Application Report

Sensored Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors Using TMS320F2837x

Ramesh T Ramamoorthy, Brett Larimore, Manish Bhardwaj

ABSTRACT

This application report 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 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.

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

Table of Contents

List of Figures

List of Tables

Trademarks

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

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 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 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.

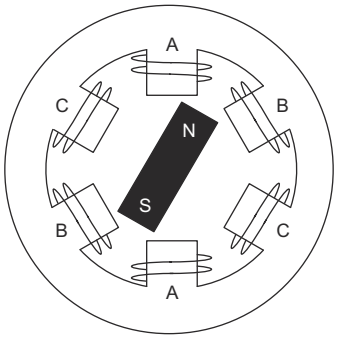


Figure 2-1. A Three-Phase Synchronous Motor With a One Permanent Magnet Pair Pole Rotor

3 Synchronous Motor Operation

- Synchronous motor construction: Permanent magnets are rigidly fixed to the rotating axis to create a constant rotor flux. 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 the number of desired poles, and the desired flux gradients.
- The interaction between the stator and the 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, fine tuning is needed after closing the speed loop 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° 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.

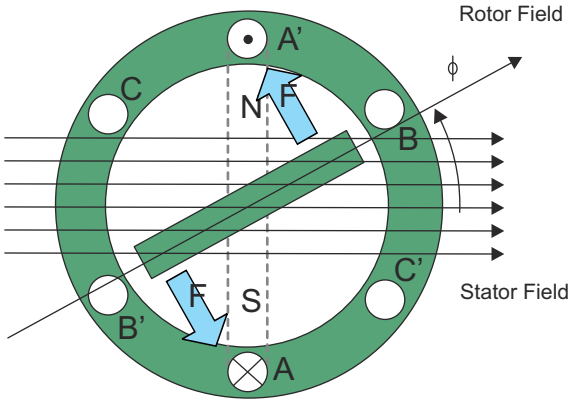


Figure 3-1. Interaction Between the Rotating Stator Flux and the Rotor Flux Produces a Torque That Causes the Motor to Rotate

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 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 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. 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 (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.

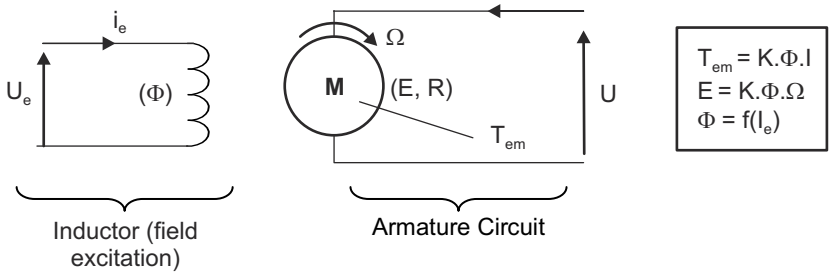


Figure 4-1. Separated Excitation DC Motor Model (Flux and Torque are Independently Controlled and the Current Through the Rotor Windings Determines How Much Torque is Produced)

AC machines do not have the same key features as the DC motor. 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 shown in Equation 1.

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

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.

4.3 Technical Background

The FOC effectively controls the stator current vector. This control is based on projections that 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 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. The FOC thus solves the classic scheme problems, in the following ways:

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

$$m \propto \Psi_R \, i_{Sq} \tag{2}$$

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

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, then the complex stator current vector $\bar{i}_s$ is defined in Equation 3 .

$$\bar{i} = i_a + \alpha i_b + \alpha^2 i_c \tag{3}$$

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

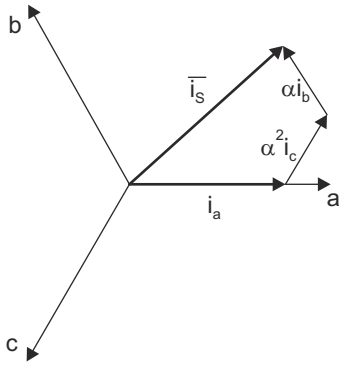


Figure 4-2. Stator Current Space Vector and Its Component in (a,b,c)

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 co-ordinate system. This transformation can be split into two steps:

- (a,b,c) → (*α, β*) (the Clarke transformation), which outputs a two co-ordinate time variant system
- (*α, β*) → (d, q) (the Park transformation), which outputs a two co-ordinate 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 4-3.

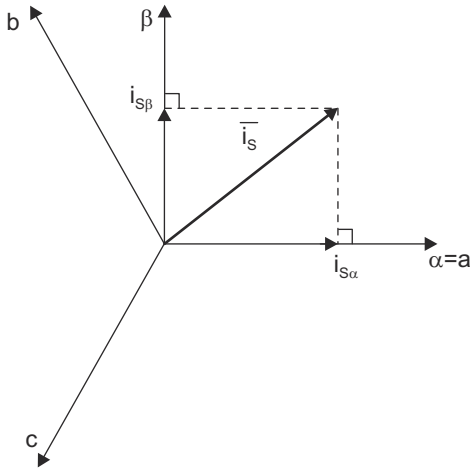


Figure 4-3. Stator Current Space Vector and Its Components in the Stationary Reference Frame

The projection that modifies the three phase system into the (*α, β*) two dimension orthogonal system is presented in Equation 4.

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

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

4.6 The (α, β) → (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 you consider the d axis aligned with the rotor flux, Figure 4-4 shows, for the current vector, the relationship from the two reference frame.

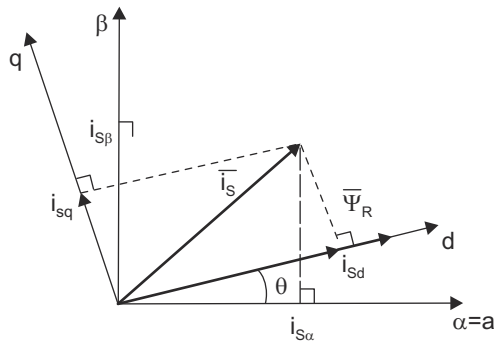


Figure 4-4. Stator Current Space Vector and Its Component in (*α, β*) and in the d,q Rotating Reference Frame

where, θ is the rotor flux position. The flux and torque components of the current vector are determined by Equation 5

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

These components depend on the current vector (*α, β*) 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 5-1 summarizes the basic scheme of torque control with FOC.

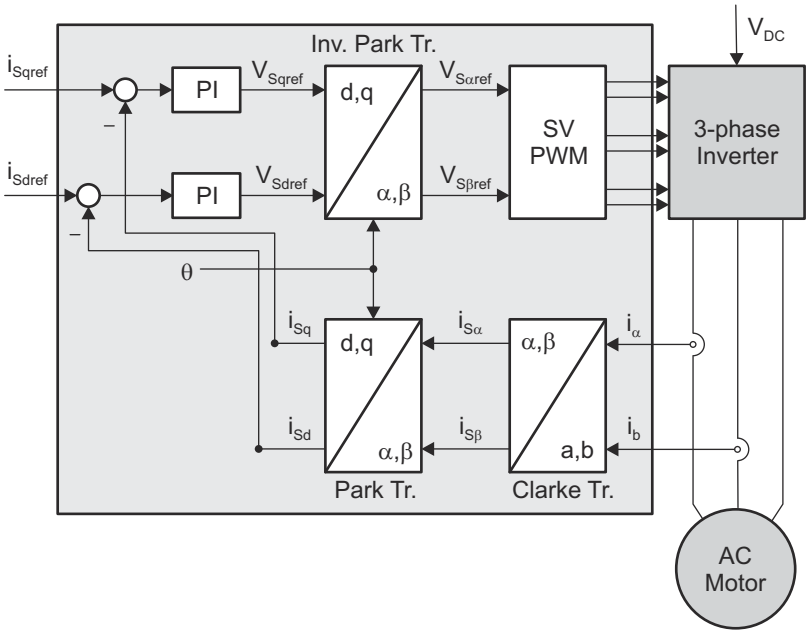


Figure 5-1. Basic Scheme of FOC for AC Motor

Two motor phase currents are measured. These measurements feed the Clarke transformation module. The outputs of this projection are designated $i_{sα}$ and $i_{sβ}$. These two components of the current along with rotor flux position are the inputs of the Park transformation that transform them to currents (i_{sd} and i_{sq}) in d,q rotating reference frame. The i_{sd} and$_{isq}$ components are compared to the references i_{sdref} (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_{sdref} 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 the connected to the output of the speed regulator. The outputs of the current regulators are V_{sdref} 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 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} will represent incorrect flux and torque components of the stator current. Figure 5-2 shows the (a,b,c), (*α, β*) and (*d,q*) reference frames, and the correct position of the rotor flux, the stator current and stator voltage space vector that rotates with d,q reference at synchronous speed.

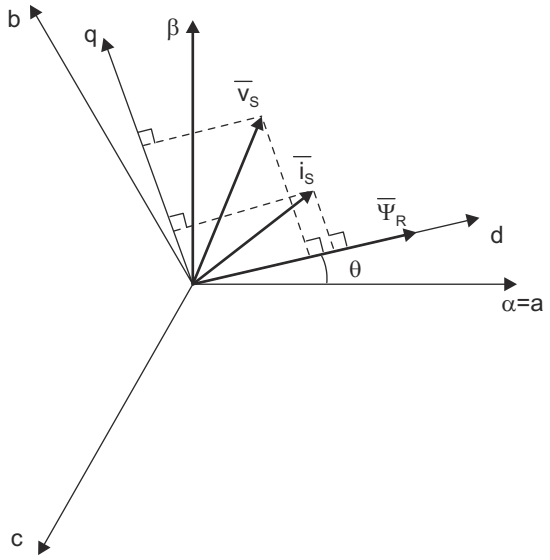


Figure 5-2. Current, Voltage and Rotor Flux Space Vectors in the d,q Rotating Reference Frame and Their Relationship With a,b,c and (*α, β*) Stationary Reference Frame

The measure of the rotor flux position is different if you 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 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 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 5-3.

The overall block diagram of this project is depicted in Figure 5-3.

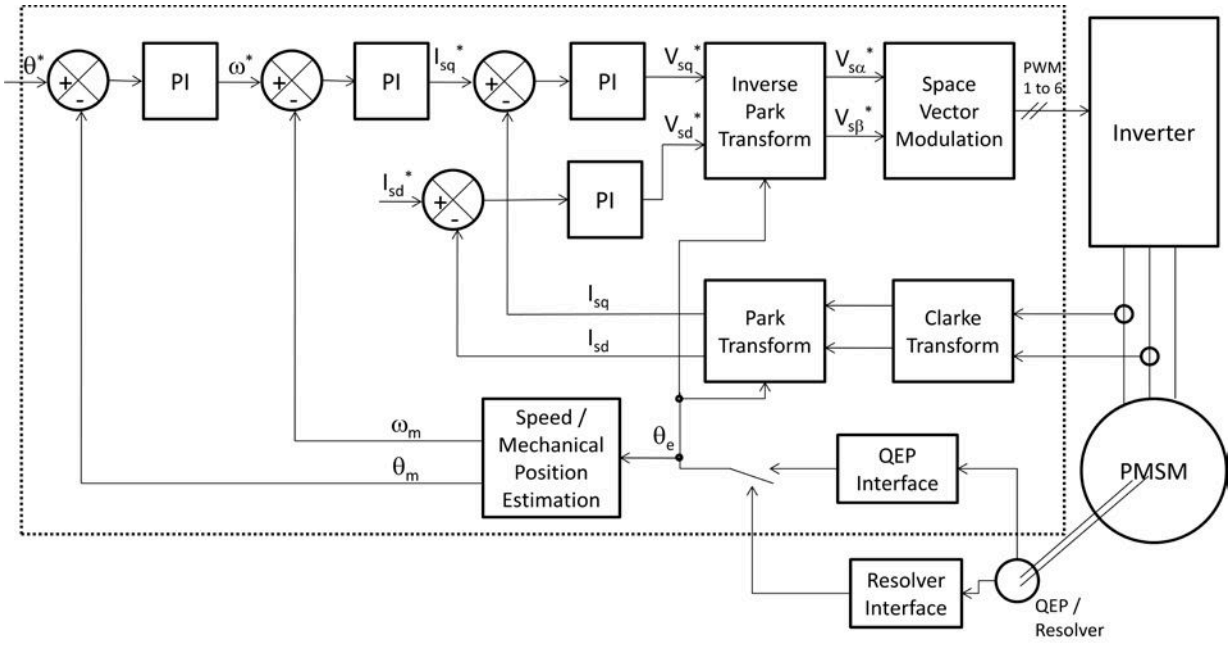


Figure 5-3. Overall Block Diagram of Sensored Field Oriented Control

6 Benefits of 32-Bit C2000™ Controllers for Digital Motor Control (DMC)

The C2000 family of devices posses 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 analog-to-digital converter (ADC), enhanced pulse width modulator (ePWM), quadrature encoder pulse (QEP), enhanced capture (ECAP), and so forth. These peripherals have all the necessary hooks for implementing systems that 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.

Therefore, 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
- Use of 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 reduces harmonics generated by the inverter, reducing filter cost.
- Use of sensorless algorithms eliminates the need for speed or position sensor.
- Decreases the number of look-up tables that 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 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 and choose:

- 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 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. At initialization, all DMC Library variables are defined and inter-connected. At run-time, the macro functions are called in order. The 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 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 7-1, 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; you can easily convert the macro definition to a C functions.

Example 7-1. 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 sensored FOC of HVPM motors. The “C” framework is designed to run on C2000-based controllers on Code Composer Studio™ software. The framework uses the following modules: [1] [1]:

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

The overall system implementing sensored FOC of PMSM is depicted in Figure 7-1. The control experimented with various current sense methods and position sense methods helping to explore the impact of feedback methods on the system performance. The PM motor is driven by a conventional voltage-source inverter. The 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
- 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.

[1] For more information, see the pdf documents in the motor control folder explaining the details and theoretical background of each macro.

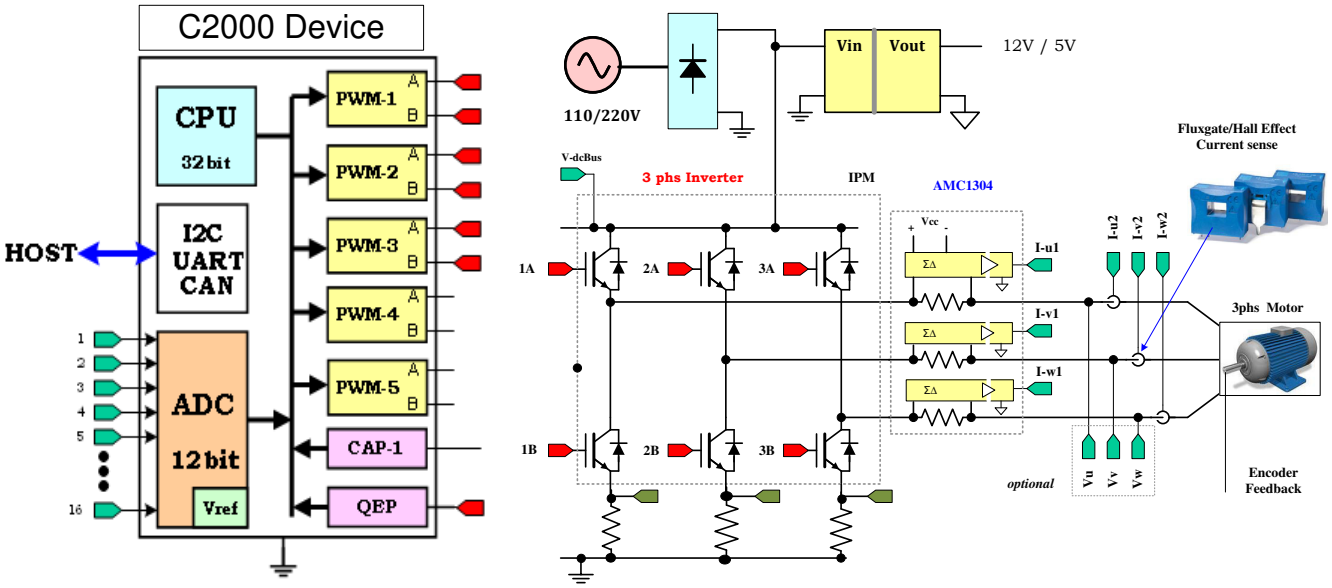


Figure 7-1. A 3-Phase Induction Motor Drive Implementation

The software flow is described in the Figure 7-2.

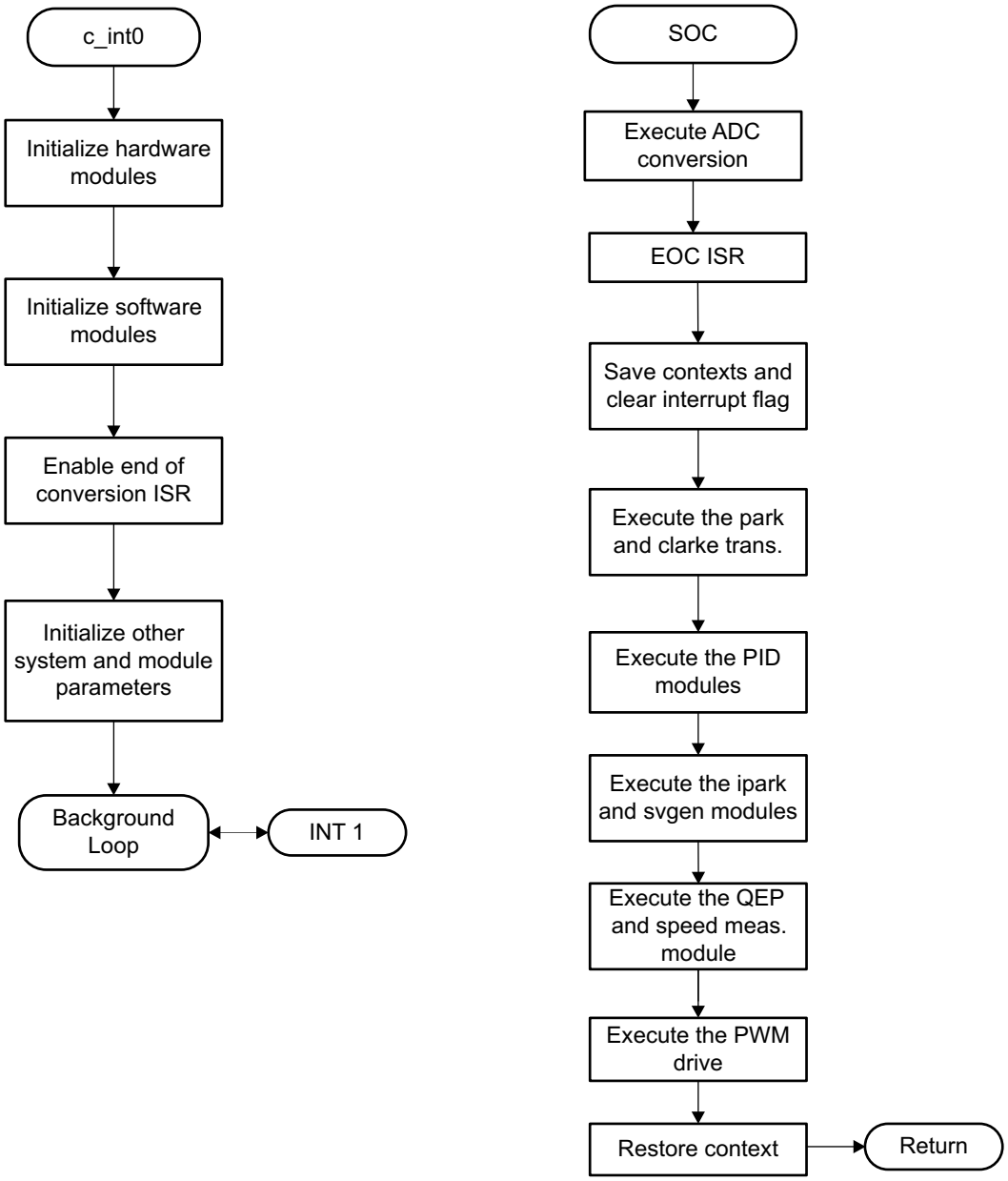


Figure 7-2. System Software Flowchart

8 Hardware Configuration (IDDK)

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

C:\TI\controlSUITE\development_kits\TMDSIDDK_v2.0 \~Docs.

For an immediate reference about powering the board, Figure 8-1 shows how to power it from an external DC supply.

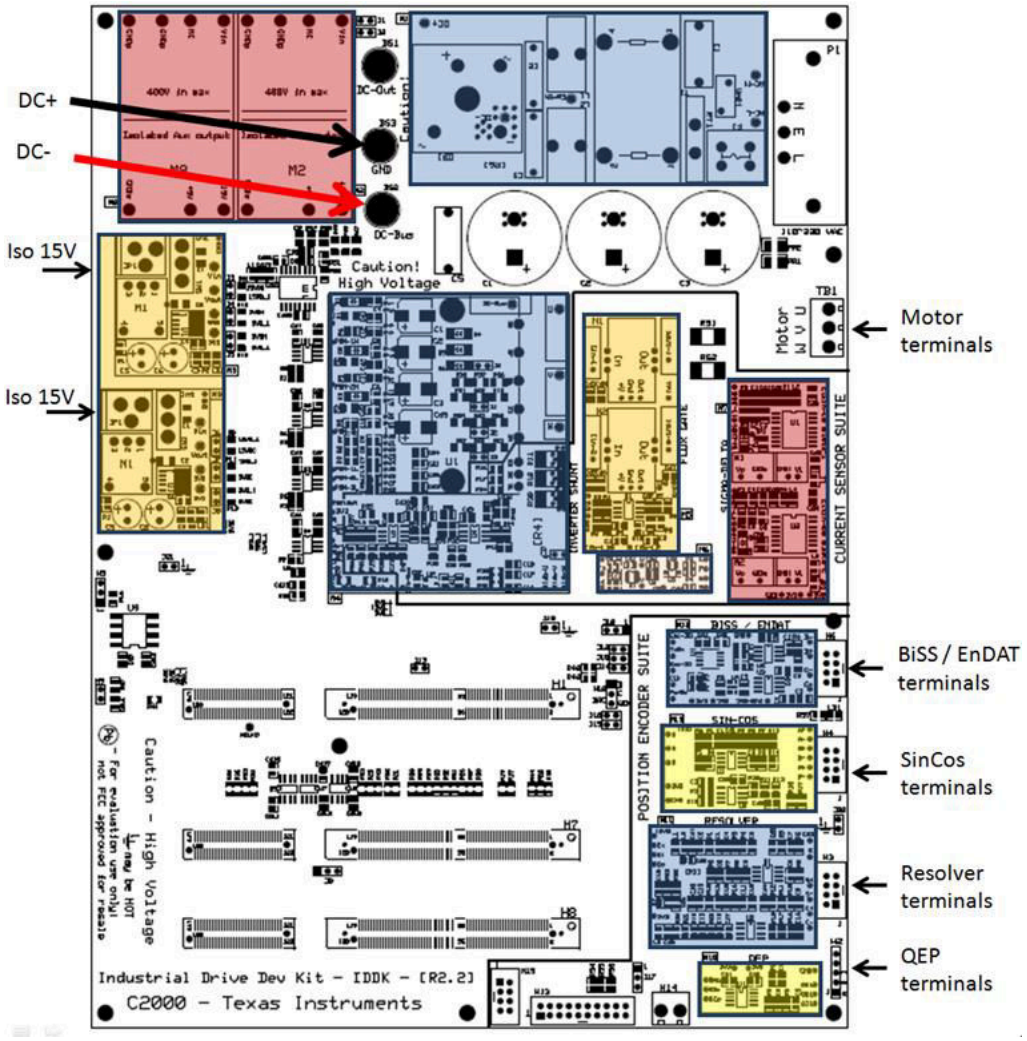


Figure 8-1. Powering IDDK From External DC Power Supply

CAUTION

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

Figure 8-2 shows powering up the board from the AC mains; remember to connect BS1 and BS3 with a banana jumper to feed the rectifier output to the DC bus as shown by the thick red line.

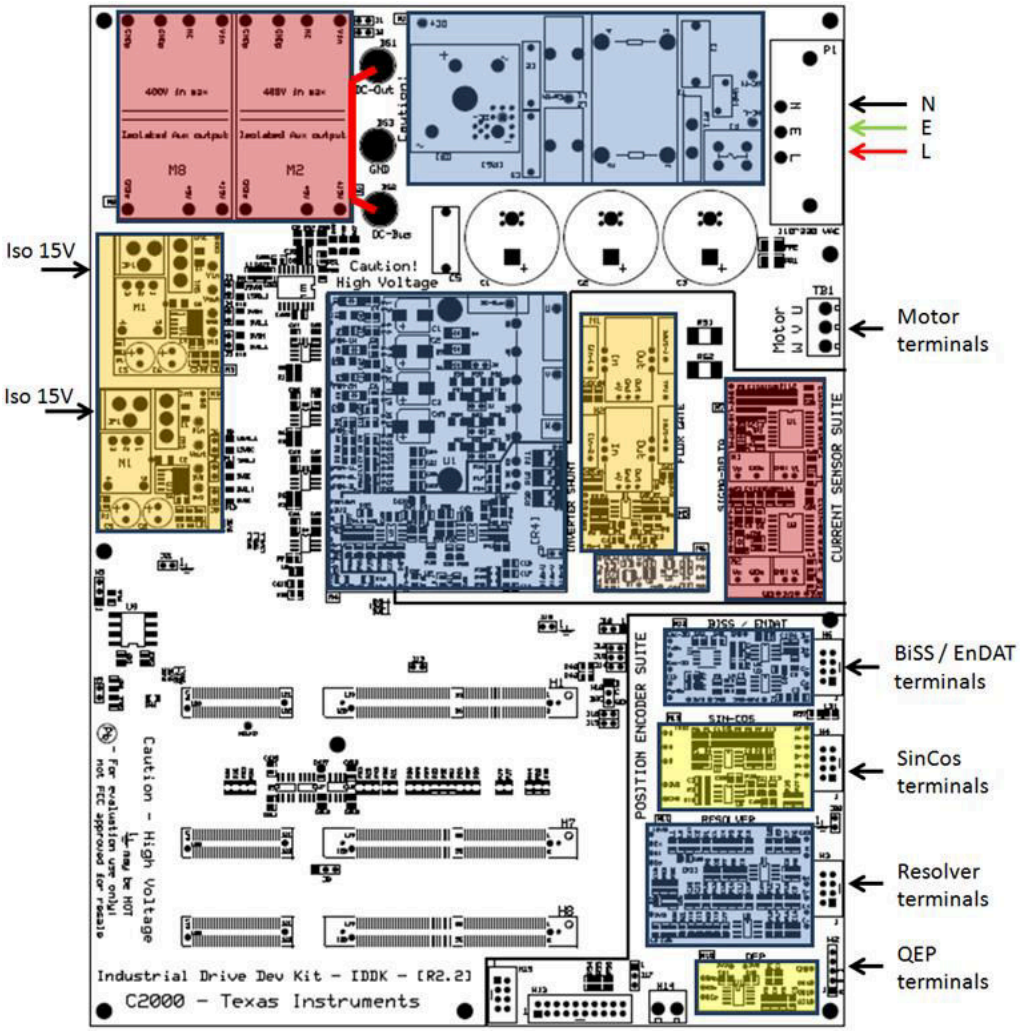


Figure 8-2. Powering IDDK From an AC Source

CAUTION

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

During development, it is preferable to use an isolation transformer for equipment and personnel safety as shown in Figure 8-3.

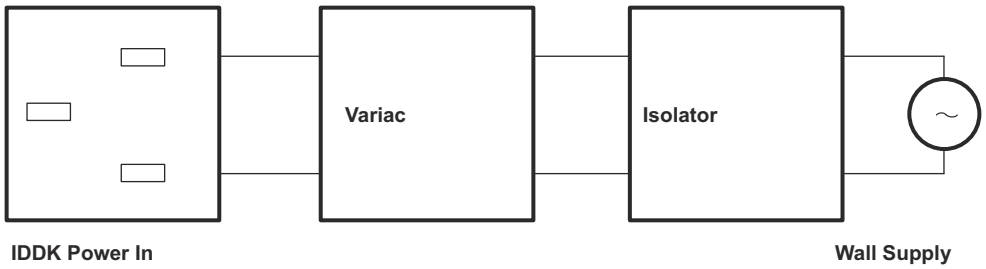


Figure 8-3. AC Power Connection to IDDK Through an Isolation Transformer

CAUTION

Depending on the choice of power GND and control GND plane interconnections, even the control GND may be HOT. Proceed with caution!

8.1 Software Setup Instructions to Run HVPM_Sensored Project

For more information, see the IDDK User's Guide at:

C:\TI\controlSUITE\development_kits\TMDSIDDK_v2.0 \~Docs

Open CCS and load the IDDK project by browsing to: C:\TI\ controlSUITE\development_kits\TMDSIDDK_v2.0 \IDDK_PM_Servo_F2837x.

1. Select HVPM_Sensored as the active project.
2. Select the active build configuration to be set as F2837x_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 into CPU1 of the controller.
4. Add variables to the expressions window by ‘Right Clicking’ within the Expressions Window and ‘Importing’ the file ‘Variables_IDDK_Level1.txt’ from the root directory. Find the Expressions Window as shown in Figure 8-4.

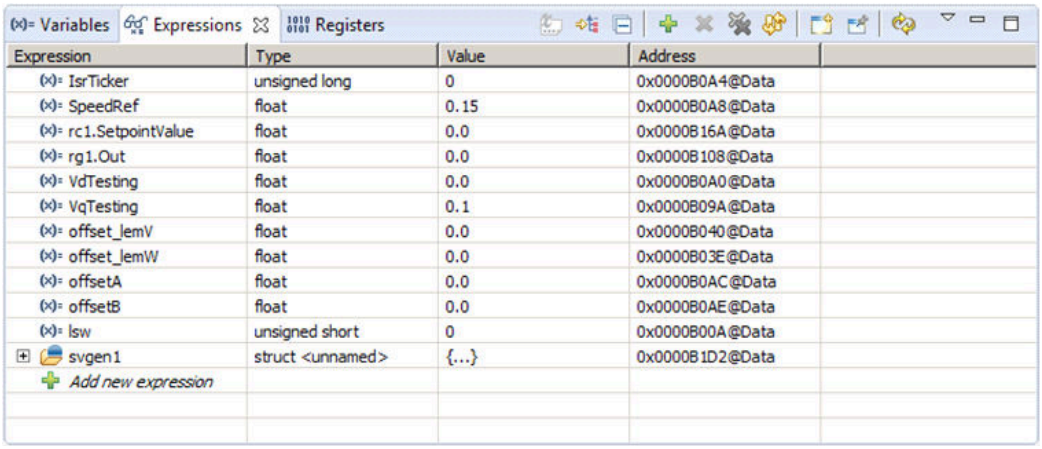


Figure 8-4. Watch Window Variables

5. Setup the time graph windows by importing Graph1.graphProp and Graph2.graphProp from the following location: C:\TI\ControlSUITE\developement_kits\TMDSIDDK_v2.0 \IDDK_PM_Servo_F2837x\.
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. Most modules are written

as software MACROs, and the remaining are written as callable functions. Table 9-1 summarizes the modules testing and using in each incremental system build.

Table 9-1. Testing Modules in Each Incremental System Build

Software Module (1)	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)		√√	√√	√	√
PI_MACRO (SPD)			√√	√√	√
PI_POS_MACRO (POS)					√√

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

9.1 Level 1 - Incremental Build

Figure 9-1 shows the block diagram of the system built in BUILDLEVEL 1

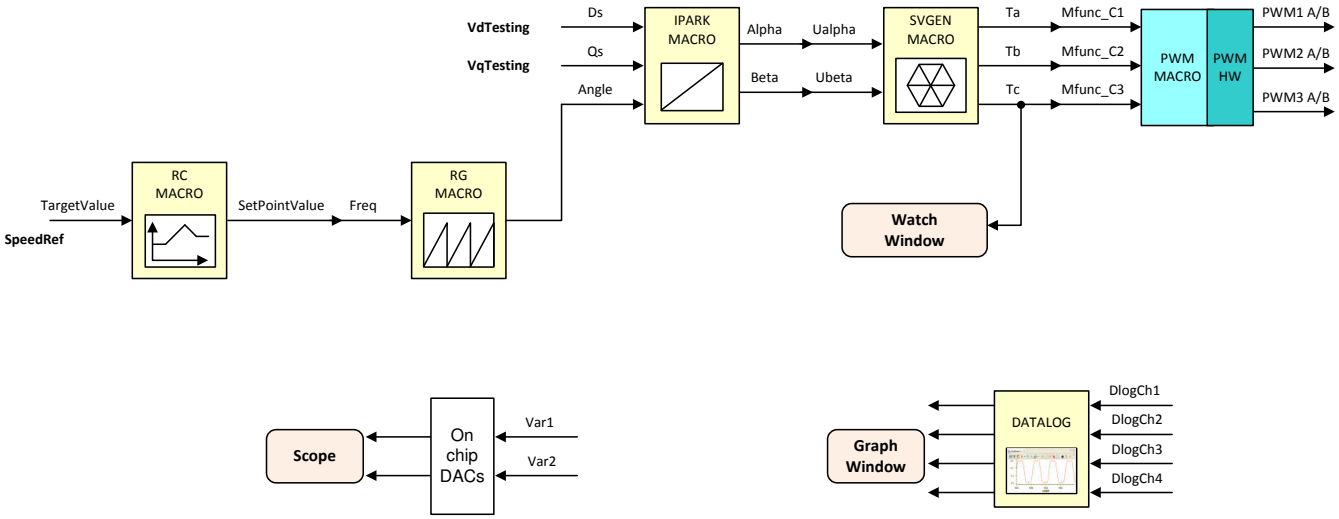


Figure 9-1. Level 1 - Incremental System Build Block Diagram

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

Keep the motor disconnected at this step. Assuming the load and build steps described in the *DesignDRIVE Development Kit IDDK v2.2 User's Guide* (SPRUI24) 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 IDDK_PM_Servo_F2837x-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. 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.
5. Set the “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

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 9-2 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.

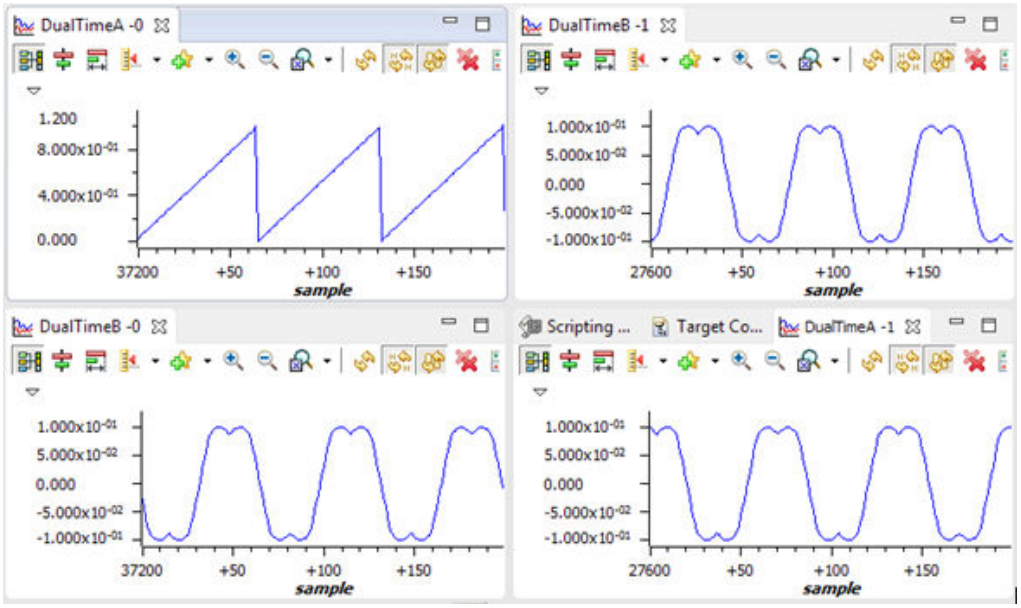


Figure 9-2. Output of SVGEN, Ta, Tb, Tc and Tb-Tc Waveforms

9.3 Level 1B - Testing the DACs

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

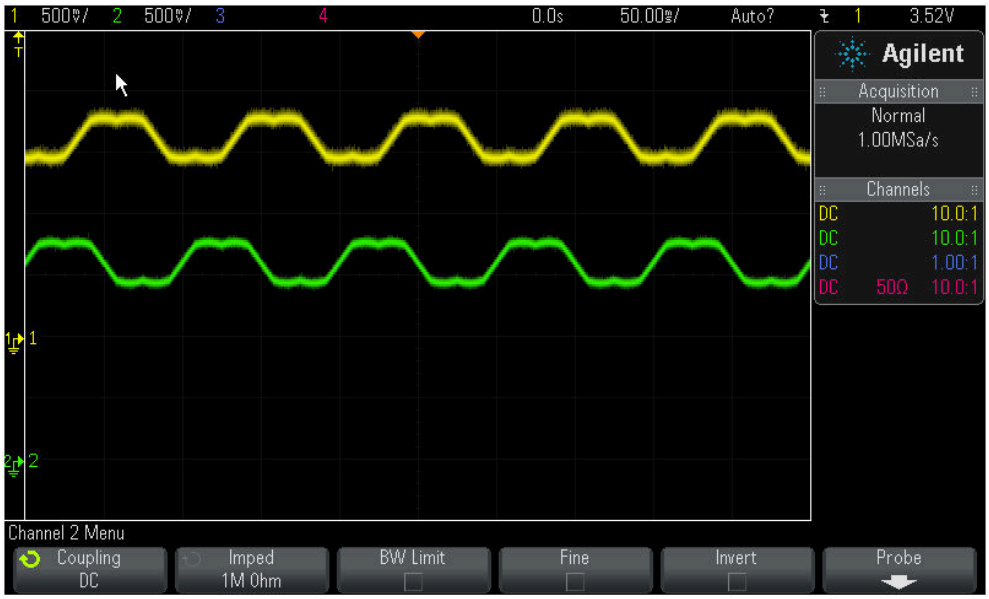


Figure 9-3. DAC 1-4 Outputs Showing Ta, Tb Waveforms

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 *IDDK User's 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 certain states where high currents can be drawn, therefore, caution needs to be taken while doing these experiments.

9.5 Level 1D - Tuning Resolver Loop Parameters

Information related to tuning the resolver parameters and resolver is provided in:

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

In this test, you 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 150° (approximate) for the loop. This value can also be varied for experimenting.

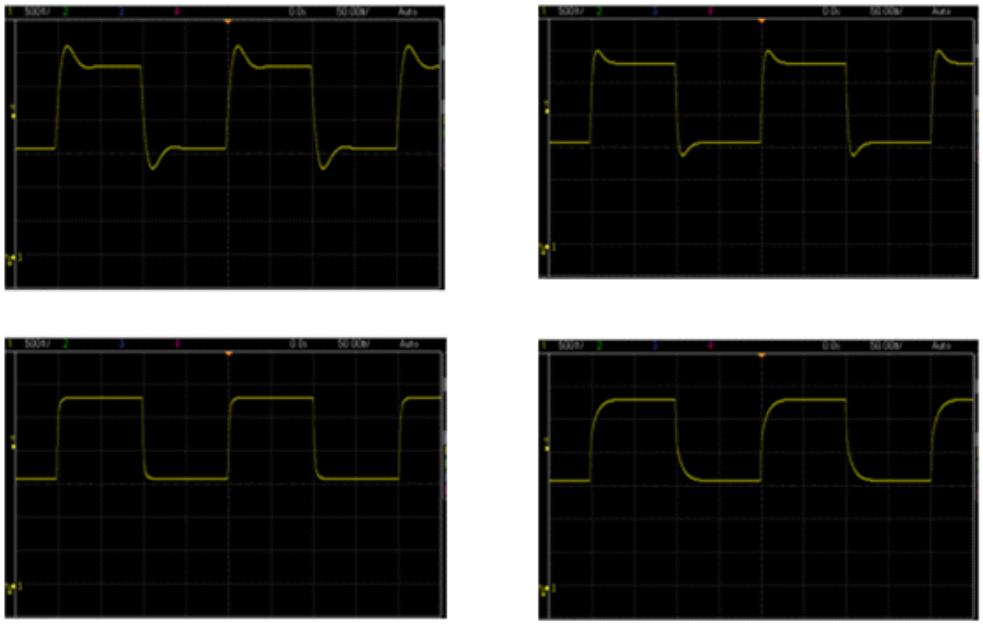


Figure 9-4. Observer Angle Response

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 9-4 shows 'rslvrOut.angleObs' for increasing values of Kp, while test reference angle toggles between 0° and 150°.

9.6 Level 2 - Incremental Build

Figure 9-5 shows the block diagram of the system built in BUILDLEVEL 2.

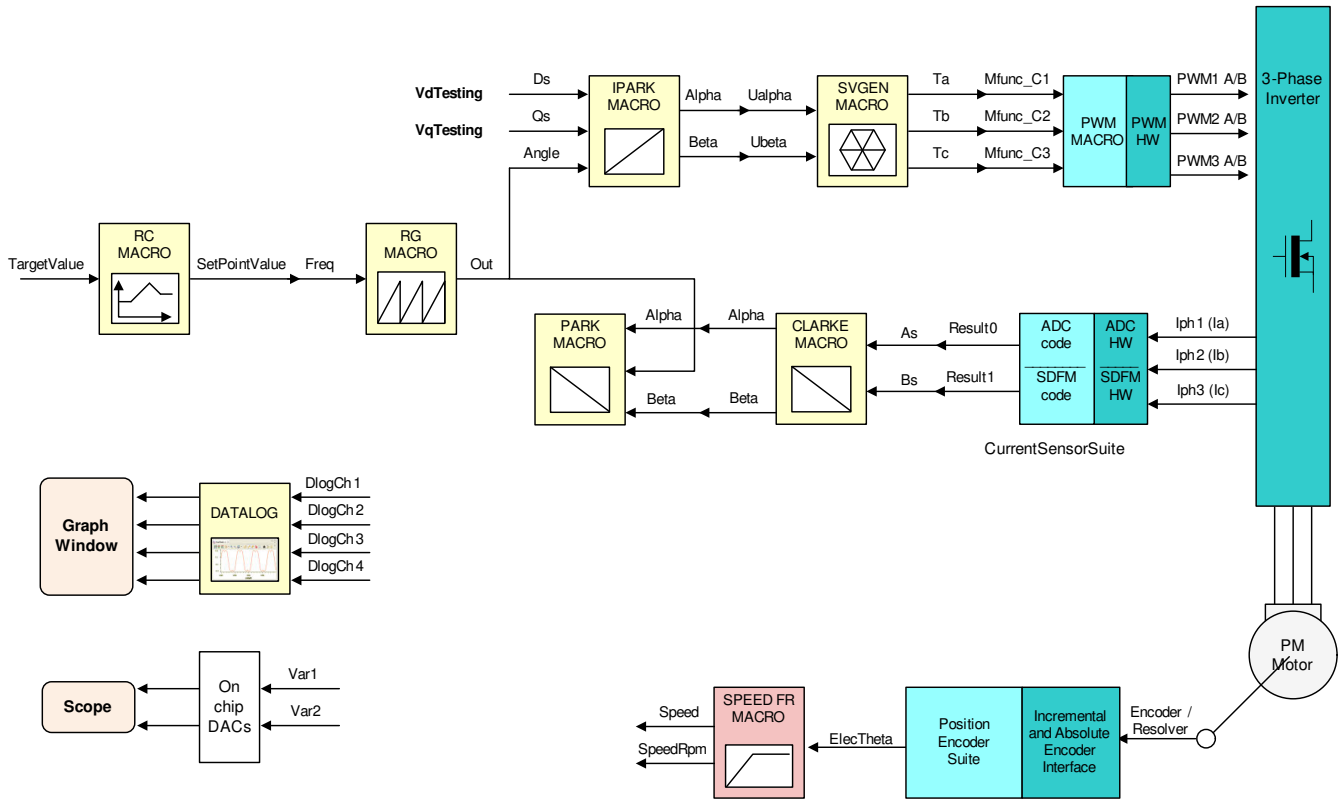


Figure 9-5. Level 2 - Incremental System Build Block Diagram

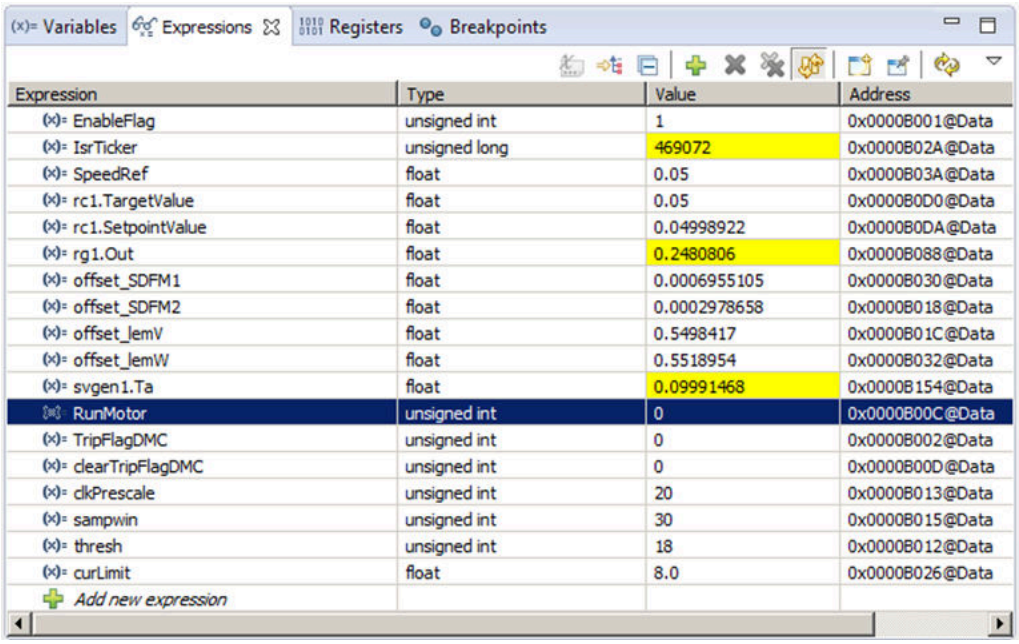
Level 2 verifies the analog-to-digital conversion, offset compensation, Clarke and Park transformations.

In this section, some more blocks are added to level 1 and tested. Assuming section 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, EnDat, BiSS or resolver) used in the set up.

The motor can be connected to the IDDK 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 the motor under load or at various operating points is not recommended.

1. Open IDDK_PM_Servo_F2837x-Settings.h and select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL LEVEL2).
2. Select CURRENT_SENSE to LEM_CURRENT SENSE and POSITION_ENCODER to QEP_POS_ENCODER or RESOLVER_POS_ENCODER or BISS_POS_ENCODER or ENDAT_POS_ENCODER depending on the encoder in use.The default value is set to QEP_POS_ENCODER.
3. Right Click on the project name and click Rebuild Project.
4. After the build is complete, click on the Debug button, reset the CPU, restart, enable real-time mode and run, once the build is complete.

5. Import variables from ‘Variables_IDDK_Level2.txt’ file in the root directory and the expressions window will look as shown in Figure 9-6.



Expression	Type	Value	Address
EnableFlag	unsigned int	1	0x0000B001@Data
IsrTicker	unsigned long	469072	0x0000B02A@Data
SpeedRef	float	0.05	0x0000B03A@Data
rc1.TargetValue	float	0.05	0x0000B0D0@Data
rc1.SetpointValue	float	0.04998922	0x0000B0DA@Data
rg1.Out	float	0.2480806	0x0000B088@Data
offset_SDFM1	float	0.0006955105	0x0000B030@Data
offset_SDFM2	float	0.0002978658	0x0000B018@Data
offset_IemV	float	0.5498417	0x0000B01C@Data
offset_IemW	float	0.5518954	0x0000B032@Data
svgen1.Ta	float	0.09991468	0x0000B154@Data
RunMotor	unsigned int	0	0x0000B00C@Data
TripFlagDMC	unsigned int	0	0x0000B002@Data
clearTripFlagDMC	unsigned int	0	0x0000B00D@Data
clkPrescale	unsigned int	20	0x0000B013@Data
sampwin	unsigned int	30	0x0000B015@Data
thresh	unsigned int	18	0x0000B012@Data
curLimit	float	8.0	0x0000B026@Data
Add new expression			

Figure 9-6. Expressions Window for Build Level 2

6. 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.
7. 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 Bemf is lower than the average voltage applied to motor winding. This will prevent the motor from stalling or vibrating.

9.7 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’
- ‘curLimit’ sets the permitted current max through both SHUNT and LEM current sensors

‘TripFlagDMC’ is a flag variable used to represent the over current trip status of the inverter. If this flag is set, then adjust the settings above and retry running the inverter by setting ‘clearTripFlagDMC’ to 1. This clears ‘TripFlagDMC’ and restarts the PWMs.

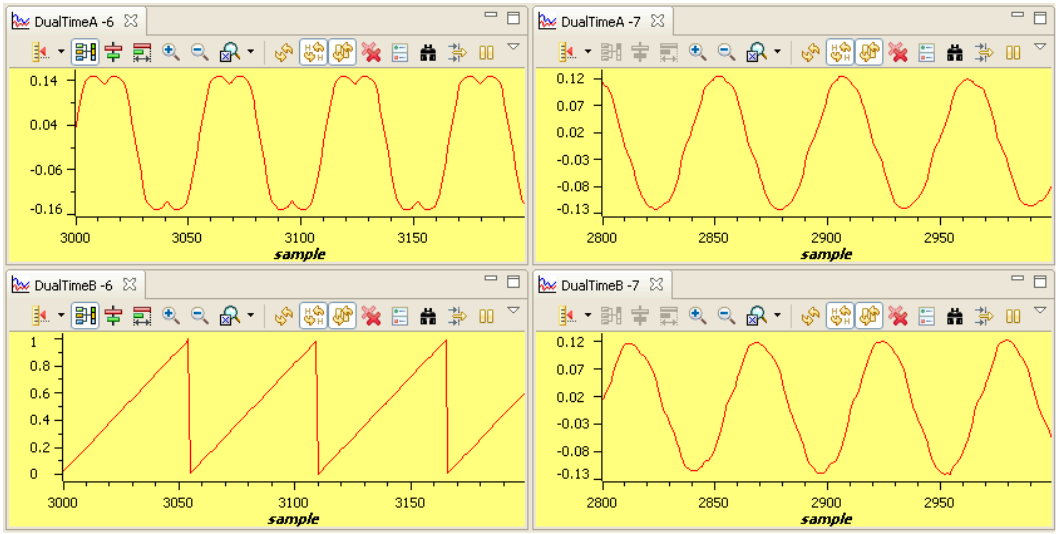
The default current limit setting is to shutdown at 8A. You can fine tune any of these settings to suit your system. Once satisfactory values are identified, make a note of them and modify the code with these new values; rebuild and load them for further tests.

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 use it as an exercise to experiment and learn.

9.8 Level 2B – Testing the Clarke Module

In this part, the Clarke module is tested. The three measured line currents* are transformed to two phase alpha and beta 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 90° at the same magnitude.



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

Figure 9-7. The Waveforms of Svgen_dq1.Ta, rg1.Out and Phase A and B Currents

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]-Ifb-U, and [M5]-Ifb-V) are composed of pulses as shown in Figure 9-8.

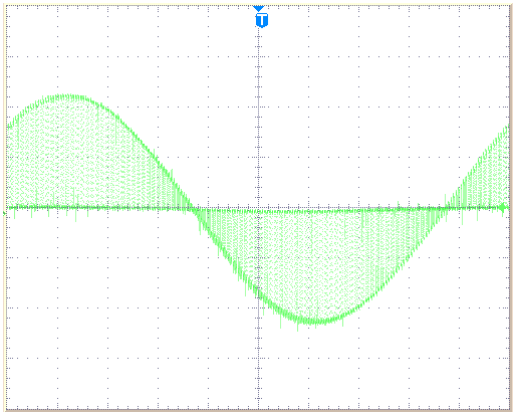


Figure 9-8. Amplified Phase A Current

9.9 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 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 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 real-time mode and reset.

9.10 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 you to pick the one of your choice to close the current loop. You can choose the current sense method that will eventually be used in your development.

9.11 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 or absolute encoder (EnDat or BiSS-C) 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 in Figure 9-9.

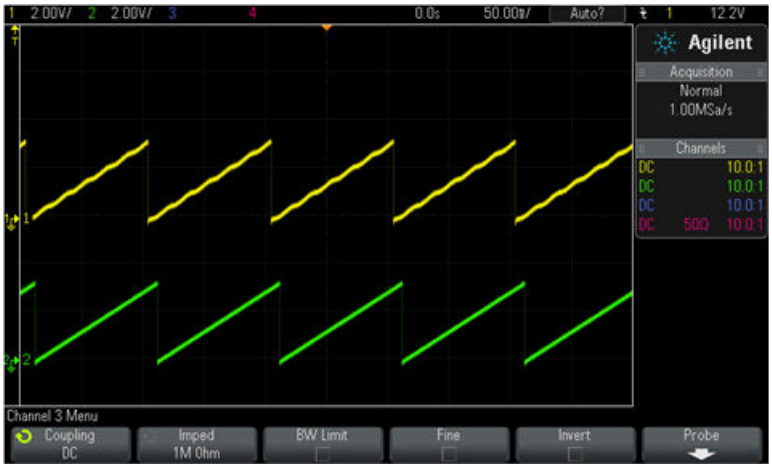


Figure 9-9. Scope Plot of Reference Angle and Rotor Position

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 the 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 the 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.

angle = 1.0 – angle

To make sure that the SPEED_MACRO works fine, change the 'SpeedRef' variable in Expressions Window as shown in Figure 9-10 and check whether 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.

Variables | Expressions | Registers | Breakpoints

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

Figure 9-10. Expressions Window

9.11.1 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:

Calibration Angle = Offset Angle ± n . Line Encoder

9.11.2 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.

9.11.3 With EnDat Encoder:

Watch out for ‘endat1.InitTheta’ in the Expressions window. It represents the EnDat encoder 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 EnDat encoder and the motor’s electrical zero.

Note

For more details regarding EnDat22 library and usage on IDDK, see controlSUITE/libs/app_libs/position_manager/vxx_xx_xx_xx/endat22.

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.

9.11.4 With BiSS-C Encoder:

Watch out for ‘biss1.InitTheta’ in the Expressions window. It represents the BiSS-C encoder 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 BiSS-C encoder 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 real-time mode and reset. Now the motor is stopping.

Note

For more details regarding BiSS-C library and usage on IDDK, see controlSUITE/libs/app_libs/position_manager/vxx_xx_xx_xx/bissc.

9.12 Level 3 - Incremental Build

Figure 9-11 shows the block diagram of the system built in BUILDLEVEL 3.

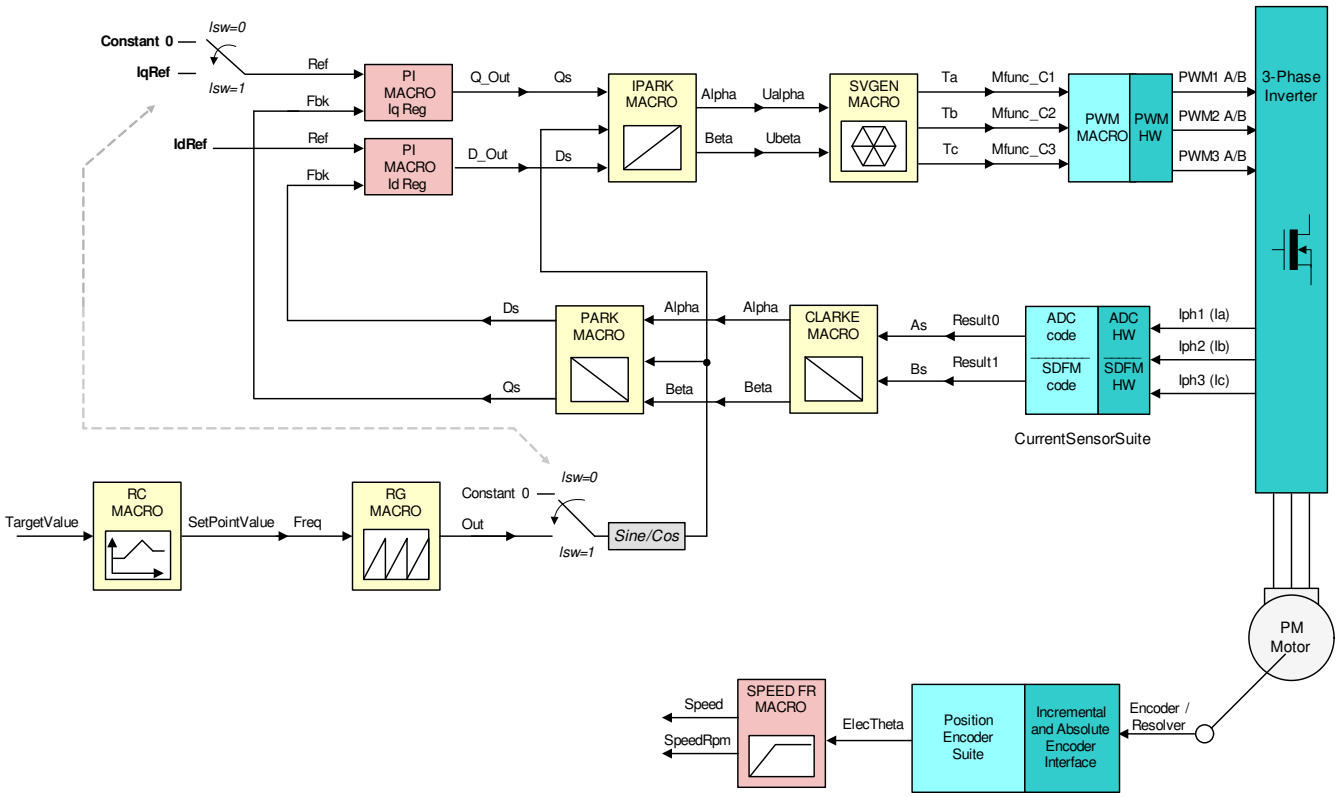


Figure 9-11. Level 3 - Incremental System Build Block Diagram

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

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 or absolute encoder (EnDat or BiSS-C) is used, its initial position at electrical angle zero is identified for run time corrections.

1. Open IDDK_PM_Servo_F2837x-Settings.h and select the level 3 incremental build option by setting the BUILDLEVEL to LEVEL3 (#define BUILDLEVEL LEVEL3). Choose any of the three supported CURRENT_SENSE methods.
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:

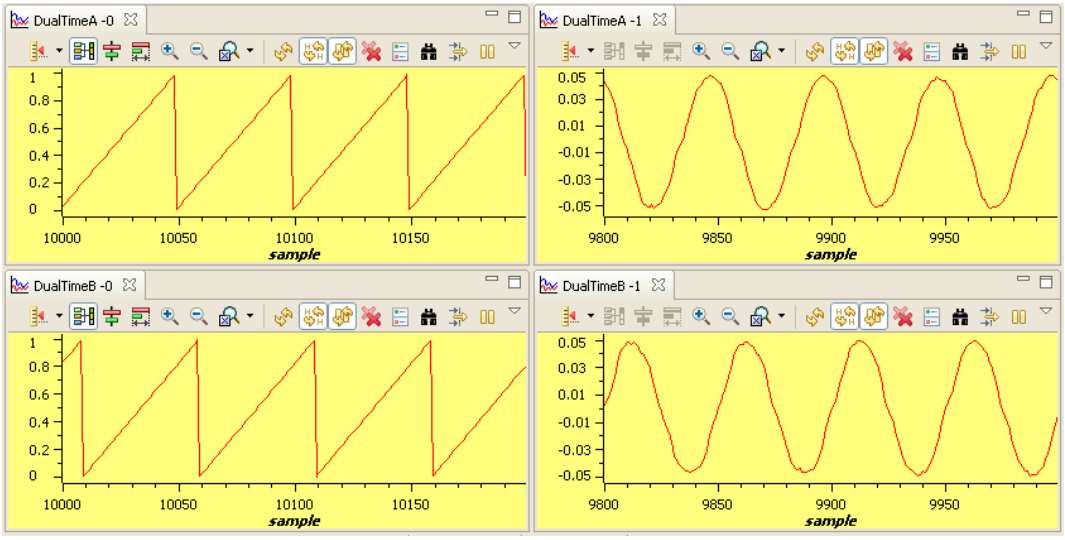
- SpeedRef: for changing the rotor speed in per-unit.
- IdRef: for changing the d-qxis voltage in per-unit.
- IqRef: for changing the q-axis voltage in per-unit.

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 key steps are explained as follows:

1. Compile, load, and run the program with real-time mode.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different), Idref to zero and Iqref to 0.05 pu (or another suitable value).
3. Add variables ‘pi_id.Fbk’, ‘pi_id.Kp’ and ‘pi_id.Ki’ and corresponding elements for ‘pi_iq’ to the expressions window
4. Gradually increase the voltage at the variac and dc power supply to get an appropriate DC-bus voltage.
5. Set ‘RunMotor’ flag to 1
6. Check pi_id.fbk in the watch windows with the continuous refresh feature whether or not it should be keeping track of pi_id.Ref for the PI module. If not, adjust its PI gains properly.
7. Check pi_iq.fbk in the watch windows with the continuous refresh feature whether or not it should be keeping track of IqRef for the PI module. If not, adjust its PI gains properly.
8. Try different values of pi_id.Ref and pi_iq.Ref or SpeedRef to confirm these two PI modules.
9. For both PI controllers, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied responses.
10. Bring the system to a safe stop (as described at the end of build 1) by reducing the bus voltage, taking the controller out of real-time mode and reset. Now the motor is stopping.

When running this build, the current* waveforms in the CCS graphs should appear as shown in Figure 9-12.



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

Figure 9-12. Measured theta, rg1.out, Phase A and B Current Waveforms

9.13 Level 4 - Incremental Build

Figure 9-13 shows the block diagram of the system built in BUILDLEVEL 4.

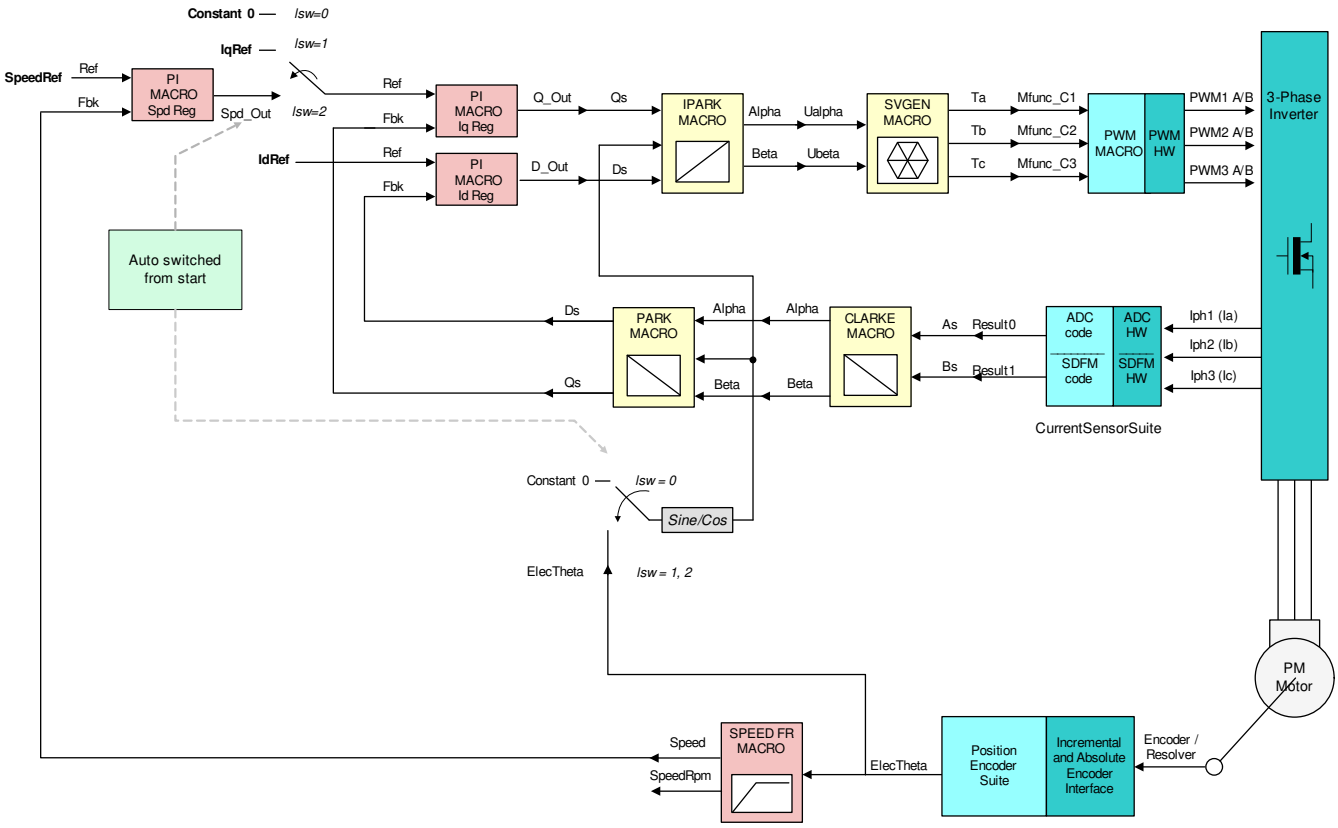


Figure 9-13. Level 4 - Incremental System Build Block Diagram

Level 4 verifies the speed PI module and speed loop.

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 or absolute encoder (EnDat or Biss-C) is used, its initial position at electrical angle zero is identified for run time corrections.

1. Open HVPM_Sensored-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.

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:

1. Set compile, load, and run the program with real-time mode.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
3. Gradually increase the voltage at the variac to get the appropriate DC-bus voltage. Now, the motor is running close to reference speed (0.3 pu).
4. 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:
   a. lsw = 0, lock the rotor of the motor
   b. lsw = 1, for QEP feedback only – motor in run mode and waiting for first instance of QEP Index pulse
   c. lsw = 2, motor in run mode, for all encoders (for QEP - first Index pulse occurred)
5. 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.
6. To confirm this speed PI module, try different values of SpeedRef (positive or negative). For the speed PI controller, the proportional, integral, derivative and integral correction gains may be re-tuned to have the satisfied response.
7. At very low speed range, the performance of speed response relies heavily on the good rotor position angle provided by QEP encoder.
8. 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, terminate the debug session.

While running this build, the current* waveforms in the CCS graphs should appear as shown in Figure 9-14 through Figure 9-16.

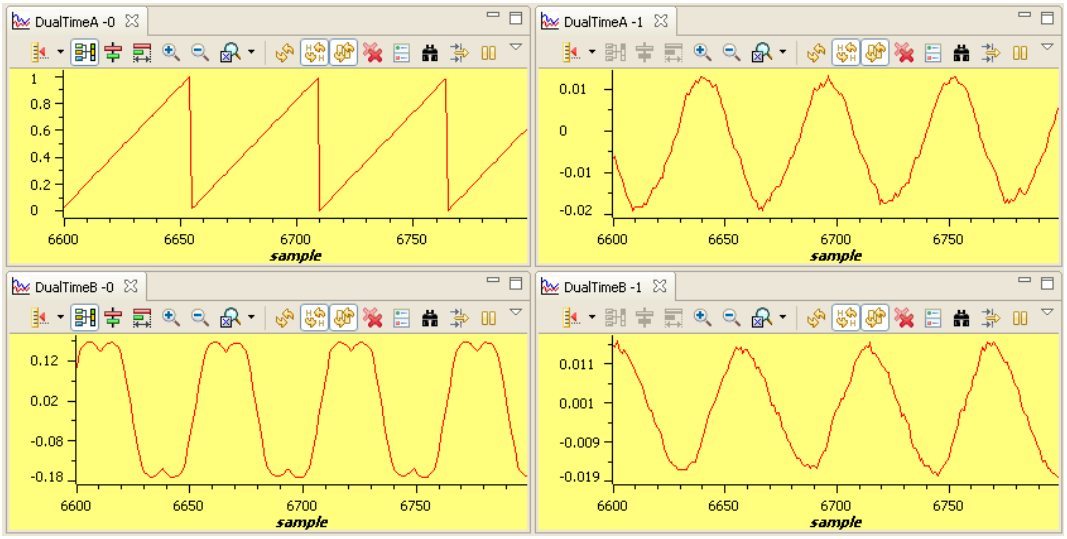
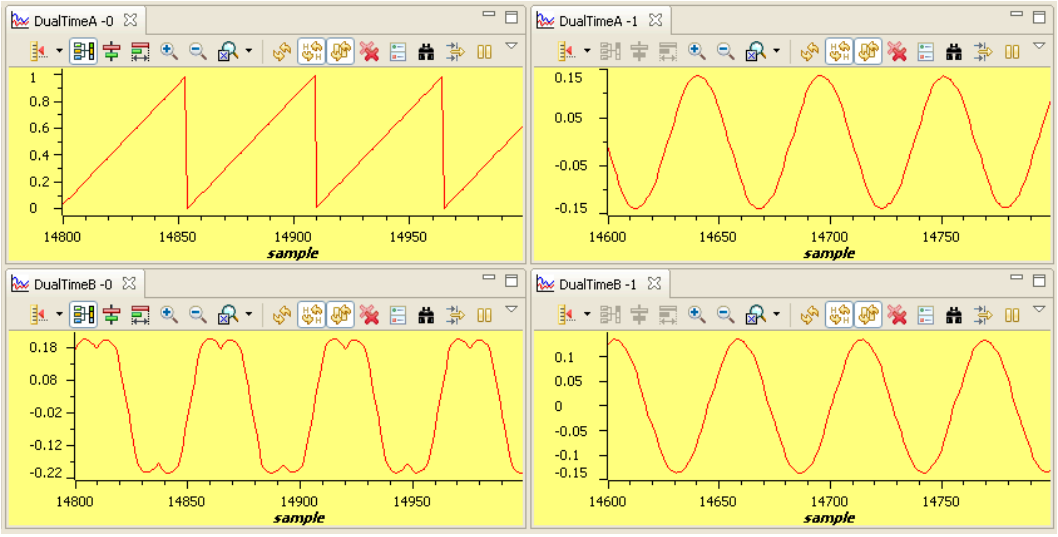


Figure 9-14. Measured theta, svgen Duty Cycle and Phase A and B Current Waveforms Under No-Load and 0.3 pu Speed



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

Figure 9-15. Measured theta, svgen Duty Cycle, and Phase A and B Current Waveforms Under 0.33 pu Load and 0.3 pu Speed

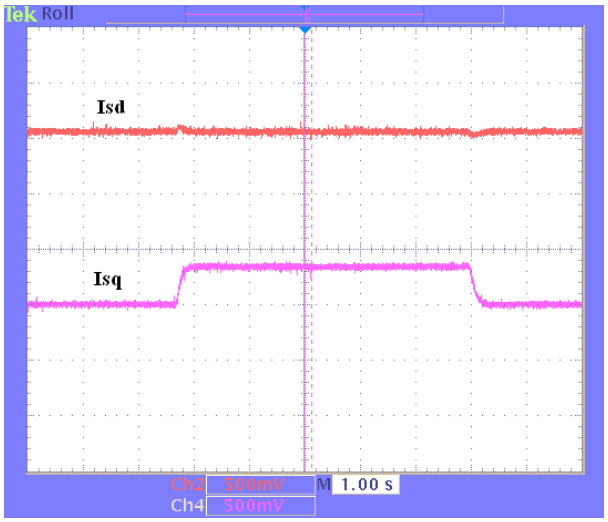


Figure 9-16. Flux and Torque Components of the Stator Current in the Synchronous Reference Frame Under 0.33 pu Step-Load and 0.3 pu Speed Monitored From PWMDAC Output

9.14 Level 5 - Incremental Build

Figure 9-17 shows the block diagram of the system built in BUILDLEVEL 5.

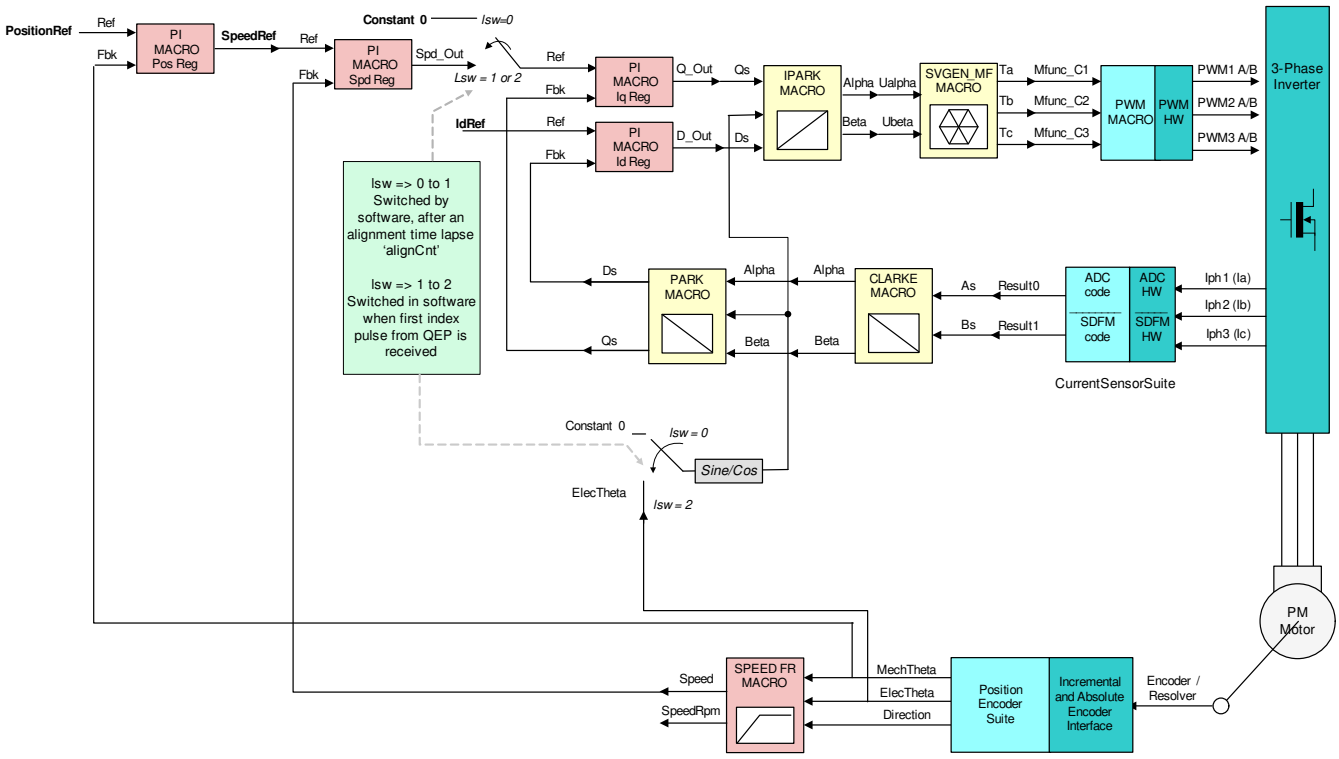


Figure 9-17. Level 5 - Incremental System Build Block Diagram

Level 5 verifies the position PI module and position loop.

This section verifies the position PI module and position loop with a QEP or a resolver. For this loop to work properly, Section 9.13 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 or absolute encoder (EnDat or Biss-C) 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.

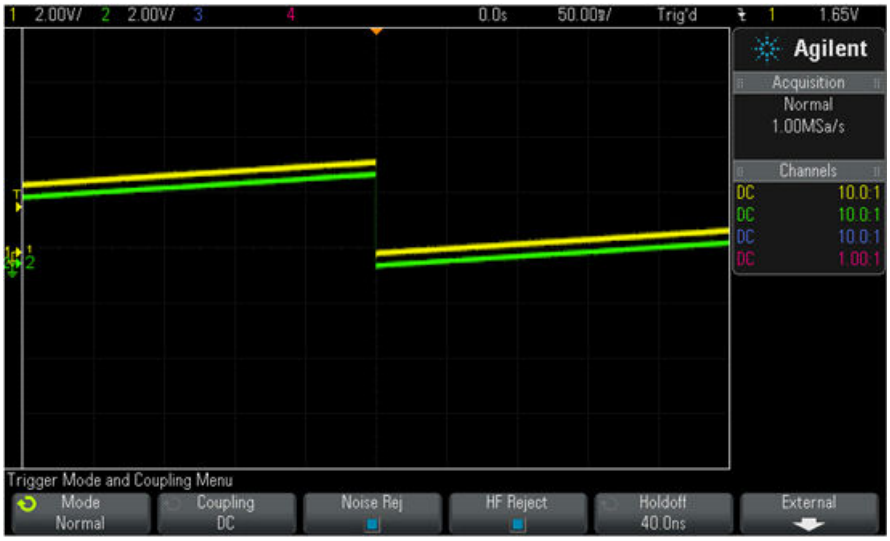
1. Open HVPM_Sensored-Settings.h and select level 5 incremental build option by setting the BUILDLEVEL to LEVEL5 (#define BUILDLEVEL LEVEL5).
2. Right Click on the project name and click Rebuild Project.
3. Click on debug button, reset the CPU, restart, enable real-time mode and run, once the build is 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.
5. Set ‘RunMotor’ to 1 in the Expressions window. Setting this flag runs the motor through predefined motion profiles and position settings as set by the ‘refPosGen()’ module.

The refPosGen() 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. Therefore, 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’ to pass through before restarting 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:

1. Compile, load, and run the program with real-time mode.
2. Add variables ‘pi_pos’, ‘posArray’, ‘ptrMax’ and ‘posSlewRate’ to the expressions window
3. Gradually increase the voltage at the variac to get the appropriate DC-bus voltage.
4. Set RunMotor = 1 to run the motor. The motor should be turning to follow the commanded position. if the motor doesn’t turn properly, see (1).
5. The parking positions in ‘posArray’ can be changed to different values to see if the motor turns as many rotations as set.
6. The number of parking positions ‘ptrMax’ can also be changed to set a rotation pattern
7. Position slew rate can be changed using ‘posSlewRate’. This represents the angle (in pu) per sampling instant.
8. 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.
9. Bring the system to a safe stop as described at the end of build 1 by reducing the bus voltage, taking the controller out of real time mode and reset. Now the motor is stopping

In Figure 9-18, the position reference and position feedback are plotted. It can be seen that they are aligned with negligible lag, which may be attributed to the software. If the Kp, Ki gains of the position loop controller are not chosen properly, it may lead to oscillations in the feedback or a lagged response.



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 or EnDat or Biss-C, the above may not be an issue as they give unique angular value for each position

Figure 9-18. Scope Plot of Reference Position to Servo and Feedback Position

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)

11 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (February 2016) to Revision A (May 2021)	Page
• Updated the numbering format for tables, figures and cross-references throughout the document	2

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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