

# Sensorless Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors

Bilal Akin and Manish Bhardwaj

## ABSTRACT

This application report presents a solution to control a permanent magnet synchronous motor (PMSM) using the TMS320F2803x microcontrollers. TMS320F2803x devices are part of the family of C2000 microcontrollers, which enables the cost-effective design of intelligent controllers for three phase motors by reducing system components and increasing efficiency. With these devices, it is possible to realize far more precise digital vector control algorithms like the field orientated control (FOC). This algorithm's implementation is discussed in this document. The FOC algorithm maintains efficiency in a wide range of speeds and takes into consideration torque changes with transient phases by processing a dynamic model of the motor. Among the solutions proposed are ways to eliminate the phase current sensors and use an observer for speed sensorless control.

This application report covers the following:

- A theoretical background on field oriented motor control principle
- Incremental build levels based on modular software blocks
- Experimental results

## Contents

1	Introduction .....	2
2	Permanent Magnet Motors .....	3
3	Synchronous Motor Operation .....	3
4	Field Oriented Control (FOC) .....	4
5	The Basic Scheme for the FOC .....	8
6	Benefits of 32-Bit C2000™ Controllers for Digital Motor Control (DMC) .....	10
7	TI Literature and Digital Motor Control (DMC) Library .....	11
8	Hardware Configuration (HVDMC R1.1 Kit) .....	15
9	Incremental System Build .....	18
10	References .....	42

## List of Figures

1	A Three-Phase Synchronous Motor With a One Permanent Magnet Pair Pole Rotor.....	3
2	Interaction Between the Rotating Stator Flux, and the Rotor Flux Produces a Torque That Causes the Motor to Rotate.....	4
3	Separated Excitation DC Motor Model (Flux and Torque are Independently Controlled and the Current Through the Rotor Windings Determines How Much Torque is Produced) .....	4
4	Stator Current Space Vector and Its Component in (a,b,c) .....	6
5	Stator Current Space Vector and Its Components in the Stationary Reference Frame .....	6
6	Stator Current Space Vector and Its Component in ( $\alpha, \beta$ ) and in the d,q Rotating Reference Frame .....	7
7	Basic Scheme of FOC for AC Motor .....	8
8	Current, Voltage and Rotor Flux Space Vectors in the d,q Rotating Reference Frame and Their Relationship With a,b,c and ( $\alpha, \beta$ ) Stationary Reference Frame .....	9
9	Overall Block Diagram of Sensorless Field Oriented Control of PMSM.....	10

C2000, Code Composer Studio are trademarks of Texas Instruments.  
 All other trademarks are the property of their respective owners.

10	A 3-ph Induction Motor Drive Implementation .....	13
11	System Software Flowchart .....	14
12	Using AC Power to Generate DC Bus Power .....	16
13	Using External DC Power Supply to Generate DC-Bus for the Inverter .....	17
14	Watch Window Variables .....	18
15	Output of SVGEN, Ta, Tb, Tc and Tb-Tc Waveforms .....	19
16	DAC-1-4 Outputs Showing Ta, Tb, Tc and Tb-Tc Waveforms .....	20
17	Level 1 - Incremental System Build Block Diagram.....	21
18	Calculated Phase A and B Voltages by volt1 Module, rg1.Out and svgen_dq1.Ta .....	22
19	The Waveforms of Svgen_dq1.Ta, rg1.Out, and Phase A and B Currents.....	23
20	Level 2 - Incremental System Build Block Diagram.....	25
21	rg1.Out, Measured theta and Phase A and B Current Waveforms .....	27
22	Level 3 - Incremental System Build Block Diagram.....	29
23	Measured theta, Estimated theta (SMO), rg1.Out and Phase A Current .....	31
24	Level 4 - Incremental System Build Block Diagram.....	32
25	Level 5A - Incremental System Build Block Diagram.....	35
26	Level 5B - Incremental System Build Block Diagram.....	37
27	Waveforms of Phase A and B Currents, Calculated Phase A Voltage, and Estimated theta by SMO Under No-Load and 0.3pu Speed .....	39
28	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.....	39
29	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 PWM DAC Output .....	40
30	Level 6 - Incremental System Build Block Diagram.....	41

#### List of Tables

1	Tested Modules in Each Incremental System Build .....	18
---	---	----

## 1 Introduction

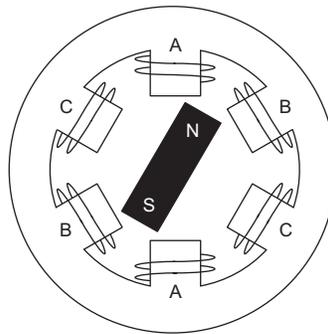
A brushless PMSM has a wound stator, a permanent magnet rotor assembly, and internal or external devices to sense rotor position. The sensing devices provide position feedback for adjusting frequency and amplitude of stator voltage reference properly to maintain rotation of the magnet assembly. The combination of an inner permanent magnet rotor and outer windings offers the advantages of low rotor inertia, efficient heat dissipation, and reduction of the motor size. Moreover, the elimination of brushes reduces noise, EMI generation and suppresses the need of brushes maintenance.

This document presents a solution to control a permanent magnet synchronous motor using the TMS320F2803x. This new family of DSPs enables cost-effective design of intelligent controllers for brushless motors which can fulfill enhanced operations, consisting of fewer system components, lower system cost and increased performances. The control method presented relies on the FOC. This algorithm maintains efficiency in a wide range of speeds and takes torque changes with transient phases into consideration by controlling the flux directly from the rotor coordinates. This application report presents the implementation of a control for the sinusoidal PMSM motor. The sinusoidal voltage waveform applied to this motor is created by using the space vector modulation technique. The minimum amount of torque ripple appears when driving this sinusoidal BEMF motor with sinusoidal currents.

## 2 Permanent Magnet Motors

There are primarily two types of three-phase permanent magnet synchronous motors: 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 many commutator segments. The drawbacks of this type of structure are maintenance needs and lower reliability.

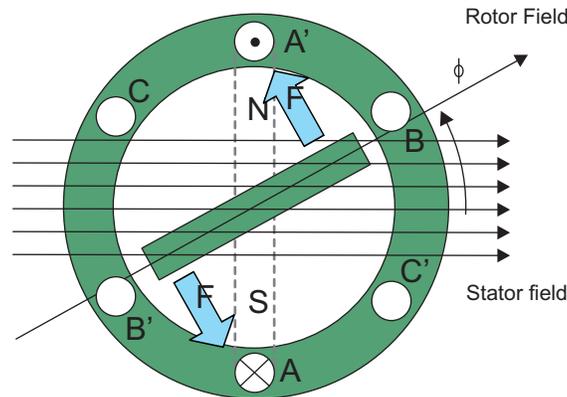
Replacing the common rotor field windings and pole structure with permanent magnets puts the motor into the category of brushless motors. It is possible to build brushless permanent magnet motors with any even number of magnet poles. The use of magnets enables an efficient use of the radial space and replaces the rotor windings, therefore, suppressing the rotor copper losses. Advanced magnet materials permits a considerable reduction in motor dimensions while maintaining a very high power density.



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

## 3 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. When energized, the stator windings create a rotating electromagnetic field. To control the rotating magnetic field, it is necessary to control the stator currents.
- The actual structure of the rotor varies depending on the power range and rated speed of the machine. Permanent magnets are suitable for synchronous machines ranging up-to a few Kilowatts. For higher power ratings, the rotor usually consists of windings in which a dc current circulates. The mechanical structure of the rotor is designed for the number of desired poles, and the desired flux gradients.
- The interaction between the stator and rotor fluxes produces a torque. Since the stator is firmly mounted to the frame, and the rotor is free to rotate, the rotor will rotate, producing a useful mechanical output.
- The angle between the rotor magnetic field and the stator field must be carefully controlled to produce maximum torque and achieve high electro-mechanical conversion efficiency. For this purpose a fine tuning is needed after closing the speed loop using the sensorless algorithm in order 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 will experience rapidly alternating positive and negative torque. This results in less than optimal torque production, and excessive mechanical vibration, noise, and mechanical stresses on the machine parts. In addition, if the rotor inertia prevents the rotor from being able to respond to these oscillations, the rotor stops rotating at the synchronous frequency, and responds to the average torque as seen by the stationary rotor: zero. This means that the machine experiences a phenomenon known as 'pull-out'. This is also the reason why the synchronous machine is not self starting.
- The angle between the rotor field and the stator field must be equal to  $90^\circ$  to obtain the highest mutual torque production. This synchronization requires knowing the rotor position in order to generate the right stator field.
- The stator magnetic field can be made to have any direction and magnitude by combining the contribution of the different stator phases to produce the resulting stator flux.



**Figure 2. Interaction Between the Rotating Stator Flux, and the Rotor Flux Produces a Torque That Causes the Motor to Rotate**

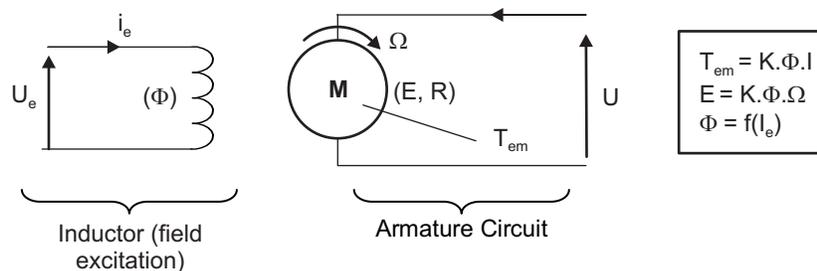
## 4 Field Oriented Control (FOC)

### 4.1 Introduction

In order to achieve better dynamic performance, a more complex control scheme needs to be applied to control the PM motor. With the mathematical processing power offered by the microcontrollers, advanced control strategies can be implemented, which uses mathematical transformations in order to decouple the torque generation and the magnetization functions in the PM motors. Such decoupled torque and magnetization control is commonly called rotor flux oriented control, or simply FOC.

### 4.2 The Main Philosophy Behind the FOC

In order to understand the spirit of the FOC technique, start with an overview of the separately excited direct current (DC) motor. In this type of motor, the excitation for the stator and rotor is independently controlled. The electrical study of the DC motor shows that the produced torque and the flux can be independently tuned. The strength of the field excitation (the magnitude of the field excitation current) sets the value of the flux. The current through the rotor windings determines how much torque is produced. The commutator on the rotor plays an interesting part in the torque production. The commutator is in contact with the brushes, and the mechanical construction is designed to switch into the circuit the windings that are mechanically aligned to produce the maximum torque. This arrangement then means that the torque production of the machine is fairly near optimal all the time. The key point here is that the windings are managed to keep the flux produced by the rotor windings orthogonal to the stator field.



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

AC machines do not have the same key features as the DC motor. In both cases, there is only one source that can be controlled, which is 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. Obviously, as opposed to the DC motor, the flux and torque depend on each other.

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

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

$$T_{em} = \vec{B}_{stator} \times \vec{B}_{rotor}$$

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

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

### 4.3 Technical Background

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

$$m \propto \Psi_R i_{sq}$$

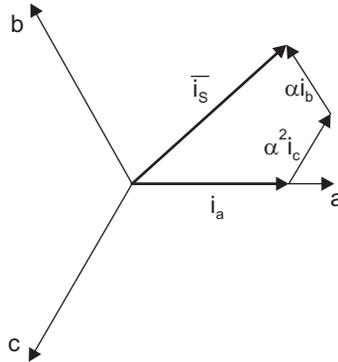
By maintaining the amplitude of the rotor flux ( $\phi_R$ ) at a fixed value, you have a linear relationship between the torque and torque component ( $i_{sq}$ ). You can then control the torque by controlling the torque component of stator current vector.

### 4.4 Space Vector Definition and Projection

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

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

where,  $\alpha = e^{j\frac{2}{3}\Pi}$  and  $\alpha^2 = e^{j\frac{4}{3}\Pi}$  represent the spatial operators. Figure 4 shows the stator current complex space vector.



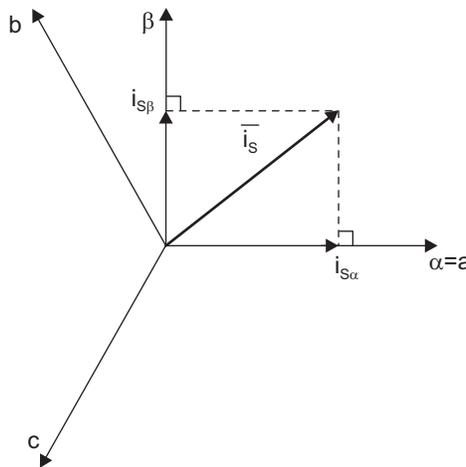
**Figure 4. Stator Current Space Vector and Its Component in (a,b,c)**

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

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

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

The space vector can be reported in another reference frame with only two orthogonal axis called (α, β). Assuming that axis a and axis α are in the same direction, see Figure 5.



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

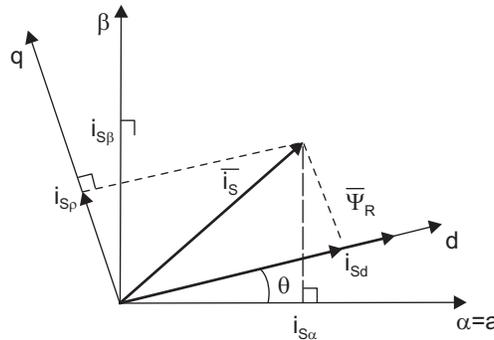
The projection that modifies the three phase system into the (α, β) two dimension orthogonal system is presented below:

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

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

#### 4.6 The $(\alpha, \beta) \rightarrow (d,q)$ Projection (Park Transformation)

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



**Figure 6. Stator Current Space Vector and Its Component in  $(\alpha, \beta)$  and in the d,q Rotating Reference Frame**

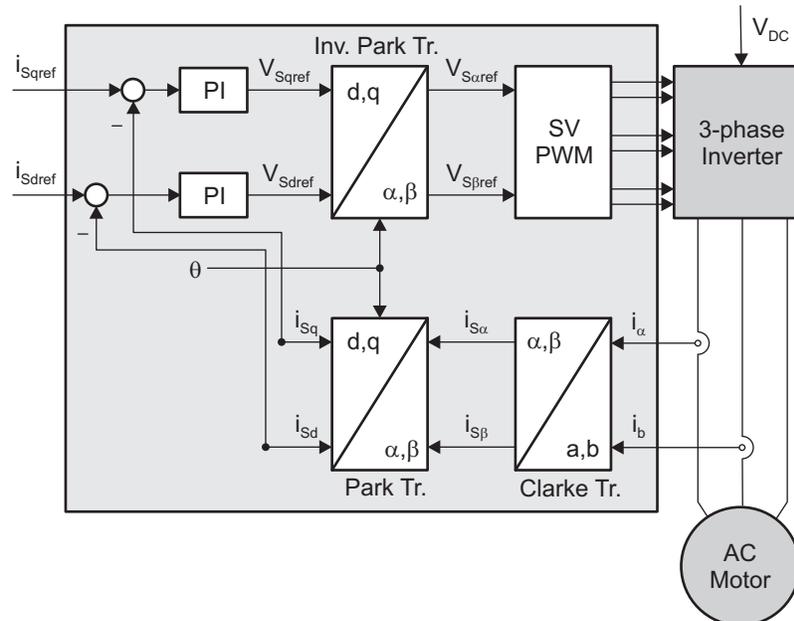
where,  $\theta$  is the rotor flux position. The flux and torque components of the current vector are determined by the following equations:

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

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

## 5 The Basic Scheme for the FOC

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

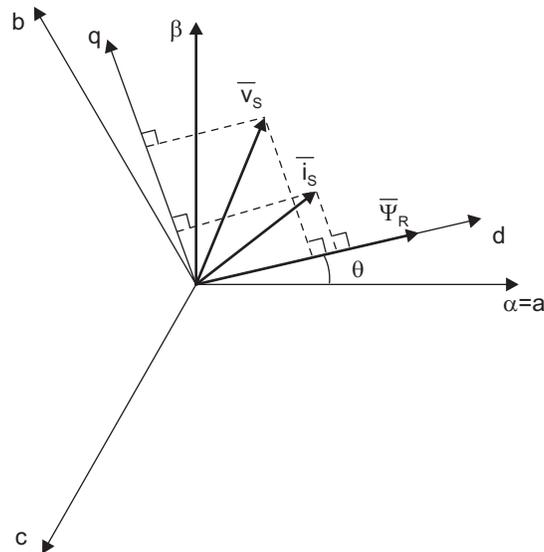


**Figure 7. Basic Scheme of FOC for AC Motor**

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 provide the current in the  $d,q$  rotating reference frame. The  $i_{sd}$  and  $i_{sq}$  components are compared to the references  $i_{sdrref}$  (the flux reference) and  $i_{sqref}$  (the torque reference). At this point, this control structure shows an interesting advantage: it can be used to control either synchronous or HVPM machines by simply changing the flux reference and obtaining rotor flux position. As in synchronous permanent magnet a motor, the rotor flux is fixed determined by the magnets; there is no need to create one. Hence, when controlling a PMSM,  $i_{sdrref}$  should be set to zero. As HVPM motors need a rotor flux creation in order 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 you use a speed FOC. The outputs of the current regulators are  $V_{sdrref}$  and  $V_{sqref}$ ; they are applied to the inverse Park transformation. The outputs of this projection are  $V_{saref}$  and  $V_{sbrref}$ , which are the components of the stator vector voltage in the  $(\alpha, \beta)$  stationary orthogonal reference frame. These are the inputs of the space vector PWM. The outputs of this block are the signals that drive the inverter. Note that both Park and inverse Park transformations need the rotor flux position. Obtaining this rotor flux position depends on the AC machine type (synchronous or asynchronous machine). The rotor flux position considerations are discussed in [Section 5.1](#).

### 5.1 Rotor Flux Position

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



**Figure 8. Current, Voltage and Rotor Flux Space Vectors in the  $d,q$  Rotating Reference Frame and Their Relationship With  $a,b,c$  and  $(\alpha, \beta)$  Stationary Reference Frame**

The measure of the rotor flux position is different if you consider synchronous or asynchronous motors:

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

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



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

- Enables control of multi-variable and complex systems using modern intelligent methods such as neural networks and fuzzy logic.
- Performs adaptive control. 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, online parameter estimation update.
- 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.

## 7 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 is separately documented with source code, use, and technical theory. For the source codes and explanations of macro blocks, install controlSUITE from [www.ti.com/controlsuite](http://www.ti.com/controlsuite) and choose the HV/MotorKit installation.

- C:\TI\controlSUITE\libs\app\_libs\motor\_control\math\_blocks\v4.0
- C:\TI\controlSUITE\libs\app\_libs\motor\_control\drivers\mf2803x\v2.0

These modules allow you to quickly build or customize your own systems. The library supports the three motor types: ACI, BLDC, PMSM, and comprises both peripheral dependent (software drivers) and target dependent modules.

The DMC Library components have been used by TI to provide system examples. All DMC Library variables are defined and inter-connected at initialization. At run-time, the macro functions are called in order. Each system is built using an incremental build approach, which allows sections of the code to be built at different times 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 so many different variables can affect the system and many different motor parameters need to be tuned.

---

**NOTE:** TI DMC modules are written in the form of macros for optimization purposes. For more details, see *Optimizing Digital Motor Control (DMC) Libraries (SPRAAK2)*. The macros are defined in the header files. You 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 [Example 1](#), which means that the code continues in the next line. Any character including invisible ones like a “space” or “tab” after the backslash will cause compilation error. Therefore, make sure that the backslash is the last character in the line. In terms of code development, the macros are almost identical to C function, and that you can easily convert the macro definition to a C functions.

---

### Example 1. A Typical DMC Macro Definition

```
#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);
```

## 7.1 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 TMS320F2803x-based controllers on Code Composer Studio™ software. The framework uses the following modules: <sup>(1)</sup>:

<sup>(1)</sup> Please refer to pdf documents in the motor control folder explaining the details and theoretical background of each macro.

Macro Names	Explanation
CLARKE	Clarke Transformation
PARK and IPARK	Park and Inverse Park Transformation
PI	PI Regulators
RC	Ramp Controller (slew rate limiter)
RG	Ramp and Sawtooth Generator
QEP and 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 Transformation)
PHASEVOLT	Phase Voltage Calculator
PWM and PWMDAC	PWM and PWMDAC Drives

In this system, the sensorless FOC of the PMSM is experimented with and will explore the performance of speed control. The PM motor is driven by a conventional voltage-source inverter. The TMS320x2803x control card is used to generate three PWM signals. The motor is driven by an integrated power module by means of space vector PWM technique. Two phase currents of PM motor ( $i_a$  and  $i_b$ ) are measured from the inverter and sent to the TMS320x2803x via two ADCs. In addition, the DC-bus voltage in the inverter is measured and sent to the TMS320x2803x via an ADC. This DC-bus voltage is necessary to calculate the three phase voltages when the switching functions are known.

The HVPM\_Sensorless project has the following properties:

C Framework		
System Name	Program Memory Usage 2803x	Data Memory Usage 2803x <sup>(1)</sup>
HVPM_Sensorless	4740 words <sup>(2)</sup>	1352 words

<sup>(1)</sup> Excluding the stack size

<sup>(2)</sup> Excluding “IQmath” Look-up Tables

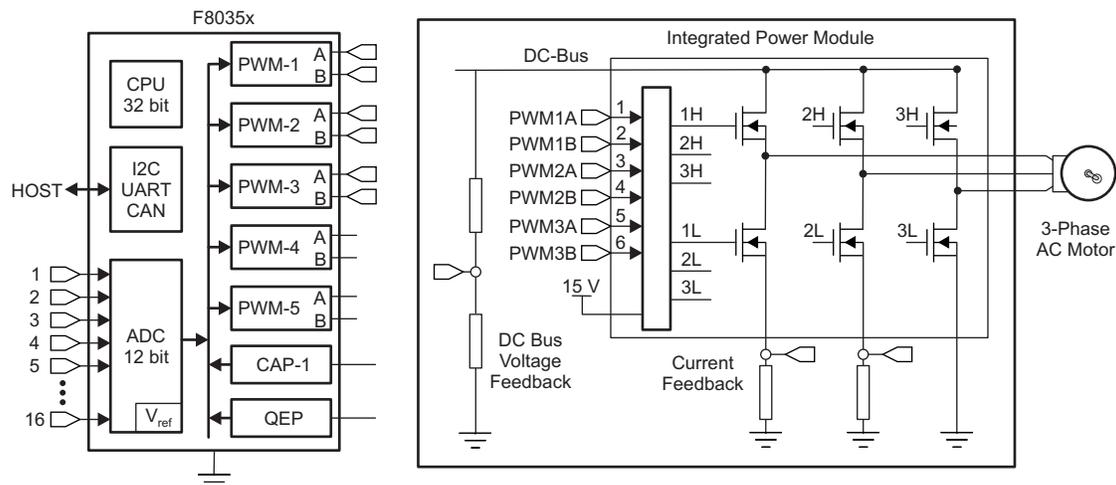
CPU Utilization	
Total Number of Cycles	943 <sup>(1)</sup>
CPU Utilization @ 60 Mhz	15.7%
CPU Utilization @ 40 Mhz	27.3%

<sup>(1)</sup> At 10 kHz ISR frequency. Debug macros excluded (in other words, PWMDAC, Datalog and RG). IQSin and Cos tables used.

**System Features**

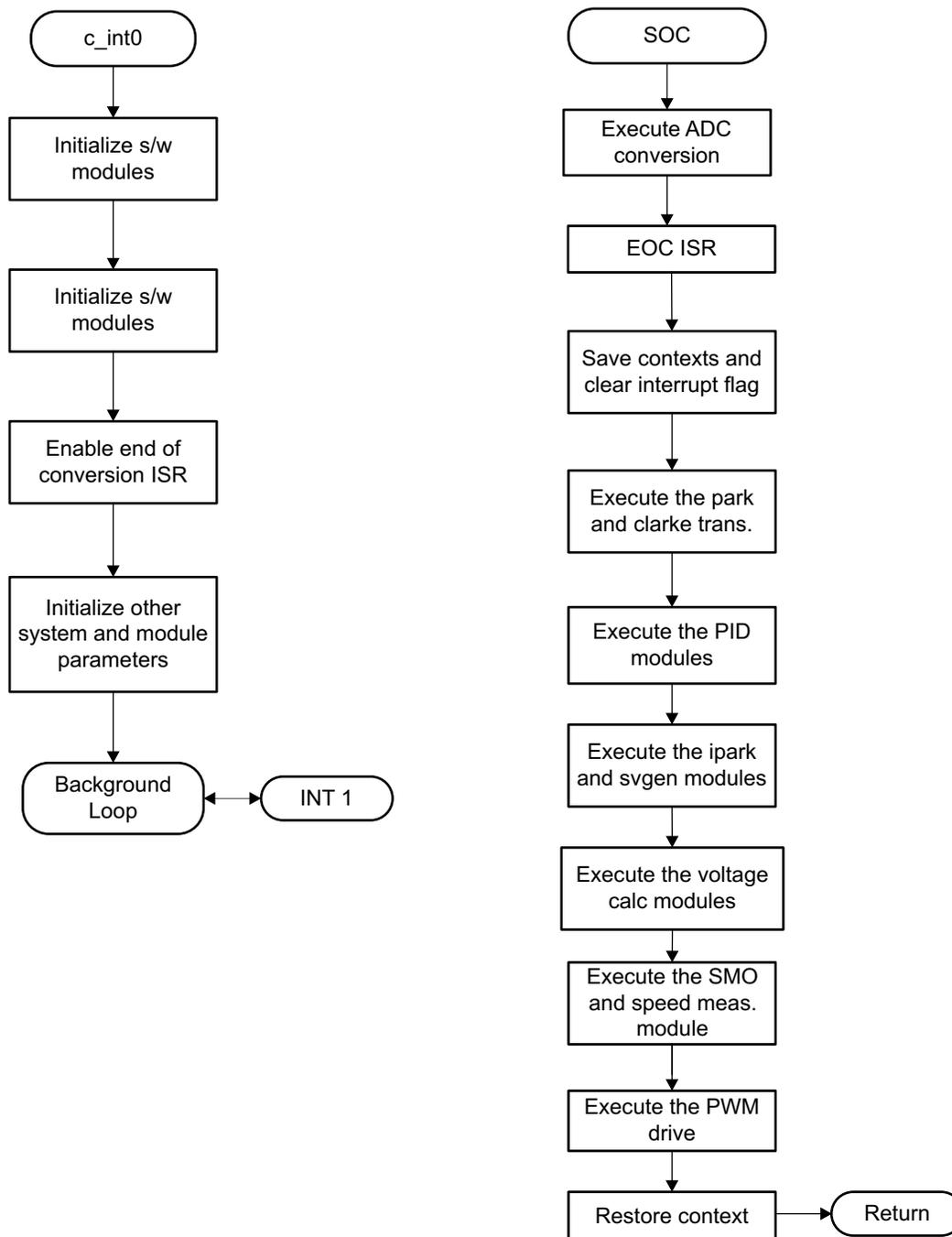
Development and Emulation	Code Composer Studio V4.0 (or above) with real-time debugging
Target Controller	TMS320F2803x
PWM Frequency	10 kHz PWM (Default), 60 kHz PWM DAC
PWM Mode	Symmetrical with a programmable dead band
Interrupts	ADC, end of conversion – Implements 10 kHz ISR execution rate
Peripherals Used	PWM 1, 2, 3 for motor control PWM 6A, 6B, 7A and 7B for DAC outputs QEP1 A, B, I or CAP1 ADC A7 for DC Bus voltage sensing, A1 and B1 for phase current sensing

The overall system implementing a 3-ph HVPM motor control is depicted in [Figure 10](#). The HVPM motor is driven by the conventional voltage-source inverter. The TMS320F2803x is being used to generate 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 they are sent to the TMS320F2803x via two ADCs. In addition, the DC-bus voltage in the inverter is measured and sent to the TMS320F2803x via an ADC as well. This DC-bus voltage is necessary in order to calculate three phase voltages of the HVPM motor when the switching functions are known.



**Figure 10. A 3-ph Induction Motor Drive Implementation**

The software flow is described in the [Figure 11](#).



**Figure 11. System Software Flowchart**

## 8 Hardware Configuration (HVDMC R1.1 Kit)

For an overview of the kit's hardware and steps on how to setup this kit, see the *HVMotorCtrl+PFC How to Run Guide* located at: [www.ti.com/controlsuite](http://www.ti.com/controlsuite) and choose the HVMotorKit installation.

Some of the hardware setup instructions are listed below for quick reference.

1. Open the lid of the HV kit.
2. Install the Jumpers [Main]-J3, J4 and J5, J9 for 3.3 V, 5 V and 15 V power rails and JTAG reset line.
3. Unpack the DIMM style controlCARD and place it in the connector slot of [Main]-J1. Push down vertically using even pressure from both ends of the card until the clips snap and lock. To remove the card, simply spread open the retaining clip with your thumbs.
4. Connect a USB cable to the connector [M3]-JP1. This enables an isolated JTAG emulation to the C2000 device. [M3]-LD1 should turn on. Make sure [M3]-J5 is not populated. If the included Code Composer Studio is installed, the drivers for the onboard JTAG emulation will automatically be installed. If a windows installation window appears, try to automatically install drivers from those already on your computer. The emulation drivers are found at <http://www.ftdichip.com/Drivers/D2XX.htm>. The correct driver is the one listed to support the FT2232.
5. If a third party JTAG emulator is used, connect the JTAG header to [M3]-J2 and additionally the [M3]-J5 needs to be populated to put the onboard JTAG chip in reset.
6. Ensure that [M6]-SW1 is in the "Off" position. Connect the 15 V DC power supply to [M6]-JP1.
7. Turn on [M6]-SW1. Now [M6]-LD1 should turn on. Notice that the control card LED lights up as well indicating that the control card is receiving power from the board.
8. Note that the motor should be connected to the [M5]-TB3 terminals after you finish with the first incremental build step.
9. Note the DC Bus power should only be applied during incremental build levels when instructed to do so. The two options to get DC Bus power are discussed below:
  - Set the power supply output to zero and connect [Main]-BS5 and BS6 to the DC power supply and ground, respectively, to use DC power supply.
  - Connect [Main]-BS1 and BS5 to each other using the banana plug cord to use AC Mains power. Now, connect one end of the AC power cord to [Main]-P1. The other end needs to be connected to the output of a variac. Make sure that the variac output is set to zero and it is connected to the wall supply through an isolator.

---

**NOTE:** Since the motor is rated at 200 V, the motor can only run at a certain speed and torque range properly without saturating the PID regulators in the control loop, when the DC bus is fed from 110 V AC entry. As an option, you can run the PFC on the HV DMC drive platform as boost converter to increase the DC bus voltage level or directly connect a DC power supply.

---

For reference, Figure 12 and Figure 13 show the jumper and connectors that need to be connected for this lab.

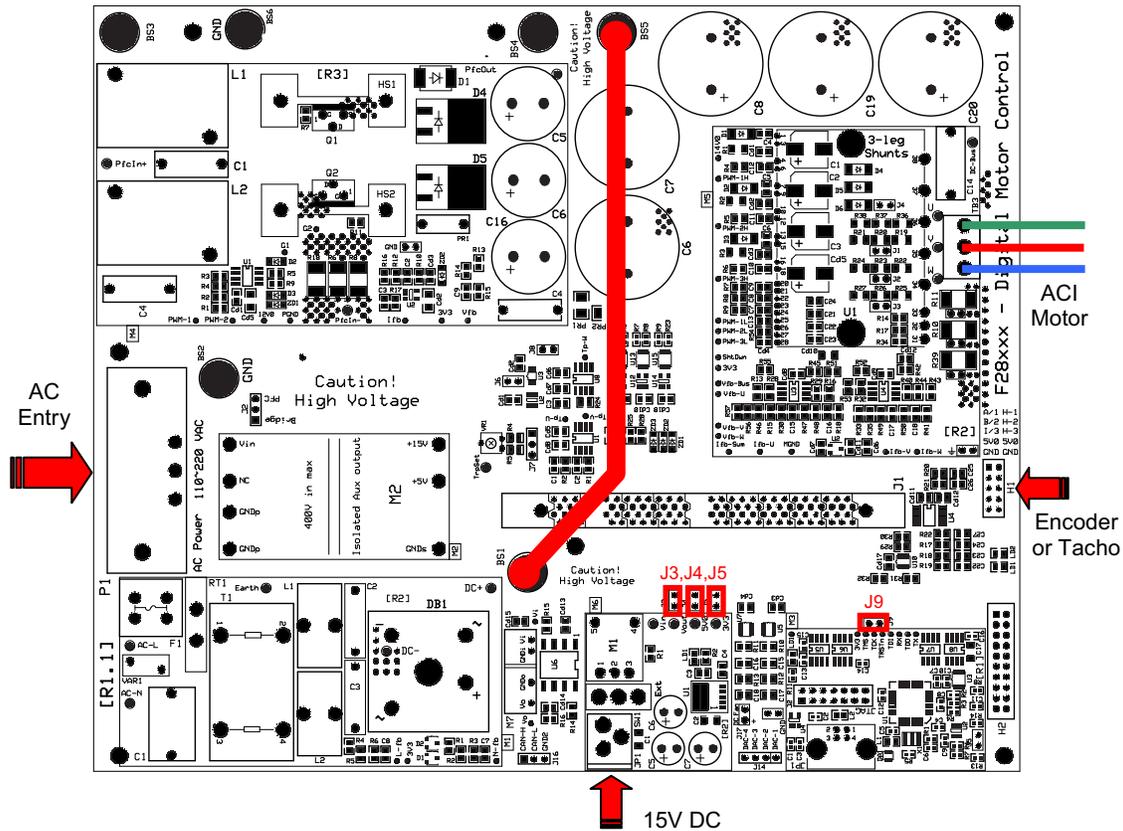


Figure 12. Using AC Power to Generate DC Bus Power

**CAUTION**

The inverter bus capacitors remain charged for a long time after the high power line supply is switched off or disconnected. Proceed with caution!

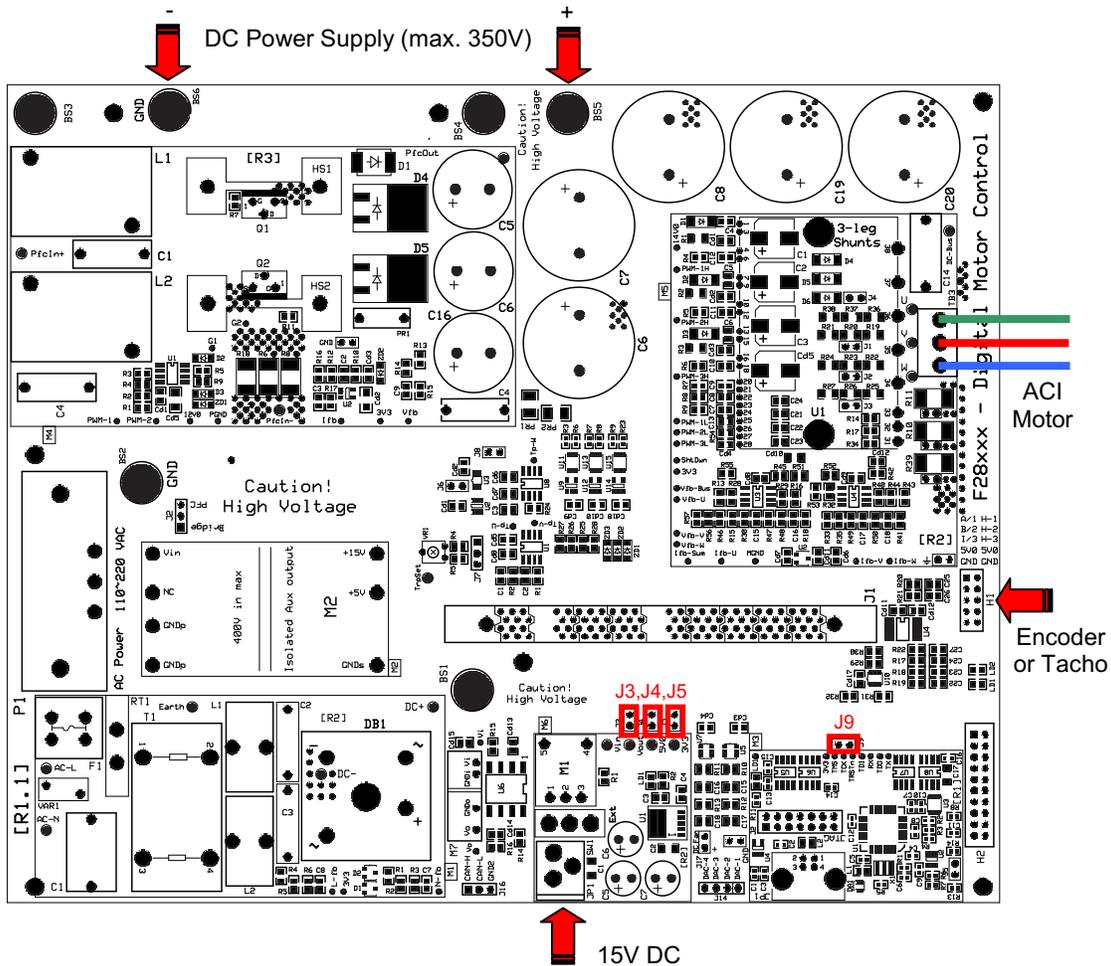


Figure 13. Using External DC Power Supply to Generate DC-Bus for the Inverter

**CAUTION**

The inverter bus capacitors remain charged for a long time after the high power line supply is switched off or disconnected. Proceed with caution!

### 8.1 Software Setup Instructions to Run HVPM\_Sensorless Project

For more information, see the *Software Setup for the HVMotorCtrl+PFC Kit Projects* section in the *HVMotorCtrl+PFC Kit How to Run Guide* that can be found at [www.ti.com/controlsuite](http://www.ti.com/controlsuite), then choose the HVMotorKit installation.

1. Select HVPM\_Sensorless as the active project.
2. Select the active build configuration to be set as F2803x\_RAM.
3. Verify that the build level is set to 1, and then right click on the project name and select "Rebuild Project". Once the build completes, launch a debug session to load the code into the controller.
4. Open a watch window and add the critical variables as shown in Figure 14 and select the appropriate Q format for them.

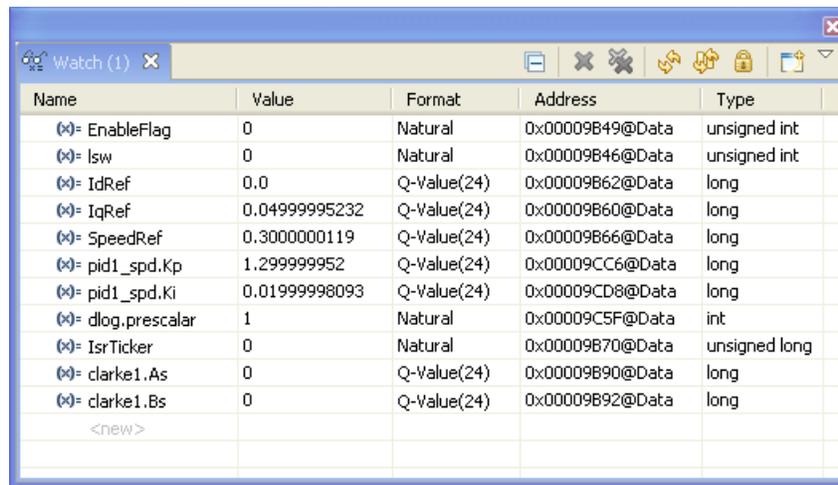


Figure 14. Watch Window Variables

5. Setup the time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location: [www.ti.com/controlsuite](http://www.ti.com/controlsuite) - (development\_kits\HVMotorCtrl+PfcKit\_v2.0\HVPM\_sensorless\).
6. Click on the Continuous Refresh button  on the top left corner of the graph tab to enable periodic capture of data from the microcontroller.

## 9 Incremental System Build

The system is gradually built up so the final system can be confidently operated. Four phases of the incremental system build are designed to verify the major software modules used in the system. Table 1 summarizes the modules testing and using in each incremental system build.

Table 1. Tested Modules in Each Incremental System Build <sup>(1)</sup>

Software Module	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
PWMDAC_MACRO	√	√	√	√	√
RC_MACRO	√	√	√	√	√
RG_MACRO	√	√	√	√	√
IPARK_MACRO	√√	√	√	√	√
SVGEN_MACRO	√√	√	√	√	√
PWM_MACRO	√√	√	√	√	√
CLARKE_MACRO		√√	√	√	√
PARK_MACRO		√√	√	√	√
QEP_MACRO			√√	√	√
SPEED_FR_MACRO			√√	√	√
PI_MACRO (IQ)			√√	√	√
PI_MACRO (ID)			√√	√	√
SMO_MACRO				√√	√√
SE_MACRO					√√
PI_MACRO (SPD)					√√

<sup>(1)</sup> The symbol √ means this module is using and the symbol √√ means this module is testing in this phase.

## 9.1 Level 1 Incremental Build

Keep the motor disconnected at this step. Assuming the load and build steps described in the *HVMotorCtrl+PFC Kit How To Run Guide* completed successfully, this section describes the steps for a “minimum” system check-out, which confirms the operation of the system interrupt, the peripheral and target independent I\_PARK\_MACRO (inverse park transformation) and SVGEN\_MACRO (space vector generator) modules, and the peripheral dependent PWM\_MACRO (PWM initializations and update) modules.

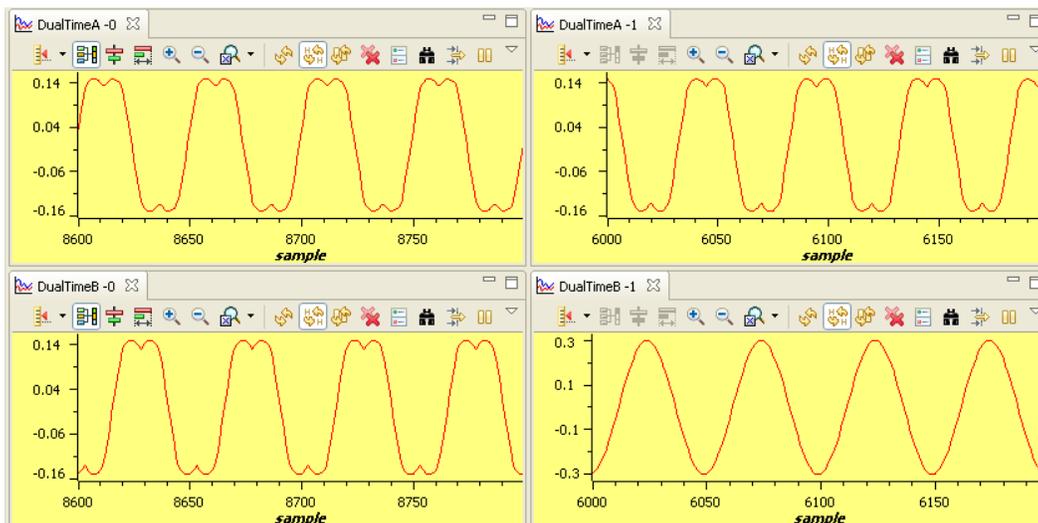
1. Open HVPM\_Sensorless-Settings.h and select the level 1 incremental build option by setting the BUILDLEVEL to LEVEL1 (#define BUILDLEVEL LEVEL1).
2. Right click on the project name and click Rebuild Project.
3. Click on the debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.
4. Set the “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will now keep on increasing.
5. Confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

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

- 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

## 9.2 Level 1A (SVGEN\_MACRO Test)

The SpeedRef value is specified to the RG\_MACRO module via the RC\_MACRO module. The IPARK\_MACRO module is generating the outputs to the SVGEN\_MACRO module. Three outputs from SVGEN\_MACRO module are monitored via the graph window as shown in [Figure 15](#) where Ta, Tb, and Tc waveform 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 make sure that the PWM module is running properly.



**Figure 15. Output of SVGEN, Ta, Tb, Tc and Tb-Tc Waveforms**

### 9.3 Level 1B (Testing The PWMDAC Macro)

To monitor the internal signal values in real time, PWM DACs are very useful tools. Present on the HV DMC board are PWM DAC's that use external low-pass filters to generate the waveforms ([Main]-J14, DAC-1 to 4). A simple first-order low-pass filter RC circuit is used to filter out the high frequency components. The selection of R and C value (or the time constant,  $\tau$ ) is based on the cut-off frequency ( $f_c$ ), for this type of filter; the relation is as follows:

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

For example,  $R = 1.8 \text{ k}\Omega$  and  $C = 100 \text{ nF}$ , it gives  $f_c = 884.2 \text{ Hz}$ . This cut-off frequency has to be below the PWM frequency. Using the formula above, one can customize the low-pass filters used for signal being monitored.

The DAC circuit low-pass filters ([Main]-R10 to13 and [Main]-C15 to18) are shipped with  $2.2 \text{ k}\Omega$  and  $220 \text{ nF}$  on the board. For more details, see *Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller* ([SPRAA88](#)).

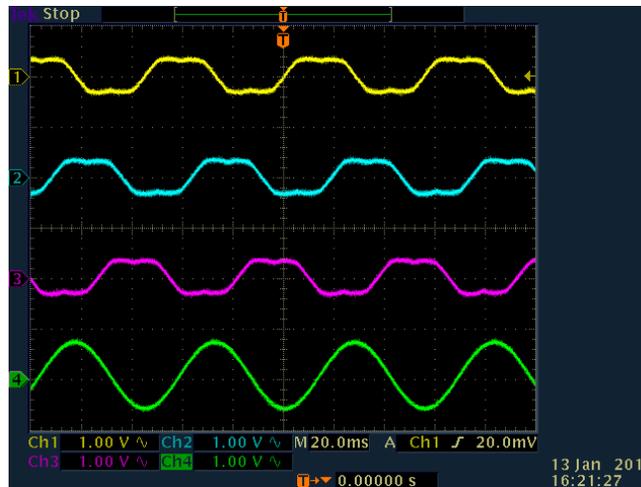


Figure 16. DAC-1-4 Outputs Showing  $T_a$ ,  $T_b$ ,  $T_c$  and  $T_b$ - $T_c$  Waveforms

### 9.4 Level 1C (PWM\_MACRO and INVERTER Testing)

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, V and W test points using an oscilloscope or 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, you will start observing the inverter phase voltage dividers and waveform monitoring filters (Vfb-U, Vfb-V, Vfb-W) enable the generation of the waveform, which ensures that the inverter is working appropriately. Note that 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), and reset the processor  (for details, see the *HVMotorCtrl+PFC Kit How To Run Guide*). Note that after each test, this step needs to be repeated for safety purposes. Also note that improper shutdown might halt the PWMs at some certain states where high currents can be drawn, therefore, caution needs to be taken while doing these experiments.

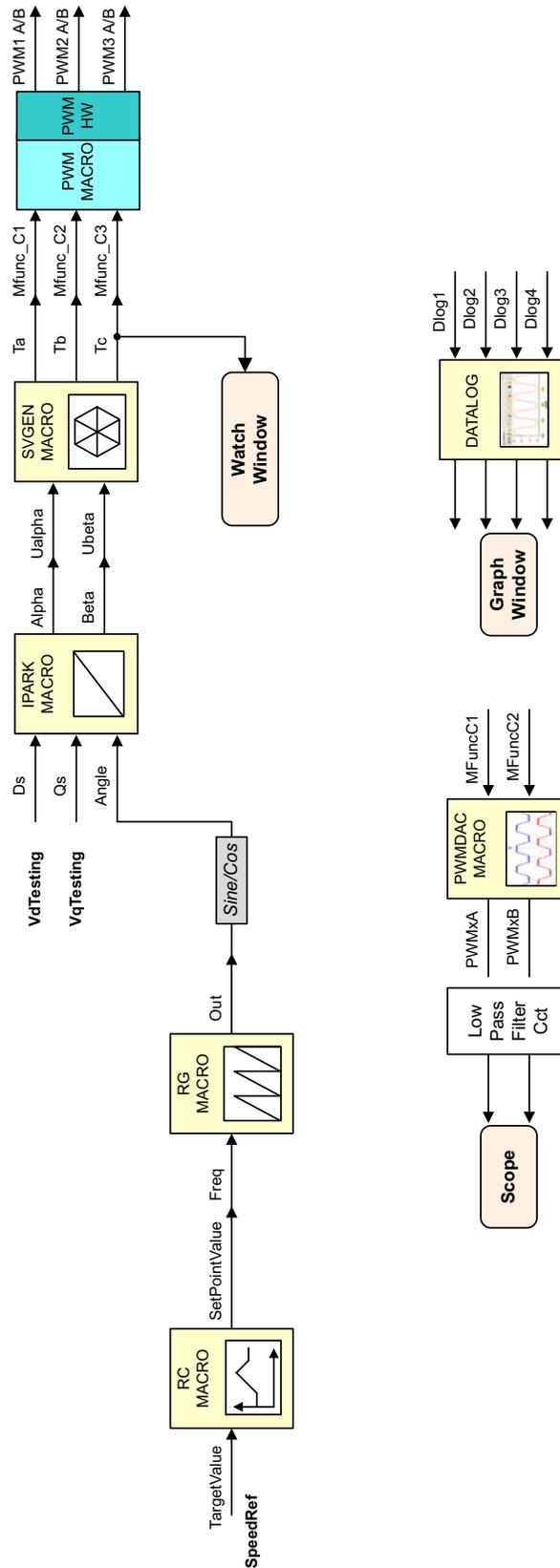


Figure 17. Level 1 - Incremental System Build Block Diagram

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

## 9.5 Level 2 - Incremental Build

Assuming section BUILD 1 is completed 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 HVDMC board since the PWM signals are successfully proven through level 1 incremental build. Note that the open loop experiments are meant to test the ADCs, inverter stage, software modules, and so forth. Therefore, running the motor under load or at various operating points is not recommended.

1. Open HVPM\_Sensorless-Settings.h and select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL LEVEL2) and save the file.
2. Right Click on the project name and click Rebuild Project.
3. Click on debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.
4. Set the "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" is incrementally increased as seen in the watch windows to confirm the interrupt working properly.

In the software, the key variables to be adjusted are summarized below.

- 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 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.

## 9.6 Level 2A – Testing the Phase Voltage Module

In this part, the phase voltage calculation module, VOLT\_MACRO, is tested. Now, gradually increase the DC bus voltage. The outputs of this module can be checked via the graph window as follows:

- 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 should be the same as the VphaseA waveform.
- The Valpha waveform should be leading the Vbeta waveform by 90° at the same magnitude.

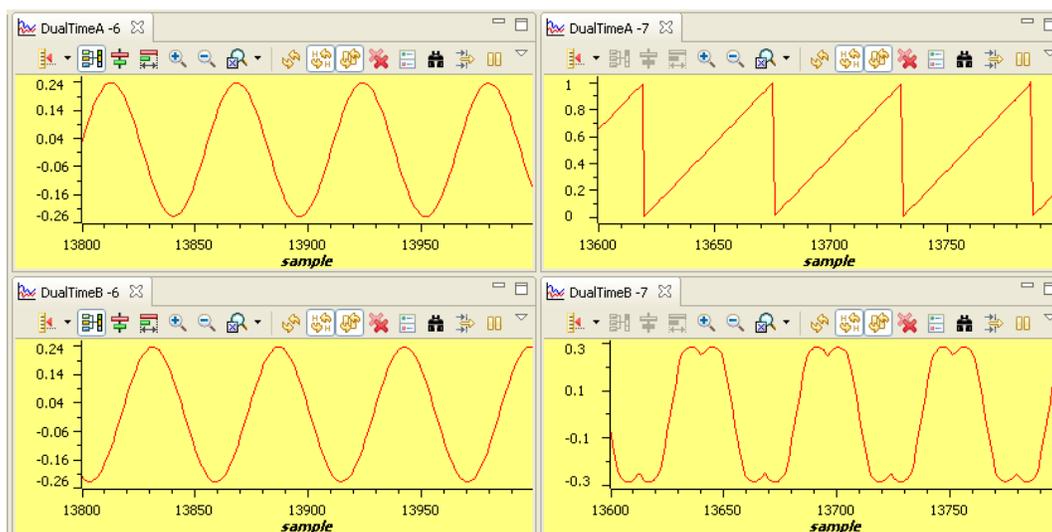


Figure 18. Calculated Phase A and B Voltages by volt1 Module, rg1.Out and svgen\_dq1.Ta

## 9.7 Level 2B – Testing the Clarke Module

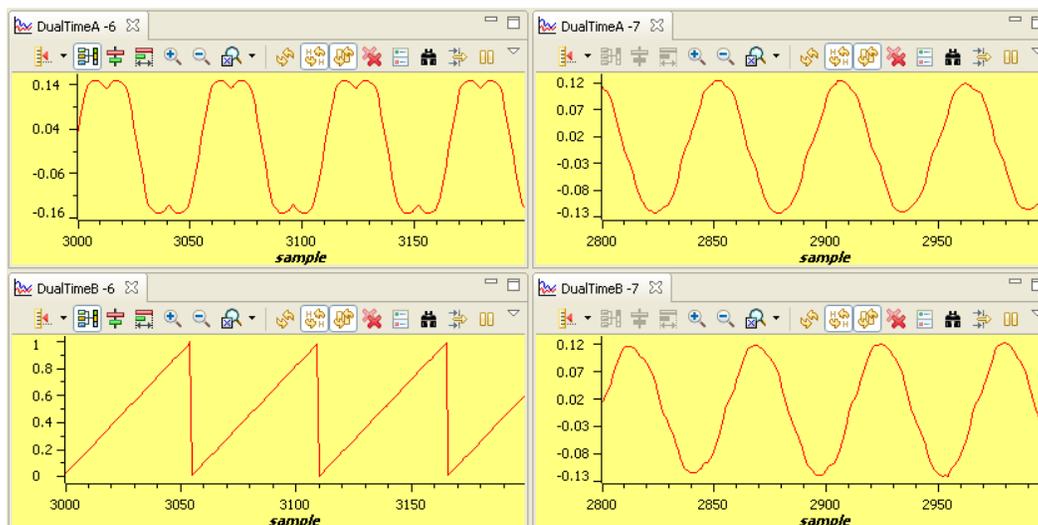
In this part, the Clarke module is tested. The three measured line currents are transformed to two phase dq currents in a stationary reference frame. The outputs of this module can be checked from the graph window.

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

It is important that the measured line current must be lagging with the reconstructing phase voltage because of the nature of the AC motor. This can be easily checked as follows:

- The clark1.Alpha waveform should be lagging the Valpha waveform at an angle by the nature of the reactive load of motor.
- The clark1.Beta waveform should be lagging 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 lagging relationship, then set OutofPhase to 1 at the beginning of the PHASEVOLT\_MACRO module. The outputs of this test can be checked via the graph window.



A Deadband = 0.83  $\mu$ sec, Vdcbus = 300 V , dlog.prescaler = 3

**Figure 19. The Waveforms of Svgen\_dq1.Ta, rg1.Out, and Phase A and B Currents**

## 9.8 Level 2C – Adjusting PI Limits

Note that the vectorial sum of d-q PI outputs should be less than 1.0, which refers to the maximum duty cycle for SVGEN macro. Another duty cycle limiting factor is the current sense through shunt resistors, which depends on hardware and software implementation. Depending on the application requirements 3, 2 or a single shunt resistor can be used for current waveform reconstruction. The higher number of shunt resistors allow higher duty cycle operation and better dc bus utilization.

Run the system with default VdTesting, VqTesting and SpeedRef and gradually increase VdTesting and VqTesting values. Meanwhile, watch the current waveforms in the graph window. Keep increasing until you notice distorted current waveforms and write down the maximum allowed VdTesting and VqTesting values. Make sure that these values are consistent with expected d-q current component maximums while running the motor. After this build level, PI outputs automatically generate the voltage reference and determine the PWM duty cycle depending on the d-q current demand, therefore, set pi\_id.Umax and min and pi\_iq.Umax and min according to recorded VdTesting and VqTesting values, respectively.

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

---

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 realtime mode and reset.

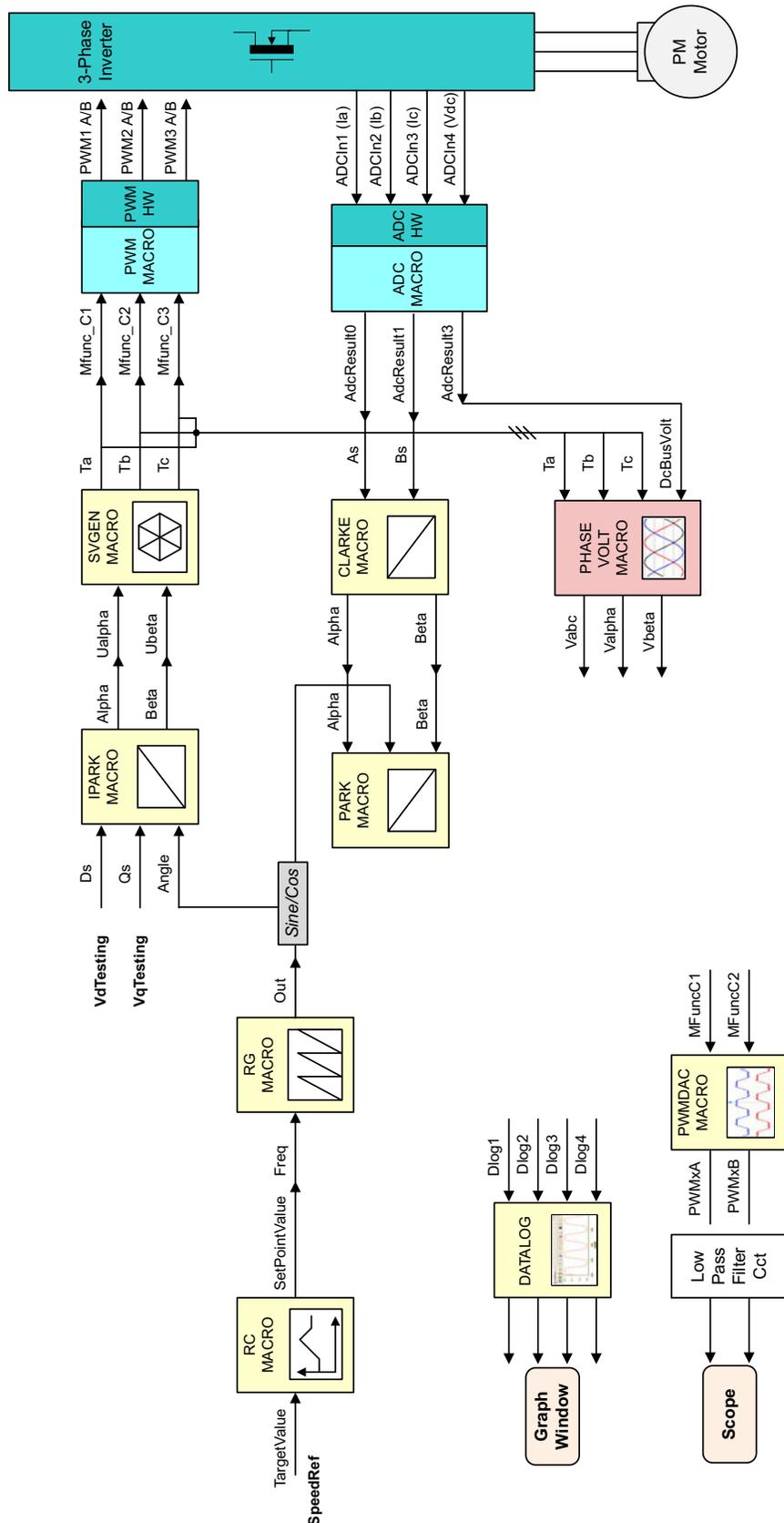


Figure 20. Level 2 - Incremental System Build Block Diagram

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

### 9.9 Level 3 Incremental Build

Assuming the previous section is completed successfully, this section verifies the dq-axis current regulation performed by PI modules and speed measurement modules. To confirm the operation of current regulation, the gains of these two PI controllers are necessarily tuned for proper operation.

1. Open `HVPM_Sensorless-Settings.h` and select the level 3 incremental build option by setting the `BUILDLLEVEL` to `LEVEL3` (`#define BUILDLLEVEL LEVEL3`).
2. Right click on the project name and click Rebuild Project.
3. Click on the debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.
4. Set the “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” is incrementally increased as seen in the watch windows to confirm the interrupt working properly.

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

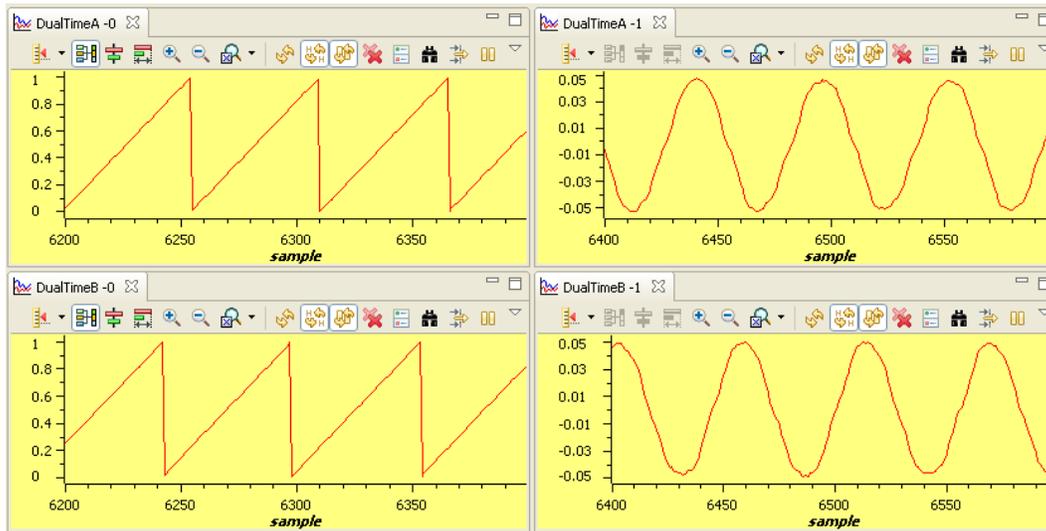
- `SpeedRef` (Q24): for changing the rotor speed in per-unit.
- `IdRef`(Q24): for changing the d-qxis voltage in per-unit.
- `IqRef`(Q24): for changing the q-axis voltage in per-unit.

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

The key steps are explained as follows:

- Compile, load, and run the program with real-time mode.
- Set `SpeedRef` to 0.3 pu (or another suitable value if the base speed is different), `Idref` to a certain value to generate rated flux.
- Gradually increase the voltage at the variac and dc power supply to get an appropriate DC-bus voltage.
- Add the soft-switch variable “`lsw`” to the watch window in order to switch from the current loop to the speed loop. In the code `lsw` manages the loop setting as follows:
  - `lsw = 0`, lock the rotor of the motor
  - `lsw = 1`, run the motor with closed current loop
- Check `pi_id.Fdb` in the watch windows with the continuous refresh feature whether or not it should be keeping track `pi_id.Ref` for the PI module. If not, adjust its PI gains properly.
- Check `pi_iq.Fdb` in the watch windows with the continuous refresh feature whether or not it should be keeping track `pi_iq.Ref` for PI module. If not, adjust its PI gains properly.
- Try different values of `pi_id.Ref` and `pi_iq.Ref` or `SpeedRef` to confirm these two PI modules.
- For both PI controllers, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied responses.
- 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. Now the motor should stop, once stopped terminate the debug session.

When running this build, the current waveforms in the Code Composer Studio graphs should appear as shown in [Figure 21](#).<sup>(2)</sup>



**Figure 21. rg1.Out, Measured theta and Phase A and B Current Waveforms**

### 9.10 Level 3B – QEP and 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. Make sure that the output of the incremental encoder is connected to the [Main]-H1, the QEP, and the SPEED\_FR macros are initialized properly in the HVPM\_Sensorless.c file depending on the features of the speed sensor. Refer to the pdf files regarding the details of related macros in the motor control folder located at [www.ti.com/controlsuite](http://www.ti.com/controlsuite) - (libs\app\_libs\motor\_control).

The steps to verify these two software modules related to the speed measurement can be described as follows:

- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
- Compile, load, and run the program with real-time mode and then increase the voltage at the variac and dc power supply to get the appropriate DC-bus voltage.
- Add the soft-switch variable “lsw” to the watch window in order to switch from the current loop to the 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
- Set lsw to 1. Now the motor is running close to reference speed.
- Check the “speed1.Speed” in the watch windows with the continuous refresh feature whether or not the measured speed is around the speed reference.
- Try different values of SpeedRef to test the speed to confirm these modules.
- Use the oscilloscope to view the electrical angle output, ElecTheta, from the QEP\_MACRO module and the emulated rotor angle, rg1.Out, from RG\_MACRO at PWM DAC outputs with external low-pass filters.
- Check that both ElecTheta and rg1.Out are of saw-tooth wave shape and have the same period. If the measured angle is in the opposite direction, then change the order of motor cables connected to the inverter output (TB3 for HVDMC kit).
- Qep1.ElecTheta should be slightly lagging rg1.out, if not adjust the calibration angle.
- Check from the watch window that qep1.IndexSyncFlag is set back to 0xF0 every time it resets to 0 by hand. Add the variable to the watch window if it is not already in the watch window.

<sup>(2)</sup> Deadband = 0.83  $\mu$ sec, Vdcbus = 300 V, dlog.trig\_value = 100

- Qep1.ElecTheta should be slightly lagging rg1.out, if the calibration angle needs to be adjusted due to the angle offset between index and locked rotor position.
- 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.
- The calibration angle of the encoder is detected in the code as detailed below. Note that this is an optional procedure for sensorless FOC speed loop tuning. If you prefer to use the encoder to tune the speed loop and apply the schemes given in [Section 9.13](#), the exact rotor position is not needed. However, if you tune the speed loop as in [Section 9.15](#), the exact rotor position information is needed and the calibration angle has to be detected.

Next, the following are key steps to verify or perform the calibration angle of the encoder.

- Make sure EQep1Regs.QPOSCNT, EQep1Regs.QPOSILAT, Init\_IFlag, qep1.CalibratedAngle, and lsw are displayed in the watch window.
- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
- Compile, load, and run the program with real-time mode and then increase the voltage at the variac and dc power supply to get the appropriate DC-bus voltage.
- Lock the rotor.
- 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 in between the 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 HVPM\_Sensorless.c or it can be detected in the code each time the motor is restarted. The calibration angle might be different for different start-ups and can be formulated as follows:

$$\text{Calibration Angle} = \text{Offset Angle} \pm n \cdot \text{Line Encoder}$$

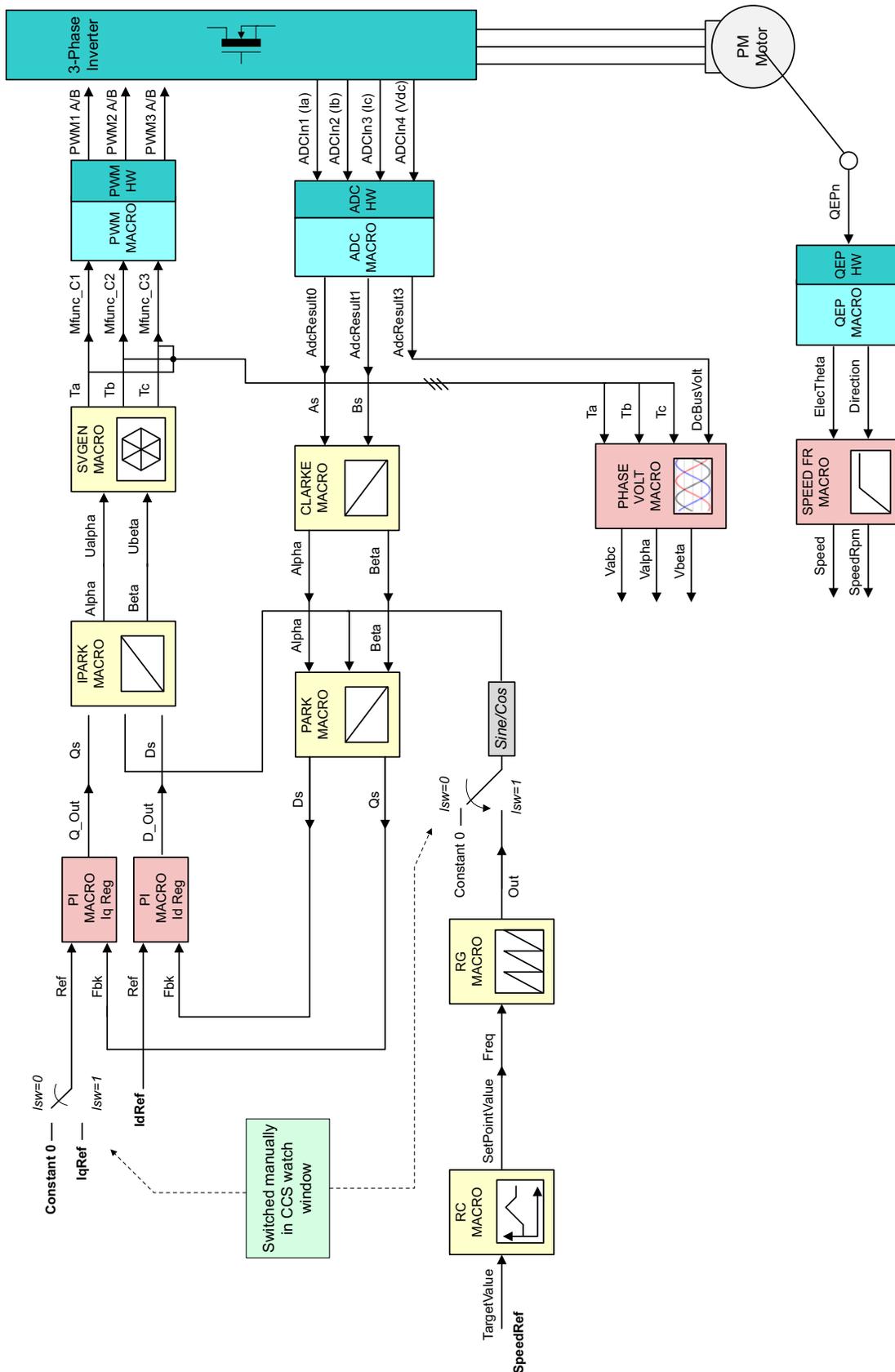


Figure 22. Level 3 - Incremental System Build Block Diagram

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

### 9.11 Level 4 Incremental Build

Assuming the previous section is completed successfully, this section verifies the estimated rotor position and speed estimation performed by SMOPOS (sliding mode observer) and SPEED\_EST modules, respectively.

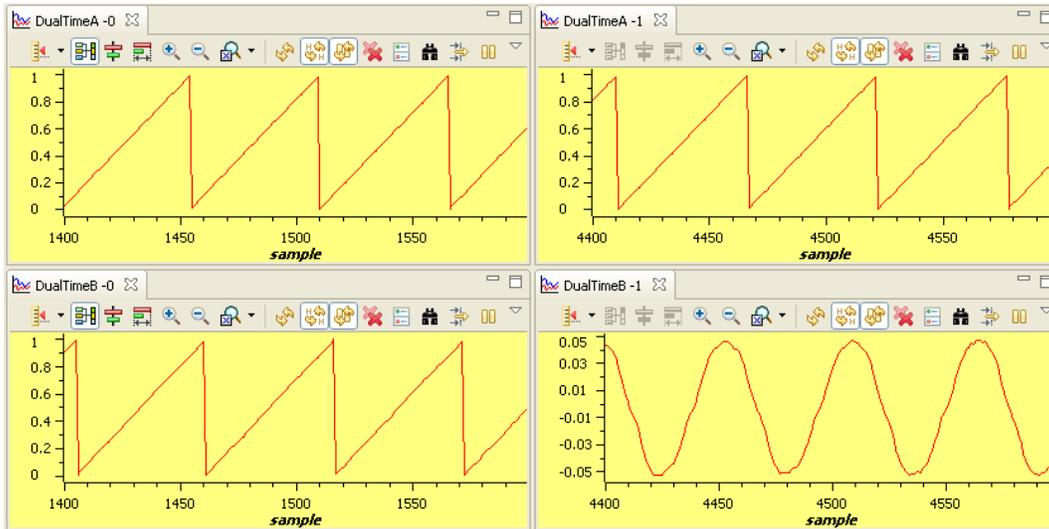
1. Open HVPM\_Sensorless-Settings.h and select level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4).
2. Right Click on the project name and click Rebuild Project.
3. Click on debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.
4. Set the “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” is incrementally increased as seen in the watch windows to confirm the interrupt working properly.
  - 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) inside the rotor position estimator may be critical for low speed operation.

The key steps can be explained as follows:

- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
- Compile, load, and run the program with real-time mode and then increase voltage at the variac and dc power supply to get the appropriate DC-bus voltage.
- Add the soft-switch variable “lsw” to the watch window in order to switch from the current loop to the 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
- Set lsw to 1. Now the motor is running close to the reference speed. Compare smo1.Theta with rg1.Out via PWMDAC with the external low-pass filter and an oscilloscope. They should be identical with a small phase shift.
- If smo1.Theta does not give the sawtooth waveform, the Kslide and Kslf inside the sliding mode observer are required to be re-tuned.
- To confirm the rotor position estimation, try different values of SpeedRef.
- Compare se1.WrHat (estimated speed) with reference speed or measured speed in the watch windows with the continuous refresh feature whether or not it should be nearly the same.
- Try different values of SpeedRef to confirm this open-loop speed estimator.
- 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.

When running this build, the current waveforms in the Code Composer Studio graphs should appear as shown in [Figure 23](#).<sup>(3)</sup>



**Figure 23. Measured theta, Estimated theta (SMO), rg1.Out and Phase A Current**

<sup>(3)</sup> dlog.trig\_value = 100, deadband = 1.66  $\mu$ sec, Vdcbus = 300 V



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

### 9.12 Level 5 Incremental Build

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

1. Open HVPM\_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. Click on the debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.
4. Set the "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will now keep on increasing.
5. Confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are summarized below.

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

### 9.13 Level 5A

The speed loop is closed by using measured speed. The key steps can be explained as follows:

- Compile, load, and run the program with real-time mode.
- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
- Add the soft-switch variable "lsw" to the watch window in order to switch from the current loop to speed loop. In the code lsw1 manages the loop setting as follows:
  - lsw = 0, lock the rotor of the motor
  - lsw = 1, close the current loop
  - lsw = 2, close the speed loop
- Set lsw to 1. Gradually increase the voltage at the variac and the dc power supply to get an appropriate DC-bus voltage. Now the motor is running around the reference speed (0.3 pu). Next, set lsw to 2 and close the speed loop.
- Compare Speed with SpeedRef in the watch windows with the continuous refresh feature whether or not it should be nearly the same.
- Try different values of SpeedRef to confirm this speed PI module.
- For speed PI controller, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied responses.
- At very low speed range, the performance of speed response relies heavily on the good rotor flux angle computed by flux estimator.
- 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 realtime mode and reset.
- Note that the IdRef is set to be zero all the times.

### 9.14 Level 5B (Alternative Method)

Tuning both speed PI and SMO at the same time may not be easy for some applications. In order to test the SMO only without speed PI in the loop, disconnect the speed PI and Iq PI modules in the code as shown in [Figure 26](#), 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.

Note that in this scheme the speed is not controlled, therefore, a non-zero torque reference ( $I_{qref}$ ) will spin the motor very fast unless loaded. Keep the  $I_{qref}$  low initially, and load the motor using a brake, generator, and so forth (or manually if the motor is small enough). If the motor speed is too low or the generated torque by the motor is not enough to handle the applied load, increase  $I_{qref}$  or reduce the amount of load. After tuning the SMO, add speed PI into the system as shown in the [Figure 30](#) and tune the PI coefficients, if needed. This method help you 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 HVPM\_Sensorless-Settings.h. Make sure that the base (pu) quantities are set to maximum measurable current, voltage, and so forth, and that the motor electrical parameters are correct.
- The DC bus voltage should be high enough in order not to saturate the PI outputs
- Run the same experiment again and keep tuning SMO gains

### 9.15 Level 5C (Alternative Method)

Use the final scheme proposed in the sensorless FOC of the PMSM project in controlSUITE. This allows you to eliminate the position estimation algorithm and tune the speed loop employing the precise position and speed feedback. For further details of this scheme, please refer to the lab manual of sensed FOC of PMSM explaining the incremental build levels in detail.



Level 5 verifies the speed PI macro and speed loop.



## 9.16 Level 6 Incremental Build

Assuming the previous section is completed 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 `HVPM_Sensorless-Settings.h` and select 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. Click on the debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.
4. Set the “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will now keep on increasing.
5. Confirm this by watching the variable in the watch window. This confirms that the system interrupt is working properly.

In the software, the key variables to be adjusted are summarized below.

- SpeedRef (Q24): for changing the rotor speed in per-unit
- SpeedRef (Q24): for changing the rotor speed in per-unit

The speed loop is closed by using measured speed. The key steps can be explained as follows:

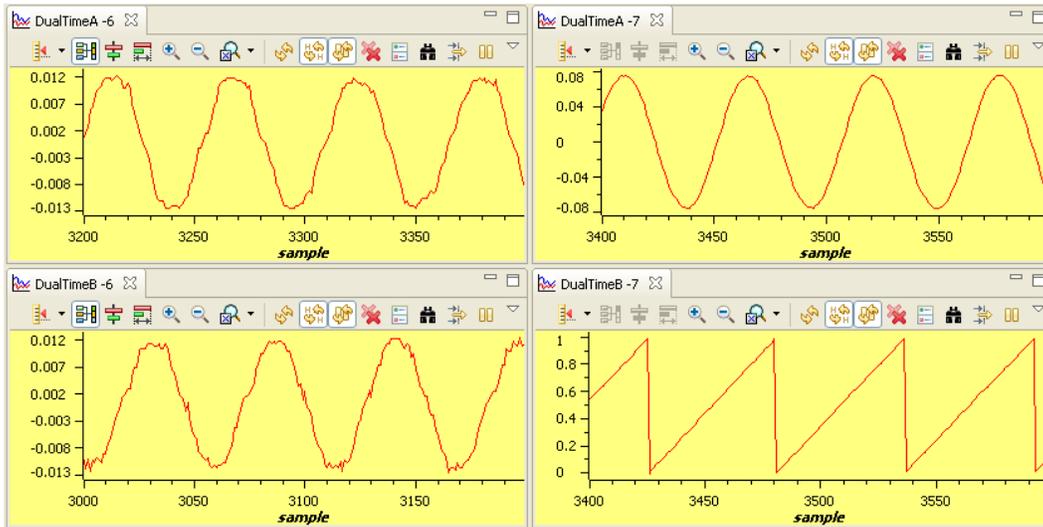
- Compile, load, and run the program with real-time mode.
- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
- Add the soft-switch variable “lsw” to the watch window in order to switch from the current loop to speed loop. In the code lsw1 manages the loop setting as follows:
  - lsw = 0, lock the rotor of the motor
  - lsw = 1, close the current loop
  - lsw = 2, close the speed loop
- Set lsw to 1. Gradually increase the voltage at the variac and the dc power supply to get an appropriate DC-bus voltage. Now the motor is running around the reference speed (0.3 pu). Next, set lsw to 2 and close the speed loop. After a few tests, you can determine the best time to close the speed loop depending on the load-speed profile and then close the speed loop in the code. For most of the applications, the speed loop can be closed before the motor speed reaches to SpeedRef.
- Compare `se1.WrHat` with SpeedRef in the watch windows with the continuous refresh feature whether or not it should be nearly the same.
- Try different values of SpeedRef to confirm this speed PI module.
- For speed PI controller, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied responses.
- At very low speed range, the performance of speed response relies heavily on the good rotor flux angle computed by flux estimator.
- 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 realtime mode and reset.
- Note that the `ldRef` is set to be zero all the times.

---

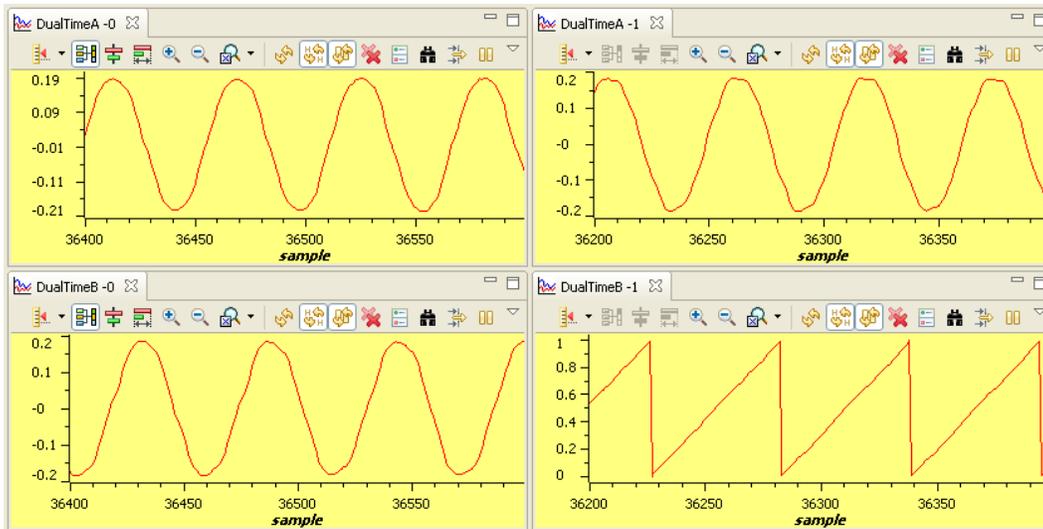
**NOTE:** The first-order low-pass filter inside the SMO module causes small amount of estimated angle delay. In order to achieve accurate field orientation, it is recommended to compensate this delay. Once the delays are detected for different operating points, they can be interpolated by means of a simple second or third order equation and this equation can be added to the code. Please refer to the `smopos.pdf` for the details of SMO: `..controlSUITE\libs\app_libs\motor_control\math_blocks\v4.0\~Docs`.

---

When running this build, the current waveforms in the CCS graphs should appear as follows: <sup>(4)</sup>

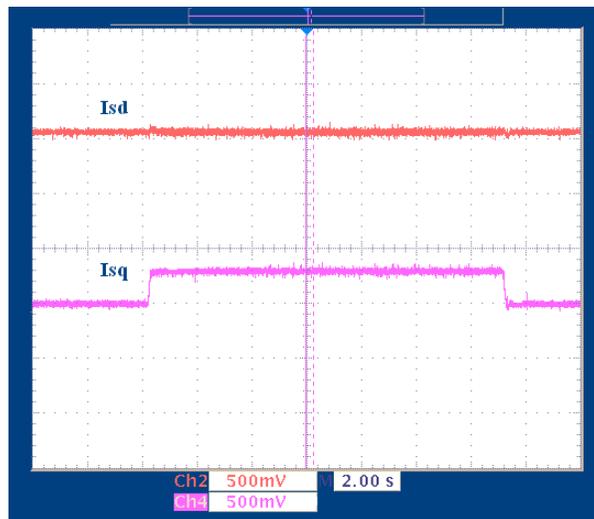


**Figure 27. Waveforms of Phase A and B Currents, Calculated Phase A Voltage, and Estimated theta by SMO Under No-Load and 0.3pu Speed**



**Figure 28. 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**

<sup>(4)</sup> dlog.trig\_value = 100, deadband = 1.66  $\mu$ sec, Vdcbus = 300 V, pi\_spd.Kp = 1.0



**Figure 29. 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 PWM DAC Output**

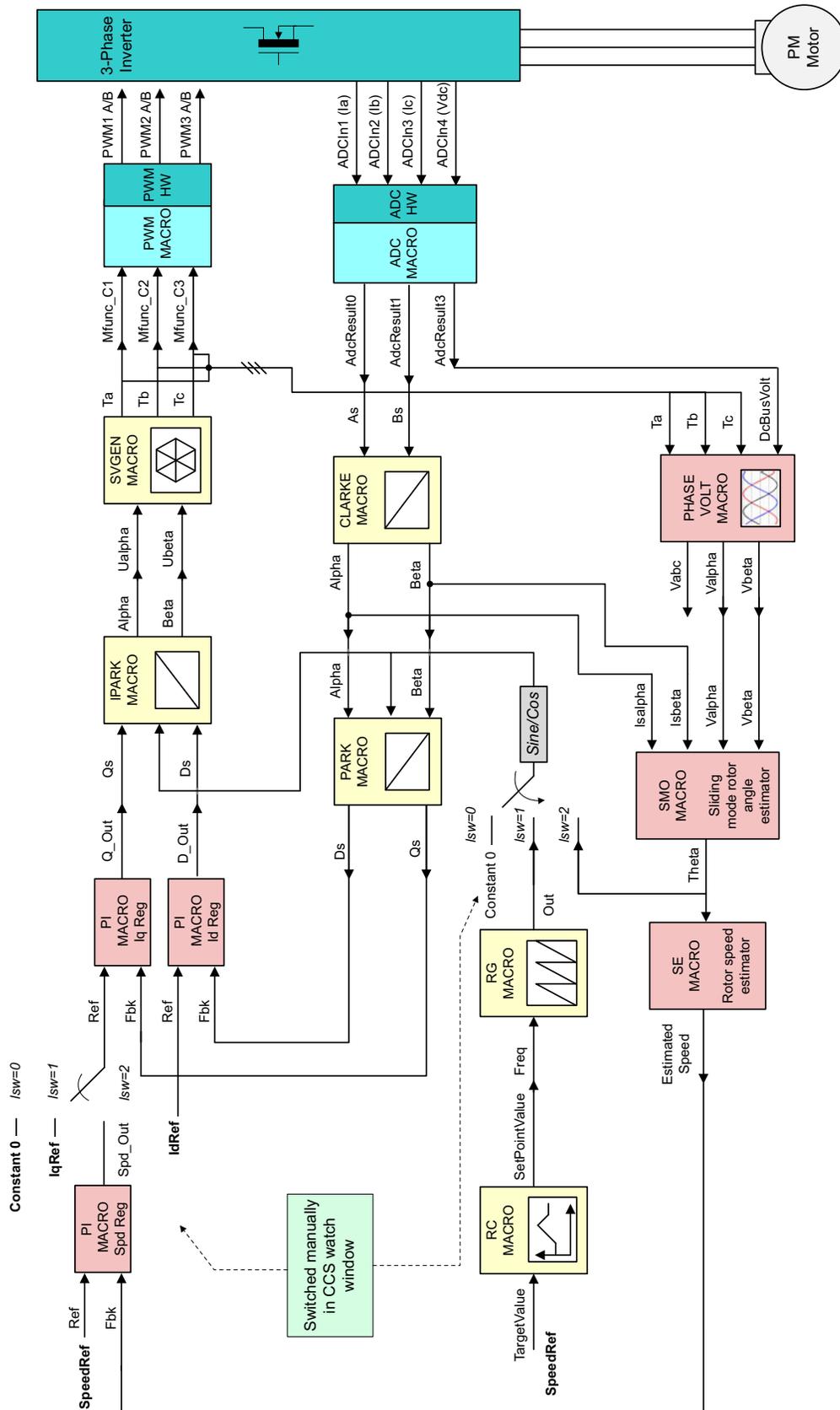


Figure 30. Level 6 - Incremental System Build Block Diagram

## 10 References

- *Using PWM Output as a Digital-to-Analog Converter on a TMS320F280x Digital Signal Controller* ([SPRAA88](#))
- *Optimizing Digital Motor Control (DMC) Libraries* ([SPRAAK2](#))

## 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

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
OMAP Applications Processors	<a href="http://www.ti.com/omap">www.ti.com/omap</a>
Wireless Connectivity	<a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a>

### Applications

Automotive and Transportation	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Communications and Telecom	<a href="http://www.ti.com/communications">www.ti.com/communications</a>
Computers and Peripherals	<a href="http://www.ti.com/computers">www.ti.com/computers</a>
Consumer Electronics	<a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>
Energy and Lighting	<a href="http://www.ti.com/energy">www.ti.com/energy</a>
Industrial	<a href="http://www.ti.com/industrial">www.ti.com/industrial</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Space, Avionics and Defense	<a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a>
Video and Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>

### TI E2E Community

[e2e.ti.com](http://e2e.ti.com)