

Motor Control Blockset™

User's Guide



MATLAB® & SIMULINK®

R2023a



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

Motor Control Blockset™ User's Guide

© COPYRIGHT 2020–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

March 2020	Online only	New for Version 1.0 (Release 2020a)
September 2020	Online only	Revised for Version 1.1 (Release R2020b)
March 2021	Online only	Revised for Version 1.2 (Release R2021a)
September 2021	Online only	Revised for Version 1.3 (Release R2021b)
March 2022	Online only	Revised for Version 1.4 (Release R2022a)
September 2022	Online only	Revised for Version 1.5 (Release R2022b)
March 2023	Online only	Revised for Version 2.0 (Release R2023a)

Design the Controller

1

Design Field-Oriented Control Algorithm	1-2
Design Current and Position Scaling Subsystems	1-3
Design Current Controller Subsystem	1-6
Perform Manual Gain-Tuning of Current Controller	1-10
Design Speed Control Algorithm	1-12
Perform Manual Gain-Tuning of Speed Controller	1-14
PMSM Drive Characteristics and Constraint Curves	1-15

Deploy and Validate System

2

Prepare Target Hardware	2-2
Verify Direction of Rotation of Motor	2-2
Calibrate Current Sensor	2-2
Calibrate Position Sensor	2-3
Add Hardware Drivers to Simulation Model and Deploy to Target Hardware	2-4
Task Scheduling in Target Hardware	2-6
Adding ADC Driver Library Block	2-8
Adding Quadrature Encoder Driver Block	2-11
Add PWM Driver Block	2-14
Add Hardware Interrupt Trigger Block for Current Control Loop	2-18
Run in Open-Loop and Switch to Closed-Loop	2-19
Model Configuration and Hardware Deployment	2-23

Validate System	2-25
Calculate Physical Motor Load in Target Hardware	2-26
Compare Speed Controller Response During Simulation With Target Hardware Results	2-27
Compare Current Controller Response During Simulation With Target Hardware Results	2-29

Plant Modeling

3

Creating Plant Model Using Motor Control Blockset	3-2
Use PMSM Block and Motor Parameters to Design Plant Model	3-3
Add Average-Value Inverter Block	3-5
Create Motor Phase Current Sensing and Signal Conditioning Subsystem	3-6
Create Position Sensing Subsystem	3-7
Add Delay in Plant Model	3-8
Integrate Blocks and Subsystems	3-9

Hardware Troubleshooting

4

Check ADC Inputs	4-2
Description	4-2
Action	4-2
Verify PWM Outputs	4-4
Description	4-4
Action	4-4
Check Hardware Connections	4-6
Description	4-6
Action	4-6
Test Algorithm Design	4-7
Description	4-7
Action	4-7
Check Generated Code	4-8
Description	4-8
Action	4-8

Design the Controller

- “Design Field-Oriented Control Algorithm” on page 1-2
- “Design Current and Position Scaling Subsystems” on page 1-3
- “Design Current Controller Subsystem” on page 1-6
- “Perform Manual Gain-Tuning of Current Controller” on page 1-10
- “Design Speed Control Algorithm” on page 1-12
- “Perform Manual Gain-Tuning of Speed Controller” on page 1-14
- “PMSM Drive Characteristics and Constraint Curves” on page 1-15

Design Field-Oriented Control Algorithm

To implement the speed control algorithm for a motor, perform these tasks:

- Current scaling — Convert current from ADC counts to PU.
- Quadrature encoder position decoding — Read the quadrature encoder position counts and calculate the rotor electrical position.
- Torque control — Implement current control in the d - q axis.
- Speed control — Implement speed control.

These steps help you implement the speed control algorithm for a PMSM using Motor Control Blockset and are related to the model `mcb_pmsm_foc_qep_f28379d` used in the example “Field-Oriented Control of PMSM Using Quadrature Encoder”. They explain the procedure to tune the control parameters for d -axis and q -axis current controllers and the speed controller.

- 1 “Design Current and Position Scaling Subsystems” on page 1-3
- 2 “Design Current Controller Subsystem” on page 1-6
- 3 “Perform Manual Gain-Tuning of Current Controller” on page 1-10
- 4 “Design Speed Control Algorithm” on page 1-12
- 5 “Perform Manual Gain-Tuning of Speed Controller” on page 1-14

In these steps, variables are used to define datatypes and execution times of the current and speed controllers. See the initialization script linked to the example model `mcb_pmsm_foc_qep_f28379d` for details about the variables defined in these steps.

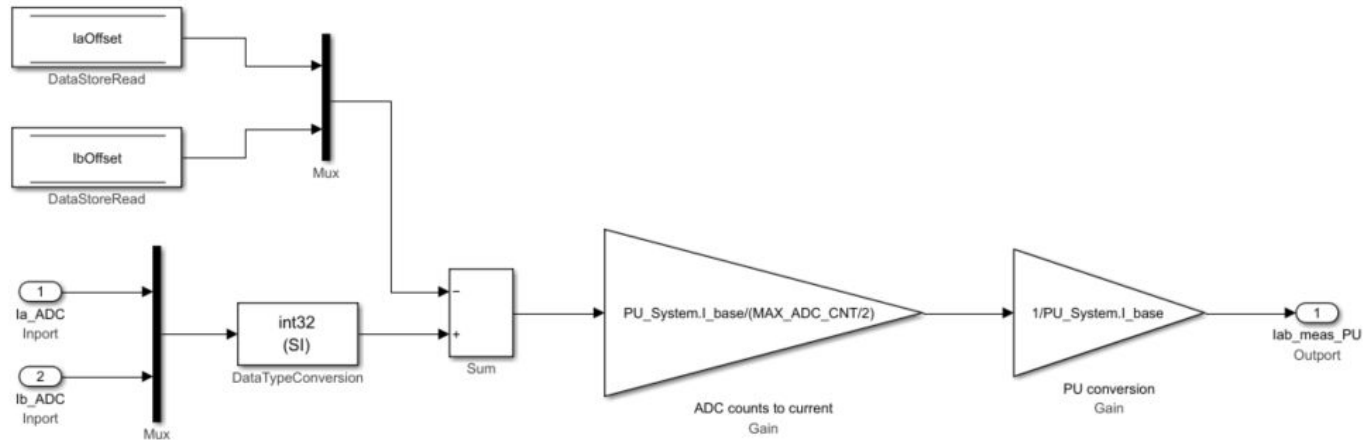
Tip A basic understanding of Simulink® is a prerequisite for following this workflow as these workflow steps do not provide details on tasks like defining a datatype in a constant block or using math operations blocks in Simulink.

See “Estimate PMSM Parameters Using Recommended Hardware” for estimating the motor parameters. Then, see “Creating Plant Model Using Motor Control Blockset” on page 3-2 to design a plant model. This helps you verify the control algorithm in simulation.

Design Current and Position Scaling Subsystems

Use these steps to design the current and position scaling subsystems:

- 1 Create the current scaling subsystem.



This subsystem reads the current in ADC counts and converts it to per-unit (PU) values.

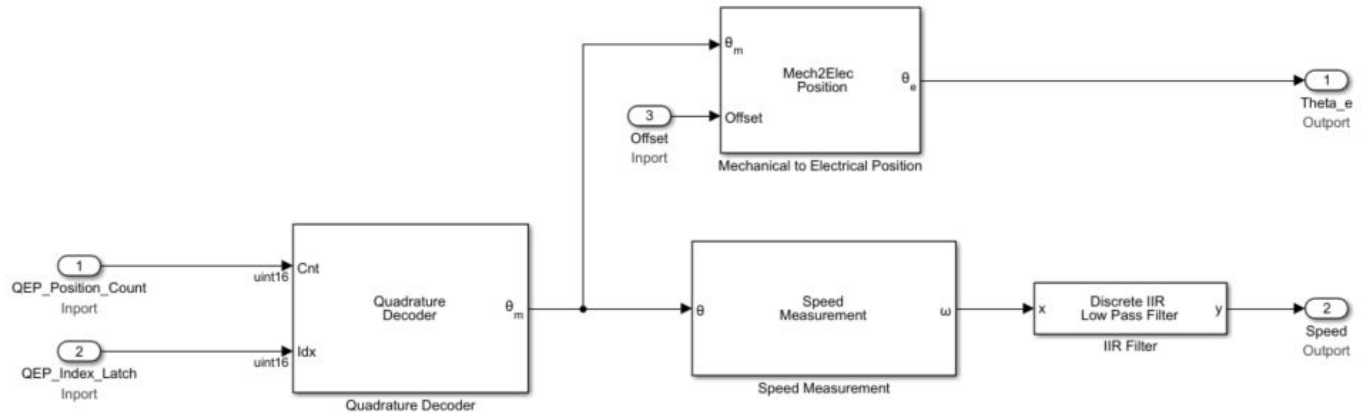
In this subsystem, the *IaOffset* and *IbOffset* Data Store Memory blocks are the ADC offsets for current measurement and they are hardware specific. The file `mcb_SetInverterParameters.m` contains the default ADC offset (*CtSensAOffset* and *CtSensBOffset*) for few commercially available inverters. For details about ADC offset calibration in hardware, see “Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset”.

In this subsystem, the motor phase current measured in ADC counts is converted to current in PU. The `PU_System.I_base` value refers to the base current in this subsystem. For details about the PU system, see “Per-Unit System”. See the `mcb_SetPUSystem.m` file that computes the PU values for the system.

You can use the base values for computing the real-world values from per-unit. To implement the real-world or SI unit values, see the model `mcb_pmsm_foc_qep_f28379d_SIUnit` used in the example “Field-Oriented Control of PMSM Using SI Units”.

The *IaOffset* and *IbOffset* Data Store Memory blocks are used to share data between the current and position subsystems.

- 2 Create the position scaling subsystem.

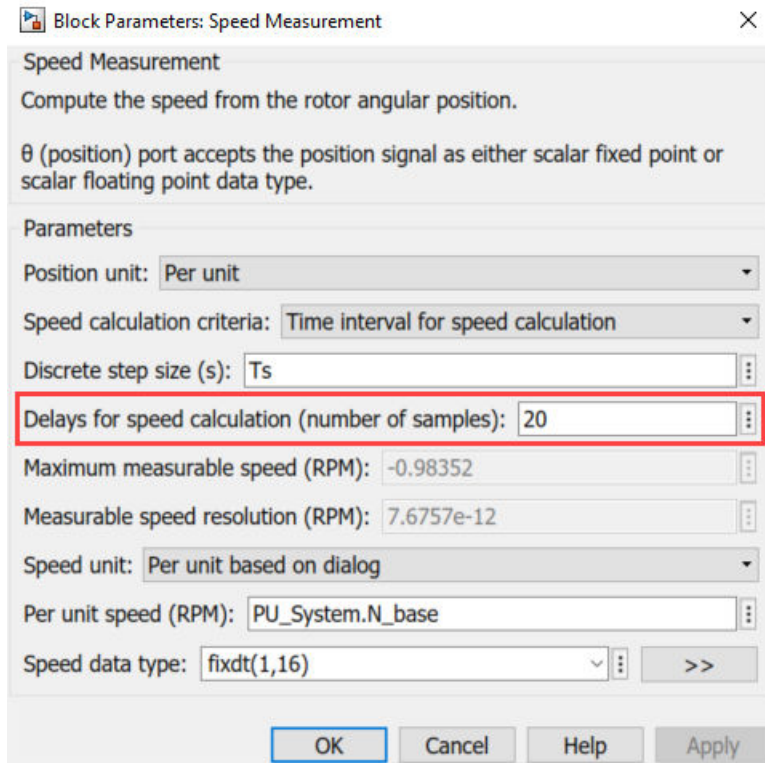


This subsystem reads the rotor position from the QEP pulse count.

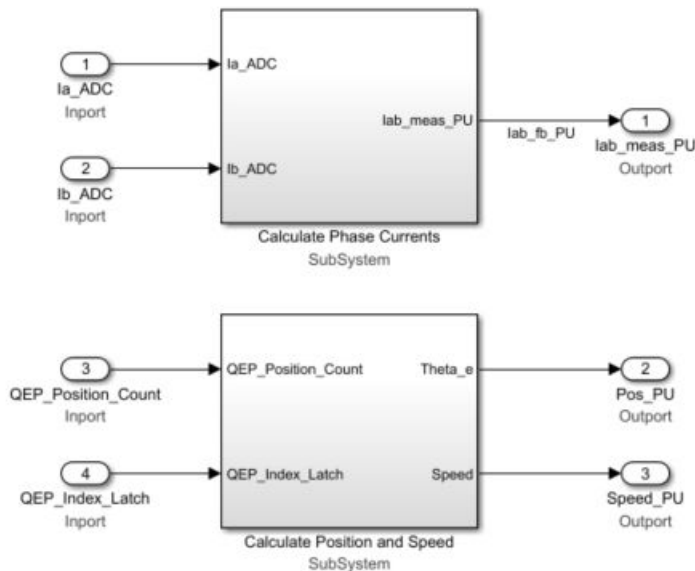
In this subsystem, the Quadrature Decoder block reads the position count from the plant model or hardware driver block. The block converts the rotor mechanical position in encoder position counts to rotor mechanical angle in PU ($\theta-1$).

The Mechanical to Electrical Position (Mech2Elec Position) block adjusts the rotor mechanical angle for QEP offset and converts it to electrical angle. The FOC algorithm needs the rotor electrical angle to run the motor. To calculate the QEP encoder offset, see “Quadrature Encoder Offset Calibration for PMSM Motor”.

The Speed Measurement block calculates speed from the rotor position. In the Speed Measurement block parameters dialog box, set the **Delays for speed calculation (number of samples)** parameter to 20. We selected the value 20 in this workflow so that the block can measure the maximum speed of the motor that is under test. The Speed Measurement block outputs the speed in PU.



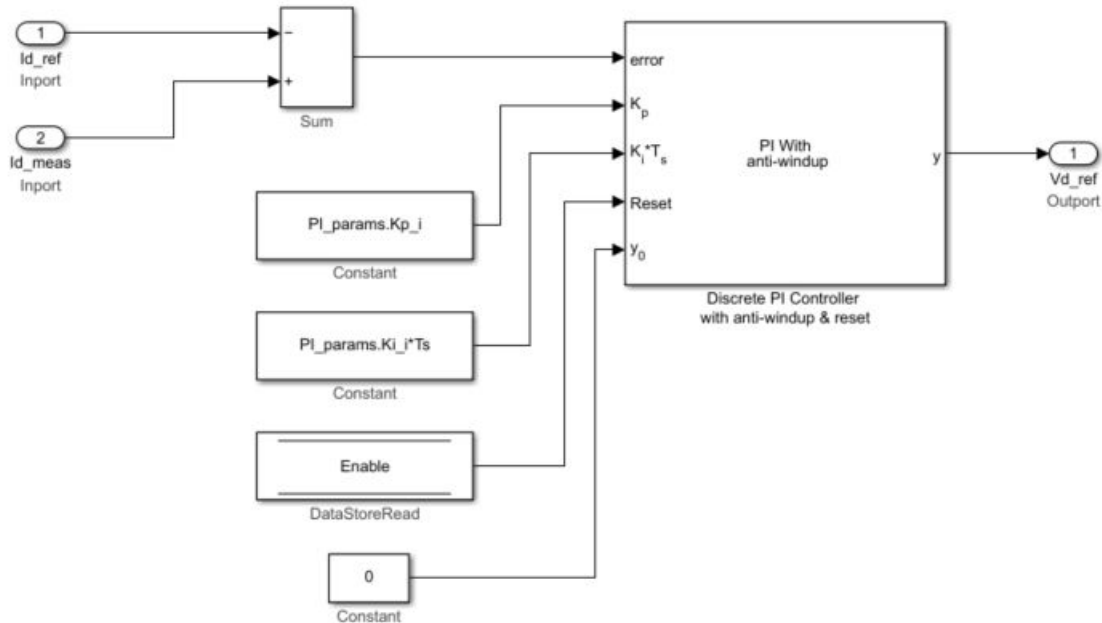
The resulting two subsystems (Calculate Phase Currents and Calculate Position and Speed) contain the current scaling and position decoding logic.



Design Current Controller Subsystem

Use these steps to design the current controller subsystem:

- 1 From the Motor Control Blockset library in the Simulink Library Browser, use the Discrete PI Controller with anti-windup & reset block (in the Controls/Controllers library) to design the d -axis and q -axis current control. For example, this image shows the d -axis current controller subsystem.



The MATLAB[®] function `mcb_SetControllerParameters` (in the model initialization script) calculates the PI control gains for the d -axis and q -axis current controller and the speed controller. For details about calculation of the controller gains, see “Estimate Control Gains and Use Utility Functions”. For example, see the model initialization script file `mcb_pmsm_foc_qep_f28379d_data.m` (used in the example “Field-Oriented Control of PMSM Using Quadrature Encoder”) that uses a sampling time (T_s) of 50 μ s.

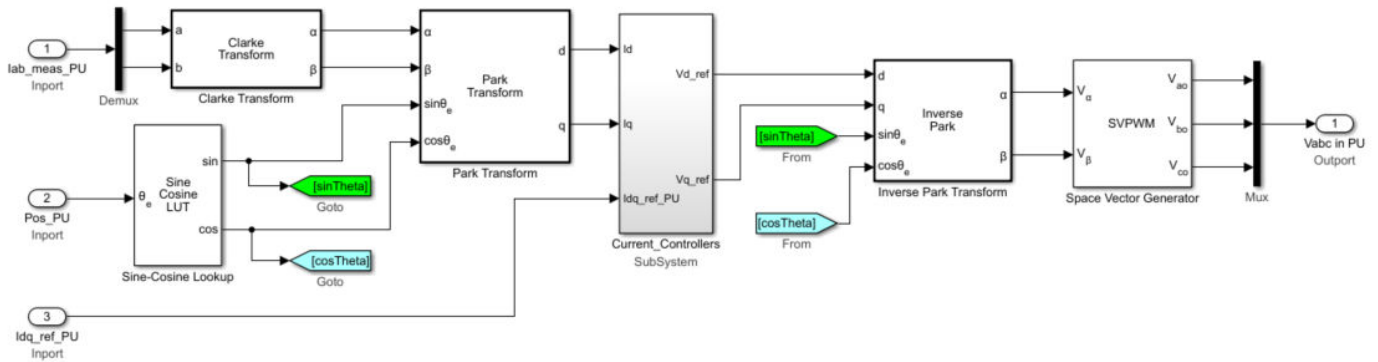
In the subsystem diagram, the *Enable* variable is a Data Store Memory used to reset the controller. Adding *Enable* variable is optional.

The subsystem also uses three constant blocks with these values:

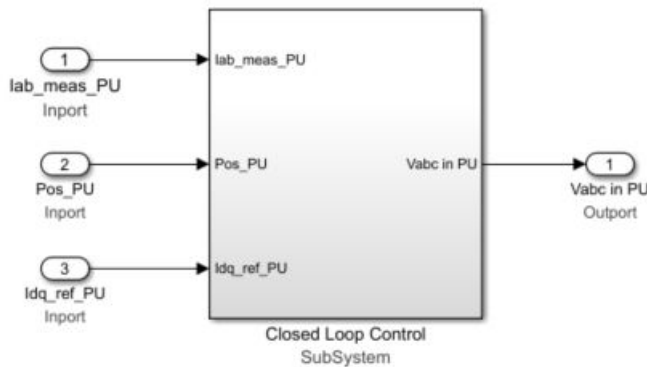
- $PI_params.Kp_i$
- $PI_params.Ki_i*Ts$
- 0

Create a similar subsystem for the q -axis current PI controller. Integrate the subsystems for d -axis and q -axis PI controllers into a single subsystem (Current_Controllers) that controls the d -axis and q -axis currents.

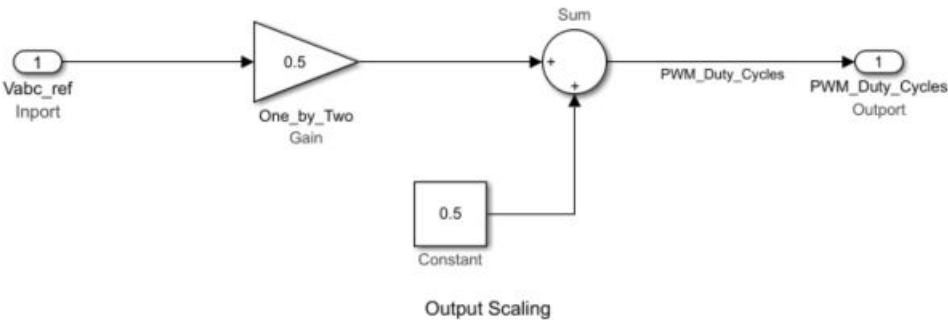
- 2 Add the Clarke Transform, Park Transform, Inverse Park Transform, and Space Vector Generator blocks from the Motor Control Blockset/Controls/Math Transforms library to the Current_Controllers subsystem (that you created in step 1) as shown in this figure.



- Integrate the components that you created in step 2 into a single subsystem (Closed Loop Control that implements closed loop field-oriented control) as shown in this figure.

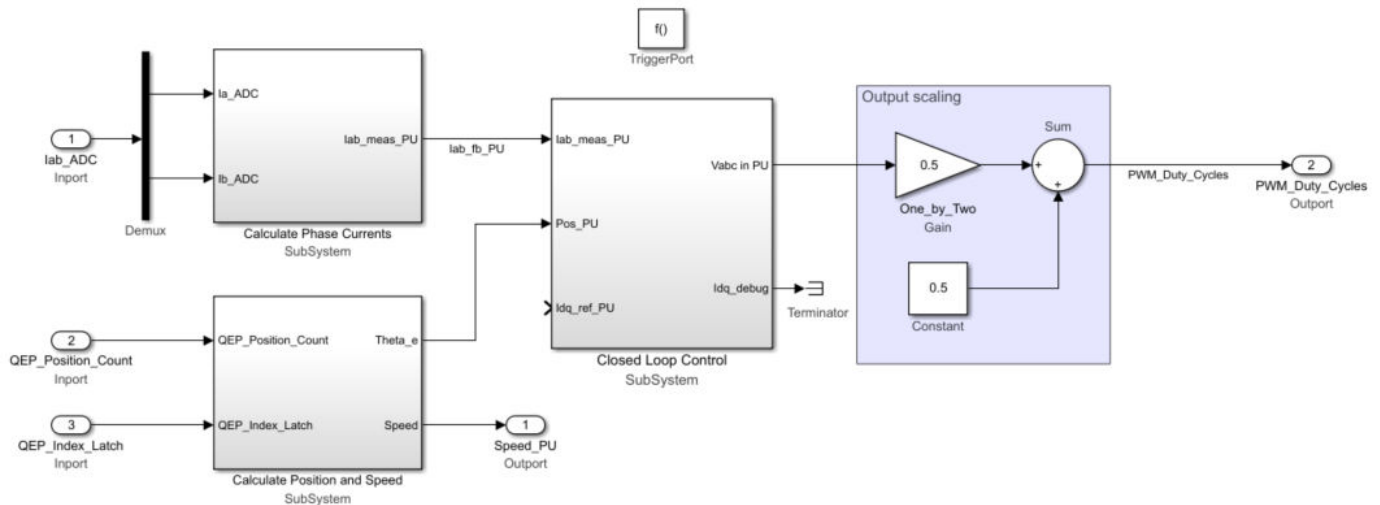


- Create an Output Scaling subsystem to scale the Pulse Width Modulation (PWM) outputs. This subsystem outputs the normalized PWM duty cycles (0-1) for the plant model.

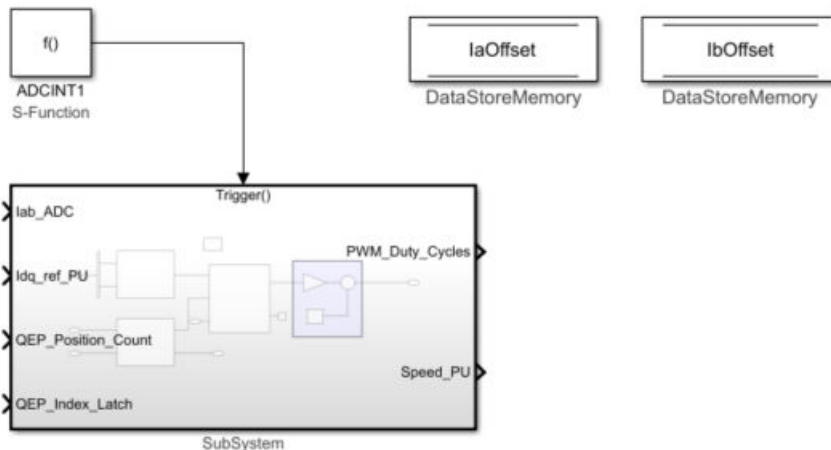


- Create a new subsystem by integrating the current scaling (Calculate Phase Currents), QEP position decoding (Calculate Position and Speed), Closed Loop Control, and Output Scaling subsystems. Add the Trigger block from the Simulink/Ports & Subsystems library to this subsystem and set the **Trigger type** parameter to function-call.

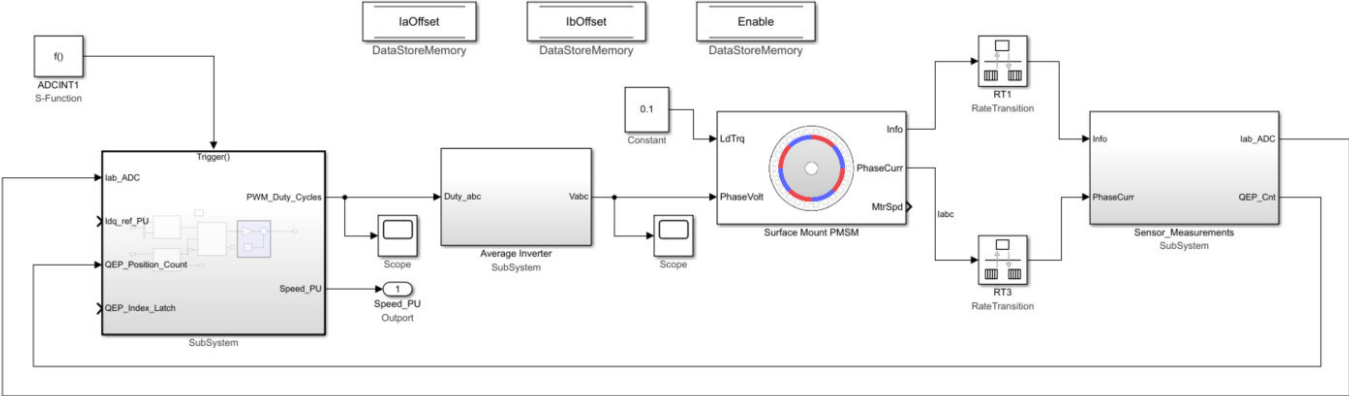
1 Design the Controller



- 6 Add a Function-Call Generator block from the Simulink/Ports & Subsystems library to the subsystem created in step 5. Set the **Sample time** parameter of the block to equal the control-loop sample time, T_s (that has a default value of $50e-6$ s).



- 7 Integrate the plant model and the controller subsystem that you created in step 6. For detailed steps on how to create a plant model for a motor control system, see “Creating Plant Model Using Motor Control Blockset” on page 3-2.

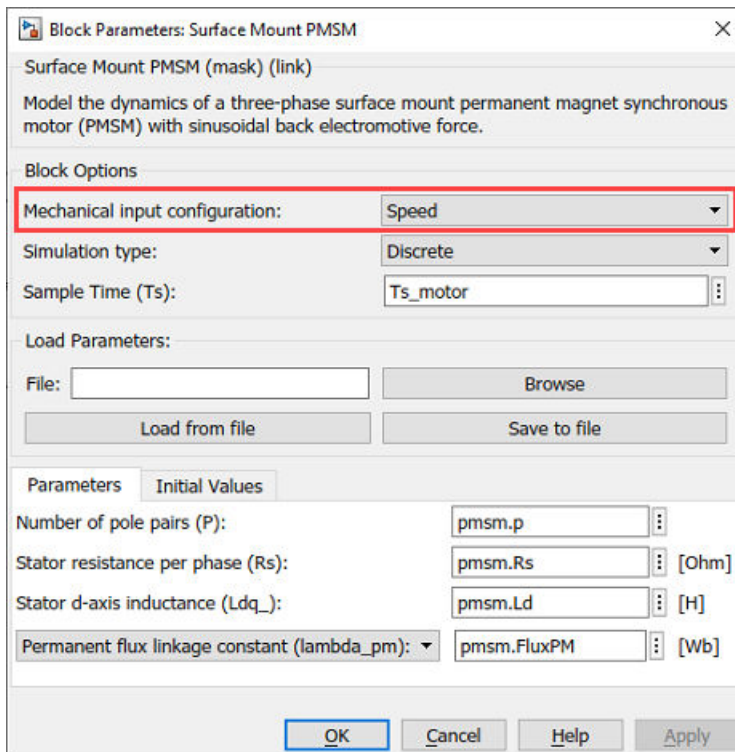


Perform Manual Gain-Tuning of Current Controller

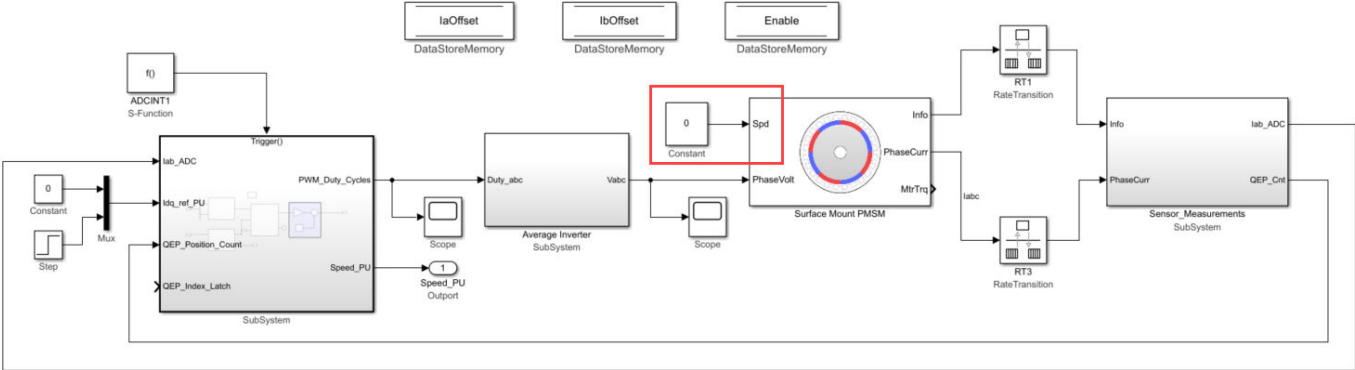
This step shows you how to manually tune the gains of the d -axis and q -axis current controllers. This step is optional, however you can use it to tune the control gain parameters.

The procedure includes adding a step change to the I_{d_ref} current and analyzing the current controller performance using the step response of the I_{d_meas} current to tune the d -axis current controller. It explains a similar process for the I_{q_ref} current to tune the q -axis current controller.

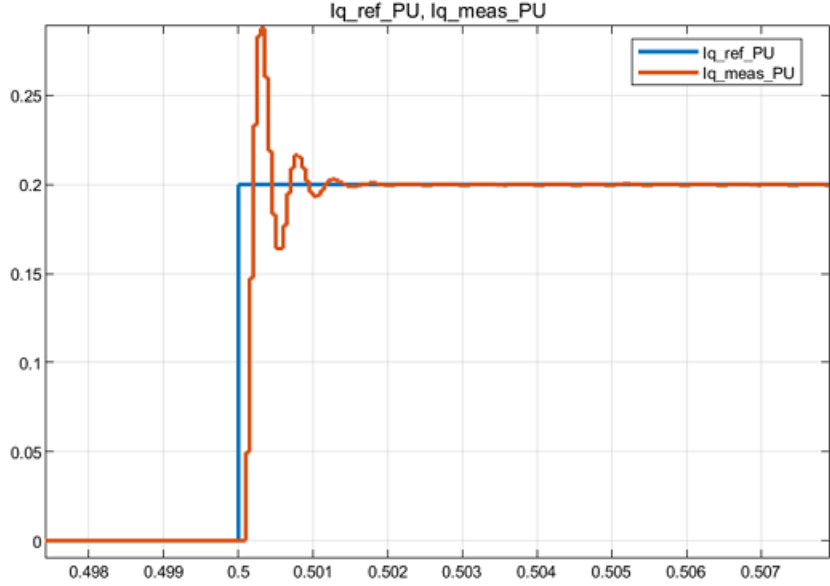
Before starting the manual tuning procedure, you should lock the rotor in the plant model to ensure that the motor does not run when you provide a step change to I_{d_ref} or I_{q_ref} currents. In the Surface Mount PMSM block parameters dialog box, set the **Mechanical input configuration** parameter to **Speed**. Set the **Spd** input (of the Surface Mount PMSM block) to θ to ensure that the rotor is locked.



The integrated plant and controller subsystem simulation model enables you to manually tune the gains of the current controllers. Provide a step input to I_{q_ref} in the range (0 to 0.2) PU and observe the measured I_{q_meas} current feedback. Adjust the control parameters of the q -axis current controller to meet your control objectives.



Simulate the model and plot the $I_{q_ref_PU}$ and $I_{q_meas_PU}$ current signals and analyze the step response. This helps you to tune the control parameters for the q -axis current controller to meet the control objectives.



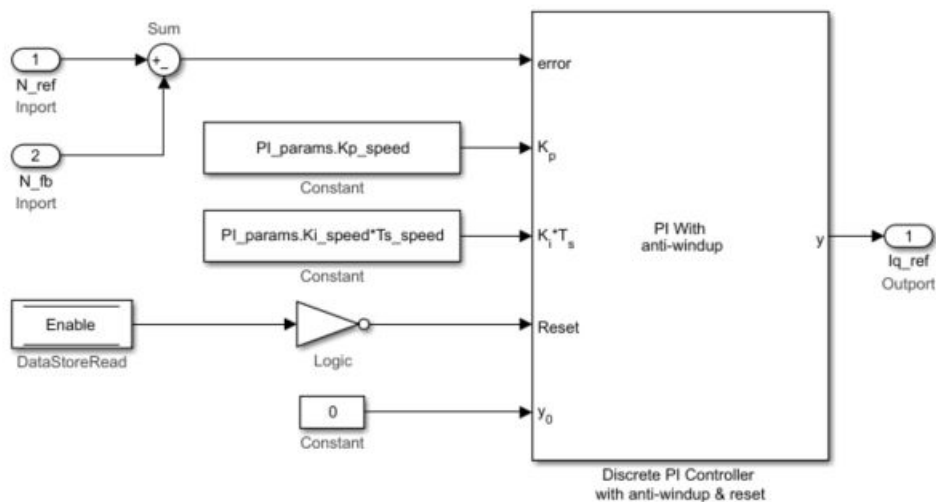
Follow the same procedure for the I_d_ref current to tune the d -axis current controller. After tuning both current controllers, set the **Mechanical input configuration** parameter, in the Surface Mount PMSM block parameters dialog box, back to Torque.

Design Speed Control Algorithm

Use these steps to design a speed control algorithm:

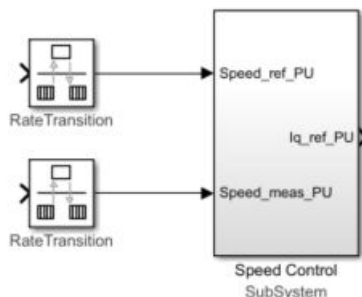
- 1 Create a speed controller subsystem. The current controller subsystem that you created earlier uses the I_{q_ref} current output of the speed controller subsystem as an input.

To create a speed controller subsystem, open the Simulink Library Browser and select the Discrete PI Controller with anti-windup & reset block from the Motor Control Blockset/Controls/Controllers library.



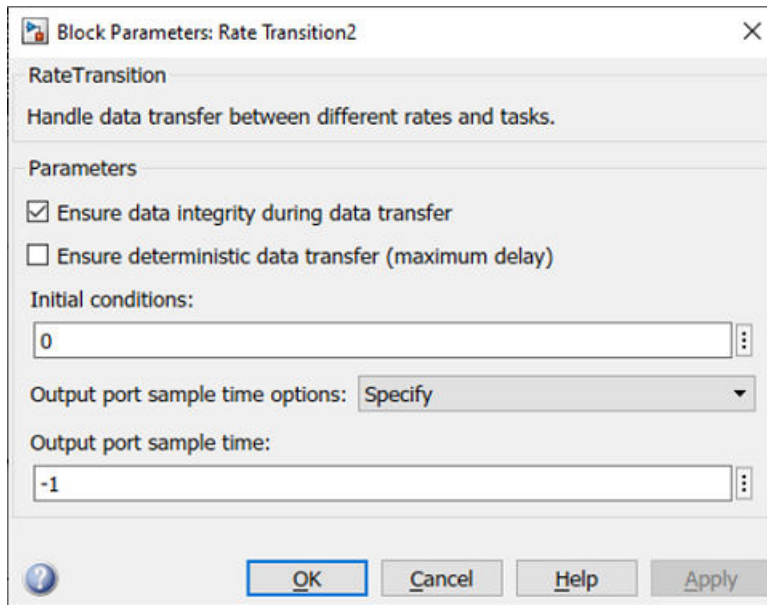
The MATLAB function `mcb_SetControllerParameters` (in the model initialization script) calculates the PI control gains for the d -axis and q -axis current controller and the speed controller. For details about calculation of the controller gains, see “Estimate Control Gains and Use Utility Functions”. For example, see the model initialization script file `mcb_pmsm_foc_qep_f28379d_data.m` (used in the example “Field-Oriented Control of PMSM Using Quadrature Encoder”) that uses a sampling time (T_{s_speed}) of 500 μ s. Optionally, you can use the **Enable** Data-Store Memory block to reset the controller.

- 2 Create a subsystem for the speed controller and add Rate Transition blocks (from the Simulink/Signal Attributes library) to the subsystem inputs with a sample time of T_{s_speed} (execution time of the speed control loop).

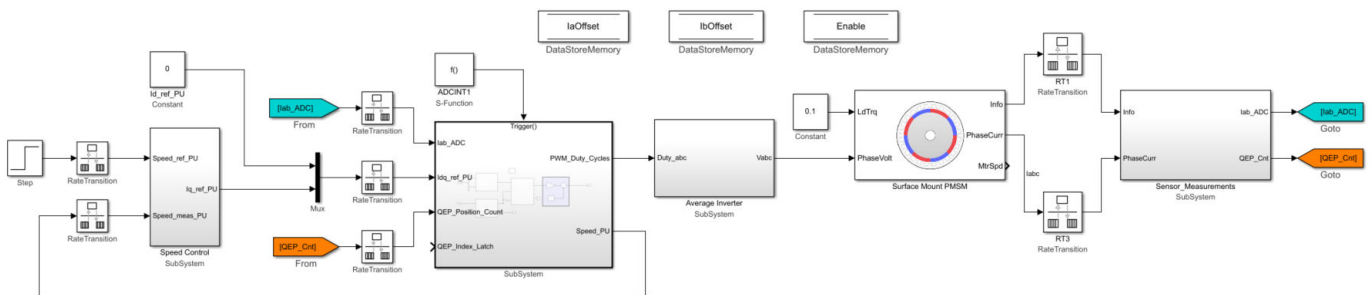


- 3 Integrate the speed controller subsystem (that you created in step 2) with the integrated current controller and plant model subsystems. Connect the $I_{q_ref_PU}$ output port of the speed

controller subsystem to the current controller subsystem input port through a Rate Transition block. The Rate Transition block is needed because the two ports execute at different sample rates. This figure shows an example of the parameter settings of the Rate Transition block connected to the speed controller and the current controller subsystems.



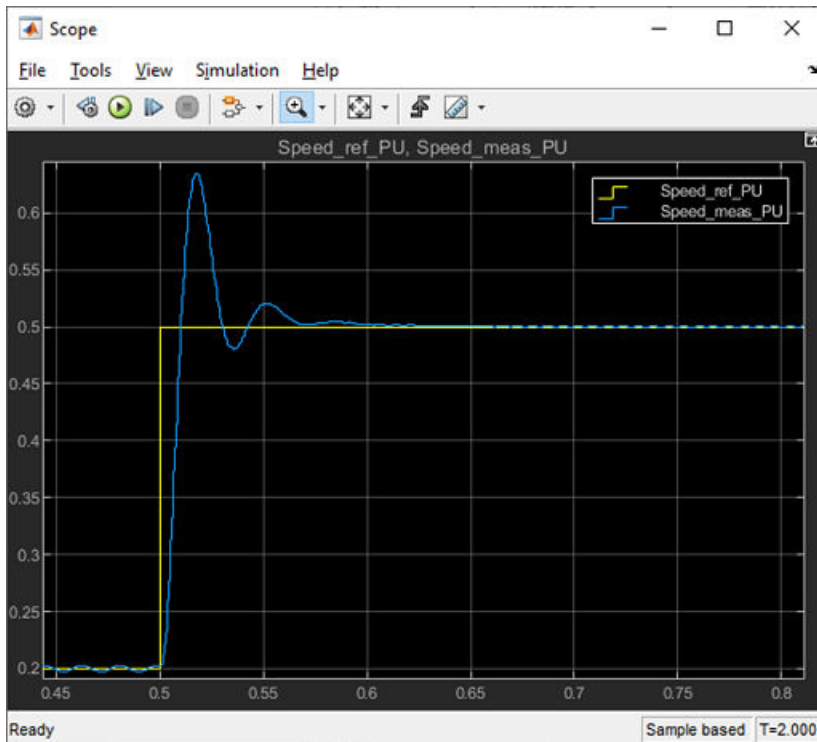
This figure shows the integrated speed controller, current controller, and plant model subsystems.



Perform Manual Gain-Tuning of Speed Controller

To manually tune the speed controller subsystem, provide a step input (in the range 0.2 to 0.5 PU) to the Speed_ref_PU input in the speed controller subsystem (Speed Control). Monitor the measured speed step response Speed_meas_PU and adjust the speed controller subsystem parameters to meet your control objectives.

This figure shows the step response of the speed controller.



This procedure shows a method to implement speed control for a PMSM in simulation. Run the simulation and analyze the controller performance.

You can generate C code from this control algorithm using Embedded Coder[®]. In addition, you can deploy this code and the hardware drivers to the target hardware.

PMSM Drive Characteristics and Constraint Curves

This example shows how to use the PMSM characteristic plotting and PMSM milestone speed identification functions to obtain a control trajectory.

PMSM Drive Characteristics and Constraint Curves

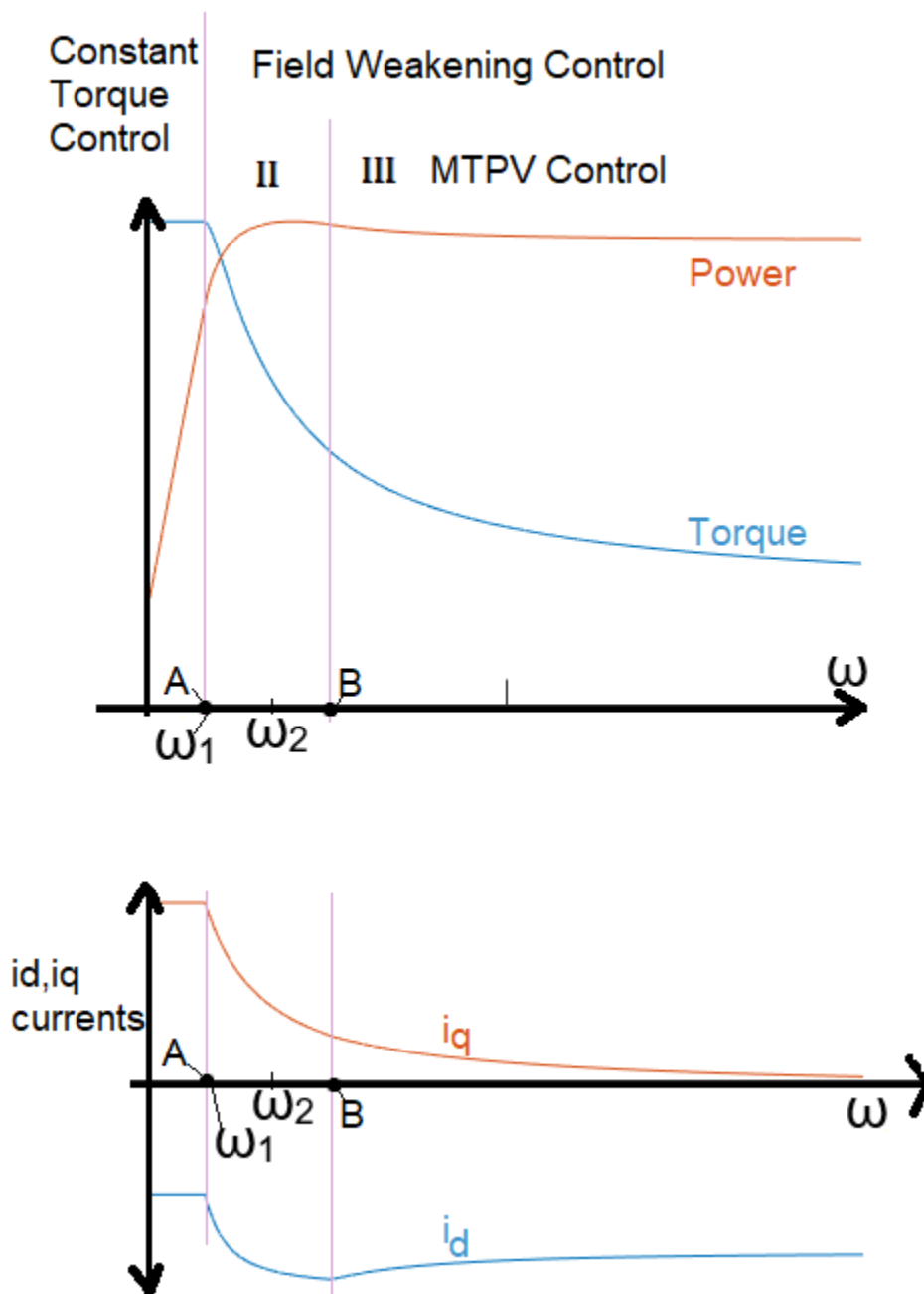
The permanent magnet synchronous motors (PMSMs) come in different configurations.

- When the permanent magnets (PMs) are mounted on the surface of the rotor, the motor is called a surface-mounted PMSM (SPMSM).
- When the PMs are embedded inside the rotor below the surface, the motor is called an interior PMSM (IPMSM).

Various parameters affect whether you select an SPMSM or IPMSM for a given application.

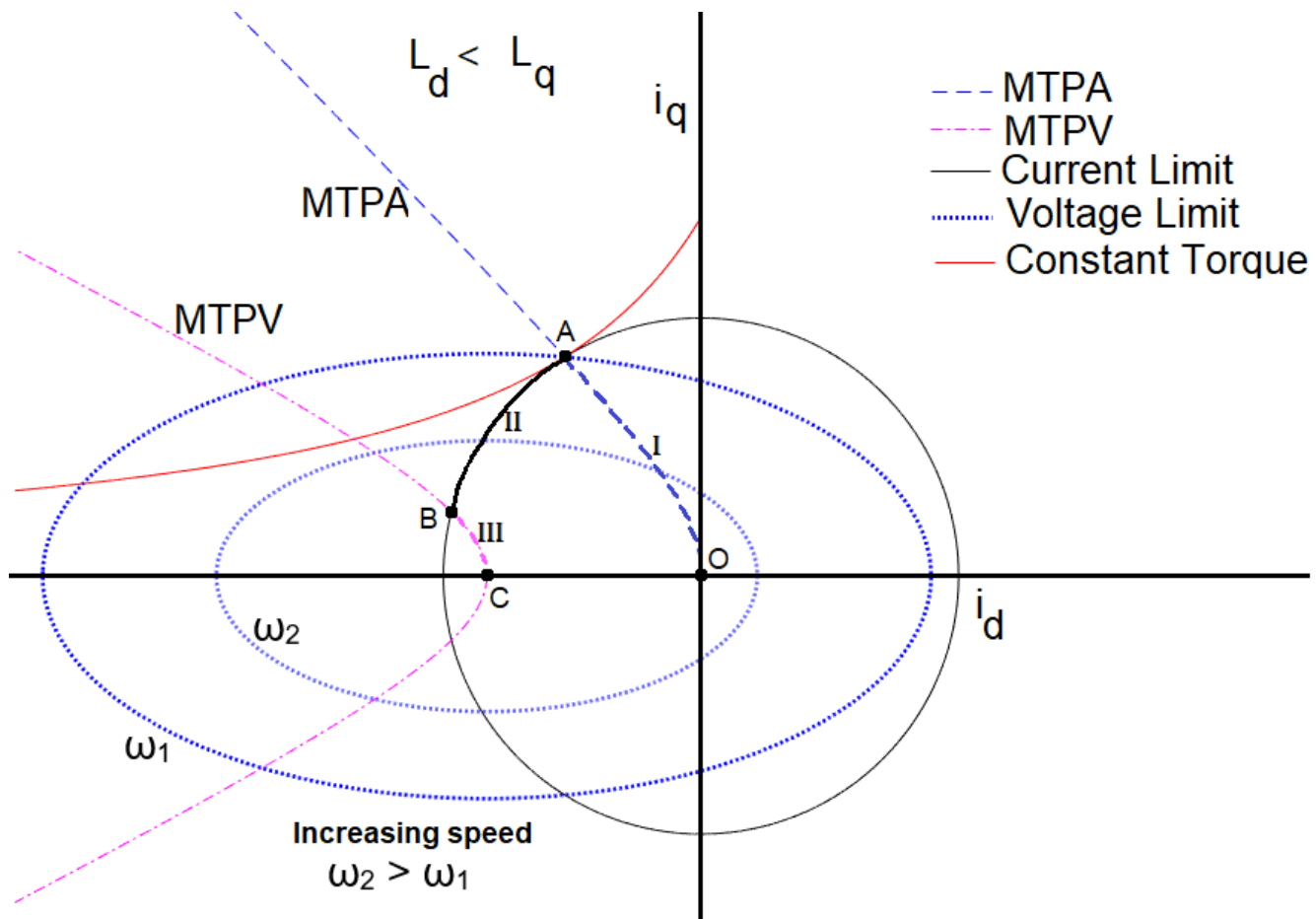
In the d - q rotor reference frame for PMSMs, the d -axis is the axis parallel to the rotor's magnetic orientation and the q -axis is the perpendicular axis, leading the d -axis. An SPMSM has equal d -axis and q -axis inductances. An IPMSM has a higher q -axis inductance than its d -axis inductance. This helps the IPMSM to utilize the reluctance torque in addition to the magnetic torque.

The corner speed of a PMSM is the speed at which the torque-vs-speed (drive characteristics) curve of the PMSM changes shape (has a corner) for a given current limit. When you plot the drive characteristics for rated torque (at rated current), the corner speed is also known as the rated speed or base speed.



You draw the constraint curves for a PMSM to understand the possible operational area and boundaries.

- Maximum torque per ampere (MTPA) trajectory of an IPMSM provides the highest torque for the allowed stator current.
- Maximum torque per voltage (MTPV) trajectory of an IPMSM provides the highest torque while satisfying the stator voltage constraint.
- Voltage limit curve (ellipse for an IPMSM or circle for SPMSM) is centered around a point that is called the characteristic current. This curve shrinks with increasing rotor speed.



These figures show three segments corresponding to the motor operation. The motor operates in each segment as follows.

Segment I

- To begin the operation, the motor increases the torque from zero to the rated torque.
- The operating point shifts from point O to A along the MTPA trajectory (line segment OA). This corresponds to the i_d and i_q values at the tip of MTPA.
- The motor speed ω_1 , at which the voltage limit ellipse touches the MTPA curve and the current limit circle, is the rated speed. This is the first milestone speed of the characteristic curve.

Segment II

- At the operating point A, the control reaches the maximum possible current, and the field weakening strategy begins.
- The current trajectory moves along the current limit circle while maintaining the current constraint from point A to B (line segment AB).
- The operating point B is the speed at which the voltage ellipse touches the current limit circle and the MTPV curve. This is the second milestone speed of the characteristic curve.

Segment III

- Beyond the point B, the voltage ellipse shrinks further and the MTPV trajectory provides the optimal torque until the maximum speed is reached at the operating point C (line segment BC).
- The theoretical maximum speed for a friction-free motor with MTPV is infinite. Practically, the frictional components in the motor limit the maximum speed to a finite value. In Motor Control Blockset™ software, the maximum speed, including the friction, is when the frictional torque balances the motor torque produced at that speed. The practical speed limit of the motor is usually lower than this maximum speed value. Several factors regulate this limit, including mechanical stability, lower efficiency at higher speeds, and so on.
- The calculated maximum speed is the third milestone speed. In some motor configurations, MTPV is not possible. In such cases, the frictional torque limits the operating point B, and there is no operating point C. Such motors have two milestone speeds.

Additionally, note that in these figures:

- The drive characteristic curves show the operating points O, A, and B. These curves do not show the operating point C because it is far away from the remaining points.
- The operating point A is also at the rated speed ω_1 . The second speed ω_2 for the curve is higher than rated speed, but lower than the speed at operating point B. The speed at the operating point B is also the speed at which the current trajectory changes direction from following the current limit circle to following the MTPV trajectory.
- The drive characteristics do not show segment I because the motor maintains torque at maximum value, and the i_d and i_q values remain at values corresponding to the rated torque.

This table shows the functions used in this example.

Function name	Required inputs	Optional arguments	Expected output
mcbPMSMSpeeds	pmsm, inverter	voltageEquation, constraintCurves, FWCMMethod, outputAll, verbose	Milestone speeds in rpm. Matrix of d and q axis currents at milestone speeds (in electrical rad/s)
mcbPMSMCharacteristics	pmsm, inverter	voltageEquation, driveCharacteristics, constraintCurves, FWCMMethod, opacity, idqExternal, speed, torque, imax	Drive characteristics plot Constraint curves plot

The pmsm and inverter arguments are structures. This table shows the fields in each structure.

Structure	Field	Description	Units
pmsm	p	Pole pairs	-
	Rs	Per phase resistance	Ohm
	Ld	D-axis inductance	Henry
	Lq	Q-axis inductance	Henry
	FluxPM	Permanent magnet flux linkage	Weber
	B	Viscous damping coefficient	Newton.Meters/(radians/second)
	I_rated	Rated current of the pmsm	Amperes
inverter	V_dc	DC bus voltage	Volts

Maximum and Milestone Speeds of PMSM

To get all the milestone speeds of the PMSM, use the `mcbPMSMSpeeds` function. The operable speeds are different for different field weakening control (FWC) strategies. The default FWC method is set to voltage current limited maximum torque (VCLMT) control, also called optimal current vector control. The function returns the speed values in rpm.

Set the fields for the `pmsm` and `inverter` structures and calculate the milestone speeds.

Run this section

```
%inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305'); % Use an inverter structure from a ter
inverter.V_dc=24;
pmsm      = mcb_SetPMSMMotorParameters( 'BLY171D' );
milestone_speeds=mcbPMSMSpeeds(pmsm,inverter);
disp(milestone_speeds);

    5393
    9373

pmsm.I_rated=8;% Assuming a higher I_rated to be able to get MTPV trajectory
[milestone_speeds]=mcbPMSMSpeeds(pmsm,inverter);
disp(milestone_speeds);

    2342
    2958
    24369
```

Speed Milestones of PMSM with Different FW Control Methods

Calculate speed milestones for different field weakening methods. When you set the `verbose` option to 1, each command displays the milestone speeds with the description.

Set the fields for the `pmsm` and `inverter` structures and calculate the milestone speeds.

Run this section

```
inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305');
pmsm     = mcb_SetPMSMMotorParameters( 'BLY171D' );
verbose=0; % setting the verbose=1 will also print the messages

milestone_speeds=mcbPMSMSpeeds(pmsm,inverter, 'verbose', verbose, 'FWCMethod', 'vcLmt');
disp(milestone_speeds);
```

```

5393
9373

milestone_speeds=mcbPMSMSpeeds(pmsm,inverter,'verbose',verbose,'FWCMethod','cvcp');
disp(milestone_speeds);

```

```

5393
7008
8361

```

```

milestone_speeds=mcbPMSMSpeeds(pmsm,inverter,'verbose',verbose,'FWCMethod','cccp');
disp(milestone_speeds);

```

```

5393
7008

```

```

milestone_speeds=mcbPMSMSpeeds(pmsm,inverter,'verbose',verbose,'FWCMethod','none'); % no id cont
disp(milestone_speeds);

```

```

5393
6226

```

i_d and i_q at Different Milestone Speeds for Different FW Control Methods

Calculate i_d and i_q values at the speed milestones for different field weakening methods. Each command outputs the array with columns for each milestone speed, and each row of the column provides i_d , i_q , and speed (in electrical rad/s).

Set the fields for the pmsm and inverter structures and calculate currents and milestone speeds.

Run this section

```

inverter = mcb_SetInverterParameters('BoostXL-DRV8305');
pmsm     = mcb_SetPMSMMotorParameters('BLY171D');

```

```

[milestone_speeds]=mcbPMSMSpeeds(pmsm,inverter,'FWCMethod','vclmt','outputAll',0);
[outputArray]=mcbPMSMSpeeds(pmsm,inverter,'FWCMethod','vclmt','outputAll',1);
disp(outputArray);

```

```

1.0e+03 *

      0   -0.0018
0.0018   0.0004
2.2591   3.9262

```

```

disp(round([milestone_speeds(1) outputArray(3,1)*(60/(2*pi))/pmsm.p;    milestone_speeds(2) outp

```

```

5393      5393
9373      9373

```

```

outputArray=mcbPMSMSpeeds(pmsm,inverter,'FWCMethod','cvcp','outputAll',1);
disp(outputArray);

```

```

1.0e+03 *

      0   -0.0011  -0.0018
0.0018   0.0014   0.0003
2.2591   2.9355   3.5023

```



```
outputArray=mcbPMSMSpeeds(pmsm,inverter,'FWCMethod','cccp','outputAll',1);
disp(outputArray);
```

```
1.0e+03 *
      0      -0.0011
0.0018      0.0014
2.2591      2.9355
```

```
outputArray=mcbPMSMSpeeds(pmsm,inverter,'FWCMethod','none','outputAll',1);
disp(outputArray);
```

```
1.0e+03 *
      0      0
0.0018      0.0002
2.2591      2.6081
```

Plot Drive Characteristics

The drive characteristics display the torque-vs-speed, power-vs-speed, and current-vs-speed characteristic plots of a motor under the given operating constraints. Use the `mcbPMSMCharacteristics` function to plot the drive characteristics.

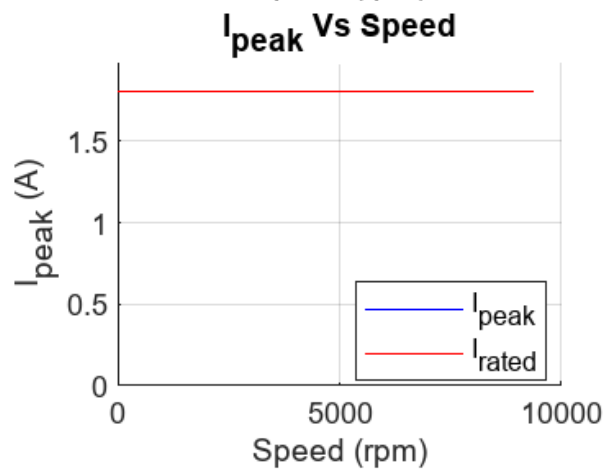
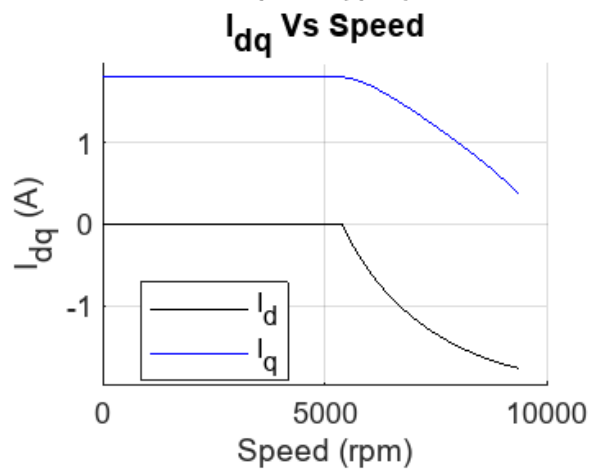
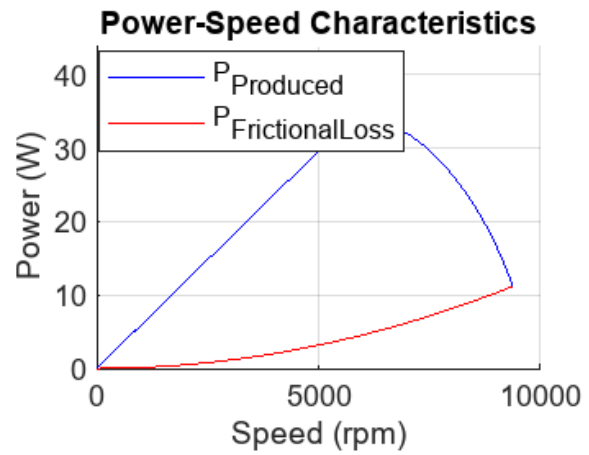
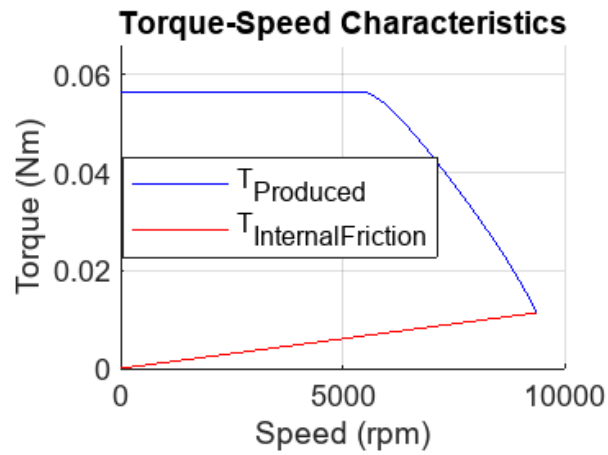
Plot the drive characteristics of the PMSM under different field weakening control methods. You can also plot the constraint curves simultaneously, in which case, you also plot the current trajectory (to get maximum torque for a given speed) for the chosen drive characteristics. When you set the `driveCharacteristics` option to 2, the function plots an additional figure in the v_d - v_q space with the voltage constraint curve and the voltage trajectory. The default field weakening control is set to VCLMT.

Set the fields for the `pmsm` and `inverter` structures and plot the characteristics.

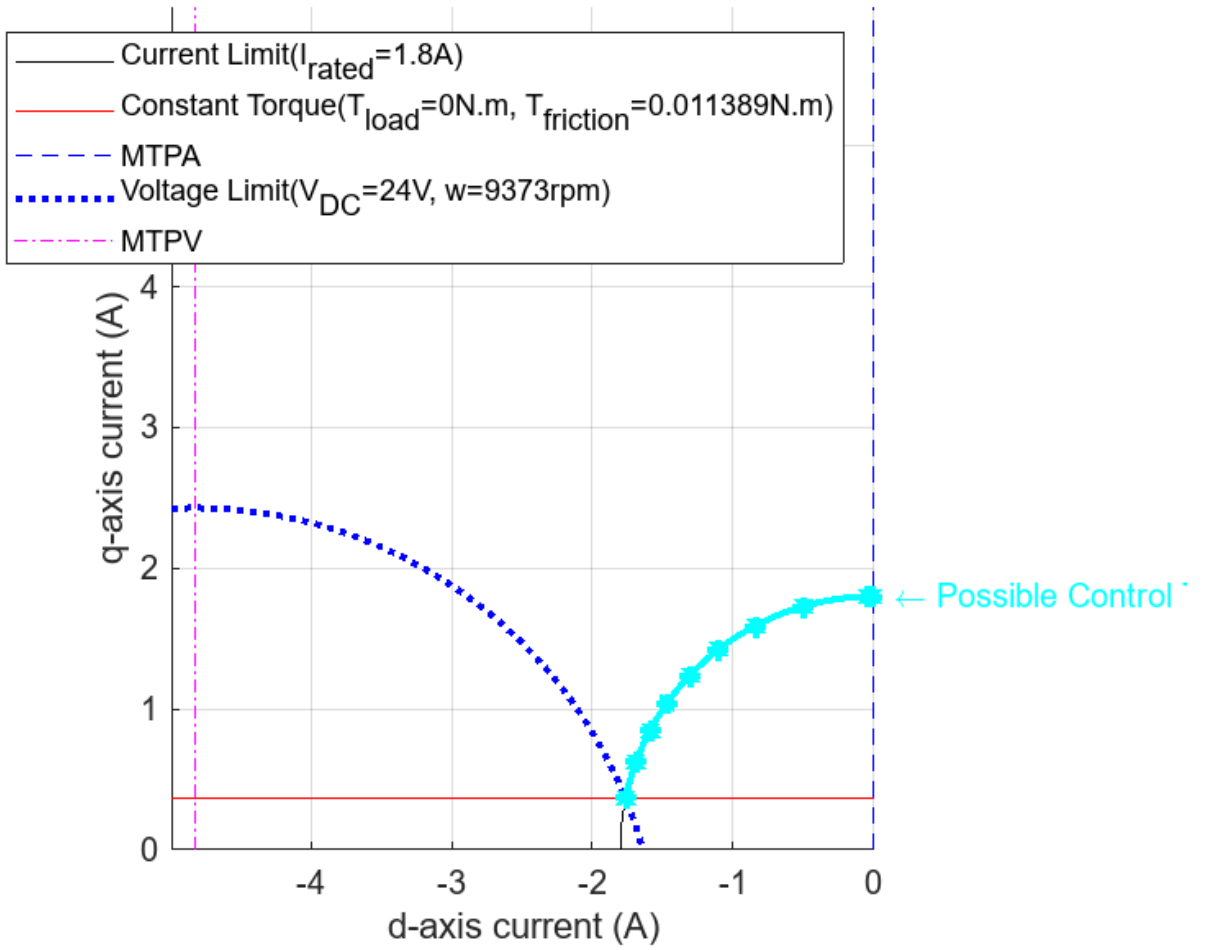
Run this section

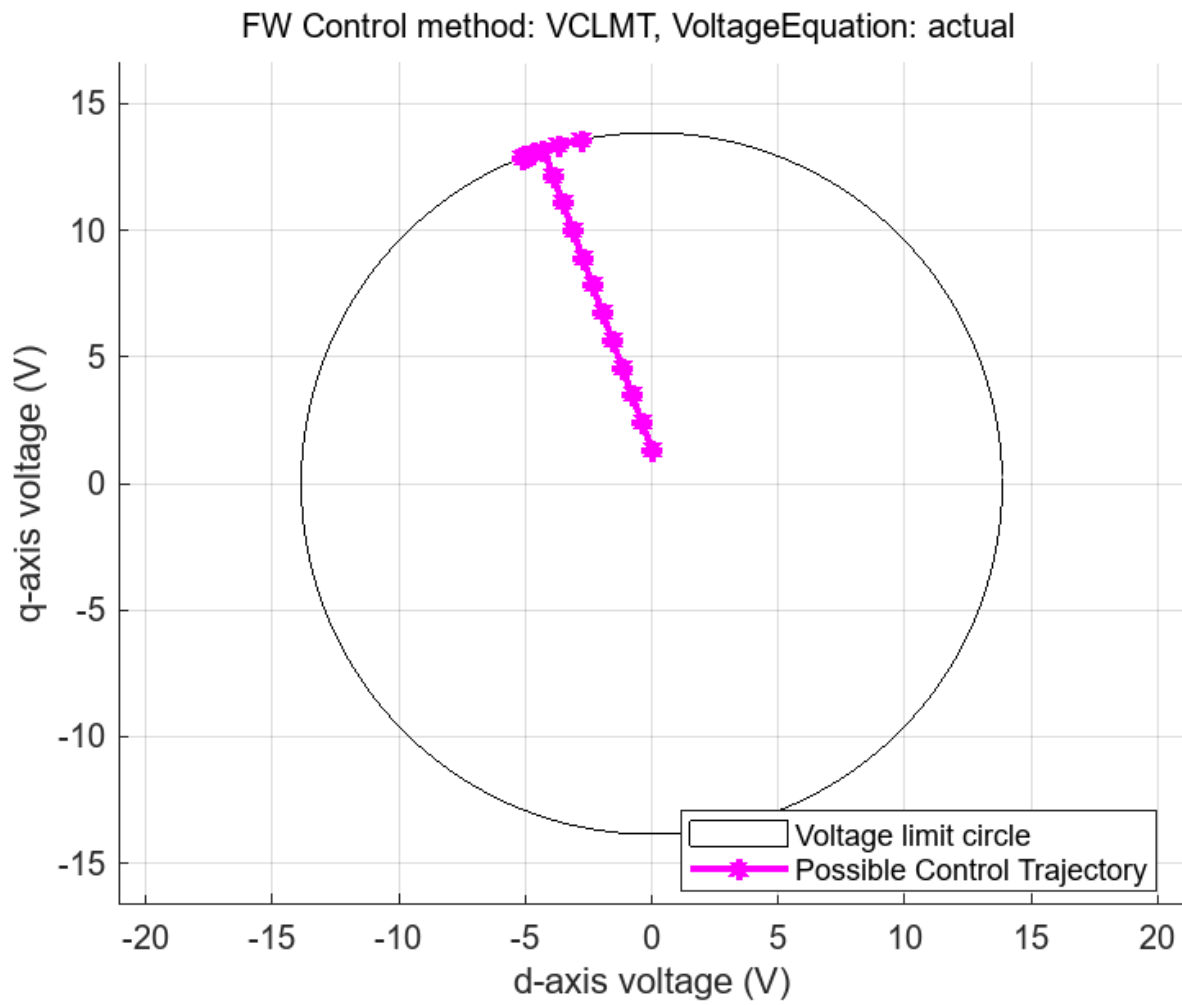
```
inverter = mcb_SetInverterParameters('BoostXL-DRV8305');
pmsm     = mcb_SetPMSMMotorParameters('BLY171D');
mcbPMSMCharacteristics(pmsm,inverter,'driveCharacteristics',2,'constraintCurves',1,'FWCMethod','VCLMT');
```

FW Control method: VCLMT, VoltageEquation: actual



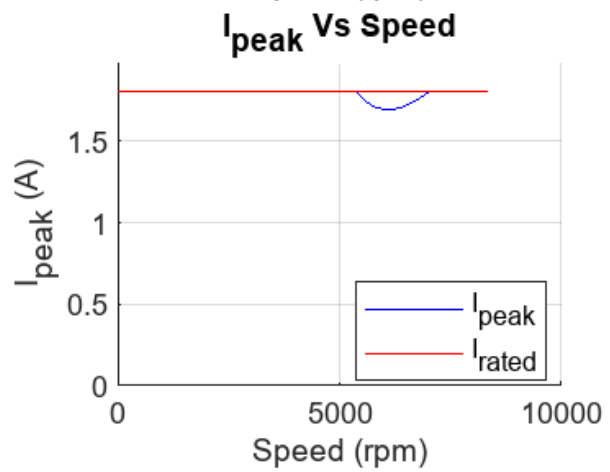
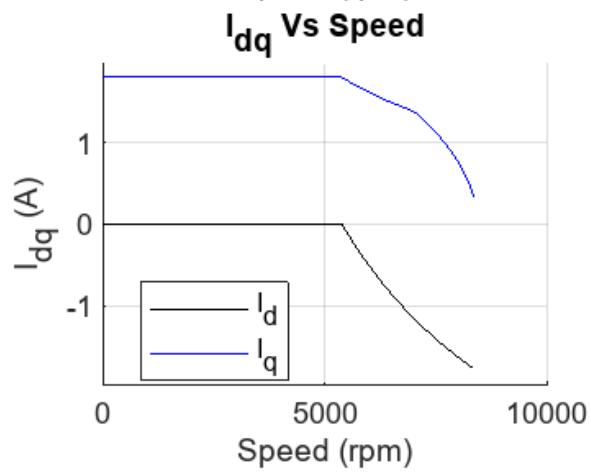
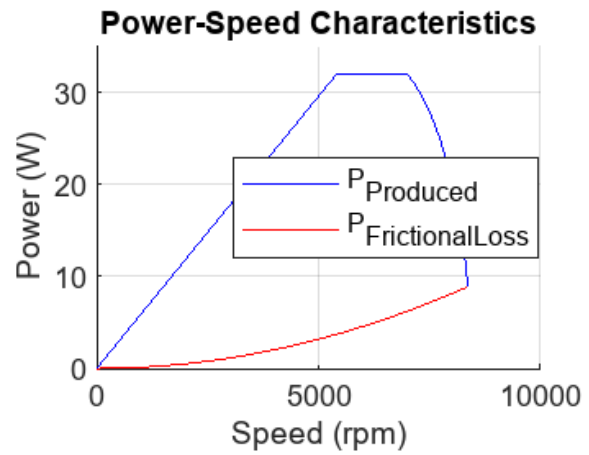
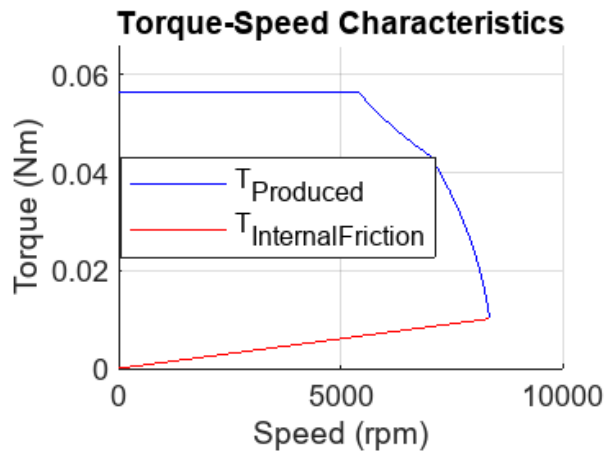
FW Control method: VCLMT, VoltageEquation: actual



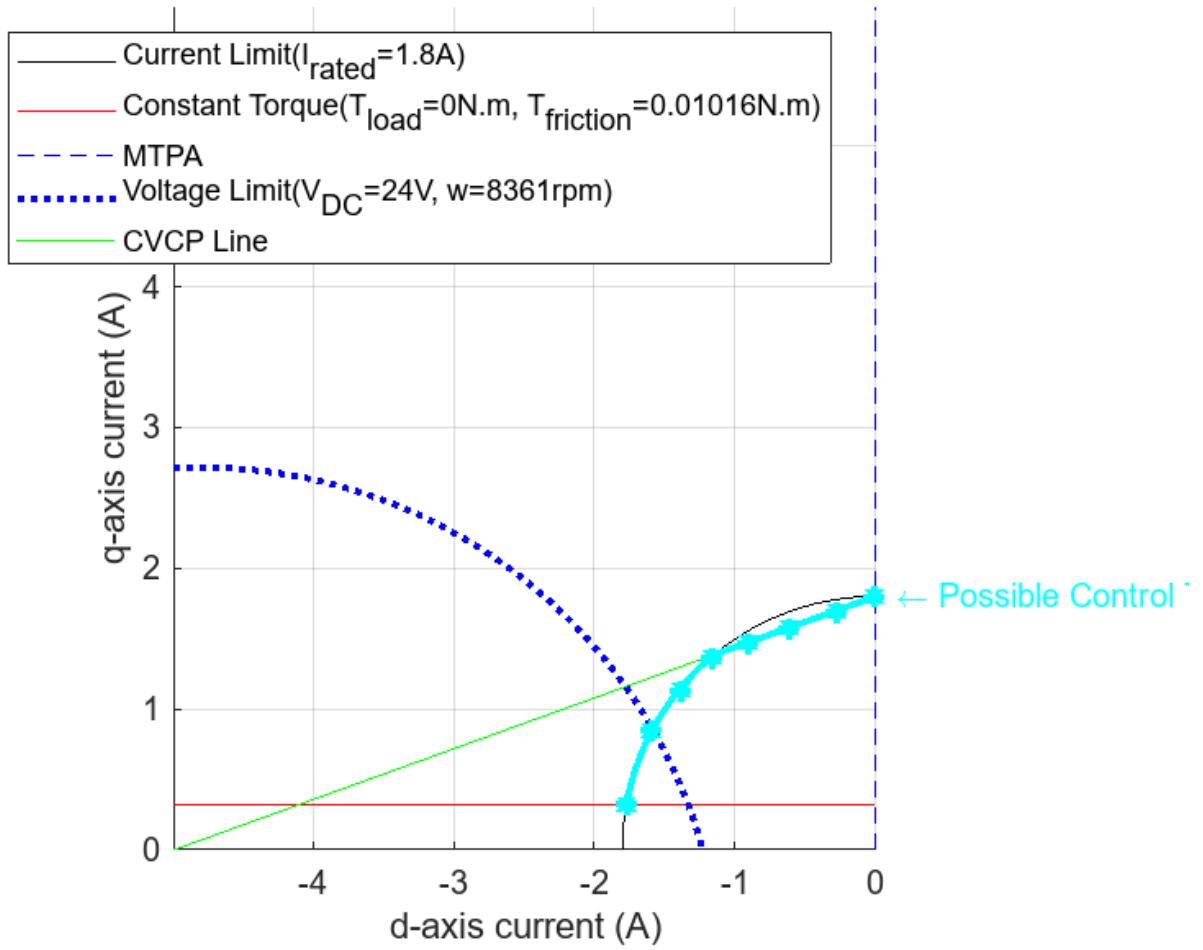


```
mcbPMSMCharacteristics(pmsm,inverter,'driveCharacteristics',1,'constraintCurves',1,'FWCMethod','
```

FW Control method: CVCP, VoltageEquation: actual

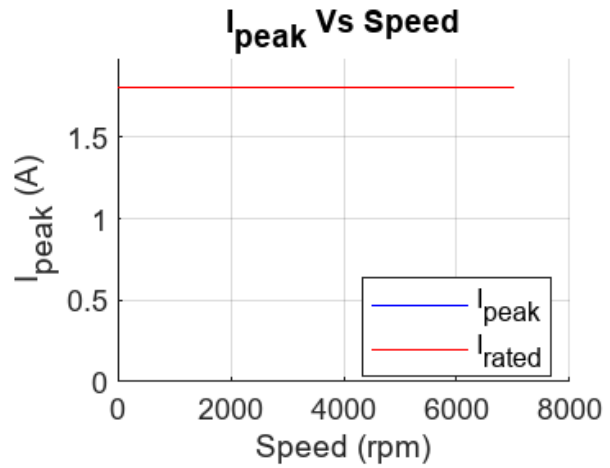
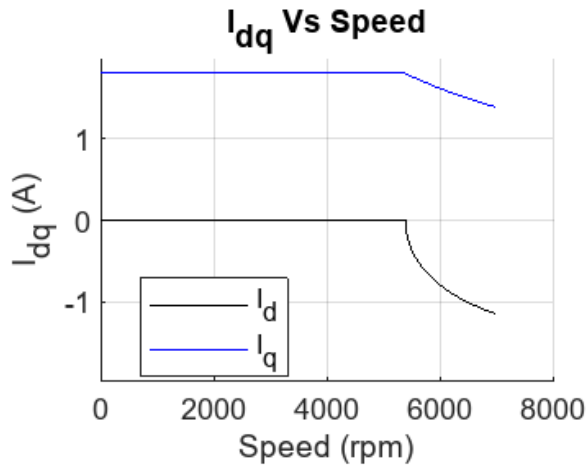
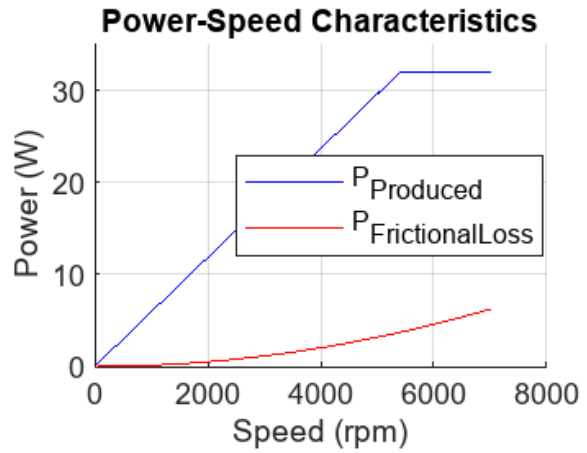
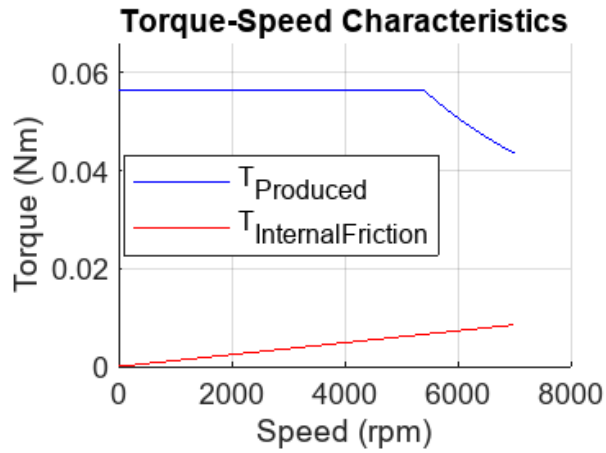


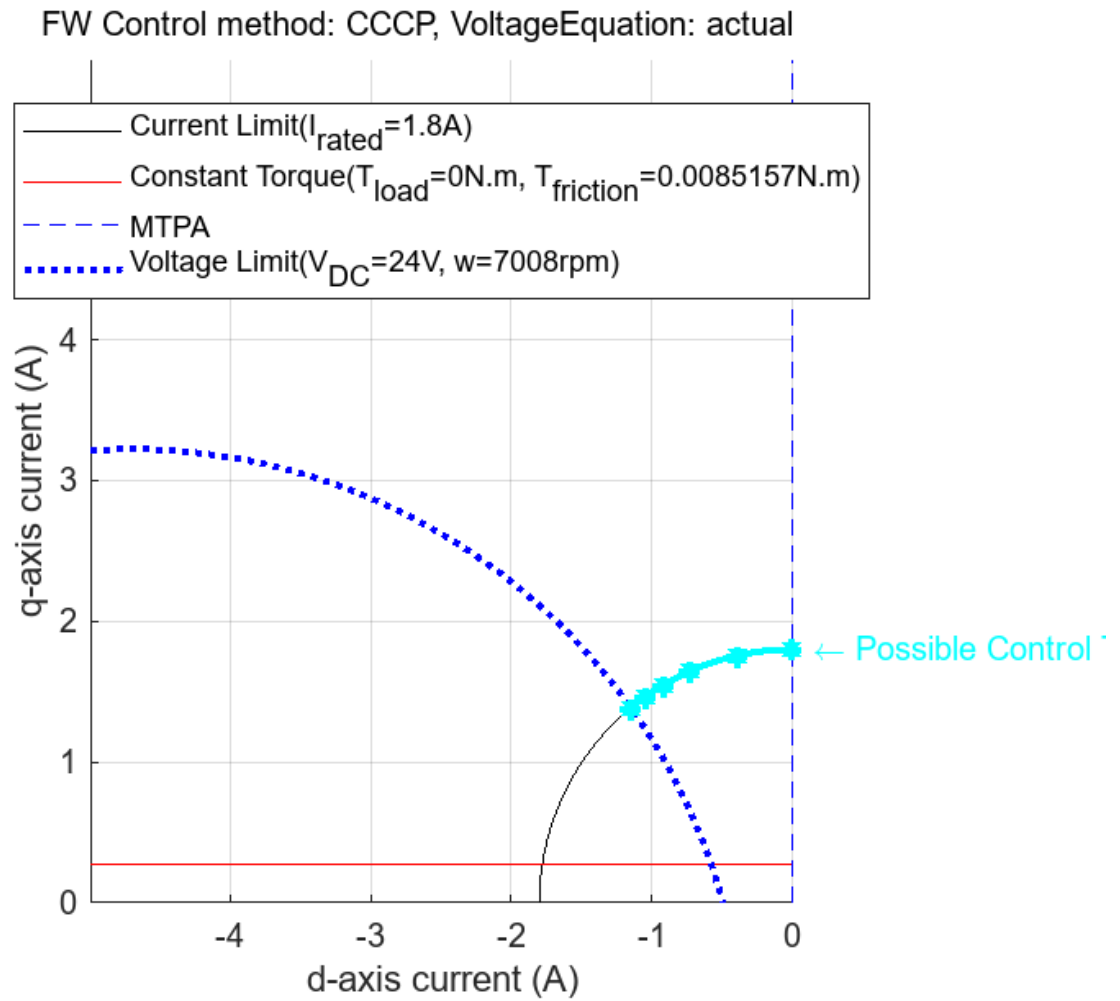
FW Control method: CVCP, VoltageEquation: actual



mcbPMSMCharacteristics(pmsm,inverter,'driveCharacteristics',1,'constraintCurves',1,'FWMethod', 'CVCP')

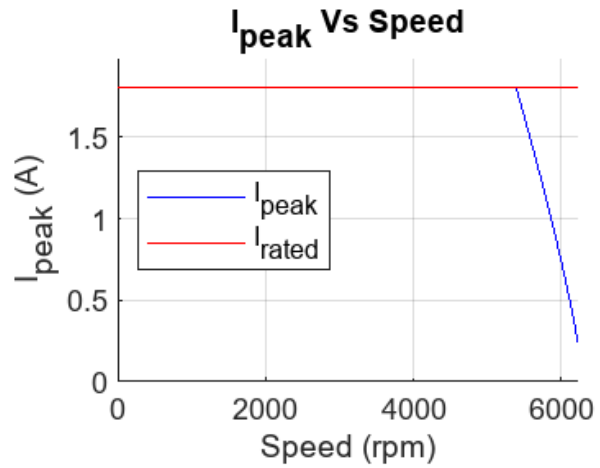
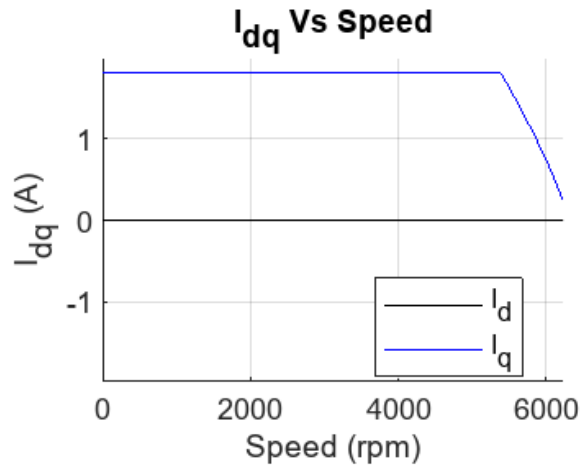
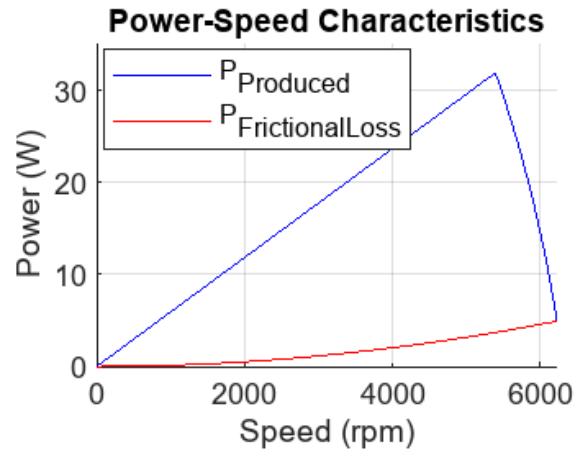
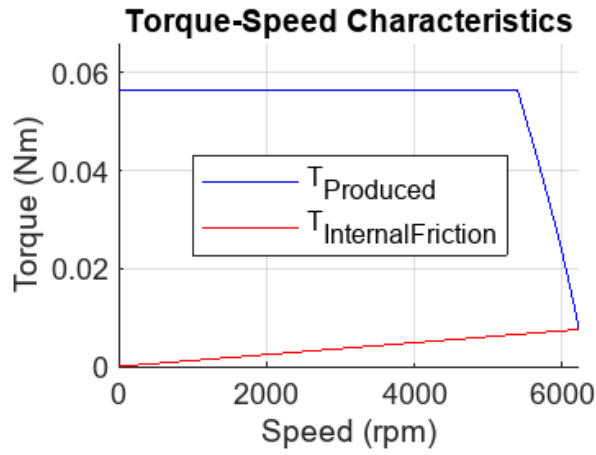
FW Control method: CCCP, VoltageEquation: actual

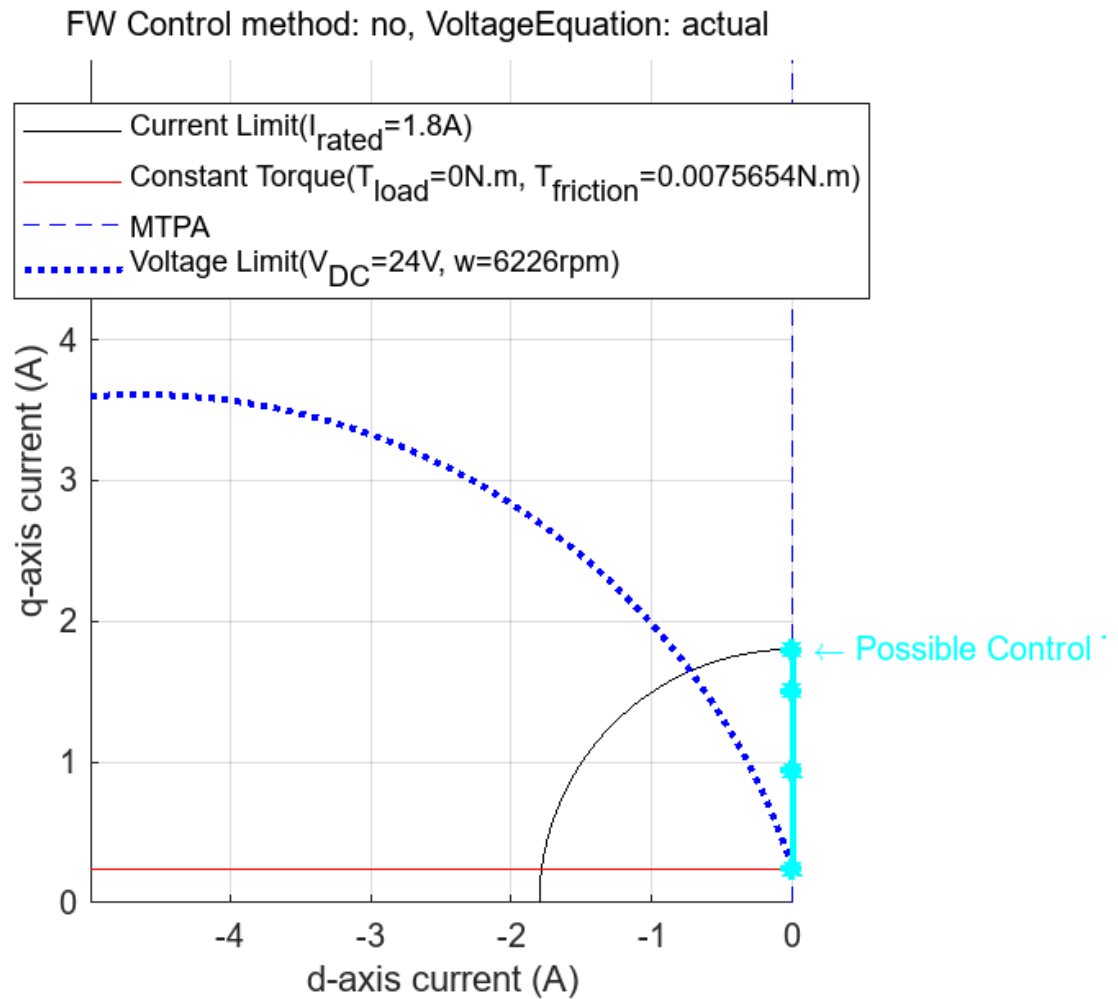




```
mcbPMSMCharacteristics(pmsm,inverter,'driveCharacteristics',1,'constraintCurves',1,'FWCMethod',1)
```


FW Control method: no, VoltageEquation: actual





Plot Drive Characteristics for I_{max}

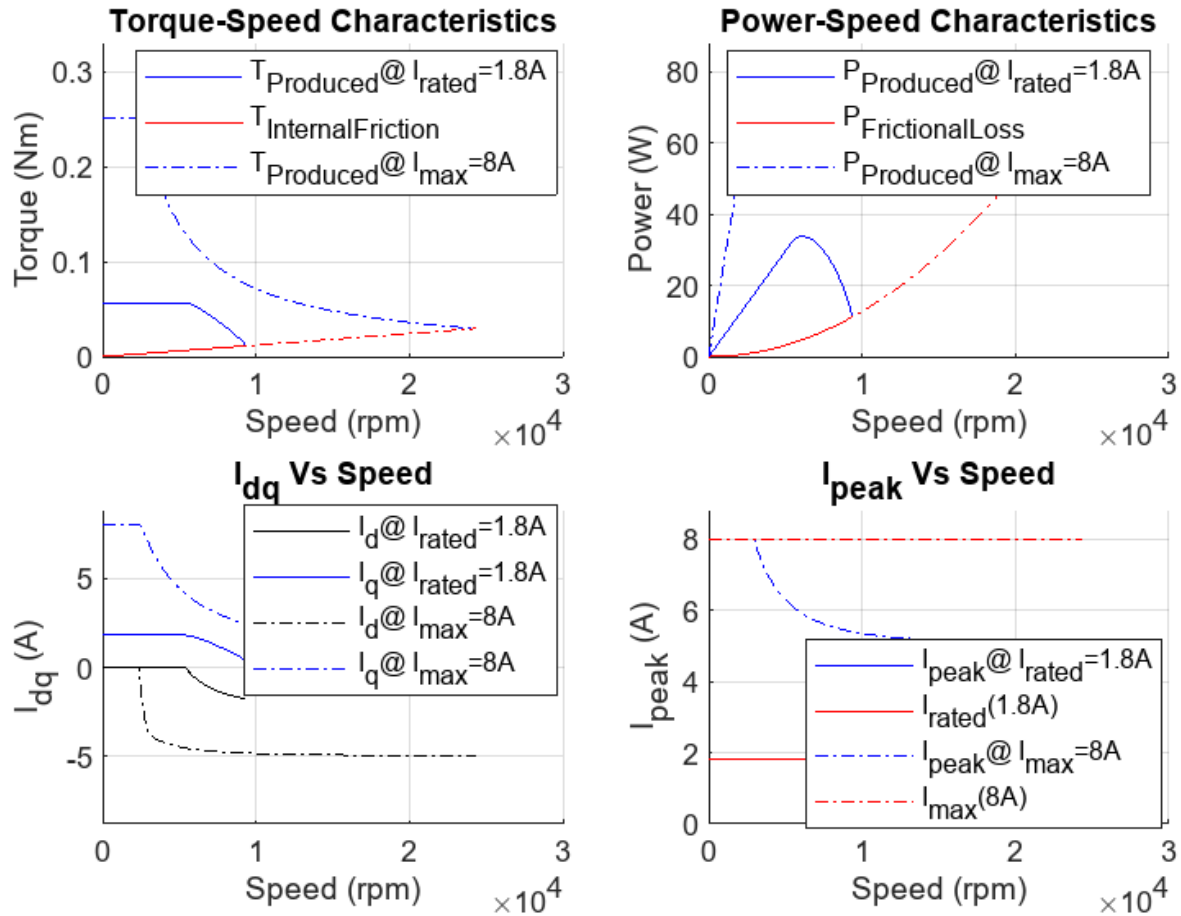
When the motor operates continuously supporting a constant load torque, that condition is the rated operating condition, and the operating current is the rated current. The short-term rating of the motor is higher than the continuous-current rating.

For this example, `pmsm.I_rated` is set to 1.8 A and `pmsm.I_max` is set to 8 A. You can see the comparison between the drive characteristics and use this to determine the short-term operating characteristics.

Run this section

```
inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305' );
pmsm     = mcb_SetPMSMMotorParameters( 'BLY171D' );
mcbPMSMCharacteristics(pmsm,inverter,"FWCMethod","vclmt","driveCharacteristics",1,"imax",8,"cons
```

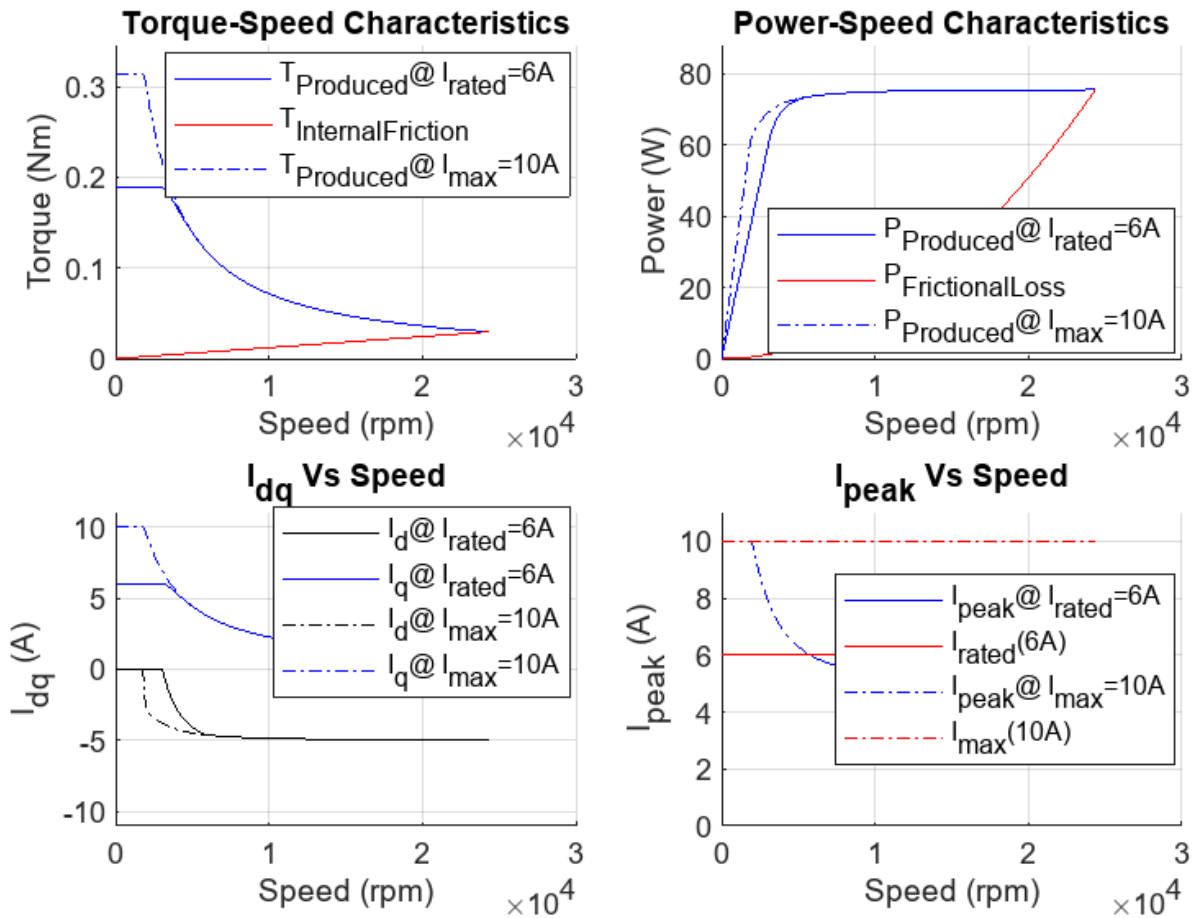
FW Control method: VCLMT, VoltageEquation: actual



Next, assume a high continuous rating to let the motor run with MTPV under rated conditions. The maximum current I_{max} also operates the motor under the MTPV condition, resulting in no change in the maximum speed.

```
pmsm.I_rated=6;
mcbPMSMCharacteristics(pmsm,inverter,"FWMethod","vclmt","driveCharacteristics",1,"imax",10,"cons
```

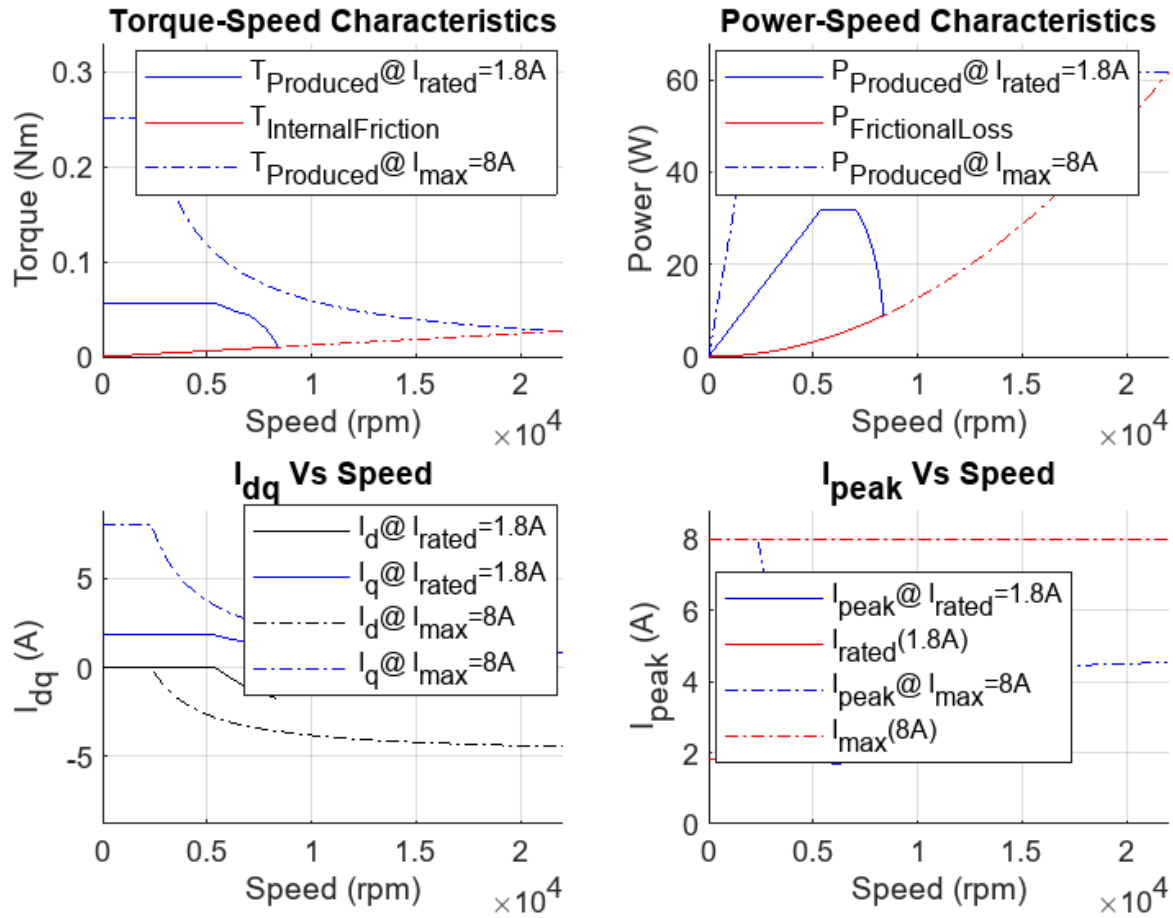
FW Control method: VCLMT, VoltageEquation: actual



When you specify the field weakening method as none, you might not see a change in maximum speed for the original motor (unchanged rated current) even with higher current limit, but only the peak torque and power changes.

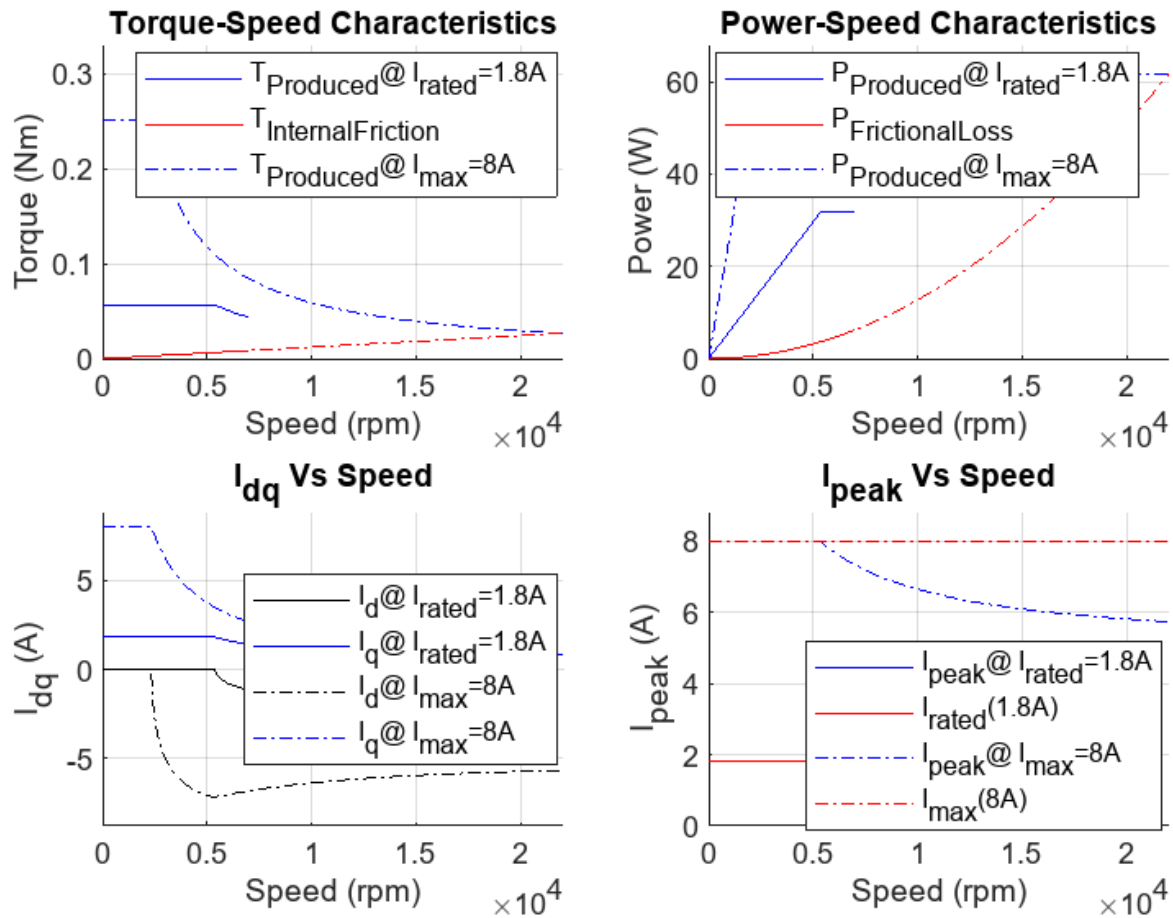
```
pmsm.I_rated=1.8; % setting the rated current to the normal (low) value)
mcbPMSMCharacteristics(pmsm,inverter,"FWMethod","cvcp","driveCharacteristics",1,"imax",8,"const")
```

FW Control method: CVCP, VoltageEquation: actual



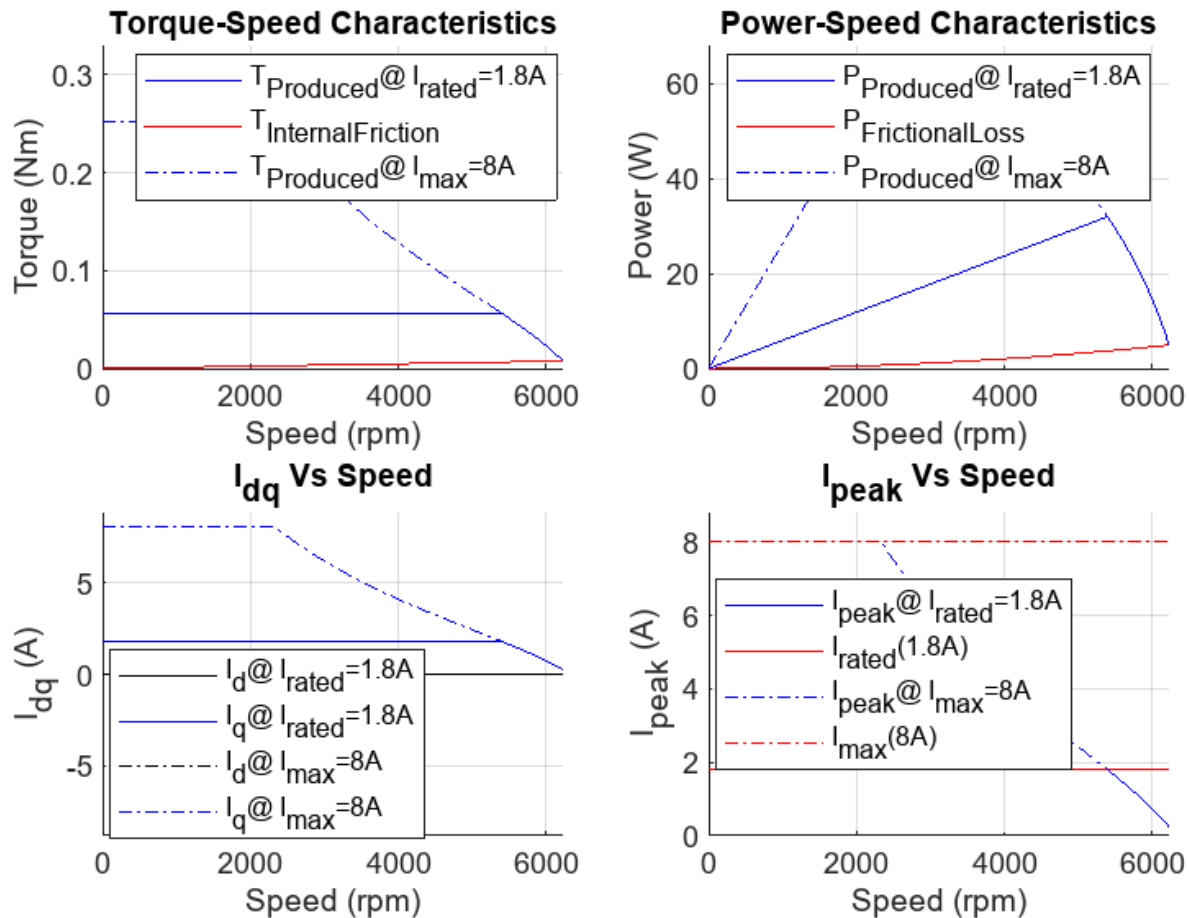
mcbPMSMCharacteristics(pmsm,inverter,"FWMethod","cccp","driveCharacteristics",1,"imax",8,"const

FW Control method: CCCP, VoltageEquation: actual



mcbPMSMCharacteristics(pmsm,inverter,"FWControl","none","driveCharacteristics",1,"imax",8,"const

FW Control method: no, VoltageEquation: actual



Customize Drive Characteristics

Customize the plots using the optional name-value arguments for the characteristics function.

The pmsm structure expects values for the p, Rs, Ld, Lq, FluxPM, B and I_rated fields. The inverter structure expects a value for the V_dc field. Set the field values and plot the characteristics.

Run this section

```

inverter.V_dc= 24 ;
pmsm.p=4;

pmsm.Rs= 0.45 ;
pmsm.Ld=1e-3;

pmsm.Lq=pmsm.Ld* 1.4 ;
pmsm.FluxPM=5.2e-3;
pmsm.B=1.16e-5;

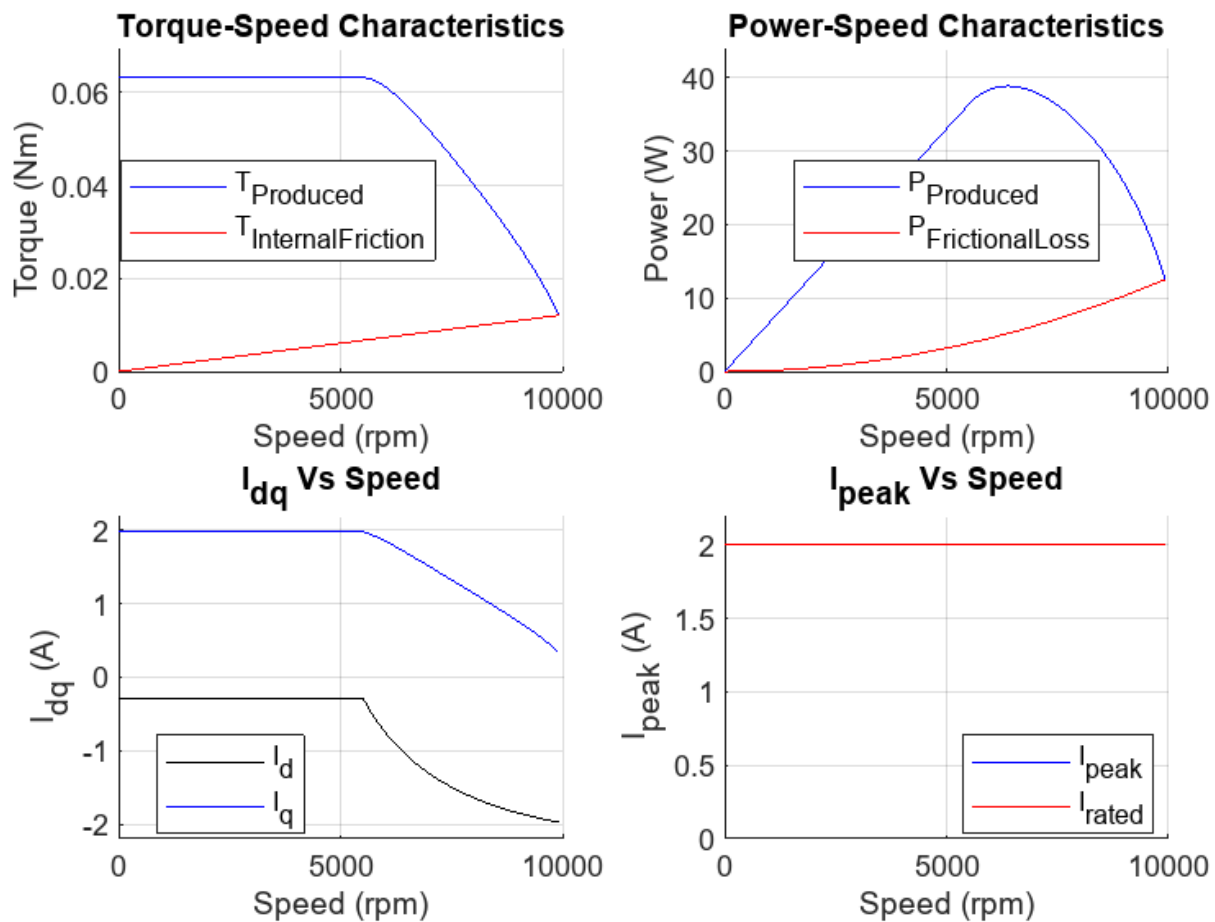
```

```

pmsm.I_rated= 2 ;
w_rpm= 7000 ;
T_load= 0.01 ;
FWCMethod= vclmt ;
mcbPMSMCharacteristics(pmsm,inverter,'speed',w_rpm,'torque',T_load,'FWCMethod',FWCMethod,'const

```

FW Control method: VCLMT, VoltageEquation: actual



Plot Constraint Curves in i_d-i_q Space

You can plot the constraint curves for PMSM in the i_d-i_q space using the `mcbPMSMCharacteristics` function.

Set the fields for the `pmsm` and `inverter` structures and plot the characteristics.

```

Run this section
inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305' );
pmsm     = mcb_SetPMSMMotorParameters( 'BLY171D' );

```

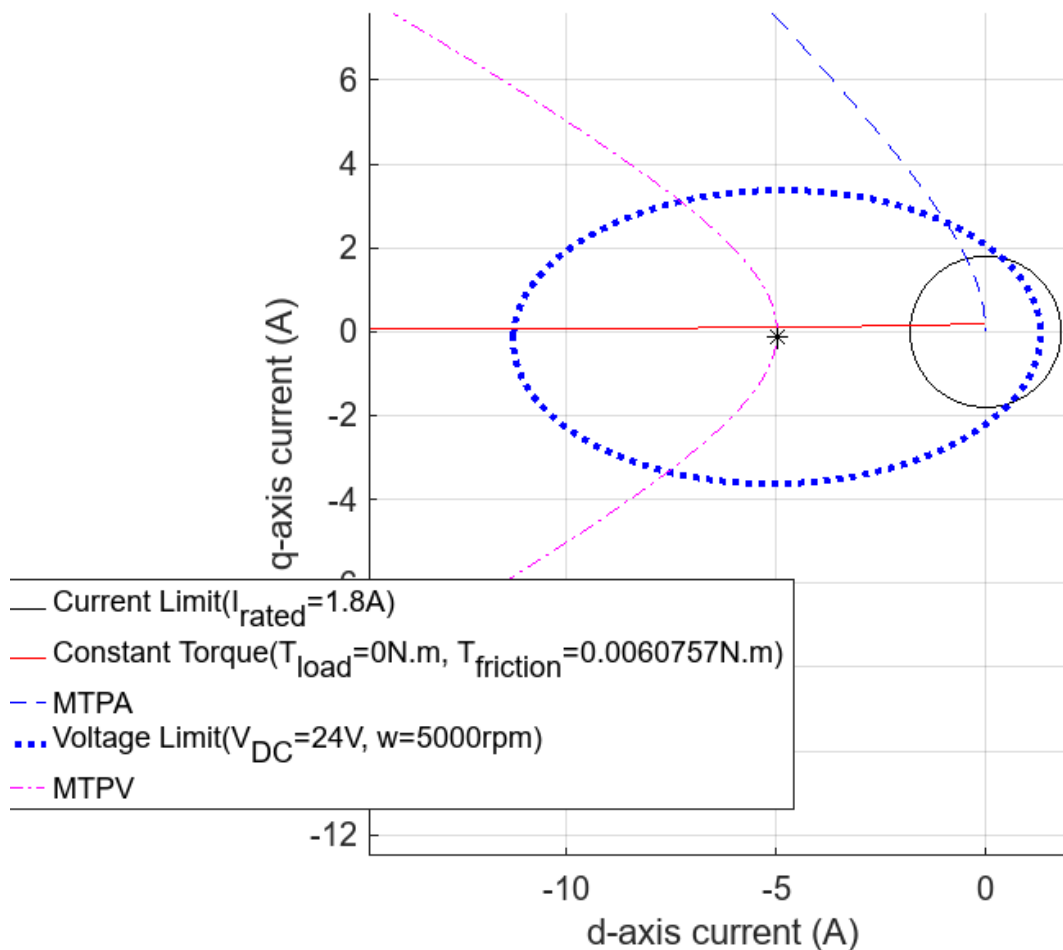


```

pmsm.Rs=0.1;
pmsm.Lq=pmsm.Ld* 1.8 ;
w_rpm= 5000 ;
T_load= 0 ;
mcbPMSMCharacteristics(pmsm,inverter,'speed',w_rpm,'torque',T_load);
legend("Position", [-0.044093,0.16512,0.60536,0.2])
xlim([-14.7 2.1])
ylim([-12.5 7.6])

```

FW Control method: VCLMT, VoltageEquation: actual



Plot Constraint Curves at Maximum Speed

The constraint curves provide a secondary check for whether the calculated speeds are correct. When you plot the constraint curves at maximum speed, the plot shows three intersecting curves: current limit circle, voltage limit curve, and constant torque curve.

Run this section

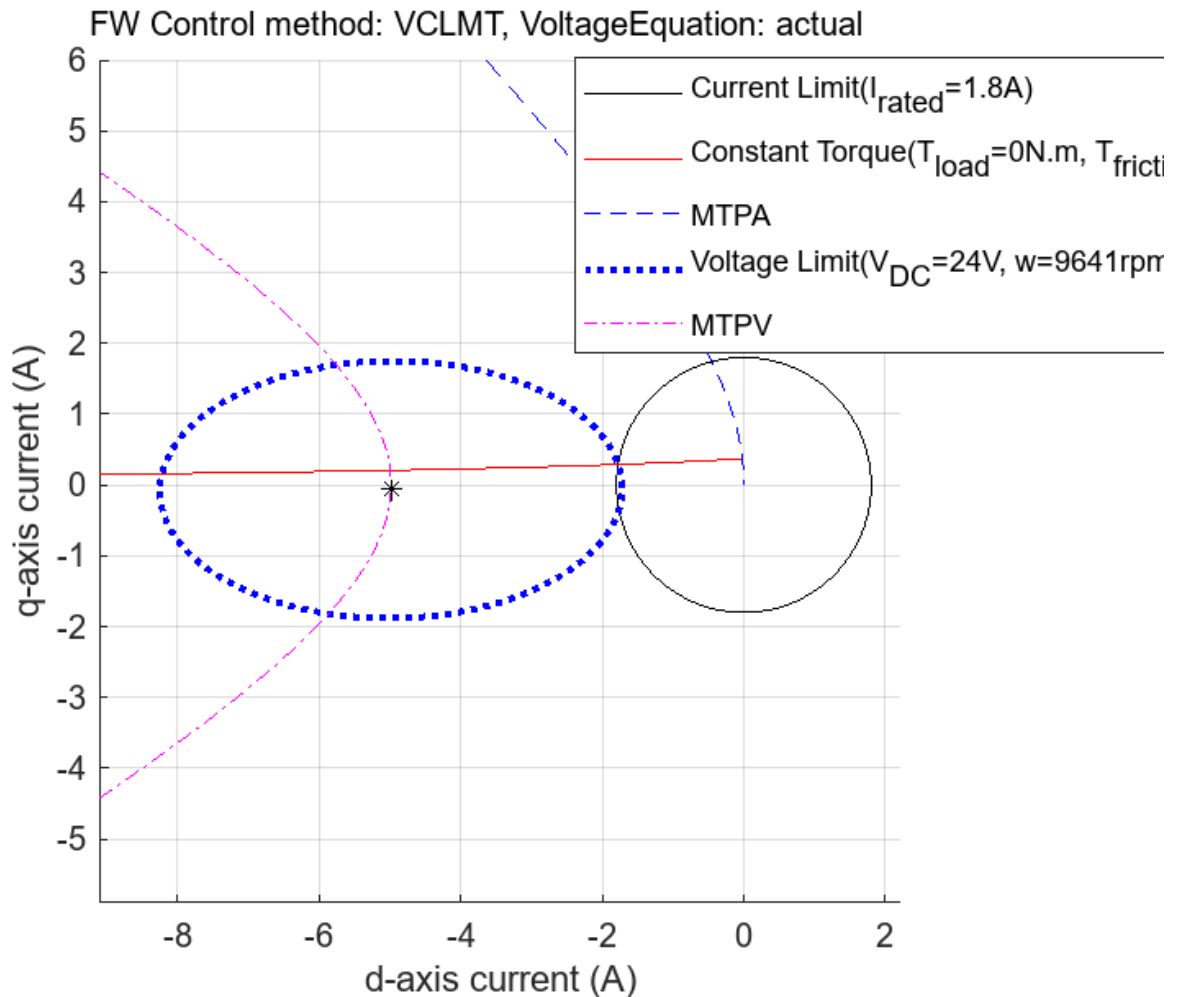
```
inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305' );
```

```

pmsm = mcb_SetPMSMMotorParameters('BLY171D');
pmsm.Rs=0.1;

pmsm.Lq=pmsm.Ld* 1.8 ;
milestone_speeds=mcbPMSMSpeeds(pmsm,inverter);
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(end),torque=0)
xlim([-9.1 2.2])
ylim([-5.9 6.0])
legend("Position",[0.57813,0.64246,0.59643,0.2869])

```



Plot Constraint Curves at Milestone Speeds

Instead of plotting the constraint curves for all the speed milestones in different figures, you can plot them together in the same characteristic plot. In addition to the intersecting curves at maximum speed, notice that at the corner speed, the MTPA curve, the current limit circle, and the voltage constraint curve intersect.

Run this section

```

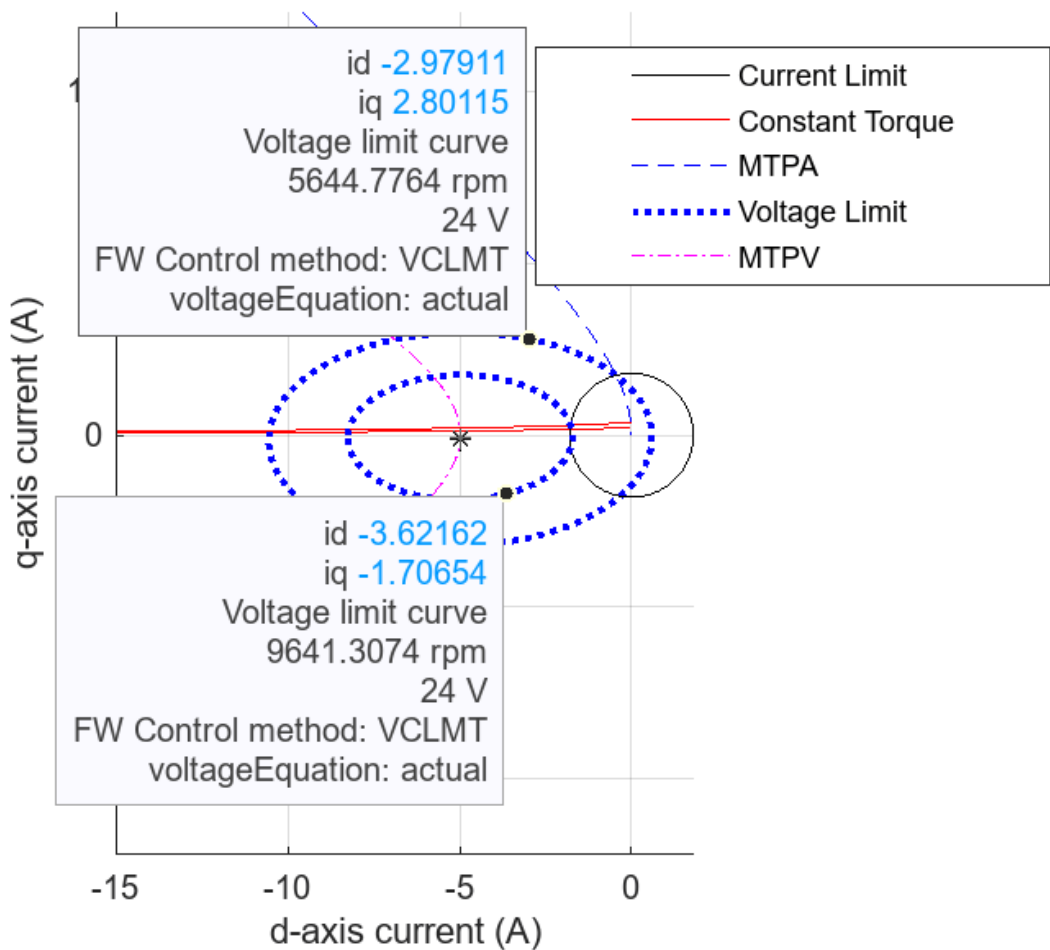
inverter = mcb_SetInverterParameters('BoostXL-DRV8305');
pmsm = mcb_SetPMSMMotorParameters('BLY171D');

```

```

pmsm.Lq=pmsm.Ld* 1.8 ;
pmsm.Rs=0.1;
[milestone_speeds]=mcbPMSMSpeeds(pmsm,inverter,'constraintCurves',1);
legend("Position",[0.61205,0.66151,0.375,0.22976])
ax = gca;
chart = ax.Children(3);
datatip(chart,-3.099,-3.697,"Location","southwest");
chart = ax.Children(9);
datatip(chart,-2.764,3.847,"Location","northwest");
xlim([-15.0 1.8])
ylim([-12.2 12.3])

```



```
disp(milestone_speeds);
```

```

5645
9641

```

Plot Constraint Curves with Different Speeds in Same Figure

Plot the constraint curves of the motor at different speeds in the same figure.

Use the `opacity` option to set the opacity of the presently plotted constraint curve.

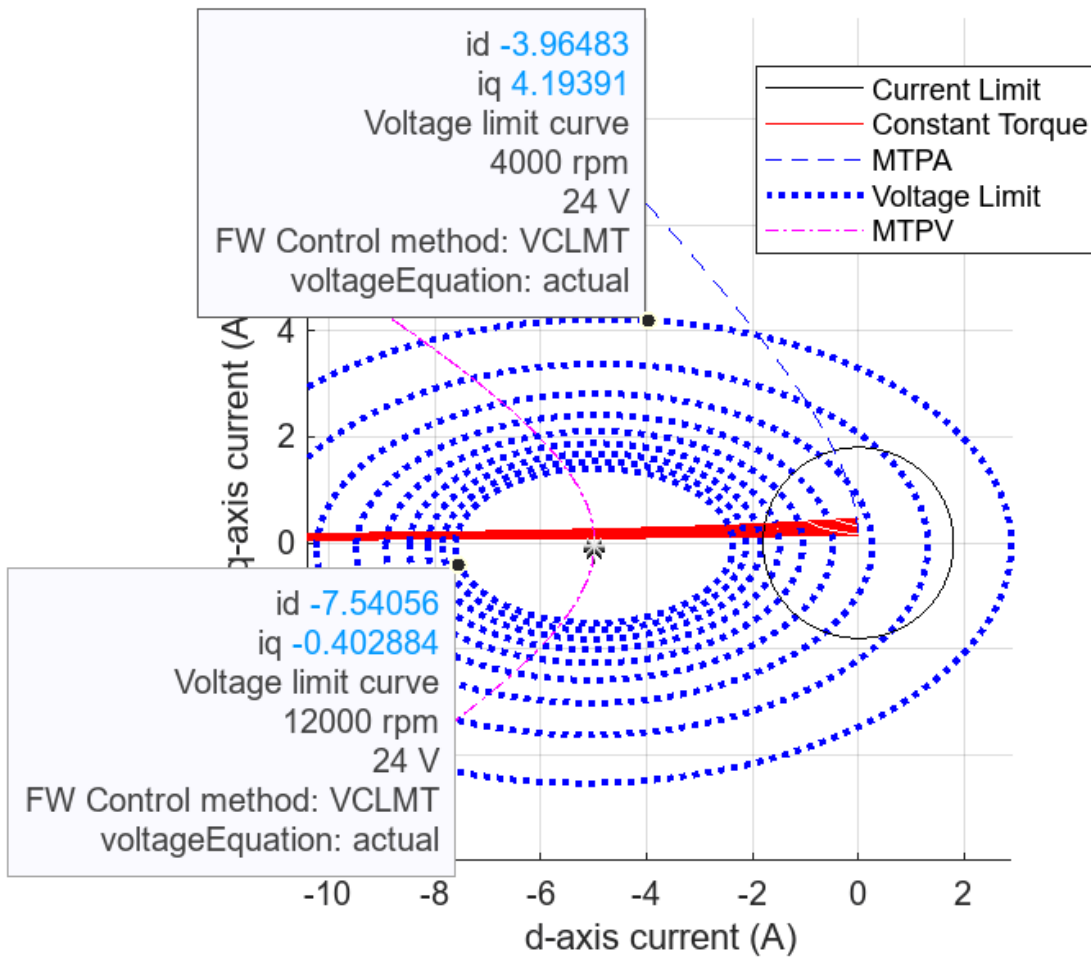
- When you set `opacity` to 1 (default), the function creates a new figure window for plotting.
- When you set `opacity` to a value less than 1 (but more than 0), the function plots the voltage constraint, MTPA, and MTPV curves with changing opacity.

The `opacity` option is useful when you want to identify changes and track trends in these curves.

Run this section

```
inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305' );
pmsm     = mcb_SetPMSMMotorParameters( 'BLY171D' );
pmsm.Rs=0.1;

pmsm.Lq=pmsm.Ld* 1.8 ;
mcbPMSMCharacteristics(pmsm,inverter,'speed',4000,'torque',0,'opacity',1)
mcbPMSMCharacteristics(pmsm,inverter,'speed',5000,'opacity',0.8)
mcbPMSMCharacteristics(pmsm,inverter,'speed',6000,'opacity',0.7)
mcbPMSMCharacteristics(pmsm,inverter,'speed',7000,'opacity',0.6)
mcbPMSMCharacteristics(pmsm,inverter,'speed',8000,'opacity',0.5)
mcbPMSMCharacteristics(pmsm,inverter,'speed',9000,'opacity',0.4)
mcbPMSMCharacteristics(pmsm,inverter,'speed',10000,'opacity',0.3)
mcbPMSMCharacteristics(pmsm,inverter,'speed',11000,'opacity',0.2)
mcbPMSMCharacteristics(pmsm,inverter,'speed',12000,'opacity',0.1)
xlim([-7.61 1.23])
ylim([-1.55 9.05])
ax2 = gca;
chart2 = ax2.Children(3);
datatip(chart2,-7.541,-0.4029,"Location","southwest");
chart2 = ax2.Children(51);
datatip(chart2,-3.893,5.438,"Location","northwest");
xlim([-10.4 2.9])
ylim([-6.0 9.9])
```



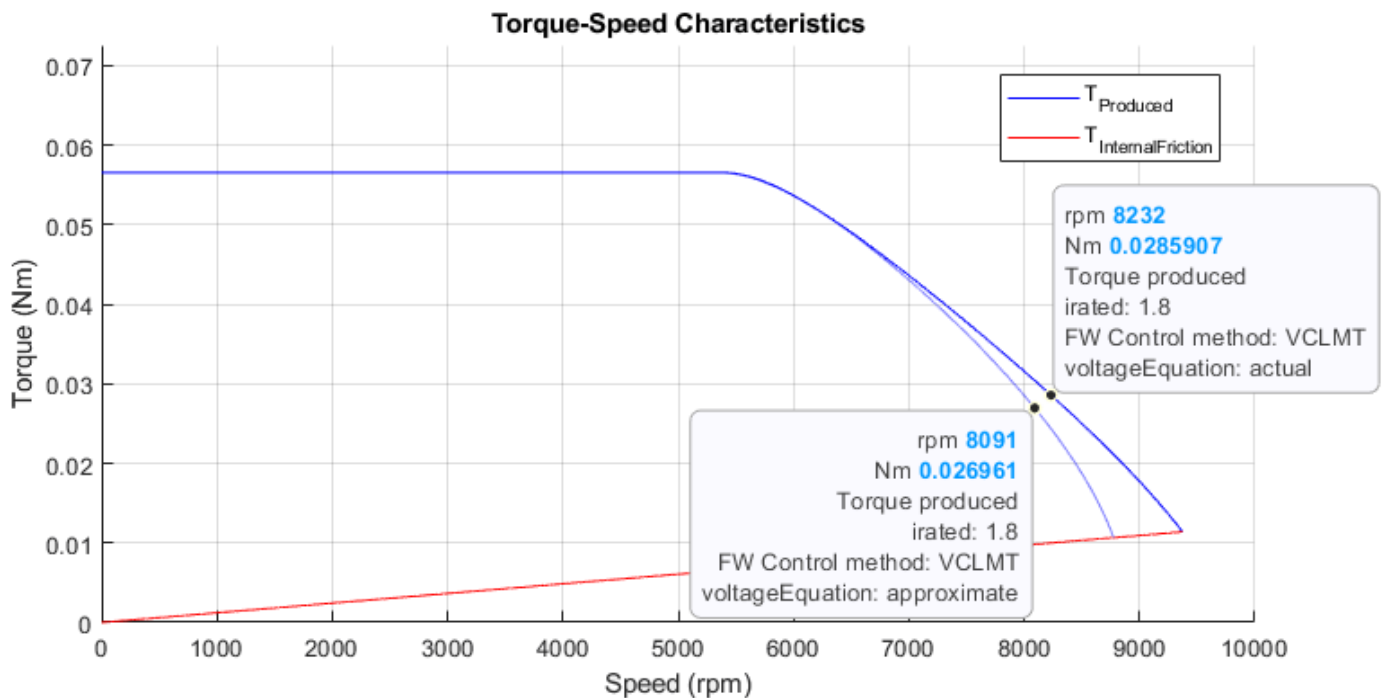
Actual vs Approximate Voltage Equations for Maximum Speed

Usually, the PMSM characteristic equations for v_d and v_q have an $i \times r$ term, which the computations ignore because this value becomes negligible at higher speeds. In this example, you can include or exclude the $i \times r$ term from the v_d and v_q equations while calculating the maximum speeds. The actual form of the equation is as follows.

$$v_d = i_d r - \omega L_q I_q$$

$$v_q = i_q r + \omega L_d i_d + \omega \psi_m$$

This figure shows the difference in the drive characteristics with actual and approximate equations.



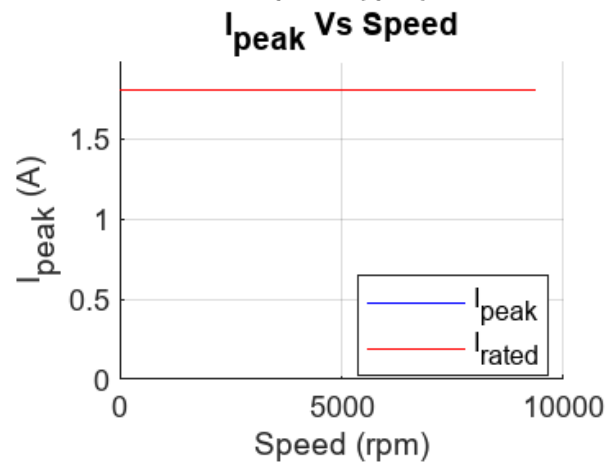
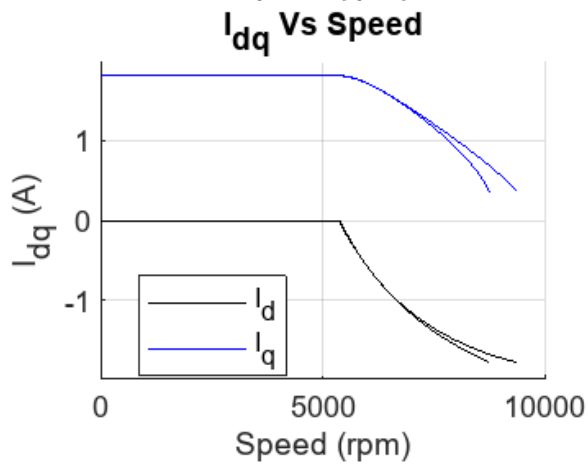
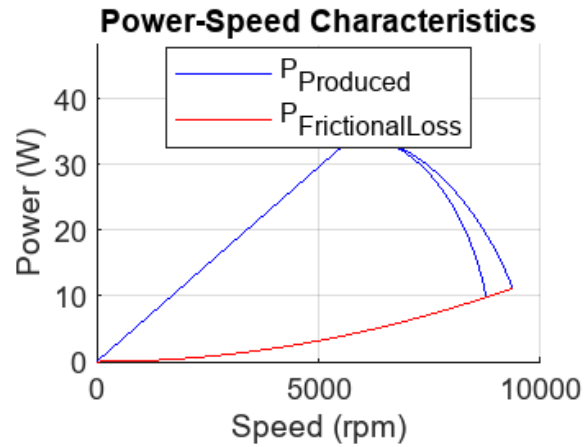
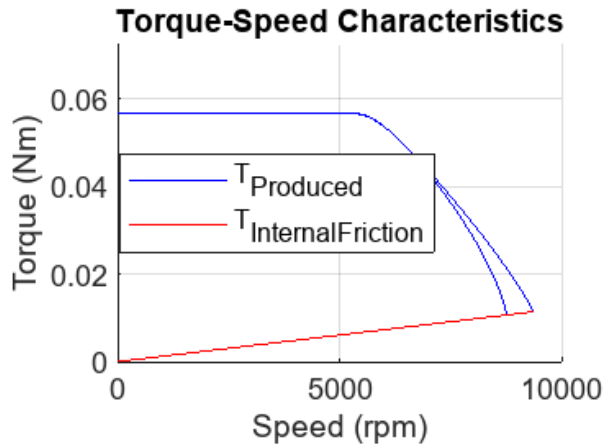
Set the fields for the psm and inverter structures and plot the characteristics with both the actual and approximate equations.

Run this section

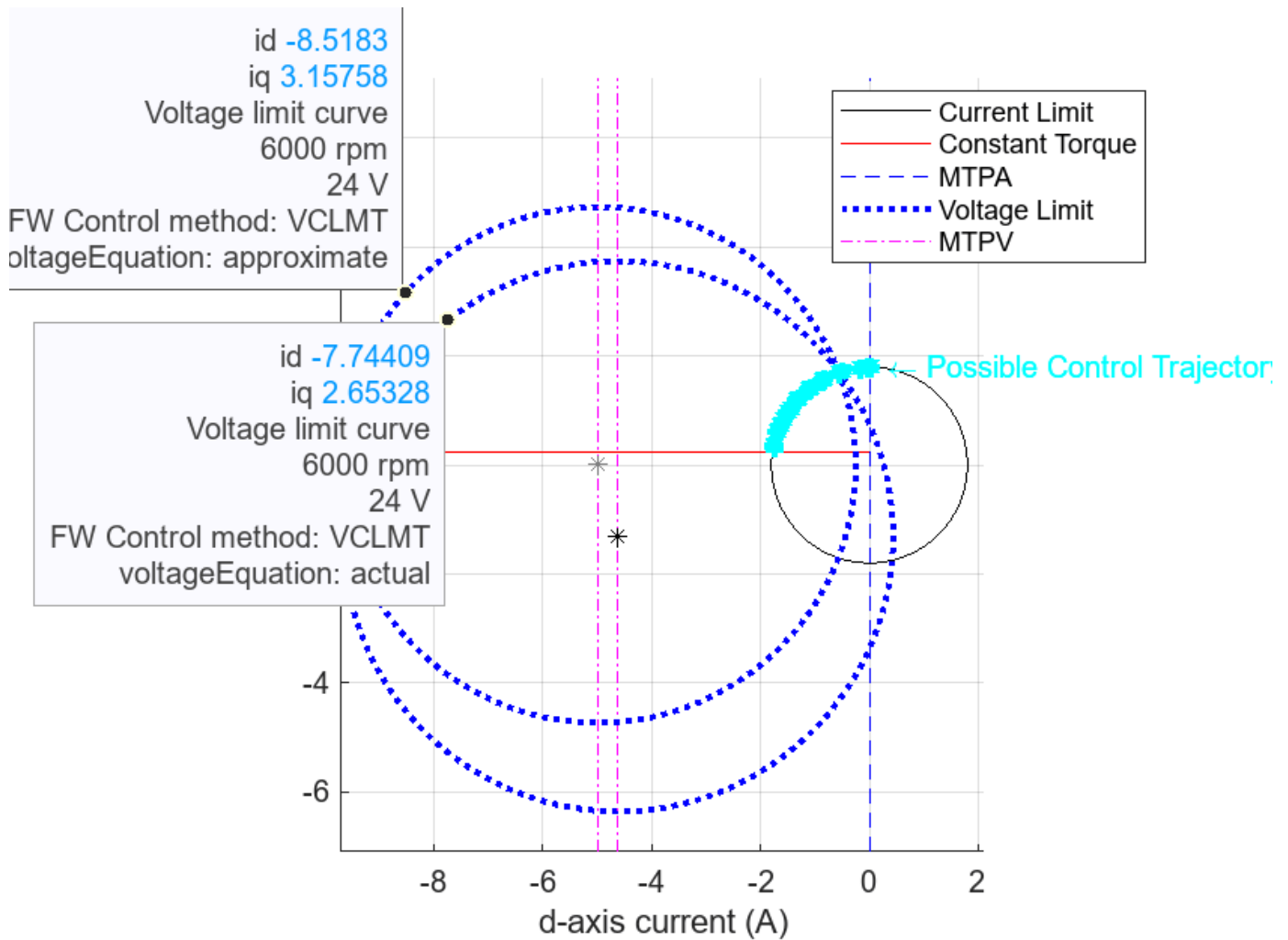
```
inverter = mcb_SetInverterParameters( 'BoostXL-DRV8305' );
psm      = mcb_SetPMSMotorParameters( 'BLY171D' );
max_speed_actual_eqn=mcbPMSMSpeeds(psm,inverter,'verbose',verbose,'voltageEquation','actual');
max_speed_approx_eqn=mcbPMSMSpeeds(psm,inverter,'verbose',verbose,'voltageEquation','approximate');
disp([max_speed_actual_eqn max_speed_approx_eqn])
```

```
5393      5359
9373      8775
```

```
temporary_speed=6000;
mcbPMSMCharacteristics(psm,inverter,'speed',temporary_speed,'voltageEquation','actual',driveChar);
mcbPMSMCharacteristics(psm,inverter,'speed',temporary_speed,'voltageEquation','approximate','op
```



```
xlim([-9.7 2.1])
ylim([-7.1 7.1])
legend("Position", [0.65054,0.72607,0.25,0.19286])
ax3 = gca;
chart3 = ax3.Children(13);
datatip(chart3, -7.744, 2.653, "Location", "southwest");
chart3 = ax3.Children(5);
datatip(chart3, -8.518, 3.158, "Location", "northwest");
```



Customize Constraint Curves

Change the parameters to plot the constraint curves.

The pmsm structure expects values for the p, Rs, Ld, Lq, FluxPM, B and I_rated fields. The inverter structure expects a value for the V_dc field. Set the field values and plot the characteristics.

```

Run this section
inverter.V_dc= 24 ;
pmsm.p=4;
pmsm.Rs= 0.15 ;
pmsm.Ld=1e-3;
pmsm.Lq=pmsm.Ld* 2.3 ;
pmsm.B=1.16e-5;
pmsm.FluxPM=5.2e-3;
    
```

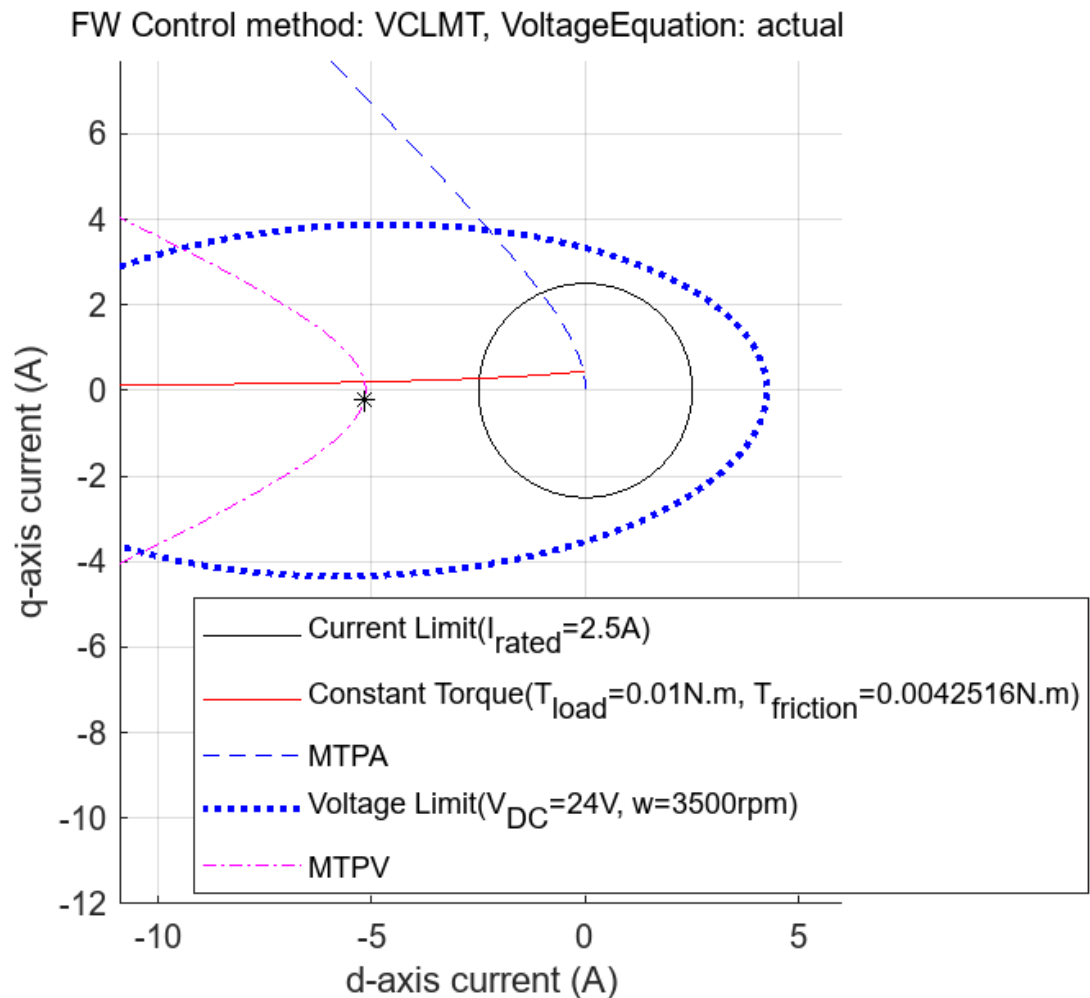


```

w_rpm= 3500 ;
T_load= 0.01 ;
pmsm.I_rated= 2.5 ;
mcbPMSMCharacteristics(pmsm,inverter,'speed',w_rpm,'torque',T_load);
xlim([-10.9 6.0])
ylim([-12.0 7.7])

legend("Position",[0.32518,0.11984,0.61786,0.2869])

```



Case Studies

This section provides case studies that show how to find a possible control trajectory for an IPMSM and SPMSM. The case studies use the following workflow:

- 1 Plot the drive characteristics (including MTPV) and the constraint curves using the `mcbPMSMCharacteristics` function. For this plot, use the corner speed.

- 2 Use the `mcbPMSMCharacteristics` function with `driveCharacteristics` set to `0` and `opacity` set to `0.5` to plot the constraint curves over the figure generated in the previous step. For this plot, use the speed at which MTPV starts.
- 3 Use the `mcbPMSMCharacteristics` function with `driveCharacteristics` set to `0` and `opacity` set to `0.3` to plot the constraint curves over the figure generated in the previous step. For this plot, use the maximum speed.

Doing so plots the three voltage limiting curves overlapping the current limit circle at appropriate points in the final figures for both motors. These figures show the current trajectory for field weakening along the highlighted cyan line. This line follows the MTPV path. When the motor resistance is high enough, the MTPV curve shifts dynamically along with the speed, resulting in the current trajectory not following any single MTPV curve. The resulting cyan curve is the achievable MTPV curve because the example uses the actual equations to dynamically solve for operating points and plot the curves.

To observe the shift in the MTPV curve at different resistance values, use the drop-down menu in the following sections to pick a division factor for the resistance. With a higher resistance, the MTPV curves are farther apart, and with a lower resistance, the MTPV curves are closer to each other.

Plot these characteristics for each type of motor.

IPMSM Case Study

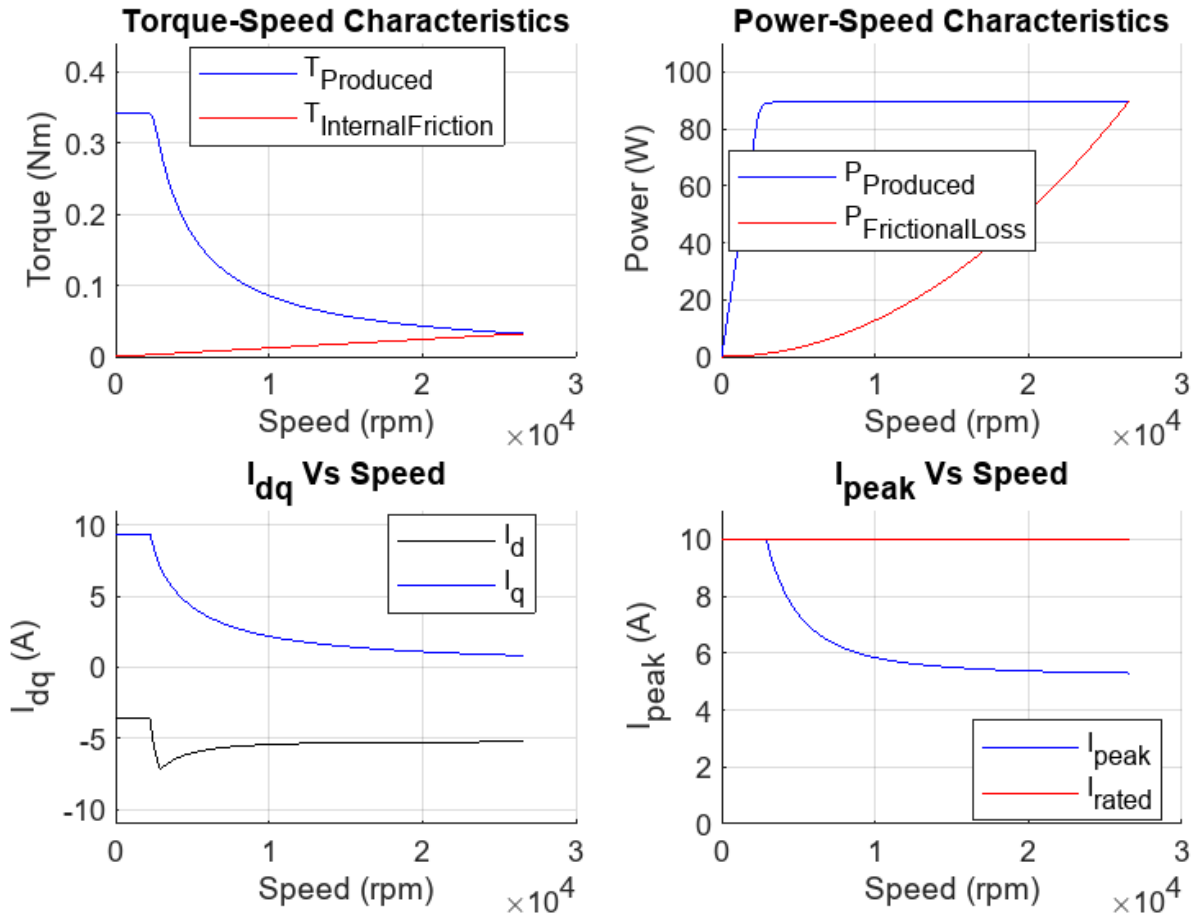
Set the field values and plot the characteristics.

Run this section

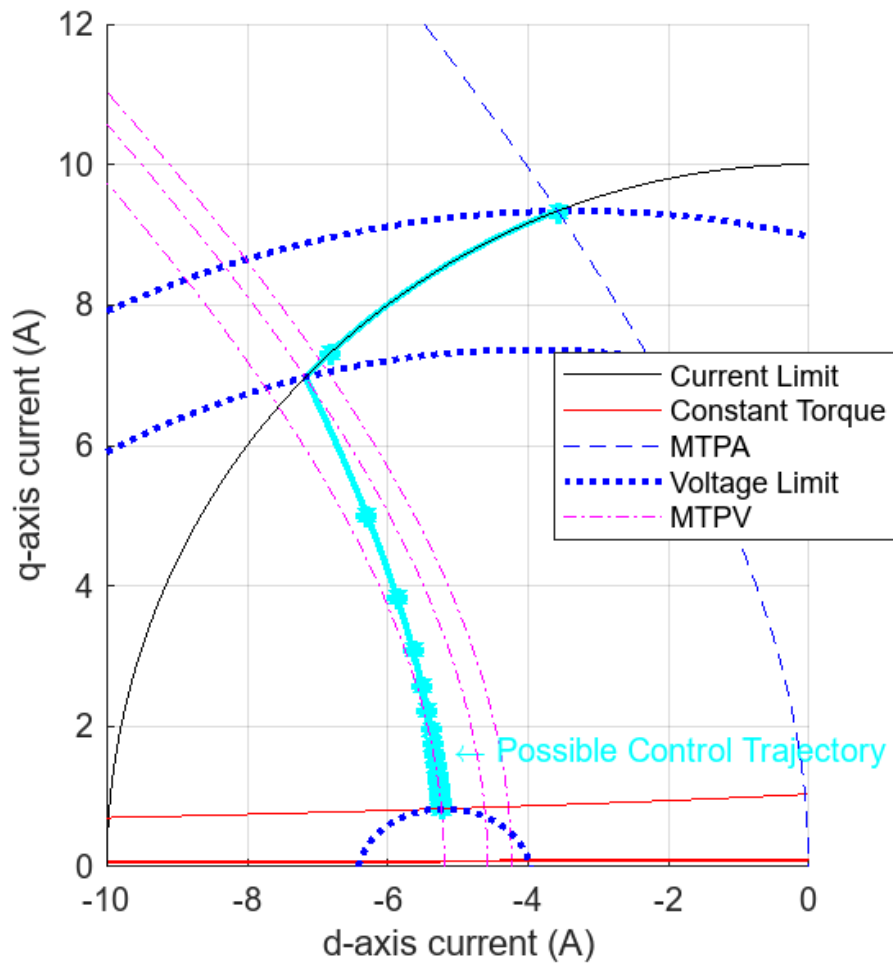
```
inverter.V_dc=24;
pmsm.p=4;
pmsm.Rs=0.45;

pmsm.Rs=pmsm.Rs/;
pmsm.Ld=1e-3;
pmsm.Lq=1.25*pmsm.Ld;
pmsm.FluxPM=5.2e-3;
pmsm.I_rated=10;
pmsm.B=1.16e-5;
[milestone_speeds]=mcbPMSMSpeeds(pmsm,inverter);
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(1),'driveCharacteristics',1,'const
```

FW Control method: VCLMT, VoltageEquation: actual



```
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(2),'driveCharacteristics',0,'const
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(end),'driveCharacteristics',0,'cons
```



SPMSM Case Study

Set the field values and plot the characteristics.

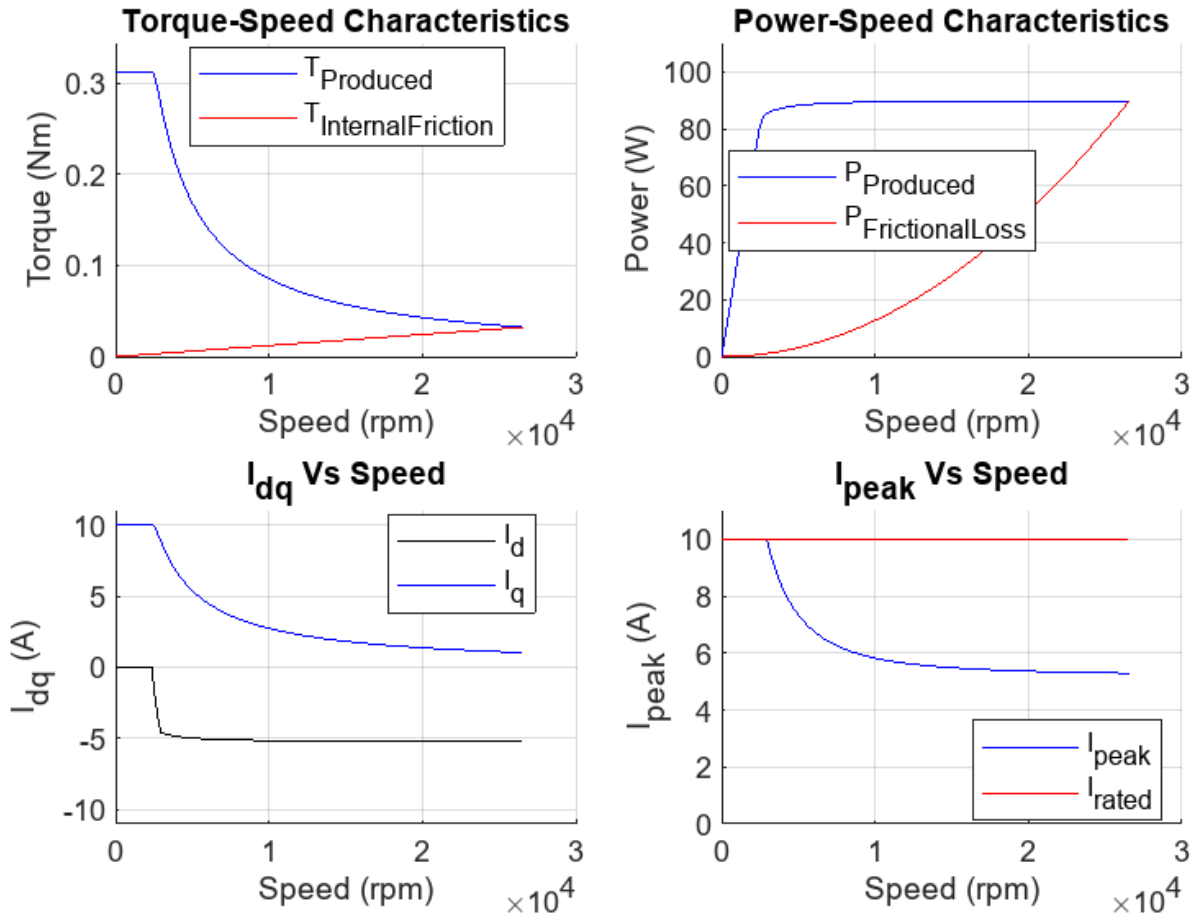
Run this section

```
inverter.V_dc=24;
pmsm.p=4;
pmsm.Rs=0.45;
```

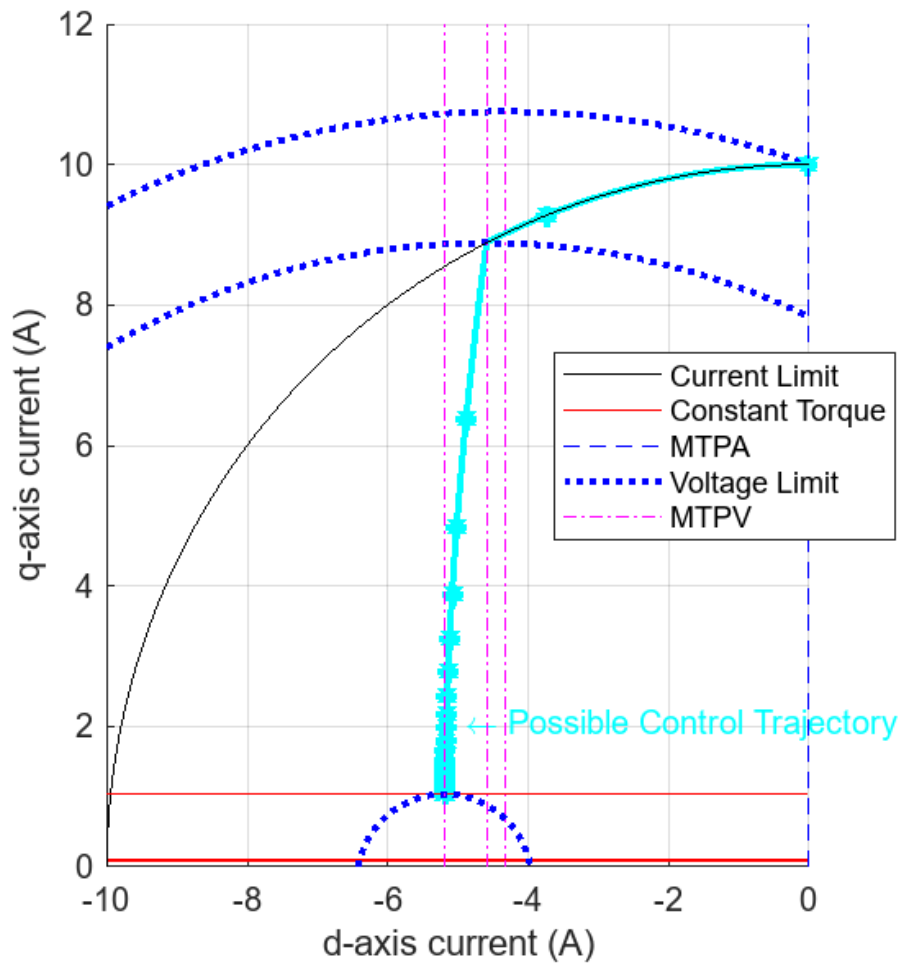
```
pmsm.Rs=pmsm.Rs/  ;
pmsm.Ld=1e-3;
pmsm.Lq=pmsm.Ld;
pmsm.FluxPM=5.2e-3;
pmsm.I_rated=10;
pmsm.B=1.16e-5;
```

```
[milestone_speeds]=mcbPMSMSpeeds(pmsm,inverter);
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(1),'driveCharacteristics',1,'const
```

FW Control method: VCLMT, VoltageEquation: actual



```
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(2),'driveCharacteristics',0,'const
mcbPMSMCharacteristics(pmsm,inverter,'speed',milestone_speeds(end),'driveCharacteristics',0,'cons
```



Note: You can use the **Tab** button to get a list of supported optional arguments.

In the command line, in the live-script environment, or in the editor, the function provides suggestions for the name-value arguments.

```

inverter = mcb_SetInverterParameters(psm, speed, torque);
pmsm     = mcb_SetPMSMMotorParameters(psm, speed, torque);
mcbPMSMCharacteristics(pmsm, inverter, "constraintCurves", 1, "driveCharacteristics", "FWCMethod", "vclmt", "imax", "idqExternal", [0 0; 0 0.1; 0 0]);

```

See also [mcbPMSMSpeeds](#)

```

>> mcbPMSMCharacteristics(pmsm, inverter, "

```

"constraintCurves"
"driveCharacteristics"
"FWCMethod"
"idqExternal"
"imax"
"opacity"
"speed"
"torque"

References

- [1] Mihailovic, Zoran. "Modeling and Control Design of Vsi-Fed Pmsm Drive Systems With Active Load." Thesis, Virginia Tech, 1998. <https://vtechworks.lib.vt.edu/handle/10919/31493>.
- [2] Li, Muyang. "Flux-Weakening Control for Permanent-Magnet Synchronous Motors Based on Z-Source Inverters." Thesis, 2014. http://epublications.marquette.edu/theses_open/284.

Deploy and Validate System

- “Prepare Target Hardware” on page 2-2
- “Add Hardware Drivers to Simulation Model and Deploy to Target Hardware” on page 2-4
- “Task Scheduling in Target Hardware” on page 2-6
- “Adding ADC Driver Library Block” on page 2-8
- “Adding Quadrature Encoder Driver Block” on page 2-11
- “Add PWM Driver Block” on page 2-14
- “Add Hardware Interrupt Trigger Block for Current Control Loop” on page 2-18
- “Run in Open-Loop and Switch to Closed-Loop” on page 2-19
- “Model Configuration and Hardware Deployment” on page 2-23
- “Validate System” on page 2-25

Prepare Target Hardware

Follow these steps to prepare the target hardware before you deploy the control algorithm developed using Motor Control Blockset to it.

Note You need Embedded Coder Support Package for Texas Instruments® C2000™ Processors to run these steps.

We recommend that you see these references before following this procedure:

- Getting Started with Embedded Coder Support Package for TI C2000 Processors
- “Getting Started with Texas Instruments C2000 Microcontroller Blockset” (C2000 Microcontroller Blockset)

In addition, try running the motor using open-loop control first using the “Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset” example.

Verify Direction of Rotation of Motor

The phase sequence of the motor connection in the target hardware determines the direction of rotation of the motor. The Motor Control Blockset example models consider the direction of rotation during the position ramp-up as a positive direction and the corresponding measured speed as positive. It is recommended that you run the motor in open-loop control with a position ramp from 0 to 1 and ensure that the position feedback is positive. The example models in Motor Control Blockset use this convention for the motor's direction of rotation.

For the supported hardware, the algorithm in the example “Quadrature Encoder Offset Calibration for PMSM Motor”, runs the motor and finds the offset between the *d*-axis of the rotor and the encoder index pulse (when the rotor is aligned to the *d*-axis of the stator). The red LED in the host model for this example turns on when the direction of rotation is opposite. When this happens, you should change the phase sequence of the motor wiring (swap any two motor wires).

See the example “Hall Offset Calibration for PMSM Motor” to identify the direction of rotation of a motor that uses Hall sensors.

Note When you use a Hall sensor, ensure that the Hall sequence updated in the Hall Speed and Position and Hall Validity blocks matches the sequence of the actual Hall signals. If you update an incorrect Hall sequence, the direction read by the target hardware is the opposite of the actual direction.

Calibrate Current Sensor

The signal conditioning circuits for the current sensor introduces a voltage offset in the analog to digital converter (ADC) input when measuring both the positive and negative current. For example, an ADC with a voltage reference of 3.3 V can have an offset of 1.65 V when using the Texas Instruments BOOSTXL-DRV8305 hardware. This offset varies due to tolerances of the passive components available in the signal conditioning circuit. It is recommended that you measure the ADC offset of the hardware during initialization.

The hardware initialization subsystem, which is used in the majority of Motor Control Blockset example models, computes the average current sensor ADC values and uses them as ADC offset values for measuring the current. The subsystem represents the ADC offset values in ADC counts.

See the example “Run 3-Phase AC Motors in Open-Loop Control and Calibrate ADC Offset” to manually calibrate the ADC offset and update the computed offset value in the model initialization script file.

See the **Hardware Init** subsystem available in the example “Field-Oriented Control of PMSM Using Quadrature Encoder” to understand the ADC offset calculations that the example model performs before starting the closed-loop motor control.

Calibrate Position Sensor

For a PMSM, the position used in the current control algorithm should align with the d -axis position of the rotor. By default, the quadrature encoder position sensor reads the mechanical position of the rotor with reference to its index pulse. The position offset is the position read by the quadrature encoder when d -axis of the rotor aligns with phase a . To obtain an accurate motor position, use this position offset value to correct the position read by the quadrature encoder sensor. Then provide the corrected motor position value as an input to the current control algorithm.

A mismatch between the actual rotor position and the position provided to the current controller affects the motor functionality and performance.

For more details, see the examples “Quadrature Encoder Offset Calibration for PMSM Motor” and “Hall Offset Calibration for PMSM Motor”.

Add Hardware Drivers to Simulation Model and Deploy to Target Hardware

This topic explains the steps for adding the hardware drivers to the simulation model and deploying the model to the target hardware.

This topic uses the model `mcb_pmsm_foc_sim` as an example to explain the procedure for hardware deployment. The model `mcb_pmsm_foc_sim` simulates the field-oriented control (FOC) algorithm for implementing speed control for a PMSM.

As an example, the procedure explains about deploying the speed control algorithm to the target hardware Texas Instruments LAUNCHXL-F28379D (connected to Texas Instruments BOOSTXL-DRV8305). These are the hardware interface details:

Interface	Pin on LAUNCHXL-F28379D
Phase-A input of the motor	ADCINC2
Phase-B input of the motor	ADCINB2
PWM A output from the motor	EPWM1A
PWM B output from the motor	EPWM2A
PWM C output from the motor	EPWM3A
Enable Driver BOOSTXL-DRV8305	GPIO124

These steps explain how to add the hardware driver blocks from the Embedded Coder Support Package for Texas Instruments C2000 Processors to the simulation model before deploying the control algorithm to the target hardware LAUNCHXL-F28379D (connected to BOOSTXL-DRV8305).

- 1 “Task Scheduling in Target Hardware” on page 2-6
- 2 “Adding ADC Driver Library Block” on page 2-8
- 3 “Adding Quadrature Encoder Driver Block” on page 2-11
- 4 “Add PWM Driver Block” on page 2-14
- 5 “Add Hardware Interrupt Trigger Block for Current Control Loop” on page 2-18
- 6 “Run in Open-Loop and Switch to Closed-Loop” on page 2-19
- 7 “Model Configuration and Hardware Deployment” on page 2-23

You can use MATLAB variables to define or customize parameters like the execution time of the current controller or the speed controller. See the model initialization script associated with the example model `mcb_pmsm_foc_sim` for details about the variables defined in these steps.

To understand the prerequisites for deploying the control algorithm to any target hardware, see “Prepare Target Hardware” on page 2-2. For details about the hardware connections, see “Hardware Connections”.

To implement a simulation model that uses FOC algorithm for a PMSM, see “Design Field-Oriented Control Algorithm” on page 1-2.

A basic understanding of Simulink is a prerequisite to follow these steps. For details about the ADC driver, the quadrature encoder driver, and the hardware interrupt block, see the example model `mcb_pmsm_foc_qep_f28379d`, which uses an architecture similar to what we describe.

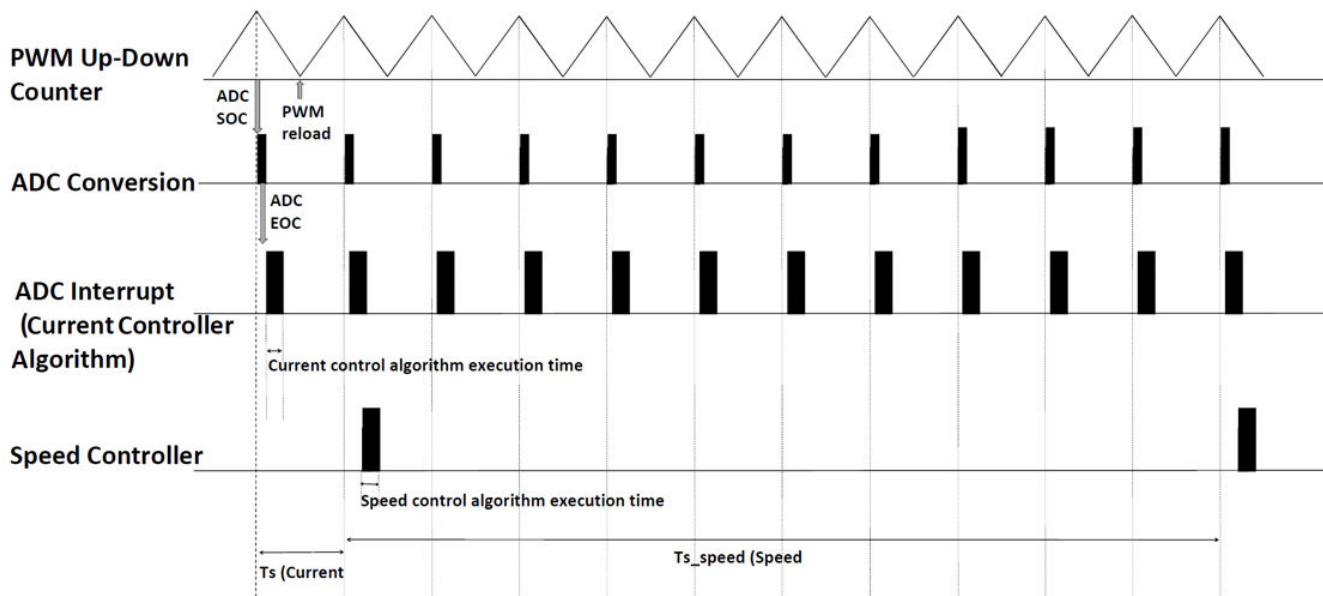
Note For target hardware other than LAUNCHXL-F28379D (connected to BOOSTXL-DRV8305), you can follow these steps, but select the driver blocks (ADC, PWM, Interrupt) from the appropriate supported hardware library.

Task Scheduling in Target Hardware

In the example model `mcb_pmsm_foc_sim`, configuring the current controller and the speed controller are the two important tasks. The current controller is scheduled to run after every T_s (50 μ sec for a 20 kHz switching frequency) and the speed controller runs after every T_{s_speed} ($10 \cdot T_s$). The current controller reads the motor phase currents and position and computes the PWM duty cycle to run the motor. The speed controller runs the control loop, calculates I_q reference for the current controller, and controls the motor speed in the closed-loop.

In the target hardware, the current controller is synchronized with the ADC interrupt (for every T_s) and the speed controller is triggered after every T_{s_speed} ($10 \cdot T_s$).

This figure shows the event sequence, interrupt trigger, and software execution time for the control algorithm running in the target hardware.



In this figure, the execution times for the current controller and speed controller are not in scale. See the processor datasheet to better understand the functionality of the processor peripherals such as the ADC (analog-to-digital converter) and the PWM (pulse-width modulation).

The model follows this event sequence:

- 1 The processor peripheral PWM, which is center-aligned (Up-Down Counter), triggers the start-of-conversion (SOC) event for the ADC module when the PWM counter value equals the PWM period.
- 2 The ADC module converts the sampled analog signal into digital counts and triggers the end-of-conversion (EOC) event.
- 3 The EOC triggers the ADC interrupt.
- 4 The current controller is scheduled to execute with the ADC interrupt.
- 5 The speed controller is scheduled to run after every T_{s_speed} .

You can also use SoC Blockset™ for task scheduling, profiling, and addressing 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”.

Adding ADC Driver Library Block

In the example model `mcb_pmsm_foc_sim`, the subsystem for current controller receives the motor phase current in ADC counts from the plant model that converts the motor phase current from Amperes to ADC counts. In the target hardware, the current controller reads the motor phase current from the ADC driver block. Follow this workflow to add and configure the ADC driver block.

These steps explain addition and configuration of the ADC driver blocks in detail. In the Simulink library browser, select and add the ADC block from the F2837xD library in Embedded Coder Support Package for Texas Instruments C2000 Processors. Use the following steps to configure the ADC blocks to read the phase-A and phase-B currents of the motor.

In the Texas Instruments BOOSTXL-DRV8305 inverter hardware, the phase-A current of the motor is read from ADC C2 channel and phase-B current is read from ADC B2 channel. In the ADC driver block for phase-A current (see the following figure), select **ADC module C** and conversion channel 2 to obtain the phase-A current of the motor. In the ADC driver block for phase-B current, select **ADC module B** and conversion channel 2 to obtain the phase-B current of the motor. For other target hardware, select the **ADC module** and channel where the motor phase currents are interfaced.

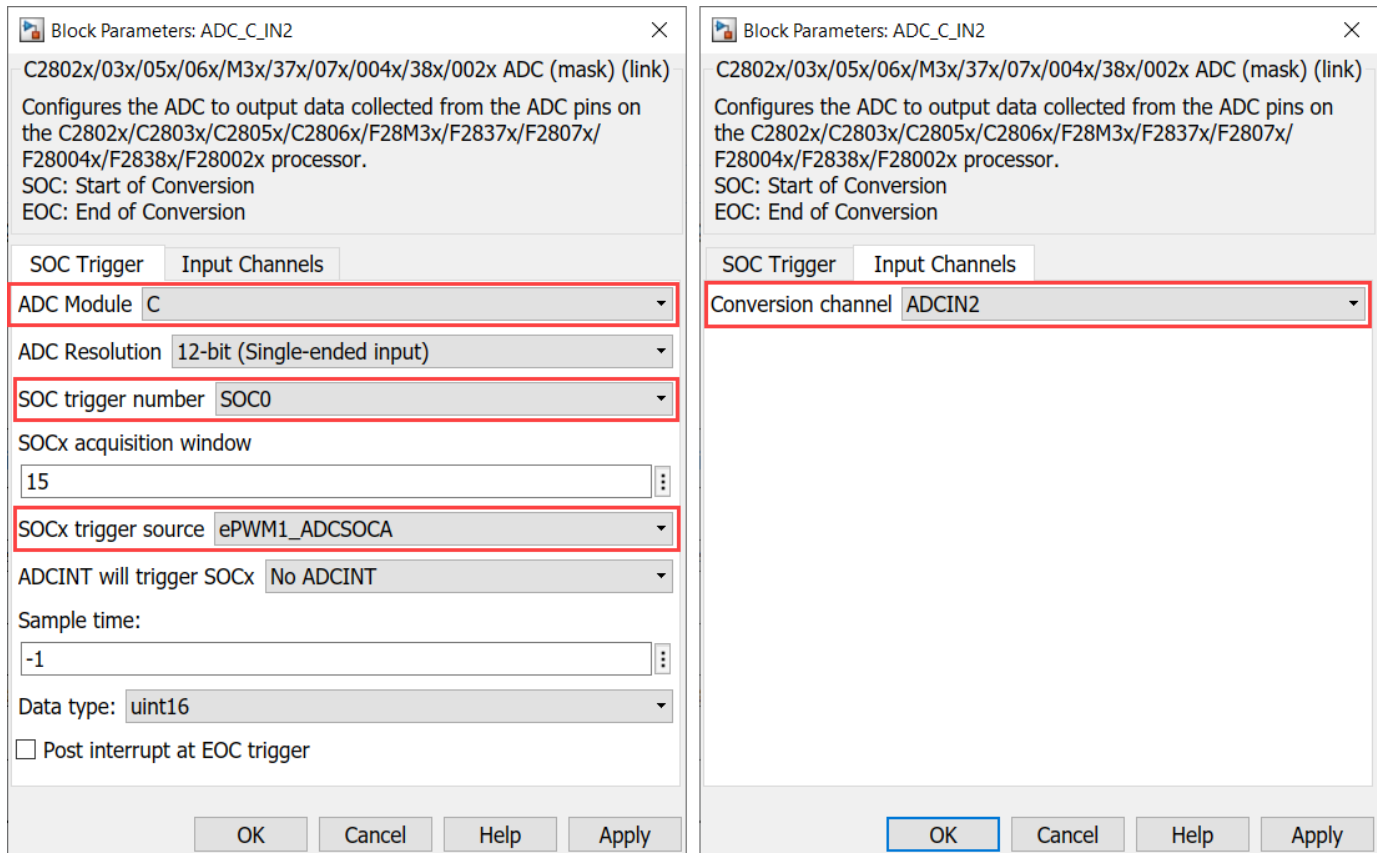
Select `ePWM1_ADCSOCA` as the SOC trigger source in the ADC driver blocks for phase-A and phase-B currents because the PWM library block triggers the start-of-conversion event SOC0 when the PWM counter equals the PWM period register.

In the ADC driver block for phase-B current (that uses **ADC module B**), select `ADCINT1`. This triggers an ADC interrupt at the end-of-conversion (EOC) event. When the ADC interrupt occurs, the FOC current control algorithm executes.

In the block parameters dialog box of ADC driver block for phase-A current, configure the ADC C module and channel 2 to read the phase-A current of the motor, as shown in this table.

Tab and Parameter in ADC Block	Settings
SOC Trigger > ADC Module	C
SOC Trigger > SOC trigger number	SOC0
SOC Trigger > SOC trigger source	ePWM1_ADCSOCA
Input Channels > Conversion channel	ADCIN2

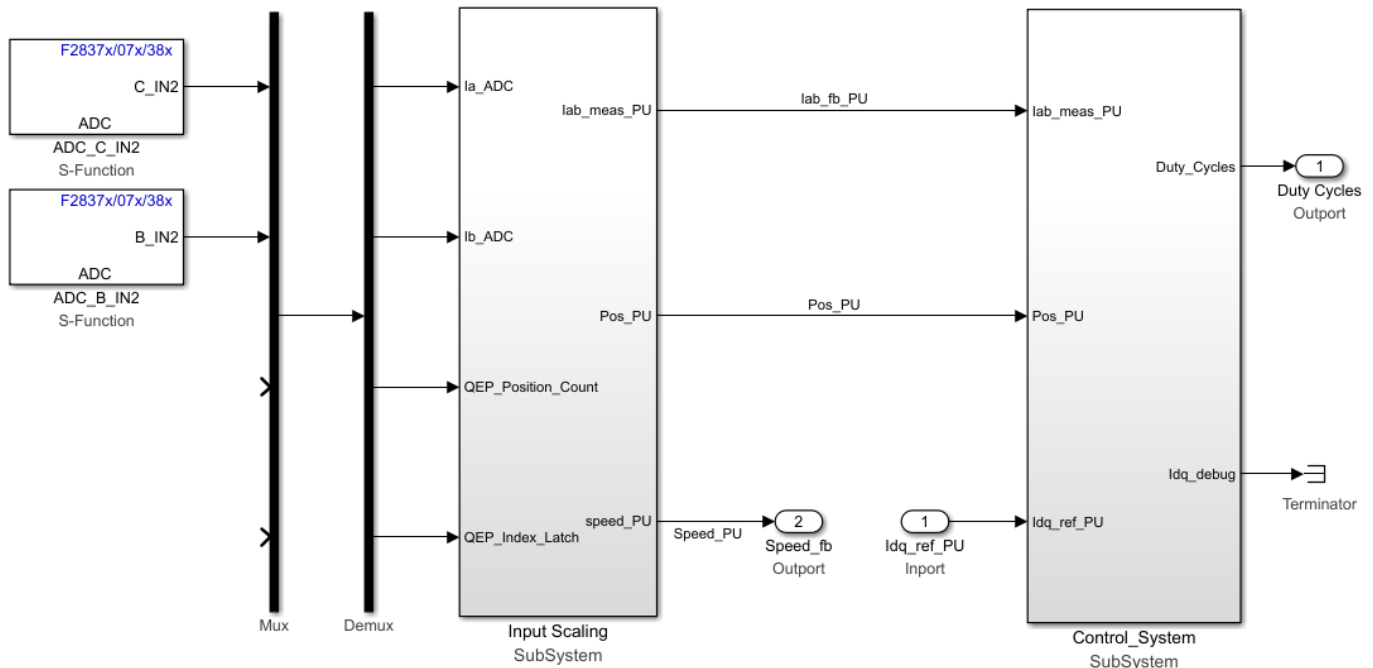
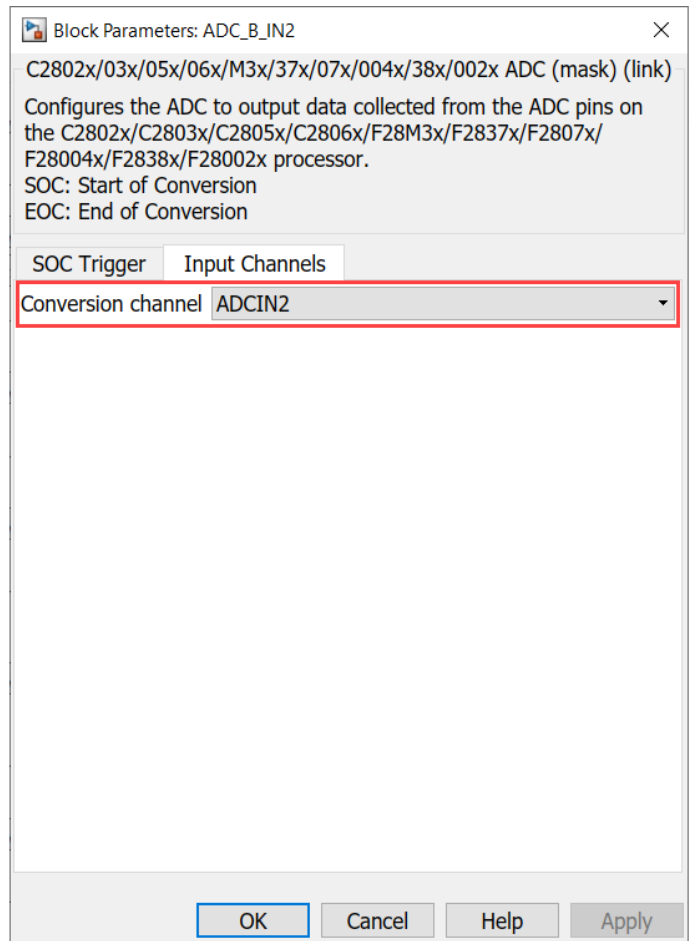
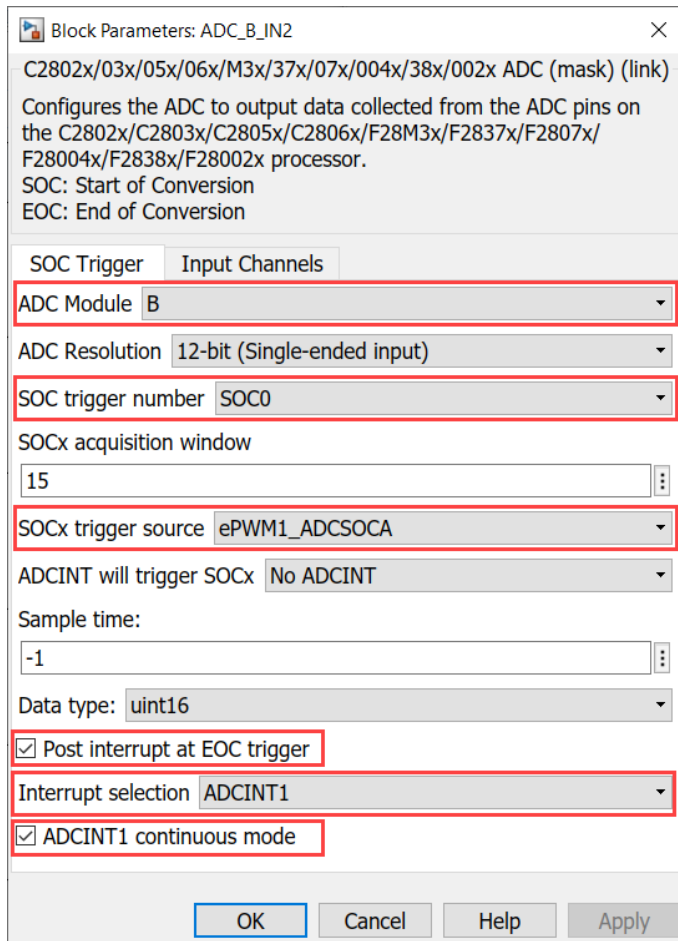
Rename the block as `ADC_C_IN2`.



In the block parameters dialog box of ADC driver block for phase-B current, configure the ADC B module and channel 2 to read phase-B current of the motor. In addition, configure ADC interrupt as ADCINT1, as shown in this table.

Tab and Parameter in ADC Block	Settings
SOC Trigger > ADC Module	B
SOC Trigger > SOC trigger number	SOC0
SOC Trigger > SOC trigger source	ePWM1_ADCSOCA
SOC Trigger > Post interrupt at EOC trigger	on
SOC Trigger > Interrupt selection	ADCINT1
SOC Trigger > ADCINT1 continuous mode	on
Input Channels > Conversion channel	ADCIN2

Rename the block as ADC_B_IN2.



Adding Quadrature Encoder Driver Block

In the Simulink Library Browser, add the eQEP block from **Embedded Coder Support Package for Texas Instruments C2000 Processors > F2837xD**.

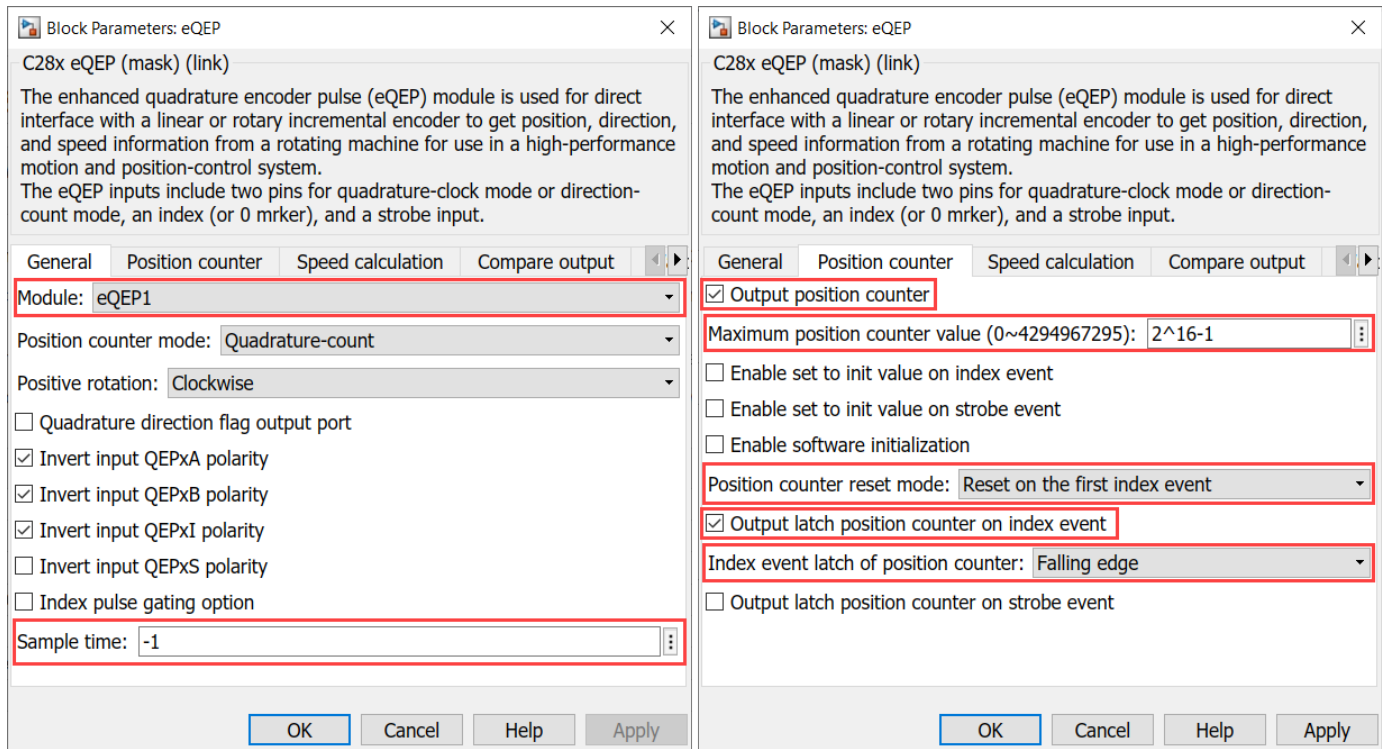
The eQEP block reads the quadrature encoder pulses and increments the position count. This block outputs the quadrature encoder pulse for the mechanical rotor position wraparound when the quadrature encoder index pulse is read.

See the section Quadrature Encoder Interface Configuration in “Model Configuration Parameters” for configurations related to the quadrature encoder.

In C28x eQEP block parameters dialog box, configure the quadrature encoder to read the quadrature encoder pulse count in the Texas Instruments processor and wrap the pulse counter output when index pulse is found as shown in this table.

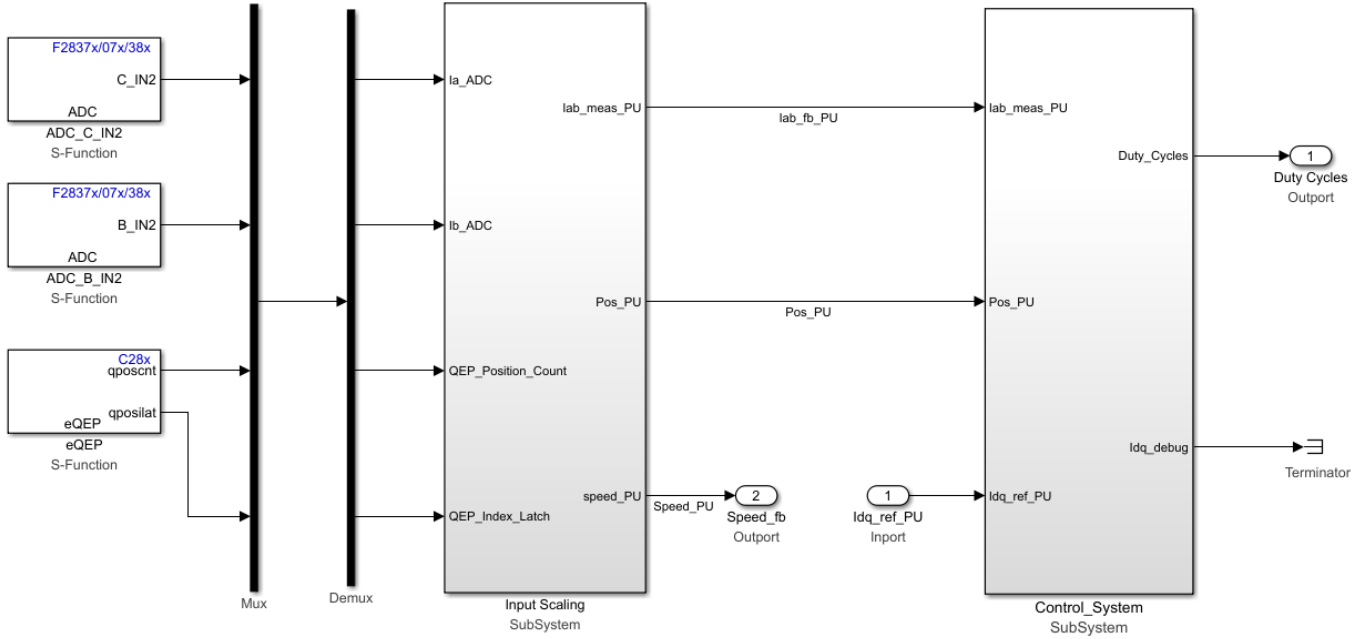
Tab and Parameter in eQEP Block	Settings
General > Module	eQEP1
General > Sample time	-1
Position counter > Output position counter	on
Position counter > Maximum position counter value (0~4294967295)	2¹⁶-1
Position counter > Position counter reset mode	Reset on the first index event
Position counter > Output latch position counter on index event	on
Position counter > Index event latch of position counter	Falling edge

Rename the block as eQEP.



eQEP1 module is selected because the quadrature encoder is connected to the QEP_A interface on the LaunchPadXL28379d hardware board. The sample time is -1 because the library block is function-call triggered by the ADC interrupt synchronously. The maximum position counter value is $2^{16} - 1$ because the position counter uses a 16-bit architecture in the library driver block. The position counter reset mode setting wraps the position count when the index pulse is read.

Add the eQEP driver block module to the `mcb_pmsm_foc_sim/Current control` subsystem as shown in this figure.



Add PWM Driver Block

In the Simulink Library Browser, add the ePWM block from **Embedded Coder Support Package for Texas Instruments C2000 Processors > F2837xD**.

Configure the ePWM1, ePWM2, and ePWM3 blocks for generating the PWM pulse. In the ePWM block parameters dialog box, specify the pulse width modulation (PWM) counter period register value calculated from CPU frequency and PWM frequency. For center-aligned PWM, divide the computed value by 2.

$$PWM \text{ counter period} = CPU \text{ clock frequency} / PWM \text{ frequency} / 2$$

For more details, see the TMS320f28379d processor ePWM peripheral.

In the F2837x/07x/004x/38x ePWM block parameters dialog box, update these settings to configure PWM1 to generate PWM pulses in the target hardware.

Tab and Parameter in ePWM Block	Settings
General > Module	ePWM1
General > Timer Period	Enter the PWM period value in the CPU clock cycle <ul style="list-style-type: none"> • PWM counter period = CPU clock frequency / PWM frequency / 2 • For LaunchPad 28379D, clock frequency is 200 MHz. For PWM frequency of 20 kHz, PWM counter period = 200e6 / 20e3 / 2; PWM counter period = 5000
Counter Compare > Specify CMPA via	Input port
Counter Compare > CMPA initial value	Enter the PWM counter period/ 2 (2500)
Counter Compare > Specify CMPB via	Input port
Counter Compare > CMPB initial value	Enter the PWM counter period/ 2 (2500)
Deadband unit > Use deadband for ePWM1A	on
Deadband unit > Use deadband for ePWM1B	on
Deadband unit > Deadband polarity	Active high complementary (AHC)
Deadband unit > Deadband Rising edge (RED) period (0~16383)	15
Deadband unit > Deadband Falling edge (FED) period (0~16383)	15
Event Trigger > Enable ADC start of conversion for module A check box (only for PWM1)	on
Event Trigger > Start of conversion for module A event selection (only for PWM1)	Counter equals to period (CTR=PRD)

Rename the block as ePWM1.

In the F2837x/07x/004x/38x ePWM block parameters dialog box, update the settings to configure PWM2 and PWM3 to generate PWM pulses in the target hardware. PWM2 and PWM3 are synchronized with PWM1. Follow ePWM1 configurations (other than **Event Trigger**) and add these configurations.

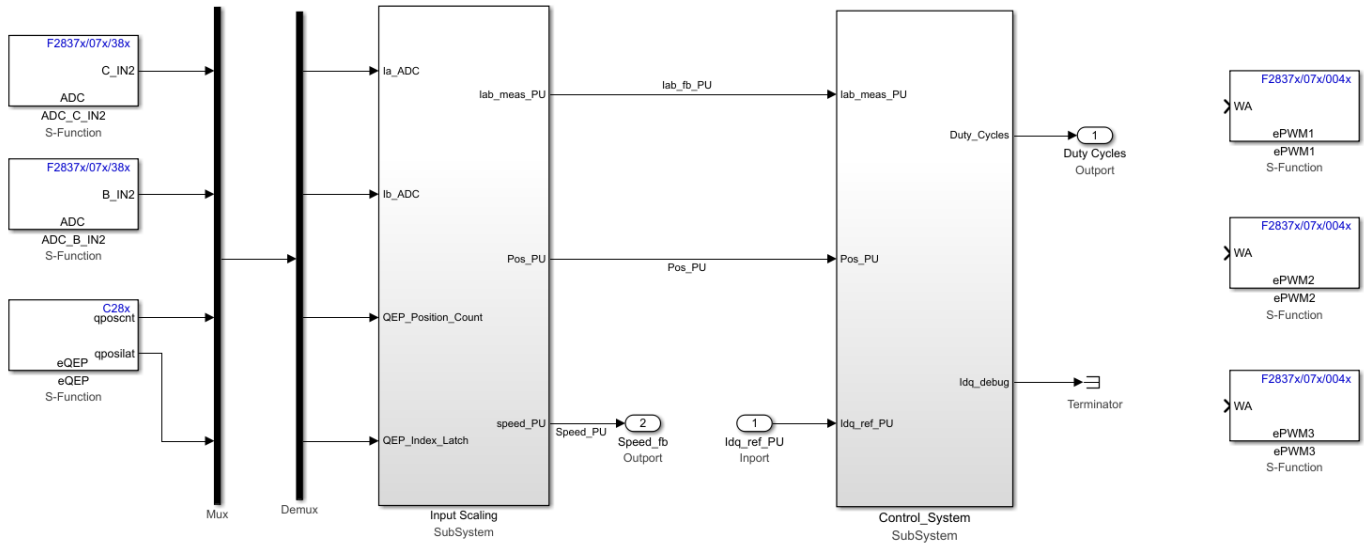
Tab and Parameter in ePWM Block	Settings
General > Module	ePWM2
General > Timer Period	Enter the PWM period value in the CPU clock cycle <ul style="list-style-type: none"> • PWM counter period = CPU clock frequency / PWM frequency / 2 • For LaunchPad 28379D, clock frequency is 200 MHz. For PWM frequency of 20 kHz, <p style="margin-left: 40px;">PWM counter period = $200e6 / 20e3 / 2$;</p> <p style="margin-left: 40px;">PWM counter period = 5000</p>
General > Synchronization action	Set counter to phase value specified via dialog
General > Counting direction after phase synchronization	Count up after sync
General > Phase offset value (TBPHS)	0
Counter Compare > Specify CMPA via	Input port
Counter Compare > CMPA initial value	Enter the PWM counter period/ 2 (2500)
Counter Compare > Specify CMPB via	Input port
Counter Compare > CMPB initial value	Enter the PWM counter period/ 2 (2500)
Deadband unit > Use deadband for ePWM1A	on
Deadband unit > Use deadband for ePWM1B	on
Deadband unit > Deadband polarity	Active high complementary (AHC)
Deadband unit > Deadband Rising edge (RED) period (0~16383)	15
Deadband unit > Deadband Falling edge (FED) period (0~16383)	15

Rename the blocks as ePWM2 and ePWM3.

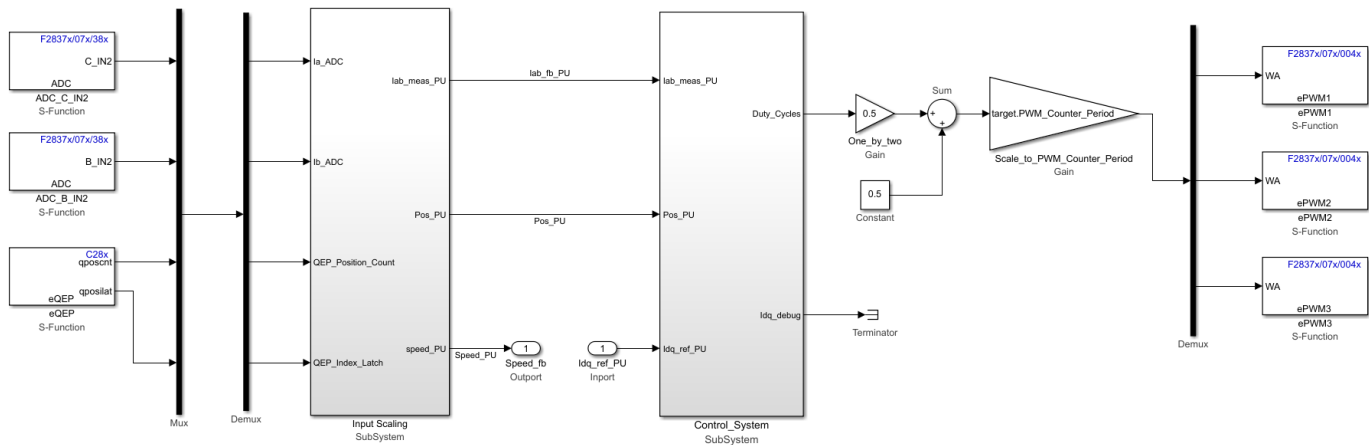
The range varies from 0 to *PWM_counter_period*. PWM outputs when PWM up-counter matches CMPA and PWM down-counter matches CMPB. By default, the system inputs a duty cycle of 50% by selecting PWM counter period / 2.

On the **Event Trigger** tab of PWM1 module, configure the ADC start of conversion event to begin when the PWM counter equals the PWM period.

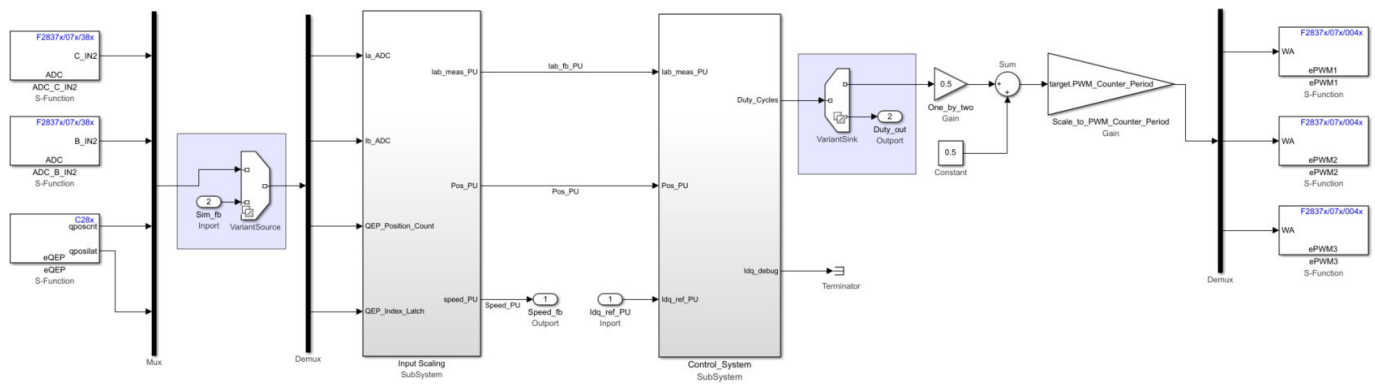
Synchronize the ePWM2 and ePWM3 blocks with the ePWM1 block by setting the synchronization timing to the moment when the PWM counter equals to zero in the ePWM2 and ePWM3 blocks.



The ePWM blocks expect the duty cycle value to range from 0 to the PWM counter period value (5000). The Control_System subsystem outputs the PWM in the range -1 to 1. The model needs to scale the output to 0 to 5000 (PWM counter period value).



For simulation, add a variant source/sink to the hardware driver block for simulation and code generation.



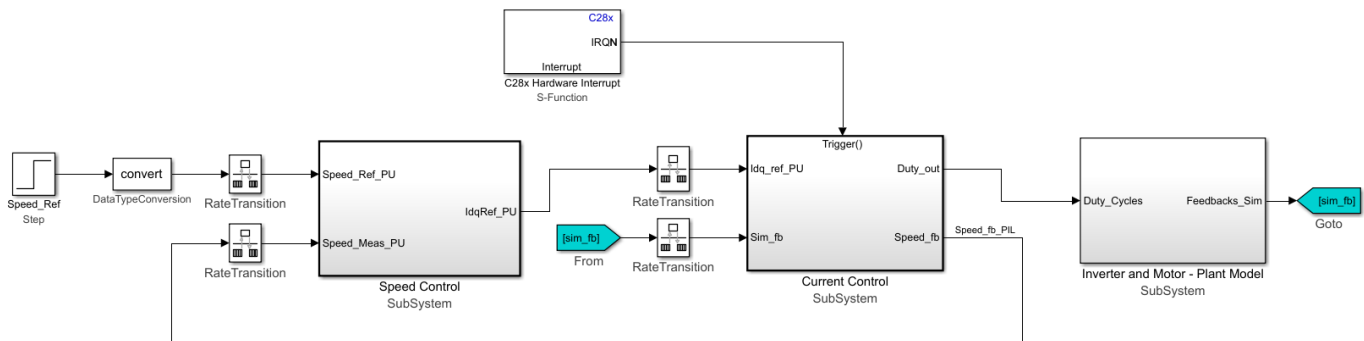
Add Hardware Interrupt Trigger Block for Current Control Loop

In the Simulink Library Browser, select and add the C28x Hardware Interrupt block from **Embedded Coder Support Package for Texas Instruments C2000 Processors > Scheduling**.

In the block parameters dialog box, update the settings to configure the hardware interrupt ADCINT1. Also, identify and update the CPU and PIE interrupts for the hardware interrupt ADCINT1.

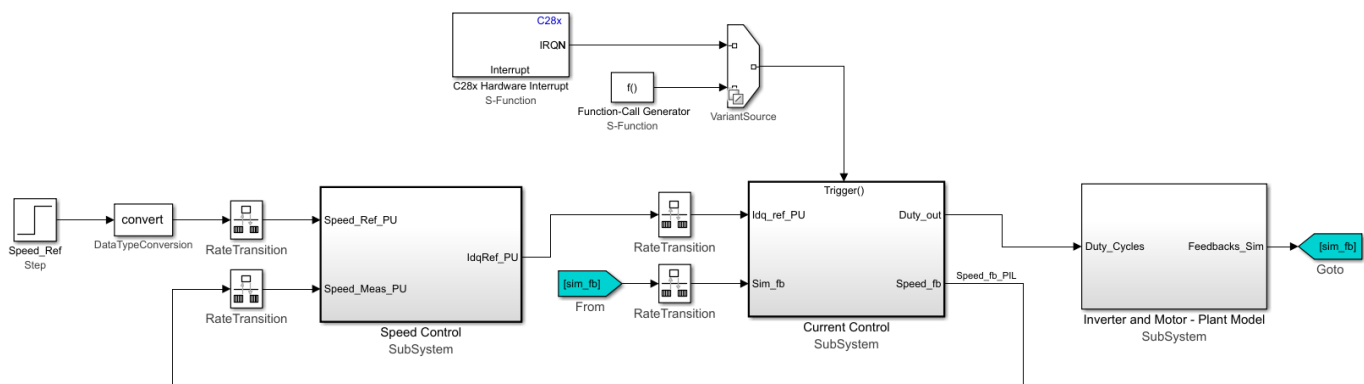
Parameter in C28x Hardware Interrupt Block	Settings
CPU interrupt numbers	[1]
PIE interrupt numbers	[2]

In the current control subsystem, add a Trigger block and set the **Trigger type** block parameter to function-call. Connect this subsystem trigger input to the C28x Hardware Interrupt block as shown in this figure.



In the Rate Transition block input to Current Control subsystem, change the **Output port sample time** to -1.

Add a Function-Call Generator block in variant source_ref to support the model simulation. In the Function-Call Generator block, set the **Sample time** parameter as T_s ($50e-6$).



Simulate the model with the updated driver blocks and check the simulation results in the Simulation Data Inspector. Variants ensure that ADC, PWM drivers, and interrupts are not active during simulation.

Run in Open-Loop and Switch to Closed-Loop

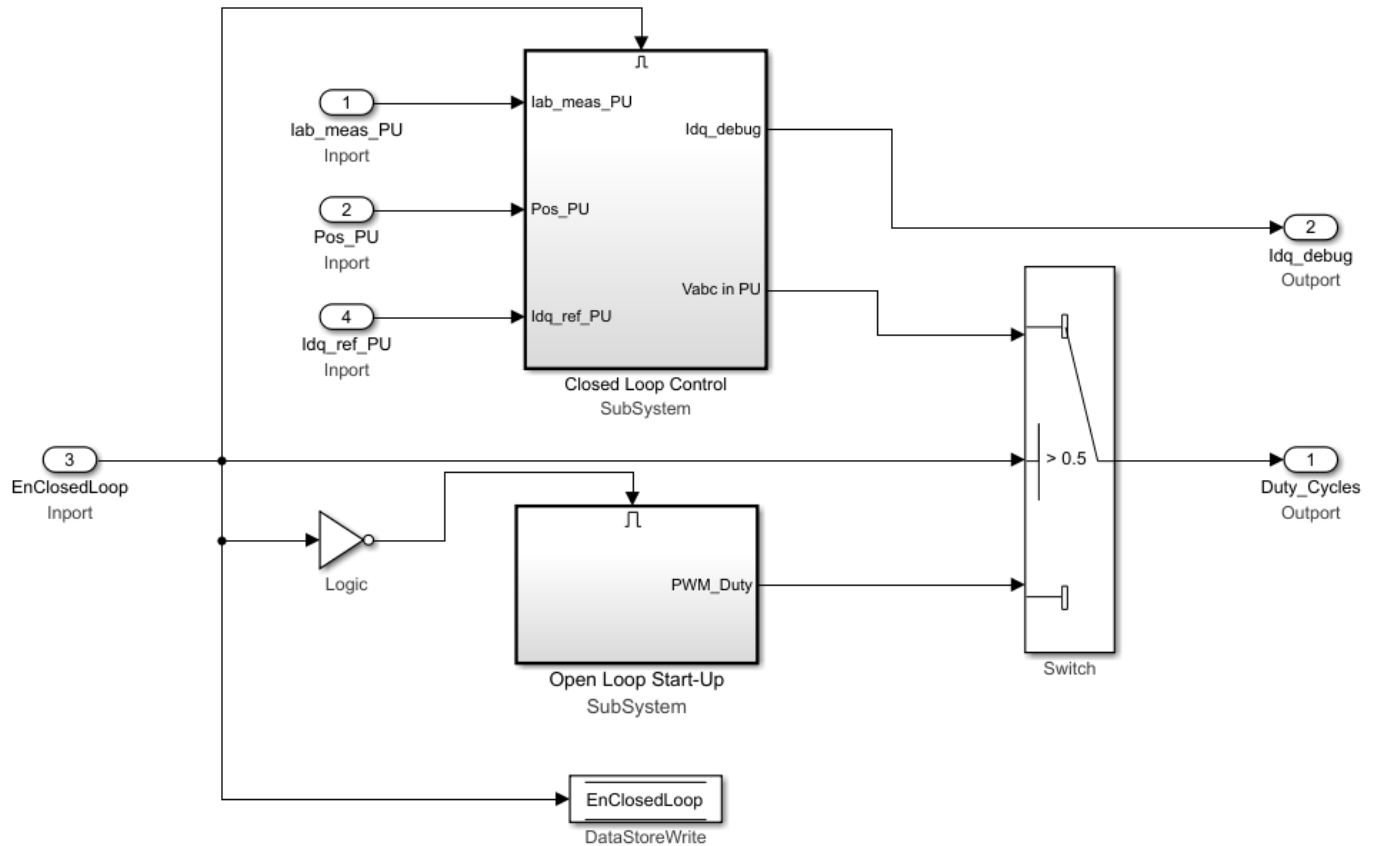
When operating a permanent magnet synchronous motor (PMSM) with a quadrature encoder sensor, we need an initial position to start running the motor. Because we do not have a method to determine the initial position at the beginning (before starting the motor), run the motor using open-loop control and ensure that the quadrature encoder index pulse is read at least once. At the quadrature encoder index pulse, the quadrature encoder sensor resets its position to align with the mechanical angle of the motor. The motor switches from an open-loop run to closed-loop speed control to maintain the reference speed. This step is only applicable for a quadrature encoder sensor (and is not needed for a Hall position sensor). A Hall sensor outputs the initial position of the rotor segment from the Hall signal port inputs.

Follow these steps to implement an open-loop motor run with a transition to closed-loop control:

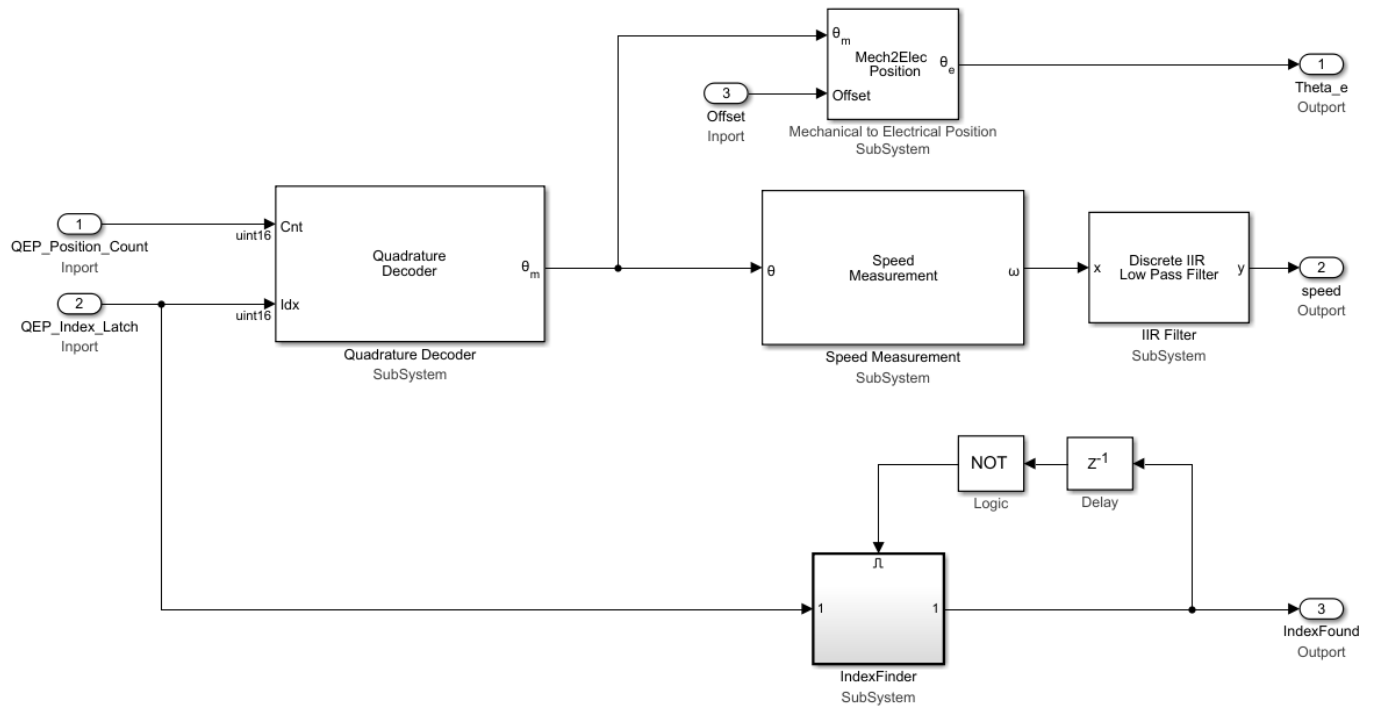
- 1 Copy the `mcb_pmsm_foc_qep_f28379d/Current Control/Control_system` subsystem to your model. This adds the algorithm to run the motor in open-loop. This subsystem switches the control from open-loop to closed-loop if *EnClosedLoop* input is 1. Add an input port **EnClosedLoop**.

Addition of the **Open Loop Start-Up** subsystem adds the Data Store Read blocks for *Enable* and *SpeedRef*. In addition, add the Data Store Memory blocks for *Enable*, *EnClosedLoop*, and *SpeedRef* at the topmost level of the model.

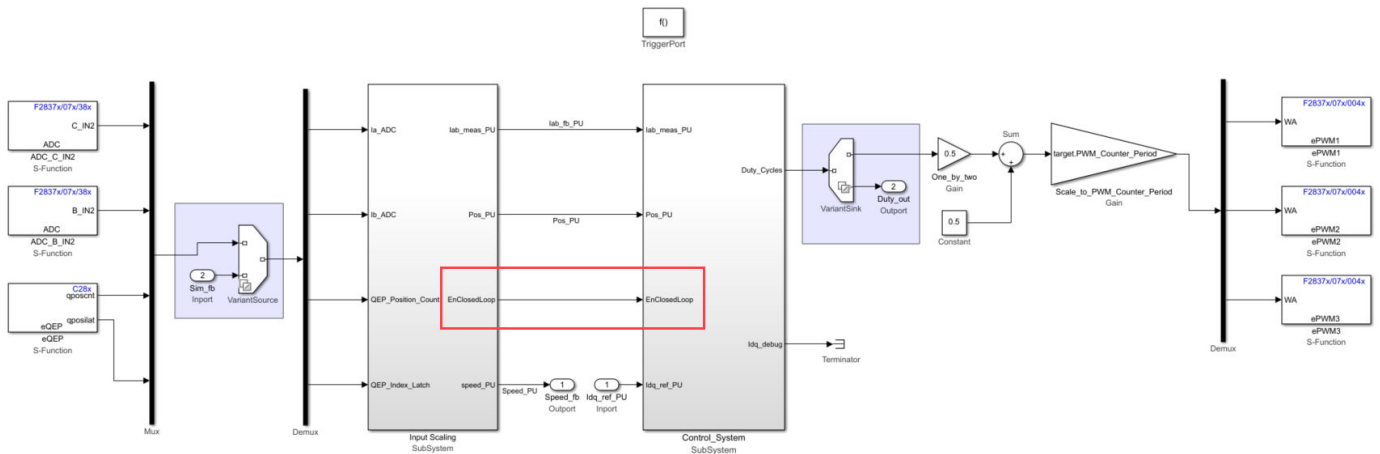
When the open-loop run begins, the sign of *SpeedRef* (for algorithm details, see the **Open Loop Start-Up** subsystem) decides the direction of the initial motor run. If *SpeedRef* is negative, the motor spins in the opposite direction during the open-loop run.



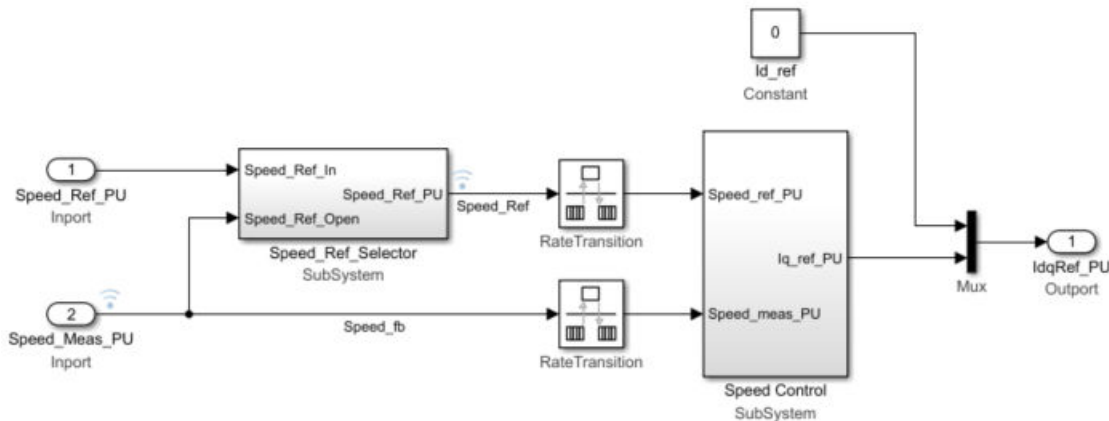
- Copy the `mcb_pmsm_foc_qep_f28379d/Current Control/Input Scaling/Calculate Position and Speed` subsystem to your model. This adds the `IndexFinder` subsystem to your model. When quadrature encoder index pulse is detected for the first time, this subsystem sets the `IndexFound` port to 1. Add an output port (that is connected to the `IndexFound` port) to the **Calculate Position and Speed** subsystem and rename it to `EnClosedLoop`.



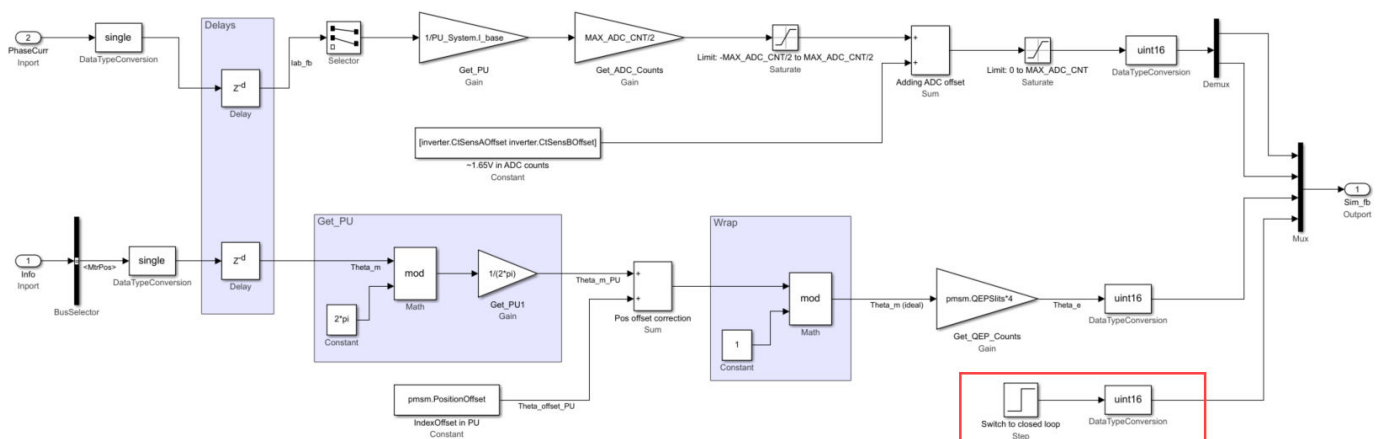
- 3 Connect the output port `EnClosedLoop` from the **Input Scaling** subsystem to the input port `EnClosedLoop` in the **Control_System** subsystem as shown in this figure.



- 4 Copy the `mcb_pmsm_foc_qep_f28379d/Speed Control/Speed_Ref_Selector` subsystem to your model and integrate it with the speed controller subsystem. When the closed-loop control begins, this subsystem provides the `Speed_Ref` output signal. For a smooth transition from open-loop to closed-loop, the speed measured is used as the speed reference during the open-loop run. Add a Data Store Write block `SpeedRef` to the `PI_Controller_Speed` input port.



- In the plant model, add a step input to simulate the IndexFinder block for simulation. Rename the step input to Switch to closed loop. See the `mcb_pmsm_foc_qep_f28379d/Inverter and Motor - Plant Model/Sensor_Measurments` subsystem to see how the step input switches to closed-loop. Select the step time of 0.1 and sample time of T_{s_motor} .



- Create Data Store Memory blocks for `EnClosedLoop`, `Enable`, and `SpeedRef`. `Enable` block is used to reset the PI integrator before running the motor.

Add these default values in the Data Store Memory blocks:

- `Enable = 1`
- `EnClosedLoop = 0`
- `SpeedRef = 0.25`

The Data Store Memory blocks are used to share data across the subsystem.

- Run the simulation and observe the speed reference and the speed feedback signals.

Model Configuration and Hardware Deployment

Use these steps to select the target hardware in the Configuration Parameters dialog box.

- 1 In the Simulink model, click **Hardware > Hardware Settings** to open the Configuration Parameters dialog box.
- 2 Open the **Hardware Implementation** tab and set **Hardware board** to **TI Delfino F28379D LaunchPad**.

For any other custom board, navigate to the **Hardware Implementation** tab of the Configuration Parameters dialog box and select the appropriate processor and edit the peripheral details in **Hardware board settings > Target hardware resources**.

For the solver and quadrature encoder interface configuration details, see “Model Configuration Parameters”.

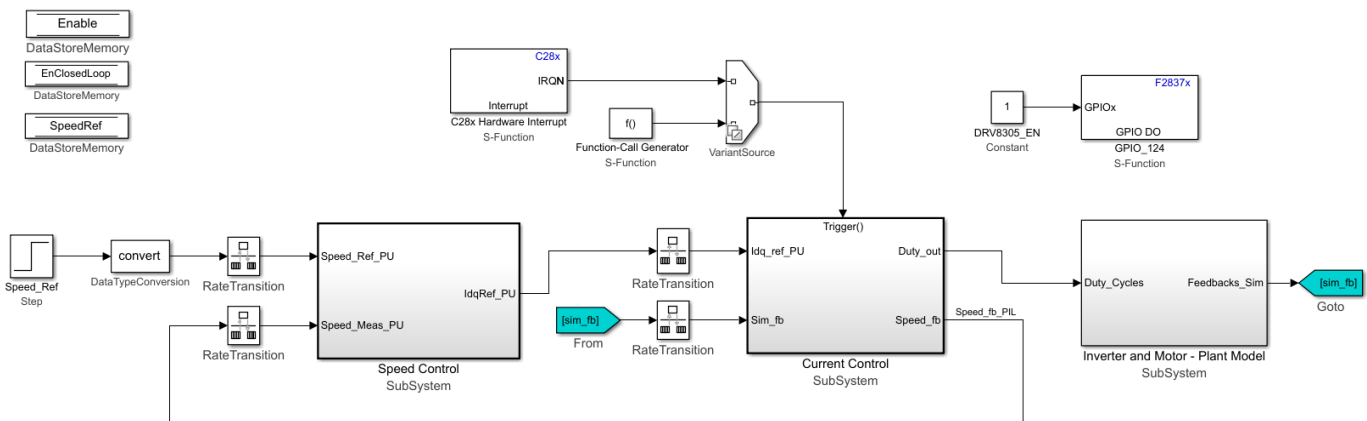
Connect the Texas Instruments BOOSTXL-DRV8305 board and QEP connector to the Texas Instruments LaunchPad XL hardware board. For hardware connection details related to Texas Instruments C2000 LaunchPadXL, see “Hardware Connections”. The BOOSTXL-DRV8305 (attached to the LaunchPadXL board) requires an enable signal. This signal is connected to the GPIO124 pin of the processor.

In the Simulink Library Browser, add **Embedded Coder Support Package for Texas Instruments C2000 Processors > F2837xD > Digital Output**. In the Digital Output block parameters dialog box, change these settings:

Parameter in Digital Output Block	Settings
GPIO Group	GPIO120~GPIO127
GPIO124	on

Rename the block as GPIO_124.

Add a constant block with the value 1 as an input to the GPIO124 block as shown in this figure.



On the **Hardware** tab of the Simulink model, select **Build, Deploy & Start**. This generates the C code, CCS project, and a target-specific .out file. The system uses serial communication to download this target specific .out file to the target hardware and runs the downloaded algorithm in the hardware.

When the model is deployed to the target, the motor runs in open-loop and then runs in closed-loop speed control. This example recommends that you use serial communication to monitor and debug the signals. For details about implementing serial receive and transmit communications between the host and target models, see the example model `mcb_pmsm_foc_qep_f28379d`. From the Serial Receive block, update the *Enable* Data Store Memory block to start and stop the motor using the serial commands received from the host model.

Validate System

In this section...

“Calculate Physical Motor Load in Target Hardware” on page 2-26

“Compare Speed Controller Response During Simulation With Target Hardware Results” on page 2-27

“Compare Current Controller Response During Simulation With Target Hardware Results” on page 2-29

This section explains how to evaluate the accuracy of the plant (motor and inverter) model of the physical motor and load connected to the motor. Validate the plant model and verify that the results are close to the physical system measurements before using the plant model for implementing advanced algorithms. You can validate the system by comparing the step response of speed control and current control during simulation and after deployment to the target hardware connected to the motor.

Use the example “Tune Control Parameter Gains in Hardware and Validate Plant” to measure the step response of the current and speed controllers. The host model in this example communicates the current reference to the target hardware and measures the step response of the current controller.

- You can use any speed control example from Motor Control Blockset to validate the system.
- Validate speed control by comparing the step response during simulation with the hardware test values.
- Validate the d -axis current control by electrically or mechanically locking the rotor and comparing the step response during simulation with the hardware test results.

You can use another method to validate the d -axis current control. Run the motor at a constant speed and provide a step change in the reference d -axis current. This requires two modifications in the speed control subsystem of the target model. Set a constant speed reference input. Command I_d reference from the host model. Compare the step response of the d -axis current during simulation with the response obtained during the hardware tests.

- Validate the q -axis current control by mechanically coupling the motor with an external dynamometer running in speed control. This requires two modifications in the speed control subsystem of the target model. Discard the I_d and I_q reference from the speed PI controller output. Command I_d reference from the host model. Compare the step response of the q -axis current during simulation with the response obtained during the hardware tests.

Warning When capturing the step response in d -axis current control, always use a positive step. Negative values of I_d can damage the permanent magnet in the PMSM.

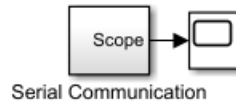
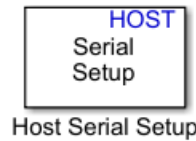
Host model for Control Parameter Gain Tuning (Manual) in Hardware and Plant Validation

Prerequisites:

1. Deploy the target model to the hardware [mcb_pmsm_operating_mode_f28379d](#)
2. You should see and verify the variables from the target model in the base workspace.

Steps:

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



Control

Select Motor operating mode _____

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

Operating Mode Variables

Open-loop mode

Voltage ref (PU) Speed ref (PU)

Motor torque control mode

 Unlock Pos lock

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

Motor speed control mode

Speed Ref (PU)

Monitor

Monitor Signal #1 _____

- V_alpha
- V_beta
- I_alpha
- I_beta
- Va_out
- Vb_out
- Vc_out
- Ia_meas
- Ib_meas
- Id_ref
- Id_meas
- Vd_ctrl_out
- Iq_ref
- Iq_meas
- Vq_ctrl_out
- Position_meas
- Speed_ref
- Speed_meas

Monitor Signal #2 _____

- V_alpha
- V_beta
- I_alpha
- I_beta
- Va_out
- Vb_out
- Vc_out
- Ia_meas
- Ib_meas
- Id_ref
- Id_meas
- Vd_ctrl_out
- Iq_ref
- Iq_meas
- Vq_ctrl_out
- Position_meas
- Speed_ref
- Speed_meas

Control loop gains

d-axis current controller

Kp Gain Ki Gain

q-axis current controller

Kp Gain_ Ki Gain_

Speed controller

Kp Gain__ Ki Gain__

Copyright 2020 The MathWorks, Inc.

See the example “Tune Control Parameter Gains in Hardware and Validate Plant” to deploy the model to the hardware. Perform motor parameter estimation because an accurate plant model is important to ensure that the simulation results match the hardware test results.

Calculate Physical Motor Load in Target Hardware

Before comparing the controller responses during simulation with the responses obtained after target hardware deployment, the load torque in plant simulation must match the motor load in the physical system. Follow these steps to calculate the load torque in the physical system and update the calculated load torque in the plant model.

- 1 Run the host model to connect it to the target hardware through serial communication.
- 2 Set **Select Motor operating mode** to **Speed control**.

The motor spins in speed control.

- 3 Select **Id_meas** in **Monitor Signal #1** and **Iq_meas** in **Monitor signal #2**. Read the Id_meas and Iq_meas values from the scope.
- 4 Convert the per-unit (PU) current to Amperes by multiplying it with PU_System.I_base.
- 5 Calculate the load torque in Nm using this equation:

$$T_{load} = 1.5 \times pole_pair \times [(flux_pm \cdot I_q) + (L_d - L_q)I_d \cdot I_q]$$

where,

$flux_pm$ = Permanent magnet flux linkage (pmsm.Flux_PM)

L_d, L_q = Inductance in Henry (pmsm.Ld, pmsm.Lq)

I_d, I_q = Current measured in Amperes

I_{d_meas} , the measured I_d current (in PU), equals 0.

- 6 In the mcb_pmsm_operating_mode_f28379d/Motor and Inverter/Plant Model (sim) sub system, provide the calculated load torque value as an input to the **LdTrq** port of the PMSM motor block.

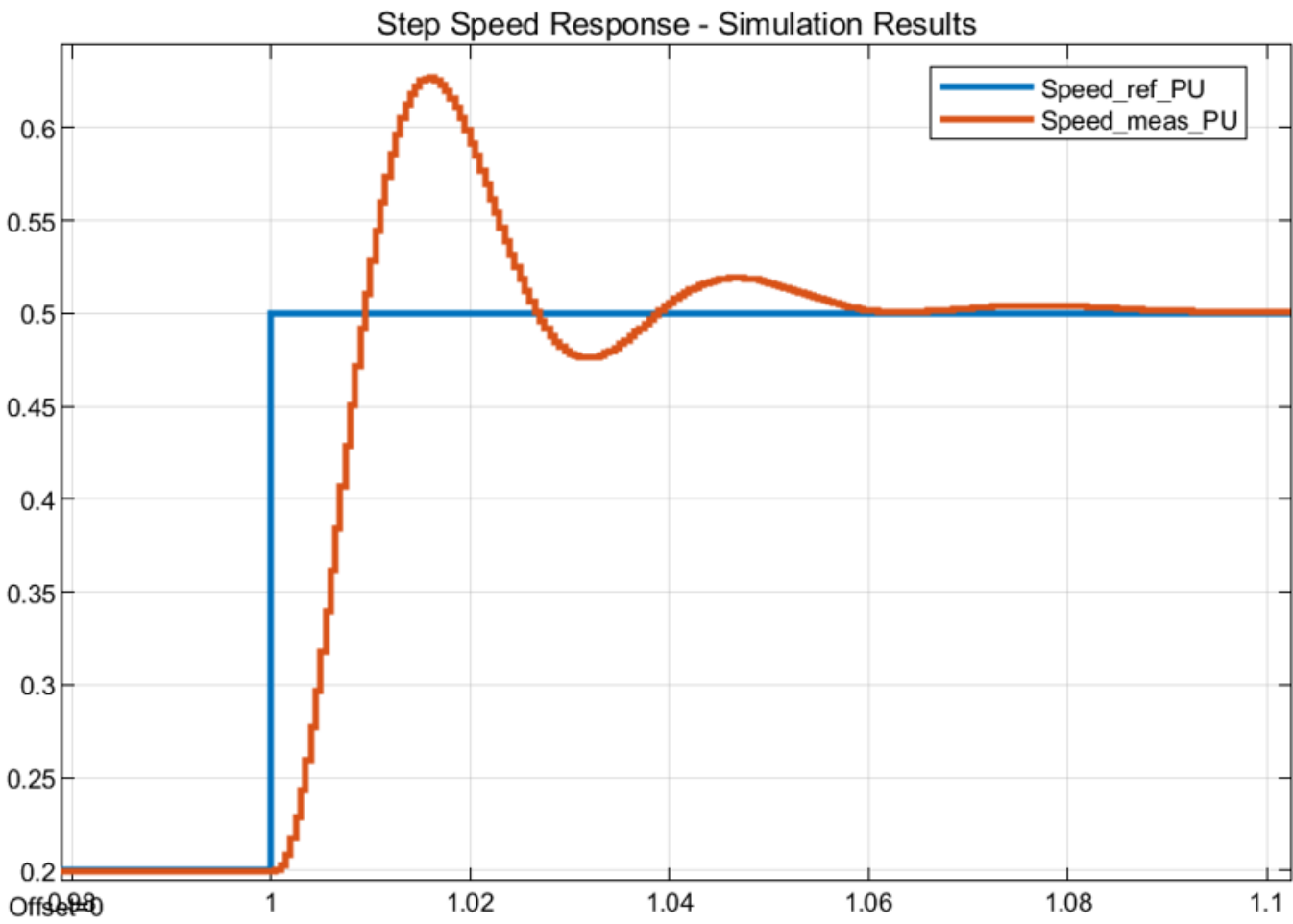
Compare Speed Controller Response During Simulation With Target Hardware Results

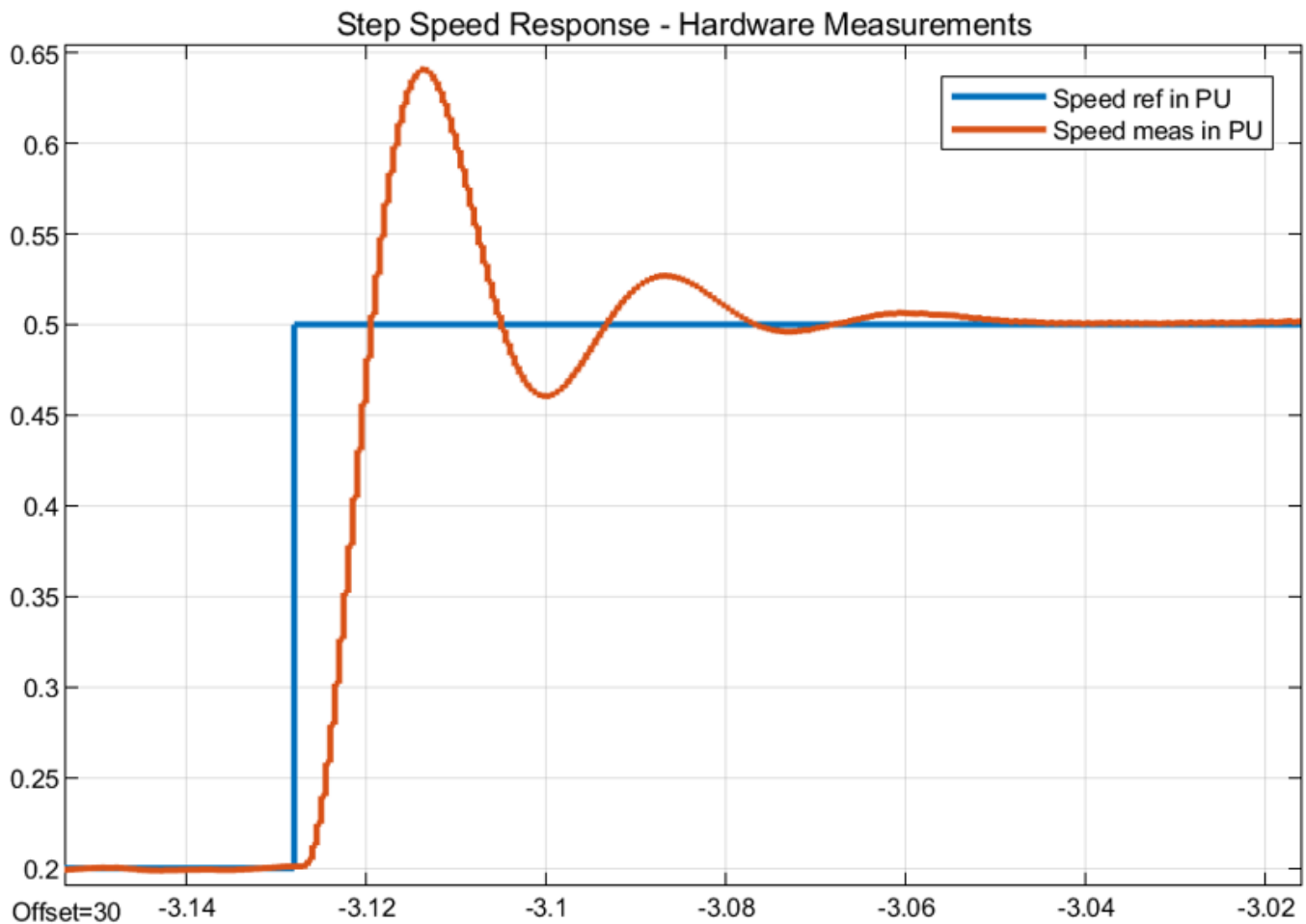
During simulation, provide a speed step input and note the speed response. On the target hardware, command the speed reference step input and observe the speed feedback. Compare the resulting step response during simulation with the response obtained from target hardware to determine the accuracy of the plant model.

- 1 Simulate the model mcb_pmsm_operating_mode_f28379d. Plot the reference speed and the measured speed signals. By default, this example provides a step input of 0.2 to 0.5 to the simulation model.
- 2 Run the host model to communicate with the target hardware.
- 3 Change **Select Motor operating mode** from **Stop** to **Speed control**.
- 4 In the host model, select **Speed_ref** in **Monitor Signal#1** and **Speed_meas** in **Monitor Signal#2**.
- 5 Open the scope in the host model.
- 6 In host model interface, change the speed_ref from 0.2 to 0.5 and observe the step change in the scope.
- 7 Compare the step response obtained from the hardware with the simulation results.

Step Response Analysis for Speed Controller

Compare the step response obtained from simulation with the measurements obtained from the target hardware. The results may vary depending on the tolerances in the plant model. Generally, simulation results are close to the values measured on the target hardware.





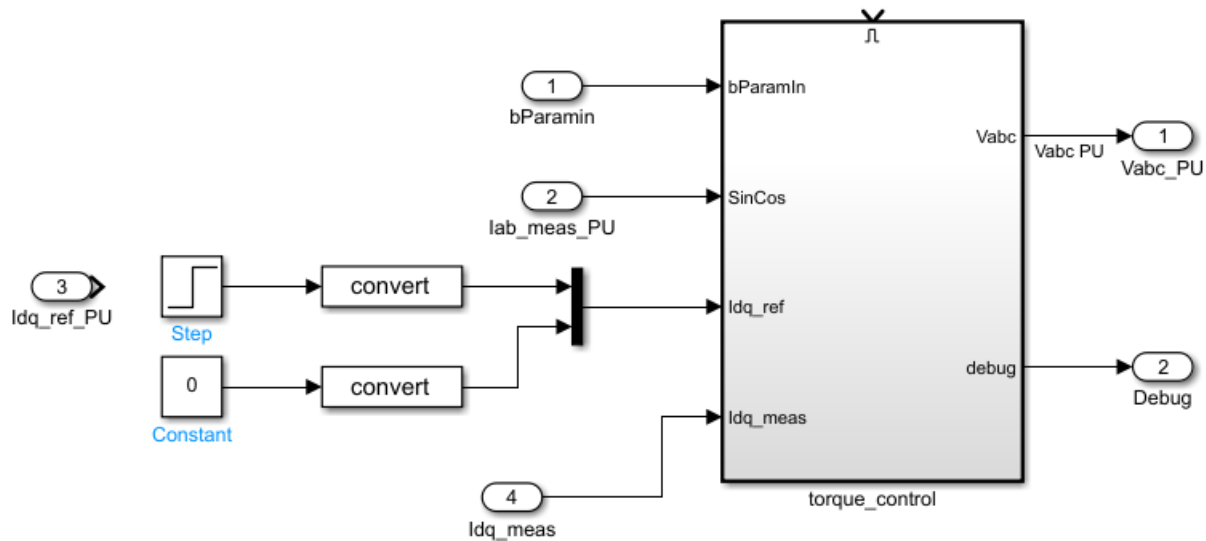
	Peak overshoot (%)	Peak time (ms)	Rise time (ms)	Settling time (ms)
Simulation results	20.13%	16.023	5.561	61.027
Hardware results	22 %	14.324	5.041	51.148

Compare Current Controller Response During Simulation With Target Hardware Results

During simulation, provide a step current reference and note the current response. This example needs some changes to simulate the current reference step input. Follow these steps to perform the model changes. When using the target hardware, command the current reference step input and observe the current feedback. Compare the resulting step response in simulation with the response obtained from the target hardware to determine the accuracy of the plant model.

- 1 For hardware measurements, run the host model.
- 2 Change **Select Motor operating mode** from **Stop** to **Torque control**.
- 3 Select **Id_ref** in **Monitor Signal#1** and **Id_meas** in **Monitor Signal#2** in the host model.
- 4 Open the scope in the host model.

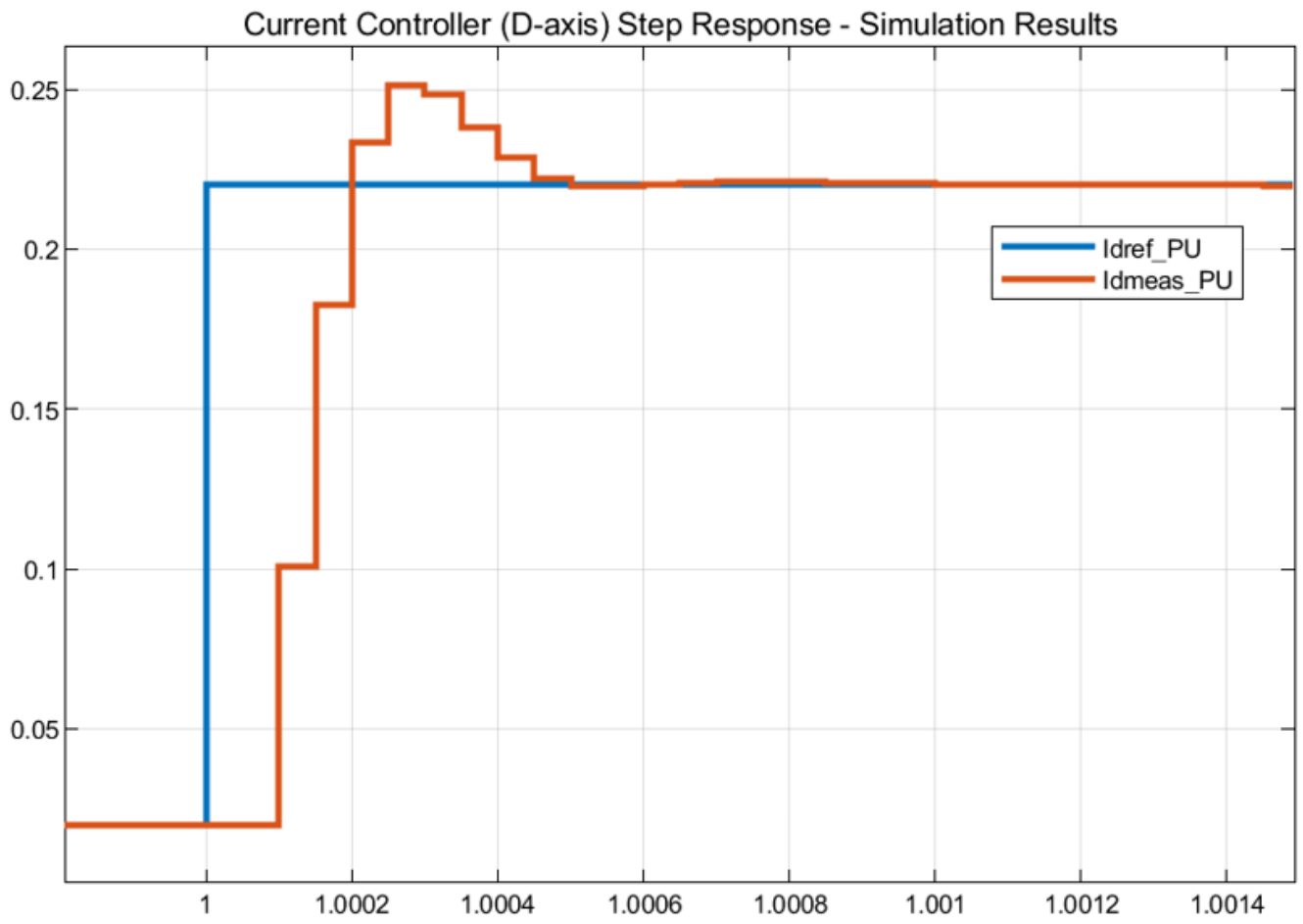
- 5 Change I_{d_ref} from 0.02 to 0.22 and observe the step change in the scope. Ensure that the motor is not running. The scope displays the step response for the I_{d_ref} input.
- 6 For simulation, make these two changes in the model. In the `mcb_pmsm_operating_mode_f28379d/TorqueControl/Control Modes/torque_control` subsystem add a step input for the d -axis current controller. Choose a step input of 0.02 to 0.22 at 1 second. Select time sample as -1. In the data-type conversion block, select the output datatype as `fixdt(1,32,17)`.

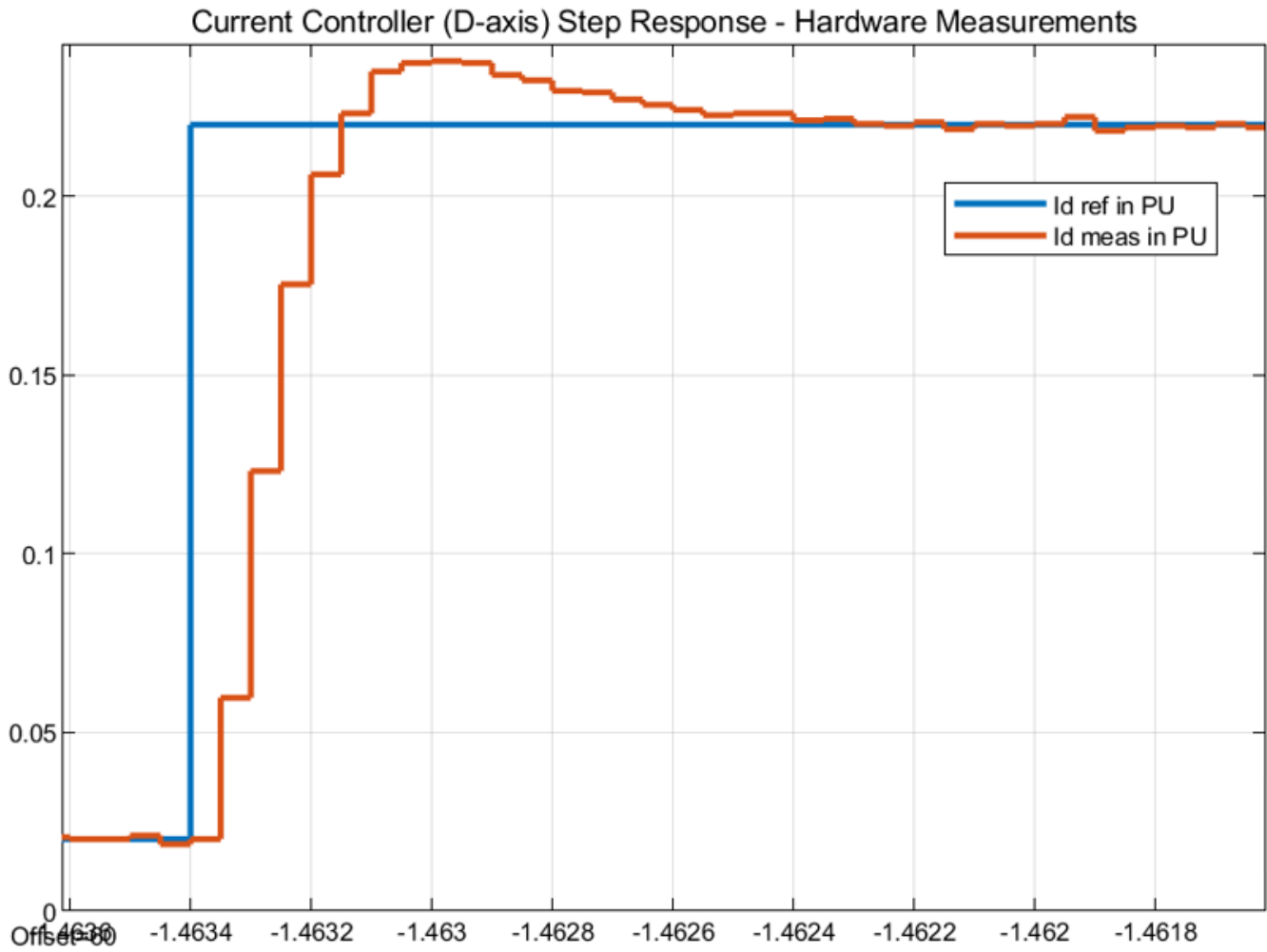


- 7 In the PMSM motor block available in the `mcb_pmsm_operating_mode_f28379d/Motor and Inverter/Plant Model (sim)` subsystem, change the **Mechanical input configuration** to **Speed** and input 0 to the **Spd** input port.
- 8 Run the simulation and measure the `Idref_PU` and `Idmeas_PU` values in the Simulation Data Inspector.
- 9 Compare the step response obtained from the hardware with the simulation results.

Step Response Analysis for d-axis Current Controller

Compare the scope results obtained from simulation with the measurements from the target hardware. The results may vary depending on the tolerances in the plant model. With an accurate plant model, the simulation results are closer to the measured results from the target hardware.





	Peak overshoot (%)	Peak time (μs)	Rise time (μs)	Settling time (μs)
Simulation results	14 %	300	150	500
Hardware results	8.18 %	400	150	800

The accuracy of the plant model improves the accuracy of simulation, and therefore, it helps match the simulation results to the hardware test results.

Tip If the simulation results differ considerably from the hardware measurements, verify the delay and scaling factor in the plant model.

Note For the q -axis current controller, align the motor to the d -axis and mechanically lock the rotor. Follow this for the d -axis current controller for comparative analysis. You can achieve external mechanical locking through the mechanical braking system or by coupling with a dynamo-meter motor running in speed control.

Plant Modeling

- “Creating Plant Model Using Motor Control Blockset” on page 3-2
- “Use PMSM Block and Motor Parameters to Design Plant Model” on page 3-3
- “Add Average-Value Inverter Block” on page 3-5
- “Create Motor Phase Current Sensing and Signal Conditioning Subsystem” on page 3-6
- “Create Position Sensing Subsystem” on page 3-7
- “Add Delay in Plant Model” on page 3-8
- “Integrate Blocks and Subsystems” on page 3-9

Creating Plant Model Using Motor Control Blockset

An accurate plant model is a vital part of motor control system development. After creating an accurate plant model, you can verify the functionality of the control system, conduct closed-loop model-in-the-loop tests, tune the controller gains using simulation, and optimize the algorithm before you deploy the model in the actual plant.

When you create a plant model using Motor Control Blockset, you model these components to simulate the functional behavior in a simulation environment:

- Permanent Magnet Synchronous Motor (PMSM)
- Average-value Inverter
- Sensors and signal conditioning circuits
- Processor peripherals: Analog-to-Digital converter (ADC) and Pulse-width-modulator (PWM)

You can verify the functionality of the plant model you create by:

- 1 Reading the normalized PWM duty cycle from the control algorithm.
- 2 Simulating the motor for the connected load.
- 3 Obtaining the output motor phase current (in terms of ADC counts) and the output motor position (in terms of encoder pulse counts) from the simulation.

The workflow to create a plant model involves these steps.

Note See the plant model in the `mcb_pmsm_foc_qep_f28379d.slx` model that is used in the example “Field-Oriented Control of PMSM Using Quadrature Encoder”.

- 1 “Use PMSM Block and Motor Parameters to Design Plant Model” on page 3-3
- 2 “Add Average-Value Inverter Block” on page 3-5
- 3 “Create Motor Phase Current Sensing and Signal Conditioning Subsystem” on page 3-6
- 4 “Create Position Sensing Subsystem” on page 3-7
- 5 “Add Delay in Plant Model” on page 3-8
- 6 “Integrate Blocks and Subsystems” on page 3-9

Use PMSM Block and Motor Parameters to Design Plant Model

You can use the surface mount or interior PMSM blocks from Motor Control Blockset in two ways to create a plant model.

- Estimate motor parameters by using Motor Control Blockset and open a Simulink model with PMSM motor block (auto-populated with estimated parameters):

The Motor Control Blockset parameter estimation workflow helps you to determine the motor parameters by performing a series of tests on the motor. For details, see “Estimate PMSM Parameters Using Recommended Hardware”. After successfully estimating the motor parameters, click **Open Model** in the parameter estimation host model. A new model opens with the Interior PMSM block updated with the estimated motor parameters.

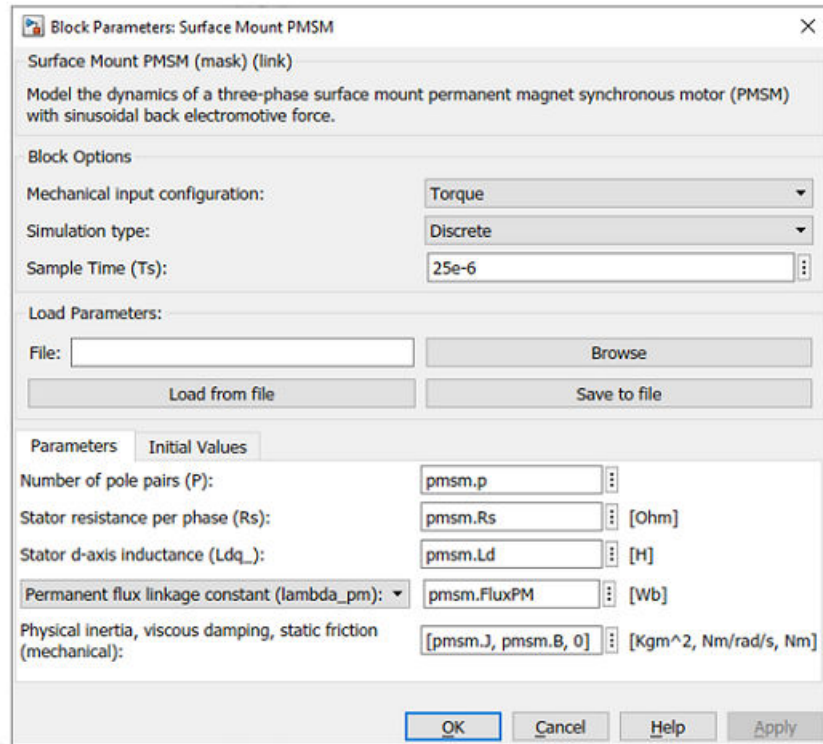
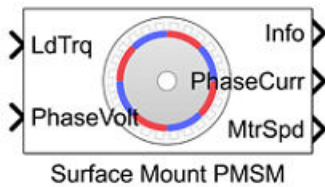
The screenshot displays the Motor Control Blockset parameter estimation interface. It is organized into several functional areas:

- Select Board:** Shows the selected board as "DRV8305 and F28379D La...".
- Communication Port:** Indicates the connection is via "HOST Serial Setup".
- Required Inputs:** Lists various motor parameters to be estimated:
 - Input DC Voltage: 24 V
 - Nominal Current: 7.1 A (peak value)
 - Nominal Speed: 4000 rpm
 - Pole pairs: 4
 - Nominal Voltage: 24 V
 - Sensor Selection: Sensorless
 - Position Offset: 0.8669 Per Unit Position
 - Total QEP Slits: 1000
- Test Status:** Features a "Run" button and a "Stop" button.
- Estimated Motor Parameters:** Lists parameters such as R_s (ohm), L_d (H), L_q (H), B_{emf} (V/krpm), Motor Inertia (Kg.m²), and Friction constant (N.m.s).
- Signal Conditioning and Scaling:** Shows a "SelectedSignal" output.
- Fault Status:** Includes indicators for "Over Current", "Under Voltage", and "Serial communication".
- Signal from Target:** Shows a "Speed" dropdown and a "SelectedSignal" input.
- Buttons:** "Save Parameters" and "Open Model" (highlighted with a red box).
- Steps:**
 - Provide required inputs.
 - Press **Ctrl+D** to update the workspace.
 - Build, Deploy & Start required [target models](#).
 - Run this model to estimate motor parameters.
- Copyright 2020 The MathWorks, Inc.**

- Create a new model and manually add the PMSM motor block from the Motor Control Blockset library:

Create a new Simulink model and add the Surface Mount PMSM block from the Motor Control Blockset library in the Simulink library browser. Open the block mask and enter the motor parameters manually. You can obtain these parameters by using:

- The Motor Control Blockset parameter estimation workflow. For details, see “Estimate PMSM Parameters Using Recommended Hardware”.
- The motor datasheet or from other known sources.



In Surface Mount PMSM block, set the **Simulation type** parameter to Discrete and the **Sample Time (Ts)** parameter to 25e-6 (half of the control frequency). Discrete simulation improves the simulation speed.

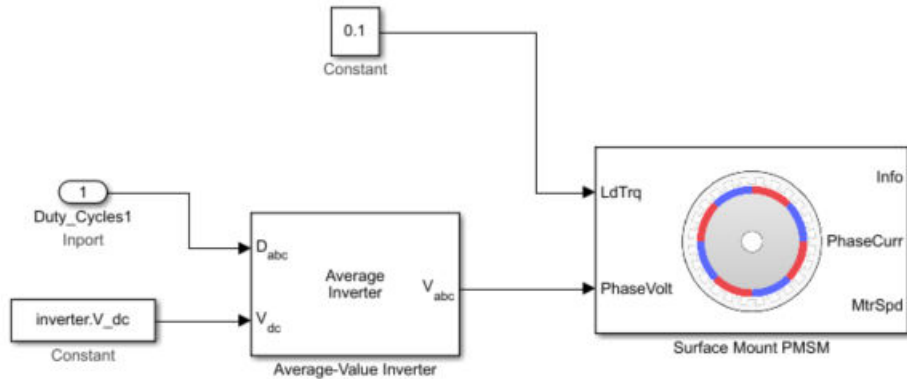
If the parameters are available in a MAT-file, click the **Browse** button on the block parameters dialog to locate the MAT-file and then click **Load from file** to load the parameters.

The files containing the default motor parameters are available in the location `<matlabroot>\toolbox\autobks\autobksshared\mcbtemplates` as a reference.

In the Surface Mount PMSM block parameters dialog, you can also represent the motor parameters as workspace variables and use the model initialization script (m-script) to automatically update these variables using the model initialization callback. Parameters of some commercially available motors are available in the file `mcb_SetPMSMMotorParameter.m` as a reference. For details about this m-script file, see “Estimate Control Gains and Use Utility Functions”.

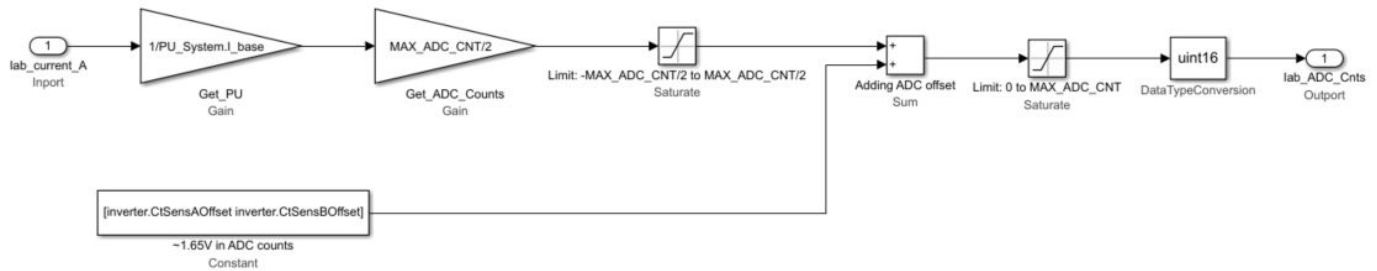
Add Average-Value Inverter Block

In the Simulink model that contains the Surface Mount PMSM block, add an Average-Value Inverter block from the Motor Control Blockset library. The Average-Value Inverter block reads the normalized PWM duty-cycle and DC voltage input (in volts) and outputs the phase voltages. Connect the **V_{abc}** output port of the Average-Value Inverter block to the **PhaseVolt** input port of the Surface Mount PMSM block.



Create Motor Phase Current Sensing and Signal Conditioning Subsystem

In the physical hardware, the motor current read by the current sensors is filtered and scaled to an ADC measurable range. The ADC peripheral in the processor reads the current signals and outputs the ADC counts for the current control algorithm. This figure shows an example of how you can model the motor phase current sensing and signal conditioning algorithms.



The maximum measurable peak current is considered as the base current. The ADC counts can be calculated from the base current and full-scale ADC values, along with the ADC offset, by using this equation:

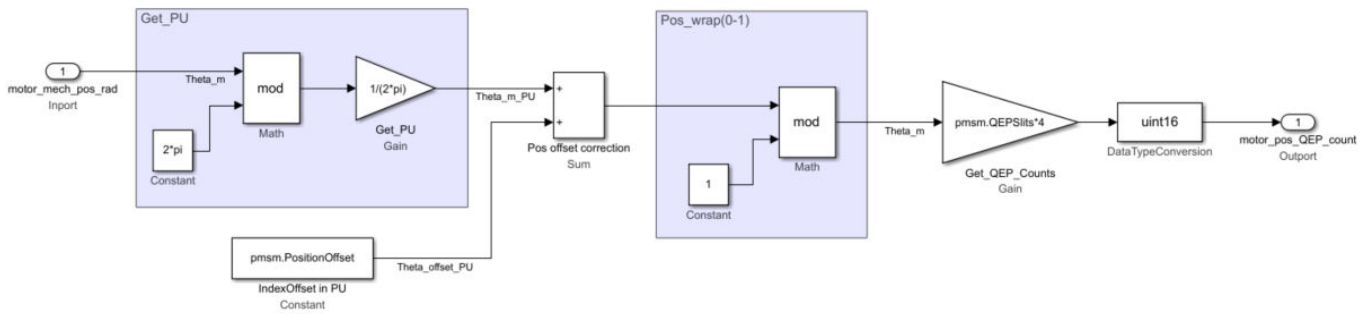
$$\text{ADC counts} = \frac{(\text{Full scale ADC counts}/2)}{\text{Base current (in amperes)}} + \text{ADC offset}$$

For the default inverter and signal conditioning circuit parameters for commercially available inverters, see the `mcb_SetInverterParameters.m` file. To add a new inverter configuration, create an inverter type in this file and use this in the model initialization script for parameter initialization. If you are using low-pass filters for measuring the current, add an average model to filter the current.

Create Position Sensing Subsystem

The position sensing subsystem reads the motor position from the Surface Mount PMSM block and simulates the QEP encoder pulse counts. The Surface Mount PMSM block outputs the mechanical position of the motor in rad/s.

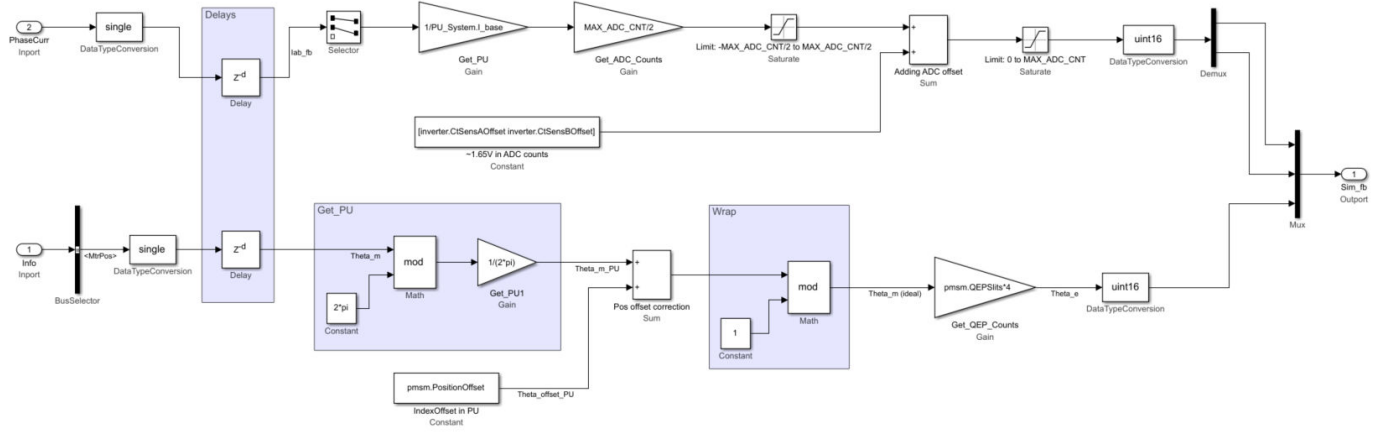
Convert the position in the range 0 to 2π rad/s to QEP encoder counts as shown in this figure.



For details about detecting the QEP index position offset with respect to the rotor d -axis, see “Quadrature Encoder Offset Calibration for PMSM Motor”.

Add Delay in Plant Model

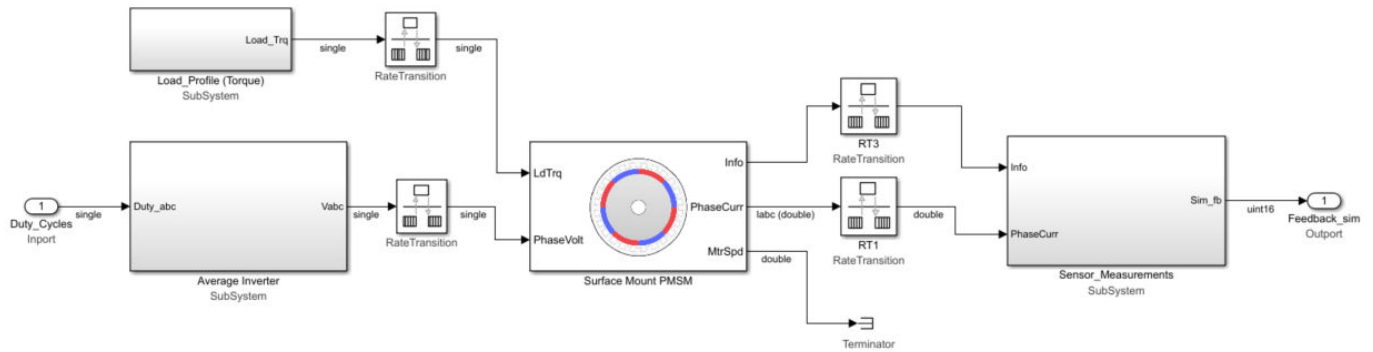
You can add delays in the plant model to simulate the control algorithm processing delays in the hardware and the PWM switching delays. The algorithm processing delay in the processor is the time taken to update the PWM. PWM switching delay is usually half the switching time period.



For adding delays in the discrete time solver with a sample rate of $T_s/2$ (half the switching time period), the processor computation delay and PWM switching delay are factored as Z^{-1} ($T_s/2$).

Integrate Blocks and Subsystems

The final step of designing the plant model using Motor Control Blockset is to integrate the blocks and subsystems that you created earlier. The completed plant model accepts the normalized PWM from the controller and outputs the motor phase currents and position.



Hardware Troubleshooting

- “Check ADC Inputs” on page 4-2
- “Verify PWM Outputs” on page 4-4
- “Check Hardware Connections” on page 4-6
- “Test Algorithm Design” on page 4-7
- “Check Generated Code” on page 4-8

Check ADC Inputs

Description

The analog to digital converter (ADC) can measure incorrect values. For example, in custom-designed analog circuits, currents measured by ADC can be incorrect due to noise, out-of-phase measurements, or sampling issues. This results in faulty feedback to the control system that leads to instability.

Action

Verify ADC Pin

See the hardware schematics and verify that you identified and configured the correct ADC pins for a given measurement (a-phase, b-phase).

Verify ADC Block Configuration

Open the ADC block and verify that the **Input Channels, ADC module, SOC trigger, SOCx acquisition window** parameters are configured correctly.

ADC sampling begins with the SOC event. In some cases, for example, when sensing the current through the shunt resistors, ADC sampling requires synchronization with the bottom leg switches. In this case, verify that the SOC event is configured correctly with ADC-PWM interrupt synchronization. This also results in reduced EMI/EMC noise in the sampling because ADC conversion happens outside the PWM transition. For more information, see “Task Scheduling in Target Hardware” on page 2-6.

Reduce Noise in ADC Sampling

You may notice noise in the ADC samples. This may happen either if there is EMI/EMC or if sampling is faster than what the device can support. EMI/EMC can be reduced by improving the hardware design.

To avoid problems due to faster sampling, see the device datasheet and determine the maximum supported clock frequency of the ADC. For example, if you are using a Texas Instruments TMS320F28379D series microcontroller, it can support a CPU clock frequency of 200 MHz, but the maximum clock frequency supported by the ADC module is 50 MHz. Use this value to set **ADC clock prescaler (ADCCLK)** parameter on the **Hardware Implementation** tab in the **Configuration Parameter** dialog box of your model.

Check VDD of Current Measurement Device

Many current measurement devices derive VDD from the DC power supply (V_{DC}). In addition, the device enable pin also determines the supply voltage to the internal current measurement circuit (for example, Texas Instruments BOOSTXL-DRV8305). Absence of VDD (or the device enable pin) results in 0 V at the ADC of the target hardware. Ensure that these conditions are not present in your hardware.

Check ADC Current Conventions

Check if you are using the correct conventions for ADC current sensing. Motor Control Blockset considers the current entering the motor (or leaving the inverter) as positive. This convention changes with the hardware because of the differences in the inverting or non-inverting op-amp and

the analog current sensing circuit. Check the inverter current sensing circuit op-amp and set the `inverter.invertingAmp` variable (control parameter) to:

- 1 — If the current sensing circuit detects the current entering motor as positive.
- -1 — If the current sensing circuit detects the current entering motor as negative.

For more information about setting a control parameter, see “Estimate Control Gains and Use Utility Functions”.

Test Readability of Unipolar and Bipolar Signals

Check if the measurement circuit is designed to read unipolar and bipolar signals.

Check if the `inverter.ISenseVoltPerAmp` variable (control parameter) is set correctly according to the hardware specification. For more information about this parameter, see “Estimate Control Gains and Use Utility Functions”.

DC signal measurement circuits are usually unipolar. For example, BoostXL-DRV8305 has a DC voltage measurement circuit that converts the voltage range of 0 - 44.3 V to 0 - 3.3 V at the ADC. Voltage ADCs cannot measure negative voltages.

AC signal measurement circuits are usually bipolar. For example, BoostXL-DRV8305 has an AC current measurement circuit that converts the current range of -23.57 to +23.57 A to 0 - 3.3 V at the ADC with an offset of 1.65V.

Check ADC Offset and Gain computation

Verify the ADC offset values before deploying and executing the code on the target hardware. For more information, see “Current Sensor ADC Offset and Position Sensor Calibration”.

Check the accuracy of the computed gain for conversion of the ADC counts to signal value in the real world as described in the previous section.

Check ADC Resolution

Check the ADC resolution to determine the minimum value of the signal that it can measure. For example, a 3.3 V 12-Bit ADC that can measure ± 16.5 A has a resolution of 0.1 Volts/Ampere. The minimum current that the ADC can measure (excluding EMI/EMC and noise) is approximately 8 mA.

Determine the minimum measurable current by the ADC. Verify that this current is greater than the ADC signal-to-noise ratio, tolerance, and errors. Ensure that you simulate and check the model before deploying it to the target hardware.

Low ADC resolution can result in difficulties when implementing sensorless algorithms to control motors that consume very small currents (for example, 50 mA AC) on no load. In addition, EMI/EMC and noise affects ADC measurements. It is a good practice to simulate the model and verify if the ADC resolution is appropriate. Increase the gain of the sensor amplifier on the hardware to increase the ADC resolution.

Verify PWM Outputs

Description

The motor control algorithm generates the pulse width modulation (PWM) signals to control the motor through inverter. In some cases, the PWM signals can be incorrect due to improper switching frequency, wrong interrupt and PWM generation configurations, or error in the duty cycles. Incorrect PWM signals result in improper switching of the inverter.

Action

Verify PWM Frequency

Use an oscilloscope to verify that the generated PWM signals has the expected switching frequency. In embedded targets, configuration of the PWM module depends on factors such as target hardware and clock frequency. For example, you can use these equations to calculate PWM_Counter_Period for Texas Instruments C2000 targets that have the ePWM module configured to work with the Up-Down counting mode:

$$\text{CPU_frequency (Hz)} = 200\text{e}6$$

$$\text{PWM_frequency (Hz)} = 20\text{e}3$$

$$\text{PWM_Counter_Period (PWM timer counts)} = \text{CPU_frequency} / \text{PWM_frequency} / 2$$

Verify PWM Generation

Ensure that you feed a correct PWM duty cycle to the switching device (for example, MOSFET or IGBT). PWM generation depends on these active-high and active-low configurations:

- Active high — 25% duty results in 25% on-time for upper leg MOSFET or IGBT (recommended).
- Active low — 25% duty results in 75% on-time for upper leg MOSFET or IGBT.

In addition, check if there is any inversion of the PWM signal between the target and MOSFET due to the gate driver or isolator circuit (25% gate pulse must be 25% on-time by the driver chip).

Verify Interrupt Configuration

Majority of the controller algorithms are designed to work with the ADC-PWM synchronization for advantages like current sensing, reduced EMI/EMC interference.

ADC sampling begins with the SOC event. In some cases, for example, when sensing the current through the shunt resistors, ADC sampling requires synchronization with the bottom leg switches. In this case, verify that the SOC event is configured correctly with the ADC-PWM interrupt synchronization. This also results in reduced EMI/EMC noise in the sampling because ADC conversion happens outside the PWM transition. For more information, see “Task Scheduling in Target Hardware” on page 2-6.

Verify Updates to PWM Duty

Verify if the PWM duty is updated or refreshed in synchronization with the PWM module. To implement a robust control, it is a good practice to timely refresh the PWM duty (for example, once in T_{pwm} , preferably before $T_{\text{pwm}}/2$).

Check Behavior at PWM Generation Limits

Check the datasheet of the PWM driver circuit for support at the 0% duty and 100% duty limits. For functional safety, it is a good practice to limit the maximum duty cycle somewhere between 95 and 98% by setting the corresponding value in the DQ Limiter block.

Check for Incorrect PWM Generation Configuration

Verify that the hardware uses the correct PWM generation configuration. For example, BoostXL-DRV8305 supports 3-PWM mode, 6-PWM mode, and 1-PWM mode.

Check for Default Dead Bands

Check if there are dead bands introduced by the motor driver board. Consider this while generating dead bands from the PWM module.

Confirm Maximum Switching Frequency

Determine the maximum possible switching frequency for the inverter and driver from the device datasheets. Ensure that the model does not exceed this value.

Check Hardware Connections

Description

When you try to run the motor, you may face problems due to incorrect hardware connections. This may result in rise in temperature of the motor, inverter, hardware board or an abnormal behavior such as uncontrolled motor speed.

Action

Verify Hardware Connections

Check the wiring and connections before getting started. For details, see “Hardware Connections”. For instructions to determine the serial port connected to the hardware, see “Find Communication Port”.

Manually Check Rotation of Shaft

Verify that the shaft of your motor is rotating freely with minimal rotational friction. A mechanical failure in the bearings may result in thermal overloads, which can damage the motor windings.

Verify Rated Currents for Motor and Inverter

Determine the rated currents of the motor and inverter from the manufacturer datasheet. Ensure that you do not overload the motor for durations longer than what the original equipment manufacturer (OEM) has specified.

Check Motor and Inverter Temperature

Ensure that the temperature of the motor windings and inverter heat sink are within the expected temperature range. Overloading the hardware results in excessive heat that can damage the hardware.

Verify Measurements from Analog Circuits

Verify the range of the signals that you measure from the analog circuits (for example, the maximum current of the inverter).

Check for Additional Resistors

After you complete the process of estimating the motor parameters, you should not change the motor connections because this leads to differences in the contact and cable resistances. In addition, verify that the initialization script of the model takes into consideration any additional resistors present in the power circuit.

Verify Fault Pin and Enable Pin Connections

Check and verify that the fault pins and enable pins are connected correctly on the target hardware board.

Test Algorithm Design

Description

When simulating or running a model on the target hardware, you can face problems because of defects in the implementation of the control algorithm. This can lead to an uncontrolled motor speed, differences in the current waveforms or mismatch in PI controller gains between simulation and target hardware.

Action

Verify Parameters and Other Input Data

Verify that you identified and entered the inputs (for example, motor and inverter parameters, clock speed, and switching frequency) correctly. If the input data is incorrect, the motor control algorithm will not work. Use the Motor Control Blockset parameter estimation tool to compute the motor parameters. For more details, see “Estimate PMSM Parameters Using Recommended Hardware”.

Verify Waveforms of Measured Currents

After you load the motor shaft, verify that the waveforms for the measured signals match the shape visible in the simulations. For example, field-oriented control ensures perfect sinusoidal waveforms for currents. For exceptions, see “Check ADC Inputs” on page 4-2.

Verify Control System Design

Verify that all the controllers used in the model (for example, PI controllers and sliding mode observer) are designed correctly.

You can start by simulating the model by using the estimated motor parameters before deploying the model to the target hardware. Observe and verify the step responses for the current and speed by using both simulation and deployment on the target hardware.

Model-Based Design ensures that correct simulation of the model results in identical outcomes on the target hardware with identical gains (that match the gain values computed during simulation) for all the controllers.

Verify Signal Representation

Check if you can represent the signals correctly for a selected data type. For example, it is not possible to store the value 1024 in the 8-bit data-type. Similarly, it may not be possible to represent some gain values in the selected fixed-point resolution.

Verify Base Values for PU Representation

If you are working with the Per-Unit system, please check that the base value of a quantity (for example, base current), is selected correctly. For more details, see “Per-Unit System”.

Check Generated Code

Description

When simulating or running a model on the target hardware, you may face problems due to errors in the software architecture of a model. These errors can affect the performance of control algorithm and increase the code execution time on the hardware.

Action

Check Sample Times

Verify the base rates and other execution rates of the model by using **Debug > Information Overlays > Sample Time > Colors**. The different sample times of the model decide the execution of different tasks in the simulation and in the generated code.

Check for Overruns

Verify that there are no overruns beyond the available sample time. Algorithms with overruns affect the control system stability. If required, optimize the model for code execution. For more details, see “Code Verification and Profiling Using PIL Testing”.

Verify Low-Priority Interrupt Service Routines (ISR)

Verify that the low-priority interrupt service routines (ISR) (for example, speed control loop and communication service routines) are executed according to the design and are not ignored by any overruns in the high-priority ISRs.

Check Execution Order Priority

Check that the model uses a correct execution order priority. Verify that all the interrupts are configured correctly.

Verify Software Initialization

To allow the analog circuits to get ready, check that the software initialization delay (for example, ADC blanking time, PWM driver, and charge pump) is greater than the required value specified by the manufacturer (for example, 2 μ s).

Check Hardware Initialization

Verify that you initialized the target hardware and inverter correctly. Generally, the driver is disabled, which brings all the switches to a high impedance state and initializes the important variables to the default values.

Verify Third-Party Tool Version

Verify that you are using the recommended versions of the third-party tools. Check that bugs in the third-party software do not cause regressions.