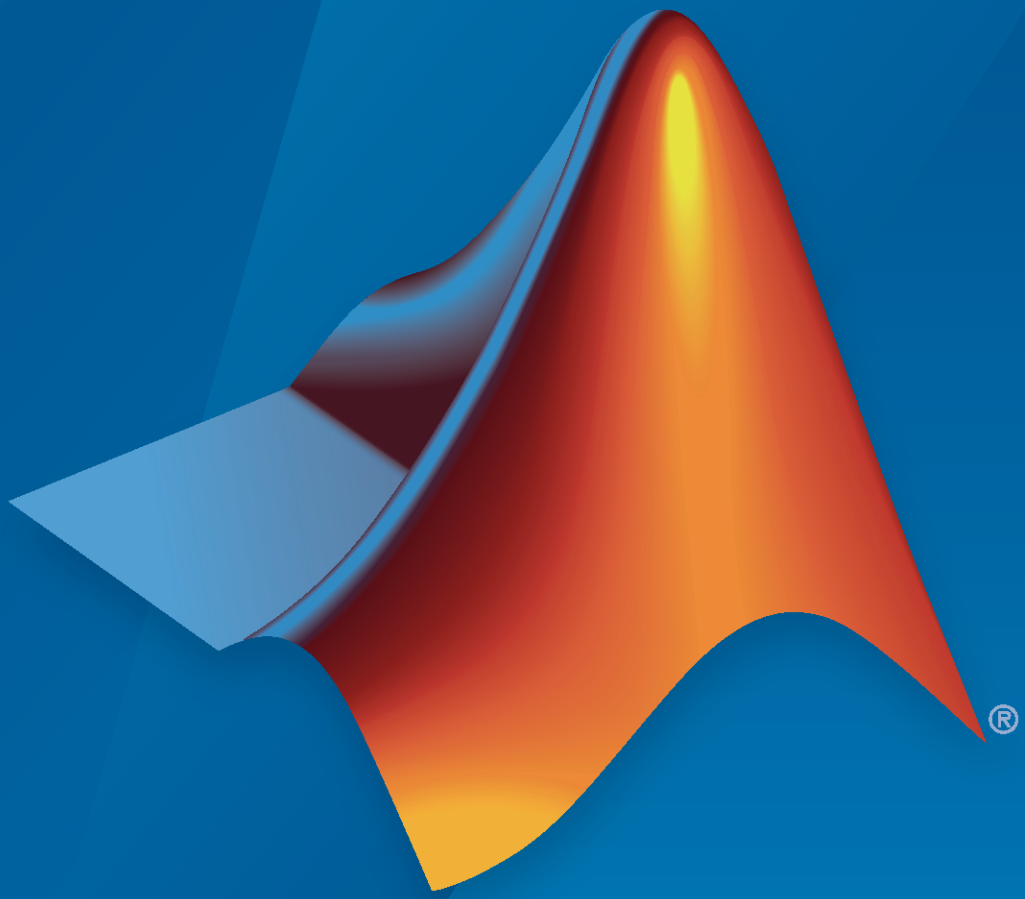
Motor Control Blockset™

User's Guide



MATLAB® & SIMULINK®

R2020a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

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

Phone: 508-647-7000

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

Motor Control Blockset™ User's Guide

Revision History

March 2020	Online only	New for Version 1.0 (Release 2020a)

Contents

1 Design the Controller

2 Deploy and Validate System

3 Plant Modeling

Design the Controller

Design Field-Oriented Control Algorithm

The implementation of speed control algorithm for a motor involves these tasks:

- Current scaling - Read the current from ADC counts and convert to PU
- QEP position decoding - Read the QEP position counts and calculate the rotor electrical position
- Torque control - Current control in d-q axis
- Speed control

The below steps can help you to implement speed control algorithm for a PMSM using Motor Control Blockset. These guidelines to build speed control algorithm are also related to the example model `mcb_pmsm_foc_qep_f28379d` and explains the steps to tune the control parameters for d-axis and q-axis current controllers and the speed controller.

1. “Design Current and Position Scaling Subsystem” on page 1-3
2. “Design Current Controller Subsystem” on page 1-6
3. “Perform Manual Gain-tuning of Current Controller” on page 1-9
4. “Design Speed Control Algorithm” on page 1-12
5. “Perform Manual Gain-Tuning of Speed Controller” on page 1-15

Note In these workflow steps, variables are used for defining datatypes, execution time of current controller, and execution time of speed controller. Refer to the initialization script of the example model `mcb_pmsm_foc_qep_f28379d` for the details of the variables defined in these steps.

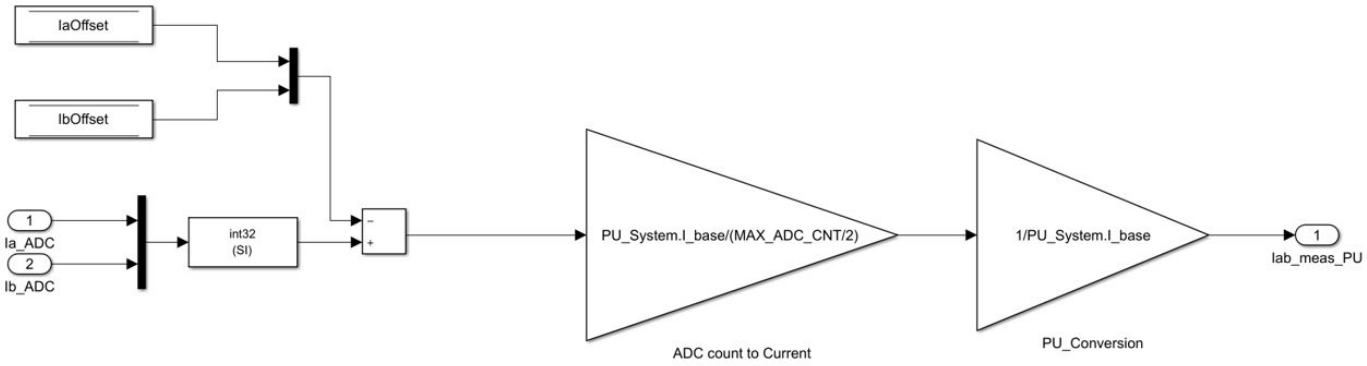
Tip A basic understanding of Simulink is a prerequisite for following this workflow as these workflow steps do not provide details about tasks like defining datatype in a constant block or usage of math operations blocks in Simulink.

Refer to the section “Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool” for estimating the motor parameters, and then refer to “Creating Plant Model Using Motor Control Blockset” on page 3-2 to design a plant model, which helps to verify the control algorithm in simulation.

Design Current and Position Scaling Subsystem

To design current and position scaling subsystem:

1. Read the current in ADC counts and convert to Per-Unit (PU).



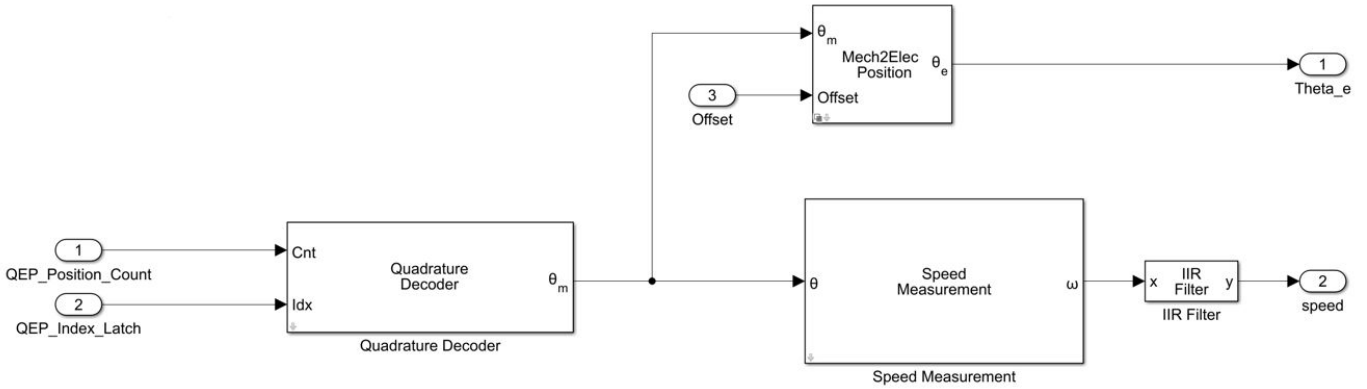
IaOffset and *IbOffset* are the ADC offsets for current measurement and they are hardware specific. The default ADC offset (*CtSensAOffset* and *CtSensBOffset*) is given in the file `mcb_SetInverterParameters.m` for few commercially available inverters. For details about ADC offset calibration in hardware,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

In this subsystem shown in the above figure, the motor phase current measured in ADC counts is converted to current in Per-Unit (PU). `PU_System.I_base` refers to the base current. For details, see “Per-Unit System”. Refer to `mcb_SetPUSystem.m` that computes the Per-unit values for the system.

Use base values for computing real world values from per-unit. For real world value or SI unit implementation, refer to the example model `mcb_pmsm_foc_qep_f28379d_SIUnit`.

The data-store memory *IaOffset* and *IbOffset* are used for sharing the data between the subsystems.

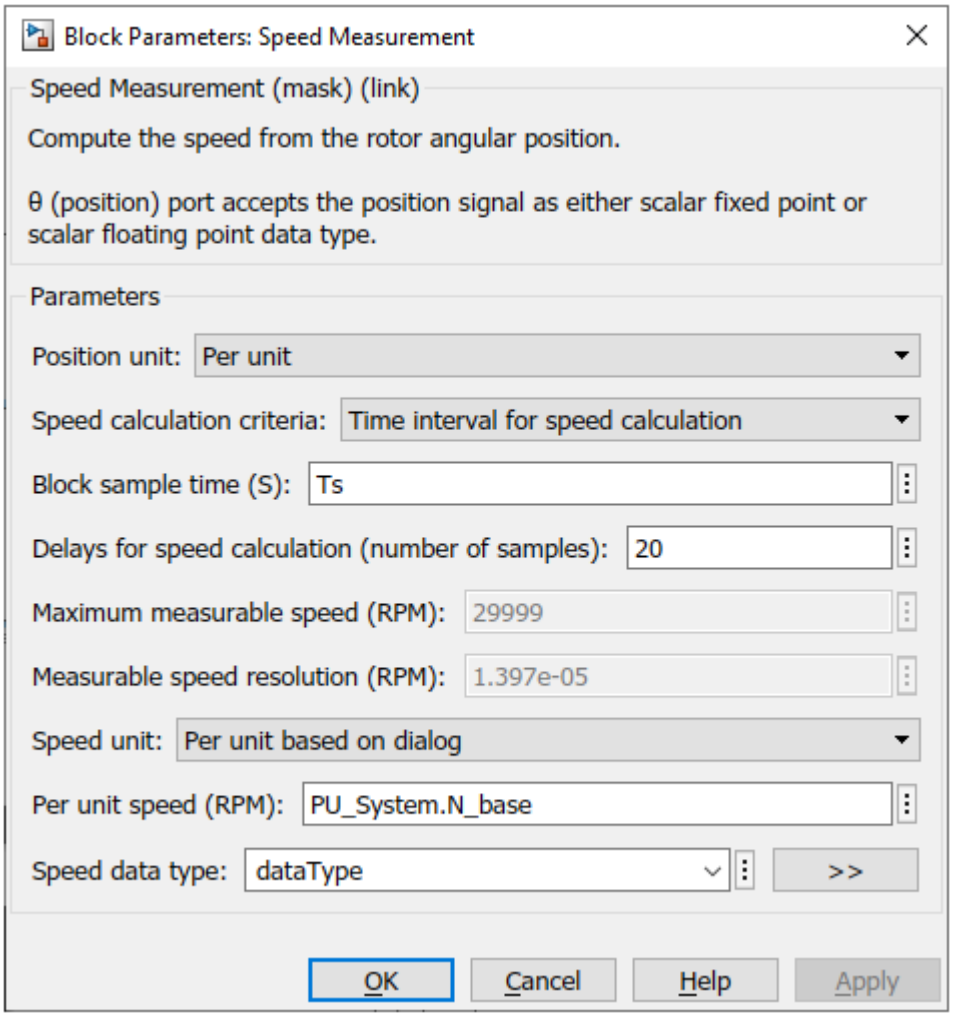
2. Read the position from the QEP pulse count.



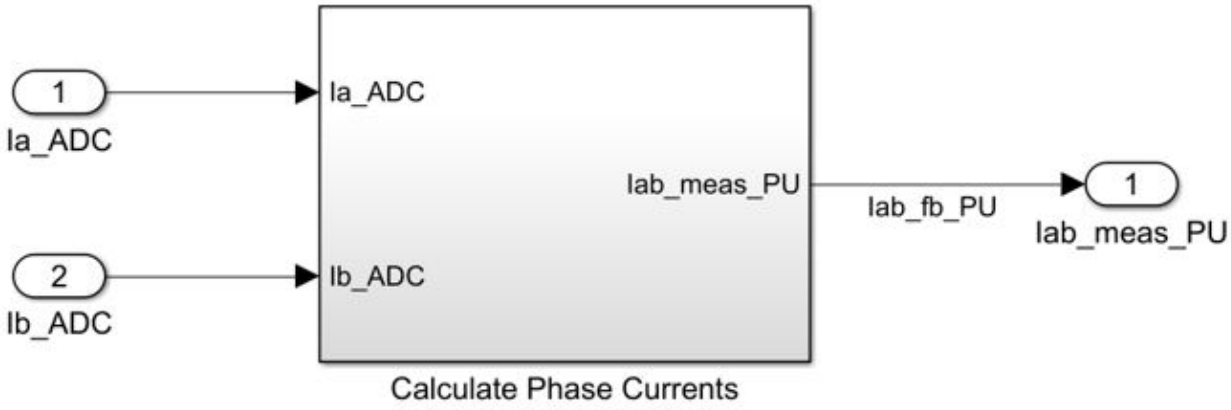
The Quadrature Decoder block reads the position count from plant model or hardware driver block. The block converts the rotor mechanical position in encoder position counts to rotor mechanical angle in Per-Unit (0-1).

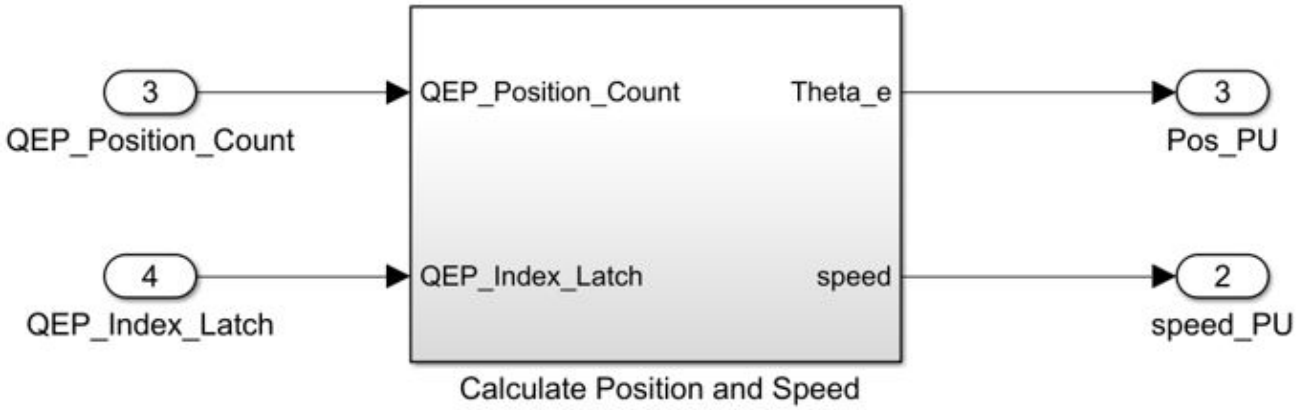
The Mech2Elec Position block adjusted the mechanical angle for QEP offset and converts them to electrical angle. This rotor electrical angle is required for the FOC algorithm to spin the motor. Refer to “Quadrature Encoder Offset Calibration for PMSM Motor” for calculating the QEP encoder offset.

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



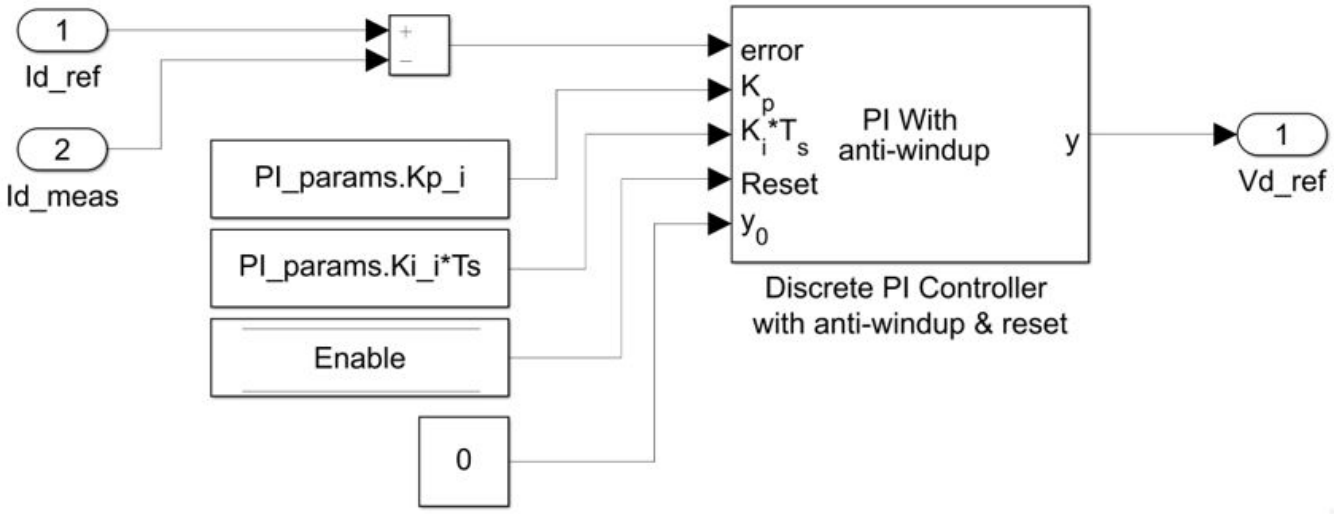
3 Create subsystems to include the current scaling and position decoding logic, as explained in the above steps.


1
Ia_ADC
Ia_ADC
2
Ib_ADC
Ib_ADC
Iab_meas_PU
Iab_fb_PU
1
Iab_meas_PU
Calculate Phase Currents


3
QEP_Position_Count
QEP_Position_Count
4
QEP_Index_Latch
QEP_Index_Latch
Theta_e
speed
3
Pos_PU
2
speed_PU
Calculate Position and Speed

Design Current Controller Subsystem

1 From the Motor Control Blocksetlibrary in Simulink Library browser, use the Discrete PI controller with anti-windup & reset block (under /Controls/Controllers library), for designing d-axis and q-axis current control.

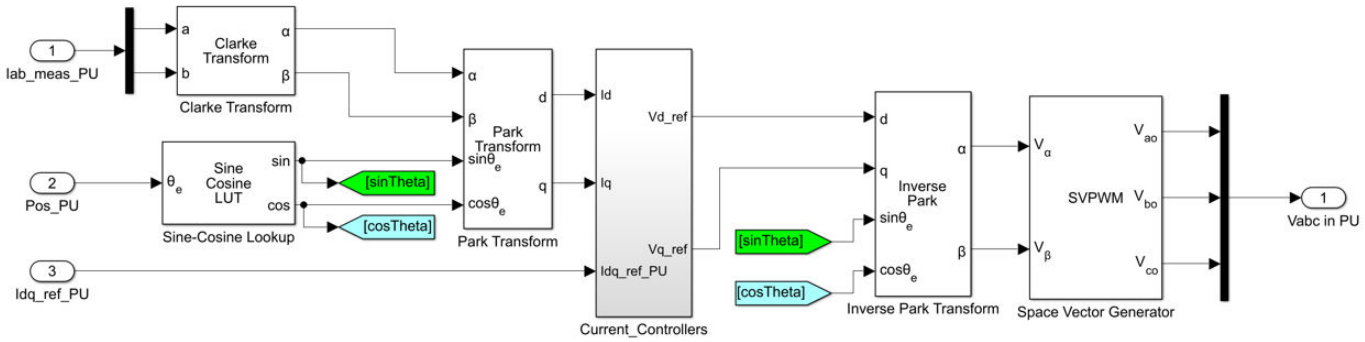


The MATLAB function `mcb.internal.SetControllerParameters` calculates the PI control gains for d and q axis current controller and speed controller. For details on control parameter gain estimation, refer to “Estimate Control Gains from Motor Parameters”. Refer to the file `mcb_pmsm_foc_qep_f28379d_data.m` for Ts (50 µs).

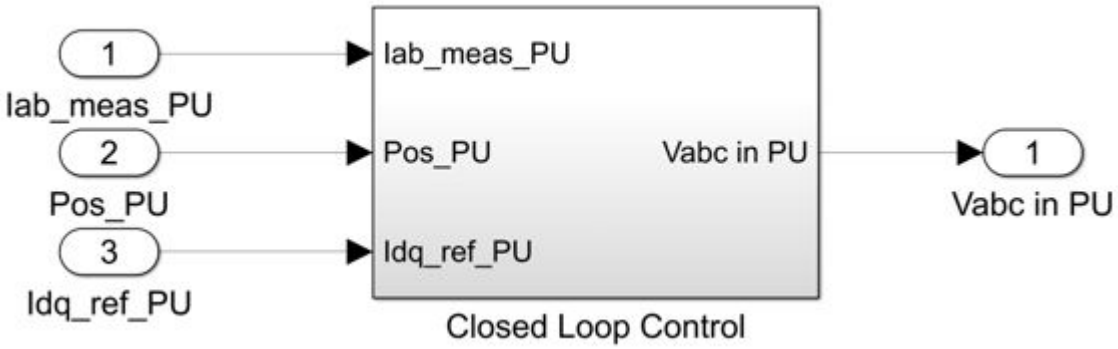
The `Enable` variable is Data-store memory to reset the controller and this is optional.

Create a subsystem (`Current_Controllers`) for d-axis and q-axis PI controllers for controlling the d-axis and q-axis current.

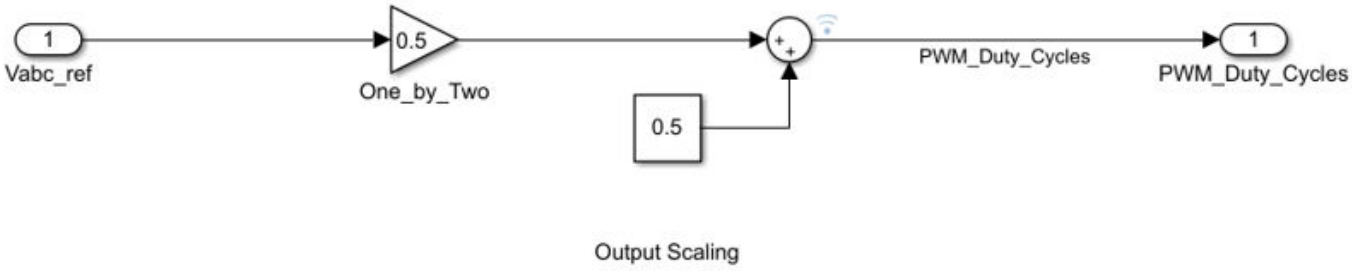
2 Add the blocks Clarke Transform, Park Transform, Inverse Park Transform, and Space Vector Generator from Motor Control Blockset/Controls/Math Transforms to the `Current_controllers` subsystem (from previous step) as shown in this figure:



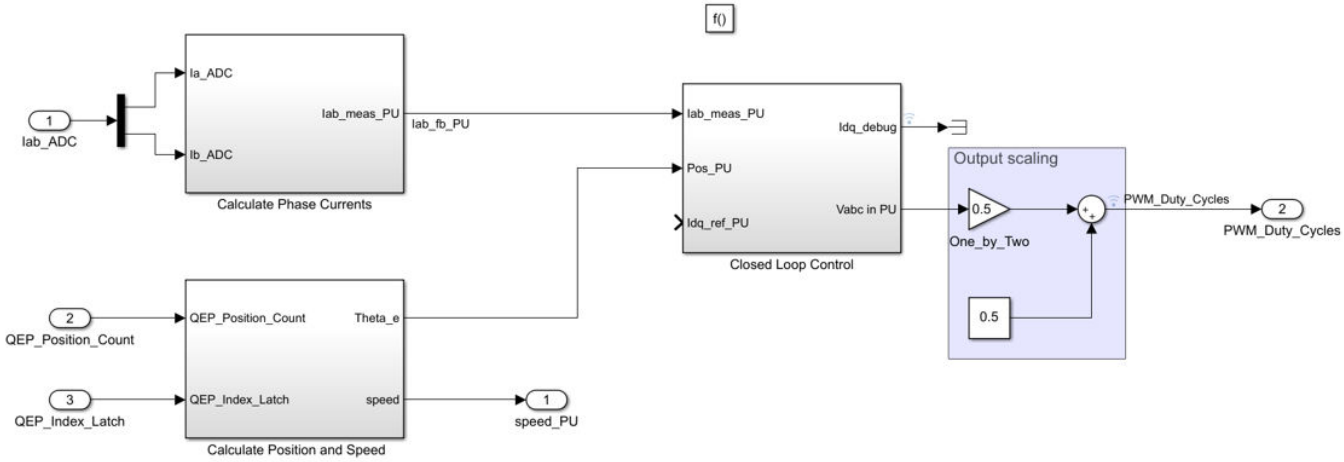
3 Create a subsystem named `Closed Loop Control` as shown in this figure:



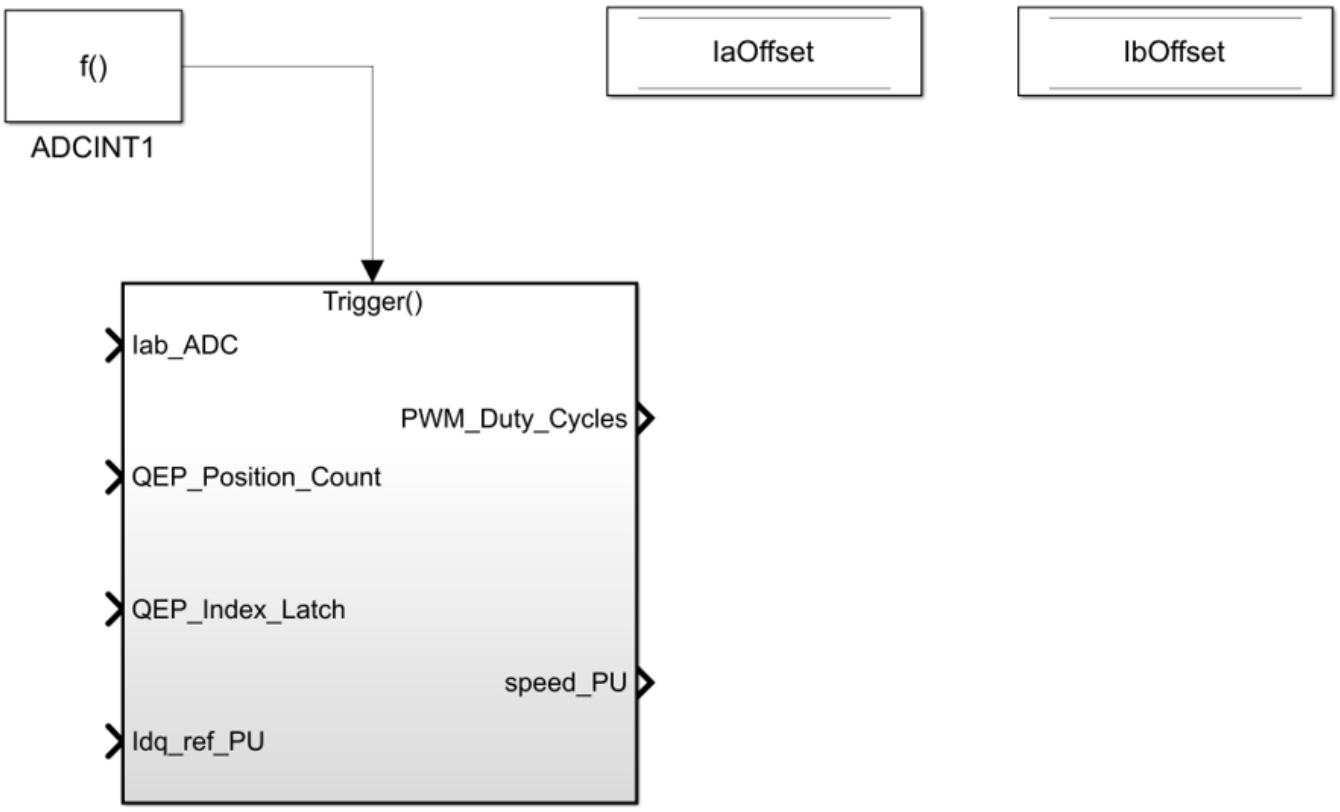
4 Create subsystem for scaling the PWM outputs. This subsystem outputs normalized PWM duty (0-1) for the plant model.



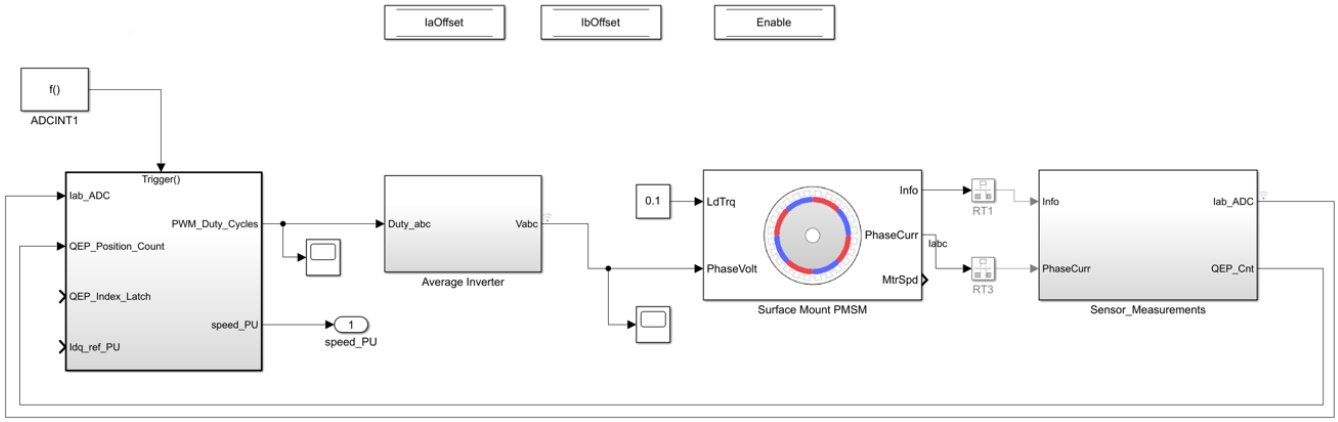
5 Integrate the Current scaling, QEP position decoding, Closed Loop Control, and Output Scaling logic. Add trigger from Simulink\Ports & Subsystems to the subsystem and select the Trigger type as function-call.



6 Integrate the current controller to form a subsystem. Add Function-Call Generator from Simulink/Ports & Subsystems. In Function-Call Generator dialog, enter **Sample time** as the control-loop sample time `Ts` (default 50e-6 s).



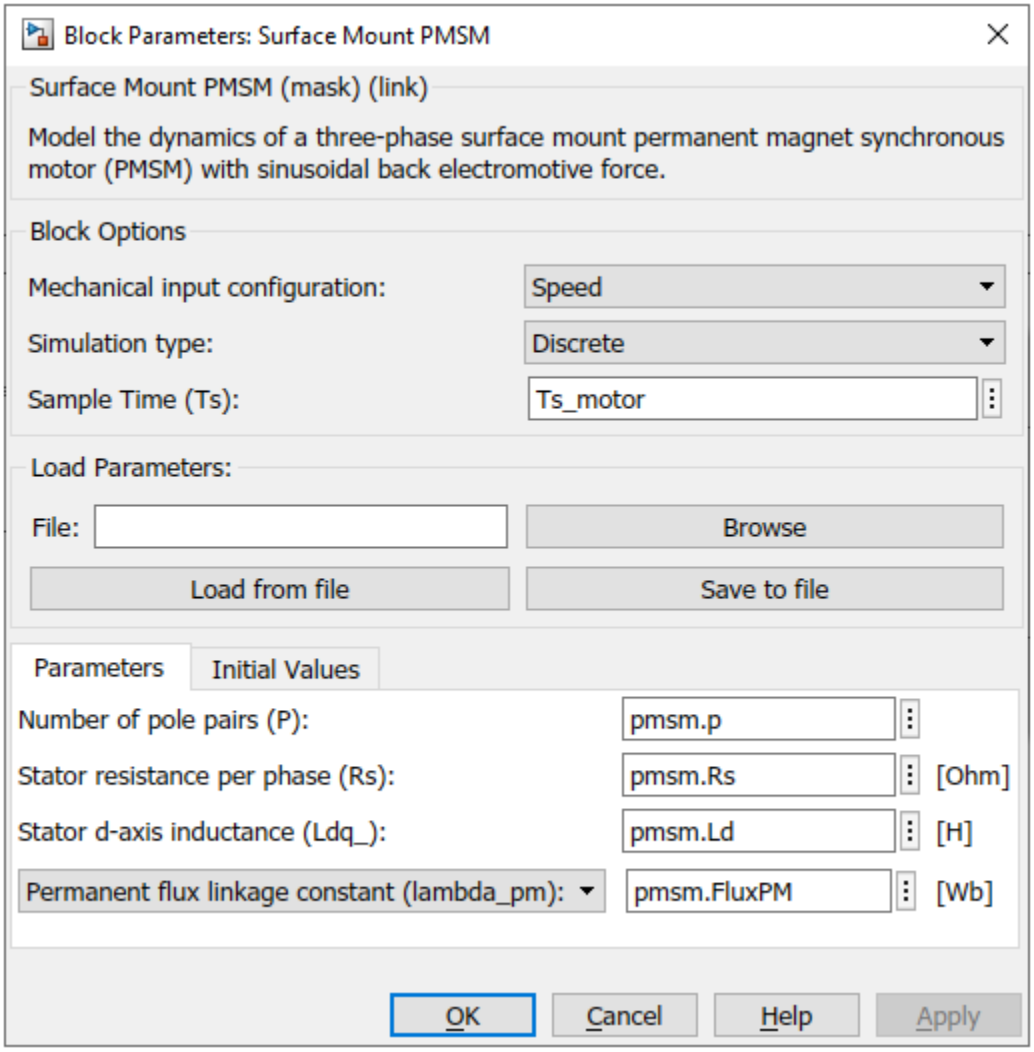
7 Integrate the plant model and controller. For detailed steps on how to create a plant model for motor control system, refer to “Creating Plant Model Using Motor Control Blockset” on page 3-2.



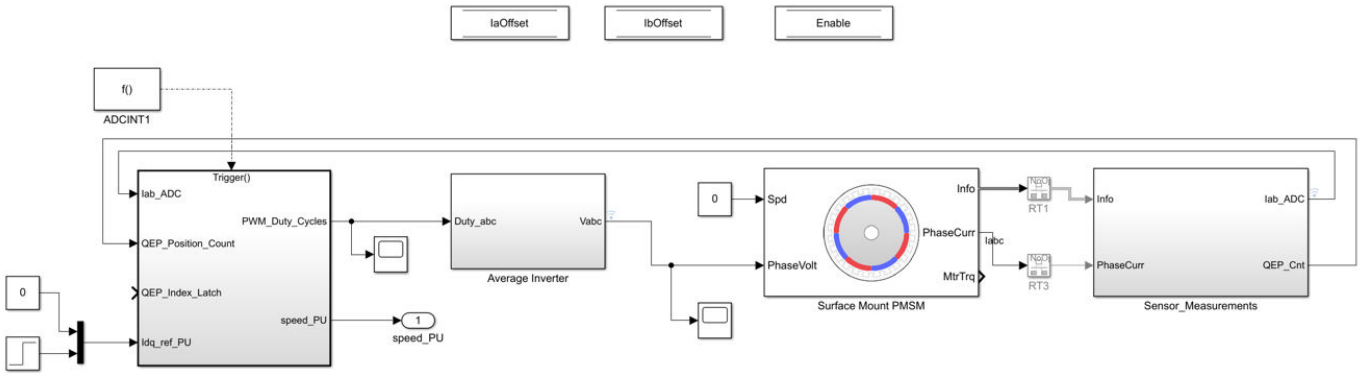
Perform Manual Gain-tuning of Current Controller

In this step, you perform gain tuning for d-axis and q-axis current controller manually.

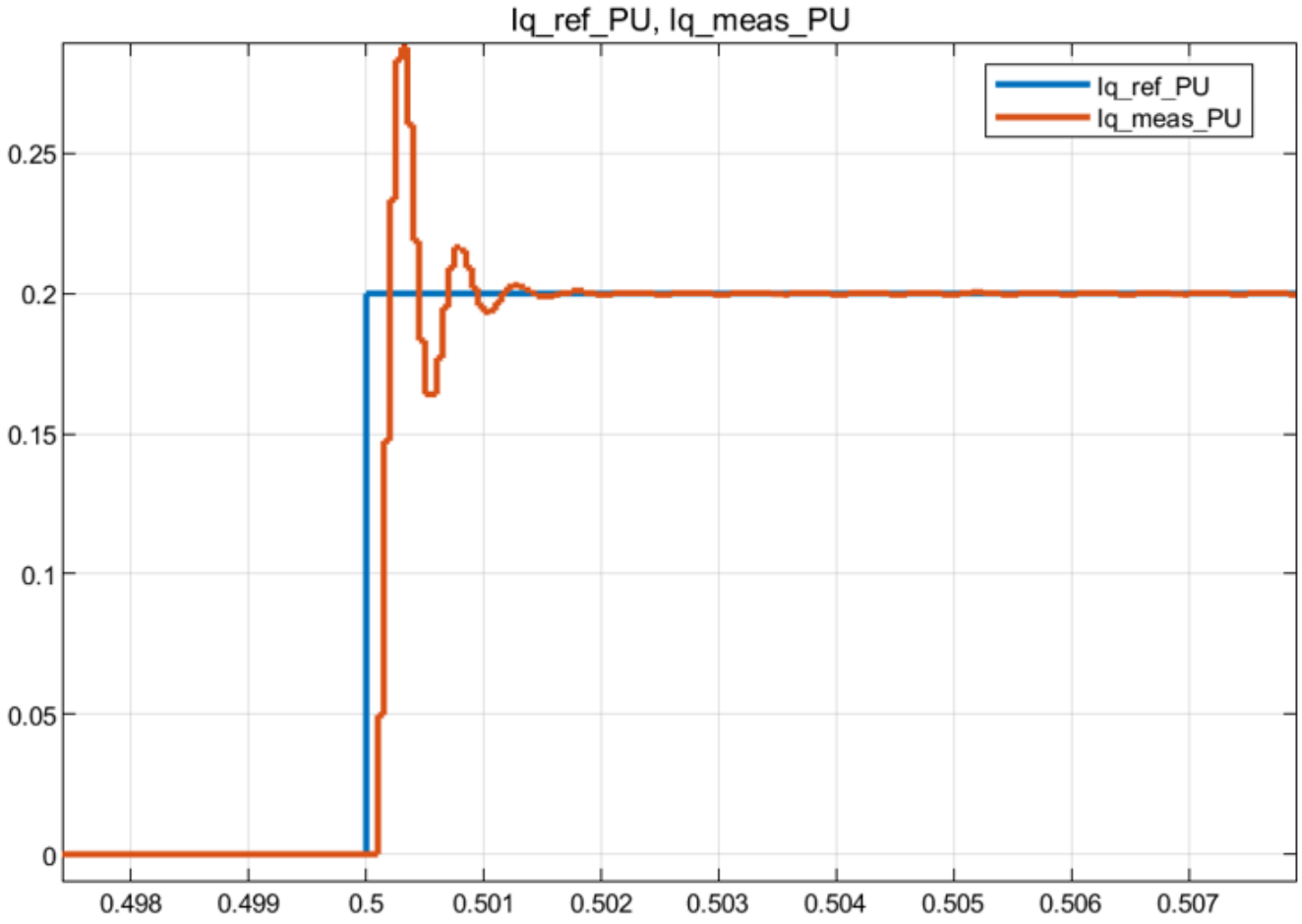
This step is optional and helps you to tune the control gain parameters for current controller. Provide step change for `Id_ref` and analyze the current controller performance from step response of `Id_meas`. Repeat the same with `Iq_ref` for tuning the q-axis current controller. In the plant model, lock the rotor to ensure motor is not spinning when step change is given for `Id_ref` or `Iq_ref`. In the Surface Mount PMSM block mask, select `Speed` for **Mechanical input configuration**. Provide speed input as 0 and this ensures that the rotor is locked.

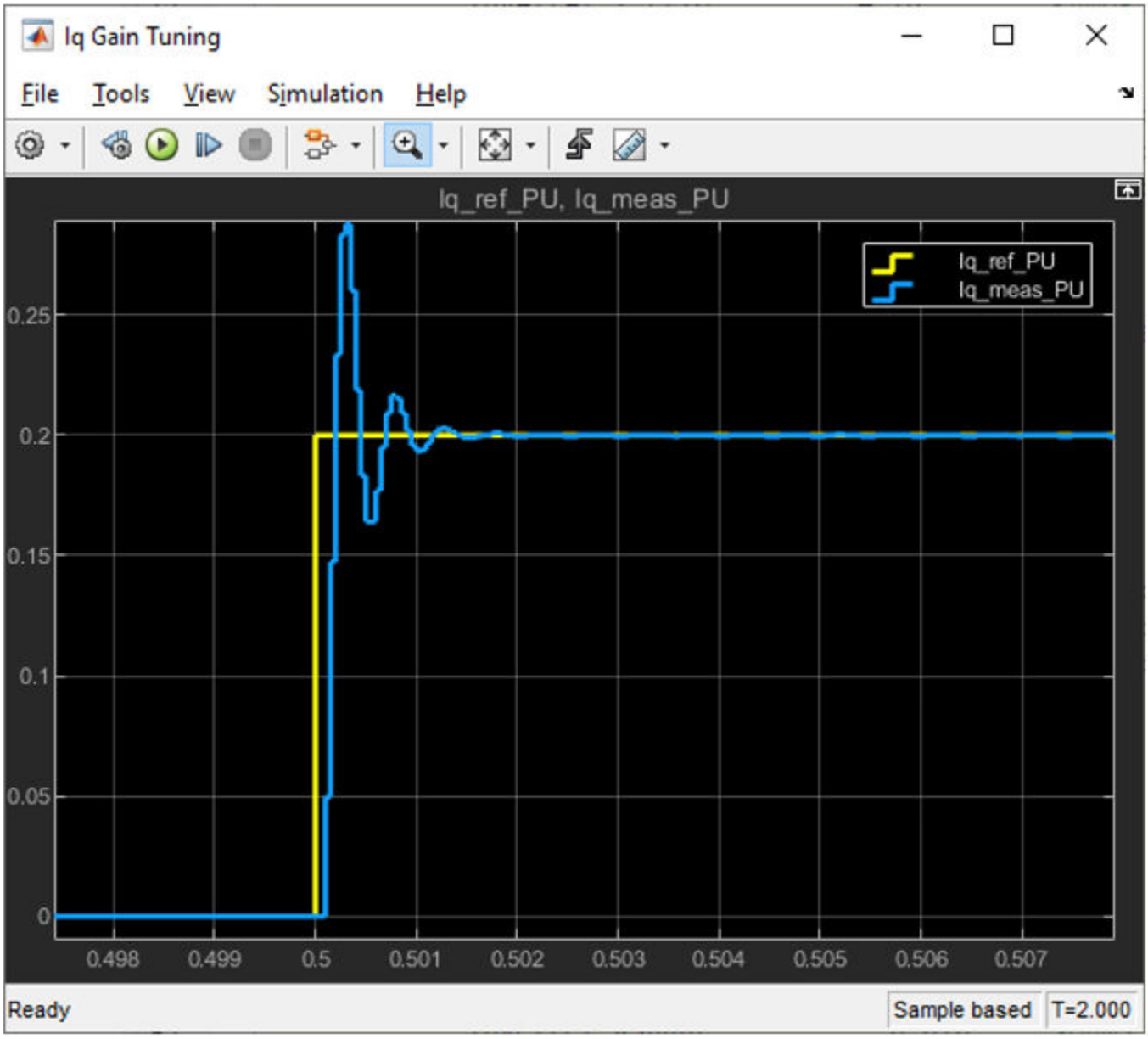


This simulation model allows manual gain tuning for current controller. Give a step input to `Iq_ref` in the range (0 to 0.2) PU and observe the measured feedback. Adjust control parameters of current controller to meet the control objectives.



Run simulation and plot the `Iq_ref_PU` and `Iq_meas_PU` and analyze the step response. This simulation model allows to tune the control parameters for q-axis controller to meet the control objectives.





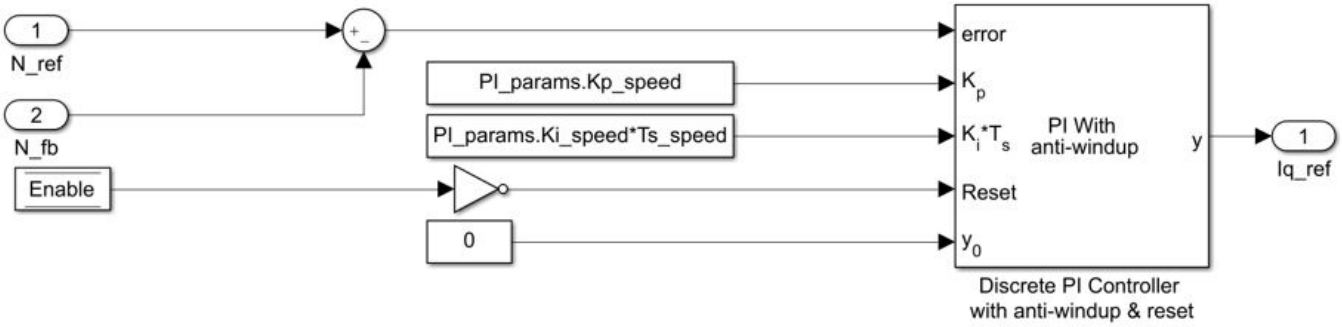
Adjust the control parameters to meet the control objective. You can follow the same method for tuning the d-axis current controller.

Design Speed Control Algorithm

To design a speed control algorithm:

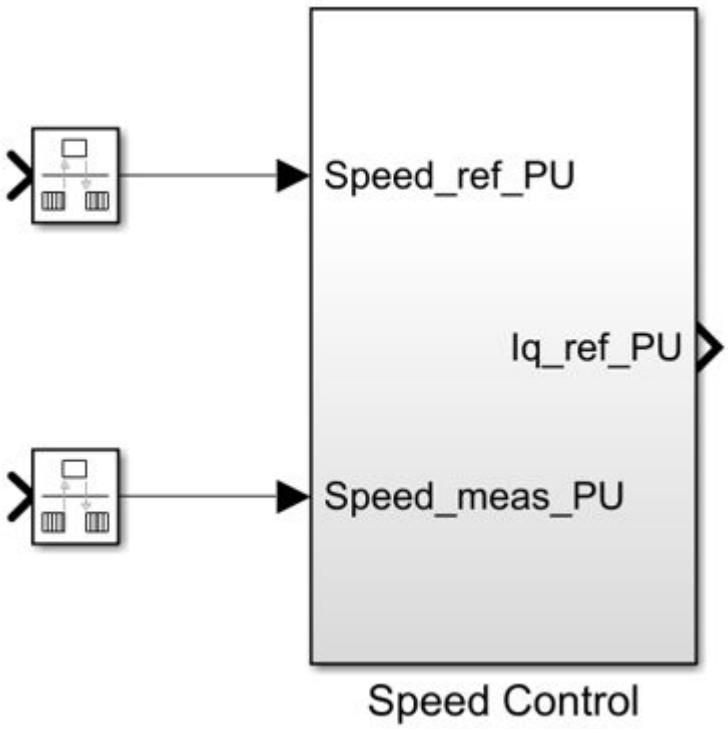
1. Add a speed controller for PMSM motor. Speed control loop outputs Iq_ref current and this is the input for the current controller.

   In the Simulink library browser, select the Discrete PI controller with anti-windup & reset block from Motor Control Blockset/Controls/Controllers library.

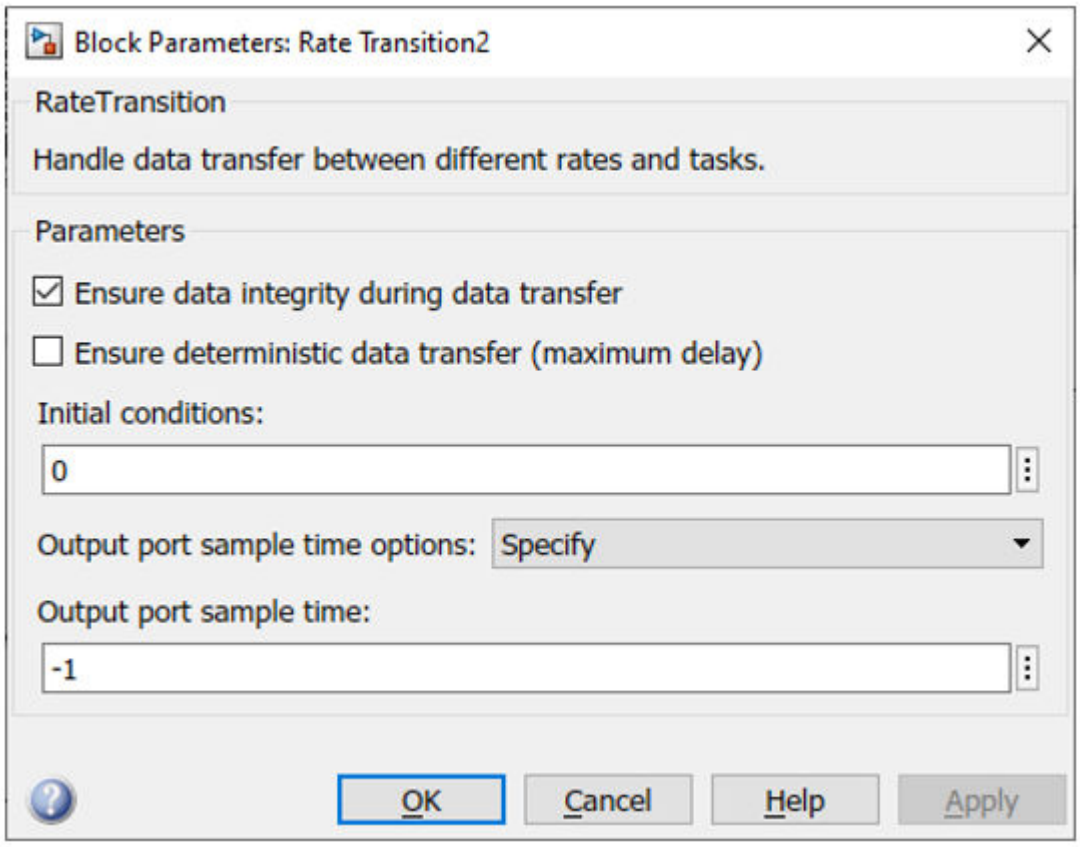


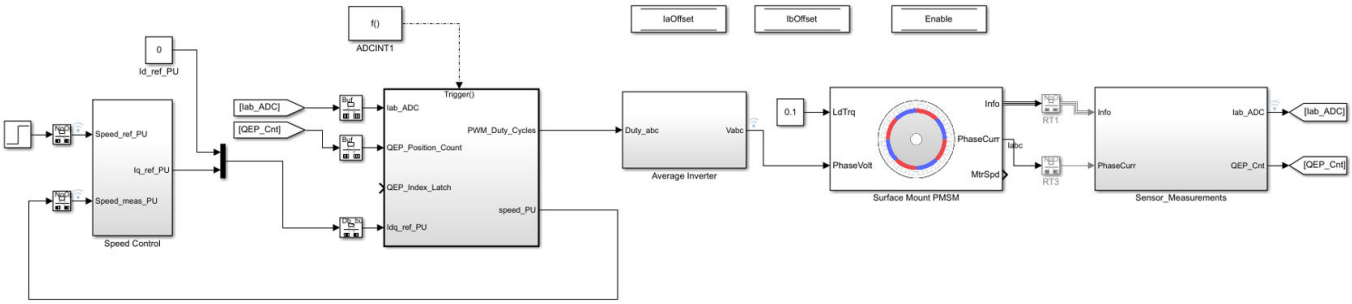
   The MATLAB function `mcb.internal.SetControllerParameters` calculates the PI control gains for d and q axis current controller and speed controller. For more details on control parameter gain estimation, refer to the section “Estimate Control Gains from Motor Parameters”. Refer to `mcb_pmsm_foc_qep_f28379d_data.m` for Ts_speed (500 µs). Enable Data-store memory to reset the controller and this is optional.

2. Create a subsystem for speed controller and add a rate-transition block to the inputs with sample time as *Ts_speed* (execution time of speed control loop).



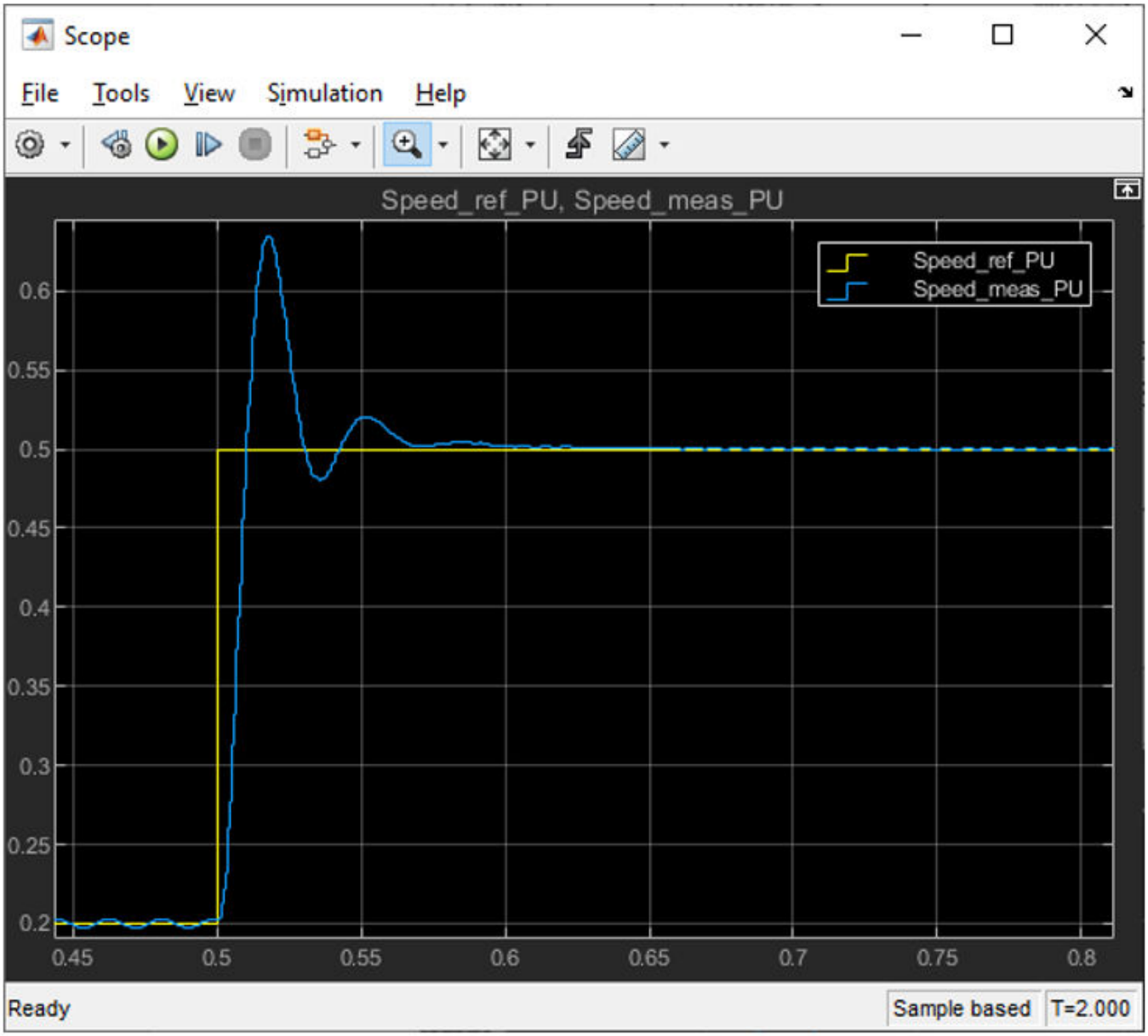
3 Integrate the above speed controller that you designed with the current controller and plant integrated model. Connect *Iq_ref_PU* output from Speed controller to the current controller input through a rate-transition block as both are executing in different sample rates. The below figure shows the settings for the Rate-Transition block connected between the speed controller and current controller.


Block Parameters: Rate Transition2
RateTransition
Handle data transfer between different rates and tasks.
Parameters
Ensure data integrity during data transfer
Ensure deterministic data transfer (maximum delay)
Initial conditions:
0
Output port sample time options:
Specify
Output port sample time:
-1
OK
Cancel
Help
Apply


f()
ADCINT1
0
Id_ref_PU
Speed_ref_PU
Iq_ref_PU
Speed_meas_PU
Speed Control
[Iab_ADC]
[QEP_Cnt]
Trigger()
Iab_ADC
QEP_Position_Count
QEP_Index_Latch
Idq_ref_PU
PWM_Duty_Cycles
speed_PU
IaOffset
IbOffset
Enable
Duty_abc
Vabc
Average Inverter
0.1
LdTrq
PhaseVolt
Info
PhaseCurr
MtrSpd
Surface Mount PMSM
RT1
RT3
Iabc
Info
PhaseCurr
Iab_ADC
QEP_Cnt
Sensor_Measurements
[Iab_ADC]
[QEP_Cnt]

Perform Manual Gain-Tuning of Speed Controller

To manually tune the speed controller, add a step input (0.2 to 0.5 PU) to the speed reference input in Speed Controller subsystem. Monitor the step response of the speed and tune the speed controller control parameters. The below figure shows the step response of the speed controller.



The preceding steps gives an approach for speed controller implementation for a PMSM motor in simulation. Run the simulation and analyze the controller performance.

You can generate the C code from this control algorithm using Embedded Coder®. You can deploy this code and the hardware drivers to the target hardware.

Code Verification and Profiling Using Processor-In-the-Loop Testing

In Processor-In-the-Loop (PIL) simulation, the control algorithm executes in the target hardware but the plant model runs on the host machine. The plant model (running on the host machine) simulates the input and output signals for the controller (running on the target hardware) and communicates with the controller by using serial communication interface. Therefore, you can use PIL simulation to determine the execution time on the target hardware, which you can compare with the execution time for simulating the model on the host machine.

The execution time or the performance metrics of an algorithm that you get from PIL simulation, helps you to detect algorithm overrun in the target hardware. The PIL profiling report indicates the average and maximum execution time of an algorithm on the target hardware. As an example, we explain PIL profiling on the LAUNCHXL-F28379D hardware board.

We use the example "mcb_pmsm_foc_sim.slx" to demonstrate code verification in PIL. This example shows PIL profiling for the "Current Control" subsystem available in the model. This subsystem includes Field-Oriented Control (FOC), current scaling (per-unit conversion), speed measurement, and rotor position scaling (computation of angle from the encoder position counts) algorithms. The PIL profiling report shows the average execution and maximum execution times of the control algorithm in the target hardware.

This section addresses these tasks:

- Verify code execution by using PIL testing by comparing the algorithm in the simulation and target hardware operating modes.
- Perform PIL profiling by measuring the algorithm execution time in the target hardware and generate the PIL profiling report.

Required MathWorks Products

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

Supported Hardware

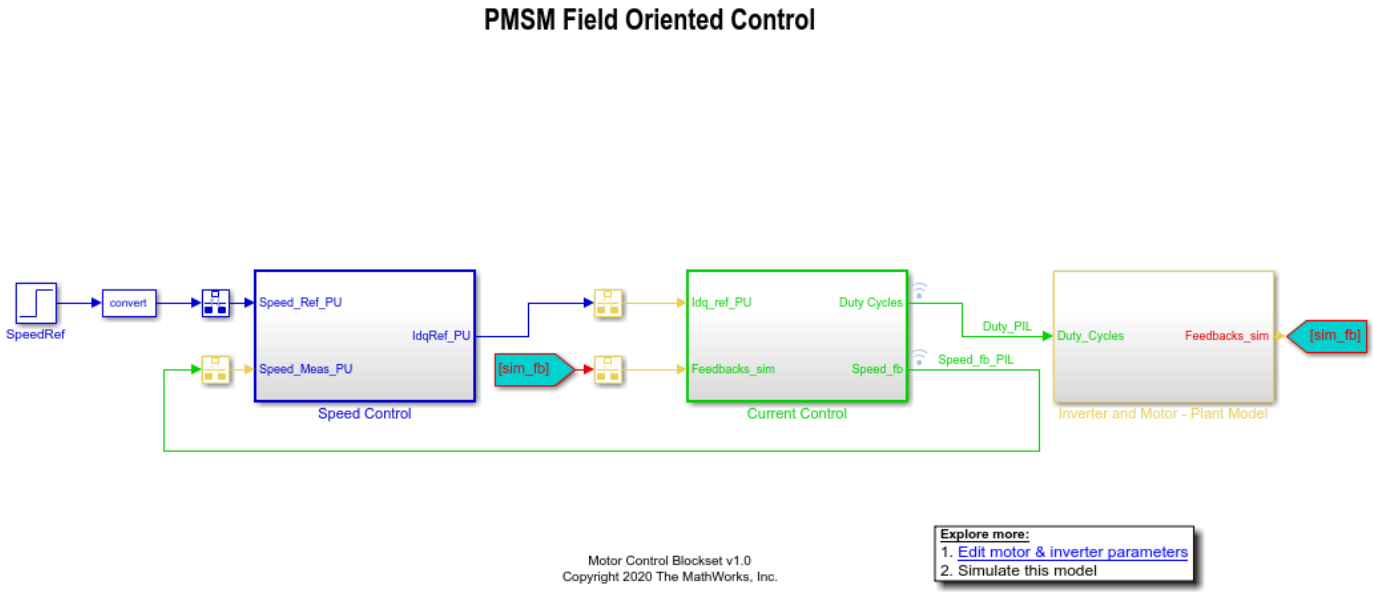
- LAUNCHXL-F28379D controller hardware board
- BOOSTXL-DRV8305 and BOOSTXL-3PHGANINV (supported inverters)
- Teknic motor M-2310P, BLY171D, and BLY172S (supported motors with Hall sensors) or Teknic motor M-2310P and BLY171D (motors that support quadrature encoder)
- DC power supply (24V)

Prepare PIL model

Use these steps to prepare the PIL model for profiling:

1. Open the model "mcb_pmsm_foc_sim.slx" by using this command:

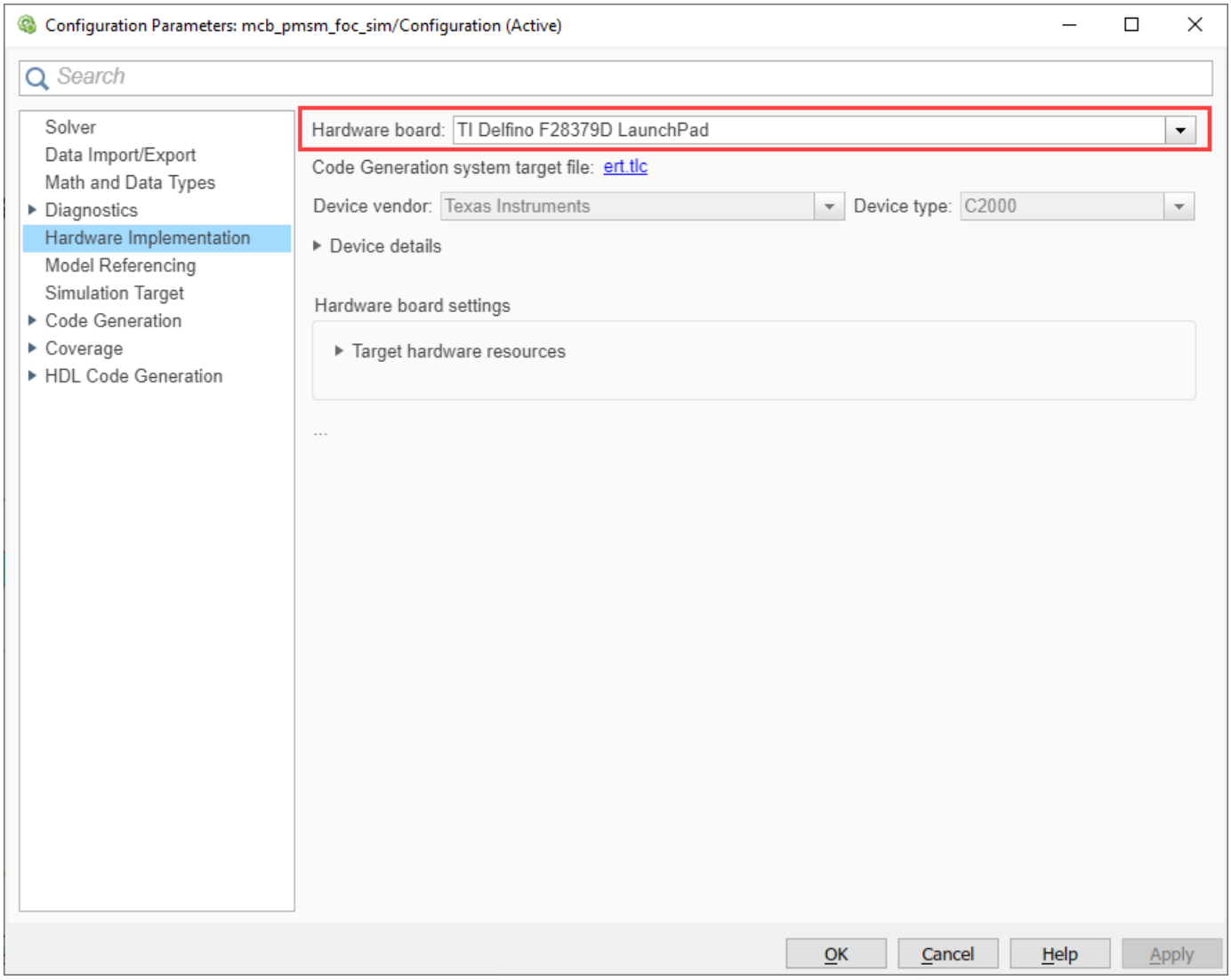
```
open_system('mcb_pmsm_foc_sim.slx');
```



This model simulates the PMSM motor and FOC algorithm for closed-loop speed control.

2 Click **Hardware Settings** in the **Hardware** tab of the Simulink tool strip.

3 Select `TI Delfino F2837xD` in the **Hardware board** field available in the **Hardware Implementation** tab of the **Configuration Paramters** window.



Verify Code by Using PIL

Use these steps to verify the code in PIL:

1. Open the script file "mcb_PIL_config_TI.m" to set the configuration parameters:

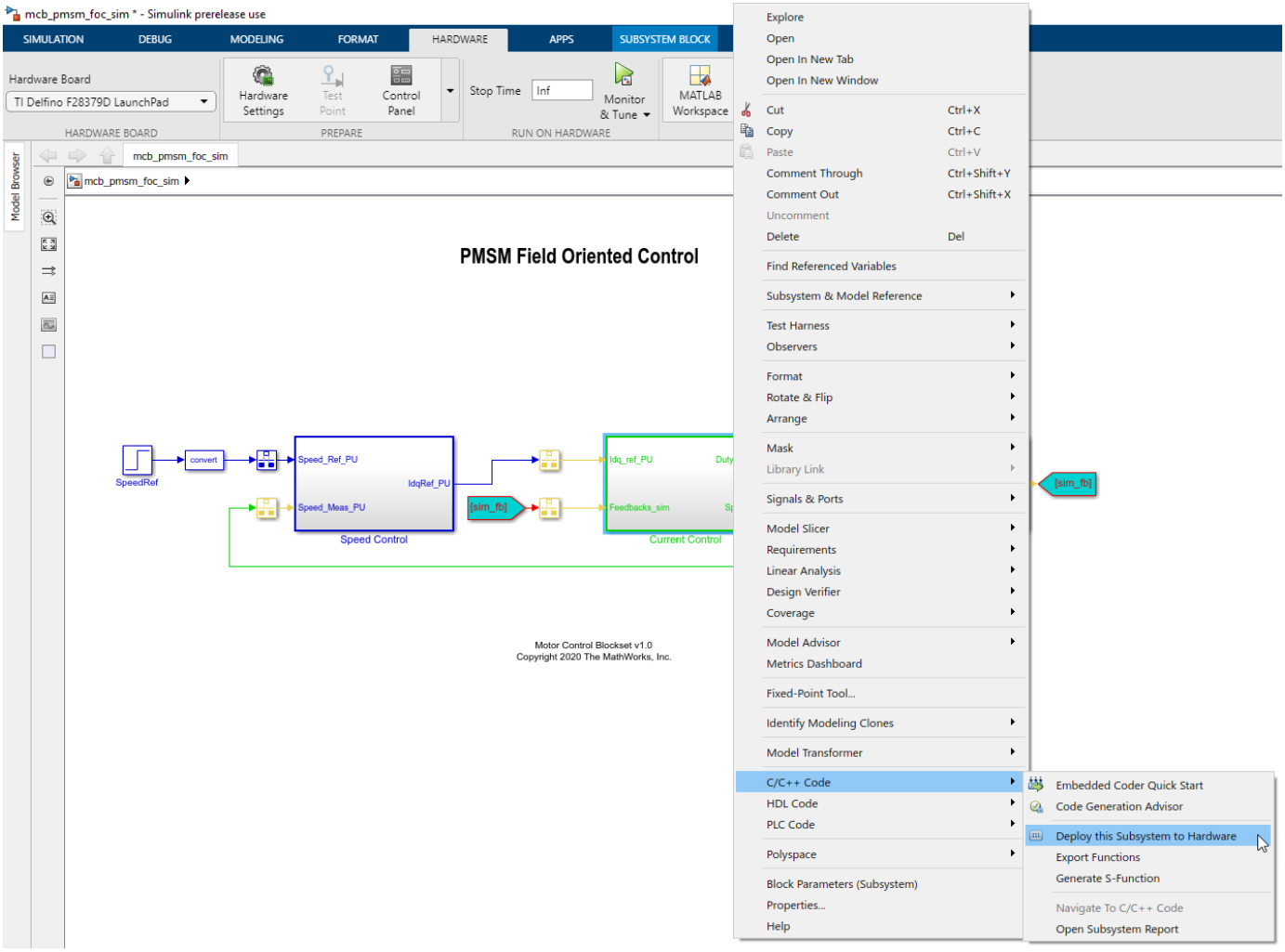
   ```
   edit('mcb_PIL_config_TI.m');
   ```

2. Update the communication port number that you are using.

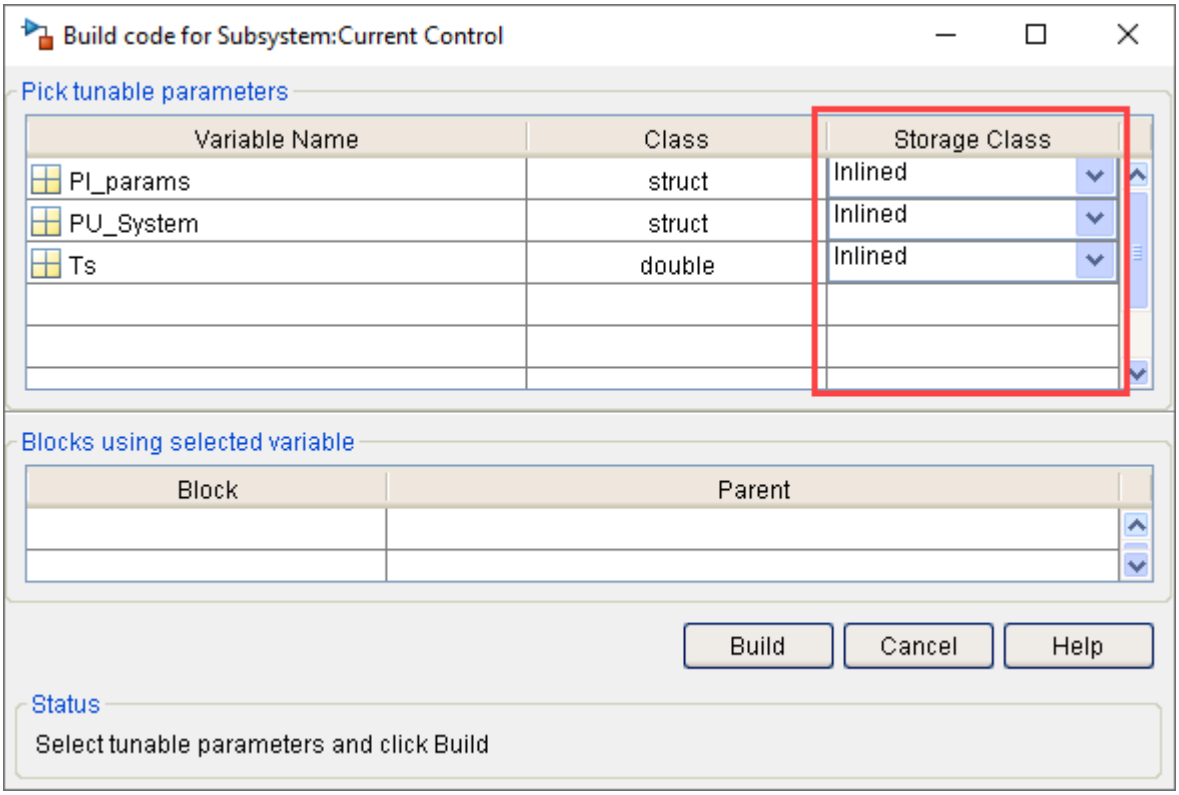
mcb_PIL_config_TI.m

```
% mcb_PIL_config_TI initializes the congiuration parameter for PIL profiling
%
% Ensure the below settings for PIL profiling,
% model - name of the model identified for PIL profing
% StopTime - time required for profiling. Ensure algorithm reaches steady
% state within this specified time
% COMPort - Update the comm port number which is connected to the hardware
% This code is tested for TI c2000 Lauchpad XL (TMS320F28379d)
%
% Copyright 2020 The MathWorks, Inc.

model = 'mcb_pmsm_foc_sim';
set_param(model,'StopTime','0.5');
set_param(model,'SimulationMode','normal');
set_param(model,'ReturnWorkspaceOutputs','on');
set_param(model,'CodeExecutionProfiling','on');
set_param(model,'CodeProfilingInstrumentation','coarse');
set_param(model,'CodeProfilingSaveOptions','SummaryOnly');
set_param(model,'CreateSILPILBlock','PIL');
set_param(model,'DefaultParameterBehavior','Inlined');

disp('Set PIL preferences and setting for model');
setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','COMPort','COM18');
setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','BaudRate',115200);
setpref('MathWorks_Embedded_IDE_Link_PIL_Preferences','enableserial',true);


% subsysToBuild = [model,'/Current Control'];
% rtwbuild(subsysToBuild);
```

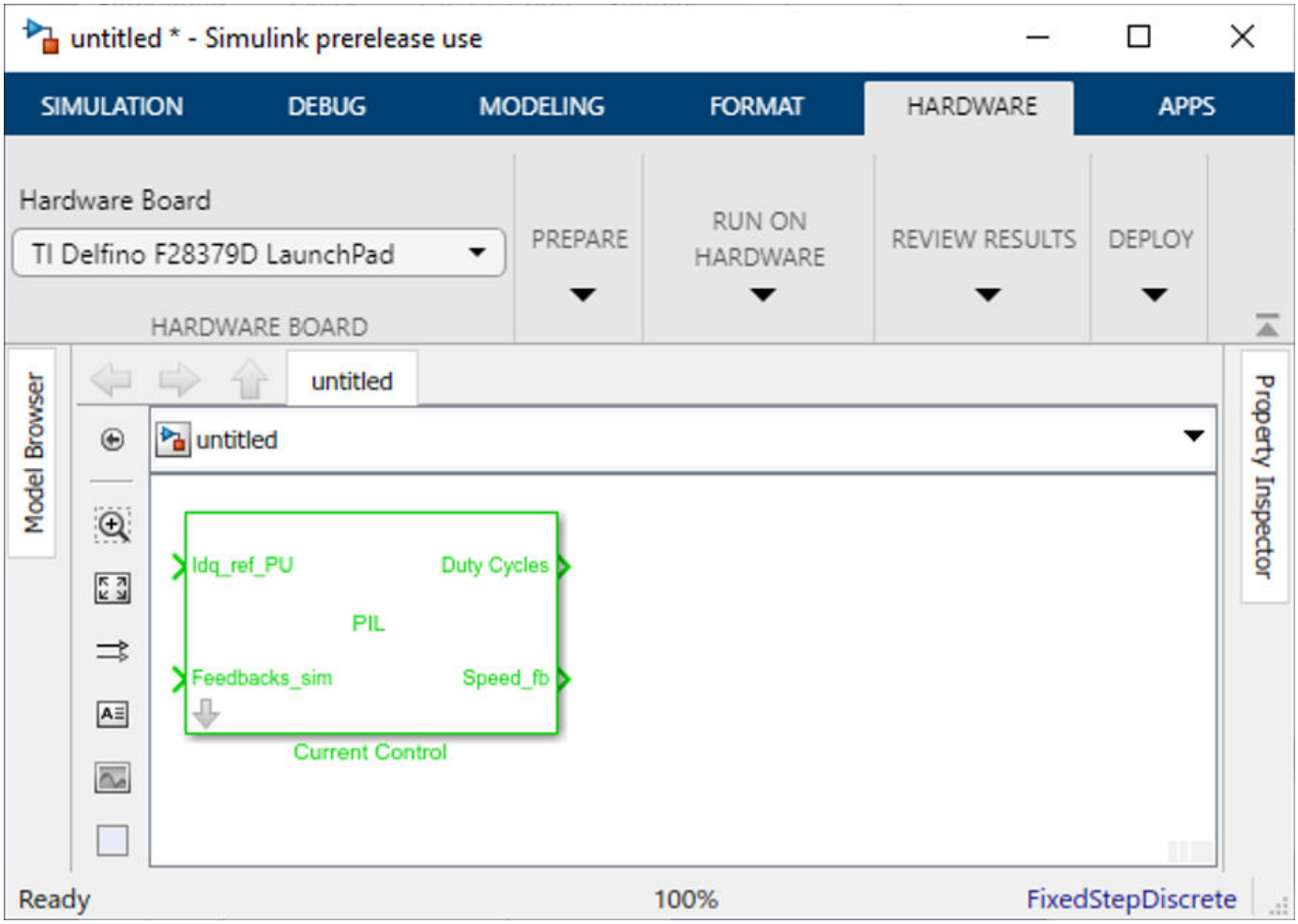
3. Run the script to update the configuration parameters of the simulation model and the PIL preferences.
4. Right-click the "Current Control" subsystem of the "mcb_pmsm_foc_sim.slx" example model and select **Deploy this Subsystem to Hardware** in the **C/C++ Code** menu.



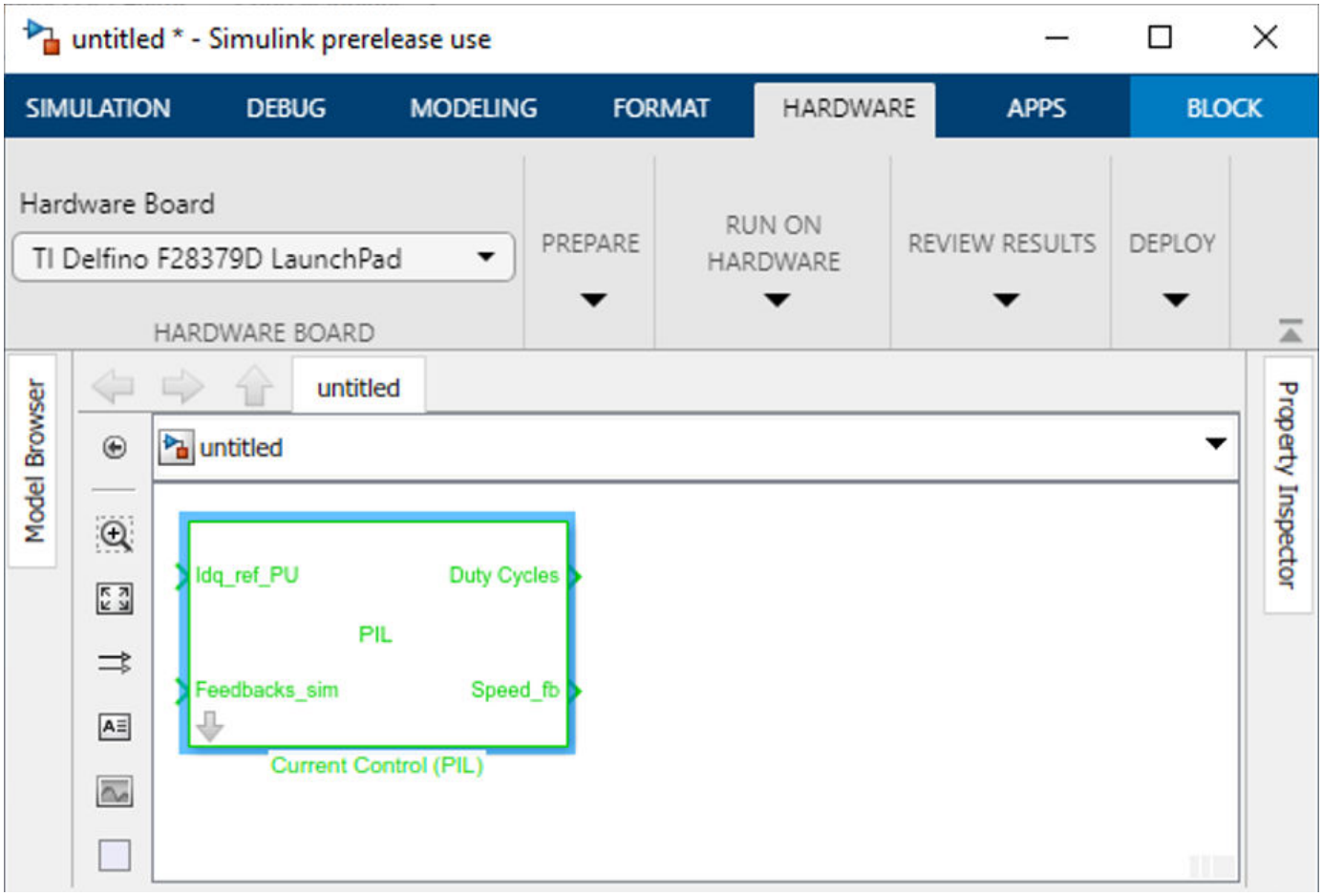
The system displays the **Build code for Subsystem** dialog box. Select `Inlined` storage class for all parameters.



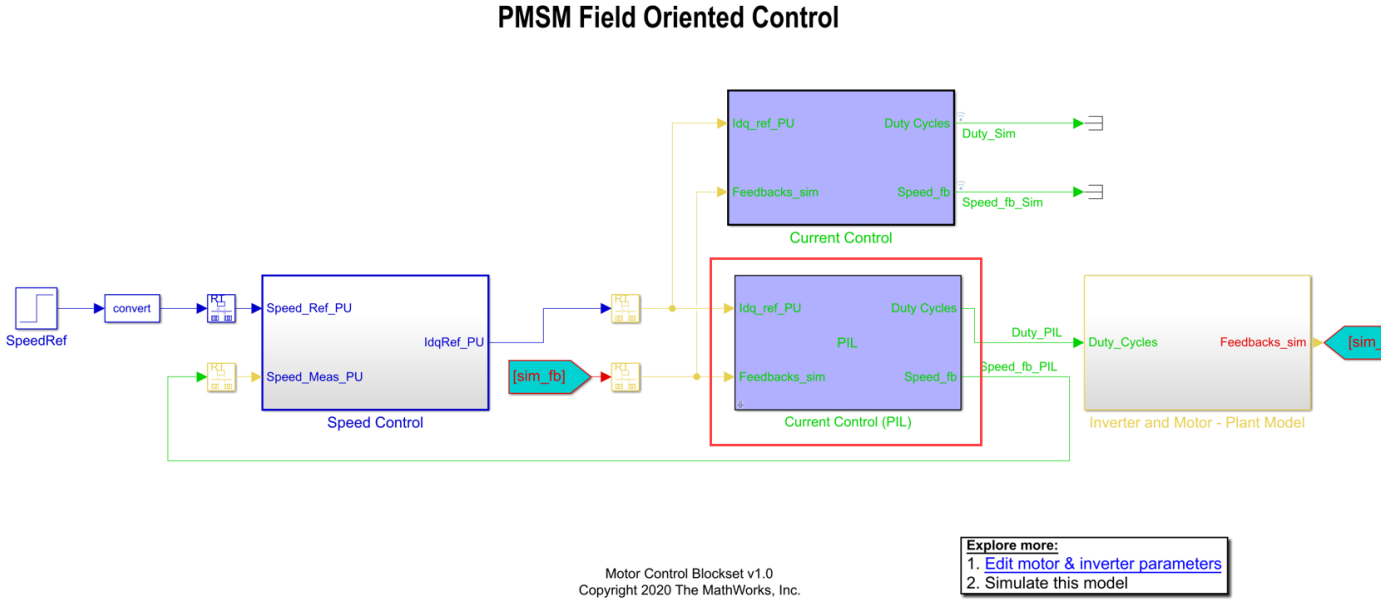
5 Click **Build** to create a model named "untitiled" that includes a PIL subsystem called "Current Control."



6 Rename the "Current Control" subsystem to "Current Control (PIL)."

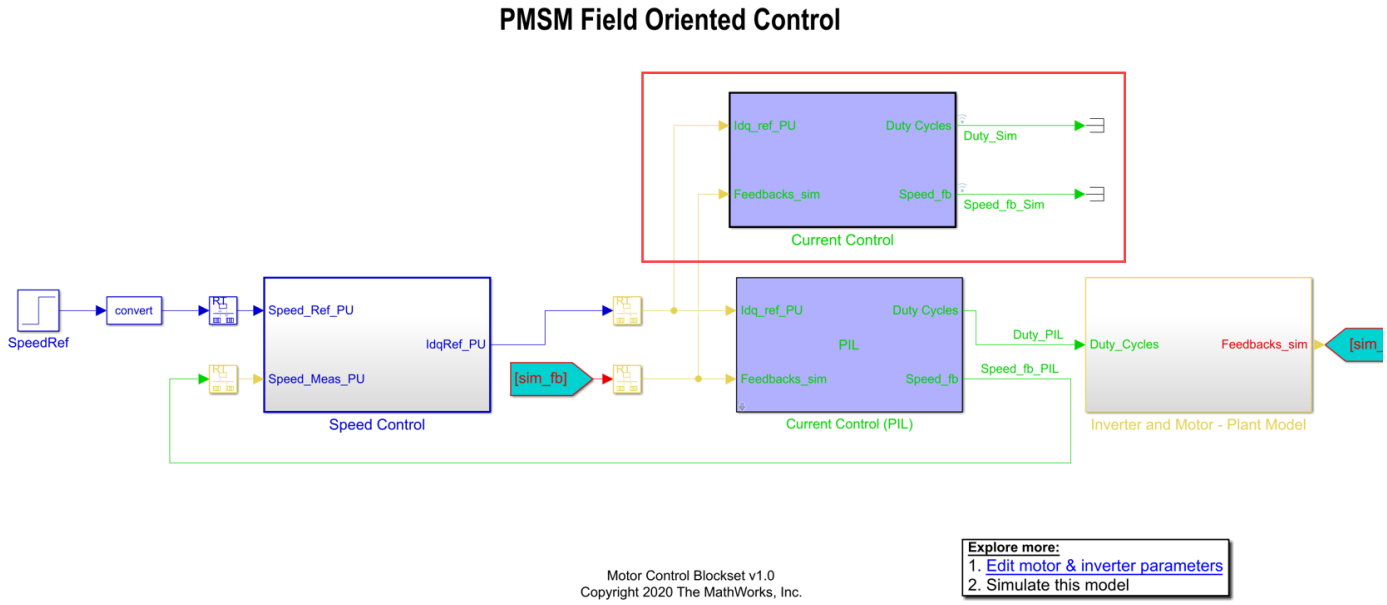


7 Copy the "Current Control (PIL" subsystem and replace the "Current Control" subsystem in the "mcb_pmsm_foc_sim.slx" example model.

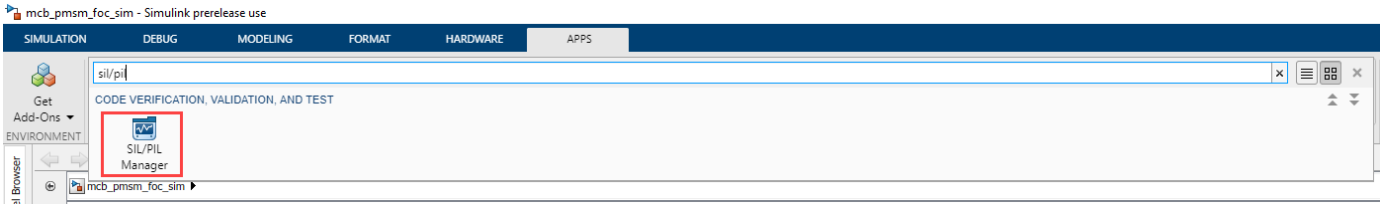


In the PIL mode, the system deploys the "Current Control (PIL)" subsystem to the target and executes the subsystem in the target hardware.

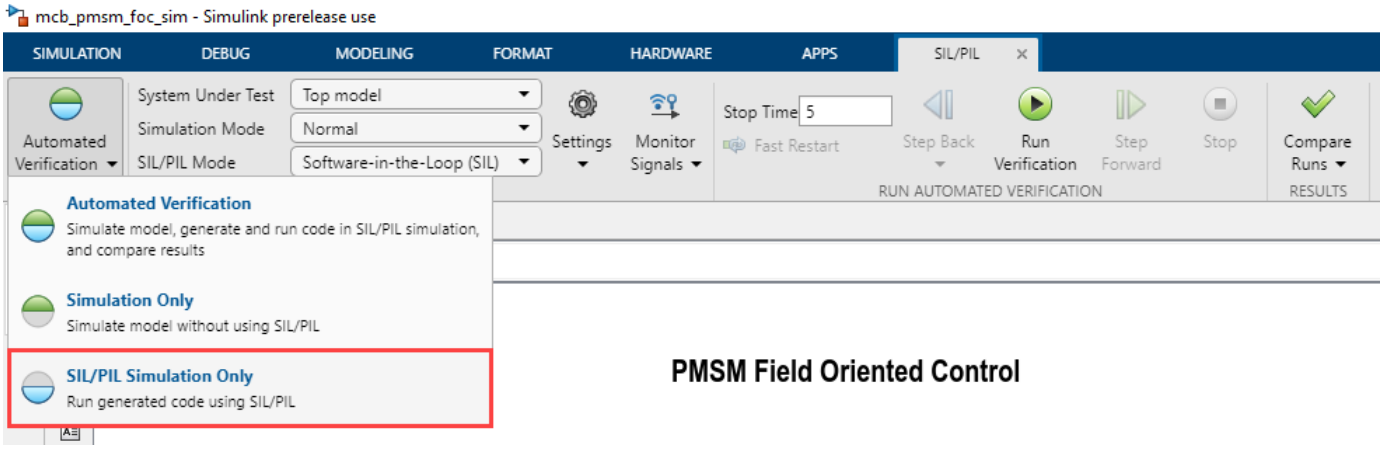
8 To compare the algorithm execution in the host machine simulation and the PIL simulation, connect the "Current Control" subsystem in parallel to the "Current Control (PIL)" subsystem. In addition, enable signal logging in the subsystem outputs.



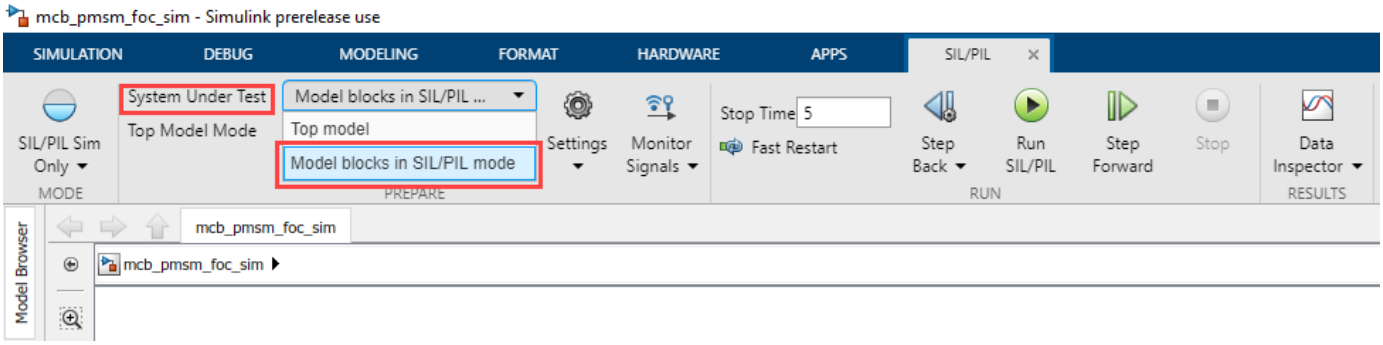
9 In the Simulink toolstrip, select the SIL/PIL Manager app in the **Apps** tab.



10 In the **SIL/PIL** toolstrip, select `SIL/PIL Sim Only`.



11 Select `Model blocks in SIL/PIL mode` in the **System Under Test** field.



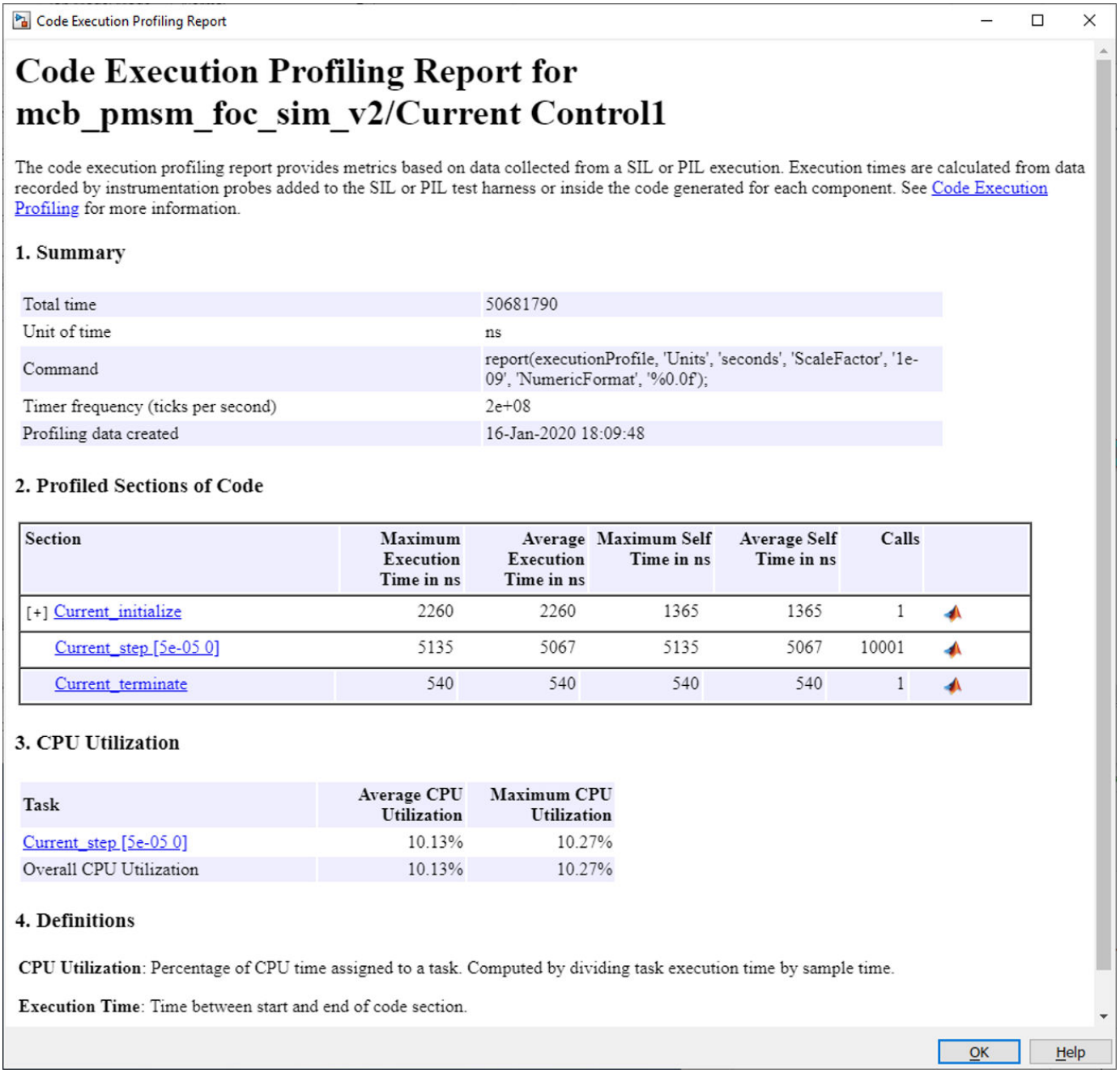
12 Click **Run SIL/PIL** on the **SIL/PIL** toolstrip to build the "Current Control (PIL)" subsystem and deploy it to the target.

After the system deploys the subsystem, the "Current Control (PIL)" subsystem executes on the target hardware processor, whereas the plant model runs on the host machine.

Analyze PIL profiling results

When PIL simulation ends, the system generates a profiling report.

Note The PIL simulation takes more time than the host machine based simulation. This happens because of the serial communications (related to inputs and outputs of the "Current Control (PIL)") between the host machine and subsystem that runs on the target hardware.



Code Execution Profiling Report

Code Execution Profiling Report for mcb_pmsm_foc_sim_v2/Current Control1

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

Total time	50681790
Unit of time	ns
Command	report(executionProfile, 'Units', 'seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	2e+08
Profiling data created	16-Jan-2020 18:09:48

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
[+] Current_initialize	2260	2260	1365	1365	1	
Current_step [5e-05 0]	5135	5067	5135	5067	10001	
Current_terminate	540	540	540	540	1	

3. CPU Utilization

Task	Average CPU Utilization	Maximum CPU Utilization
Current_step [5e-05 0]	10.13%	10.27%
Overall CPU Utilization	10.13%	10.27%

4. Definitions

CPU Utilization: Percentage of CPU time assigned to a task. Computed by dividing task execution time by sample time.

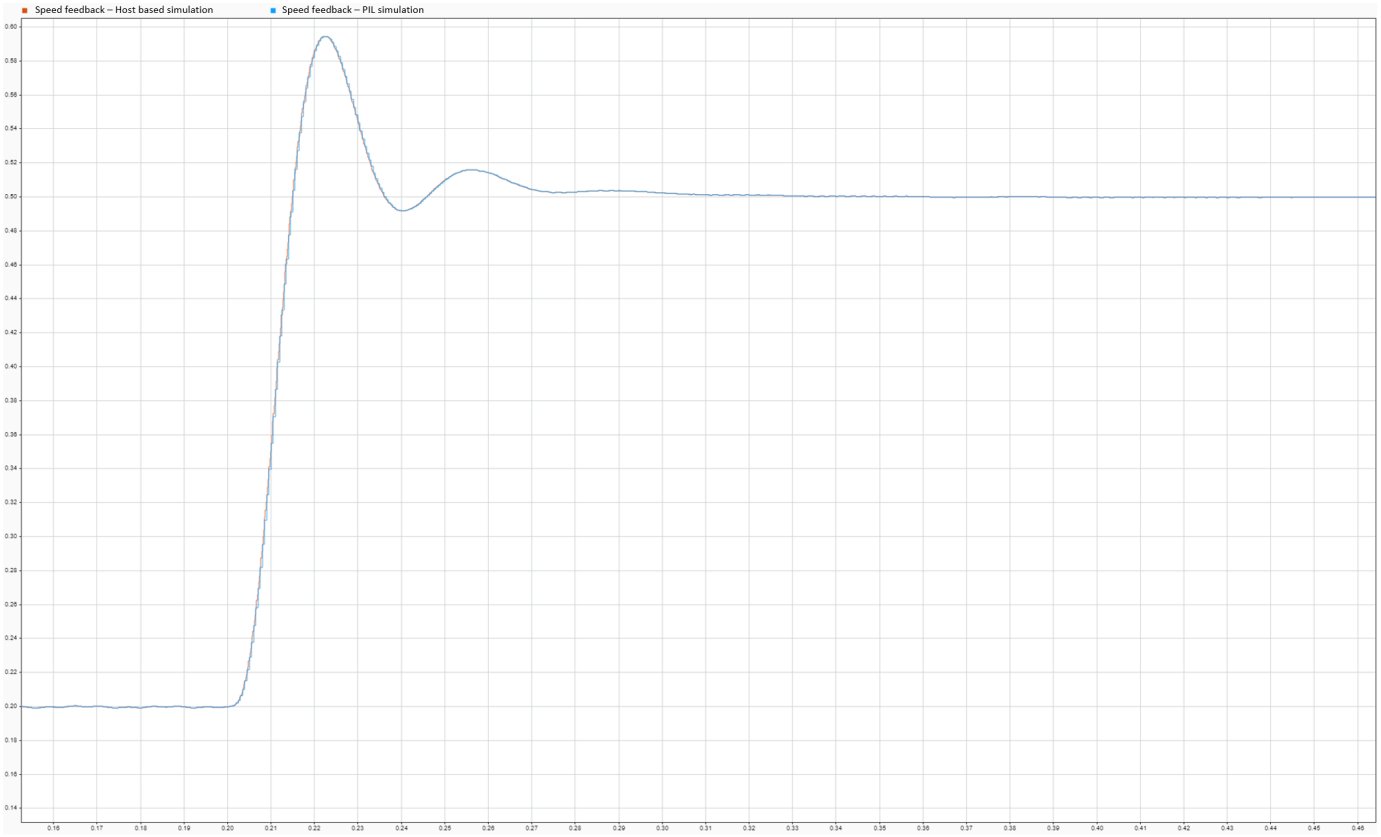
Execution Time: Time between start and end of code section.

OK Help

The profiling report shows the maximum and average execution times of the "Current Control (PIL)" subsystem running on the target hardware.

You can use **Data Inspector** on the **Simulink** tab to compare the signals logged during the host machine based simulation and the PIL simulation (executed on the target). Therefore, you can verify the accuracy of the host machine based simulation and PIL simulation.

This plot compares the "speed feedback" signals from the "Current Control (PIL)" and "Current Control" subsystems.



Note If the execution time exceeds 60% of the budgeted time, we recommend you optimize the algorithm by one of the following techniques:

- Execute from RAM.
- Offload some functionalities to CLA or other CPU.
- Scale the algorithm for every alternate cycle.
- Move the less critical functionalities like speed calculation, to a slower rate.

For more details on SIL/PIL code verification, see:

- Code Verification and Validation with PIL
- Code Execution Profiling with SIL and PIL
- SIL/PIL Manager Verification Workflow

Deploy and Validate System

Prepare Target Hardware

Perform these steps to prepare the hardware before you deploy the control algorithm, which you develop using Motor Control Blockset, to the target hardware:

Verify the Direction of Rotation of Motor

The phase sequence of the motor connection in the target hardware determines the direction of the motor rotation. The example models that are available in Motor Control Blocksetuse the position ramp-up as positive direction or the speed measured as positive. It is recommended that you to spin the motor in open loop with position ramp from 0 to 1 and ensure that the position feedback reads positive. The example models in Motor Control Blockset use this convention for direction of rotation.

The example Quadrature Encoder Offset Calibration for PMSM Motor, for the supported hardware, spins the motor and finds the offset between the d-axis of the rotor and encoder index pulse, when rotor is aligned to the stator d-axis. The host model for this example shows red LED when direction is opposite. In this case, you need to change the phase sequence of the motor wiring (swap any two motor wires).

Refer to the example Hall Offset Calibration for PMSM Motor to identify the direction of rotation for Hall sensor.

Note For the Hall sensor, ensure that you update Hall sequence in the Hall decoder library block is the same sequence as the actual hall signals. If this is not correct, the direction read by the target hardware is opposite to the actual direction.

Measure the Current Sensor Calibration

Signal conditioning for current sensor introduces an offset voltage in ADC input to measure both positive and negative current. For example, ADC with 3.3 V voltage reference has an offset of 1.65 V (BOOSTXL-DRV8305). This offset value varies on the tolerances of the passive components in signal conditioning circuit. It is recommended to measure the ADC offset of the board in the initialization.

In most of the example models in Motor Control Blockset, in the hardware initialization block, the average of current sensor ADC values are computed and used as ADC offset values for measuring the current. ADC offset values are represented in ADC counts.

Refer to the example Run PMSM in Open-loop Control and Calibrate ADC Offset for calibrate ADC offset manually and update in the script file.

Refer to the example Field Oriented Control of PMSM by Using Quadrature Encoder, Hardware Init subsystem for calculate ADC offset before starting the closed-loop control.

Position Sensor Calibration

For PMSM, the position in current control algorithm should align with the rotor d-axis position. By default, the QEP position sensor reads the mechanical position of the rotor with reference to its index pulse. The position offset is the position read by the QEP when rotor d-axis is aligned to Ph-A. The position read by QEP is corrected for this offset value to read the motor position. This corrected motor position is fed as input to current control algorithm.

Any delay between the rotor position and position in current controller affects the motor functionality and performance.

Refer the example Quadrature Encoder Offset Calibration for PMSM Motor for more details.

Refer the example for Hall sensor position offset calibration.

See Also

Add Hardware Drivers to the Simulation Model and Deploy to the Target Hardware

This section provides the steps to add hardware drivers to a simulation model and deploy the model to a target hardware.

As a reference, the example model `mcb_pmsm_foc_sim` is chosen to explain the steps involved in hardware deployment. The model `mcb_pmsm_foc_sim` simulates the FOC algorithm for PMSM speed control.

In this workflow, the target hardware LAUNCHXL-F28379D + BOOSTXL-DRV8305 is considered for deploying a speed control algorithm for PMSM. The target hardware interface details are given below:

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

These steps explain adding the hardware driver blocks from Embedded Coder Support Package for Texas Instruments C2000 Processors to deploy the control algorithm in the Target hardware (LAUNCHXL-F28379D + BOOSTXL-DRV8305).

1. “Understanding the Task Scheduling in Target Hardware” on page 2-6
2. “Adding ADC Driver Library Block” on page 2-7
3. “Adding Quadrature Encoder Driver Block” on page 2-9
4. “Add PWM Driver Block” on page 2-11
5. “Add Hardware Interrupt Trigger Block for Current Control Loop” on page 2-14
6. “Run in Open-loop and Switch to Closed-loop” on page 2-15
7. “Model Configuration and Hardware Deployment” on page 2-18

Refer Prepare Target Hardware for the pre-requisites for deploying the control algorithm in any target hardware. For hardware details, refer to Hardware connections

Note In these workflow steps, variables are used for defining datatypes, execution time of current controller, and execution time of speed controller. Refer to the initialization script of the example model `mcb_pmsm_foc_sim` for the details of the variables defined in these steps.

Refer to Control Algorithm Design to implement a simulation model for PMSM motor FOC control algorithm.

Tip A basic understanding of Simulink is a prerequisite for following this workflow. Refer the example model `mcb_pmsm_foc_qep_f28379d` for the details of ADC driver, QEP driver and Hardware interrupt block as the same strategy is explained in these steps.

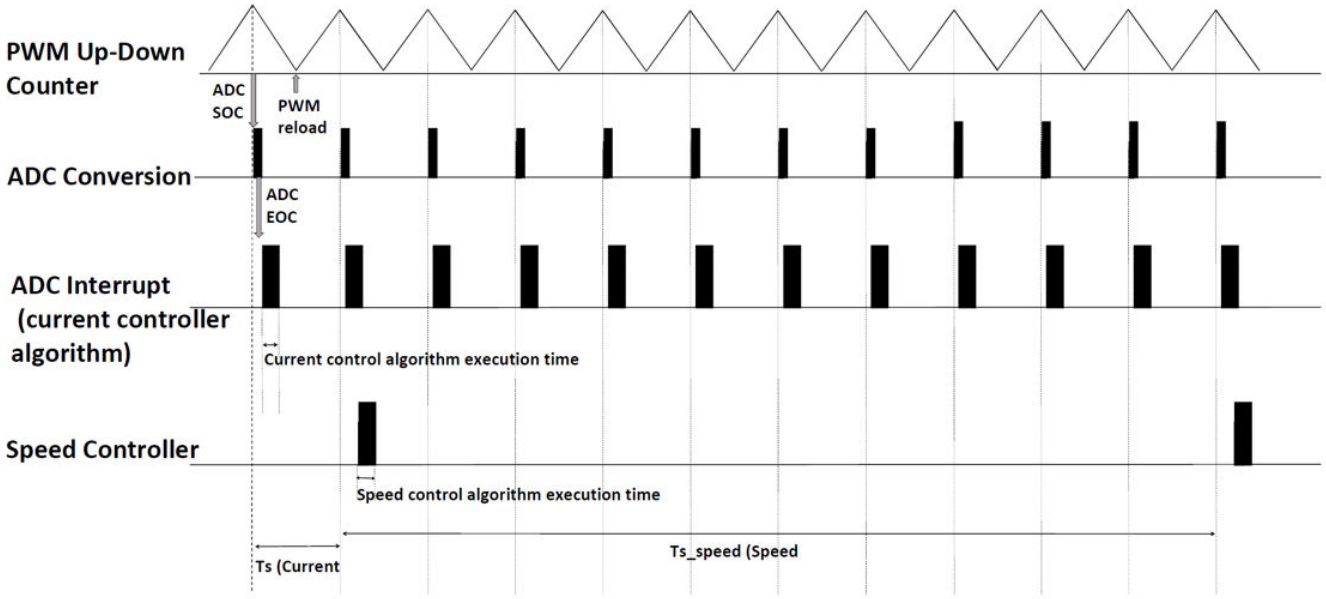
Note For target hardware other than LAUNCHXL-F28379D + BOOSTXL-DRV8305, follow the workflow steps as explained in these steps, but select the driver blocks (ADC, PWM, Interrupt) from the appropriate supported hardware library.

Understanding the Task Scheduling in Target Hardware

In the example model `mcb_pmsm_foc_sim`, the configuration of current controller and speed controller are the two major software tasks. The current controller is scheduled to run for every Ts (50 µsec for 20 kHz switching frequency) and the speed controller is Ts_speed (10*Ts). Current controller reads the motor phase currents, reads the position from QEP positon decoder and computes the PWM duty cycle to run the motor. Speed controller runs control loop and calculates Iq reference for the current controller and thereby controls motor speed in closed-loop.

In the target hardware, current controller is synchronized with ADC interrupt (for every Ts) and speed controller is time triggered for every Ts_speed (10*Ts).

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



The event sequence is:

1. The processor peripheral PWM, which is center aligned (Up-Down Counter), triggers the SOC (start-of-conversion) event to ADC module when PWM-counter-value equals the PWM-period.
2. ADC module converts the sampled analog signal to digital counts and triggers EOC (end-of-conversion) event.
3. ADC's EOC triggers the ADC interrupt.
4. Current controller is scheduled to execute in ADC interrupt.
5. Speed controller is time-triggered and scheduled for every Ts_speed.

In the above figure, Current controller execution time and Speed controller execution time are not in scale. Refer to the processor datasheet for better understanding the functionality of the processor peripherals ADC (Analog-to-digital converter) and PWM (Pulse-width modulation).

Adding ADC Driver Library Block

In the example model `mcb_pmsm_foc_sim`, the Current controller block receives motor phase current in ADC counts. Plant modelling converts the motor phase current in Ampere to ADC counts. In the target hardware, current controller reads the motor phase current from ADC block.

For BOOSTXL-DRV8305, the motor's Phase-A current is read from ADC C2 and Phase-B current is read from ADC B2. Select ADC module C, channel 2, for Motor Phase-A current; and ADC module B, channel 2, for Motor Phase-B Current. For other target hardware, select the ADC module and channel where motor phase currents are interfaced.

Select `ePWM1_ADCSOCA` as trigger source in ADC block as PWM library block triggers SOC0 (start-of-conversion) event when PWM counter equals PWM period register.

Select `ADCINT1` in ADC B module and this triggers ADC interrupt at EOC (end-of-conversion). In ADC interrupt, FOC Current control algorithm is scheduled to execute.

In the Simulink library browser, select the ADC library block from Embedded Coder Support Package for Texas Instruments C2000 Processor > F2837xD. Configure the ADC block to read the motor's Phase-A and Phase-B current.

In ADC block mask dialog box, configure the ADC C module, Channel 2, to read Motor Phase-A current, as shown in the below table.

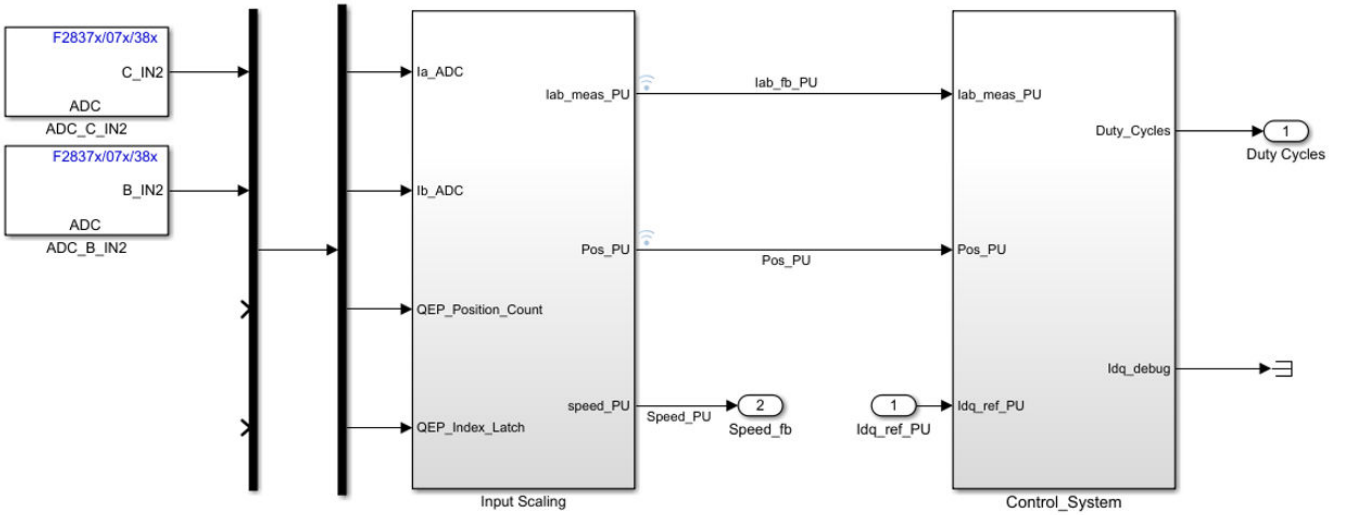
Tab and Parameter in ADC Block	Settings
SOC Trigger > ADC Module	**C**
SOC Trigger > SOC trigger number	**SOC0**
SOC Trigger > SOC trigger source	**ePWM1_ADCSOCA**
Input Channels > Conversion channel	**ADCIN2**

Rename the block as ADC_C_IN2

In ADC block mask dialog box, configure the ADC B module, channel 2 to read Motor Phase-B current and trigger ADC interrupt (ADCINT1), as shown in the below 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 AOC trigger check box	on
SOC Trigger > Interrupt selection	**ADCINT1**
SOC Trigger > ADCINT1 continuous mode check box	on
Input Channels > Conversion channel	**ADCIN2**

Rename the block as ADC_B_IN2.



Adding Quadrature Encoder Driver Block

In Simulink library browser, select eQEP block from Embedded Coder Support Package for Texas Instruments C2000 Processor > F2837xD.

eQEP block reads the QEP pulses and increment the position count. This block outputs QEP encoder pulse for mechanical rotor position wrap around when QEP Index pulse is read.

Refer section Quadrature Encoder Interface Configuration in Model Configuration Parameters for Sensors for QEP configurations.

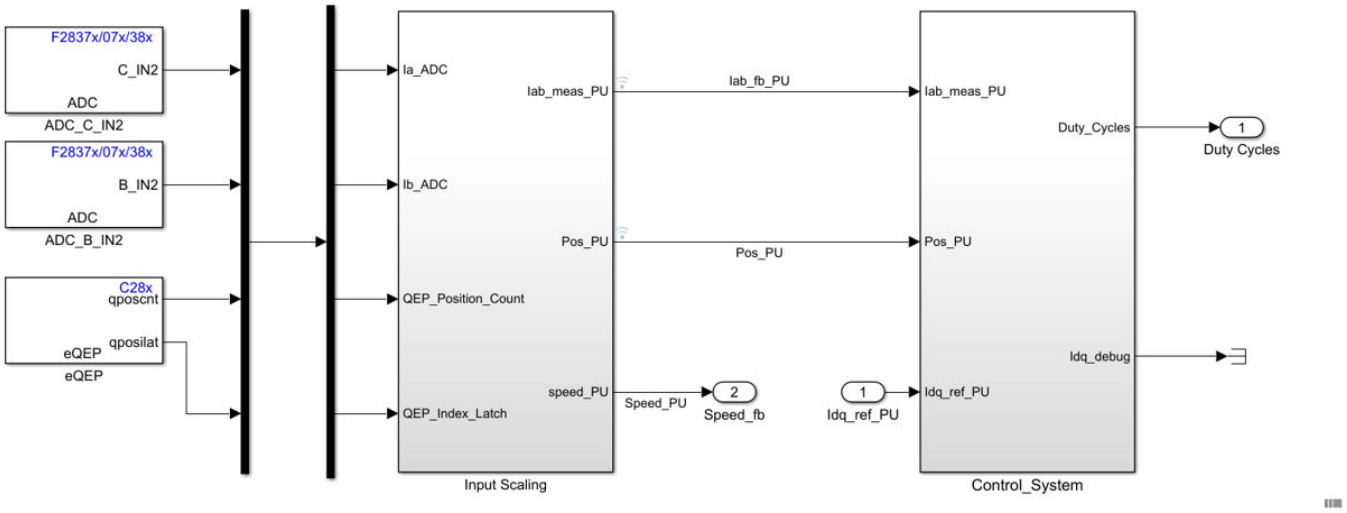
In c28x eQEP block mask dialog, configure the QEP (encoder pulse) to read QEP pulse count in TI processor and wrap the QEP pulse counter output when index pulse is found, as shown in the below table.

Tab and Parameter in eQEP Block	Settings
General > Module	**eQEP1**
General > Sample time	**-1**
Position counter > Output position counter check box	on
Position counter > Maximum position counter value (0~4294967295)	**2^16-1**
Position counter > Position counter reset mode	**Reset on the first index event**
Position counter > Output latch position counter on index event check box	on
Position counter > Index event latch of position counter	**Falling edge**

Rename the block as eQEP.

eQEP1 module is selected because QEP is connected to the QEP_A interface in LaunchPadXL28379d. The Sample time is selected as -1 because the library block is in function-call triggered by ADC interrupt synchronously. Maximum position counter value is 2^16-1 because Position counter is 16-bit in library driver block. Position counter reset mode on index pulse wraps the position count on index pulse.

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


F2837x/07x/38x
C_IN2
ADC
ADC_C_IN2
F2837x/07x/38x
B_IN2
ADC
ADC_B_IN2
C28x
qposcnt
qposilat
eQEP
eQEP
Ia_ADC
Ib_ADC
QEP_Position_Count
QEP_Index_Latch
Iab_meas_PU
Pos_PU
speed_PU
Input Scaling
Iab_fb_PU
Pos_PU
Speed_PU
2
Speed_fb
1
Idq_ref_PU
Iab_meas_PU
Pos_PU
Idq_ref_PU
Duty_Cycles
Idq_debug
Control_System
1
Duty Cycles

Add PWM Driver Block

In Simulink library browser, select ePWM block from Embedded Coder Support Package for Texas Instruments C2000 Processor > F2837xD

Configure ePWM1, ePWM2, ePWM3 for generating the PWM pulse. In the ePWM block mask dialog, calculate the PWM counter period register value from the CPU frequency and PWM frequency. For center-aligned PWM, divide by 2.

PWM counter period = CPU clock frequency / PWM frequency /2

Refer to the TMS320f28379d processor: ePWM peripheral for more details.

In the F2837x/07x/004x/38x ePWM block mask dialog, update the settings to configure PWM1 to generate PWM pulses in the target hardware, as shown in the below table.

Tab and Parameter in ePWM Block	Settings
General > Module	**ePWM1**
General > Timer Period	Enter the PWM period value in CPU clock cycle • 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 check box	on
Deadband unit > Use deadband for ePWM1B check box	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 F2837x/07x/004x/38x ePWM block mask dialog, 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 the configurations as shown in the below table:

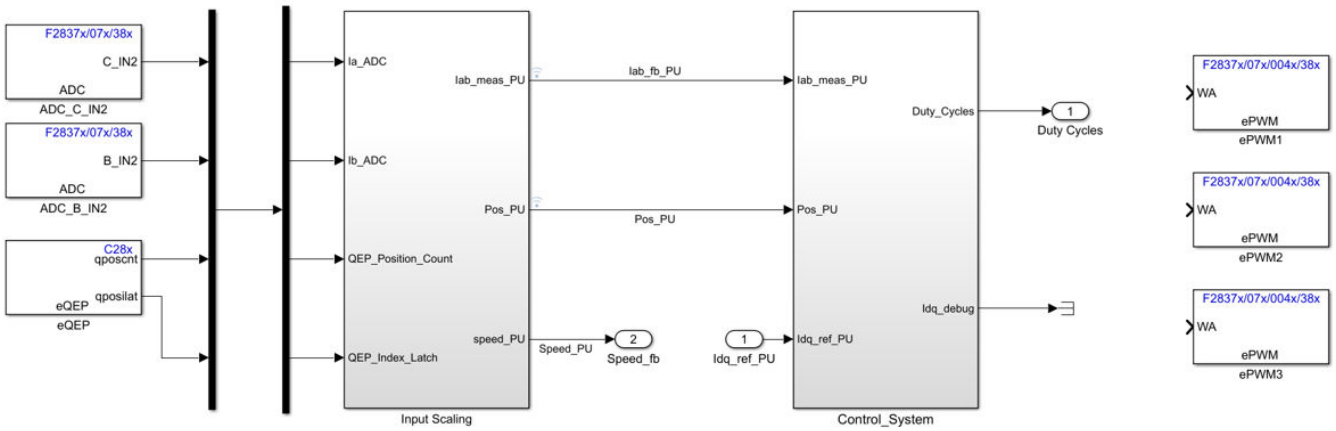
Tab and Parameter in ePWM Block	Settings
General > Module	**ePWM2**
General > Timer Period	Enter the PWM period value in CPU clock cycle • PWM counter period = CPU clock frequency / PWM frequency /2 • For launchpad 28379d, clock frequency is 200 MHz. For PWM frequency of 20 kHz, PWM counter period = 200e6 / 20e3 / 2; PWM counter period = 5000
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 check box	`on`
Deadband unit > Use deadband for ePWM1B check box	`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.

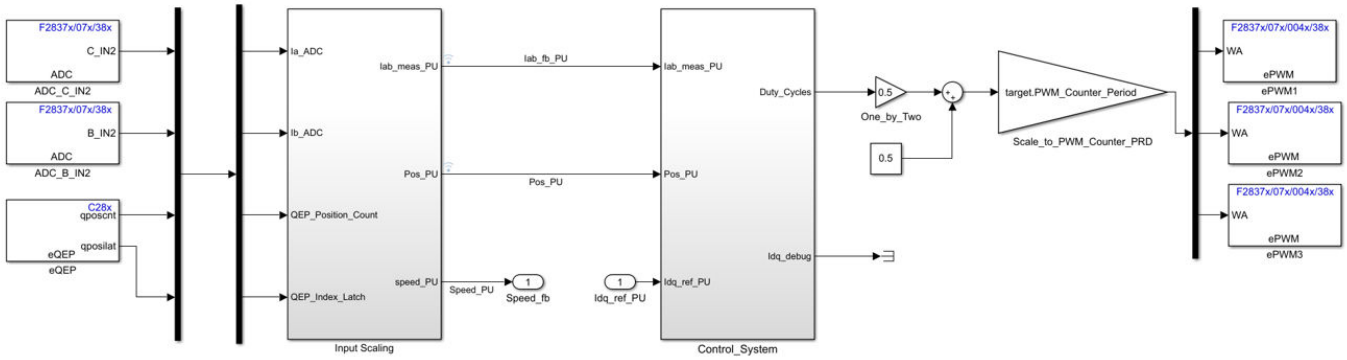
CMPA and CMPB are selected as input ports where PWM duty is given as input. The range vary from 0 to *PWM_counter_period*. PWM outputs when PWM up-counter matches CMPA and PWM down-counter matches CMPB. By default, duty cycle 50% is input by selecting PWM counter period/2.

Enable dead time in the ePWM configuration. In the Event trigger of PWM1, select ADC SOC event when PWM counter equals the PWM period. In ADC, select the start trigger as ePWM1_ADCSOCA.

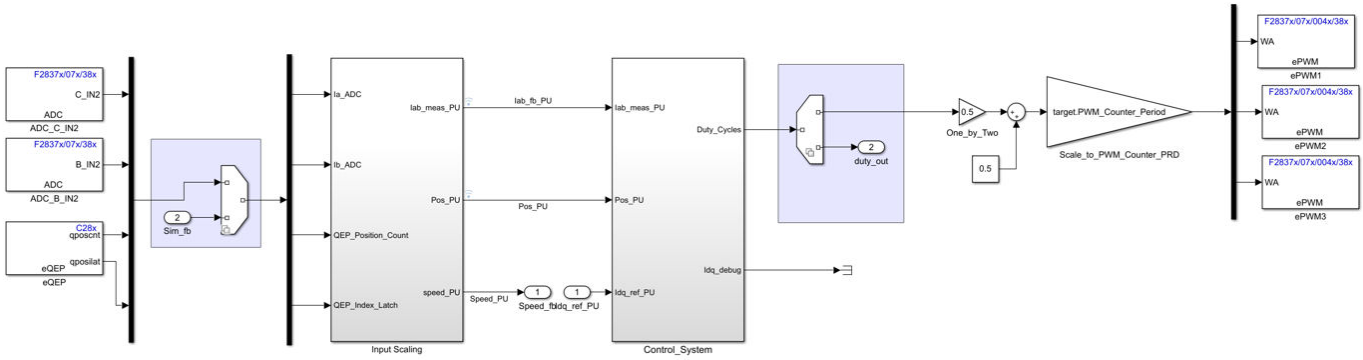
Synchronize ePWM2 and ePWM3 with ePWM1 by selecting the synchronizing when PWM counter is 0 in ePWM2 and ePWM3.



ePWM blocks expects the duty cycle value to range from 0 to period counter register (5000). Control_System outputs PWM in the range -1 to 1. The Control_System subsystem output -1 to 1 is scaled to 0 to 5000 (period counter value).



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



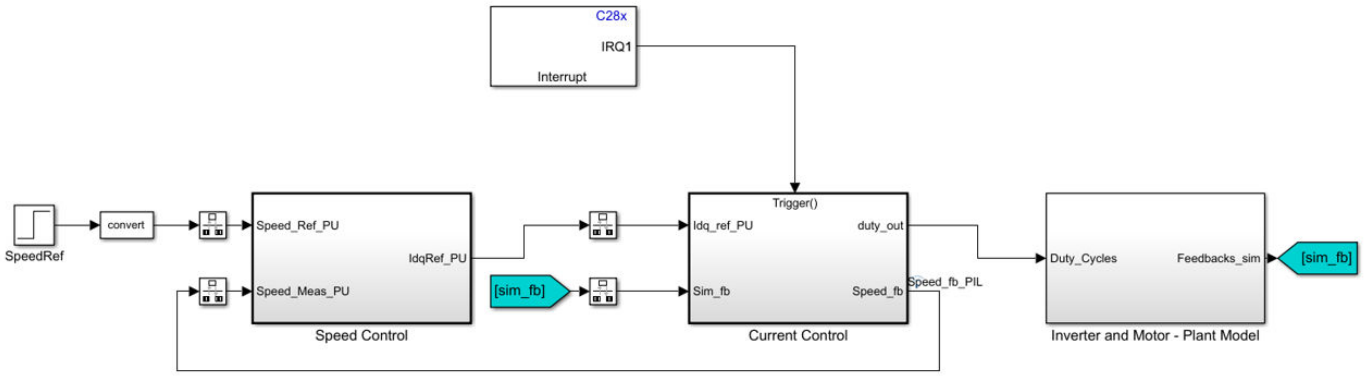
Add Hardware Interrupt Trigger Block for Current Control Loop

In Simulink library browser, select C28x Hardware Interrupt block from Embedded Coder Support Package for Texas Instruments C2000 Processor > Scheduling.

In C28x Hardware Interrupt block mask dialog, update the settings to configure hardware interrupt ADCINT1. From the C28x Hardware Interrupt, identify the PIE and CPU for ADCINT1.

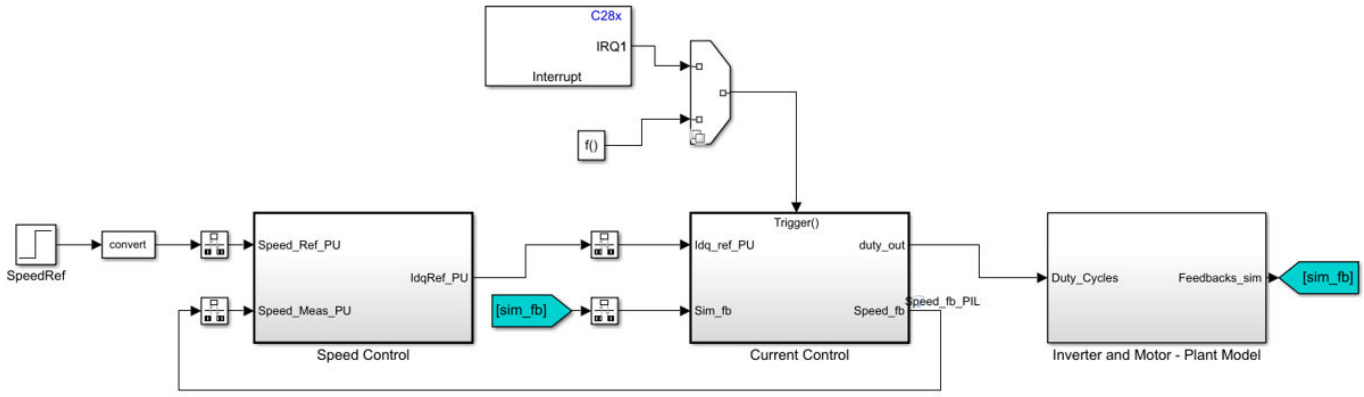
Parameter in C28x Hardware Interrupt Block	Settings
CPU interrupt numbers	**[1]**
PIE interrupt numbers	**[2]**

In Current control, add a Trigger block and change the Trigger type to `function-call`. Connect this subsystem trigger input to the Hardware interrupt block as shown in the below figure.



In the Rate Transition block's input to Current Control, change the sample time to -1.

Add a Function-Call Generator block in variant source to support the model simulation. In Function-Call Generator, set the Sample time as Ts (50e-6).



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

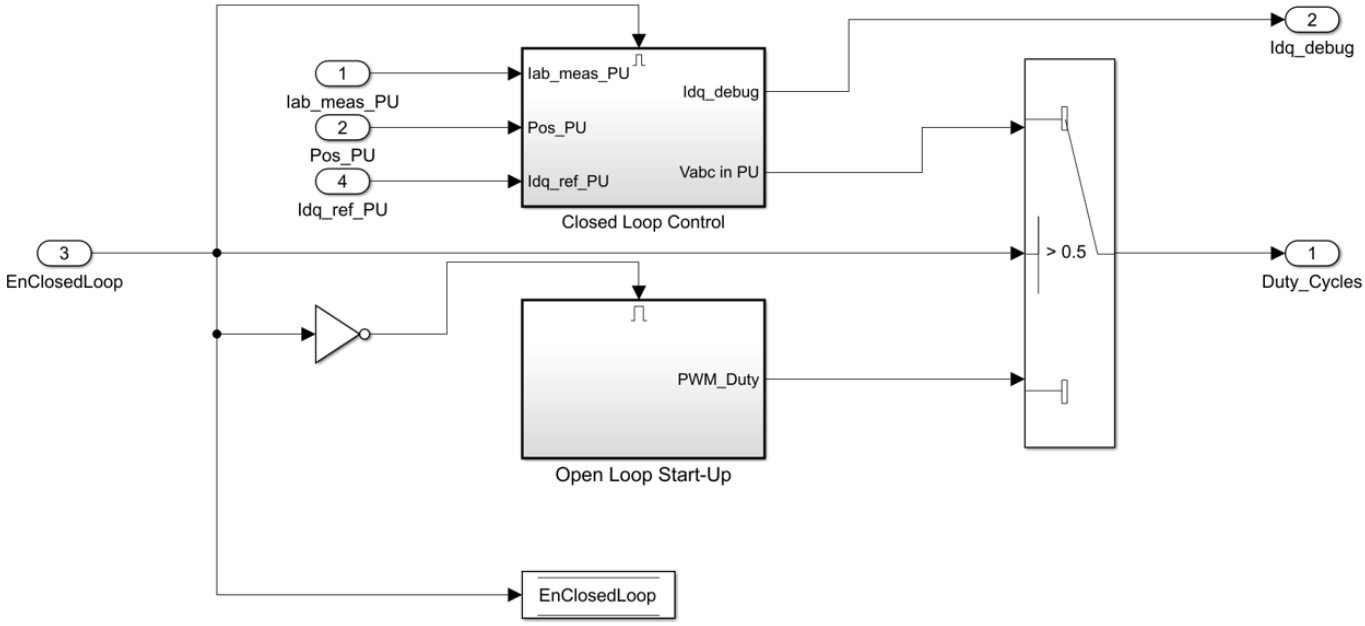
Run in Open-loop and Switch to Closed-loop

PMSM motor with QEP requires initial position to start the motor. Because we do not have a way to know the initial position at start, you need to spin the motor in open-loop and ensure QEP index pulse is read at least once. At QEP index pulse, QEP resets its position to align with the motor mechanical angle. The motor switches from open-loop run to closed-loop speed control to maintain the reference speed. This step is applicable for QEP sensor and not for Hall position sensor. Hall signal outputs initial position of the rotor segment from Hall signal port inputs.

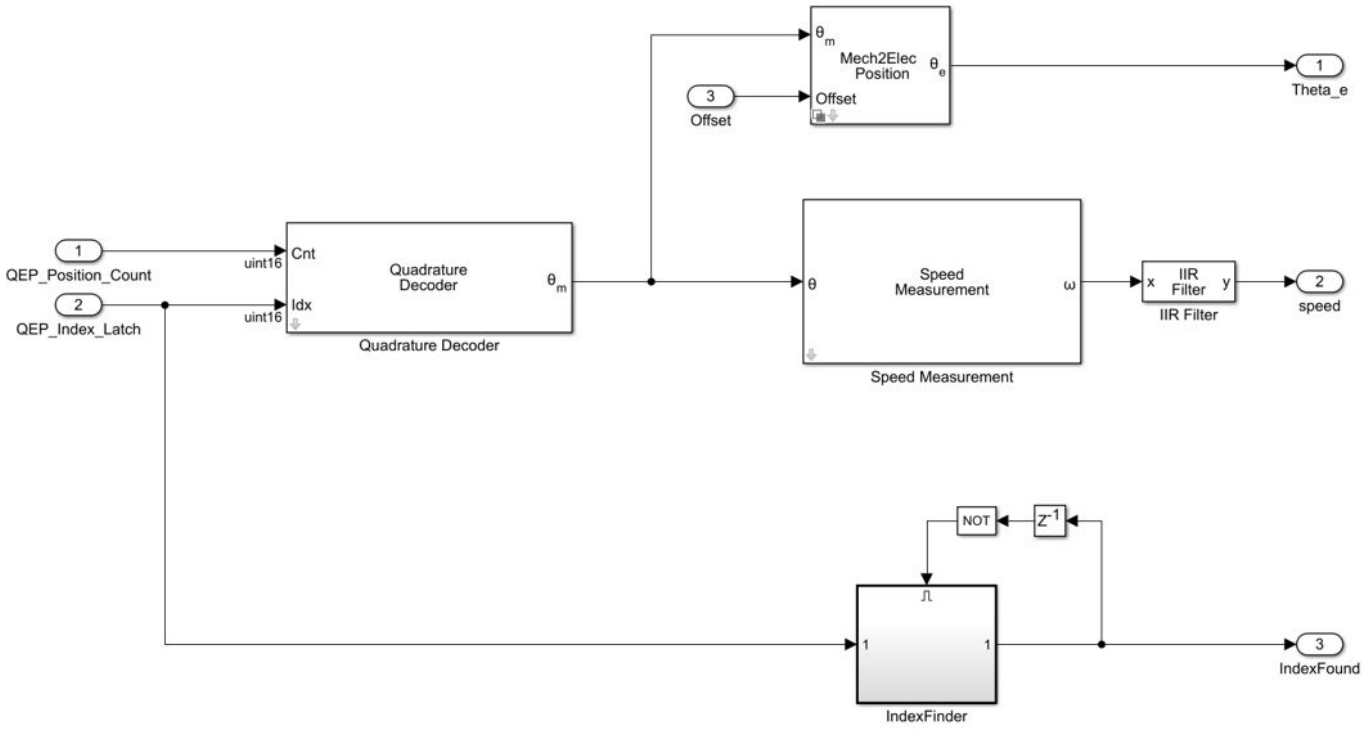
Follow these steps to implement open-loop run and switching to closed-loop control:

1. Copy `mcb_pmsm_foc_qep_f28379d/Current Control/Control_system`. This adds the logic to run the motor in open-loop. This switches the control from open-loop to closed-loop if *EnClosedLoop* input is 1. Add an input port *EnClosedLoop*. This adds data store read for *Enable* and *SpeedRef*. Add a data store memory *Enable*, *EnClosedLoop* and *SpeedRef* in Top model level.

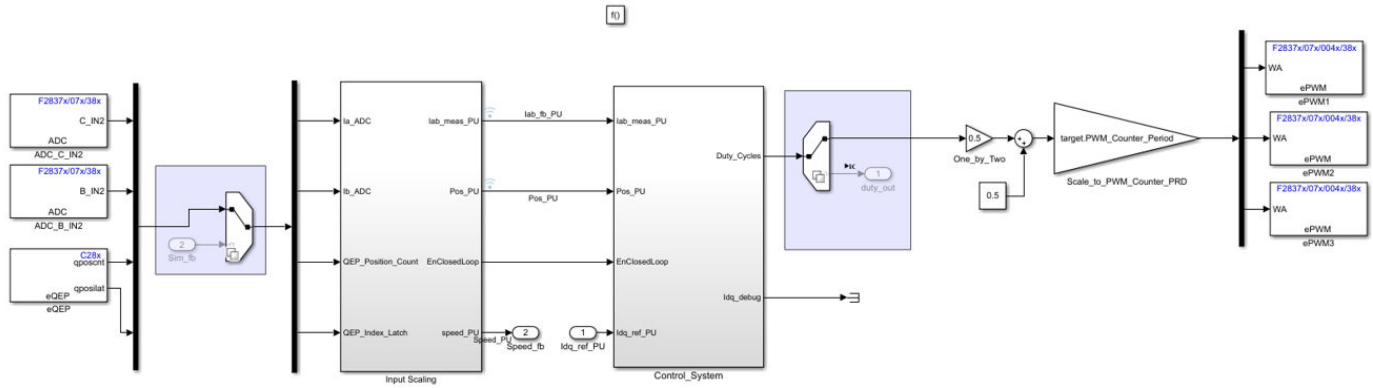
   In Open Loop Start-up, the sign of the *SpeedRef* decides the direction of the initial run. If *SpeedRef* is negative, motors spins in opposite direction in open loop start-up.



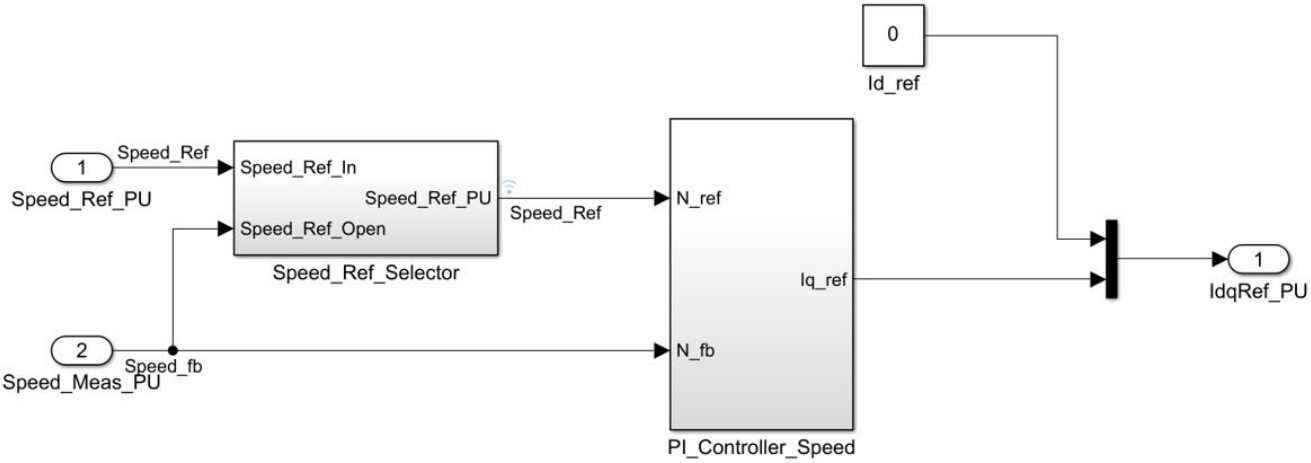
2. Copy `mcb_pmsm_foc_qep_f28379d/Current Control/Input Scaling/Calculate Position and Speed`. This adds a block `IndexFinder`. When QEP index pulse is detected for the first time, `IndexFound` is set as 1. Add an output port `IndexFound` and rename as `EnClosedLoop`.



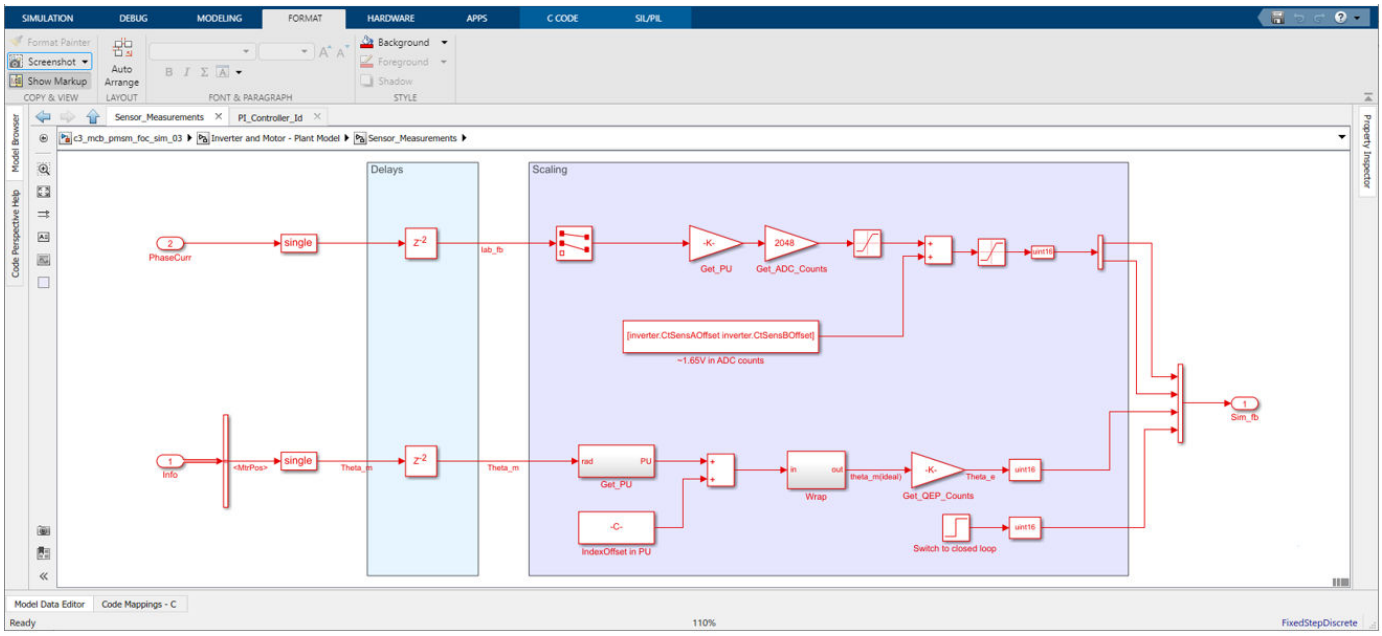
3 Connect the output port *EnClosedLoop* from Input Scaling subsystem to Control_System input port *EnClosedLoop* as shown in the below figure.



4 Copy `mcb_pmsm_foc_qep_f28379d/Speed Control/Speed_Ref_Selector`. This block uses `speed_ref` when closed-loop control starts. For smooth transition from open-loop to closed-loop, the speed measured is commanded as speed reference in open-loop. Add a data store write, *SpeedRef* to the `PI_Controller_Speed` input.



5 In Plant modelling, a step input is added to simulate the *IndexFound* for simulation. Rename the step input as `Switch to closed loop`. Refer to `mcb_pmsm_foc_qep_f28379d/Inverter and Motor -Plant Model/Sensor_Measurments` for the step input to switch to closed loop. Choose Step time as 0.1 and sample time as Ts_motor.



6 Create Data store memory for *EnClosedLoop*, *Enable* and *SpeedRef*. *Enable* is used to reset the PI integrator before spinning the motor. Add default values for data store memory: *Enable* = 1, *EnClosedLoop* = 0, and *SpeedRef* = 0.25.

Note Data store memory is used to share the data across the subsystem

7 Run the simulation and observe the speed reference and speed feedback.

Model Configuration and Hardware Deployment

Select the target hardware in the model configuration. To do this, open the Configuration Parameters dialog box, go to **Hardware Implementation > Hardware board**, and select **TI Delfino F28379D LaunchPad**.

For any other custom board, select the appropriate processor and edit the peripheral details in **Hardware board > Target hardware resources**.

Refer Model Configuration Parameters for Sensors for solver configurations, Quadrature Encoder Interface configuration.

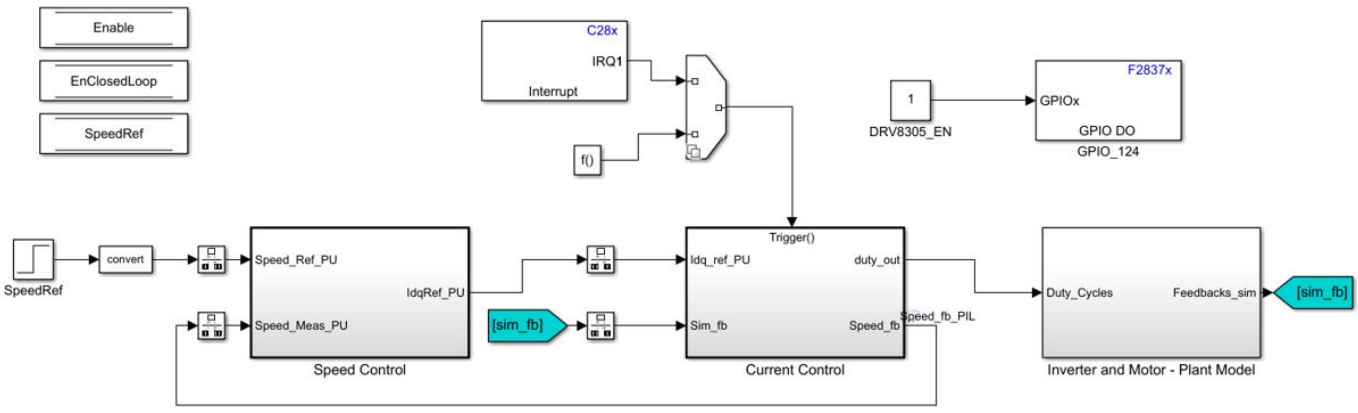
Refer Hardware Connections for C2000 LaunchPadXL hardware connection. Connect the BOOSTXL-DRV8305 and QEP connector to the LaunchPad XL. The BOOSTXL-DRV8305 fixed to the LaunchPadXL requires Enable signal. This signal is connected to the GPIO124 of the processor.

In Simulink library browser, select Embedded Coder Support Package for Texas Instruments C2000 Processor > F2837xD > Digital Output, and change the settings as shown in this table.

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

Rename the block as GPIO_124.

Add a constant block with value 1 as input to the GPIO124 as shown in the figure below:



In the model, select **Build, Deploy & Start** in the **Hardware** tab. This generates C-code, CCS project and target-specific out file. This target specific out-file is downloaded to the target through serial communication and runs the algorithm in hardware.

When model is deployed to the target, motor spins in open-loop and starts running in closed loop speed control. In order to monitor and debug the signals, serial communication is recommended. Refer the example model `mcb_pmsm_foc_qep_f28379d` for implementing the serial receive and transmit to the host model. From serial receive block update the data store memory to start and stop the motor.

Validate the System

In this section...

This section explains how to evaluate the accuracy of the plant (motor and inverter) model of physical motor and load connected to the motor. You need to validate the plant model and verify the results are close to the physical system measurements before using the plant model for Advanced algorithm implementations. You can validate the system by comparing the step response of speed control and current control in simulation and in target hardware connected to the motor.

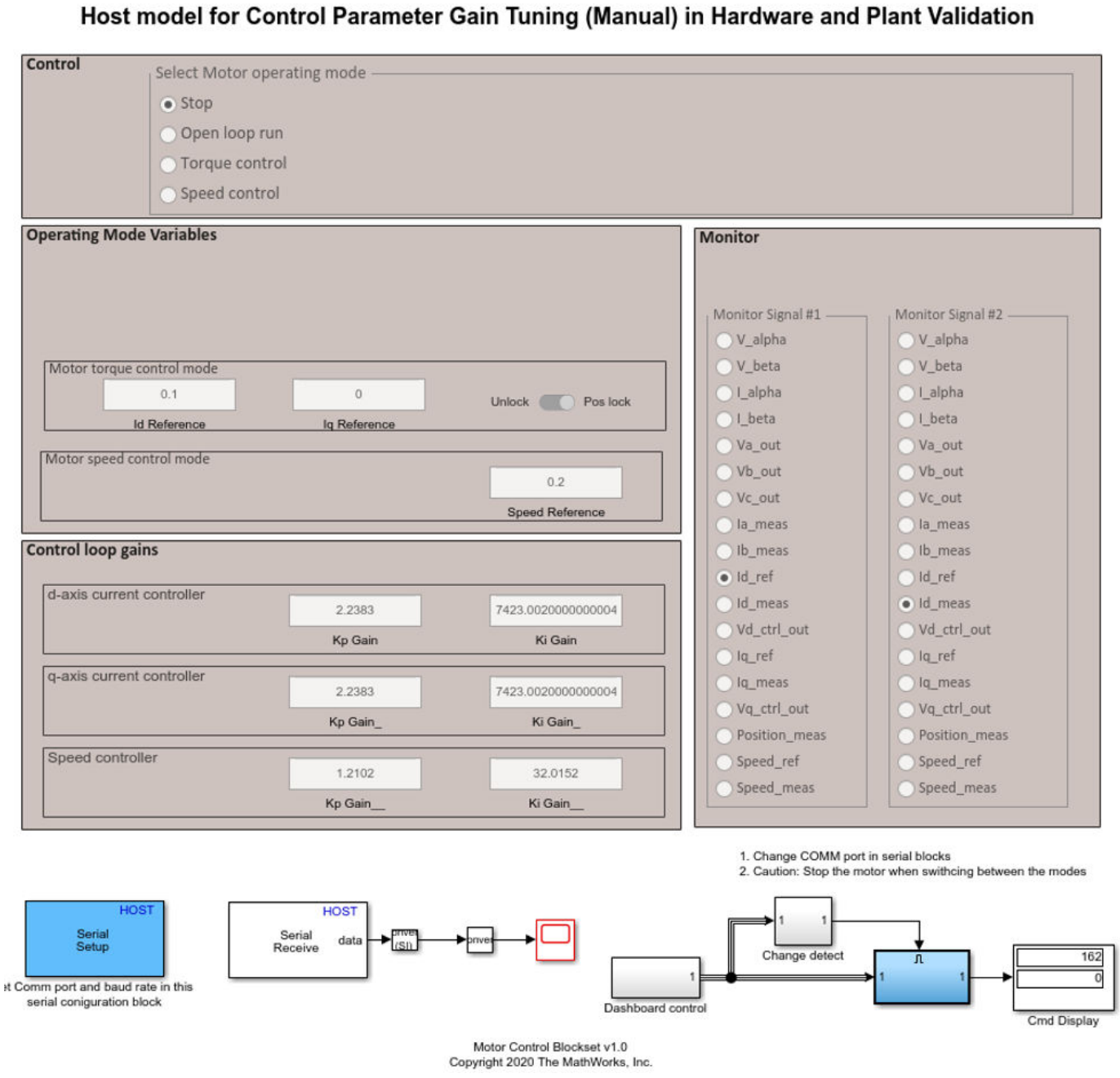
Use the example Control Parameter Gain Tuning (Manual) in Hardware and Plant Validation to measure the step response of current controller and speed controller. The host model in this example communicates the current reference to the target hardware and measures the step response of the current controller.

Note

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

  There is also another way to validate d-axis current control: Run the motor at a constant speed and give step change in reference d-axis current. This requires two modifications in the Speed Control subsystem of the target model. Set a constant speed reference input. Command Id reference from host model. Compare the step response of d-axis current in simulation and in hardware tests.
- Validate 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 Id and Iq reference from Speed PI Controller output. Command Id reference from host model. Compare the step response of q-axis current in simulation and in hardware tests.

Warning While taking step response in d-axis current control, always use positive step. Negative values of Id can damage the Permanent Magnet in the motor.



Refer the example Control Parameter Gain Tuning (Manual) in Hardware and Plant Validation for deploying the model to the hardware. Perform motor parameter estimation as accuracy of the plant modelling is important to match the simulation results with hardware measurements.

Calculate the Physical Motor Load in Target Hardware

Before comparing the controller responses in simulation and target hardware, the load torque in plant simulation must match the motor load in physical system. Follow the below steps to calculate the load torque in physical system and update the calculated load torque to the plant model:

1. Run the host model and connect the target hardware through serial communication.
2. In Select Motor Operating mode, select **Speed control**.

   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` from scope.
4. Convert the PU current to Amperes by multiplying with `PU_System.I_base`.
5. Calculate load torque in Nm using the below formula:

   Tload = 1.5 * pole_pair [flux_pm * Iq + (Ld – Lq) id*Iq]

   where,

   Flux_pm = permanment magnet flux linkage (pmsm.Flux_PM)

   Ld, Lq = Inductance in H (pmsm.Ld, pmsm.Lq)

   Id, Iq = current measured in A

   Note The measured Id current `Id_meas` (in PU) equals 0.
6. In `mcb_pmsm_operating_mode_f28379d/Motor and Inverter/Plant Model` (sim), enter the above calculated value in `LdTrq` as an input to PMSM motor block.

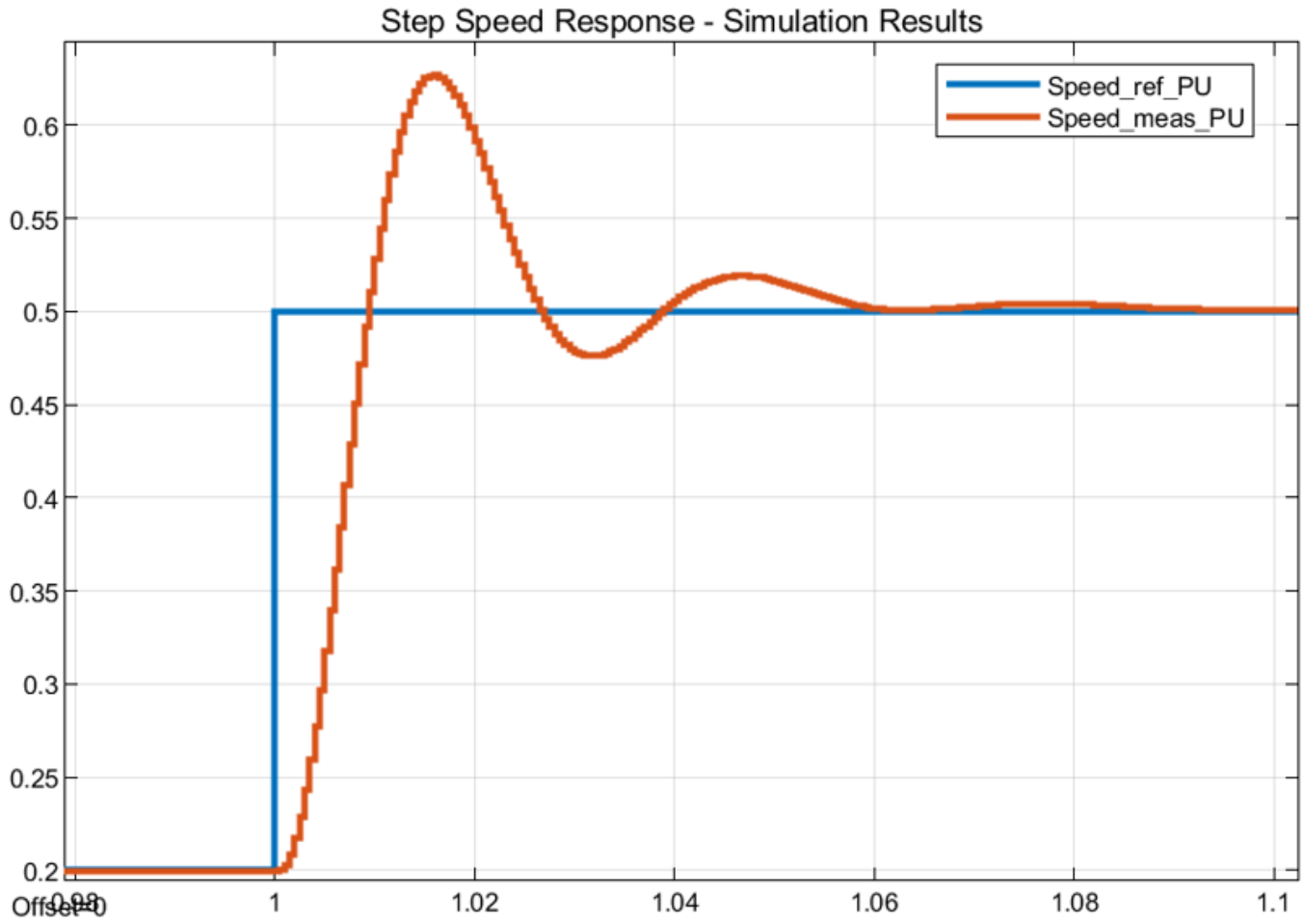
Compare the Speed Controller Response in Simulation and in Target Hardware

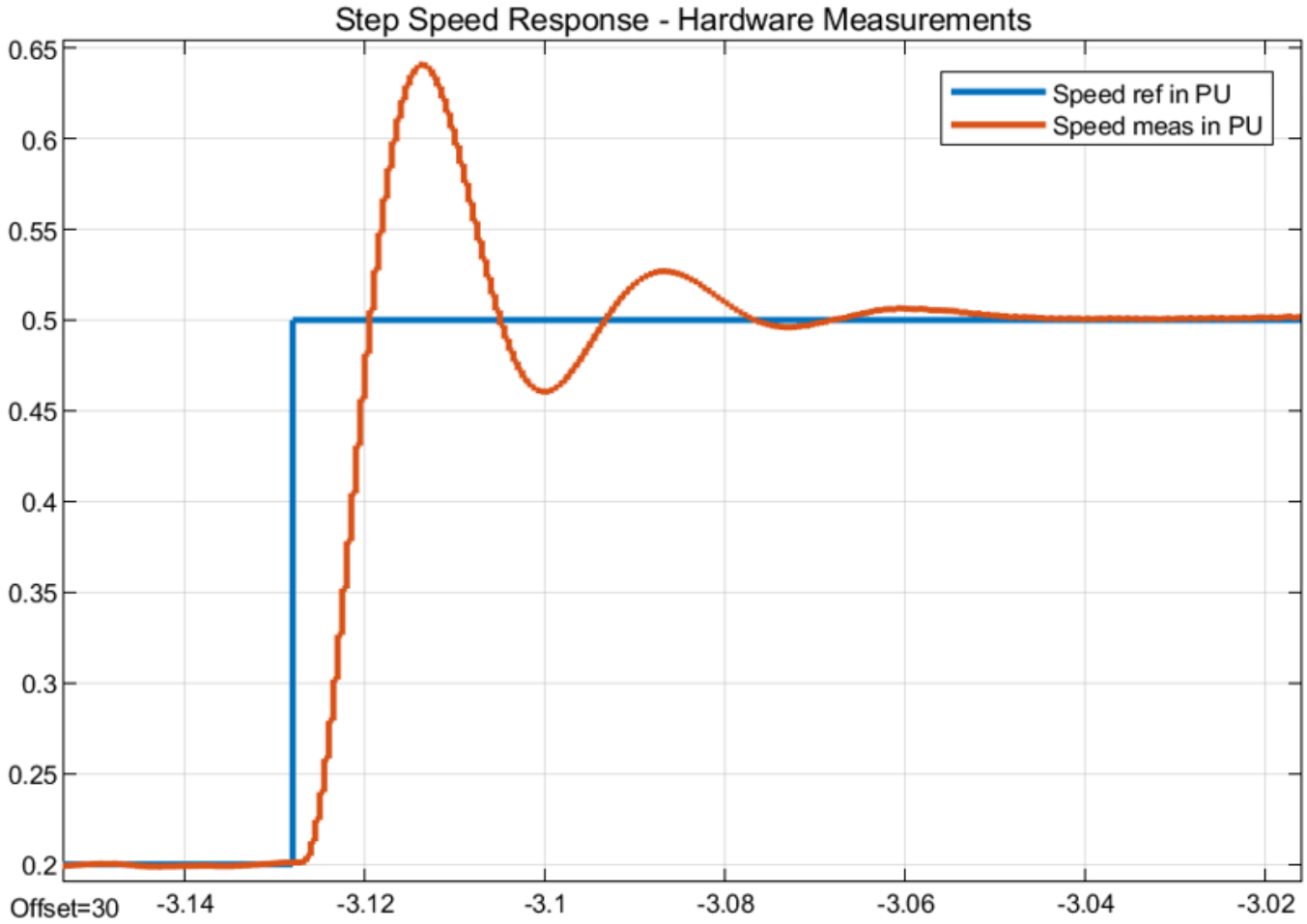
In simulation, give a speed step input and note the speed response. On target hardware, command the speed reference step input and observe the speed feedback. Compare the results step response of simulation and target hardware to know the accuracy of the plant modelling.

1. Simulate the model `mcb_pmsm_operating_mode_f28379d`. Plot the speed reference and speed measured signal. By default, a step input of 0.2 to 0.5 is input in the simulation model.
2. Run the host model and communicate the target hardware.
3. In Select Motor Operating Mode, change the mode from **Stop** to **Speed control**
4. Select the **Speed_ref** in Monitor Signal#1 and **Speed_meas** in Monitor Signal#2 in host model.
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 scope.
7. As shown in below figure, compare the step response of the hardware results with simulation results.

Step Response Analysis for Speed Controller

The step response from simulation is compared with the measurements made from the target hardware. The results may vary depends on the tolerances in plant modelling. Overall, the simulation results are close to measurement values from hardware.


Step Speed Response - Simulation Results
Speed_ref_PU
Speed_meas_PU
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
Offset=0
1
1.02
1.04
1.06
1.08
1.1



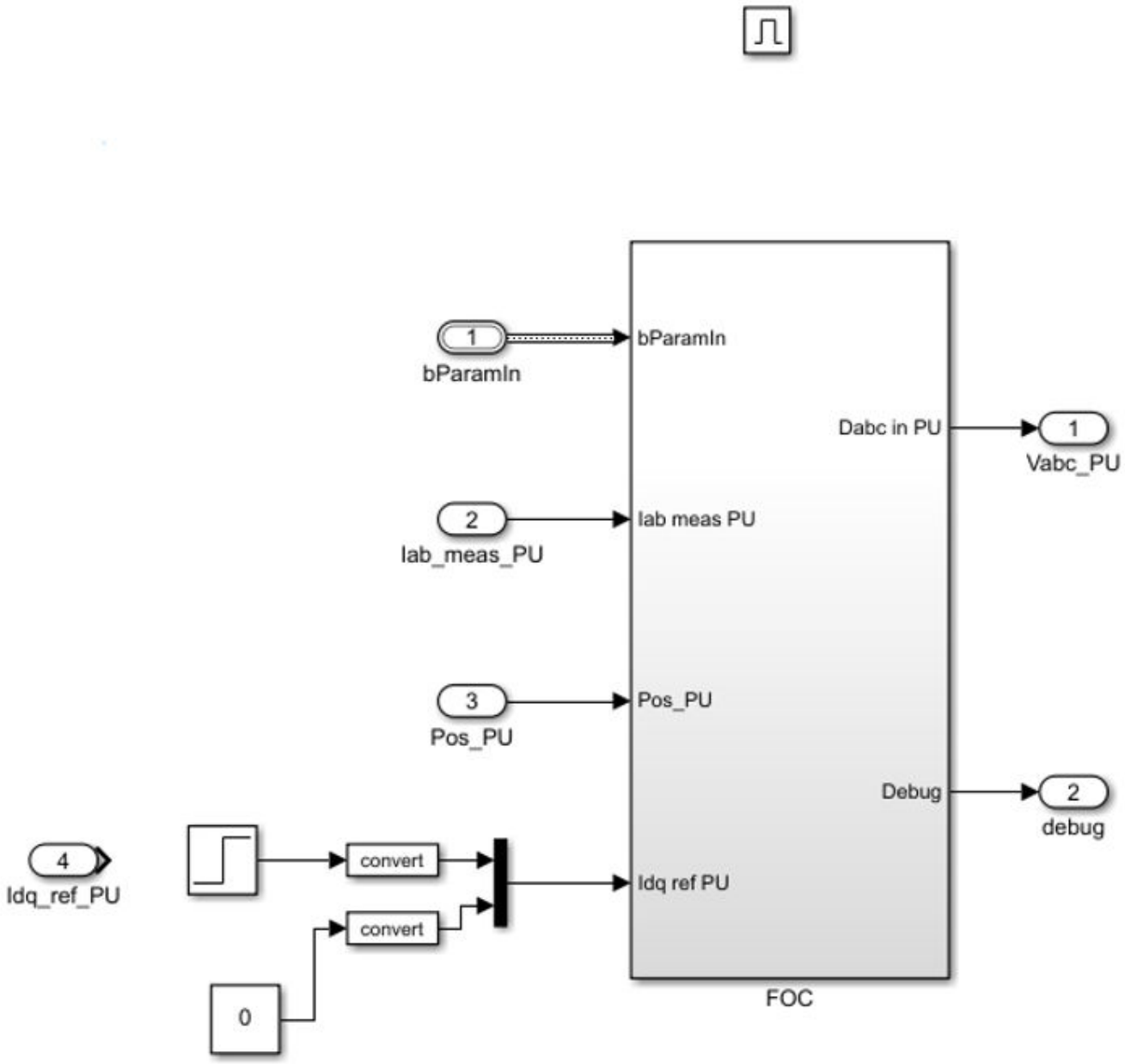
	Peak overshoot (%)	Peak time (ms)	Rise time (ms)	Settling time (ms)
Simulation results	20.13%	16.023	5.561	61.027
Hardware results	22 %	14.324	5.041	51.148

Compare the Current Controller Response in Simulation and in Target Hardware

In simulation, give step current reference and note the current response. This example requires changes to simulate the current reference step input. Refer the below steps for model changes and this applies only for simulation. In target hardware, command the current reference step input and observe the current feedback. Compare the results step response of simulation and target hardware to know the accuracy of the plant modelling.

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

5. Change the `Id_ref` from 0.02 to 0.22 and observe the step change in scope. Ensure that the motor is not spinning. The scope displays the step response for `Id_ref` input.
6. For simulation, two changes are required in the simulation model. In `mcb_pmsm_operating_mode_f28379d/TorqueControl/Control Modes/torque_control` add a step input for d-axis current controller. Choose step input as 0.02 to 0.22 at 1 second. Select Time sample as -1. In Data-type conversion block, select the output datatype as `fixdt(1,32,17)`.

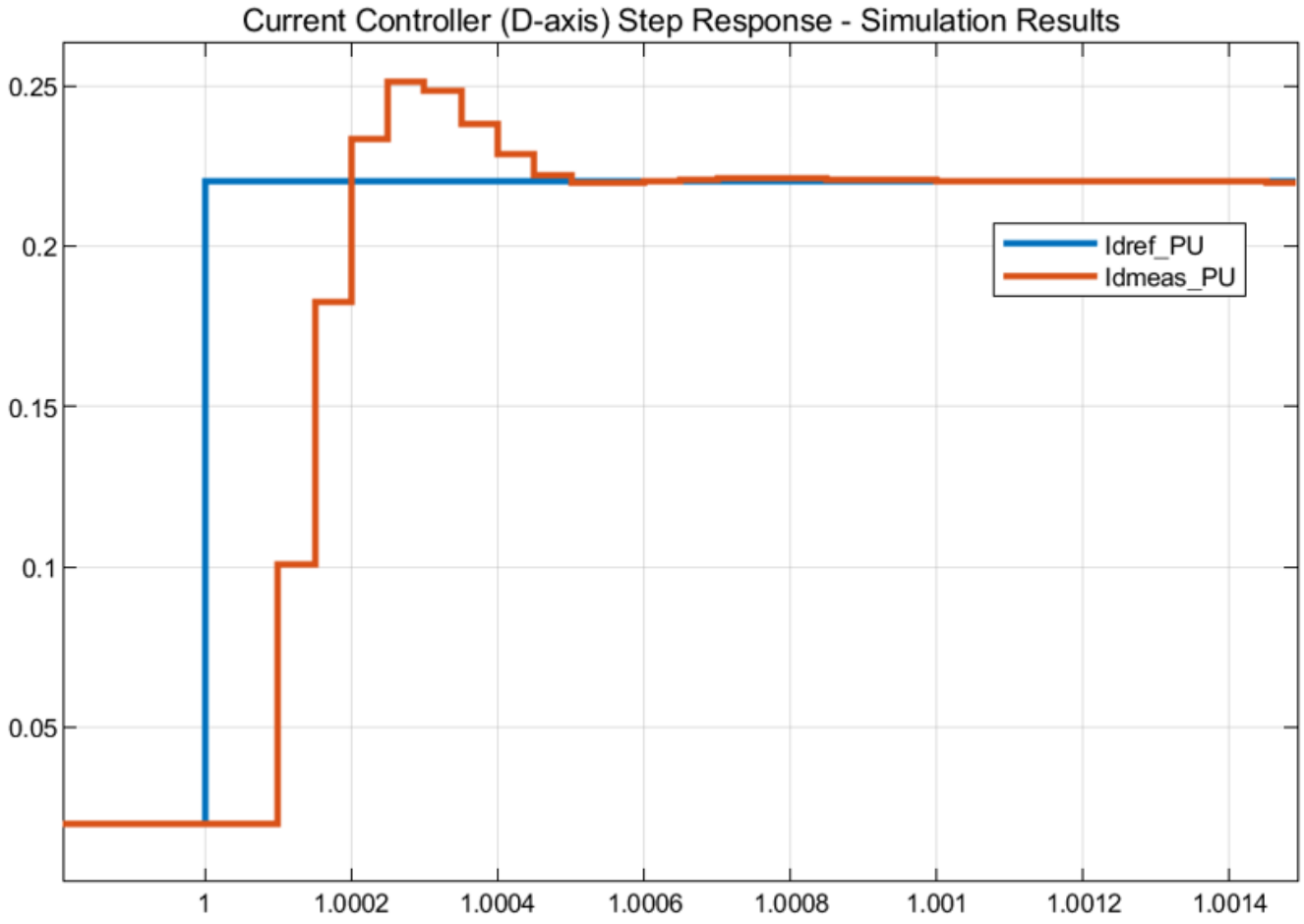


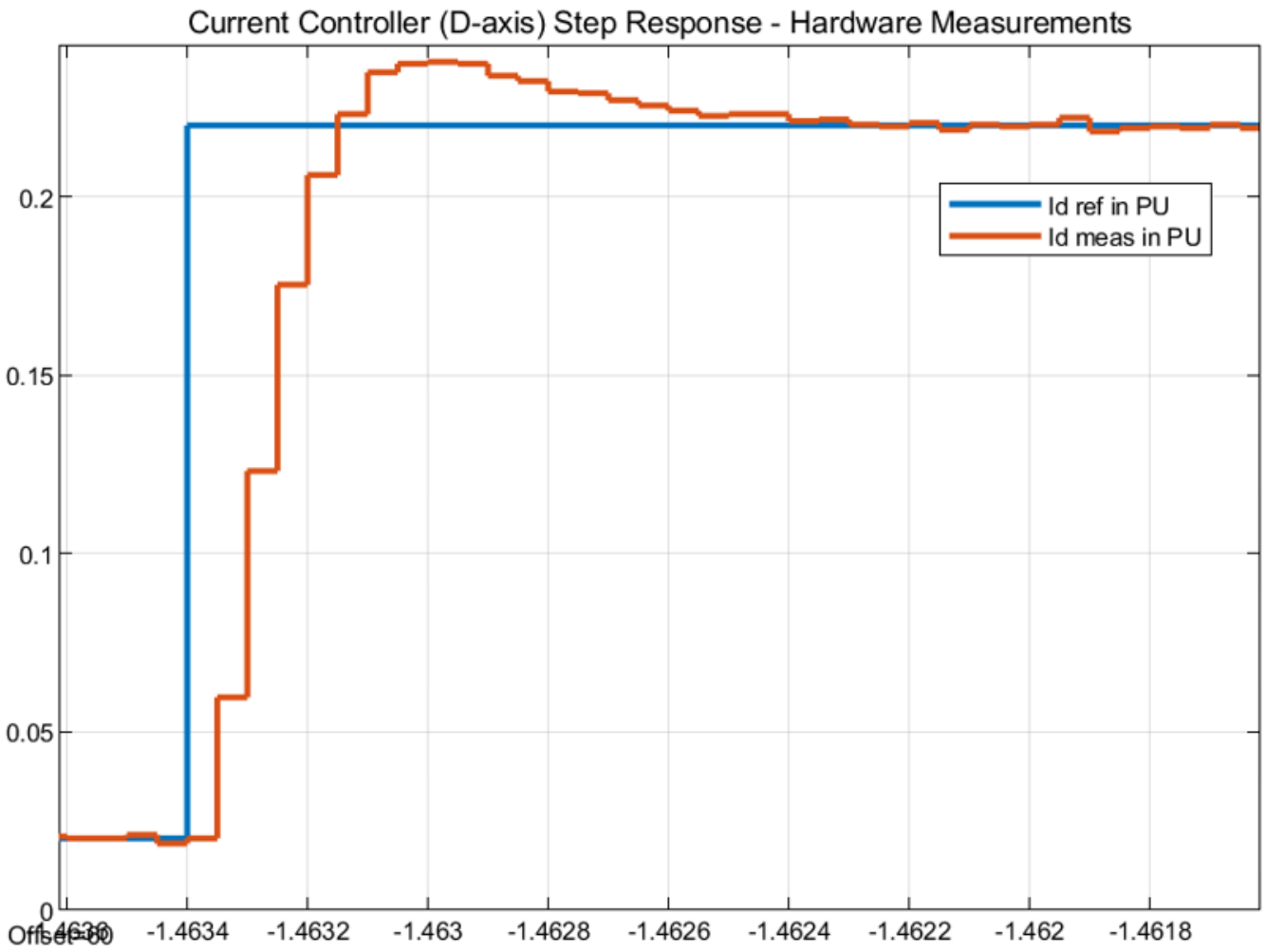
7. In `mcb_pmsm_operating_mode_f28379d/Motor and Inverter/Plant Model` (sim), PMSM motor block, change the Mechanical Input Configuration to **Speed** and input 0 to Spd (input port).
8. Run the simulation and measure the `Idref_PU` and `Idmeas_PU` in Simulink Data Inspector.

9 As shown in the below figure, compare the step response of the hardware results with simulation results.

Step response analysis for D-axis current controller

The scope image from simulation is compared with the measurements made from the target hardware. The results may vary depending on the tolerances in plant modelling. With an accurate plant model, the simulation results get closer to the measured results from 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

Tip If simulation results differs a lot with the hardware measurements, verify the delay and scaling factor in the plant model.

Note For q-axis current controller, align motor to the d-axis and mechanically lock the rotor. Follow the same steps as D-axis current controller for comparative analysis. External mechanical locking can be achieved through mechanical braking system or coupled with a dyno motor running in speed control.

The accuracy of the plant modelling improves the accuracy of simulation and match the test hardware results.

Plant Modeling

Creating Plant Model Using Motor Control Blockset

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

The creation of plant model using Motor Control Blockset includes the modelling of the following 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)

The functionality of the plant model that you create can be verified by performing these tasks:

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 `mcb_pmsm_foc_qep_f28379d`.

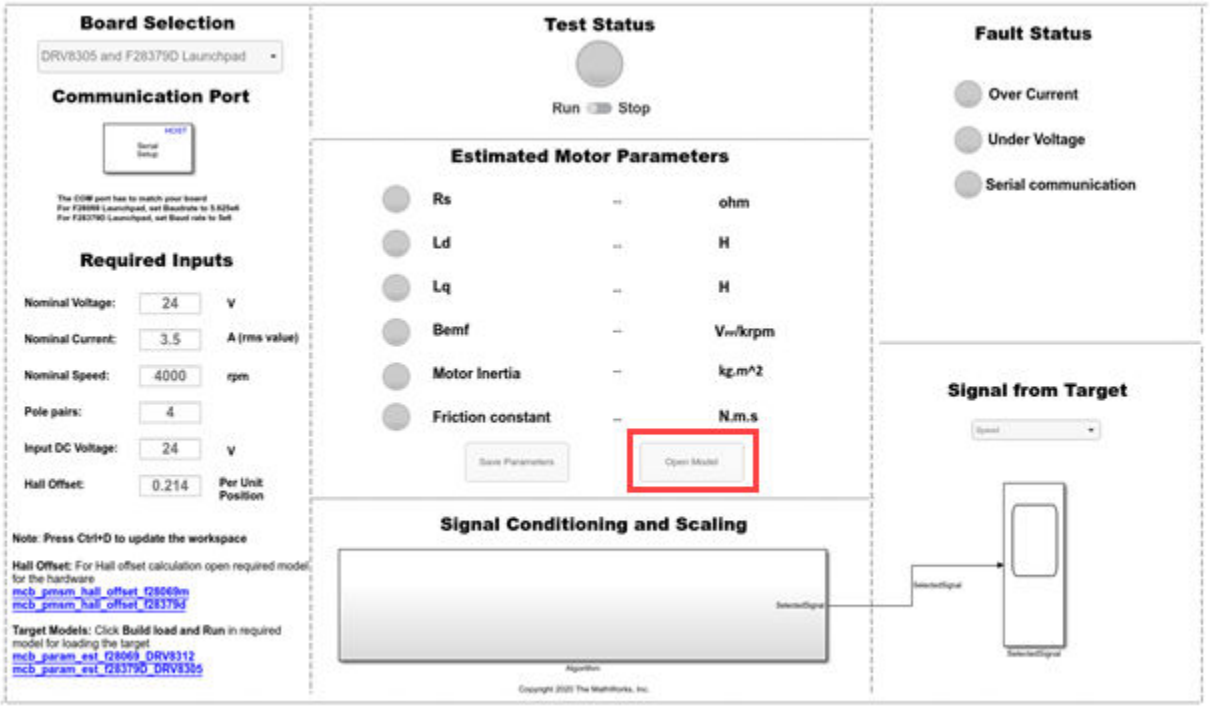
1. “Create a Model with PMSM Block and Use Motor Parameters” on page 3-3
2. “Add Average-value Inverter Block” on page 3-5
3. “Create Motor Phase Current Sensing and Signal Conditioning Subsystem” on page 3-6
4. “Create Position Sensing Subsystem” on page 3-7
5. “Add Delay in Plant Model” on page 3-8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

Create a Model with PMSM Block and Use Motor Parameters

In Motor Control Blockset, there are two methods to create a model with PMSM motor block:

- Perform the Parameter Estimation operation using Motor Control Blockset and open the Simulink model with PMSM motor block:

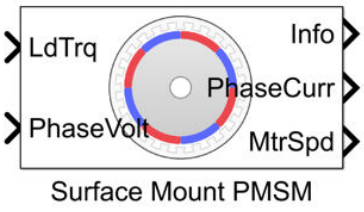
  Motor Parameter Estimation workflow in Motor Control Blockset helps you to find the motor parameters by performing the series of tests on the motor. For details, refer to “Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool”. After a successful motor parameter estimation, click **Open Model** in the parameter estimation host model. A new model opens with the Interior PMSM block (from the Simulink Library of Motor Control Blockset) along with the estimated motor parameters.

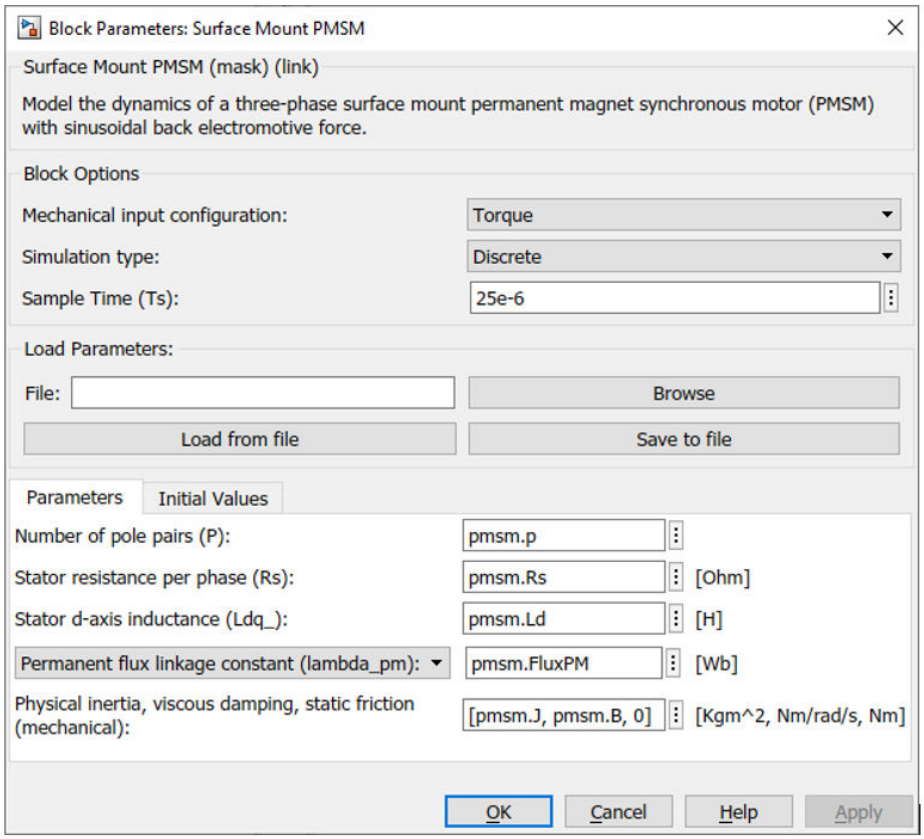


- Create a new model and add PMSM motor block from Motor Control Blockset library:

  Create a new Simulink model and add the Surface Mount PMSM block from the Motor Control Blockset library in Simulink library browser. Open the block mask and enter the motor parameters. The motor parameters can be obtained from:

  - Parameter Estimation workflow in Motor Control Blockset (for details, refer to “Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool”)
  - From the datasheet of the motor or from known sources





In Surface Mount PMSM block, choose **Simulation type** as `Discrete` and enter the value of **Sample Time (Ts)** as `25e-6` (half of the control frequency). Discrete simulation improves the simulation speed.

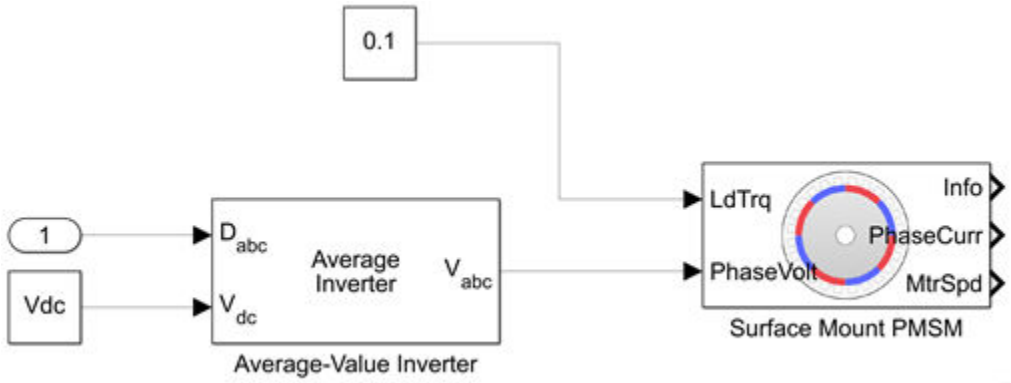
If parameters are available in mat-file format, click **Browse** in the Surface Mount PMSM block mask to locate the mat-file and then click **Load from file** to load the parameter.

The default motor parameter files are available in the location. `<matlabroot>\toolbox\autoblks\autoblksshared\mcbtemplates` for your reference.

In the Surface Mount PMSM block mask, motor parameters can also be represented as workspace variables and update these workspace variables using m-script in the model initialization callback. Refer to the file `mcb_SetPMSMMotorParameter.m` for the reference motor parameters for some of the commercially available motors (for details about this script, see “Estimate Control Gains from Motor Parameters”)

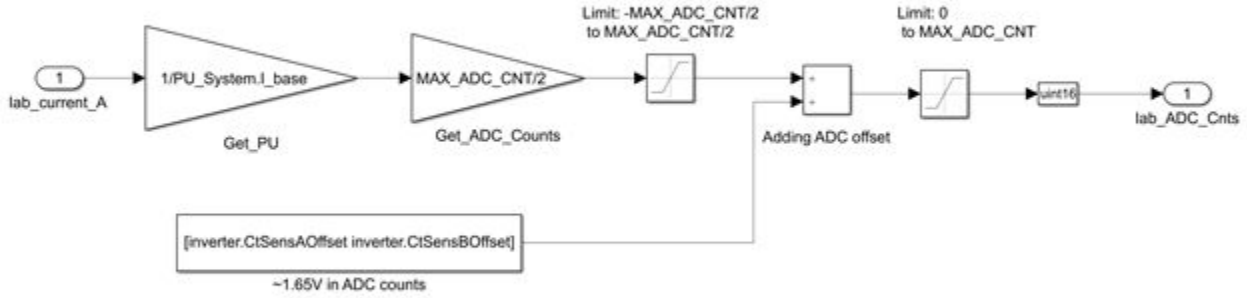
Add Average-value Inverter Block

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



Create Motor Phase Current Sensing and Signal Conditioning Subsystem

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



The maximum measurable peak current is considered as Base current and this corresponds to the full-scale ADC values from offset.

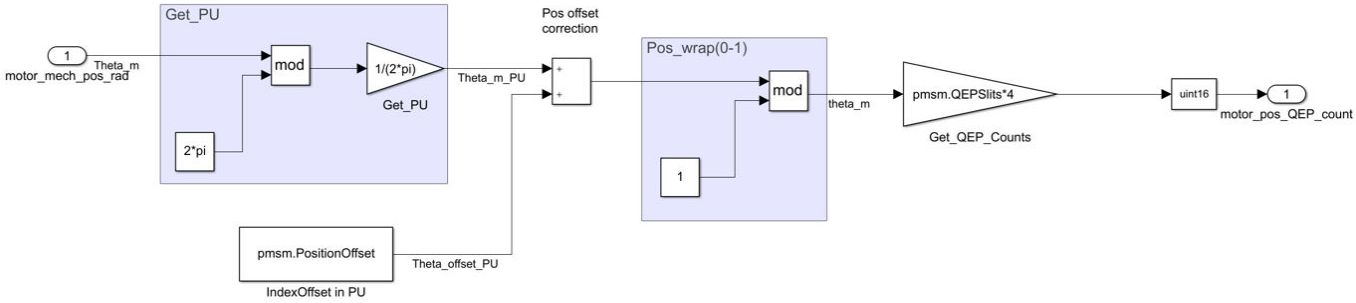
ADC counts = ((ADC counts full scale/2) / Base current or max measurable current in A) + ADC offset.

Refer to the file `mcb_SetInverterParameters.m` for default inverter and signal conditioning circuit parameters for commercially available inverters. For any other configurations, create an inverter type in `mcb_SetInverterParameters.m` and use this in the model initialization script for parameter initialization. If low-pass filters involved in the current measurement, add an average model for filtering the current.

Create Position Sensing Subsystem

The Position Sensing Subsystem reads motor position from the Surface Mount PMSM block and simulates QEP encoder pulse counts. The Surface Mount PMSM block outputs mechanical position in rad/s.

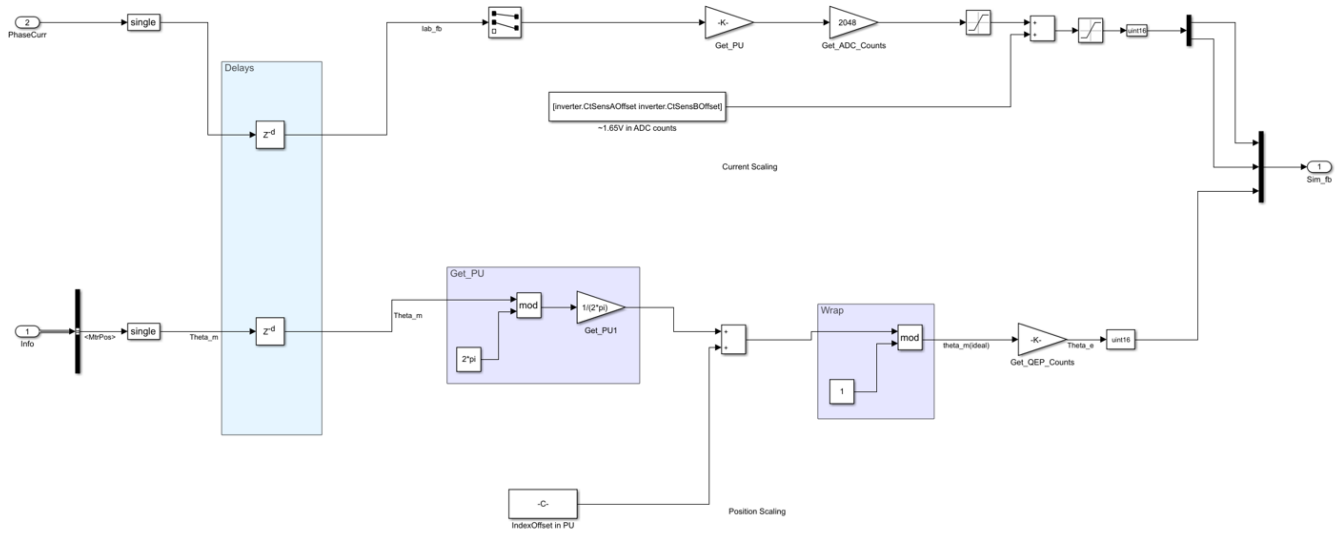
Convert the position in (0 to 2π) rad/s to QEP encoder counts as shown in this figure.



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



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

Integrate the Blocks and Subsystems

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