enabled
- [ ] Hypervisor Instruction Cache Enabled

Disable cache to be able real time debugging of variables stored in OCRAM

Figure 4-30. Disable Data Cache for Real Time Debugging

The following code shows the initialization and setting up of the datalog object, handle and parameters. This code is located in the *sys_main.c* file.

```
// Initialize Datalog
datalogHandle = DATALOG_init(&datalog, sizeof(datalog), manual, 0, 1);
DATALOG_Obj *datalogObj = (DATALOG_Obj *)datalogHandle;
```

The following code shows the configuration of the module inputs to point to the address of variables. The datalog module inputs point to different system variables depending on the build level. This code is located in the *sys_main.c* file:

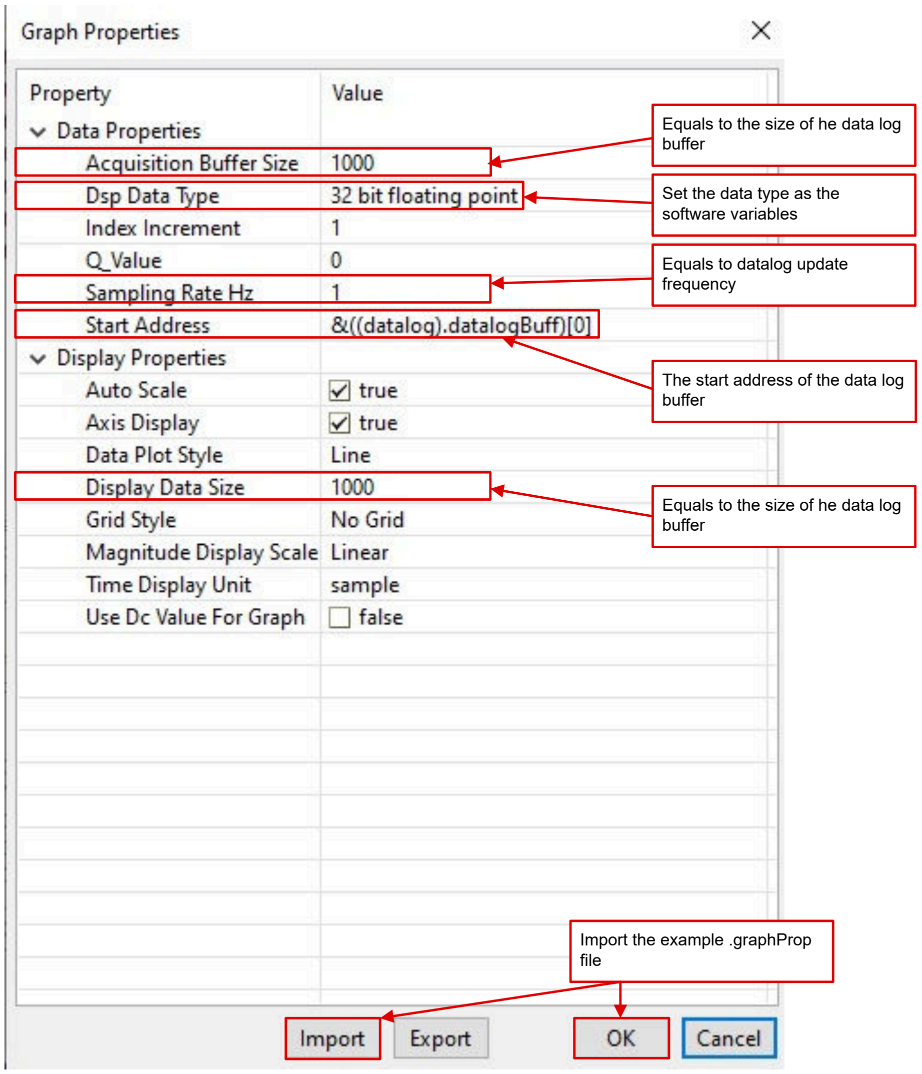
```
datalogObj->iptr[0] = (float32_t*) &motorVars_M1.adcData.V_V.value[0];
datalogObj->iptr[1] = (float32_t*) &motorVars_M1.adcData.I_A.value[0];
datalogObj->iptr[2] = (float32_t*) &motorVars_M1.adcData.I_A.value[1];
datalogObj->iptr[3] = (float32_t*) &motorVars_M1.angleFOC_rad;
```

The following code shows the periodic updating of the datalog buffer with the new data during the execution of the **motor1ctrlISR()** interrupt. This code is located in the *motor1_drive.c* file.

```
#if defined(DATALOG_EN)
DATALOG_update(datalogHandle);
#endif  // DATALOG_EN
```

The datalog module is used with the graph tool, which provides a means to visually inspect the variables and judge system performance. The graph tool is available in CCS, which can display arrays of data in various graphical types. The arrays of data are stored in a device's memory in various formats.

While the project is in debug mode, open and setup time graph windows to plot the data log buffers as shown in Figure 4-31. Alternatively, the user can import the graph configurations files that are located in the project folder. To import them, Click: Tools -> Graph -> Single Time… and select import and browse to the following location `<workspace>\universal_motorcontrol_am263x_r5fss0-0_nortos_ti-arm-clang\src_control\debug\` and select *datalog.graphProp* file. Hit OK, this adds the Graphs to your debug perspective. Click on Continuous Refresh button on the top left corner of the graph tab.



Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	1000
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	&((datalog).datalogBuff)[0]
Display Properties	
Auto Scale	✓ true
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	1000
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	false

Equals to the size of he data log buffer

Set the data type as the software variables

Equals to datalog update frequency

The start address of the data log buffer

Equals to the size of he data log buffer

Import the example .graphProp file

Figure 4-31. Graph Window Settings

4.5.2 Using PWMDAC Function

The PWMDAC module converts the software variables into PWM signals using ePWM 5A, 5B, 6A, and 6B as shown in Figure 4-32. The PWMDAC module is only supported on the high voltage kit (TMDSHVMTRINSPIN) since the module has extra PWM outputs with RC filters available on the board. If the PWMDAC module is used with a motor driver board that does not support the PWMDAC module, then the PWM signals is routed to spare PWMs on the TI LaunchPad and the user needs to add RC filters to those pins to utilize the PWMDAC design.

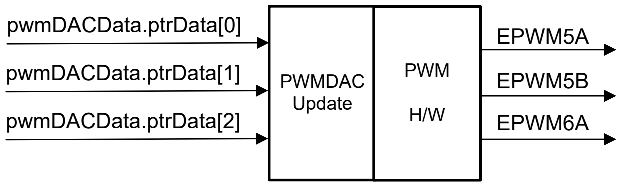


Figure 4-32. PWMDAC Module Block Diagram

The PWMDAC module can be used to view the signal, represented by the variable, at the outputs of the related pins through the external low-pass filters. Therefore, the external low-pass filters are necessary to view the actual signal waveforms as seen in Figure 4-33. The (1st-order) RC low-pass filter is used to filter out the high frequency component embedded in the actual low frequency signals. To select R and C values, the time constant can be expressed in terms of the cut-off frequency (f_c) as shown in the following equations.

$$\tau = RC = \frac{1}{2\pi f_c} \tag{48}$$

$$f_c = 2\pi RC \tag{49}$$

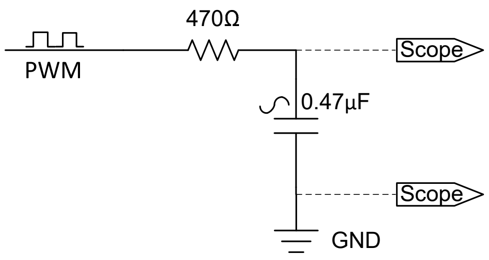


Figure 4-33. External RC Low-Lass Filter Connecting to a PWM Pin

To enable the ePWM DAC functionality, the predefined symbol EPWMDAC_MODE must be added in the project properties as shown in Figure 4-2.

The following code shows the declaration of the PWMDAC object. This code is located in the *sys_main.c* file.

```
#if defined(EPWMDAC_MODE)
#if defined(HVMTRPFC_REV1P1)
__attribute__ ((section("sys_data"))) HAL_PWMDACData_t pwmDACData;
  // HVMTRPFC_REV1P1
#else
#error EPWMDAC is not supported on this kit!
#endif  // !HVMTRPFC_REV1P1
#endif  // EPWMDAC_MODE
```

The following code shows the initialization and setting up of the PWMDAC object, handle and parameters. Four module inputs, ptrData[0], ptrData[1], ptrData[2], and ptrData[3] are configured to point to the addresses of four variables. The PWMDAC module inputs point to different system variables depending on the build level. This code is located in the *sys_main.c* file.

```
// set DAC parameters
pwmDACData.periodMax =
        PWMDAC_getPeriod(halHandle->pwmDACHandle[PWMDAC_NUMBER_1]);

pwmDACData.ptrData[0] = &motorVars_M1.angleFOC_rad;              // PWMDAC1
pwmDACData.ptrData[1] = &motorVars_M1.speedAbs_Hz;               // PWMDAC2
pwmDACData.ptrData[2] = &motorVars_M1.speedAbs_Hz;             // PWMDAC3
pwmDACData.ptrData[3] = &motorVars_M1.adcData.I_A.value[1];    // PWMDAC4

pwmDACData.offset[0] = 0.5f;    // PWMDAC1
pwmDACData.offset[1] = 0.0f;    // PWMDAC2
pwmDACData.offset[1] = 0.0f;    // PWMDAC3
pwmDACData.offset[3] = 0.5f;    // PWMDAC4

pwmDACData.gain[0] = 1.0f / MATH_TWO_PI;                       // PWMDAC1
pwmDACData.gain[1] = 1.0f / USER_MOTOR1_FREQ_MAX_Hz;           // PWMDAC2
pwmDACData.gain[2] = 1.0f / USER_MOTOR1_FREQ_MAX_Hz;           // PWMDAC3
pwmDACData.gain[3] = 2.0f / USER_M1_ADC_FULL_SCALE_CURRENT_A;  // PWMDAC4
```

The following code shows the updating of the PWM outputs with new data during the execution of the **motor1ctrlISR()** interrupt. This code is located in the *motor1_drive.c* file.

```
// connect inputs of the PWMDAC module.
HAL_writePWMDACData(halHandle, &pwmDACData);
```

4.5.3 Adding CAN Functionality

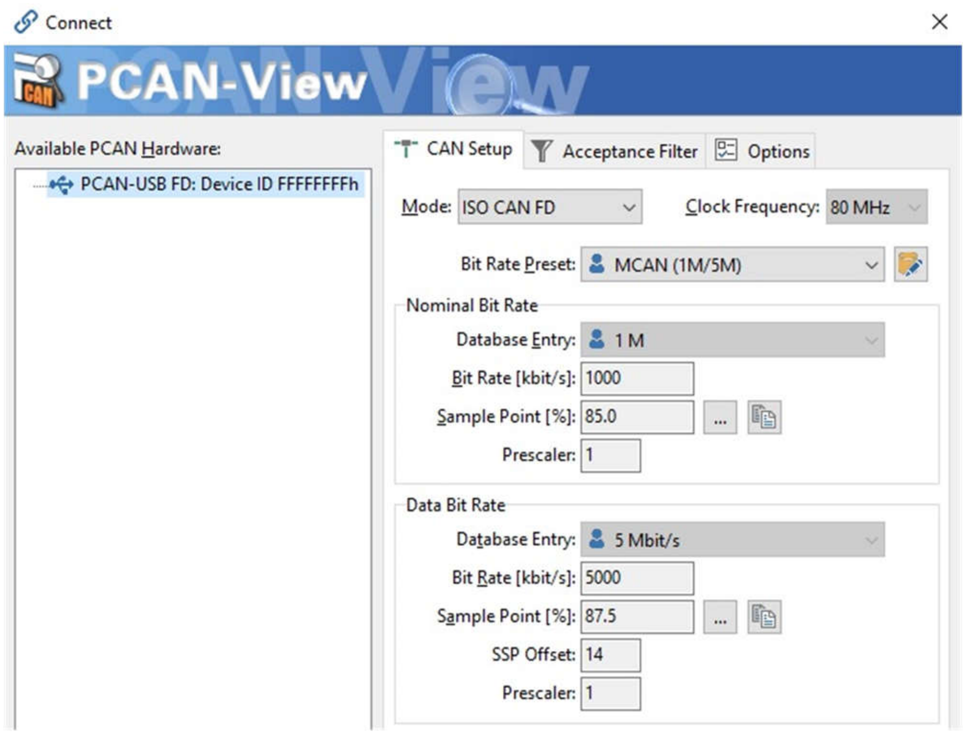
CAN functionality can be added into the lab project to provide the user a communication bus for sending the start/stop command and getting the feedback running states. To utilize this, enable the pre-define symbol CMD_CAN in project build properties as shown in Figure 4-2. PCAN-View is used to simply monitor, transmit, and record CAN data traffic. Different type of command messages can be defined in *motor_common.h*file. In this project as an example, sender specifies a starting command and defines a target speed value. The recipient then receives a message containing the target speed.

Please be noted that depending on whether you are using the LaunchPad or EVM in the kit, certain pin mux settings are required. These configurations are achieved using the *mcanEnableTransceiver* and *tca6416ConfigOutput* functions, as outlined in the following code:

```
#if defined(AM263_CC)
void tca6416ConfigOutput(uint16_t port, uint16_t pin, uint16_t level);
#endif // AM263_CC
```

```
#if defined(CMD_CAN)
#if defined(AM263_LP)

void mcanEnableTransceiver(void)
{
    uint32_t    gpioBaseAddr, pinNum;

    gpioBaseAddr = (uint32_t)AddrTranslateP_getLocalAddr(MCAN_ENABLE_BASE_ADDR);
    pinNum       = MCAN_ENABLE_PIN;

    GPIO_setDirMode(gpioBaseAddr, pinNum, GPIO_DIRECTION_OUTPUT);

    GPIO_pinWriteLow(gpioBaseAddr, pinNum);
}
#endif // AM263_LP

#if defined(AM263_CC)
/* ========================================================================== */
/*                           Macros & Typedefs                                */
/* ========================================================================== */

/* Input status register */
#define TCA6416_REG_INPUT0              ((UInt8) 0x00U)
#define TCA6416_REG_INPUT1              ((UInt8) 0x01U)

/* Output register to change state of output BIT set to 1, output set HIGH */
#define TCA6416_REG_OUTPUT0             ((uint8_t) 0x02U)
#define TCA6416_REG_OUTPUT1             ((uint8_t) 0x03U)

/* Configuration register. BIT = '1' sets port to input, BIT = '0' sets
 * port to output */
#define TCA6416_REG_CONFIG0             ((uint8_t) 0x06U)
#define TCA6416_REG_CONFIG1             ((uint8_t) 0x07U)

/* ========================================================================== */
/*                          Function Declarations                           */
/* ======================================================================== */
static void SetupI2CTransfer(I2C_Handle handle,  uint32_t targetAddr,
                      uint8_t *writeData, uint32_t numWriteBytes,
                      uint8_t *readData,  uint32_t numReadBytes);

void mcanEnableTransceiver(void)
{
    I2C_Handle      i2cHandle;
    uint8_t         dataToSlave[4];

    i2cHandle = gI2cHandle[CONFIG_I2C0];
    dataToSlave[0] = TCA6416_REG_CONFIG0;
    dataToSlave[1] = 0x0U;
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 1, &dataToSlave[1], 1);
    /* set the P00 to 0 make them output ports. */
    dataToSlave[1] &= ~(0x1U);
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 2, NULL, 0);

    /* Get the port values. */
    dataToSlave[0] = TCA6416_REG_INPUT0;
    dataToSlave[1] = 0x0U;
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 1, &dataToSlave[1], 1);

    /* Set P10 and P11 to 0.
     */
    dataToSlave[0] = TCA6416_REG_OUTPUT0;
    dataToSlave[1] &= ~(0x1);
    SetupI2CTransfer(i2cHandle, 0x20, &dataToSlave[0], 2, NULL, 0);
}

static void SetupI2CTransfer(I2C_Handle handle,  uint32_t targetAddr,
                      uint8_t *writeData, uint32_t numWriteBytes,
                      uint8_t *readData,  uint32_t numReadBytes)
{
    int32_t status;
    I2C_Transaction i2cTransaction;

    /* Enable Transceiver */
    I2C_Transaction_init(&i2cTransaction);
    i2cTransaction.targetAddress = targetAddr;
    i2cTransaction.writeBuf = (uint8_t *)&writeData[0];
    i2cTransaction.writeCount = numWriteBytes;
    i2cTransaction.readBuf = (uint8_t *)&readData[0];
    i2cTransaction.readCount = numReadBytes;
    status = I2C_transfer(handle, &i2cTransaction);
    DebugP_assert(SystemP_SUCCESS == status);
}

#endif // AM263_CC
#endif // CMD_CAN
```

After launching "PCAN-View", make setup the CAN adaptor as Figure 4-34:



Available PCAN Hardware:
PCAN-USB FD: Device ID FFFFFFFFh

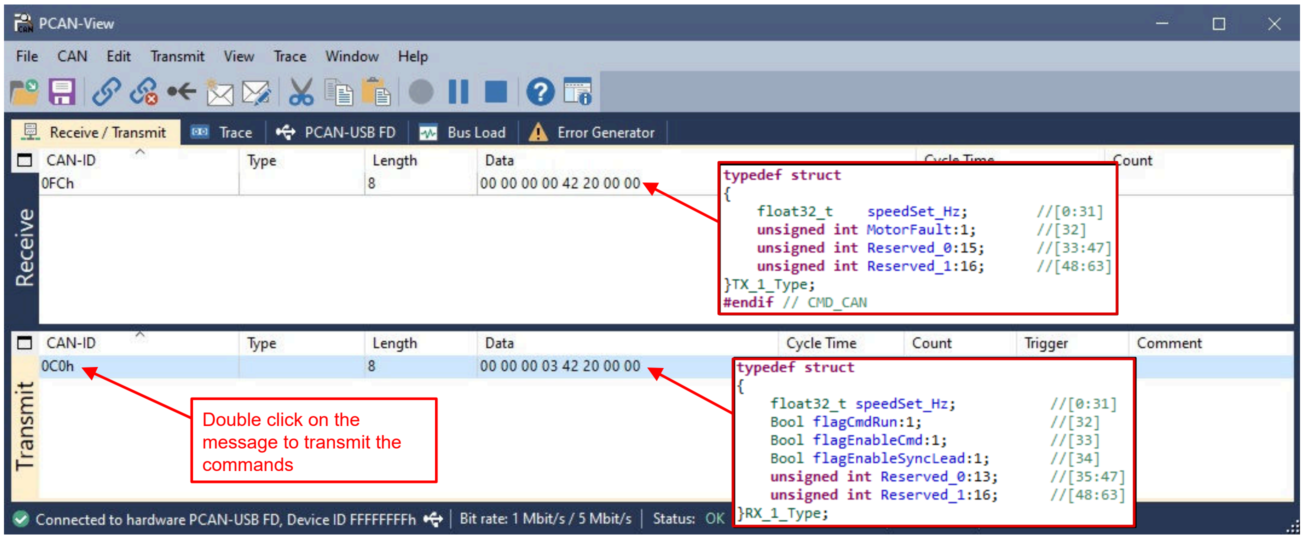
Mode: ISO CAN FD
Clock Frequency: 80 MHz
Bit Rate Preset: MCAN (1M/5M)

Nominal Bit Rate
Database Entry: 1 M
Bit Rate [kbit/s]: 1000
Sample Point [%]: 85.0
Prescaler: 1

Data Bit Rate
Database Entry: 5 Mbit/s
Bit Rate [kbit/s]: 5000
Sample Point [%]: 87.5
SSP Offset: 14
Prescaler: 1

Figure 4-34. PCAN-View Setup

Double click to initiate the CAN data transmission as shown in Figure 4-35.



CAN-ID	Type	Length	Data	Cycle Time	Count
0FCh		8	00 00 00 00 42 20 00 00		

```
typedef struct
{
    float32_t    speedSet_Hz;          //[0:31]
    unsigned int MotorFault:1;         //[32]
    unsigned int Reserved_0:15;        //[33:47]
    unsigned int Reserved_1:16;        //[48:63]
}TX_1_Type;
#endif // CMD_CAN
```

CAN-ID	Type	Length	Data	Cycle Time	Count	Trigger	Comment
0C0h		8	00 00 00 03 42 20 00 00				

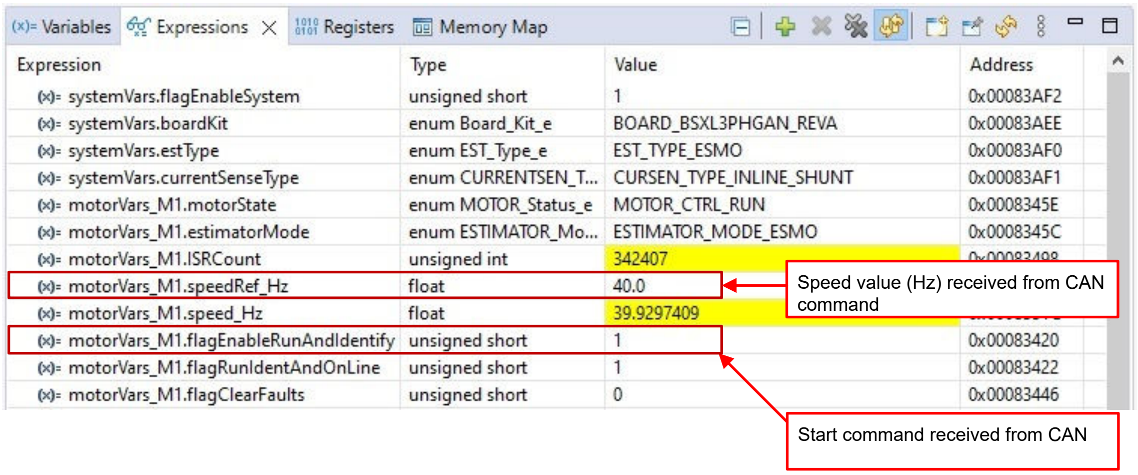
Double click on the message to transmit the commands

```
typedef struct
{
    float32_t speedSet_Hz;             //[0:31]
    Bool flagCmdRun:1;                 //[32]
    Bool flagEnableCmd:1;              //[33]
    Bool flagEnableSyncLead:1;         //[34]
    unsigned int Reserved_0:13;        //[35:47]
    unsigned int Reserved_1:16;        //[48:63]
}RX_1_Type;
```

Connected to hardware PCAN-USB FD, Device ID FFFFFFFFh | Bit rate: 1 Mbit/s / 5 Mbit/s | Status: OK

Figure 4-35. CAN Data Transmission Start

As shown in Figure 4-36, the *motorVars_M1.flagEnableRunAndIdentify* is configured with a value of 1, and the *motorVars_M1.speedRef_Hz* is set to 40Hz. This speed value is then transmitted back to the recipient through the CAN communication.


Variables
Expressions
Registers
Memory Map
Speed value (Hz) received from CAN command
Start command received from CAN

Expression	Type	Value	Address
systemVars.flagEnableSystem	unsigned short	1	0x00083AF2
systemVars.boardKit	enum Board_Kit_e	BOARD_BSXL3PHGAN_REVA	0x00083AEE
systemVars.estType	enum EST_Type_e	EST_TYPE_ESMO	0x00083AF0
systemVars.currentSenseType	enum CURRENTSEN_T...	CURSEN_TYPE_INLINE_SHUNT	0x00083AF1
motorVars_M1.motorState	enum MOTOR_Status_e	MOTOR_CTRL_RUN	0x0008345E
motorVars_M1.estimatorMode	enum ESTIMATOR_Mo...	ESTIMATOR_MODE_ESMO	0x0008345C
motorVars_M1.ISRCount	unsigned int	342407	0x00083498
motorVars_M1.speedRef_Hz	float	40.0	
motorVars_M1.speed_Hz	float	39.9297409	[illegible]
motorVars_M1.flagEnableRunAndIdentify	unsigned short	1	0x00083420
motorVars_M1.flagRunIdentAndOnLine	unsigned short	1	0x00083422
motorVars_M1.flagClearFaults	unsigned short	0	0x00083446

Figure 4-36. CAN Command: Variables in Expressions Window

4.5.4 Adding SFRA Functionality

Texas Instruments' software frequency response analyzer (SFRA) library is designed to enable frequency response analysis on power converters using software only and without the need for an external frequency response analyzer. The optimized library can be used in high frequency power conversion applications to identify the plant, the closed loop and the open loop gain characteristics of a closed loop power converter, which can be used to get stability information such as gain margin, phase margin and open loop gain crossover frequency, to evaluate the control loop performance.

Consider a digitally controlled closed loop power converter, as shown in Figure 4-37, where:

- H is the transfer function of the plant that needs to be controlled
- G is the digital compensator
- GH is referred to as the open loop transfer function
- CL is referred to as the closed loop transfer function and is GH/(1+GH)
- r is the instantaneous set point or the reference of the converter
- Ref is the DC set point reference
- y the analog-to-digital converter (ADC) feedback
- e the instantaneous error
- d the sensor noise and disturbance
- u the PWM duty cycle

The key objectives of the compensator in a closed loop system can be summarized as:

- Make sure that the system is stable (for example, the system tracks the reference asymptotically)

$$e = \lim_{t \to \infty} e(t) = \lim_{t \to \infty} \frac{r}{(1+GH)} \to 0 \tag{50}$$

- System provides disturbance rejection to maintain robust operation

$$S = \frac{y}{d} = \frac{1}{1+GH} \to 0 \tag{51}$$

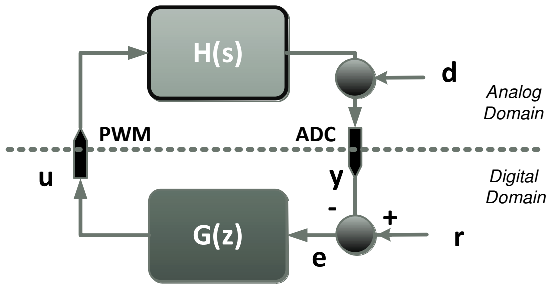


Figure 4-37. Digitally Controlled Power Converter

Whether or not the system meets the objectives can be determined by knowing the open loop transfer function (GH), as shown in Equation 50 and Equation 51.

A Bode plot of the open loop transfer function GH is frequently used for this purpose and quantities such as gain margin (GM), phase margin (PM) and open loop gain crossover frequency (Folg_cf) are often used to comment on the stability and robustness of a closed loop power converter.

The closed loop transfer function (GH/(1+GH)) provides an idea of the tracking that is how good the system is able to track to the reference commanded.

The SFRA library can enable measurements of the GH, GH/(1+GH) and H frequency response by software. This data can be used to:

- Verify the plant model (H) or extract the plant model (H)
- Design a compensator (G) for the closed loop plant
- Verify the close loop performance of the system by plotting the open loop (GH) or Closed Loop (GH/(1+GH)) Bode diagram

As the frequency response of GH and H carry information of the plant, the data can be used to comment on the health of the power stage by periodically measuring the frequency response.

The SFRA library is based on sinusoidal injection principle, where the assumption is that the injection amplitude causes very small deviation to the normal operating point of the converter. The SFRA library can be integrated into the control code of the power converter, this document details the steps to do so. All computations for the GH, H and CL calculations are done on the MCU and the entire arrays of the GH, H and CL magnitude and phase response are stored on the controller.

Once integrated into the code, the SFRA library can be used to design or fine tune the controller. For this, a typical flow of using SFRA library is:

1. Initiate a SFRA sweep in open loop and store the data in an excel file. This information can then be used to identify the plant model for the steady state operating point at which the SFRA sweep has been conducted.
2. The MATLAB® script provided with this project can be used to read that data into MATLAB and then curve fit the response to a transfer function. Sisotool can then be used to design the compensator.
3. New compensator values can be copied from the MATLAB into the Code Composer Studio™ project.
4. Compile and load the code with new coefficients into the microcontroller controlling the power stage. SFRA algorithm (Step 1) can be re-run to verify the closed loop system performance by measuring the open loop gain GH (also referred to as loop gain in literature).

In summary, TI's software frequency response analyzer provides a methodology to tune power converters in a systematic way and enables quick and easy frequency response analysis for power converters without the need of external connections and equipment. Since no external connections are used, the SFRA can be run repeatedly to periodically assess the health of the power converter and get diagnostic information.

4.5.4.1 Principle of Operation

The software frequency response analyzer is based on the principle of small signal sinusoidal injection. A small signal is injected on the reference of the controller, as shown in Figure 4-38, and the frequency response on feedback and controller outputs are calculated. This provides the plant frequency response characteristics and the open loop frequency response of the closed loop system.

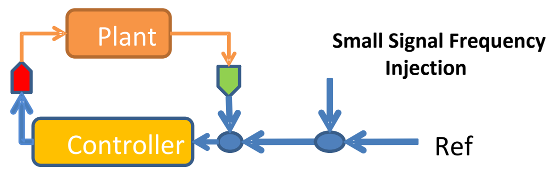


Figure 4-38. SFRA Principle of Operation

4.5.4.2 Object Definition

The SFRA library defines the floating-point-based SFRA structure as discussed in the following:

```
typedef struct{
  float32_t *h_magVect;     //!< Plant Mag SFRA Vector
  float32_t *h_phaseVect;   //!< Plant Phase SFRA Vector
  float32_t *gh_magVect;    //!< Open Loop Mag SFRA Vector
  float32_t *gh_phaseVect;  //!< Open Loop Phase SFRA Vector
  float32_t *cl_magVect;    //!< Closed Loop Mag SFRA Vector
  float32_t *cl_phaseVect;  //!< Closed Loop Phase SFRA Vector
  float32_t *freqVect;      //!< Frequency Vector
  float32_t amplitude;      //!< Injection Amplitude
  float32_t isrFreq;        //!< SFRA ISR frequency
  float32_t freqStart;      //!< Start frequency of SFRA sweep
  float32_t freqStep;       //!< Log space between frequency points (optional)
  int16_t start;            //!< Command to start SFRA
  int16_t state;            //!< State of SFRA
  int16_t status;           //!< Status of SFRA
  int16_t vecLength;        //!< No. of Points in the SFRA
  int16_t freqIndex;        //!< Index of the frequency vector
  int16_t storeH;           //!< Flag to indicate if H vector is stored
  int16_t storeGH;          //!< Flag to indicate if GH vector is stored
  int16_t storeCL;          //!< Flag to indicate if CL vector is stored
  int16_t speed;            //!< variable to change the speed of the sweep
}SFRA_F32;
```

4.5.4.3 Module Interface Definition

Table 4-7. Floating Point Module Interface Definition

Module Element Name	Type	Description	Acceptable Range
h_magVect,gh_magVect, cl_magVect	Input	Pointer to the array that stores the magnitude of H, GH and CL measurements by SFRA . Pass NULL if you do not want SFRA to save that vector.	Pointer to 32 bit location, the location stores the value of the magnitude vectors in single precision (32-bit) floating point
h_phaseVect,gh_phaseVect, cl_phaseVect	Input	Pointer to the array that stores the phase of H, GH and CL measurements by SFRA . Pass NULL if you do not want SFRA to save that vector.	Pointer to 32 bit location, the location stores the value of the phase vectors in single precision (32-bit) floating point
freqVect	Input	Pointer to array of frequency values at which SFRA is performed.	Pointer to 32 bit location, the location stores the value of the frequency vectors in single precision (32-bit) floating point
amplitude	Input	Amplitude of small signal injection in pu.	Single precision (32-bit) floating point(-1,1)
isrFreq	Input	Frequency at which SFRA routine is called.	Single precision (32-bit) floating point
freqStart	Input	Frequency of the first frequency sweep data point.	Single precision (32-bit) floating point
freqStep	Input	10^(1/(no of steps per decade)).	Single precision (32-bit) floating point
start	Input	Command to start SFRA.	int16_t
state	Output	SFRA state. Non zero when SFRA injection is in progress, '0' if SFRA injection is not active/ in progress.	int16_t
status	Output	SFRA status. '1' is SFRA injection is in progress, '0' if SFRA injection is not active/ in progress.	int16_t
vecLength	Input	No of points for which SFRA is performed.	int16_t

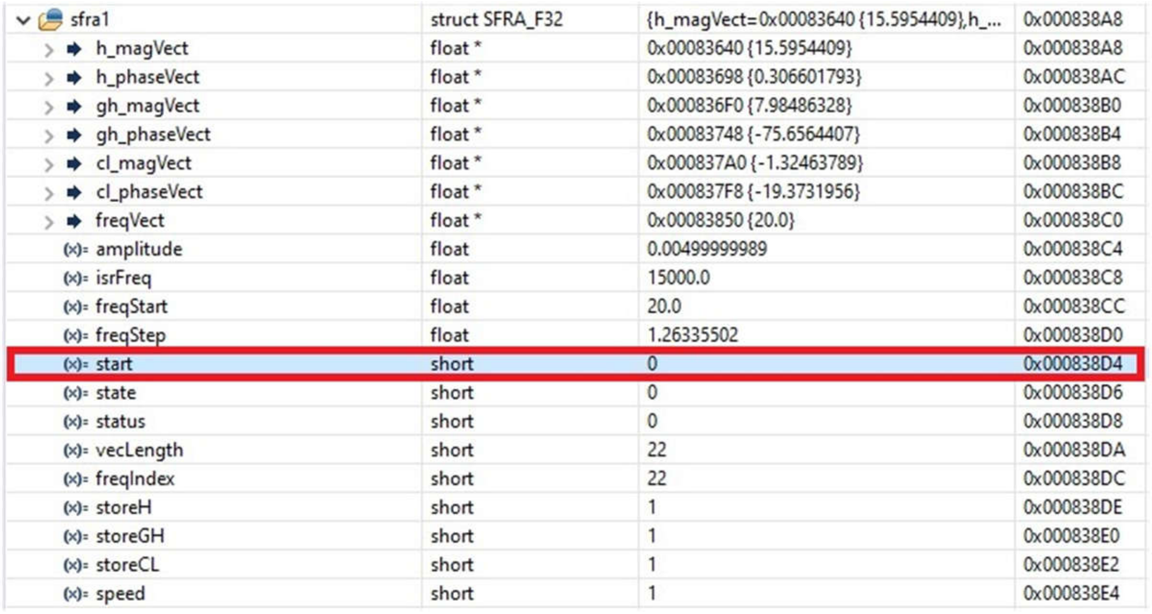
Table 4-7. Floating Point Module Interface Definition (continued)

Module Element Name	Type	Description	Acceptable Range
freqIndex	Output	Frequency index number of freqVect at which SFRA is being performed.	int16_t (0-vecLength)
storeH	Output	Reflects the SFRA configuration, If one, H vector is stored If zero, H vector is not stored this happens when a NULL vector is passed for H mag or phase vector during SFRA configuration.	int16_t (0 or 1)
storeGH	Output	Reflects the SFRA configuration, If one, GH vector is stored If zero, GH vector is not stored this happens when a NULL vector is passed for GH mag or phase vector during SFRA configuration.	int16_t (0 or 1)
storeCL	Output	Reflects the SFRA configuration, If one, CL vector is stored If zero, CL vector is not stored this happens when a NULL vector is passed for CL mag or phase vector during SFRA configuration.	int16_t (0 or 1)
speed	Input	Used to change the speed of the sweep, need to be greater than 1. With 1 the STB example template sweep takes about 58 seconds. Actual speed in the system depends on the frequency point being measured and the ISR rate used for calling the SFRA module. Higher the speed number the slower the sweep.	int16_t (Greater than 1)

4.5.4.4 Using SFRA

Use the following steps to integrate the SFRA in the project:

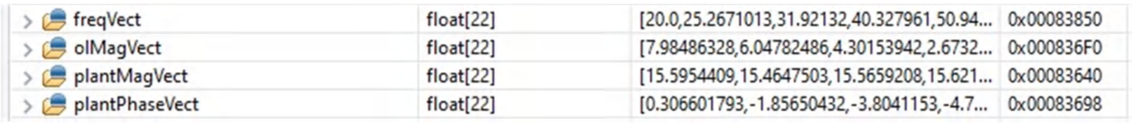
1. To enable the SFRA functionality, the predefine symbol SFRA_ENABLE must be added in the project properties as shown in Figure 4-2.
2. To start an SFRA sweep, put the SFRA object in the watch window.
3. Write SFRA_OBJ.start to 1, when you want the SFRA sweep to start as shown in Figure 4-39.



sfra1	struct SFRA_F32	{h_magVect=0x00083640 {15.5954409},h_...	0x000838A8
h_magVect	float *	0x00083640 {15.5954409}	0x000838A8
h_phaseVect	float *	0x00083698 {0.306601793}	0x000838AC
gh_magVect	float *	0x000836F0 {7.98486328}	0x000838B0
gh_phaseVect	float *	0x00083748 {-75.6564407}	0x000838B4
cl_magVect	float *	0x000837A0 {-1.32463789}	0x000838B8
cl_phaseVect	float *	0x000837F8 {-19.3731956}	0x000838BC
freqVect	float *	0x00083850 {20.0}	0x000838C0
amplitude	float	0.00499999989	0x000838C4
isrFreq	float	15000.0	0x000838C8
freqStart	float	20.0	0x000838CC
freqStep	float	1.26335502	0x000838D0
start	short	0	0x000838D4
state	short	0	0x000838D6
status	short	0	0x000838D8
vecLength	short	22	0x000838DA
freqIndex	short	22	0x000838DC
storeH	short	1	0x000838DE
storeGH	short	1	0x000838E0
storeCL	short	1	0x000838E2
speed	short	1	0x000838E4

Figure 4-39. Start SFRA Functionality

4. Monitor the SFRA_OBJ.FreqIndex variable; the variable gradually increments as SFRA sweep is performed.
5. Once the SFRA_OBJ.FreqIndex reaches Vec_Length, the SFRA sweep is complete.



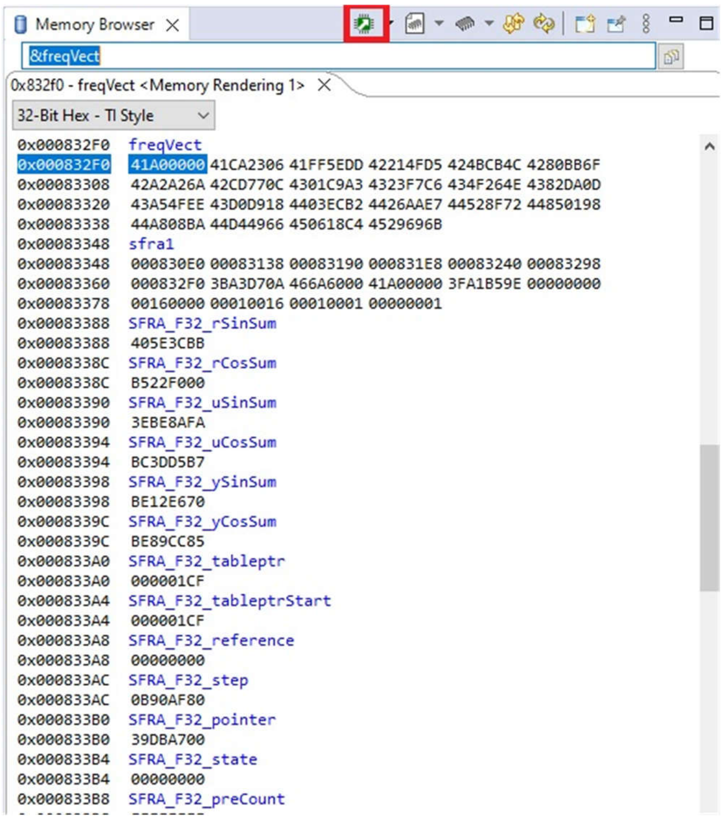
freqVect	float[22]	[20.0,25.2671013,31.92132,40.327961,50.94...	0x00083850
olMagVect	float[22]	[7.98486328,6.04782486,4.30153942,2.6732...	0x000836F0
plantMagVect	float[22]	[15.5954409,15.4647503,15.5659208,15.621...	0x00083640
plantPhaseVect	float[22]	[0.306601793,-1.85650432,-3.8041153,-4.7...	0x00083698

Figure 4-40. SFRA Data Arrays

6. As part of the SFRA initialization, the Open Loop and Plant Magnitude and phase are stored in arrays called.

```
__attribute__ ((section(".sfradata"))) float32_t plantMagVect[SFRA_FREQ_LENGTH];
__attribute__ ((section(".sfradata"))) float32_t plantPhaseVect[SFRA_FREQ_LENGTH];
__attribute__ ((section(".sfradata"))) float32_t olMagVect[SFRA_FREQ_LENGTH];
__attribute__ ((section(".sfradata"))) float32_t olPhaseVect[SFRA_FREQ_LENGTH];
__attribute__ ((section(".sfradata"))) float32_t freqVect[SFRA_FREQ_LENGTH];
```

7. Put these in the watch window to inspect and study the response.
8. Once the sweep is complete, click on View-> MemoryBrowser inside CCS.
9. Inside Memory Browser, enter &freqVect to see the frequency vector and select 32-bit floating point,

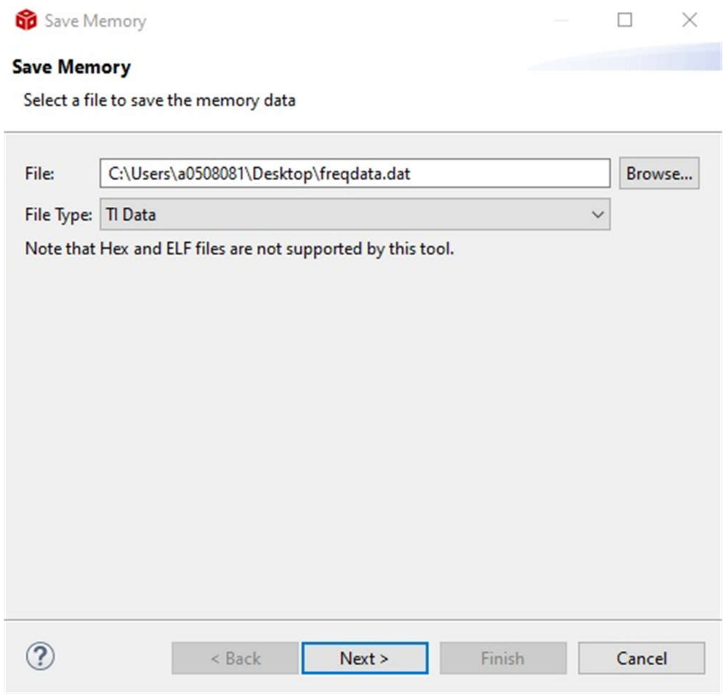


```
0x000832F0  freqVect
0x000832F0  41A00000 41CA2306 41FF5EDD 42214FD5 424BCB4C 4280BB6F
0x00083308  42A2A26A 42CD770C 4301C9A3 4323F7C6 434F264E 4382DA0D
0x00083320  43A54FEE 43D0D918 4403ECB2 4426AAE7 44528F72 44850198
0x00083338  44A808BA 44D44966 450618C4 4529696B
0x00083348  sfra1
0x00083348  000830E0 00083138 00083190 000831E8 00083240 00083298
0x00083360  000832F0 3BA3D70A 466A6000 41A00000 3FA1B59E 00000000
0x00083378  00160000 00010016 00010001 00000001
0x00083388  SFRA_F32_rSinSum
0x00083388  405E3CBB
0x0008338C  SFRA_F32_rCosSum
0x0008338C  B522F000
0x00083390  SFRA_F32_uSinSum
0x00083390  3EBE8AFA
0x00083394  SFRA_F32_uCosSum
0x00083394  BC3DD5B7
0x00083398  SFRA_F32_ySinSum
0x00083398  BE12E670
0x0008339C  SFRA_F32_yCosSum
0x0008339C  BE89CC85
0x000833A0  SFRA_F32_tableptr
0x000833A0  000001CF
0x000833A4  SFRA_F32_tableptrStart
0x000833A4  000001CF
0x000833A8  SFRA_F32_reference
0x000833A8  00000000
0x000833AC  SFRA_F32_step
0x000833AC  0B90AF80
0x000833B0  SFRA_F32_pointer
0x000833B0  39DBA700
0x000833B4  SFRA_F32_state
0x000833B4  00000000
0x000833B8  SFRA_F32_preCount
```

Figure 4-41. Memory Browser View of Stored SFRA Vectors

10. Click on save memory, shown encircled in Figure 4-41.

11. A pop-up window appears. Select TI data and specify the file name *.dat in the location you prefer.



Save Memory

Select a file to save the memory data

File: C:\Users\a0508081\Desktop\freqdata.dat

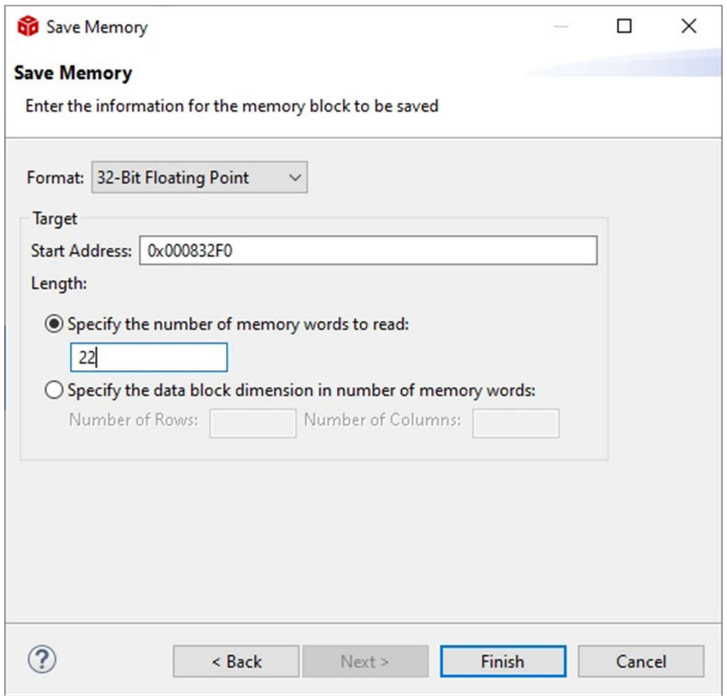
File Type: TI Data

Note that Hex and ELF files are not supported by this tool.

Figure 4-42. Save Memory Pop-Up Window

12. Click on Next and specify the address from the memory browser for the start of the array and then the length.

13. Make sure 32-bit floating point is selected. Click Finish.



Save Memory

Enter the information for the memory block to be saved

Format: 32-Bit Floating Point

Target

Start Address: 0x000832F0

Length:

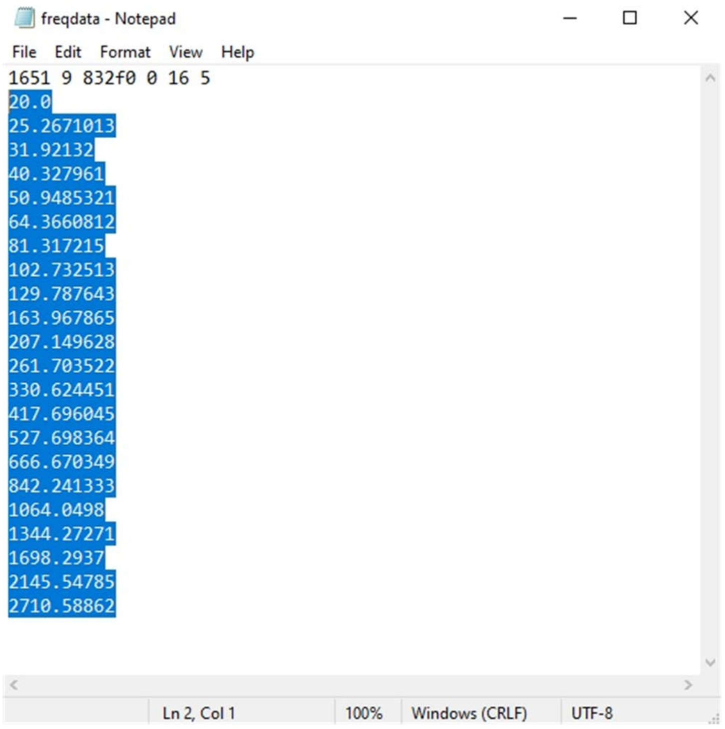
(●) Specify the number of memory words to read: 22

( ) Specify the data block dimension in number of memory words:

Number of Rows: Number of Columns:

Figure 4-43. Save Memory Options

14. This saves the data in *.dat file.
15. Repeat this step for plantMagVect, plantPhaseVect, olMagVect, olPhaseVect, so you have 5 *.dat files.
16. If you want to use this data in MATLAB or other tools, the data can be populated to an excel file.
17. Open the SFRA.xlsx file located at <project directory>\libraries\SFRA\scripts in excel.
18. You can choose to re-name and save the file.
19. This excel sheet has five columns, in the first column is the frequency data.
20. Open the *dat file that was saved.

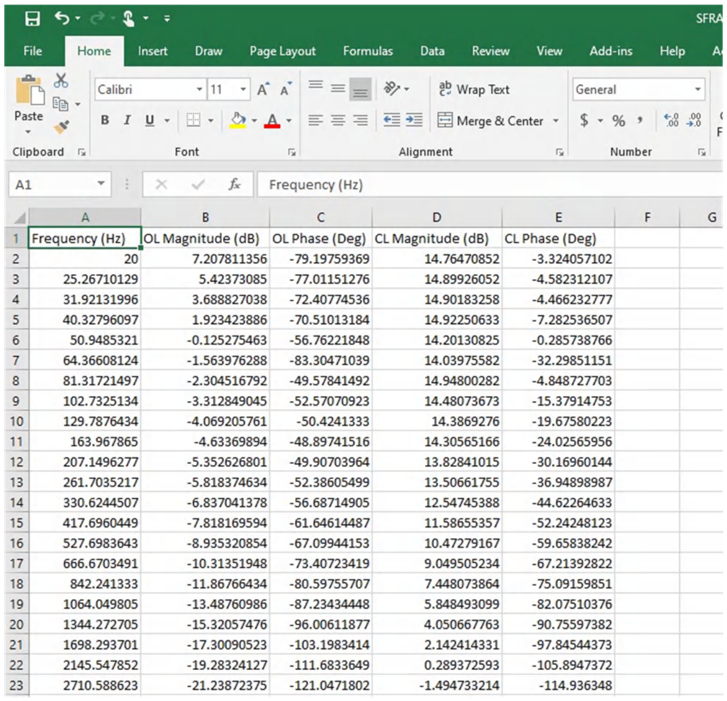


```
1651 9 832f0 0 16 5
20.0
25.2671013
31.92132
40.327961
50.9485321
64.3660812
81.317215
102.732513
129.787643
163.967865
207.149628
261.703522
330.624451
417.696045
527.698364
666.670349
842.241333
1064.0498
1344.27271
1698.2937
2145.54785
2710.58862
```

Figure 4-44. Selecting Data From .dat File to Put in the Excel

21. Select the data from the second line onwards to the end of the file and do Ctrl+C to copy the data.

22. Open the Excel File, go to the first element under the corresponding vector and do Ctrl+V to copy the array.



	A	B	C	D	E
1	Frequency (Hz)	OL Magnitude (dB)	OL Phase (Deg)	CL Magnitude (dB)	CL Phase (Deg)
2	20	7.207811356	-79.19759369	14.76470852	-3.324057102
3	25.26710129	5.42373085	-77.01151276	14.89926052	-4.582312107
4	31.92131996	3.688827038	-72.40774536	14.90183258	-4.466232777
5	40.32796097	1.923423886	-70.51013184	14.92250633	-7.282536507
6	50.9485321	-0.125275463	-56.76221848	14.20130825	-0.285738766
7	64.36608124	-1.563976288	-83.30471039	14.03975582	-32.29851151
8	81.31721497	-2.304516792	-49.57841492	14.94800282	-4.848727703
9	102.7325134	-3.312849045	-52.57070923	14.48073673	-15.37914753
10	129.7876434	-4.069205761	-50.4241333	14.3869276	-19.67580223
11	163.967865	-4.63369894	-48.89741516	14.30565166	-24.02565956
12	207.1496277	-5.352626801	-49.90703964	13.82841015	-30.16960144
13	261.7035217	-5.818374634	-52.38605499	13.50661755	-36.94898987
14	330.6244507	-6.837041378	-56.68714905	12.54745388	-44.62264633
15	417.6960449	-7.818169594	-61.64614487	11.58655357	-52.24248123
16	527.6983643	-8.935320854	-67.09944153	10.47279167	-59.65838242
17	666.6703491	-10.31351948	-73.40723419	9.049505234	-67.21392822
18	842.241333	-11.86766434	-80.59755707	7.448073864	-75.09159851
19	1064.049805	-13.48760986	-87.23434448	5.848493099	-82.07510376
20	1344.272705	-15.32057476	-96.00611877	4.050667763	-90.75597382
21	1698.293701	-17.30090523	-103.1983414	2.142414331	-97.84544373
22	2145.547852	-19.28324127	-111.6833649	0.289372593	-105.8947372
23	2710.588623	-21.23872375	-121.0471802	-1.494733214	-114.936348

Figure 4-45. SFRA Data Copied in Excel File

23. Repeat the steps for each column.
24. Once the excel file is updated for all five columns, use the MATLAB script to import the SFRA data. Then, use the script inside sisotool to design compensator and carry out stability analysis.

4.6 Building a Custom Board

4.6.1 Building a New Custom Board

This section discusses how the user can design an application board to drive a motor, and how to migrate this project for use with their own board.

4.6.1.1 Hardware Setup

If using a custom board, make sure that the power supply to the microcontroller and to the gate driver is correct, and that the JTAG emulator can be connected successfully. Modify the reference code to be compatible with the custom board as described in the following sections, and then run the code starting with build level 1 and working the way to build level 4 as shown in Section 4.4.

4.6.1.2 Migrating Reference Code to a Custom Board

To migrate the reference code to a new TI motor driver kit or to a custom board, the user needs to configure the hardware parameters and the motor control parameters in the *user_mtr1.h* file according to the motor driver circuit, and configure the relevant peripherals in the *AM263_xxx.syscfg*, *hal.h* and *hal.c* files as described in the following sections.

The following block diagram summarizes the function calls that are used to configure the motor control settings and the TI MCU peripherals (Figure 4-46).

In this project, there are several HAL functions that are called only once, related to the configuration of the Hardware. All of these functions deal with the configuration of either a peripheral or of a motor driver IC.

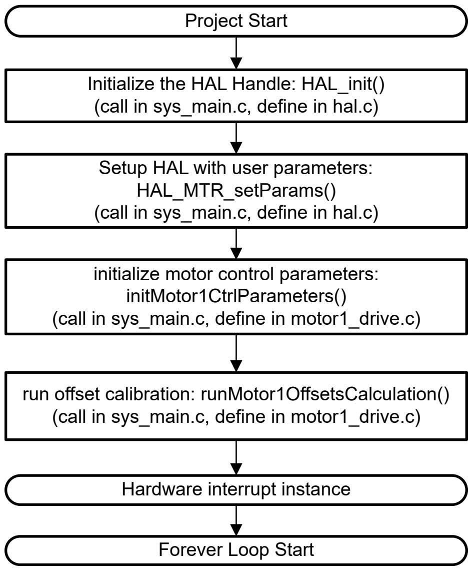

Project Start
Initialize the HAL Handle: HAL_init()
(call in sys_main.c, define in hal.c)
Setup HAL with user parameters:
HAL_MTR_setParams()
(call in sys_main.c, define in hal.c)
initialize motor control parameters:
initMotor1CtrlParameters()
(call in sys_main.c, define in motor1_drive.c)
run offset calibration: runMotor1OffsetsCalculation()
(call in sys_main.c, define in motor1_drive.c)
Hardware interrupt instance
Forever Loop Start

Figure 4-46. HAL Configuration and Motor Control Setting Block Diagram

4.6.1.2.1 Setting Hardware Board Parameters

The *user_mtr1.h* file is where all the user parameters are stored for motor control. The maximum phase current and phase voltage at the input to the AD converter are hardware dependent and must be based on the current and voltage sensing circuitry and scaling at the ADC input. The number of phase current sensors and phase voltage sensors are also defined in the *user_mtr1.h* file. These values are hardware-dependent.

All of the configurable parameters defined in the *user_mtr1.h* file can be calculated using the `Motor Control Parameters Calculation.xlsx` Excel® spreadsheet. This file is included in the project folder: `\examples\universal_motorcontrol_lab\doc`. Copy parameters marked in ***bold*** to the *user_mtr1.h* file as shown in the following code.

```
//! \brief Defines the maximum voltage at the AD converter
#define USER_M1_ADC_FULL_SCALE_VOLTAGE_V         (57.52845691f)
```

```
//! \brief Defines the analog voltage filter pole location, Hz
#define USER_M1_VOLTAGE_FILTER_POLE_Hz          (680.4839141f)     // 47nF

//! \brief Defines the maximum current at the AD converter
#define USER_M1_ADC_FULL_SCALE_CURRENT_A         (47.14285714f)     // gain=10
```

4.6.1.2.2 Modifying Motor Control Parameters

The parameters provided in the *user_mtr1.h* file for a PMSM motor are listed as shown in the following code. The motor parameters can be identified from the motor data sheet.

```
#define USER_MOTOR1_TYPE                      MOTOR_TYPE_PM
#define USER_MOTOR1_NUM_POLE_PAIRS            (4)

#define USER_MOTOR1_Rs_Ohm                    (0.38157931f)
#define USER_MOTOR1_Ls_d_H                    (0.000188295482f)
#define USER_MOTOR1_Ls_q_H                    (0.000188295482f)
#define USER_MOTOR1_RATED_FLUX_VpHz           (0.0396642499f)
#define USER_MOTOR1_MAX_CURRENT_A             (6.0f)
```

4.6.1.2.3 Changing Pin Assignment

AM263_xxx.syscfg file configures the function of the GPIO pins and sets the direction and mode of the specified pin according to the hardware motor driver board/kit that is used. For modifying the code for a custom board, a TI motor driver EVM that does not currently have universal lab code support, or for use with a different TI MCU, these GPIO assignments need to be changed to correspond properly with the motor driver board.

4.6.1.2.4 Configuring the PWM Module

The SysConfig file configures the PWM channels. The base addresses of the PWM channels that are used for the motor controller PWM inputs are defined in the *hal.h* file, and the base addresses are assigned to the PWM handles in the *hal.c* file. The connection diagram for the PWM signals between the LP-AM263 and BOOSTXL-3PHGANINV is shown in Figure 4-47.

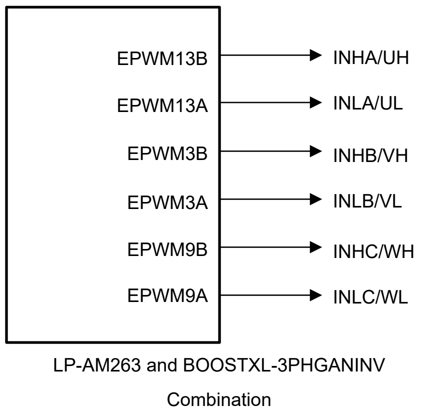


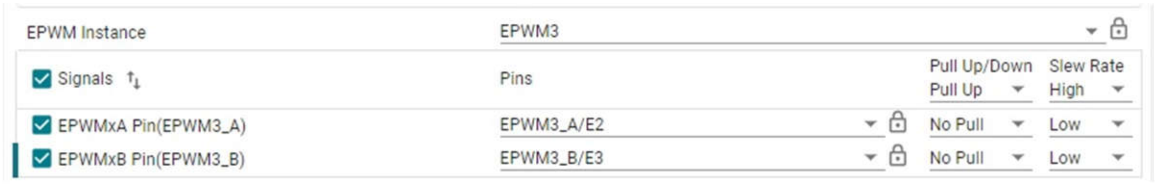
Figure 4-47. PWM Connection Diagram

The code to configure the PWM signals is shown in the following, taken from the *.syscfg*, *hal.h* and *hal.c* files.

1. The base addresses of the PWM modules are defined in the *hal.h* file as shown in the following.

```
#define MTR1_PWM_U_BASE          CONFIG_EPWM13_BASE_ADDR
#define MTR1_PWM_V_BASE          CONFIG_EPWM3_BASE_ADDR
#define MTR1_PWM_W_BASE          CONFIG_EPWM9_BASE_ADDR
```

2. The GPIOs are set up as PWM outputs in the *.syscfg* file.



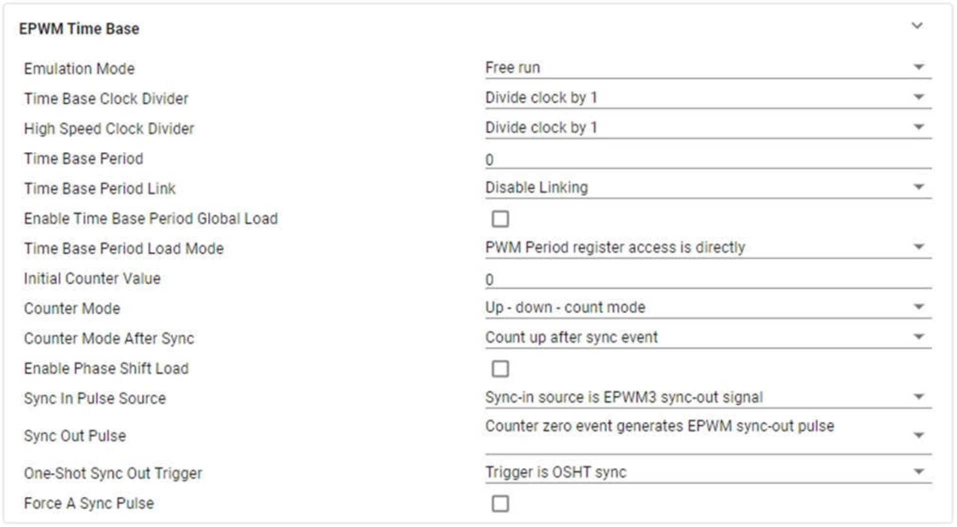
EPWM Instance	EPWM3		
Signals	Pins	Pull Up/Down: Pull Up	Slew Rate: High
✓ EPWMxA Pin(EPWM3_A)	EPWM3_A/E2	No Pull	Low
✓ EPWMxB Pin(EPWM3_B)	EPWM3_B/E3	No Pull	Low

Figure 4-48. GPIO Configuration for PWM Modules

3. The following code assigns the corresponding base addresses of the PWM modules to the PWM handle in the HAL_MTR1_init() function that is located in the *hal.c* file. The following code does not need to be changed when adapting the code to a new board or TI MCU, the following code block shows how the PWM handle is initialized in the code.

```
    // initialize PWM handles for Motor 1
    obj->pwmHandle[0] = MTR1_PWM_U_BASE;        //!< the PWM handle
    obj->pwmHandle[1] = MTR1_PWM_V_BASE;        //!< the PWM handle
    obj->pwmHandle[2] = MTR1_PWM_W_BASE;        //!< the PWM handle
```

4. Figure 4-49 shows the EPWM time base configuration. Sync out pulse for phase A is used as sync in pulse source for other PWMs.

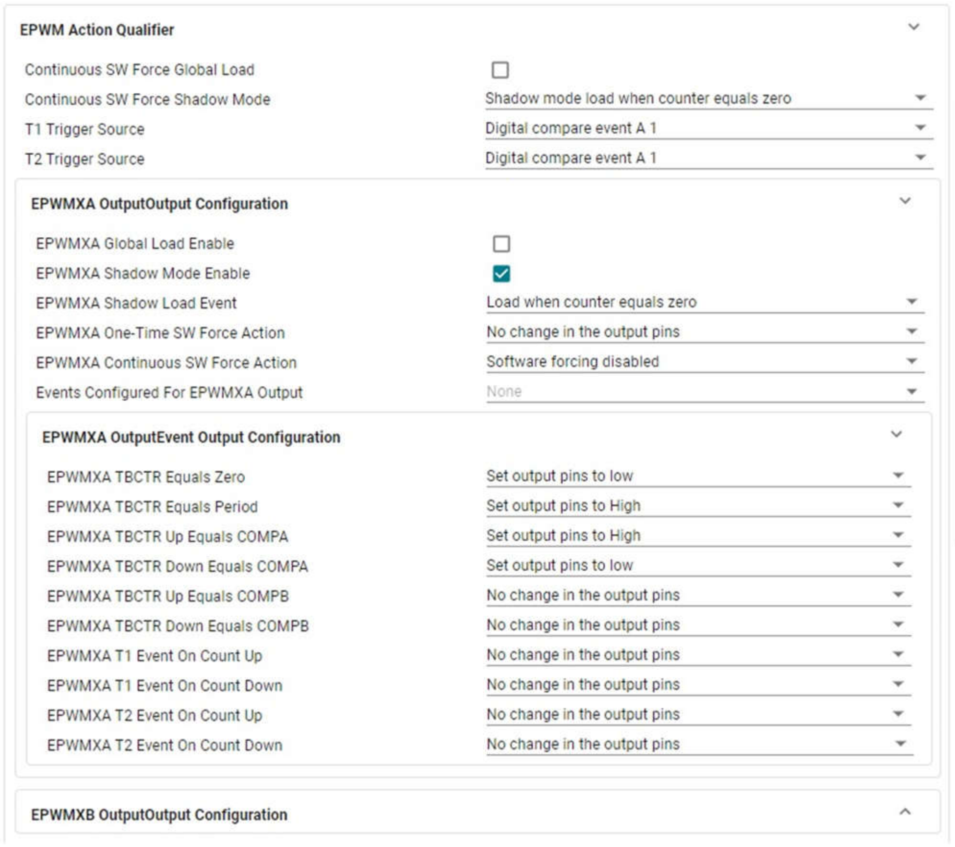


EPWM Time Base

Emulation Mode	Free run
Time Base Clock Divider	Divide clock by 1
High Speed Clock Divider	Divide clock by 1
Time Base Period	0
Time Base Period Link	Disable Linking
Enable Time Base Period Global Load	☐
Time Base Period Load Mode	PWM Period register access is directly
Initial Counter Value	0
Counter Mode	Up - down - count mode
Counter Mode After Sync	Count up after sync event
Enable Phase Shift Load	☐
Sync In Pulse Source	Sync-in source is EPWM3 sync-out signal
Sync Out Pulse	Counter zero event generates EPWM sync-out pulse
One-Shot Sync Out Trigger	Trigger is OSHT sync
Force A Sync Pulse	☐

Figure 4-49. EPWM Time Base Configuration

EPWM Action Qualifier Configurationshows the EPWM action qualifier output event configuration forLP-AM263 and BOOSTXL-3PHGANINV combination. PWM action qualifier outputs need to be set up based on the hardware board.



EPWM Action Qualifier

Continuous SW Force Global Load	☐
Continuous SW Force Shadow Mode	Shadow mode load when counter equals zero
T1 Trigger Source	Digital compare event A 1
T2 Trigger Source	Digital compare event A 1

EPWMXA OutputOutput Configuration

EPWMXA Global Load Enable	☐
EPWMXA Shadow Mode Enable	✓
EPWMXA Shadow Load Event	Load when counter equals zero
EPWMXA One-Time SW Force Action	No change in the output pins
EPWMXA Continuous SW Force Action	Software forcing disabled
Events Configured For EPWMXA Output	None

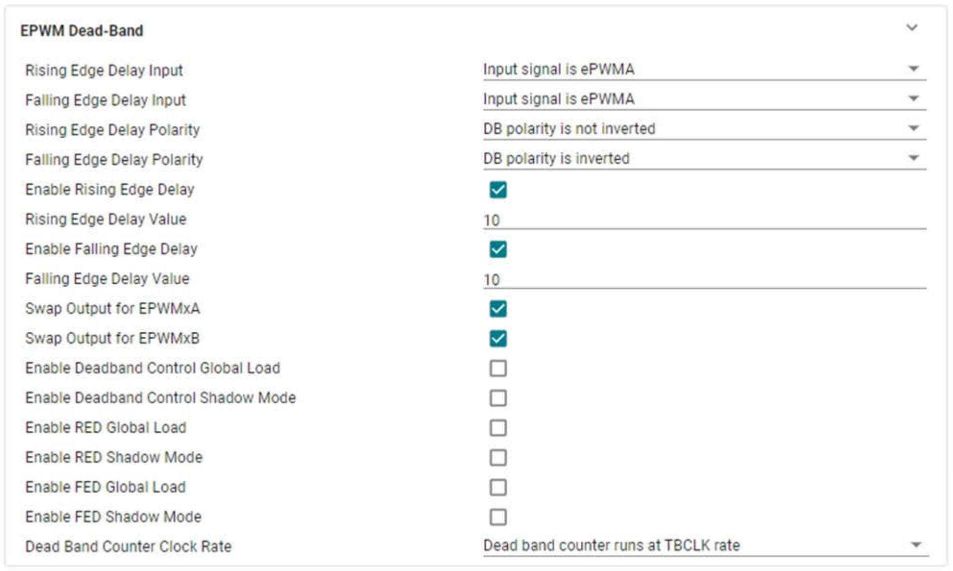
EPWMXA OutputEvent Output Configuration

EPWMXA TBCTR Equals Zero	Set output pins to low
EPWMXA TBCTR Equals Period	Set output pins to High
EPWMXA TBCTR Up Equals COMPA	Set output pins to High
EPWMXA TBCTR Down Equals COMPA	Set output pins to low
EPWMXA TBCTR Up Equals COMPB	No change in the output pins
EPWMXA TBCTR Down Equals COMPB	No change in the output pins
EPWMXA T1 Event On Count Up	No change in the output pins
EPWMXA T1 Event On Count Down	No change in the output pins
EPWMXA T2 Event On Count Up	No change in the output pins
EPWMXA T2 Event On Count Down	No change in the output pins

EPWMXB OutputOutput Configuration

Figure 4-50. EPWM Action Qualifier Configuration

Figure 4-51 shows the EPWM dead-band configuration for LP-AM263 . Swap output is checked for EPWMxA-B to match the high side and low side PWMs in LaunchPad™ and Booster Pack™.



EPWM Dead-Band

Rising Edge Delay Input	Input signal is ePWMA
Falling Edge Delay Input	Input signal is ePWMA
Rising Edge Delay Polarity	DB polarity is not inverted
Falling Edge Delay Polarity	DB polarity is inverted
Enable Rising Edge Delay	✓
Rising Edge Delay Value	10
Enable Falling Edge Delay	✓
Falling Edge Delay Value	10
Swap Output for EPWMxA	✓
Swap Output for EPWMxB	✓
Enable Deadband Control Global Load	☐
Enable Deadband Control Shadow Mode	☐
Enable RED Global Load	☐
Enable RED Shadow Mode	☐
Enable FED Global Load	☐
Enable FED Shadow Mode	☐
Dead Band Counter Clock Rate	Dead band counter runs at TBCLK rate

Figure 4-51. EPWM Dead-Band Configuration

4.6.1.2.5 Configuring the ADC Module

Similar to the previous PWM section, the ADC connections can also be changed for a custom board or a TI motor control kit that is not supported with the universal motor control project. The *.syscfg* file configures the ADC channels to correctly correspond with the motor driver board. As an example, the connection diagram for the LP-AM263 and BOOSTXL-3PHGANINV combination is shown in Figure 4-52. The ADC modules configuration is described in the following steps.

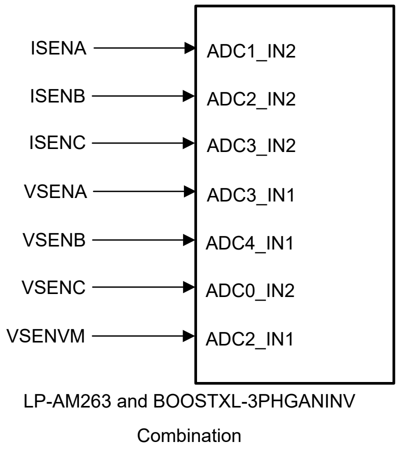
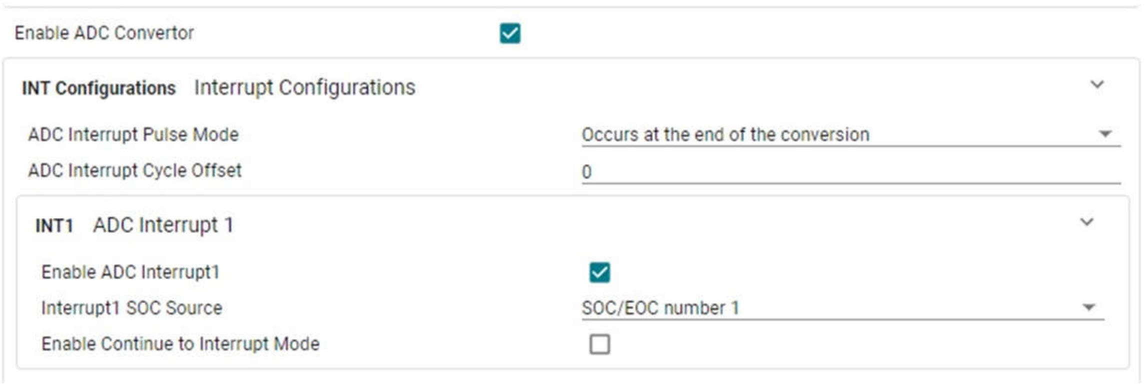


Figure 4-52. ADC Connection Diagram

1. The following code shows the defines of the base addresses, assigned channels, and SOCs of the ADC modules in the *hal.h* file. Note that for the SOC number, multiple ADCs can be associated with the same SOC number as long as they belong to different ADC modules (in the following case, module A and module C). Try to sample all the currents and all the voltages as close together as possible, so configure the SOC numbers with this in mind. The following code does not need to be changed when adapting the code to a new board or TI MCU, the following code is just to show how the ADC is initialized and the change can be done in the *.syscfg* file.

```
#define MTR1_IU_ADC_BASE        CONFIG_ADC1_BASE_ADDR //J7.67 ADC1_AIN2
#define MTR1_IV_ADC_BASE        CONFIG_ADC2_BASE_ADDR //J7.68 ADC2_AIN2
#define MTR1_IW_ADC_BASE        CONFIG_ADC3_BASE_ADDR //J7.69 ADC3_AIN2
#define MTR1_VU_ADC_BASE        CONFIG_ADC3_BASE_ADDR //J7.64 ADC3_AIN1
#define MTR1_VV_ADC_BASE        CONFIG_ADC4_BASE_ADDR //J7.65 ADC4_AIN1
#define MTR1_VW_ADC_BASE        CONFIG_ADC0_BASE_ADDR //J7.66 ADC0_AIN2
#define MTR1_VDC_ADC_BASE       CONFIG_ADC2_BASE_ADDR //J7.63 ADC2_AIN1

#define MTR1_IU_ADCRES_BASE     CONFIG_ADC1_RESULT_BASE_ADDR
#define MTR1_IV_ADCRES_BASE     CONFIG_ADC2_RESULT_BASE_ADDR
#define MTR1_IW_ADCRES_BASE     CONFIG_ADC3_RESULT_BASE_ADDR
#define MTR1_VU_ADCRES_BASE     CONFIG_ADC3_RESULT_BASE_ADDR
#define MTR1_VV_ADCRES_BASE     CONFIG_ADC4_RESULT_BASE_ADDR
#define MTR1_VW_ADCRES_BASE     CONFIG_ADC0_RESULT_BASE_ADDR
#define MTR1_VDC_ADCRES_BASE    CONFIG_ADC2_RESULT_BASE_ADDR

#define MTR1_IU_ADC_CH_NUM      ADC_CH_ADCIN2
#define MTR1_IV_ADC_CH_NUM      ADC_CH_ADCIN2
#define MTR1_IW_ADC_CH_NUM      ADC_CH_ADCIN2
#define MTR1_VU_ADC_CH_NUM      ADC_CH_ADCIN1
#define MTR1_VV_ADC_CH_NUM      ADC_CH_ADCIN1
#define MTR1_VW_ADC_CH_NUM      ADC_CH_ADCIN2
#define MTR1_VDC_ADC_CH_NUM     ADC_CH_ADCIN1

#define MTR1_IU_ADC_SOC_NUM     ADC_SOC_NUMBER0        // SOC0-PPB1
#define MTR1_IV_ADC_SOC_NUM     ADC_SOC_NUMBER0        // SOC0-PPB1
#define MTR1_IW_ADC_SOC_NUM     ADC_SOC_NUMBER0        // SOC0-PPB2
#define MTR1_VU_ADC_SOC_NUM     ADC_SOC_NUMBER1        // SOC1
#define MTR1_VV_ADC_SOC_NUM     ADC_SOC_NUMBER1        // SOC1
#define MTR1_VW_ADC_SOC_NUM     ADC_SOC_NUMBER1        // SOC1
#define MTR1_VDC_ADC_SOC_NUM    ADC_SOC_NUMBER1        // SOC1

#define MTR1_IU_ADC_PPB_NUM     ADC_PPB_NUMBER1        // SOC0-PPB1
#define MTR1_IV_ADC_PPB_NUM     ADC_PPB_NUMBER1        // SOC0-PPB1
#define MTR1_IW_ADC_PPB_NUM     ADC_PPB_NUMBER1        // SOC0-PPB2
```

2. Figure 4-53 shows the defines for the interrupt sources for the ISR in the *.syscfg* file.


Enable ADC Convertor ✓

INT Configurations Interrupt Configurations

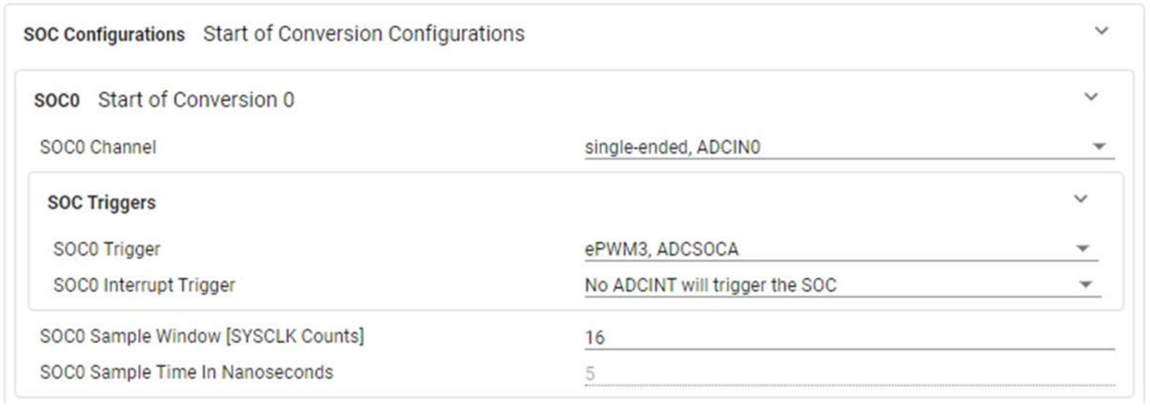
ADC Interrupt Pulse Mode: Occurs at the end of the conversion
ADC Interrupt Cycle Offset: 0

INT1 ADC Interrupt 1

Enable ADC Interrupt1: ✓
Interrupt1 SOC Source: SOC/EOC number 1
Enable Continue to Interrupt Mode: ☐

Figure 4-53. ADC Interrupt Configuration

3. Figure 4-54 defines the source of the ADC start of conversion trigger. This ePWM SOC trigger must correspond to the same ePWM SOC that was enabled in the code and the same ePWM that is associated with pwmHandle[0]. In this case, EPWM3 A is used as the SOC for the ADC.


SOC Configurations Start of Conversion Configurations

SOC0 Start of Conversion 0

SOC0 Channel: single-ended, ADCIN0

SOC Triggers

SOC0 Trigger: ePWM3, ADCSOCA
SOC0 Interrupt Trigger: No ADCINT will trigger the SOC

SOC0 Sample Window [SYSCLK Counts]: 16
SOC0 Sample Time In Nanoseconds: 5

Figure 4-54. ADC Start of Conversion Configuration

4.6.1.2.6 Configuring the CMPSS Module

The CMPSS module is used for overcurrent monitoring for the phase currents. A threshold is set using the CMPSS DAC, and if the output of the current sense amplifier exceeds that threshold then the CMPSS output trips.

If using a custom motor driver board, or migrating the code to a TI MCU or a TI motor driver EVM that is not supported with the current Universal Motor Control Project, then the connections between the ADC pins and the CMPSS modules need to be properly modified in the *.syscfg* file based on the motor driver and TI MCU connections. For more details on the internal connections of the CMPSS module, see the *ADC Signal Descriptions* tables in the AM263x Sitara™ Microcontrollers data sheet.

The *.syscfg* file configures the CMPSS modules according to the motor driver board that is used. For example, the diagram of the connections between the LP-AM263 and BOOSTXL-3PHGANINV are shown in Figure 4-55. Figure 4-56 shows the CMPSSA block diagram. CMPSSA has the additional support of INH and INL as a muxable input for the COMPL positive signal.

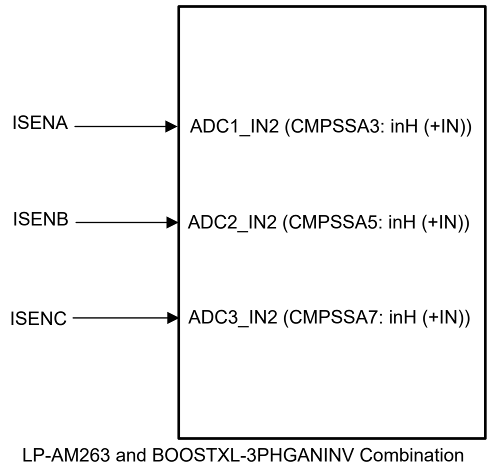

ISENA
ISENB
ISENC
ADC1_IN2 (CMPSSA3: inH (+IN))
ADC2_IN2 (CMPSSA5: inH (+IN))
ADC3_IN2 (CMPSSA7: inH (+IN))
LP-AM263 and BOOSTXL-3PHGANINV Combination

Figure 4-55. CMPSS Connection Diagram

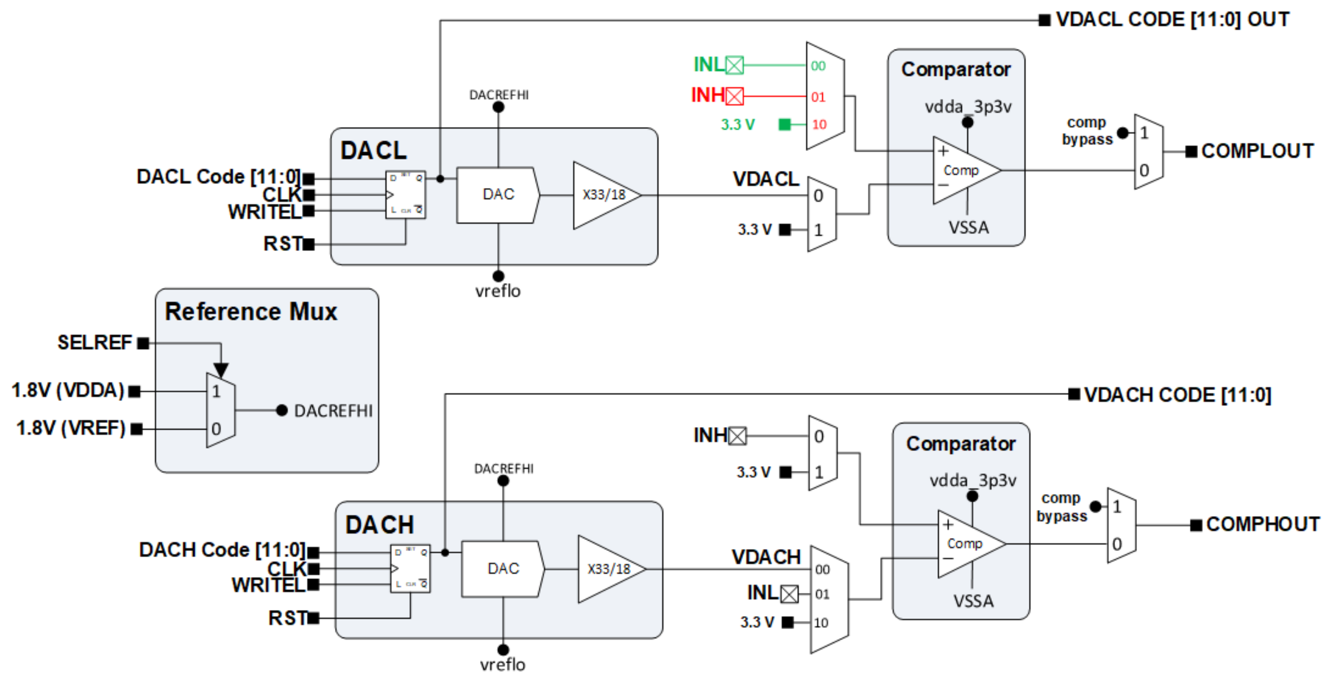
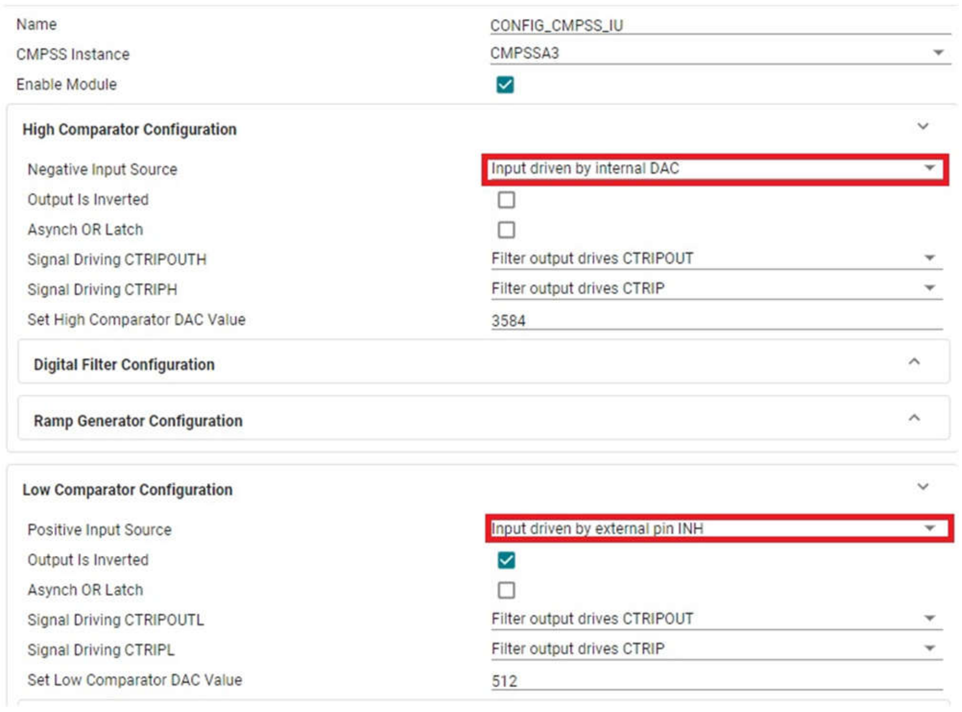


Figure 4-56. CMPSSA Block Diagram

Each CMPSS comparator has a high and low comparator, so the signals must be muxed appropriately to the desired input of the desired comparator. For more information on these connections, please refer to the Analog Pins and internal connections table in the data sheet of the microcontroller that is being used. Figure 4-57 shows the CMPSS comparator configuration for the LP-AM263 and BOOSTXL-3PHGANINV combination to do window comparison for phase currents. DAC values are updated in the code based on the defined maximum current. Note that for AM263x devices, the ADCx_AIN1 and ADCx_AIN3 are only connected to the (INL) which results limitation for both positive and negative overcurrent trips. For TMDSHVMTRINSPIN and TMDSCNCD263 combination, the ADC used for U-pahse current measurement are connected to ADC1_AIN3.

Also, please note that to select the right voltage reference in Launch Pad or EVM Control Card matching the DAC Reference Voltage. For example, in AM263x LaunchPad, use DAC VREF Switch (S1) to choose AM263x on-die LDO.



Name	CONFIG_CMPSS_IU
CMPSS Instance	CMPSSA3
Enable Module	✓

High Comparator Configuration

Negative Input Source	Input driven by internal DAC
Output Is Inverted	☐
Asynch OR Latch	☐
Signal Driving CTRIPOUTH	Filter output drives CTRIPOUT
Signal Driving CTRIPH	Filter output drives CTRIP
Set High Comparator DAC Value	3584

Digital Filter Configuration

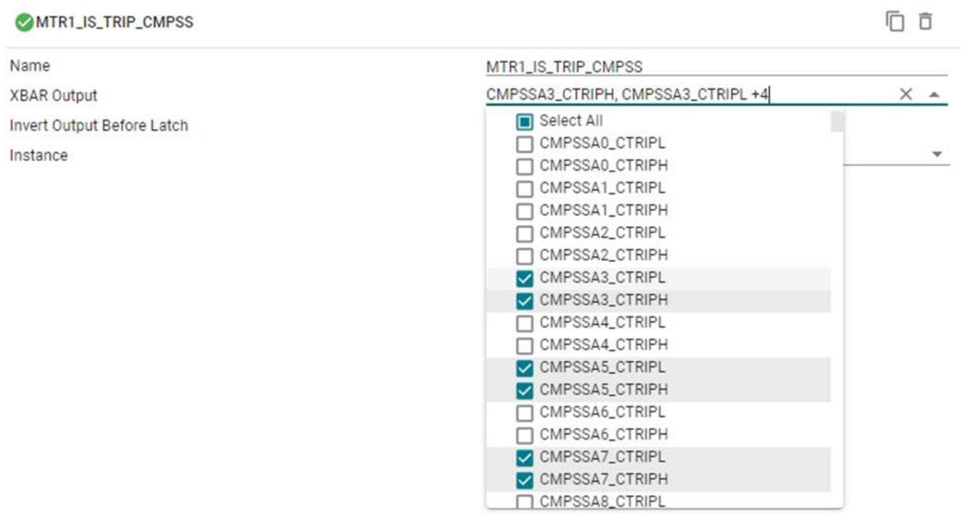
Ramp Generator Configuration

Low Comparator Configuration

Positive Input Source	Input driven by external pin INH
Output Is Inverted	✓
Asynch OR Latch	☐
Signal Driving CTRIPOUTL	Filter output drives CTRIPOUT
Signal Driving CTRIPL	Filter output drives CTRIP
Set Low Comparator DAC Value	512

Figure 4-57. CMPSS Comparator Configuration

Figure 4-58 shows the selecting corresponding CMPSS CTRIPL and CTRIPH for the EPWM XABR to generate trip in the event of overcurrent and undercurrent.



MTR1_IS_TRIP_CMPSS

Name	MTR1_IS_TRIP_CMPSS
XBAR Output	CMPSSA3_CTRIPH, CMPSSA3_CTRIPL +4
Invert Output Before Latch	
Instance	

- ☐ Select All
- ☐ CMPSSA0_CTRIPL
- ☐ CMPSSA0_CTRIPH
- ☐ CMPSSA1_CTRIPL
- ☐ CMPSSA1_CTRIPH
- ☐ CMPSSA2_CTRIPL
- ☐ CMPSSA2_CTRIPH
- ✓ CMPSSA3_CTRIPL
- ✓ CMPSSA3_CTRIPH
- ☐ CMPSSA4_CTRIPL
- ☐ CMPSSA4_CTRIPH
- ✓ CMPSSA5_CTRIPL
- ✓ CMPSSA5_CTRIPH
- ☐ CMPSSA6_CTRIPL
- ☐ CMPSSA6_CTRIPH
- ✓ CMPSSA7_CTRIPL
- ✓ CMPSSA7_CTRIPH
- ☐ CMPSSA8_CTRIPL

Figure 4-58. EPWM XBAR Configuration

5 General Texas Instruments High Voltage Evaluation (TI HV EVM) User Safety Guidelines

WARNING

Always follow TI's setup and application instructions, including use of all interface components within the recommended electrical rated voltage and power limits. Always use electrical safety precautions to help maintain your personal safety and those working around you. Contact TI's Product Information Center for further information.

WARNING

Failure to follow warnings and instructions can result in personal injury, property damage or death due to electrical shock and burn hazards.

The term TI HV EVM refers to an electronic device typically provided as an open framed, unenclosed printed circuit board assembly. The TI HV EVM is *intended strictly for use in development laboratory environments, solely for qualified professional users having training, expertise and knowledge of electrical safety risks in development and application of high voltage electrical circuits. Any other use and/or application are strictly prohibited by Texas Instruments.* If you are not qualified, you must immediately stop from further use of the HV EVM.

1. Work Area Safety
   a. Keep work area clean and orderly.
   b. Qualified observer(s) must be present anytime circuits are energized.
   c. Effective barriers and signage must be present in the area where the TI HV EVM and the interface electronics are energized, indicating operation of accessible high voltages can be present, for the purpose of protecting inadvertent access.
   d. All interface circuits, power supplies, evaluation modules, instruments, meters, scopes and other related apparatus used in a development environment exceeding 50Vrms/75VDC must be electrically located within a protected Emergency Power Off EPO protected power strip.
   e. Use stable and non conductive work surface.
   f. Use adequately insulated clamps and wires to attach measurement probes and instruments. No freehand testing whenever possible.
2. Electrical Safety
   As a precautionary measure, assume that the entire EVM has fully accessible and active high voltages.
   a. De-energize the TI HV EVM and all the inputs, outputs and electrical loads before performing any electrical or other diagnostic measurements. Re-validate that TI HV EVM power has been safely de-energized.
   b. With the EVM confirmed de-energized, proceed with required electrical circuit configurations, wiring, measurement equipment connection, and other application needs, while still assuming the EVM circuit and measuring instruments are electrically live.
   c. After EVM readiness is complete, energize the EVM as intended.

   WARNING

   While the EVM is energized, never touch the EVM or the electrical circuits, as these can be at high voltages capable of causing electrical shock hazard.
3. Personal Safety
   a. Wear personal protective equipment (for example, latex gloves or safety glasses with side shields) or protect EVM in an adequate lucent plastic box with interlocks to protect from accidental touch.

Limitation for safe use:

EVMs are not to be used as all or part of a production unit.

6 Design and Documentation Support

6.1 Design Files

6.1.1 Schematics

To download the BOOSTXL-3PHGANINV schematics, see the design files at BOOSTXL-3PHGANINV.

To download the TMDSHVMTRINSPIN schematics, see the hardware files located in <install_location>\solutions\tmdshvmtrinspin\hardware> folder of C2000WARE-MOTORCONTROL-SDK.

6.1.2 BOM

To download the BOOSTXL-3PHGANINV bill of materials (BOM), see the design files at BOOSTXL-3PHGANINV .

To download the TMDSHVMTRINSPIN bill of materials (BOM), see the hardware files are in <install_location>\solutions\tmdshvmtrinspin\hardware> folder of C2000WARE-MOTORCONTROL-SDK.

6.1.3 PCB Layout Recommendations

6.1.3.1 Layout Prints

To download the BOOSTXL-3PHGANINV layout prints, see the design files at BOOSTXL-3PHGANINV.

To download the TMDSHVMTRINSPIN layout prints, see the hardware files are in <install_location>\solutions\tmdshvmtrinspin\hardware> folder of C2000WARE-MOTORCONTROL-SDK.

6.2 Tools and Software

Tools

TMDSCNCD263	TMDSCNCD263 is an HSEC180 controlCARD based evaluation and development tool for the AM263x series Sitara™ high-performance microcontrollers. This board is designed for initial evaluation and prototyping as the board provides a standardized and easy-to-use platform to develop your next application.
Code Composer Studio™	The Code Composer Studio™ IDE is a complete integrated suite that enables developers to create and debug applications of all Texas Instruments Embedded Processors (Sitara, DSP, Automotive, Keystone), Microcontrollers (SimpleLink™, C2000 Digital Control, MSP430, TM4C, Hercules), as well as Digital Power (UCD) and Programmable Gain Amplifier (PGA) devices.
ARM-CGT-CLANG	The tiarmclang compiler tools provide software development tools including the compiler, assembler, and linker, among others, which can be used to develop applications with C/C++ source code for loading and running on Arm Cortex-M and Cortex-R series core processors.
SYSCONFIG	SysConfig is a configuration tool designed to simplify hardware and software configuration challenges to accelerate software development. SysConfig provides an intuitive graphical user interface for configuring pins, peripherals, radios, software stacks, RTOS, clock tree and other components. SysConfig automatically detects, expose and resolve conflicts to speed software development.

Software

MCU-PLUS-SDK-AM263X	The AM263x microcontroller (MCU) plus software development kit (SDK) is a unified software platform for embedded processors providing easy setup and fast out-of-the-box access to examples, benchmarks and demonstrations.
MOTOR-CONTROL-SDK-AM263X	The Motor Control SDK for AM263X contains examples, libraries and tools to develop RTOS and no-RTOS based applications enabling real-time communication for position sense from motors, and real-time control libraries for Arm R5F CPU and related peripherals.

6.3 Documentation Support

1. Texas Instruments: *Motor Control SDK Universal Project and Lab*, user's guide.
2. Texas Instruments: *AM263x Sitara™ Microcontrollers* data sheet.
3. Texas Instruments: *AM263x Sitara™ Microcontrollers Texas Instruments Families of Products*, technical reference manual.
4. Texas Instruments: *AM263x Control Card Hardware*, user's guide.

6.4 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.

6.5 Trademarks

LaunchPad™, BoosterPack™, controlCARD™, TI E2E™, Sitara™, Code Composer Studio™, and are trademarks of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited.
EtherCAT® and PROFINET® are registered trademarks of Beckhoff Automation GmbH.
MATLAB® is a registered trademark of The MathWorks, Inc.
All trademarks are the property of their respective owners.

7 About the Author

Masoud Farhadi is a system engineer at Automotive Application Specific MCUs, where he contributes to the designs for some of the semiconductor industry's most pressing needs. A graduate of the University of Texas at Dallas, Masoud holds a PhD degree in electrical engineering – power electronics. He is passionate about designing power electronics systems and leveraging latest patent trends to advance the EV technology.