Application Report

EtherCAT-Based Connected Servo Drive Using Fast Current Loop on PMSM

Ramesh T Ramamoorthy

ABSTRACT

This application report helps to evaluate EtherCAT® communication and to perform frequency response analysis of fast current loop (FCL) enabled control loops of a connected servo drive using TI's TMS320F28388D real-time controller. The Configurable Logic Block (CLB) present in this device can help to interface to a wide range of absolute serial encoders, typically seen in many industrial drives for position sensing, without external logics or FPGAs. The position encoder used in this evaluation is QEP and T-format type. The current sensing method is hall sensor using ADC or shunt resistor using SDFM. The platform used is TI's DesignDRIVE IDDK EVM kit and the C2000Ware MotorControl SDK.

Frequency response analysis is performed in software using TI's Software Frequency Response Analyzer (SFRA) library running in real-time on the control MCU F28388D. Besides controlling the motor, the MCU also performs EtherCAT communication with a TwinCAT running on PC. F2838x MCU is a part of the C2000™ family of microcontrollers. It has has two C28x CPU cores and an Arm® Cortex®-M4 core for performing control and communication effectively using a range of powerful peripherals for digital power and motor control applications. In servo drive space, it enables cost-effective design of intelligent, high-bandwidth controllers for three-phase motors, by reducing the system components and increasing efficiency. Due to the higher level of integration of hardware accelerators, features and software support, FCL algorithms can be effectively implemented on this device 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 and EtherCAT functions spread across CPU, M4 and CLA cores
- Inline current sensing with LEM sensor using ADC or isolated shunt resistor based current measurement using SDFM
- Position feedback using QEP or T-Format encoder
- Experimental results

Table of Contents

List of Figures

List of Tables

Trademarks

C2000™ is a trademark of Texas Instruments.
EtherCAT® is a registered trademark of Beckhoff Automation GmbH, Germany.
Arm® and Cortex® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

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 highly time critical computational burden of these systems and the need for flexible PWMs, a combination of FPGAs, fast external ADCs, and multiple MCUs are used by many designers.

With the TMS320F2838x MCU, due to its higher level of integration, it is possible to implement fast current loop (FCL) algorithms that provide a high current loop bandwidth with the same external hardware as used in classical FOC methods. TI has developed the FCL algorithm on this MCU and implemented it on the Design DRIVE IDDK platform.

With a 10-kHz PWM carrier, the current loop gain crossover frequency is expected to exceed 3 kHz, and the closed loop bandwidth is expected at about 5 KHz (per NEMA ICS 16 and Chinese GBT 16439-2009 guidelines) and the maximum duty cycle is expected to be approximately 96%. Using TI's Software Frequency Response Analyzer (SFRA) library, frequency response analysis of the current loops can be performed in real time to verify the above benchmarks. 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.

Due to the presence of a configurable logic block (CLB), it is now possible to implement custom interface logics for various absolute encoders that utilize various protocols such as EnDAT, BiSS, T-format, and so forth, without external logics or FPGAs.

Besides control, TMS320F2838x MCU has an Arm Cortex-M4 based Connectivity Manager and an EtherCAT Slave Control peripheral. This helps to seamlessly integrate control and communication in a single chip to enable development of cost effective solutions for industrial servo drives.

This document evaluates the implementation of FCL algorithms on C2000 devices, studies the frequency response analysis of current loops in real time, verifies the interface logics for T-format encoder using on-chip configurable logic blocks (CLB) and also performs EtherCAT communication with a master to serve as a connected drive. The position loop can be closed using a QEP encoder or a T-format encoder and FCL can be implemented in both cases. Quantitative test results from frequency response analysis are discussed.

1.1 Acronyms Used in This Document

- 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
- ECAT - EtherCAT
- ePWM – Enhanced Pulse Width Modulator
- eQEP – Enhanced Quadrature Encoder Pulse Module
- FCL – Fast Current Loop
- FOC – Field-Oriented Control
- FPGA – Field Programmable Gate Array
- IDDK – Industrial Drive Development Kit (from TI)
- HVDMC – High Voltage DMC
- MCU – Microcontroller Unit
- PMSM – Permanent Magnet Synchronous Motor
- PWM – Pulse Width Modulation
- SDFM - Sigma-Delta Filter Module
- TMU – Trigonometric Mathematical Unit (in C2000 MCU)

2 Benefits of the TMS320F2838x 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, such as trip zones for PWMs and comparators.

The C28x core of F2838x MCU contains additional hardware features such as the following:

- Higher CPU and CLA clock frequency
- Four high-speed, 12- and 16-bit ADCs
- Trigonometric and Math Unit (TMU)
- Parallel processing block, such as the CLA
- Configurable Logic Block (CLB)
- Sigma-Delta Filter Module (SDFM)

Together, these features provide enough hardware support to increase computational bandwidth compared to its predecessors and offer superior real-time control performance. CLBs provide the flexibility to interface to a variety of absolute serial encoders employing various types of serial communication protocols, thereby making an FPGA redundant in a typical high bandwidth servo control application. In addition, the C2000 ecosystem of software (libraries and application software) and hardware (TMDXIDDK379D) help reduce the time and effort needed to develop a high-end digital motor control solution.

3 Current Loops in Servo Drives

Figure 3-1 shows the basic current loop used in FOC servo drives.

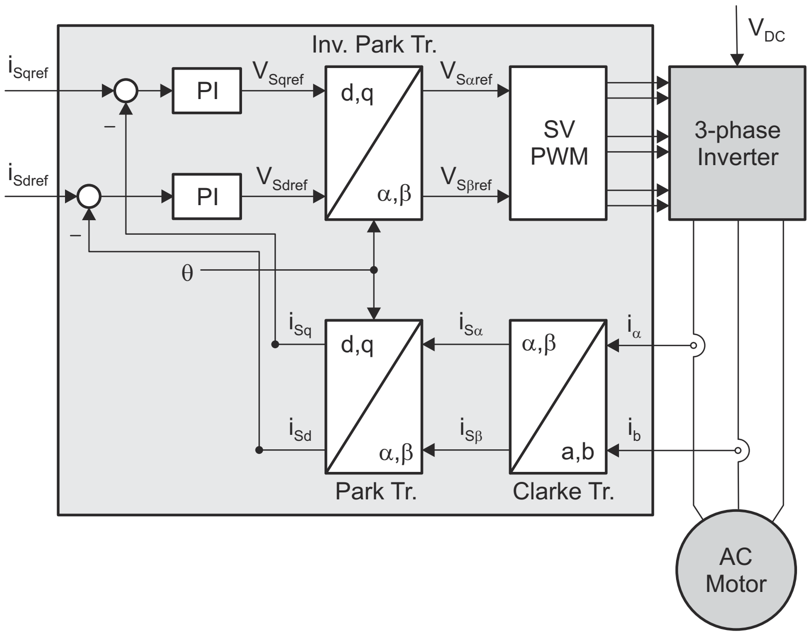


Figure 3-1. Basic Scheme of FOC for AC Motor

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. Fixing this problem will improve the current loop bandwidth. However, to achieve this without losing out on DC bus utilization, the following are necessary:

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

While the TMS320F2838x provide the necessary hardware support for higher performance, TI's FCL library, which runs on this real-time controllers, provide the necessary algorithmic support. The source code is available from MC SDK v2.01.00.00 and onward.

In order to make the current loop faster and to improve the operational range of current loops, the latency between feedback sampling and the PWM update must be as small as possible. This is depicted in Figure 4-1. Basically, it shows a PWM carrier cycle where system sampling is done at the carrier peak and the corresponding PWM update resulting from its control action is happening after a time T_{PWM_update}. This substantially reduces the transportation lag and improves the bandwidth. However, the down side is the loss of active PWMing time window that is spent in computing the new values of PWM duty cycles for the inverter. This is called the blanking window. The lower this blanking window, the higher the DC bus utilization and the operating speed range of motor.

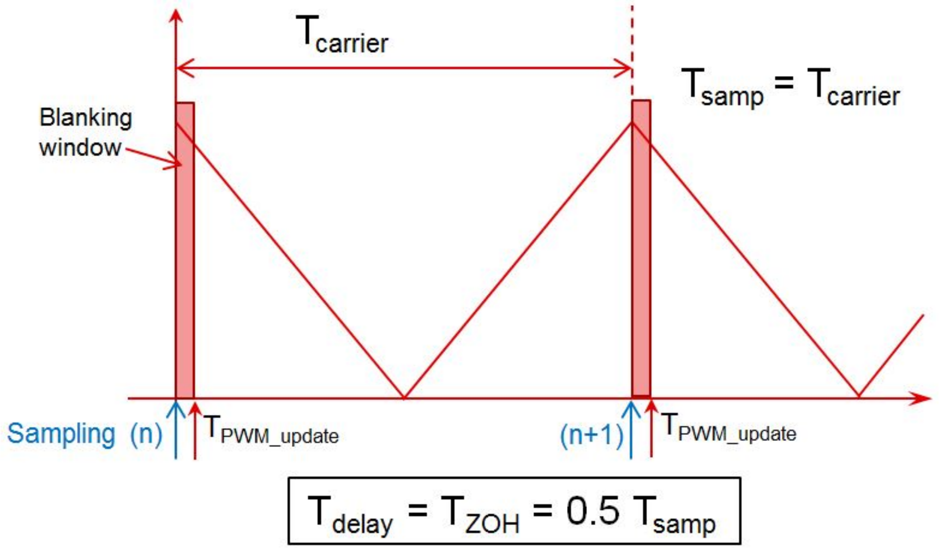


Figure 4-1. Time Lapse Between Feedback Sampling and PWM update

Typically, a latency of 2 μs or less is considered acceptable in many applications where the carrier frequency is 10 KHz. Traditionally, this task is implemented using a combination of high-end FPGAs, external ADCs, and MCUs. However, due to the availability of hardware support such as trigonometric math unit (TMU), CLA, high speed ADC or SDFM, single cycle ADC read and PWM write, it is possible to implement this control on this MCU without FPGAs or external ADCs.

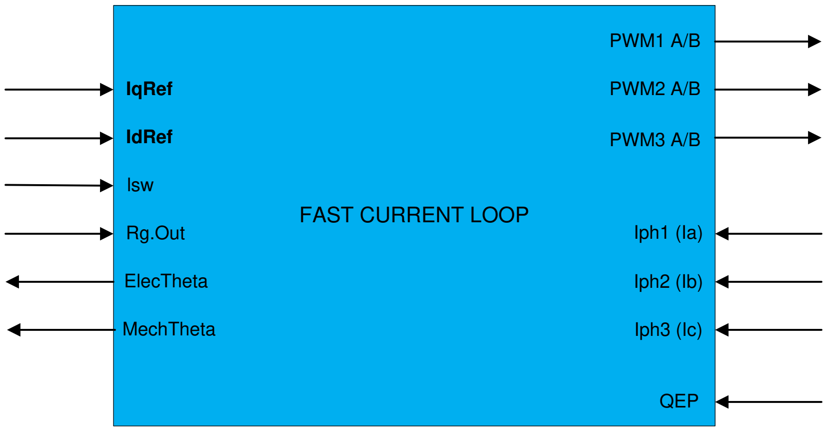


Figure 4-2. Fast Current Loop Library Block Diagram

The FCL library uses the following features in the F2838x MCU:

- TMU
- Four high-speed 12- or 16-bit ADCs, or Sigma-Delta Filter Module (SDFM)
- Additional processing blocks such as CLA

Figure 4-2 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 2.0 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, the current regulator can be selected by setting the FCL_CNTLR macro appropriately and studying how they compare.

Table 4-1 lists the FCL API functions and their descriptions.

Table 4-1. Summary of FCL Interface Functions

API Function	Description
uint32_t FCL_getSwVersion(void)	Function that returns a 32-bit constant, and for this version the value returned is 0x00000007.
void FCL_runComplexCtrl(void)	Function that performs the complex control as part of the FCL with QEP encoder and hallsensor using ADC
FCL_runSDFMComplexCtrl(void)	Function that performs the complex control as part of the FCL with QEP encoder and shunt resistor using SDFM
void FCL_runAbsEncComplexCtrl(void)	Function that performs the complex control as part of FCL with typical absolute encoders and hall sensor using ADC
void FCL_runSDFMAbsEncComplexCtrl(void)	Function that performs the complex control as part of FCL with typical absolute encoders and shunt resistor using SDFM
void FCL_runPICtrl(void)	Function that performs the PI control as part of the FCL with QEP encoder and hall sensor using ADC
void FCL_runSDFMPICtrl(void)	Function that performs the PI control as part of the FCL with QEP encoder and shunt resistor using SDFM
void FCL_runAbsEncPICtrl(void)	Function that performs the PI control as part of the FCL with typical absolute encoders and hall sensor using ADC
void FCL_runSDFMAbsEncPICtrl(void)	id)Function that performs the PI control as part of the FCL with typical absolute encoders and shunt resistor using SDFM
void FCL_runPICtrlWrap(void)	Wrap up function to be called by the user application at the completion of FCL in PI control mode before exiting ISR when QEP is the position sensor
void FCL_runAbsEncPICtrlWrap(void)	Wrap up function to be called by the user application at the completion of FCL in PI control mode before exiting ISR, when using abs encoders
void FCL_runQEPWrap(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_runComplexCtrlWrap(void)	Wrap up function to be called by the user application at the completion of FCL in complex control mode before exiting ISR when QEP is the position sensor
void FCL_runAbsEncComplexCtrlWrap(void)	Wrap up function to be called by the user application at the completion of FCL in complex control mode before exiting ISR, when using absolute encoders
void FCL_initPWM(uint32_t basePhaseU, uint32_t basePhaseV, uint32_t basePhaseW)	Function to initialize PWMs for the FCL operation, this will be called by the user application during the initialization or setup process
void FCL_resetController(void)	This function is called to reset the FCL variables and is useful when you want to stop the motor and restart the motor
void FCL_initQEP(uint32_t baseA)	This function initializes the eQEP peripheral for connecting to the QEP
void FCL_initADC(uint32_t resultBaseA, ADC_PPBNumber baseA_PPB, uint32_t resultBaseB, ADC_PPBNumber baseB_PPB, uint32_t adcBasePhaseW)	This function initializes the ADCs that are used to sense the motor phase currents

For more information on the library, see the *Fast Current Loop MotorControl SDK Library User's Guide* available at:

\ti\c2000\C2000Ware_MotorControl_SDK_2_01_00_00\libraries\fcl\docs.

The source code for FCL is available at

\ti\c2000\C2000Ware_MotorControl_SDK_2_01_00_00\libraries\fcl\source

Note

The library is written in a modular format and is able to port over to user platforms using F2838x devices if 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.
- PWM modules controlling motor phase A, B, and C are linked to the library.
- A QEP module connecting to the QEP sensor is linked to the library.
- CLA tasks one through four are used by the library. This must be accommodated in the user application.
- Separate, but similar, control functions are written to work with high latency T-format position encoder
- Similar control functions are written to work with SDFM current sensing

5 Fast Current Loop Evaluation

TI provides the FCL algorithm software library and its source code (from MC SDK v2.01.00.00 and onward) for evaluation using the TMS320F2838x, TMS320F2837x, or TMS320F28004x 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 PMSM motor
- FCL
- Position, speed, and torque control loops
- Position sensor support:
  - Incremental encoder (QEP)
  - Absolute encoder - T-Format
- Current sensing:
  - analog feedback using the ADC (from LEM sensors (Fluxgate/HALL))
  - shunt resistor using SDFM

5.1 Evaluation Setup

5.1.1 Hardware

The example project is evaluated on TI's Design DRIVE Development Kit IDDK - TMDXIDDK379D. The *DesignDRIVE IDDK Hardware Reference Guide* and *DesignDRIVE IDDK User Guide* for the kit can be found at *C:\ti\c2000\C2000Ware_MotorControl_SDK_2_01_00_00\solutions\tmdxiddk379d\docs\*.

The *DesignDRIVE IDDK Hardware Reference Guide* gives an overview of the various hardware functional blocks in the kit, discusses the various ground configurations it supports and the safety measures necessary to work with the kit. It is important to understand the hardware and the safety aspects before working with the kit.

The *DesignDRIVE IDDK User Guide* helps in setting up the hardware and in getting started with software, connecting to the target platform and working with the debug environment using Code Composer Studio.

5.1.2 Software

When the MCSDK software package is installed, the FCL software can be found at *C:\ti\c2000\C2000Ware_MotorControl_SDK_2_01_00_00\libraries\fcl\*.

The FCL example project can be found at:

C:\ti\c2000\C2000Ware_MotorControl_SDK_2_01_00_00\solutions\tmdxiddk379d\f2838x\ccs\sensored_foc

Software is built over various build levels to step through with integrating the FCL library to run the motor in speed mode and position mode.

5.1.3 FCL With T-Format Type Position Encoder

5.1.3.1 Connecting T-Format Encoder to IDDK

The T-format encoder is connected to header H6 on the IDDK that connects to macro M12, see the *DesignDRIVE IDDK Hardware Reference Guide*. This macro supports interfacing to various serial encoder protocols such as EnDAT, BiSS, T-format, and so forth. Make sure to match the pin out details of H6 with encoder signals and that jumpers [Main]-J6, [Main]-J7 and [Main]-J8 (in front of macro M9) on IDDK are populated. The CLK+ and CLK- signals of H6 are not needed for this encoder type. Only D+ and D- are needed.

Note

In the serial encoder interface macro M12 on IDDK, there is a load switch U4 to switch 5 V power supply to the encoder as needed. Due to heavy capacitive loading of some encoders, the 5 V power supply may glitch down and reset the MCU.

If it happens, there are two ways to fix this:

- Replace capacitor C5 to 10 nF (from 330 pF)
- Bypass this switch U4 by directly connecting *'5V0'* and *'5V0-Enc'* test points on this macro. This leads to encoder being powered up when ever the IDDK is powered.

5.1.3.2 T-Format Interface Software

TI provides a software library and its source code to interface to T-format encoders, which is based off Configurable Logic Blocks (CLB) in the device. An user guide to help with the *CLB tool*, and application reports to help with *Designing With The C2000 Configurable Logic Block* and *How to Migrate Custom Logic From an FPGA/CPLD to C2000 Microcontrollers* are available online. They are also available in C2000ware (release 2_00_00_03 or later) at the following location

c:\ti\c2000\C2000Ware_<version>\utilities\clb_tool\clb_syscfg\doc

In the implementation of T-format interface, communication is achieved primarily by the integration of following components:

- CPU
- Configurable Logic Block (CLB)
- Serial Peripheral Interface (SPI)
- Device Interconnect (XBARs)

While SPI performs the encoder data transmit and receive functions, clock generation is controlled by CLB. The following functions are implemented inside the CLB module. Note that the CLB module can only be accessed via library functions provided in the PM tfomat Library and not otherwise configurable by users.

- Ability to generate clock to the serial peripheral interface on chip and loop back to SPICLK input
- Identification of the critical delay between the clock edges sent to the encoder and the received data
- Ability to adjust the clock delay
- Monitoring the data coming from encoder, via SPI Save In Master Out (SPISIMO), and poll for start pulse
- Ability to measure the propagation delay at a specific interval as needed by the interface
- Ability to configure the block and adjust delay the via software

Position information is received through SPI and is read through the SPI receive ISR *spiRxFIFOISR()* using *readTformatEncPosition()*.

Detailed information about the T-format implementation on a launch pad platform is available at *c:\ti \c2000\C2000Ware_MotorControl_SDK_2_01_00_00\libraries\position_sensing\tformat\Docs*

The launch pad uses different GPIOs for T-format encoder interface as compared to IDDK. But for that, the core implementation is same. The list of GPIOs used on IDDK platform for cross reference are listed below:

```
#define  ENCODER_SPI_BASE    SPIB_BASE
#define  ENC_CLK_PWM_PIN     7
#define  ENC_SPI_SIMO_PIN    24
#define  ENC_SPI_SOMI_PIN    25
#define  ENC_SPI_CLK_PIN     26
#define  ENC_SPI_STE_PIN     27
#define  ENC_TXEN_PIN        34
#define  ENC_PWREN_PIN       32
```

5.1.3.3 T-Format Encoder Latency Considerations

The T-format encoder used in the laboratory tests is TS 5700 N 8501. It is a 17 bit encoder with a latency of nearly 50 micro seconds. Therefore, it is not possible to perform closed loop control with a system sampling frequency of 20 KHz. This means that neither SINGLE_SAMPLING with 20 KHz PWM frequency nor DOUBLE_SAMPLING with 10 KHz PWM frequency can be implemented.

5.1.4 SDFM

High-resolution, accurate, isolated phase current measurement is vital in automotive traction and servo drive applications, where high-performance torque and motion control are required. The options available for phase-current measurement are to use Hall-effect sensors, flux-gate sensors, current transformers, and shunt resistors. The first three options have inherent galvanic isolation benefits and a high current measurement range, but the linearity, bandwidth, and drift are of lower performance when compared to the shunt-resistor option. Shunt resistors provide a highly linear, high-bandwidth, and cost-effective measurement solution. Reinforced isolation provides both benefits of isolation and high linearity and bandwidth. Isolated shunt-based current measurement is accomplished either by using an isolated amplifier or an isolated delta-sigma modulator.

Figure 5-1 shows the mechanism of phase current sensing in servo drive motors, there are two paths implemented in the design, the high-speed bit stream output from the isolated modulators is filtered by TI's C2000 family real-time controllers that have a built-in sigma-delta filter module (SDFM), allowing the user to fine-tune signal bandwidth and accuracy. The SDFM is a four-channel digital filter designed specifically for current measurement in motor control applications. Each channel can receive an independent delta-sigma modulator bit stream which is processed by four individually programmable digital decimation filters. The filters include a fast comparator for immediate digital threshold comparisons for over-current and undercurrent monitoring. Also, a filter-bypass mode is available to enable data logging, analysis, and customized filtering. The SDFM pins are configured using the GPIO multiplexer. A key benefit of the SDFM is it enables a simple, cost-effective, and safe high-voltage isolation boundary.

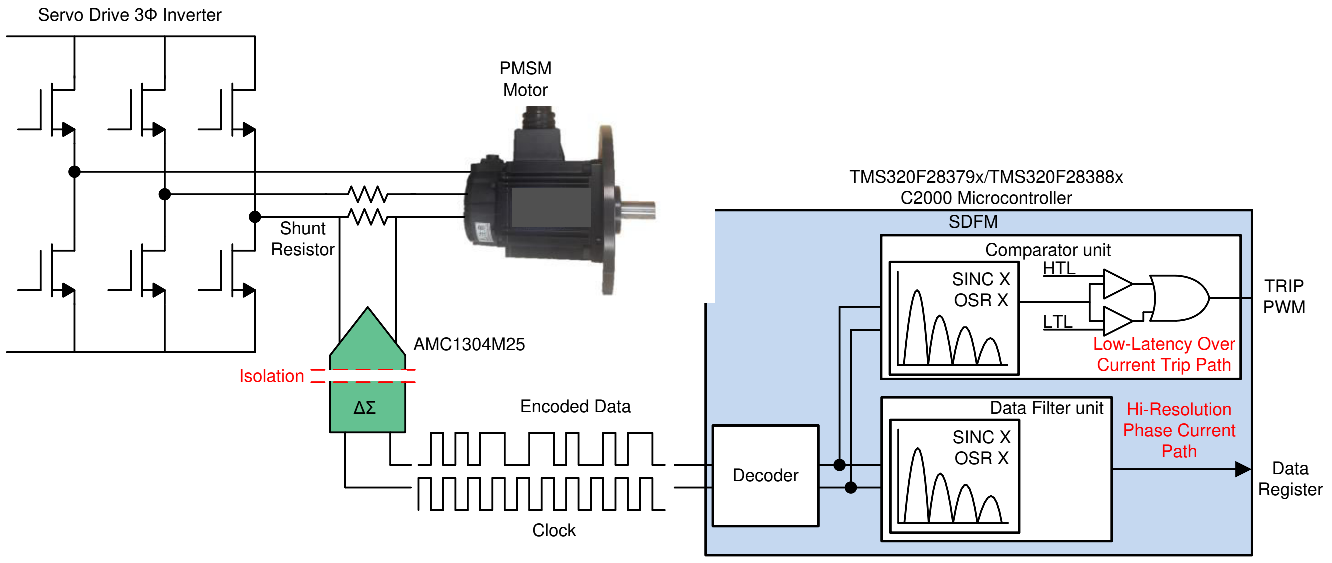


Figure 5-1. Current Sense Using Delta-Sigma Modulator in Servo Drive

On the IDDK board, the voltage across the shunt resistor is fed into the AMC1304M25 sigma-delta modulator, which generates the sigma-delta stream that is decoded by the SDFM demodulator present on the C2000 MCU. The clock for the modulator is generated from the EPWM5 module on the C2000 MCU, and the AMC1304M25 data is decided using the built-in SDFM modulator.

To select the shunt-resistor based current sense with SDFM by setting the configuration in fcl_f2838x_tmdxiddk_settings_cpu1.h as shown below.

```
#define  CURRENT_SENSE       SD_CURRENT_SENSE
```

5.1.5 Incremental System Build

The system is gradually built up through various build levels verifying specific functionality at each level so that at the final level, the system is fully verified and complete in all aspects. Eight levels of incremental system build are designed to verify the various modules used in the system. Levels 1 through 5 build up to a FCL enabled position controlled servo drive. Level 6 helps to perform the frequency response analysis using SFRA. Levels 7 and 8 build up to a connected servo drive performing speed control or position control.

Table 5-1 and Table 5-2 summarizes the core functions integrated and tested at each build level as well as the cores used to perform the tasks in each of the incremental builds.

Table 5-1. Functions Verified in Each Incremental System Build

	Cores Used		
Buildlevel	**CPU1**	**CM**	**Functional Integration / Verification**
LEVEL 1	√		Basic PWM generation
LEVEL 2	√		Open loop control of motor / calibration of feedbacks
LEVEL 3	√		CURRENT MODE - Closing current loop using FCL library
LEVEL 4	√		SPEED MODE - Closing speed loop using inner FCL verified in LEVEL 3
LEVEL 5	√		POSITION MODE - Closing position loop using inner speed loop verified in LEVEL 4
LEVEL 6	√		SFRA ANALYSIS - Performing SFRA on current loop running motor in speed mode (LEVEL 4)
LEVEL 7	√	√	ECAT loop back to TwinCAT - No motor control involved
LEVEL 8	√	√	Fully connected servo drive using TwinCAT to control the motor in speed mode (LEVEL 4) or position mode (LEVEL 5)

Table 5-2. Functional Modules Used in Each Incremental System Build

Software Module	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
PWM Generation	√√	√	FCL lib	FCL lib	FCL lib	FCL lib		FCL lib
QEP Interface in CLA / T-format interface		√√	√	√	√	√		√
FOC functions			√√	√	√	√		√
SFRA functions						√√		
ECAT functions							√√	√

6 Incremental Build Level 1

The block diagram of the system built in BUILD LEVEL 1 is shown in Figure 6-1. During this step, keep the motor disconnected.

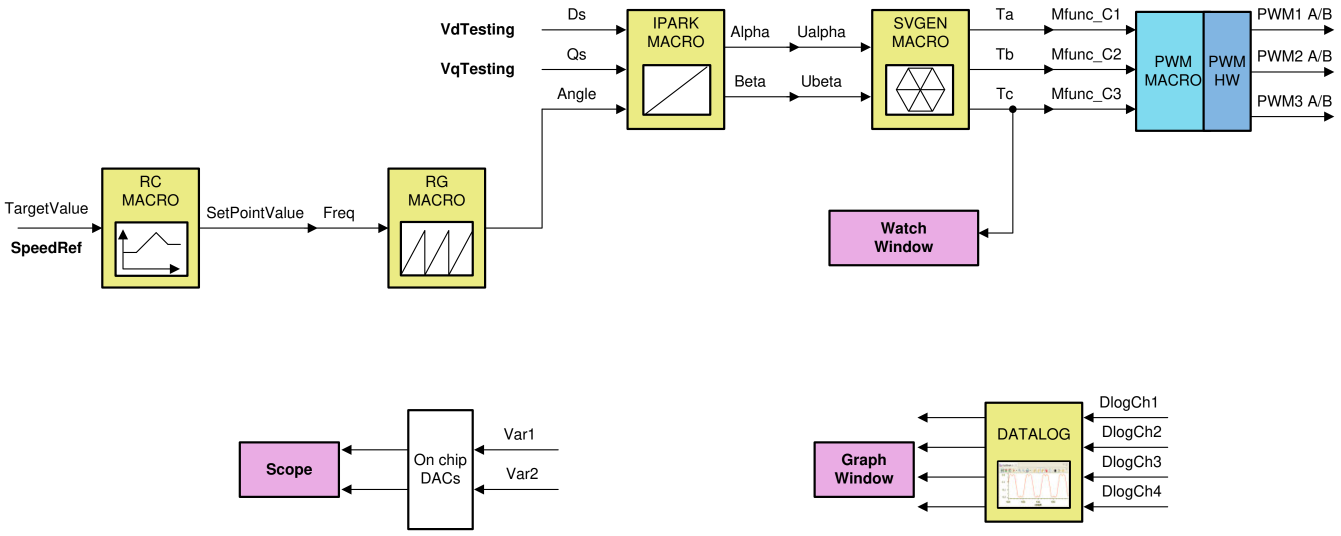
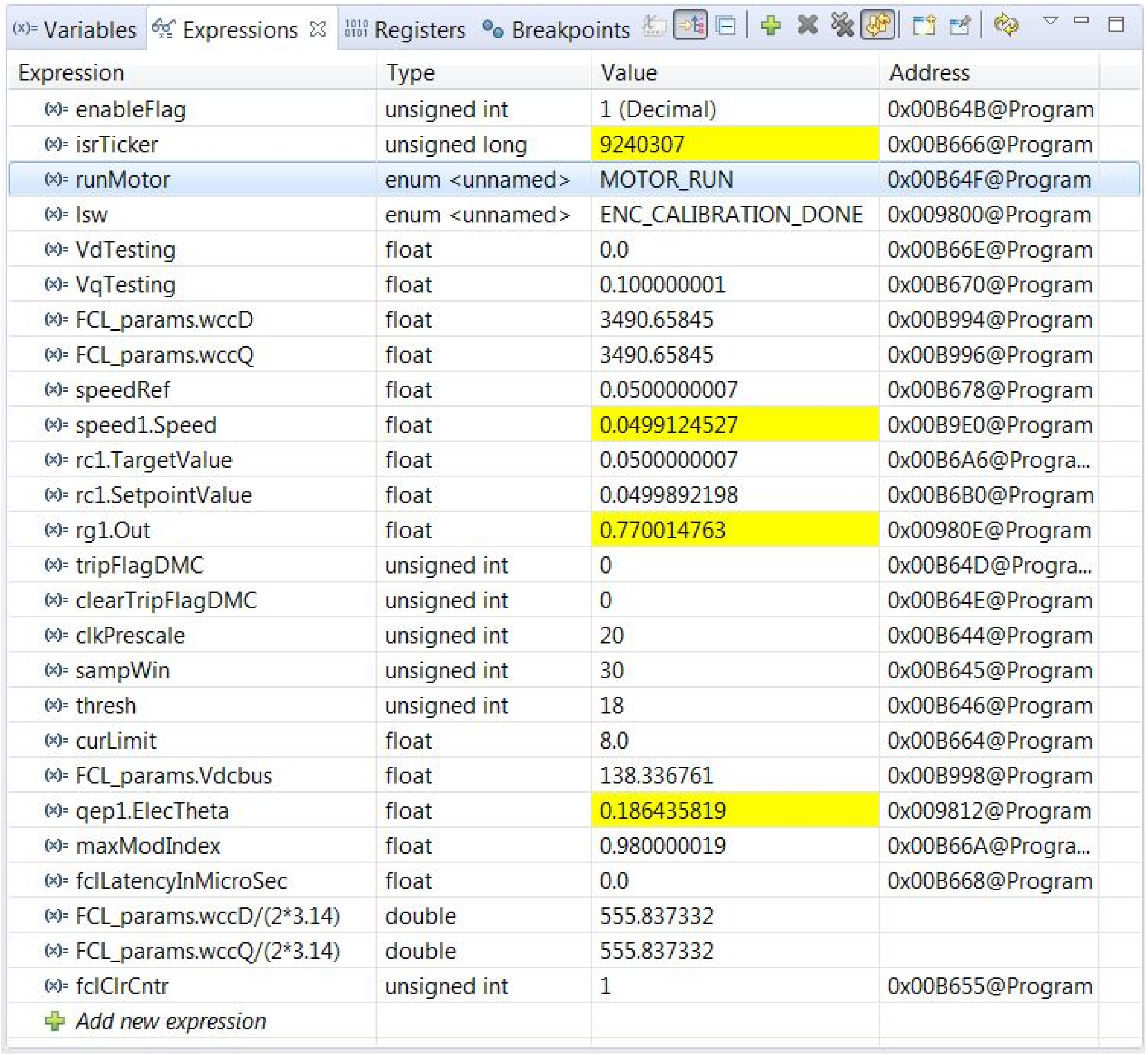


Figure 6-1. Level 1 Block Diagram

Assuming the load and build steps described in the *DesignDRIVE IDDK User Guide* completed successfully, this section describes the steps for a “minimum” system check-out which confirms operation of system interrupt, the peripheral and target independent inverse park transformation and space vector generator modules, and the PWM initializations and update modules.

1. Open *fcl_f2838x_tmdxiddk_settings_cpu1.h* and select the 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.
3. When the build is complete, click on debug button, reset the CPU, restart, enable real time mode, and run.
4. Add variables to the expressions window by right-clicking within the Expressions Window and importing the *fcl_f2838x_tmdxiddk_vars_cpu1.txt* file from the debug directory. Figure 6-2 shows the variables imported into the Expressions Window from this file. Ignore the values shown against the variables for now.



Expression	Type	Value	Address
enableFlag	unsigned int	1 (Decimal)	0x00B64B@Program
isrTicker	unsigned long	9240307	0x00B666@Program
runMotor	enum <unnamed>	MOTOR_RUN	0x00B64F@Program
lsw	enum <unnamed>	ENC_CALIBRATION_DONE	0x009800@Program
VdTesting	float	0.0	0x00B66E@Program
VqTesting	float	0.100000001	0x00B670@Program
FCL_params.wccD	float	3490.65845	0x00B994@Program
FCL_params.wccQ	float	3490.65845	0x00B996@Program
speedRef	float	0.0500000007	0x00B678@Program
speed1.Speed	float	0.0499124527	0x00B9E0@Program
rc1.TargetValue	float	0.0500000007	0x00B6A6@Progra...
rc1.SetpointValue	float	0.0499892198	0x00B6B0@Program
rg1.Out	float	0.770014763	0x00980E@Program
tripFlagDMC	unsigned int	0	0x00B64D@Progra...
clearTripFlagDMC	unsigned int	0	0x00B64E@Program
clkPrescale	unsigned int	20	0x00B644@Program
sampWin	unsigned int	30	0x00B645@Program
thresh	unsigned int	18	0x00B646@Program
curLimit	float	8.0	0x00B664@Program
FCL_params.Vdcbus	float	138.336761	0x00B998@Program
qep1.ElecTheta	float	0.186435819	0x009812@Program
maxModIndex	float	0.980000019	0x00B66A@Progra...
fclLatencyInMicroSec	float	0.0	0x00B668@Program
FCL_params.wccD/(2*3.14)	double	555.837332	
FCL_params.wccQ/(2*3.14)	double	555.837332	
fclClrCntr	unsigned int	1	0x00B655@Program
Add new expression			

Figure 6-2. Expressions Window for Build Level 2

5. 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 working properly.
   In the software, the key variables to be adjusted are:
   a. *speedRef* : for changing the rotor speed in per-unit.
   b. *VdTesting* : for changing the d-qxis voltage in per-unit.
   c. *VqTesting* : for changing the q-axis voltage in per-unit.

6.1 SVGEN Test

The *speedRef* value is fed into the ramp control module to ramp up the speed command. The output of the ramp module is fed into a ramp generator to generate the angle for sinewave generation. This angle as well as the variables *VdTesting* and *VqTesting* feeds into inverse park transformation block which then feeds into space vector modulation modules to generate three phase PWMs.

The outputs from space vector generation module can be viewed using the graph tool from the debug environment by clicking Tools --> Graph --> Dual Time. Then, from the graph window, click Import and browse to and select:

\solutions\tmdxiddk379d\f2838x\debug\fcl_f2838x_tmdxiddk_graph1.graphProp

This plots two graphs representing variables pointed by dlogCh1 and dlogCh2. Likewise, another graph can be opened by selecting:

solutions\tmdxiddk379d\f2838x\debug\fcl_f2838x_tmdxiddk_graph2.graphProp

This plots two graphs representing variables pointed by dlogCh3 and dlogCh4.

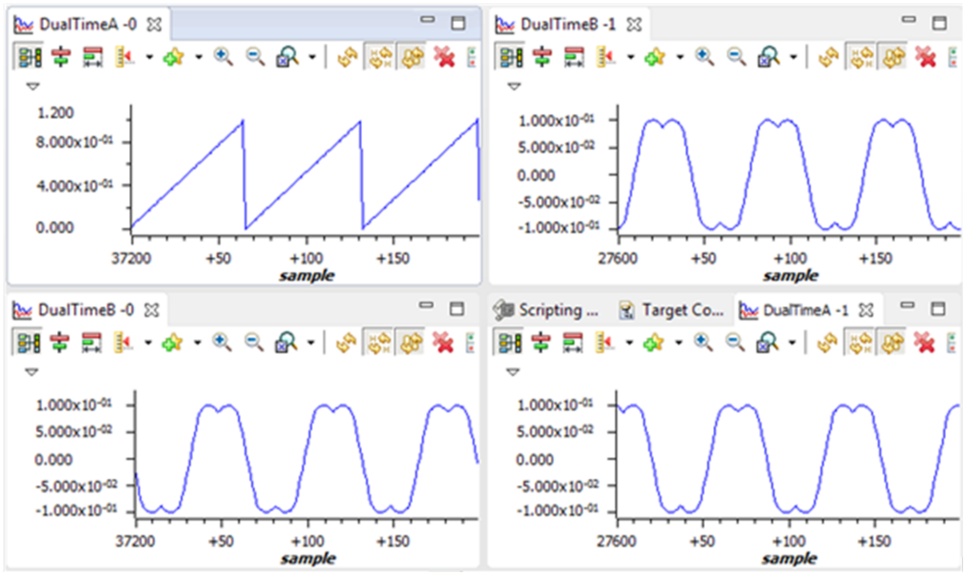


Figure 6-3. Voltage Angle and SVGEN Ta, Tb, and Tc

These are shown in Figure 6-3. These are the voltage vector angle, and the pulse width values for the phases A, B, and C and are denoted as Ta, Tb, and Tc, where Ta, Tb, and Tc waveforms are 120° apart from each other. Specifically, Tb lags Ta by 120° and Tc leads Ta by 120°. These are generated based on the values of *speedRef*, *VdTesting* and *VqTesting*. These values can be changed to see the impact on these waveforms. Check the PWM test points on the board to observe PWM pulses (PWM-1H to 3H and PWM-1L to 3L) and ensure that the PWM module is running properly.

6.2 Testing SVGEN With DACs

To monitor internal signal values in real time, onchip DACs are used. DACs are part of the analog module. DACs B and C are available for this purpose. This is shown in Figure 6-4.

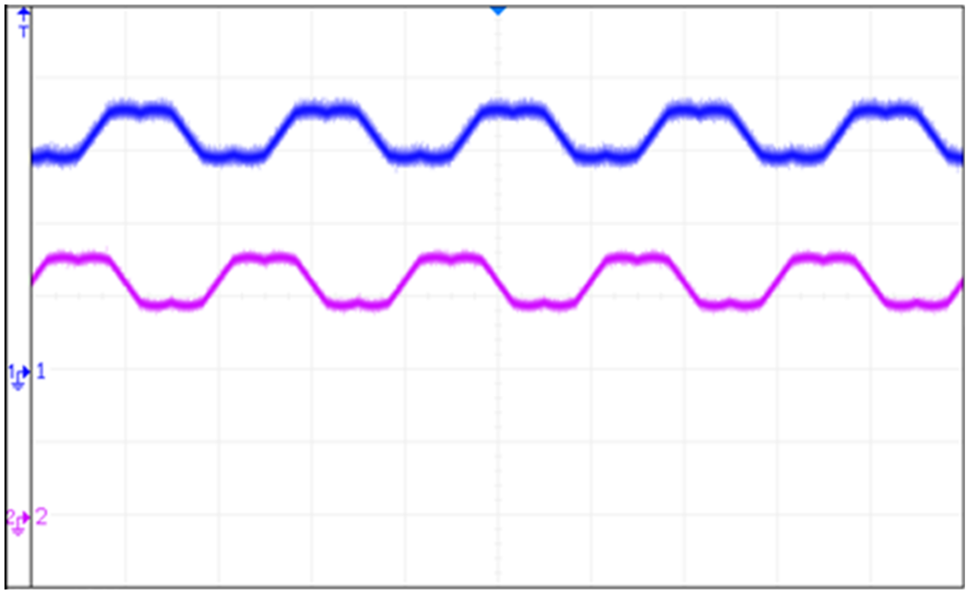


Figure 6-4. DAC Outputs Showing Ta, Tb Waveforms

6.3 Inverter Functionality Verification

After verifying the space vector generation and PWM modules, the 3-phase inverter hardware can be tested by looking at inverter outputs U, V, and W on a scope. This can be compared against the PWM pulses (PWM-1H to 3H) fed into the inverter. It is advisable to gradually increase the DC bus voltage during this test. Check inverter outputs U, V, and W using an oscilloscope with respect to the inverter GND, while taking care of scope isolation requirements. This ensures that the inverter is working properly.

7 Incremental Build Level 2

Assuming build level 1 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 incremental build level 1.

1. Open *fcl_f2838x_tmdxiddk_settings_cpu1.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
3. Select POSITION_ENCODER to QEP_POS_ENCODER or T_FORMAT_ENCODER depending on the encoder coupled to motor.
4. Right-click on the project name and click *Rebuild Project*.
5. When the build is complete, click *Debug*, reset the CPU, restart, enable real-time mode, and run.

   Figure 7-1 shows the level 2 block diagram.

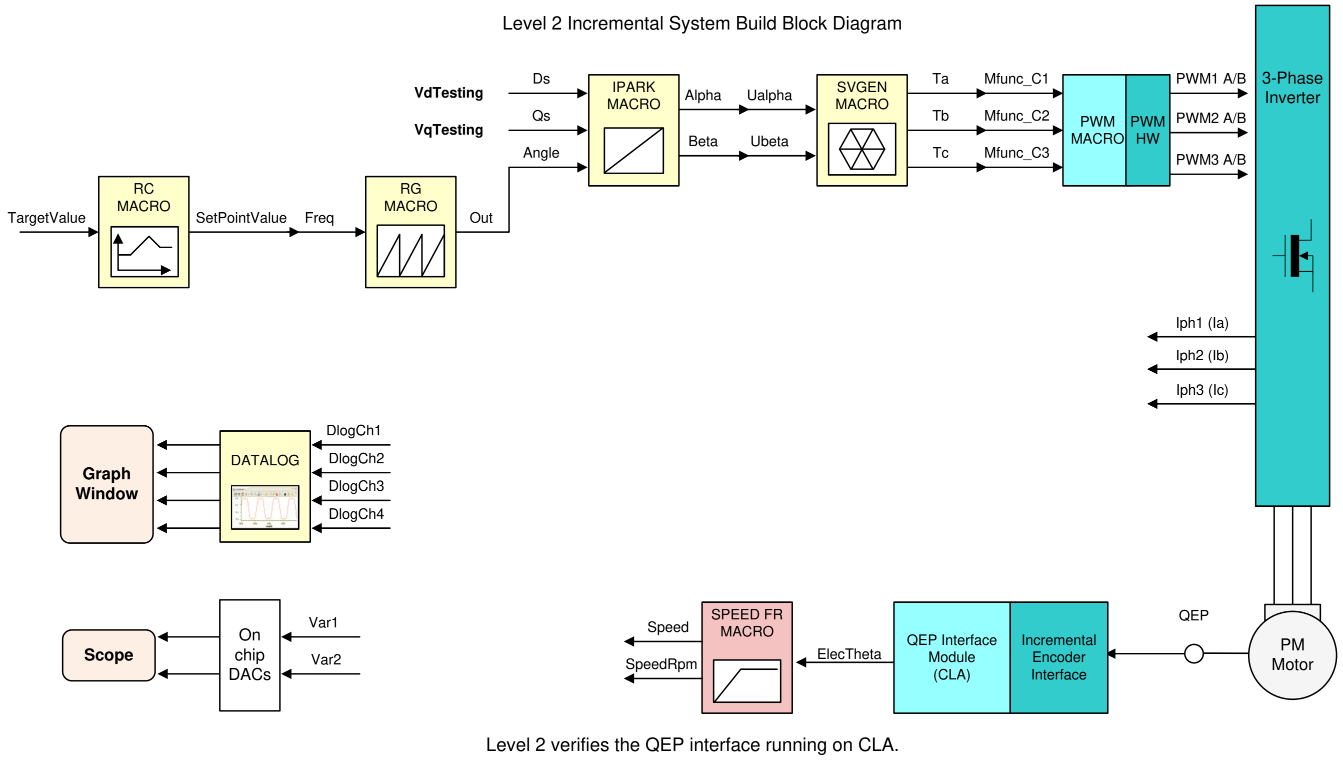


Figure 7-1. Level 2 Block Diagram

6. 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 MOTOR_RUN; 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 are same as in previous level and are given again below for reference.
   a. *speedRef:* for changing the rotor speed in per-unit
   b. *VdTesting:* for changing the d-axis voltage in per-unit
   c. *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.

7.1 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. 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 you 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 8A. Any of these settings can be fine-tuned to suit your system. When 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 currently, but it can be used as an exercise to experiment and learn.

7.2 Current Sense Method

The CURRENT_SENSE method chosen for this project is LEM_CURRENT_SENSE or SD_CURRENT_SENSE. To select the current sense method by setting the configuration in fcl_f2838x_tmdxiddk_settings_cpu1.h as below.

```
#define  CURRENT_SENSE        SD_CURRENT_SENSE
or
#define  CURRENT_SENSE        LEM_CURRENT_SENSE
```

7.3 Voltage Sense Method

Voltage is sensed using the sigma delta filter module 3. Look out for the variable *FCL_params.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.

7.4 Setting Current Regulator Limits

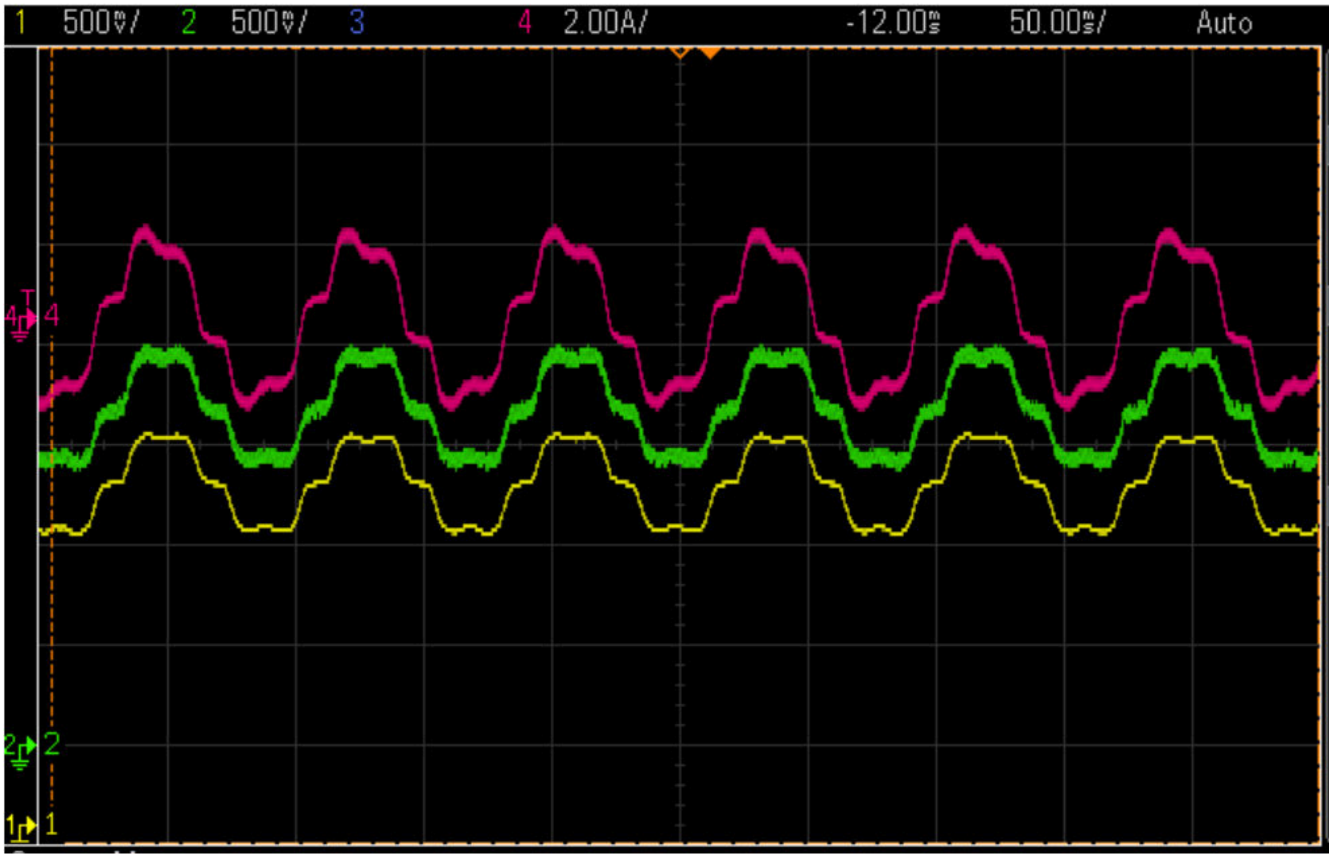
The outputs of the current regulators control the voltages applied on both the d-axis and q-axis. The vector sum of the d and q outputs must be less than 1.0, which refers to the maximum duty cycle for the SVGEN macro. In this particular application, the maximum allowed duty cycle is set to 0.96. Higher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

7.5 Verification of Current Sense

The motor phase currents are measured by LEM sensor using ADC and shunt-resistor using SDFM during motor runs with open-loop control, both sense current waves are displayed on DAC-B and DAC-C. Figure 7-2 shows these signals brought out on H10 on the IDDK, and their scope plots.



CH4: Test Current Wave from Current Probe
CH2: DAC output, Current sense by LEM sensor with ADC
CH1: DAC output, Current sense by Shunt-resistor with SDFM

Figure 7-2. Scope Plot of Measured and Sensing Currents

7.6 Position Encoder Feedback

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 or T-Format) is used, its initial position at electrical angle zero is identified for run-time corrections.

7.6.1 Speed Observer and Position Estimator

The position sensing latency of absolute serial encoders are typically in the order of a few tens of microseconds and will not gel with fast current loop as it looks to apply the control action within 1 µs to 2 µs from the start of feedback sampling instant. To work around this, a position observer cum estimator is used to filter out jitters in position measurement and to estimate the position at the next sampling instant. Accurate estimation depends on knowing when the position information is sampled with respect to the PWM carrier cycle.

Note

In this document, positioning of the position sampling instant is not specifically addressed and is left for you to resolve. Here, position sampling is initiated at the end of motor control algorithm which is a few micro seconds after ADC SOC for current sensing.

7.6.2 Verification of Position Encoder Orientation

Measured or 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 7-3 shows these signals brought out on H10 on the IDDK, and their scope plots.

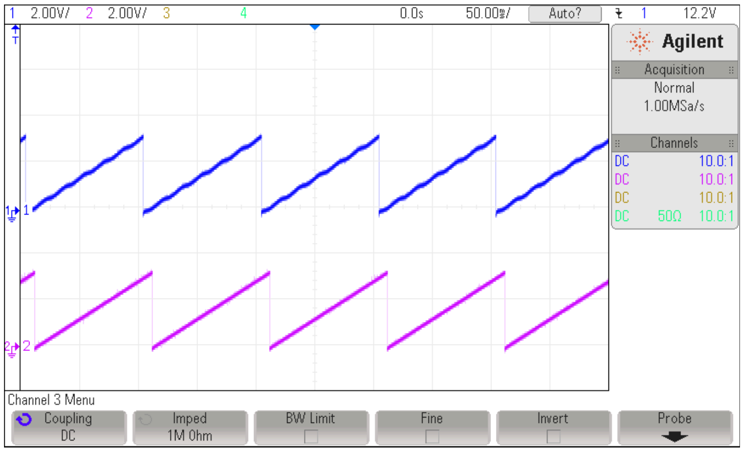
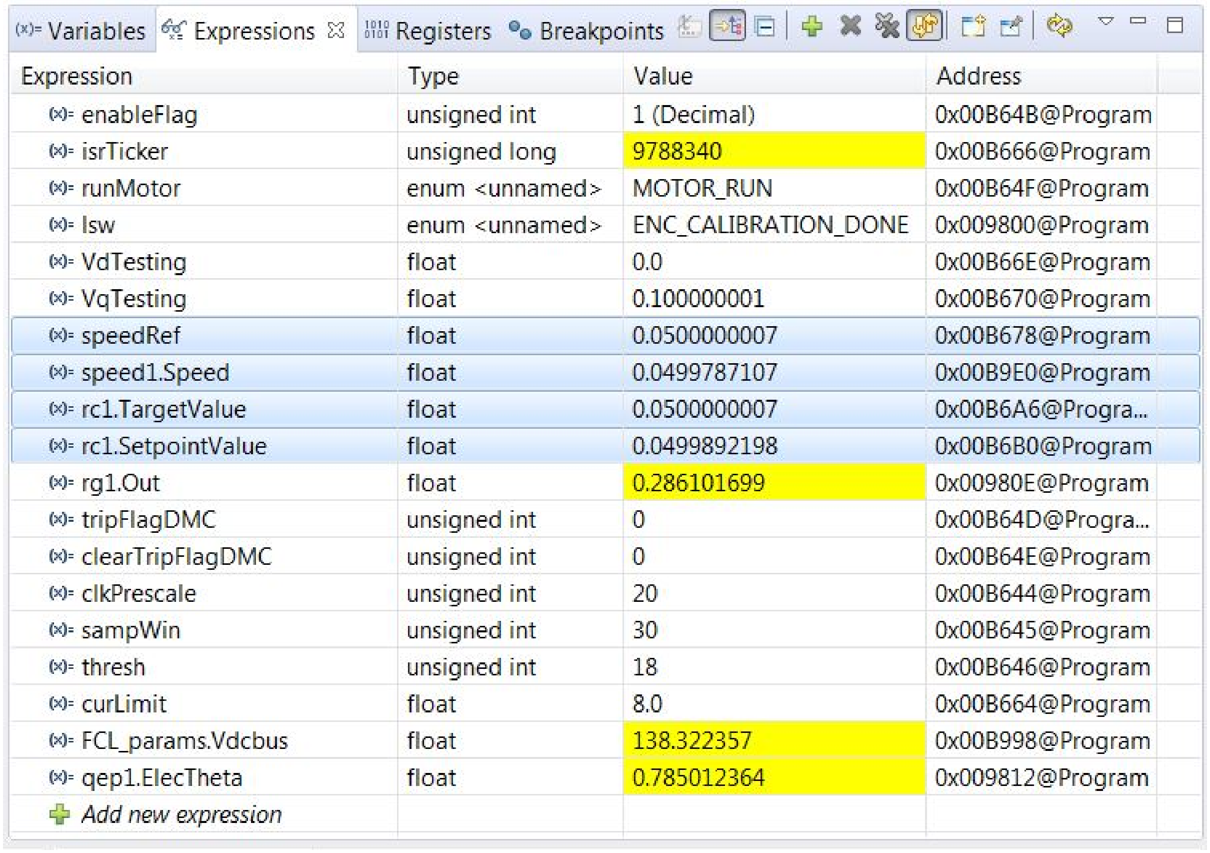


Figure 7-3. Scope Plot of Reference Angle and Rotor Position

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 pseudo code in the software (see Equation 1).

$$\text{angle} = 1.0 - \text{angle} \quad (1)$$

To ensure that the speed estimation is correct, change the *speedRef* variable in the Expressions window, as shown in Figure 7-4, and check whether the estimated speed variable, *speedWe*, 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.


Variables | Expressions | Registers | Breakpoints

Expression	Type	Value	Address
enableFlag	unsigned int	1 (Decimal)	0x00B64B@Program
isrTicker	unsigned long	9788340	0x00B666@Program
runMotor	enum <unnamed>	MOTOR_RUN	0x00B64F@Program
lsw	enum <unnamed>	ENC_CALIBRATION_DONE	0x009800@Program
VdTesting	float	0.0	0x00B66E@Program
VqTesting	float	0.100000001	0x00B670@Program
speedRef	float	0.0500000007	0x00B678@Program
speed1.Speed	float	0.0499787107	0x00B9E0@Program
rc1.TargetValue	float	0.0500000007	0x00B6A6@Progra...
rc1.SetpointValue	float	0.0499892198	0x00B6B0@Program
rg1.Out	float	0.286101699	0x00980E@Program
tripFlagDMC	unsigned int	0	0x00B64D@Progra...
clearTripFlagDMC	unsigned int	0	0x00B64E@Program
clkPrescale	unsigned int	20	0x00B644@Program
sampWin	unsigned int	30	0x00B645@Program
thresh	unsigned int	18	0x00B646@Program
curLimit	float	8.0	0x00B664@Program
FCL_params.Vdcbus	float	138.322357	0x00B998@Program
qep1.ElecTheta	float	0.785012364	0x009812@Program
Add new expression			

Figure 7-4. 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.

8 Incremental Build Level 3

Assuming the previous section is completed successfully, this section verifies the dq-axis current regulation performed by the FCL. One of the two current controllers can be chosen: 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 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 encoder angle count are set to zero.

If the encoder is a QEP, then the QEP count is set to zero. The motor is then forced to run based on an enforced angle until the QEP index pulse is received. Then the motor can run in full self-control mode based on its own angular position.

If the encoder is an absolute type, then the angle at the end of alignment is captured as an initial angle (*initTheta*) that is used as a reference for position determination from then on. The motor can run in full self-control mode based on its own angular position right after the alignment.

The following initial steps can help with evaluating this build level;

1. Open *fcl_f2838x_tmdxiddk_settings_cpu1.h* and select the level 3 incremental build option by setting BUILDLEVEL to FCL_LEVEL3 (#define BUILDLEVEL FCL_LEVEL3).
2. The current loop regulator can be selected to be PI controller or complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR.
3. Select CURRENT_SENSE to LEM_CURRENT_SENSE
4. Select POSITION_ENCODER to QEP_POS_ENCODER or T_FORMAT_ENCODER depending on the encoder coupled to motor.
5. 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 single sampling method, and to carrier maximum and carrier zero in double sampling method. This sample method selection is done in the example by selecting SAMPLING_METHOD to SINGLE_SAMPLING or DOUBLE_SAMPLING. 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. If a T-format encoder is used, select SINGLE_SAMPLING and no more than 10KHz PWM frequency. For more information, see Section 5.1.3.3.
6. 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_params.WccD* : preferred bandwidth of d-axis current loop
- *FCL_params.WccQ* : preferred bandwidth of q-axis current loop
- *fclLatencyInMicroSec* : shows latency between ADC sampling and PWM update in µs
- *fclClrCntr* : flag to clear the variable *fclLatencyInMicroSec* and let it refresh
- *runMotor* : flag to RUN or STOP the motor

Figure 8-1 shows the level 3 block diagram.

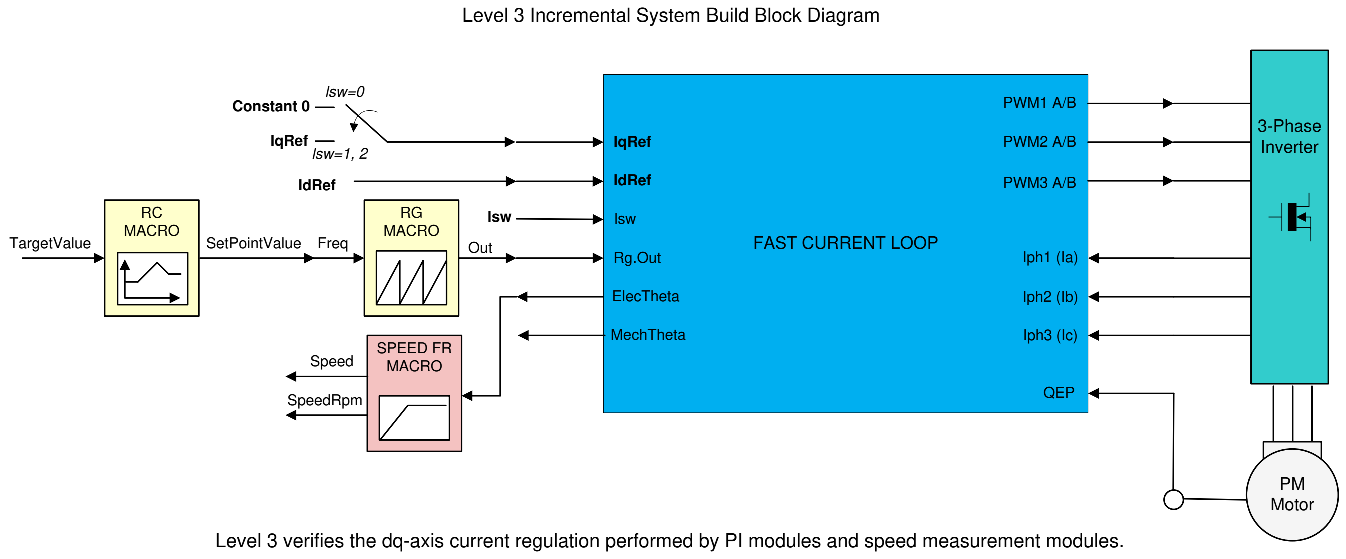


Figure 8-1. Level 3 Block Diagram Showing Inner Most Loop - FCL

The key steps are explained as follows:

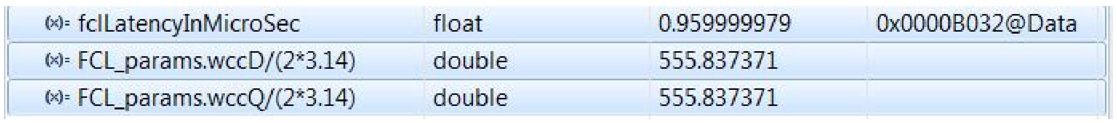
1. Set *enableFlag* to 1 in the watch window. The *isrTicker* variable is incrementally increased, as seen in watch windows to confirm the interrupt is working properly.
2. Verify if the *maxModIndex* value is either 0.96 in double-sampling method or 0.98 in single-sampling method.
3. Set *speedRef* 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). With a QEP encoder, *speedRef* helps to run the motor in forced mode only until the QEP index pulse is received. Thereafter, the motor is controlled based on its rotor position. Whereas, with an absolute encoder, there is no such forced mode and the motor runs based on rotor position right after alignment.
4. Gradually increase voltage at variac / DC power supply to, for example, 20% of the rated voltage.
5. *Isw* is a state machine variable representing the functional state of interacting with quadrature encoder. Its various states are as follows:
   a. *Isw* = ENC_ALIGNMENT --> lock the rotor of motor into alignment with stator phase A
   b. *Isw* = ENC_WAIT_FOR_INDEX --> motor in run mode and waiting for the first instance of QEP index pulse (applicable with QEP encoder only)
   c. *Isw* = ENC_CALIBRATION_DONE --> motor in run mode - (signifies occurence of QEP index pulse and completion of calibration when QEP is used, or, completion of calibration when other encoders are used)
6. Set *runMotor* flag to MOTOR_RUN to run the motor.
7. Notice that *Isw* state is autopromoted in a sequence (code is inside the FCL library)
8. Check *pi_id.fbk* in the watch windows with the continuous refresh feature and see if it can track *IdRef*.
9. Check *pi_iq.fbk* in the watch windows with the continuous refresh feature and see if it can track *IqRef*.
10. 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.
11. Try different bandwidths for the current loop by tweaking the values of *FCL_params.wccD* and *FCL_params.wccQ*. The default setting for the bandwidth is 1/18 of the sampling frequency.
12. 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.
13. 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.

8.1 Observation One – PWM Update Latency

8.1.1 From the Expressions Window

While running the motor in this build level and subsequent build levels, observe the variable *fclLatencyInMicroSec* in the Expressions window.

Figure 8-2 shows a snapshot of the Expressions window.



fclLatencyInMicroSec	float	0.959999979	0x0000B032@Data
FCL_params.wccD/(2*3.14)	double	555.837371	
FCL_params.wccQ/(2*3.14)	double	555.837371	

Figure 8-2. 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. Immediately after setting the *runMotor* flag to MOTOR_RUN and the motor begins to run, the latency time shows up as nearly 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 is therefore acceptable. After this period, you can refresh the latency time by setting *fclClrCntr* 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 0.96 μs compared to 0.98 μs with a CMPLX_CNTLR (see the following note).

Note

- These times can be reduced further by around 0.1 μs range using **code inline** and other optimization techniques. Because the evaluation code is in library format, it has certain overheads.
- The **sampling window** for ADC is kept wide enough to ensure a cleaner signal acquisition. Depending on board layout and circuits feeding in to ADC channels, it may be possible to reduce this time window by nearly 60%.

8.1.2 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, comment out the associated code and restore the functional assignment.

Figure 8-3 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 *fclLatencyInMicroSec*. This additional time is needed for the CPU to return from FCL library and to set GPIO .

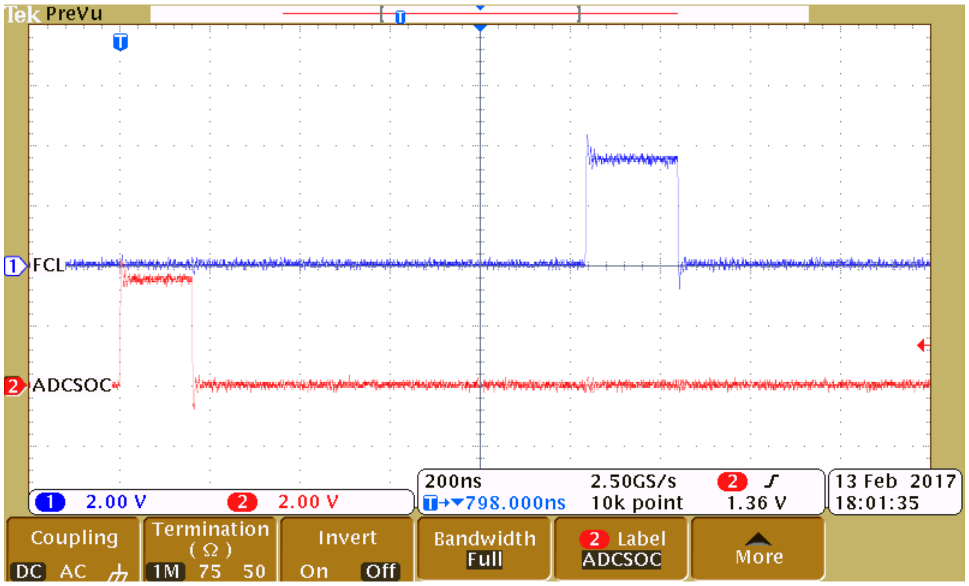


Figure 8-3. Scope Plot of ADCSoC and FCL Completion Events

9 Incremental Build Level 4

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 position encoder angle count are set to zero. After ensuring a stable alignment, the motor starts running.

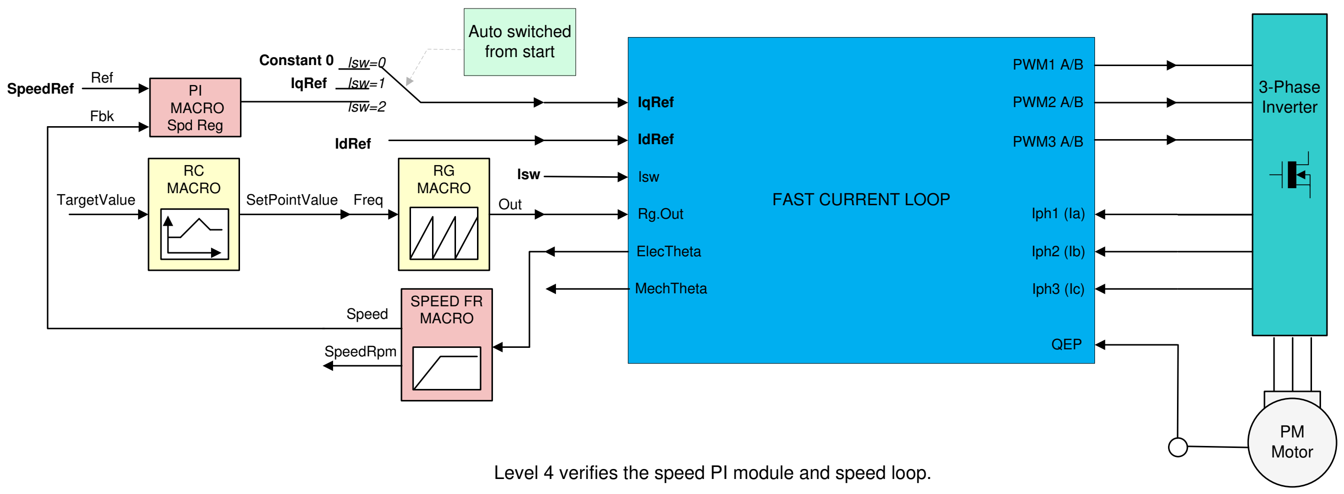
1. Open *fcl_f2838x_tmdxiddk_settings_cpu1.h* and select the level 4 incremental build option by setting the BUILDLEVEL to FCL_LEVEL4 (#define BUILDLEVEL FCL_LEVEL4).
2. The current loop regulator can be selected to be PI controller or complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR.
3. Select CURRENT_SENSE to LEM_CURRENT_SENSE
4. Select POSITION_ENCODER to QEP_POS_ENCODER or T_FORMAT_ENCODER depending on the encoder coupled to motor.
5. Select SAMPLING_METHOD to SINGLE_SAMPLING or DOUBLE_SAMPLING. If a T-format encoder is used, select SINGLE_SAMPLING and no more than 10KHz PWM frequency. For more information, see Section 5.1.3.3.
6. 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 9-1 shows the implementation block diagram.

Level 4 Incremental System Build Block Diagram



Level 4 verifies the speed PI module and speed loop.

Figure 9-1. 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. Set *runMotor* to MOTOR_RUN
6. Notice that the state machine variable (*lsw*) is auto-promoted in a sequence, its states are as follows:
   a. *lsw* = ENC_ALIGNMENT --> lock the rotor of motor into alignment with stator phase A
   b. *lsw* = ENC_WAIT_FOR_INDEX --> motor in run mode and waiting for the first instance of QEP index pulse (applicable with QEP encoder only)
   c. *lsw* = ENC_CALIBRATION_DONE --> motor in run mode - (signifies occurence of QEP index pulse and completion of calibration when QEP is used, or, completion of calibration when other encoders are used)
7. Compare the speed *(speed1.Speed)* with *speedRef* in the watch windows with the continuous refresh feature to see whether or not it is nearly the same.
8. 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.
9. At a very low speed range, the performance of the speed response relies heavily on the accuracy of rotor position angle provided by position encoder.
10. 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 9-2 shows flux and torque components of the stator current in the synchronous reference frame.

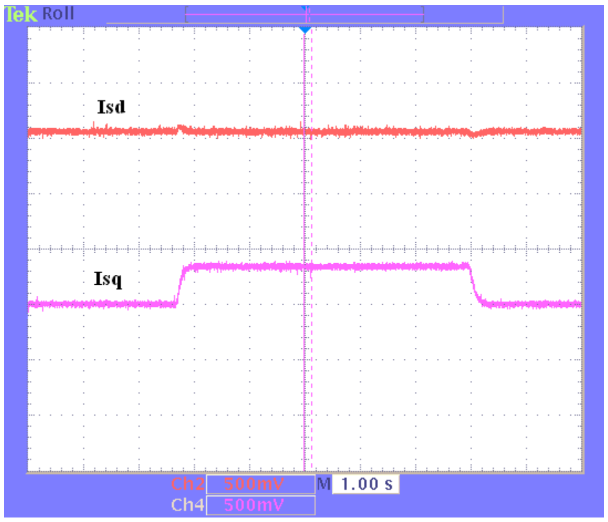


Figure 9-2. Flux and Torque Components of the Stator Current in the Synchronous Reference Frame Under 0.33-pu Step-Load and 0.3-pu Speed

9.1 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, see the note below. The gain cross over frequency (or open loop bandwidth) can be taken up to 1/6 of the SAMPLING_FREQUENCY and still get good transient response over the entire speed range including higher speeds. If the motor rotation direction is reversed occasionally due to any malfunction, try restarting it by setting *runMotor* to MOTOR_STOP and then MOTOR_RUN again. With a QEP, it may need some fine tuning in transitioning from *lsw = ENC_WAIT_FOR_INDEX* to *lsw = ENC_GOT_INDEX*. This can be used as an exercise to fix it.

Note

The fast current loop is a high bandwidth enabler. When the designed bandwidth is high, the loop gains can also be high. This pretty much ties the loop performance to the quality of current feedback. If the SNR of current feedback signal into the digital domain is poor, then the loop can be very audibly noisy as the controller tries to minimize the error. If the noise is bothersome, you may be required to reduce the bandwidth to avoid the audible noise.

10 Incremental Build Level 5

This section verifies the position PI module and position loop. 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 position encoder angle count are set to zero. After ensuring a stable alignment, the motor starts to run.

1. Open *fcl_f2838x_tmdxiddk_settings_cpu1.h* and select the level 5 incremental build option by setting the BUILDLEVEL to FCL_LEVEL5 (#define BUILDLEVEL FCL_LEVEL5).
2. The current loop regulator can be selected to be PI controller or complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR.
3. Select CURRENT_SENSE method to LEM_CURRENT_SENSE.
4. Select POSITION_ENCODER to QEP_POS_ENCODER or T_FORMAT_ENCODER depending on the encoder coupled to motor.
5. Select SAMPLING_METHOD to SINGLE_SAMPLING or DOUBLE_SAMPLING. If a T-format encoder is used, select SINGLE_SAMPLING and no more than 10 KHz PWM frequency. For more information, see Section 5.1.3.3.
6. 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.

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 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 MOTOR_RUN 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).
   a. The motor runs 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.

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 returned 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 10-1 shows the implementation block diagram.

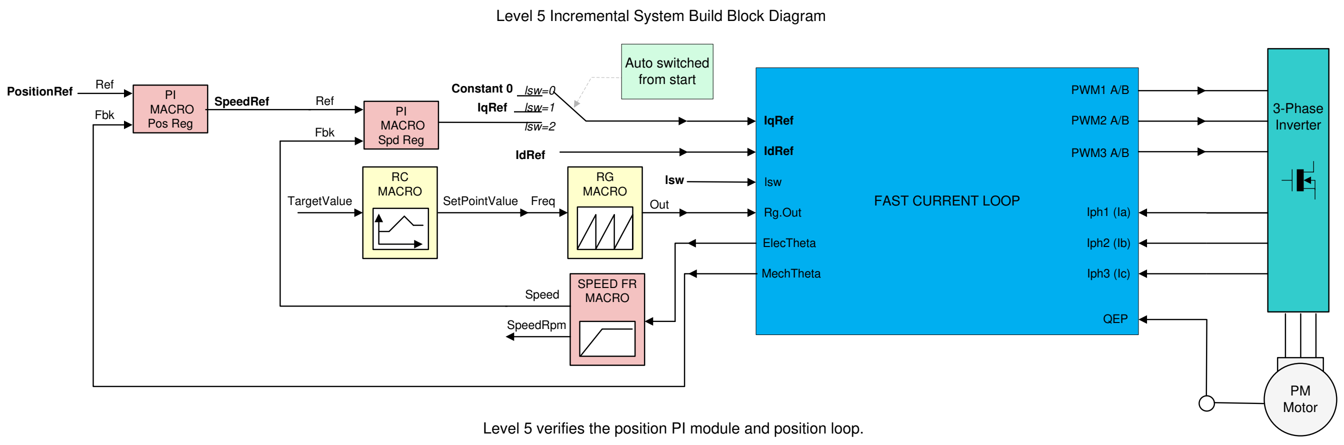


Figure 10-1. Level 5 Block Diagram Showing Position Loop With Inner FCL

In the scope plot shown in Figure 10-2, the position reference, and position feedback are plotted. 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.

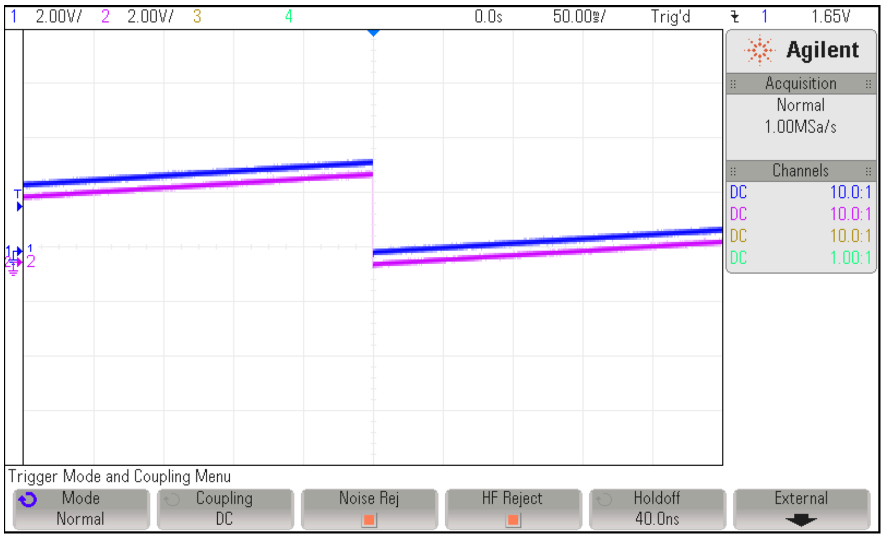


Figure 10-2. Scope Plot of Reference Position to Servo and Feedback Position

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. With an absolute encoder, this may be obvious.

11 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 DigitalPower SDK.

11.1 Integrating SFRA Library

C2000™ Software Frequency Response Analyzer (SFRA) Library and Compensation Designer in SDK Framework describes the SFRA tool and guides you on how to integrate it into the C2000 platform can be found at:

C:\ti\c2000\C2000Ware_DigitalPower_SDK_<version>\libraries\sfra\Doc

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

C:\ti\c2000\C2000Ware_DigitalPower_SDK_<version>\libraries\sfra

The SFRA GUIs are available as executable applications in the DigitalPower SDK at:

C:\ti\c2000\C2000Ware_DigitalPower_SDK_<version>\libraries\sfra\gui

Some example projects to understand SFRA are available at:

c:\ti\c2000\C2000Ware_DigitalPower_SDK_<version>\libraries\sfra\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:

- *injectSFRA()*
- *collectSFRA()*

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.

Note

- The disturbances due to analog signal path and quantization will impact the loop performance and hinder high bandwidth selections that can be verified using SFRA results. Therefore it is important to provide a current feedback with a higher SNR.
- When evaluating current loops, if it is possible to hold the motor speed constant, it will help to minimize the impact of speed jitter related errors in the SFRA results. This is particularly useful when studying the Iq loop. If the voltage decoupling of current loop is good, then this requirement may not matter.

This tool provides the ability to study the D-axis or Q-axis current loops or the speed loop. 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 can be seen that the controller can provide the designed bandwidth under all these conditions with a certain tolerance.

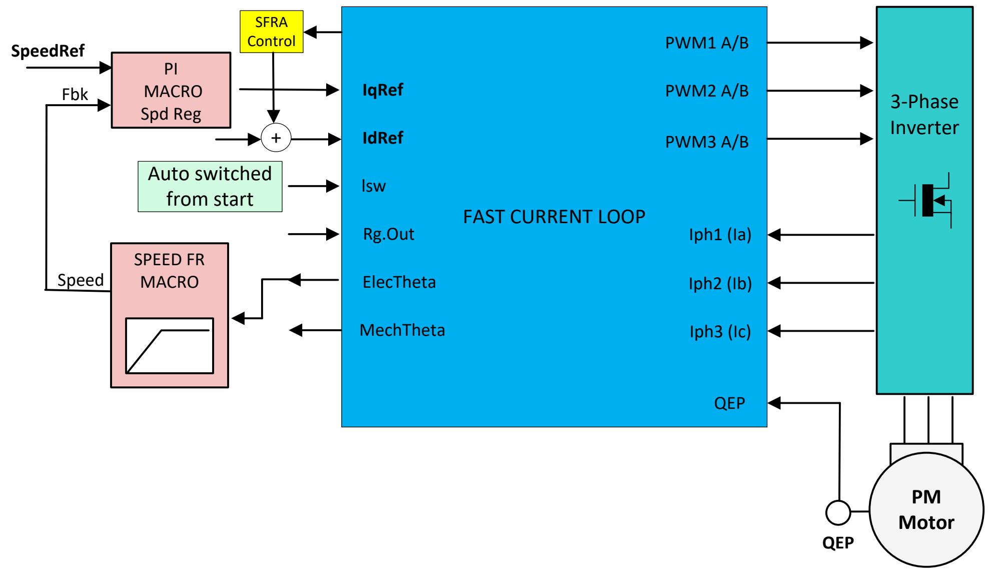


Figure 11-1. Level 6 Block Diagram

The implementation block diagram is given in Figure 11-1. 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.

11.2 Initial Setup Before Starting SFRA

The setting up involves co-ordination between the debug environment and the SFRA GUI. Until getting familiar with connecting the SFRA GUIs to the target platform, it is a good idea to turn off the high voltage power input to the target platform.

The following initial steps can help with evaluating this build level:

1. Open '*fcl_f2838x_tmdxiddk_settings_cpu1.h*' and select level 6 incremental build option by setting the BUILDLEVEL to FCL_LEVEL6 (#define BUILDLEVEL FCL_LEVEL6).
2. The current loop regulator can be selected to be PI controller or complex controller by setting FCL_CNTLR to PI_CNTLR or CMPLX_CNTLR.
3. Select CURRENT_SENSE method to LEM_CURRENT_SENSE.
4. Select POSITION_ENCODER to QEP_POS_ENCODER or T_FORMAT_ENCODER depending on the encoder coupled to motor.
5. Select SAMPLING_METHOD to SINGLE_SAMPLING or DOUBLE_SAMPLING. If a T-format encoder is used, select SINGLE_SAMPLING and no more than 10 KHz PWM frequency. For more information, see Section 5.1.3.3.

Open '*fcl_f2838x_sfra_settings_cpu1.h*' and 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.

In this motor control project, SFRA can be performed on any of the three control loops such as the speed loop, D axis current loop and Q axis current loop. Technically, this could be performed on position loop as well, but is not included in this project scope and you can take it as an experiment, if desired.

1. 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. From the debug environment, the steps to be followed are shown below. Add the following variable in the ‘Expressions Window’:
   a. sfraTestLoop : for selecting the control loop on which to evaluate SFRA, letting you choose between:
      i. SFRA_TEST_D_AXIS - D axis current loop
      ii. SFRA_TEST_Q_AXIS - Q axis current loop
      iii. SFRA_TEST_SPEEDLOOP - speed loop

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. The SCI initialisations needed to communicate with the GUIs should be all done by now.
3. Further steps with the debug window will follow after setting up the GUIs to connect to target platform.

11.3 SFRA GUIs

There are two GUIs available to perform the frequency response analysis, one (SFRA_GUI) to plot open loop and plant Bode diagram, and another (SFRA_GUI_MC) to plot open loop and closed loop Bode diagram. They can be invoked and connected to the target platform to study the control loops. The GUI executables are available in the location as mentioned in Section 11.1.

Double click on the choice of GUI executable and the GUI screen will appear as shown in Figure 11-2 for SFRA_GUI or in Figure 11-3 for SFRA_GUI_MC. They look almost identical, the difference is in the pull down menu under 'FRA Settings' starting with the label 'Open Loop'.

- In the SFRA_GUI, this pull down menu helps to select between Open Loop Model and Plant Model.
- In the SFRA_GUI_MC, this pull down menu helps to select between Open Loop Model and Closed Loop Model.

Note

The GUIs interpretation of bandwidth is different.

- In the SFRA_GUI, bandwidth is defined as the open loop gain cross over frequency
- In SFRA_GUI_MC, bandwidth is defined according to Chinese standard **GBT 16439-2009** and **NEMA ICS 16 (Speed Loop)** for servo drives. It defines bandwidth as the frequency where the closed loop output magnitude drops by 3dB or the phase shift lag exceed 90° for the speed loop. It is suggestive that bandwidth is the frequency where the first of these two instances would occur. This approach is used to analyze current loops as well in this demo.

This demo uses the GUI 'SFRA_GUI_MC'. However, you are encouraged to experiment with the other GUI as well to study the plant. With the SFRA_GUI, you can plot the same graph as in SFRA_GUI_MC by passing the argument 'SFRA_GUI_PLOT_GH_CL' in the function *configureSFRA()*, as shown in the code snippet below.

```
configureSFRA(SFRA_GUI_PLOT_GH_CL, ISR_FREQUENCY);    // to plot GH and CL plots using SFRA_GUI
```

But the inferences from the plots are not according to that in SFRA_GUI_MC. Therefore, it is advised to configure SFRA for 'SFRA_GUI_PLOT_GH_H' so that you can see open loop, closed loop and plant model plots using these two GUIs, by using one at a time. For digital power applications, SFRA_GUI_MC inferences may not apply and so SFRA_GUI can be used to display closed loop plots as well.

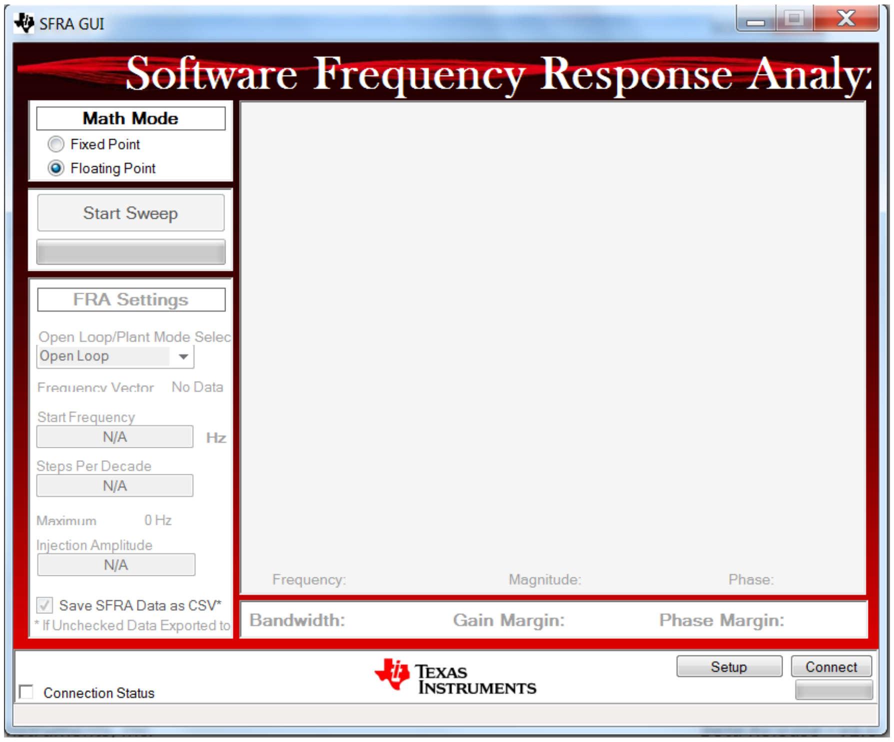


Figure 11-2. SFRA GUI

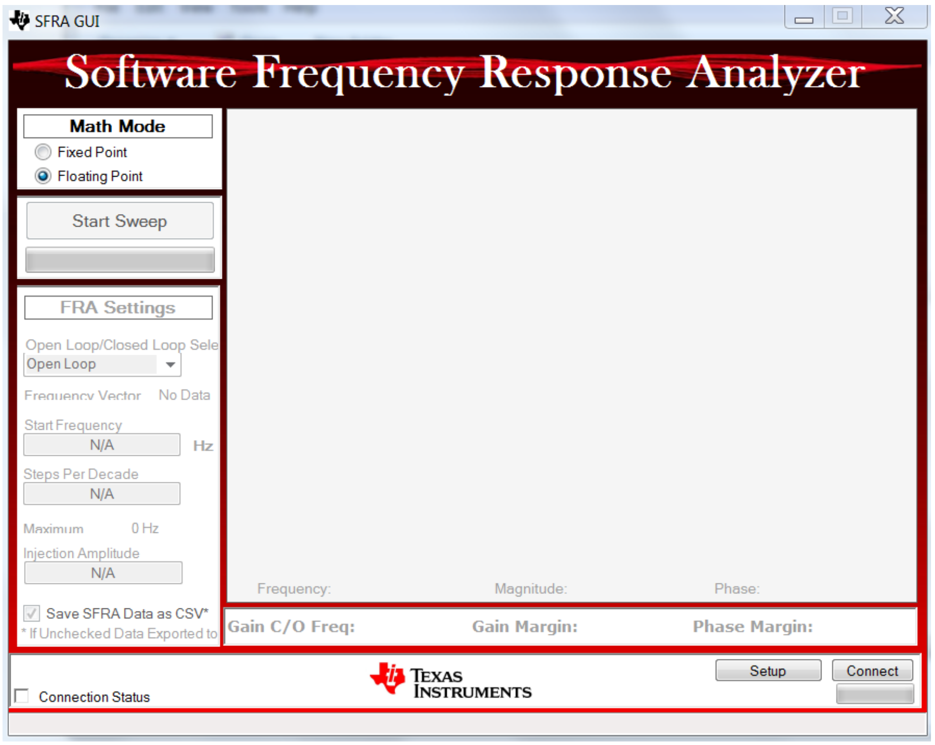


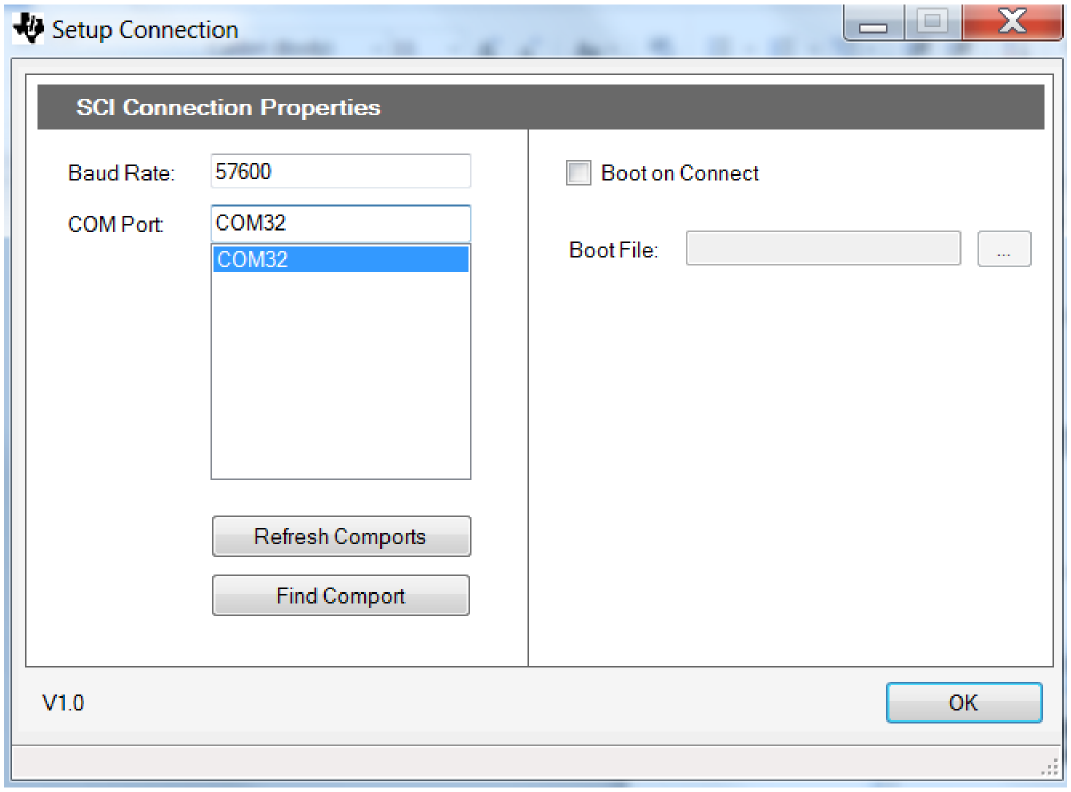
Figure 11-3. SFRA GUI MC

11.4 Setting Up the GUIs to Connect to Target Platform

Both GUIs have identical procedures to connect to the target platform. The GUI lets you select appropriate settings based on the target platform and development computer.

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 F2837x or later C2000 MCUs, select the 'Floating Point' option
2. Since the USB port on the control card / launch pad is already connected to the computer for JTAG purposes, no additional connection is required. However, for a standalone operation, an USB connector needs to be connected to the target board. In the XDS100 emulator present on the control card / launch pad, in addition to a JTAG link, an SCI port link is also provided and the GUI uses this link to connect to the SCI port of the target platform. 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 11-4.



Baud Rate: 57600

COM Port: COM32

COM32

☐ Boot on Connect

Boot File:

Figure 11-4. 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 MCU control card/launchpad and the GUI will appear as shown in Figure 11-5 indicating the connection status at the bottom left corner.

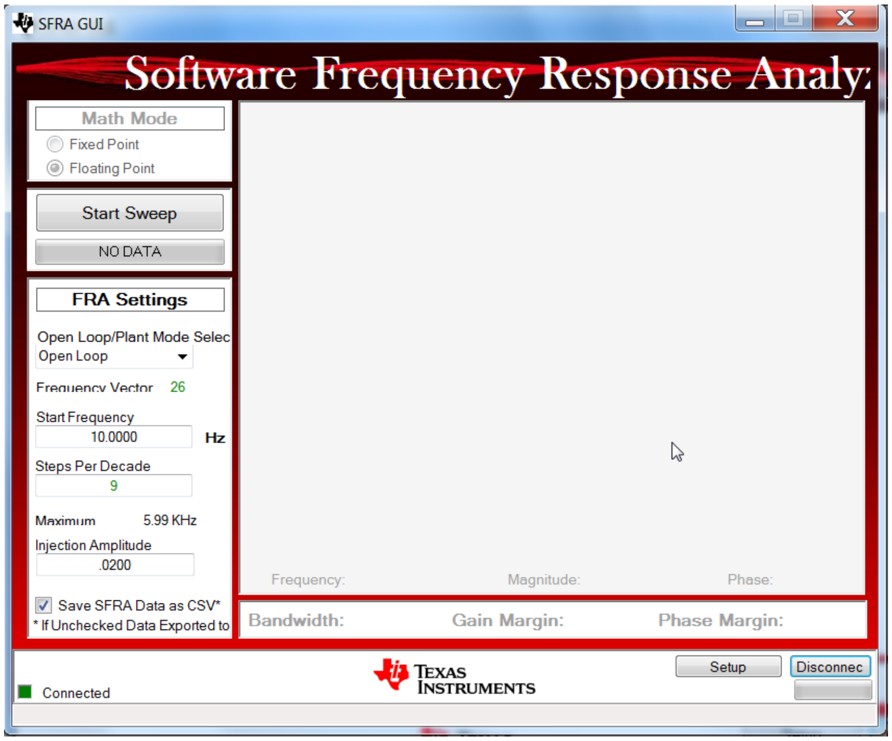


Figure 11-5. SFRA GUI Connected to the C2000 MCU

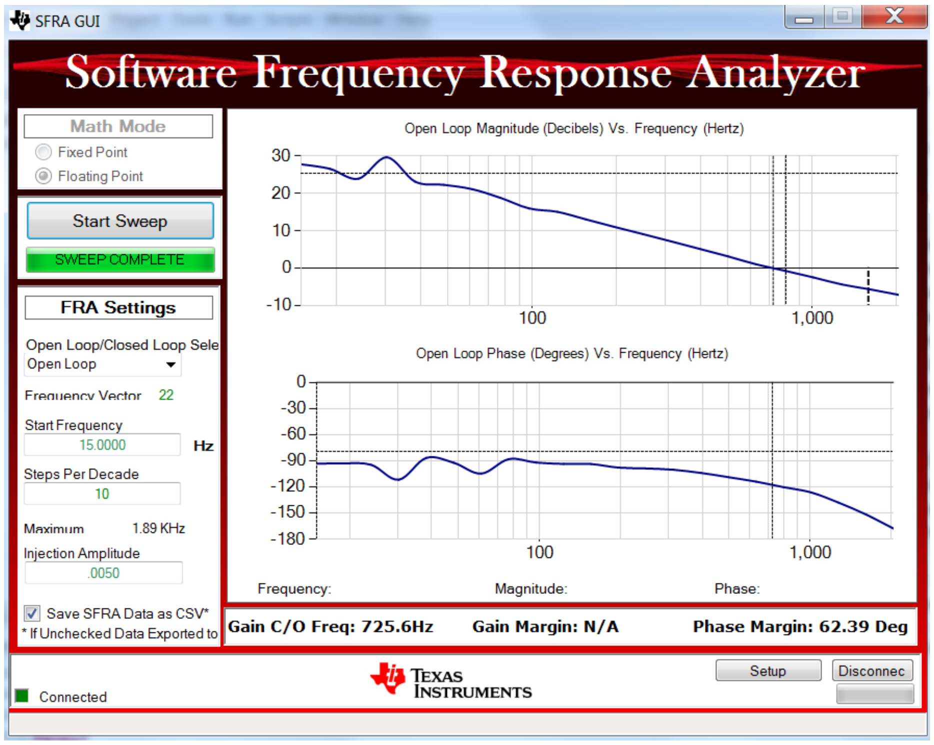
9. The frequency sweep related settings are shown in ‘FRA Settings’ Panel. These values are already pre-filled from the C2000 device and they can be left as is.
10. As mentioned earlier, the visual difference between the two GUIs is in the pull down menu under 'FRA Settings' starting with Open Loop.
    a. In the SFRA_GUI, this pull down menu helps to select between Open Loop Model and Plant Model
    b. In the SFRA_GUI_MC, this pull down menu helps to select between Open Loop Model and Closed Loop Model
    c. This menu becomes relevant after a complete noise injection sweep of the system at various frequencies. Then, you can pick and view the plot of choice using this menu.
    d. Bandwidth reporting is different as mentioned earlier. Gain cross over frequency is reported in the open loop plot of the SFRA_GUI_MC instead of bandwidth as in SFRA_GUI.

This completes the initial setup of GUI environment.

11.5 Running the SFRA GUIs

If the high voltage (HV) power input to the target platform is turned off before, restore it back now. From the debug environment, the steps to be followed are as shown below:

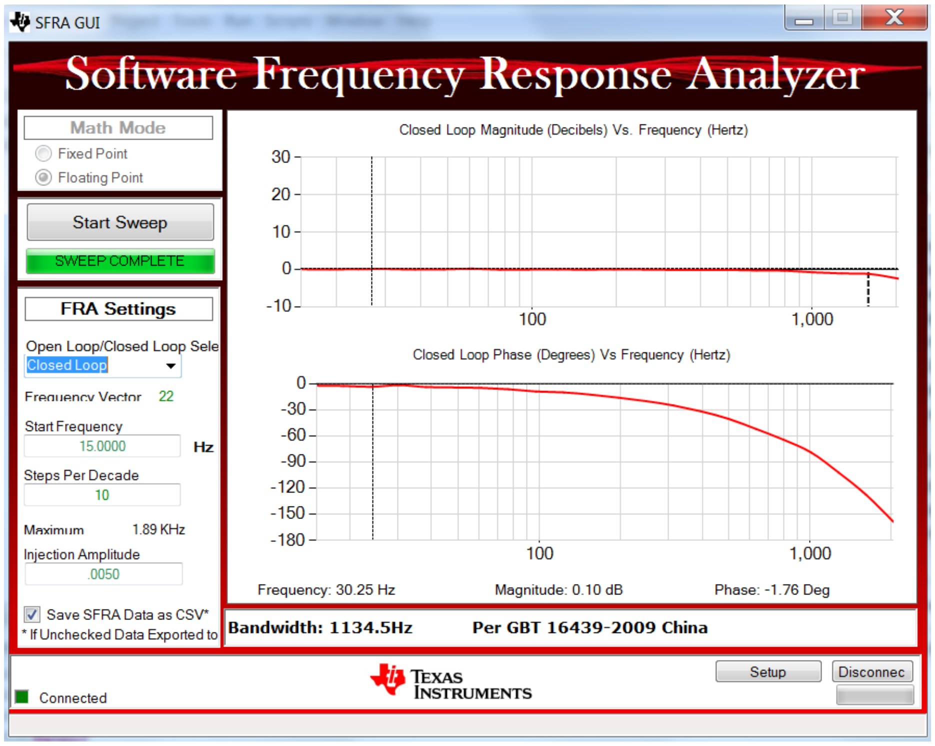
1. Verify that *sfraTestLoop* is set to *SFRA_TEST_D_AXIS* so as to test Id loop.
2. Set *FCL_params.wccD* to the desired value, within limits, (when test is performed for Q axis, adjust this parameter for Q axis - *FCL_params.wccQ*)
3. Set *speedRef* = 0.05 (in pu, 1 pu = 250Hz) and then set *runMotor* = MOTOR_RUN to run the motor. Now motor shaft should start spinning and settle at the commanded speed.
4. The state machine variable (*lsw*) is auto-promoted in a sequence, its states are as follows:
   a. *lsw* = ENC_ALIGNMENT --> lock the rotor of motor into alignment with stator phase A
   b. *lsw* = ENC_WAIT_FOR_INDEX --> motor in run mode and waiting for the first instance of QEP index pulse (applicable with QEP encoder only)
   c. *lsw* = ENC_CALIBRATION_DONE --> motor in run mode - (signifies occurence of QEP index pulse and completion of calibration when QEP is used, or, completion of calibration when other encoders are used)
5. Now the GUI can be called into perform a frequency sweep of the D axis current loop 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’.
6. When the frequency sweep is fully done, it will compute the Bode plot and display the results as shown in Figure 11-6 and Figure 11-7.
7. The GUI also computes and displays the loop bandwidth, gain margin and phase margin.
8. Repeat the test, if desired, by chaning *FCL_params.wccD*, and at different speed and load conditions.
9. To disconnect the GUI, click the 'Disconnect' button on the GUI.
10. To stop the motor, reduce the HV dc input voltage and set *runMotor* to MOTOR_STOP.
11. After the motor stops, take the controller out of real-time mode and reset.



Open Loop/Closed Loop Sele
Open Loop
Frequency Vector 22
Start Frequency 15.0000 Hz
Steps Per Decade 10
Maximum 1.89 KHz
Injection Amplitude .0050
☑ Save SFRA Data as CSV*
* If Unchecked Data Exported to

Gain C/O Freq: 725.6Hz Gain Margin: N/A Phase Margin: 62.39 Deg

Figure 11-6. SFRA Open Loop Bode Plots of the Current Loop Showing Magnitude and Phase Angle


SFRA GUI
Software Frequency Response Analyzer
Math Mode
Fixed Point
Floating Point
Start Sweep
SWEEP COMPLETE
FRA Settings
Closed Loop Magnitude (Decibels) Vs. Frequency (Hertz)
Closed Loop Phase (Degrees) Vs Frequency (Hertz)
100
1,000
Setup
Disconnec
Connected

Open Loop/Closed Loop Sele: Closed Loop

Frequency Vector 22

Start Frequency 15.0000 Hz

Steps Per Decade 10

Maximum 1.89 KHz

Injection Amplitude .0050

☑ Save SFRA Data as CSV*

* If Unchecked Data Exported to

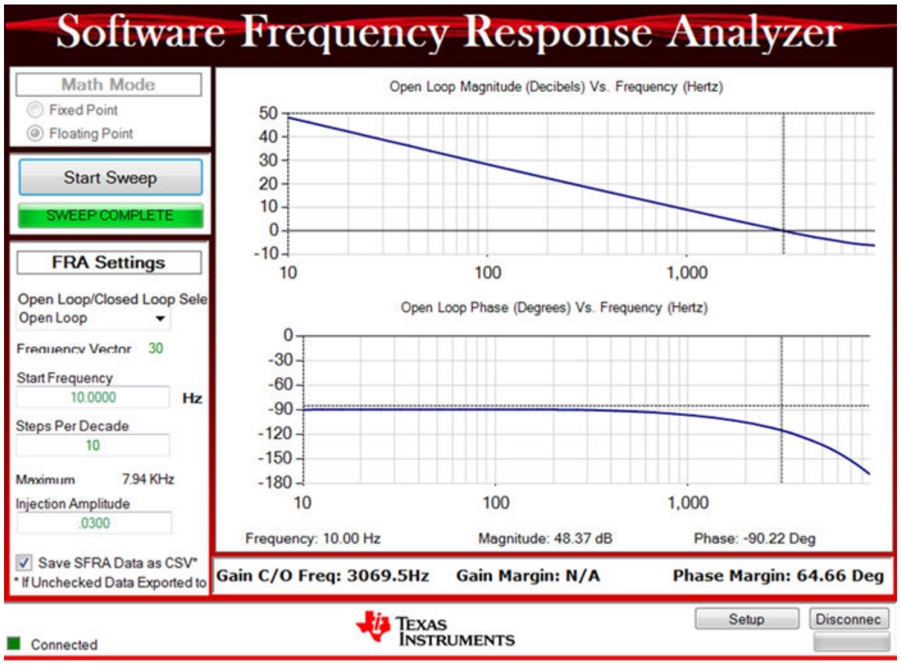
Frequency: 30.25 Hz Magnitude: 0.10 dB Phase: -1.76 Deg

Bandwidth: 1134.5Hz **Per GBT 16439-2009 China**

Figure 11-7. SFRA Closed Loop Bode Plots of the Current Loop Showing Magnitude and Phase Angle

11.6 Influence of Current Feedback SNR

The fast current loop is a high bandwidth enabler. When the designed bandwidth is high, the loop gains can also be high. This pretty much ties the loop performance to the quality of current feedback. If the SNR of current feedback signal into the digital domain is poor, then the loop can be very audibly noisy as the controller tries to minimize the error. If the noise is bothersome, you may be required to reduce the bandwidth to avoid the audible noise. Therefore, for a higher bandwidth and higher performance, the feedbacks should be of higher SNR to get the frequency responses a shown in Figure 11-8 and Figure 11-9.

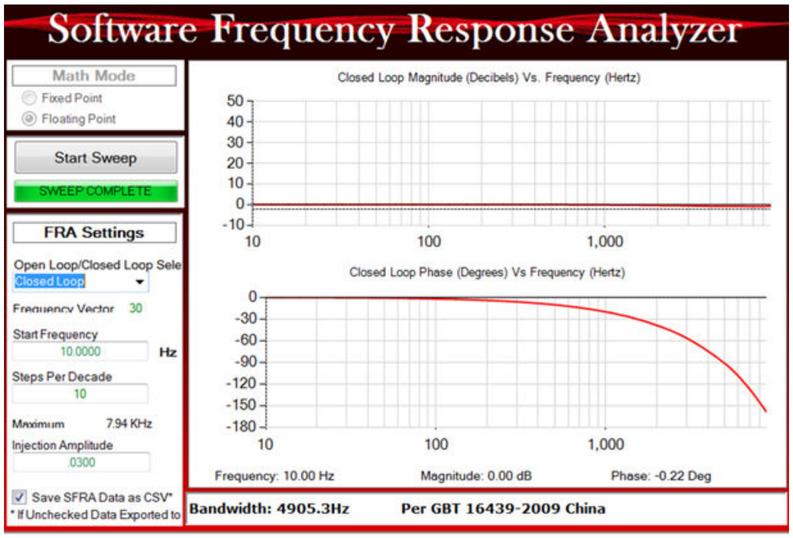

Software Frequency Response Analyzer
Math Mode
Fixed Point
Floating Point
Start Sweep
SWEEP COMPLETE
FRA Settings
Open Loop Magnitude (Decibels) Vs. Frequency (Hertz)
Open Loop Phase (Degrees) Vs. Frequency (Hertz)
Setup
Disconnec
Connected

Open Loop/Closed Loop Sele: Open Loop
Frequency Vector 30
Start Frequency 10.0000 Hz
Steps Per Decade 10
Maximum 7.94 KHz
Injection Amplitude .0300
Save SFRA Data as CSV*
* If Unchecked Data Exported to

Frequency: 10.00 Hz Magnitude: 48.37 dB Phase: -90.22 Deg

Gain C/O Freq: 3069.5Hz Gain Margin: N/A Phase Margin: 64.66 Deg

Figure 11-8. SFRA Open Loop Bode Plots of the Current Loop - Current Feedback With High SNR


Software Frequency Response Analyzer
Math Mode
Fixed Point
Floating Point
Start Sweep
SWEEP COMPLETE
FRA Settings
Closed Loop Magnitude (Decibels) Vs. Frequency (Hertz)
Closed Loop Phase (Degrees) Vs Frequency (Hertz)

Open Loop/Closed Loop Sele: Closed Loop
Frequency Vector 30
Start Frequency 10.0000 Hz
Steps Per Decade 10
Maximum 7.94 KHz
Injection Amplitude .0300
Save SFRA Data as CSV*
* If Unchecked Data Exported to

Frequency: 10.00 Hz Magnitude: 0.00 dB Phase: -0.22 Deg

Bandwidth: 4905.3Hz Per GBT 16439-2009 China

Figure 11-9. SFRA Closed Loop Bode Plots of the Current Loop - Current Feedback With High SNR

Note

With the hardware platform IDDK, there is scope for improving the SNR of the current signal feeding into the ADCs of the MCU. Therefore, higher bandwidth tests can be more noisier (chattering in nature) on this platform.

11.7 Inferences

11.7.1 Bandwidth Determination From Closed Loop Plot

The controller implemented for the open loop and closed loop plots shown in Figure 11-8 and Figure 11-9 is a dead beat controller where the output catches up to the input in just one sample cycle without any overshoots or requiring multiple cycles. From the closed loop plot, it is clear that the closed loop gain is always 0dB (unity gain) at all frequencies and therefore, magnitude based bandwidth determination is not practical. Hence, the phase plot is chosen as reference, and the frequency at which the phase lag goes beyond 90° is taken as bandwidth per the Chinese standard GBT 16439-2009 or NEMA ICS 16 (speed loop). In this test case, the PWM frequency is chosen as 10 KHz and the sampling frequency is 20 KHz and the current loop bandwidth obtained from the closed loop plot is about 5000Hz per these guidelines.

11.7.2 Phase Margin Determination From Open Loop Plot

From the open loop plots, the phase margin obtained is about 65°. Such a high margin should give a very robust performance across the frequency ranges within the bandwidth obtained.

11.7.3 Maximum Modulation Index Determination From PWM Update Time

From Section 8.1, the time lapse between feedback sampling to PWM update is about 0.9μs. In this system where the PWM frequency is 10 KHz, the maximum modulation index is limited by the sampling method as follows

- Double sampling - just above 96%
- Single sampling - just about 98%

This is quite comparable to FPGA based systems where the entire algorithm is implemented in hardware.

11.7.4 Voltage Decoupling in Current Loop

The SFRA test can be performed at zero speed (by rigidly locking the shaft if needed) to get one plot as reference. The gain crossover frequency 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.

11.8 Phase Margin vs Gain Crossover Frequency

By varying the control bandwidth and repeating these tests and noting down the resulting gain cross over frequency and phase margin, a set of plots are obtained for Id loop as shown in Figure 11-10. Two sets of tests are performed, one based on classical current control method, and the other based on FCL. Both these tests were performed using different current regulators. They all gave converging results.

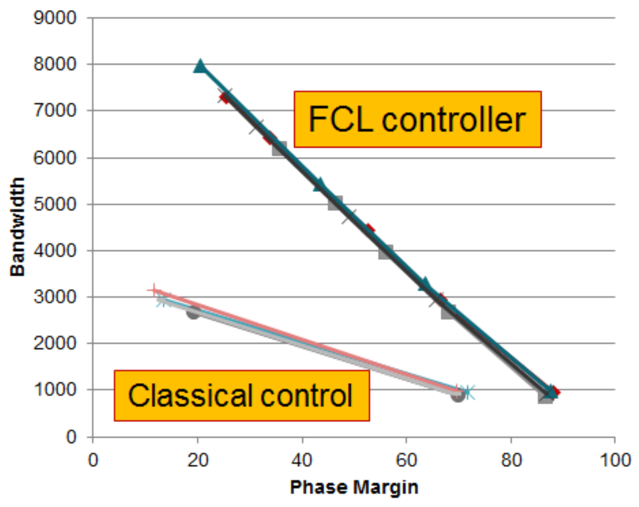


Figure 11-10. Plot of Gain Cross over Frequency vs Phase Margin as Experimentally Obtained

The group of plots at the bottom is obtained for conventional control, and it is obvious that the gain cross over frequency is too low and that as the gain cross over frequency is increased, the phase margin drops a lot faster.

The group of plots at the top is obtained with FCL. The gain cross over frequency is nearly thrice that of the classical method for a given phase margin. And also, as the gain cross over frequency is increased, the relative drop in phase margin is very low compared to the classical method. This effectively means that FCL can provide a higher bandwidth or gain cross over frequency at a higher phase margin.

12 Incremental Build Level 7

This build level deals with setting up the EtherCAT slave in F2838x device to link up with TwinCAT master in a PC. Among the three cores of F2838x MCU (two C28x CPU cores (CPU1/2) + one Arm Cortex-M4 core (CM)), the EtherCAT peripheral can be connected to CPU1 or M4. In the example here, M4 (also called as connectivity manager (CM)) is chosen to interact with the EtherCAT peripheral, while the C28x core (CPU1) will perform all control functions for the servo drive. The setup procedure involves CPU1 setting up the GPIOs and clocking required for EtherCAT and then allocating the EtherCAT ownership to the Connectivity Manager (CM). Then a dedicated code on CM is required to pass the data between the EtherCAT slave and the CPU1 though inter processor communication peripheral (IPC). This is all taken care of in FCL_LEVEL7.

12.1 Run the Code on CPU1 to Allocate ECAT to CM

1. Open '*fcl_f2838x_tmdxiddk_settings_cpu1.h*' and select level 7 incremental build option by setting the BUILDLEVEL to FCL_LEVEL7 (#define BUILDLEVEL FCL_LEVEL7). Right-click on the project name, and then click *Rebuild Project*.

Note

Control power supply to the IDDK is alone needed for this build level, therefore, you can turn off the high voltage dc power to the kit.

2. When the build is complete, click the *Debug* button, reset the CPU, restart, enable real-time mode, and run.
3. Right-click within the CCS Expressions window and choose Import. Then browse to and select *fcl_f2838x_tmdxiddk_ecat_vars.txt* file from the debug directory. This will add the following variables within the 'Expressions Window':
   a. *countMainLoop* : counter to just show activity in ECAT main loop function
   b. *dataBufferFromCM* : buffer copy of message from CM
   c. *dataBuffertoCM* : buffer copy of messge to CM
   d. *ipcCMToCPUDataBuffer* : data buffer written by CM for use by CPU1
   e. *ipcCPUToCMDataBuffer* : data buffer written by CPU1 for use by CM
4. Running the code will enable the GPIOs needed for EtherCAT and also passed over the ownership of EtherCAT slave peripheral to CM, besides setting up the IPC on CPU1 side for data transfer between CPU1 and CM. Now is the time to load / run the code on CM to initialize the EtherCAT peripheral.

Note

If ever the EtherCAT was assigned to CPU1 in any of the immediate previous evaluation projects and is now being assigned to CM, then make sure to power cycle the controlCARD before running the CM project to ensure that CPU1 has fully relinquished the EtherCAT peripheral.

12.2 Run the Code on CM to Setup ECAT

The scope of the demo example is to emulate a debug panel for the servo drive using ECAT to feed certain command settings and to retrieve the status of certain drive parameters. Therefore, the demo example on CM is a precompiled demonstration of the EtherCAT slave stack code tweaked for this application.

Note

Any modifications to the requirement given in demo will require the generation of new slave stack files via the SSC tool and also corresponding changes to the application source code. This will have to be rebuilt to get a new executable for loading into the CM.

The source files given by TI for EtherCAT support are only HAL files needed for this function and will not be sufficient to compile and generate an executable. However, if you generate a slave stack, the CM project provided here will be able to generate an executable.

Here are the steps to follow:

1. Ensure that CPU1 is actively running the project described in the previous section handing off the EtherCAT ownership to CM
2. Besides USB connection between controlCARD and computer for JTAG purposes, connect an Ethernet cable between controlCARD RJ45 Port 0 and computer
3. Run *f2838x_connected_drive_ssc_file_and_demo.exe* installer to extract the F2838x SSC configuration, device system files required by the SSC tool and the demo executable that will run on CM. These will be located in the newly created *ssc_configuration* directory.
4. From within the debug window, click on Cortex_M4, right click and select *Connect Target*. This will facilitate downloading the M4 executable.
5. From within the CCS debug perspective, click *Run --> Load --> Load Program* and browse to the executable *fcl_f2838x_ecat_cm.out* available at *\solutions\tmdxiddk379d\f2838x\ssc_configuration\cm.* This is extracted in step 3 above.
6. Run the code. This would configure the ECAT slave controller and also the IPC on CM side for data transfer between CM and CPU1.

Now the ECAT slave controller is ready to connect to ECAT master, and in this demo, the ECAT master is TwinCAT running on the user's development computer.

12.3 Setup TwinCAT

The TwinCAT software turns almost any compatible PC into a real-time controller with a multi-PLC system, NC axis control, programming environment and operating station. TwinCAT is a PC software functioning as an EtherCAT master to control various EtherCAT slave nodes connected to the PC. The following are the steps to setup TwinCAT

1. Optional: Install Microsoft Visual Studio. This isn't required since TwinCAT will install a Visual Studio shell if no Visual Studio installation is found.
   a. Download and install Microsoft Visual Studio
   b. TwinCAT supports integration into Visual Studio 2010/2012/2013/2015/2017
2. Download and install TwinCAT3 from the Beckhoff
   a. Follow the left sidebar to *Download->Software->TwinCAT 3->TE1xxx | Engineering* and select the software product *TwinCAT 3.1 eXtended Automation Engineering (XAE)*
3. Once installation is complete, verify that the TwinCAT Runtime is active
   a. Check that the TwinCAT Config Mode icon is shown in the Windows notification panel as shown in Figure 12-2.

Figure 12-1. TwinCAT Config Mode Icon

   b. Right click on this icon and select *Tools->TwinCAT Switch Runtime*. From the Tc- SwitchRuntime window, verify that it is *active*. When active, it will only provide the option to *Deactivate*. Do not deactivate!

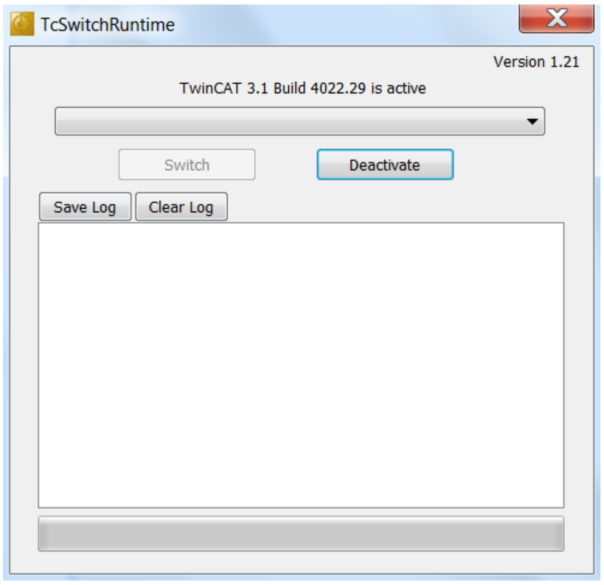


Figure 12-2. TcSwitchRunTime Window Activated

   c. If the icon is not present, then locate the TwinCAT Runtime executable from the file system. (Default installation location is typically: *C:/TwinCAT/TcSwitchRuntime*)

4. Start up Visual Studio with TwinCAT using one of the following methods:
   a. Recommended: Right click the TwinCAT Config Mode icon from the Windows notification panel and select *TwinCAT XAE*
   b. installed desktop icon: *TwinCAT XAE*
   c. Use installed Start Menu icon under Beckhoff folder: *TwinCAT XAE*
5. Once Visual Studio running, verify that the main toolbar has options *TwinCAT* and *PLC* shown. If these aren't present, then the TwinCAT Switch Runtime isn't active.
6. Within Visual Studio, create a new EtherCAT project. Select *File -> New -> Project* and under templates select *TwinCAT Projects* then *TwinCAT XAE Project (XML format)*. Fill in a name (say *f2838x_iddk*) and click *OK*.
7. Now that the project is created, verify that a realtime Ethernet adapter is installed.
   a. In Visual Studio, select the *TwinCAT* menu from the main toolbar and select *Show Realtime Ethernet Compatible Devices*
   b. In the popup window, under *Installed and ready to use devices (realtime capable)* category, if no connections are shown, select one from the list of Compatible devices and click *Install*.
8. TwinCAT setup is now complete.

12.4 Scanning for EtherCAT Devices via TwinCAT

Us the following steps to scan EtherCAT devices via TwinCAT:

1. Open the TwinCAT project created in Section 12.3 (*f2838x_iddk*)
2. Verify that the controlCARD is running the demo code (CM running *fcl_f2838x_ecat_cm.out*) as described in Section 12.2 and that the development computer (running TwinCAT) is connected via an Ethernet cable to port 0 connection in controlCARD.

Note

Port 0 is the top Ethernet port on the side of the controlCARD with two Ethernet connections.

3. In Visual Studio on the left side solution explorer, expand the *Project*, then expand *I/O*
4. Right click on *Devices* and select *Scan*
   a. A dialog will popup stating that *Not all types of devices can be found automatically.* Click *OK.*

5. Once scanning is complete, a popup window will appear. The following options may appear:
   a. A popup stating that *1 new I/O devices found* where the device is *Device 2 (EtherCAT Automation Protocol)*. This or any other device numbers besides *Device 1* is correct, click *OK*.
   b. A popup stating that no devices have been found or stating that *1 new I/O devices found* where the device is *Device 1 (EtherCAT Automation Protocol)*. This means some setup is incorrect. Verify that the example is running on the device (or at least has gone through the GPIO setup and reset of the EtherCAT IP). If the procedure is followed correctly, it should have identified *Device 2*. The setup procedure may be repeated again carefully.
6. After clicking *OK*, another popup will ask to *Scan for boxes*. Click *Yes*.
7. After clicking *Yes*, another popup will ask to *Activate Free Run*. Click *Yes*.
8. In the solution explorer on the left, under devices you should see *Device 2 (EtherCAT).* Under that, there will be a *Box #*. This *Box* is the controlCARD ESC.
9. The EtherCAT master communication is now setup with the slave device.

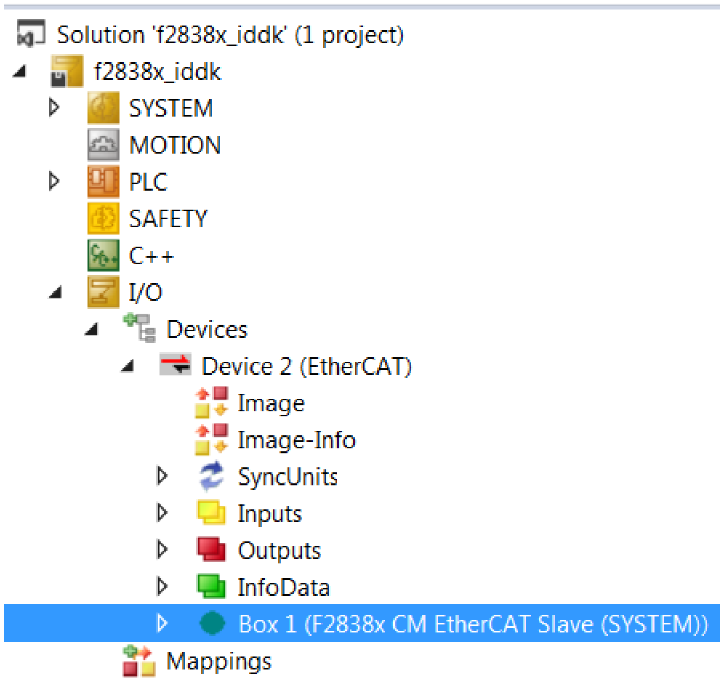


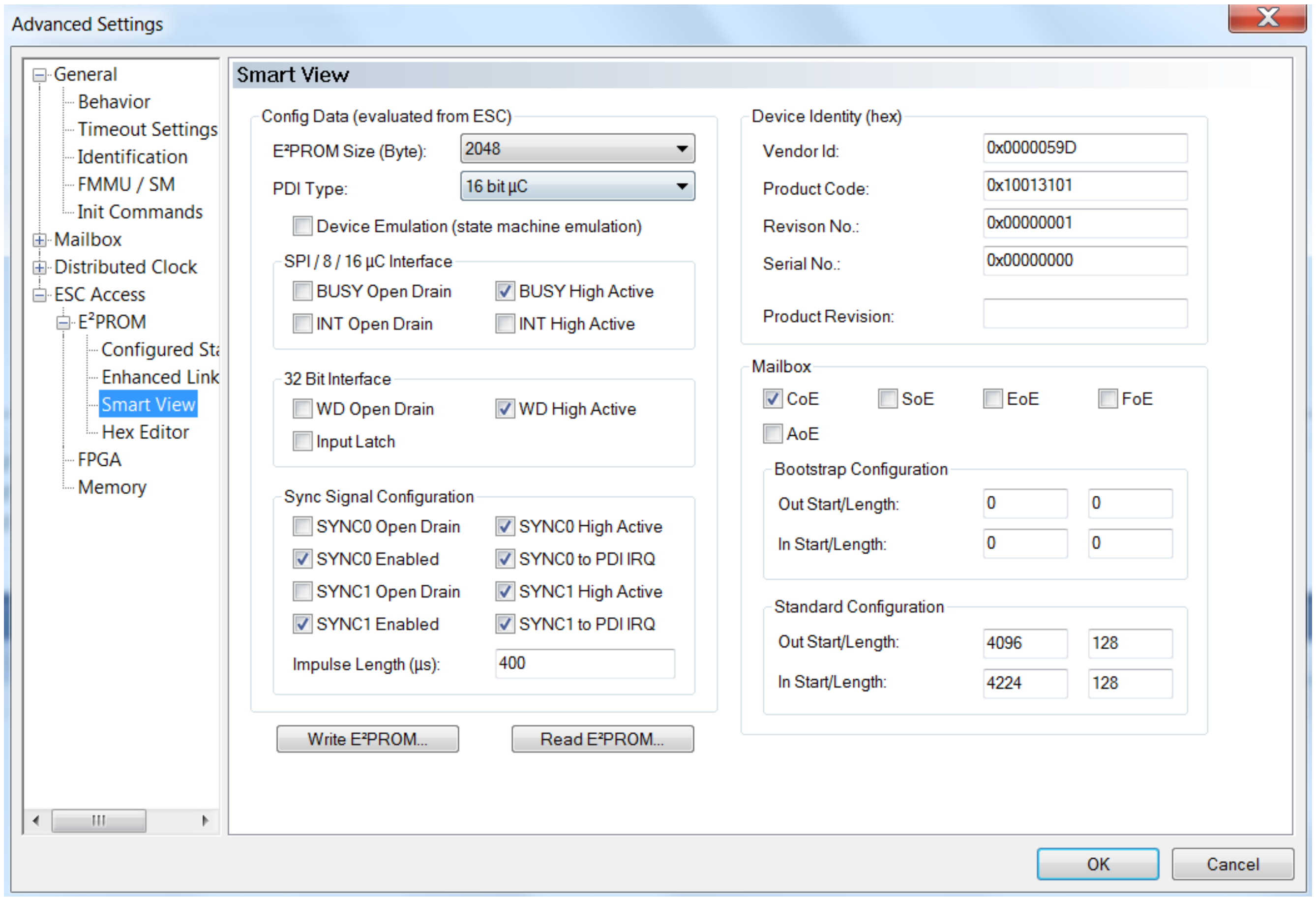
Figure 12-3. TwinCAT Solution Explorer

12.5 Program ControlCard EEPROM for ESC

Verify first that TwinCAT has discovered the ESC as described in Section 12.4. Also, make sure to copy the ESC XML file into TwinCAT directory before starting off to program the EEPROM.

1. Copy the ESI file (*F2838x CM EtherCAT Slave (System).xml* available at *\solutions\tmdxiddk379d\f2838x \ssc_configuration*) into the TwinCAT directory (Default location: *C:\TwinCAT\3.1\Config\Io\EtherCAT*). If TwinCAT was already open, it has to be closed and re-opened again for it to see new XML file(s).
2. In the Visual Studio solution explorer, double click on *Box #* under *Device 2 (EtherCAT)*.
3. The TwinCAT project window should be open to the right of the solution explorer and have some tabs such as *General, EtherCAT, and so forth*
4. Select the *EtherCAT* tab and then click on *Advanced Settings*
5. In the new window, expand the *ESC Access* menu, then expand the *E2PROM* menu. Click on *Smart View*
6. Click on *Write E2PROM* and expand the *Texas Instruments Incorporated* menu within the *Available EEPROM* Descriptions window.
7. Expand *TI C28xx Slave Devices* and select *F2838x CM EtherCAT Slave*. Click *OK*.
8. Visual Studio will indicate that the EEPROM is being programmed. When it completes, if the Smart View doesn't automatically update with the new contents, you can select *Read E2PROM* to read back the newly programmed values.

9. The Product Code for this CM application is *0x10003101*
10. Once EEPROM is programmed, disconnect and power cycle the controller.



Smart View

Config Data (evaluated from ESC)
- E²PROM Size (Byte): 2048
- PDI Type: 16 bit µC
- ☐ Device Emulation (state machine emulation)

SPI / 8 / 16 µC Interface
- ☐ BUSY Open Drain
- ☑ BUSY High Active
- ☐ INT Open Drain
- ☐ INT High Active

32 Bit Interface
- ☐ WD Open Drain
- ☑ WD High Active
- ☐ Input Latch

Sync Signal Configuration
- ☐ SYNC0 Open Drain
- ☑ SYNC0 High Active
- ☑ SYNC0 Enabled
- ☑ SYNC0 to PDI IRQ
- ☐ SYNC1 Open Drain
- ☑ SYNC1 High Active
- ☑ SYNC1 Enabled
- ☑ SYNC1 to PDI IRQ
- Impulse Length (µs): 400

Device Identity (hex)
- Vendor Id: 0x0000059D
- Product Code: 0x10013101
- Revison No.: 0x00000001
- Serial No.: 0x00000000
- Product Revision:

Mailbox
- ☑ CoE
- ☐ SoE
- ☐ EoE
- ☐ FoE
- ☐ AoE

Bootstrap Configuration
- Out Start/Length: 0 0
- In Start/Length: 0 0

Standard Configuration
- Out Start/Length: 4096 128
- In Start/Length: 4224 128

Figure 12-4. TwinCAT EEPROM Window

12.6 Running the Application

This section deals with running the application.

1. Verify or restore the connections if disconnected before.
2. Reload CPU1 and CM applications
3. Run CPU1 first and hand-off the EtherCAT peripheral to CM
4. Run the CM next to setup the EtherCAT connection with master and IPC link with CPU1
5. From TwinCAT window, rescan for devices, and restart TwinCAT in *config mode*.

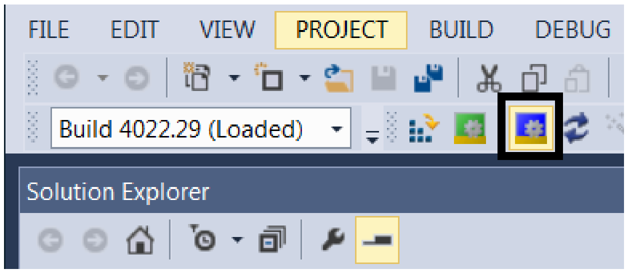
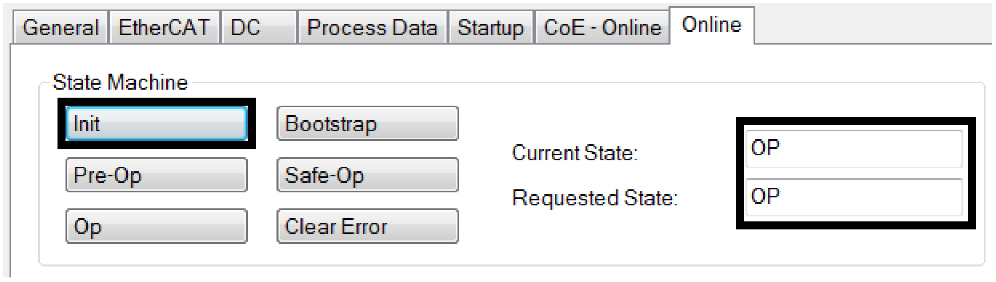


Figure 12-5. TwinCAT Restart in Config Mode Icon

6. ECAT messages can be viewed as variables in CCS debug window. If not done already, right-click within the CCS Expressions window and choose Import. Then browse to and select *fcl_f2838x_tmdxiddk_ecat_vars.txt* file from the debug directory. Variables 'countMainLoop', 'dataBufferFromCM' and 'dataBuffertoCM' are used by C28x core only while 'ipcCMToCPUDataBuffer' and 'ipcCPUToCMDataBuffer' are used by both C28 and CM cores and hence can be viewed from both '*Debug*' spaces.
7. Within TwinCAT, double-click on the discovered EtherCAT box and observe that the EtherCAT slave is running in OP mode.



Current State: OP

Requested State: OP

Figure 12-6. EtherCAT Slave in OP Mode

8. Within TwinCAT, expand the explorer to the EtherCAT *box* (I/O --> Devices -->Device 2 (EtherCAT) --> Box 1 (F2838x CM EtherCAT Slave(SYSTEM))) and find the various output/input mappings as shown in Figure 12-7. Double click on Box1 to get the F2838x CM slave input /output data window as shown in Figure 12-8.
   a. Select Input mapping 0 to view all the status feedback from the drive. The following are the drive parameters monitored through EtherCAT:
      i. SpeedStatus
      ii. PositionStatus
      iii. TorqueStatus
      iv. DriveStatus
   b. Select Output mapping 0 to view and enter all the commands to the drive:
      i. DriveCommand
      ii. SpeedReference
      iii. PositionReference

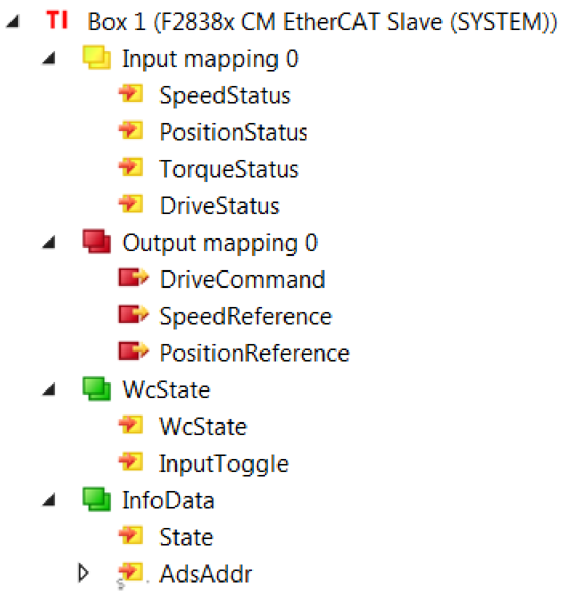
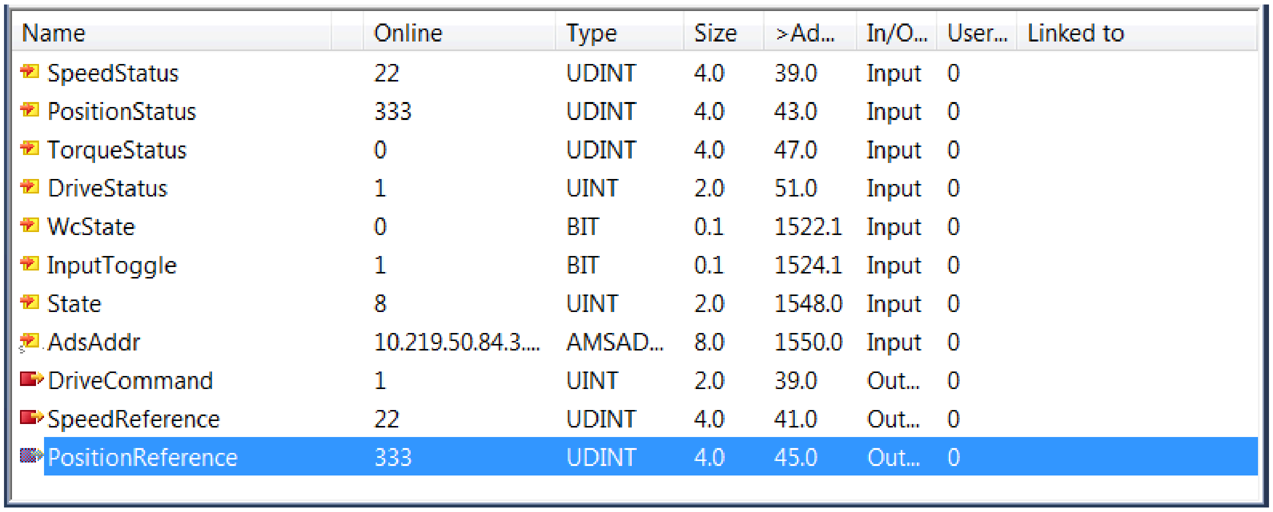


Figure 12-7. TwinCAT Solution Explorer Inputs and Outputs



Name	Online	Type	Size	>Ad...	In/O...	User...	Linked to
SpeedStatus	22	UDINT	4.0	39.0	Input	0	
PositionStatus	333	UDINT	4.0	43.0	Input	0	
TorqueStatus	0	UDINT	4.0	47.0	Input	0	
DriveStatus	1	UINT	2.0	51.0	Input	0	
WcState	0	BIT	0.1	1522.1	Input	0	
InputToggle	1	BIT	0.1	1524.1	Input	0	
State	8	UINT	2.0	1548.0	Input	0	
AdsAddr	10.219.50.84.3....	AMSAD...	8.0	1550.0	Input	0	
DriveCommand	1	UINT	2.0	39.0	Out...	0	
SpeedReference	22	UDINT	4.0	41.0	Out...	0	
PositionReference	333	UDINT	4.0	45.0	Out...	0	

Figure 12-8. F2838x CM EtherCAT Slave (SYSTEM) Input/Output Data

9. This build level is a loop back example to verify the EtherCAT functionality. So, if the command data entered in a certain output mapping variable is returned back in a certain input mapping variable, it confirms the functionality of EtherCAT slave on the F2838x. The loop back association is as follows:
   a. SpeedReference --> SpeedStatus
   b. PositionReference --> PositionStatus
   c. DriveCommand --> DriveStatus

   For example, the data entered in 'SpeedReference' in output mapping should return back to show up in 'SpeedStatus' in input mapping if EtherCAT slave is functioning properly.

13 Incremental Build Level 8

Assuming the previous build level is successfully completed, this build level attempts to control the drive by feeding in commands from the TwinCAT window and monitoring the status feedback from the drive using the same through ECAT.

13.1 Run the Code on CPU1 to Allocate ECAT to CM

1. Open '*fcl_f2838x_tmdxiddk_settings_cpu1.h*' and select level 8 incremental build option by setting the BUILDLEVEL to FCL_LEVEL8 (#define BUILDLEVEL FCL_LEVEL8). Right-click on the project name, and then click *Rebuild Project*.
2. When the build is complete, click the *Debug* button, reset the CPU, restart, enable real-time mode, and run.

Note

Since this build level attempts to drive the motor, high voltage dc power to the kit should be restored at this point.

3. CPU1 should have given the ownership of ECAT to CM
4. CPU1 would be waiting to receive commands from CM through IPC channel.

13.2 Run the Code on CM to Setup ECAT

This build level uses the same precompiled executable of CM that is used in the previous build level. Here, CM is used as a conduit to pass information between TwinCAT and drive controller.

Note

Any modifications to the requirement given in demo will require the generation of new slave stack files via the SSC tool and also corresponding changes to the application source code. This will have to be rebuilt to get a new executable for loading into the CM.

Here are the steps to follow:

1. Ensure that CPU1 is actively running the project (described in Section 13.1) handing off the EtherCAT ownership to CM
2. Besides USB connection between controlCARD and computer for JTAG purposes, connect an Ethernet cable between controlCARD RJ45 Port 0 and computer
3. From within the CCS debug perspective, click *Run* --> *Load* --> *Load Program* and browse to the executable *fcl_f2838x_ecat_cm.out* available at *\solutions\tmdxiddk379d\f2838x\\ssc_configuration\cm*
4. Run the code. This will configure the ECAT slave controller and the IPC on CM side for data transfer between CM and CPU1.

Now the ECAT slave controller is ready to connect to ECAT master, which is the TwinCAT running on user's development computer.

13.3 Running the Application

1. Follow the same procedure given in Section 12.4 to bring up the TwinCAT and to scan the EtherCAT devices connected to it.
2. Follow the same procedure given in Section 12.6 to interact with the drive to feed in command references and to get the status back.
3. Unlike the previous build level, here the commands received by the controller are not looped back and are used to actually set the operational mode or references for the drive. The operational status of the drive is sent back via IPC --> CM --> TwinCAT.
4. The output mapping in TwinCAT sets the commands and references for the drive while the input mapping reveals the operating status of the drive.
5. Under output mapping, *'DriveCommand'* sets three different operating modes. It can take a value of 0, 1 or 2 and its significance is as follows
   a. 0 - STOP motor
   b. 1 - RUN the motor in SPEED mode --> outer loop is speed loop (as in BUILDLEVEL 4). *'SpeedReference'* under output mapping will act as the speed reference setting for this mode
   c. 2 - RUN the motor in POSITION mode --> outer loop is position loop (as in BUILDLEVEL 5). *'PositionReference'* under output mapping will act as the position reference setting for this mode
   d. any other value --> defaults to STOP motor
6. *'SpeedReference'* and *'PositionReference'* take values in a 1000 point scale (between 0 and +/-1000) as command. Values outside this range will be clamped in control side software (CPU1) to fit within range in the drive controller. These are then converted into per unit by CPU1, which is the default format used by the control software.
7. As the motor runs, it reports the speed feedback in speed mode and position feedback in position mode. Try varying the drive commands by editing the parameters under output mapping, and see the drive response in input mapping. Position feedback will always be given as a positive number within 0 and 1000. For example, if '*PositionReference*' is given as -200, the feedback '*PositionStatus*' will display a value close to 800. In angular terms, they represent the same position.

14 References

- Texas Instruments: *Designing With The C2000 Configurable Logic Block*
- Texas Instruments: *How to Migrate Custom Logic From an FPGA/CPLD to C2000 Microcontrollers*
- Texas Instruments: *DesignDRIVE IDDK Hardware Reference Guide*
- Texas Instruments: *DesignDRIVE IDDK User Guide*

15 Revision History

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

Changes from Revision A (October 2019) to Revision B (November 2020)	Page
• Updated the numbering format for tables, figures and cross-references throughout the document	4
• Updates were made in Section 4	6
• Added new Section 5.1.4	11
• Added new Section 7.5	19