TI Spins Motors

Servo Drive with CAN - Lab Projects User's guide

Contents

1. Abstract

This document discusses how to set up a motor drive platform using Texas Instruments' motor EVM kits, to implement sensored-FOC based motor drive using QEP encoder and integrate the CAN communication for starting the motor and setting the speed.

2. Lab Description

The system is incrementally built up in order for the final system can be confidently operated. Three phases of the incremental system build are designed to verify the major modules in the system. In each build level, a certain operation of the system, could be hardware or software, is verified and integrated incrementally. In the final build level, all operations are integrated to make a complete system. Table 1 summarizes the modules testing and using in each incremental system build.

Table 1: Functions Verified in Each Incremental System Build

Build Level	Functional Integration
Level 1	Open loop control to verify integrity of user hardware and calibration of feedbacks
Level 2	Closing speed loop with step response Generation & Graphing for Controller Tuning
Level 3	Closing speed loop using QEP encoder with CAN communication

2.1. Supporting Software

1 Always make sure you are using the latest version of MotorControl SDK

a. http://www.ti.com/tool/download/C2000WARE-MOTORCONTROL-SDK

b. MotorControl SDK contains all of the modules, drivers, example Code Composer Studio based motor control projects, and associated documentation.

c. Detect and import example projects of MotorControl SDK by using Resource Explore in CCS. View MotorControl SDK Resource Explorer on the web, or Resource Explorer can also be opened by going to the menu View-> Resource Explorer in CCS.

d. Read the solution documentation:

I. C2000Ware Motor Control SDK Getting Started Guide.

2 Always use the latest version of Code Composer Studio for example projects.

a. The latest version of Code Composer Studio can be obtained at the following link: http://www.ti.com/ccstudio

b. The latest C2000 compiler can be obtained at the following link:

http://www.ti.com/tool/c2000-cgt

Or update from CCS App Center. App Center can be opened by going to the menu View-> CCS App Center in CCS.

2.2. Supporting Kits

- C2000 LaunchPad™ development kits
  - LAUNCHXL_F280049C LaunchPad
    - PN: LAUNCHXL-F280049C
    - Includes on-card XDS110 JTAG (isolated)
    - Piccolo F28004x Series LaunchPad Evaluation Kit Guide
  - LAUNCHXL_F280025C LaunchPad
    - PN: LAUNCHXL-F280025C
    - Includes on-card XDS110 JTAG (isolated)
    - F28002x Series LaunchPad™ Development Kit
- 3-phase Inverters
  - Low Voltage / Medium Current: boostxldrv8320rs
    - PN: BOOSTXL-DRV8320RS
    - BOOSTXL-DRV8320RS EVM User's Guide
  - Low Voltage / Medium Current: boostxldrv8323rs
    - PN: BOOSTXL-DRV8323RS
    - BOOSTXL-DRV8323RS EVM User's Guide

2.3. Software Setup for Servo Drive with CAN project

1. Open Code Composer Studio. Note that this document assumes version 10.4.0 or later.
2. Once Code Composer Studio opens, the workspace launcher to select a workspace location.
   a. Click the "Browse…" button. Create the path below by making new folders as necessary.
   b. Click "Launch" to open CCS
3. Importing the servo drive with can project by clicking "Project->Import CCS Project", and then clicking "Browse…" to import the project from

"*\ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\servo_drive_with_can\f28002x\ccs\sensored_foc*" or "*\ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\servo_drive_with_can\f28004x\ccs\sensored_foc*".

4. The project can be configured to create code and run in either flash or RAM. You may select either of the two, however, for lab experiments use the RAM configuration most of the time and move to the FLASH configuration for production. Right-click on an individual project name and select "Build Configurations->Set Active->RAM_Lib" configuration.
5. From the CCS Edit Perspective, open the file "servo_main.h" and make sure that DMC_BUILDLEVEL is set to DMC_LEVEL_1. Right click on the project name and click on "Rebuild Project" on pop-up menu and watch the Console window. Any errors in the project will be displayed, which needs to be fixed.
6. Mount BoosterPack BOOSTXL-DRV8320RS on LaunchPad LAUNCHXL-F280049C headers J1-J4. or BOOSTXL-DRV8323RS on LAUNCHXL-F280025C headers J1-J4. DO NOT connect motor right now. Connect the LaunchPad to computer through an USB cable, this will light some LEDs on the emulator section of LaunchPad, indicating that the emulator is on. Power the BoosterPack with appropriate input dc voltage, 24V.
7. Click the "Debug" button located in the top-left side of the screen or click "Run->Debug" on CCS menu bar. The IDE will now connect to the target, load the output file into the device and change to the Debug perspective.
8. Click "View->Expressions" on the menu bar to open an Expressions window to view the variables being used in the project. Add variables to the expressions window manually. Alternately, a group of variables can be imported into the Expressions window, by right clicking within Expressions Window and clicking Import, and browse to the .txt file containing these variables. Here, browse to the directory of the project at "\ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\servo_drive_with_can\common\debug" and pick " servo_drive_with_can_vars.txt" and click OK to import the variables to Expressions window. Click on the Continuous Refresh button in the expressions window.
9. Run the code by clicking "Run" Button in the Debug Tab or clicking "Run->Resume" on the CCS menu bar.
10. In the Expressions window, set the related variable to run the motor.

3. System Software Integration and Testing

This section deals with various incremental build levels starting from first level up to the final level where the system is fully integrated.

3.1. Incremental Build Level 1

Assuming the load and build steps described in the Section 2.3 completed successfully. This section implements a scalar volts/frequency control to test the integrity of the hardware, namely to test the signal chain integrity, such as hardware current/voltage sensing and implement closed-loop control by introducing an incremental quadrature encoder to measure the motor angle.

This build level introduces an incremental quadrature encoder to measure shaft position. However, to begin with, it utilizes an angle generator module to generate the angle based on the motor target frequency. This uses a Volt/Hertz profile to generate an output command voltage to drive the motor. During initial rotation of the motor the QEP is calibrated. When an index pulse is detected, the angle determined by the QEP module is fed into the FOC, hence closed loop operation. MotorControl SDK API function calls will be used to simplify the microprocessor setup.

The setup of peripherals and even the inverter will be taken care of by MotorControl SDK APIs, specifically the HAL object and APIs. Important commands to initialise the drive are listed in Figure 1.

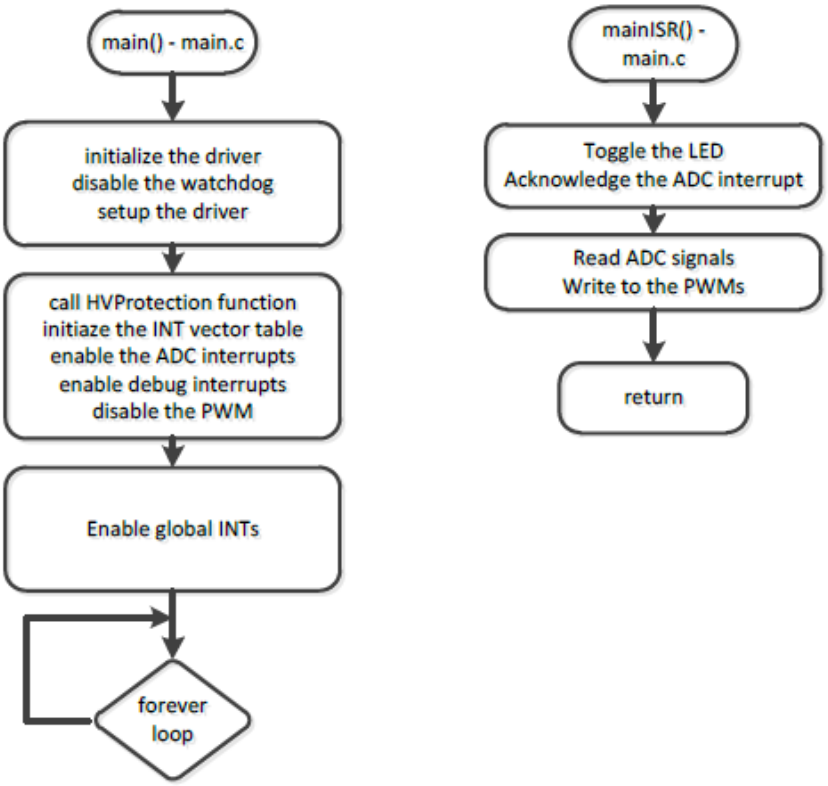


Figure 1: Software Flow Chart in Build Level 1

All build levels in this project are based on either the LAUNCHXL-F280025C and BOOSTXL-DRV8323RS or the LAUNCHXL-F280049C and BOOSTXL-DRV8320RS. The low voltage servo motor, LVSERVOMTR is used in this lab which includes an incremental quadrature encoder.

The use of the LaunchXL-F280025C and BoostXL-DRV8323RS is illustrated as the following. Set the switch and connector as shown in Figure 2.

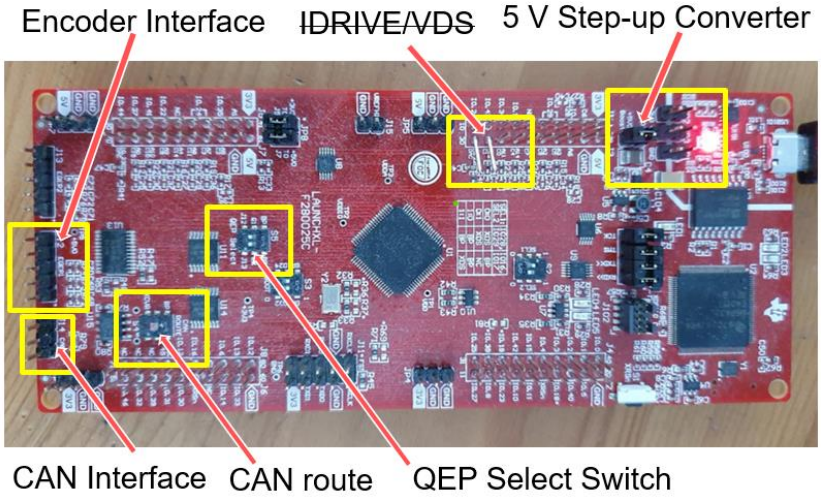


Figure 2: Switches on LAUNCHXL-F280025C

Idrive and **VDS** pins need to be physically disconnected from the BoostXl-DRV8323RS which can be achieved by simply bending them as in the picture above.

Take a flying wire from GPIO8 to SPIA_STE/GPIO5 as Figure 3. This connects Chip select of the LaunchPad to the BoosterPack. This step is unnecessary if using the LaunchXL-F280049C and BoostXL-DVR8320RS, simply connect the Boosterpack to the LaunchPad (J1-J4).

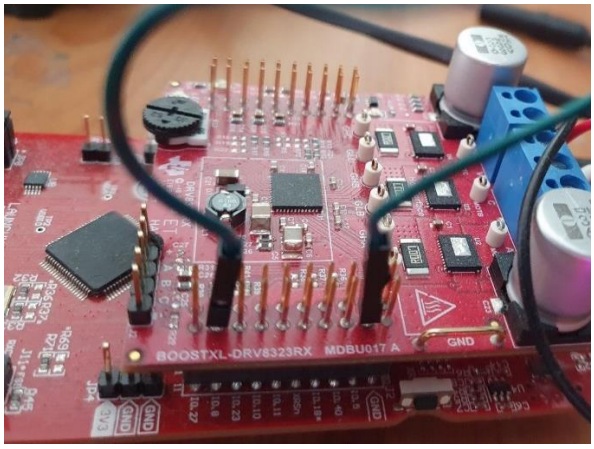


Figure 3: Jumper Wires on BoosterPack

Encoder interface: Connect Ground, 5 V, Index, A and B wires from the motor to the corresponding pins on the LaunchPad as Table 2. Header J12 is connected to eQEP1, by default this connection is not active, ensure S5 matches Figure 2.

Table 2: Encoder wire connections for reference kit motor

J12 on LaunchPad	**J4 on Teknic M-2310P-LN-04K**
J12-1/A	J4-1. A (Blue)
J12-2/B	J4-2. B (Pink)
J12-3/I	J4-3. I (Brown)
J12-4/+5V	J4-4. +5V (Red)
J12-5/GND	J4-5. GND (Black)

From left to right the connections on the BoosterPack are Ground, 24V, Phase A, Phase B and Phase C as shown in Figure 4. Connect 24V and ground to the power supply to the head of the BoosterPack. Set the power supply to 24 V and a current limit of 1 A.

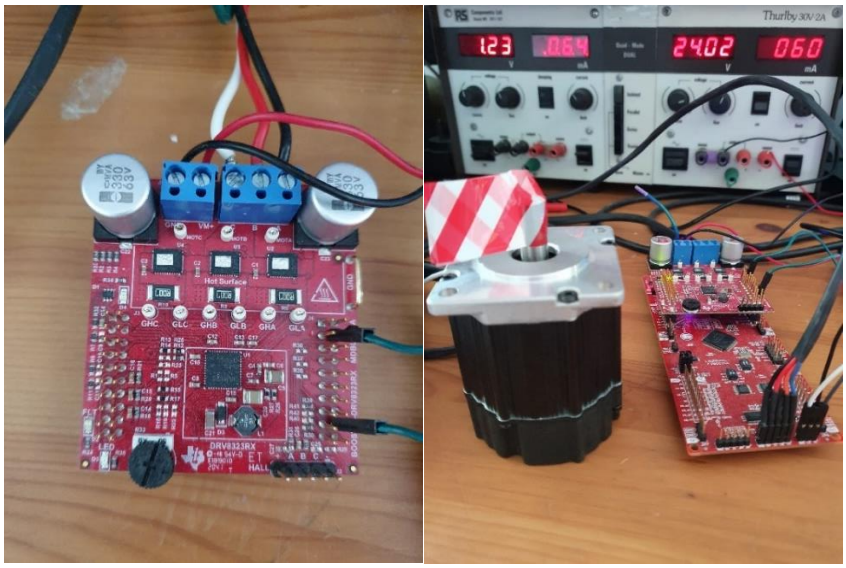


Figure 4: Motor Phase and Power Supply Connections

Connect the corresponding Phases of the motor to the head of the BoosterPack as Figure 4 and Table 3.

Table 3: Motor phase connections for reference kit motor

J5 on BoosterPack	**Motor Phase of Teknic M-2310P-LN-04K**
A/U	T (Black)
B/V	R (Red)
C/W	S (White)

1. Connect the LaunchPad/BoostXL USB cable and switch on the power. Open project within CCS and set DMC_BUILDLEVEL to DMC_LEVEL_1 in "servo_main.h", build and load project as described in the Section 2.3
2. To run the motor, set the following variables in the Expression Window.
   a. motorVars.flagEnableSys = 1
   b. motorVars.flagRunIdentAndOnLine = 1
3. To view the electrical angle of the encoder together with that of angleGen using Graph Tool
   a. Import the graph by Tools -> Graph -> Dual Time -> Import as shown in Figure 5, browse to "\ti\c2000\C2000Ware_MotorControl_SDK_<version>\solutions\servo_drive_with_can\common\debug", and choose "servo_drive_with_can_view.graphProp". This passes the electrical angle of the angleGen function and that of the QEP module to the graph function.

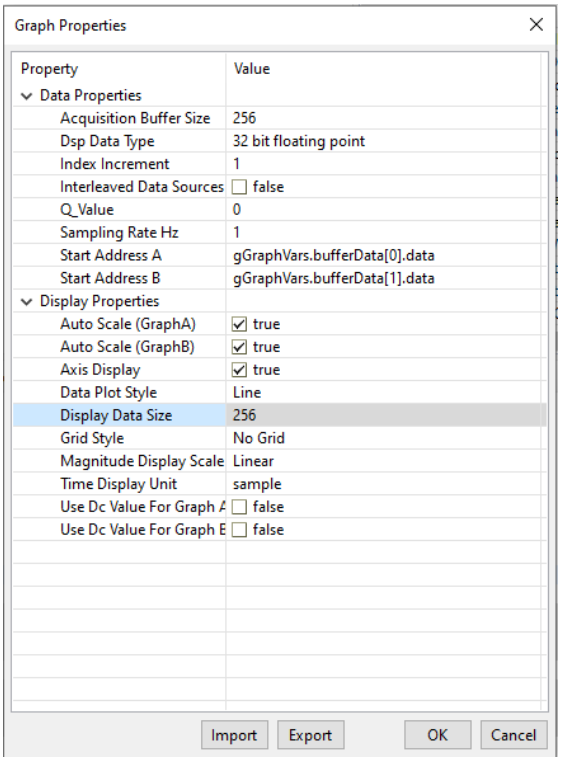


Figure 5: Graph Properties Settings for Build Leve 1

b. Change bufferMode to Graph_PH_A_VIEW by setting gGraphVars.bufferMode, and set gStepVars.stepResponse to “1” in Expression Window as shown in Figure 6, this will graph the electrical angle of the angleGen module and that of the QEP module as shown in Figure 7.

gGraphVars	struct GRAPH_Vars_t_	{bufferCounter=256,bufferTickCounter=1,buff...	0x0000A040@Data
bufferCounter	unsigned int	256	0x0000A040@Data
bufferTickCounter	unsigned int	1	0x0000A041@Data
bufferTick	unsigned int	10	0x0000A042@Data
bufferMode	enum <unnamed>	GRAPH_PH_A_VIEW	0x0000A043@Data
bufferData	struct GRAPH_Buffer_t_[2]	[{data=[-0.864325523,-0.703202248,-0.5593423...	0x0000A044@Data
gStepVars	struct GRAPH_StepVars_t_	{stepResponse=0,pSpeed_in=0x000004E0 {-0.8...	0x0000A000@Data
stepResponse	unsigned int	0	0x0000A000@Data
pSpeed_in	float *	0x000004E0 {1.88421726}	0x0000A002@Data
pId_in	float *	0x000004E2 {1.23294258}	0x0000A004@Data
pIq_in	float *	0x000004E4 {-1.5158577}	0x0000A006@Data
pSpeed_ref	float *	0x000004E6 {0.140455127}	0x0000A008@Data
pId_ref	float *	0x000004E8 {-0.0767199993}	0x0000A00A@Data
pIq_ref	float *	0x000004EA {-0.0067072995}	0x0000A00C@Data
spdRef_Default	float	20.0	0x0000A00E@Data
spdRef_StepSize	float	20.0	0x0000A010@Data
IdRef_Default	float	0.0	0x0000A012@Data
IdRef_StepSize	float	-1.0	0x0000A014@Data

Figure 6: Setup Variables for Graphing Angle

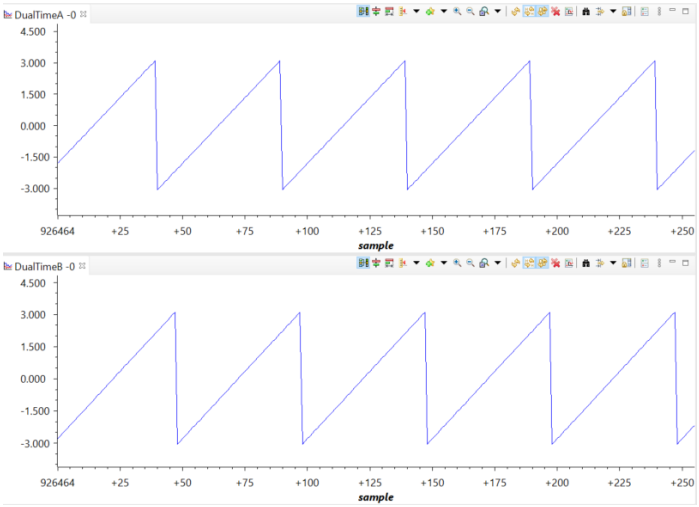


Figure 7: Angles from Angle Generator and QEP module

c. Change bufferMode to Graph_PH_B_VIEW by setting gGraphVars.bufferMode, and set gStepVars.stepResponse to “1” in Expression Window, this will graph the using angle for motor drive and phase A sampling current as shown in Figure 8Figure 7.

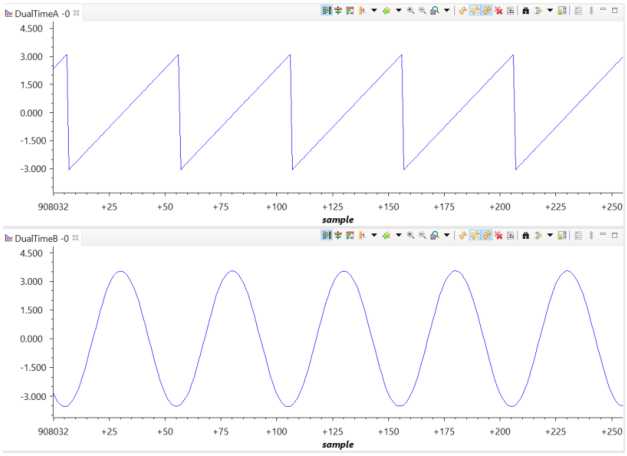


Figure 8: Using Angle and Phase A Current

d. Change bufferMode to Graph_PH_C_VIEW by setting gGraphVars.bufferMode, and set gStepVars.stepResponse to “1” in Expression Window, this will graph phase B and C sampling current as shown in Figure 9.

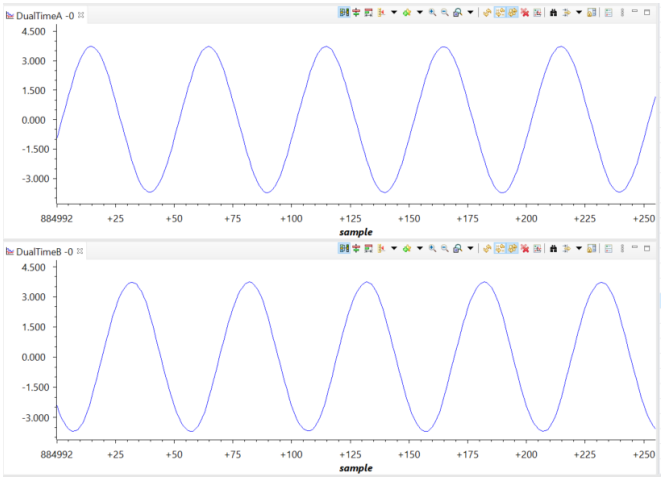


Figure 9: Phase B and C Current

4. To stop the motor by setting the motorVars.flagRunIdentAndOnLine to “0” in Expression Window.

3.2. Incremental Build Level 2

This build level is to generate step responses for the current and the speed controller. With these step responses it is possible to configure the speed and current PI controller to fit the customer system requirements.

In this build level, learn how to create a real-time step response for both PI current and PI speed controllers and display the step response in CCS graph tool. Use the tool to adjust the corresponding PI-controller parameters in MotorWare using the C2000 Real Time Debug capability with Code Composer Studio. This can be used to adjust the motor dynamic performance according to your system specification.

Step response generation will be performed in real-time with the motor running in closed loop using the Real Time Debug features. This project supports step response generation for either Id current or speed.

Build level 2 demonstrates a technique of tuning the proportional gain (Kp) and integral gain (Ki) for the current and speed controllers, this lab utilizes the graphing function of CCS to do step response testing. The default PI controller is just a starting point and needs tuning to meet the dynamic performance of the customer requirements.

Testing and adjusting the controller gains allows you to optimize the system for your specific requirements. Using a step response, it is possible to test the effects of the controller gains in your application and choose the values which meet your system response, stability, and efficiency targets. An overview of the used variables is shown in Figure 10.

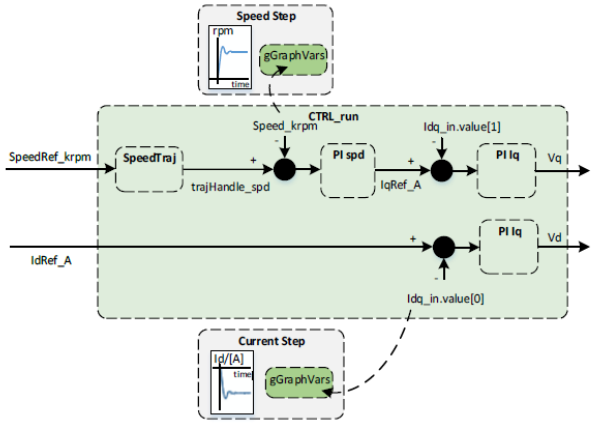


Figure 10: Cascaded Speed and Current Controller of the Sensored-FOC

The purpose is to generate a step response in the Id current controller to be almost independent of load torque changes. This removes the need for a load emulator like a dynamometer when trying to generate the step response in Iq.

1. Set DMC_BUILDLEVEL to DMC_LEVEL_2 in "servo_main.h", build and reload project as described in the Section 2.3.
2. To run the motor by setting motorVars.flagEnableSys and motorVars.flagRunIdentAndOnLine to “1” in Expression Window. The motor will start running at the speed defined in the variable “motorVars.speedRef_Hz”, and the actual motor speed can be seen in the variable “motorVars.speed_Hz”.
3. The variable “gGraphVars.BufferMode” selects between current step response generation or speed step response generation.
   a. Current step response generation is done setting the variable to GRAPH_STEP_RP_CURRENT (default) as shown in Figure 11.
   b. Speed step response generation is done setting the variable to GRAPH_STEP_RP_SPEED

gGraphVars	struct GRAPH_Vars_t_	{bufferCounter=256,bufferTickCounter=1,buff...
bufferCounter	unsigned int	256
bufferTickCounter	unsigned int	1
bufferTick	unsigned int	1
bufferMode	enum <unnamed>	GRAPH_STEP_RP_CURRENT
bufferData	struct GRAPH_Buffer_t_[2]	[{data=[0.0,0.0,0.0,0.0,0.0...],read=30280,write=...

Figure 11: Cascaded Speed and Current Controller of the Sensored-FOC

The variables are used in the following way.
“bufferCounter” used as an index to write into the buffer away structure
“bufferTickCounter” used to count the interrupts.
“bufferTick” defines how many interrupts happen per graph write
“bufferMode” can be used to define different values to record for each mode during run time
“bufferData” is the array where the data is stored

4. Load “servo_drive_with_can_current_step.graphProp” as in previous steps to make sure that the graph tool Y-scale to view the 0.5 A step response.

When using the graph there are several ways of updating the window. These must be selected for each Graph A and B.

 Press this button to refresh the current graph window with the current values written into the data array. This button needs to be pressed every time you have received new data, to show the new graph.

 Press this button to enable automatic graph update, every time the data array receives new data.

 Reset the current graph window to auto fit the new scale of the data array.

3.2.1. Step response generation for PI current controller

For Sensored-FOC in this lab, the speed and current PI controller are cascaded, so to start generating the step response the user has to begin with the current controller.

- gGraphVars.BufferMode = GRAPH_STEP_RP_CURRENT (default)

For your real application it will be important to test the step response generation while running with loads similar to those in the real system, but you can start testing the system with a lightly loaded or no load system. For this example, the motor is running without load and the speed is set to relatively low around 10% of rated value to ensure the motor is running stable. The graphs measured are done on the BOOSTXL-DRV8323RS.

- motorVars.speedRef_Hz = 40.0 (Hz)

To generate the current step the Id current is used. Because both Id and Iq controllers will use the same gains, we can keep test the effects of the step response on Id more simply without having to disconnect the output of the speed controller from Idq_ref_A.value[1]. The Id current is set using the variable Idq_ref_A.value[0], which is the reference current that is put on the direct axis of the magnetic field. Normally this is set to 0.0 A to orient the stator flux to the rotor. A negative “Idq_ref_A.value[0]” results in field weakening control.

The step chosen with the Teknic M2310PLN04K motor was be -0.5A, which was around 10% of the maximum rated Id current for that motor. For your specific motor you may choose, for example, a step of 10% of the rated Id current as an initial value.

Note that when the step response is executed and data has been logged the value of d-axis current, returns back to the default value to “gStepVars.IdRef_Value[0]”. For the current step response the following setting needs to be defined, default values are as below.

- gStepVars.IdRef_Default = 0.0 // Default starting Id reference value
- gStepVars.IdRef_StepSize = -0.5 // Step size of Id reference value
- gGraphVars. bufferTick = 1

Now set the following variable to:

- gStepVars.StepResponse = 1

To tune your controller watch the impact on the step response of the current controller changing the following variables in Expression Window.

- motorVars.Kp_Id
- motorVars.Ki_Id

Now you change either Kp or Ki one at a time. When one of the values has been changed, generate a new step response by setting the variable “gStepVars.StepResponse” to 1 to see the effect as shown in **Figure 12**.

Every time a new step response has been generated refresh the graph window to see the new step response (or turn on continuous refresh).

As an example, increase the value of Kp to get to the target Id reference value with a quick (stiff) response. Increasing it too much results in overshoot. Reducing it too much results in a very slow (soft) response.

The Ki value is calculated based on the ratio of Rs to Ls, and the controller period, so it will be as accurate as those values were identified. In most cases the Ki value does not need to be changed, but you can try out values (half the default, twice the default) to see the effect on the settling time or ringing of the step response.

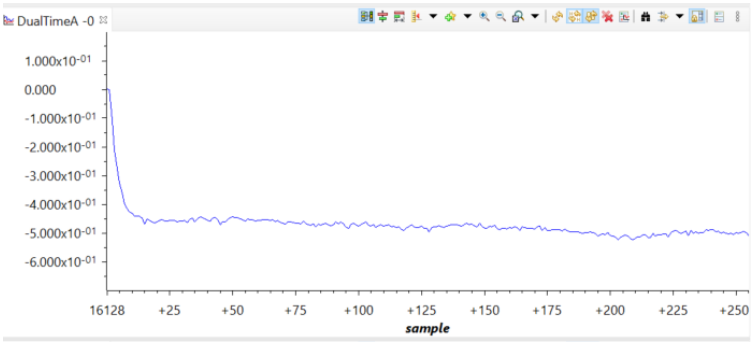


Figure 12: FOC Current step response showing no over shoot at high bandwidth

3.2.2. Step response generation for PI speed controller

Load “servo_drive_with_can_speed_step.graphProp” as in previous steps to make sure that the graph tool Y-scale to view the 60Hz step response.

To change the configuration of measuring the PI controller from current step response to a speed step response change the variable:

- gGraphVars.BufferMode = GRAPH_STEP_RP_SPEED
- gGraphVars. bufferTick = 1

To create a step speed response set:

- motorVars.accelerationMax_Hzps = 100

For the speed step response, the following setting needs to be defined, default values are as below.

- gStepVars.spdRef_Default = 40.0 // Default starting value of reference speed (Hz)
- gStepVars.spdRef_Stepsize = 60.0 // Step size of reference speed (Hz)

Depending on your motor max speed, define a step from around 1/8 to 2/8 of the max speed of the motor with the given Vbus voltage. When the step response is executed and the date has been logged the value of the “motorVars.speed_Hz” returns back to the default value to “motorVars.speedRef_Hz

As done with the current step response generation, set the following variable to:

- gStepVars.StepResponse = 1

Now refresh the graph window, to see the speed step response as shown in Figure 13.

To change the step response of the speed controller the following variables from the motorVars are used.

- motorVars.Kp_spd
- motorVars.Ki_spd

It can be seen as expected at with the chosen PI configuration and ramp, no overshoot of the speed is happening when changing speed.

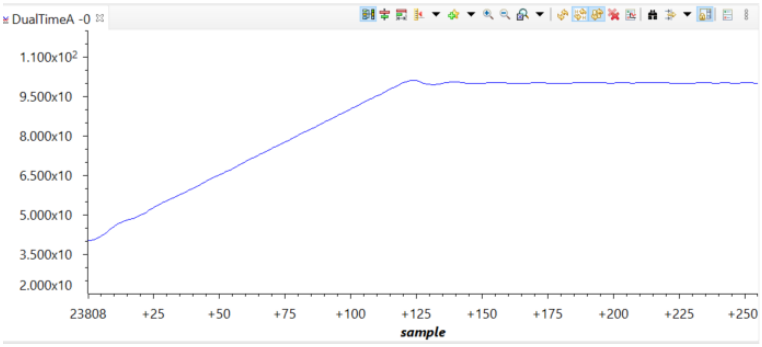


Figure 13: Speed step response showing no overshoot at high bandwidth

Using this project and its options it is possible to generate step responses for the current and the speed controller. With these step responses it is possible to configure the PI speed and current controller to fit the customer system requirements.

This adjusting of the PI controller can improve the overall system performance, efficiency and reliability, ensuring that the system is not regulating the current and speed too fast or too slow.

Note: For more information on the theory of control loop considerations, it is highly recommended to either see the InstaSPIN user guide (spruhj1) or read the book “Control Theory Fundamentals” ISBN-13 978-1496040732.

3.3. Incremental Build Level 3

This lab incorporates a CAN interface to receive a speed reference from a host motion controller. In this lab, communicate with the LAUNCHXL-F280025C using the onboard CAN transceiver. To achieve this another LaunchPad (F280049C or F280025C) is used to send a speed reference and start/stop command to the motor controller.

Figure 14 shows a typical setup of a mobile robot motor control system. In this lab the CAN interface is used to define a new speed reference from a host motion controller but this could be used to update any other parameter.

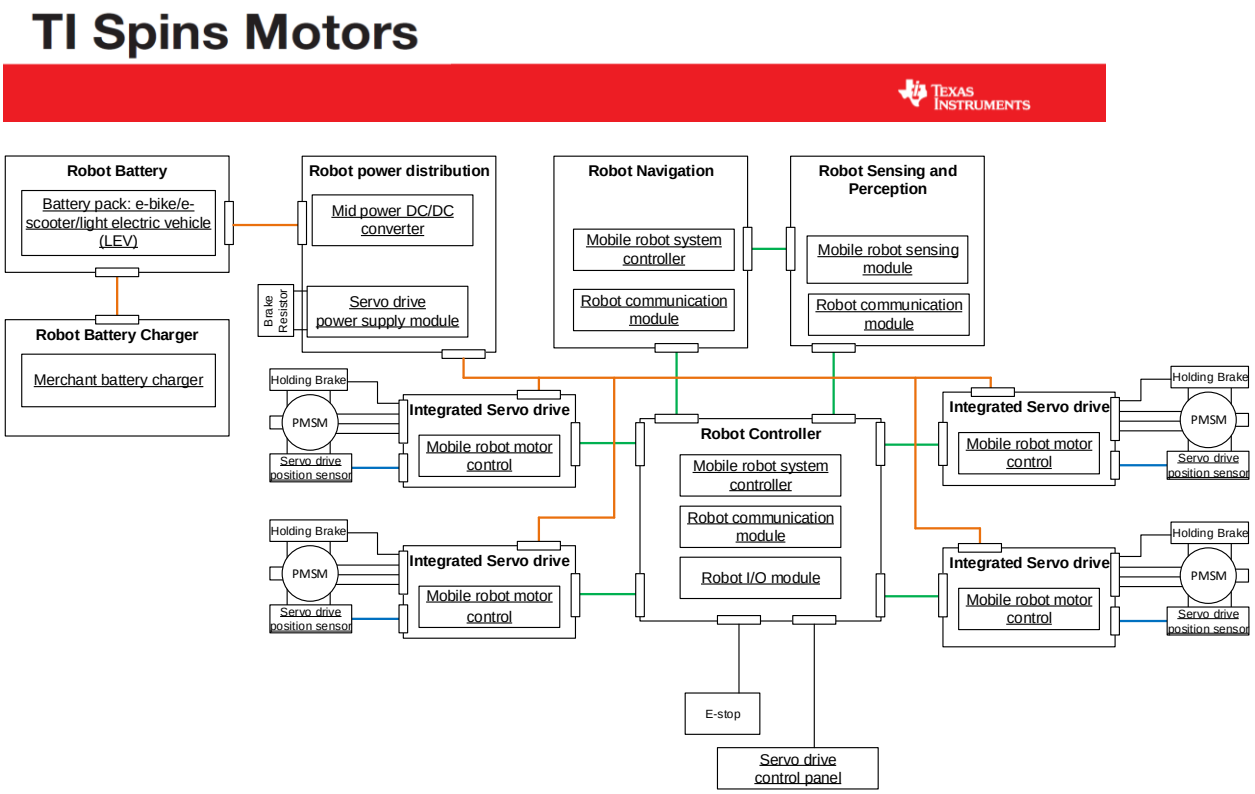


Figure 14: Block diagram of typical 3-phase motor drive system

To begin this lab connect the CAN high, CAN low and ground pins of the LaunchPads together as shown in Figure 15.

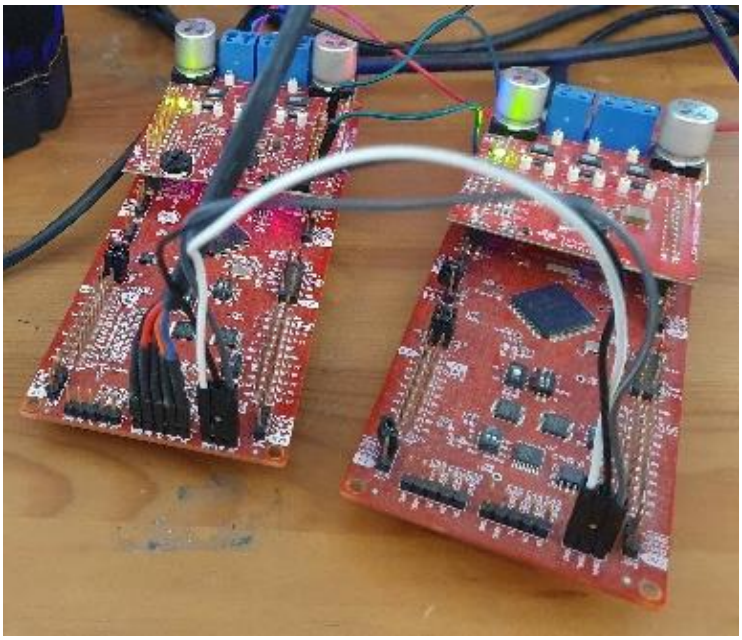


Figure 15: CAN Connection Between Two LaunchPads

1. Set DMC_BUILDLEVEL to DMC_LEVEL_3 in "servo_main.h", build and reload project for the host and slave controllers as described in the Section 2.3. The project on host and slave controllers use the same project.

2. Open the project in another CCS workspace for the host controller as shown in Figure 16.

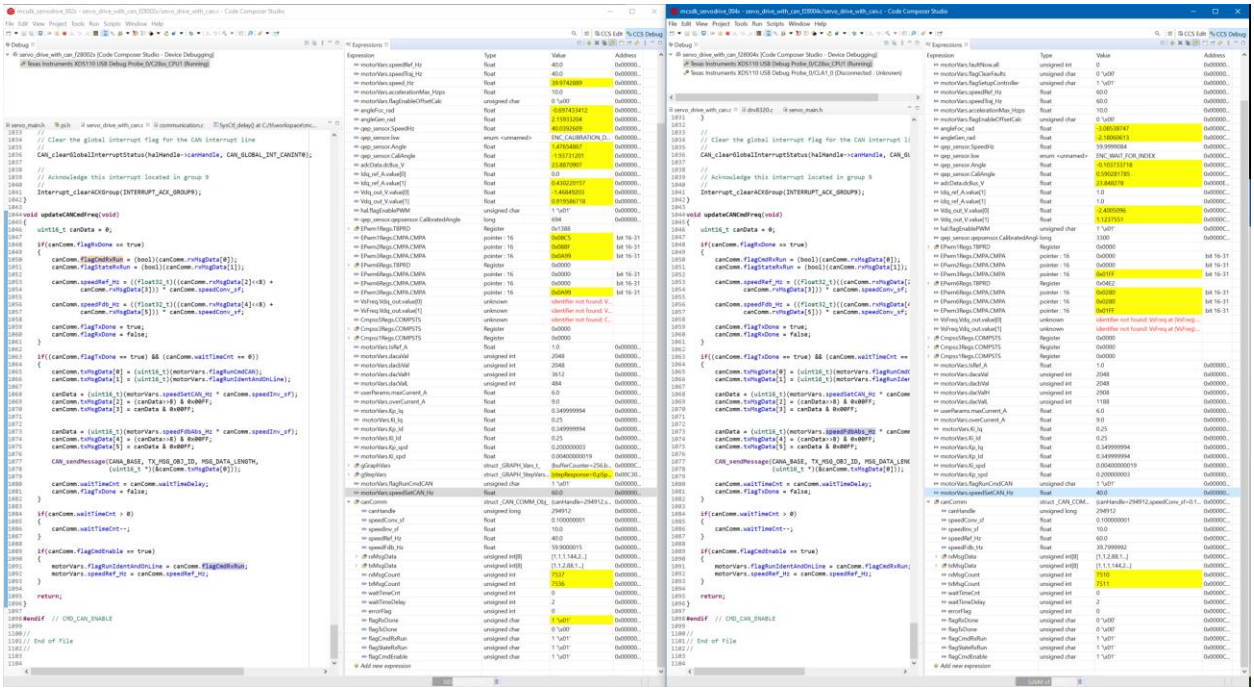


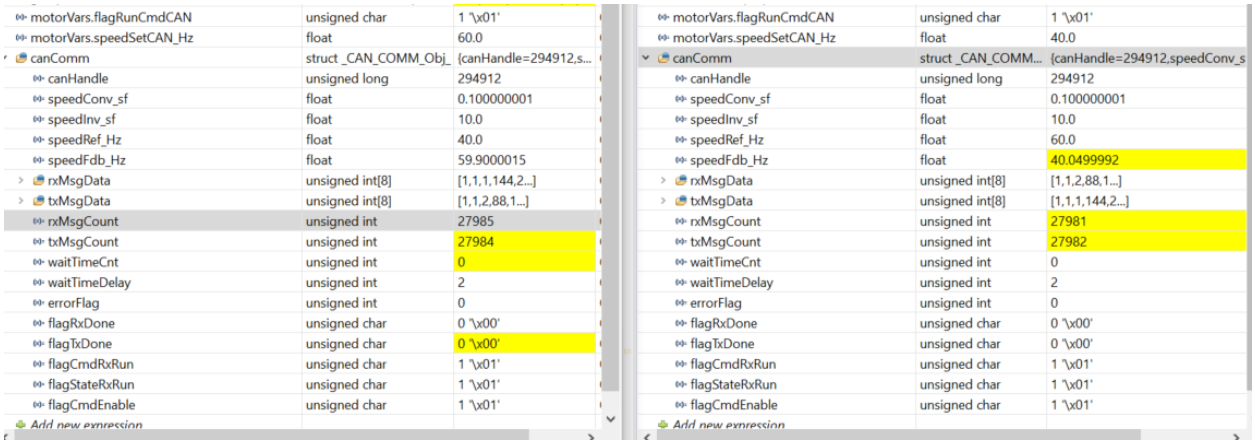
Figure 16: Open the lab project in two CCS workspace as host and slave

3. To view the reference speed by adding “canComm”, “motorVars.flagRunCmdCAN”, and “motorVars.speedSetCAN_Hz” to the Expressions Window.

4. To start the motor on another LaunchPad and BoosterPack by setting the variables in host controller.
   - motorVars.flagRunCmdCAN = 1 // 0-Stop the motor, 1-Start the motor
   - motorVars.speedSetCAN_Hz = 40.0 // Reference speed

5. The slave controller will receive the command and reference speed in the following variables to start run the motor as shown in Figure 17.
   - canComm.flagCmdRxRun is equal to motorVars.flagRunCmdCAN
   - canComm.speedRef_Hz is equal to motorVars.speedSetCAN_Hz

6. The host control will receive the running state and speed from the slave controller as shown in Figure 17.
   - canComm.flagStateRxRun
   - canComm.speedFdb_Hz



motorVars.flagRunCmdCAN	unsigned char	1 '\x01'
motorVars.speedSetCAN_Hz	float	60.0
canComm	struct _CAN_COMM_Obj_	{canHandle=294912,s...
canHandle	unsigned long	294912
speedConv_sf	float	0.100000001
speedInv_sf	float	10.0
speedRef_Hz	float	40.0
speedFdb_Hz	float	59.9000015
rxMsgData	unsigned int[8]	[1,1,1,144,2...]
txMsgData	unsigned int[8]	[1,1,2,88,1...]
rxMsgCount	unsigned int	27985
txMsgCount	unsigned int	27984
waitTimeCnt	unsigned int	0
waitTimeDelay	unsigned int	2
errorFlag	unsigned int	0
flagRxDone	unsigned char	0 '\x00'
flagTxDone	unsigned char	0 '\x00'
flagCmdRxRun	unsigned char	1 '\x01'
flagStateRxRun	unsigned char	1 '\x01'
flagCmdEnable	unsigned char	1 '\x01'
Add new expression		

motorVars.flagRunCmdCAN	unsigned char	1 '\x01'
motorVars.speedSetCAN_Hz	float	40.0
canComm	struct _CAN_COMM...	{canHandle=294912,speedConv_s
canHandle	unsigned long	294912
speedConv_sf	float	0.100000001
speedInv_sf	float	10.0
speedRef_Hz	float	60.0
speedFdb_Hz	float	40.0499992
rxMsgData	unsigned int[8]	[1,1,2,88,1...]
txMsgData	unsigned int[8]	[1,1,1,144,2...]
rxMsgCount	unsigned int	27981
txMsgCount	unsigned int	27982
waitTimeCnt	unsigned int	0
waitTimeDelay	unsigned int	2
errorFlag	unsigned int	0
flagRxDone	unsigned char	0 '\x00'
flagTxDone	unsigned char	0 '\x00'
flagCmdRxRun	unsigned char	1 '\x01'
flagStateRxRun	unsigned char	1 '\x01'
flagCmdEnable	unsigned char	1 '\x01'
Add new expression		

Figure 17: Send Command and Reference Speed, and Receive Running State and Speed