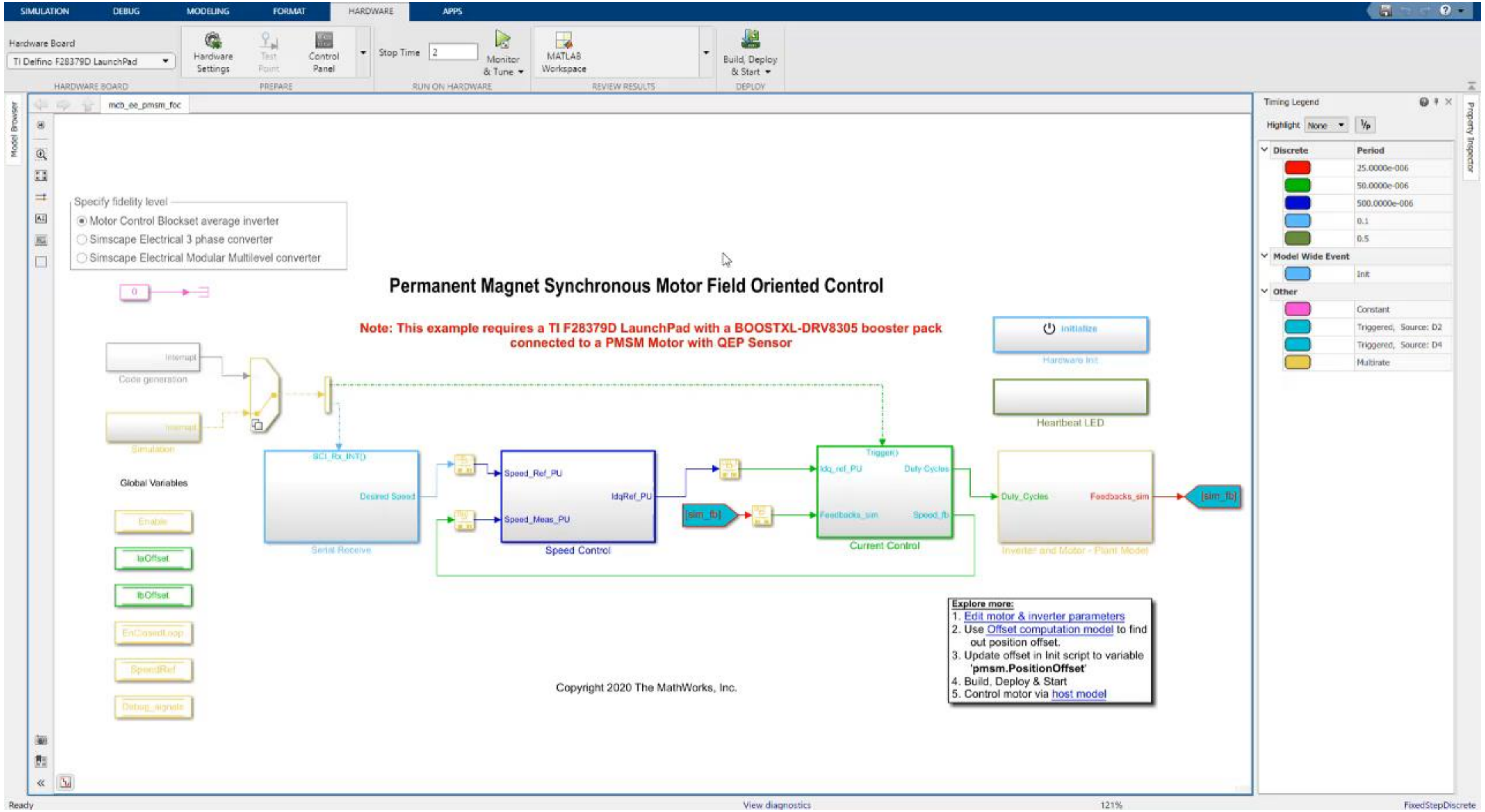
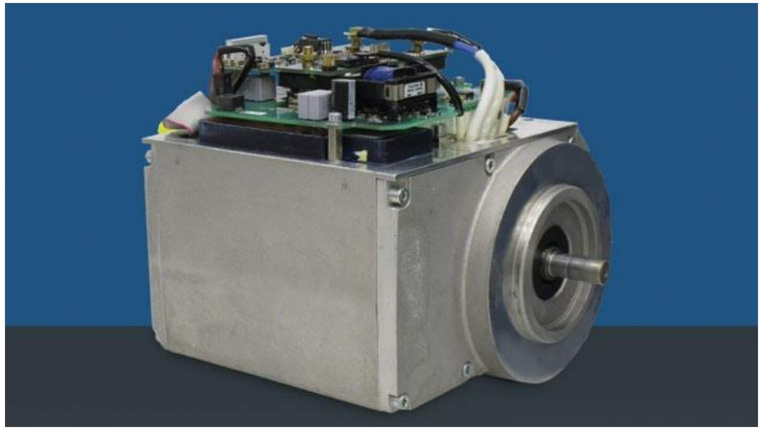
Simulink를 이용한 브러시리스 모터 제어 개발

강효석, MathWorks Korea

Spinning a Brushless Motor Using Simulink and Model-Based Design



Brushless Motors Are Everywhere



Developing Embedded Motor Control Software Has Its Challenges

ITK Engineering Develops IEC 62304– Compliant Controller for Dental Drill Motor with Model-Based Design

Challenge

Develop and implement field-oriented controller software for sensorless brushless DC motors for use in dental drills

Solution

Use Model-Based Design with Simulink, Stateflow, and Embedded Coder to model the controller and plant, run closed-loop simulations, generate production code, and streamline unit testing

Results

- **Development time halved**
- Hardware problems discovered early
- Contract won, client confidence established

Dental drills featuring ITK Engineering's sensorless brushless motor control.

"Model-Based Design with Simulink enabled us to reduce costs and project risk through early verification, shorten time to market on an IEC 62304–certified system, and deliver high-quality production code that was first-time right."

- Michael Schwarz, ITK Engineering

Developing Embedded Motor Control Software Has Its Challenges

- Design work needed to be started **before motor hardware** was available and needed extensive testing to comply with standards

- Team needed to **rapidly implement control software** on embedded processor once more hardware became available

- Complex algorithms running at **high sample rates** were difficult to implement in short amount of time

ITK Engineering Develops IEC 62304– Compliant Controller for Dental Drill Motor with Model-Based Design

Dental drills featuring ITK Engineering's sensorless brushless motor control.

Challenge

Develop and implement field-oriented controller software for sensorless brushless DC motors for use in dental drills

Solution

Use Model-Based Design with Simulink, Stateflow, and Embedded Coder to model the controller and plant, run closed-loop simulations, generate production code, and streamline unit testing

Results

- **Development time halved**
- Hardware problems discovered early
- Contract won, client confidence established

"Model-Based Design with Simulink enabled us to reduce costs and project risk through early verification, shorten time to market on an IEC 62304–certified system, and deliver high-quality production code that was first-time right."

- Michael Schwarz, ITK Engineering

Why Simulink for Motor Control?

- Verify control algorithm with desktop simulation

- Generate compact and fast code from models

- Minimize development time using reference examples

ITK Engineering Develops IEC 62304– Compliant Controller for Dental Drill Motor with Model-Based Design

Challenge

Develop and implement field-oriented controller software for sensorless brushless DC motors for use in dental drills

Solution

Use Model-Based Design with Simulink, Stateflow, and Embedded Coder to model the controller and plant, run closed-loop simulations, generate production code, and streamline unit testing

Results

- **Development time halved**
- Hardware problems discovered early
- Contract won, client confidence established

Dental drills featuring ITK Engineering's sensorless brushless motor control.

"Model-Based Design with Simulink enabled us to reduce costs and project risk through early verification, shorten time to market on an IEC 62304–certified system, and deliver high-quality production code that was first-time right."

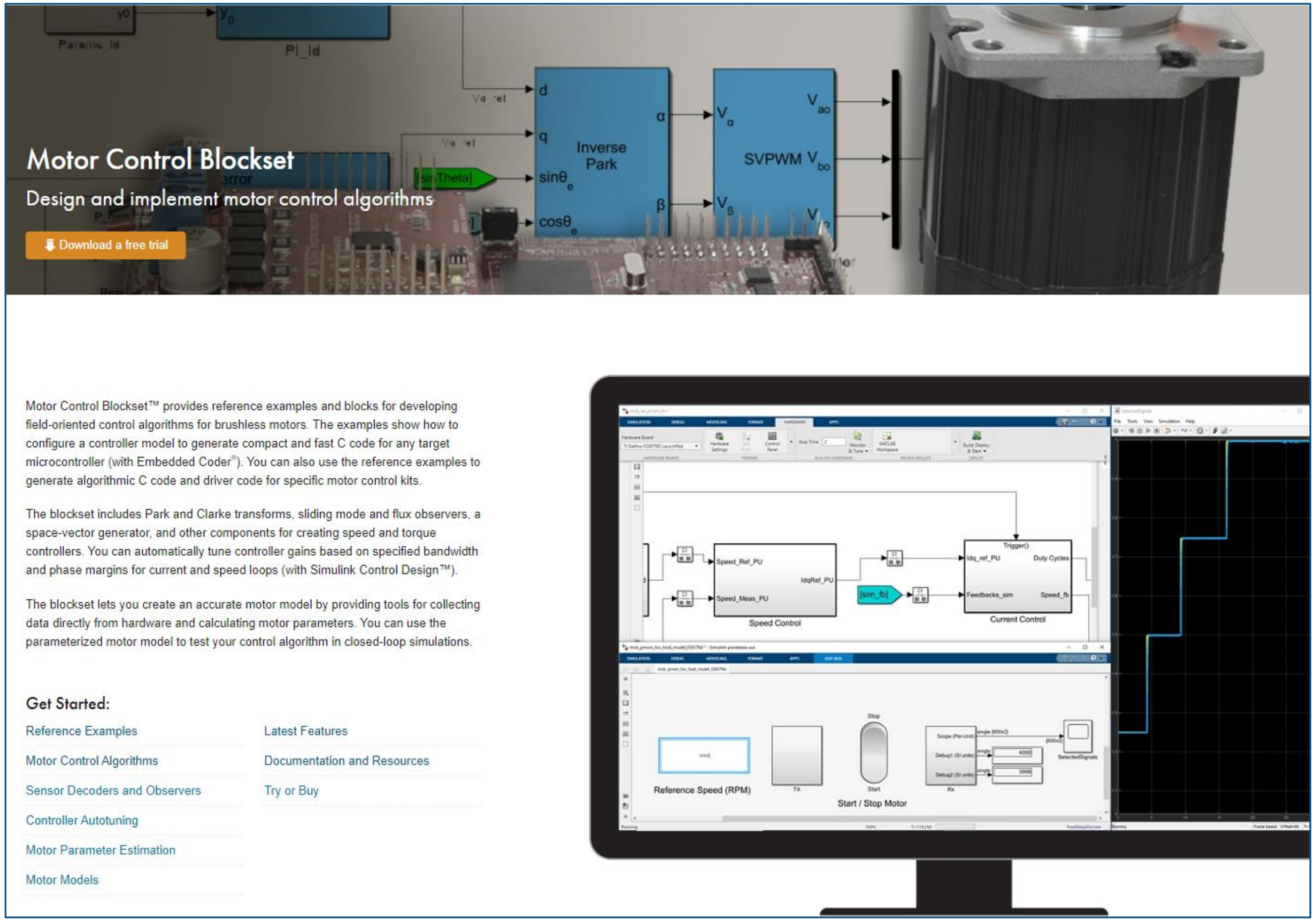
- Michael Schwarz, ITK Engineering

Customers routinely report 50% faster time to market

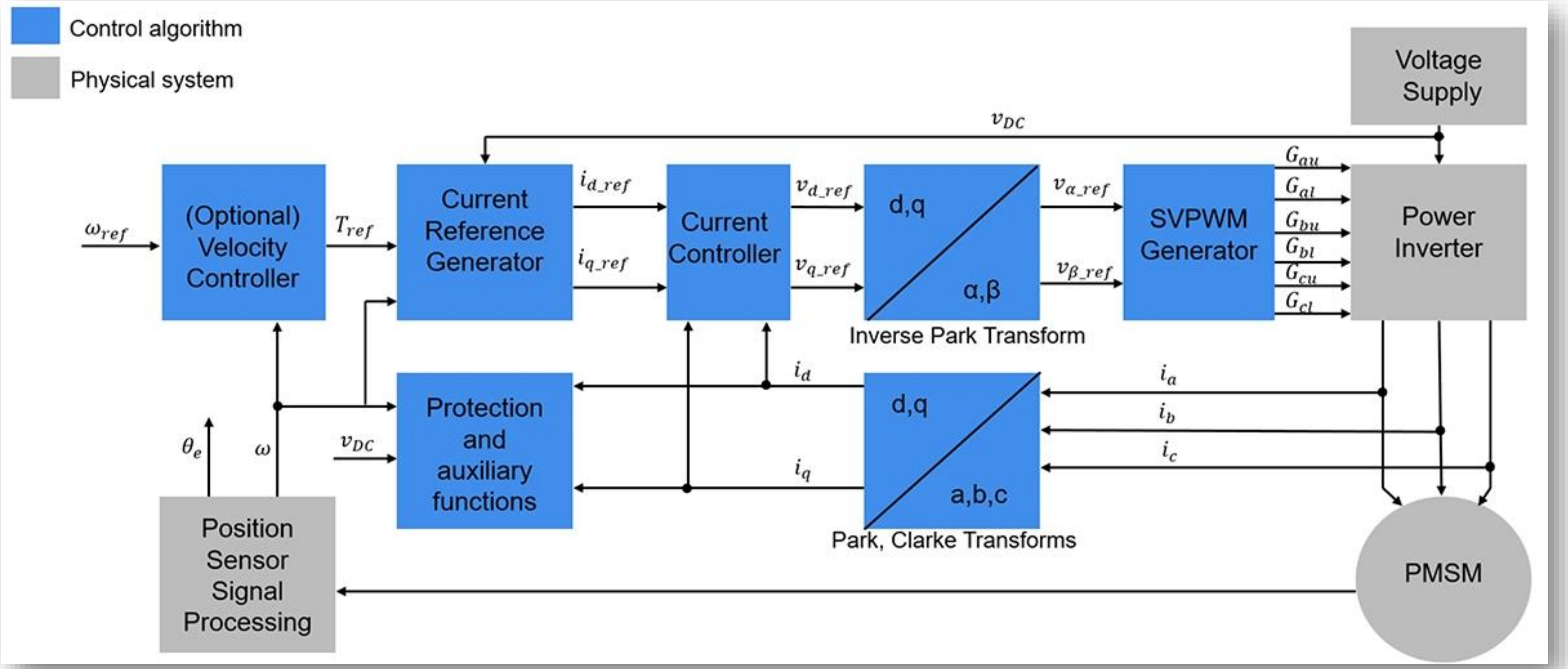
Motor Control Blockset Simplifies the Workflow

R2020a

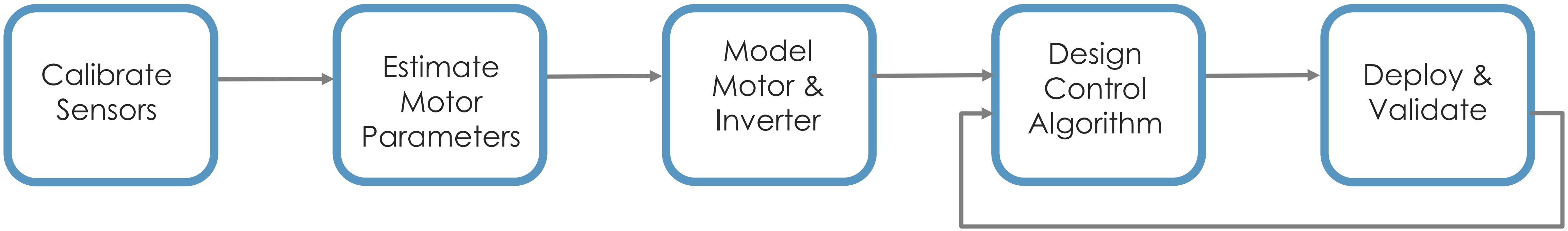
- Control blocks optimized for code generation
- Sensor decoders and observers
- Motor parameter estimation
- Controller autotuning
- Reference examples



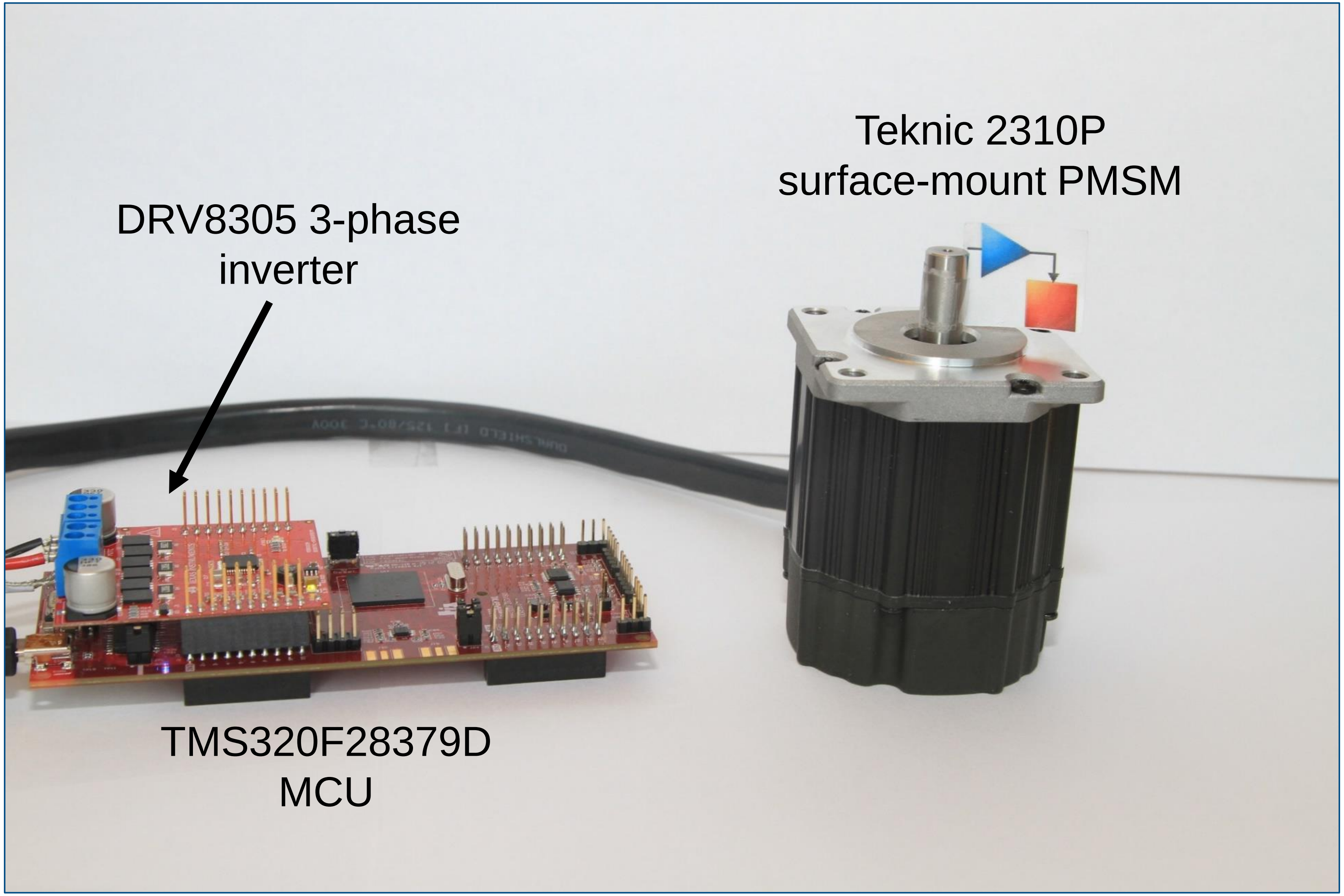
Field-Oriented Control(FOC) for Brushless Motors



Workflow for Implementing Field-Oriented Control



We Will Use Texas Instruments Motor Control Kit

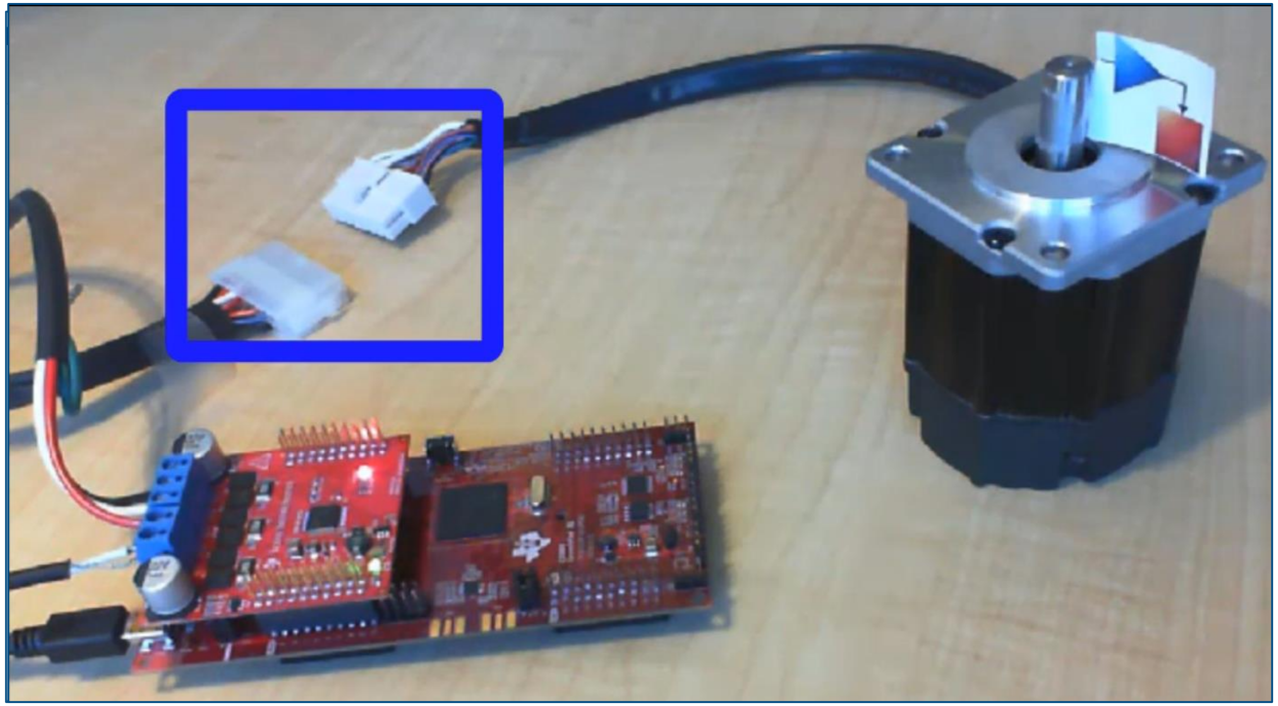


Sensor Calibration

Calibrate Sensors → Estimate Motor Parameters → Model Motor & Inverter → Design Control Algorithm → Deploy & Validate

- Calibrate ADC offsets

- Calibrate position sensor offset



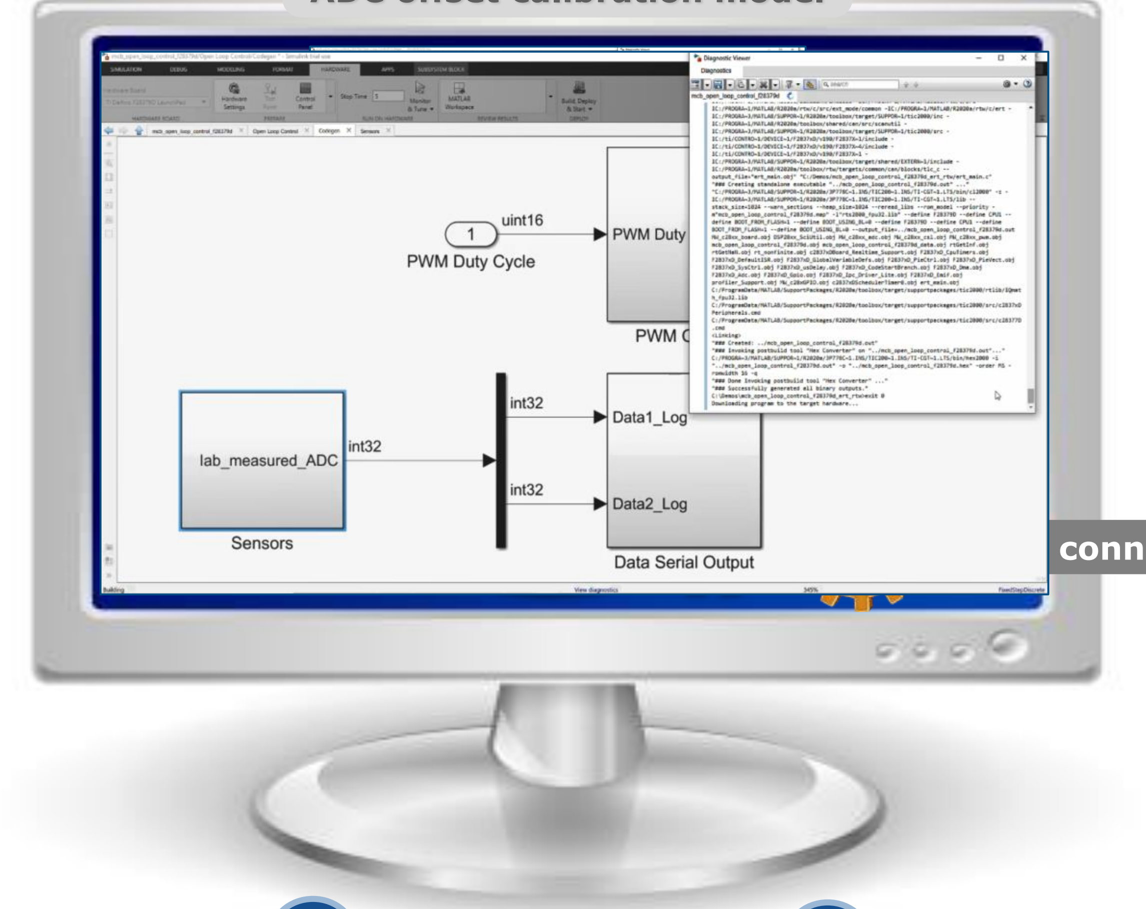
Sensor Calibration

ADC Offsets Calibration

Calibrate Sensors → Estimate Motor Parameters → Model Motor & Inverter → Design Control Algorithm → Deploy & Validate

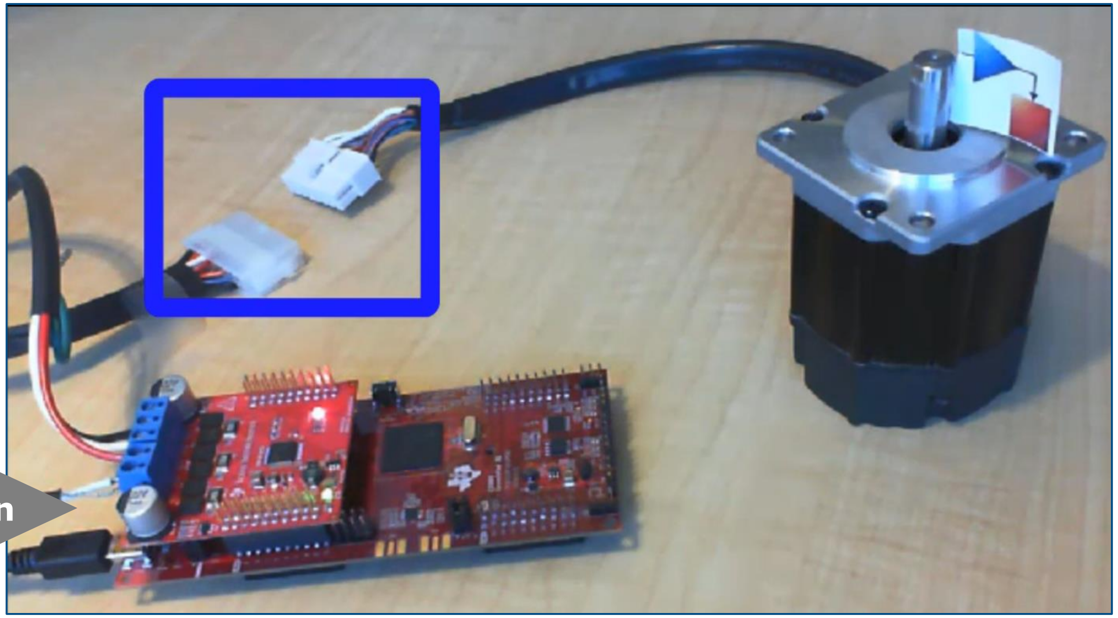
Development Computer

ADC offset calibration model



connection

Target Hardware



1 Code Generation   2 Compile and Link   3 Download and Ready to Run

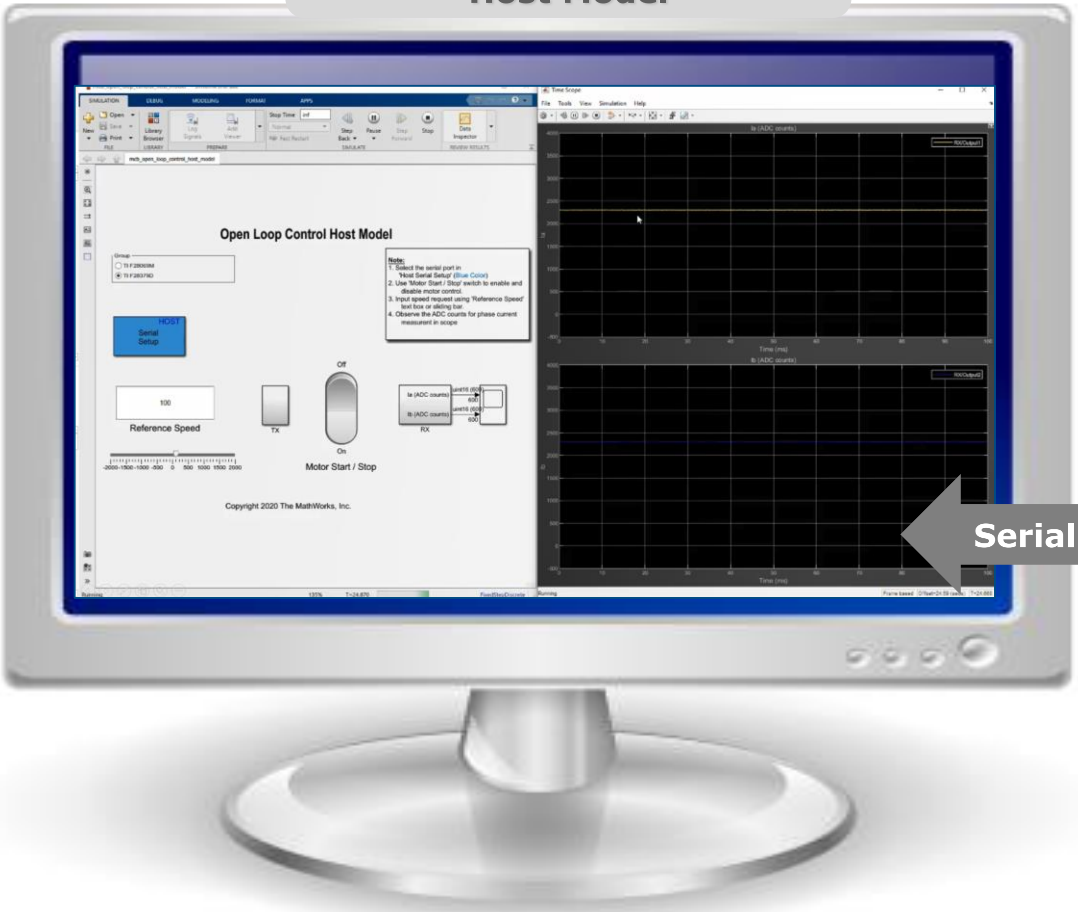
Sensor Calibration

ADC Offsets Calibration

Calibrate Sensors → Estimate Motor Parameters → Model Motor & Inverter → Design Control Algorithm → Deploy & Validate

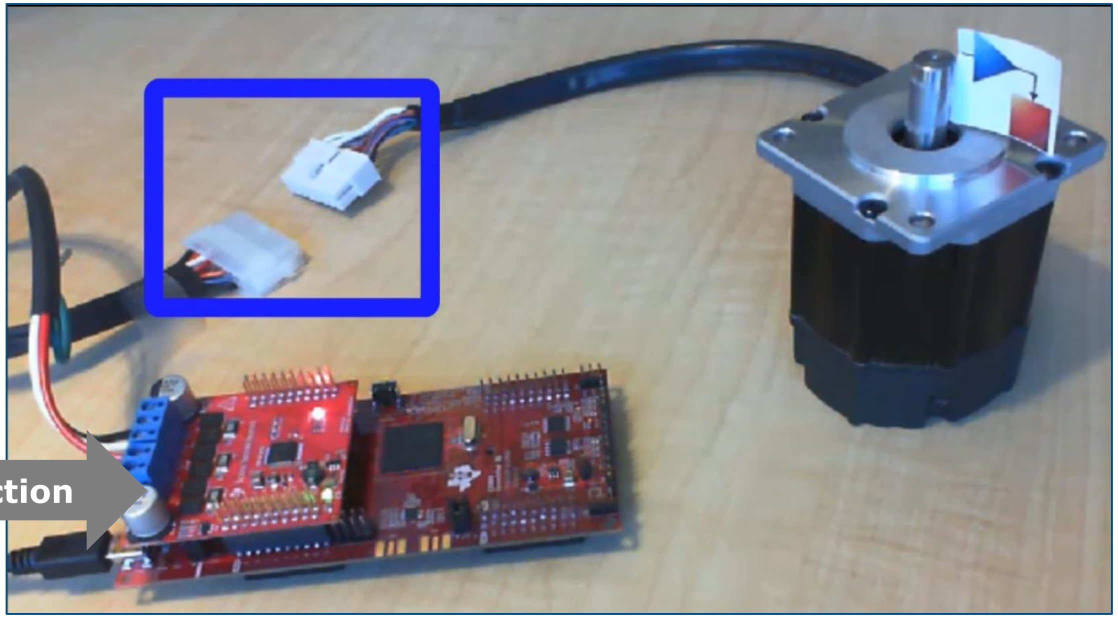
Development Computer

Host Model



Serial connection

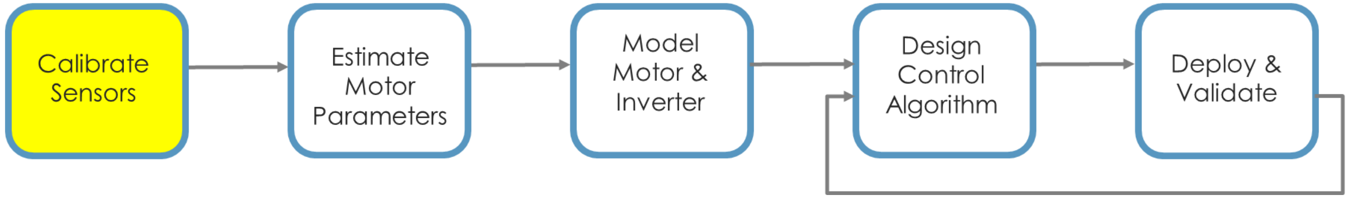
Target Hardware



4 External mode Simulation

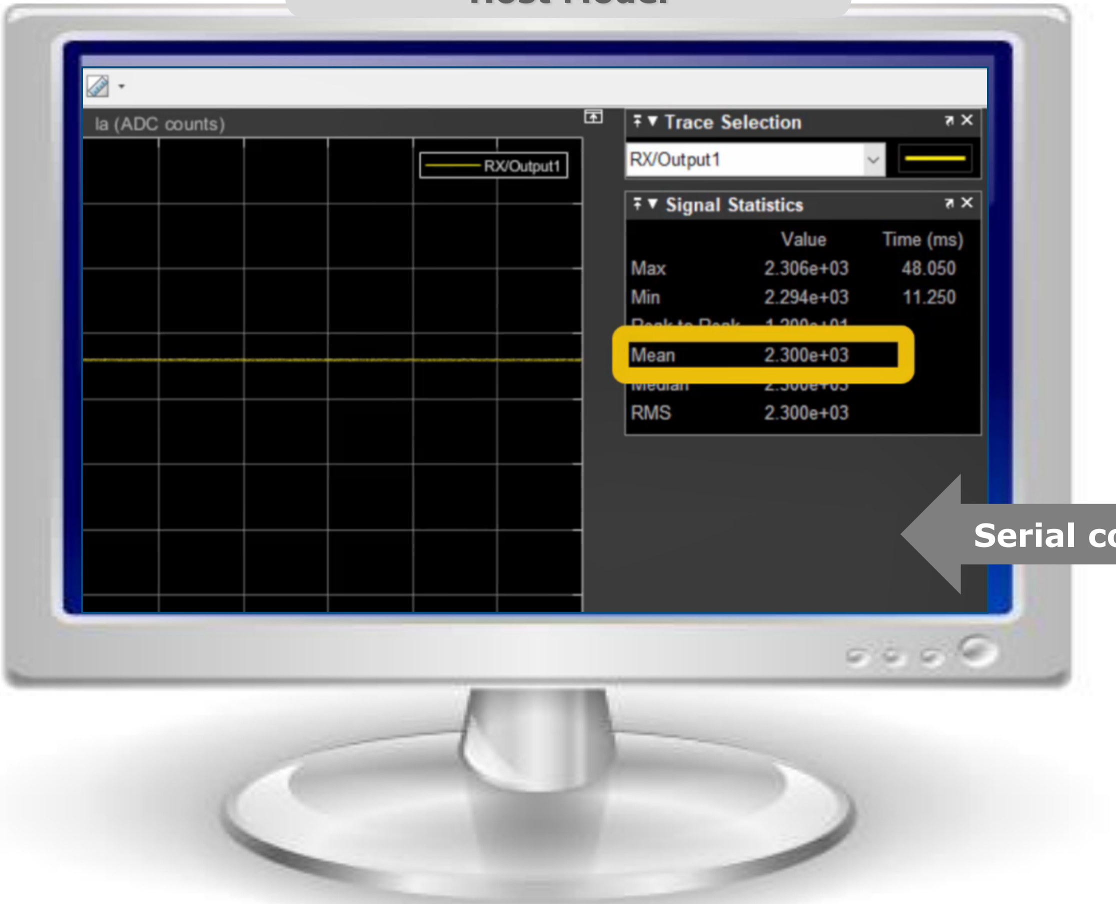
Sensor Calibration

ADC Offsets Calibration



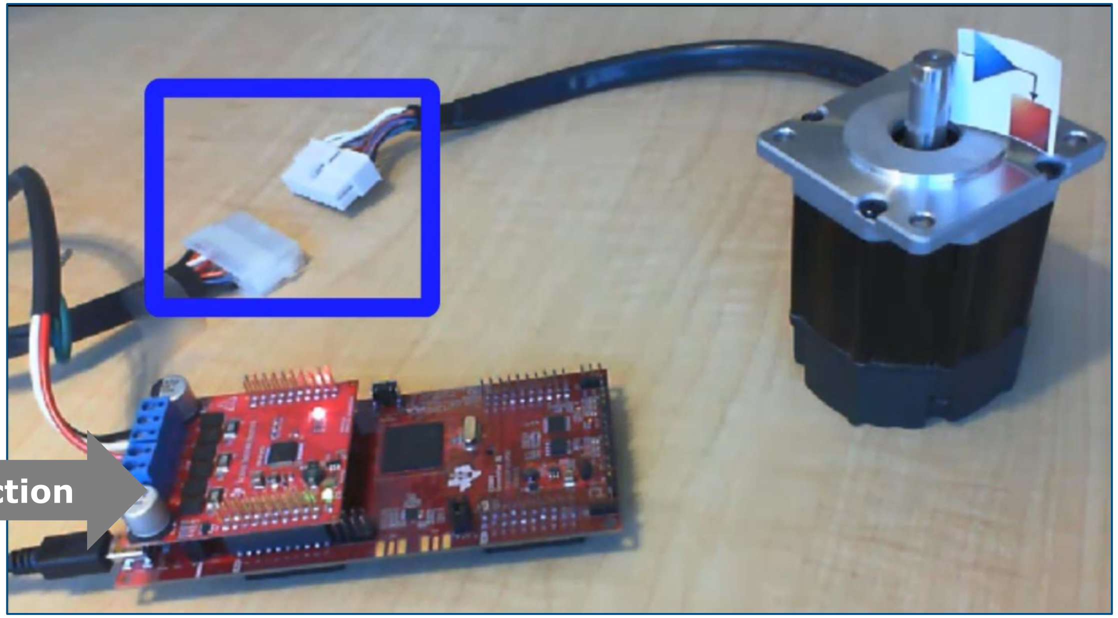
Development Computer

Host Model



Serial connection

Target Hardware

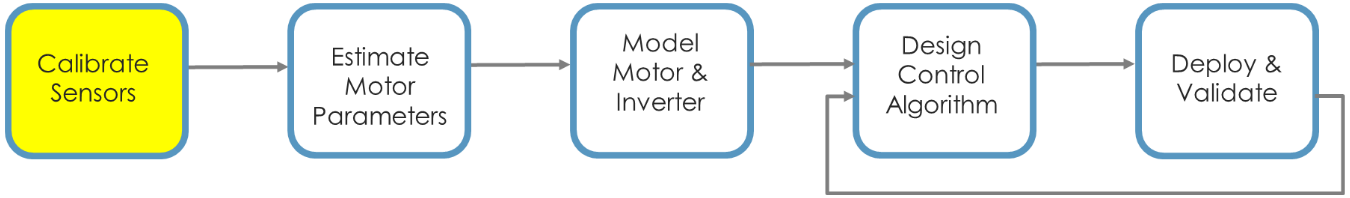


4 External mode Simulation

5 Get offset for phase A and B

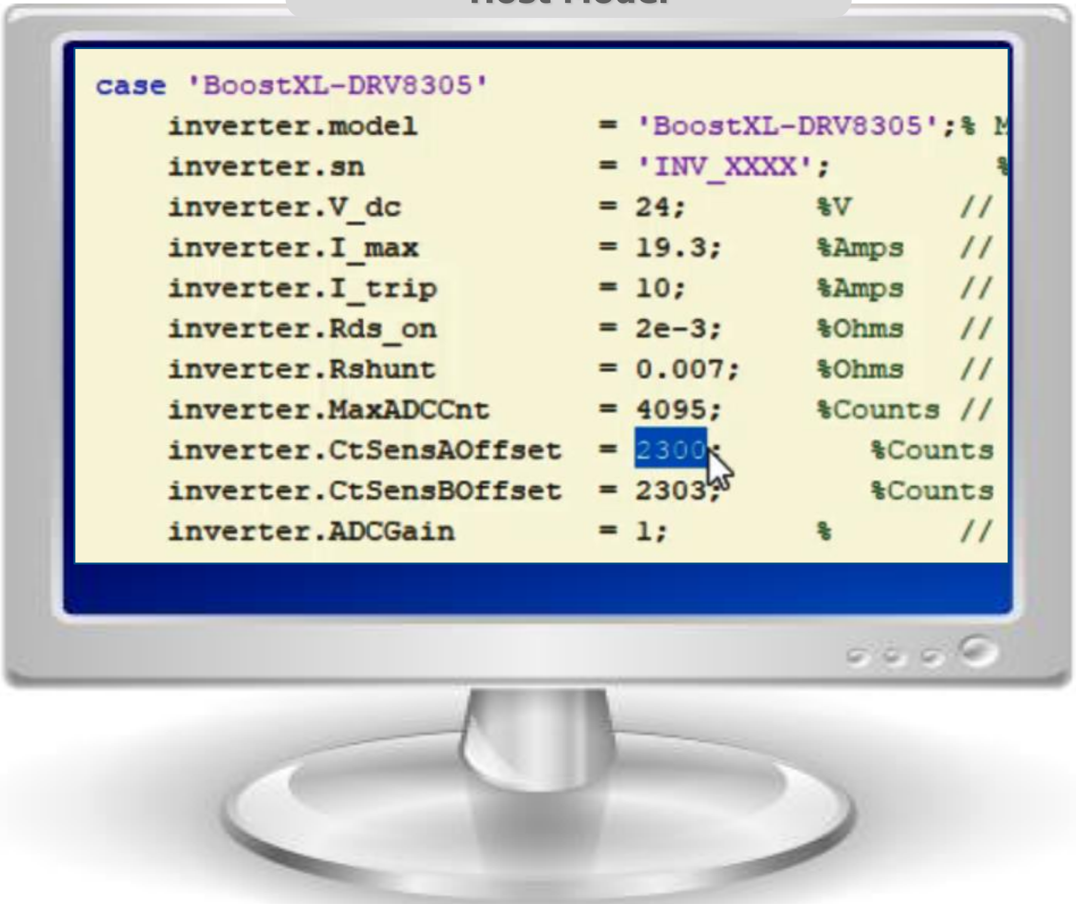
Sensor Calibration

ADC Offsets Calibration

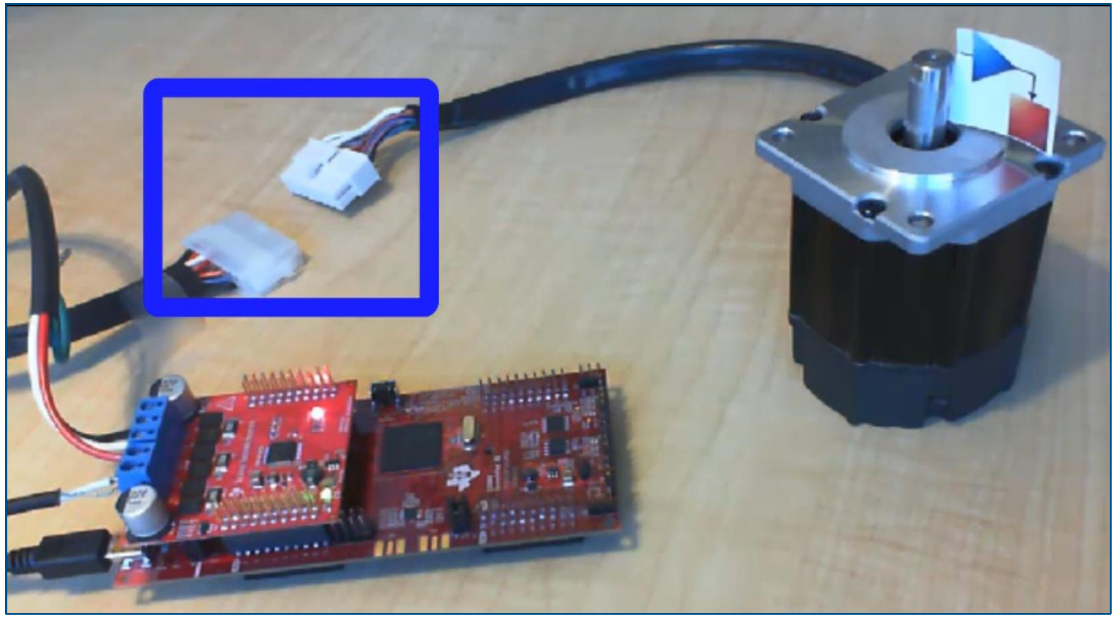


Development Computer

Host Model



Target Hardware

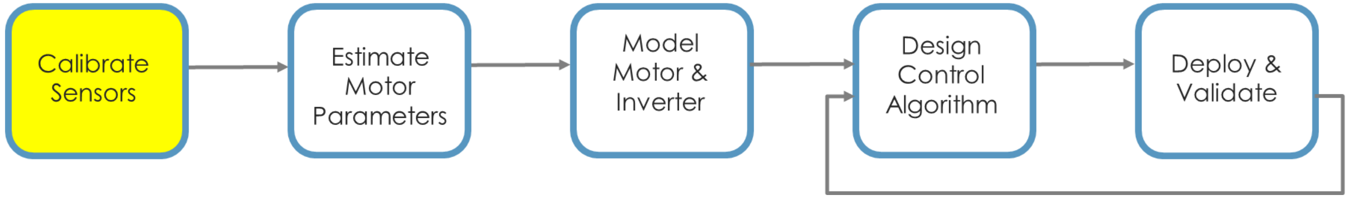


4 External mode Simulation

5 Get offset for phase A and B

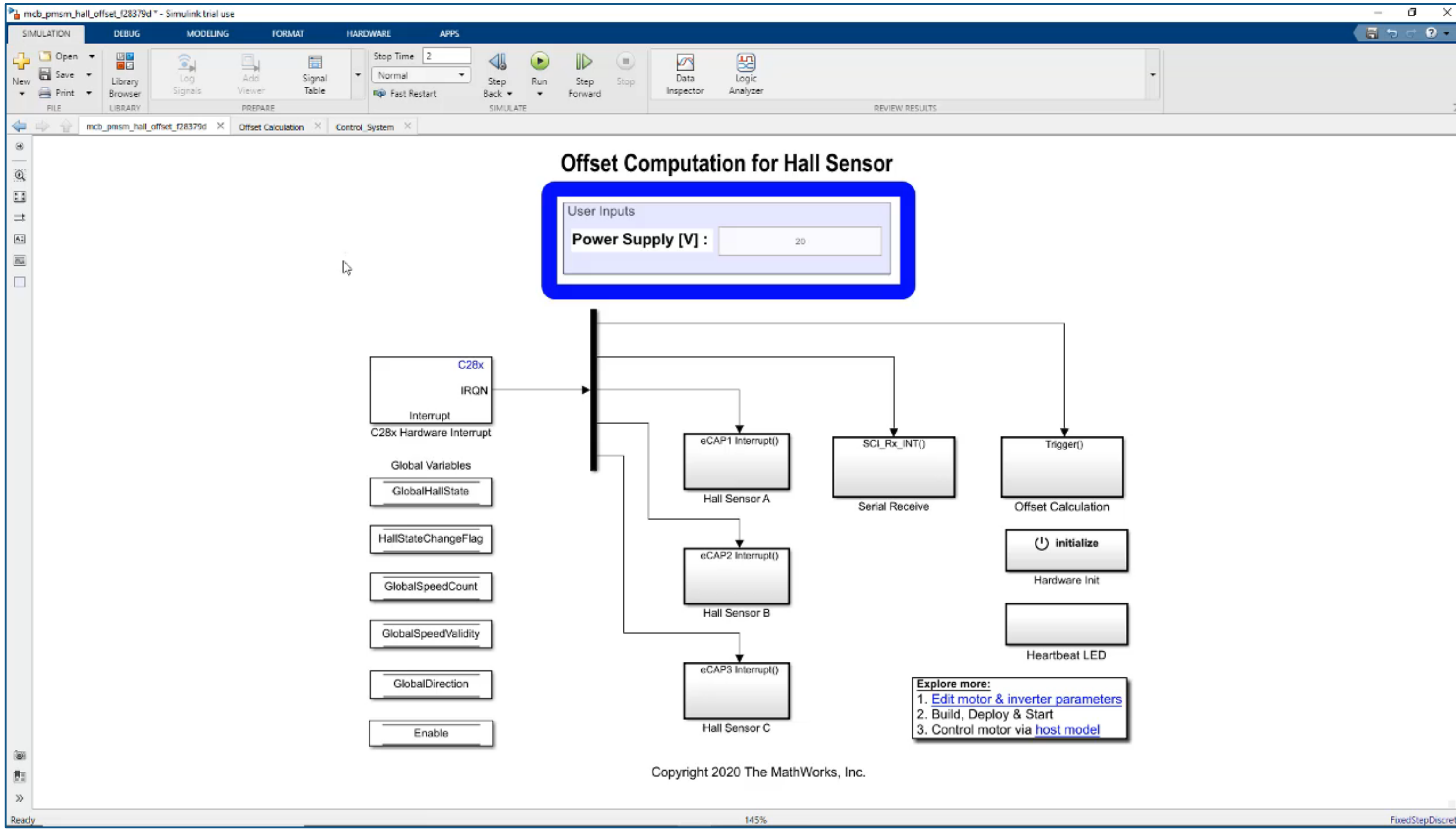
6 Enter these offsets into a setup script

Sensor Calibration

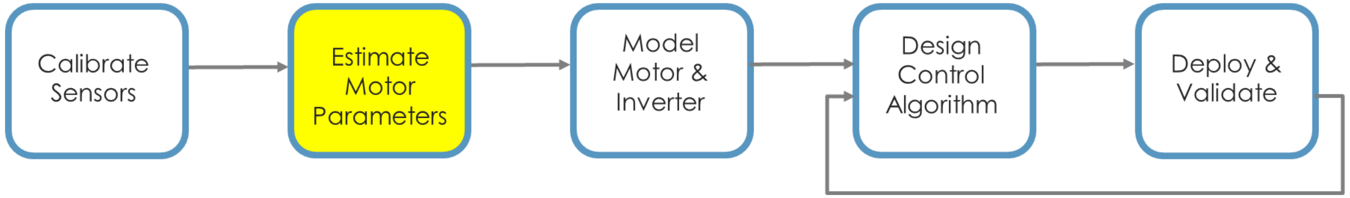


- Calibrate ADC offsets

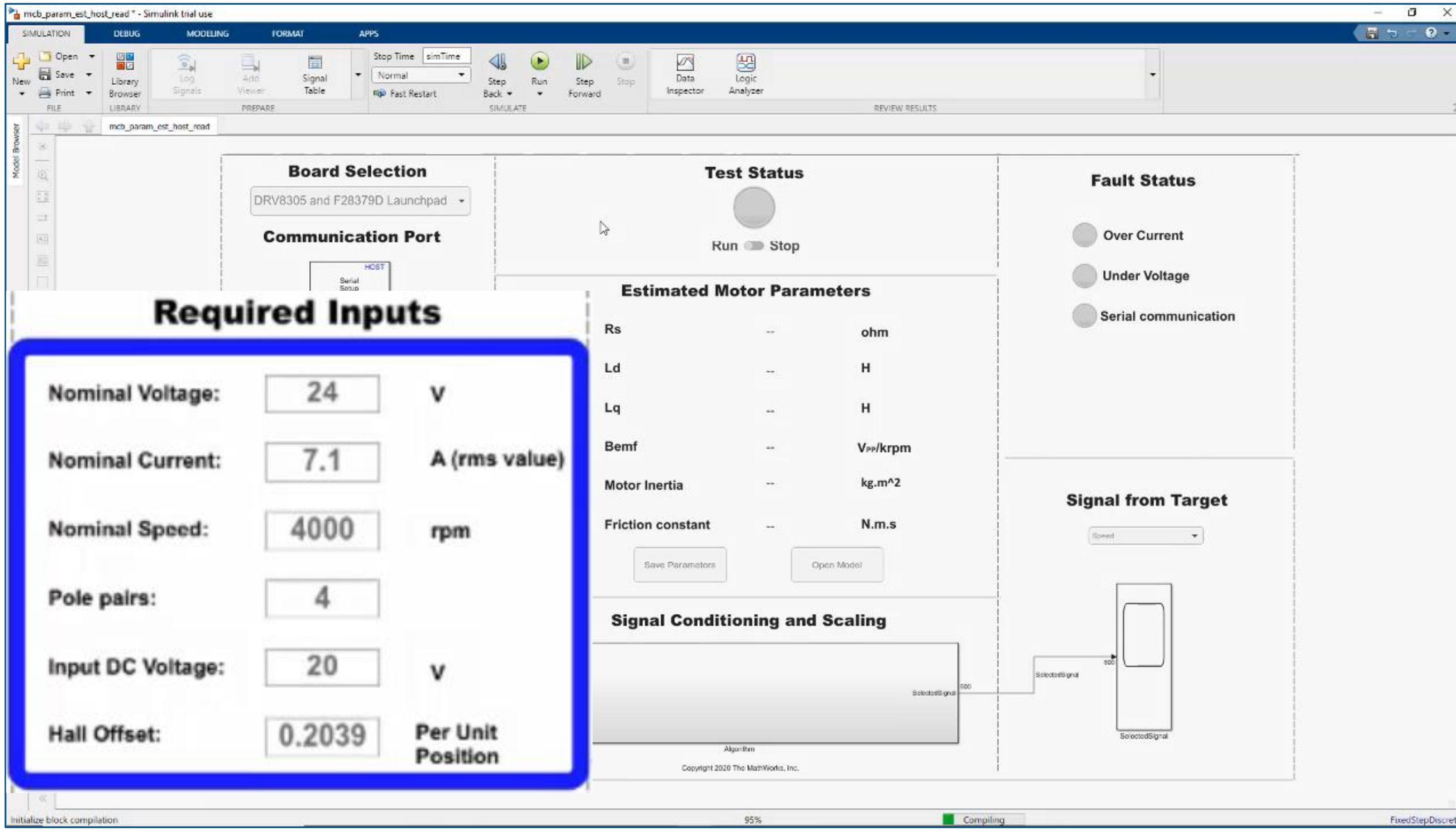
- Calibrate position sensor offset



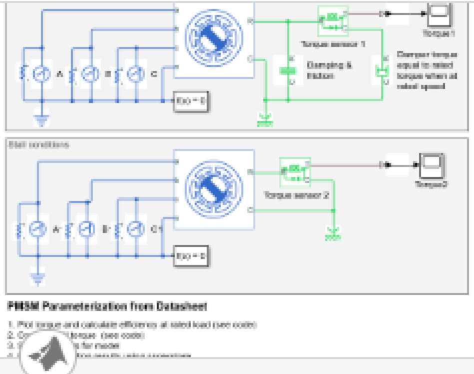
Parameter Estimation



- Instrumented tests running on the target
- Host model to start and control parameter estimation



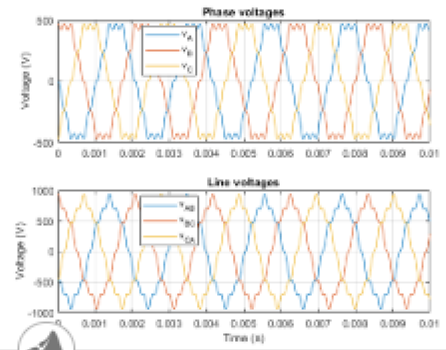
Bonus: Other Techniques to Parameterize Motor Models



PMSM Parameterization from Datasheet

Two test harnesses that add confidence that a PMSM is correctly parameterized from a datasheet. It also calculates motor efficiency at

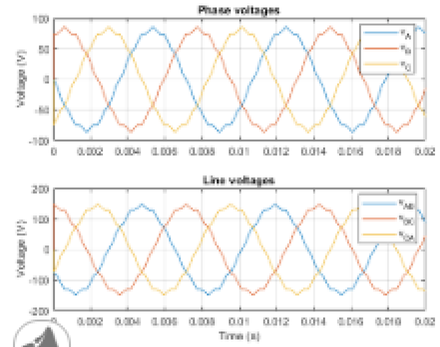
Open Model



Import IPMSM Flux Linkage Data from ANSYS Maxwell

Import a motor design from ANSYS® Maxwell® into a Simscape™ simulation.

Open Model



Import IPMSM Flux Linkage Data from Motor-CAD

Import a motor design from Motor-CAD into a Simscape™ simulation.

Open Model

Generate Parameters for Flux-Based PMSM Block

Using MathWorks tools, you can create lookup tables for an interior permanent magnet synchronous motor (PMSM) controller that characterizes the *d*-axis and *q*-axis current as a function of *d*-axis and *q*-axis flux.

To generate the flux parameters for the Flux-Based PMSM block, follow these workflow steps. Example script `CreatingIdqTable.m` calls `gridfit` to model the current surface using scattered or semi-scattered flux data.

Workflow	Description
Step 1: Load and Preprocess Data	Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA): • *d*- and *q*- axis current • *d*- and *q*- axis flux • Electromagnetic motor torque
Step 2: Generate Evenly Spaced Table Data From Scattered Data	Use the `gridfit` function to generate evenly spaced data. Visualize the flux surface plots.
Step 3: Set Block Parameters	Set workspace variables that you can use for the Flux-Based PM Controller block parameters.

From datasheet

From ANSYS Maxwell, JMAG, Motor-CAD FEA tools

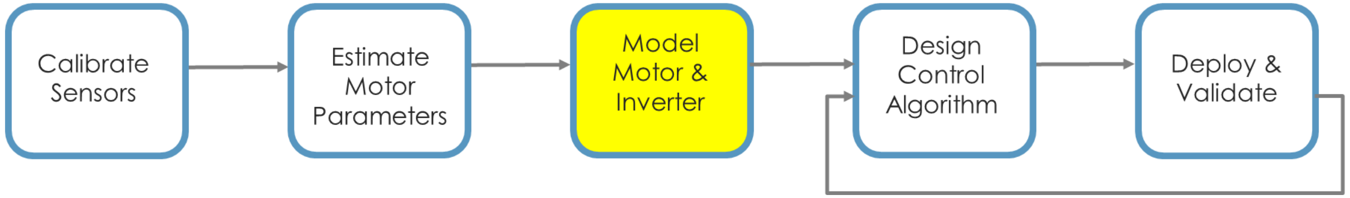
From dyno data

Simscape Electrical

Simscape Electrical

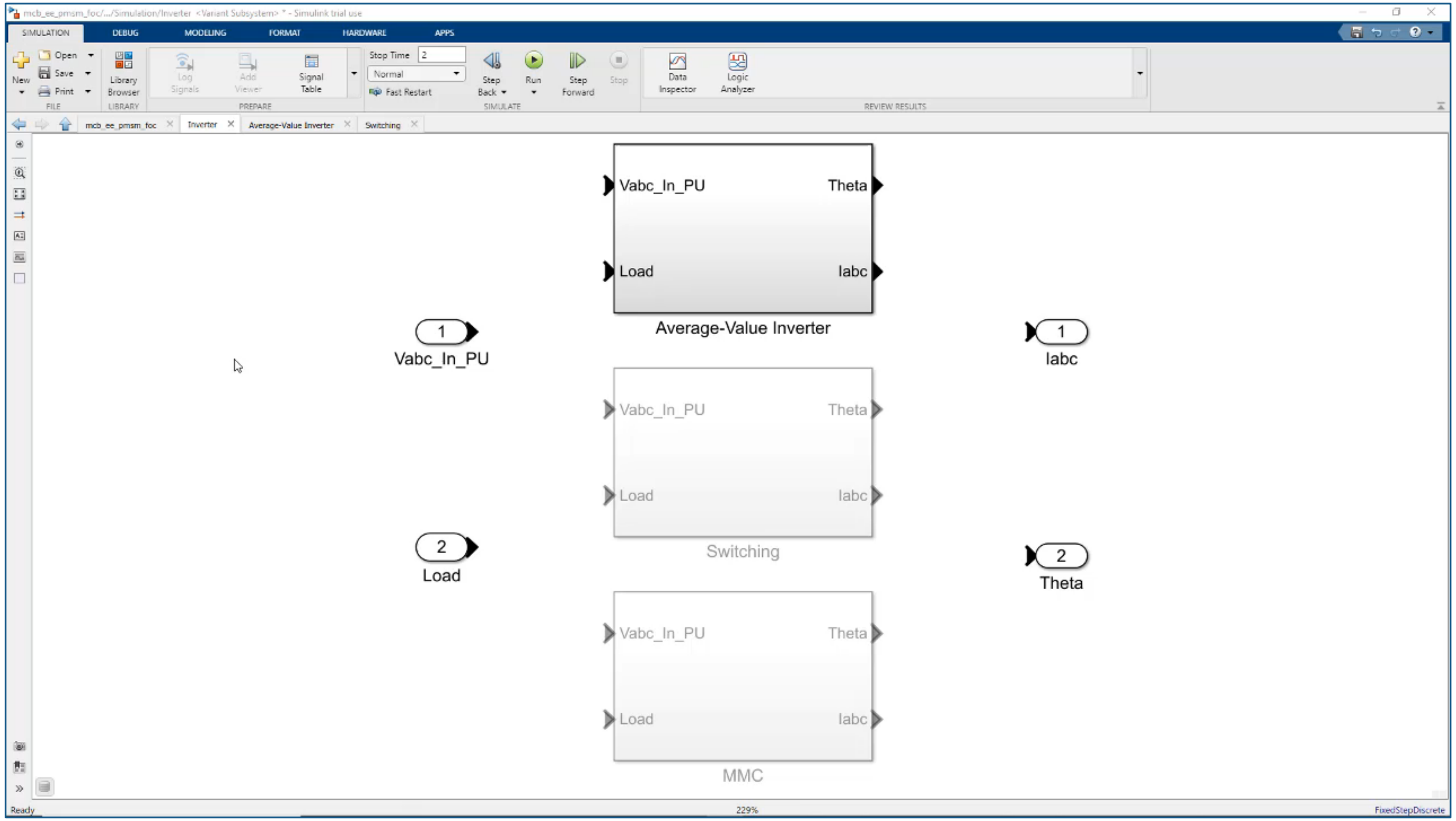
Powertrain Blockset

Modeling Motor and Inverter

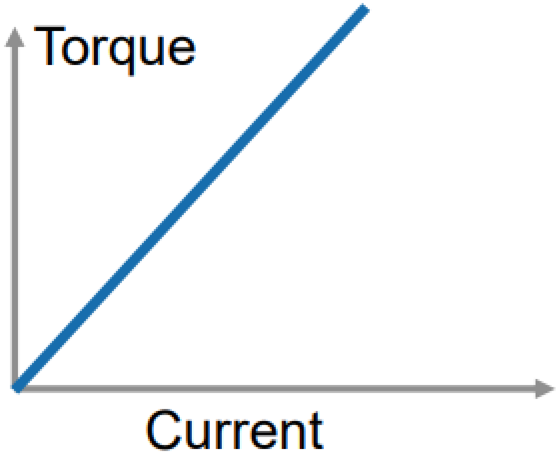


- Use linear lumped-parameter motor model

- Model inverter as an average-value inverter or model switching with Simscape Electrical

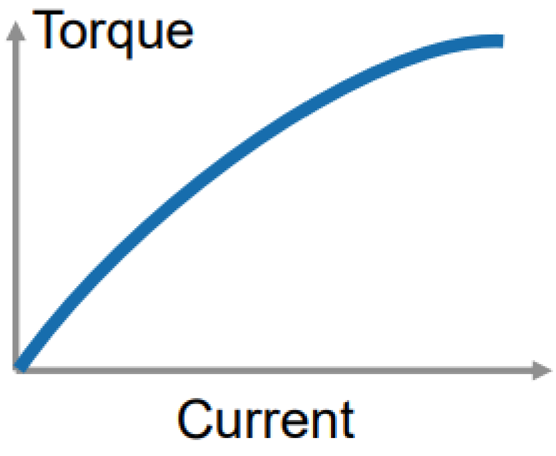


Bonus: Modeling at Needed Level of Fidelity



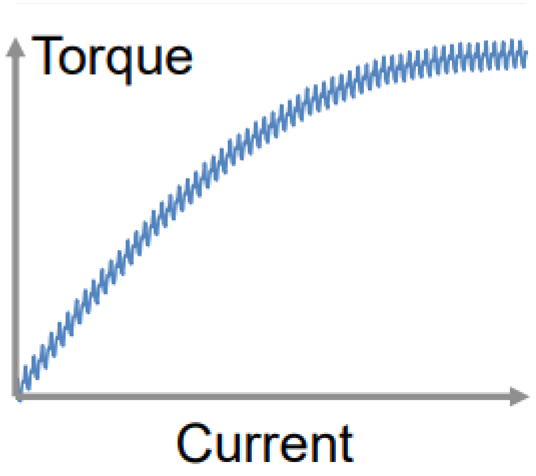
Lumped Parameter

Motor Control Blockset
Simscape Electrical



Saturation

Simscape Electrical



Saturation +
Spatial Harmonics

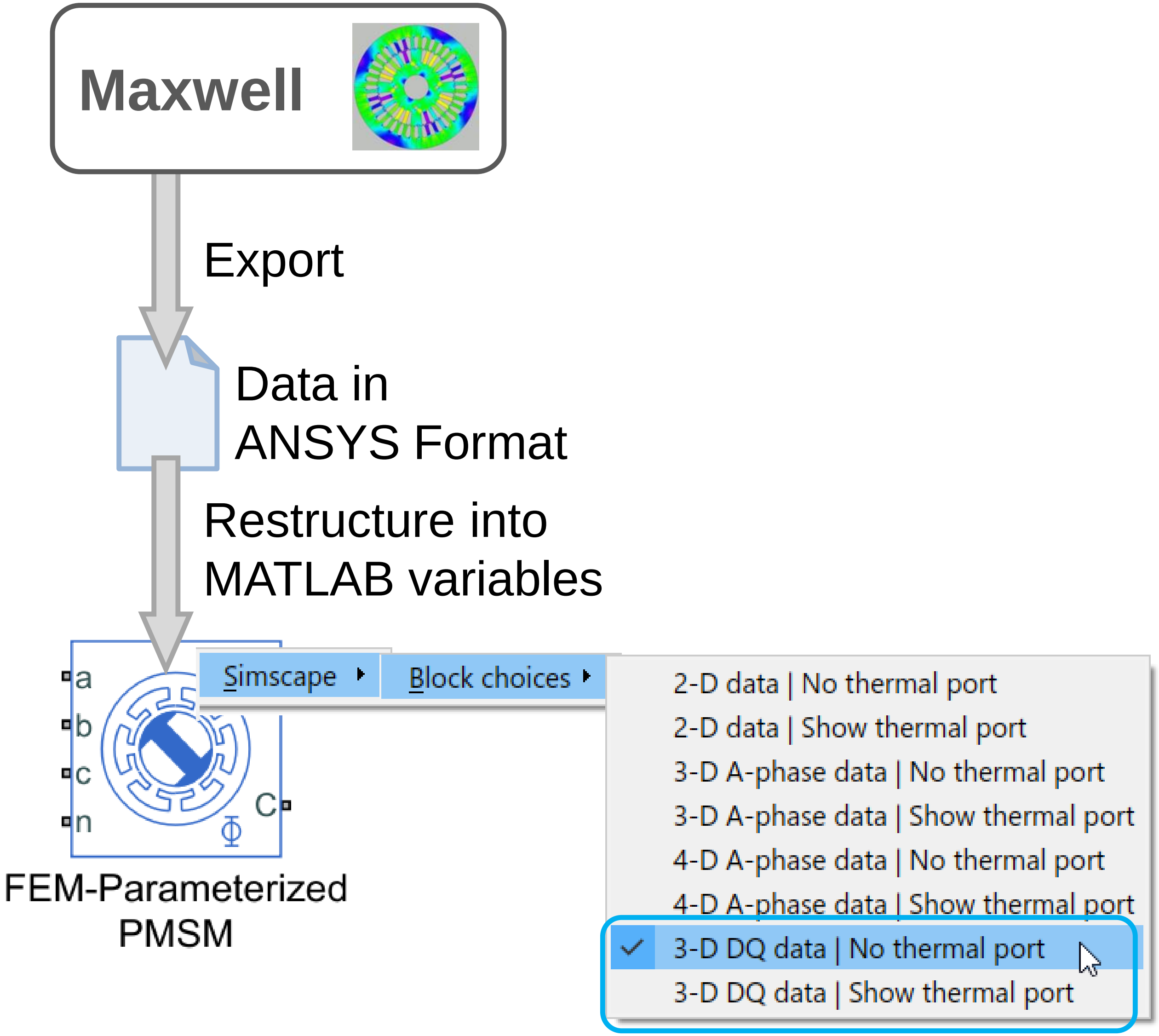
Simscape Electrical

Bonus: Motor Modeling Using Simscape Electrical

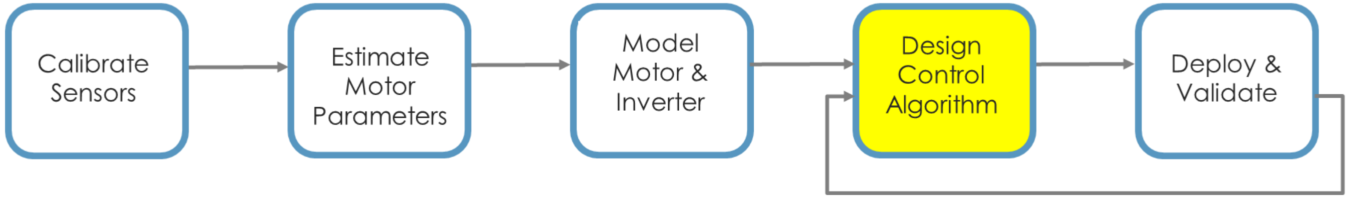
Nonlinear PMSM Model

- Define PMSM behavior using d- and q-axis flux linkage

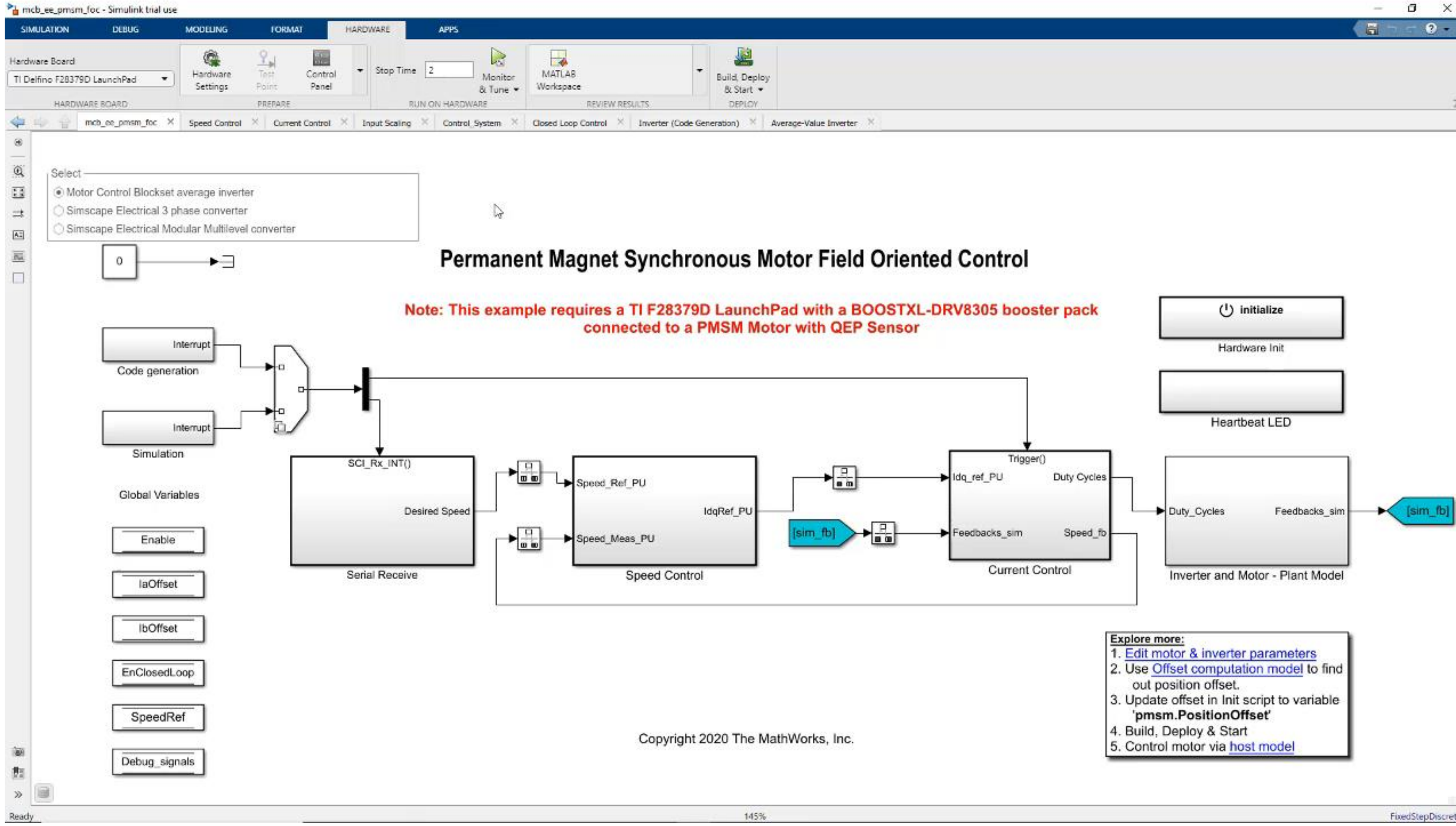
- Parameterization option is directly compatible with Maxwell, JMAG and Motor-CAD data
  - With a few changes to text file, MATLAB variables that match block parametrization can be generated



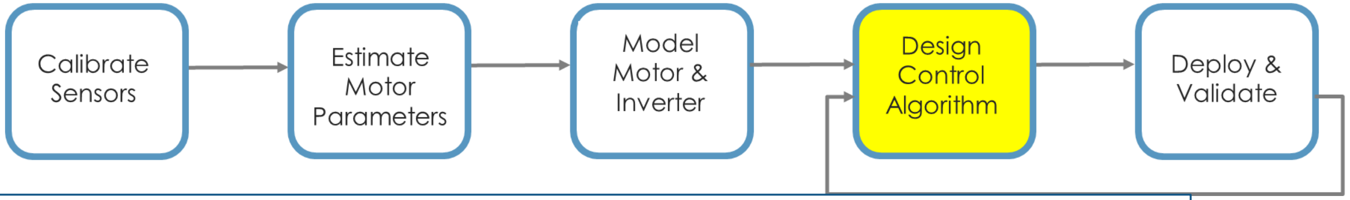
Control Algorithm Design



- Model field-oriented control algorithm
- Model sensor decoders or sensorless observers
- Tune loop gains
- Verify in closed-loop simulation



Control Algorithm Design




```
%% Controller design // Get ballpark values!
PI_params = mcb.internal.SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed);
%Updating delays for simulation
PI_params.delay_Currents    = int32(Ts/Ts_simulink);
PI_params.delay_Position    = int32(Ts/Ts_simulink);
PI_params.delay_Speed       = int32(Ts_speed/Ts_simulink);
PI_params.delay_Speed1      = (PI_params.delay_IIR + 0.5*Ts)/Ts_speed;
mcb_getControlAnalysis(pmsm,inverter,PU_System,PI_params,Ts,Ts_speed);
```

- Model field-oriented control algorithm
- Model sensor decoders or sensorless observers
- Tune loop gains
- Verify in closed-loop simulation

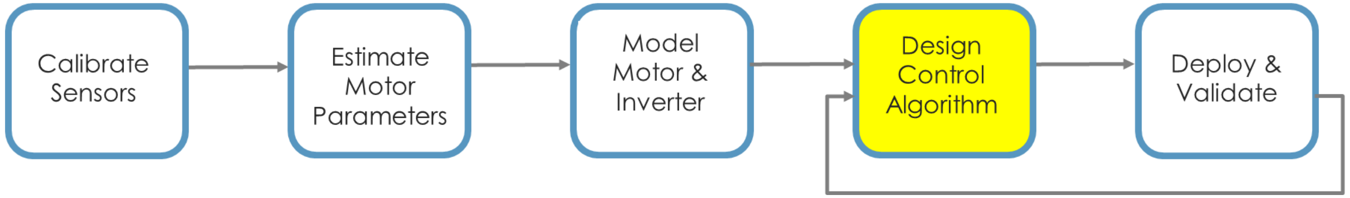
```
PI_params = mcb.internal.SetControllerParameters(pmsm,inverter,PU_System,T_pwm,Ts,Ts_speed);
```

Field	Value
T1	5.0000e-05
T2	5.0000e-04
sigma	5.0000e-05
Ti_i	3.1922e-04
Ti_id	3.3273e-04
damping	0.7071
Kp_i	2.5778
Ki_i	8.0752e+03
Kp_id	2.6869
Ki_id	8.0752e+03
Ki_texas	0.1566
Ki_d_texas	0.1503
delta	0.0263
delay_IIR	0.0200
x	1.2000
Ti_speed	0.0378
Kp_speed	0.9231
Ki_speed	24.4215

Input :

- pmsm: Motor object
- inverter: Inverter object
- PU_System: Per-Unit System
- T_pwm: PWM switching time period
- Ts: Sample time for current controllers
- Ts_speed: Sample time for speed controller

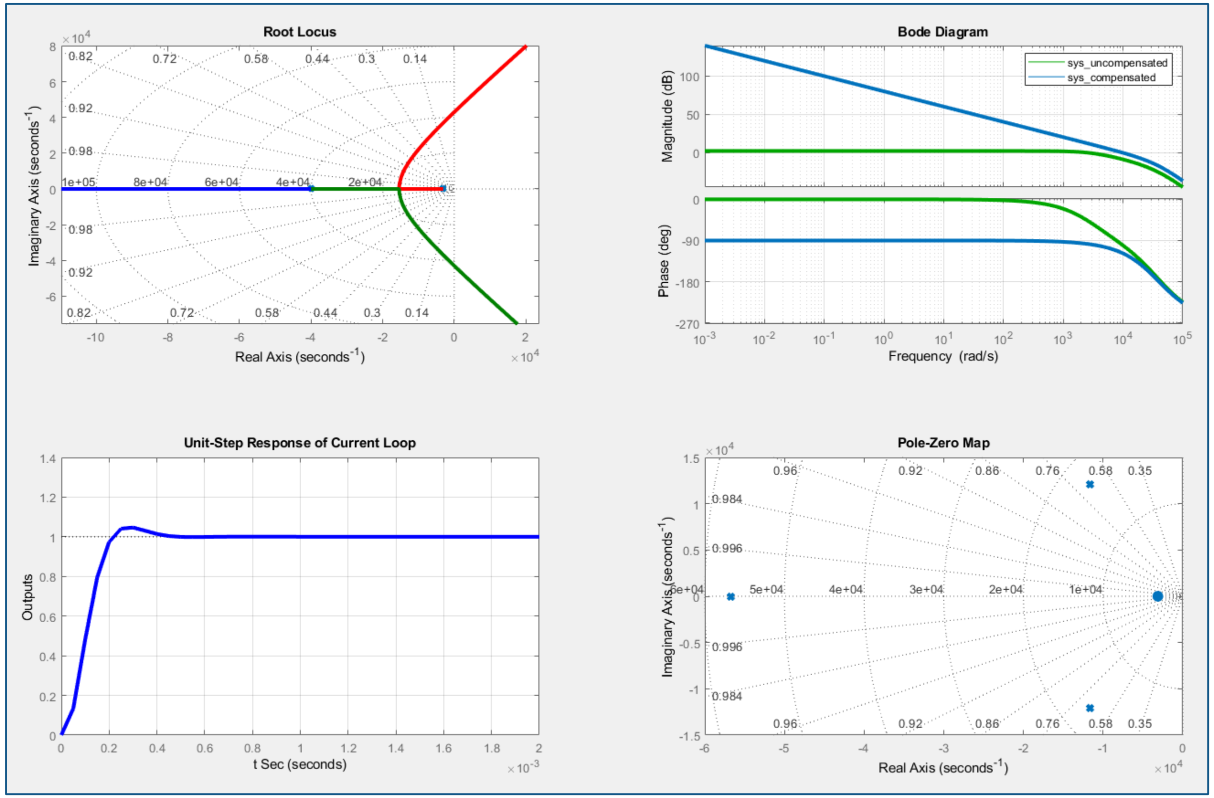
Control Algorithm Design



- Model field-oriented control algorithm

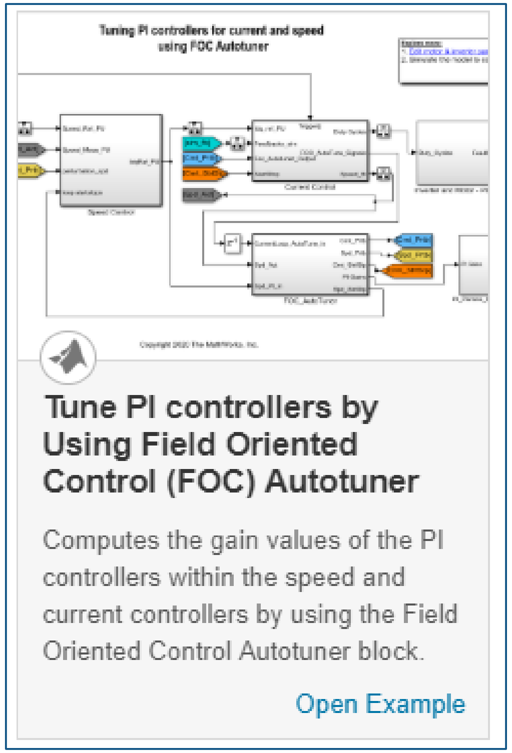
```
mcb_getControlAnalysis(pmsm,inverter,PU_System,PI_params,Ts,Ts_speed);
```

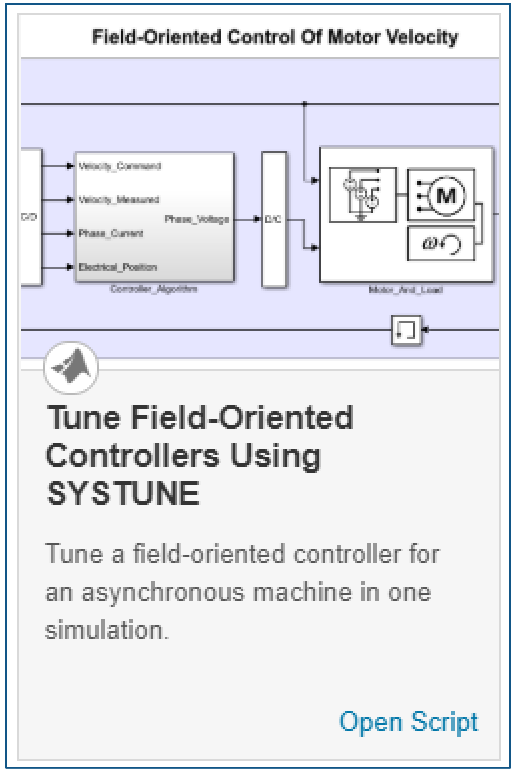
- Model sensor decoders or sensorless observers
- **Tune loop gains**
- Verify in closed-loop simulation



Bonus: Several Techniques to Tune Loop Gains

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

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

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

%% System Parameters // Hardware parameters

pmsm = mcb_SetPMSMMotorParameters('BLY171D');

%% Parameters below are not mandatory for offset computation

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

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

target = mcb_SetProcessorDetails('F28069M',PWM_frequency);

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

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

PU_System = mcb_SetPUSystem(pmsm,inverter);

%% Controller design // Get ballpark values!

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





Empirical Computation

Motor Control Blockset

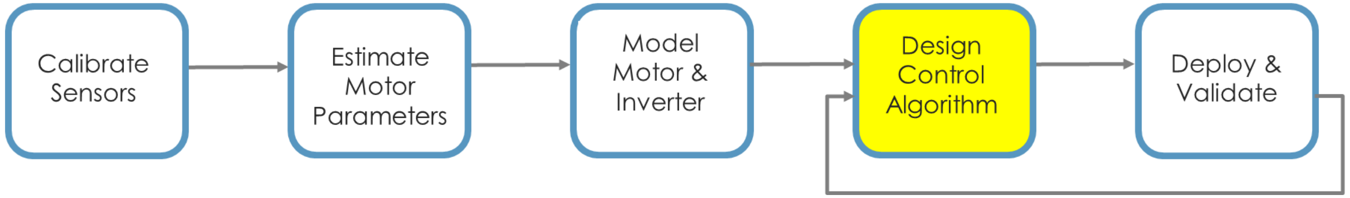
FOC Autotuner

Motor Control Blockset and Simulink Control Design

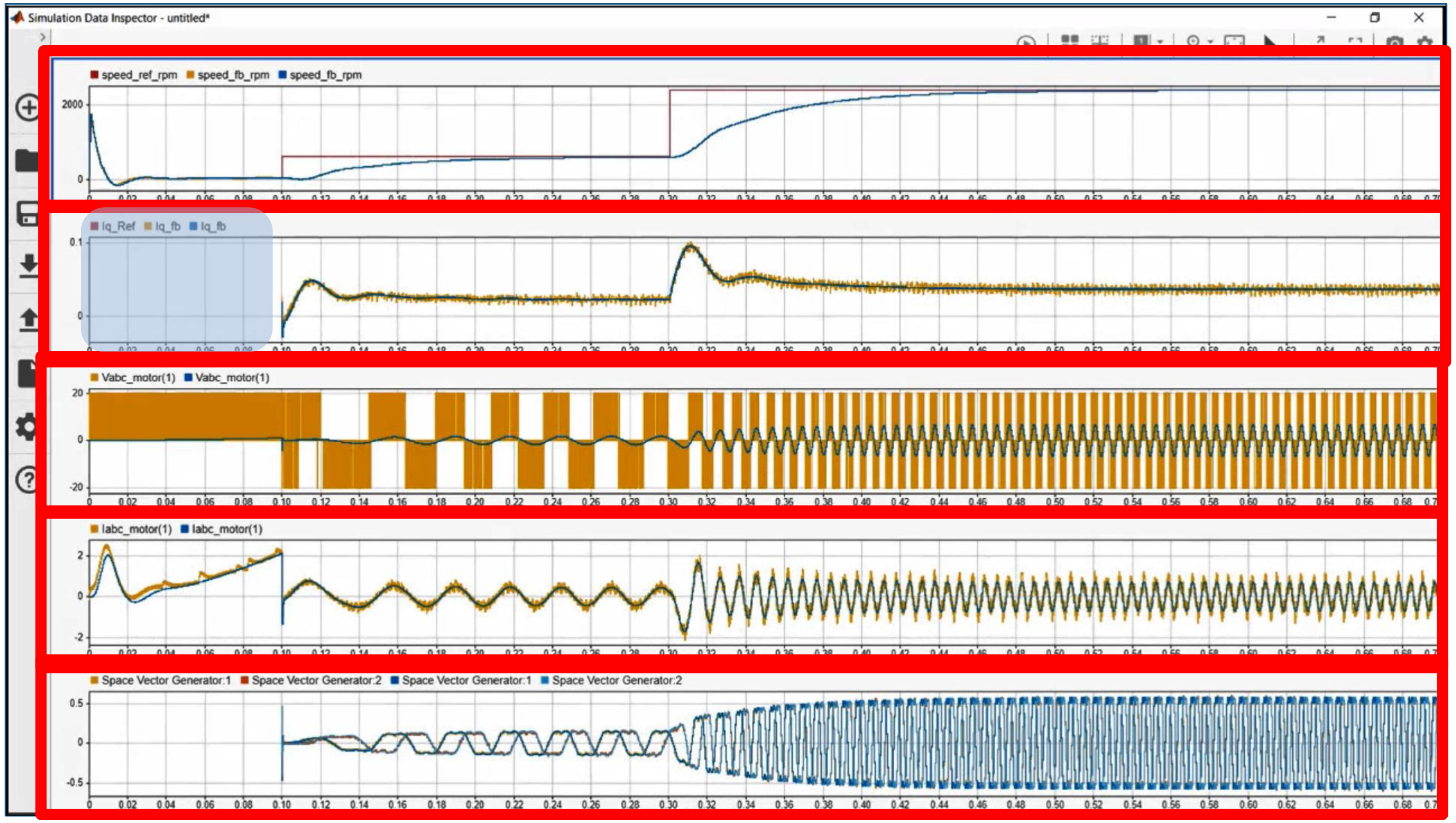
Classic Control Theory

Simulink Control Design

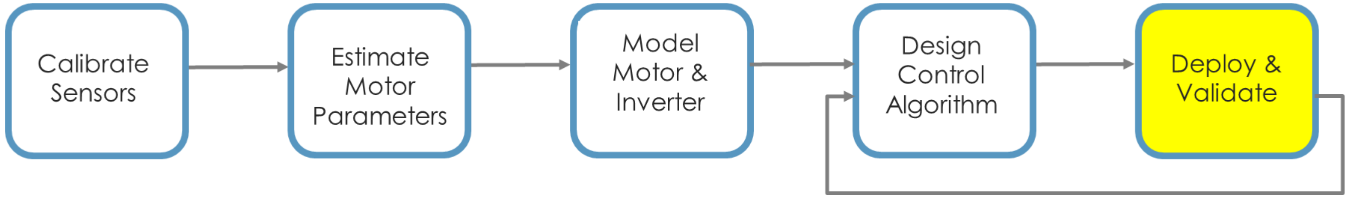
Control Algorithm Design



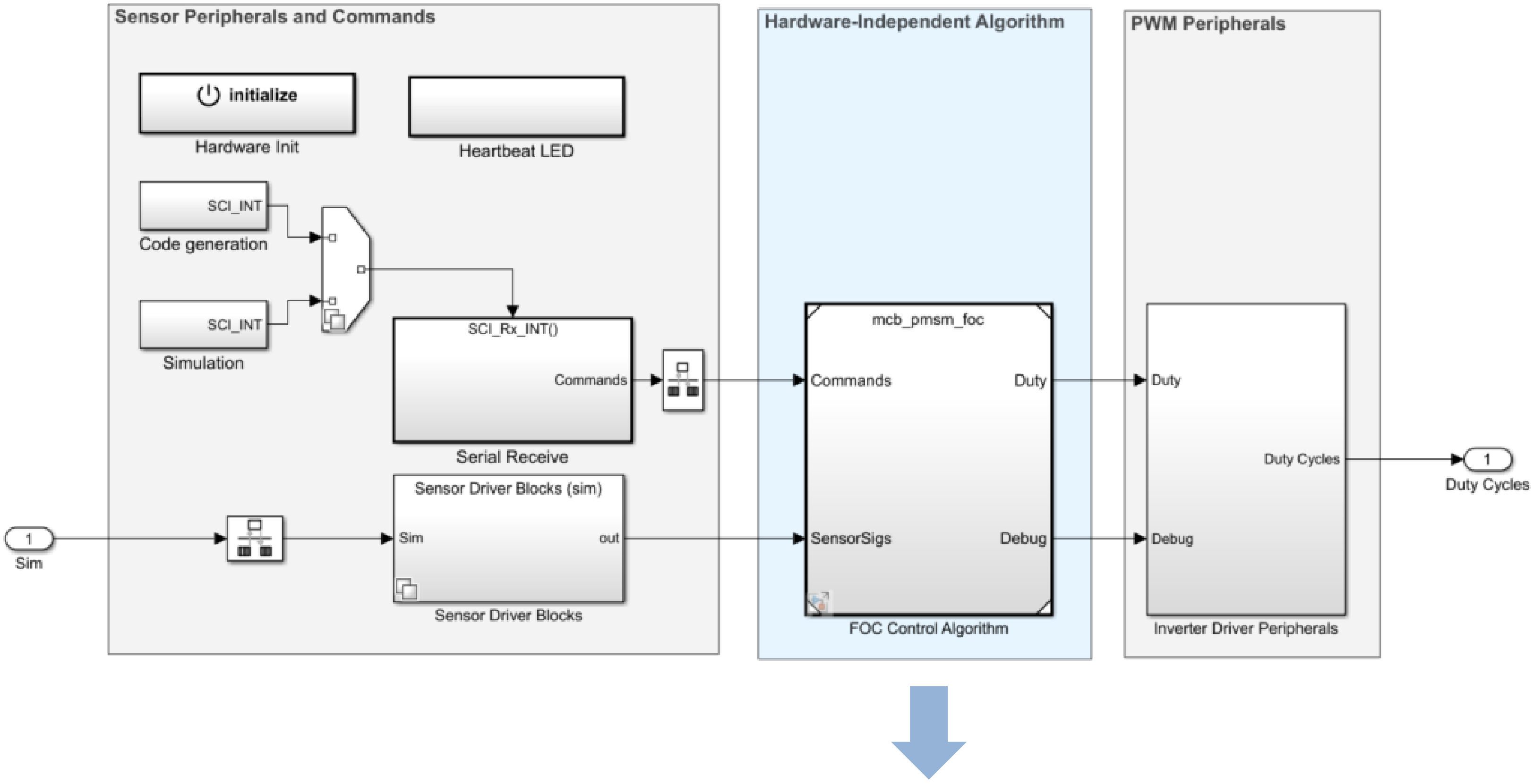
- Model field-oriented control algorithm
- Model sensor decoders or sensorless observers
- Tune loop gains
- Verify in closed-loop simulation



Deployment

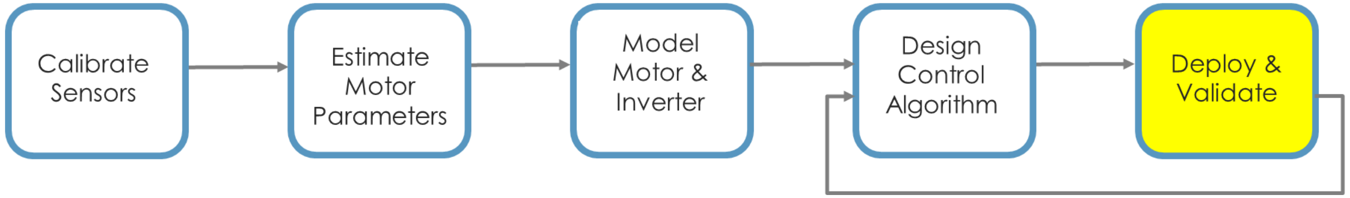


- Target any processor with ANSI C code
- Use provided example to partition the model into algorithmic and hardware-specific parts
- Generate algorithmic code for integration into embedded application

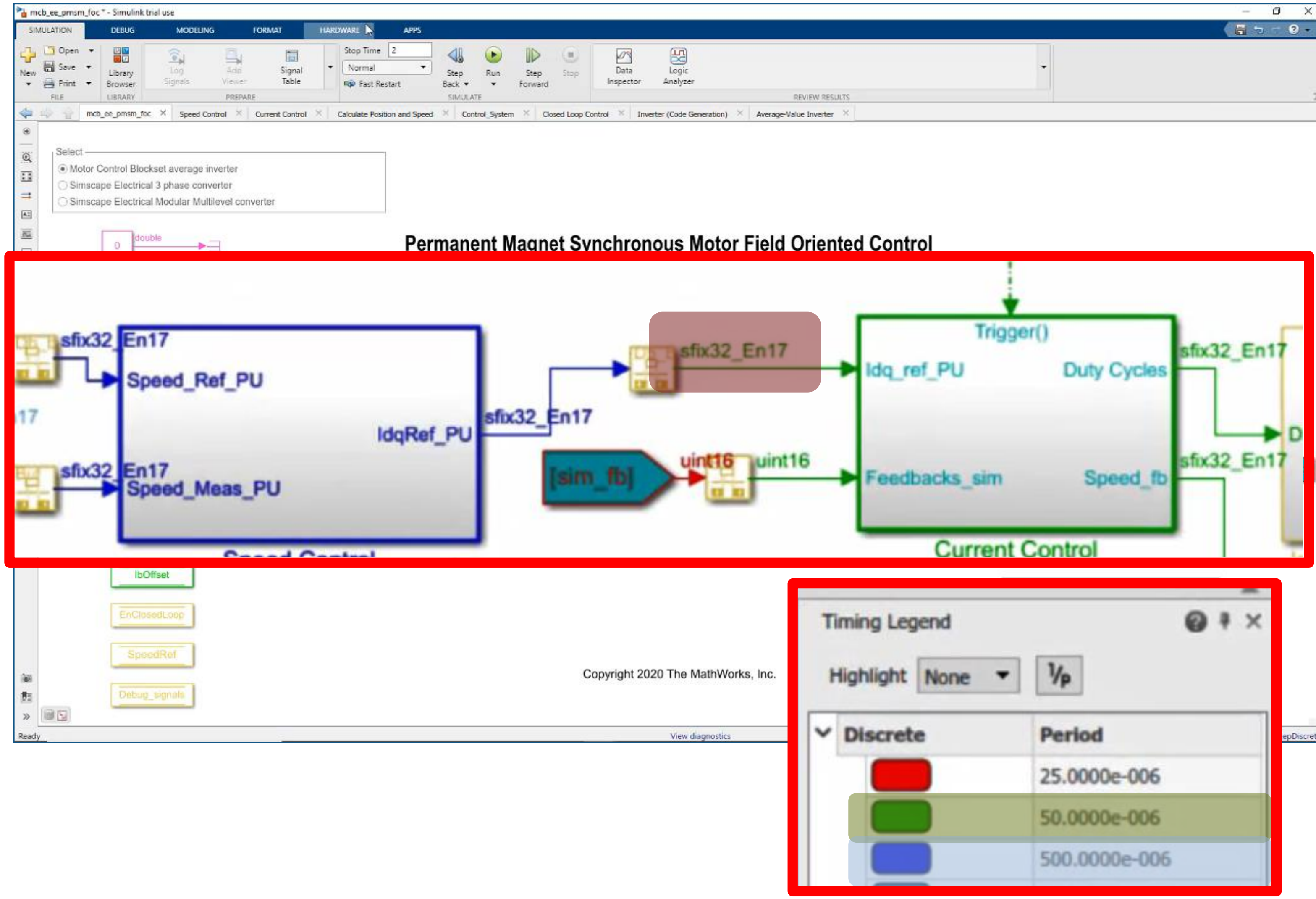


Algorithmic Code

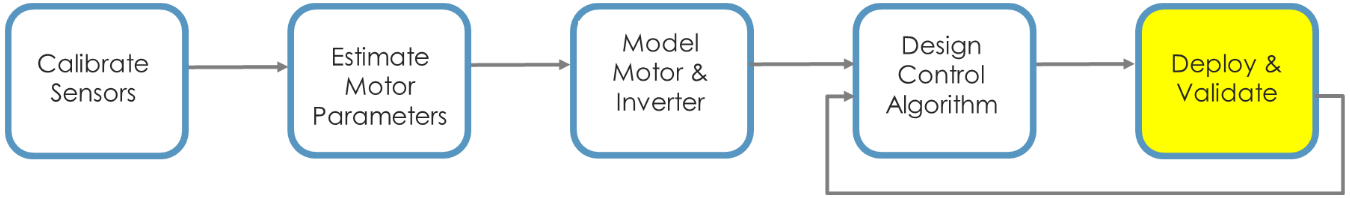
Deployment



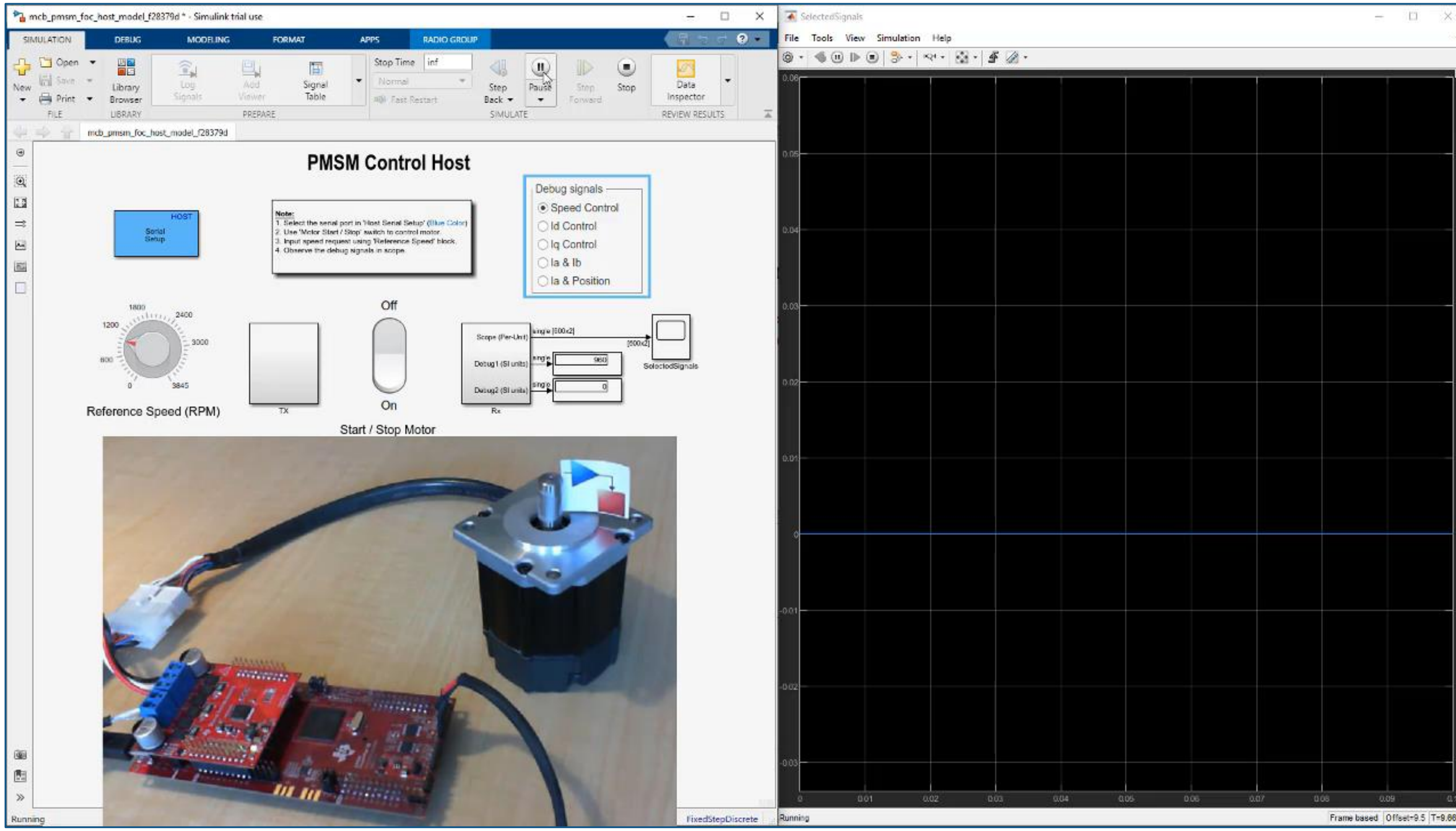
- Generate code (floating and fixed-point)
- Use host model to control and debug
- Validate on hardware



Deployment



- Generate code (floating and fixed-point)
- Use host model to control and debug
- Validate on hardware



Verify and Profile Code Using Processor-In-the-Loop(PIL) Testing

Code Execution Profiling Report for mcb_pmsm_foc_sim_v2/Current Control1

The code execution profiling repor
recorded by instrumentation probe
Profiling for more information.

1. Summary

Total time
Unit of time
Command
Timer frequency (ticks per second
Profiling data created

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Ma
[+] Current_initialize	2260	2260	
Current_step [5e-05 0]	5135	5067	
Current_terminate	540	540	

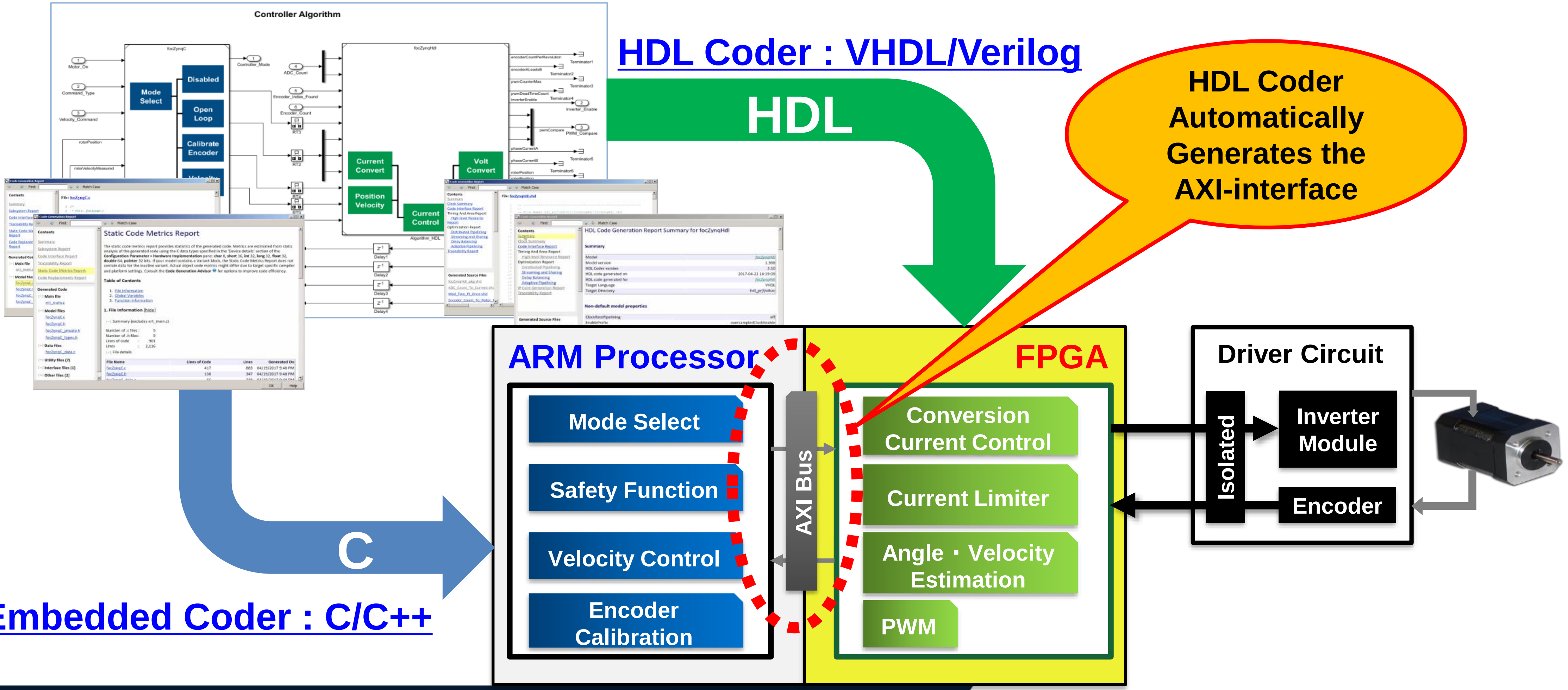
3. CPU Utilization

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

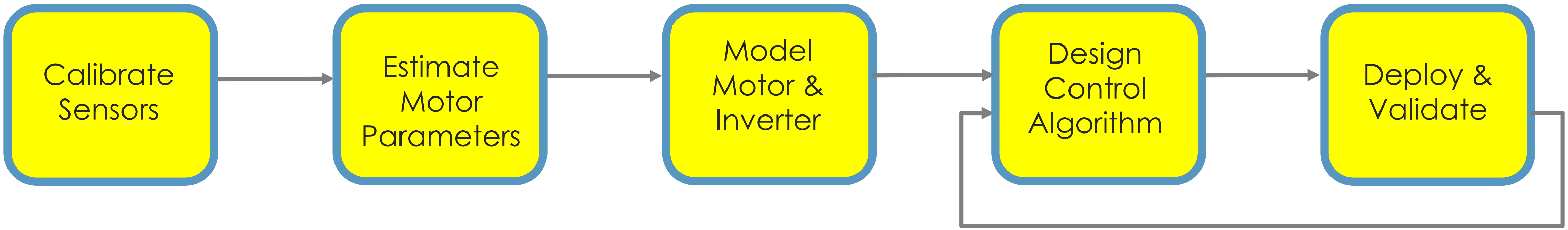
Bonus: Code Generation for MCU/FPGA/SoC

Functions Supported for HDL Code Generation
HDL Code Generation from Simulink — Blocks

- ✓ MCU or FPGA : C or HDL Code Generation through Coders
- ✓ SoC : Need to consider interface between ARM and FPGA (AXI-Bus)



Workflow for Implementing Field-Oriented Control



ATB Technologies Cuts Electric Motor Controller Development Time by 50% Using Code Generation for TI's C2000 MCU

Challenge

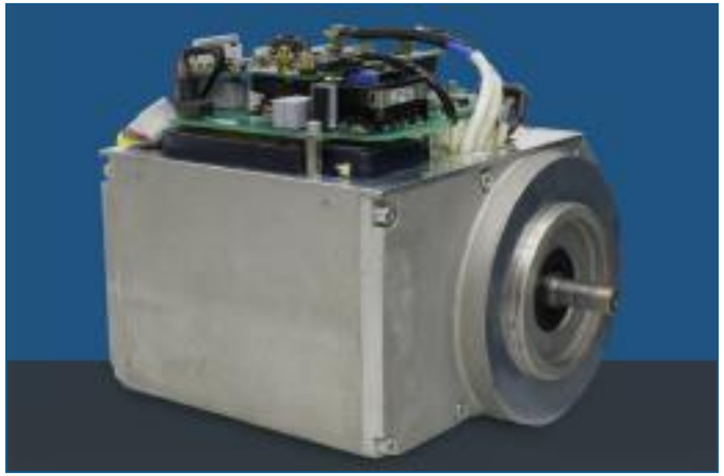
Develop control software to maximize the efficiency and performance of a permanent magnet synchronous motor

Solution

Use MathWorks tools for Model-Based Design to model, simulate, and implement the control system on a target processor

Results

- Development time cut in half
- Design reviews simplified
- Target verification and deployment accelerated



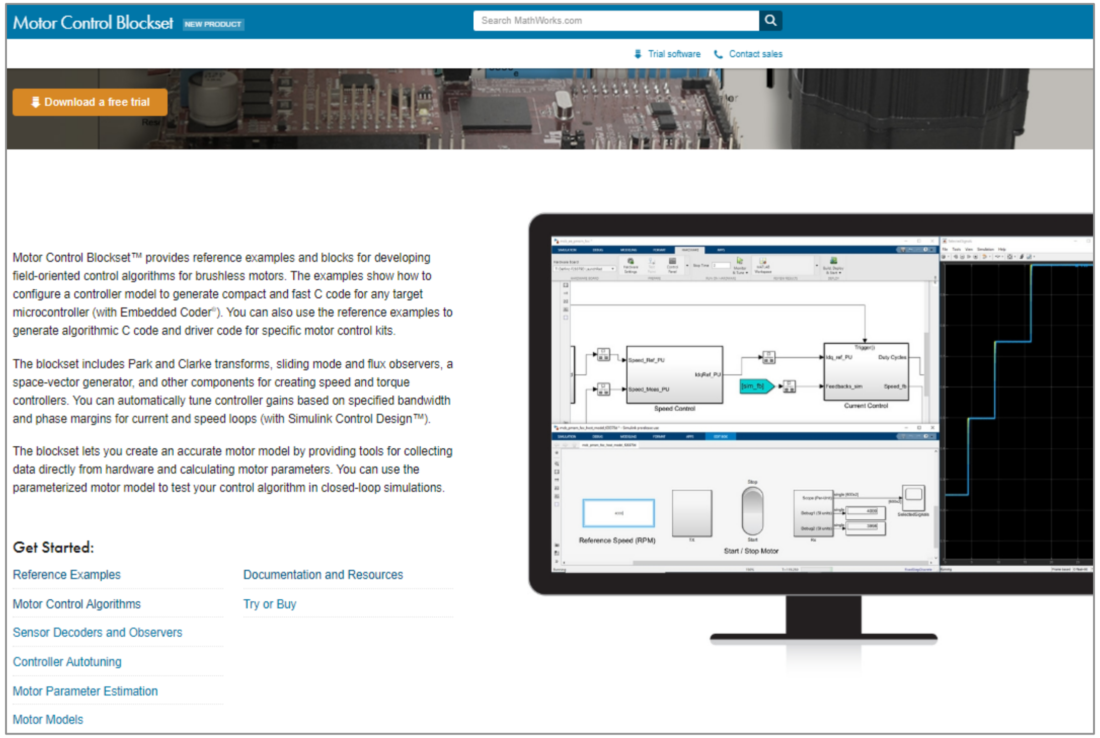
ATB Technologies permanent magnet synchronous motor.

"MathWorks tools enabled us to verify the quality of our design at multiple stages of development, and to produce a high-quality component within a short time frame."

- Markus Schertler, ATB Technologies

Use Model-Based Design for Your Next Motor Control Project!

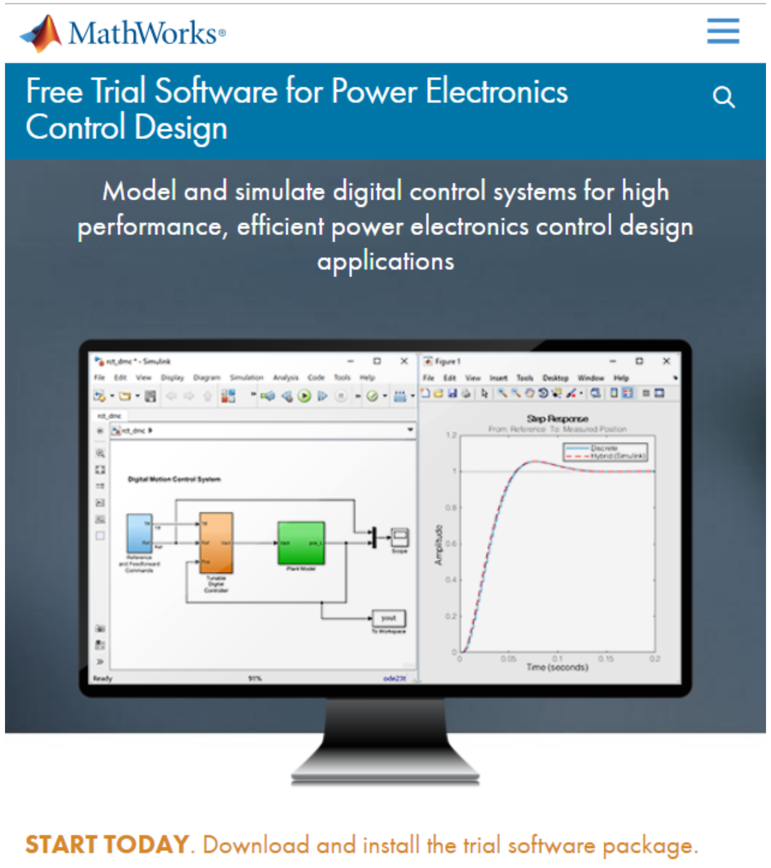
- Verify control algorithm with desktop simulation
- Generate compact and fast code from models
- Minimize development time using reference examples, sensor calibration, built-in algorithmic blocks, automated parameter estimation, and gain-tuning



Learn More

- Visit [mathworks.com/products/motor-control] and [mathworks.com/solutions/power-electronics-control]

- Get [power electronics control design trial package] with necessary tools for desktop modeling, simulation, control design, and production code generation of your next motor control project



Thank You !!