TI Designs

Sensored Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors Using LAUNCHXL-F28377S and BOOSTXL-DRV830x

TI Designs

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize the system. TI Designs help *you* accelerate your time to market.

Design Resources

TIDM-SERVO-LAUNCHXS	Tool Folder Containing Design Files
LAUNCHXL-F28377S	Tool Folder
BOOSTXL-DRV8301	Tool Folder
BOOSTXL-DRV8305EVM	Tool Folder

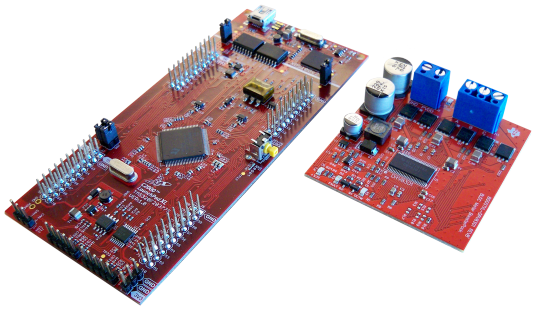
TI E2E™ Community — ASK Our E2E Experts

Design Features

- Position Servo Control of PMSM with QEP
- Power Rating:
  - BOOSTXL-DRV8301: 24 V/ 14 A
  - BOOSTXL-DRV8305EVM: 45 V/ 20 A
- LaunchPad™: LAUNCHXL-F28377S
- Incremental Software Build
- Compact

Featured Applications

- CNC
- Robotics
- Industrial Controls
- Automation



An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

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

Copyright © 2015, Texas Instruments Incorporated

1 Introduction

A brushless Permanent Magnet Synchronous motor (PMSM) has a wound stator, a permanent magnet rotor assembly and internal or external devices to sense rotor position. The sensing devices provide position feedback 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. The elimination of brushes reduces noise, EMI generation and suppresses brush maintenance.

This design guide presents a solution to control a permanent magnet synchronous motor using the TMS320F28377S LaunchPad™ and integrated motor driver DRV830x BoosterPack™. The LaunchPad and BoosterPack 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 field orientated control (FOC). This algorithm maintains efficiency in a wide range of speeds and takes into consideration transient torque changes controlling the flux directly from rotor coordinates. This design guide 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.

1.1 Permanent Magnet Motors

There are primarily two types of three-phase 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 brushes slide over the rings. 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 shows the three-phase synchronous motor with one permanent magnet pair pole rotor.

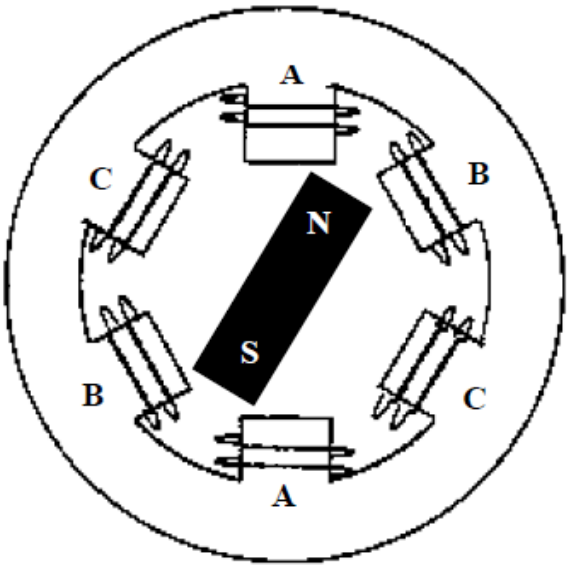


Figure 1. Three-Phase Synchronous Motor With One Permanent Magnet Pair Pole Rotor

1.2 Synchronous Motor Operation

- Synchronous motor construction
  - Permanent magnets are rigidly fixed to the rotating axis to create a constant rotor flux. The stator

Copyright © 2015, Texas Instruments Incorporated

windings when energized with three-phase voltages create a rotating electromagnetic field.

– To control the rotating magnetic field, it is required 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 90º 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.

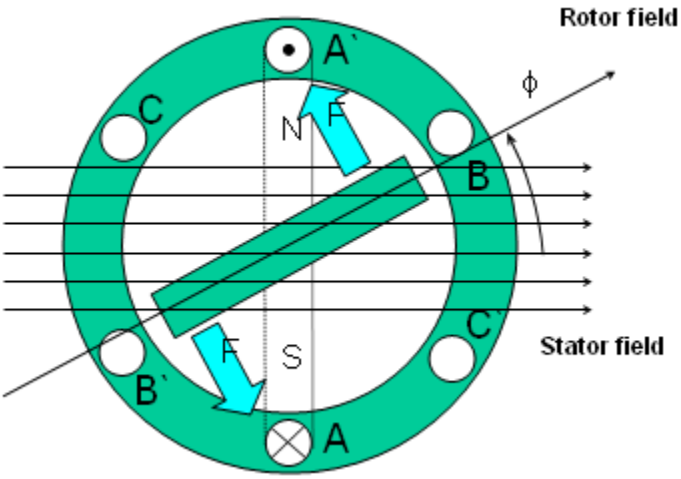


Figure 2. The Interaction Between the Roating Stator Fulx, and the Rotor Flux Produces a Torque Which Causes the Motor to Rotate

Copyright © 2015, Texas Instruments Incorporated

2 Field Oriented Control

Introduction

To achieve better dynamic performance, a more complex control scheme must 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

To understand the spirit of the FOC technique, an overview of the separately excited direct current (DC) Motor is required. 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 are 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 windings are managed to keep the flux produced by the rotor windings orthogonal to the stator field and current.** Figure 3 shows the separately excited DC motor model, flux and torque are independently controlled and the current through the rotor windings determine how much torque is produced.

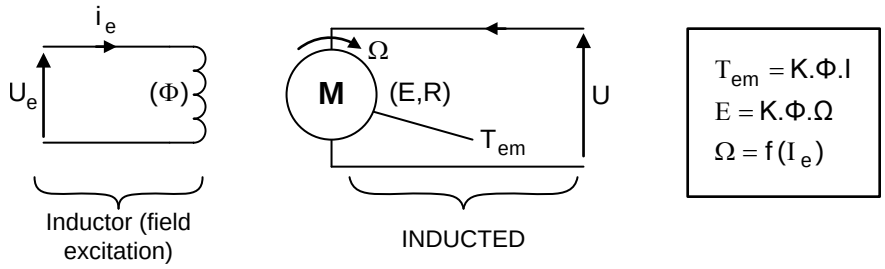


Figure 3. Separately Excitated DC Motor Model...

In a separately excitated 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 shown in Equation 1.

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

Copyright © 2015, Texas Instruments Incorporated

Equation 1 shows that the torque is at a maximum for any given stator and rotor magnetic fields when they are orthogonal. To ensure this condition all of the time, if the fulx is oriented correctly, the torque ripple is reduced, and a better dynamic response is ensured. However, the constraint is to know the rotor position: this is 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 are applied to get rid of position sensor.

A 3-phase PM synchronous machine is 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, impacts the strength of magnetic field and the current in Q axis, called a quadrature current, will interact with the magnetic flux in D-axis to produce torque. 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 is 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 condition (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 current reference (the torque component and the flux component of the stator current).
- The ease of the torque control in the (d,q) reference frame, the expression of the torque is:

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

By maintaining the amplitude of the rotor flux (ΨR)at a fixed value, a linear relationship between the torque and the torque component of the stator current vector (is_q) is established. The torque is then controlled by controlling the torque component.

Copyright © 2015, Texas Instruments Incorporated

Space Vector Definition and Projection

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

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

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

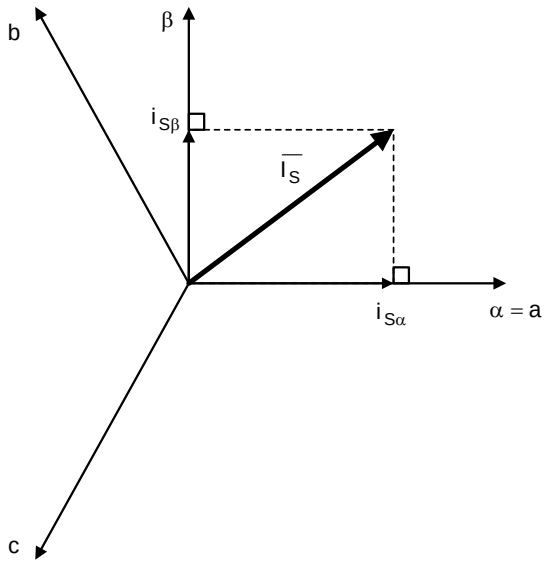


Figure 4. Stator Current Space Vector and its Component (in a,b,c)

Where (a,b,c) are the three phase system axes. The current space vector depicts the three-phase sinusoidal system. It still needs to be transformed into a two time invariant coordinate system. The transformation is split into two steps:

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

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 the axis a and the axis α are in the same direction, as shown in Figure 5.

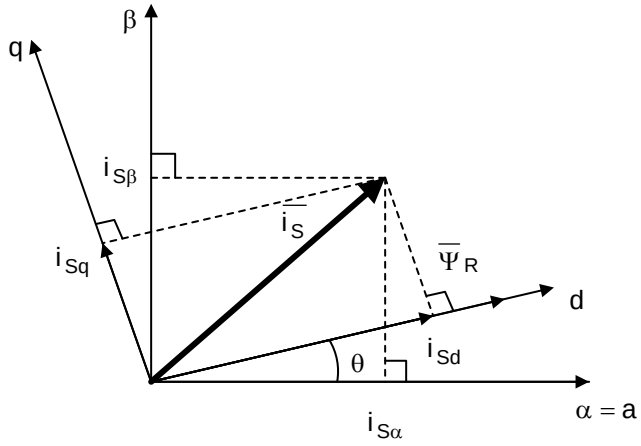


Figure 5. Stator Current Space Vector and its Components in the Stationary Reference Frame

Copyright © 2015, Texas Instruments Incorporated

Equation 4 shows the projection that modifies the three-phase system into the (α,β) two-dimension orthogonal system.

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

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

The $(\alpha,\beta)\rightarrow$ (d,g) Projection (Park Transformation)

The Park Transformation is the most important transformation in the FOC. This projection modifies a two-phase orthogonal system (α,β), in the d,g rotating reference frame. If the d axis is aligned with the rotor flux, as shown in Figure 6, for the current vector, the relationship from the two reference frame.

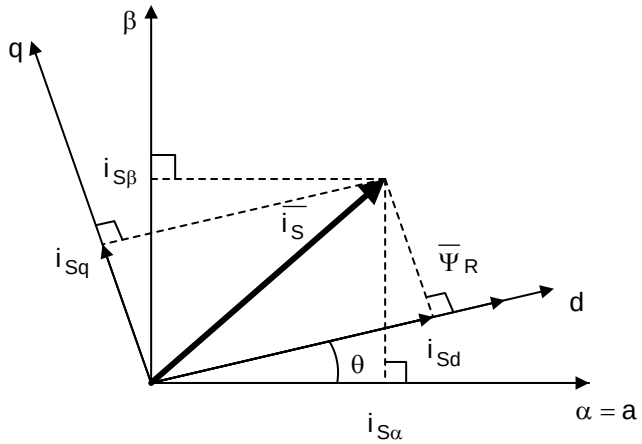


Figure 6. Stator Current Space Vector and its Component in (a,b) and in d,q Rotating Reference Frame

Figure 6 shows the stator current space vector and it's 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:

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

These components depend on the current vector (α,β) components and on the rotor flux position. If the right rotor flux position is known at this point, 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 are controlled independently.

Copyright © 2015, Texas Instruments Incorporated

3 The Basic Scheme for the FOC

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

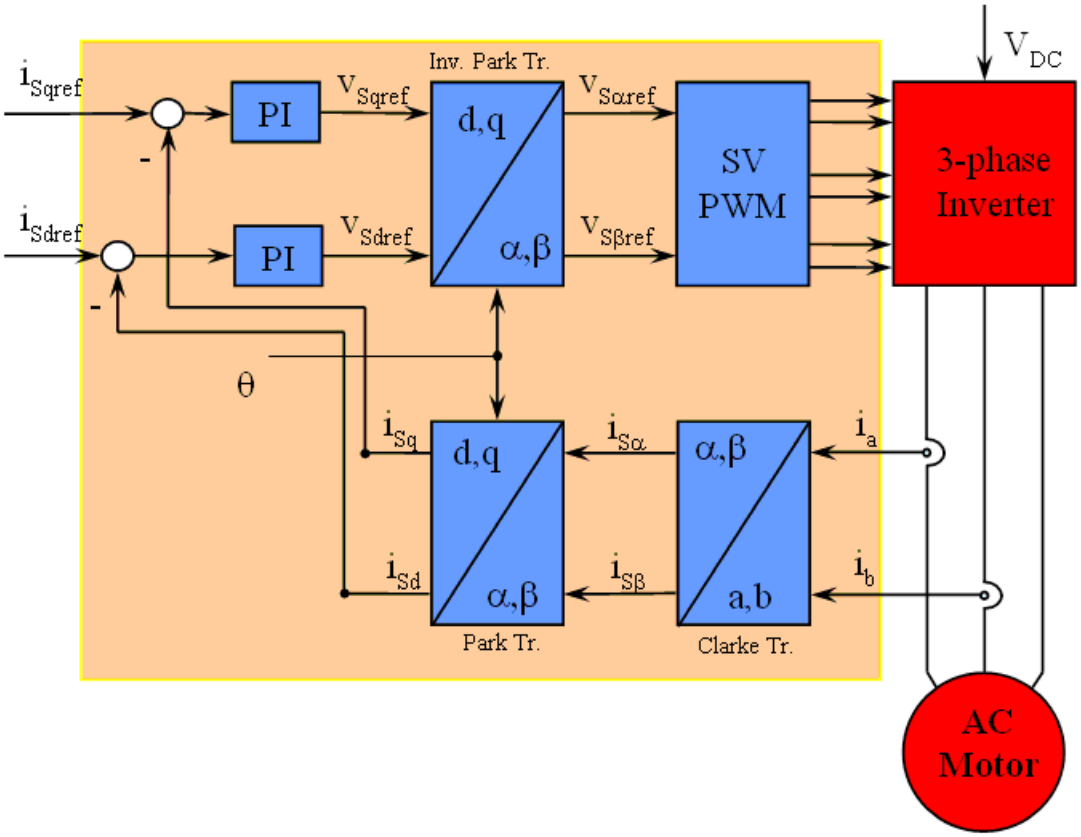


Figure 7. Basic Scheme of FOC for AC Motor

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 the rotor flux position are the inputs of the Park transformation, which transforms the 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_{sdref} (the flux reference) and i_{sqref} (the torque reference). At this pont, this control structure shows an interesting advantage, it is used to control either synchronous or HVPM machines by changing the flux reference and obtaining rotor flux position.

In synchronous permanent magnet motor, the rotor flux is fixed as determined by the magnets, thus, there is no need to create it. When controlling a PMSM, i_{sdref} is set to zero. As ACIM motors require a rotor flux creation to operate, the flux reference must not be zero. This conveniently solves one of the major drawbacks of the classic control structures: the probability from asynchronous to synchronous drives. The torque command i_{sqref} must be connected to the output of the speed regulator. The outputs of the current regulators are V_{sdref} and V_{sqref} and are applied to the inverse Park transformation. Using the position of the rotor flux, this projection generates V_{sqref} 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: Both Park and Inverse Park transformations require the rotor flux position. Obtaining this rotor flux position depends on the AC machine type (synchronous or asynchronous machine). Section 3.1 discusses rotor flux position considerations.

Copyright © 2015, Texas Instruments Incorporated

3.1 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 the d-axis and i_{sd} and i_{sq} will represent incorrect flux and torque components of the stator current. Figure 8 shows the (a,b,c), (α,β), and *(d,q)* reference frames as well as the correct position of the rotor flux, the stator current and stator voltage space vector that rotates with the d,q reference frame at synchronous speed.

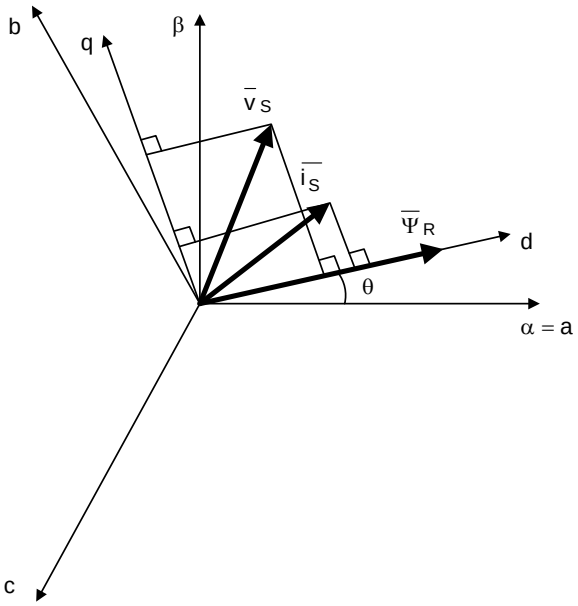


Figure 8. Vector Diagram in d-q and Stationary Reference Frames

Copyright © 2015, Texas Instruments Incorporated

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

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. Figure 9 shows the overall block diagram of this project.

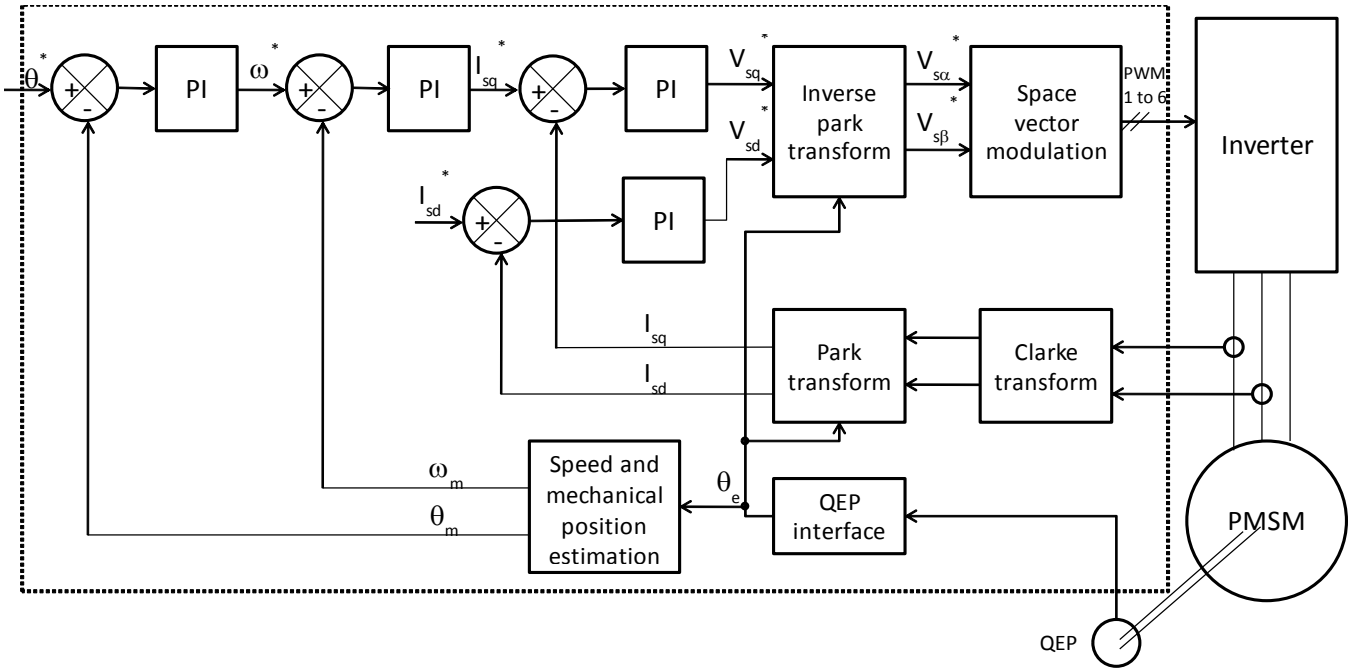


Figure 9. Overall Block Diagram of Sensored Field Oriented Control

3.2 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 ADC, ePWM, QEP, eCAP, and more. 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.

With the 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.
- Uses advanced control algorithms to reduce torque ripple, resulting in a longer life of the motor, and low vibration.
- Advanced control algorithms reduce harmonics generated bu the inverter, resulting in lower filter cost.

Copyright © 2015, Texas Instruments Incorporated

- Use of sensorless algorithms eliminates the need for a speed or position sensor.
- Decreases the number of look-up tables, resulting in the amount of required memory.
- Better performance due to the real-time generation of smooth near-optimal reference profiles and move trajectories.
- Generation of high resolution PWms is possible with the use of the ePWM peripheral for controlling the power-switching inverters.
- Provides a single-chip control system.
- For advanced controls, the C2000 controllers 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 algorithm adapts itself in real time to variations in system behavior.
  - Performs parameter identification for sensorless control algorithms, self-commissioning, and online parameter estimation update.
  - Performs advanced torque ripple and acoustic noise reduction.
  - Provides diagnostic monitoring with spectrum analysis by observing the frequency spectrum of mechnical vibrations. Failure moses are 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.

4 TI Literature and Digital Motor Control (DMC) Library

The Digital Motor Control (DMC) library is composed of functions represented as blocks. The 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 or higher.
- C:\TI\controlSUITE\libs\app_libs\motor_control\drivers\f2803x_v2.0 or higher.

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. 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 step by step. 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 opens the respective header file and changes the macro definition, if required. In the macro definitions, there is a backslash \ at the end of each line, as shown in a typical macro definition, which means that the code continue in the next line. Any character including invisible ones like “space” after the backslash will cause compilation error. 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 easily converts the macro definition to a C functions.

A Typical Macro Definition:

Copyright © 2015, Texas Instruments Incorporated

define **PARK_MACRO**(v)

v.Ds= **_IQmpy** (v.Alpha, v. Cosine) + **_IQmpy** (v.Beat, v.Sine) ; \

v.Qs= **_IQmpy** (v.Beta, v.Cosine) - **_IQmpy** (v.Alpha, v. Sine) ; \

5 System Overview

This document describes the C real-time control framework used to demonstrate the sensored 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 modules, listed in Table 1.

Table 1. System Overview

MACRO NAMES	EXPLANATION
CLARKE	Clarke Transformation
PARK/IPARK	Park and Inverse Park Transformation
PI	PI Regulators
PID	PID Regulator
PI_POS	PI Regultor for Position Loop
RC	Ramp Controller (slew rate limiter)
RG	Ramp/Sawtooth Generator
QEP	QEP Drive
SPEED_FR	Speed Measurement
SVGEN	Space Vector PWM WIth Quadrature Control (includes IClark Transformation)
PWM	PWM Drives

Figure 10 shows the overall system implementing sensored Field Oriented Control (FOC) of Permanent Magnet Synchronous Motor (PMSM). The PMSM motor is driven using a DRV830x BoosterPack (BOOSTXL-DRV8301/5) that has an integrated motor driver device DRV8301/5. LaunchPad LAUNCHXL-F28377S, that has a TMS320F28377S CPU, is used to generate three sets of complementary pulse width modulation (PWM) signals for the inverter. The three phase currents of the PMSM are measured from the inverter using shunt resistors connected to the bottom of inverter half bridges, which are amplified by shunt current amplifiers of the DRV830x driver. The amplified shunt voltage is then measured using ADCs. The DC-bus voltage of the inverter, as well as the motor phase voltages, is measured using ADC.

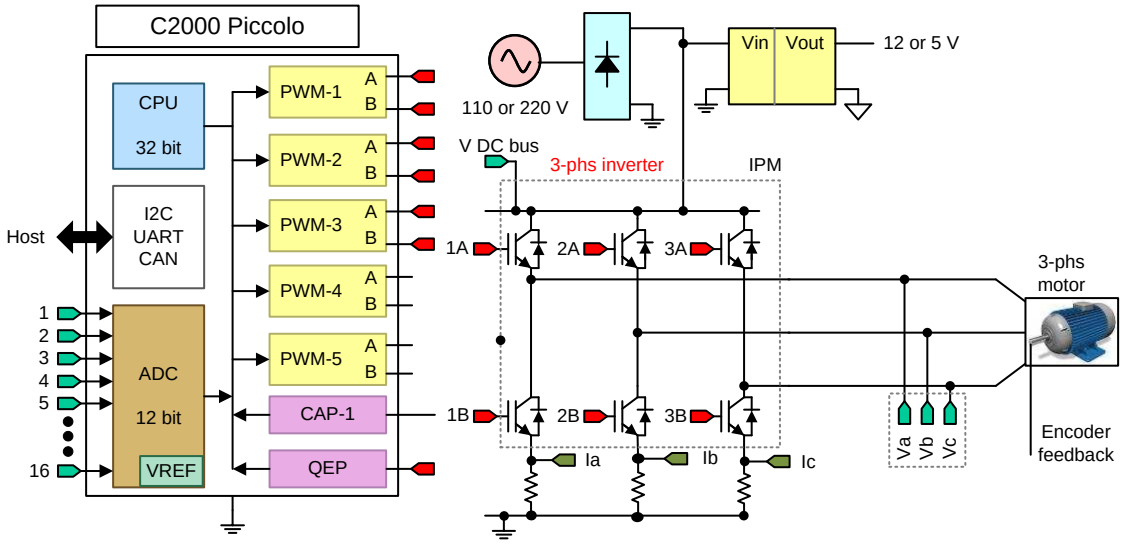


Figure 10. A 3-ph PM Motor Drive Implementation

Copyright © 2015, Texas Instruments Incorporated

Figure 11 shows the software flow.

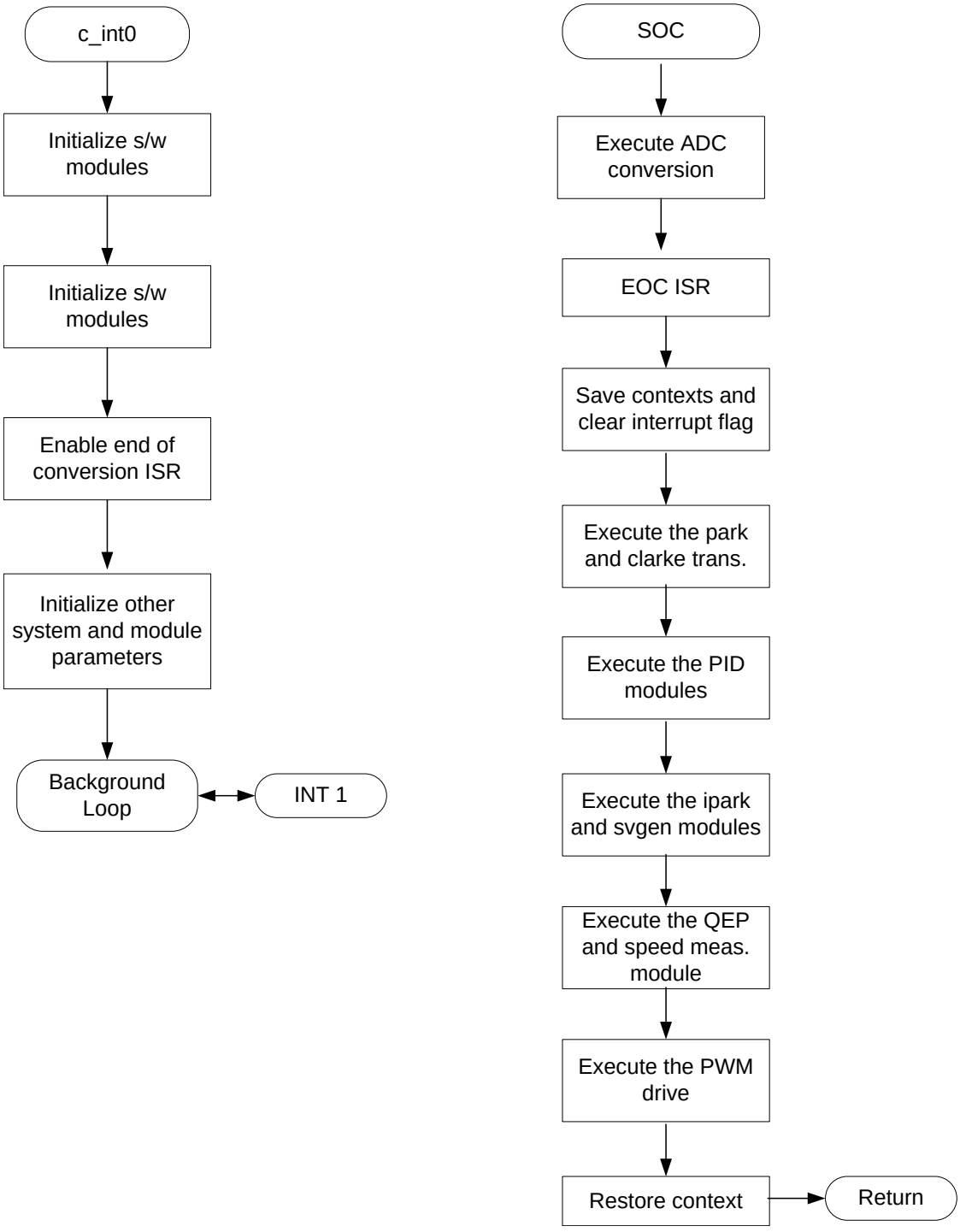


Figure 11. Software Flow Sequence

Copyright © 2015, Texas Instruments Incorporated

6 Hardware Configuration

The hardware for this project is based on the LaunchPad LAUNCHXL-F28377S and BoosterPacks based on integrated motor drive controller DRV8301 and DRV8305, namely, BOOSTXL-DRV8301 and BOOSTXL-DRV8305 EVM respectively. DRV830x parts have integrated three phase gate drivers with current sense amplifiers for sensing voltages across current shunts tied to the bottom of inverter legs. It also has programmable over current protection circuits, and the features are programmable through a SPI interface. For more details about the evaluation hardware, referto the following documents:

- LAUNCHXL-F28377S Overview User's Guide (SPRUI25A)
- Motor Drive BoosterPack Quick Start Guide: BOOSTXL-DRV8301 (SLDC0006)
- BOOSTXL-DRV8301 Hardware User's Guide (SLVU974)
- BOOSTXL-DRV8305 EVM User's Guide (SLVUAI8)

System Hardware Connections:

7 Software Setup for Launch XS Projects

Typical of TI LaunchPad's and BoosterPacks, the BoosterPack BOOSTXL-DRV830x mounts onto the LaunchPad LAUNCHXL-F28377S. This connects the analog/digital IOs of the BoosterPack to the appropriate IOs of the CPU

For this project, the BoosterPack is mounted on to headers J1, J3, J2 and J4 of the LaunchPad. The orientation of BoosterPack is so the power terminals are on the same side as the LaunchPad's USB terminal. The QEP signals from motor should be connected to header QEP-B of the LaunchPad, while the motor phase terminals are connected to the power connectors with the labels MOTA, MOTB, MOTC for DRV8301 BoosterPack or A,B,C for DRV8305 BoosterPack. Figure 12 shows the LaunchPad connection outline diagram.

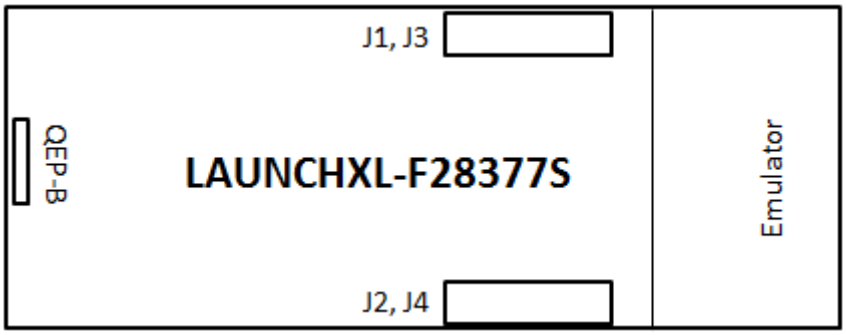


Figure 12. LaunchPad Connection Outline Diagram

Copyright © 2015, Texas Instruments Incorporated

The maximum input DC voltage for DRV8301 BoosterPack is 24 V, and is 48 V for the DRV8305. Even though these two BoosterPacks are interchangable on a LaunchPad, care must be taken when swapping in DRV8301 BoosterPack by setting the dc bus voltage to not exceed 24 V before powering on the system.

Powering the LaunchPad:

For immediate reference, Figure 13 shows the layout of LAUNCHXL-F28377S and the pinouts of LAUNCHXL-F28377S, BOOSTXL-DRV8301 and BOOSTXL-DRV8305.

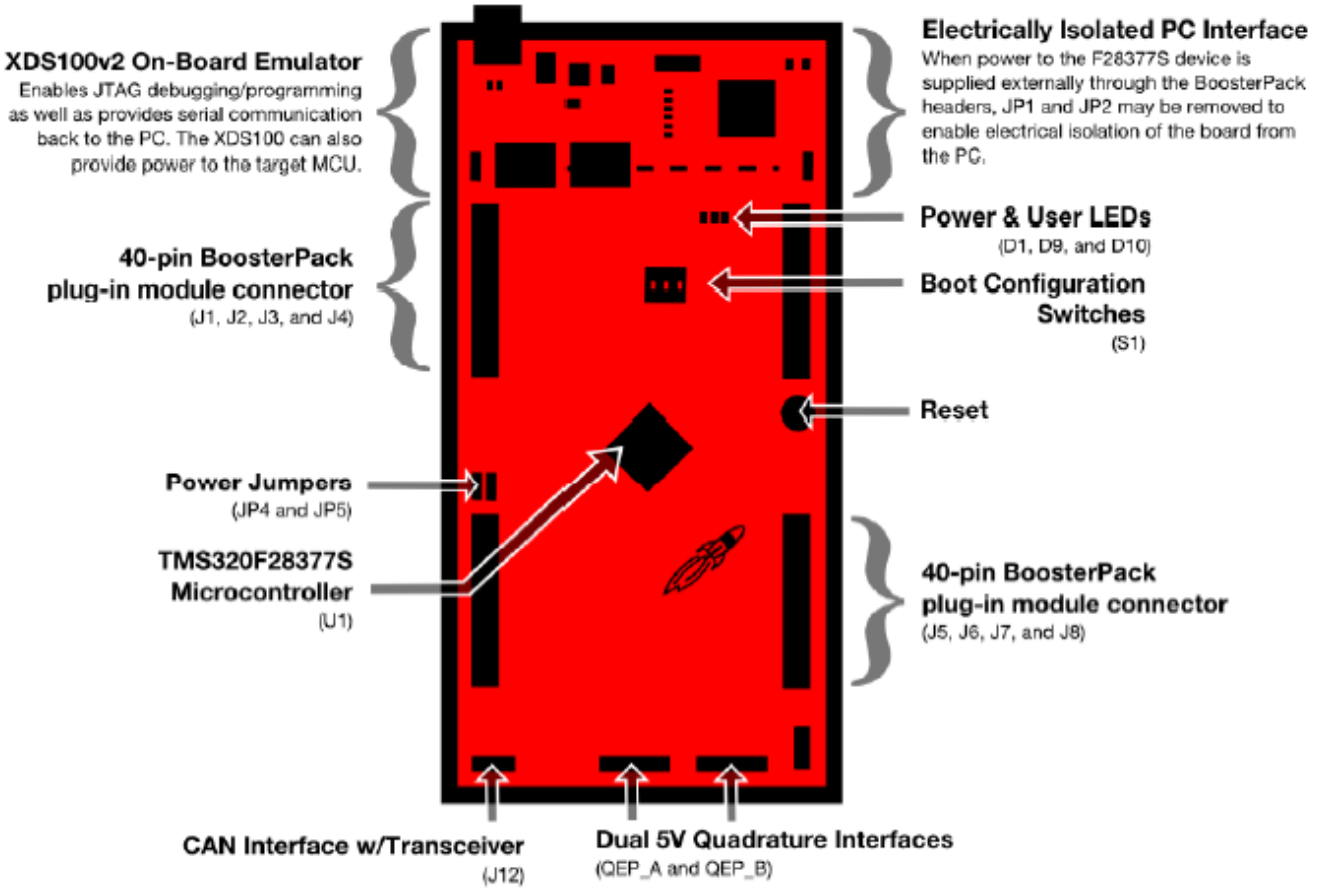


Figure 13. Layout of the LAUNCHXL-F28377S

For early code development without the BoosterPack, retain the jumpers JP1 and JP2 of the LaunchPad. This feeds 3.3-V power from the emulator section to the CPU. Alternately, the CPU may be powered from an external source through the header pins of header J1, J3 or J5 and J7, in which case remove jumpers JP1 and JP2. Make sure to remove these jumpers or external power supply before mounting the DRV BoosterPack because the BoosterPack is designed to power the LaunchPad when mounted. Jumpers JP4 and JP5 on the LaunchPad are meant to provide power supply to any booster pack mounted on headers J5, J6, J7 and J8. However, if the booster pack generates its own power supply, then do not populate JP4 and JP5. If there is no booster pack on J5-J8, and it does not matter if JP4 and JP5 are absent or present.

CAUTION:

If these jumpers are present when the BoosterPack is powered, then it may:

1. Potentially damage the development computer as it will tie the computer's USB to the external power supply GND.
2. Create circulating current between the 3.3-V power supplies generated by the DRV BoosterPack and the emulator powered by the USB.

Copyright © 2015, Texas Instruments Incorporated

8 LAUNCHXL F28377S

Table 2 shows the pin out details of the BoosterPack heads and the associated GPIO Mux signals.

Table 2. Pin Out and GPIO Pin MUX Options–Header J1, J3

MUX VALUE				J1 PIN	J3 PIN	MUX VALUE			
3	**2**	**1**	**0**	J1 PIN	J3 PIN	**0**	1	**2**	**3**
			+3.3V	1	21	+5 V			
	EM1D13		GPIO71	2	22	GND			
EM1DQM2	EM1A17		GPIO90	3	23	ADCIN14			
EM1DQM1	EM1A16		GPIO89	4	24	ADCINB1			
	EM1A3		GPIO41	5	25	ADCINB4			
				6	26	ADCINB2			
EM2D8	EM1D24	MCLKRB	GPIO60	7	27	ADCINA0			
EM2D7	EM1D23	MFSRB	GPIO61	8	28	ADCINB0			
			GPIO43	9	29	ADCINA4			
				10	20				

Table 3 shows the pin out details of the BoosterPack heads and the associated GPIO Mux signals.

Table 3. Pin Out and GPIO Pin MUX Options–Header J4, J2

MUX VALUE				J4 PIN	J2 PIN	MUX VALUE			
3	2	1	0	J4 PIN	J2 PIN	0	1	2	3
MDXB	CANTXB	EPWM7A	GPIO12	40	20	GND			
MDRB	CANRXB	EPWM7B	GPIO13	39	19	GPIO4	EPWM3A		
MCLKXB	SCITXDB	EPWM8A	GPIO14	38	18	GPIO62	SCIRXDC	EM1D2	EM2D6
MFSXB	SCIRXDB	EPWM8B	GPIO15	37	17				
OUTPUTXBAR7	CANTXB	SPISOMIA	GPIO16	36	16	RESET #			
OUTPUTXBAR6	CANRXB	SPISOMIA	GPIO17	35	15	GPIO58	MCLKRA	EM1D26	EM2D10
CANTXB	MDXA	EQEP1A	GPIO20	34	14	GPIO59	MFSRA	EM1D25	EM2D9
CANRXB	MDRA	EQEP1B	GPIO21	33	13	GPIO72		EM1D12	
			DAC 1	32	12	GPIO73		EM1D11	XCLKOUT
			DAC 2	31	11	GPIO78		EM1D6	

Table 4 shows the pin out details of the BoosterPack heads and the associated GPIO Mux signals.

Copyright © 2015, Texas Instruments Incorporated

Table 4. Pin Out and GPIO Pin MUX Options–Header J5, J7

MUX VALUE				J5 PIN	J7 PIN	MUX VALUE			
3	2	1	0	J5 PIN	J7 PIN	0	1	2	3
			+3.3V	41	61	+5V			
				42	62	GND			
EM1RAS	EM1A14		GPIO87	43	63	ADCIN15			
			GPIO86	44	64	ADCINA2			
				45	65	ADCINA5			
				46	66	ADCINB5			
			GPIO65	47	67	ADCINA3			
				48	68	ADCINB3			
			GPIO69	49	69	ADCINA4			
			GPIO66	50	70				

Table 5 shows the pin out details of the BoosterPack heads and the associated GPIO Mux signals.

Table 5. Pin Out and GPIO Pin MUX Options–Header J8, J6

MUX VALUE				J8 PIN	J6 PIN	MUX VALUE			
3	2	1	0	J8 PIN	J6 PIN	0	1	2	3
		EPWM1A	GPIO2	80		GND			
		EPWM1B	GPIO3	79		GPIO91		EM1A18	EEM1DQM3
ADCSOCB0	CANRXB	EPWM6A	GPIO10	78					
OUTPUTXBAR7	SCIRXDB	EPWM6B	GPIO11	77	57				
CANRXA	SCITXDB	SPICLKA	GPIO18	76	56	RESET #			
CANTXA	SCIRXDB	SPISTEA	GPIO19	75	55	GPIO63	SCITXDC	EM1D21	EM2D5
				74	54	GPIO64		EM1D20	EM2D4
				73	53	GPIO99			EM2A1
				72	52	GPIO92		EM1D19	EM1BA1
				71	51				

Copyright © 2015, Texas Instruments Incorporated

8.1 Pinouts of BOOSTXL-DRV830x

Figure 14 shows the layout and pinout diagram of the BOOSTXL-DRV8301, and Figure 15 shows the layout and pinout diagram of BOOSTXL-DRV8305.

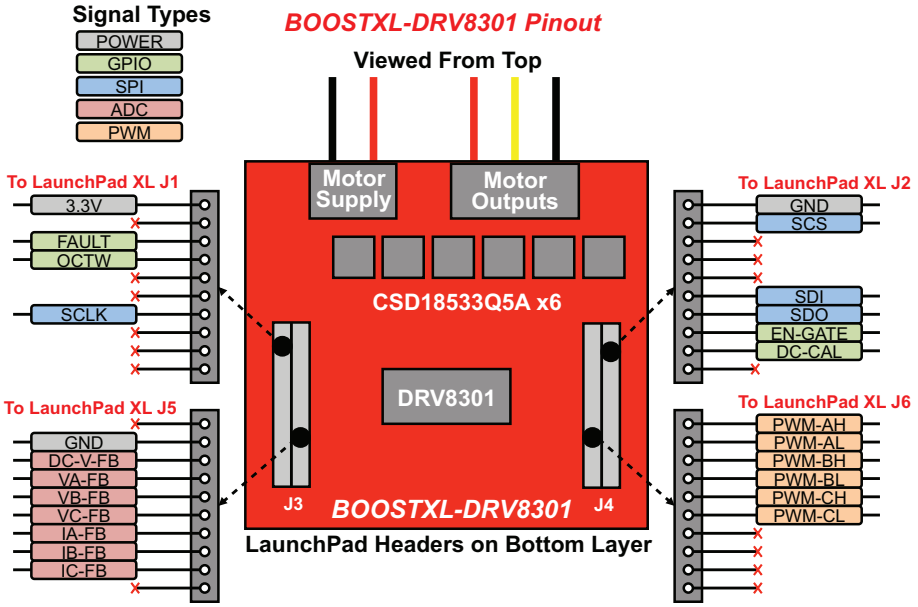


Figure 14. Layout and Pinout Diagram of the BOOSTXL-DRV8301

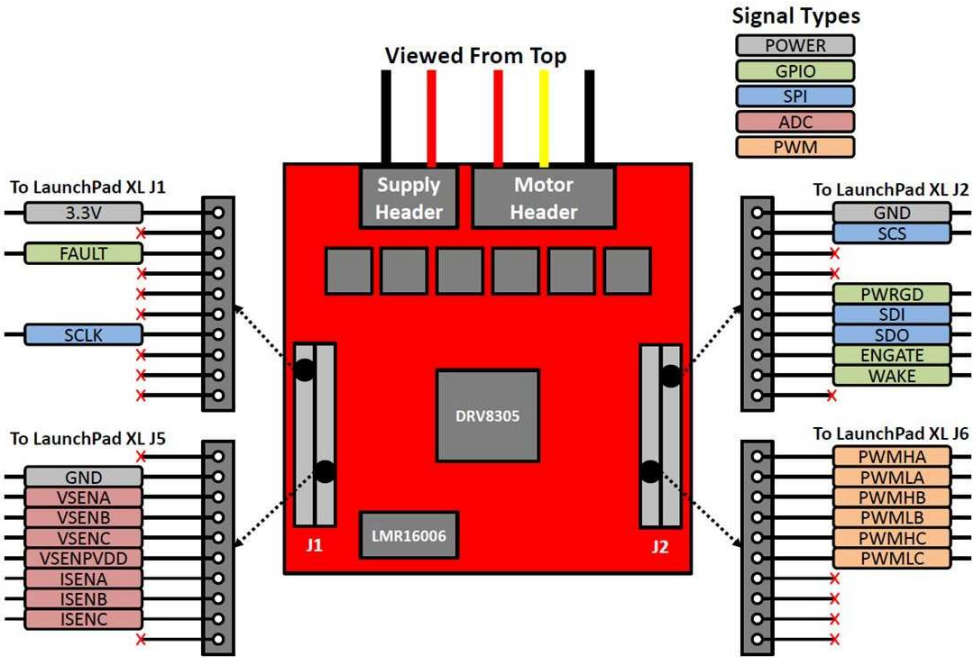


Figure 15. Layout and Pinout Diagram of the BOOSTXL-DRV8305

CAUTION:

The BOOSTXL-DVR8301 has a permitted DC voltage of 24 V, while the BOOSTXL-DRV8305 has up to a 48 V range. If the BOOSTXL-DRV8305 is replaced with a BOOSTXL-DRV8301, ensure that the input DC voltage is set within the safe range of the BOOSTXL-DRV8301.

Copyright © 2015, Texas Instruments Incorporated

NOTE: BOOSTXL-DRV8301 has a permitted dc voltage range of 24-V, while BOOSTXL-DRV8305 has up to 48-V. If BOOSTXL-DRV8305 is replaced with a BOOSTXL-DRV8301, ensure that the input dc voltage is set within the safe range of BOOSTXL-DRV8301.

9 Software Setup for LaunchXS Projects

Installing Code Composer and ControlSUITE

1. If not already installed, install the Code Composer v6.x or later from http://www.ti.com/tool/CCSTUDIO.
2. Go to http://www.ti.com/controlsuite and run the controlSUITE installer.
3. Allow the installer to download and update any automatically checked software for C2000.

Setup Code Composer Studio for Project Evaluation

1. Open Code Composer Studio (CCS).
2. Once CCS opens, the workspace launcher may ask the user to select a workspace location.

NOTE: A workspace is a location on the hard drive where all of the user settings for the IDE (for example, which projects are open, what configuration is selected) are saved. This can be anywhere on the disk, and the location mentioned in the following steps is just for reference. Also, if this is not your first time running CCS, the following dialog may not appear.

3. Click the *browse* button.
4. Create the *C:\c2000 projects\CCSv6_workspaces\XL377s_DRV_workspace* path, and make new folders as necessary.
5. Uncheck the use this as the default and do not ask again box.
6. Click ok. Figure 16 shows the workspace launcher.

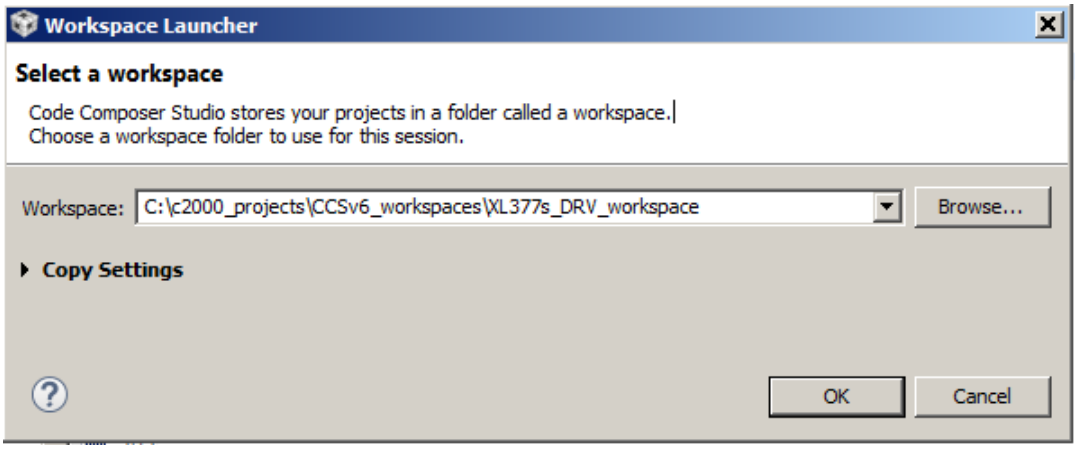


Figure 16. Workspace Launcher

Copyright © 2015, Texas Instruments Incorporated

7. A *Getting Started* tab will open with links to various tasks including creating a new project, importing an existing project, and watching on a tutorial on CCS.
8. Click *Project→Import CCS Project* to add the motor control projects into the current workspace.
9. Select the evaluation project by browsing to *C:\TI\ControlSUITE\development_kits\TIDM-SERVO-LAUNCHXS*. Figure 17 shows adding servo projects to the workspace.

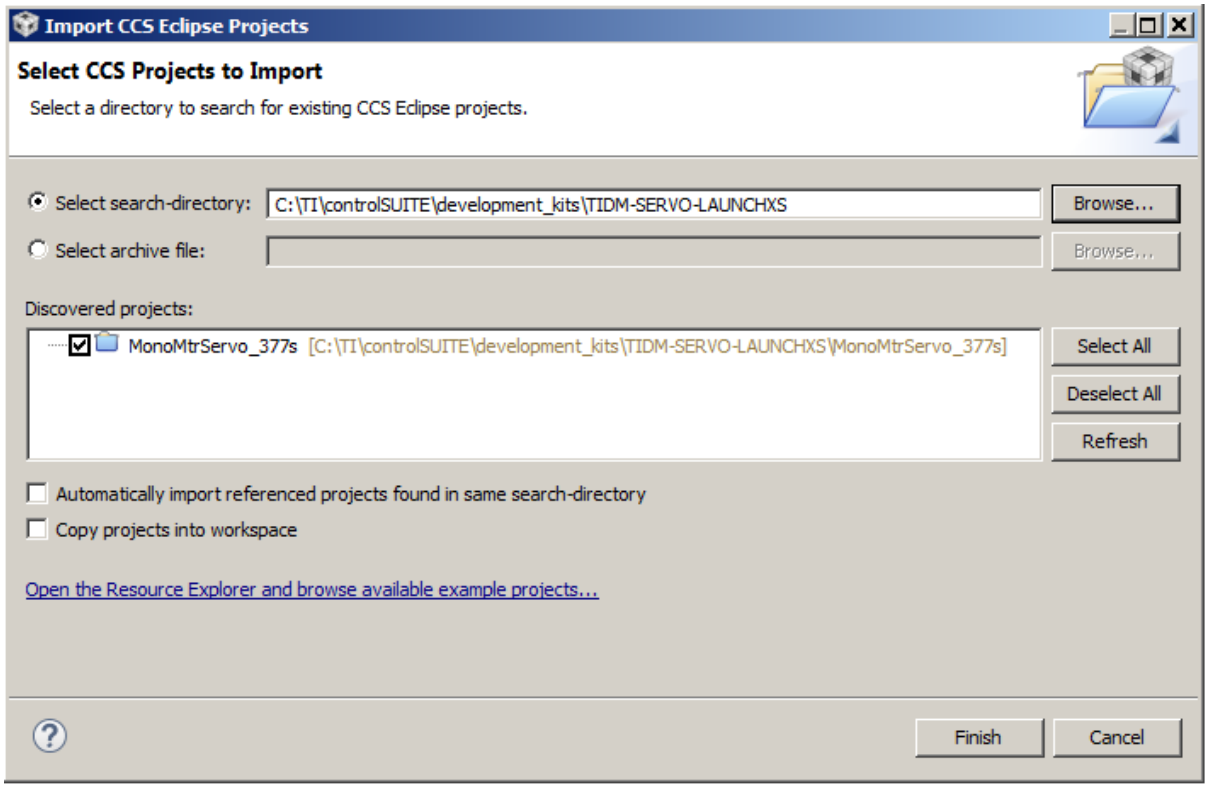


Figure 17. Adding Servo Project to Workspace

10. Click and choose the projects to import if there are multiple projects in this directory.
11. Click *Finish*.
12. All of the selected projects will be copied into the workspace.

Setting Up the Target Configuration

Setting up the target configuration helps the user to configure the Code Composer to know which MCU it connects to. The setup is already configured in *xds100v2_F2837xS.ccxml* provided in the files of this project, and the user may skip to if required.

1. Click *View→Target Configuration* to set a new configuration file.
2. The Target Configuration Window opens.
3. Name the new configuration file depending on the target device.

Copyright © 2015, Texas Instruments Incorporated

4. If the *Use Shared Location* checkbox is checked, this configuration file is stored in a common location by CCS for use by other projects as well.
5. Click *Finish*.
6. A new tab opens, as shown in .
7. Select and enter the options as shown:

Copyright © 2015, Texas Instruments Incorporated

- **Connection**–Texas Instruments XDS100v2 USB Emulator (or) Texas Instruments XDS100v2 USB Debug Probe.
- **Device**–the C2000 MCU on the control card, TMS320F28377S.
- Click *Save* and close. Figure 18 shows how configuring a new target appears.

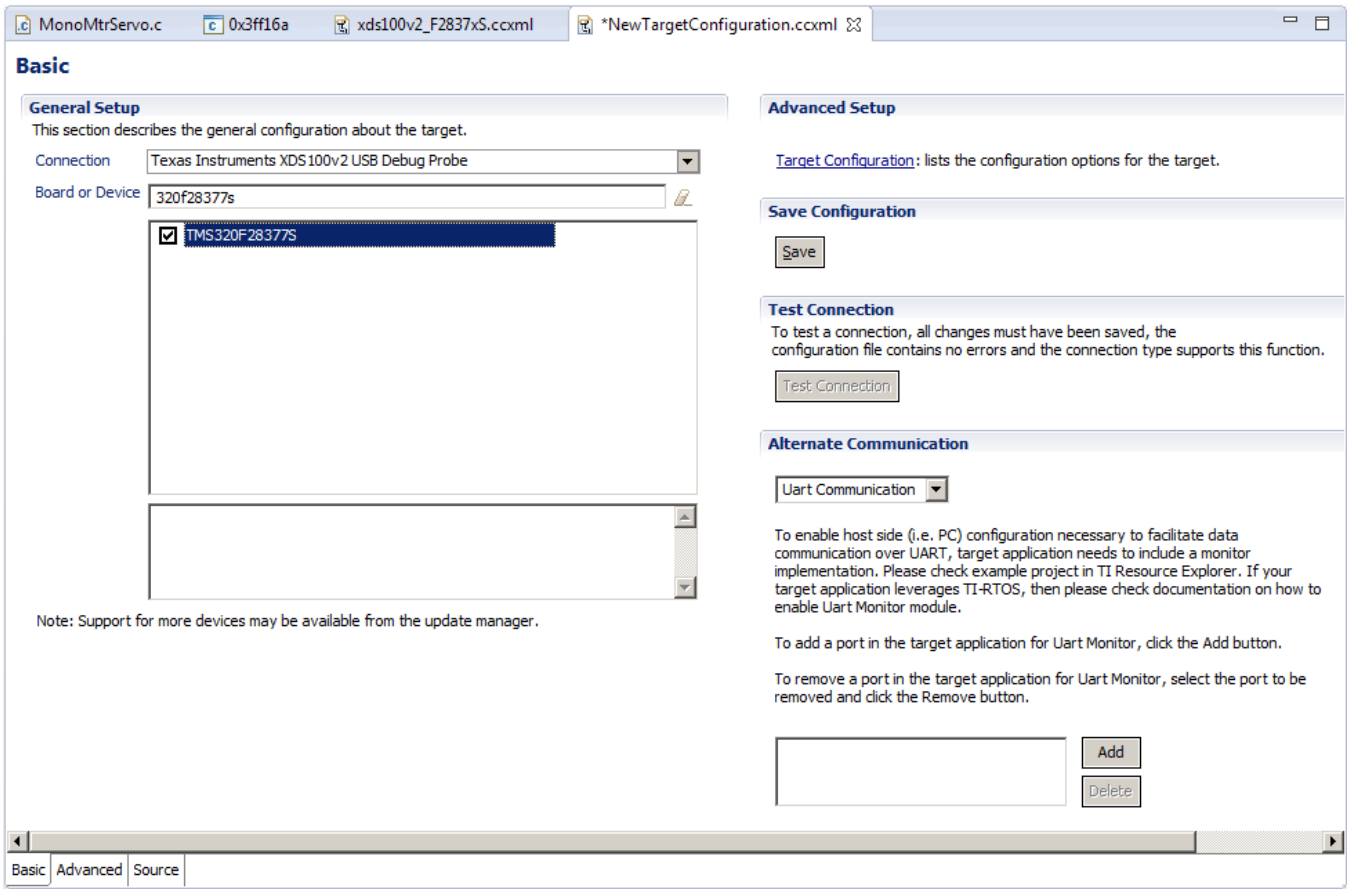


Figure 18. Configuring a New Target

8. To use the congfiguration file, click *View→Target Configurations*.
9. In the *User Defined* section, find the file that was just created.
10. Right click on the file.
11. Select *Set as Default*, and skip the remaining steps.
12. To use the configuration file supplied with the project, click *View→Target Configurations* to use the configuration file supplied with the project.
13. Expand *Projects→MonoMtrServo_377s*.
14. Right click on the file *xds100v2_F2837xS.ccxml* and *Set as Default*.
15. This tab also allows the user to reuse existing target configurations and links them to specific projects.

Copyright © 2015, Texas Instruments Incorporated

Configuring a Project

1. The project is configured to create code and run in either Flash or RAM.
2. The user may select either Flash or RAM, however, for lab experiments RAM configurations were used, and for production Flash configurations were used.
3. Right click on an individual project, as shown in Figure 19 and select *Active Build Configuration→F2837x_Ram* configuration.

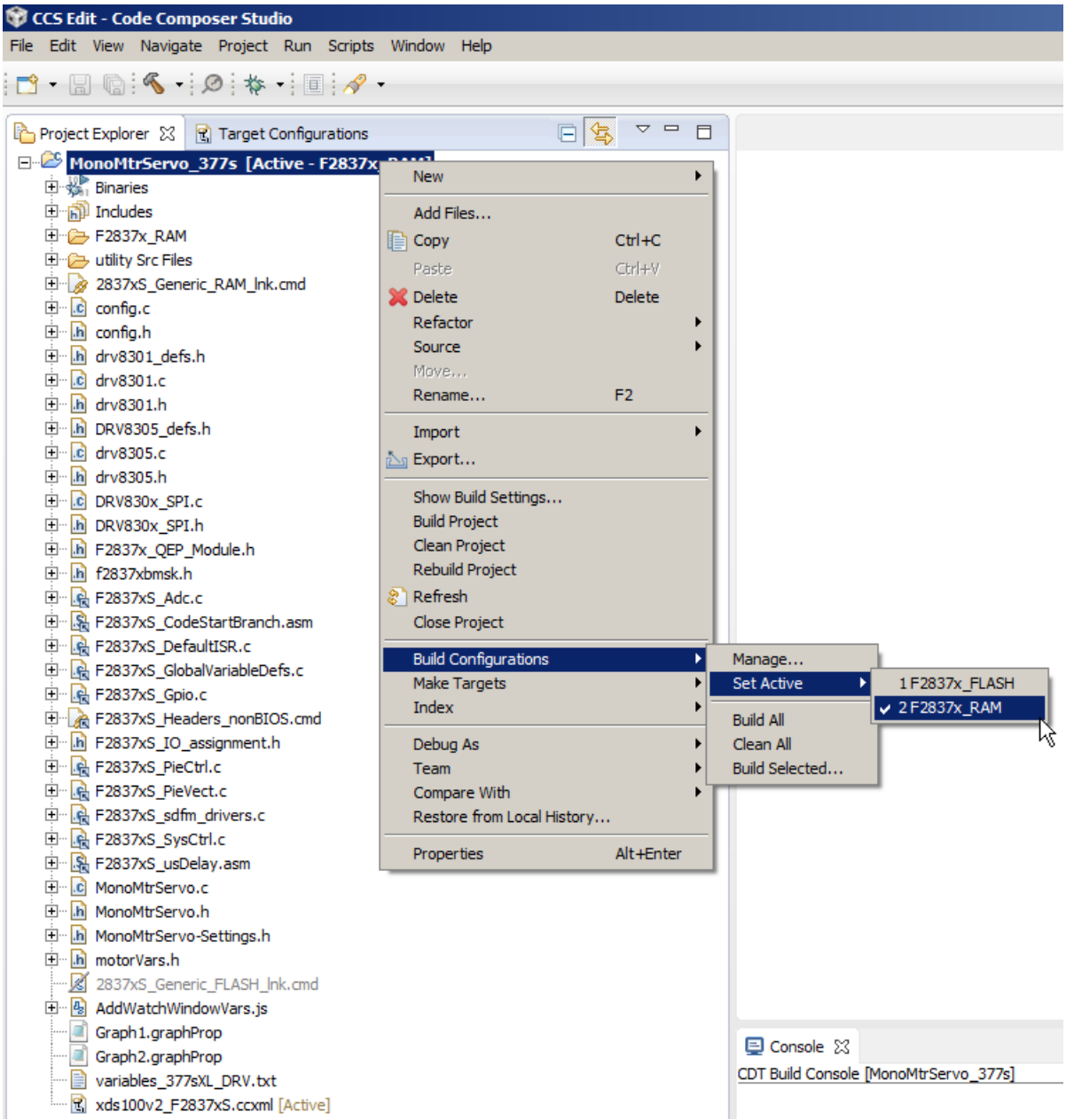


Figure 19. Selecting the F2837x_RAM Configuration

Build and Load the Project

The TI motor control software is provided with incremental builds where different components and macro blocks of the system are pieced together one by one to form the entire system. This helps in step by step debug and understanding of the system.

1. Open the file *[Project-Name]-Settings.h* from the *CCS Edit Perspective*.

Copyright © 2015, Texas Instruments Incorporated

2. Make sure that *BUILDLEVEL* is set to *LEVEL1*.
3. Set *MOTOR1_DRV* to *DRV8301* or *DRV8305* to indicate the DRV kit that will be used in the LaunchPad.
4. Save the file.

NOTE: The variable BUILDLEVEL must be redefined after testing build 1 to move on to build 2, and so on until all builds are complete.

5. Open the {Project-Name}.c file.
6. Go to the function BuildLevel1.
7. Locate the following piece of code in incremental build 1.
8. Confirm that the datalog buffers point to the correct variables.

   DlogCh1=rg1.Out;

   DlogCh2=svgen1.Ta;

   DlogCh3=svgen1.Tb;

   DlogCh4=svgen1.Tc;
9. Confirm that the datalog buffers point to the correct variables.

NOTE: The datalog buffers are large arrays that contain value-triggered fata that is displayed to a graph. In other incremental builds, different variables are placed into this buffer to be graphed.

10. The code shown in is an example of when the datalog buffers are pointed to the space vector generator module.
11. Right click on the project name.
12. Click *Rebuild Project*.
13. Watch the console window for any errors that must be fixed.
14. Warning messages about certain functions or variables not being used may occur. These warning messages may be ignored, as they may be used in another build level.

Connecting the Hardware to the Computer

1. Remove the jumpers JP1 and JP2 from LaunchPad LAUNCHXL-F28377S.
2. Mount the BoosterPack BOOSTXL-DRV830x on LaunchPad headers J1–J4.
3. **DO NOT** connect the motor.
4. Power the BoosterPack with the appropriate input DC voltage, 24-V for DRV8301 and 48-V for DRV8305.
5. Various LEDs will glow on both BoosterPacks and LaunchPads.
6. Connect the LaunchPad to the computer through a USB cable.
7. Some of the LEDs on the emulator section of the LaunchPad will light up, indicating that the emulator is on.
8. Upon successful completion of the build, click the (button) *Debug* button, located on the top-left side of the screen.
9. The IDE connects to the target.
10. Load the output file into the device, and change to the *Debug perspective*.
11. Click *Tools→Debugger Options→Program/Memory Load Options*.
12. Check *reset the target on program load or restart*.
13. Click *Remember My Settings* to make the setting permanent. This enables the debugger to reset the processor each time it reloads the program
14. Click *Enable silicon real-time mode button* (button) which also auto-selects the *Enable polite real-time mode* button. This allows the user to edit and view variables in real-time.

Copyright © 2015, Texas Instruments Incorporated

15. Do not reset the CPU without disabling the real-time options.
16. A message box may appear. If so, select *yes* to enable debug events. This will set bit 1 (DGBM but) if status register 1 (ST1) to 0.

NOTE: The DGBM is the debug enable mask bit. When the DGBM bit is set to 0, memory and register values are passed to the host processor for updating the debugger windows.

Setup Expressions Window and Graphs

1. Click *View→Expressions* on the menu bar to open an *Expressions Window* to view the variables that are used in the project.
2. Add variables to the expressions window, as shown in Figure 20.
3. The window uses the number format associated with the variables during declaration.
4. Right click to select a desired number format for the variable, and choose it.
5. Figure 20 shows a typical expressions window.

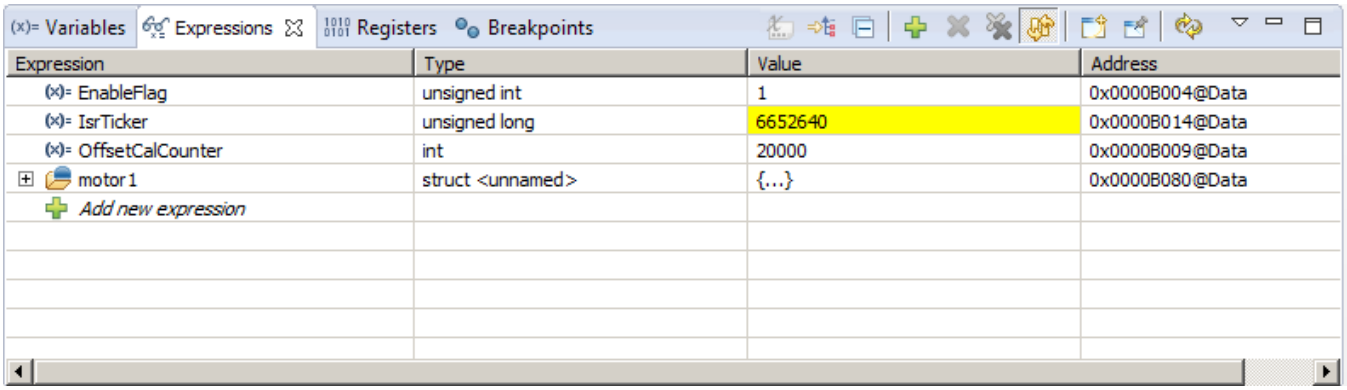


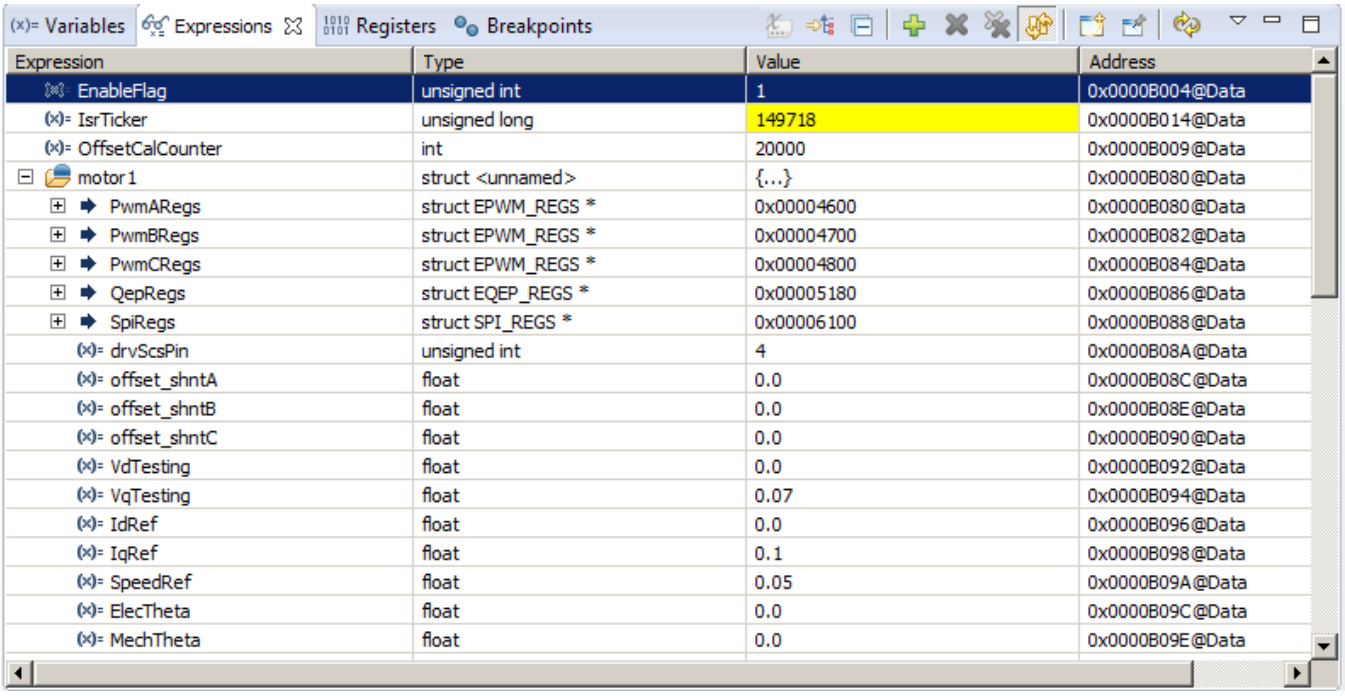
Figure 20. Configuring the Expressions Window

Copyright © 2015, Texas Instruments Incorporated

Alternatively, a group of variables may be imported into the expressions window.

1. Right click within the expressions window.
2. Click Import.
3. Browse to the .txt file containing the variables.
4. Browse to the root directory of the project.
5. Select variables_377sXL_DRV.txt.
6. Click Ok to import the variables, as shown in Figure 21.

NOTE: The structure variable 'motor1' has references to all peripherals and variables that are related to controlling the motor. By expanding this variable, the user can see them all and edit as needed.



Expression	Type	Value	Address
EnableFlag	unsigned int	1	0x0000B004@Data
IsrTicker	unsigned long	149718	0x0000B014@Data
OffsetCalCounter	int	20000	0x0000B009@Data
motor1	struct <unnamed>	{...}	0x0000B080@Data
PwmARegs	struct EPWM_REGS *	0x00004600	0x0000B080@Data
PwmBRegs	struct EPWM_REGS *	0x00004700	0x0000B082@Data
PwmCRegs	struct EPWM_REGS *	0x00004800	0x0000B084@Data
QepRegs	struct EQEP_REGS *	0x00005180	0x0000B086@Data
SpiRegs	struct SPI_REGS *	0x00006100	0x0000B088@Data
drvScsPin	unsigned int	4	0x0000B08A@Data
offset_shntA	float	0.0	0x0000B08C@Data
offset_shntB	float	0.0	0x0000B08E@Data
offset_shntC	float	0.0	0x0000B090@Data
VdTesting	float	0.0	0x0000B092@Data
VqTesting	float	0.07	0x0000B094@Data
IdRef	float	0.0	0x0000B096@Data
IqRef	float	0.1	0x0000B098@Data
SpeedRef	float	0.05	0x0000B09A@Data
ElecTheta	float	0.0	0x0000B09C@Data
MechTheta	float	0.0	0x0000B09E@Data

Figure 21. Variables Within Structure Motor1

7. Click the *Continuous Refresh* button (button) in the expressions window. This button enables to window to run with real-time mode.
8. Select Customize Continuous Refresh Interval by clicking the down arrow in the expressions window.
9. Edit the refresh rate of the expressions window.

NOTE: Choosing too fast of an interval may affect performance.

10. The datalog buffers point out different system variables depending on the build level. The buffers provide a means to visually inspect the variables and judge system performance.
11. Open and setup time graph windows to plot the data log buffers as shown in Figure 22.
12. *Click Tools→Graph→DualTime* to import graph configuration files in the folder.
13. Select *Import*.
14. Browse to *C:\TI\controlSUITE\development_kits\TIDM-SERVO-LAUNCHXS\MonoMtrServo_377s*.
15. Select *Graph1.graphProp*. The graph properties window will appear as shown in Figure 11, by following the steps below.
16. Click Ok. The graphs are now added to the debug perspective.

Copyright © 2015, Texas Instruments Incorporated

17. Click the *Continuous Refresh* button (button) on the top left corner of the graph tab.

NOTE: If a second graph window is used, import the *Graph2.prop* and the start address is *DBUFF_4CH3 and DBUFF_4CH4*.

NOTE: The default *dlog.prescaler* is set to 5, which allows the dlog function to only log one out of every five samples.

NOTE: The default *dlog.trig_value*is set to the correct value to generate trigger for the plot as in oscilloscopes.

Figure 22 shows the graph window settings.

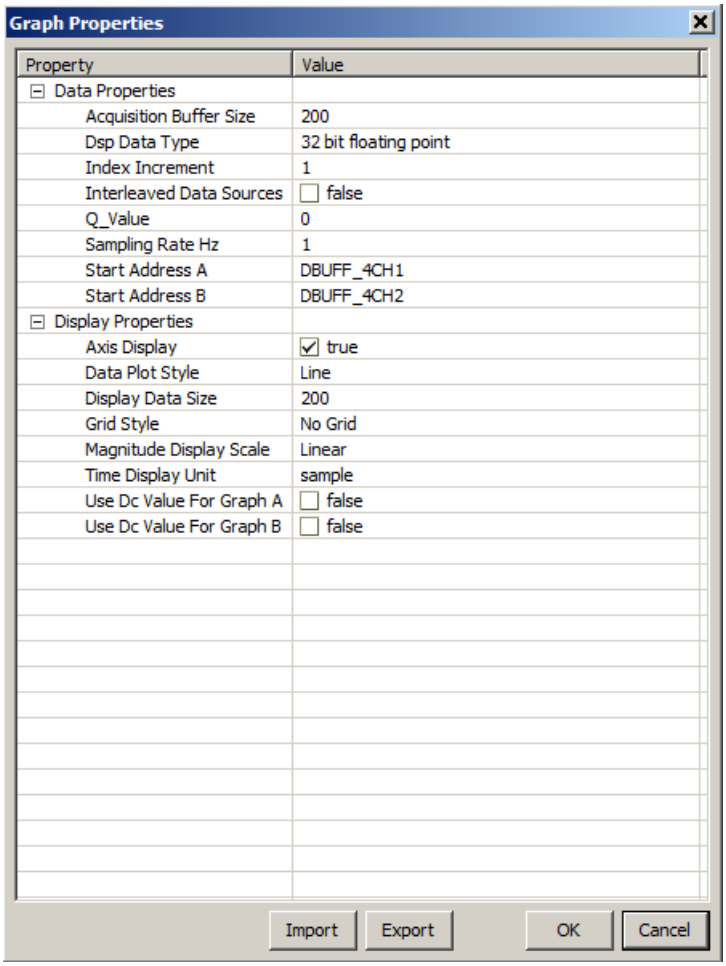


Figure 22. Graph Window Settings

Copyright © 2015, Texas Instruments Incorporated

Run the Code

1. Run the code by pressing the Run button (button) in the debug tab.
2. In the expressions window, set the variable to *EnableFlag* to 1.
3. The project runs and the values in the graphs and expressions window continuously update.
4. Figure 23 and Figure 24 show screen captures of typical CCS perspectives while using the project.
5. Resize the window if required.
6. Once complete, turn off real-time mode.
7. Click (black button) to reset the processor, or go to *Run→Reset→CPU Reset*.
8. Click (red button) to terminate the debug session, or go to *Run→Terminate*.
9. The program halts and disconnects the code composer from the MCU.

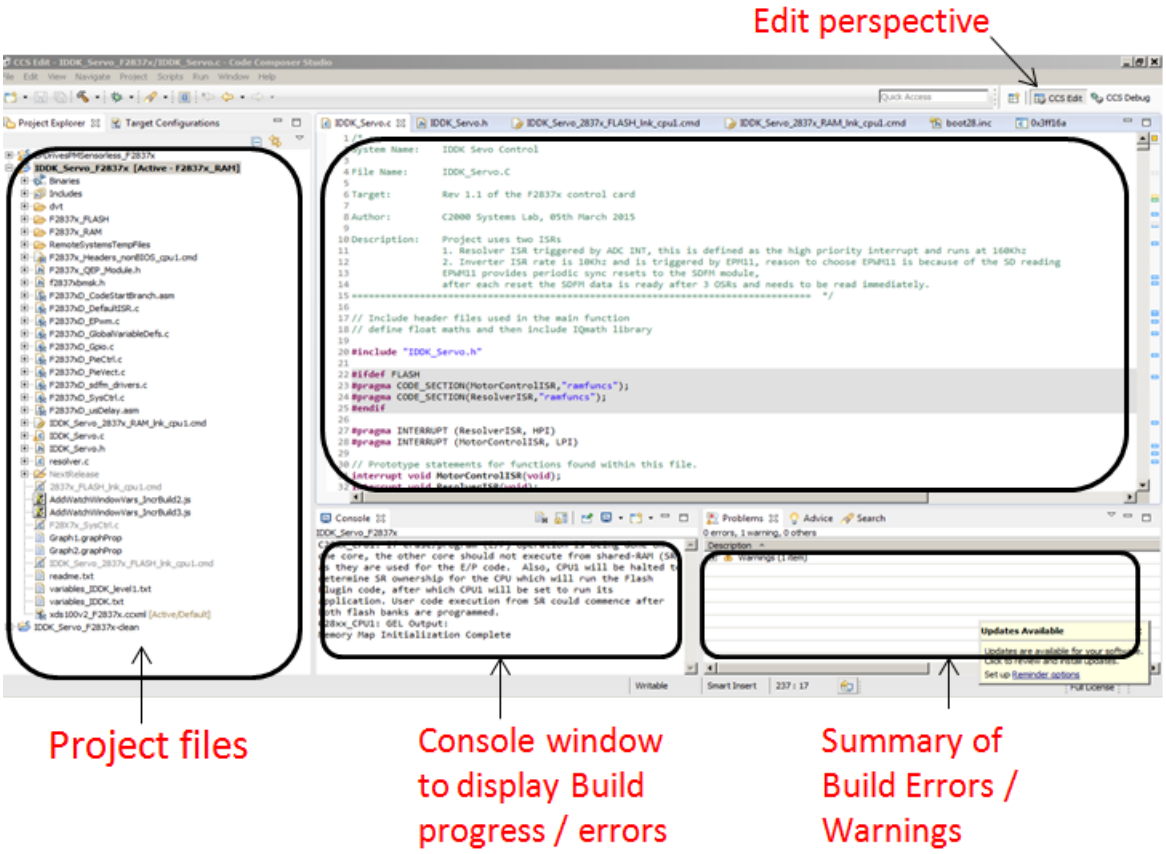


Figure 23. CCS IDE Showing Edit Perspective

Copyright © 2015, Texas Instruments Incorporated

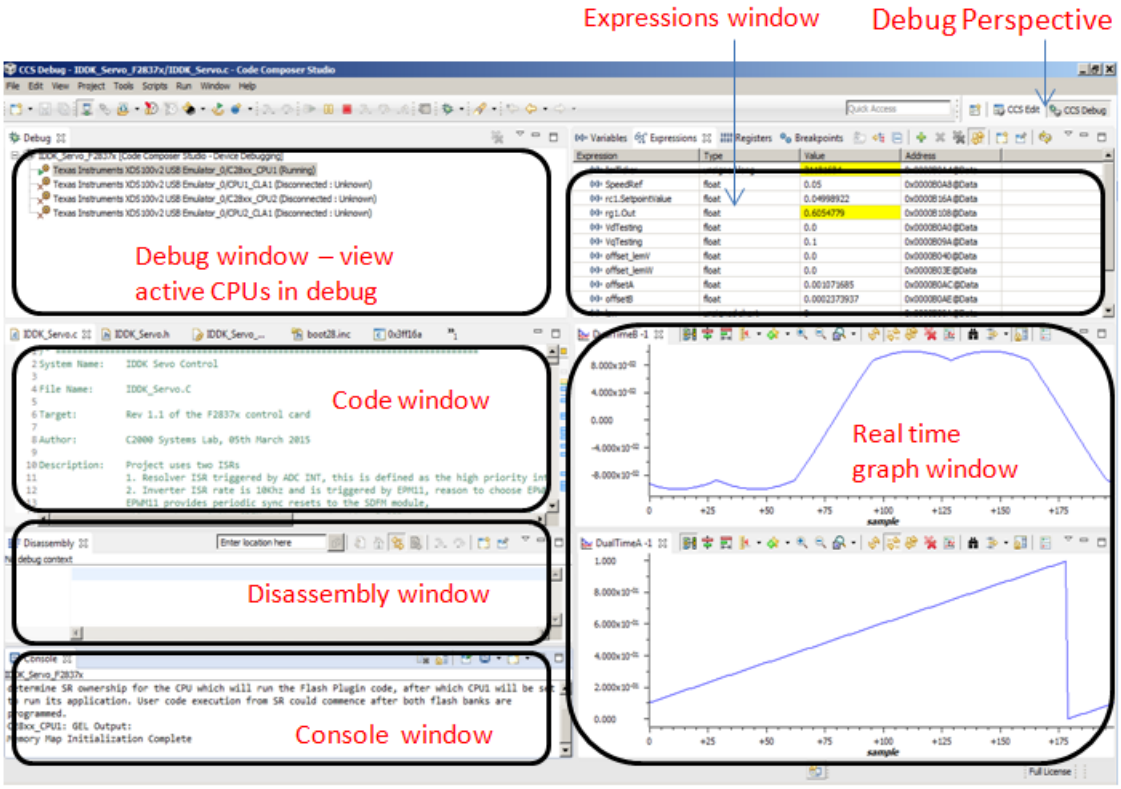


Figure 24. CCS IDE Showing Debug Perspective

The Next Steps

1. Do not terminate the debug session each time after the code runs or changes again.
2. Click (black button) to rebuild the project.
3. Go to *Run→Reset→CPU Reset*.
4. Go to *Run→Reset*.
5. Enable real-time options.
6. Once complete, disable the real time options.
7. Reset the CPU.
8. Terminate the project if the target device or the configuration is changed (RAM to FLASH or FLASH to RAM), and prior to shutting down CCS.

NOTE: The header file *MonoMtrServo-settings.h* in the CCS *Edit Perspective* has macro definitions for *DRV_Gain* and *PWM_FREQUENCY*. Modify these parameters as needed after thinking through the impact of the change.

9. Proceed to experiment various build levels in detail.

Copyright © 2015, Texas Instruments Incorporated

10 Incremental System Build

The system is gradually built up in order for the final system can be confidently operated. Five phases of the incremental system build are designed to verify the major software modules used in the system. Most modules are written as software MACROs, and the remaining are written as callable functions. Table 6 lists the modules testing and using in each incremental system build.

Table 6. Testing Modules 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 in this phase. The symbol √√ means this module is testing in this phase.					

10.1 Level 1: Incremental Build

Figure 25 shows the incremental system build block diagram.

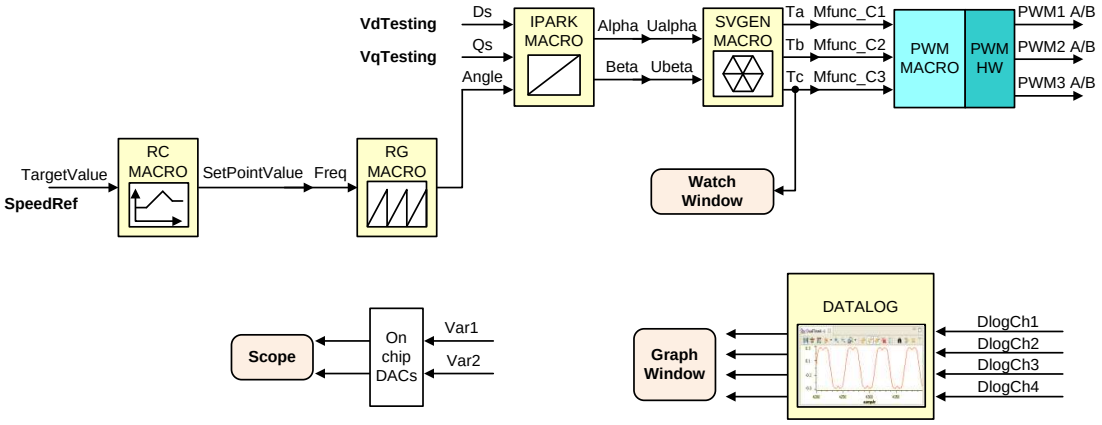


Figure 25. Level 1–Incremental System Build Block Diagram

At this step, keep the motor disconnected. This section describes the steps for a minimum system check-out which confirms operation of system interrupt, the peripheral and target independent *I_PARK_MACRO* (inverse park transformation) and *SVGEN_MACRO* (space vector generator) modules. Also included is the confirmation of the peripheral dependent *PWM_MACRO* (PWM initializations and update) modules.

1. Open *MonoMtrServo-Settings.h* .
2. Select the *level 1 incremental build option* by setting the *BUILDLEVEL* to *LEVEL1* (*#define BUILDLEVEL LEVEL1*).
3. Right click on the project name.
4. Click *Rebuild Project*.

Copyright © 2015, Texas Instruments Incorporated

5. Once the build is complete, click on the *debug button*.
6. Reset the CPU.
7. Restart the CPU.
8. Enable real-time mode and run.
9. Right click within the expressions window and import the file *variables_377sXL_DRV.txt* if not added into the expressions window already.
10. Set *EnableFlag* to 1 in the expressions window.
11. The variable named *IsrTicker* incrementally increases as seen in the expressions window confirming that the interrupt is working properly.

In the software, the key variables within the structure of *motor1* where adjustment is required are:

- **SpeedRef:** Changes the rotor speed per-unit.
- **VdTesting:** Changes the d-axis voltage per-unit.
- **VqTesting:** Changes the q-axis per-unit.

Level 1A (SVGEN_MACRO Test)

The SpeedRef value is specified to the *RG_MACRO* module through the *RC_MACRO* module. The *IPARK_MACRO* module generates the outputs to the *SVGEN_MACRO* module. Three outputs from the *SVGEN_MACRO* module are monitored through the graph window as shown in Figure 15 where the Ta, Tb, and Tc waveforms are 120º apart from each other. Specifically, Tb lags Ta by 120º ,and Tc leads Ta by 120º. Check the PWM test points on the board to observe PWM pulses (PWM-1 H to 3 H and PWM-1L to 3 L). Make sure that the PWM module is running properly. Figure 26 shows the output of SVGEN, Ta, Tb, Tc, and Tb-Tc waveforms.

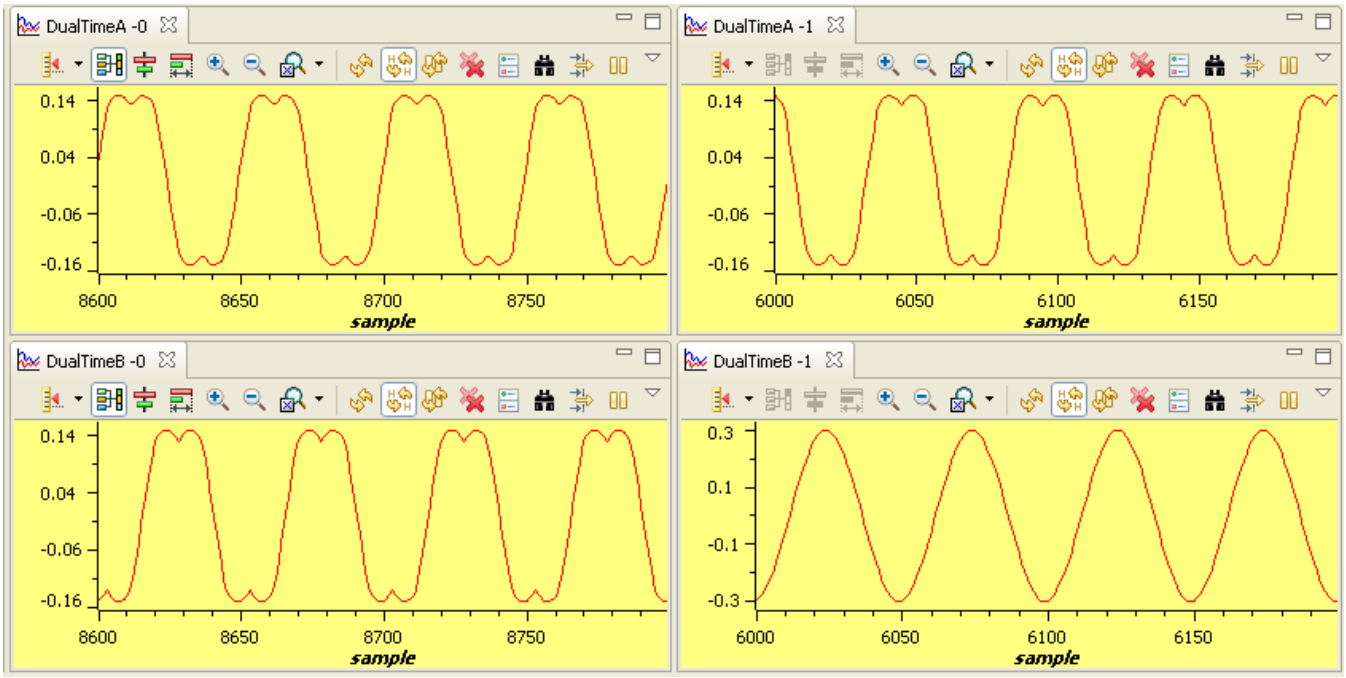


Figure 26. Output of SVGEN, Ta, Tb, Tc and Tb-Tc Waveforms

Copyright © 2015, Texas Instruments Incorporated

Level 1B (*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. Check VA-FB / VSENA, VB-FB / VSENB and VC-FB / VSENC test points using an oscilloscope. If you observe waveforms similar to what is shown in Level 1A, it ensures that the inverter is working properly.

CAUTION:

After verifying this build level, take the controller out of real-time mode (disable), and reset the processor.

NOTE: After each test, this step must be repeated for safety purposes, otherwise, an improper shutdown may halt the PWMs at certain states where the inverter may draw high currents. Thus, caution must be taken while performing these experiments.

10.2 Level 2: Incremental Build

Figure 27 shows the incremental system build block diagram.

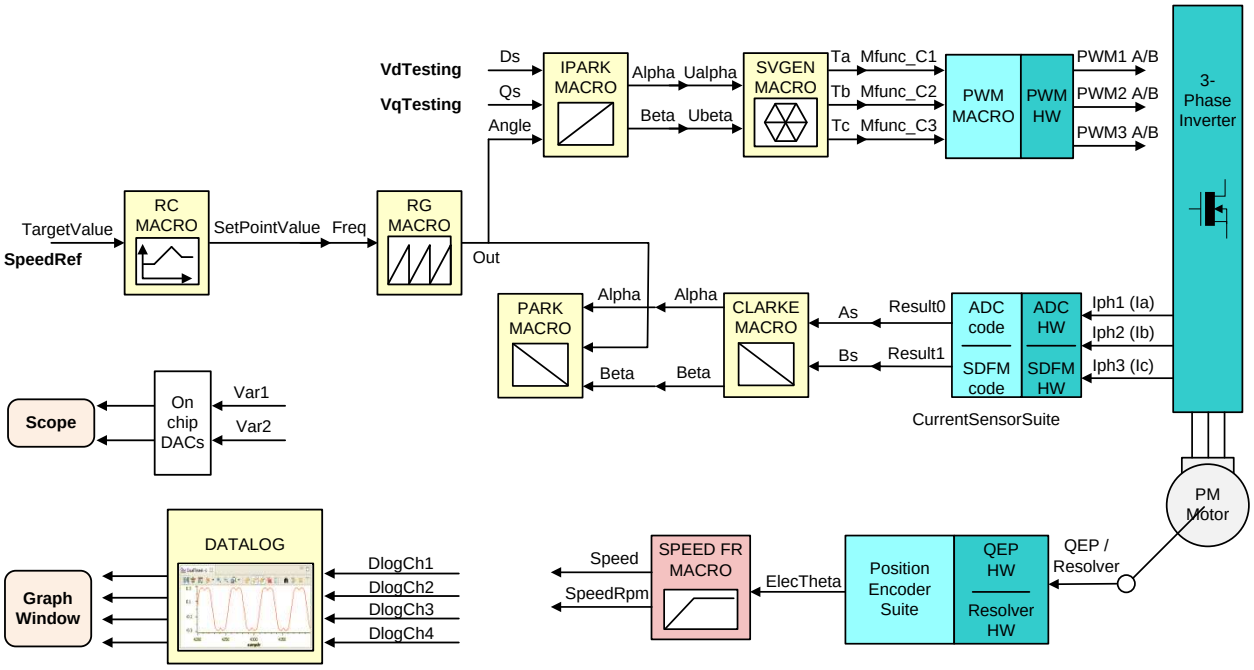


Figure 27. Level 2–Incremental System Build Block Diagram

In this section, more blocks are added to level 1 and tested. Assuming BUILD 1 is completed successfully, this section verifies the analog-to-digital conversion, and the Clarke/Park transformations. In this build, the motor is run in open loop to verify current sensing (SHUNT) and position sensing (QEP) used in the set up.

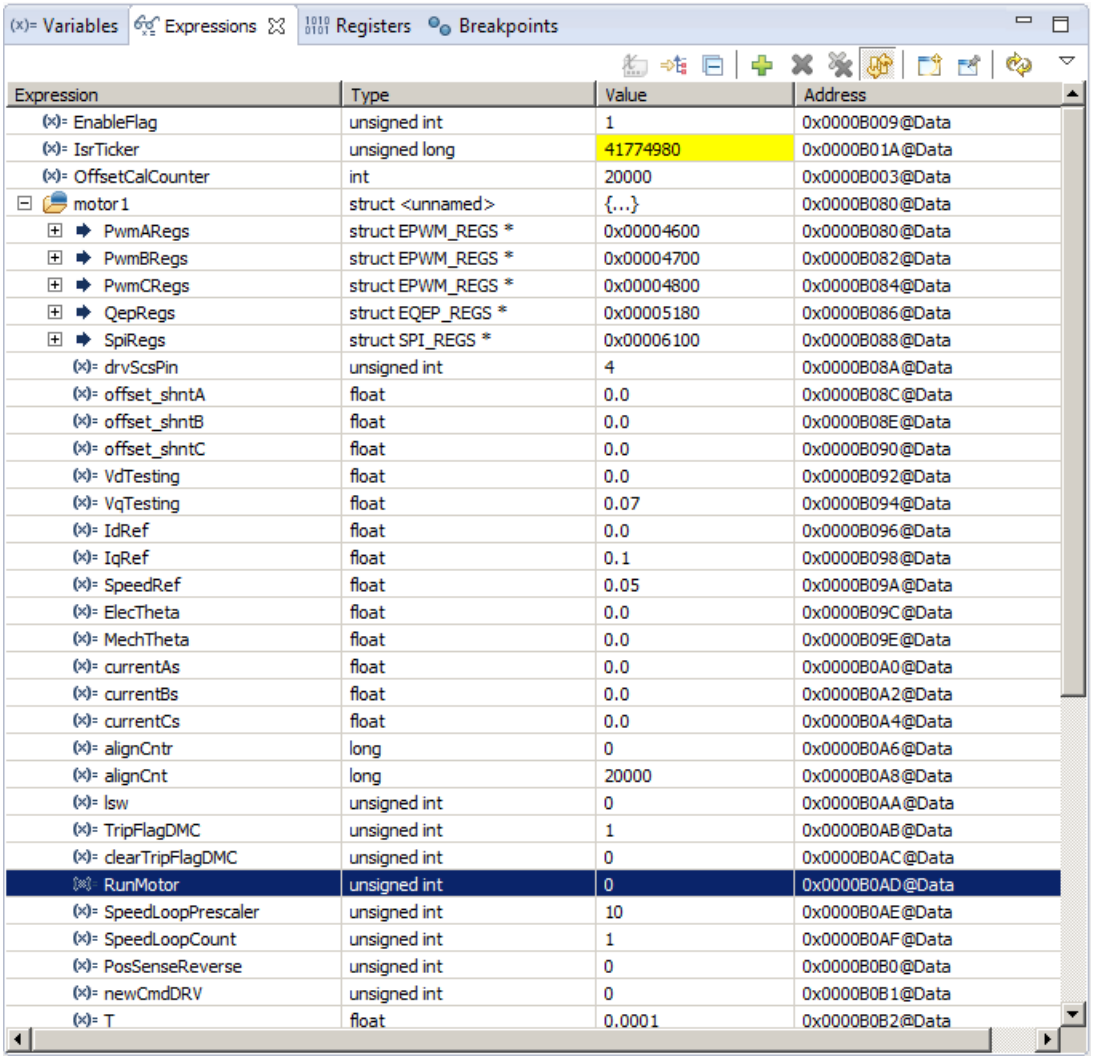
1. Turn off the power to BOOSTXL-DRV830x kit.
2. Connect the motor to the DRV kit since the PWM signals are successfully proven through the level 1 incremental build.
3. Connect the QEP connector from the motor to the QEP-B header on the LaunchPad.
4. If the Teknic motor M-2310S-LN-04K is being used, then the QEP connector is J4. The arrow mark on the connector label aligns with pin 1 of the QEP-B header while mounting.

NOTE: 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.

5. Turn on the power to the DRV kit.

Copyright © 2015, Texas Instruments Incorporated

6. Open *MonoMtrServo-Settings.h*.
7. Set the *BUILDLEVEL* to *LEVEL 2* (*#define BUILDLEVEL LEVEL 2*) to select the level 2 incremental build option.
8. Right click on the project name.
9. Click *Rebuild Project*.
10. Click the *debug* button once the build is complete.
11. Reset the CPU.
12. Restart the CPU.
13. Enable real-time mode and run. Figure 28 shows the expressions window for build level 2.



Expression	Type	Value	Address
EnableFlag	unsigned int	1	0x0000B009@Data
IsrTicker	unsigned long	41774980	0x0000B01A@Data
OffsetCalCounter	int	20000	0x0000B003@Data
motor1	struct <unnamed>	{...}	0x0000B080@Data
PwmARegs	struct EPWM_REGS *	0x00004600	0x0000B080@Data
PwmBRegs	struct EPWM_REGS *	0x00004700	0x0000B082@Data
PwmCRegs	struct EPWM_REGS *	0x00004800	0x0000B084@Data
QepRegs	struct EQEP_REGS *	0x00005180	0x0000B086@Data
SpiRegs	struct SPI_REGS *	0x00006100	0x0000B088@Data
drvScsPin	unsigned int	4	0x0000B08A@Data
offset_shntA	float	0.0	0x0000B08C@Data
offset_shntB	float	0.0	0x0000B08E@Data
offset_shntC	float	0.0	0x0000B090@Data
VdTesting	float	0.0	0x0000B092@Data
VqTesting	float	0.07	0x0000B094@Data
IdRef	float	0.0	0x0000B096@Data
IqRef	float	0.1	0x0000B098@Data
SpeedRef	float	0.05	0x0000B09A@Data
ElecTheta	float	0.0	0x0000B09C@Data
MechTheta	float	0.0	0x0000B09E@Data
currentAs	float	0.0	0x0000B0A0@Data
currentBs	float	0.0	0x0000B0A2@Data
currentCs	float	0.0	0x0000B0A4@Data
alignCntr	long	0	0x0000B0A6@Data
alignCnt	long	20000	0x0000B0A8@Data
lsw	unsigned int	0	0x0000B0AA@Data
TripFlagDMC	unsigned int	1	0x0000B0AB@Data
clearTripFlagDMC	unsigned int	0	0x0000B0AC@Data
RunMotor	unsigned int	0	0x0000B0AD@Data
SpeedLoopPrescaler	unsigned int	10	0x0000B0AE@Data
SpeedLoopCount	unsigned int	1	0x0000B0AF@Data
PosSenseReverse	unsigned int	0	0x0000B0B0@Data
newCmdDRV	unsigned int	0	0x0000B0B1@Data
T	float	0.0001	0x0000B0B2@Data

Figure 28. Expressions Window for Build Level 2

14. Set the *EnableFlag* to 1 in the expressions window.
15. The variable *IsrTicker* incrementally increases as seen in the expression window, which confirms that the interrupt is working properly.

In the software, the key variables within the structure of *motor1* where adjustment is required are:

- RunMotor: Set to 1/0 to start and stop the motor.
- SpeedRef: Changes the rotor speed per-unit.
- VdTesting: Changes the d-axis per unit.

Copyright © 2015, Texas Instruments Incorporated

- VqTesting: Changes the q-axis per unit.

Set the *RunMotor* to 1 and the motor spins after a few seconds. During open loop tests, VqTesting and SpeedRef are adjusted carefully for PM motors so that the motor does not stall or vibrate.

Phase 2A–Testing the Clarke Module

The three measured line currents are transformed to two phase alpha, beta currents in a stationary reference frame. The outputs of this module are checked from graph window. Figure 29 shows the waveforms of Svgen_dq1.Ta,rg1.Out, and phase A and B currents.

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

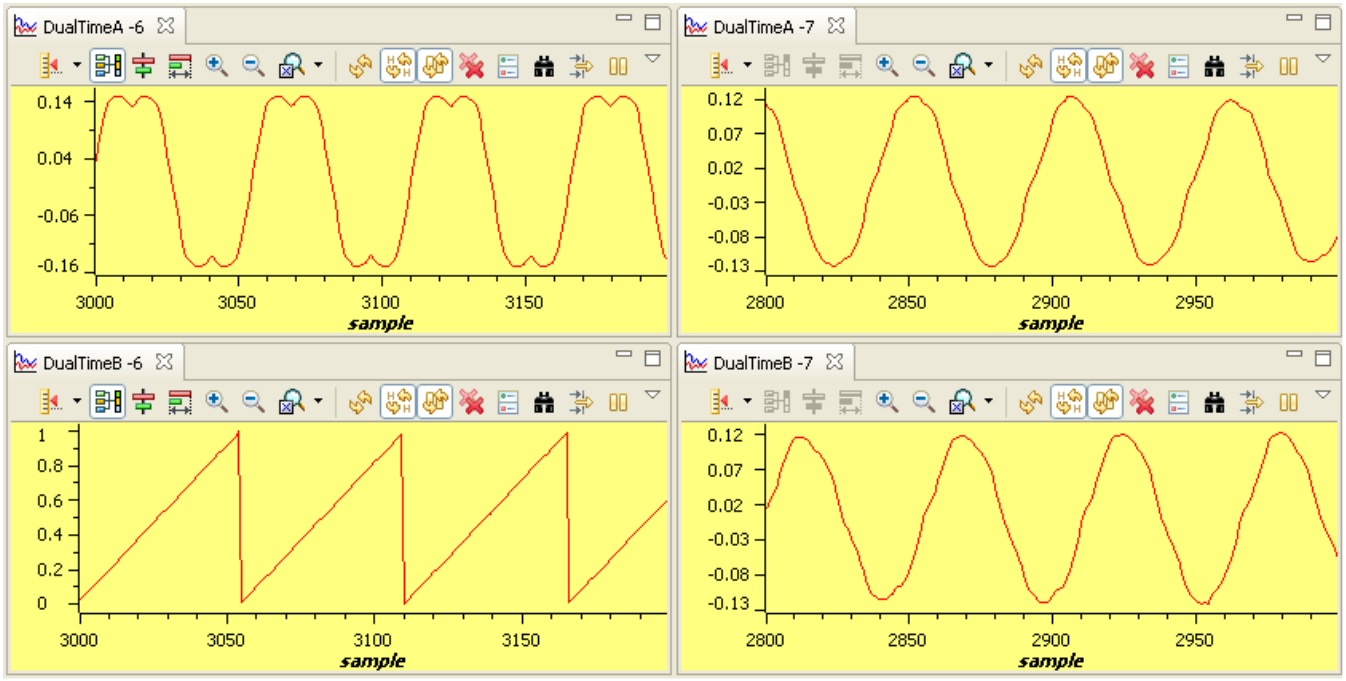


Figure 29. Waveforms of Svgen_dq1.Ta,rg1.Out, and Phase A and B Currents

Copyright © 2015, Texas Instruments Incorporated

Because the low side current measurement technique is used employing shunt resistors on inverter phase legs, the phase current waveforms sampled by ADC are composed of pulses as shown in Figure 20. Figure 30 shows the amplified phase A current.

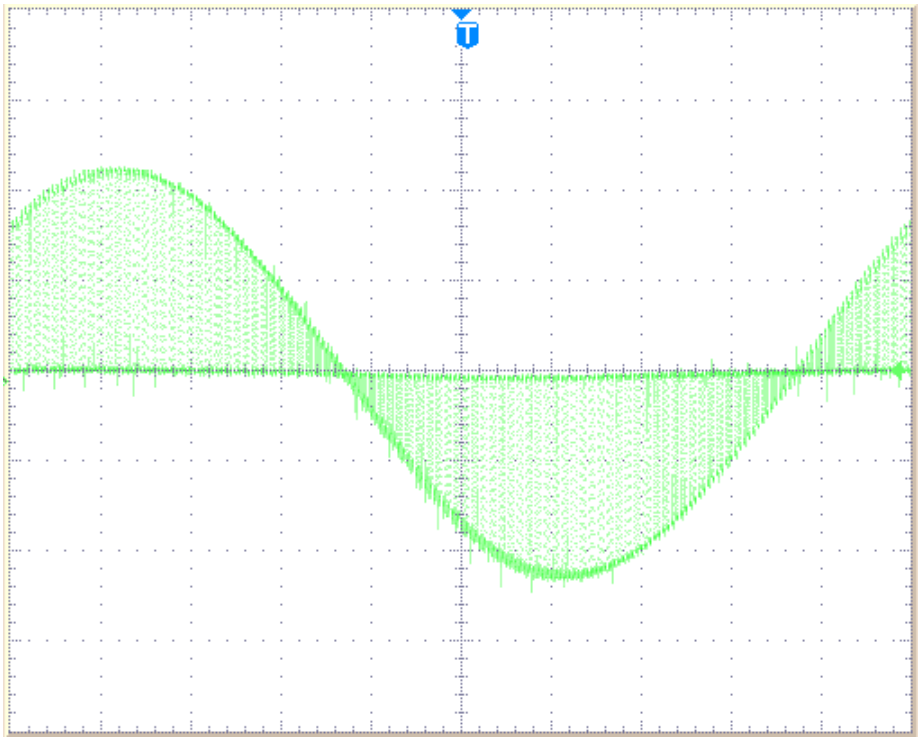


Figure 30. Amplified Phase A Current

10.3 Level 2B–Adjusting PI Limits

The vectorial sum of d-q PI outputs are 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 and software implementation. Depending on the application requirements 3, 2, or a single shunt resistor may be used for current waveform reconstruction. The higher number of shunt resistors allow higher duty cycle operation and better dc bus use.

1. Run the system with default VdTesting, VqTesting, and SpeedRef.
2. Gradually increase VdTesting and VqTesting values.
3. Watch the current waveforms in the graph window.
4. Keep increasing the VdTesting and VqTesting values until you notice distorted current waveforms.
5. Write down the maximum allowed VdTesting and VqTesting values.
6. Make sure that the VdTesting and VqTesting values are consistent with expected d-q current component maximums while running the motor.
7. After this build level, PI outputs automatically generate the voltage reference and determine the PWM duty cycle depending on the d-q current demand.
8. Set *pi_id.Umax/min* and *pi_iq.Umax/min* according to recorded VdTesting and VqTesting values respectively.

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

Bring the system to a safe stop as described in Section 10.1 by taking the controller out of real time mode and reset.

Copyright © 2015, Texas Instruments Incorporated

10.4 Level 2C–Position Encoder Feedback and SPEED_FR Test

During the above tests, the position encoder interface is continuously estimating position information, 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 before the rotor spins continuously. Reference position (rg1.Out) and encoder feedback angle (speed.ElectTheta) when plotted using the graph tool should look like Figure 31. .

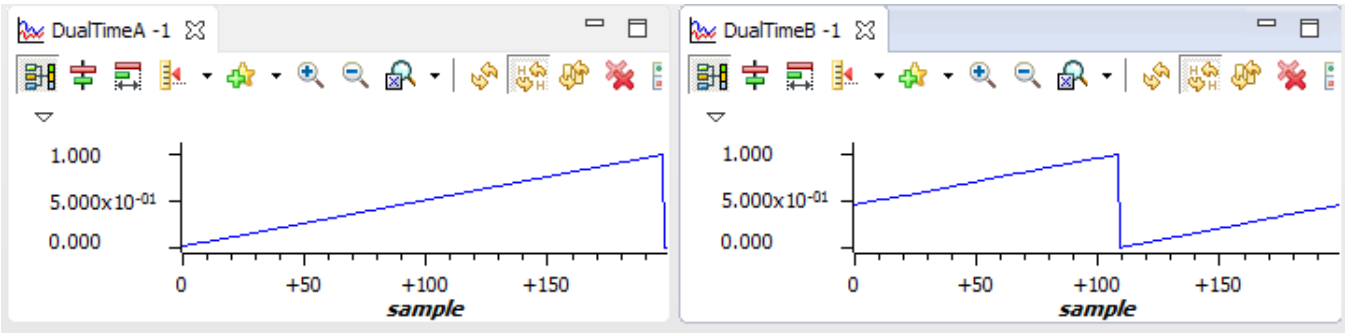


Figure 31. Scope Plot of the Reference Angle and Rotor Position

The waveform of channel A represents the reference position, while channel B represents the feedback position. The ripple in feedback 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, make sure that the sense of change of the estimated angle is 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 setting the flag *motor1.PosSenseReverse* to 1.

With a QEP:

Watch out for Qep1.CalibratedAngle in the expressions window. The Qep1. CalibratedAngle represents the electrical angle (or) position of rotor at the event of Index pulse. The Qep1. CalinratedAngle may vary depending on starting position of rotor. When a closed loop operation is performed, determine the angular offset between electrical zero and QEP's index pulse that resets the QEP counter to zero. The calibration angle is formulated as:

Calibration Angle= Offset Angle ±n. Line Encoder

Copyright © 2015, Texas Instruments Incorporated

11 Level 3–Incremental Build

Figure 32 shows the level 3 incremental system build block diagram.

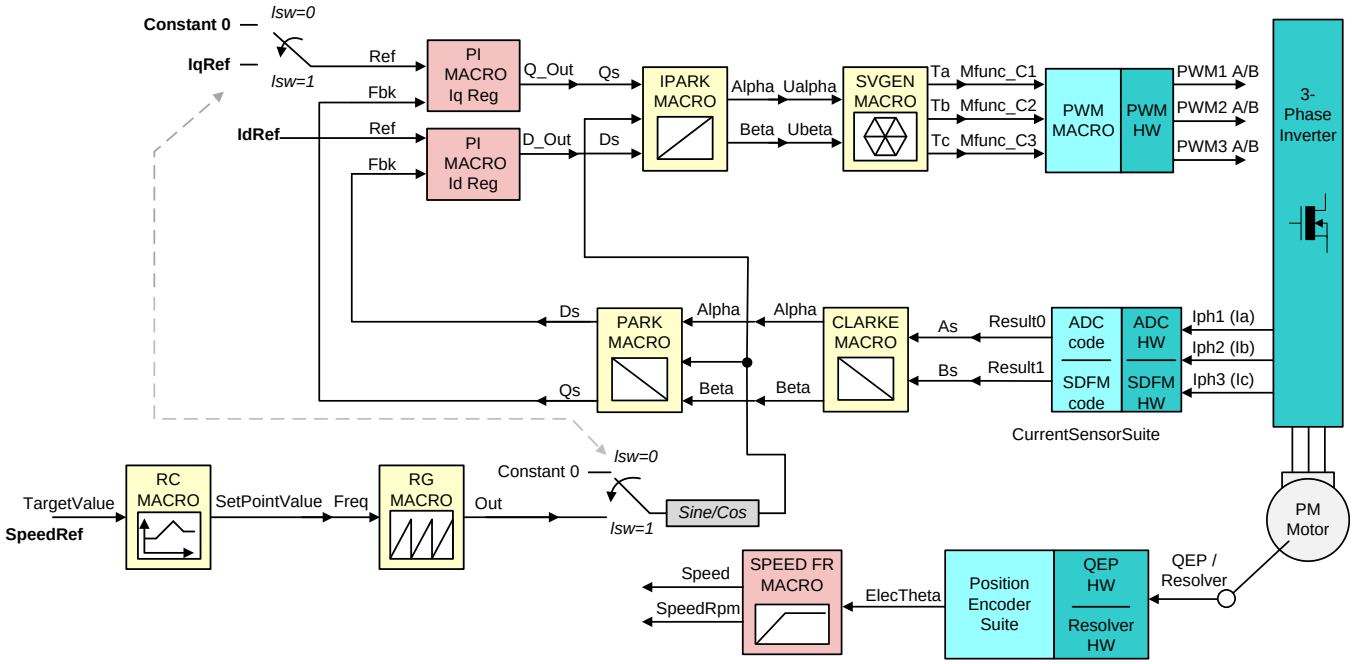


Figure 32. Level 3–Incremental System Build Block Diagram

Assuming the previous section is completed successfully, this section verifies the dq-axis current regulation performed by the PI modules. To confirm the operation of current regulation, the gains of the two PI controllers are tuned for proper operation. In this build, transformations are completed based on the reference angle generated manually, rather than the actual rotor position. This is to ensure that the test is completed without a large load on the motor. Otherwise, the Iq loop testing cannot be completed easily. When the motor is commanded to run, an initial alignment stage occurs where the electrical angle and the QEP angle count are set to zero.

1. Open *MonoMtrServo-Settings.h*.
2. Set the *BUILDLEBEL* to *LEVEL 3* (#define BUILDLEVEL LEVEL3) to select the level 3 incremental build option.
3. Right-click on the project name.
4. Click *Rebuild Project*.
5. Once the build is complete, click on the *debug button*.
6. Reset the CPU.
7. Restart the CPU.
8. Enable real time mode and run.
9. Set the *EnableFlag* to 1 in the expressions window.
10. The *IsrTicker* variable increases incrementally as seen in the expression window, which confirms that the interrupt is working properly.

In the software, the key variables within the structure of *motor1* to be adjusted are:

- **RunMotor:** Set to 1/0 to start and stop the motor.
- **SpeedRef:** Changes the rotor speed per-unit.
- **IdRef**: Changes the d-axis per-unit.
- **IqRef**: Changes the q-axis voltage per-unit.

In this build, the motor current is dynamically regulated by the PI modules.

Copyright © 2015, Texas Instruments Incorporated

1. Set the *SpeedRef* to 0.3 pu (or another suitable value if the base speed is different).
2. Set *Idref* to zero.
3. Set *Iqref* to 0.05 pu (or another suitable value).
4. Expand *motor1* to view variables *pi_id.Fbk*, *pi_id.Kp* and *pi_id.Ki* and corresponding elements for *pi_iq*.
5. Set the *RunMotor* flag to 1.
6. Check *pi_id.fbk* in the expressions window with the *continuous refresh* feature.
7. Verify if *pi_id.Ref* (IdRef) is being tracked.
8. Adjust the PI gains if *pi_id.Ref* (IdRef) is **not** being tracked.
9. To confirm these two PI modules, try different values of *pi_id.Ref*, *pi_iq.Ref*, or *SpeedRef*.

NOTE: 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 by taking the controller out of real time mode and reset.
11. The motor stops running.

Figure 33 shows how the current the waveforms appear in the CCS graphs will running this build.

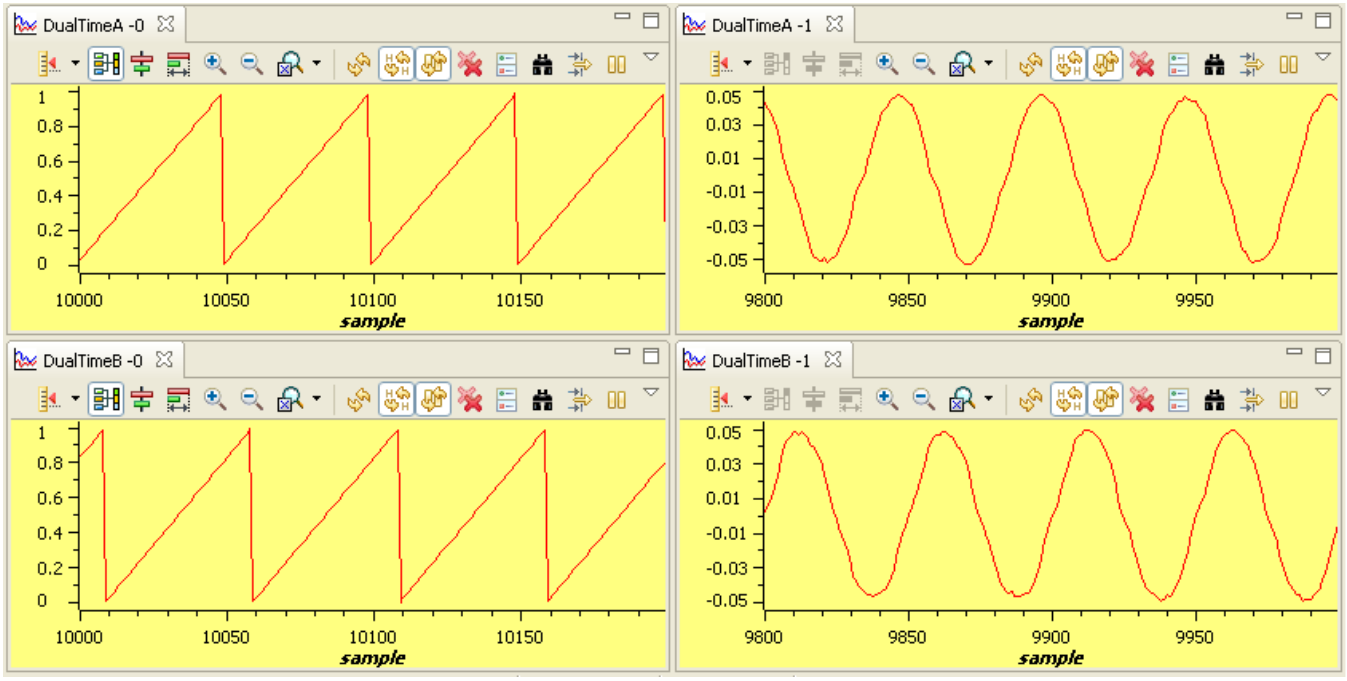


Figure 33. Measured Theta, rg1.out and Phase A & B current waveforms

Copyright © 2015, Texas Instruments Incorporated

12 Level 4–Incremental Build

Figure 34 shows the level 4 incremental build system block diagram.

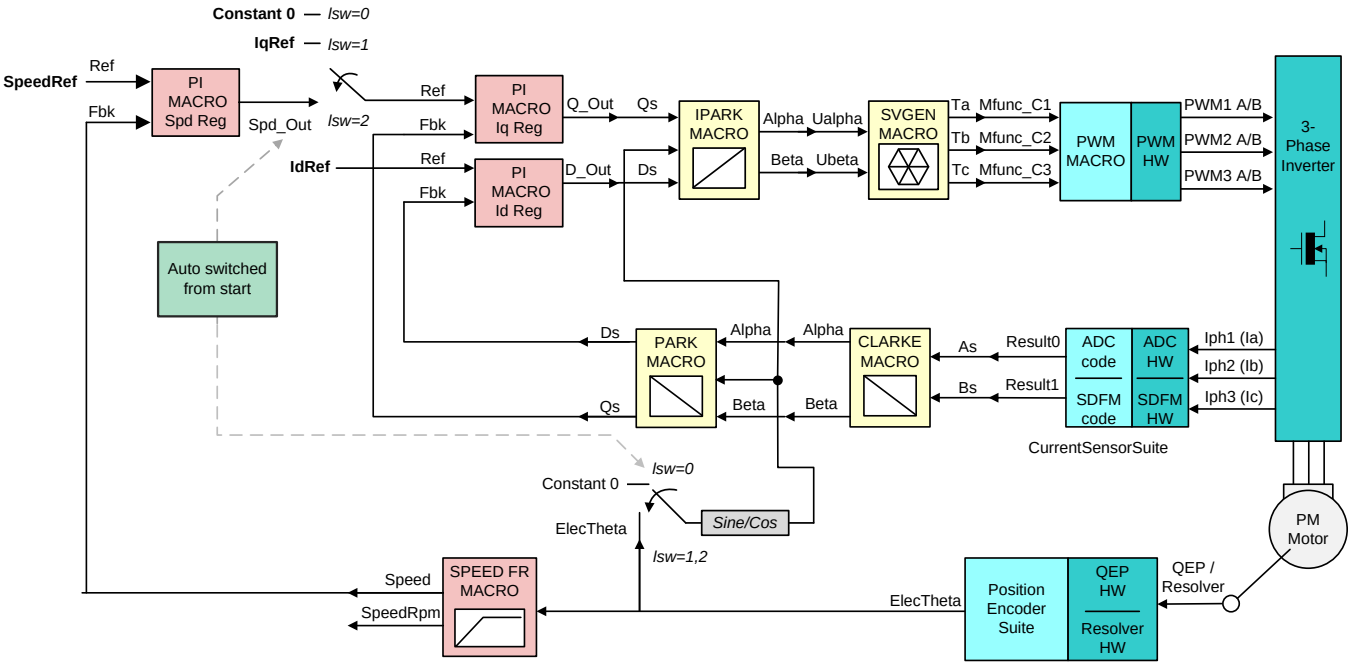


Figure 34. Level 4–Incremental System Build Block Diagram

Assuming the previous section is completed successfully; this section verifies the speed PI module and speed loop. All transformations are completed based on the actual rotor position. When the motor is commanded to run, an initial alignment stage occurs where the electrical angle and the QEP angle count are set to zero.

1. Open *MonoMtrServo-Settings.h*.
2. Set the BUILDLEVEL to LEVEL 4 (#define BUILDLEVEL LEVEL 4) to select the level 4 incremental build option.
3. Right-click on the project name.
4. Click *Rebuild Project*.
5. Once the build is complete, click on the *debug* button.
6. Reset the CPU.
7. Restart the CPU.
8. Enable real-time mode and run.
9. Set the *EnableFlag* to 1 in the expressions window.
10. The *IsrTicker* variable incrementally increases as shown in the expressions window, which confirms the interrupt working properly.

In the software, the key variables of *motor 1* to be adjusted are:

- **RunMotor:** Set to 1/0 to stat and stop motor.
- **SpeedRef:** Changes the rotor speed per-unit.

Key Steps:

1. Set *SpeedRef* to 0.3 pu (or another suitable value if the base speed is different)/
2. Set *RunMotor* to 1 to start the motor.
3. The soft-switch variable (lsw) is auto-promoted in a sequence depending on receipt of the QEP index pulse.

- lsw= 0, Lock the rotor of the motor.

Copyright © 2015, Texas Instruments Incorporated

- lsw= 1, The motor is in run mode and is waiting for the first instance of QEP Index Pulse.
- lsw= 2, The motor is is run mode, QEP–the first index pulse has occurred.

1. The motor starts running.
2. Compare speed with *SpeedRef* in the expressions window with the continuous refresh feature.
3. Verify that they are approximately the same.
4. Try different values of *SpeedRef* (positive or negative) to confirm the speed PID module.
5. For the speed PID controller, the proportional, integral, derivative, and integral correction gains may be re-tuned to achieve the desired response.
6. Bring the system to a safe stop as described at the end of build 1 by taking the controller out of real-time mode and reset.
7. The motor stops running.

Copyright © 2015, Texas Instruments Incorporated

Figure 35 shows how the current waveforms appear in the CCS graphs while running this build. Figure 36 shows the measured theta, svgen duty cycle and phase A and B current waveforms under no-load and 0.3-pu speed. .

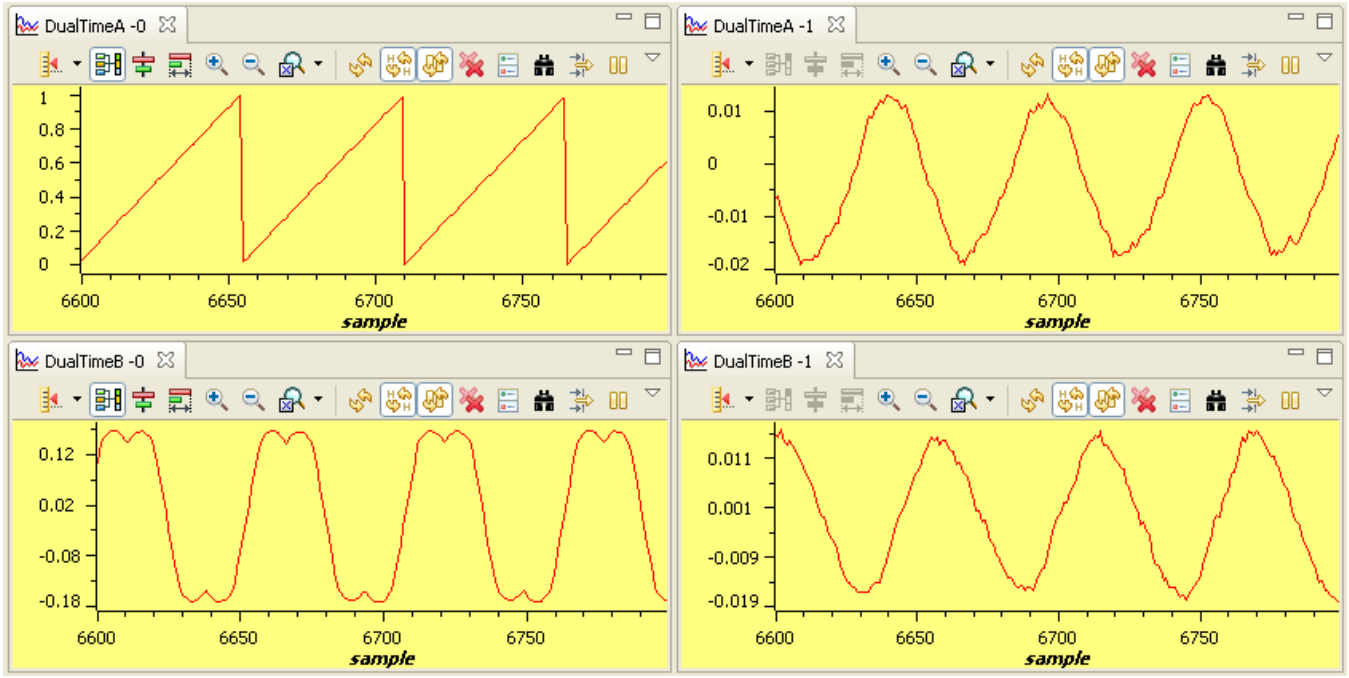


Figure 35. Measured Theta, Svgen Duty Cycle and Phase A and B Current Waveforms Under No-Load and 0.3-pu Speed

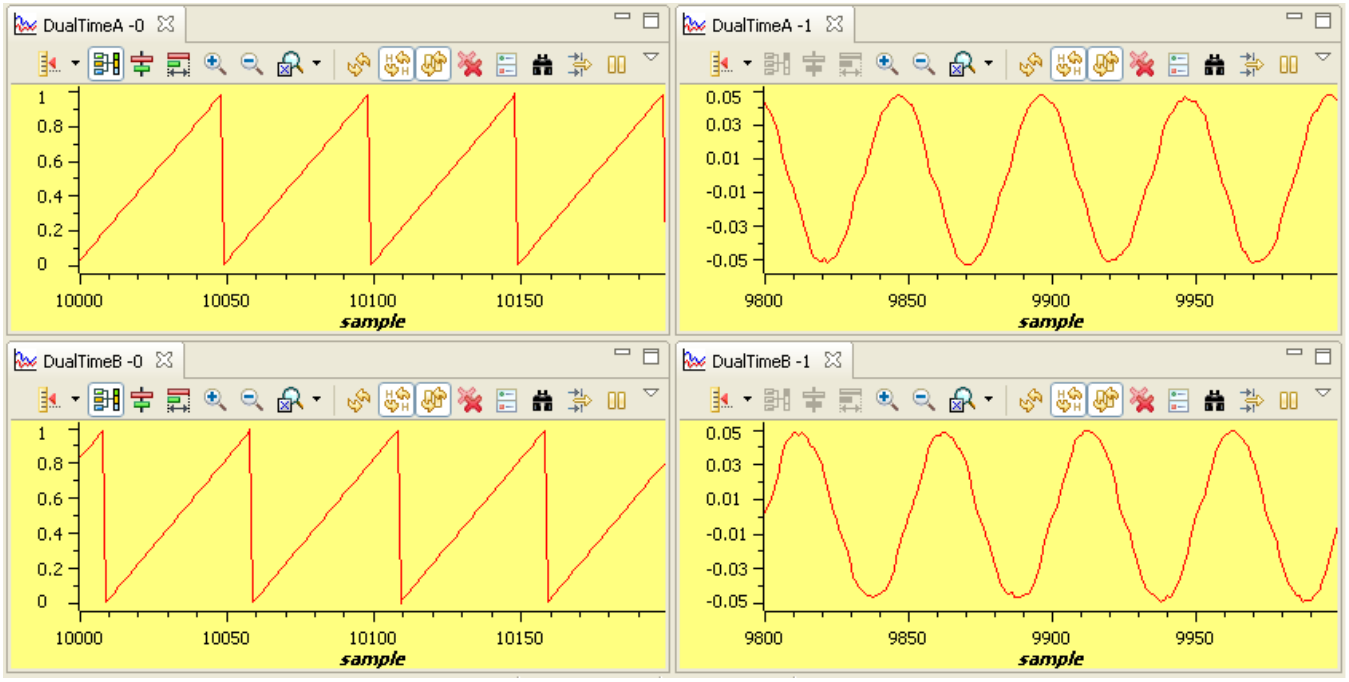


Figure 36. Measured Theta, Svgen Duty Cycle and Phase A and B Current Waveforms Under No-Load and 0.33-pu Speed

Copyright © 2015, Texas Instruments Incorporated

Figure 37 shows the flux and torque components of the stator current in the synchronous reference frame under 0.33-pu step-load and 0.3-pu step speed.

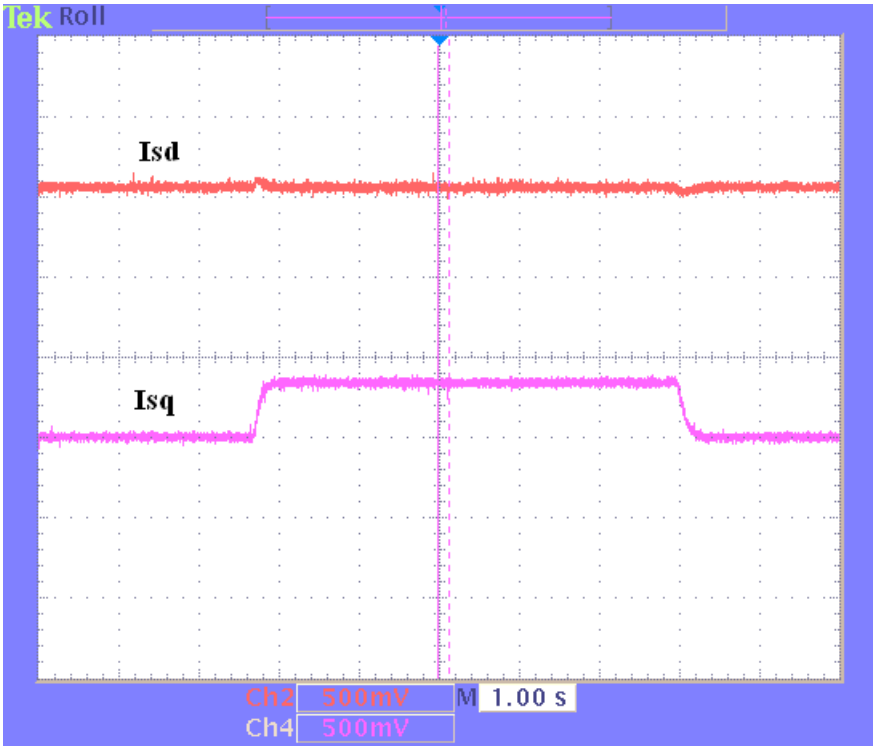


Figure 37. Flux and Torque Components of the Stator Current in the Synchronous Reference Frame Under 0.33-pu Step-Load and 0.3-pu Speed

Copyright © 2015, Texas Instruments Incorporated

13 Level 5–Incremental Build

Figure 38 shows the level 5 incremental system build block diagram.

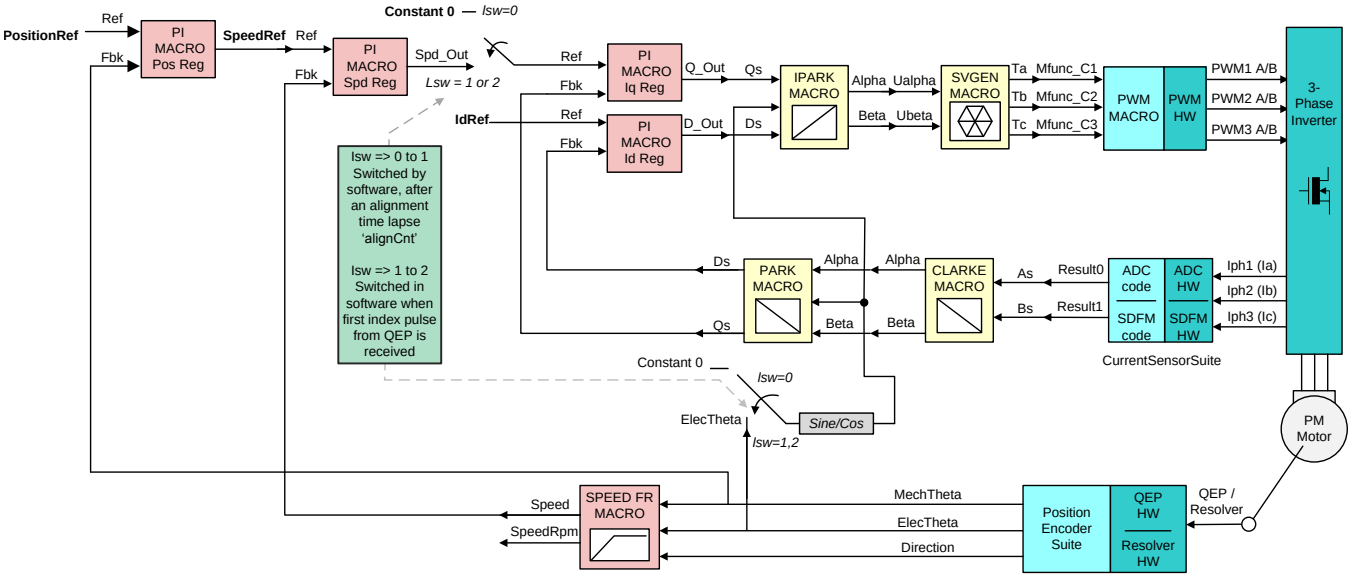


Figure 38. Level 5–Incremental System Build Block Diagram

This section verifies the position PI module and position loop with a QEP. For this loop to work properly, Section 12 must have been completed successfully. When the motor is commanded to run, an initial alignment stage occurs where the electrical angle and the QEP angle count are set to zero. After ensuring a stable alignment, the rotor is spun in FOC from start.

1. Open *MonoMtrServo-Settings.h*.
2. Set the BUILDLEVEL to LEVEL 5 (#define BUILDLEVEL LEVEL 5) to select the level 5 incremental build option.
3. Right-click on the project name
4. Click *Rebuild Project*.
5. Click the *debug* button once the build is complete.
6. Reset the CPU.
7. Restart the CPU.
8. Enable real-time mode and run.
9. Add the *posArray*, *ptrMax* and *posSlewRate* variables to the expressions window.
10. The *IsrTicker* variable incrementally increased as shown in the expressions window, confirming that the interrupt is working properly.
11. Set *RunMotor* to 1 in the expressions window. The motors run through predefined motion profiles and position settings as set by the *refPosGen* module.
12. The *refPosGen* module cycles the position reference through a set of values as defined in the *posArray*.
13. The values represent the number of the rotations and turns with respect to the initial alignment position.
14. Once a certain position value as defined in the array is reached, there is a pause before slewing toward the next one.
15. The array values may be referred to as **parking positions**.
16. During transition from one parking position to the next, the rate of transition (or speed) is set by the **posSlewRate**.
17. The number of positions in *posArray* pass through before restarting again from the first value decided by *ptrMax*.

Copyright © 2015, Texas Instruments Incorporated

18. Set *EnableFlag* to 1 in the expressions window.

For Experimenting:

1. The parking positions in the *posArray* may be changed to different values to see if the motor turns as many rotations as set.
2. The number of parking positions *prtMax* may also be changed to a set rotation pattern.
3. Position slew rate may be changed using *posSlewRate*. This represents the angle (in pu) per sampling instant.
4. The proportional and integral gains of the speed and position of the PI controllers may be re-tuned to achieve satisfactory responses.
5. TI recommends to tune the speed loop first, and then the position loop.
6. Bring the system to a safe stop by taking the controller out of real-time mode and reset.
7. The motor stops running.

Figure 39 shows the position reference and position feedback, observe that they are closely aligned.

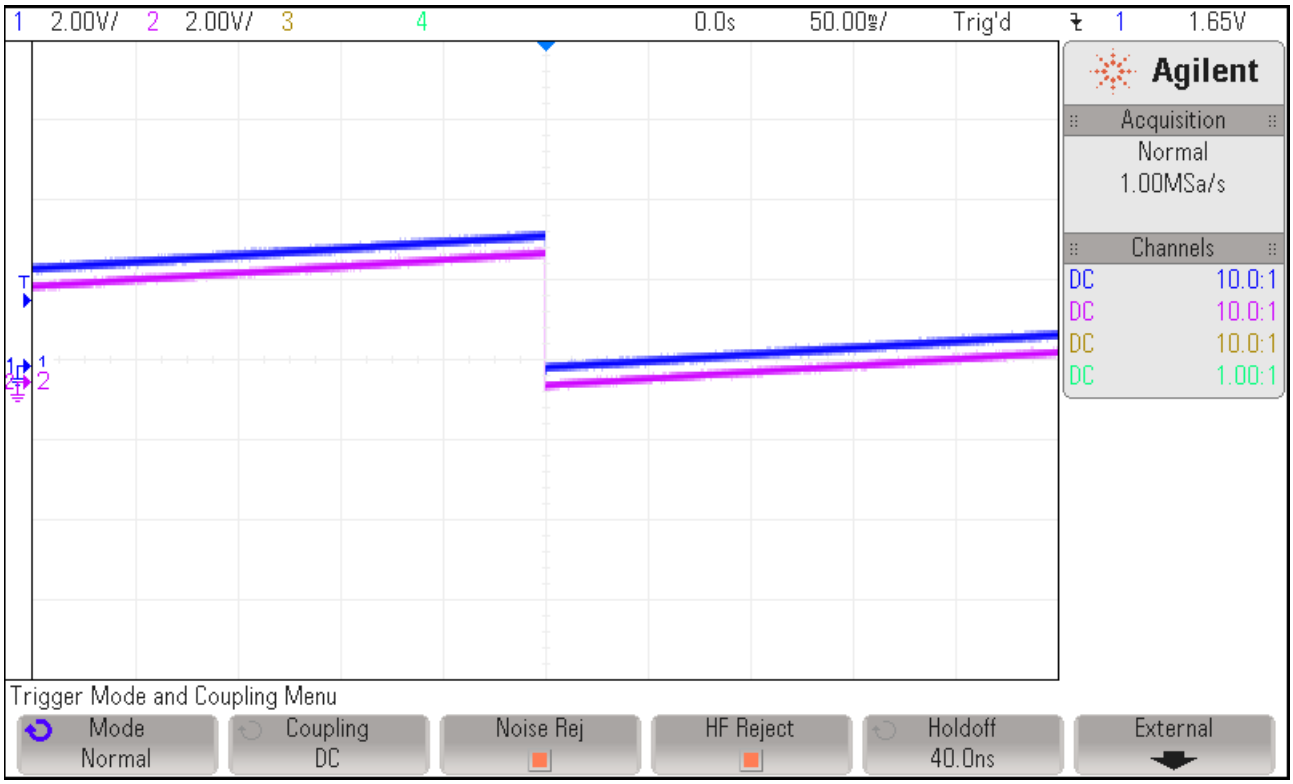


Figure 39. Scope Plot of the Reference Position to the Servo and Feedback Position

NOTE: 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 or set *motor1.PosSenseReverse* to 1 and repeat the test.

NOTE: The position control implemented in this project 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.

Copyright © 2015, Texas Instruments Incorporated

NOTE: With an absolute encoder like resolver/EnDAT/BiSS, the above may not be an issue as the resolver gives unique angular value for each position.

As an experiment, the user may generate a position reference by integrating an assumed speed reference and feed it to the position loop and see how the position loop tracks the reference.

Copyright © 2015, Texas Instruments Incorporated

14 About the Author

RAMESH RAMAMOORTHY works as a motor drives and power electronics expert in the C2000 business unit of Texas Instruments. He has worked extensively on various motor drives and control topologies for a variety of applications. Ramesh holds a Master of Technology degree from the Indian Institute of Technology, Madras India. He has also published a few technical papers and holds a few patents.

Copyright © 2015, Texas Instruments Incorporated

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ("TI") reference designs are solely intended to assist designers (“Buyers”) who are developing systems that incorporate TI semiconductor products (also referred to herein as “components”). Buyer understands and agrees that Buyer remains responsible for using its independent analysis, evaluation and judgment in designing Buyer’s systems and products.

TI reference designs have been created using standard laboratory conditions and engineering practices. **TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.** TI may make corrections, enhancements, improvements and other changes to its reference designs.

Buyers are authorized to use TI reference designs with the TI component(s) identified in each particular reference design and to modify the reference design in the development of their end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO TI REFERENCE DESIGNS OR USE THEREOF. TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY BUYERS AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON A COMBINATION OF COMPONENTS PROVIDED IN A TI REFERENCE DESIGN. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF TI REFERENCE DESIGNS OR BUYER’S USE OF TI REFERENCE DESIGNS.

TI reserves the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

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

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

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

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

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

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

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

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

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2015, Texas Instruments Incorporated