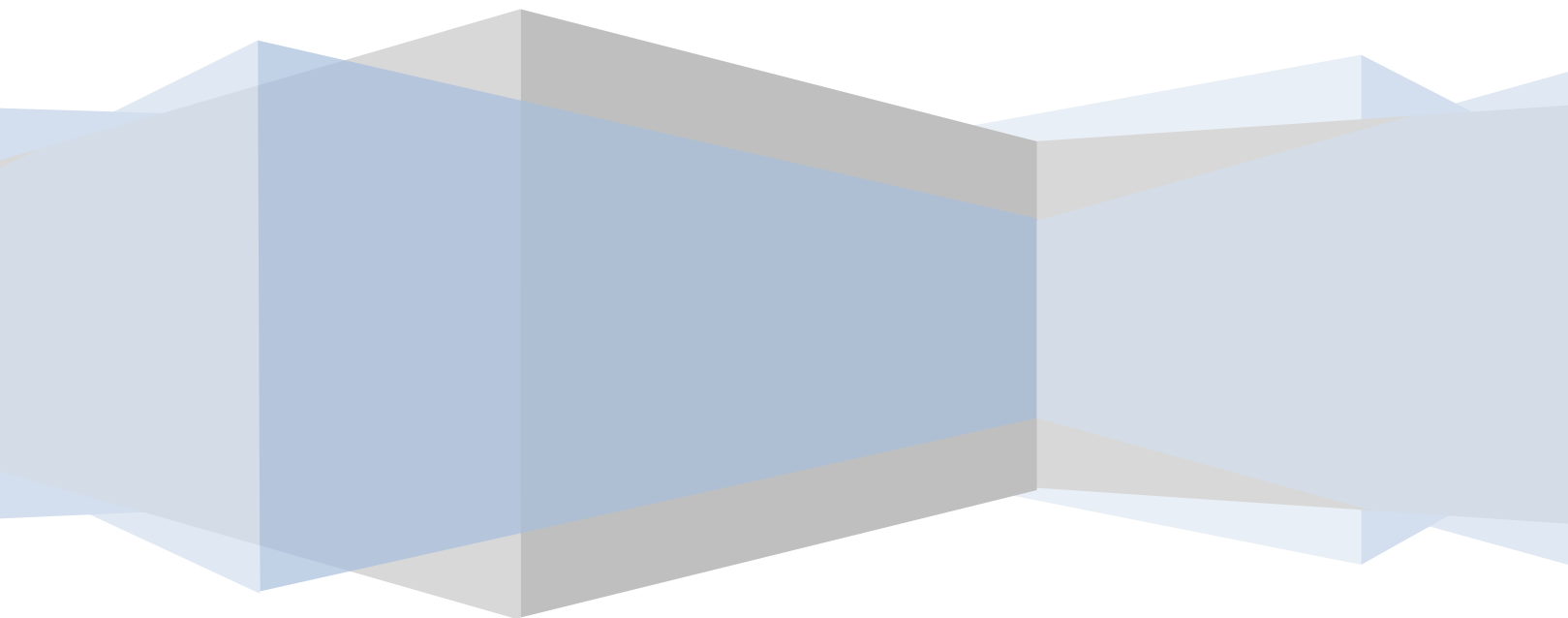


Texas Instruments, Inc.
C2000 Systems and Applications

Sensored Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors using F2837x

Authors: Ramesh T Ramamoorthy, Brett Larimore, Manish Bhardwaj



Contents

Introduction	3
PMSM Motors	3
Field Oriented Control	5
Benefits of 32-bit C2000 Controllers for Digital Motor Control	11
TI Motor Control Literature and DMC Library	12
System Overview.....	13
Hardware Configuration.....	15
Software Setup Instructions to Run HVPMSM_Sensored Project	18
Incremental System Build.....	19

Abstract

This application note presents a solution to control a permanent magnet synchronous motor (PMSM) using the TMS320F2837x microcontrollers. TMS320F2837x devices are part of the family of C2000 microcontrollers which enable cost-effective design of intelligent controllers for three phase motors by reducing the system components and increase 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.

This application note covers the following:

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

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 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 TMS320F2837x . It 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 field orientated control (FOC). This algorithm maintains efficiency in a wide range of speeds and takes into consideration torque changes with transient phases by controlling the flux directly from the rotor coordinates. This application report presents the implementation of a control for sinusoidal PMSM motor. The sinusoidal voltage waveform applied to this motor is created by using the Space Vector modulation technique. Minimum amount of torque ripple appears when driving this sinusoidal BEMF motor with sinusoidal currents.

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 any 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 permit a considerable reduction in motor dimensions while maintaining a very high power density.

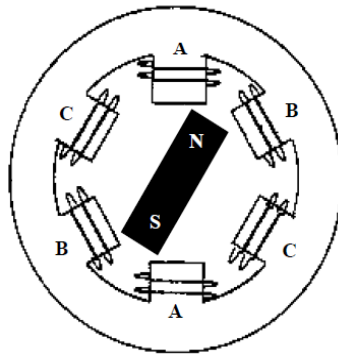


Figure 1 A three-phase synchronous motor with one permanent magnet pair pole rotor

Synchronous Motor Operation

- Synchronous motor construction: Permanent magnets are rigidly fixed to the rotating axis to create a constant rotor flux. The stator windings when energized with three phase voltages create a rotating electromagnetic field. To control the rotating magnetic field, it is necessary to control the stator currents.
- The actual structure of the rotor varies depending on the power range and rated speed of the machine. Permanent magnets are suitable for synchronous machines ranging up-to a few Kilowatts. For higher power ratings the rotor usually consists of windings in which a DC current circulates. The mechanical structure of the rotor is designed for number of poles desired, and the desired flux gradients.
- 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 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 in order to draw minimum amount of current under the same speed and torque conditions.
- The stator field must rotate at the same frequency as the rotor field; otherwise the rotor will experience rapidly alternating positive and negative torque. This will result in less than optimal torque production, and excessive mechanical vibration, noise, and mechanical stresses on the machine parts. In addition, if the rotor inertia prevents the rotor from being able to respond to these oscillations, the rotor will stop rotating at the synchronous frequency, and respond to the average torque as seen by the stationary rotor: Zero. This means that the machine experiences a phenomenon known as 'pull-out'. This is also the reason why the synchronous machine is not self starting.
- The angle between the rotor field and the stator field must be equal to 90° to produce maximum torque. 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 different stator phase currents to produce the resulting stator flux.

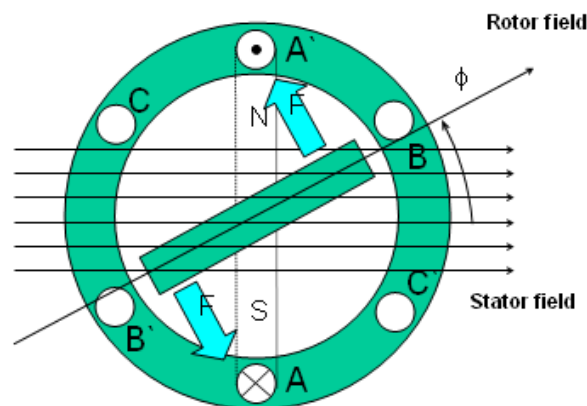


Figure 2 The interaction between the rotating stator flux, and the rotor flux produces a torque which will cause the motor to rotate.

Field Oriented Control

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, we can implement advanced control strategies, which use mathematical transformations to control AC machines like DC machines, providing independent control of flux and torque producing currents. Such de-coupled torque and magnetization control is commonly called Field Oriented Control (FOC).

The main philosophy behind the FOC

In order to understand the spirit of the Field Oriented Control technique, let us start with an overview of the separately excited direct current (DC) Motor. Torque is defined as the cross product of armature current and stator flux. **Electrical study of the DC motor shows that the armature current and the stator flux can be independently tuned.** The strength of the field excitation (i.e. the magnitude of the field excitation current) sets the value of the stator flux. If the flux is held constant, then 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/current.**

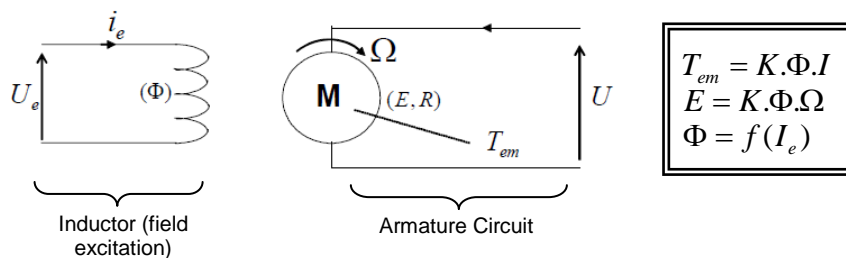


Fig 3 Separately excited DC motor model, flux and torque are independently controlled and the current through the rotor windings determines how much torque is produced.

AC machines do not have the same key features as the DC motor. The flux and torque producing current are not necessarily orthogonal. In PM synchronous machines, the rotor excitation is given by the permanent magnets mounted onto the shaft and stator carries the torque producing current. In induction machines, the stator carries both flux producing and torque producing currents and its only source of power is the stator phase voltage. The flux and torque producing components of currents are strongly coupled unlike in a DC machine.

The goal of the FOC (also called vector control) on synchronous and asynchronous machine is to be able to control those like a separately excited DC machine wherein the flux producing and torque producing currents are separately controlled. In other words, the control technique goal is, in a sense, to imitate the control of a DC motor. FOC control will allow us to decouple the flux and torque producing currents enabling them to be controlled independently. To decouple the torque and flux producing currents, 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.

Torque can be defined in multiple ways, as the cross product of stator current and rotor flux, or, as the cross product of stator flux and rotor flux as given below

$$T_{em} = \vec{B}_{stator} \times \vec{B}_{rotor}, \text{ or, } T_{em} = \vec{I}_{stator} \times \vec{B}_{rotor}$$

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

A 3 phase PM synchronous machine can be represented as a DC machine in synchronous DQ reference frame, where the D-axis is aligned along the rotor magnet flux and the Q axis is orthogonal to D axis. Any current flowing along D-axis, called direct component of current, can impact the strength of magnetic field and the current in Q axis, called quadrature current, will interact with the magnetic flux in D-axis to produce torque. In brief, for a PM motor, the goal is to maintain the d-axis current at zero and adjust the magnitude of current in Q-axis to generate the commanded torque. The direct component of the stator current can be kept negative in some cases for field weakening, which has the effect of reducing the rotor flux, and reducing the back-emf allowing for operation at higher speeds.

Technical Background

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

- The ease of reaching constant reference (torque component and flux component of the stator current)
- The ease of 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 (ψ_R) at a fixed value we have a linear relationship between torque and torque component of stator current vector (i_{sq}). We can then control the torque by controlling the torque component.

Space Vector Definition and Projection

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

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

where $\alpha = e^{j\frac{2}{3}\pi}$ and $\alpha^2 = e^{j\frac{4}{3}\pi}$, represent the spatial operators. The following diagram shows the stator current complex space vector:

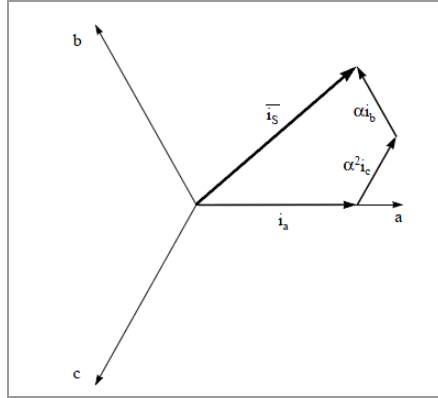


Figure 4 Stator current space vector and its component in (a,b,c)

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

- (a,b,c) \Rightarrow (α, β) (the Clarke transformation) which outputs a two co-ordinate time variant system
- (α, β) \Rightarrow (d,q) (the Park transformation) which outputs a two co-ordinate time invariant system

The (a,b,c) \Rightarrow (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 the axis a and the axis α are in the same direction we have the following vector diagram:

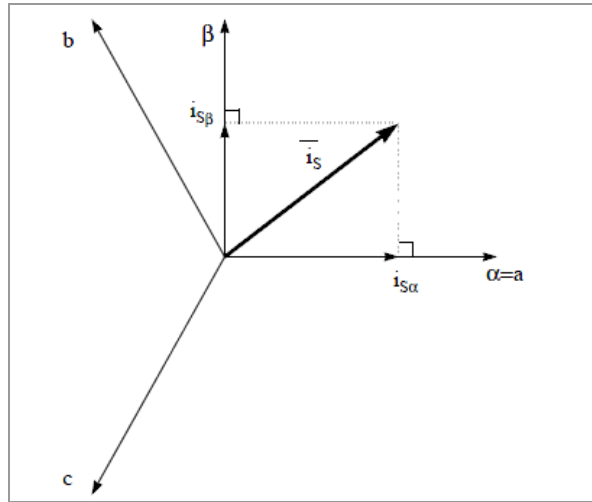


Figure 5 Stator current space vector and its components in the stationary reference frame

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 still depends on time and speed.

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 (α, β) in the d,q rotating reference frame. If we consider the d axis aligned with the rotor flux, the next diagram shows, for the current vector, the relationship from the two reference frame:

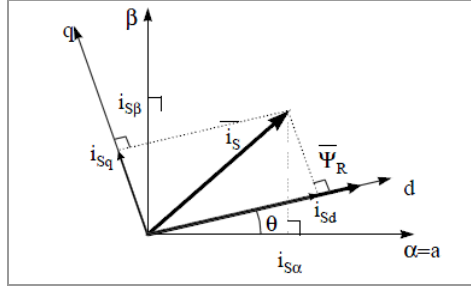


Figure 6 Stator current space vector and its component in (α, β) and in d,q rotating reference frame

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

The Basic Scheme for the FOC

The following diagram summarizes the basic scheme of torque control with FOC:

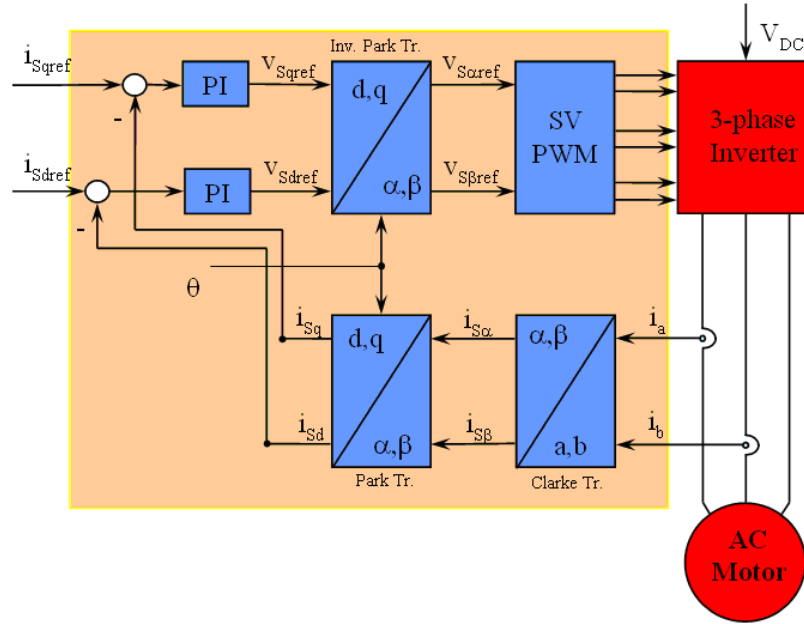


Figure 7 Basic scheme of FOC for AC motor

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 along with rotor flux position are the inputs of the Park transformation, which transform them to currents (i_{sd} and i_{sq}) in 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. In synchronous permanent magnet motor, the rotor flux is fixed as determined by the magnets. Hence there is no need to create it. Hence, when controlling a PMSM, i_{sdrref} should be set to zero. As ACIM 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} can be connected to the output of the speed regulator. The outputs of the current regulators are V_{sdrref} and V_{sqref} ; they are applied to the inverse Park transformation. Using the position of rotor flux, this projection generates $V_{s\alpha ref}$ and $V_{s\beta ref}$ which are the components of the stator vector voltage in the (α, β) stationary orthogonal reference frame. These are the inputs of the Space Vector PWM. The outputs of this block are the signals that drive the inverter. Note that both Park and inverse Park transformations need the rotor flux position. Obtaining this rotor flux position depends on the AC machine type (synchronous or asynchronous machine). Rotor flux position considerations are made in a following paragraph.

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} will represent 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 frame at synchronous speed.

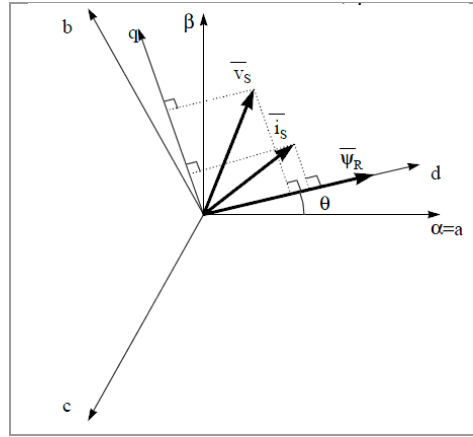


Figure 8 Current, voltage and rotor flux space vectors in the d,q rotating reference frame and their relationship with a,b,c and (α, β) stationary reference frame

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

Theoretically, the field oriented control for PMSM drive allows the motor torque be controlled independently like DC motor. In other words, the torque component of current and flux are decoupled from each other. The rotor position is required for variable transformation from stationary reference frame to synchronously rotating reference frame. Therefore, the key module of this system is the information of rotor position from QEP encoder. The overall block diagram of this project is depicted in Figure 9.

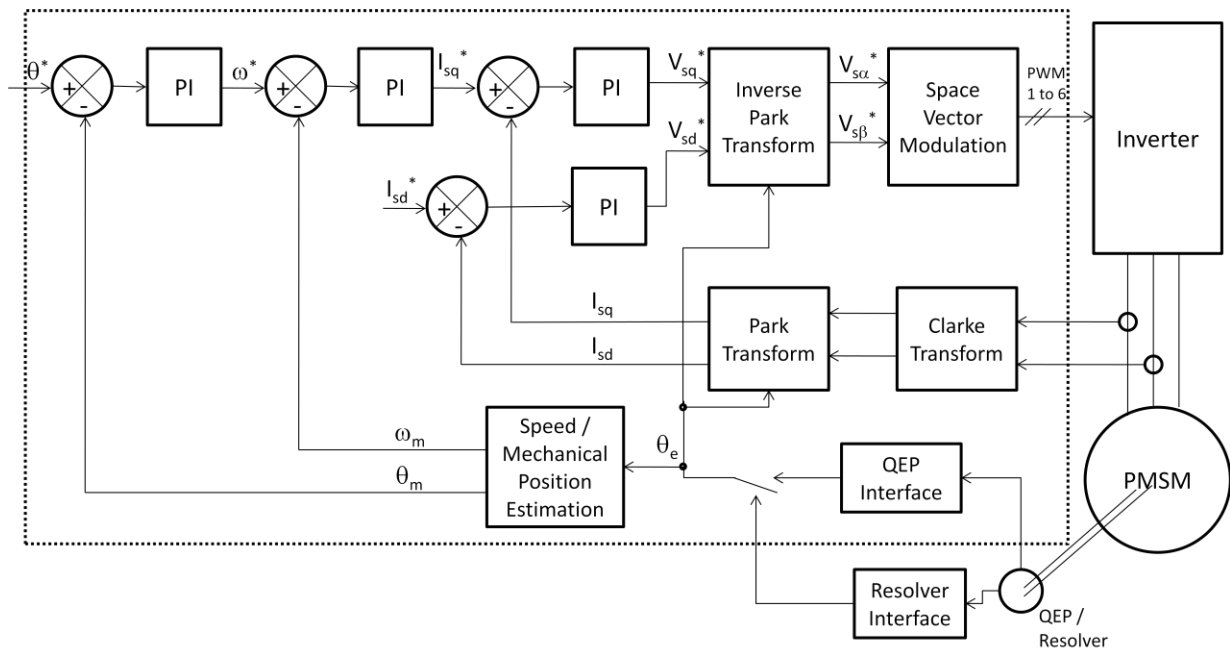


Figure 9 Overall block diagram of Sensored Field Oriented Control

Benefits of 32-bit C2000 Controllers for Digital Motor Control (DMC)

C2000 family of devices possess the desired computation power to execute complex control algorithms along with the right mix of peripherals to interface with the various components of the DMC hardware like the ADC, ePWM, QEP, eCAP etc. These peripherals have all the necessary hooks for implementing systems which meet safety requirements, like the trip zones for PWMs and comparators. Along with this the C2000 ecosystem of software (libraries and application software) and hardware (application kits) help in reducing the time and effort needed to develop a Digital Motor Control solution. The DMC Library provides configurable blocks that can be reused to implement new control strategies. IQMath Library enables easy migration from floating point algorithms to fixed point thus accelerating the development cycle.

Thus, with C2000 family of devices it is easy and quick to implement complex control algorithms (sensored and sensorless) for motor control. 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 range implying right dimensioning of power device circuits
- Using advanced control algorithms, it is possible to reduce torque ripple, thus resulting in lower vibration and longer life time of the motor
- Advanced control algorithms reduce harmonics generated by the inverter thus reducing filter cost.
- Use of sensorless algorithms eliminates the need for speed or position sensor.
- Decreases the number of look-up tables which reduces the amount of memory required
- The Real-time generation of smooth near-optimal reference profiles and move trajectories, results in better-performance
- Generation of high resolution PWM's is possible with the use of ePWM peripheral for controlling the power switching inverters
- Provides single chip control system

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 behaviour.
- 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.

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 & 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. Check the folders below for the source codes and explanations of macro blocks:

- C:\TI\controlSUITE\libs\app_libs\motor_control\math_blocks\v4.0
- C:\TI\controlSUITE\libs\app_libs\motor_control\drivers\f2803x_v2.0

These modules allow users to quickly build, or customize, their 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. At initialization all DMC Library variables are defined and inter-connected. At run-time the macro functions are called in order. Control system is built using an incremental build approach, which allows some sections of the code to be built at a time, 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 form of macros for optimization purposes (refer to application note *SPRAAK2* for more details at TI website). 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 below which means that the code continue in the next line. Any character including invisible ones like “space” 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 the user can easily convert the macro definition to a C functions.

```
#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);
```

A typical DMC macro definition

System Overview

This document describes the “C” real-time control framework used to demonstrate the sensed field oriented control of HVPM motors. The “C” framework is designed to run on C2000 based controllers on Code Composer Studio. The framework uses the following modules¹:

Macro Names	Explanation
CLARKE	Clarke Transformation
PARK / IPARK	Park and Inverse Park Transformation
PI	PI Regulators
PID	PID Regulator
PI_POS	PI Regulator for position loop
RC	Ramp Controller (slew rate limiter)
RG	Ramp / Sawtooth Generator
QEP	QEP Drive
SPEED_FR	Speed Measurement (based on sensor signal frequency)
SVGEN	Space Vector PWM with Quadrature Control (includes IClarke Trans.)
PWM	PWM Drives

¹ Please refer to pdf documents in motor control folder explaining the details and theoretical background of each macro

The overall system implementing sensed Field Oriented Control (FOC) of Permanent Magnet Synchronous Motor (PMSM) is depicted in Figure 10. The control will be experimented with various current sense methods and position sense methods helping the users to explore the impact of feedback methods on system performance. The PM motor is driven by a conventional voltage-source inverter. TMS320F2837x control card is used to generate three sets of complementary pulse width modulation (PWM) signals for the inverter, and, the inverter is built using an integrated power module. Two/three phase currents of PM motor are measured from the inverter using

- shunt sense connected to the bottom of inverter half bridges,
- LEM's flux gate sensor connected in series to the motor phases, and
- shunt measurement, based on Delta-Sigma, connected in series to the motor phases

While ADCs are used for the first two methods, an on-chip SDFM (Sigma Delta Filter Module) is used for the third method. The DC-bus voltage of the inverter is measured using both ADC and SDFM for experimentation purposes. The choice of current sensor depends on the evaluation needs at the customer site. However, the kit provides all three current sense results.

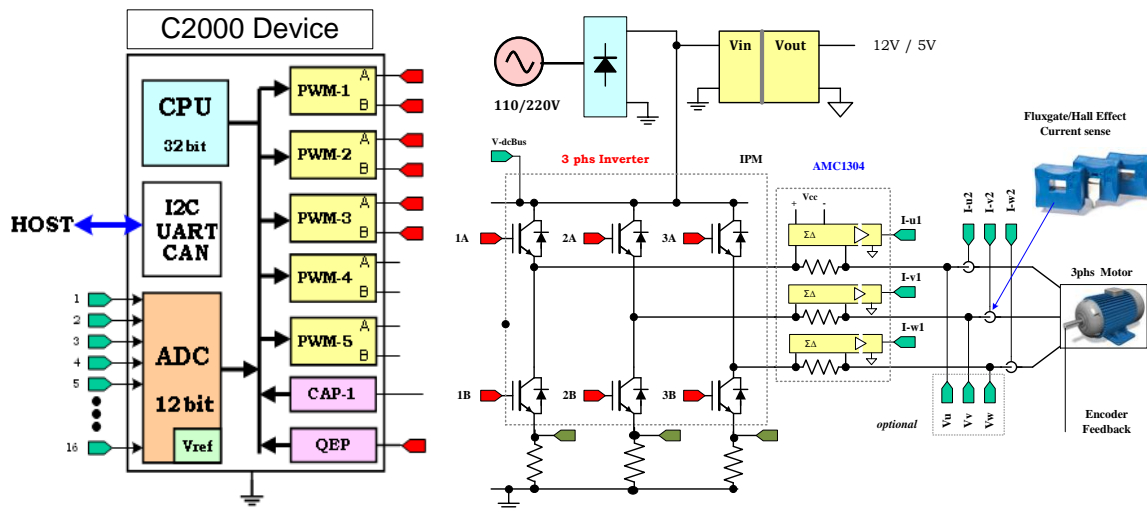


Figure 10 A 3-ph PM motor drive implementation

The software flow is described in the Figure 11 below.

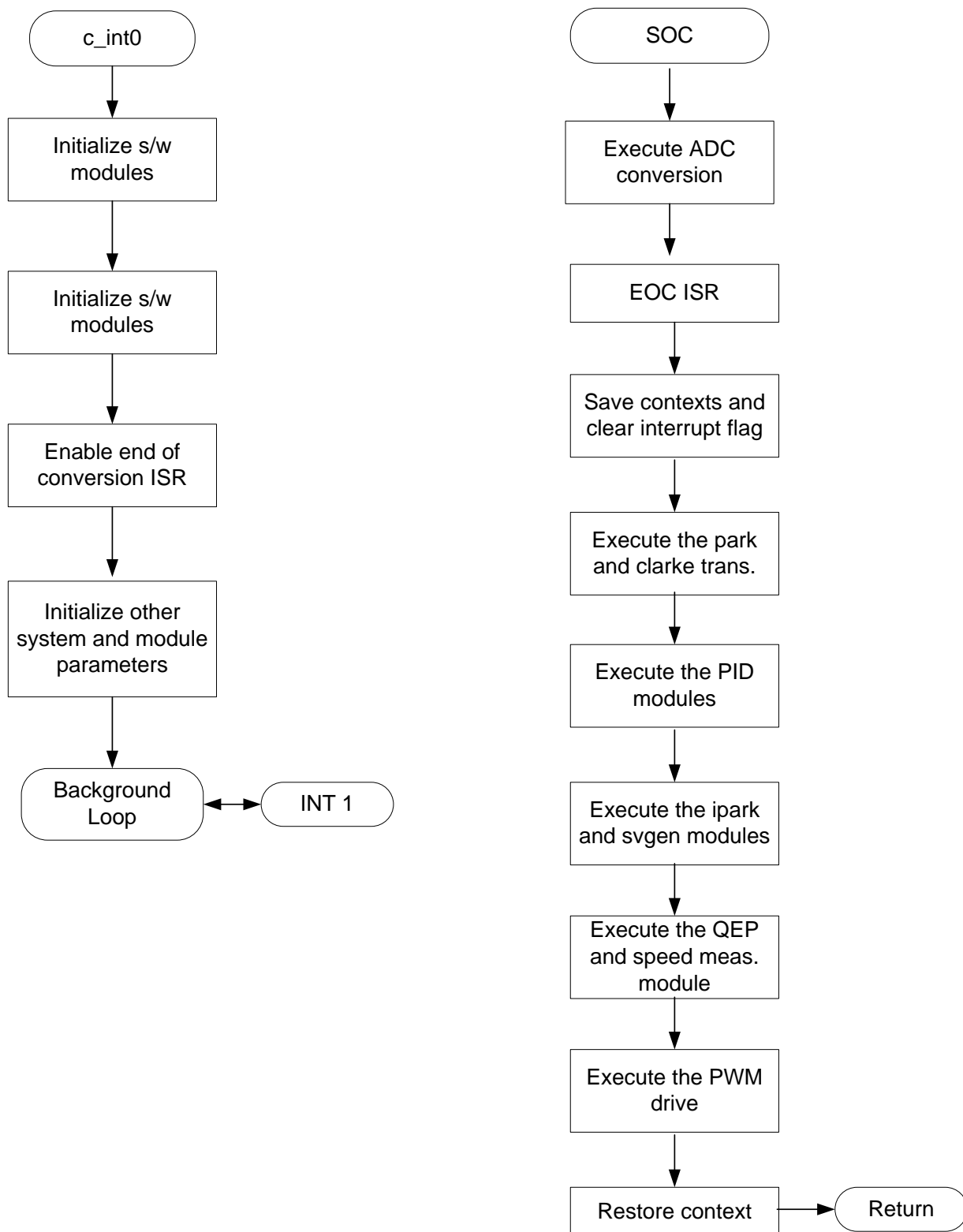


Figure 11 Software flow sequence

Hardware Configuration (IDDK)

For an overview of the kit's hardware and steps on how to setup this kit, please refer to the IDDK Hardware Manual and IDDK User's Guide at

C:\TI\controlSUITE\development_kits\TMDSIDDK_v1.0\~Docs

For immediate reference about powering the board, the picture below shows how to power it from an external DC supply. Always keep the power supply OFF or at 0V while making connections.

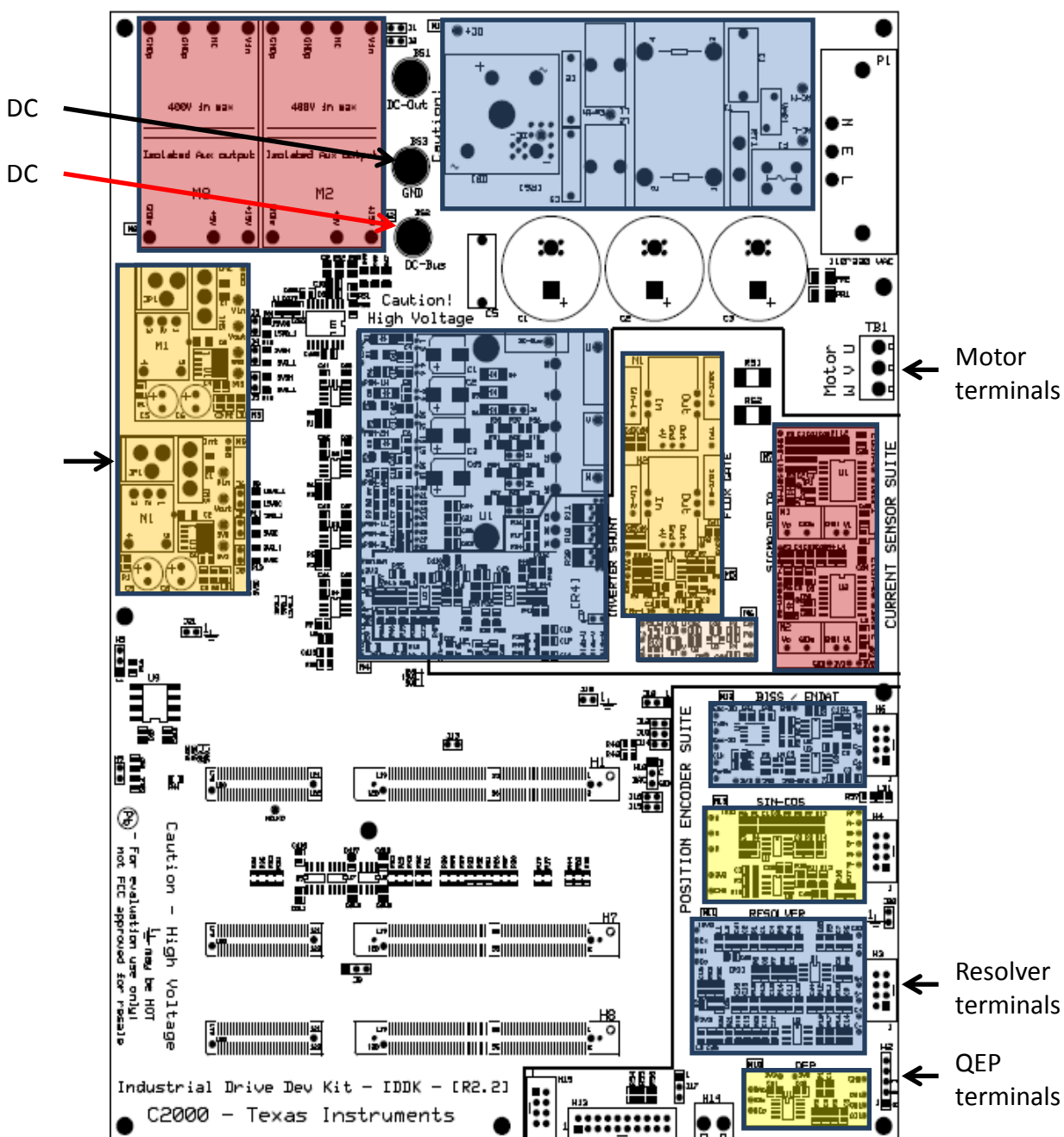


Figure 12 Powering IDDK from external DC power supply



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

The picture below shows powering up the board from AC mains, remember to connect BS1 and BS3 with a banana jumper to feed the rectifier output to DC bus as shown by the thick red line.

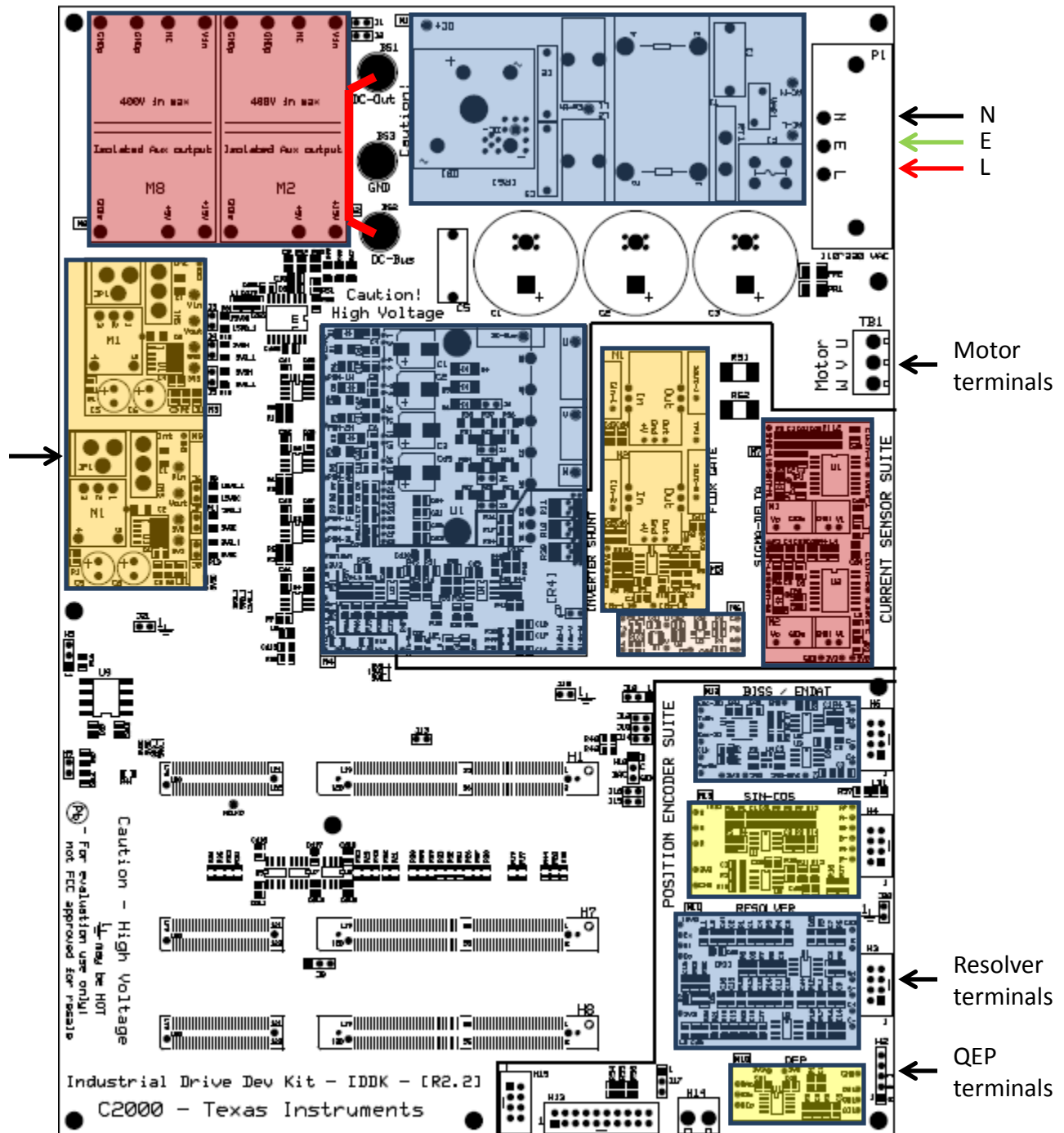


Figure 13 Powering IDDK from an AC source



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

During development, it is preferable to use an isolation transformer for equipment and personnel safety as shown in the picture below

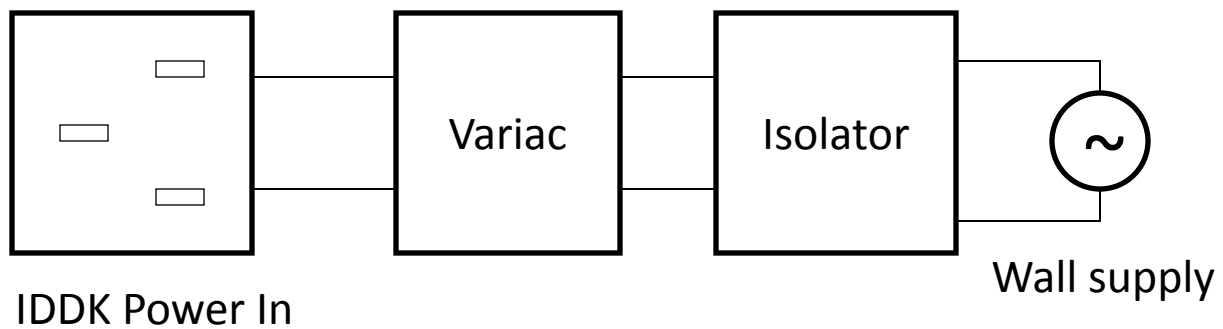


Figure 14 AC power connection to IDDK through an isolation transformer



CAUTION: Depending on the choice of power GND and control GND plane interconnections, even the control GND may be HOT

Proceed with caution!

Software Setup Instructions to Run HVPM_Sensored Project

Please refer to the IDDK_ User's Guide at

C:\TI\controlSUITE\development_kits\TMDSIDDK_v1.0\~Docs


Open CCS and load the IDDK project by browsing to

C:\TI\controlSUITE\development_kits\TMDSIDDK_v1.0\IDDK_PM_Servo_F2837x

Select the active build configuration to be set as F2837x_RAM. Verify that the build level is set to 1, and then right click on the project name and select "Rebuild Project". Once build completes, launch a debug session and load the code into CPU1 of the controller. Add variables to the expressions window by 'Right Clicking' within the Expressions Window and 'Importing' the file 'Variables_IDDK_Level1.txt' from root directory, and find the Expressions Window as shown in the table below.

<div> <div>(x)= Variables</div> <div>Expressions</div> <div>Registers</div> <div>Breakpoints</div> </div>				
Expression	Type	Value	Address	
(x)= EnableFlag	unsigned int	1	0x0000B006@Data	
(x)= IsrTicker	unsigned long	672403	0x0000B036@Data	
(x)= SpeedRef	float	0.05	0x0000B03E@Data	
(x)= rc1.SetpointValue	float	0.04998922	0x0000B12A@Data	
(x)= rg1.Out	float	0.1622219	0x0000B094@Data	
(x)= VdTesting	float	0.0	0x0000B02A@Data	
(x)= VqTesting	float	0.1	0x0000B016@Data	
(x)= svgen1.Ta	float	-0.08920144	0x0000B156@Data	
(x)= rslvrIn.TUNING	unsigned int	0	0x0000B9EE@Data	
(x)= testCntMax	long	1000	0x0000B9CE@Data	
(x)= testAngleMax	float	0.8	0x0000B9CC@Data	
(x)= offsetFc	float	79.57747	0x0000B9DE@Data	
(x)= errorFc	float	1000.0	0x0000B9EA@Data	
(x)= piconFz	float	200.0	0x0000B9CA@Data	
(x)= rslvrKp	float	5000.0	0x0000B9E4@Data	
(x)= rslvrOut.angleRaw	float	0.8275713	0x0000BA40@Data	
(x)= rslvrOut.angleObs	float	0.8243439	0x0000BA42@Data	
+ Add new expression				

Table 1 Expressions Window Variables

Setup time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location C:\TI\ControlSUITE\development_kits\TMDSIDDK_v1.0\IDDK_PM_Servo_F2837x\l. Click on Continuous Refresh button  on the top left corner of the graph tab to enable periodic capture of data from the microcontroller.

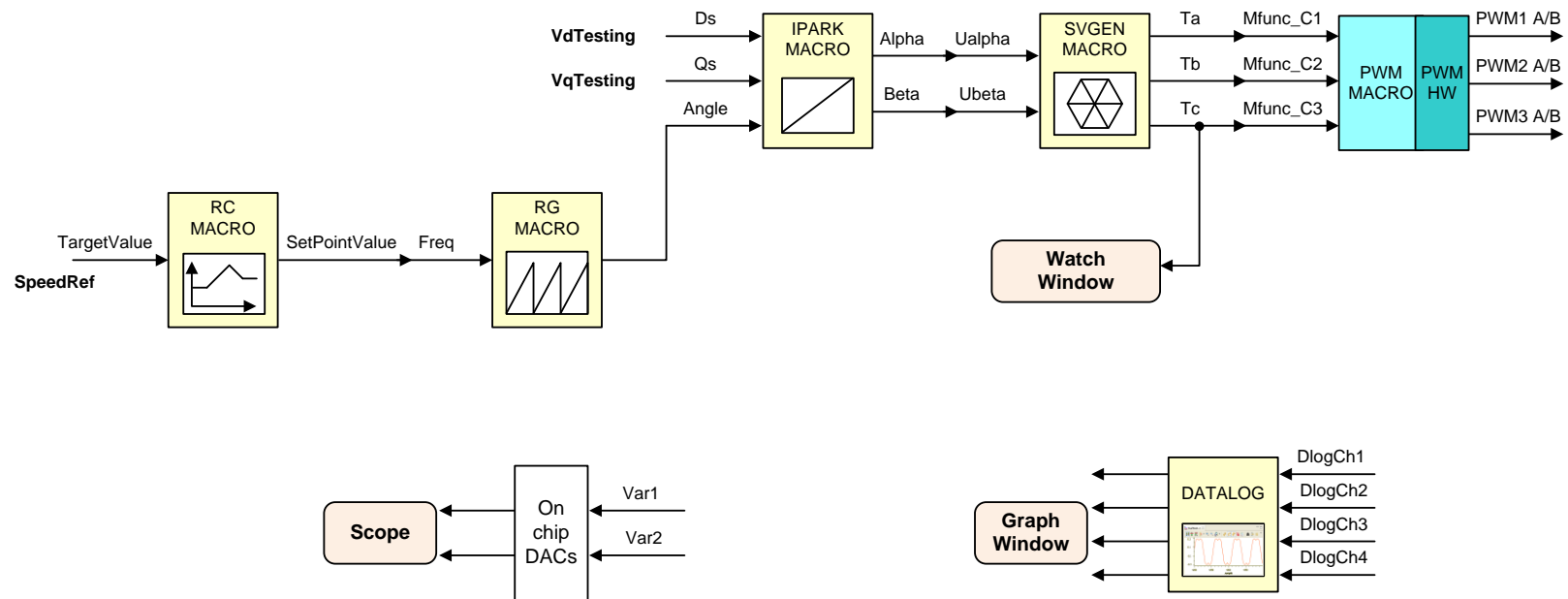
Incremental System Build

The system is gradually built up in order for 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. Most modules are written as software MACROs, and the remaining are written as callable functions. Table 1 summarizes the modules testing and using in each incremental system build.

Software Module	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
RC_MACRO	√√	√	√	√	√
RG_MACRO	√√	√	√	√	√
IPARK_MACRO	√√	√	√	√	√
SVGEN_MACRO	√√	√	√	√	√
PWM_MACRO	√√	√	√	√	√
CLARKE_MACRO		√√	√	√	√
PARK_MACRO		√√	√	√	√
CurrentSensorSuite()		√√	√	√	√
PosEncoderSuite()		√√	√	√	√
SPEED_FR_MACRO		√√	√	√	√
PI_MACRO (IQ)			√√	√	√
PI_MACRO (ID)			√√	√	√
PID_MACRO (SPD)				√√	√
PI_POS_MACRO (POS)					√√
Note: the symbol √ means this module is using and the symbol √√ means this module is testing in this phase.					

Table 2 Testing modules in each incremental system build

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.

Level 1 Incremental Build

The block diagram of the system built in BUILDLEVEL 1 is shown in the previous page. At this step keep the motor disconnected. Assuming the load and build steps described in the “IDDK User’s Guide” completed successfully, this section describes the steps for a “minimum” system check-out which confirms operation of system interrupt, the peripheral & 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. Open IDDK_PM_Servo_F2837x-Settings.h and select level 1 incremental build option by setting the BUILDLEVEL to LEVEL1 (#define BUILDLEVEL LEVEL1). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. If not already done, add variables to the expressions window by ‘Right Clicking’ within the Expressions Window and ‘Importing’ the file ‘Variables_IDDK_Level1.txt’ from root directory. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in watch windows to confirm the interrupt working properly.

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

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

Level 1A (SVGEN_MACRO Test)

The SpeedRef value is specified to the RG_MACRO module via 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.

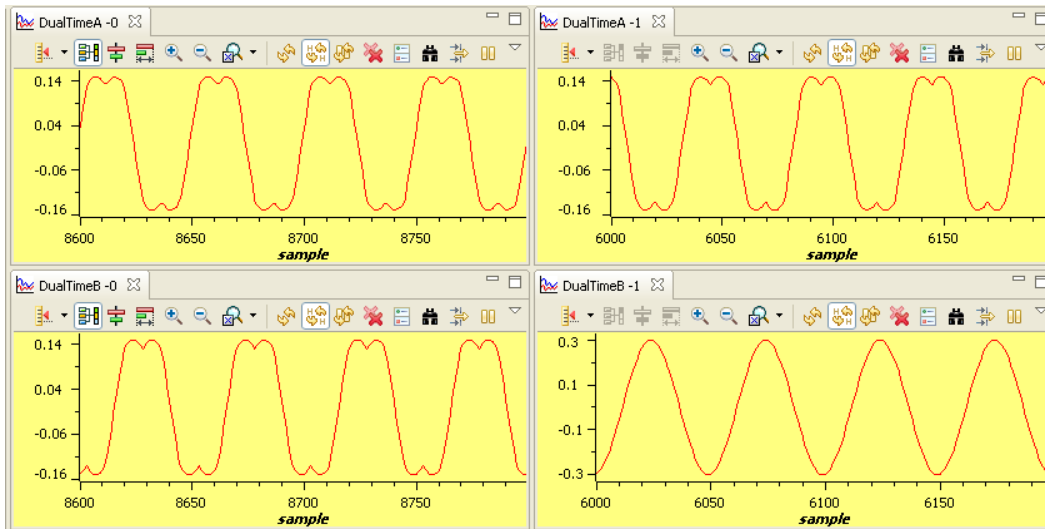


Figure 15 Output of SVGEN, Ta, Tb, Tc and Tb-Tc waveforms

Level 1B (testing the DACs)

To monitor internal signal values in real time, onchip DACs are used. DACs are part of the analog module. DACs B and C are available for this purpose.

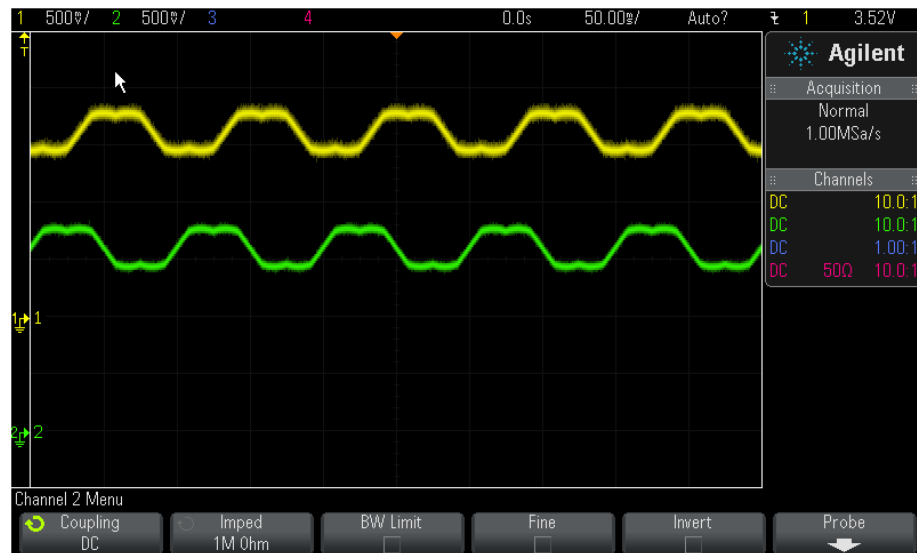


Figure 16 DAC 1-4 outputs showing Ta, Tb waveforms

Level 1C (PWM_MACRO and INVERTER Test)

After verifying 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 15V to 20V you would start observing the Inverter phase voltage dividers and waveform monitoring filters (Vfb-U, Vfb-V, Vfb-W) enable the generation of the waveform and ensures theta the inverter is working appropriately. Note that the default RC values are optimized for future use of AC motor state observers employing phase voltages.



After verifying this, reduce the DC Bus voltage, take the controller out of real time mode (disable), reset the processor (see "IDDK User's Guide" for details). 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, hence caution need to be taken while doing these experiments

Level 1D Tuning Resolver Loop Parameters

Information related to tuning the resolver parameters and resolver, in general, are given in detail in the location

C:\TI\controlSUITE\development_kits\TMDSRSLVR_v1.0\~Docs

In this test, the user can fine tune the PI controller parameters of the resolver loop to verify if the transient performance is satisfactory. This step is optional.

In the expressions window, set the variable 'RslvrIn.TUNING' to 1. This changes the DAC variables to 'rslvrOut.angleRaw' and 'rslvrOut.angleObs' and makes the setting 'rslvrIn.FIR32' irrelevant. Internally, the software generates a square wave angle reference varying between 0 and 150deg (approximate) for the loop. This value can also be varied for experimenting.

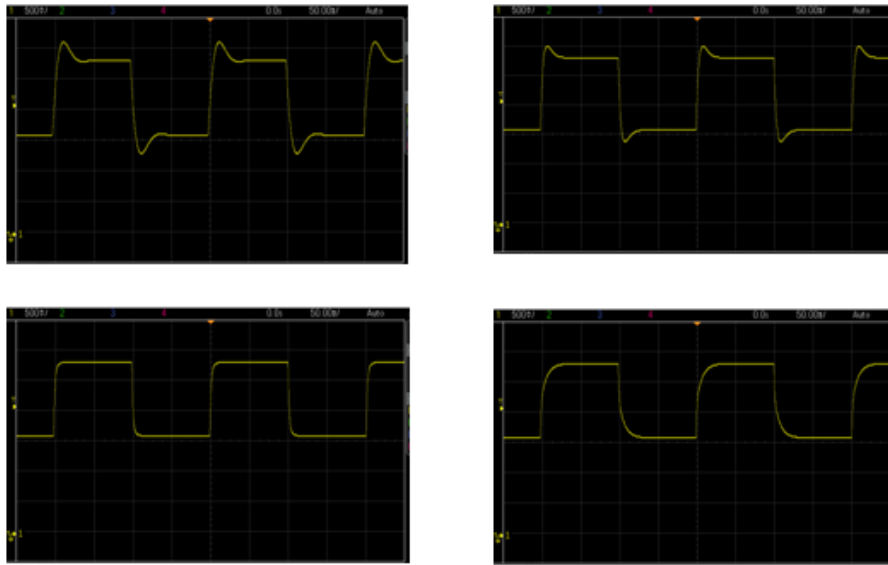
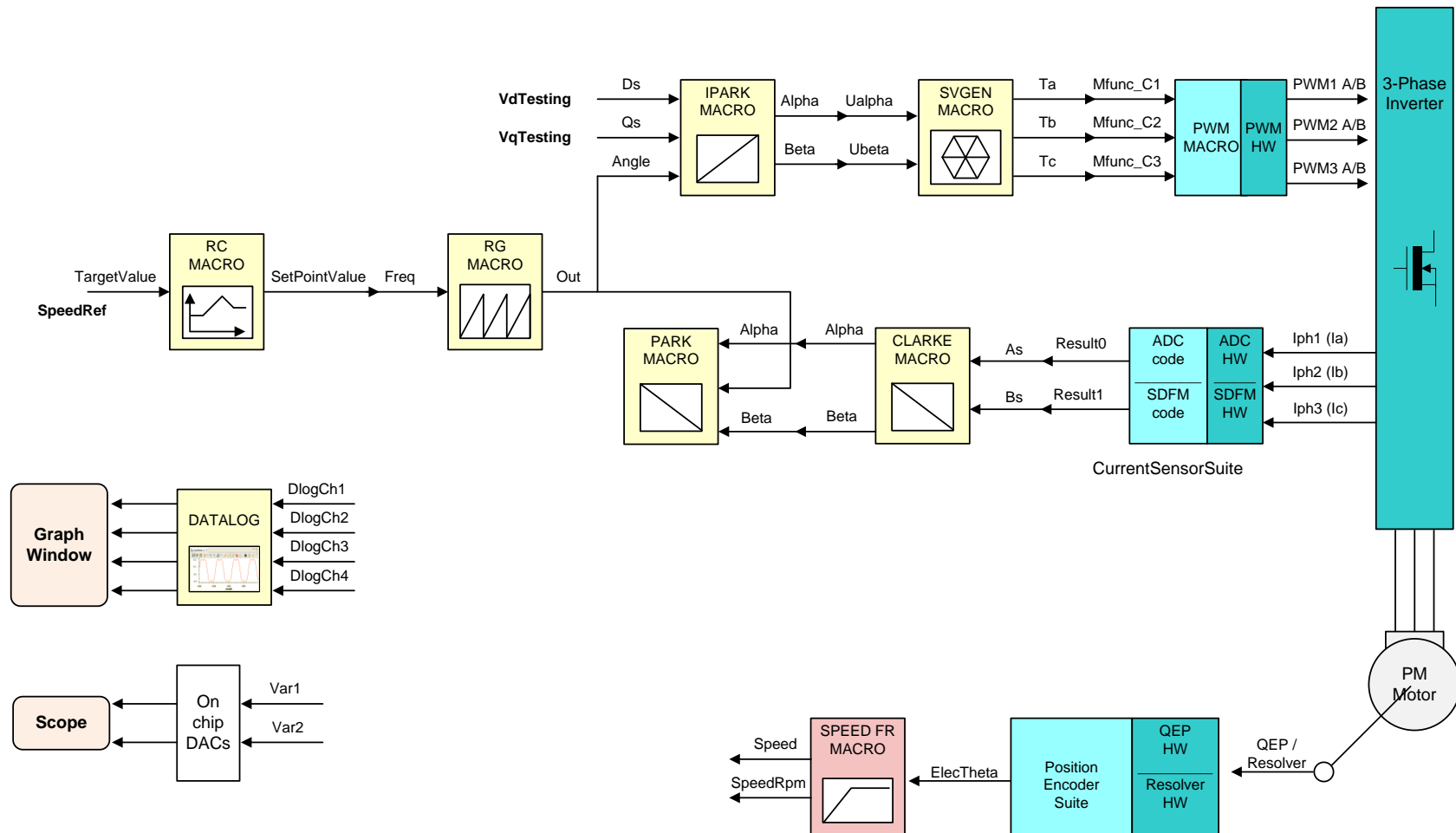


Figure 17 Observer angle response

All relevant variables needed for tuning the resolver loop are brought out in the Expressions Window already during import. Tune filters' corner frequencies based on noise considerations. Then, adjust PI coefficients and view the results on a scope by probing DAC outputs B and C. When a satisfactory response is obtained, note down the values chosen for different parameters and modify the code to initialize them with these values from the next build onwards. At this point, set RslvrIn.TUNING to 0 to run the loop using sin / cos based angle estimation. Figure 17 shows 'rslvrOut.angleObs' for increasing values of K_p , while test reference angle toggles between 0 and 150 deg.

Level 2 - Incremental System Build Block Diagram



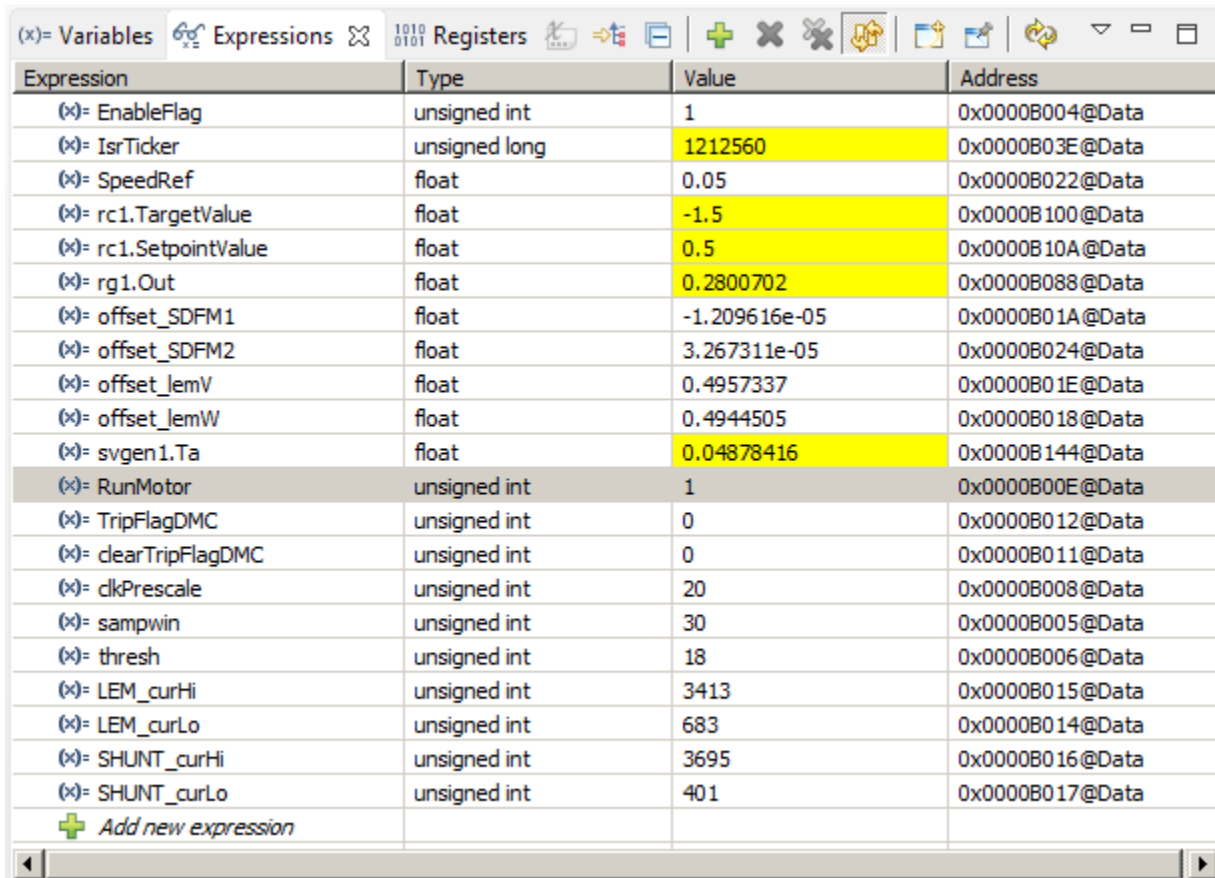
Level 2 verifies the ADCs, offset compensation, SDFM, clarke / park transformations and position sensor interfaces.

Level 2 Incremental Build

The block diagram of the system built in BUILDLEVEL 2 is shown in the previous page. In this section, some more blocks are added to level 1 and tested. Assuming BUILD 1 is completed successfully, this section verifies the over current protection limits of the inverter, analog-to-digital conversion, Delta-Sigma Filter Module (SDFM), Clarke / Park transformations. In this Build, the motor is run in open loop to verify the functionality of various current sense options using the SHUNT, LEM or SDFM methods, and also the functionality of position encoder (QEP or resolver) used in the set up.

The motor can be connected to HVDMC board since the PWM signals are successfully proven through level 1 incremental build. Note that the open loop experiment is meant to test the various feedback modules. Therefore running motor under load or at various operating points is not recommended.

Open IDDK_PM_Servo_F2837x-Settings.h and select level 2 incremental build option by setting the **BUILDLEVEL to LEVEL2** (#define BUILDLEVEL LEVEL2). Select **CURRENT_SENSE to SHUNT_CURRENT_SENSE** and **POSITION_ENCODER to QEP_POS_ENCODER** if the user encoder is QEP or **POSITION_ENCODER to RESOLVER_POS_ENCODER** if the user encoder is a resolver. Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Import variables from 'Variables_IDDK_Level2.txt' file in the root directory and the expressions window will look as shown in Figure 18 below.



Expression	Type	Value	Address
(x)= EnableFlag	unsigned int	1	0x0000B004@Data
(x)= IsrTicker	unsigned long	1212560	0x0000B03E@Data
(x)= SpeedRef	float	0.05	0x0000B022@Data
(x)= rc1.TargetValue	float	-1.5	0x0000B100@Data
(x)= rc1.SetpointValue	float	0.5	0x0000B10A@Data
(x)= rg1.Out	float	0.2800702	0x0000B088@Data
(x)= offset_SDFM1	float	-1.209616e-05	0x0000B01A@Data
(x)= offset_SDFM2	float	3.267311e-05	0x0000B024@Data
(x)= offset_lemV	float	0.4957337	0x0000B01E@Data
(x)= offset_lemW	float	0.4944505	0x0000B018@Data
(x)= svgen1.Ta	float	0.04878416	0x0000B144@Data
(x)= RunMotor	unsigned int	1	0x0000B00E@Data
(x)= TripFlagDMC	unsigned int	0	0x0000B012@Data
(x)= clearTripFlagDMC	unsigned int	0	0x0000B011@Data
(x)= clkPrescale	unsigned int	20	0x0000B008@Data
(x)= sampwin	unsigned int	30	0x0000B005@Data
(x)= thresh	unsigned int	18	0x0000B006@Data
(x)= LEM_curHi	unsigned int	3413	0x0000B015@Data
(x)= LEM_curLo	unsigned int	683	0x0000B014@Data
(x)= SHUNT_curHi	unsigned int	3695	0x0000B016@Data
(x)= SHUNT_curLo	unsigned int	401	0x0000B017@Data
+ Add new expression			

Figure 18 Expressions window for build level 2

Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in Expression window to confirm the interrupt working properly. Now set the variable named “RunMotor” to 1 and the motor will start spinning after a few seconds if enough voltage is applied to the DC-Bus.

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

- SpeedRef : for changing the rotor speed in per-unit.
- VdTesting : for changing the d-qxis voltage in per-unit.
- VqTesting : 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 B_{emf} is lower than the average voltage applied to motor winding. This will prevent the motor from stalling or vibrating.

Phase 2A – Setting over current limit in software

The board has various current sense methods, such as shunt, LEM and SDFM. Over current monitoring is provided for signals generated from shunt and LEM using on-chip **Comparator Sub System (CMPSS) module**. It has a programmable comparator and a programmable digital filter. Obviously, the comparator generates the protection signal. The reference to the comparator is user programmable for both positive and negative currents. The digital filter module qualifies the comparator output signal by verifying its sanity by periodically verifying the genuineness of the signal for a certain count times within a certain count window, where the periodicity, count and count window are user programmable.

In the Epressions 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’,
- ‘LEM_curHi’ sets the positive current max through LEM current sensor
- ‘LEM_curLo’ sets the negative current max through LEM current sensor
- ‘SHUNT_curHi’ sets the positive current max through SHUNT current sensor
- ‘SHUNT_curLo’ sets the negative current max through SHUNT current sensor

NOTE:- Median value corresponding to zero current is 2048

‘TripFlagDMC’ is a flag variable used to represent the over current trip status of the inverter. If this flag is set, then the user can adjust the above 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 8A. The user can fine tune any of these settings to suit their system. Once satisfactory values are identified, note them down and modify the code with these new values and rebuild/ load for further tests.

It is possible to shut down the inverter using a digital signal from an external source through H9. No code is provided right now, but the user can take it as an exercise to experiment and learn.

Phase 2B – Testing the Clarke module

In this part the Clarke module will be 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 graph window.

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

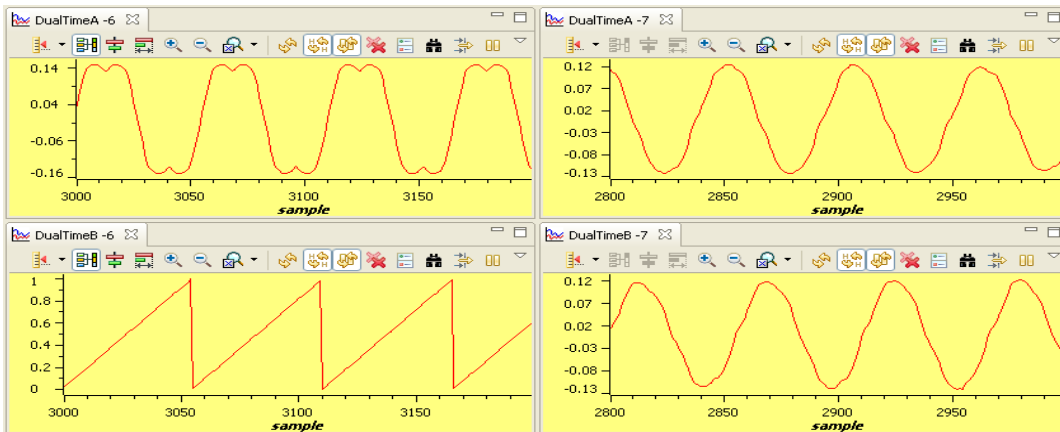


Figure 19 The waveforms of Svgen_dq1.Ta, rg1.Out, and phase A&B currents*

* Deadband = 0.83 usec, Vdcbus=300V , dlog.prescalar=3

Since the low side current measurement technique is used employing shunt resistors on inverter phase legs, the phase current waveforms observed from current test points ([M5]-lfb-U, and [M5]-lfb-V) are composed of pulses as shown in Figure 20.

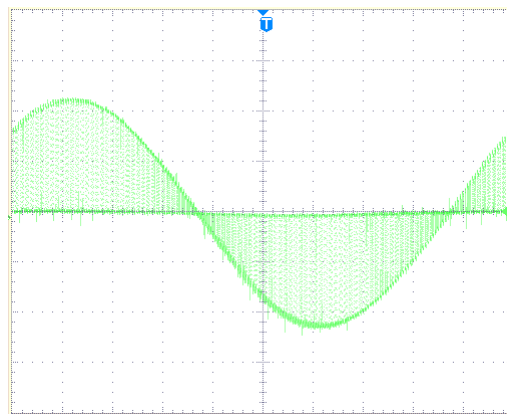


Figure 20 Amplified Phase A current

Level 2C – Adjusting PI Limits

Note that the vectorial sum of d-q PI outputs should be less than 1.0 which refers to maximum duty cycle for SVGEN macro. Another duty cycle limiting factor is the current sense through shunt resistors which depends on hardware/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 will automatically generate the voltage reference and determine the PWM duty cycle depending on the d-q current demand, therefore set pi_id.Umax/min and pi_iq.Umax/min according to recorded VdTesting and VqTesting values respectively.

Running 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.

Level 2D – Various Current Sense Methods

Repeat this BUILD again after changing the CURRENT_SENSE to **LEM_CURRENT_SENSE** and review the performance. After this test, repeat this BUILD again changing the CURRENT_SENSE to **SD_CURRENT_SENSE**. The software module gets the current feedback from various sense methods and makes it available for the user to pick the one of his choice to close the current loop. The user may choose the current sense method that will be eventually used in their development.

Level 2E – Position Encoder Feedback / SPEED_FR test

During all the above tests, the position encoder interface was continuously estimating position information and so no new code is needed to verify the position encoder interface. When the motor is commanded to run, it is taken through an initial alignment stage where the electrical angle and the QEP angle count are set to zero. If a resolver is used, its initial position at electrical angle zero is identified for run time corrections. Estimated position information is made available on DAC-c, while the reference position (rg1.Out) used to perform open loop motor control is displayed on DAC-b. These signals are brought out on H10 on IDDK, and their scope plots are given below in Figure 21.

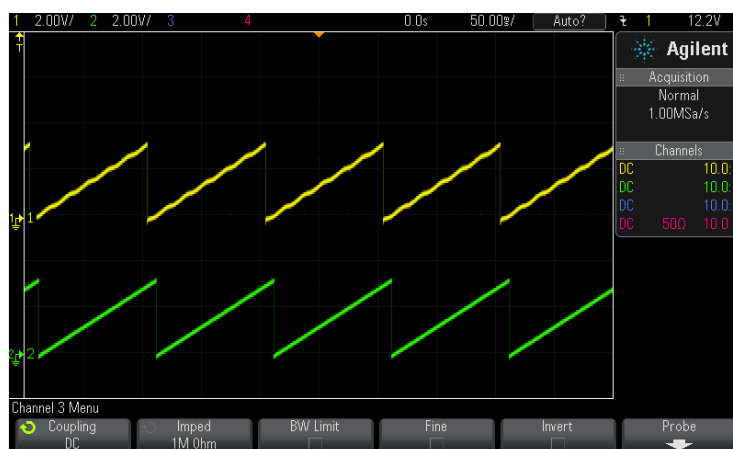


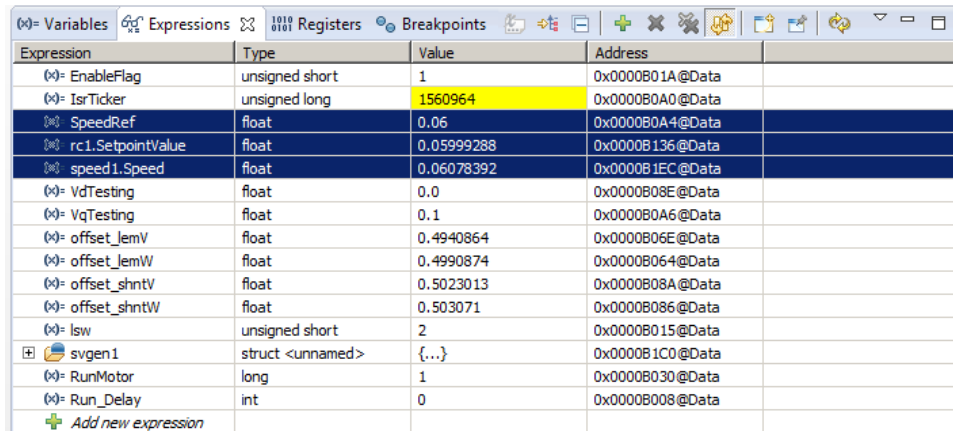
Figure 21 Scope plot of reference angle and rotor position

The waveform of channel 2 represents the reference position, while channel 1 represents the estimated position. The ripple in position estimate is indicative of the fact that the motor runs with some minor

speed oscillation. Because of open loop control, the rotor position and reference position may not align. However, it is important to make sure that the sense of change of estimated angle should be same as that of the reference; otherwise it indicates that the motor has a reverse sense of rotation. This can be fixed either by swapping any two wires connecting to the motor or in software by reversing the angle estimate as in the pseudo code

$$\text{angle} = 1.0 - \text{angle}$$

To make sure that the SPEED_MACRO works fine, change the 'SpeedRef' variable in Expressions Window as shown in Figure 22 and check if the estimated speed variable 'speed1.Speed' follows the commanded speed. Since the motor is a PM motor, where there is no slip, the running speed will follow the commanded speed regardless of the control being open loop.



Expression	Type	Value	Address
(x) EnableFlag	unsigned short	1	0x0000B01A@Data
(x) IsrTicker	unsigned long	1560964	0x0000B0A0@Data
(x) SpeedRef	float	0.06	0x0000B0A4@Data
(x) rc1.SetpointValue	float	0.05999288	0x0000B136@Data
(x) speed1.Speed	float	0.06078392	0x0000B1EC@Data
(x) VdTesting	float	0.0	0x0000B08E@Data
(x) VqTesting	float	0.1	0x0000B0A6@Data
(x) offset_lemV	float	0.4940864	0x0000B06E@Data
(x) offset_lemW	float	0.4990874	0x0000B064@Data
(x) offset_shntV	float	0.5023013	0x0000B08A@Data
(x) offset_shntW	float	0.503071	0x0000B086@Data
(x) lsw	unsigned short	2	0x0000B015@Data
(x) svgen1	struct <unnamed>	{...}	0x0000B1C0@Data
(x) RunMotor	long	1	0x0000B030@Data
(x) Run_Delay	int	0	0x0000B008@Data

Figure 22 Expressions window

With a QEP:

- Watch out for 'Qep1.CalibratedAngle' in the Expressions window. It represents the electrical angle (or) position of rotor at the event of Index pulse. It can vary depending on starting position of rotor. When a closed loop operation is performed, it becomes important to ascertain the angular offset between electrical zero and QEP's index pulse that would reset the QEP counter to zero. The calibration angle can be formulated as follows:

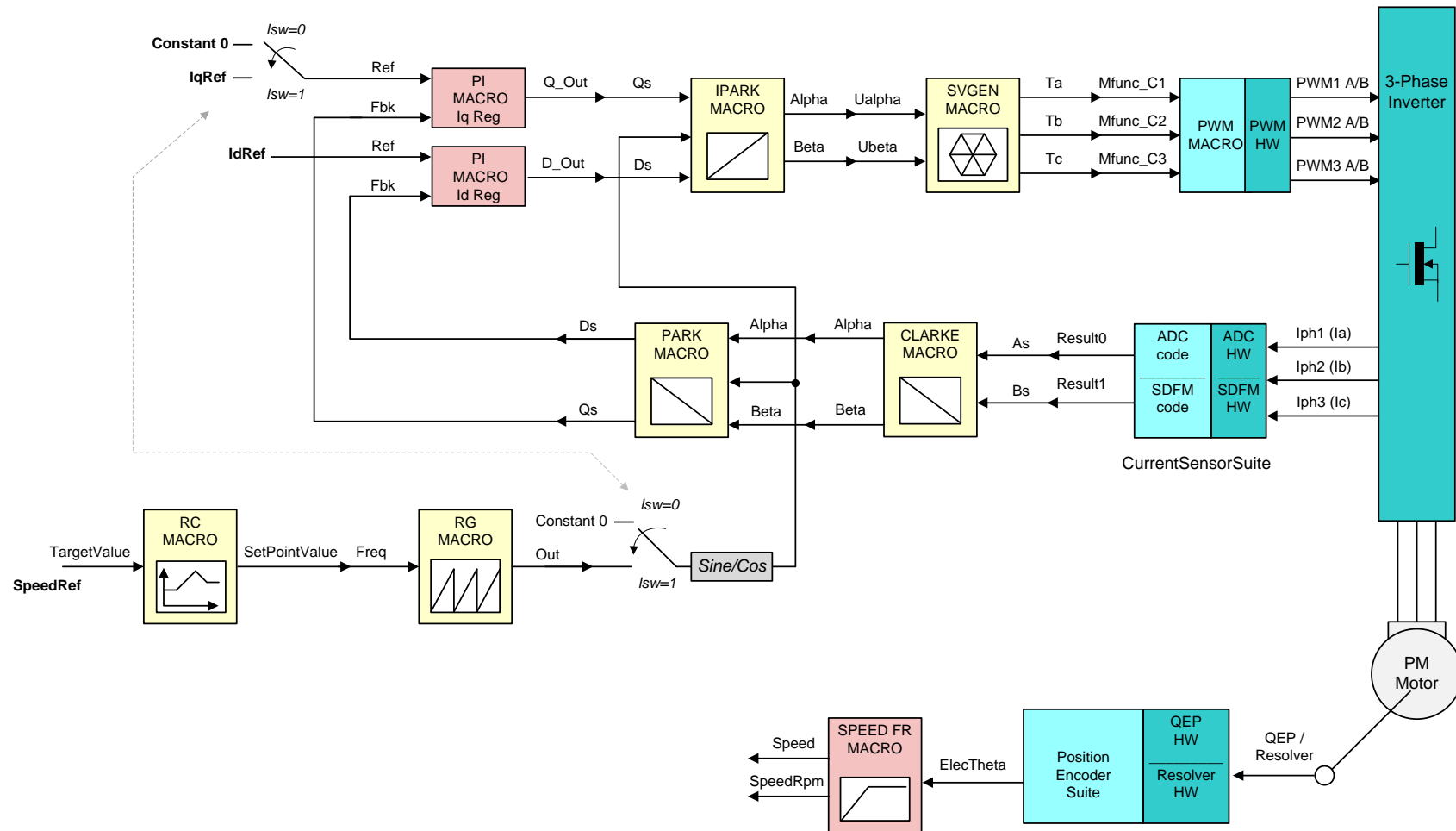
$$\text{Calibration Angle} = \text{Offset Angle} \pm n \cdot \text{Line Encoder}$$

With a RESOLVER:

- Watch out for 'resolver1.InitTheta' in the Expressions window. It represents the resolver angle at the time of starting. Again, it can vary depending on initial position of rotor. For closed loop operation, it represents the angular offset between resolver and the motor's electrical zero.

After the tests are done, 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. Now the motor is stopping.

Level 3 - Incremental System Build Block Diagram



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

Level 3 Incremental Build

Assuming the previous section is completed successfully, this section verifies the dq-axis current regulation performed by PI modules. To confirm the operation of current regulation, the gains of these two PI controllers are necessarily tuned for proper operation. In this build, transformations are done based on the reference angle generated manually rather than the actual rotor position. This is to ensure that this test can be done without a big load on the motor; otherwise Iq loop testing cannot be done easily. When the motor is commanded to run, it is taken through an initial alignment stage where the electrical angle and the QEP angle count are set to zero. If a resolver is used, its initial position at electrical angle zero is identified for run time corrections.

Open IDDK_PM_Servo_F2837x-Settings.h and select level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3). The user is free to pick any of the three supported CURRENT_SENSE methods. Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in watch windows to confirm the interrupt working properly. In the software, the key variables to be adjusted are summarized below.

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

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

The steps are explained as follows:

- Compile/load/run program with real time mode.
- 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 (or another suitable value).
- Add variables 'pi_id.Fbk', 'pi_id.Kp' and 'pi_id.Ki' and corresponding elements for 'pi_iq' to the expressions window
- Gradually increase voltage at variac / dc power supply to, say, 20% of the rated voltage.
- Set 'RunMotor' flag to 1
- Check pi_id.fbk in the watch windows with continuous refresh feature whether or not it should be keeping track IdRef for PI module. If not, adjust its PI gains properly.
- Check pi_iq.fbk in the watch windows with continuous refresh feature whether or not it should be keeping track IqRef for PI module. If not, adjust its PI gains properly.
- To confirm these two PI modules, try different values of pi_id.Ref and pi_iq.Ref or SpeedRef.
- 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 realtime mode and reset. Now the motor is stopping.

During running this build, the current waveforms in the CCS graphs should appear as follows*:

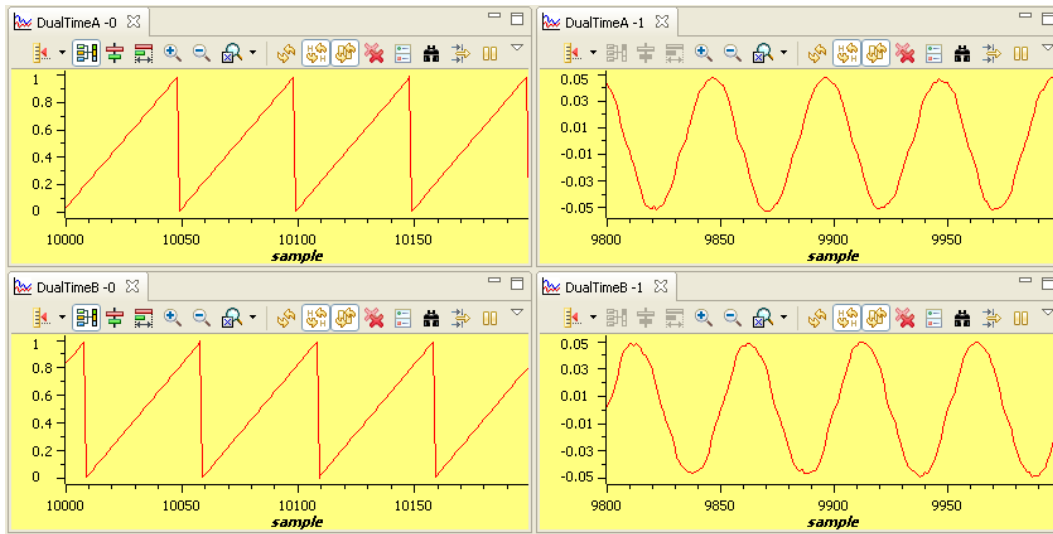
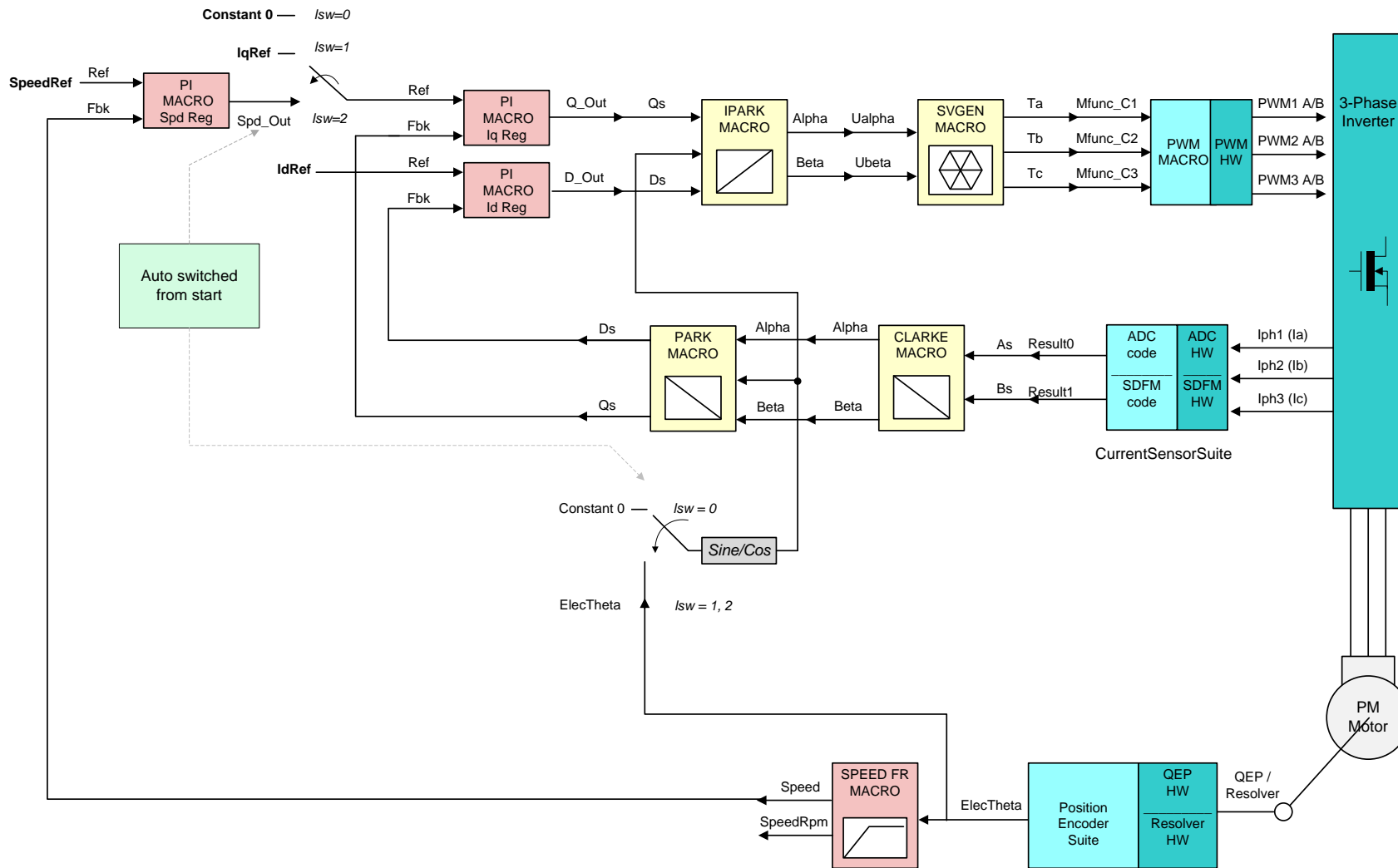


Figure 23 Measured theta, rg1.out and Phase A & B current waveforms

* Deadband = 0.83 usec, Vdcbus=300V, dlog.trig_value=100

Level 4 - Incremental System Build Block Diagram



Level 4 verifies the speed PI module and speed loop

Level 4 Incremental Build

Assuming the previous section is completed successfully; this section verifies the speed PI module and speed loop. All transformations are done based on the actual rotor position. When the motor is commanded to run, it is taken through an initial alignment stage where the electrical angle and the QEP angle count are set to zero. If a resolver is used, its initial position at electrical angle zero is identified for run time corrections.

Open HVPM_Sensored-Settings.h and select level 4 incremental build option by setting the BUILDLEVEL to LEVEL4 (#define BUILDLEVEL LEVEL4). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in watch windows to confirm the interrupt working properly. In the software, the key variables to be adjusted are summarized below.

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

The key steps can be explained as follows:

- Set Compile/load/run program with real time mode.
- Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
- Add 'pid_spd' variable to the expressions window
- Gradually increase voltage at variac to get an appropriate DC-bus voltage and now the motor is running with this reference speed (0.3 pu).
- Add the switch variable "RunMotor" to watch window in order to start the motor. The soft-switch variable (lsw) is auto promoted in a sequence. In the code lsw manages the loop setting as follows:
 - lsw=0, lock the rotor of the motor
 - lsw=1, for QEP feedback only – motor in run mode and waiting for first instance of QEP Index pulse
 - lsw=2, motor in run mode, both QEP and resolver (for QEP - first Index pulse occurred)
- Set 'RunMotor' to 1. Compare Speed with SpeedRef in the watch windows with continuous refresh feature whether or not it should be nearly the same.
- To confirm this speed PID module, try different values of SpeedRef (positive or negative). For speed PID controller, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied response.
- At very low speed range, the performance of speed response relies heavily on the good rotor position angle provided by QEP encoder.
- 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. Now the motor is stopping.

During running this build, the current waveforms in the CCS graphs should appear as follows* :

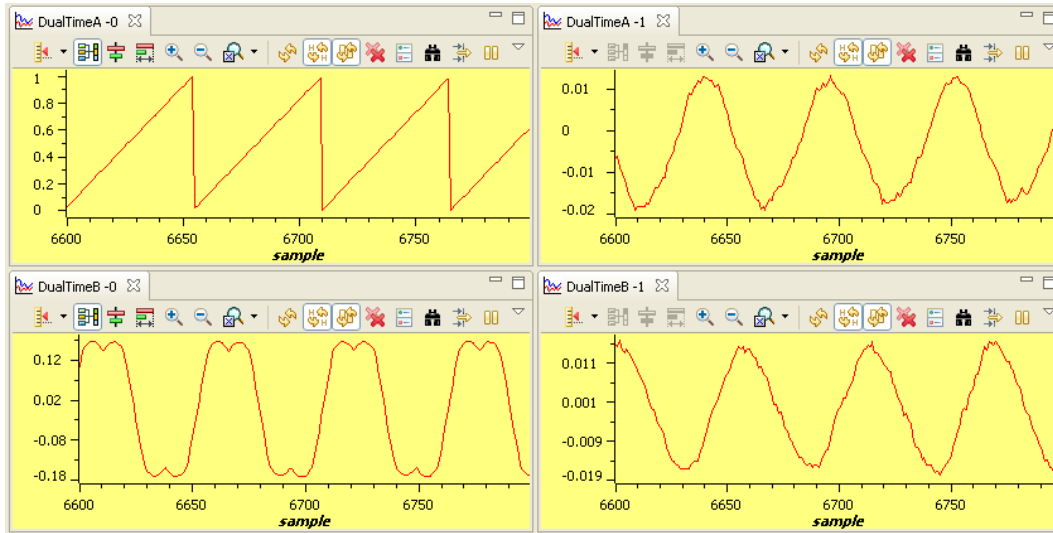


Figure 24 Measured theta, svgen duty cycle, and Phase A&B current waveforms under no-load & 0.3 pu speed

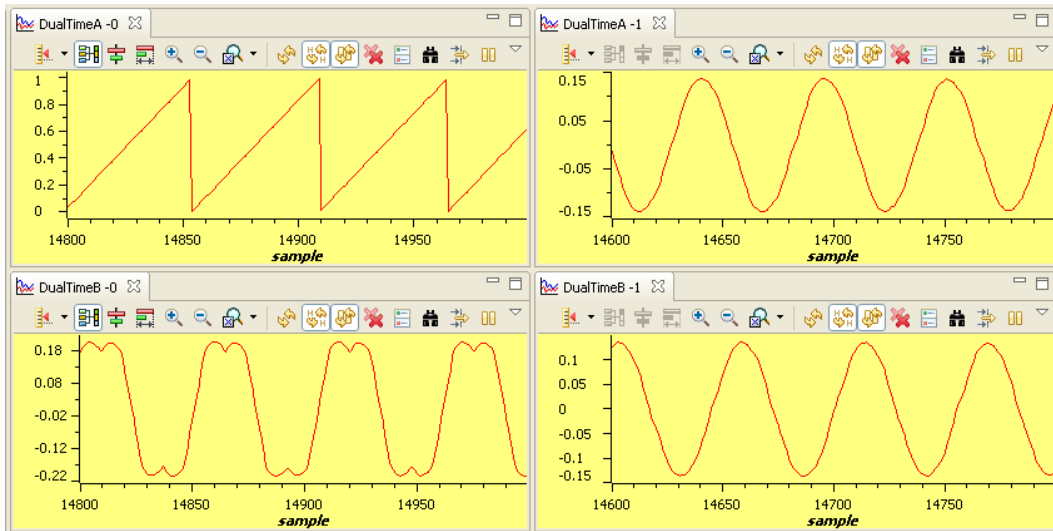


Figure 25 Measured theta, svgen duty cycle, and Phase A&B current waveforms under 0.33pu load & 0.3 pu speed

* Deadband = 0.83 usec, dlog.trig_value=100, Vdcbus=300V

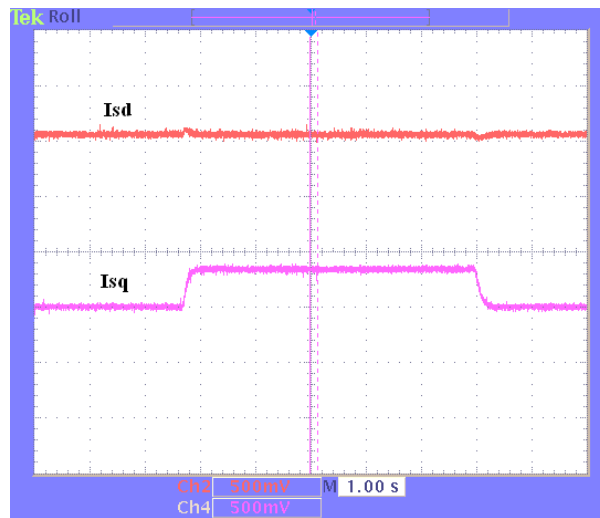
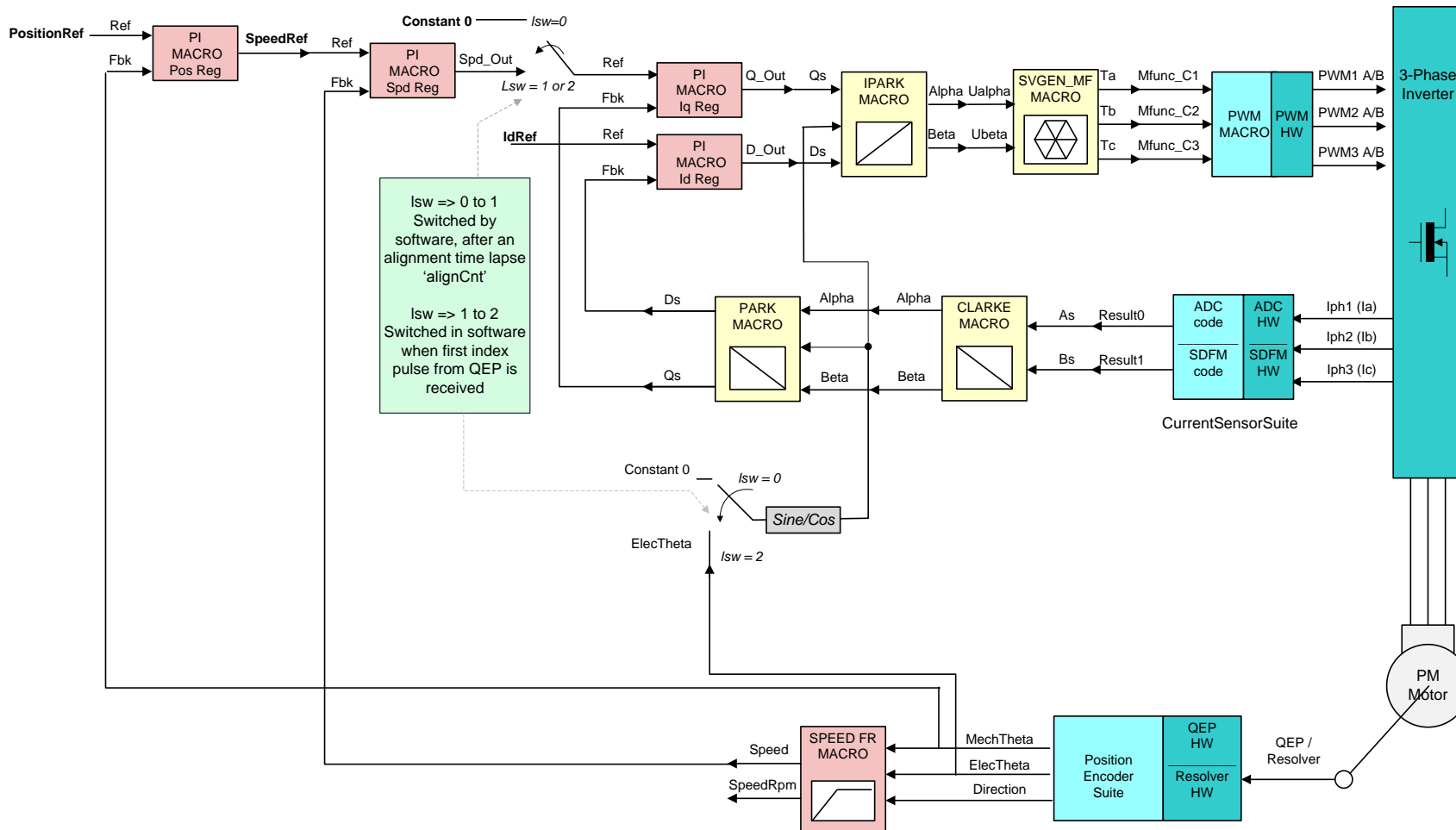


Figure 26 Flux and torque components of the stator current in the synchronous reference frame under 0.33pu step- load and 0.3 pu speed

Level 5 - Incremental System Build Block Diagram



Level 5 verifies the position PI module and position loop

Level 5 Incremental Build

This section verifies the position PI module and position loop with a QEP or a resolver. For this loop to work properly, the previous section must have been completed successfully. When the motor is commanded to run, it is taken through an initial alignment stage where the electrical angle and the QEP angle count are set to zero. If a resolver is used, its initial position at electrical angle zero is identified for run time corrections. After ensuring a stable alignment, the rotor is spun in FOC from start.

Open `HVPM_Sensored-Settings.h` and select level 5 incremental build option by setting the `BUILDLEVEL` to `LEVEL5` (`#define BUILDLEVEL LEVEL5`). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in watch windows to confirm the interrupt working properly. Set ‘RunMotor’ to 1 in the Expressions window. Setting this flag will run the motor through predefined motion profiles / position settings as set by the ‘refPosGen()’ module. This module basically cycles the position reference through a set of values as defined in an array ‘posArray’. These values represent the number of the rotations/ turns with respect to the initial alignment position. Once a certain position value as defined in the array is reached, it will pause for a while before slewing towards the next one. Hence these array values can be referred as parking positions. During transition from one parking position to the next, the rate of transition (or speed) is set by ‘posSlewRate’. The number of positions in ‘posArray’ it will pass through before restarting again from the first value is decided by ‘ptrMax’. Hence add the variables “posArray”, “ptrMax” and “posSlewRate” to the expressions window.

The key steps can be explained as follows:

- Compile/load/enable real-time mode and run the program.
- Add variables ‘pi_pos’, ‘posArray’, ‘ptrMax’ and ‘posSlewRate’ to the expressions window
- Gradually increase voltage at variac to get an appropriate DC-bus voltage
- Set RunMotor = 1 to run the motor. The motor should be turning to follow the commanded position (see note 1 below if the motor doesn’t turn properly).
- The parking positions in ‘posArray’ can be changed to different values to see if the motor turns as many rotations as set.
- The number of parking positions ‘ptrMax’ can also be changed to set a rotation pattern.
- Position slew rate can be changed using ‘posSlewRate’. This represents the angle (in pu) per sampling instant.
- The proportional and integral gains of the speed and position PI controllers may be re-tuned to get satisfactory responses. It is advised to tune the speed loop first and then the position loop.
- 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 is stopping

In the scope plot shown below, position reference and position feedback are plotted. It can be seen that they are aligned with negligible lag, which may be attributed to software. If the K_p , K_i gains of the position loop controller are not chosen properly, then it may lead to oscillations in the feedback or a lagged response.



Figure 27 Scope plot of reference position to servo and feedback position

NOTE:

1. If the motor response is erratic, then the sense of turn of motor shaft and the encoder may be opposite. Swap any two phase connections to the motor and repeat the test.
2. The position control implemented here is based on an initial aligned electrical position (= 0). If the motor has multiple pole pairs, then this alignment can leave the shaft in different mechanical positions depending on the pre start mechanical position of rotor. If mechanical position repeatability or consistency is needed, then QEP index pulse should be used to set a reference point. This may be taken as an exercise.
3. With an absolute encoder like resolver, the above may not be an issue as the resolver gives unique angular value for each position.