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

Motor Control Blockset Product Description

Design and implement motor control algorithms

Motor Control Blockset provides Simulink® blocks and reference examples for developing and deploying motor control algorithms as optimized C and HDL code on target microcontrollers, FPGAs, or systems-on-chip (SoC). Build motor control algorithms with Clarke and Park transform, Maximum Torque Per Ampere (MTPA), six-step commutation, and lookup table (LUT)-based field weakening control blocks. You can process signals from encoders, hall sensors, and resolvers with sensor decoder blocks or implement sensorless control with estimator blocks to compute rotor position and speed. The blocks generate code that you can use in workflows involving MISRA C™ and ISO® 26262 functional safety standards.

Reference examples included in Motor Control Blockset help you understand how to develop, tune, and validate motor control algorithms using desktop and real-time simulation. Examples include algorithms for closed-loop motor control for induction motors, switched reluctance motors (SRM), synchronous motors like brushless DC motors (BLDC), and surface-mount and interior permanent magnet synchronous motors (PMSM). You can reuse the same algorithms to generate production-ready, compact, and traceable fixed- or floating-point code. You can also use the reference examples to implement algorithms for motor control hardware kits supported by the blockset.

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.
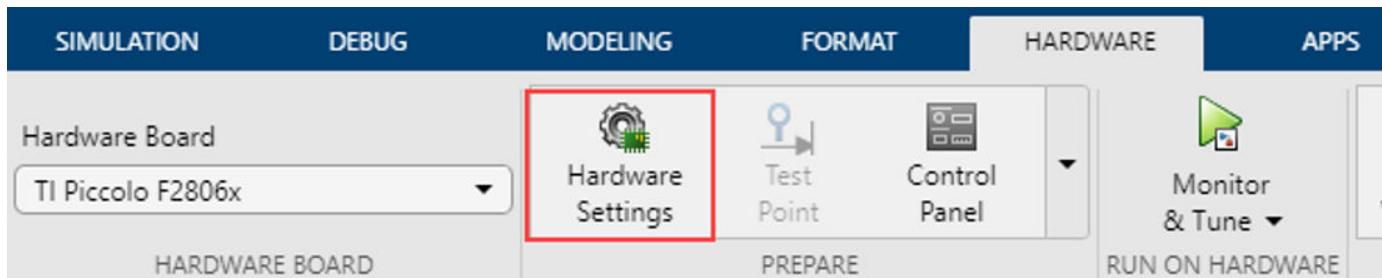
Solver Configuration

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.
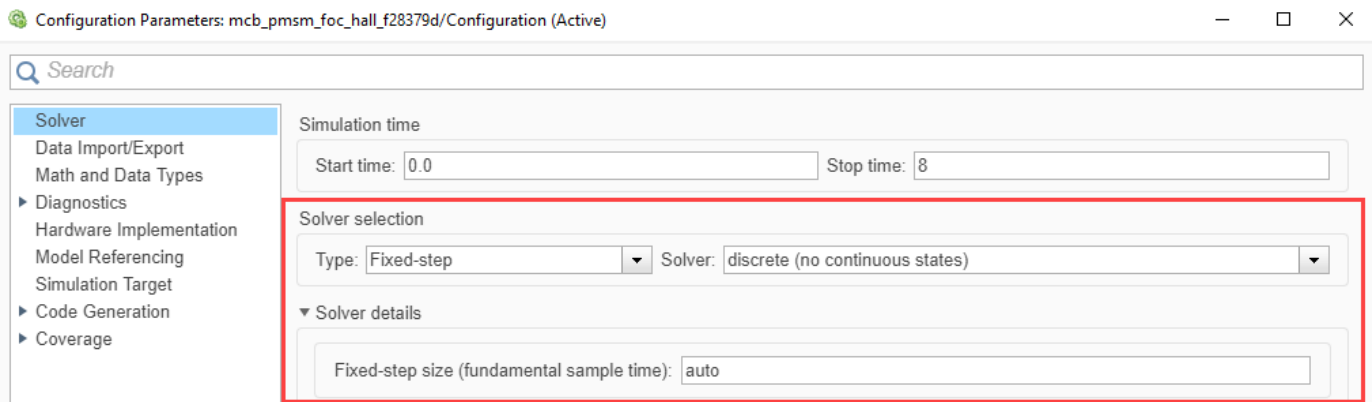
ADC Interface Configuration

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 these 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.
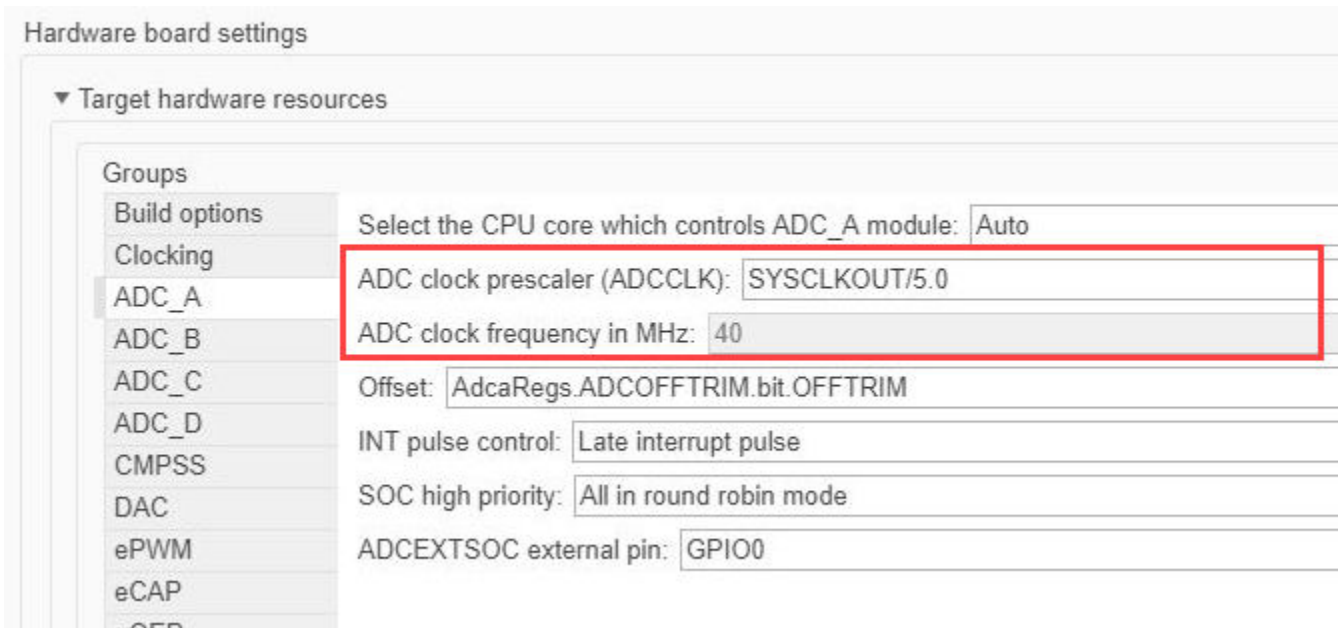
PWM Interface Configuration

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
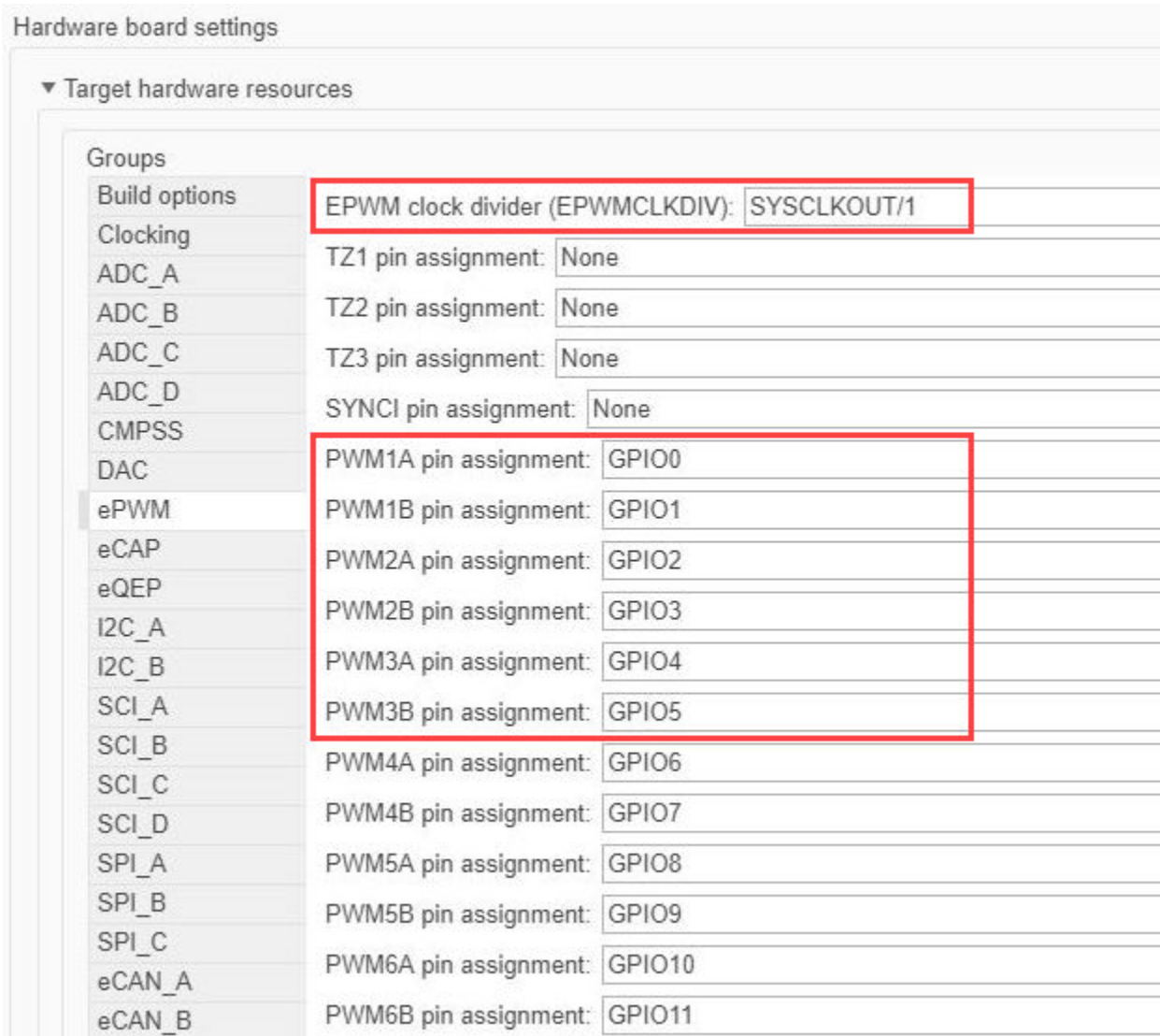
Hall Sensor Interface Configuration

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:
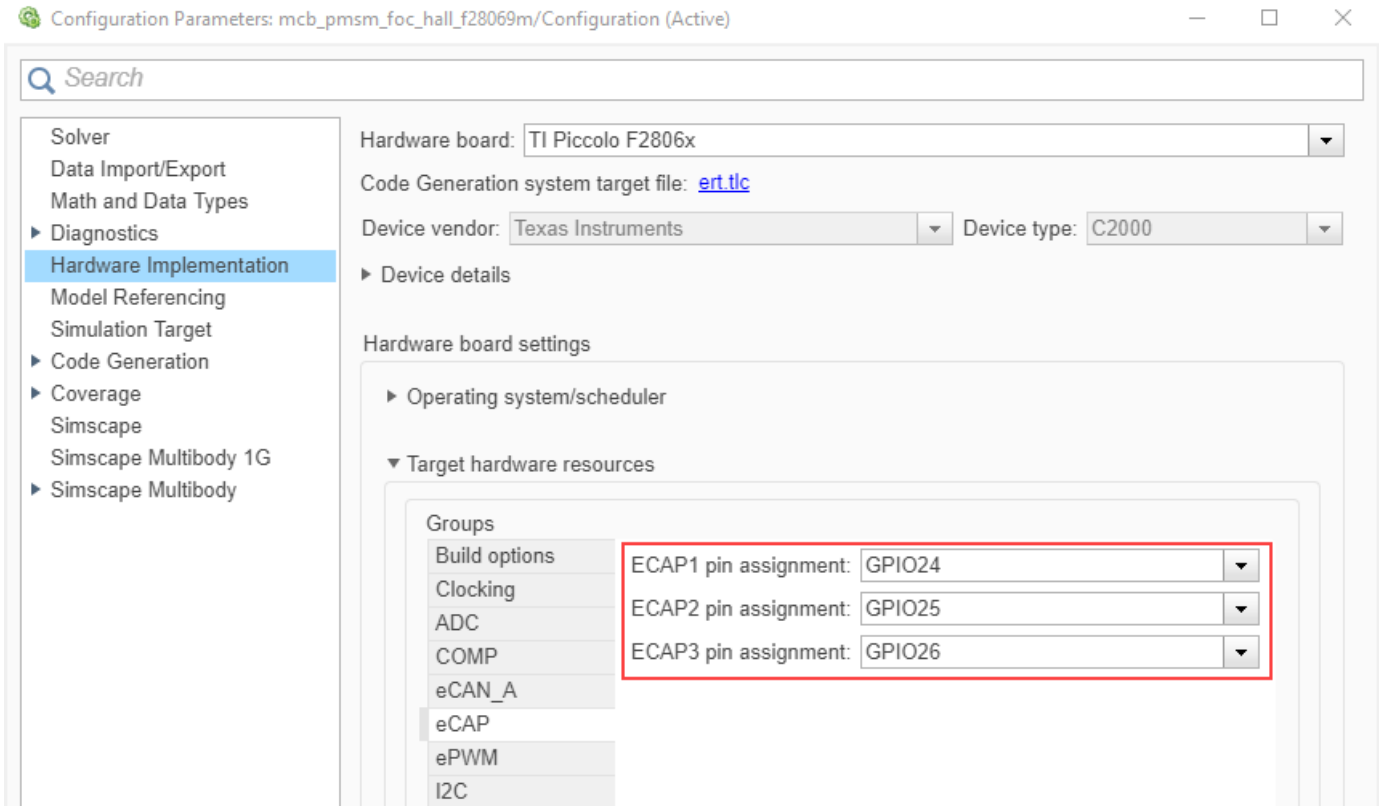
Quadrature Encoder Interface Configuration

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:
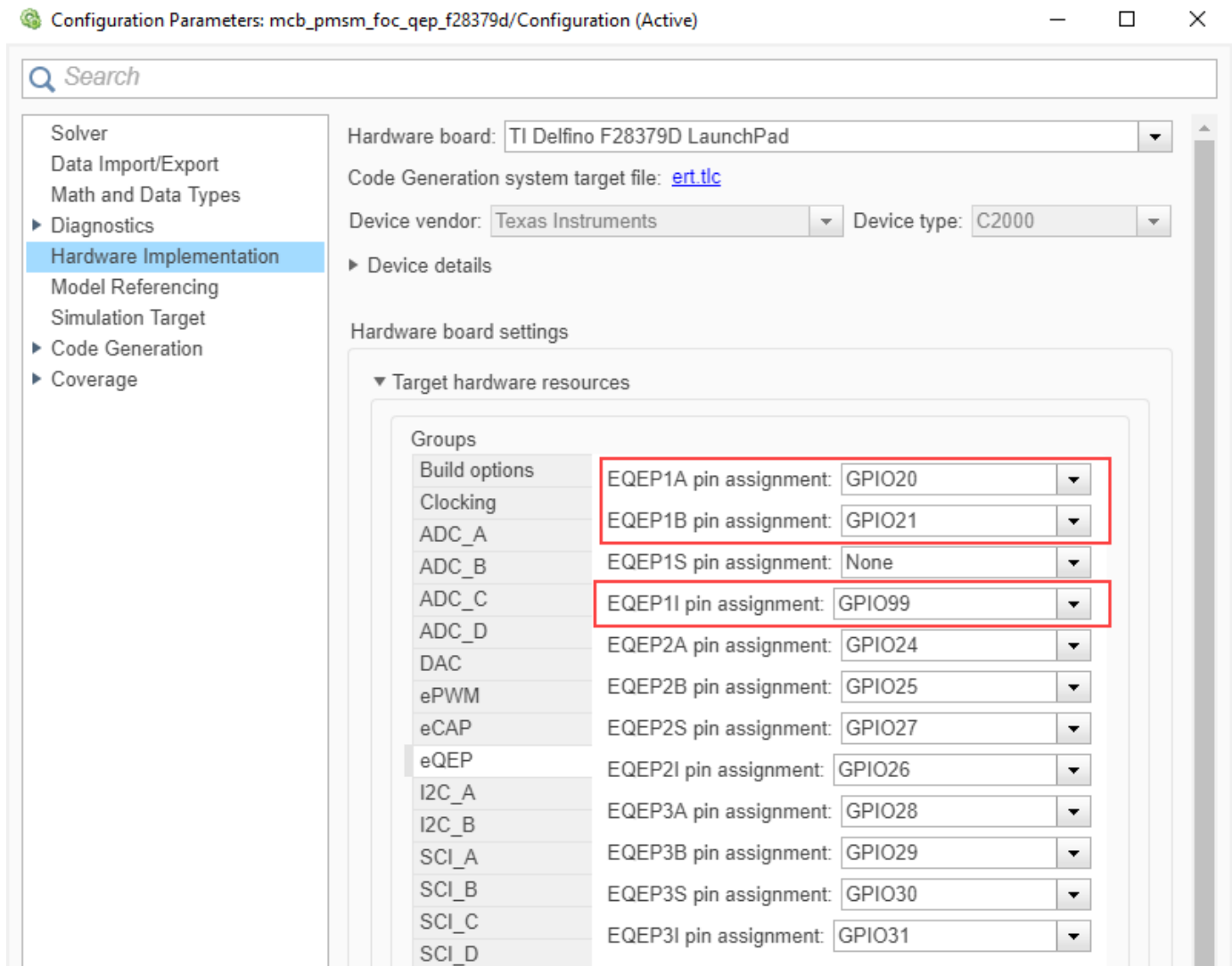
Serial Communication Interface Configuration

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:
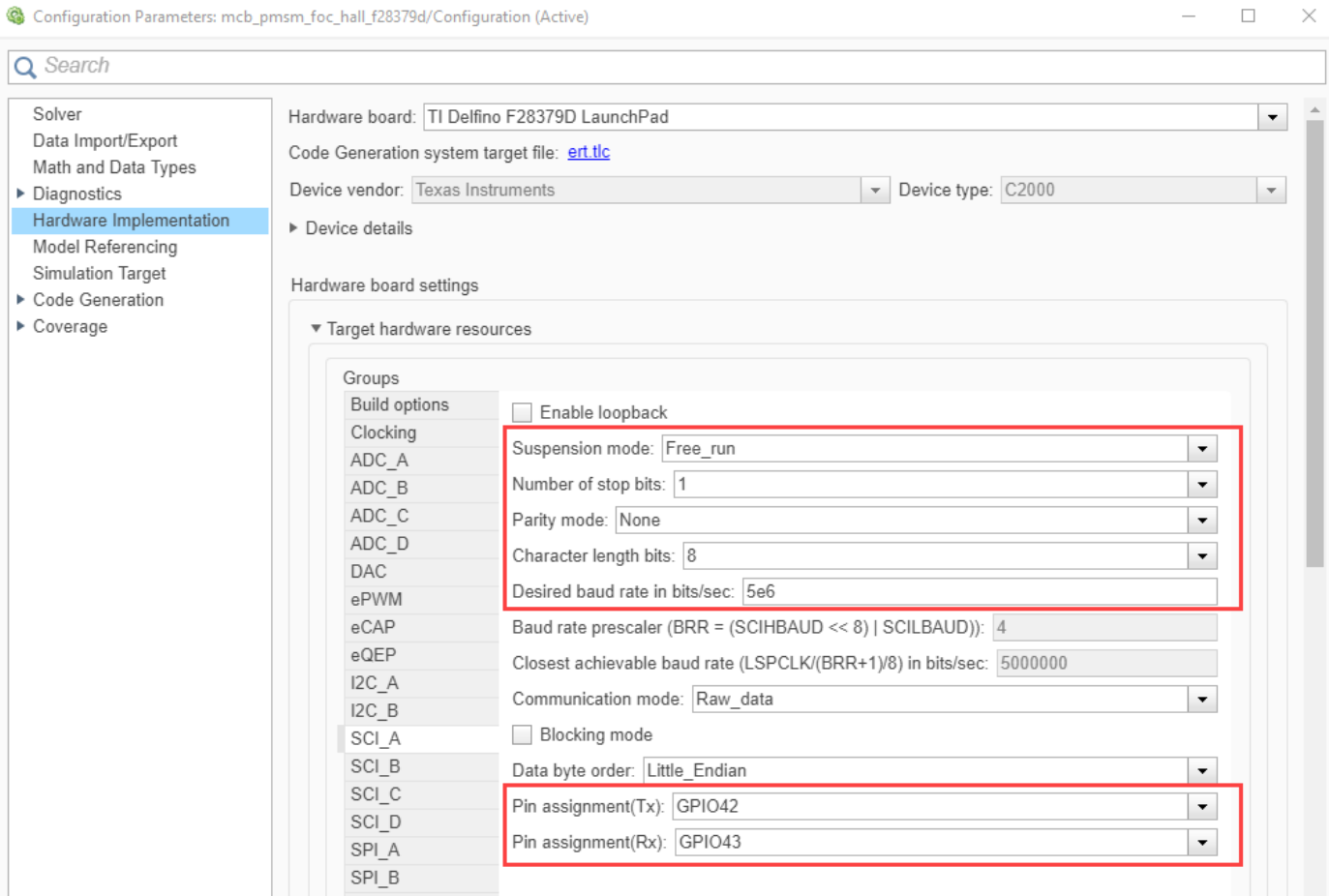
Estimate Control Gains from Motor Parameters

Estimate Control Gains and Use Utility Functions

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.
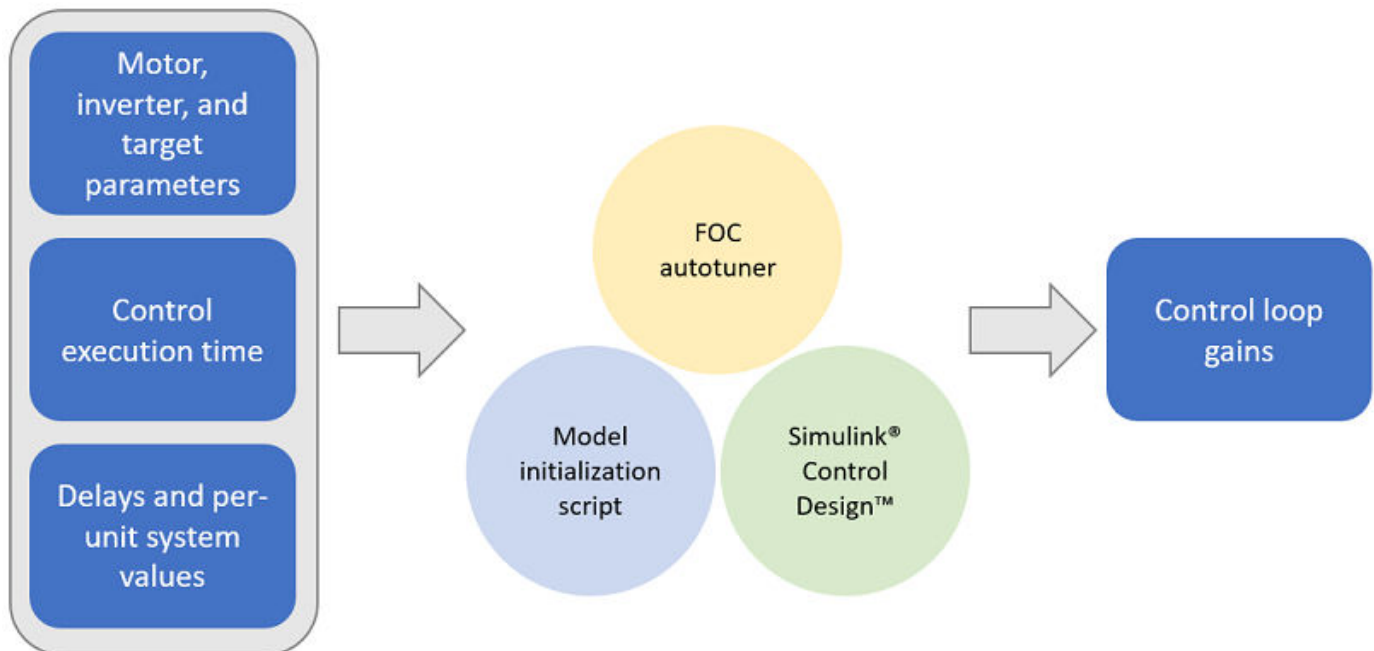
Field-Oriented Control Autotuner

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 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 Using Field Oriented Control Autotuner" on page 4-28.

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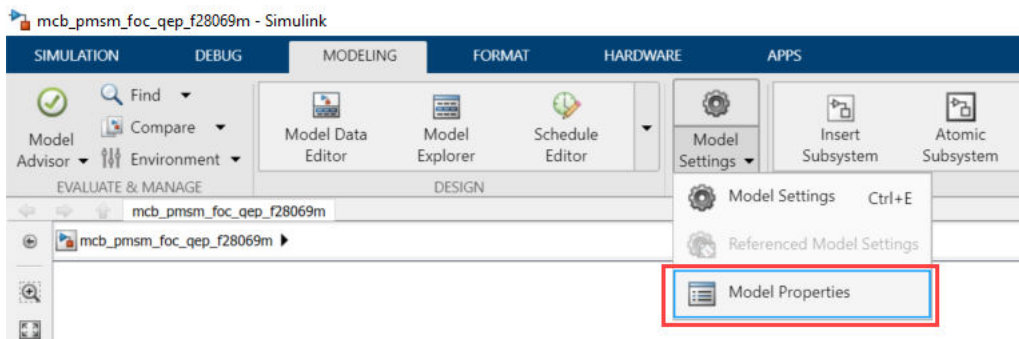
    b　In the Model Properties dialog box, navigate to the **Callbacks** tab > **InitFcn** to find the name of the script file that Simulink opens before running the example.
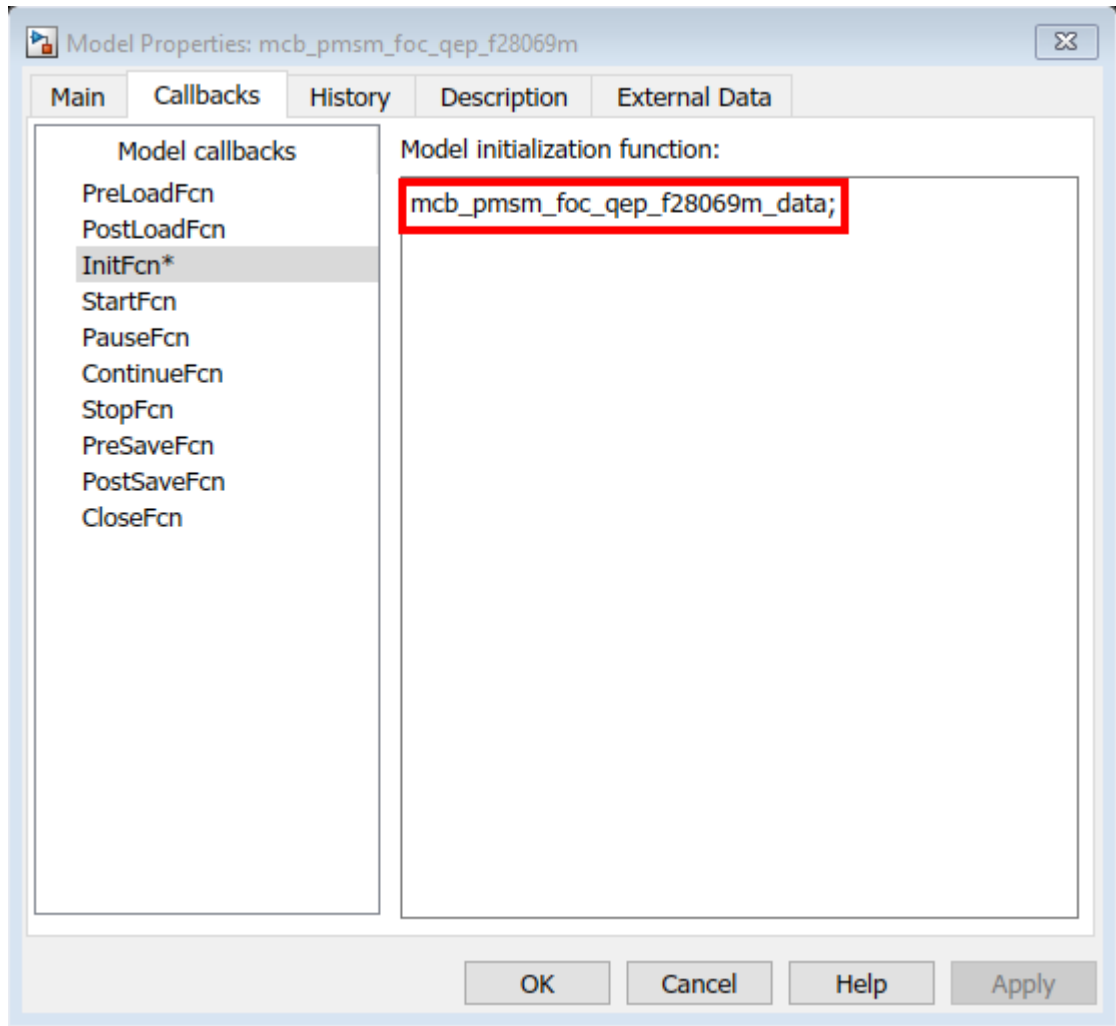
2 This figure shows an example of the initialization script (`.m`) file.

```
EDITOR        PUBLISH        VIEW

New   Open   Save   | Find Files    ⇐ ⇒   Insert  fx  🔲 ▾   |  🔲        ▷     🔀    Run Section   🕐
 ▾     ▾      ▾      | Compare ▾     Go To ▾  Comment % ⅞ ⅞   | Breakpoints  Run   Run and  Advance   Run and
              |      | Print ▾       Find ▾   Indent 🔲 🔲 🔲  |     ▾        ▾    Advance            Time
     FILE              NAVIGATE              EDIT              BREAKPOINTS    RUN

mcb_pmsm_foc_qep_f28069m_data.m    ✕    +

 1    %% ******************************************************************
 2    % Model          :    PMSM Field Oriented Control
 3    % Description     :    Set Parameters for PMSM Field Oriented Control
 4    % File name       :    mcb_pmsm_foc_qep_f28069m_data.m
 5    % Copyright 2020 The MathWorks, Inc.
 6
 7    %% Parameters needed for Offset computation are
 8    % target.PWM_Counter_Period  - PWM counter value for epwm blocks
 9    % target.CPU_frequency       - CPU frequency of the microcontroller
10    % Ts                         - Control sample time
11    % PU_System.N_base           - Base speed for per unit conversion
12    % pmsm.p                     - Number pole pairs in the motor
13
14    % Other parameters are not mandatory for offset computation
15
16    %% Set PWM Switching frequency
17    PWM_frequency  = 20e3;      %Hz        // converter s/w freq
18    T_pwm          = 1/PWM_frequency;  %s  // PWM switching time period
19
20    %% Set Sample Times
21    Ts           = T_pwm;        %sec        // simulation time step for controller
22    Ts_simulink  = T_pwm/2;      %sec        // simulation time step for model simulation
23    Ts_motor     = T_pwm/2;      %Sec        // Simulation sample time
24    Ts_inverter  = T_pwm/2;      %sec        // simulation time step for average value inverter
25    Ts_speed     = 10*Ts;        %Sec        // Sample time for speed controller
26
27    %% Set data type for controller & code-gen
28    % dataType = fixdt(1,32,17);     % Fixed point code-generation
29    dataType = 'single';            % Floating point code-generation
30
31    %% System Parameters // Hardware parameters
32
33    pmsm = mcb_SetPMSMMotorParameters('BLY171D');
34    pmsm.PositionOffset = 0.17;
35
36    %% Parameters below are not mandatory for offset computation
37
38    inverter = mcb_SetInverterParameters('DRV8312-C2-KIT');
39
40    inverter.ADCOffsetCalibEnable = 1; % Enable: 1, Disable:0
41
42    target = mcb_SetProcessorDetails('F28069M',PWM_frequency);
43
44    %% Derive Characteristics
45    pmsm.N_base = mcb_getBaseSpeed(pmsm,inverter); %rpm // Base speed of motor at given Vdc
46    % mcb_getCharacteristics(pmsm,inverter);
47
48    %% PU System details // Set base values for pu conversion
49
50    PU_System = mcb_SetPUSystem(pmsm,inverter);
51
52    %% Controller design // Get ballpark values!
53
54    PI_params = mcb.internal.SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed);
55
56    %Updating delays for simulation
57    PI_params.delay_Currents   = int32(Ts/Ts_simulink);
58    PI_params.delay_Speed      = int32(Ts_speed/Ts_simulink);
59
60    % 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 the type of PMSM (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_SetACIMMotorParameters`	Input to the function is the type of AC induction motor (for example, EM_Synergy). The function populates a structure named `acim` in the MATLAB workspace, which is used by the model. 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 `acim`. If the structure `motorParam` is not available in the MATLAB workspace, the function loads the default parameters.

Model Initialization Script	Function Called By Model Initialization Script	Description
	`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. |
| | `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 3-16 for more details. |

Model Initialization Script	Function Called By Model Initialization Script	Description
	`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.
	`mcb_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 3-18 for more details.
	`mcb_updateInverterParameters`	Inputs to the function are motor and inverter parameters. The function updates the inverter parameters based on the selected hardware and motor.

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`
	Nbase	`pmsm.N_base`
	Maximum motor speed used in the `mcb_getCharacteristics(pmsm,inverter)` function `pmsm.N_max = 2 ✖ pmsm.N_base`	`pmsm.N_max`
	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 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`
	ADC Offsets for current sensor (I_a and I_b)	`inverter.CtSensAOffset` `inverter.CtSensBOffset`

Control Parameter Category	Control Parameter Name	MATLAB Workspace Variable
	Maximum limit of automatically calibrated ADC offsets for current sensor (I_a and I_b)	`inverter.CtSensOffsetMax`
	Minimum limit of automatically calibrated ADC offsets for current sensor (I_a and I_b)	`inverter.CtSensOffsetMin`
	Enable Auto-calibration for current sense ADCs	`inverter.ADCOffsetCalibEnable`
	ADC gain factor configured by SPI	`inverter.ADCGain`
	Type of inverter: 1 — Active high-enabled inverter 0 — Active low-enabled inverter	`inverter.EnableLogic`
	Convention for current entering motor: 1 — Current entering motor sensed as positive by current sense amplifier –1 — Current entering motor sensed as negative by current sense amplifier	`inverter.invertingAmp`
	Reference voltage for the inverter current sensing circuit (V)	`inverter.ISenseVref`
	Output voltage of the inverter current sensing circuit corresponding to 1 Ampere current (V/A) You can compute this parameter using the datasheet values of current shunt resistance (`inverter.Rshunt`) and current sense amplifier gain of the inverter. `inverter.ISenseVoltPerAmp` = `inverter.Rshunt` ✖ current sense amplifier gain	`inverter.ISenseVoltPerAmp`
	Maximum measurable peak-neutral current by the inverter current sensing circuit (A)	`inverter.ISenseMax`
Processor	Manufacturer's model number	`target.model`

Control Parameter Category	Control Parameter Name	MATLAB Workspace Variable
	Manufacturer's serial number	`target.sn`
	CPU Frequency	`target.CPU_frequency`
	PWM frequency	`target.PWM_frequency`
	PWM counter period	`target.PWM_Counter_Perio d`
	Reference voltage for ADC (V)	`Target.ADC_Vref`
	Maximum count output for 12-bit ADC	`Target.ADC_MaxCount`
	Baud rate for serial communication	`Target.SCI_baud_rate`
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`
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`

Control Parameter Category	Control Parameter Name	MATLAB Workspace Variable
	Proportional gain for Field weakening controller	`PI_params.Kp_fwc`
	Integral gain for Field weakening controller	`PI_params.Ki_fwc`
Sensor delay parameters	Current sensor delay	`Delays.Current_Sensor`
	Speed sensor delay	`Delays.Speed_Sensor`
	Delay for low-pass speed filter	`Delays.Speed_Filter`
Controller delay parameters	Damping factor (ζ) of the current control loop	`Delays.OM_damping_factor`
	Symmetrical optimum factor of the speed control loop	`Delays.SO_factor_speed`

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 3-16.

Control Parameter Category	Function	Functionality
Motor characteristics for the given motor and inverter	`mcb_getCharacteristics`	Obtain these drive characteristics of a PMSM motor. • Torque as opposed to speed characteristics • Power as opposed to speed characteristics • I_{dq} as opposed to speed characteristics • Maximum phase current ($I_{peak} = \sqrt{I_d^2 + I_q^2}$) of the motor as opposed to speed characteristics For details, type `help mcb_getCharacteristics` at the MATLAB command prompt. For details, see section "Obtain Motor Characteristics" on page 3-17.

Control Parameter Category	Function	Functionality
	`mcb_getCharacteristicsAcim`	Obtain these motor characteristics of an induction motor.
		• Torque as opposed to speed characteristics
		• Power as opposed to speed characteristics
		Obtain these drive characteristics of an induction motor.
		• Torque as opposed to speed characteristics
		• Power as opposed to speed characteristics
		• I_{dq} as opposed to speed characteristics
		• Maximum phase current ($I_{peak} = \sqrt{I_d^2 + I_q^2}$) of the motor as opposed to speed characteristics
		For details, type `help mcb_getCharacteristicsAcim` at the MATLAB command prompt.
		For details, see section "Obtain Motor Characteristics" on page 3-17.
Control algorithm parameters	`mcb_SetControllerParameters`	Compute the gains for these PI controllers:
		• Current (torque) control loop gains (K_p, K_i) for currents I_d and I_q
		• Speed control loop gains (K_p, K_i)
		• Field weakening control gains (K_p, K_i)
		For details, see section "Obtain Controller Gains" on page 3-18.

Control Parameter Category	Function	Functionality
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:

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

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

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

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:

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

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, power, and current characteristics of a PMSM, 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_{dq} as opposed to Speed
- I_{peak} as opposed to Speed
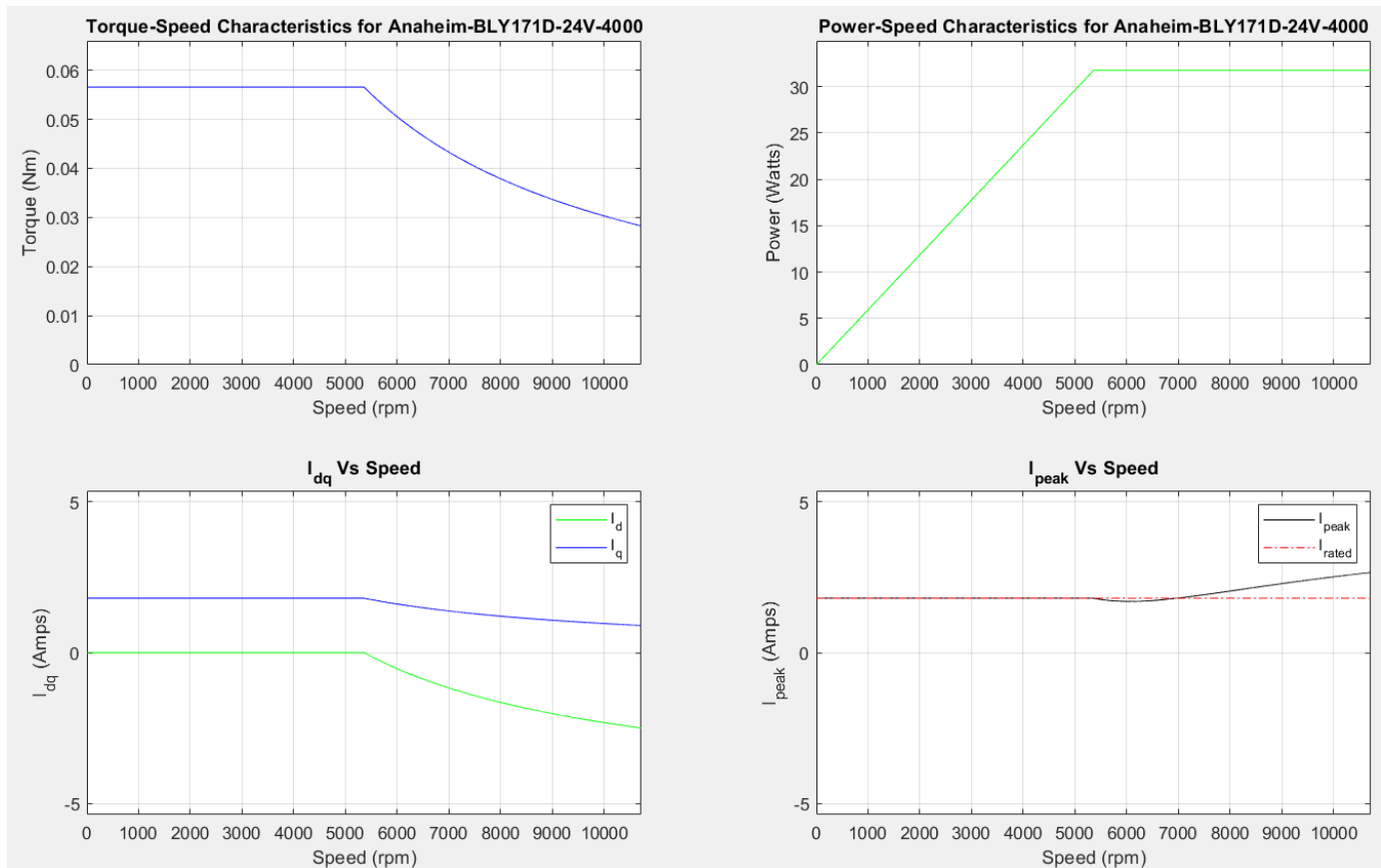
The function `mcb_getCharacteristicsAcim` calculates the motor and drive characteristics of an induction motor, which helps you to develop the control algorithm for the motor.

The function returns these motor characteristics for the given induction motor:

- Torque as opposed to Speed
- Power as opposed to Speed

The function returns these drive characteristics for the given induction motor:

- Torque as opposed to Speed
- Power as opposed to Speed
- I_{dq} as opposed to Speed
- I_{peak} as opposed to Speed

Obtain Controller Gains

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

You can use this command to call the function `mcb_SetControllerParameters`:

```
PI_params = mcb_SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed);
```

The function 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, delays, damping factor) based on the input arguments.

You can modify the default system responses by an optional input to the function that specifies the system delays, damping factor, and symmetrical optimum factor:

```
PI_params = mcb_SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed,Delays);
```

Damping factor (ζ) defines the dynamic behavior of the standard form of a second-order system, where $0 < \zeta < 1$ [1]. An underdamped system gets close to the final value more quickly than a critically damped or an overdamped system. Among the systems that respond without oscillations, a critically damped system shows the quickest response. An overdamped system is always slow in responding to any inputs. This parameter has a default value of $\frac{1}{\sqrt{2}}$.

Symmetrical optimum factor (*a*) defines the placement of the cross-over frequency at the geometric mean of the two corner frequencies, to obtain maximum phase margin that results in optimum damping of the speed loop, where $a > 1$ [2]. This parameter has a default value of `1.2`.

This example explains how to customize the parameters:

```
% Sensor Delays
Delays.Current_Sensor = 2*Ts;          %Current Sensor Delay
Delays.Speed_Sensor = Ts;              %Speed Sensor Delay
Delays.Speed_Filter = 20e-3;           %Delay for Speed filter (LPF)

% Controller Delays
Delays.OM_damping_factor = 1/sqrt(2);  %Damping factor for current control loop
Delays.SO_factor_speed = 1.5;          %Symmetrical optimum factor 1 < x < 20

% Controller design
PI_params = mcb_SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed,Delays);
```

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:
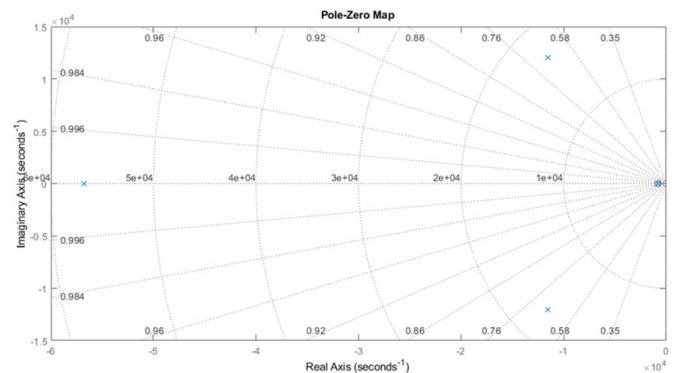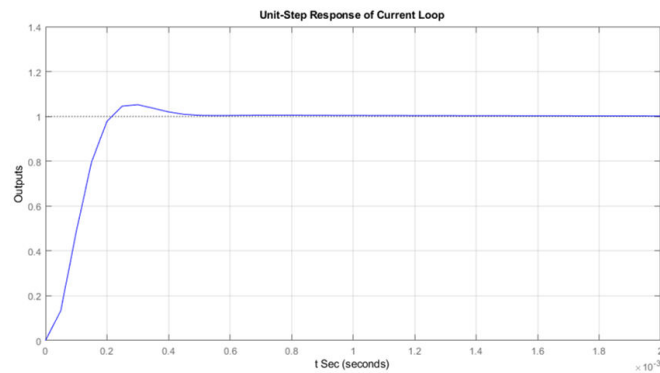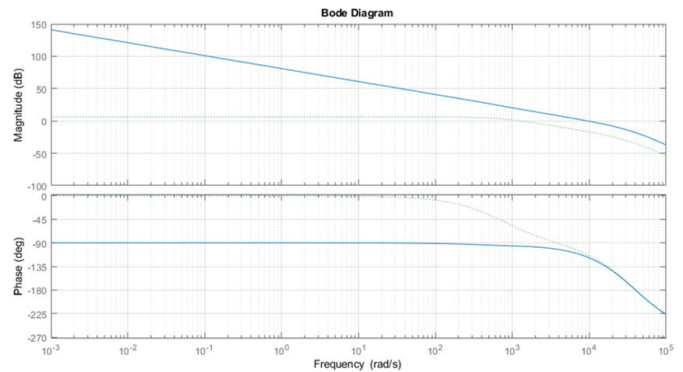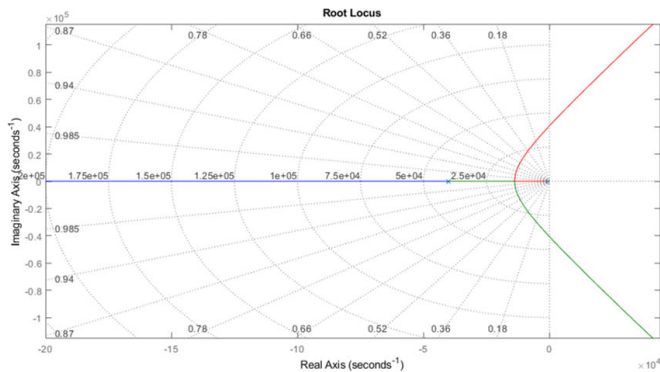
References

[1] *Ogata, K. (2010). Modern control engineering. Prentice hall.*

[2] *Leonhard, W. (2001). Control of electrical drives. Springer Science & Business Media. pp. 86.*

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 AC induction motors (ACIM). 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.

Note Motor Control Blockset examples use current reference (*Iq_ref*, instead of torque reference *T_ref*) as the speed controller output because of considerations related to the per-unit (PU) computations. The example algorithm selects the base values for current and torque (*Iq_base* and *T_base*) such that PU reference values of current and torque are identical (*Iq_ref_pu = T_ref_pu*).

Permanent Magnet Synchronous Motor (PMSM)

This figure shows the FOC architecture for a PMSM. For detailed set of equations and assumptions that Motor Control Blockset uses to implement FOC of a PMSM, see "Mathematical Model of PMSM".
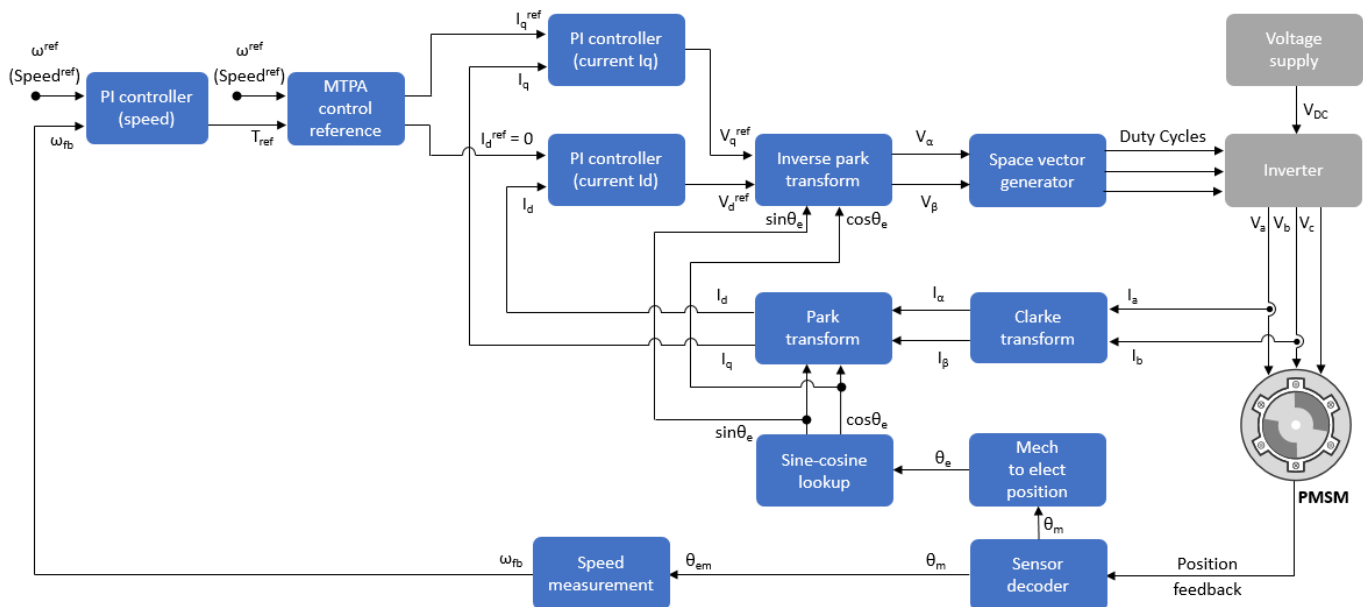
AC Induction Motor (ACIM)

This figure shows the FOC architecture for an AC induction motor (ACIM). For detailed set of equations and assumptions that Motor Control Blockset uses to implement FOC of an induction motor, see "Mathematical Model of Induction Motor".
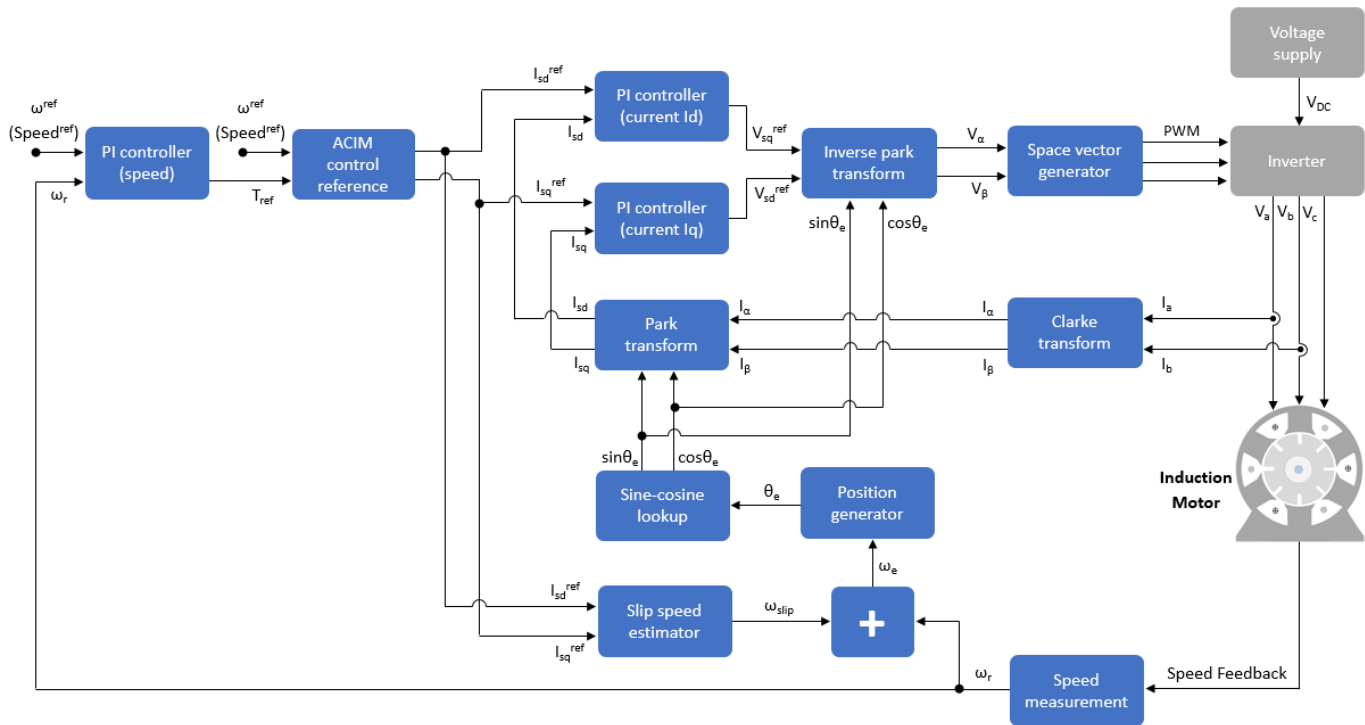
Six-Step Commutation

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.
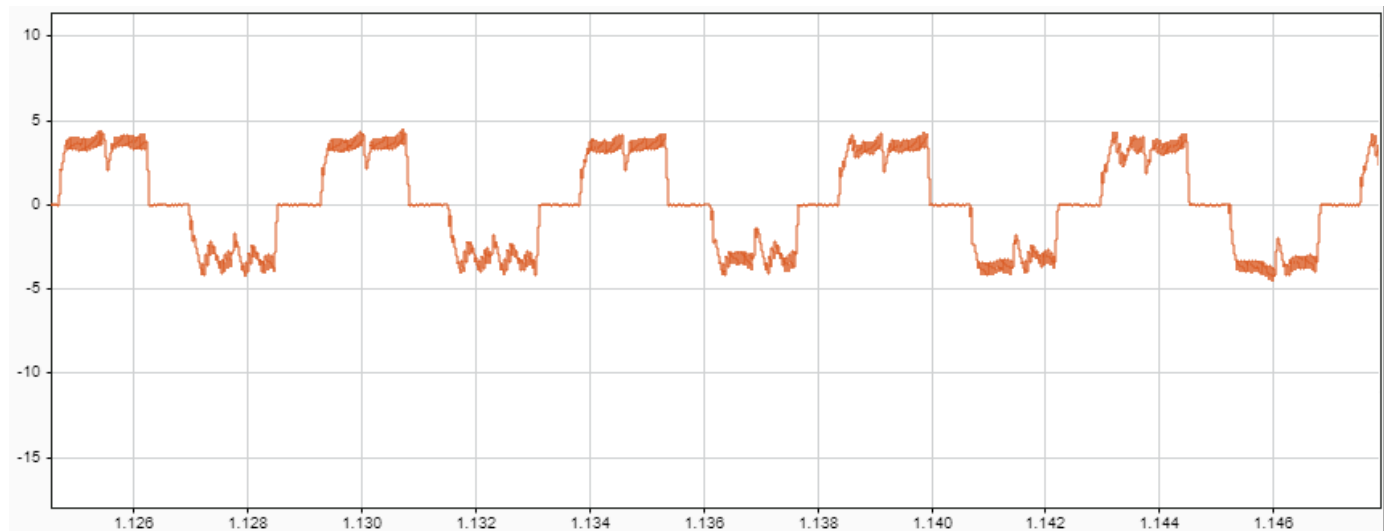
The 120 degree conduction mode is a less complex technique that provides good speed control for the BLDC motors. This figure shows the six-step commutation architecture for a BLDC motor.
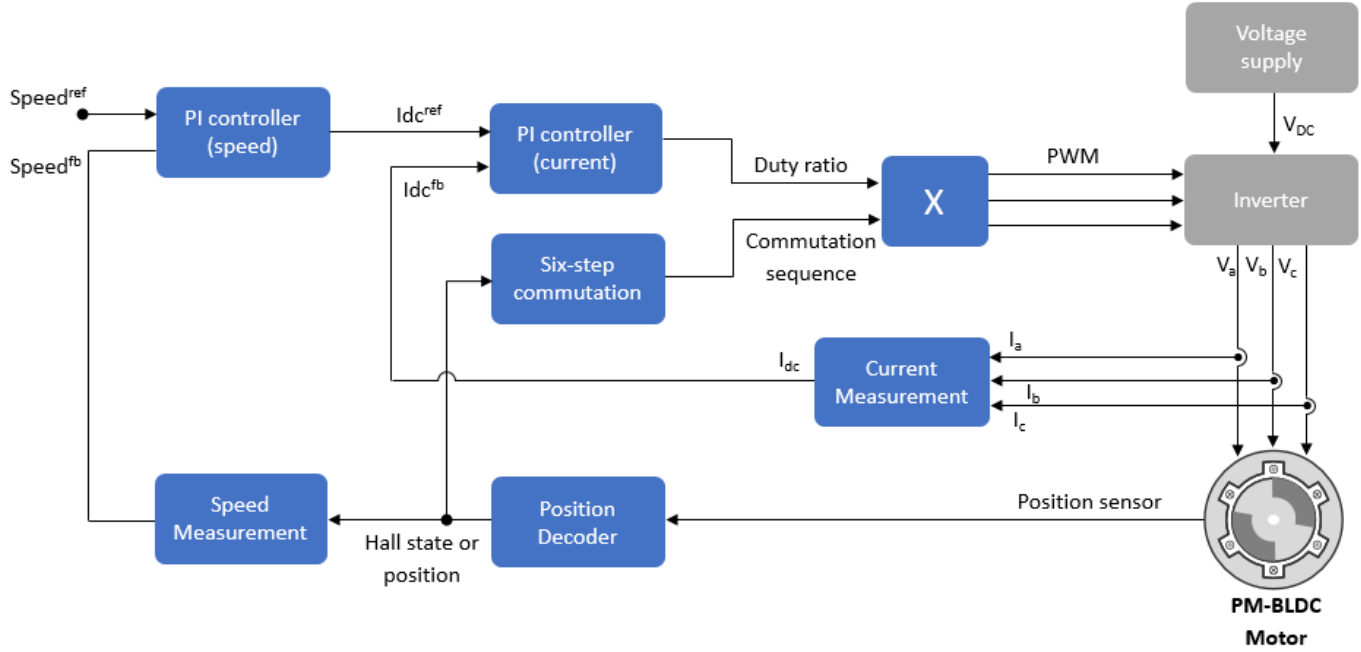
Direct Torque Control (DTC)

Direct Torque Control (DTC) is a vector motor control technique that implements motor speed control by directly controlling the flux and torque of the motor. Unlike field-oriented control (FOC) that controls *d*- and *q*-axis motor currents, the DTC algorithm estimates the torque and flux values from the motor position and currents. Then it uses PI controllers to control the motor torque and flux to eventually generate the optimum voltages that run the motor.

Motor Control Blockset uses the DTC space vector pulse-width modulation (DTC-SVPWM) variant to control a permanent magnet synchronous motor (PMSM). The technique uses space vector modulation (SVM) to produce the pulse-width modulation (PWM) duty cycles that are used by the inverter to generate the three-phase voltages that run the PMSM.

The DTC-SVPWM algorithm estimates the motor torque and flux feedback values using the current feedback (in the *α-β* reference frame) from the motor. The algorithm uses the motor speed feedback to compute the flux reference value. The speed PI controller (part of the outer control loop) uses the speed error input to compute the torque reference value. The flux and torque PI controllers (part of the inner control loop) use these flux and torque references and flux and torque feedback values to compute the *d*-axis and *q*-axis reference voltages. The algorithm uses the PWM Reference Generator block to generate PWM duty-cycles (using SVM) from these reference voltages.

You can determine current rotor position using both sensor-based or sensorless approaches.
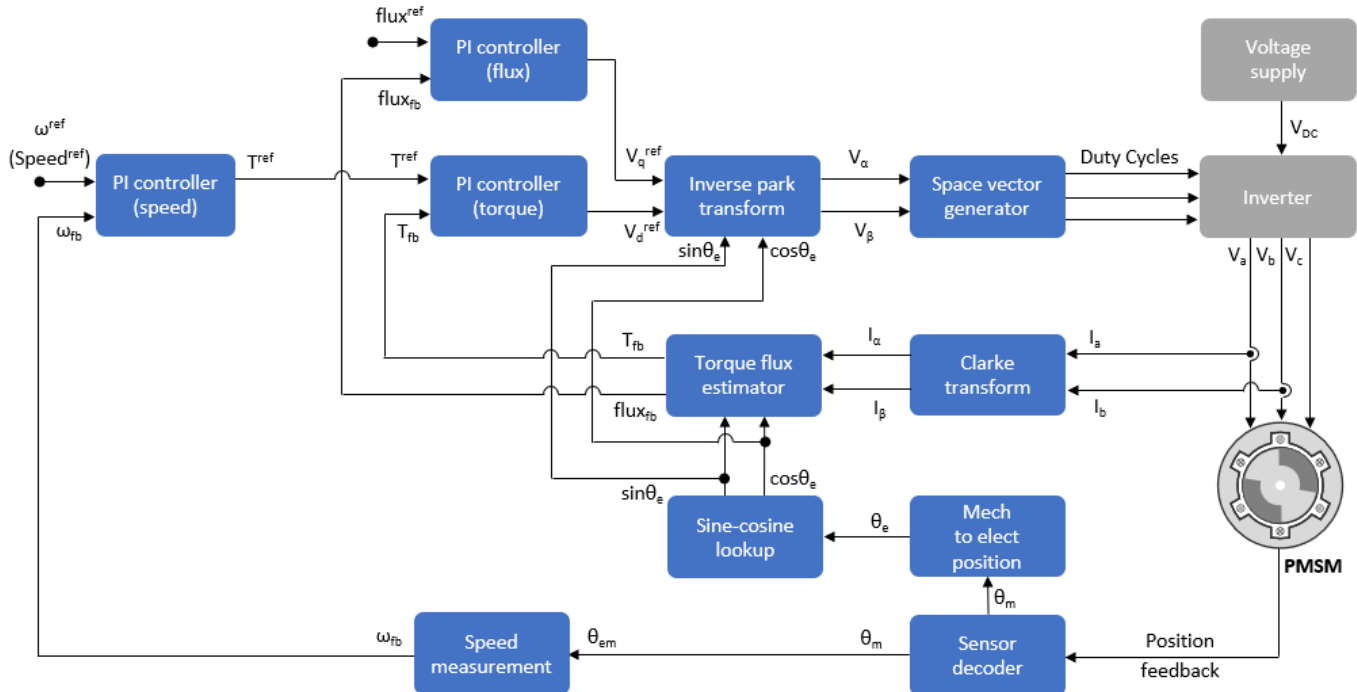
Flux and Torque Estimation

The DTC-SVPWM algorithm used by Motor Control Blockset uses these transient machine model equations to estimate flux and torque of a PMSM.

These equations describe flux estimation from the currents in the α-β reference frame and rotor position:

$$\psi_\alpha = L_s \cdot i_\alpha + \psi_{PM} \cdot \cos\theta$$

$$\psi_\beta = L_s \cdot i_\beta + \psi_{PM} \cdot \sin\theta$$

$$\psi = \sqrt{\psi_\alpha^2 + \psi_\beta^2}$$

These equations describe the per-unit (PU) computation of flux:

$$\psi_\alpha^{pu} = \left(\omega_{base} \cdot L_s^{pu} \cdot i_\alpha^{pu}\right) + \left(\psi_{PM}^{pu} \cdot \cos\theta\right)$$

$$\psi_\beta^{pu} = \left(\omega_{base} \cdot L_s^{pu} \cdot i_\beta^{pu}\right) + \left(\psi_{PM}^{pu} \cdot \sin\theta\right)$$

$$\psi^{pu} = \sqrt{\left(\psi_\alpha^{pu}\right)^2 + \left(\psi_\beta^{pu}\right)^2}$$

$$\omega_{base} = 2 \cdot \pi \cdot f_{base}$$

These equations describe torque estimation from the currents in the α-β reference frame:

$$T = \frac{3}{2} \cdot p \cdot \left(\psi_\alpha i_\beta - \psi_\beta i_\alpha\right)$$

These equations describe the per-unit (PU) computation of torque:

$$T^{pu} = \frac{1}{\psi_{PM}^{pu}} \cdot \left(\psi_\alpha^{pu} i_\beta^{pu} - \psi_\beta^{pu} i_\alpha^{pu}\right)$$

where:

- ψ is the rotor flux of PMSM (Weber).
- ψ^{pu} is the per-unit version of ψ (Weber).
- ψ_α is the rotor flux along the α-axis of the α-β reference frame (Weber).
- $\psi_\alpha{}^{pu}$ is the per-unit version of ψ_α (Weber).
- ψ_β is the rotor flux along the β-axis of the α-β reference frame (Weber).
- $\psi_\beta{}^{pu}$ is the per-unit version of ψ_β (Weber).
- ψ_{PM} is the permanent magnet flux linkage of the PMSM (Weber).
- $\psi_{PM}{}^{pu}$ is the per-unit version of ψ_{PM} (Weber).
- L_s is the stator inductance of the PMSM (Henry).
- $L_s{}^{pu}$ is the per-unit version of L_s (Henry).
- i_α is the motor current along the α-axis of the α-β reference frame (Amperes).
- $i_\alpha{}^{pu}$ is the per-unit version of i_α (Amperes).
- i_β is the motor current along the β-axis of the α-β reference frame (Amperes).
- $i_\beta{}^{pu}$ is the per-unit version of i_β (Amperes).
- θ is the rotor position (captured by a sensor or determined by sensorless position estimation) (Radians).

- ω_{base} is the mechanical base speed of the motor (Radians/ sec).
- f_{base} is the mechanical frequency of the motor (Hertz).
- T is the rotor torque (Nm).
- T^{pu} is the per-unit version of T (Nm).
- p is the number of pole pairs of the 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 these 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 also use the open_system command to open the Simulink® models. For example, use this command for a F28069M based controller:

```
open_system('mcb_open_loop_control_f28069M_DRV8312.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 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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

2. For the model: **mcb_open_loop_control_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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. You can also use the target model name 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 7-3.

• 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 7-6.

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

Prerequisites:
1. Deploy the target model to the hardware
   F28069m + DRV8312
   F28069m Launchpad
   F28379d Launchpad
2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Select hardware in **Target Selelction**. Select 'Other' option if you want to manually set the baud rate in **'Host Serial Setup'** block.
2. Select the port in Host Serial Setup, Host Serial Receive and Host Serial Transmit
3. Simulate this model
4. Use **'Motor Start / Stop'** switch to control the motor.
5. Enter speed request in RPM using 'Reference Speed' edit box. Limit the reference speed to half of the rated speed.
6. Observe the ADC counts for phase current measurent in Scope.
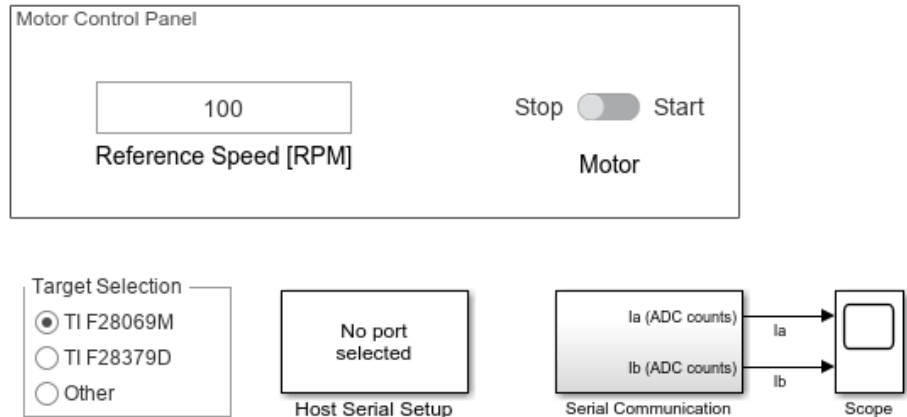
Motor Control Panel

100
Reference Speed [RPM]

Stop Start
Motor

Target Selection
◉ TI F28069M
○ TI F28379D
○ Other

No port selected
Host Serial Setup

Ia (ADC counts)   Ia
Ib (ADC counts)   Ib
Serial Communication   Scope

Copyright 2020-2021 The MathWorks, Inc.

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

8. Select a target (either `TI F28069M`, `TI F28379D`, or `Other`) in the **Target Selection** area of the host model.

NOTE: If you select `Other`, you can enter the **Baud rate** for the target hardware that you are using, in the Host Serial Setup block parameter dialog box.

9. In the host model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a Port.

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 model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a Port.

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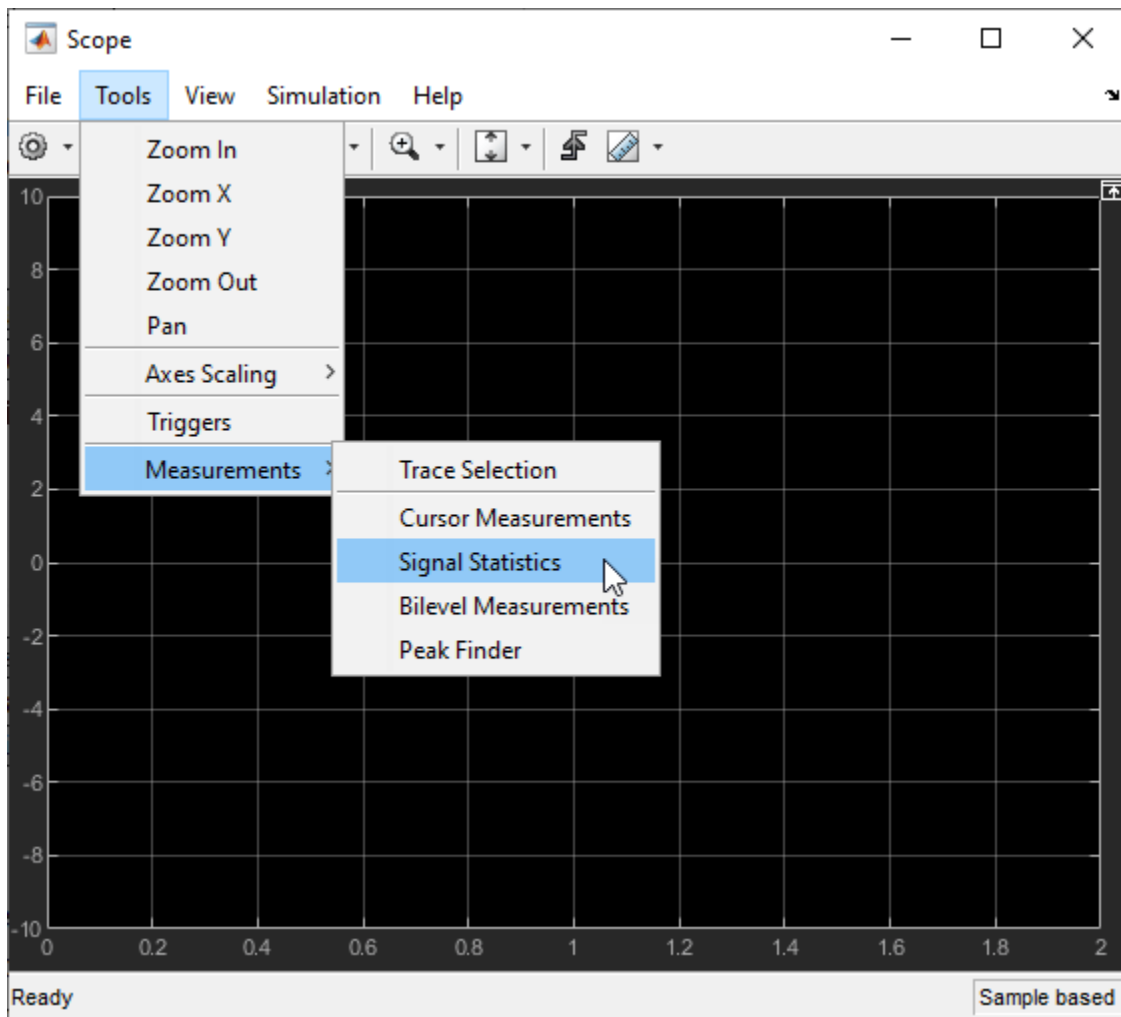
- Under **Trace Selection**, select a signal (I_a or I_b). The characteristics of the selected signal are displayed in the **Signal Statistics** pane. You can see the median value of the selected signal in the Median field.
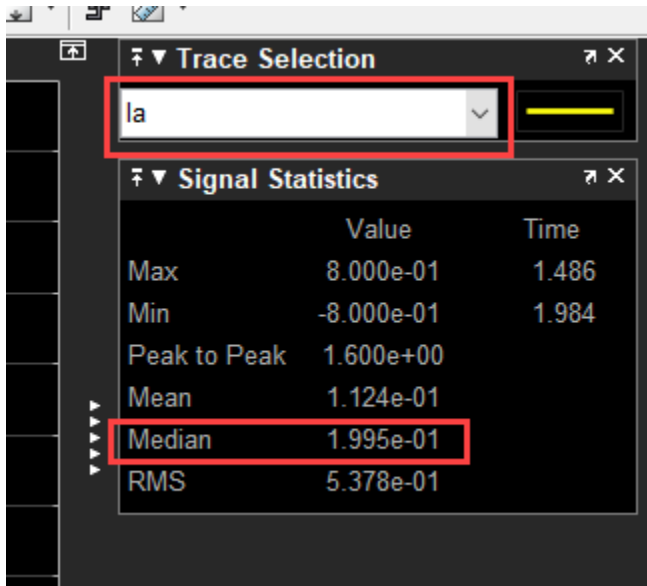
For the Motor Control Blockset examples, update the computed ADC (or current) offset value in the `inverter.CtSensAOffset` and `inverter.CtSensBOffset` variables in the model initialization script linked to the example. For instructions, see "Estimate Control Gains and Use Utility Functions" on page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

NOTE: The computed ADC offset depends on the ADC gain value `inverter.SPI_Gain_Setting` that you configure in the model initialization script. Changing ADC gain also changes the ADC offset.

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 4-3.

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 > Open-loop mode** area of the host model to change the output voltage of the inverter (in per-unit) and the rotor speed (in per-unit). 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 > Motor torque control mode** area of the host model to change the I_d reference and I_q reference current values (in per-unit). You can also set the maximum speed limit of the motor (in per-unit).

You can 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 > Motor speed control mode** 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 6-20.

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 the model mcb_pmsm_operating_mode_f28379d.

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

open_system('mcb_pmsm_operating_mode_f28379d.slx');

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

1. Motor Control Blockset™

2. Embedded Coder®

3. C2000™ Microcontroller Blockset

4. 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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

Follow these steps to simulate the model.

1. Open the target model included with this example.

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

3. Open the **mcb_pmsm_operating_mode_f28379d > Simulate Dashboard** subsystem.

Control Panel for Simulation

Control	Select Motor operating mode
	● Stop
	○ Open loop run
	○ Torque control
	○ Speed control

Every run default values are updated from init script.
To update the default values in dashboard, update the values in init script.

Operating Mode Variables

Open-loop mode

0.15	0.15
Voltage ref (PU)	Speed ref (PU)

Motor torque control mode

0	0	0.6	Unlock ⬤ Pos lock
Id Reference	Iq Reference	Speed limit	

Motor speed control mode

0.2
Speed Reference

Control loop gains

d-axis current controller

3.0929	5634.3199999999997
Kp Gain	Ki Gain

q-axis current controller

3.0929	5634.3199999999997
Kp Gain	Ki Gain

Speed controller

0.91894	24.264399999999998
Kp Gain	Ki Gain

Serial Communication
1 → 1 msgId
2 → 2 msgData

Instructions for Open-Loop Run Mode:

1. If the current operating mode is other than open-loop run, select **Stop** in the **Control** area to stop the motor. Select **Open loop run** to start the motor.

2. Set the reference voltage and reference speed values (in per-unit) in the **Voltage ref (PU)** and **Speed ref (PU)** fields available in the **Operating Mode Variables > Open-loop mode** area.

Instructions for Torque Control Mode:

1. If the current operating mode is other than torque control, select **Stop** in the **Control** area to stop the motor. Select **Torque control** in the **Control** area.

2. Enter the value 0 (per-unit) in the **Iq Reference** field in the **Operating Mode Variables > Motor torque control mode** area. In addition, set the speed limit of the motor using the **Speed limit** field.

3. Move the slider switch to **Unlock** position in the **Operating Mode Variables > Motor torque control mode** area.

4. 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.

5. Open Simulation Data Inspector and select the **Iq_ref_PU** and **Iq_fb_PU** signals for monitoring.

6. Follow steps 2 to 5 for **Id Reference** and monitor the **Id_ref_PU** and **Id_fb_PU** 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. If the current operating mode is other than speed control, select **Stop** in the **Control** area to stop the motor. Select **Speed control** in the **Control** area.

2. Enter the value 0.5 (per-unit) in the **Speed Reference** field in the **Operating Mode Variables > Motor speed control mode** area.

3. Open Simulation Data Inspector and select the **Speed_ref_PU** and **Speed_fb_PU** signals for monitoring.

Instructions for Tuning Gain of Torque Controller:

1. If the current operating mode is other than torque control, select **Stop** in the **Control** area to stop the motor. Select **Torque control** in the **Control** area.

2. Turn the slider switch to **Pos lock** position in the **Operating Mode Variables > Motor torque control mode** area.

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

4. Open Simulation Data Inspector, select the **Id_ref_PU** and **Id_fb_PU** signals, and observe the step response of these signals.

5. Tune the control gains Kp and Ki for the d-axis current controller. Perform step change to validate the controller gains.

Instructions for Tuning Gain of Speed Controller:

1. If the current operating mode is other than speed control, select **Stop** in the **Control** area to stop the motor. Select **Speed control** in the **Control** area.

2. Enter the value 0.5 (per-unit) in the **Speed Reference** field in the **Operating Mode Variables > Motor speed control mode** area.

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

4. Open Simulation Data Inspector, select the **Speed_ref_PU** and **Speed_fb_PU** signals, and observe the speed step response.

5. Tune the control gains Kp and Ki for the speed controller. Perform step change to validate the controller gains.

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

The example supports this hardware configuration. You can also use the target model name 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 7-6.

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 4-10.

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 4-79.

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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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.

7. Set the reference voltage and reference speed values (in per-unit) in the **Voltage ref (PU)** and **Speed ref (PU)** fields available in the **Operating Mode Variables > Open-loop mode** area.

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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 Ref (PU)** and **Iq Ref (PU)** fields in the **Operating Mode Variables > Motor torque control mode** area. In addition, set the speed limit of the motor using the **Speed limit (PU)** field.

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

8. Move the slider switch to **Unlock** position in the **Operating Mode Variables > Motor torque control mode** 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 Ref (PU)** 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 Ref (PU)** field in the **Operating Mode Variables > Motor speed control mode** 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 > Motor torque control mode** 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 Ref (PU)** 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 **Control** 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 Ref (PU)** field in the **Operating Mode Variables > Motor speed control mode** 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 Ref (PU)** 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. Open the target model included with this example.

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

3. Open the **mcb_pmsm_operating_mode_f28379d > Simulate Dashboard** subsystem.

4. If the current operating mode is other than speed control, select **Stop** in the **Control** area to stop the motor. Select **Speed control** in the **Control** area.

5. Enter the value `0.2` (per-unit) in the **Speed Reference** field in the **Operating Mode Variables > Motor speed control mode** area.

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

7. Open Simulation Data Inspector, select the **Speed_ref_PU** and **Speed_fb_PU** signals, and observe the speed step response.

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:

```
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. Set the serial port to **target.comport** in the
   init script of the target model.
2. Caution: Stop the motor when switching
   between the modes

target.comport
Baud rate : 12e

Host Serial Setup

Scope →

Serial Communication
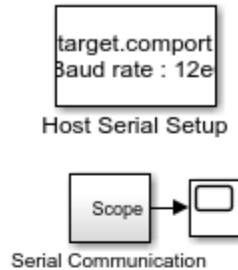
Control
Select Motor operating mode
- (●) Stop
- ( ) Open loop run
- ( ) Torque control
- ( ) Speed control

Operating Mode Variables

Open-loop mode

| 0.2 | 0.15 |

Voltage ref (PU)   Speed ref (PU)

Motor torque control mode

| 0 | 0 | 0.6 | Unlock ⬤ Pos lock

Id Ref (PU)   Iq Ref (PU)   Speed limit (PU)

Motor speed control mode

| 0.2 |

Speed Ref (PU)

Control loop gains

d-axis current controller

| 2.1651 | 4131.6629000000 |

Kp Gain   Ki Gain

q-axis current controller

| 2.1651 | 4131.6629000000 |

Kp Gain_   Ki Gain_

Speed controller

| 0.9106 | 24.0901 |

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

Copyright 2020-2022 The MathWorks, Inc.

10. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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

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

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

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

15. Enter the value 0.2 (per-unit) in the **Speed Ref (PU)** field in the **Operating Mode Variables > Motor speed control mode** area.

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

17. Enter the value 0.5 (per-unit) in the **Speed Ref (PU)** field.

18. Observe the speed step response in the scope.

19. Compare the speed step responses obtained in steps 7 (with simulation) and 18 (with code generation).

NOTE: 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 Using Field Oriented Control Autotuner

This example computes the gain values of PI controllers available in the speed and current control loops by using the Field Oriented Control Autotuner block. For details about this block, see Field Oriented Control Autotuner. For details about field-oriented control, see "Field-Oriented Control (FOC)" on page 4-3.

Use the code-generation capability of the example to deploy the gain-tuning algorithm to the target hardware. This enables you to run the algorithm by using hardware connected to a motor and to compute accurate PI controller gains by processing motor feedback in real-time on the target hardware. The example uses a quadrature encoder sensor to measure the rotor position.

Note: This example uses the field-oriented control algorithm as a reference. You can refer this example and use a similar approach to add the Field Oriented Control Autotuner block and the gain-tuning algorithm to the FOC logic available in your model.

Model

The example includes the target model mcb_pmsm_foc_autotuner_f28379d.

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

open_system('mcb_pmsm_foc_autotuner_f28379d.slx');

The Field Oriented Control Autotuner block iteratively tunes the d- and q-axis current control and speed control loops and computes the gains of the current and speed PI controllers. Use this command to locate the Field Oriented Control Autotuner block available inside the model:

open_system('mcb_pmsm_foc_autotuner_f28379d/Current Control/Control_System/Closed Loop Control/FO

The block processes the current and speed feedback from the plant. It also processes the voltage output of the d- and q-axis current PI controllers to compute the PI controller gains (Kp and Ki).

For more details on the FOC architecture, see "Field-Oriented Control (FOC)" on page 4-3.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Simulink Control Design™
- Stateflow® (needed only if you modify the example model)

To generate code and deploy model:

- Motor Control Blockset
- Simulink Control Design
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Stateflow (needed only if you modify the example model)

Prerequisites for Simulation and Hardware Deployment

1. Open the model initialization script for the target model. Check and update the motor, inverter, and other control system and hardware parameters available in the script. For instructions on locating and editing the model initialization script associated with a target model, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

2. In the **Inverter & Target Parameters** section of the model initialization script, verify that the `mcb_SetInverterParameters` function uses the argument `BoostXL-DRV8305`. This enables the script to use the preprogrammed parameters for the BOOSTXL-DRV8305 inverter.

3. Configure these parameters correctly in the model initialization script. These variables are essential for the gain-tuning algorithm to compute the PI controller gains. If the values of these variables are incorrect, the model may fail to bring the motor to the steady speed state.

- `pmsm.p`

- `pmsm.I_rated`

- `pmsm.PositionOffset`

- `pmsm.QEPSlits`

4. If you are using a motor that is not listed in the `mcb_SetPMSMMotorParameters` function (used in the **System Parameters // Hardware parameters** section of the model initialization script), tune the default values of the following initial gains available in the **Initial PI parameters** section of the model initialization script. This ensures that the motor reaches the steady state of speed-control operation:

- `PI_params.Kp_Id`

- `PI_params.Ki_Id`

- `PI_params.Kp_Iq`

- `PI_params.Ki_Iq`

- `PI_params.Kp_Speed`

- `PI_params.Ki_Speed`

When you either simulate or run the example on a target hardware, the example uses crude values of the PI controller gains to achieve the steady state of speed-control operation.

Note: When using this example, if the motor (whether it is listed or not in the `mcb_SetPMSMMotorParameters` function) does not run, try tuning the default values of these parameters.

5. In the **FOC Autotuner parameters** section of the model initialization script, check and update the parameters of the Field Oriented Control Autotuner block. This sets the reference bandwidth and phase margin values for both the speed and the current PI controllers.

Simulate the Target Model

Simulating the example is optional. Follow these steps to simulate the target model:

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 `mcb_pmsm_foc_autotuner_f28379d/Current Control/PI_Params_Display_and_Logging` subsystem.

The computed gains might not be accurate because step 3 in the Prerequisites for Simulation and Hardware Deployment section checks the accuracy of only four motor parameters.

If you want to compute and test the PI controller gains using simulation, follow these steps before clicking **Run** on the **Simulation** tab of the target model.

- In the **System Parameters // Hardware parameters** section of the model initialization script, verify that the `mcb_SetPMSMMotorParameters` function uses an argument that represents your

motor (for example, `Teknic2310P`). Open the `mcb_SetPMSMMotorParameters` function to see the preprogrammed cases that store the motor parameters of commonly used PMSMs.

If the `mcb_SetPMSMMotorParameters` function does not list your PMSM, determine the parameters for your motor using these steps:

- If you have motor control hardware, you can estimate the parameters for your motor, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate PMSM Parameters Using Recommended Hardware" on page 4-186 and "Estimate PMSM Parameters Using Custom Hardware" on page 4-209.

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

- If you obtain the motor parameters from the datasheet or other sources, add and configure the motor parameters in the model initialization script. These parameter values override the selected pre-programmed case in the function `mcb_SetPMSMMotorParameters`.

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

After you simulate the target model and determine the gains, update your model (that implements FOC) with the computed gain values to quickly bring the motor to a steady speed state.

Deploy the example to the target hardware to tune the PI controller gains more accurately by using an actual hardware connected to a motor. For more details, see the Generate Code and Deploy Model to Target Hardware section.

Generate Code and Deploy Model to Target Hardware

This section shows how to generate code and run the algorithm for tuning the PI controller gains on the target hardware. Running the example on the hardware enables you to compute the PI controller gains more accurately by processing the feedback from an actual plant in real-time.

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. Before you run the host model on the host computer, 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 closed-loop control.

Required Hardware

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB® command prompt.

LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_pmsm_foc_autotuner_f28379d

For more information on connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. The model automatically computes the analog to digital converter (ADC) offset (also known as current offset). To disable this functionality (enabled by default), update the value of the `inverter.ADCOffsetCalibEnable` variable in the model initialization script to `0`.

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 4-10.

3. Compute the quadrature encoder index offset value and update it in the `pmsm.PositionOffset` variable in the model initialization script of the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79.

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

5. Load a sample program to the CPU2 of the LAUNCHXL-F28379D board. For example, load the program that operates the CPU2 blue LED by using GPIO31 (`c28379D_cpu2_blink.slx`). This ensures 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_host_autotuner_f28379d.slx');
```


PMSM Field Orient Control Autotuner Host

Copyright 2021-2022 The MathWorks, Inc.

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

8. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

9. Turn the **Motor** slider switch to the **Start** position to start running the motor.

10. Update the reference speed value in the **Speed Ref [RPM]** field. It is recommended that you use a value that is approximately half the rated speed of the motor.

11. In the **Debug signals** section, select **Speed_Ref & Speed_Feedback** and monitor the speed signals in the **SelectedSignals** time scope. Wait until the motor reaches a steady speed.

The example can begin tuning only in the steady speed state.

12. Check that the **PI Parameters** slider switch is in the **Autotuner** position.

13. Turn the **Autotuner** slider switch to the **Start** position to start autotuning the PI controller gains. The tuning process introduces perturbations depending on the controller goals (bandwidth and phase margin) in the controller output. The example uses the system response to the perturbations to calculate the optimal controller gain values.

The model performs these tests iteratively on the motor and determines an accurate set of `Kp` and `Ki` gains for the current and speed PI controllers.

The **Tuning Status** display changes status from `Tuning not started` to `Tuning in progress`.

Note: When tuning is in progress, ensure that the **PI Parameters** slider switch remains in the **Autotuner** position.

14. When the tuning process successfully completes, the **Tuning Status** display changes status from `Tuning in progress` to `Tuning complete`.

The target model updates the speed and current PI controllers running on the target hardware with the computed `Kp` and `Ki` gains. In addition, the host model displays these values.

15. If the gain-tuning algorithm encounters an error during the tuning process, the **Tuning Status** display shows `Tuning failed`. Turn the **Autotuner** slider switch to the **Stop** position and see the **Troubleshooting** section for the troubleshooting instructions.

16. If you have successfully completed the tuning process, turn the **Autotuner** slider switch to the **Stop** position.

17. Validate the computed gain values. For instructions, see the **Validate Computed PI Controller Gains** section.

Validate Computed PI Controller Gains

1. Check that the motor is running and that the **PI Parameters** slider switch is in the **Autotuner** position.

2. Select the **Speed_Ref & Speed_Feedback** debug signal in the **Debug signals** section of the host model.

3. Open the **SelectedSignals** time scope to monitor the reference speed and speed feedback signals.

4. Update the reference speed (for your motor control application) in the **Speed Ref [RPM]** field and monitor the signals in the time scope.

5. In the **SelectedSignals** window, navigate to **Tools > Measurements** and select **Cursor Measurements** to display the **Cursor Measurements** area.

6. Drag cursor-1 to a position that indicates zero `Speed_Ref` (just before Speed_ref rises). Drag cursor-2 to a position where `Speed_Feedback` meets `Speed_Ref` for the first time.

ΔT indicates the actual response time of the FOC algorithm (time taken by the motor to reach 100% of the reference speed from zero reference speed).

7. For the speed PI controller, use the `PI_params.SpeedBW` variable available in the model initialization script to determine the bandwidth of the speed PI controller. Compute the theoretical response time using this relation:

$$Response\_time = \left(\frac{2}{PI\_params.SpeedBW} \right)$$

Compare the theoretical `Response_time` with the actual response time ΔT to validate the speed PI controller gains.

Similarly, you can validate the current PI controller gains by analyzing the step responses of the **d** and **q** current PI controllers.

Troubleshooting

Follow these steps to troubleshoot failed gain-tuning instances.

1. Identify the loop (either **d** current, **q** current, or speed) for which the tuning process failed.

The target model tunes the PI controllers in this sequence:

d current controller → **q** current controller → speed controller

Tuning failure of one controller in this sequence results in incorrect gain-tuning for the subsequent controllers.

Check the computed gains for the three controllers using the Display blocks available in the host model. A zero `Kp` or `Ki` controller gain value indicates that the tuning process failed for the respective controller.

Follow the subsequent steps for the first PI controller in the preceding sequence for which the tuning failed.

2. Select the controller reference and feedback signals for the controller identified in step 1, in the **Debug signals** section (for example, **Iq_Ref & Iq_Feedback** for the **q** current controller) and open the **SelectedSignals** time scope.

3. Check that the **PI Parameters** slider switch is in the **Autotuner** position.

4. Turn the **Autotuner** slider switch to the **Start** position to run the tuning process again.

5. Monitor the feedback signal for the controller identified in step 1 (for example, `Iq_Feedback`) in the **SelectedSignals** time scope.

Case 1: Follow these steps if the peak value of the controller feedback signal satisfies one of these conditions:

- Value is too high (greater than 1)

- Value is too low (less than `PI_params.CurrentSineAmp` for the current controllers or less than `PI_params.SpeedSineAmp` for the speed controller)

Note: The `PI_params.CurrentSineAmp` and `PI_params.SpeedSineAmp` variables are defined in the model initialization script.

a. If the controller identified in step 1 is the d or the q current controller, modify the `PI_params.CurrentSineAmp` variable such that it is less than the peak value of the controller feedback signal.

b. If the controller identified in step 1 is the speed controller, modify the `PI_params.SpeedSineAmp` variable such that it is less than the peak value of the controller feedback signal.

c. Turn the **Autotuner** slider switch to the **Stop** position and then to the **Start** position to run the tuning process again.

Case 2: Follow these steps if the peak value of the controller feedback signal lies in the range:

- $[PI\_params.CurrentSineAmp, 1]$ (for the current controllers)

- $[PI\_params.SpeedSineAmp, 1]$ (for the speed controller)

Note: The `PI_params.CurrentSineAmp` and `PI_params.SpeedSineAmp` variables are defined in the model initialization script.

a. Update the parameters of the Field Oriented Control Autotuner block (that set the reference bandwidth and phase margin values) available in the **FOC Autotuner parameters** section of the model initialization script.

b. Turn the **Autotuner** slider switch to the **Stop** position and then to the **Start** position to run the tuning process again.

Field-Oriented Control of PMSM 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 4-3.

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.
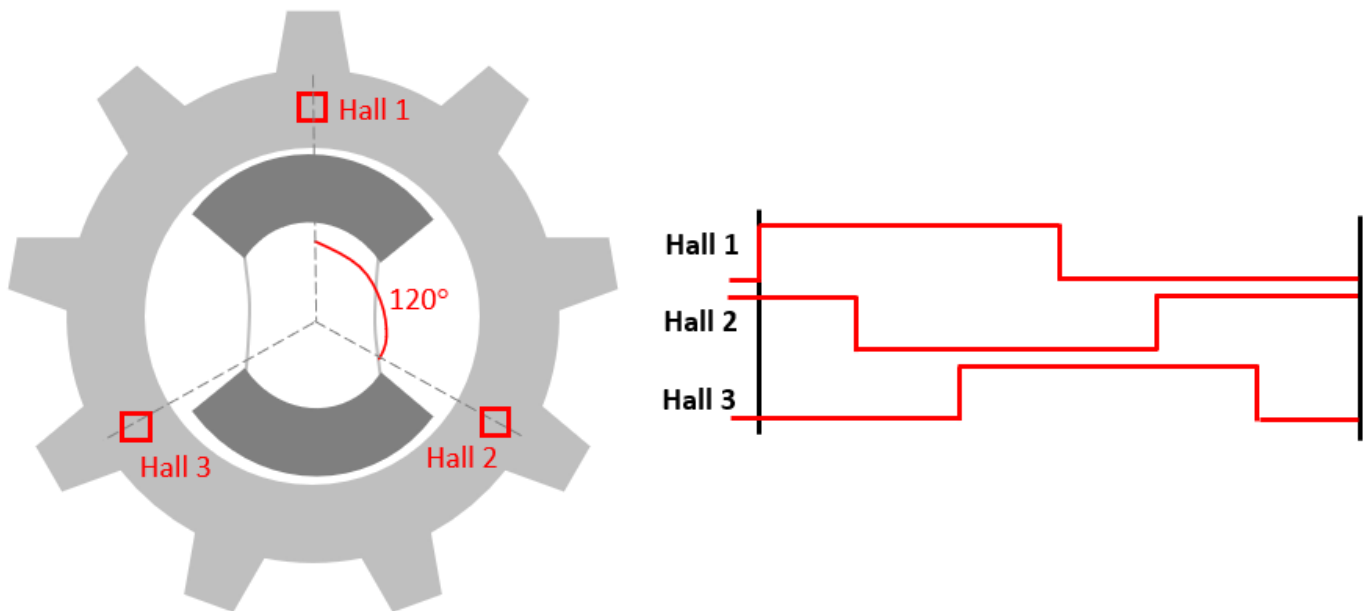
Models

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 also 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');
```

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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_foc_hall_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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. You can also use the target model name 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 7-3.

- 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 7-6.

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 4-10.

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 4-69.

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_host_model_f28069m.slx');
```

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_foc_qep_f28069m
   mcb_pmsm_foc_qep_f28069LaunchPad
   mcb_pmsm_foc_hall_f28069m
   mcb_pmsm_foc_sensorless_f28069MLaunchPad
   mcb_bldc_sixstep_f28069mLaunchPad

2. The variables from the target model are loaded to the base workspace.

Steps:
1. Set serial port to **target.comport** in the init script of the target model.
2. Use **Motor** switch to enable and disable motor control.
3. Input speed request using 'Reference Speed' edit box or sliding bar.
4. Observe the actual speed of motor and phase A current in the scope.
5. Start the motor in open loop under no load condition and transition to close loop for Sensorless Example. The model works in open loop for speed ref below 0.1pu.

PMSM / BLDC Control

Reference Speed [--]   Off ◯ On   Motor

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

target.comport
Baud rate : 5.625e6
Host Serial Setup

Speed (RPM)
Ia (amps)
Serial Communication

Copyright 2020-2022 The MathWorks, Inc.

4-41

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

9. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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.

NOTE: When you run this example on the hardware at a low Reference Speed, due to a known issue, the PMSM may not follow the low Reference Speed.

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 4-3.

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

You can use these models for both simulation and code generation. You can also use the open_system command to open the Simulink® models. For example, use this command for a F28069M based controller.
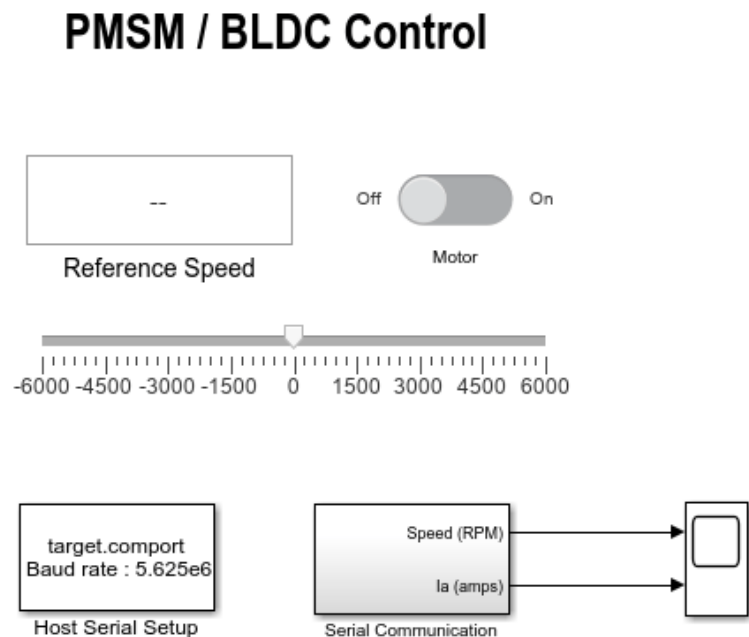
```
open_system('mcb_pmsm_foc_qep_f28069m.slx');
```

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 models: **mcb_pmsm_foc_qep_f28069m** and **mcb_pmsm_foc_qep_f28069LaunchPad**

- Motor Control Blockset™
- Fixed-Point Designer™

2. For the model **mcb_pmsm_foc_qep_f28379d**

- 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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

2. For the model **mcb_pmsm_foc_qep_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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. You can also use the target model name 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 7-3.

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

- 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 7-6.

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 4-10.

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 4-79.

NOTE: Verify the number of slits available in the quadrature encoder sensor attached to your motor. Check and update the variable `pmsm.QEPSlits` available in the model initialization script. This variable corresponds to the **Encoder slits** parameter of the quadrature encoder block. For more details about the **Encoder slits** and **Encoder counts per slit** parameters, see Quadrature Decoder.

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.
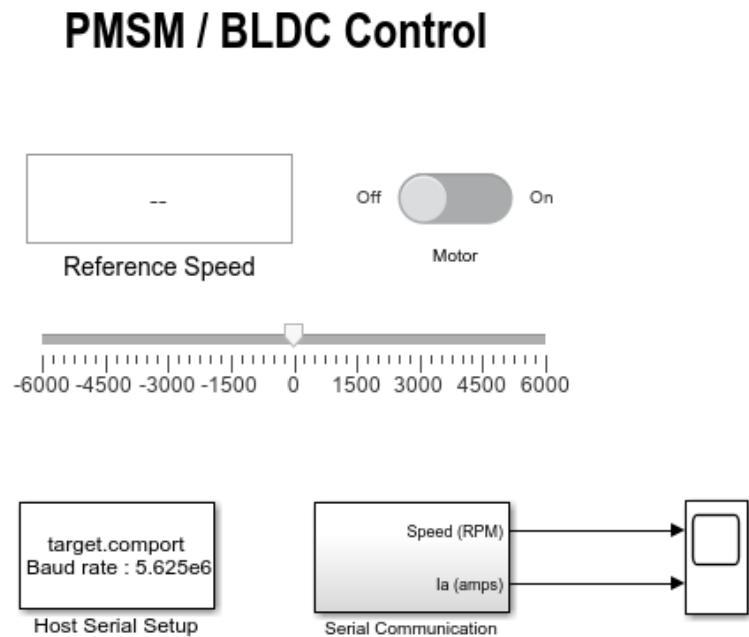
```
open_system('mcb_host_model_f28069m.slx');
```

Prerequisites:

1. Deploy the target model to the hardware
   mcb_pmsm_foc_qep_f28069m
   mcb_pmsm_foc_qep_f28069LaunchPad
   mcb_pmsm_foc_hall_f28069m
   mcb_pmsm_foc_sensorless_f28069MLaunchPad
   mcb_bldc_sixstep_f28069mLaunchPad

2. The variables from the target model are loaded to the base workspace.

Steps:

1. Set serial port to **target.comport** in the init script of the target model.
2. Use **Motor** switch to enable and disable motor control.
3. Input speed request using 'Reference Speed' edit box or sliding bar.
4. Observe the actual speed of motor and phase A current in the scope.
5. Start the motor in open loop under no load condition and transition to close loop for Sensorless Example. The model works in open loop for speed ref below 0.1pu.

PMSM / BLDC Control

Reference Speed

Motor — Off / On

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

target.comport
Baud rate : 5.625e6

Host Serial Setup

Speed (RPM)
Ia (amps)

Serial Communication

Copyright 2020-2022 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 4-3.

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.
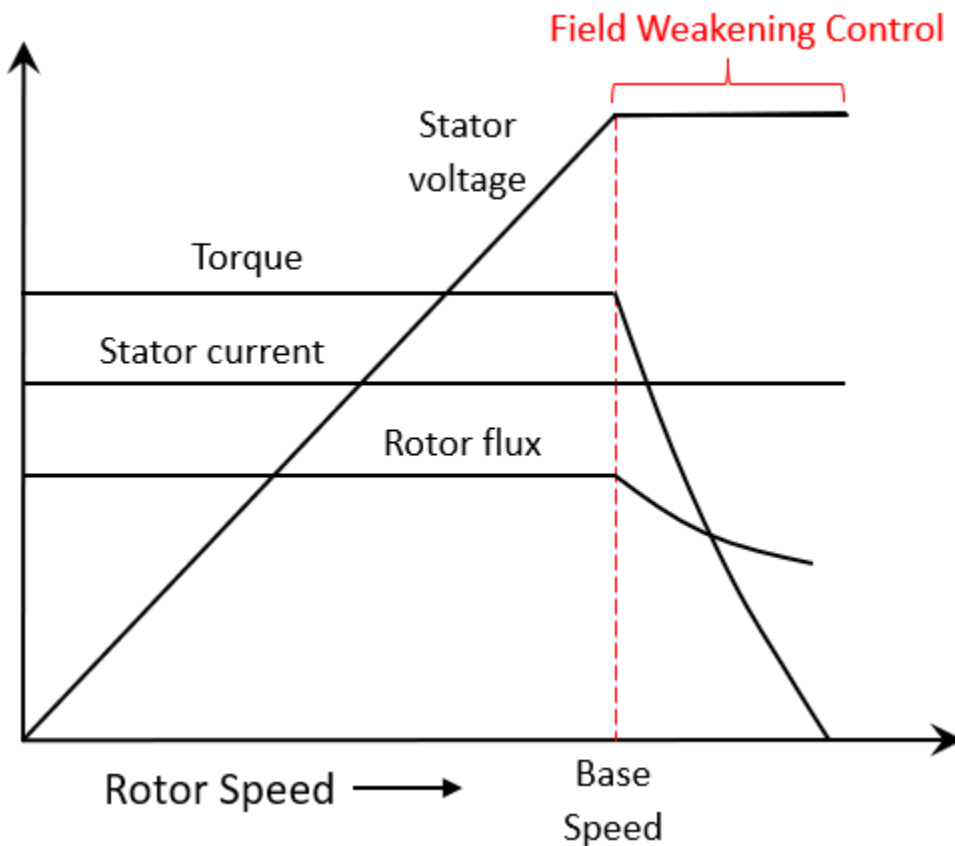
Depending upon the connected load and rated current of the machine, the reference d-axis current (I_d) in the field-weakening control also limits the reference q-axis current (I_q), and therefore, limits the torque output. Therefore, the motor operates in the constant torque region until the base speed. 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).
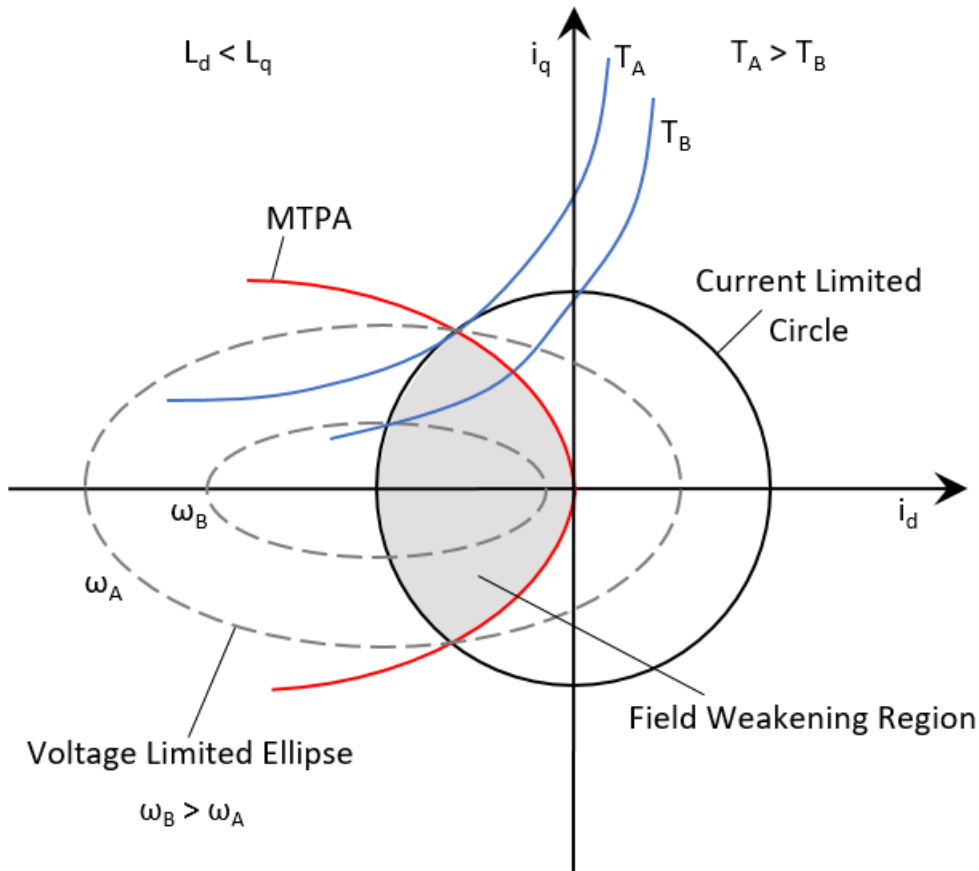
For an Interior PMSM (IPMSM), this example computes the reference I_d and I_q currents using the MTPA method until the base speed. For a Surface PMSM (SPMSM), the example achieves MTPA operation by using a zero d-axis reference current, until the base speed.

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.

Both field-weakening control and MTPA require generation of reference currents that follow the limitations related to:

- Current limited circle

- Voltage limited ellipse

- Motor temperature

To determine the operating point that follows these limits, see the plot generated by the function "Obtain Motor Characteristics" on page 3-17.

In the field-weakening region, some PMSMs may need a stator current that is higher than the rated current of the motor. For details, see the plot generated by the function "Obtain Motor Characteristics" on page 3-17.

Models

This example 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

Speed control of Interior PMSM (IPMSM) with field-weakening and MTPA:

- mcb_ipmsm_fwc_qep_f28379d

Note: This model uses the ADLEE-BM-180E IPMSM parameters that are defined in the model initialization script. ADLEE-BM-180E IPMSM has a saliency of approximately 10% (L_q is approximately 10% higher than L_d). Because of low saliency, this motor demands higher I_d currents to enter the field-weakening region and run at speeds higher than the rated speed. However, the motor has a rated current of only 9A. Therefore, when you run this motor in the field-weakening region, the low saliency makes the motor draw high I_d currents quickly (and quickly reach the rated curent limit) while gaining only a limited speed increase above the base speed. You can use this model to achieve higher speeds above the base speed by using an IPMSM that has a higher saliency.
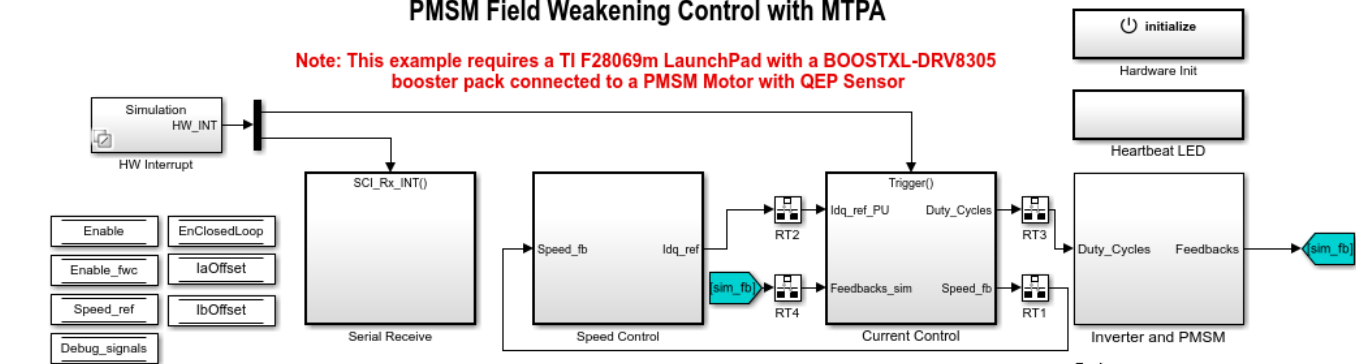
Torque control of PMSM with MTPA:

- mcb_pmsm_mtpa_qep_f28069LaunchPad

- mcb_pmsm_mtpa_qep_f28379d

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

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

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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

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

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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.
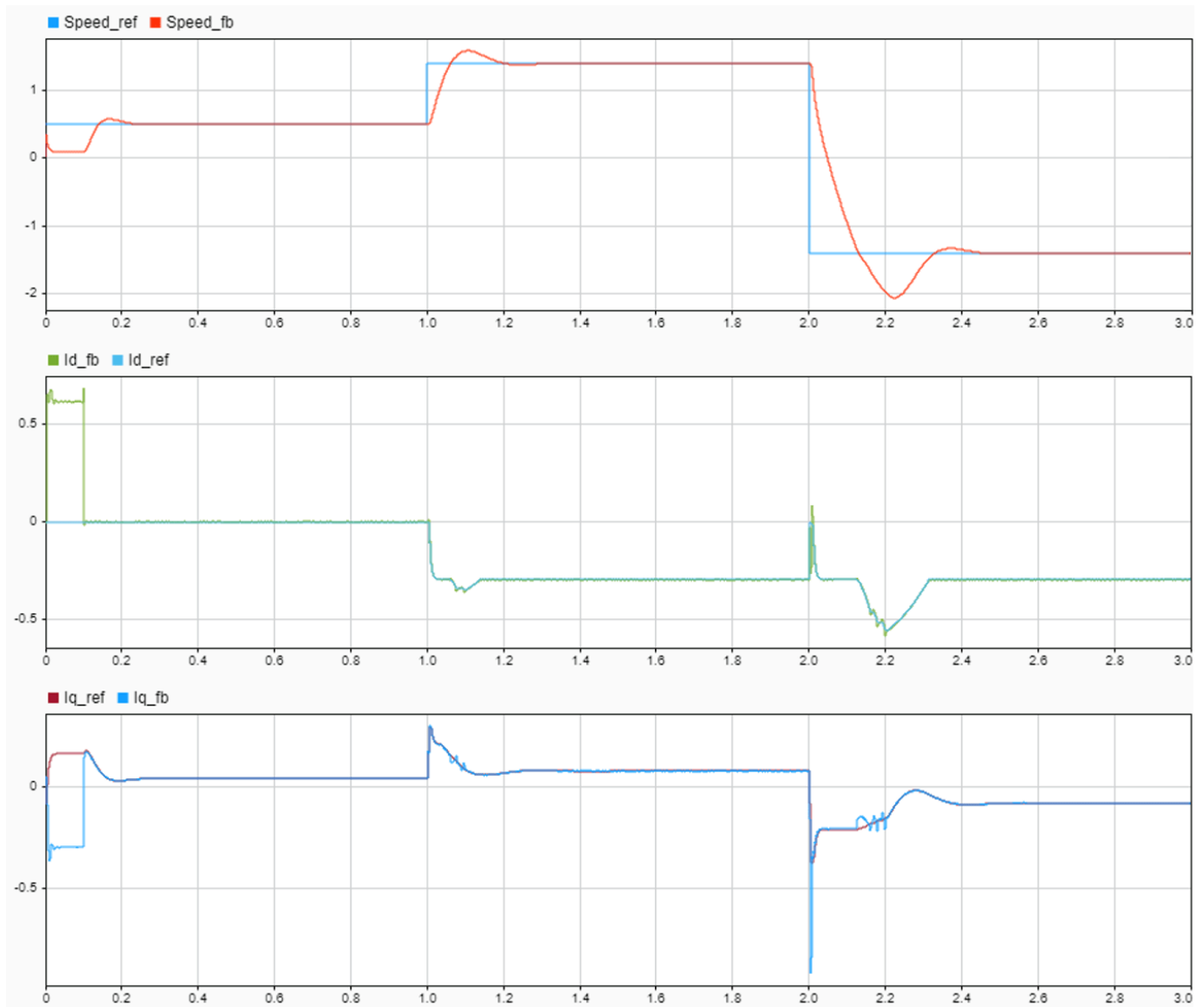
Note: In the beginning, the example runs the motor in open-loop control. After it detects the index pulse of the quadrature encoder sensor, the motor starts running using a closed-loop control. A start-up algorithm takes approximately 0.5 seconds to perform this transition. Ignore any transients observed in the speed and position feedback signals during this initial period.

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.
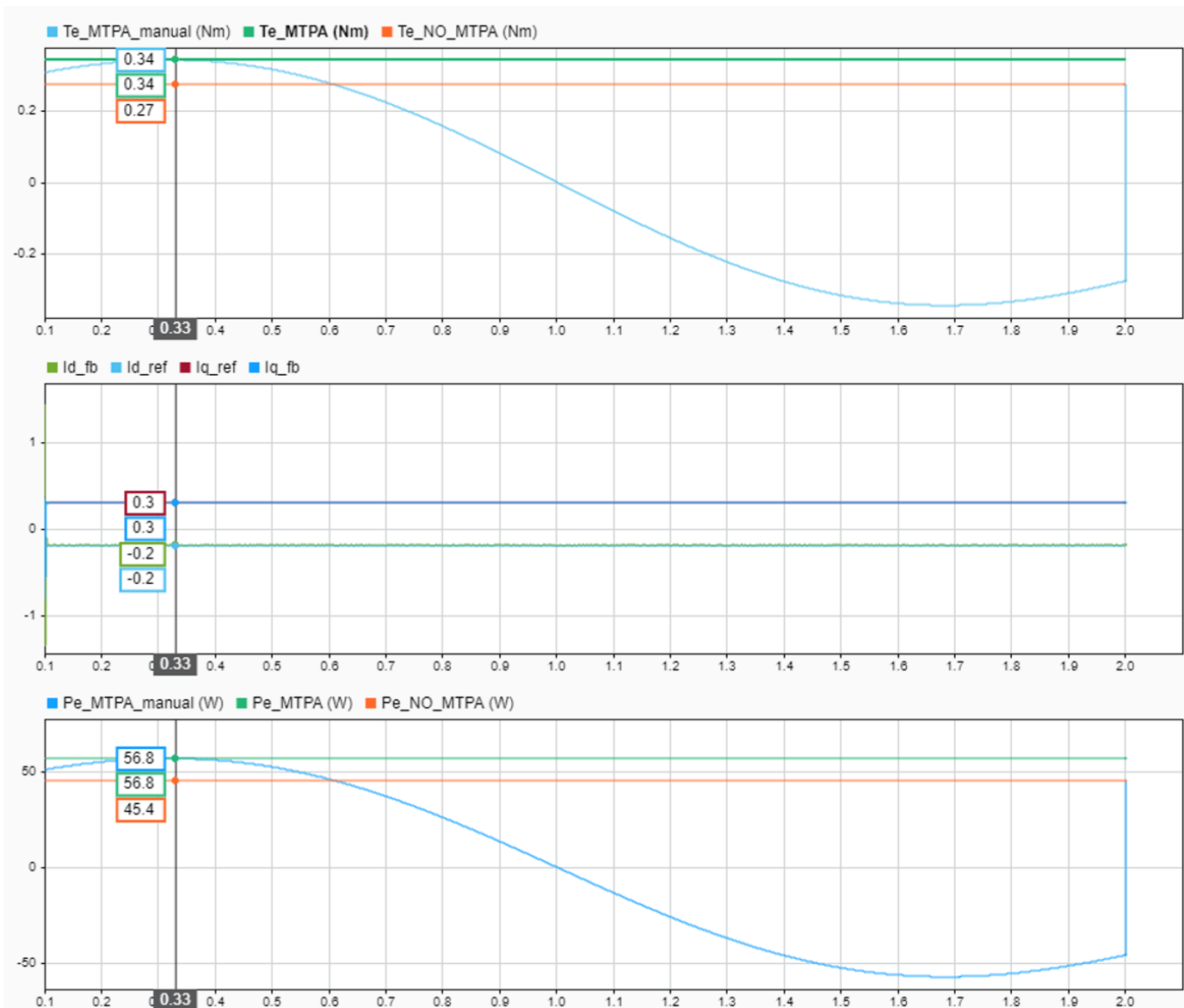
In the preceding example, you can notice that the electrical torque generated using MTPA is 0.34PU whereas electrical torque generated without MTPA is 0.27PU. You can also notice that with a varying torque angle, the maximum generated torque matches the torque produced by MTPA. The negative d-axis current 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 these locations:

- mcb_pmsm_fwc_qep_f28069LaunchPad/Speed Control
- mcb_pmsm_fwc_qep_f28379d/Speed Control
- mcb_pmsm_mtpa_qep_f28069LaunchPad/Torque Control/MTPA_Reference
- mcb_pmsm_mtpa_qep_f28379d/Torque Control/MTPA_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. You can also use the target model name 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 7-6.

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 4-10.

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 4-79.

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');
```
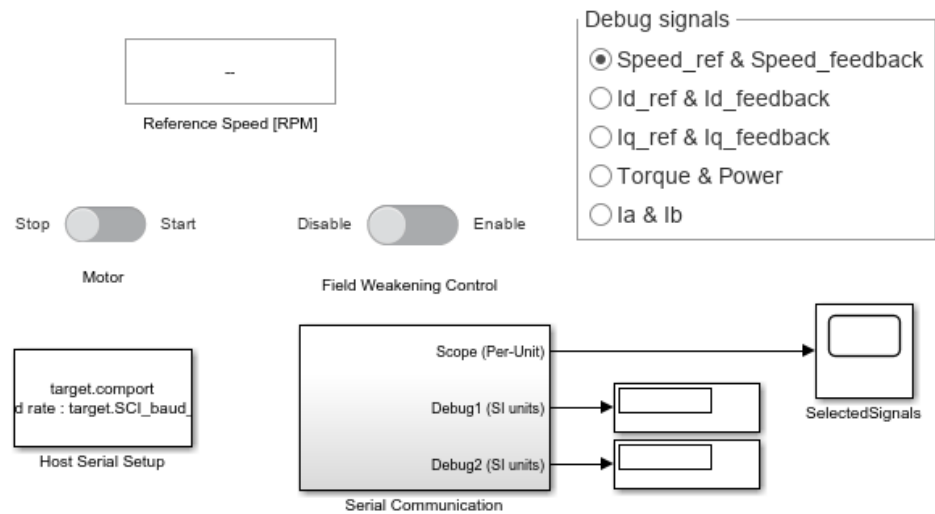
PMSM Field Weakening Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_fwc_qep_f28069LaunchPad
   mcb_pmsm_fwc_qep_f28379d

2. Verify that the variables from the target model are loaded to the base workspace.

Steps:
1. Set the serial port to **target.comport** in the init script of the target mode.
3. Use 'Motor' switch to control motor.
4. Input speed request using 'Reference Speed' edit box.
5. Observe the debug signals in scope.

Reference Speed [RPM]

Stop — Start   Disable — Enable

Motor   Field Weakening Control

Debug signals
- ◉ Speed_ref & Speed_feedback
- ○ Id_ref & Id_feedback
- ○ Iq_ref & Iq_feedback
- ○ Torque & Power
- ○ Ia & Ib

target.comport
d rate : target.SCI_baud_

Host Serial Setup

Scope (Per-Unit)

Debug1 (SI units)

Debug2 (SI units)

Serial Communication

SelectedSignals

Copyright 2020-2022 The MathWorks, Inc.

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

10. In the model initialization script associated with the target model, specify the communication port using the variable target.comport. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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."

See Also

- "Field-Weakening Control (with MTPA) of Non-Linear PMSM Using Look-Up Table" on page 4-295
- "PMSM Drive Characteristics and Constraint Curves"

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 4-3.

This example uses the sensorless position estimation technique. You can select either the sliding mode observer or flux observer or extended EMF 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.

The Extended EMF Observer block computes the electrical position, θ_e (or $\sin \theta_e$ and $\cos \theta_e$) in addition to the mechanical speed of PMSM from the measured voltage and current in the stationary alpha-beta reference frame.

To ensure that the detected rotor position is accurate, add the inverter board resistance value to the stator phase resistance parameter of the motor block and the stator resistance parameter of the Sliding Mode Observer, Flux Observer, and Extended EMF Observer blocks.

If you use flux observer, the example can run both PMSM and brushless DC (BLDC) motors.

The sensorless observers and algorithms have known limitations regarding motor operations beyond the base speed. We recommend that you use the sensorless examples for operations upto base speed only.
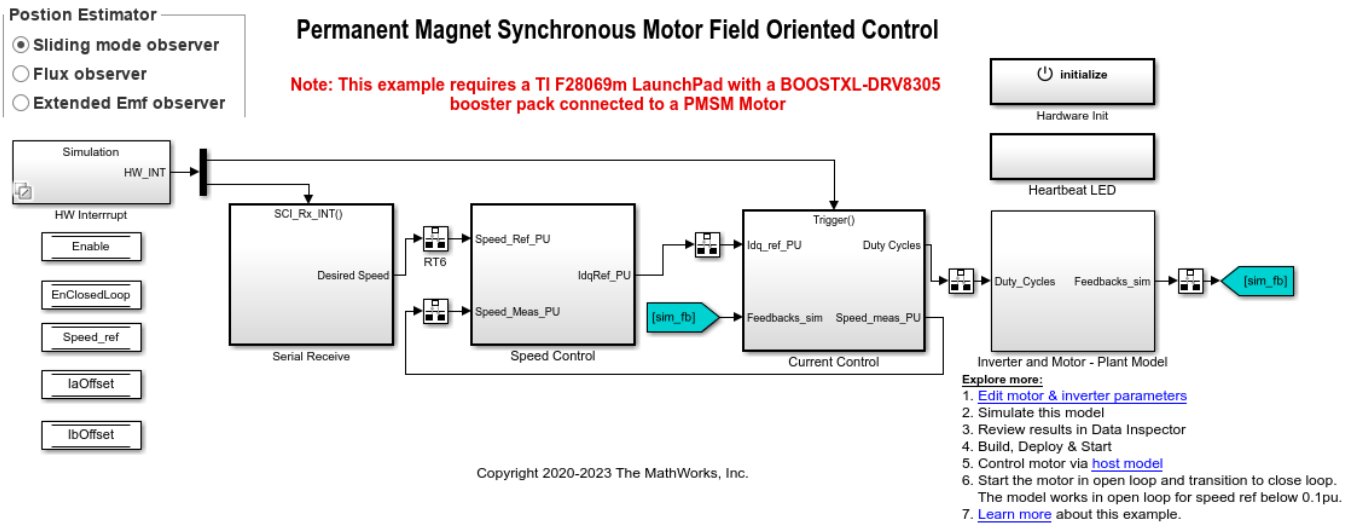
Models

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 also 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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_foc_sensorless_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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.

Sliding Mode Observer parameters require tuning if you are using Sliding Mode Observer with the motor parameters estimated using the parameter estimation tool.

Simulate Model

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

1. Open a target model included with this example.

2. Use the **Position Estimator** button to select one of the following sensorless position estimation techniques:

- **Sliding Mode Observer**

- **Flux Observer**

- **Extended EMF Observer**

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

4. 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. You can also use the target model name 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 7-6.

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 4-10.

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. Use the **Position Estimator** button to select a sensorless position estimation technique.

7. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target 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:
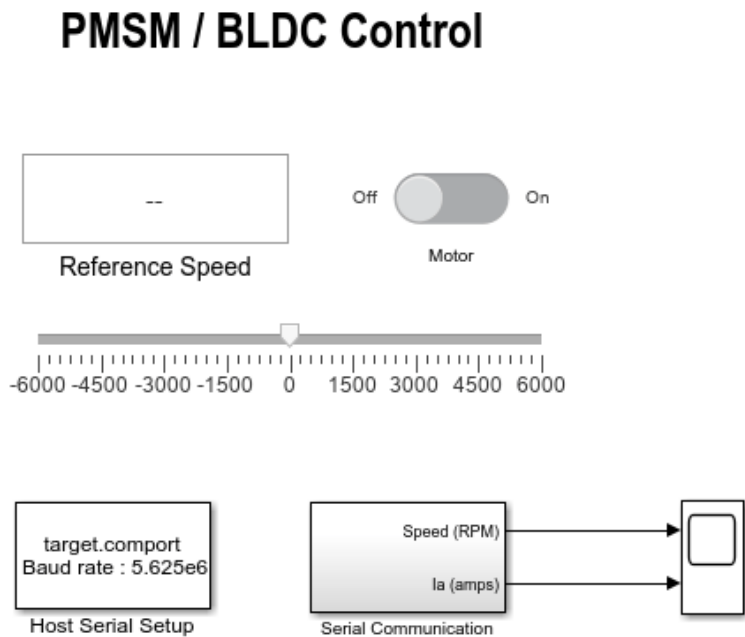
```
open_system('mcb_host_model_f28069m.slx');
```

Prerequisites:

1. Deploy the target model to the hardware
   mcb_pmsm_foc_qep_f28069m
   mcb_pmsm_foc_qep_f28069LaunchPad
   mcb_pmsm_foc_hall_f28069m
   mcb_pmsm_foc_sensorless_f28069MLaunchPad
   mcb_bldc_sixstep_f28069mLaunchPad

2. The variables from the target model are loaded to the base workspace.

Steps:

1. Set serial port to **target.comport** in the init script of the target model.
2. Use **Motor** switch to enable and disable motor control.
3. Input speed request using 'Reference Speed' edit box or sliding bar.
4. Observe the actual speed of motor and phase A current in the scope.
5. Start the motor in open loop under no load condition and transition to close loop for Sensorless Example. The model works in open loop for speed ref below 0.1pu.

PMSM / BLDC Control

Reference Speed

Motor: Off / On

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

target.comport
Baud rate : 5.625e6

Host Serial Setup

Speed (RPM)
Ia (amps)

Serial Communication

Copyright 2020-2022 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 model initialization script associated with the target model, specify the communication port using the variable target.comport. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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

NOTE:

• Before you run the motor at the required Reference Speed (by using either Sliding Mode Observer, Flux Observer, or Extended EMF Observer), start running the motor at 0.1 x `pmsm.N_base` speed by using open-loop control. Then transition to closed-loop control by increasing the speed to 0.25 x `pmsm.N_base` (where, `pmsm.N_base` is the MATLAB workspace variable for base speed of the motor).

• High acceleration and deceleration may affect the sensorless position computation.

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 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.

When you run this example on the hardware at a low Reference Speed, due to a known issue, the PMSM may not follow the low Reference Speed.

13. 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.

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

NOTE:

- A high reference speed and a high reference torque can affect the Sliding Mode Observer block performance.

- 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" on page 4-134.

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" on page 4-144.

Field-Oriented Control of PMSM 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 6-20), 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 4-3.

Models

The example includes the model mcb_pmsm_foc_qep_f28379d_SIUnit.

You can use this model for both simulation and code generation. You can also 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®
- C2000™ Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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

The example supports this hardware configuration. You can also use the target model name 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 7-6.

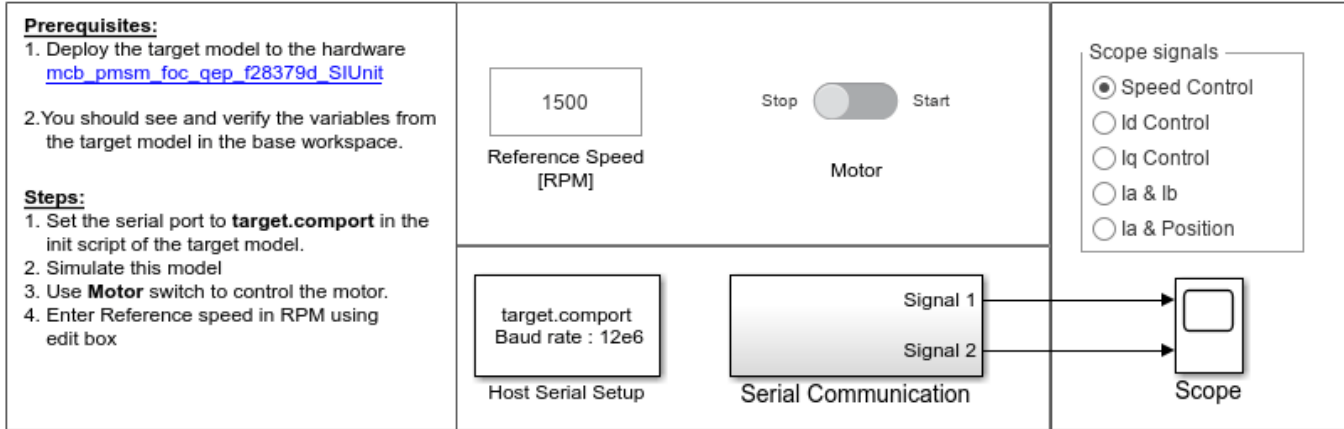**Generate Code and Run Model on Target Hardware**

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 4-10.

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 4-79.

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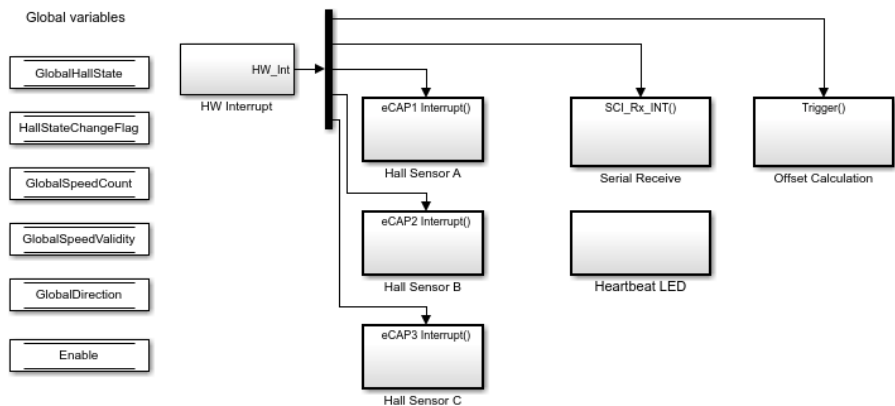
```
open_system('mcb_pmsm_SIUnit_host_model.slx');
```

FOC Host for SI Unit Example

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_foc_qep_f28379d_SIUnit

2. You should see and verify the variables from
   the target model in the base workspace.

Steps:
1. Set the serial port to **target.comport** in the
   init script of the target model.
2. Simulate this model
3. Use **Motor** switch to control the motor.
4. Enter Reference speed in RPM using
   edit box

1500

Reference Speed
[RPM]

Stop ⬤ Start

Motor

Scope signals
- ⦿ Speed Control
- ◯ Id Control
- ◯ Iq Control
- ◯ Ia & Ib
- ◯ Ia & Position

target.comport
Baud rate : 12e6

Host Serial Setup

Signal 1
Signal 2

Serial Communication

Scope

Copyright 2020-2022 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 also use the open_system command to open the Simulink® models. For example, use this command for a F28069M based controller:

```
open_system('mcb_pmsm_hall_offset_f28069m.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 generate code and deploy model:

1. For the model: **mcb_pmsm_hall_offset_f28069m**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_hall_offset_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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. You can also use the target model name 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 7-3.

- 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 7-6.

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 the 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');
```


PMSM Position Sensor (Hall / QEP) Offset Calibration Host

Copyright 2020-2021 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 Scope in the host model to monitor the rotor position and offset values.

7. In the host model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a **Port**.

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

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

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

- The **Calibration complete** LED turns green when the calibration process is complete. Then the `Calibration Output` field displays the computed offset value.

- The **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: To immediately stop the motor, click the **Emergency Motor Stop** button.

This example does not support simulation. The example automatically saves the computed offset value in the `PositionOffset` variable available in the base workspace.

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 and Use Utility Functions" 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 (secondary) windings placed around the resolver rotor (primary) winding. After you mount the resolver sensor over a PMSM, the resolver rotor winding rotates along with the shaft of the running motor. Meanwhile, the controller provides a fixed frequency excitation signal (either alternating sinusoidal or square pulse signal) to the primary 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.

Therefore, the resolver uses primary excitation input signal to generate the modulated secondary sine and cosine waveforms. It utilizes one winding to two winding transformations. The sine and cosine modulation occurs in the secondary windings because of the design and construction of these windings, which places them at positions that are 90 degrees apart.
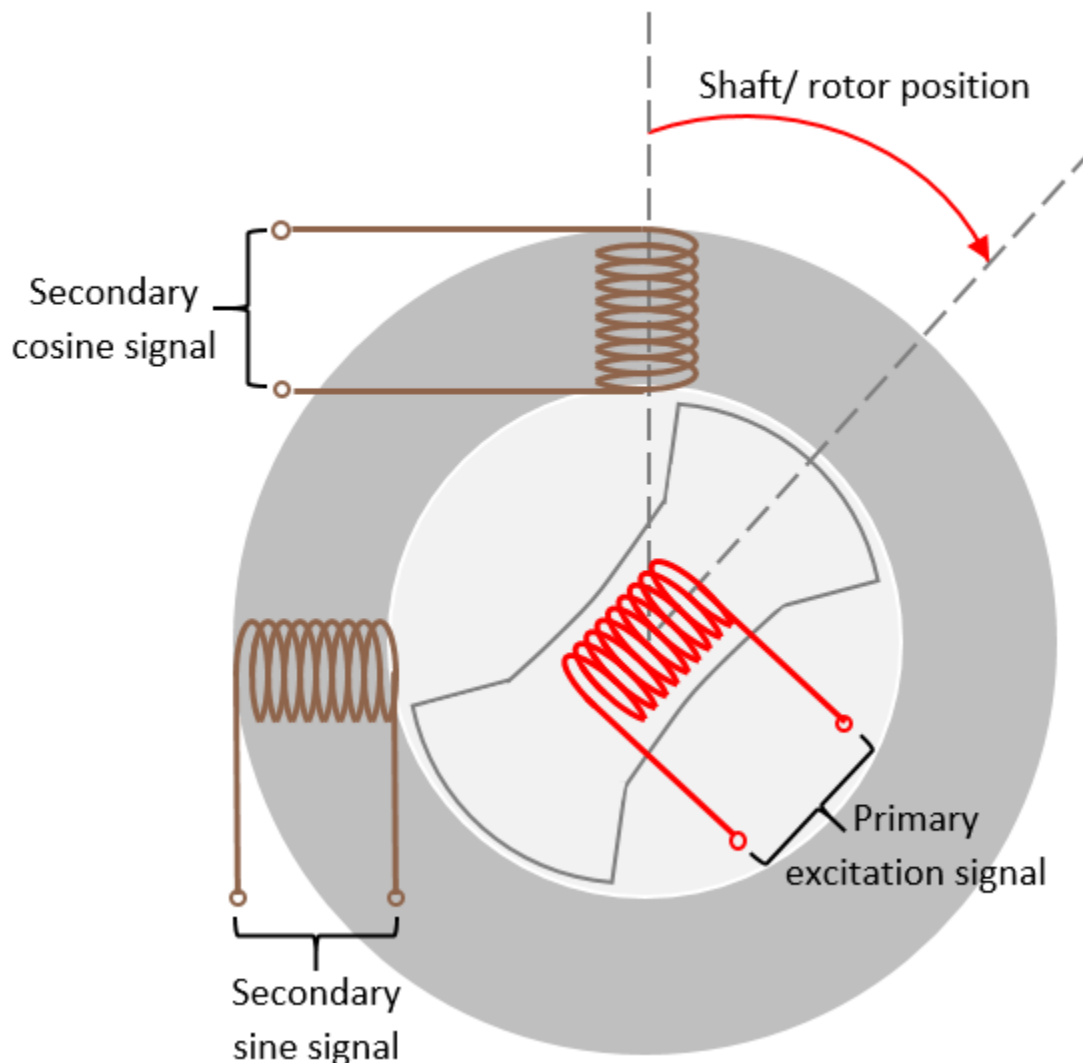
After receiving the secondary signals, the controller samples them using ADC and normalizes them.

Resolver excitation methods

The example uses the following two excitation methods:

- Sinusoidal excitation — When the primary excitation signal is sinusoidal. This generates sinusoidal secondary waveforms after modulation.
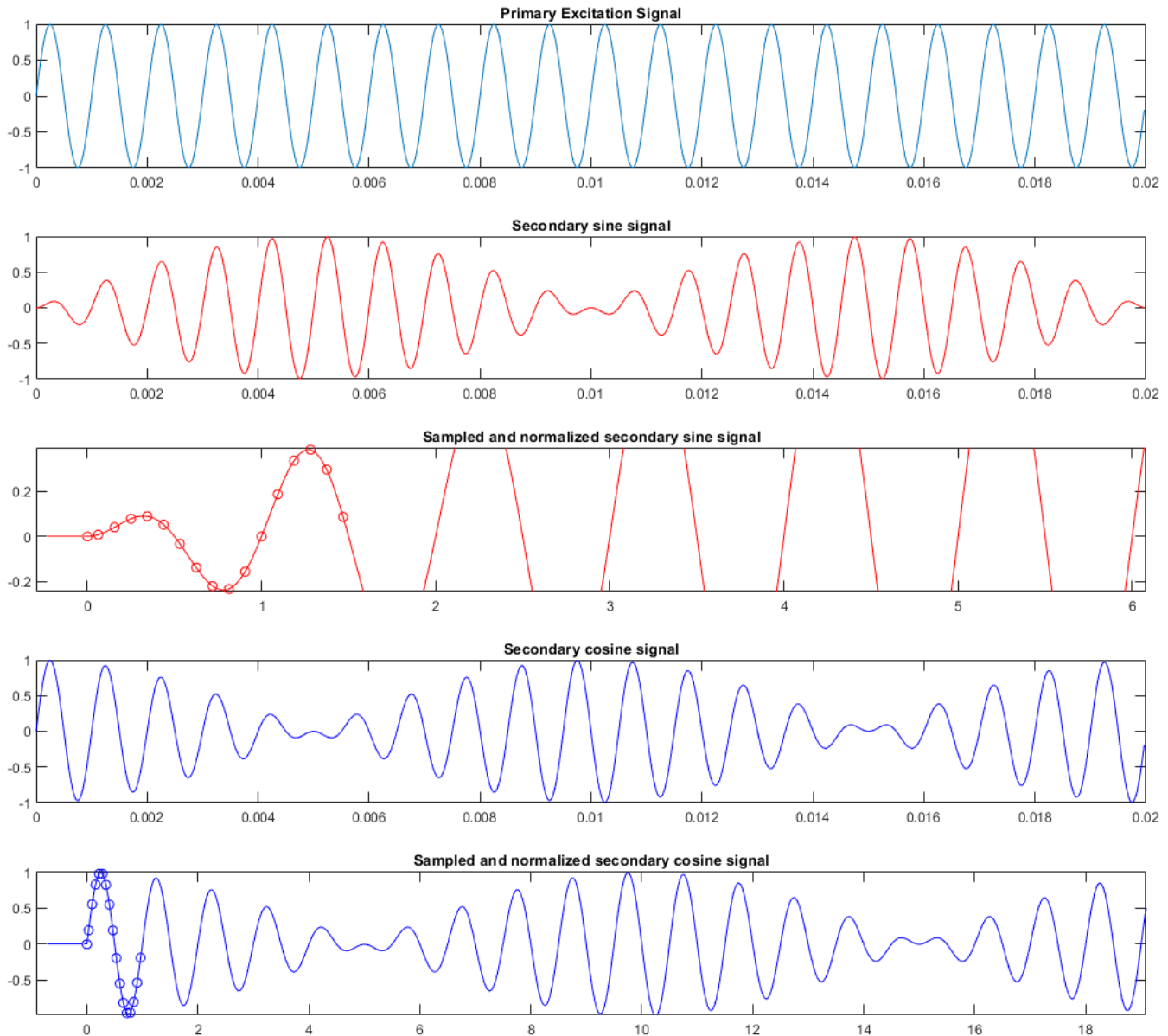
- Square pulse excitation — When the primary excitation signal is a square pulse signal. This generates square pulse secondary signals after modulation.
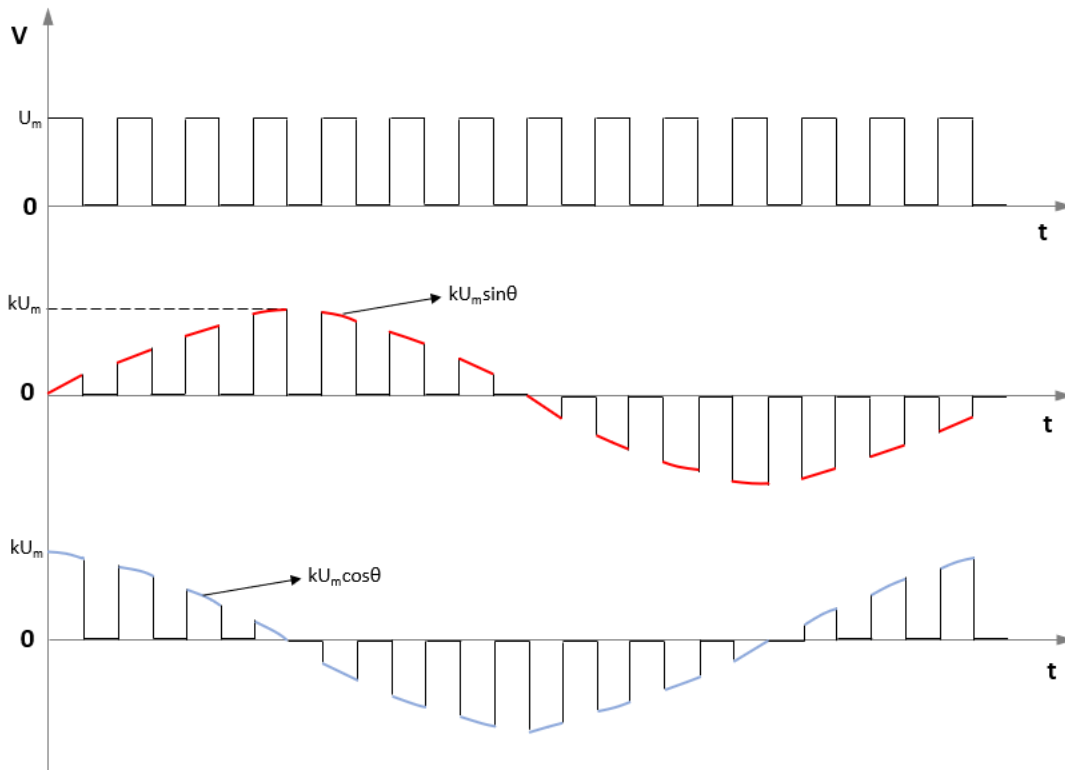
The example uses the Resolver Decoder block to compute the mechanical position of the resolver or motor. By default, this block uses a discrete step size which is identical to the sampling time used by the example. For more details about how the block functions with these excitation techniques, see Resolver Decoder.

Models

The example includes these models:

- mcb_resolver_f28069m — This model:

    - Supports sinusoidal excitation technique.

    - Supports LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter.

    - Supports only one pole pair resolver.

    - Computes mechanical position of resolver or motor.

- mcb_resolver_f28379d — This model:

    - Supports square pulse excitation technique.

    - Supports LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter.

    - Supports resolver with any number of pole pairs.

    - Computes mechanical position of resolver or motor.

You can use these models only for code generation.

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

```
open_system('mcb_resolver_f28069m.slx');
```
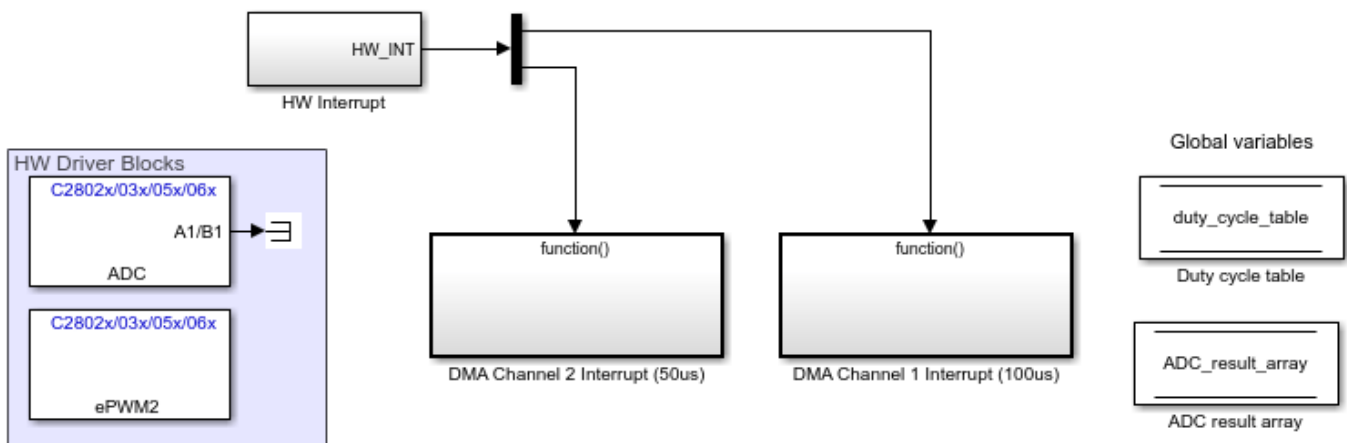

Required MathWorks® Products

For the models **mcb_resolver_f28069m** and **mcb_resolver_f28379d**:

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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 and Use Utility Functions" 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 mechanical shaft angle from the controller. You can observe the computed shaft position in the Time Scope block of the host model.

Required Hardware

The example supports these hardware configurations. You can also use the target model name to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

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

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_resolver_f28379d

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections and open one of these target models:

- **mcb_resolver_f28069m** — If you are using sinusoidal excitation technique.

- **mcb_resolver_f28379d** — If you are using square pulse excitation technique.

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:

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

Rotor position measurement using Resolver - Host

Prerequisites:
1. For square-pulse excitation method, deploy the mcb_resolver_f28379d model to the hardware.
2. For sinusoidal excitation method, deploy the mcb_resolver_f28069m model to the hardware.
3. Verify the variables from the target model in the base workspace.

Steps:
1. Select hardware in **Target Selelction.** Select 'Other' option if you want to manually set the baud rate in **'Host Serial Setup'** block.
2. Set the serial port to **target.comport** in the init script of the target model.
3. Observe the resolver position in scope
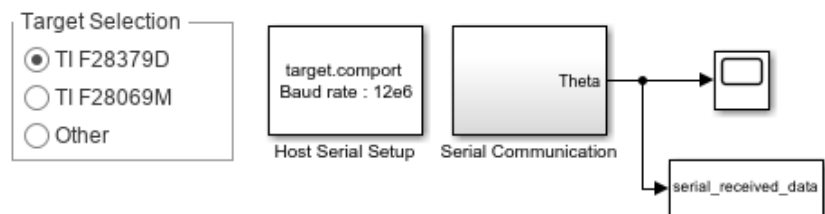
Target Selection
- ⦿ TI F28379D
- ◯ TI F28069M
- ◯ Other

target.comport
Baud rate : 12e6

Host Serial Setup

Theta

Serial Communication

serial_received_data

Copyright 2020-2022 The MathWorks, Inc.

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

5. In the host model, select the hardware that you are using:

• **TI F28379D** — If you are using the target model **mcb_resolver_f28379d**.

• **TI F28069M** — If you are using the target model **mcb_resolver_f28069m**.

6. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

Alternatively, in the masks of the following host model blocks, you can manually select a communication **Port**:

• mcb_resolver_host_read/Host Serial Setup

• mcb_resolver_host_read/Serial Communication/Host Serial Receive/Data_Type_Float/ Host Serial Receive

• mcb_resolver_host_read/Serial Communication/Host Serial Receive/Data_Type_Fixed_Point/ Host Serial Receive

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

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

9. Open the **Time Scope** block in the host model.

10. Rotate the resolver shaft and observe the computed shaft mechanical 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 also use the open_system command to open the Simulink® models. For example, use this command for a F28069M based controller:

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


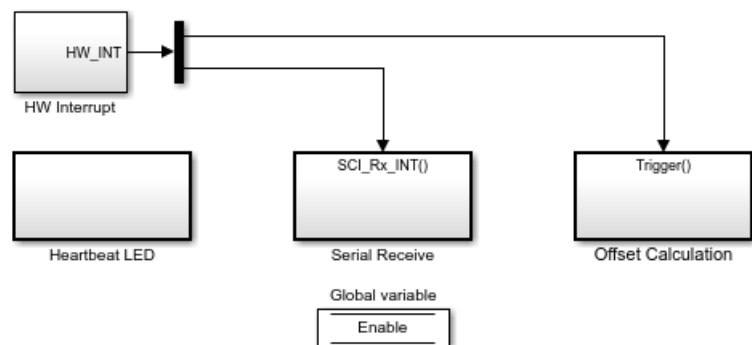For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To 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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™

2. For the model: **mcb_pmsm_qep_offset_f28379d**

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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. You can also use the target model name 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 7-3.

- 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.

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 7-6.

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 the 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**

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');
```

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 port in Host Serial Setup, Host Serial Receive and Host Serial Transmit
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]

Calibration Status

--

Note:
If motor is not rotating in correct direction, turn off the power supply to the target, interchange any two motor phase connections, and simulate the host model again. If motor is not rotating, check hardware setup.

Communication Port

No port selected

Host Serial Setup

Position_PU

Serial Communication

Emergency Motor Stop

Push for emergency stop

Scope

Copyright 2020-2021 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 Scope in the host model to monitor the rotor position and offset values.

7. In the host model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a **Port**.

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

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

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

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

- The **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: To immediately stop the motor, click the **Emergency Motor Stop** button.

This example does not support simulation. The example automatically saves the computed offset value in the `PositionOffset` variable available in the base workspace.

For examples that implement FOC using a quadrature encoder sensor, 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 and Use Utility Functions" 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 4-3.

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 the model mcb_ee_pmsm_foc.

You can use this model for both simulation and code generation. You can also 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');
```


Copyright 2020-2023 The MathWorks, Inc.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Simscape™ Electrical™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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 to simulate an inverter variant:

- **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. This option simulates the Simscape Electrical modular multilevel converter using a low voltage.

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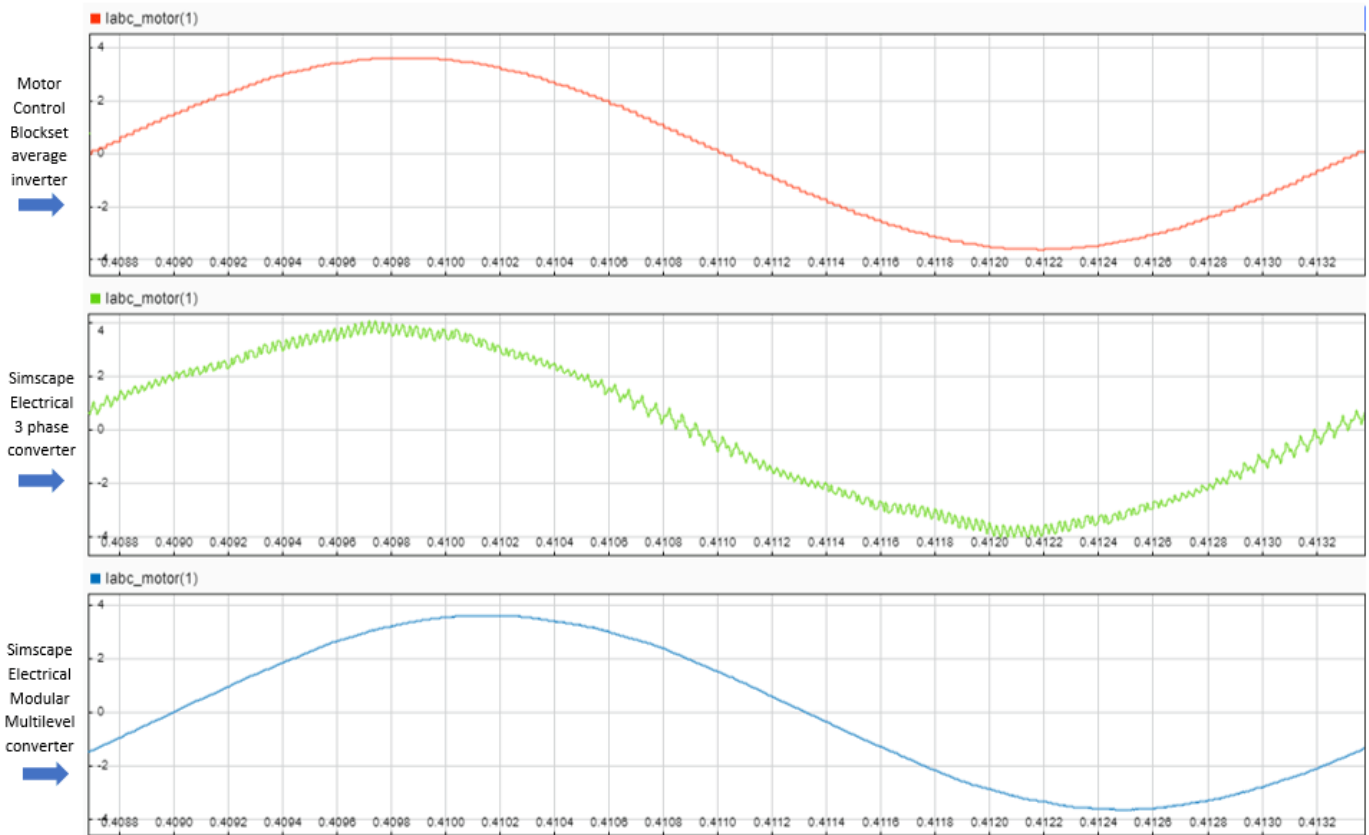
These images show the comparison of rotor speed, I_q current, I_{ab} phase current, and rotor position for the three inverter types:
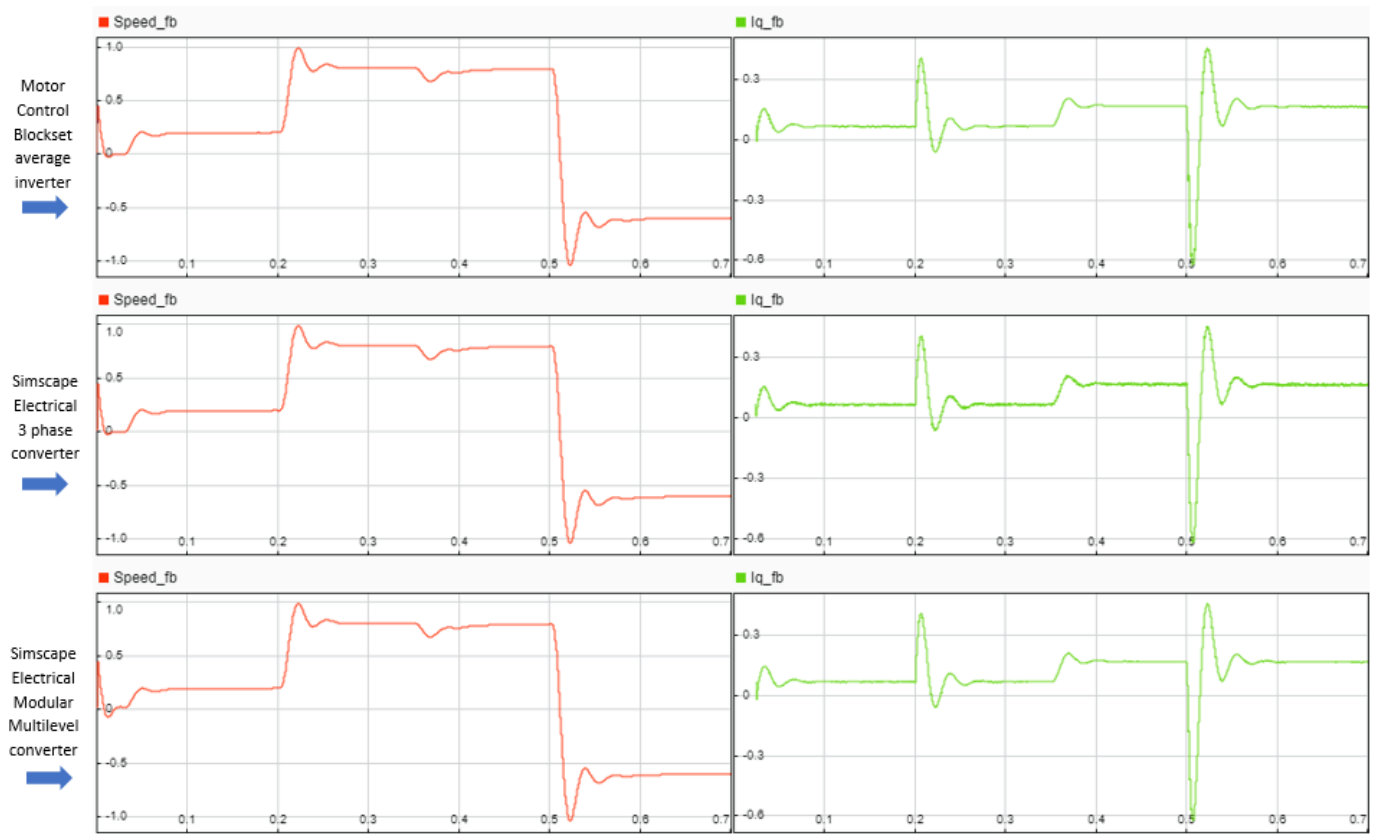
Motor Control Blockset average inverter

Simscape Electrical 3 phase converter

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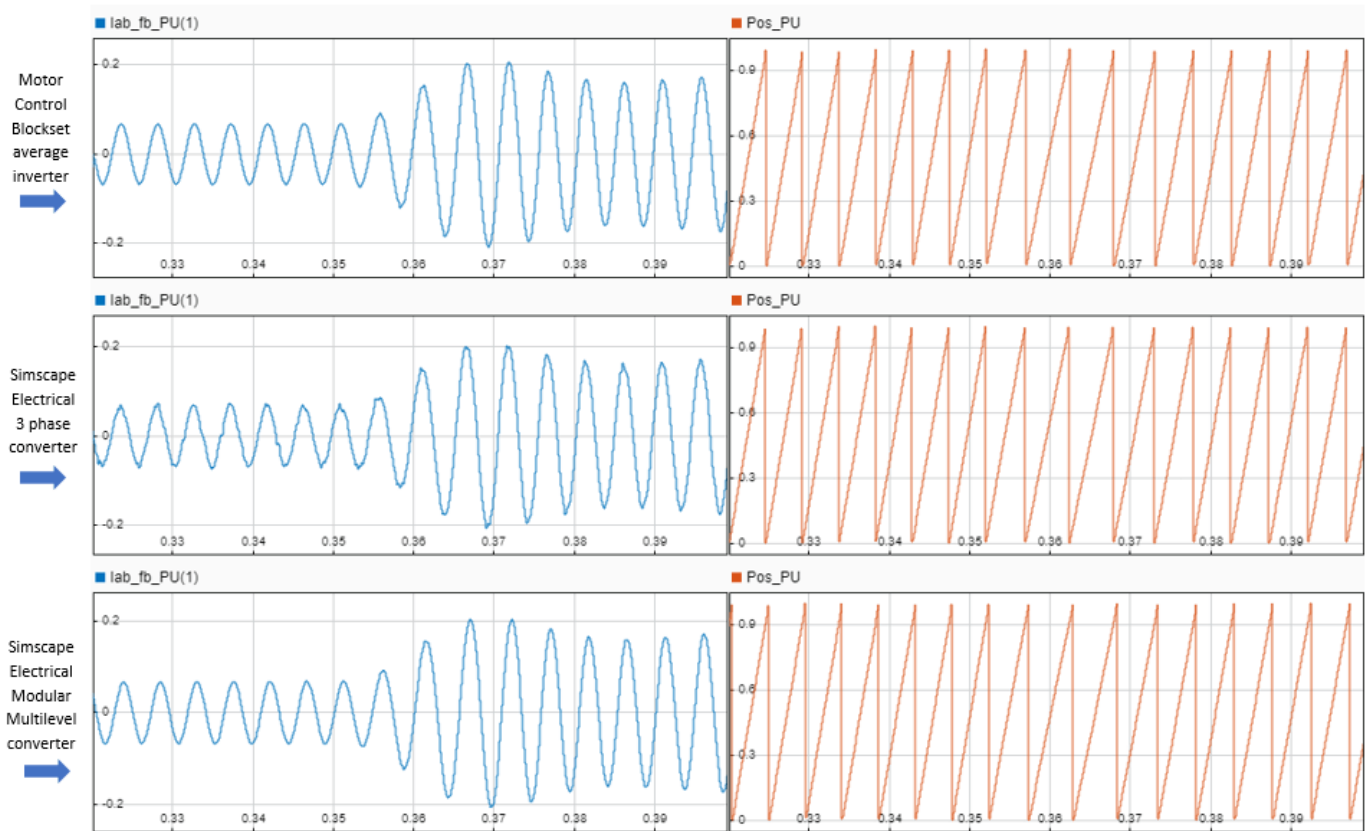
These images show the comparison of PWM modulation waveforms for the three inverter types:

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

The example supports this hardware configuration. You can also use the target model name 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 7-6.

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 4-10.

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 4-79.

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

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_foc_qep_f28379d
   mcb_pmsm_foc_hall_f28379d
   mcb_pmsm_foc_sensorless_f28379d
   mcb_pmsm_foc_mpc_qep_f28379d

2. The variables from the target model are loaded to the base workspace.
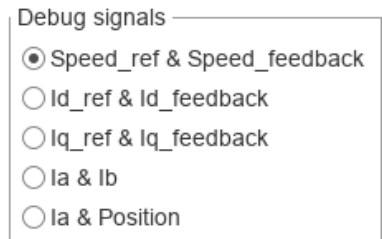
Debug signals
- ⦿ Speed_ref & Speed_feedback
- ◯ Id_ref & Id_feedback
- ◯ Iq_ref & Iq_feedback
- ◯ Ia & Ib
- ◯ Ia & Position

Stop ◉ Start
Motor

Reference Speed (RPM)
--

Steps:
1. Set the serial port to **target.comport** in the init script of the target model.
2. Use **Motor** switch to control motor.
3. Input speed request using 'Reference Speed' edit box.
4. Observe the debug signals in scope.
5. Start the motor in open loop under no load condition and transition to close loop for Sensorless Example. The model works in open loop for speed ref below 0.1pu.

target.comport
Baud rate : 12e6

Host Serial Setup

Scope (Per-Unit)
Debug1 (SI units)
Debug2 (SI units)

SelectedSignals

Copyright 2020-2022 The MathWorks,

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

9. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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" on page 4-134.

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. Motor 1 runs in the closed-loop speed control mode. Motor 2 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 Motor 1 and a different torque reference for Motor 2 (derived from the magnitude and electrical position of the Motor 2 reference stator current). Motor 1 runs at the reference speed for the load conditions provided by Motor 2 (with a different torque reference).

These equations describe the computation of d-axis and q-axis components of the Motor 2 reference stator current.

$$Id^{ref} = Imag^{ref} \times \cos\theta_e$$

$$Iq^{ref} = Imag^{ref} \times \sin\theta_e$$

where:

- Id^{ref} is the d-axis component of the Motor 2 reference stator current.

- Iq^{ref} is the q-axis component of the Motor 2 reference stator current.

- $Imag^{ref}$ is the magnitude of the Motor 2 reference stator current.

- θ_e is the electrical position of the Motor 2 reference stator current.

The example runs in the controller hardware board. You can input the speed reference for Motor 1 and current reference for Motor 2 using a host model. The host model uses serial communication to communicate with the controller hardware board.

Current control loops in Motor 1 and Motor 2 control algorithms are offset by Ts/2, where Ts is the control-loop execution rate.

Models

The example includes the model mcb_pmsm_foc_f28379d_dyno.

You can use this model for both simulation and code generation. You can also 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');
```
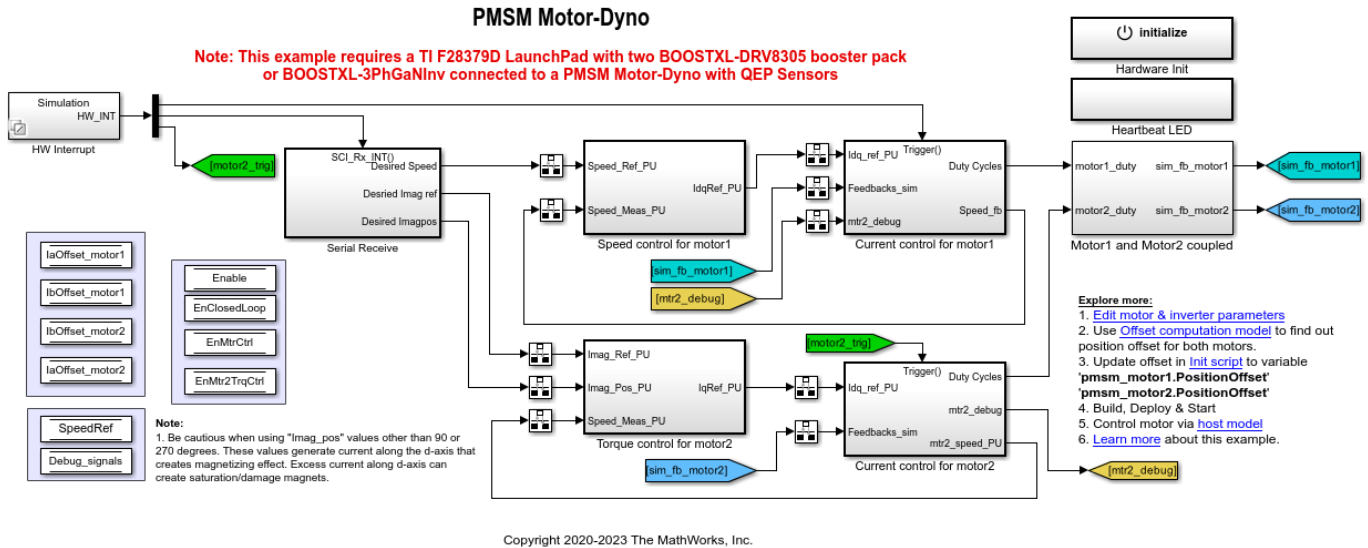
Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

To generate code and deploy model:

- Motor Control Blockset™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters for both Motor 1 and Motor 2. 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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 Motor 1 and a different current reference (load) for Motor 2. 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. You can also use the target model name 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

- LAUNCHXL-F28379D controller + 2 BOOSTXL-3PHGANINV inverters: mcb_pmsm_foc_f28379d_dyno

For connections related to the preceding hardware configuration, see "Instructions for Dyno (Dual Motor) Setup" on page 7-10.

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 4-10.

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 4-79.

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');

9. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available 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 1** switch to On, to start running the motor.

12. Update the **Motor 1 - Reference Speed (RPM)**, **Motor 2 - Imag Ref (A)**, and **Motor 2 - Imag Pos (deg)** in the host model.

Note: Be cautious when using values other than 90 or 270 degrees in the **Motor 2 - Imag Pos (deg)** field. These values generate current along the d-axis that creates a magnetizing effect. Excess current along the d-axis can create saturation and can damage the motor magnets.

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" on page 4-144.

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 4-3.

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.
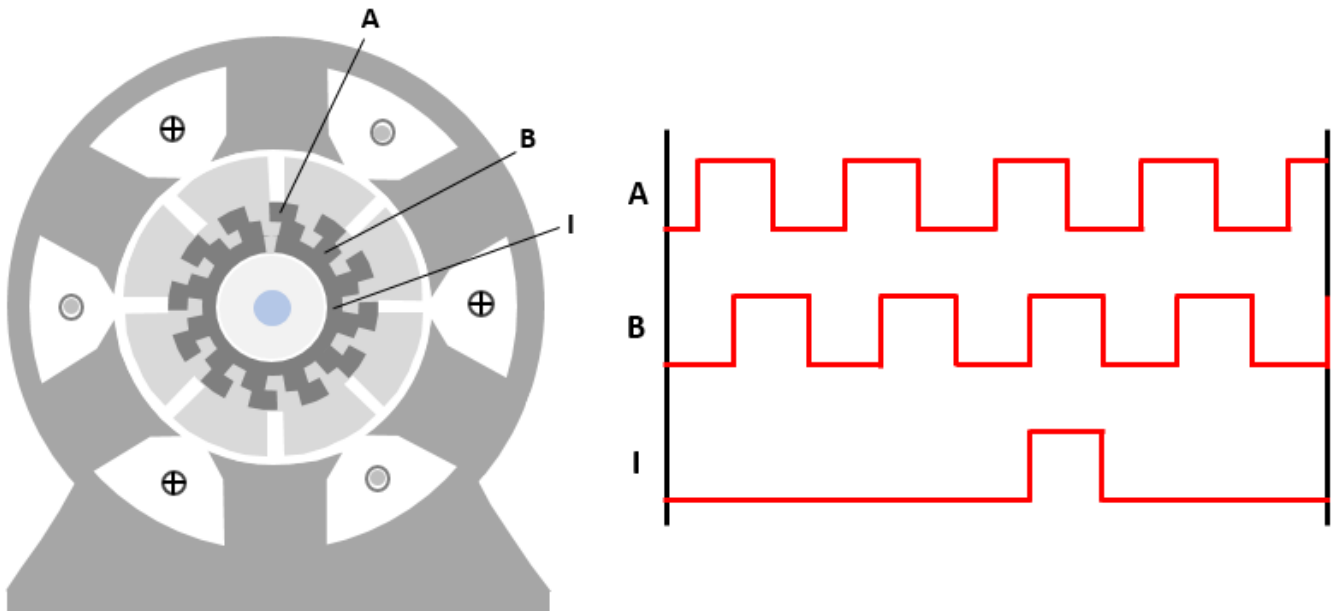
Model

The example includes the model mcb_acim_foc_qep_f28379d.

You can use this model for simulation and code generation. You can also 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 2020-2023 The MathWorks, Inc.

Note:
1) To achieve higher speeds, increase the "Max current" value in "Speed Control \ ACIM Control Reference" block (e.g. set to 2xIrated).
2) It is recommended to monitor motor's temperature for operation above base speed, while working with hardware.

Explore more:
1. Edit motor & inverter parameters
2. Simulate this model
3. Review results in Data Inspector
4. Generate code from hardware tab with "Build, Deploy & Start"
5. Control motor via host model
6. Learn more about this example.

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®
- C2000™ Microcontroller Blockset
- 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 and Use Utility Functions" on page 3-2.

3. The initialization script also computes the derived parameters. For example, total leakage factor, rated flux, rated torque, stator and rotor inductances of the induction motor.

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 the following hardware configuration. You can also use the target model name 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 7-6.

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 4-10.

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');
```

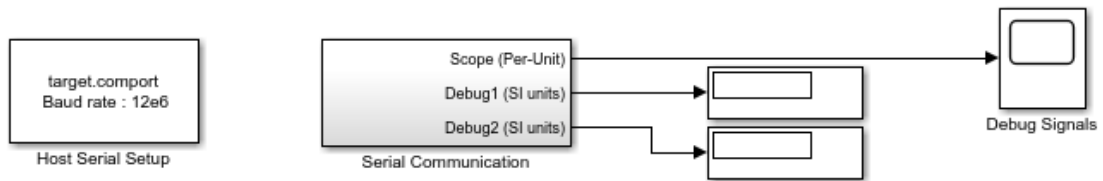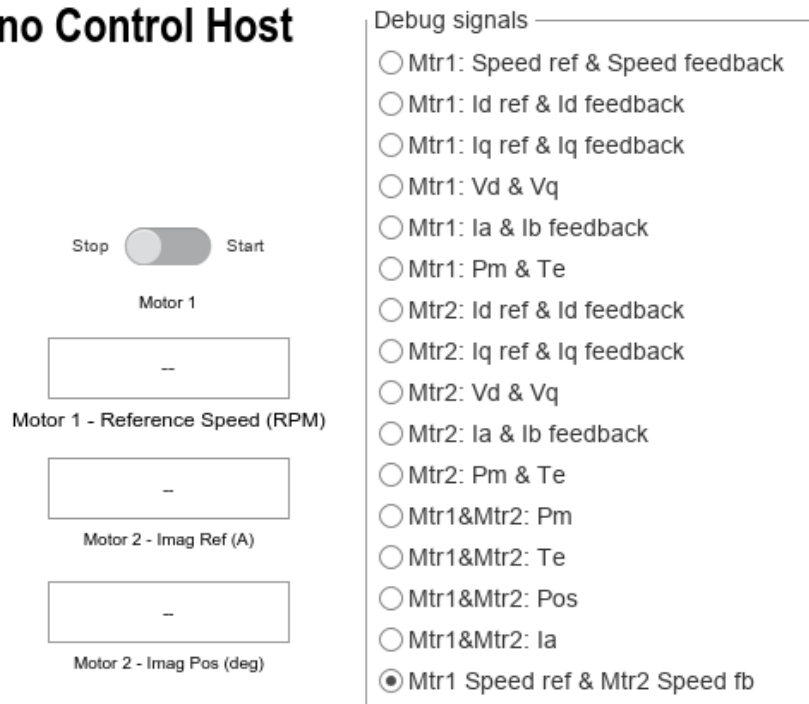## AC Induction Motor
Field Oriented Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_acim_foc_qep_f28379d
   mcb_acim_foc_sensorless_f28379d

2. Verify the variables from the target model in the base workspace.

Steps:
1. Set DC power supply to **inverter.V_dc (volts)**
2. Set the serial port to **target.comport** in the init script of the target model.
3. Simulate this model
4. Use **Start / Stop Motor** switch to control the motor.
5. Enter Reference Speed in RPM using the edit box
6. Observe the selected Debug signal in the Scope

300
Reference Speed (RPM)

Stop — Start   Stop — Start
Motor   Field Weakening Control

Debug signals
- Speed_ref & Speed_feedback
- Id_ref & Id_feedback
- Iq_ref & Iq_feedback
- Torque & Power
- Ia & Ib

target.comport
d rate : target.SCI_baud_
Host Serial Setup

Scope (Per-Unit)
Debug1 (SI units)
Debug2 (SI units)
Serial Communication

Scope

Copyright 2020-2022 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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.

NOTE: This example depends on the positive speed feedback for the positive rotation of the space vectors. If the motor does not run, try these steps to resolve the issue:

- Try interchanging any two motor phase connections.

- Modify and use the example "Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-10 with a speed feedback and confirm the positive direction of rotation for a positive reference speed.

See Also

- Field-Oriented Control of Induction Motors with Simulink and Motor Control Blockset

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 4-3.

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.

To ensure that the detected position is accurate, add the inverter board resistance value to the stator phase resistance parameter of the motor block and the stator resistance parameter of the Flux Observer block.

The sensorless observers and algorithms have known limitations regarding motor operations beyond the base speed. We recommend that you use the sensorless examples for operations upto base speed only.

NOTE: The speed estimated by the Flux Observer block has an error against the actual rotor speed. This error is within tolerance of one percent of the base speed. You can introduce an offset compensation to the position output of the Flux Observer block to minimize this error.

Model

The example includes the model mcb_acim_foc_sensorless_f28379d.

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

```
open_system('mcb_acim_foc_sensorless_f28379d.slx');
```


Field-Oriented Control of AC Induction Motor

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®
- C2000™ Microcontroller Blockset
- 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 and Use Utility Functions" on page 3-2.

3. The initialization script also computes the derived parameters. For example, total leakage factor, rated flux, rated torque, stator and rotor inductances of the induction motor.

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

The example supports this hardware configuration. You can also use the target model name 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 7-6.

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 4-10.

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');
```


AC Induction Motor Field Oriented Control Host

Copyright 2020-2022 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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.

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 4-3.

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');
```
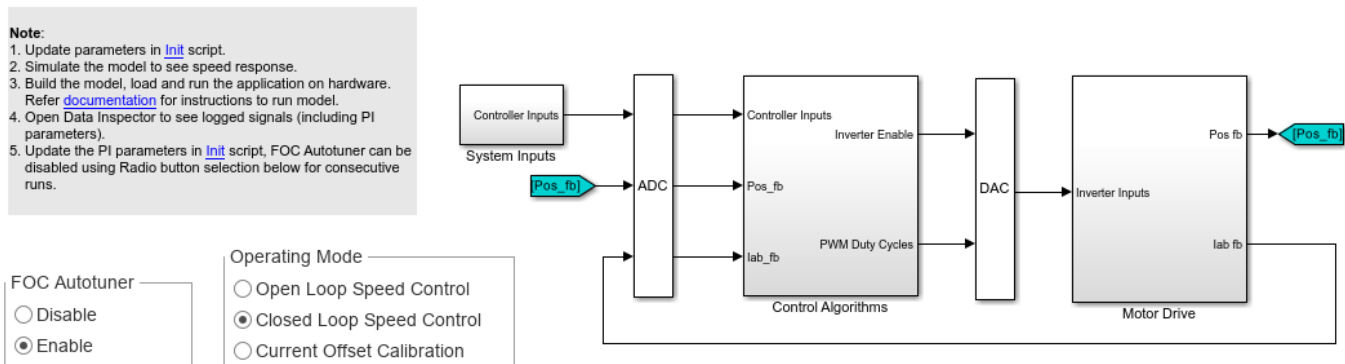

For details on the supported hardware configuration, see the Required Hardware section under Generate Code and Deploy Model to Target Hardware.

Required MathWorks® Products

- Motor Control Blockset™
- Simulink Control Design™

- Simulink Real-Time™

Additional Product

- Speedgoat I/O Blockset (from Speedgoat)

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 Using Field Oriented Control Autotuner" on page 4-28.

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 rated for 48 V and 20 A from Speedgoat

- 100 W three-phase brushless DC motor from Maxon Motor

- Quadrature encoder with 4096 impulses

- 150 W 254 V DC power supply

NOTE: Contact Speedgoat for the bit stream file that is valid for your hardware.

For details about Speedgoat 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.
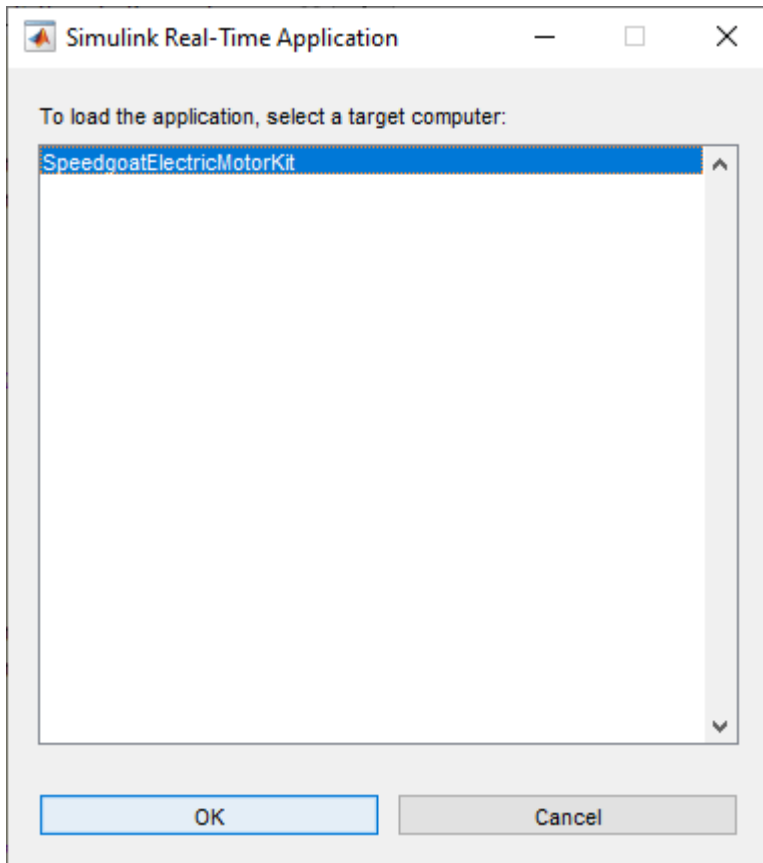
4. In the Simulink Real-Time Application dialog box, select the target computer to which you are connected. Click **OK** to load the application file to the hardware.

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.
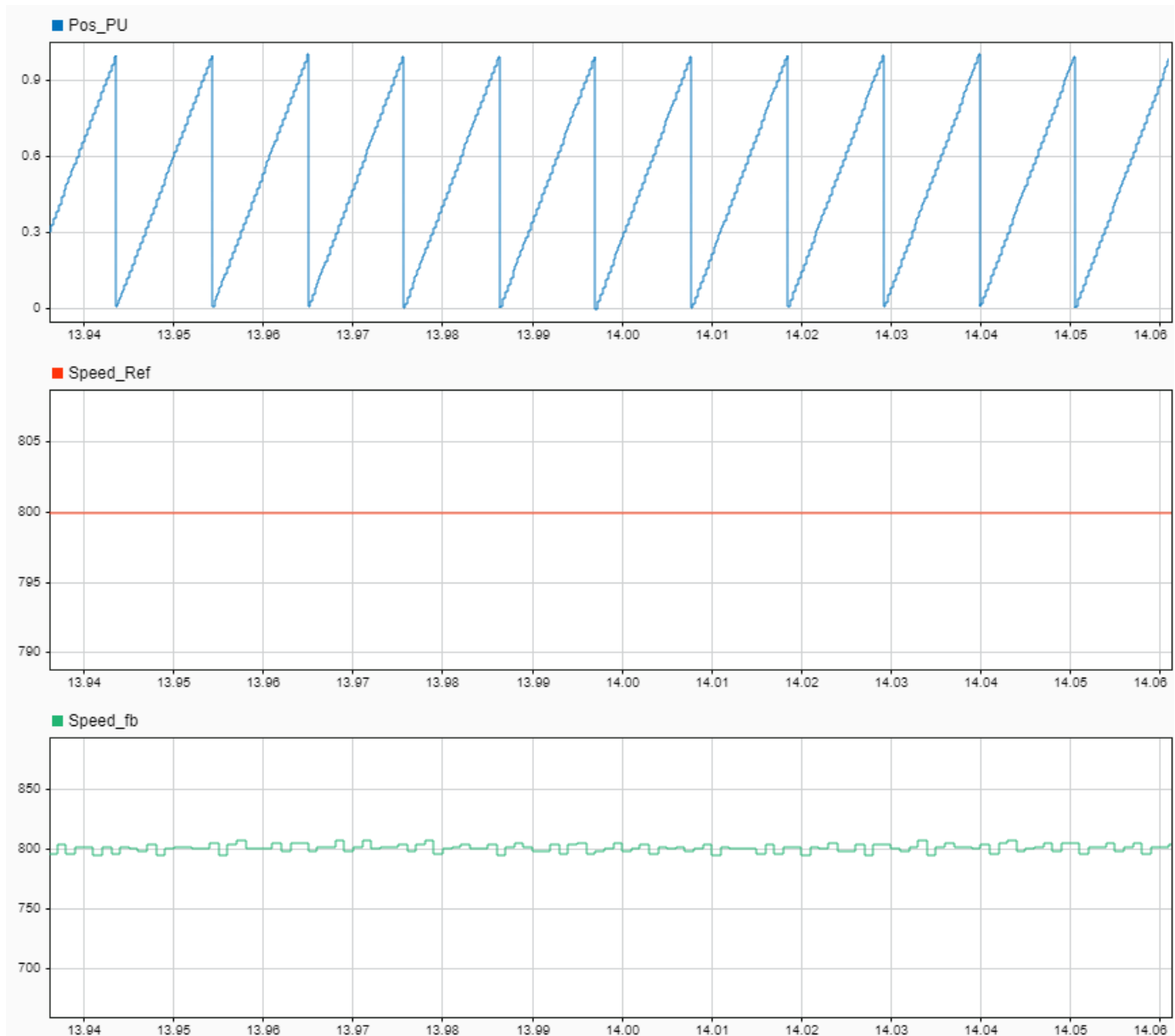
For example, verify that the positive reference speed has a positive speed feedback, and the position signal (`Pos_PU`) has a positive ramp.

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.
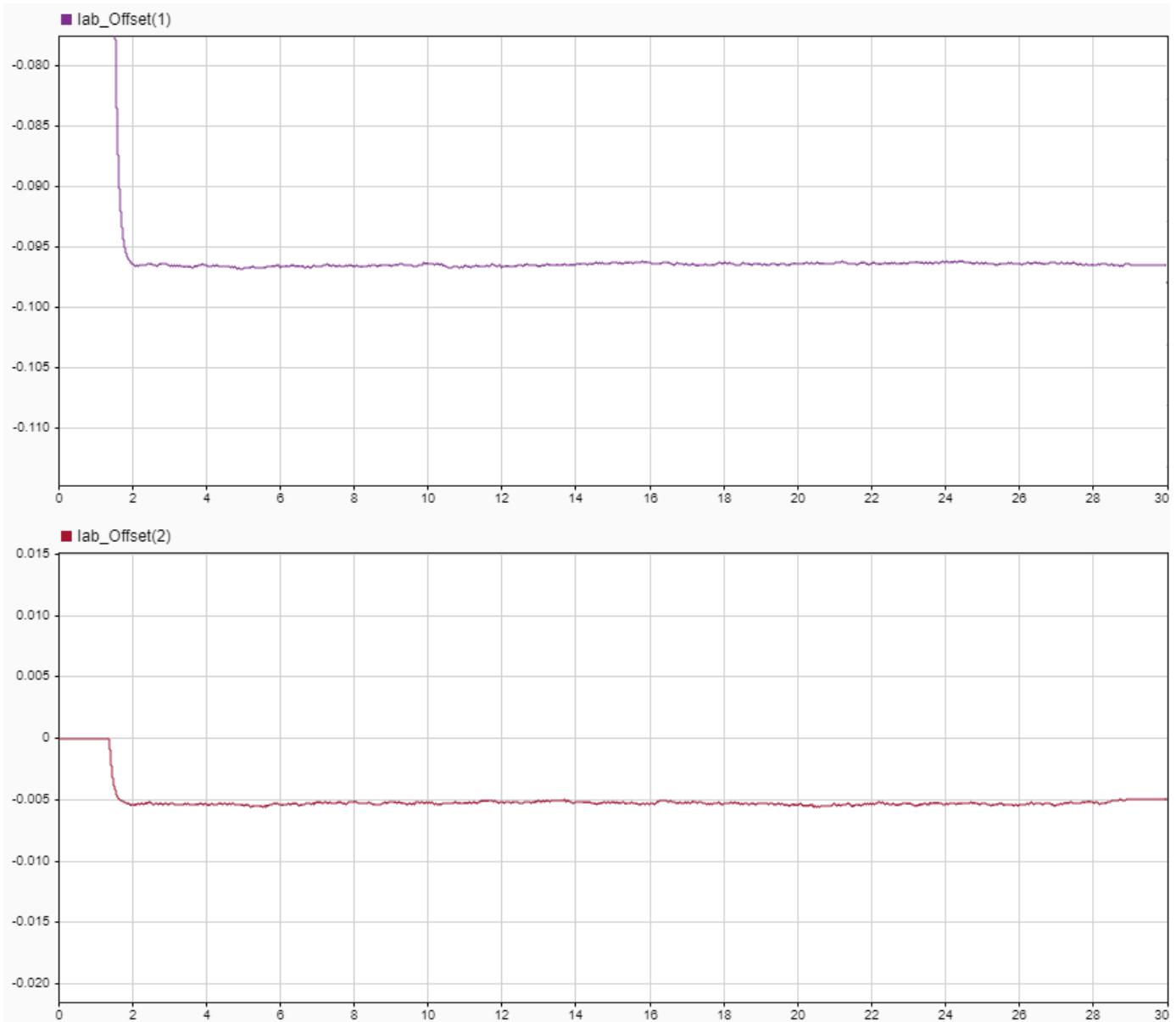
5. In the Simulink Real-Time Application dialog box, select the target computer to which you are connected. Click **OK** to load the application file to the hardware.

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.
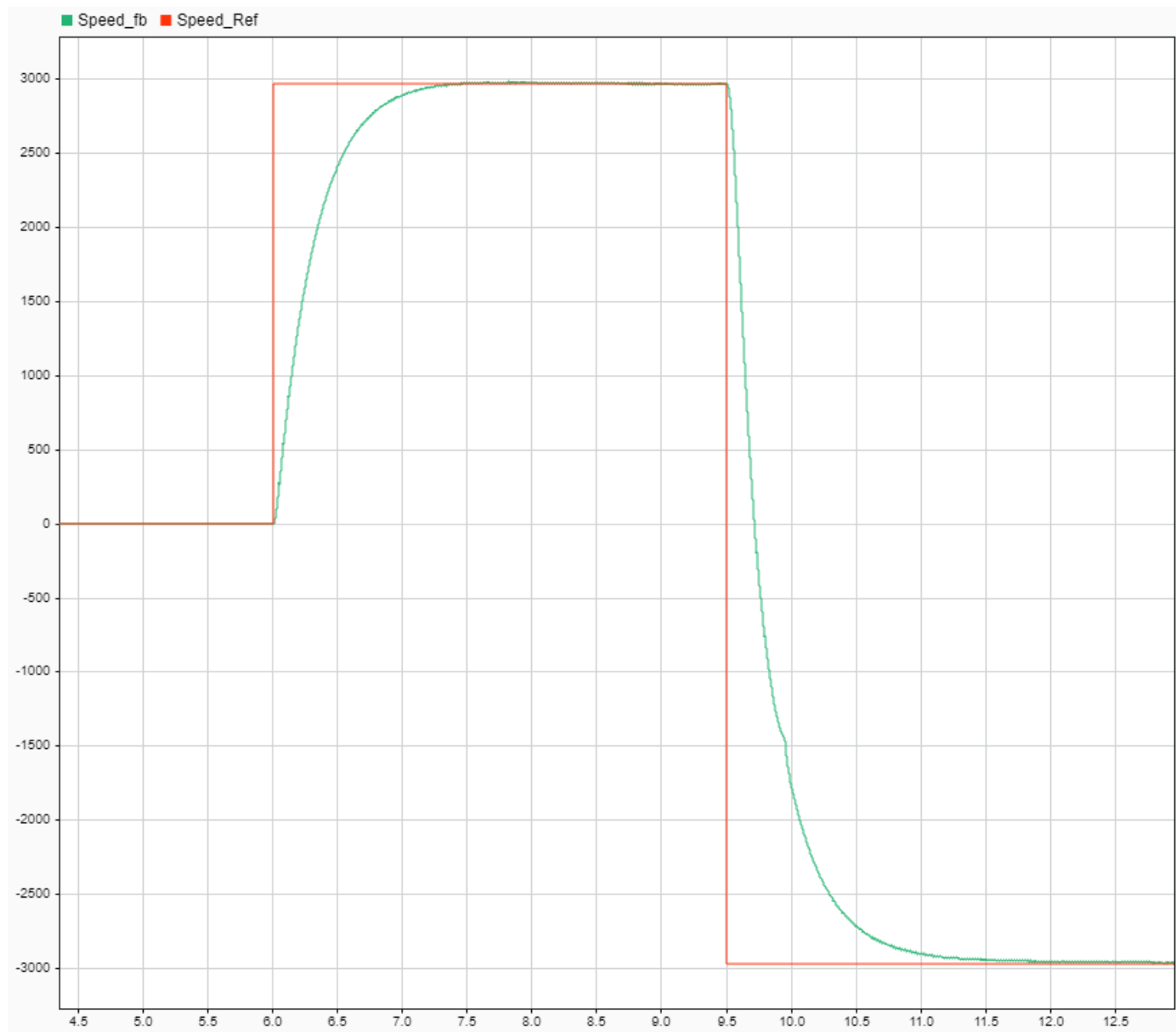
7. Verify that the motor reaches steady state operation because the FOC Autotuner will not work if the motor speed is unstable.

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.
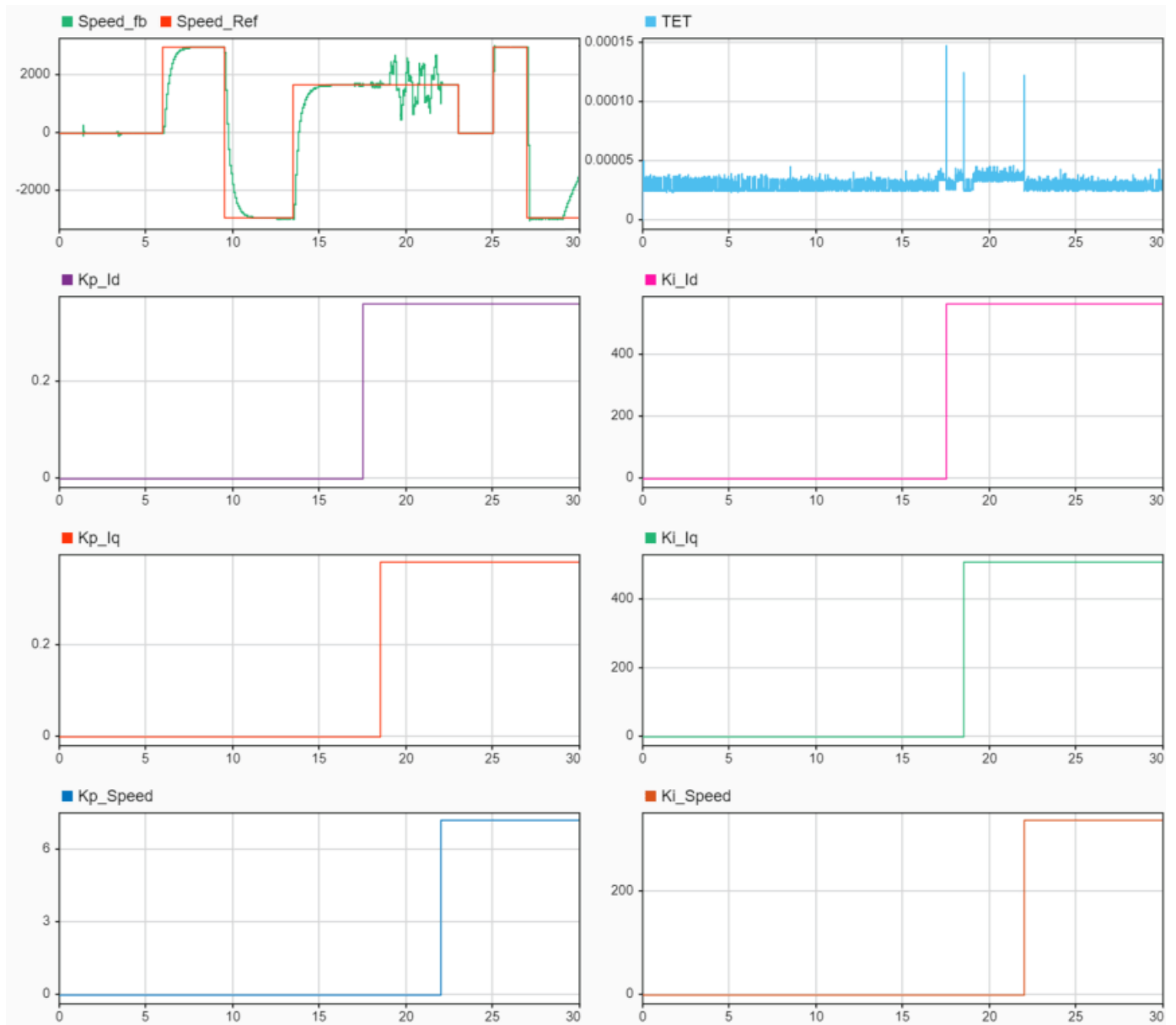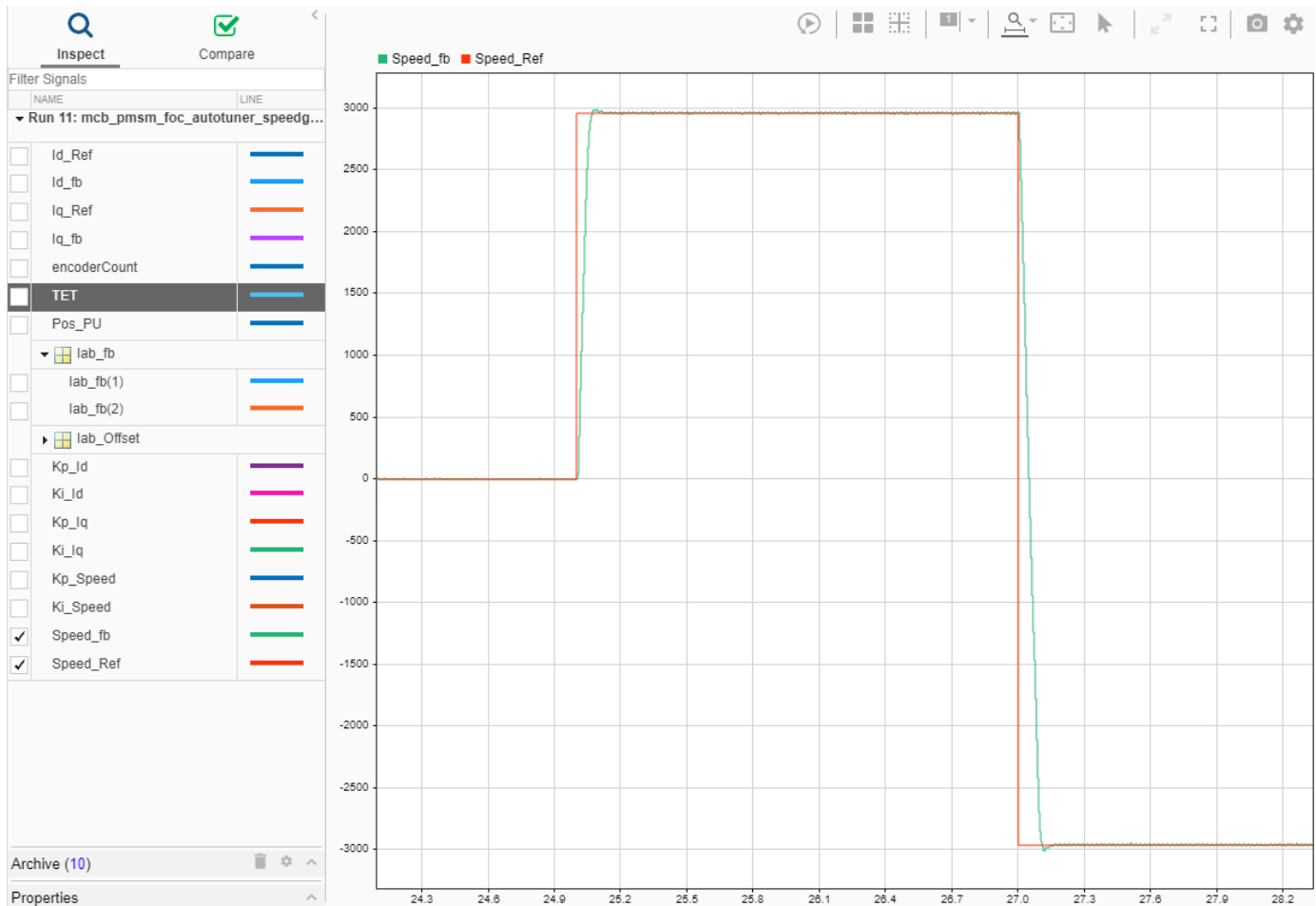
9. If the system response after tuning is satisfactory, update the gain values in the model initialization script associated with the model. For consecutive model executions, you can disable the FOC tuning using the FOC Autotuner button in the model and continue with the closed-loop testing using the 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.
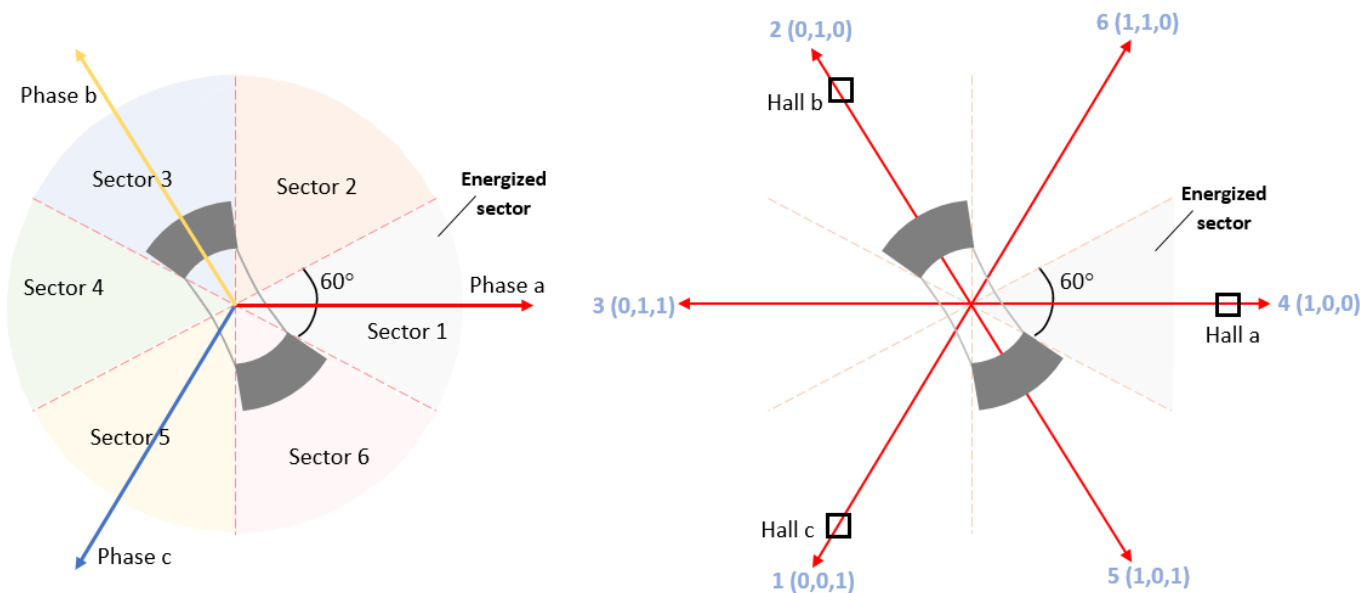
Note: The Hall sequence can vary. Use the example "Hall Sensor Sequence Calibration of BLDC Motor" on page 4-123 to determine the Hall sequence.

The example supports simulation. You can simulate a plant model containing a three-phase switching inverter along with wye-wound BLDC from Simscape™ Electrical™.

Models

The example includes these models:

- mcb_bldc_sixstep_f28069mLaunchPad

- mcb_bldc_sixstep_f28379d

You can use these models for both simulation and code generation. To open a Simulink® model, you can also use the open_system command at the MATLAB command prompt. For example, use this command for a F28379D based controller:

```
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:

1. For the model **mcb_bldc_sixstep_f28069mLaunchPad**

- Motor Control Blockset™
- Fixed-Point Designer™
- Simscape™ Electrical™

2. For the model **mcb_bldc_sixstep_f28379d**

- Motor Control Blockset
- Simscape Electrical

To generate code and deploy model:

1. For the model **mcb_bldc_sixstep_f28069mLaunchPad**

- Motor Control Blockset
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer

2. For the model **mcb_bldc_sixstep_f28379d**

- Motor Control Blockset
- Embedded Coder
- C2000 Microcontroller Blockset
- 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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 six-step commutation 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

The example supports these hardware configurations. You can also use the target model name to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: mcb_bldc_sixstep_f28069mLaunchPad

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_bldc_sixstep_f28379d

For connections related to these hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

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 4-10.

4. If you are using a quadrature encoder, compute the quadrature encoder index offset value and update it in the `bldc.PositionOffset` variable available in the model initialization script associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79.

5. If you are using a Hall sensor, compute the Hall sequence value and update it in the `bldc.hallsequence` variable available in the model initialization script associated with the target model. For instructions, see "Hall Sensor Sequence Calibration of BLDC Motor" on page 4-123.

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 **Quadrature Encoder** or **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 and load the variables from the target model to the base workspace.

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 F28379D based controller.

```
open_system('mcb_bldc_host_model_f28379d.slx');
```

BLDC Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_bldc_sixstep_f28379d

2. The variables from the target model are
   loaded to the base workspace.
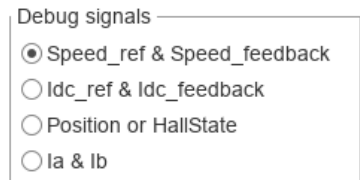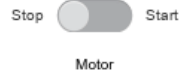
Debug signals
- ⊙ Speed_ref & Speed_feedback
- ○ Idc_ref & Idc_feedback
- ○ Position or HallState
- ○ Ia & Ib

--
Reference Speed (RPM)

Stop ◯ Start
Motor

Steps:
1. Set the serial port to **target.comport** in the init script
   of the target model.
2. Simulate this model.
3. Use Motor switch to control the motor.
4. Input speed request in the Reference Speed edit box.
5. Observe the debug signals in scope.

target.comport
Baud rate : 12e6

Host Serial Setup

Scope (Per-Unit)
Debug1 (SI units)
Debug2 (SI units)

Serial Communication

SelectedSignals
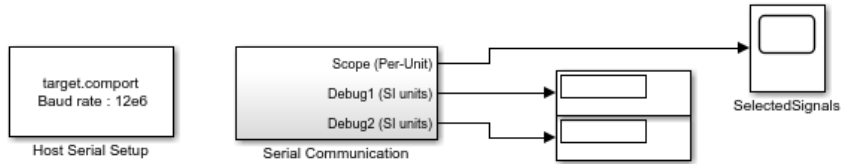
Copyright 2020-2022 The MathWorks, Inc.

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

11. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. This variable updates the **Port** parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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 **Motor** switch to **Start**, 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. This workflow helps you to spin the motor using six-step commutation without the need to label the hall sensors or derive the switching sequence. Run this example and obtain the hall sequence, and use this hall sequence with the Six Step Commutation block to run the motor in closed loop as explained in "Six-Step Commutation of BLDC Motor Using Sensor Feedback" on page 4-118 example.

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. Once the hall sequence with respect to rotor zero is obtained, use this hall sequence with Six Step Commutation block. And use same order of halls (to derive the hall sequence) obtained in this example in the "Six-Step Commutation of BLDC Motor Using Sensor Feedback" on page 4-118 example to run the motor in closed loop control.

When the rotor reaches the open-loop position zero, it aligns with the phase a-axis of the stator. At this position (for the corresponding 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. Refer to Six Step Commutation block and use the hall sequence obtained from this workflow.

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.
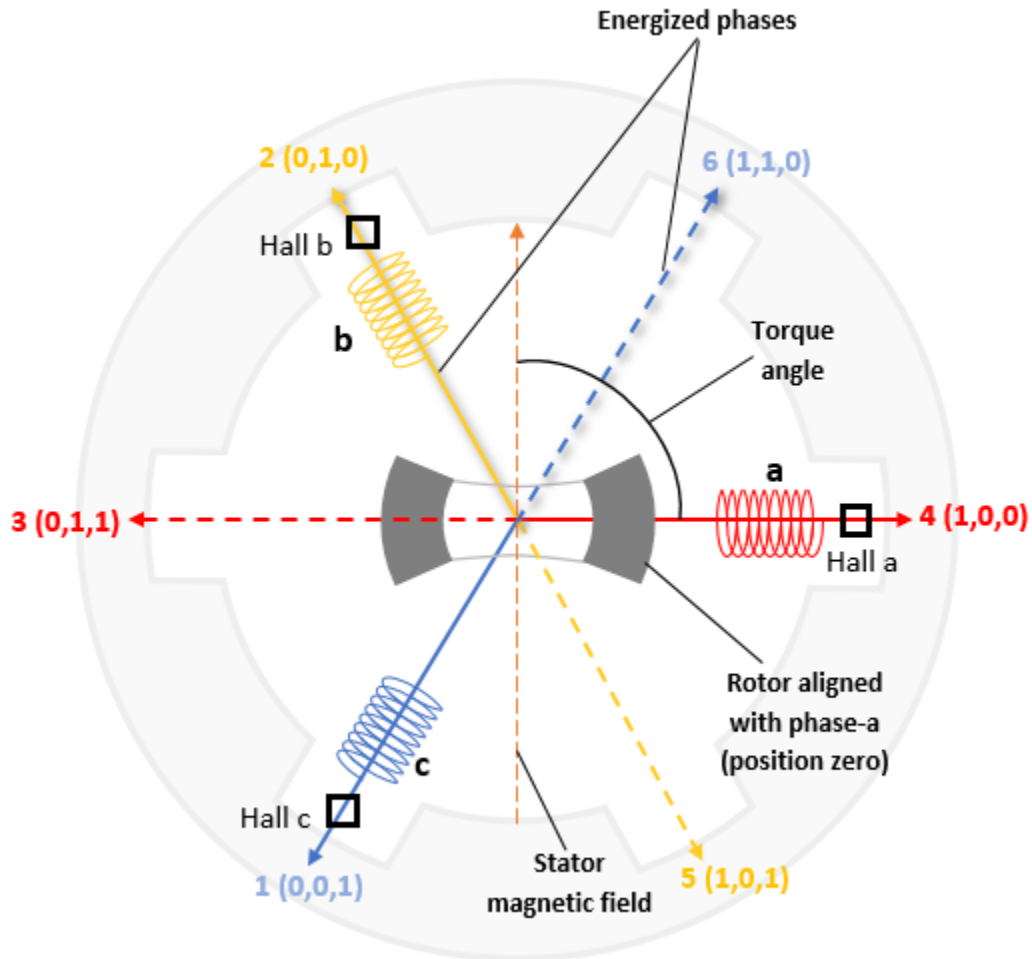
Note: For examples that use six-step commutation using a Hall sensor, update the computed Hall sequence value in the *bldc.hallsequence* variable in the model initialization script linked to the example. For instructions, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

Models

The example includes these models:

- mcb_hall_calibration_f28069mLaunchPad

- mcb_hall_calibration_f28379d.

You can use these models only for code generation. To open a Simulink® model, you can also use the open_system command at the MATLAB® command prompt. For example, use this command for a F28379D based controller:

```
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 2020-2023 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®
- C2000™ Microcontroller Blockset
- 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

The example supports these hardware configurations. You can also use the target model name to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

- LAUNCHXL-F28069M controller + BOOSTXL-DRV8305 inverter: mcb_hall_calibration_f28069mLaunchPad

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_hall_calibration_f28379d

4-125

For connections related to these hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

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');
```
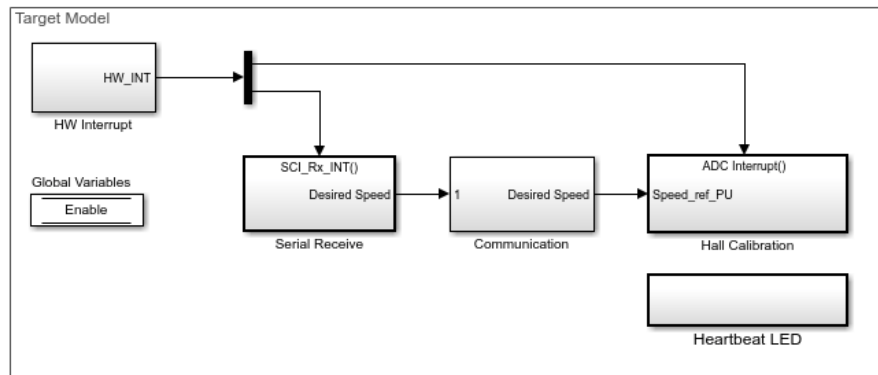
Hall Sequence Calibration Host

Prerequisites:

1. Deploy the target model to the hardware
mcb_hall_calibration_f28379d

2. You should see and verify the variables from the target model in the base workspace.

Steps:

1. Select port in Host Serial Setup, Host Serial Receive and Host Serial Transmit.
2. Simulate this model to start the Hall sequence calibration for six step control. Motor starts spinning when calibration starts.
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

--

Hall Sequence

No port selected

Host Serial Setup

OpenLoopPos

HallSequence

Serial Communication

Calibration in progress

Calibration complete

Emergency Motor Stop

Push for emergency stop

Scope

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 model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a **Port**.

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.
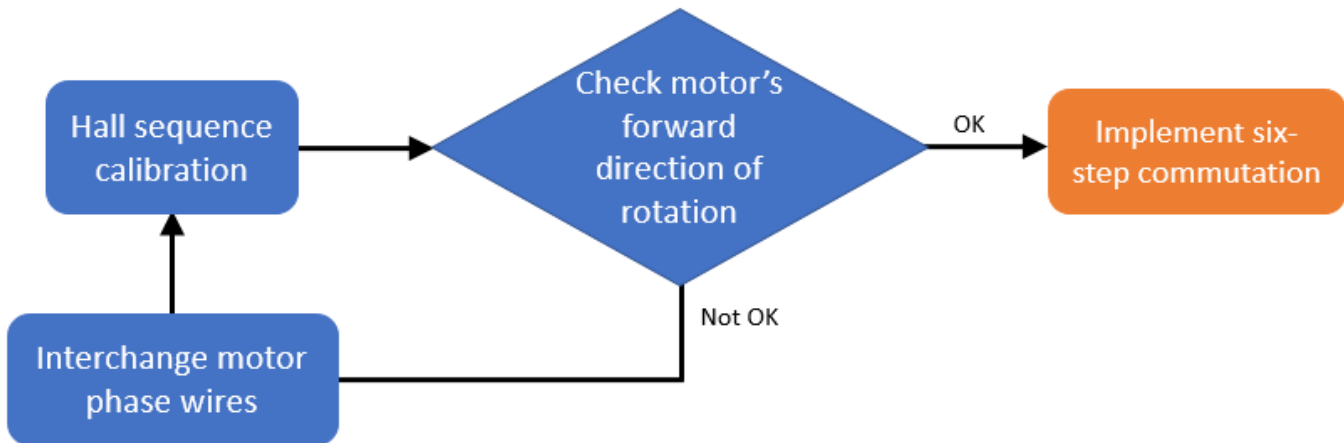
9. See these LEDs on the host model to know the status of calibration process:

- 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.

Position Control of PMSM Using Quadrature Encoder

This example implements the field-oriented control (FOC) technique to control the position of a three-phase permanent magnet synchronous motor (PMSM). The FOC algorithm requires rotor position feedback, which it obtains from a quadrature encoder sensor.

You can use this example to implement position control applications by using closed-loop FOC. The example drives the motor to reach the input reference-position value. You can also configure the maximum number of rotations (in either direction) for the motor in the model initialization script.

For details about closed-loop FOC, see "Field-Oriented Control (FOC)" on page 4-3 and "Closed-Loop Motor Control" on page 6-14.

Model

The example includes the mcb_pmsm_PosCtrl_f28379d model.

You can use this model for both simulation and code generation. You can also open the Simulink® model using this command at the MATLAB® Command Window.

```
open_system('mcb_pmsm_PosCtrl_f28379d.slx');
```

Permanent Magnet Synchronous Motor Position Control

HW Prerequisites
1. TI F8379D LaunchPad
2. BOOSTXL-DRV8305 Booster pack
    or BOOSTXL-3PhGaNInv
3. PMSM motor with QEP sensor

Steps:
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. Click **Build, Deploy & Start** in the Hardware tab.
5. Control motor via host model
6. Learn more about this example

Global Variables

| Enable | IaOffset | SpeedRef |

| EnClosedLoop | IbOffset | PosRef |

| Debug_signals |

Simulation Input & Output

Reference Position [degree]

Measured Position [degree]

Processor — Embedded Processor — Duty Cycles — M — Inverter and Motor - Plant Model

Simulation Feedbacks

Copyright 2020-2023 The MathWorks, Inc.

For details about the supported hardware configuration, see the Required Hardware topic 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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. The Simulink® model uses default parameters 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 PMSM Parameters Using Recommended Hardware" on page 4-186. The parameter estimation tool updates the `motorParam` variable (in the MATLAB® workspace) with the estimated motor parameters.

2. Update motor parameters. If you obtain the motor parameters from the datasheet or from other sources, update the motor and inverter parameters in the model initialization script associated with the Simulink® model. For instructions, see "Estimate Control Gains and Use Utility Functions" 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. Click **Run** on the **Simulation** tab to simulate the model.

3. Click **Data Inspector** in the **Review Results** section to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section shows 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. Before you can run the host model on the host computer, 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

The example supports this hardware configuration. You can also use the target model name to open the model from the MATLAB® command prompt.

LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter: mcb_pmsm_PosCtrl_f28379d

Note: When using the BOOSTXL-3PHGANINV inverter, ensure that you have proper insulation between the bottom layer of BOOSTXL-3PHGANINV and the LAUNCHXL board.

For connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

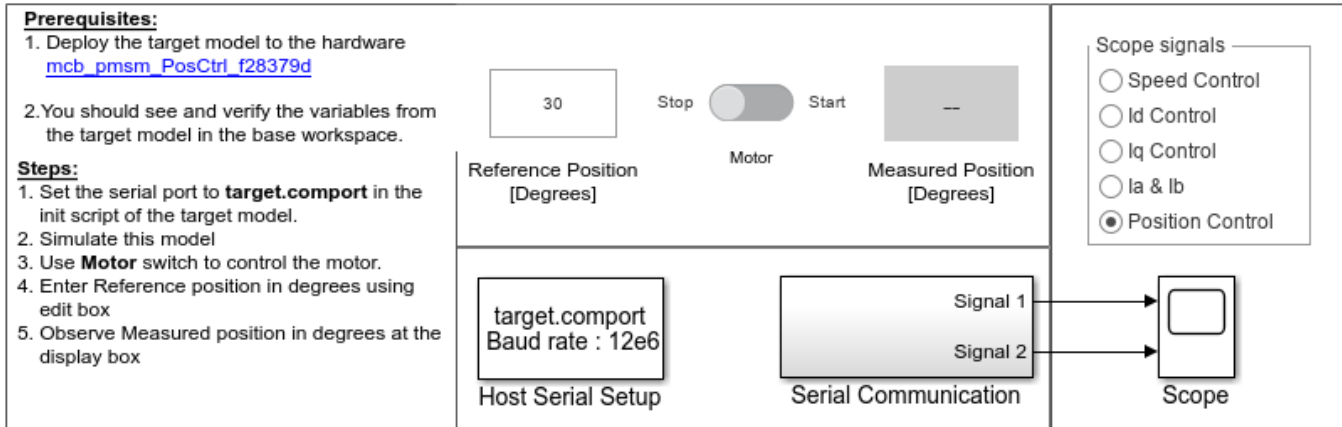**Generate Code and Run Model on Target Hardware**

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

2. Complete the hardware connections.

3. The model by default computes the ADC offset values for phase current measurement. To disable this functionality, update the value of the `inverter.ADCOffsetCalibEnable` variable in the model initialization script to `0`.

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 4-10.

4. 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 4-79.

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 the LAUNCHXL-F28379D board. For example, load the program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx). This ensures 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.

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

Position Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_PosCtrl_f28379d

2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Set the serial port to **target.comport** in the init script of the target model.
2. Simulate this model
3. Use **Motor** switch to control the motor.
4. Enter Reference position in degrees using edit box
5. Observe Measured position in degrees at the display box

| 30 | Stop ⬤ Start |
| Reference Position [Degrees] | Motor | Measured Position [Degrees] |

Scope signals
- ○ Speed Control
- ○ Id Control
- ○ Iq Control
- ○ Ia & Ib
- ⦿ Position Control

target.comport
Baud rate : 12e6

Host Serial Setup

Signal 1
Signal 2

Serial Communication

Scope

Copyright 2020-2022 The MathWorks, Inc.

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

9. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

10. Update the Reference Position [Degrees] value in the host model. By default, the maximum number of rotations (in either the positive or negative direction) is five. You can change this value by setting the `PosCtrlPosLimit` variable in the model initialization script. You can open this script by using the hyperlink named **Init script** in the target model.

Maximum rotation limit (degrees) = `PosCtrlPosLimit` x 360

Note: You cannot control the speed of rotation of the motor, but you can limit it by setting the `PosCtrlSpeedLimit` variable (in per-units). For details about the per-unit system, see "Per-Unit System" on page 6-20.

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

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

13. Observe the debug signals from the RX subsystem, in the Time Scope of host model. You can select the debug signals that you want to monitor in the **Scope signals** section of the host model.

- **Speed Control** - Display speed reference and speed feedback signals in the scope.

- **Id Control** - Display Id reference and Id feedback signals in the scope.

- **Iq Control** - Display Iq reference and Iq feedback signals in the scope.

- **Ia & Ib** - Display Ia and Ib current signals in the scope.

- **Position Control** - Display position reference and position feedback signals in the scope.

Integrate MCU Scheduling and Peripherals in Motor Control Application

This example shows how to identify and resolve issues with respect to peripheral settings and task scheduling early during development.

The following are typical challenges associated with MCU peripherals and scheduling:

- ADC-PWM synchronization to achieve current sensing at mid point of PWM period
- Incorporate sensor delays to achieve the desired controller response for the closed loop system
- Studying different PWM settings while designing special algorithms

This example shows how to use SoC Blockset to address these challenges for a motor control closed-loop application in simulation and verify on hardware by deploying on to the TI Delfino F28379D LaunchPad.
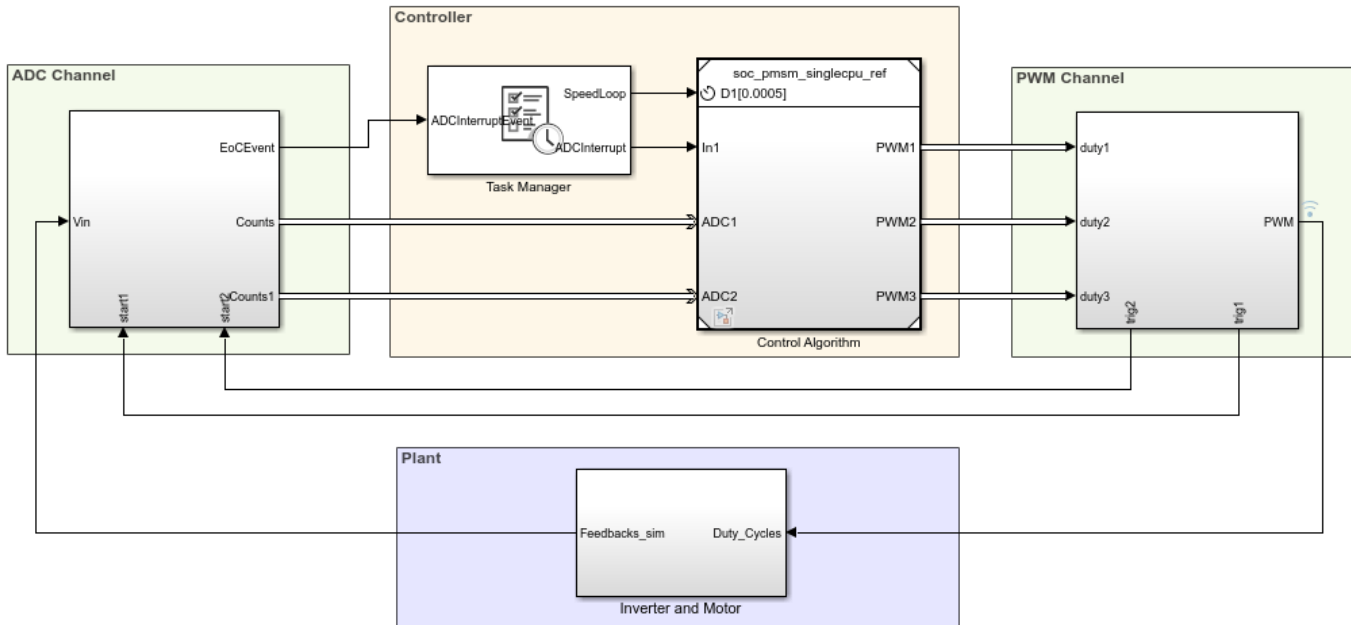
Required hardware:

- TI Delfino F28379D LaunchPad or TI Delfino F2837xD based board
- BOOSTXL-DRV8305EVM motor driver board
- Teknic M-2310P-LN-04K PMSM motor

Model Structure

```
open_system('soc_pmsm_singlecpu_foc');
```

Field Oriented Control In Single CPU

Copyright 2020 The MathWorks, Inc.

Open the soc_pmsm_singlecpu_foc model. This model simulates single CPU motor controller, contained in soc_pmsm_singlecpu_ref model, for a Permanent magnet synchronous motor inverter system. Controller senses the outputs from the plant using ADC Interface (SoC Blockset) and actuates using PWM Interface (SoC Blockset) that drives the inverter. Algorithm blocks from Motor Control Blockset™ is used in this example.

ADC Acquisition Time

ADC hardware contains a sample and hold circuit to sense the analog inputs. To ensure complete ADC measurement, the minimum acquisition time must be selected to account for the combined effects of input circuit and the capacitor in the sample and hold circuit.

Open ADC Interface block and change the default acquisition time to 100ns. Run the simulation and view the results in Simulation Data Inspector and observe there is a distortion in current waveforms. The low acquisition time resulted in ADC measurements not reaching their true value. As a result, the controller reacts by generating a relative duty cycle causing variations in current drawn by the motor. These figures show the reaction to the incorrect ADC measurement and overdraw in the phase A current channel, with phase A current in blue and phase B current in orange. The simulated speed feedback shows significant oscillations during open loop to closed loop transition, which in real world will halt the motor.
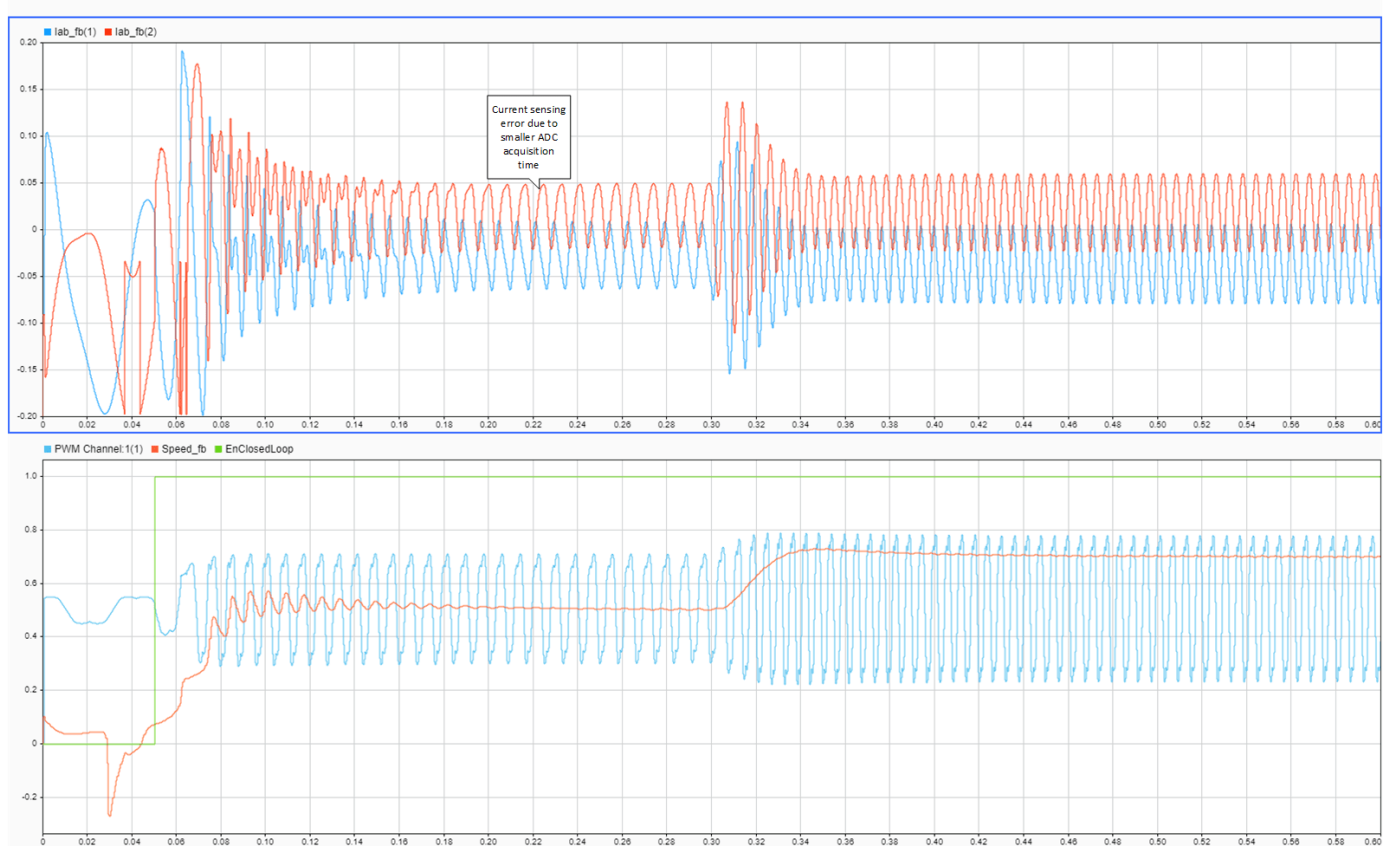
To fix this issue, open ADC Interface blocks change and change acquisition time to a larger value, 320ns. This value is above the minimum ADC acquistion time recommended in section ADC Operating Conditions (12-Bit Single-Ended Mode) of the TI Delfino F28379D LaunchPad data sheet. Run the simulation and view the results in Simulation Data Inspector. This figure shows the accurately sampled ADC values and the controller tracking the reference value as expected.
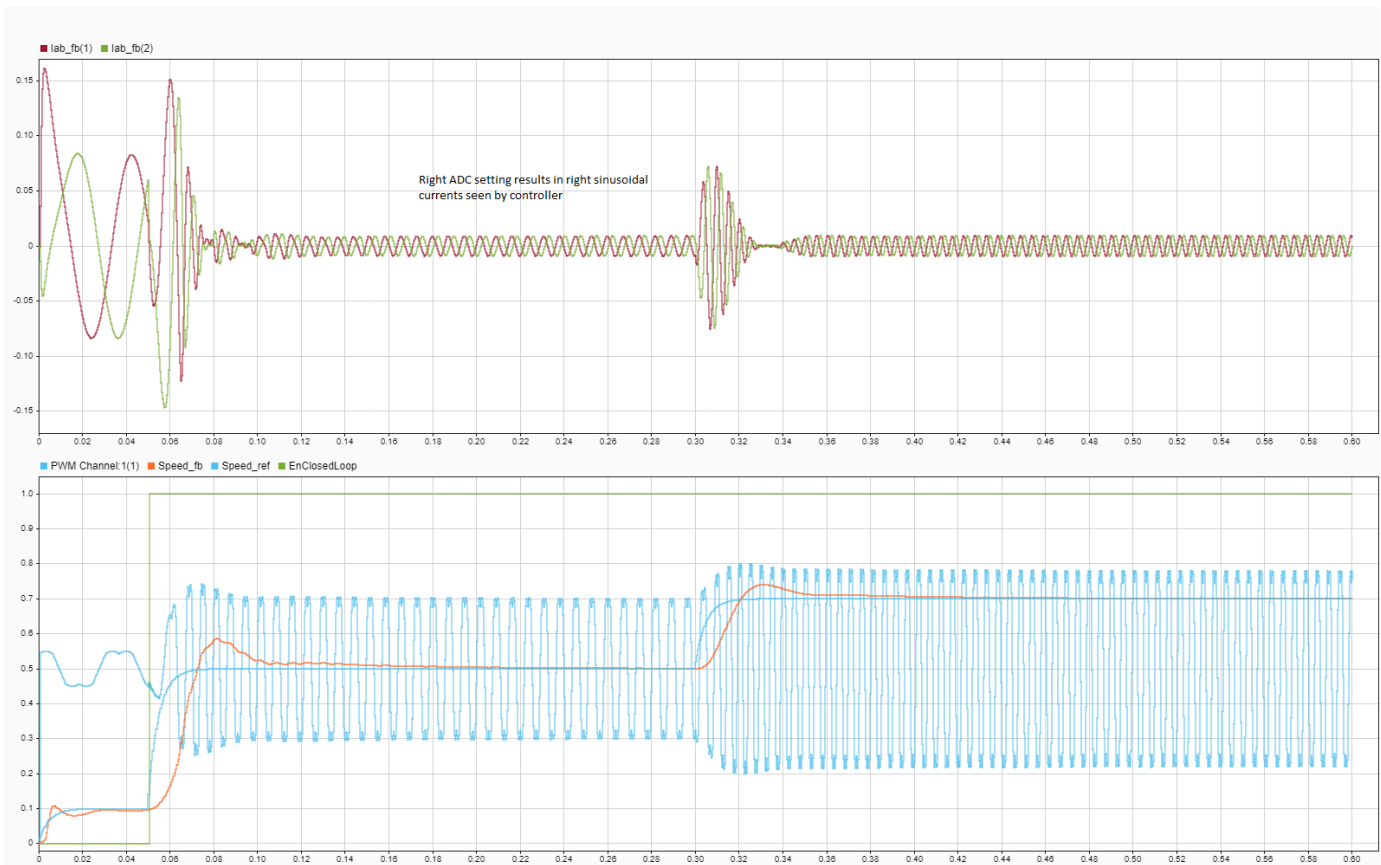
Verify the simulation results against hardware by deploying the model to the TI Delfino F28379D LaunchPad. On the **System on Chip** tab, click **Configure, Build, & Deploy** to open the SoC Builder (SoC Blockset) tool.

In the SoC Builder tool, on **Peripheral Configuration** tool, set **ADC > SOCx acquisition window cycles** parameter to `13 ADC clock ticks` for the ADC B and C modules. The ADC acquisition clock ticks parameter must be set to the simulation time value, set in the ADC Interface block, multiplied by the ADC clock frequency. You can get the ADC clock frequency from the model hardware settings. Open the soc_pmsm_singlecpu_ref model. On the **System on Chip** tab, click **Hardware Settings** to open the **Configuration Parameters** window. In the **Hardware Implementation > Target hardware resources > ADC_x** section, you can see the `ADC clock frequency in MHz` parameter value. This figure shows the ADC Interface block setting for simulation and peripheral app setting for deployment. Use same setting in simulation and codegen to ensure expected behavior.
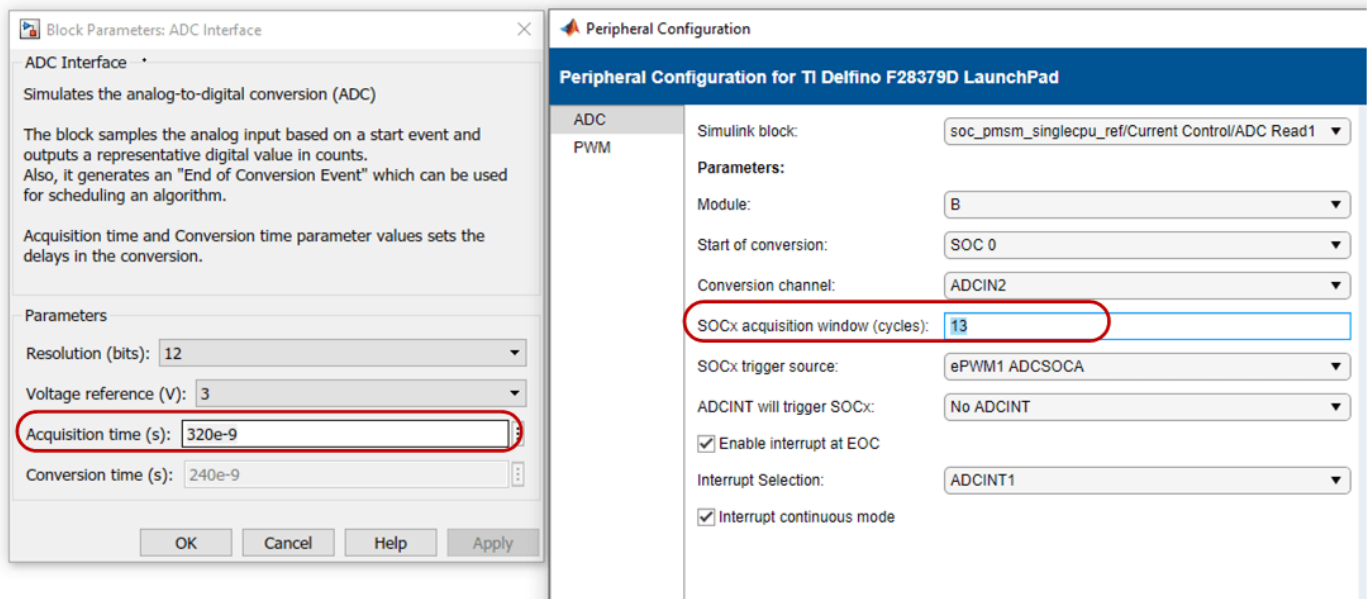
On **Select Build Action** page, to monitor data from hardware select `Build and load for External mode`. This figure shows the data from hardware with accurately sampled ADC values and the controller tracking the reference value as expected.
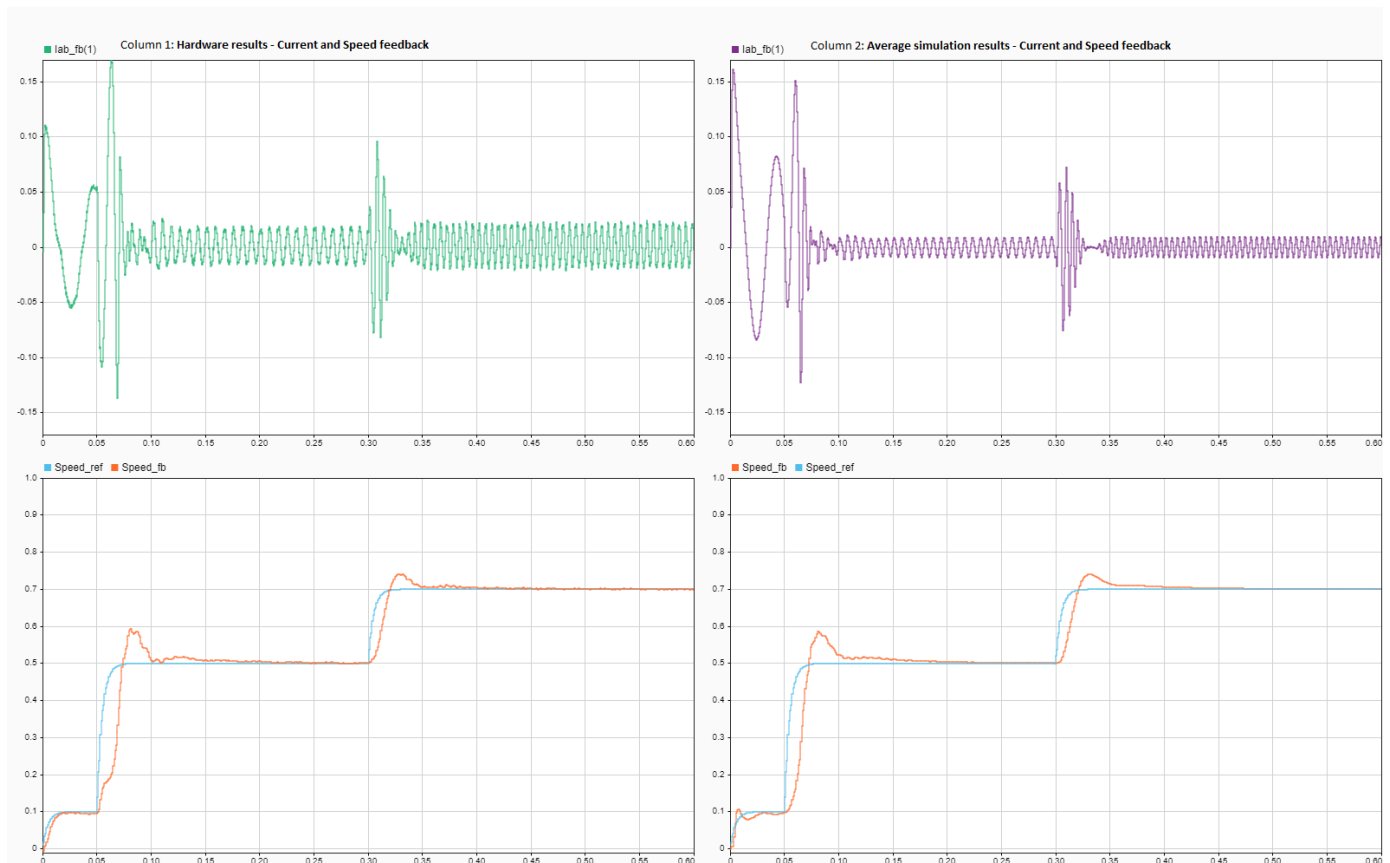
ADC-PWM Synchronization

The BOOSTXL-DRV8305EVM motor driver has a 3-phase inverter built using 6 power MOSFETS. This motor driver board uses a low-side shunt resistor to sense motor currents. The Current sense circuit amplifies the voltage drop across the shunt. This setup ensures low power dissipation, since the current only flows through the shunt when the bottom switches are on and away from PWM commutation noise. This figure shows the low-side shunt resistor circuit in BOOSTXL-DRV8305EVM motor drive.
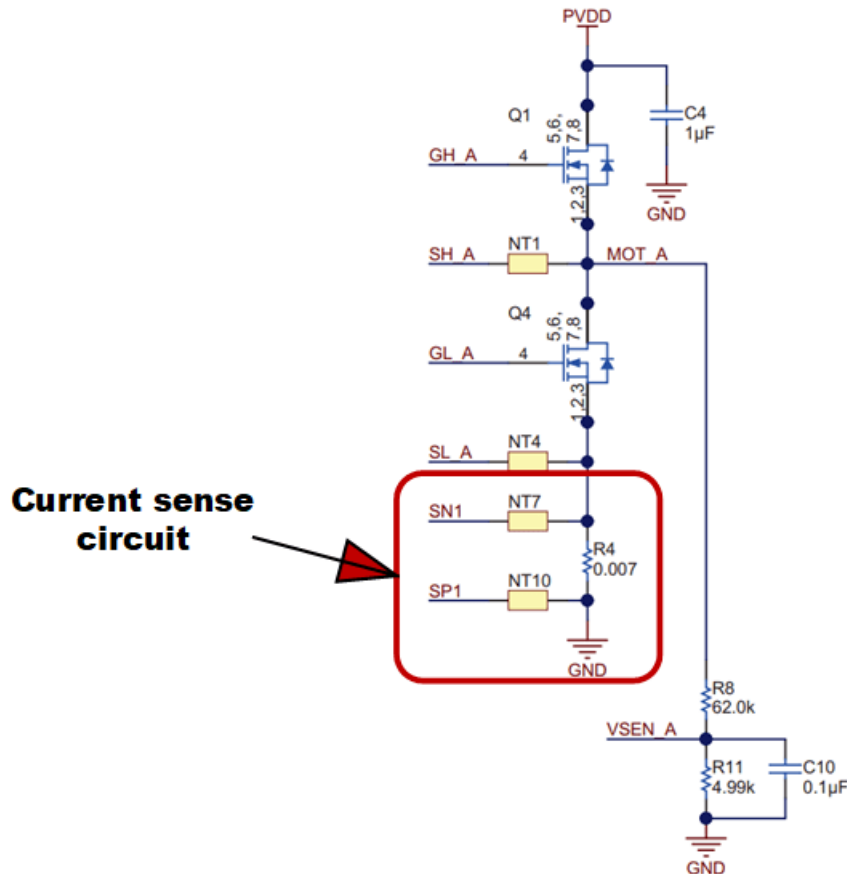
For correct operation, current sensing must occur during the mid point of the PWM period when ADCs trigger. Specifically, the PWM counter must be at the maximum value when the bottom switches are active in the Up-Down counter mode. Current sampling at a different instance results in a measured currents of zero.

To analyze this case, switch the model to **high fidelity inverter simulation** mode. Change the plant variant to use detailed MOSFET based 3-phase inverter to replicate BOOSTXL-DRV8305EVM.

```
set_param('soc_pmsm_singlecpu_foc/Inverter and Motor/Average or Switching',...
'LabelModeActivechoice','SwitchingInverter');
```

Change the `Output mode` parameter of PWM Interface (SoC Blockset) to `Switching` and connect 6 PWMs to the Mux block.

```
set_param('soc_pmsm_singlecpu_foc/PWM Channel/PWM Interface', 'OutSigMode', 'Switching');
set_param('soc_pmsm_singlecpu_foc/PWM Channel/PWM Interface1', 'OutSigMode','Switching');
set_param('soc_pmsm_singlecpu_foc/PWM Channel/PWM Interface2', 'OutSigMode', 'Switching');
```

Delete existing connection between PWM Interface block and Mux.

```
h = get_param('soc_pmsm_singlecpu_foc/PWM Channel/Mux','LineHandles');
delete_line(h.Inport);
```

As a last step, connect 6 PWM outputs to Mux.

```
set_param('soc_pmsm_singlecpu_foc/PWM Channel/Mux','Inputs','6');
```

```
add_line('soc_pmsm_singlecpu_foc/PWM Channel', ...
{'PWM Interface/1', 'PWM Interface/2', 'PWM Interface1/1',...
'PWM Interface1/2', 'PWM Interface2/1', 'PWM Interface2/2'}, ...
{'Mux/1','Mux/2','Mux/3','Mux/4','Mux/5','Mux/6'}, 'autorouting', 'smart');
```

Open the PWM Interface blocks and set **Event trigger mode** to `End of PWM period`. Run the simulation and view the results in Simulation Data Inspector. In the figure, phase A and phase B currents are approximately zero current. This results in a loss of feedback and no actuation in the control loop. Select `Enable task simulation` in Task Manager block to simulate and visualize tasks in Simulation Data Inspector.
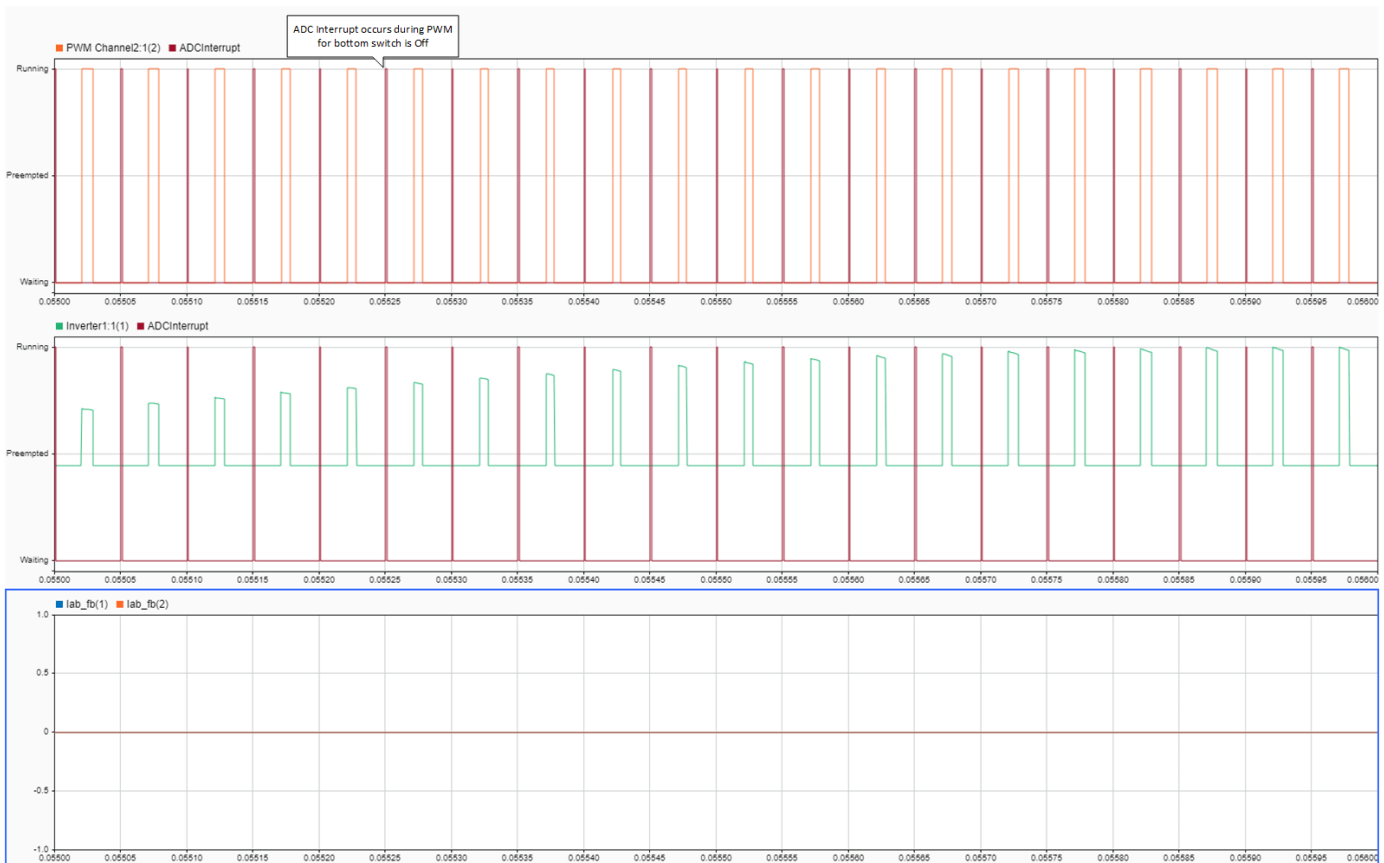
To fix this issue, change the **Event trigger mode** to `Mid point of PWM period`, equivalent to the PWM internal counter being at a maximum. Run the simulation and view the results in Simulation Data Inspector.
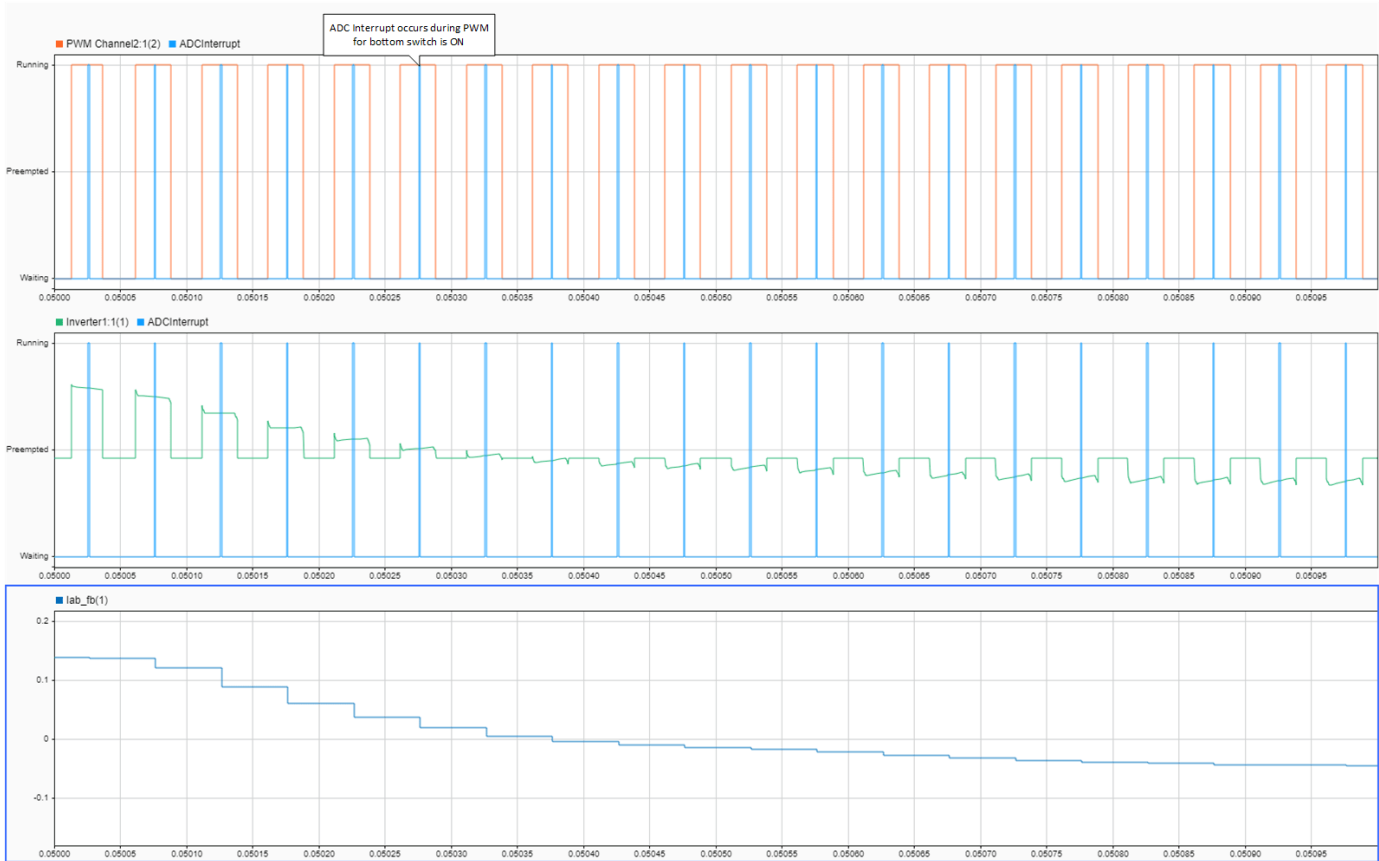
Deploy the model on to the TI Delfino F28379D LaunchPad using the SoC Builder (SoC Blockset) tool. In the SoC Builder tool, on **Peripheral configuration** tool, set **PWM event condition** to `Counter equals to period`. Use same setting in simulation and codegen to ensure expected behavior. This figure shows the PWM Interface block setting for simulation and the Peripheral Configuration tool setting for deployment.

Block Parameters: PWM Interface   ×

PWM Interface

Simulates the pulse width modulation (PWM)

The block outputs either a switching pulse width modulated
waveform or pass the duty cycle value to the output.
Also, it generates an "Event" which can be used for synchronizing
PWM with ADC or as PWM interrupts to trigger a control algorithm.

Parameters

1 PWM waveform period (s): 50e-6

    Output mode: Switching

2 Counter mode: Up-Down

3 Sampling mode: End of PWM period

4 Dead time (s): 100e-9

5 Event trigger mode: Mid of PWM period

    Data type: single

    [OK]   [Cancel]   [Help]   [Apply]

Peripheral Configuration

Peripheral Configuration for TI Delfino F28379D LaunchPad

ADC
PWM

Simulink block: soc_pmsm_singlecpu_ref/Current Control/PWM Write ▼

Parameters:

PWM Module: ePWM1

High speed clock divider: 1

Timer base clock divider: 1

1 Period (clock cycles): 5000

    ☐ Enable phase offset

2 Count mode: Up-Down

    Action on counter=zero: Do nothing

   Action on counter=period: Do nothing

    Action on counter=CMPA on up count: Clear

    Action on counter=CMPA on down count: Set

    ☑ Enable shadow mode

3 Reload CMPA register: Counter equals to zero (CTR=Zero)

5 ADC Start of conversion for ePWMxA module: Counter equals to period (CTR=PRD)

4 Dead band (cycles): 20

This figure shows the data from simulation and hardware with correct ADC-PWM synchronization and the controller tracking the reference value as expected.
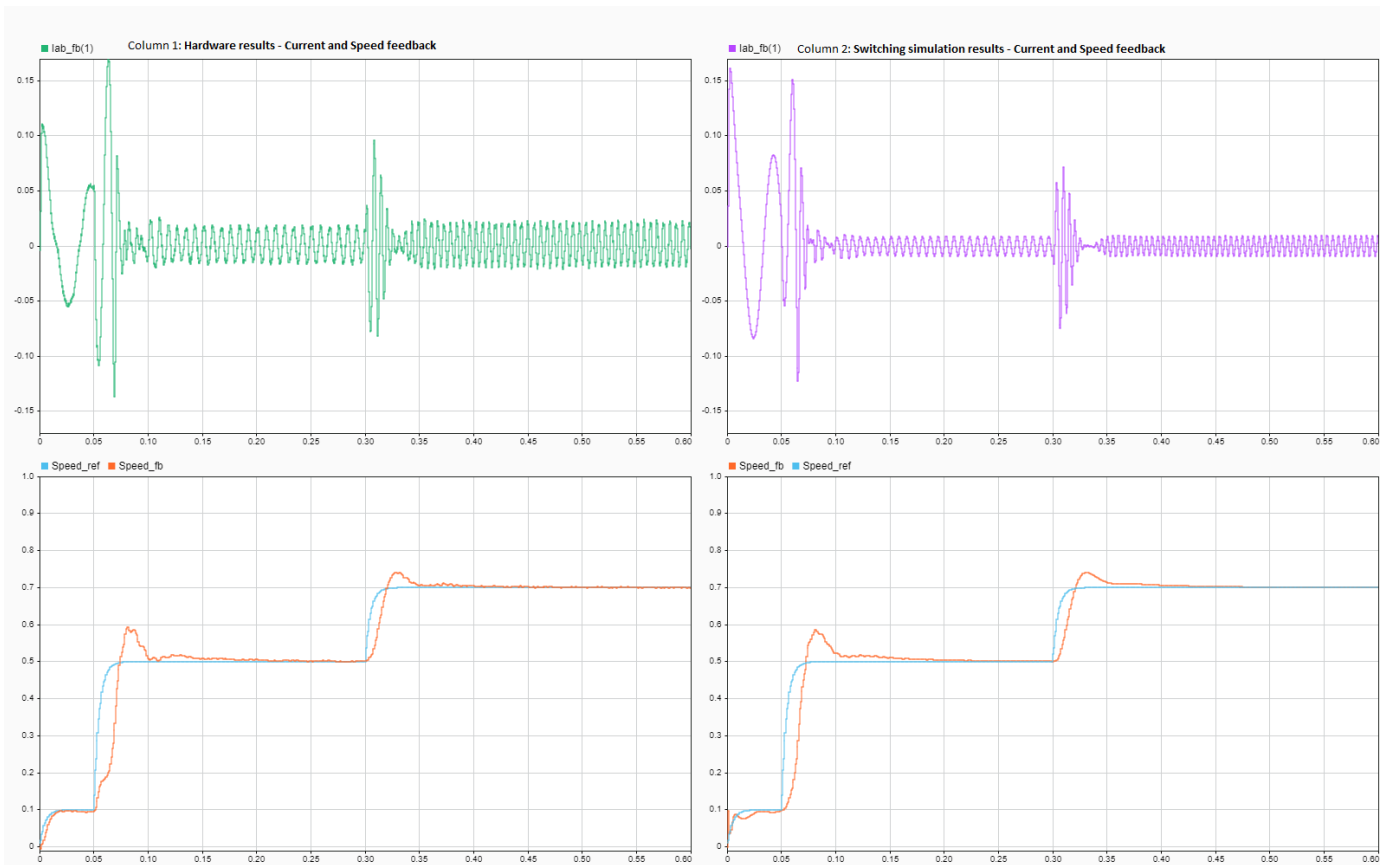
See Also

- "Get Started with Multiprocessor Blocks on MCUs" (SoC Blockset)
- "Partition Motor Control for Multiprocessor MCUs" on page 4-144

Copyright 2020-2021 The MathWorks, Inc.

Partition Motor Control for Multiprocessor MCUs

This example shows how to partition real-time motor control application on to multiple processors to achieve design modularity and improved control performance.

Many MCUs provide multiple processor cores. These additional cores can be leveraged to achieve a variety of design goals:

- Divide the application into real-time tasks, such as control laws, and non-real time tasks, such as external communication, diagnostics, or machine learning
- Partition the control algorithm to run on multiple CPUs to achieve higher loop rate
- Run the same application in multiple CPU's for safety critical applications

This example shows how to partition motor control application across two CPUs of the TI Delfino F28379D to achieve higher sampling time/PWM frequency.

Required hardware:

- TI Delfino F28379D LaunchPad or TI Delfino F2837xD based board
- BOOSTXL-DRV8305EVM motor driver board
- Teknic M-2310P-LN-04K PMSM motor

Partition Motor Control Algorithm

Open the soc_pmsm_singlecpu_foc model. This model simulates a single CPU motor controller, contained in the soc_pmsm_singlecpu_ref model, for a permanent magnet synchronous machine (PMSM).

## Permanent Magnet Synchronous Motor Field Oriented Control
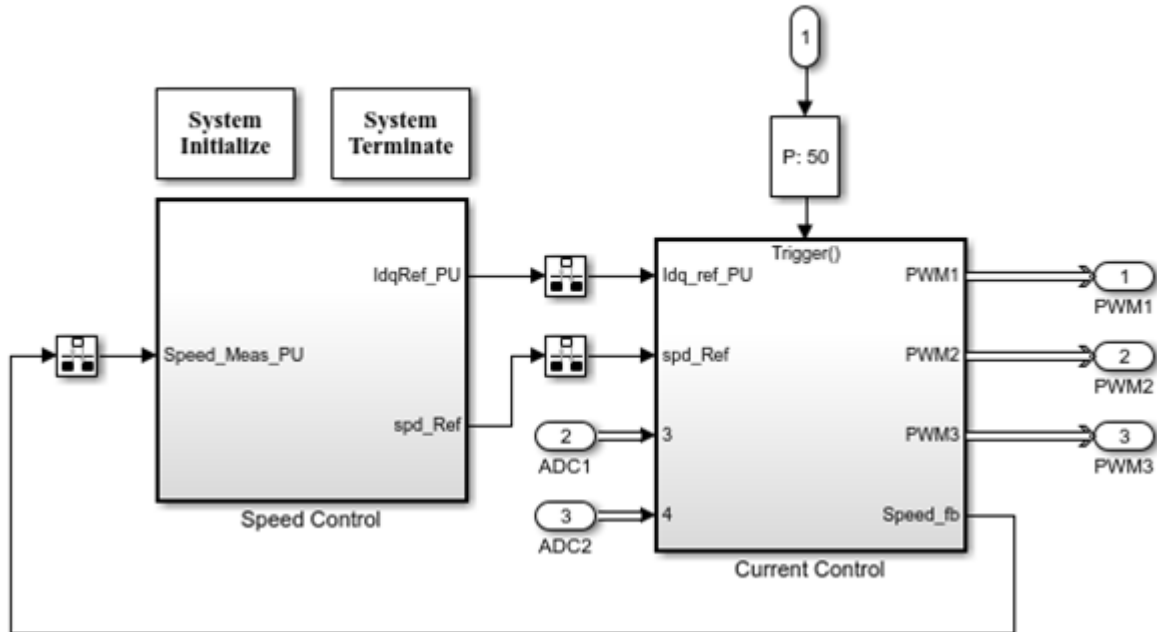
We partition the control algorithm by executing current control on CPU2, and speed control and position estimation on CPU1 respectively. Data transfer between the CPU's are handled by Interprocess Data Channel block. For more information see "Interprocess Data Communication via Dedicated Hardware Peripheral" (SoC Blockset).

Open the soc_pmsm_dualcpu_foc model.

```
open_system('soc_pmsm_dualcpu_foc');
```

## Field-Oriented Control on Dual CPU Processor
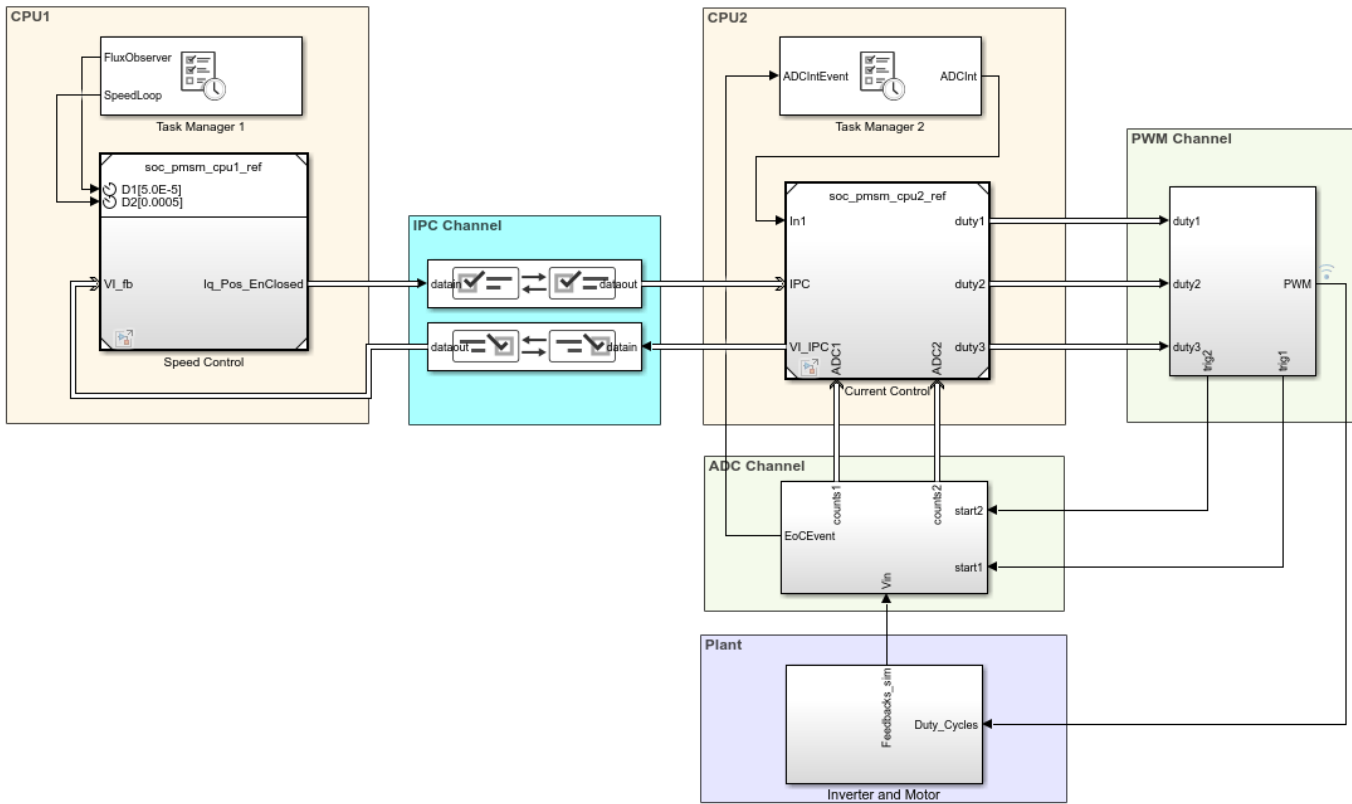
Copyright 2020 The MathWorks, Inc.

On the **System on Chip** tab, click **Hardware Settings** to open the **Configuration Parameters** window. In the **Hardware Implementation** tab, the **Processing Unit** parameter is configured to "None" indicating it is the top-level system model.

Open the soc_pmsm_cpu1_ref model and open the soc_pmsm_cpu2_ref model to view algorithms configured for each CPU. Model references contained within the system model are configured to run on c28xCPU1 (CPU1) and c28xCPU2 (CPU2).

On the Simulation tab, click **Run** to simulate the model. Open the Simulation Data Inspector and view signals. This figure shows results from the single and dual CPU models in simulation and deployment.
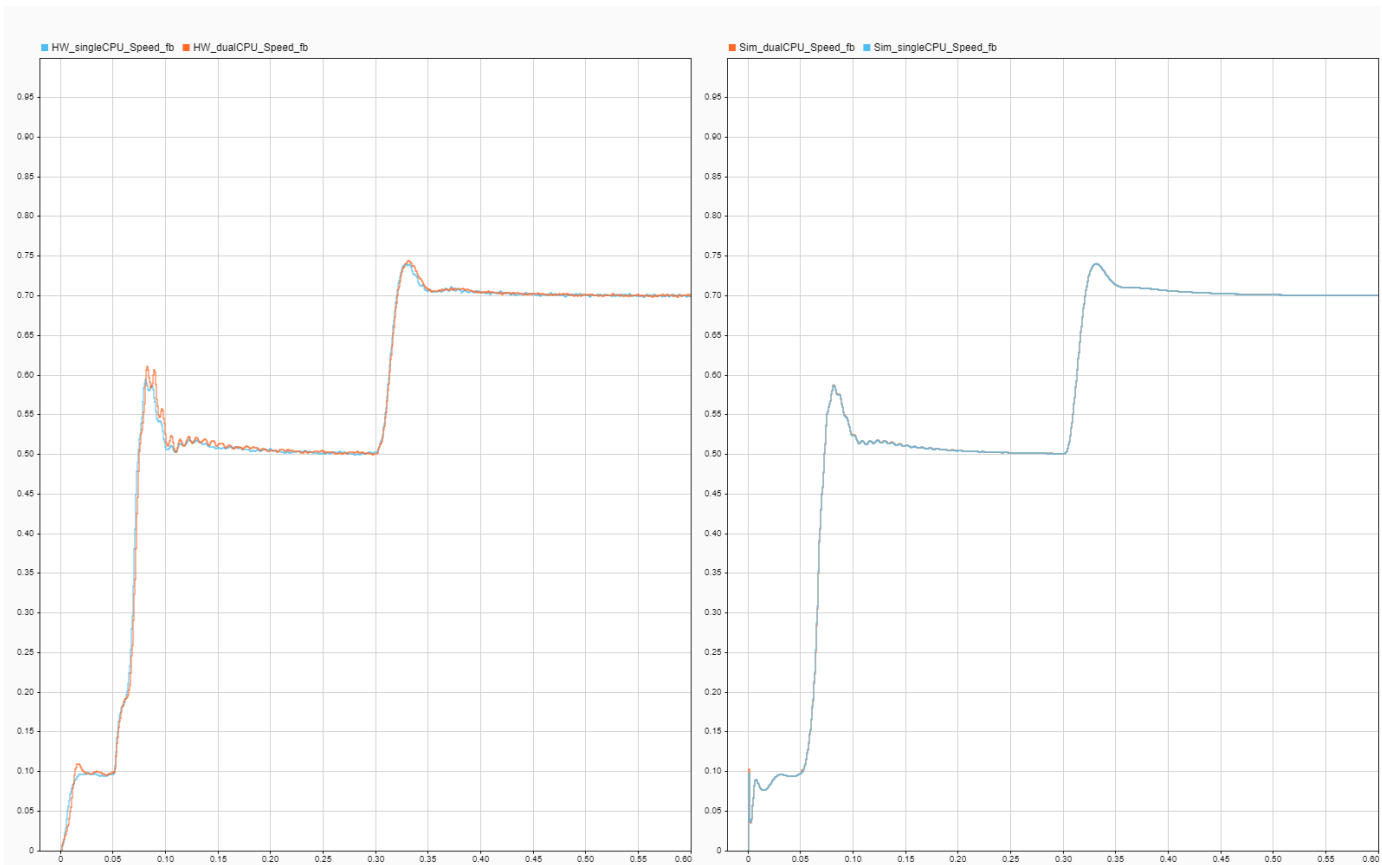
Performance Improvement with Concurrent Execution

Using both the CPUs to execute control algorithms allows us to achieve higher controller bandwidth. In the original single CPU model, the control algorithm takes just over 25us to execute. To provide a safety margin, single CPU model uses a PWM frequency of 20kHz, equivalent to 50us period.

After partitioning, the CPU1 and CPU2 execution times reduce to less than 20us. Allowing the PWM frequency to be increase to 40kHz. In the `soc_mcb_pmsm_foc_sensorless_f28379d_data.m` script, set PWM_frequency to 40e3 and run the script to configure the model to the new PWM frequency. With faster sampling of currents, controller gains can then be tuned to achieve faster response times.

Deploy the model to the TI Delfino F28379D LaunchPad using the SoC Builder (SoC Blockset) tool. To open the tool, on the **System on Chip** tab, click **Configure, Build, & Deploy** and follow the guided steps.

This figure shows the controller response from simulation and deployment at 25us current loop with 40kHz PWM frequency compared with 50us current loop at 20kHz frequency. As expected, the rise time in speed improves with faster current loop by approximately 50 percent.

4-147
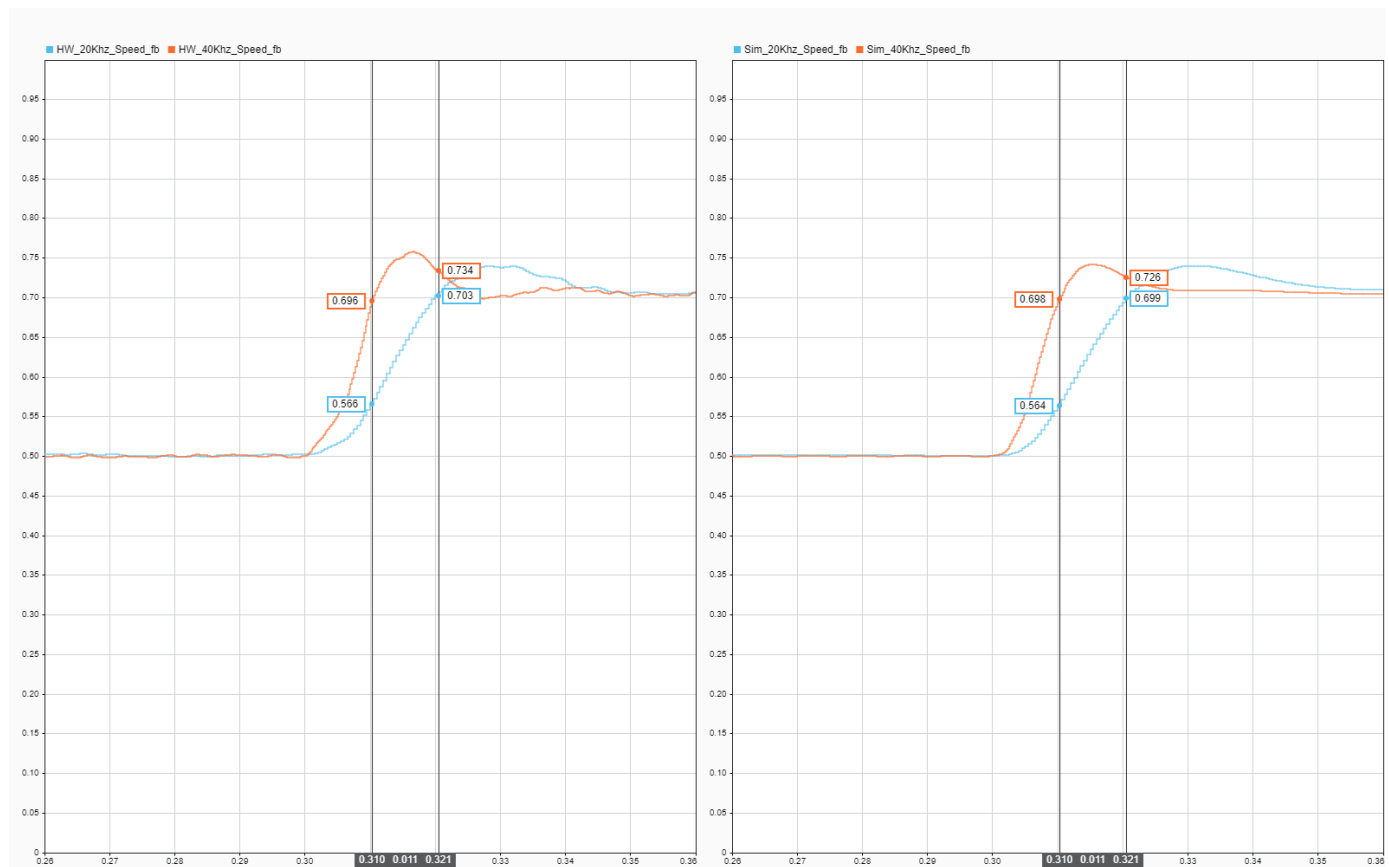
Speed response is oscillatory because of sensorless algorithm, for more information see "Sensorless Field-Oriented Control of PMSM" on page 4-59

For higher simulation granularity, set the PWM Interface block output to Switching Mode and change the plant model variant to use the MOSFET simulation.

See Also

- "Get Started with Multiprocessor Blocks on MCUs" (SoC Blockset)
- "Integrate MCU Scheduling and Peripherals in Motor Control Application" on page 4-134

Copyright 2020-2022 The MathWorks, Inc.

Frequency Response Estimation of PMSM Using Field-Oriented Control

This example performs frequency response estimation (FRE) of a plant model running a three-phase permanent magnet synchronous motor (PMSM). When you simulate or run the model on the target hardware, the model runs tests to estimate the frequency response as seen by each PI controller (also known as raw FRE data) and plots the FRE data to provide a graphical representation of the plant model dynamics.

When the motor runs in a steady state, the online Frequency Response Estimator block that is connected to each PI control loop (Id current, Iq current, and speed) sequentially perturbs the PI controller output and estimates the frequency response of the plant model as seen by each PI controller. You can use the frequency response of the plant to estimate the PI controller gains.
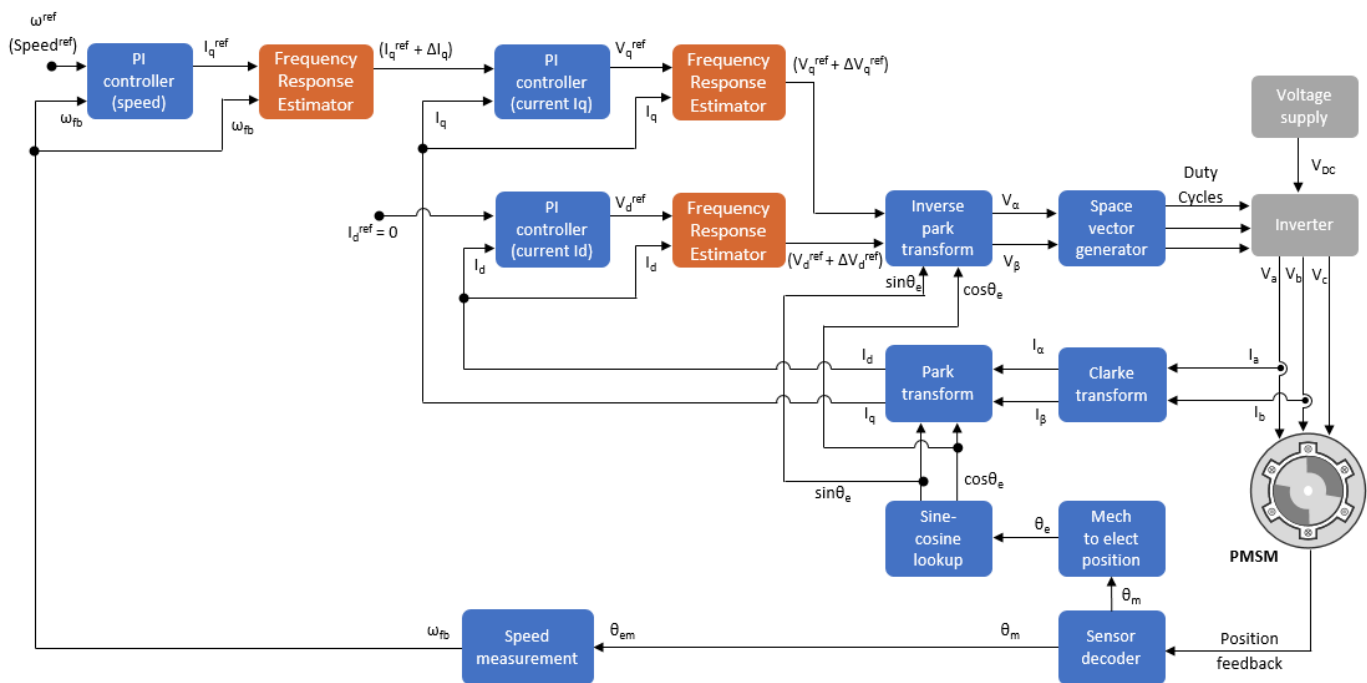
The model uses the field-oriented control (FOC) technique to control the PMSM. The FOC algorithm requires rotor position feedback, which is obtained by a quadrature encoder sensor. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-3.
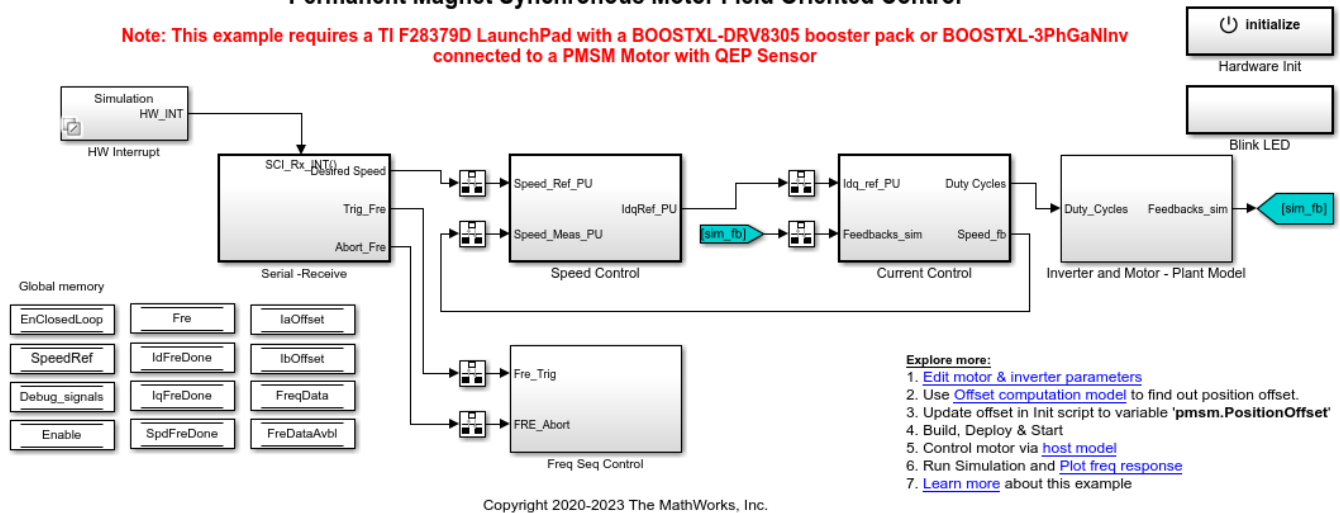
Models

The example includes the model (target model) mcb_pmsm_freq_est_f28379d.

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

```
open_system('mcb_pmsm_freq_est_f28379d.slx');
```

Permanent Magnet Synchronous Motor Field Oriented Control

For details regarding the supported hardware configuration, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

- Simulink Control Design™

- Stateflow® (needed only if you modify the example model)

To generate code and deploy model:

1. Motor Control Blockset

2. Embedded Coder®

3. C2000™ Microcontroller Blockset

4. Simulink Control Design

5. Stateflow (needed only if you modify the example model)

Prerequisites

1. Obtain the motor parameters. The Simulink® model uses default motor parameters 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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" 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 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. On the target model, click the **Plot freq response** hyperlink to plot the frequency response data of the plant model (`sys_sim_id`, `sys_sim_iq`, and `sys_sim_spd` variables in the workspace) that the speed control loop and the current control loops measure.

Generate Code and Deploy Model to Target Hardware

This section shows you how to generate code, run the FOC algorithm on the target hardware, start frequency response estimation, and plot the FRE data.

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. You can also use the target model name to open the model from the MATLAB® command prompt.

- LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter: mcb_pmsm_freq_est_f28379d

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

For connections related to the hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

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 variable `inverter.ADCOffsetCalibEnable` to `0` in the model initialization script.

4-151

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 4-10.

4. Compute the quadrature encoder index offset value and update it in the `pmsm.PositionOffset` variable available in the model initialization script associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79.

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, load the 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.
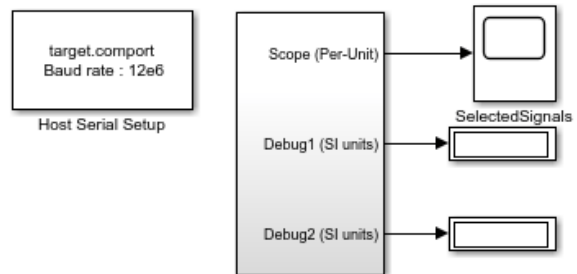
```
open_system('mcb_pmsm_freq_host_f28379d.slx');
```


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

9. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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

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

12. Select the debug signal that you want to monitor in the **Debug signals** section of the host model. Observe these signals in the SelectedSignals time scope window.

13. Click the **FRE Trigger** button to start the FRE process on the target hardware.

14. Select **Position & Raw FRE data** in the **Debug signals** section of the host model to start receiving the raw FRE data from the target hardware. The **FRE Status** LED turns amber to indicate that the host model is receiving raw FRE data from the target hardware.

Note: The LED shows the correct status only when you select **Position & Raw FRE data** in the **Debug signals** section. Otherwise, the LED remains grey.

15. Check the status of the **FRE Status** LED on the host model. The LED turns green after the host model receives all the raw FRE data from the target hardware.

16. Click the **FRE Plot** button to plot the raw FRE data received from the target hardware.

17. On the host model, click **Stop** on the **Simulation** tab to stop the simulation.

18. Click the **compare** hyperlink in the host model to plot the raw FRE data generated during simulation and hardware run and compare them.

For an accurate comparison, use the same reference speed during simulation and when running the example on the target hardware.
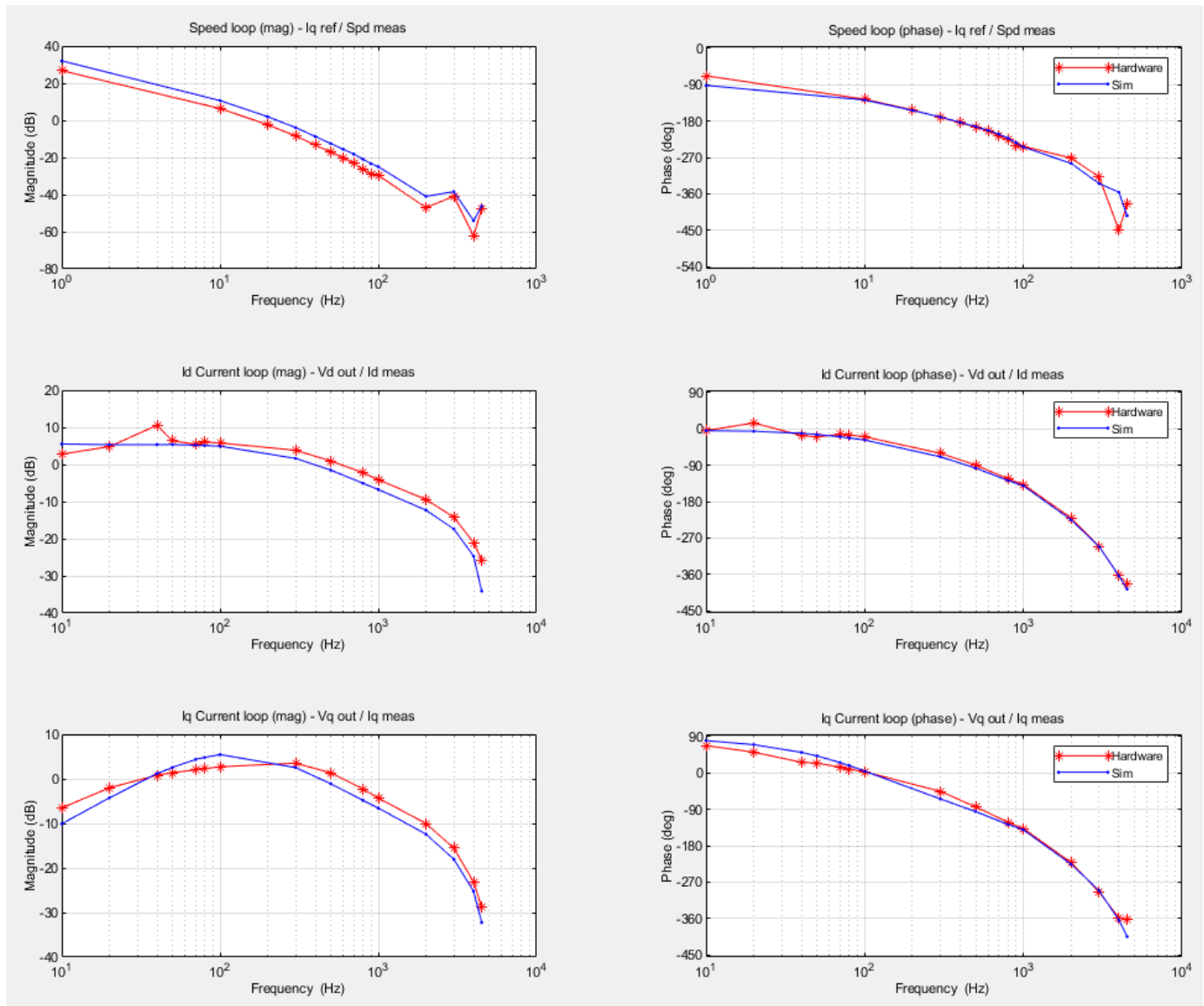
NOTE:

- To stop the FRE process any time, click the **FRE Abort** button.

- To stop the motor immediately, turn the Start / Stop Motor switch Off.

Configure Frequency Response Estimator Block

Configure these parameters in the Frequency Response Estimator block (from Simulink Control Design™ toolbox) mask:

- Sample time (Ts) - Enter a block sample time that is identical to that of the PI controller.

- Frequencies - Enter an array of frequencies at which the block perturbs the PI controller output to estimate the frequency response of the plant. This field uses the (single data type) workspace variable `fre.i_freq` to store the array of perturbation signal frequencies.

Note: By default, the model uses an array size of 15. However, you can configure the array size.

The **start/stop** signal value of 1 that started the FRE experiment should change to 0 only after the perturbations and tests for all the frequencies are complete and the FRE experiment ends.

- Amplitudes - Enter the amplitude of the perturbation signals that the block applies to the PI controller output to estimate the frequency response of the plant. This field uses the (single data type) workspace variable `fre.i_amp` to store the common amplitude value of the perturbation signals.

A high amplitude produces disturbances when the motor runs. An amplitude that is too low results in an inaccurate FRE.

For more details about the Frequency Response Estimator block, see Frequency Response Estimator (Simulink Control Design).

Frequency Response Estimator Block Output

The Frequency Response Estimator block (connected to each PI controller) performs an FRE experiment by perturbing the PI controller output using the sequence of frequencies stored in `fre.i_freq`.

For each perturbation signal (represented by a frequency) the block estimates the plant frequency response in the form of a complex number. Therefore, the block uses the array of frequencies to generate an array of complex numbers (raw FRE data). The sequence of complex numbers contains the information related to gain and phase delay.

Controlling FRE Experiments

The State Machine Control subsystem algorithm enables the three Frequency Response Estimator blocks one at a time (and runs the three FRE experiments) in this order by using the **start/stop** input port of the Frequency Response Estimator block:

1. FRE block connected to Id control loop

2. FRE block connected to Iq control loop

3. FRE block connected to speed control loop

The state machine control ensures that the time interval between the start and stop signals is greater than or equal to the FRE experiment length (as displayed by the Frequency Response Estimator block dialog box). If you change the perturbation signal frequencies, ensure that the state machine control sends the stop signal only after the FRE experiment ends.

For more details about the Frequency Response Estimator block, see Frequency Response Estimator (Simulink Control Design).

Plot Frequency Response After Simulation

After the simulation ends, the target model stores the frequency response (or the raw FRE data) in these workspace variables:

- `out.Idfreqdata`- Raw FRE data for the Id current PI controller.

- `out.Iqfreqdata` - Raw FRE data for the Iq current PI controller.

4-155

- `out.Spdfreqdata` - Raw FRE data for the speed PI controller.

When you click the **Plot freq response** hyperlink in the target model, the model plots the frequency response for the three PI controllers.
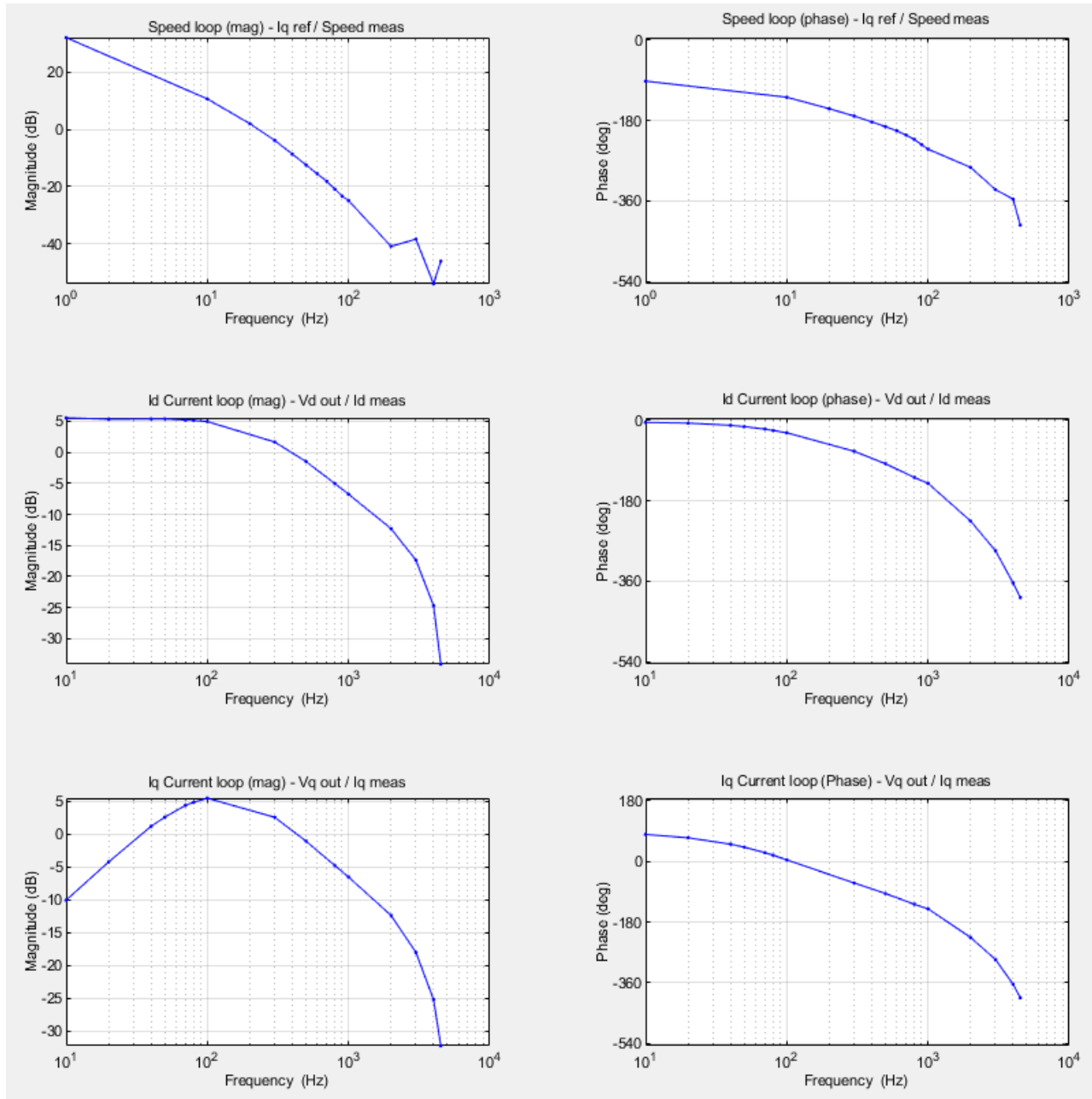
The target model uses these commands to plot the frequency responses as seen by the three PI controllers.

Frequency response of Id current PI controller:

```
sys_sim_id = frd(out.Idfreqdata,fre.i_freq*2*pi);
```

```
bode(sys_sim_id);
```

Frequency response of Iq current PI controller:

```
sys_sim_iq = frd(out.Iqfreqdata,fre.i_freq*2*pi);
```

```
bode(sys_sim_iq);
```

Frequency response of speed PI controller:

```
sys_sim_spd = frd(out.Spdfreqdata,fre.spd_freq*2*pi);
```

```
bodeplot(sys_sim_spd);
```

For more information about these commands, see these files:

- `mcb_pmsm_freq_est_plot.m`

- `mcb_pmsm_freq_host_est_plot.m`

Send Raw FRE Data to Host Model

When running the target model on the hardware, the target model transfers the raw FRE data continuously to the host model.

The target model splits the entire sequence of complex numbers (or raw FRE data) from each FRE block into real and imaginary arrays and adds headers to separate them. It uses this format to send the raw FRE data from each FRE block to the host model using serial communication.
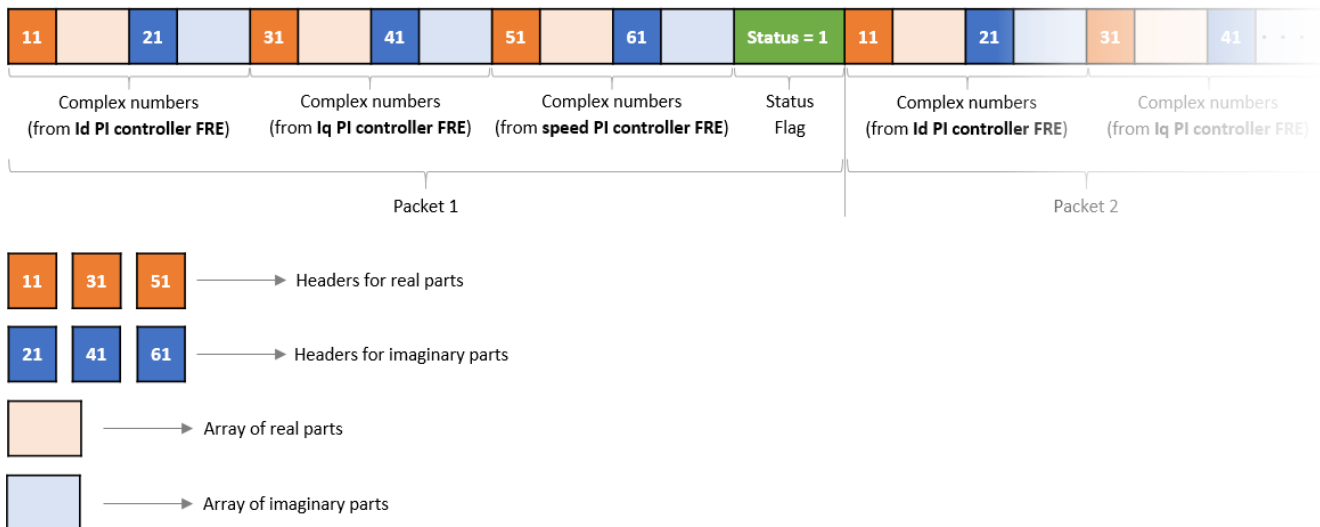
Plot Frequency Response When Using Target Hardware

After receiving the message from the target hardware, the host model decrypts the message and stores the array of complex numbers (raw FRE data) in these workspace variables:

- `IdfreqData` - Raw FRE data for the Id current PI controller.

- `IqfreData` - Raw FRE data for the Iq current PI controller.

- `SpdfreqData` - Raw FRE data for the speed PI controller.

When you click the **FRE Plot** button, the host model plots the frequency response for the three PI controllers.
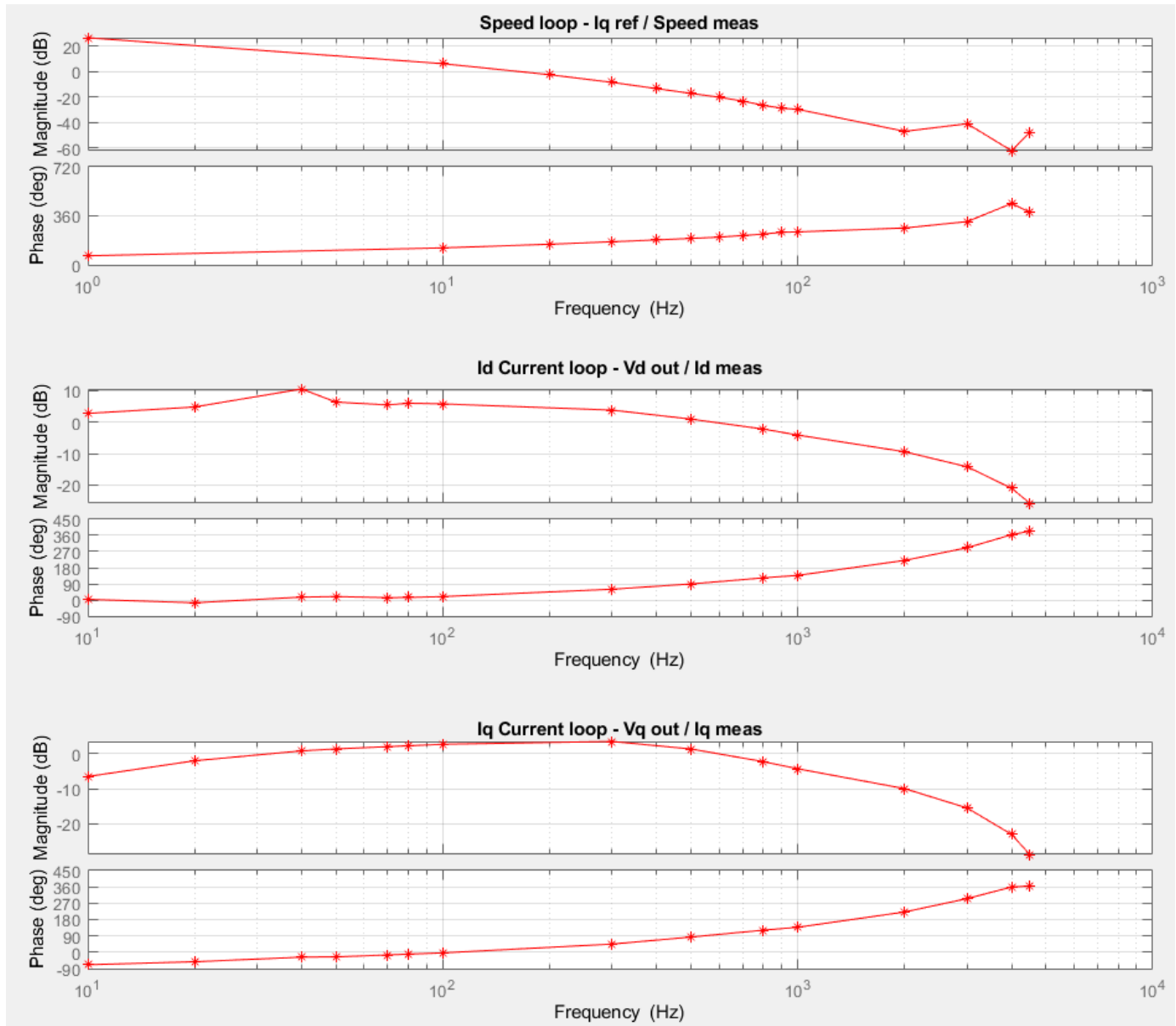
The host model uses these commands to plot the frequency responses observed for the three PI controllers.

Frequency response of Id current PI controller:

```
sys_hw_id=frd(IdFreqData.signals.values,fre.i_freq*2*pi);
```

```
bode(sys_hw_id);
```

Frequency response of Iq current PI controller:

sys_hw_iq=frd(IqFreqData.signals.values,fre.i_freq*2*pi);

bode(sys_hw_iq);

Frequency response of speed PI controller:

sys_hw_spd=frd(SpdFreqData.signals.values,fre.spd_freq*2*pi);

bode(sys_hw_spd);

For more information about these commands, see these files:

- mcb_pmsm_freq_est_plot.m
- mcb_pmsm_freq_host_est_plot.m

Tuning PI Controller Gains

These steps show you how to tune and determine the gains for the Id and Iq currents and the speed of the PI controllers:

1. Navigate to the **Apps** tab on the Simulink toolstrip and open the **PID Tuner** app.
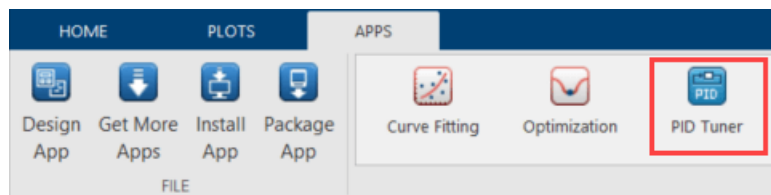
2. In the **PID Tuner** tab, select **PI** for **Type**, **Parallel** for **Form**, and **Frequency** for **Domain**.
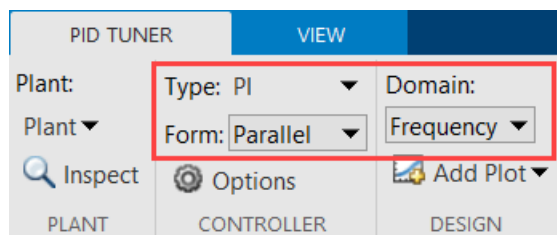
3. Under **Plant**, select **Import** to open the **Import Linear System** window.
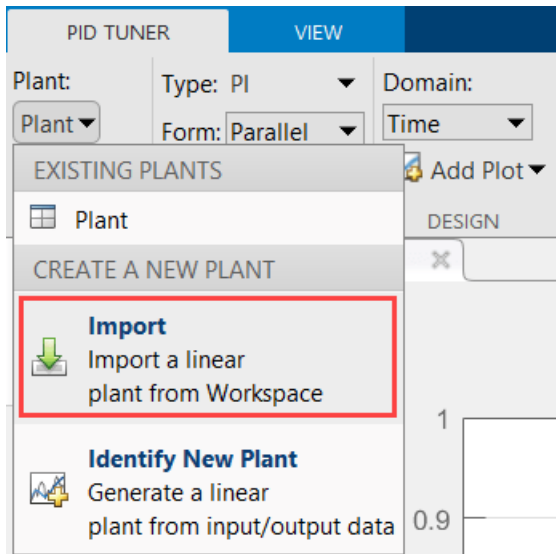
4. In the **Import Linear System** window, select **sys_sim_id** and click **Import** to import the FRE data for the Id PI controller.
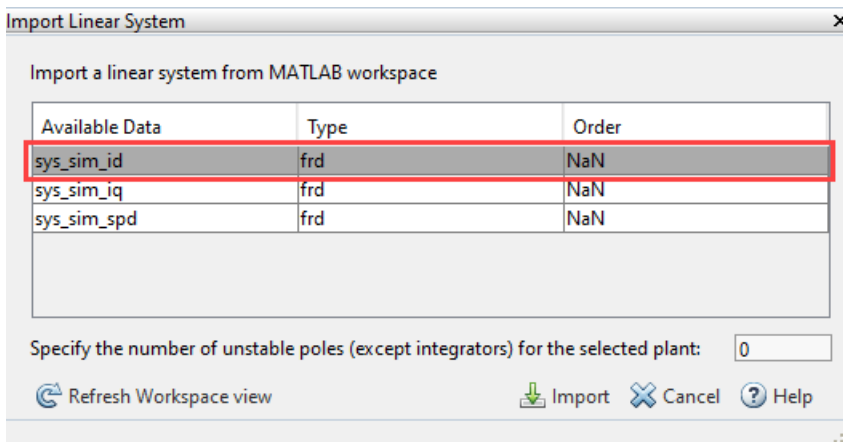
5. Select **Add Plot > Bode > Open-loop** to open the open-loop bode plot for the Id PI controller.
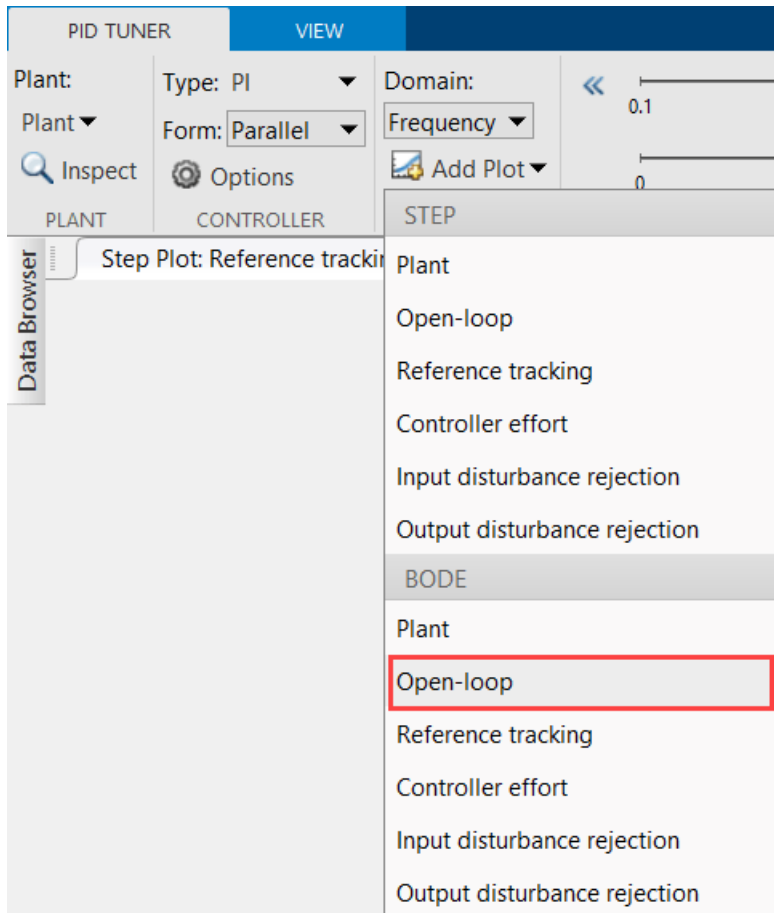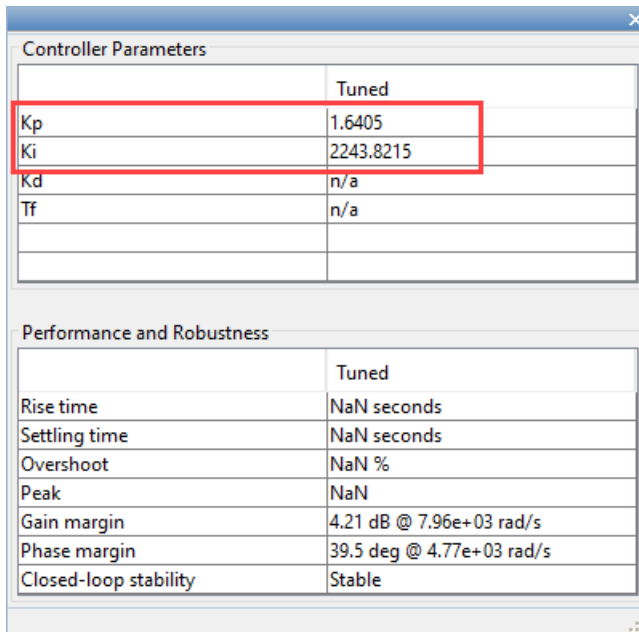
6. Use the **Tuning Tools** section in the **PID Tuner** tab to tune the bandwidth and phase margin and observe the results in the open-loop and plant bode plots.
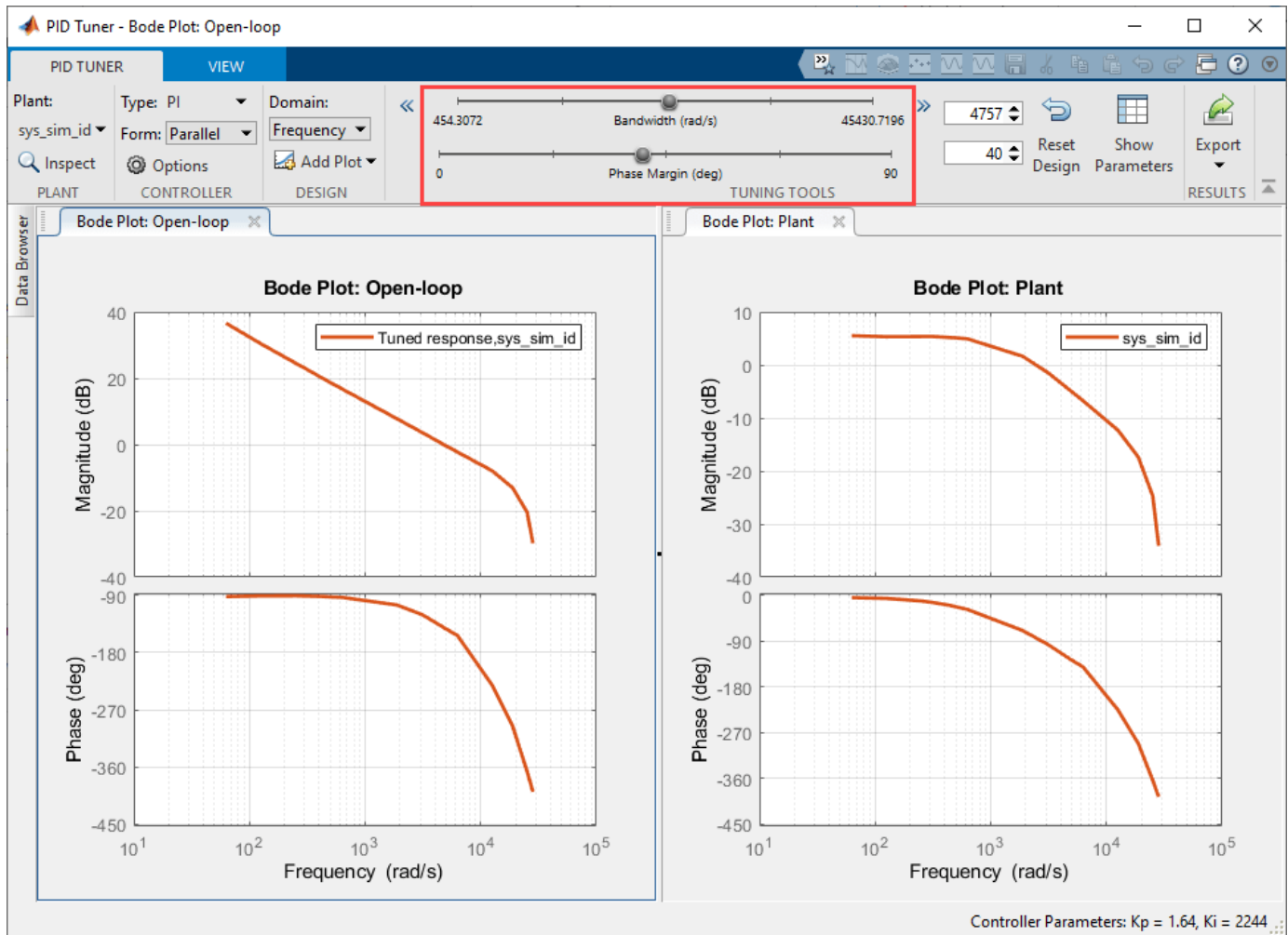
7. After you finish tuning, click **Show Parameters** to display the tuned controller parameters Kp and Ki for the Id current PI controller.

8. Repeat steps 3 to 7 by selecting **sys_sim_iq** in the **Import Linear System** window to obtain the tuned parameters Kp and Ki for the Iq PI controller.

9. Update the Kp and Ki gain values for both Id and Iq current PI controllers in the initialization script of the example model `mcb_pmsm_freq_est_f28379d.slx`. For instructions, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

10. Perform the frequency response estimation again using the updated PI controller gains by either simulating the example or running it on the target hardware.

11. Perform steps 3 to 7 by selecting **sys_sim_spd** in the **Import Linear System** window to obtain the tuned parameters Kp and Ki for the speed PI controller.

See Also

• "PID Controller Tuning in Simulink" (Simulink Control Design)

Other Things to Try

You can try estimating the transfer functions and state-space models from the FRE data by using these functions from the System Identification Toolbox™:

• `ssest`

• `tfest`

MATLAB Project for FOC of PMSM with Quadrature Encoder

This MATLAB® project provides a motor control example model that uses field-oriented control (FOC) to run a three-phase permanent magnet synchronous motor (PMSM) in different modes of operation. Implementing the FOC algorithm needs real-time rotor position feedback. This example uses a quadrature encoder sensor to measure the rotor position. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-3.

The example can run a motor in these modes:

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

- **Calibration** - In this mode, the example calibrates the ADC (or current) offset and the quadrature encoder offset (offset between the d-axis of the rotor and the encoder index pulse position as detected by the quadrature encoder sensor).

- **Open Loop Speed Control** - In this mode, the example controls the rotor speed by running the motor in the open-loop control.

- **Closed Loop Torque Control** - In this mode, the example controls the torque output of the motor by running it in the closed-loop control.

- **Closed Loop Speed Control** - In this mode, the example controls the rotor speed by running the motor in the closed-loop control.

Note: When running the example model on the hardware, we recommend that you stop the motor (by switching to the StandBy mode) before transitioning from one operating mode to another.

Open MATLAB Project

Use one of these methods to open the MATLAB project to follow this workflow:

1. Click **Open Example**.

2. Run the command `mcb_QEPWorkflowDemoStart` at the command prompt.

Model

The MATLAB project includes the model `mcb_qep_workflow`.

This model (also called target model) automatically opens when you open the MATLAB project. You can also use the Project window to open this model available in the `model` folder.

Workflow for FOC of PMSM with QEP sensor

HW Prerequisites
1. TI F28379D LaunchPad
2. BOOSTXL-DRV8305 Booster pack or BOOSTXL-3PhGaNInv
3. PMSM motor with QEP sensor

Steps:
1. Edit motor & inverter parameters and run the script to update the Simulink Data Dictionary file.
2. Simulate the model to see motor operation in different modes
3. Click **Build, Deploy & Start** in Hardware tab
4. Control motor via host model
5. Learn more about this example
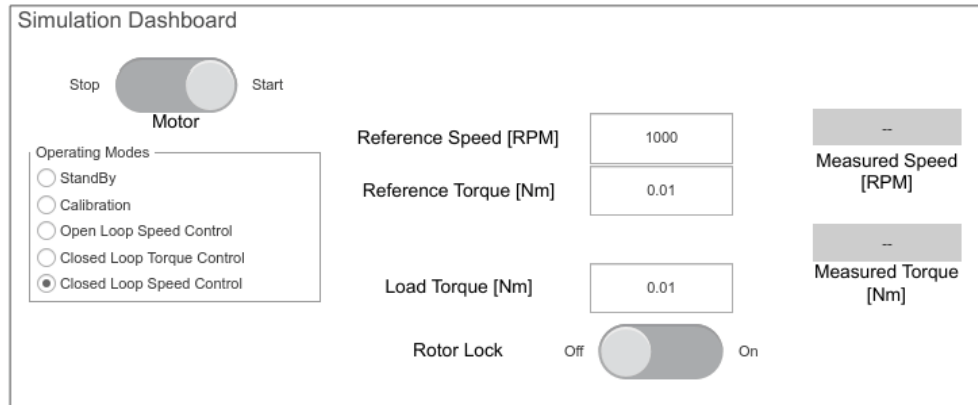
Note:
Reference Speed [RPM] input is active in both Open Loop Speed Control and Closed Loop Speed Control modes.
Reference Torque [Nm] input is active only in Closed Loop Torque Control mode.
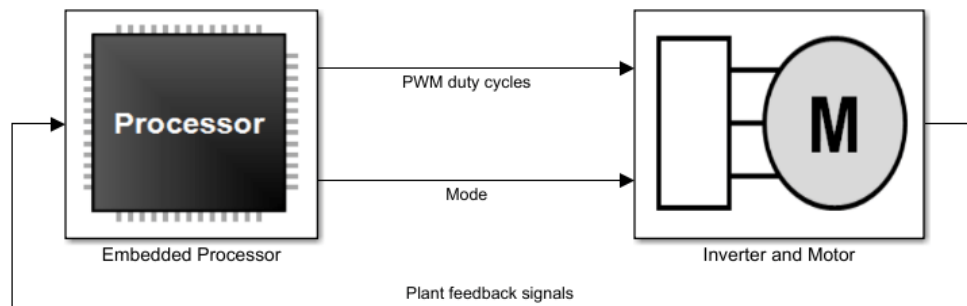
Simulation Dashboard

Stop | Start
Motor

Operating Modes
- StandBy
- Calibration
- Open Loop Speed Control
- Closed Loop Torque Control
- ● Closed Loop Speed Control

Reference Speed [RPM]   1000

Reference Torque [Nm]   0.01

Load Torque [Nm]   0.01

Rotor Lock   Off ——— On

Measured Speed [RPM] --

Measured Torque [Nm] --

Processor — Embedded Processor
PWM duty cycles
Mode
Inverter and Motor — M
Plant feedback signals

Copyright 2020 The MathWorks, Inc.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

- Stateflow®

To generate code and deploy model:

1. Motor Control Blockset™

2. Embedded Coder®

3. C2000™ Microcontroller Blockset

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

5. Stateflow®

Prerequisites

1. Obtain the motor and inverter parameters. The MATLAB project uses default motor and inverter parameters that you can replace with values from either the motor and inverter datasheets or from other sources.

- You can estimate the parameters for the motor that you want to use with the motor control hardware, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate PMSM Parameters Using Recommended Hardware" on page 4-186.

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

2. Update the motor and inverter parameters in the `mcb_qep_data.m` parameter script associated with the MATLAB project. This script automatically opens when you open the MATLAB project. You can also use the Project window to open this script from the `utils` folder.

3. Click **Run** on the **Editor** tab to run the parameter script and update the script parameters in the data dictionary. The data dictionary file (`pmsm_qep_data.sldd`) is available inside the `data` folder in the Project window.

Note: When you simulate the target model or run it on the hardware, if you change any parameter value in the parameter script, you must run the parameter script to update the data dictionary.

Simulate Model

Follow these steps to simulate the target model.

1. Open the target model included in the MATLAB project.

2. Turn the **Stop-Start** slider switch available in the **Simulation Dashboard** area to the **Start** position to allow the model to simulate and run the motor.

During simulation, you can turn the switch to the **Stop** position anytime to immediately stop the motor.

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

Open Loop Speed Control mode

1. Select **Open Loop Speed Control** in the **Simulation Dashboard > Operating Modes** area of the target model.

2. Enter the values in the **Reference Speed [RPM]** and **Load Torque [Nm]** fields.

Note: In the open-loop mode, the motor runs only if the load torque value is either zero or a very small value. If you use a high load torque value, the motor can stop.

Closed Loop Torque Control mode

1. Select **Closed Loop Torque Control** in the **Simulation Dashboard > Operating Modes** area of the target model.

2. Enter the reference torque value in the **Reference Torque [Nm]** field.

3. You can simulate the locked rotor situation by moving the **Rotor Lock** slider switch to **On** position.

When you move the switch to **Off** position, the rotor rotates freely within a maximum speed limit that is defined by the variable `data.pmsm.wLimit_TorqueMode` in the `mcb_qep_data.m` parameter script.

Note: The **Rotor Lock** slider switch is applicable only when performing simulation in this operating mode. It has no effect during the other modes.

Closed Loop Speed Control mode

1. Select **Closed Loop Speed Control** in the **Simulation Dashboard > Operating Modes** area of the target model.

2. Enter the values in the **Reference Speed [RPM]** and **Load Torque [Nm]** fields.

NOTE:

- In the closed-loop mode, the motor runs only if the load torque value is less than or equal to the rated torque of the motor. If you use a higher load torque, the motor starts running in the opposite direction.

- When you simulate the target model, the calibration operating mode produces an invalid simulation output because this mode is designed to calibrate the hardware setup.

When simulating this example, you can observe the measured speed and torque values in the **Measured Torque [Nm]** and **Measured Speed [RPM]** fields in the **Simulation Dashboard** area.

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.

In addition to the target model, the MATLAB project uses a host model. The host model, which is a user interface to the controller hardware board, runs on the host computer. To use the host model, first 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 the selected operating mode, and collect and display the calibration output, and debug signals from the controller.

Required Hardware

The example supports this hardware configuration.

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

You can select one of these inverters by setting the mcb_SetInverterParameters argument in the parameter script file (`mcb_qep_data.m`) to one of these values:

- `BoostXL-DRV8305`

- `BOOSTXL-3PhGaNInv`

For connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

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

2. Complete the hardware connections.

3. 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.

4. To ensure that CPU2 is not 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).

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.

7. Turn the **Stop-Start** slider switch in the **Control Dashboard** area to the **Start** position to allow the model to run the motor.

When running the motor using the target hardware, turn the switch to the **Stop** position anytime to immediately stop the motor.

8. In the host model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a **Port**.

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

Note: Always stop the motor (by using the StandBy mode) before changing the operating mode.

Instructions for Calibration mode

1. Select **StandBy** in the **Control Dashboard > Operating Modes** area of the host model to stop the motor.

2. Select **Calibration** in the **Control Dashboard > Operating Modes** area of the host model.

The controller runs the motor and performs ADC (or current) offset and quadrature encoder offset calibration and updates these offset parameters in the data dictionary file (`pmsm_qep_data.sldd`):

- `pmsm.PositionOffset`

- `inverter.CtSensAOffset`

- `inverter.CtSensBOffset`

The host model also displays the offset values in these fields available in the **Calibration Output** area:

- **Position Offset**

- **Current Sensor Offset A**

- **Current Offset B**

3. Update these offset parameters in the parameter script file (`mcb_qep_data.m`) before you run the parameter script:

- `data.pmsm.PositionOffset`

- `data.inverter.CtSensAOffset`

- `data.inverter.CtSensBOffset`

Note: Update the parameter script immediately to avoid losing these offset values. MATLAB project rewrites the data dictionary (with the existing parameter script values) every time you run the parameter script.

For details about these parameters, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

4. After the calibration completes, the offset parameters are erased if you reset the target hardware. Click **Build, Deploy & Start** on the **Hardware** tab to program the target hardware with the calibrated offset parameters.

Instructions for Open Loop Speed Control mode

1. Select **StandBy** in the **Control Dashboard > Operating Modes** area of the host model to stop the motor.

2. Select **Open Loop Speed Control** in the **Control Dashboard > Operating Modes** area of the host model.

The controller runs the motor in the open-loop control.

3. You can change the default reference speed value by using the **Reference Speed [RPM]** in the **Control Dashboard > Speed Control** area of the host model.

Note:

- Be cautious when providing a reference speed. The motor may not run optimally at all speeds. We recommend that you use a low speed initially and increase it gradually.

- In the open-loop mode, the motor runs only if the load is either zero or negligible. If you use a higher load, the motor stops running.

- You do not need to calibrate the ADC (or current) and quadrature encoder sensor when you run the motor in open-loop control.

Instructions for Closed Loop Speed Control mode

1. Run the motor in the open-loop configuration to validate the hardware setup. Follow the steps described in the Instructions for Open Loop Speed Control mode section.

2. Perform ADC (or current) offset and quadrature encoder offset calibration. Follow the steps described in the Instructions for Calibration mode section.

3. Select **StandBy** in the **Control Dashboard > Operating Modes** area of the host model to stop the motor.

4. Select **Closed Loop Speed Control** in the **Control Dashboard > Operating Modes** area of the host model.

The controller runs the motor in the closed-loop control and controls the rotor speed.

5. You can change the default reference speed value by using the **Reference Speed [RPM]** in the **Control Dashboard > Speed Control** area of the host model.

Note: In the closed-loop mode, the motor runs only if the load torque is less than or equal to the rated load of the motor. If you use a higher load torque, the motor stops running.

Instructions for Closed Loop Torque Control mode

1. Run the motor in the open-loop configuration to validate the hardware setup. Follow the steps described in the Instructions for Open Loop Speed Control mode section.

2. Perform the ADC (or current) offset and quadrature encoder offset calibration if you have not done so earlier. Follow the steps described in the Instructions for Calibration mode section.

3. Select **StandBy** in the **Control Dashboard > Operating Modes** area of the host model to stop the motor.

4. Select **Closed Loop Torque Control** in the **Control Dashboard > Operating Modes** area of the host model.

The controller runs the motor in the closed-loop configuration and controls the torque of the motor.

5. You can change the default reference torque value by using the **Reference Torque [Nm]** in the **Control Dashboard > Speed Control** area of the host model.

You can configure the maximum speed limit of the motor in the closed loop torque control mode using the variable `data.pmsm.wLimit_TorqueMode` in the `mcb_qep_data.m` parameter script.

When running the motor using the target hardware in these operating modes, you can select the debug signals (in the **Scope signals** area) that you want to monitor in the **SelectedSignals** time scope.

Estimate Initial Rotor Position Using Pulsating High-Frequency and Dual-Pulse Methods

This example estimates initial position (in electrical radians) of a stationary interior PMSM by using pulsating high frequency (PHF) injection and dual pulse (DP) techniques.

The example determines the best possible initial estimation for the rotor position using open-loop PHF injection, which it uses further for running closed-loop PHF.

It executes closed-loop PHF by injecting a high frequency signal into the estimated rotor position to determine the actual rotor position, without spinning the motor. This technique works when the motor has a saliency ratio (*Lq/Ld*) greater than 1. Due to a limitation in the PHF method, the estimated position may show ambiguity of π(pi). The dual-pulse (DP) method uses polarity detection to resolve the ambiguity of π and applies π compensation if there is an error. The estimated rotor position ranges from 0 to 2π electrical radians.

The example uses the Pulsating High Freq Observer block to implement the position estimation algorithm. For more details, see Pulsating High Freq Observer.

The example runs only for Stage 1 - initial position estimation (IPE) (as defined in the Pulsating High Freq Observer block page), which includes 3 parts.

Stage 1 focuses on determining the initial position of rotor when the rotor is stationary.

Note: Stage 2 is an extended operation mode where you can use this technique to compute position while the motor runs using closed loop control. The example does not cover Stage 2.

Model

The example includes the target model mcb_ipmsm_pos_est_f28379d.

You can simulate this model or run it on the hardware. Use the open_system command to open the model.
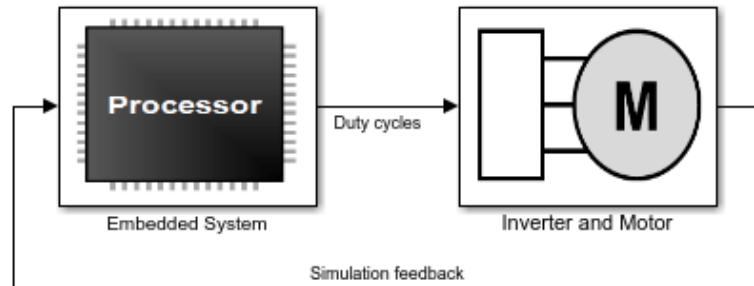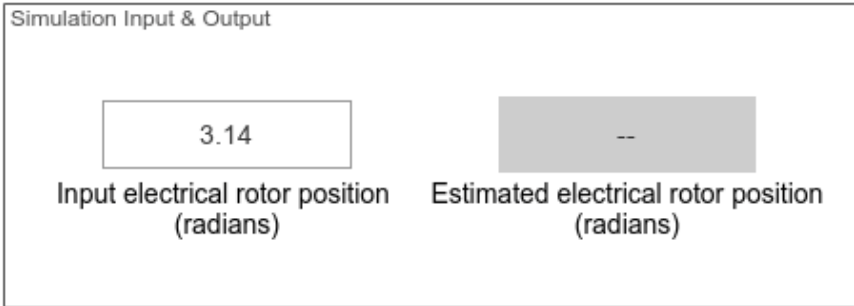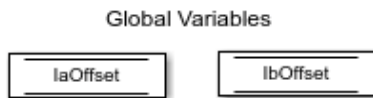
```
open_system('mcb_ipmsm_pos_est_f28379d.slx');
```

Estimation of Initial Rotor Angle (zero speed)
Interior Permanent Magnet Synchronous Motor

HW Prerequisites
1. TI F28379D LaunchPad
2. BOOSTXL-DRV8305 Booster pack
3. IPMSM motor

Steps:
1. Edit motor & inverter parameters
2. Input the rotor electrical angle (radians) in the edit box "Input electrical rotor position"
3. Simulate the model and observe the estimated rotor angle in display box "Estimated electrical rotor position"
4. Click **Build, Deploy & Start** in the Hardware tab.
5. Start or stop algorithm via host model
6. Learn more about this example

Global Variables

| IaOffset | | IbOffset |

Simulation Input & Output

| 3.14 | | -- |

Input electrical rotor position (radians) | Estimated electrical rotor position (radians)

Processor — Embedded System

Duty cycles → Inverter and Motor (M)

Simulation feedback

Copyright 2021-2023 The MathWorks,

Required MathWorks® Products

- Motor Control Blockset™

- Embedded Coder®

- C2000™ Microcontroller Blockset

- Simscape™ Electrical™ (only for simulation)

Prerequisites

1. Determine the motor parameters. The target model uses default parameters that you can replace with values from either the motor datasheet or other sources.

However, if you have motor control hardware, you can estimate the parameters for your motor by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate PMSM Parameters Using Recommended Hardware" on page 4-186. The parameter estimation tool updates the `motorParam` variable (in the MATLAB® workspace) with the estimated motor parameters.

2. Update the motor and inverter parameters in the model initialization script associated with the target model. For instructions, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

Simulate Model

Follow the instructions in the Simulate Nonlinear Stator Core Behavior section to introduce high saliency in the motor block. Then follow these steps to simulate the target model.

1. Open the target model included in this example.

2. Enter the initial rotor position value (that the Interior PMSM block should use for simulation) in the **Input electrical rotor position (radians)** field available in the **Simulation Input & Output** area of the target model.

3. Click **Run** on the **Simulation** tab to simulate the target model. The model estimates the rotor position using the PHF method. Then it uses the DP method to determine if the estimated rotor position needs π compensation. The target model applies π compensation if needed.

It then displays the estimated rotor position in the **Estimated electrical rotor position (radians)** field.

4. Observe the logged signals in the Simulation Data Inspector.

This plot shows the current along the d-axis (I_d) during simulation.
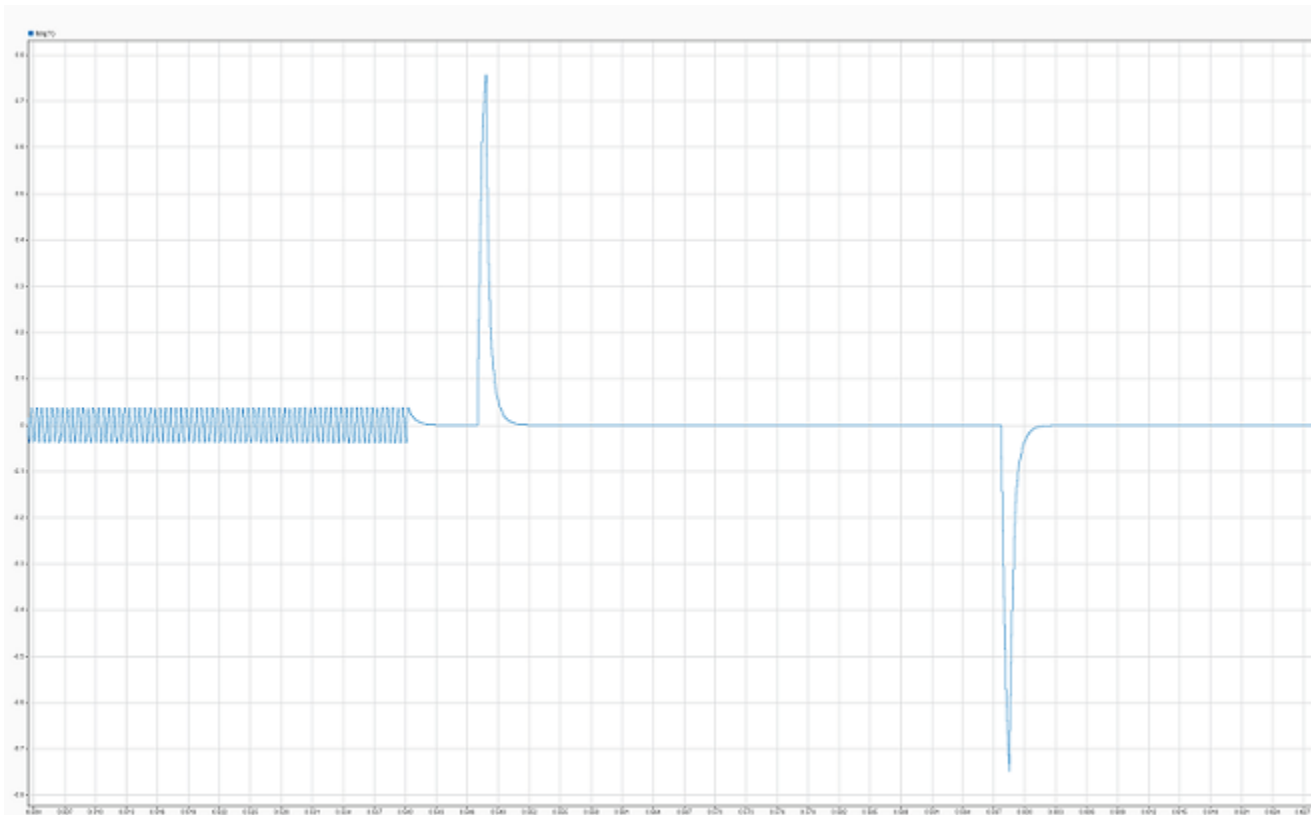
This plot shows the estimated rotor position when the target model does not apply π compensation $(\theta_{init} = \theta_{est})$ and when the target model applies π compensation $(\theta_{init} = \theta_{est} + \pi)$.

4-173
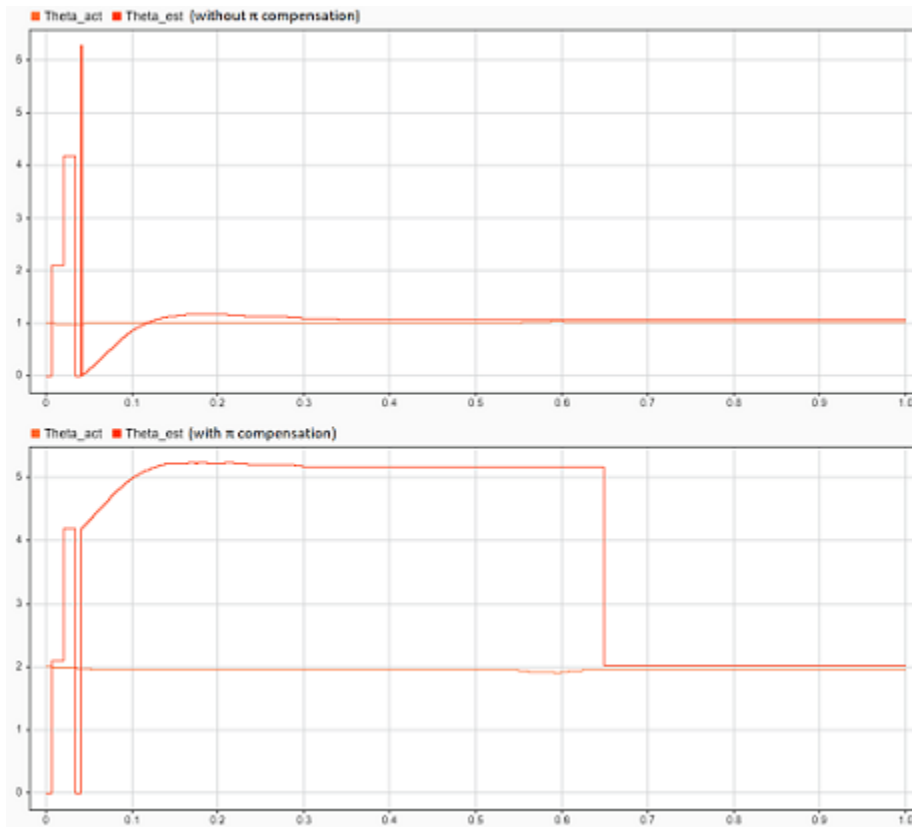
5. The **Estimated Rotor Position (Electrical Radians)** display block in the **Output** area of the target model shows the initial rotor position estimated by the target model.

Generate Code and Deploy Model to Target Hardware

Follow the instructions in this section to generate code and run the example 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. Before you can run the host model on the host computer, 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

The example supports this hardware configuration. You can also use the target model name to open the model from the MATLAB® command prompt.

LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter + iPMSM (such as `Adlee` BM-180 motor): mcb_ipmsm_pos_est_f28379d

For connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

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

2. Complete the hardware connections.

3. By default, the model computes the ADC offset values for phase current measurement. To disable this functionality, update the value of the `inverter.ADCOffsetCalibEnable` variable in the model initialization script to `0`.

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 4-10.

4. 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.

5. Load a sample program to CPU2 of the LAUNCHXL-F28379D board. For example, load the program that operates the CPU2 blue LED by using GPIO31 (`c28379D_cpu2_blink.slx`). This ensures that CPU2 is not mistakenly configured to use the board peripherals intended for CPU1.
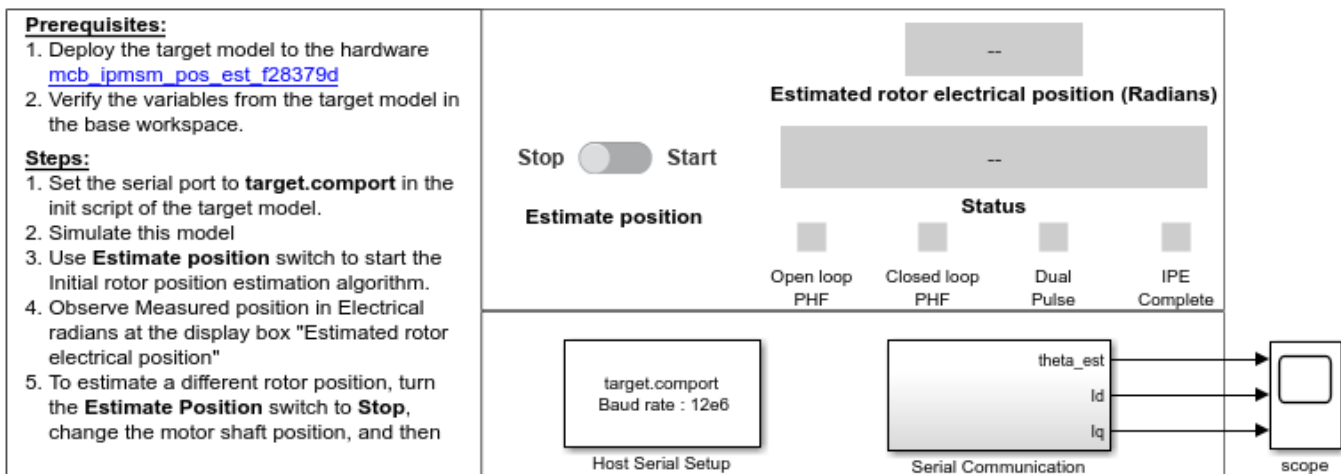
6. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model to the hardware. Verify the variables that the target model adds to the workspace.

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_ipmsm_pos_est_f28379d_host_model.slx');
```

Estimation of Initial Rotor Angle (zero speed) Control Host
Interior Permanent Magnet Synchronous Motor

Initial Position Estimation (IPE)

Prerequisites:
1. Deploy the target model to the hardware
   mcb_ipmsm_pos_est_f28379d
2. Verify the variables from the target model in the base workspace.

Steps:
1. Set the serial port to **target.comport** in the init script of the target model.
2. Simulate this model
3. Use **Estimate position** switch to start the Initial rotor position estimation algorithm.
4. Observe Measured position in Electrical radians at the display box "Estimated rotor electrical position"
5. To estimate a different rotor position, turn the **Estimate Position** switch to **Stop**, change the motor shaft position, and then

Estimated rotor electrical position (Radians)

Stop   Start

Estimate position

Status

| Open loop PHF | Closed loop PHF | Dual Pulse | IPE Complete |

target.comport
Baud rate : 12e6

Host Serial Setup

theta_est
Id
Iq

Serial Communication

scope

Copyright 2021-2022 The MathWorks,

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

8. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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

10 Change the position of the **Estimate Position** switch to **Start** to start running the example algorithm. The target model estimates the rotor position using PHF method. Then it uses the DP method to determine if the estimated rotor position needs π compensation. The target model applies π compensation if needed.

The host model obtains the estimated rotor position from the target hardware and displays it in the **Estimated rotor electrical position (radians)** field.

11 To estimate a different rotor position, change the position of the **Estimate Position** switch to **Stop**. The **Estimated rotor electrical position (radians)** field displays the value 0.

Turn the motor shaft to a new position. Then change the position of the **Estimate Position** switch to **Start** to run the example algorithm again and estimate the new position.

12 Observe the estimated position, Id, and Iq signals in the time scope available in the host model.

This plot shows the estimated rotor electrical position (θ_{init}) and the stator currents (Id) and (Iq).
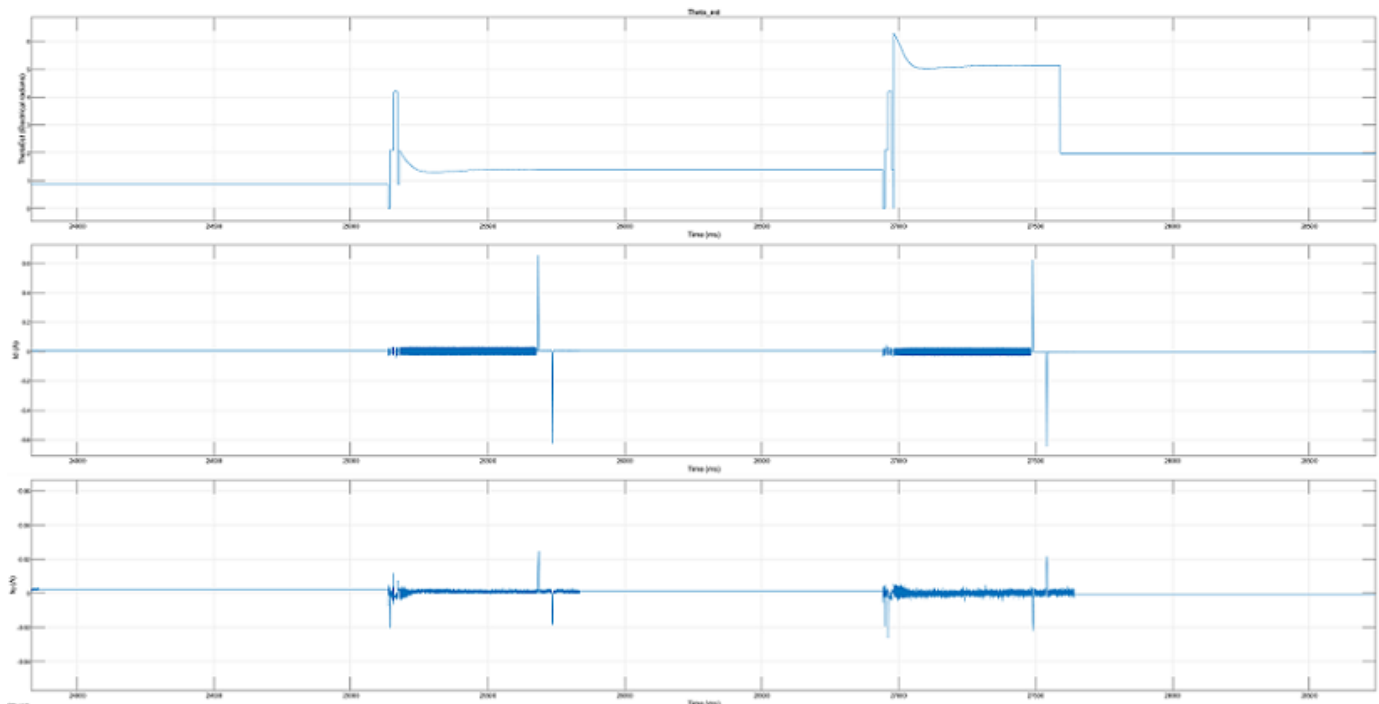
13 The **Status** field displays the following statuses:

- "IPE not started" — Stage 1 operation not yet started.

- "IPE running" — Stage 1 operation is presently running.

- "IPE completed" — Stage 1 operation is complete.

- "IPE failed" — Stage 1 operation failed due to high error in estimated position.

Simulate Nonlinear Stator Core Behavior

The DP method algorithm works only when the stator core saturates and shows a nonlinear behavior. To generate this behavior in simulation, the target model uses the PMSM (Simscape Electrical) block from Simscape™ Electrical™. Follow this procedure to experimentally determine the tabulated L_d, L_q, and I_d data for the actual motor and use this data to make the motor block nonlinear for simulation purposes.

Determine L_d Data

1. Create a target model to run a PMSM using open-loop control. Modify the algorithm so that it injects the voltages $V_d = V_{ac}\sin(\omega t) + V_{dc}$ and $V_q = 0$ into the actual motor. This generates the current $I_d = I_{ac}\sin(\omega t + \phi) + I_{dc}$ in the motor where:

V_d is the voltage that the algorithm computes and injects along the rotor's d-axis.

ω is the frequency at which we can measure the phase difference. This frequency should be approximately 10 times lower than the PWM switching frequency but should be high enough to keep the rotor stationary (a frequency value that is too low can spin the rotor).

$V_{ac}\sin(\omega t)$ is the AC component of V_d (such that V_{ac} and ω are constants that the algorithm should assume).

V_{dc} is the DC component of V_d that you provide as an input.

I_d is the current along the d-axis of the rotor that the algorithm computes from the measured I_a and I_b phase currents.

$I_{ac}\sin(\omega t + \phi)$ is the AC component of I_d (ϕ is the phase difference between V_d and I_d that the algorithm should measure).

I_{dc} is the DC component of I_d, which is fixed for a V_{dc} input (the algorithm should compute this value by using V_{dc} and the motor resistance).

Note: Because $V_q = I_q = 0$, the rotor remains stationary.

a. Add an algorithm to read the I_a and I_b phase currents that the motor draws.

b. Add an algorithm to convert the I_a and I_b phase current values into the equivalent I_d value by using Clarke and Park transforms.

c. Add an algorithm to compute ϕ and I_{dc}.

d. Add an algorithm to compute I_{ac}.

$$I_{ac} = \left(\frac{I_d - I_{dc}}{\sin(\omega t + \phi)} \right)$$

e. Add an algorithm to compute L_d.

$$L_d = \frac{\sqrt{V_{ac}^2 - I_{ac}^2 R^2}}{\omega I_{ac}}$$ (for a fixed I_{dc})

For details about designing a control algorithm and configuring it for deployment to the target hardware, see "Control Algorithm Design" and "Deployment and Validation".

2. Create a model initialization script (.m file) to initialize the paramters and perform other calculations. Use the **Model Properties > Callbacks** to integrate this script with the target model. For details about how Motor Control Blockset™ uses this script, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

3. Configure the target model to run on the target hardware. For instructions, see "Model Configuration Parameters" on page 2-2.

4. Create a separate host model to control and communicate with the target model running on the motor-control hardware in real-time. You can add an **Edit** box on the host model and use it to send V_{dc} input to the target hardware. Use **Display** boxes to obtain the computed L_d and I_{dc} values from the target hardware.

For details about the host-target communication interface, see "Host-Target Communication" on page 6-2.

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

6. Simulate the host model. Send a V_{dc} input to the target hardware and record the L_d and I_{dc} values that you obtain.

7. Change V_{dc} manually across a range between a negative and positive voltage and record the resulting L_d and I_{dc} values for each operating point.
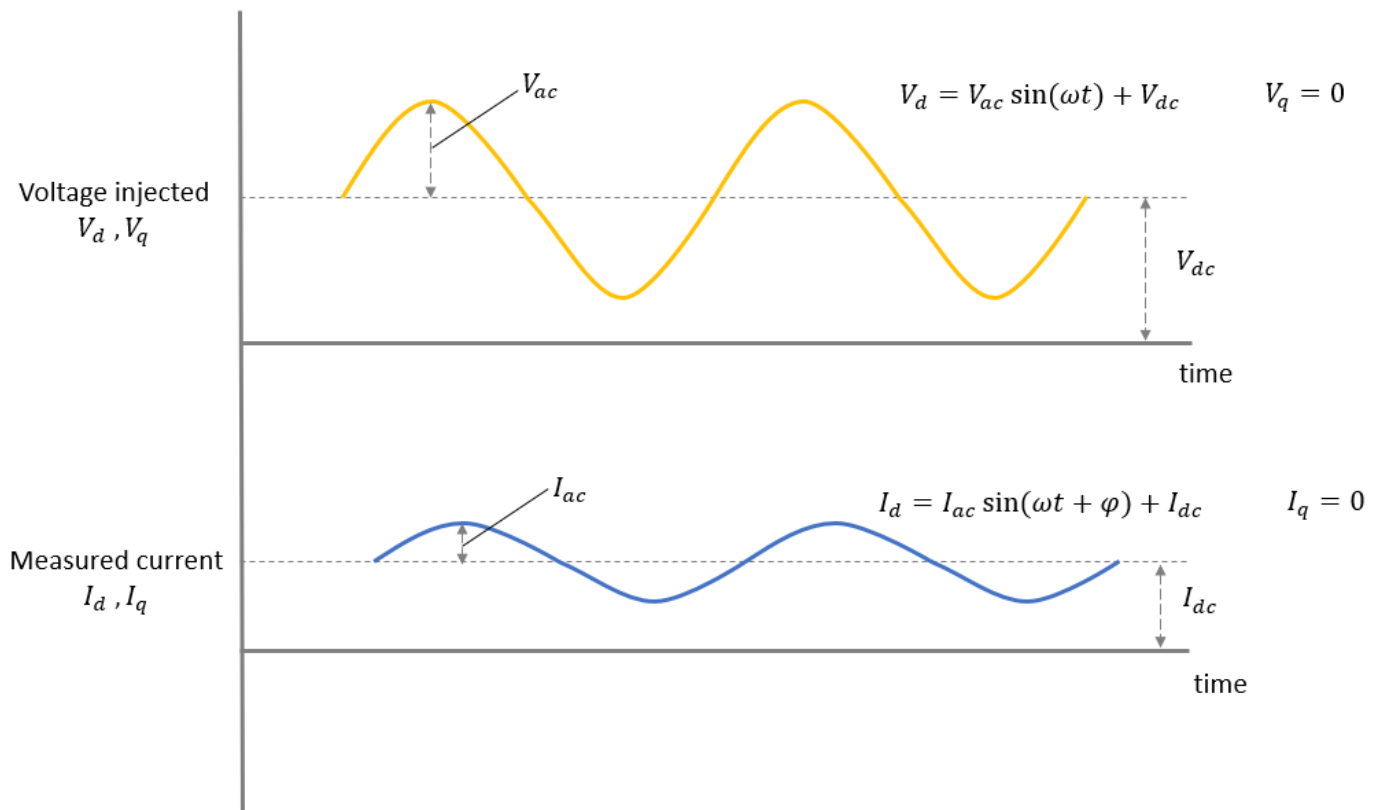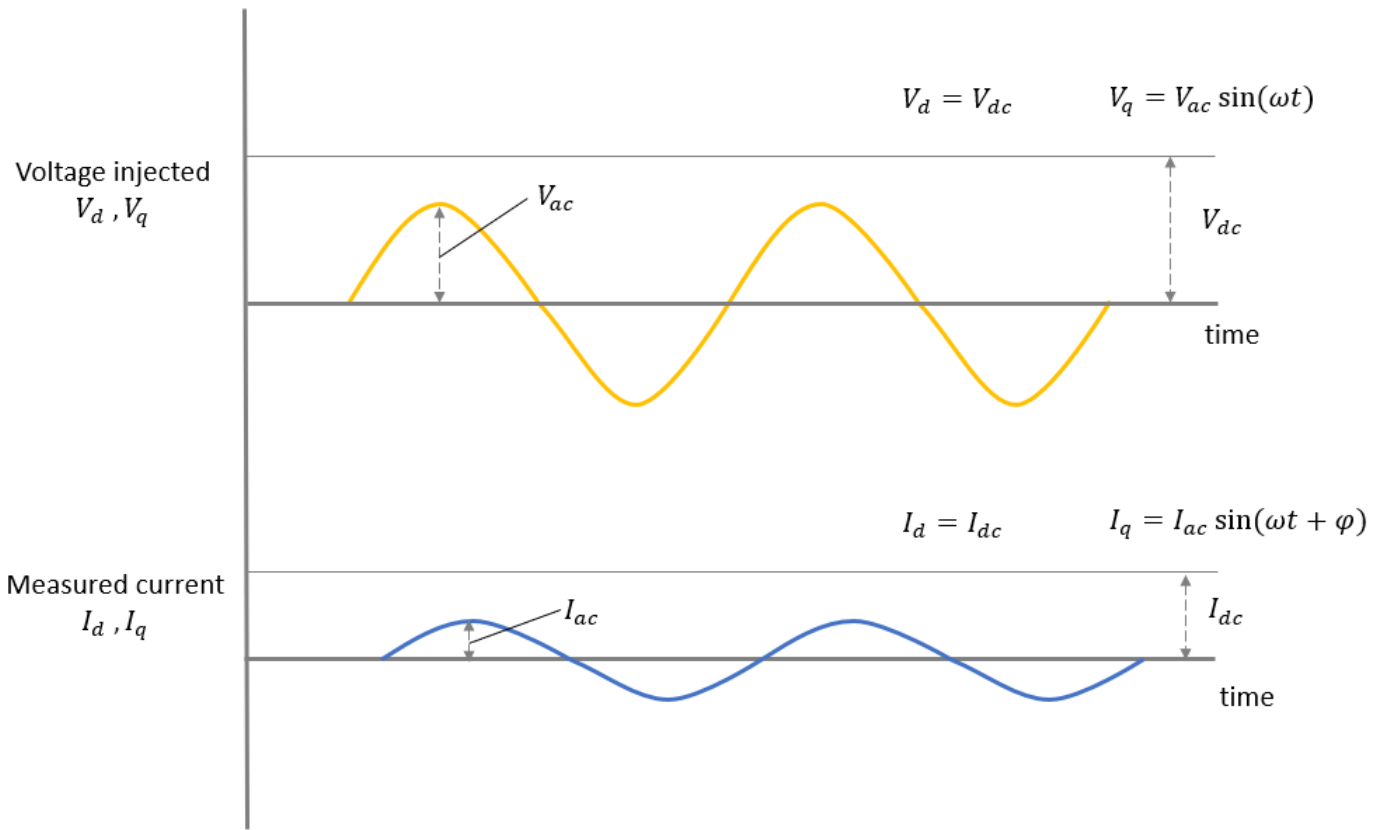
Determine L_q Value

Perform these steps to determine a L_q value that is higher than the sequence of L_d values obtained from the Determine Ld section. This is necessary to maintain a high saliency ratio ($L_q > L_d$) for the simulated motor block.

1. Modify the target model that you used in the Determine Ld section so that it injects the voltages $V_d = V_{dc}$ and $V_q = V_{ac} \sin(\omega t)$ into the actual motor. This generates a current $I_q = I_{ac} \sin(\omega t + \phi)$ in the motor where:

$V_d = V_{dc}$ is the constant DC voltage that the algorithm injects along the d axis of the rotor. V_{dc} should be high enough to hold the rotor at zero position and keep it stationary.

$V_q = V_{ac} \sin(\omega t)$ is the AC voltage that the algorithm computes and injects along the q-axis of the rotor. (such that V_{ac} and ω are constants that the algorithm assumed in the Determine Ld section)

$I_q = I_{ac} \sin(\omega t + \phi)$ is the AC current along the q-axis of the rotor that the algorithm computes from the measured I_a and I_b phase currents (ɸ is the phase difference between V_q and I_q that the algorithm should measure)

4-179

$$V_d = V_{dc} \qquad V_q = V_{ac}\sin(\omega t)$$

Voltage injected V_d, V_q

V_{ac}

V_{dc}

time

$$I_d = I_{dc} \qquad I_q = I_{ac}\sin(\omega t + \varphi)$$

Measured current I_d, I_q

I_{ac}

I_{dc}

time

a. Update the existing algorithm to convert the I_a and I_b phase current values into the equivalent I_q value by using Clarke and Park transforms.

b. Update the existing algorithm to compute ϕ.

c. Update the existing algorithm to compute I_{ac}.

$$I_{ac} = \left(\frac{I_q}{\sin(\omega t + \phi)} \right)$$

d. Add an algorithm to compute L_q.

$$L_q = \frac{\sqrt{V_{ac}^2 - I_{ac}^2 R^2}}{\omega I_{ac}}$$

2. Update the model initialization script to perform the preceding modified calculations.

3. Update the host model to obtain and display the L_q value computed by the target hardware.

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

5. Simulate the host model. Send a V_{dc} input to the target hardware and record the L_q value that you obtain.

Design High Saliency Motor Block

Sections Determine Ld Data and Determine Lq Value provide the following data:

- Sequence of L_d values

- Sequence of I_{dc} (I_d for a DC current) values

- Constant L_q value

In addition, we can assume a set of I_q values that we can use to add high saliency to the motor block.

Use the variables `pmsm.nonlin.idVec`, `pmsm.nonlin.Ld_data`, `pmsm.nonlin.iqVec`, and `pmsm.nonlin.Lq_data` to store this data in the model initialization script `mcb_ipmsm_pos_est_f28379d_data.m` associated with the example target model `mcb_ipmsm_pos_est_f28379d.slx`.

Note: This script is different from the model initialization script used in sections Determine Ld Data and Determine Lq Value.

In addition, define the variables `pmsm.nonlin.LdMatrix`, `pmsm.nonlin.LqMatrix`, and `pmsm.nonlin.PmMatrix`.

```
% Ld vs Id data computed through experiment mention in documentation
% Stator current in d-axis
pmsm.nonlin.idVec    = [-16.385 -12.69 -10.105 -6.305 -3.755 -1.475 1.57 3.925 6.505 10.44 13.04 14.3325]; %A
% Stator d-axis inductance
pmsm.nonlin.Ld_data  = [1.77E-04 1.76E-04 1.76E-04 1.75E-04 1.76E-04 2.63E-04 2.59E-04 1.76E-04 1.73E-04 1.72E-04 1.70E-04  1.69E-04]; %H

% Stator current in q-axis
pmsm.nonlin.iqVec    = linspace(-13,13,20);    %A
% Q-axis inductance value at low currents
pmsm.nonlin.Lq_data  = 2.88E-04;               %H

% Here Ld has been considered to be dependent only Id
pmsm.nonlin.LdMatrix = pmsm.nonlin.Ld_data'*ones(1,20);  %H

% Here Lq has been considered to be independent of Id, Iq
% Here low current Lq value has been chosen because Ld<Lq has to be
% satisfied for all Id, Iq
pmsm.nonlin.LqMatrix = pmsm.nonlin.Lq_data*ones(12,20);  %H

% Here PM flux has been considered to be independent of Id, Iq
pmsm.nonlin.PmMatrix = pmsm.FluxPM * ones(12,20);
```

In the PMSM (Simscape Electrical) block parameter dialog box, set the **Modeling fidelity** parameter to **Tabulated Ld, Lq and PM** and add these variables as shown in the following figure.

The variables `pmsm.nonlin.idVec`, `pmsm.nonlin.Ld_data`, `pmsm.nonlin.iqVec`, and `pmsm.nonlin.Lq_data` help saturate the stator core and introduce a nonlinear behavior during simulation. This enables the algorithm of the DP method to generate the I_d current impulses (corresponding to Pulse-1 and Pulse-2) with different peaks.

References

[1] W. Zine, L. Idkhajine, E. Monmasson, Z. Makni, P. Chauvenet, B. Condamin, and A. Bruyere, "Optimisation of HF signal injection parameters for EV applications based on sensorless IPMSM drives", IET Electric Power Applications, Volume 12, Issue 3, March 2018, p. 347 - 356 (doi:10.1049/iet-epa.2017.0228).

[2] Gaolin Wang, Guoqiang Zhang, and Dianguo Xu, "Position Sensorless Control Techniques for Permanent Magnet Synchronous Machine Drives", Springer, Singapore, 2020 p. 41 - 43 (doi: https://doi.org/10.1007/978-981-15-0050-3).

Algorithm-Export Workflows for Custom Hardware

This example enables you to use any custom motor-control hardware (hardware not used in the Motor Control Blockset™ examples) to run a three-phase permanent magnet synchronous motor (PMSM) using field-oriented control (FOC). Using the algorithm export workflows, which involve generating code for the control algorithm by using Simulink® and Embedded Coder® and then integrating it with either manually written or externally generated hardware driver code. This example explains the algorithm export workflows along with the intermediate steps.

The example uses the following hardware as a reference, but you can use any motor-control hardware:

• Controller: STMicroelectronics® STM32F302R8

• Inverter: STMicroelectronics® X-NUCLEO-IHM07M1

• Motor: BLY171D (includes quadrature encoder sensor)

You can use this example to customize the control algorithm and integrate it with the drivers for your motor-control hardware. In this example, we use the STM32 Cube MX software to configure and generate code for the hardware drivers. This example supports any three-phase PMSM.

Implementing the FOC algorithm needs real-time rotor position feedback. This example uses a quadrature encoder sensor to measure the rotor position. For details about FOC, see "Field-Oriented Control (FOC)" on page 4-3.

The example includes three workflows.

1. Open-loop control and ADC offset calibration — This workflow uses an algorithm that runs a PMSM using open-loop control (also known as scalar control or Volts/Hz control). You can use this workflow to check the integrity of the hardware connections and calculate the ADC offsets for the current sensors available on the hardware.

2. Quadrature encoder offset calibration — This workflow uses an algorithm that calculates the offset between the d-axis of the rotor and the index pulse position as detected by the quadrature encoder sensor. The control algorithm (available in the field-oriented control workflow) needs this offset to accurately compute the rotor position, which is necessary to implement FOC.

3. Field-oriented control — This workflow uses an algorithm that runs a PMSM using closed-loop field-oriented control (FOC). The workflow uses the ADC and quadrature sensor offsets as inputs.

Each workflow includes these steps to prepare, deploy, and run the algorithm on your hardware:

1. Generate code for the control algorithm using Embedded Coder®

2. Obtain C Code For Hardware Drivers

3. Integrate control algorithm code with the driver code

4. Deploy the integrated code to hardware

5. Control the motor using a host Simulink® model.

Open MATLAB Project

Use one of these methods to open the MATLAB® project:

- Click **Open Example**.

- Run the command `mcb_FOCAlgorithmExportDemoStart` at the command prompt.

The project contains three folders, one for each workflow required to run the final FOC algorithm. Each folder contains these contents:

- Data script containing motor, inverter, and target hardware details.

- Algorithm model for generating code for the control algorithm. The generated code will be available in the folder `[project_root]/work/code`.

- Host model for communication with the target hardware.

- C code which shows how to integrate the generated algorithm code and the hardware driver code (specific to STM32F302R8 & X-NUCLEO-IHM07M1).

In addition to the three folders, the project also includes an `.IOC` file. You can use this file with STM32 Cube MX to configure the peripherals of the target and generate code. The `.IOC` file available in the project is specific to the STM32F302R8 and X-NUCLEO-IHM07M1 hardware.

Workflows for Custom Hardware

Follow these workflows in sequence.

1. "Open-Loop Control and ADC Offset Calibration" on page 8-2

2. "Quadrature Encoder Offset Calibration" on page 8-11

3. "Field-Oriented Control" on page 8-18

Estimate PMSM Parameters Using Recommended Hardware

This example determines the parameters of a permanent magnet synchronous motor (PMSM) using the recommended Texas Instruments™ hardware. The tool determines these parameters:

- Phase resistance, R_s (Ohm)

- d and q axis inductances, L_d and L_q (Henry)

- Back-EMF constant, K_e (Vpk_LL/krpm, where Vpk_LL is the peak voltage line-to-line measurement)

- Motor inertia, J (Kg.m^2)

- Friction constant, B (N.m.s)

The example accepts the minimum required inputs, runs tests on the target hardware, and displays the estimated parameters.

NOTE: This example does not support simulation. Use one of the supported hardware configurations to run this example.

Prerequisites

The parameter estimation tool needs the motor position as detected by either a quadrature encoder, a Hall sensor, or a sensorless flux observer. To detect the motor position correctly by using a position sensor, calibrate the quadrature encoder or Hall sensor attached to 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 4-69.

If you are using a quadrature encoder sensor:

- Ensure that the PMSM has a quadrature encoder sensor.

- Calibrate the quadrature encoder offset. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79.

NOTE: If you set the **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 a Hall or a 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 a Hall or a quadrature encoder sensor

- DC power supply

Required MathWorks® Products

To run parameter estimation, you need these products:

- Motor Control Blockset™

- Fixed-Point Designer™

- Embedded Coder®

- C2000™ Microcontroller Blockset

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 a Hall sensor, connect the Hall sensor encoder output to J10 on the inverter board.

5. If you are using a 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 a Hall sensor, connect the Hall sensor output to QEP_B (configured as eCAP) on LAUNCHXL.

5. If you are using a 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 details about the hardware setup and 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 the required tests and displays the estimated parameters.

Prepare Workspace

Open the parameter estimation host model. You can also use this command to open the host model:

```
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** — In the block parameter dialog boxes of Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks, select a serial **Port** to which the hardware is connected. Select an available port from the list. For more details, see "Find Communication Port" on page 6-4.

- **Required Inputs** — Enter the motor specification and hardware setup data. You can obtain these values either from the motor datasheet or from 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 4-69, "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79, and "Per-Unit System" on page 6-20.

  - **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 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 25000 RPM.

- 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.

Update Advanced Parameters

You can optionally update the advanced parameters related to the parameter estimation algorithm. Click the link available in the host model to access and update these parameters:

Steps
1. Provide required inputs.
2. Press **Ctrl+D** to update the workspace
3. **Build, Deploy & Start** required target models
4. **Run** this model to estimate motor parameters
5. To modify the parameters of the estimation algorithm click here

- **Openloop Vd reference Voltage for Rs estimation** — Enter the reference voltage V_d, in per-unit (PU)(Volts), for open-loop configuration that the algorithm uses to estimate phase resistance, R_s. The V_d value should be high enough to bring the rotor to zero position and hold it there. This value should also be high enough to generate readable current feedback from the motor. The V_d value should be low enough to avoid rapid increase in motor temperature. This parameter uses the default value of `0.1` PU.

- **Rs estimation time** — Enter the time (in seconds) that the algorithm should take to estimate the phase resistance, R_s. This time should be high enough for the algorithm to obtain sufficient samples for average value computations (for a particular switching frequency). If the measured voltage and current debug signals contain noise, increase this parameter value so that the algorithm captures more samples for the average value computations. This parameter uses the default value of 2 seconds. The maximum Rs estimation time allowed is 9 seconds.

- **Frequency Sweep Range for Ld and Lq estimation** — Enter the frequency sweep range (in Hertz) that the algorithm uses for measuring the inductances L_d and L_q. The lower and upper frequency limits of this range should be high enough to make the rotor motionless. At the same time, these frequencies should be low enough for the algorithm to obtain sufficient samples (for a particular switching frequency). Very high frequency values result in a higher inductive reactance that can lead to inaccurate current measurements. This parameter uses the default frequency values of `400` and `1000` Hertz.

- **Frequency step size for Ld and Lq estimation** — Enter the step size (in Hertz) for the frequency sweep that the algorithm uses for measuring the inductances L_d and L_q. The parameter uses the default value of `10` Hertz.

- **DC bias for Vd during Ld & Lq estimation** — Enter the DC bias voltage, in PU (Volts), for the V_d and V_q voltage perturbations that the algorithm uses for measuring the inductances L_d and L_q. The DC bias voltage should be high enough to lock the rotor shaft. At the same time, it should be low enough for the algorithm to avoid overcurrents at the time of application of sinusoidal voltage perturbations. The parameter uses the default value of `0.1` PU.

- **Amplitude for Vd & Vq during Ld & Lq estimation** — Enter the amplitude, peak-to-peak value in PU (Volts), of the V_d and V_q voltage perturbations that the algorithm uses for measuring the inductances L_d and L_q. This amplitude should be high enough to avoid introducing noise during ADC measurements. At the same time, it should be low enough for the algorithm to avoid overcurrents at the time of application of sinusoidal voltage perturbations. The parameter uses the default value of `0.05` PU.

- **Iq reference for torque control** — Enter the reference I_q current, in PU (Amperes), for the closed-loop torque control tests performed by the algorithm. This current should be low enough to avoid sudden jolts to the rotor shaft. At the same time, it should be high enough for the algorithm to overcome the inertia of the rotor shaft. The parameter uses the default value of `0.2` PU.

- **Under Voltage limit** — Enter the voltage limit (as percentage of the input DC voltage) for undervoltage protection that the algorithm provides to the motor. The parameter uses `80` as the default value.

- **Over Current limit** — Enter the current limit (as percentage of PMSM nominal current) for overcurrent protection that the algorithm provides to the motor. This value should be high enough for the algorithm to successfully run parameter estimation tests under normal conditions using the configured parameters. At the same time, this value should not exceed `100`. The parameter uses `100` as the default value.

- **End speed for Inertia estimation** — Enter the motor speed, in PU (RPM), used by the algorithm to calculate the motor inertia. The parameter uses the default value of `0.25`.

Deploy Target Models

Before starting the tests by using the parameter estimation tool, you need to 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 values 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 F28069M-based controller that uses the sensorless Flux Observer block: mcb_param_est_sensorless_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.

4-191
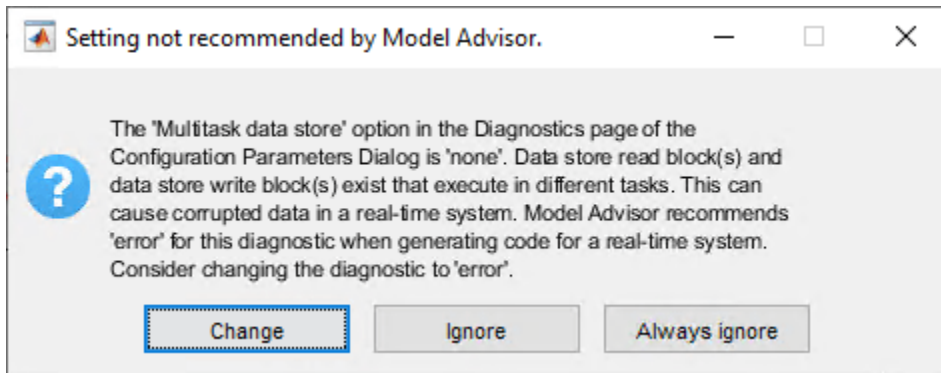
Workflow 2: Manually Download Target Model

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_sensorless_f28069_DRV8312.out

    - < matlabroot >\toolbox\mcb\mcbexamples\mcb_param_est_f28379D_DRV8305.out

    - < matlabroot >\toolbox\mcb\mcbexamples \mcb_param_est_sensorless_f28379D_DRV8305.out

NOTE: The files mcb_param_est_f28069_DRV8312.out and mcb_param_est_f28379D_DRV8305.out use a fixed quadrature encoder slits count of 1000. Therefore, when you set the required input **Sensor Selection** to **QEP** in the host model, you can use these files only for motors connected to a quadrature encoder sensor with 1000 slits (for example, the Teknic M-2310P motor).

- 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.

Then update the required details in the host model. See the section Prepare Workspace for information about the required inputs.

2. In the host model, check if the **Run-Stop** slider switch position is **Run**. Then, 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.

During an emergency, you can manually turn the **Run-Stop** slider switch to **Stop** position to stop the parameter estimation tests.

4. The host model displays the estimated motor parameters after successfully completing the tests.

When computing the electrical parameters (phase resistance R_s, d axis inductance L_d, and q axis inductance L_q), the parameter estimation tool keeps the motor stationary. However, the parameter estimation tool runs the motor to compute the mechanical parameters (back-EMF constant K_e, motor inertia J, and friction constant B).

Note: You can ignore the displayed mechanical parameters, if the motor does not run.

- When the parameter estimation tests successfully complete, the **Test Status** LED turns green.

- If the parameter estimation tests are interrupted, the **Test Status** LED turns red. The model also interrupts the tests and turns these LEDs red to protect the hardware from the following faults:

     - 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)

     - 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)

     - Serial communication fault

5. When the **Test Status** LED turns red, run the host model again to rerun the parameter estimation tests.

If the **Test Status** LED is green, check the **Ld** and **Lq** LEDs available in the **Parameter Validity** section of the host model. These LEDs indicate the following statuses:

- Green — Indicates that the computed **Ld** and **Lq** values are valid.

- Amber — Indicates that the computed **Ld** and **Lq** values are invalid. Run the host model again to rerun the parameter estimation tests.

6. Use the **Signal from Target** field on the host model to select a debug signal that you want to monitor. After selecting a signal, open the **SelectedSignal** time scope (available in the **Signal Conditioning, Scaling and Advanced Algorithm Parameters** subsystem) to view the selected debug signal.

The parameter estimation tool uses the following algorithm to estimate parameters:

- Phase resistance, R_s — The tool uses Ohm's law to estimate this value.

- d axis inductance, L_d — The tool uses frequency injection method to estimate these values.

- q axis inductance, L_q — The tool uses frequency injection method to estimate these values.

- Back-EMF constant, K_e — The tool measures the currents and voltages and uses the electric motor equation to estimate this value.

- Motor inertia, J — The tool estimates this value by using retardation test.
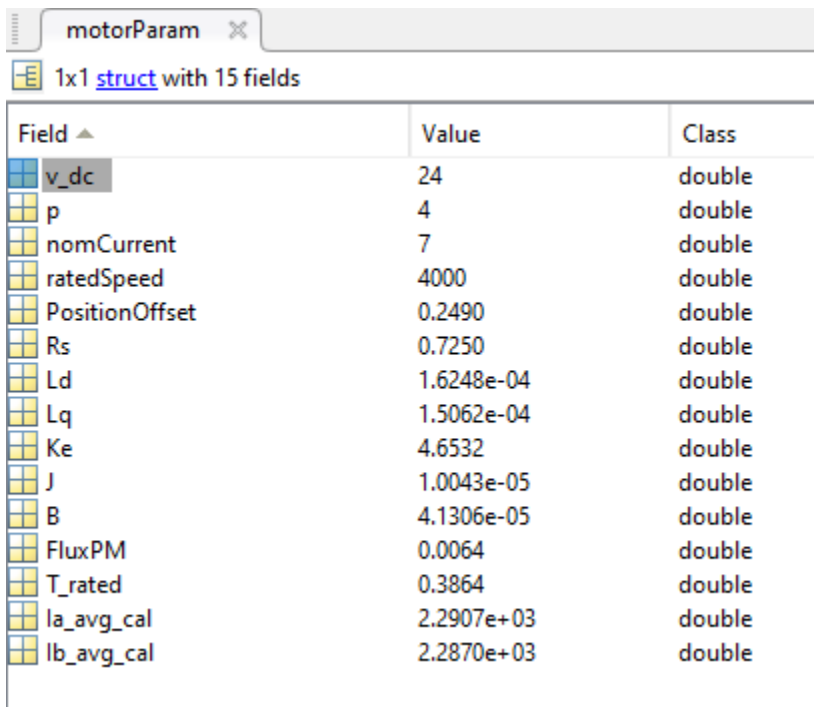
- Friction constant, B — The tool estimates this value by using the torque equation for a motor running at a constant speed.

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.

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
Ia_avg_cal	2.2907e+03	double
Ib_avg_cal	2.2870e+03	double

motorParam — 1x1 struct with 15 fields

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.

Field-Oriented Control of PMSM Using Reinforcement Learning

This example shows you how to use the control design method of reinforcement learning to implement field-oriented control (FOC) of a permanent magnet synchronous motor (PMSM). The example uses FOC principles. However, it uses the reinforcement learning (RL) agent instead of the PI controllers. For more details about FOC, see "Field-Oriented Control (FOC)" on page 4-3.

This figure shows the FOC architecture with the reinforcement learning agent. For more details about the reinforcement learning agents, see "Reinforcement Learning Agents" (Reinforcement Learning Toolbox).
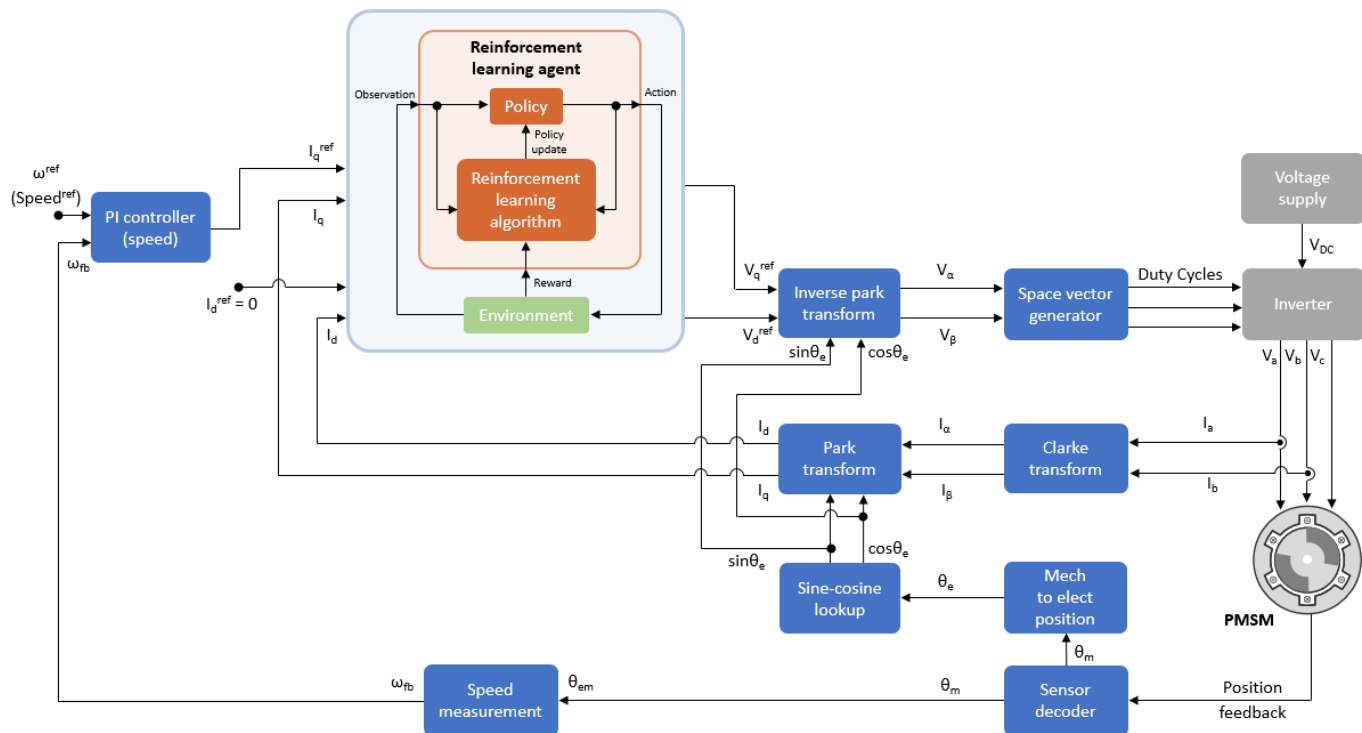
The reinforcement learning agent regulates the d-axis and q-axis currents and generates the corresponding stator voltages that drive the motor at the required speed.

The speed-tracking performance of an FOC algorithm that uses a reinforcement learning agent is similar to that of a PI-controller-based FOC.

Model

The example includes the mcb_pmsm_foc_sim_RL model.

Note: You can use this model only for simulation.

This model includes the FOC architetcure that uses the reinforcement learning agent. You can use the open_system command to open the Simulink® model.

```
mdl = 'mcb_pmsm_foc_sim_RL';
open_system(mdl);
```

Permanent Magnet Synchronous Motor - Reinforcement Learning

Note: This example demonstrates PMSM speed control with Reinforcement learning control
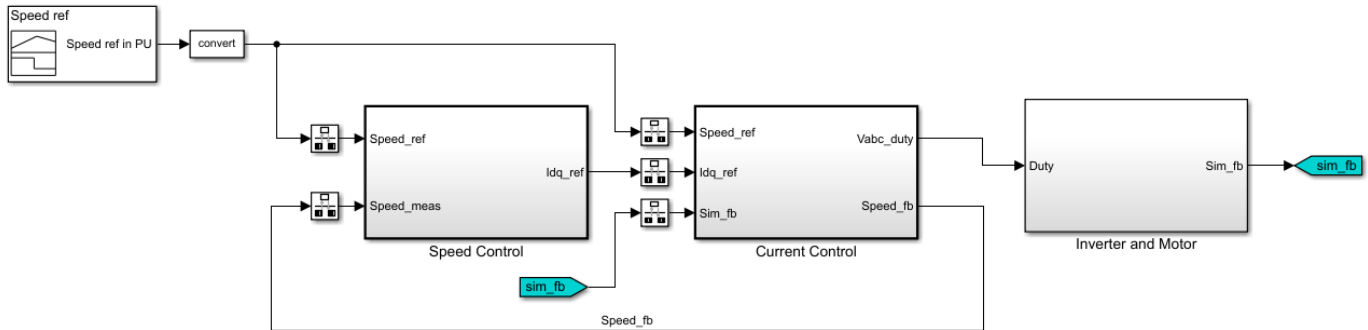

Steps:
1. Use live script, compare Reinforcement learning and PI
2. Learn more about this example.

Copyright 2021 The MathWorks, Inc.

When you open the model, it loads the configured parameters including the motor parameters to the workspace for simulation. To view and update these parameters, open the `mcb_pmsm_foc_sim_RL_data.m` model initialization script file. For details about the control parameters and variables available in this script, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

You can access the reinforcement learning setup available inside the Current Controller Systems subsystem by running this command.

`open_system('mcb_pmsm_foc_sim_RL/Current Control/Control_System/Closed Loop Control/Current Contr`
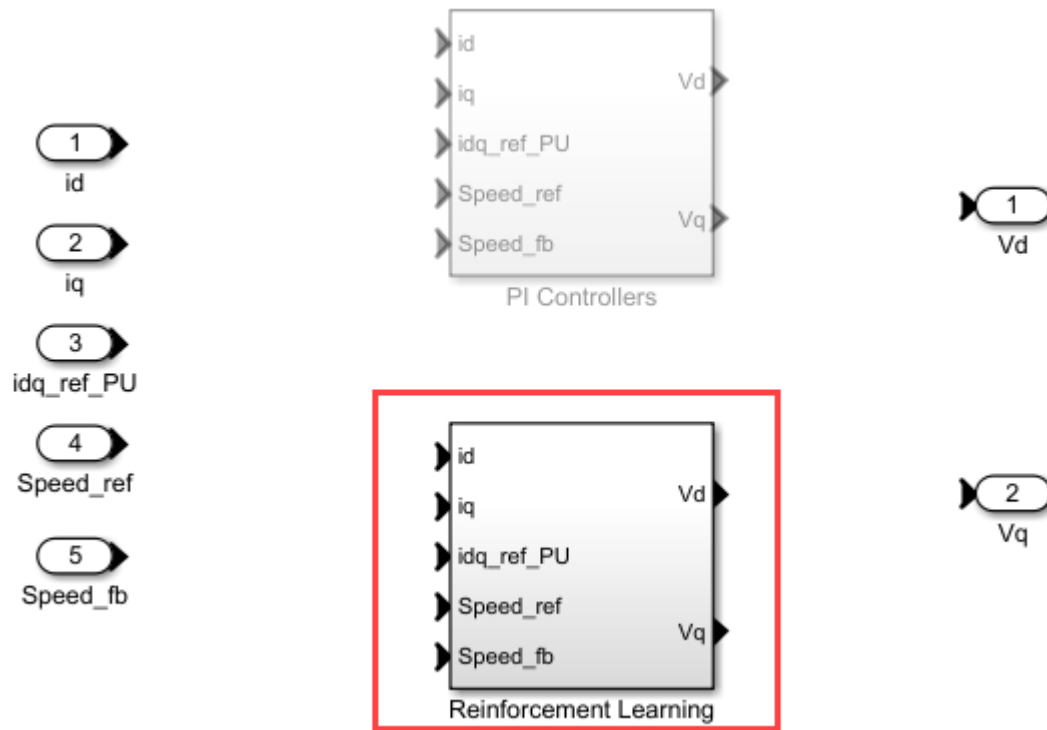
For more information about setting up and training a reinforcement learning agent to control a PMSM, see "Train TD3 Agent for PMSM Control" (Reinforcement Learning Toolbox).

Note:

- Training a reinforcement learning agent is a computationally intensive process that may take several hours to complete.
- The agent in this example was trained using the PWM frequency of 5 KHz. Therefore, the model uses this frequency by default. To change this value, train the reinforcement learning agent again by using a different PWM frequency and update the `PWM_frequency` variable in the `mcb_pmsm_foc_sim_RL_data.m` model initialization script. You can use the following command to open the model initialization script.

`edit mcb_pmsm_foc_sim_RL_data;`

Required Mathworks® Products

- Motor Control Blockset™
- Reinforcement Learning Toolbox™

Simulate Model

Follow these steps to simulate the model.

1. Open the model included with this example.

2. Run this command to select the Reinforcement Learning variant of the Current Controller Systems subsystem available inside the FOC architecture.

```
ControllerVariant='RL';
```

You can navigate to the Current Controller Systems subsystem to verify if the Reinforcement Learning subsystem variant is active.

```
open_system('mcb_pmsm_foc_sim_RL/Current Control/Control_System/Closed Loop Control/Current Contr
```
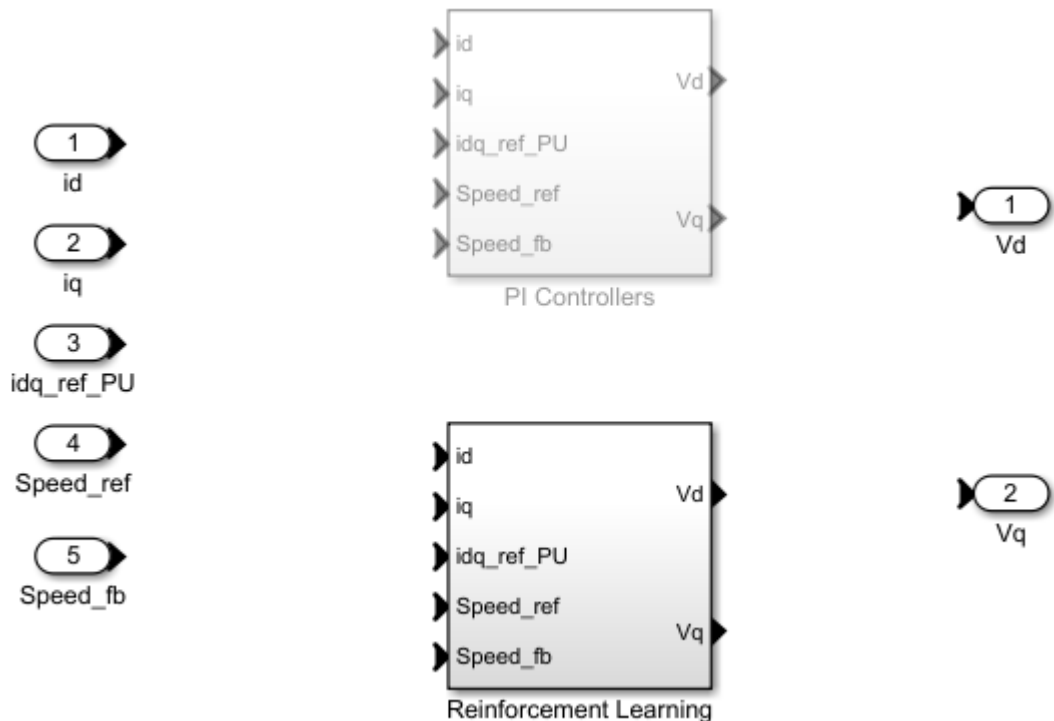

Note: The model selects the Reinforcement Learning subsystem variant by default.

3. Run this command to load the pre-trained reinforcement learning agent.

```
load('rlPMSMAgent.mat');
```

Note: The reinforcement learning agent in this example was trained to use the speed references of 0.2, 0.4, 0.6, and 0.8 PU (per-unit). For information related to the per-unit system, see "Per-Unit System" on page 6-20.

4. Click **Run** on the **Simulation** tab to simulate the model. You can also run this command to simulate the model.

```
sim(mdl);
```

```
### The Lq is observed to be lower than Ld. ###
### Using the lower of these two for the Ld (internal variable)  ###
### and higher of these two for the Lq (internal variable) for computations. ###
### The Lq is observed to be lower than Ld. ###
### Using the lower of these two for the Ld (internal variable)  ###
### and higher of these two for the Lq (internal variable) for computations. ###
```

5. Click **Data Inspector** on the **Simulation** tab to open the Simulation Data Inspector. Select one or more of these signals to observe and analyze the simlation results related to speed tracking and controller performance.

- `Speed_ref`
- `Speed_fb`
- `iq_ref`
- `iq`
- `id_ref`
- `id`
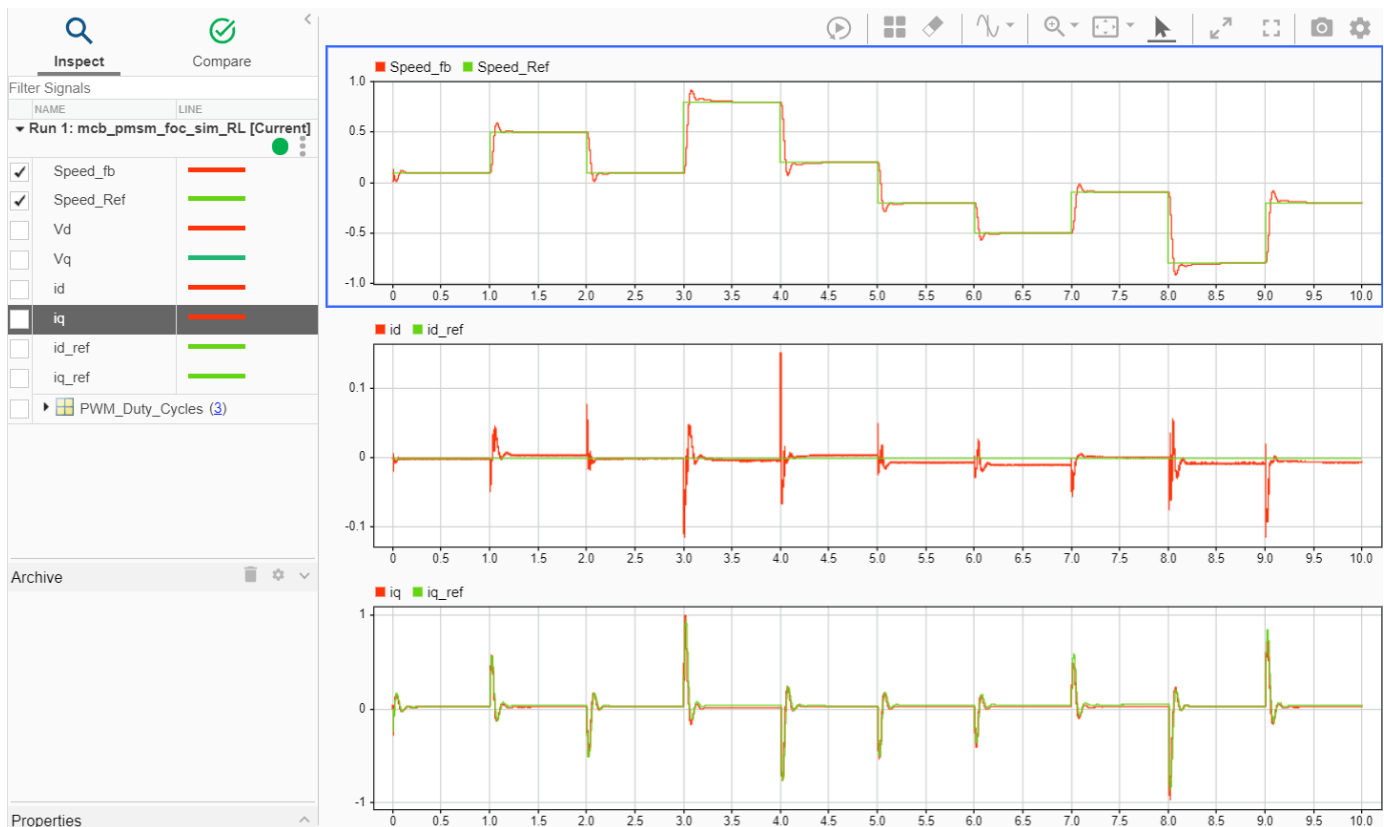
In the preceding example:

- The combination of PI and reinforcement learning controllers achieve the required speed by tracking the changes to the speed reference signal.
- The second and third data inspector plots show that the trained reinforcement learning agent acts as a current controller and successfully tracks both the `Id` and `Iq` reference currents. However, a small steady state error exists between the reference and acutal values of `Id` and `Iq` currents.

Use Simulation to Compare RL Agent with PI Controllers

Use these steps to analyze the speed tracking and controller performance of PI controllers and compare them with that of reinforcement learning agent:

1. Open the model included with this example.

2. Run this command to select the PI Controllers variant of the Current Controller Systems subsystem available inside the FOC architecture.

```
ControllerVariant='PI';
```

You can navigate to the Current Controller Systems subsystem to verify if the PI Controllers subsystem variant is active.

```
open_system('mcb_pmsm_foc_sim_RL/Current Control/Control_System/Closed Loop Control/Current Contr
```
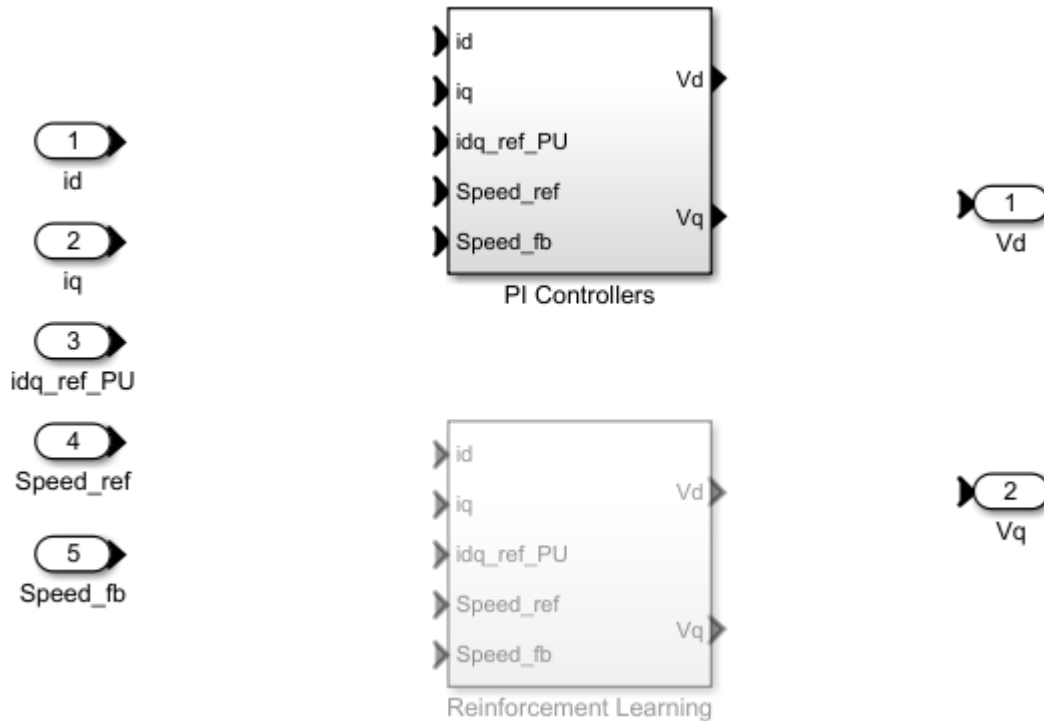

NOTE: The model selects the Reinforcement Learning subsystem variant by default.

3. Click **Run** on the **Simulation** tab to simulate the model. You can also run this command to simulate the model.

```
sim(mdl);
```

```
### The Lq is observed to be lower than Ld. ###
### Using the lower of these two for the Ld (internal variable)  ###
### and higher of these two for the Lq (internal variable) for computations. ###
### The Lq is observed to be lower than Ld. ###
### Using the lower of these two for the Ld (internal variable)  ###
### and higher of these two for the Lq (internal variable) for computations. ###
```

4. Click **Data Inspector** on the **Simulation** tab to open the Simulation Data Inspector. Select one or more of these signals to observe and analyze the simlation results related to speed tracking and controller performance.

- `Speed_ref`
- `Speed_fb`

- `iq_ref`
- `iq`
- `id_ref`
- `id`

5. Compare these results with the previous simulation run results obtained by using the RLAgent (Reinforcement Learning) subsystem variant.
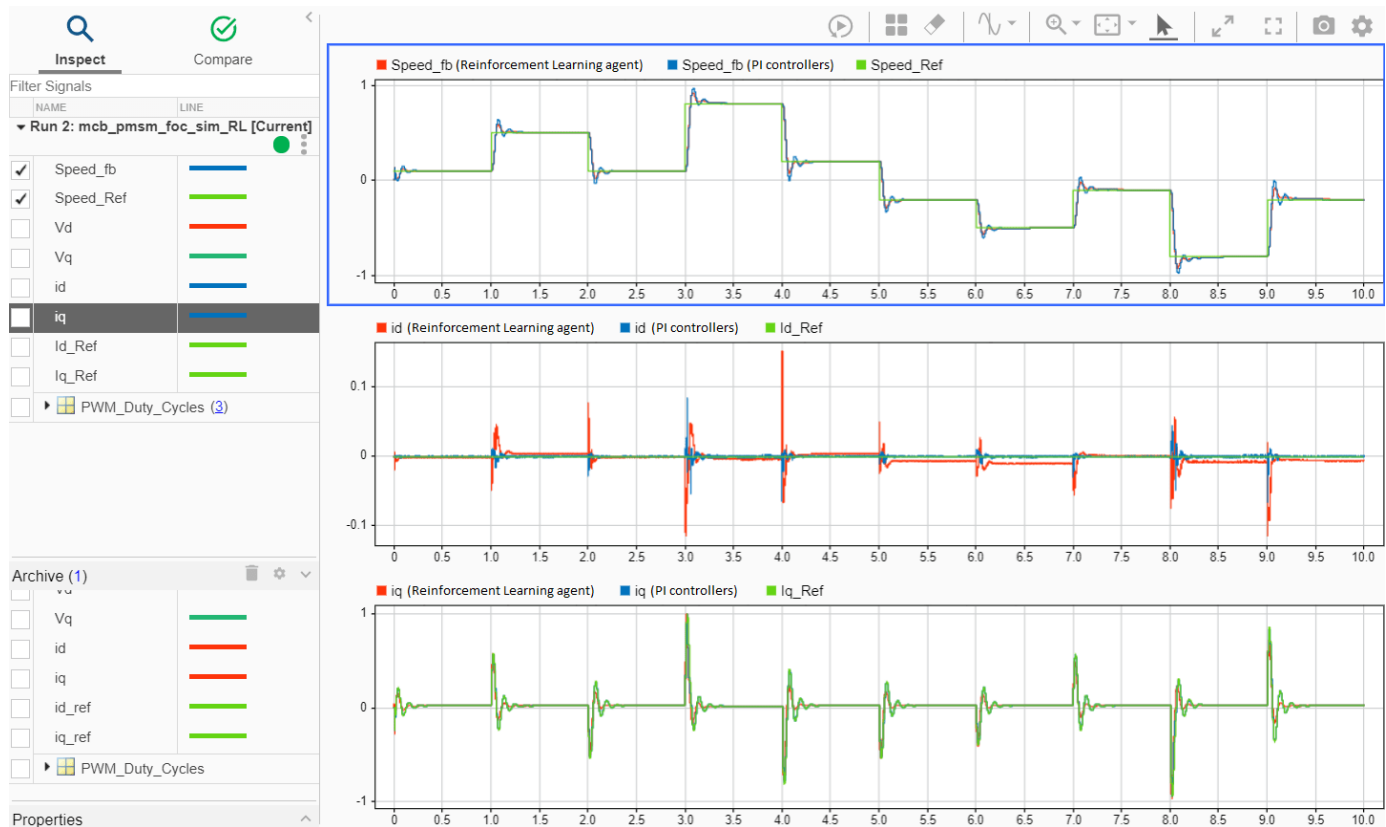
In the preceding example:

- The red signals show the simulation results that you obtain using the RLAgent (Reinforcement Learning) subsystem variant.
- The blue signals show the simulation results that you obtain using the PIControllers (PI Controllers) subsystem variant.
- The plots indicate that (with an exception of `Id` reference current tracking) the performance of reinforcement learning agent is similar to the PI controllers. You can improve the current tracking performance of the reinforcement learning agent by further training the agent and tuning the hyperparameters.

NOTE: You can also update the reference speed to higher values and similarly compare the performances between reinforcement learning agent and PI controllers.

Estimate Induction Motor Parameters Using Recommended Hardware

This example determines the parameters of a three-phase AC induction motor (ACIM) using the recommended Texas Instruments™ hardware. The example determines these parameters:

- Nominal Magnetizing Current I_{d0} (Ampere)

- Stator resistance R_s (Ohm)

- Rotor resistance R_r (Ohm)

- Magnetizing inductance L_m (H)

- Stator leakage inductance L_{ls} (H)

- Rotor leakage inductance L_{lr} (H)

- Motor inertia J (Kg.m^2)

- Friction constant B (N.m.s)

The example accepts the minimum required motor and hardware parameters, runs tests on the target hardware, and displays the estimated parameters.

Note:

- This example does not support simulation. Use the supported hardware configuration to run this example.

- This example computes nominal magnetizing current I_{d0} only if you set the **Nominal Magnetizing current (Id0)** required input field to 0. For more details, see the Prepare Workspace section.

Prerequisites

- Ensure that the motor is in the no-load condition.

- Ensure that the motor has a quadrature encoder sensor. The parameter estimation tool needs the quadrature encoder sensor to measure the rotor speed.

Supported Hardware

This example supports only the following hardware configuration:

- LAUNCHXL-F28379D controller

- BOOSTXL-DRV8305 inverter

- A three-phase AC induction motor with a quadrature encoder sensor

- DC power supply

Required MathWorks® Product

To run parameter estimation, you need:

- Motor Control Blockset™

- Embedded Coder®

- C2000™ Microcontroller Blockset

Prepare Hardware

1. Attach the inverter board to the controller board such that J1 and J2 of BOOSTXL align with J1 and 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. 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, the 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 details about the hardware setup and 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 the required tests and displays the estimated parameters.

Note: Ensure that the target model and host model that you use for parameter estimation belong to the same release version of Motor Control Blockset.

Prepare Workspace

Open the parameter estimation host model. You can also use this command to open the host model:

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

4-203

ACIM Parameter Estimation

Communication Port

No port selected

Host Serial Setup

Required Inputs

Nominal Voltage:	16.970	V (Line-Line RMS)
Input DC Voltage:	24	V
Nominal Current:	1.8	A (phase peak value)
Nominal Magnetizing current (Id0)	1.08	A (phase peak value)
Pole pairs:	2	
Rated Frequency:	50	Hz
Total QEP Slits:	1000	

Steps
1. Provide required inputs.
2. Press **Ctrl+D** to update the workspace
3. **Build, Deploy & Start** required target model
4. Select port in Host Serial Setup, Host Serial Receive, Host Serial transmit
5. **Run** this model to estimate motor parameters

Note
1. It is recommended to set the power supply DC bus voltage as:
   Input DC Voltage (V)
       = sqrt(2) * Nominal Voltage of motor (Line-Line RMS V).
2. For some phase sequences, the parameter estimation tool may not compute the inertia and friction constant parameters. Interchange any two motor phase connections and try running the parameter estimation host model again to estimate these parameters.
3. Use Nominal Magnetizing current (Id0) input as zero, to estimate Id0 using open loop control

Test Status

Run   Stop

Estimated Motor Parameters

Id0	--	A
Rs	--	Ohm
Rr	--	Ohm
Lm	--	H
Lls	--	H
Llr	--	H
Motor Inertia	--	Kg.m^2
Friction constant	--	N.m.s

Save Parameters   Open Model

Signal Conditioning and Scaling

SelectedSignal

Copyright 2021 The MathWorks, Inc.

Fault Status

Over Current

Under Voltage

Serial communication

Signal from Target

Iq

SelectedSignal

Signal

Target Model (F28379D + DRV8305): mcb_acim_param_est_f28379D_DRV8305

Enter these details in the host model to prepare the workspace.

- **Communication Port** — In the block parameter dialog boxes of Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks, select the serial **Port** to which the hardware is connected. Select an available port from the list. For more details, see "Find Communication Port" on page 6-4.

- **Required Inputs** — Enter the motor specification and hardware setup data. You can obtain these values either from the motor datasheet or from the motor nameplate.

  - **Nominal Voltage** — The rated voltage (line-to-line RMS value) of the motor (Volts).

  - **Input DC Voltage** — The DC supply voltage for the inverter (Volts).

  - **Nominal Current** — The rated current (phase peak value) of the motor (Ampere).

  - **Nominal Magnetizing current (Id0)** — The rated magnetizing current of the motor (Ampere). If you do not know the nominal magnetizing current of your motor and want the parameter

estimation tool to automatically calculate I_{d0} and use the computed value as input, set this field to 0. The tool updates the display box **Id0** (available in the **Estimated Motor Parameters** section of the host model) with the computed I_{d0} value only if you set this field to 0.

If you know I_{d0} of your motor and do not want the tool to compute this value, set this field with the (positive) I_{d0} value. The tool uses this input to perform the subsequent calculations.

- **Pole Pairs** — The number of pole pairs of the motor.

- **Rated Frequency** — The rated frequency of operation of the motor (Hertz).

- **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 tests protect the hardware from over-current faults. However, to ensure that these faults do not occur, keep the rated current of the motor (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 need to 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 values from the host model.

Click this hyperlink available in the parameter estimation host model to open the target model:

- mcb_acim_param_est_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.
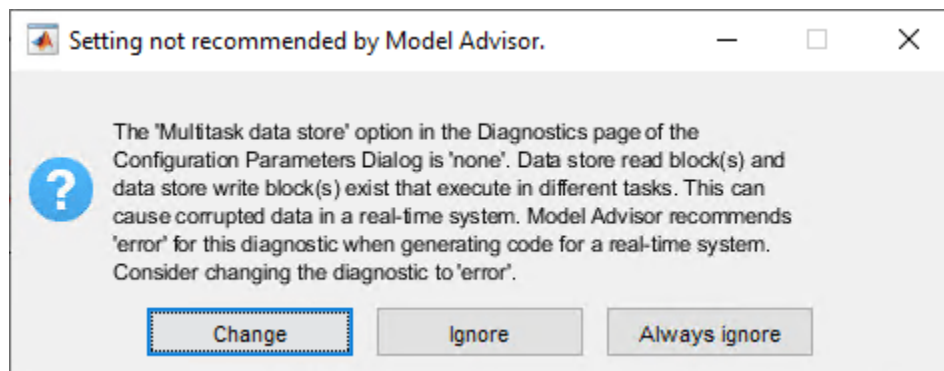
Workflow 2: Manually Download Target Model

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).

- Locate the binary files (.hex/ .out) at this location:

    -< matlabroot >\toolbox\mcb\mcbexamples
\mcb_acim_param_est_f28379D_DRV8305.out

The mcb_acim_param_est_f28379D_DRV8305.out file uses fixed quadrature encoder slits count of 1000, therefore you can use this file only for motors connected to a quadrature encoder with 1000 slits.

- 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, check if the **Run-Stop** slider switch position is **Run**. Then, click **Run** in the **Simulation** tab to run the parameter estimation tests.

3. The host model displays the estimated motor parameters after successfully completing the tests.

When computing the electrical parameters (stator resistance R_s, rotor resistance R_r, magnetizing inductance L_m, stator leakage inductance L_{ls}, and rotor leakage inductance L_{lr}), the parameter estimation tool keeps the motor stationary. However, the tool runs the motor to compute the mechanical parameters (nominal magnetizing current I_{d0}, motor inertia J, and friction constant B).

Note: You can ignore the displayed mechanical parameters, if the motor does not run.

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 to rerun the parameter estimation tests.

During an emergency, you can manually turn the **Run-Stop** slider switch to the **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:
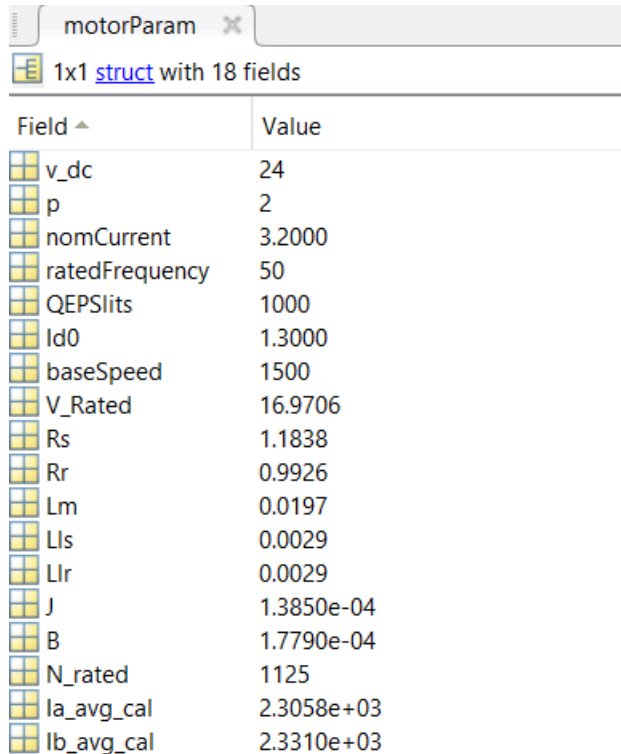
1. Over-current fault (this fault occurs when actual current drawn from the power supply is more than the **Nominal Current** value specified in the **Required Inputs** section of the host model)

2. Under-voltage fault (this fault occurs when input DC voltage drops below 80% of the **Input DC Voltage** value specified in the **Required Inputs** section of the host model)

3. Serial communication fault

Save Estimated Parameters

You can export the estimated motor parameters and then use them for the simulation and control system design.

To export, click **Save Parameters** to save the estimated parameters in 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 18 fields	
Field ▲	**Value**
v_dc	24
p	2
nomCurrent	3.2000
ratedFrequency	50
QEPSlits	1000
Id0	1.3000
baseSpeed	1500
V_Rated	16.9706
Rs	1.1838
Rr	0.9926
Lm	0.0197
Lls	0.0029
Llr	0.0029
J	1.3850e-04
B	1.7790e-04
N_rated	1125
Ia_avg_cal	2.3058e+03
Ib_avg_cal	2.3310e+03

Click **Open Model** to create a new Simulink® model with an Induction Motor block. The block uses the `motorParam` structure variables from the MATLAB workspace.

Note:

- For some phase sequences, the parameter estimation tool may not compute the **Motor Inertia** and **Friction Constant** parameters. Interchange any two motor phase connections and try running the parameter estimation host model again to estimate these parameters.

- Under the following conditions, slightly increase the **Nominal Current** required input in the host model (for example, increase by 10% of the original value) and run the parameter estimation tests again:

    - When the host model runs and executes the tests, the **Speed** debug signal (in the time scope available in the host model) does not reach a stable value of around `0.6` per-unit (PU).

    - When running the host model multiple times, the **Motor Inertia** and **Friction Constant** values vary.

Repeat this step until the **Speed** signal stabilizes (at around `0.6` PU) in the time scope. This ensures that the computed **Motor Inertia** and **Friction Constant** values are accurate.

- It is recommended to set the power supply DC bus voltage to the following value:

$$Input\ DC\ Voltage\ (V) = (\sqrt{2}) \times [Nominal\ Voltage\ of\ motor\ (V)\ (line-line\ RMS\ value)]$$

- You can determine the total leakage inductance (in Henry) using the L_{ls} and L_{lr} values computed by the parameter estimation tool:

$$L_{ls} = L_{lr} = \left(\frac{Total\ leakage\ inductance}{2} \right)$$

- The parameter estimation tool does not estimate rated slip (`slip_rated`). The tool computes the rated speed $((1 - \text{slip\_rated}) \times \text{synchronous speed})$ assuming that $\text{slip\_rated} = 0.25$. We recommend that you measure `slip_rated` or obtain it from the motor datasheet.

Estimate PMSM Parameters Using Custom Hardware

This example includes an algorithm to determine the parameters of a permanent magnet synchronous motor (PMSM) using any custom motor-control hardware (hardware not used in the Motor Control Blockset™ examples). The algorithm determines these parameters:

- Phase resistance, R_s (Ohm)

- d axis inductance, L_d (Henry)

- q axis inductance, L_q (Henry)

- Back-EMF constant, K_e (Vpk_LL/krpm, where Vpk_LL is the peak voltage line-to-line measurement)

- Motor inertia, J (Kg.m^2)

- Friction constant, B (N.m.s)

The algorithm accepts the minimum required inputs and runs tests on the target hardware to estimate the PMSM parameters.

The example needs a quadrature encoder sensor to measure the rotor position and provide real-time rotor position feedback. This workflow helps you to integrate the parameter estimation algorithm with the drivers for your motor-control hardware. It supports any three-phase PMSM.

The workflow includes these four steps to prepare, deploy, and run the PMSM parameter estimation algorithm on your hardware:

1. Generate code for the parameter estimation algorithm using Embedded Coder®

2. Obtain C code for the custom hardware drivers

3. Integrate parameter estimation algorithm code with the driver code

4. Deploy the integrated code to hardware

Note: This workflow does not support simulation. You can use any motor-control hardware to run this example.

Prerequisites

- Ensure that the PMSM has a quadrature encoder sensor and calibrate the sensor.

The parameter estimation alogrithm needs the motor position as detected by a quadrature encoder position sensor. To detect the motor position correctly by using the sensor, calibrate the quadrature encoder that is attached to the motor under test. For instructions, see "Quadrature Encoder Offset Calibration" on page 8-11.

- Calibrate the offset values of the ADC (or current sensor) peripheral available in your hardware. For instructions, see "Open-Loop Control and ADC Offset Calibration" on page 8-2.

- Ensure that the PMSM is in no-load condition.

Custom Hardware Configuration

- Controller hardware

- Inverter hardware

- A PMSM with a quadrature encoder sensor

- DC power supply

Required MathWorks® Products

To build the parameter estimation algorithm included in this example, you need these products:

- Motor Control Blockset™

- Fixed-Point Designer™

- Embedded Coder®

Open MATLAB® Project and Prepare Parameter Estimation Model

Use one of these methods to open the MATLAB project:

- Click **Open Example**.

- Run the command `mcb_ParameterEstimationAlgorithmStart` at the command prompt.

The MATLAB project opens and shows the following files:

- parameter_estimation_algorithm.slx (model containing the parameter estimation algorithm)

- parameter_estimation_init.m (model initialization script for the parameter estimation algorithm)

Note:

- Verify and update the motor, inverter, and other parameters related to the target hardware and parameter estimation algorithm in the model initialization script `parameter_estimation_init.m`.

- The rated speed of the motor (variable `motorParam.ratedSpeed`) must be less than 25000 RPM.

- The tests protect the hardware from over-current faults. However, to ensure that these faults do not occur, keep the rated current of the motor (variable `motorParam.nomCurrent`) 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.

Generate Code For Parameter Estimation Algorithm Using Embedded Coder

1. After you open the MATLAB project, double-click the `parameter_estimation_algorithm.slx` model.
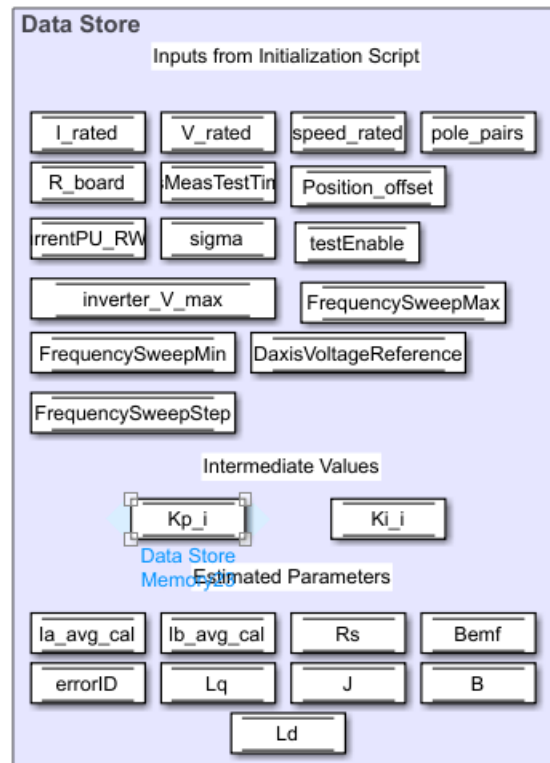
Copyright 2021 The MathWorks, Inc.

2. Select **Modeling > Model Settings > Model Settings** to open the Configuration Parameters dialog box.

3. In the **Solver Selection** area of the **Solver** tab, update the **Type** and **Solver** fields.

4. In the **Hardware Implementation** tab of the Configuration Parameters dialog box, configure the parameters according to your hardware.

5. Update the hardware setup parameters (including the quadrature encoder and ADC offsets) in the model initialization script (parameter_estimation_init.m).

6. In the Simulink toolstrip of the model, select **Apps > Embedded Coder** to open the Embedded Coder application.
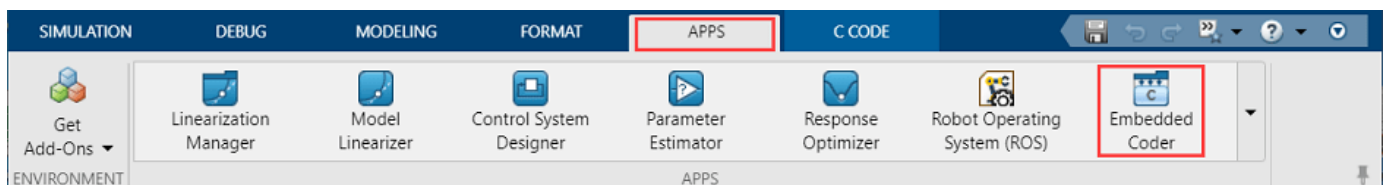
7. In the Simulink toolstrip, select **C Code > Code Interface > Default Code Mappings** to open the Code Mappings - C dialog box.
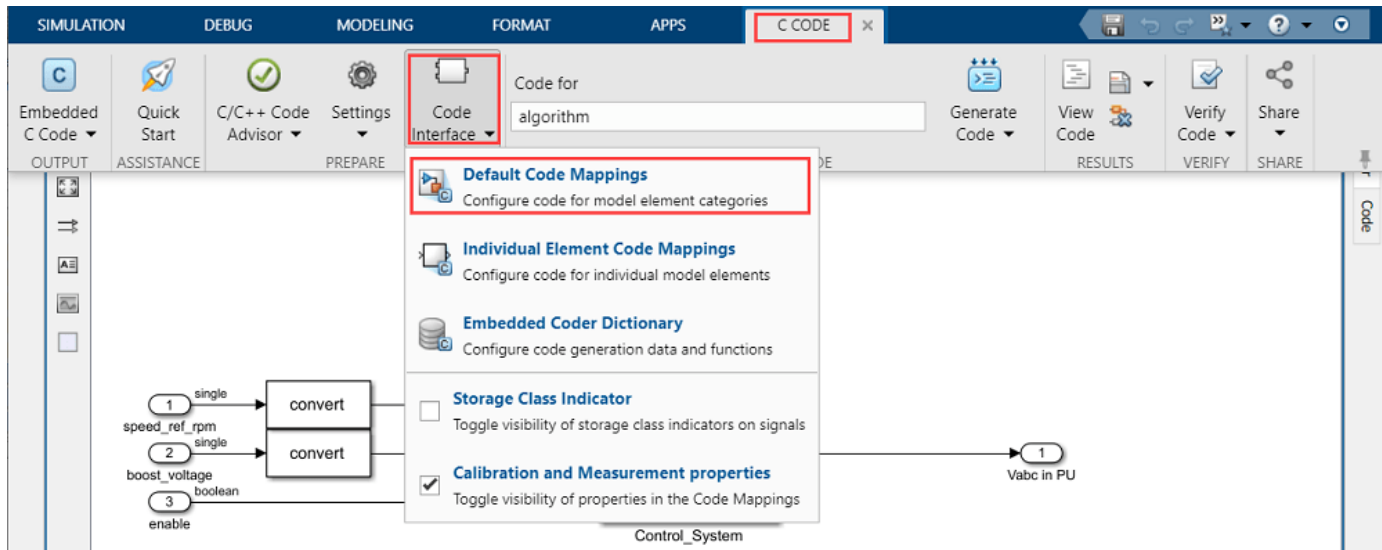
8. In the Code Mappings - C dialog box, open the **Functions** tab.

9. For a listed C function, click the hyperlink under the **Function Preview** column to open the Configure C Initialize Function Interface dialog box.

10. Use the Configure C Initialize Function Interface dialog box to configure the interface and arguments of the C function.

11. Click **Apply** and **OK** to complete configuring the C function.

12. Repeat steps 9 to 11 for all the listed functions.

13. In the Simulink toolstrip of the target model, select **C Code > Generate Code > Build** to build the model and generate a `.c` file for the target model for the current controller.
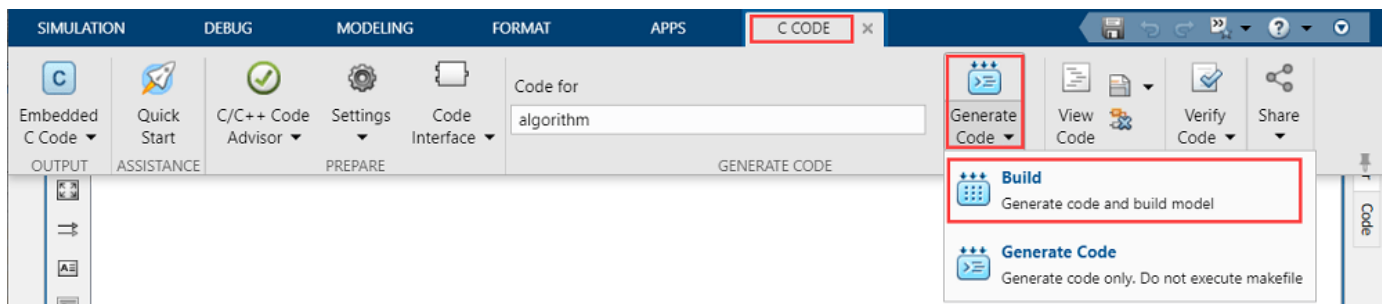
This image shows an example of a C function available in the generated current controller code.

```
/* Model step function */
void parameter_estimation_step(real32_T arg_enable_tests, uint16_T arg_Vabc_Vdc
  [4], uint16_T arg_Iab[2], uint16_T arg_QEP_Count, uint16_T arg_QEP_Index,
  real32_T *arg_enable_inverter, real32_T arg_PWM_duty[3], real32_T
  *arg_position, real32_T *arg_speed, uint32_T *arg_estParaIdx, real32_T
  *arg_estimatedParameter)
{
  real32_T rtb_Add1_e;
  real32_T rtb_Add3;
  real32_T rtb_InvertingNonInverting_idx_0;
  real32_T rtb_InvertingNonInverting_idx_1;
  real32_T rtb_Merge;
  real32_T rtb_Merge_f;
  real32_T rtb_MultiportSwitch_idx_0;
  real32_T rtb_MultiportSwitch_idx_1;
  real32_T rtb_PositionGain;
  real32_T rtb_Product_kr;
  real32_T rtb_Switch_bn;
  real32_T rtb_Switch_m_idx_0;
  uint32_T rtb_PositionToCount;
  uint16_T rtb_Sum3_b;
  boolean_T rtb_UnitDelay_kd;

  /* Gain: '<S9>/Inverting//Non Inverting' incorporates:
```

Note: The generated C function uses the interface that you configured in step 10.

Obtain C Code For Custom Hardware Drivers

You can use the code generation software supported by the hardware manufacturer to configure the hardware peripherals and generate C code for the hardware drivers.

Alternatively, you can also use a manually written driver code.

Integrate Parameter Estimation Algorithm Code With Driver Code

1. Call the parameter estimation algorithm functions from the driver code using the configured function parameters. This image shows a call to a parameter estimation algorithm C function.

```
// ADC conversion complete interrupt configured to trigger at every 50 us
void HAL_ADCEx_InjectedConvCpltCallback(ADC_HandleTypeDef *hadc){

    // call to parameter_estimation_step function here
```

2. Use the return value from the function call to complete integrating the driver with the parameter estimation algorithm.

This figure describes the program control flow of the example.
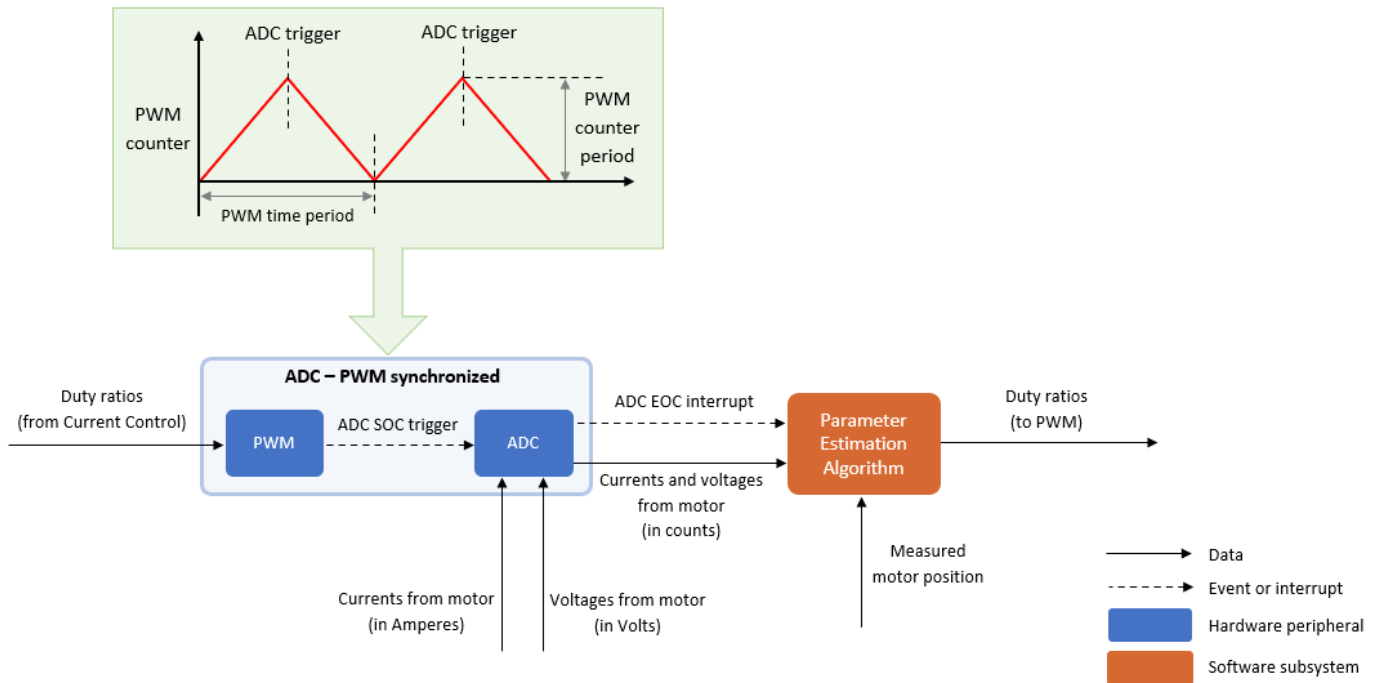
For other details about the recommended code structure (that is used by the Motor Control Blockset™ examples), see "Program Control Flow of Motor Control Blockset Examples" on page 6-23.
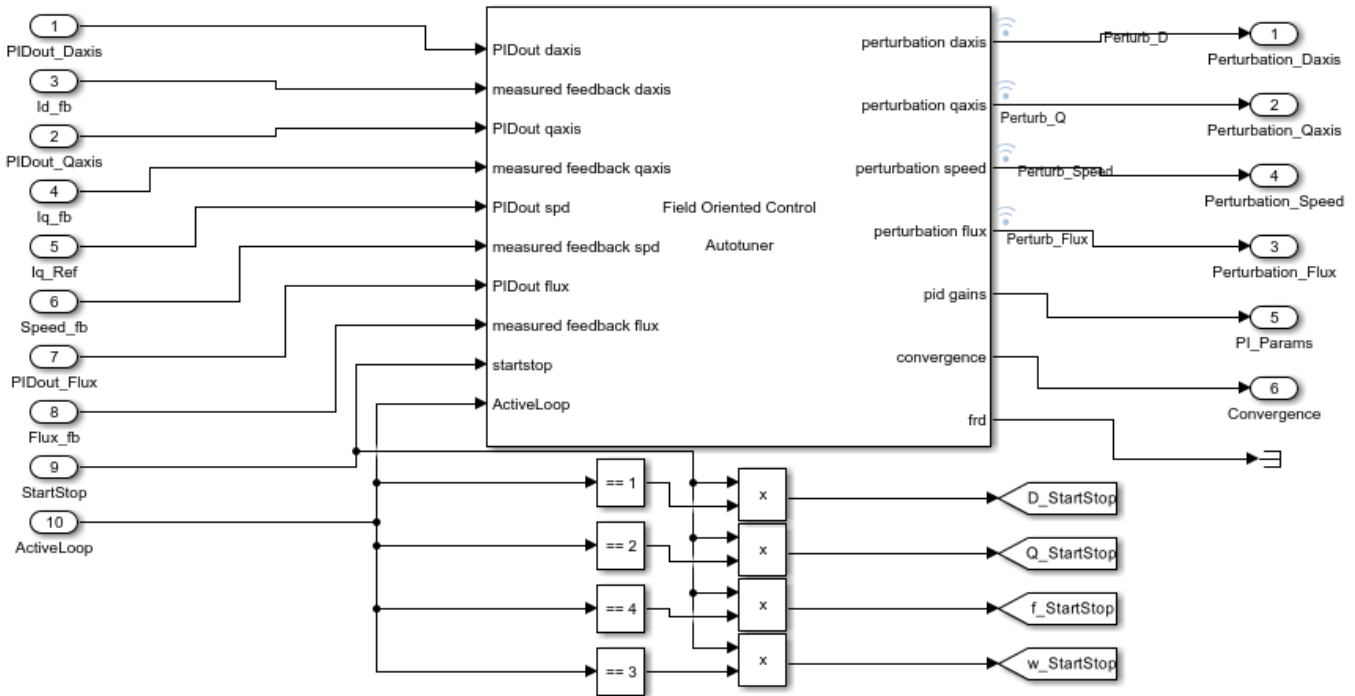
Deploy Integrated Code to Hardware

1. Complete the hardware connections.

2. Use the code generation and deployment software supported by the hardware manufacturer to compile, build, and generate a binary (for example .HEX) file from the integrated code. Use the software to flash the binary file to the target hardware.

3. The integrated code running on the target hardware saves the computed motor parameters in these global variables:

- Rs — Phase resistance

- Ld — d axis inductance

- Lq — q axis inductance

- Bemf — Back-EMF constant

- J — Motor inertia

- B — Friction constant

Tune PI Controllers (in Field-Weakening Control Mode) Using FOC Autotuner Block

This example uses the Field Oriented Control Autotuner block to compute the gain values of the PI controllers available in the speed, current, and flux control loops of a field-weakening control algorithm. For details about this block, see Field Oriented Control Autotuner.

The example automatically computes the PI controller gains to run an Interior Permanent Magnet Synchronous Motor (IPMSM) using field-weakening control. Field-weakening control is an operating mode that runs the motor at speeds greater than the base speed or rated speed. To enter this mode the example reduces the d-axis stator current (I_d) to a negative value, which reduces the rotor flux linkage. This enables the motor to run above the base speed. For more information about this operating mode, see "Field-Weakening Control (with MTPA) of PMSM" on page 4-48.

Use the code-generation capability of the example to deploy the gain-tuning algorithm to the target hardware. This enables you to run the algorithm on the target hardware connected to a motor and compute accurate PI controller gains by processing motor feedback in real-time on the hardware. The example uses a quadrature encoder sensor to measure the rotor position.

Note: This example provides the field-weakening control algorithm as a reference. You can refer this example and use a similar approach to add the Field Oriented Control Autotuner block and the gain-tuning algorithm to the field-weakening logic available in your model.

Model

The example includes the target model mcb_ipmsm_fwc_autotuner_f28379d.

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

```
open_system('mcb_ipmsm_fwc_autotuner_f28379d.slx');
```

The Field Oriented Control Autotuner block iteratively tunes the d- and q-axis current control, speed and flux control loops and computes the gains of the current, speed, and flux PI controllers. Use this command to locate the Field Oriented Control Autotuner block available inside the model:

`open_system('mcb_ipmsm_fwc_autotuner_f28379d/Current Control/Control_System/Closed Loop Control/`

In addition to the current and speed feedback from the plant, the block processes the reference flux and flux feedback values. It also processes the current and speed PI controller outputs to compute the PI controller gains (Kp and Ki).

For more details on the FOC architecture, see "Field-Oriented Control (FOC)" on page 4-3.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Simulink Control Design™
- Stateflow® (needed only if you modify the example model)

To generate code and deploy model:

- Motor Control Blockset™
- Simulink Control Design™
- Embedded Coder®
- C2000™ Microcontroller Blockset
- Stateflow (needed only if you modify the example model)

Prerequisites for Simulation and Hardware Deployment

1. Open the model initialization script for the target model. Check and update the motor, inverter, and other control system and hardware parameters available in the script. For instructions on locating and editing the model initialization script associated with a target model, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

2. In the **Inverter & Target Parameters** section of the model initialization script, verify that the `mcb_SetInverterParameters` function uses the argument `BoostXL-DRV8305`. This enables the script to use the preprogrammed parameters for the BOOSTXL-DRV8305 inverter.

3. Configure these parameters correctly in the model initialization script. These variables are essential for the gain-tuning algorithm to compute the PI controller gains. If the values of these variables are incorrect, the model may fail to bring the motor to the steady speed state.

- `pmsm.p`

- `pmsm.I_rated`

- `pmsm.PositionOffset`

- `pmsm.QEPSlits`

4. If you are using a motor that is not listed in the `mcb_SetPMSMMotorParameters` function (used in the **System Parameters // Hardware parameters** section of the model initialization script), tune the default values of the following initial gains available in the **Initial PI parameters** section of the model initialization script. This ensures that the motor reaches the steady state of speed-control operation:

- `PI_params.Kp_Id`

- `PI_params.Ki_Id`

- `PI_params.Kp_Iq`

- `PI_params.Ki_Iq`

- `PI_params.Kp_Speed`

- `PI_params.Ki_Speed`

- `PI_params.Kp_Flux`

- `PI_params.Ki_Flux`

When you either simulate or run the example on a target hardware, the example uses crude values of the PI controller gains to achieve the steady state of speed-control operation.

Note: When using this example, if the motor (whether it is listed or not in the `mcb_SetPMSMMotorParameters` function) does not run, try tuning the default values of these parameters.

5. In the **FOC Autotuner parameters** section of the model initialization script, check and update the parameters of the Field Oriented Control Autotuner block. This sets the reference bandwidth and phase margin values for both the speed and the current PI controllers.

Simulate the Target Model

Simulating the example is optional. Follow these steps to simulate the target model:

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 `mcb_ipmsm_fwc_autotuner_f28379d/Current Control/PI_Params_Display_and_Logging` subsystem.

The computed gains might not be accurate because step 3 in the Prerequisites for Simulation and Hardware Deployment section checks the accuracy of only four motor parameters.

If you want to compute and test the PI controller gains using simulation, follow these steps before clicking **Run** on the **Simulation** tab of the target model.

- In the **System Parameters // Hardware parameters** section of the model initialization script, verify that the `mcb_SetPMSMMotorParameters` function uses an argument that represents your motor (for example, `Teknic2310P`). Open the `mcb_SetPMSMMotorParameters` function to see the preprogrammed cases that store the motor parameters of commonly used PMSMs.

If the `mcb_SetPMSMMotorParameters` function does not list your PMSM, determine the parameters for your motor using these steps.

- If you have motor control hardware, you can estimate the parameters for your motor, by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate PMSM Parameters Using Recommended Hardware" on page 4-186 and "Estimate PMSM Parameters Using Custom Hardware" on page 4-209.

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

- If you obtain the motor parameters from the datasheet or other sources, add and configure the motor parameters in the model initialization script. These parameter values override the selected pre-programmed case in the function `mcb_SetPMSMMotorParameters`.

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

After you simulate the target model and determine the gains, update your model (that implements field-weakening control) with the computed gain values to quickly bring the motor to a steady speed state.

Deploy the example to the target hardware to tune the PI controller gains more accurately by using an actual hardware connected to a motor. For more details, see the Generate Code and Deploy Model to Target Hardware section.

Generate Code and Deploy Model to Target Hardware

This section shows how to generate code and run the algorithm for tuning the PI controller gains on the target hardware. Running the example on the hardware enables you to compute the PI controller gains more accurately by processing the feedback from an actual plant in real-time.

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. Before you run the host model on the host computer, 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 closed-loop control.

Required Hardware

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB® command prompt.

LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_ipmsm_fwc_autotuner_f28379d

For more information on connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. The model automatically computes the analog to digital converter (ADC) offset (also known as current offset). To disable this functionality (enabled by default), update the value of the `inverter.ADCOffsetCalibEnable` variable in the model initialization script to `0`.

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 4-10.

3. Compute the quadrature encoder index offset value and update it in the `pmsm.PositionOffset` variable in the model initialization script of the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79.

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

5. Load a sample program to the CPU2 of the LAUNCHXL-F28379D board. For example, load the program that operates the CPU2 blue LED by using GPIO31 (`c28379D_cpu2_blink.slx`). This ensures 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. Verify the variables updated by the target model in the base MATLAB workspace.

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_ipmsm_fwc_autotuner_host_f28379d.slx');
```


Tuning PI Controllers for Field-Weakening Control of IPMSM Using FOC Autotuner - Host

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

8. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

9. Turn the **Motor** slider switch to the **Start** position to start running the motor.

10. Enter the motor's rated speed value in the **Speed Ref [RPM]** field to start running the motor at the rated speed.

Use the **Vm_Limit & Vm_Feedback** signals listed in the **Debug signals** section to determine a steady value of the **Vm_Feedback** signal.

11. Enter a value that is less than the steady **Vm_Feedback** signal value in the **Max Voltage Limit [PU]** field. For example, if **Vm_Feedback** has a steady value of `0.9`, then you can enter a value such as `0.8` in the **Max Voltage Limit [PU]** field.

Note: Keeping **Max Voltage Limit [PU]** greater than **Vm_Feedback** will result in incomplete tuning due to an inactive flux loop.

12. Enter a reference speed the **Speed Ref [RPM]** field that is greater than the motor's rated speed.

Note: The model tunes the flux control loop only if the reference speed that you provide is greater than the motor's rated speed.

13. In the **Debug signals** section, select **Speed_Ref & Speed_Feedback** and monitor the speed signals in the **SelectedSignals** time scope. Wait until the motor reaches a steady speed.

The example can begin tuning only in the steady speed state.

14. Check that the **PI Parameters** slider switch is in the **Autotuner** position.

15. Turn the **Autotuner** slider switch to the **Start** position to start autotuning the PI controller gains. The tuning process introduces perturbations depending on the controller goals (bandwidth and phase margin) in the controller output. The example uses the system response to the perturbations to calculate the optimal controller gain values.

The model performs these tests iteratively on the motor and determines an accurate set of `Kp` and `Ki` gains for the current, speed, and flux PI controllers.

The **Tuning Status** display changes status from `Tuning not started` to `Tuning in progress`.

Note: When tuning is in progress, ensure that the **PI Parameters** slider switch remains in the **Autotuner** position.

16. When the tuning process successfully completes, the **Tuning Status** display changes status from `Tuning in progress` to `Tuning complete`.

The target model updates the speed, current, and flux PI controllers running on the target hardware with the computed `Kp` and `Ki` gains. In addition, the host model displays these values.

17. If the gain-tuning algorithm encounters an error during the tuning process, the **Tuning Status** display shows `Tuning failed`. Turn the **Autotuner** slider switch to the **Stop** position and see the **Troubleshooting** section for the troubleshooting instructions.

18. If you have successfully completed the tuning process, turn the **Autotuner** slider switch to the **Stop** position. Then turn the **PI Parameters** slider switch to the **Default** position to enable the default operating mode of the target model. In this mode, the target model uses the computed gain values to operate the motor using field-weakening control.

Note:

- If you want to run the tuning process again, ensure that you turn the **PI Parameters** switch to the **Default** position. This ensures that the host model resets the gain values that it displays. You can restart the tuning process (using the **Autotuner** switch) after turning the **PI Parameters** switch back to **Autotuner** position.

- If the Tuning Status is "Field-Weakening Control was not active," then further reduce **Max Voltage Limit [PU]** and restart the tuning process.

19. Validate the computed gain values. For instructions, see the **Validate Computed PI Controller Gains** section.

Validate Computed PI Controller Gains

1. Check that the motor is running and that the **PI Parameters** slider switch is in the **Default** position.

2. Select the **Speed_Ref & Speed_Feedback** debug signal in the **Debug signals** section of the host model.

3. Open the **SelectedSignals** time scope to monitor the reference speed and speed feedback signals.

4. Update the reference speed (for your motor control application) in the **Speed Ref [RPM]** field and monitor the signals in the time scope.

5. In the **SelectedSignals** window, navigate to **Tools > Measurements** and select **Cursor Measurements** to display the **Cursor Measurements** area.

6. Drag cursor-1 to a position that indicates zero `Speed_Ref` (just before Speed_ref rises). Drag cursor-2 to a position where `Speed_Feedback` meets `Speed_Ref` for the first time.

ΔT indicates the actual response time of the FOC algorithm (time taken by the motor to reach 100% of the reference speed from zero reference speed).

7. For the speed PI controller, use the `PI_params.SpeedBW` variable available in the model initialization script to determine the bandwidth of the speed PI controller. Compute the theoretical response time using this relation:

$$Response\_time = \left(\frac{2}{PI\_params.SpeedBW} \right)$$

Compare the theoretical `Response_time` with the actual response time ΔT to validate the speed PI controller gains.

Similarly, you can validate the current PI controller gains by analyzing the step responses of the flux and the d and q current PI controllers.

Troubleshooting

Follow these steps to troubleshoot failed gain-tuning instances.

1. Identify the loop (either d current, q current, flux, or speed) for which the tuning process failed.

The target model tunes the PI controllers in this sequence:

d current controller → q current controller → flux controller → speed controller

Tuning failure of one controller in this sequence results in incorrect gain tuning for the subsequent controllers.

Check the computed gains for the four controllers using the Display blocks available in the host model. A zero `Kp` or `Ki` controller gain value indicates that the tuning process failed for the respective controller.

Follow the subsequent steps for the first PI controller in the preceding sequence for which the tuning failed.

2. Select the controller reference and feedback signals for the controller identified in step 1, in the **Debug signals** section (for example, **Iq_Ref & Iq_Feedback** for the q current controller) and open the **SelectedSignals** time scope.

3. Check that the **PI Parameters** slider switch is in the **Autotuner** position.

4. Turn the **Autotuner** slider switch to the **Start** position to run the tuning process again.

5. Monitor the feedback signal for the controller identified in step 1 (for example, `Iq_Feedback`) in the **SelectedSignals** time scope.

Case 1: Follow these steps if the peak value of the controller feedback signal satisfies one of these conditions:

- Value is too high (greater than 1)

- Value is too low (less than `PI_params.CurrentSineAmp` for the current controllers, less than `PI_params.FluxSineAmp` for the flux controller, or less than `PI_params.SpeedSineAmp` for the speed controller)

Note: The `PI_params.CurrentSineAmp`, `PI_params.FluxSineAmp`, and `PI_params.SpeedSineAmp` variables are defined in the model initialization script.

a. If the controller identified in step 1 is the d or the q current controller, modify the `PI_params.CurrentSineAmp` variable such that it is less than the peak value of the controller feedback signal.

b. If the controller identified in step 1 is the flux controller, modify the `PI_params.FluxSineAmp` variable such that it is less than the peak value of the controller feedback signal.

c. If the controller identified in step 1 is the speed controller, modify the `PI_params.SpeedSineAmp` variable such that it is less than the peak value of the controller feedback signal.

d. Turn the **Autotuner** slider switch to the **Stop** position and then to the **Start** position to run the tuning process again.

Case 2: Follow these steps if the peak value of the controller feedback signal lies in the range:

- $[PI\_params.CurrentSineAmp, 1]$ (for the current controllers)

- $[PI\_params.FluxSineAmp, 1]$ (for the flux controller)

- $[PI\_params.SpeedSineAmp, 1]$ (for the speed controller)

Note: The `PI_params.CurrentSineAmp`, `PI_params.FluxSineAmp`, and `PI_params.SpeedSineAmp` variables are defined in the model initialization script.

a. Update the parameters of the Field Oriented Control Autotuner block (that set the reference bandwidth and phase margin values) available in the **FOC Autotuner parameters** section of the model initialization script.

b. Turn the **Autotuner** slider switch to the **Stop** position and then to the **Start** position to run the tuning process again.

Field-Oriented Control (FOC) of PMSM Using Hardware-In-The-Loop (HIL) Simulation

This example uses hardware-in-the-loop (HIL) simulation to implement the field-oriented control (FOC) algorithm 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 more information on FOC, see "Field-Oriented Control (FOC)" on page 4-3.

NOTE: This example runs only on Windows and Linux platforms. It is not supported on the Mac platform.

When the actual motor and inverter hardware are not available, you can use the HIL simulation workflow to validate the FOC algorithm in real-time by operating a realistic virtual plant. The HIL simulation setup consists of these elements:

- Desktop computer or development hardware running Simulink®

- Controller hardware running the code for the controller

- Target hardware (FPGA) running the code for the physical plant

You need to deploy the controller code to the controller hardware and the HDL code for the plant to the FPGA target hardware. After deploying the controller code, the controller runs the FOC algorithm and outputs actual PWM signals. Whereas, after HDL code deployment, the FPGA hardware effectively emulates the actual inverter and motor by running the HDL code for the plant. Therefore, it replicates the actual plant by accepting the PWM signals and providing realistic current and rotor position feedback to the controller in real-time. The FPGA hardware runs in external mode and logs data in the Simulation Data Inspector of the Simulink Real-Time model.

The Simulink host model running on the desktop computer, interacts with the controller using serial communication protocol. You can use the host model to communicate with the controller, and therefore, control the motor operation.

Hardware-In-The-Loop (HIL) Simulation
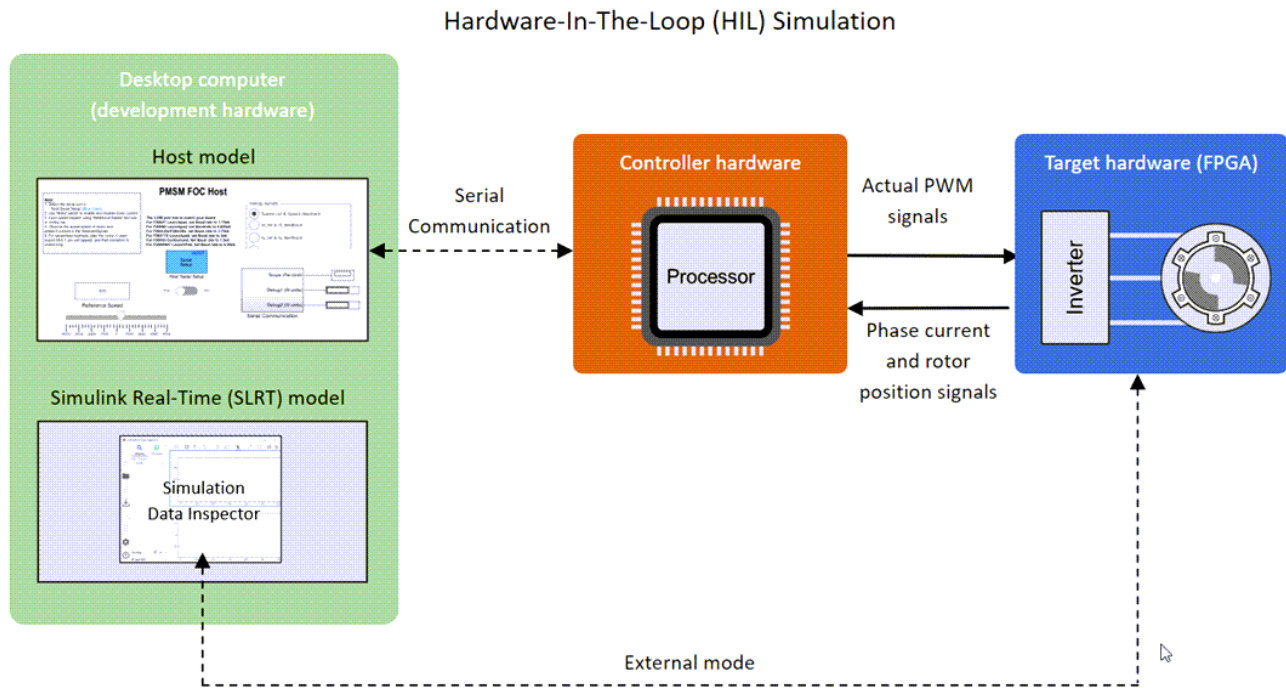
You can also use this setup to effectively test scenarios like hardware failures, motor burn-out, and other faults. To know more about HIL simulation, see "Basics of Hardware-in-the-Loop simulation" (Simscape).

Open MATLAB Project

The example is packaged as a MATLAB® project. Use one of these methods to open the MATLAB project window:

1. Click **Open Example**.

2. Run the command `mcb_foc_hil_start` at the command prompt.

Model

The MATLAB project has a `model` folder that includes the following models:

- `mcb_pmsm_foc_f28379d.slx` - This target model contains the FOC algorithm that you can deploy and run on the controller hardware. The model algorithm applies the `Subcycle Averaging` method for simulation and to run on the controller hardware to validate the controller FOC algorithm. You can use this model to generate the embedded C code for the controller hardware.

Steps:
1. Edit motor and controller parameters.
2. Edit sample times for inverter and motor.
3. Simulate this model and observe results in Simulation Data Inspector
4. Use slrt_ex_pmsm model to generate HDL model and program FPGA with inverter and motor.
6. Use host model to control the embedded processor.

Learn more about this example.

Permanent Magnet Synchronous Motor
Field Oriented Control

Embedded Processor	Inverter and Motor
Feedbacks_sim → Duty Cycles	Duty_Cycles → Feedbacks_sim
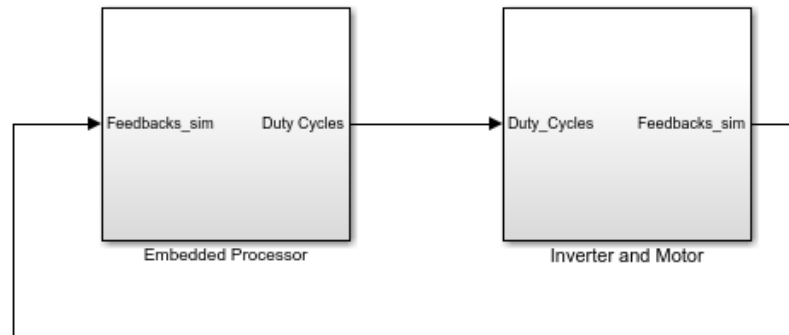
Copyright 2021-2022 The MathWorks, Inc.

- `mcb_host_f28379d.slx` - This is a host model to communicate with the embedded target hardware. Using this model, you can start the simulation and run the motor. You can then generate PWM signals and log data from controller hardware using this model.

PMSM FOC Host

Steps:
1. Select port in Host Serial Setup, Host Serial Receive and Host Serial Transmit
2. Simulate this model
3. Use **Motor** switch to control the motor.
4. Run motor in open loop and observe Ia & Ib ADC counts in scope. Compute average counts for each phase and update offsets to variables inverter.CtSensAOffset and inverter.CtSensBOffset here
5. Use **Reference Speed** edit box to update speed reference to the motor.

Stop — Start
Motor
Open loop — Closed loop

1500
Reference Speed

Debug signals
- ◉ Speed_ref & Speed_feedback
- ○ Id_ref & Id_feedback
- ○ Iq_ref & Iq_feedback
- ○ Ia & Ib
- ○ Ia & Ib ADC counts
- ○ Pos & Ia

No port selected
Host Serial Setup

Scope (Per-Unit)
Debug1 (SI units)
Debug2 (SI units)
Serial Communication
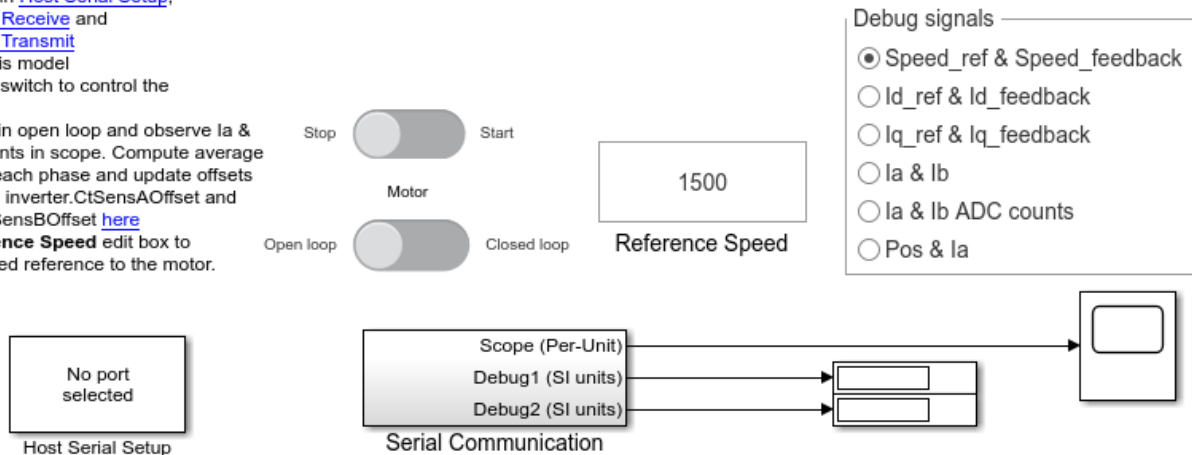
Copyright 2021-2022 The MathWorks, Inc.

Apart from these two models, you will need an FPGA plant model (a combination of inverter and motor equations), which you can deploy and run on the FPGA target hardware, and a Simulink® Real-Time™ application model, which you can use to choose the target hardware and deploy the FPGA plant model algorithms. This example uses the plant model `slrt_ex_pmsm.slx`. The plant model captures the PWM duty cycles provided by the controller and, in turn, runs the inverter and motor equations to generate and send back the actual ADC voltage and position signals to the controller hardware. You can use this plant model to generate the Simulink Real-Time (SLRT) model and the HDL code for the FPGA target hardware. For more information, see "Run Real-Time Simulation of Permanent Magnet Synchronous Motor" (Simulink Real-Time).

Required MathWorks® Products

To simulate model:

• Motor Control Blockset™

To generate code and deploy model:

1. Motor Control Blockset™

2. Simulink Real-Time™

3. Embedded Coder®

4. C2000™ Microcontroller Blockset

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

6. Stateflow®

7. HDL Coder™ (required only if you are changing the plant model)

8. Speedgoat I/O Blockset

Prerequisites

1. Obtain the motor and inverter parameters. The MATLAB project uses default motor and inverter parameters that you can replace with values from either the motor and inverter datasheets or from other sources.

Optionally, if you have the actual motor, you can estimate the parameters for the motor that you want to use with the motor control hardware by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate PMSM Parameters Using Recommended Hardware" on page 4-186. The parameter estimation tool updates the `motorParam` variable (in the MATLAB workspace) with the estimated motor parameters.

2. Update the motor and inverter parameters in the `mcb_pmsm_foc_f28379d_data.m` parameter script associated with the target models available in the MATLAB project. This script automatically opens when you open the MATLAB project. You can also use the Project window to open this script from the `scripts` folder.

3. Click **Run** on the **Editor** tab to run the parameter script.

Simulate Model

Follow these steps to simulate the models included in the project:

4-231

1. Open target model `mcb_pmsm_foc_f28379d.slx` from the `model` folder included in the MATLAB project.

2. Click **Run** on the **Simulation** tab to validate the motor operation. You can increase the simulation speed by reducing the FPGA frequency `f_base` to 2MHz in `pmsm_hil_data.m` script. Observe the simulation results. Make sure to revert `f_base` to 200MHz before deploying to FPFA hardware.

3. Open plant model `slrt_ex_pmsm.slx`. If you want to change the plant model, update `slrt_ex_pmsm.slx`. Simulate `mcb_pmsm_foc_f28379d.slx` to verify the changes in simulation and proceed with HDL workflow to generate the new SLRT model.

Generate Code and Deploy Model to Target Hardware

This section shows you how to generate code and run the FOC and plant model algorithms on the controller and FPGA target hardware.

In addition to the target model, the MATLAB project uses a host model. The host model, which is a user interface to the controller hardware board, runs on the host desktop computer. To use the host model, you need to deploy the target model `mcb_pmsm_foc_f28379d.slx` to the controller hardware board. The host model uses serial communication to command and interface with the `slrt_ex_pmsm_gm.slx` model to run (and control) the inverter and motor equations (HDL code that emulates the actual plant) on the FPGA target hardware.

Generating the SLRT model is optional. You can use the `slrt_ex_pmsm_gm.slx` model to generate an SLRT model to run on the host desktop computer. The SLRT model uses the Simulation Data Inspector to collect and log the debugging data from the plant model HDL code (running on the FPGA hardware in external mode).

Required Hardware

The example supports this hardware configuration: LAUNCHXL-F28379D controller + Speedgoat IO-334 programmable FPGA card.

Prepare Hardware

1. Connect the Speedgoat board to the LAUNCHXL-F28379D controller board as shown in this table.

TI-LaunchXL-28379D Functionality	TI-LaunchXL-28379D Pin Out	IO334-21 Pin Out	IO334-21 Channel Number HDL Coder Workflow	Plant
EPWM1A	J4 40	1	0	S1
EPWM1B	J4 39	2	1	S2
EPWM2A	J4 38	3	2	S3
EPWM2B	J4 37	4	3	S4
EPWM3A	J4 36	5	4	S5
EPWM3B	J4 35	6	5	S6
GND	J2 GND	9	GND	
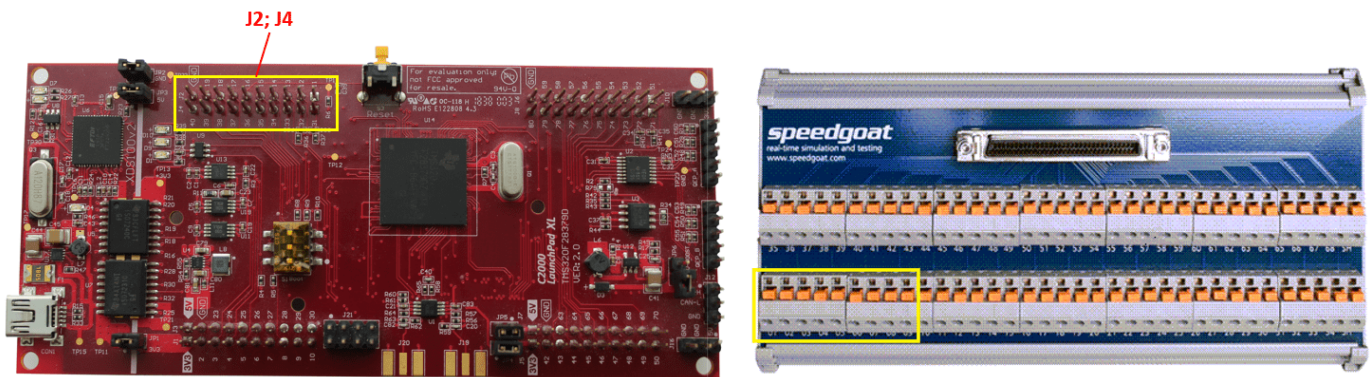
2. For encoder signals, connect the Speedgoat to controller board as shown in this table.

TI-LaunchXL-28379D Functionality	TI-LaunchXL-28379D Pin Out	IO334-21 Pin Out	IO334-21 Channel Number HDL Coder Workflow	Plant
QEPA ENCA	QEPA ENCA	11	9	Encoder
QEPA ENCB	QEPA ENCB	12	10	
QEPA ENCIndex	QEPA ENCI	13	11	
GND	J2 GND	9		
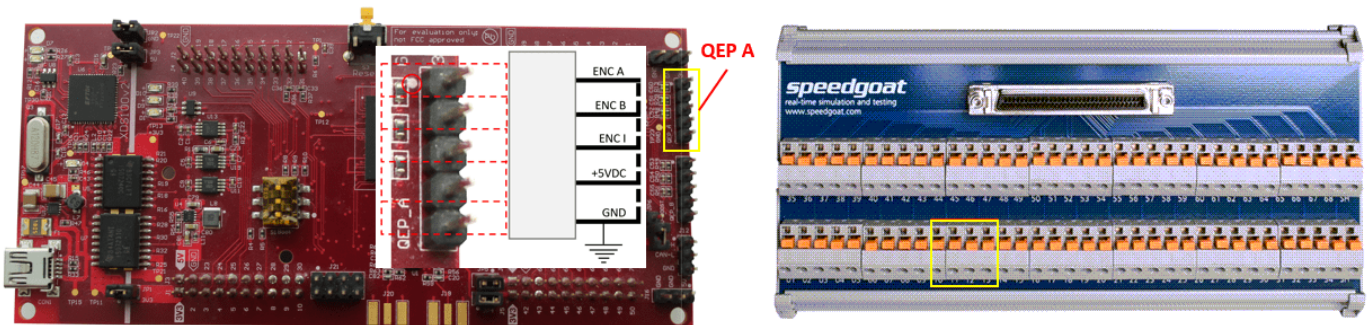
3. For analog signals, connect the Speedgoat to controller as shown in this table.

TI-LaunchXL-28379D Functionality	TI-LaunchXL-28379D Pin Out	IO334 Analog Pin Out (Port ; Wire)	IO334 Analog Channel Number HDL Coder Workflow	Plant
ADCINC2	J3 27	RJ 45 AO 1:4 ; 2	DAC 1	Ia
ADCINB2	J3 28	RJ 45 AO 1:4 ; 6	DAC 2	Ib
GND	J3 GND	RJ 45 AO 1:4 ; 1/3	GND	

For more details on the Speedgoat board, see Speedgoat IO-334 and Speedgoat Connection box.

For connections related to the hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. Open the target model `mcb_pmsm_foc_f28379d.slx`. If you want to change the default hardware configuration settings for this model, see "Model Configuration Parameters" on page 2-2.

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

4-233

4. Check that you have updated the correct motor and inverter parameters in the parameter script `mcb_pmsm_foc_f28379d_data.m`.

5. Click **Build, Deploy & Start** on the **Hardware** tab to deploy the target model `mcb_pmsm_foc_f28379d.slx` to the controller hardware.

6. Open the plant model `slrt_ex_pmsm_gm.slx` and build the model. Program the FPGA using this model and run it for 500 seconds by executing below lines in MATLAB command window.

```
tg = slrealtime;
tg.stop
mdl = 'slrt_ex_pmsm_gm';
tg.load(mdl);
tg.setStopTime(500)
tg.start;
```

NOTE: If you want to update the plant model, update `slrt_ex_pmsm.slx` and follow HDL workflow to generate the SLRT model `slrt_ex_pmsm_gm.slx` and program the FPGA.

7. Click the **host model** hyperlink in `mcb_pmsm_foc_f28379d.slx` target model to open the host model.

You can also use the MATLAB project window to open the host model `mcb_host_f28379d.slx`.

8. Turn the **Stop-Start** slider switch available in the **Simulation Dashboard** area to the **Start** position to allow the model to simulate and run the motor. Run the **Open loop** control first. Select **Ia & Ib ADC counts** from **Debug signals** and observe the ADC counts of the phase current.
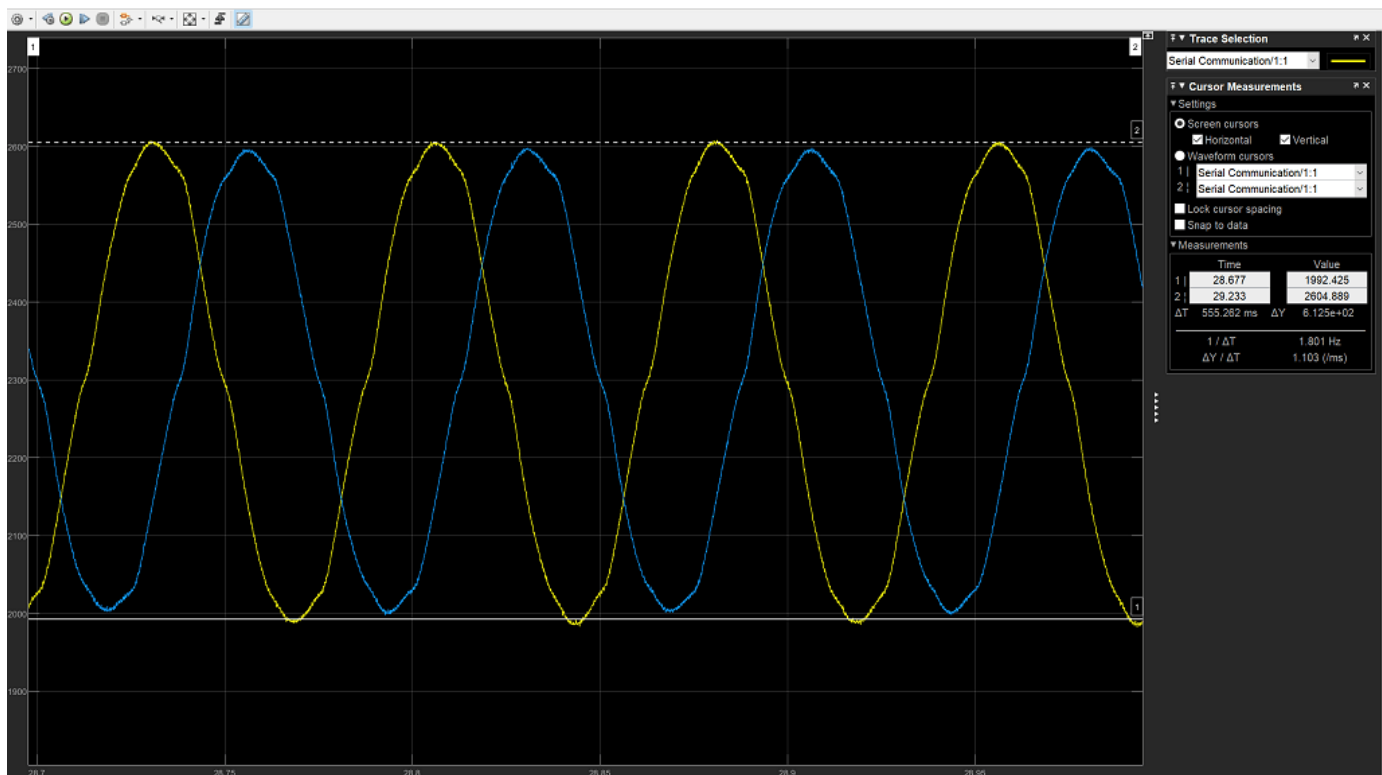
9. Take the average value from peak to peak of the sinusoidal ADC count waveform for both Ia and Ib. This is the current offset. Update this current offset in `mcb_pmsm_foc_f28379d_data.m` for

variables `inverter.CtSensAOffset` and `inverter.CtSensBOffset`. Build and flash the `mcb_pmsm_foc_f28379d.slx` model again to the controller hardware. Continue with the host model by running the motor in a closed loop operation.

During simulation, you can turn the switch to the **Stop** position anytime to immediately stop the motor.

10. In the host model, open the blocks Host Serial Setup, Host Serial Receive, and Host Serial Transmit, and select a **Port**.

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

12. Review the logged signals using the Simulation Data Inspector.
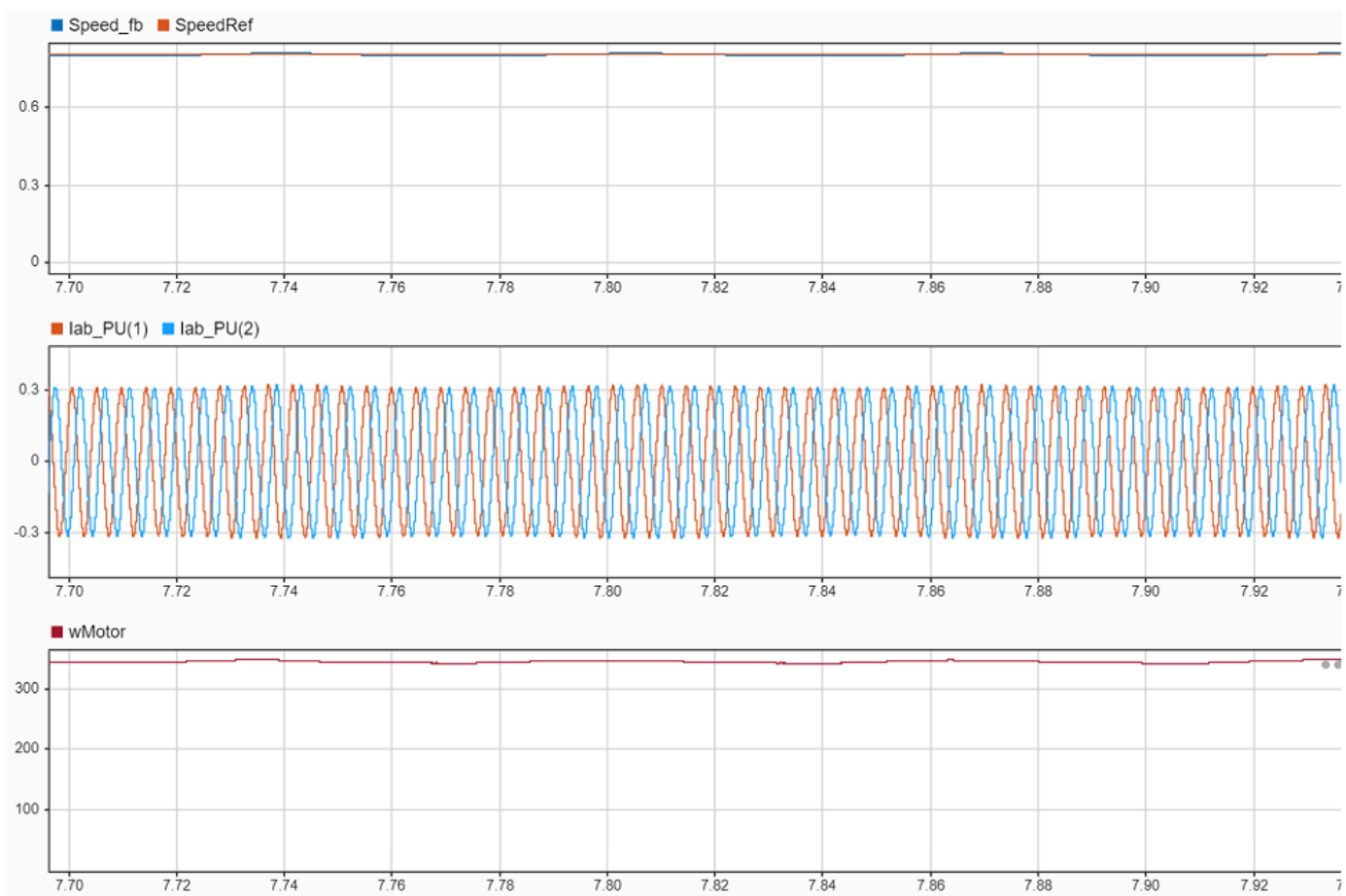
Direct Torque Control of PMSM Using Quadrature Encoder or Sensorless Flux Observer

This example implements direct torque control (DTC) technique to control the speed of a three-phase permanent magnet synchronous motor (PMSM). Direct Torque Control (DTC) is a vector motor control technique that implements motor speed control by directly controlling the flux and torque of the motor. The example algorithm needs motor currents and position feedback from PMSM. It uses space vector pulse-width modulation (DTC-SVPWM) variant of DTC, which uses space vector modulation (SVM) to produce the pulse-width modulation (PWM) duty cycles that are used by the inverter. For more details about the DTC-SVPWM algorithm used in this example, see "Direct Torque Control (DTC)" on page 4-7.

The example enables you to use either quadrature encoder sensor or sensorless flux observer to determine the rotor position. For details about the Flux Observer Simulink® block, see Flux Observer.

Model

The example includes the mcb_pmsm_dtc_f28379d model (target model).

You can use this model for both simulation and code generation. You can also open the Simulink® model using this command at the MATLAB® Command Window.

```
open_system('mcb_pmsm_dtc_f28379d.slx');
```


For details about the supported hardware configuration, see the Required Hardware topic 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®
- C2000™ Microcontroller Blockset
- Fixed-Point Designer™ (only needed for optimized code generation)

Prerequisites

1. Obtain the motor parameters. The Simulink® model uses default parameters 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 PMSM Parameters Using Recommended Hardware" on page 4-186. The parameter estimation tool updates the `motorParam` variable (in the MATLAB® workspace) with the estimated motor parameters.

2. Update motor parameters. If you obtain the motor parameters from the datasheet or from other sources, update the motor and inverter parameters in the model initialization script associated with the Simulink® model. For instructions, see "Estimate Control Gains and Use Utility Functions" 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 target model included with this example.

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

3. Click **Data Inspector** in the **Review Results** section to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section shows how to generate code and run the DTC 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. Before you can run the host model on the host computer, 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

The example supports this hardware configuration. You can also use the target model name to open the model from the MATLAB® command prompt.

LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305 or BOOSTXL-3PHGANINV) inverter: mcb_pmsm_dtc_f28379d

Note: When using the BOOSTXL-3PHGANINV inverter, ensure that you have proper insulation between the bottom layer of BOOSTXL-3PHGANINV and the LAUNCHXL board.

For connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

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

2. Complete the hardware connections.

3. 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.

4. Set the **Position Feedback** radio button on the target model to **Quadrature Encoder** if you are using a quadrature encoder sensor to read the rotor position. Select **Flux Observer** if you want to use sensorless position estimation using a flux observer.

5. Compute the quadrature encoder index offset value and update it in the `pmsm.PositionOffset` variable available in the model initialization script associated with the target model. For instructions, see "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79.

Note: Skip step 4 if you are using the sensorless flux observer for position estimation.

6. The model by default computes the ADC offset values for phase current measurement. To disable this functionality, update the value of the `inverter.ADCOffsetCalibEnable` variable in the model initialization script to `0`.

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 4-10.

7. Load a sample program to CPU2 of the LAUNCHXL-F28379D board. For example, load the program that operates the CPU2 blue LED by using GPIO31 (c28379D_cpu2_blink.slx). This ensures 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. Check if the MATLAB base workspace shows the variables from the deployed target model.

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.

```
open_system('mcb_pmsm_dtc_host_f28379d.slx');
```

PMSM Direct Torque Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_dtc_f28379d

2. The variables from the target model are loaded to the base workspace.

Steps:
1. Set the serial port to **target.comport** in the init script of the target model.
2. Simulate this model
3. Use **Start / Stop Motor** switch to control the motor.
4. Enter Reference speed in RPM (limit to base speed) using edit box.
5. Use the radio button to select the pair of debug signals. Observe these signals in
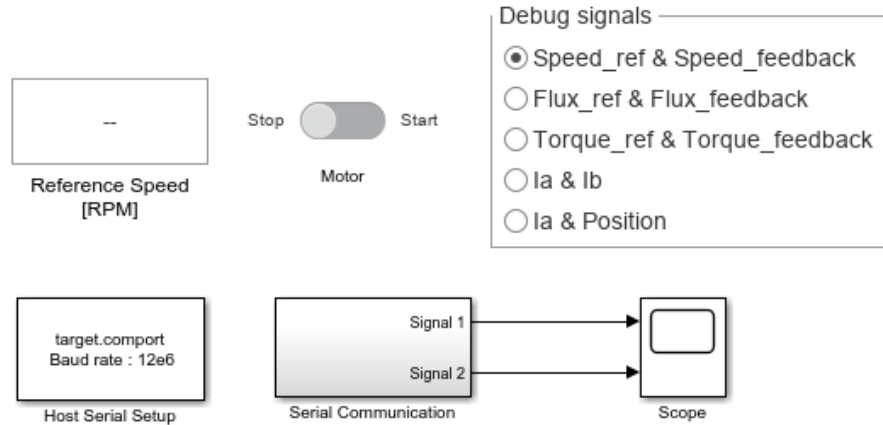
Debug signals
- ⦿ Speed_ref & Speed_feedback
- ◯ Flux_ref & Flux_feedback
- ◯ Torque_ref & Torque_feedback
- ◯ Ia & Ib
- ◯ Ia & Position

Reference Speed [RPM]

Stop ⬤ Start

Motor

target.comport
Baud rate : 12e6

Host Serial Setup

Signal 1
Signal 2

Serial Communication

Scope

Copyright 2021-2022 The MathWorks, Inc.

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

10. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available 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. Update the Reference Speed value (in RPM) in the host model.

14. Use the **Debug signals** section of the host model to select the debug signals that you want to monitor:

- **Speed_ref & Speed_feedback** — Display the speed reference and speed feedback signals in the scope.

- **Flux_ref & Flux_feedback** — Display the flux reference and flux feedback signals in the scope.

- **Torque_ref & Torque_feedback** — Display the torque reference and torque feedback signals in the scope.

- **Ia & Ib** — Display the phase-|a| and phase-|b| currents in the scope.

- **Ia & Position** — Display the phase-|a| current and rotor position signals in the scope.

15. Use the time scope available in the host model to monitor the selected debug signals.

Determine Power Losses and THD for PWM Methods

This example calculates the inverter power loss and total harmonic distortion (THD) in motor current for different pulse-width modulation (PWM) methods. The example uses field-oriented control (FOC) algorithm that runs a permanent-magnet synchronous motor (PMSM) in speed control mode as a reference. The example only supports simulation.

The example model simulates the PWM methods sequentially in this order:

1. SPWM — sinusoidal PWM

2. SVM — space vector modulation

3. 60 DPWM — 60 degree discontinuous PWM

4. 60 DPWM (+30 degree shift) — +30 degree shift from 60 DPWM

5. 60 DPWM (-30 degree shift) — -30 degree shift from 60 DPWM

6. 30 DPWM — 30 degree discontinuous PWM

7. 120 DPWM — Positive DC component

8. 120 DPWM — Negative DC component

The model simulates each method for a fixed time period before changing the PWM method. You can configure this time period by updating the variable `Ts_MethodDuration` available in the model initialization script associated with the example model. For instructions to locate the model initialization script, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

After the simulation completes, the script `mcb_InverterLossTHDExtract.m` uses the simulation data available in the MATLAB® workspace to generate the power loss in the inverter switches and motor current THD plots for the different PWM methods.

Note: The inverter's MOSFET used in this example are configured using the datasheet of MOSFETs available in the Texas Instruments™ BOOSTXL-DRV8305 board.

Model

The example includes the mcb_inverter_powerloss model.

You can use this model only for simulation. You can also open the Simulink® model using this command at the MATLAB Command Window.

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

PMSM Field Oriented Control
Losses in Inverter switches and Total Harmonic Distortion - Comparison between different PWM strategies
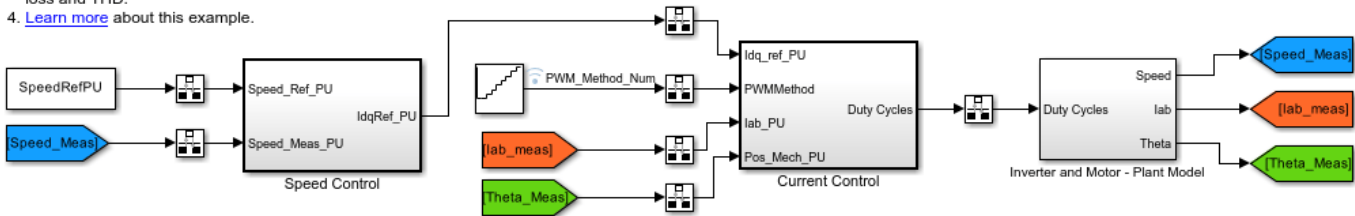
Copyright 2021 The MathWorks, Inc.

Required MathWorks® Products

- Motor Control Blockset™

- Simscape™ Electrical™

Prerequisites

1. Obtain the motor parameters. The Simulink® model uses default parameters 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 PMSM Parameters Using Recommended Hardware" on page 4-186. The parameter estimation tool updates the `motorParam` variable (in the MATLAB® workspace) with the estimated motor parameters.

2. Update the motor and inverter parameters in the model initialization script associated with the Simulink® model. For instructions, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

Update the motor parameters in the PMSM block available in the `mcb_inverter_powerloss/lnverter and Motor - Plant Model/Motor` subsystem of the example model. In addition, update the parameters of the MOSFET available in `mcb_inverter_powerloss/lnverter and Motor - Plant Model/lnverter/Switch1` (`mcb_ref_switchmodel`) of the example model.

3. Configure the simulation time period for the PWM methods using the `Ts_MethodDuration` variable available in the model initialization script associated with the example model. Ensure that this time period is long enough for the speed control loop to stabilize and enable the motor to reach a steady speed state.

Simulate Model

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

1. Open the model included with this example.

2. Click the **Simulate** hyperlink (step 2) available in the **Explore More** section of the example model to run the simulation. Alternatively, you can simulate the example model by clicking **Run** on the **Simulation** tab, however, ensure that you do not interrupt the simulation process.

Note: The simulation takes few minutes to complete. If you interrupt the simulation process, the example does not compute and plot the power loss and THD information.

After the simulation completes, the model stores the simulation data in the MATLAB workspace that is used by the `mcb_InverterLossTHDExtract.m` script to extract the power loss and THD information and plot them for the different PWM methods. After generating the plots, the script saves the power loss and THD information in the `LossTHDdata` variable (in the MATLAB workspace).

3. You can also click the **function** hyperlink available in the **Explore more** section of the model, to view the `mcb_InverterLossTHDExtract.m` script that computes the power loss and THD information and generates the plots.

You can use this example to analyze and determine a suitable PWM technique for your motor control application.

Note:

- The example algorithm averages the power loss and THD information of a PWM method over the last electrical cycle (among a series of cycles) corresponding to that PWM method.

- When speeds are greater than 0.75 per-unit, the controllers may get saturated for sinusoidal PWM (SPWM) method. Therefore, in such cases, the results of SPWM will not be comparable with other methods.

These are the examples of the plots generated by the model:

Total Inverter Loss (Averaged over an electrical cycle)

THD of Phase Current - Phase A (Averaged over an electrical cycle)

SPWM:Sinusoidal PWM
SVM:Space Vector Modulation
60 DPWM:60 degree Discontinuous PWM
60 DPWM (+30 degree shift)
60 DPWM (-30 degree shift)
30 DPWM:30 degree Discontinuous PWM
120 DPWM:Positive DC Component
120 DPWM:Negative DC Component
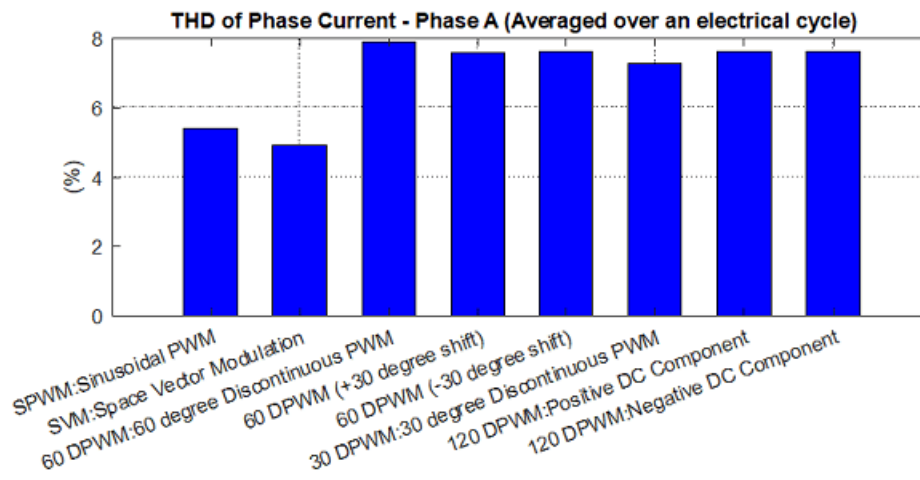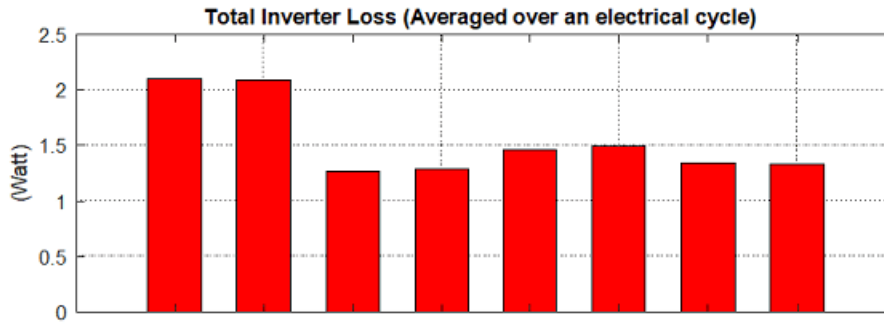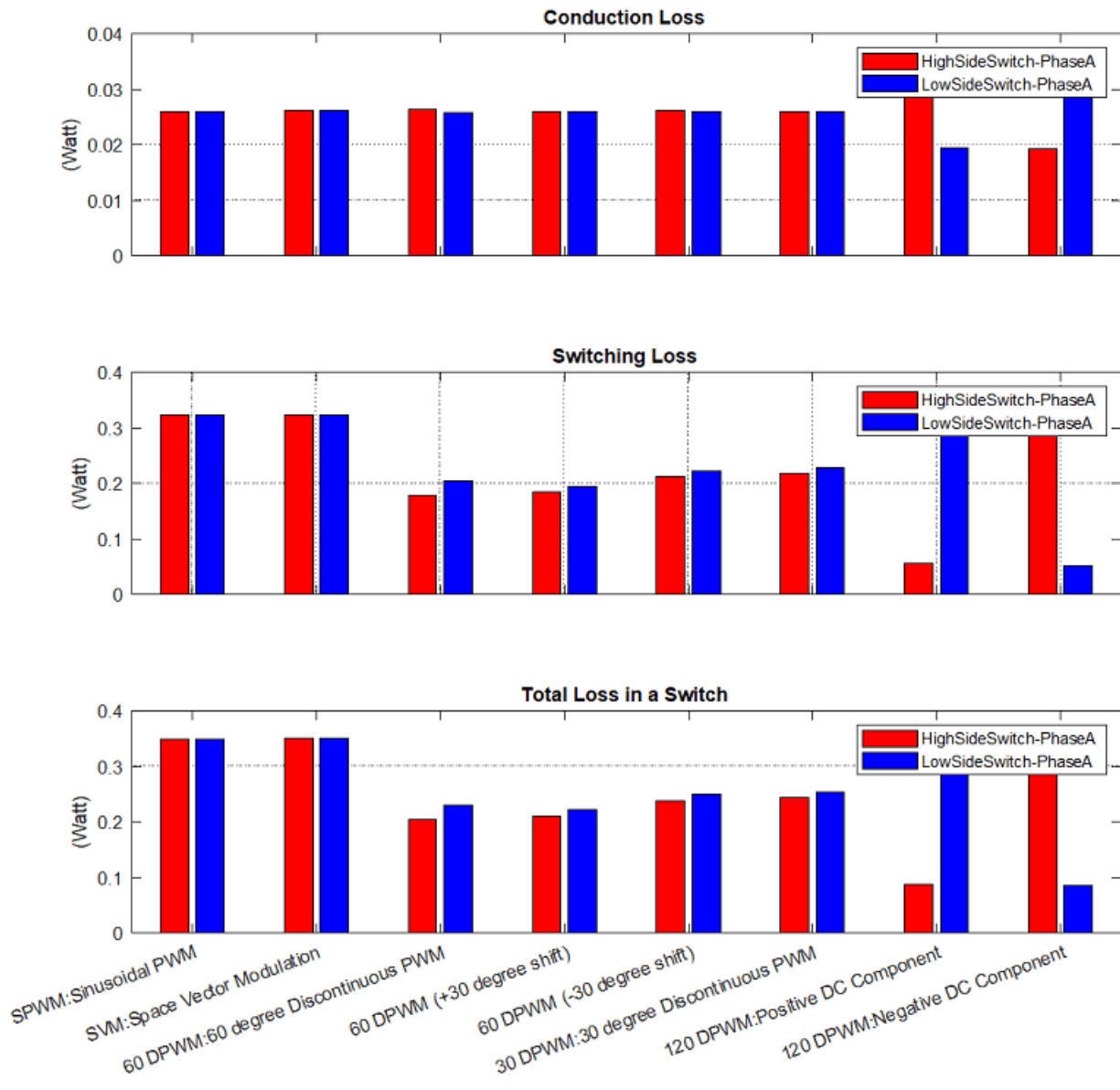
Run Field Oriented Control of PMSM Using Model Predictive Control

This example uses Model Predictive Control (MPC) to control the speed of a three-phase permanent magnet synchronous motor (PMSM). MPC is a control technique that tunes and optimizes the inputs to a control system to minimize the error in the predicted system output and achieve the reference control objective over a period of time. This technique involves solving the objective function and finding an optimal input sequence at every sample time (T_s). After each time step, the current state of the plant is considered as the initial state and the above process is repeated.
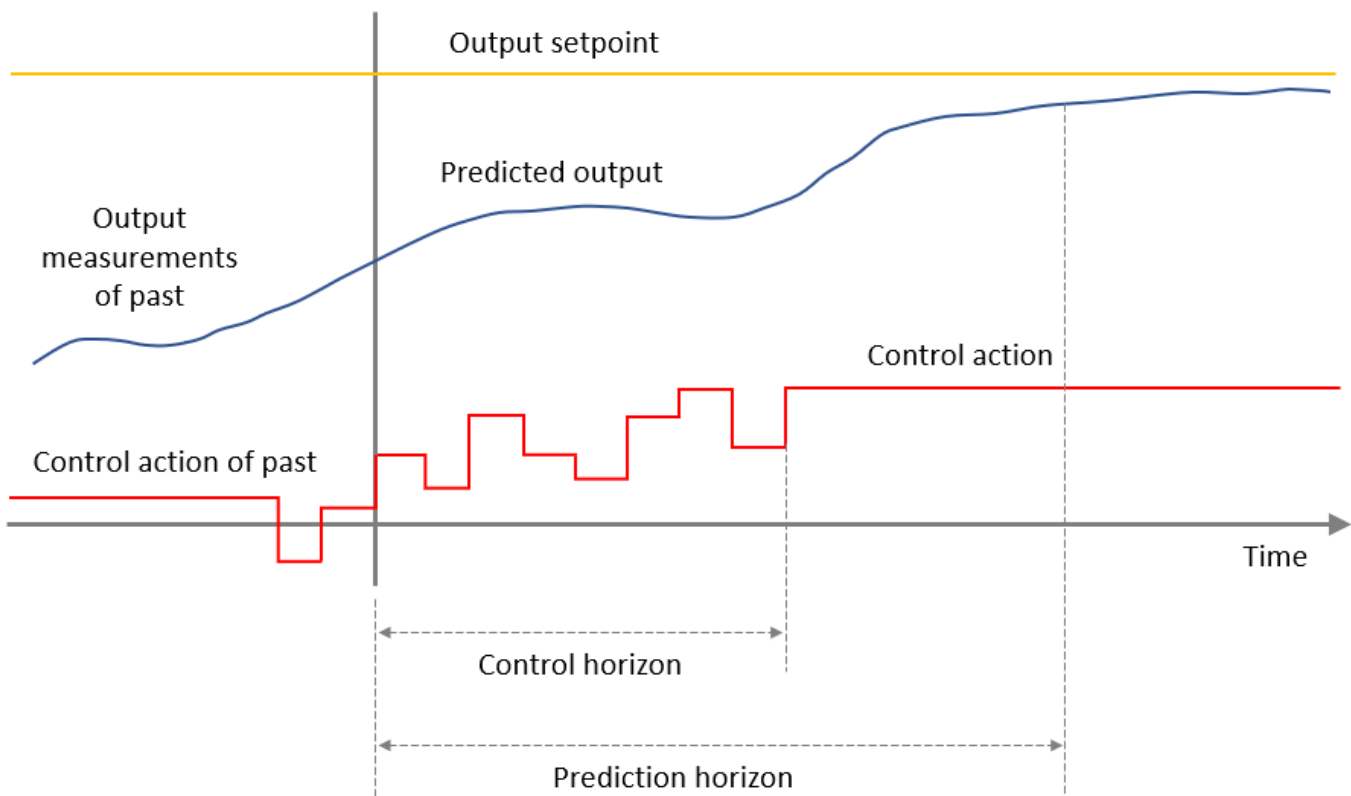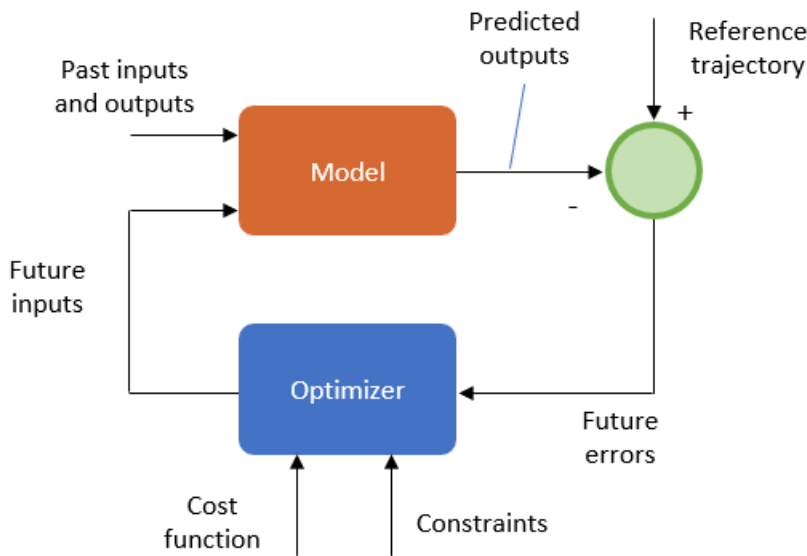
The optimizer provides the optimal inputs to the model based on solving the objective function under specific bounds and constraints. During Prediction step, the future response of a plant is predicted with the help of a dynamic discrete-time model up to Np sampling intervals, which is called the prediction horizon. During Optimization step, the objective function is solved to obtain the optimal control inputs up to Nc sampling intervals, which is called control horizon for the predicted response. Control horizon remains less than or equal to the prediction horizon.
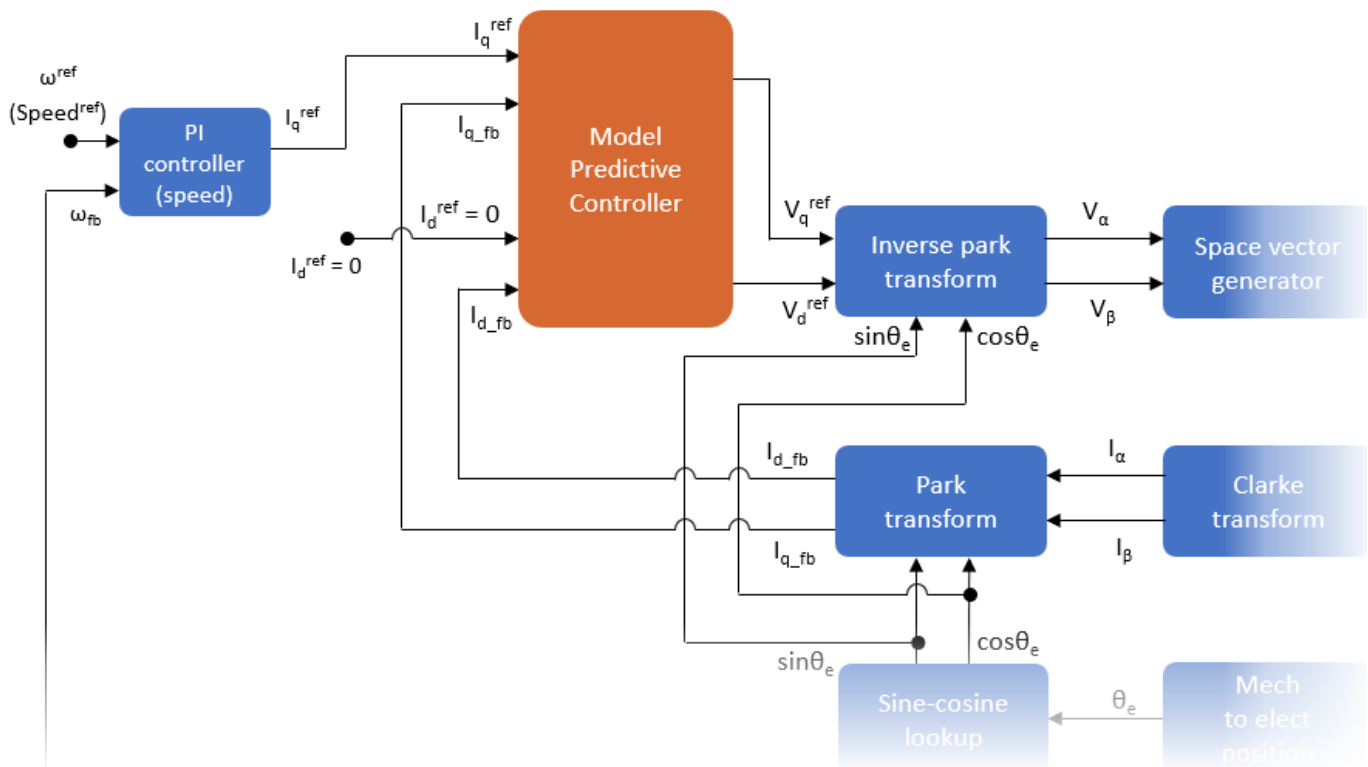
The example uses an MPC controller as a current controller (in a field-oriented control or FOC algorithm) to optimize the I_d and I_q currents and change the d-axis and q-axis controller voltage outputs so that they meet the reference control objectives over a period of time.

The objective function is derived as a linear sum of these:

```
[W1 * (error in output)] + [W2 * (rate of change of input)] + [W3 *
(error in input)]
```

where, W1, W2, and W3 are the weightages.

The example uses the model initialization script to define these weightage (or weights) of these three parameters.

1. Inputs: $[I_d = 0, I_q = 0]$

2. Rate of change of input: $[I_d = 0.01, I_q = 0.01]$

3. Measured Outputs: $[V_d = 1, V_q = 1]$

Therefore, by default, the example gives maximum weightage to the output variables parameter (corresponding to V_d and V_q voltages) when calculating the error in the predicted output. You can change the weightage values for error computation using the model initialization script available in the example.

The example also operates the MPC inputs (I_d and I_q) and the MPC outputs (V_d and V_q) under the following lower and upper bounds:

- Inputs

    $-1 \le I_d \le 1$

    $-1 \le I_q \le 1$

- Measured outputs

    $-0.1 \le V_d \le 0.1$

    $-1 \le V_q \le 1$

Note: The rate of change of input does not have any lower and upper bounds.

To retain linearity of the constraints, you can consider polytopic approximations. An acceptable trade-off between the accuracy and number of constraints can be acheived by approximating the feasible region using a hexagon. Because the direct component of the stator current I_d is almost always very close to zero, except during flux weakening operation when it takes negative values, you can consider the constraint I_d is less that or equal to 0, to reduce the number of constraints.

The following image shows the pictorial representation of the contraints for the MPC output voltages (V_d and V_q, with circle approximation having 6 faces), and MPC input currents (I_d and I_q, with half-circle approximation having 4 faces). You can generate these plots by using the MATLAB command `mcb_getMPCObject(pmsm,PU_System,Ts_current,T_pwm,1)`.
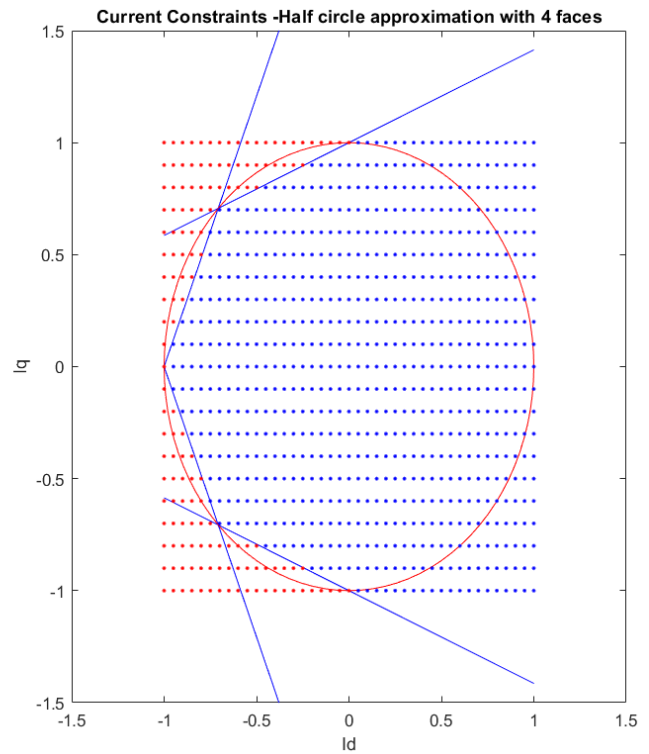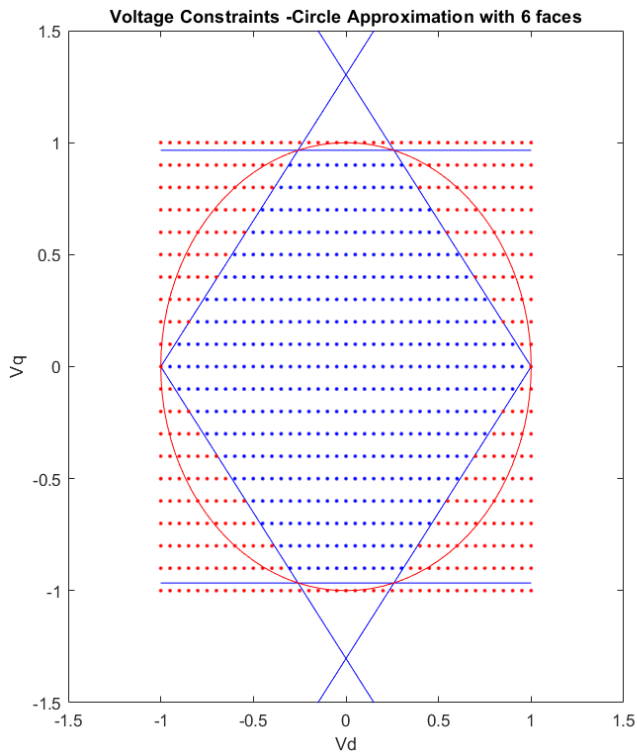
Note: The sample time (T_s) used in the model intitialization script of this example is based on tests on the particular hardware. You can change the sample time for a differet kind of hardware, which will in turn impact the MPC operation.

For more information about MPC, see "What is Model Predictive Control?" (Model Predictive Control Toolbox).

Models

The example includes the model mcb_pmsm_foc_mpc_qep_f28379d

You can use these models for both simulation and code generation. You can also use the open_system command to open the Simulink® models. For example, use this command for a F28379d based controller.

```
open_system('mcb_pmsm_foc_mpc_qep_f28379d.slx');
```

Permanent Magnet Synchronous Motor Field Oriented Control using Model Predictive Controller
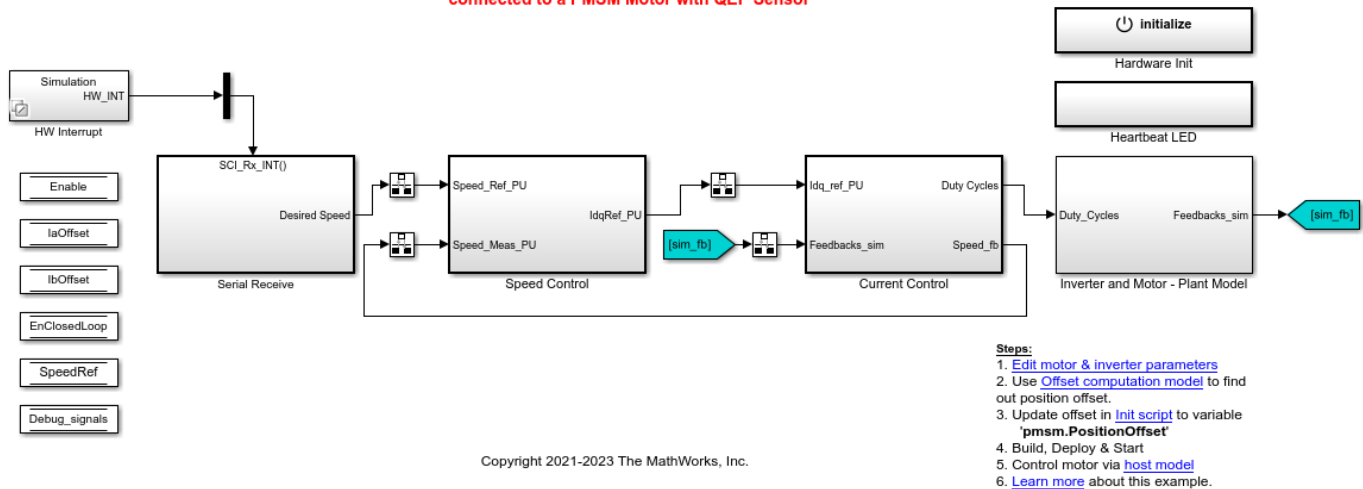
For the model names that you can use for different hardware configurations, see the Required Hardware topic in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Model Predictive Control Toolbox™

To generate code and deploy model:

- Motor Control Blockset™
- Model Predictive Control Toolbox™
- Embedded Coder®
- C2000™ Microcontroller Blockset

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 PMSM Parameters Using Recommended Hardware" on page 4-186.

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 and Use Utility Functions" on page 3-2.

4-249

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 this hardware configuration. You can also use the target model name to open the model for the corresponding hardware configuration, from the MATLAB® command prompt.

*   LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_pmsm_foc_mpc_qep_f28379d

For connections related to the preceding hardware configurations, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

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 4-10.

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 4-79.

NOTE: Verify the number of slits available in the quadrature encoder sensor attached to your motor. Check and update the variable `pmsm.QEPSlits` available in the model initialization script. This variable corresponds to the **Encoder slits** parameter of the quadrature encoder block. For more details about the **Encoder slits** and **Encoder counts per slit** parameters, see Quadrature Decoder.

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_f28379d.slx');
```


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

9. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

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. Use the **Debug signals** section to select the debug signals that you want to monitor. Observe the debug signals from the RX subsystem, in the Time Scope of the host model.

References

- G. Cimini, D. Bernardini, A. Bemporad and S. Levijoki, "Online model predictive torque control for Permanent Magnet Synchronous Motors," 2015 IEEE International Conference on Industrial Technology (ICIT), 2015, pp. 2308-2313, doi: 10.1109/ICIT.2015.7125438.

- S. Chai, L. Wang and E. Rogers, "Cascade model predictive control of a PMSM with periodic disturbance rejection," 2011 Australian Control Conference, 2011, pp. 309-314.

Commutation of SRM Using Sensor Feedback

This example implements a commutation system to control the speed of a three-phase 12/8 switched reluctance motor (SRM).

It uses the switching sequences generated by the SRM Commutation block to control switch the motor phases ON and OFF, and therefore, run the motor while controlling the motor speed. For more details about this block, see SRM Commutation.

The example uses the following values to generate a switching (or commutation) sequence for each phase available in the motor.

- Electrical position signals with respect to a phase of SRM.
- Position range for all phases during which they should be activated (also known as dwell angle).

Each switching sequence (for example, ctr(1) signal in the following image, which forms a pulse train) can be used to control (turn On or Off) the corresponding motor phase. Each pulse in a switching sequence represents the activation period of the phase, which is defined by turn-on (θeOn) turn-off ($\theta eOff$) angles as shown below:
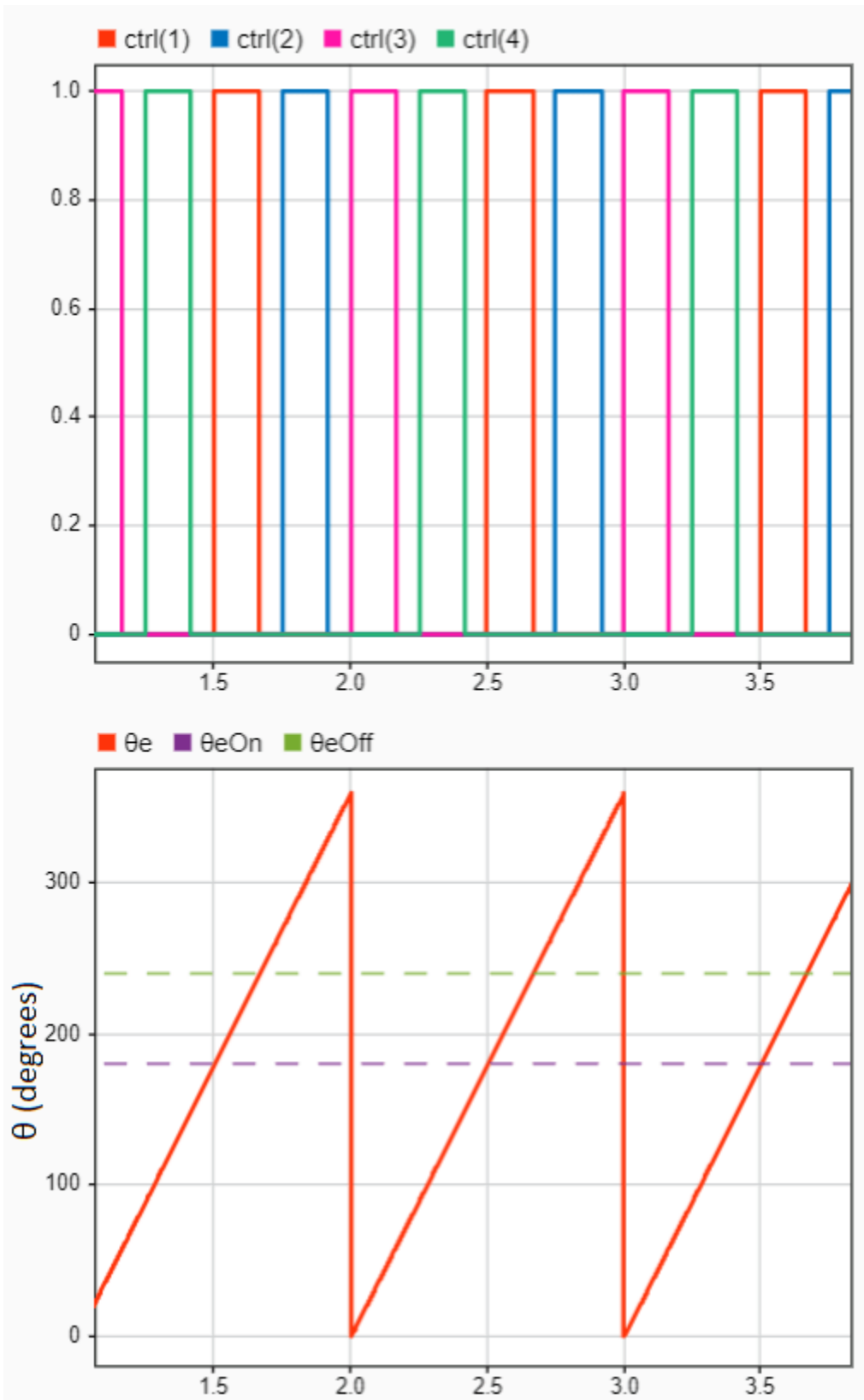
The commutation system needs real-time motor position feedback, which is obtained from a quadrature encoder sensor.

The example utilizes Mechanical to Electrical Position block, which uses the mechanical position feedback and the number of rotor poles available in SRM to compute the motor electrical position. It computes electrical position with respect to each phase (Theta_e), which lies between 0 and 2π radians (0 and 360 degrees or 0 and 1 per unit).

The example uses this feedback along with the commutation logic to drive and control the speed of the rotor.

You can configure the example to use either per-unit and SI unit computation. When the controller uses per-unit computation, the example supports both fixed point and floating-point datatypes. However, when using SI unit, the example only supports floating-point datatype.

Model

The example includes the model mcb_srm_spdctrl_f28379d.

You can use these models for both simulation and code generation. To open a Simulink® model, you can also use the open_system command at the MATLAB command prompt. For example, use this command for the F28379D based controller:

```
open_system('mcb_srm_spdctrl_f28379d.slx');
```
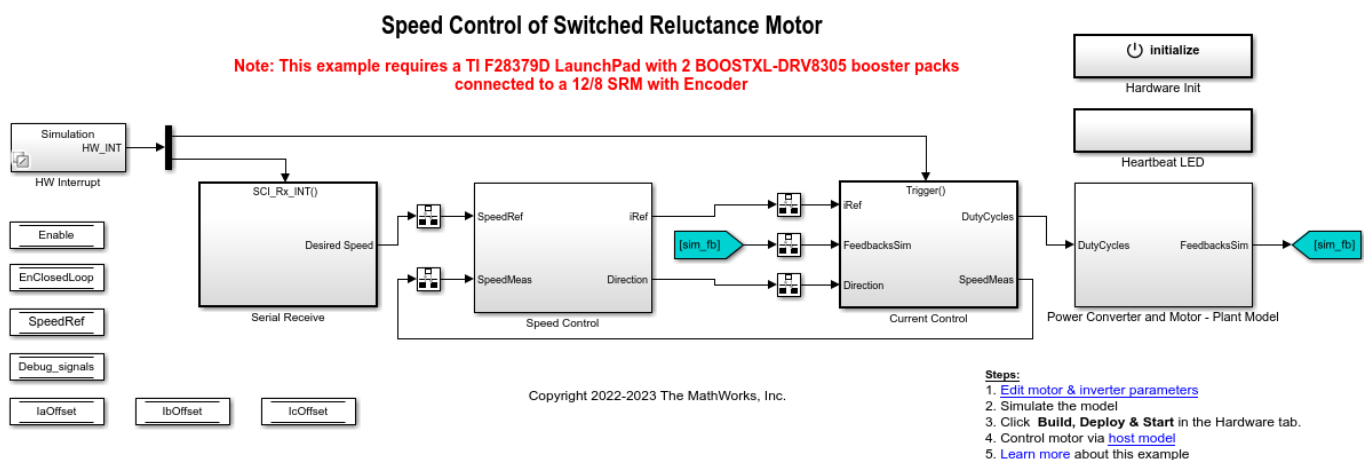

For details about the supported hardware configuration, see Required Hardware in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Simscape™ Electrical™

To generate code and deploy model:

- Motor Control Blockset™
- Fixed-Point Designer™ (only needed for optimized code generation)
- Embedded Coder®
- C2000™ Microcontroller Blockset

Prerequisites

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

Update SRM parameters in the model initialization script associated with the Simulink model. For instructions, see "Estimate Control Gains and Use Utility Functions" on page 3-2

In addition, update the following parameters in the model initialization script:

- PWM_frequency — Enter the switching frequency of PWM. Note that you may need to tune controller gains again if you change this value.

- datatype

    - single — Use this value for floating point code generation. In this mode, the example supports both per-unit and SI unit computation.

    - fixdt(1,32,17) — Use this value for fixed point code generation. In this mode, the example only supports per-unit computation.

- controllerunit

    □ 1 — Use this value to enable SI unit computation.

    □ 0 — Use this value to enable per-unit computation.

- Kp and Ki — Enter the PI controller gains. Note that the example does not automatically compute these values. Fine tune these gains according to the hardware setup.

- thetaON — Enter the motor phase turn-on electrical position at which the switching sequence turns 0 to 1 and energizes the corresponding phase.

- thetaOFF — Enter the motor phase turn-off electrical position at which the switching sequence turns 1 to 0 and de-energizes the corresponding phase.

Simulate Model

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

1. Open the target 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 shows you how to generate code and run the control 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

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB® command prompt.

• LAUNCHXL-F28379D controller + 2 BOOSTXL-DRV8305 inverters: mcb_srm_spdctrl_f28379d

The example uses the ADC current sensors available in the inverter boards to measure the 3 phase motor currents needed for controlling SRM.

Hardware Connections

The example requires 2 BOOSTXL-DRV8305 inverters connected in the following configuration.

For more information about hardware connections for SRM, see "Instructions for Switched Reluctance Motor (SRM) Setup" on page 7-11.

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.

4. 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.

5. 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.

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

7. Observe and verify the variables populated by the target model in the base workspace.

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. Use this command for a F28379D based controller.

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

Switched Reluctance Motor Control Host

Prerequisites:
1. Deploy the target model to the hardware
   mcb_srm_spdctrl_f28379d
   mcb_srm_dwellctrl_f28379d

2. You should see and verify the variables from the target model in the base workspace.

Steps:
1. Set the serial port to **target.comport** in the init script of the target model.
2. Use **Motor** switch to control the motor.
3. Enter Reference speed in RPM using edit box.
4. Start the motor under no load.
5. Recommended to have lower reference speeds during startup and later on increase the speed to required values.
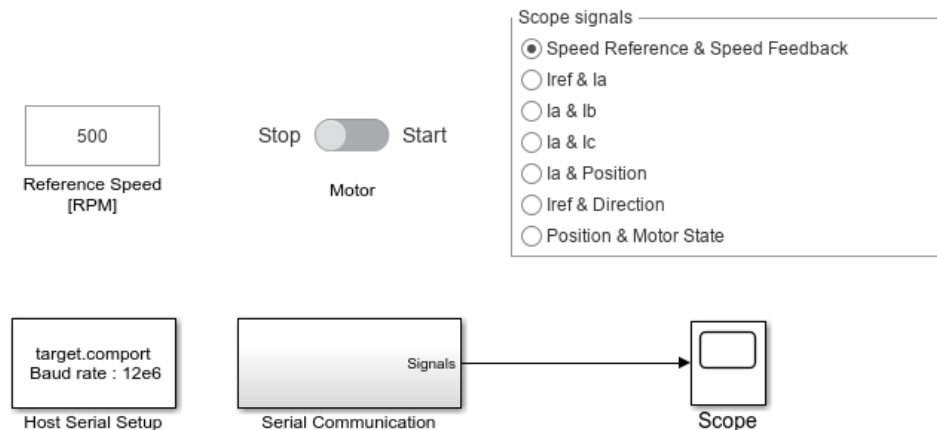
Scope signals
- ◉ Speed Reference & Speed Feedback
- ○ Iref & Ia
- ○ Ia & Ib
- ○ Ia & Ic
- ○ Ia & Position
- ○ Iref & Direction
- ○ Position & Motor State

500

Reference Speed [RPM]

Stop ○ Start

Motor

target.comport
Baud rate : 12e6

Host Serial Setup

Signals

Serial Communication

Scope

Copyright 2022 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 model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the Port parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

10. Update the reference speed value in the **Reference Speed (RPM)** field in the host model. We recommend that you enter lower speeds initially during motor start up and gradually increase the speed later.

11. In the host model, select the debug signals that you want to monitor.

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

13. Ensure that the motor is in no-load condition. Change the position of the **Motor** switch to **Start**, to start running the motor.

If the motor does not align or if it fails to start running in open-loop mode, increase the values of the start-up parameters listed in the **Motor Start Up Parameters** section of the model initialization script.

14. The example performs the following procedures:

- Calibrates the quadrature encoder sensor (which includes alignment of the rotor with stator phase a).

- Runs SRM in closed loop according to Reference Speed (RPM).

15. Observe the debug signals received from target in the Scope available in the host model.

FOC of PMSM Using FPGA-Based Motor Control Development Kit

This example shows how to use a Field-Oriented Control (FOC) algorithm for a Permanent Magnet Synchronous Motor (PMSM) by using blocks from the Motor Control Blockset™ on an FPGA device (Trenz Electronic™ Motor Control Development Kit TE0820).

The example lets you test the control algorithm by using a closed-loop system simulation. You can then generate HDL code for the control algorithm. The example includes a MATLAB® project that provides the models and the reference design. Use the project with the HDL Workflow Advisor for bit stream generation and for running an external mode model to interact with the processor.

You can apply the techniques shown in this example to partition and deploy the controller algorithms. This image shows the generalized workflow from model simulation to deployment of the controller algorithm onto an embedded hardware board.
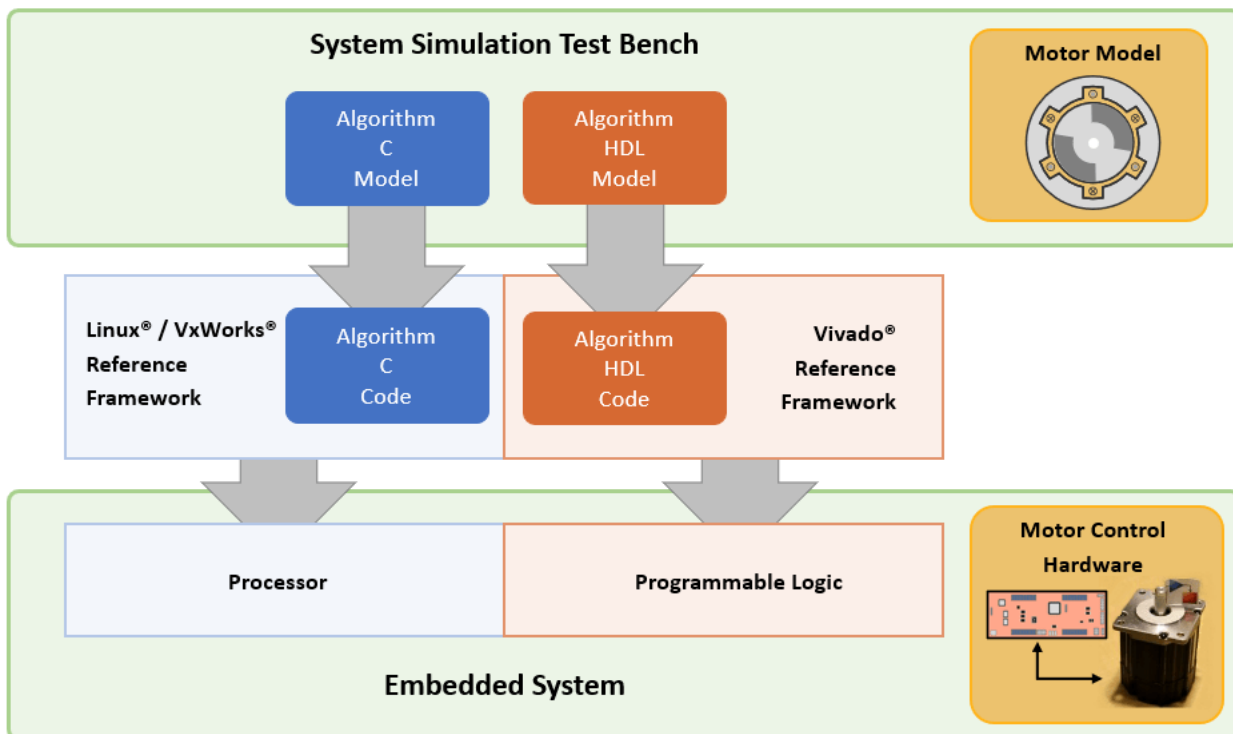
The example supports both quadrature encoder sensor-based and sensorless flux observer-based closed-loop motor control during simulation and hardware deployment.

Simulate a system test bench to gain insight into the behavior of the controller algorithm design. Then explore the design to see how the algorithm is partitioned. The high rate portion of the algorithm is partitioned into a model that is configured for HDL code generation. The low rate portion of the algorithm is partitioned into a model that is configured for C code generation. Generate C and HDL code from these models and learn how you can integrate this code into your design.

After exploring C and HDL code (model) for the algorithm, use automated deployment of code into the reference frameworks for the processor and programmable logic. Then, execute a test on the deployed application, log the results, and compare them with the simulation results.

Because this example deploys a bitstream and ARM® executable to a Xilinx® Zynq® hardware, you must set up the Xilinx Zynq hardware board before starting the example. To ensure that you have setup the environment correctly, follow and complete the "Getting Started with Targeting Xilinx Zynq Platform" (HDL Coder) example using your hardware configuration before starting this example. The figure shows the **Hardware Board** selection used in this demo.
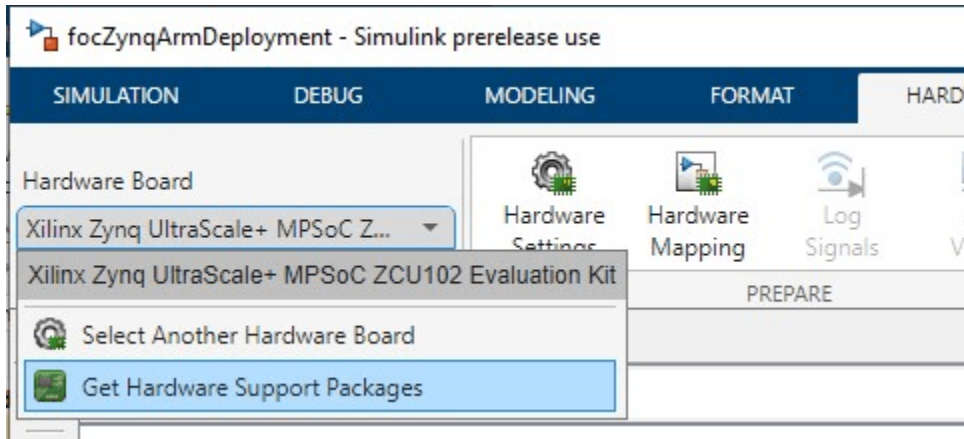
Required Products

To run this example, these products are required:

- MATLAB®

- Simulink®

- Embedded Coder®

- Embedded Coder Support Package for Xilinx Zynq Platform

- HDL Coder™

- HDL Coder Support Package for Xilinx Zynq Platform

- Motor Control Blockset

- Simscape™ Electrical™

Required Hardware

Trenz Electronic Motor Control Development Kit TE0820 (Xilinx Zync UltraScale+ and ZU2CG-1E MPSoC Module) with these components:

     ◦ TEB0707-02 carrier board

     ◦ CR00140-02 driver module

Create Working Copy of MATLAB Project

To begin, create a working copy of the MATLAB project.

1. Use the following command to open the MATLAB project:

```
mcb_foc_fpga_demo_start
```

The `+task` folder in the project has the scripts to automate different activities. The following image shows the summary of activities and dependencies listed inside a MATLAB project. For more information about MATLAB projects, see "What Are Projects?"

Note:

- The model focZynqC (or focZynqCFO) automatically computes the ADC (or current) offset values after deployment. To disable this functionality (enabled by default), update the value 0 to the variable `paramAdcEnablecalibration` in the model initialization script or data dictionary. If you disable this functionality, you can compute the ADC offset manually and update it in the `paramAdcCountAtZeroAmpere` variable available in the model initialization script.

  ○ Open the block parameters dialog box for the **Input_Source** block.

  ○ Set the **Value** parameter to `inputSourceEnum.Interactive UserInput`.
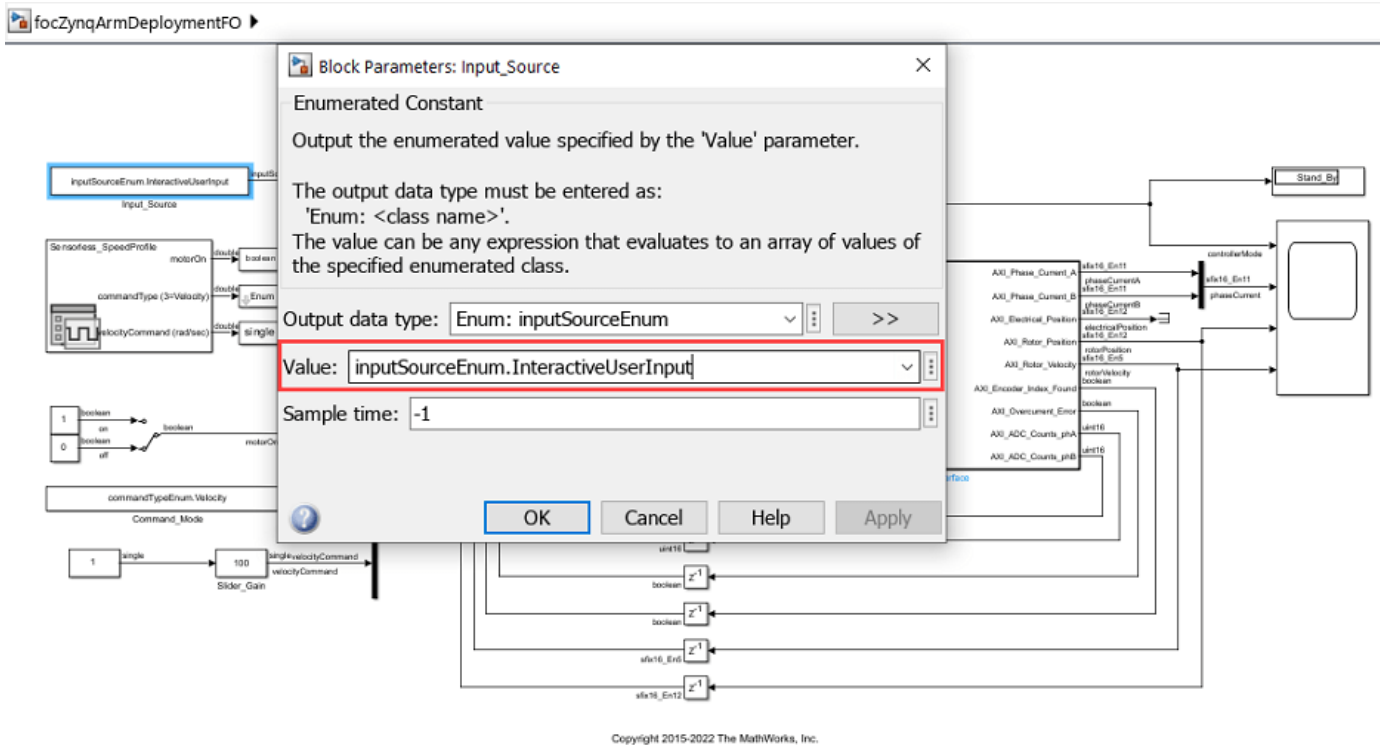
◦ Measure the signals highlighted in the following figure. The measured values indicate the ADC offsets. Update these values in the variable `paramAdcCountAtZeroAmpere` available in the model initialization script.
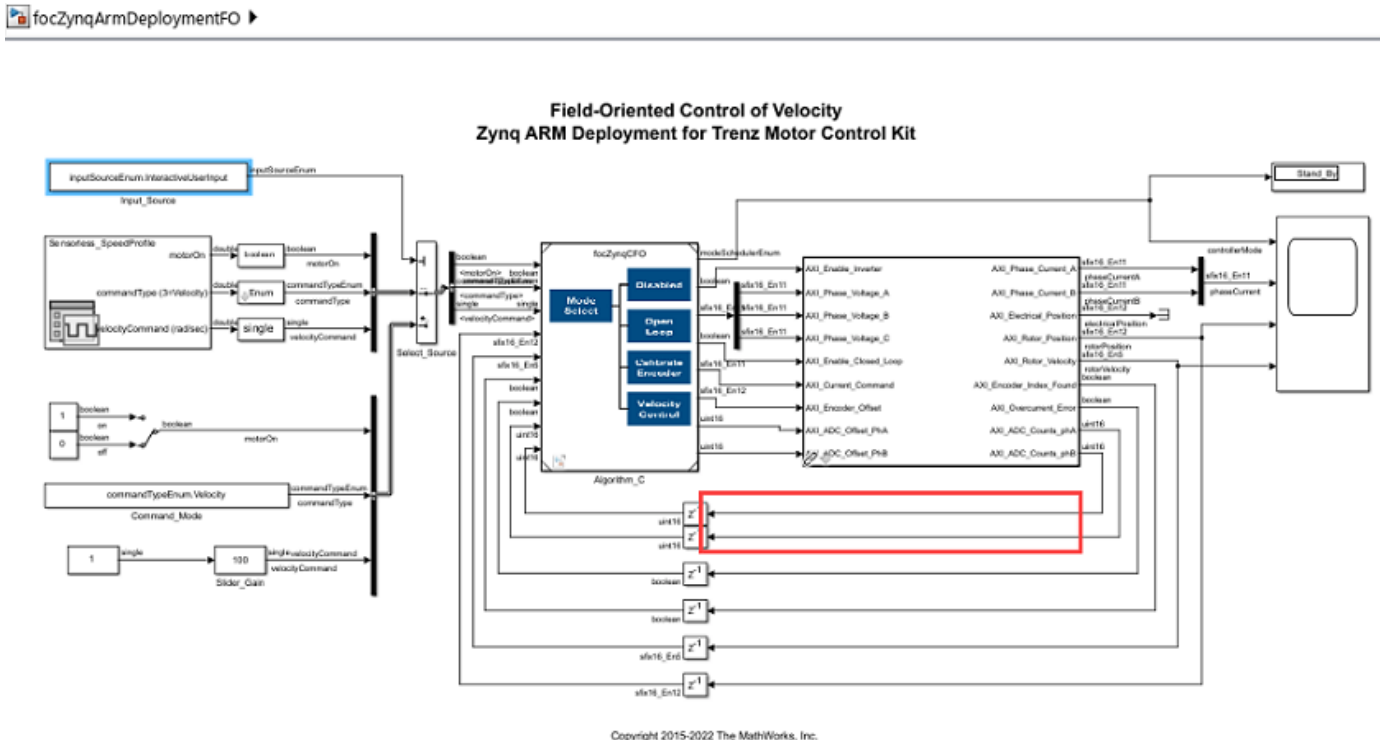
> **Note:** Ensure that you power OFF the inverter before calibrating the ADC offsets for phase A and phase B.

- The script `mcb_foc_fpga_data.m` included in the example automatically computes the PI controller gains.

- The script `mcb_update_data_dictionary.m` included in the example computes and updates the data dictionary variables.

- The example supports two plant models for simulation. The first plant uses the motor block from Motor Control Blockset. The second plant uses Simscape Electrical motor block.

Configure MATLAB Project to Use Sensor-Based or Sensorless Position Estimation

The MATLAB project includes the resources for both sensor-based and sensorless position estimation techniques. You can configure the project shortcuts to use resources for either one of these techniques:

1. Select **Project Shortcuts** to view the available project shortcuts.

2. Right-click the `t0_selectAlgorithm` shortcut and click **Open File** to edit the file `t0_algorithmChoice.m` containing the configuration code in the editor.

- Set `algorithmType = 'QEPFOCPMSM';` to enable and configure all the existing shortcuts to use the resources for quadrature encoder sensor-based control algorithm.

- Set `algorithmType = 'FOFOCPMSM';` to enable and configure all the existing shortcuts to use the resources for sensorless flux observer-based control algorithm.

3. Save and close the file `t0_algorithmChoice.m`.

These steps ensure that the MATLAB project opens the resources corresponding to the selected position estimation technique when you click any shortcut.

Simulate Algorithm Behavior

You can use the project shortcuts to open the Simulink models and simulate the algorithm behavior.

1. Select **Project Shortcuts** to view the available project shortcuts.

2. You can right-click the shortcuts to edit, open, and configure them.

3. Click the `t1_openSimulationTestBenchModel` project shortcut or use the following MATLAB command to open either `focZynqTestBench` (for sensor-based position estimation) or `focZynqTestBenchFO` (for sensorless position estimation) model.

`task.t1_openSimulationTestBenchModel`

The `Motor_And_Load` subsystem consists of a mathematical model of two surface PMSMs (one from Motor Control Blockset and the other from Simscape Electrical), motor load, encoder, and current sensor. The `Controller_Algorithm` subsystem includes an algorithm for I/O engineering unit conversion, an electrical position calculation algorithm, a rotor velocity calculation algorithm, a mode scheduler, and four control modes (disabled, open loop velocity control, encoder calibration, and closed loop velocity control). The C/D and D/C subsystems convert the data from continuous-time, variable time step solver and floating-point data type, to discrete-time, fixed time step solver and fixed-point data type, and vice-versa during simulation.
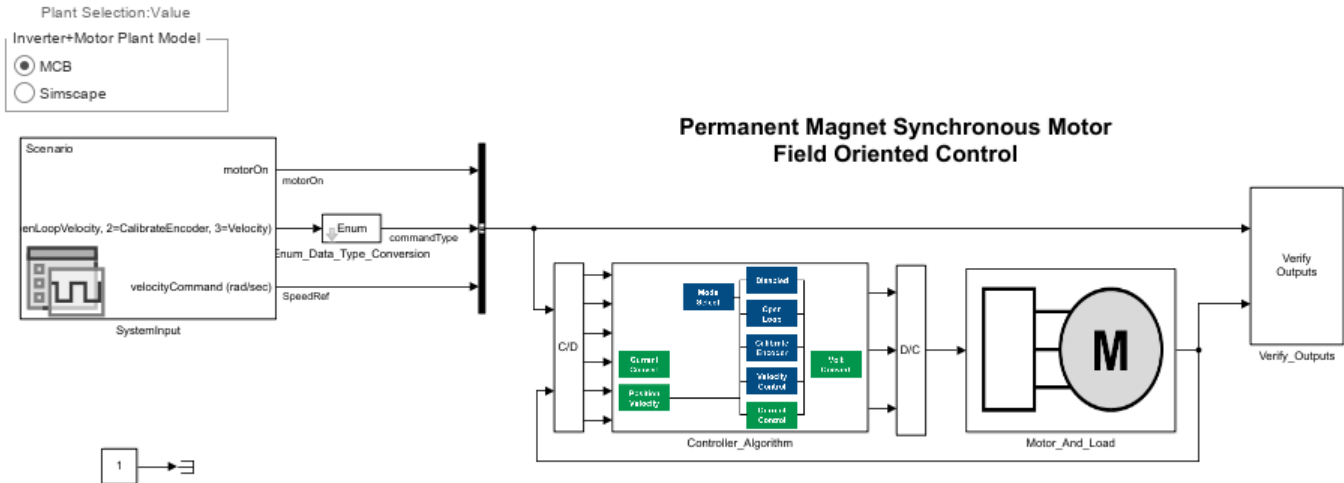
4. Use the **Plant Selection Value** button to select the type of plant that you want to use:

- MCB — Select this value to enable the motor block from Motor Control Blockset.

- Simscape — Select this value to enable the motor block from Simscape Electrical.

5. On the **Simulation** tab of the Simulink Toolstrip, click **Run** to simulate the model.

6. When the model finishes running, navigate to **Simulation** tab and click **Data Inspector** to open the Simulation Data Inspector.

7. In the Simulation Data Inspector, select the **commandType**, **SpeedRef**, **SpeedFb**, and **controllerMode** signals.

For the first two seconds, the controller calibrates the encoder position sensor. The encoder position sensor must be calibrated before the controller can achieve closed loop control. When position calibration begins, the motor accelerates using open loop control in order to identify the index pulse of the encoder.
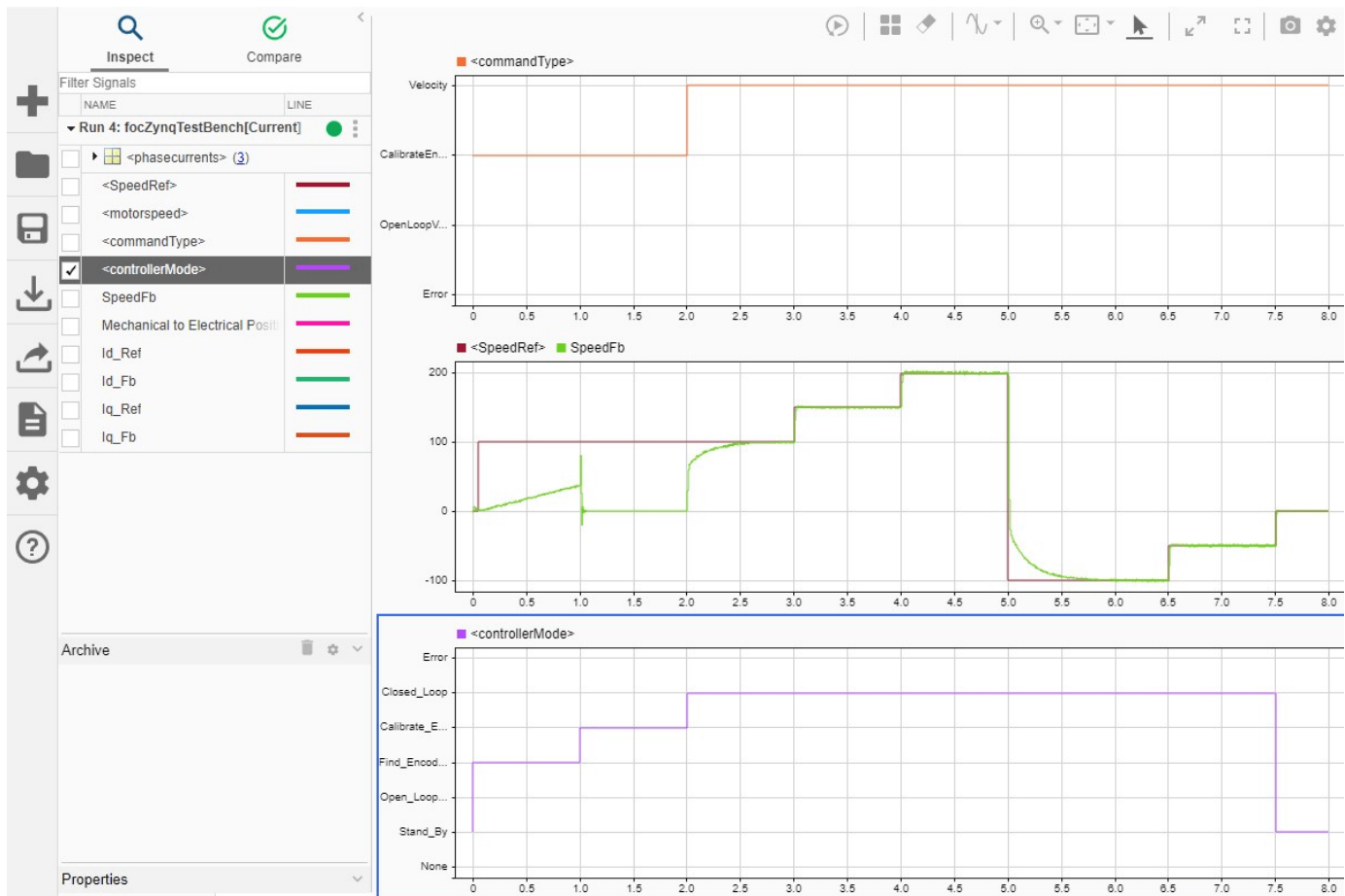
For quadrature encoder sensor-based controller algorithm — After finding the index pulse, the controller commands and holds a zero position until the encoder offset is identified. During this period the motor speed is zero (when using sensor-based position estimation, you can optionally run the motor using open-loop control). After about two seconds, the controller begins closed-loop control and follows the given speed profile.

For sensorless flux observer-based controller algorithm — The example begins simulation by running the motor using open-loop control. After about two seconds, the controller begins closed-loop control and follows the given speed profile.

During the closed-loop speed control, the FOC algorithm regulates the phase currents in the PMSM.

The behavior related to open-loop and closed-loop execution of the motor remains same during hardware execution. See the next sections for details about code generation and deployment.

Example Design Architecture

The following figure shows the architecture of the field-oriented control design partitioned between the part that runs in embedded software and the other part that runs on the FPGA hardware.
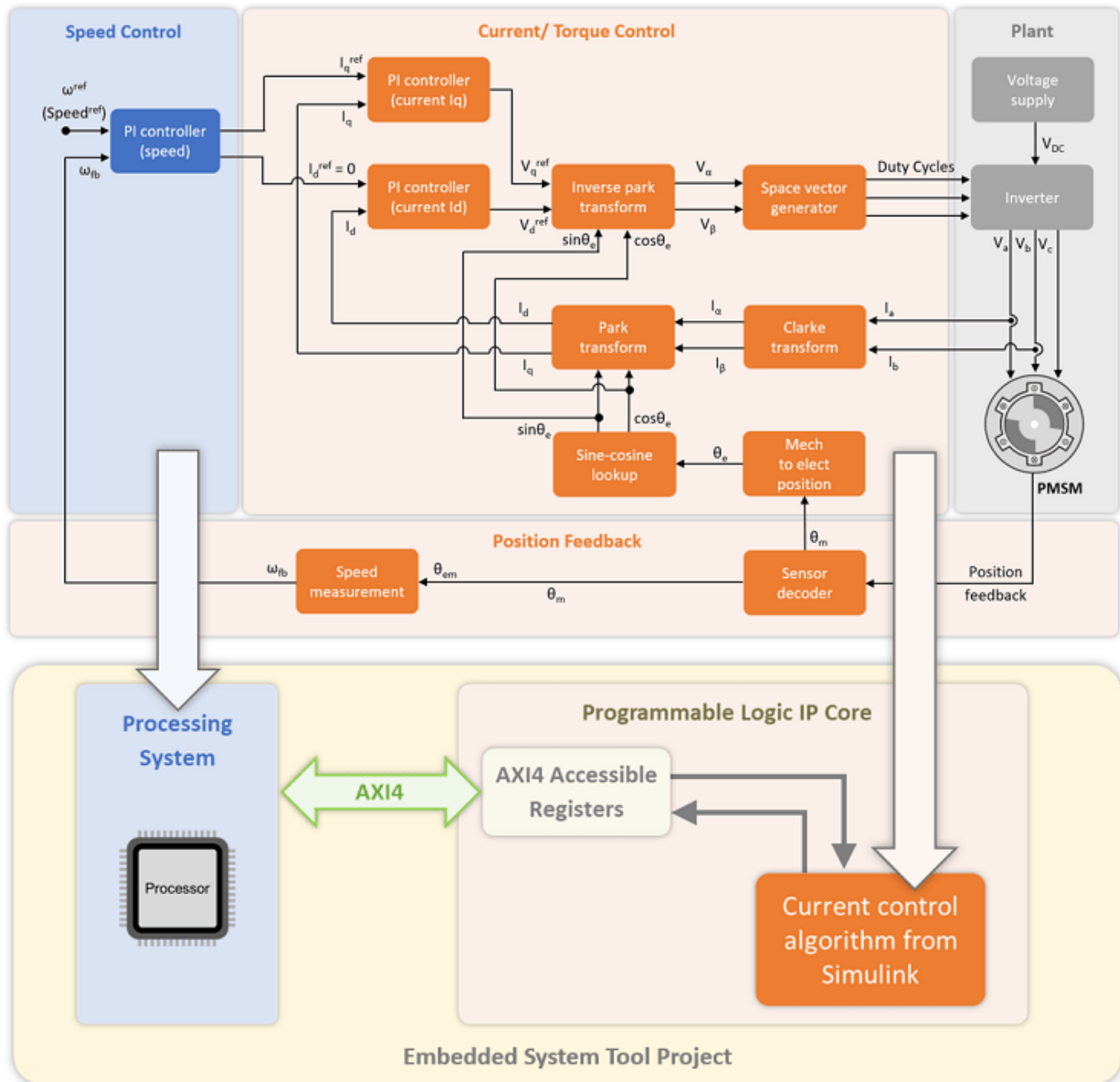
The speed control design runs in embedded software using a processor. The current or torque control design supports HDL code generation and runs on the FPGA hardware.

The hardware part of the design is enclosed in the models `focZynqHdl.slx` (for sensor-based position estimation) or `focZynqHdlFO.slx` (for sensorless position estimation). This part uses HDL Workflow Advisor to define and generate an HDL IP core, which includes components such as AXI4 accessible registers, AXI4 interfaces, external ports, and ADC.

Using the HDL Workflow Advisor IP core generation workflow, you insert this IP core into a reference design and generate an FPGA bitstream that runs on the SoC hardware. You can also use the HDL

4-267

Workflow Advisor to generate a software interface containing AXI driver blocks that connects the embedded software and the FPGA hardware. The example uses external mode to interface with the target model.

For more details, see "Hardware-Software Co-Design Workflow for SoC Platforms" (HDL Coder) and "Targeting FPGA & SoC Hardware" (HDL Coder).

Partition Algorithm and Generate Code

You can partition the controller algorithm into sections that separately generate C code (for the designed software) and HDL code (for the designed hardware). See the reports created during the code generation step to determine ways to integrate the generated code and build your application according to your own embedded design.

1. In the `focZynqTestBench` (for sensor-based position estimation) or `focZynqTestBenchFO` (for sensorless position estimation) model, open the `Controller_Algorithm` subsystem. The controller algorithm contains the `Algorithm_C` and `Algorithm_HDL` blocks, which reference the following models respectively:

- `focZynqC` (for sensor-based position estimation) or `focZynqCFO` (for sensorless position estimation)

- `focZynqHdl` (for sensor-based position estimation) or `focZynqHdlFO` (for sensorless position estimation)

The `focZynqC` (or `focZynqCFO`) model contains the portion of the algorithm to be implemented in software (ARM processor). Similarly, the `focZynqHdl` (or `focZynqHdlFO`) model contains the portion of the algorithm to be implemented on the FPGA hardware.

The parameters for the model are stored in the data dictionary. To access and modify the parameters, select:

- **Modeling > Model Explorer > focZynqTestBench > ExternalData** (for sensor-based position estimation)

- **Modeling > Model Explorer > focZynqTestBenchFO > ExternalData** (for sensorless position estimation)

Controller Algorithm
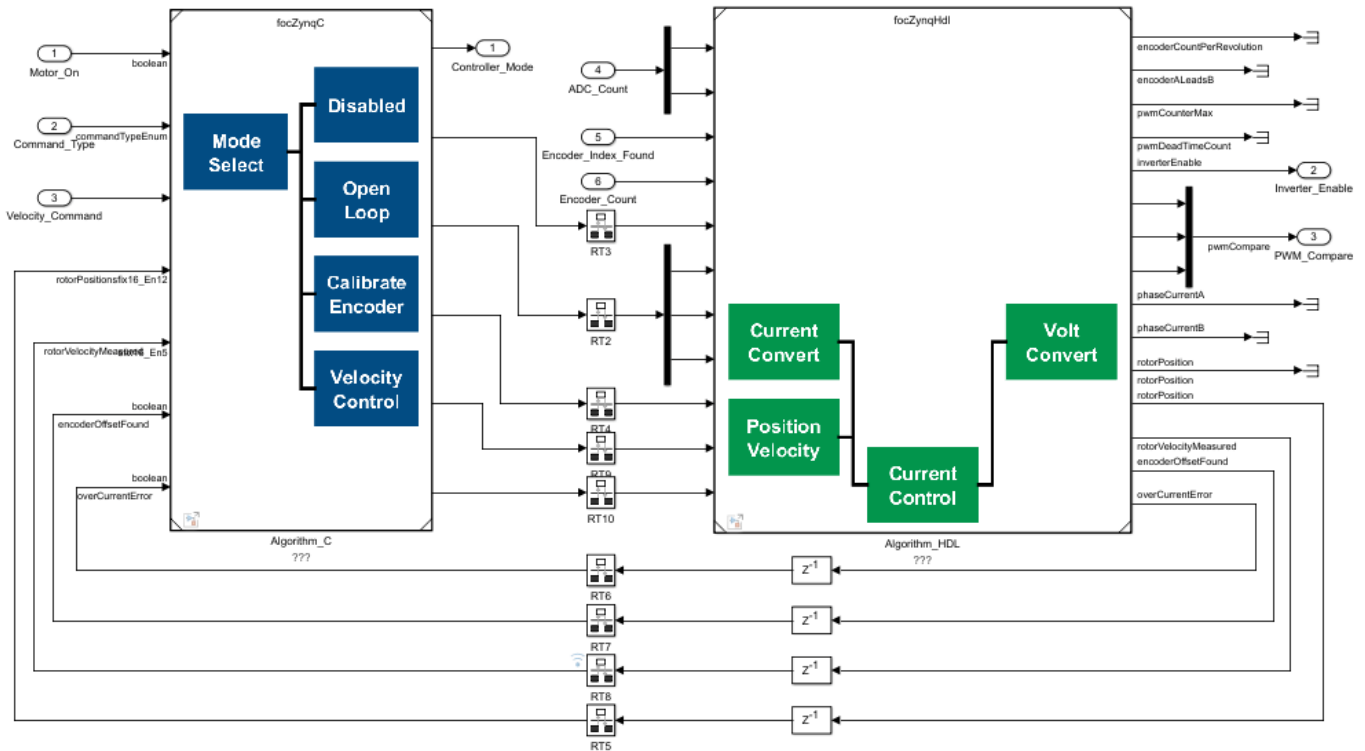
2. Click the `t2_generateCCode` project shortcut or use the following MATLAB command to open the model `focZynqC` (or `focZynqCFO`), generate C code, and generate a report.

`task.t2_generateCCode`

The `focZynqC` model contains the mode scheduler, velocity control loop, open-loop velocity controller, and a routine to automatically calibrate the encoder offset.

**Field-Oriented Control of Velocity
Zynq C Specification**

Copyright 2015-2020 The MathWorks, Inc.
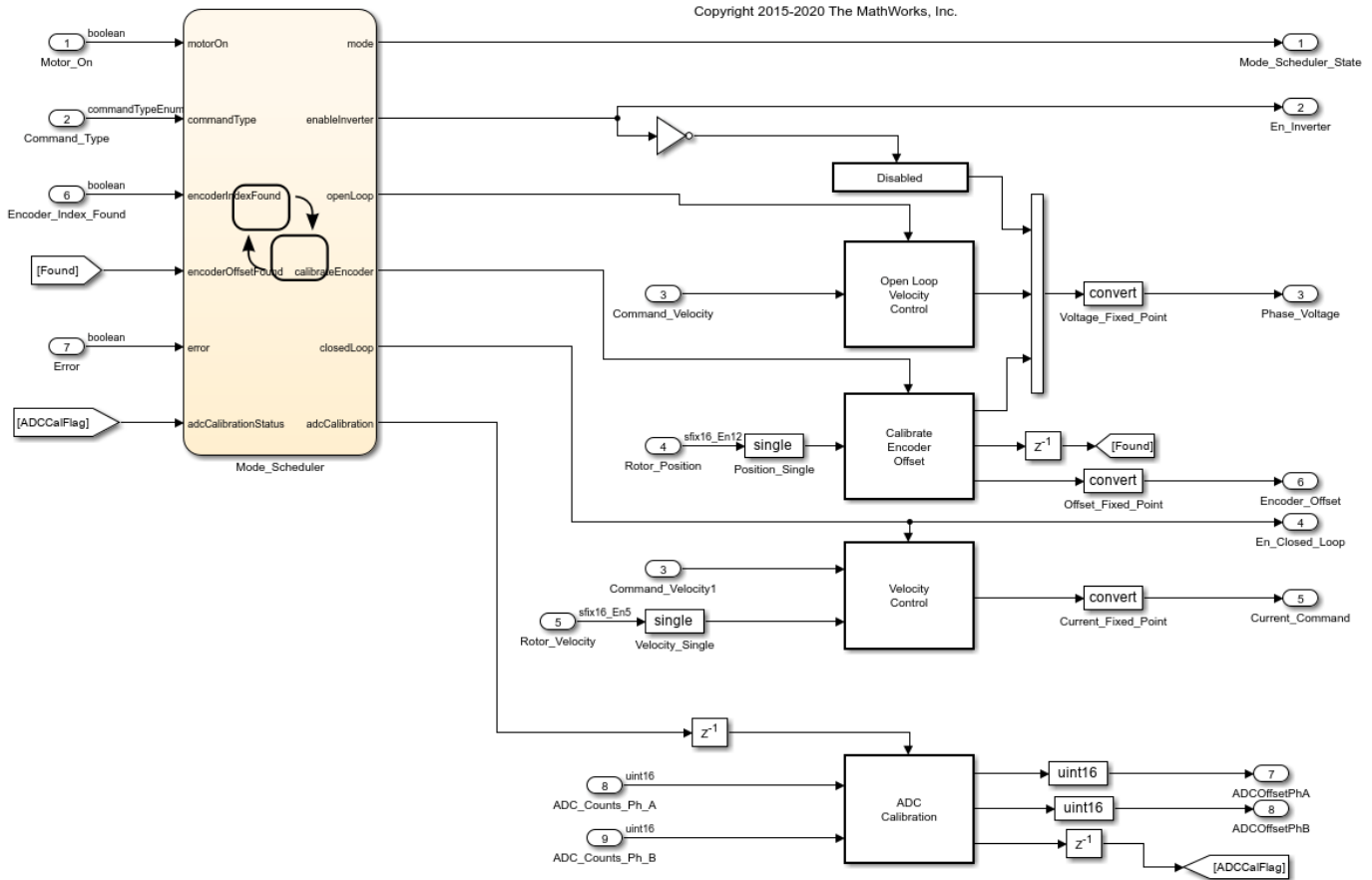
The `focZynqCFO` model contains the mode scheduler, velocity control loop, and open-loop velocity controller.

Field-Oriented Control of Velocity
Zynq C Specification

Copyright 2022 The MathWorks, Inc.
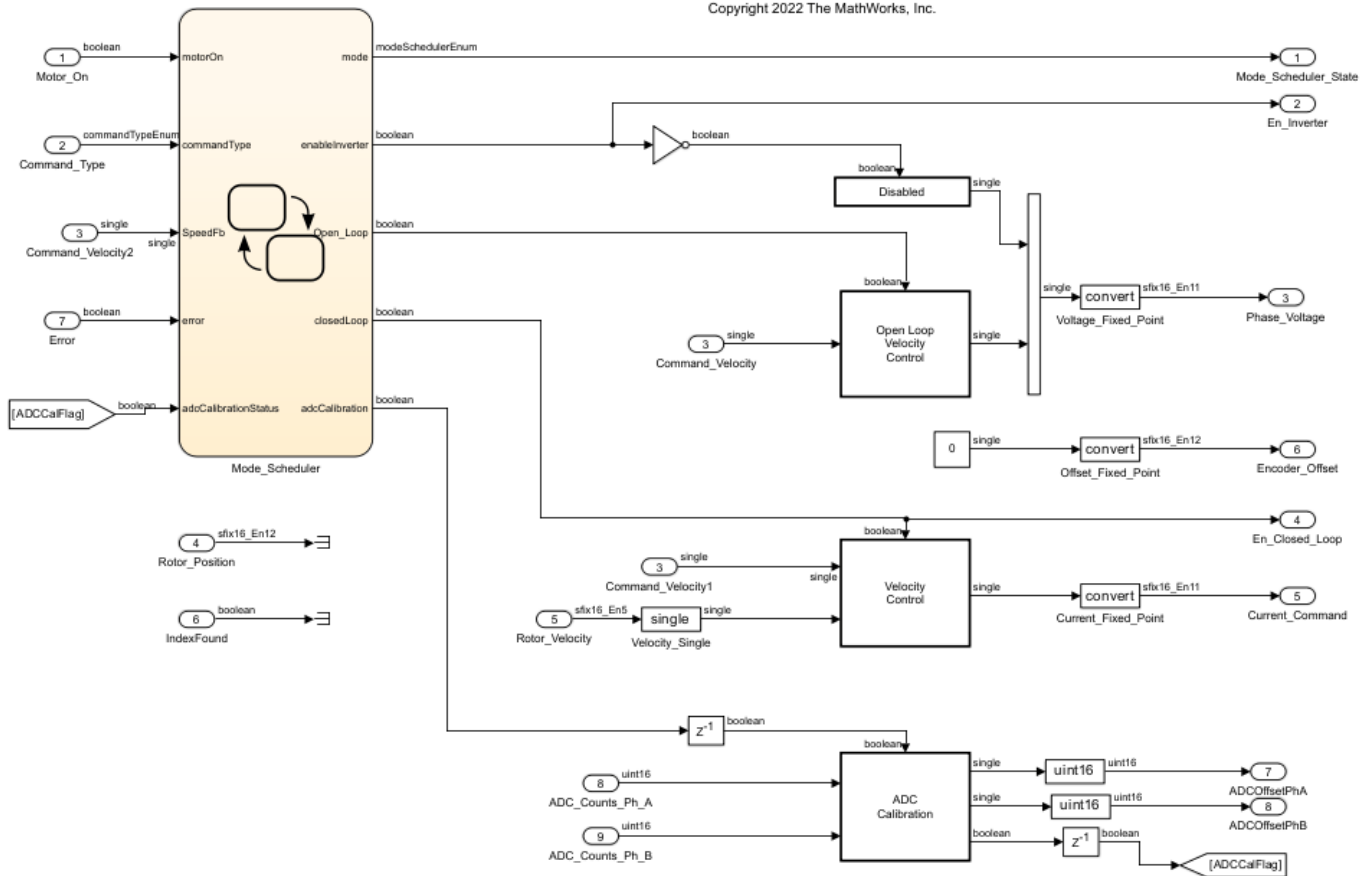
3. The Code Generation Report shows how the generated code corresponds to the model. If you are new to the Code Generation report, you can start with the **Code Interface Report** to view the function interface of the code. The C code is portable and can be integrated with any floating-point embedded processor that uses an ANSI-C compiler. For more information on the Code Generation Report, see "Reports for Code Generation" (Simulink Coder).
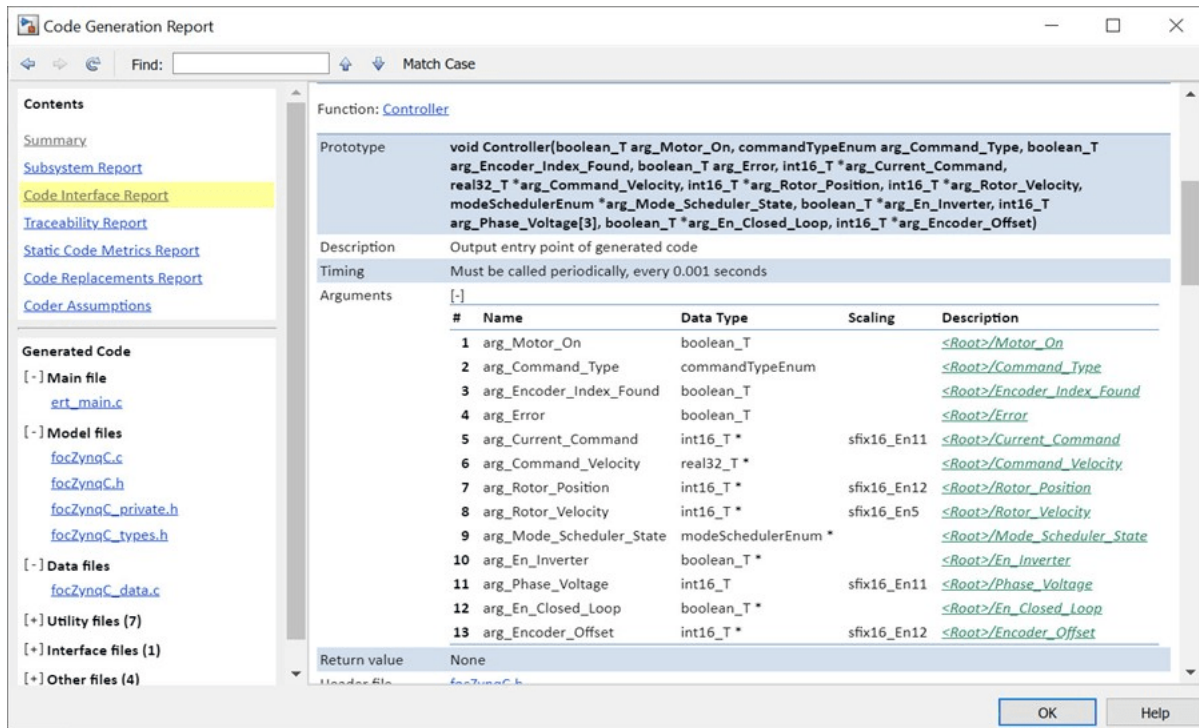
4. Click the `t3_generateHdlCode` project shortcut or use the following MATLAB command to open the model `focZynqHdl` (or `focZynqHdlFO`), generate HDL code, and generate a report.

```
task.t3_generateHdlCode
```

The `focZynqHdl` (and `focZynqHdlFO`) model contains the algorithm for electrical position calculation and rotor velocity calculation, over-current checks, and the field-oriented controller algorithm.
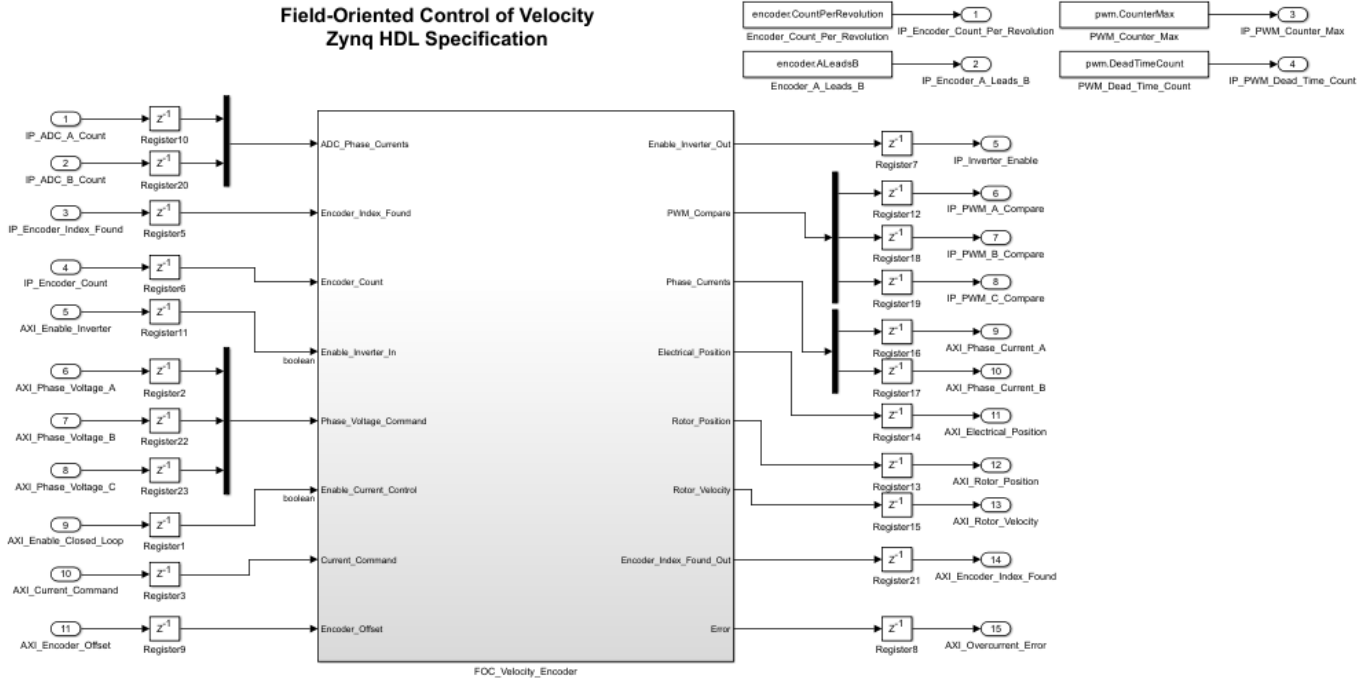
5. The Code Generation Report shows how the HDL code corresponds to the model. If you are new to the Code Generation report, you can start by exploring the **Generated Source Files** pane of the report and selecting the `focZynqHdl.vhd` (or `focZynqHdlFO.vhd`) file that contains the entity specification. The HDL code for the algorithm is portable and can integrate with any FPGA that supports VHDL code.

Setup Xilinx Zynq Platform and Motor Boards

Use this procedure to connect and setup the hardware required for this example.

1. Setup the Xilinx Zynq hardware platform. For information about hardware setup, see "Install Support for Xilinx Zynq Platform" (Embedded Coder Support Package for Xilinx Zynq Platform). Install both Embedded Coder Support Package for Xilinx Zynq Platform and HDL Coder Support Package for Xilinx Zynq Platform.

2. Connect the Trenz Electronic Motor Control Development Kit TE0820 as shown below:
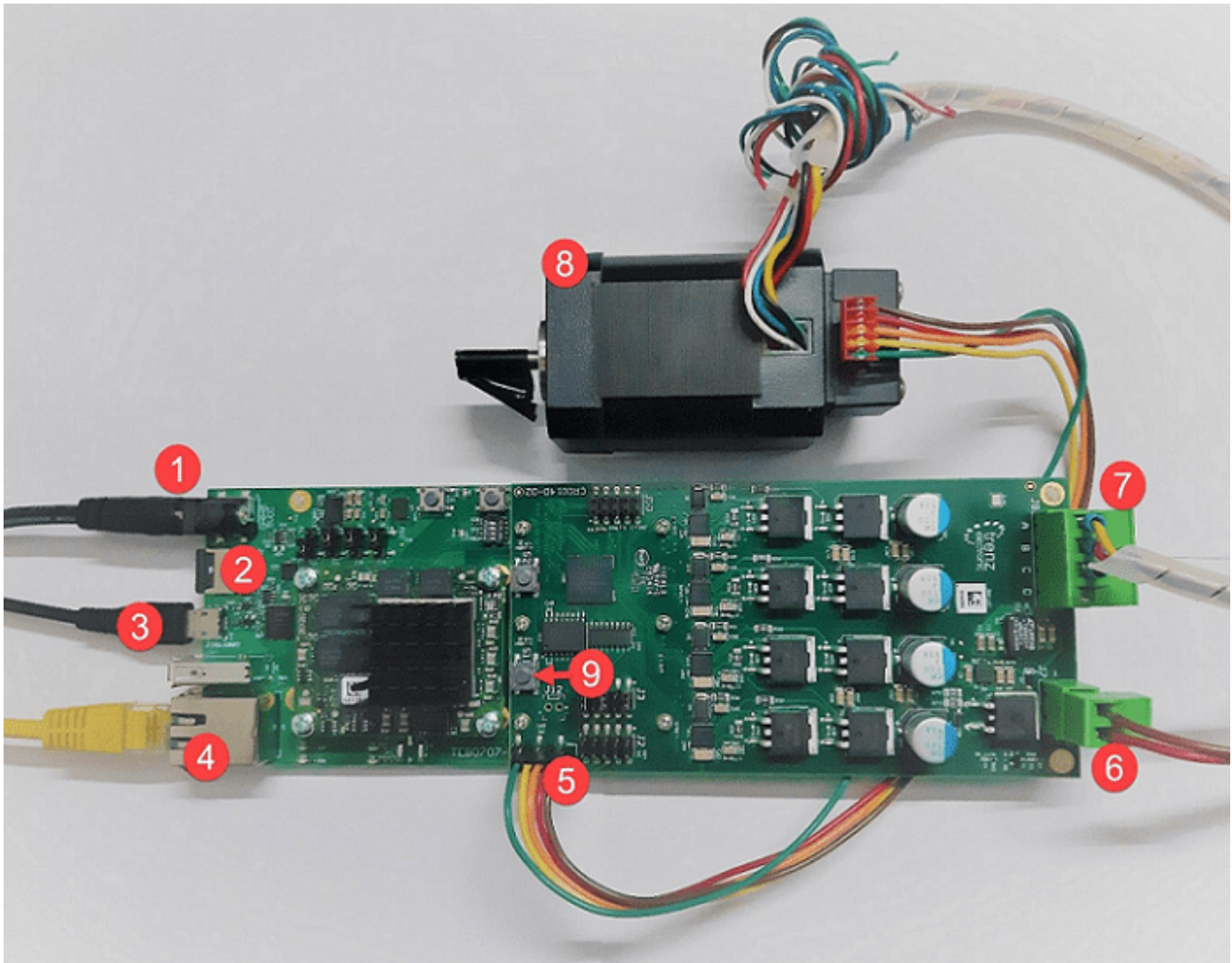
The preceding image shows these components:

1. 5 V DC Power Supply

2. SD - Card

3. Micro USB cable for UART and JTAG

4. Ethernet cable

5. Encoder connector

6. 24 V DC Power Supply

7. Motor Power cable (A, B, C)

8. 24 V Brushless DC Motor

9. Switch 1 (S1) controls power to the driver board

3. Ensure that jumper J4 on the carrier module is set to SD and J3 on the motor driver card is set up for a single ended encoder.
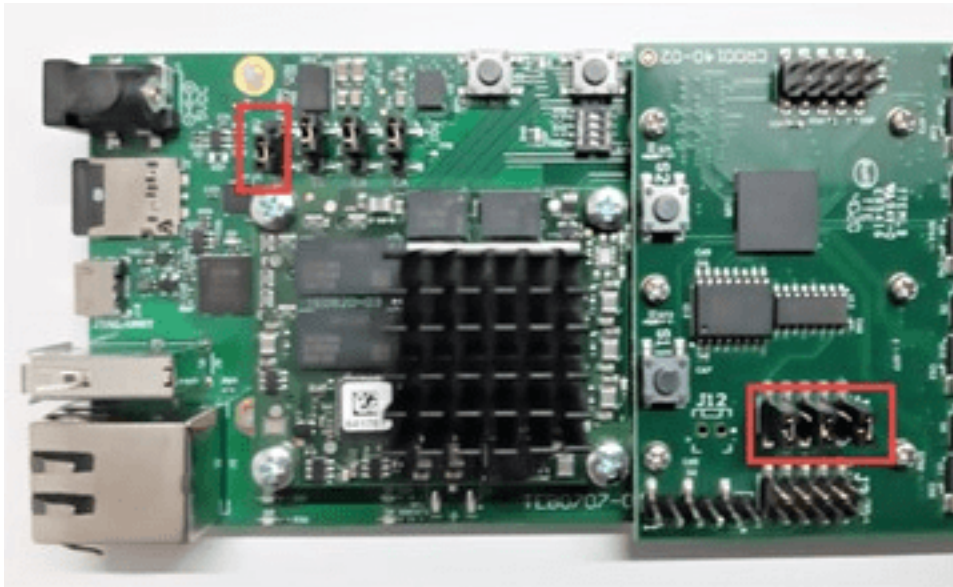
4. Insert the encoder cable according to the picture below.

Note: Step 4 is required only if you are using quadrature encoder sensor-based control algorithm.
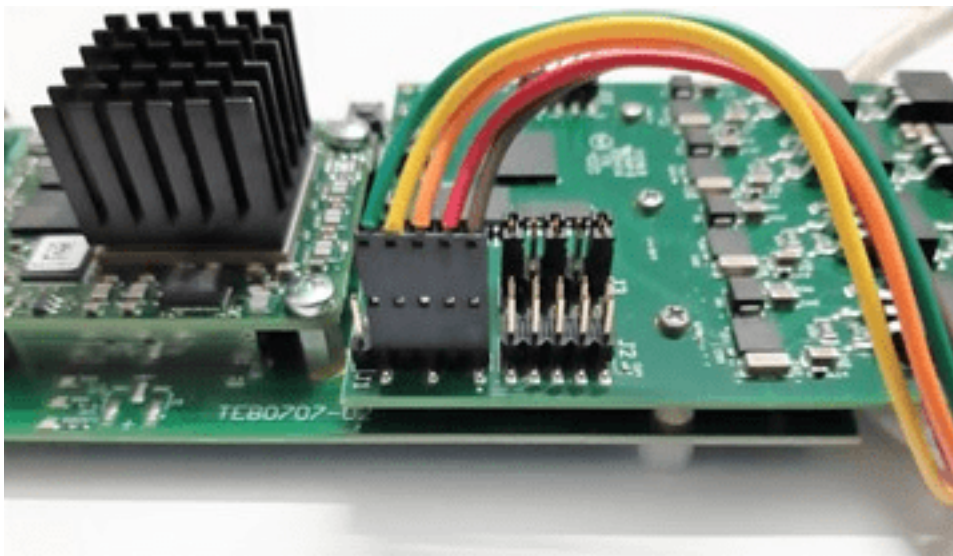
5. Download the Trenz TE0820 Linux Image, extract the ZIP archive, and copy the contents to the microSD card. Insert the microSD card in the J8 connector.

6. After you program the FPGA bitstream, press the S1 switch located on the motor driver card once to connect the 24V power supply to the MOSFETS. In case of any unexpected behavior from the device, use the S1 switch to disconnect the power.

Deploy Bitstream to Programmable Logic

Use HDL Workflow Advisor to generate the HDL code for the algorithm. Follow these steps to package HDL into an IP core, integrate the IP core into a Xilinx reference design, and then create a bitstream:

1. Click the `t4_openHdlWorkflowAdvisor` project shortcut or use the following MATLAB command to open the HDL Workflow Advisor.

```
task.t4_openHdlWorkflowAdvisor
```

2. In the **HDL Workflow Advisor > 1. Set Target > 1.1 Set Target Device and Synthesis Tool** group, set the **Target platform** to **Trenz TE0820 with CR00140**. **Trenz TE0820 with CR00140** is a Vivado® reference design containing the ADC, encoder, and PWM components. For information about creating this reference design, see "Define Custom Board and Reference Design for Zynq Workflow" (HDL Coder).
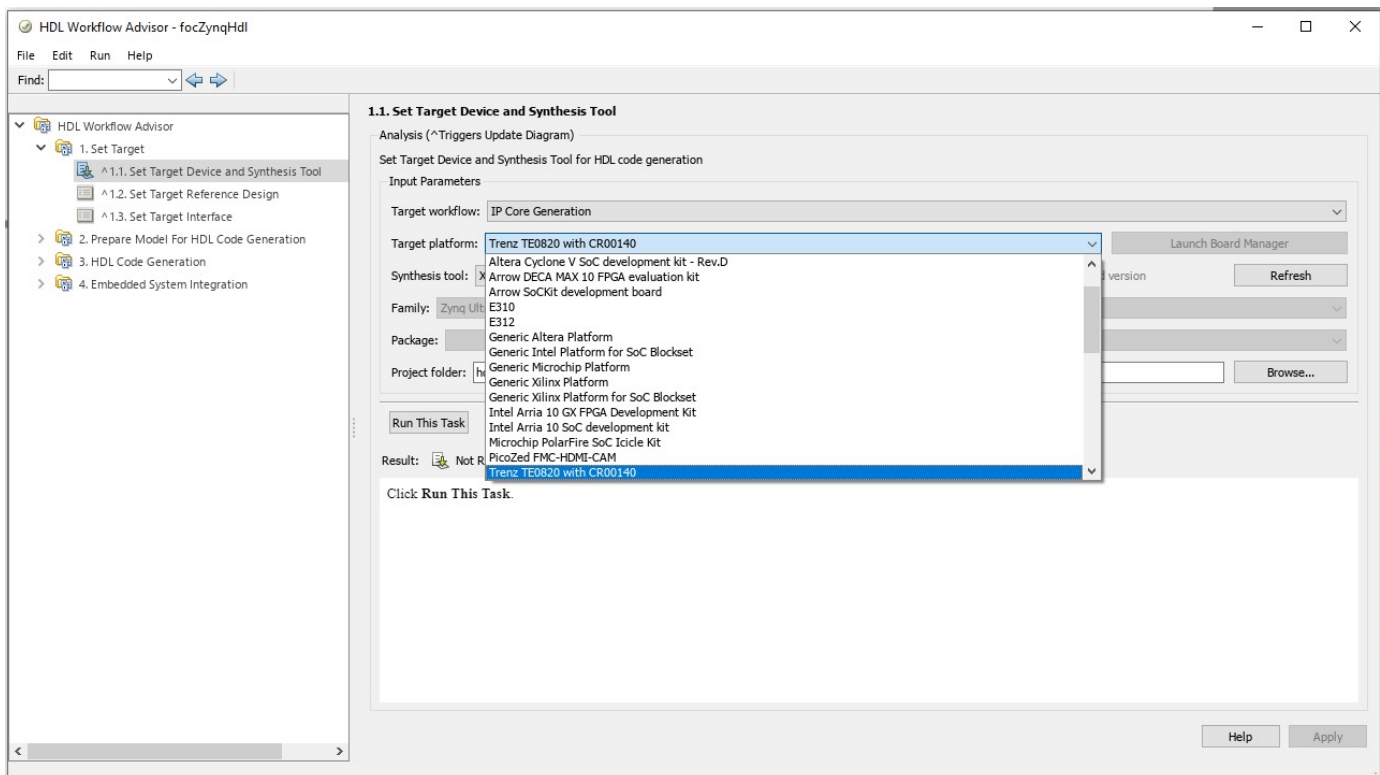
3. Select **1.3. Set Target Interface** to identify the ports. The **Target Platform Interfaces** column entries that have the prefix "IP" indicate the connections that are registered with the Trenz motor control reference design.
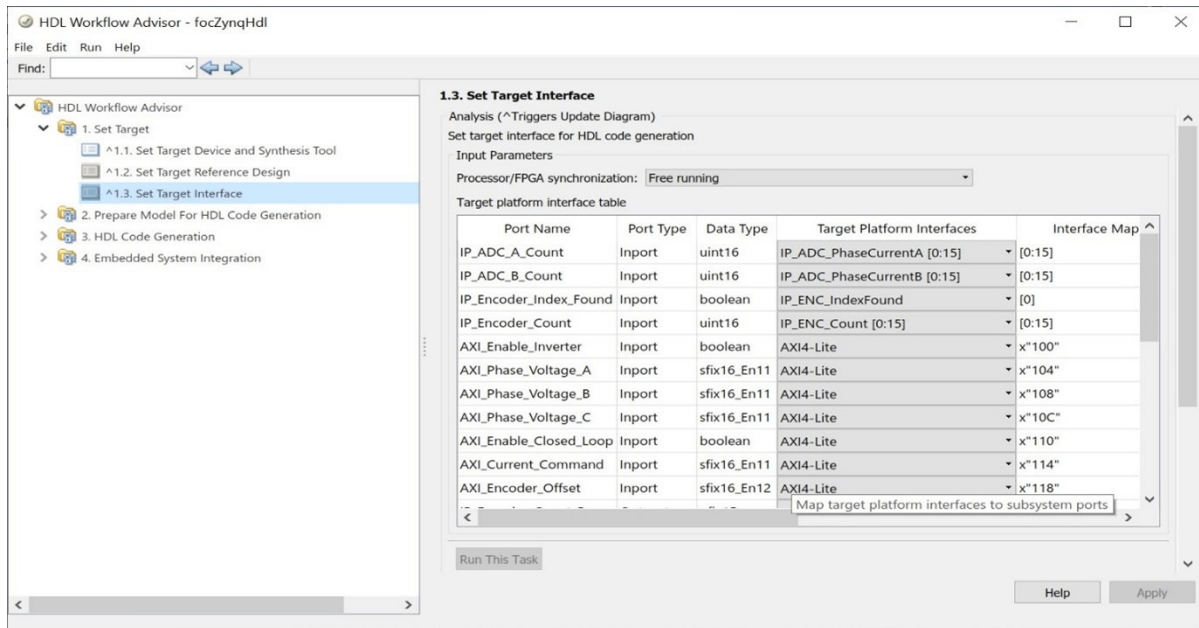
4. Select **4.3 Build FPGA Bitstream > Run to Selected Task** or click the `t5_generateBitstreamAndInterfaceBlock` project shortcut to generate the HDL code for the algorithm and create the FPGA bitstream from the Xilinx reference design. Alternatively, you can use the following MATLAB command to perform this action:

```
task.t5_generateBitstreamAndInterfaceBlock
```
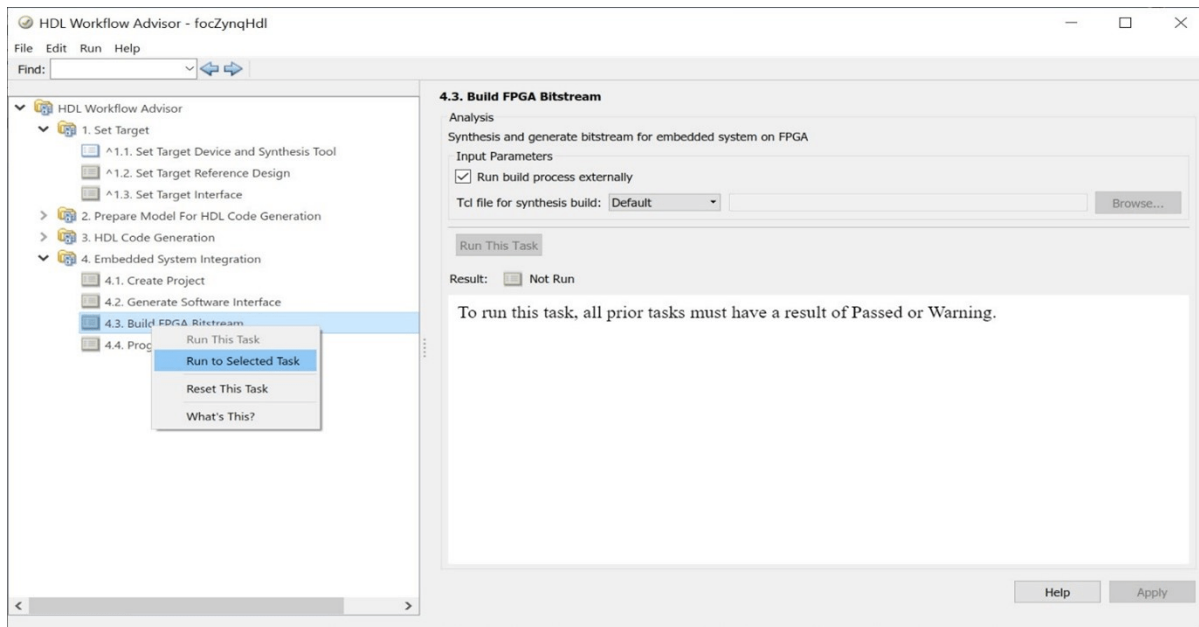

5. Follow the progress of the bitstream generation on the new DOS command prompt that opens. In addition to generating the bitstream, the customized target generates the `focZynqHdlAxiInterfaceLib` software interface library. The library contains an `AXI_Interface` block. The `AXI_Interface` block, which contains the AXI4-Lite interface components, provides

connectivity from the model deployed on the ARM processor to the model deployed on the programmable logic.
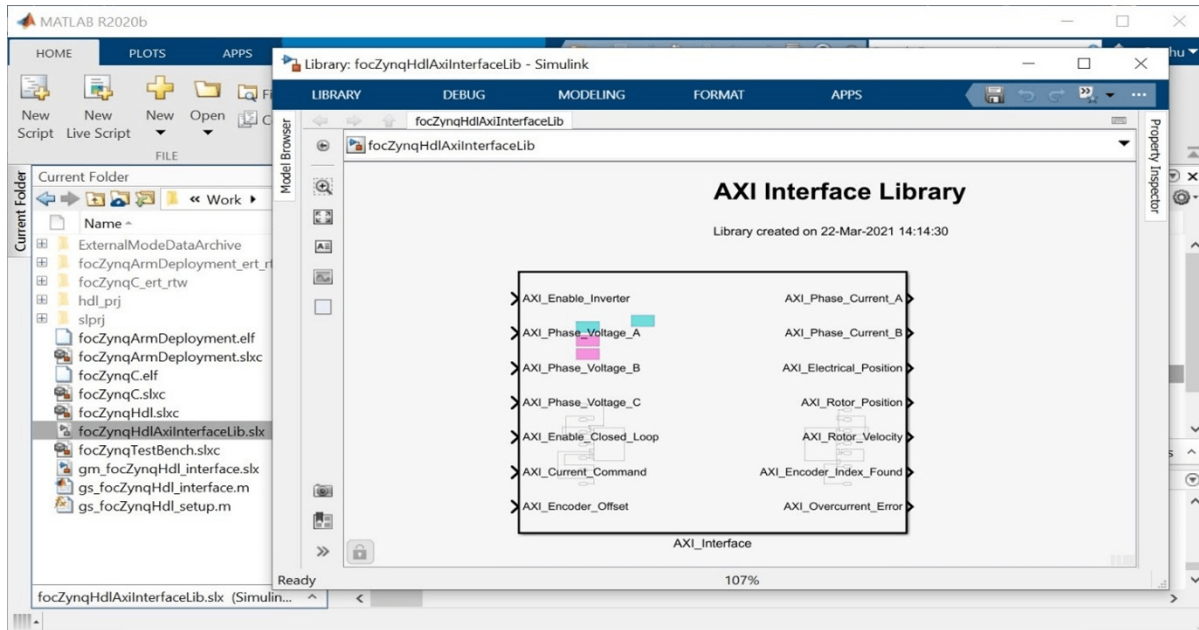
6. Run task **4.4 Program Target Device** or click the `t6_downloadBitstream` project shortcut to program the FPGA. Alternatively, you can use the following MATLAB command to perform this action:

```
task.t6_downloadBitstream
```

Deploy Executable to ARM Processor

You can generate the C code for the controller and automatically integrate this code with a Linux® reference framework to build, deploy, and run the model as an executable on the ARM processor. You can then compare the data logged by the model running on the processor with the simulation results.

1. Click the `t7_openZynqArmModel` project shortcut to open the `focZynqArmDeployment` model. Alternatively, you can use the following MATLAB command to perform this action:

```
task.t7_openZynqArmModel
```

The `focZynqArmDeployment` model can generate C code, automatically integrate this code with a Linux ARM reference framework, and deploy the executable to the ARM processor on the Xilinx Zynq platform. The deployable model uses references to the original controller model and contains test stimulus, scope, and `AXI_Interface` library block that you created in the section Deploy Bitstream to Programmable Logic.

4-279

Field-Oriented Control of Velocity
Zynq ARM Deployment for Trenz Motor Control Kit
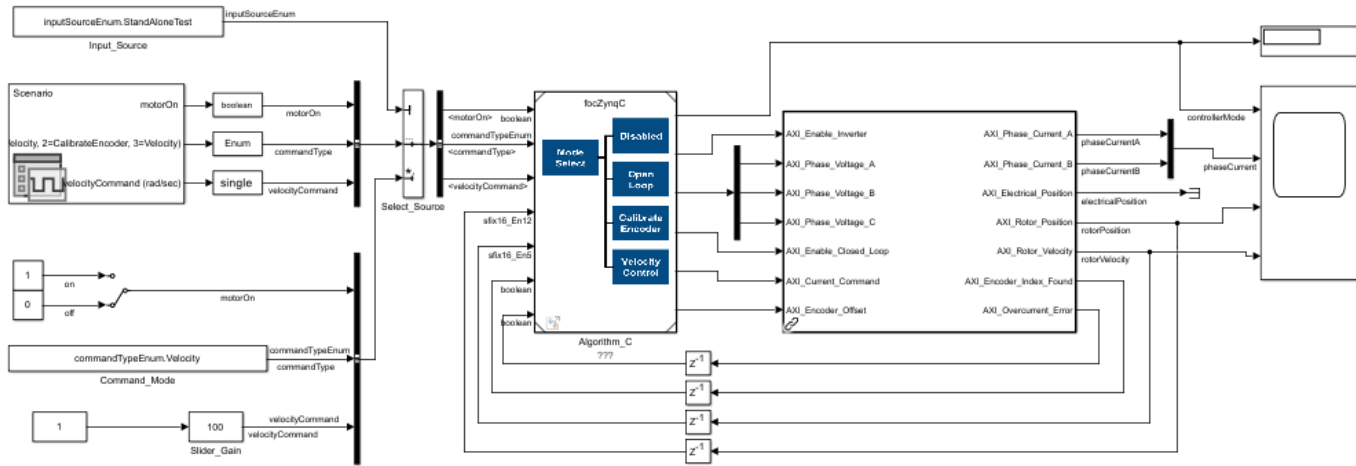
On the **Hardware** tab, click **Monitor & Tune** to build, deploy, and run the model as an executable on the ARM processor. The system compiles the generated code against the reference framework to create an executable. During model execution, Simulink monitors the signals and shows them in the scope.

Press the S1 switch located on the Motor Driver card once. This connects the 24V power supply to the MOSFETS, after which the motor should run for 8 seconds. If you see any unexpected behavior in the device, use the S1 switch to disconnect the power.

3. Open the simulation data inspector to view the logged signals and compare them with the signals logged previously from the `focZynqTestBench` model. You can open simulation data inspector by clicking **Data Inspector** on the **Simulation** tab.

4. In the simulation data inspector, select the `rotorVelocity` signal.

During the encoder calibration mode, the compared signals initially differ and then match afterwards. This is because the simulated and real motors started with different rotor positions.

In contrast, the closed loop velocity control signals are very similar during simulation and hardware run. The small differences occur because the simulation model of the motor and sensors use data sheet values and do not explicitly account for the manufacturing tolerances of the physical motor.
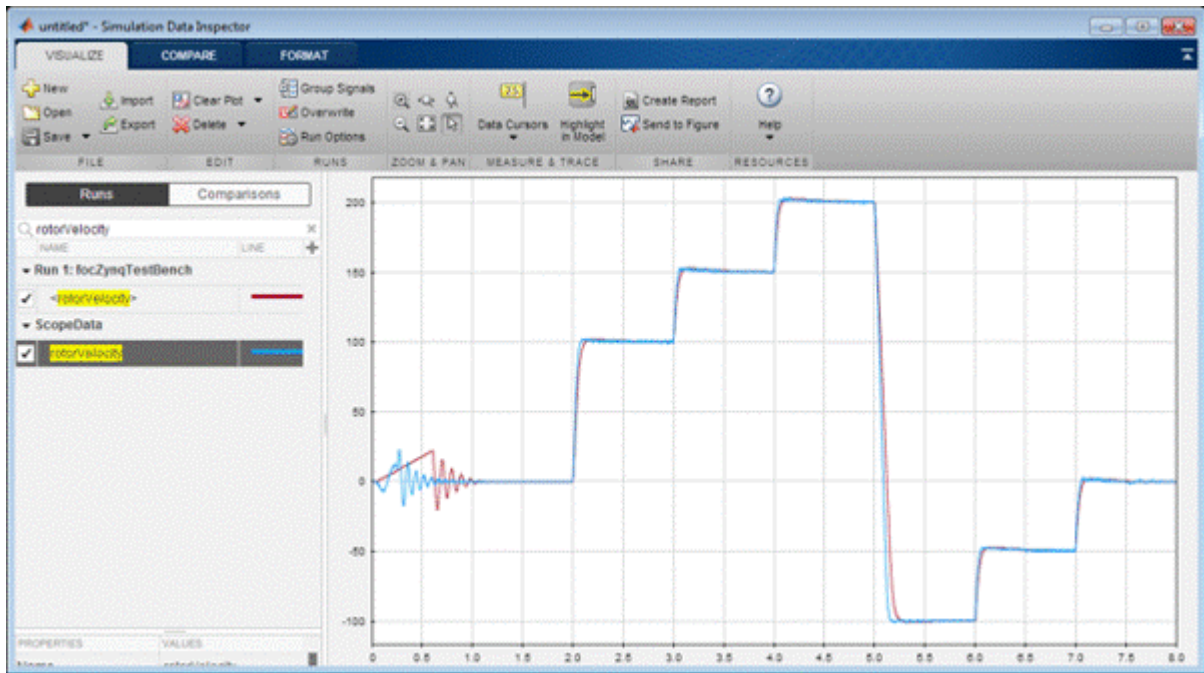
More to Try in This Example

Variations to the example workflow that you can further try include:

- HDL Coder supports floating-point single precision datatype. In the floating-point variant of the foczynqhdl model, you can utilize the ability of HDL Coder to generate HDL code from a model containing a mix of fixed-point and floating-point data types. Follow the same tasks in the example to implement the control algorithm with single-precision current control on a Xilinx Zynq SoC platform. To open the floating-point variant, right-click focZynqHdl/FOC_Velocity_Encoder/FOC_Current_Control and select **Label Mode Active Choice > FloatingPoint** within the **Variant** option.
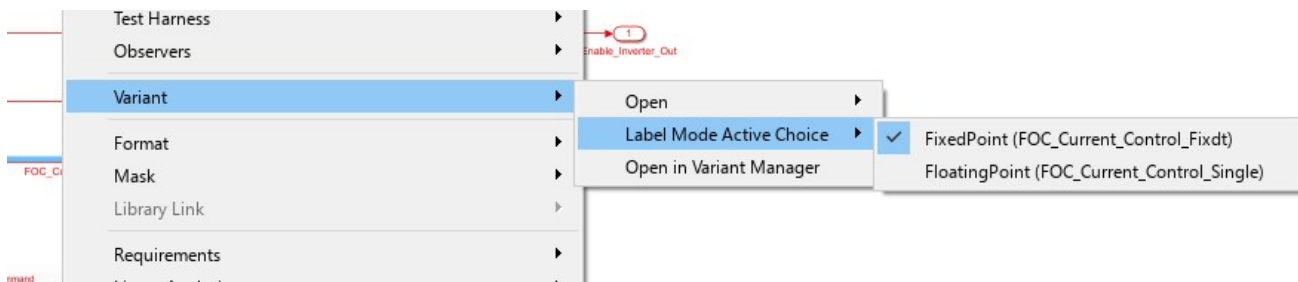
See Also

- Xilinx Zynq Support from MATLAB and Simulink - Hardware Support - MATLAB & Simulink

- Trenz Electronic Motor Control Development Kit Support from Simulink - Hardware Support - MATLAB & Simulink

- "Hardware-Software Co-Design Workflow for SoC Platforms" (HDL Coder)

- "Targeting FPGA & SoC Hardware" (HDL Coder)

Code Verification and Profiling Using PIL Testing

This example explains PIL profiling on Texas Instruments™ LAUNCHXL-F28379D hardware board. In processor-in-the-loop (PIL) simulation, the control algorithm executes in the target hardware, but the plant model runs on the host machine. The plant model simulates the input and output signals for the controller and communicates with the controller by using the serial communication interface. This functionality allows you to use PIL simulation to determine the execution time on the target hardware, which you can then compare with the execution time for simulating the model on the host machine.

The execution time, or the performance metric of an algorithm that you obtain from PIL simulation, helps you detect algorithm overrun on the target hardware. The PIL profiling report shows the average and maximum execution times of an algorithm on the target hardware.

This example uses the `mcb_pmsm_foc_sim.slx` model to show code verification in PIL simulation. This example shows PIL profiling for the Current Control subsystem in the model. This subsystem includes the Field-Oriented Control (FOC), current scaling (per-unit conversion), speed measurement, and rotor position scaling (computation of angle from the encoder position counts) algorithms. The PIL profiling report shows the average and maximum execution times of the control algorithm in the target hardware.

This example consists of these tasks:

- Verify code execution by using PIL testing by comparing the algorithm in the simulation and target hardware operating modes.

- Perform PIL profiling by measuring the algorithm execution time in the target hardware and generate the PIL profiling report.

Required MathWorks Products

- Motor Control Blockset™

- Embedded Coder®

- C2000™ Microcontroller Blockset

Supported Hardware

- LAUNCHXL-F28379D controller hardware board

Prepare PIL Model

1. Open the `mcb_pmsm_foc_sim.slx` model using the following MATLAB command:

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

4-283

PMSM Field Oriented Control
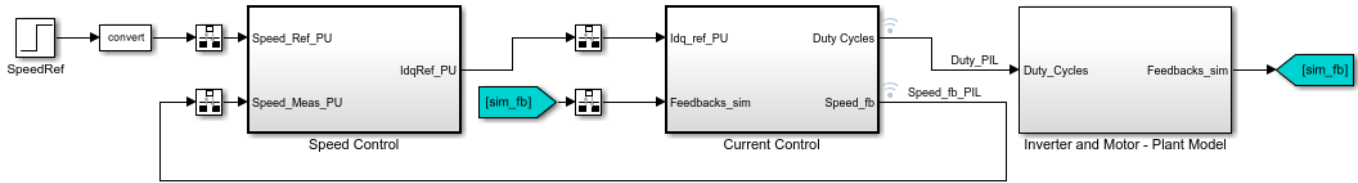
Motor Control Blockset v1.0
Copyright 2020 The MathWorks, Inc.
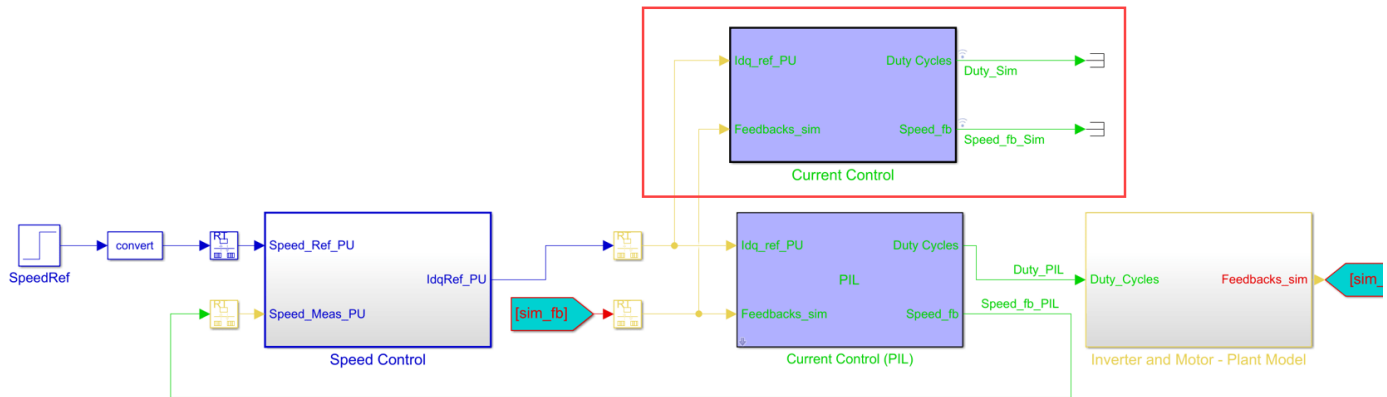
Explore more:
1. Edit motor & inverter parameters
2. Simulate this model

This model simulates the PMSM motor and the FOC algorithm for closed-loop speed control.

2. On the **Hardware** tab of the Simulink toolstrip, click **Hardware Settings**.

3. In the **Configuration Parameters** dialog box, under **Hardware Implementation**, set the **Hardware board** field to `TI Delfino F28379D LaunchPad`.
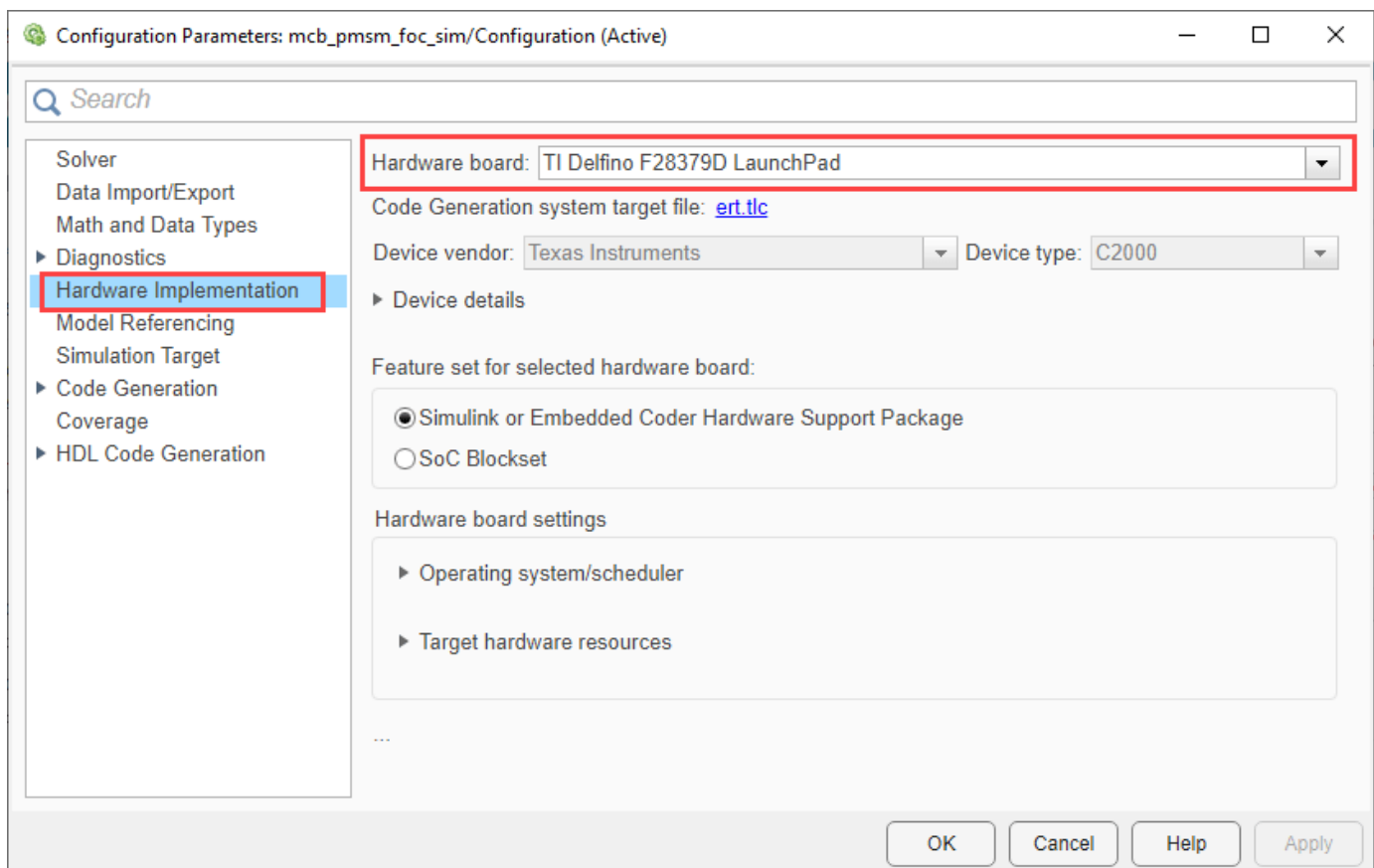
Verify Code by Using PIL

1. In the **Configuration Parameters** dialog box, select these configuration settings under **Hardware Implementation > Hardware board settings > Target hardware resources > PIL**:

> **a. Communication Interface** - Select `serial`.

> **b. SCI module** - Select `SCI_A`.

> **c. Serial port in MATLAB preferences** - The model automatically detects the communication port to which you have connected the hardware. This parameter remains unchanged for the rest of the currently active MATLAB® session. Click the **Refresh** button to detect the communication port again.
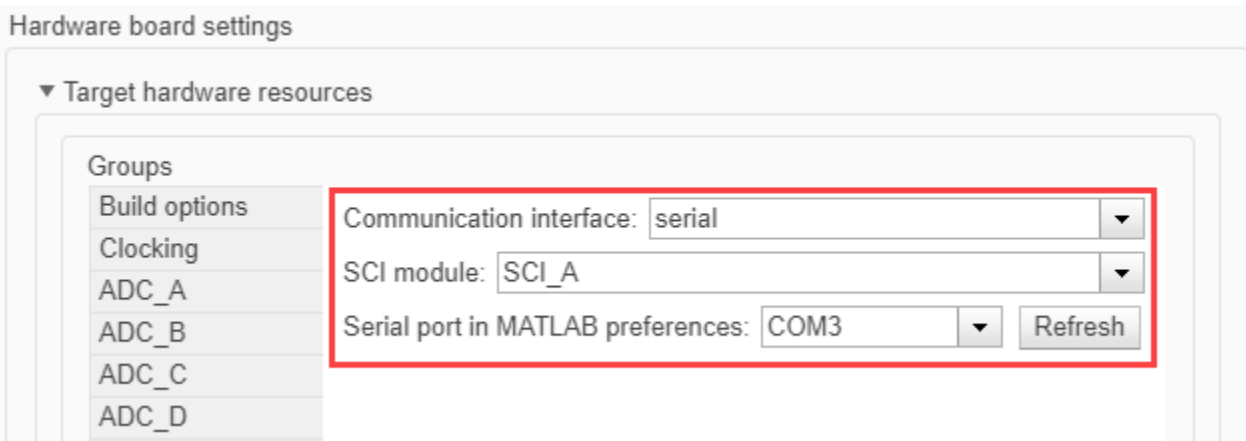
2. Open the `mcb_PIL_config_TI.m` script file to set the configuration parameters using this command:

```
edit('mcb_PIL_config_TI.m');
```

3. Update the model name and stop time in the script.

```
  mcb_PIL_config_TI.m  ✕  +

 1       % mcb_PIL_config_TI initializes the congiuration parameter for PIL profiling
 2       %
 3       % For PIL profiling, update the below in MATLAB script
 4       % model - name of the model identified for PIL profing
 5       % StopTime - time required for profiling. Ensure algorithm reaches steady
 6       % state within this specified time.
 7       %
 8       % This code is tested for TI LAUNCHXL-F28379D (TMS320F28379d)
 9       %
10       % Note: Before running the script, ensure COM port is updated in
11       % config set->hardware implementation->Target hardware resources->PIL
12       %
13       % Copyright 2020 The MathWorks, Inc.
14
15 -     model = 'mcb_pmsm_foc_sim';
16 -     set_param(model,'StopTime','0.5');
17 -     set_param(model,'SimulationMode','normal');
18 -     set_param(model,'ReturnWorkspaceOutputs','on');
19 -     set_param(model,'CodeExecutionProfiling','on');
20 -     set_param(model,'CodeProfilingInstrumentation','coarse');
21 -     set_param(model,'CodeProfilingSaveOptions','SummaryOnly');
22 -     set_param(model,'CreateSILPILBlock','PIL');
23 -     set_param(model,'DefaultParameterBehavior','Inlined');
24
```

4. Run the script to update the configuration parameters of the simulation model and the PIL preferences.

5. Right-click the Current Control subsystem in the `mcb_pmsm_foc_sim.slx` example model. Under the **C/C++ Code** menu, select **Deploy this Subsystem to Hardware**.
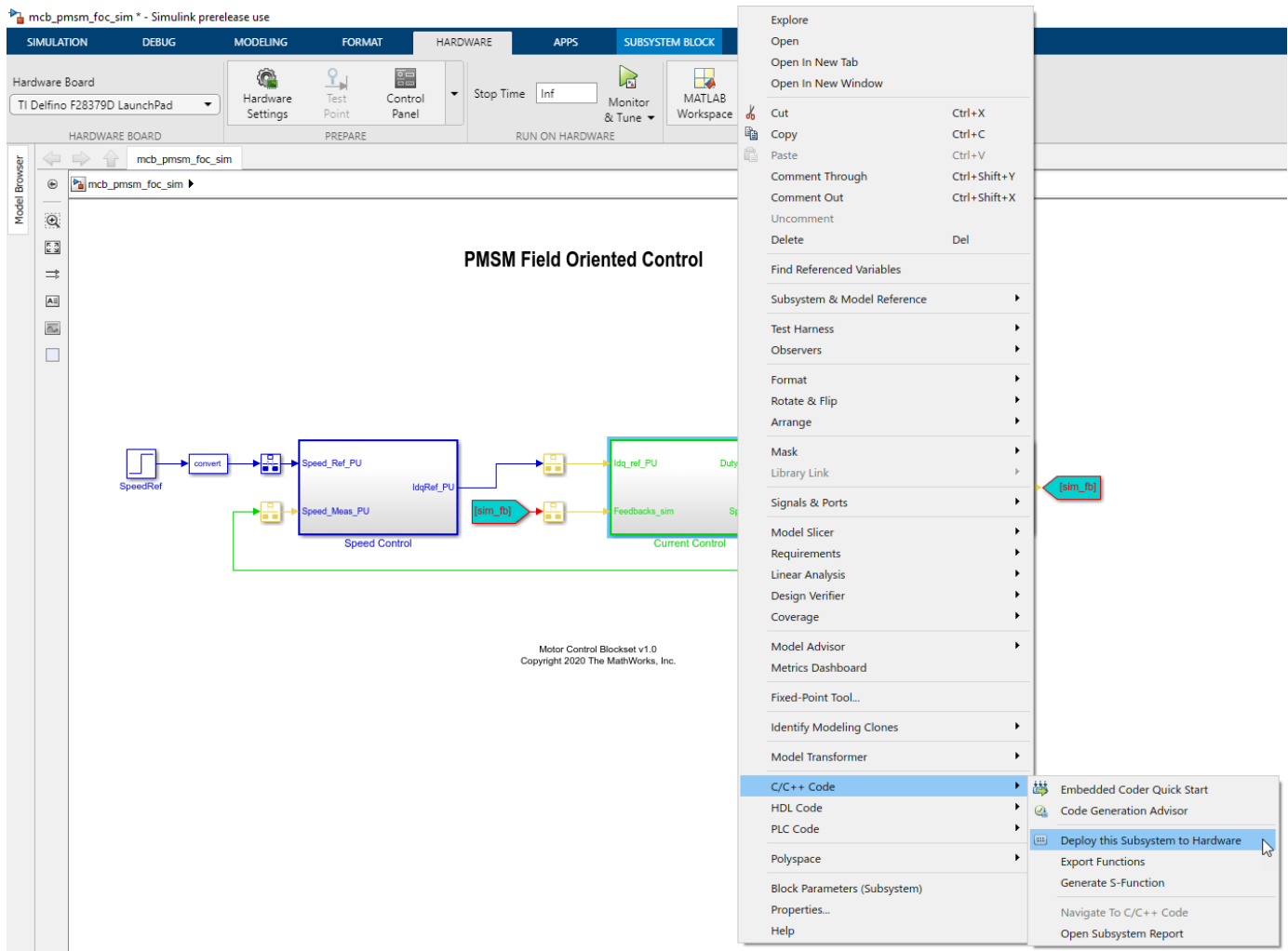
The system displays the **Build code for Subsystem** dialog box. Set the Storage Class to `Inlined` for all parameters.
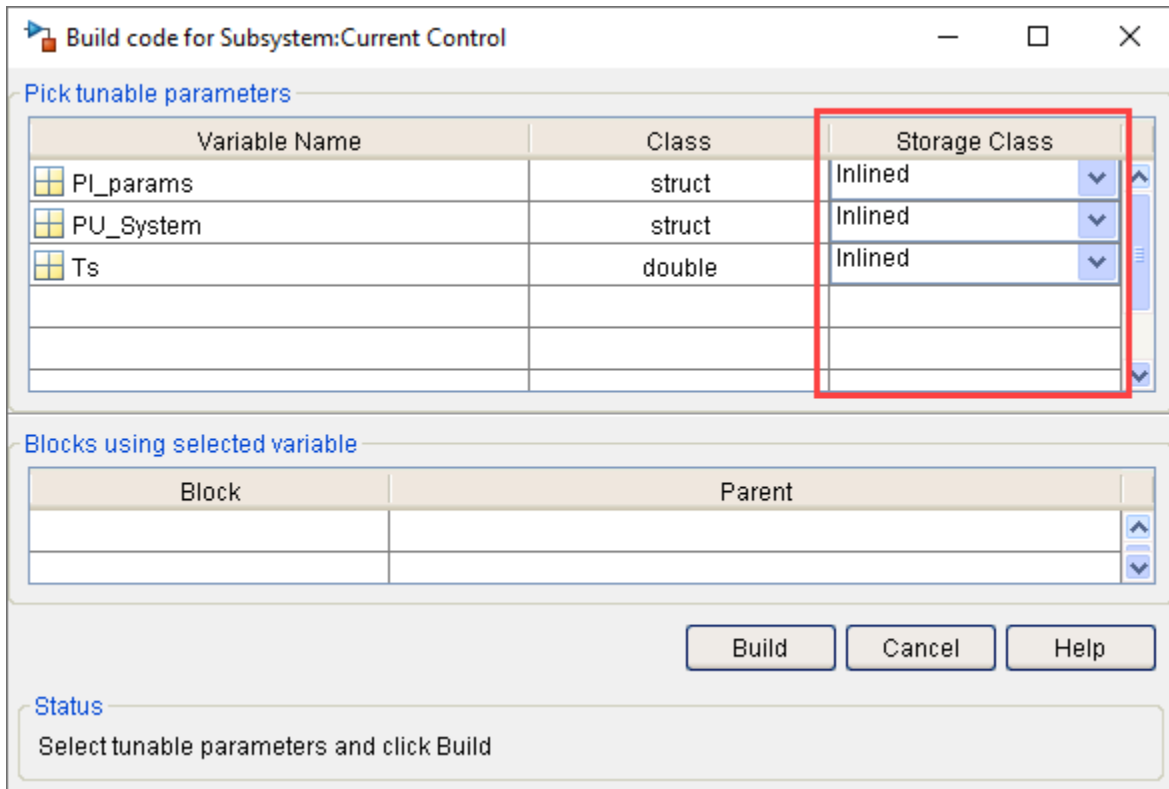
6. Click **Build** to create a model named `untitiled` that includes a PIL subsystem called Current Control.
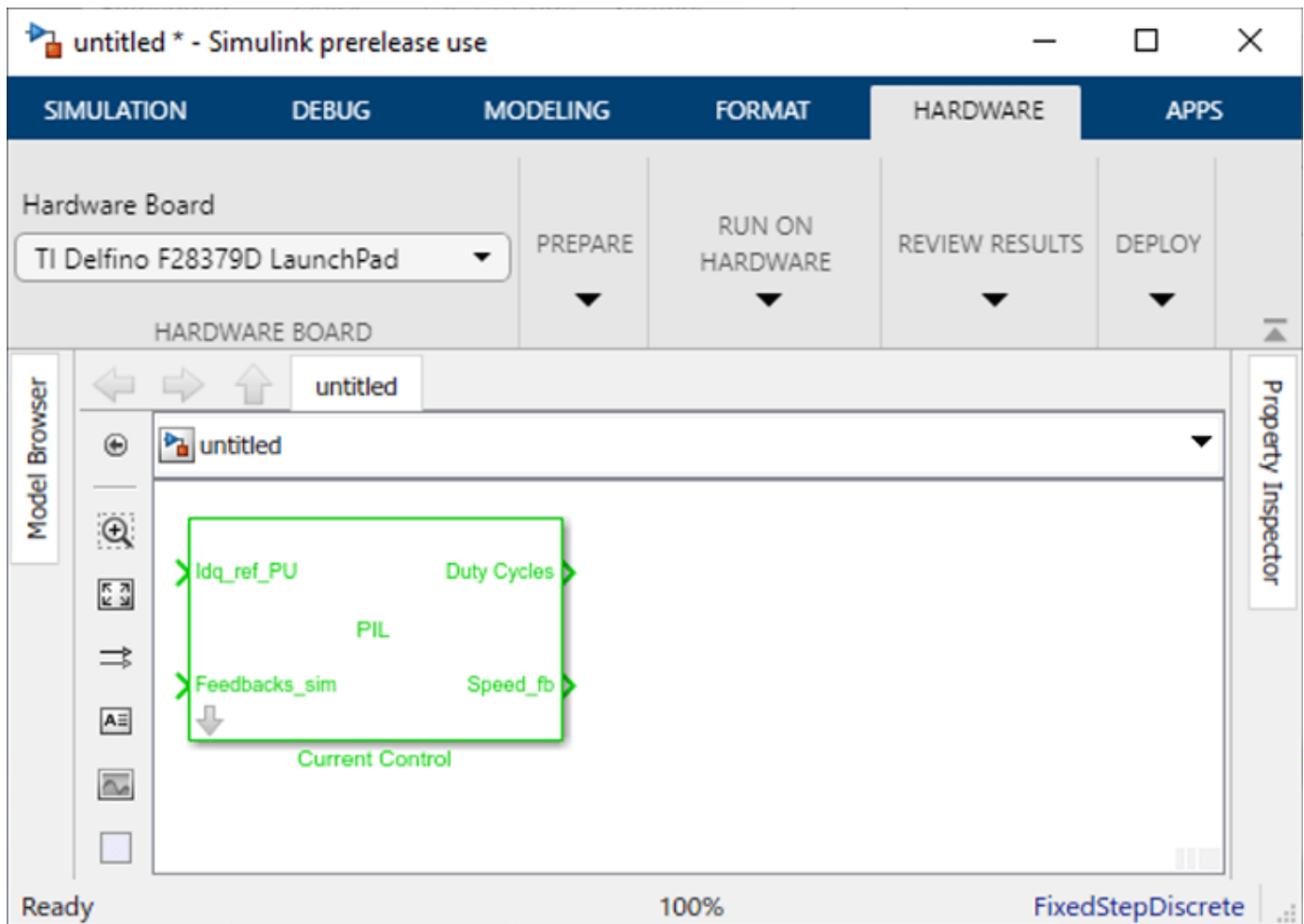
7. Rename the Current Control subsystem to Current Control (PIL).
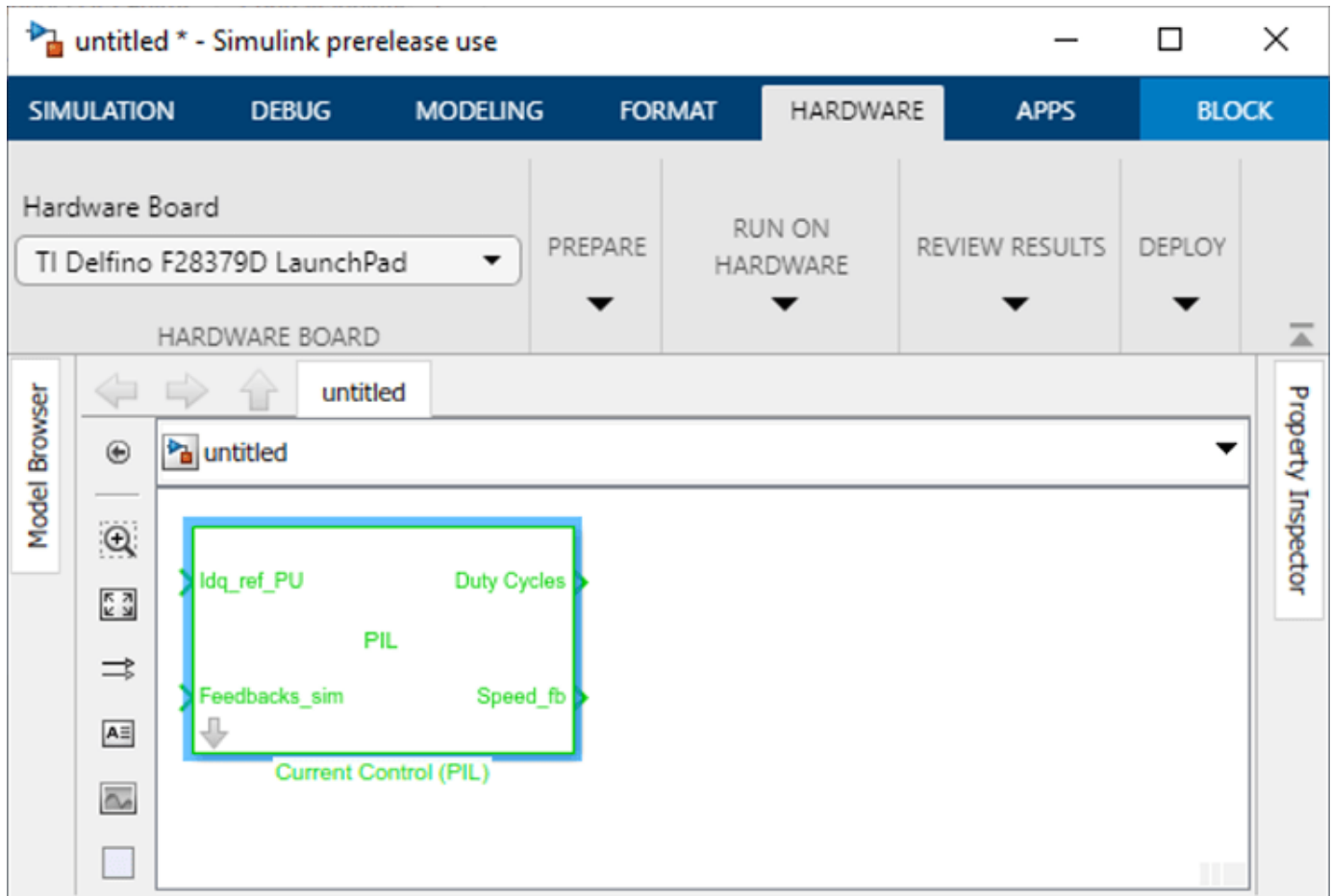
8. Copy the Current Control (PIL) subsystem and replace the Current Control subsystem in the `mcb_pmsm_foc_sim.slx` example model.

PMSM Field Oriented Control
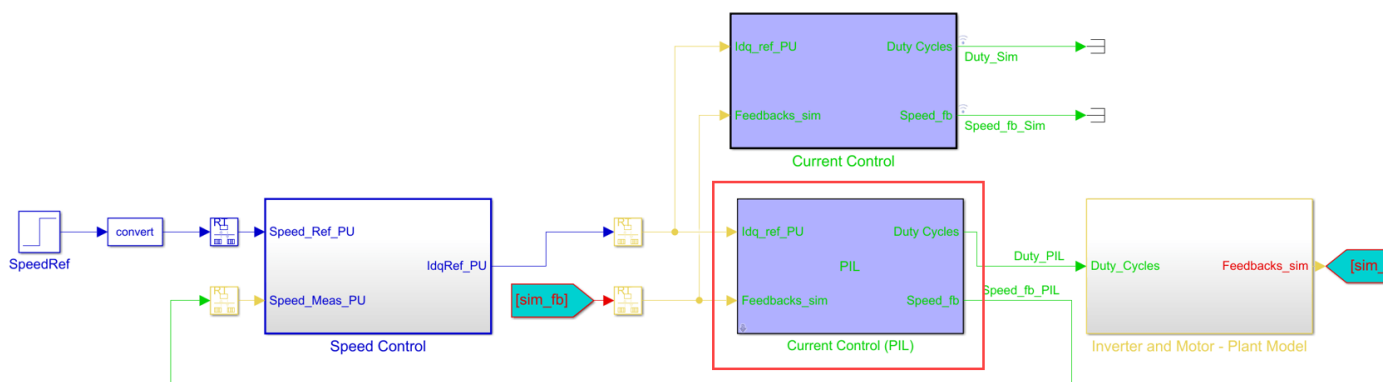
Motor Control Blockset v1.0
Copyright 2020 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Simulate this model

In the PIL mode, the system deploys the Current Control (PIL) subsystem to the target and executes the subsystem in the target hardware.

9. To compare the algorithm execution on the host machine simulation and in the PIL simulation, connect the Current Control subsystem parallely to the Current Control (PIL) subsystem. In addition, enable signal logging in the subsystem outputs.

PMSM Field Oriented Control

Motor Control Blockset v1.0
Copyright 2020 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Simulate this model

10. On the Simulink toolstrip, select the **SIL/PIL Manager** app from the **Apps** tab.
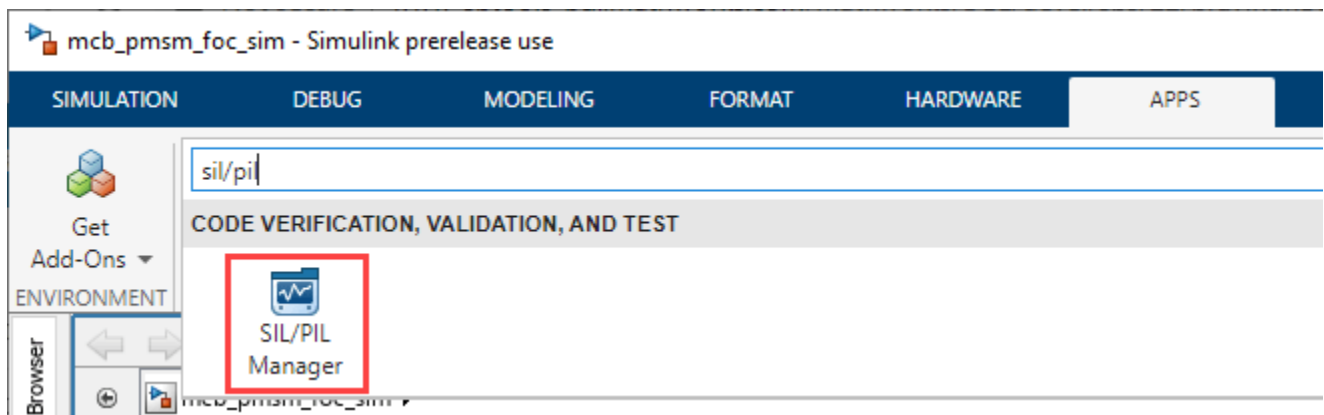
11. On the **SIL/PIL** toolstrip, select SIL/PIL Sim Only.

PMSM Field Oriented Control
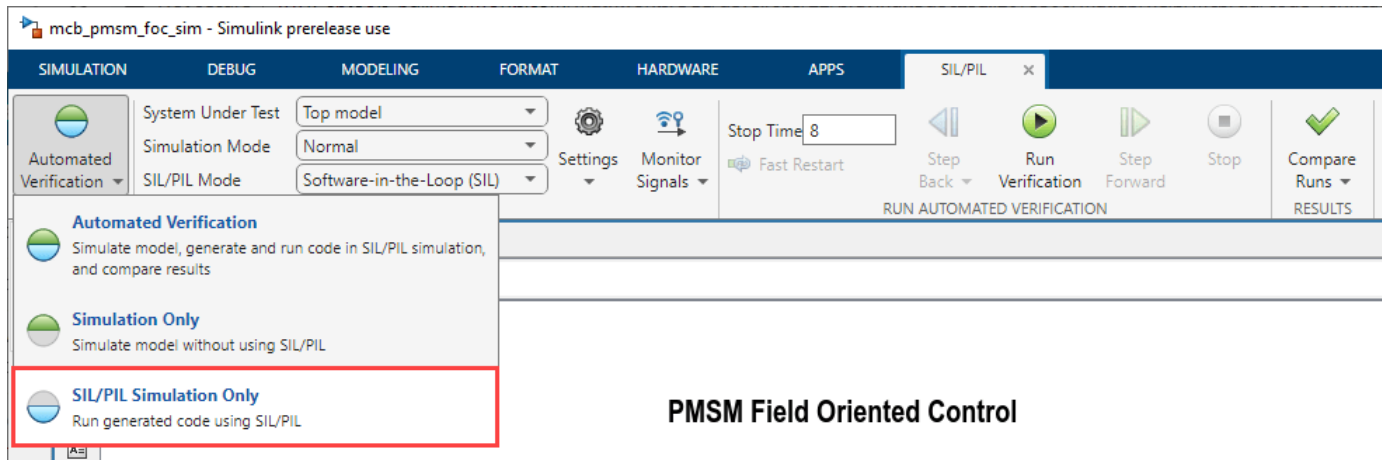
12. Select `Model blocks in SIL/PIL mode` in the **System Under Test** field.
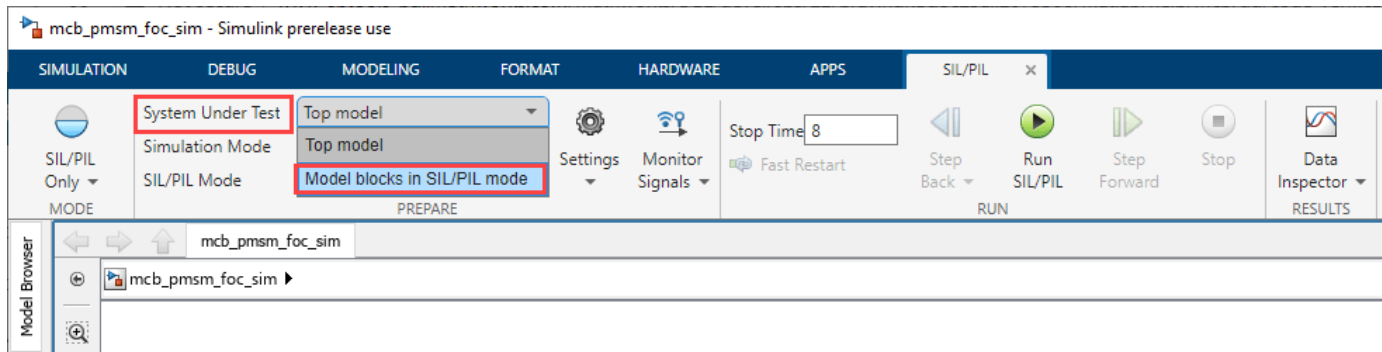
13. Click **Run SIL/PIL** on the **SIL/PIL** toolstrip to build the Current Control (PIL) subsystem and deploy it to the target.

After the system deploys the subsystem, the Current Control (PIL) subsystem executes on the target hardware processor, while the plant model runs on the host machine.

Analyze PIL Profiling Results

When PIL simulation ends, the system generates a profiling report.

Note: PIL simulation takes more time than the host-machine-based simulation. This is because of the serial communication (related to inputs and outputs of the Current Control (PIL) subsystem) between the host machine and subsystem that runs on the target hardware.

Code Execution Profiling Report for mcb_pmsm_foc_sim/Current Control1

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

Total time	54531910
Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	2e+08
Profiling data created	15-Jan-2021 13:00:17

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls			
[+] Current_initialize	1935	1935	1010	1010	1			
[+] Current_step [5e-05 0]	5560	5452	580	580	10001			
Current_terminate	140	140	140	140	1			

3. CPU Utilization [hide]

This profiling report, which is for the fixed-point datatype, shows the maximum and average execution times of the Current Control (PIL) subsystem running on the target hardware.

You can use the **Data Inspector** button on the **Simulation** tab to compare the signals logged during host-machine-based simulation and PIL simulation (executed on the target). This helps you verify the accuracy of host-machine-based simulation and PIL simulation.

This plot compares the speed feedback signals from the Current Control (PIL) and Current Control subsystems.
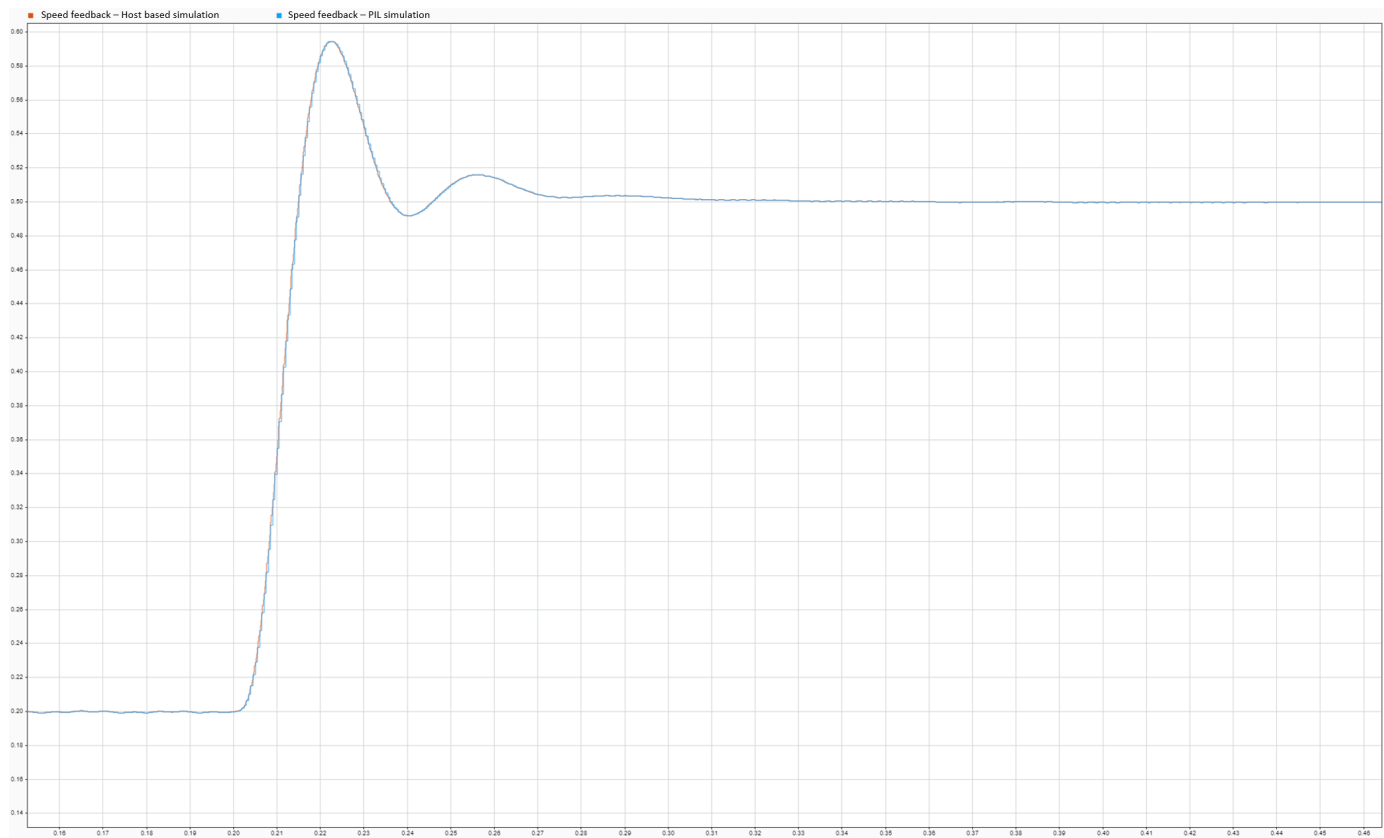
If the execution time exceeds 60% of the budgeted time, you can optimize the algorithm using one of these techniques:

- Execute from RAM.

- Offload some functionalities to CLA or other CPUs.

- Scale the algorithm to run at every alternate cycle.

- Move less critical functionalities like speed calculation to a slower rate.

For more details on SIL/PIL code verification, see:

- "Code Verification and Validation with PIL" (C2000 Microcontroller Blockset)

- "Create Execution-Time Profile for Generated Code" (Embedded Coder)

- "SIL/PIL Manager Verification Workflow" (Embedded Coder)

Field-Weakening Control (with MTPA) of Non-Linear PMSM Using Look-Up Table

This example uses a look-up table (LUT) for a non-linear PMSM and controller to run the motor using field-weakening control (with MTPA). Use this example to replicate and run a finite element analysis (FEA)-based non-linear high fidelity permanent magnet synchronous motor (PMSM) in simulation. This example helps motor design engineers to simulate high performance motors for real-world motor control applications. In addition, control system engineers can use this example to design control algorithms for a given set of motor parameter data to achieve high levels of accuracy in tracking and controlling speed and torque as well as meet efficiency requirements especially for high-performance motors.

The following image shows the MTPA and field-weakening operating points for an interior PMSM (IPMSM).
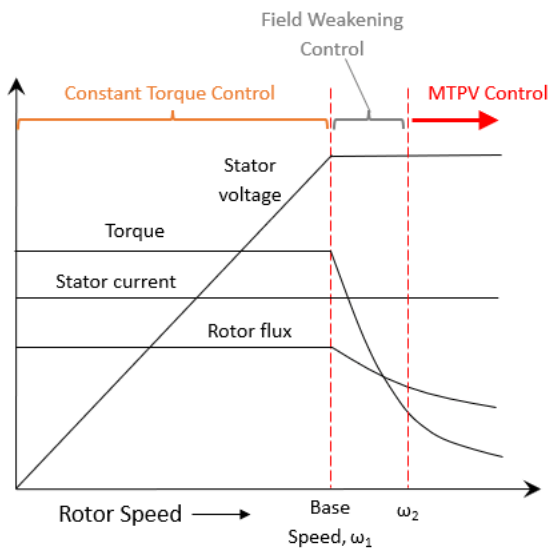
Figure a: IPMSM Characteristics for region highlighted in Figure b.
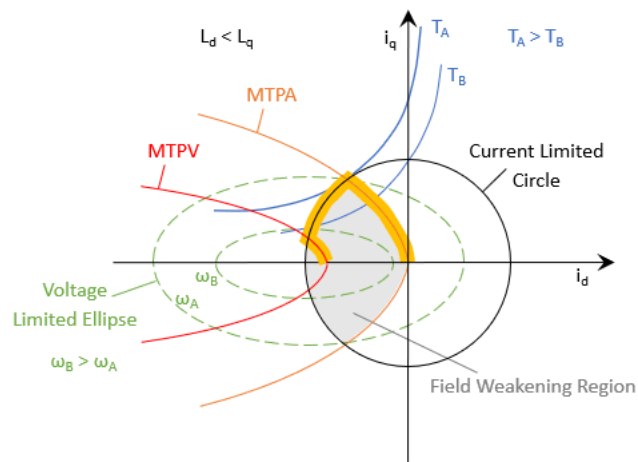
Figure b: Operating region for IPMSM

Using an actual PMSM and hardware to design a motor as well as the control algorithm can be impractical due to higher operation costs and time, safety issues, as well as sporadic machine and inverter failures. The example effectively solves these problems by using a simulation-based model driven by an LUT that contains motor parameter data. It uses an LUT that contains `Ld` and `Lq` data with respect to currents `id` and `iq`.

Traditionally, generation of optimal reference current data would require an actual motor and hardware in a dyno setup. The example reduces this effort by using the LUT based PMSM Control Reference block that characterizes the PMSM and obtains the machine's nonlinear inductance and permanent magnet flux linkage data, which can then be used to design and implement flux-weakening torque control `id`(torque, speed), `iq`(torque, speed) LUTs by using iterative analytical methods.

By default, the example uses pre-generated and calibrated inductance and permanent magnet flux linkage LUT data sets provided by JMAG. However, as an alternative, you can configure this block and

generate your own reference current `id`(torque, speed), `iq`(torque, speed) LUT and automatically use it in the same example.

You can use the example to simulate the LUT-based motor model and run tests to design the control algorithm. This enables you to run challenging tests easily and perform robust machine and control algorithm validation, which in turn reduces the time required for hardware testing and saves the overall development time.

The example model accepts pre-calibrated reference current LUT data files for non-linear PMSMs. Because solving the underlying motor equations for non-linear high-performance motors is computation-intensive, the example relies on LUTs generated by solving these (non-approximated) equations. This enables you to run control algorithms on the controller hardware with high accuracy (because LUT data mimics the actual non-linear motor characteristics).

Using this approach, you can reduce the complexities that occur when the algorithm relies on the controller hardware to solve approximated equations for non-linear PMSM.

The example helps you build a field-weakening (with MTPA) control algorithm that can compensate for the reduced torque caused by non-linearity developed due to a magnetically saturated PMSM stator core.

For more details about these operating regions, see "Field-Weakening Control (with MTPA) of PMSM" on page 4-48 and "PMSM Drive Characteristics and Constraint Curves".

Model

This example includes the following Simulink® model:

mcb_pmsm_nonlin_fwc

You can use this model only for simulation. Click the preceding link or use the following open_system command to open the model:

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

Interior-PMSM Field Weakening Control using FEM data

Note: This is a simulation-only example, and uses the data from JMAG-RT.

PlantSelection:Value

Motor Plant Model
- (●) Non-linear PMSM with 2D LUTs
- ( ) FEM Parameterized PMSM with 2D LUTs
- ( ) FEM Parameterized PMSM with 3D LUTs

Speed Reference (RPM)

Torque Reference (Nm)
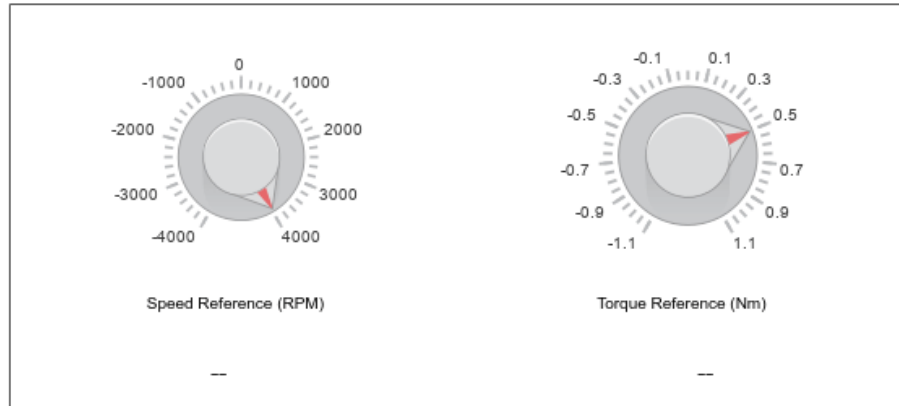
Steps:
1. To use a custom motor's FEM data, use the instructions
2. Select the motor plant model from the PlantSelection options
3. Simulate the model
4. Review results in Data Inspector
5. Learn more about this example.
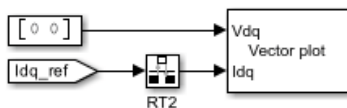
Feedbacks_sim   Duty_Cycles

Control System

Duty_Cycles   Feedbacks_sim

Inverter and Motor

[0 0]

Idq_ref

RT2

Vdq
Vector plot
Idq

Copyright 2022-2023 The MathWorks, Inc.
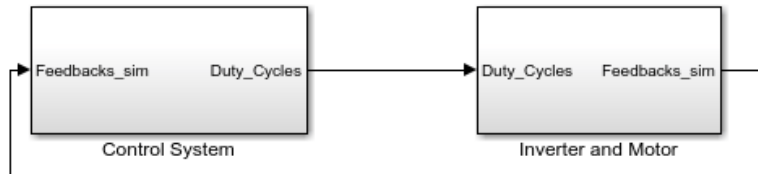
Required MathWorks® Products

- Motor Control Blockset™
- Simscape™ Electrical™

About LUT Data Files

By default, the example uses the following two LUT data files, which are loaded and read by the model initialization script `mcb_pmsm_nonlin_fwc_data.m` associated with the example model:

- `1k_S_C_IV.mat` — LUT from the product Simscape Electrical

- `JMAG_RTML-004_LdLq.mat` — Data generated from the inductance LUTs sourced from JMAG (for RTML-004 PMSM)

Note: Both the preceding `.mat` files include data for the same PMSM model from JMAG's model library.

For more details, see the **System Parameters** section of the model initialization script.

The following blocks in the example model uses the loaded LUT data:

- `mcb_pmsm_nonlin_fwc/Control System/Torque Control/LUT based PMSM Control Reference` — Uses the reference current LUTs derived from the inductance LUTs.

4-297
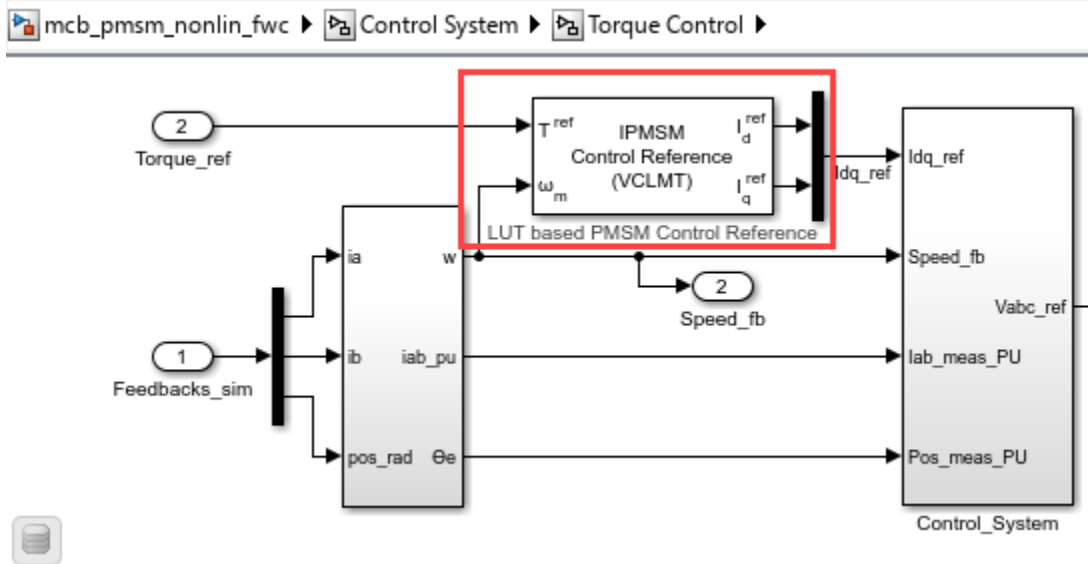
- `mcb_pmsm_nonlin_fwc/Control System/Torque Control/Control_System/ Current_Controllers/PMSM Feed Forward Control` — Uses the inductance LUTs.
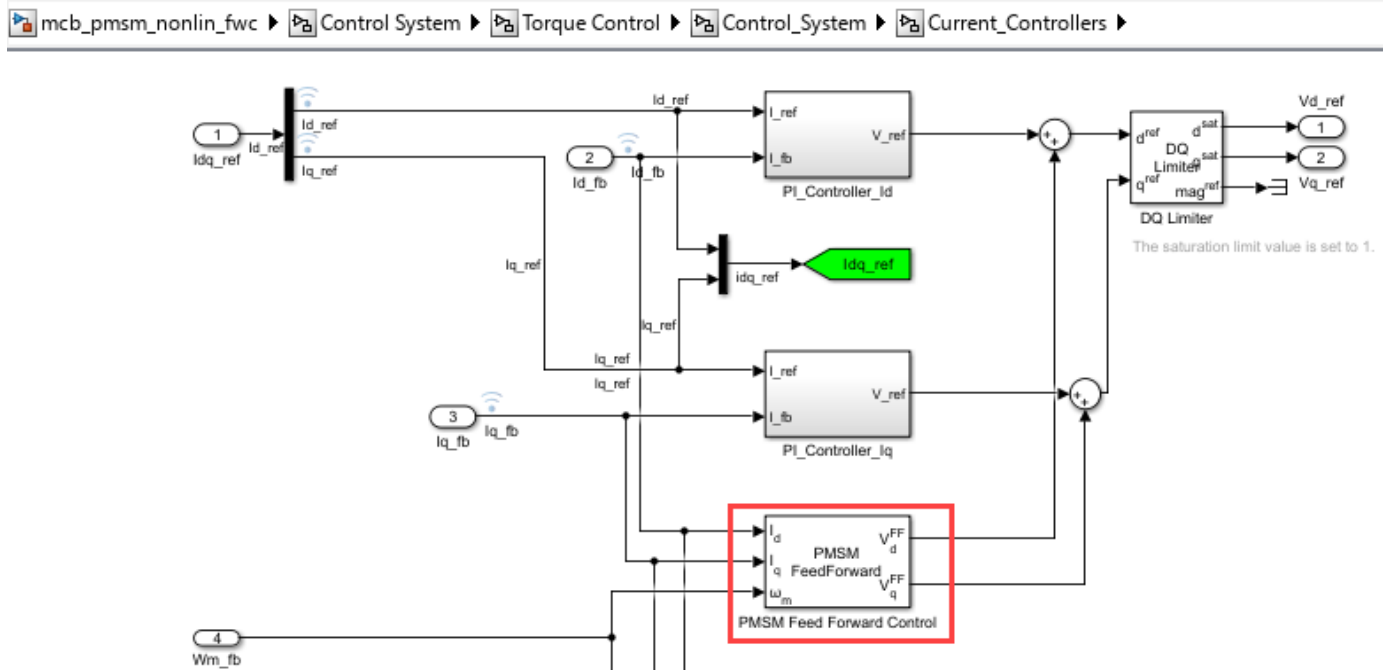
- `mcb_pmsm_nonlin_fwc/Inverter and Motor/Motor Model`

    - **Non-linear PMSM with 2D LUTs** — This motor model uses a PMSM (Simscape Electrical) block from Simscape Electrical that extracts and uses 2D data (`id`, `iq`) from the loaded `.mat` file.

    - **FEM Parameterized PMSM 2D LUTs** — This motor model uses the FEM-Parameterized PMSM (Simscape Electrical) block from Simscape Electrical that extracts and uses 2D data (`id`, `iq`) from the loaded `.mat` file.

- **FEM Parameterized PMSM 3D LUTs** — This motor model uses the FEM-Parameterized PMSM (Simscape Electrical) block from Simscape Electrical that extracts and uses 3D data (`imag`, `iangle`, `rotortheta`) from the loaded `.mat` file.
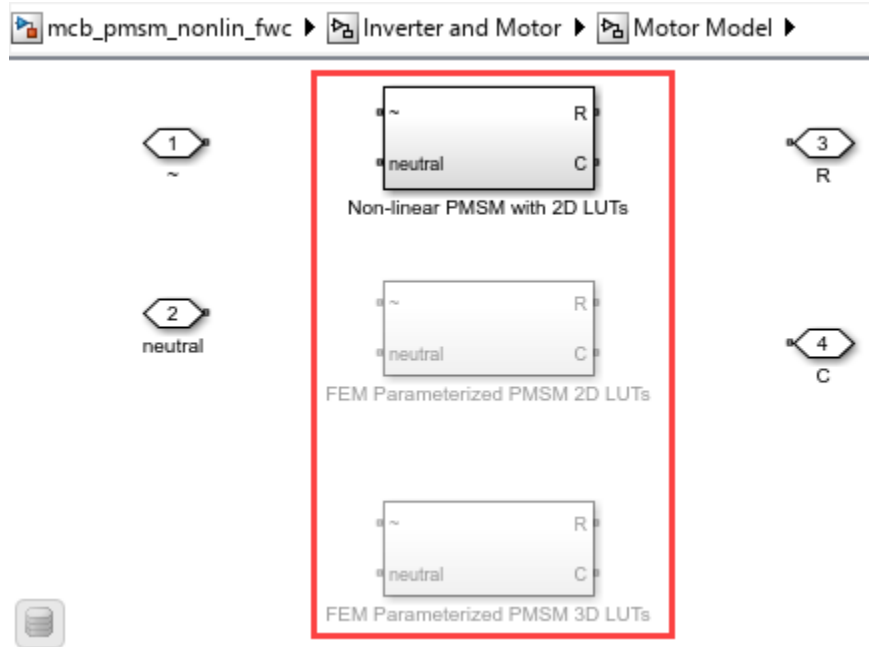
Simulate Model

1. Open the Simulink model `mcb_pmsm_nonlin_fwc.slx` associated with this example.

2. Use the **Motor Plant Model** radio button in the example model to select one of the following motor models that you want to simulate using the LUT data:

    a. Non-linear PMSM with 2D LUTs

    b. FEM Parameterized PMSM with 2D LUTs

    c. FEM Parameterized PMSM with 3D LUTs

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

4. Set the reference speed and reference torque values using the **Speed Reference (RPM)** and **Torque Reference (Nm)** knobs.

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

Generate LUT Data File for Custom Motor Design

The following procedure explains the steps needed to obtain the inductance LUT data set (either manually generated or downloaded from JMAG) and use it with the same or a similar example model.

1. Obtain the `.rtt` file and RT-Viewer tool:

    a. Visit JMAG-RT Model Library to download the LUT data set (`.rtt` file) for your motor, which includes the Ld, Lq maps.

Note: Downloading and using this file requires Type-2 (free) license that you can obtain by requesting for it.

b. Visit JMAG-RT Viewer to obtain the tool RT-Viewer.

2. Use the RT Viewer tool to open the `.rtt` file and export the Ld, Lq table values into an identical grid of the `id`, `iq` values. For example, save the Ld, Lq table values as `JMAG_Ld.csv`, and `JMAG_Lq.csv`.

- In the RT Viewer tool menu, select **Table Value > Inductance > Id-Iq-Ld**.

- In the Map Window menu, select **File > CSV Export** to open the CSV Export window.

- In the CSV Export window, provide `id`, `iq` grid details and the file name to proceed with the data export.

Note: If you have a manually generated LUT data set, skip steps 1 and 2.

3. In the model initialization script associated with the Simulink model, create template structures for PMSM and inverter as shown below:

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
```

```
inverter= mcb_SetInverterParameters("BoostXL-DRV8305");
```

```
pmsm.B=pmsm.B*100; % increasing the viscous damping coefficient since
this is a bigger motor than the Teknic2310p.
```

4. In the script, find the motor parameters and update the following fields:

```
pmsm.p (pole pairs)
```

```
pmsm.Rs (stator/Coil Resistance)
```

```
pmsm.Ld (d_inductance)
```

```
pmsm.Lq (q_inductance)
```

```
pmsm.J (inertia)
```

```
pmsm.FluxPM (magflux)
```

```
pmsm.I_rated (rated_current)
```

```
inverter.V_dc (rated_voltage)
```

5. Calculate the rated torque value using the following function:

```
pmsm.T_rated=mcbPMSMRatedTorque(pmsm);
```

6. Create a seed structure with the file paths to read the .csv files that you created in step 2.

```
seed.Ldfilepath='JMAG_Ld.csv';
```

```
seed.Lqfilepath='JMAG_Lq.csv';
```

Note: If you are using a manually generated inductance LUT data set, use the paths to the associated `.csv` files instead.

7. Create LUTs using the following API:

```
PMSMLUT=mcbGenerateTables(pmsm,inverter,'jmagindfiles2inddq',seed);
```

The preceding command converts the `.csv` files generated by the JMAG RT-Viewer into a regular matrix, filling the empty places using extrapolation. Because we also need the permanent magnet flux linkage (FluxPM) in the same LUT format, create an array using the following command:

```
PMSMLUT.FluxPMTable=pmsm.FluxPM*ones(size(PMSMLUT.LdTable));
```

8. Configure the following blocks in the example model to use the non-linear LUTs:

- LUT based PMSM Control Reference — In the block dialog box, set the parameter **Motor parameter input method** to `Non-linear model with id and iq LUTs`.

- PMSM Feed Forward Control — In the block dialog box, set the parameter **Motor parameter input method** to `Non-linear model with Ld, Lq and FluxPM LUTs`.

- PMSM Torque Estimator — In the block dialog box, set the parameter **Motor parameter input method** to `Non-linear model with Ld, Lq and FluxPM LUTs`.

9. In the example model, create a plant using the FEM-Parameterized PMSM block.

10. In the FEM-Parameterized PMSM block dialog box, set the **Modelling option** parameter to *2-D partial derivative data | No thermal port*.

11. In the **Electrical** section of the dialog-box, set the **Parameterization** parameter to `Assume sinusoidal back emf - tabulate with d- and q- axis currents`.

12. To generate the `id`, `iq` LUTs from inductance LUT data, use the following command:

```
PMSMLUT=mcbGenerateTables(pmsm,inverter,'idiqLUTs',seed);
```

For more details about structure of the variable seed, use the MATLAB command `help mcbGenerateTables`.

13. In the model initialization script, populate the PMSM LUT fields at appropriate places depending on the parameter input method:

```
PMSMLUT.idVec
```

```
PMSMLUT.iqVec
```

```
PMSMLUT.LdTable
```

```
PMSMLUT.LqTable
```

```
PMSMLUT.FluxPMTable
```

or

```
PMSMLUT.trefVec
```

```
PMSMLUT.wrpmVec
```

```
PMSMLUT.idTable
```

```
PMSMLUT.iqTable
```

14. Similarly, populate the rest of the motor parameters at their respective places.

15. For the PMSM plant model:

- Maintain the default value for the stator zero sequence inductance.

- Update the variables `pmsm.B` and `pmsm.J` in the mechanical parameters section.

- Set initial targets for rotor speed and rotor position to `0`.

Related Examples

- "PMSM Drive Characteristics and Constraint Curves"

- "Field-Weakening Control (with MTPA) of PMSM" on page 4-48

See Also

- LUT based PMSM Control Reference

- Designing a Torque Controller for a PMSM through Simulation on a Virtual Dynamometer

Dwell Angle Computation for SRM Speed Control

This example computes the dwell angle of a 12/8 Switched Reluctance Motor (SRM) online and uses it with a delta controller to control the motor speed.

SRMs need commutation switching sequences to run. These sequences determine when the motor phases can be switched On and Off, and therefore, run the motor while controlling the motor speed. Each commutation switching sequence (which forms a pulse train) can be used to control (turn On or Off) the corresponding motor phase. The activation period defined by the turn On (θ_{On}) and turn Off (θ_{Off}) angles is known as the dwell angle.

Dwell angle is one of the control variables for the motor. The optimal dwell angle depends on the motor parameters and the control objective.

A large dwell angle may result in overlapped phase current pulses, which results in simultaneous activation of adjacent phases leading to greater power consumption. This enhances motor performance by reducing torque ripples as shown below. However, a larger dwell angle makes the motor less energy efficient.
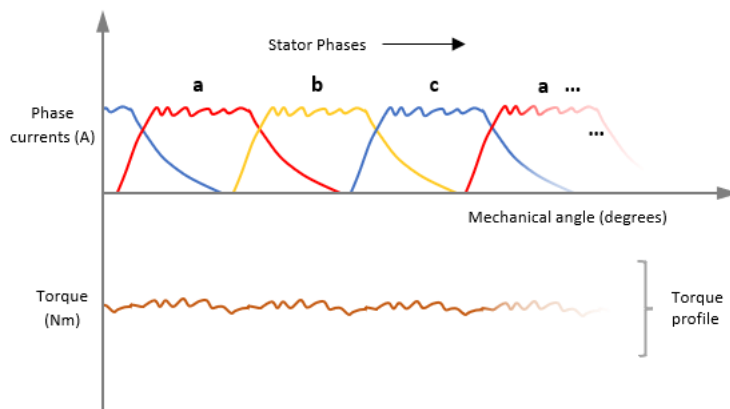
A small dwell angle leads to a reduced overlap of phase current pulses. This causes the motor to draw less power, and therefore, become more energy efficient. However, this usually results in larger torque ripples as shown below, which may impact motor performance.
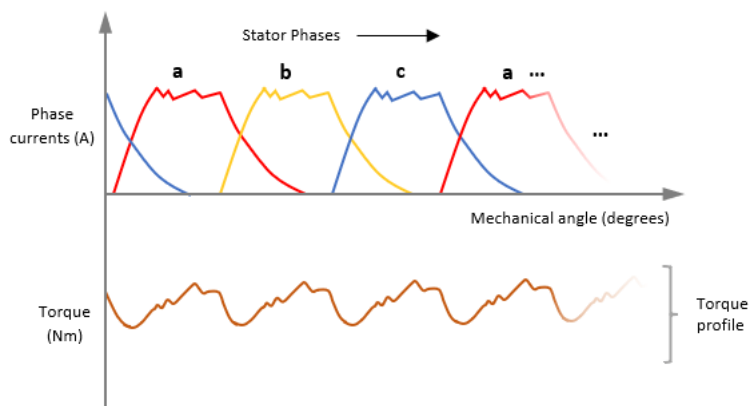
This example computes an optimum dwell angle such that it achieves a balance between motor energy efficiency and motor torque ripple amplitude. For a given reference speed and load condition, the example computes the optimum dwell angle using only a few SRM design parameters.

The following image shows an example of torque profile corresponding to an optimized dwell angle.
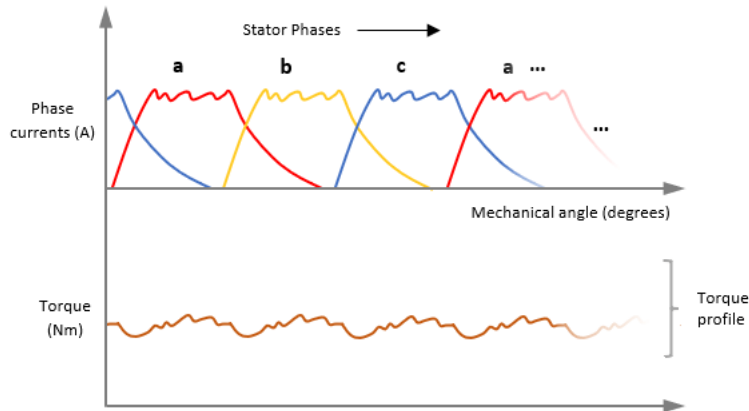
The example computes the dwell angle using the angles θ_{01} and θ_e, as shown below.
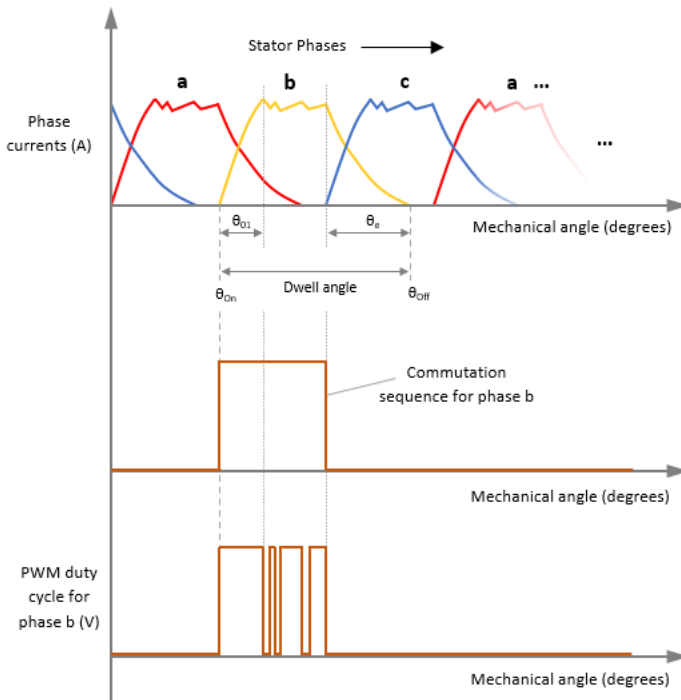
θ_{01} is the angle measured between phase activation and the point when the initial rise in phase current ends. θ_e is the angle measured between phase shutdown and the point when the phase current is zero. For a given phase, usually $\theta_e > \theta_{01}$ due to the rising inductance profile in an SRM.

Note: The example computes θ_{01} and θ_e only for phase A, and assumes that all the phases behave identically.

The example model uses a PI controller (for speed control) and a delta controller (for current control), along with an online dwell angle computation algorithm to optimize performance of the SRM. The example uses an easy-to-implement, low-runtime, and computationally inexpensive delta controller for current control. The current control loop runs at 50KHz (20µs). In addition, the example uses the necessary filters to enable smoother transitions for θ_{On} and θ_{Off}.

Note: Using online dwell angle computation to control SRM speed may reduce the range of achievable torque and speed.

Alternatively, you may use this example to run the SRM with delta current controller using a fixed dwell angle by specifying θ_{On} and θ_{Off} in the model initialization script.

Model

The example includes the model mcb_srm_dwellctrl_f28379d.

You can use the model for both simulation and code generation. To open a Simulink® model, you can also use the open_system command at the MATLAB® command prompt. For example, use this command for the F28379D based controller:

```
open_system('mcb_srm_dwellctrl_f28379d.slx');
```
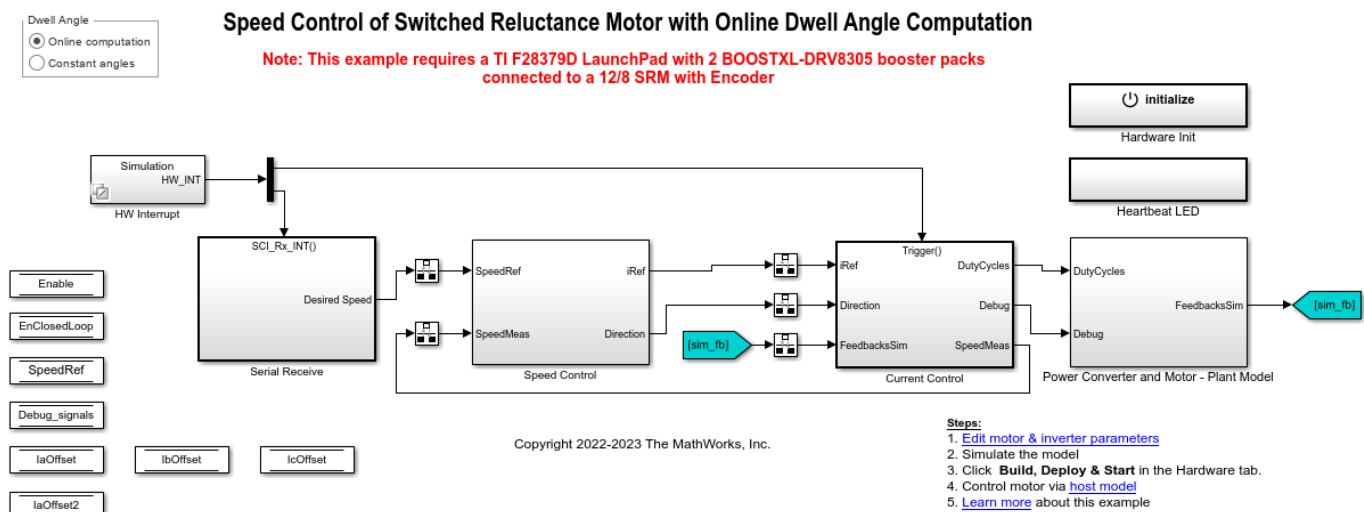

For details about the supported hardware configuration, see Required Hardware in the Generate Code and Deploy Model to Target Hardware section.

Required MathWorks® Products

To simulate model:

- Motor Control Blockset™
- Simscape™ Electrical™

To generate code and deploy model:

- Motor Control Blockset™

- Fixed-Point Designer™ (only needed for optimized code generation)
- Embedded Coder®
- C2000™ Microcontroller Blockset

Prerequisites

Obtain the motor parameters. The example model includes default SRM parameters. If required, you can replace them with values from a motor datasheet or other sources.

Update SRM parameters in the model initialization script associated with the Simulink model. For instructions, see "Estimate Control Gains and Use Utility Functions" on page 3-2.

If required, update the following parameters in the model initialization script:

- `PWM_frequency` — Enter the switching frequency of PWM. Note that you may need to tune controller gains again if you change this value.

- `Kp` and `Ki` — Enter the PI controller gains. Note that the example does not automatically compute these values. Set these gains according to the hardware setup.

- `thetaON` — Enter the motor phase turn-on electrical position at which the switching sequence turns 0 to 1 and energizes the corresponding phase. The model uses this value only when you set the **Dwell Angle** button in the target model to `Constant angles`.

- `thetaOFF` — Enter the motor phase turn-off electrical position at which the switching sequence turns 1 to 0 and de-energizes the corresponding phase. The model uses this value only when you set **Dwell Angle** button in the target model to `Constant angles`.

Note: The example supports only single data type and per-unit computation.

Simulate Model

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

1. Open the target 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 shows how to generate code and run the control 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

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB command prompt.

- LAUNCHXL-F28379D controller + 2 BOOSTXL-DRV8305 inverters: mcb_srm_dwellctrl_f28379d

The example uses the ADC current sensors available in the inverter boards to measure the 3 phase motor currents needed for controlling SRM.

Hardware Connections

The example requires 2 BOOSTXL-DRV8305 inverters connected in the following configuration.

For more information about hardware connections for SRM, see "Instructions for Switched Reluctance Motor (SRM) Setup" on page 7-11.

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.

4. 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.

5. 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.

6. Use the **Dwell Angle** button to select one of the following options:

• `Online computation` — Select this option to enable online dwell angle computation algorithm and use the dwell angle computed in real-time for speed control of SRM.

- `Constant angles` — Select this option to disable online dwell angle computation and use the variables `thetaON` and `thetaOFF` (that define a fixed dwell angle) in the model initialization script for speed control of SRM.

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

8. Observe and verify the variables populated by the target model in the base workspace.

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. Use this command for a F28379D based controller.

open_system('mcb_srm_spdctrl_host.slx');

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

10. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. This variable updates the **Port** parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

11. Update the reference speed value in the **Reference Speed (RPM)** field in the host model. We recommend that you enter lower speeds initially during motor start up and gradually increase the speed later.

12. In the host model, select the debug signals that you want to monitor.

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

14. After you connect the load to the motor shaft, change the position of the **Motor** switch to **Start**, to start running the motor.

If the motor does not align or if it fails to start running in open-loop mode, increase the values of the start-up parameters listed in the **Motor Start Up Parameters** section of the model initialization script.

15. The example performs the following procedures:

- Calibrate the quadrature encoder sensor (which includes alignment of the rotor with stator phase A).

- Use the fixed dwell angle defined in the model initialization script to run SRM in closed loop according to **Reference Speed (RPM)**.

OR

Compute optimal dwell angle according to the load conditions and uses it to run SRM in closed loop according to **Reference Speed (RPM)**.

16. Observe the debug signals received from target in the **Scope** available in the host model.

References

[1] C. Mademlis and I. Kioskeridis, "Performance optimization in switched reluctance motor drives with online commutation angle control," in IEEE Transactions on Energy Conversion, vol. 18, no. 3, pp. 448-457, Sept. 2003, doi: 10.1109/TEC.2003.815854.

See Also

- SRM Commutation

- "Commutation of SRM Using Sensor Feedback" on page 4-253

Estimate Induction Motor Parameters Using Parameter Estimation Blocks

This example uses the parameter estimation blocks provided by Motor Control Blockset™ to estimate these parameters of an AC induction motor (ACIM):

- Magnetizing current, *Id0* (A)

- Stator resistance, *Rs* (Ohm)

- Rotor resistance, *Rr* (Ohm)

- Magnetizing inductance, *Lm* (Henry)

- Stator leakage inductance, *Lls* (Henry)

- Rotor leakage inductance, *Llr* (Henry)

- Motor inertia, *J* (Kg.m^2)

- Friction constant, *B* (N.m.s)

Motor parameter estimation is vital for implementing motor control algorithms accurately. Accurate motor parameters enable the algorithm to compute the control parameters with precision. Therefore, an accurate representation of motor parameters results in efficient speed tracking when you run induction motors using control techniques such as field-oriented control (FOC). Motor parameter estimation also enables you to verify the parameter values provided by the motor datasheet. In addition, it enables you to accurately replicate the plant model in Simulink using which you can simulate real-world scenarios and tests that are difficult to execute using a physical hardware setup.

An accurate simulation helps you reconfigure and run these tests multiple times, which can be difficult when you use physical hardware. Therefore, these exhaustive tests enable you to predict, find, and fix problems including hard-to-reproduce failures with ease.

The example demonstrates usage of the following blocks to design and implement an induction motor parameter estimation algorithm on a Texas Instruments™ hardware:

- ACIM Parameter Estimation Configurator

- Id0 Estimator

- ACIM Rs Estimator

- RrL Estimator

- ACIM Mechanical Parameter Estimation

Alternatively, using this example as a reference, you can utilize these blocks to implement a similar algorithm on a different hardware or configuration.

The target model included in the example uses the preceding parameter estimation blocks inside the subsystem mcb_acim_param_est_f28379d/Parameter Estimation/Param Est:
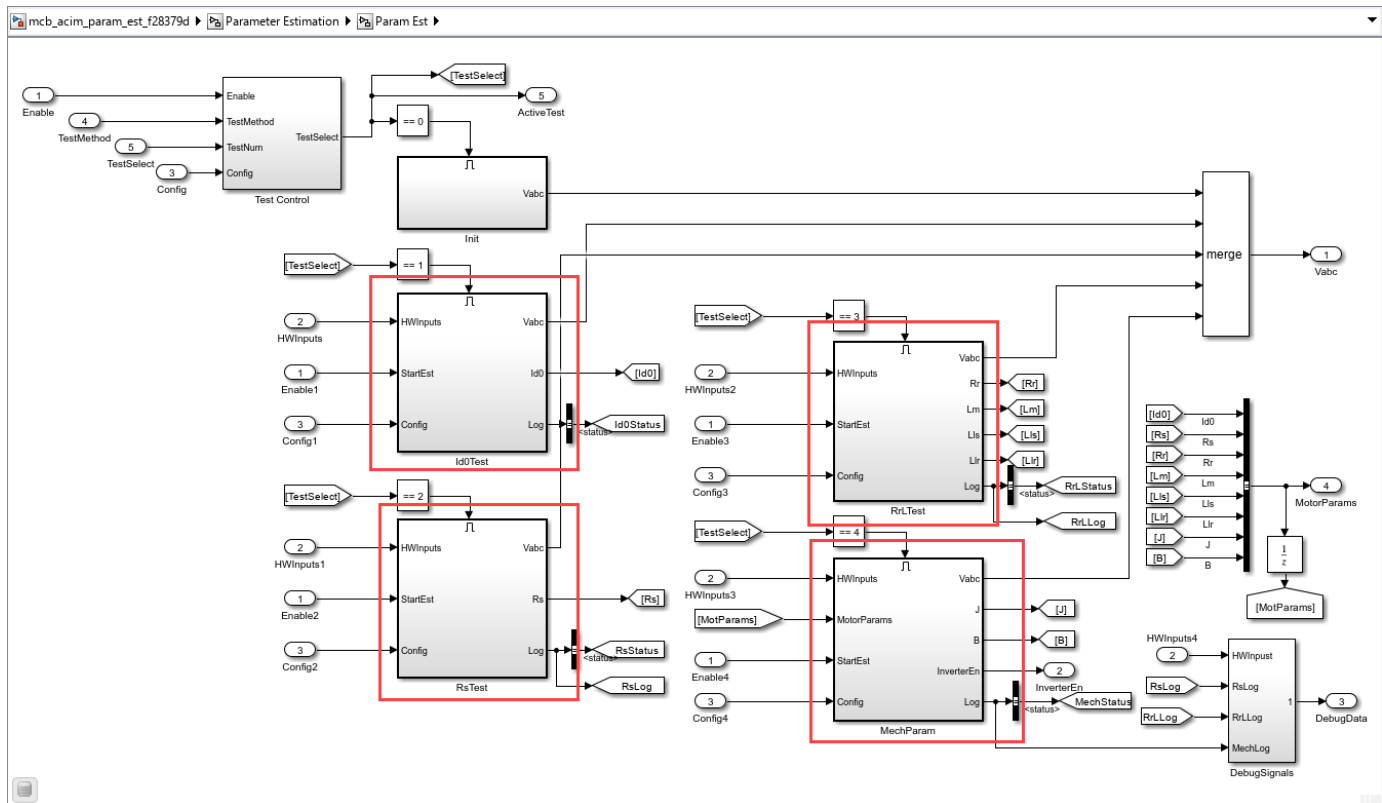
The following subsystems use the parameter estimation blocks to estimate *Rs*, *Ld*, *Lq*, and mechanical parameters:

- `Id0Test` — The subsystem uses the Id0 Estimator block to estimate the magnetizing current.

- `RsTest` — The subsystem uses the ACIM Rs Estimator block to estimate the stator resistance.

- `RrLTest` — The subsystem uses the RrL Estimator block to estimate rotor resistance, magnetizing inductance, stator leakage inductance, and rotor leakage inductance.

- `MechParam` — The subsystem uses the ACIM Mechanical Parameter Estimation block to estimate motor inertia and friction constant.

If you are only simulating the example, the target model displays the estimated parameters. When you deploy the target model to the hardware, use the host model to estimate the parameters using hardware.

Model

The example includes the model mcb_acim_param_est_f28379d.

You can use the model for both simulation and code generation. To open the Simulink® model, you can also use the open_system command at the MATLAB® command prompt:

```
open_system('mcb_acim_param_est_f28379d.slx');
```

For details about 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

- Stateflow® (needed only if you modify the example model)

To generate code and deploy model:

- Motor Control Blockset

- Embedded Coder®

- C2000™ Microcontroller Blockset

- Stateflow (needed only if you modify the example model)

Prerequisite

Check and update the mandatory motor, inverter, and other system parameters listed in the model initialization script `mcb_acim_param_est_f28379d_data.m`.

In addition, you can update the optional motor parameters for simulation.

You can use the inverter and motor hardware datasheet to determine these values.

Simulate Model

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

1. Open the target model included in this example.

2. If required, change the configuration elements for parameter estimation, by updating the mask parameters of the ACIM Parameter Estimation Configurator block available in the `mcb_acim_param_est_f28379d/Serial Receive/Variant Subsystem/Sim` subsystem. For more details, see ACIM Parameter Estimation Configurator.

3. In the User Controls section, ensure that the **Parameter Estimation** switch is set to **Stop**.

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

5. Set the **Test Method** switch to either **Manual** or **Auto** mode.

- **Manual** — This mode enables you to manually execute the tests one-by-one in the prescribed sequence.

- **Auto** — This mode enables you to automatically run all the tests in the prescribed sequence.

If you are using **Manual** mode:

a. Set the **Test Method** switch to **Manual**.

b. Use the **Test Select** button to select the first test `Id0`.

c. Turn the **Parameter Estimation** switch to **Start**. The model runs the *Id0* estimation test after which it displays the computed value in the **Estimated Parameters** field.

d. Turn the **Parameter Estimation** switch to **Stop**.

e. Similarly, use the **Test Select** button to select the next test and repeat steps c to e.

Note: Ensure that you run all tests in the prescribed sequence of `Id0 > Rs > (Rr, Lm, Lls, Llr) > (J, B)`. Do not skip any test.

During simulation, the target model does not support computation of *J*.

If you are using Auto mode:

a. Set the **Test Method** switch to **Auto**.

b. Turn the **Parameter Estimation** switch to **Start**. The model starts running all tests in the prescribed sequence of `Id0 > Rs > (Rr, Lm, Lls, Llr) > (J, B)` after which it displays the computed parameter values in the **Estimated Parameters** field.

c. Turn the **Parameter Estimation** switch to **Stop**.

Note: In the **Auto** mode, the **Test Select** button selection has no effect.

During simulation, the target model does not support computation of *J*.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the parameter estimation algorithm on the target hardware to estimate parameters of the connected motor.

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 to run the parameter estimation tests.

Required Hardware

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_acim_param_est_f28379d

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. 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 4-10.

3. 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 4-79.

Note: Verify the number of slits available in the quadrature encoder sensor attached to your motor. Check and update the variable *pmsm.QEPSlits* available in the model initialization script. This variable corresponds to the **Encoder slits** parameter of the quadrature decoder block. For more details about the **Encoder slits** and **Encoder counts per slit** parameters, see Quadrature Decoder.

4. 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.

5. 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.

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

7. Verify that the variables from the target model are available in the MATLAB base workspace.

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_acim_param_est_host_f28379d.slx');
```

ACIM Parameter Estimation Host

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

9. If required, change the configuration elements for parameter estimation, by updating the mask parameters of the ACIM Parameter Estimation Configurator block available in the host model. For more details, see ACIM Parameter Estimation Configurator.

10. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. This variable updates the **Port** parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

11. In the User Controls section, ensure that the **Parameter Estimation** switch is set to **Stop**.

12. Click **Run** on the Simulation tab to run the host model. This initiates communication between the host model and target hardware.

13. Use the **Test Method** switch to select either **Manual** or **Auto** mode.

    - **Manual** — This mode enables you to manually execute the tests one-by-one in the prescribed sequence.

    - **Auto** — This mode enables you to automatically run all the tests in the prescribed sequence.

    If you are using **Manual** mode:

        a. Set the **Test Method** switch to **Manual**.

        b. Use the **Test Select** button to select the first test `Id0`.

4-315

c. Turn the **Parameter Estimation** switch to **Start**. The model runs the *Id0* estimation test after which it displays the computed value in the host model.

d. Turn the **Parameter Estimation** switch to **Stop**.

e. Similarly, use the **Test Select** button to select the next test and repeat steps c to e.

Note: Ensure that you run all tests in the prescribed sequence of `Id0 > Rs > (Rr, Lm, Lls, Llr) > (J, B)`. Do not skip any test. The host model uses the **Active Test** field to display the test that is currently running.

If you are using **Auto** mode:

a. Set the **Test Method** switch to **Auto**.

b. Turn the **Parameter Estimation** switch to **Start**. The model starts running all tests in the prescribed sequence of `Id0 > Rs > (Rr, Lm, Lls, Llr) > (J, B)` after which it displays the computed parameter values in the host model.

c. Turn the **Parameter Estimation** switch to **Stop**.

Note: In the **Auto** mode, the **Test Select** button selection has no effect.

The host model uses the **Active Test** field to display the test that is currently running.

14. Use the **Debug Signal** combo box to select a debug signal. Monitor the selected signal in the scope.

Note: The model can interrupt the currently running test and turn the following LEDs red to protect the hardware from the following faults:

- **Over-current fault** — This fault occurs when the actual current drawn from the power supply is more than the overcurrent limit specified by the variables *motor.OverCurrLim* and *motor.I_rated* in the model initialization script.

- **Under-voltage fault** — This fault occurs when input DC voltage drops below the undervoltage limit specified by the variables *inverter.UnderVoltLim* and *inverter.V_dc* in the model initialization script.

After the parameter estimation tests are complete, the host model saves the computed parameters in the *motorParam* variable inside the base workspace.

Estimate PMSM Parameters Using Parameter Estimation Blocks

This example uses the parameter estimation blocks provided by Motor Control Blockset™ to estimate these parameters of a permanent magnet synchronous motor (PMSM) with a quadrature encoder sensor:

- Phase resistance, *Rs* (Ohm)

- D-axis inductance, *Ld* (Henry)

- Q-axis inductance, *Lq* (Henry)

- Back-EMF constant, *Ke* (Vpk_LL/krpm, where Vpk_LL is the peak voltage line-to-line measurement)

- Motor inertia, *J* (Kg.m^2)

- Friction constant, *B* (N.m.s)

Motor parameter estimation is vital for implementing motor control algorithms accurately. Accurate motor parameters enable the algorithm to compute the control parameters with precision. Therefore, an accurate representation of motor parameters results in efficient speed tracking when you run PMSMs using control techniques such as field-oriented control (FOC). Motor parameter estimation also enables you to verify the parameter values provided by the motor datasheet. In addition, it enables you to accurately replicate the plant model in Simulink using which you can simulate real-world scenarios and tests that are difficult to execute using a physical hardware setup.

An accurate simulation helps you reconfigure and run these tests multiple times, which can be difficult when you use physical hardware. Therefore, these exhaustive tests enable you to predict, find, and fix problems including hard-to-reproduce failures with ease.

The example demonstrates usage of the following blocks to design and implement a PMSM parameter estimation algorithm on a Texas Instruments™ hardware:

- PMSM Parameter Estimation Configurator

- PMSM Rs Estimator

- Ld Estimator

- Lq Estimator

- PMSM Mechanical Parameter Estimation

Alternatively, using this example as a reference, you can utilize these blocks to implement a similar algorithm on a different hardware or configuration.

The target model included in the example uses the preceding parameter estimation blocks inside the subsystem mcb_pmsm_param_est_f28379d/Parameter Estimation/Param Est:

The following subsystems use the parameter estimation blocks to estimate *Rs*, *Ld*, *Lq*, and mechanical parameters:

- RsTest — The subsystem uses the Rs Estimator block to estimate the motor resistance.

- LdTest — The subsystem uses the Ld Estimator block to estimate *d*-axis inductance.

- LqTest — The subsystem uses the Lq Estimator block to estimate *q*-axis inductance.

- MechParam — The subsystem uses the PMSM Mechanical Parameter Estimation block to estimate back-emf constant, motor inertia, and friction constant.

If you are only simulating the example, the target model displays the estimated parameters. When you deploy the target model to the hardware, use the host model to estimate the parameters using hardware.

Model

The example includes the model mcb_pmsm_param_est_f28379d.

You can use the model for both simulation and code generation. To open the Simulink® model, you can also use the open_system command at the MATLAB® command prompt:

```
open_system('mcb_pmsm_param_est_f28379d.slx');
```

For details about 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

- Stateflow® (needed only if you modify the example model)

To generate code and deploy model:

- Motor Control Blockset

- Embedded Coder®

- C2000™ Microcontroller Blockset

- Stateflow (needed only if you modify the example model)

Prerequisite

Check and update the mandatory motor, inverter, and other system parameters listed in the model initialization script `mcb_pmsm_param_est_f28379d_data.m`.

In addition, you can update the optional motor parameters for simulation.

You can use the inverter and motor hardware datasheet to determine these values.

Simulate Model

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

1. Open the target model included in this example.

2. If required, change the configuration elements for parameter estimation, by updating the mask parameters of the PMSM Parameter Estimation Configurator block available in the `mcb_pmsm_param_est_f28379d/Serial Receive/Variant Subsystem/Sim` subsystem. For more details, see PMSM Parameter Estimation Configurator.

3. In the **User Controls** section, ensure that the **Parameter Estimation** switch is set to **Stop**.

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

5. Set the **Test Method** switch to either **Manual** or **Auto** mode.

   - **Manual** — This mode enables you to manually execute the tests one-by-one in the prescribed sequence.

   - **Auto** — This mode enables you to automatically run all the tests in the prescribed sequence.

   If you are using **Manual** mode:

   a. Set the **Test Method** switch to **Manual**.

   b. Use the **Test Select** button to select the first test `Rs`.

   c. Turn the **Parameter Estimation** switch to **Start**. The model runs the *Rs* estimation test after which it displays the computed value in the **Estimated Parameters** field.

   d. Turn the **Parameter Estimation** switch to **Stop**.

   e. Similarly, use the **Test Select** button to select the next test and repeat steps c to e.

   Note: Ensure that you run all tests in the prescribed sequence of `Rs > Ld > Lq > (Ke, J, B)`. Do not skip any test.

   During simulation, the target model does not support computation of *J*.

   If you are using **Auto** mode:

   a. Set the **Test Method** switch to **Auto**.

   b. Turn the **Parameter Estimation** switch to **Start**. The model starts running all tests in the prescribed sequence of `Rs > Ld > Lq > (Ke, J, B)` after which it displays the computed parameter values in the **Estimated Parameters** field.

   c. Turn the **Parameter Estimation** switch to **Stop**.

   Note: In the **Auto** mode, the **Test Select** button selection has no effect.

   During simulation, the target model does not support computation of *J*.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the parameter estimation algorithm on the target hardware to estimate parameters of the connected motor.

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 to run the parameter estimation tests.

Required Hardware

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB command prompt.

- LAUNCHXL-F28379D controller + BOOSTXL-DRV8305 inverter: mcb_pmsm_param_est_f28379d

For connections related to the preceding hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections.

2. 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 4-10.

3. 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 4-79.

Note: Verify the number of slits available in the quadrature encoder sensor attached to your motor. Check and update the variable *pmsm.QEPSlits* available in the model initialization script. This variable corresponds to the **Encoder slits** parameter of the quadrature encoder block. For more details about the **Encoder slits** and **Encoder counts per slit** parameters, see Quadrature Decoder.

4. 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.

5. 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.

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

7. Verify that the variables from the target model are available in the MATLAB base workspace.

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_param_est_host_f28379d.slx');
```

4-321

PMSM Parameter Estimation Host

Copyright 2022-2023 The MathWorks, Inc.

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

9. If required, change the configuration elements for parameter estimation, by updating the mask parameters of the PMSM Parameter Estimation Configurator block available in the host model. For more details, see PMSM Parameter Estimation Configurator.

10. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. This variable updates the **Port** parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

11. In the **User Controls** section, ensure that the **Parameter Estimation** switch is set to **Stop**.

12. Click **Run** on the **Simulation** tab to run the host model. This initiates communication between the host model and target hardware.

13. Use the **Test Method** switch to select either **Manual** or **Auto** mode.

    - **Manual** — This mode enables you to manually execute the tests one-by-one in the prescribed sequence.

    - **Auto** — This mode enables you to automatically run all the tests in the prescribed sequence.

    If you are using **Manual** mode:

        a. Set the **Test Method** switch to **Manual**.

        b. Use the **Test Select** button to select the first test Rs.

        c. Turn the **Parameter Estimation** switch to **Start**. The model runs the *Rs* estimation test after which it displays the computed value in the host model.

        d. Turn the **Parameter Estimation** switch to **Stop**.

        e. Similarly, use the **Test Select** button to select the next test and repeat steps c to e.

    Note: Ensure that you run all tests in the prescribed sequence of Rs > Ld > Lq > (Ke, J, B). Do not skip any test. The host model uses the **Active Test** field to display the test that is currently running.

    If you are using **Auto** mode:

        a. Set the **Test Method** switch to **Auto**.

        b. Turn the **Parameter Estimation** switch to **Start**. The model starts running all tests in the prescribed sequence of Rs > Ld > Lq > (Ke, J, B) after which it displays the computed parameter values in the host model.

        c. Turn the **Parameter Estimation** switch to **Stop**.

    Note: In the **Auto** mode, the **Test Select** button selection has no effect.

    The host model uses the **Active Test** field to display the test that is currently running.

14. Use the **Debug Signal** combo box to select a debug signal. Monitor the selected signal in the scope.

Note: The model can interrupt the currently running test and turn the following LEDs red to protect the hardware from the following faults:

- **Over-current fault** — This fault occurs when the actual current drawn from the power supply is more than the overcurrent limit specified by the variables *motor.OverCurrLim* and *motor.I_rated* in the model initialization script.

- **Under-voltage fault** — This fault occurs when input DC voltage drops below the undervoltage limit specified by the variables *inverter.UnderVoltLim* and *inverter.V_dc* in the model initialization script.

After the parameter estimation tests are complete, the host model saves the computed parameters in the *motorParam* variable inside the base workspace.

Estimate PMSM Parameters Using Parameter Estimation Blocks on Real-Time Systems

This example uses the parameter estimation blocks provided by Motor Control Blockset™ to estimate these parameters of a permanent magnet synchronous motor (PMSM):

- Phase resistance, *Rs* (Ohm)

- D-axis inductance, *Ld* (Henry)

- Q-axis inductance, *Lq* (Henry)

- Back-EMF constant, *Ke* (Vpk_LL/krpm, where Vpk_LL is the peak voltage line-to-line measurement)

- Motor inertia, *J* (Kg.m^2)

- Friction constant, *B* (N.m.s)

Motor parameter estimation is vital for implementing motor control algorithms accurately. Accurate motor parameters enable the algorithm to compute the control parameters with precision. Therefore, an accurate representation of motor parameters results in efficient speed tracking when you run PMSMs using control techniques such as field-oriented control (FOC). Motor parameter estimation also enables you to verify the parameter values provided by the motor datasheet. In addition, it enables you to accurately replicate the plant model in Simulink using which you can simulate real-world scenarios and tests that are difficult to execute using a physical hardware setup.

An accurate simulation helps you reconfigure and run these tests multiple times, which can be difficult when you use physical hardware. Therefore, these exhaustive tests enable you to predict, find, and fix problems including hard-to-reproduce failures with ease.

The example demonstrates usage of the following blocks to design and implement a PMSM parameter estimation algorithm on a Speedgoat® hardware:

- PMSM Parameter Estimation Configurator

- PMSM Rs Estimator

- Ld Estimator

- Lq Estimator

- PMSM Mechanical Parameter Estimation

Alternatively, using this example as a reference, you can utilize these blocks to implement a similar algorithm on a different hardware or configuration.

The target model included in the example uses the preceding parameter estimation blocks inside the subsystem mcb_pmsm_param_est_speedgoat/Parameter Estimation/Parameter Estimation:
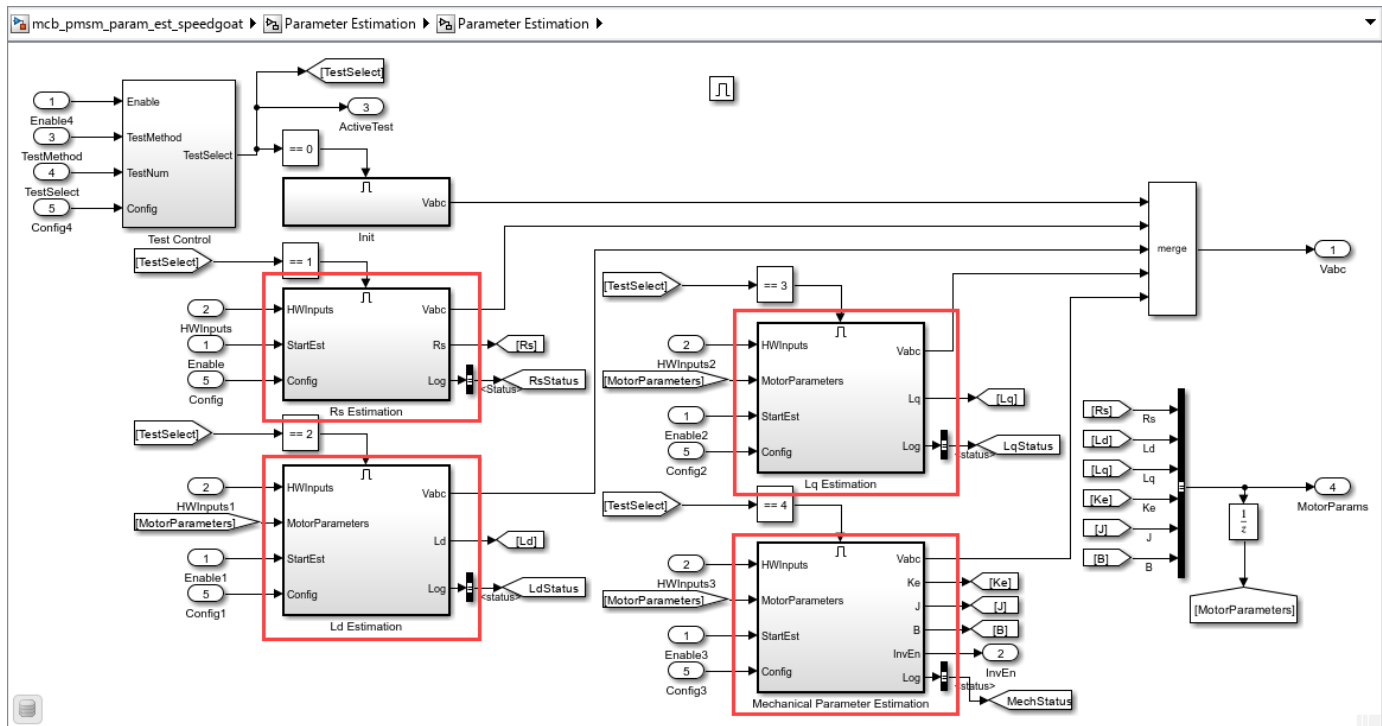
The following subsystems use the parameter estimation blocks to estimate *Rs*, *Ld*, *Lq*, and mechanical parameters:

- `Rs Estimation` — The subsystem uses the Rs Estimator block to estimate the motor resistance.

- `Ld Estimation` — The subsystem uses the Ld Estimator block to estimate *d*-axis inductance.

- `Lq Estimation` — The subsystem uses the Lq Estimator block to estimate *q*-axis inductance.

- `Mechanical Parameter Estimation` — The subsystem uses the PMSM Mechanical Parameter Estimation block to estimate back-emf constant, motor inertia, and friction constant.

The target model included in the example runs on the hardware in external mode. After running the parameter estimation tests, the target model displays the estimated parameters.

Model

The example includes the model mcb_pmsm_param_est_speedgoat.

You can use the model only for both simulation and code generation.

To open the Simulink® model, you can also use the open_system command at the MATLAB® command prompt:

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

PMSM Parameter Estimation on Real-Time Target

Note: This example requires Speedgoat Baseline Real-Time Target machine with IO-397 and Electric motor control kit
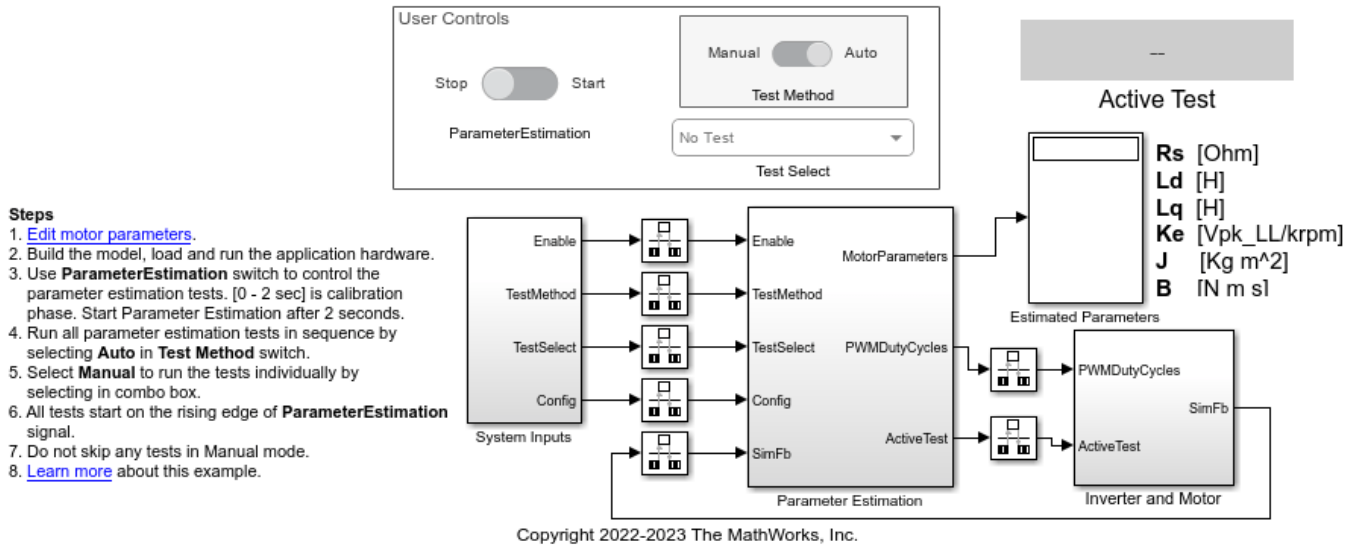
Copyright 2022-2023 The MathWorks, Inc.

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

Required MathWorks® Products

- Motor Control Blockset

- Simulink Control Design™

- Simulink Real-Time™

- Stateflow® (needed only if you modify the example model)

Additional Product

- Speedgoat I/O Blockset (from Speedgoat)

Prerequisite

Check and update the mandatory motor, inverter, and other system parameters listed in the model initialization script `mcb_pmsm_param_est_speedgoat_data.m`.

In addition, you can update the optional motor parameters for simulation.

You can use the inverter and motor hardware datasheet to determine these values.

Simulate Model

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

1. Open the target model included in this example.

2. If required, change the configuration elements for parameter estimation, by updating the mask parameters of the PMSM Parameter Estimation Configurator block available in the

mcb_pmsm_param_est_speedgoat/System Inputs subsystem. For more details, see PMSM Parameter Estimation Configurator.

3. In the **User Controls** section, ensure that the **Parameter Estimation** switch is set to **Stop**.

4. Click **Run** on the **Simulation** tab to simulate the model. The model performs current and position sensor calibration for the first 2 seconds. Proceed to the next steps after 2 seconds.

5. Set the **Test Method** switch to either **Manual** or **Auto** mode.

- **Manual** — This mode enables you to manually execute the tests one-by-one in the prescribed sequence.

- **Auto** — This mode enables you to automatically run all the tests in the prescribed sequence.

If you are using **Manual** mode:

a. Set the **Test Method** switch to **Manual**.

b. Use the **Test Select** button to select the first test Rs.

c. Turn the **Parameter Estimation** switch to **Start**. The model runs the *Rs* estimation test after which it displays the computed value in the **Estimated Parameters** field.

d. Turn the **Parameter Estimation** switch to **Stop**.

e. Similarly, use the **Test Select** button to select the next test and repeat steps c to e.

Note: Ensure that you run all tests in the prescribed sequence of Rs > Ld > Lq > (Ke, J, B). Do not skip any test.

During simulation, the target model does not support computation of *J*.

The target model uses the **Active Test** field to display the test that is currently running.

If you are using **Auto** mode:

a. Set the **Test Method** switch to **Auto**.

b. Turn the **Parameter Estimation** switch to **Start**. The model starts running all tests in the prescribed sequence of Rs > Ld > Lq > (Ke, J, B) after which it displays the computed parameter values in the **Estimated Parameters** field.

c. Turn the **Parameter Estimation** switch to **Stop**.

Note: In the **Auto** mode, the **Test Select** button selection has no effect.

During simulation, the target model does not support computation of *J*.

The target model uses the Active Test field to display the test that is currently running.

Generate Code and Deploy Model to Target Hardware

This section instructs you to generate code and run the parameter estimation algorithm on the target hardware to estimate parameters of the connected motor.

Required Hardware

This example supports `Speedgoat Electric Motor Control Kit` that includes these components:

- Three-phase inverter rated for 48V and 20A from Speedgoat

- 100W three-phase brushless DC motor from Maxon Motor

- Quadrature encoder with 4096 counts per revolution

- 150W 24V DC power supply

Note: Contact Speedgoat for the bit stream file that is valid for your hardware.

For details about Speedgoat hardware setup, see `Speedgoat Software Configuration Guide`.

Generate Code and Run Model on Target Hardware

1. Complete the hardware connections for the Speedgoat Electric Motor Control Kit.

2. Open the target model.

3. If required, change the configuration elements for parameter estimation, by updating the mask parameters of the PMSM Parameter Estimation Configurator block available in the `mcb_pmsm_param_est_speedgoat/System Inputs` subsystem. For more details, see PMSM Parameter Estimation Configurator.

4. Check and update the rotor direction using the **A leads B** parameter of the Condition Encoder block available in the subsystem `mcb_pmsm_param_est_speedgoat/Parameter Estimation/ Input HW Drivers/Codegen`.
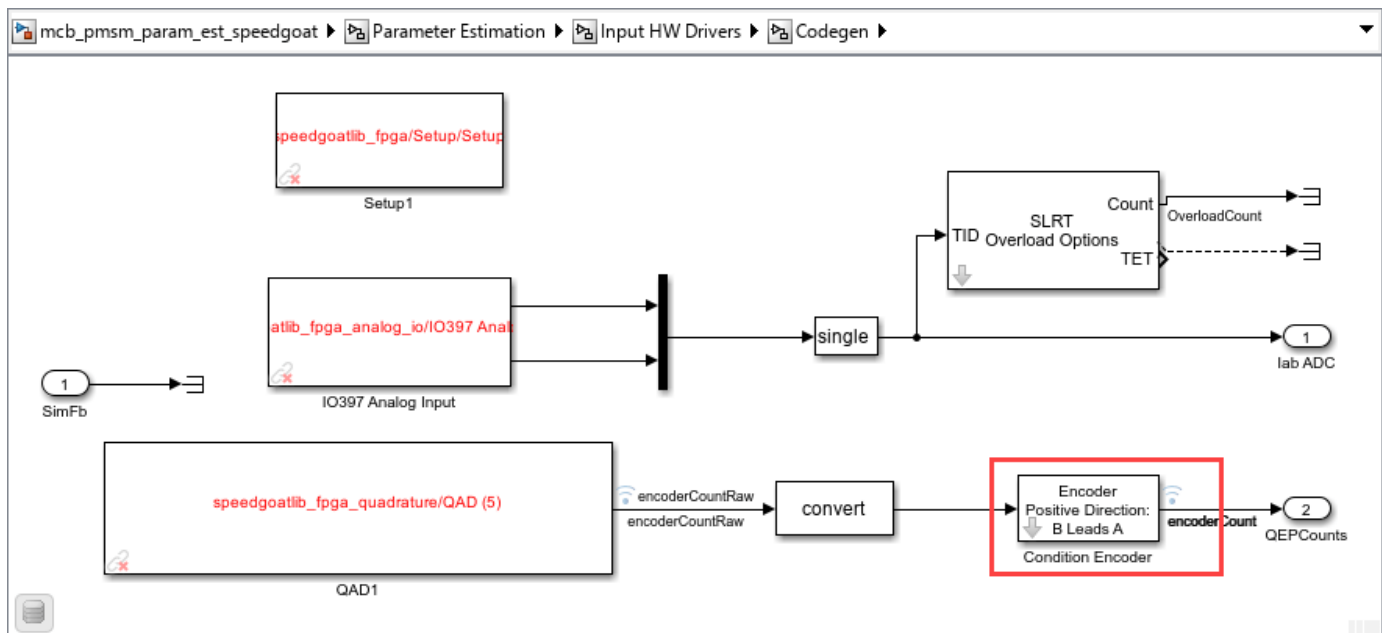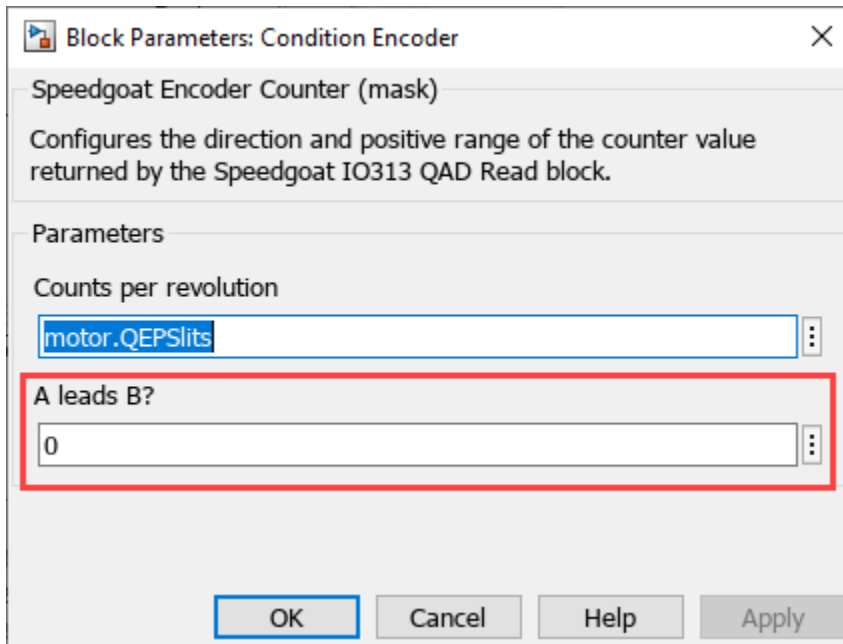
To determine the parameter value, you can either contact Speedgoat or follow the instructions in the Run motor in open-loop control section of the example "Tune PI Controllers Using Field Oriented Control Autotuner Block on Real-Time Systems" on page 4-107.

5. In the **Real-Time** tab on the Simulink toolstrip, click **Run on Target** to build and load the model to the hardware and start running the model in external mode.

The model performs current and position sensor calibration for the first 2 seconds. Proceed to the next steps after 2 seconds.

6. In the **User Controls** section, ensure that the **Test Enable** switch is set to **Off**.

7. Use the **Test Method** switch to select either **Manual** or **Auto** mode.

     - **Manual** — This mode enables you to manually execute the tests one-by-one in the prescribed sequence.

     - **Auto** — This mode enables you to automatically run all the tests in the prescribed sequence.

     If you are using **Manual** mode:

          a. Set the **Test Method** switch to **Manual**.

          b. Use the **Test Select** button to select the first test Rs.

          c. Turn the **Parameter Estimation** switch to **Start**. The model runs the *Rs* estimation test after which it displays the computed value in the **Estimated Parameters** field.

          d. Turn the **Parameter Estimation** switch to **Stop**.

          e. Similarly, use the **Test Select** button to select the next test and repeat steps c to e.

     Note: Ensure that you run all tests in the prescribed sequence of Rs > Ld > Lq > (Ke, J, B). Do not skip any test.

The target model uses the **Active Test** field to display the test that is currently running.

If you are using **Auto** mode:

a. Set the **Test Method** switch to **Auto**.

b. Turn the **Parameter Estimation** switch to **Start**. The model starts running all tests in the prescribed sequence of Rs > Ld > Lq > (Ke, J, B) after which it displays the computed parameter values in the **Estimated Parameters** field.

c. Turn the **Parameter Estimation** switch to **Stop**.

Note: In the **Auto** mode, the **Test Select** button selection has no effect.

The target model uses the **Active Test** field to display the test that is currently running.

Implement PMSM Speed Control Using Active Disturbance Rejection Control

This example shows how to implement active disturbance rejection control (ADRC) of the speed of a permanent magnet synchronous motor (PMSM) modelled in Simulink® using Active Disturbance Rejection Control (Simulink Control Design) block. You can use the example to implement field-oriented control (FOC) using either PI or ADRC-based controller to run the motor in speed control mode. Therefore, you can compare the performance of the PI and ADRC controllers.

ADRC is a model-free control technique that you can use to design and implement controllers for plants that have unknown dynamics as well as internal and external disturbances. The ADRC block uses second-order approximation of the plant dynamics and disturbances modelled as an extended state of the plant and provides robust disturbance rejection without an overshoot.

The following equation defines the second order approximation of the control system that is used by the ADRC block:

$$y(t) = b_0 u(t) + f(t)$$

Where:

- $y(t)$ is the plant output.

- $u(t)$ is the plant input.

- b_0 is the critical gain, which is the estimated gain that describes the plant response to an input $u(t)$.

- $f(t)$ is the total disturbance in motor plant, which includes unknown dynamics and other disturbances.

The block uses an extended state observer to estimate $f(t)$ and implements a disturbance rejection control that reduces the effect of the estimated disturbances on the known system dynamics. The ADRC controllers used in the example run in discrete-time domain.

Model

This example includes the following target Simulink model:

mcb_pmsm_adrc_qep_f28379d

You can use this model only for both simulation and hardware deployment. Click the preceding link or use the following open_system command to open the model:

open_system('mcb_pmsm_adrc_qep_f28379d.slx');
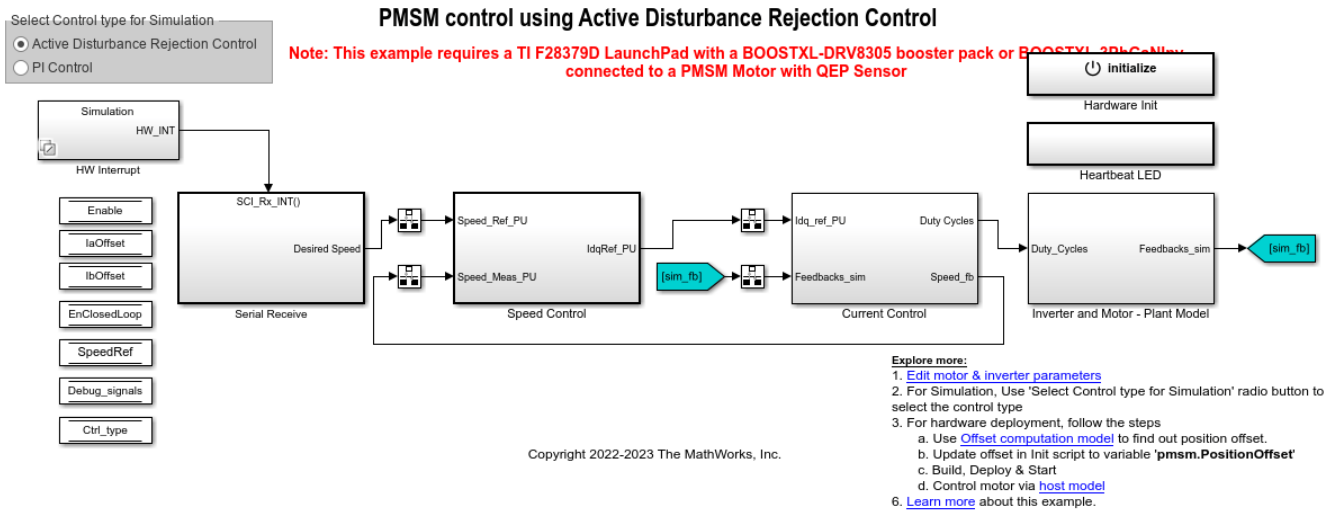
Required MathWorks® Products

To simulate model:

- Motor Control Blockset™

- Control System Toolbox™

- Simulink Control Design™

To generate code and deploy model:

- Motor Control Blockset™

- Control System Toolbox™

- Simulink Control Design™

- Embedded Coder®

- C2000™ Microcontroller Blockset

Prerequisites

1. Obtain the motor parameters. The Simulink model uses default parameters 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 PMSM Parameters Using Recommended Hardware" on page 4-186. The parameter estimation tool updates the `motorParam` variable (in the MATLAB® workspace) with the estimated motor parameters.

2. Update motor parameters. If you obtain the motor parameters from the datasheet or from other sources, update the motor and inverter parameters in the model initialization script associated with the Simulink model. For instructions, see "Estimate Control Gains and Use Utility Functions" 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.

Estimate Critical Gain for ADRC

To implement and use ADRC for your plant, it is essential that you compute the critical gain value b_0 of ADRC. The example includes a script that computes the critical gain value for a given set of controller and plant parameters using a simple experiment. The following steps describe the procedure to estimate the critical gain value using this script.

1. Open the target model mcb_pmsm_adrc_qep_f28379d included in the example.

2. Use the **Select Control type for Simulation** button to select `PI Control`. This enables the PI controllers included in the model.

3. Click **Run** on the **Simulation** tab, to begin simulation and run the PMSM plant model in closed-loop mode (using some PI controller gains).

4. Open the subsystem `mcb_pmsm_adrc_qep_f28379d/Current Control/Control_System/ Closed Loop Control/Current_Controllers/Current_Controllers/`.

5. Measure the q-axis voltage, Vq highlighted below to record the initial plant input, $u(t_1)$.
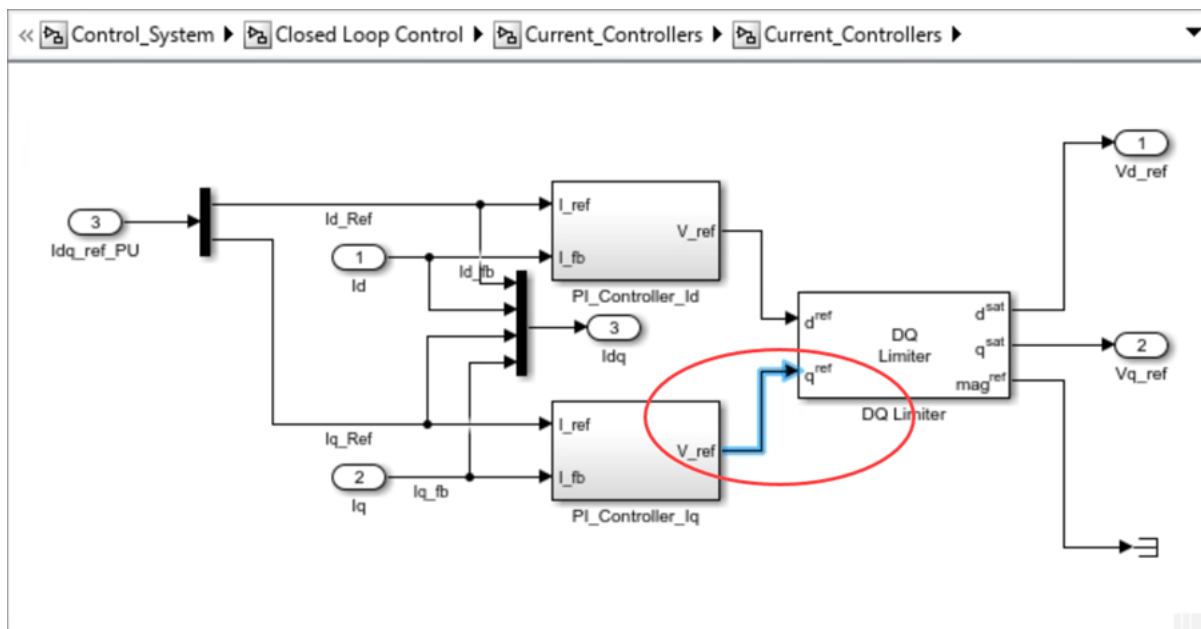
6. Measure the q-axis current, Iq feedback highlighted below to record the initial plant output, $y(t_1)$:
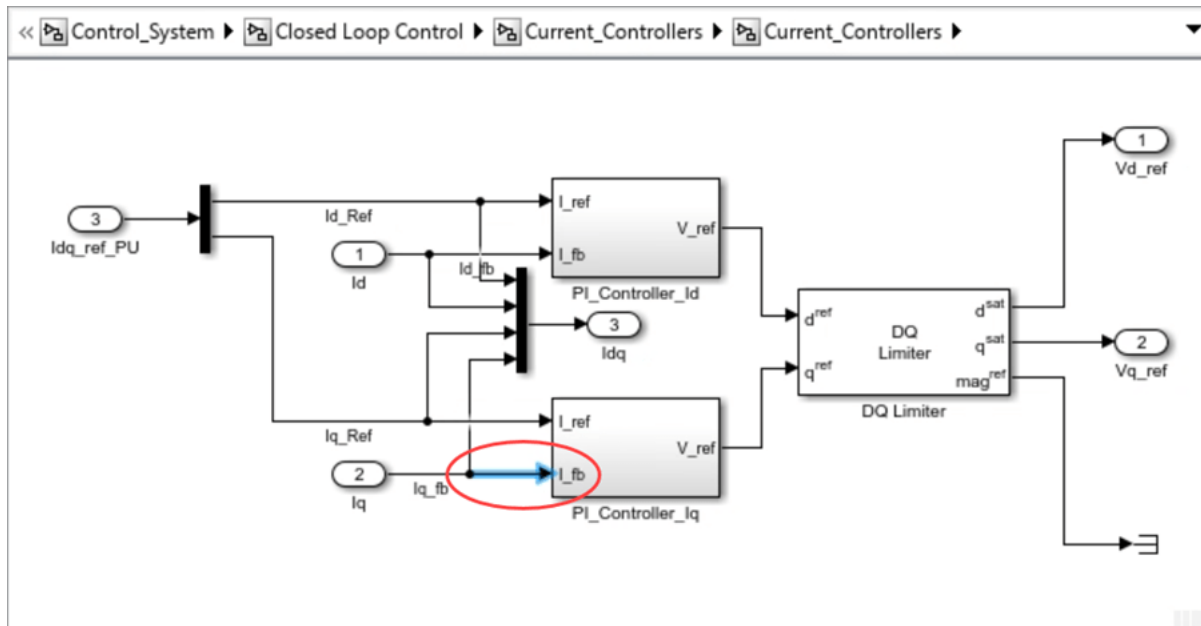
7. Inject a step signal into the initial plant input signal (described in step 5) so that the plant input, rises from $u(t_1)$ to $u(t_2)$.
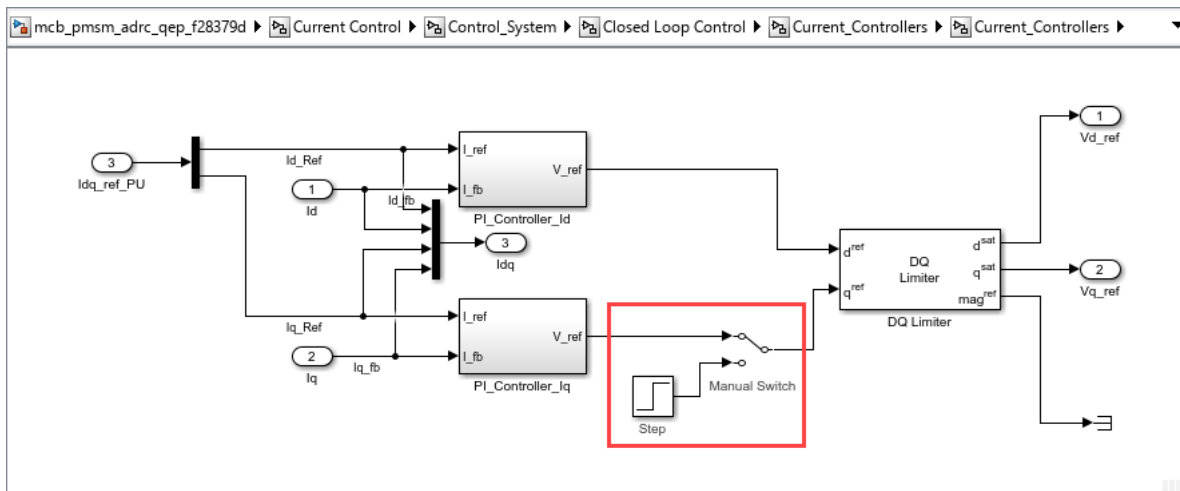
Note: The preceding screenshot shows an experimental modification that is not part of the actual model.

8. Measure the q-axis current, Iq feedback signal (described in step 6) and record the updated plant output, $y(t_2)$. At the same time, measure the time taken (t) by the plant output to reach $y(t_2)$ from $y(t_1)$.

9. Open the model initialization script `mcb_pmsm_adrc_qep_f28379d_data.m` and update the following variables:

- t — Update this variable with the time taken by the plant output to reach $y(t_2)$ from $y(t_1)$.

- u — Update this variable with $u\left(t_2\right) - u\left(t_1\right)$.

- y — Update this variable with $y\left(t_2\right) - y\left(t_1\right)$.

The example script uses these variables to automatically compute and update the critical gain value of the Iq and Id ADRC controller, *ct_b0*:

```
%% Current controller - ADRC
t=300e-6;          % rise time
u=0.190-0.169;     % Vq (plant input)
y=0.061-0.015;     % Iq ref (plant output)

a=2*y/t^2;
ct_b0=(a/u)*50;
ct_ctrl_bw=2000*pi;   % critical gain
ct_obs_bw=2000*pi*8;
```

The ADRC block (in the *Iq* and *Id* current control loops) extracts the critical gain (*ct_b0*) as well as current controller bandwidth (*ct_ctrl_bw*) and observer bandwidth (*ct_obs_bw*) from the model initialization script.
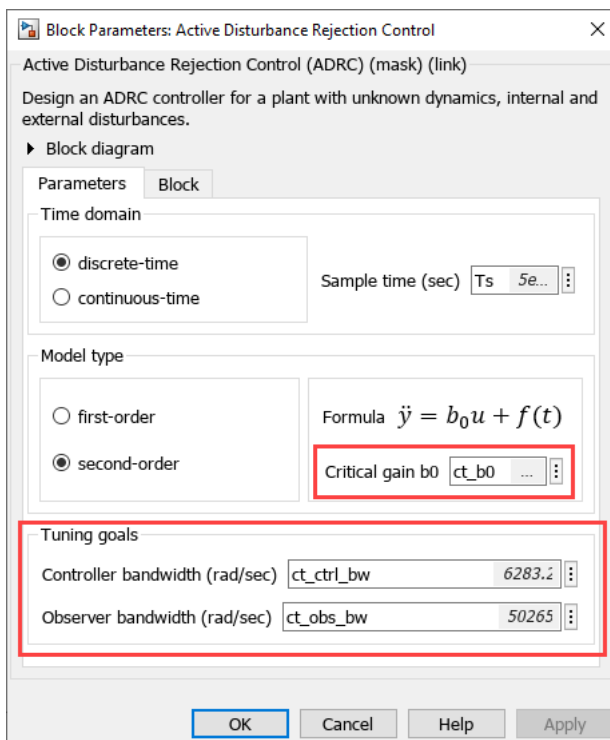
10. Use similar steps to compute the critical gain, *spd_b0* of the speed ADRC controller.

Note: Because multiple control loops execute simultaneously, this process can be challenging and may require considerable time and expertise.

Simulate Model

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

1. Open the target model included with this example.

2. Use the **Select Control type for Simulation** button to select the type of controller that you want to use:

       ◦ `Active Disturbance Rejection Control` — Select this option to enable ADRC controllers for speed and current control loops. To use this option, ensure that you follow the instructions in the Estimate Critical Gain for ADRC section to determine the critical gain value of ADRC.

       ◦ `PI Control` — Select this option to enable PI controllers for speed and current control loops.

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

4. Click **Data Inspector** in the **Review Results** section to view and analyze the simulation results.

Generate Code and Deploy Model to Target Hardware

This section shows 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. Before you can run the host model on the host computer, 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

The example supports the following hardware configuration. You can also use the target model name to open the model from the MATLAB command prompt.

LAUNCHXL-F28379D controller + (BOOSTXL-DRV8305) inverter: mcb_pmsm_adrc_qep_f28379d

Note: When using the BOOSTXL-3PHGANINV inverter, ensure that you have proper insulation between the bottom layer of BOOSTXL-3PHGANINV and the LAUNCHXL board.

For connections related to this hardware configuration, see "LAUNCHXL-F28069M and LAUNCHXL-F28379D Configurations" on page 7-6.

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 4-10.

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 4-79.

Note: Verify the number of slits available in the quadrature encoder sensor attached to your motor. Check and update the variable *pmsm.QEPSlits* available in the model initialization script. This variable corresponds to the **Encoder slits** parameter of the Quadrature Encoder block. For more details about the **Encoder slits** and **Encoder counts per slit** parameters, see Quadrature Decoder.

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. Use the **Select Control type for Simulation** button to select the type of controller that you want to run on the hardware:

    ◦ `Active Disturbance Rejection Control` — Select this option to enable ADRC controllers for speed and current control loops. To use this option, ensure that you follow the instructions in the Estimate Critical Gain for ADRC section to determine the critical gain value of ADRC.

    ◦ `PI Control` — Select this option to enable PI controllers for speed and current control loops.

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. Verify that the MATLAB base workspace reflects the variables from the target model.

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

```
open_system('mcb_pmsm_adrc_host_model_f28379d.slx');
```

PMSM Control Host
Active Disturbance Rejection Control (ADRC)

Prerequisites:
1. Deploy the target model to the hardware
   mcb_pmsm_adrc_qep_f28379d

2. The variables from the target model are
   loaded to the base workspace.

Note:
1. Set the serial port to **target.comport**
   in the init script of the target model.
2. Use **Motor Control Type** switch to
   control motor.
3. Input speed request using **Reference
   Speed** block.
4. Observe the debug signals in scope.

Caution:
When the motor is running, do not change
the **Motor Control Type** switch position
directly from PI to ADRC or ADRC to PI.
Instead, use the switch to stop the motor
first before making a transition to a new
controller type.

Ctrl_type:Value

Off

ADRC ● ● PI

Reference Speed (RPM)

Motor Control Type

Debug signals
- ◉ Speed_ref & Speed_feedback
- ○ Id_ref & Id_feedback
- ○ Iq_ref & Iq_feedback
- ○ Ia & Ib
- ○ Ia & Position

target.comport
Baud rate : 12e6

Host Serial Setup

Scope (Per-Unit)
Debug1 (SI units)
Debug2 (SI units)

Serial Communication

SelectedSignals

Copyright 2022-2023 The MathWorks, Inc.

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

10. In the model initialization script associated with the target model, specify the communication port using the variable *target.comport*. The example uses this variable to update the **Port** parameter of the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks available in the host model.

11. Based on the **Select Control type for Simulation** button setting on the target model, set the **Motor Control Type** switch to either **PI** (to enable the PI Controller) or to **ADRC** (to enable the ADRC controller).

12. Update the reference speed value in the **Reference Speed (RPM)** field in the host model. We recommend that you enter lower speeds initially during motor start up and gradually increase the speed later.

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

14. Use the **Debug signals** section to select the debug signals that you want to monitor. Observe the debug signals from the **Serial Communication** subsystem, in the **SelectedSignals** scope of the host model.

Caution: When the motor is running, do not use the **Motor Control Type** to switch directly from **PI** to **ADRC** or **ADRC** to **PI** controller. Instead, use the switch to stop the motor first before making a transition to a new controller type.

PMSM Constraint Curves and Their Application

This example explains the fundamentals of constraint curves, untilization of these curves to determine operating currents, and usage of the grid of these currents in simulation or deployment environments.

The first section Fundamentals of Constraint Curves of PMSM explains about the constraint curves of a Permanent Magnet Synchronous Machine (PMSM). The five curves (current limit circle, constant torque curve, MTPA curve, voltage limit curve, and MTPV curve) represent the constraints and characteristics of the motor system.

The second section Using Constraint Curves explores the applications of the constraint curves in identifying the optimal operating points to find reference currents in d- and q-axes for a given operating speed and reference torque. The section shows how to create Look-up tables (LUT) of reference currents. You will come across a group of five operating points across different regions, which the example elaborates.

The third section Identifying Possible (MTPA-Field Weakening-MTPV) Control Currents shows how to use the LUT based PMSM Control Reference block to find the operating points in a simulation environment.

Fundamentals of Constraint Curves of PMSM

Constraint curves represent the operational constraints of the PMSM control. Each curve corresponds to either a constraint or a characteristic behavior.

The following code can be used to re-create figure 1.

```
%{

pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
mcbPMSMCharacteristics(pmsm,inverter,speed=5500,torque=0.1394);

%}
```
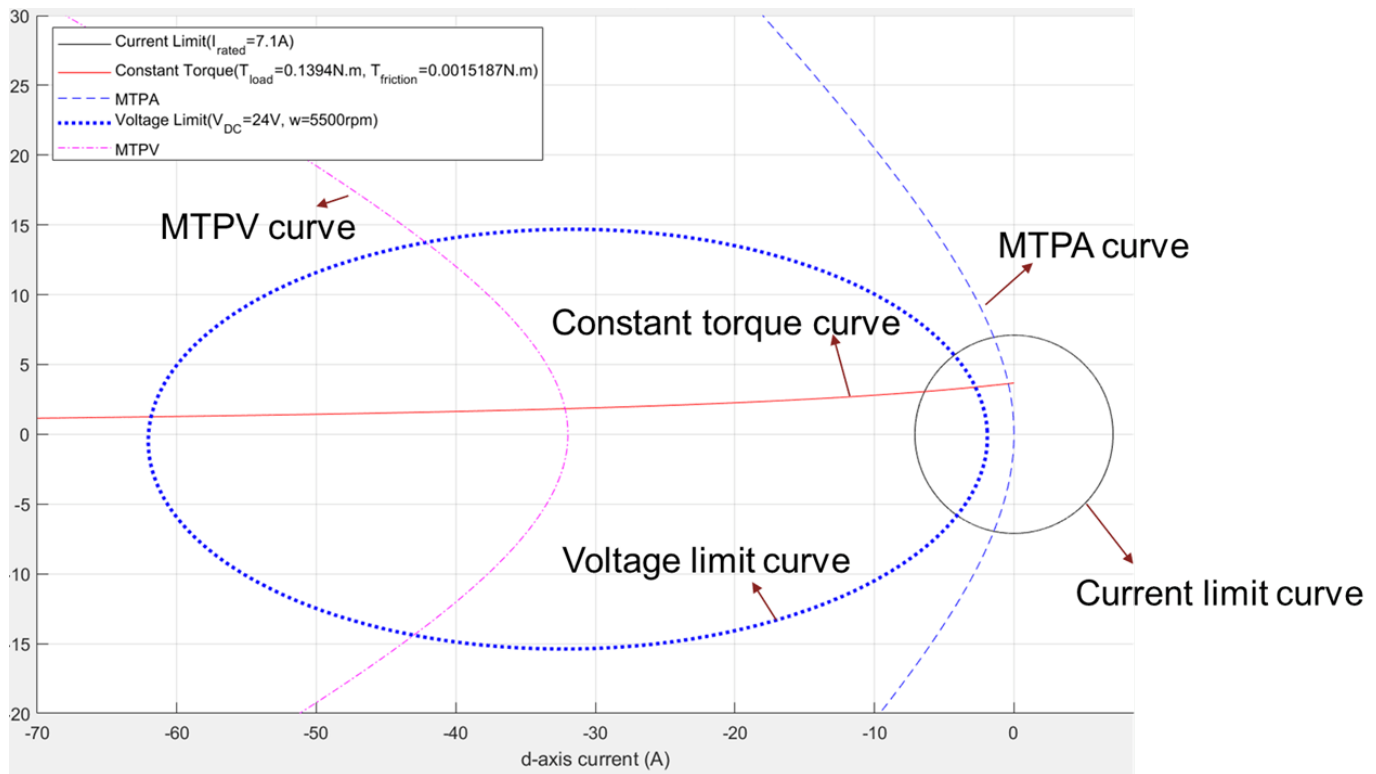
Figure 1: Representation of constraint curves of a PMSM

The d-q Frame of Reference

The d-q frame of reference is a rotating frame of reference, that moves along with the rotor. This frame of reference, as opposed to the stationary frame of reference, simplifies the representation of motor's electrical quantities, such as voltages and currents.

You can measure the motor phase currents *ia*, *ib* and *ic* and transform them into d-q reference frame currents (*id*, *iq*) using the knowledge of rotor position. For a balanced three phase system, you can assume the zero-sequence current to be zero. For more details about the Clarke and Park transformations, see Clarke and Park Transforms - MATLAB & Simulink (mathworks.com).

The d-q Current Space

This is the imaginary frame of reference with the *d*-axis current represented in the *x*-axis, and the *q*-axis current represented in the *y*-axis.

You can plot a given PMSM's limits and characteristics (when represented in terms of *id* and *iq*) in the *d-q* current space.

Current Limit Circle

This is the operating limit of the system on the current. Because the net current of the system is limited as well as the *id* and *iq* components are orthogonal, these currents are related in the following manner:

$$i_d^2 + i_q^2 = i_s^2 \leq i_{s\_\max}^2$$

where,

- i_d is the *d*-axis current (in Amperes).

- i_q is the *q*-axis current (in Amperes).

- i_s is the source current (in Amperes).

- $i_{s\_\max}$ is the maximum source current (in Amperes).

Because *is_max* is constant for a given constraint, locus of all the points that satisfy this relation represents a circle.

Any point inside this circle (*id, iq*) has a magnitude that is less than the current limit *is_max*. Therefore, all the points inside the circle are valid points when considering the current constraint.

Irrespective of the motor topology (SPMSM or IPMSM), the current limit circle remains a circle.

The following figure shows the plot of a current limit circle in black color with an *is_max* of 7.1A.

FW Control method: VCLMT, VoltageEquation: actual

Current Limit(I_{rated}=7.1A)

Current limit curve

q-axis current (A)

d-axis current (A)
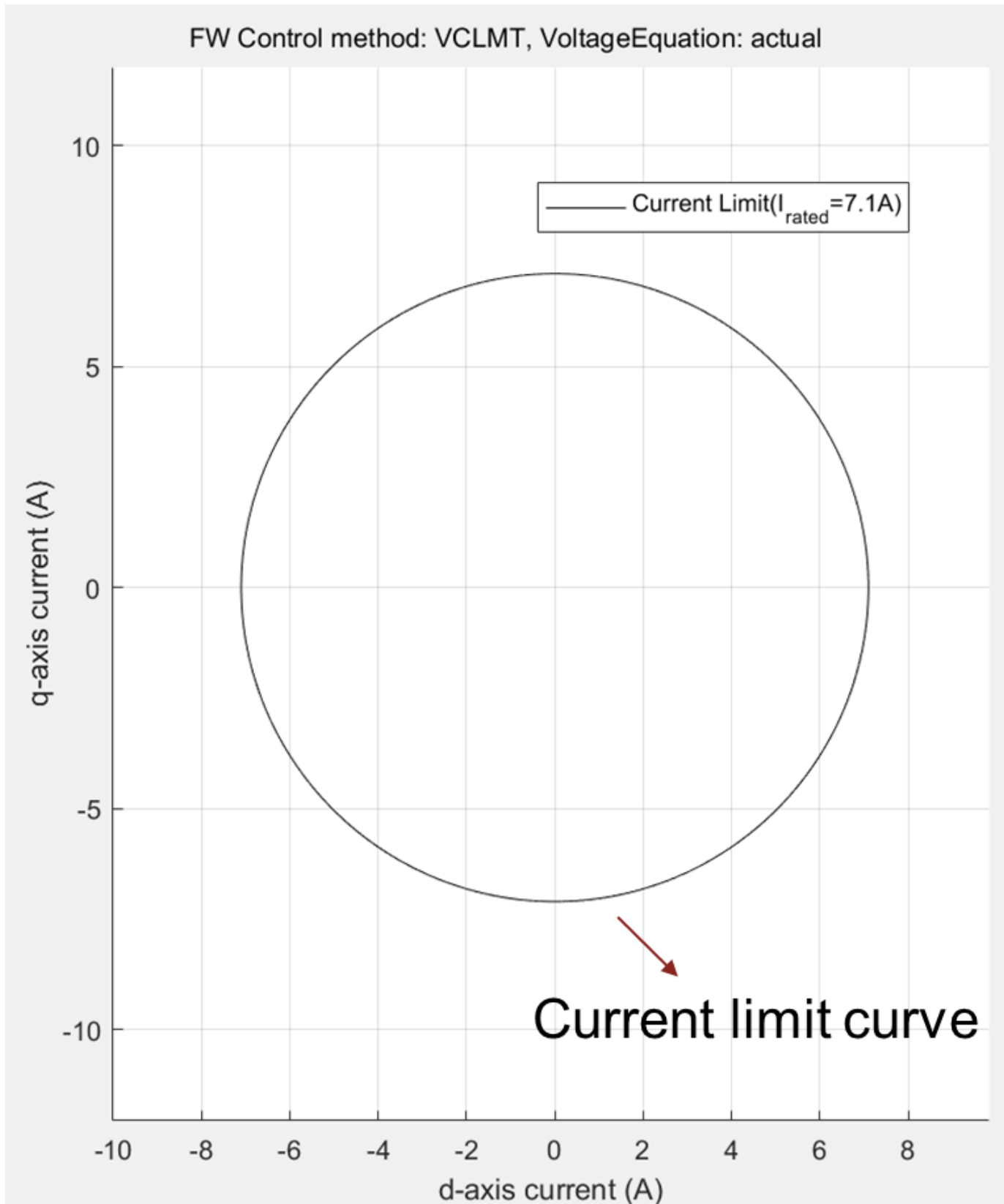
Figure 2: A current limit circle or current limit curve of a PMSM

Constant Torque Curve

Constant torque curve is a locus of all (*id,iq*) points that deliver a same torque.

The torque of a PMSM includes reluctance torque and PM torque. The following equation represents the generated torque:

$$T = 1.5\ polePairs \left(i_q\ FluxPM + \left(L_d - L_q \right) i_d\ i_q \right)$$

where,

- *polePairs* is the number of pole pairs in the PMSM.
- *FluxPM* is the permanent magnet flux linkage (Weber).
- i_d and i_q are the *d*-axis and *q*-axis currents respectively (in Amperes).
- L_d and L_q are the *d*-axis and *q*-axis inductances respectively (in Henry).

For a given fixed torque, the preceding equation transforms to $i_q = \dfrac{K_1}{K_2 + i_d}$, which forms a hyperbola representation.

Any (*id, iq*) value pair on the constant torque curve represents currents that satisfy the load torque. To ensure that (*id,iq*) also satisfy the constraint, you can check if this point satisfies the current constraint (inside the closed circle).
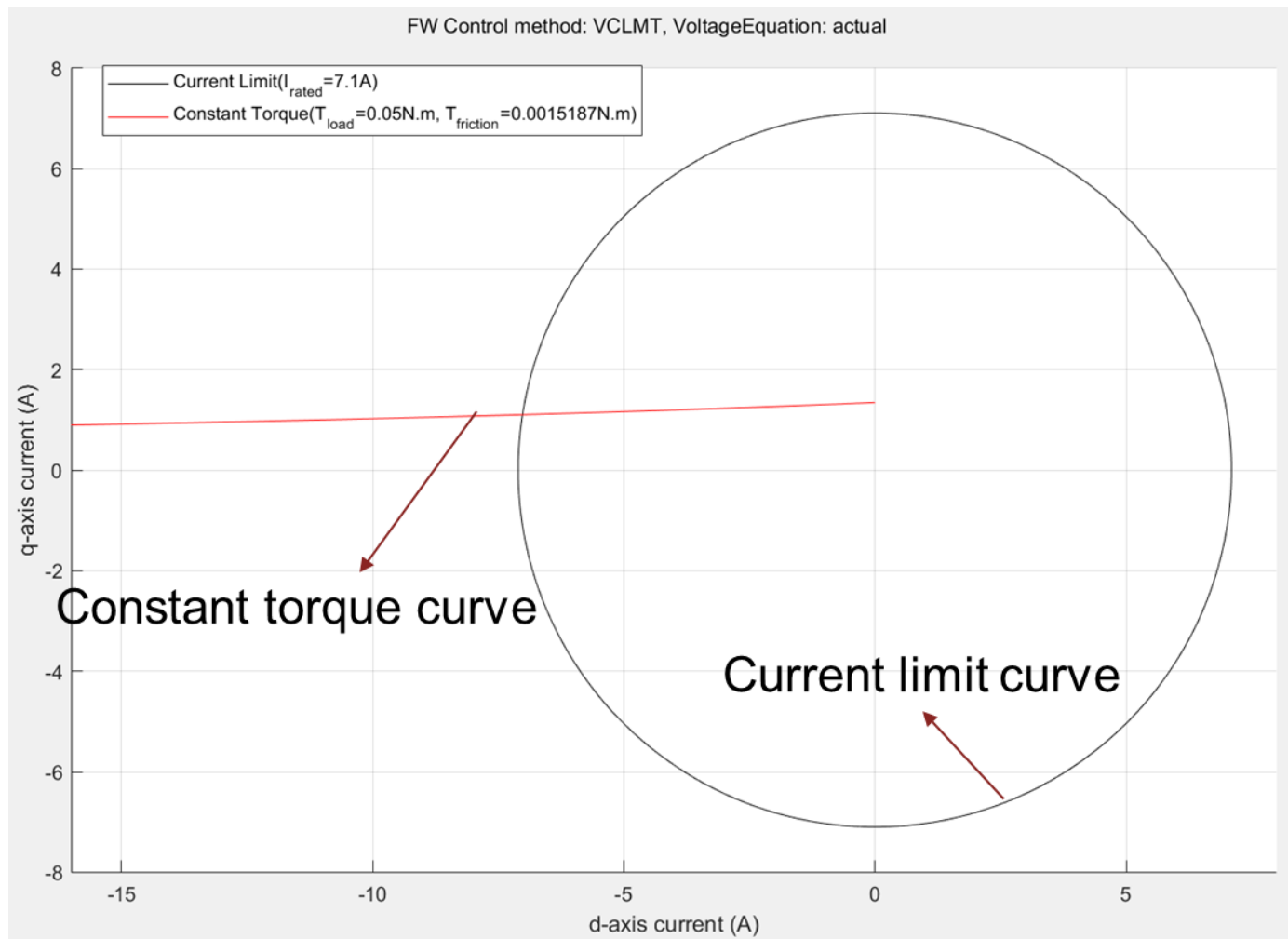
Figure 3: Current limit circle and constant torque curve of PMSM.
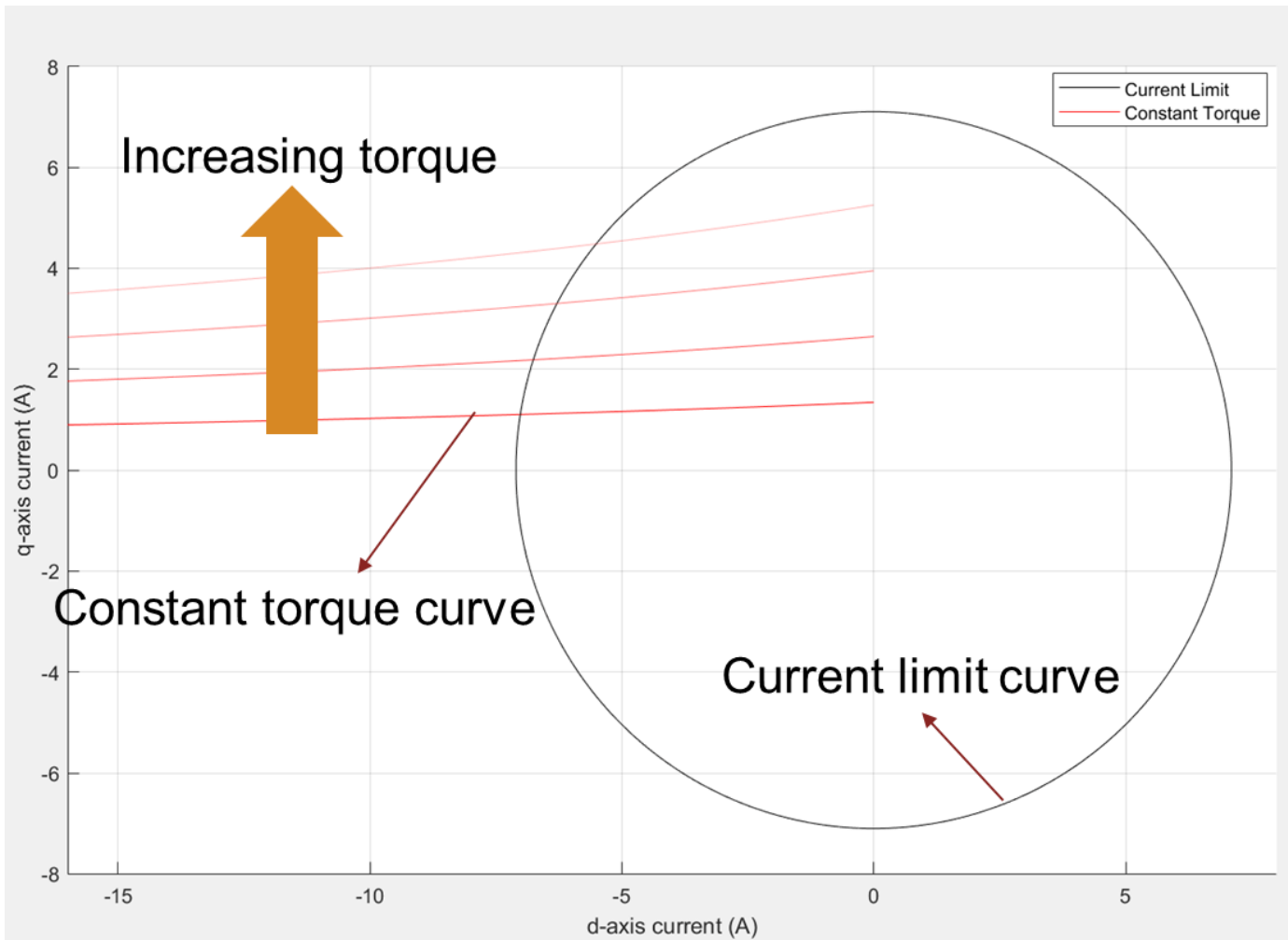
Figure 4: Current limit circle and a group of constant torque curves drawn at different torque values in an increasing order. The lighter shade of the constant torque curve represents an increasing torque.

MTPA Curve

MTPA stands for Maximum Torque Per Ampere. An MTPA curve is a locus of all points that satisfy the optimization condition providing a maximum torque for a given current.

You can obtain the expression for MTPA curve (using linear motor parameters) with an assumption that the partial derivative of the torque with respect to current (either *id* or *iq*) is equal to zero.

For an IPMSM, the following expression represents an MTPA trajectory:

$$i_d^2 + \frac{i_d\, FluxPM}{L_d - L_q} = i_q^2$$

where,

- *FluxPM* is the permanent magnet flux linkage (Weber).

4-345

- i_d and i_q are the d-axis and q-axis currents respectively (in Amperes).
- L_d and L_q are the d-axis and q-axis inductances respectively (in Henry).

The preceding equation represents a parabola. All points on the MTPA curve satisfy the condition that they provide maximum torque per current.
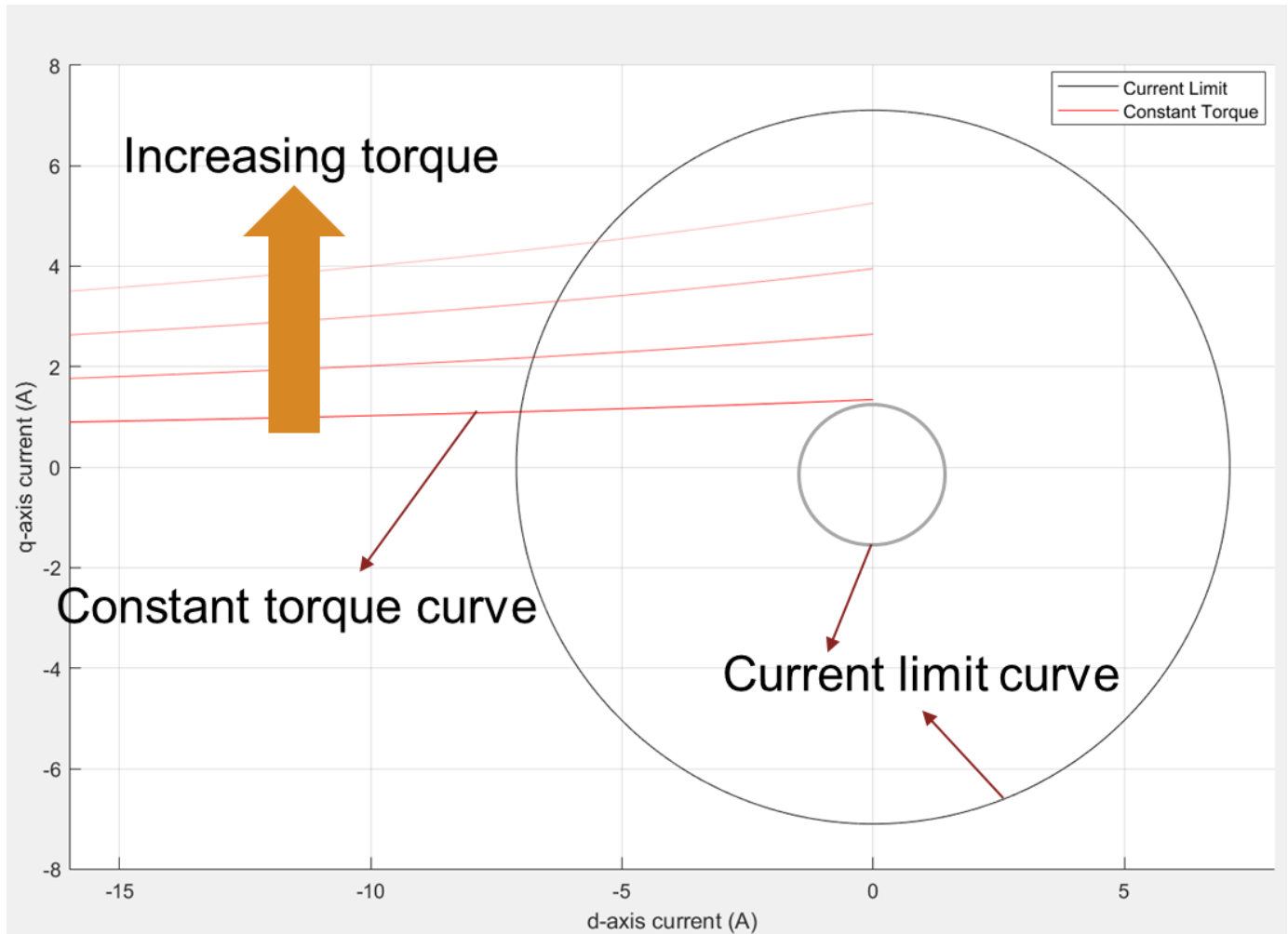
Figure 5: Current limit circle, a group of constant torque curves, and a smaller current limit circle representing the current limit for one of the constant torque curves.

When we consider the operating current as a virtual current limit for a hypothetical system and draw the constraint curves including a constant torque curve and a current limit circle, they both intersect tangentially at a single point. This point represents the maximum possible torque that you can achieve using the virtual current limit for the hypothetical system.
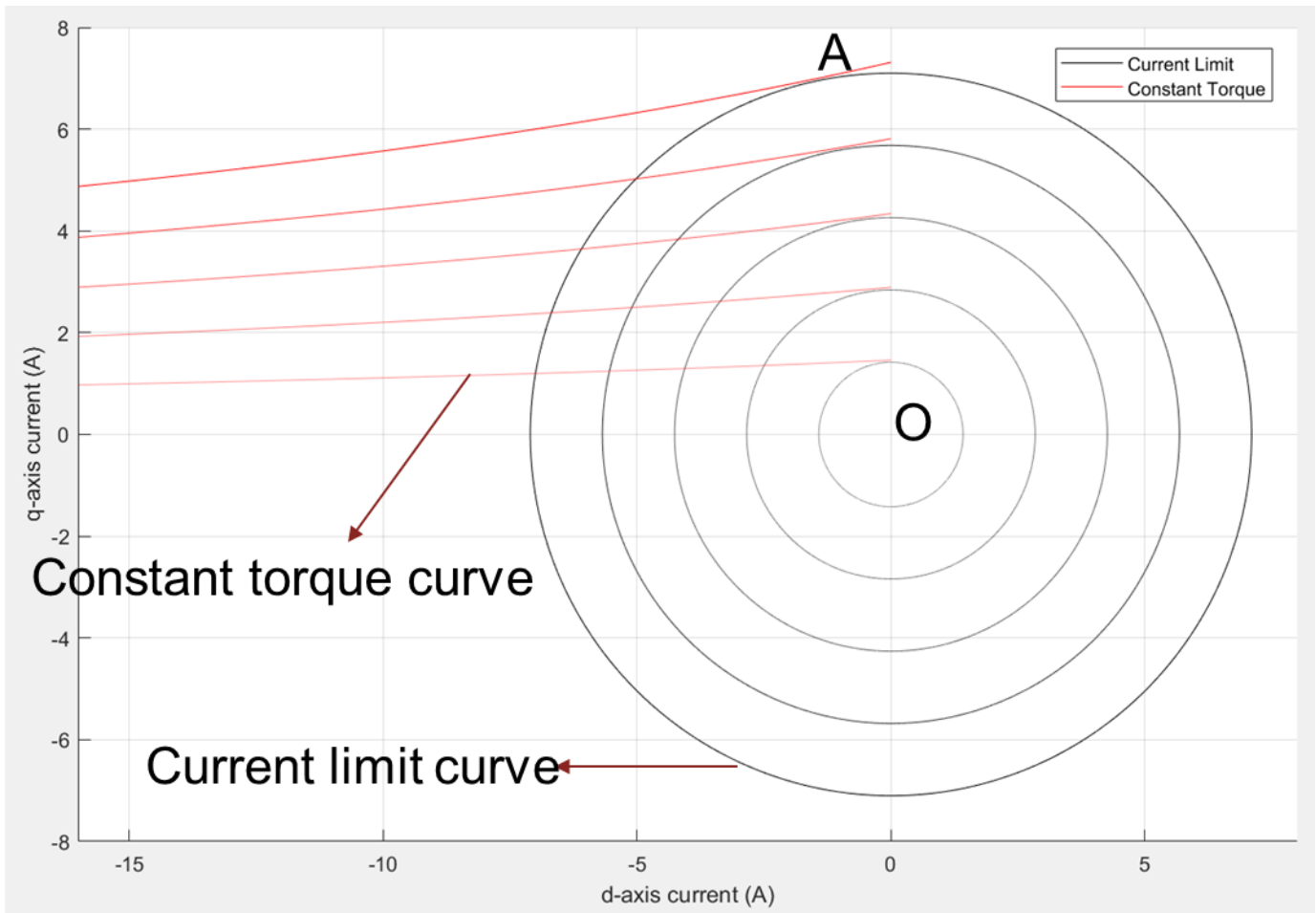
Figure 6: A group of pairs of current limit circles with different limit values and the corresponding maximum torques.
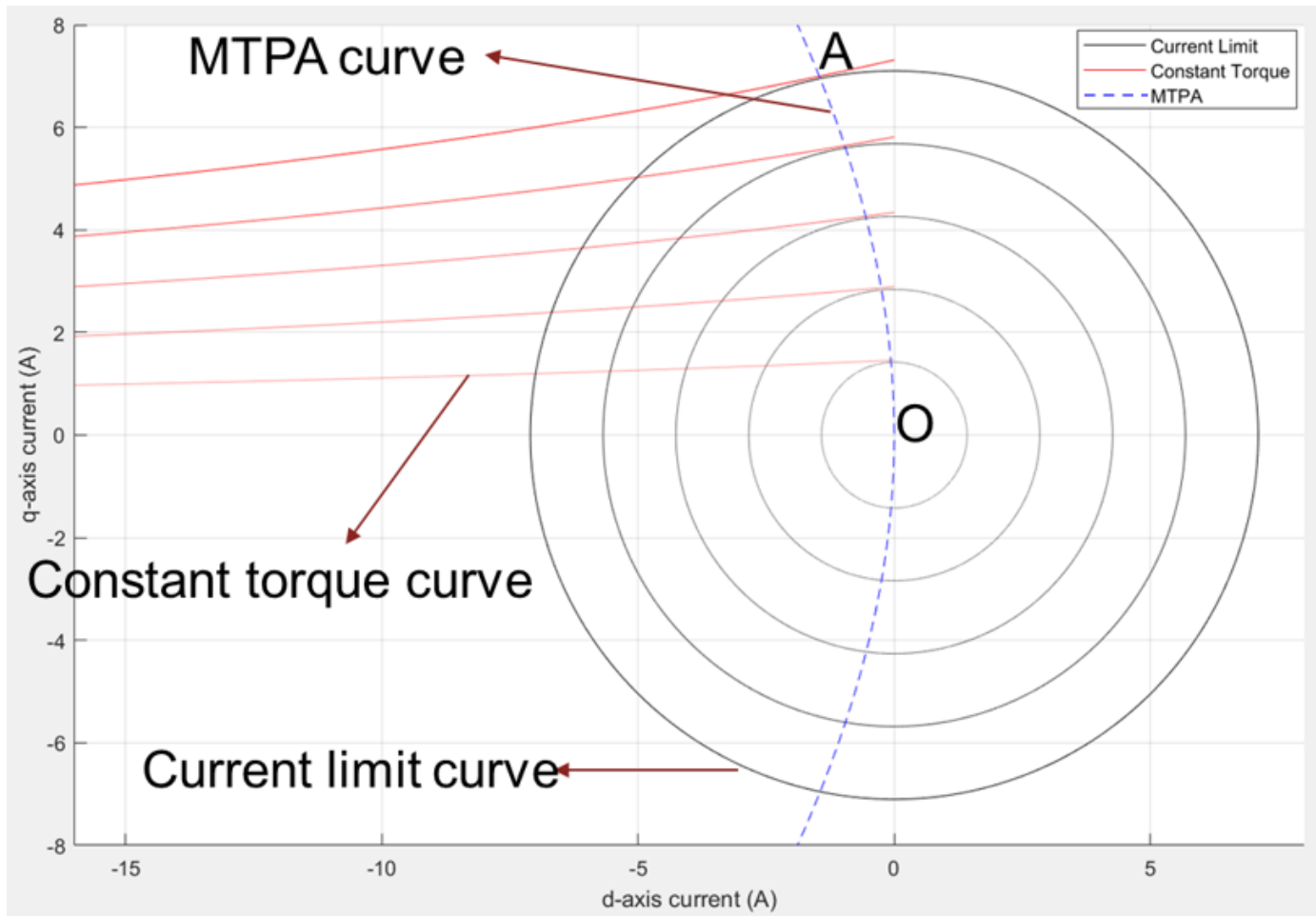
Figure 7: A group of pairs of current limit circles with different limit values with the corresponding maximum torques and the MTPA curve.

As shown in figure 7, the MTPA curve is a collection of all points which are intersecting points of current limit circles and the corresponding constant torque curves (drawn at maximum possible torque for the given current). This curve represents visually how MTPA can be understood in the form of points representing the maximum torque for a given current.

Voltage Limit Curve

Voltage limit curve is the locus of all (id,iq) points for a specified speed and voltage, which result in the system voltage, $\sqrt{v_d^2 + v_q^2}$, reaching the voltage limit, *vmax*.

The following equations represent the phase voltages in the *d-q* reference frame:

$v_d = i_d\,R_s - w\,FluxQ$, where $FluxQ = L_q\,i_q$

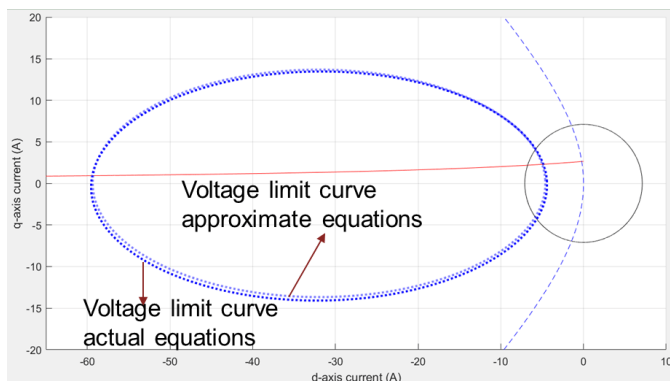$v_q = i_q\,R_s - w\,FluxD$, where $FluxD = L_d\,i_d + FluxPM$

where,

- *FluxD* and *FluxQ* are the *d*-axis and *q*-axis flux linkages respectively (in Weber).
- *FluxPM* is the permanent magnet flux linkage (Weber).
- i_d and i_q are the *d*-axis and *q*-axis currents respectively (in Amperes).
- v_d and v_q are the *d*-axis and *q*-axis voltages respectively (in Volts).
- L_d and L_q are the *d*-axis and *q*-axis inductances respectively (in Henry).
- w is the rotor speed (in Radians/ sec).
- R_s is the stator phase winding resistance (in Ohms).

The system voltage is constrained by the operating power supply voltage and the PWM scheme. For SVPWM, the available voltage is $v_{\max} = \dfrac{V_{dc}}{\sqrt{3}}$, while for Sine PWM, the available voltage is $v_{\max} = \dfrac{V_{dc}}{2}$.

$$v_d^2 + v_q^2 = v_s^2 \le v_{\max}^2$$

where,

- v_d and v_q are the *d*-axis and *q*-axis voltages respectively (in Volts).
- v_s is the source voltage (in Volts).
- $v_{\max}$ is the maximum fundamental line to neutral voltage (peak) supplied to the motor (in Volts).

The system voltage *vs* is equal to or less than *vmax* to satisfy the voltage constraint.

After inserting *vd* and *vq* representations in the preceding equation, you get:

$$\left(i_d\, R_s - w\, L_q\, i_q\right)^2 + \left(i_q\, R_s + w\, L_d\, i_d + w\, FluxPM\right)^2 = v_s^2 \le v_{\max}^2$$

where,

- *FluxPM* is the permanent magnet flux linkage (Weber).
- i_d and i_q are the *d*-axis and *q*-axis currents respectively (in Amperes).
- v_d and v_q are the *d*-axis and *q*-axis voltages respectively (in Volts).
- v_s is the source voltage (in Volts).
- $v_{\max}$ is the maximum fundamental line to neutral voltage (peak) supplied to the motor (in Volts).
- L_d and L_q are the *d*-axis and *q*-axis inductances respectively (in Henry).
- w is the rotor speed (in Radians/ sec).
- R_s is the stator phase winding resistance (in Ohms).

This equation represents a generalized ellipse.

SPMSM has *Ld=Lq*, while for IPMSM, *Lq>Ld*, where *Ld* and *Lq* are inductances along the *d*- and *q*-axes respectively.

Due to the above parameter variation, the voltage limit curve of the SPMSM is a circle, with a center away from the origin. The shift from the origin (*id=0*, *iq=0*) depends on the motor system parameters *FluxPM*, speed, *Rs*, *Ld*, and *Lq*.

Because for an IPMSM, *Ld* and *Lq* are not identical, the generated voltage limit curve is an ellipse.
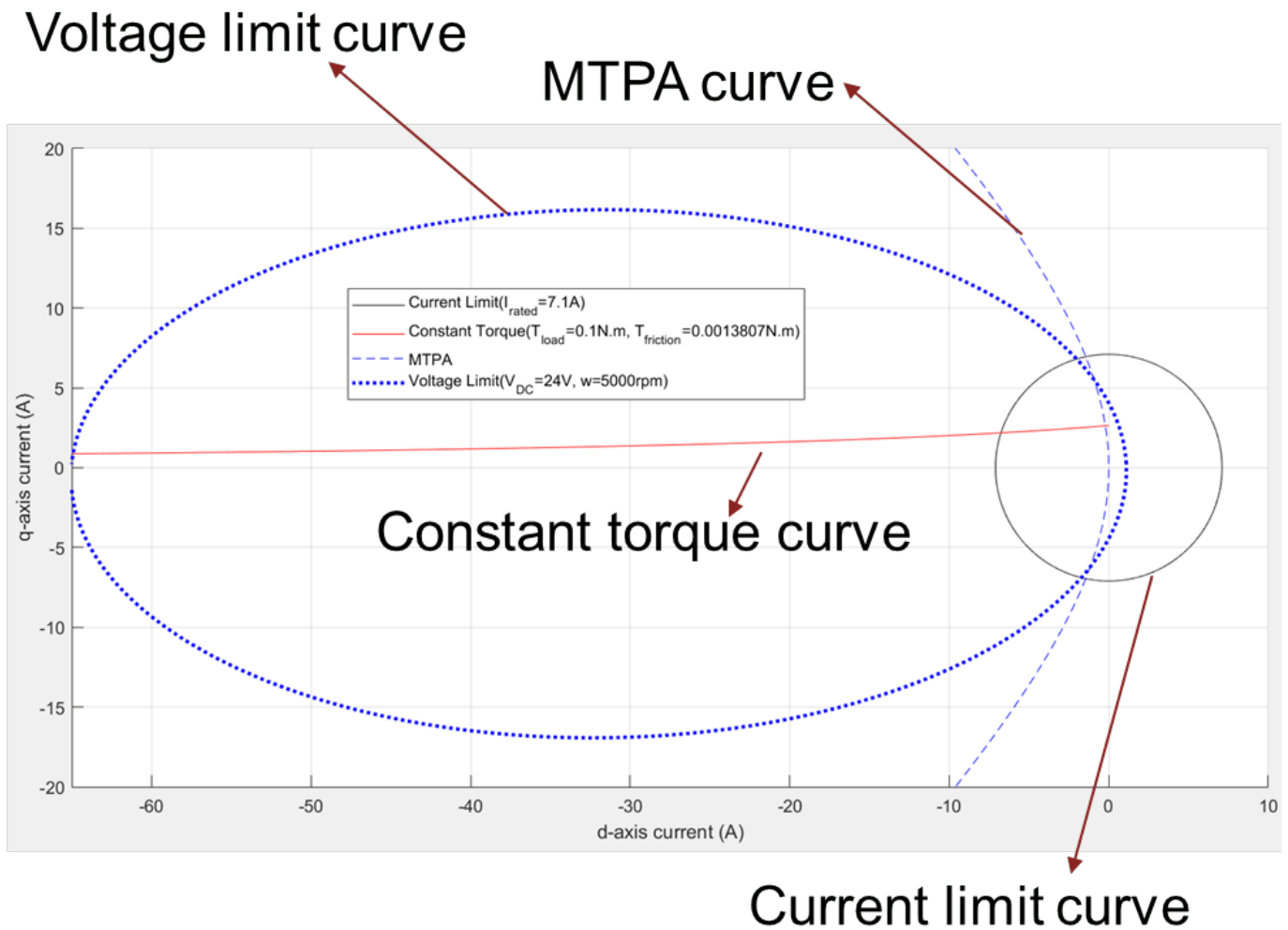
Figure 8: Current limit circle, voltage limit curve, and MTPA curve of a PMSM.

Figure 8 includes the voltage limit ellipse for a given motor. You can see that the current limit circle is centered around the origin and the voltage limit ellipse is not centered around the origin. The region of overlap between the voltage limit ellipse and the current limit circle represent the possible (*id, iq*) current values that satisfy both the voltage and current constraints.
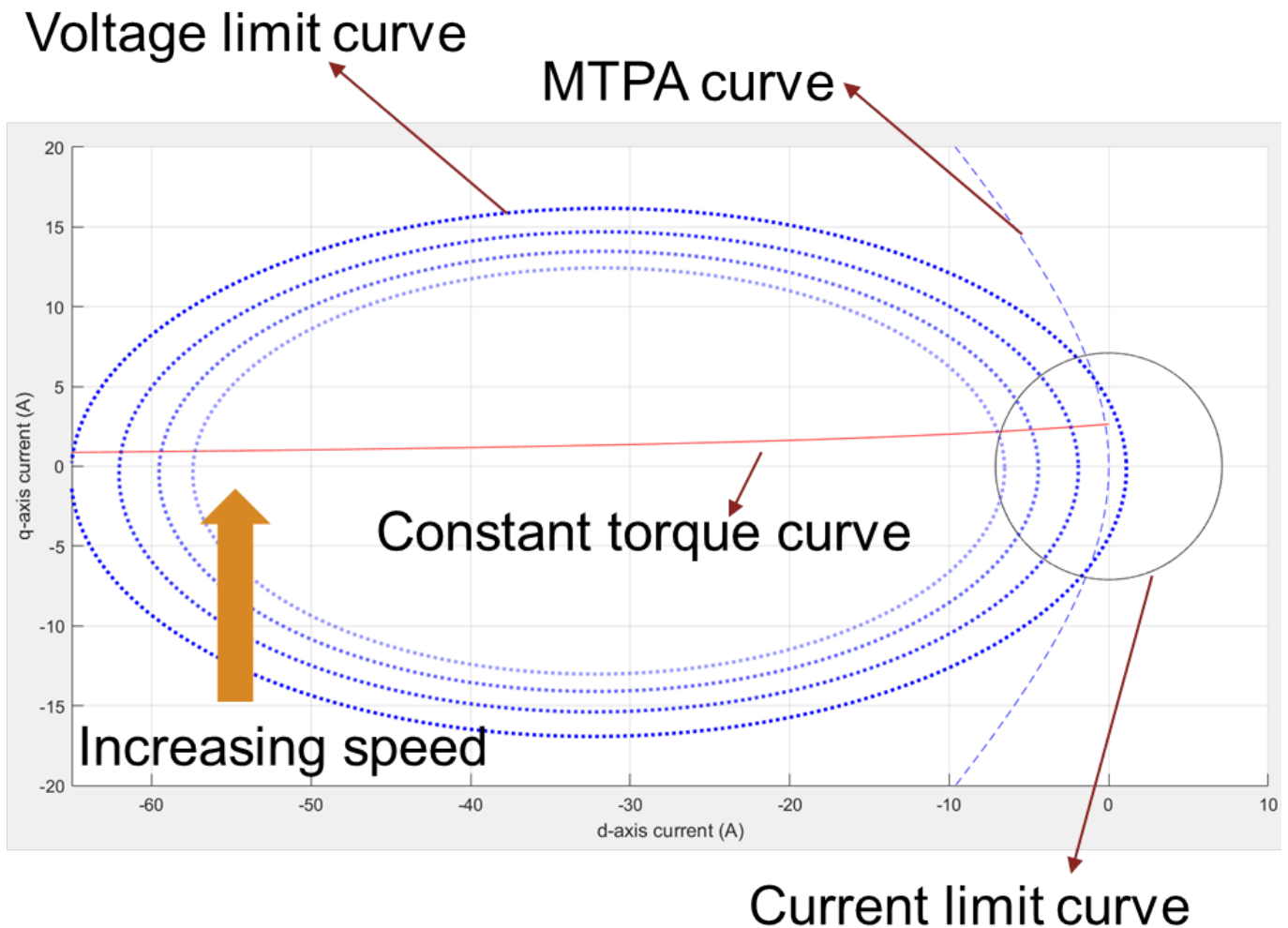
Figure 9: Current limit circle, MTPA curve, and a group of voltage limit curves drawn at different speeds. The lighter dotted blue lines represent the increasing direction of speed.

Voltage Limit Curve - Approximate Equations vs Actual Equations

A simpler form of the voltage limit curve is called the approximate representation. In this representation, you can ignore the $i\,r$ component in the vd and vq equations and compensate them in the $vmax$ term.

In the approximate representation, the following equations represent the phase voltages in the d-q reference frame:

$v_d = -w\,FluxQ$, where $FluxQ = L_q\,i_q$

$v_q = w\,FluxD$, where $FluxD = L_d\,i_d + FluxPM$

The system voltage is constrained by the operating power supply voltage and the PWM scheme. For SVPWM, the available voltage is $v_{max} = \dfrac{V_{dc}}{\sqrt{3}} - i_{max} R_s$, while for Sine PWM, the available voltage is $v_{max} = \dfrac{V_{dc}}{2} - i_{max} R_s$.

$$v_d^2 + v_q^2 = v_s^2 \leq v_{max}^2$$

The system voltage vs that is equal to or less than $vmax$ satisfies the voltage constraint.

After inserting vd and vq representations in the preceding equation, you get:

$$\left(- w\, L_q\, i_q\right)^2 + \left(w\, L_d\, i_d + w\, FluxPM\right)^2 = v_s^2 \leq v_{max}^2$$

where,

- *FluxD* and *FluxQ* are the *d*-axis and *q*-axis flux linkages respectively (in Weber).
- *FluxPM* is the permanent magnet flux linkage (Weber).
- i_d and i_q are the *d*-axis and *q*-axis currents respectively (in Amperes).
- i_{max} is the maximum phase current (peak) of the motor (in Amperes).
- v_d and v_q are the *d*-axis and *q*-axis voltages respectively (in Volts).
- v_s is the source voltage (in Volts).
- v_{max} is the maximum fundamental line to neutral voltage (peak) supplied to the motor (in Volts).
- V_{dc} is the DC voltage supplied to the inverter (in Volts).
- L_d and L_q are the *d*-axis and *q*-axis inductances respectively (in Henry).
- w is the rotor speed (in Radians/ sec).
- R_s is the stator phase winding resistance (in Ohms).

This equation represents an ellipse for an IPMSM, centered on the *x*-axis of the *id-iq* plane.

In the figure 10(b), the voltage limit curve with actual equations is a tilted ellipse, while the voltage limit curve with approximate equations is an ellipse along the *x*-axis and centered on *x*-axis.
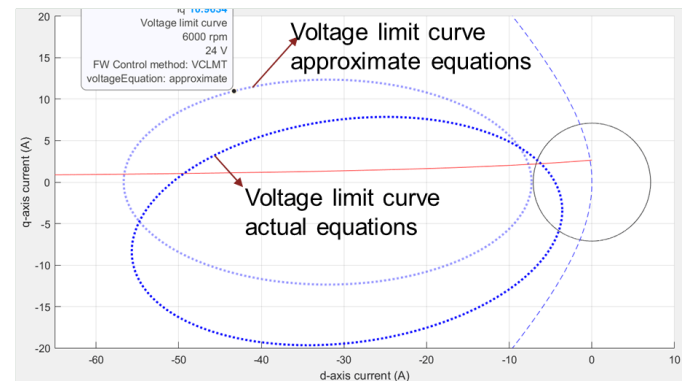
Figure 10(a): A comparison of voltage limit curves for actual and approximate equations for a system with low Rs. Figure 10(b): A comparison of voltage limit curves for actual and approximate equations for a system with high Rs.

MTPV Curve

MTPV stands for Maximum Torque Per Voltage. MTPV curve is a group of all points that satisfy the voltage constraint while providing the maximum possible torque. When considering the linear motor parameters, you can derive an empirical expression for the MTPV curve. However, this can become complicated when the voltages are represented in their actual form. You can derive the empirical form by equating the partial derivative of torque with respect to the voltage (*vd* or *vq*) to zero.

The following expression defines the MTPV curve:

$v_d^2 + v_d K_1 + v_d v_q K_2 = v_q^2 + v_q K_3$, where *K1, K2* and *K3* are terms involving *Ld, Lq,* speed, *Rs,* and *FluxPM*. This equation requires expansion into *id* and *iq* terms. However, for an SPMSM, the MTPV curve equation is $i_d = \dfrac{-\left(w^2\, L\, FluxPM\right)}{\left(R_s^2 + w^2\, L^2\right)}$, which is a straight line parallel to *y*-axis with a position that changes based on the motor parameters and the operating speed.

where,

- *FluxPM* is the permanent magnet flux linkage (Weber).
- i_d is the *d*-axis current (in Amperes).
- v_d and v_q are the *d*-axis and *q*-axis voltages respectively (in Volts).
- L_d and L_q are the *d*-axis and *q*-axis inductances respectively (in Henry).
- *L* is the total motor inductance (in Henry).
- *w* is the rotor speed (in Radians/ sec).
- R_s is the stator phase winding resistance (in Ohms).
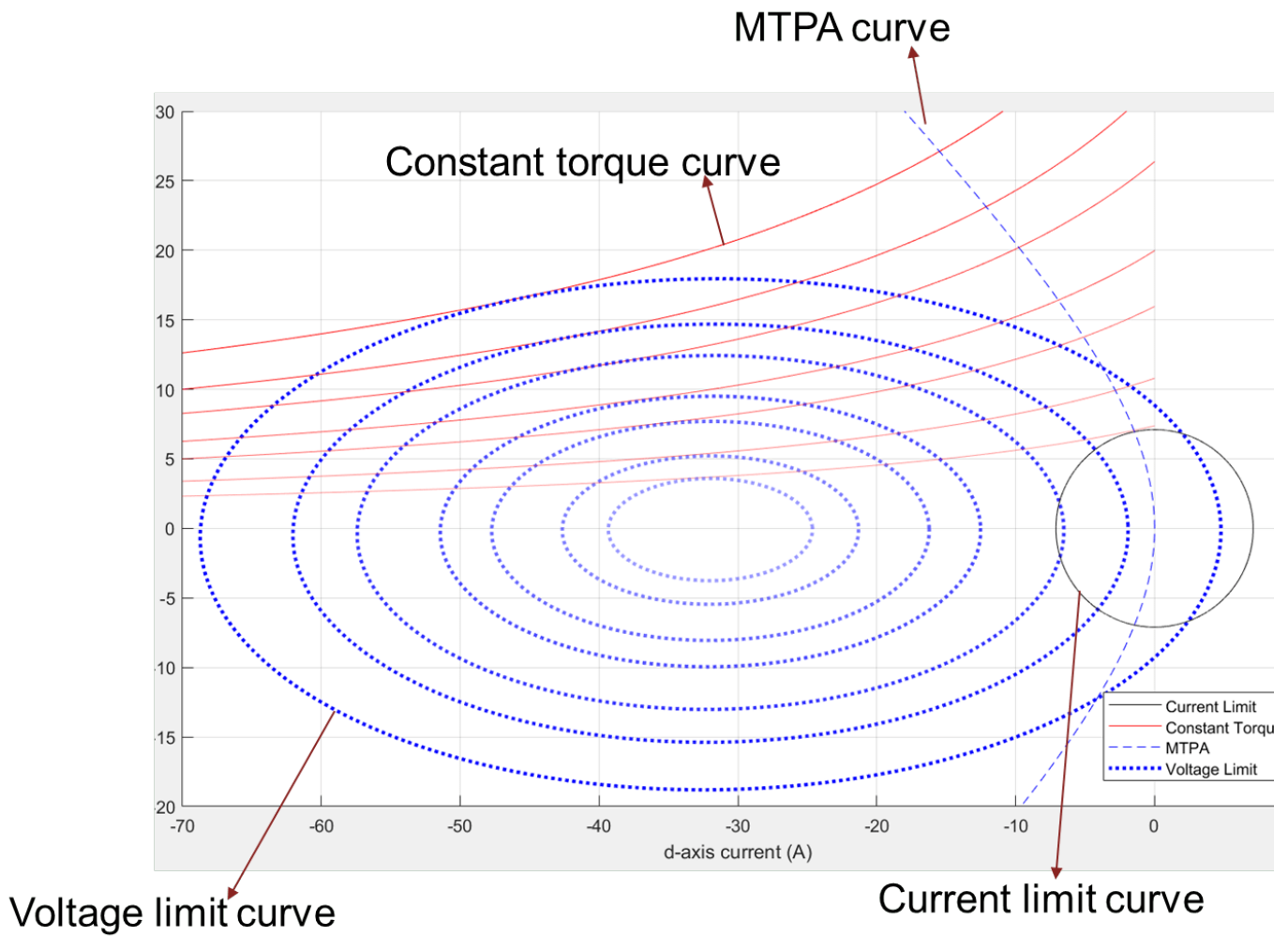
Figure 11: Current limit circle, MTPA curve, and a group of pairs of voltage limit curves and constant torque curves that tangentially touch the corresponding voltage limit curve.
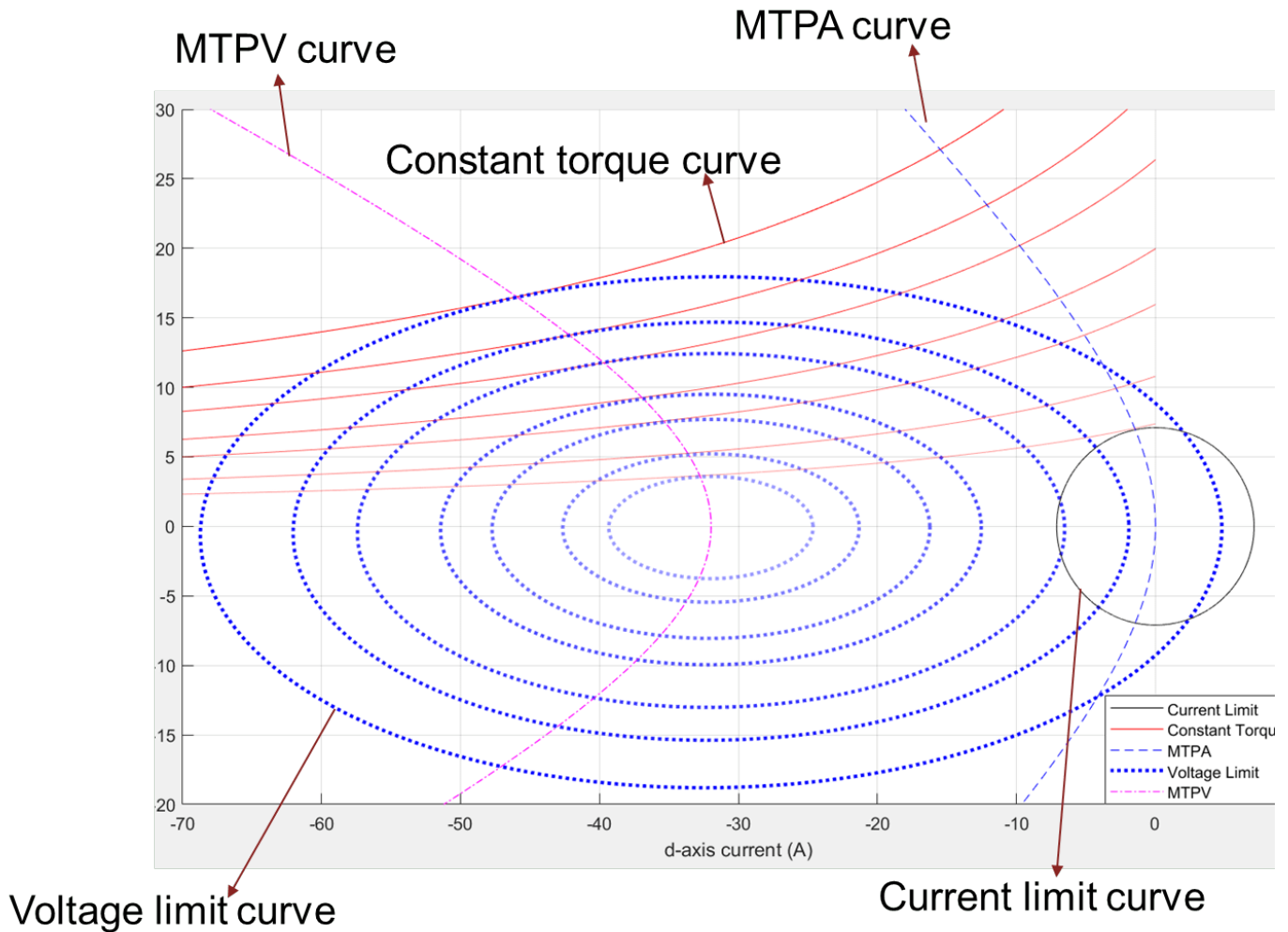
Figure 12: Current limit circle, MTPA curve, a group of pairs of voltage limit curves and constant torque curves that tangentially touch the corresponding voltage limit curve, and MTPV curve.

Using Constraint Curves

You can use the constraint curves in multiple ways. This section lists some of these methods.

Find Maximum Operable Speed

You can find the maximum operable speed of a PMSM (SPMSM or IPMSM) in different ways. You can use the analytical equations to find the maximum speed. Using the constraint curves is another method. Note that you can use the machine for a limited speed due to several constraints (such as efficiency, mechanical stability, and maximum required speed).

From the constraint curves, you can obtain the maximum operable speed of a machine which is a point where the internal frictional torque (with no load torque) can be supported by the motor's produced torque. When the speed increases beyond the maximum speed, the frictional load torque increases and the motor fails to support the load torque.

FW Control method: VCLMT, VoltageEquation: actual

Legend:
- Current Limit(I_{rated}=7.1A)
- Constant Torque(T_{load}=0N.m, $T_{friction}$=0.0018357N.m)
- MTPA
- Voltage Limit(V_{DC}=24V, w=6648rpm)
- MTPV

x-axis: d-axis current (A)
y-axis: q-axis current (A)
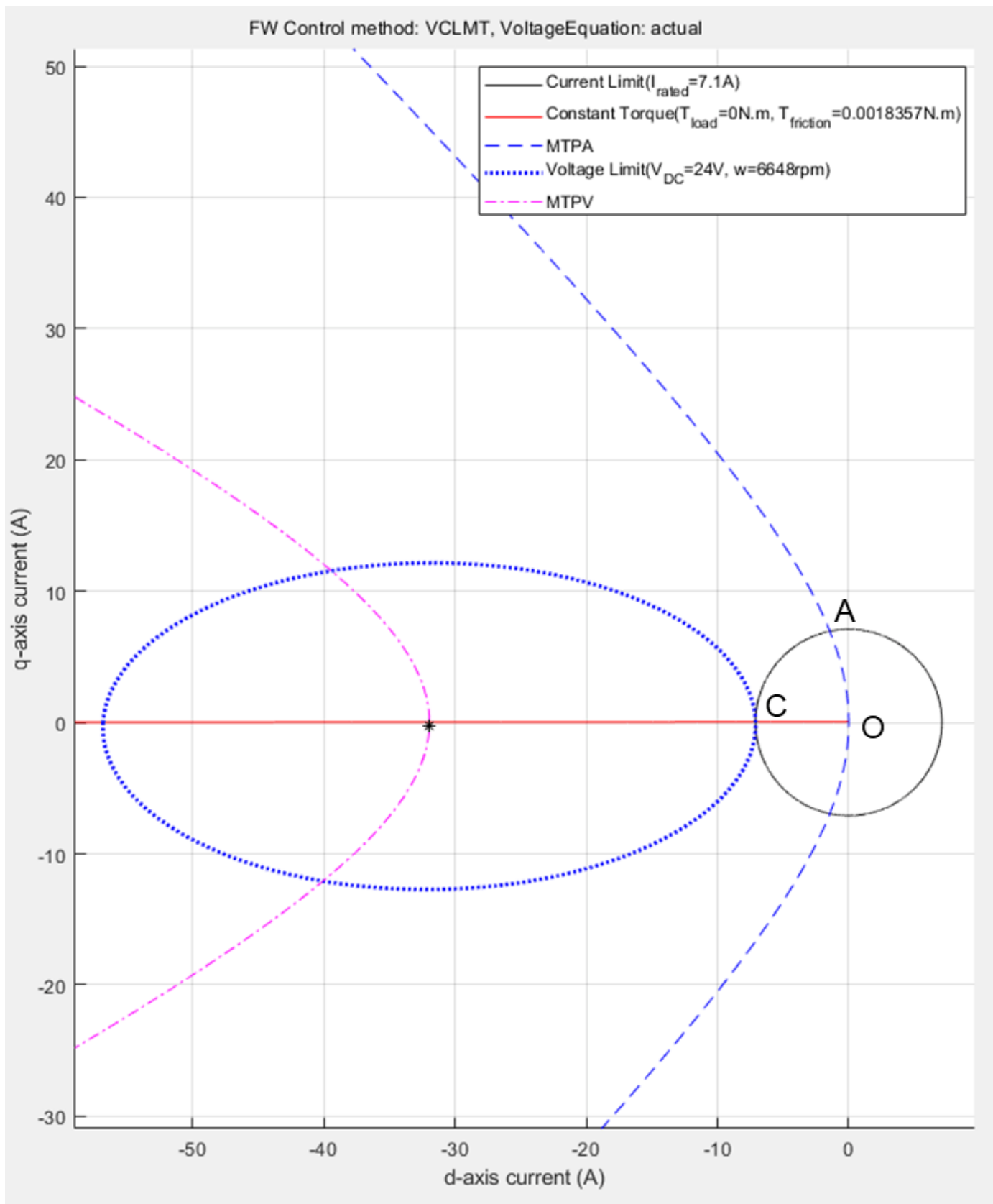
Figure 13 : Constraint curves of a PMSM, drawn at maximum operable speed, at 0 N.m. load torque.
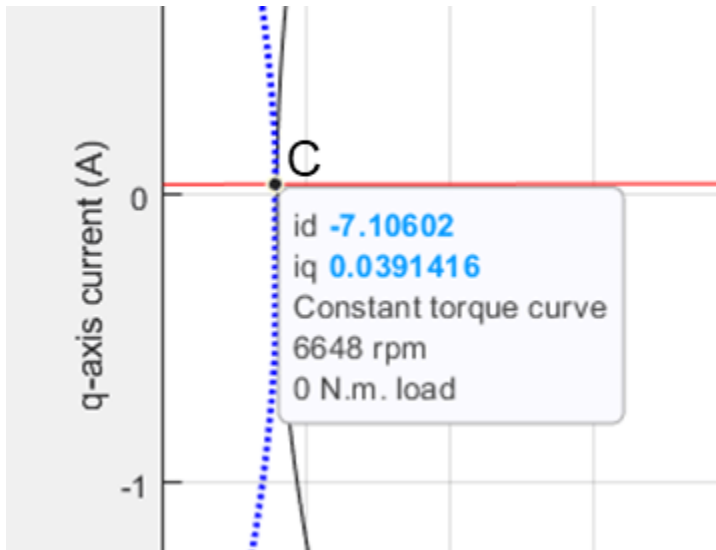
Figure 14: Zoomed-in portion from figure 13 to show the intersection of three curves - current limit circle, voltage limit ellipse, and constant torque curve.
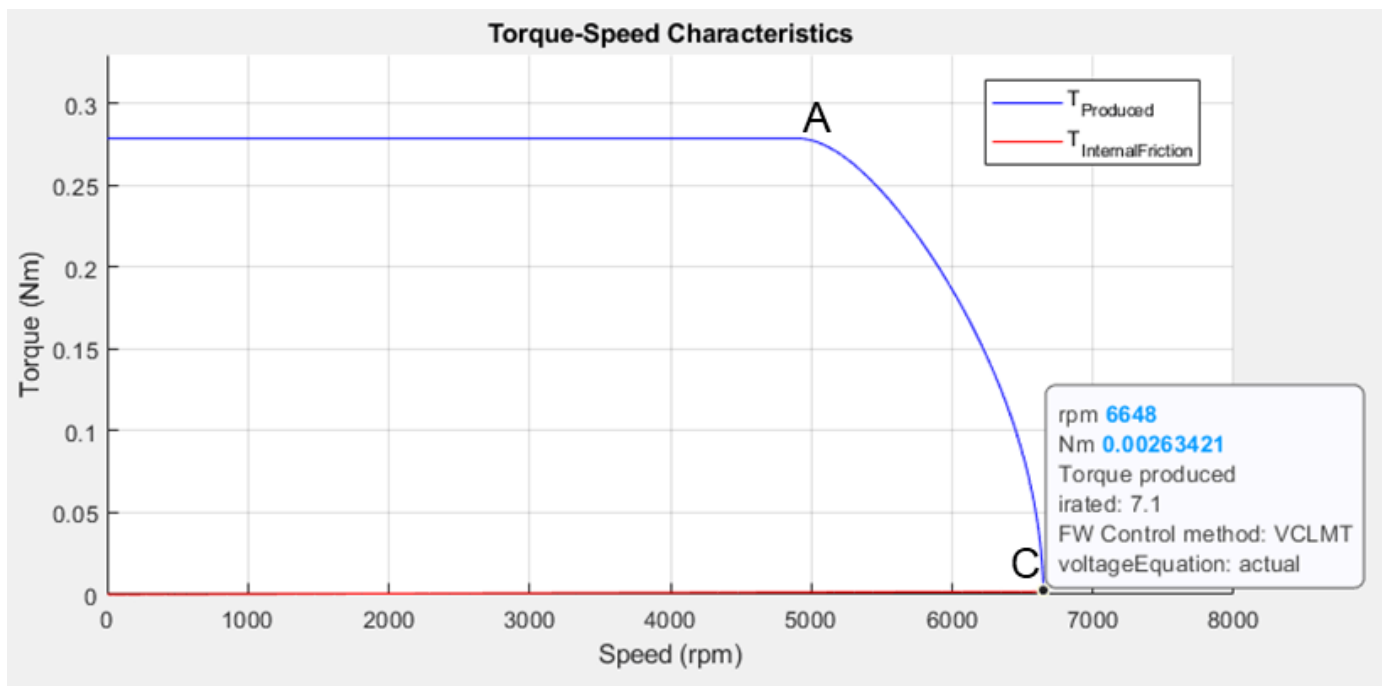
Figure 15: Drive characteristics of the motor that is marked with the operating points at the maximum speed.

The following code can be used to find the maximum speed of the motor.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.Lq=pmsm.Ld*2;
```

4-357

```
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
max_speed=mcbPMSMMaxSpeed(pmsm,inverter);
disp('maximum speed is ' + string(max_speed) + ' rpm');
```

```
maximum speed is 6648 rpm
```

Finding Corner Speed of Motor

The rated speed of a motor is the speed at which the motor can be loaded with the rated torque, while operating at rated voltage. This speed is also known as corner speed because at this speed, PMSM's speed-torque characteristics has a change of shape. For example, in figure 18, the point A marks the corner speed of the motor. This speed is also called the base speed. You can use analytical equations to find the corner speed. Using the constraint curves is another method.

From the constraint curves, you can determine the corner speed of a machine, which is a point where the voltage limit curve crosses the intersection of MTPA curve and the current limit circle. When the speed increases beyond the corner speed, the voltage limit curve does not enclose the intersection of MTPA curve and the current limit circle.

FW Control method: VCLMT, VoltageEquation: actual

Legend:
- Current Limit(I_{rated}=7.1A)
- Constant Torque(T_{load}=0.27744N.m, $T_{friction}$=0.0013547N.m)
- MTPA
- Voltage Limit(V_{DC}=24V, w=4906rpm)
- MTPV

x-axis: d-axis current (A)

y-axis: q-axis current (A)
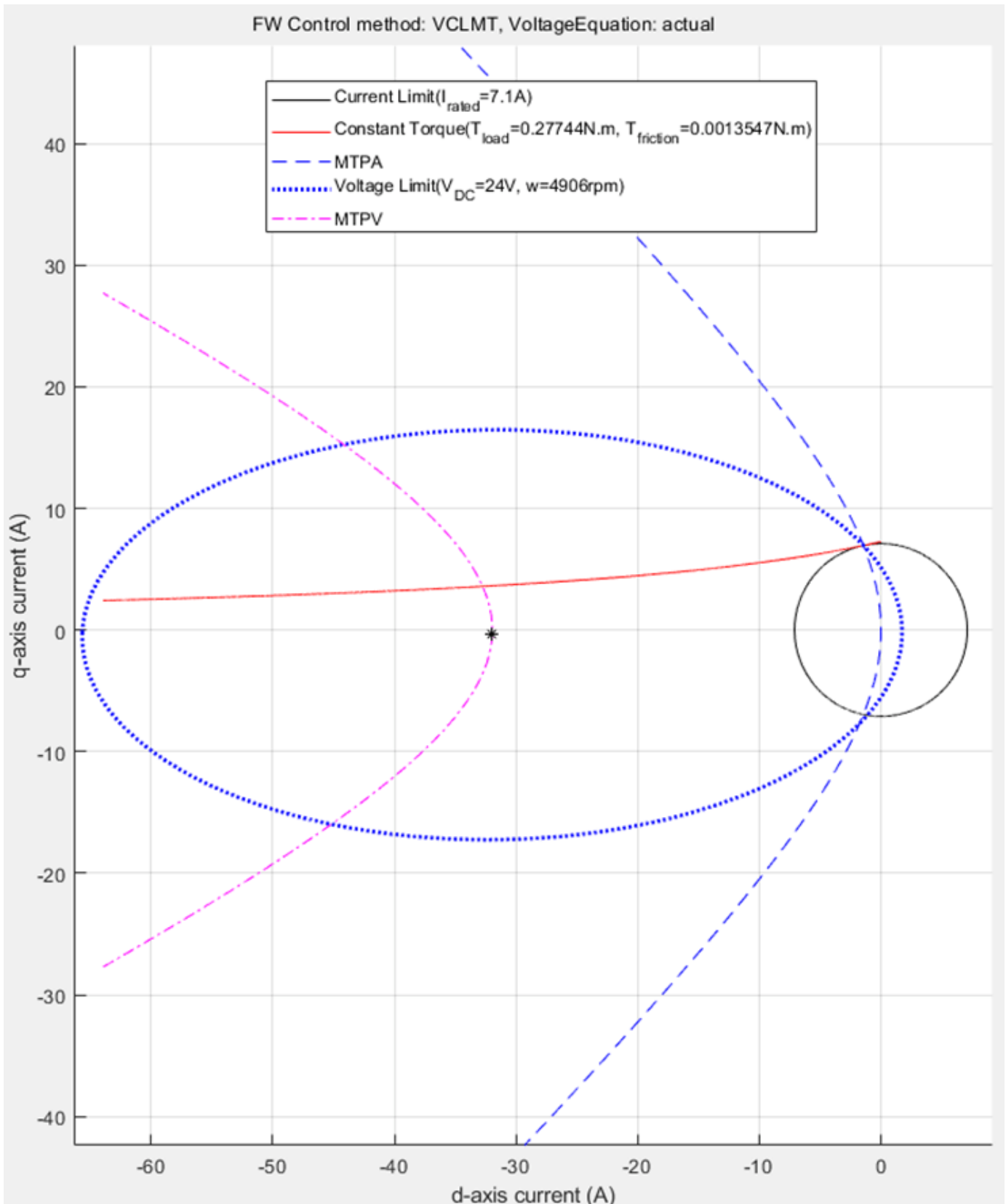
Figure 16: Constraint curves of a PMSM drawn at base speed (corner speed), and at rated torque.
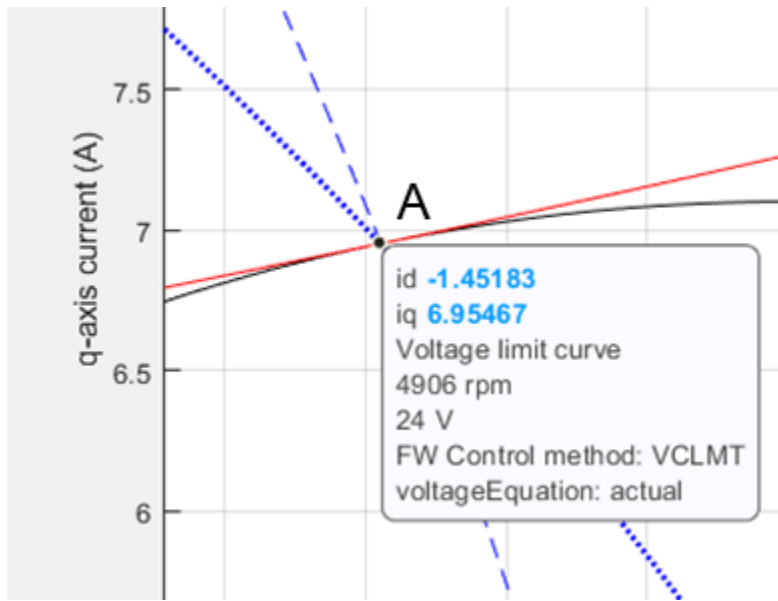
Figure 17: Zoomed in portion of figure 16 to show the intersection of MTPA curve, current limit circle, and voltage limit curve.
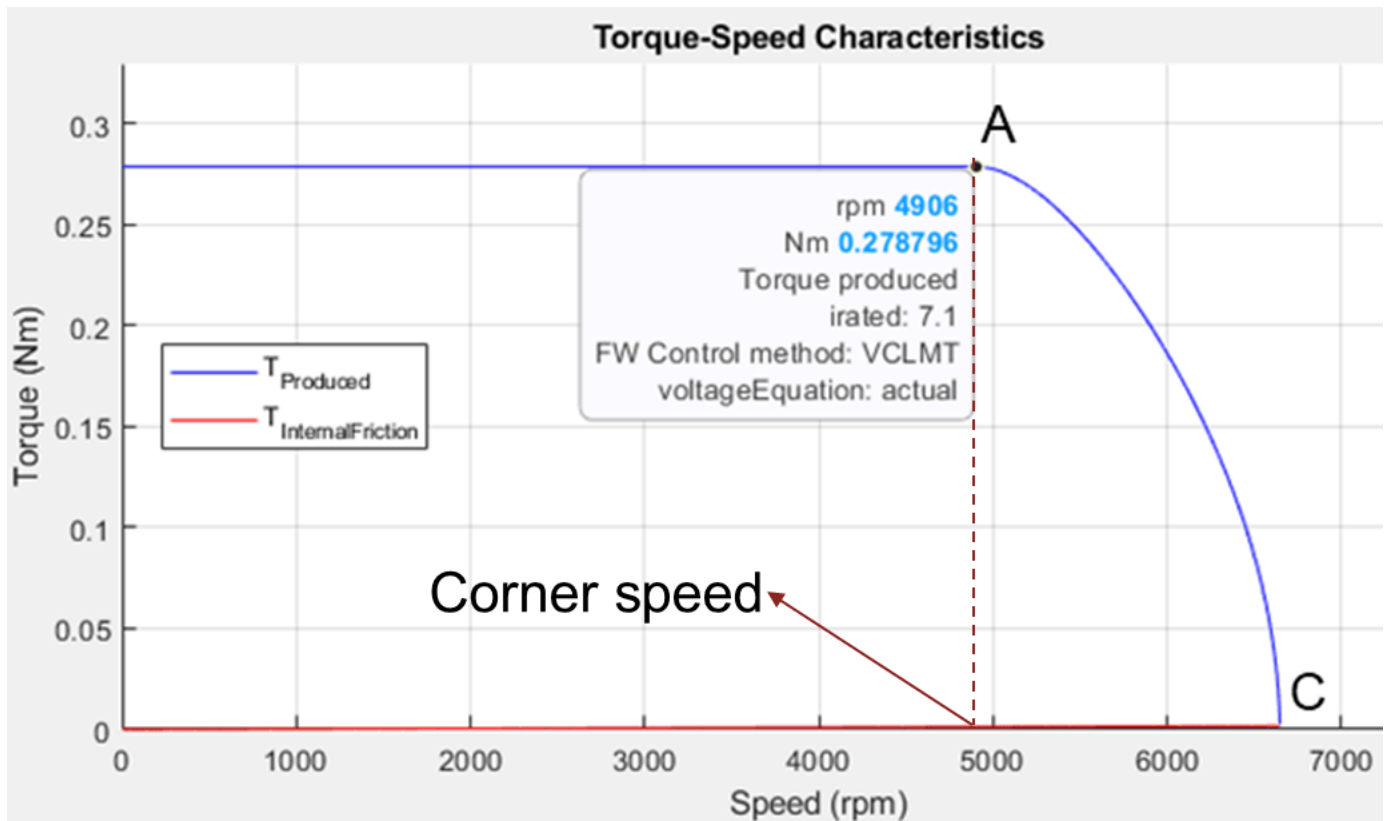
Figure 18: The speed-torque characteristics of a PMSM that is marked at corner speed at rated torque.

You can use the following code to find the corner speed of the PMSM:

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
mcbPMSMSpeeds(pmsm,inverter,verbose=1);

Proceeding with the motorType as 'ipmsm'.
Corner speed [rpm] is: 4905.
Maximum speed [rpm] with VCLMT Field Weakening is: 6648.
```

Identifying Operating Points (id,iq)

For a given speed and load torque, you can find the operating points of a machine. The following sections explain five selected operating points.

Case 1: MTPA Region of Operation

MTPA stands for Maximum Torque Per Ampere. A motor operates in MTPA region, when the $v_s = \sqrt{\left(v_d^2 + v_q^2\right)}$ is less than v_max.

To compute the optimal operating point, you can draw the constraint curves for the given speed and torque conditions.
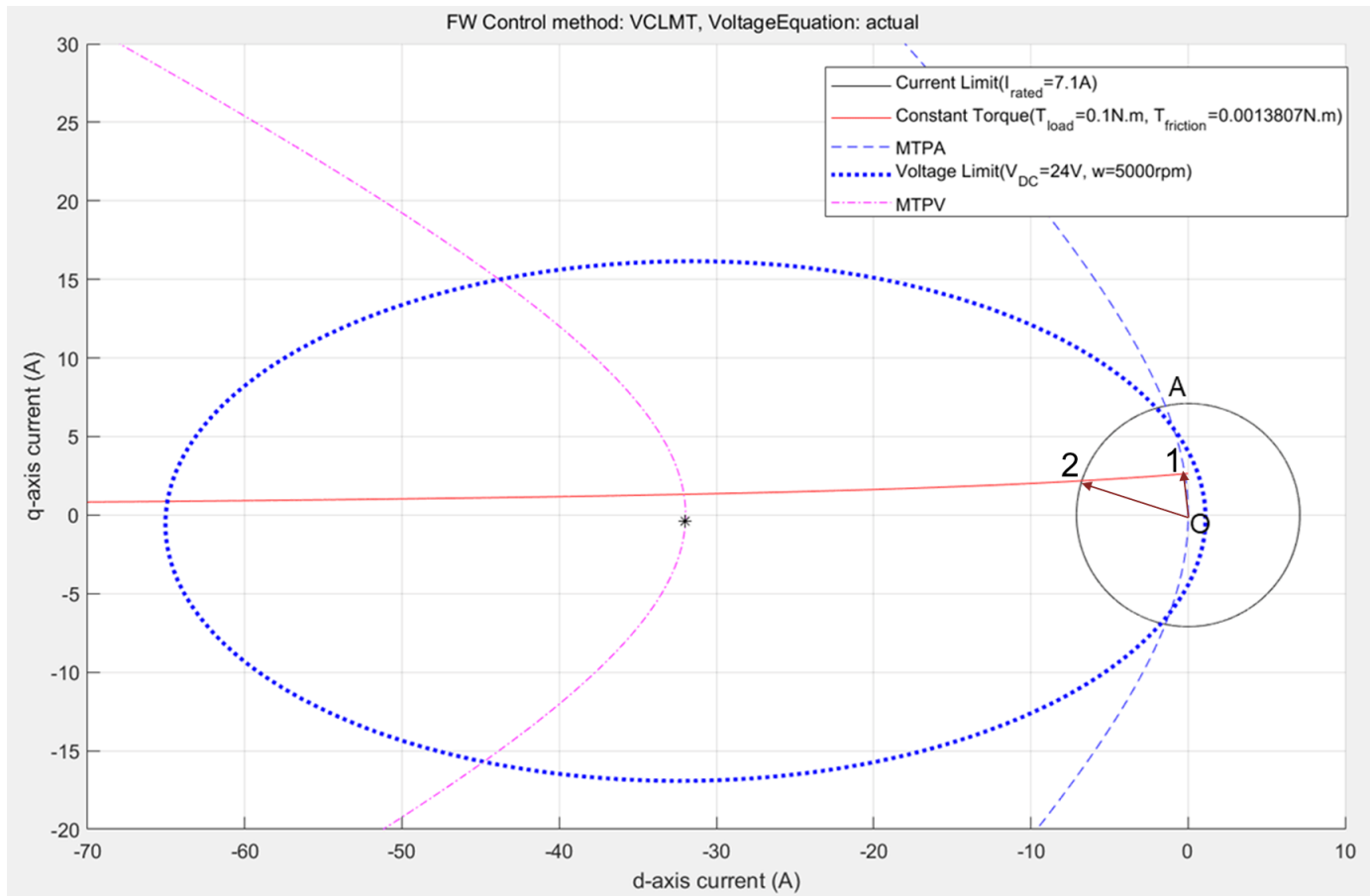
Figure 19: Constraint curves of a PMSM drawn at a torque of 0.1N.M. and at a speed of 5000rpm. The corner speed at the given torque is higher than the given speed, resulting in the possibility of MTPA operation.
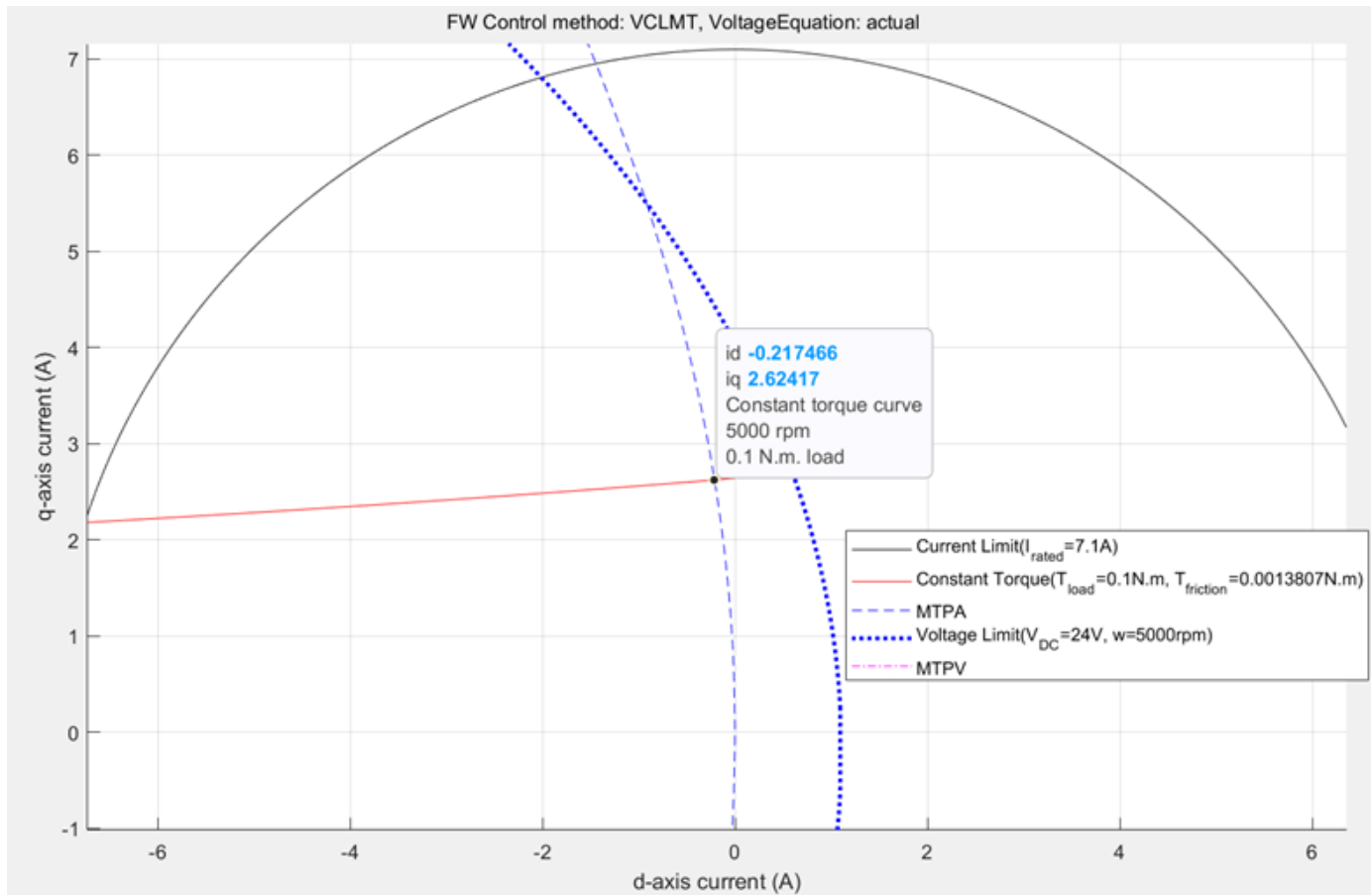
Figure 20: Zoomed-in portion of the figure 19 to show the selected operating point for the given speed and torque.

Any point along the portion of constant torque curve (between the marked points 1 and 2) that is within the region of overlap between the voltage limit ellipse and current limit circle results in the same torque output, while satisfying the voltage and current constraints. Point 1 results in the lowest current magnitude, and therefore, it is an optimal point of operation, which satisfies the voltage limit constraint and current limit constraint. In addition, this point meets the load torque demand and speed requirement.
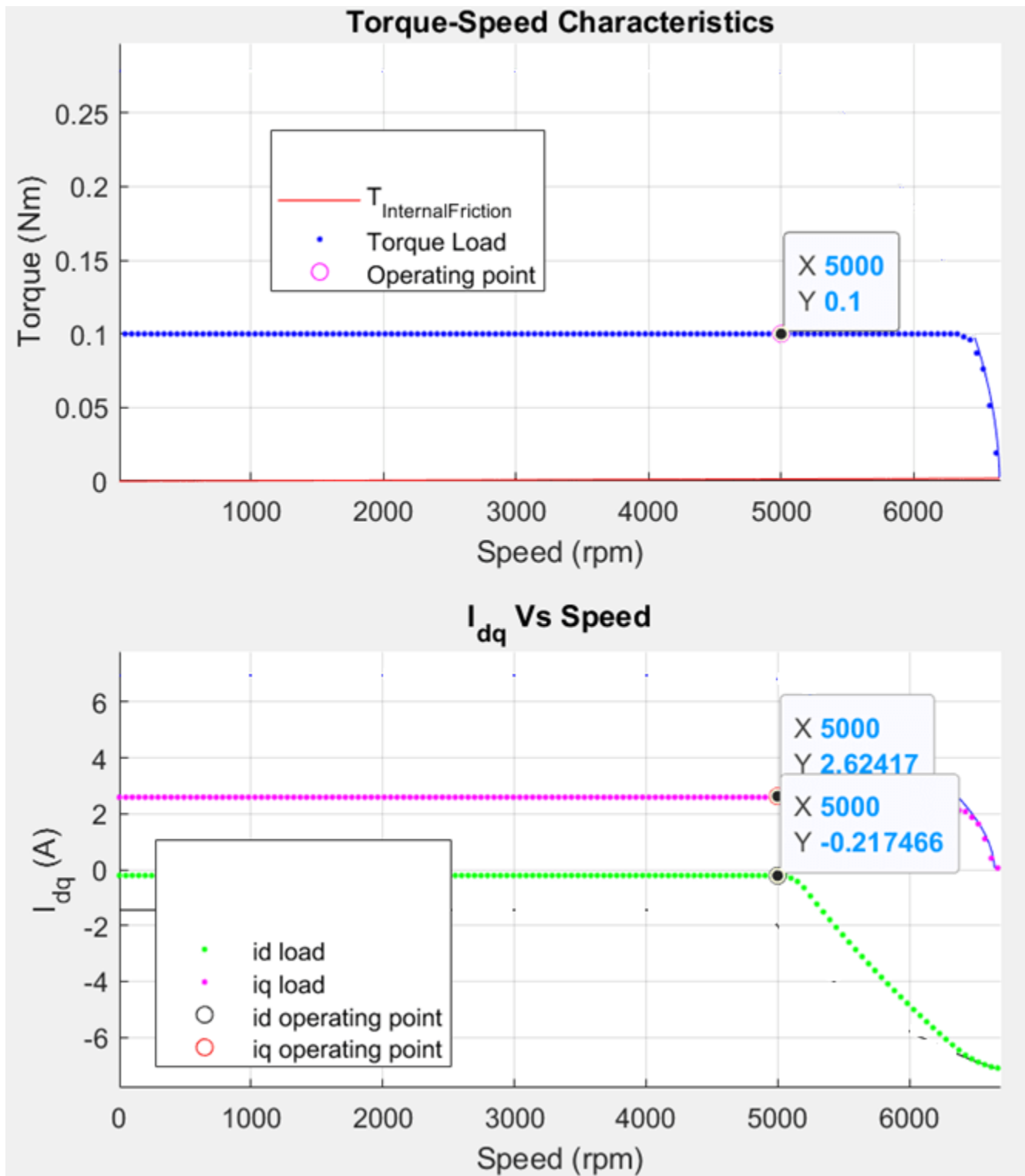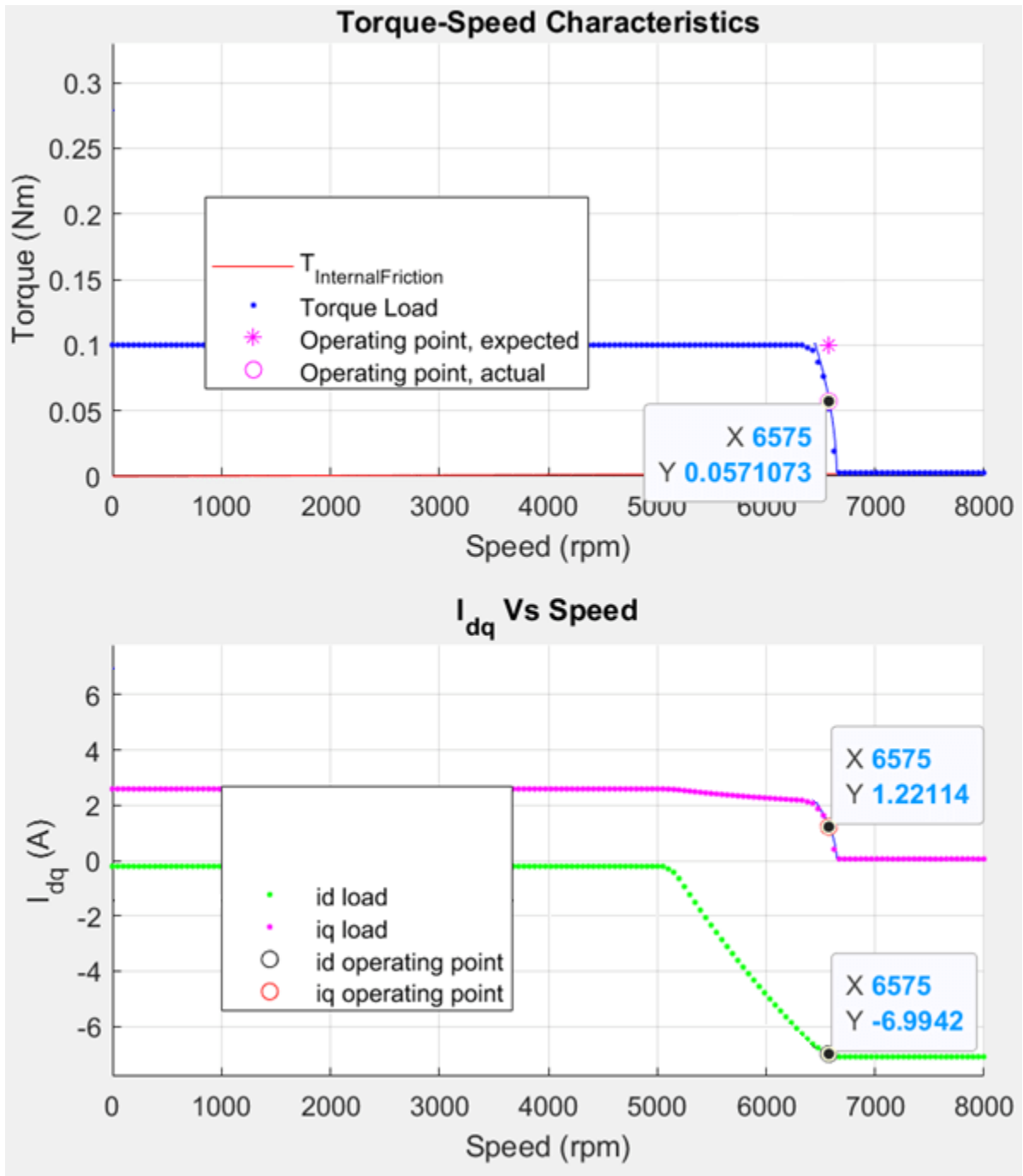
Figure 21 (a): Drive characteristics of a PMSM with the speed and torque point marked.

Figure 21 (b): Current curves of the PMSM for the loaded torque, with the operating point marked on them.

From the constraint curves shown in figure 20, you can notice that for the given speed, the voltage limit ellipse does not touch the operating point (id, iq), but encloses it. At a slightly higher speed (until the voltage limit curve touches the identified operating point) the same operating point (id, iq) is still the optimal operating point. Note that when the voltage limit curve touches the identified operating point (id, iq), v_s is identical to v_{max}, and the corresponding speed is the corner speed at the load torque. At a speed higher than this corner speed, the voltage limit is exceeded if you follow the MTPA trajectory, and therefore, you can start the field weakening (FW) operation.

You can use the following code to plot the figures.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
mcbPMSMCharacteristics(pmsm,inverter,torque=0.1,speed=5000,driveCharacteristics=0);
```
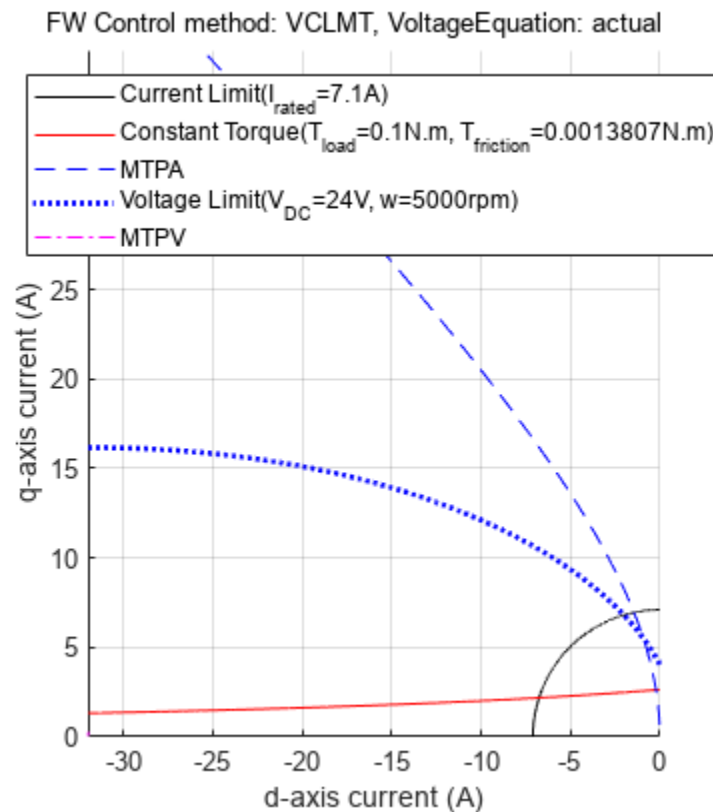
Case 2: Field-Weakening, Load Torque Lower Than Maximum Possible Torque at Given Speed
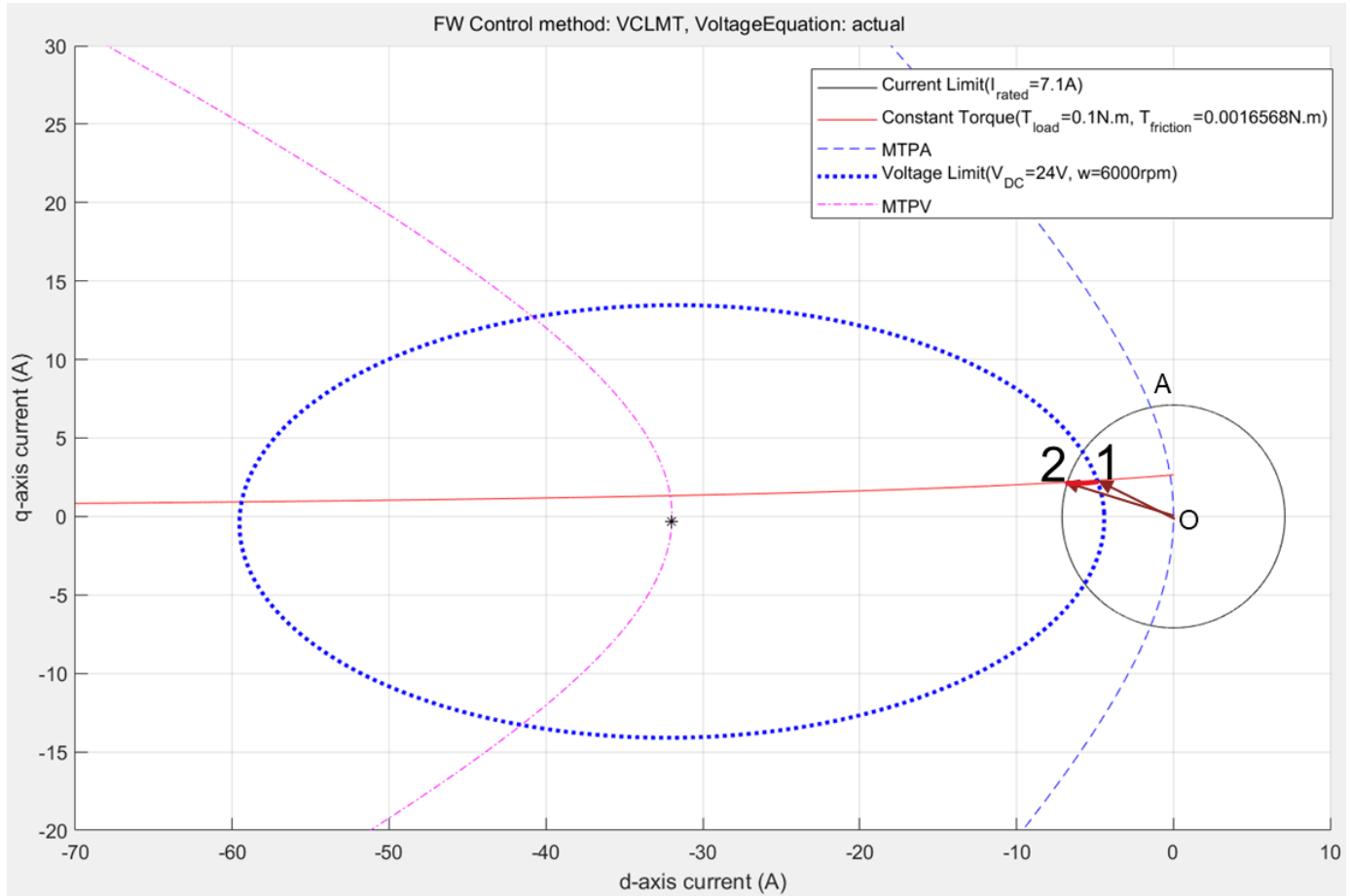
Figure 22: Constraint curve of the PMSM drawn at 0.1N.m. load torque and at 6000 rpm, with the constant torque curve passing through the intersection of current limit circle and voltage limit curve.

When you operate the motor at a speed higher than the corner speed along the MTPA trajectory, the voltage limit exceeds. Then the operating point moves along the constant torque curve to maintain the load torque demand. In figure 22, the constant torque curve passes through the intersection region of voltage limit curve and the current limit circle. This indicates that the when (*id, iq*) values are selected along the constant torque curve within this overlapping region, the load demand is satisfied. However, the extremes of the possible points are marked 1 and 2 in the figure. Point 1 results in the lowest current magnitude, and therefore, the most optimal point for operation while satisfying the voltage limit constraint, current limit constraint. It also meets the load torque demand and the speed requirement.
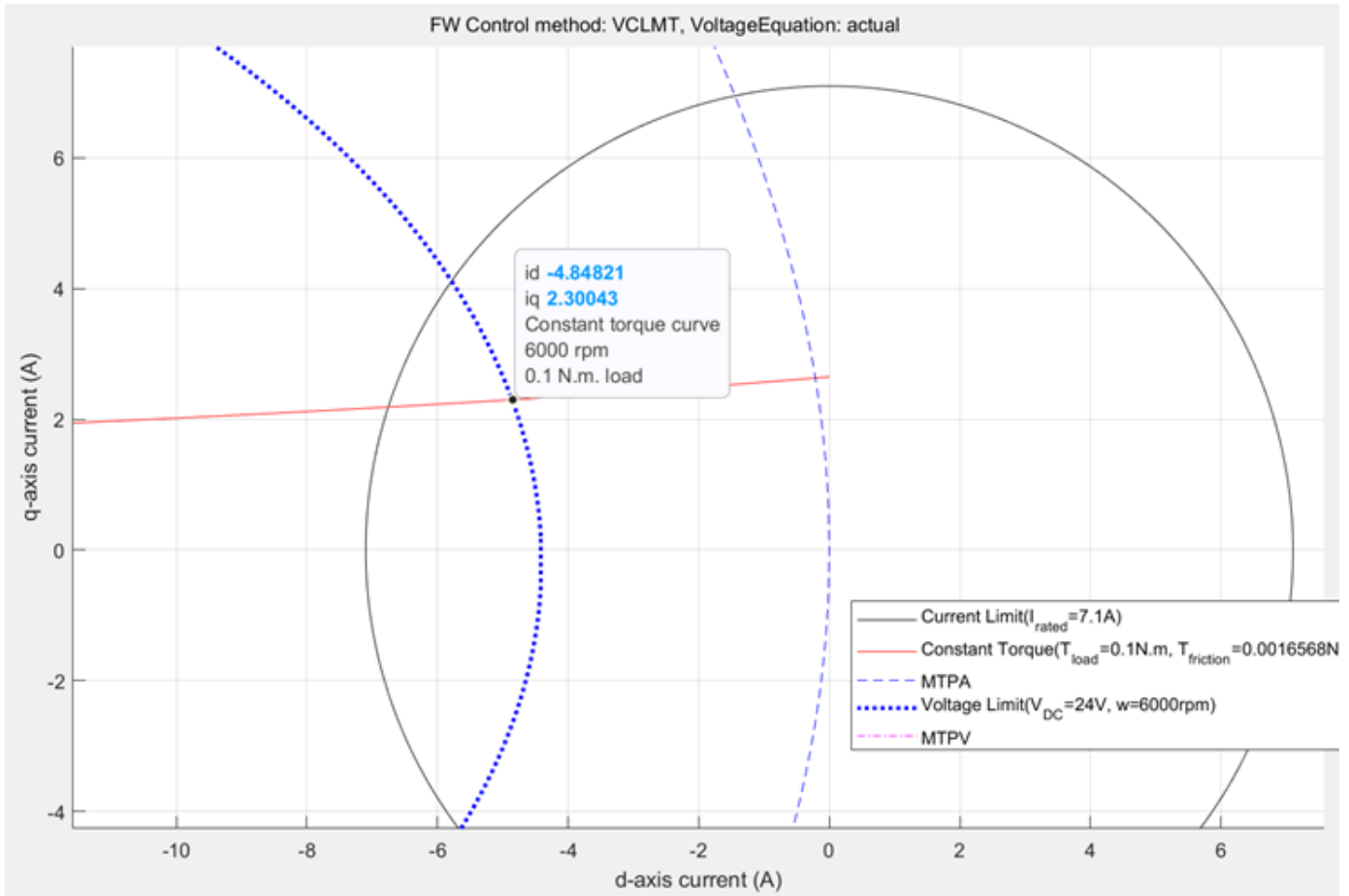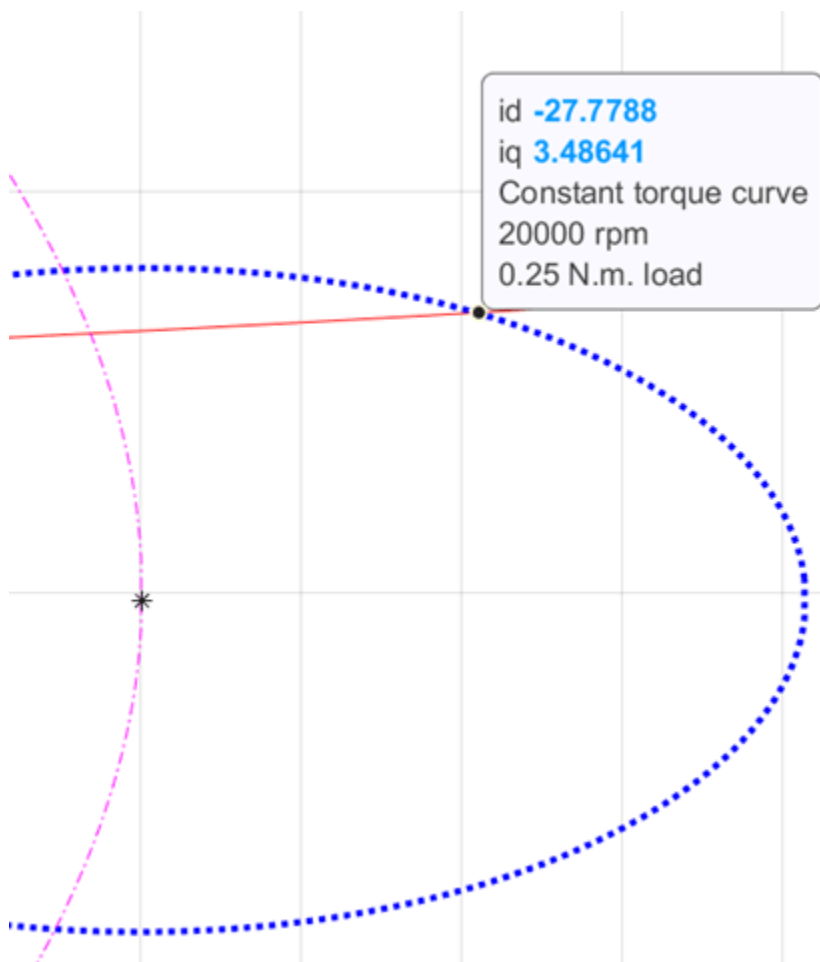
Figure 23: Zoomed in portion of the figure 22 to show the selected operating point for optimal operation.
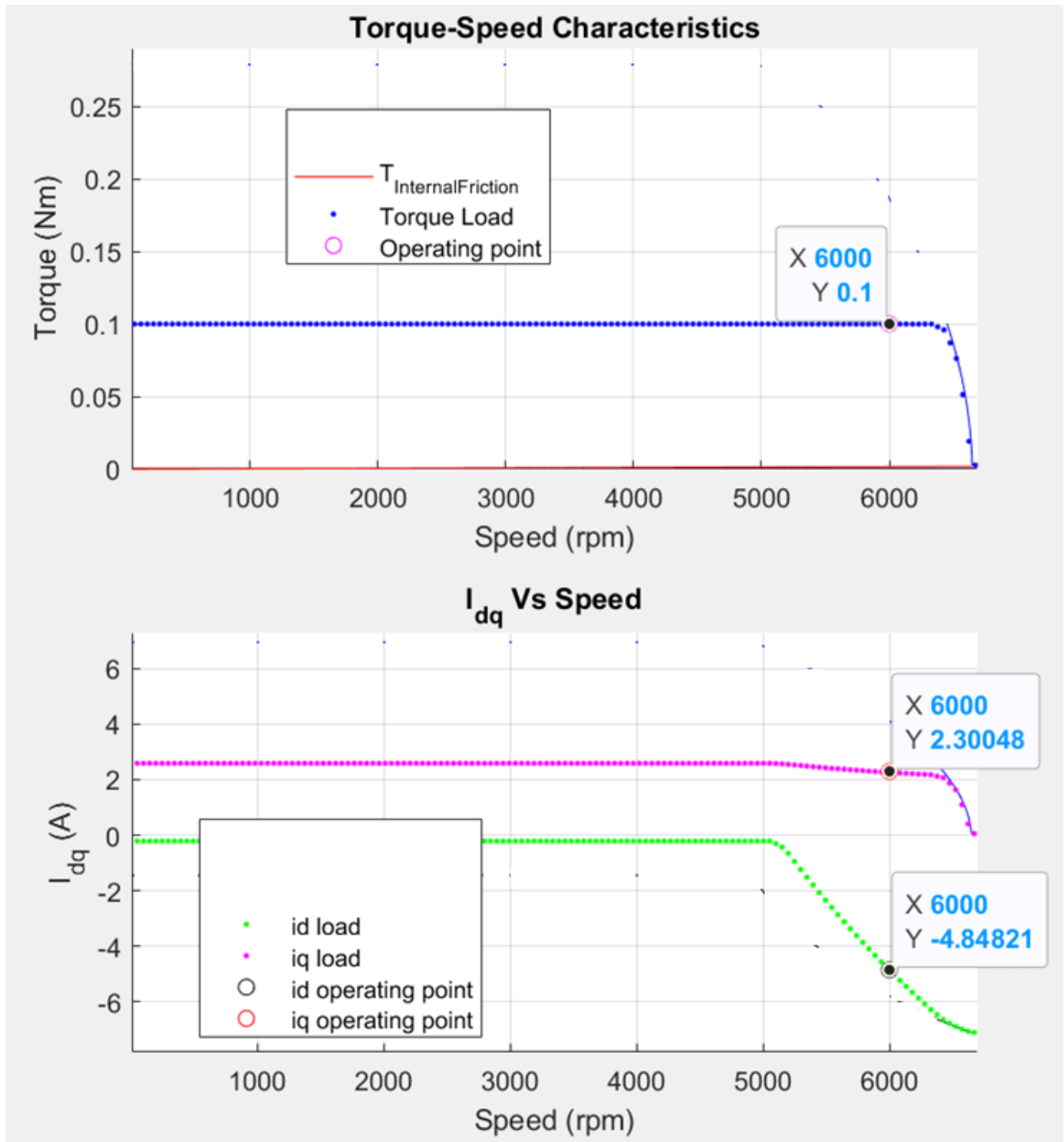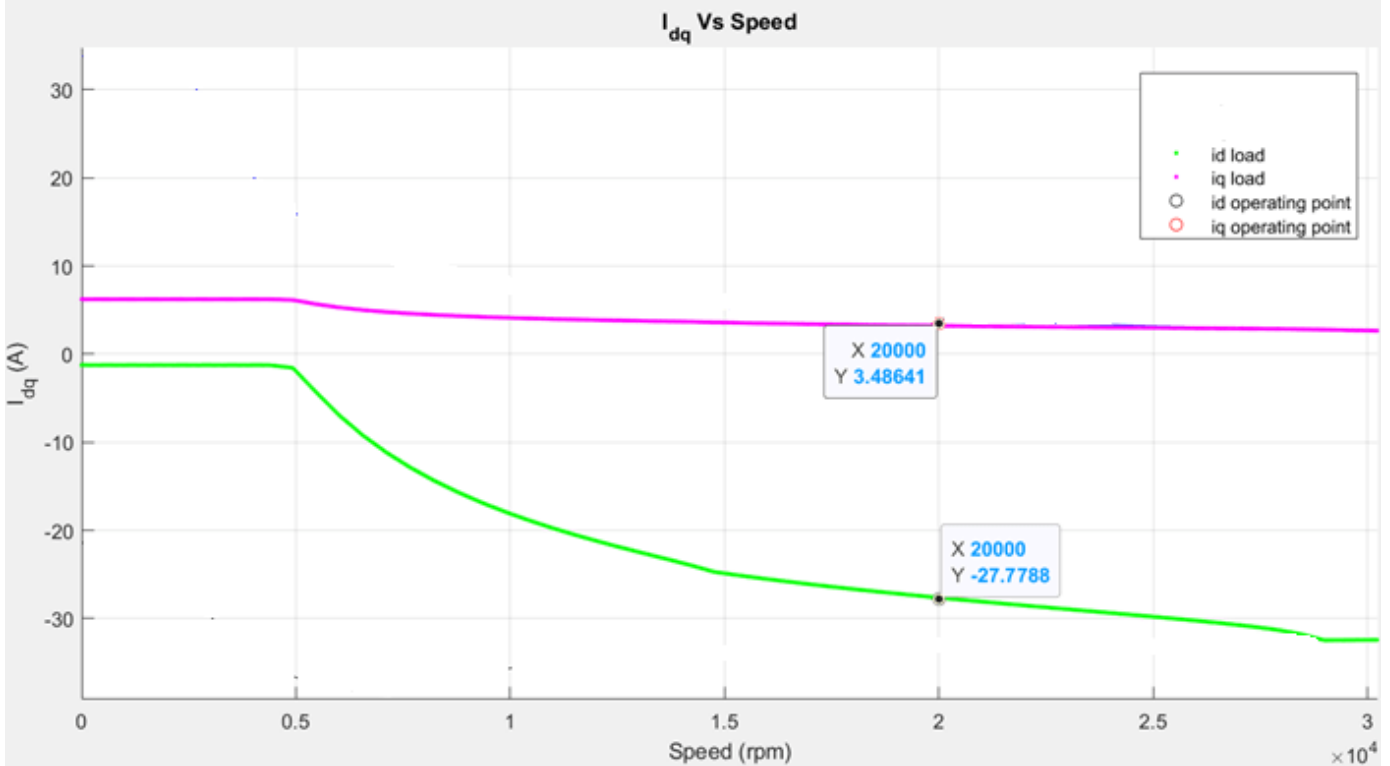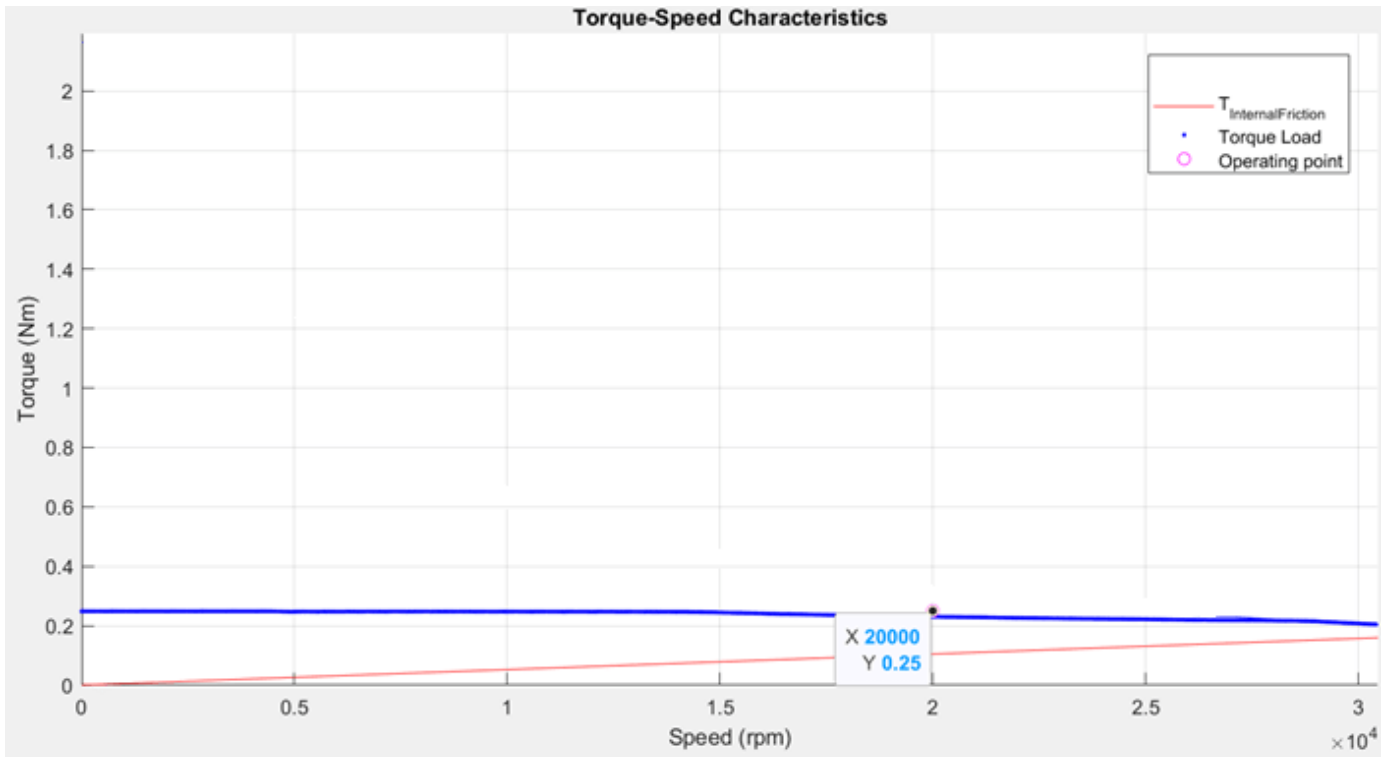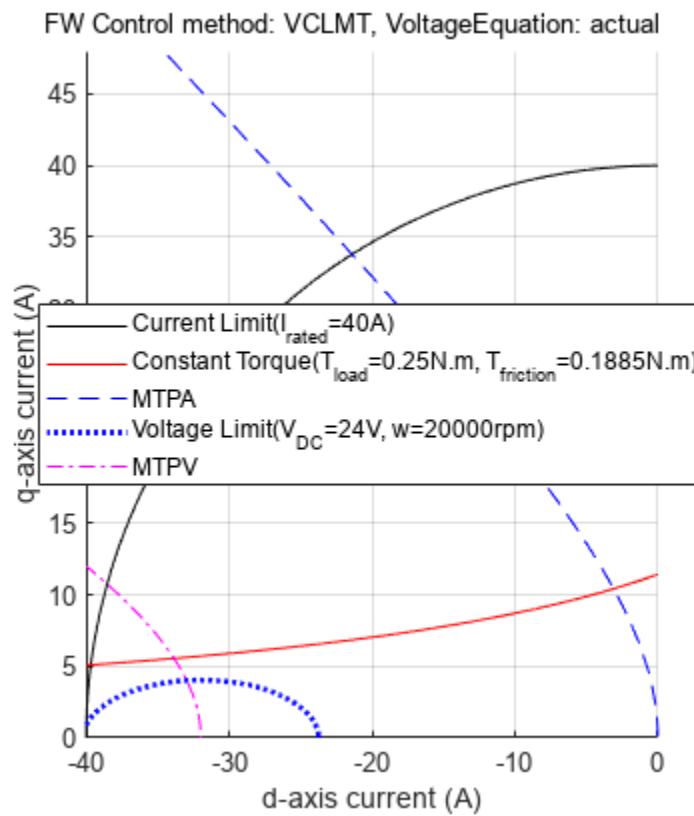
Figure 24 (a): Drive characteristics of a PMSM with the speed and torque point marked.

Figure 24 (b): Current curves of the PMSM for the loaded torque, with the operating point marked on them.

You can use the following code to plot the figures.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
data=mcbPMSMCharacteristics(pmsm,inverter,torque=0.1,speed=6000,driveCharacteristics=0);
```
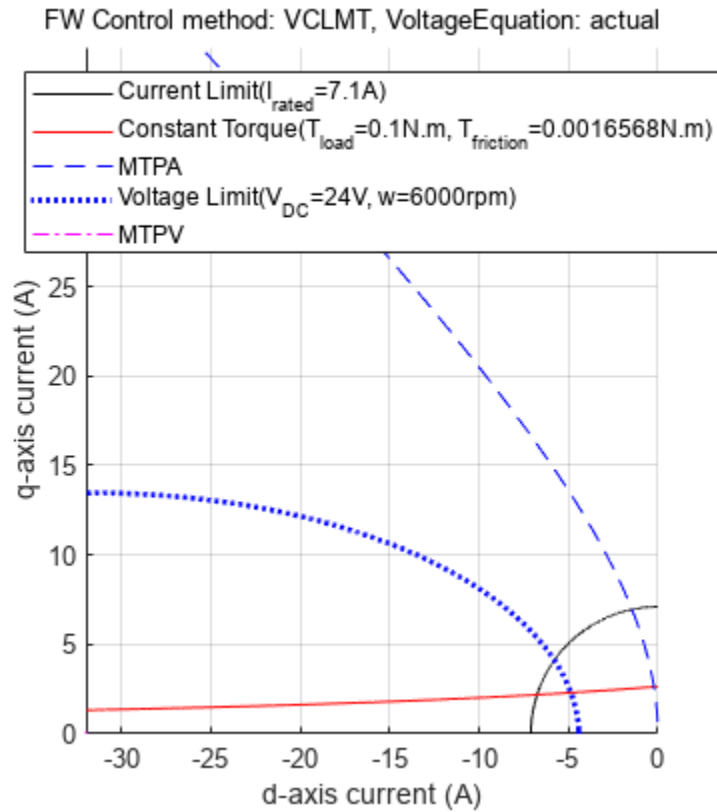
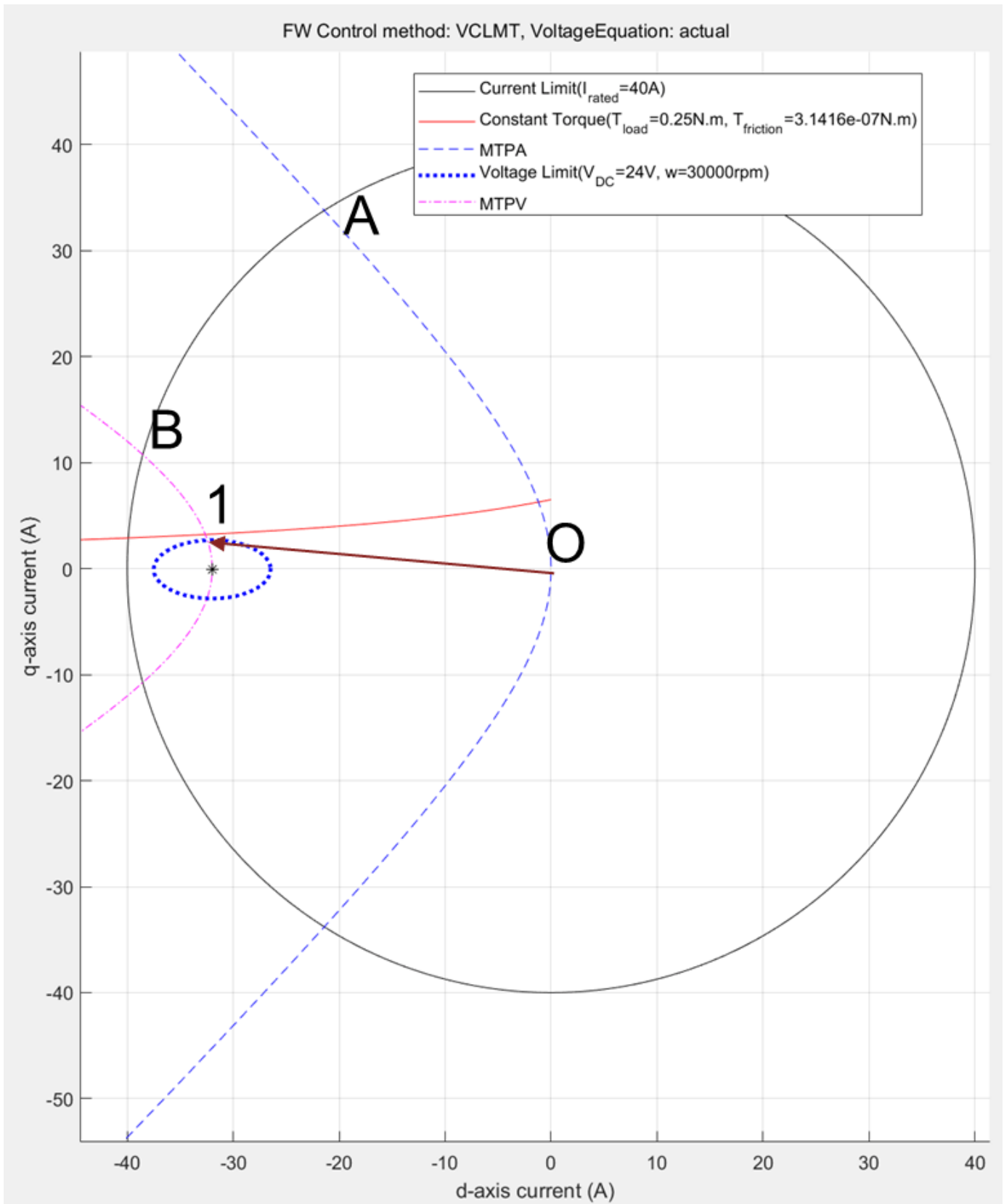**Case 3: Field Weakening, Load Torque More Than Maximum Possible Torque at Given Speed**
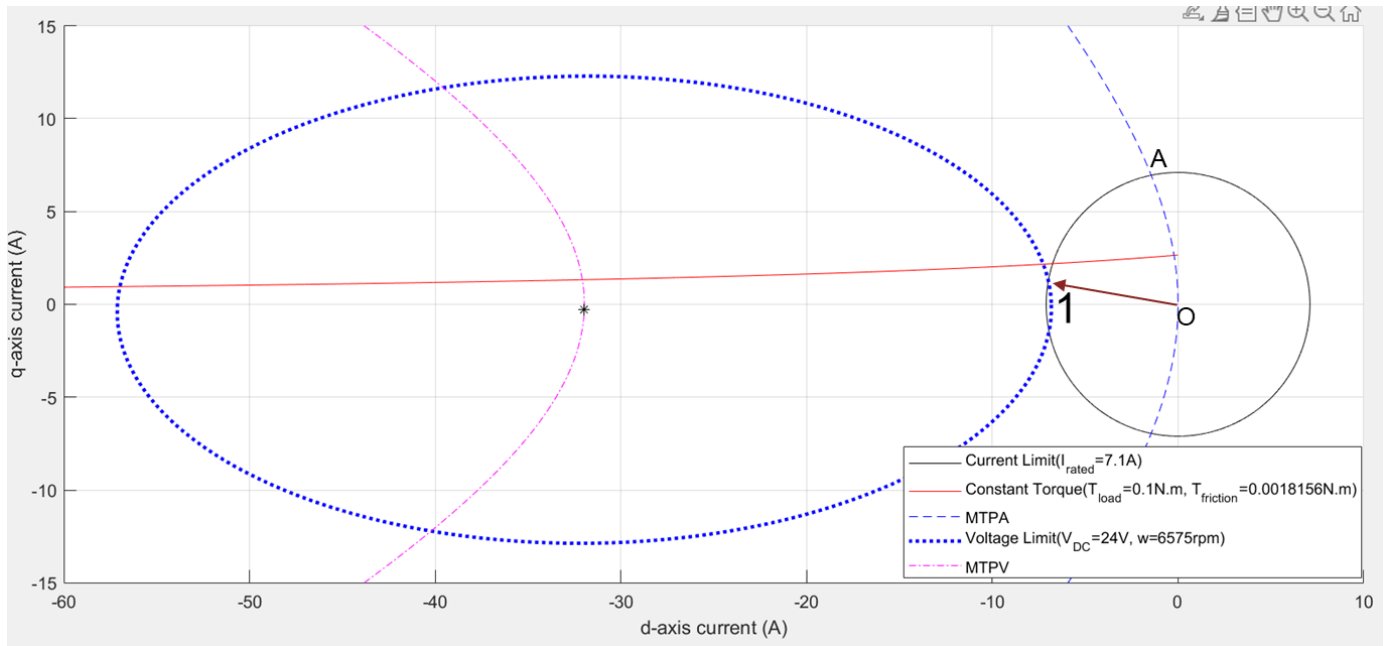
Figure 25: Constraint curves of the PMSM drawn at 0.1 N.m. and 6000rpm speed, with the constant torque curve not passing through the intersection of current limit circle and voltage limit curve.

Figure 25 shows a situation when the demanded torque is higher than the possible torque at the given speed. In the figure, observe that the constant torque curve does not pass though the intersection region. Any point on the constant torque curve does not satisfy the current and voltage constraints. Therefore, only the maximum possible torque is returned as the operating point. This point is marked as 1 in the preceding figure.
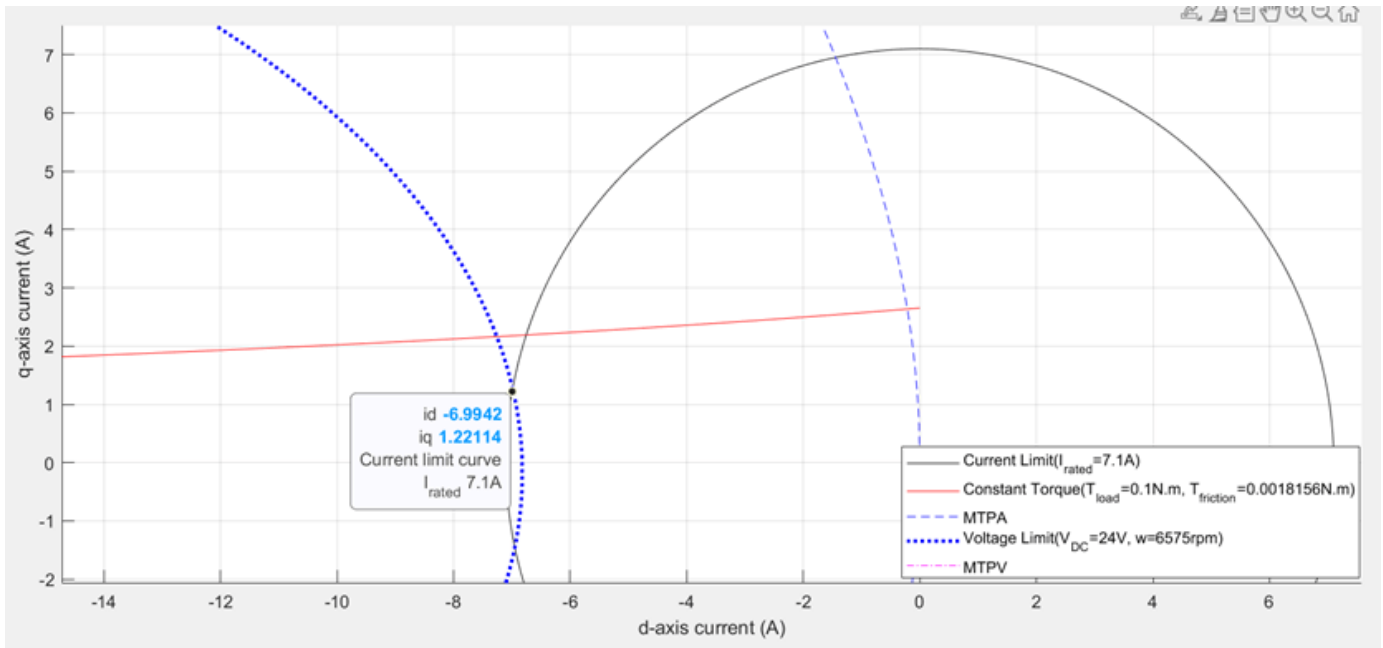
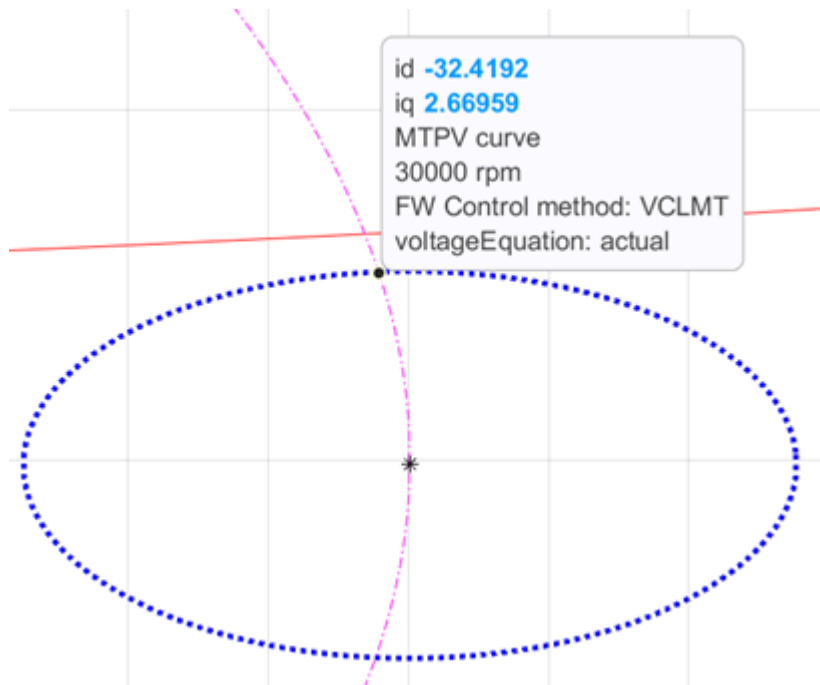Figure 26: Zoomed in portion of the figure 25 showing the selected operating point.

Figure 27 (a): Drive characteristics of a PMSM with the speed and torque point marked.

Figure 27 (b): Current curves of the PMSM for the loaded torque, with the operating point marked on them.

You can use the following code to plot the figures.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
```

```
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
mcbPMSMCharacteristics(pmsm,inverter,torque=0.1,speed=6575,driveCharacteristics=0);
```


FW Control method: VCLMT, VoltageEquation: actual
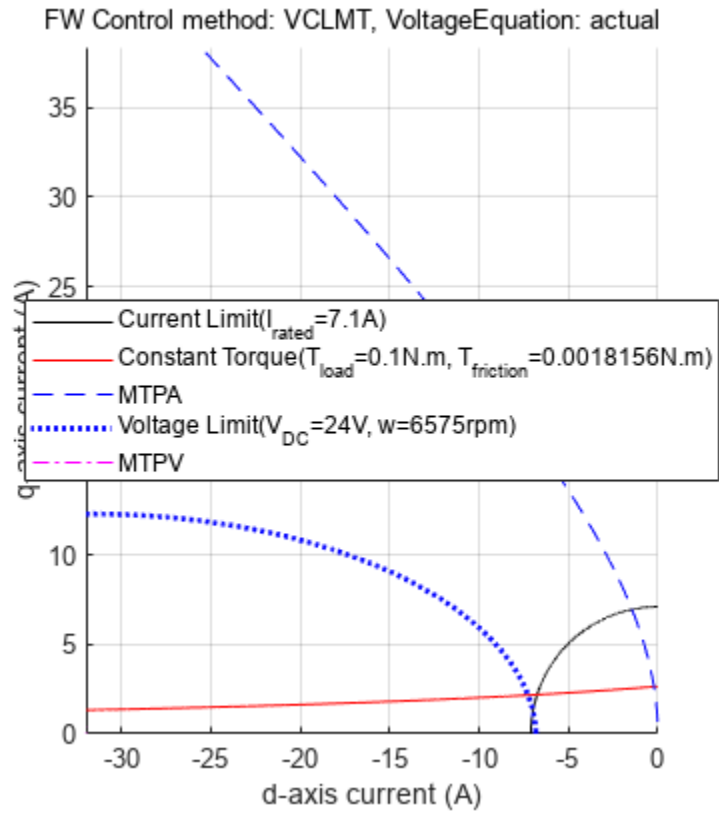
Case 4: MTPV, Load Torque Less Than Maximum Possible Torque at Given Speed
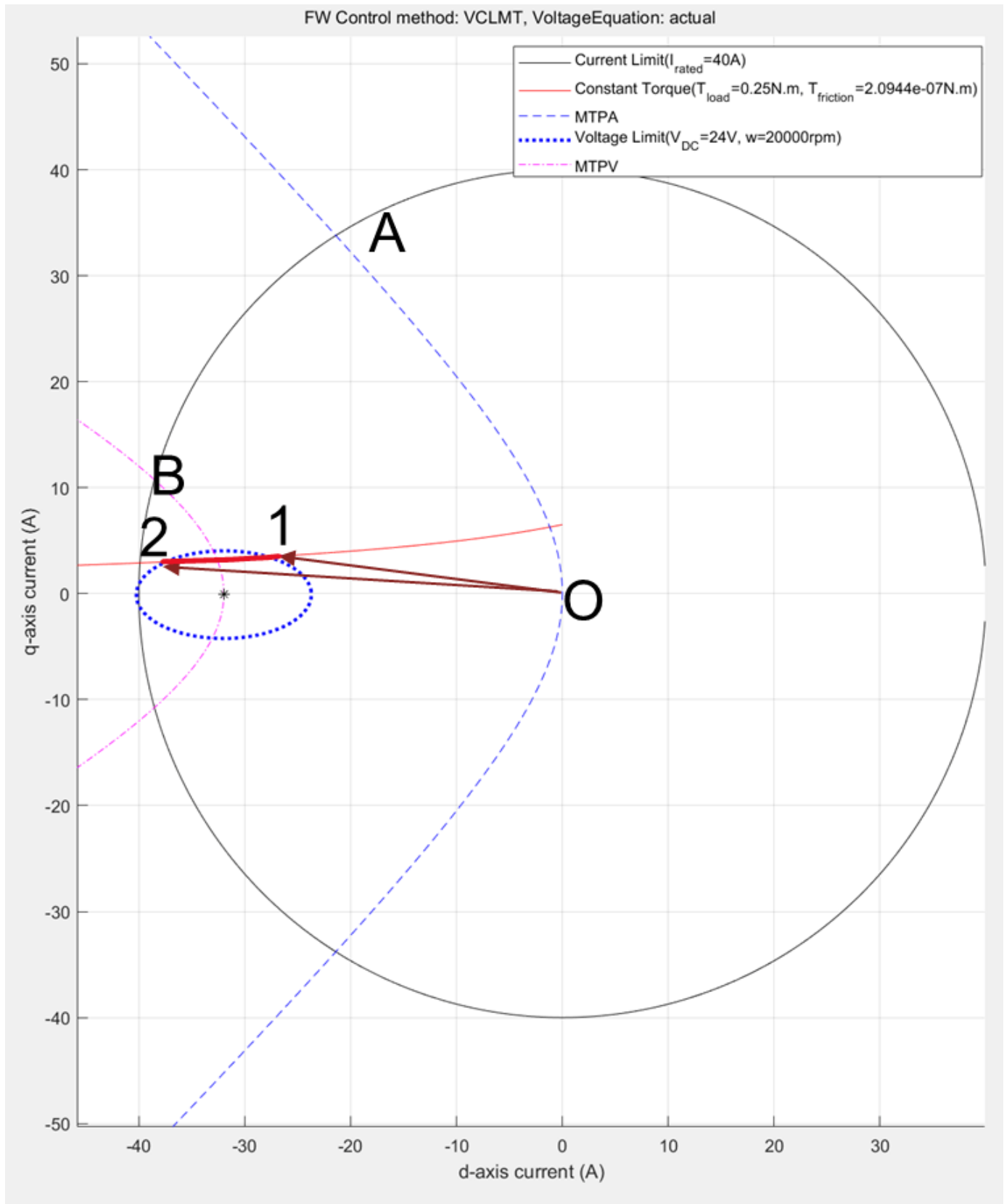
Figure 28: Constraint curves of a PMSM (designed to have MTPV region of operation) drawn at 0.25N.m load and 20000rpm speed, with the constant torque curve passing through the intersection of current limit circle and voltage limit curve.

You can design a motor to have a possible MTPV operating region, by selecting a weaker magnet or by having a high current limit, or a combination of both. For the motors which have a possible MTPV operating region, the constraint curves are drawn in the preceding figure. For these motors, theoretical speed limit is infinite, but the motors' practical maximum achievable speed is limited by the frictional load.

For these motors, field weakening operation starts after the MTPA region of operation. Beyond a specific speed of operation, which can be called the MTPV initiation speed, the voltage ellipse shrinks further. At higher speeds, it converges approximately at the critical current point. To satisfy the load torque, you can pick any point along the constant torque curve, which also satisfies the voltage constraint and current constraints. That is, a point which lies within the intersection region of the current limit curve and the voltage limit curve.

The figure 28 shows a case where the demanded load torque is lower than a torque that the motor can provide at the operating speed. For such operation, the points 1 and 2 are marked as the extremes of the possible points along the constant torque curve that satisfy the voltage and current constraints. The point 1 has the lowest current magnitude, making it the optimal point of operation.

id **-27.7788**
iq **3.48641**
Constant torque curve
20000 rpm
0.25 N.m. load

Figure 29: Zoomed-in portion of figure 28 showing the selected operating point.

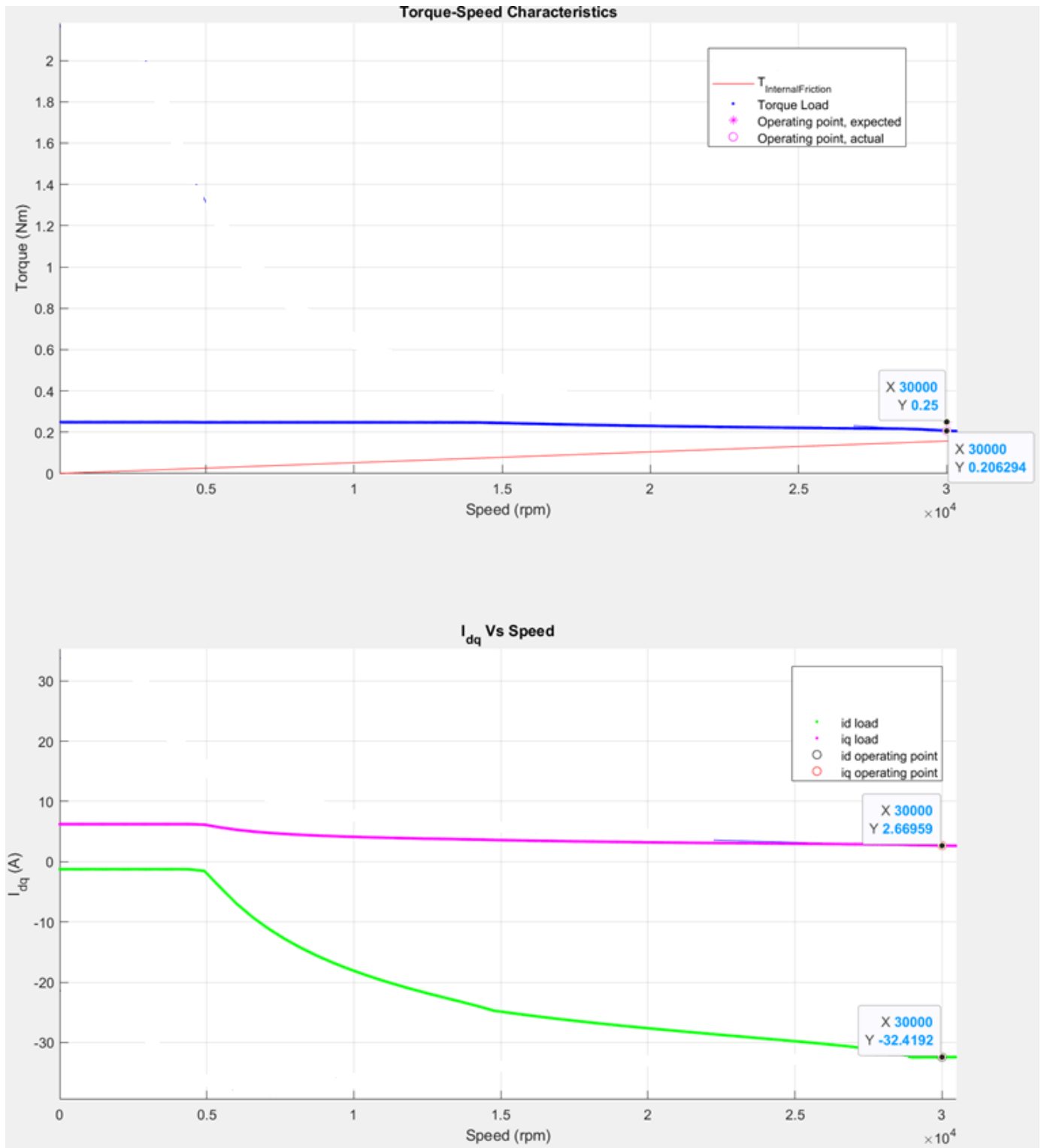**Torque-Speed Characteristics**

I$_{dq}$ Vs Speed

Figure 30 (a): Drive characteristics of a PMSM with the speed and torque point marked (b) Current curves of the PMSM for the loaded torque, with the operating point marked on them.

You can use the following code to plot the figures.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.I_rated=40;
pmsm.B=9e-5;
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
mcbPMSMCharacteristics(pmsm,inverter,torque=0.25,speed=20000,driveCharacteristics=0);
```

Case 5: MTPV, Load Torque More Than Maximum Possible Torque at Given Speed

FW Control method: VCLMT, VoltageEquation: actual

Legend:
- Current Limit(I_{rated}=40A)
- Constant Torque(T_{load}=0.25N.m, $T_{friction}$=3.1416e-07N.m)
- MTPA
- Voltage Limit(V_{DC}=24V, w=30000rpm)
- MTPV

Figure 31: Constraint curve of a PMSM (designed to have MTPV region of operation) drawn at 0.25N.m load and 30000rpm speed, with the constant torque curve not passing through the intersection of current limit circle and voltage limit curve.

Sometimes, the load torque can be higher than the torque that the motor can provide at the operating speed. In the preceding figure, this is represented by the constant torque curve not passing through the intersection of the current limit curve and voltage limit curve. Under this condition, the maximum torque is picked from the voltage limit curve. This will be the point along the MTPV curve, intersecting the voltage limit curve.

Figure 32: Zoomed in portion of the figure 31, showing the selected operating point for optimal operation.

Figure 33 (a): Drive characteristics of a PMSM with the speed and torque point marked (b) Current curves of the PMSM for the loaded torque, with the operating point marked on them.

The preceding figure shows the selected operating point with circles. In addition, the speed-torque plot is also depicted with the expected torque point, which is higher than the possible torque at that speed.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.I_rated=40;
pmsm.B=9e-5;
pmsm.Lq=pmsm.Ld*2;
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
mcbPMSMCharacteristics(pmsm,inverter,torque=0.25,speed=30000,driveCharacteristics=0);
```
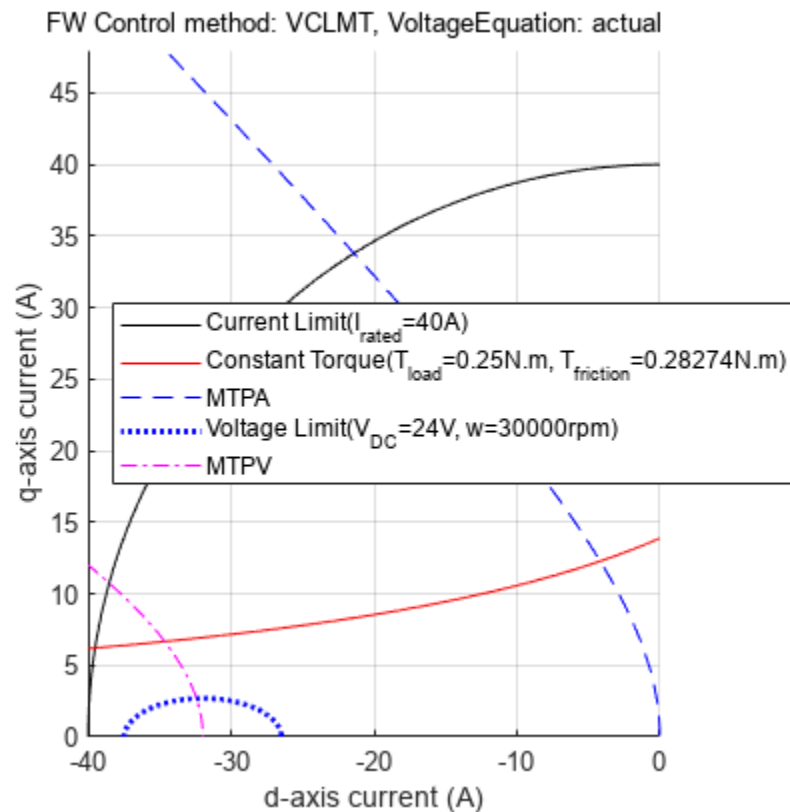

Identifying Possible (MTPA-Field Weakening-MTPV) Control Currents

The preceding section explains that for different speeds and torques, optimal (*id, iq*) values can be derived.

In a typical motor control application, you need a reference current value for the operating speed and the reference torque. Such reference currents can be derived using the preceding methods, which result in optimal operations.

LUT based PMSM Control Reference

The block "*LUT based PMSM Control Reference*" computes the *id* and *iq* look-up tables using the given motor parameters (either linear lumped parameters, or non-linear parameter maps) or uses the user-provided *id*, *iq* look-up table values.

IPMSM
Control Reference
(VCLMT)

T^{ref}
ω_m
I_d^{ref}
I_q^{ref}
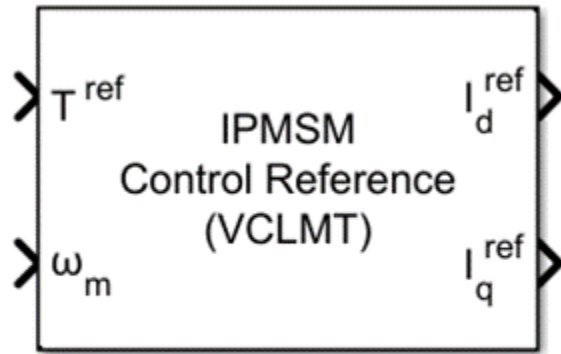
LUT based PMSM Control Reference

Motor parameters
from Datasheet

Motor parameters
from Parameter
Estimation

Motor parameters
from
measurements

Motor parameters
from FEA sim

Motor parameters
from dyno tests

Motor parameters
from vendor data

id, iq Look-up
Tables using
custom algorithms

Block Parameters: LUT based PMSM Control Reference ✕

LUT based PMSM Control Reference (mask) (link)

Generate LUT based PMSM reference currents for field oriented control of Surface &
Interior PMSMs.

The block accepts reference torque and feedback speed. The block outputs
corresponding d-axis & q-axis reference currents.

Enter the motor parameters and input/output signal units in the dialog box below.

| Motor parameters | Input units |

Motor topology Interior PMSM ⌄

Number of pole pairs: 4

Stator resistance per phase, Rs (Ohm): 0.36

Motor parameter input method Linear model with lumped parameters ⌄
Linear model with lumped parameters
Stator d-axis inductance, Ld (H) Non-linear model with Ld,Lq and FluxPM LUTs
Non-linear model with id and iq LUTs
Stator q-axis inductance, Lq (H): 0.4e-3   0.0004

Permanent magnet flux linkage, FluxPM (Wb): 6.4e-3   0.0064

Vdc input method Specify via dialog ⌄

DC bus voltage, Vdc (V): 24

Current Limit, I_limit (A): 7.1

Viscous damping coefficient, Bv (Nm-s/rad) 2.636875217824e-6   2.6369e-...

OK   Cancel   Help   Apply

Figure 34 (a): LUT based PMSM Control Reference block, shipped as a part of MCB library. Figure 34 (b): The mask view for the block with options showing the input possibilities of motor parameters.

The LUT based PMSM Control reference block accepts multiple types of inputs that can utilize the externally generated (*id, iq*) LUTs or calculate the LUT based on the provided motor parameters (linear or non-linear). When the user selects the motor parameters (either linear or non-linear) as parameter input method, the LUT based PMSM Control Reference block computes the LUTs on the host before the start of simulation or before code-generation step.

mcbGenerateTables

The function *mcbGenerateTables* ships with Motor Control Blockset™. You can use it to create *id*, *iq* LUTs on the host for custom grid sizes and grid point densities. The LUT based PMSM Control Reference block uses this function to compute the *id*, *iq* LUTs using the given motor parameters.

When the *id, iq* LUTs are generated, you can plot them in the *d-q* current space for multiple speed and torque values. When the point grid overlaps with the motor's constraint curves, you can obtain figure 36. In the figure, the dots at the intersection of the curves are only for representation purposes. The output figure window of the function will have segments of voltage limit curve and constant torque curves that are drawn within the current limit circle.

FW Control method: VCLMT, VoltageEquation: actual
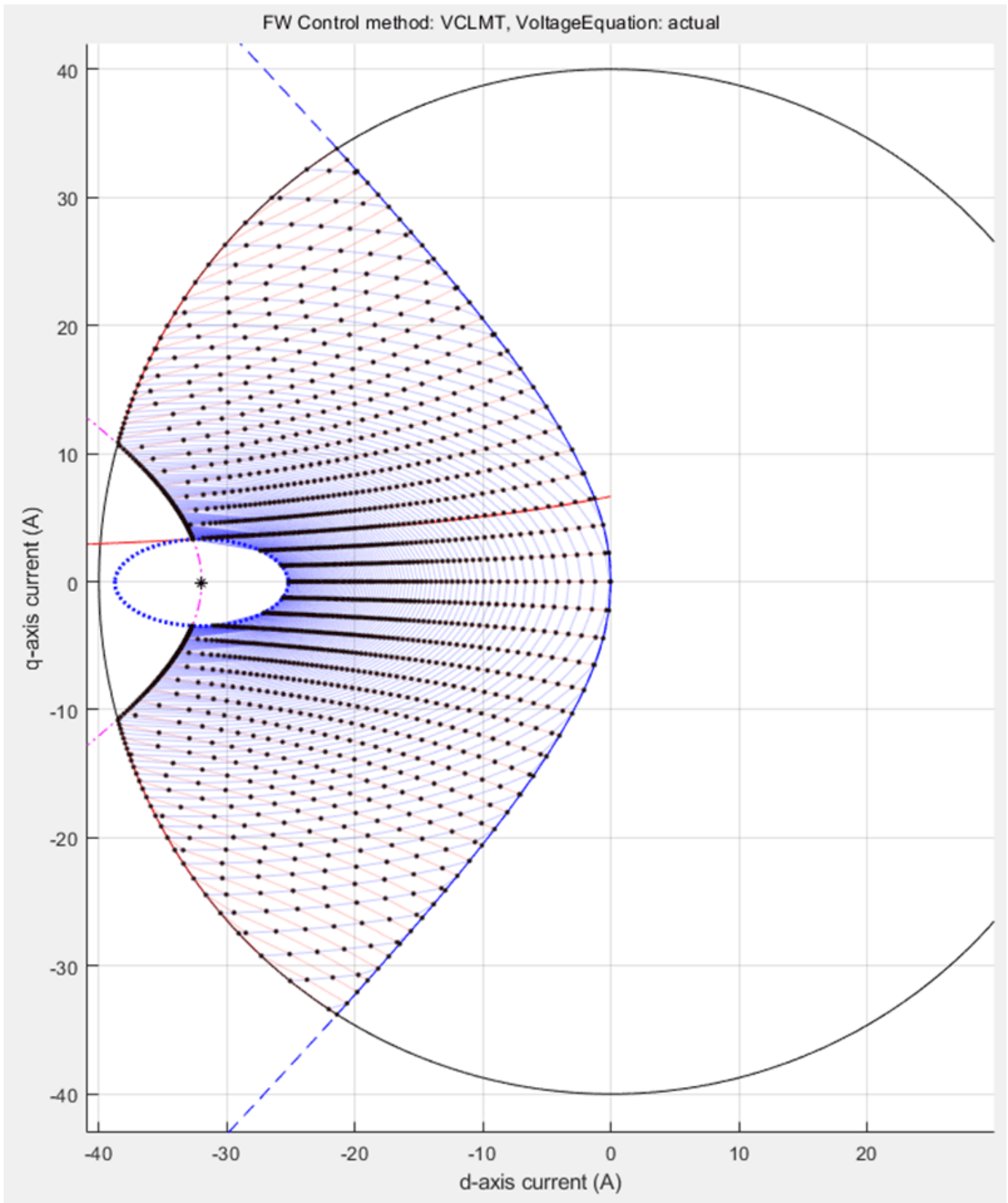
id
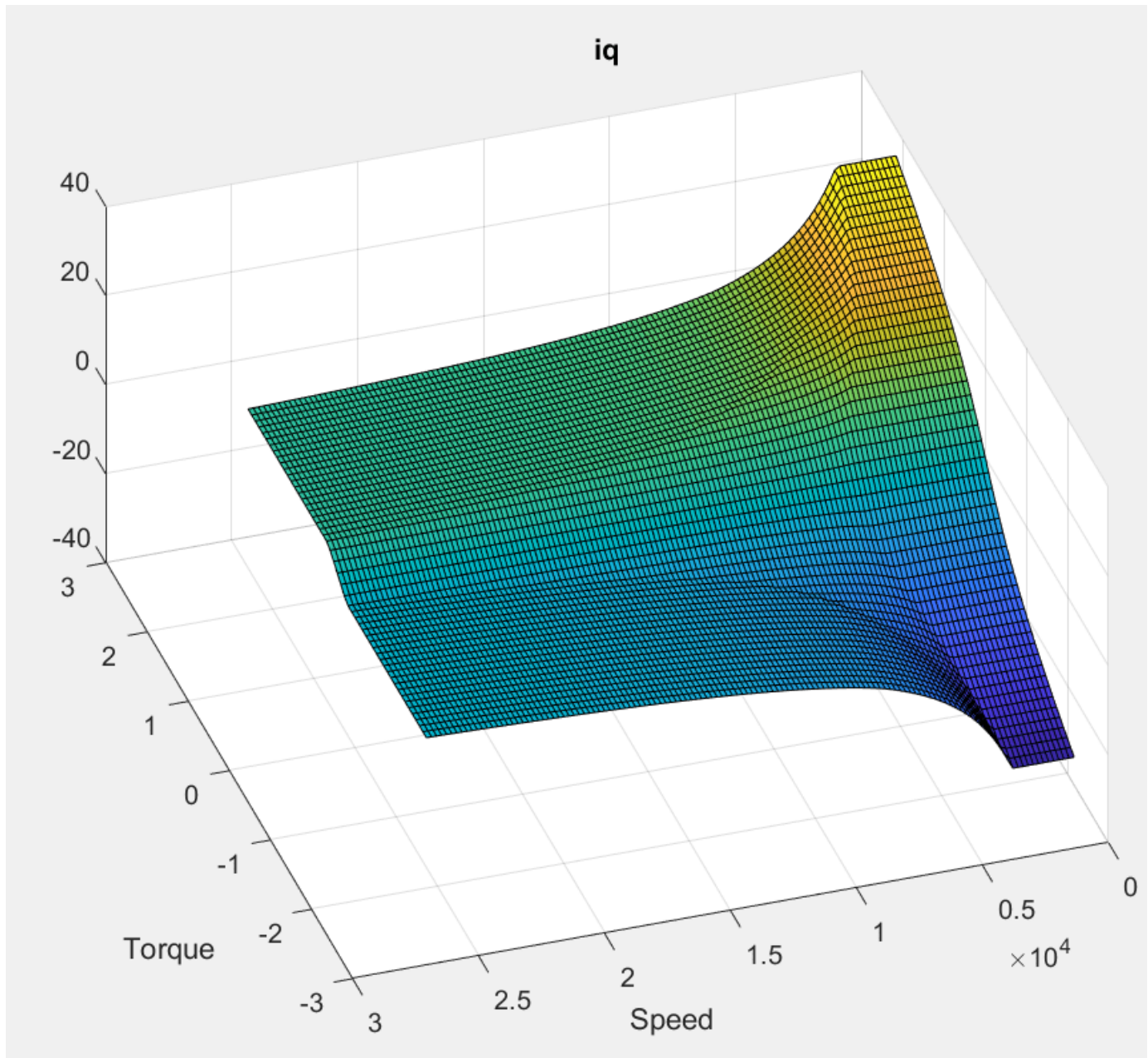
Figure 35 (a): Constraint curves of a PMSM (designed to have MTPV operating region) overlapped with the grid of (id, iq) Look-up Tables for different speeds and load torques. Figure 35 (b): id LUT. Figure 35 (c): iq LUT.

You can use the following code to create and plot a grid of the *id, iq* with respect to speed and torque values.

```
pmsm=mcb_SetPMSMMotorParameters('Teknic2310P');
pmsm.Rs=0.01;
pmsm.I_rated=40;
pmsm.B=9e-5;
pmsm.Lq=pmsm.Ld*2;
```

```
inverter=mcb_SetInverterParameters('BoostXL-DRV8305');
seed.drawLUTonConstraintCurves=1;
seed.TBreakPointsCount=31;
seed.wBreakPointsCount=32;
PMSMLUT=mcbGenerateTables(pmsm,inverter,'idiqLUTs',seed);
```

FW Control method: VCLMT, VoltageEquation: actual
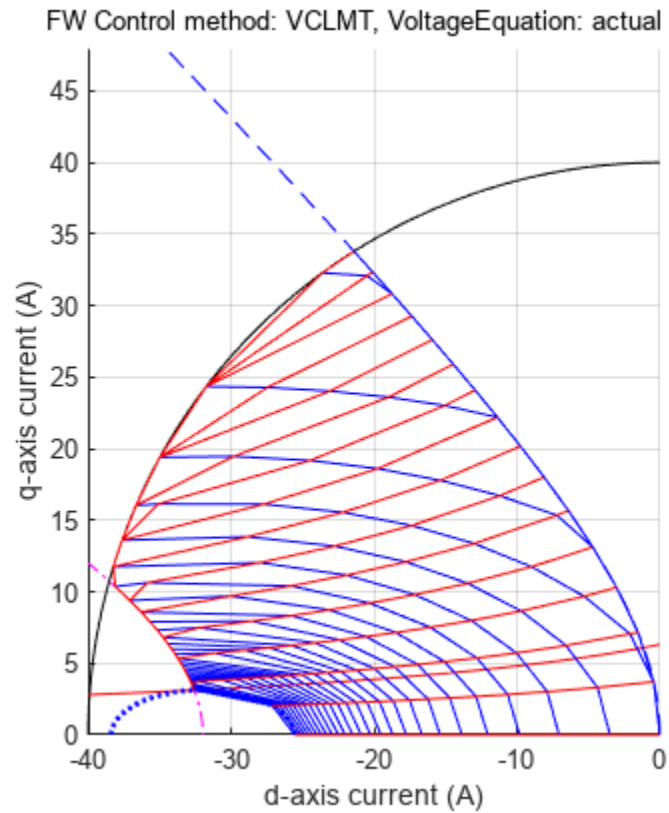

```
disp(PMSMLUT)

    drawLUTonConstraintCurves: 1
            TBreakPointsCount: 31
            wBreakPointsCount: 32
                      trefVec: [-2.1646 -2.0203 -1.8760 -1.7317 -1.5874 -1.4431 -1.2988 -1.1545
                      wrpmVec: [1 830 1660 2489 3318 4147 4977 5806 6635 7464 8294 9123 9952 1078
                      idTable: [31x32 double]
                      iqTable: [31x32 double]
```

Simulation Using id, iq LUTs
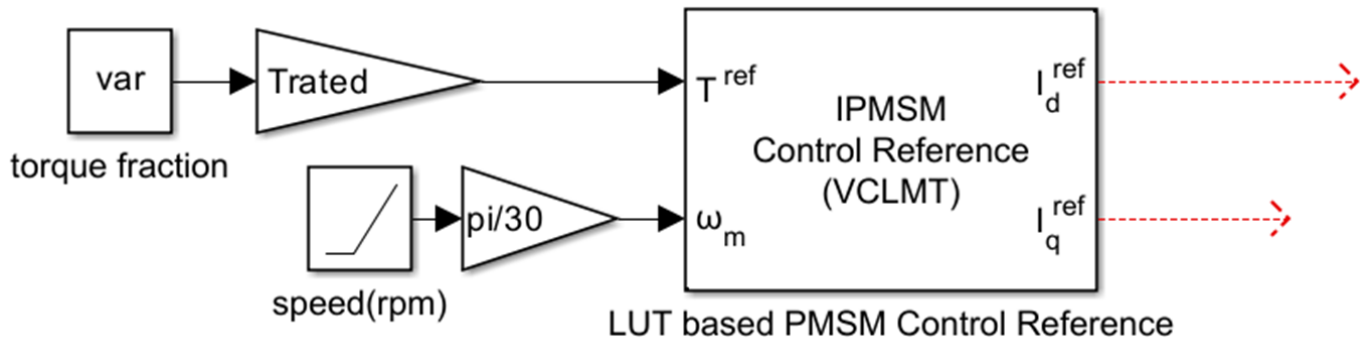
Figure 36: View of Simulink model that is created to use the LUT based PMSM Control Reference block to generate [id, iq] reference values for a given load torque and a range of speed values. This model has been used to create the (id, iq) reference values for fractional rated torque values for different speeds drawn in the figures 21, 24, 27, 30, and 33.

As you can see, the LUT based PMSM Control Reference block can be used with a given load torque and a range of speed values to derive the possible speed-torque characteristic curve, similar to the ones shown in the figures in the earlier sections.

The following figures are constraint curves of the motor that overlaps with the data from the simulations using the LUT based PMSM Control Reference block.

FW Control method: VCLMT, VoltageEquation: actual

FW Control method: VCLMT, VoltageEquation: actual
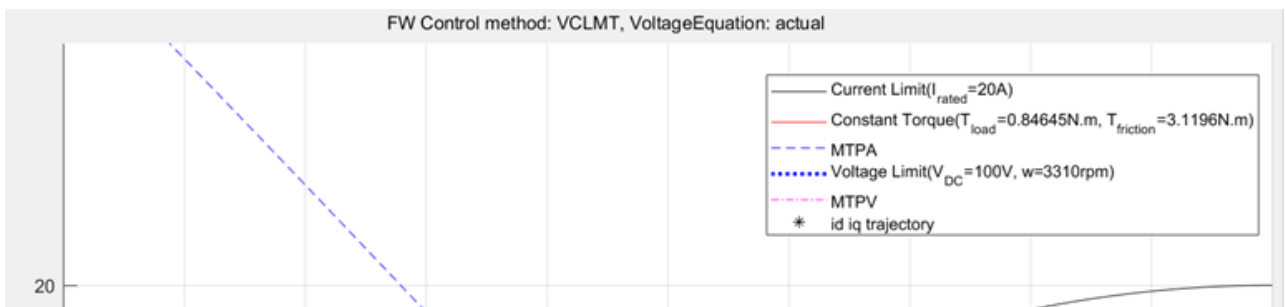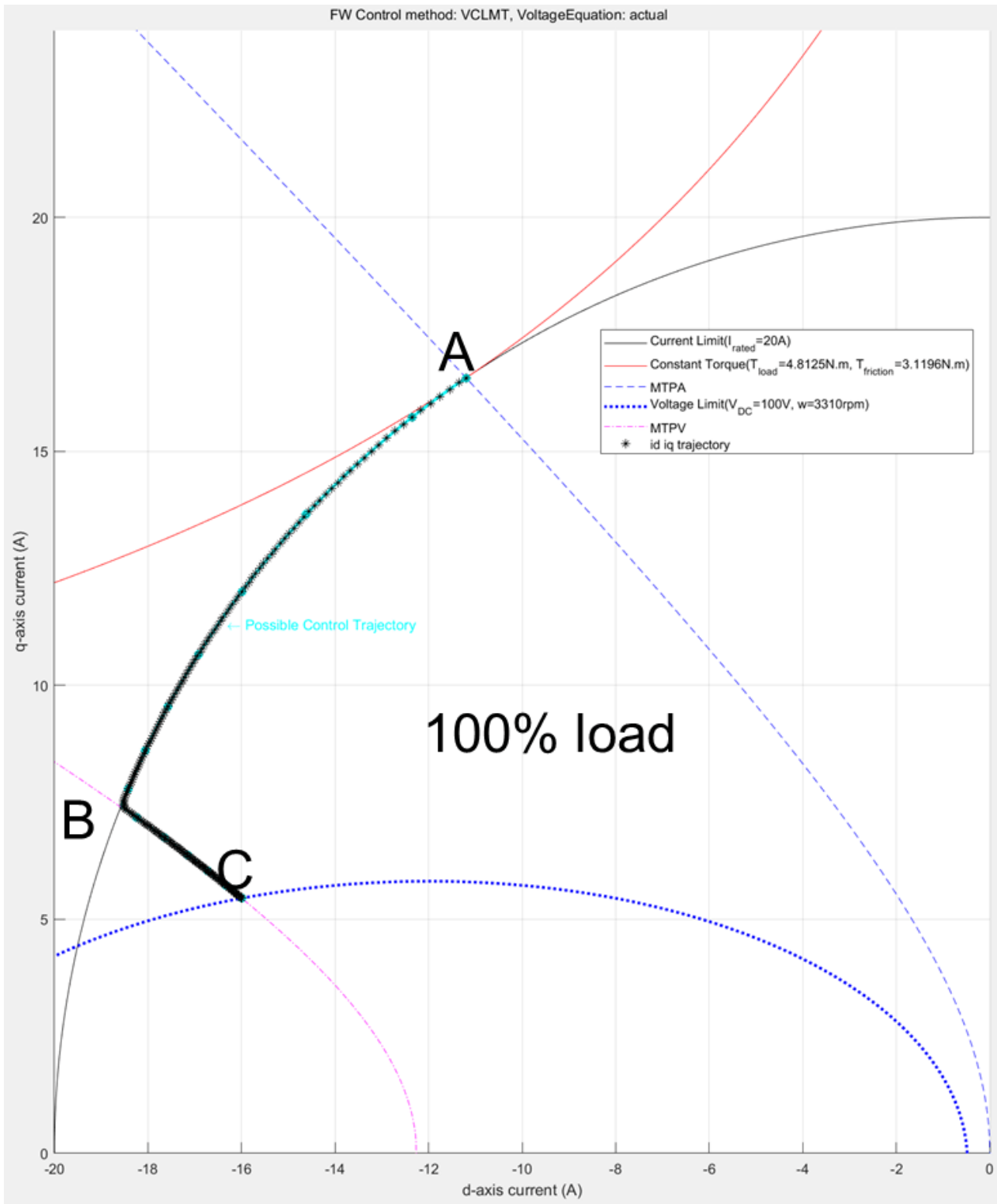
Figure 37 (a): Constraint curves of a PMSM drawn at envelope torque, showing the overlapped (id, iq) reference values generated from the LUT based PMSM Control Reference block.

Figure 37 (b): Constraint curves of a PMSM drawn at half the rated torque, showing the overlapped [id, iq] reference values generated from the LUT based PMSM Control Reference block.

From the figures 37(a) and 37(b), notice that the black stars are the contour of points derived from the simulation that overlaps the corresponding constraint curves. For both these cases, the MTPA region is not shown, because the torque demand starts from the maximum, and does not increase. For the case of 100% load, you can see that the operating points follow the field weakening path along the current constraint curve and then the MTPV curve. For the case of 50%, you can see that the operating points follow the constant torque curve and then the MTPV curve.

An example *mcb_pmsm_nonlin_fwc.slx* uses the non-linear data from FEA simulation software to demonstrate usage of non-linear motor parameters to derive the *id, iq* LUTs and derives the reference currents based on the operating conditions.
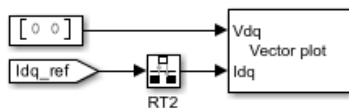
Figure 38: The example model *mcb_pmsm_nonlin_fwc.slx*.

See Also

https://www.mathworks.com/help/mcb/ug/pmsm-characteristics-constraint-curves.html

https://www.mathworks.com/help/mcb/ref/lutbasedpmsmcontrolreference.html

https://www.mathworks.com/help/mcb/gs/fwc-with-mtpa-of-non-linear-pmsm-using-lut.html

https://www.mathworks.com/help/sps/ug/ipmsm-torque-control-in-an-axle-drive-ev.html

Estimate Motor Parameters Using Motor Control Blockset Parameter Estimation Tool

Estimate Motor Parameters Using Motor Control Blockset Parameter Estimation Tool

Motor Control Blockset provides a parameter estimation tool that estimates the motor parameters accurately. You can use the estimated motor parameters to simulate the motor model and design the control system. Therefore, the simulation response of the motor model (configured with the estimated parameters) is close to the behavior of the actual motor under test.

The parameter estimation tool can determine the parameters for both permanent magnet synchronous motors (PMSM) and induction motors. You can use these workflows to use the parameter estimation tool according to the type of motor and motor-control hardware that you want to use:

- "Estimate PMSM Parameters Using Recommended Hardware" on page 4-186
- "Estimate PMSM Parameters Using Custom Hardware" on page 4-209
- "Estimate PMSM Parameters Using Parameter Estimation Blocks" on page 4-317
- "Estimate PMSM Parameters Using Parameter Estimation Blocks on Real-Time Systems" on page 4-324
- "Estimate Induction Motor Parameters Using Recommended Hardware" on page 4-202
- "Estimate Induction Motor Parameters Using Parameter Estimation Blocks" on page 4-310

See Also
Interior PMSM | Surface Mount PMSM | Induction Motor

Concepts

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.
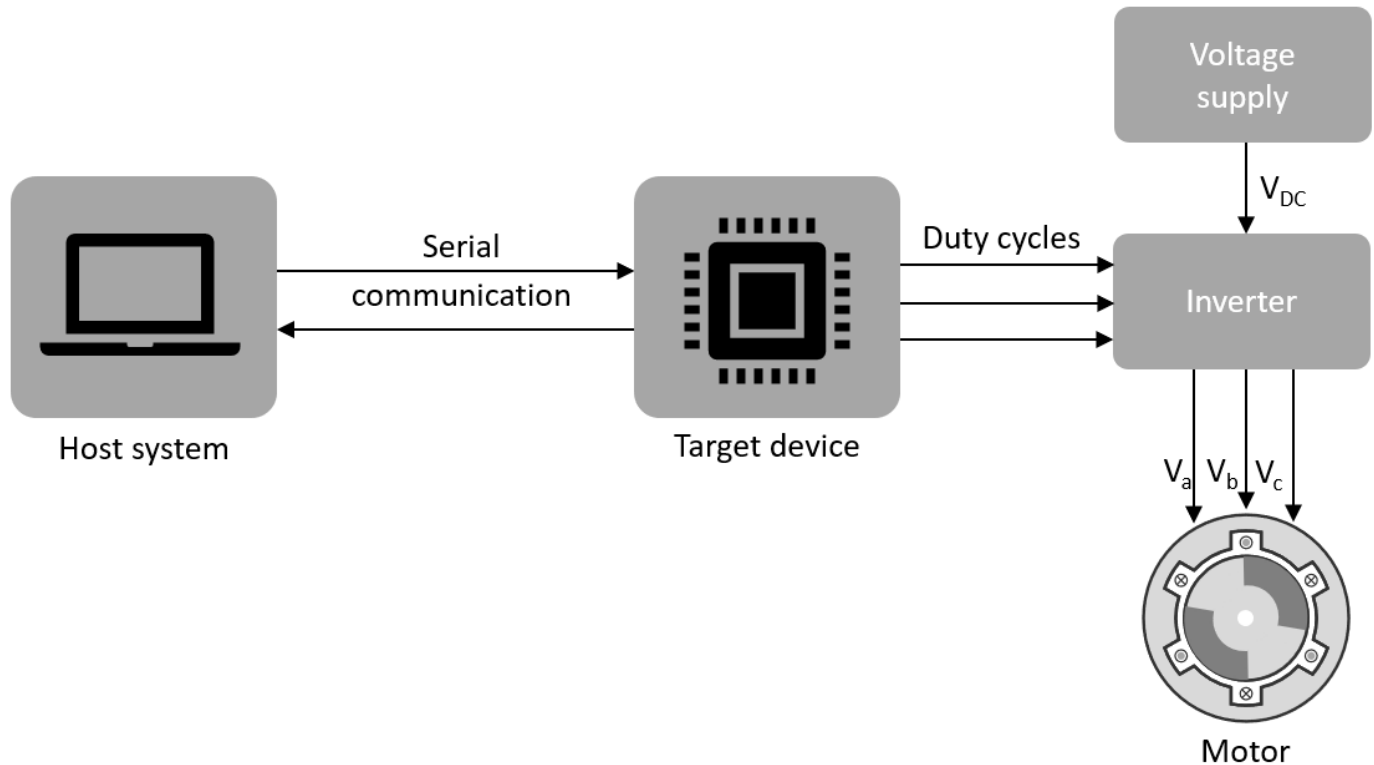
Host Model

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.

For example, consider a situation where a model needs to sample two signals A and B every 50 μs and send them to the host model for monitoring and debugging. To fulfil this requirement, the Motor Control Blockset examples divide the entire signal data into packets of 600 data points. Therefore, a packet from signal A combined with a signal B packet results in 1200 data points. Using this approach, the target hardware sequentially sends a pair of data packets (from signals A and B) to the host model. The target further groups these packet pairs into sections. Each section begins with a header and ends with a terminator. Following a header, the host model starts buffering the data points until it receives a terminator, after which Simulink reads the buffered data that you can monitor.

To read the buffered data we select **Enable blocking mode** and set **Data size** to [2 n] and **Sample time** to n*50 μs in the Host Serial Receive block parameter dialog box. Using this configuration, the Host Serial Receive block reads 2×n data points every n×50 μs. We select a value for *n* such that the Simulink host model can run efficiently in real time.
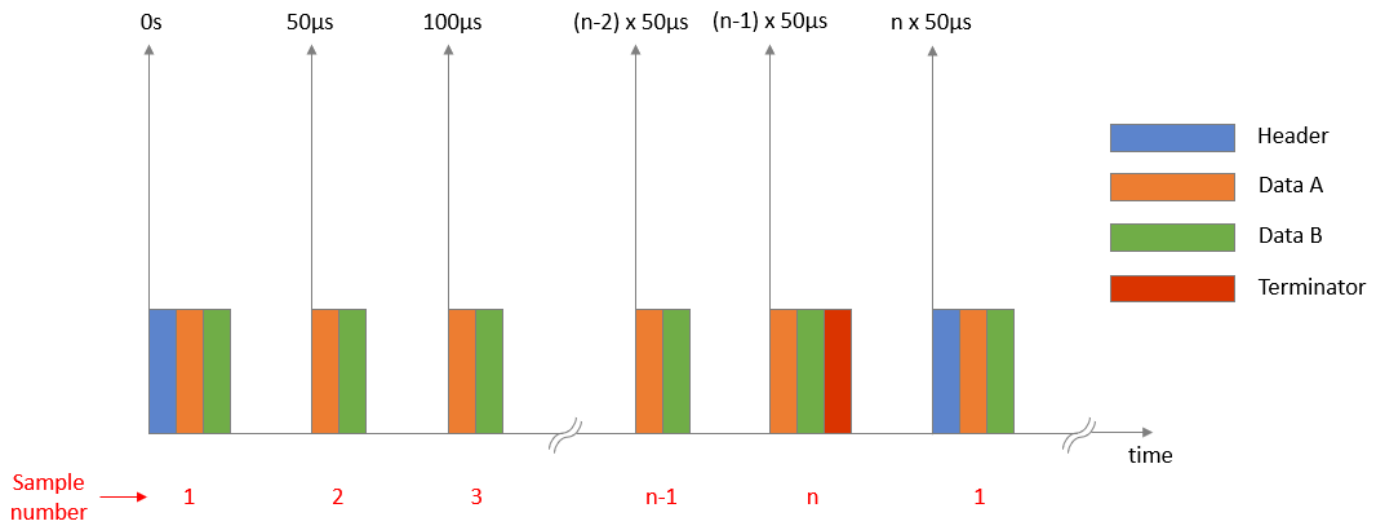
Motor Control Blockset examples follow this approach because Simulink shows high efficiency when processing big packets of data at low data speeds and the target hardware (used by Motor Control Blockset) efficiently processes smaller data packets at higher data speeds.

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.
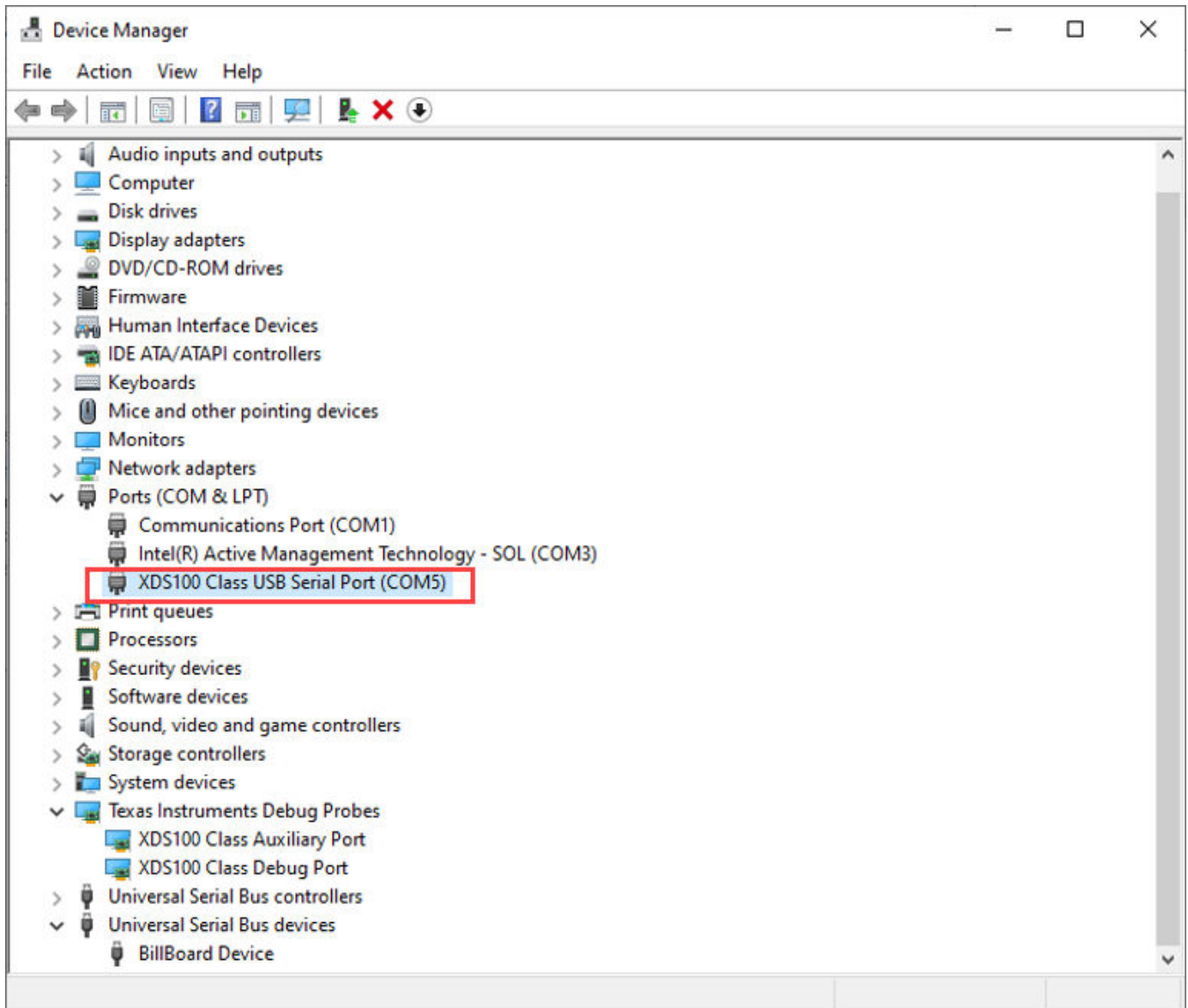
If you face difficulty in finding the COM port, follow these steps to determine the COM port:

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).
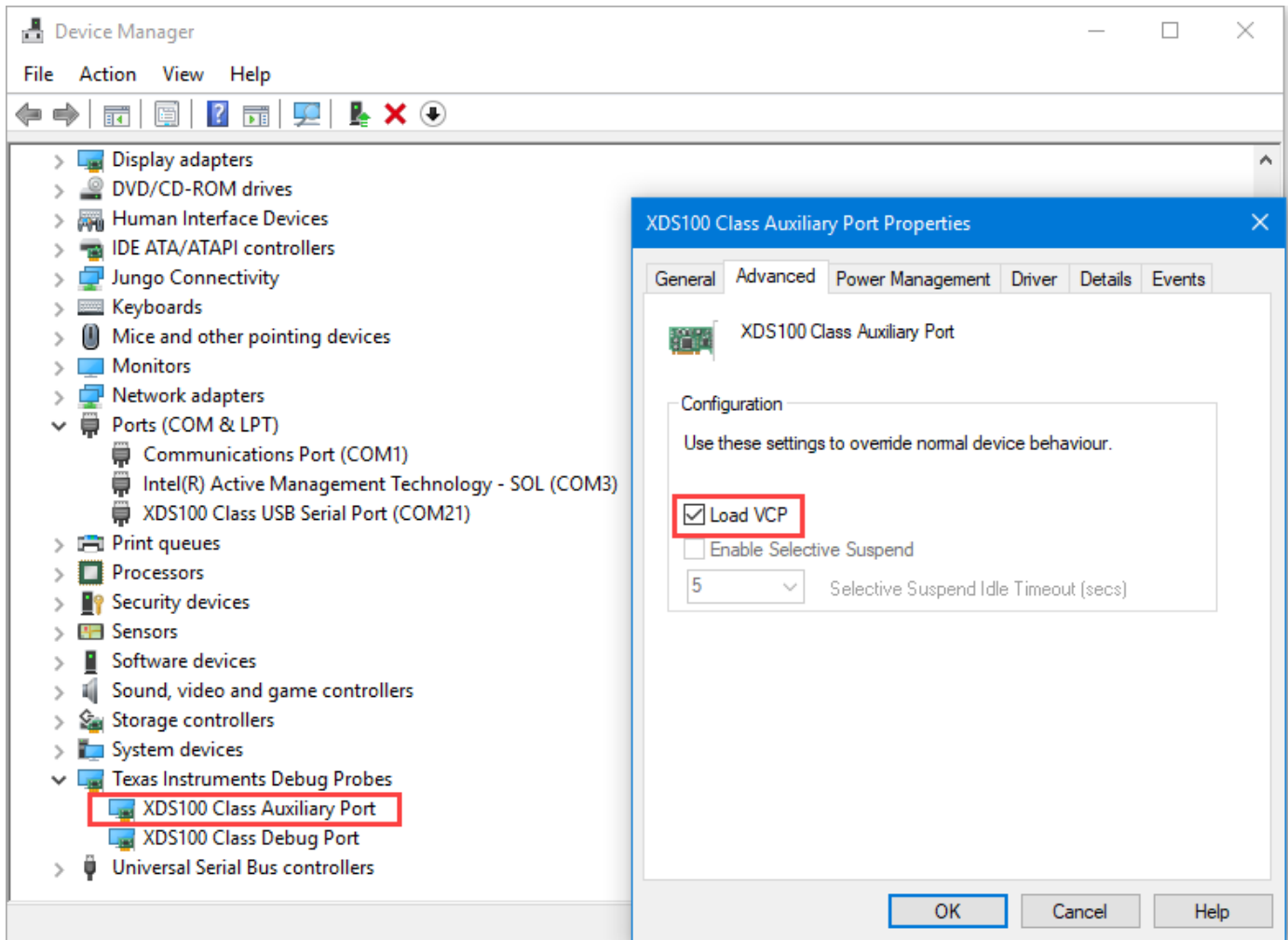
5 If **Texas Instruments Debug Probes** do not appear in the Device Manager, expand **Universal Serial Bus controllers** in the Device Manager and follow these steps:

    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.

Add Debug Signals from Target Hardware

The host models included in the Motor Control Blockset examples provide a list of signals in the **Debug signals** section. You can select these signals and monitor them using the time scope available on the host model for debugging purposes.
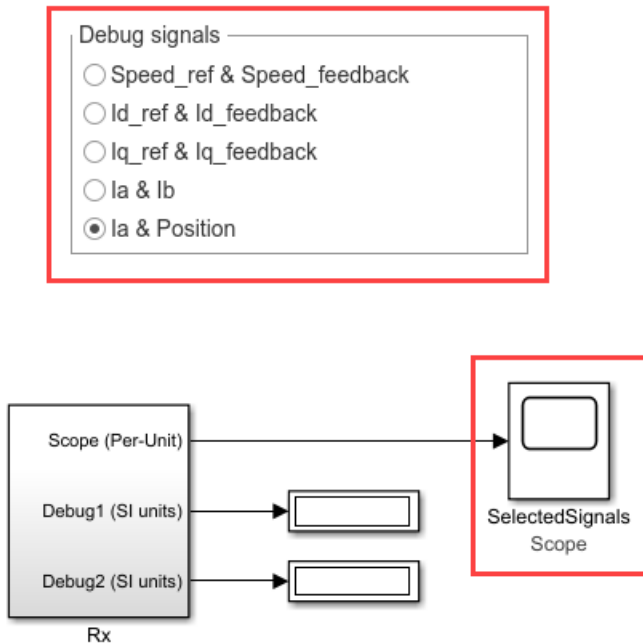
You can only add a pair of debug signals to this section at one time. Alternatively, you can modify the existing item in the list (for example, **Speed_ref & Speed_feedback**) to show the signals that you want. However, this procedure explains how to add a new pair of debug signals to the **Debug signals** section.

1 Double-click the Debug signals radio button to open the block parameters dialog box. Add a new state value (for example, `6 - Id_ref_PU & Iq_ref_PU`) to the existing list.
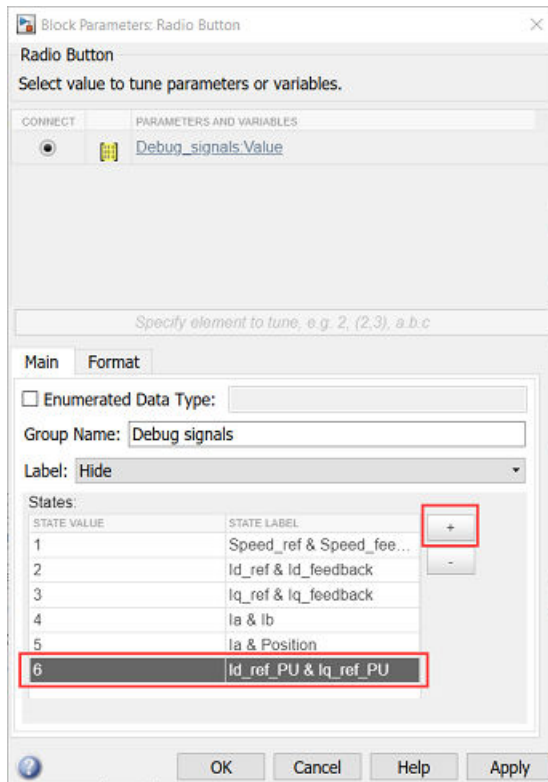
2   Open the block parameters dialog box of the **Debug_signals** constant block available in the **TX** subsystem of the host model. Set the constant to the new state value that you added in step 1 (for example, change the value 5 to 6).
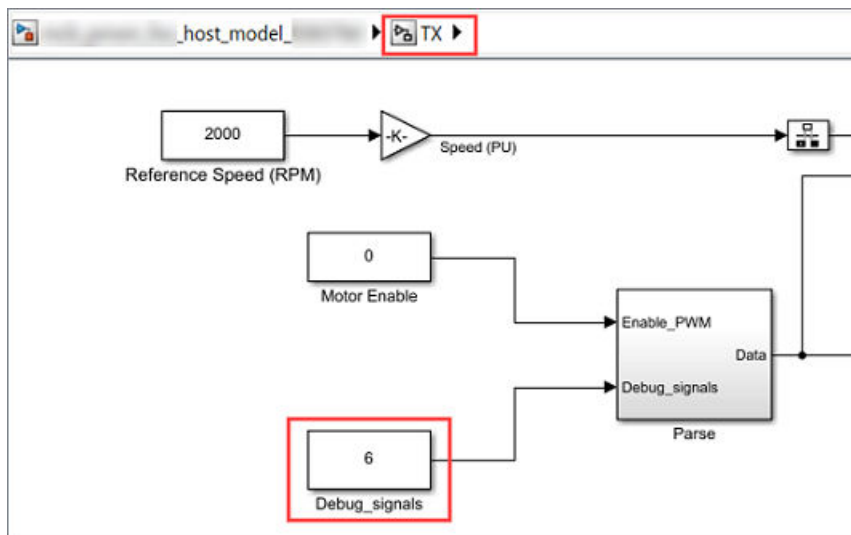
3   Open the target model associated with the host model and open the `Current Control/Debug_signals` subsystem.

4   Add two more inputs to the mux block highlighted in the following figure:
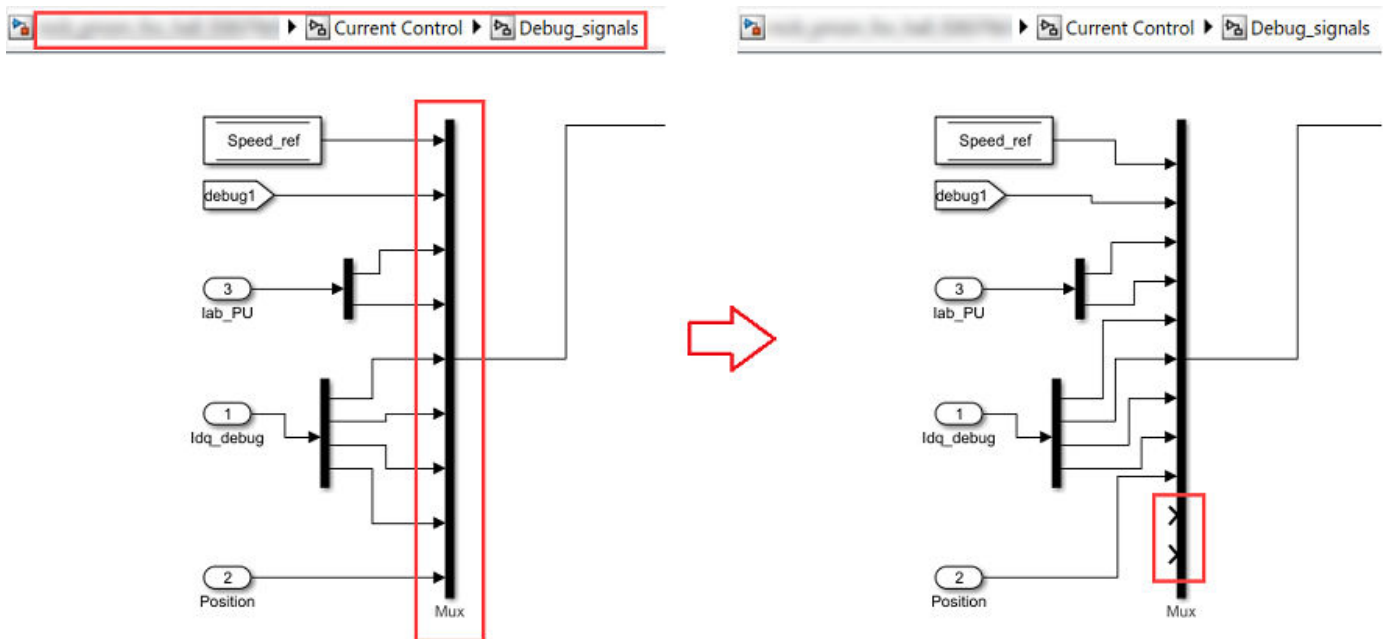
5 Open the block parameters dialog box of the Selector block and set the **Input port size** parameter value to the number of inputs now available in the mux (for example, change the value from 9 to 11).
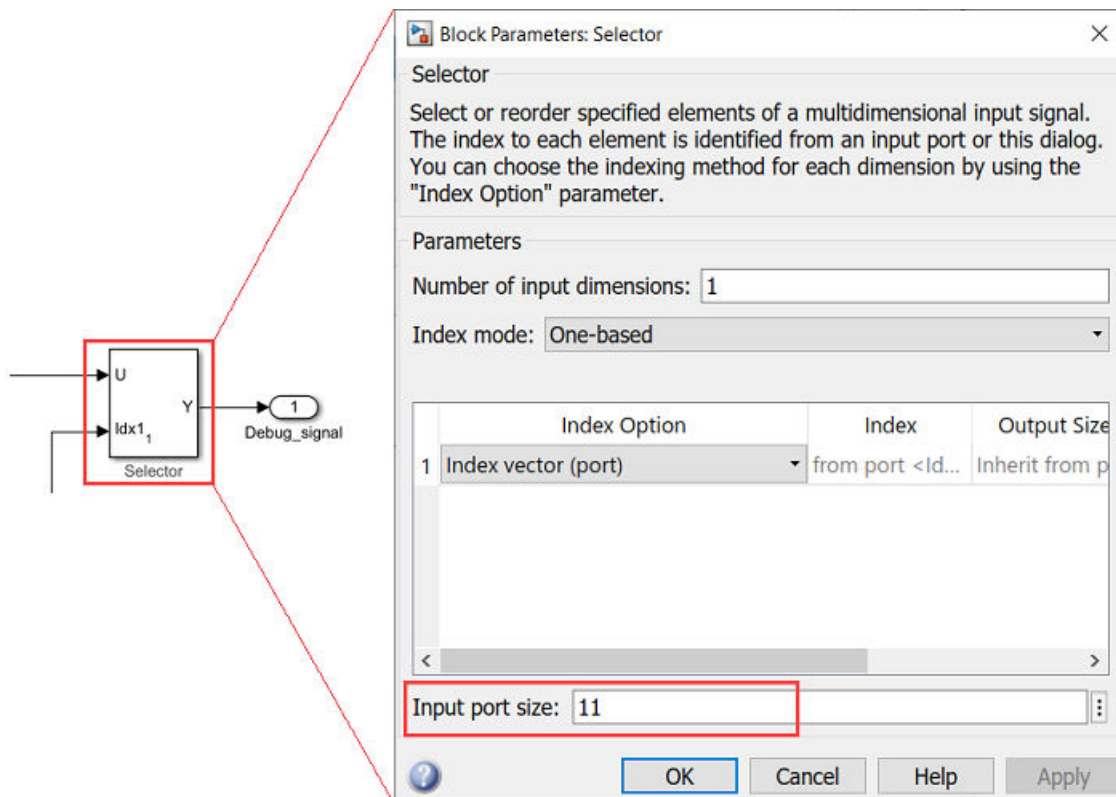
6 Connect the new signals (for example, *Id_ref_PU* and *Iq_ref_PU*) to the new mux ports that you added in step 4.
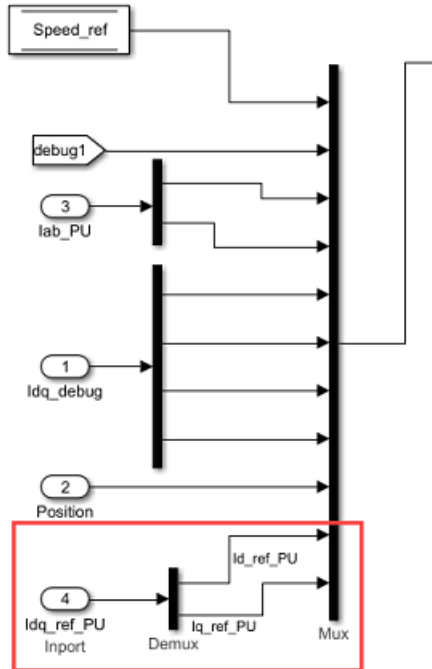
7 Open the block parameter dialog box of the Multiport Switch block and set the **Number of data ports** parameter to the number of debug signal pairs now available in the host model (for example, change the value from 5 to 6).
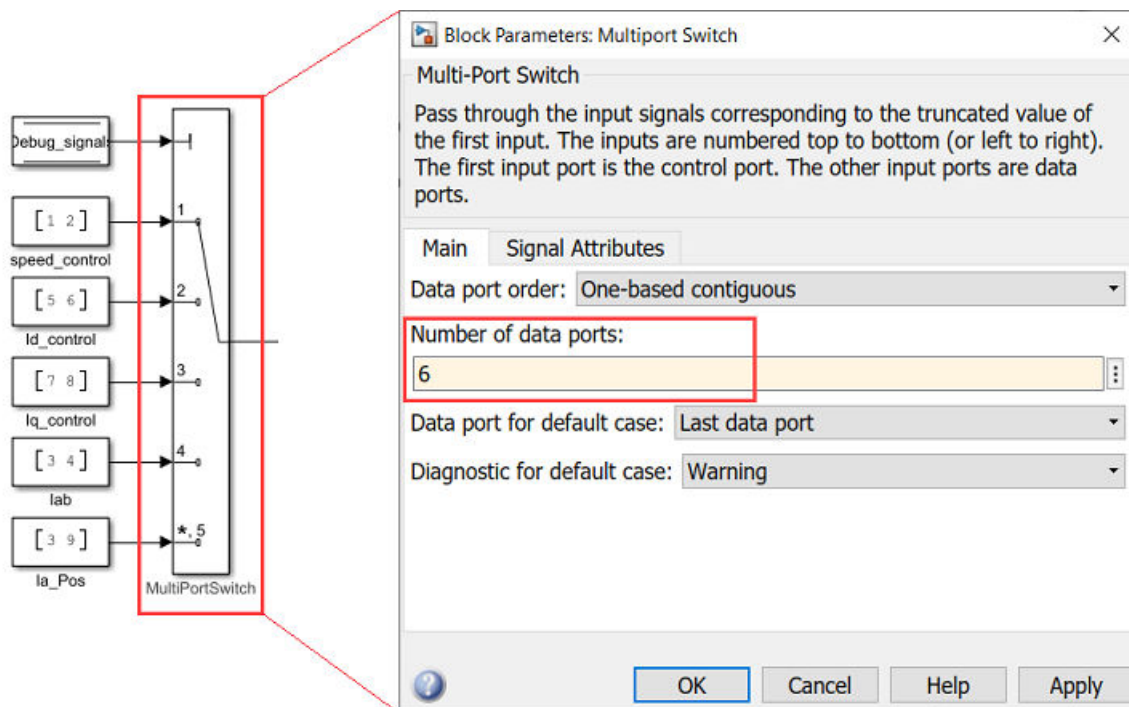
8 Add a constant block having a vector value that indicates the new signal positions on the mux (for example, use the vector `[10, 11]` for the mux inputs that you added in step 4). Connect this constant block to the newly added port on the Multiport Switch block.
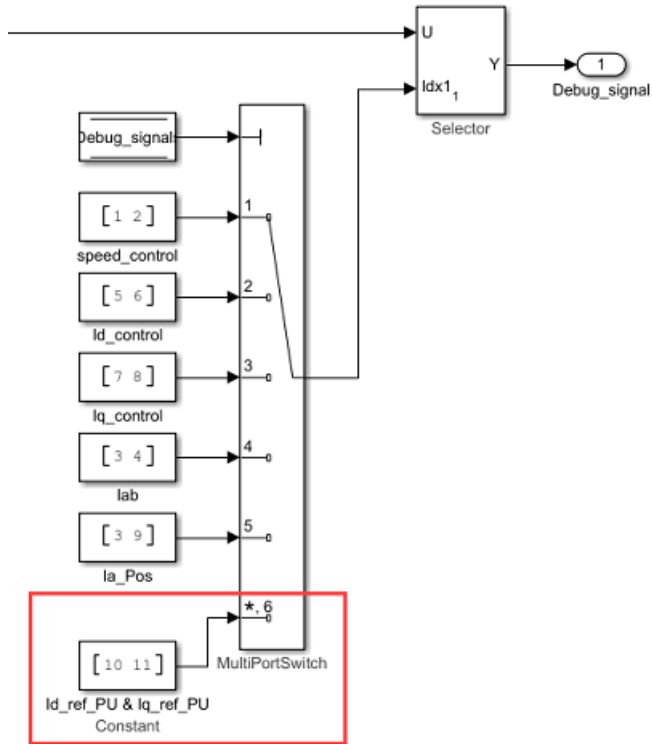
Open-Loop and Closed-Loop Control

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.
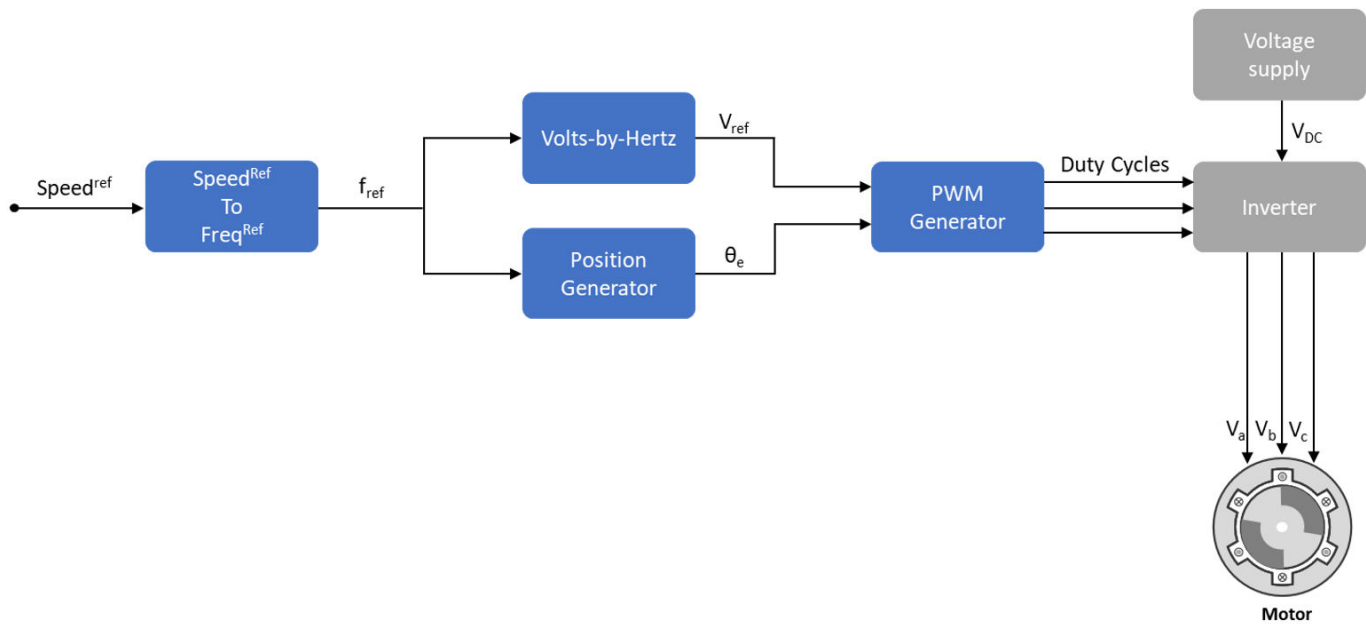
This figure shows an open-loop control system. The power circuit consists of a PWM voltage fed inverter supplied by a DC source. The system does not use any feedback signal for control implementation. It uses the reference speed to determine the frequency of the stator voltages. The system computes the 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.

$$\lambda_m \propto \frac{V_s}{f_s}$$

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.
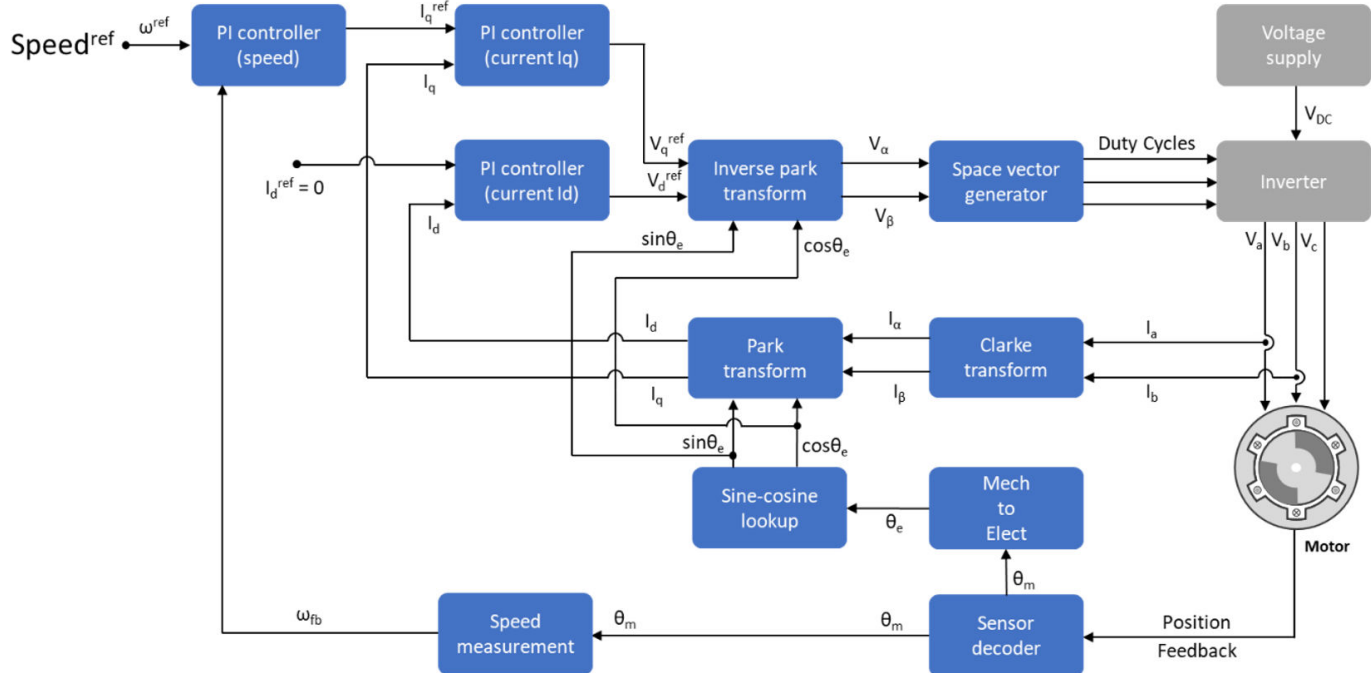
Field-Oriented Control (FOC) (or vector control) is a popular closed-loop system that is used in motor control applications. The FOC technique is used to implement closed-loop torque, speed, and position control of motors. This technique also provides good control capability over the full torque and 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 4-10 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 4-79 and "Hall Offset Calibration for PMSM Motor" on page 4-69 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 6-13 for more details.
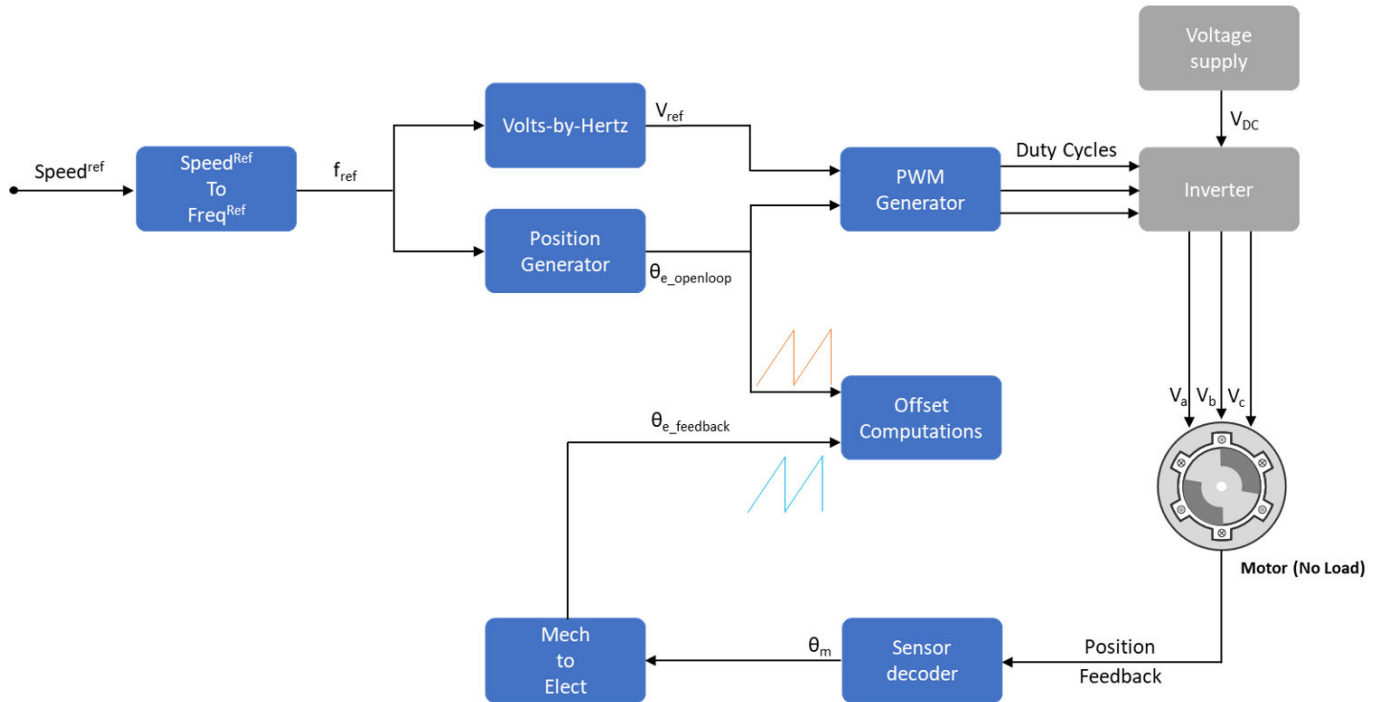
By using this algorithm, the offset calibration examples detect the position offset in this manner:

- 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.
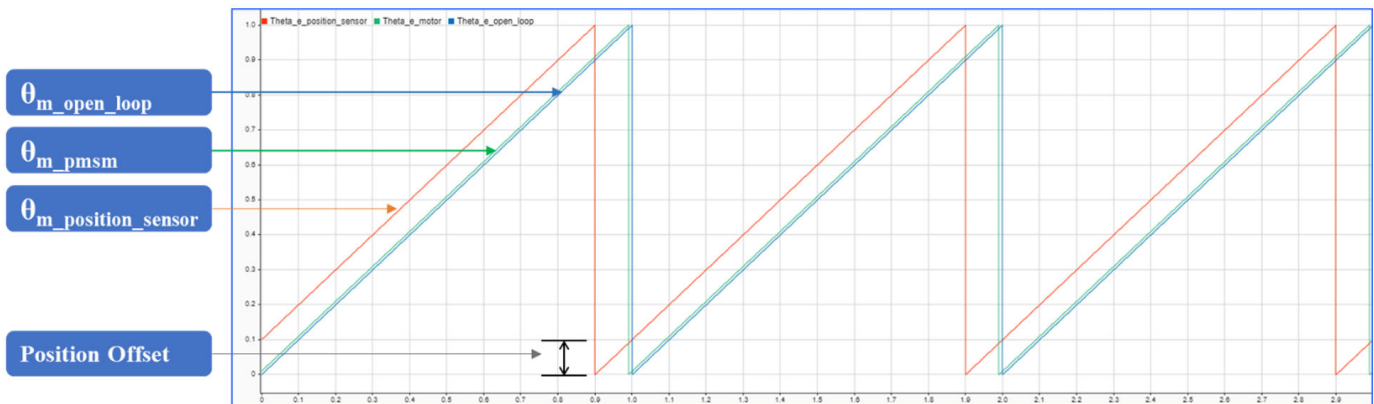
This figure shows the comparison of open-loop position from the control algorithm along with the actual position of the motor. The figure also shows the feedback from the position sensor. The position offset, which is the difference between the open-loop position and feedback position from the sensor, 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 4-69 example outputs the electrical position offset. Whereas, the "Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79 example outputs the mechanical position offset.

For steps to compute the offsets, see these examples:

"Quadrature Encoder Offset Calibration for PMSM Motor" on page 4-79

"Hall Offset Calibration for PMSM Motor" on page 4-69

"Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset" on page 4-10

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,

quantity expressed in SI units = quantity expressed in PU × 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 = . $\left(\dfrac{\text{inverter.V\_dc}}{\sqrt{3}}\right)$ For Sinusoidal PWM, it is PU_System.V_base = . $\left(\dfrac{\text{inverter.V\_dc}}{2}\right)$
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 $I_{SenseMax}$ of the inverter. PU_System.I_base = inverter.ISenseMax
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 torque is mathematically derived from the base current. Physically, the motor may or may not be able to produce this torque. Generally, it is PU_System.T_base = $\dfrac{3}{2}$. × pmsm.p × pmsm.FluxPM × 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 $$\text{PU\_System.P\_base} = \frac{3}{2}.$$ $\times$ 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 that saves the motor variables.
- *inverter* is the MATLAB workspace parameter structure that saves the inverter variables.
- *PU_System* is the MATLAB workspace parameter structure that saves the 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.

Program Control Flow of Motor Control Blockset Examples

This section describes the control flow of the field-oriented control (FOC) algorithm that the Motor Control Blockset examples use. The control flow operates using the hardware events along with the triggered software and hardware interrupts. This figure describes the interactions between the hardware modules and software subsystems.
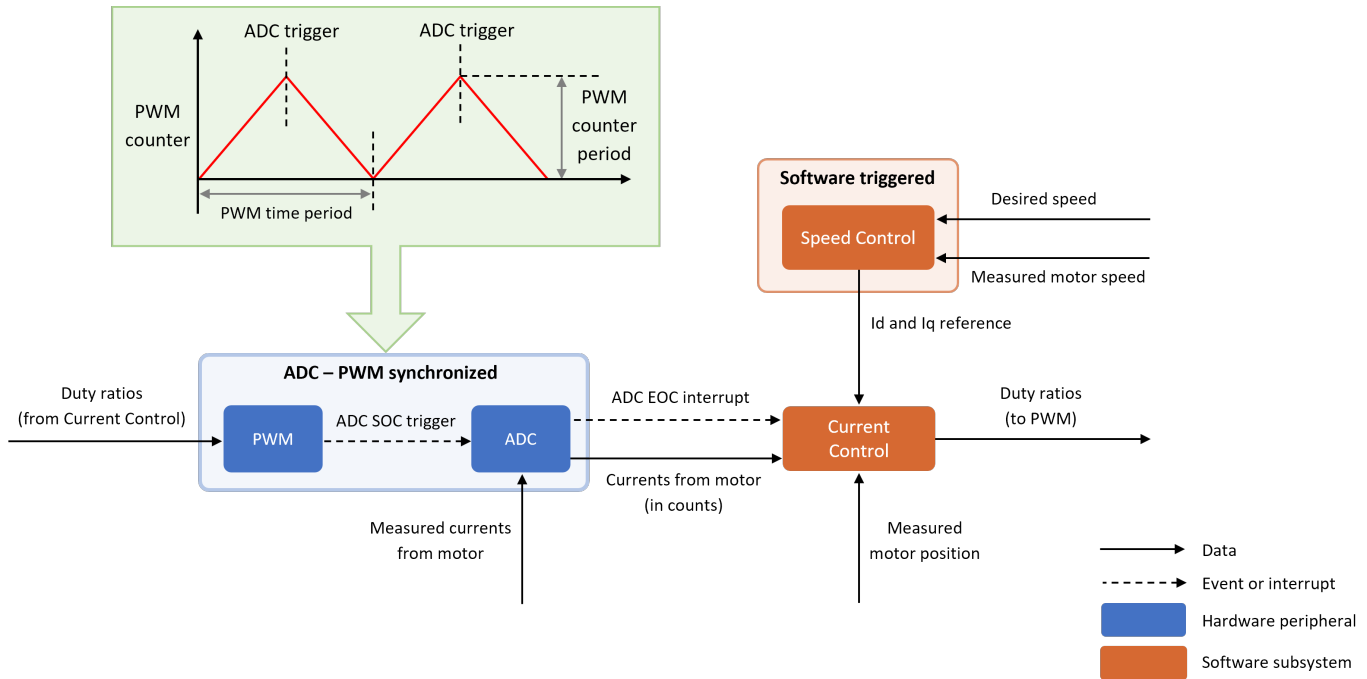
The FOC based examples generally use the data speed of 20KHz for the **Current Control** triggered software subsystem. Similarly, the examples use the data speed of 2KHz for the **Speed Control** triggered software subsystem. The ADC end of conversion (EOC) interrupt (a hardware interrupt) triggers the current control subsystem. The PWM-ADC synchronization controls the rate of this trigger. Similarly, a software interrupt triggers the speed control subsystem.

The entire system uses external inputs for motor position, motor speed, motor currents, and the desired motor speed. The preceding figure shows the interactions between these data points and the enclosed subsystems.

Motor Control Blockset provides the "Algorithm-Export Workflows for Custom Hardware" on page 4-184 example that includes Simulink models for current control and speed control systems. The example provides instructions to generate code for these models. To create a FOC algorithm that can run on any motor control hardware, integrate this generated code with the hardware peripheral code (either auto-generated or manually written). Ensure that you activate the **Current Control** and **Speed Control** systems at appropriate times for the prescribed time intervals as shown in the preceding figure.

The following figure shows the entire structure of the integrated code.

| Global variables and buffers |
| Hardware initialization |
| Peripheral initialization |
| SL model initialize |
| Main function (infinite loop) |
| ADC EOC 20KHz interrupt – Current control step function |
| Timer based 2KHz interrupt – Speed control step function |

User-generated (or manually coded) code

Simulink-generated code
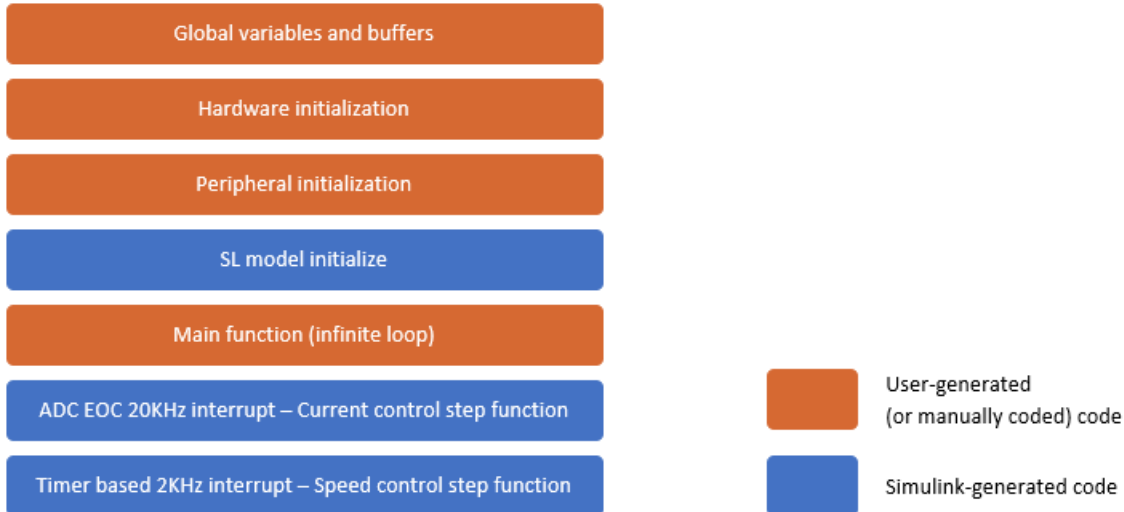
ADC-PWM Synchronization

The interaction between the PWM and ADC modules controls the rate of triggering of the **Current Control** subsystem. To reduce harmonics in the system, the PWM counter runs in center-aligned or up-down mode. We configure the PWM such that when PWM counter reaches the PWM counter period value, it triggers the ADC start of conversion (SOC) event. This ensures that the currents available at the ADC inputs are updated currents, and therefore, helps to measure the ADC currents correctly. Usually when you use shunt current sensors to measure the motor currents, the current-sense resistor is located on the lower legs of the inverter. Therefore, triggering the ADC SOC in the middle of the PWM period ensures that settled current values (No switching transients) are measured. This figure shows this interaction.
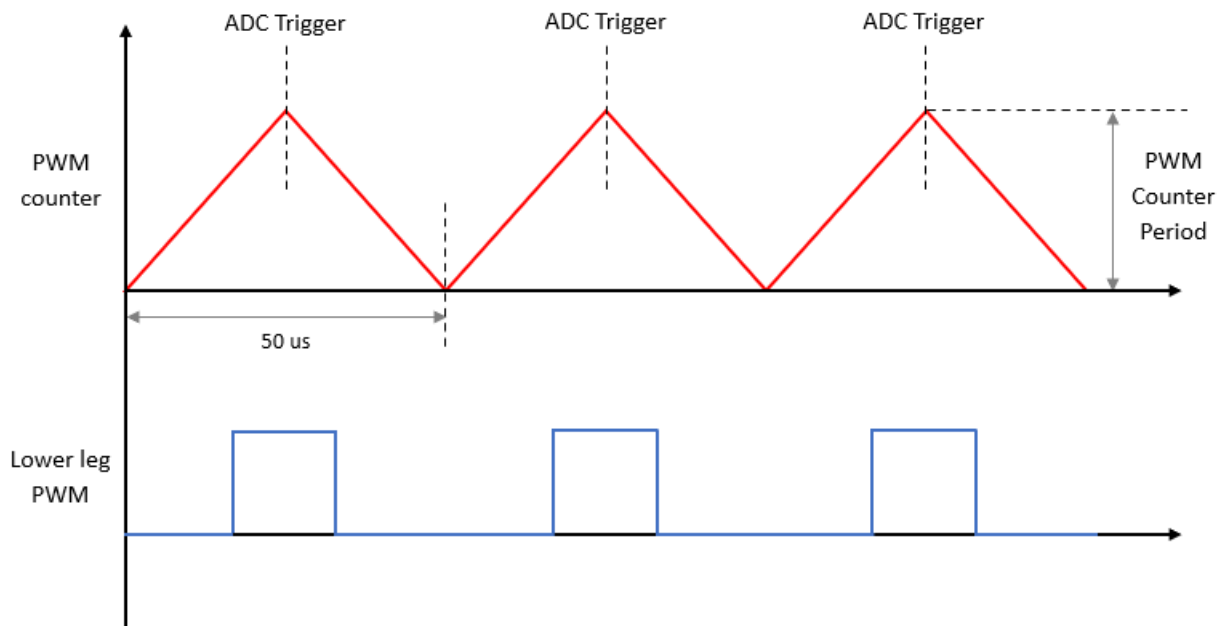
Motor Speed and Position Measurement

The FOC algorithm needs current position and speed of the motor. Usually either the position sensors or the sensorless estimation techniques help determine these values. The choice of the position sensing method depends on the factors such as cost, available space, required accuracy, and the motor control application itself. Motor Control Blockset supports these position sensing methods.

- Position sensors:

  - Quadrature encoder sensor
  - Hall sensor
  - Resolver
- Sensorless position estimation techniques:

  - Sliding mode observer
  - Flux observer

Serial Communication

Motor Control Blockset FOC examples use serial communication protocols for providing commands to the motor and reading debugging-related information from the motor control hardware. For details about this protocol, see "Fast Serial Data Monitoring" on page 6-3.

Hardware Connections

Hardware Connections

Motor Control Blockset supports implementing motor control algorithms using hardware kits from different vendors.

Vendor	Hardware Kit / Development Board with Demonstrated Capabilities of Motor Control Blockset	Used in
Texas Instruments	LAUNCHXL-F28069M	Multiple examples in this product [Motor Control Blockset]
	LAUNCHXL-F28379D	
	C2000 MCU Resolver Eval Kit [R2]	
	DRV8312-69M-KIT	
	LAUNCHXL-F280049C	Examples that list Motor Control Blockset as one of the required products in Embedded Coder Support Package for Texas Instruments C2000 Processors
STMicroelectronics®	NUCLEO-F401RE	Examples that list Motor Control Blockset as one of the required products in Embedded Coder Support Package for STMicroelectronics STM32 Processors
	X-NUCLEO-IHM07M1	
NXP™	MCSPTE1AK144	Field-Oriented Control of PMSM Using NXP™ S32K144 Kit
Microchip	dsPICDEM™ MCLV-2	Demo for Motor Control Deployment on Microchip Controllers
Speedgoat	Baseline real-time target machine with IO397 I/O module and Electric Motor Control Kit	"Tune PI Controllers Using Field Oriented Control Autotuner Block on Real-Time Systems" on page 4-107
Infineon®AURIX™	AURIX TC3xx Motor Control Power Board	Examples that list Motor Control Blockset as one of the required products in Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
Trenz Electronic™	Trenz Electronic™ Motor Control Development Kit TE0820	"FOC of PMSM Using FPGA-Based Motor Control Development Kit" on page 4-260

This section explains the hardware connection for the following hardware configurations, which are used in multiple examples in Motor Control Blockset:

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.
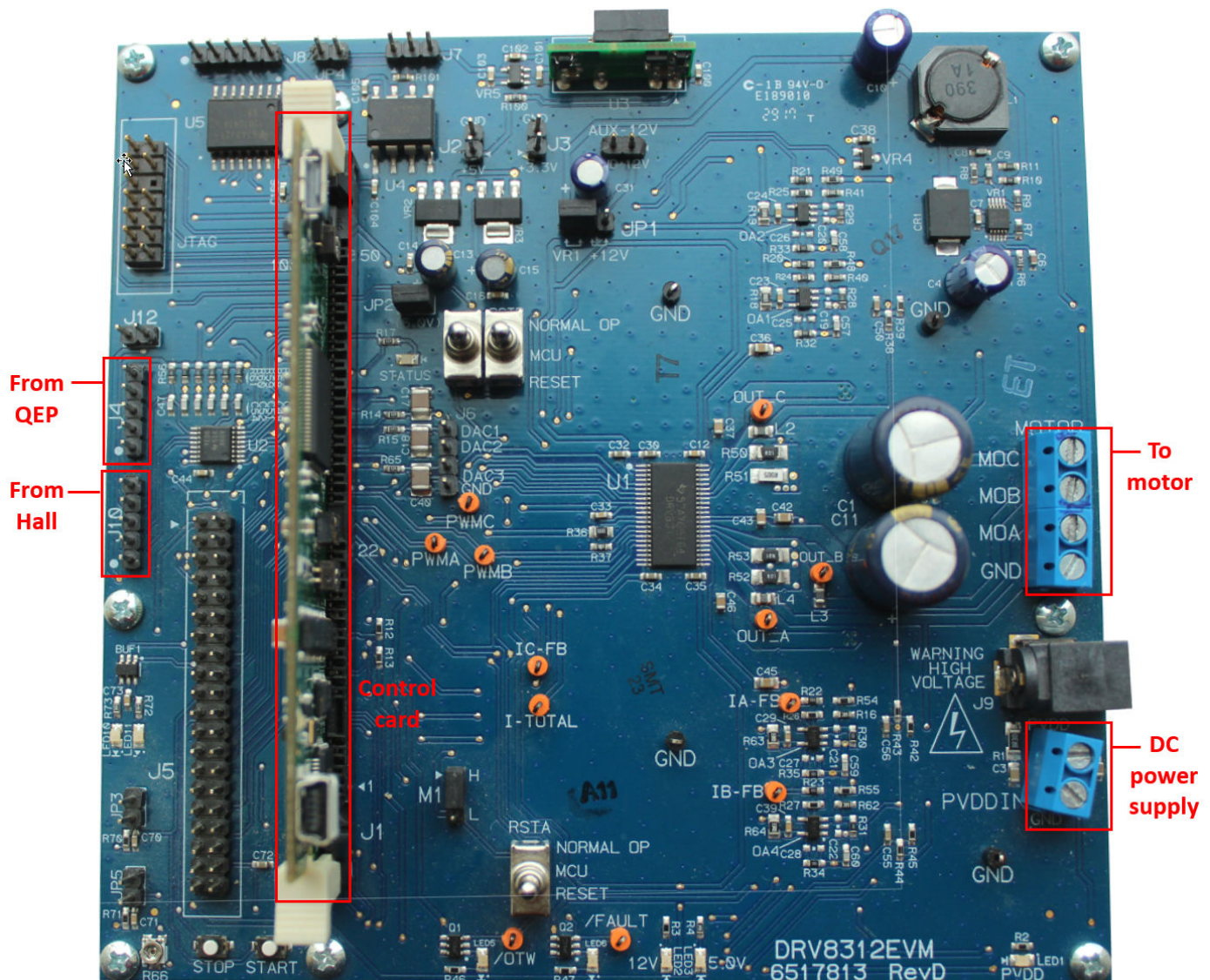
The following step describes about interfacing the quadrature encoder sensor:

- 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
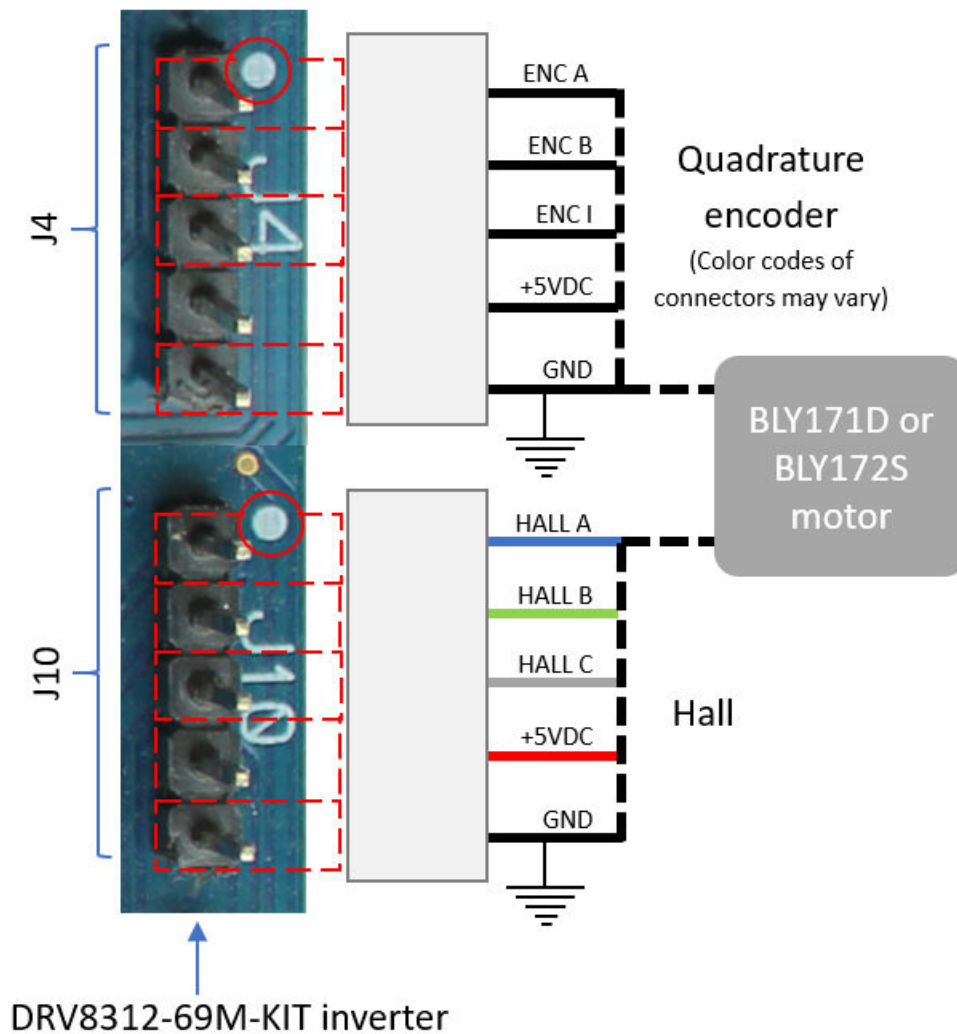
We recommend the following jumper settings for DRV8312-69M-KIT inverter board when working with Motor Control Blockset. You can customize these settings depending on the application requirements. For more information about these settings, see the device user guide available on Texas Instruments website.

- 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
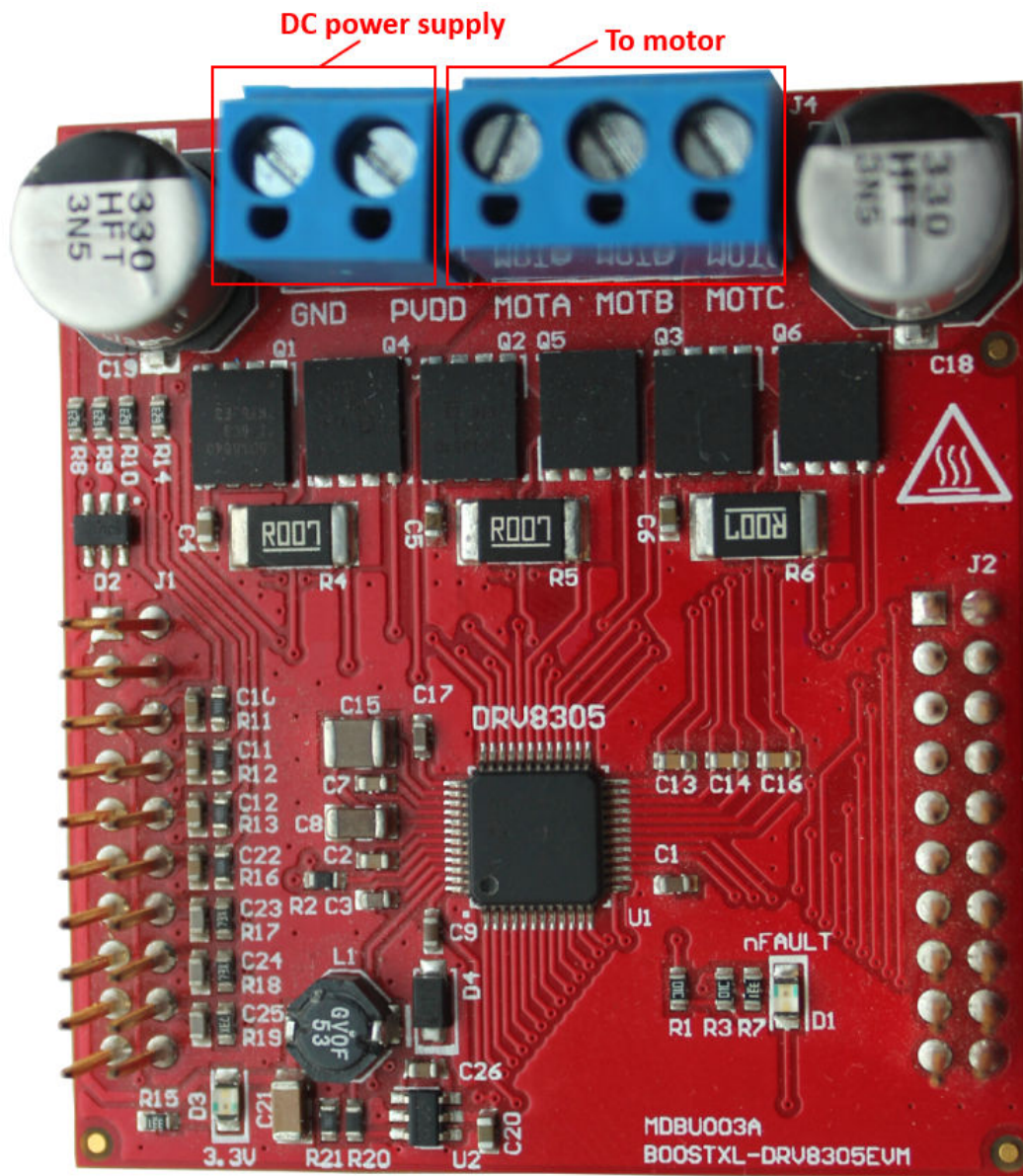
The following step describes about interfacing the quadrature encoder sensor:

- 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
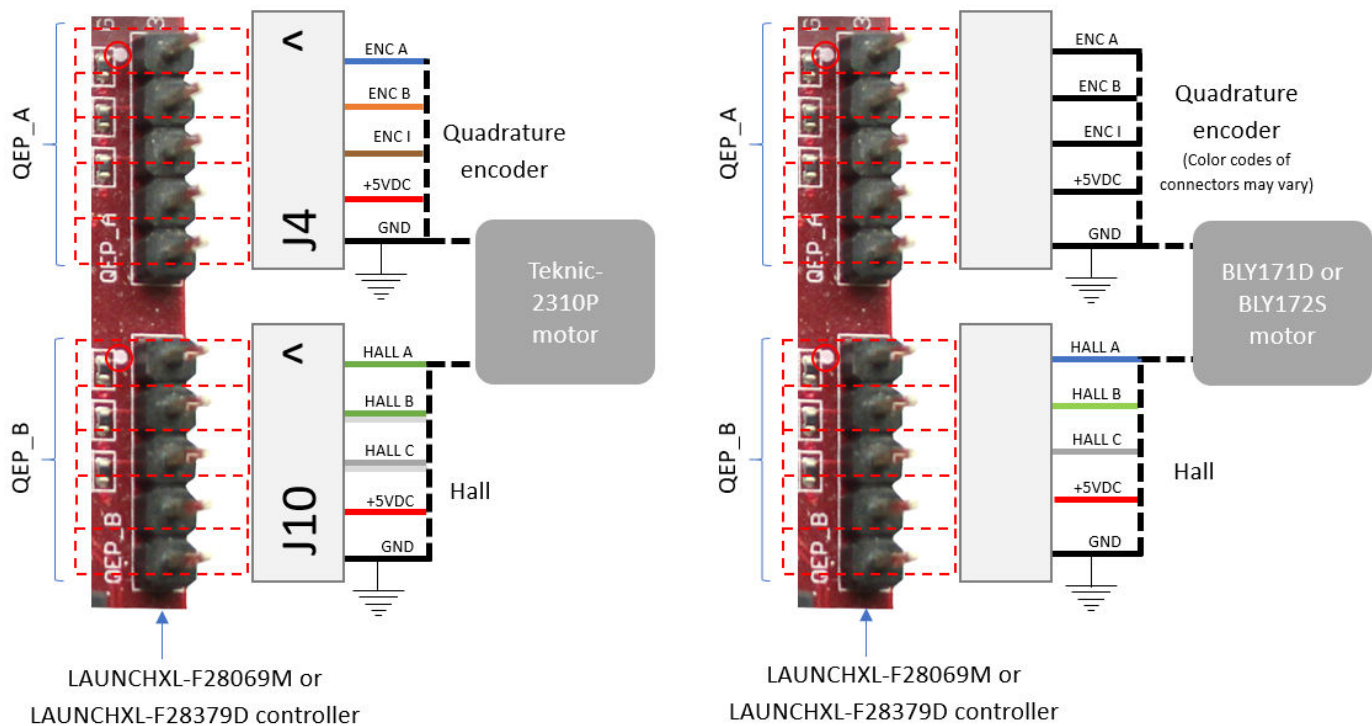
We recommend the following jumper settings for the LAUNCHXL inverter boards when working with Motor Control Blockset. You can customize these settings depending on the application requirements. For more information about these settings, see the device user guide available on Texas Instruments website.

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.
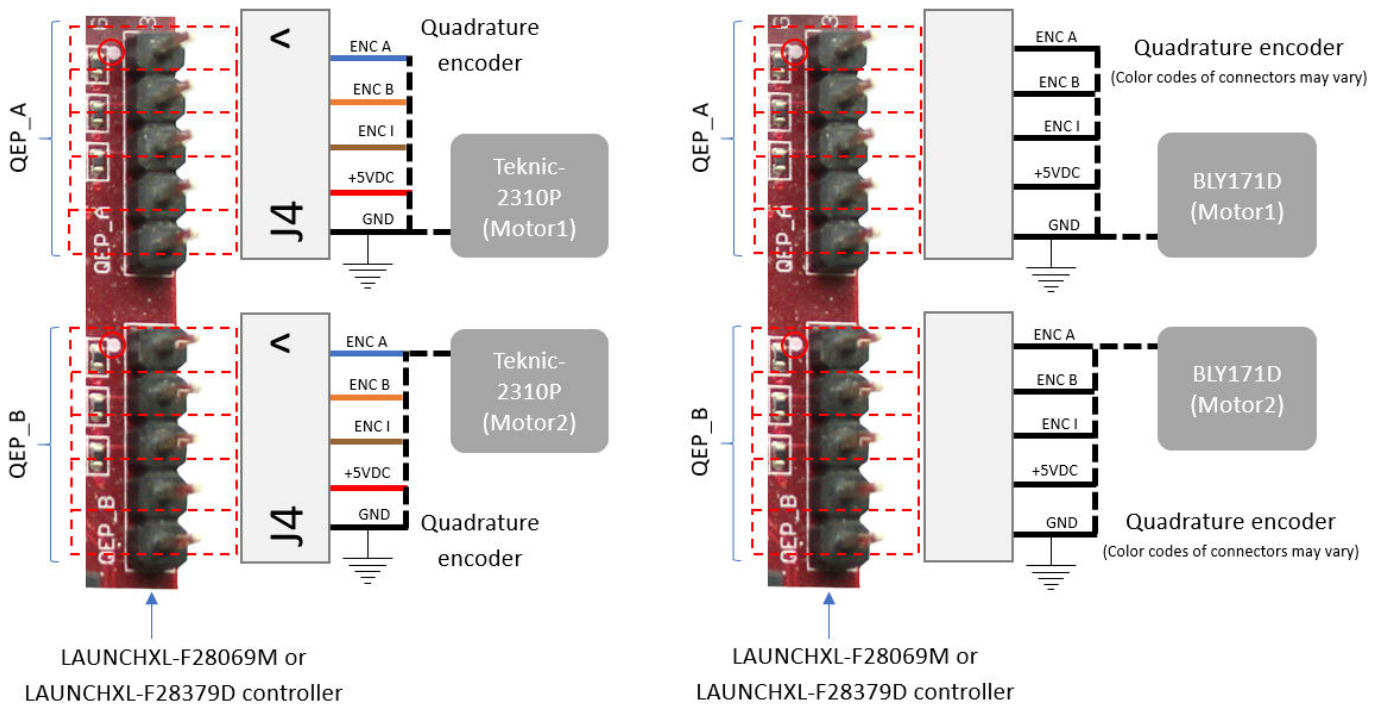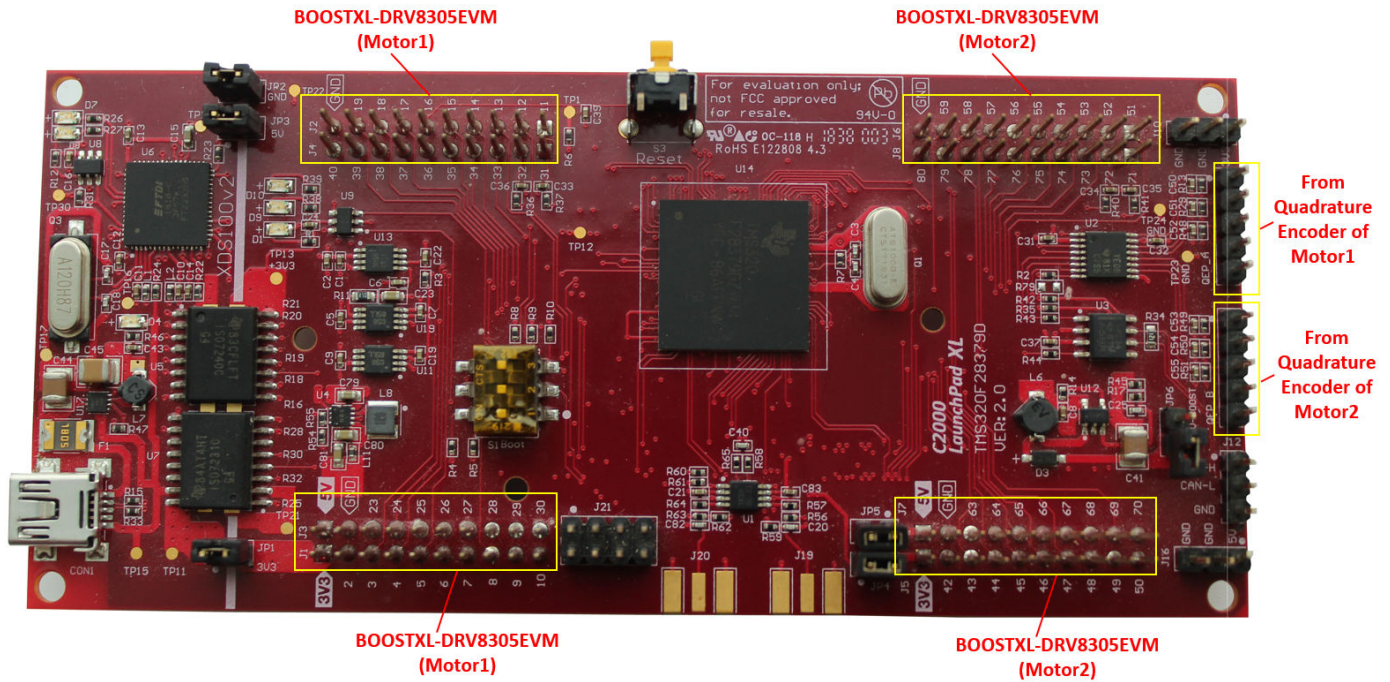
Instructions for Switched Reluctance Motor (SRM) Setup

A switched reluctance motor (SRM) connects to two BOOSTXL-DRV8305 inverters, which are mounted on the LAUNCHXL-F28379D controller card. These instructions describe the required hardware connections to run an SRM:

1   Attach the first BOOSTXL-DRV8305 inverter to J1, J2, J3, J4 on the LAUNCHXL-F28379D controller card.

> **Note** Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J1, J2 of LAUNCHXL.

2   Attach the second BOOSTXL-DRV8305 inverter to J5, J6, J7, J8 on the LAUNCHXL-F28379D controller card.

> **Note** Attach the inverter board to the controller board such that J1, J2 of BOOSTXL aligns with J5, J6 of LAUNCHXL.

3   Join the connectors A1, B1, C1, A2, B2, and C2 (which are connected to the inverter MOSFETs and diodes) with the motor phase windings as shown below:

- Connectors A1, A2 — phase A terminals of SRM
- Connectors B1, B2 — phase B terminals of SRM
- Connectors C1, C2 — phase C terminals of SRM

4   Connect the quadrature encoder pins (G, I, A, 5V, B) to QEP_A on the LAUNCHXL controller board.

5   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 to the same power supply.

> **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.
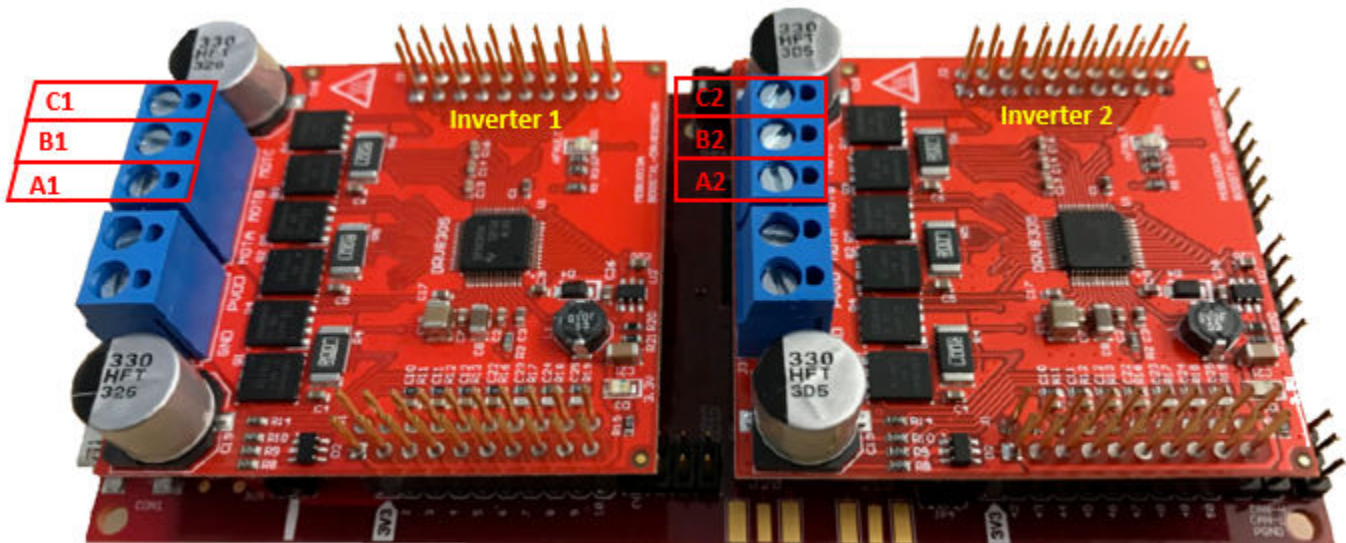
TMDSRSLVR C2000 Resolver to Digital Conversion Kit

The TMDSRSLVR C2000 Resolver to Digital Conversion Kit configuration includes the following hardware components:

- LAUNCHXL-F28069M controller
- BOOSTXL-DRV8305 (supported inverter)
- DC power supply
- TMDSRSLVR C2000 Resolver to Digital Conversion Kit (Resolver Eval Kit [R2])
- Resolver encoder

The following steps describe the hardware connections for the TMDSRSLVR board:

1 Connect DC power supply (15V) to J2 on the TMDSRSLVR board.
2 Connect the resolver output pins for sine wave to pins 1, 2 of J10 on the TMDSRSLVR board.
3 Connect the resolver output pins for cosine wave to pins 3, 4 of J10 on the TMDSRSLVR board.
4 Connect the resolver input pins to the PWM_dither and PWM_SINE pins of J10 on the TMDSRSLVR 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 the TMDSRSLVR board to pin 24 of J3 on the LAUNCHXL-F28069M controller board.
2 Connect the SIN(T8) pin on the TMDSRSLVR board to pin 29 of J3 on the LAUNCHXL-F28069M controller board.
3 Connect the GPIO2 pin on the TMDSRSLVR board 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
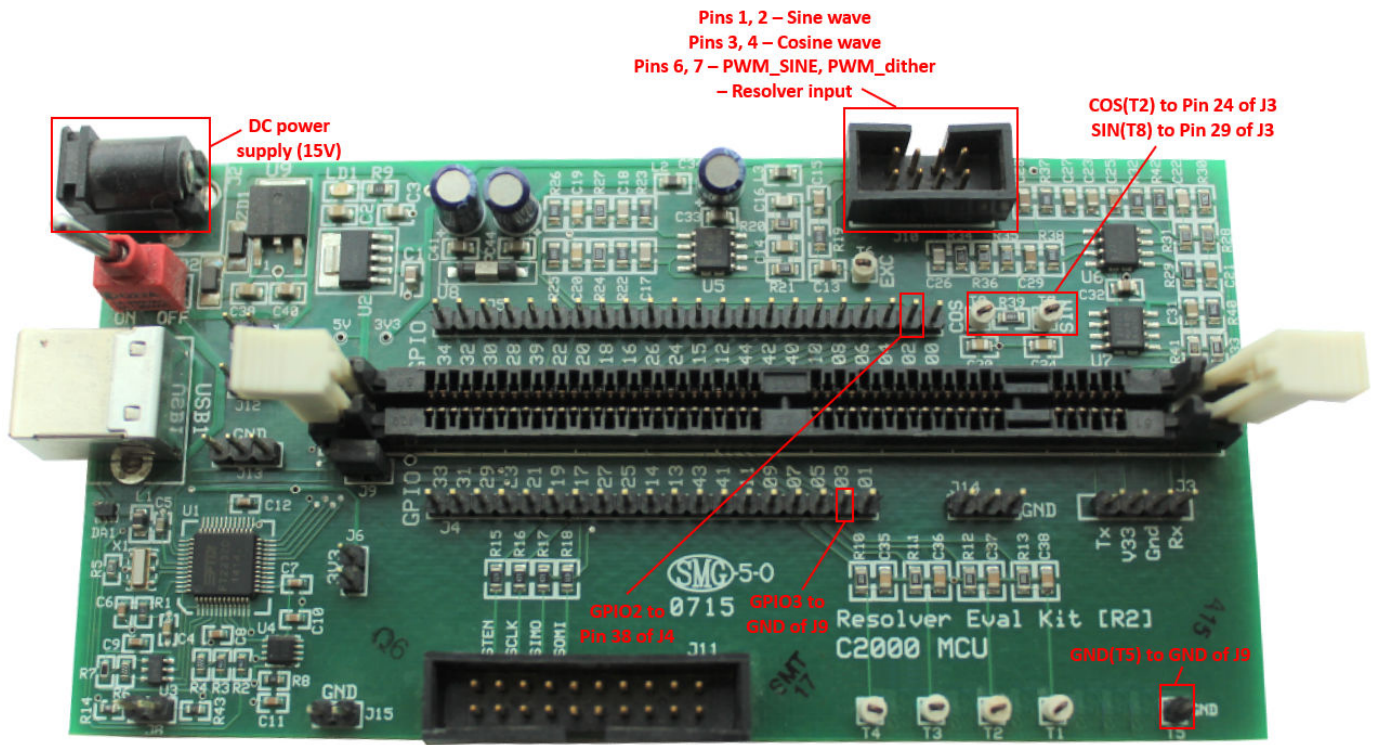
Algorithm Export Workflows for Custom Hardware

Open-Loop Control and ADC Offset Calibration

This is the first workflow that uses an algorithm to run a three-phase permanent magnet synchronous motor (PMSM) using open-loop control. The workflow 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. Before you run the host model on the host computer, build and deploy the target model algorithm (integrated with the hardware drivers) to the controller hardware board. The host model uses serial communication to command the target model algorithm and run the motor.

Expand the `open_loop` folder to access these files.

- open_loop_algorithm.slx (target model)
- open_loop_data.m (model initialization script associated with the target model)
- open_loop_host.slx (host model)

Generate Code For Control Algorithm Using Embedded Coder

1 After you open the MATLAB project, double-click the `open_loop_algorithm.slx` file in the `open_loop` folder.

2 Select **Modeling** > **Model Settings** > **Model Settings** to open the Configuration Parameters dialog box.

3 In the **Solver Selection** area of the **Solver** tab, update the **Type** and **Solver** fields.

4 In the **Hardware Implementation** tab of the Configuration Parameters dialog box, configure the parameters according to your hardware.

5 In the Simulink toolstrip of the target model, select **Apps > Embedded Coder** to open the Embedded Coder application.

6   In the Simulink toolstrip, select **C Code > Code Interface > Default Code Mappings** to open the Code Mappings - C dialog box.

7   In the Code Mappings - C dialog box, open the **Functions** tab.

8   For a listed C function, click the hyperlink under the **Function Preview** column to open the corresponding configuration interface dialog box.
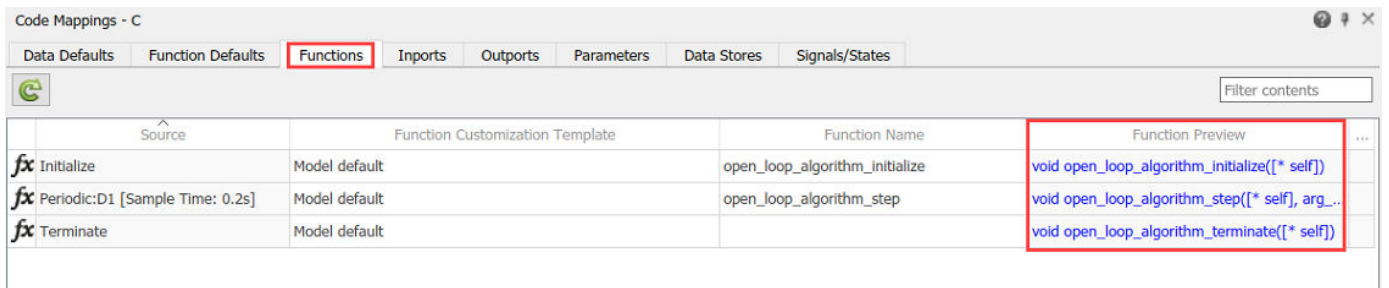
9   Use the Configure C Step Function Interface dialog box to configure the interface and arguments of the C function.
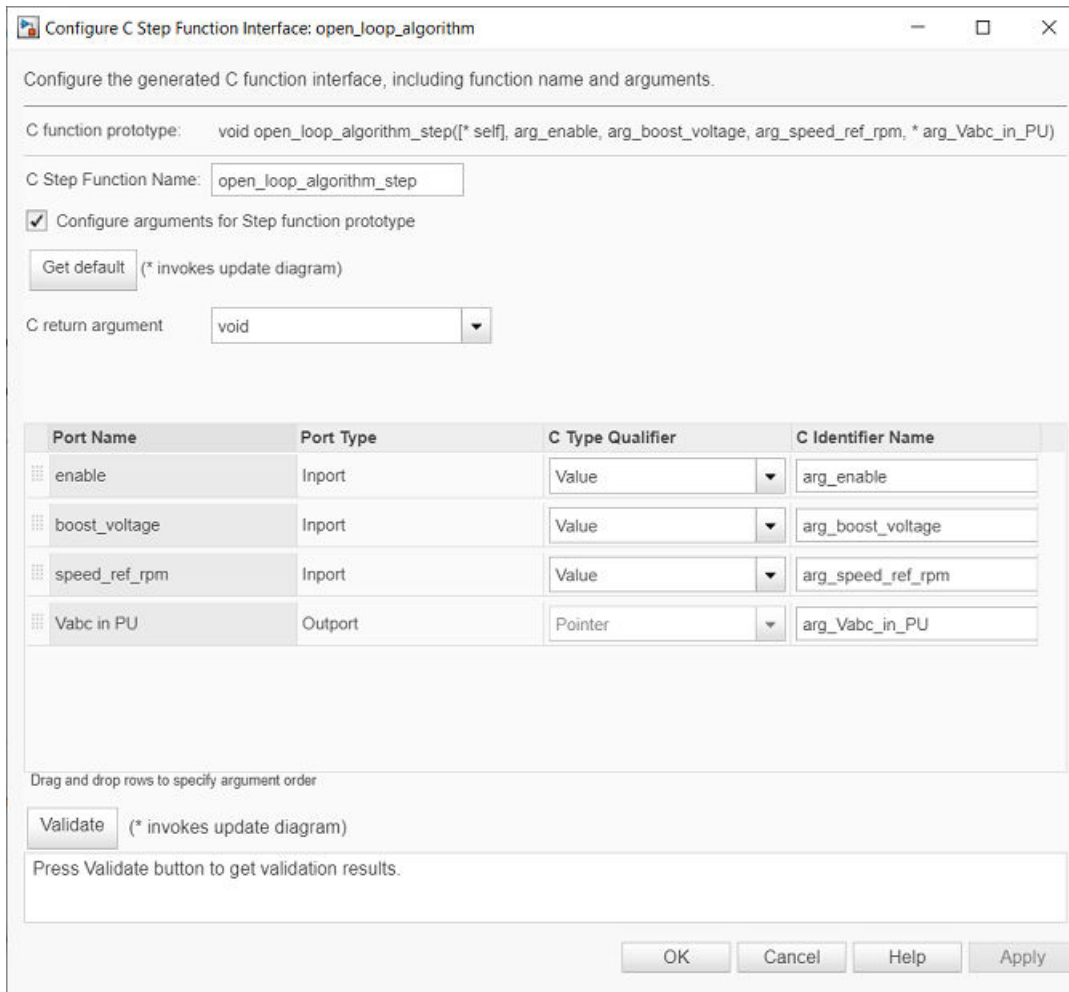
10   Click **Apply** and **OK** to complete configuring the C function.

11   Repeat steps 10 to 12 for all the listed functions.

12   In the Simulink toolstrip of the target model, select **C Code** > **Generate Code** > **Build** to build the model and generate a `.c` file for the target model.
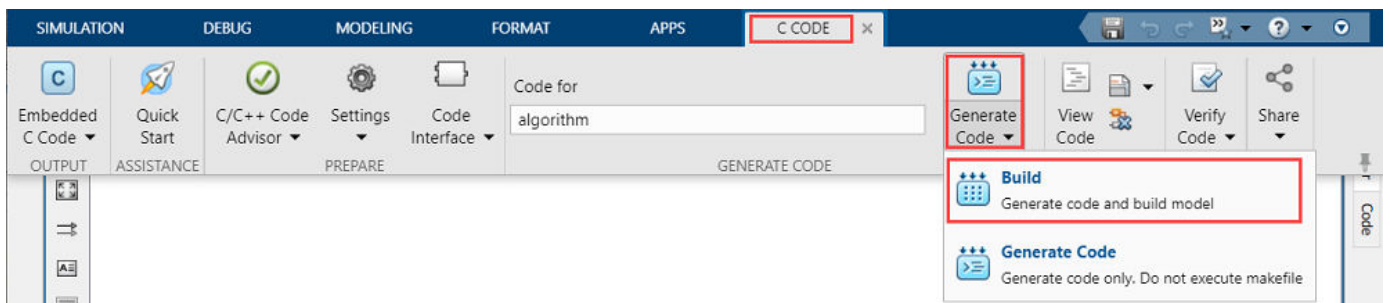
This image shows an example of the generated code for a C function.

```
/* Model step function */
void open_loop_algorithm_step(boolean_T arg_enable, real32_T arg_boost_voltage,
  real32_T arg_speed_ref_rpm, real32_T arg_Vabc_in_PU[3])
{
  real32_T rtb_Abs;
  real32_T rtb_Abs_g;
  real32_T rtb_Sum4;
  real32_T rtb_Sum6;
  real32_T rtb_Switch_o_idx_0;
  real32_T rtb_add_b;
  uint16_T rtb_Get_Integer_tmp_tmp;

  /* DiscreteIntegrator: '<S4>/Ramp Generator' incorporates: */
```

Note

- The generated C function uses the interface that you configured.
- For details about the per-unit system used in the algorithm, see "Per-Unit System" on page 6-20.

Obtain C Code For Hardware Drivers

You can use the code generation software supported by the hardware manufacturer to generate the code for the hardware drivers. For example, for the reference STM32F302R8 controller and X-NUCLEO-IHM07M1 inverter, you can use the STM32CubeMX STM32Cube initialization code generator software to configure the hardware peripherals and generate C code for the hardware drivers. The example also includes the FOC_QEP.ioc file (created by STM32CubeMX software) containing the hardware initialization data for the reference hardware.

Alternatively, you can also use a manually written driver code.

Integrate Control Algorithm Code With Driver Code

1. Call the control algorithm functions from the driver code using the configured control algorithm function parameters. This image shows a call to the speed control algorithm C function.

```
// call the open loop algorithm step function
open_loop_algorithm_step(ENABLE_INV, VOLTAGE_AMP_LOW_LIMIT, SPEED_REF, duty_vals);

// scale duty ratio to PWM counter period
for (int i = 0; i < 3; i++)
{
    duty_vals[i] = (1 - duty_vals[i]) * htim1.Init.Period;
}

// update duty cycles
```

2. Use the return value from the function call to complete integrating the driver with the controller algorithm.

For details about the code structure and program control flow used by the Motor Control Blockset examples, see "Program Control Flow of Motor Control Blockset Examples" on page 6-23.

View the integrated sample code `main.c` available in the `open_loop\STM32Code` folder as a reference.

Deploy Integrated Code to Hardware

1 Complete the hardware connections.

2 Use the code generation and deployment software supported by the hardware manufacturer to compile, build, and generate a binary (for example `.HEX`) file from the integrated code. Use the software to flash the binary file to the target hardware.

For example, for the reference STM32F302R8 controller and X-NUCLEO-IHM07M1 inverter, use the STM32CubeMX STM32Cube initialization code generator to generate and flash the binary file.

Control Motor Using Host Simulink Model

Follow these steps to run a three-phase PMSM using open-loop control:

1 Click the **host model** hyperlink in the target model to open the associated host model. You can also double-click the `open_loop_host.slx` file in the `open_loop` folder.

2 In the **Serial 1** tab of the block parameters dialog boxes for the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks, select a **Port name** and enter a **Baud rate** for serial communication.

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

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

4 Turn the **Motor** slider switch to the **Start** position to start running the motor.

5 Update the reference speed value in the **Speed Reference (RPM)** field. It is recommended that you set the speed to a value that is approximately half the rated speed of the motor.

6 After the motor runs, observe the ADC counts for the I_a and I_b currents in the time scope.

7 Stop the host model simulation and turn the **Motor** slider switch to the **Stop** position to stop the motor.

Follow these steps to determine the ADC offsets for the current sensors:

1 Disconnect the DC voltage supply from the hardware.

2 Disconnect the motor wires for three phases from the hardware board terminals and then reconnect the DC voltage supply to the hardware.

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

4 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. Follow these steps 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.
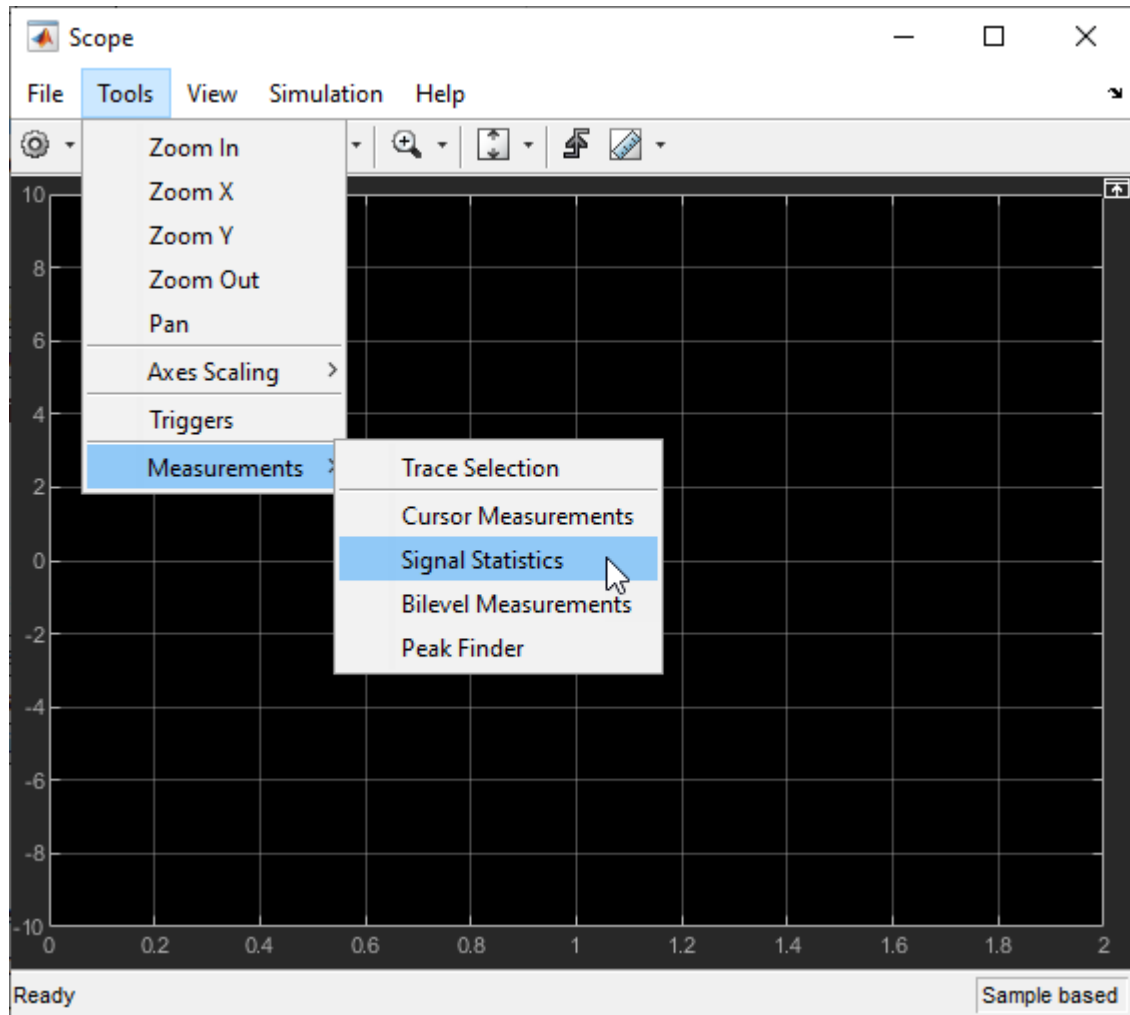
- Under **Trace Selection**, select a signal (I_a or I_b). The time scope displays the characteristics of the selected signal in the **Signal Statistics** area. You can see the median value of the selected signal in the **Median** field.
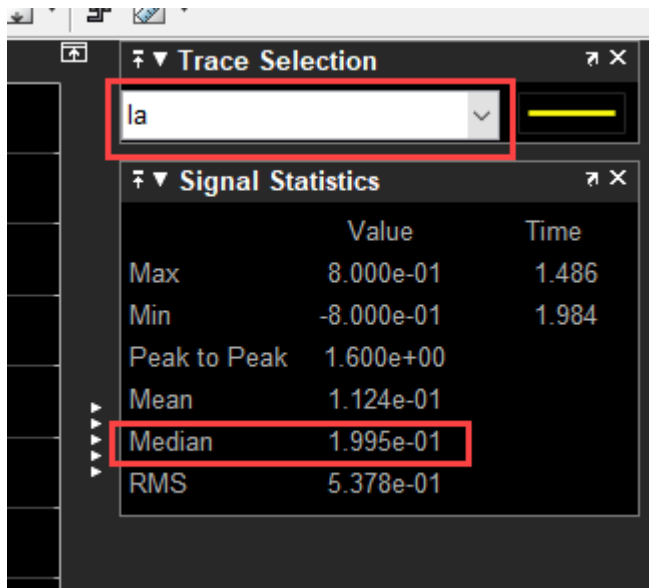
Update this ADC (or current) offset value in the *inverter.CtSensAOffset* and *inverter.CtSensBOffset* variables in the model initialization script linked to "Field-Oriented Control" on page 8-18.

To compute the offset of the quadrature encoder position sensor attached to the motor, see "Quadrature Encoder Offset Calibration" on page 8-11.

Quadrature Encoder Offset Calibration

This is the second workflow that uses an algorithm to calculate the offset between the *d*-axis of the rotor and the index pulse position as detected by the quadrature encoder sensor attached to the permanent magnet synchronous motor (PMSM). The workflow 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. Before you run the host model on the host computer, build and deploy the target model algorithm (integrated with the hardware drivers) to the controller hardware board. The host model uses serial communication to command the target model algorithm and run the motor.

Expand the `qep_calibration` folder to access these files.

- `qep_calibration_algorithm.slx` (target model)
- `qep_calibration_data.m` (model initialization script associated with the target model)
- `qep_calibration_host.slx` (host model)

Generate Code For Control Algorithm Using Embedded Coder

1 After you open the MATLAB project, double-click the `qep_calibration_algorithm.slx` file in the `qep_calibration` folder.

2 Select **Modeling > Model Settings > Model Settings** to open the Configuration Parameters dialog box.

3 In the **Solver Selection** area of the **Solver** tab, update the **Type** and **Solver** fields.

4   In the **Hardware Implementation** tab of the Configuration Parameters dialog box, configure the parameters according to your hardware.

5   In the Simulink toolstrip of the target model, select **Apps > Embedded Coder** to open the Embedded Coder application.

6 In the Simulink toolstrip, select **C Code > Code Interface > Default Code Mappings** to open the Code Mappings - C dialog box.

7 In the Code Mappings - C dialog box, open the **Functions** tab.

8 For a listed C function, click the hyperlink under the **Function Preview** column to open the Configure C Initialize Function Interface dialog box.

9 Use the Configure C Initialize Function Interface dialog box to configure the interface and arguments of the C function.

10   Click **Apply** and **OK** to complete configuring the C function.

11   Repeat steps 10 to 12 for all the listed functions.

12   In the Simulink toolstrip of the target model, select **C Code > Generate Code > Build** to build the model and generate a `.c` file for the target model.

This image shows an example of the generated code for a C function.

```
/* Model step function */
void qep_calibration_algorithm_step(boolean_T arg_enable, uint16_T arg_index,
  uint16_T arg_count, real32_T arg_Vabc_pu[3], real32_T *arg_mode, real32_T
  *arg_pos_pu)
{
  real32_T rtb_Abs;
  real32_T rtb_Sum4;
  real32_T rtb_Switch_j;
  real32_T rtb_Switch_o_idx_0;
  real32_T rtb_add_c;
  uint16_T rtb_Sum3;

  /* Outputs for Enabled SubSystem: '<S1>/counter' incorporates:
```

Note

- The generated C function uses the interface that you configured.
- For details about the per-unit system used in the algorithm, see "Per-Unit System" on page 6-20.

Obtain C Code For Hardware Drivers

You can use the code generation software supported by the hardware manufacturer to generate the code for the hardware drivers. For example, for the reference STM32F302R8 controller and X-NUCLEO-IHM07M1 inverter, you can use the STM32CubeMX STM32Cube initialization code generator software to configure the hardware peripherals and generate C code for the hardware drivers. This example also includes the `FOC_QEP.ioc` file (created by STM32CubeMX software) containing the hardware initialization data for the reference hardware.

Alternatively, you can also use a manually written driver code.

Integrate Control Algorithm Code With Driver Code

1   Call the control algorithm functions from the driver code using the configured control algorithm function parameters. This image shows a call to the speed control algorithm C function.

```
// call the QEP calibration algorithm step function
qep_calibration_algorithm_step(ENABLE_INV, QEP_INDEX_COUNT, QEP_COUNT, duty_vals, &mode, &position_meas);

uint16_t debug_1 = (uint16_t)(position_meas * 65535);
uint16_t debug_2 = (uint16_t)mode;

// scale duty ratio to PWM counter period
for (int i = 0; i < 3; i++)
{
    duty_vals[i] = (1 - duty_vals[i]) * htim1.Init.Period;
}

// update duty cycles
```

2   Use the return value from the function call to complete integrating the driver with the controller algorithm.

For details about the code structure and program control flow used by the Motor Control Blockset examples, see "Program Control Flow of Motor Control Blockset Examples" on page 6-23.

View the integrated sample code `main.c` available in the `qep_calibration\STM32Code` folder as a reference.

Deploy Integrated Code to Hardware

1   Complete the hardware connections.

2   Use the code generation and deployment software supported by the hardware manufacturer to compile, build, and generate a binary (for example `.HEX`) file from the integrated code. Use the software to flash the binary file to the target hardware.

For example, for the reference STM32F302R8 controller and X-NUCLEO-IHM07M1 inverter, use the STM32CubeMX STM32Cube initialization code generator to generate and flash the binary file.

Control Motor Using Host Simulink Model

Follow these steps to determine the offset of the quadrature encoder sensor attached to a three-phase PMSM:

1   Click the **host model** hyperlink in the target model to open the associated host model. You can also double-click the `qep_calibration_host.slx` file in the `qep_calibration` folder.

Copyright 2021 The MathWorks, Inc.

2 In the **Serial 1** tab of the block parameters dialog boxes for the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks, select a **Port name** and enter a **Baud rate** for serial communication.

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

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

4 Turn the **Calibration** slider switch to the **Start** position to start the calibration process by running the motor.

5 Observe the *position* signal in the time scope.

After the calibration process is complete, simulation ends and the motor stops automatically. The model displays the computed encoder offset value in the **QEP offset (pu)** field.

6 Check if the `position` signal has a positive slope. Follow these steps, if the slope is negative:

- Turn the **Calibration** slider switch to the **Stop** position to stop the calibration process and stop running the motor.
- Disconnect the DC voltage supply from the hardware.
- Interchange any two motor phase connections and then reconnect the DC voltage supply to the hardware.
- Follow steps 4 and 5 to run the calibration algorithm again and determine the encoder offset value.

The example automatically saves the computed offset value in the *PositionOffset* variable available in the base workspace.

Update computed offset value in the *pmsm.PositionOffset* variable available in the model initialization script linked to "Field-Oriented Control" on page 8-18.

Field-Oriented Control

This is the third workflow that runs a three-phase permanent magnet synchronous motor (PMSM) using closed-loop field-oriented control (FOC). For more details about FOC, see "Field-Oriented Control (FOC)" on page 4-3. The workflow 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. Before you run the host model on the host computer, build and deploy the target model algorithm (integrated with the hardware drivers) to the controller hardware board. The host model uses serial communication to command the target model algorithm and run the motor.

Expand the `foc_qep` folder to access these files.

- `current_control_algorithm.slx` (target model for current controller)
- `speed_control_algorithm.slx` (target model for speed controller)
- `foc_qep_sim.slx` (target model for simulation)
- `foc_qep_data.m` (model initialization script associated with the target models)
- `foc_qep_host.slx` (host model)

Before using this workflow, complete "Open-Loop Control and ADC Offset Calibration" on page 8-2 and "Quadrature Encoder Offset Calibration" on page 8-11 to compute the ADC and position sensor offsets. Update the offsets in these variables available in the model initialization script `foc_qep_data.m`:

- *inverter.CtSensAOffset* and *inverter.CtSensBOffset* variables (ADC offsets for I_a and I_b current sensors)
- *pmsm.PositionOffset* (position offset for quadrature encoder sensor)

Simulate Model

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

1 Open the model `foc_qep_sim.slx`.
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 For Control Algorithm Using Embedded Coder

1 After you open the MATLAB project, double-click the `current_control_algorithm.slx` file in the `foc_qep` folder.

Current Control Algorithm
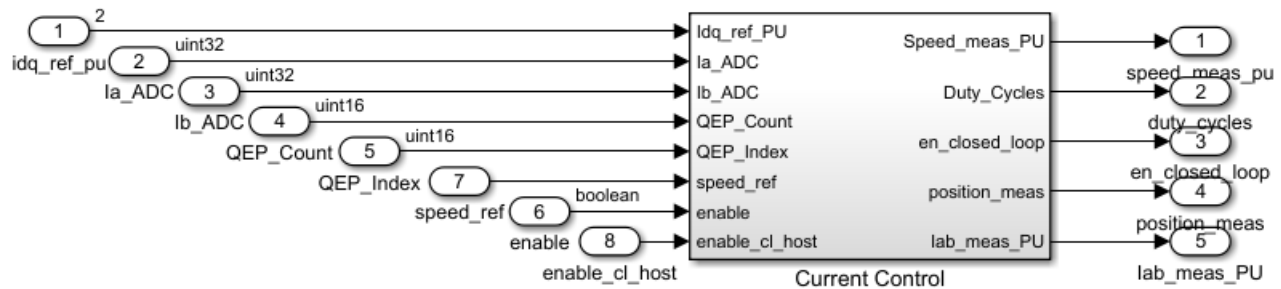

Copyright 2021 The MathWorks, Inc.

Explore more:
1. Edit motor & inverter parameters
2. Generate c code using the 'Embedded Coder' app
3. Integrate generated code with driver code
4. Control motor via host model

2   Select **Modeling** > **Model Settings** > **Model Settings** to open the Configuration Parameters dialog box.

3   In the **Solver Selection** area of the **Solver** tab, update the **Type** and **Solver** fields.
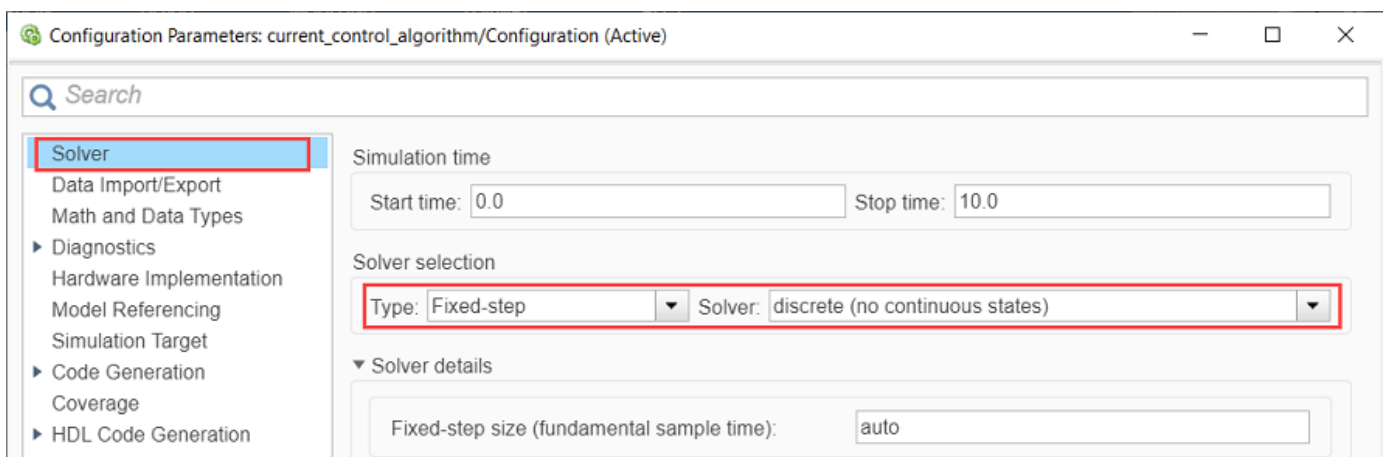
4   In the **Hardware Implementation** tab of the Configuration Parameters dialog box, configure the parameters according to your hardware.
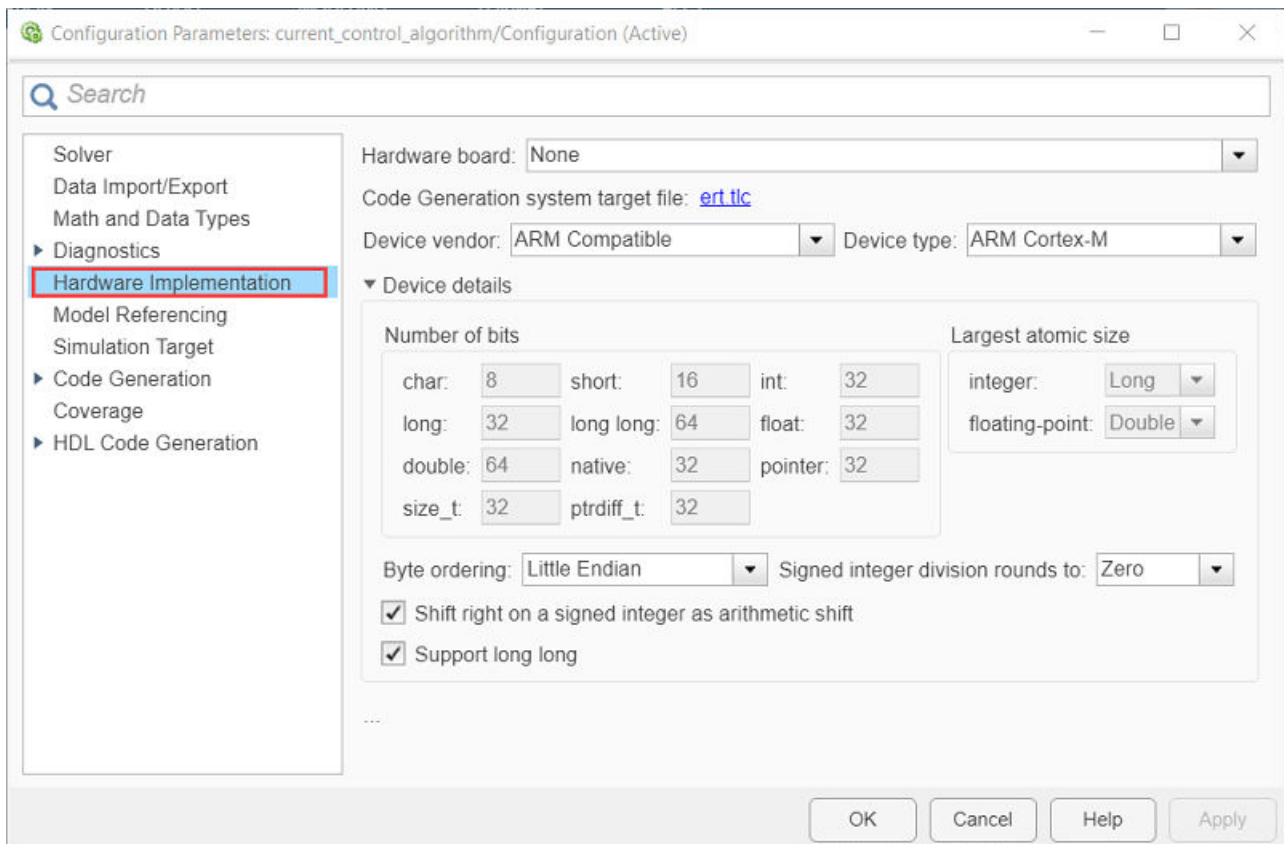
5   Obtain the motor and inverter parameters. The target model uses default motor parameters that you can replace with values from either the motor datasheet or other sources.

However, you can use your motor control hardware to estimate the parameters for the motor that you want to use by using the Motor Control Blockset parameter estimation tool. For instructions, see "Estimate PMSM Parameters Using Custom Hardware" on page 4-209. The parameter estimation tool updates the *motorParam* variable (in the MATLAB workspace) with the estimated motor parameters.

6   Update the motor and inverter parameters. If you obtain the motor parameters from the datasheet or from other sources, update the motor and inverter parameters in the model initialization script (`foc_qep_data.m`) associated with the Simulink model. For instructions, see "Estimate Control Gains and Use Utility Functions" 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.

7   In the Simulink toolstrip of the target model, select **Apps > Embedded Coder** to open the Embedded Coder application.
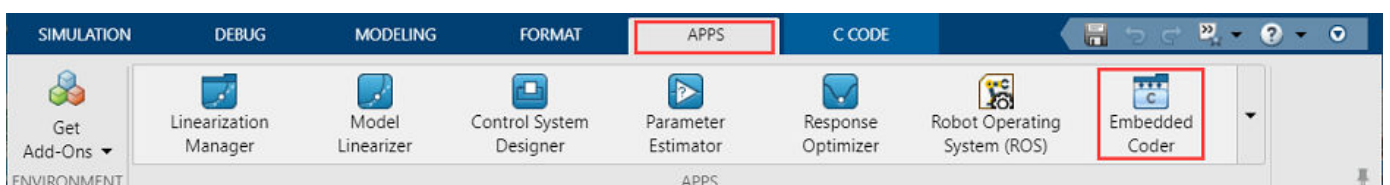
8 In the Simulink toolstrip, select **C Code > Code Interface > Default Code Mappings** to open the Code Mappings - C dialog box.
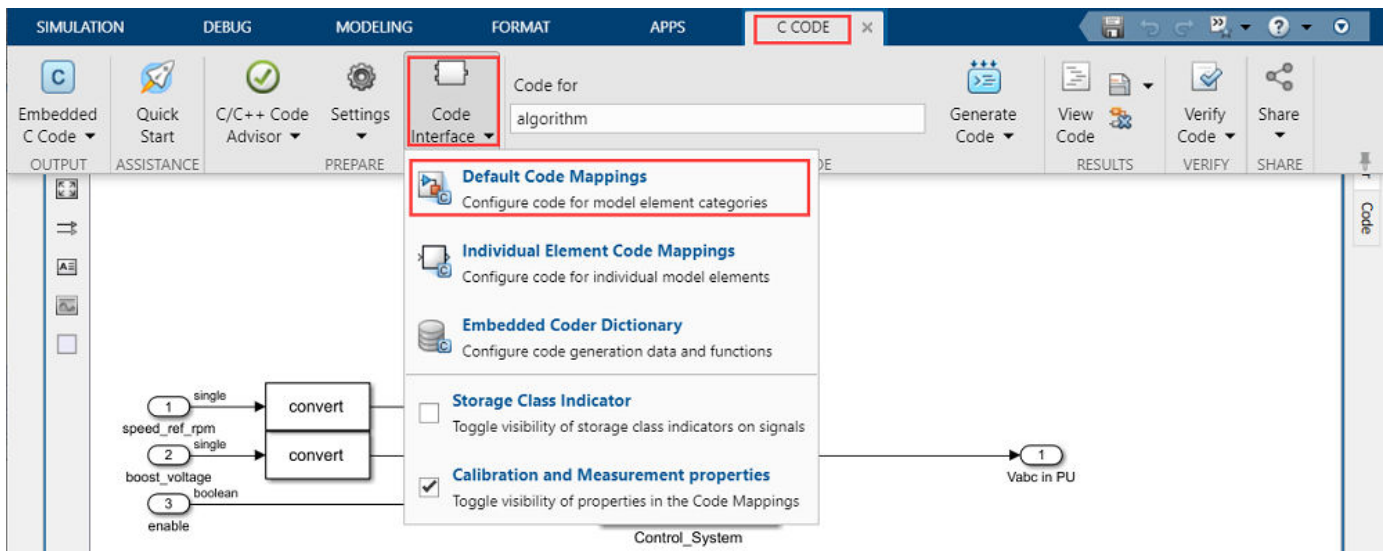
9 In the Code Mappings - C dialog box, open the **Functions** tab.

10 For a listed C function, click the hyperlink under the **Function Preview** column to open the Configure C Initialize Function Interface dialog box.

11 Use the Configure C Initialize Function Interface dialog box to configure the interface and arguments of the C function.

12 Click **Apply** and **OK** to complete configuring the C function.

13 Repeat steps 10 to 12 for all the listed functions.

14 In the Simulink toolstrip of the target model, select **C Code > Generate Code > Build** to build the model and generate a `.c` file for the target model for current controller.
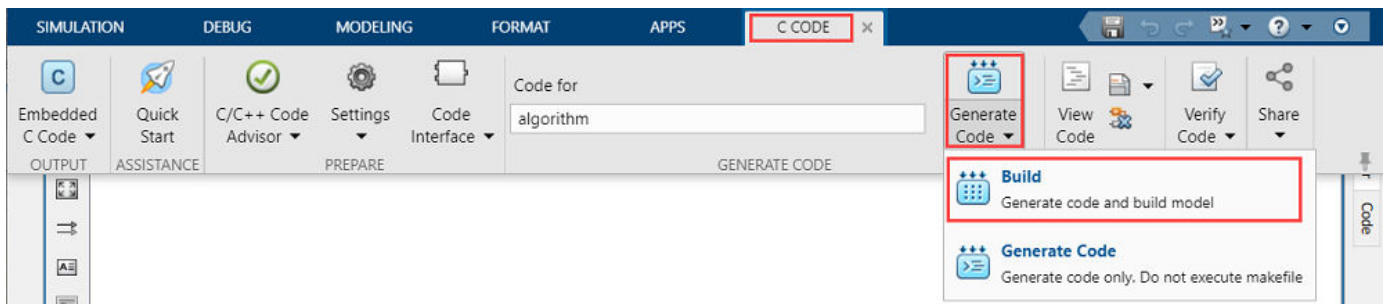
This image shows an example of a C function available in the generated current controller code.

```
/* Model step function */
void current_control_algorithm_step(boolean_T arg_enable, boolean_T
  arg_enable_cl_host, uint32_T arg_Ia_ADC, uint32_T arg_Ib_ADC, uint16_T
  arg_QEP_Index, uint16_T arg_QEP_Count, real32_T arg_speed_ref, real32_T
  arg_idq_ref_pu[2], real32_T *arg_speed_meas_pu, real32_T arg_duty_cycles[3],
  boolean_T *arg_en_closed_loop, real32_T *arg_position_meas, real32_T
  arg_Iab_meas_PU[2])
{
  real32_T rtb_Add2_n;
  real32_T rtb_Frequency;
  real32_T rtb_PositionGain;
  real32_T rtb_algDD_o1_c;
  real32_T rtb_one_by_two;
  uint32_T rtb_PositionToCount;
  uint16_T rtb_Sum3;

  /* Outputs for IfAction SubSystem: '<S55>/PositionNoReset' incorporates:
```

Note

- The generated C function uses the interface that you configured in step 11.
- For details about the per-unit system used in the algorithm, see "Per-Unit System" on page 6-20.

Repeat steps 1 to 14 for the target model for speed controller (`speed_control_algorithm.slx`) to generate the speed controller code.

Obtain C Code For Hardware Drivers

You can use the code generation software supported by the hardware manufacturer to generate the code for the hardware drivers. For example, for the reference STM32F302R8 controller and X-NUCLEO-IHM07M1 inverter, you can use the STM32CubeMX STM32Cube initialization code generator software to configure the hardware peripherals and generate C code for the hardware drivers. The example also includes the `FOC_QEP.ioc` file (created by STM32CubeMX software) containing the hardware initialization data for the reference hardware.

Alternatively, you can also use a manually written driver code.

Integrate Control Algorithm Code With Driver Code

1 Integrate the code for the speed and current controllers.
2 Call the control algorithm functions from the driver code using the configured control algorithm function parameters. This image shows a call to the speed control algorithm C function.

```
// call the current control step function
current_control_algorithm_step(ENABLE_INV, ENABLE_CL_HOST, Ia, Ib, QEP_INDEX_COUNT, QEP_COUNT, SPEED_REF, (real32_T *)IDQ_REF,
        (real32_T *)&SPEED_MEAS_PU, duty_vals, (boolean_T *)&CL_ENABLE, &pos_meas, Iab_meas_pu);

uint16_t debug_1 = (uint16_t)(((Iab_meas_pu[0] + 1)/2) * 65535);
uint16_t debug_2 = (uint16_t)(((SPEED_MEAS_PU + 1)/2) * 65535);

// scale duty ratio to PWM counter period
for (int i = 0; i < 3; i++)
{
    duty_vals[i] = (1 - duty_vals[i]) * htim1.Init.Period;
}

// update duty cycles
```

3   Use the return value from the function call to complete integrating the driver with the controller algorithm.

For details about the code structure and program control flow used by the Motor Control Blockset examples, see "Program Control Flow of Motor Control Blockset Examples" on page 6-23.

View the integrated sample code `main.c` available in the `foc_qep\STM32Code` folder as a reference.

Deploy Integrated Code to Hardware

1   Complete the hardware connections.

2   Use the code generation and deployment software supported by the hardware manufacturer to compile, build, and generate a binary (for example `.HEX`) file from the integrated code. Use the software to flash the binary file to the target hardware.

   For example, for the reference STM32F302R8 controller and X-NUCLEO-IHM07M1 inverter, use the STM32CubeMX STM32Cube initialization code generator to generate and flash the binary file.

Control Motor Using Host Simulink Model

Follow these steps to determine the offset of the quadrature encoder sensor attached to a three-phase PMSM:

1   Click the **host model** hyperlink in the target model to open the associated host model. You can also double-click the `foc_qep_host.slx` file in the `foc_qep` folder.

FOC QEP Host Control

Steps:

1. Set the baud rate for serial communication in **'Host Serial Setup'** block.

2. Select the serial port in **'Serial 1'** tab of **'Host Serial Setup'** block.

3. Use **'Motor Start / Stop'** switch to control the start and stop of the motor.

4. Use the **'Motor Control Open Loop / Closed Loop'** switch to switch between open loop and closed loop control.

4. Enter speed request in RPM using 'Speed Reference' edit box. Limit the reference speed to half of the rated speed while starting the motor in open loop.

5. Observe the Ia and measured speed signals on the scope.
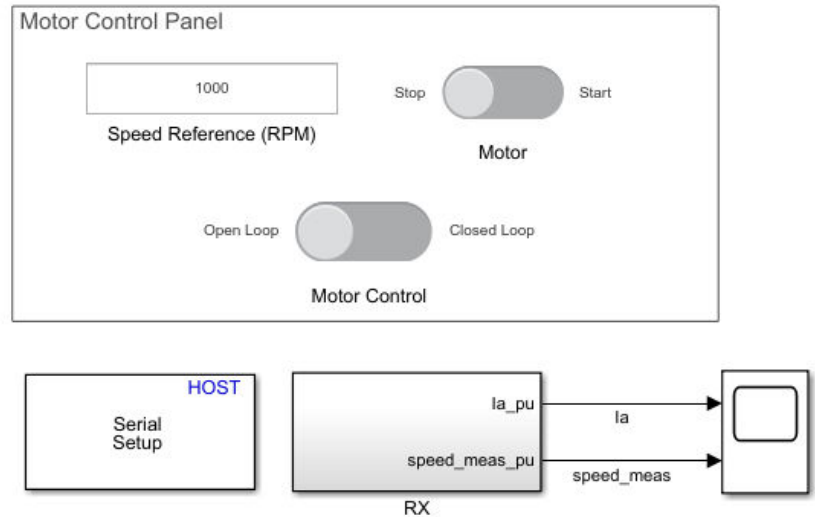
Motor Control Panel

1000 | Stop ⬤ Start
Speed Reference (RPM) | Motor

Open Loop ⬤ Closed Loop
Motor Control

HOST
Serial Setup

RX
Ia_pu → Ia
speed_meas_pu → speed_meas

Copyright 2021 The MathWorks, Inc.

2 In the **Serial 1** tab of the block parameters dialog boxes for the Host Serial Setup, Host Serial Receive, and Host Serial Transmit blocks, select a **Port name** and enter a **Baud rate** for serial communication.

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

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

4 Ensure that the current position of **Motor Control** slider switch is **Open Loop**. Turn the **Motor Start / Stop** slider switch to the **Start** position to start running the motor using open-loop control.

5 Before entering closed-loop control, enter the reference speed for the motor in the **Speed Reference (RPM)** field. It is recommended that you set the speed to a value that is approximately half the rated speed of the motor.

6 Turn the **Motor Control** slider switch to the **Closed Loop** position to start running the motor using closed-loop field-oriented control.

7 Observe the I_a and speed signals in the time scope.

Modeling Guidelines for Motor Control Applications

Create and Validate Model for Motor Control System

Use Motor Control Blockset to design a Simulink model for a motor control system and validate it after deploying the model to the motor control hardware. This table lists the fundamental steps for creating and deploying a field-oriented control algorithm for a PMSM.

Note You can use a similar procedure to design and deploy a control algorithm for the other types of motors.

Modeling Steps	Learn More
Compute estimated motor parameters and create plant model	"Estimate PMSM Parameters Using Recommended Hardware" on page 4-186
	"Creating Plant Model Using Motor Control Blockset"
Create controller algorithm for motor control system	"Design Field-Oriented Control Algorithm"
	"Estimate Control Gains and Use Utility Functions" on page 3-2
	"Code Verification and Profiling Using PIL Testing" on page 4-283
Deploy and validate motor control system	"Prepare Target Hardware"
	"Add Hardware Drivers to Simulation Model and Deploy to Target Hardware"
	"Validate System"

See these instructions to identify and debug common problems that can occur when you run the model on the target hardware.

Troubleshooting Steps	Learn More
Check for errors in measurements by analog to digital converter (ADC) peripheral in motor control applications	"Check ADC Inputs"
Verify pulse width modulation (PWM) signals	"Verify PWM Outputs"
Check for issues in hardware connections	"Check Hardware Connections"
Check for issues in algorithm used for model	"Test Algorithm Design"
Check for problems related to software architecture, code performance, and code execution time	"Check Generated Code"

Using Hall Validity and Hall Decoder Blocks

How to Use Hall Validity and Hall Decoder Blocks

If you are using Hall position sensors to obtain the position feedback, follow this procedure to integrate the field-oriented control (FOC) algorithm with the Hall sensors and decode the rotor position and speed values.

To determine the rotor position, direction of rotation, and an accurate rotor speed, we need at least three Hall sensors inside the motor. To increase the accuracy of the computed rotor position and speed values, you can use a motor that has more than three Hall sensors. This procedure uses the model `mcb_pmsm_foc_hall_f28069m.slx` as a reference. In addition, it assumes a hardware setup that uses a permanent magnet synchronous motor (PMSM) with three Hall sensors that are placed 120 degrees apart.
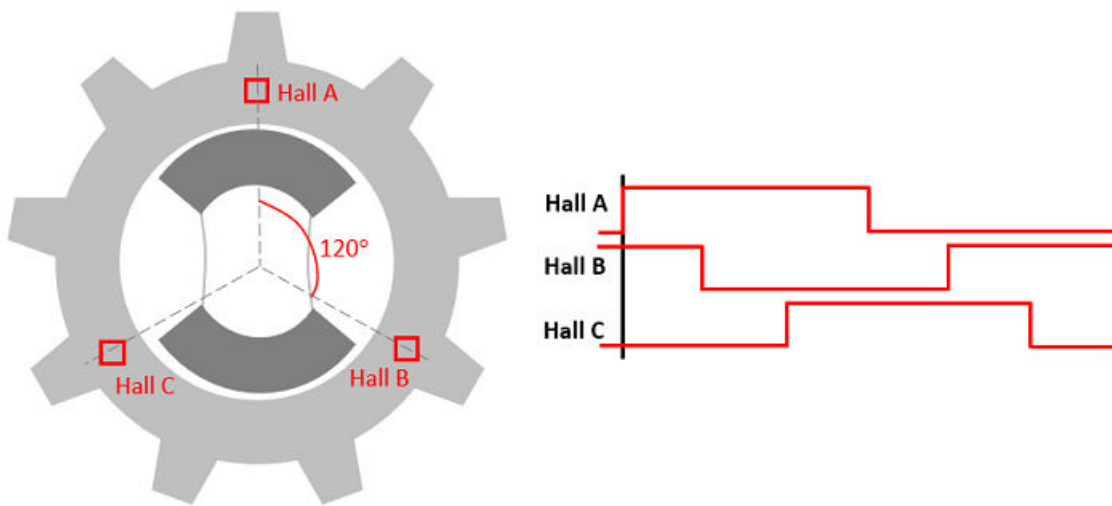
Configure eCAP Pins

After you connect the three Hall sensors to the GPIO pins of the hardware, use the model configuration parameters dialog box to connect these GPIO pins to the eCAP module registers. The eCAP timer captures the time elapsed between two consecutive Hall value changes (0 to 1 or 1 to 0) for a single Hall sensor. You can use this time interval along with the current Hall state (Hall sensor A + Hall sensor B + Hall sensor C) to compute the rotor position and speed values. For example, for Hall sensor A, eCAP module should read eCAP1 register and record two time intervals (T1 and T2) between the Hall value changes that occur during a 360 electrical degree rotation cycle.
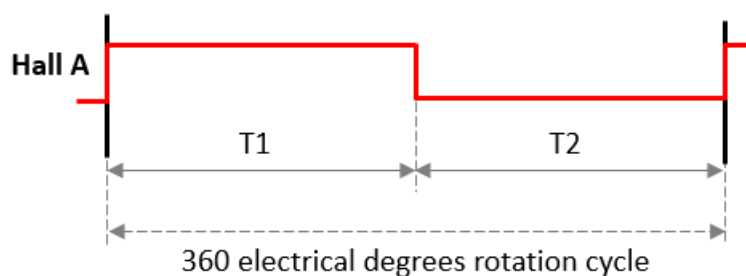
Similarly, eCAP2 and eCAP3 registers should read the Hall values and record the time intervals for Hall sensors 2 and 3 respectively. Use the following procedure to configure the eCAP connections.

In the Simulink model, use the Simulink toolstrip to open **Model Settings** to open the Configuration Parameters dialog box. In the **Target hardware resources** section of the **Hardware Implementation** tab, click the **eCAP** group. Use these fields to assign eCAP registers to the general-purpose I/O (GPIO) pins connected to the Hall sensors:

- **ECAP1 pin assignment** — Select the GPIO pin connected to the Hall sensor A.
- **ECAP2 pin assignment** — Select the GPIO pin connected to the Hall sensor B.
- **ECAP3 pin assignment** — Select the GPIO pin connected to the Hall sensor C.
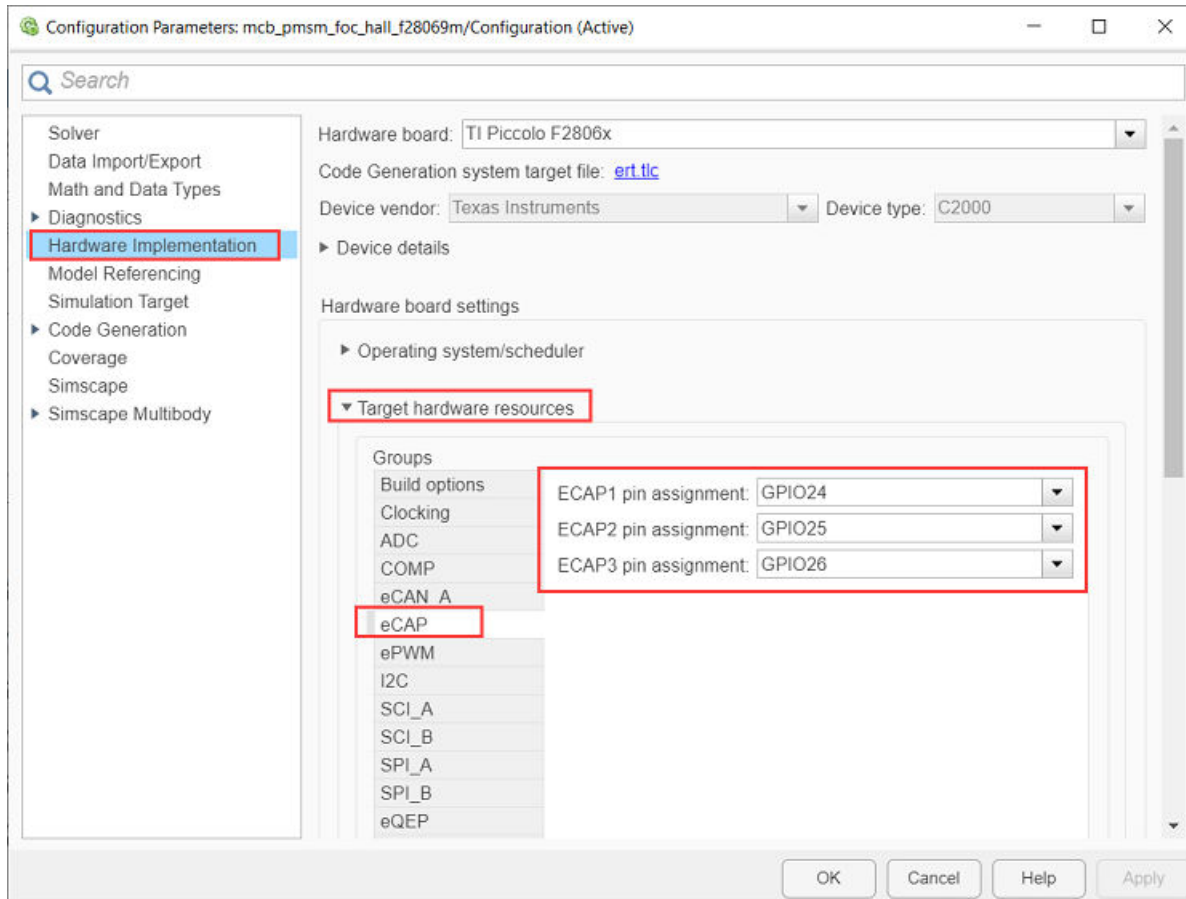
Generate Interrupts for Hall Value Transitions

For each Hall sensor, the eCAP timer should reset and start when the Hall value changes (transitions from 0 to 1 or 1 to 0). A hardware interrupt can trigger this action when a Hall value changes. Use this procedure to generate separate set of hardware interrupts for the three eCAP registers (connected to the three Hall sensors).

1. Use the Simulink browser to add the C28x Hardware Interrupt block from **Embedded Coder Support Package for Texas Instruments C2000 Processors > Scheduling**.
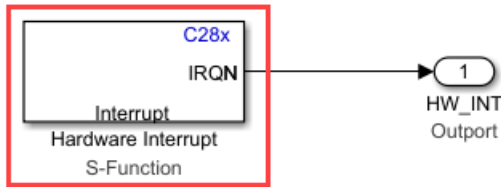
2   Use the C28x Interrupt Block Parameters dialog box to configure the block. Set the **CPU interrupt numbers** and **PIE interrupt numbers** parameters to configure the block to trigger an interrupt for each Hall value change (transition from 0 to 1 or 1 to 0) for every Hall sensor. In addition, this configures the block to interface with the configured eCAP registers to detect a Hall value change.
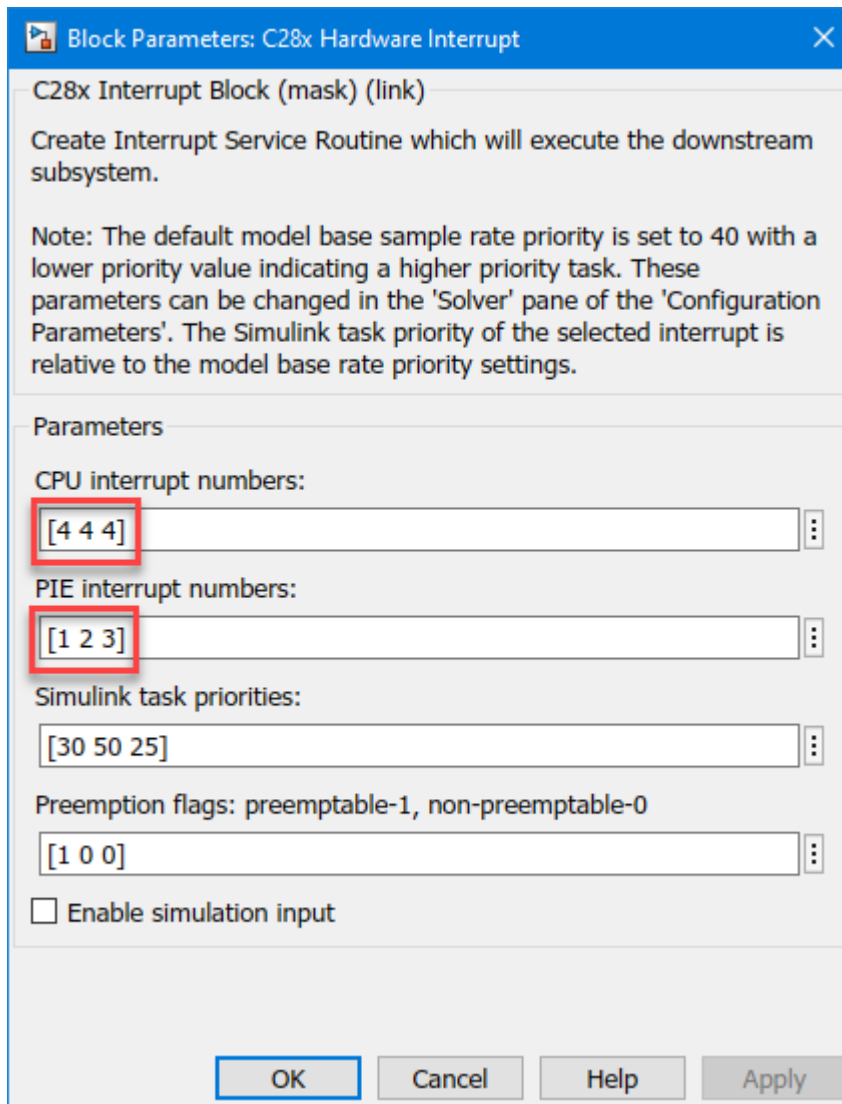
3   Place the C28x Hardware Interrupt block inside a subsystem named **Code Generation** and connect it to a demultiplexer to generate separate hardware interrupt signals for the three Hall sensors.
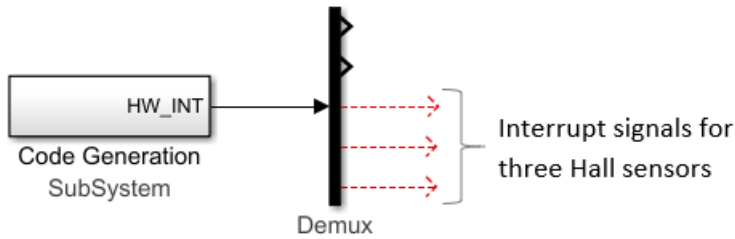
Service Generated Interrupts

Service the hardware interrupt signals so that for each interrupt:

- The eCAP timer (for a Hall sensor) resets and restarts.
- The algorithm captures the current Hall state (Hall sensor A + Hall sensor B + Hall sensor C).
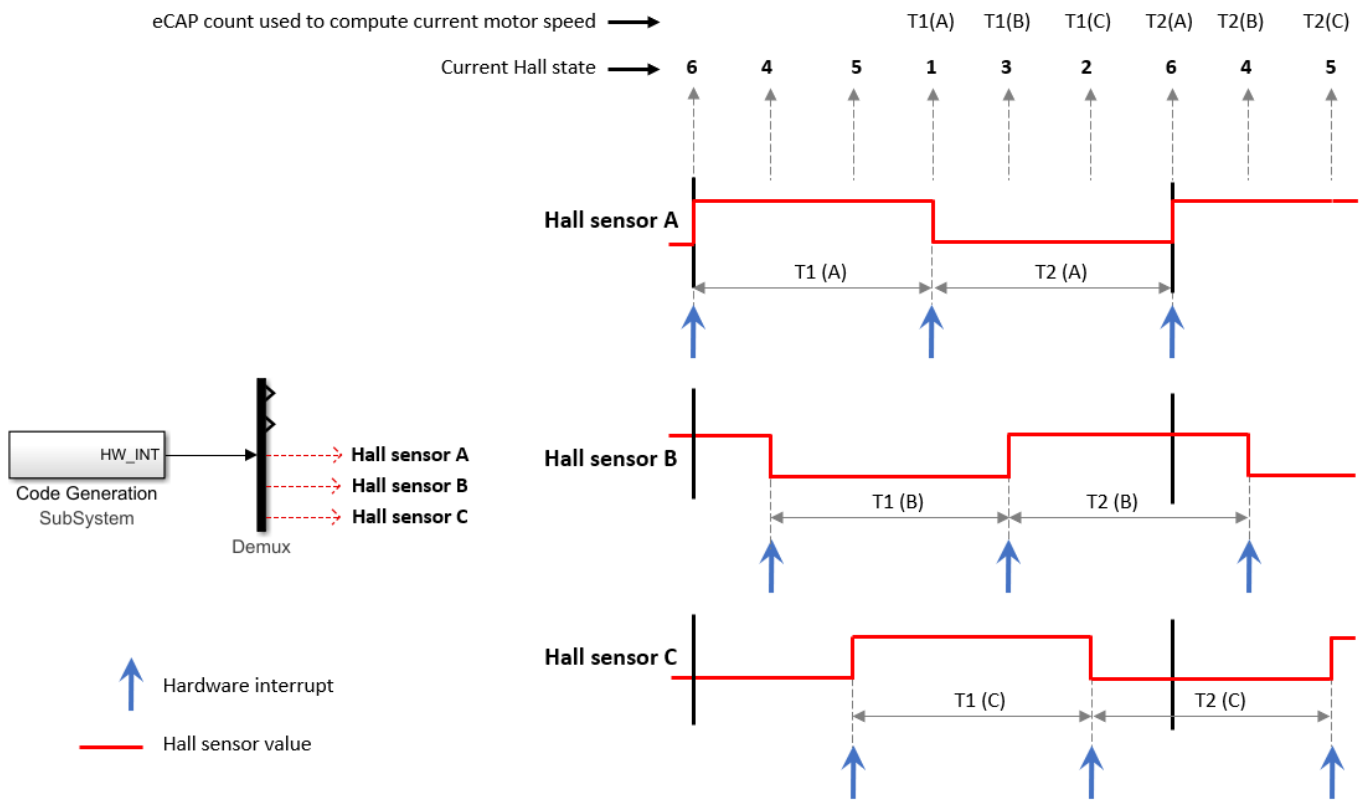
Use this procedure to implement the algorithm to service the hardware interrupts:

1 Use the Simulink browser to add the eCAP block from **Embedded Coder Support Package for Texas Instruments C2000 Processors > C280x**.
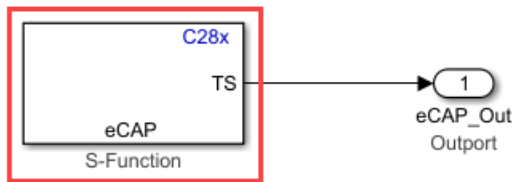
Place this block inside a subsystem named **Code generation**. This subsystem captures the time elapsed (timer count) between two Hall value changes for a single Hall sensor.

2 Use the Simulink browser to add the Hall Validity block from **Motor Control Blockset > Sensor Decoders**. Connect the **Code generation** subsystem to the **Cnt** input port of the Hall Validity block.
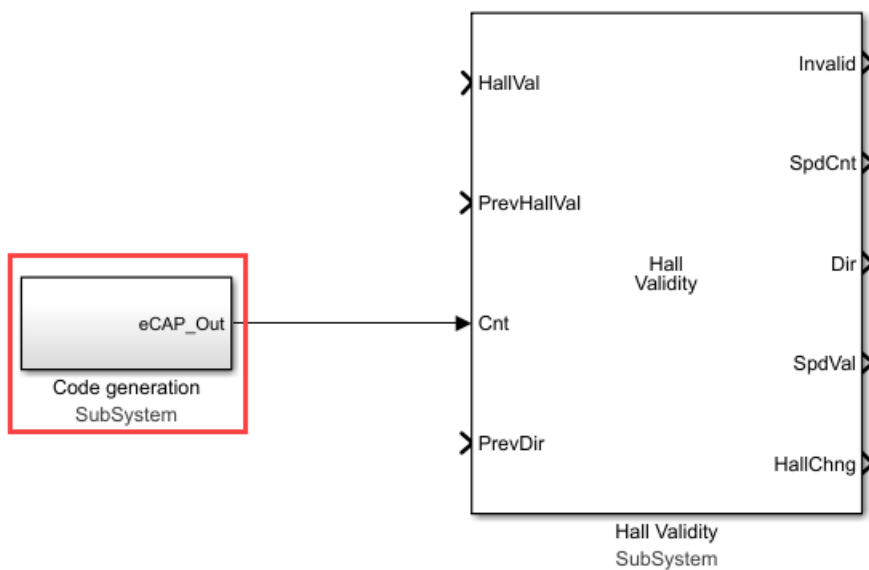
3 Create these global variables (by using Data Store Memory, Data Store Read, and Data Store Write blocks):

- *GlobalHallState* — Stores the Hall state (Hall sensor A + Hall sensor B + Hall sensor C).
- *GlobalDirection* — Stores the direction of the rotor spin (either +1 or –1 indicating positive or negative direction of rotation, respectively).
- *GlobalSpeedCount* — Stores the eCAP timer output.
- *GlobalSpeedValidity* — Stores either 0 (indicating an invalid speed count) or 1 (indicating a valid speed count).
- *HallStateChangeFlag* — Stores 1 (to indicate that a Hall value has changed) or 0 (to indicate that speed and position computation for the previous Hall state (Hall sensor A + Hall sensor B + Hall sensor C) is complete).

Connect these variables to the Hall Validity block as shown in the following figure:
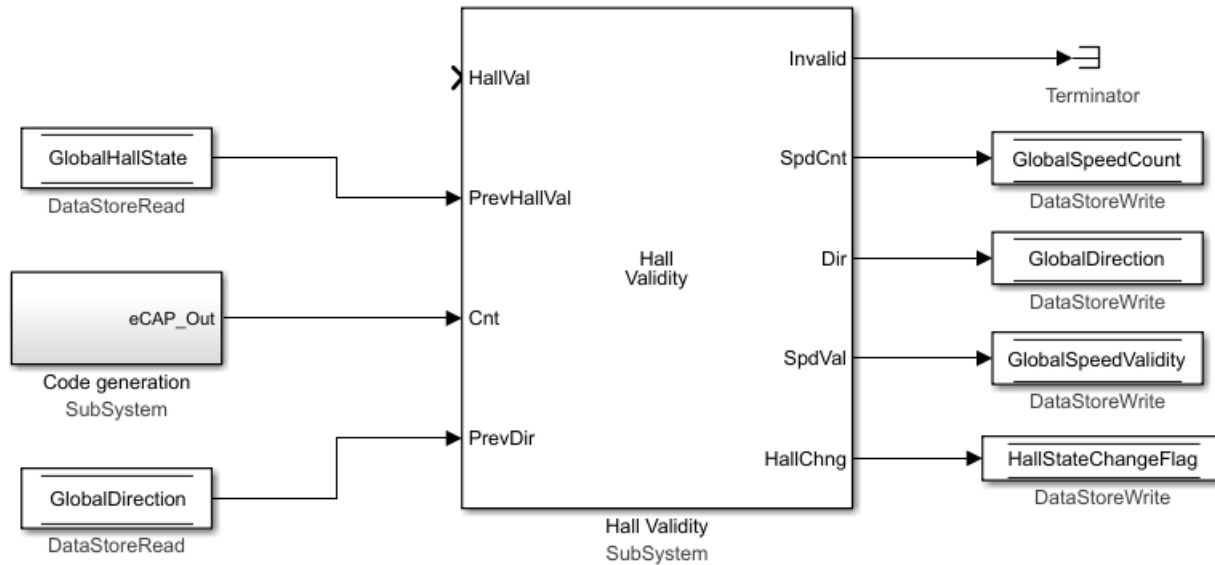
4 Use the Simulink browser to add the Memory Copy block from **Embedded Coder Support Package for Texas Instruments C2000 Processors > Memory Operations**. In the Memory Copy block parameters dialog box, set the **Copy from** parameter to `Specified source code symbol`. Use the **Source code symbol** parameter to specify the variable name available in the source code symbol table that stores the current Hall state.
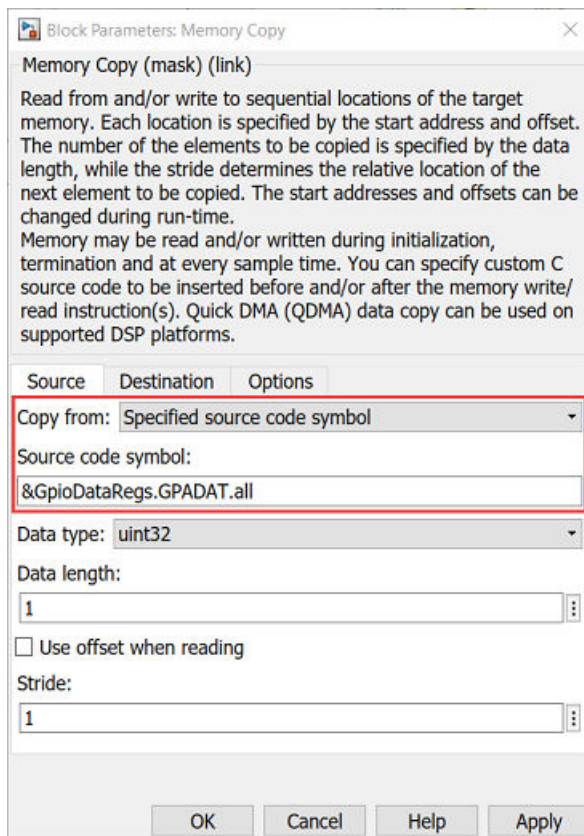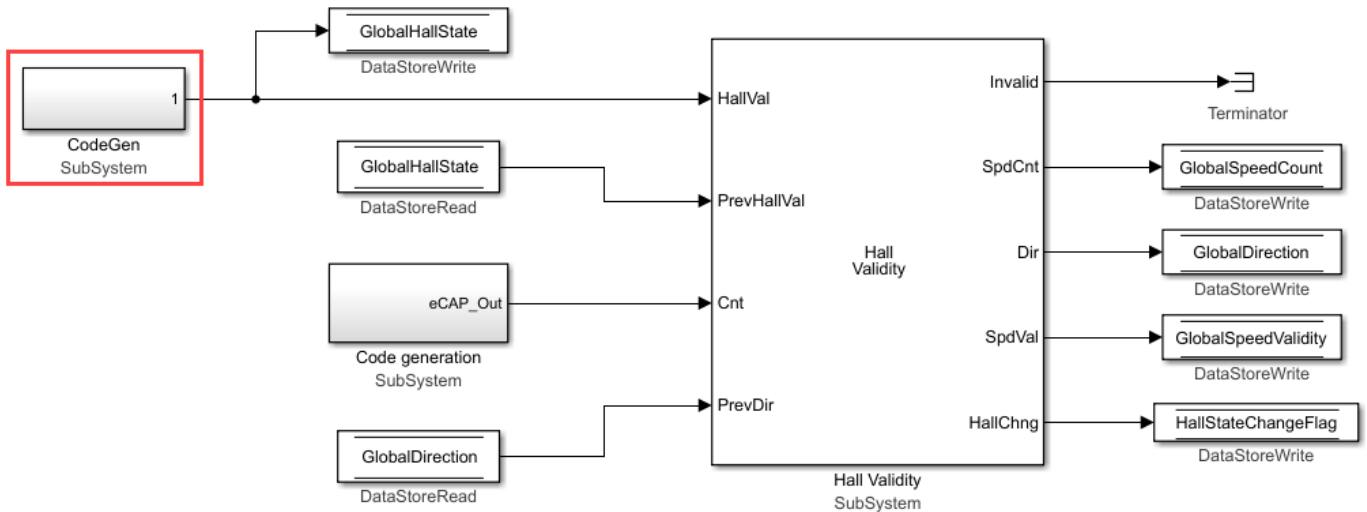
5 Connect the Memory Copy block output to an outport.
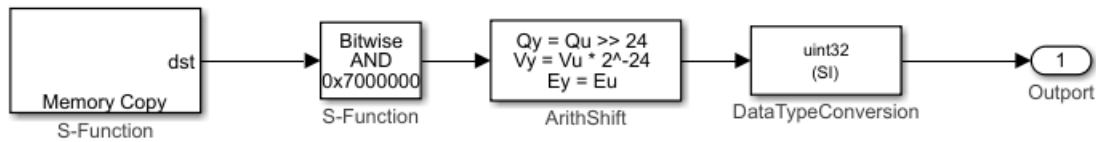
Place these blocks inside a subsystem named **CodeGen**. Therefore, this subsystem outputs the current Hall state.

6 Connect the **CodeGen** subsystem to the Hall Validity block as shown in the following figure:

7 Integrate the Hall Validity block and the entire hardware interrupt service algorithm into a single subsystem named **Hall sensor A**. Add the Trigger block from the **Simulink > Ports & Subsystems** library to this subsystem and set the **Trigger type** parameter to `function-call`. Rename the trigger block as **eCAP1 Interrupt**.
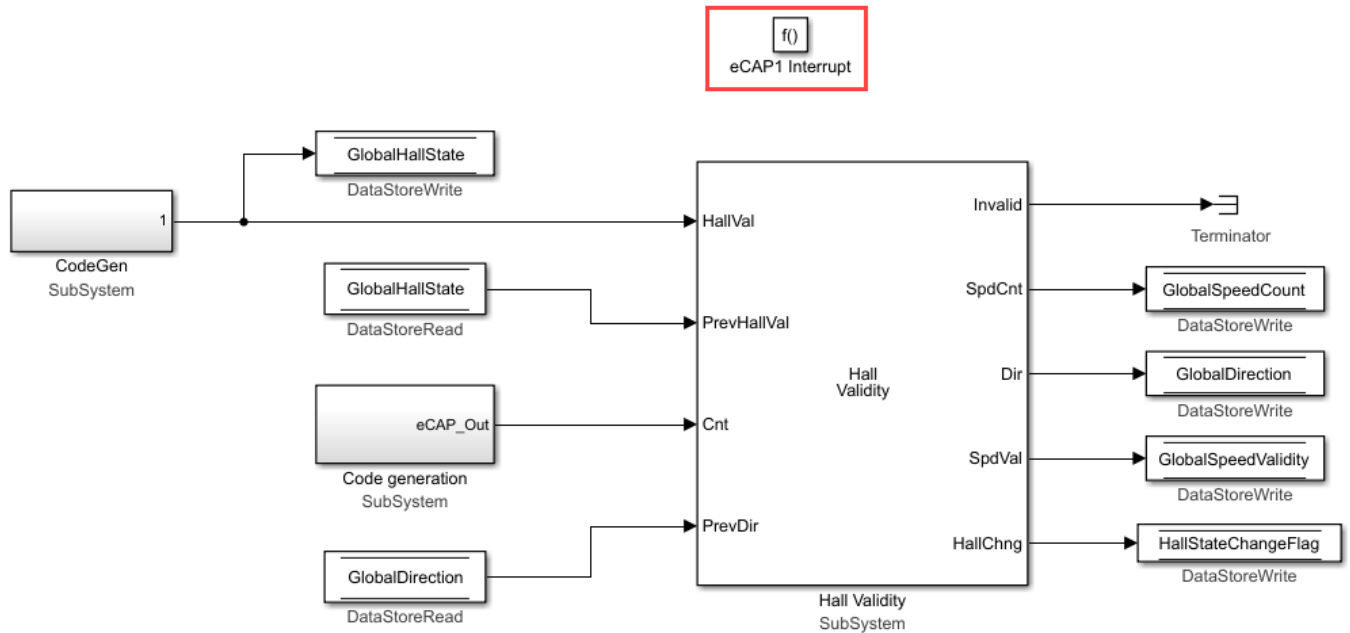
The trigger block acts as the hardware interrupt signal for Hall sensor A. When the **Hall sensor A** subsystem receives an interrupt (indicating a Hall sensor A value change), the hardware interrupt service algorithm resets the eCAP timer to output the recorded timer count (during the previous Hall state) and also captures the current Hall state.

8 Similarly, create the subsystems **Hall sensor B** and **Hall sensor C** containing the hardware interrupt service algorithms for Hall sensors B and C, respectively. Connect these subsystems to the **Code Generation** subsystem that you created in the section Generate Interrupts for Hall Value Transitions.
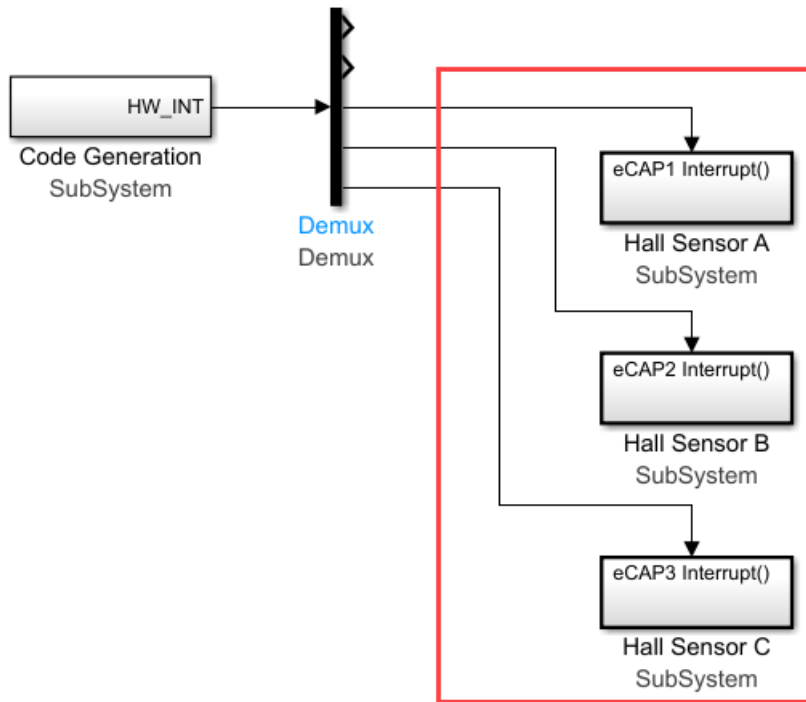
Compute Electrical Position and Mechanical Speed

Use the following procedure to add the position and speed computation algorithm to the Current Control subsystem. For more details, see the model `mcb_pmsm_foc_hall_f28069m.slx` as a reference.

1. Add the algorithm used in steps 4 and 5 of the section Service Generated Interrupts in the current controller. The algorithm enables the current controller to read the current Hall state. Add the Trigger block from the **Simulink > Ports & Subsystems** library to this subsystem and set the **Trigger type** parameter to `function-call`. Place this algorithm in a subsystem.
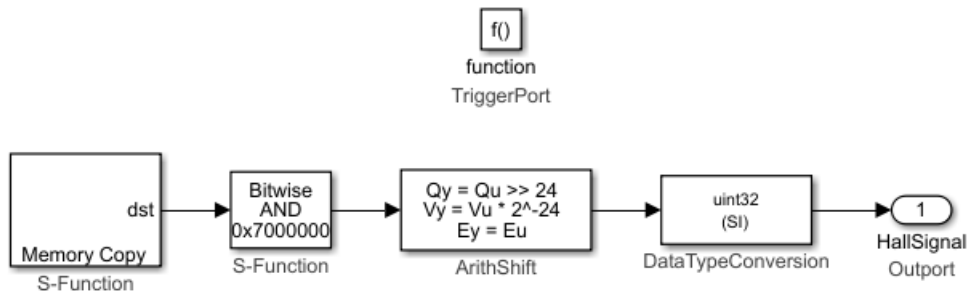

2. Use the Memory Copy blocks (from **Embedded Coder Support Package for Texas Instruments C2000 Processors > Memory Operations**) to read the three eCAP timer counter values. Add the Trigger block from the **Simulink > Ports & Subsystems** library to this subsystem and set the **Trigger type** parameter to `function-call`. Place this algorithm in a subsystem.
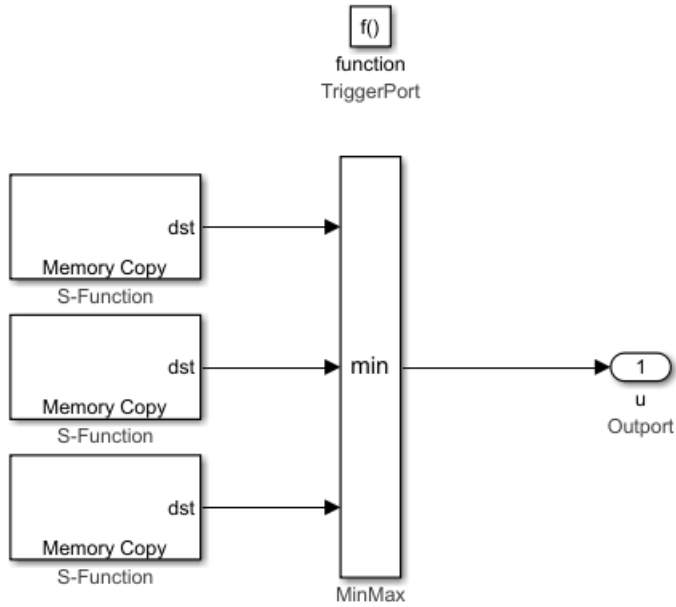
3   Integrate the two function-call subsystems (that you crated in steps 1 and 2), global variables, and an integrity check algorithm for eCAP counter and Hall state values into a subsystem named **Atomic Hall Reading**.
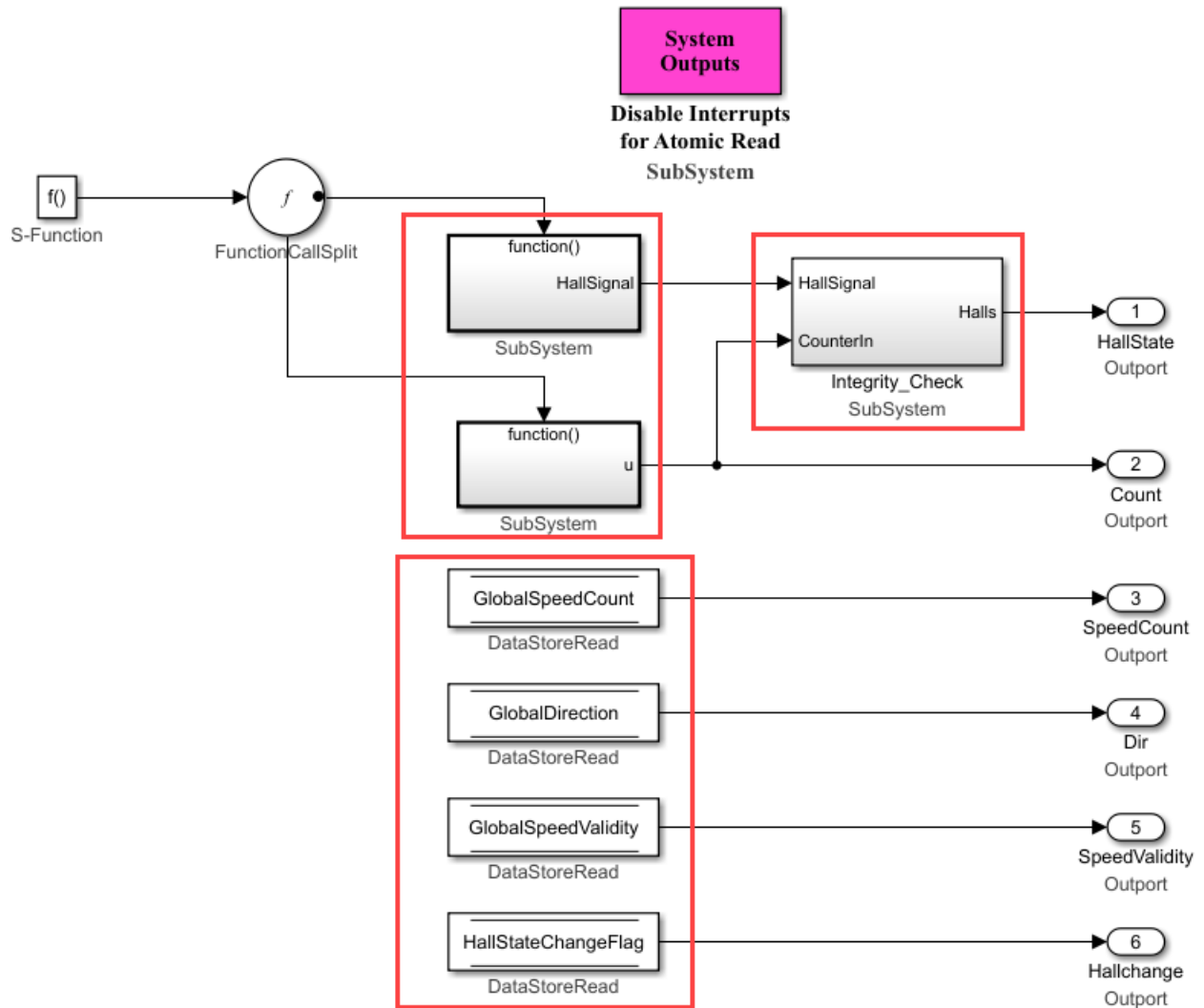
4. Use the Simulink browser to add the Hall Speed and Position block from **Motor Control Blockset > Sensor Decoders**. Connect this block, the **Atomic Hall Reading** subsystem, and the **HallStateChangeFlag** variable as shown in the following figure:
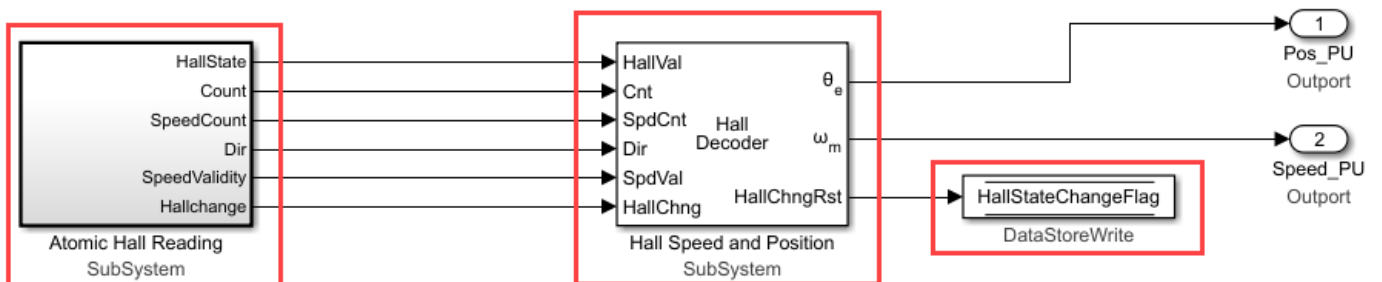
5 Use the Simulink browser to add the Mechanical to Electrical Position block from **Motor Control Blockset > Sensor Decoders**. In the block parameters dialog box enter the variable *pmsm.PositionOffset* for the **Mechanical offset** parameter.
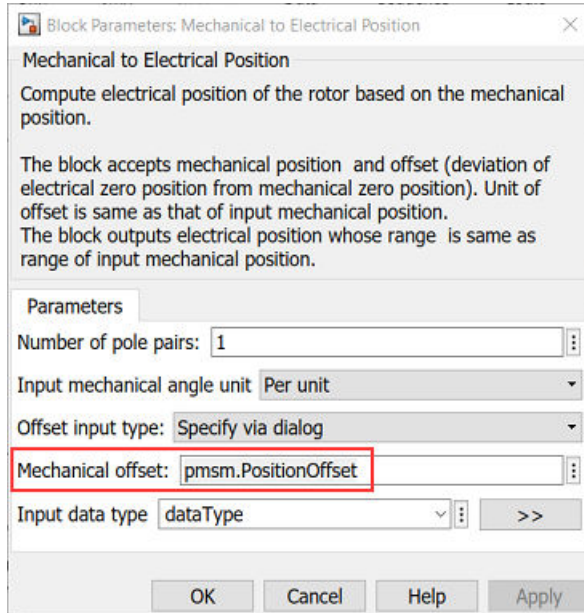
The variable *pmsm.PositionOffset* (available in the model initialization script associated with the reference model `mcb_pmsm_foc_hall_f28069m.slx`) stores the offset value for the Hall sensor. We use the Mechanical to Electrical Position to apply this offset to the computed electrical position value.
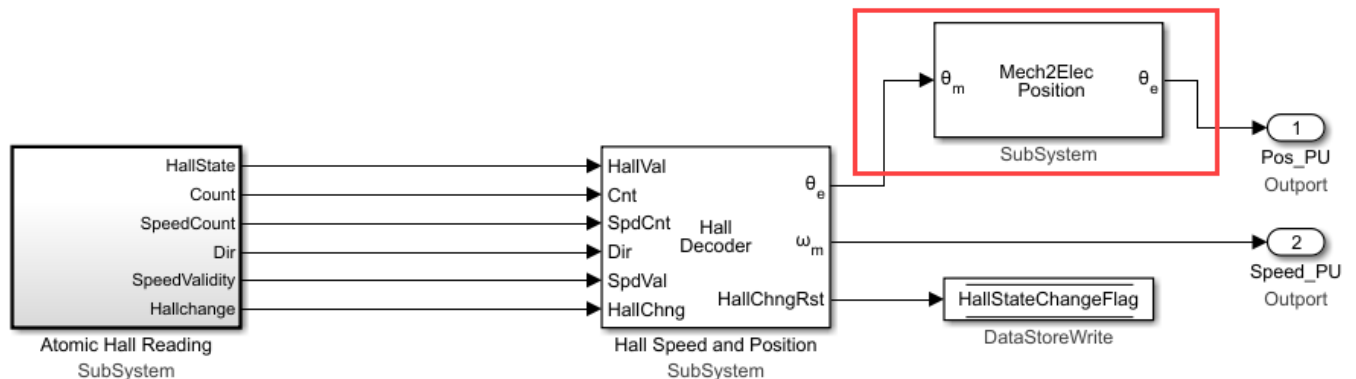
6 Add a glitch filter to the computed speed count value. This filter rejects low values of computed speed count. This enables the motor to run at speeds greater than ten times the base speed.
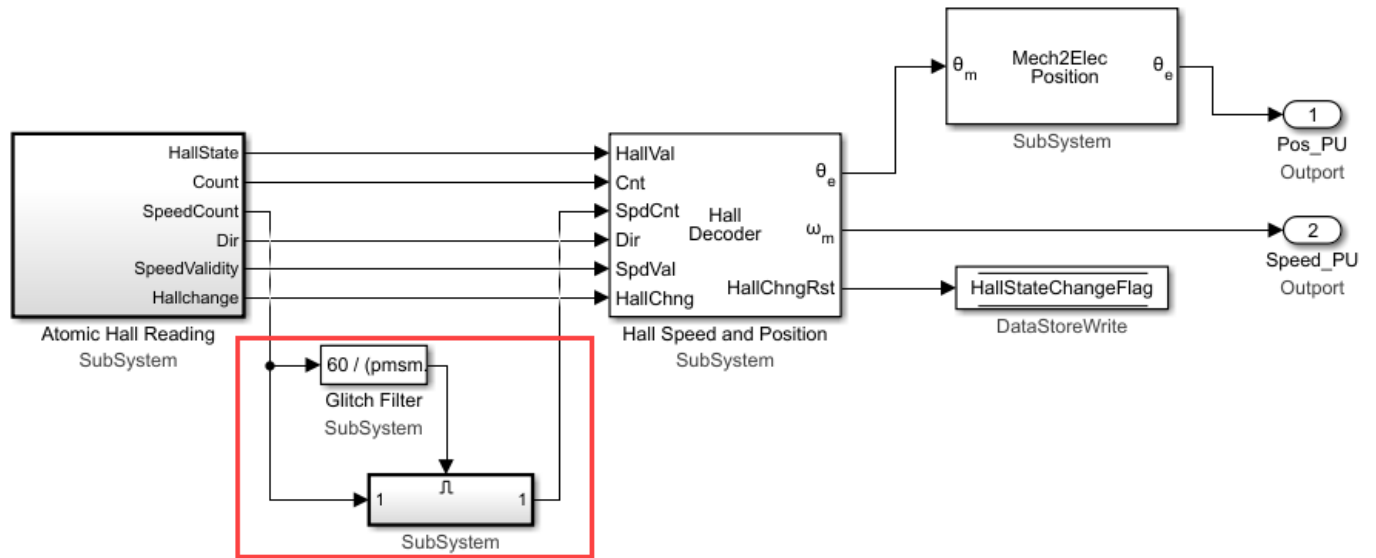
7 Add the resulting algorithm inside the `Current Control/Input Scaling` subsystem of the reference Simulink model. For more details, see `mcb_pmsm_foc_hall_f28069m.slx`.