Application Report
SPRAC55–May 2017

High-Bandwidth Current Control of 3-Phase PMSM Using the F2837x Fast Current Loop Library

Ramesh Ramamoorthy, Santosh Kumar Athuru

ABSTRACT

This application note helps to evaluate the fast current loop (FCL) software library for high-bandwidth inner loop control of AC servo drives based on the TMS320F2837x MCU using TI's IDDK evaluation platform. TMS320F2837x devices are a part of the C2000™ family of microcontrollers, which enable cost-effective design of intelligent, high-bandwidth controllers for 3-phase motors, by reducing the system components and increasing efficiency. The devices are appropriately supported by hardware accelerators to implement FCL algorithms to rival or surpass similar implementations using FPGA in terms of performance, cost, and development time.

The application report describes the following:

- Incremental build levels calling modular FCL functions spread across CPU and CLA
- Experimental results

Abbreviations

- DMC – Digital Motor Control
- IDDK – Industrial Drive Development Kit (from TI)
- MCU – Microcontroller Unit
- FOC – Field-Oriented Control
- TMU – Trigonometric Mathematical Unit (in C2000 MCU)
- CLA – Control Law Accelerator (in C2000 MCU)
- CLB – Configurable Logic Block (in C2000 MCU)
- PMSM – Permanent Magnet Synchronous Motor
- ACIM – AC Induction Motor
- FCL – Fast Current Loop
- HVDMC – High Voltage DMC
- CMPSS – Comparator Subsystem peripheral (in C2000 MCU)
- CNC – Computer Numerical Control
- PWM – Pulse Width Modulation
- FPGA – Field Programmable Gate Array
- ADC – Analog-to-Digital Converter
- ePWM – Enhanced Pulse Width Modulator
- eQEP – Enhanced Quadrature Encoder Pulse Module
- eCAP – Enhanced Capture Module

Contents

List of Figures

List of Tables

1 Introduction

The concept of FOC of AC drives is well known and is already outlined in many earlier documents from TI. Modern AC servo drives, depending on the end application, need high-bandwidth current control and speed control to enable superior performance, such as in CNC machines or in fast and precision control applications. Because of the high computational burden and the need for flexible PWMs, a combination of FPGAs, fast external ADCs, and multiple MCUs are used by many designers.

Now with the TMS320F2837x MCU, due to its higher level of integration, it is possible to implement fast current loop algorithms with the same external hardware used in classical FOC methods. TI has developed the FCL algorithm on the F2837x MCU and implemented it on the DesignDRIVE IDDK platform. With a 10-kHz PWM carrier, the current loop bandwidth is expected to exceed 3 kHz and the maximum duty cycle is expected to be about 96%. Quantitative test results will be published when they are available. This document helps the user evaluate this algorithm on the DesignDRIVE IDDK platform.

2 Benefits of the TMS320F2837x MCU for High-Bandwidth Current Loop

The C2000 MCU family of devices possesses the desired computation power to execute complex control algorithms and the correct combination of peripherals to interface with the various components of the DMC hardware such as the ADC, ePWM, QEP, and eCAP. These peripherals have all the necessary hooks to provide flexible PWM protection, like trip zones for PWMs and comparators.

The F2837x MCU contains additional hardware features such as the following:

- Higher CPU and CLA clock frequency
- Four high-speed, 12- and 16-bit ADCs
- TMU
- Parallel processing block such as the CLA
- CLBs

Together, these features provide enough hardware support to increase computational bandwidth compared to its predecessors and offer superior real-time control performance. In addition, the C2000 ecosystem of software (libraries and application software) and hardware (TMDXIDDK379D) help users reduce the time and effort needed to develop a high-end digital motor control solution.

3 Current Loops in Servo Drives

Figure 1 shows the basic current loop used in FOC servo drives.

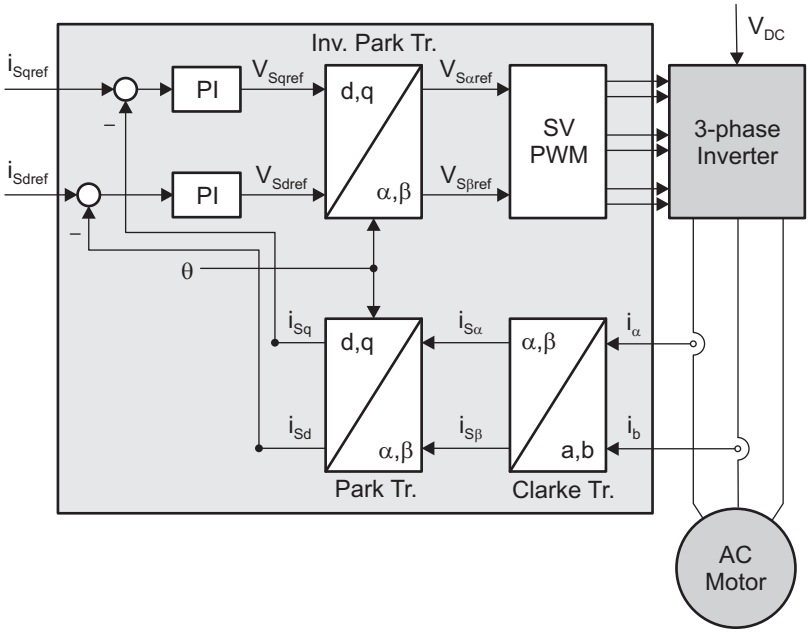


Figure 1. Basic Scheme of FOC for AC Motor

Two motor phase currents are measured. These measurements feed the Clarke transformation module. The outputs of this projection are designated $i_{s\alpha}$ and $i_{s\beta}$. These two components of the current along with the rotor flux position are the inputs of the Park transformation, which transform them to currents (i_{sd} and i_{sq}) in d and 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 point, the control structure shows an interesting advantage; it can be used to control either synchronous or asynchronous machines by simply changing the flux reference and obtaining the rotor flux position. In the synchronous permanent magnet motor, the rotor flux is fixed as determined by the magnets, so there is no need to create it. Therefore, when controlling a PMSM motor, i_{sdref} can be set to zero, except during field weakening.

Because ACIM motors need a rotor flux creation to operate, the flux reference must not be zero. This conveniently solves one of the major drawbacks of the classic control structures: the portability from asynchronous to synchronous drives. The torque command i_{sqref} can be connected to the output of the speed regulator. The outputs of the current regulators are V_{sdref} and V_{sqref}. These outputs 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 components 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 need the rotor flux position. Obtaining this rotor flux position depends on the AC machine type (synchronous or asynchronous).

4 Outline of the Fast Current Loop Library

The major challenge in digital motor control systems is the influence of sample and hold, as well as transportation lag inside the loop that slows down the system, impacting its performance at higher frequencies and running speeds. To overcome this problem, improve the current loop bandwidth, and enable higher DC bus usage, the following are necessary:

- High computational power
- The correct set of control peripherals
- Superior control algorithm

While the TMS320F2837x provides the necessary hardware support for higher performance, TI's FCL library, which runs on this C2000 MCU, provides the necessary algorithmic support.

To improve the operational range of FCLs, the latency between feedback sampling and the PWM update must be as small as possible. Typically, a latency of 2 µS or less is considered acceptable in many applications. Traditionally, this task is implemented using a combination of high-end FPGAs, external ADCs, and MCUs.

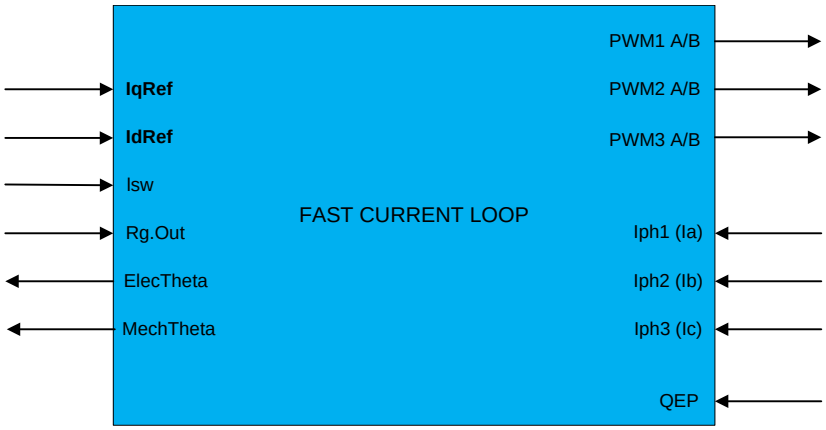


Figure 2. Fast Current Loop Library Block Diagram

The FCL library uses the following features in the F2837x MCU:

- TMU
- Four high-speed 12- and 16-bit ADCs
- Multiple parallel processing blocks such as CLA

Figure 2 shows the block diagram of the FCL library with its inputs and outputs. The FCL library partitions the algorithm across the CPU, CLA, and TMU to bring down the latency to under 1.0 µs compared to the acceptable 2 µs. Further optimization is possible if the algorithm is written in assembly.

The FCL library supports two types of current regulators, a standard PI controller and a complex controller. The complex controller can provide additional bandwidth over the standard PI controller at higher speeds. Both current regulators are provided for user evaluation. In the example project for which this document provides steps, the user can select the current regulator by setting the FCL_CNTLR macro appropriately and studying how they compare.

Table 1 lists the FCL API functions and their descriptions.

Table 1. Summary of FCL Interface Functions

API Function	Description
Uint32 FCL_GetSwVersion(void)	Function that returns a 32-bit constant, and for this version the value returned is 0x00000002.
void FCL_Complex_Ctrl(void)	Function that performs the complex control as part of the FCL
void FCL_PI_Ctrl(void)	Function that performs the PI control as part of the FCL
void FCL_PI_CtrlWrap(void)	Wrap up function to be called by the user application at the completion of FCL in PI control mode before exiting ISR
void FCL_QEP_wrap(void)	Function to be called by the user application to wrap up the QEP feedback procedure. This function is used only in FCL_LEVE2.
void FCL_Complex_CtrlWrap(void)	Wrap up function to be called by the user application at the completion of FCL in complex control mode before exiting ISR
void FCL_initPWM(volatile struct EPWM_REGS *ePWM);	Function to initialize PWMs for the FCL operation, this will be called by the user application during the initialization or setup process.
void FCL_ControllerReset(void)	This function is called to reset the FCL variables and is useful when the user wants to stop the motor and restart the motor.

For more information on the library, see the *Fast Current Loop Library API Reference Guide* available at controlSUITE\libs\app_libs\motor_control\libs\FCL\v02_00_00_00\lib.

NOTE: The library is written in a modular format and is able to port over to user platforms using F2837x devices so long as the following conditions are met:

- Motor phase current feedbacks are read into variables internal to the library. However, D axis and Q axis current feedbacks are available to the user.
- EPwmRegs controlling motor phase A, B, and C are duly linked to the library.
- QepRegs connecting to the QEP sensor is duly linked to the library.
- CLA tasks one through four are used by the library and the user can accommodate this in their application.

5 Fast Current Loop Library Evaluation

TI provides the FCL algorithm software library for evaluation using the TMS320F2837x MCU on TI's DesignDRIVE IDDK platform. This section presents a step-by-step approach to evaluating the FCL software library to control a permanent magnet synchronous motor using an example project.

Example project features:

- Sensored FOC of the PMSM motor
- FCL using the FCL library
- Position, speed, and torque control loops
- Position sensor support: incremental encoder (QEP)
- Current sensing: analog feedback using the ADC from the LEM sensors (Fluxgate/HALL)

This example project is derived out of the native IDDK software project at controlSUITE\development_kits\TMDSIDDK_v2.0\IDDK_PM_Servo_F2837x_v2_00_00_00.

The user is therefore advised to evaluate the native project and become familiar with the hardware and software before evaluating this example project. This guide must be regarded as augmenting the original guide used in the native project, and therefore basic information is not repeated. The original guide is available at controlSUITE\development_kits\TMDSIDDK_v2.0\IDDK_PM_Servo_F2837x_v2_00_00_00\~Docs

5.1 Evaluation Setup

5.1.1 Hardware

The example project is evaluated on TI's Design DRIVE Development Kit IDDK - TMDXIDDK379D.

It is important to read the Hardware Reference Guide of the kit and understand all the safety measures necessary to work with the kit.

The Hardware Reference Guide can be found at controlSUITE\development_kits\TMDSIDDK_v2.0\~Docs\

5.1.2 Software

When the FCL software package is installed, the FCL software library can be found at controlSUITE\libs\app_libs\motor_control\libs\FCL\v02_00_00_00\lib

The FCL example project can be found at controlSUITE\libs\app_libs\motor_control\libs\FCL\ v02_00_00_00\Examples

Here, a few more build levels are added to step through with integrating the FCL library to run the motor in speed mode and position mode.

5.1.3 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 modules are written as callable functions. Table 2 summarizes the module testing and use in each incremental system build.

Table 2. Testing Modules in Each Incremental System Build

Software Module	Level 2	Level 3	Level 4	Level 5
QEP Interface in CLA	√√	√	√	√
SPEED_FR_MACRO	√√	√	√	√
fastCurrentLoop_PI ( ) fastCurrentLoop_Complex ( )		√√	√	√
PID_MACRO (SPD)			√√	√
PI_POS_MACRO (POS)				√√

6 Level 2 Incremental Build

Assuming build 1 of the native project is completed successfully, this section verifies the overcurrent protection limits of the inverter and QEP interface running out of the CLA. In this build, the motor is run in open loop.

The motor can be connected to the HVDMC board because the PWM signals are successfully proven through the level 1 incremental build.

1. Open FCL_IDDK_PM_Servo_F2837x-Settings.h and select the level 2 incremental build option by setting the BUILDLEVEL to FCL_LEVEL2 (#define BUILDLEVEL FCL_LEVEL2).
2. Select CURRENT_SENSE to LEM_CURRENT_SENSE and POSITION_ENCODER to QEP_POS_ENCODER.
3. Right-click on the project name and click *Rebuild Project*.
4. When the build is complete, click *Debug*, reset the CPU, restart, enable real-time mode, and run.

Figure 3 shows import variables from the *Variables_FCL_IDDK.txt* file in the root directory and the Expressions window.

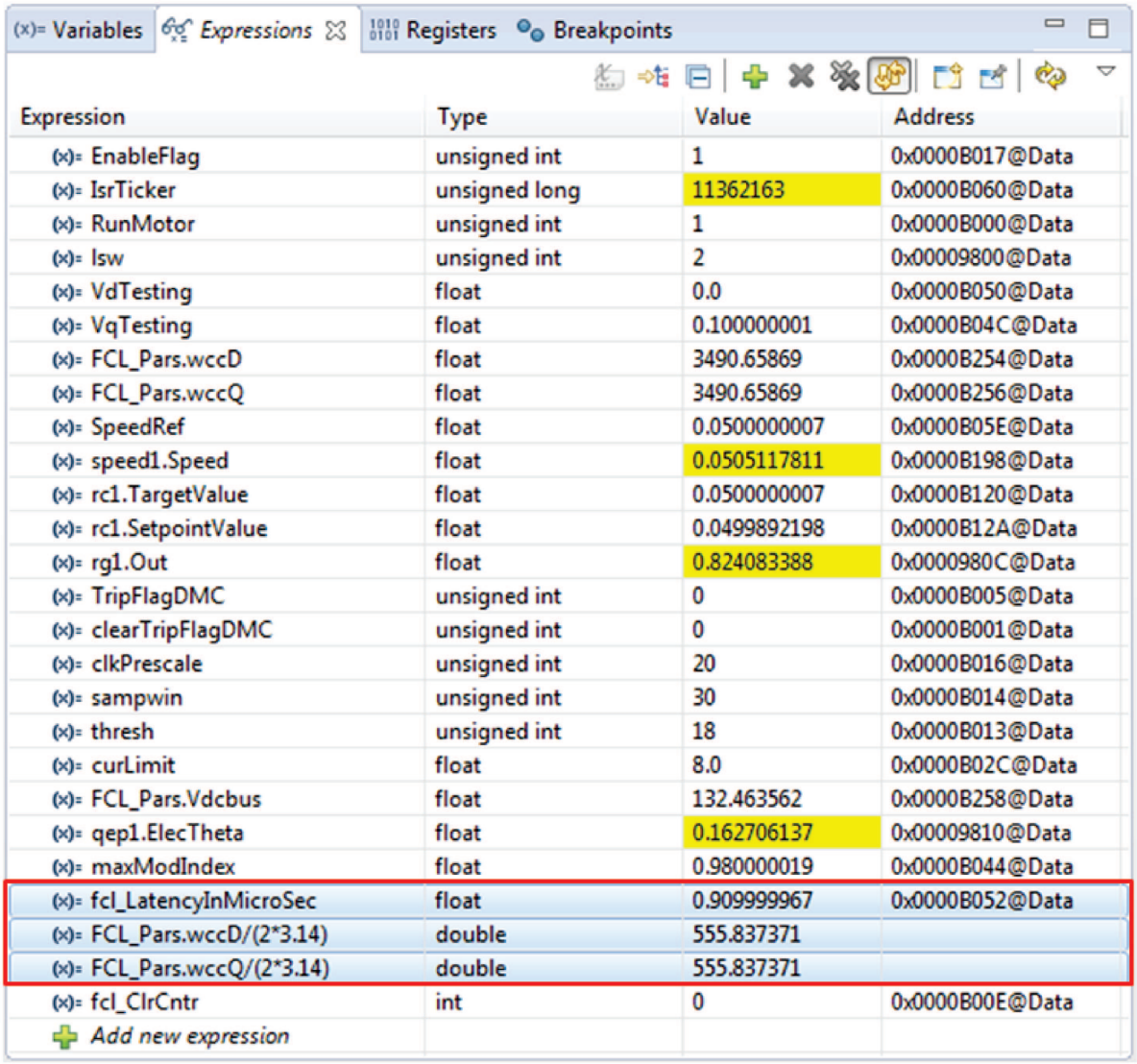


Figure 3. Expressions Window for Build Level 2

Figure 4 shows the level 2 block diagram.

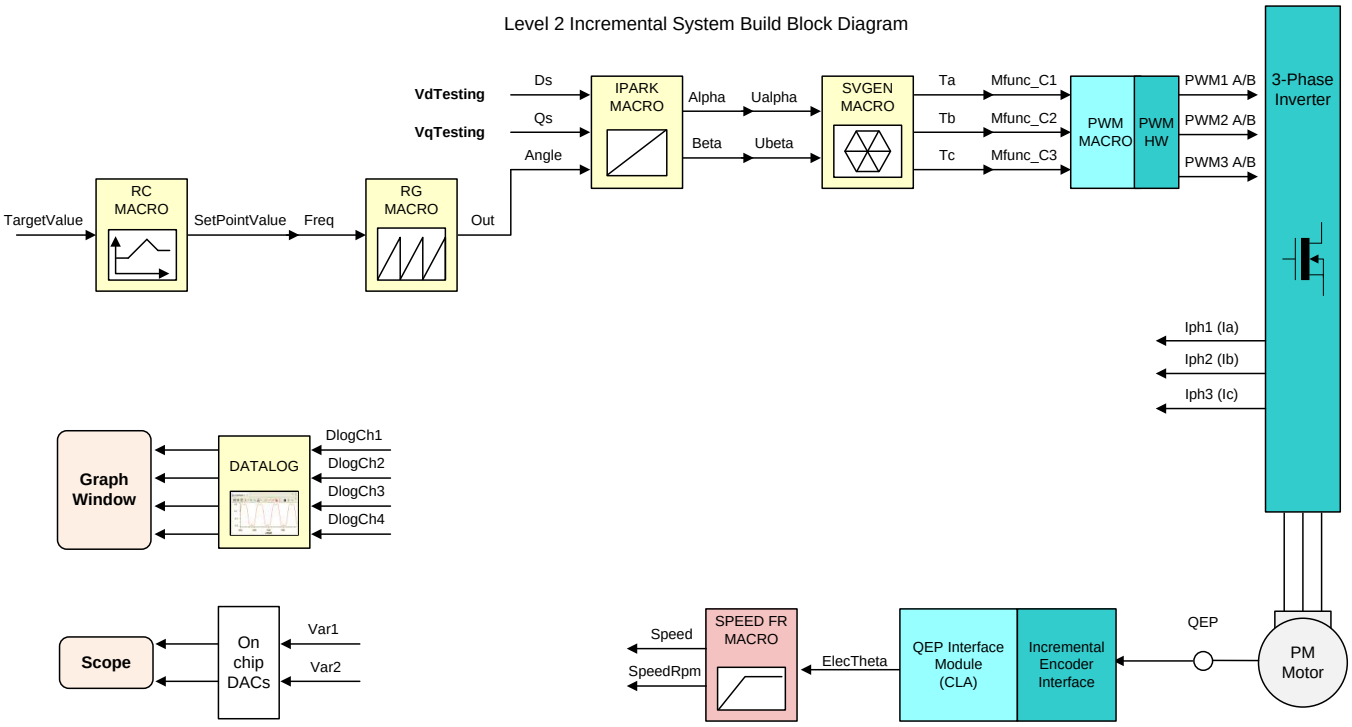


Figure 4. Level 2 Block Diagram

Set EnableFlag to 1 in the watch window. The variable named IsrTicker is incrementally increased as shown in the Expressions window to confirm the interrupt is working properly. Now set the variable named RunMotor to 1; the motor starts spinning after a few seconds if enough voltage is applied to the DC-Bus.

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

- SpeedRef: for changing the rotor speed in per-unit
- VdTesting: for changing the d-axis 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 must be adjusted carefully for PM motors so that the generated Bemf is lower than the average voltage applied to motor winding. This adjustment prevents the motor from stalling or vibrating.

6.1 Setting the Overcurrent Limit in the Software

The board has various current sense methods, such as shunt, LEM, and SDFM. Overcurrent monitoring is provided for signals generated from shunt and LEM using an on-chip comparator subsystem (CMPSS) module. The module 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, verifying its sanity by periodically sampling and validating the signal for a certain count time within a certain count window, where the periodicity, count, and count window are user programmable.

In the Expressions window, users can see the following variables:

- clkPrescale – sets the sampling frequency of the digital filter
- sampwin – sets the count window
- thresh – sets the minimum count to qualify the signal within sampwin
- curLimit – sets the permitted current maximum through both shunt and LEM current sensors

TripFlagDMC is a flag variable that represents the overcurrent trip status of the inverter. If this flag is set, then the user can adjust the previous settings and try to rerun the inverter by setting clearTripFlagDMC to 1. This clears TripFlagDMC and restarts the PWMs.

The default current limit setting is to shut down at 8 A. The user can fine-tune any of these settings to suit their system. Once satisfactory values are identified, write them down, modify the code with these new values, and rebuild and reload for further tests.

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

6.2 Current Sense Method

The CURRENT_SENSE method chosen for this library is LEM_CURRENT_SENSE. It is possible to use the SHUNT_CURRENT_SENSE method in certain configurations, but it is not included in the library.

6.3 Voltage Sense Method

Voltage is sensed using the sigma delta filter module 3. Look out for the variable *FCL_Pars.Vdcbus* in the Expressions window. Vary the DC bus voltage slowly and verify whether this variable tracks this change properly. For example, a 100-V DC voltage should be shown as 100.0 by this variable.

6.4 Setting Current Regulator Limits

The outputs of the current regulators control the voltages applied on both the d-axis and q-axis. The vector sum of the d and q outputs must be less than 1.0, which refers to the maximum duty cycle for the SVGEN macro. In this particular application, the maximum allowed duty cycle is set to 0.96. Higher computational speeds allow higher duty cycle operation and better use of the DC bus voltage.

The current regulator output is represented by the same variable pi_id.Out and pi_iq.Out in both PI and complex controller modes. The regulator limits are set by pi_id.Umax/min and pi_iq.Umax/min.

Bring the system to a safe stop by reducing the bus voltage to zero, taking the controller out of real-time mode, and resetting.

6.5 Position Encoder Feedback and SPEED_FR Test

During all the previous tests, the position encoder interface was continuously estimating position information. Therefore no new code is needed to verify the position encoder interface. When the motor is commanded to run, it is subjected to 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. Figure 5 shows these signals brought out on H10 on the IDDK, and their scope plots.

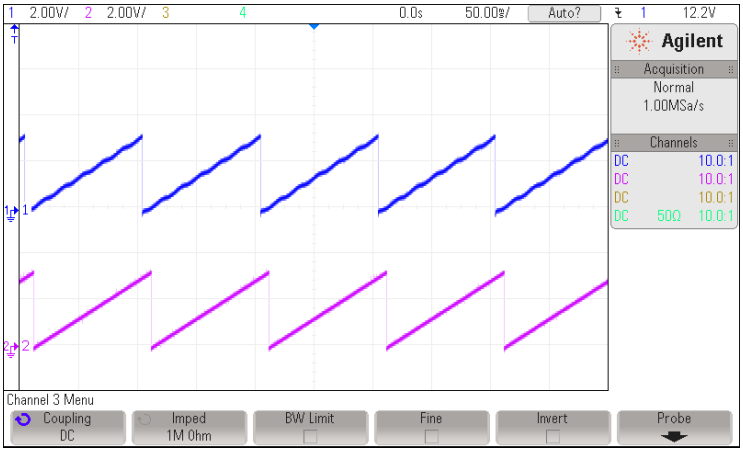
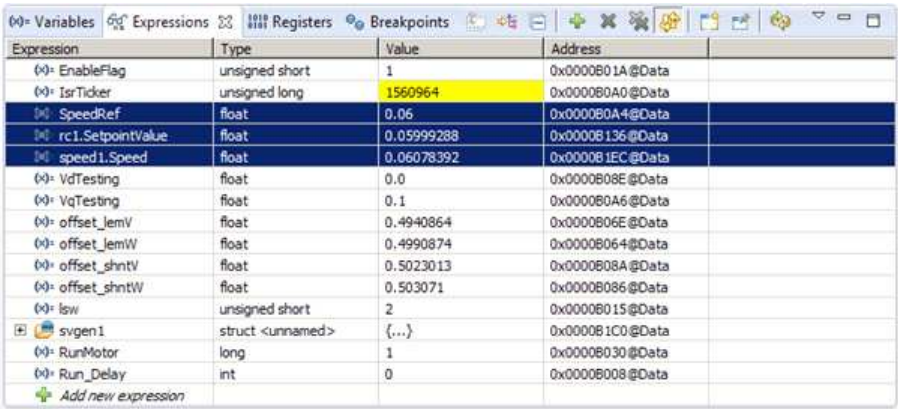


Figure 5. Scope Plot of Reference Angle and Rotor Position

The waveform of channel 2 represents the reference position, while channel 1 represents the estimated position. The ripple in position estimate is indicative of the fact that the motor runs with some minor speed oscillation. Because of open-loop control, the rotor position and reference position may not align. However, it is important to ensure 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 by either swapping any two wires connecting to the motor, or reversing the angle estimate as in the pseudocode in the software (see Equation 1).

$$angle = 1.0 - angle \quad (1)$$

To ensure that the SPEED_MACRO works, change the SpeedRef variable in the Expressions window, as shown in Figure 6, and check whether the estimated speed variable, speed1.Speed, follows the commanded speed. Because the motor is a PM motor, where there is no slip, the running speed follows the commanded speed regardless of the control being open loop.



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_IemV	float	0.4940864	0x0000B06E@Data
offset_IemW	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 6. Expressions Window

When the tests are complete, bring the system to a safe stop by reducing the bus voltage, taking the controller out of real-time mode and resetting it. Now the motor stops.

7 Level 3 Incremental Build

Assuming the previous section is completed successfully, this section verifies the dq-axis current regulation performed by the FCL. The user can choose to call one of the two current controllers, PI or complex. The bandwidth of the controllers can be set in the debug window.

NOTE: In this build, control is done based on the actual rotor position; therefore, the motor can run at higher speeds if the commanded IqRef is higher and there is no load on the motor. TI advises users to either add some mechanical load on the motor before the test or apply lower values of IqRef. When the motor is commanded to run, it is subjected to an initial alignment stage where the electrical angle and the QEP angle count are set to zero. They are then forced to run based on an enforced angle until the QEP index pulse is received. Then the motor runs in full self-control mode based on its own angular position.

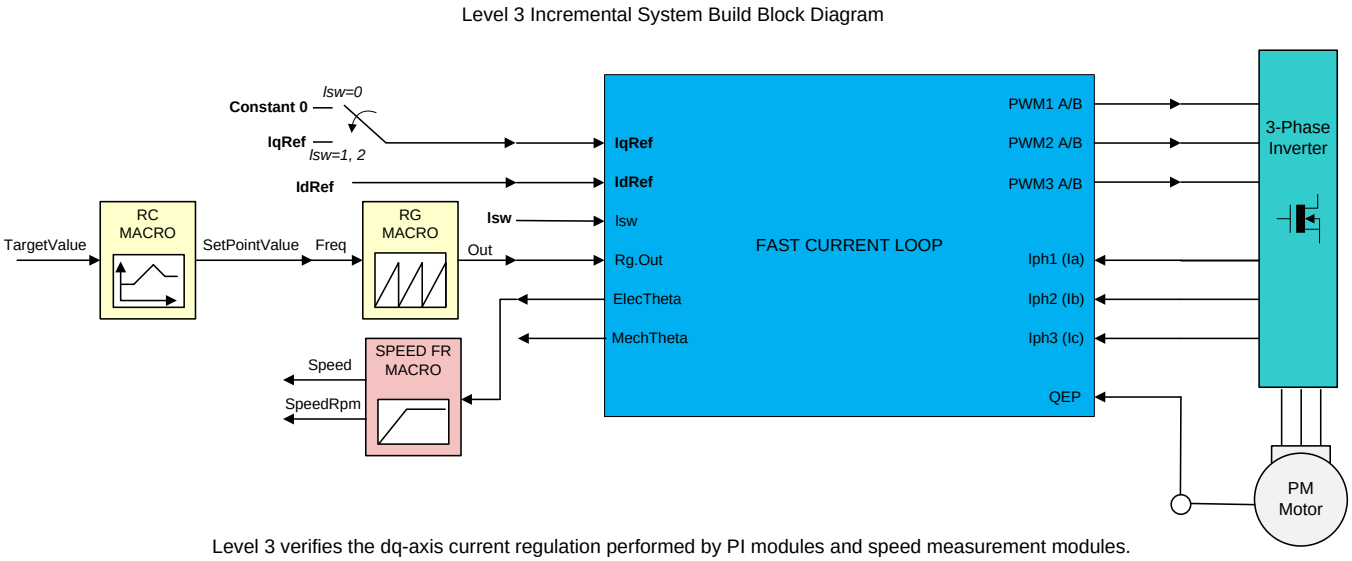
Open FCL_IDDK_PM_Servo_F2837x-Settings.h and select the level 3 incremental build option by setting BUILDLEVEL to FCL_LEVEL3 (#define BUILDLEVEL FCL_LEVEL3). The user can select the current loop regulator to be the PI controller or the complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR. The CURRENT_SENSE method chosen for this library is LEM_CURRENT_SENSE. The current and position feedbacks can be sampled once or twice per PWM period depending on the sampling method. The sampling is synchronized to the carrier maximum in the single sampling method, and to the carrier maximum and carrier zero in the double sampling method. This sample method selection is done in the example by selecting SAMPLING_METHOD to SINGLE_SAMPLING or SAMPLING_METHOD to DOUBLE_SAMPLING. Notice that the maximum modulation index changes from 0.98 in the SINGLE_SAMPLING method to 0.96 in the DOUBLE_SAMPLING method. If the PWM_FREQUENCY is changed from 10 kHz, the maximum modulation index also changes.

Right-click on the project name, and then click *Rebuild Project*. When the build is complete, click the *Debug* button, reset the CPU, restart, enable real-time mode, and run.

In the software, the key variables to add, adjust, or monitor are summarized as follows:

- maxModIndex: maximum modulation index
- IdRef: changes the d-axis voltage in per-unit
- IqRef: changes the q-axis voltage in per-unit
- FCL_Pars.WccD: preferred bandwidth of d-axis current loop
- FCL_Pars.WccQ: preferred bandwidth of q-axis current loop
- fcl_LatencyInMicroSec: shows latency between ADC and QEP sampling, and PWM in µs
- fcl_ClrCntr: flag to clear the variable fcl_latency and let it refresh
- RunMotor: flag to run or stop the motor

Figure 7 shows the level 3 block diagram.



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

Figure 7. Level 3 Block Diagram Showing Inner Most Loop - FCL

The key steps are explained as follows:

1. Set EnableFlag to 1 in the watch window. The IsrTicker variable is incrementally increased, as seen in watch windows to confirm the interrupt is working properly.
2. Verify if the maxModIndex value is either 0.96 in double-sampling method or 0.98 in single-sampling method.
3. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different), Idref to zero, and Iqref to 0.03 pu (or another suitable value). SpeedRef helps only until the QEP index pulse is received. Thereafter, the motor is controlled based on its rotor position.

   The soft-switch variable (lsw) is autopromoted in a sequence inside the FCL library. Here, lsw manages the loop setting as follows:

   - lsw = 0, lock the rotor of the motor
   - lsw = 1, for QEP feedback only – motor in run mode and waiting for the first instance of the QEP index pulse
   - lsw = 2, motor in run mode, for all encoders (for QEP – first index pulse occurred)
4. Gradually increase voltage at variac / DC power supply to, for example, 20% of the rated voltage.
5. Set RunMotor flag to 1 to run the motor.
6. Check pi_id.Fbk in the watch windows with the continuous refresh feature and see if it can track IdRef.
7. Check pi_iq.Fbk in the watch windows with the continuous refresh feature and see if it can track IqRef.
8. To confirm these two current regulator modules, try different values of pi_id.Ref and pi_iq.Ref, by changing the values of IdRef and IqRef, respectively.
9. Try different bandwidths for the current loop by tweaking the values of FCL_Pars.wccD and FCL_Pars.wccQ. The default setting for the bandwidth is 1/18 of the sampling frequency.
10. If the motor shaft can be held tight, then the IqRef value can be changed back and forth from 0.5 to –0.5, to study the effect of loop bandwidth.
11. Bring the system to a safe stop by reducing the bus voltage, taking the controller out of real-time mode, and resetting. Now the motor stops.

7.1 Observation One – Latency

7.1.1 From the Expressions Window

While running the motor in this build level and subsequent build levels, notice the variable fcl_LatencyInMicroSec in the Expressions window.

Figure 8 shows a snapshot of the Expressions window.

Figure 8. Expressions Window Snapshot For Latency

This variable indicates the amount of time elapsed between the feedback sampling and PWM updating. The elapsed time, or latency, is computed based on the count of the EPWM timer right after the PWM update. The value shown here is more than the actual update time by a few clock cycles. Right after starting to run the motor, by setting the RunMotor flag to 1, the latency time shows as around 1.25 µS due to initial setup in the code. This amount of latency occurs at a time when the duty cycle is moderate and therefore acceptable. Right after this period, the user can refresh the latency time by setting fcl_clrcntr to 1. Regardless of SAMPLING_METHOD, latency remains the same for a given FCL_CNTLR. When FCL_CNTLR is a PI_CNTLR, the latency is about 0.91 µs compared to 0.94 µs with a CMPLX_CNTLR (see the following note).

NOTE: These times can be reduced further by around 0.1 µs range using code inling and other optimization techniques. Because the evaluation code is in library format, it has certain overheads.

7.1.2 From the Scope Plot

NOTE: Because H7 is not populated, GPIO16 and GPIO18 are used for timing purposes instead of for designed functional assignment. If H7 is to become populated, remember to comment out the associated code and restore the functional assignment.

Figure 9 shows the latency discussed previously in the form of a scope plot, where the rising edge of the channel 2 and channel 1 waveforms signify the instances of the ADC SoC event and completion of all PWM updates, respectively. These events are brought out over GPIO16 and GPIO18, and may be probed over [Main]-R31 and [Main]-R33, respectively. The time seen in the scope plot may be slightly more than fcl_LatencyinMicroSec. This additional time is to run the code for the GPIO toggle.

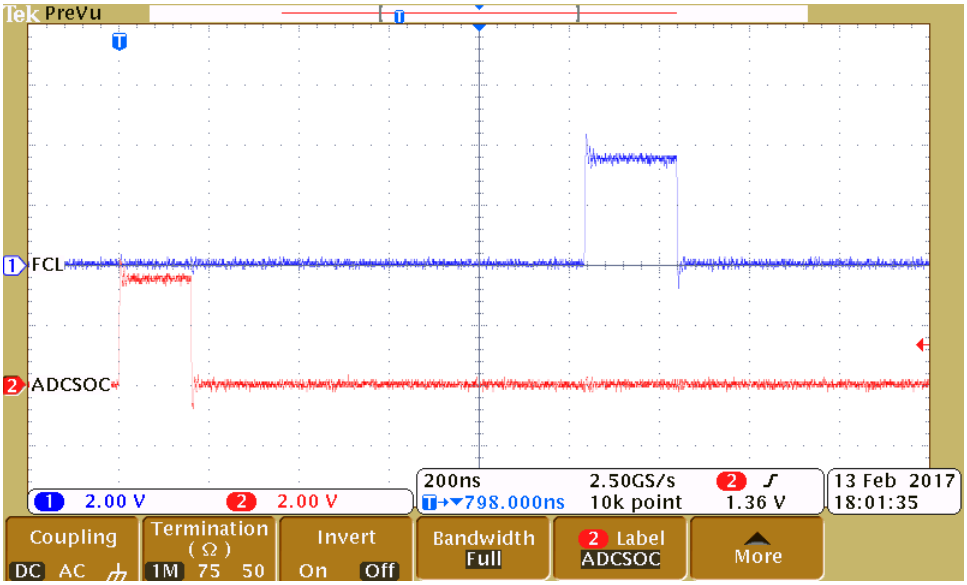


Figure 9. Scope Plot of ADCSoC and FCL Completion Events

8 Level 4 Incremental Build

Assuming the previous section is completed successfully, this section verifies the speed PI module and speed loop. When the motor is commanded to run, it is subjected to an initial alignment stage where the electrical angle and the QEP angle count are set to zero. After ensuring a stable alignment, the motor will start running.

Open FCL_IDDK_PM_Servo_F2837x-Settings.h and select level 4 incremental build option by setting the BUILDLEVEL to FCL_LEVEL4 (#define BUILDLEVEL FCL_LEVEL4). The user can select the current loop regulator to be PI controller or complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR. The CURRENT_SENSE method chosen for this library is LEM_CURRENT_SENSE.

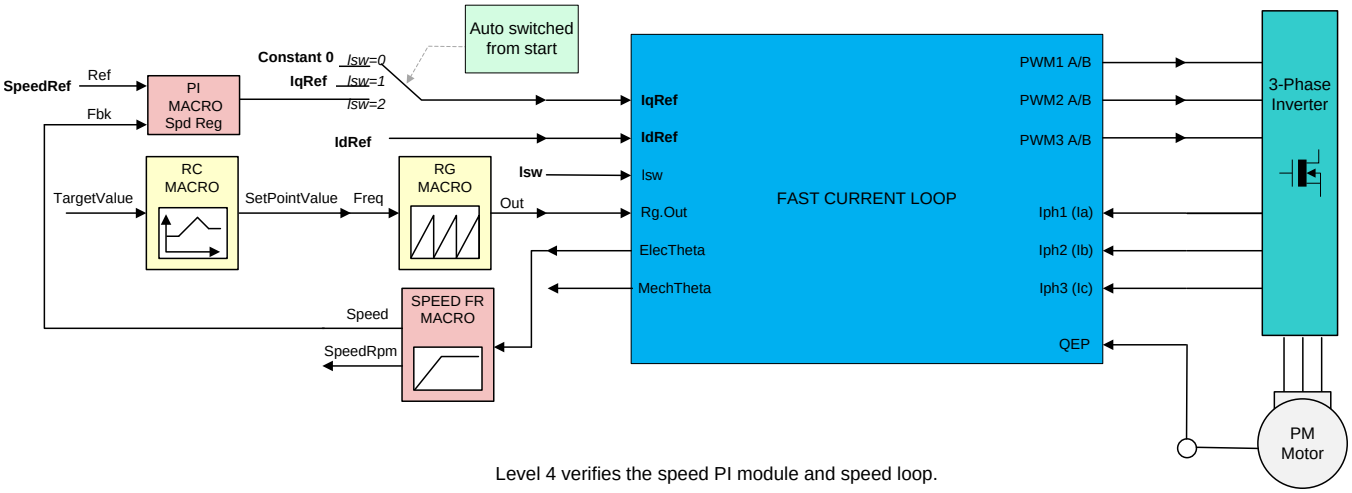
Right-click on the project name, and then click *Rebuild Project*. When the build is complete, click the *Debug* button, reset the CPU, restart, enable real-time mode, and run.

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

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

Figure 10 shows the implementation block diagram.

Level 4 Incremental System Build Block Diagram



Level 4 verifies the speed PI module and speed loop.

Figure 10. Level 4 Block Diagram Showing Speed Loop With Inner FCL

The key steps are explained as follows:

1. Set EnableFlag to 1 in the watch window. The IsrTicker variable is incrementally increased as seen in watch windows to confirm the interrupt is working properly.
2. Set SpeedRef to 0.3 pu (or another suitable value if the base speed is different).
3. Add pid_spd variable to the Expressions window
4. Gradually increase voltage at variac to get an appropriate DC-bus voltage.
5. Add the switch variable RunMotor to the watch window to start the motor. The soft-switch variable (lsw) is autopromoted in a sequence. In the code lsw manages the loop setting as follows:
   - lsw = 0, lock the rotor of the motor
   - lsw = 1, for QEP feedback only – motor in run mode and waiting for first instance of QEP index pulse
   - lsw = 2, motor in run mode, for all encoders (for QEP – first Index pulse occurred)
6. Set RunMotor to 1 and now the motor runs with this reference speed (0.3 pu). Compare the speed with SpeedRef in the watch windows with the continuous refresh feature to see whether or not it is nearly the same.
7. To confirm this speed PID module, try different values of SpeedRef (positive or negative). The P, I and D gains may be tweaked to get a satisfactory response.
8. At a very low speed range, the performance of the speed response relies heavily on the good rotor position angle provided by the QEP encoder.
9. Bring the system to a safe stop by reducing the bus voltage, taking the controller out of real-time mode, and resetting. Now the motor stops.

Figure 11 shows flux and torque components of the stator current in the synchronous reference frame.

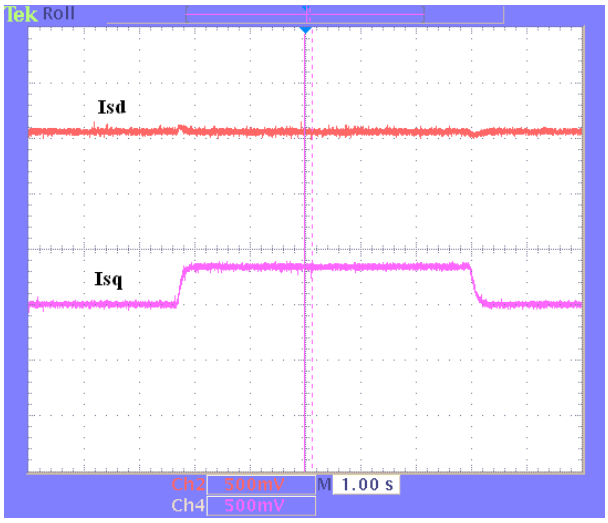


Figure 11. Flux and Torque Components of the Stator Current in the Synchronous Reference Frame Under 0.33-pu Step-Load and 0.3-pu Speed

8.1 Observation

In the default set up, the current loop bandwidth is set up as 1/18 of the SAMPLING FREQUENCY for both CMPLX_CNTLR and PI_CNTLR. As the bandwidth is increased, control becomes stiff and the motor operation becomes noisier as the controller reacts to tiny perturbations in the feedback trying to correct them. With CMPLX_CNTLR, the bandwidth can be taken up to 1/6 of the SAMPLING_FREQUENCY and get good transient response over the entire speed range including higher speeds. Whereas with PI_CNTLR, at such higher bandwidth, the transient performance at low to mid speed range of the motor may be comparable to that of a CMPLX_CNTLR, but at high speeds it may be inferior. The user can experiment with different values for bandwidth and evaluate the performance. If the motor rotation direction is reversed occasionally due to a malfunction, try restarting it by setting RunMotor to 0 and then 1 again. It may need some fine tuning in transitioning from lsw = 1 to lsw = 2. The user can take it as an exercise to fix it.

9 Level 5 Incremental Build

This section verifies the position PI module and position loop with a QEP. For this loop to work properly, the speed loop must have been completed successfully. When the motor is commanded to run, it is subjected to an initial alignment stage where the electrical angle and the QEP angle count are set to zero. After ensuring a stable alignment, the motor will start to run.

Open FCL_IDDK_PM_Servo_F2837x-Settings.h and select level 5 incremental build option by setting the BUILDLEVEL to FCL_LEVEL5 (#define BUILDLEVEL FCL_LEVEL5). The user can select the current loop regulator to be PI controller or complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR. The CURRENT_SENSE method chosen for this library is LEM_CURRENT_SENSE.

Right-click on the project name, and then click *Rebuild Project*. When the build is complete, click the *Debug* button, reset the CPU, restart, enable real-time mode, and run. 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. This module basically cycles the position reference through a set of values as defined in an array posArray. These values represent the number of the rotations and turns with respect to the initial alignment position. Once a certain position value as defined in the array is reached, it pauses for a while before slewing toward the next position in the array. Therefore these array values can be referred to 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 through which it passes 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 are explained as follows:

1. Set EnableFlag to 1 in the watch window. The variable named IsrTicker is incrementally increased as seen in the watch windows to confirm the interrupt is working properly.
2. Add variables pi_pos, posArray, ptrMax, and posSlewRate to the Expressions window.
3. Gradually increase voltage at variac to get an appropriate DC-bus voltage.
4. Set RunMotor to 1 to run the motor. The motor must be turning to follow the commanded position (see the following note if the motor does not turn properly).
5. The parking positions in posArray can be changed to different values to determine 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. The position slew rate can be changed using posSlewRate. This rate represents the angle (in pu) per sampling instant.
8. The proportional and integral gains of the speed and position PI controllers may be retuned to get satisfactory responses. TI advises to first tune the speed loop and then the position loop.
9. Bring the system to a safe stop by reducing the bus voltage, taking the controller out of real-time mode and reset. Now the motor stops.

Figure 12 shows the implementation block diagram.

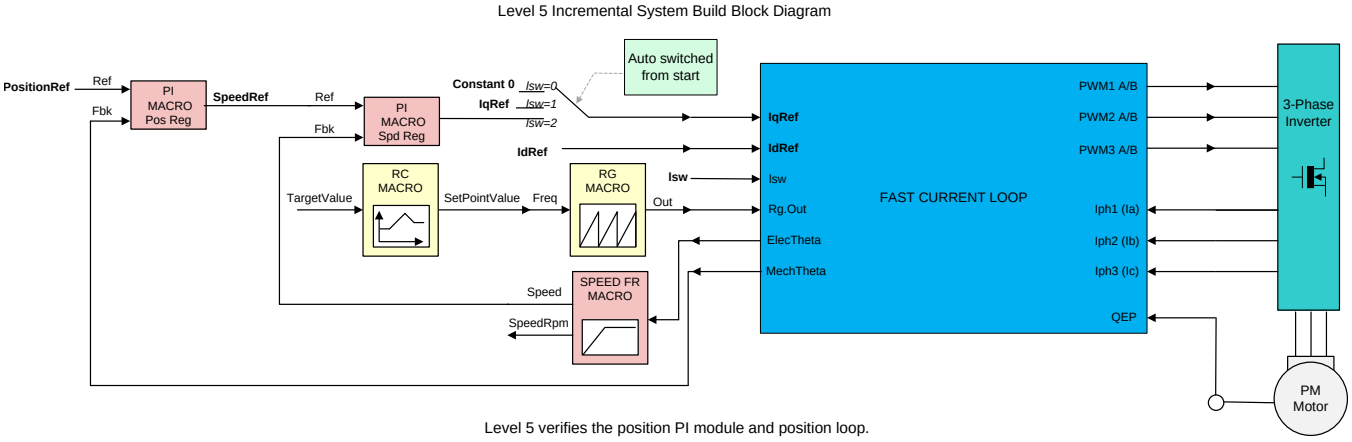


Figure 12. Level 5 Block Diagram Showing Position Loop With Inner FCL

In the scope plot shown in Figure 13, the position reference and position feedback are plotted. It can be seen that they are aligned with negligible lag, which may be attributed to software. If the Kp and Ki gains of the position loop controller are not chosen properly, this may lead to oscillations in the feedback or a lagged response.

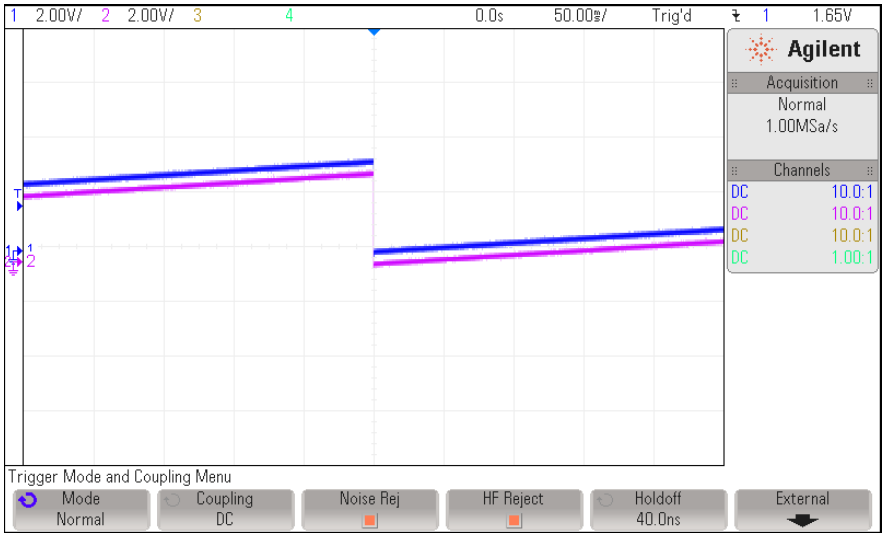


Figure 13. Scope Plot of Reference Position to Servo and Feedback Position

NOTE:

- If the motor response is erratic, then the sense of turn of the motor shaft and the encoder may be opposite. Swap any two phase connections to the motor and repeat the test.
- 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 prestart mechanical position of the rotor. If the mechanical position repeatability or consistency is needed, then the QEP index pulse must be used to set a reference point. This may be taken as an exercise.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY 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 products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources 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 RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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