User's Guide

C29x Servo Drive With Incremental Encoder for PMSM FOC Evaluation

ABSTRACT

The *Servo Drive with Incremental Encoder for PMSM FOC Evaluation*, or *Servo Drive with QEP* for short, is a framework to develop, and experiment with, field-oriented control (FOC) of permanent magnet synchronous motors (PMSM). In this framework, the PMSM's position is sensed by an incremental encoder (QEP). The framework runs on a low-cost F29H85x-based LaunchPad plus a three-phase inverter evaluation module (EVM). Testing can be performed with a low-voltage PMSM.

Table of Contents

List of Figures

List of Tables

Trademarks

Code Composer Studio™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

Key features of the Servo Drive with QEP are:

- Leverages the F29H85x device family featuring the C29x CPU
- Demonstrates:
  - Sensored field oriented control (FOC) evaluation of permanent magnet synchronous motors (PMSM)
  - Position sensing via incremental encoder (QEP)
- Supports multiple three-phase inverter evaluation kits
- Facilitates step-by-step learning through incremental implementation
- Enables low-cost experimentation
- Provides a reference for your own design implementation
- Complete source provided

Note

This F29H85x project is a direct port of the *C28x Universal Servo Drive Lab* available in the C2000 Motor Control SDK. The *C28x Universal Servo Drive Lab* was originally based on the *C2000 Universal Motor Control Project*, with removal of sensorless algorithms. As such, the *C2000 Universal Motor Control Project and Lab Guide* can be a useful reference for motor control theory and additional details of the incremental build levels.

1.1 Hardware Block Diagram

The required hardware is:

- A low-cost LaunchPad featuring a C29x-based MCU
- A 3-phase inverter BoosterPack
- A low-voltage PMSM motor with incremental encoder
- External DC power supply

An optional digital-to-analog converter (DAC) evaluation module is supported for debug.

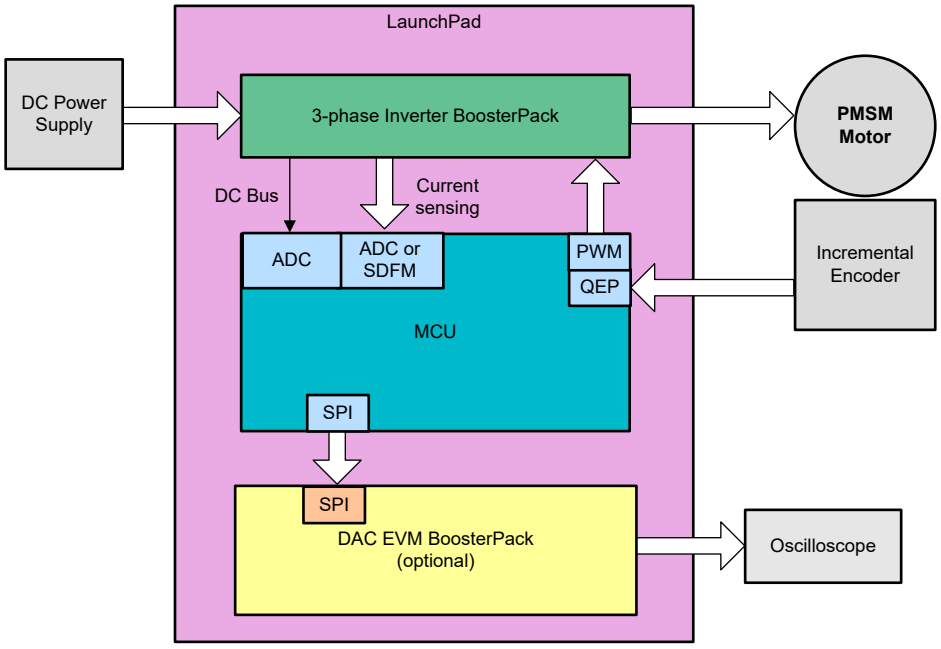

Figure 1-1. Hardware Block Diagram

Refer to:

- Section 2.1: list of specific part numbers supported.
- Section 2.2: hardware setup.

1.2 Software Flowchart

The Servo Drive with QEP software consists of:

- Startup routine: configures the device, calculates analog-to-digital converter (ADC) offsets
- Motor control interrupt service routine (ISR): started by the ADC end of conversion (EOC) or by the SDFM module.
- Background loop: periodically updates the motor control parameters.

Tools required to run the application:

- Code Composer Studio™
- F29x Motor Control SDK

Refer to: Section 2.3

- Required tool versions
- Instillation instructions
- Application software configuration
- Running the example.

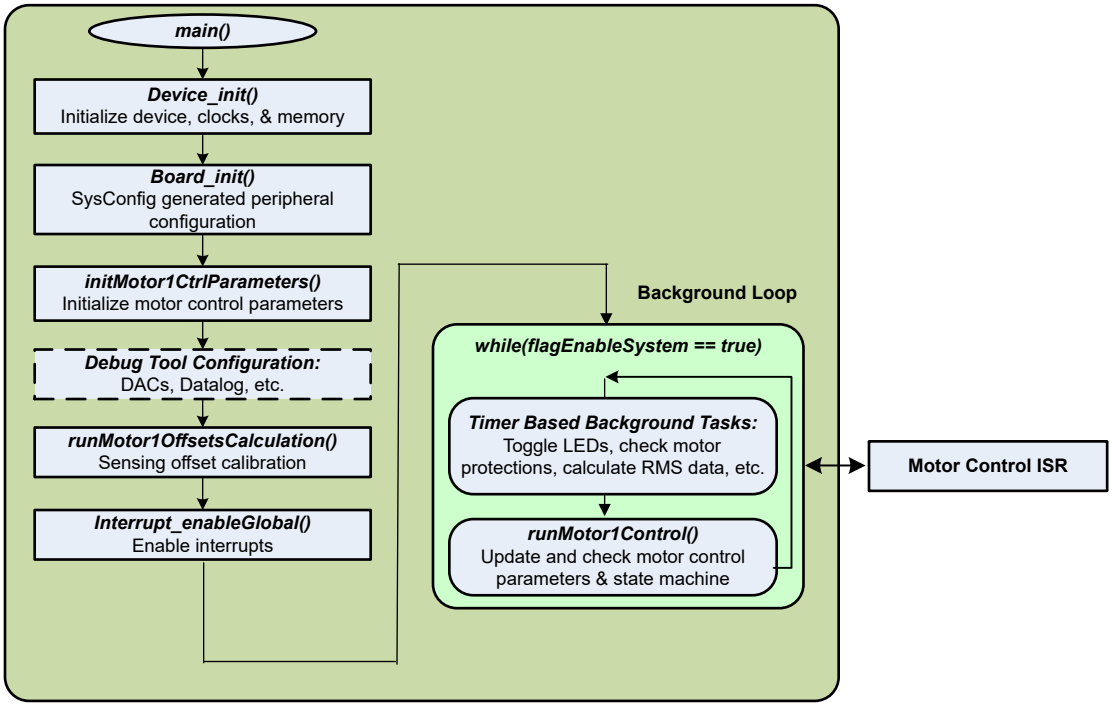

Figure 1-2. Flowchart: Startup and Background Loop

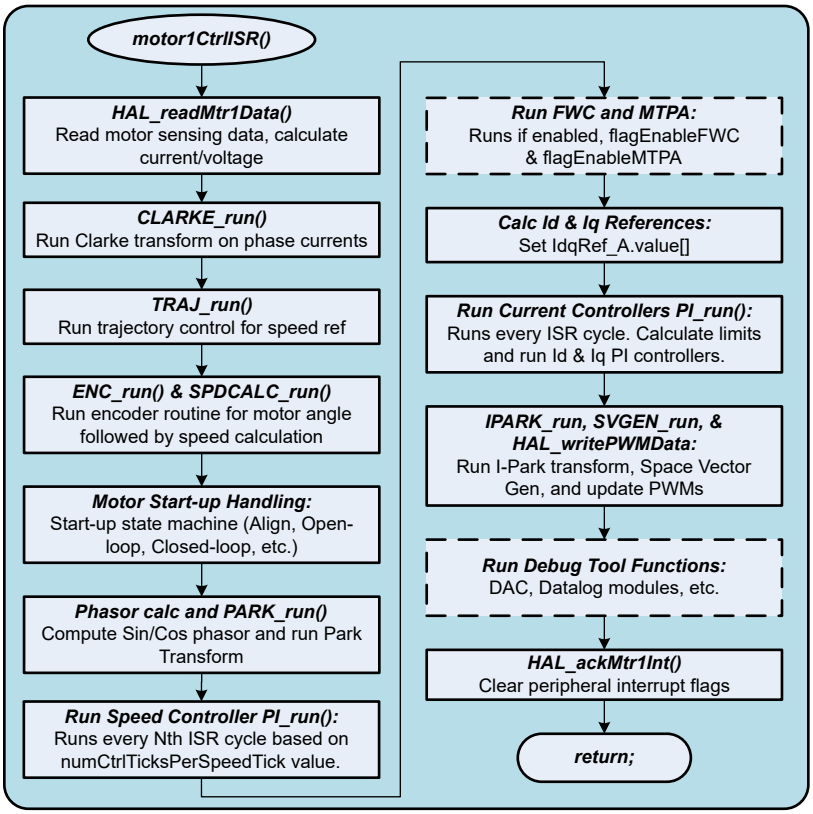

Figure 1-3. Flowchart: Motor Control Interrupt Service Routine

1.3 MCU Resources Used

Table 1-1. Resources Used

Resource		
CPU	C29x CPU 1	
PWM Channels	6	
ADC Channels	BOOSTXL-3PHGANINV	4: 3 current, 1 voltage
	BOOSTXL-LMG2100-MD	4: 3 current, 1 voltage
	BP-AMC0106-LMG-MD	1: voltage
SDFM Channels	BOOSTXL-3PHGANINV	0
	BOOSTXL-LMG2100-MD	0
	BP-AMC0106-LMG-MD	3: current sensing
eQEP Modules	1	
Control Methodology	Field oriented control (FOC)	
ISR Frequency	20kHz	
ISR Performance	Consumes 2.2 micro seconds with a 200MHz CPU clock and the motor spinning	
Memory Consumption	RAM: 12.4KB, FLASH: 60KB with compiler optimization level O2	

2 Running the Servo Drive With QEP on TI Hardware

This section is a guide to running the servo example on TI hardware.

2.1 Supported Hardware

Table 2-1 lists the supported inverter evaluation kits. Each inverter board has been tested with:

- C2000 MCU Evaluation Module: LAUNCHXL-F29H85X
- Motor kit: LVSERVOMTR (Encoder Embedded)
- DAC EVM (optional): DAC128S085EVM: 12-bit 8 Channel DAC Evaluation Module

Table 2-1. Inverter Evaluation Kits Supported

Inverter Board		
Part Number	**Description**	**Current Sensing Topology**
BOOSTXL-3PHGANINV	12~60V, 7A RMS, 10A Peak, 3ph GaN inverter	Three shunt-based inline motor phase current sensing
BOOSTXL-LMG2100-MD	12~60V, 27A RMS without heat sink, 3ph GaN inverter	Three shunt-based inline motor phase current sensing
BP-AMC0106-LMG-MD	12~60V, 27A RMS without heat sink, 3ph GaN inverter	Three shunt-based inline sensing with Delta-Sigma Modulator

2.2 Hardware Setup

This section describes how to setup the hardware to run the servo drive:

- Configure the LaunchPad switches
- Optional: connect the DAC EVM to the LaunchPad
- Connect the inverter board to the LaunchPad
- Connect the motor to the LaunchPad + inverter board

2.2.1 LAUNCHXL-F29H85X Setup

The LAUNCHXL-F29H85X is a low-cost development board that supports the connection of two BoosterPack plug-in modules. For documentation such as user's guide and schematics, see the tool folder.

Configure the LaunchPad as follows:

- BOOT: set to flash (UP, UP)
- ADC reference: set to external reference (UP)
- QEP: connect Q1 to J12 (RIGHT switch UP)
- Encoder connection: J12 connects to the motor harness J4. The J12 ground pin is on the LEFT.
- Connect a USB cable to the on-board USB connector on the C2000 Launchpad. This connection supplies power to the LaunchPad and enables an isolated JTAG connection to the C2000 device. Leave the cable disconnected until testing begins.

Note

Use of parallel I/O boot can lead to random F29H85x device resets. The BOOT mode switch **must not** be set to parallel I/O boot unless a host is present to drive the control lines.

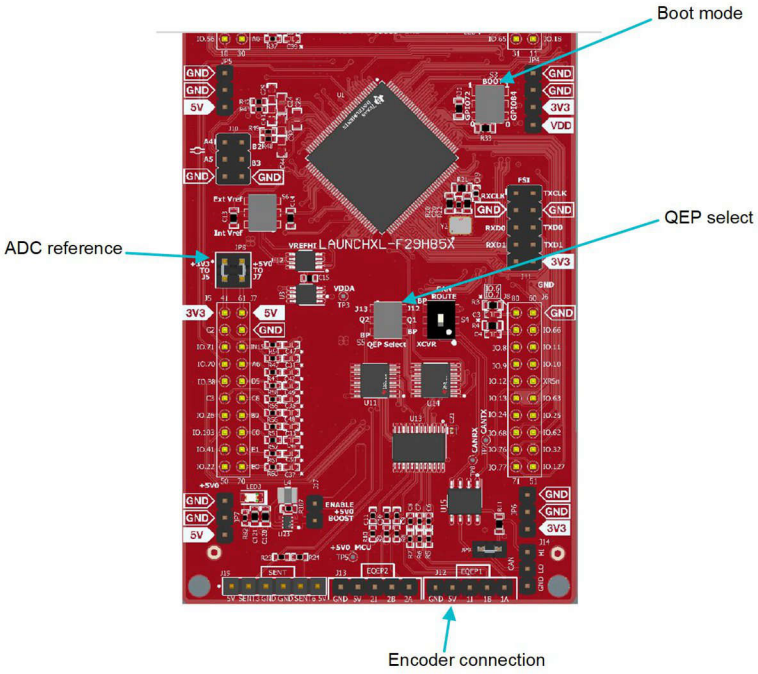

Figure 2-1. LAUNCHXL-F29H85X Configuration

2.2.2 DAC128S085EVM Setup (Optional)

The DAC128S085EVM evaluation module is a useful debug tool. The board converts digital data, received through the SPI, to analog values that can be examined on an oscilloscope. Typical variables sent to the DAC are angle, voltage, and current measurements.

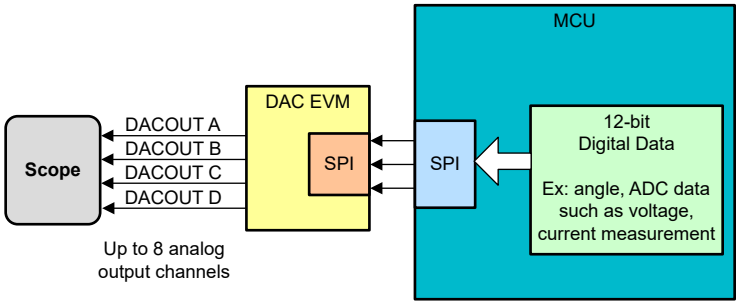

Figure 2-2. DAC128S Module Block Diagram

Setup the DAC EVM:

1. Connect to LaunchPad connectors J5/J7 and J6/J8 (BoosterPack site2).
2. On the DAC EVM, connect pin JA-2 to pin JB-2 using a jumper wire.

For more information:

- For EVM documentation, schematics, visit the DAC128S085EVM tool page.
- For usage in the Servo Drive with QEP project, see Section 2.3.4.2.

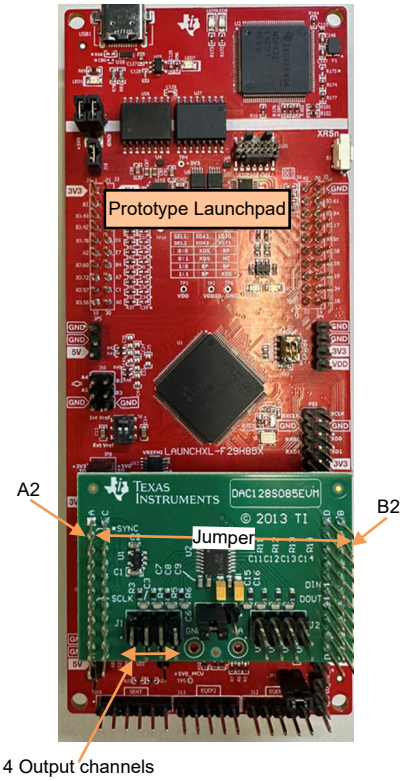

Figure 2-3. DAC128S EVM Connected to F29H85X LaunchPad

2.2.3 BOOSTXL-3PHGANINV Setup

Note

If using the optional DAC128S085EVM, add risers to elevate the BOOSTXL-3PHGANINV above the DAC EVM.

1. Set the LaunchPad jumpers as described in Section 2.2.1.
2. Connect BOOSTXL-3PHGANINV to J1/J3 and J4/J2 of the LaunchPad.
3. Connect the motor and encoder as described in Section 2.2.6. For initial testing of the hardware setup (Section 2.4.2) leave the motor disconnected from the harness.
4. Connect a DC power supply with a voltage ranging from 12V to 60V to the inverter board's voltage supply pins. Leave the power supply off for now.

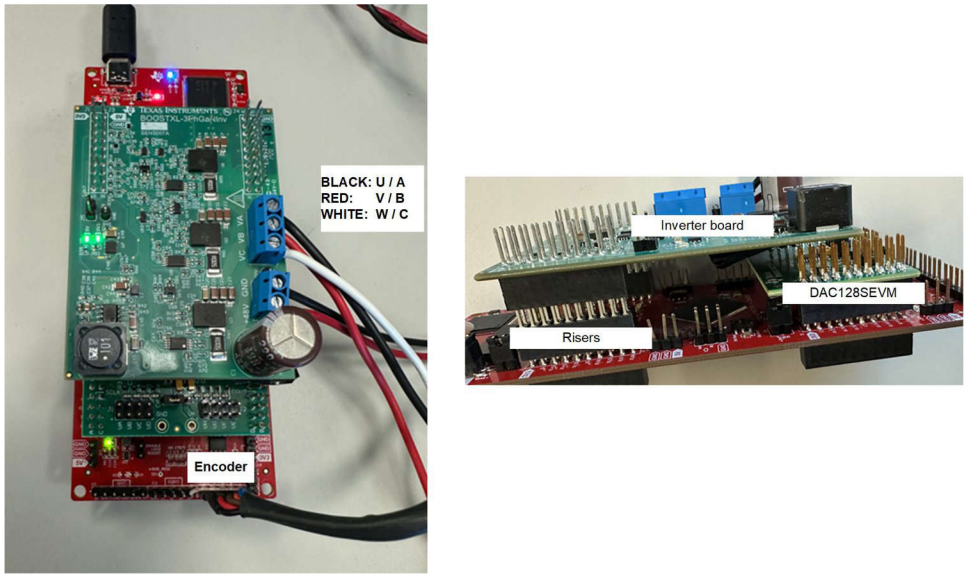

Figure 2-4. BOOSTXL-3PHGANINV Hardware Setup

2.2.4 BOOSTXL-LMG2100-MD Setup

Note

For this BoosterPack, connecting to the bottom of the LaunchPad is suggested.

1. Set the LaunchPad jumpers as described in Section 2.2.1.
2. Connect BOOSTXL-LMG2100-MD to J1/J3 and J4/J2 of the LaunchPad. Use the bottom connectors.
3. Connect the motor and encoder as described in Section 2.2.6. For initial testing of the hardware setup (Section 2.4.2) leave the motor disconnected from the harness.
4. Connect a DC power supply with a voltage ranging from 12V to 60V to the inverter board's voltage supply pins. Leave the power supply off for now.

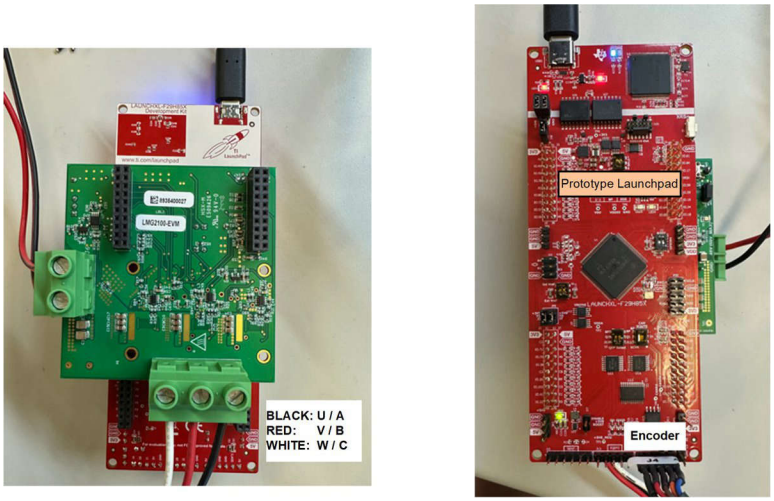

Figure 2-5. BOOSTXL-LMG2100-MD Hardware Setup

2.2.5 BP-AMC0106-LMG-MD

1. Set the LaunchPad jumpers as described in Section 2.2.1.
2. Connect BP-AMC0106-LMG-MD to J1/J3 and J4/J2 of the LaunchPad.
3. Connect the motor and encoder as described in Section 2.2.6. For initial testing of the hardware setup (Section 2.4.2) leave the motor disconnected from the harness.
4. Connect a DC power supply with a voltage ranging from 12V to 60V to the inverter board's voltage supply pins. Leave the power supply off for now.

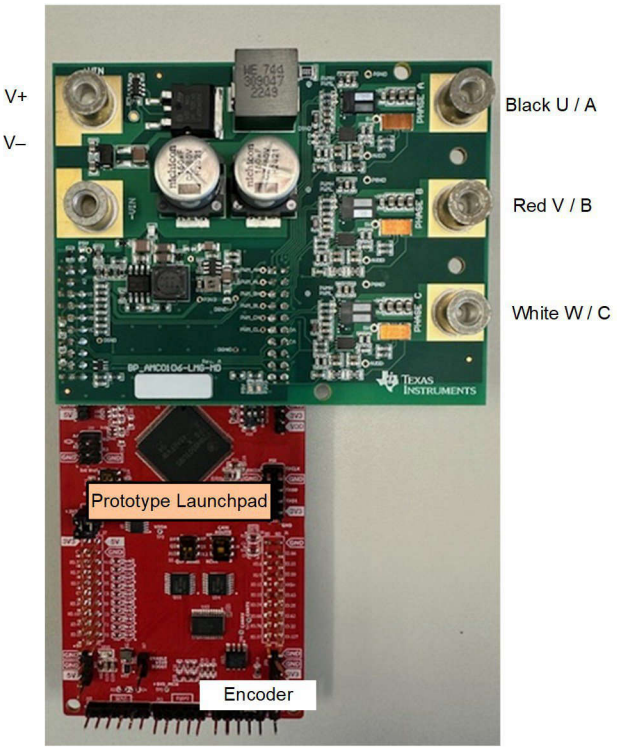

Figure 2-6. BP-AMC0106-LMG-MD Hardware Setup

2.2.6 Motor Setup

Table 2-2. Connections for Inverter Boards and Motor

		LVSERVOMTR
Motor Phase Lines	U / A	BLACK (16AWG)
	V / B	RED (16AWG)
	W / C	WHITE (16AWG)
Encoder	**J12 of LAUNCHXL-F29H85X**	**J4 of the Motor Harness**
	GND (J12-1 LEFT)	BLACK (J4-1)
	+5V (J12-2)	RED (J4-2)
	1A (J12-3 QEP1_A)	BROWN (J4-3)
	1B (J12-4 QEP1_B)	ORANGE (J4-4)
	1I (J12-5 QEP1_I RIGHT)	BLUE (J4-5)

2.3 Lab Software

Note

As provided, the projects use compiler V2.0.0.0.**STS**. While this version is a **sort-term support** release, V2.0.0.0.STS includes performance improvements, and bug fixes, over V1.0.0.0.LTS. The intention is to use V2.0.0.0.STS for early evaluation only. Please move to an long-term support (<version>**.LTS**) release when available.

Follow the steps in this section to run the Servo with QEP application on hardware.

- Setup the software development environment.
- Configure the application software
- Become familiar with key watch variables
- Build, and test, the project step-by-step with incremental functionality

2.3.1 Software Development Environment

Table 2-3. Development Setup Steps

Step	Item	Details
1	Code Composer Studio	• V20.2 or later • Download: Code Composer Studio (CCS)
2	Learn about CCS	• Code Composer Studio Academy
3	F29 Motor Control SDK	• V1.0 or later • Download: F29X-MOTOR-CONTROL-SDK
4	Install Python and OpenSSL	• Getting Started Guide in the F29x SDK includes prerequisites highlighted in the Build, Load and Run section. • <install>\c29_sdk\mcu_sdk_f29h85x\docs\html\GETTING_STARTED.html
5	Open CCS and import the project.	• *File → Import Project(s)* • <install>\solutions\servo_drive_qep • Select the project matching your inverter hardware. – Example: *servo_drive_qep_**3phGanInv*** corresponds to BOOSTXL-**3phGanInv**

Note

Make sure you have the correct version of Python and OpenSSL installed before proceeding (step 4).

2.3.2 Project Orginization

Once imported into Code Composer Studio, the Servo Drive with QEP project has the folder structure described in Table 2-4.

Table 2-4. Folder Structure

Folder	Includes
README.html	Overview of the project. Links to the SDK documentation.
sys_main.c	main() function and background loop
sys_main.h	#include of other header files, definition of SYSTEM_Vars_t
sys_settings.h	#define DMC_BUILDLEVEL DMC_LEVEL_x. This define determines what control functionality is used: open loop, closed current loop, closed current + speed loop.
libraries/	Typical FOC modules including Park, Clark, and inverse Park and Clark transforms. Support for incremental encoder, datalog and the DAC EVM.
src_board/	Device specific drivers to run the servo drive. To migrate the project to another board, changes can be made primarily to: *<board>.syscfg*, *hal.c, hal.h,* and *user_mtr1.h*.
src_control/	Motor drive control files that call the motor control core algorithm functions within the interrupt service routines and background tasks. motor1_drive.c includes the *motor1CtrlISR()*.

Table 2-4. Folder Structure (continued)

Folder	Includes
src_device/	Device specific startup file, driver library, linker command file.

2.3.3 Configuration of the Software

1. **Predefined Symbols:**

   Compiler predefine macros are provided to configure the software. A list of options is shown in Table 2-5. To modify, add, or remove macros:

   right click on project → Properties → Build → C2000 Compiler → Predefined Symbols

   Note

   A pre-define symbol with suffix "_N" is disabled. For example, field weakening control (FWC) is enabled by removing the "_N" on "MOTOR1_FWC_N" to change the predefine to "MOTOR1_FWC".

2. **Motor model definition:**

   Defined in *src_board/user_mtr1.h* and *src_board/user_common.h* files.

   Locate the definition for USER_MOTOR1. Confirm the definition matches the motor being tested. Only the Teknic_M2310PLN04K was tested for this release.

```
#define USER_MOTOR1 Teknic_M2310PLN04K
```

Table 2-5. Predefine Definitions

Predefine	Description	Required or Optional	Default
MOTOR1_ENC	Incremental encoder	Required	Enabled
ADC_EXT_REF	ADC uses the external reference. If this predefine is disabled, the SysCfg file must be modified to use the internal reference.	Recommended	Enabled
MOTOR1_FWC	Field weakening control. Usually enabled along with MTPA.	Optional	Disabled
MOTOR1_MTPA	Maximum torque per amphere. Usually enabled along with FWC.	Optional	Disabled
DATALOG_EN	Data export through datalog buffers	Optional	Disabled
DAC128S_ENABLE	Data export through the DAC EVM	Optional	Disabled
DAC_ON_CHIP_ENABLE	Data export through the on-chip DAC	Optional	Disabled

2.3.4 Debug Interfaces

2.3.4.1 Datalogging

The datalog software module supports logging into software buffers. These buffers can then be displayed using Code Composer Studio's graphing function.

The datalog library is documented the SDK documentation. *<install>/docs/html_guide/index.html* under Libraries.

1. Add the symbol DATALOG_EN to the project's predefined symbols.
2. Select the number of buffers, the size of the buffers, and the datatype in *libraries/utilities/datalog_input.h*.
3. Call the setupDatalog() function in *src_board/diagnostics.c*:

```
// Initialize Datalog
datalogHandle = DATALOG_init(&datalog, sizeof(datalog), manual, 0, 1);
DATALOG_Obj *datalogObj = (DATALOG_Obj *)datalogHandle;
datalogObj->flag_enableLogData = false;
datalogObj->flag_enableLogOneShot = false;
....
datalogObj->iptr[0] = (float32_t*) &motorVars_M1.senseData.I_A.value[0];
datalogObj->iptr[1] = (float32_t*) &motorVars_M1.senseData.I_A.value[1];
datalogObj->iptr[2] = (float32_t*) &motorVars_M1.angleFOC_rad;
```

This function can be modified to output different signals. For this project, you can monitor the angle and the sensed current values as shown here.

4. Start the datalog when appropriate. This can be either one-shot or continuous log.

```
#if defined(DATALOG_EN)
    // datalog.flag_enableLogOneShot = true;
   datalog.flag_enableLogData = true;
#endif  // DATALOG_ENABLE
```

5. Send data to the datalog:

```
#if defined(DATALOG_EN)
    DATALOG_update(datalogHandle);
#endif  // DATALOG_ENABLE
```

2.3.4.2 Digital to Analog Converters

The DAC128S software module supports the DAC EVM described in Section 2.2.2. The software module is capable of converting up to eight software variables to 12-bit integer values and transmitting the data through the SPI to the DAC EVM.

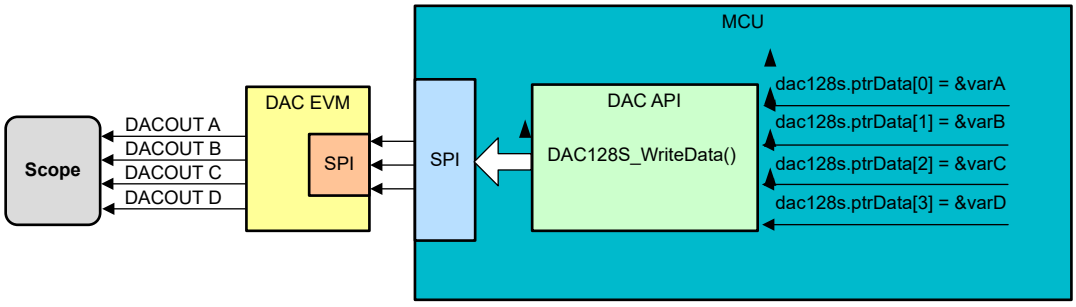

Figure 2-7. DAC128S Block Diagram

To use the EVM, follow these steps:

1. Connect the hardware as described in Section 2.2.2.
2. Add the symbol DAC128S_ENABLE to the project's predefined symbols.
3. Select the number of channels in *libraries/dacs/dac128s085.h*.
4. Call the setupDAC1285S() function in *src_board/diagnostics.c*:

```
dac128s.ptrData[0] = &motorVars_M1.angleENC_rad;                    // CH_A
dac128s.ptrData[1] = &motorVars_M1.senseData.I_A.value[0];          // CH_B
dac128s.ptrData[2] = &motorVars_M1.senseData.I_A.value[1];          // CH_C
dac128s.ptrData[3] = &motorVars_M1.senseData.I_A.value[2];          // CH_D
```

   a. This function can be modified to output different signals. For this project, you can monitor the angle and the sensed current values as shown here.

5. Send the data.

```
// Write the variables data value to DAC128S085 through the SPI
DAC128S_writeData(dac128sHandle);
```

   a. The code sends the data periodically during the execution of the control interrupt (motor1ctrlISR()) located in *src_control/motor1_drive.c*.
   b. The number of the DAC outputs transmitted is determined by the configuration in *dac128s085.h*.

The EVM has an eight channel, 12-bit digital-to-analog converter (DAC). As provided, the output number is set to 4 since most oscilloscopes only have four probes. More outputs consume time to convert and transmit the data, which can negatively effect the time to spend on other tasks. The user can, however, set the output number between 1 and 8 by changing definitions in *libraries/dacs/dac128s085.h* file.

2.4 Testing the Project in Incremental Steps

The servo drive system can be tested in incremental stages. Each stage is enabled by defining DMC_BUILDLEVEL in *sys_settings.h* and rebuilding the project. In the code example shown "Step 4: closed speed loop and current loop" is enabled by the DMC_BUILDLEVEL.

```
//=============================================================================
// Incremental Build options for System check-out
//=============================================================================
#define DMC_LEVEL_1     1   //Y 50% duty, offset calibration and verify phase shift
#define DMC_LEVEL_2     2   //Y open loop control to check sensing signals
#define DMC_LEVEL_3     3   //Y closed current loop to check the hardware settings
#define DMC_LEVEL_4     4   //Y run with sensored FOC

#define DMC_BUILDLEVEL  DMC_LEVEL_4
```

1. **Step 1: Hardware setup validation.**

   #define DMC_BUILDLEVEL DMC_LEVEL_1

   Verify the ADC offset calibration, ePWM 50% duty output, deadband, and phase shift.
2. **Step 2: Open loop control.**

   #define DMC_BUILDLEVEL DMC_LEVEL_2

   Spin the motor using open loop control to verify the motor current and voltage sensing signals.
3. **Step 3: Close the current loop.**

   #define DMC_BUILDLEVEL DMC_LEVEL_3

   Spin the motor with a closed current loop to validate current sensing and the current control.
4. **Setp 4: Close the speed loop and the current loop.**

   #define DMC_BUILDLEVEL DMC_LEVEL_4

   The final step is to run the motor with both the speed loop and the current loop closed.

2.4.1 Watch Variables

The structure motorVars_M1 has references to most variables that are related to controlling the servo drive. Table 2-6 is a recommended list to populate the CCS watch window.

Note

Code Composer Studio V20.2 does not have a method to export/import the watch window variables. This feature is expected to be implemented in V20.3.

Table 2-6. Watch Variable Overview

Variable	Description
motorVars_M1.ISRCount	Increments once each motor ISR execution
systemVars.flagEnableSystem	Transitions from 0 to 1 automatically
motorVars_M1.flagEnableRunAndIdentify	• Set to 1 to start the motor after the flagEnableSystem variable is automatically set. • Set to 0 to disable the PWMs. The MCU can then be halted. Note: For build 1 this starts the PWMs for inspection.
motorVars_M1.flagRunIdentAndOnLine	Changes to 1 if there are no faults
motorVars_M1.motorState	Shows the current state of the motor control such as: • MOTOR_STOP_IDLE • MOTOR_FAULT_STOP • MOTOR_ALIGNMENT • MOTOR_OL_START • MOTOR_CL_RUNNING • MOTOR_CTRL_RUN

Table 2-6. Watch Variable Overview (continued)

Variable	Description
motorVars_M1.estimatorMode	ESTIMATOR_MODE_ENC for incremental encoder
motorVars_M1.faultMtrUse.all	Value is non-zero if there is an over-current fault
motorVars_M1.faultMtrUse.bit	Expand and check for fault flags. Of the faults that can turn off the motor, these are currently active. (faultMtrMask applied to faultMtrNow)
motorVars_M1.faultMtrNow.bit	Of all possible faults, these are currently active.
motorVars_M1.faultMtrMask.bit	Of the possible faults, these are the ones that can turn off the motor.
motorVars_M1.senseData.VdcBus_V	The near the DC bus voltage
motorVars_M1.senseData.offset_I	• Current offset values used for current sensing by the ADC • For ADC sensing: 2048 (half of a 12-bit ADC scale value) • For SDFM sensing: these are very small numbers
motorVars_M1.speedRef_Hz	• The reference speed of the motor. • Change this variable to increase or decrease the speed. • A negative value reverses the direction • Not used in build level 1
motorVars_M1.speed_Hz	Current spee of the motor. Not used in build level 1.
motorVars_M1.overCurrent_A	By decreasing the value, the fault protection by the CMPSS modules can be verified.

2.4.2 Step 1 Hardware Setup Validation

Objectives:

- Use SysConfig and the HAL object to initialize the peripherals of the MCU for the servo drive hardware.
- Verify the PWM and ADC driver modules and hardware connections
- Verify the sensing feedback values from the inverter: phase current offset and DC bus voltage

In this step, the PWM runs with a fixed 50% duty cycle and the motor is not connected.

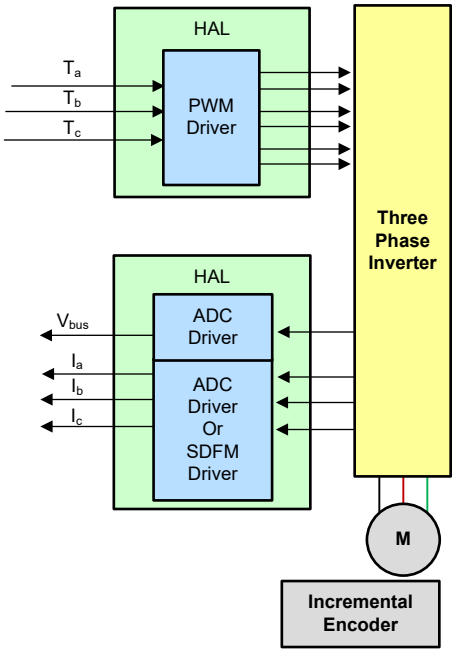

Figure 2-8. Step 1: Hardware Setup Validation Block Diagram

2.4.2.1 Build, Load and Run Project

Follow the steps in Table 2-7 and Table 2-8 to verify the hardware setup. The steps in Table 2-7 also apply to the open loop, closed current loop and the closed current+speed loop builds.

Table 2-7. Setup Steps

Step	Description	Notes
1	Hardware setup	Refer to Section 2.2
2	Connect PC to LaunchPad via USB	Powers the LaunchPad and establishes JTAG connection
3	Power the inverter board	External DC supply
4	Import the project	In CCS: *File → Import Project(s)*
5	Set DMC_BUILDLEVEL	• *sys_settings.h* • #define DMC_BUILDLEVEL DMC_LEVEL_x where x = 1, 2, 3, 4 • Do not connect the motor when running DMC_LEVEL_1
5	• Connect to the LaunchPad • Build the project • Program the flash	In CCS: *Run → Debug Project* (or F5).

Table 2-8. Hardware Verification Procedure

Step	Description	
1	Start execution in CCS: *Run → Continue* (or F5)	
2	**Confirm:** variable increments	*motorVars_M1.ISRCount*
3	**Confirm:** variable changes to 1	*systemVars.flagEnableSystem*
4	**Set:** to 1 to start the ePWMs	*motorVars_M1.flagEnableRunAndIdentify*
5	**Confirm:** variable changes to 1	*motorVars_M1.flagRunIdentAndOnLine*
6	**Confirm:** approximately half the ADC scale	*motorVars_M1.senseData.offset_I.value[]*
7	**Confirm:** matches hardware DC bus voltage	*motorVars_M1.senseData.VdcBus_V*
8	**Confirm:** PWM duty and switching frequency with an oscilloscope	
9	**Clear:** (set to 0) to disable the ePWM	*motorVars_M1.flagEnableRunAndIdentify*
10	The CPU can now be halted and CCS disconnected.	
11	Turn off power to the inverter board	
12	Disconnect the LaunchPad from the PC	

If any of the steps result in unexpected results, check the following:

1. PWM
   a. The PWM frequency is configured in the .syscfg file
   b. The #define USER_M1_PWM_FREQ_kHz in *usr_mtr1.h* needs to match the .syscfg configuration.
2. The motor driver board is properly setup and powered
3. The project imported matches the motor driver board. The board is reflected in the name.
4. Check the switches on the LaunchPad.

2.4.3 Step 2 Open Loop Control

Objectives:

- Run the system with open-loop control
- Validate the current and voltage sensing circuits
- Validate that the incremental encoder is correctly configured and connected

Note

In open-loop control:

- The current values sensed by the ADC (or SDFM) are only used for verification and validation. The values are not actually used to control the motor.
- Incremental encoder data is used to estimate the speed for validation. This speed is not used to control the motor.

Note

The number of slots per rotation for the encoder must be provided. This is required to correctly convert the encoder signal into an angle. Check the definition of **USER_MOTOR1_NUM_ENC_SLOTS** in *user_mtr1.h.*

An incorrect value results in the motor spinning faster, or slower. Note that this value is the number of slots on the encoder, not the resulting number of counts after figuring the quadrature accuracy.

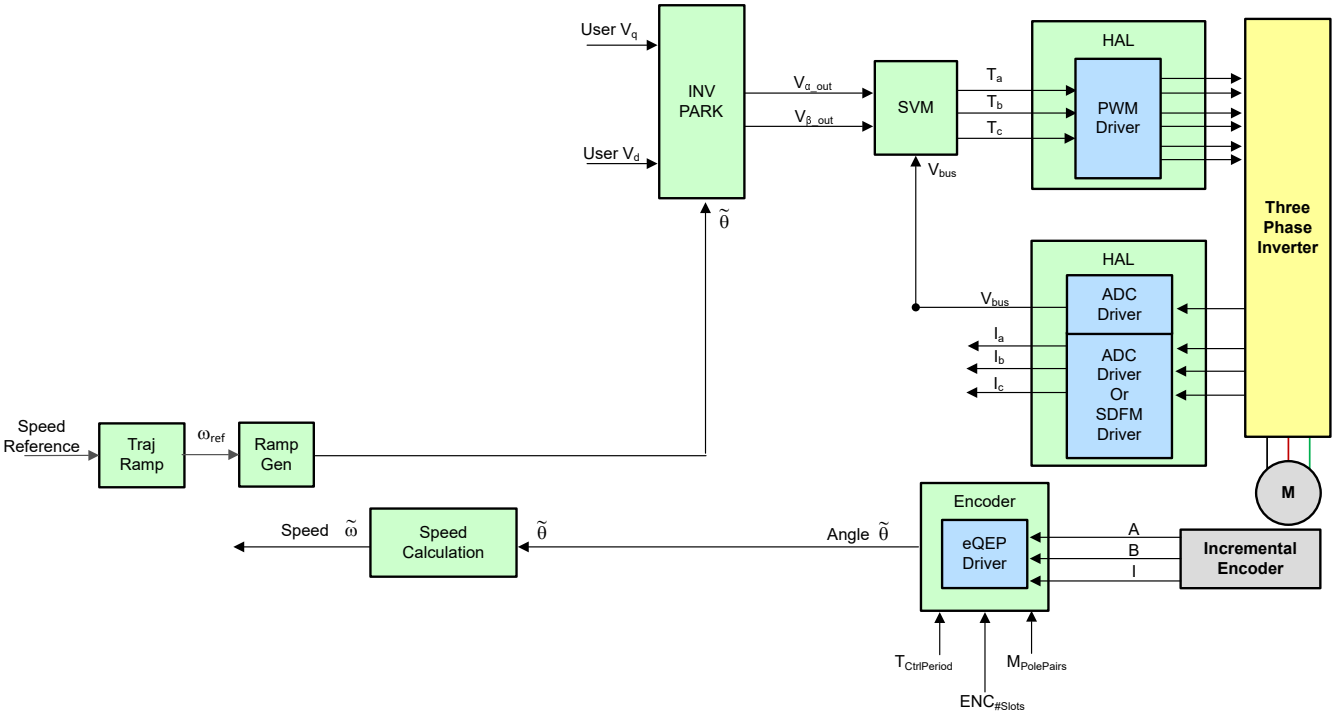

Figure 2-9. Step 2: Open Loop Control Block Diagram

2.4.3.1 Build, Load and Run Project

Follow the setup in Table 2-7 with:

- The motor connected to the harness.
- #define DMC_BUILDLEVEL DMC_LEVEL_2.

Control Test Procedure has the control test procedure. This test procedure is re-used for the closed-current loop test, and for the closed speed+current loop test.

Table 2-9. Control Test Procedure

Step	Description	
1	Start execution in CCS: *Run* → *Continue* (or F5)	
2	**Confirm:** variable increments	*motorVars_M1.ISRCount*
3	**Confirm:** variable changes to 1	*systemVars.flagEnableSystem*
4	**Set:** to 1 to start the ePWMs	*motorVars_M1.flagEnableRunAndIdentify*
5	**Confirm:** • Variable changes to 1 (no faults) • The motor begins to spin	*motorVars_M1.flagRunIdentAndOnLine*
6	**Confirm:** motor peed ramps up to the reference speed	• *motorVars_M1.speed_Hz* • *motorVars_M1.speedRef_Hz*
7	**Modify:** • Reference speed and observe current speed change • Negative reference speed changes the direction	
9	**Monitor:** Phase currents and angle • Use datalog buffers, the on-chip DAC or the DAC EVM. • Compare currents with actual phase currents measured with a current probe	
10	**Clear:** (set to 0) to disable the ePWM	*motorVars_M1.flagEnableRunAndIdentify*
10	The CPU can now be halted and CCS disconnected.	
11	Turn off power to the inverter board	
12	Disconnect the LaunchPad from the PC	

2.4.4 Step 3 Close the Current Loop

Objectives for this build level:

- Evaluate the closed current loop operation of the motor.
- Verify the current sensing parameter settings

In this build level, the motor is controlled by using i/f control. The rotor angle is generated from ramp generator module.

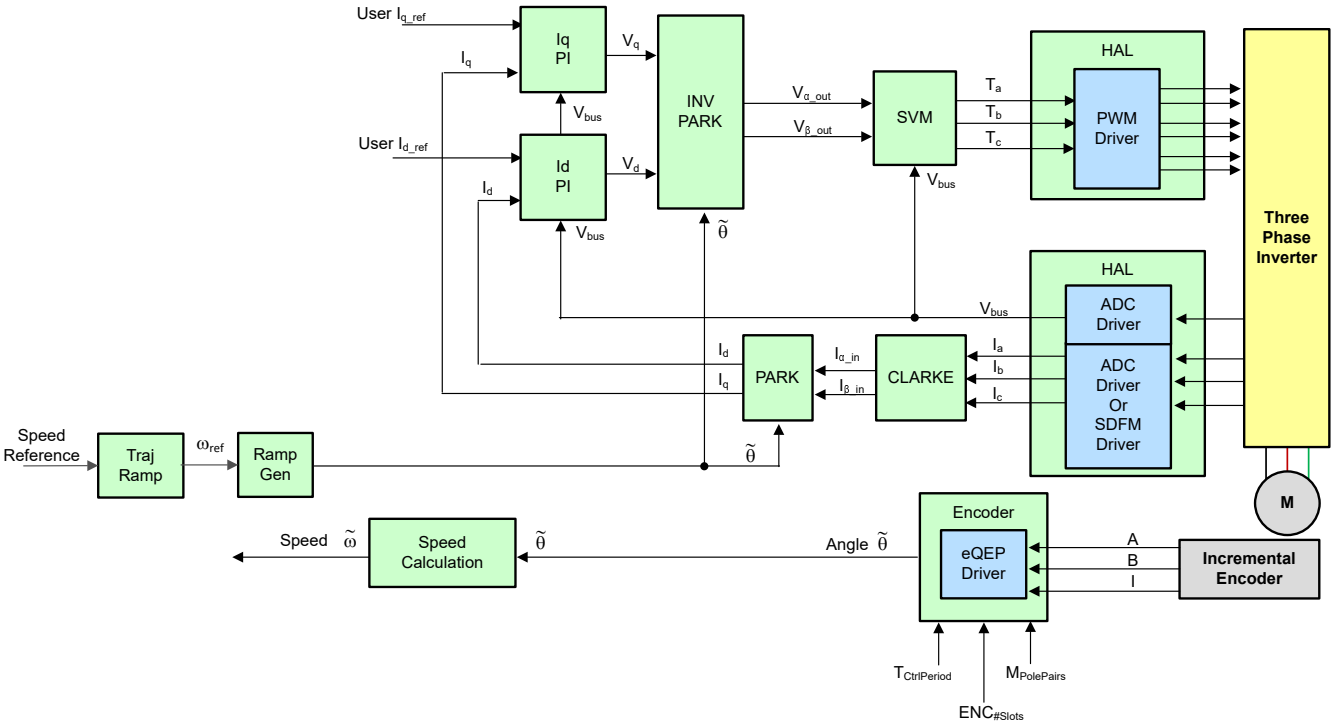

Figure 2-10. Step 3: Current Closed Loop Control Block Diagram

2.4.4.1 Build, Load and Run Project

To verify the motor with current closed-loop control:

- Follow the setup in Table 2-7. This time:
  - Connect the motor to the harness
  - Specify #define DMC_BUILDLEVEL DMC_LEVEL_3 in *sys_settings.h*
- Follow the test procedure in Table 2-9

2.4.5 Step 4 Close the Speed and Current Loop

Objectives learned in this build level:

- Evaluate the complete motor drive with incremental encoder-based field-oriented control (FOC).

In this build level, the outer speed loop is closed with the inner current loop. The rotor angle is measured from an incremental encoder connected to the QEP.

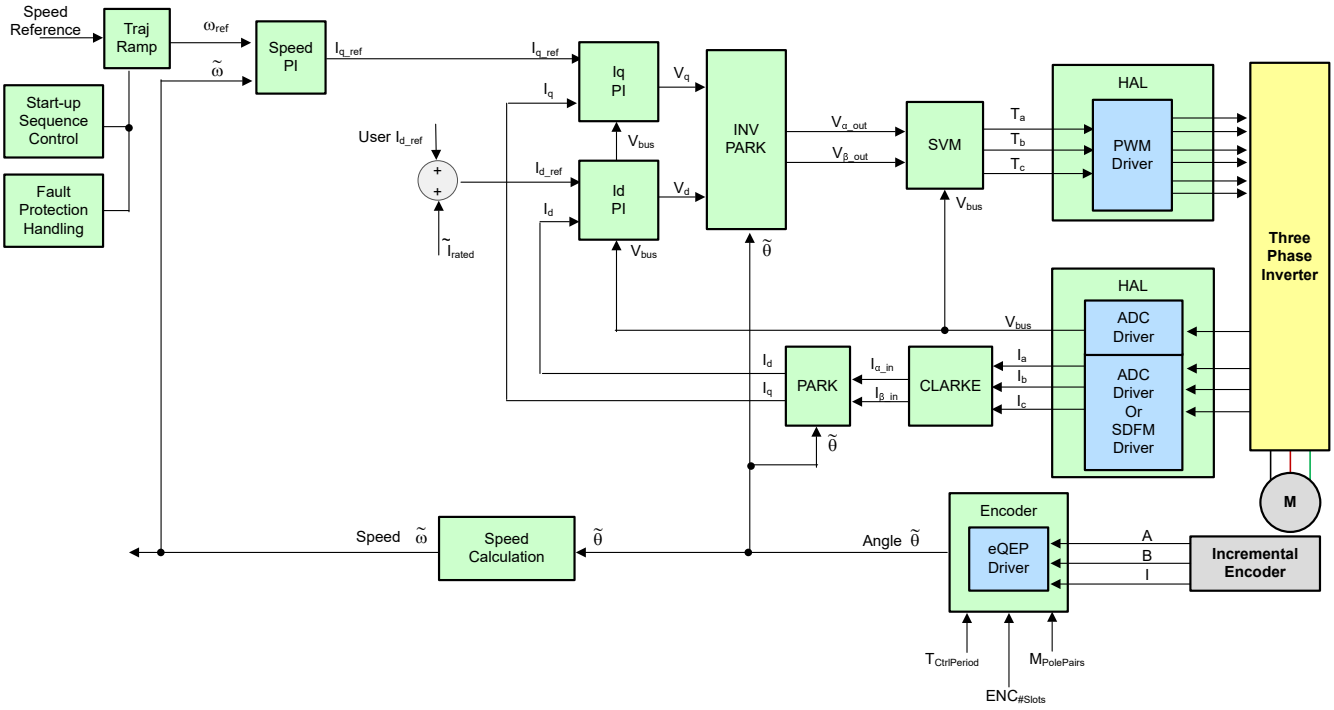

Figure 2-11. Step 4: Speed and Current Closed Loop Control Block Diagram

2.4.5.1 Build and Load Project

To verify the motor with current closed-loop control:

- Follow the same setup (Table 2-7). This time:
  - Connect the motor to the harness
  - Specify #define DMC_BUILDLEVEL DMC_LEVEL_4 in *sys_settings.h*
- Follow the same test procedure in Table 2-9

References

1. C29 Academy
2. Code Composer Studio Academy
3. Foundational Software Development Kits for F29 real-time MCUs
4. TI C29x Clang Compiler Tools user's guide
5. Texas Instruments: *The C29 CPU - Unrivaled Real-Time Performance with Optimized Architecture on C2000 MCUs*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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