Application Report
SPRABZ5–July 2015

Single Axis Motor Control and PFC

Ramesh Ramamoorthy, Shamim Choudhury

ABSTRACT

This application note describes how to control a power factor correction (PFC) and a permanent magnet synchronous motor (PMSM) using the TMS320F2805x microcontrollers. TMS320F2805x devices are part of the family of C2000 microcontrollers with programmable gain amplifiers (PGAs), which enable cost-effective design of intelligent controllers for power electronics applications, by reducing the system components and increasing efficiency. With these devices, it is possible to realize more precise digital control of PFC, and field-oriented control (FOC) of motors. The evaluation of the implementation of these devices is discussed in this document, involving 2-phase interleaved PFC and sensorless FOC of PMSM. The FOC algorithm maintains efficiency in a wide range of speeds, and considers torque changes with transient phases by processing a dynamic model of the motor.

This application note describes the following:

- A theoretical background on field-oriented motor control principle
- Incremental motor builds based on modular software blocks, for evaluation of PFC and sensorless PMSM
- Experimental results

Contents

List of Figures

List of Tables

1 Introduction

A brushless 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 to adjust frequency and amplitude of stator voltage, referenced 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 reduced motor size. Additionally, the elimination of brushes reduces noise and EMI generation, and suppresses the need for brush maintenance.

Power factor correction (PFC) of the motor drive system improves the line-side power factor, in addition to providing a stable DC bus voltage for the drive. This factor extends the line-side low voltage operating range of the drive.

This document describes how the TMS320F2805x device can control both PFC and a PMSM. This new family of DSPs with programmable gain amplifiers (PGAs) enables a cost-effective design of intelligent controllers for PFC and brushless motors, which can fulfill enhanced operations consisting of fewer system components, lower system cost, and increased performance.

2 Permanent Magnet Motors

The control method presented relies on the field-oriented control (FOC). This algorithm maintains efficiency in a wide range of speeds, and considers torque changes with transient phases by controlling the flux directly from the rotor coordinates. This application report presents the implementation of a control for a sinusoidal PMSM motor. The sinusoidal voltage waveform applied to this motor is created by using the space vector modulation technique. A minimum amount of torque ripple appears when driving this sinusoidal BEMF motor with sinusoidal currents.

There are two primary types of 3-phase PMSM. One uses rotor windings fed from the stator, and the other uses permanent magnets. A motor fitted with rotor windings requires brushes to obtain its current supply and generate rotor flux. The contacts are made of rings, and have commutator segments. The drawbacks of this type of structure are increased maintenance needs and lower reliability. For a brushless motor, replace the common rotor field windings and pole structure with permanent magnets. Brushless permanent magnet motors can be built with any even number of magnet poles. The use of magnets enables an efficient use of the radial space and replaces the rotor windings, therefore suppressing the rotor copper losses. Advanced magnet materials permit a considerable reduction in motor dimensions while maintaining a high power density.

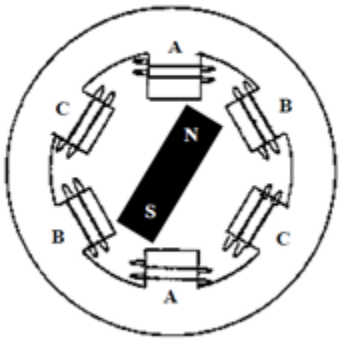


Figure 1. A Three-Phase Synchronous Motor With One Permanent Magnet Pair Pole Rotor

2.1 Synchronous Motor Operation

- 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, the user must control the stator currents.
- The structure of the rotor varies depending on the power range and rated speed of the machine. Permanent magnets are suitable for synchronous machines ranging up to a few kilowatts. For higher power ratings, the rotor usually consists of windings in which a DC current circulates. The mechanical structure of the rotor is designed for the desired number of poles, and the desired flux gradients.
- The interaction between the stator and rotor fluxes produces a torque. Because the stator is firmly mounted to the frame and the rotor is free to rotate, the rotor will rotate and produce a useful mechanical output.
- 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 a sensorless algorithm to draw the minimum amount of current under the same speed and torque conditions.
- The rotating stator field must rotate at the same frequency as the rotor permanent magnetic field; otherwise, the rotor experiences rapidly alternating positive and negative torque. Alternating torques results in less than optimal torque production, and excessive mechanical vibration, noise, and mechanical stresses on the machine parts. Also, if the rotor inertia prevents the rotor from responding to these oscillations, the rotor stops rotating at the synchronous frequency, and responds to the average torque as seen by the stationary rotor: zero. This means that the machine experiences a phenomenon known as pull-out. This is why the synchronous machine is not self-starting.
- The angle between the rotor field and the stator field must be equal to 90 degrees 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 have any direction and magnitude, by combining the contribution of different stator phases to produce the resulting stator flux.

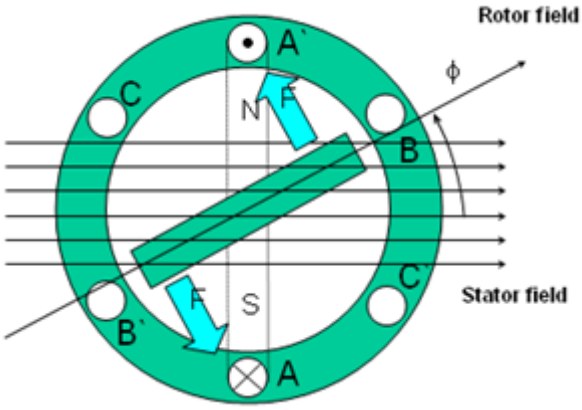


Figure 2. Interaction Between Rotating Stator Flux and Rotor Flux Produces a Torque, Causing Motor to Rotate

3 Field-Oriented Control

3.1 Introduction

For better dynamic performance, apply a more complex control scheme to control the PM motor. With the mathematical processing power offered by the microcontrollers, advanced control strategies can be implemented. These strategies use mathematical transformations to decouple the torque generation and the magnetization functions in PM motors. Such decoupled torque and magnetization control is commonly called rotor flux-oriented control, or FOC.

3.2 The Philosophy Behind the FOC

To understand the spirit of the FOC technique, start with an overview of the separately excited DC motor. In this type of motor, the excitation for the stator and rotor is independently controlled. Electrical study of the DC motor shows that the produced torque and the flux can be independently tuned. The strength of the field excitation (such as 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 means that the torque production of the machine is nearly optimal all the time. The windings are managed to keep the flux produced by the rotor windings orthogonal to the stator field.

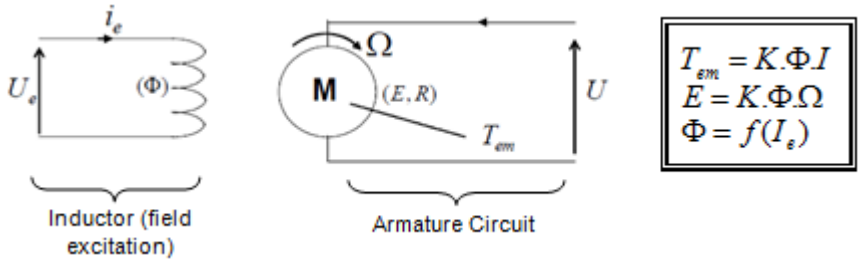


Figure 3. Separated Excitation DC Motor Model, Flux, and Torque are Independently Controlled and Current Through Rotor Windings Determines How Much Torque is Produced

AC machines do not have the same key features as the DC motor. Both cases have only one source that can be controlled: the stator currents. On the synchronous machine, the rotor excitation is given by the permanent magnets mounted onto the shaft. On the synchronous motor, the only source of power and magnetic field is the stator phase voltage. Unlike the DC motor, in AC machines flux and torque depend on each other.

The goal of the FOC (also called vector control) on synchronous and asynchronous machines is to separately control the torque-producing and magnetizing flux components. The control technique goal is to imitate the operation of the DC motor. FOC allows decoupling of the torque and the magnetizing flux components of the stator current. With decoupled control of the magnetization, the torque-producing component of the stator flux is now independent torque control. To decouple the torque and flux, the user must engage several mathematical transforms, as this is where the microcontrollers add the most value. The processing capability provided by the microcontrollers enables these mathematical transformations to be carried out quickly. This implies that the entire algorithm controlling the motor can be executed rapidly, 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 indicates that their effect is accounted for, and the overall quality of control is better.

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

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

Equation 1 shows that the torque is maximum if the stator and rotor magnetic fields are orthogonal, and if the load is maintained at 90 degrees. If this condition is constant and the flux is oriented correctly, the torque ripple is reduced and a better dynamic response ensured. The constraint is knowing the rotor position, which can be achieved with a position sensor such as an incremental encoder. For a low-cost application where the rotor is not accessible, different rotor position observer strategies are applied to remove the position sensor.

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

3.3 Technical Background

The FOC consists of controlling the stator currents represented by a vector. This control is based on projections which transform a 3-phase time and speed-dependent system into a 2-coordinate (d and q coordinates) time-invariant system. These projections lead to a structure similar to that of a DC machine control. Field-oriented controlled machines need two constants as input references: the torque component (aligned with the q coordinate) and the flux component (aligned with d coordinate). As FOC is based on projections, the control structure handles instantaneous electrical quantities. The control is thus 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:

$$m \alpha \Psi_R i_{Sq} \quad (2)$$

By maintaining the amplitude of the rotor flux (Ψ_R) at a fixed value, there is a linear relationship between torque and torque component (i_{Sq}). The user can then control the torque by controlling the torque component of stator current vector.

3.4 Space Vector Definition and Projection

The 3-phase voltages, currents, and fluxes of AC motors can be analyzed in terms of complex space vectors. With regard to the currents, the space vector can be defined as follows. Assuming that i_a, i_b, and i_c are the instantaneous currents in the stator phases, then the complex stator current vector $\bar{i}_s$ is defined by:

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

where

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

Figure 4 shows the stator current complex space vector.

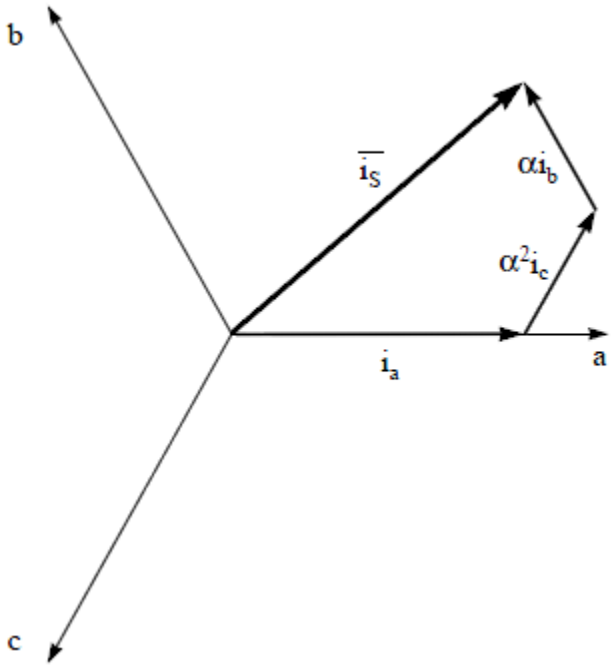


Figure 4. Stator Current Space Vector and its Component in (a,b,c)

where (a,b,c) are the 3-phase system axes. This current space vector depicts the 3-phase sinusoidal system, which still must be transformed into a 2-time invariant coordinate system. This transformation can be split into two steps:

- (a,b,c) → (α, β) (the Clarke transformation) which outputs a 2-coordinate time-variant system
- (α, β) → (d,q) (the Park transformation) which outputs a 2-coordinate time-invariant system

3.4.1 The (a,b,c) → (α, β) Projection (Clarke Transformation)

The space vector can be reported in another reference frame with only two orthogonal axes called (α, β). Assuming that axis a and axis α are in the same direction, observe the following vector diagram:

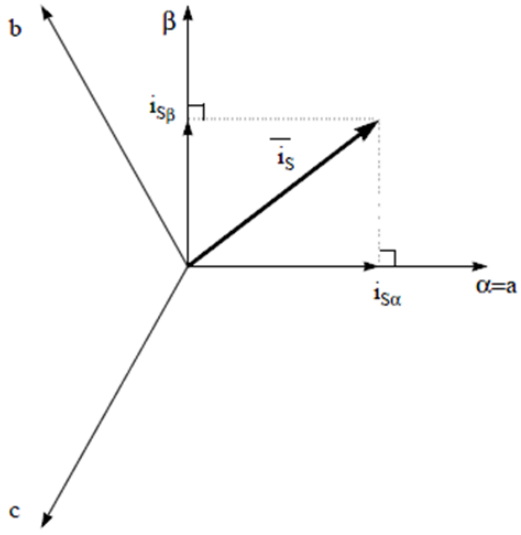


Figure 5. Stator Current Space Vector and Its Components in the Stationary Reference Frame

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

$$\begin{cases} i_{s\alpha} = i_a \\ i_{s\beta} = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b \end{cases} \tag{4}$$

The 2-phase (α, β) currents still depend on time and speed.

3.4.2 The (α, β) → (d,q) Projection (Park Transformation)

This is the most important transformation in the FOC, as it modifies a 2-phase orthogonal system (α, β) in the d,q rotating reference frame. If the d axis is aligned with the rotor flux, Figure 6 shows, for the current vector, the relationship from the 2-reference frame:

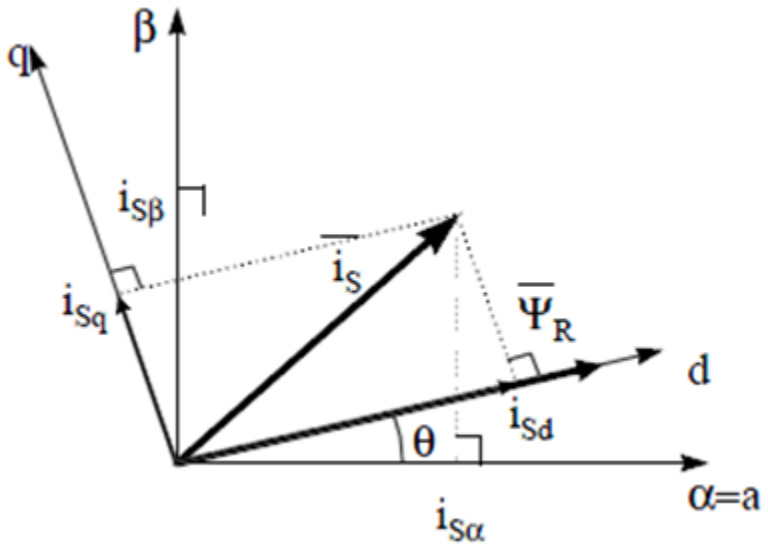


Figure 6. Stator Current Space Vector and Its Components in (α, β) and in the d,q Rotating Reference Frame

The flux and torque components of the current vector are determined by Equation 5:

$$\begin{cases} i_{sd} = i_{s\alpha} \cos\theta + i_{s\beta} \sin\theta \\ i_{sq} = -i_{s\alpha} \sin\theta + i_{s\beta} \cos\theta \end{cases}$$

where

- θ is the rotor flux position

(5)

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

3.5 The Basic Scheme for the FOC

Figure 7 summarizes the basic scheme of torque control with FOC.

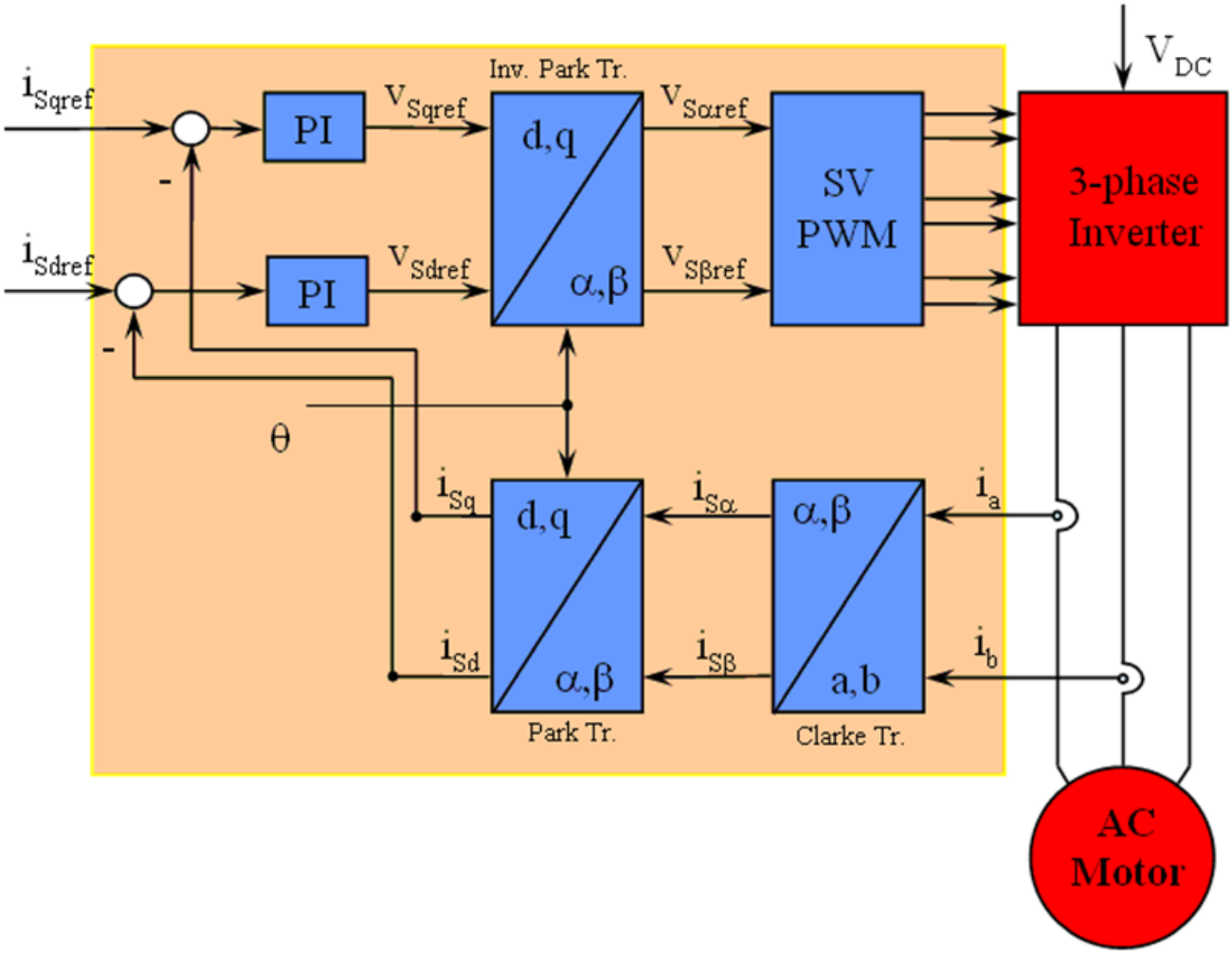


Figure 7. Basic Scheme of FOC for AC Motor

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

3.6 Rotor Flux Position

Knowledge of the rotor flux position is the core of the FOC. If there is an error in this variable, the rotor flux is not aligned with the d-axis, and i_{sd} and i_{sq} are incorrect flux and torque components of the stator current. The following diagram 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.

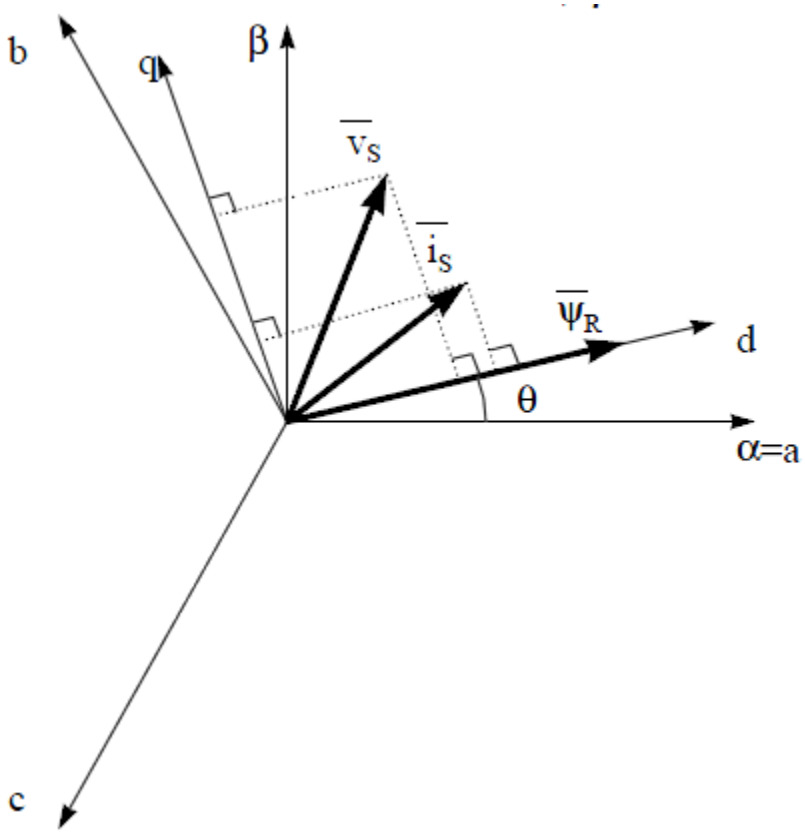


Figure 8. Current, Voltage, and Rotor Flux Space Vectors in the d,q Rotating Reference Frame and their Relationship with a,b,c and (α, β) Stationary Reference Frame

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

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

Theoretically, the FOC for the PMSM drive allows the motor torque to be controlled independently with the flux-like DC motor operation. In other words, the torque and flux are decoupled from each other. The rotor position is required for variable transformation from a stationary reference frame to a synchronously rotating reference frame. As a result of this transformation (the Park transformation), the q-axis current controls torque while the d-axis current is forced to zero. Thus, the key module of this system is the estimation of rotor position using a sliding-mode observer. Figure 9 shows the overall block diagram of this project.

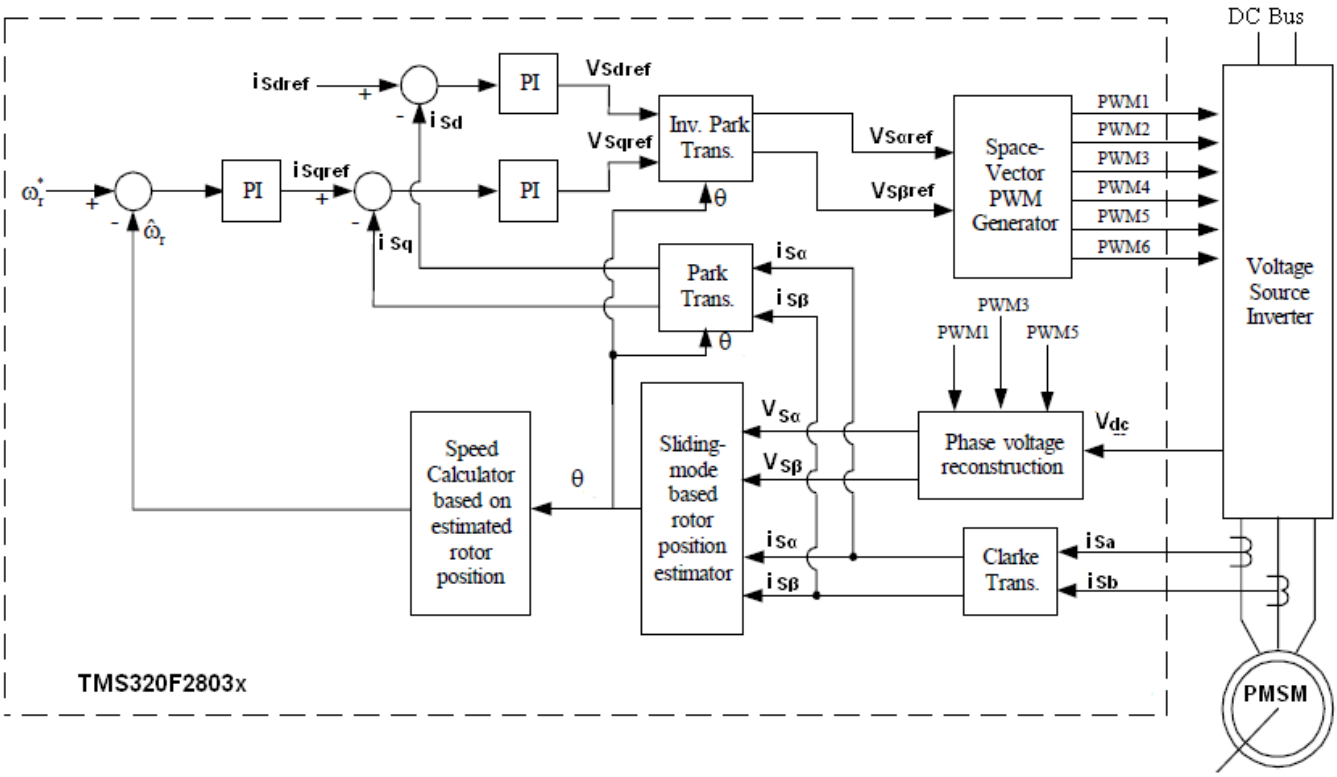


Figure 9. Overall Block Diagram of Sensorless FOC of PMSM

4 Benefits of 32-Bit C2000 Controllers

The C2000 family of devices possesses the desired computation power to execute complex control algorithms, along with the correct mix of peripherals to interface with the various components of the DMC hardware such as the ADC, ePWM, QEP, eCAP, and so forth. These peripherals have all the necessary hooks for implementing systems which meet safety requirements, such as the trip zones for PWMs and comparators. The F2805x device has a PGA to help amplify shunt current signals, thus reducing kit BOM and saving board space and cost.

Additionally, the C2000 ecosystem of software (libraries and application software) and hardware (application kits) can reduce the time and effort required to develop the solution. The digital power and digitial motor control library provides configurable blocks that can be reused to implement new control strategies. The IQMath library enables easy migration from floating point algorithms to fixed point, thus accelerating the development cycle.

Thus, with the C2000 family of devices, complex control algorithms (sensored and sensorless) for digital power and motor control can be implemented quickly and easily. The use of C2000 devices and advanced control schemes provides the following system improvements:

- Favors system cost reduction by an efficient control in all speed ranges, implying correct dimensioning of power device circuits
- Uses advanced control algorithms to reduce torque ripple, thus resulting in lower vibration and longer life time of the motor
- Uses advanced control algorithms to reduce harmonics generated by the inverter, thus reducing filter cost
- Uses sensorless algorithms to eliminate the need for speed or position sensors
- The real-time generation of smooth, near-optimal reference profiles and move trajectories results in increased performance
- Generates high resolution PWMs by using the ePWM peripheral to control the power-switching inverters

- Provides a single-chip control system

For advanced controls, C2000 controllers can also perform the following:

- Enables control of multivariable and complex systems using modern intelligent methods, such as neural networks and fuzzy logic
- Performs adaptive control, as C2000 controllers have the speed capabilities to concurrently monitor the system and control it. A dynamic control algorithm adapts itself in real time to variations in system behavior.
- Performs parameter identification for sensorless control algorithms, self-commissioning, and online parameter estimation updates
- Performs advanced torque ripple and acoustic noise reduction
- Provides diagnostic monitoring with spectrum analysis. By observing the frequency spectrum of mechanical vibrations, failure modes can be predicted in early stages.
- Produces sharp-cut-off notch filters that eliminate narrow band mechanical resonance. Notch filters remove energy that would otherwise excite resonant modes and possibly make the system unstable.

5 TI Literature and Digital Motor Control (DMC) Library

The digital motor control (DMC) library is composed of functions represented as blocks. These blocks are categorized as transforms and estimators (Clarke, Park, sliding mode observer, phase voltage calculation, and resolver, flux, and speed calculators and estimators), control (signal generation, PID, BEMF commutation, space vector generation), and peripheral drivers (PWM abstraction for multiple topologies and techniques, ADC drivers, and motor sensor interfaces). Each block is a modular software macro and is separately documented with source code, use, and technical theory. Check the following folders for the source codes and explanations of macro blocks:

- C:\TI\controlSUITE\libs\app_libs\motor_control\math_blocks\v4.2
- C:\TI\controlSUITE\libs\app_libs\motor_control\drivers\f2805x_v2.0

These modules let users quickly build or customize their own systems. The library supports the three motor types (ACI, BLDC, and PMSM), and comprises both peripheral-dependent (software drivers) and target-dependent modules.

TI uses the DMC library components to provide system examples. At initialization, all DMC library variables are defined and interconnected. At runtime, the macro functions are called in order. Each system is built using an incremental build approach, which allows building some sections of the code, so that the developer can verify each section of their application one step at a time. This is critical in real-time control applications, where many different variables can affect the system and many different motor parameters must be tuned.

TI DMC modules are written in the form of macros for optimization purposes (refer to application note SPRAAK2 for more details). The macros are defined in the header files. The user can open the respective header file and change the macro definition, if needed. In the macro definitions, there should be a backslash (\) at the end of each line, as shown in Figure 10, which means that the code continues in the next line. Any character after the backslash, including invisible ones such as a space cause a compilation error. Ensure that the backslash is the last character in the line. In terms of code development, the macros are almost identical to a C function, and the user can easily convert the macro definition to a C function.

```
#define PARK_MACRO(v)                                                    \
                                                                         \
    v.Ds = _IQmpy(v.Alpha,v.Cosine) + _IQmpy(v.Beta,v.Sine);             \
    v.Qs = _IQmpy(v.Beta,v.Cosine)  - _IQmpy(v.Alpha,v.Sine);
```

Figure 10. A Typical DMC Macro Definition

5.1 Note on PWM Frequencies

Ensure that the PWM frequency of PFC is an appropriate multiple of the PWM frequency of the motor, so that current sampling instances do not overlap and create GND disturbances, and inject noise in the current measurement of the other block.

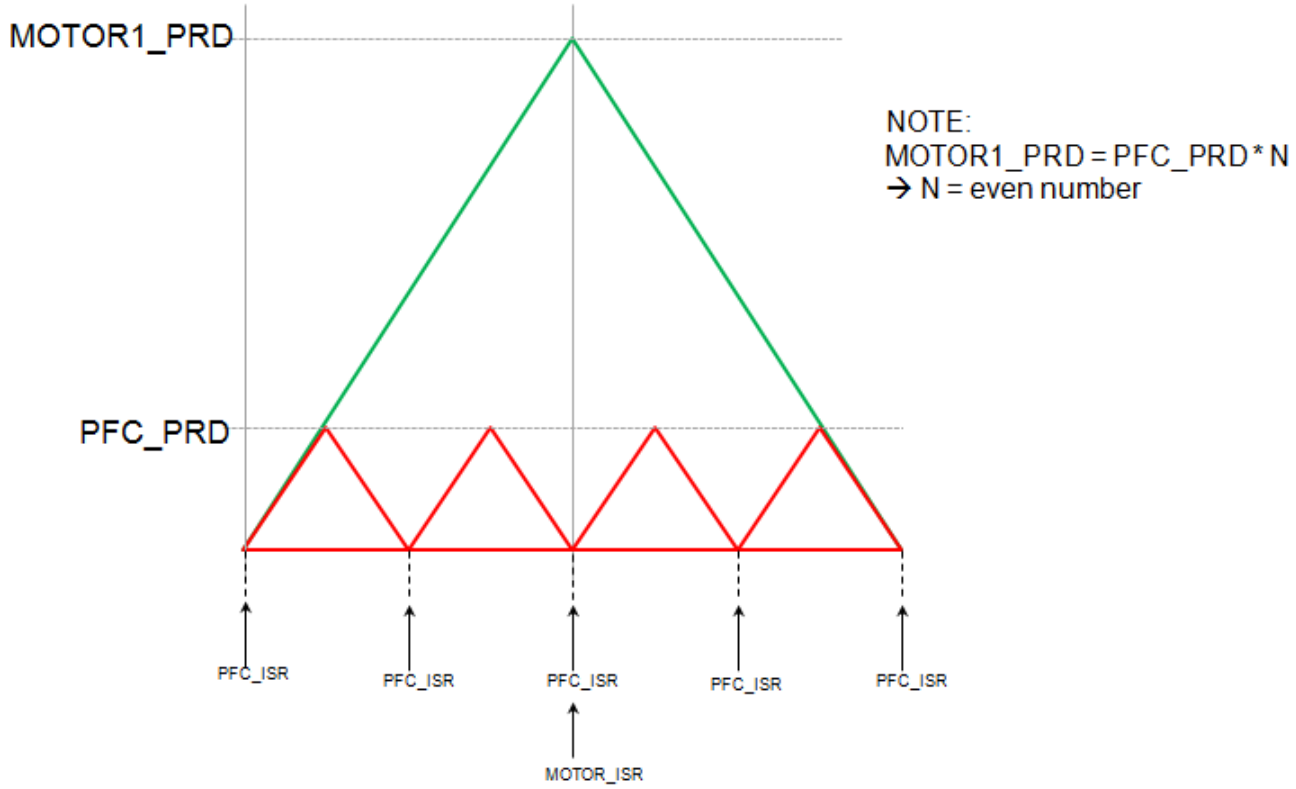


Figure 11. PWM Carrier Alignment for Inverter and PFC

6 System Overview

This document describes the “C” real-time control framework used to demonstrate the sensorless FOC of HVPM motors. The C framework is designed to run on TMS320C280x-based controllers on Code Composer Studio. The framework uses the following modules:

Macro Names	Explanation[(1)]
CLARKE	Clarke transformation
PARK / IPARK	Park and Inverse Park transformation
PI	PI regulators
RC	Ramp controller (slew rate limiter)
RG	Ramp and sawtooth generator
QEP / CAP	QEP and CAP drives (optional for speed loop tuning with a speed sensor)
SE	Speed estimation (based on sensorless position estimation)
SPEED_FR	Speed measurement (based on sensor signal frequency)
SMO	Sliding mode observer for sensorless applications
SVGEN	Space vector PWM with quadrature control (includes Clarke transformation)
PHASEVOLT	Phase voltage calculator
PWM / PWMDAC	PWM and PWMDAC drives

(1) Refer to the .pdf documents in the motor control folder explaining the details and theoretical background of each macro.

Figure 12 shows the overall system implementing a 2-phase interleaved PFC and speed control of a 3-phase permanent magnet motor. In Figure 12, the sensorless FOC of the PMSM is experimented with for speed control. The PM motor is driven by a conventional voltage-source inverter. The TMS320x2805x CPU generates the pulse-width modulation (PWM) signals for the motor and the PFC module. The motor is driven by an integrated power module, using the space vector PWM technique. By connecting shunt resistors to the bottom switch of the inverter and PFC current return path, voltages representing the motor phase currents (I_a, I_b, and I_c) and PFC current (I_{pfc}) are generated and Kelvin-connected to the PGAs of the TMS320F2805x CPU, where they are amplified and fed to analog-to-digital converters (ADCs). In addition, the DC bus voltage in the inverter is measured and sent to the TMS320x2805x using an ADC. This DC bus voltage is required to calculate the 3-phase voltages when the switching functions are known.

This project has the following properties:

Table 1. System Features

Feature	Description
Development /Emulation	Code Composer Studio V6.0 (or above) with real-time debugging
Target controller	TMS320F2805x
PFC PWM frequency	100 kHz
Motor PWM frequency	10-kHz PWM (default), 60-kHz PWMDAC
PWM mode	Symmetrical with a programmable dead band
Interrupts	ADC, end of conversion – Implements 10-kHz ISR execution rate
Peripherals used	PWM 1, 2, and 3 for motor control PWM 5 for PFC PWM 6A, 6B, 4A, and 4B for DAC outputs QEP1 A, B, I or CAP1 ADC A7 for DC bus voltage sensing, A1 and B1 for phase current sensing PGA for amplifying motor and PFC shunt currents

Figure 12 shows the overall system implementing a 3-phase HVPM motor control. The HVPM motor is driven by the conventional voltage-source inverter. The TMS320F2805x generates the six PWM signals using a space vector PWM technique, for six power-switching devices in the inverter. Two input currents of the HVPM motor (i_a and i_b) are measured from the inverter and sent to the PGAs of the TMS320F2805x, then on to two ADCs. In addition, the DC bus voltage in the inverter is measured and sent to the TMS320F2805x using an ADC. This DC bus voltage is required to calculate 3-phase voltages of the HVPM motor when the switching functions are known.

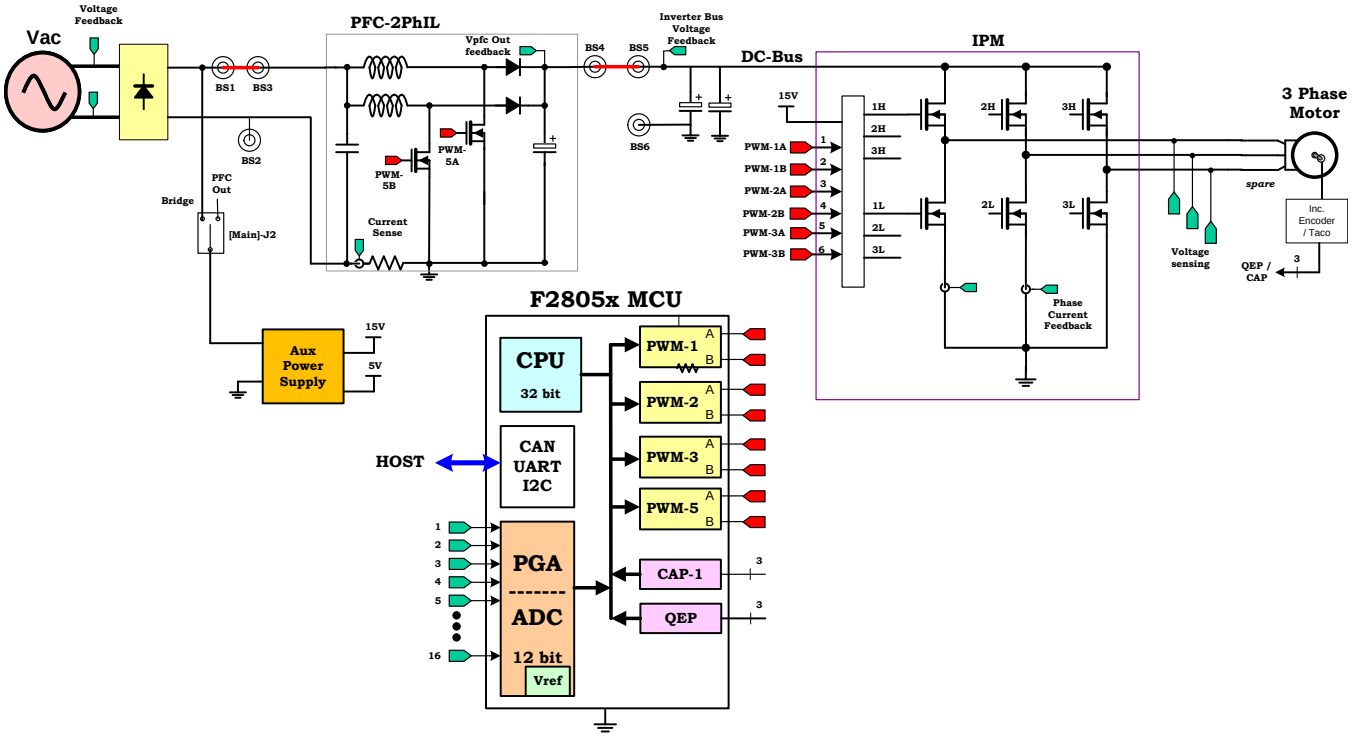


Figure 12. Single-Axis Motor + PFC Board Diagram With F2805x

7 Hardware Configuration

For an overview of the kit hardware and steps on how to setup this kit, refer to the 1AxisMtrPfc_5x kit's hardware guide and user's guide found at:

C:\TI\controlSUITE\development_kits\TIDM_1AXISMTR-PFC-5x_v1.0\~Docs

Some of the hardware setup instructions are described in Section 7.1 for quick reference.

7.1 Hardware Setup Instructions

1. Program the FTDI chip for xds100 emulation on board. Follow the instructions given in this TI E2E forum post. If the link is broken, search in https://e2e.ti.com.
2. Install the jumper [Main]-J5 for JTAG reset line.
3. Connect a USB cable to connector [M3]-JP1. This enables isolated JTAG emulation to the C2000 device. [M3]-LD1 should turn on.
4. Ensure that [M6]-SW1 is in the Off position. Connect a 15-V DC power supply to [M6]-JP1.
5. Turn on [M6]-SW1. [M6]-LD1 should turn on.
6. Note that the motor should be connected to the [Main]-TB1 terminals after finishing with the first incremental build step.
7. Only apply the DC bus power when instructed. The various options for DC bus power are:
   - To use the DC power supply: Set the power supply output to zero and connect [Main]-BS5 and BS2 to the DC power supply and ground, respectively. Also, connect [Main]-BS5 and [Main]-BS4, as the CPU senses bus voltage information from the PFC output stage where [Main]-BS4 is located. See Figure 13.
   - To use AC mains power without PFC: Connect [Main]-BS1 and BS5 to each other using a banana plug cord. Also, connect [Main]-BS5 and [Main]-BS4. Now connect one end of the AC power cord to [Main]-P1. The other end must be connected to the output of an isolated variable AC supply. If a variable AC supply is not available, then connect it to a variac that is connected to AC mains through an isolation transformer, as shown in Figure 16. Ensure that the variac output is set to

zero before connecting. See Figure 14.

- To use AC mains power with PFC: Connect [Main]-BS1 and BS3 to each other using a banana plug cord. Also, connect [Main]-BS5 and [Main]-BS4 as shown in Figure 15. Now connect one end of the AC power cord to [Main]-P1. The other end must be connected to the output of an isolated variable AC supply. If a variable AC supply is not available, connect it to a variac that is connected to AC mains through an isolation transformer, as shown in Figure 15. Ensure that the variac output is set to zero before connecting. See Figure 16.

CAUTION

Isolation is required if measurement equipment is connected to the board.

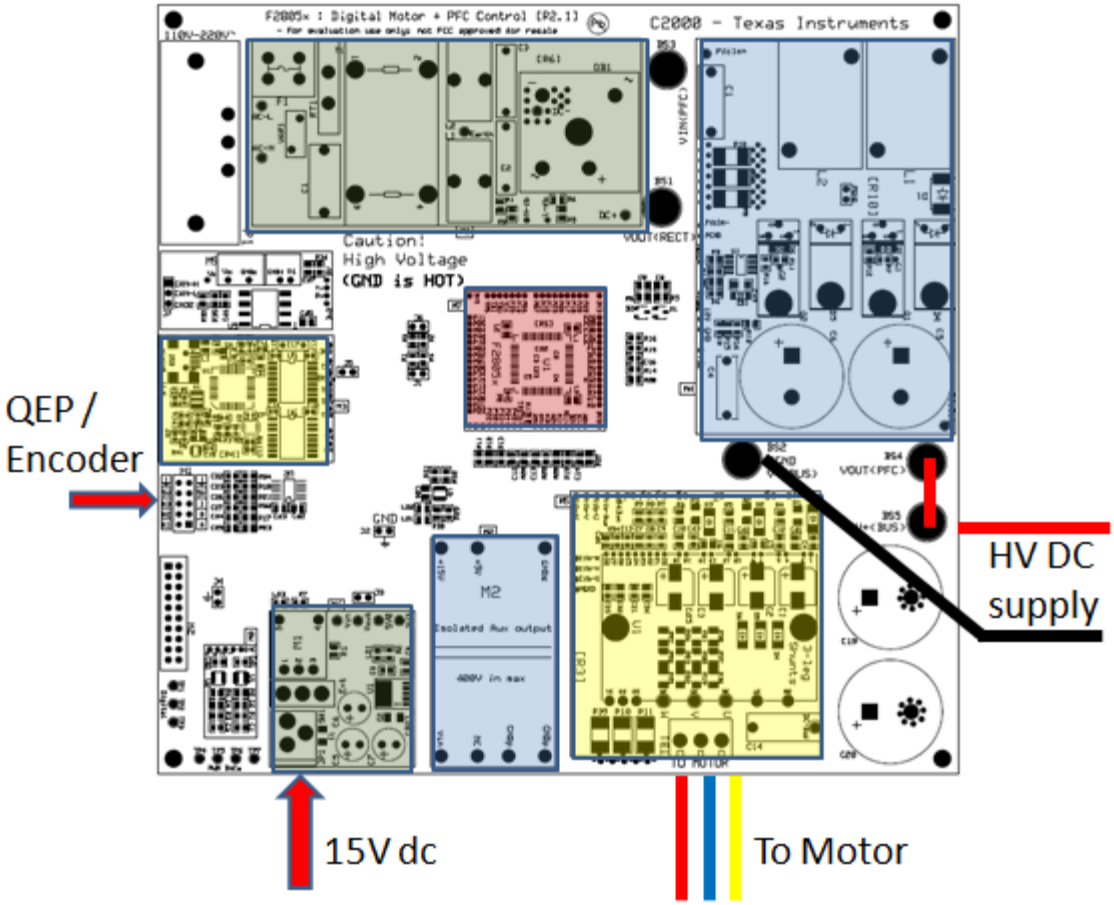


Figure 13. Using External Power Supply to Provide the DC Bus Voltage for the Inverter

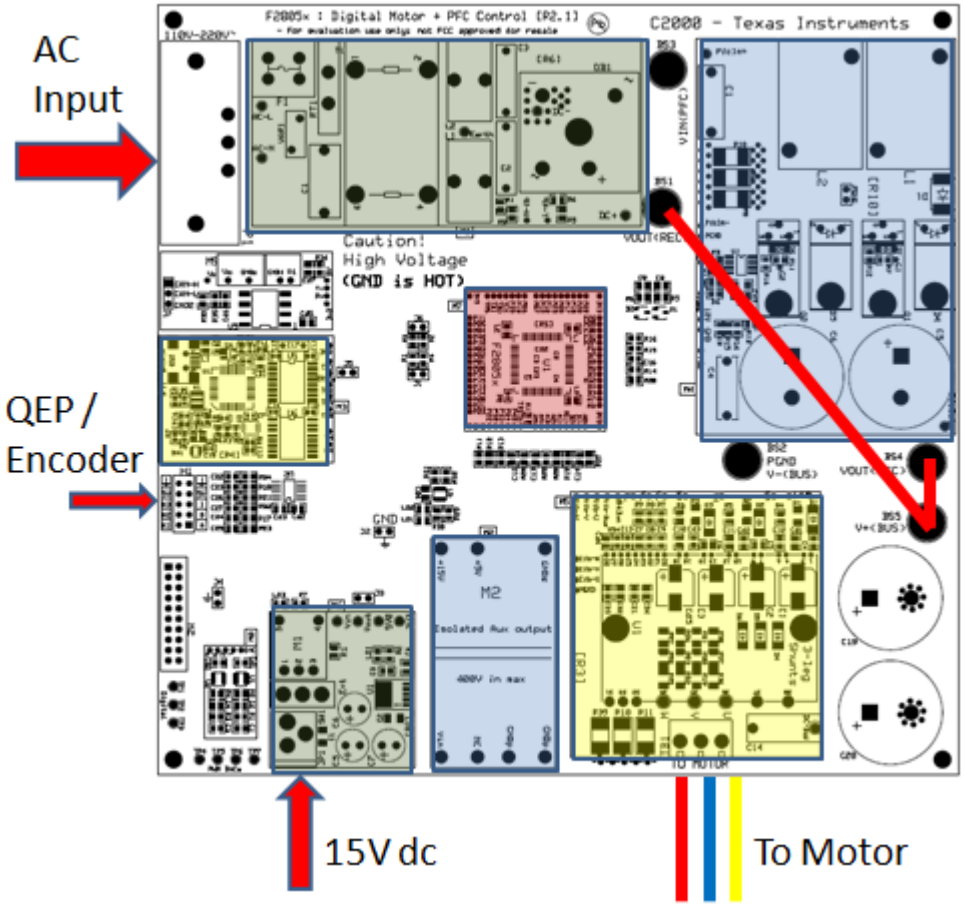


Figure 14. Using AC Input to Generate DC Bus Voltage for the Inverter, Bypassing PFC

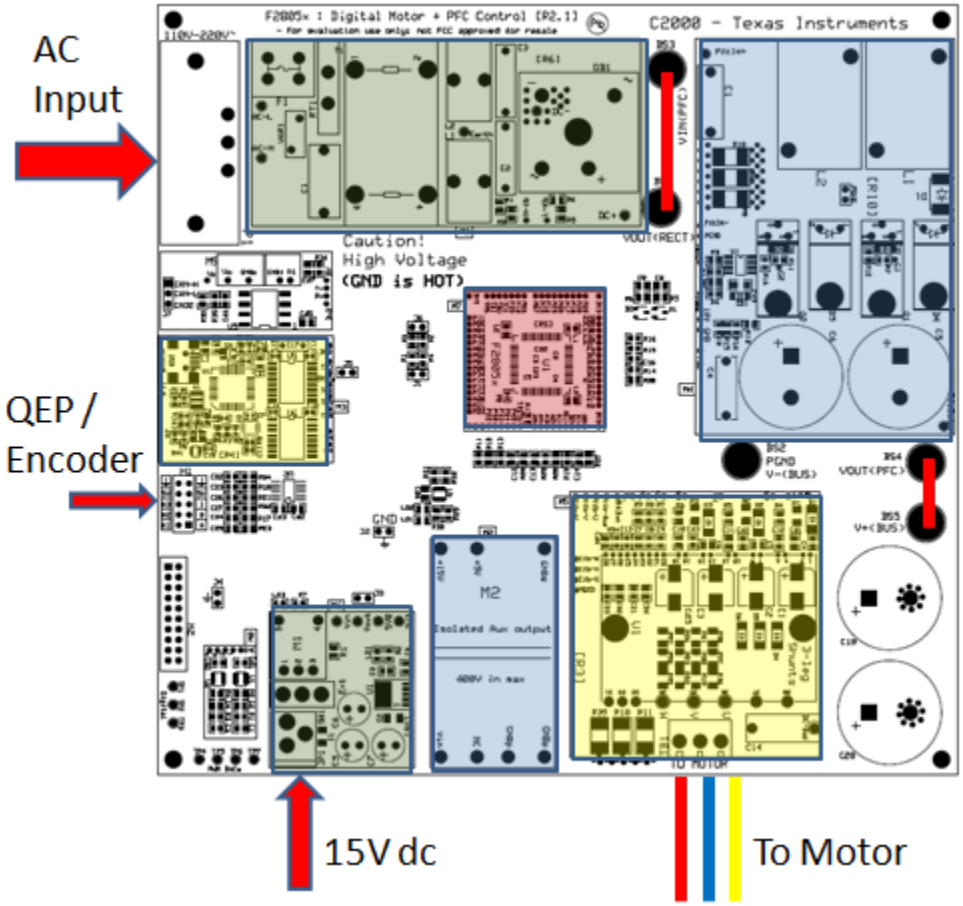


Figure 15. Using AC Input and PFC to Generate DC Bus Voltage for the Inverter

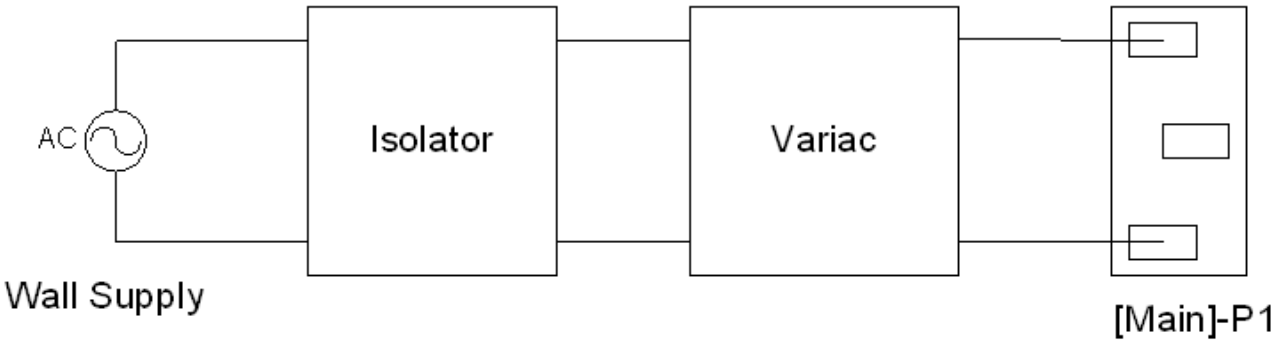


Figure 16. Wiring Diagram of the Kit With an Isolator and a Variac

CAUTION

DC bus capacitors remain charged for a long time after the mains supply is disconnected. Use caution.

8 Control Software Flow

Figure 17 shows the software flow for PFC and sensorless PMSM control. After initializing the peripherals for the ADC and PWM generations, the CPU performs background tasks by default, and is pulled into the PFC and motor control ISRs based on when the latest samples of ADC are available, and the position of tthe motor control PWM carrier.

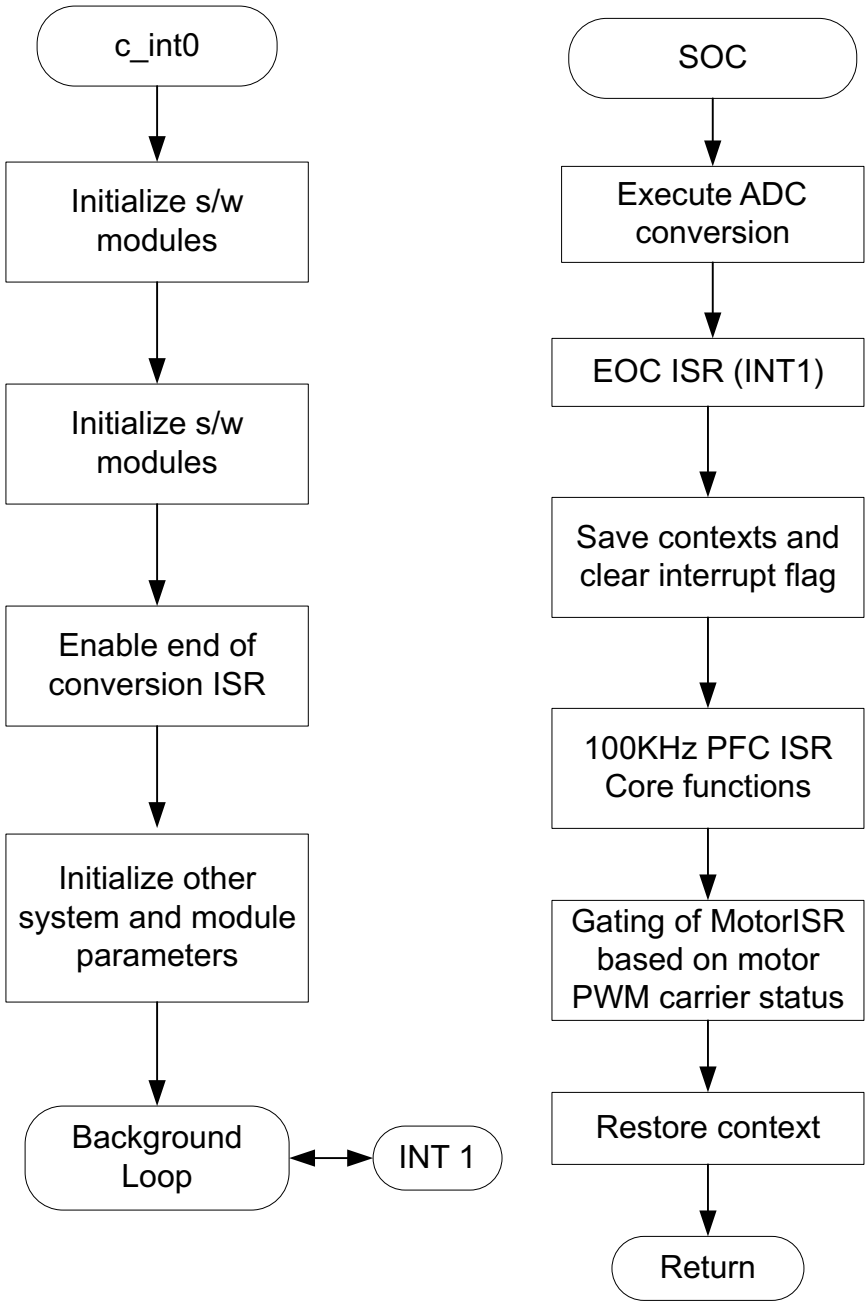


Figure 17. Software Flow Chart for PFC and Motor Control

9 PFC Software Overview

The main fast interrupt service routine (ISR) (100 kHz) for the PFC runs in an assembly environment. The motor ISR runs slower than the PFC ISR, and is made interruptible by the fast PFC ISR.

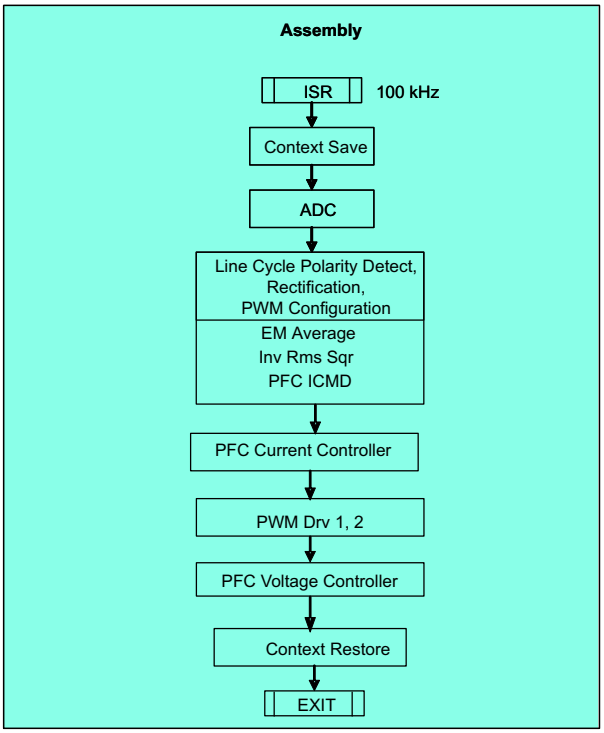


Figure 18. IL PFC Software Flow Diagram

The assembly code is strictly limited to the fast ISR, which runs the critical PFC control code. Typically, this includes reading ADC values, input line cycle polarity detect, sensed line volt rectification, control calculations, and PFC PWM updates. The slower motor control ISR in the C environment calculates the RMS input voltage and frequency of the input line voltage.

The fast ISR consists of a single file:

PFC_PM-ISR.asm – This file contains all time-critical control-type code. This file has an initialization section (one time execute) and a run-time section which executes at 100 kHz.

The slow ISR uses the following file:

SineAnalyzer.h – This file contains code for calculating the RMS voltage and frequency of the input line voltage. This file has an initialization section (one time execute) and a run-time section which executes at 10 kHz.

The power library functions (modules) are called from the fast ISR framework.

Library modules may have both a C and an assembly component. In this project, the following library modules are used. Table 2lists the C and corresponding assembly module names.

Table 2. Library Modules

C Configure Function	ASM Initialization Macro	ASM Run-Time Macro
PWM_2ch_UpDwnCnt_Cnf.c	PWMDRV_2ch_UpDwnCnt_INIT n	PWMDRV_2ch_UpDwnCnt n
ADC_SOC_Cnf.c	ADCDRV_1ch_INIT m,n,p,q	ADCDRV_1ch m,n,p,q
	PFC_InvRmsSqr_INIT n	PFC_ InvRmsSqr n
	MATH_EMAVG_INIT n	MATH_EMAVG n
	PFC_ICMD_INIT n	PFC_ICMD n
	CNTL_2P2Z_INIT n	CNTL_2P2Z n

9.1 PFC Incremental Builds

The complete CCS project for the PFC is divided into two incremental builds. This approach also simplifies the task of debugging and testing the boards.

The build options are shown in Table 3. To select a particular build option, set the parameter INCR_BUILD to the corresponding build selection as shown. This parameter is found in the Pfc_PM_Sensorless-Settings.h file. Once the build option is selected, compile the complete project by selecting rebuild-all compiler option.

Table 3. Incremental Build Options for PFC

INCR_BUILD = 1	Open loop test for boost PFC and ADC feedback (Check sensing circuitry)
INCR_BUILD = 2	Closed voltage loop and closed current loop control of IL PFC

10 Motor Control Software Overview

Figure 19 shows the software flow for sensorless PMSM control. Most of the building blocks for motor control are available as macros, while the sliding mode observer that helps to identify the position of the rotor, is available in library format.

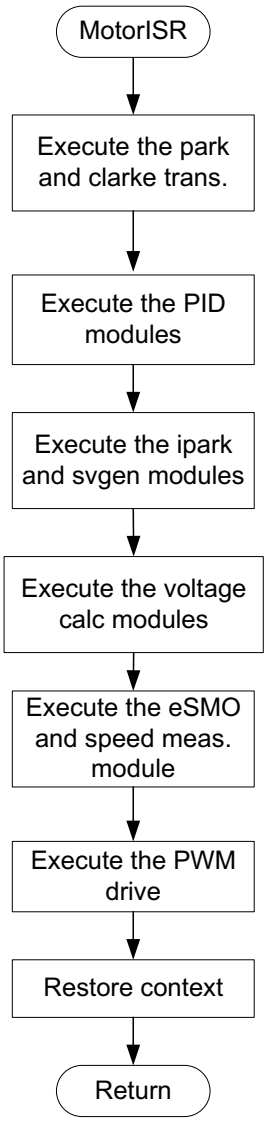


Figure 19. Software Flow for Sensorless PMSM Control

Table 4 shows the software modules and macros to demonstrate the sensorless field-oriented control of HVPM motors. The C framework is designed to run on TMS320C280x-based controllers on Code Composer Studio. The framework uses the following modules; refer to the .pdf documents in the motor control folder for the details and theoretical background of each macro.

Table 4. Software Modules and Macros

Macro Names	Explanation
CLARKE	Clarke Transformation
PARK / IPARK	Park and Inverse Park Transformation
PI	PI Regulators
RC	Ramp Controller (slew rate limiter)
RG	Ramp / Sawtooth Generator
QEP / CAP	QEP and CAP Drives (optional for speed loop tuning with a speed sensor)
SE	Speed Estimation (based on sensorless position estimation)
SPEED_FR	Speed Measurement (based on sensor signal frequency)
SMO	Sliding Mode Observer for Sensorless Applications
SVGEN	Space Vector PWM with Quadrature Control (includes IClarke Trans.)
PHASEVOLT	Phase Voltage Calculator
PWM / PWMDAC	PWM and PWMDAC Drives

10.1 PFC Incremental Builds

The complete CCS project for PFC is divided into two incremental builds. This approach simplifies the task of debugging and testing the boards.

Table 5 shows the build options. To select a particular build option, set the parameter INCR_BUILD to the corresponding build selection as shown. This parameter is found in the *Pfc_PM_Sensorless-Settings.h* file. Once the build option is selected, compile the complete project by selecting rebuild-all compiler option.

Table 5. Incremental Build Options for PFC

INCR_BUILD = 1	Open loop test for boost PFC and ADC feedback (check sensing circuitry)
INCR_BUILD = 2	Closed voltage loop and closed current loop control of IL PFC

10.2 Incremental System Build for Motor Control

The system is gradually built up so that the final system can be confidently operated. Five phases of the incremental system build are designed to verify the major software modules used in the system. Table 6 summarizes the module tested and used in each incremental system build.

Table 6. Testing Modules in Each Incremental System Build

Software Module [(1)]	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6
PWMDAC_MACRO	√	√	√	√	√	√
RC_MACRO	√	√	√	√	√	√
RG_MACRO	√	√	√	√	√	√
IPARK_MACRO	√√	√	√	√	√	√
SVGEN_MACRO	√√	√	√	√	√	√
PWM_MACRO	√√	√	√	√	√	√
CLARKE_MACRO		√√	√	√	√	√
PARK_MACRO		√√	√	√	√	√
PHASEVOLT_MACRO		√√	√	√	√	√
QEP_MACRO			√√	√	√	√
SPEED_FR_MACRO			√√	√	√	√

(1) The symbol √ means this module is being used, and the symbol √√ means this module is being tested, in this phase.

Table 6. Testing Modules in Each Incremental System Build (continued)

Software Module [1]	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6
PI_MACRO (IQ)			√√	√	√	√
PI_MACRO (ID)			√√	√	√	√
eSMO_MODULE()				√√	√√	√
SE_MACRO					√√	√
PI_MACRO (SPD)					√√	√

10.2.1 Software Setup Instructions

Refer to the Software Setup for 1AxisMtrPFC5X Kit Projects section in the 1AxisMtrPFC5X user's_guide, which can be found at:

C:\TI\controlSUITE\development_kits\TIDM-1AXIS-PFC-5x_v1.0\~Docs

1. Select 1AxisMtrPfc5x_PM_Sensorless as the active project.
2. Set the active build configuration as F2805x_FLASH.
3. Verify that the motor build is set to 1, then right-click on the project name and select Rebuild Project.
4. When the build completes, launch a debug session to load the code into the controller.
5. Open an Expressions window and add the critical variables as shown in Figure 20, and select the appropriate Q format.

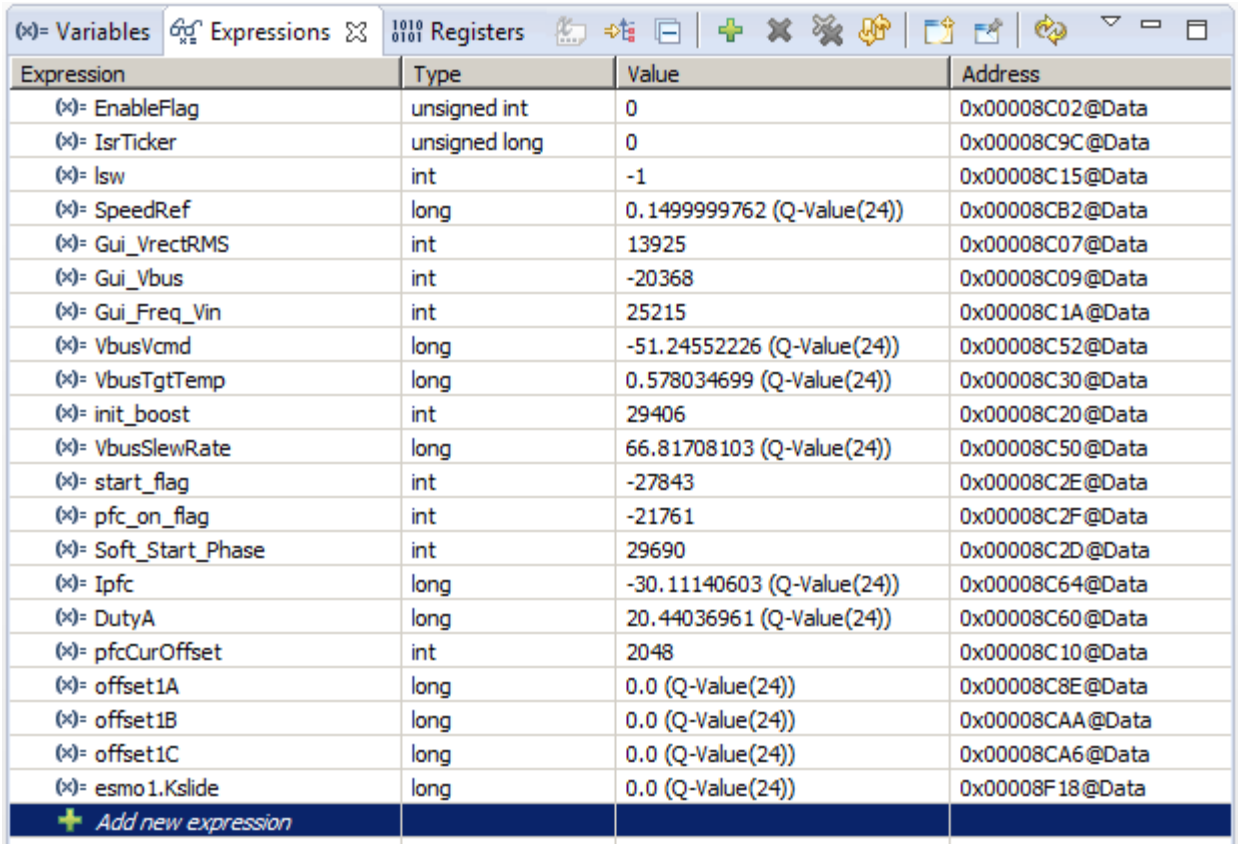


Figure 20. Expressions Window Variables

6. Alternately, a group of variables can be imported into the Expressions window as follows. Click on View, then select Scripting Console. Within Scripting Console, click the Open icon, browse to Variables_1AxisMtrPfc5x.js, and click Open. This loads the variables as shown in Figure 20.

7. Set up time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location: C:\TI\ControlSUITE\developement_kits\TIDM-1AXISMTR-PFC-5x\1AxisMtrPfc5x_PM_Sensorless.
8. Click on the Continuous Refresh button on the top-left corner of the graph tab to enable periodic capture of data from the microcontroller.

11 Procedure for Running PFC Incremental Builds

11.1 Build 1: Open-Loop PFC With ADC Measurements

The objectives of this build are:

- Evaluate PFC PWM and ADC software driver modules
- Verify the MOSFET gate driver circuit
- Verify voltage and current sensing circuits

Under this build the system runs in open-loop mode, and thus the measured ADC values are used for circuit verification and instrumentation purposes only.

11.1.1 Overview

The software in Build 1 is configured so that the user can quickly evaluate the PWM driver module (software driver) by viewing the related waveforms on a scope, and observing the effect of duty cycle change on PFC output voltage. The user can adjust the PWM duty cycle from the CCS watch window. The user can also evaluate the ADC driver module (software driver) by viewing the ADC-sampled data in the CCS watch window.

The PWM and ADC driver macro instantiations are executed inside the fast PFC ISR. Figure 21 shows the software blocks used in this build. In this board, the two PWM signals for the two PFC switches are obtained from the ePWM5 module (see Figure 9). ePWM5A drives one of the PFC switches, while ePWM5B drives the other. These PWM outputs are generated using the PWM driver module PWMDRV_2chUpDownCnt.

The PFC stage signals sensed and input to the MCU include:

- Line and neutral voltages (VL_fb, VN_fb)
- PFC input inductor current (Ipfc)
- DC bus voltage (V_{pfc})

These quantities are read using the ADC driver module and are indicated in Figure 21. The ADC driver module converts the 12-bit ADC result to a 32-bit Q24 value. A few lines of code in the ISR implements the detection of the input AC line half-cycle (positive and negative half cycles) and the rectification of the input voltage. This generates the input rectified signal Vrect, shown in Figure 21.

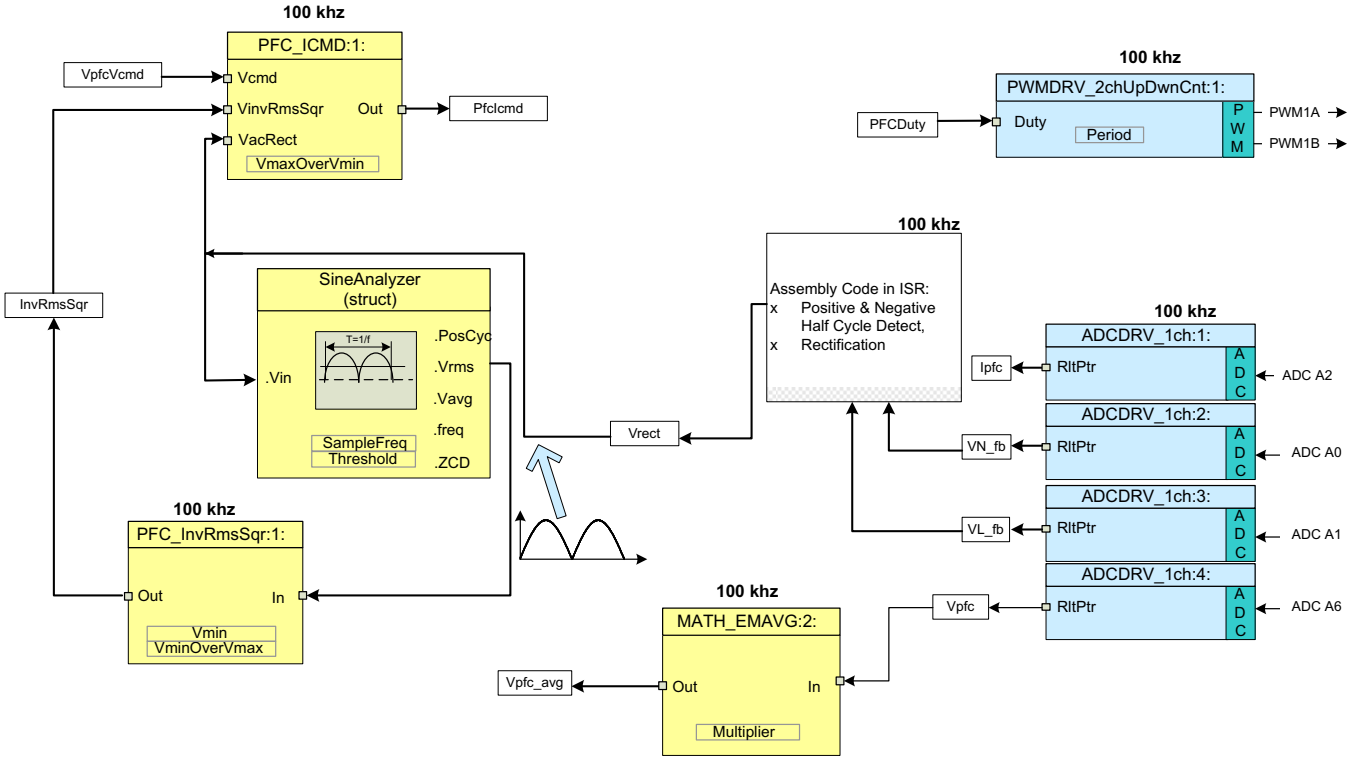


Figure 21. Build 1 Software Blocks

The PWM signals are generated at a frequency of 100 kHz, that is, a period of 10 µs. With the controller operating at 60 MHz, one count of the time base counter of ePWM1 corresponds to 16.6667 ns. This implies a PWM period of 10 µs is equivalent to 600 counts of the time base counter (TBCNT5). The ePWM5 module is configured to operate in up-down count mode. This means a time-base period value of 300 (period register value) gives a total PWM period value of 600 counts (that is, 10 µs).

The total inductor current of the PFC is sampled at the midpoint of the PWM ON pulse, because the sampled value represents the average inductor current under continuous conduction mode (CCM) condition. The voltage signal conversions are also initiated when the PFC switch is on.

All ADC results are read in the ISR by executing the ADC driver module from the 100-kHz ISR labeled as *PFC_PM-ISR.asm*.

This ISR in assembly is triggered by the ADC at the end of the ADC conversion. Inside the ISR *PWMDRV_2ch_UpDwnCnt*, macros are executed and the PWM compare shadow registers are updated. These registers are loaded into the active register at the next TBCNT5 = ZERO event.

11.1.2 Protection

An overvoltage protection (OVP) mechanism is implemented in software for this PFC stage.

The sensed DC bus output voltage from the ADC input is compared against the OVP threshold set by the user. The OV threshold point for this kit has been set to 410 V. In case of an OV condition, the PWM outputs are shut off using the TZ (trip zone) registers. The flexibility of the trip mechanism on C2000 devices provides the possibilities for taking different actions on different trip events. In this project, both PWM outputs are driven low in case of a trip event. Both outputs are held in this state until a device reset is executed.

1. Select the PFC incremental build option as 1.
2. Click the Project → Rebuild All button and watch the tools run in the build window. The program is compiled and loaded into the flash. You should now be at the start of Main().
3. Enable real-time mode in CCS.

4. Add the PFC-related variables DutyA, start_flag, Gui_Vbus, Gui_Freq_Vin, and Gui_VrectRMS in the watch window. Set the Q format for the DutyA as Q24. Set the Q format for three parameters Gui_Vbus, Gui_Freq_Vin, and Gui_VrectRMS as Q6.
5. Click on the Continuous Refresh buttons for the watch view.
6. With the AC power off, run the code by using the Run button on the toolbar.
7. Set EnableFlag to 1.
8. In the watch view, check the variable DutyA. This should be set to 0.1 (=1677721 in Q24). This variable sets the duty cycle for the PFC converter. Use an oscilloscope to verify the PWM outputs at the PFC MOSFETs gate (gate to source voltage). These PWM outputs should show 100-kHz PWM with a duty ratio of 10%.
9. Apply a resistive load to the PFC system at the DC output (8 k/20 W, or 4 k/40 W, or 2 k/80 W).
10. Use a fully isolated AC source and slowly apply AC power to the board. Slowly increase input AC voltage to 100 V AC, 50/60 Hz. Use a multimeter (DMM) to measure the DC bus voltage and input RMS voltage. Verify these voltage readings with the corresponding values shown in the watch window. For example, input RMS voltage measured with a voltmeter should match the watch window parameter Gui_VrectRMS. Input AC frequency (set at the AC source) should also match that displayed at the watch window using the variable Gui_Freq_Vin.
11. Use DutyA to slowly change the duty from the watch window. The boost converter output voltage should change accordingly. Verify that the measured values and the watch window parameters again match.
12. Reduce the AC input voltage to zero and turn off the AC source.

CAUTION

In Step 10, observe the output voltage carefully, as this should not be allowed to exceed the maximum voltage rating of the board (400 V).

11.2 Build 2: PFC With Closed Voltage and Current Loop

Figure 22 shows the software blocks used in this build. Notice the additional software blocks added to the Build 1 diagram (see Figure 21) to implement this closed-current and voltage-loop system. The *SineAnalyzer* block calculates the RMS voltage and frequency of the input voltage. The *PFC InvRmsSqr* block calculates the inverse of the square of the RMS input voltage. This calculated value, together with the rectified voltage (Vrect), is used in the software block *PFC_ICMD* to generate the reference current command *PfcIcmd* for the PFC current control loop. The *PFC_ICMD* block uses a third input *VpfcVcmd* to control the magnitude of the reference current command. This parameter *VpfcVcmd* is connected to voltage loop controller output, which adjusts the magnitude of the reference current, and thus the PFC bus voltage. Two different 2p2z (two pole two zero) controllers are used to implement the voltage and the current control loops. These are software blocks shown in Figure 22 as *CNTL_2P2Z:1* and *CNTL_2P2Z:2*, for current and voltage loops, respectively.

Figure 22 shows the current loop control block is executed at a rate of 100-KHz. CNTL_2P2Z is a second-order compensator realized from an IIR filter structure. This function is independent of any peripherals and thus does not require a CNF function call.

This 2p2z controller requires five control coefficients. These coefficients, and the clamped output of the controller, are stored as the elements of a structure named *CNTL_2P2Z_CoefStruct1*. The CNTL_2P2Z block can be instantiated multiple times if the system requires multiple loops. Each instance can have a separate set of coefficients. The *CNTL_2P2Z* instance for the current loop uses the coefficients stored as the elements of structure *CNTL_2P2Z_CoefStruct1*. This way, a second instantiation of *CNTL_2P2Z* with a different structure, *CNTL_2P2Z_CoefStruct2*, can be used for PFC voltage loop control.

The controller coefficients can be changed directly by modifying the values for B0, B1, B2, A1, and A2 inside the structure *CNTL_2P2Z_CoefStruct1*. Alternately, the 2p2z controller can be expressed in PID form, and the controller coefficients changed by changing the PID coefficients. The following equations relate the five controller coefficients to the three PID gains. For the current loop, these P, I, and D coefficients are named as: Pgain_I, Igain_I, and Dgain_I, respectively. For the voltage loop, these coefficients are named as: Pgain_V, Igain_V, and Dgain_V, respectively. These coefficients are used in Q26 format.

The compensator block (*CNTL_2P2Z*) has a reference input and a feedback input. The feedback input labeled as *Fdbk* comes from the ADC driver block. The reference input labeled as *Ref* comes from the *PFC_ICMD* block as mentioned earlier. The z-domain transfer function for *CNTL_2P2Z* is given by:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{6}$$

The recursive form of the PID controller is given by the difference equation:

$$u(k) = u(k-1) + b_0 e(k) + b_1 e(k-1) + b_2 e(k-2)$$

where

- $b_0 = K_p + K_i + K_d$
- $b_1 = -K_p + K_i - 2K_d$
- $b_2 = K_d$ (7)

And the z-domain transfer function of this PID is:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - z^{-1}} \tag{8}$$

Comparing this with the general form, we can see that PID is a special case of CNTL_2P2Z control, where:

$a_1 = -1$ and $a_2 = 0$

The *MATH_EMAVG* (exponential moving average) block calculates the average of the output DC bus voltage. The output from this block is used to detect an overvoltage condition followed by a PWM shutdown.

Figure 22 shows the software block labeled as *CNTL_2P2Z:2*. This is the second instantiation of the 2p2z control block, to implement the PFC voltage loop control. This voltage loop controller is executed at a 50-kHz rate, which is half the rate for the current loop. The output from this control block drives the input node *VpfcVcmd* of the *PFC_ICMD* block.

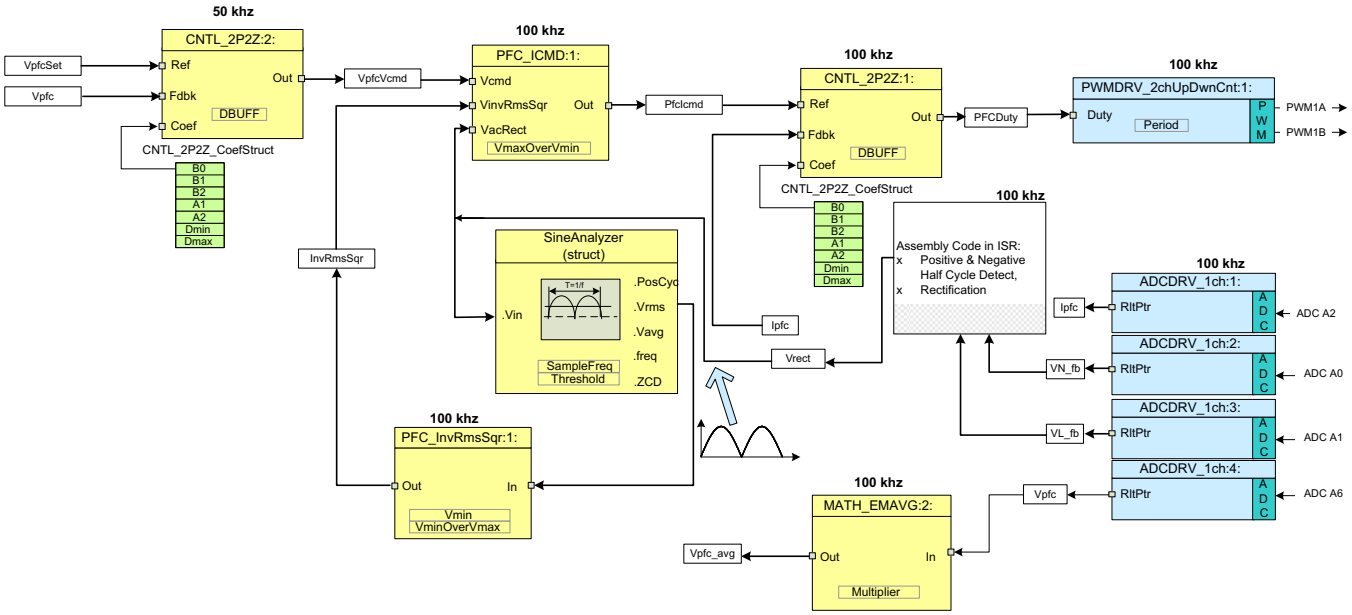


Figure 22. Build 2 Software Blocks

Similar to the current loop controller, this voltage loop controller, *CNTL_2P2Z:2*, also requires five control coefficients. These coefficients and the clamped output of the controller are stored as the elements of a second structure named *CNTL_2P2Z_CoefStruct2.* The coefficients for this controller can be changed directly by modifying the values for B0, B1, B2, A1, and A2 inside the structure *CNTL_2P2Z_CoefStruct2,* or by changing the equivalent PID gains as discussed earlier.

11.2.1 Start-up, Inrush Current Control, and Slew-Limit

At start-up, the controller monitors the variable start_flag and PFC DC bus voltage. When the start_flag is set to 1 and the DC bus voltage reaches a minimum level (the minimum level for this PFC is set around 160 Vdc), PFC action is enabled and the output DC bus slowly ramps up to the preset value of approximately 390 Vdc. This output voltage level is set by the constant VBUS_RATED_VOLTS, defined in the header file *Pfc_PM_Sensorless-Settings.h*. The ramp-up speed is set by the parameter *VbusSlewRate*.

11.3 Build and Load Project

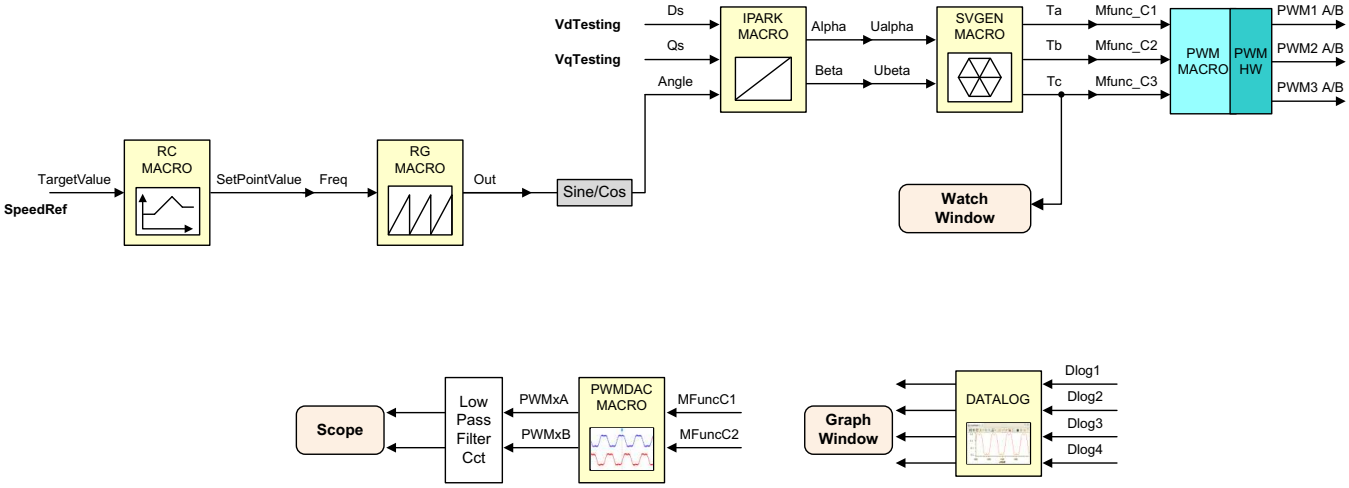
Follow these steps to execute this build:

1. Follow steps 1 through 7 exactly as in build 1 (see Section 11.1.2), the only difference being selecting the build 2 option instead of build 1 in the first step.
2. Apply an appropriate resistive load to the PFC output. For example, allow for four power resistors, in parallel, across the PFC output, each with a rating of 2.0 kΩ and 160 W (each). This provides a total load of approximately 300 W at 390-V bus voltage. Initially, only use two resistors before the PFC powers up, so that the PFC powers up with an initial load less than 150 W.
3. Configure the isolated AC source to output 120-V, 60-Hz AC voltage output. Use a voltmeter to monitor the DC bus voltage.
4. Turn on the AC source output for 120 Vrms.
5. When the DC bus voltage reaches 160 V or higher, set the start_flag to 1. PFC action will start, and the bus voltage will slowly increase to approximately 390 V. As in build 1, verify the watch window values for Gui_Vbus and Gui_VrectRMS with those from the multimeter (DMM). The frequency reading from the AC source should also match the watch widow variable Gui_Freq_Vin. The variable pfc_on_flag is set to 1 when the soft-start is complete and the PFC bus voltage reaches the desired set point.
6. Connect two more power resistors across the PFC output to increase the PFC load to 300 W.

7. Use an oscilloscope with a voltage probe (high-voltage differential probe) and a current probe to observe the input voltage and input current waveforms. With 120-Vrms input, 500-Ω resistive load, and bus voltage set to 390 V, the power factor (PF) should be greater than 0.95.
8. Vary the input voltage (100 Vrms~260 Vrms) or the load resistance (0~600 W) to see the PFC operation under closed current and voltage control loop. The bus voltage should be regulated at 390 V.
9. Reduce the AC input voltage to zero and turn off the AC source.

12 Procedure for Running Motor Control Incremental Builds

12.1 Level 1 Incremental Build



Level 1 verifies the target-independent modules, duty cycles, and PWM updates. The motor is disconnected at this level.

Figure 23. Level 1 Incremental System Build Block Diagram

At this step, keep the motor disconnected. Assuming the load and build steps described in the 1AxisMtrPFC5X user's guide completed successfully, this section describes the steps for a minimum system check-out, which confirms operation of system interrupt, the peripheral and target-independent I_PARK_MACRO (inverse park transformation), SVGEN_MACRO (space vector generator), and the peripheral-dependent PWM_MACRO (PWM initializations and update) modules.

1. Open Pfc_PM_Sensorless-Settings.h, and select the level 1 incremental build option by setting the BUILDLEVEL to LEVEL1 (#define BUILDLEVEL LEVEL1).
2. Right-click on the project name, and click Rebuild Project.
3. When the build completes, click the Debug button, reset the CPU, restart, enable real-time mode, and run the program.
4. Add variables to the Expressions window by opening Variables_1AxisMtrPfc5x.js, located in the root directory, from the Scripting Console window.
5. Set the EnableFlag to 1 in the Expressions window. The IsrTicker variable counts upwards, and when it stops Isr1Ticker continues increasing. This confirms that the system interrupt is working properly.

Notice the variable lsw in the Expressions window. This variable controls the motor as follows:

- –1: All bottom switches of inverter ON, no display variables are updated.
- 0: DC current flows in the motor, bringing the PM motor to initial alignment.
- 1: Forces rotation of the PM motor by applying sinusoidal PWM voltage on the motor.
- 2: Sensorless control of motor -> rotor flux position determines the angle of applied voltage.

In the software, the key variables to be adjusted are summarized as follows:

- SpeedRef (Q24): for changing the rotor speed in per-unit
- VdTesting (Q24): for changing the d-qxis voltage in per-unit
- VqTesting (Q24): for changing the q-axis voltage in per-unit

12.1.1 Level 1A (SVGEN_MACRO Test)

Because the motor is disconnected, the significance of the lsw variable is limited to enabling PWM algorithms and updating display variables by changing to any value other than –1. Thus, change lsw to any value value other than –1. The SpeedRef value is specified to the RG_MACRO module using the RC_MACRO module. The IPARK_MACRO module generates the outputs to the SVGEN_MACRO module. Three outputs from SVGEN_MACRO module are monitored through the graph window, as shown in Figure 24, where the Ta, Tb, and Tc waveforms are 120° apart from each other. Specifically, Tb lags Ta by 120° and Tc leads Ta by 120°. Check the PWM test points on the board to observe PWM pulses (PWM-1H to 3H and PWM-1L to 3L) and ensure that the PWM module is running properly.

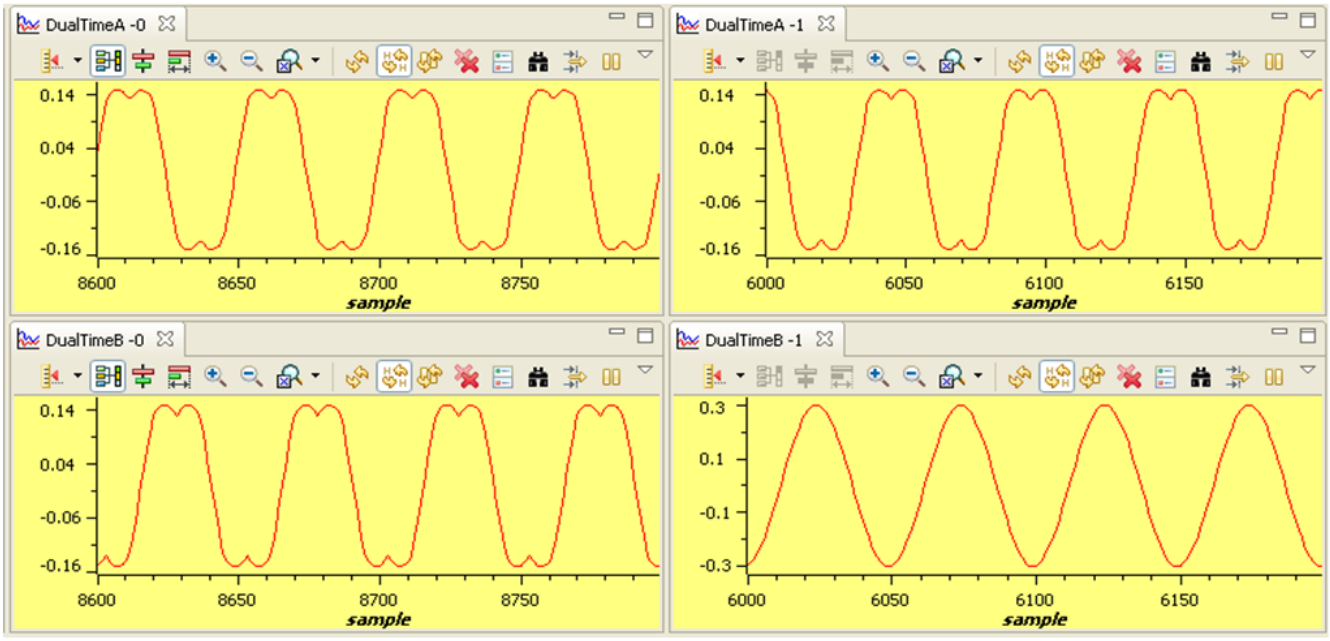


Figure 24. Output of SVGEN, Ta, Tb, Tc, and Tb-Tc Waveforms

12.1.2 Level 1B (Testing the PWMDAC Macro)

To monitor internal signal values in real time, PWM DACs are very useful tools. PWM DACs use an external low-pass filter, such as a simple first-order RC filter, to filter out the high-frequency PWM carrier and show the modulated waveform. These are brought out on test points TP4 to TP7. The selection of R and C values (or the time constant, τ) is based on the cut-off frequency (f_c). For this type of filter the relation is as follows:

$$\tau = RC = \frac{1}{2\pi f_c} \tag{9}$$

For example, if R = 1.8 kΩ and C = 100 nF, Equation 9 gives f_c = 884.2 Hz. This cut-off frequency must be below the PWM frequency. Using Equation 9, one can customize low-pass filters used for signals being monitored. Macro block M8 has the DAC circuit low-pass filters with 2.2 kΩ and 220 nF, bringing out filtered analog signals on TP4 to TP7. For more details, refer to application note SPRAA88.

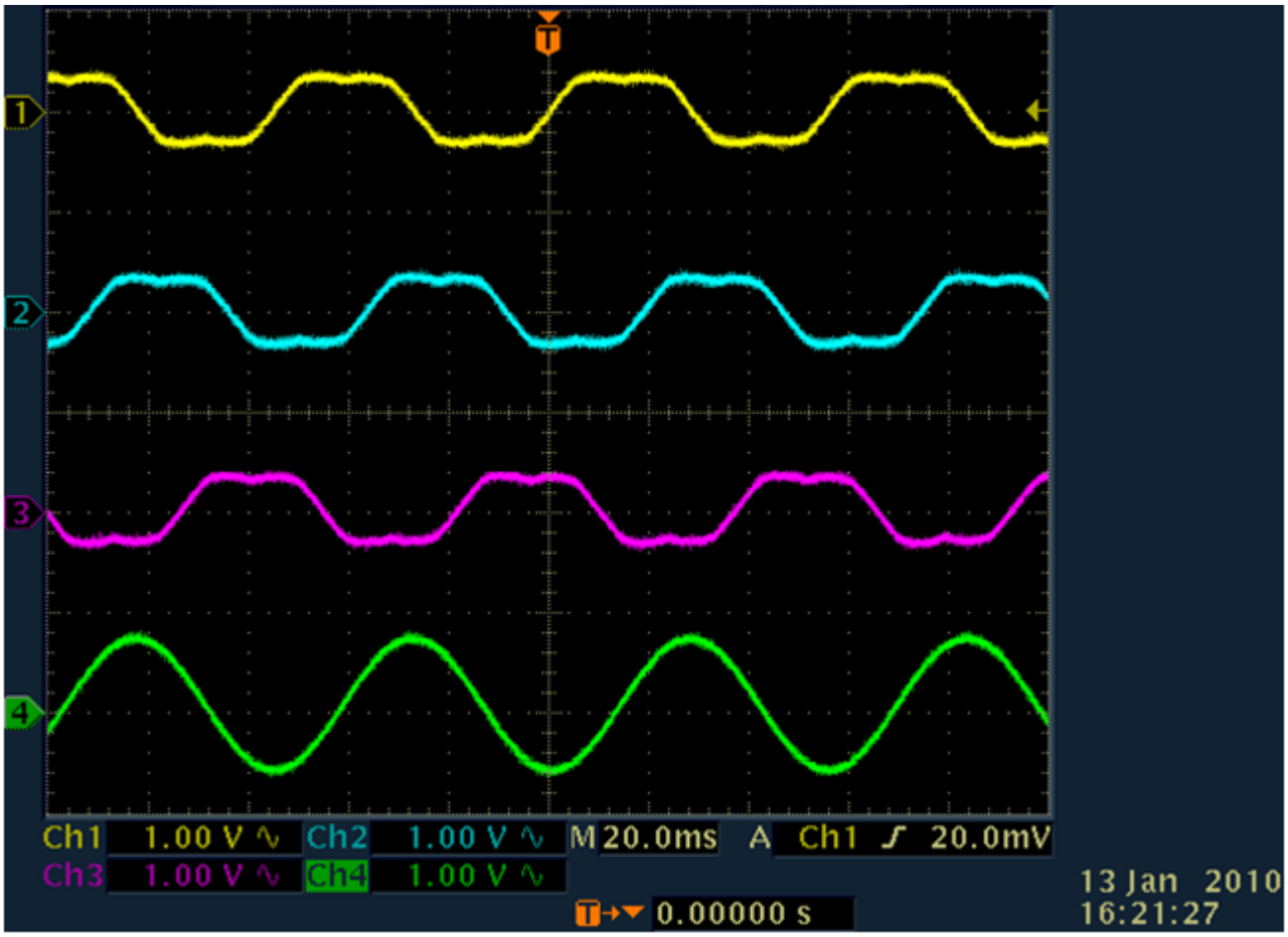


Figure 25. DAC 1 – 4 Outputs Showing Ta, Tb, Tc, and Tb-Tc Waveforms

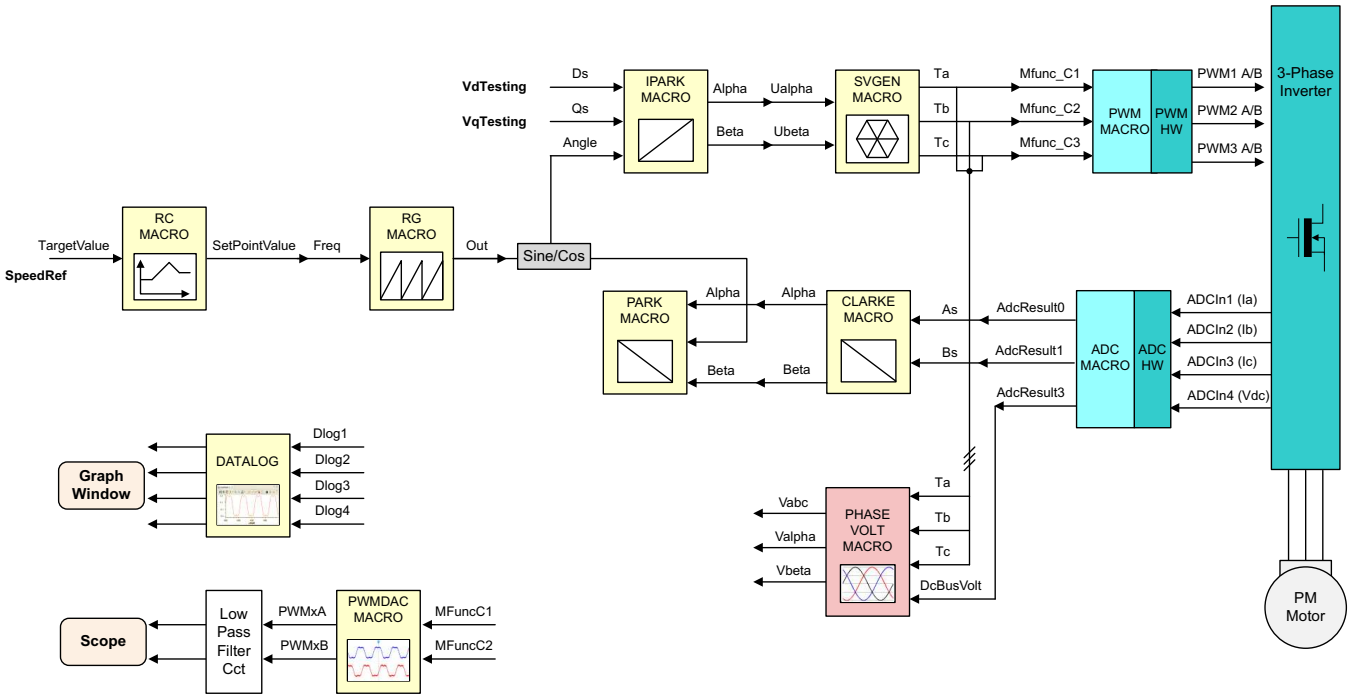
12.1.3 Level 1C (PWM_MACRO and INVERTER Test)

After verifying the SVGEN_MACRO module in level 1a, the PWM_MACRO software module and the 3-phase inverter hardware are tested by looking at the low-pass filter outputs. For this purpose, if using the external DC power supply, gradually increase the DC bus voltage and check the Vfb-U, Vfb-V, and Vfb-W test points using an oscilloscope; if using AC power entry, slowly change the variac to generate the DC bus voltage. Once the DC bus voltage is greater than 15 V to 20 V, start observing the inverter phase voltage dividers and waveform monitoring filters (Vfb-U, Vfb-V, Vfb-W), enable the generation of the waveform, and ensure the inverter is working appropriately. The default RC values are optimized for AC motor state observers employing phase voltages.

CAUTION

After verifying this, reduce the DC bus voltage, take the controller out of real-time mode (disable), reset the processor (see the 1AxisMtrPFC5X user's guide for details). After each test, this step must be repeated for safety purposes. Improper shutdown might halt the PWMs at some certain states where high currents can be drawn, thus take caution while performing these experiments.

12.2 Level 2 Incremental Build



Level 2 verifies the analog-to-digital conversion, offset compensation, Clarke / Park transformations, and phase voltage calculations.

Figure 26. Level 2 Incremental System Build Block Diagram

Assuming build 1 completes successfully, this section verifies the analog-to-digital conversion, Clarke and Park transformations, and phase voltage calculations. Now the motor can be connected to the eval board, as the PWM signals are successfully proven through the level 1 incremental build. The open-loop experiments are meant to test the ADCs, inverter stage, software modules, and so forth. Thus, running the motor under load or at various operating points is not recommended.

1. Open Pfc_PM_Sensorless-Settings.h, and select the level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL LEVEL2).
2. Right-click on the project name, and click Rebuild Project.
3. When the build completes, click the Debug button, reset the CPU, restart, enable real-time mode, and run the program.
4. Set the EnableFlag to 1 in the Expressions window.
5. The IsrTicker variable continues increasing; confirm this by watching the variable in the Expressions window. While this occurs, the CPU computes the offset of the analog channels, sensing phase-current feedback.
6. Isr1Ticker continues to count, indicating the main control ISR is running. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are:

- SpeedRef (Q24): for changing the rotor speed in per-unit
- VdTesting(Q24): for changing the d-qxis voltage in per-unit
- VqTesting(Q24): for changing the q-axis voltage in per-unit

During the open-loop tests, VqTesting, SpeedRef, and the DC bus voltages should be adjusted carefully for PM motors so that the generated Bemf is lower than the average voltage applied to motor winding. This prevents the motor from stalling or vibrating. Set lsw to 1 to apply sinusoidal PWM voltage on the motor.

12.2.1 Phase 2A – Setting Overcurrent Limit in Software

The F2805x has on-chip programmable comparators in the analog front end (AFE) that can detect positive and negative overcurrents. Thus, overcurrent monitoring is provided for shunt signals, to generate TRIP signals to shut down the inverter. To avoid spurious overcurrent tripping, use the programmable digital filter that qualifies the comparator output before generating the TRIP signal. The reference to the comparator is user-programmable for both positive and negative current limits. The digital filter module qualifies the comparator output signal by periodically verifying the genuineness of the signal over a certain period of time, by counting the output for a certain number of count times within a certain count window, where the periodicity, count, and count window are user-programmable.

In the Expressions window, some new variables are added:

- clkPrescale sets the frequency of sampling of digital filter.
- sampwin sets the count window.
- thresh sets the minimum count to qualify the signal within sampwin.
- SHUNT_curHi sets the positive current maximum through the SHUNT current sensor.
- SHUNT_curLo sets the negative current maximum through the SHUNT current sensor.

NOTE: The median value corresponding to zero current is 32.

TripFlagDMC is a flag variable used to represent the overcurrent trip status of the inverter. If this flag is set, the user can adjust the preceeding settings and retry running the inverter by setting clearTripFlagDMC to 1. This clears TripFlagDMC and restarts the PWMs.

The default current limit setting is to shutdown at 10 A. The user can change any of these settings to suit their system. Once satisfactory values are identified, write down and modify the code with these new values, and rebuild and load for further tests.

12.2.2 Level 2B – Testing the Phase Voltage Module

The phase voltage calculation module, PHASEVOLT_MACRO, can now be tested. Gradually increase the DC bus voltage. The outputs of this module can be checked using the graph window, as shown in Figure 27.

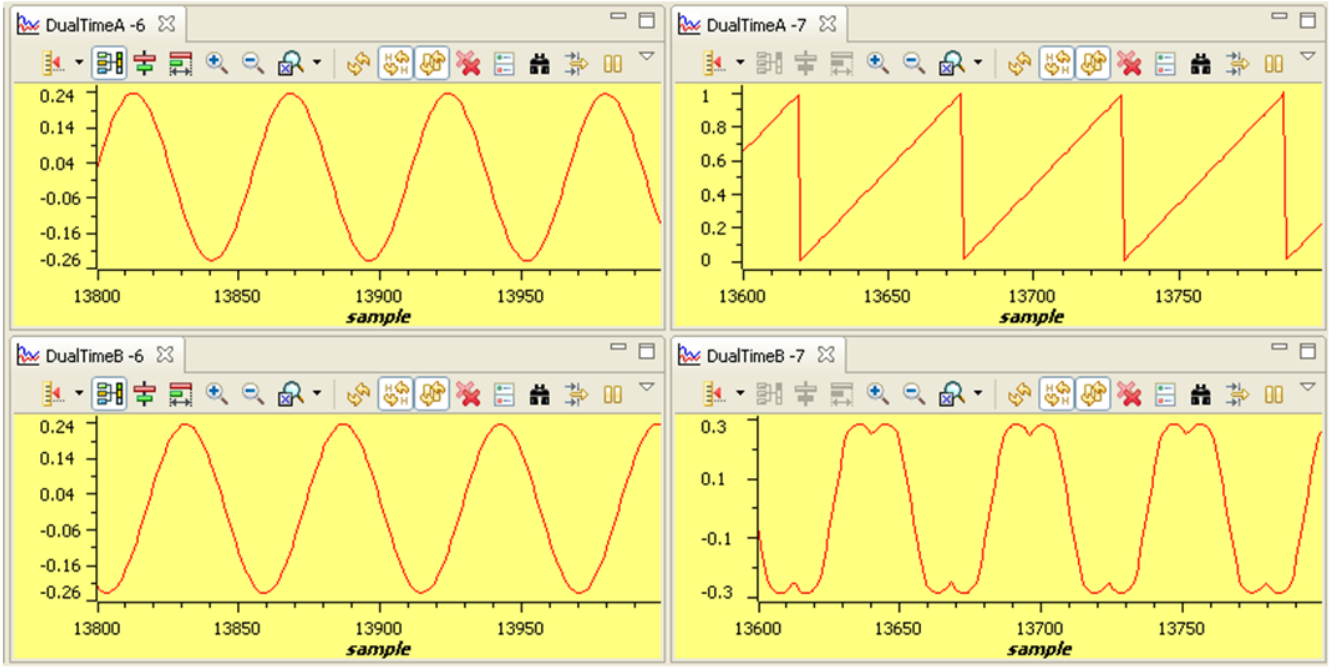


Figure 27. Calculated Phase A and B Voltages by volt1 Module, rg1.Out and svgen_dq1.Ta

- The VphaseA, VphaseB, and VphaseC waveforms should be 120° apart from each other. Specifically, VphaseB lags VphaseA by 120° and VphaseC leads VphaseA by 120°.
- The Valpha waveform and the VphaseA waveform should be the same.
- The Valpha waveform should lead the Vbeta waveform by 90° at the same magnitude.

12.2.3 Phase 2C – Testing the Clarke Module

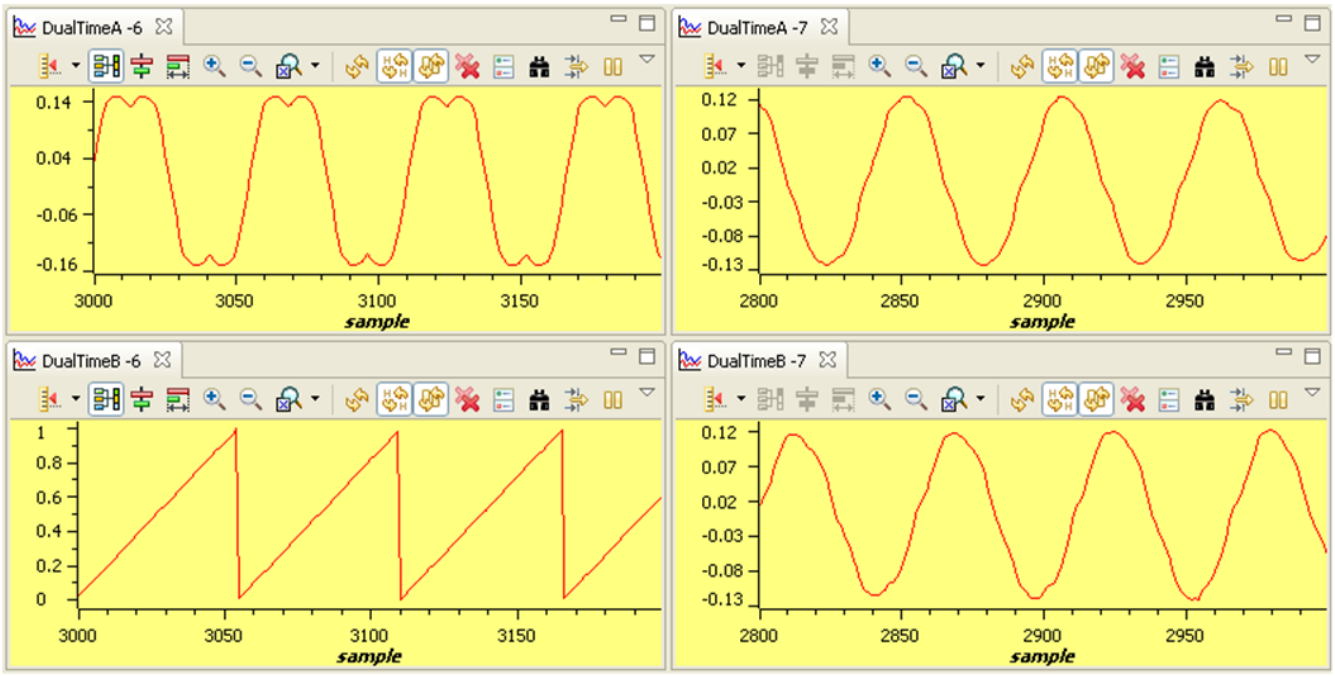
In this section, the Clarke module is tested. The three measured line currents are transformed to two phase dq currents in a stationary reference frame. The outputs of this module can be checked from a graph window.

- The clark1.Alpha waveform should be the same as the clark1.As waveform.
- The clark1.Alpha waveform should lead the clark1.Beta waveform by 90° at the same magnitude.

The measured line current must be trailing with the reconstructing phase voltage, because of the nature of the AC motor. This can be checked as follows:

- The clark1.Alpha waveform should trail the Valpha waveform at an angle by nature of the reactive load of the motor.
- The clark1.Beta waveform should trail the Vbeta waveform at the same angle.

If the clark1.Alpha and Valpha or clark1.Beta and Vbeta waveforms in the previous step are not truly affecting the trailing relationship, then set OutofPhase to 1 at the beginning of the PHASEVOLT_MACRO module. The outputs of this test can be checked using the graph window.



Deadband = 1.66 µsec, Vdcbus = 300 V , dlog.prescalar = 3

Figure 28. The Waveforms of Svgen_dq1.Ta, rg1.Out, and Phase A and B Currents

12.2.4 Level 2D – Adjusting PI Limits

The vectorial sum of the d-q PI outputs should be less than 1.0, which refers to the maximum duty cycle for the SVGEN macro. Another limiting factor of the duty cycle is the current sense through shunt resistors, which depends on hardware and software implementation. Depending on the application requirements, three, two, or a single shunt resistor can be used for current waveform reconstruction. A higher number of shunt resistors allow for higher duty cycle operation and better DC bus use.

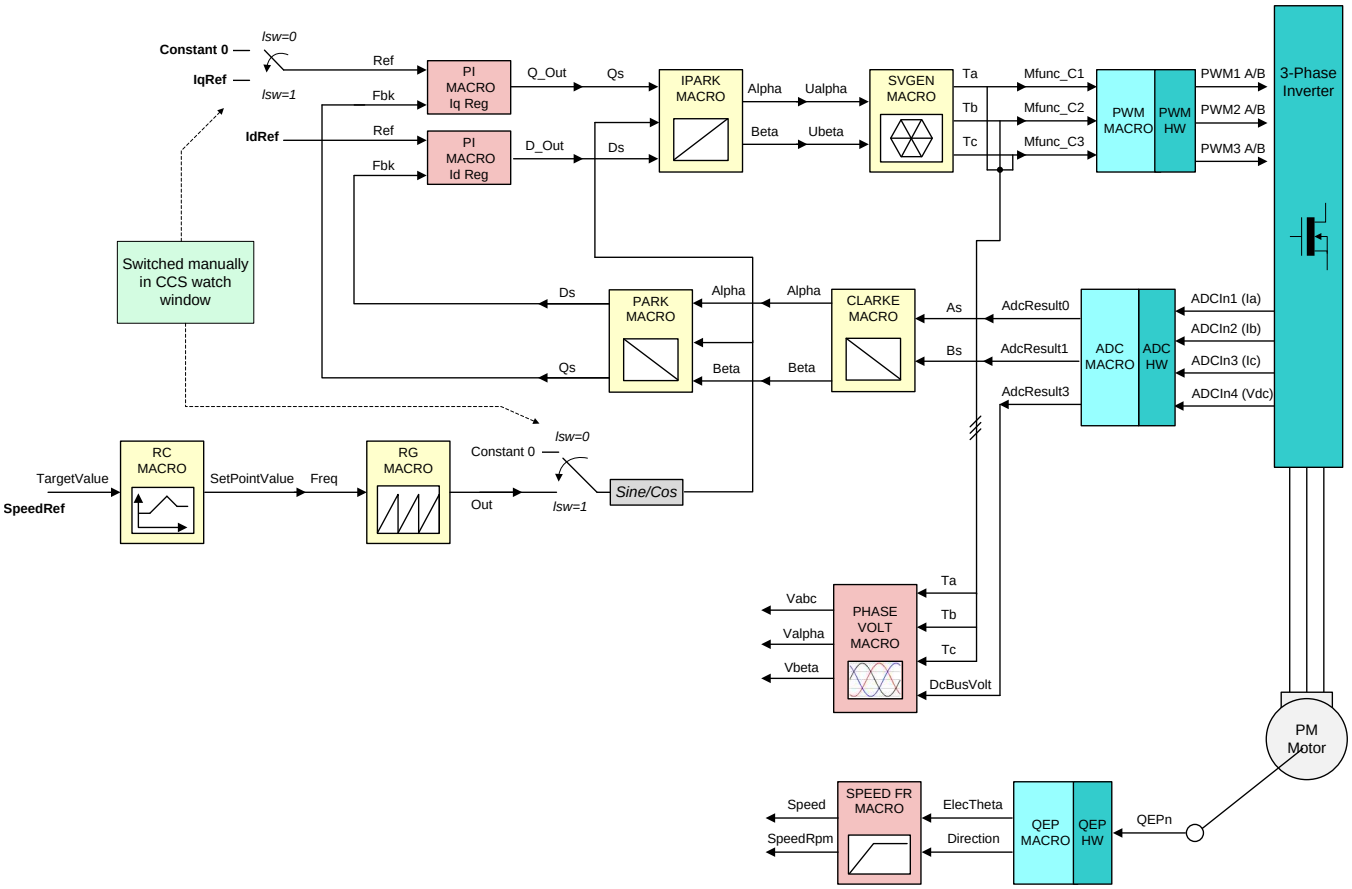
1. Run the system with default VdTesting, VqTesting, and SpeedRef, and gradually increase the VdTesting and VqTesting values.
2. Watch the current waveforms in the graph window. Keep increasing the VdTesting and VqTesting values until you notice distorted current waveforms, and write down the maximum allowed.
3. Ensure that these values are consistent with expected d-q current component maximums while running the motor.
4. After this motor build, PI outputs automatically generate the voltage reference and determine the PWM duty cycle depending on the d-q current demand, thus set pi_id.Umax/min and pi_iq.Umax/min according to recorded VdTesting and VqTesting values.

CAUTION

Running the motor without proper PI limits can yield distorted current waveforms and unstable closed-loop operations, which may damage the hardware.

5. Bring the system to a safe stop, as described at the end of build 1, by reducing the bus voltage, taking the controller out of real-time mode, and reset.

12.3 Level 3 Incremental Build



Level 3 verifies the dq-axis current regulation performed by PI macros and speed measurement macros

Figure 29. Level 3 Incremental System Build Block Diagram

Assuming the previous section completes successfully, this section verifies the dq-axis current regulation performed by the PI modules and speed measurement modules using QEP (optional). To confirm the operation of current regulation, the gains of these two PI controllers must be tuned for proper operation.

1. Open Pfc_PM_Sensorless-Settings.h, and select the level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3).
2. Right-click on the project name, and click Rebuild Project.
3. When the build completes, click the Debug button, reset the CPU, restart, enable real-time mode, and run the program.
4. Set EnableFlag to 1 in the Expressions window. The IsrTicker variable will continue increasing; confirm this by watching the variable in the Expressions window. While this occurs, the CPU computes the offset of the analog channels sensing phase-current feedback. After this is done, Isr1Tickerill continues to count, indicating the main control ISR is running. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are:

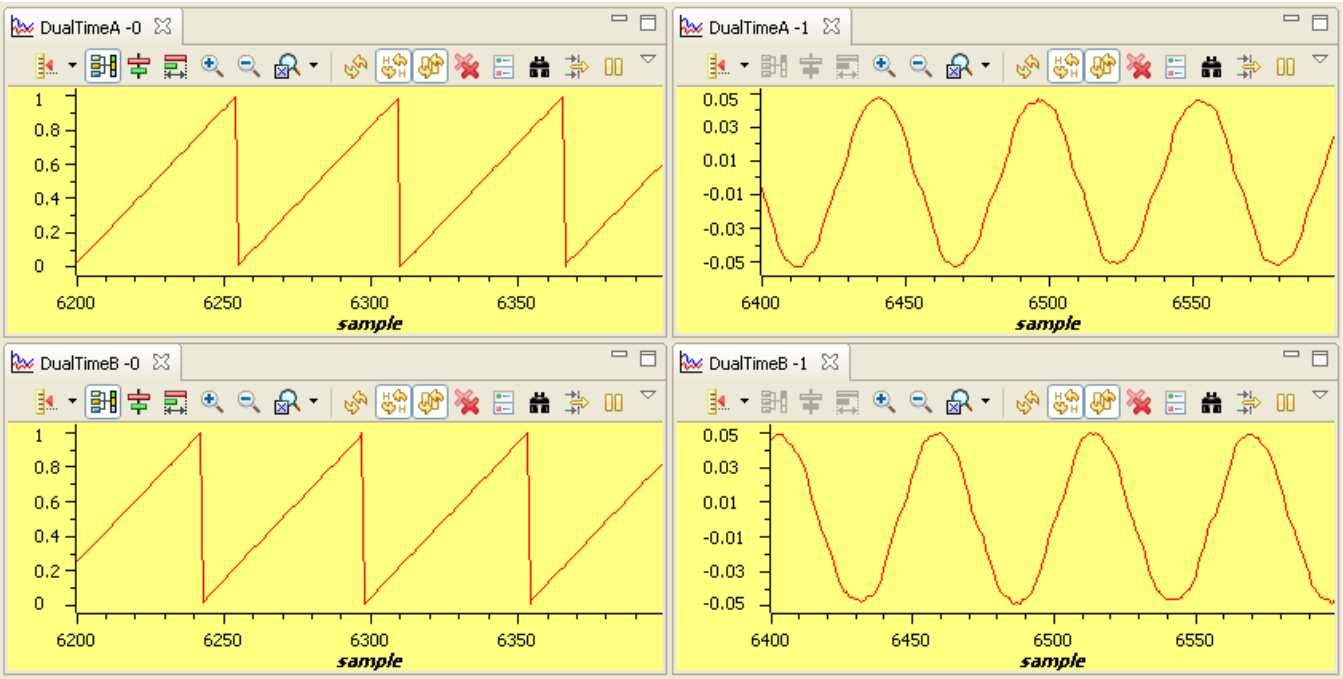
- SpeedRef (Q24): for changing the rotor speed in per-unit
- IdRef(Q24): for changing the d-qxis voltage in per-unit
- IqRef(Q24): for changing the q-axis voltage in per-unit

In this build, the motor is supplied by AC input voltage, and the (AC) motor current is dynamically regulated by using the PI module through the Park transformation on the motor currents.

The steps are explained as follows:

1. Compile, load, and run the program with real-time mode.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different), Idref to zero, and Iqref to 0.05 pu.
3. Gradually increase voltage at the variac or DC power supply to get an appropriate DC bus voltage.
4. Add the soft-switch variable Isw to the Expressions window to switch from current loop to speed loop. In the code, Isw manages the loop setting as follows:
   - Isw=0, lock the rotor of the motor
   - Isw=1, run the motor with closed-current loop
5. Check pi_id.Fbk in the Expressions windows with the continuous refresh feature, whether or not it should be keeping track of pi_id.Ref for the PI module. If not, adjust its PI gains properly.
6. Check pi_iq.Fbk in the Expressions windows with the continuous refresh feature, whether or not it should be keeping track of pi_iq.Ref for the PI module. If not, adjust its PI gains properly.
7. Confirm these two PI modules, by trying different values of pi_id.Ref and pi_iq.Ref, or SpeedRef.
8. For both PI controllers, the proportional, integral, derivative, and integral correction gains may be re-tuned for the satisfied responses.
9. Bring the system to a safe stop as described at the end of build 1, by reducing the bus voltage, taking the controller out of real-time mode, and resetting.
10. The motor should stop. Once stopped, terminate the debug session.

While running this build, the current waveforms in the CCS graphs should appear as shown in Figure 30.



Deadband = 1.66 µsec, Vdcbus = 300 V , dlog.trig_value = 100

Figure 30. rg1.Out, Measured Theta and Phase A and B Current Waveforms

12.3.1 Level 3B – QEP / SPEED_FR Test

This section verifies the QEP1 driver and its speed calculation. The QEP drive macro determines the rotor position and generates a direction (of rotation) signal from the shaft position encoder pulses. Ensure that the output of the incremental encoder is connected to [Main]-H1, and the QEP and SPEED_FR macros are initialized properly in the Pfc_PM_Sensorless.c file, depending on the features of the speed sensor. Refer to the .pdf files regarding the details of related macros in motor control folder (C:\TI\controlSUITE\libs\app_libs\motor_control). The steps to verify these two software modules related to the speed measurement are:

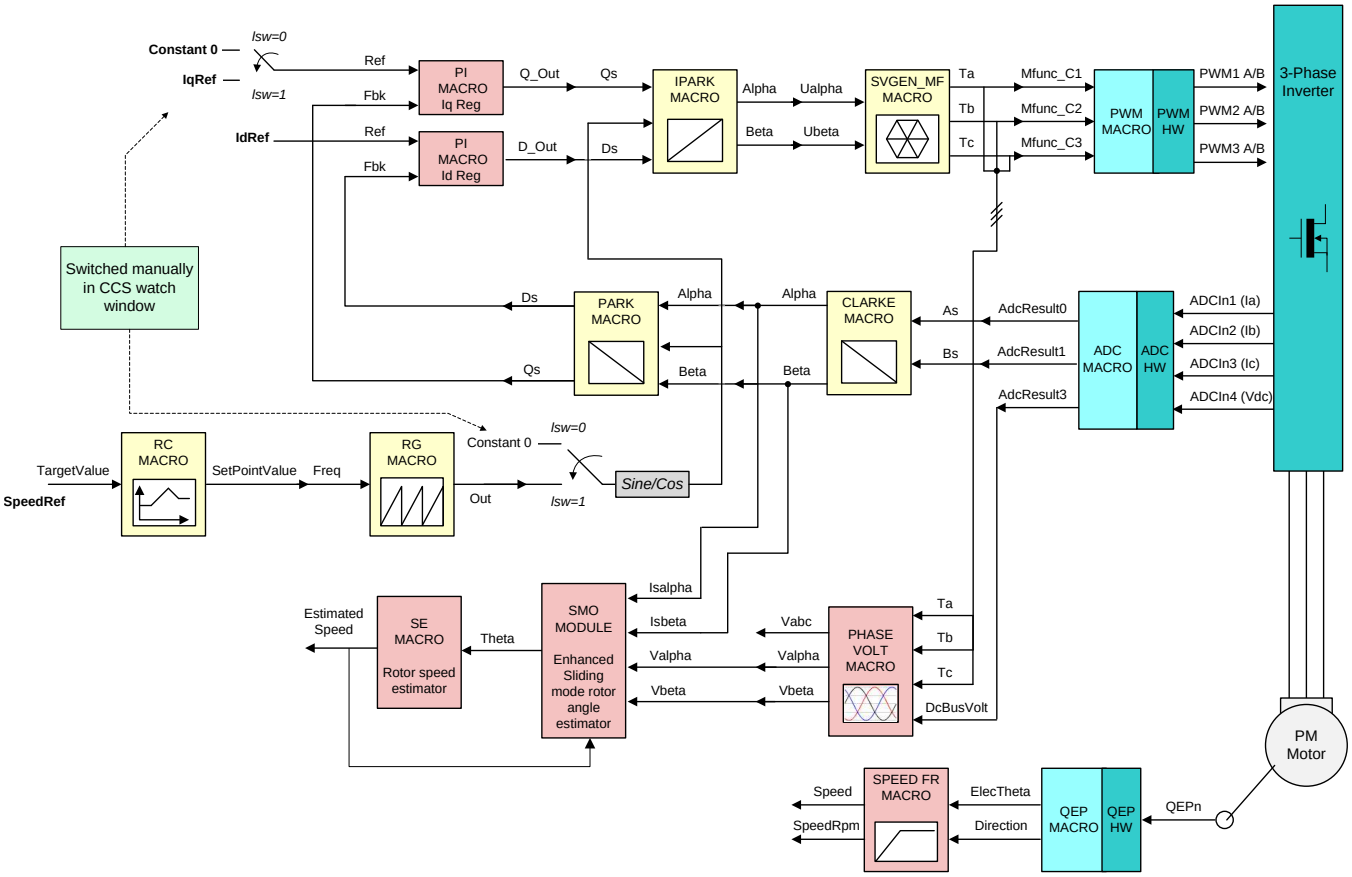
1. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
2. Compile, load, and run the program with real-time mode, then increase the voltage at the variac or DC power supply for the appropriate DC bus voltage.
3. Add the soft-switch variable lsw to the Expressions window to switch from current loop to speed loop. In the code, lsw manages the loop setting as follows:
   - lsw=0, lock the rotor of the motor
   - lsw=1, close the current loop
4. Set lsw to 1. Now the motor is running close to reference speed. Check the speed1.Speed in the Expressions window with the continuous refresh feature, whether or not the measured speed is around the speed reference.
5. Confirm these modules by trying different values of SpeedRef to test the speed.
6. Use an oscilloscope to view the electrical angle output, ElecTheta, from the QEP_MACRO module, and the emulated rotor angle, Out, from the RG_MACRO at the PWMDAC outputs with external low-pass filters.
7. Check that both qep1.ElecTheta and rg1.Out have a saw-tooth wave shape and have the same period. If the measured angle is in opposite direction, change the order of the motor cables connected to the inverter output (TB3 for HVDMC kit).
8. Qep1.ElecTheta should be slightly trailing rg1.out; if not, adjust the calibration angle.
9. Check in the Expressions window that qep1.IndexSyncFlag is set back to 0xF0 every time it resets to 0 by hand. Add the variable to the Expressions window if it is not already in the Expressions window.
10. Bring the system to a safe stop, as described at the end of build 1, by reducing the bus voltage, taking the controller out of real-time mode, and resetting. Once stopped, terminate the debug session.
11. The calibration angle of the encoder is detected in the code as detailed in the following steps. This is an optional procedure for sensorless FOC speed loop tuning. If the user prefers to use an encoder to tune the speed loop and apply the schemes given in level 5A, then the exact rotor position is not needed. However, if the user tunes the speed loop as in level 5C, then the exact rotor position information is needed, thus the calibration angle must be detected.

Use the following steps to verify and perform the calibration angle of the encoder.

1. Ensure EQep1Regs.QPOSCNT, EQep1Regs.QPOSILAT, Init_IFlag, qep1.CalibratedAngle, and lsw are displayed in the Expressions window.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
3. Compile, load, and run the program with real-time mode, then increase the voltage at the variac or DC power supply for the appropriate DC bus voltage.
4. The rotor should now be locked. Set lsw to 1 to spin the motor. When the first index signal is detected by QEP, the EQep1Regs.QPOSILAT register latches the angle offset between initial rotor position and the encoder index in the code. Later, EQep1Regs.QPOSILAT is set to the maximum of EQep1Regs.QPOSCNT as it latches the counter value for each index signal. In the code, qep1.CalibratedAngle keeps the initial offset value. This value can be recorded to initialize qep1.CalibratedAngle at the initialization section in Pfc_PM_Sensorless.c, or it can be detected in the code each time the motor restarts. The calibration angle might be different for different start-ups and can be formulated as follows:

Calibration Angle = Offset Angle ± n . Line Encoder

12.4 Level 4 Incremental Build



Level 4 verifies the rotor position and speed estimation performed by eSMO and SE macros

Figure 31. Level 4 Incremental System Build Block Diagram

Assuming the steps in the previous section complete successfully, this section verifies the estimated rotor position and speed estimation performed by the SMO_MODULE (enhanced sliding mode observer) and SE_MACRO modules, respectively.

1. Open Pfc_PM_Sensorless-Settings.h, and select the level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4).
2. Right-click on the project name, and click Rebuild Project.
3. When the build completes, click the Debug button, reset the CPU, restart, enable real-time mode, and run the program.
4. Set EnableFlag to 1 in the Expressions window. The IsrTicker variable continues increasing; confirm this by watching the variable in the Expressions window. While this occurs, the CPU computes the offset of the analog channels sensing phase-current feedback. After this is done, Isr1Ticker continues to count, indicating the main control ISR is running. This confirms that the system interrupt is working properly.

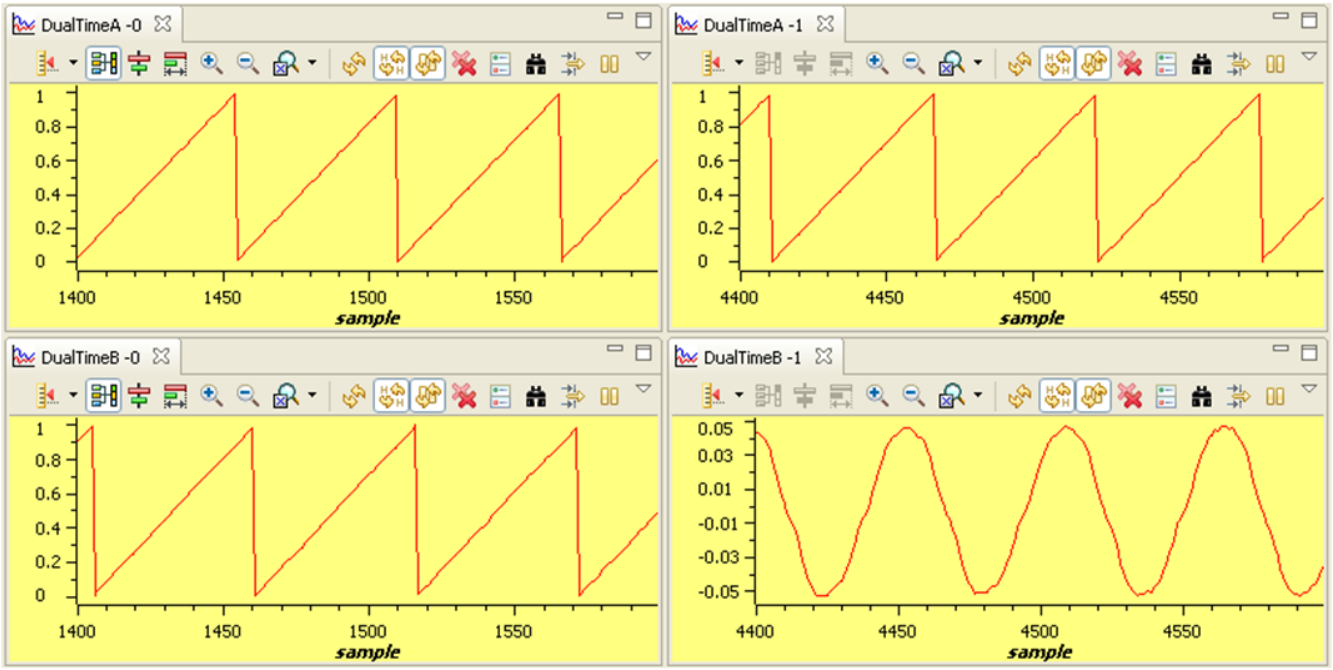
In the software, the key variables to be adjusted are:

- SpeedRef (Q24): for changing the rotor speed in per-unit
- IdRef (Q24): for changing the d-qxis voltage in per-unit
- IqRef (Q24): for changing the q-axis voltage in per-unit

The tuning of sliding-mode and low-pass filter gains (Kslide and Kslf, respectively) inside the rotor position estimator may be critical for low-speed operation. The key steps are:

1. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
2. Compile, load, and run the program with real-time mode, then increase the voltage at the variac or DC power supply to get the appropriate DC bus voltage.
3. Set lsw = 0 to align and lock the rotor of the motor.
4. Set lsw to 1 to get the motor running close to reference speed. Compare esmo1.Theta with rg1.Out at the test points TP6 and TP4, respectively. They points should be identical, with a small phase shift.
5. If esmo1.Theta does not give the saw tooth waveform, the Kslide and Kslf inside the sliding mode observer must be retuned.
6. Try different values of SpeedRef to confirm rotor position estimation.
7. Compare speed3.EstimatedSpeed with the reference speed in the Expressions windows with the continuous refresh feature, whether or not it is nearly the same.
8. Try different values of SpeedRef to confirm the open-loop speed estimator.
9. Bring the system to a safe stop as described at the end of build 1, by reducing the bus voltage, taking the controller out of real-time mode, and resetting.

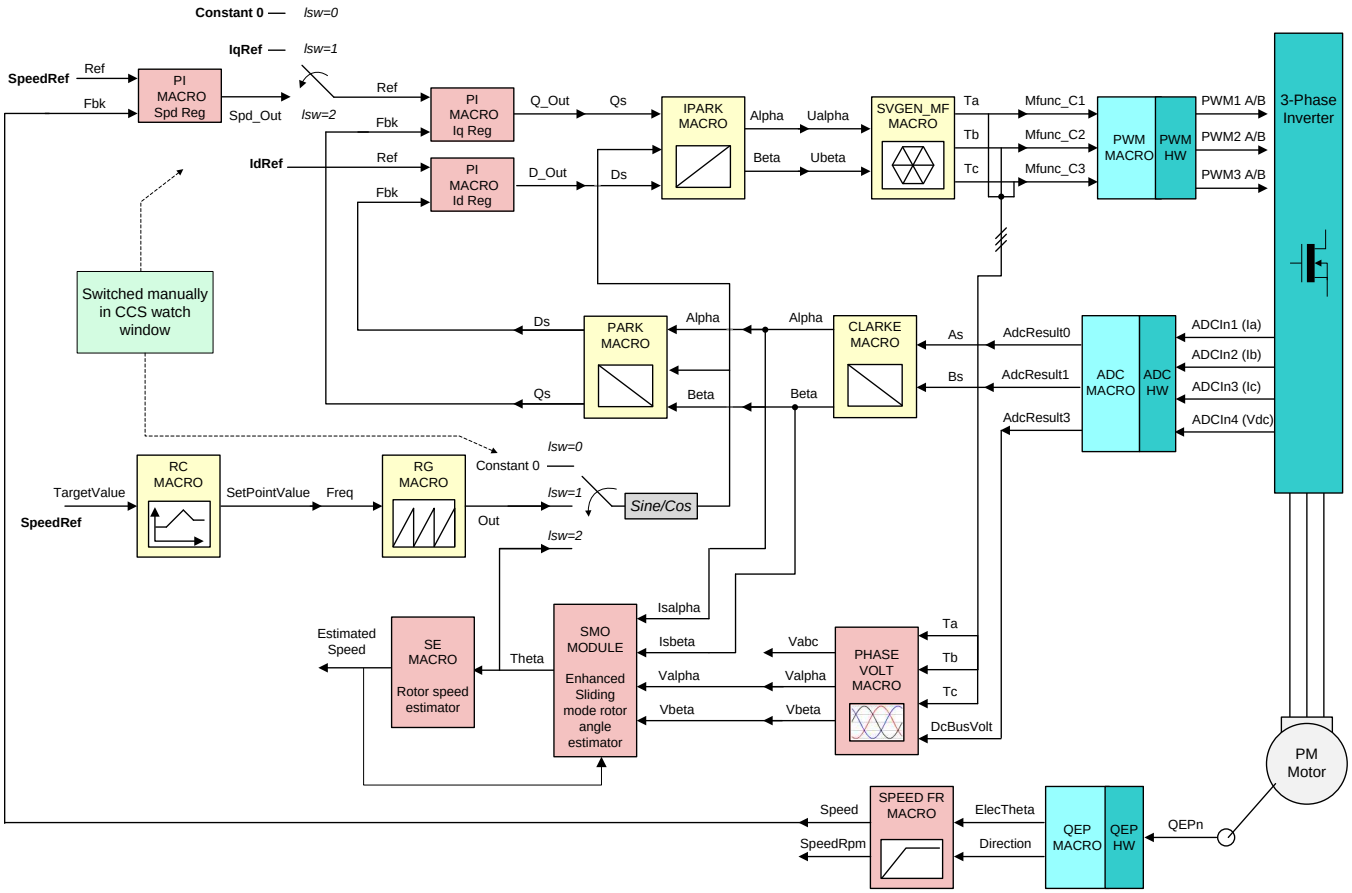
While running this build, the current waveforms in the CCS graphs should appear as shown in Figure 32.



dlog.trig_value = 100, deadband = 1.66 µsec, Vdcbus = 300 V

Figure 32. Measured Theta, Estimated Theta (SMO), rg1. Out and Phase A Current

12.5 Level 5 Incremental Build



Level 5 verifies the speed PI macro and speed loop. The Speed loop is closed using measured speed as feedback

Figure 33. Level 5 Incremental System Build Block Diagram

Assuming the steps in the previous section complete successfully, this section verifies the speed regulator performed by the PI module. The system speed loop is closed by using the measured speed from QEP as a feedback. Because the SMO MODULE is already verified, its angle output is used to decouple the motor currents to Id and Iq.

1. Open Pfc_PM_Sensorless-Settings.h, and select the level 5 incremental build option by setting the BUILDLEVEL to LEVEL5 (#define BUILDLEVEL LEVEL5).
2. Right-click on the project name, and click Rebuild Project.
3. When the build completes, click the Debug button, reset the CPU, restart, enable real-time mode, and run the program.
4. Set EnableFlag to 1 in the Expressions window. The IsrTicker variable continues increasing; confirm this by watching the variable in the Expressions window. While this occurs, the CPU computes the offset of the analog channels sensing phase-current feedback. After this is done, Isr1Ticker continues to count, indicating the main control ISR is running. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are:

- SpeedRef (Q24): for changing the rotor speed in per-unit
- IdRef (Q24): for changing the d-qxis voltage in per-unit

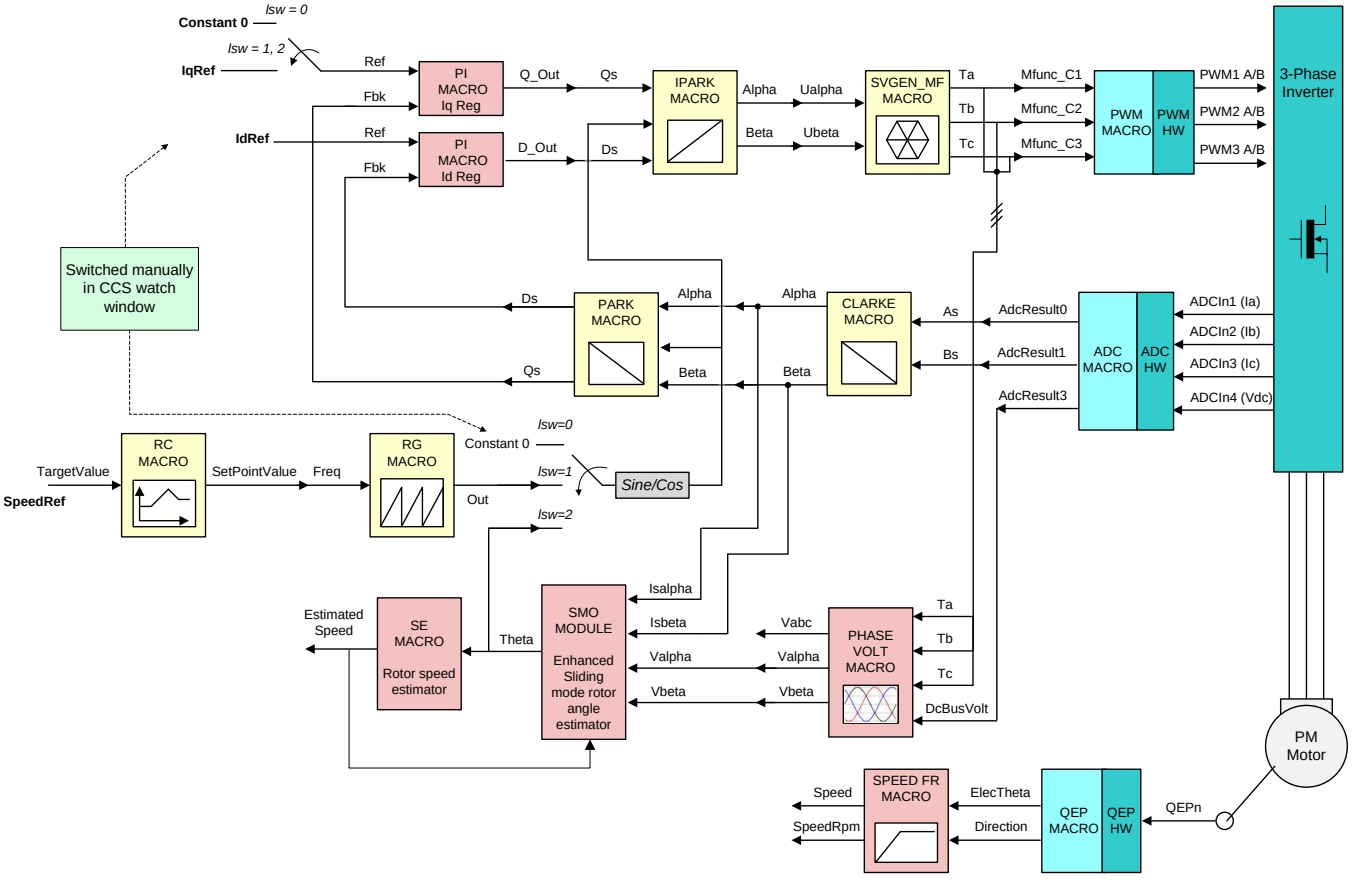
12.5.1 Level 5A

The speed loop is closed by using measured speed. The key steps are:

1. Compile, load, and run the program with real-time mode.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
3. Set lsw to 0 to align and lock the rotor of the motor.
4. Set lsw to 1. Gradually increase the voltage at the variac or DC power supply to get an appropriate DC bus voltage. The motor should now be running at the reference speed (0.3 pu).
5. Set lsw to 2 and close the speed loop.
6. Compare Speed with SpeedRef in the Expressions windows with the continuous refresh feature, whether or not it is nearly the same.
7. To confirm this speed PI module, try different values of SpeedRef.
8. For the speed PI controller, the proportional, integral, derivative, and integral correction gains may be retuned for good responses.
9. At very low-speed range, the performance of speed response relies heavily on the good rotor flux angle computed by the flux estimator.
10. Bring the system to a safe stop as described at the end of build 1, by reducing the bus voltage, taking the controller out of real-time mode, and resetting.

NOTE: IdRef must be set to zero at all times.

12.5.2 Level 5B (Alternative Method)



Level 5 verifies the estimated theta

Figure 34. Level 5B Incremental System Build Block Diagram

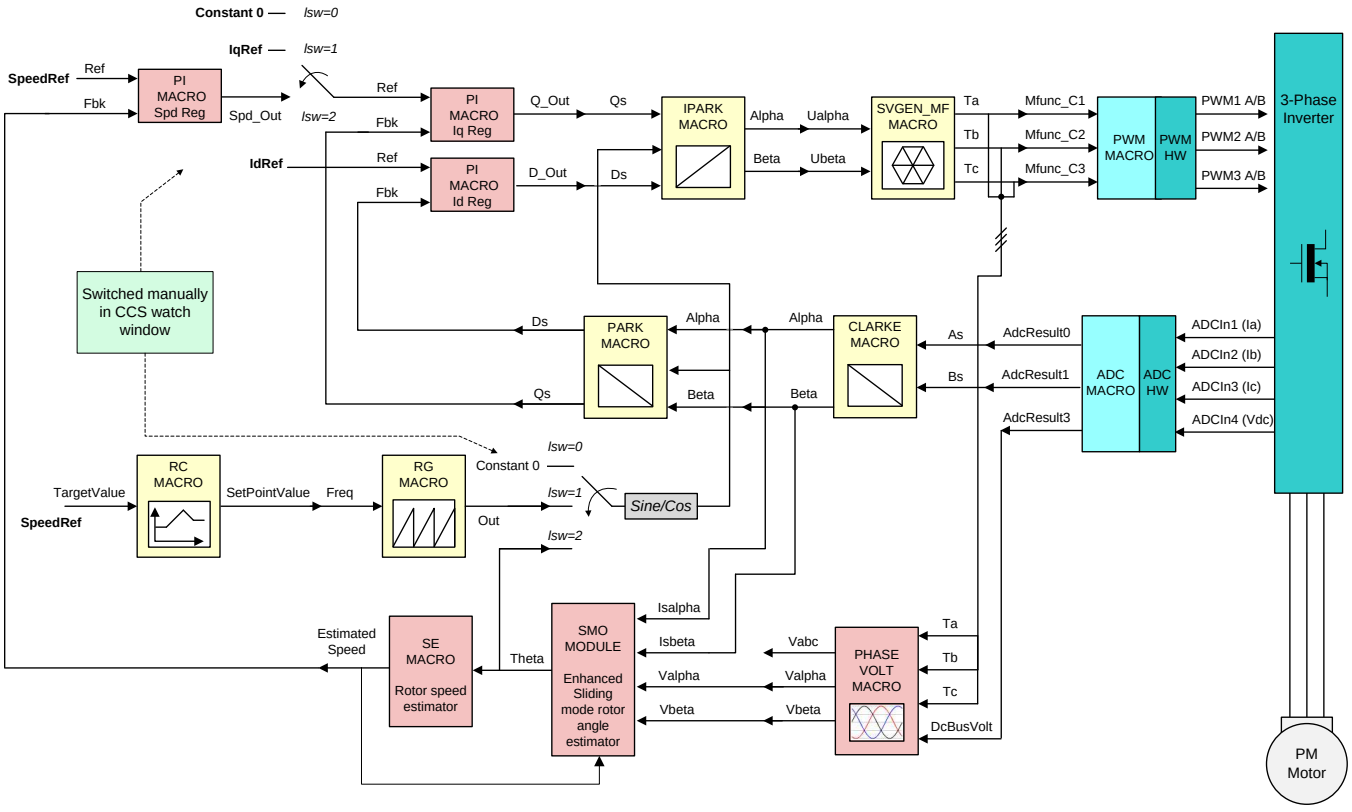
Tuning both speed PI and SMO at the same time may not be possible for some applications. Test the eSMO without speed PI in the loop by disconnecting the speed PI and Iq PI modules in the code, as shown in Figure 34, and apply constant Iqref as the reference for Iq PI. After tuning SMO (Kslide), the motor should spin smoothly, and the estimated angle should be clear sawtooth.

In this scheme the speed is not controlled, and a nonzero torque reference (Iqref) spins the motor very fast, unless loaded. Thus, keep the Iqref low initially, and load the motor using a brake or generator (or manually if the motor is small enough). If the motor speed is too low or the torque generated by the motor cannot handle the applied load, increase Iqref or reduce the amount of load. After tuning the SMO MODULE, add speed PI into the system, as shown in Figure 35, and tune the PI coefficients, if necessary. This method helps the user tune SMO and speed PI separately.

If the test is implemented on a custom inverter or a different motor is used, then:

- Check the parameters in Pfc_PM_Sensorless-Settings.h. Ensure that the base (pu) quantities are set to maximum measurable current and voltage, and the motor electrical parameters are correct.
- The DC bus voltage should be high enough so as not to saturate the PI outputs.
- Re-run the same experiment and keep tuning the SMO gains.

12.6 Level 6 Incremental Build



Level 6 verifies the complete system

Figure 35. Level 6 Incremental System Build Block Diagram

Assuming the steps in the previous section complete successfully, this section verifies the speed regulator performed by the PI module. The system speed loop is closed by using the estimated speed as a feedback.

1. Open Pfc_PM_Sensorless-Settings.h, and select the level 6 incremental build option by setting the BUILDLEVEL to LEVEL6 (#define BUILDLEVEL LEVEL6).
2. Right-click on the project name, and click Rebuild Project.

3. When the build completes, click the Debug button, reset the CPU, restart, enable real-time mode, and run the program.
4. Set EnableFlag to 1 in the Expressions window. The IsrTicker variable continues increasing; confirm this by watching the variable in the Expressions window. While this occurs, the CPU computes the offset of the analog channels sensing phase-current feedback. After this is done, Isr1Ticker continues to count, indicating the main control ISR is running. This confirms that the system interrupt is working properly.

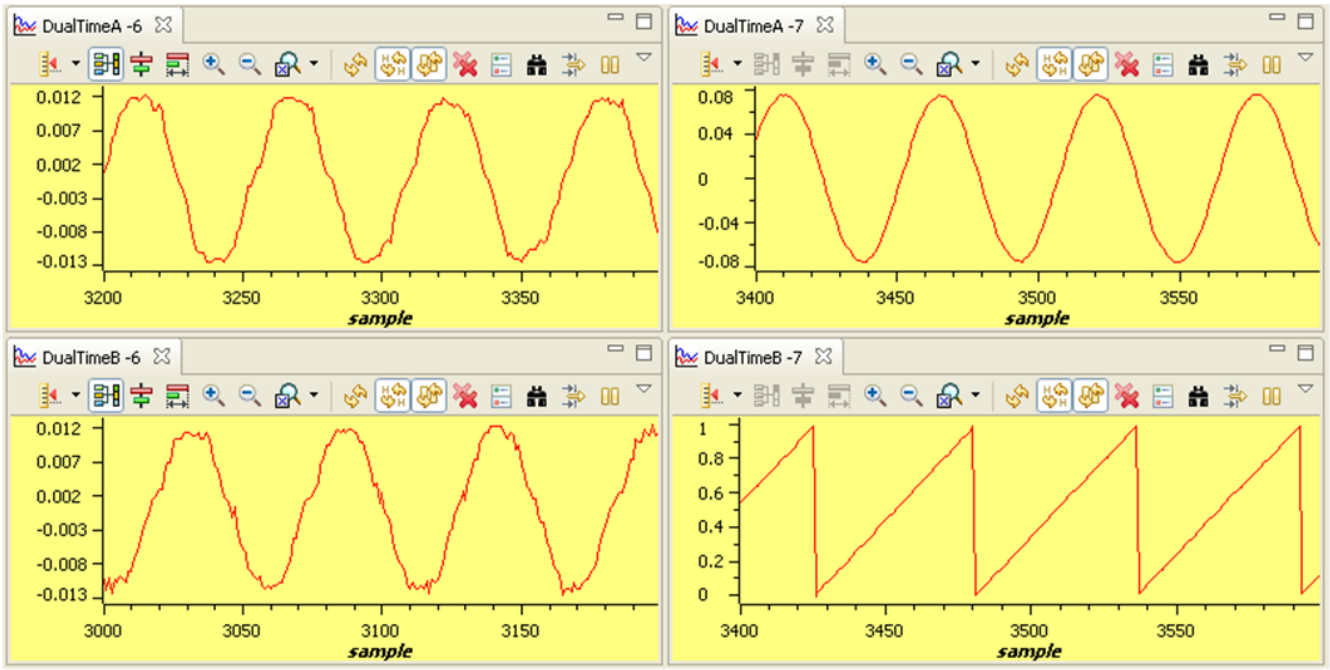
In the software, the key variables to be adjusted are:

- SpeedRef (Q24): for changing the rotor speed in per-unit
- IdRef (Q24): for changing the d-qxis voltage in per-unit

The speed loop is closed by using estimated speed. The key steps are:

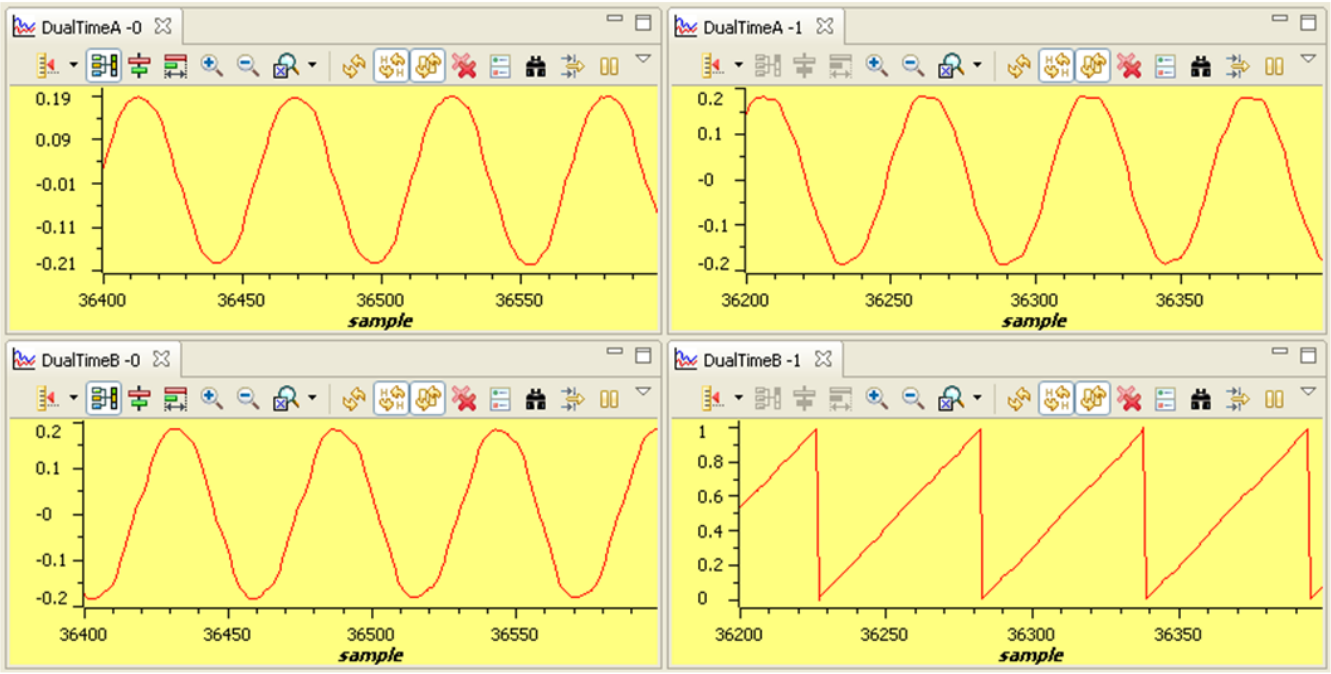
1. Compile, load, and run the program with real-time mode.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different) and Iqref to 0.1 pu.
3. Set lsw to 0 to align and lock the rotor of the motor.
4. Set lsw to 1. Gradually increase the voltage at the variac or DC power supply to get an appropriate DC bus voltage. The motor should now be running at this reference speed (0.3 pu).
5. Set lsw to 2 to close the speed loop. After a few tests, the user can determine the best time to close the speed loop, depending on the load-speed profile, then close the speed loop in the code. For most applications, the speed loop can be closed before the motor speed reaches SpeedRef.
6. Compare speed3.EstimatedSpeed with SpeedRef in the Expressions windows with the continuous refresh feature, whether or not it is nearly the same.
7. To confirm this speed PI module, try different values of SpeedRef.
8. For the speed PI controller, the proportional, integral, derivative, and integral correction gains may be retuned for good responses.
9. At very low-speed range, the performance of speed response relies heavily on the good rotor flux angle, computed by the flux estimator.
10. Bring the system to a safe stop as described at the end of build 1, by reducing the bus voltage, taking the controller out of real-time mode, and resetting.

While running this build, the current waveforms in the CCS graphs should appear as shown in Figure 36 and Figure 37.



dlog.trig_value = 100, deadband = 1.66 µsec, Vdcbus = 300 V, pi_spd.Kp = 1.0

Figure 36. Waveforms of Phase A and B Currents, Calculated Phase A Voltage, and Estimated Theta by SMO Under No-load and 0.3-pu Speed



dlog.trig_value = 100, deadband = 1.66 µsec, Vdcbus = 300 V, pi_spd.Kp = 1.0

Figure 37. Waveforms of Phase A and B Currents, Calculated Phase A Voltage, and Estimated Theta by SMO Under 0.33-pu Load and 0.5-pu Speed

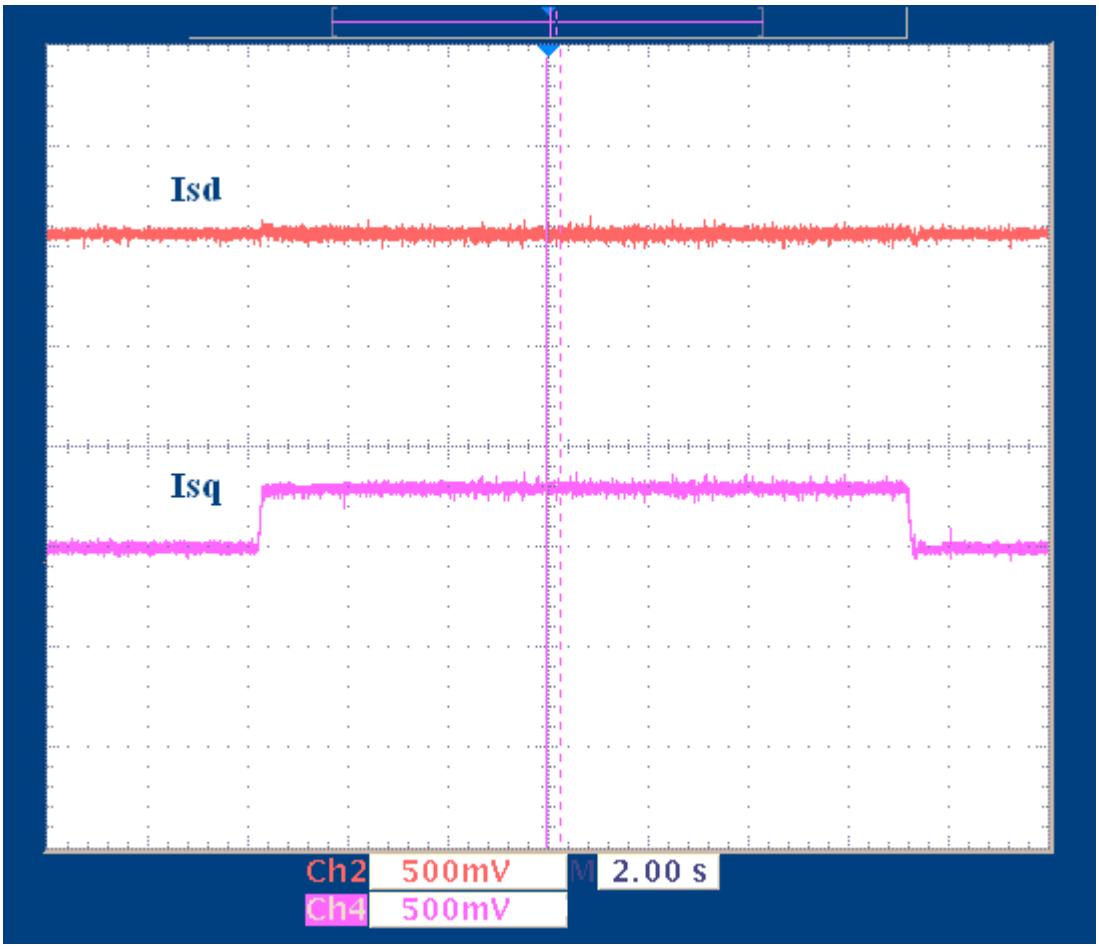


Figure 38. Flux and Torque Components of the Stator Current in the Synchronous Reference Frame Under 0.33-pu Step-Load and 0.5-pu Speed Monitored from PWMDAC Output

13 Integrated Software Test Strategy

If the PFC and motor control blocks are individually verified, they can be connected together and tested.

1. Set up the hardware configuration as shown in Figure 15. Do not turn on the power switch yet.
2. Ensure that isolation requirements are met for the safety of the test setup, and for any instruments that may be connected to the test.
3. Apply a control power supply to the board through [M6]-JP1, and turn on the switch [M6]-SW1.
4. Because both the PFC and motor control software are already verified, the user can set the BUILD_LEVEL of PFC and MOTOR at their highest LEVEL, which is 2 and 6, respectively.
5. Ensure the PFC output voltage command is set to a safe rating of the motor. The default setting is for 300 V, which will suit a 110-V/60-Hz supply. For 220-V/50-Hz mains, a higher voltage command (>380 V) may be required for a better line current. The motor must be rated for the bus voltage. If a voltmeter is available, connect it across the DC bus to track the bus voltage.
6. Compile, load, and run the program with real-time mode.
7. Set the AC voltage to 120 V and turn it on to power the board
8. Preferably, operate the PFC with minimal load. Before turning on the PFC, turn on the motor by setting Isw = 0, which forces some current through the motor for alignment.

9. Turn the PFC on by setting start_flag = 1 in the Expressions window. Watch the DC bus voltage ramp gently to the commanded level and hold there.
10. Watch the line current waveform improve. At light loads and low DC bus voltage conditions, the power factor improvement may be moderate. After this, the motor control section may be evaluated as in BUILD_LEVEL 6.
11. After the evaluation completes, turn off the AC power and wait until the motor stops, or until there is no DC bus voltage.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265