Application Report
SPRACB8A–March 2018–Revised March 2018

Dual Motor Control Using FCL and Performance Analysis Using SFRA on TMS320F28379D LaunchPad

Ramesh T Ramamoorthy, Sam Sabapathy and Santosh K Athuru

ABSTRACT

The latest C2000™ Delfino™ family of microcontrollers supports fast current loop (FCL) implementation for high bandwidth control of motor drives over a wide speed range in high end multi axes industrial servo control applications. Due to the stringent computational demands of the control algorithm and the demands of interfacing to various position feedback sensors used in these applications, traditionally FPGAs and discrete analog-to-digital converters (ADCs) have been widely used to implement the core control solution. However, recent C2000 MCUs can cost effectively replace FPGAs and external ADCs in these applications and exceed the functional requirements due to superior features. This application report helps to analyze the functional behavior of the servo loops using fast current loop algorithms in terms of bandwidth and phase margin. The results show a three times improvement in current loop bandwidth for a given phase margin. The test bench used in this study consists of a motor-generator set (2MTR-DYNO), an F28379D launch pad (LAUNCHXLF28379D) and TI's low voltage inverter modules based on GaN (BOOSTXL-3PHGANINV) or DRV8305(BOOSTXL-DRV8305EVM).

This application report covers the following:

- Incremental build levels calling modular FCL and SFRA functions
- Experimental results

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

High performance motor drives in servo control applications are expected to provide high precision and high bandwidth control of current, speed and position loops for superior control of end applications such as robotic arm, CNC machines, and so forth. Since the current loop makes up the inner most control loop, it must have a high bandwidth to enable the outer speed or position loops to be faster. Hence, a high bandwidth FCL is needed in high performance industrial servo control applications. However, the delay due to ADC conversion and algorithm execution limit the current controller bandwidth to about a tenth of the sampling frequency.

Until recently, because of the time critical computational demand of the control algorithm and interface demands of various position encoders, FPGAs and external ADCs were needed to implement the fast current loop. However, with the advent of latest C2000 Delfino family of microcontrollers, it is now possible to replace FPGAs and external ADCs with this MCU for a cost effective solution. This paper outlines the implementation of fast current loop on a C2000 platform running two mechanically coupled motors, and verifies the frequency response of the control loops using TI's Software Frequency Response Analyzer (SFRA) software library. Dynamic frequency response analysis in real-time on a motor drive system is unique among MCU suppliers and is currently capable only on C2000 MCUs.

Using the earlier release of FCL library for this device and the Software Frequency Response Analyzer (SFRA) library for C2000 MCUs from TI, the control bandwidth of fast current loop and the operating speed range of motor are experimentally verified. This application report documents the test setup, procedure and the quantitative results obtained. It is important to note that when the PWM carrier frequency is 10 KHz, the current loop bandwidth obtained is 5 KHz for a phase margin of 45 degrees over a wide speed range.

Since the device has two CPU cores, it is possible to control each motor out of one CPU to maximize the performance of each drive. Considering the pin out connections of launch pad and that the scope of this experiment is to study the performance metric of FCL using SFRA, the FCL library is customized to run both motors out of one CPU core for ease of use and analysis. This leads to a drop in modulation index of about 6% in dual sampling mode compared to that in the original library. However, in real applications needing higher modulation index, the code can be split across two CPU cores to achieve a higher modulation index.

1.1 Acronyms and Descriptions

Table 1. Acronyms and Descriptions

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

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

The recent C2000 Delfino family of MCUs possess the desired computation power to execute complex control algorithms along with the right combination of peripherals such as ADC, enhanced pulse width modulator (ePWM), enhanced quadrature encoder pulse (eQEP) and enhanced capture (eCAP) to interface with various components of the digital motor control (DMC) hardware. These peripherals have necessary hooks to provide flexible PWM protection like trip zones for PWMs and comparators.

The key FCL enablers of this MCU are hardware features such as:

- Parallel processing floating point core (CLA) augumenting the main CPU
- Higher CPU and control law accelerator (CLA) clock frequency
- Trigonometric Math Unit (TMU) to support high speed, precision trigonometric and math functions
- Four high speed precision 12b and 16b ADCs
- Configurable Logic Blocks (CLB) - using PM library from TI, a range of absolute encoders can be interfaced

Together, these features provide enough hardware support to increase computational bandwidth per CPU core 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 or LAUNCHXL-F28379D+Inverter booster packs such as BOOSTXL-DRV8305EVM or BOOSTXL-3PhGaNInv) help 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 speed control block diagram of a field oriented control (FOC) based AC motor control system used in servo drives. The current loop is highlighted here because this is the inner most loop and has a higher influence on the bandwidth of the outer speed and position loops. For the outer loop to have a higher bandwidth, the inner loop must have a far higher bandwidth, typically more than 3 times.

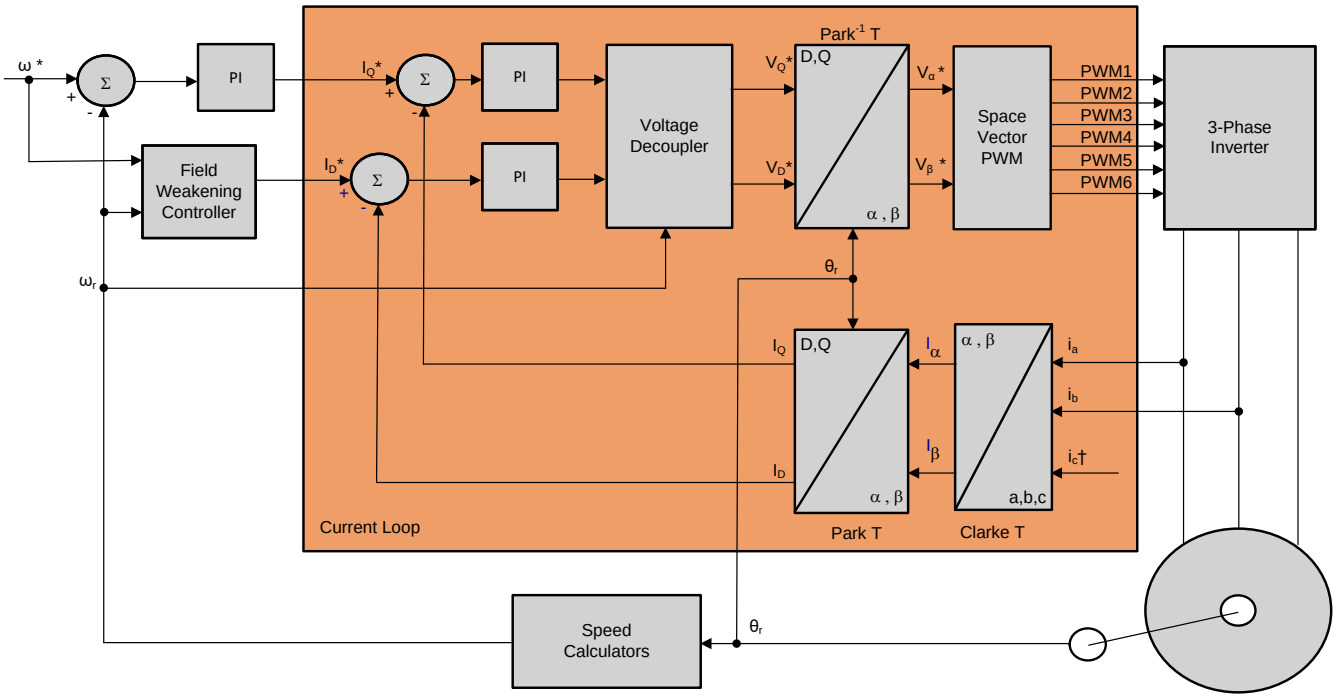


Figure 1. Basic Scheme of FOC for AC Motor

In the current loop, any two of the motor phase currents are measured, while the third can be estimated from these two. These measurements feed the Clarke transformation module. The outputs of this projection are designated Iα and Iβ . These two components of the current along with the rotor flux position are the inputs of the Park transformation, which transform them to currents (Id and Iq) in D-Q rotating reference frame. The Id and Iq components are compared to the references Idref (the flux reference) and Iqref (the torque reference). At this point, the control structure shows an interesting advantage; it can be used to control either synchronous (PM) or asynchronous (ACIM) 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, idref can be set to zero, except during field weakening. Unlike PM motor, ACIM motors do not have a rotor flux by default. Since the flux need to be created, the flux reference current must be greater than zero.

The torque command Iqref can be fed from the output of the speed regulator. The outputs of the current regulators are Vdref and Vqref. These outputs are applied to the inverse Park transformation. Using the position of rotor flux, this projection generates Vαref and Vβref, which are the components of the stator vector voltage in the stationary orthogonal reference frame. These components are the inputs of the PWM generation block. The outputs of this block are the signals that drive the inverter.

Both Park and inverse Park transformations need the rotor flux position. Obtaining this rotor flux position depends on the choice of AC machine observer in sensorless control or the position encoder in the case sensored control.

4 Outline of the Fast Current Loop Library

The major challenge in digital motor control systems is the influence of the sample and hold (S/H) and transportation delay inside the loop that slows down the system impacting its performance at higher frequencies and running speeds. In a time critical algorithm such as the fast current loop, the latency between feedback sampling and PWM update should be as small as possible. A minimal current loop time not only helps to improve the control bandwidth, but it also enables a higher modulation index (M-I) for the inverter. A higher M-I translates into the higher phase voltage that the inverter can apply on the motor. Higher loop latency will reduce the maximum available voltage and can restrict the rate of current change in the motor, thereby, adversely impacting the controller performance.

To overcome these challenges, a controller with high computational power, right set of control peripherals and superior control algorithm are needed. The Delfino MCU, TMS320F2837x, has the much needed architecture and hardware support to enable higher performance, and the Fast Current Loop (FCL) library from TI that runs on this C2000 MCU provides the needed algorithmic support.

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

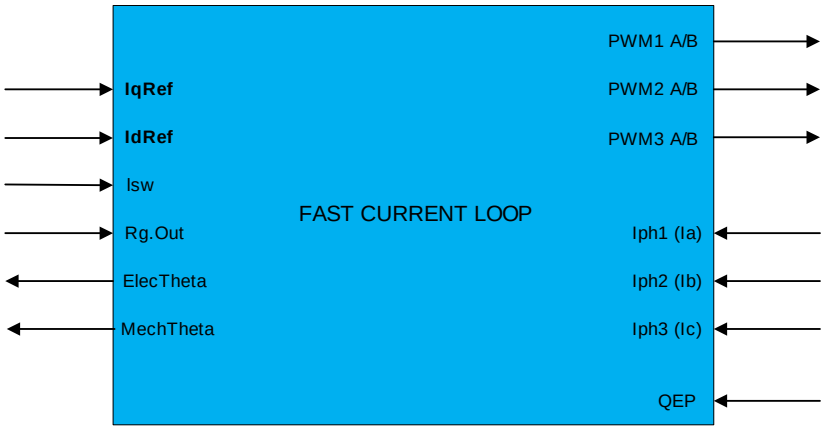


Figure 2. Fast Current Loop (FCL) Library Block Diagram

The original FCL library utilizes the following features in the F2837x MCU

- Trigonometric and Math Unit (TMU)
- 4 high speed 12/16 bit ADCs
- Multiple parallel processing blocks such as Control Law Architecture (CLA)

The block diagram of FCL library with its inputs and outputs is shown in Figure 2. The original FCL library, *Fast Current Loop Library* , partitions the algorithm across single set of CPU, CLA and TMU to bring down the latency to under 1 micro second as against the acceptable 2 micro second. Further optimization is possible if the algorithm is written in assembly. As mentioned earlier, considering the scope of this evaluation, the coding process is simplified for better customer appreciation. Therefore, CLA is not used here as well as the second CPU. The modified FCL library, *Fast Current Loop (C28x) Library* , supports a complex PI controller.

The list of FCL API functions and their description are given in Table 2.

Table 2. Summary of FCL Interface Functions

API Function	Description
Uint32 FCL_GetSwVersion(void);	Returns a 32-bit constant and for this version the value returned is 0x00000004
void FCL_initPWM(MOTOR_VARS * m);	Initializes all motor control PWMs for FCL operation, this function is called by the user application during initialization process.
void FCL_PI_Ctrl(MOTOR_VARS * motor);	Function that performs the PI Control as part of the Fast Current Loop
void FCL_PI_CtrlWrap(MOTOR_VARS * motor);	Wrap up function to be called by the user application at the completion of FCL in PI Control Mode
void FCL_CC_Ctrl(MOTOR_VARS * motor);	Function that performs the Complex control as part of the Fast Current Loop
void FCL_CC_CtrlWrap (MOTOR_VARS * motor);	Wrap up function to be called by the user application at the completion of FCL in Complex Control Mode
void QepPosEstModule(MOTOR_VARS * motor);	Function to calculate the rotor position from electrical zero of the motor phase A, using QEP data
void FCL_QEP_wrap(MOTOR_VARS * motor);	Function to wrap up the QEP application at the completion of Fast Current Loop routine

5 SFRA Library

Texas Instruments' Software Frequency Response Analyzer (SFRA) library is designed to enable frequency response analysis on any digitally controlled closed loop system using software only. This enables performing frequency response analysis of the closed loop system with relative ease as no external connections or equipment is required. The optimized library can be used to identify the plant and the open loop characteristics of a closed loop system. In this study, it can be used to get stability information such as the gain margin, phase margin and bandwidth to evaluate the control loop performance.

Consider a digitally controlled closed loop power converter, as shown in Figure 3, where:

- H is the transfer function of the plant that needs to be controlled,
- G is the digital compensator,
- GH is referred to as the open loop transfer function
- r is the instantaneous set point or the reference of the converter,
- Ref is the DC set point reference,
- y the ADC feedback,
- e the instantaneous error,
- d the sensor noise/disturbance,
- u the PWM duty cycle.

The key objectives of the compensator in a closed loop system can be summarized as:

- Ensure system is stable, that is, system tracks the reference asymptotically:

$$e = \lim_{t\to\infty} e(t) = \lim_{t\to\infty} \frac{r}{(1+GH)} \to 0 \tag{1}$$

- System provides disturbance rejection to guarantee robust operation:

$$S = \frac{y}{d} = \frac{1}{1+GH} \to 0 \tag{2}$$

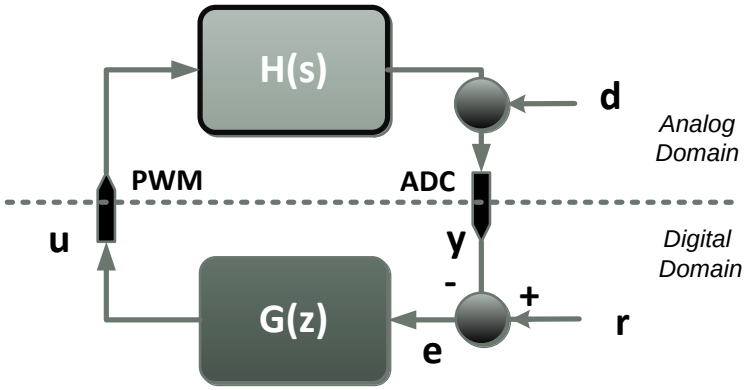


Figure 3. Digitally Controlled Control System

It is clear from Equation 1 and Equation 2 that by knowing the open loop transfer function (GH), one can determine if the system meets the objectives. A Bode plot of the open loop transfer function GH is frequently used for this purpose and quantities such as gain margin(GM), phase margin(PM) and bandwidth (BW) are often used to comment on the stability and robustness of a closed loop system.

SFRA library can enable measurement of the GH and H frequency response by software. This data can be used to:

- Verify the plant model (H) or extract the plant model (H)
- Design a compensator (G) for the closed loop plant
- Verify the close loop performance of the system by plotting the open loop (GH) Bode diagram

As the frequency response of GH and H carry information of the plant, the data can be used to comment on the health of the power stage or control loop by periodically measuring the frequency response.

This library is used to study the current loop performance in motor drive system.

6 Evaluation Setup

In order to study the current loop bandwidth, the back emf component of the loop needs proper decoupling or compensation. At zero speed when there is no back emf, the loop performance can be analyzed using frequency response analysis (FRA) methods. This can be used as a reference to verify the loop performance at different speeds to see if there is any change in the controller behavior. To do this effectively, a motor generator set that can hold zero speed will be helpful. The software is built such that two different motors can be controlled independently and then coupled together as motor-generator for performing frequency response analysis.

The evaluation setup is built using hardware that is readily available from TI. It consists of an F28379D CPU based launch pad, inverter booster pack based off either DRV8305 or GaN+INA240, and a motor-dyno set for load testing the drive motor, their details are given in the Section 6.1. This evaluation set up will use GaN+INA240 as the power stage to make use of line current sensing facility available in this hardware.

Example Project Features

- Sensored FOC of PMSM motor
- Fast current loop library
- Speed and Torque control loops
- Position Sensor Support: Incremental Encoder (QEP)
- Current Sensing: Analog feedback using ADC

This example projects are derived out of two existing control suite projects. They are as follows:

- controlSUITE\libs\app_libs\motor_control\libs\FCL\v02_00_00_00\Examples\
- controlSUITE\development_kits\TIDM-SERVO-LAUNCHXS\ MonoMtrServo_377s_v1_00_00_00\

6.1 Hardware

The details of the evaluation hardware, most of them available from TI eStore, and references to the user's guide are listed below:

- CPU - LAUNCHXL-F28379D - one unit - *LAUNCHXL-F28379D Overview User's Guide*
- Inverter (INV) - Any of the following
  - BOOSTXL-DRV8305EVM - two units - *BOOSTXL-DRV8305 Hardware User's Guide*
  - BOOSTXL-3PhGaNInv - two units - *BOOSTXL-3PhGaNInv Evaluation Module User's Guide*
- Motor Dyno Set - 2MTR-DYNO - one unit (two motors)
- A variable DC power supply rated at 48V/5A

6.1.1 LaunchPad

For immediate reference, the layout of LAUNCHXL-F28379D is given here. For further details, see the *LAUNCHXL-F28379D Overview User's Guide* .

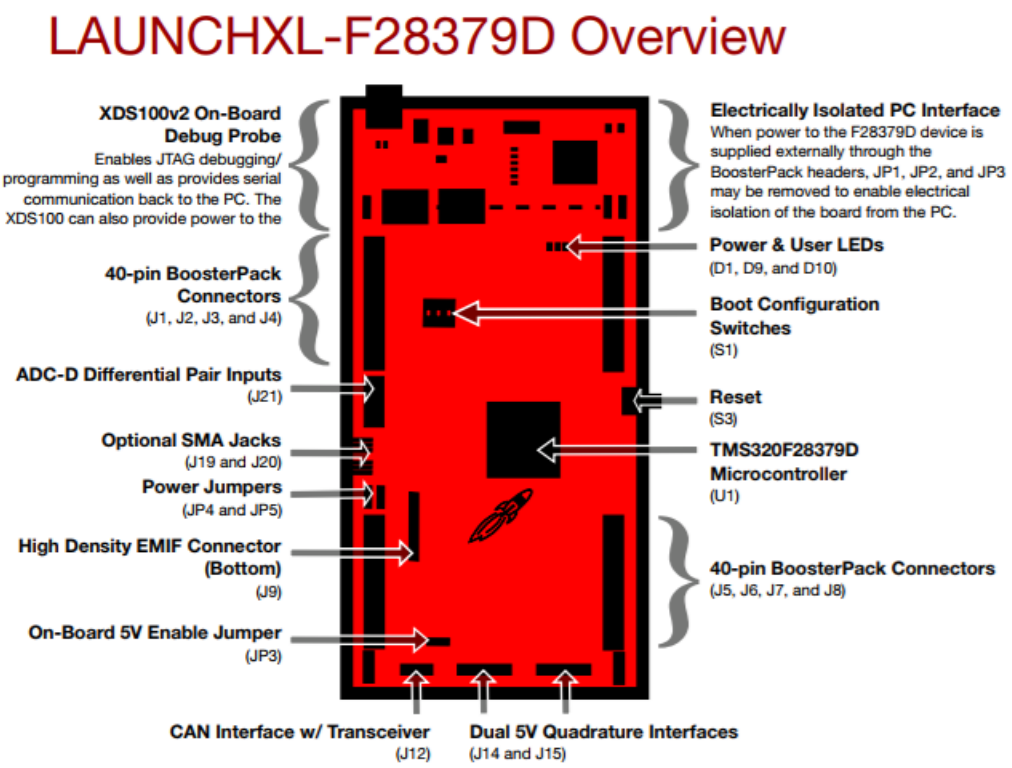


Figure 4. LAUNCHXL-F28379D Layout Diagram

It can support driving two motors with QEP position feedback. In addition, it has a couple of SPI ports available that can be used to drive SPI configurable inverters.

For early code development without the BoosterPacks, it is ok to retain the jumpers JP1, JP2 and JP3 of the launch pad that feeds 5 V and 3.3 V power from the emulator section to the CPU. Alternately, the CPU can be powered from an external source through the header pins of header J1, J3 or J5 and J7, in which case remove jumpers JP1, JP2 and JP3. Make sure to remove these jumpers or external power supply before mounting the inverter BoosterPack because they are designed to power the launch pad when mounted. Jumpers JP4 and JP5 on the launch pad 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, then JP4 and JP5 are don't care.

CAUTION

If jumpers JP1-JP3 are present when the BoosterPack is powered, then it can:

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

6.1.1.1 PWMDACs

Launch pad has four PWM-DACs, referenced in this document as PWMDAC1-4. They are available on jumper pins J4-31 and J4-32, and J8-71 and J8-72. PWM-DACs are basically low pass filtered signals that are originally generated as high frequency PWM carrier signals modulating the signal of interest. In the evaluation project, these PWMDACs are used to display intermediate system variables for debug purposes.

6.1.1.2 DACs

Launch pad also has a couple of DACs available on jumper pins J3-30 and J7-70. Depending on the booster pack in use, they may or may not be available as DAC. With DRV8305EVM, these DACs could be availed, whereas with the GaN board, it is not.

6.1.2 Inverter Booster Pack - DRV8305

For immediate reference, the layout and pinout diagram of BOOSTXL-DRV8305EVM is shown here in Figure 5. For more details, see the *BOOSTXL-DRV8305 Hardware User's Guide* .

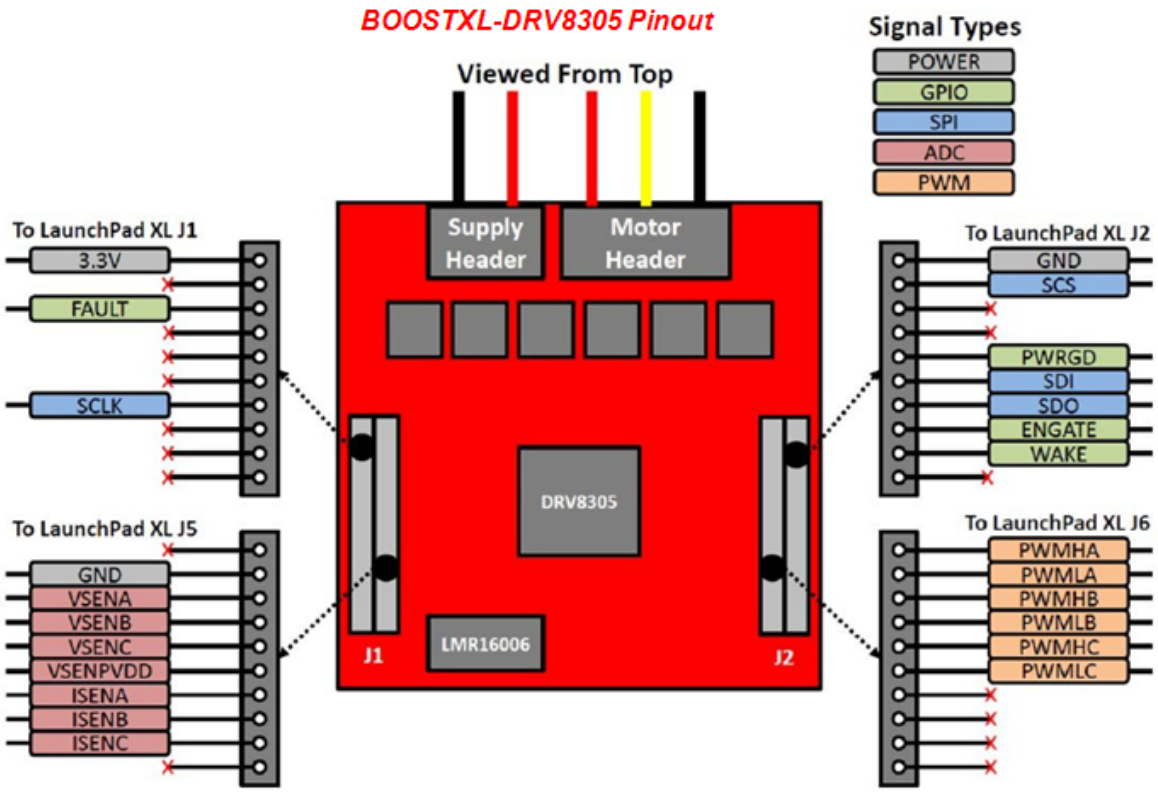


Figure 5. DRV Board Layout Diagram

The assembly with the DRV8305 inverter is as shown in Figure 6.

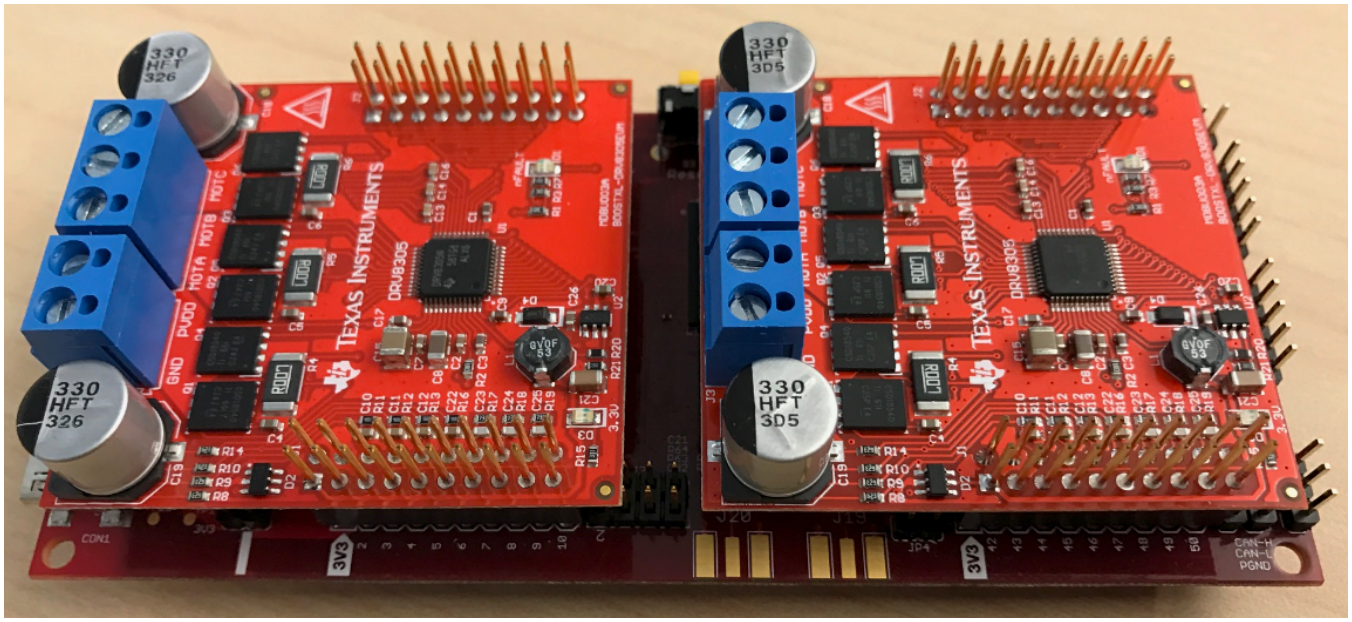


Figure 6. DRV8305 Assembly Diagram

It is important to note that in the BOOSTXL-DRV8305EVM inverter, the current shunt resistor is installed below the bottom switch of inverter half bridges and so current sense is possible only when the bottom switch is ON. Therefore, only single feedback sample and single PWM update per carrier cycle is alone possible. This can reduce the maximum achievable bandwidth by a half. If a higher bandwidth per PWM frequency is needed, then an alternate inverter booster pack that provides line current feedback is needed.

6.1.3 Inverter Booster Pack - GaN + INA240

For immediate reference, the layout and pinout diagram of BOOSTXL-3PhGaNInv is shown in Figure 7. For more details, see the *BOOSTXL-3PhGaNInv Evaluation Module User's Guide* .

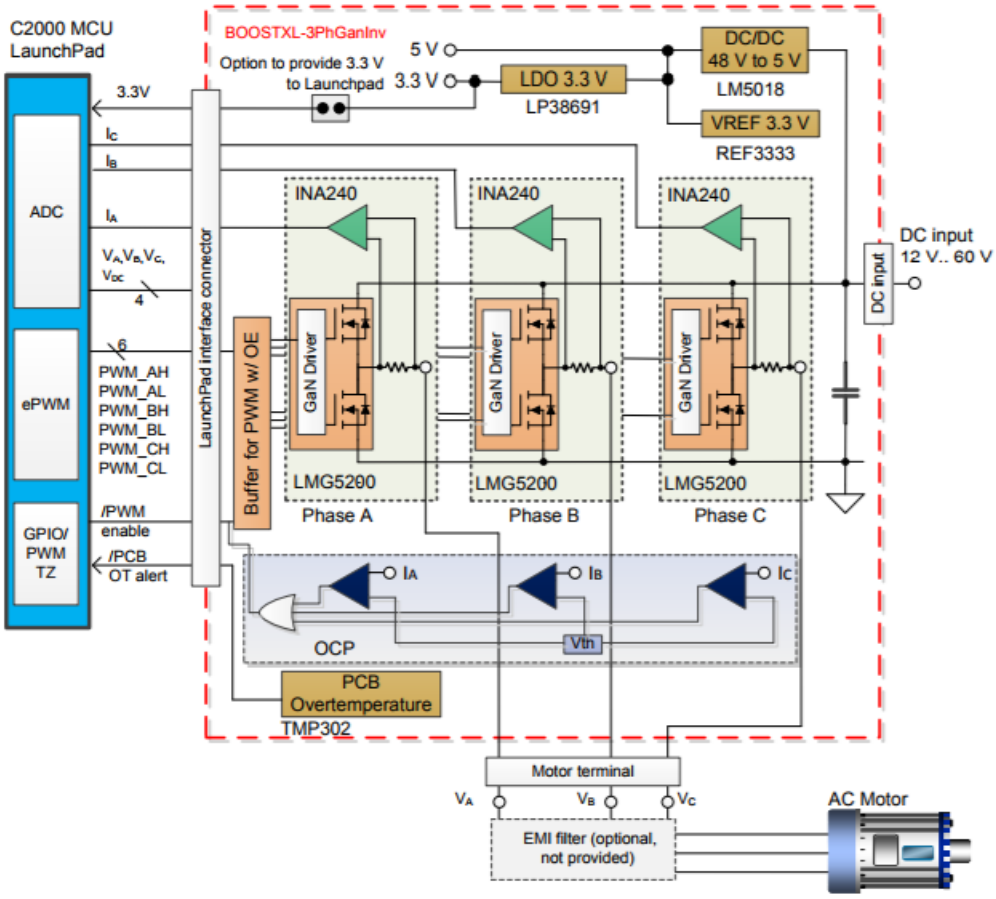


Figure 7. BOOSTXL-3PhGaNInv Functional Block Diagram

A dual booster pack assembly with LAUNCHXL-F28379D is shown in Figure 8.

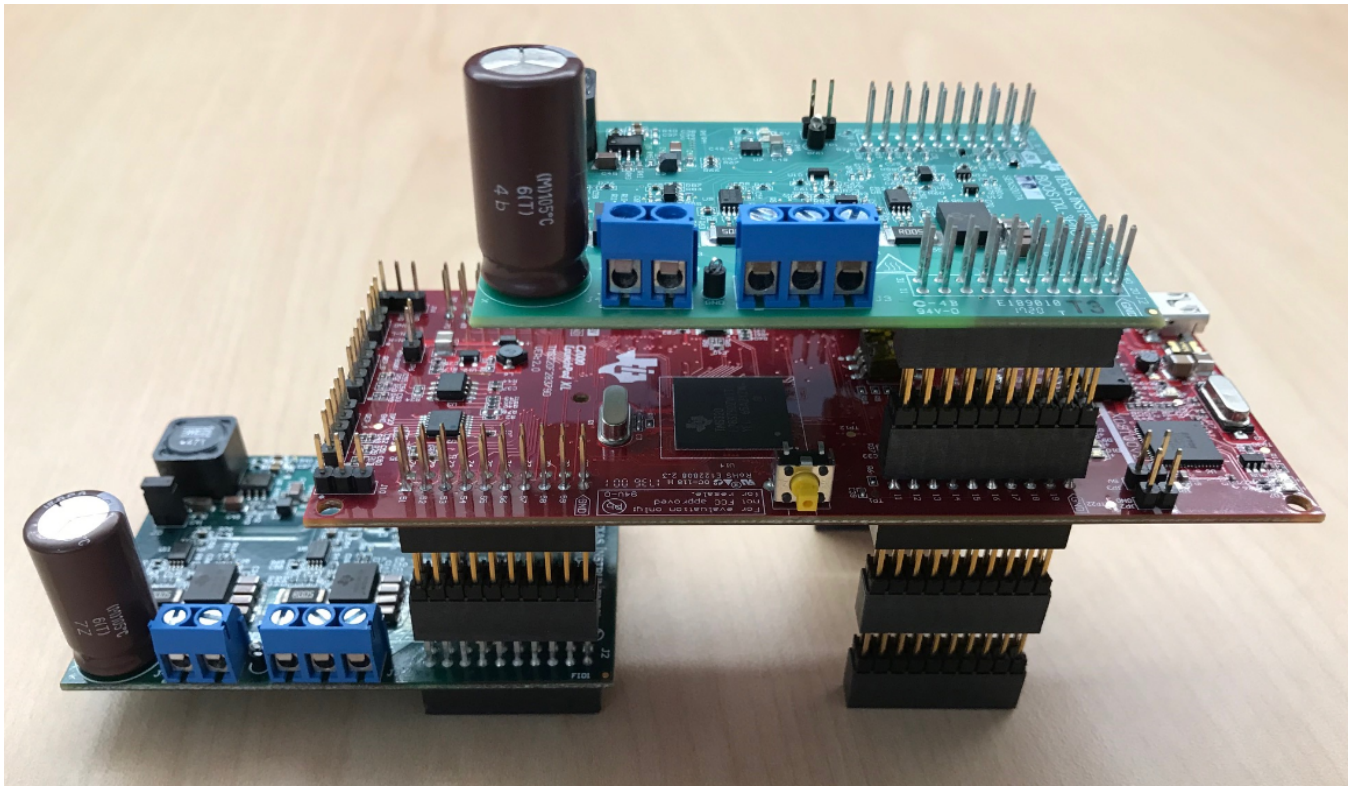


Figure 8. Dual Motor Control Assembly With LAUNCHXL-F28379D and BOOSTXL-3PhGaNInv

The booster pack is a GaN based inverter with INA240 based series current sensor that senses the inverter output phase current at all times. This enables double sampling of motor currents (corresponding to the peak and zero of PWM carrier), thereby, doubling the control bandwidth compared to that with single sampling. The size of the booster pack is large enough that it restrains fitting two of them on the same launch pad. Due to this mechanical constraint, a work around may be needed. One of the mounting methods is shown in Figure 8 that uses 2x10 headers for not only extending the signals out but also for providing some spatial clearance between various PCBs giving better clarity. Depending on user convenience, a passive extender board can be designed to bring out a pair of launch pad headers to mount the second booster pack or a short flat 20 pin ribbon cable can be used.

6.1.4 Two Motor Dyno

The two motor dyno setups help to perform load test on the drive motor by mechanically coupling it to the other motor that acts as a generator. The kit comes with a coupler, mounting screws and key. Assemble the motor-dyno set as shown in Figure 9.

Figure 9. Two Motor Dyno Set

6.1.5 System Hardware Connections

There are various jumpers JP1-6 present on the launch pad, and their purpose are given in the launch pad *LAUNCHXL-F28379D Overview User's Guide* . Before mounting the booster packs, ensure that jumpers JP1 - JP5 in the launch pad are not populated and that JP6 is populated.

The motor that comes with 2MTR-DYNO kit is a PMSM motor with both QEP and HALL sensors available on its headers J4 and J10, respectively. The control scheme is based on QEP feedback; therefore, its QEP header J4 is fed into the LAUNCHXL. Do not use the HALL sensor header J10.

The booster packs suggested for this evaluation will mount directly on to the launch pad LAUNCHXL-F28379D. This connects the analog/digital IOs of the booster pack to the appropriate IOs of the CPU. Make sure to match the orientation of inverter booster packs as shown in Figure 6 or Figure 8 before mounting. Mount one inverter booster pack on launch pad connectors J1-J4, let us call it inverter INV1. Likewise, mount the other inverter on launchpad connectors J5-J8, and call it inverter INV2.

Until instructed, leave the INV output headers and QEP headers open. When instructed to connect motor 1, connect motor 1 terminal A, B and C to INV1 connector terminal Va, Vb and Vc and motor's QEP header to QEP-A on launch pad. Likewise, when instructed, connect motor 2 to INV2 and its QEP header to QEP-B on launch pad.

6.1.6 Powering Up the Setup

The following are important points to keep in mind while powering the setup.

1. Make sure that jumpers JP1-JP5 on the launch pad are removed.
2. With DRV8305 EVM, the max voltage is limited to 40V.
3. With Gan+INA240 EVM, the max voltage is limited to 48V.
4. The current limit on power supply can be set at 2A. Depending on need, the user can increase the current limit up to 5A.
5. With INV1 mounted on the launch pad, and when the permitted max dc voltage (40V or 48V) is fed into INV1 through its connectors, it will generate 3.3 V for the launch pad and you will notice LEDs blinking on the launch pad.
6. To make INV2 to source 3.3 V for launch pad instead, populate JP4 on launch pad, but make sure that 3.3 V from INV1 is disconnected to avoid circulating currents between the 3.3 V power supplies of the two INVs.
7. The 3.3V supply from the INV can be disconnected to the launch pad when using GaN+INA240 booster pack by removing jumper J5 from the booster pack, but this feature is not available in DRV8305 booster pack.
8. If both INV1 and INV2 are DRV8305 booster packs, then INV1 will power the launch pad and jumper JP4 on launchpad should be removed.

CAUTION

If jumpers JP1-JP5 are present when the BoosterPack is powered, then it can:

- Create circulating current between the 3.3V power supplies generated by the inverter BoosterPacks and the emulator powered by USB
- If a development computer is connected through USB, then it can potentially damage the development computer as it will tie the computer's USB to external power supply GND.
- It is recommended to use isolated power supply to power the booster pack.

6.2 Software

The software is developed using FCL and SFRA libraries released already in ControlSUITE. FCL is used to improve the current loop bandwidth and SFRA is used to do frequency response analysis of any control loop. As mentioned earlier, in this release, the FCL library is customized and executed out of CPU1 to control two motors. Thisrelease can be ported to use dual CPU cores and their CLAs to speed up the computation, which will facilitate a higher DC bus utilization by the inverter and increase the speed range.

The project can be found at
controlSUITE\libs\app_libs\motor_control\libs\FCL_SFRA\v01_00_00_00\Examples

The customized FCL software library can be found at
controlSUITE\libs\app_libs\motor_control\libs\FCL_SFRA\v01_00_00_00\lib

The SFRA software library can be found at ti\controlSUITE\libs\app_libs\SFRA\v1_10_00_00\

The software is built such that two different motors can be controlled independently and then coupled together as motor-generator set for performing frequency response analysis.

6.2.1 Incremental Build

The system is incrementally built up like any other projects in ControlSUITE. In each build level, a certain operation of the system, could be hardware or software, is verified and integrated incrementally. In the final build level, all operations are integrated to make a complete system. Software modules are written as either C macros or C callable functions. Table 3 summarizes the functional testing at each incremental system build.

Table 3. Testing Modules in Each Incremental System Build

Build Level	Functional Integration	Library
Level 1	PWM Generation	
Level 2	Run motor Open Loop - feedback verification and calibration	
Level 3	Run motor current loops using FCL library	FCL
Level 4	Run motor speed loop with the inner current loop using FCL library	FCL
Level 5	Run both motors – one as motor and another as generator	FCL
Level 6	Use SFRA GUI to run SFRA on target CPU – CPU using SFRA library	SFRA

Build Levels 1 through 4 can be done with the motors shafts decoupled or separate. If they are separated, then both motors can be tested independently in each build level, otherwise only one motor can be tested at any given time and the terminals of the other motor should be safely disconnected from the INV and insulated so that when the shaft spins, its back emf would not lead to any hazards. Levels 5 and 6 need the motor shafts to be coupled to perform load tests as well as frequency response analysis.

6.2.2 Software Development Environment

If you are not familiar with TI's software Development Environment, see the *Software Setup for LaunchXS Projects* section on page 21 of the control suite project located at:

controlSUITE\development_kits\TIDM-SERVO-LAUNCHXS\MonoMtrServo_377s_v1_00_00_00\~Docs

It gives a detailed step-by-step guidance into the development environment while introducing the project as well. It is one of the projects that this current project is based off. Therefore, it is easy to correlate variables and concepts being discussed. Use the current project while working through the guide to understand the initial setup. CPU1 is used in this project.

7 System Software Integration and Testing

This section deals with various incremental build levels starting from first level up to the final level where the system is fully integrated. After full integration, SFRA GUI is called into initiate frequency response analysis in the selected loop and the Bode plot is displayed by the GUI. Various trials can be performed at this point tweaking various parameters for performance tuning.

7.1 Integrating Inverter (INV) Selection in Software

The user can select between DRV8305EVM and BOOSTXL-3PHGANINV as inverter for running the motor. These inverters can have different feedback gain settings. Hence it is important for the user to make appropriate selections in software to integrate the choice of inverter booster pack used in the system. It can be done by following the steps provided below:

1. Open \app_headers\FCL_SFRA_XL_DualServo-Settings.h from the project directory.
2. For inverter selection of each motor, choose between DRV8305 or GaN_BP. For example

```
#define MOTOR1_DRV DRV8305 // select between DRV8305 and GaN_BP
                or
#define MOTOR1_DRV GaN_BP // Select between DRV8305 and GaN_BP
```

By default, GaN_BP is selected as inverter for both motors.

7.2 Incremental Build Level 1

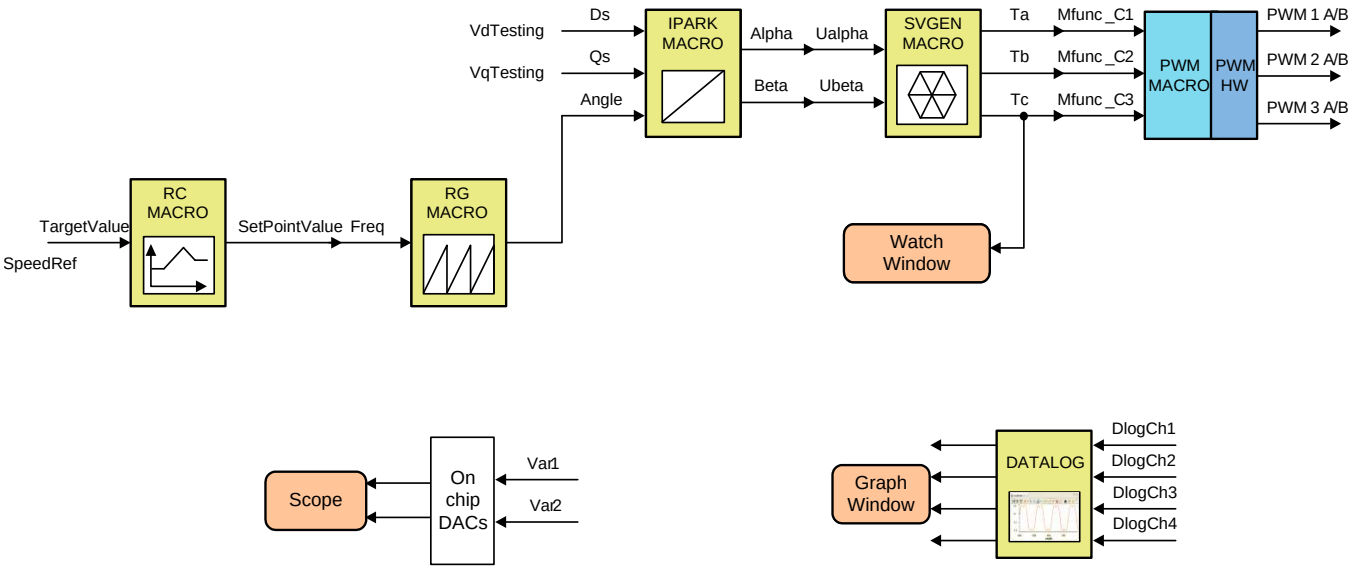
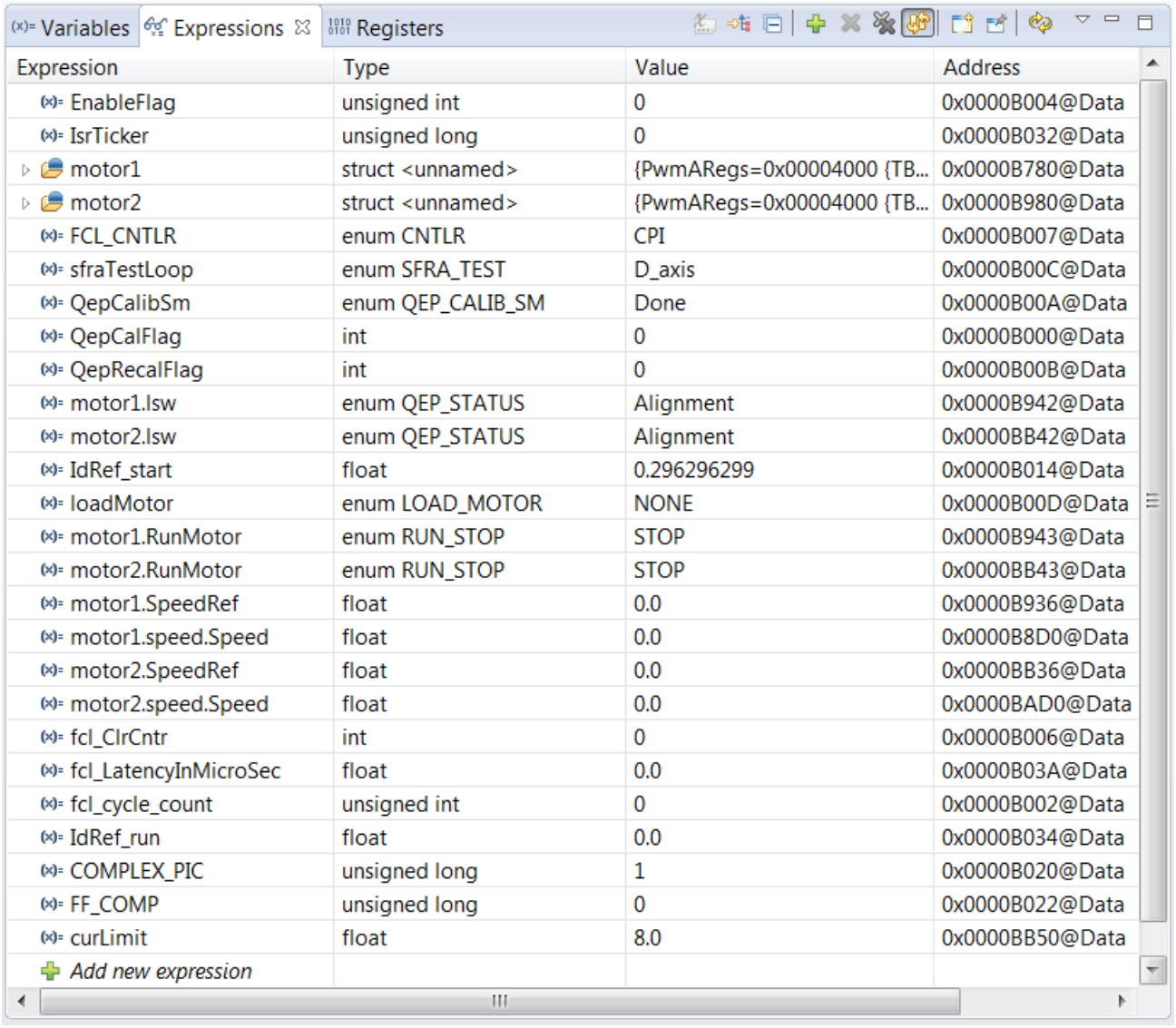


Figure 10. Level 1 Block Diagram

The block diagram of the system built in BUILD LEVEL 1 is shown in Figure 10. 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 and the peripheral dependent PWM_MACRO (PWM initializations and update) modules.

1. Open \app_headers\FCL_SFRA_XL_DualServo-Settings.h from the project directory. Select level 1 incremental build option by setting the BUILDLEVEL to FCL_LEVEL1 (#define BUILDLEVEL FCL_LEVEL1).
2. Right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run. If not already done, add variables to the expressions window by ‘Right Clicking’ within the Expressions Window and ‘Importing’ the file ‘variables_FCL_SFRA_DualMtr.txt’ from root directory and the expressions window will look as shown in Figure 11.



Expression	Type	Value	Address
EnableFlag	unsigned int	0	0x0000B004@Data
IsrTicker	unsigned long	0	0x0000B032@Data
motor1	struct <unnamed>	{PwmARegs=0x00004000 {TB...	0x0000B780@Data
motor2	struct <unnamed>	{PwmARegs=0x00004000 {TB...	0x0000B980@Data
FCL_CNTLR	enum CNTLR	CPI	0x0000B007@Data
sfraTestLoop	enum SFRA_TEST	D_axis	0x0000B00C@Data
QepCalibSm	enum QEP_CALIB_SM	Done	0x0000B00A@Data
QepCalFlag	int	0	0x0000B000@Data
QepRecalFlag	int	0	0x0000B00B@Data
motor1.lsw	enum QEP_STATUS	Alignment	0x0000B942@Data
motor2.lsw	enum QEP_STATUS	Alignment	0x0000BB42@Data
IdRef_start	float	0.296296299	0x0000B014@Data
loadMotor	enum LOAD_MOTOR	NONE	0x0000B00D@Data
motor1.RunMotor	enum RUN_STOP	STOP	0x0000B943@Data
motor2.RunMotor	enum RUN_STOP	STOP	0x0000BB43@Data
motor1.SpeedRef	float	0.0	0x0000B936@Data
motor1.speed.Speed	float	0.0	0x0000B8D0@Data
motor2.SpeedRef	float	0.0	0x0000BB36@Data
motor2.speed.Speed	float	0.0	0x0000BAD0@Data
fcl_ClrCntr	int	0	0x0000B006@Data
fcl_LatencyInMicroSec	float	0.0	0x0000B03A@Data
fcl_cycle_count	unsigned int	0	0x0000B002@Data
IdRef_run	float	0.0	0x0000B034@Data
COMPLEX_PIC	unsigned long	1	0x0000B020@Data
FF_COMP	unsigned long	0	0x0000B022@Data
curLimit	float	8.0	0x0000BB50@Data
Add new expression			

Figure 11. Expressions Window for Build Level 1

3. Set “EnableFlag” to 1 in the expressions window. The variable named “IsrTicker” will incrementally increase as seen in expressions window confirming that the interrupt is working properly.

In the software, the key variables within the structure of ‘motor1’ to be adjusted are summarized below:

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

The tests suggested below for ‘motor1’ may be repeated for ‘motor2’ also to verify the hardware and software integrity.

7.2.1 Level 1A (SVGEN_MACRO Test)

The SpeedRef value is specified to the RG_MACRO module via RC_MACRO module. The IPARK_MACRO module is generating the outputs to the SVGEN_MACRO module. Three outputs from SVGEN_MACRO module are monitored via the graph window as shown in Figure 12, or using PWMDAC1-4, the code are already included in the build level. From the debug environment, DualTime Graphs can be plotted from Tools menu and using the Import feature to import 'Graph1.graphProp' and 'graph2.graphProp' files. The output waveforms Ta, Tb, and Tc are 120° apart from each other. Specifically, Tb lags Ta by 120° and Tc leads Ta by 120°. Check the PWM test points on the board to observe PWM pulses (PWM-1H to 3H and PWM-1L to 3L) and make sure that the PWM module is running properly. 'SpeedRef' can be changed to 0.05 to get full cycle view of the inverter fundamental output waveform.

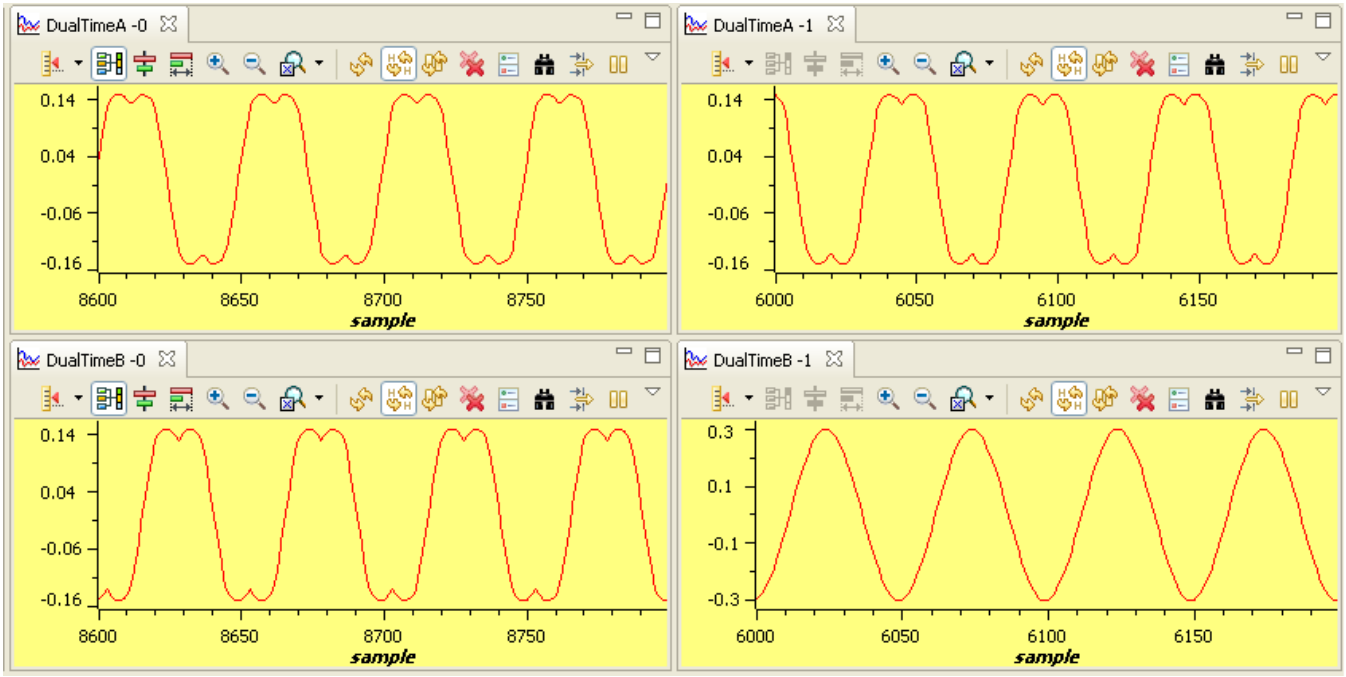


Figure 12. Output of SVGEN, Ta, Tb, Tc and Tb-Tc Waveforms

7.2.2 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, Vb and Vc feedback signals at J1(5,7,9) on DRV8305EVM or J3(24,25,26) on GaN+INA240 EVM 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, take the controller out of real time mode (disable), reset the processor. Note that after each test, this step needs to be repeated for safety purposes, otherwise an improper shutdown might halt the PWMs at certain states where the inverter can draw currents high enough to damage itself. Hence caution needs to be taken while doing these experiments.

7.3 Incremental Build Level 2

The control block diagram of the system built in BUILDLEVEL 2 for each motor is shown in Figure 13. Assuming build 1 is completed successfully, this section verifies the overcurrent protection limits of the inverter and QEP interface running out of library. In this build, the motor is run in open loop.

7.3.1 Connecting motor to INV

Testing both motors at the same time with a coupled shaft, as shown in Figure 9, leads to chaotic behavior. If the assembly is already made, then it is better to test only one motor at a given time. Turn off DC bus voltage before connecting/ disconnecting motor 1 or 2 to INV1 or INV2. To test motor 1, connect motor 1 power terminals to INV1 while keeping motor 2 terminals disconnected and safely insulated from each other. This is to avoid any short circuit between motor 2 phases whenever the shaft spins. Once motor 1 / INV1 test is complete, disconnect motor 1 from INV1 and keep its terminals safely insulated from each other. Then connect motor 2 terminals to INV2 for motor 2/ INV2 tests.

If the motor shafts are disengaged, then both motors can be connected to INV1 and INV2 and tested simultaneously.

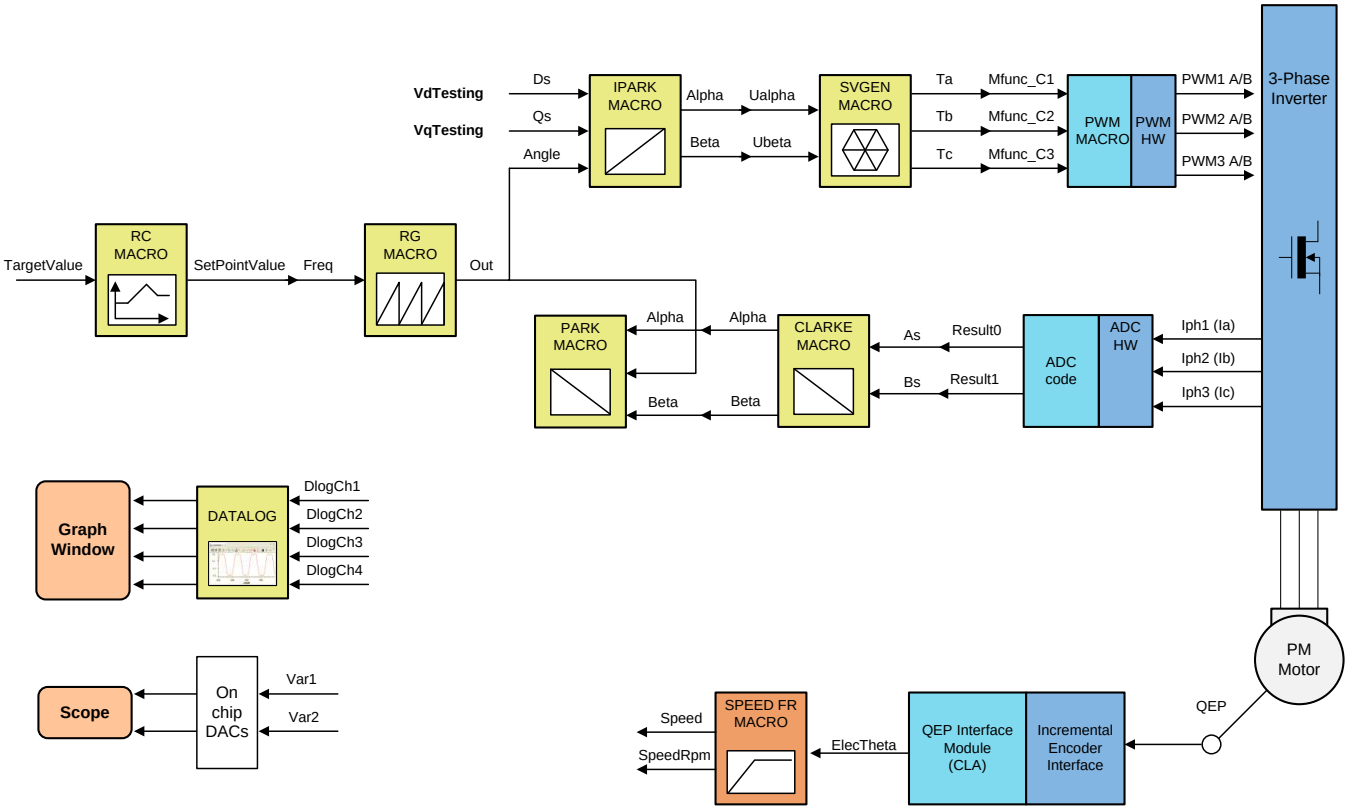


Figure 13. Level 2 Block Diagram

7.3.2 Testing the Motors and INVs

The motor and INV can be tested by following the steps given below, as the PWM signals are already successfully proven through level 1 incremental build. Connect motor 1 to INV1 and / or motor 2 to INV2 as discussed earlier. If possible, the two motors shafts can be disconnected for better clarity or else only one motor can be tested at a given time.

1. Open '\app_headers\FCL_SFRA_XL_DualServo-Settings.h' and select level 2 incremental build option by setting the BUILDLEVEL to LEVEL2 (#define BUILDLEVEL FCL_LEVEL2). Now Right Click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run.
2. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in Expression window to confirm the interrupt working properly. Within the 'motor1' structure, set the variable named "RunMotor" to 'RUN' and the motor will start spinning after a few seconds.

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

- 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, variables VqTesting and SpeedRef must be adjusted carefully so that the motor runs smoothly without stalling or vibrating.

7.3.3 Setting Overcurrent Limit in Software

Overcurrent monitoring is provided using 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 limits. 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, you 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 you can adjust the above settings and retry running the inverter by setting the flag 'clearTripFlagDMC' to 1. This clears TripFlagDMC and restarts the PWMs for the inverter.

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

7.3.4 Setting Current Regulator Limits

The outputs of the current regulators control the voltages applied on the d-axis and q-axis of the motor. The vector sum of the d and q outputs must be less than 1.0, which refers to the maximum duty cycle for PWM generation algorithm. In this particular application, the maximum allowed duty cycle is lesser than 1.0 due to the impact of adc conversion time and pwm computation time. Higher computational speeds allow higher duty cycle operation and better use of the DC bus voltage.

The current regulator output is represented by the variable cntlr_id.Out and cntlr_iq.Out. The regulator limits are set by cntlr_id.Umax/min and cntlr_iq.Umax/min.

7.3.5 Position Encoder Feedback

During all the above tests, the QEP interface was continuously estimating position information. 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) signals, when plotted using PWMDAC utility on PWMDAC1 and PWMDAC2, should look as shown in Figure 14.

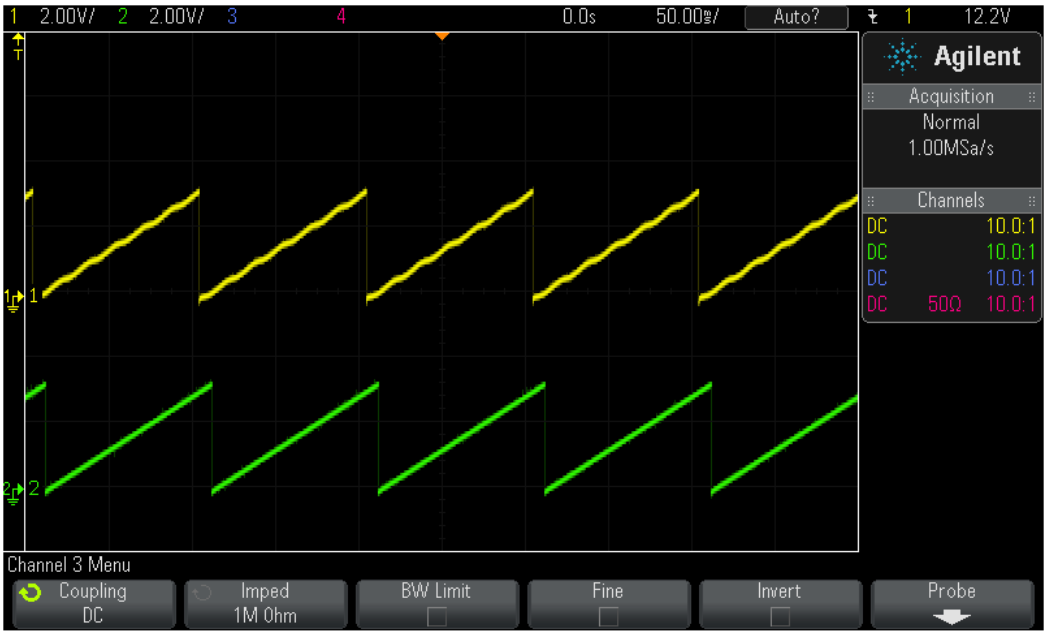
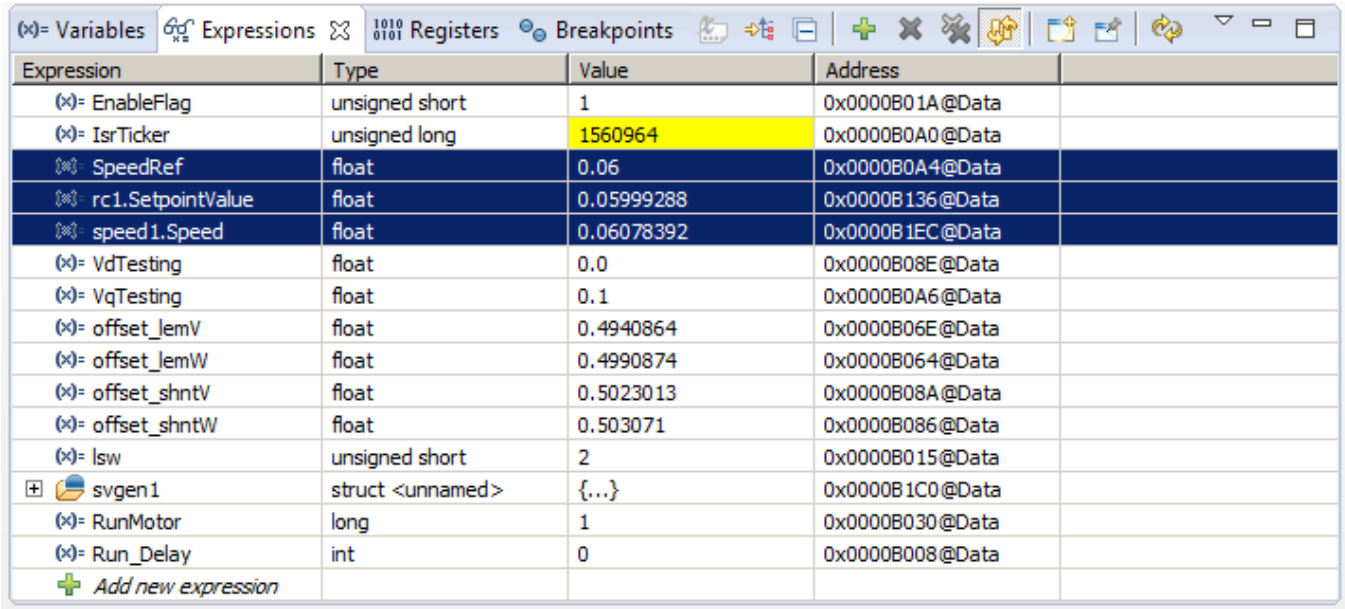


Figure 14. Scope Plot of Reference Angle and Rotor Position

If both motors are controlled at the same time, then motor2 variables will only be displayed by the PWMDACs as it is the last called module. To verify individually, call build level 2 function only for the motor to be tested and comment out the function call to the other motor.

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, it is important to make sure that the sense of change of estimated angle should be same as that of the reference; otherwise, it indicates that the motor has a reverse sense of rotation. This can be fixed by swapping any two wires connecting the motor.

To make sure that the SPEED_MACRO works fine, change the 'SpeedRef' variable in Expressions Window as shown in Figure 15 and check if the estimated speed variable 'speed1.Speed' follows the commanded speed. Since the motor is a PM motor, where there is no slip, the running speed 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_lemV	float	0.4940864	0x0000B06E@Data
offset_lemW	float	0.4990874	0x0000B064@Data
offset_shntV	float	0.5023013	0x0000B08A@Data
offset_shntW	float	0.503071	0x0000B086@Data
Isw	unsigned short	2	0x0000B015@Data
svgen1	struct <unnamed>	{...}	0x0000B1C0@Data
RunMotor	long	1	0x0000B030@Data
Run_Delay	int	0	0x0000B008@Data
Add new expression			

Figure 15. Expressions Window

Set 'RunMotor' to 'STOP' to stop the motor.

The same set of tests can be done on motor 2 by working with structure variable 'motor2'. Once done, take the controller out of real time mode and reset.

7.4 Incremental Build Level 3

This section verifies the dq-axis current regulation performed by the fast current loop. The loop bandwidth can be set in the debug window. The implementation block diagram is given in Figure 16. The two motors shafts can be kept disconnected for more clarity. The bandwidth of the controllers can be set in the debug window.

WARNING

In this build, control is done based on the actual rotor position and hence the motor can run at higher speeds if the commanded 'IqRef' is higher and if there is no load on the motor. Therefore, it is advised to either add some mechanical load on the motor before the test or to apply lower values of 'IqRef'.

At start up, the motors are taken through QEP calibration phase where each motor takes turns to calibrate its QEP's angular offset with respect to its stator angle zero. To start with, motor 1 goes through an alignment and then spins slowly based on an enforced angle waiting on index pulse for calibration. After the index pulse is received, it goes to float mode. Then motor 2 does the same sequence for calibrating its QEP.

After calibration is done, when any of the motor is commanded to run, it runs in full self-control mode based on its own angular position. This calibration step happens for this build level and beyond. Since the calibration routine calibrates both motors, connect both motors but the shafts need not be connected together necessarily yet. If the shafts are already coupled, only one motor can be tested at a given time, otherwise both motors can be tested simultaneously.

1. Open '\app_headers\FCL_SFRA_XL_DualServo-Settings.h' and select level 3 incremental build option by setting the BUILDLEVEL to FCL_LEVEL3 (#define BUILDLEVEL FCL_LEVEL3). The current and position feedbacks can be sampled once or twice per PWM period depending on the sampling method. The sampling is synchronized to carrier max in single sampling method and to carrier max and carrier zero in double sampling method. This is done in the example by selecting the SAMPLING_METHOD to SINGLE_SAMPLING or DOUBLE_SAMPLING.

   NOTE: The maximum modulation index changes from 0.92 in SINGLE_SAMPLING method to 0.84 in DOUBLE_SAMPLING method. If the PWM_FREQUENCY is changed from 10KHz, this will also change. Since a single CPU is controlling two motors, the max modulation index is reduced. The scope of this experiment is to quantify the performance of fast current loop and hence a single CPU based uncomplicated programming approach is used. If each CPU is used to control individual motors, this would be 0.97 in SINGLE_SAMPLING method to 0.94 in DOUBLE_SAMPLING method.

2. 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, within the 'motor1' structure, 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.FccD: preferred bandwidth of d-axis current loop
- FCL_Pars.FccQ: preferred bandwidth of q-axis current loop
- fcl_LatencyInMicroSec: shows latency between ADC / 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
- QepCalibSm : state machine variable indicating the QEP calibration status of both motors
- QepCalFlag : flag to initiate QEP calibration of both motor when CPU starts from RESET
- QepRecalFlag : flag to redo QEP calibration of both motors

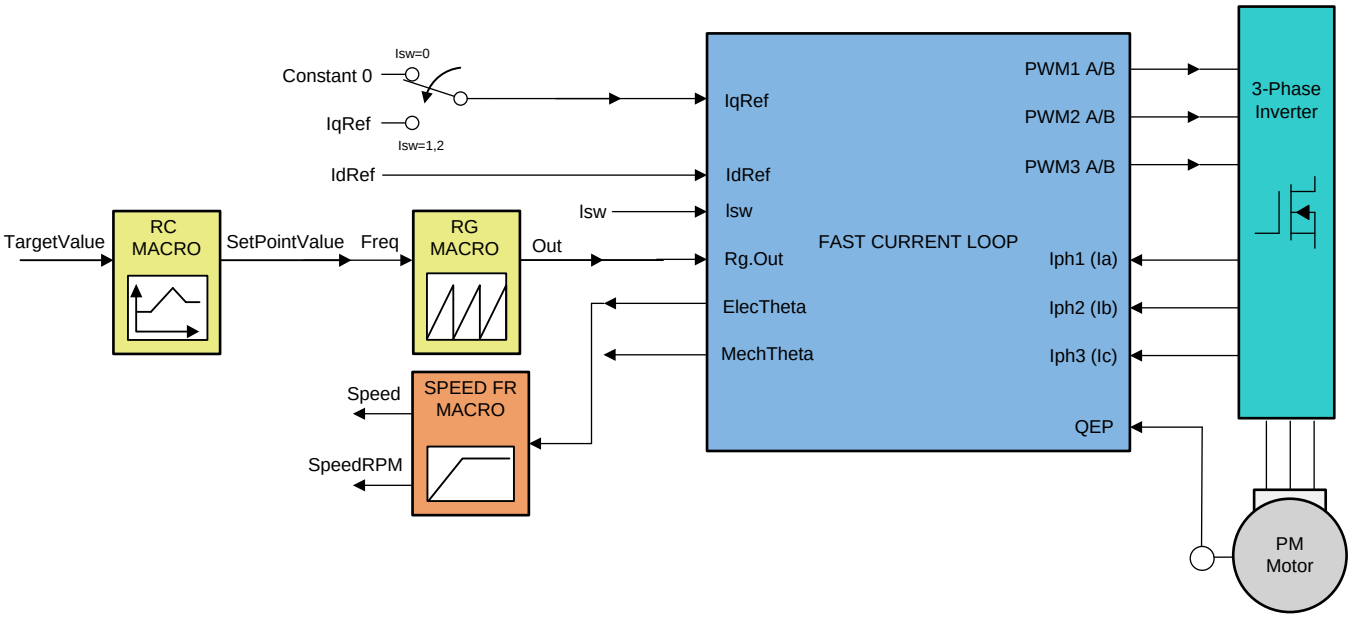


Figure 16. Level 3 Block Diagram

The key steps are explained as follows:

1. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in watch windows to confirm the interrupt working properly.
2. Set ‘QepCalFlag’ to 1 to initiate QEP calibration. Each motor will be spun for a maximum of one rotation until it catches the QEP index pulse. First, notice that the shaft moves to a certain position and holds for a while (alignment by motor 1) before spinning slowly. When QEP1 index pulse is received, motor 1 will stop spinning the shaft and leave it to the control of motor 2. Second, notice that the shaft realigns to another position for a while (alignment by motor 2), before spinning in the opposite direction. During this calibration process, the soft-switch variable (lsw) for each motor is auto promoted in a sequence. ‘lsw’ manages the loop setting as follows:
   a. lsw = Alignment --> lock the rotor of the motor
   b. lsw = WaitForIndex --> motor running, waiting for first instance of QEP Index pulse
   c. lsw = GotIndex --> first Index pulse occurred
3. If calibration is successful, the lsw of both motor1 and motor2 variables should show as ‘GotIndex’
4. ‘QepRecalFlag’ can be set to 1 if QEP calibration should be verified. Notice that that depending on the pole pair of the motor, the shaft can reorient to lay within any of the pole pairs during alignment.
5. Verify if ‘maxModIndex’ value is either 0.88 in double sampling method or 0.94 in single sampling method.
6. Set Idref_run to 0.05 pu (0.68A) default 0 is also fine) and Iqref to 0.05 pu (0.68A) (or any other suitable value).
7. Set ‘RunMotor’ flag to RUN to run the motor.
8. Check cntlr_id.fbk in the watch windows with continuous refresh feature and see if it can track IdRef.

9. Check cntlr_iq.fbk in the watch windows with continuous refresh feature and see if it can track IqRef.
10. To confirm these two current regulator modules, try different values of cntlr_id.Ref and cntlr_iq.Ref. If the motor shaft can be rigidly locked from spinning, then the IqRef value can be changed back and forth from 0.5 to –0.5, to study the effect of loop bandwidth.
11. Try different bandwidths for the current loop by tweaking the values of 'FCL_Pars.FccD' and 'FCL_Pars.FccQ'. The default setting for bandwidth is 1/18 of sampling frequency.
12. Set 'RunMotor' to STOP to stop the motor.
13. The same set of tests can be done on motor 2 by working with structure variable 'motor2'.
14. After testing both motors, take the controller out of real time mode and reset.

7.4.1 Observation One – Sample to PWM update Latency

While running the motor in this build level and subsequent build levels, notice the variable fcl_LatencyInMicroSec in the Expressions window. Figure 17 shows a snapshot of the Expressions window.

fcl_ClrCntr	int	0	0x0000B001@Data
fcl_LatencyInMicroSec	float	1.63	0x0000B03A@Data
fcl_cycle_count	unsigned int	160	0x0000B003@Data

Figure 17. Expressions Window Snapshot for Latency

It indicates the amount of time elapsed between feedback sampling and PWM updating for the first motor. This value can be refreshed by setting 'fcl_clrCntr' to 1. The elapsed time, or latency, is computed based on the count of EPWM timer right after PWM update. Regardless of SAMPLING_METHOD, latency remains the same. For this demo, where the CLA is not used, and that SFRA code is also included in the critical path, the latency time shown is fairly large. Refer to the original FCL release for optimized FCL performance.

NOTE: By using CLA1 and CPU2/CLA2, it is possible to reduce the latency time to within a micro second for each motor.

7.4.2 Bandwidth

If the bandwidth is increased beyond a fourth of the sampling frequency, control can be noisy. However, depending on the quality of current sensing, the system can be more noisy at much lower bandwidths.

7.5 Incremental Build Level 4

Assuming the previous section is completed successfully; this section verifies the speed PI module and speed loop. The implementation block diagram is given in Figure 18. The motor shafts must be kept disconnected to run these motors at different speed settings simultaneously or only one motor can be tested at a given time.

1. Open '\app_headers\FCL_SFRA_XL_DualServo-Settings.h' and select level 4 incremental build option by setting the BUILDLEVEL to FCL_LEVEL4 (#define BUILDLEVEL FCL_LEVEL4).
2. Right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run.
3. In the software, within the 'motor1' structure, the key variables to be adjusted are summarized as follows:
   a. SpeedRef: for changing the rotor speed in per-unit.
   b. IdRef: for changing the d-axis voltage in per-unit.

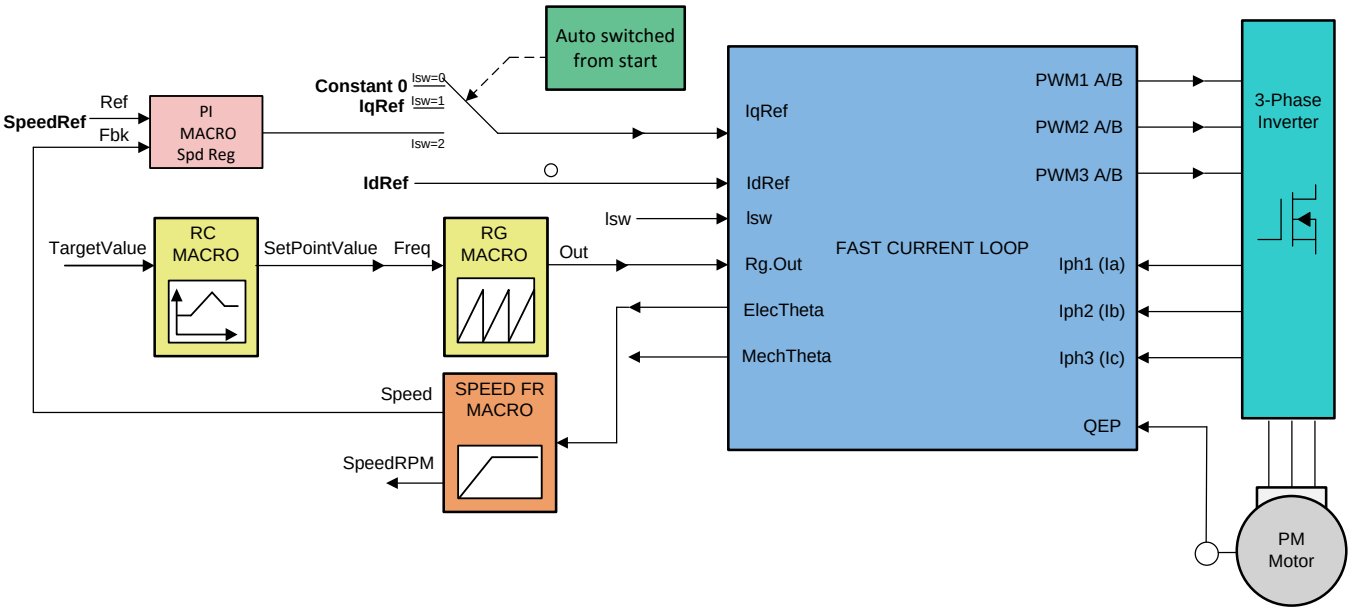


Figure 18. Level 4 Block Diagram

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 'QepCalFlag' to 1 to initiate QEP calibration. Each motor will be spun for a maximum of one rotation until it catches the QEP index pulse. Firstly, notice that the shaft will move to a certain position and hold for a while (alignment by motor 1) before spinning slowly. When QEP1 index pulse is received, motor 1 will stop spinning the shaft and leave it to the control of motor 2. Notice that the shaft will realign another position for a while (alignment by motor 2), before spinning in opposite direction. During this calibration process, the soft-switch variable (lsw) for each motor is auto promoted in a sequence. 'lsw' manages the loop setting as follows:
   a. lsw = Alignment --> lock the rotor of the motor
   b. lsw = WaitForIndex --> motor running, waiting for first instance of QEP Index pulse
   c. lsw = GotIndex --> first Index pulse occurred
3. If calibration is successful, the lsw of both motor1 and motor2 variables should show as 'GotIndex'
4. Set SpeedRef to 0.1 pu (1.0 pu speed is 500Hz, rated speed of motor is 0.8 pu)or any other lower value until familiarity with test setup.
5. Set 'RunMotor' to RUN and now the motor should run with this reference speed (0.3 pu). Compare Speed with SpeedRef in the watch windows with continuous refresh feature whether or not it should be nearly the same.
6. To confirm this speed PID module, try different values of SpeedRef (positive or negative). The P, I and D and integral correction gains may be tweaked to get satisfactory response.
7. At very low speed range, the performance of speed loop relies heavily on the quality of rotor position angle provided by QEP encoder.
8. Set 'RunMotor' to STOP to stop the motor.
9. The same set of tests can be done on motor 2 by working with structure variable 'motor2'.
10. After testing both motors, take the controller out of real time mode and reset.

7.6 Incremental Build Level 5

Assuming the previous build level is verified successfully where two motors were independently run using FCL at various speed and load settings, this build level verifies their operation working together in sync. In this build level also, each motor will operate at the functional level of BUILDLEVEL 4. But then, one of the motors will act as drive motor, controlling the shaft speed, and the other as a generator applying load torque on the drive motor. Since these two drives share the same dc bus, the power generated out of the generator is fed back in to the drive motor and therefore a larger power supply is not needed for this experiment.

1. Couple the two motors back to back as shown in Figure 9. In the software, you can configure which motor will act as the load motor or drive motor. The implementation block diagram is given in Figure 19.

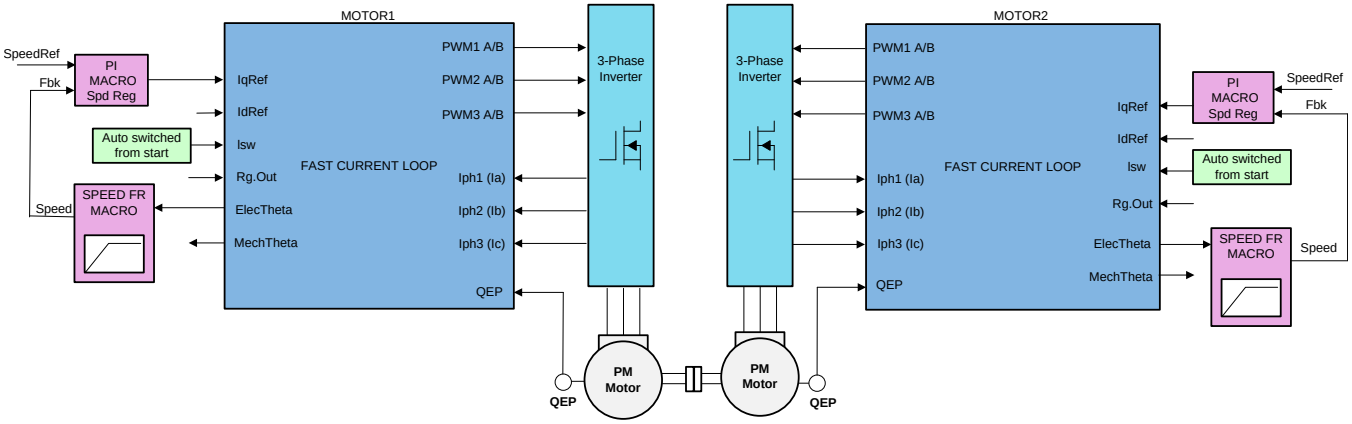


Figure 19. Level 5 Block Diagram

2. Open '\app_headers\FCL_SFRA_XL_DualServo-Settings.h' and select level 5 incremental build option by setting the BUILDLEVEL to FCL_LEVEL5 (#define BUILDLEVEL FCL_LEVEL5).
3. Right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run.

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

loadMotor : for selecting either 'motor1' or 'motor2' as load motor

The key steps can be explained as follows:

1. Set "EnableFlag" to 1 in the watch window. The variable named "IsrTicker" will be incrementally increased as seen in watch windows to confirm the interrupt working properly.
2. Set 'QepCalFlag' to 1 to initiate QEP calibration. Each motor will be spun for a maximum of one rotation until it catches the QEP index pulse. First, notice that the shaft moves to a certain position and holds for a while (alignment by motor 1) before spinning slowly. When QEP1 index pulse is received, motor 1 will stop spinning the shaft and leave it to the control of motor 2. Second, notice that the shaft realigns another position for a while (alignment by motor 2), before spinning in the opposite direction. During this calibration process, the soft-switch variable (lsw) for each motor is auto promoted in a sequence. 'lsw' manages the loop setting as follows:
   a. lsw = Alignment --> lock the rotor of the motor
   b. lsw = WaitForIndex --> motor running, waiting for first instance of QEP Index pulse
   c. lsw = GotIndex --> first Index pulse occurred
3. If calibration is successful, the lsw of both motor1 and motor2 variables should show as 'GotIndex'
4. Set 'loadMotor' to, say, MOTOR2. This will set up motor 2 as generator (or load motor) and motor 1 as drive motor. This will also set the max torque capacity of the drive motor a little larger than that of the load motor so that it can drive the shaft to the commanded speed regardless of load torque and across the entire speed command range. Watch out for the variables pid_spd.param.Umax (Umin) within the structure of motor1 and motor2 to confirm.

5. For motor1, set ‘SpeedRef’ = 0.05 (in pu, 1 pu = 500Hz) and then set ‘RunMotor’ = RUN to run the drive motor. Now motor 1 shaft should start spinning and settle at the commanded speed.
6. The load motor is not running yet, so the drive motor would not see much load.
7. For motor 2, the ‘speedRef’ should be zero. (If not, set it to 0). Now, set ‘RunMotor’ to RUN. The load motor will try to resist the drive motor from spinning the shaft until its allowed torque capacity. Since the drive motor is allowed more torque capacity than the load (motor1.pid_spd.param.Umax > motor2.pid_spd.param.Umax), the load motor is overpowered and the drive motor will spin the shaft at commanded speed.

You can change the value of motor2.pid_spd.param.Umax (or Umin) to see how it impacts the system, drive side and load side. This can be repeated at various speeds and directions and current loop bandwidths for evaluation purposes.

7.7 Incremental Build Level 6

Assuming the previous build level is successfully completed, this build level attempts to study the frequency response analysis of the Fast Current Loop using C2000’s Software Frequency Response Analyzer (SFRA) tool, available as a library in the ControlSuite.

7.7.1 Integrating SFRA Library

The SFRA tool runs only on C2000 MCU platform to study the frequency response analysis of any control loop that it controls. It consists of an embedded firmware part that runs on the MCU and a graphical user interface part (GUI) that runs on the development computer.

The embedded firmware is available as a library in the controlSUITE at:

\controlSUITE\libs\app_libs\SFRA\v1_10_00_00 \

Tthe SFRA GUI is available as an executable application in the controlSUITE at:

\controlSUITE\libs\app_libs\SFRA\v1_10_00_00\GUI\SFRA_GUI.exe

The *C2000™ Software Frequency Response Analyzer (SFRA) Library and Compensation Designer User's Guide* for working with SFRA can be found at:

controlSUITE\libs\app_libs\SFRA\v1_10_00_00\Doc

Some example projects to understand SFRA are available at:

\controlSUITE\libs\app_libs\SFRA\v1_10_00_00\Examples

In the ISR code, there are two functions that inject noise for SFRA and then collect the feedback data from the loop, they are:

- sfraInject(MOTOR_VARS *)
- sfraCollect(MOTOR_VARS *)

The roles of these functions are self-explanatory from their names. They should be used in the sequence they are used in the code to collect the data in right sequence. The motor handle passed as argument to these functions will determine which motor’s loop will cycle through for SFRA evaluation.

This provides the ability to study the D-axis or Q-axis current loops or the speed loop. When evaluating current loops, it is better to operate the test motor in load mode and the other motor run in drive mode at a certain constant speed. This will help to minimize the impact of speed jitter related errors in the test motor under SFRA study. This is particularly useful while studying the Iq loop. The loading level of the load motor can be varied by varying the ‘pid_spd.param.Umax’(and Umin) in its speed controller. The motor can be run at different speed / load conditions and at different bandwidths and the performance can be evaluated at each of these conditions. It will show that the controller can provide the designed bandwidth under all conditions, of course with a certain tolerance.

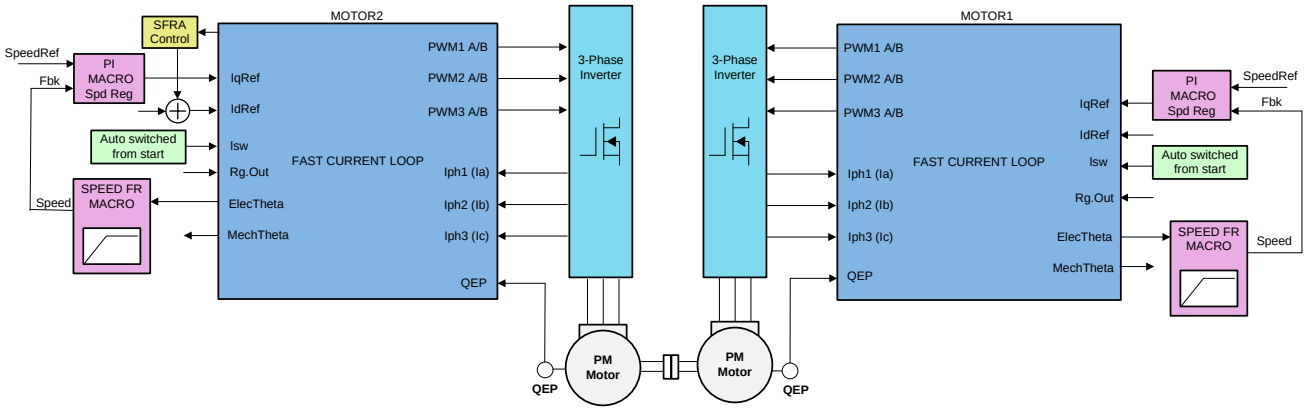


Figure 20. Level 6 Block Diagram

The implementation block diagram is given in Figure 20. The SFRA tool injects noise signal into the system at various frequencies and analyzes the system response and provides a Bode Plot of the actual physical system as seen during the test. In a fully compensated and decoupled current loop, back emf should not have any influence in the current loop dynamics. To verify this, it makes sense to study the behavior of the loop at zero speed when the back emf is zero, and then use it as a reference to study the behavior at different speeds. Especially, when the Q axis current loop response is studied, injection of noise into the system for FRA study can cause minor rotation of the motor shaft at low signal frequencies. This may have a tendency to influence the frequency response analysis inadvertently. To ensure a robust speed hold, this test setup will help where the drive motor will try to hold constant speed of the shaft regardless of noise injections in Iq current of the load motor.

7.7.2 Initial Setup Before Starting SFRA

Open '\app_headers\FCL_SFRA_XL_DualServo-Settings.h' and select level 6 incremental build option by setting the BUILDLEVEL to FCL_LEVEL6 (#define BUILDLEVEL FCL_LEVEL6).

Also, in this file, watch out for the definitions:

- SFRA_FREQ_START
- SFRA_FREQ_LENGTH
- FREQ_STEP_MULTIPLY

These definitions inform the GUI about the starting value of noise frequency, number of different noise frequencies to sweep and the ratio between successive sweep frequencies respectively. More information is available in the *C2000™ Software Frequency Response Analyzer (SFRA) Library and Compensation Designer User's Guide* associated with SFRA. In the context of this evaluation project, it is important to know and appreciate these parameters to tweak them for further repeat tests. The default setting of the code is that SFRA will be performed on motor2. To do SFRA on motor 1, call the SFRA functions with the 'motor1' handle (or pointer).

In this motor control project, there are three different loops such as the speed loop, D axis current loop and Q axis current loop. Any of these loops could be analyzed for frequency response. Technically, this could be performed on position loop as well, but is not included in this project scope.

Right click on the project name and click Rebuild Project. Once the build is complete click on debug button, reset CPU, restart, enable real time mode and run.

Add the following variable in the 'Expressions Window':

- sfraTestLoop : for selecting the control loop on which to evaluate SFRA, letting you choose between:
  - 'D_axis' current loop
  - 'Q_axis' current loop
  - 'speedLoop'

The key steps can be explained as follows:

1. Set “EnableFlag” to 1 in the watch window. The variable named “IsrTicker” will be incrementally increased as seen in watch windows to confirm the interrupt working properly.
2. Set ‘QepCalFlag’ to 1 to initiate QEP calibration. Each motor will be spun for a maximum of one rotation until it catches the QEP index pulse. First, notice that the shaft moves to a certain position and holds for a while (alignment by motor 1) before spinning slowly. When QEP1 index pulse is received, motor 1 will stop spinning the shaft and leave it to the control of motor 2. Second, notice that the shaft realigns another position for a while (alignment by motor 2), before spinning in the opposite direction. During this calibration process, the soft-switch variable (lsw) for each motor is auto promoted in a sequence. ‘lsw’ manages the loop setting as follows:
   a. lsw = Alignment --> lock the rotor of the motor
   b. lsw = WaitForIndex --> motor running, waiting for first instance of QEP Index pulse
   c. lsw = GotIndex --> first Index pulse occurred
3. If calibration is successful, the lsw of both motor1 and motor2 variables should show as ‘GotIndex’

7.7.3 Setting up SFRA GUI

The SFRA GUI can be invoked and connected to the target platform to study the control loops. The GUI executable is available in the location as mentioned in Section 7.7.1.

Double click on it and the GUI screen will appear as shown in Figure 21. The GUI lets you select appropriate settings based on the target platform and the development computer.

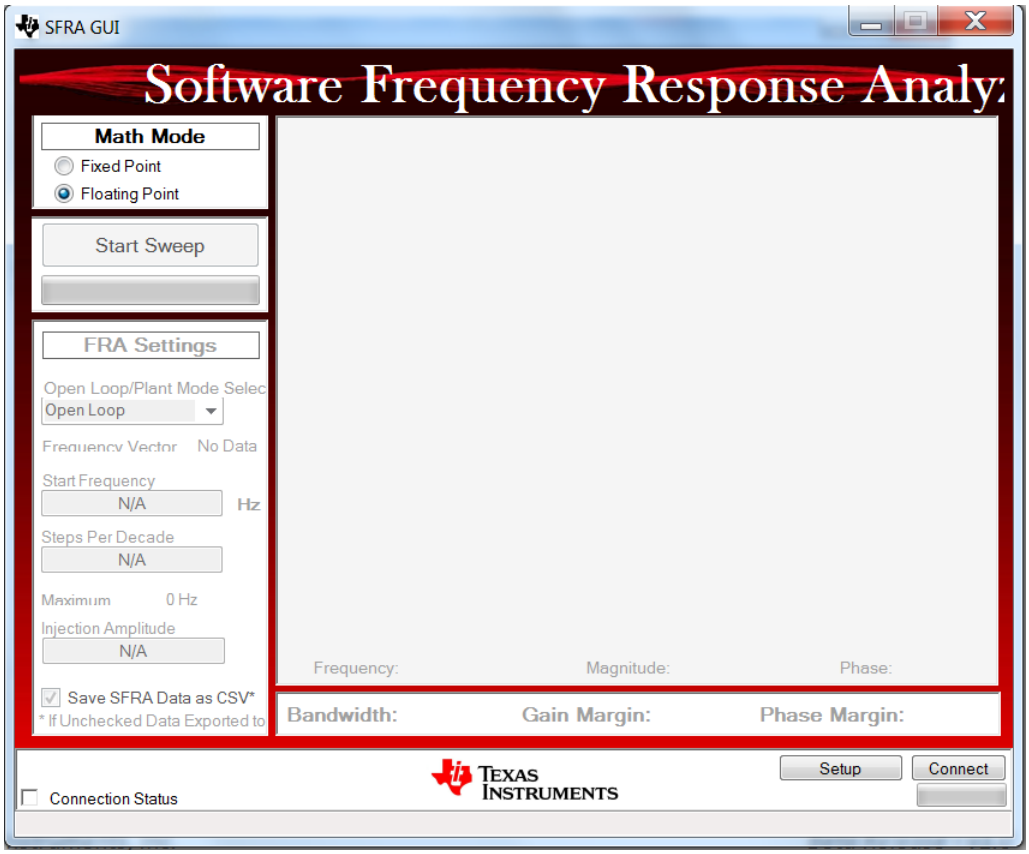


Figure 21. SFRA GUI

The following is a list of things to do on the GUI before starting the analysis.

1. Math Mode: Depending on the target C2000 development platform, either the fixed point or floating point option is chosen. For F28379DXL, select the 'Floating Point' option
2. Since the USB port on the launch pad is already connected to the computer for JTAG purposes, nothing new need to be connected. However, for a standalone operation, an USB connector needs to be connected to the target board at this point. In the XDS100 emulator present on the launch pad, along with JTAG setup, an SCI port is also configured and the GUI uses this SCI port for interface. While the debug environment of CCS is using JTAG, the GUI can also use SCI at the same time.
3. Click on the Setup button at the bottom right corner. This will pop open a Setup Connection window as shown in Figure 22.

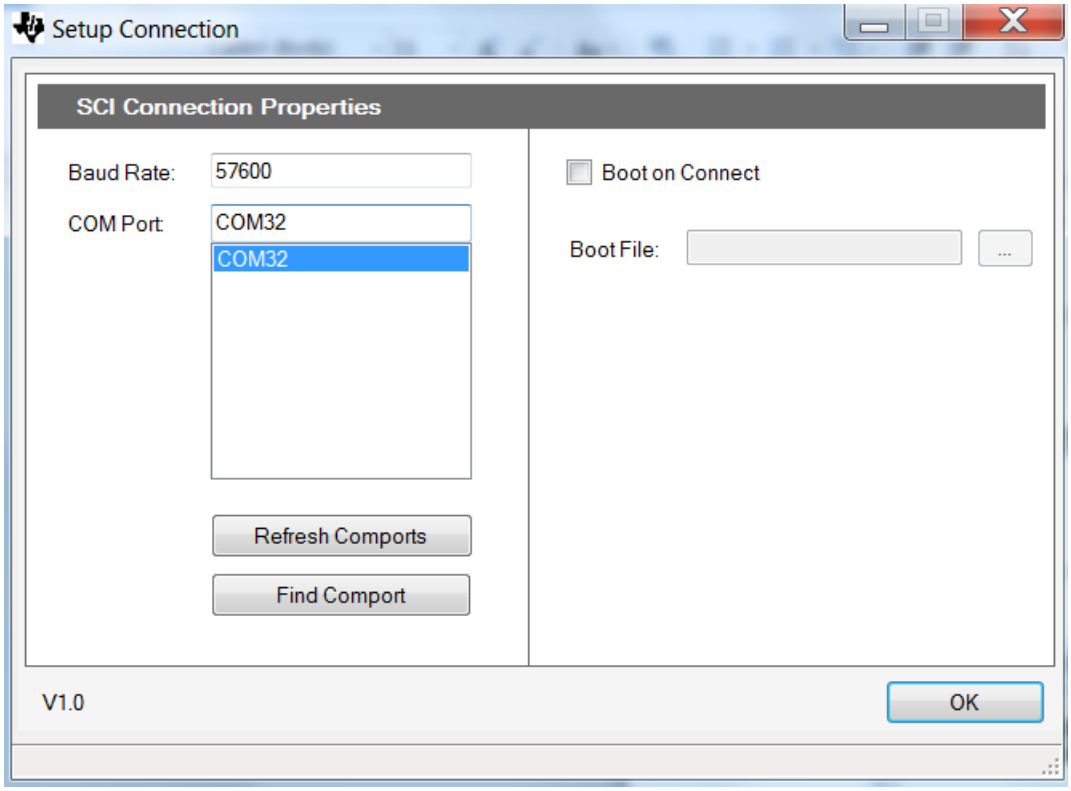


Figure 22. GUI Setup Diagram

4. Click 'Refresh Comports' button to get the Comport number show up in the window.
5. Select the Comport representing the connection to the target C2000 board.
6. Uncheck 'Boot on Connect'
7. Click OK button

8. This should establish the connection to the F28379D Launchpad and the GUI will appear as shown in Figure 23 indicating the connection status at the bottom left corner.

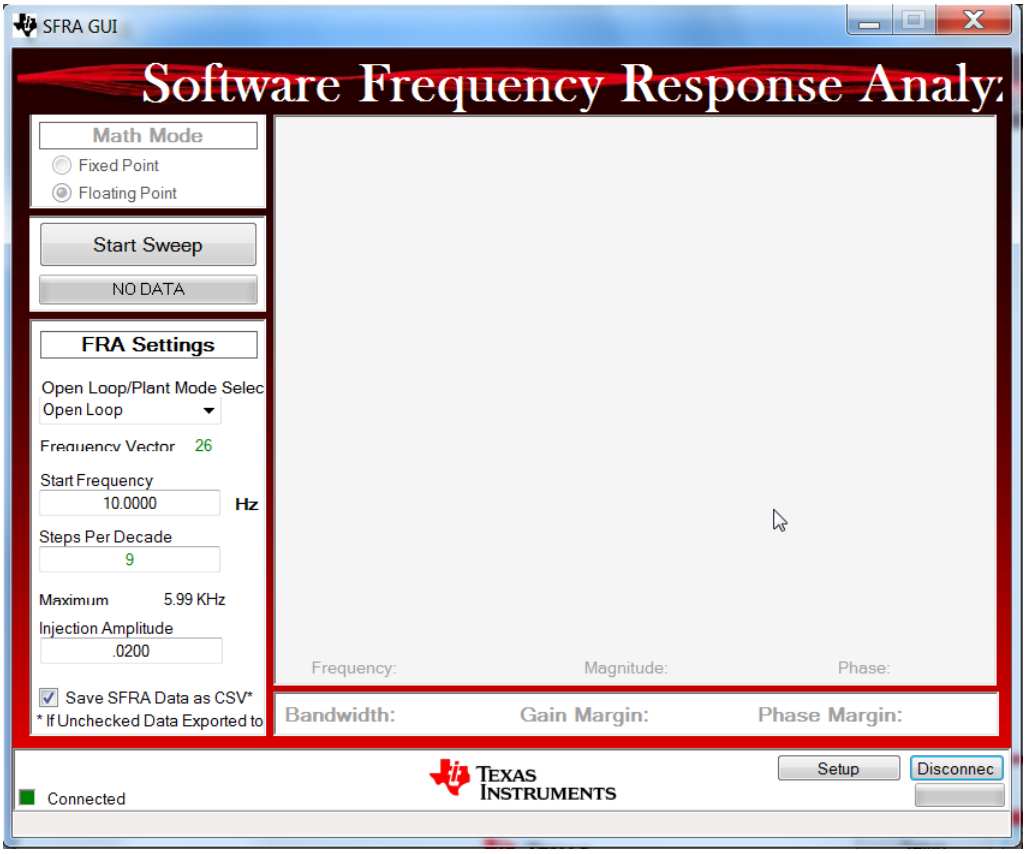


Figure 23. SFRA GUI Connected to F28379SXL

9. The frequency sweep related settings are shown in 'FRA Settings' Panel. These values are already pre-filled from the C2000 device and they can be left as is.

This completes the initial setup of GUI environment.

7.7.4 Running the SFRA GUI

This step involves co-ordination between the debug environment and the SFRA GUI. From the debug environment, the steps to be followed are shown below:

1. Set 'loadMotor' to, say, MOTOR2. This will set up motor 2 as generator (or load motor) and motor 1 as drive motor. This will also set the max torque capacity of the drive motor a little larger than that of the load motor so that it can drive the shaft to the commanded speed regardless of load torque and across the entire speed command range. Watch out for the variables pid_spd.param.Umax (Umin) within the structure of motor1 and motor2 to confirm.
2. Set 'sfraTestLoop' as 'D_axis' to test Id loop (of motor 2 by default).
3. Set 'motor2.FCL_Pars.FccD' to the desired value, within limits, (when test is performed for Q axis, adjust this parameter for Q axis - 'motor2.FCL_Pars.FccQ')
4. For motor1, set 'SpeedRef' = 0.05 (in pu, 1 pu = 500Hz) and then set 'RunMotor' = RUN to run the drive motor. Now motor 1 shaft should start spinning and settle at the commanded speed.
5. The load motor is not running yet, so the drive motor would not see much load.
6. For motor 2, the 'speedRef' should be zero. (If not, set it to 0). Now, set 'RunMotor' to RUN. The load motor will try to resist the drive motor from spinning the shaft until its allowed torque capacity. Since the drive motor is allowed more torque capacity than the load (motor1.pid_spd.param.Umax > motor2.pid_spd.param.Umax), the load motor is overpowered and the drive motor will spin the shaft at commanded speed.

7. Now that both motors are running, the GUI can be called into perform a frequency sweep of the D axis current loop in motor 2, by clicking on the 'Start Sweep' button in the GUI. The sweep progress will be indicated by a green bar in the location marked as ‘NO DATA’.
8. When the frequency sweep is fully done, it will compute the Bode plot and display the results as shown in Figure 24.
9. The GUI also computes and displays the loop bandwidth, gain margin and phase margin.

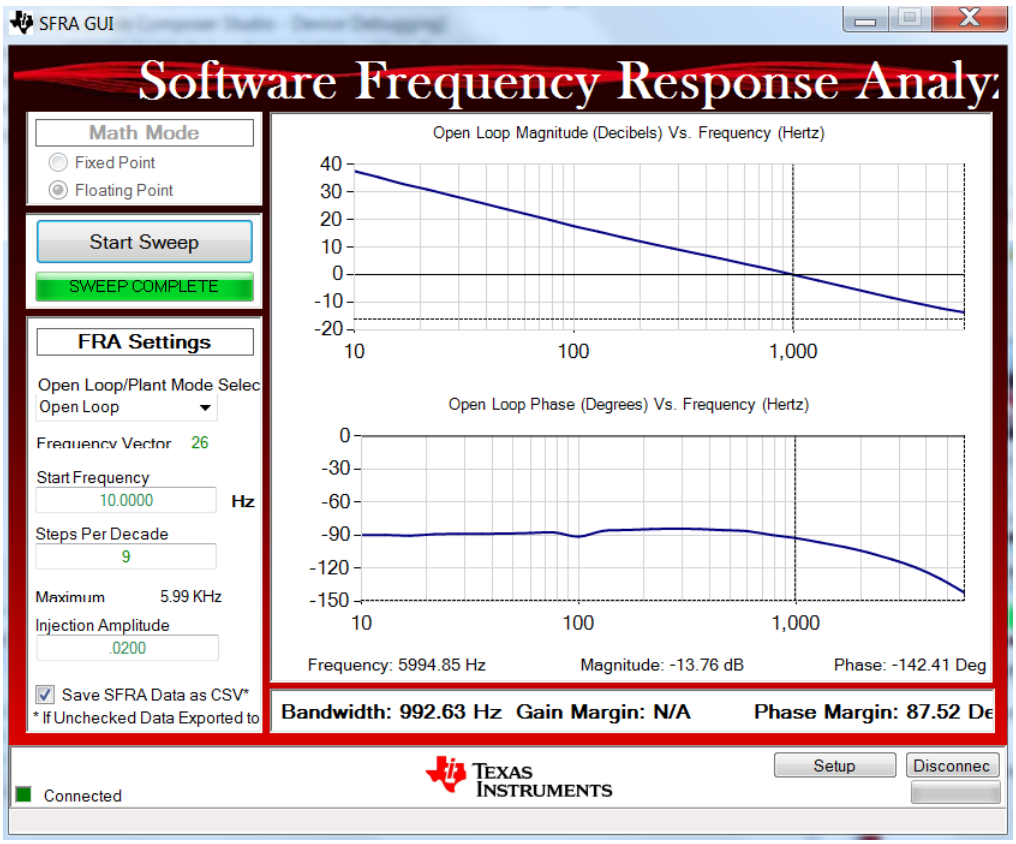


Figure 24. SFRA Bode Plots of the Current Loop Showing the Loop Magnitude and Phase Angle

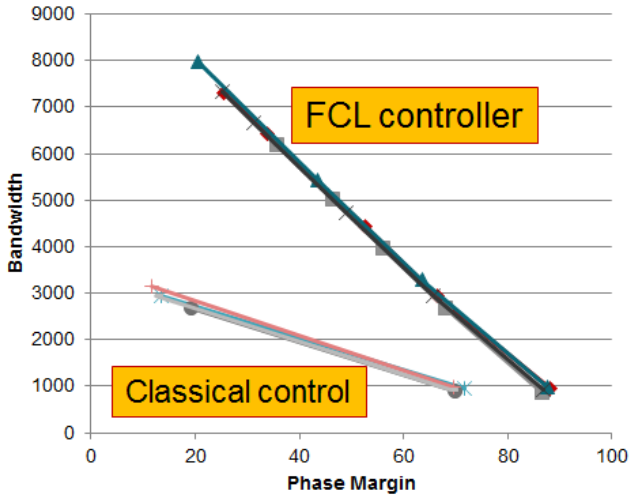


Figure 25. Plot of Gain Margin vs Phase Margin as Experimentally Obtained

As mentioned earlier, the test can be performed at zero speed to get the plot as reference. The bandwidth and phase margin at zero speed may be noted down. Then at different speeds and load conditions, this test can be repeated to verify if there is any change in bandwidth or phase margin. Any variation in the plot at different speed is indicative of the quality of decoupling in current loops.

NOTE: The SFRA can be performed on the drive motor (MOTOR1) by simply setting 'loadMotor' to 'MOTOR1', in which case, the MOTOR2 becomes the drive motor. Alternately, the same can be achieved by changing the code such as to pass the pointer of 'motor1' to the 'sfraInject(MOTOR_VARS * m)' and 'sfraCollect(MOTOR_VARS * m)' functions. The resulting plot can be compared against that obtained from the load motor for further analysis depending on user interest. The factors that could influence the difference between these plots would be the inherent feedback signal quality and the quality of voltage decoupling in the current loops.

By varying the control bandwidth and repeating these tests and noting down the resulting bandwidth and phase margin, some plots are obtained for Id loop as shown in Figure 25. Two sets of tests are performed, one with, and the other without, FCL.

The group of plots at the bottom is obtained for conventional control, that is, without FCL. It is self-revealing that the control bandwidth is too low and that as the control bandwidth increases, the phase margin drops a lot faster.

The group of plots at the top is obtained with FCL. It shows that the controller is capable of providing a higher bandwidth at the cost of relatively reduced drop in phase margin compared to that without FCL. This effectively means that FCL can provide a higher bandwidth at a higher phase margin. The best performance is found to be when the bandwidth is about 1/6th of the sampling frequency where it almost behaves like deadbeat control. For frequencies beyond that, overshoots are noticed. In this particular evaluation, the PWM frequency is 10 KHz and the sampling frequency is about 20 KHz. Therefore, the best performance is obtained when the control bandwidth is 3.3 KHz.

8 Summary

This evaluation platform helps to control two different motors, either from one CPU core or two different CPU cores, and with or without FCL technology. When FCL is used, it shows a substantial improvement in control bandwidth and phase margin. The SFRA tool showed the impact of FCL on control bandwidth and phase margin.

The presence of fast ADC, control law architecture (CLA) and trigonometric math unit (TMU) helped to reduce the latency between feedback sampling and PWM update and that, in turn, resulted in higher control bandwidth and increase in maximum modulation index. Higher modulation index helps to improve the dc bus utilization by the drive and to increase the control speed range of the motor.

Depending on the control speed range of motors in target applications, it is possible to control multiple motors in multi-axes configurations using FCL, out of the dual core F28379D MCU platform. This makes it suitable for high end servo control applications.

9 References

- *Fast Current Loop Library*
- *Fast Current Loop (C28x) Library*
- *LAUNCHXL-F28379D Overview User's Guide*
- *BOOSTXL-DRV8305 Hardware User's Guide*
- *BOOSTXL-3PhGaNInv Evaluation Module User's Guide*
- *C2000™ Software Frequency Response Analyzer (SFRA) Library and Compensation Designer User's Guide*

Revision History

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

Changes from Original (March 2018) to A Revision	Page
• Updates were made in Section 7.4.	23
• Updates were made in Section 7.5.	25
• Updates were made in Section 7.6.	27
• Updates were made in Section 7.7.4.	32

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