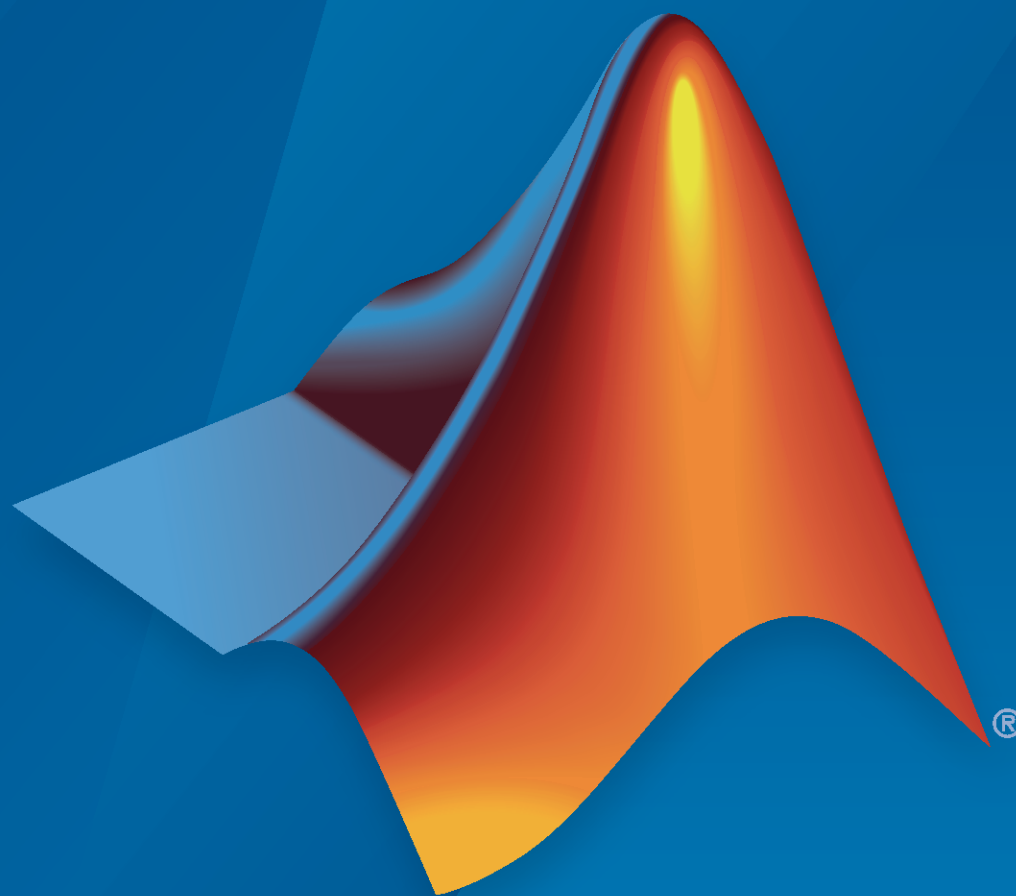# Motor Control Blockset™

Getting Started Guide

MATLAB&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Contents

Product Overview

1

Model Configuration Parameters

2

Estimate Control Gains from Motor Parameters

3

Implement Motor Speed Control by Using Field-Oriented Control (FOC)

4

Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool

5

6

7

Product Overview

Design and implement motor control algorithms

Motor Control Blockset provides reference examples and blocks for developing field-oriented control algorithms for brushless motors. The examples show how to configure a controller model to generate compact and fast C code for any target microcontroller (with Embedded Coder®). You can also use the reference examples to generate algorithmic C code and driver code for specific motor control kits.

The blockset includes Park and Clarke transforms, sliding mode and flux observers, a space-vector generator, and other components for creating speed and torque controllers. You can automatically tune controller gains based on specified bandwidth and phase margins for current and speed loops (with Simulink® Control Design™).

The blockset lets you create an accurate motor model by providing tools for collecting data directly from hardware and calculating motor parameters. You can use the parameterized motor model to test your control algorithm in closed-loop simulations.

Model Configuration Parameters

Model Configuration Parameters

Update the configuration parameters for a Simulink model that you create, before simulating or deploying the model to the controller.

In the Simulink window, click **Hardware Settings** in the **HARDWARE** tab to open the Configuration Parameters dialog box and select the target hardware in the **Hardware board** field.
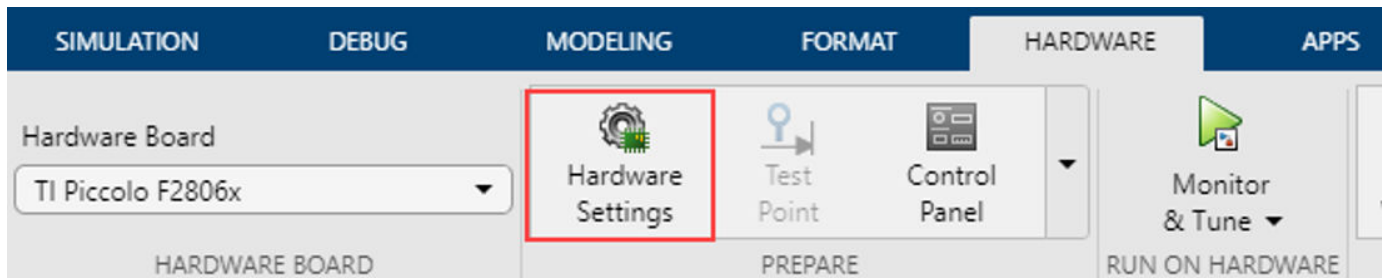
Solver Configuration

In the **Solver** tab of the Configuration Parameters dialog box, for a fixed-step discrete solver, type `auto` in the **Fixed-step size (fundamental sample time)** field.
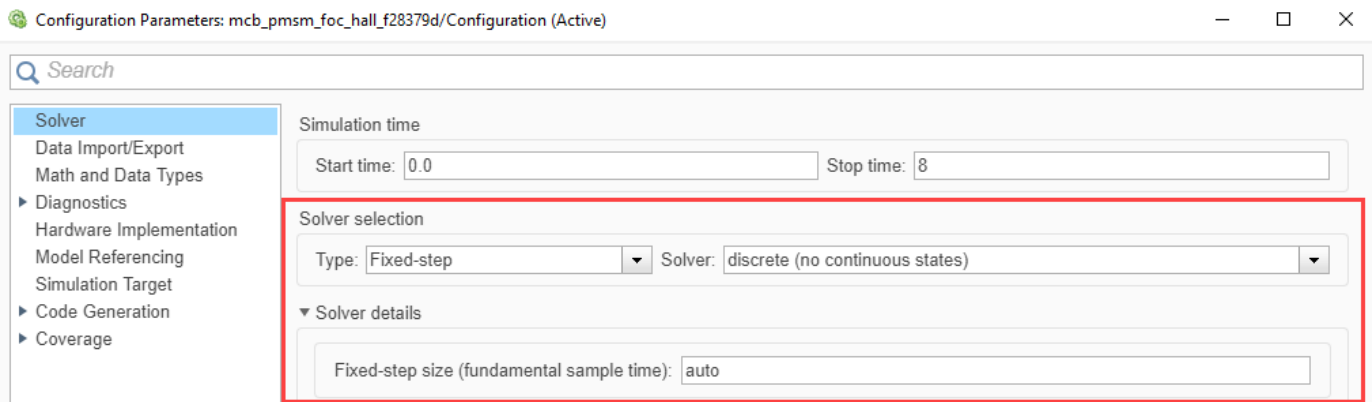
ADC Interface Configuration

If you connect analog inputs (current or voltage sensors) to the hardware board, configure the related ADC parameters in the Configuration Parameters dialog box by using the following steps:

1 Open the **Hardware Implementation** tab.

2 Set the ADC clock prescaler and check the ADC clock frequency. Ensure that the displayed ADC clock frequency is less than the maximum value specified in the device datasheet.

This example shows the ADC configuration for LAUNCHXL-F28379D board. The maximum operating frequency of ADCCLK for TMS320F28379D targets is 50 MHz.
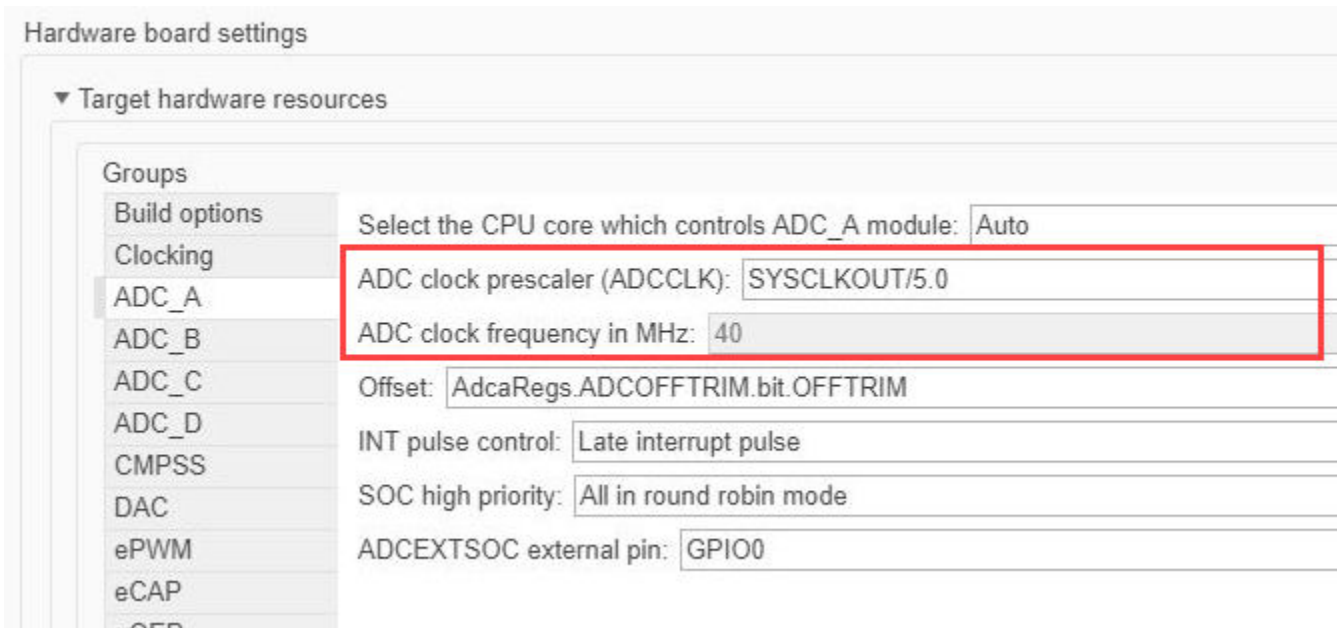
PWM Interface Configuration

If you connect PWM outputs from target device to the inverter, configure the related PWM parameters in the Configuration Parameters dialog box by using the following steps:

1   Open the **Hardware Implementation** tab.

2   Set the ePWM clock divider to **SYSCLKOUT/1**.

3   Update the following PWM pin assignment fields.

ePWM pin settings	Property
PWM1A pin assignment	Gate pulse for Phase-A high-side transistor
PWM1B pin assignment	Gate pulse for Phase-A low-side transistor
PWM2A pin assignment	Gate pulse for Phase-B high-side transistor
PWM2B pin assignment	Gate pulse for Phase-B low-side transistor
PWM3A pin assignment	Gate pulse for Phase-C high-side transistor
PWM3B pin assignment	Gate pulse for Phase-C low-side transistor

Hall Sensor Interface Configuration

If you connect a Hall sensor to the hardware board, configure the related parameters in the Configuration Parameters dialog box by using the following steps:

1   Open the **Hardware Implementation** tab.

2   Select the **eCAP** group under **Hardware board settings** > **Target hardware resources**.

3   Update the following ECAP pin assignment fields:

ECAP pin assignment field	Field value
ECAP1 pin assignment	Hall A
ECAP2 pin assignment	Hall B
ECAP3 pin assignment	Hall C

The following example shows the eCAP configuration for a Hall sensor connected to DRV8312 board with a F28069 Piccolo MCU control card:
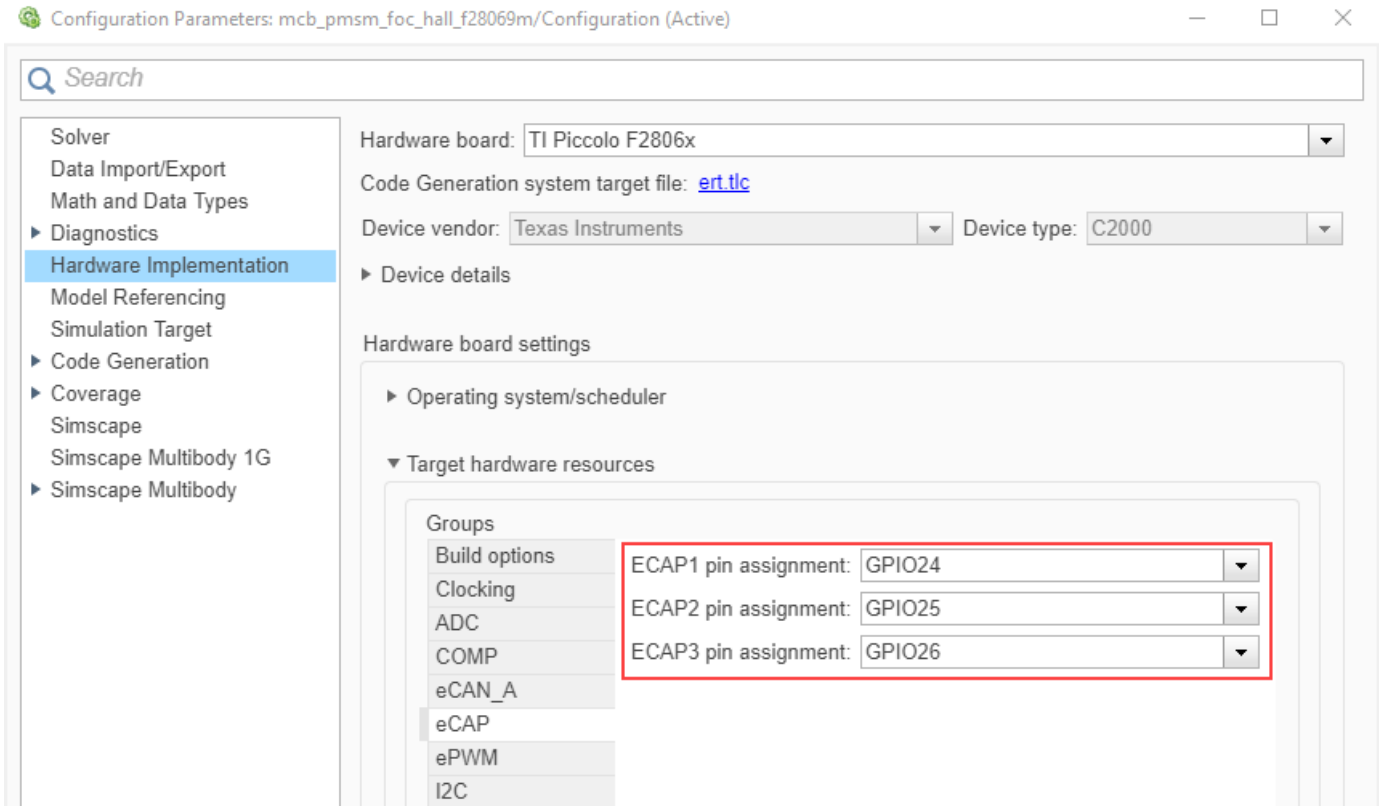
Quadrature Encoder Interface Configuration

If you connect a Quadrature Encoder sensor to the hardware board, configure the related parameters in the Configuration Parameters dialog box by using the following steps:

1   Open the **Hardware Implementation** tab.

2   Select the **eQEP** group under **Hardware board settings** > **Target hardware resources**.

3   Update the following EQEP pin assignment fields:

EQEP pin assignment field	Property
EQEP1A pin assignment	Quadrature Encoder Channel A
EQEP1B pin assignment	Quadrature Encoder Channel B
EQEP1I pin assignment	Quadrature Encoder Index

The following example shows the eQEP configuration for a quadrature encoder sensor connected to a LAUNCHXL-F28379D board:
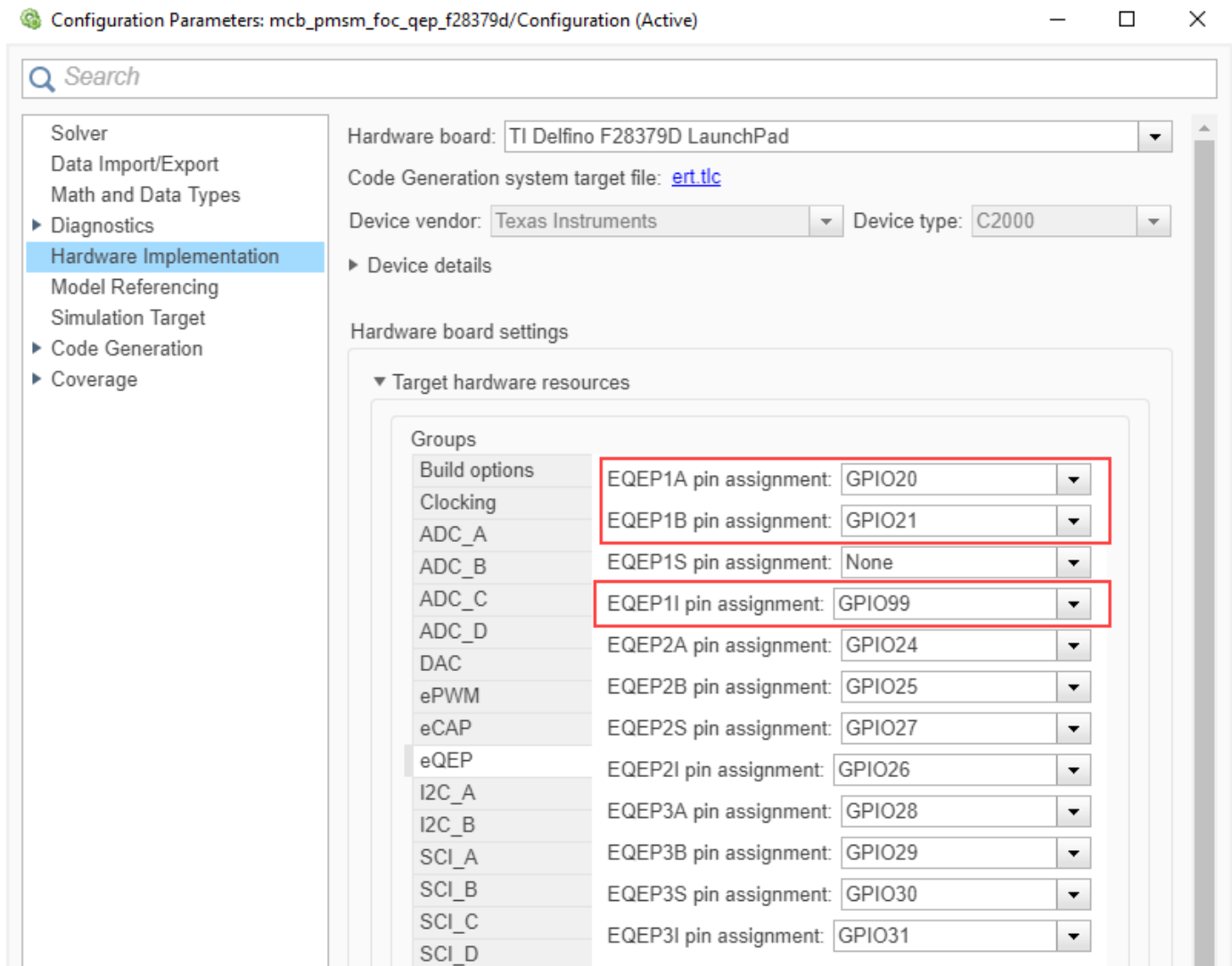
Serial Communication Interface Configuration

If you are generating code and using serial communication between host and target Simulink models, configure the related parameters in the Configuration Parameters dialog box by using the following steps:

1 Open the **Hardware Implementation** tab.

2 Select the **SCI_A** group under **Hardware board settings > Target hardware resources**.

3 Update the following SCI_A settings:

SCI_A settings	Property
Suspension mode	Serial suspension mode
Number of stop bits	Stop bits
Parity mode	Parity

SCI_A settings	Property
`Character length bits`	Data bits
`Desired baud rate in bits/sec`	Serial communication baud rate
`Pin assignment(Tx)`	Output pin for Serial Transmit
`Pin assignment(Rx)`	Input pin for Serial Receive

For example, use the following SCI_A configuration for a Hall sensor connected to a F28379D LaunchPad board:

3

Estimate Control Gains from Motor Parameters

Estimate Control Gains from Motor Parameters

Perform control parameter tuning for the speed and the torque control loops that are part of the Field-Oriented Control (FOC) algorithm. Motor Control Blockset provides you with multiple methods to compute the control loop gains from the system or block transfer functions that are available for the motors, inverter, and controller:

- Use the Field Oriented Control Autotuner block.
- Use Simulink Control Design.
- Use the model initialization script.
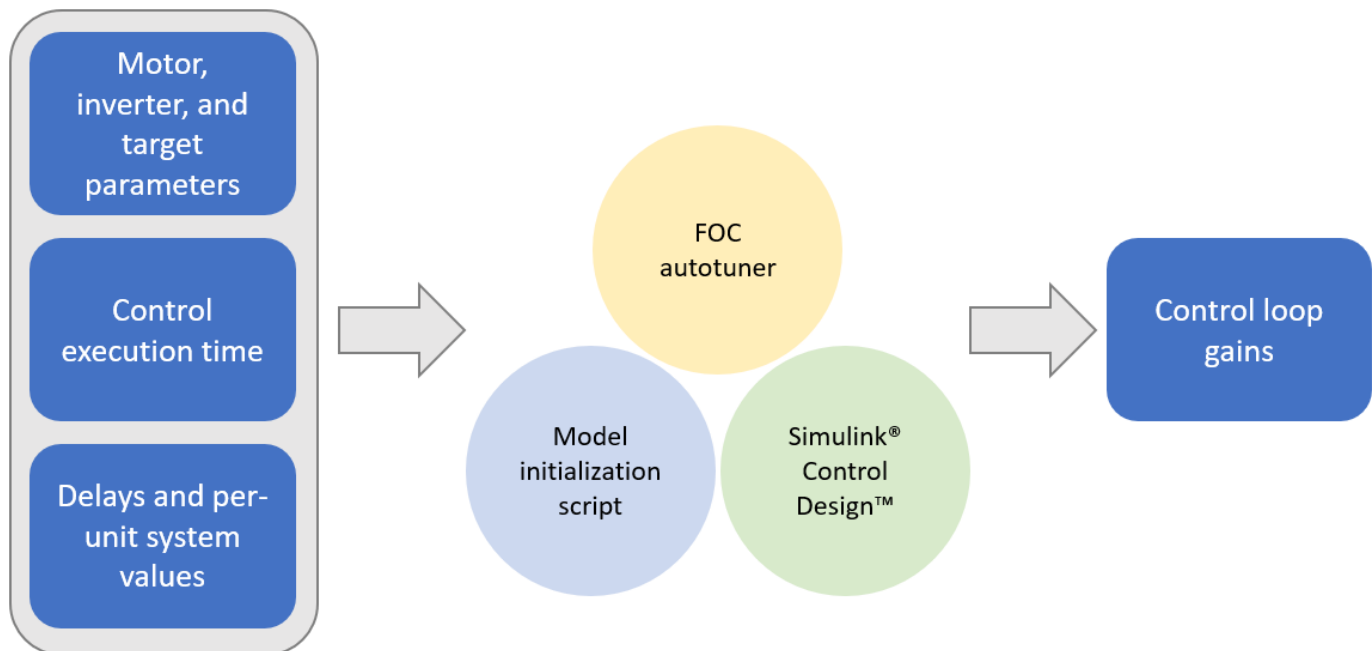
Field-Oriented Control Autotuner

The Field-Oriented Control Autotuner block of Motor Control Blockset enables you to automatically tune the PID control loops in your Field-Oriented Control (FOC) application in real time. You can automatically tune the PID controllers associated with the following loops (for more details, see "How to Use Field Oriented Control Autotuner Block"):

- Direct-axis (*d*-axis) current loop
- Quadrature-axis (*q*-axis) current loop
- Speed loop

For each loop that the block tunes, the Field-Oriented Control Autotuner block performs the autotuning experiment in a closed-loop manner without using a parametric model associated with that loop. The block enables you to specify the order in which the block tunes the control loops. When the tuning experiment runs for one loop, the block has no effect on the other loops. For more details about FOC autotuner, see Field Oriented Control Autotuner and "Tune PI Controllers by Using Field Oriented Control Autotuner" on page 4-25.

Simulink Control Design

Simulink Control Design enables you to design and analyze the control systems modeled in Simulink. You can automatically tune the arbitrary SISO and MIMO control architectures, including the PID controllers. You can deploy PID autotuning to the embedded software to automatically compute the PID gains in real time.

You can find the operating points and compute the exact linearizations of the Simulink models at different operating conditions. Simulink Control Design provides tools that let you compute the simulation-based frequency responses without modifying your model. For details, see https://www.mathworks.com/help/slcontrol/index.html

Model Initialization Script

This section explains how the Motor Control Blockset examples estimate the control gains needed to implement field-oriented control. For example, for a PMSM that is connected to a quadrature encoder, these steps describe the procedure to compute the control loop gain values from the system details by using the initialization script:

1   Open the initialization script (`.m`) file of the example in MATLAB®. To find the associated script file name:

   a   Select **Modeling > Model Settings > Model Properties** to open the model properties dialog box.
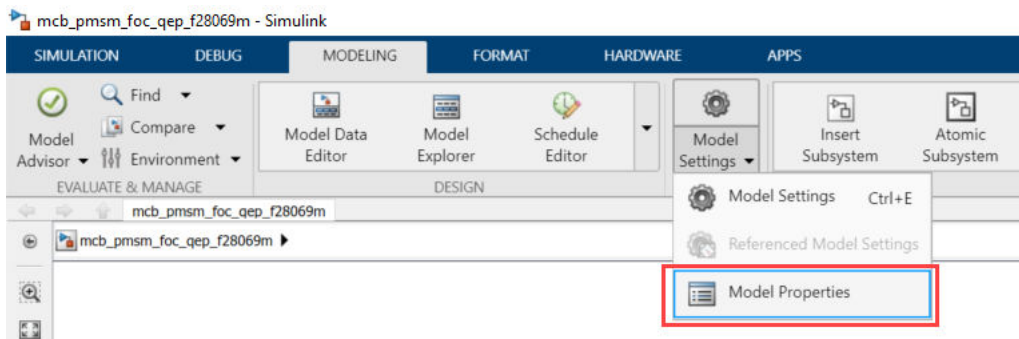
   b   In the Model Properties dialog box, navigate to the **Callbacks** tab > **InitFcn** to find the name of the script file that Simulink opens before running the example.
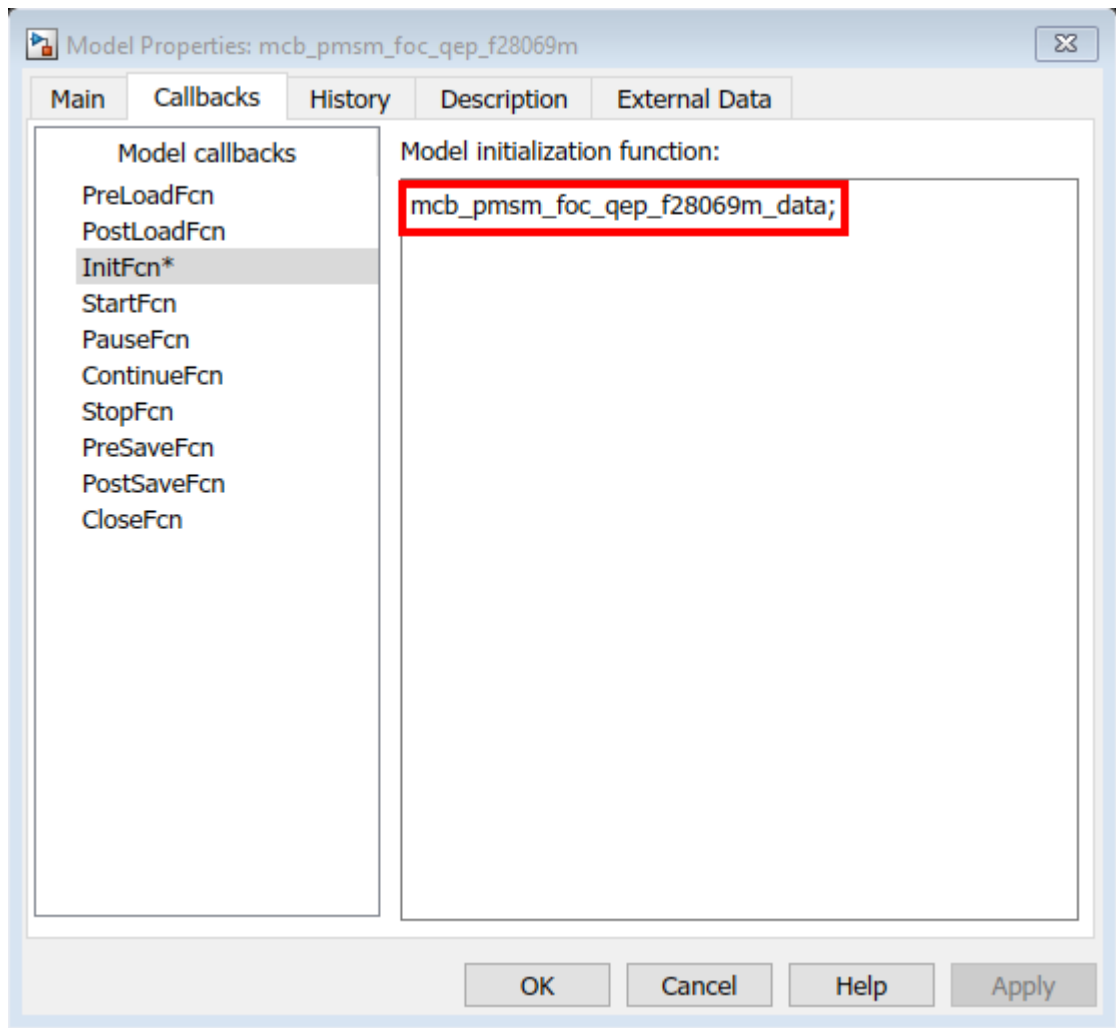
2   This figure shows an example of the initialization script (.m) file.

```matlab
%% *******************************************************************
% Model          :    PMSM Field Oriented Control
% Description    :    Set Parameters for PMSM Field Oriented Control
% File name      :    mcb_pmsm_foc_qep_f28069m_data.m
% Copyright 2020 The MathWorks, Inc.

%% Parameters needed for Offset computation are
% target.PWM_Counter_Period    - PWM counter value for epwm blocks
% target.CPU_frequency         - CPU frequency of the microcontroller
% Ts                           - Control sample time
% PU_System.N_base             - Base speed for per unit conversion
% pmsm.p                       - Number pole pairs in the motor

% Other parameters are not mandatory for offset computation

%% Set PWM Switching frequency
PWM_frequency    = 20e3;      %Hz         // converter s/w freq
T_pwm            = 1/PWM_frequency;  %s   // PWM switching time period

%% Set Sample Times
Ts              = T_pwm;          %sec         // simulation time step for controller
Ts_simulink     = T_pwm/2;        %sec         // simulation time step for model simulation
Ts_motor        = T_pwm/2;        %Sec         // Simulation sample time
Ts_inverter     = T_pwm/2;        %sec         // simulation time step for average value inverter
Ts_speed        = 10*Ts;          %Sec         // Sample time for speed controller

%% Set data type for controller & code-gen
% dataType = fixdt(1,32,17);     % Fixed point code-generation
dataType = 'single';             % Floating point code-generation

%% System Parameters // Hardware parameters

pmsm = mcb_SetPMSMMotorParameters('BLY171D');
pmsm.PositionOffset = 0.17;

%% Parameters below are not mandatory for offset computation

inverter = mcb_SetInverterParameters('DRV8312-C2-KIT');

inverter.ADCOffsetCalibEnable = 1; % Enable: 1, Disable:0

target = mcb_SetProcessorDetails('F28069M',PWM_frequency);

%% Derive Characteristics
pmsm.N_base = mcb_getBaseSpeed(pmsm,inverter); %rpm // Base speed of motor at given Vdc
% mcb_getCharacteristics(pmsm,inverter);

%% PU System details // Set base values for pu conversion

PU_System = mcb_SetPUSystem(pmsm,inverter);

%% Controller design // Get ballpark values!

PI_params = mcb.internal.SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed);

%Updating delays for simulation
PI_params.delay_Currents    = int32(Ts/Ts_simulink);
PI_params.delay_Speed       = int32(Ts_speed/Ts_simulink);

% mcb_getControlAnalysis(pmsm,inverter,PU_System,PI_params,Ts,Ts_speed);
```

3   Use the **Workspace** to edit the control variables values. For example, to update Stator resistance (Rs), use the variable `pmsm` to add the parameter value to the `Rs` field.

4   The model initialization script associated with a target model calls these functions and sets up the workspace with the necessary variables.

Model Initialization Script	Function Called By Model Initialization Script	Description
Script associated with a target model	`mcb_SetPMSMMotorParameters`	Input to the function is motor type (for example, BLY171D).
		The function populates a structure named `pmsm` in the MATLAB workspace, which is used by the model.
		It also computes the permanent magnet flux and rated torque for the selected motor.
		You can extend the function by adding an additional switch-case for a new motor.
		This function also loads the structure `motorParam`, obtained by running parameter estimation, to the structure `pmsm`. If the structure `motorParam` is not available in the MATLAB workspace, the function loads the default parameters.
	`mcb_SetInverterParameters`	Input to the function is inverter type (for example, BoostXL-DRV8305).
		The function populates a structure named `inverter` in the MATLAB workspace, which is used by the model.
		The function also computes the inverter resistance for the selected inverter.
		You can extend the function by adding an additional switch-case for a new inverter.

Model Initialization Script	Function Called By Model Initialization Script	Description
	`mcb_SetProcessorDetails`	Inputs to the function are processor type (for example, F28379D) and the Pulse-Width Modulation (PWM) switching frequency. The function populates a structure named `target` in the MATLAB workspace, which is used by the model. The function also computes the PWM counter period that is a parameter for the ePWM block in the target model. You can extend the function by adding an additional switch-case for a new processor.
	`mcb_getBaseSpeed`	Inputs to the function are motor and inverter parameters. The function computes the base speed for PMSM. Type `help mcb_getBaseSpeed` at the MATLAB command window or see section "Obtain Base Speed" on page 3-12 for more details.
	`mcb_SetPUSystem`	Inputs to the function are motor and inverter parameters. The function sets the base values of the per-unit system for voltage, current, speed, torque, and power. The function populates a structure named `PU_System` in the MATLAB workspace, which is used by the model.

Model Initialization Script	Function Called By Model Initialization Script	Description
	`mcb.internal.SetControllerParameters`	Inputs to the function are motor and inverter parameters, per-unit system base values, PWM switching time period, sample time for the control system, and sample time for the speed controller. The function computes the Proportional Integral (*PI*) parameters (*Kp*, *Ki*) for the field-oriented control implementation. The function populates a structure named `PI_params` in the MATLAB workspace, which is used by the model. See section "Obtain Controller Gains" on page 3-14 for more details.

This table explains the useful variables for each control parameter that you can update.

Note You can try starting MATLAB in the administrator mode on Windows® system, if you are unable to update the model initialization scripts associated with the example models.

Control Parameter Category	Control Parameter Name	MATLAB Workspace Variable
Motor parameters	Manufacturer's model number	`pmsm.model`
	Manufacturer's serial number	`pmsm.sn`
	Pole pairs	`pmsm.p`
	Stator resistance (Ohm)	`pmsm.Rs`
	d-axis stator winding inductance (Henry)	`pmsm.Ld`
	q-axis stator winding inductance (Henry)	`pmsm.Lq`
	Back emf constant (V_line(peak)/krpm)	`pmsm.Ke`
	Motor Inertia (kg.m^2)	`pmsm.J`
	Friction constant (N.m.s)	`pmsm.F`
	Permanent Magnet Flux (WB)	`pmsm.FluxPM`
	Trated	`pmsm.T_rated`

Control Parameter Category	Control Parameter Name	MATLAB Workspace Variable
	Nbase	`pmsm.N_base`
	Irated	`pmsm.I_rated`
Position decoders	QEP index and Hall position offset correction	`pmsm.PositionOffset`
	Quadrature encoder slits per revolution	`pmsm.QEPSlits`
Inverter parameters	Manufacturer's model number	`inverter.model`
	Manufacturer's serial number	`inverter.sn`
	DC link voltage of the inverter (V)	`inverter.V_dc`
	Maximum measurable currents by ADCs (A)	`inverter.I_max`
	Maximum permissible currents by inverter (A)	`inverter.I_trip`
	On-state resistance of MOSFETs (Ohm)	`inverter.Rds_on`
	Shunt resistance for current sensing (Ohm)	`inverter.Rshunt`
	Per-phase board resistance seen by motor (Ohm)	`inverter.R_board`
	Current scaling	`inverter.MaxADCCnt`

	ADC Offsets for current sensor (Ia and Ib)	`inverter.CtSensAOffset`
		
		`inverter.CtSensBOffset`

	Enable Auto-calibration for current sense ADCs	`inverter.ADCOffsetCalibEnable`
Processor	Manufacturer's model number	`target.model`
	Manufacturer's serial number	`target.sn`
	CPU Frequency	`target.CPU_frequency`
	PWM frequency	`target.PWM_frequency`
	PWM counter period	`target.PWM_Counter_Period`
Per-Unit System	Base voltage (V)	`PU_System.V_base`
	Base current (A)	`PU_System.I_base`
	Base speed (rpm)	`PU_System.N_base`
	Base torque (Nm)	`PU_System.T_base`
	Base power (Watts)	`PU_System.P_base`
Data-type for target device	Data-type (Fixed-point Or Floating-point) selection	`dataType`

Control Parameter Category	Control Parameter Name	MATLAB Workspace Variable
Sample time values	Switching frequency for converter	`PWM_frequency`
	PWM switching time period	`T_pwm`
	Sample time for current controllers	`Ts`
	Sample time for speed controller	`Ts_speed`
	Simulation sample time	`Ts_simulink`
	Simulation sample time for motor	`Ts_motor`
	Simulation sample time for inverter	`Ts_inverter`
Controller parameters	Proportional gain for Iq controller	`PI_params.Kp_i`
	Integral gain for Iq controller	`PI_params.Ki_i`
	Proportional gain for Id controller	`PI_params.Kp_id`
	Integral gain for Id controller	`PI_params.Ki_id`
	Proportional gain for Speed controller	`PI_params.Kp_speed`
	Integral gain for Speed controller	`PI_params.Ki_speed`
	Proportional gain for Field weakening controller	`PI_params.Kp_fwc`
	Integral gain for Field weakening controller	`PI_params.Ki_fwc`

Note For the predefined processors and drivers, the model initialization script uses the default values.

The model initialization script uses these functions for performing the computations:

Control Parameter Category	Function	Functionality
Base speed of the motor	`mcb_getBaseSpeed`	Calculates the base speed of PMSM at the rated voltage and rated load. For details, type `help mcb_getBaseSpeed` at the MATLAB command prompt or see section "Obtain Base Speed" on page 3-12.

Control Parameter Category	Function	Functionality
Motor characteristics for the given motor and inverter	`mcb_getCharacteristics`	Obtain these characteristics of the motor. • Torque as opposed to speed characteristics • Power as opposed to speed characteristics • *Iq* as opposed to speed and *Id* as opposed to speed characteristics For details, type `help mcb_getCharacteristics` at the MATLAB command prompt.
Control algorithm parameters	`mcb.internal.SetControllerParameters`	Compute the gains for these PI controllers: • Current (torque) control loop gains (*Kp*, *Ki*) for currents Id and Iq • Speed control loop gains (*Kp*, *Ki*) • Field weakening control gains (*Kp*, *Ki*) For details, see section "Obtain Controller Gains" on page 3-14.
Control analysis for the motor and inverter you are using	`mcb_getControlAnalysis`	Performs frequency domain analysis for the computed gains of PI controllers used in the field-oriented motor control system. **Note** This feature requires Control System Toolbox™. For details, type `help mcb_getControlAnalysis` at the MATLAB command prompt.

Obtain Base Speed

The function `mcb_getBaseSpeed` computes the base speed of the PMSM at the given supply voltage. Base speed is the maximum motor speed at the rated voltage and rated load, outside the field-weakening region.

When you call this function (for example, base_speed = mcb_getBaseSpeed(pmsm,inverter)), it returns the base speed (in rpm) for the given combination of PMSM and inverter. The function accepts the following inputs:

- PMSM parameter structure.
- Inverter parameter structure.

These equations describe the computations that the function performs:

The inverter voltage constraint is defined by computing the d-axis and q-axis voltages:

$$v_{do} = - \omega_e L_q i_q$$

$$v_{qo} = \omega_e \left(L_d i_d + \lambda_{pm} \right)$$

$$v_{max} = \frac{v_{dc}}{\sqrt{3}} - R_s i_{max} \geq \sqrt{v_{do}^2 + v_{qo}^2}$$

The current limit circle defines the current constraint which can be considered as:

$$i_{max}^2 = i_d^2 + i_q^2$$

In the preceding equation, i_d is zero for surface PMSMs. For interior PMSMs, values of i_d and i_q corresponding to MTPA are considered.

Using the preceding relationships, we can compute the base speed as:

$$\omega_{base} = \frac{1}{p} \cdot \frac{v_{max}}{\sqrt{\left(L_q i_q \right)^2 + \left(L_d i_d + \lambda_{pm} \right)^2}}$$

where:

- ω_e is the electrical speed corresponding to frequency of stator voltages (Radians/ sec).
- ω_{base} is the mechanical base speed of the motor (Radians/ sec).
- i_d is the d-axis current (Amperes).
- i_q is the q-axis current (Amperes).
- v_{do} is the d-axis voltage when i_d is zero (Volts).
- v_{qo} is the q-axis voltage when i_q is zero (Volts).
- L_d is the d-axis winding inductance (Henry).
- L_q is the q-axis winding inductance (Henry).
- R_s is the stator phase winding resistance (Ohms).
- λ_{pm} is the permanent magnet flux linkage (Weber).
- v_d is the d-axis voltage (Volts).
- v_q is the q-axis voltage (Volts).
- v_{max} is the maximum fundamental line to neutral voltage (peak) supplied to the motor (Volts).
- v_{dc} is the dc voltage supplied to the inverter (Volts).

- i_{max} is the maximum phase current (peak) of the motor (Amperes).

- p is the number of motor pole pairs.

Obtain Motor Characteristics

The function `mcb_getCharacteristics` calculates the torque and speed characteristics of the motor, which helps you to develop the control algorithm for the motor.

The function returns these characteristics for the given PMSM:

- Torque as opposed to Speed
- Power as opposed to Speed
- I_q as opposed to Speed
- I_d as opposed to Speed
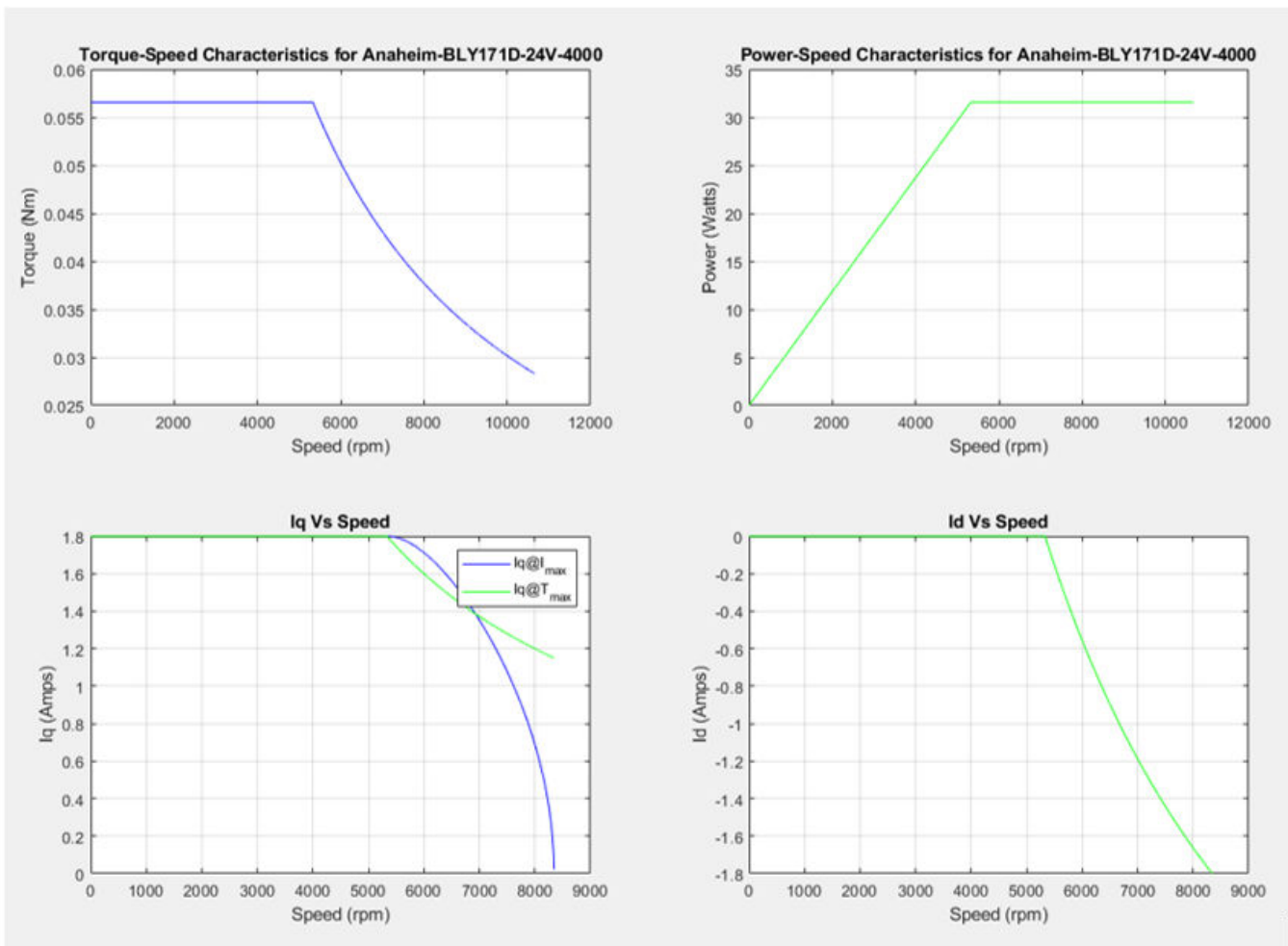
Obtain Controller Gains

The function `mcb.internal.SetControllerParameters` computes the gains for the PI controllers used in the field-oriented motor control systems.

When you call this function (for example, `PI_params = mcb.internal.SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts_control,Ts_speed)`), it returns the gains of these PI controllers used in the FOC algorithm:

- Direct-axis (*d*-axis) current loop
- Quadrature-axis (*q*-axis) current loop
- Speed loop
- Field-weakening control loop

The function accepts these inputs:

- `pmsm object`
- `inverter object`
- `PU system params`
- `T_pwm`
- `Ts_control`
- `Ts_speed`

The function does not plot any characteristic.

The design of compensators depends on the classical frequency response analysis applied to the motor control systems. We used the Modulus Optimum (MO) based design for the current controllers and the Symmetrical Optimum (SO) based design for the speed controller.

The function automatically computes the other required parameters (for example, bandwidth, damping) based on the input arguments.

Perform Control Analysis

The function `mcb_getControlAnalysis` performs the basic control analysis of the PMSM FOC current control system. The function performs frequency domain analysis for the computed PI controller gains used in the field-oriented motor control systems.

Note This function requires the Control System Toolbox.

When you call this function (for example, `mcb_getControlAnalysis(pmsm,inverter,PU_System,PI_params,Ts,Ts_speed)`), it performs the following functions for the current control loop or subsystem:

- Transfer function for the closed-loop current control system
- Root locus
- Bode diagram
- Stability margins (PM & GM)
- Step response
- PZ map

The function plots the corresponding plots:

Implement Motor Speed Control by Using Field-Oriented Control (FOC)

Field-Oriented Control (FOC)

Field-Oriented Control (FOC), also known as vector control, is a technique used to control Permanent Magnet Synchronous Motor (PMSM) and induction motors. FOC provides good control capability over the full torque and speed ranges. The FOC implementation requires transformation of stator currents from the stationary reference frame to the rotor flux reference frame (also known as *d-q* reference frame).

Speed control and torque control are the most commonly used control modes of FOC. The position control mode is less common. Most of the traction applications use the torque control mode in which the motor control system follows a reference torque value. In the speed control mode, the motor controller follows a reference speed value and generates a torque reference for the torque control that forms an inner subsystem. In the position control mode, the speed controller forms the inner subsystem.

FOC algorithm implementation requires real time feedback of the currents and rotor position. Measure the current and position by using sensors. You can also use sensorless techniques that use the estimated feedback values instead of the actual sensor-based measurements.

This figure shows the Field-Oriented Control architecture for PMSM.
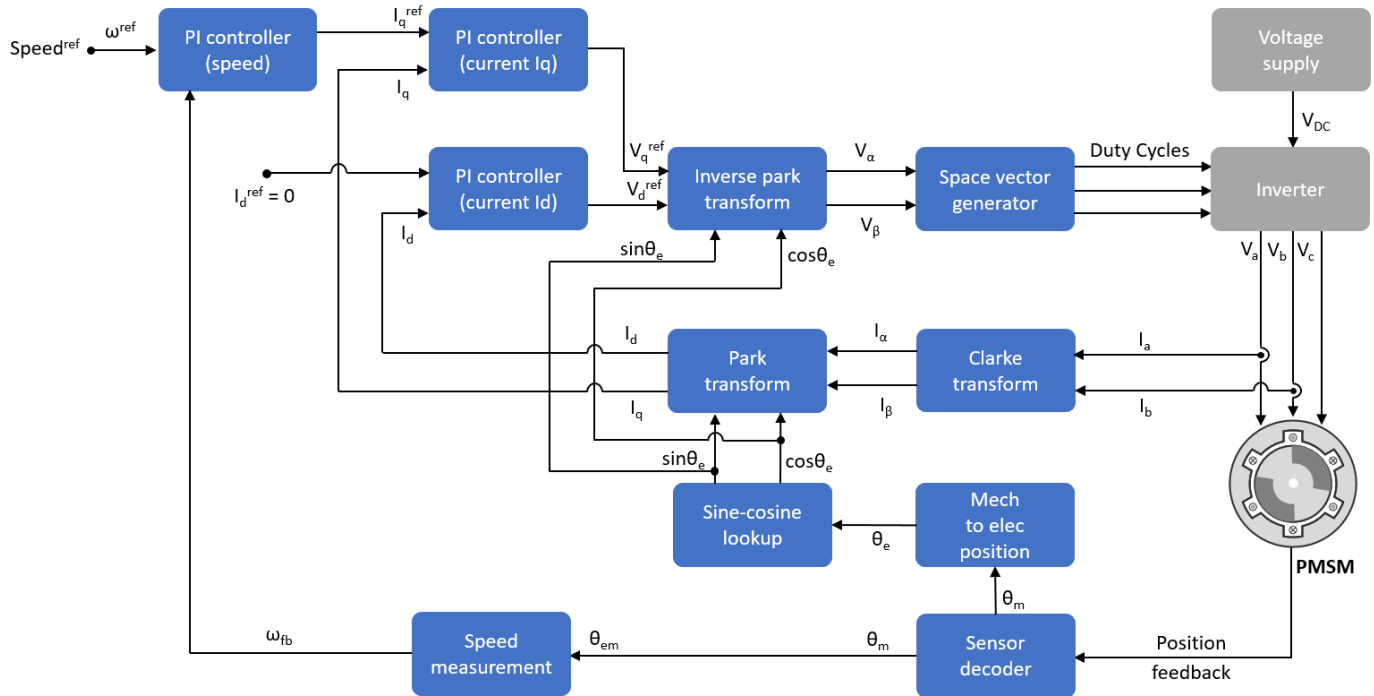
This figure shows the Field-Oriented Control architecture for an induction motor.
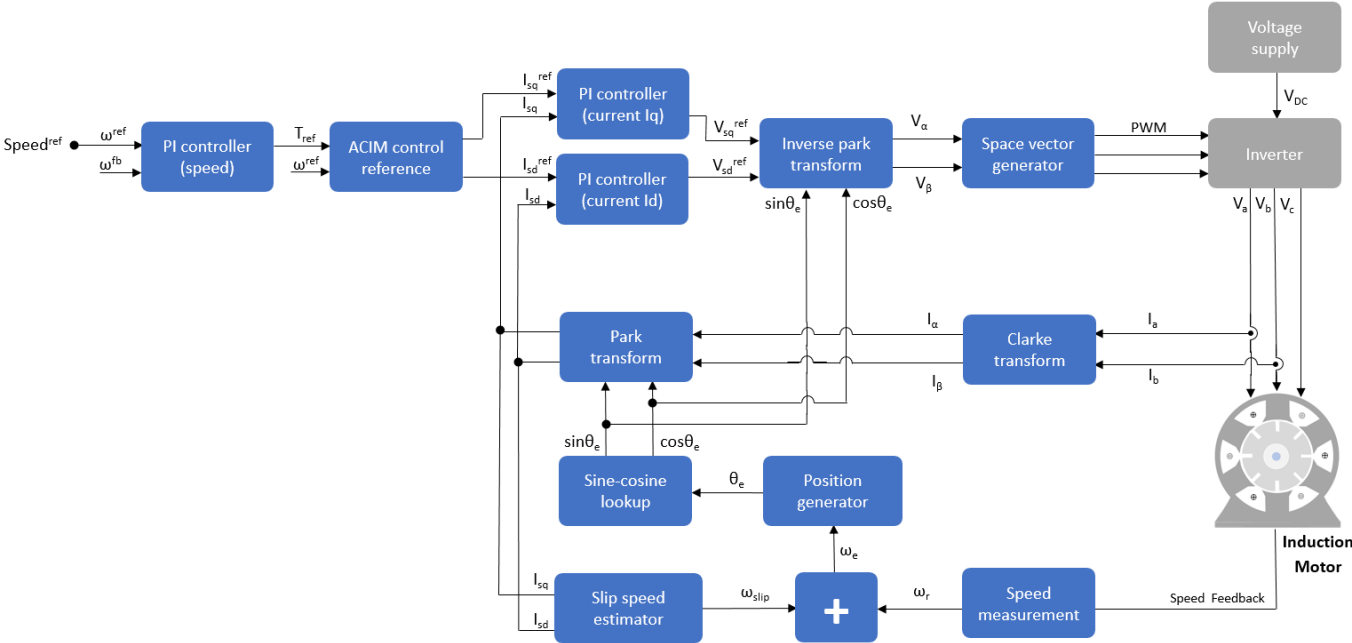
Six-Step Commutation

Six-step commutation, also known as trapezoidal commutation, is a commutation technique used to control three-phase brushless DC (BLDC) permanent magnet motor. It controls the stator currents to achieve a motor speed and direction of rotation.

Six-step commutation uses these conduction modes:

- 120 degree mode conducts current in only two stator phases.
- 180 degree mode conducts current in all three stator phases.

Motor Control Blockset supports 120 degree conduction mode. At a given time, this mode energizes only two stator phases and electrically isolates the third phase from the power supply. You can use either Hall or quadrature encoder position sensors to detect the rotor position. Motor Control Blockset provides Six Step Commutation block that uses the Hall sequence or rotor position inputs to determine the 60 degree sector where the rotor is present. It generates a switching sequence that energizes the corresponding phases. As the motor rotates, the sequence switches the stator currents every 60 degree such that the torque angle (angle between rotor d-axis and stator magnetic field) remains 90 degrees (with a deviation of 30 degrees). Therefore, the switching signals operate switches to control the stator currents, and therefore, control the motor speed and direction of rotation. For more details, see Six Step Commutation.

The stator current waveform takes a trapezoidal shape.
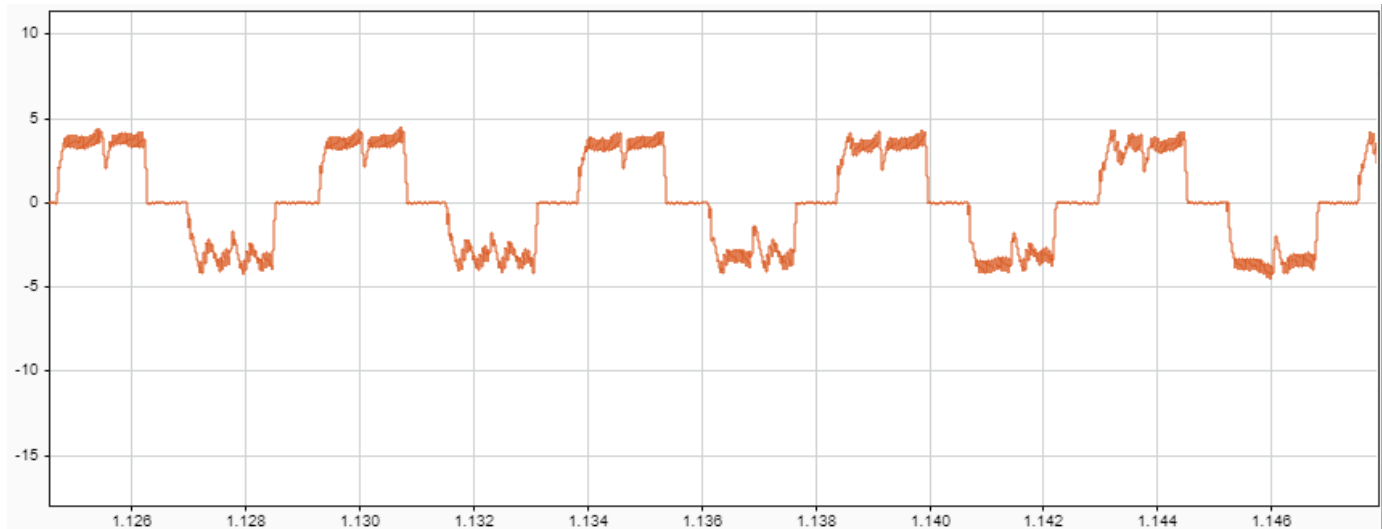
The 120 degree conduction mode is a less complex technique that provides good speed control for the BLDC motors. This figure shows the six-step commutation architecture for a BLDC motor.
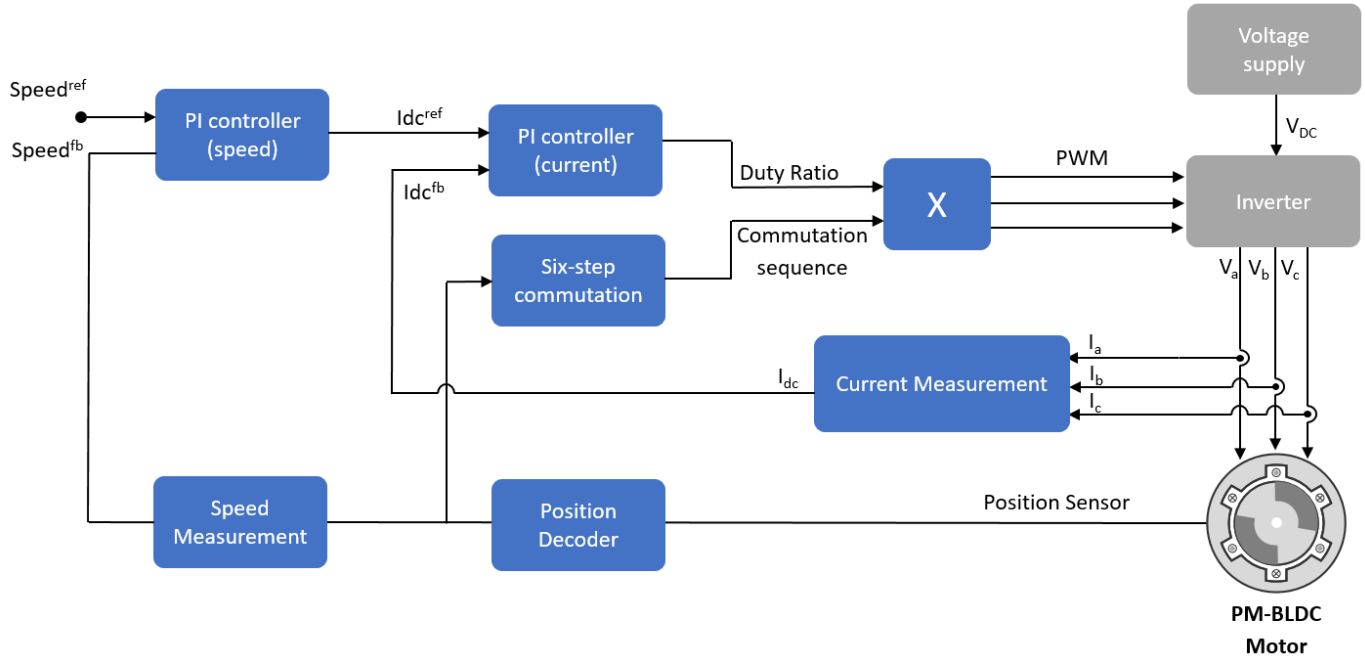
Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset

This example uses open-loop control (also known as scalar control or Volts/Hz control) to run a motor. This technique varies the stator voltage and frequency to control the rotor speed without using any feedback from the motor. You can use this technique to check the integrity of the hardware connections. A constant speed application of open-loop control uses a fixed-frequency motor power supply. An adjustable speed application of open-loop control needs a variable-frequency power supply to control the rotor speed. To ensure a constant stator magnetic flux, keep the supply voltage amplitude proportional to its frequency.

Open-loop motor control does not have the ability to consider the external conditions that can affect the motor speed. Therefore, the control system cannot automatically correct the deviation between the desired and the actual motor speed.

This model runs the motor by using an open-loop motor control algorithm. The model helps you get started with Motor Control Blockset™ and verify the hardware setup by running the motor. The target model algorithm also reads the ADC values from the current sensors and sends the values to the host model by using serial communication.

You can use this model to:

- Check connectivity with the target.
- Check serial communication with the target.
- Verify the hardware and software environment.
- Check ADC offsets for current sensors.
- Run a new motor with an inverter and target setup for the first time.

Models

The example includes these models:

- **mcb_open_loop_control_f28069M_DRV8312**

- **mcb_open_loop_control_f28069MLaunchPad**

- **mcb_open_loop_control_f28379d**

You can use these models for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller:

```
open_system('mcb_open_loop_control_f28069M_DRV8312.slx');
```

Open Loop Control of 3-phase motors

Note: This example requires a TI F28069M Control Card with DRV8312 EVM

Steps:
1. Update the Configuration panel before simulation or codegeneration.
2. Simulate the model to output the phase voltages in scope
3. Click **Build, Deploy & Start** in the **Hardware** tab
4. Control the motor by using host model

Configuration

Number of Pole Pairs	4
PWM Frequency [Hz]	20000
Base Speed [RPM]	4000
Data type for control algorithm	single

Target Model

HW_INT
Code generation

HW_INT
Simulation

Heartbeat LED

Global Variable

Enable

SCI_Rx_INT()
Desired Speed
Serial Receive

1   Desired Speed
Communication

ADC Interrupt()
Speed_ref_PU   Vabc_PU
Open Loop Control

Copyright 2020 The MathWorks, Inc.

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

1. For the models: **mcb_open_loop_control_f28069M_DRV8312** and **mcb_open_loop_control_f28069MLaunchPad**

- Motor Control Blockset™
- Fixed-Point Designer™

2. For the model: **mcb_open_loop_control_f28379d**

- Motor Control Blockset™

To generate code and deploy model:

1. For the models: **mcb_open_loop_control_f28069M_DRV8312** and **mcb_open_loop_control_f28069MLaunchPad**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

2. For the model: **mcb_open_loop_control_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. For BOOSTXL-DRV8323, use these steps to update the model:

- Navigate to this path in the model: /Open Loop Control/Codegen/Hardware Initialization.
- For LAUNCHXL-F28379D: Update **DRV830x Enable block** from GPIO124 to GPIO67.
- For LAUNCHXL-F28069M: Update **DRV830x Enable block** from GPIO50 to GPIO12.

2. For BOOSTXL-3PHGANINV, use these steps to update the model:

- For LAUNCHXL-F28379D: In the **Configuration** panel of **mcb_open_loop_control_f28379d**, set **Inverter Enable Logic** to **Active Low**.

NOTE: When using BOOSTXL-3PHGANINV inverter, ensure that proper insulation is available between bottom layer of BOOSTXL-3PHGANINV and the LAUNCHXL board.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the motor by using open-loop control.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

• F28069M controller card + DRV8312-69M-KIT inverter:
mcb_open_loop_control_f28069M_DRV8312

For connections related to the preceding hardware configuration, see "F28069 control card configuration" on page 7-2.

• LAUNCHXL-F28069M controller + (BOOSTXL-DRV8301 or BOOSTXL-DRV8305 or BOOSTXL-DRV8323 or BOOSTXL-3PHGANINV) inverter: **mcb_open_loop_control_f28069MLaunchPad**

• LAUNCHXL-F28379D controller + (BOOSTXL-DRV8301 or BOOSTXL-DRV8305 or BOOSTXL-DRV8323 or BOOSTXL-3PHGANINV) inverter: **mcb_open_loop_control_f28379d**

To configure the model **mcb_open_loop_control_f28379d**, set the **Inverter Enable Logic** field (in the **Configuration** panel of target model) to:

• **Active High**: To use the model with BOOSTXL-DRV8301 or BOOSTXL-DRV8305 or BOOSTXL-DRV8323 inverter.

• **Active Low**: To use the model with BOOSTXL-3PHGANINV inverter.

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

NOTE:

• This example supports any type of three-phase AC motor (PMSM or induction) and any type of inverter attached to the supported hardware.

• Some PMSMs do not run at higher speeds, especially when the shaft is loaded. To resolve this issue, you should apply more voltages corresponding to a given frequency. You can use these steps to increase the applied voltages in the model:

1. Navigate to this path in the model: /Open Loop Control/Control_System/VabcCalc/.

2. Update the gain Correction_Factor_sinePWM as 20%.

3. For safety reasons, regularly monitor the motor shaft, motor current, and motor temperature.

Generate Code and Run Model to Implement Open-Loop Control

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see "Model Configuration Parameters" on page 2-2.

4. Update these motor parameters in the **Configuration** panel of the target model.

- **Number of Pole Pairs**
- **PWM Frequency [Hz]**
- **Base Speed [RPM]**
- **Data type for control algorithm**
- **Inverter Enable Logic** (only available in **mcb_open_loop_control_f28379d** target model)

5. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, a program that operates the CPU2 blue LED by using GPIO31 (c28379d_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

NOTE: Ignore the warning message "Multitask data store option in the Diagnostics page of the Configuration Parameter Dialog is none" displayed by the model advisor, by clicking the Always Ignore button. This is part of the intended workflow.

7. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_open_loop_control_host_model.slx');
```
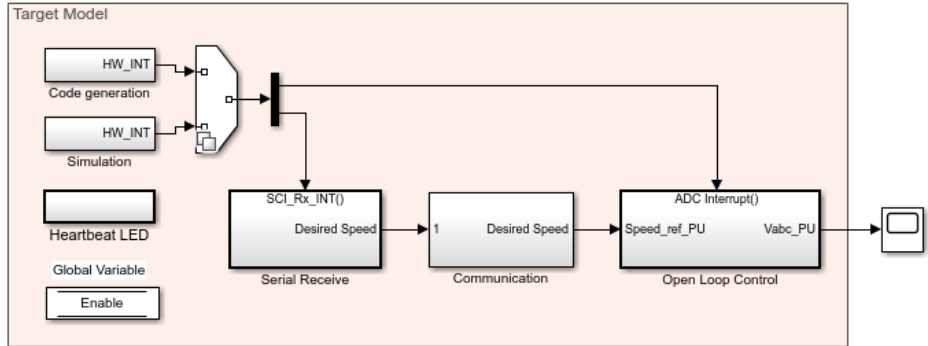
Open Loop Control Host Model

Note:
1. Open target model and compile (Ctrl + D) to load variable to workspace.
   F28069m + DRV8312
   F28069m Launchpad
   F28379d Launchpad
2. Select hardware in configuration panel
3. Select the serial port in 'Serial 1' tab of 'Host Serial Setup'
4. Use 'Motor Start / Stop' switch to enable and disable motor.
5. Input speed request using 'Reference Speed' text box or sliding bar.
6. Observe the ADC counts for phase current measurent in scope
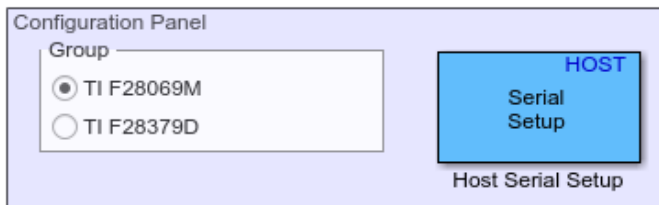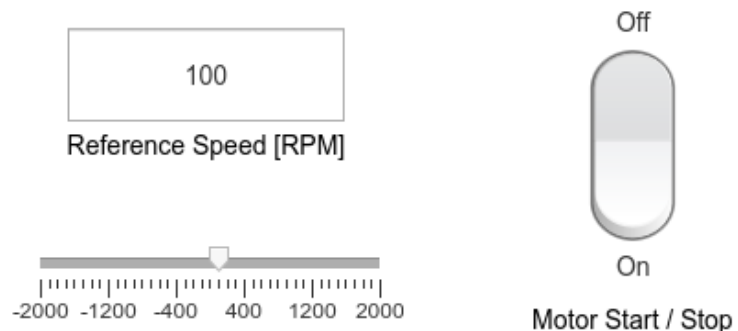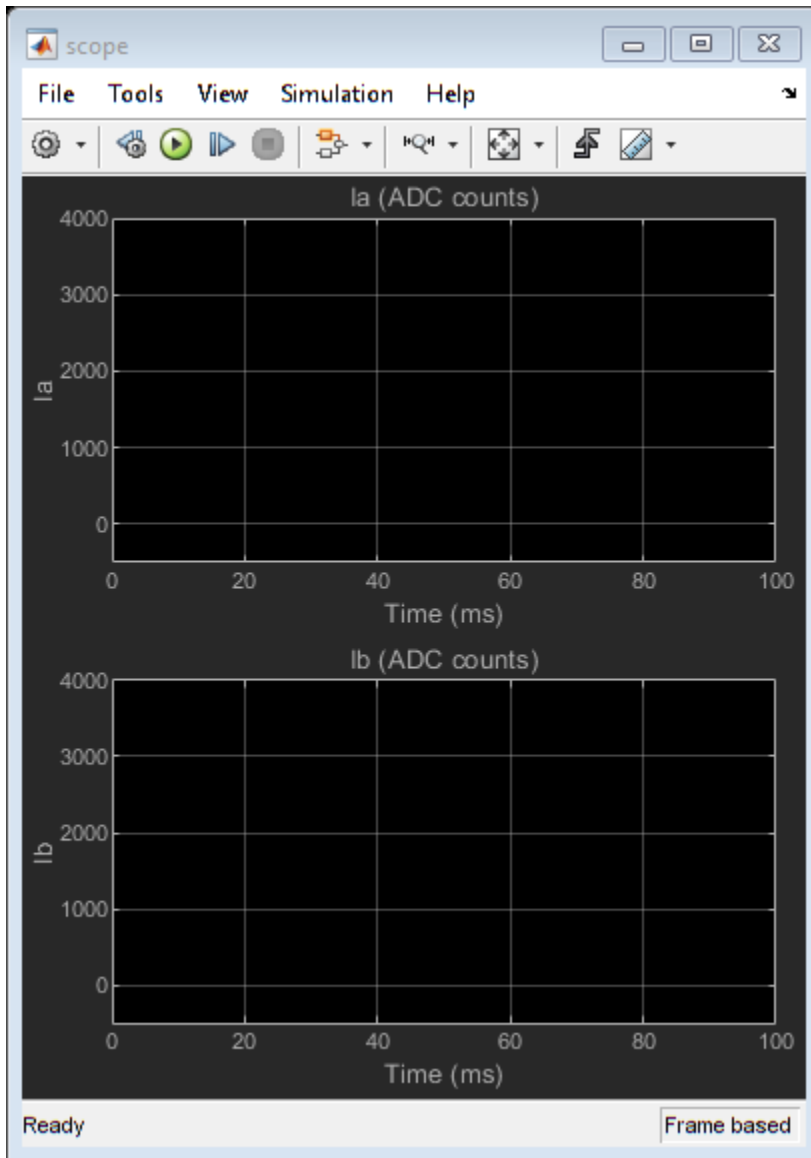
```
100
```
Reference Speed [RPM]

-2000 -1200 -400   400   1200 2000

Off

On

Motor Start / Stop

Configuration Panel
Group
- ⦿ TI F28069M
- ○ TI F28379D

HOST
Serial
Setup

Host Serial Setup

TX

Ia (ADC counts)

Ib (ADC counts)

RX

scope

Copyright 2020 The MathWorks, Inc.

4-11

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

8. In the Host Serial Setup block mask of the host model, select a **Port name**.

9. Select a target (either TI F28069M or TI F28379D) in the **Configuration Panel** of the host model.

10. Enter the Reference Speed value in the host model.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On, to start running the motor.

13. After the motor is running, observe the ADC counts for the I_a and I_b currents in the Time Scope.

NOTE: This example may not allow the motor to run at full capacity. Begin running the motor at a small speed. In addition, it is recommended to change the Reference Speed in small steps (for

example, for a motor having a base speed of 3000 rpm, start running the motor at 500 rpm and then increase or decrease the speed in steps of 200 rpm).

If the motor does not run, change the position of the Start / Stop Motor switch to Off, to stop the motor and change the Reference Speed in the host model. Then, change the position of the Start / Stop Motor switch to On, to run the motor again.

Generate Code and Run Model to Calibrate ADC Offset

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. Disconnect the motor wires for three phases from the hardware board terminals.

4. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see "Model Configuration Parameters" on page 2-2.

5. Load a sample program to CPU2 of LAUNCHXL-F28379D (for example, program that operates the CPU2 blue LED using GPIO31) to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

NOTE: Ignore the warning message "Multitask data store option in the Diagnostics page of the Configuration Parameter Dialog is none" displayed by the model advisor, by clicking the Always Ignore button. This is part of the intended workflow.

7. Click the **host model** hyperlink in the target model to open the associated host model.

8. In the Host Serial Setup block mask of the host model, select a **Port name**.

9. Click **Run** on the **Simulation** tab to run the host model.

10. Observe the ADC counts for the I_a and I_b currents in the Time Scope. The average values of the ADC counts are the ADC offset corrections for the currents I_a and I_b. To obtain the average (median) values of ADC counts:

- In the **Scope** window, navigate to **Tools > Measurements** and select **Signal Statistics** to display the **Trace Selection** and **Signal Statistics** areas.
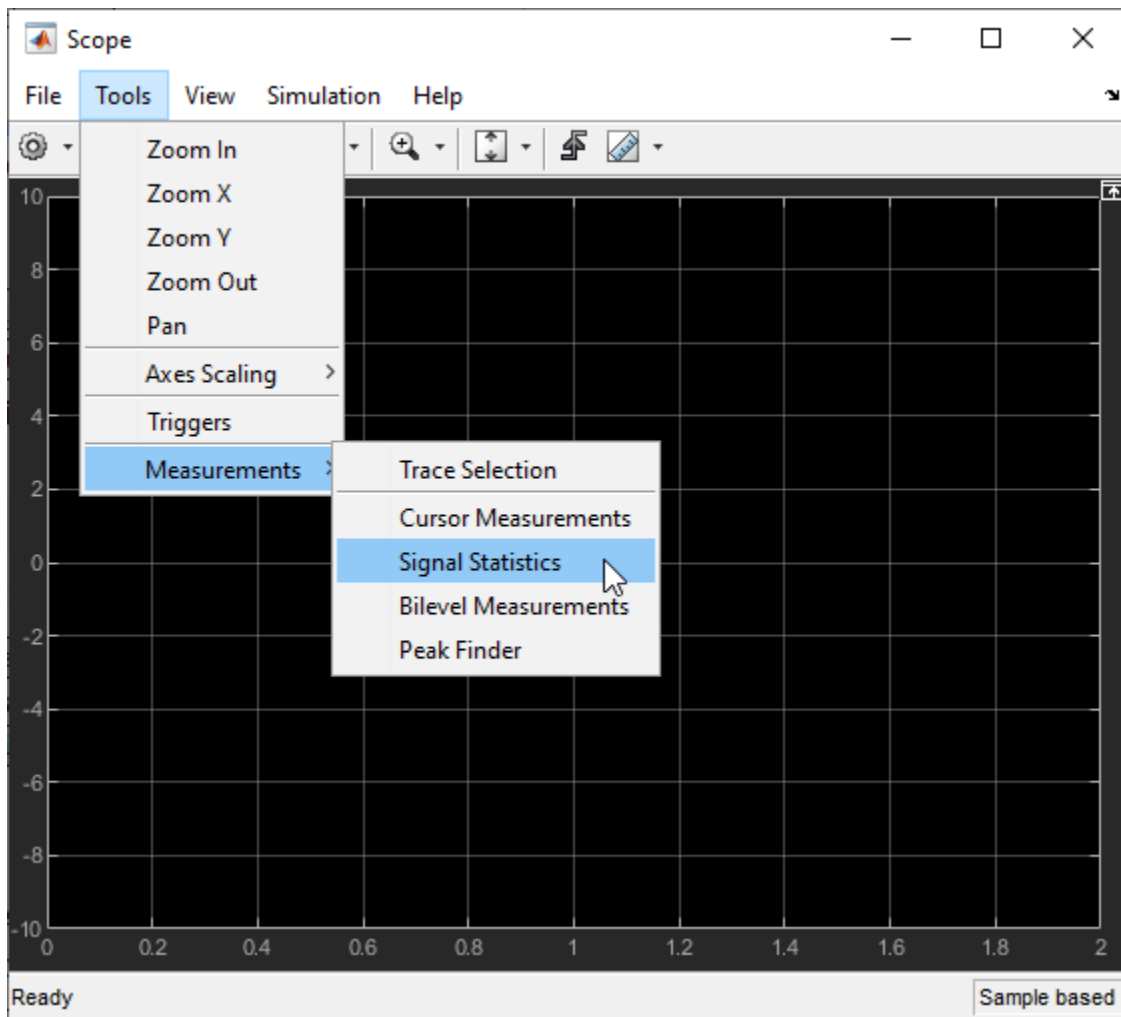
- Under **Trace Selection**, select a signal (I_a or I_b). The characteristics of the selected signal are displayed in the **Signal Statistics** pane. You can see the median value of the selected signal in the Median field.
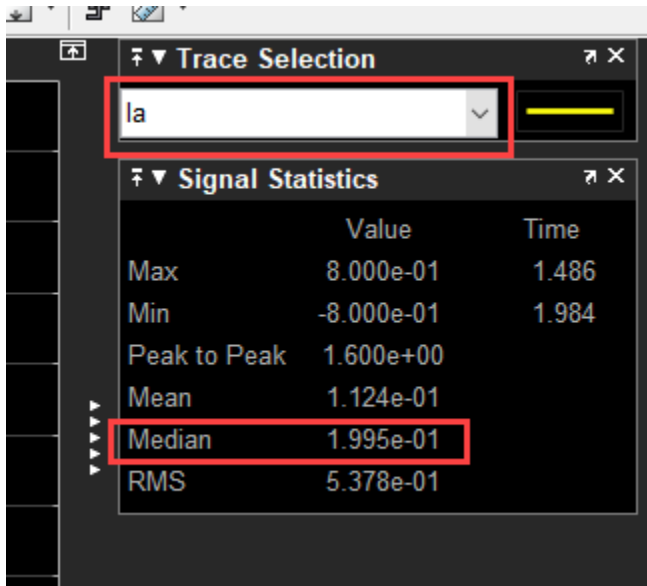
For the Motor Control Blockset examples, update the computed ADC (or current) offset value in the inverter.CtSensAOffset and inverter.CtSensBOffset variables in the model initialization script linked to the example. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Tune Control Parameter Gains in Hardware and Validate Plant

This example uses field-oriented control (FOC) to run a three-phase permanent magnet synchronous motor (PMSM) in different modes of operation for plant validation. FOC algorithm implementation needs the real-time feedback of the rotor position. This example uses a quadrature encoder sensor to measure the rotor position. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

The example runs the motor in these modes:

- **Stop** - In this mode, the motor stops running because the inverter outputs zero volts.

- **Open loop** - In this mode, the controller uses open-loop control to run the motor. You can use the **Operating Mode Variables** area of the host model to change the output voltage of the inverter and the rotor speed. Use the **Monitor** area to select the speed and rotor position values to display them on the scope for monitoring.

- **Torque control** - In this mode, the controller uses a torque control algorithm to run the motor. You can use the **Operating Mode Variables** area of the host model to change the I_d reference and I_q reference currents.

You can also lock the rotor by turning the slider switch to the Pos lock position that sets the rotor position to zero. Therefore, in this mode, the controller receives the position feedback as zero because the motor stops running. If you turn the switch to the Unlock position, the motor runs and the controller receives position feedback from the quadrature encoder (you can monitor this value by using the Position_meas signal in the **Monitor** area of host model). You can use the scope to monitor the two debug signals (Monitor Signal #1 and Monitor Signal #2) that you select in the **Monitor** area. Therefore, you can use the slider switch to tune the torque control gain parameters.

- **Speed control** - In this mode, the controller uses a speed control algorithm to run the motor. You can use the **Operating Mode Variables** area of the host model to change the Speed Reference value (in per-unit) of the rotor. You can use the scope to monitor the two debug signals (Monitor Signal #1 and Monitor Signal #2) that you select in the **Monitor** area. For information related to the per-unit system, see "Per-Unit System" on page 6-15.

To further control the motor, you can also use the **Control loop gains** area of the host model to change the control parameters of the d-axis and q-axis current controllers and the speed controller.

You can use this example to run the motor in open-loop control, torque control, and speed control modes. You can also use this example for tuning the hardware gains and validating the plant model.

Caution: Stop the motor first before transitioning from one operating mode to another.

You can select one of these operating modes in the Control area of the host model:

- Stop
- Open loop run
- Torque control
- Speed control

Model

The example includes this model:

- **mcb_pmsm_operating_mode_f28379d**

You can use the model for both simulation and code generation. You can use the open_system command to open the Simulink® model:

```
open_system('mcb_pmsm_operating_mode_f28379d.slx');
```

Control Parameter Gain Tuning (Manual) in Hardware and Plant Validation

Note: This example requires a TI F28379D LaunchPad with a BOOSTXL-DRV8305 booster pack connected to a PMSM Motor with QEP Sensor
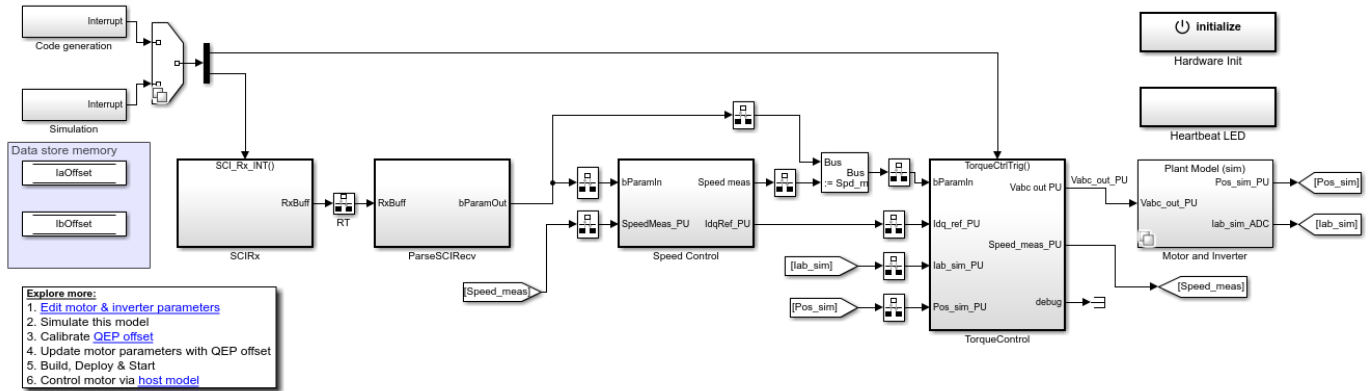
Copyright 2020 The MathWorks, Inc.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

1. Motor Control Blockset™

2. Embedded Coder®

3. Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors

4. Fixed-Point Designer™

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation only for the Speed Control operating mode. Follow these steps to simulate the model.

1. Open the model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the model, run (and control) the motor in a selected operating mode, and monitor the debug signals of the model.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter:
  mcb_pmsm_operating_mode_f28379d

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

5. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

6. To ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1, load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx).

NOTE:

- Do not directly switch between the open-loop run, torque control, and speed control operating modes. Always stop the motor before changing the operating mode.

- Before you run the motor in speed control mode for the first time, run the motor in open-loop to determine the quadratue encoder index. This helps to start the motor smoothly in the closed-loop speed control mode.

Instructions for Open-Loop Run Mode:

1. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

2. Click the **host model** hyperlink in the target model to open the associated host model.

3. In the Host Serial Setup block mask of the host model, select a **Port name**.

4. Click **Run** on the **Simulation** tab to run the host model.

5. Select **Stop** in the **Control** area to stop the motor.

6. Select **Open loop run** to start the motor.

Instructions for Torque Control Mode:

1. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

2. Click the **host model** hyperlink in the target model to open the associated host model.

3. In the Host Serial Setup block mask of the host model, select a **Port name**.

4. Click **Run** on the **Simulation** tab to run the host model.

5. Select **Stop** in the **Control** area to stop the motor.

6. Enter the value 0 (per-unit) in the **Id Reference** and **Iq Reference** fields in the **Operating Mode Variables** area.

7. Select **Torque control** in the **Control** area.

8. Move the slider switch to **Unlock** position in the **Operating Mode Variables** area.

9. Select **Iq_ref** for **Monitor Signal #1** and **Iq_meas** for **Monitor Signal #2** in the **Monitor** area.

10. Enter the value 0.1 (per-unit) in the in the **Iq Reference** field (in the **Operating Mode Variables** area) to start running the motor.

11. Open the scope in the host model and monitor the Iq_ref and Iq_meas current signals.

Note: The motor can reach high speeds if you run it under no load condition in this operating mode. In addition, the motor will not meet the Iq reference current under no load condition in this operating mode.

Instructions for Speed Control Mode:

1. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

2. Click the **host model** hyperlink in the target model to open the associated host model.

3. In the Host Serial Setup block mask of the host model, select a **Port name**.

4. Click **Run** on the **Simulation** tab to run the host model.

5. Select **Stop** in the **Control** area to stop the motor.

6. Enter the value 0.5 (per-unit) in the **Speed Reference** field in the **Operating Mode Variables** area.

7. Select **Speed control** in the **Control** area.

8. Select **Speed_ref** for **Monitor Signal #1** and **Speed_meas** for **Monitor Signal #2** in the **Monitor** area.

9. Open the scope in the host model and monitor the Speed_ref and Speed_meas output signals.

Instructions for Tuning Gain of Torque Controller:

1. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

2. Click the **host model** hyperlink in the target model to open the associated host model.

3. In the Host Serial Setup block mask of the host model, select a **Port name**.

4. Click **Run** on the **Simulation** tab to run the host model.

5. Select **Stop** in the **Control** area to stop the motor.

6. Select **Torque control** in the **Control** area.

7. Turn the slider switch to **Pos lock** position in the **Operating Mode Variables** area.

8. Select **Id_ref** for **Monitor Signal #1** and **Id_meas** for **Monitor Signal #2** in the **Monitor** area.

9. Enter the value 0.2 (per-unit) in the **Id Reference** field in the **Operating Mode Variables** area.

10. Open the scope and monitor the step response signal.

11. Tune the control gains Kp and Ki for the d-axis current controller.

Instructions for Tuning Gain of Speed Controller:

1. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

2. Click the **host model** hyperlink in the target model to open the associated host model.

3. In the Host Serial Setup block mask of the host model, select a **Port name**.

4. Click **Run** on the **Simulation** tab to run the host model.

5. Select **Stop** in the **Control** area to stop the motor.

6. Select **Speed control** in the **Operating Mode Variables** area.

7. Select **Speed_ref** for **Monitor Signal #1** and **Speed_meas** for **Monitor Signal #2** in the **Monitor** area.

8. Enter the value 0.5 (per-unit) in the **Speed Reference** field in the **Operating Mode Variables** area.

9. Open the scope and observe the reference and the measured speed values.

10. Enter the value 0.8 (per-unit) in the **Speed Reference** field.

11. Observe the speed step response in the scope.

12. Tune the control gains Kp and Ki for the speed controller.

Instructions for Validating Plant Model:

1. Click the **host model** hyperlink in the target model to open the associated host model.

2. In the Host Serial Setup block mask of the host model, select a **Port name**.

3. Click **Run** on the **Simulation** tab to run the host model.

4. Open the target model using this command:
open_system('mcb_pmsm_operating_mode_f28379d.slx');

5. Change the position of the Manual Switch (in mcb_pmsm_operating_mode_f28379d.slx/Speed Control/Speed_control) to Simulate Step Speed Input.

6. Simulate the model mcb_pmsm_operating_mode_f28379d.slx.

7. Open the scope in the model to observe the step response of the speed controller.

8. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

9. Select **Stop** in the **Control** area of the host model to ensure that the motor is not running.

10. Select **Speed control** in the **Operating Mode Variables** area.

11. Select **Speed_ref** for **Monitor Signal #1** and **Speed_meas** for **Monitor Signal #2** in the **Monitor** area.

12. Enter the value 0.2 (per-unit) in the **Speed Reference** field in the **Operating Mode Variables** area.

13. Open the scope and observe the reference and the measured speed values.

14. Enter the value 0.5 (per-unit) in the **Speed Reference** field.

15. Observe the speed step response in the scope.

16. Compare the speed step responses obtained in steps 6 (with simulation) and 15 (with code generation).

Note: You can also use the open_system command to open the host model:

```
open_system('mcb_host_mode_control.slx');
```

Host model for Control Parameter Gain Tuning (Manual) in Hardware and Plant Validation

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_operating_mode_f28379d
2. You should see and verify the variables from
   the target model in the base workspace.

Steps:
1. Select the port name in Serial 1 tab of Host
   Serial Setup block.
2. Caution: Stop the motor when switching
   between the modes

HOST
Serial
Setup

Host Serial Setup
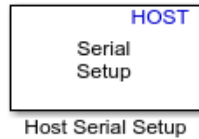
Control

Select Motor operating mode

- ⦿ Stop
- ○ Open loop run
- ○ Torque control
- ○ Speed control

Operating Mode Variables

Motor torque control mode

| -- | -- | Unlock 🔘 Pos lock |

Id Reference   Iq Reference

Motor speed control mode

| -- |

Speed Reference

Control loop gains

d-axis current controller

| -- | -- |
Kp Gain   Ki Gain

q-axis current controller

| -- | -- |
Kp Gain_   Ki Gain_

Speed controller

| -- | -- |
Kp Gain__   Ki Gain__

Monitor

Monitor Signal #1	Monitor Signal #2
⦿ V_alpha	⦿ V_alpha
○ V_beta	○ V_beta
○ I_alpha	○ I_alpha
○ I_beta	○ I_beta
○ Va_out	○ Va_out
○ Vb_out	○ Vb_out
○ Vc_out	○ Vc_out
○ Ia_meas	○ Ia_meas
○ Ib_meas	○ Ib_meas
○ Id_ref	○ Id_ref
○ Id_meas	○ Id_meas
○ Vd_ctrl_out	○ Vd_ctrl_out
○ Iq_ref	○ Iq_ref
○ Iq_meas	○ Iq_meas
○ Vq_ctrl_out	○ Vq_ctrl_out
○ Position_meas	○ Position_meas
○ Speed_ref	○ Speed_ref
○ Speed_meas	○ Speed_meas

1

Serial Communication

Scope
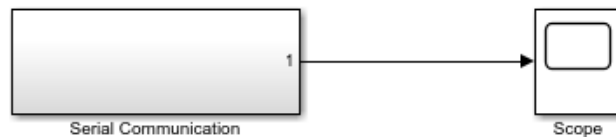
Copyright 2020 The MathWorks, Inc.
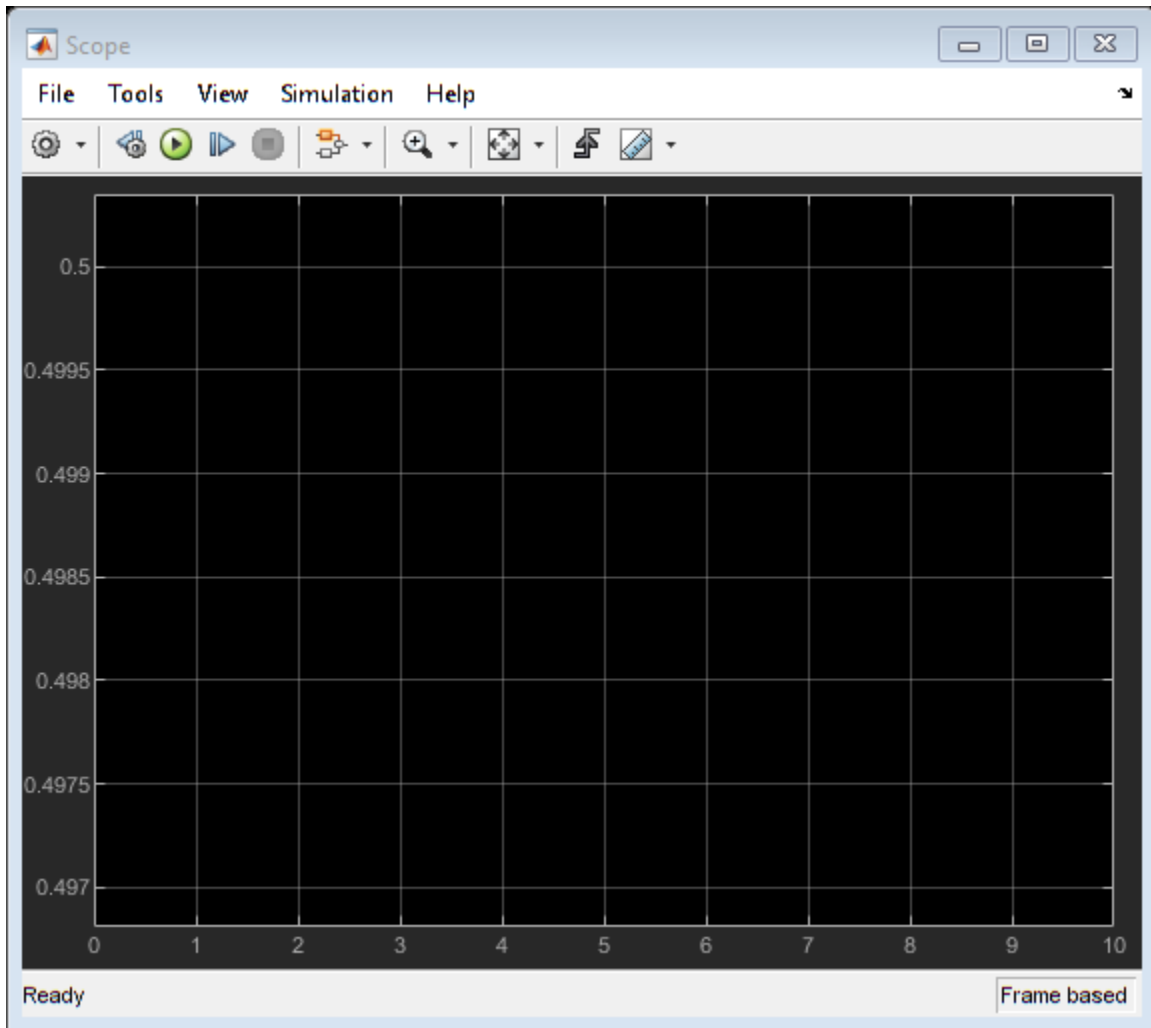
In the **Control loop gains** area, you must enter the gain values that can be represented by the datatype defined in the model initialization script.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

Tune PI Controllers by Using Field Oriented Control Autotuner

This example computes the gain values of the PI controllers within the speed and current controllers by using the Field Oriented Control Autotuner block. For details about field-oriented control, see "Field-Oriented Control (FOC)" on page 4-2.

The example supports simulation only. When you simulate the example, the model uses the crude values of gains for the PI controllers to achieve the steady state of speed-control operation.

The model begins tuning only in the steady state. It introduces disturbances depending on the controller goals (bandwidth and phase margin), in the controller output. The model uses the system response to the disturbances, to calculate the optimal controller gain.

Model

The example includes this model:

* **mcb_pmsm_foc_autotuner**

You can use this model only for simulation. You can use the open_system command to open the Simulink® model:

open_system('mcb_pmsm_foc_autotuner.slx');
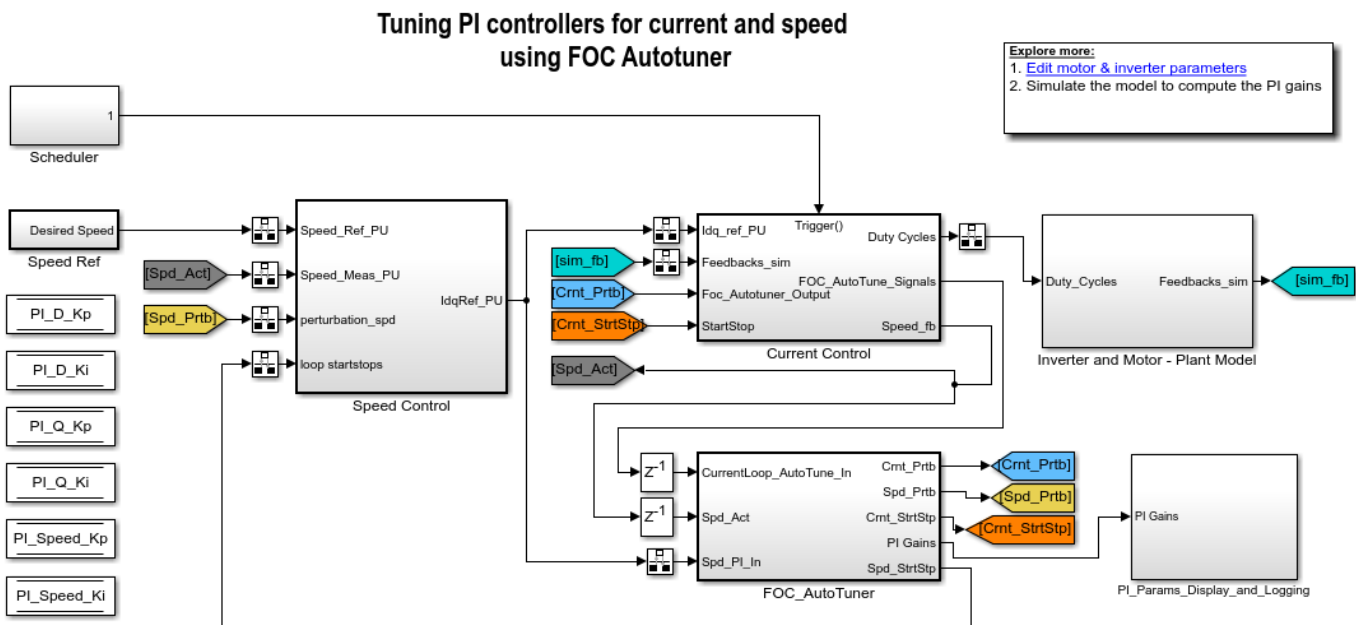
Copyright 2020 The MathWorks, Inc.

Required MathWorks® Products for Simulation

* Motor Control Blockset™
* Simulink Control Design™

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Note: In addition to the preceding products, you also need these products to use the parameter estimation tool:

- Embedded Coder®

- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors

Simulate and Run Model to Compute PI Controller Gains

1. Open the target model.

2. Click **Run** on the **Simulation** tab to simulate the target model.

3. Observe the computed PI controller gain values in the Display blocks available in the PI_Params_Display_and_Logging subsystem.

4. Update any target model with these gain values so that it brings the motor to a steady-speed state quickly.

Field-Oriented Control of PMSM by Using Hall Sensor

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). The FOC algorithm requires rotor position feedback, which is obtained by a Hall sensor. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

This example uses the Hall sensor to measure the rotor position. A Hall effect sensor varies its output voltage based on the strength of the applied magnetic field. A PMSM consists of three Hall sensors located electrically 120 degrees apart. A PMSM with this setup can provide six valid combinations of binary states (for example, 001,010,011,100,101, and 110). The sensor provides the angular position of the rotor in the multiples of 60 degrees, which the controller uses to compute the angular velocity. The controller can then use the angular velocity to compute an accurate angular position of the rotor.
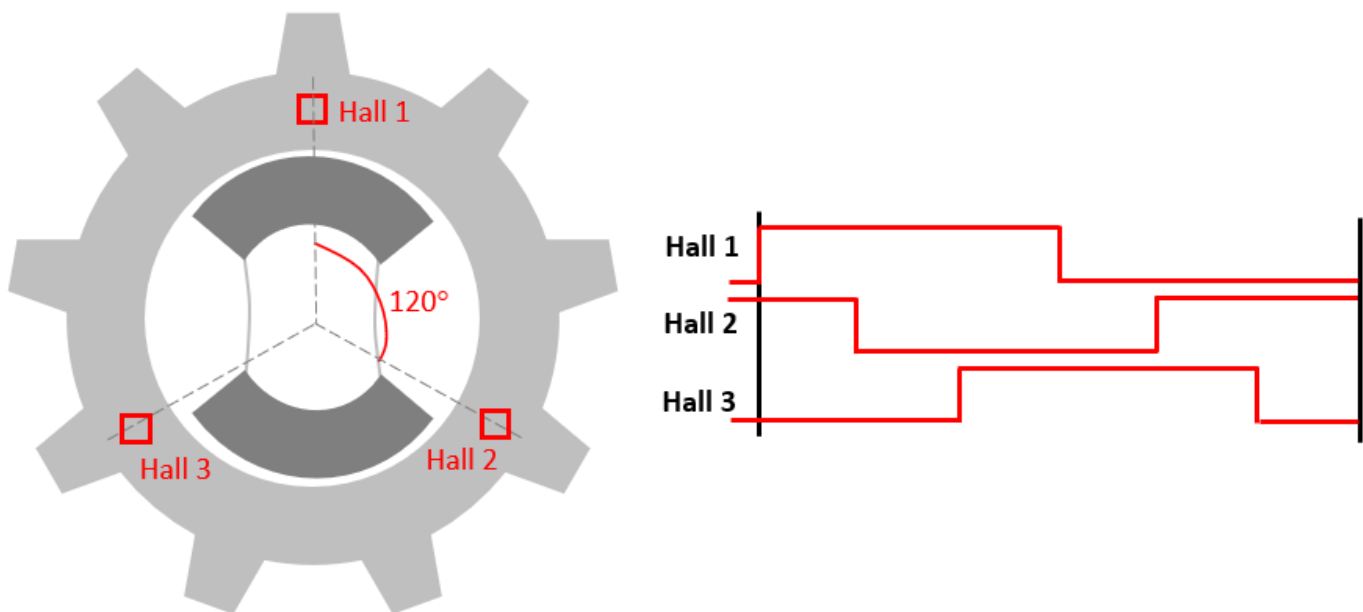
Models

The example includes these models:

- **mcb_pmsm_foc_hall_f28069m**

- **mcb_pmsm_foc_hall_f28379d**

You can use these models for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_foc_hall_f28069m.slx');
```

Field-Oriented Control for PMSM with Hall sensor

Note: This example requires a TI F28069m with DRV8312 connected to a PMSM Motor with Hall Sensor

Copyright 2020 The MathWorks, Inc.

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

1. For the model: **mcb_pmsm_foc_hall_f28069m**

- Motor Control Blockset™
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_foc_hall_f28379d**

- Motor Control Blockset™

To generate code and deploy model:

1. For the model: **mcb_pmsm_foc_hall_f28069m**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_foc_hall_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. To simulate the model, click **Run** on the **Simulation** tab.

3. To view and analyze the simulation results, click **Data Inspector** on the **Simulation** tab.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

• F28069M controller card + DRV8312-69M-KIT inverter: **mcb_pmsm_foc_hall_f28069m**

For connections related to the preceding hardware configuration, see "F28069 control card configuration" on page 7-2.

• LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: **mcb_pmsm_foc_hall_f28069m**

• LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter: **mcb_pmsm_foc_hall_f28379d**

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the Analog-to-Digital Converter (ADC) or current offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Compute the Hall sensor offset value and update it in the model initialization script associated with the target model. For instructions, see "Hall Offset Calibration for PMSM Motor" on page 4-72

5. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

6. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the model to the hardware.

8. In the target model, click the **host model** hyperlink to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_foc_host_model_f28069m.slx');
```

PMSM FOC Host

HOST
Serial
Setup

Host Serial Setup

Note:
1. Select the serial port in
   'Host Serial Setup' (Blue Color)
2. Use 'Motor Start / Stop' switch to enable and
   disable motor control.
3. Input speed request using 'Reference Speed'
   text box or sliding bar.
4. Observe the actual speed of motor and
   phase A current in the scope.

Off

1200

Reference Speed

TX

On

Motor Start / Stop

Speed (RPM)

Ia (amps)

RX

-6000 -4500 -3000 -1500 0 1500 3000 4500 6000

Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

9. In the Host Serial Setup block mask of the host model, select a **Port name**.

10. Update the Reference Speed value in the host model.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On, to start running the motor.

13. Observe the debug signals from the RX subsystem, in the Time Scope of host model.

Note: If you are using a F28379D based controller, you can also select the debug signals that you want to monitor.

Field-Oriented Control of PMSM Using Quadrature Encoder

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). The FOC algorithm requires rotor position feedback, which is obtained by a quadrature encoder sensor. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2

This example uses the quadrature encoder sensor to measure the rotor position. The quadrature encoder sensor consists of a disk with two tracks or channels that are coded 90 electrical degrees out of phase. This creates two pulses (A and B) that have a phase difference of 90 degrees and an index pulse (I). Therefore, the controller uses the phase relationship between A and B channels and the transition of channel states to determine the direction of rotation of the motor.
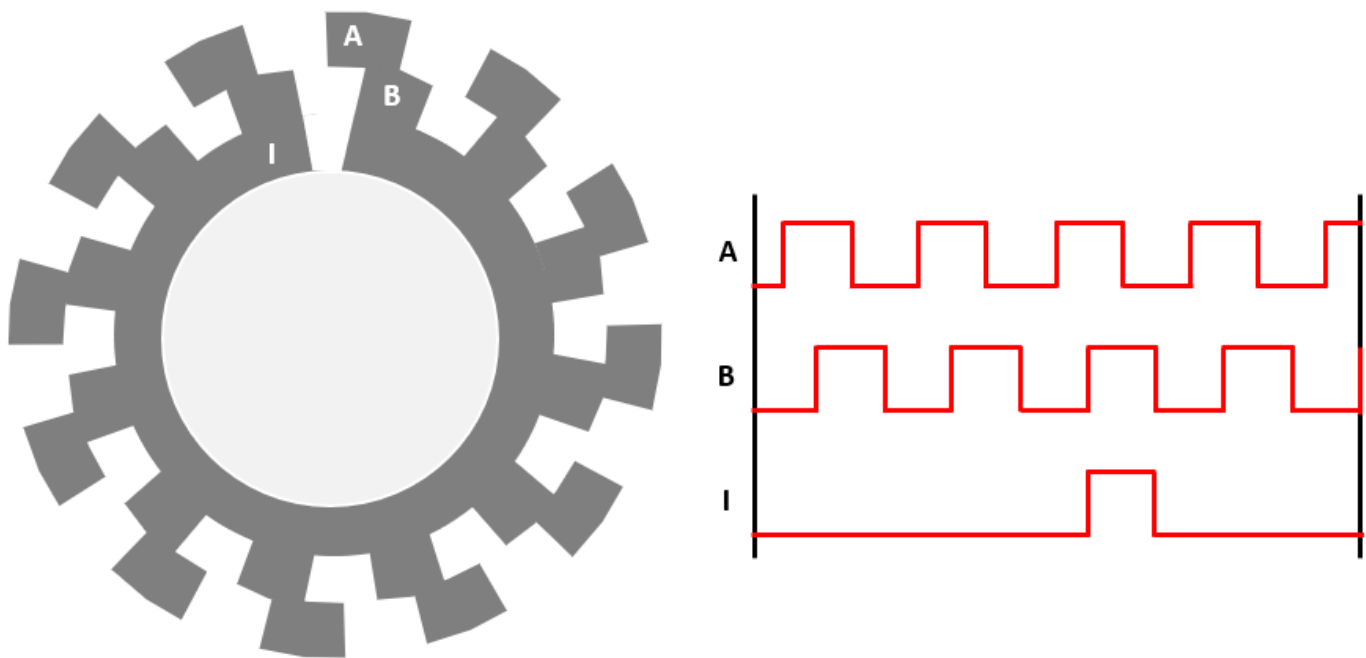
Models

The example includes these models:

- **mcb_pmsm_foc_qep_f28069m**

- **mcb_pmsm_foc_qep_f28069LaunchPad**

- **mcb_pmsm_foc_qep_f28379d**

- **mcb_pmsm_foc_qep_f28379d_GaN**

You can use these models for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller.

```
open_system('mcb_pmsm_foc_qep_f28069m.slx');
```

Field-Oriented Control for PMSM with QEP sensor

Note: This example requires a TI F28069m with DRV8312 connected to a PMSM Motor with QEP Sensor

Copyright 2020 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Use Offset Computation model to find out position offset.
3. Update offset in Init script to variable 'pmsm.PositionOffset'
4. Build, Deploy & Start
5. Control motor via host model

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

1. For the models: **mcb_pmsm_foc_qep_f28069m** and **mcb_pmsm_foc_qep_f28069LaunchPad**

- Motor Control Blockset™
- Fixed-Point Designer™

2. For the models: **mcb_pmsm_foc_qep_f28379d** and **mcb_pmsm_foc_qep_f28379d_GaN**

- Motor Control Blockset™

To generate code and deploy model:

1. For the models: **mcb_pmsm_foc_qep_f28069m** and **mcb_pmsm_foc_qep_f28069LaunchPad**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

2. For the models: **mcb_pmsm_foc_qep_f28379d** and **mcb_pmsm_foc_qep_f28379d_GaN**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- F28069M control card + DRV8312-69M-KIT inverter: **mcb_pmsm_foc_qep_f28069m**

For connections related to the preceding hardware configuration, see "F28069 control card configuration" on page 7-2.

- LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: **mcb_pmsm_foc_qep_f28069LaunchPad**

- LAUNCHXL-F28379D controller + BOOSTXL-3PHGANINV inverter: **mcb_pmsm_foc_qep_f28379d_GaN**

- LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter: **mcb_pmsm_foc_qep_f28379d**

NOTE: When using BOOSTXL-3PHGANINV inverter, ensure that proper insulation is available between bottom layer of BOOSTXL-3PHGANINV and the LAUNCHXL board.

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

5. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

6. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

8. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller.

```
open_system('mcb_pmsm_foc_host_model_f28069m.slx');
```

PMSM FOC Host

HOST
Serial
Setup

Host Serial Setup

Note:
1. Select the serial port in
   'Host Serial Setup' (Blue Color)
2. Use 'Motor Start / Stop' switch to enable and
   disable motor control.
3. Input speed request using 'Reference Speed'
   text box or sliding bar.
4. Observe the actual speed of motor and
   phase A current in the scope.

Off

1200

Reference Speed

TX

On

Motor Start / Stop

Speed (RPM)

Ia (amps)

RX

-6000 -4500 -3000 -1500 0 1500 3000 4500 6000

Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

9. In the Host Serial Setup block mask of the host model, select a **Port name**.

10. Update the Reference Speed value in the host model.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On, to start running the motor.

13. Observe the debug signals from the RX subsystem, in the Time Scope of host model.

Note: If you are using a F28379D based controller, you can also select the debug signals that you want to monitor.

Field-Weakening Control (with MTPA) of PMSM

This example implements the field-oriented control (FOC) technique to control the torque and speed of a three-phase permanent magnet synchronous motor (PMSM). The FOC algorithm requires rotor position feedback, which is obtained by a quadrature encoder sensor. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

Field-Weakening Control

When you use the FOC algorithm to run a motor with rated flux, the maximum speed is limited by the stator voltages, rated current, and back emf. This speed is called the base speed. Beyond this speed, the operation of the machine is complex because the back emf is more than the supply voltage. However, if you set the d-axis stator current (Id) to a negative value, the rotor flux linkage reduces, which allows the motor to run above the base speed. This operation is known as field-weakening control of the motor.
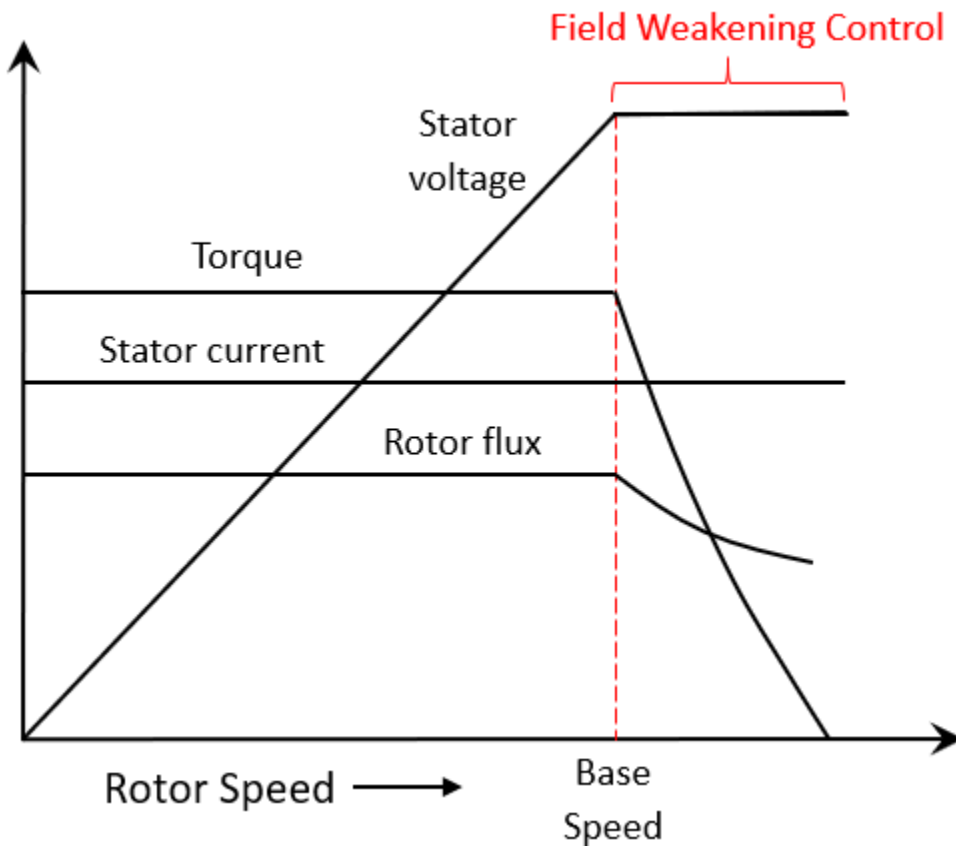
Depending upon the connected load and rated current of the machine, the reference d-axis current (I_d) in the field-weakening control also limits the reference q-axis current (I_q), and therefore, limits the torque output. Therefore, the motor operates in the constant torque region until the base speed. It operates in the constant power region with a limited torque above the base speed, as illustrated in the preceding figure.

The computations for the reference current I_d depend on the motor and inverter parameters.

Note:

- For some surface PMSMs, (depending upon the parameters) it may not be possible to achieve higher speeds at the rated current. To achieve higher speeds, you need to overload the motor with maximum currents that are higher than the rated current (if the thermal conditions of the machine are within the permissible limits).

- When you operate the motor above the base speed, we recommend that you monitor the temperature of the motor. During motor operation, if the motor temperature rises beyond the temperature recommended by the manufacturer, turn-off the motor for safety reasons.

- When you operate the motor above the base speed, we recommend that you increment the speed reference in small steps, to avoid the dynamics of field weakening that can make some systems unstable.

Maximum Torque Per Ampere (MTPA)

For the interior PMSMs, the saliency in the magnetic circuit of rotor results in higher $\frac{L_q}{L_d}$ ratio (greater than 1). This produces reluctance torque in the rotor (in addition to the existing electromagnetic torque). For more information, see MTPA Control Reference.

Therefore, you can operate the machine at an optimum combination of I_d and I_q, and obtain a higher torque for the same stator current, $I_{\max} = \sqrt{I_d^2 + I_q^2}$.

This increases the efficiency of the machine, because the stator current losses are minimized. The algorithm that you use to generate the reference I_d and I_q currents for producing maximum torque in the machine, is called Maximum Torque Per Ampere (MTPA).
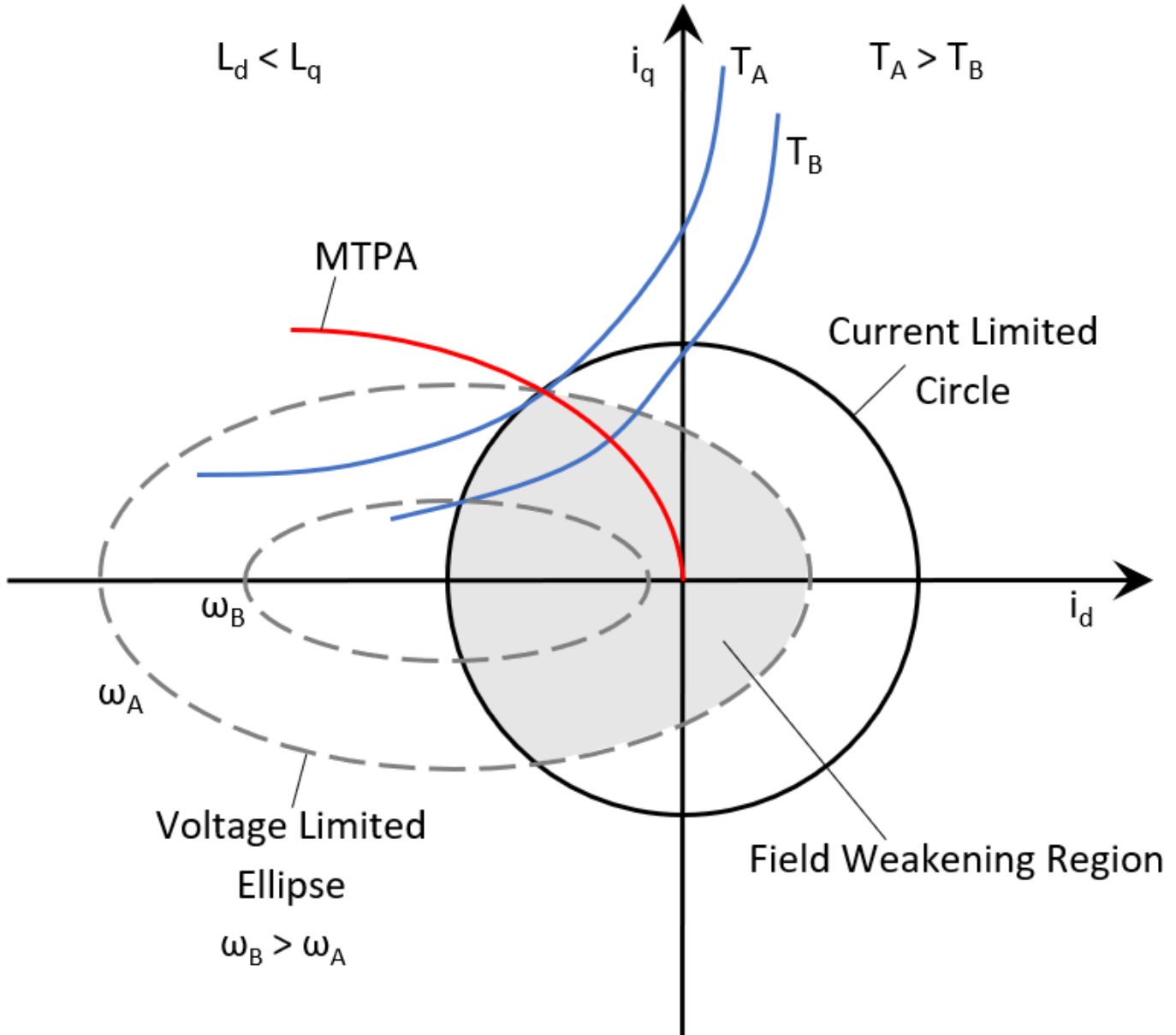
For an Interior PMSM (IPMSM), this example computes the reference I_d and I_q currents using the MTPA method until the base speed. For a Surface PMSM (SPMSM), the example achieves MTPA operation by using a zero d-axis reference current, until the base speed.

To operate the motor above the base speed, this example computes the reference I_d and I_q for MTPA and field-weakening control, depending upon the motor type. For a Surface PMSM, Constant Voltage Constant Power (CVCP) control method is used. For an Interior PMSM, Voltage and Current Limited Maximum Torque (VCLMT) control method is used.

For information related to MTPA Control Reference block, see MTPA Control Reference.

Target Communication

For hardware implementation, this example uses a host and a target model. The host model, running on the host computer, communicates with the target model deployed to the hardware connected to the motor. The host model uses serial communication to command the target model and run the motor in a closed-loop control.

Models

This examples uses multiple models for these hardware configurations:

Speed control of PMSM with field-weakening and MTPA:

- **mcb_pmsm_fwc_qep_f28069LaunchPad**

- **mcb_pmsm_fwc_qep_f28379d**

Torque control of PMSM with MTPA:

- **mcb_pmsm_mtpa_qep_f28069LaunchPad**

- **mcb_pmsm_mtpa_qep_f28379d**

You can use these models for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_fwc_qep_f28069LaunchPad.slx');
```
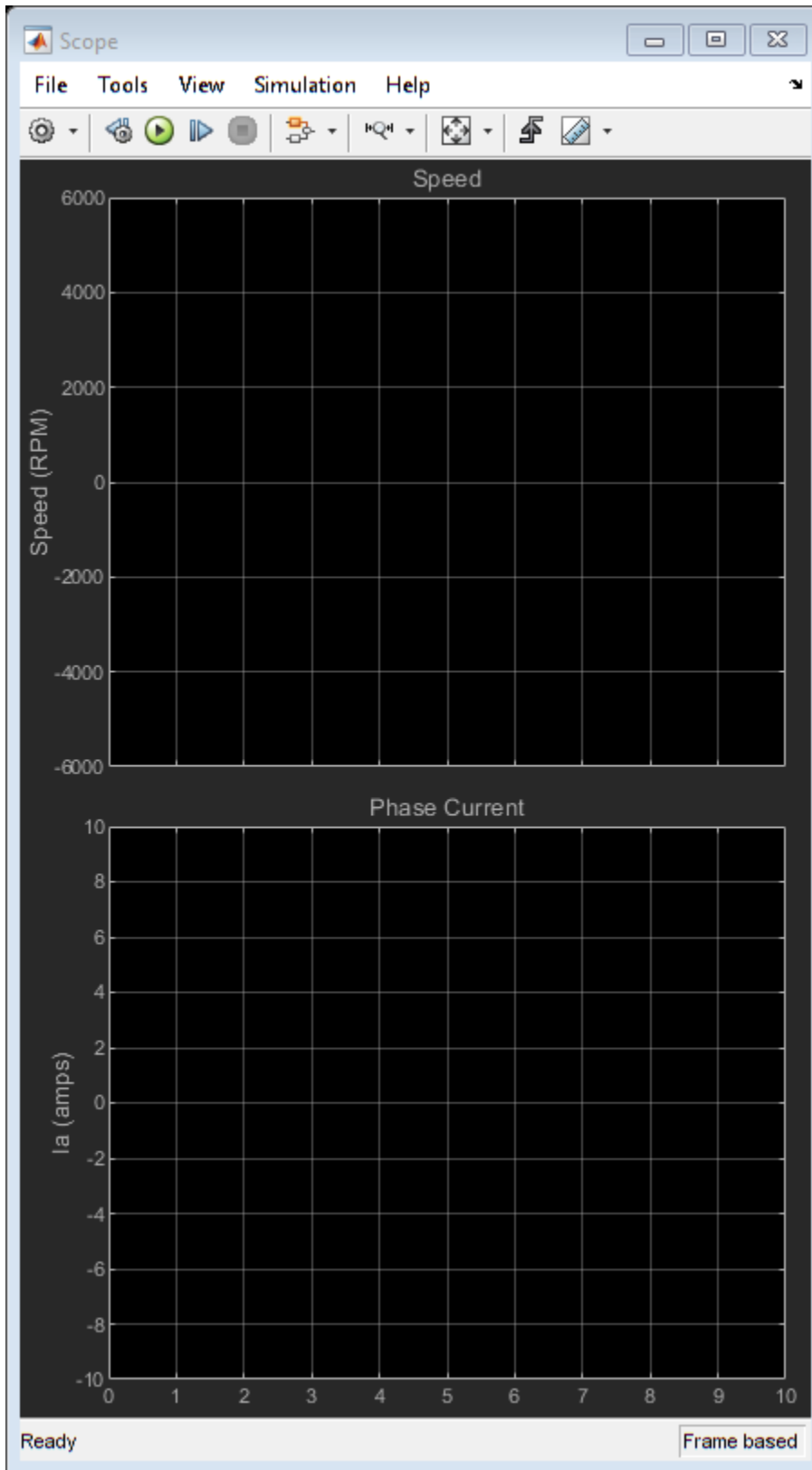

PMSM Field Weakening Control with MTPA

Required MathWorks® Products

To simulate model:

1. For the models: **mcb_pmsm_fwc_qep_f28069LaunchPad** and
mcb_pmsm_mtpa_qep_f28069LaunchPad

- Motor Control Blockset™
- Fixed-Point Designer™

2. For the models: **mcb_pmsm_fwc_qep_f28379d** and **mcb_pmsm_mtpa_qep_f28379d**

- Motor Control Blockset™

To generate code and deploy model:

1. For the models: **mcb_pmsm_fwc_qep_f28069LaunchPad** and
mcb_pmsm_mtpa_qep_f28069LaunchPad

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

2. For the models: **mcb_pmsm_fwc_qep_f28379d** and **mcb_pmsm_mtpa_qep_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor, inverter, and position sensor calibration parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter and position sensor calibration parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate (Speed Control and Torque Control) Models

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Analyze simulation results for Speed Control Model

The model uses the per-unit system to represent speed, currents, voltages, torque, and power. Type PU System at the workspace to see the conversion of one per-unit value into SI units for these quantities.

Observe the dynamics of the system for the speed and current controllers. In addition, notice the negative Id currents for motor operation above the base speed.

Note:

- For some surface PMSMs, (depending upon the parameters) it may not be possible to achieve higher speeds at the rated current. To achieve higher speeds, you need to overload the motor with maximum currents that are higher than the rated current (if the thermal conditions of the machine are within the permissible limits).

- When you operate the motor above the base speed, we recommend that you monitor the temperature of motor. During motor operation, if the motor temperature rises beyond the temperature recommended by the manufacturer, turn-off the motor for safety reasons.

- When you operate the motor above the base speed, we recommend that you increment the speed reference in small steps, to avoid the dynamics of field weakening that can make some systems unstable.

Analyze simulation results for Torque Control Model

Run simulation with the Id and Iq reference currents generated by these three methods:

1. Generate reference currents by using the MTPA Control Reference Block.

2. Generate the MTPA reference currents manually by using the Vector Control Reference Block.

3. Generate the Control Reference without MTPA.

The first method uses mathematical computations to determine the reference currents Id and Iq, after assuming linear inductances.

Use the second method to manually generate the MTPA look-up tables for motors with non-linear inductances. You can illustrate this with the Id and Iq references generated by sweeping the torque angle between $+(\pi/2)$ to $-(\pi/2)$.

Use the last method to obtain the reference currents without the MTPA algorithm.

You can compare the torque and power generated by these three methods in the data inspector.
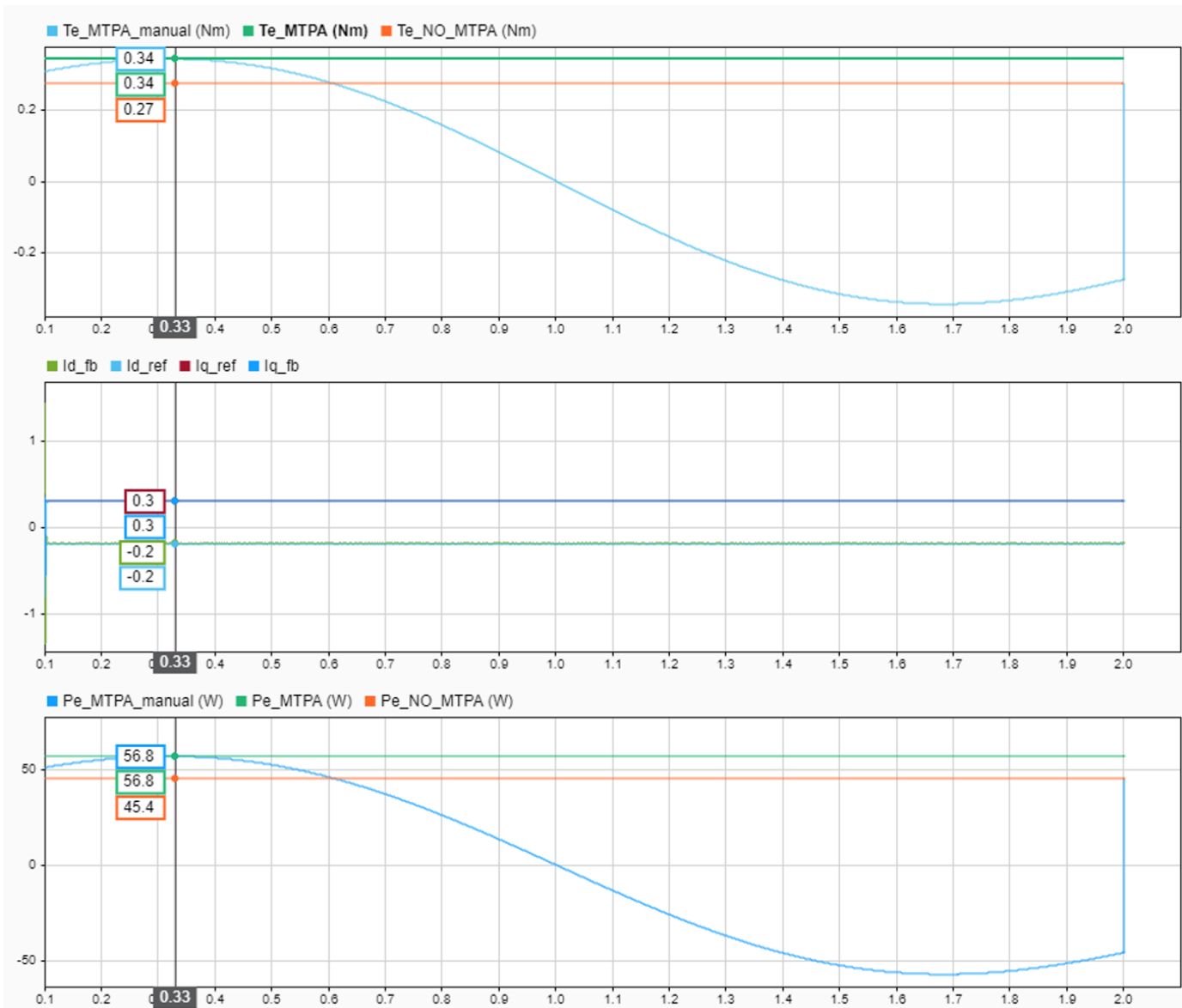
In the preceding example, you can notice that the electrical torque generated using MTPA is 0.34PU whereas electrical torque generated without MTPA is 0.27PU. You can also notice that with a varying torque angle, the maximum generated torque matches the torque produced by MTPA. The negative d-axis current indicates that the MTPA utilizes the reluctance torque for interior PMSM.

NOTE: If you are working with Surface PMSM, change the Type of motor parameter from Interior PMSM to Surface PMSM, in the MTPA Control Reference block located at the location: "Torque Control\MTPA_Reference\MTPA Control Reference."

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host

model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: **mcb_pmsm_fwc_qep_f28069LaunchPad** and **mcb_pmsm_mtpa_qep_f28069LaunchPad**

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_pmsm_fwc_qep_f28379d** and **mcb_pmsm_mtpa_qep_f28379d**

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Run Models to implement speed and torque control with field-weakening and MTPA

1. Simulate the model and analyze the simulation results by using the preceding section.

2. Complete the hardware connections.

3. The torque control model requires an Interior PMSM with QEP Sensor, driven by an external dynamometer with speed control (that uses the speed control model).

4. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value zero to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

5. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

6. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see "Model Configuration Parameters" on page 2-2.

7. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

8. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

9. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for speed control implementation:

```
open_system('mcb_pmsm_fwc_host_model.slx');
```

PMSM Field Weakening Control Host

HOST
Serial
Setup

Note:
1. Update workspace with variables used in target model
2. Select the serial port in 'Host Serial Setup' (Blue Color)
3. Use 'Motor Start / Stop' switch to control motor.
4. Input speed request using 'Reference Speed' block.
5. Observe the debug signals in scope.

Off

Off

On

On

Start / Stop
Field Weakening Control

Start / Stop Motor

Debug signals

⊙ Speed_ref & Speed_feedback

◯ Id_ref & Id_feedback

◯ Iq_ref & Iq_feedback

◯ Torque & Power

◯ Ia & Ib

2000

Reference Speed (RPM)

Speed_ref (rpm)

Data_Conditioning_Tx

Scope (Per-Unit)

Debug1 (SI units)

Debug2 (SI units)

Data_Conditioning_Rx

SelectedSignals

Copyright 2020 The MathWorks, Inc.

4-51
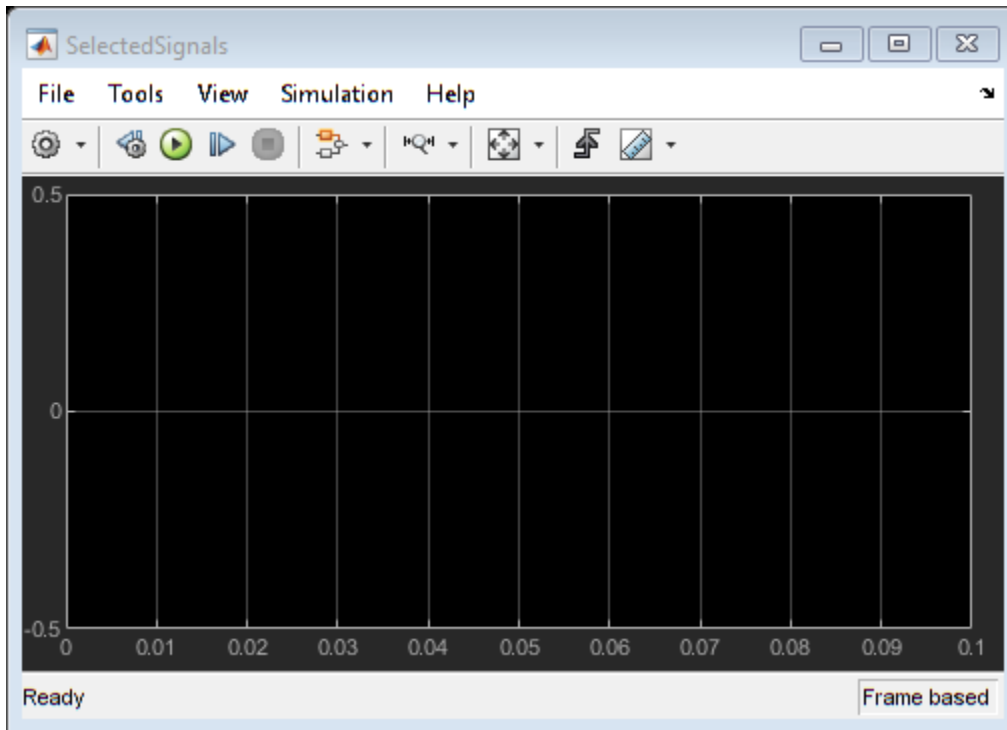
For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

10. In the Host Serial Setup block mask of the host model, select a **Port name**.

11. In the Speed control model, update the Reference Speed (RPM) block value. In the Torque control model, update the current request using Imag Reference block.

12. Click **Run** on the **Simulation** tab to run the host model.

13. Change the position of the Start / Stop Motor switch to On, to start and stop running the motor.

14. Enter different reference speeds (or currents) and observe the debug signals from the RX subsystem, in the Time Scope of host model.

Note

- If the position offset is incorrect, this example can lead to excessive currents in the motor. To avoid this, ensure that the position offset is correctly computed and updated in the workspace variable: pmsm.PositionOffset.
- When you operate the motor above the base speed, we recommend that you monitor the temperature of motor. During motor operation, if the motor temperature rises beyond the temperature recommended by the manufacturer, turn-off the motor for safety reasons.
- When you operate the motor above the base speed, we recommend that you increment the speed reference in small steps, to avoid the dynamics of field weakening that can make some systems unstable.

References

[1] B. Bose, Modern Power Electronics and AC Drives. Prentice Hall, 2001. ISBN-0-13-016743-6.

[2] Lorenz, Robert D., Thomas Lipo, and Donald W. Novotny. "Motion control with induction motors." Proceedings of the IEEE, Vol. 82, Issue 8, August 1994, pp. 1215-1240.

[3] Morimoto, Shigeo, Masayuka Sanada, and Yoji Takeda. "Wide-speed operation of interior permanent magnet synchronous motors with high-performance current regulator." IEEE Transactions on Industry Applications, Vol. 30, Issue 4, July/August 1994, pp. 920-926.

[4] Li, Muyang. "Flux-Weakening Control for Permanent-Magnet Synchronous Motors Based on Z-Source Inverters." Master's Thesis, Marquette University, e-Publications@Marquette, Fall 2014.

[5] Briz, Fernando, Michael W. Degner, and Robert D. Lorenz. "Analysis and design of current regulators using complex vectors." IEEE Transactions on Industry Applications, Vol. 36, Issue 3, May/June 2000, pp. 817-825.

[6] Briz, Fernando, et al. "Current and flux regulation in field-weakening operation [of induction motors]." IEEE Transactions on Industry Applications, Vol. 37, Issue 1, Jan/Feb 2001, pp. 42-50.

[7] TI Application Note, "Sensorless-FOC With Flux-Weakening and MTPA for IPMSM Motor Drives."

Sensorless Field-Oriented Control of PMSM

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

This example uses the sensorless position estimation technique. You can select either the sliding mode observer or flux observer to estimate the position feedback for the FOC algorithm used in the example.

The Sliding Mode Observer (SMO) block generates a sliding motion on the error between the measured and estimated position. The block produces an estimated value that is closely proportional to the measured position. The block uses stator voltages (V_α, V_β) and currents (I_α, I_β) as inputs and estimates the electromotive force (emf) of the motor model. It uses the emf to further estimate the rotor position and rotor speed. The Flux Observer block uses identical inputs $(V_\alpha, V_\beta, I_\alpha, I_\beta)$ to estimate the stator flux, generated torque, and the rotor position.

Models

The example includes these models:

- **mcb_pmsm_foc_sensorless_f28069MLaunchPad**

- **mcb_pmsm_foc_sensorless_f28379d**

You can use these models for both simulation and code generation. You can use the open_system command to open a model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_foc_sensorless_f28069MLaunchPad.slx');
```


For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

1. For the model: **mcb_pmsm_foc_sensorless_f28069MLaunchPad**

- Motor Control Blockset™
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_foc_sensorless_f28379d**

- Motor Control Blockset™

To generate code and deploy model:

1. For the model: **mcb_pmsm_foc_sensorless_f28069MLaunchPad**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_foc_sensorless_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions , see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. To simulate the model, click **Run** on the **Simulation** tab.

3. To view and analyze the simulation results, click **Data Inspector** on the **Simulation** tab.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter:
  mcb_pmsm_foc_sensorless_f28069MLaunchPad

- LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter:
  mcb_pmsm_foc_sensorless_f28379d

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the Analog-to-Digital Converter (ADC) or current offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

5. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

7. In the target model, click the **host model** hyperlink to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_foc_host_model_f28069m.slx');
```

PMSM FOC Host

Copyright 2020 The MathWorks, Inc.
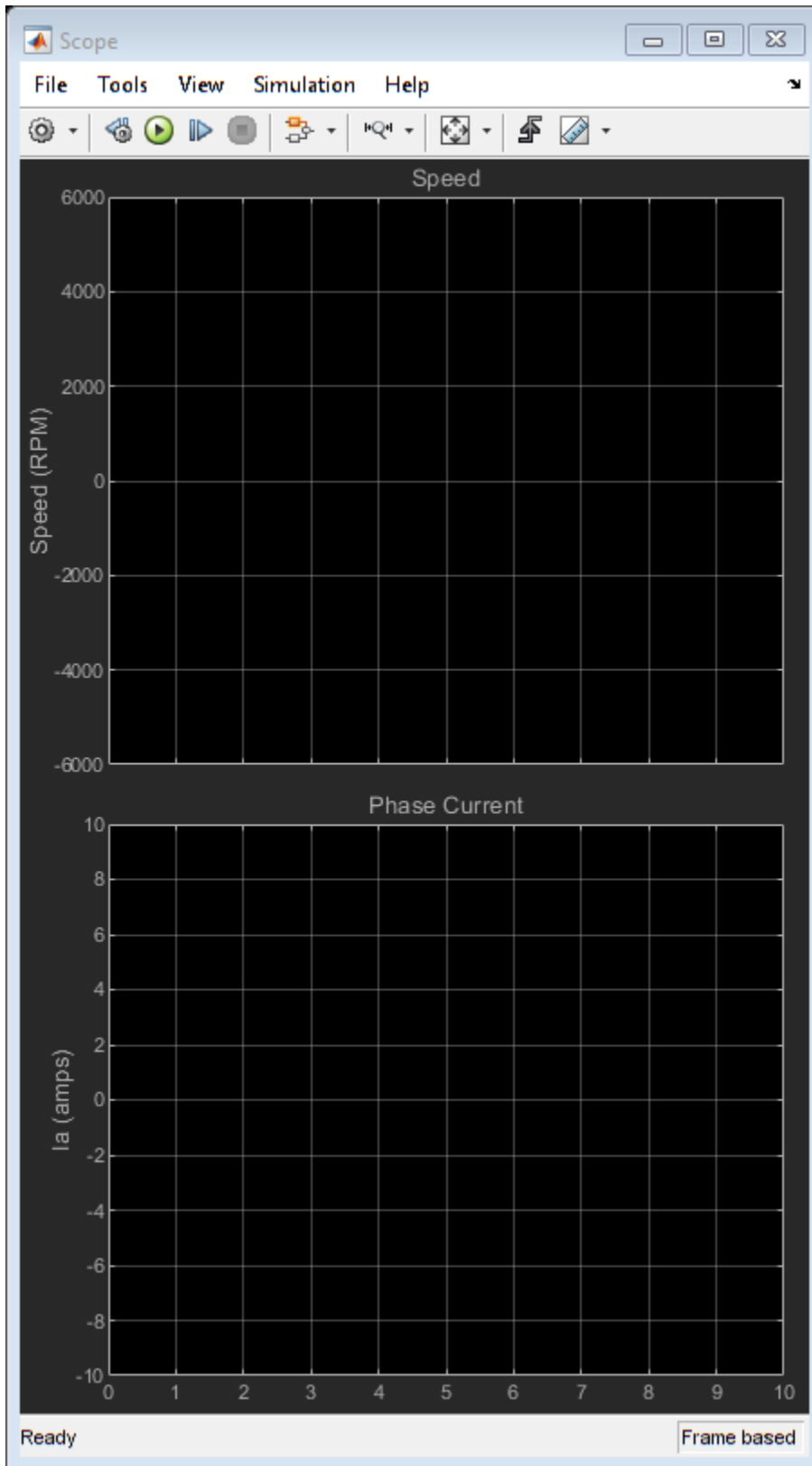
For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

8. In the Host Serial Setup block mask of the host model, select a **Port name**.

9. Update the Reference Speed value in the host model.

10. Click **Run** on the **Simulation** tab to run the host model.

11. Change the position of the Start / Stop Motor switch to On, to start running the motor in the open-loop condition (by default, the motor spins at 10% of base speed).

Note: Do not run the motor (using this example) in the open-loop condition for a long time duration. The motor may draw high currents and produce excessive heat.

We designed the open-loop control to run the motor with a Reference Speed that is less than or equal to 10% of base speed.

12. Increase the motor Reference Speed beyond 10% of base speed to switch from open-loop to closed-loop control.

NOTE: To change the motor's direction of rotation, reduce the motor Reference Speed to a value less than 10% of the base speed. This brings the motor back to open-loop condition. Change the direction of rotation but keep the Reference Speed magnitude as constant. Then transition to the closed-loop condition.

13. Observe the debug signals from the RX subsystem, in the Time Scope of host model.

Note: If you are using a F28379D based controller, you can also select the debug signals that you want to monitor.

Other Things to Try

You can use SoC Blockset™ to implement a sensorless closed-loop motor control application that addresses challenges related to ADC-PWM synchronization, controller response, and studying different PWM settings. For details, see "Integrate MCU Scheduling and Peripherals in Motor Control Application" (SoC Blockset).

You can also use SoC Blockset™ to develop a sensorless real-time motor control application that utilizes multiple processor cores to obtain design modularity, improved controller performance, and other design goals. For details, see "Partition Motor Control for Multiprocessor MCUs" (SoC Blockset).

Use Motor Control Blockset to Generate Code for Custom Target

This example shows how to use Motor Control Blockset™ with any processor.

The example shows you how to simulate and generate code from a system model configured for a Texas Instruments™ C2000™ F28069M processor. The system model uses a Field-Oriented Control (FOC) implementation that you can run on any processor. The algorithm part of the model is separated from the driver layer by using a reference model that you can deploy on any device.

Required Products

- MATLAB®
- Simulink®
- MATLAB® Coder™
- Simulink® Coder™
- Motor Control Blockset™
- Embedded Coder®
- Fixed-Point Designer™ (only for serial communication)

Verify Algorithm Behavior by Using System Simulation

This section shows you how to verify the controller in a closed-loop system simulation.

The system model mcb_pmsm_foc_system test bench consists of the test inputs, an embedded processor, power electronics, and motor hardware. To see the signals, use the **Data Inspector** button on the **Simulation** tab of the Simulink toolstrip. You can use this model to test the controller and explore its expected behavior.

Use this command to open the model.

```
open_system('mcb_pmsm_foc_system.slx');
```

Field-Oriented Control for PMSM with QEP sensor

Note: This example is configured for TI F28069m LaunchPad with a BOOSTXL-DRV8305 booster pack connected to a PMSM Motor with QEP Sensor.

Processor

Embedded Processor

M

Inverter and Motor - Plant Model
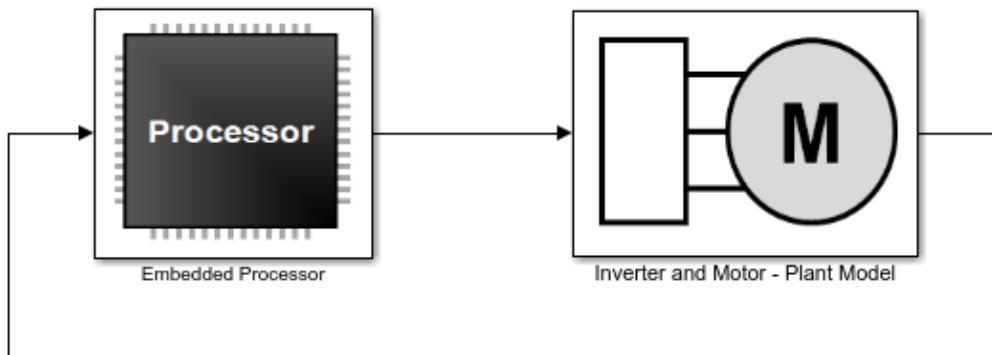
Copyright 2020 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Simulate this model
3. Calibrate QEP offset
4. Update motor parameters with QEP offset
5. Build, Deploy & Start
6. Control motor via host model

NOTE: This model supports only floating-point computations.

Run the simulation and see the logged speed reference (Speed_Reference) and measured motor speed (Speed_Motor) signals in the data inspector.
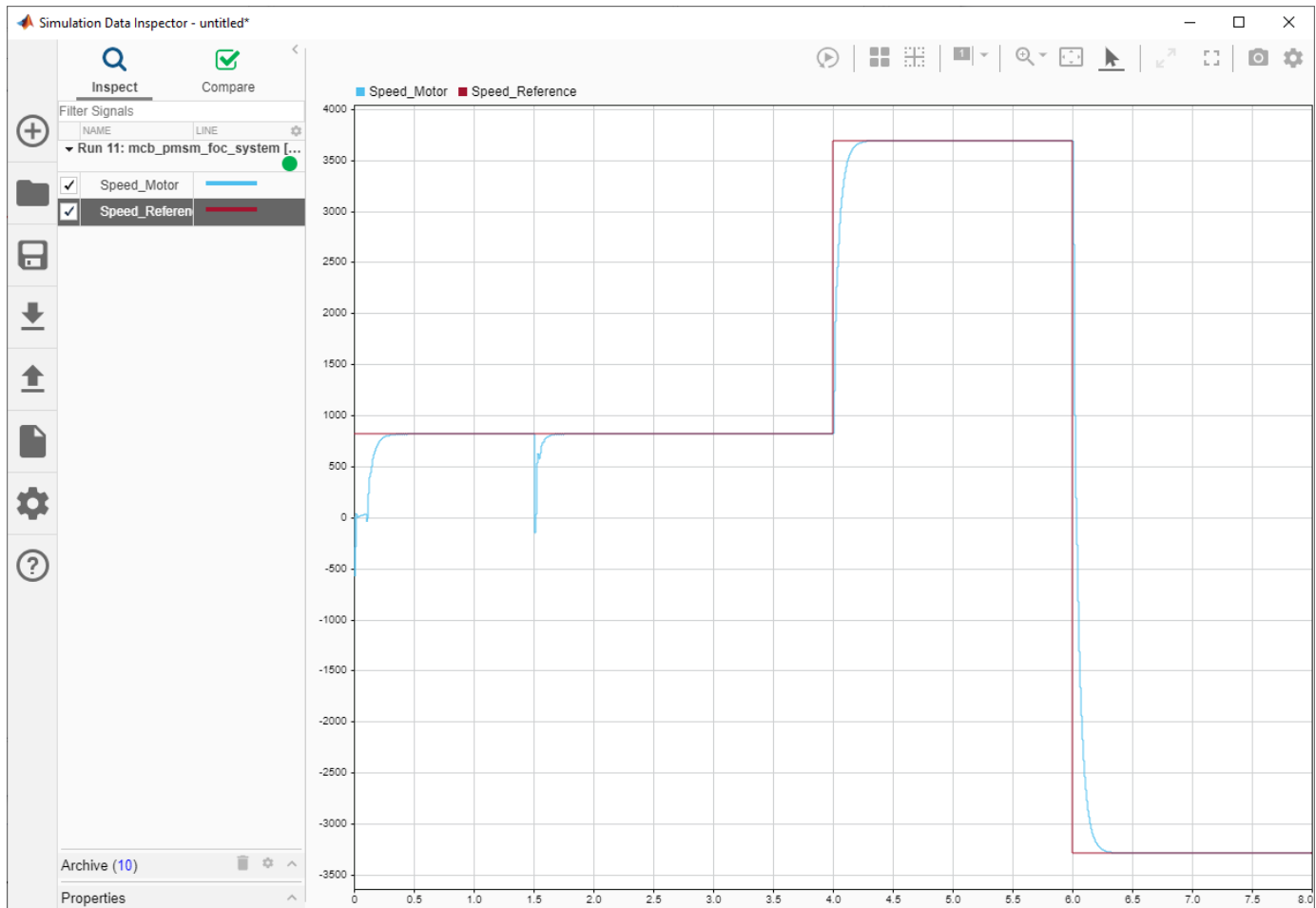
Model Architecture

This section explains the model architecture and includes these sub-sections:

- Data Specification

- Controller Partitioning from Test Bench

- Controller Scheduling

The model architecture facilitates system simulation and algorithmic code generation.

Data Specification

A data definition file creates the data needed for simulation and code generation. This data file is automatically run within the *InitFcn* callback of the system test bench model.

```
edit('mcb_algorithm_workflow_data.m')
```

Another data file mcb_pmsm_foc_qep_f28069LaunchPad_data.m defines the motor and inverter parameters.

Update the motor and inverter parameters for your hardware configuration in this file. For example, update the motor parameters in the function *mcb_SetPMSMMotorParameters* that is called from this file.

Controller Partitioning from Test Bench

Within the system test bench model, the embedded processor is modeled as a combination of the peripherals and the controller software. The block mcb_pmsm_foc_system/Embedded Processor/ Serial Receive implements the reference inputs for simulation.

open_system('mcb_pmsm_foc_system/Embedded Processor');
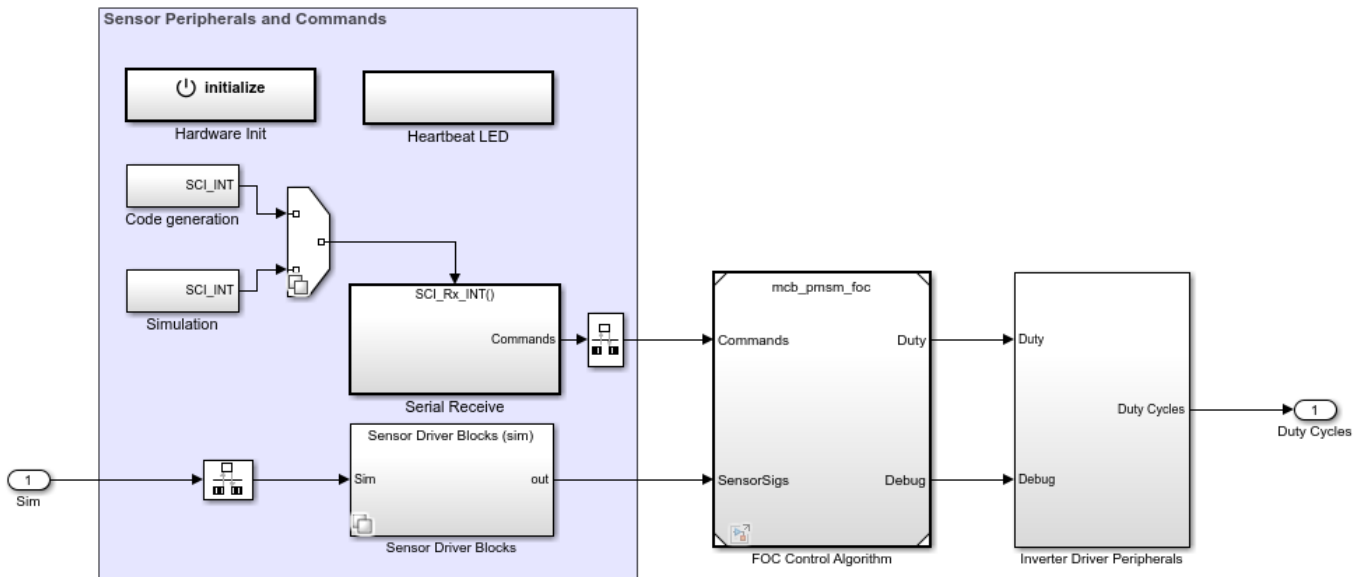
In this example, a separate model includes the controller software. The controller software model contains the Speed Control and Current Control subsystems of the FOC algorithm.

open_system('mcb_pmsm_foc.slx');

Field-Oriented Control for PMSM
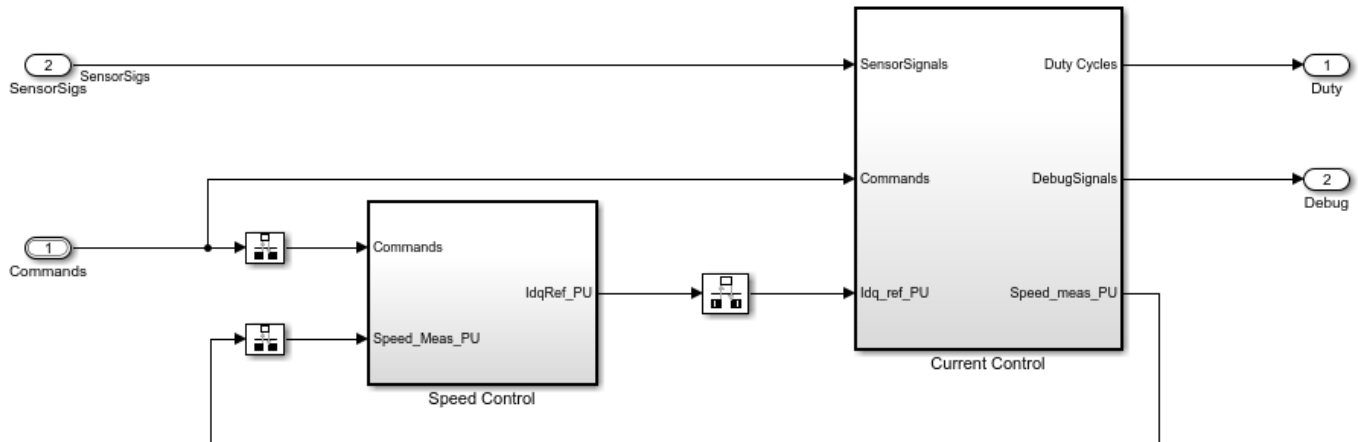
Copyright 2020 The MathWorks, Inc.

Controller Scheduling

The primary control method is field-oriented control. The controller has a low rate outer loop that controls the speed. It also has a higher rate inner loop that controls the current. Speed Control subsystem implements the PI controller for speed. The Current Control subsystem converts the ADC signals (or the current feedback) to per-unit values and passes them to the core controller algorithm. In addition, it also measures the speed and position values from the quardature encoder pulses.

The controller algorithm calculates the voltages. The voltages are then converted to a driver signal. The speed controller outer loop executes after each instance of the time period used to run the current control loop. You can view the variables that specify the speed and current control loop sample times by using these commands:

```
fprintf('Current loop sample time = %f seconds\n', Ts)
```

```
fprintf('Speed loop sample time  = %f seconds\n', Ts_speed)
```

Generate C Code to Integrate Controller into Embedded Application

This section shows you how to generate and visually inspect the C code function for the controller.

The generated code consists of three generated global functions:

- *void Controller_Init(void)*: This function should be called to perform initialization routines.

- *void Current_Controller(void)*: This function implements the current controller and should be called from a task running at 50e-6 seconds.

- *void Speed_Controller(void)*: This function implements the speed controller and should be called from a task running at 500e-6 seconds.

You can specify the function prototype in the Configure C Step Function Interface dialog box. For more details, see Override Default C Step Function Interface.

Inputs to FOC Control Algorithm:

- ExternalInputs_mcb_pmsm_foc: This is a structure with the speed reference and signal to enable the motor.

- SensorSigs: This is an array with I_a ADC counts, I_b ADC counts, quadrature encoder position counts, and quadrature encoder index latch.

Outputs of FOC Control Algorithm:

- PWM Duty: This is an array with the PWM Duty Cycles for three phases and the signal to enable PWM.

- DebugSignals: This is an array of signals that you can log while executing the control algorithm.

Parameters for FOC Control Algorithm:

- PI_params: This is a structure that contains the PI gains Kp_i, Ki_i, Kp_speed, and Ki_speed.

- IsOffset, IbOffset: These are datastore variables that contain the ADC calibration offsets.

Hardware Peripheral Integration

- Hardware peripherals are integrated with the control algorithm inside the mcb_pmsm_foc_system/ Embedded Processor subsystem.

- The ADC interrupt is used to schedule the generated code. The interrupt triggers at 50e-6 seconds.

- The subsystem mcb_pmsm_foc_system/Embedded Processor/Hardware Init finds the ADC calibration offsets and provides them to the control algorithm.

- The subsystem mcb_pmsm_foc_system/Embedded Processor/Sensor Driver Blocks implements the ADC and Quadrature Encoder peripherals.

- The subsystem mcb_pmsm_foc_system/Embedded Processor/Serial Receive has the serial blocks to receive user inputs from a host model when the generated code is executing on the target.

- The subsystem mcb_pmsm_foc_system/Embedded Processor/Inverter Driver Peripherals has the PWM driver peripherals and the Serial Transmit block to send data to the host computer. All these peripherals are used from the Texas Instruments™ C2000™ Support Package.

If you are using a custom processor, you can implement the driver logic using a custom code. You can integrate the generated code for the control algorithm with your own driver code in your preferred Integrated Development Environment (IDE).

Test Behavior of Generated Code

For details of the required hardware connections, see "Hardware Connections" on page 7-2.

- Find the Quadrature Encoder offset. For details, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

- Build and load the executable file to the target for the mcb_pmsm_foc_system model.

- Open the host model mcb_pmsm_foc_host_model_f28069m using the host model link available in the mcb_pmsm_foc_system model.

- Update the COM port name for the target in the Host Serial Setup block of the host model.

- Click **Run** in the **Simulation** tab to run the host model.

- Change the Motor Start / Stop switch position to On, to start running the motor.

- Change the Reference Speed and monitor the effects in the scope window.

Field Oriented Control of PMSM by Using SI Units

This example implements the Field-Oriented Control (FOC) technique to control the speed of a three-phase Permanent Magnet Synchronous Motor (PMSM). However, instead of the per-unit representation of quantities(for details about the per-unit system, see "Per-Unit System" on page 6-15), the FOC algorithm in this example uses the SI units of signals to perform the computations. These are the signals and their SI units:

- Rotor speed - Radians/ sec

- Rotor position - Radians

- Currents - Amperes

- Voltages - Volts

Field-oriented control (FOC) needs a real time feedback of the rotor position. This example uses the quadrature encoder sensor to measure the rotor position. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

Models

The example includes this model:

- **mcb_pmsm_foc_qep_f28379d_SIUnit**

You can use this model for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28379D based controller:

```
open_system('mcb_pmsm_foc_qep_f28379d_SIUnit.slx');
```


Permanent Magnet Synchronous Motor Field Oriented Control in SI units

Required MathWorks® Products

To simulate model:

• Motor Control Blockset™

To generate code and deploy model:

• Motor Control Blockset™
• Embedded Coder®
• Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
• Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open the model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target model and run the motor in a closed-loop control.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_pmsm_foc_qep_f28379d_SIUnit**

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

5. Open the target model. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

6. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

8. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_SIUnit_host_model.slx');
```

FOC Host for SI Unit Example
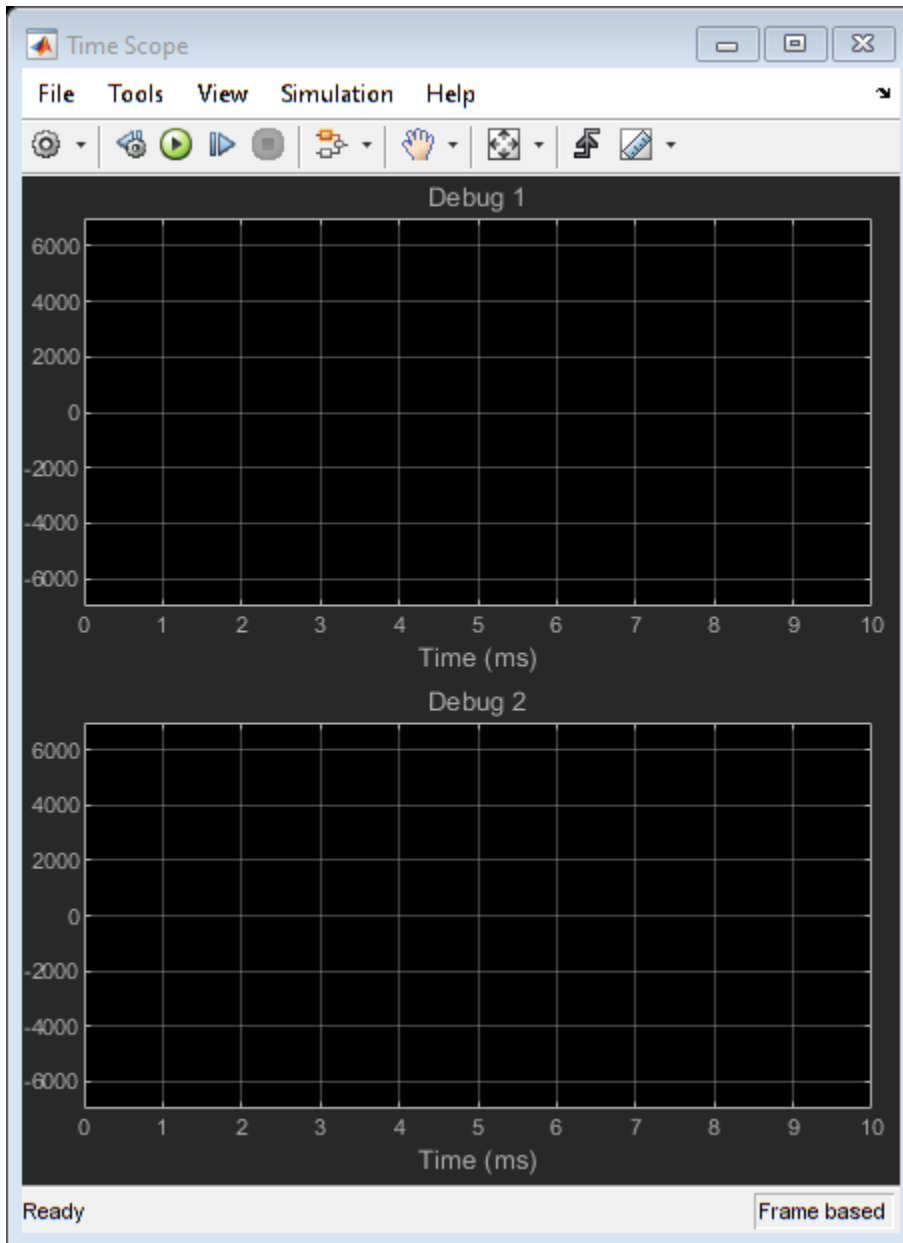
HOST
Serial
Setup

Host Serial Setup

Note:
1. Select the serial port in 'Host Serial Setup' (Blue Color)
2. Use 'Motor Start / Stop' switch to enable and disable motor control.
3. Input speed request using 'Reference Speed' text box or sliding bar.
4. Observe the actual speed of motor and 'Phase A current' in the scope.

Debug signals

⦿ Speed Control
◯ Id Control
◯ Iq Control
◯ Ia & Ib
◯ Ia & Position

Off

0

Reference Speed

TX

On

Motor Start / Stop

Debug 1

Debug 2

RX

-6000 -4500 -3000 -1500 0 1500 3000 4500 6000

Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

9. In the Host Serial Setup block mask of the host model, select a **Port name**.

10. Update the Reference Speed value in the host model.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On, to start running the motor.

13. Observe the debug signals from the RX subsystem, in the Time Scope of host model.

Hall Offset Calibration for PMSM Motor

This example calculates the offset between the rotor direct axis (d-axis) and position detected by the Hall sensor. The field-oriented control (FOC) algorithm needs this position offset to run the permanent magnet synchronous motor (PMSM) correctly. To compute the offset, the target model runs the motor in the open-loop condition. The model uses a constant V_d (voltage along the stator's d-axis) and a zero V_q (voltage along the stator's q-axis) to run the motor (at a low constant speed) by using a position or ramp generator. When the position or ramp value reaches zero, the corresponding rotor position is the offset value for the Hall sensors.

The control algorithm (available in the field-oriented control and parameter estimation examples) uses this offset value to compute an accurate position of d-axis of the rotor. The controller needs this offset to optimally run the PMSM.

Models

This example includes these models:

- **mcb_pmsm_hall_offset_f28069m**

- **mcb_pmsm_hall_offset_f28379d**

You can use these models only for code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_hall_offset_f28069m.slx');
```
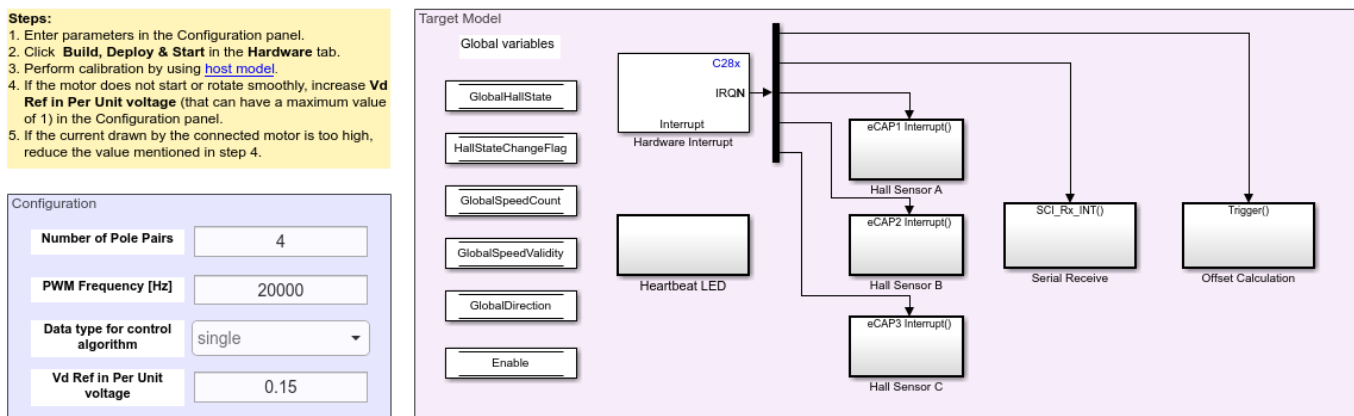

Copyright 2020 The MathWorks, Inc.

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To generate code and deploy model:

1. For the model: **mcb_pmsm_hall_offset_f28069m**

• Motor Control Blockset™

• Embedded Coder®

• Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors

• Fixed-Point Designer™

2. For the model: **mcb_pmsm_hall_offset_f28379d**

• Motor Control Blockset™

• Embedded Coder®

• Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors

• Fixed-Point Designer™ (only needed for optimized code generation)

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the motor by using open-loop control.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board.

The host model uses serial communication to command the target model and run the motor in an open-loop configuration. You can use the host model to control the motor rotations and validate direction of rotation of the motor. The **Incorrect motor direction** LED in the host model turns red to indicate that the motor is running in the opposite direction. When the LED turns red, you must reverse the motor phase connections to change the direction of rotation. The host model displays the calculated offset value.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

• F28069M controller card + DRV8312-69M-KIT inverter: **mcb_pmsm_hall_offset_f28069m**

For connections related to the preceding hardware configuration, see "F28069 control card configuration" on page 7-2.

• LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter: **mcb_pmsm_hall_offset_f28379d**

To configure the model **mcb_pmsm_hall_offset_f28379d**, set the **Inverter Enable Logic** field (in the **Configuration** panel of target model) to:

• **Active High**: To use the model with BOOSTXL-DRV8305 inverter.

• **Active Low**: To use the model with BOOSTXL-3PHGANINV inverter.

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

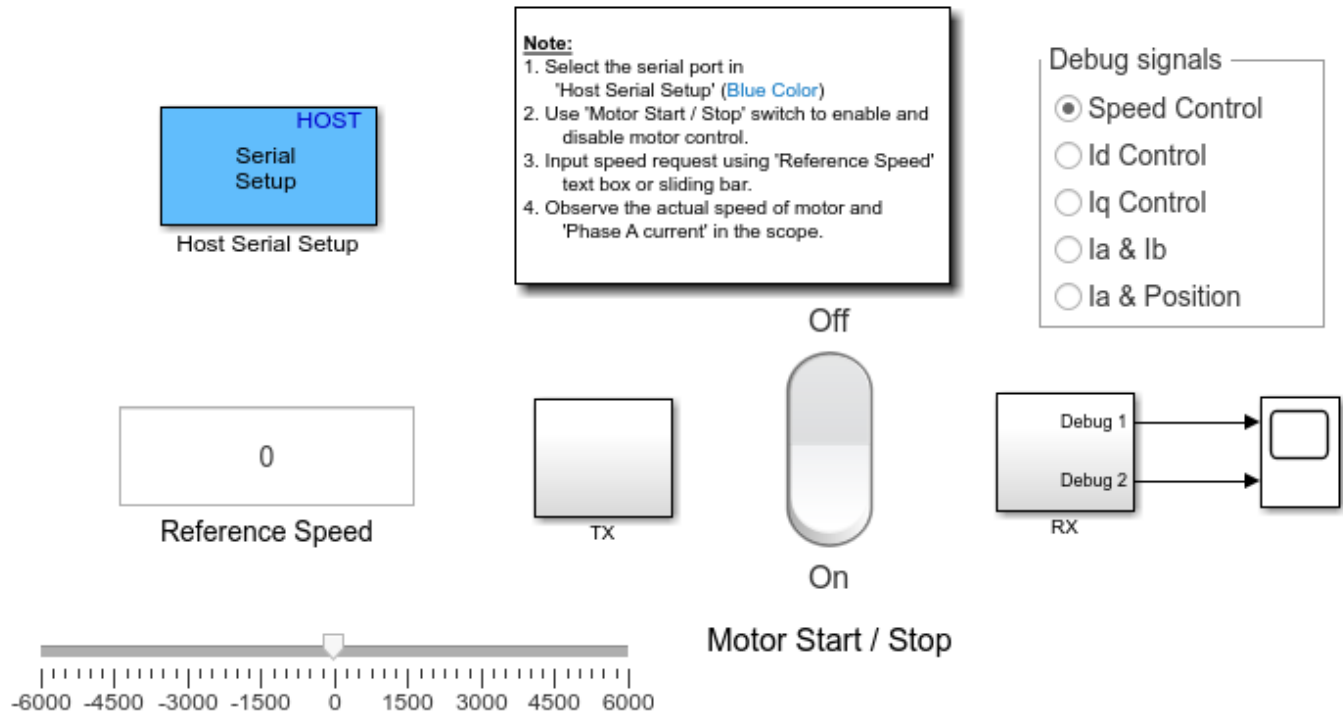**Generate Code and Run Model on Target Hardware**

1. Complete the hardware connections.

2. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see "Model Configuration Parameters" on page 2-2.

3. Update these motor parameters in the **Configuration** panel of the target model.

- **Number of Pole Pairs**
- **PWM Frequency [Hz]**
- **Data type for control algorithm**
- **Vd Ref in Per Unit voltage**

4. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

5. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

6. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_host_offsetComputation_f28069m.slx');
```


Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

You can use the Time Scope in the host model to monitor the rotor position and offset values.

7. In the Host Serial Setup block mask of the host model, select a **Port name**.

8. Click **Run** on the **Simulation** tab to run the host model. The motor runs and calibration begins when you start simulation. After calibration completes, simulation ends and the motor stops automatically.

9. See the **Calibration Status** section to know the status of calibration process:

- **Calibration in progress** LED turns orange when the motor starts running. Notice the rotor position and offset value variation in the Scope (the position signal indicates a ramp signal with an amplitude between 0 and 1). After calibration completes this LED turns grey.

- **Calibration complete** LED turns green when calibration completes. Then the Calibration Output field displays the computed offset value.

- **Incorrect motor direction** LED turns red if the motor runs in the opposite direction. Then the Calibration Output field displays the value "NaN." Turn off the DC power supply (24V) and reverse the motor phase connections from ABC to CBA. Repeat steps 5 to 8 and check if the Calibration complete LED is green. Verify that the Calibration Output field displays the offset value.

Note: This example does not support simulation.

During emergency, cick the **Emergency Motor Stop** button to stop the motor immediately.

For examples that implement FOC using a Hall sensor, update the computed offset in the pmsm.PositionOffset parameter in the model initialization script linked to the example. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Monitor Resolver Using Serial Communication

This example operates the resolver sensor to measure the rotor position. The resolver consists of two orthogonally placed stator windings placed around the resolver rotor winding. After you mount the resolver sensor over a PMSM, the resolver rotor winding rotates along with the shaft of the running motor. The controller provides a fixed frequency alternating excitation signal to the resolver rotor winding. When the resolver rotor rotates, the resolver stator windings produce output (secondary sine and cosine) signals that are modulated with the sine and cosine of the shaft angle or position. After receiving the secondary signals, the controller samples and normalizes them.
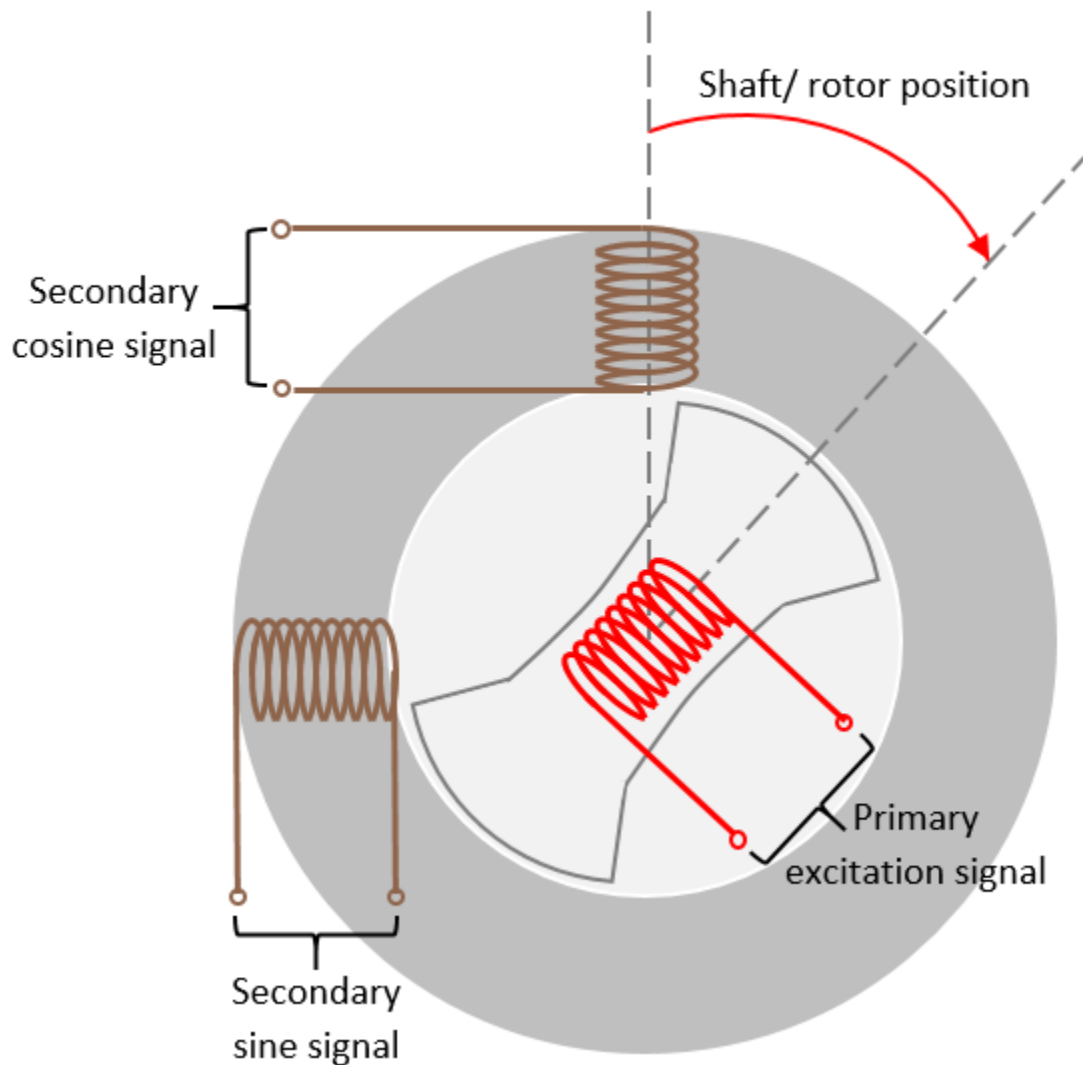
Primary Excitation Signal

Secondary sine signal

Sampled and normalized secondary sine signal

Secondary cosine signal

Sampled and normalized secondary cosine signal
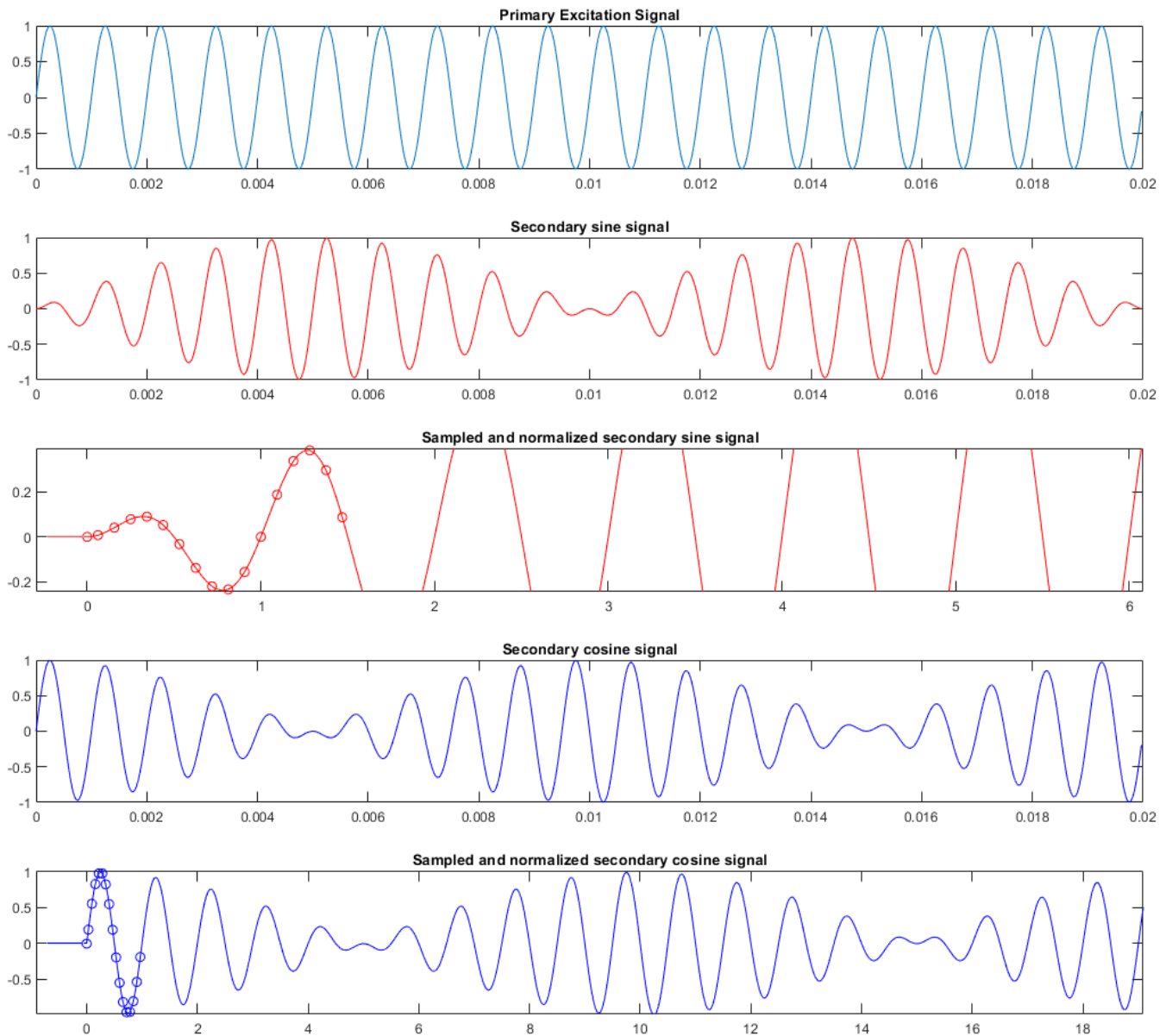
Models

The example includes this model:

* **mcb_resolver_f28069m**

You can use this models only for code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller:

```
open_system('mcb_resolver_f28069m.slx');
```
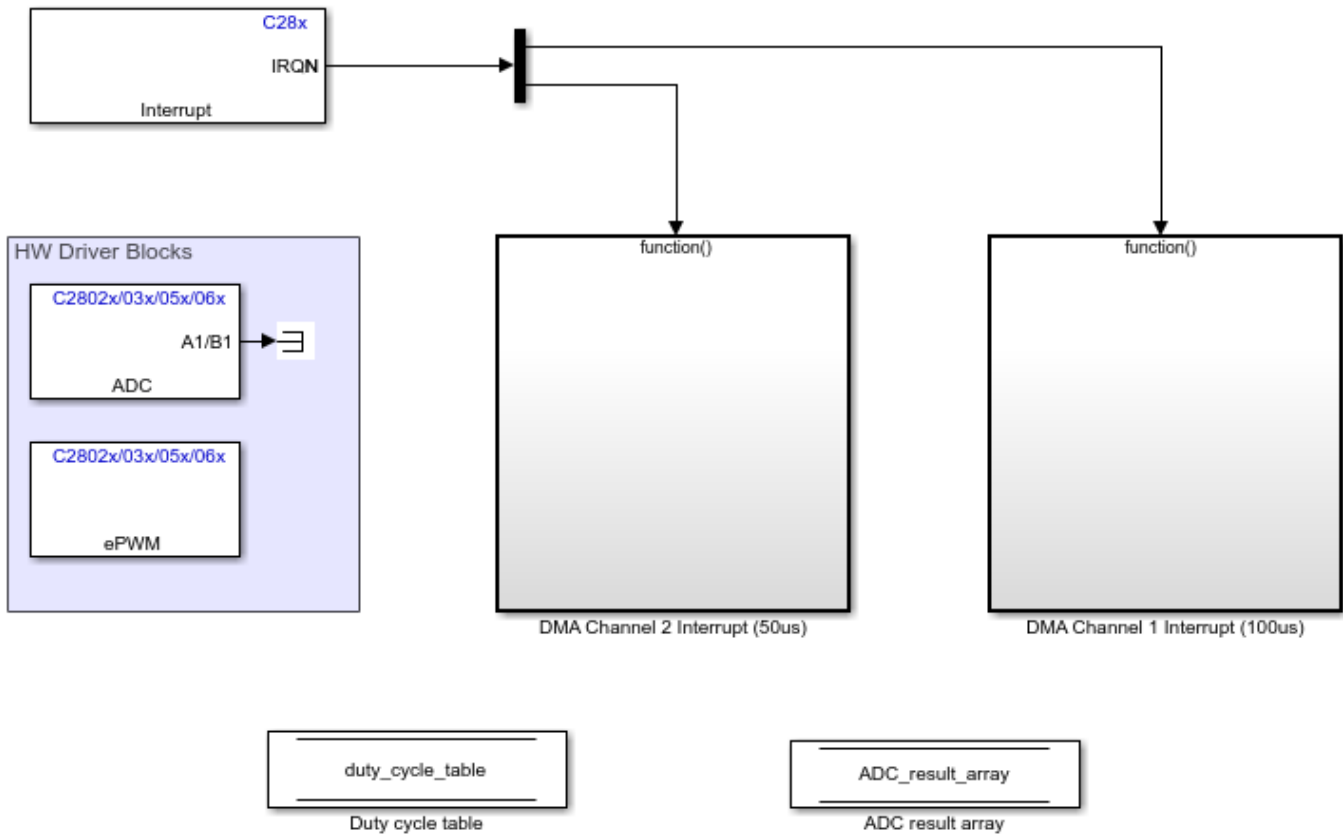
Required MathWorks® Products

For the model: **mcb_resolver_f28069m**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

Prerequisite

We provide default inverter parameters with the target model. If you want to change the default values, you can update the inverter parameters in the model initialization script associated with the Simulink® model. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The controller in the target model uses the Resolver Decoder block to process the sampled and normalized secondary sine and cosine signals to obtain the shaft (or motor) position. The host model uses serial communication to

command the target model and obtain the computed shaft angle from the controller. You can observe the computed shaft position in the Time Scope block of the host model.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

*   LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: **mcb_resolver_f28069m**

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections and open the target model **mcb_resolver_f28069m**.

2. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

3. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

4. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for the F28069M based controller:

```
open_system('mcb_resolver_host_read.slx');
```


Resolver Host
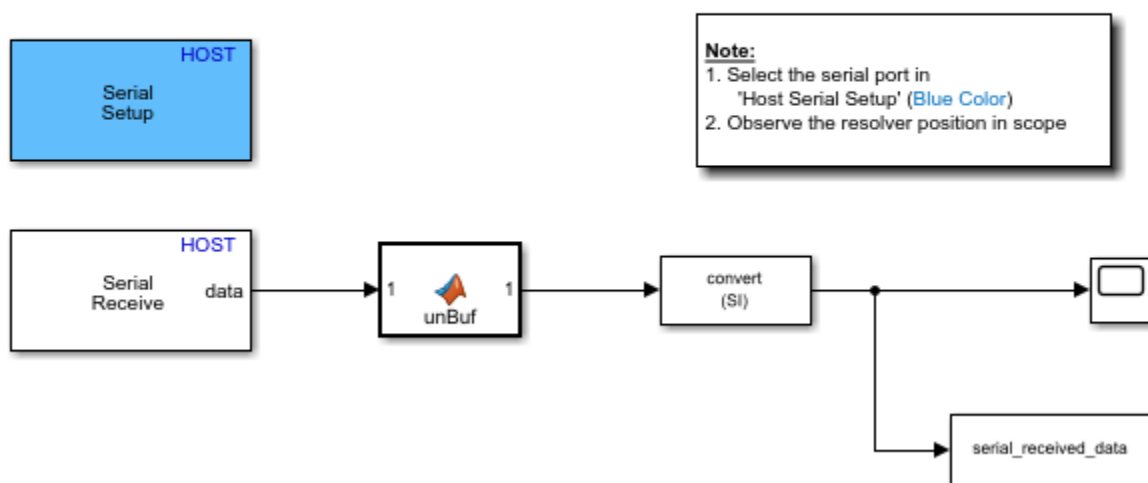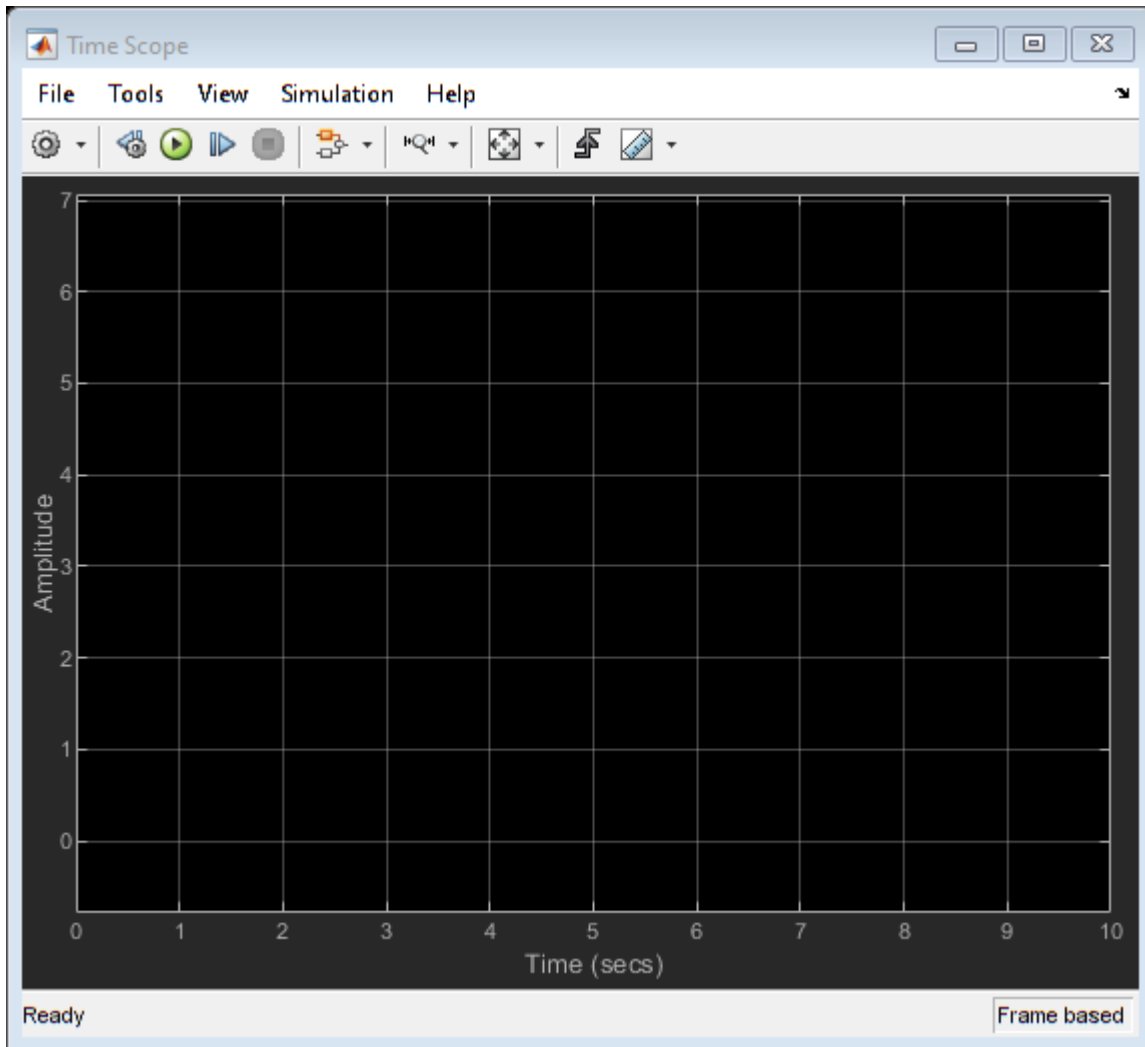
For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

5. In the Serial Receive and Serial Configuration block masks of the host model, select a Communication port value.

6. If you want to change the default baud rate (in the host and target models), use the Serial Configuration block mask in the models to select a different Baud rate value.

7. Click **Run** on the **Simulation** tab to run the host model.

8. Open the Time Scope block in the host model.

9. Rotate the resolver shaft and observe the computed shaft position signal in the Time Scope block.

Quadrature Encoder Offset Calibration for PMSM Motor

This example calculates the offset between the d-axis of the rotor and encoder index pulse position as detected by the quadrature encoder sensor. The control algorithm (available in the field-oriented control and parameter estimation examples) uses this offset value to compute an accurate and precise position of the d-axis of rotor. The controller needs this position to implement the field-oriented control (FOC) correctly in the rotor flux reference frame (d-q reference frame), and therefore, run the permanent magnet synchronous motor (PMSM) correctly.

Models

The example includes these models:

- **mcb_pmsm_qep_offset_f28069m**

- **mcb_pmsm_qep_offset_f28069mLaunchPad**

- **mcb_pmsm_qep_offset_f28379d**

You can use these models only for code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_qep_offset_f28069m.slx');
```


Copyright 2020 The MathWorks, Inc.

For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To generate code and deploy model:

1. For the models: **mcb_pmsm_qep_offset_f28069m** and **mcb_pmsm_qep_offset_f28069mLaunchPad**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_qep_offset_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the motor by using open-loop control.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board.

The host model uses serial communication to command the target model and run the motor in an open-loop configuration. You can use the host model to control the motor rotations and validate the direction of rotation of motor. The **Incorrect motor direction** LED in the host model turns red to indicate that the motor is running in the opposite direction. When the LED turns red, you must reverse the motor phase connections (from ABC to CBA) to change the direction of rotation. The host model displays the calculated offset value.

Required Hardware

This example supports these hardware configurations. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- F28069M controller card + DRV8312-69M-KIT inverter: **mcb_pmsm_qep_offset_f28069m**

For connections related to the preceding hardware configuration, see "F28069 control card configuration" on page 7-2.

- LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: **mcb_pmsm_qep_offset_f28069mLaunchPad**

- LAUNCHXL-F28379D controller + (BOOSTXL-3PHGANINV or BOOSTXL-DRV8305) inverter: **mcb_pmsm_qep_offset_f28379d**

To configure the model **mcb_pmsm_qep_offset_f28379d**, set the **Inverter Enable Logic** field (in the **Configuration** panel of target model) to:

- **Active High**: To use the model with BOOSTXL-DRV8305 inverter.
- **Active Low**: To use the model with BOOSTXL-3PHGANINV inverter.

4-83

NOTE: When using BOOSTXL-3PHGANINV inverter, ensure that proper insulation is available between bottom layer of BOOSTXL-3PHGANINV and the LAUNCHXL board.

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see "Model Configuration Parameters" on page 2-2.

3. Update these motor parameters in the **Configuration** panel of the target model.

- **Number of Pole Pairs**
- **QEP Slits**
- **PWM Frequency [Hz]**
- **Data type for control algorithm**
- **Vd Ref in Per Unit voltage**

3. Load a sample program to CPU2 of LAUNCHXL-F28379D, for example, program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

4. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

5. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_host_offsetComputation_f28069m.slx');
```
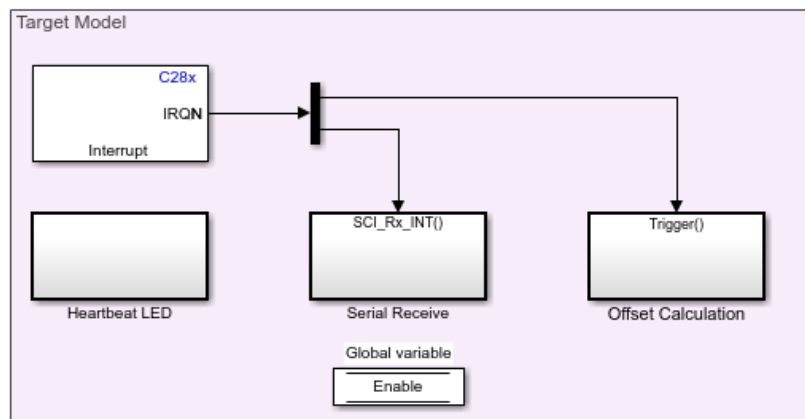
PMSM Position Sensor (Hall / QEP) Offset Calibration Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_hall_offset_f28069m
   mcb_pmsm_qep_offset_f28069m
   mcb_pmsm_qep_offset_f28069mLaunchPad
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select the port name in **Serial 1** tab of **Host Serial Setup** block.
2. Simulate this model to start calibration. Motor starts running when calibration begins.
3. After calibration completes, simulation ends and motor stops automatically.
4. Push the **Emergency Motor Stop** button to stop the motor during emergency.

Calibration Output

--

Position Sensor Offset
[Per-unit position]

Communication Port

HOST
Serial
Setup

Host Serial Setup

Position_PU

Serial Communication

Calibration Status

Calibration in progress

Calibration complete

Incorrect motor direction

Note:
If **Incorrect motor direction** LED glows red, turn off the power supply to the hardware setup, interchange any two motor phase connections, and simulate the host model again.

Emergency Motor Stop

Push for emergency stop

Position

Scope

Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

You can use the Time Scope in the host model to monitor the rotor position and offset values.

6. In the Host Serial Setup block mask of the host model, select a **Port name**.

7. Click **Run** on the **Simulation** tab to run the host model. The motor runs and calibration begins when you start simulation. After calibration completes, simulation ends and the motor stops automatically.

9. See the **Calibration Status** section to know the status of calibration process:

- **Calibration in progress** LED turns orange when the motor starts running. Notice the rotor position and offset value variations in the Time Scope (the position signal indicates a ramp signal with an amplitude between 0 and 1). After calibration completes this LED turns grey.

- **Calibration complete** LED turns green when calibration completes. Then the **Calibration Output** field displays the computed offset value.

- **Incorrect motor direction** LED turns red if the motor runs in the opposite direction. Then the Calibration Output field displays the value "NaN." Turn off the DC power supply (24V) and reverse the motor phase connections from ABC to CBA. Repeat steps 5 to 8 and check if the Calibration complete LED is green. Verify that the Calibration Output field displays the offset value.

Note: This example does not support simulation.

During emergency, cick the **Emergency Motor Stop** button to stop the motor immediately.

For examples that implement FOC using a quadrature encoder sensor, you must update the computed quadrature encoder offset value in the pmsm.PositionOffset parameter in the model initialization script linked to the example. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Model Switching Dynamics in Inverter Using Simscape Electrical

This example uses field-oriented control (FOC) to control the speed of a three-phase permanent magnet synchronous motor (PMSM). It gives you the option to use these Simscape Electrical blocks as an alternative to the Average Value Inverter block in Motor Control Blockset™:

- Converter (Three-Phase)
- Ideal Semiconductor Switch

The example also gives you the option to use the PMSM block from Simscape™ Electrical™ as an alternative to the Surface Mount PMSM block from Motor Control Blockset™. These Simscape™ Electrical™ blocks enable you to generate high-fidelity simulations.

Field-oriented control (FOC) needs a real time feedback of the rotor position. This example uses the quadrature encoder sensor to measure the rotor position. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

You can use this example to simulate the target model by using different inverters and monitor the feedback current for each inverter. You can also generate the code and use the host model along with the target model.

Models

The example includes this model:

- **mcb_ee_pmsm_foc**

You can use this model for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28379D based controller:

```
open_system('mcb_ee_pmsm_foc.slx');
```
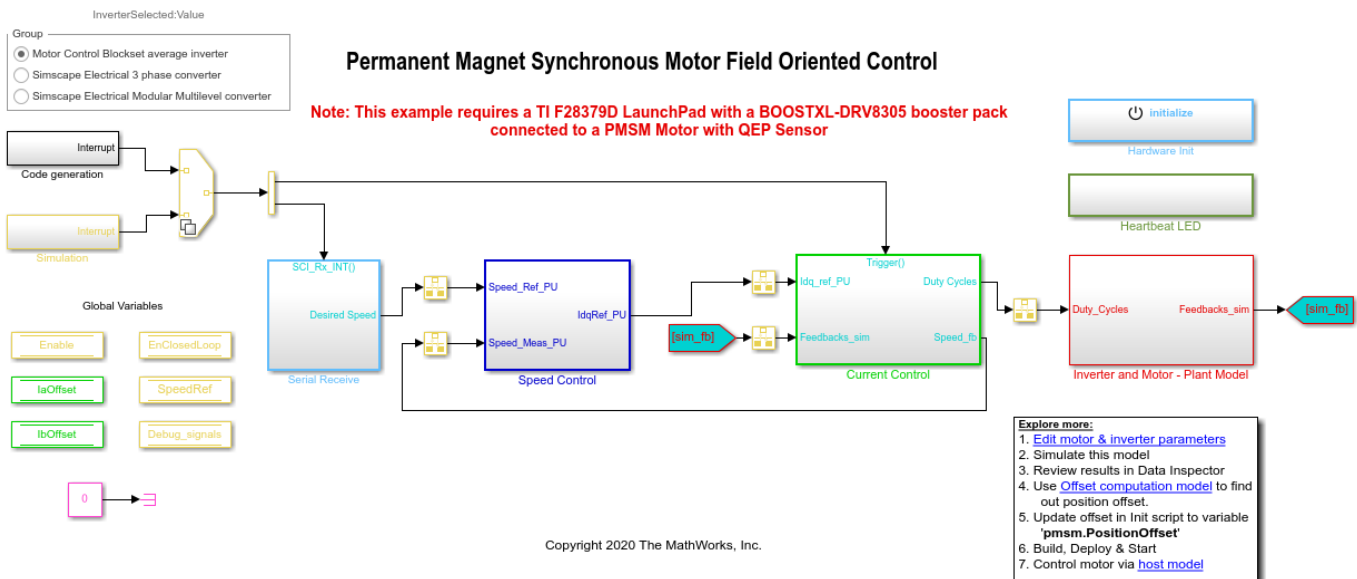
Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Simscape™ Electrical™

To generate code and deploy model:

- Motor Control Blockset™
- Simscape™ Electrical™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor parameters and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open the target model **mcb_ee_pmsm_foc**.

2. Select one of these options in the InverterSelected radio group in the target model:

- **Motor Control Blockset average inverter** - Select this option to use the Average Inverter and Surface Mount PMSM blocks.

- **Simscape Electrical 3 phase converter** - Select this option to use the Converter (Three-Phase) and PMSM blocks.

- **Simscape Electrical Modular Multilevel converter** - Select this option to use the Ideal Semiconductor Switch and PMSM blocks.

3. Select an option from the InverterSelected radio group and click **Run** on the **Simulation** tab to simulate the target model.

4. On the target model, click **Data Inspector** on the **Simulation** tab to view results from the three simulation runs.

This image shows the simulation results for I_a phase current:
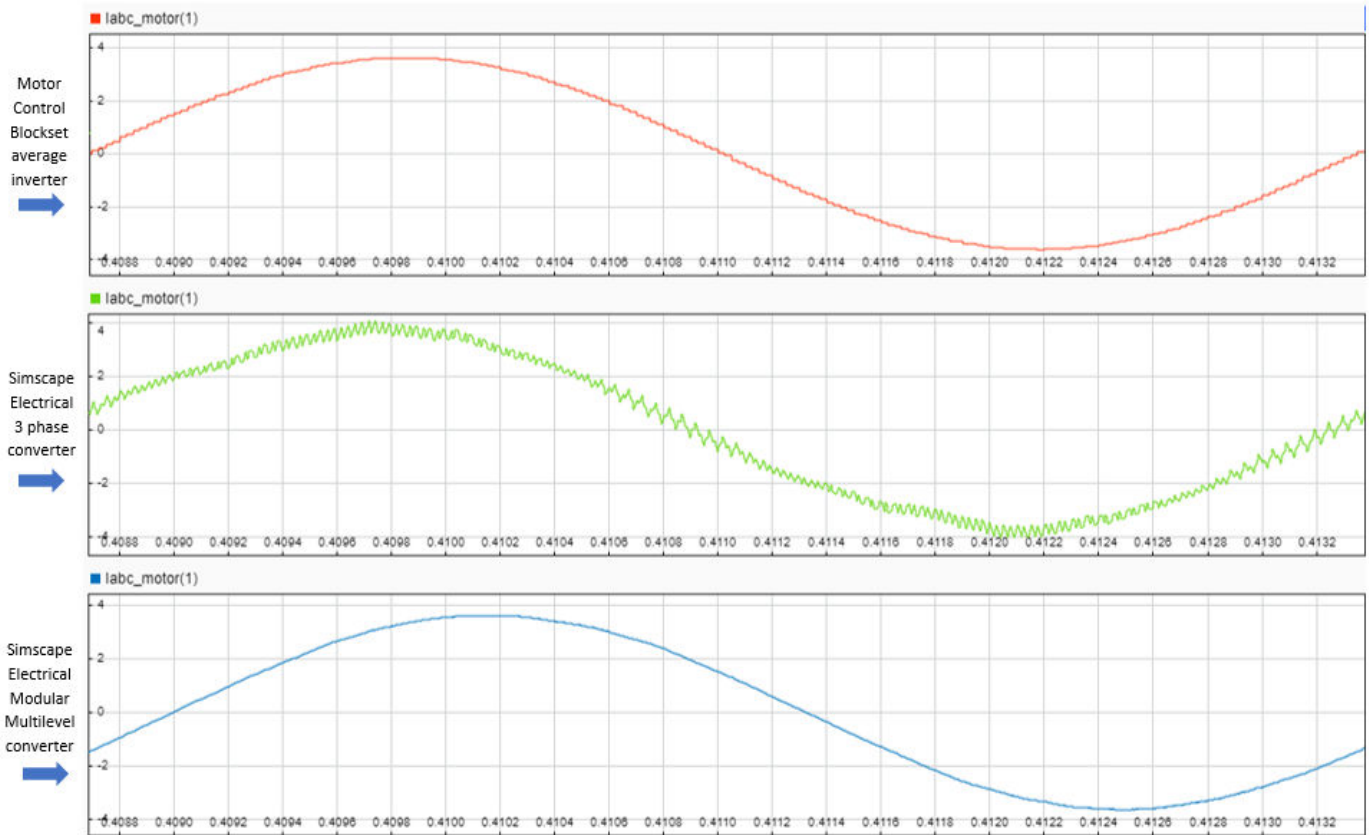
These images show the comparison of rotor speed, I_q current, I_{ab} phase current, and rotor position for the three inverter types:
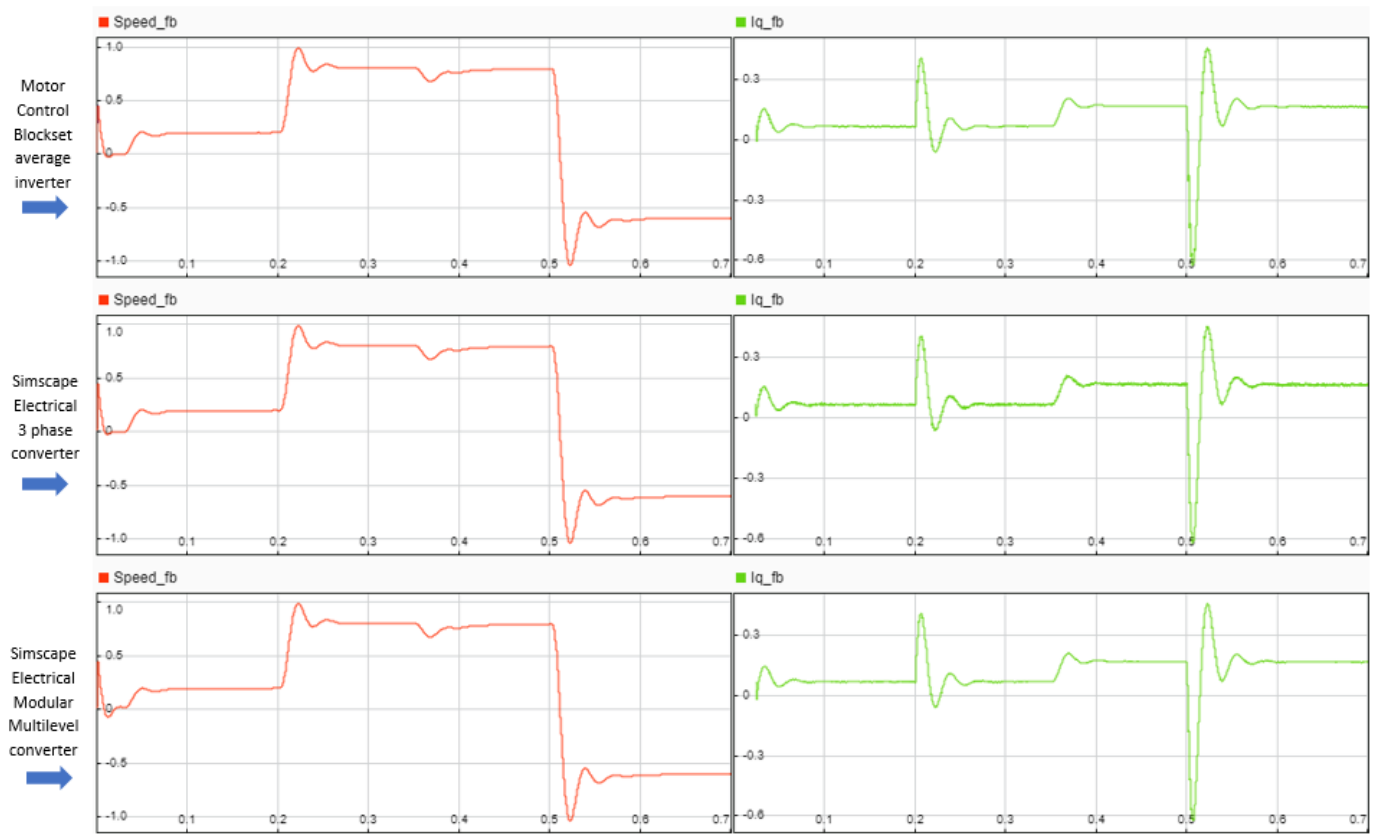
Motor Control Blockset average inverter

Simscape Electrical 3 phase converter

Simscape Electrical Modular Multilevel converter
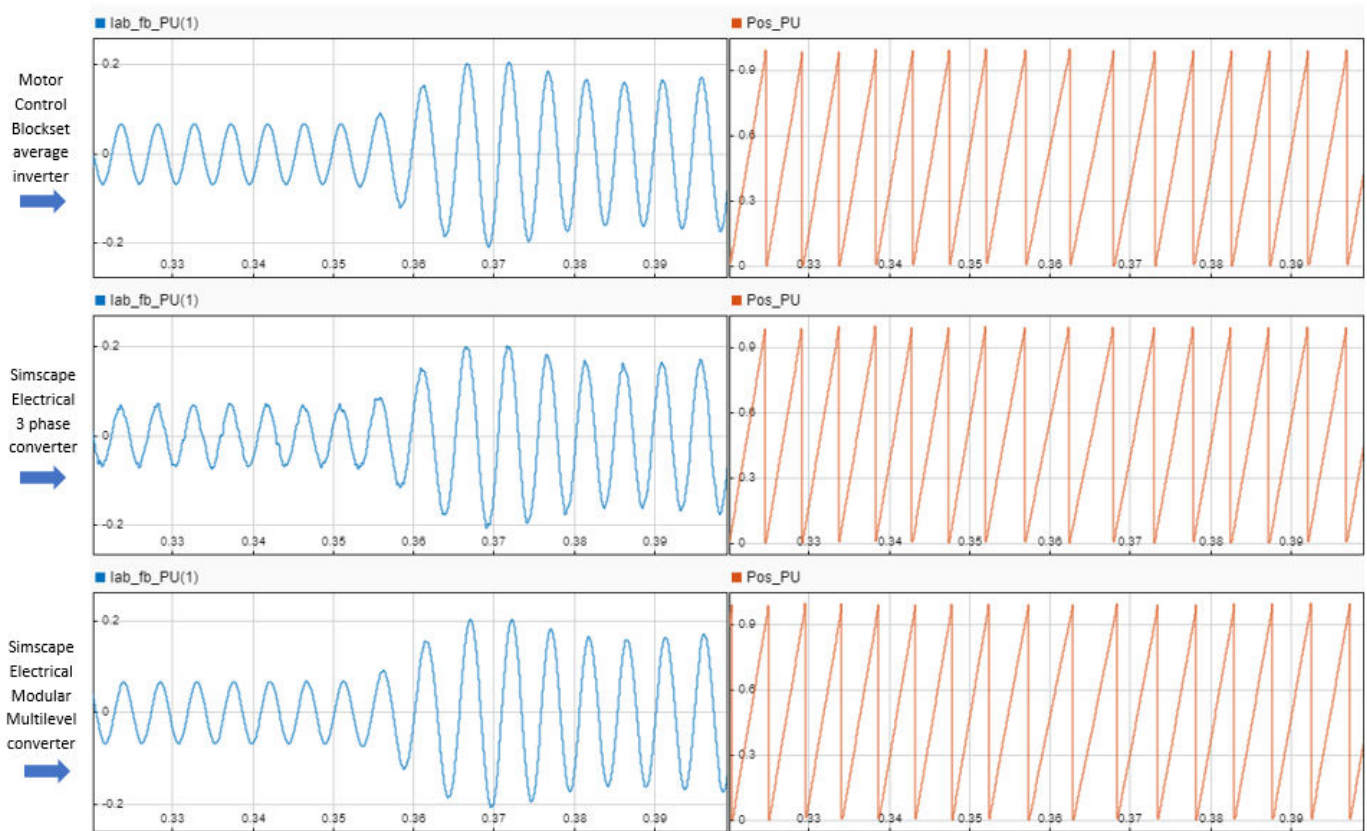
These images show the comparison of PWM modulation waveforms for the three inverter types:

Motor Control Blockset average inverter

Simscape Electrical 3 phase converter

Simscape Electrical Modular Multilevel converter

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_ee_pmsm_foc**

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

5. Open the target model. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

6. To ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1, load a sample program to CPU2 of LAUNCHXL-F28379D, for example, a program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx).

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

8. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. For example, use this command for a F28379D based controller:

```
open_system('mcb_pmsm_foc_host_model_f28379d.slx');
```

PMSM Control Host
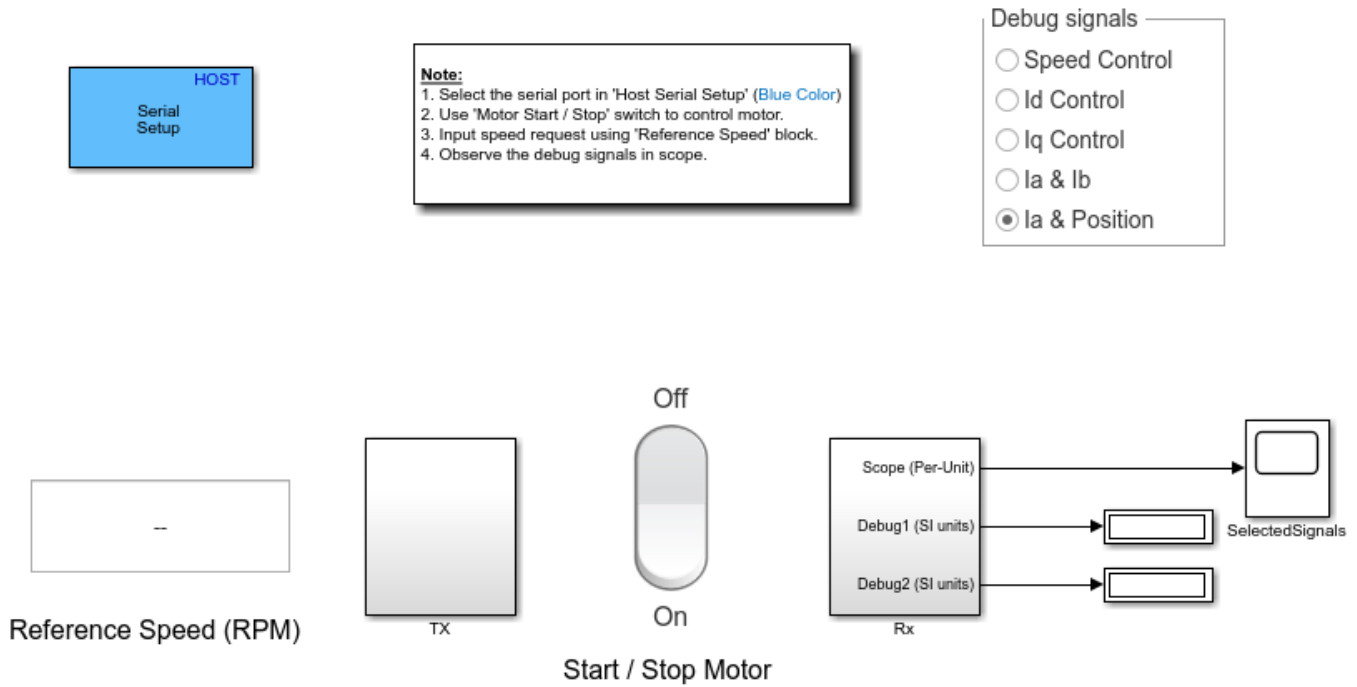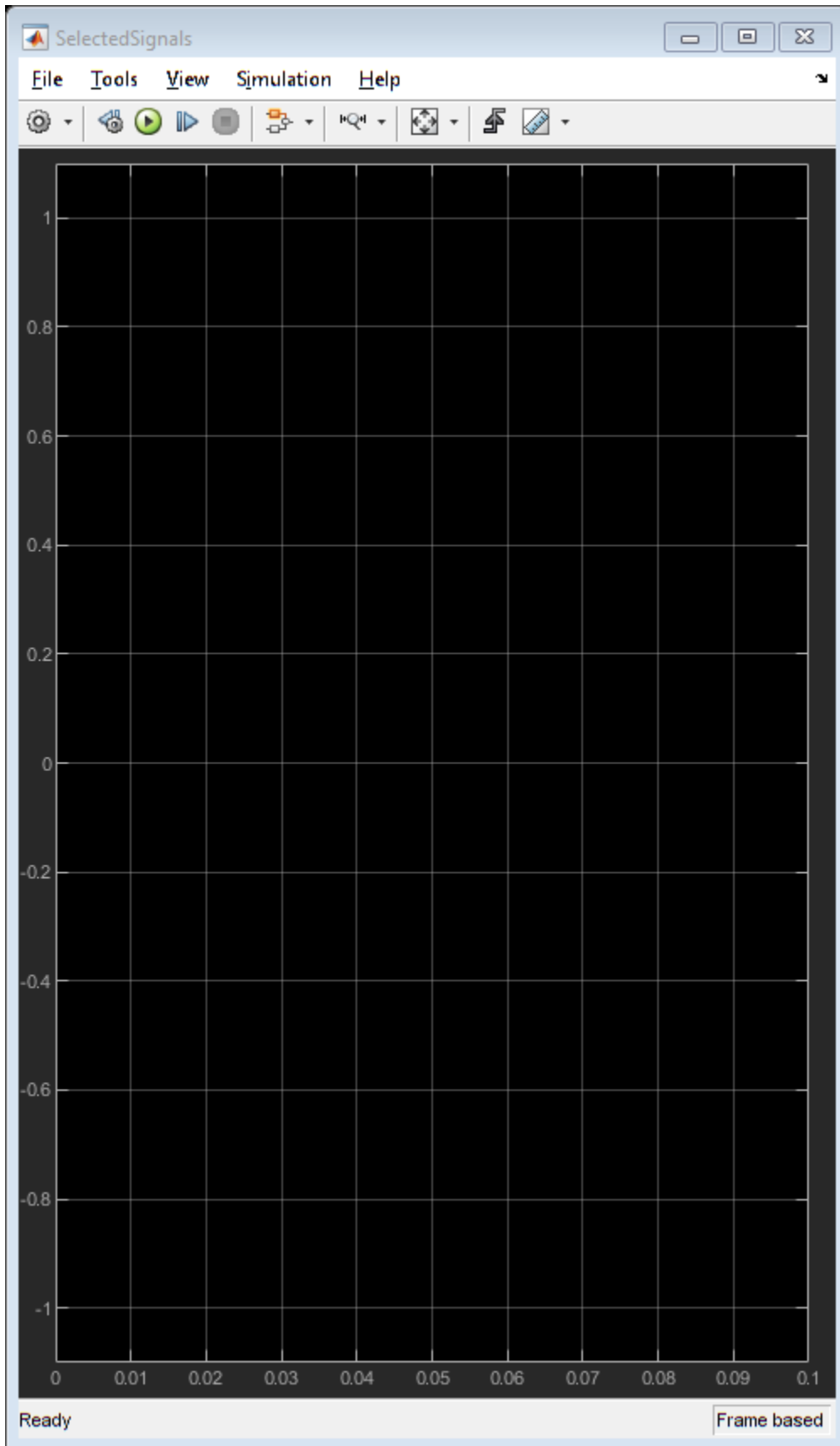
HOST
Serial
Setup

Note:
1. Select the serial port in 'Host Serial Setup' (Blue Color)
2. Use 'Motor Start / Stop' switch to control motor.
3. Input speed request using 'Reference Speed' block.
4. Observe the debug signals in scope.

Debug signals
○ Speed Control
○ Id Control
○ Iq Control
○ Ia & Ib
◉ Ia & Position

Off

On

--

Reference Speed (RPM)

TX

Start / Stop Motor

Scope (Per-Unit)

Debug1 (SI units)

Debug2 (SI units)

Rx

SelectedSignals

Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

9. In the Host Serial Setup block mask of the host model, select a **Port name**.

10. Update the **Reference Speed** value in the host model.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On, to start running the motor.

13. Observe the debug signals from the RX subsystem, in the Time Scope and Display blocks of the host model.

Note: In the host model, you can also select the debug signals that you want to monitor.

Other Things to Try

You can also use SoC Blockset™ to implement a closed-loop motor control application that addresses challenges related to ADC-PWM synchronization, controller response, and studying different PWM settings. You can use Simscape™ Electrical™ to implement high fidelity inverter simulation. For details, see "Integrate MCU Scheduling and Peripherals in Motor Control Application" (SoC Blockset).

Control PMSM Loaded with Dual Motor (Dyno)

This example uses field-oriented control (FOC) to control two three-phase permanent magnet synchronous motors (PMSM) coupled in a dyno setup. Motor1 runs in the closed-loop speed control mode. Motor2 runs in the torque control mode and loads Motor 1 because they are mechanically coupled. You can use this example to test a motor in different load conditions.

The example simulates two motors that are connected back-to-back. You can use a different speed reference for Motor1 and different torque reference or current reference (Iq) for Motor2. Motor1 runs at the reference speed for the load conditions provided by Motor2 (with different current reference).

The example runs in the controller hardware board. You can input the speed reference for Motor1 and current reference for Motor2 using a host model. The host model uses serial communication to communicate with the controller hardware board.

Current control loops in Motor1 and Motor2 control algorithms are offset by Ts/2, where Ts is the control-loop execution rate.

Models

The example includes this model:

- **mcb_pmsm_foc_f28379d_dyno**

You can use this model for both simulation and code generation. You can use the open_system command to open the Simulink® model. For example, use this command for a F28379D based controller:

```
open_system('mcb_pmsm_foc_f28379d_dyno.slx');
```
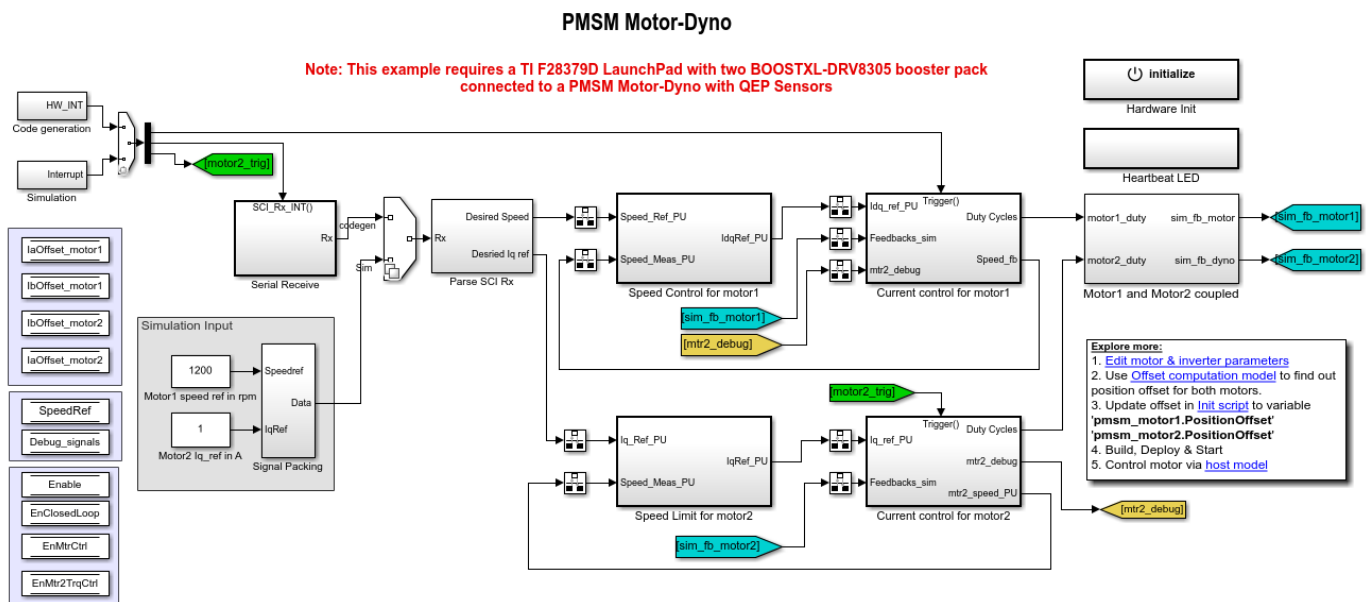

Copyright 2020 The MathWorks, Inc.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters for both Motor1 and Motor2. We provide default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use, by using the Motor Control Blockset™ parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

2. Update the motor parameters (that you obtained from the datasheet, other sources, or parameter estimation tool) and inverter parameters in the model initialization script associated with the Simulink® model. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

For this example, update the motor parameters for both the motors in the model initialization script.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open a model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

4. Input a different speed reference for Motor1 and a different current reference (load) for Motor2. Observe the measured speed and other logged signals in the Data Inspector.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the FOC algorithm on the target hardware.

The example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in a closed-loop control.

Required Hardware

The example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + 2 BOOSTXL-DRV8305 inverters: **mcb_pmsm_foc_f28379d_dyno**

For connections related to the preceding hardware configuration, see "Instructions for Dyno (Dual Motor) Setup" on page 7-9.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable inverter.ADCOffsetCalibEnable in the model initialization script.

Alternatively, you can compute the ADC offset values and update it manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Compute the quadrature encoder index offset value and update it in the model initialization scripts associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

For this example, update the QEP offset values in the pmsm_motor1.PositionOffset and pmsm_motor2.PositionOffset variables in initialization script.

5. Open the target model. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

6. To ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1, load a sample program to CPU2 of LAUNCHXL-F28379D, for example, a program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx).

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the model to the hardware.

8. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model:

```
open_system('mcb_pmsm_foc_host_model_dyno.slx');
```

PMSM Dyno Control Host

Debug signals
- ⦿ Mtr1: Speed ref & Speed feedback
- ○ Mtr1: Id ref & Id feedback
- ○ Mtr1: Iq ref & Iq feedback
- ○ Mtr1: Vd & Vq
- ○ Mtr1: Ia & Ib feedback
- ○ Mtr1: Pm & Te
- ○ Mtr2: Id ref & Id feedback
- ○ Mtr2: Iq ref & Iq feedback
- ○ Mtr2: Vd & Vq
- ○ Mtr2: Ia & Ib feedback
- ○ Mtr2: Pm & Te
- ○ Mtr1&Mtr2: Pm
- ○ Mtr1&Mtr2: Te
- ○ Mtr1&Mtr2: Pos
- ○ Mtr1&Mtr2: Ia
- ○ Mtr1 Speed ref & Mtr2 Speed fb

Note:
1. Select the serial port in 'Host Serial Setup' (Blue Color)
2. Use 'Motor Start / Stop' switch to control motor.
3. Input speed request for Motor 1 using 'Motor 1 Reference Speed' knob.
4. Input Iq ref for Motor 2 using 'Motor 2 - IqRef (PU)' Knob
4. Observe the debug signals in scope.

1200

Motor 1 - Reference Speed (RPM)

1

Motor 2 - Iq Ref (A)

Off

On

Start / Stop Motor 1

HOST
Serial
Setup

TX

Scope (Per-Unit)

Debug1 (SI units)

Debug2 (SI units)

Rx

SelectedSignals
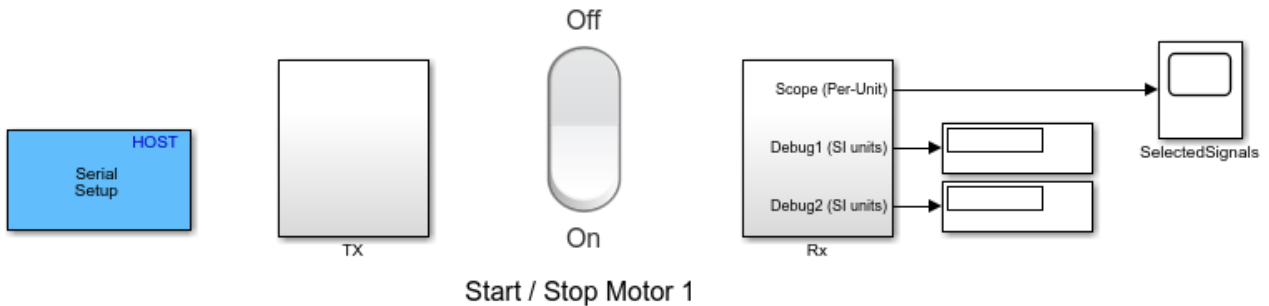
Copyright 2020 The MathWorks, Inc.

9. In the Host Serial Setup block mask of the host model, select a **Port name**.

10. Click **Run** on the **Simulation** tab to run the host model.

11. Change the position of the **Start / Stop Motor 1** switch to On, to start running the motor.

12. Update the **Reference Speed for Motor1** and **Current Reference for Motor2** in the host model.

13. Select the debug signals that you want to monitor, to observe them in the Time Scope block of host model.

Other Things to Try

You can also use SoC Blockset™ to develop a real-time motor control application for a dual motor setup that utilizes multiple processor cores to obtain design modularity, improved controller performance, and other design goals. For details, see "Partition Motor Control for Multiprocessor MCUs" (SoC Blockset)

Field-Oriented Control of Induction Motor Using Speed Sensor

This example implements the field-oriented control (FOC) technique to control the speed of a three-phase AC induction motor (ACIM). The FOC algorithm requires rotor speed feedback, which is obtained in this example by using a quadrature encoder sensor. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2

This example uses the quadrature encoder sensor to measure the rotor speed. The quadrature encoder sensor consists of a disk with two tracks or channels that are coded 90 electrical degrees out of phase. This creates two pulses (A and B) that have a phase difference of 90 degrees and an index pulse (I). Therefore, the controller uses the phase relationship between A and B channels and the transition of channel states to determine the direction of rotation of the motor.
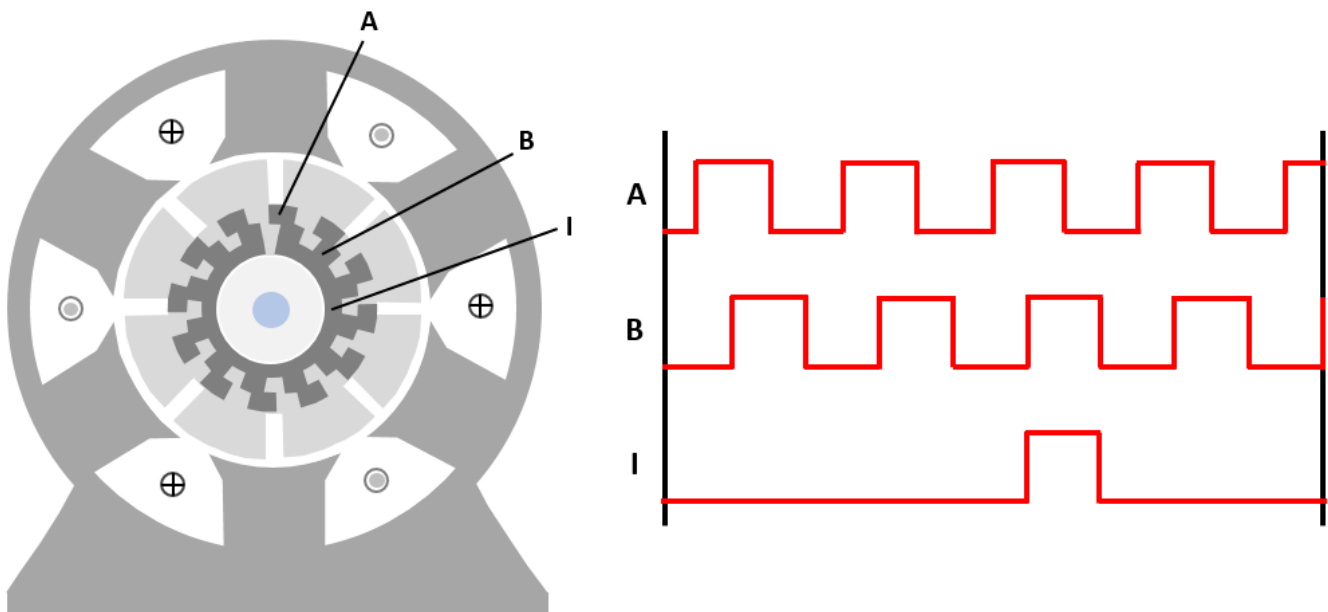
Model

The example includes the model **mcb_acim_foc_qep_f28379d**.

You can use this model for simulation and code generation. You can use the open_system command to open the Simulink® model.

```
open_system('mcb_acim_foc_qep_f28379d.slx');
```

Field-Oriented Control of AC Induction Motor

Note: This example requires a TI F28379D LaunchPad with a BOOSTXL-DRV8305 booster pack connected to an Induction Motor with QEP Sensor

Copyright 2020 The MathWorks, Inc.

For details on the supported hardware configuration, see the Required Hardware section under Generate Code and Deploy Model to Target Hardware.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (needed only for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide the default motor parameters with the Simulink® model that you can replace with values from either the motor datasheet or other sources.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open the model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

4-105

Generate Code and Deploy Model to Target Hardware

This section instructs you on how to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in closed-loop control.

Required Hardware

This example supports the following hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration from the MATLAB® command prompt.

• LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_acim_foc_qep_f28379d**

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the ADC (or current) offset values. To disable this functionality (enabled by default), update the value 0 to the variable *inverter.ADCOffsetCalibEnable* in the model initialization script.

Alternatively, you can compute the ADC offset values and update them manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Open the target model. If you want to change the default hardware configuration settings in the model, see "Model Configuration Parameters" on page 2-2.

5. Load a sample program to CPU2 of the LAUNCHXL-F28379D, for example program that operates the CPU2 blue LED, by using the GPIO31 pin (*c28379D_cpu2_blink.slx*), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

7. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model.

```
open_system('mcb_acim_foc_host_model.slx');
```

AC Induction Motor
Field Oriented Control Host
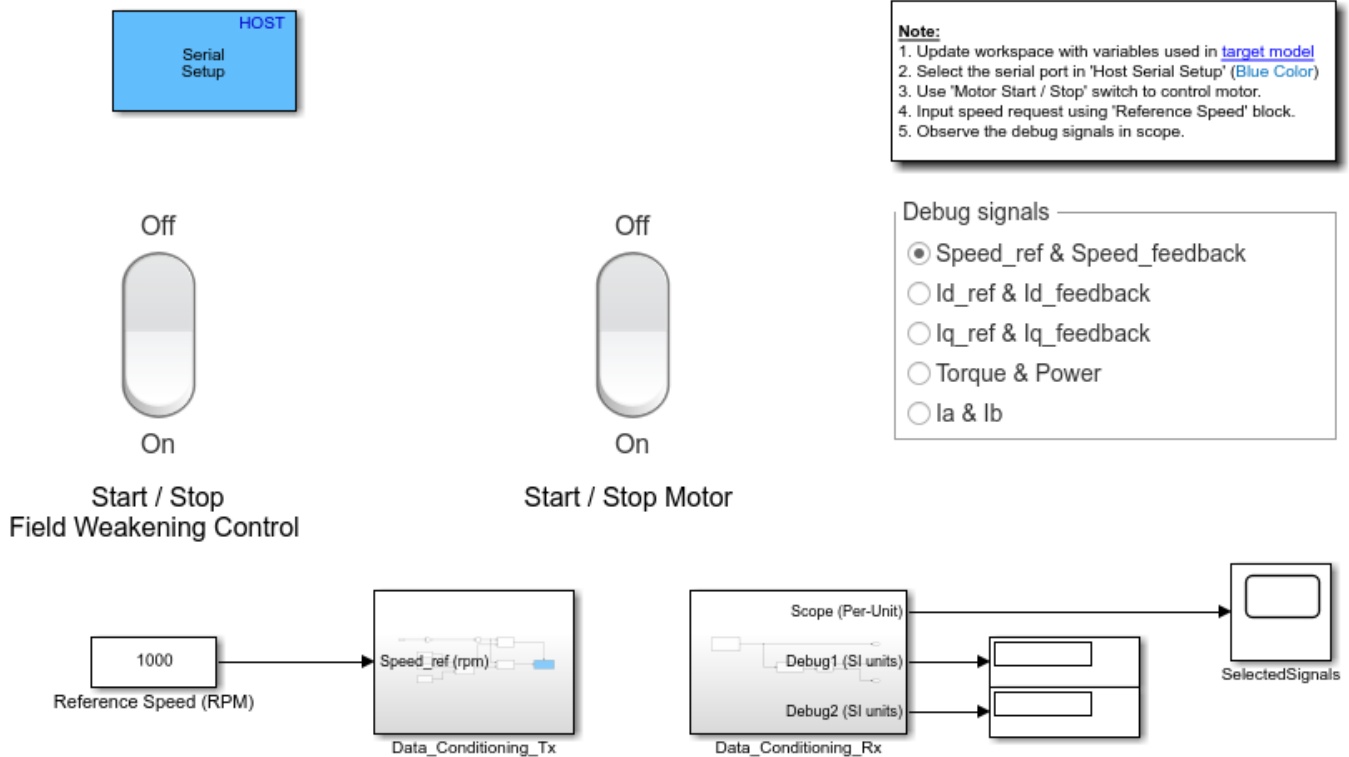
HOST
Serial
Setup

Note:
1. Update workspace with variables used in target model
2. Select the serial port in 'Host Serial Setup' (Blue Color)
3. Use 'Motor Start / Stop' switch to control motor.
4. Input speed request using 'Reference Speed' block.
5. Observe the debug signals in scope.

Off

Off

Debug signals

⦿ Speed_ref & Speed_feedback
◯ Id_ref & Id_feedback
◯ Iq_ref & Iq_feedback
◯ Torque & Power
◯ Ia & Ib

On

On

Start / Stop
Field Weakening Control

Start / Stop Motor

1000

Reference Speed (RPM)

Speed_ref (rpm)

Data_Conditioning_Tx

Scope (Per-Unit)

Debug1 (SI units)

Debug2 (SI units)

Data_Conditioning_Rx

SelectedSignals

Copyright 2020 The MathWorks, Inc.

For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

8. In the Host Serial Setup block mask of the host model, select a **Port name**.

9. Update the *Reference Speed* value in the host model.

10. In the **Debug signals** section, select a signal that you want to monitor.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On to start running the motor.

13. Observe the debug signals from the RX subsystem in the **SelectedSignals** time scope of the host model.

Sensorless Field-Oriented Control of Induction Motor

This example uses sensorless position estimation to implement the field-oriented control (FOC) technique to control the speed of a three-phase AC induction motor (ACIM). For details about FOC, see "Field-Oriented Control (FOC)" on page 4-2.

This example uses rotor Flux Observer block to estimate the position of rotor flux.

The block uses stator voltages (V_α, V_β) and currents (I_α, I_β) as inputs and estimates the rotor flux, generated torque, and the rotor flux position.

Model

The example includes the model **mcb_acim_foc_sensorless_f28379d**.

You can use this model for both simulation and code generation. You can use the open_system command to open the Simulink® model.

```
open_system('mcb_acim_foc_sensorless_f28379d.slx');
```


For details on the supported hardware configuration, see the Required Hardware section under Generate Code and Deploy Model to Target Hardware.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®

- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (needed only for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide the default motor parameters with the Simulink® model that you can replace with the values from either the motor datasheet or other sources.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open the model included with this example.

2. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section instructs you on how to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink® model and run the motor in closed-loop control.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_acim_foc_qep_f28379d**

For connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model automatically computes the Analog-to-Digital Converter (ADC) or current offset values. To disable this functionality (enabled by default), update the value 0 to the variable *inverter.ADCOffsetCalibEnable* in the model initialization script.

Alternatively, you can compute the ADC offset values and update them manually in the model initialization scripts. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. Open the target model. If you want to change the default hardware configuration settings in the model, see "Model Configuration Parameters" on page 2-2.

5. Load a sample program to CPU2 of the LAUNCHXL-F28379D, for example program that operates the CPU2 blue LED, using the GPIO31 pin (*c28379D_cpu2_blink.slx*), to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

7. In the target model, click the **host model** hyperlink to open the associated host model. You can also use the open_system command to open the host model.

```
open_system('mcb_acim_foc_host_model.slx');
```


For details about the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

8. In the Host Serial Setup block mask of the host model, select a **Port name**.

9. Update the *Reference Speed* value in the host model.

10. In the **Debug signals** section, select a signal that you want to monitor.

11. Click **Run** on the **Simulation** tab to run the host model.

12. Change the position of the Start / Stop Motor switch to On, to start running the motor in the open-loop condition (by default, the motor spins at 10% of the base speed).

Note: Do not run the motor (using this example) in the open-loop condition for long. The motor may draw high currents and produce excessive heat.

We designed the open-loop control to run the motor with a Reference Speed that is less than or equal to 10% of base speed.

13. Increase the motor *Reference Speed* beyond 10% of the base speed to switch from open-loop to closed-loop control.

NOTE: To change the motor's direction of rotation, reduce the motor *Reference Speed* to a value less than 10% of the base speed. This brings the motor back to the open-loop condition. Change the direction of rotation, but keep the *Reference Speed* magnitude constant. Then transition to the closed-loop condition.

14. Observe the debug signals from the RX subsystem in the **SelectedSignals** time scope of the host model.

NOTE: The Flux Observer block is designed to work with PMSM but its output is modified to work with induction motor. For custom motors, update the Offset_Correction block (in Current Control/Input Scaling/Calculate position and speed subsystem) to adjust the delay in the position estimation.

Tune PI Controllers Using Field Oriented Control Autotuner Block on Real-Time Systems

This example computes the gain values of proportional-integral (PI) controllers within the speed and current controllers by using the Field Oriented Control Autotuner block. For details about field-oriented control, see "Field-Oriented Control (FOC)" on page 4-2.

This model supports both simulation and code generation. When you run the model, it uses the simple values of gains for the PI controllers to achieve the steady state of the speed-control operation.

The model begins tuning only in the steady state. It introduces disturbances in the controller output depending on the controller goals (bandwidth and phase margin). The model uses the system response to disturbances to calculate the optimal controller gain.

Model

The example includes the model **mcb_pmsm_foc_autotuner_speedgoat**.

You can use this model for both simulation and code generation. You can use the open_system command to open the Simulink® model.

```
open_system('mcb_pmsm_foc_autotuner_speedgoat.slx');
```
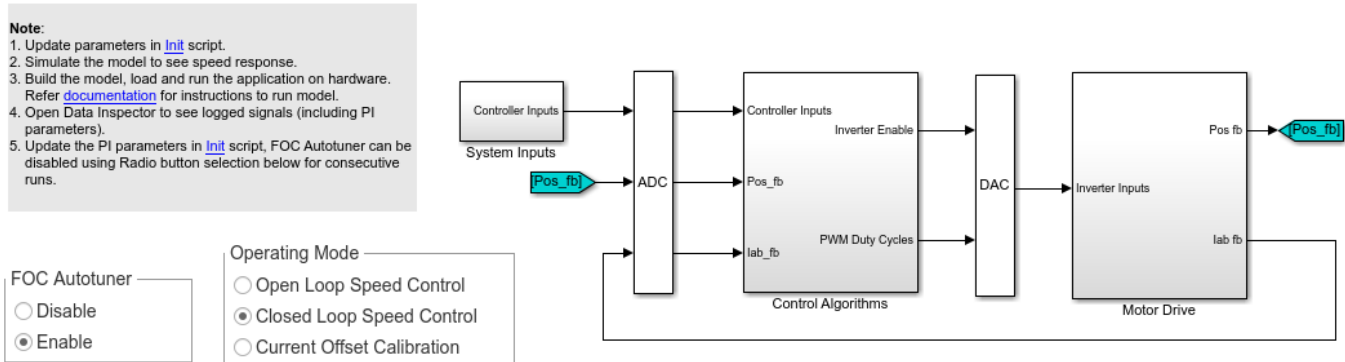

For details on the supported hardware configuration, see the Required Hardware section under Generate Code and Deploy Model to Target Hardware.

Required MathWorks® Products

- Motor Control Blockset™
- Simulink Control Design™
- Simulink Real-Time™

- Speadgoat Library

Prerequisites

1. The motor parameters available in the example model are for the motor that comes with the Speedgoat Electric Motor Control Kit. You can modify these parameters for any other motor by replacing them with values from either the motor datasheet or other sources.

2. If you obtain the motor parameters from the datasheet or other sources, update the motor and inverter parameters in the model initialization script associated with the Simulink® models. For instructions, see "Model Initialization Script" on page 3-3.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open the model included with this example.

2. Check the reference speed profile configured in the signal builder (available in mcb_pmsm_foc_autotuner_speedgoat/System Inputs/Speed Reference).

3. Check and update the FOC Autotuner parameters in the Field Oriented Control Autotuner block mask (available in the Control Algorithms/FOC_AutoTuner subsystem). For details about the Field Oriented Control Autotuner block, see Field Oriented Control Autotuner.

4. Check and update the simple gain values in the model initialization script associated with the model.

5. Click **Run** on the **Simulation** tab to simulate the model.

6. Verify that the motor reaches steady state operation for at least half of the rated speed using the simple gain values that you entered. The model begins field-oriented control (FOC) tuning (using the Field Oriented Control Autotuner block) at the seventeenth second.

7. After tuning completes, observe the computed PI controller gain values in the Display PI Params block available in the Control Algorithms subsystem.

8. Observe the system response with the newly computed PI parameters by using the Simulation Data Inspector.

For more details, see "Tune PI Controllers by Using Field Oriented Control Autotuner" on page 4-25.

Generate Code and Deploy Model to Target Hardware

This section instructs you on how to generate code and run the FOC algorithm on the target hardware.

Required Hardware

This example supports Speedgoat Electric Motor Control Kit that includes these components:

- Three-phase inverter kit from Speedgoat.

- Three-phase brushless DC motor attached with the quadrature encoder sensor.

For more details about Speedgoat Electric Motor Control Kit, see *Electric Motor Control Kit*.

For details about Speadgoat hardware setup, see *Speedgoat Software Configuration Guide*.

Generate Code and Run Model on Target Hardware

1. Simulate the model and verify that you are obtaining the desired controller response.

2. Complete the hardware connections for the Speedgoat Electric Motor Control Kit.

• **Calibrate current offset**

1. In the model, set **Operating Mode** to **Current Offset Calibration**.

2. In the **Real-Time** tab on the Simulink toolstrip, click **Build Model** in the **Run on Target** drop-down menu to build the model.

NOTE: Do not click **Run on Target** because this example model does not support real-time execution in external mode.

3. Navigate to the folder where Simulink built the model. Double click the file *mcb_pmsm_foc_autotuner_speedgoat.mldatx* to open the Simulink Real-Time Application dialog box.
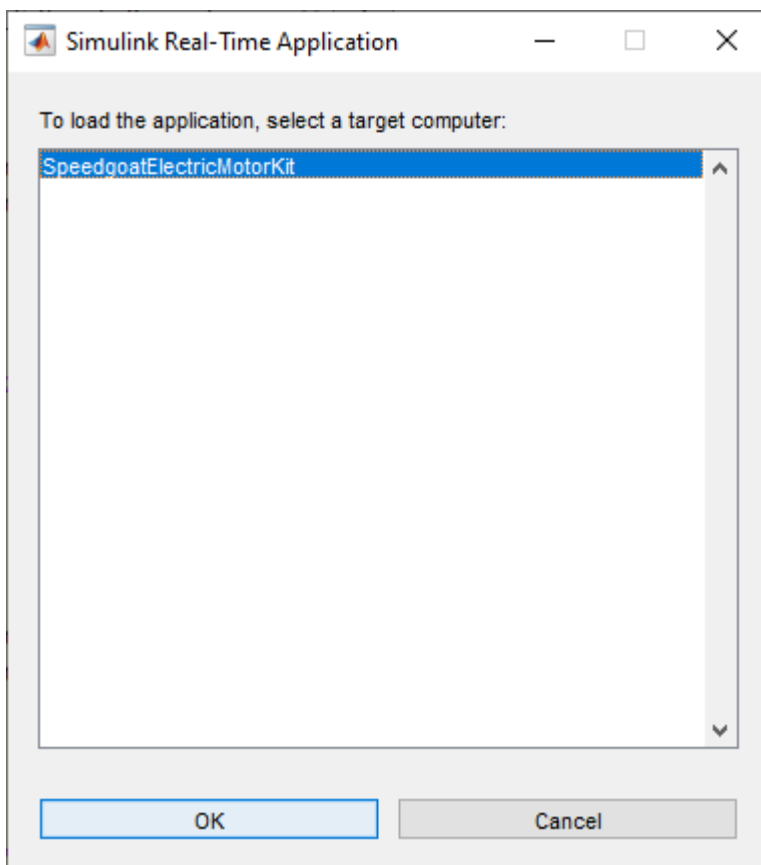
4. In the Simulink Real-Time Application dialog box, select the target computer to which you are connected. Click **OK** to load the application file to the hardware.

5. Enter these commands (in the same order) at the MATLAB command prompt to execute the loaded application on the hardware.

- *tg = slrealtime;*
- *tg.start;*

6. After the model runs successfully, use **Data Inspector** on the **Simulation** tab to see the logged signals. The stabilized *Iab_offset* signals are the current offsets.

7. Update the current offset values in the *inverter.CtSensAOffset* and *inverter.CtSensBOffset* variables available in the model initialization script associated with the Simulink model.

- **Run motor in open-loop control**

1. In the model, set **Operating Mode** to **Open Loop Speed Control**.

2. In the **Real-Time** tab on the Simulink toolstrip, click **Build Model** in the **Run on Target** drop-down menu to build the model.

3. Navigate to the folder where Simulink built the model. Double click the file *mcb_pmsm_foc_autotuner_speedgoat.mldatx* to open the Simulink Real-Time Application dialog box.
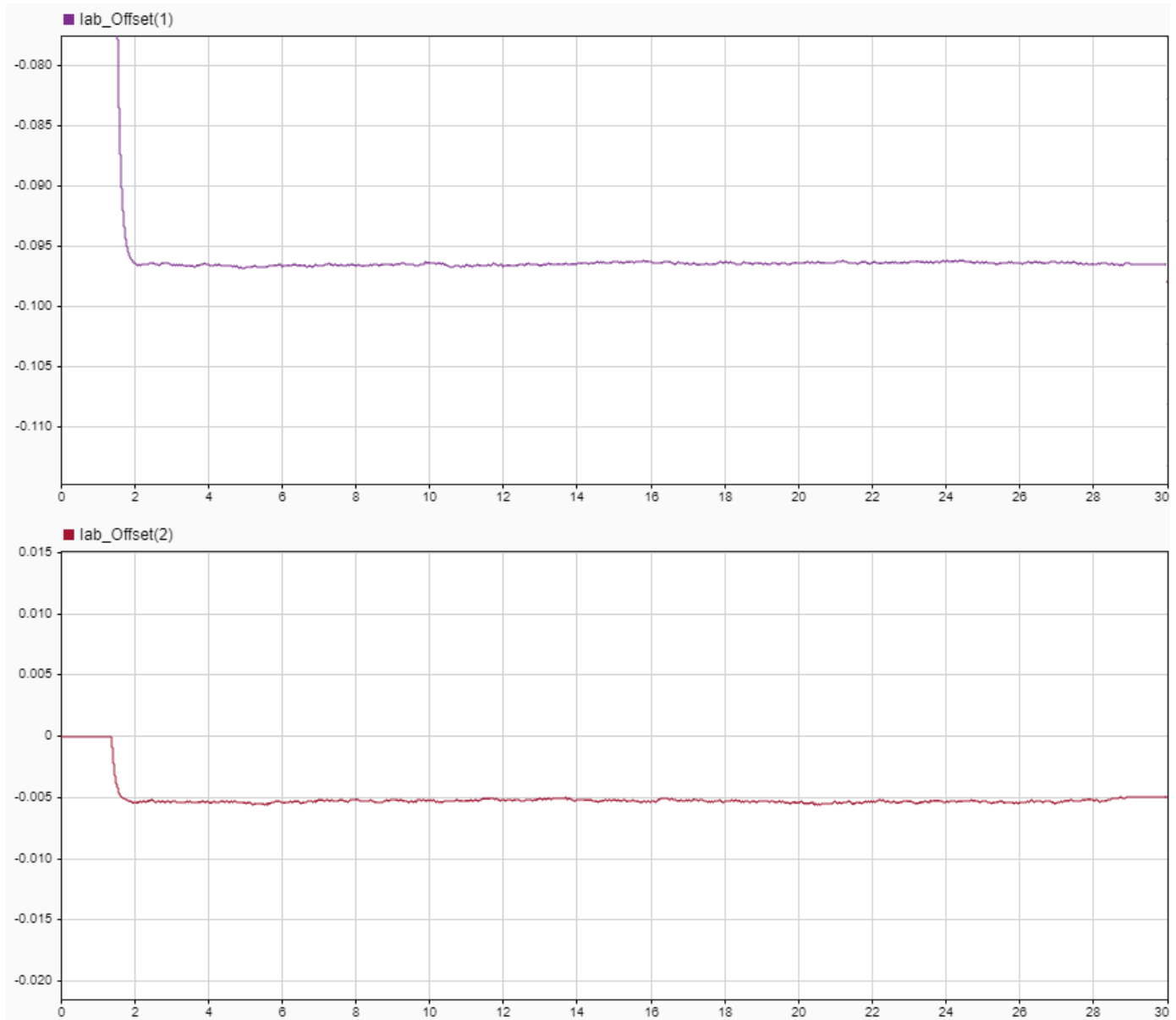
4. In the Simulink Real-Time Application dialog box, select the target computer to which you are connected. Click **OK** to load the application file to the hardware.

5. Enter these commands (in the same order) at the MATLAB command prompt to execute the loaded application on the hardware.

- *tg = slrealtime;*
- *tg.start;*

6. After the model executes, use **Data Inspector** on the **Simulation** tab to see the logged signals. Verify that speed feedback (*Speed_fb*) follows the reference speed (*Speed_Ref*) signal.
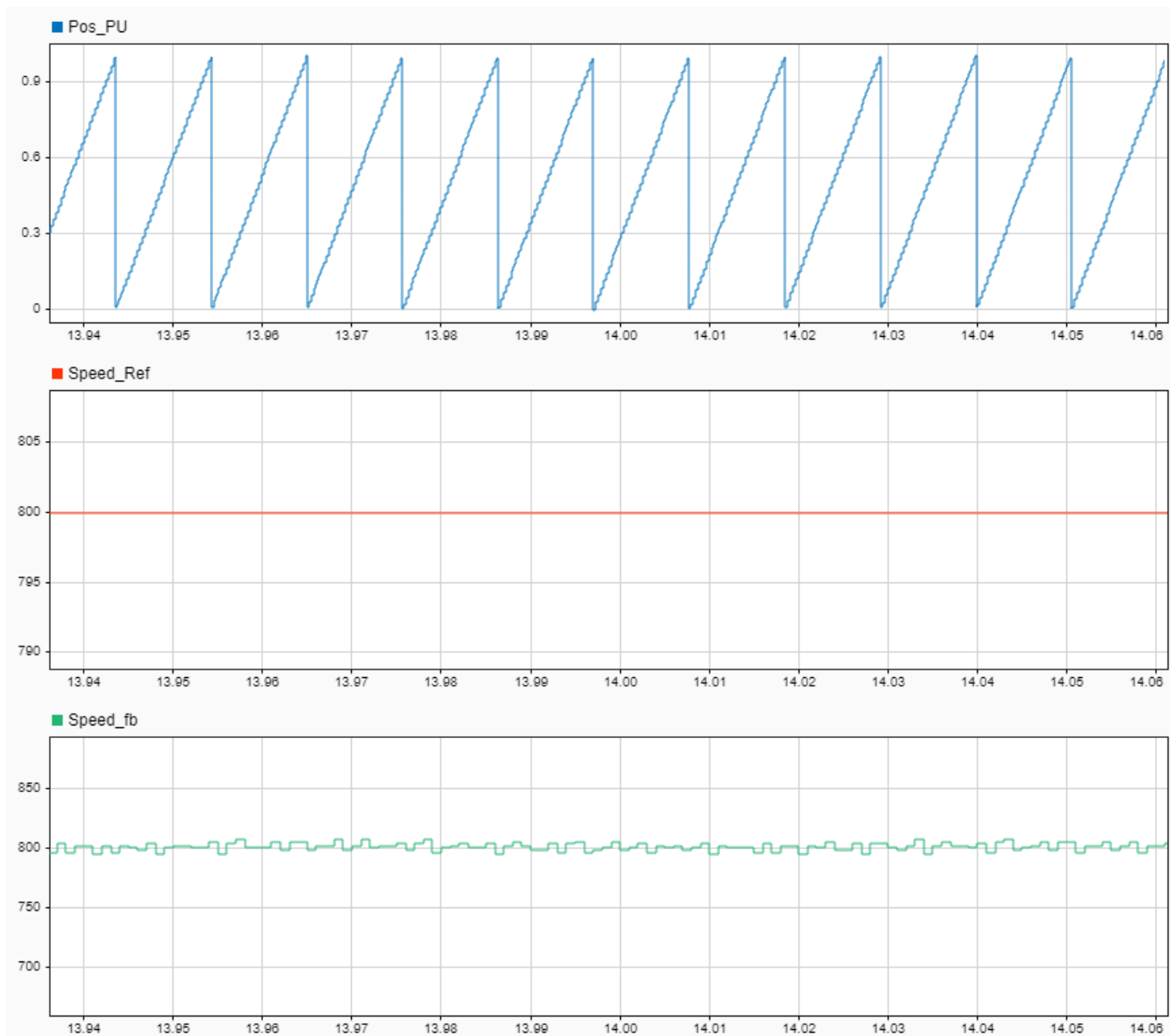
For example, verify that the positive reference speed has a positive speed feedback, and the position signal (*Pos_PU*) has a positive ramp.

If there is a mismatch in the sign of the reference speed and speed feedback signals, change the **A leads B** parameter (of the Inverter and Plant model/SpeedGoatDrivers/Condition Encoder block) either from 0 to 1 or from 1 to 0. Then follow steps 2 to 6 in this section to execute the model again on the hardware.

NOTE: In the Open Loop Speed Control mode, the motor speed is limited between 500 rpm and 1200 rpm.

- **Run motor in closed-loop control**

1. In the model, set **Operating Mode** to **Closed Loop Speed Control**.

2. Set the **FOC Autotuner** button on the model to **Disable** to disable the field-oriented control (FOC) Autotuner.

3. In the **Real-Time** tab on the Simulink toolstrip, click **Build Model** in the **Run on Target** drop-down menu to build the model.

4. Navigate to the folder where Simulink built the model. Double-click the file *mcb_pmsm_foc_autotuner_speedgoat.mldatx* to open the Simulink Real-Time Application dialog box.

5. In the Simulink Real-Time Application dialog box, select the target computer to which you are connected. Click **OK** to load the application file to the hardware.

6. Enter these commands (in the same order) at the MATLAB command prompt to execute the loaded application on the hardware and run the motor.

- *tg = slrealtime;*
- *tg.start;*

The motor runs in closed-loop control at a speed that is configured in the signal builder.
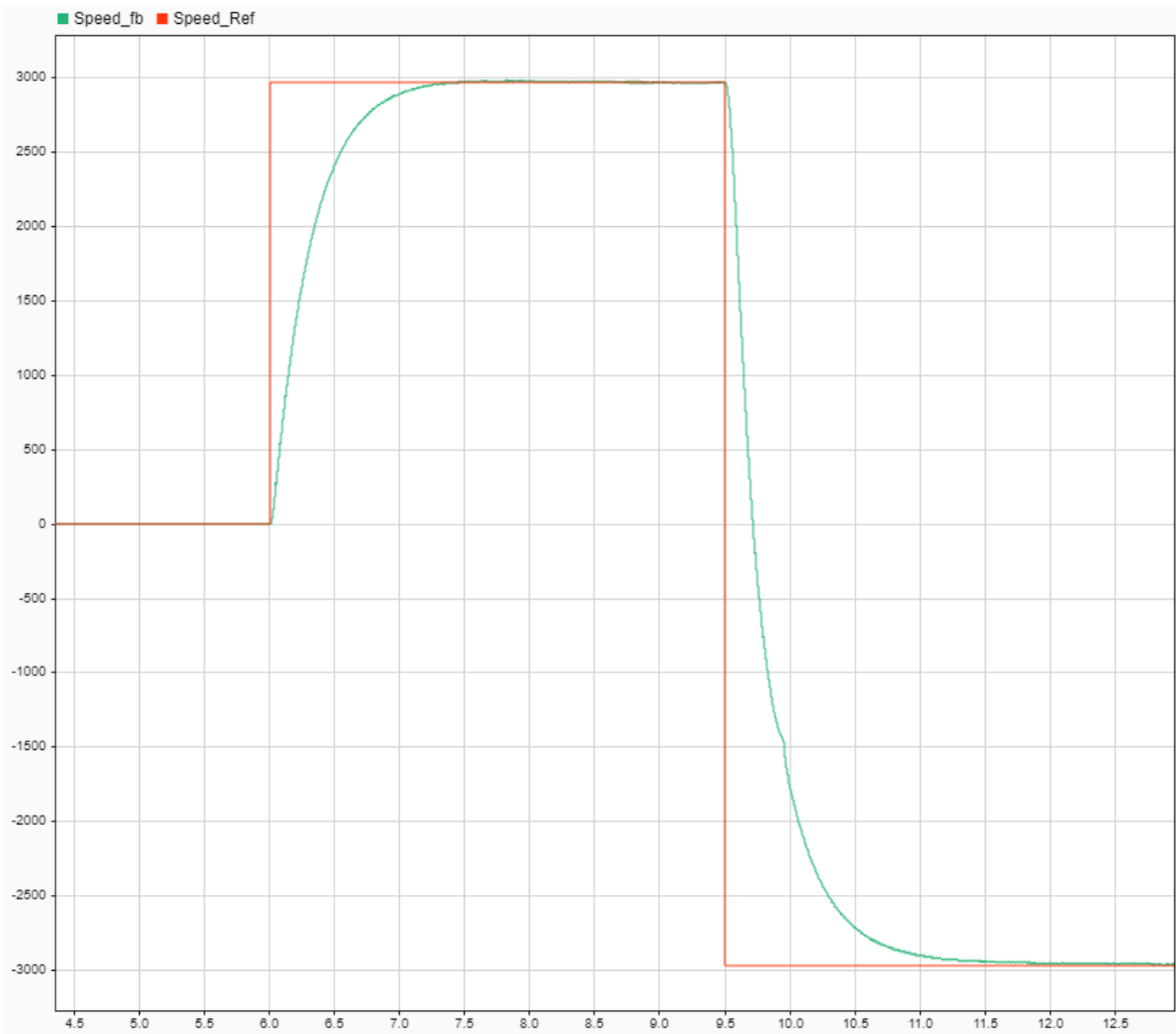
7. Verify that the motor reaches steady state operation because the FOC Autotuner will not work if the motor speed is unstable.

If the motor fails to reach the steady state, change the PI parameters manually in the model initialization script (associated with the model), until the motor speed stabilizes to half the base speed of the motor.

NOTE: When tuning the PI parameters in the model initialization script, the motor may show a slow speed response.

8. If the motor reaches a stable speed, follow the steps to run FOC Autotuner.
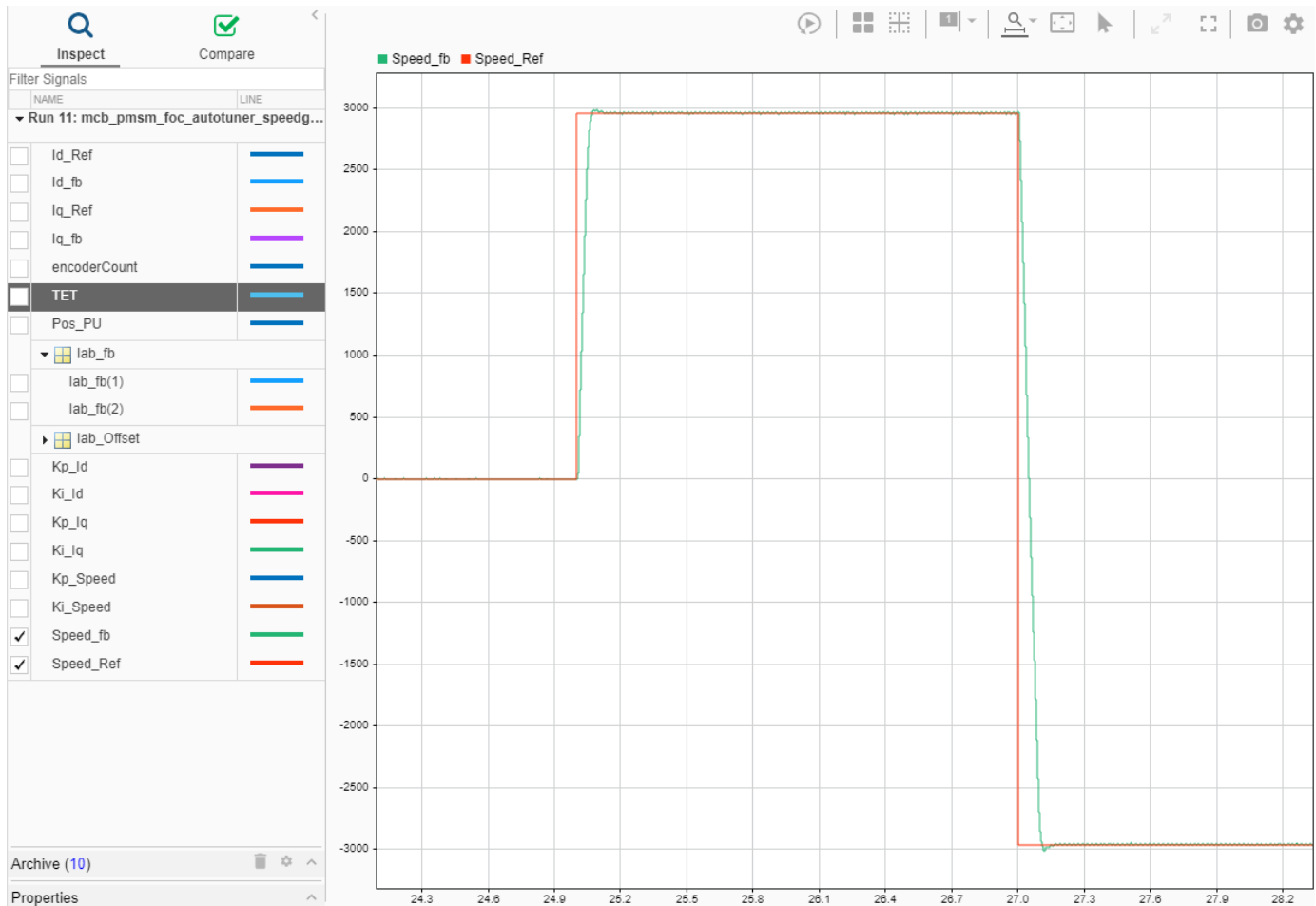
- **Run FOC Autotuner**

1. Set the **FOC Autotuner** button on the model to **Enable** to enable the field-oriented control autotuner.

2. Verify if **Operating Mode** is set to **Closed Loop Speed Control**.

3. Check and update the FOC Autotuner parameters (such as autotuner trigger timing and controller target) in the Field Oriented Control Autotuner block mask (available inside Control Algorithms/ FOC_AutoTuner subsystem). For details about the Field Oriented Control Autotuner block, see Field Oriented Control Autotuner.

4. In the **Real-Time** tab on the Simulink toolstrip, click **Build Model** in the **Run on Target** drop-down menu to build the model.

5. Navigate to the folder where Simulink built the model. Double click the file *mcb_pmsm_foc_autotuner_speedgoat.mldatx* to open the Simulink Real-Time Application dialog box.

6. In the Simulink Real-Time Application dialog box, select the target computer to which you are connected. Click **OK** to load the application file to the hardware.

7. Enter these commands (in the same order) at the MATLAB command prompt to execute the loaded application on the hardware and run the motor.

- *tg = slrealtime;*
- *tg.start;*

The model begins field-oriented control (FOC) tuning (using the Field Oriented Control Autotuner block) at the seventeenth second after model execution begins on the hardware. It logs the PI controller gain values (kp_Id, ki_Id, kp_Iq, ki_Iq, kp_speed, ki_speed) in the Simulation Data Inspector.

8. Observe and compare the system response with the PI parameters before tuning and after tuning in the Simulation Data Inspector.
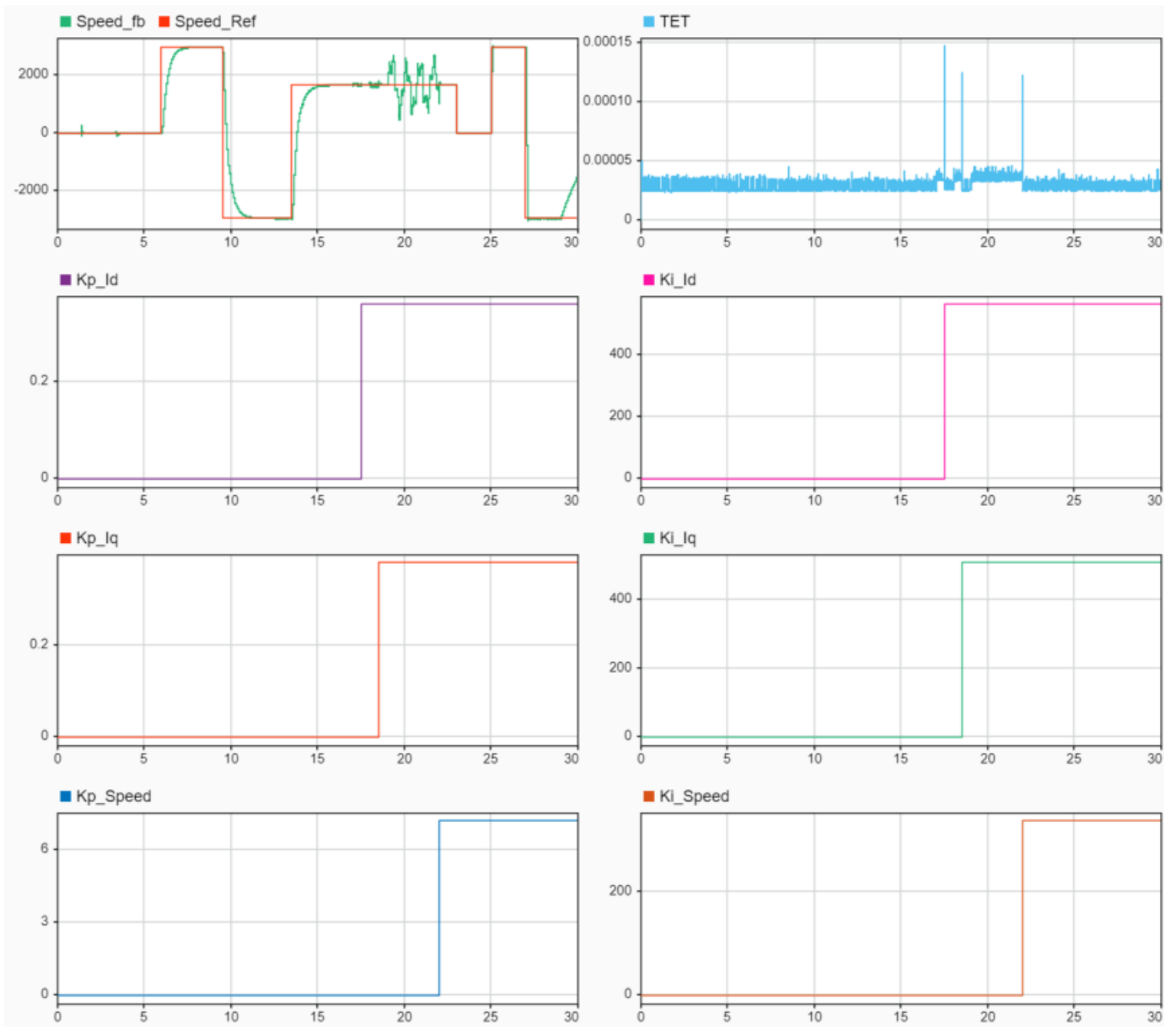
9. If the system response after tuning is satisfactory, update the gain values in the model initialization script associated with the model. For consecutive model executions, you can disable the FOC tuning using the FOC Autotuner button in the model and continue with the closed-loop testing using the new PI parameters.

NOTE: Do not reconfigure or change the reference speed value in the signal builder such that the reference speed changes during the tuning process.

# Six-Step Commutation of BLDC Motor Using Sensor Feedback

This example uses 120-degree conduction mode to implement the six-step commutation technique to control speed and direction of rotation of a three-phase brushless DC (BLDC) motor. The example uses the switching sequence generated by the Six Step Commutation block to control three-phase stator voltages, and therefore, control the rotor speed and direction. For more details about this block, see Six Step Commutation.

The six-step commutation algorithm requires a Hall sequence or a rotor position feedback value (which is obtained from either a quadrature encoder or a Hall sensor).

The quadrature encoder sensor consists of a disk with two tracks or channels that are coded 90 electrical degrees out of phase. This creates two pulses (A and B) that have a phase difference of 90 degrees and an index pulse (I). The controller uses the phase relationship between the A and B channels and the transition of channel states to determine the speed, position, and direction of rotation of the motor.

A Hall effect sensor varies its output voltage based on the strength of the applied magnetic field. According to the standard configuration, a BLDC motor consists of three Hall sensors located electrically 120 degrees apart. A BLDC with the standard Hall placement (where the sensors are placed electrically 120 degrees apart) can provide six valid combinations of binary states: for example, 001,010,011,100,101, and 110. The sensor provides the angular position of the rotor in degrees in the multiples of 60, which the controller uses to determine the 60-degree sector where the rotor is present.

The controller controls the motor by using the Hall sequence or the rotor position. It energizes the next two phases of the stator winding, so that the rotor always maintains a torque angle (angle between rotor d-axis and stator magnetic field) of 90 degrees with a deviation of 30 degrees.
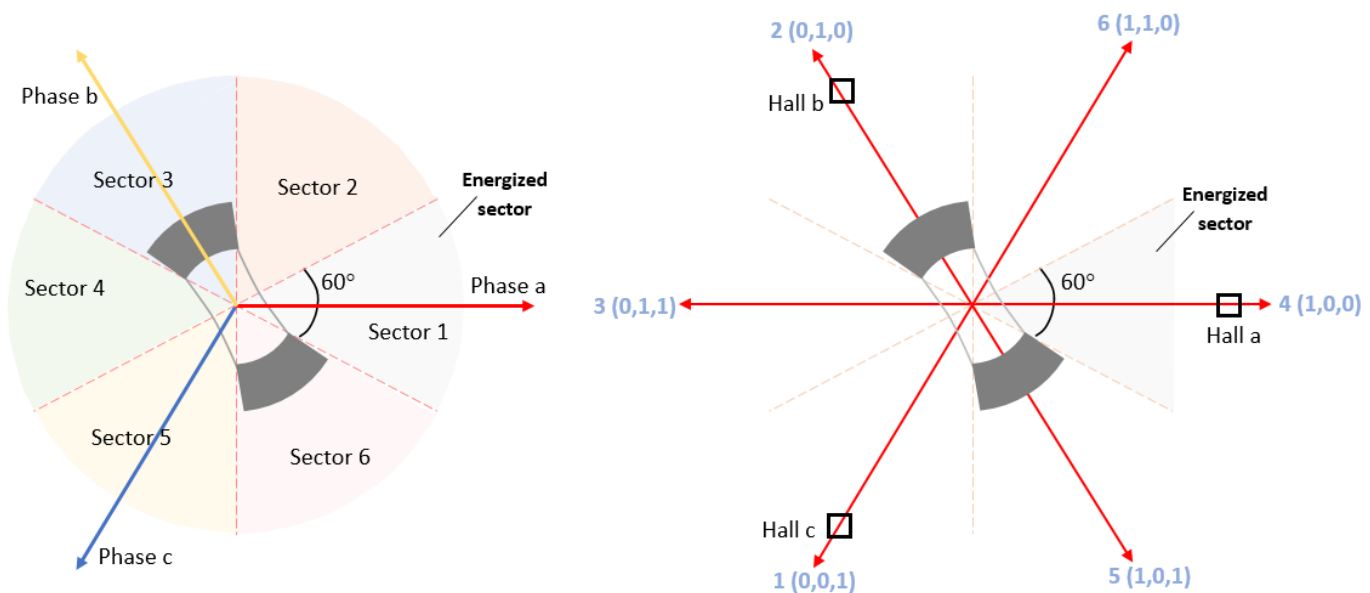
Models

The example includes the model **mcb_bldc_sixstep_f28379d**.

You can use this model for both simulation and code generation. To open the Simulink® model, use the open_system command at the MATLAB command prompt.

```
open_system('mcb_bldc_sixstep_f28379d.slx');
```
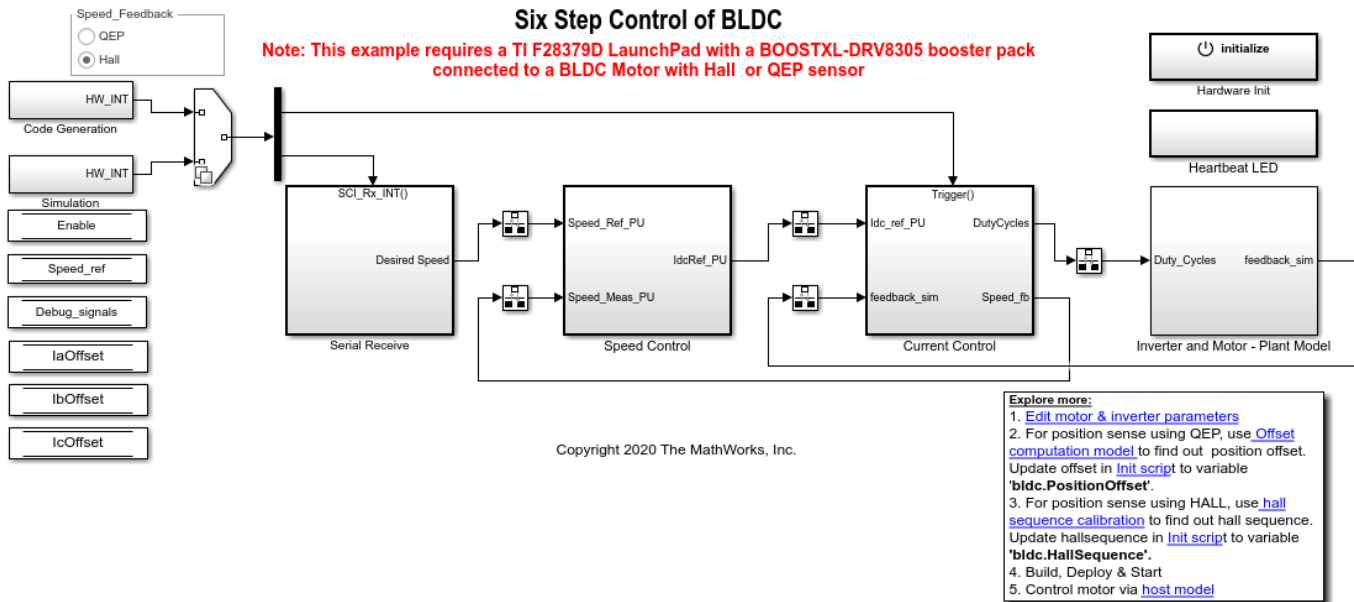

For details of the supported hardware configuration, see Required Hardware in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. We provide default motor parameters with the Simulink model that you can replace with values from either the motor datasheet or other sources.

However, if you have the motor control hardware, you can estimate the parameters for the motor that you want to use by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool" on page 5-2.

The parameter estimation tool updates the *motorParam* variable (in the MATLAB® workspace) with the estimated motor parameters.

2. If you obtain the motor parameters from a motor datasheet or from other sources, update the motor parameters and the inverter parameters in the model initialization script associated with the Simulink models. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

If you use the parameter estimation tool, you can update the inverter parameters, but do not update the motor parameters in the model initialization script. The script automatically extracts the motor parameters from the updated *motorParam* workspace variable.

Simulate Model

This example supports simulation. Follow these steps to simulate the model.

1. Open the model included with this example.

2. Select either the QEP or the Hall Speed_Feedback radio button in the model.

3. Click **Run** on the **Simulation** tab to simulate the model.

4. Click **Data Inspector** on the **Simulation** tab to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section shows you how to generate code and run the FOC algorithm on the target hardware.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board. The host model uses serial communication to command the target Simulink model and run the motor in a closed-loop control.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

• LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_bldc_sixstep_f28379d**

For connections related to the hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Simulate the target model and observe the simulation results.

2. Complete the hardware connections.

3. The model computes the ADC (or current) offset values by default. To disable this functionality, update the value 0 to the variable *inverter.ADCOffsetCalibEnable* in the model initialization script.

Alternatively, you can compute the ADC offset values and update them manually in the model initialization script. For instructions, see "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6.

4. If you are using a quadrature encoder, compute the quadrature encoder index offset value and update it in the model initialization script associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

5. If you are using a Hall sensor, compute the Hall sequence value and update it in the *bldc.hallsequence* variable in the model initialization script associated with the target model. For instructions, see "Hall Sensor Sequence Calibration of BLDC Motor" on page 4-129.

6. Open the target model. If you want to change the default hardware configuration settings for the model, see "Model Configuration Parameters" on page 2-2.

7. Select either the QEP or the Hall Speed_Feedback radio button in the target model.

8. Load a sample program to CPU2 of LAUNCHXL-F28379D. For example, you can use the program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), and ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

9. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

10. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. Use this command for a F28069M based controller.

```
open_system('mcb_bldc_host_model_f28379d.slx');
```
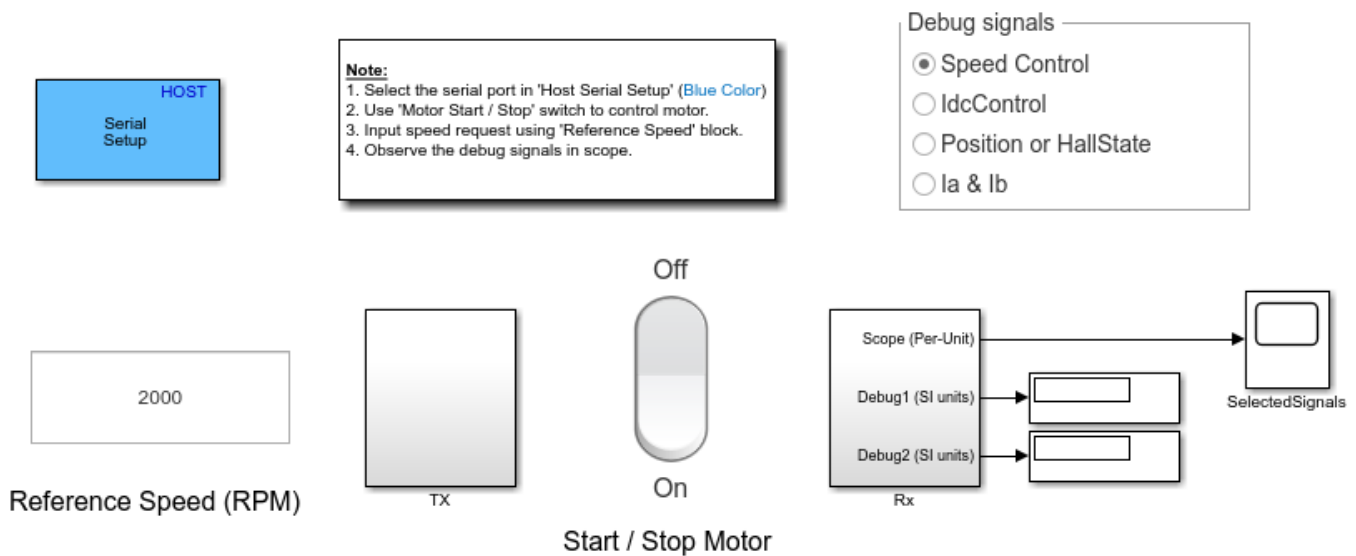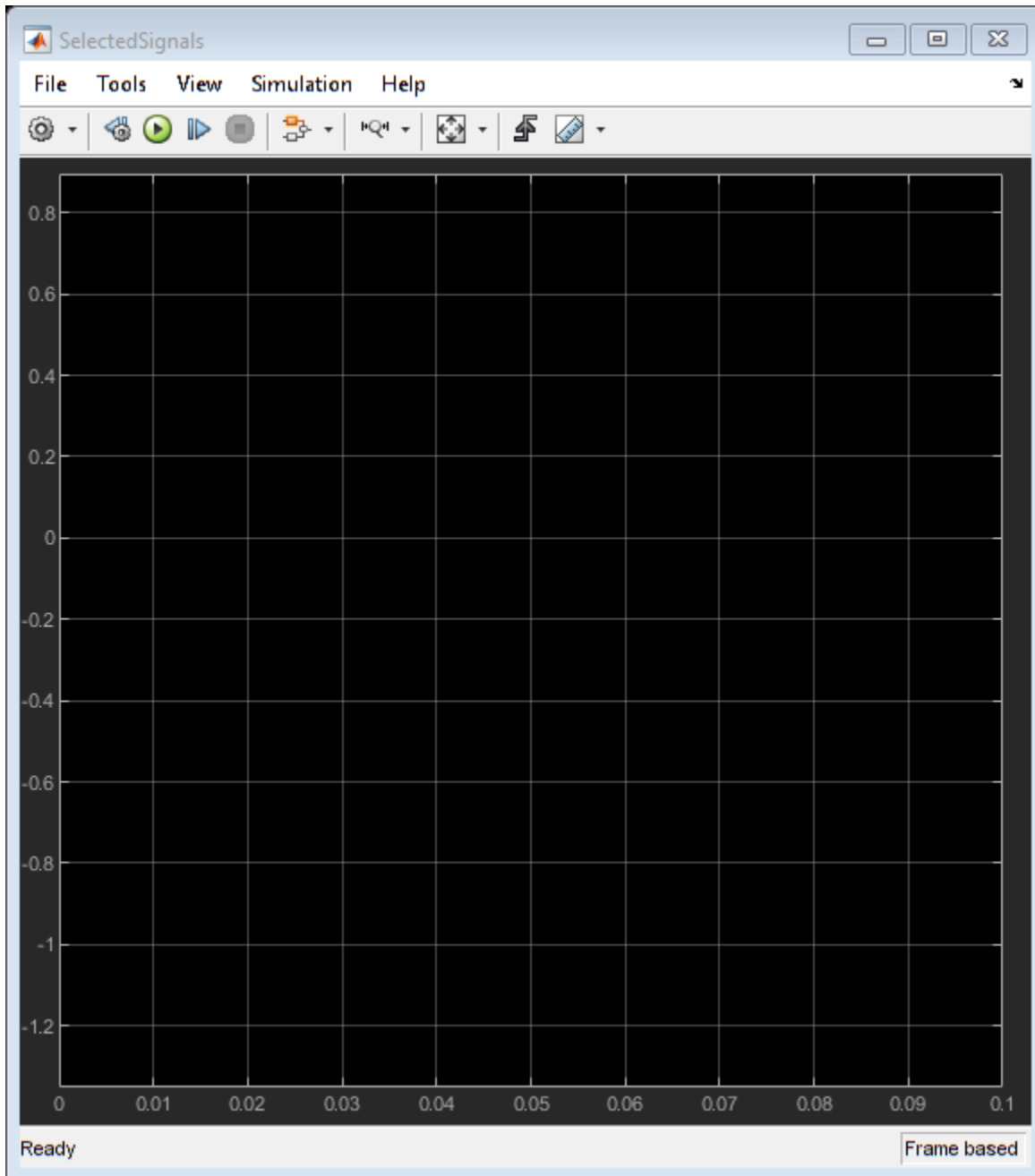
For on the serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

11. In the Host Serial Setup block mask in the host model, select a **Port name**.

12. Update the reference speed value in the *Reference Speed (RPM)* field in the host model.

13. In the host model, select the debug signals that you want to monitor.

14. Click **Run** on the **Simulation** tab to run the host model.

15. Change the position of the Start / Stop Motor switch to On, to start running the motor.

16. Observe the debug signals from the RX subsystem, in the Scope and Display blocks in the host model.

Hall Sensor Sequence Calibration of BLDC Motor

This example calculates the Hall sensor sequence with respect to position zero of the rotor in open-loop control.

A Hall effect sensor varies its output voltage based on the strength of the applied magnetic field. According to the standard configuration, a brushless DC (BLDC) consists of three Hall sensors located electrically 120 degrees apart. A BLDC motor with the standard Hall placement (where the sensors are placed electrically 120 degrees apart) can provide six valid combinations of binary states: for example, 001,010,011,100,101, and 110. The sensor provides the angular position of the rotor in degrees in the multiples of 60, which the controller uses to determine the 60-degree sector where the rotor is present.

The target model runs the motor at a low speed (10 RPM) in open loop and performs V/f control on the motor. At this speed, the d-axis of the rotor closely aligns with the rotating magnetic field of the stator.

When the rotor reaches the open-loop position zero, it aligns with the phase a-axis of the stator. At this position (corresponding to a Hall state), the six-step commutation algorithm energizes the next two phases of the stator winding, so that the rotor always maintains a torque angle (angle between rotor d-axis and stator magnetic field) of 90 degrees with a deviation of 30 degrees.

The Hall sequence calibration algorithm drives the motor over a full mechanical revolution and computes the Hall sensor sequence with respect to position zero of the rotor in open-loop control.

Note: This example works for all motor-phase or Hall sensor connections.
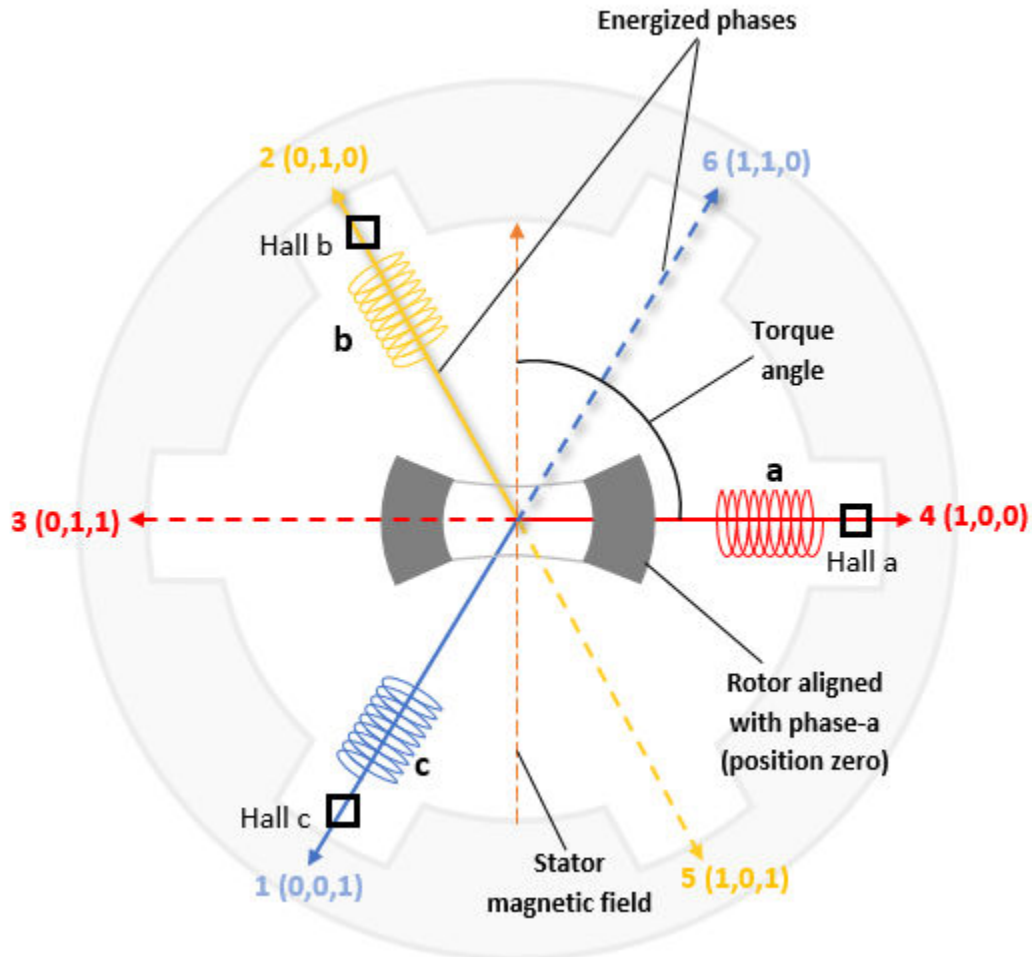
Models

The example includes the model **mcb_hall_calibration_f28379d**.

You can use this model only for code generation. To open the Simulink® model, use the open_system command at the MATLAB® command prompt.
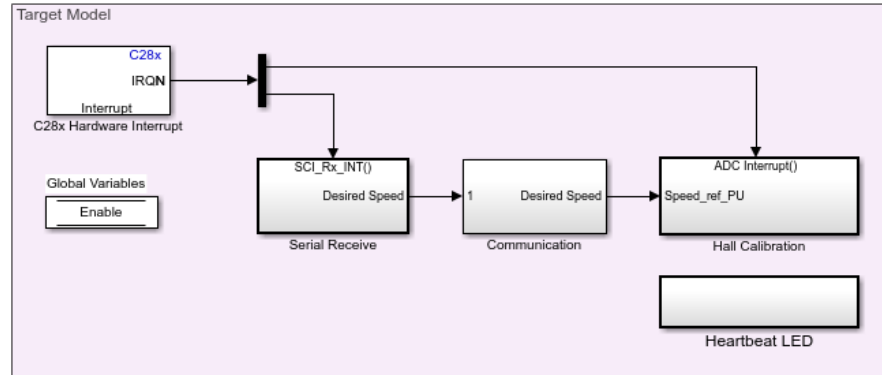
```
open_system('mcb_hall_calibration_f28379d.slx');
```

Steps:
1. Enter parameters in the Configuration panel.
2. Click **Build, Deploy & Start** in the **Hardware** tab.
3. Perform calibration by using host model.
4. If the motor does not start or rotate smoothly, increase **Vd Ref in Per Unit voltage** (that can have a maximum value of 1) in the Configuration panel.
5. If the current drawn by the connected motor is too high, reduce the value mentioned in step 4.

Configuration

Number of Pole Pairs	4
PWM Frequency [Hz]	20000
Data type for control algorithm	single
Motor Base Speed [rpm]	4000
Vd Ref in Per Unit voltage	0.1

Hall Sequence Calibration of 3-phase motors

Note: This example requires a TI F28379D LaunchPad with a BOOSTXL-DRV8305 booster pack

Copyright 2020 The MathWorks, Inc.

For details on the supported hardware configuration, see Required Hardware in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

- Motor Control Blockset™
- Embedded Coder®
- Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors
- Fixed-Point Designer™ (only needed for optimized code generation)

Generate Code and Deploy Model to Target Hardware

This section shows you how to generate code and run the motor by using open-loop control.

This example uses a host and a target model. The host model is a user interface to the controller hardware board. You can run the host model on the host computer. The prerequisite to use the host model is to deploy the target model to the controller hardware board.

The host model uses serial communication to command the target model and run the motor in an open-loop configuration by using V/f control. The host model displays the calculated Hall sensor sequence.

Required Hardware

This example supports this hardware configuration. Use the target model name (highlighted in bold) to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: **mcb_hall_calibration_f28379d**

For connections related to the hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-5.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. Open the target model for the hardware configuration that you want to use. If you want to change the default hardware configuration settings for the target model, see "Model Configuration Parameters" on page 2-2.

3. Update these motor parameters in the **Configuration** panel of the target model.

- **Number of pole pairs**
- **PWM frequency [Hz]**
- **Data type for control algorithm**
- **Motor base speed**
- **Vd Ref in per-unit voltage**

4. Load a sample program to CPU2 of LAUNCHXL-F28379D. For example, you can use the program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx), and ensures that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

5. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware.

6. Click the **host model** hyperlink in the target model to open the associated host model. You can also use the open_system command to open the host model. Use this command for a F28379D based controller:

```
open_system('mcb_hall_calibration_host_f28379d.slx');
```

Hall Sequence Calibration Host
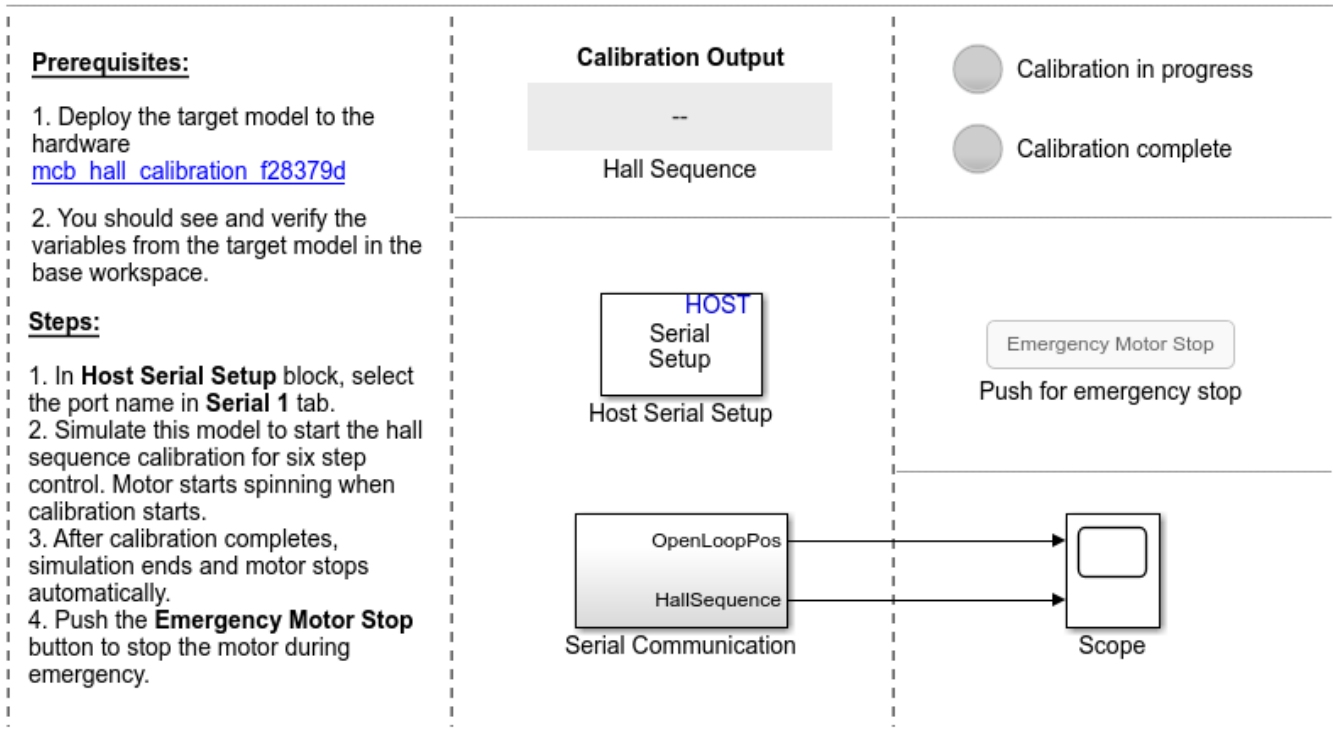
Copyright 2020 The MathWorks, Inc.

For details on serial communication between the host and target models, see "Host-Target Communication" on page 6-2.

You can use the Scope in the host model to monitor the open-loop rotor position and Hall sequence values.

7. In the Host Serial Setup block mask in the host model, select a **Port name**.

8. Click **Run** on the **Simulation** tab to run the host model and start Hall sequence calibration for six-step commutation control. The motor runs and calibration begins when you start simulation. After the calibration process is complete, simulation ends and the motor stops automatically.

Note: If the motor does not start or rotate smoothly, increase the value of the **Vd Ref in Per Unit voltage** field (maximum value is 1) in the **Configuration** panel. However, if the motor draws high current, reduce this value.

As a convention, six-step commutation control uses a forward direction of rotation that is identical to the direction of rotation used during Hall sequence calibration. To change the forward direction convention, interchange the motor phase wires, perform Hall sequence calibration again, and then run the motor by using six-step commutation control.
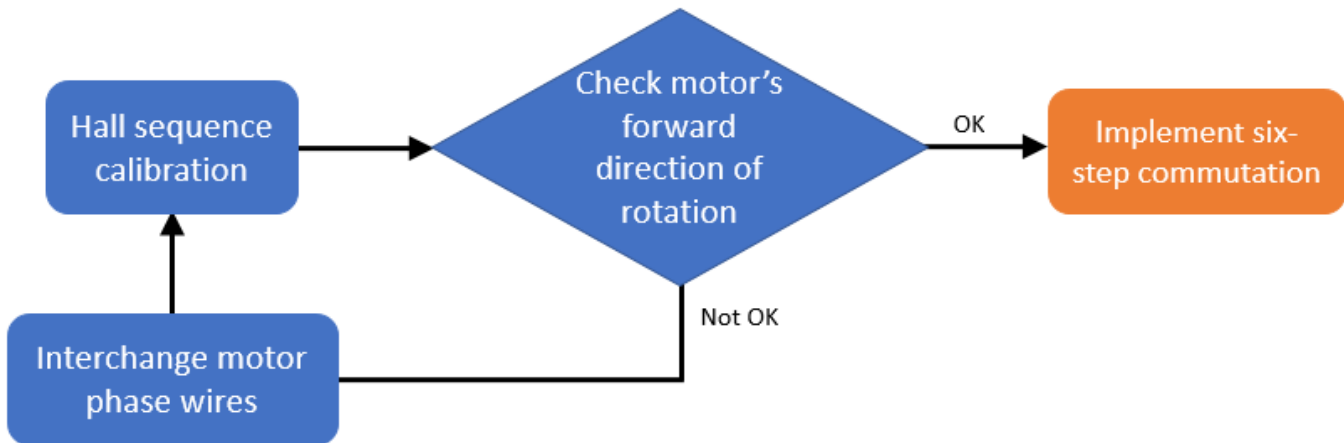
9. See these LEDs on the host model to know the status of calibration process:

- The **Calibration in progress** LED turns orange when the motor starts running. Notice the rotor position and the variation in the Hall sequence value in the Scope (the position signal indicates a ramp signal with an amplitude between 0 and 1). After the calibration process is complete, this LED turns grey.

- The **Calibration complete** LED turns green when the calibration process is complete. Then the *Calibration Output* field displays the computed Hall sequence value.

Note: This example does not support simulation.

To immediately stop the motor during an emergency, click the **Emergency Motor Stop** button.

For examples that use six-step commutation using a Hall sensor, update the computed Hall sequence value in the *bldc.hallsequence* variable in the model initialization script linked to the example. For instructions, see "Estimate Control Gains from Motor Parameters" on page 3-2.

Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool

Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool

Motor Control Blockset provides a parameter estimation tool that estimates the motor parameters accurately. Use the estimated motor parameters to simulate the motor model and design the control system. Therefore, the simulation response with the estimated parameters for the motor model is close to the behavior of motor under test.

The parameter estimation tool determines these motor parameters for a Permanent Magnet Synchronous Motor:

- Phase resistance (R_s)
- d and q axis inductances (L_d and L_q)
- Back-EMF constant (K_e)
- Motor inertia (J)
- Friction constant (F)

The parameter estimation tool accepts the minimum required inputs, runs tests on the target hardware, and displays the estimated parameters.

Prerequisites

The parameter estimation tool needs the motor position detected by either quadrature encoder, Hall sensors, or sensorless flux observer. Sensored position detection requires either Hall or quadrature encoder sensor calibration for the motor under test.

- Ensure that the PMSM is in no-load condition.

If you are using Hall sensors:

- Ensure that the PMSM has Hall sensors.
- Calibrate the Hall sensor offset. For instructions, see "Hall Offset Calibration for PMSM Motor" on page 4-72.

If you are using quadrature encoder sensor:

- Ensure that the PMSM has quadrature encoder sensor.
- Calibrate the quadrature encoder offset. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82.

Note If you set **Sensor Selection** field in the host model to `Sensorless`, you can skip the position sensor calibration step.

Supported Hardware

This example supports only these hardware configurations:

Texas Instruments™ F28069M control card configuration:

- F28069M control card
- DRV8312-69M-KIT inverter
- A PMSM with Hall or quadrature encoder sensor
- DC power supply

Note The DRV8312-69M-KIT board has a known issue in the board's power supply section. Due to this limitation, the board does not support all Hall sensor types. For example, it does not support the Hall sensor of Teknic M-2310P motor.

Texas Instruments LAUNCHXL-F28379D configuration:

- LAUNCHXL-F28379D controller
- BOOSTXL-DRV8305 inverter
- A PMSM with Hall or quadrature encoder sensor
- DC power supply

Required MathWorks Products

To run parameter estimation, you need these products:

- Motor Control Blockset
- Fixed-Point Toolbox™

Only to build the target models, you need these optional products :

- Embedded Coder
- Embedded Coder Support Package for Texas Instruments C2000™ Processors

Prepare Hardware

For the F28069M control card configuration:

1   Connect the F28069M control card to J1 of DRV8312-69M-KIT inverter board.

2   Connect the motor three phases to MOA, MOB, and MOC on the inverter board.

3   Connect the DC power supply to PVDDIN on the inverter board.

4   If you are using Hall sensor, connect the Hall sensor encoder output to J10 on the inverter board.

5   If you are using quadrature encoder sensor, connect the quadrature encoder pins (G, I, A, 5V, B) to J4 on the inverter board.

For the LAUNCHXL-F28379D configuration:

1   Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J1, J2 of LAUNCHXL.

2   Connect the motor three phases to MOTA, MOTB, and MOTC on the BOOSTXL inverter board.

3   Connect the DC power supply to PVDD and GND on the BOOSTXL inverter board.

4   If you are using Hall sensor, connect the Hall sensor output to QEP_B (configured as eCAP) on LAUNCHXL.

5   If you are using quadrature encoder sensor, connect the quadrature encoder pins (G, I, A, 5V, B) to QEP_A on the LAUNCHXL controller board.

For more details regarding these connections, see "Hardware Connections" on page 7-2.

For more details regarding the model settings, see "Model Configuration Parameters" on page 2-2.

For LAUNCHXL-F28379D, load a sample program to CPU2, for example, program that operates the CPU2 blue LED using GPIO31 (`c28379D_cpu2_blink.slx`) to ensure that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.

Parameter Estimation Tool

The parameter estimation tool includes a target model and a host model. The models communicate with each other by using a serial communication interface. For more details, see "Host-Target Communication" on page 6-2.

Enter the system details about the motor under test in the host model. The target model uses an algorithm to perform tests on the motor and estimate the motor parameters. The host model starts these tests and displays the estimated parameters.

Prepare Workspace

To open the parameter estimation host model, enter this command:

```
open_system('mcb_param_est_host_read.slx');
```
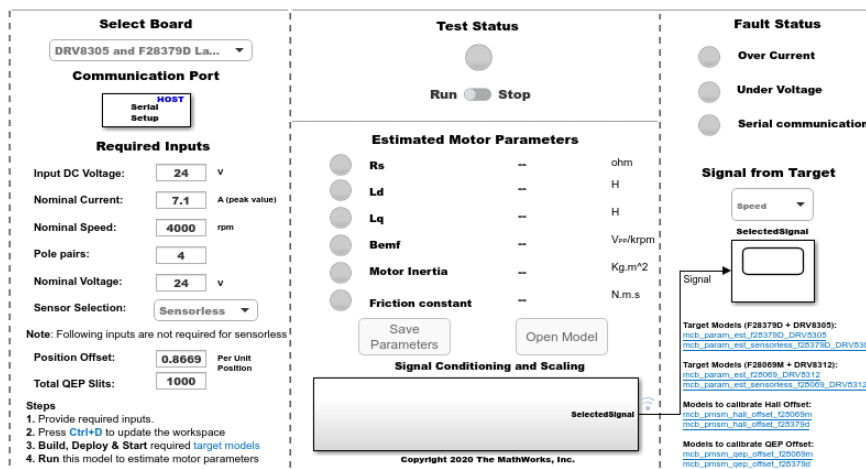

Enter these details in the host model to prepare the workspace:

- **Select Board** - Select the target hardware and inverter combination.
- **Communication Port** - Specify the serial port that you want to configure. Select an available port from the list. For more details, see "Find Communication Port" on page 6-4.
- **Required Inputs** - Enter these motor specification data. You can obtain these values either from the motor data sheet or on the motor nameplate.
  - **Input DC Voltage** - The DC supply voltage for the inverter (Volts).

- **Nominal Current** - The rated current of the motor (Ampere).
- **Nominal Speed** - The rated speed of the motor (rpm).
- **Pole Pairs** - The number of pole pairs of the motor.
- **Nominal Voltage** - The rated voltage of the motor (Volts).
- **Position Offset** - The position (Hall or quadrature encoder) sensor offset value (per-unit position) (see "Hall Offset Calibration for PMSM Motor" on page 4-72, "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82, and "Per-Unit System" on page 6-15).
- **Sensor Selection** - The type of position sensor that you are using. You can select one of these values:

  - QEP - Select this option if you are using the quadrature encoder sensor attached to your motor.
  - HALL - Select this option if you are using the Hall sensors available in your motor.
  - Sensorless - Select this option if you want to use the Flux Observer sensorless position estimation block instead of a position sensor. For details about this block, see Flux Observer.

- **Total QEP Slits** - The number of slits available in the quadrature encoder sensor. By default, this field has a value 1000.

Note When updating **Required Inputs**, consider these limitations:

- The rated speed of the motor must be less than 25000rpm.
- The tests protect the hardware from over-current faults. However, to ensure that these faults do not occur, keep the motor's rated current (entered in Nominal Current field) less than the maximum current supported by the inverter.
- If you have an SMPS based DC power supply unit, set a safe current limit on the power supply for safety reasons.

Deploy Target Models

Before starting the tests by using the parameter estimation tool, you should download the binary files (.hex/ .out) generated by the target model into the target hardware. There are two workflows to download the binary files:

Workflow 1: Build and Deploy Target Model:

Use this workflow to generate and deploy the code for the target model. Ensure that you press **Ctrl +D** to update the workspace with the required input details from the host model.

Click one of these hyperlinks in the parameter estimation host model to open the target model (for the hardware that you are using):

- For F28069M based controller attached to either Hall or quadrature encoder sensor:

  mcb_param_est_f28069_DRV8312
- For F28379D based controller attached to either Hall or quadrature encoder sensor:

  mcb_param_est_f28379D_DRV8305

- For F28379D based controller that uses the sensorless Flux Observer block:

  mcb_param_est_sensorless_f28379D_DRV8305

Click **Build, Deploy & Start** in the **Hardware** tab to deploy the target model to the hardware.

Note Ignore the warning message `Multitask data store option in the Diagnostics page of the Configuration Parameter Dialog is none` displayed by the model advisor, by clicking the **Always Ignore** button. This is part of the intended workflow.
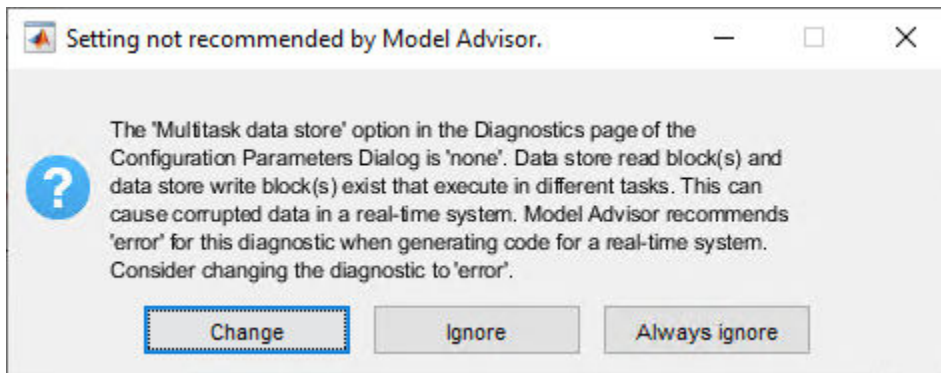
Workflow 2: Manually Download Target Model:

Use this workflow to deploy the binary files (`.hex`/ `.out`) of the target model manually by using a third party tool (the workflow does not need code-generation). This workflow is only valid for Teknic M-2310P motor.

- Locate the binary files (`.hex`/ `.out`) at these locations:

  - `<matlabroot>\toolbox\mcb\mcbexamples\mcb_param_est_f28069_DRV8312.out`
  - `<matlabroot>\toolbox\mcb\mcbexamples\mcb_param_est_f28379D_DRV8305.out`
- Open a third-party tool to deploy the binary files (`.hex`/ `.out`).
- Download and run the binary files (`.hex`/ `.out`) on the target hardware.

Estimate Motor Parameters

Use the following steps to run the Motor Control Blockset parameter estimation tool:

1. Ensure that you deploy the binary files (`.hex`/ `.out`) generated from the target model, to the target hardware and update the required details in the host model.
2. In the host model, click **Run** in the **Simulation** tab to run the parameter estimation tests.
3. The parameter estimation process takes less than a minute to perform the tests. You can ignore the beep sound produced during the tests.
4. The host model displays the estimated motor parameters after successfully completing the tests.

The tool uses the following algorithm to estimate parameters:

- Motor resistance (R) - The tool uses Ohm's law to estimate this value.
- Motor inductance (L_d and L_q) - The tool uses frequency injection method to estimate these values.
- Back EMF (K_e) - The tool measures the currents and voltages and uses the electric motor equation to estimate this value.
- Permanent magnet flux (λ) - The tool uses the estimated back EMF constant to estimate this value.
- Friction constant (B) - The tool estimates this value by using the torque equation for a motor running at a constant speed.
- Inertia (J) - The tool estimates this value by using retardation test.
- Rated Torque - The tool estimates this value by using the estimated value of permanent magnetic flux of the motor.

When the parameter estimation tests complete, the **Test Status** LED turns green.

If the tests are interrupted, the **Test Status** LED turns red. When the LED turns red, run the host model again to rerun the parameter estimation tests.

During an emergency, you can manually turn the **Run-Stop** slider switch to **Stop** position to stop the parameter estimation tests. In addition, the model interrupts the parameter estimation tests and turns these LEDs red to protect the hardware from the following faults:
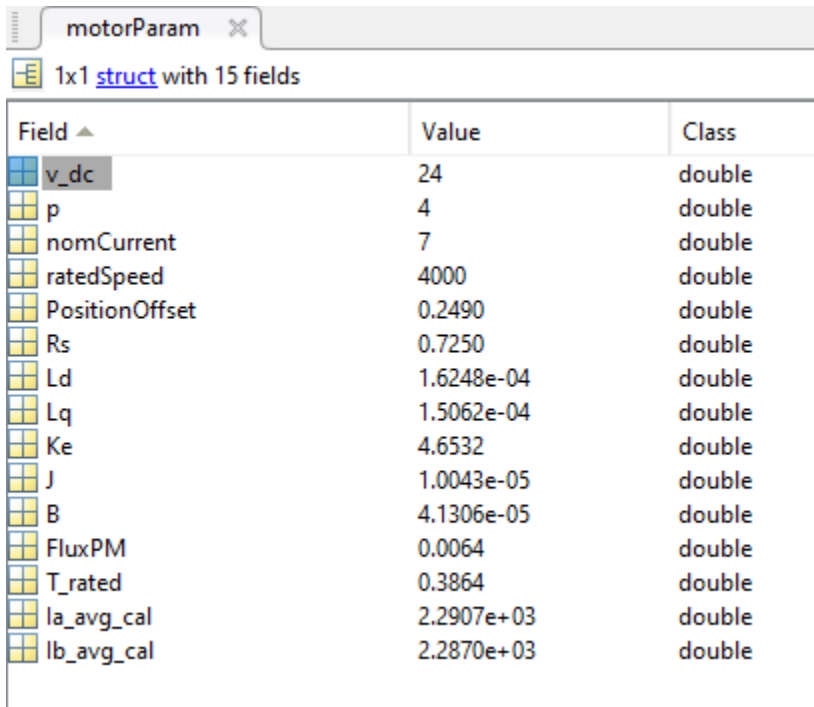
1. Over-current fault (this fault occurs when actual current drawn from the power supply is more than the **Nominal Current** value mentioned in the **Required Inputs** section of the host model)
2. Under-voltage fault (this fault occurs when input DC voltage drops below 80% of the **Input DC Voltage** value mentioned in the **Required Inputs** section of the host model)
3. Serial communication fault

Save Estimated Parameters

You can export the estimated motor parameters and further use them for the simulation and control system design.

To export, click **Save Parameters** to save the estimated parameters into a MAT (`.mat`) file.

To view the saved parameters, load the MAT (`.mat`) file in the MATLAB workspace. MATLAB saves the parameters in a structure named `motorParam` in the workspace.

motorParam ✕		

📇 1x1 struct with 15 fields

| Field ▲	Value	Class
v_dc	24	double
p	4	double
nomCurrent	7	double
ratedSpeed	4000	double
PositionOffset	0.2490	double
Rs	0.7250	double
Ld	1.6248e-04	double
Lq	1.5062e-04	double
Ke	4.6532	double
J	1.0043e-05	double
B	4.1306e-05	double
FluxPM	0.0064	double
T_rated	0.3864	double
la_avg_cal	2.2907e+03	double
lb_avg_cal	2.2870e+03	double

Click **Open Model** to create a new Simulink model with a PMSM motor block. The motor block uses the `motorParam` structure variables from the MATLAB workspace.

Concepts

- "Host-Target Communication" on page 6-2
- "Open-Loop and Closed-Loop Control" on page 6-8
- "Current Sensor ADC Offset and Position Sensor Calibration" on page 6-12
- "Per-Unit System" on page 6-15

Host-Target Communication

Motor Control Blockset uses a communication interface between the host model and the target model to control the motor and observe feedback.
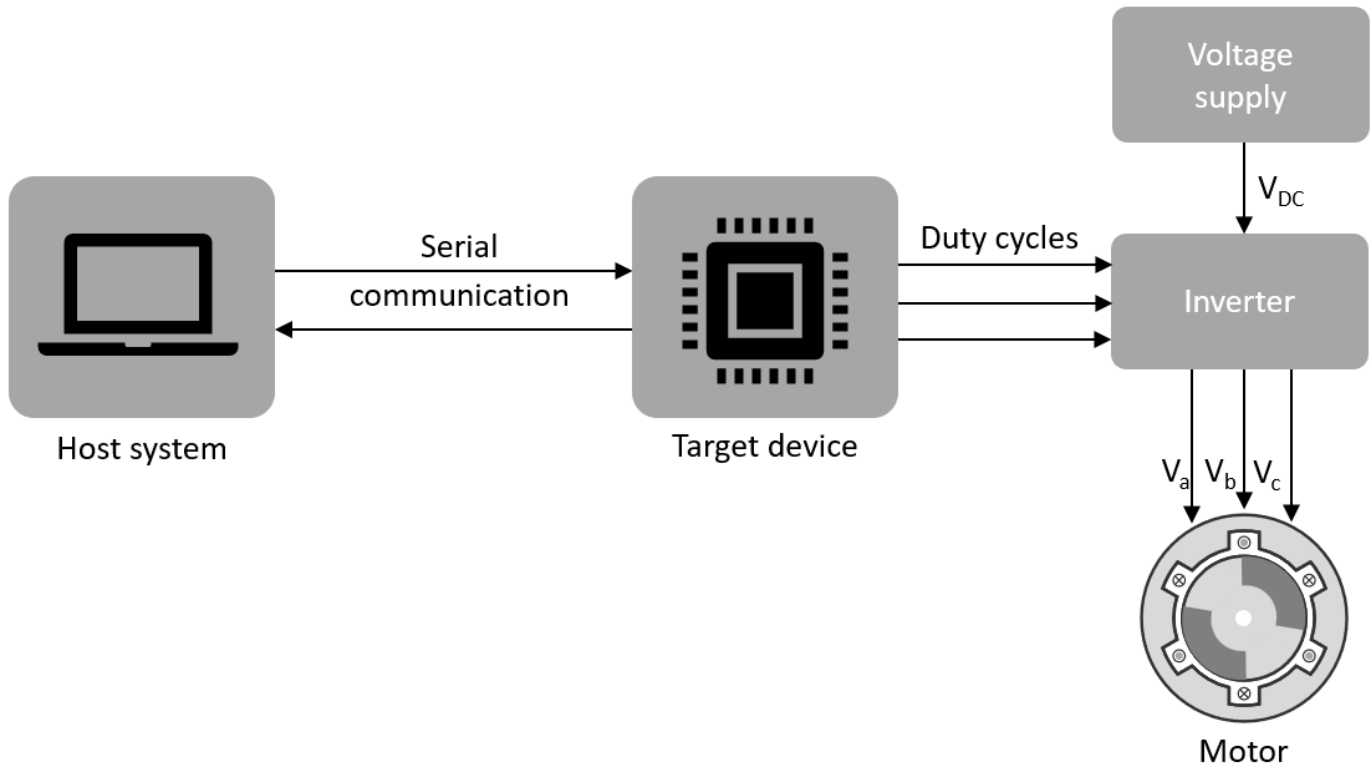
Host Model

The host model is a user interface for the controller hardware board. Run the host model on the host computer. Before you run the host model on the host computer, make sure to deploy the target model on the controller hardware board.

The host model commands, controls, and exchanges data with the target hardware. You can perform these operations using the host model available in the Motor Control Blockset:

- Find the serial communication port (COM port) in the host system. For more details, see Find Communication Port section in this page.
- Configure the serial port and baud rate by using the Serial Setup block.
- Start or stop the motor.
- Specify the motor speed.
- View the debug or output signals that the host receives from the target by using the Time Scope and Display blocks.

Target Model

The target model runs on the controller hardware board. Deploy the target model to the embedded target hardware that controls the motor. The target model communicates with the host model to

receive commands from the user (for example, the command to start or stop the motor). Some common operations that a target model available in Motor Control Blockset performs:

- Serial communication with the host model to receive user commands and exchange binary data.
- Read data from the position and current sensors attached to the motor and inverter.
- Control motor speed and torque by running the control algorithms and processing the feedback.
- Generate duty cycle inputs for the inverter.
- Enable fast serial data monitoring for debugging the signals.

Serial Communication Blocks

The host and target models interact by using these Motor Control Blockset blocks that enable serial communication:

- Host Serial Receive
- Host Serial Setup
- Host Serial Transmit

Using these blocks you can monitor, control, and customize the motor operation in real time. For example, you can view the debug signals, stop or start the motor, and change the motor speed without repeated deployment of the target model.

Fast Serial Data Monitoring

The Motor Control Blockset example models use the fast serial data monitoring algorithm, which performs control and diagnostic operations through the host model. This algorithm enables you to observe data from the target device at the same rate as the execution sample time (for example, PWM frequency of 20kHz). This, in turn, helps in diagnostics and analysis of transients.

Evaluation boards often provide serial communication over USB connections that enable fast serial transfers. The models running on the Texas Instruments LaunchPad hardware boards send signals like I_a and I_b currents over the serial interface. Use the host model to receive these signals on your host computer. The Motor Control Blockset examples implementing Field Oriented Control (FOC) algorithm for the F28379D LaunchPad use mcb_pmsm_foc_host_model_f28379d.slx. Examples that implement the FOC algorithm for the F28069M targets, use mcb_pmsm_foc_host_model_f28069m.slx. The Motor Control Blockset also provides other host models for the application-based examples.

Selecting COM port and baud rate

Select the appropriate COM port that matches your board in the Serial Setup block of the host model. Adjust the baud rate for your board:

| Texas Instruments LaunchPad	Baud Rate
F28027 LaunchPad	3.75e6
F28069 LaunchPad	5.625e6
F28377S LaunchPad	12e6
F28379D LaunchPad	12e6

After you deploy the target model on the target device, run the host model and observe the debug signals update at 20 kHz, on the time scope. You can use the same technique to monitor other signals on other processors.

Note SCI_A is usually connected to the FTDI chip that allows serial transfers over USB on the LaunchPad boards, docking stations, and ISO control cards.

Find Communication Port

Use these steps to find the serial communication port in the Device Manager of Windows PC, after you connect the target hardware to your system:

1 Open **Device Manager** on your Windows PC.
2 Look for an entry under **Ports (COM & LPT)** titled **USB Serial Port (COMX)**, where X is a number. You can note down this number to configure the serial setup block in the host model.
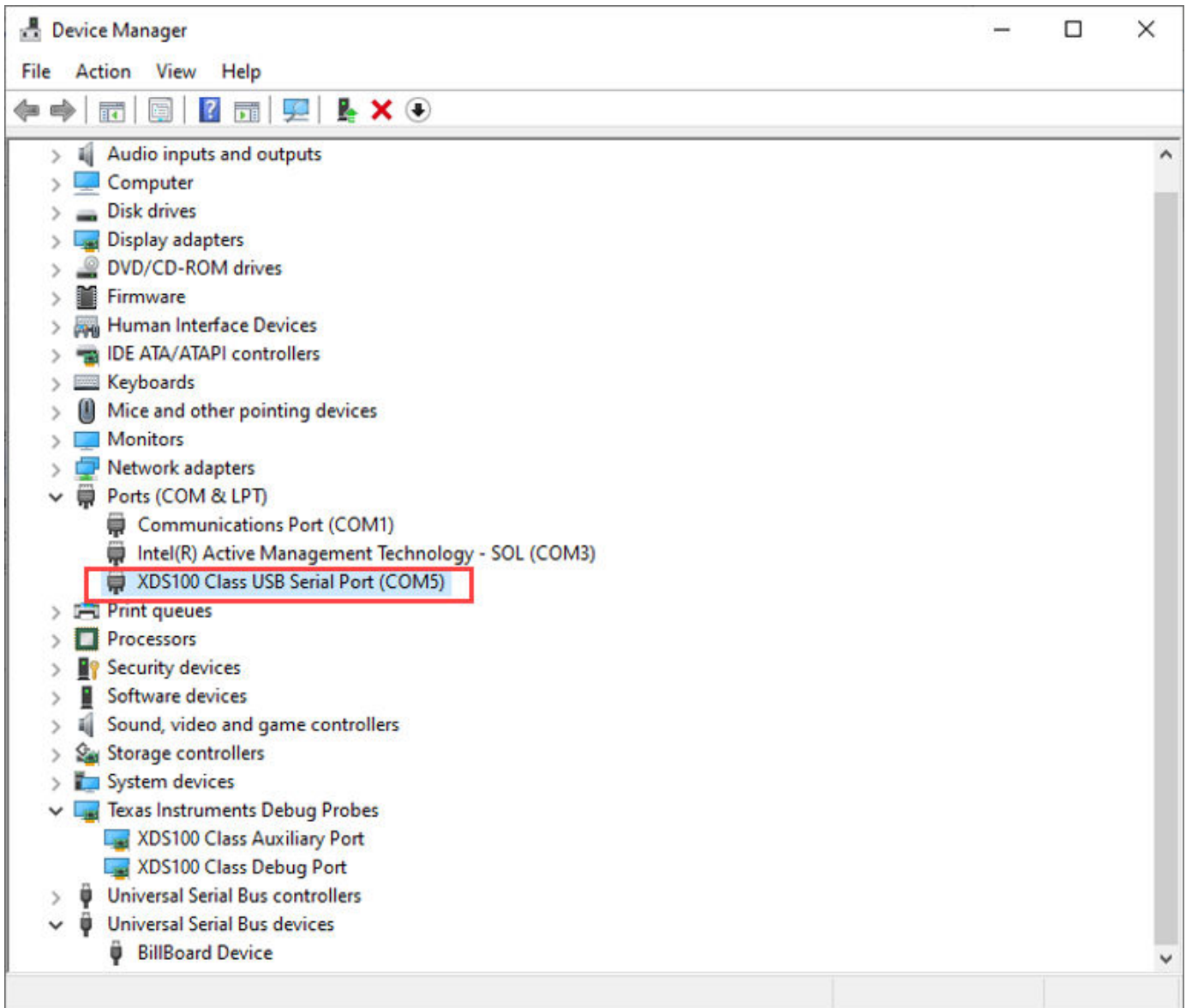
If you face difficulty in finding the COM port, follow these steps to determine the COM port:

1   Open **Device Manager** on your Windows PC.

2   Look for an entry under **Ports (COM & LPT)** titled **USB Serial Port (COMX)**, where X is a number. If there are multiple COM ports, you can disconnect and reconnect the C2000 board and observe the updates in Device Manager to determine the COM port.

3   Alternatively, follow these steps to determine the correct port name for the connected target hardware:

    a   Right-click a communication port and click **Properties**.

    b   In the **Details** tab, select **Hardware Ids** property.

    c   If the port indicates the following IDs, the communication port belongs to the connected TI's C2000™ controller hardware board:

- VID: 0403
- PID: A6D0

4 If you do not see or find the right port in **Ports (COM & LPT)**, navigate to **Texas Instruments Debug Probes** and follow these steps:

a Right-click **XDS100 Class Auxiliary Port Properties** and select **Properties**. Navigate to **Advanced** tab and select **Load VCP**.

b Right-click **XDS100 Class Debug Port Properties** and select **Properties**. Navigate to **Advanced** tab and clear **Load VCP**.

c Disconnect and reconnect the USB cable to the system and observe the updates in Device Manager to determine the COM port. The system now displays the COM port that belongs to the connected TI's C2000 controller hardware board.

Tip VCP stands for Virtual COM Port (for devices that support serial over USB communication).
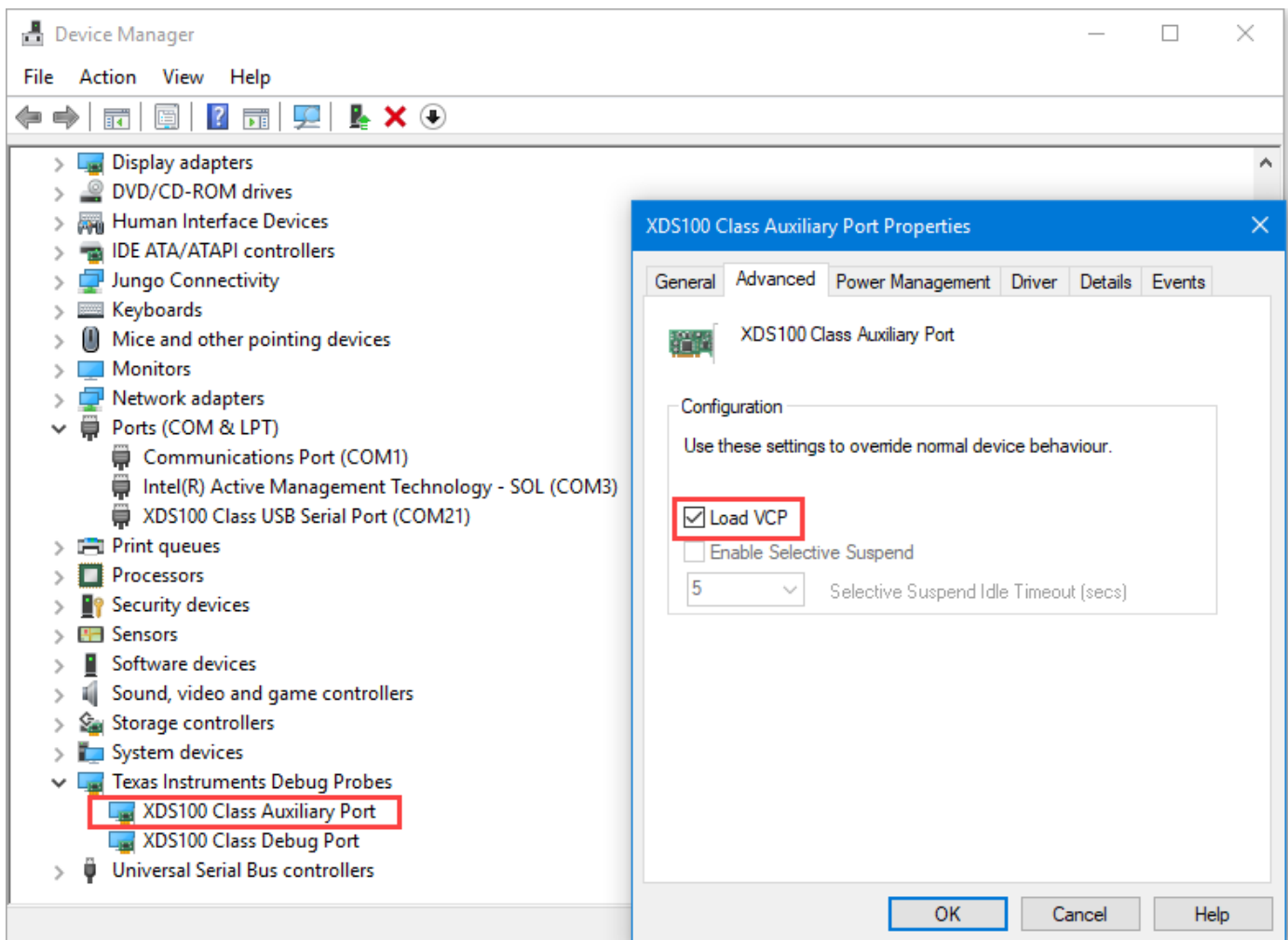
5 If **Texas Instruments Debug Probes** do not appear in the Device Manager, expand **Universal Serial Bus controllers** in the Device Manager and follow these steps:

    a   Right-click **TI XDS 100 Channel B** and select **Properties**. Navigate to **Advanced** tab and select **Load VCP**.

    b   Right-click **TI XDS 100 Channel A** and select **Properties**. Navigate to **Advanced** tab and clear **Load VCP**.

    c   Disconnect and reconnect the USB cable to the system and observe the updates in Device Manager to determine the COM port. The system now displays the COM port that belongs to the connected TI's C2000 controller hardware board.

6   If Device Manager does not detect the target hardware, follow these steps:

    a   Check that the target hardware is connected to the system.

    b   Check if the device drivers are installed correctly. Generally, device drivers are installed with the Code Composer Studio™(CCS). Check if the CCS software is installed on your system. Alternatively, try re-installing the device drivers suggested by Texas Instruments.

    c   Check if the serial connection cable is intact.

    d   If the problem persists, try connecting the hardware to another system and check if Device Manager detects the hardware.

    e   If you still face the problem, the target hardware may be faulty.

Open-Loop and Closed-Loop Control

This section describes the open-loop and closed loop motor control techniques.

Open-Loop Motor Control

Open-loop control (also known as scalar control or Volts/Hz control) is a popular motor control technique that you can use to run any AC motor. This is a simple technique that does not need any feedback from the motor. To keep the stator magnetic flux constant, we keep the supply voltage amplitude proportional to its frequency.
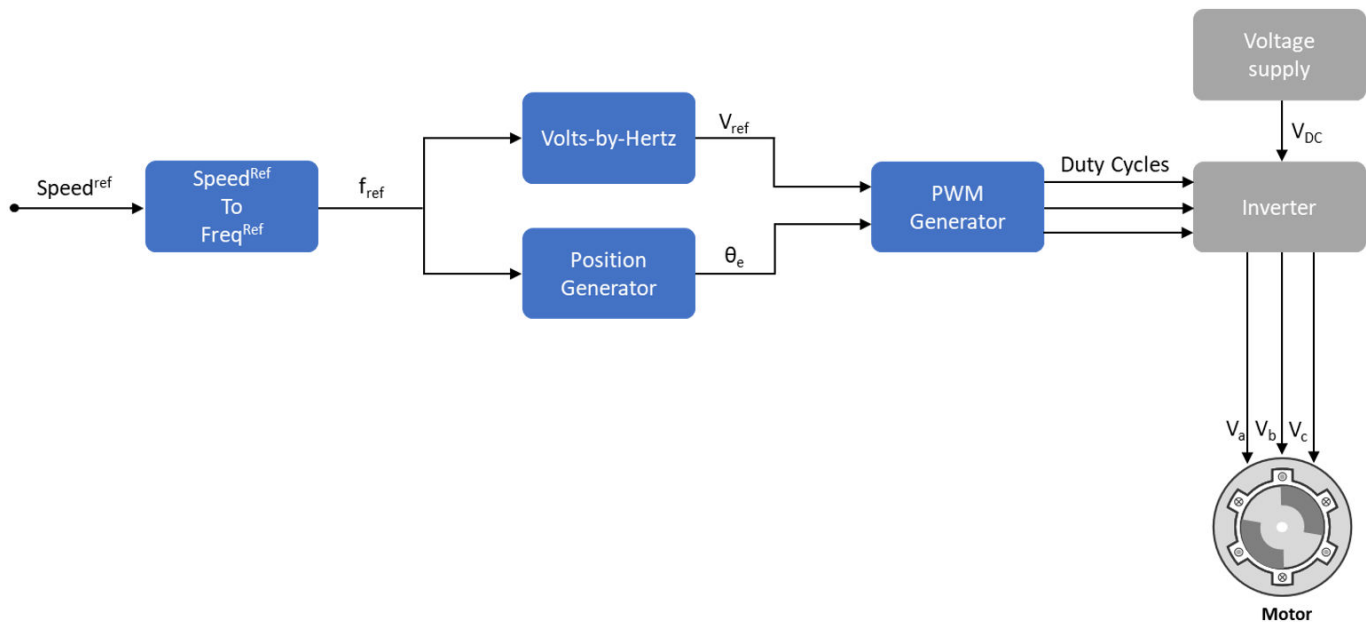
This figure shows an open-loop control system. The power circuit consists of a PWM voltage fed inverter supplied by a DC source. The system does not use any feedback signal for control implementation. It uses the reference speed to determine the frequency of the stator voltages. The system computes the voltage magnitude as proportional to the ratio of rated voltage and rated frequency (commonly known as Volts/Hz ratio), so that the flux remains constant.

$$\lambda_m \propto {}^{V_s}\!/\!_{f_s}$$

where:

1. λ_m is the rated flux of the motor in Wb.
2. V_s is the stator voltage of the AC motor in Volts.
3. f_s is the frequency of the stator voltage of the AC motor in Hz.

In an open-loop system, the speed for an AC motor is expressed as:

$$Speed_{(rpm)} = \frac{60 \times f_s}{p}$$

where:

- *Speed*$_{(rpm)}$ is the mechanical speed of the AC motor in rpm.
- f_s is the frequency of the stator voltage and currents of the AC motor in Hz.
- *p* is the number of pole pairs of the motor.

You can use the preceding expression to determine the frequency of reference voltages for a required speed (for a given machine).

$$f^{ref} = \frac{p \times RPM^{ref}}{60}$$

Use this frequency to generate PWM reference voltages for the inverter. Compute the magnitude of voltages by maintaining Volts/Hz ratio as:

$$V^{ref} = \left(\frac{V_{rated}}{f_{rated}}\right)f^{ref}$$

When using the per-unit system representation, the open-loop control system considers V_{rated} as the base quantity, which usually corresponds to 1PU or 100% duty cycle. Depending on the modulation technique (either Sinusoidal PWM or Space Vector PWM), you may need an additional gain ($\left(\frac{2}{\sqrt{3}}\right)$ for sinusoidal PWM). At lower speeds, the system needs a minimum boost voltage (15% or 25% of the rated voltage) to overcome the effect of the stator resistance voltage drop.

You can use open-loop control in applications where dynamic response is not a concern, and a cost-effective solution is required. Open-loop motor control does not have the ability to consider external conditions that can affect the motor speed. Therefore the control system cannot automatically correct the deviation between the desired and the actual motor speeds.

Note Scalar control implementation does not consider compensating voltage drop due to stator resistance and field weakening.

Closed-Loop Motor Control

Closed-loop control takes the system feedback into consideration for control. Closed-loop control of the motor considers the feedback of motor signals like current and position. The control system uses the feedback signals to regulate the voltage (applied to the motor) to keep the motor response at a reference value.
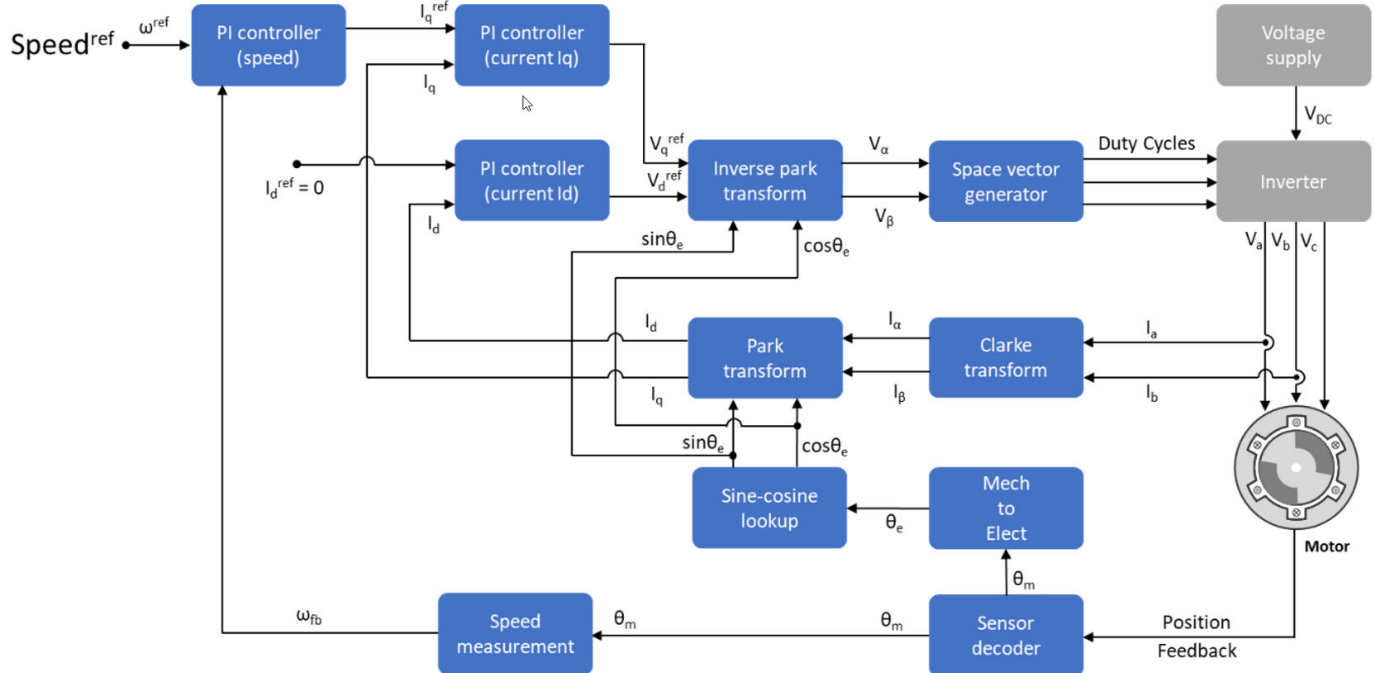
Field-Oriented Control (FOC) (or vector control) is a popular closed-loop system that is used in motor control applications. The FOC technique is used to implement closed-loop torque, speed, and position control of motors. This technique also provides good control capability over the full torque and speed ranges. The FOC implementation needs transformation of stator currents from the stationary reference frame to the rotor flux reference frame.

Speed control and torque control are the commonly used control modes in FOC. The position control mode is less commonly used. Most traction applications use the torque control mode in which the motor control system follows a reference torque value. In the speed control mode, the motor controller follows a reference speed value and generates a torque reference for torque control that forms an inner subsystem. Whereas, in the position control mode, the speed controller forms the inner subsystem.

You need real-time feedback of the current and rotor position to implement the FOC algorithm. You can use sensors to measure the current and the rotor position. You can also use sensorless techniques that use estimated feedback values instead of the actual sensor-based measurements.

Closed-loop control uses the real-time position and stator current feedback to tune the speed controller and the current controller and change the duty cycles of the inverter. This ensures that the corrected three-phase voltage supply (that runs the motor) corrects the motor feedback deviation from the desired value.

Open-Loop to Closed-Loop Transitions

Some applications require the motor to start using an open-loop control. Once the motor achieves the minimum required stability in open-loop control, the control system shifts to closed-loop.

In a quadrature encoder-based position sensing system, the motor starts up in open-loop and transitions to closed-loop once the index pulse is detected.

In sensorless position control, the motor starts running at 10% of the base speed in the open-loop. After the reference switch goes beyond 10% of the base speed, the control system transitions from open-loop to closed-loop.

To ensure smooth transition from open-loop to closed-loop, the PI controllers reset and start from the same initial condition as the open-loop outputs.

Current Sensor ADC Offset and Position Sensor Calibration

This section explains about analog to digital controller (ADC) and position sensor offset calibration.

Current Sensor ADC Offset Calibration

In an inverter, signal conditioning for the current sensor introduces an offset voltage in the ADC input to measure both positive and negative current. This offset value is different for each target hardware because it depends on the tolerances of the components in the signal sensing and conditioning circuit. It is recommend that you measure the current sensor ADC offset for the target hardware. Current sensor ADC offset is represented in ADC counts that correspond to zero ampere current.

See the example "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6 to manually measure the ADC offset value. In the Motor Control Blockset examples, update the measured value in the `inverter.CtSensAOffset` and `inverter.CtSensBOffset` variables in the model initialization script. By default, the script updates the `inverter.CtSensAOffset` and `inverter.CtSensBOffset` variables with the default values.

The examples in Motor Control Blockset calculate the current sensor ADC offset in the hardware initialization subsystem. In the model initialization script, when you set `inverter.ADCOffsetCalibEnable = 1`, the script enables the current sensor offset calibration in the target hardware during initialization. In the hardware initialization subsystem, ADC channels read the input current multiple times and averages them. The current controller uses this averaged ADC offset value. In the model initialization script, when you set `inverter.ADCOffsetCalibEnable = 0`, the script disables the current sensor offset calibration and uses the values from the initialization script.

Note Always measure the current sensor ADC offset when the motor is not running. It is recommended that you unplug the electric wires connected to the motor.

## Position Sensor Offset Calibration for Quadrature Encoder and Hall Sensor

The controller requires the position sensor offset computation to determine accurate real-time feedback of the rotor position and implement the Field-Oriented Control (FOC) algorithm correctly. It is recommended that you use the examples for offset calibration to compute the position offset before running any other example that uses FOC.

Hall sensor offset is the angle between the *d*-axis of the rotor and the position detected by the Hall sensor. You can use the offset to correct and compute an accurate position of the *d*-axis of the rotor.

Quadrature encoder sensor offset is the angle between the *d*-axis of the rotor and the encoder index pulse position detected by the quadrature encoder.

Motor Control Blockset offers examples like "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82 and "Hall Offset Calibration for PMSM Motor" on page 4-72 to obtain the accurate rotor position for implementing the control algorithm. The offset computation examples use a unique algorithm along with open-loop control to compute the position offsets of the position sensors (Hall or quadrature encoder). Open-loop control (also known as scalar control or volt/Hz control) is a popular motor control technique that can be used to run any AC motor. This is a simple

technique that does not need any feedback from the motor. To ensure a constant stator magnetic flux, keep the supply voltage amplitude proportional to its frequency. This figure shows an overview of the open-loop control. See "Open-Loop and Closed-Loop Control" on page 6-8 for more details.
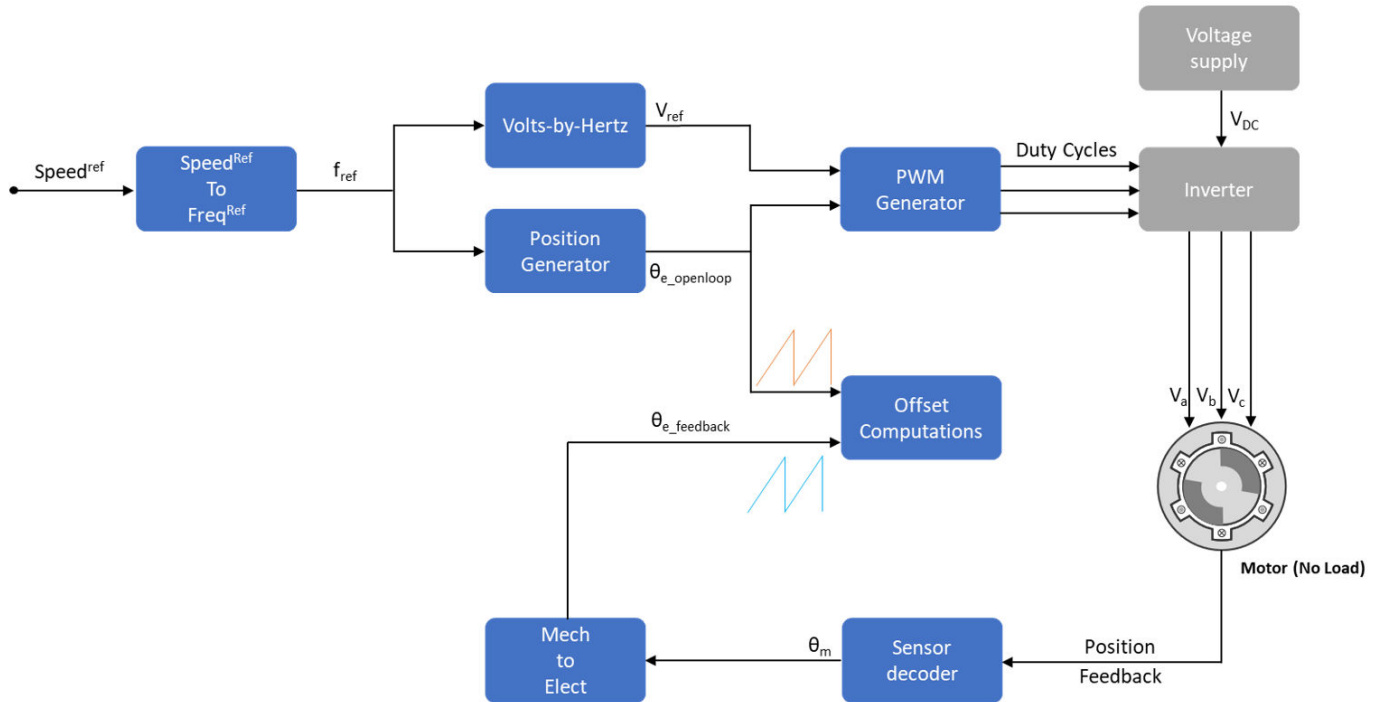
By using this algorithm, the offset calibration examples detect the position offset in this manner:

- Check if the motor is in a no-load condition.
- Start and run the motor in open-loop at a very low speed (for example, 60rpm). At a low speed, the rotor d-axis closely aligns with the rotating magnetic field of the stator.
- Measure the feedback position of the available position sensor (Hall or quadrature encoder).
- Compare the open-loop position with feedback position and check that the phase-sequence is correct. If required, correct the motor phase-sequence.
- Compute the Hall sensor position offset by obtaining the difference between the open-loop position and feedback position.
- Run the motor in the open-loop for few cycles and stop the motor. Ensure that the encoder index pulse is detected at least once. Lock the rotor in the d-axis. The quadrature encoder position offset is identical to the position feedback. This outputs the quadrature encoder mechanical offset position.
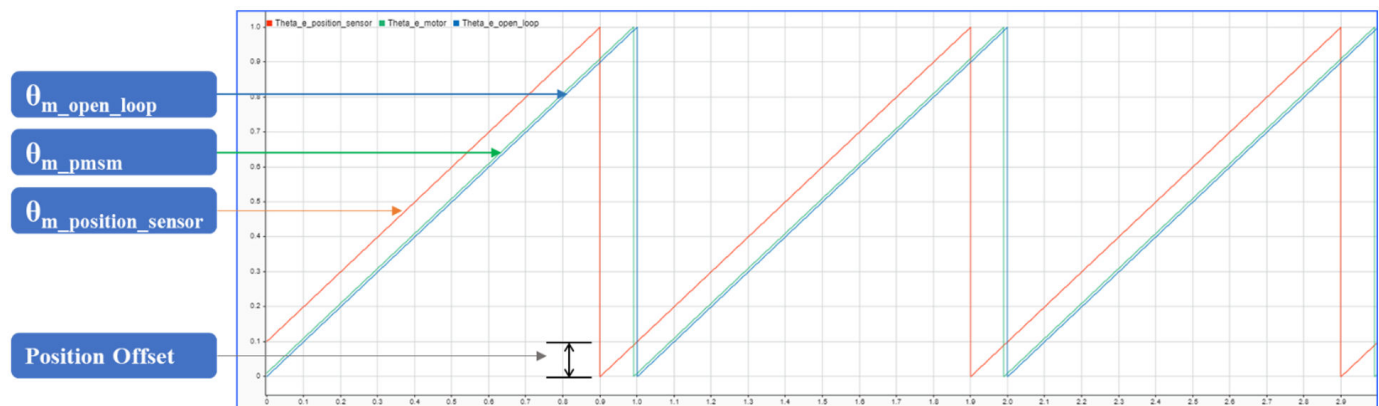
This figure shows the comparison of open-loop position from the control algorithm along with the actual position of the motor. The figure also shows the feedback from the position sensor. The position offset, which is the difference between the open-loop position and feedback position from the sensor, is computed by the algorithm provided in the offset calibration models.

- Update the measured offset in the `pmsm.PositionOffset` variable in the model initialization script of the examples.
- For parameter estimation, update the measured Hall offset in the Hall Offset field of the `mcb_param_est_host_read` model.

Note The "Hall Offset Calibration for PMSM Motor" on page 4-72 example outputs the electrical position offset. Whereas, the "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82 example outputs the mechanical position offset.

For steps to compute the offsets, see these examples:

"Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-82

"Hall Offset Calibration for PMSM Motor" on page 4-72

"Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-6

Per-Unit System

Motor Control Blockset uses these International System of Units (SI):

Quantity	Unit	Symbol
Voltage	volt	V
Current	ampere	A
Speed	radians per second	rad/s
	revolutions per minute	rpm
Torque	newton-meter	N.m
Power	watt	W

Note The SI Unit for speed is rad/s. However, most manufacturers use rpm as the unit to specify the rotational speed of the motors. Motor Control Blockset prefers rpm as the unit of rotational speed over rad/s. However, you can use either value based on your preference.

Per-Unit System

The per-unit (PU) system is commonly used in electrical engineering to express the values of quantities like voltage, current, power, and so on. It is used for transformers and AC machines for power system analysis. Embedded systems engineers also use this system for optimized code-generation and scalability, especially when working with fixed-point targets.

For a given quantity (such as voltage, current, power, speed, and torque), the PU system expresses a value in terms of a base quantity:

$$\text{quantity expressed in PU} = \frac{\text{quantity expressed in SI units}}{\text{base value}}$$

Generally, most systems select the nominal values of the system as the base values. Sometimes, a system may also select the maximum measurable value as the base value. After you establish the base values, all signals are represented in PU with respect to the selected base value.

For example, in a motor control system, if the selected base value of the current is 10A, then the PU representation of a 2A current is expressed as (2/10) PU = 0.2 PU.

Similarly,

$$\text{quantity expressed in SI units} = \text{quantity expressed in PU} \times \text{base value}$$

For example, the SI unit representation of 0.2 PU = (0.2 x base value) = (0.2 x 10) A.

Per-Unit System and Motor Control Blockset

Motor Control Blockset uses these conventions to define the base values for voltage, current, speed, torque, and power.

Quantity	Representation	Convention
Base voltage	V_{base}	This is the maximum phase voltage supplied by the inverter. Generally, for Space Vector PWM, it is PU_System.V_base = . $\left(\dfrac{inverter.V\_dc}{\sqrt{3}}\right)$ For Sinusoidal PWM, it is PU_System.V_base = . $\left(\dfrac{inverter.V\_dc}{2}\right)$
Base current	I_{base}	This is the maximum current that can be measured by the current sensing circuit of the inverter. Generally, but not necessarily, it is I_{max} of the inverter. PU_System.I_base = inverter.I_max
Base speed	N_{base}	This is the nominal (or rated) speed of the motor. This is also the maximum speed that the motor can achieve at the nominal voltage and nominal load without a field-weakening operation.
Base torque	T_{base}	This is torque that is mathematically derived at the base current. Physically, the motor may or may not be able to produce this torque. Generally, it is PU_System.T_base = $\dfrac{3}{2}$. $\times$ pmsm.p $\times$ pmsm.FluxPM $\times$ PU_System.I_base

Quantity	Representation	Convention
Base power	P_{base}	This is the power derived by the base voltage and base current. Generally, it is $$PU\_System.P\_base = \frac{3}{2}.$$ $\times$ PU_System.V_base $\times$ PU_System.I_base

where:

- V_{dc} is the DC voltage that you provide to the inverter.
- I_{max} is the maximum current measured by the ADCs connected to the current sensors of the inverter.
- p is the number of pole pairs available in the PMSM.
- *FluxPM* is the permanent magnet flux linkage of the PMSM.
- *pmsm* is the MATLAB workspace parameter structure used save motor variables.
- *inverter* is the MATLAB workspace parameter structure used save inverter variables.
- *PU_System* is the MATLAB workspace parameter structure used save PU system variables.

For the voltage and current values, you can generally consider the peak value of the nominal sinusoidal voltage (or current) as 1PU. Therefore, the base values used for voltage and current are the RMS values multiplied by $\sqrt{2}$, or the peak value measured between phase-neutral.

You can simplify your calculations by using the PU system. Motor Control Blockset uses these base value definitions for the PU-system-related conversions performed by the algorithms used in the toolbox examples. The toolbox stores the PU-system-related variables in a structure called `PU_System` in the MATLAB workspace.

Why Use Per-Unit System Instead of Standard SI Units

Per-unit representation of signals has many advantages over the SI units. This technique:

- Improves the computational efficiency of code execution, and therefore is a preferred system for fixed-point targets.
- Creates a scalable control algorithm that can be used across many systems.

Hardware Connections

Hardware Connections

Motor Control Blockset supports the following hardware configurations:

1 F28069 control card configuration
2 LAUNCHXL-F28069M configuration
3 LAUNCHXL-F28379D configuration
4 C2000 MCU Resolver Eval Kit [R2]

F28069 control card configuration

The configuration includes the following hardware components:

- Texas Instruments DRV8312-69M-KIT inverter board
- Texas Instruments F28069 microcontroller control card
- Motor BLY171D (supports both Hall and quadrature encoder sensors)
- Motor BLY172S (supports Hall sensor)
- Quadrature encoder
- DC power supply

Note Due to auxiliary power supply related hardware issues, the DRV8312-69M-KIT does not support the position sensors connected to some motors (for example, Teknic M-2310P motor).

The following steps describe the hardware connections for the F28069 control card configuration:

1 Connect the F28069 control card to J1 of DRV8312-69M-KIT inverter board.
2 Connect the motor three phases, to MOA, MOB, and MOC on the inverter board.
3 Connect the DC power supply (24V) to PVDDIN on the inverter board.

Warning Be careful when connecting PVDD and GND to the positive and negative connections of the DC power supply. A reverse connection can damage the hardware components.
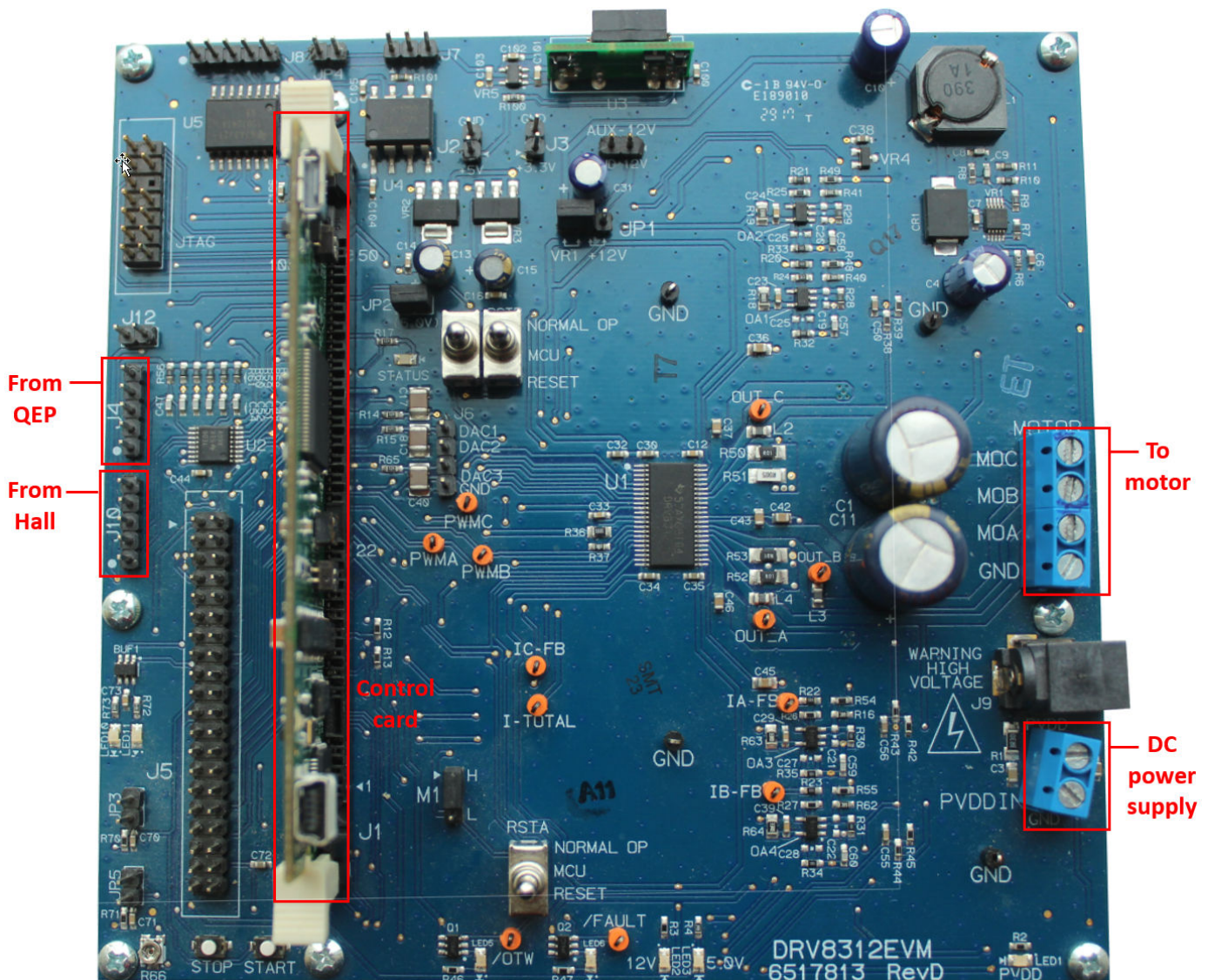
The following step describes about interfacing the quadrature encoder sensor:

- Connect the quadrature encoder pins (G, I, A, 5V, B) to J4 on the inverter board.

To implement position-sensing by using Hall sensor, use a motor that has inbuilt Hall sensors (for example, BLY171D and BLY172S). The following steps describe the steps to interface the Hall sensor:

- Connect the Hall sensor encoder output to J10 on the inverter board.

DRV8312-69M-KIT inverter
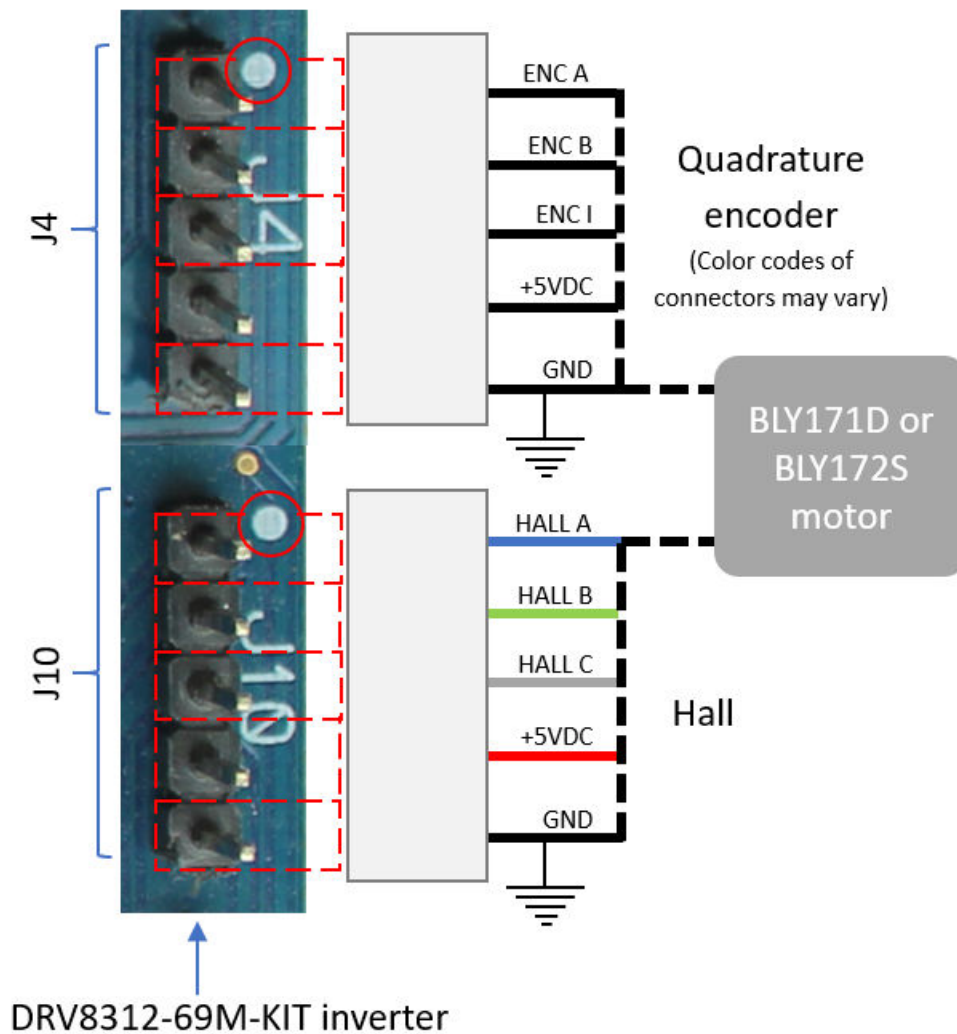
We recommend the following jumper settings for DRV8312-69M-KIT inverter board when working with Motor Control Blockset. You can customize these settings depending on the application requirements. For more information about these settings, see the device user guide available on Texas Instruments website.

- JP1 – VR1
- JP2 – ON
- JP3 – OFF
- JP4 – OFF
- JP5 – OFF
- M1 – H
- J2 – OFF
- J3 – OFF
- RSTA – MCU
- RSTB - MCU

- RSTC - MCU

LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations

The LAUNCHXL-F28069M configuration includes the following hardware components:

- LAUNCHXL-F28069M controller
- BOOSTXL-DRV8305 (supported inverter)
- Teknic motor M-2310P (supports both Hall and quadrature encoder sensors)
- Motor BLY171D (supports both Hall and quadrature encoder sensors)
- Motor BLY172S (supports Hall sensor)
- DC power supply

The LAUNCHXL-F28379D configuration includes the following hardware components:

- LAUNCHXL-F28379D controller
- BOOSTXL-DRV8305 and BOOSTXL-3PHGANINV (supported inverters)
- Teknic motor M-2310P (supports both Hall and quadrature encoder sensors)
- Motor BLY171D (supports both Hall and quadrature encoder sensors)
- Motor BLY172S (supports Hall sensor)
- DC power supply

The following steps describe the hardware connections for the LAUNCHXL-F28069M and LAUNCHXL-F28379D configurations:

1 Attach the BOOSTXL inverter board to J1, J2, J3, J4 on the LAUNCHXL controller board.

Note Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J1, J2 of LAUNCHXL.

2 Connect the motor three phases, to MOTA, MOTB, and MOTC on the BOOSTXL inverter board.

3 Connect the DC power supply (24V) to PVDD and GND on the BOOSTXL inverter board.

Warning Be careful when connecting PVDD and GND to the positive and negative connections of the DC power supply. A reverse connection can damage the hardware components.

BOOSTXL-DRV8305EVM

From Quadrature Encoder

**From Hall (GPIO)**

BOOSTXL-DRV8305EVM
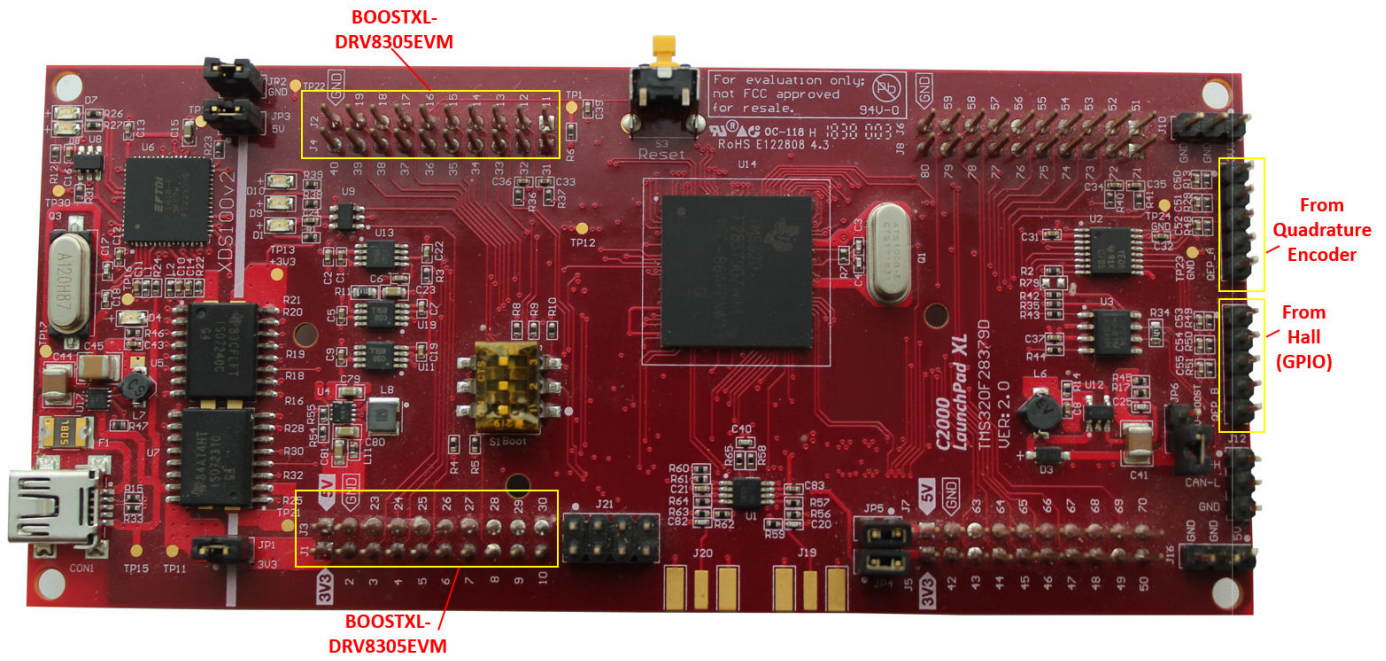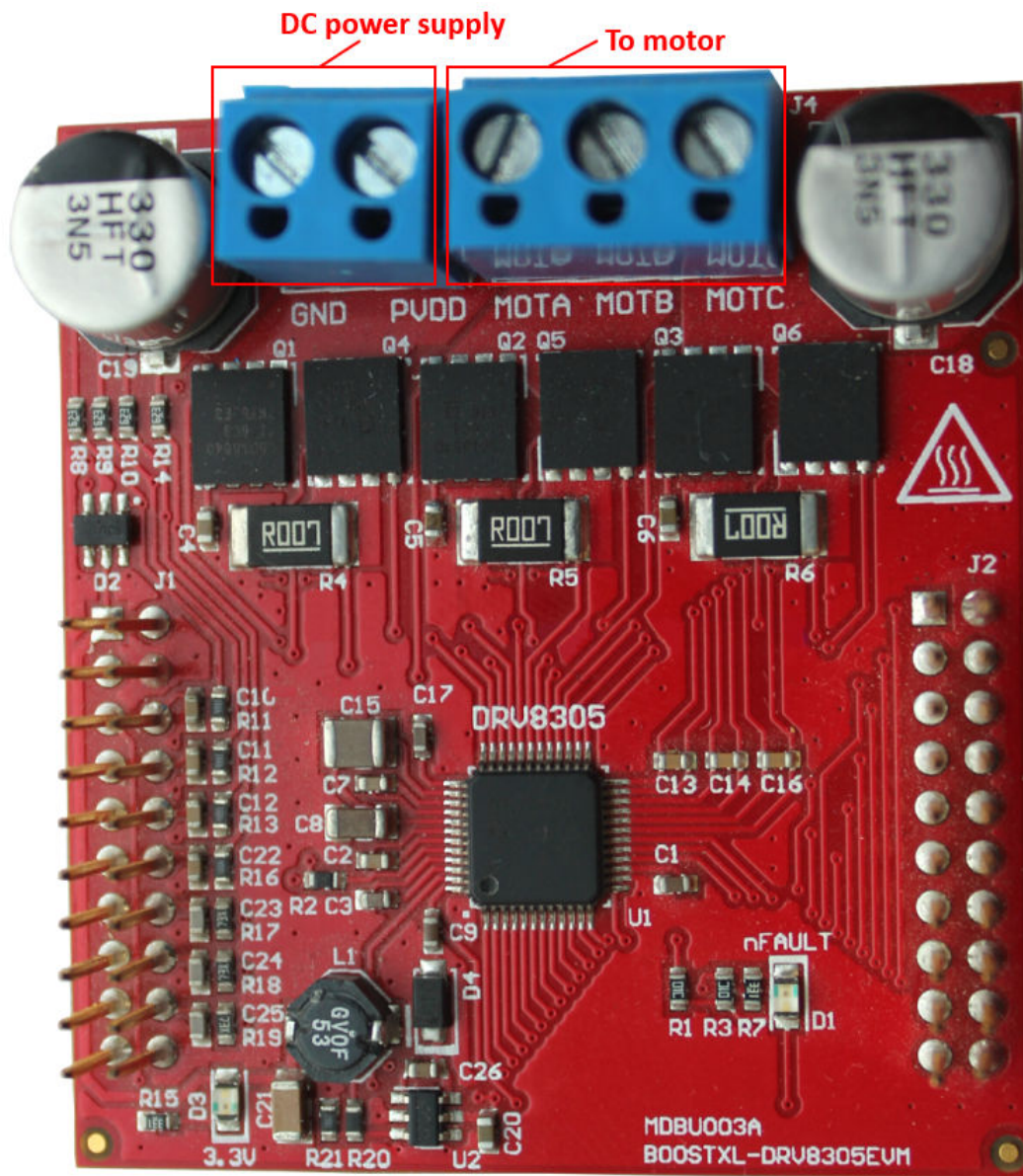
The following step describes about interfacing the quadrature encoder sensor:

- Connect the quadrature encoder pins (G, I, A, 5V, B) to QEP_A on the LAUNCHXL controller board.

To implement position-sensing by using Hall sensor, use a motor that has inbuilt Hall sensors (for example, Teknic motor M-2310P, BLY171D and BLY172S). The following steps describe the steps to interface the Hall sensor:

- Connect the Hall sensor encoder output to a GPIO port that is configured as eCAP, on the LAUNCHXL controller board.

LAUNCHXL-F28069M or
LAUNCHXL-F28379D controller

LAUNCHXL-F28069M or
LAUNCHXL-F28379D controller
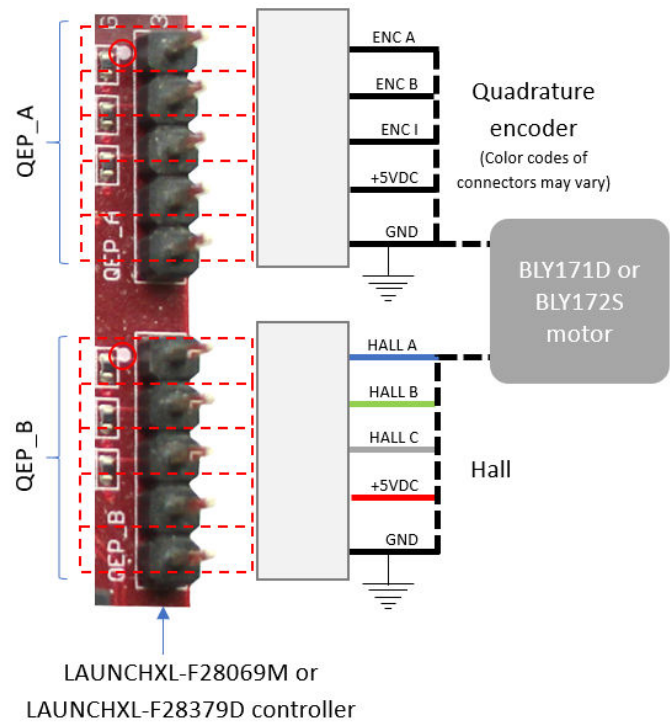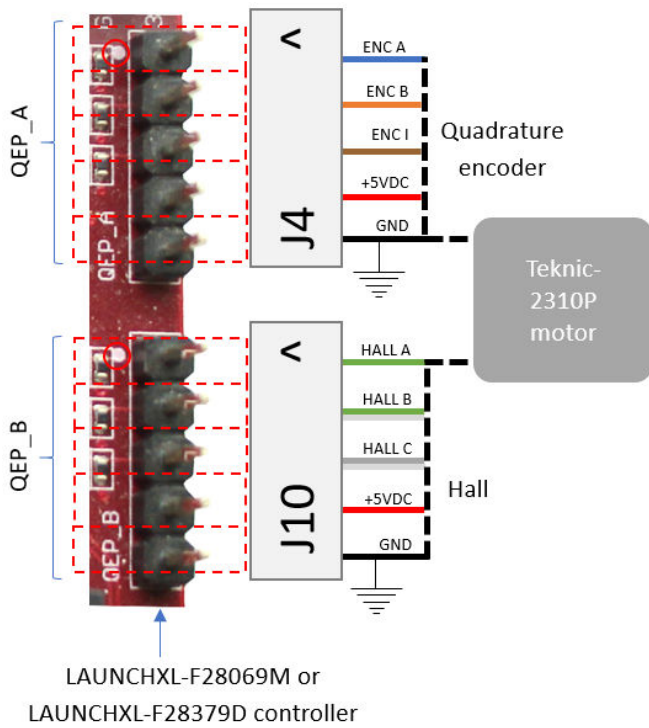
We recommend the following jumper settings for the LAUNCHXL inverter boards when working with Motor Control Blockset. You can customize these settings depending on the application requirements. For more information about these settings, see the device user guide available on Texas Instruments website.

For LAUNCHXL-F28069M controller

- JP1 – ON
- JP2 – ON
- JP3 – ON
- JP4 – ON
- JP5 – ON
- JP6 – OFF
- JP7 – ON

For LAUNCHXL-F28379D controller

- JP1 – ON
- JP2 – ON
- JP3 – ON
- JP4 – ON
- JP5 – ON
- JP6 – OFF

Instructions for Dyno (Dual Motor) Setup

1 Connect the three phases of Motor1 and Motor2, to MOTA, MOTB, and MOTC on the corresponding BOOSTXL inverter boards.

2 Attach the BOOSTXL inverter board (connected to Motor1) to J1, J2, J3, J4 on the LAUNCHXL controller board.

Note Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J1, J2 of LAUNCHXL.

3 Attach the BOOSTXL inverter board (connected to Motor2) to J5, J6, J7, J8 on the LAUNCHXL controller board.

Note Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J5, J6 of LAUNCHXL.

4 Connect the DC power supply (24V) to PVDD and GND on both BOOSTXL inverter boards.

Note Connect the PVDD and GND on the BOOSTXL boards (for MOTOR1 and MOTOR2) to the same power supply. When one motor consumes power, the second motor generates power. If you connect both motors to the same power supply, the power generated by one motor is consumed by the other motor. The DC power supply delivers power only for the losses.

5 Connect the quadrature encoder pins of Motor1 (G, I, A, 5V, B) to QEP_A on the LAUNCHXL controller board.

6 Connect the quadrature encoder pins of Motor2 (G, I, A, 5V, B) to QEP_B on the LAUNCHXL controller board.

Warning Be careful when connecting PVDD and GND to the positive and negative connections of the DC power supply. A reverse connection can damage the hardware components.
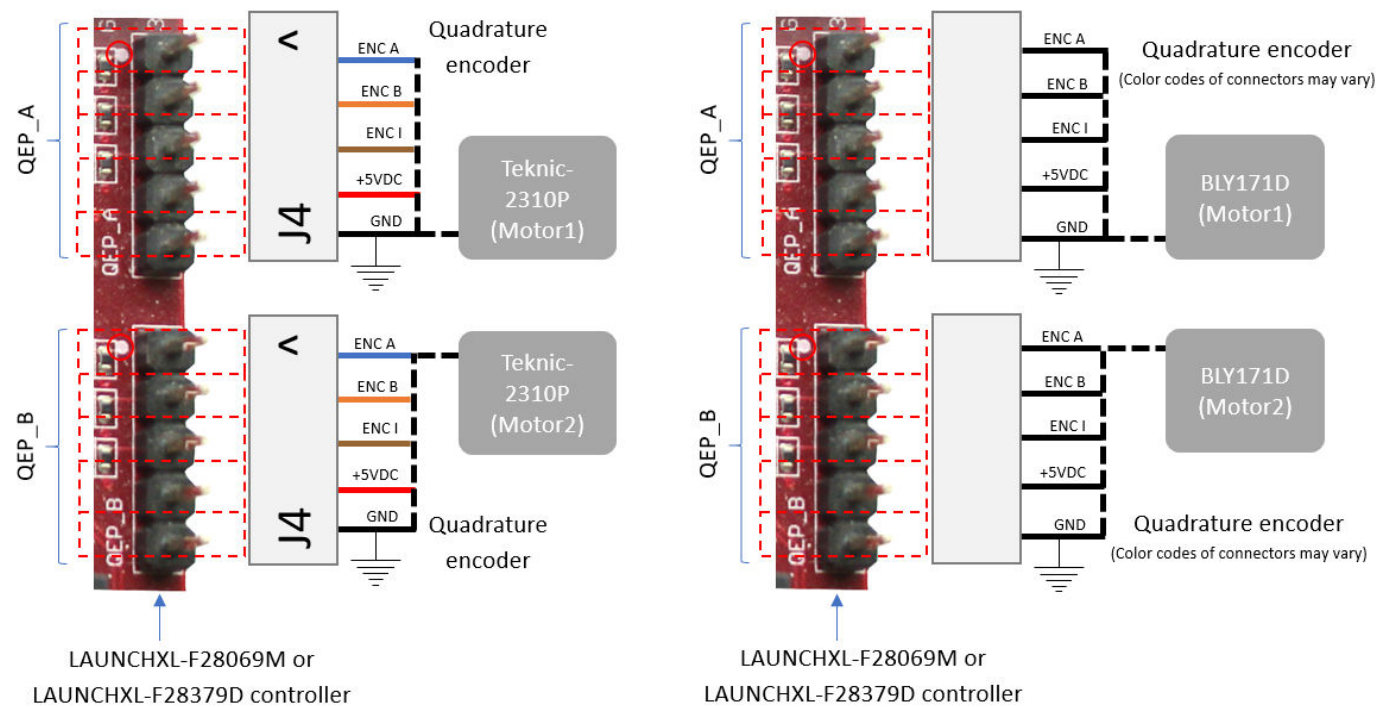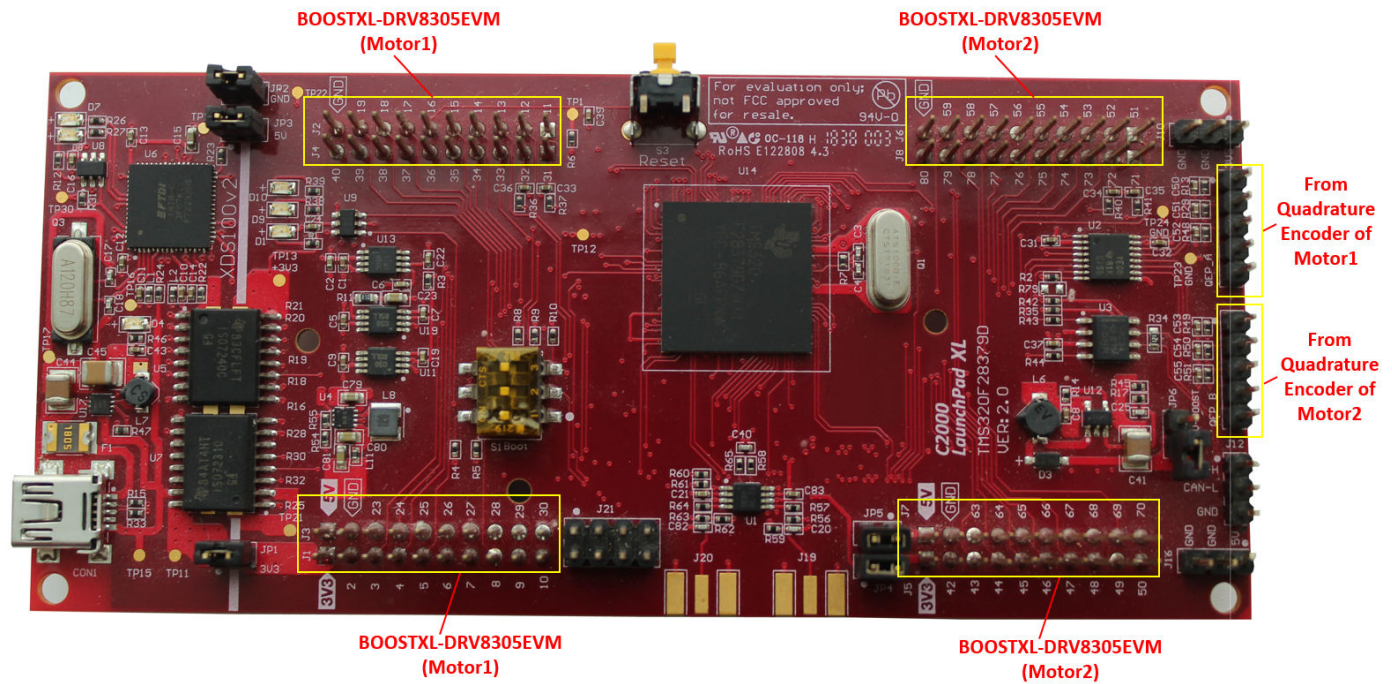
C2000 MCU Resolver Eval Kit [R2]

The C2000 MCU Resolver Eval Kit [R2] configuration includes the following hardware components:

- LAUNCHXL-F28069M controller

- BOOSTXL-DRV8305 (supported inverter)
- DC power supply
- C2000 MCU Resolver Eval Kit [R2]
- Resolver encoder

The following steps describe the hardware connections for the MCU Resolver Eval Kit [R2] board:

1 Connect DC power supply (15V) to J2 on the MCU Resolver Eval Kit board.
2 Connect the resolver output pins for sine wave to pins 1, 2 of J10 on the MCU Resolver Eval Kit board.
3 Connect the resolver output pins for cosine wave to pins 3, 4 of J10 on the MCU Resolver Eval Kit board.
4 Connect the resolver input pins to the PWM_dither and PWM_SINE pins of J10 on the C2000™ MCU Resolver Eval Kit board.

The following step describes the hardware connection for the LAUNCHXL-F28069M controller board:

- Connect the LAUNCHXL-F28069M controller board to a computer via USB port.

The following steps describe the hardware connections between the MCU Resolver Eval Kit [R2] and LAUNCHXL-F28069M controller boards:

1 Connect the COS(T2) pin on MCU Resolver Eval Kit [R2] to pin 24 of J3 on the LAUNCHXL-F28069M controller board.
2 Connect the SIN(T8) pin on MCU Resolver Eval Kit [R2] to pin 29 of J3 on the LAUNCHXL-F28069M controller board.
3 Connect the GPIO2 pin on MCU Resolver Eval Kit [R2] to pin 38 of J4 on the LAUNCHXL-F28069M controller board.

Pins 1, 2 – Sine wave
Pins 3, 4 – Cosine wave
Pins 6, 7 – PWM_SINE, PWM_dither
– Resolver input

COS(T2) to Pin 24 of J3
SIN(T8) to Pin 29 of J3

DC power
supply (15V)

GPIO2 to
Pin 38 of J4

GPIO3 to
GND of J9

GND(T5) to GND of J9