

# MATLAB EXPO

FRANCE

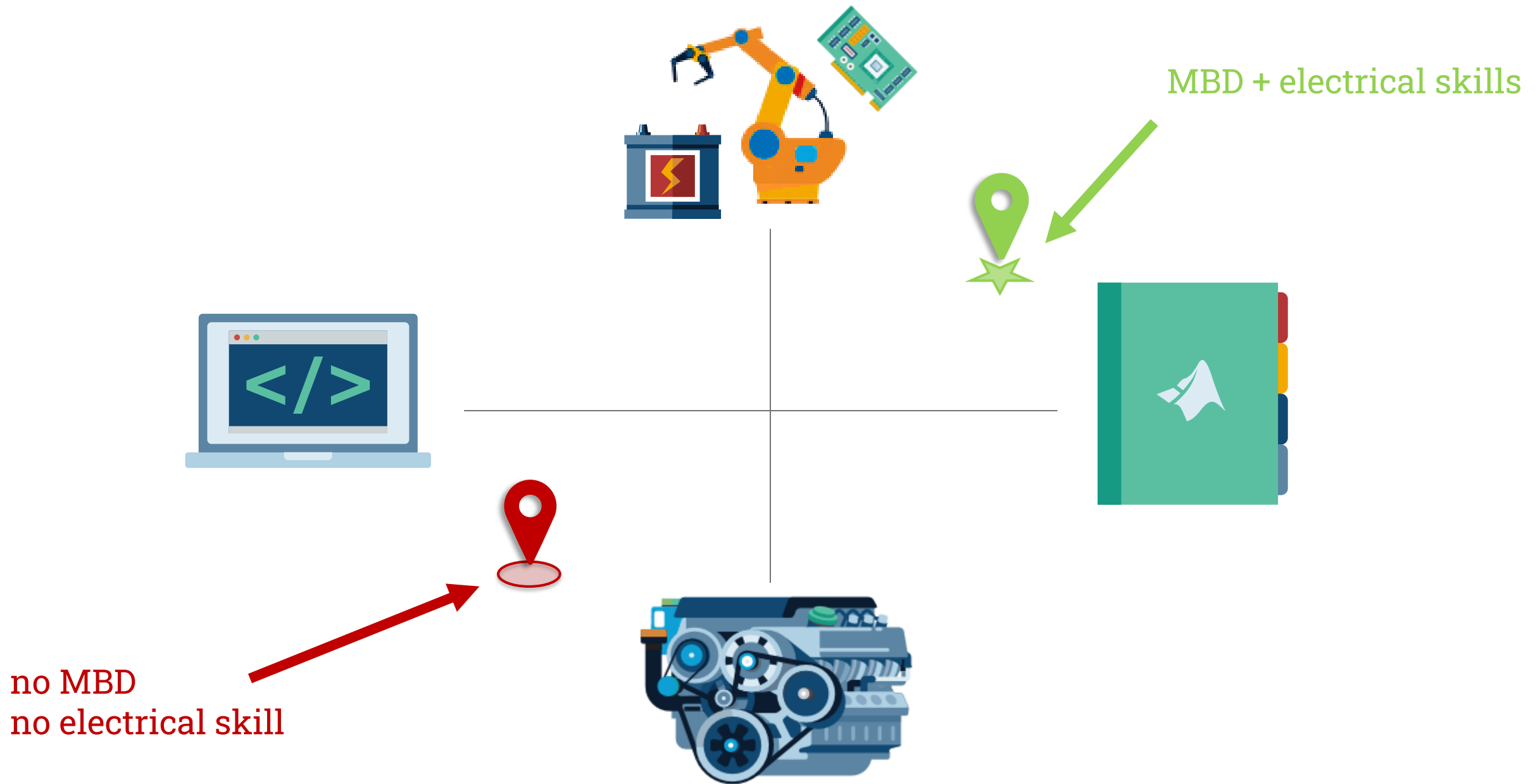
## Electrifier vos systèmes avec Simulink

*Kevin Roblet, MathWorks*



*Morgan Fremovici, MathWorks*



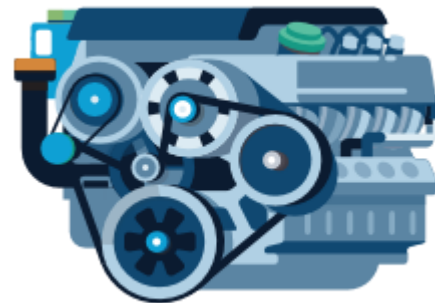
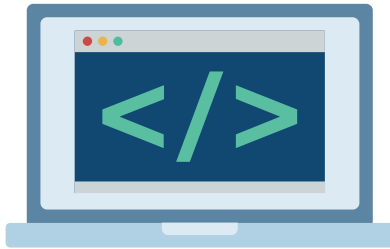
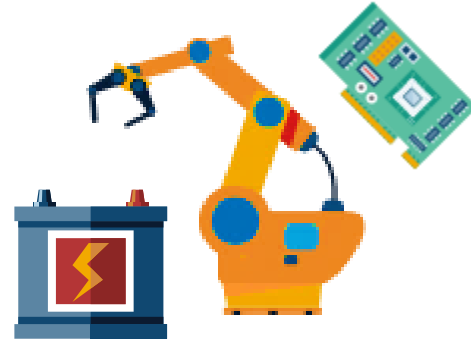


no MBD  
no electrical skill

MBD + electrical skills



Skilled with motors but not that much with MBD



Skilled with MBD but not that much with motors



I do not know that much about motors.  
Let's take this opportunity to **upskill** !



I am skilled with MBD with good ROI.  
Let's **address new markets** !

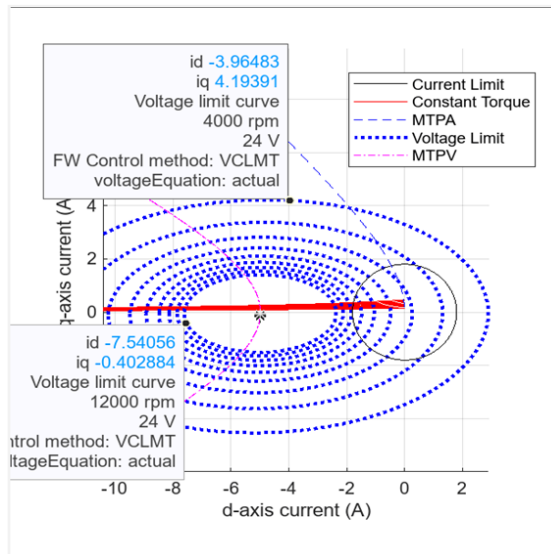
I know how to get the best of IA with  
simulation.  
Let's differentiate with **innovative features** !

Threat

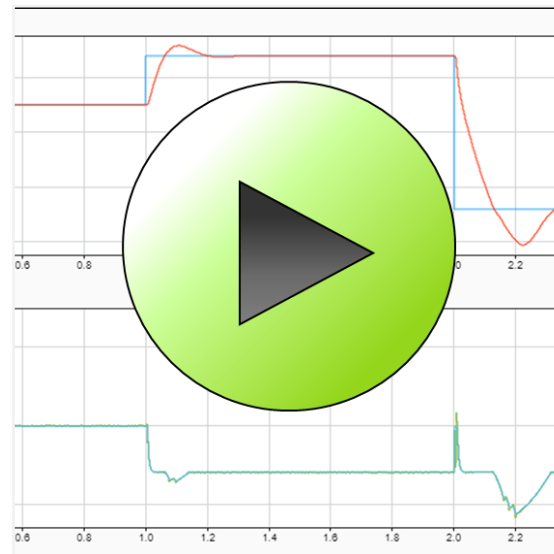
Opportunity

# models to capture knowledge and as a collaboration and communication tool

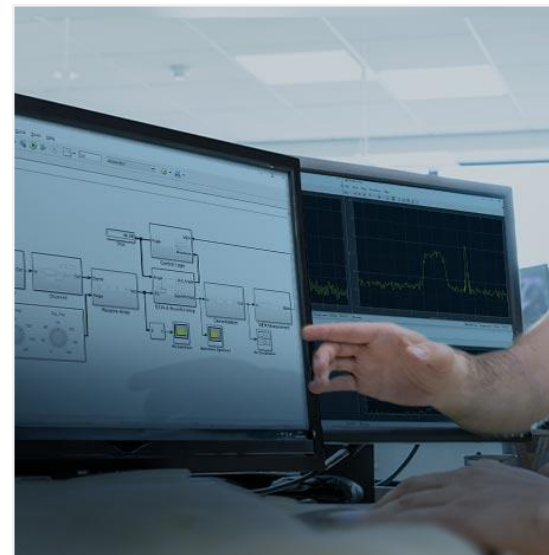
## Documentation



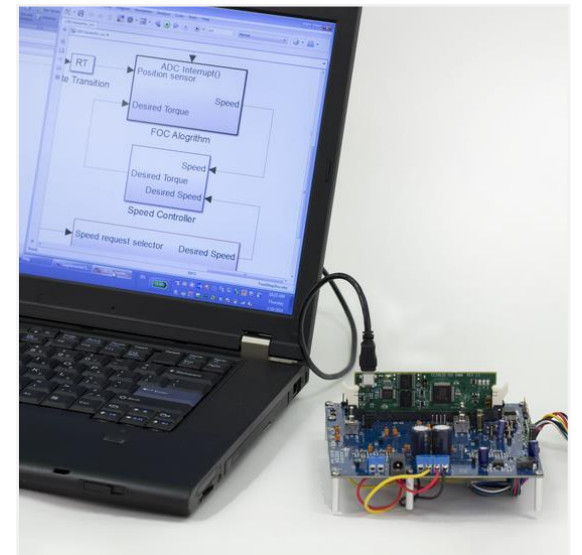
## Simulation



## Collaboration



## Hands-on kit



Our motor expertise is critical, how do we...  
... ensure **knowledge flow** in the company ?  
... handle software **maintainability** ?  
... **train** new hires ?

Raw material prices, supply chain  
disruptions... we need to...  
... **depend less on prototypes**  
... **get more job done in simulation**  
... **be less dependent on specific chips**



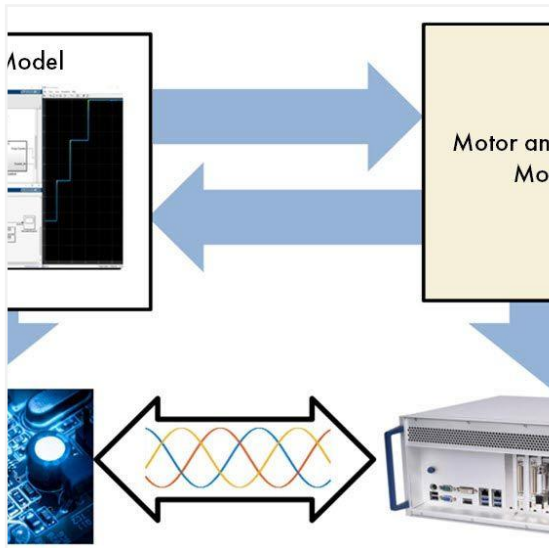
Our customers have new expectations, let's...  
... deploy on **FPGA** !  
... achieve **certification** !

Threat

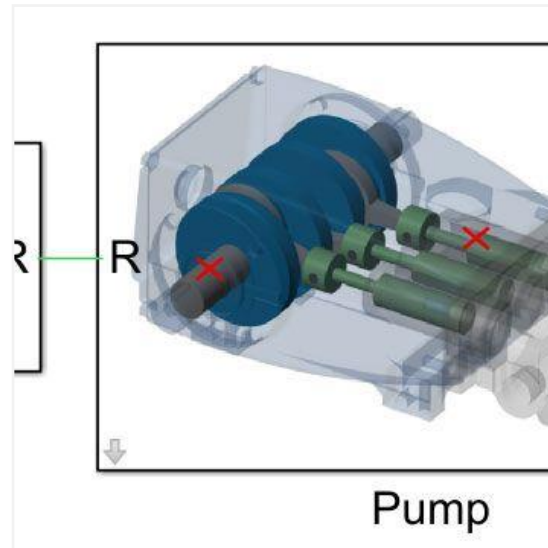
Opportunity

# models to try out new ideas through short agile iteration cycles

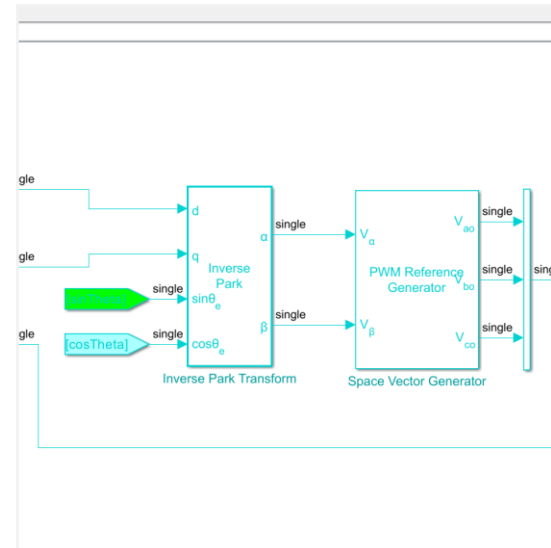
## Virtual prototyping



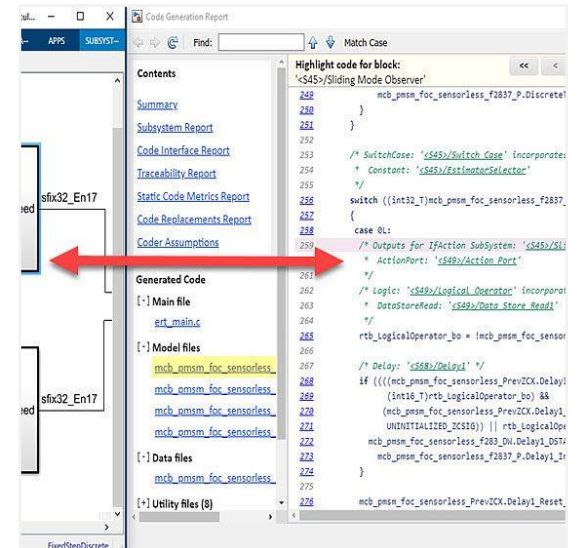
## System-level simulation



## Model elaboration



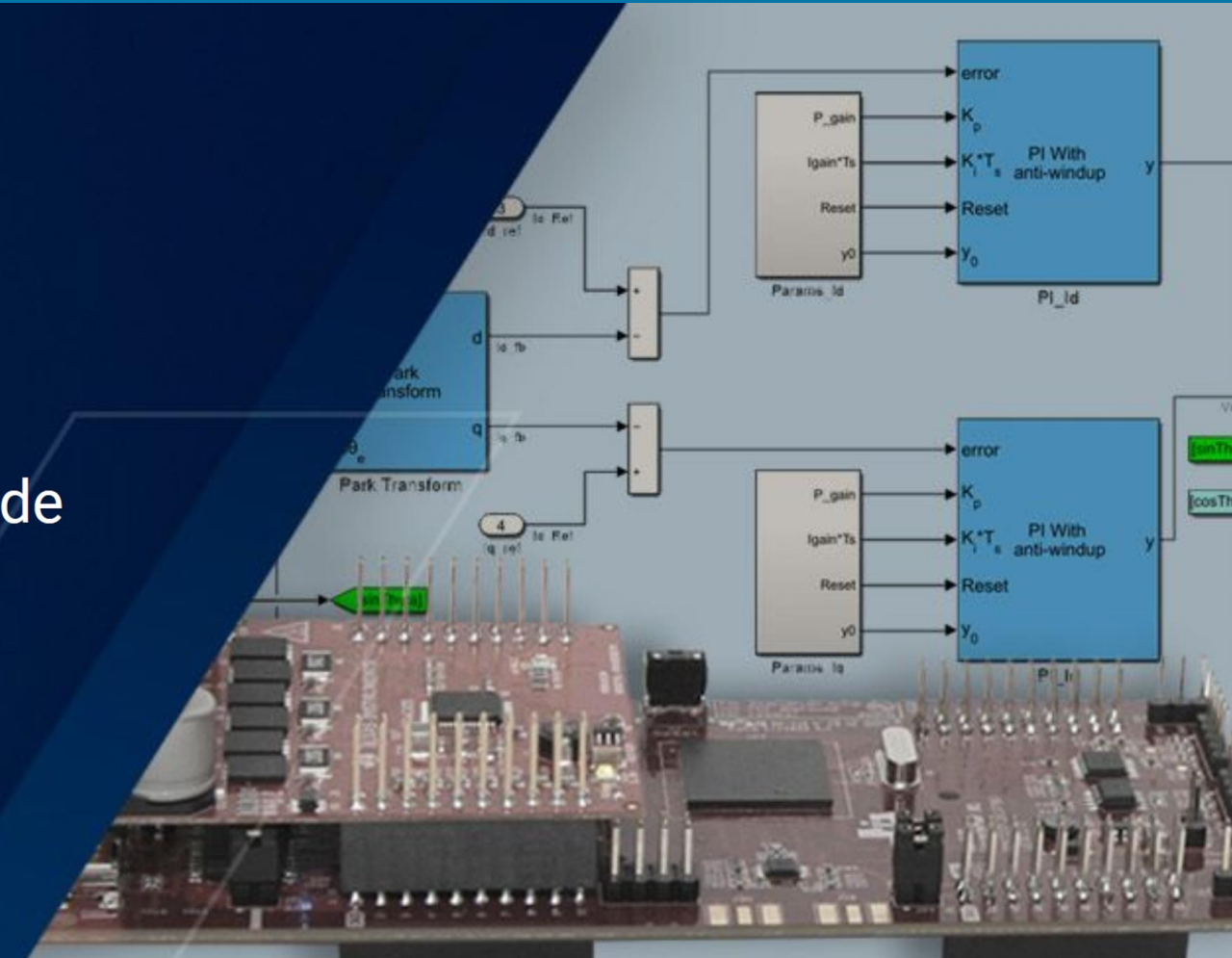
## Automation





# Motor Control Blockset

Concevoir et implémenter des algorithmes de contrôle moteur





**KNOWLEDGE**

**AGILE**

**RESILIENT**

**LEAN**

**Managing  
Model-Based  
Design**

**TIME TO MARKET**

**COST**

**INNOVATION**

**QUALITY**

**UPSKILL**



**KNOWLEDGE**

**AGILE**

**RESILIENT**

**LEAN**

## 8 Core Concepts

Model-Based Design



Executable Specification



Model elaboration



System-level simulation



What-if analysis



Continuous test and verification



Virtual prototyping



Automation



Knowledge capture

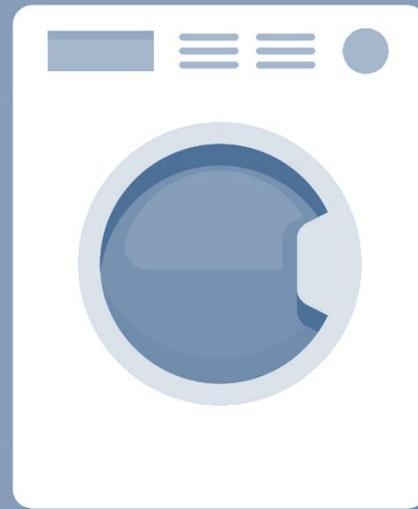
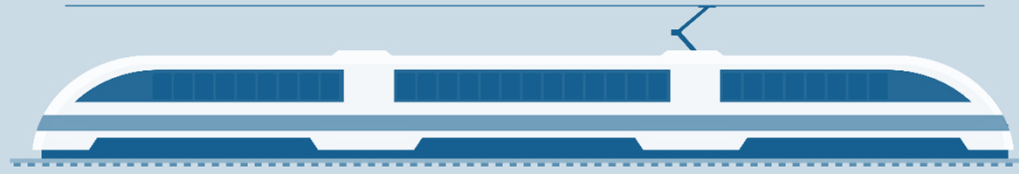
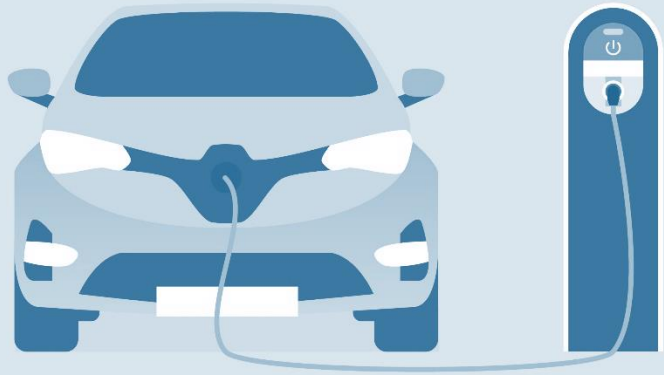
**TIME TO MARKET**

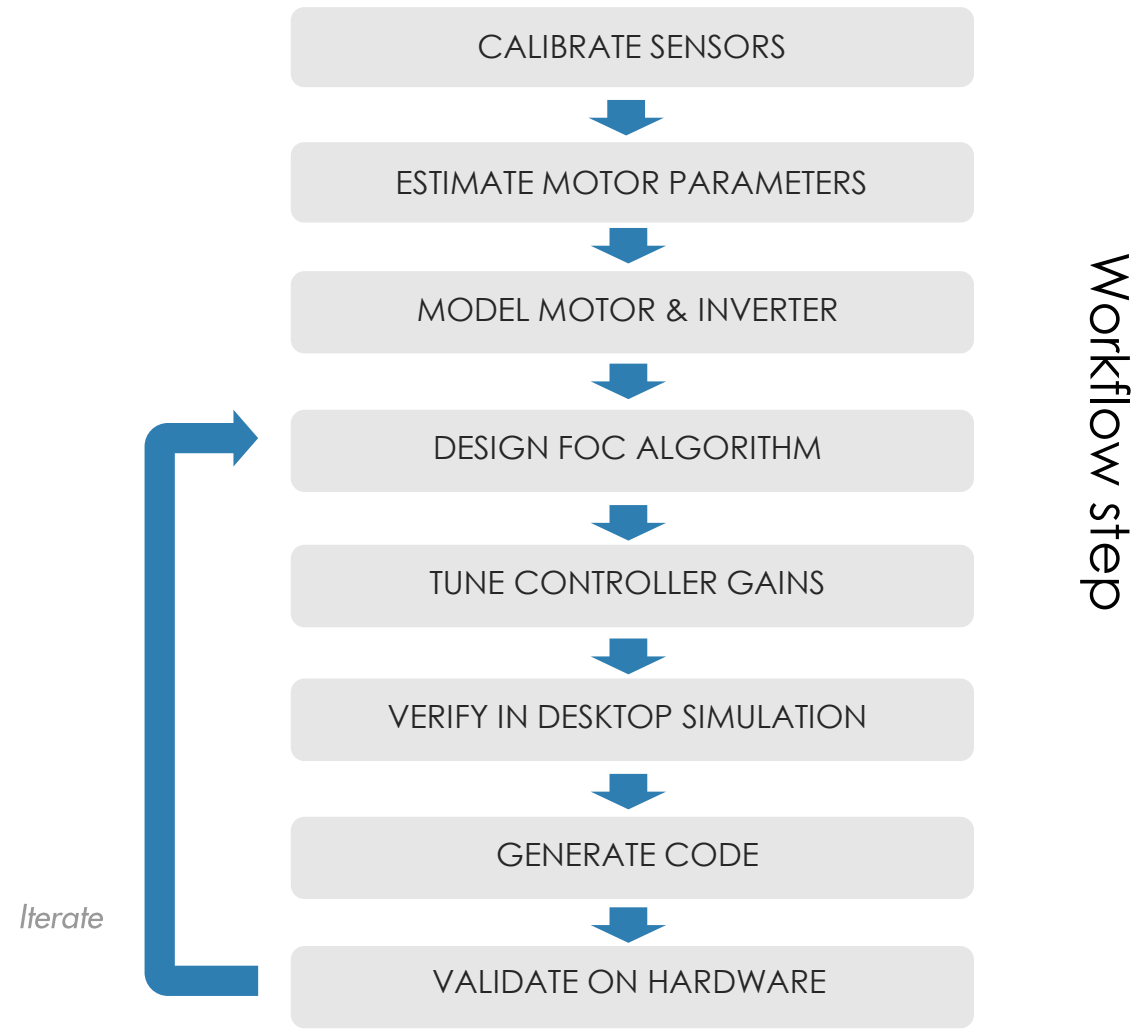
**COST**

**INNOVATION**

**QUALITY**

**UPSKILL**

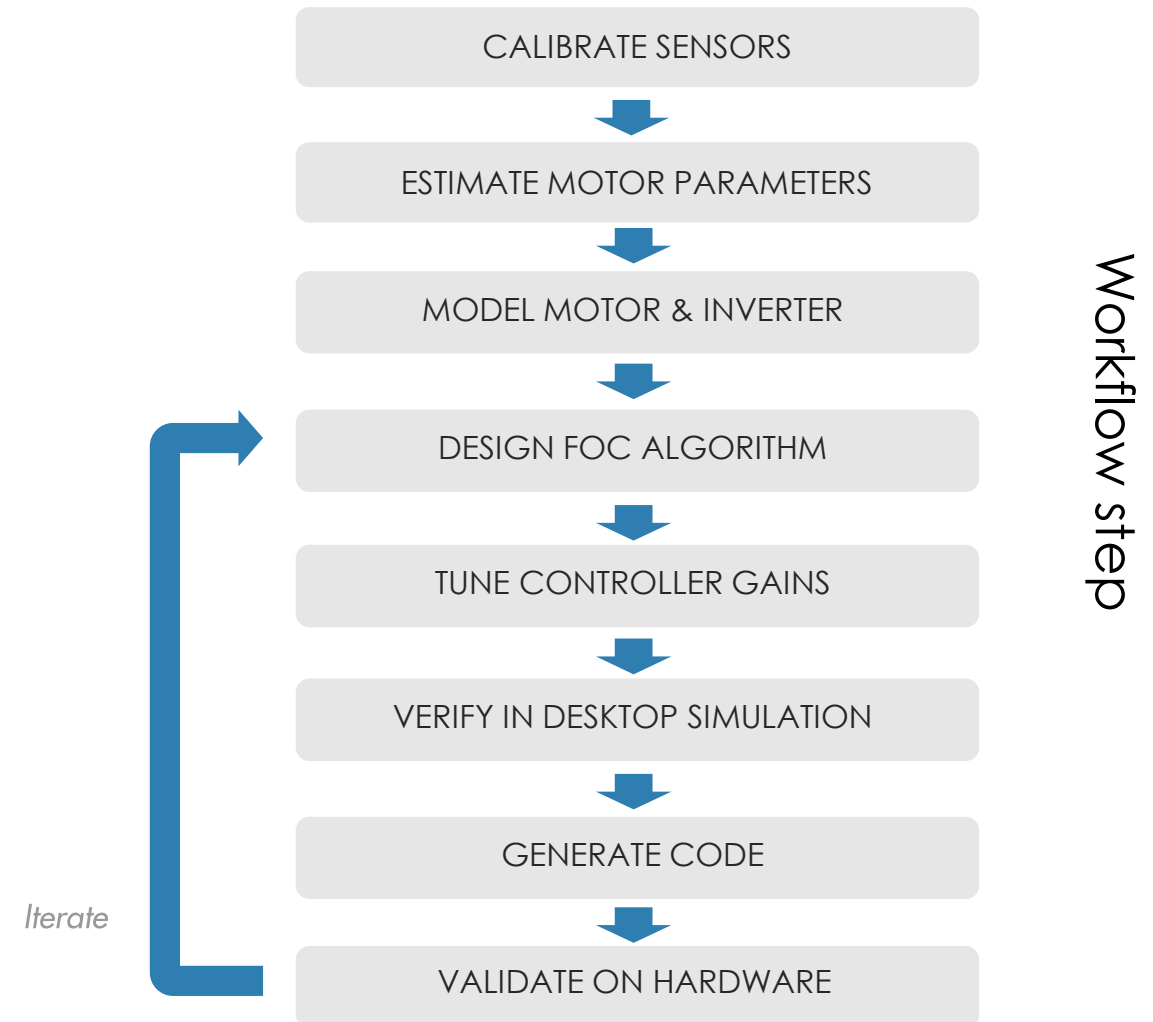




# Agenda

## From Desktop Simulation to Software Deployment

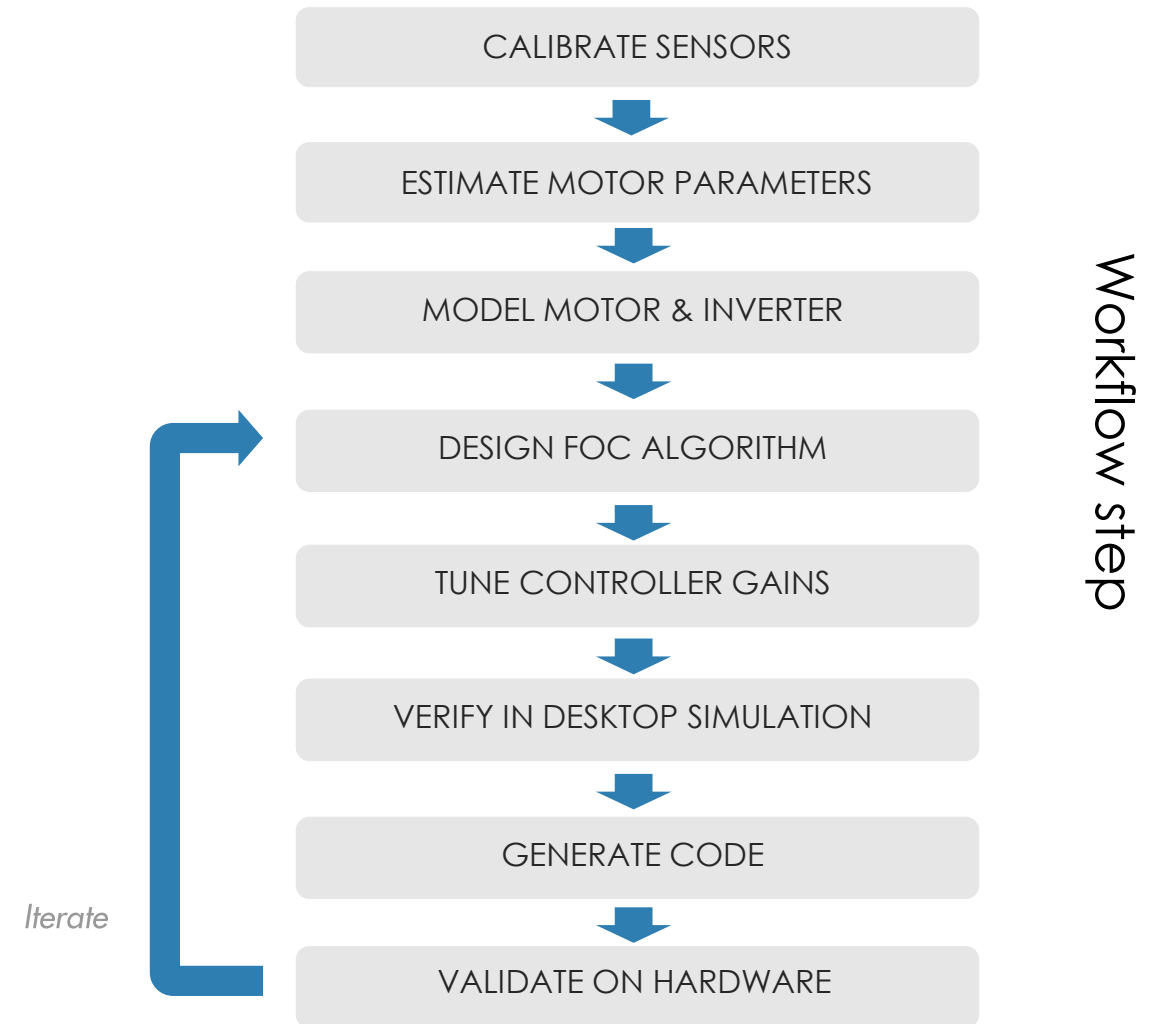
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



# Agenda

## From Desktop Simulation to Software Deployment

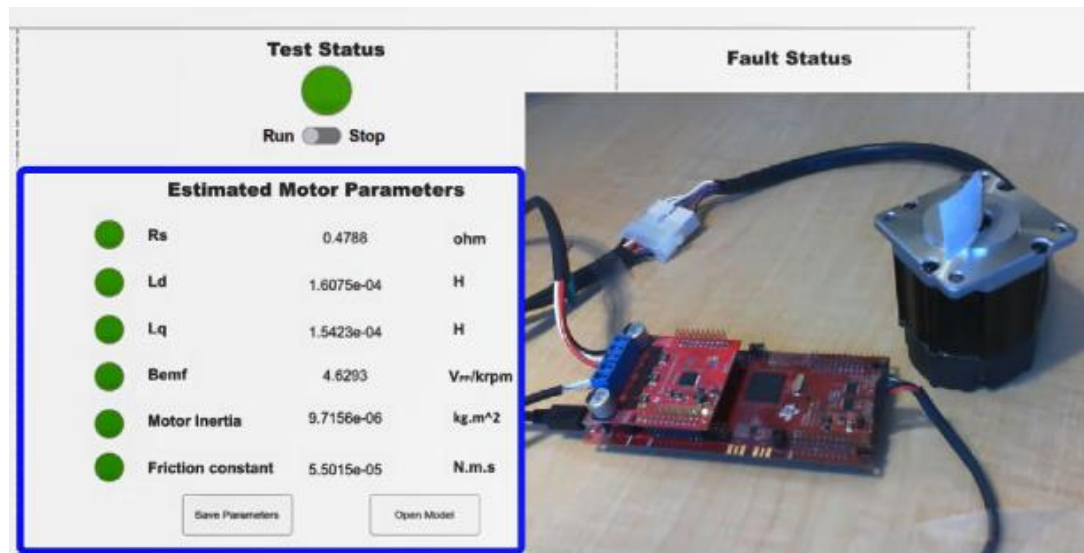
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



# Motor Parameters Estimation

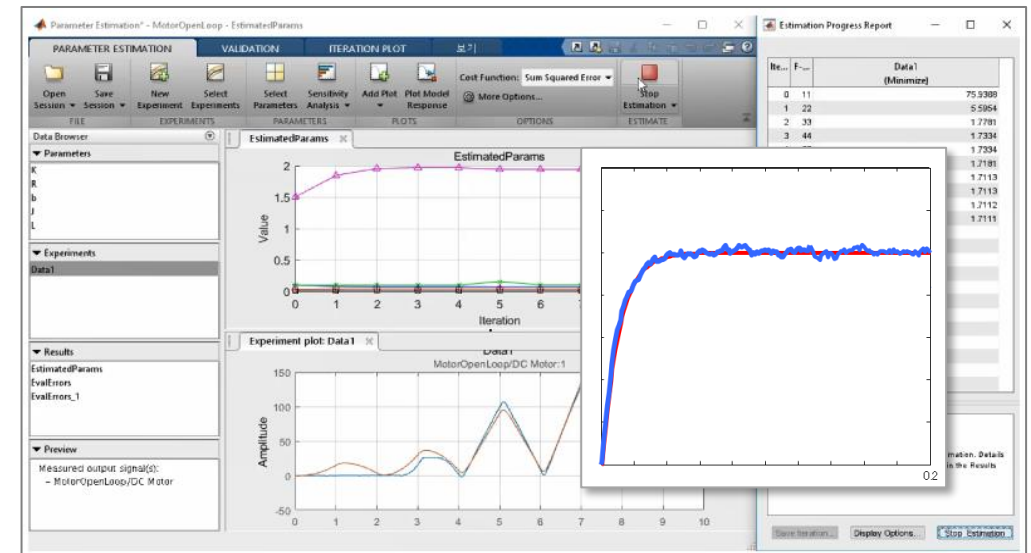
## Plant Modeling

Two types of parameter estimation methods:



Parameter Estimation with Instrumented Test

or



Parameter Estimation using Operation Data



# Other techniques to parameterize motor models

## Bonus

**PMSM Parameterization from Datasheet**

1. Pick torque and calculate efficiency at rated load (see code)
2. Create lookup table (see code)
3. Use for motor
4. Run Simulink, collect, export

**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](#)

From datasheet

Simscape Electrical

**Import IPMSM Flux Linkage Data from ANSYS Maxwell**

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

[Open Model](#)

From ANSYS Maxwell, JMAG, Motor-CAD FEA tools

Simscape Electrical

**Import IPMSM Flux Linkage Data from Motor-CAD**

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

[Open Model](#)

From dyno data

Powertrain Blockset

## 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
<a href="#">Step 1: Load and Preprocess Data</a>	Load and preprocess this nonlinear motor flux data from dynamometer testing or finite element analysis (FEA): <ul style="list-style-type: none"> <li><math>d</math>- and <math>q</math>- axis current</li> <li><math>d</math>- and <math>q</math>- axis flux</li> <li>Electromagnetic motor torque</li> </ul>
<a href="#">Step 2: Generate Evenly Spaced Table Data From Scattered Data</a>	Use the <code>gridfit</code> function to generate evenly spaced data. Visualize the flux surface plots.
<a href="#">Step 3: Set Block Parameters</a>	Set workspace variables that you can use for the Flux-Based PM Controller block parameters.

# Motor Parameters Estimation - Instrumented Test

## Plant Modeling

The screenshot displays the MATLAB/Simulink interface for the 'mcb\_param\_est\_host\_read' model. The interface is organized into several functional panels:

- Board Selection:** A dropdown menu is set to 'DRV8305 and F28379D Launchpad'.
- Communication Port:** A 'Serial Setup' dialog box is open, indicating that the COM port must be configured to match the board.
- Test Status:** A large grey circle indicates the test is currently in a 'Stop' state. Below it are 'Run' and 'Stop' buttons.
- Estimated Motor Parameters:** A table lists parameters with their current values and units:
 

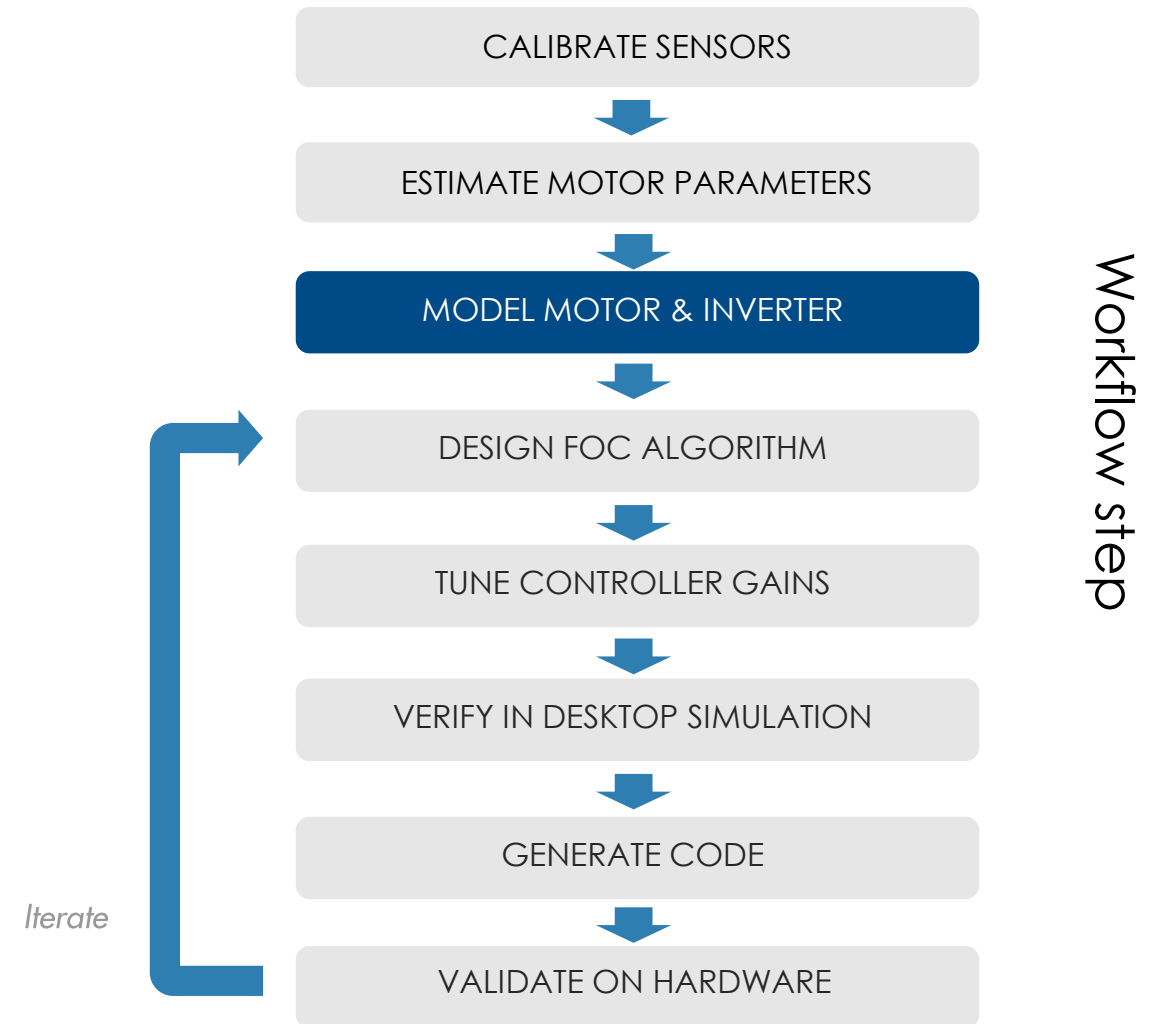
<input type="radio"/>	Rs	--	ohm
<input type="radio"/>	Ld	--	H
<input type="radio"/>	Lq	--	H
<input type="radio"/>	Bemf	--	Vpp/krpm
<input type="radio"/>	Motor Inertia	--	kg.m^2
<input type="radio"/>	Friction constant	--	N.m.s
- Signal Conditioning and Scaling:** A block diagram shows the 'Socoto2Signal' block connected to the 'SocotoSignal' block.
- Fault Status:** Three indicators are shown: 'Over Current', 'Under Voltage', and 'Serial communication', all currently inactive.
- Signal from Target:** A dropdown menu is set to 'Speed', and a corresponding signal block is visible.
- Required Inputs:** A list of input fields with values:
  - Nominal Voltage: 24 V
  - Nominal Current: 7.1 A (rms value)
  - Nominal Speed: 4000 rpm
  - Pole pairs: 4
  - Input DC Voltage: 20 V
  - Hall Offset: 0.2039 Per Unit Position

Additional information includes a note to press Ctrl+D to update the workspace, a 'Hall Offset' section with links to hardware models, and a 'Target Models' section with links to specific estimation models. The status bar at the bottom shows 'Ready', '95%' completion, and 'FixedStepDiscrete'.

# Agenda

## From Desktop Simulation to Software Deployment

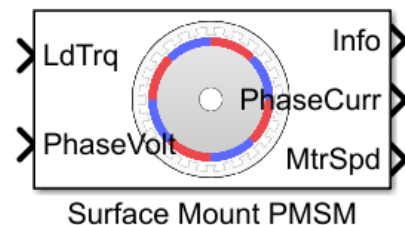
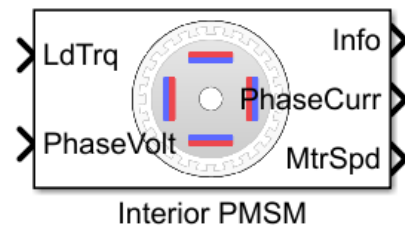
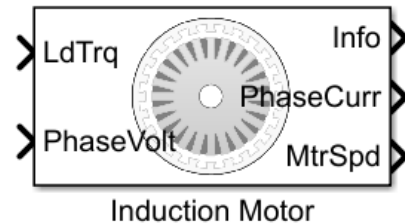
- **Plant modeling**
  - Sensors Calibration
  - Motor Parameters Estimation
  - **Motor and Inverter Model**
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



# Motor and Inverter Modeling

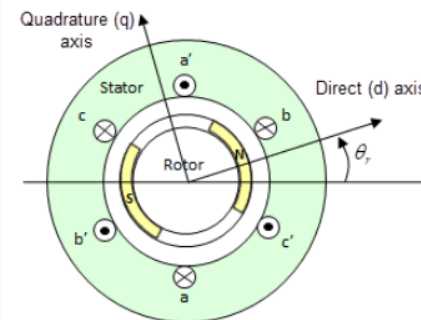
Choose the right level of fidelity

- Use linear lumped-parameter model shipped with Motor Control Blockset



### Motor Construction

This figure shows the motor construction with a single pole pair on the motor.



The motor magnetic field due to the permanent magnets creates a sinusoidal rate of change of flux with motor angle.

For the axes convention, the  $a$ -phase and permanent magnet fluxes are aligned when motor angle  $\theta_r$  is zero.

### Three-Phase Sinusoidal Model Electrical System

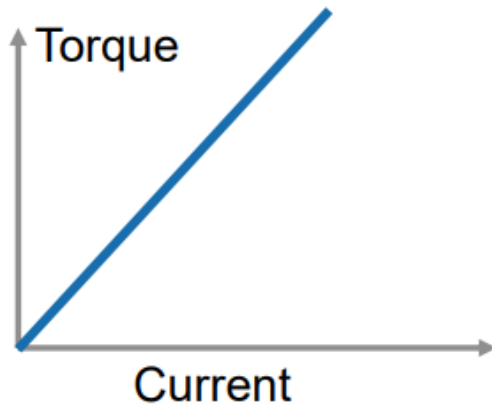
The block implements these equations, expressed in the motor flux reference frame (dq frame). All quantities in the motor reference frame are referred to the stator.

$$\omega_e = P\omega_m$$

$$\frac{d}{dt} i_d = \frac{1}{L_d} v_d - \frac{R}{L_d} i_d + \frac{L_q}{L_d} P\omega_m i_q$$

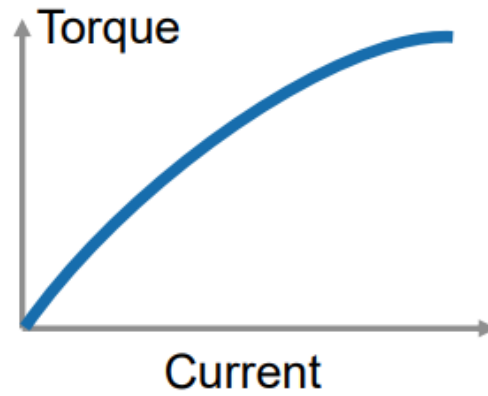
# Model Fidelity

## Plant Modeling



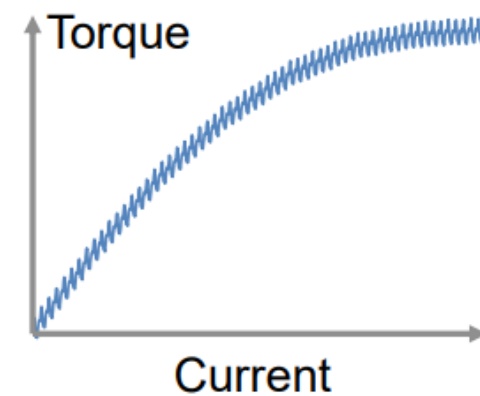
Linear Lumped Parameter

Motor Control Blockset  
Simscape Electrical



Saturation

Simscape Electrical



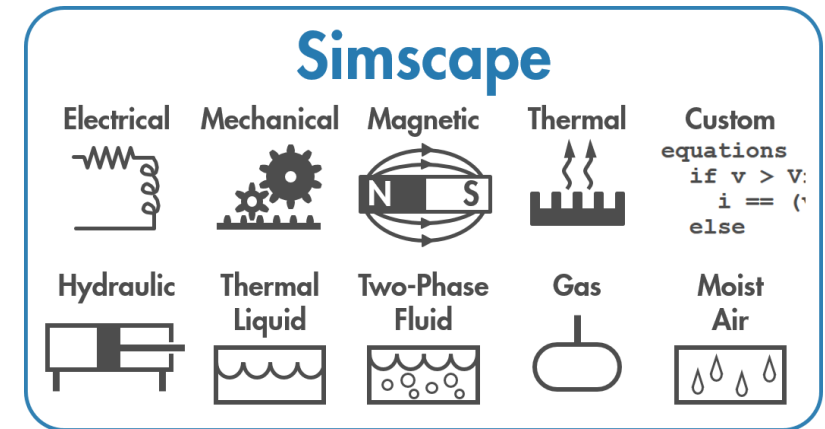
Saturation &  
Spatial Harmonics

Simscape Electrical

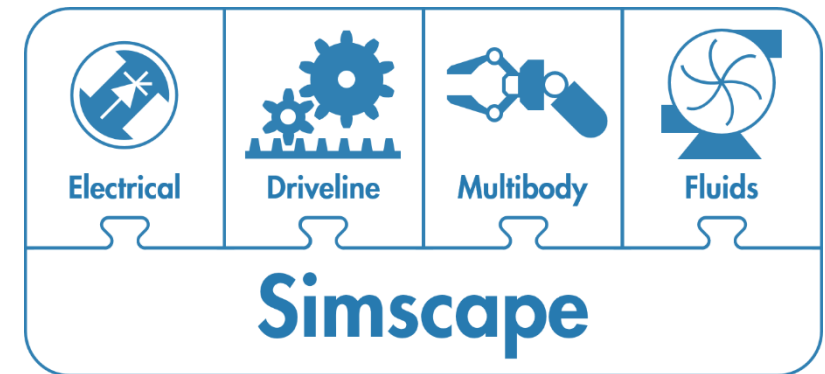
# Simscape Products

## Plant Modeling

- **Simscape platform**
  - Foundation libraries in many domains
  - Language for defining custom blocks
    - Extension of MATLAB
  - Simulation engine and custom diagnostics
  
- **Simscape add-on libraries**
  - Extend foundation domains with components, effects, parameterizations
  - Multibody simulation
  - Editing Mode permits use of add-ons with Simscape license only
  - Models can be converted to C code



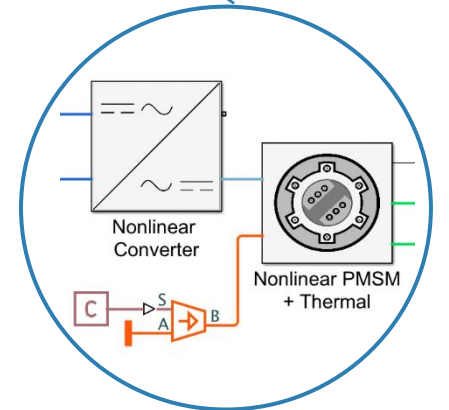
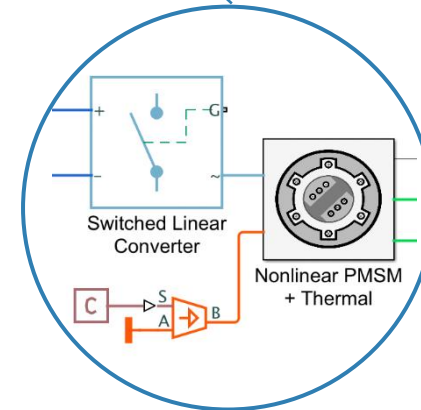
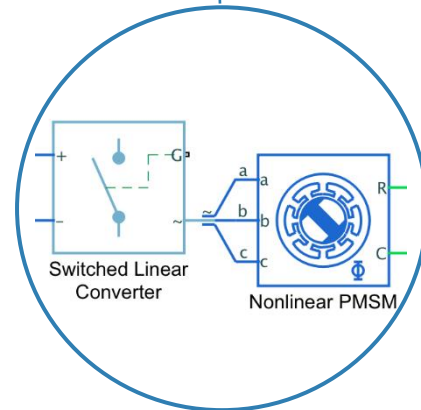
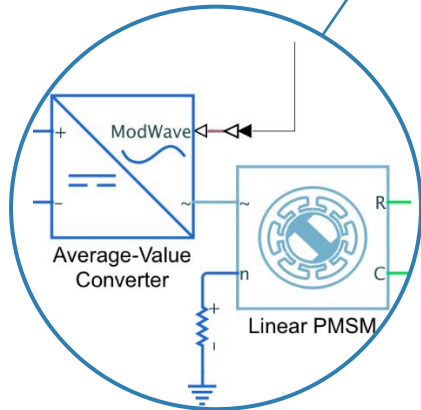
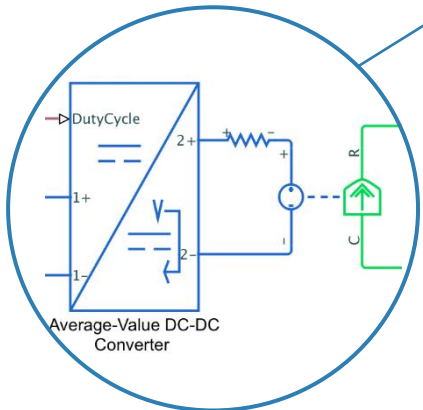
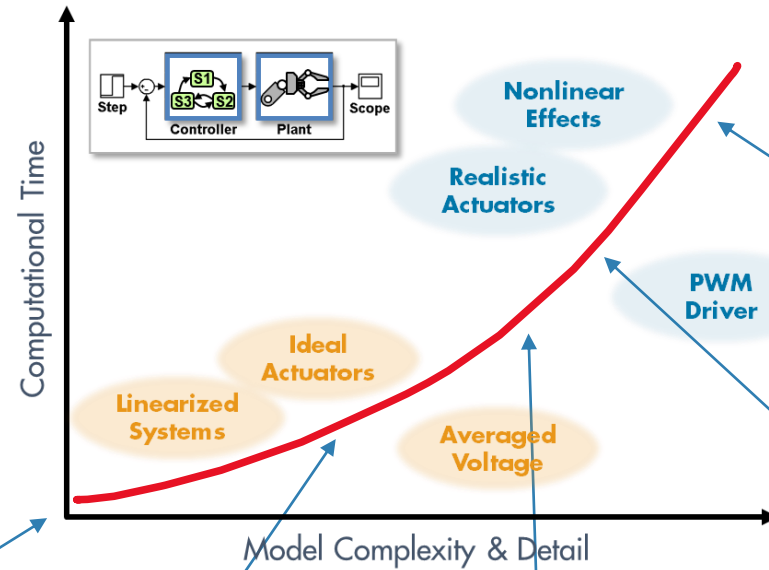
Simscape Foundation



Simscape Add-Ons

# Trade Off - Balance Model Fidelity vs Simulation Speed

## Plant Modeling

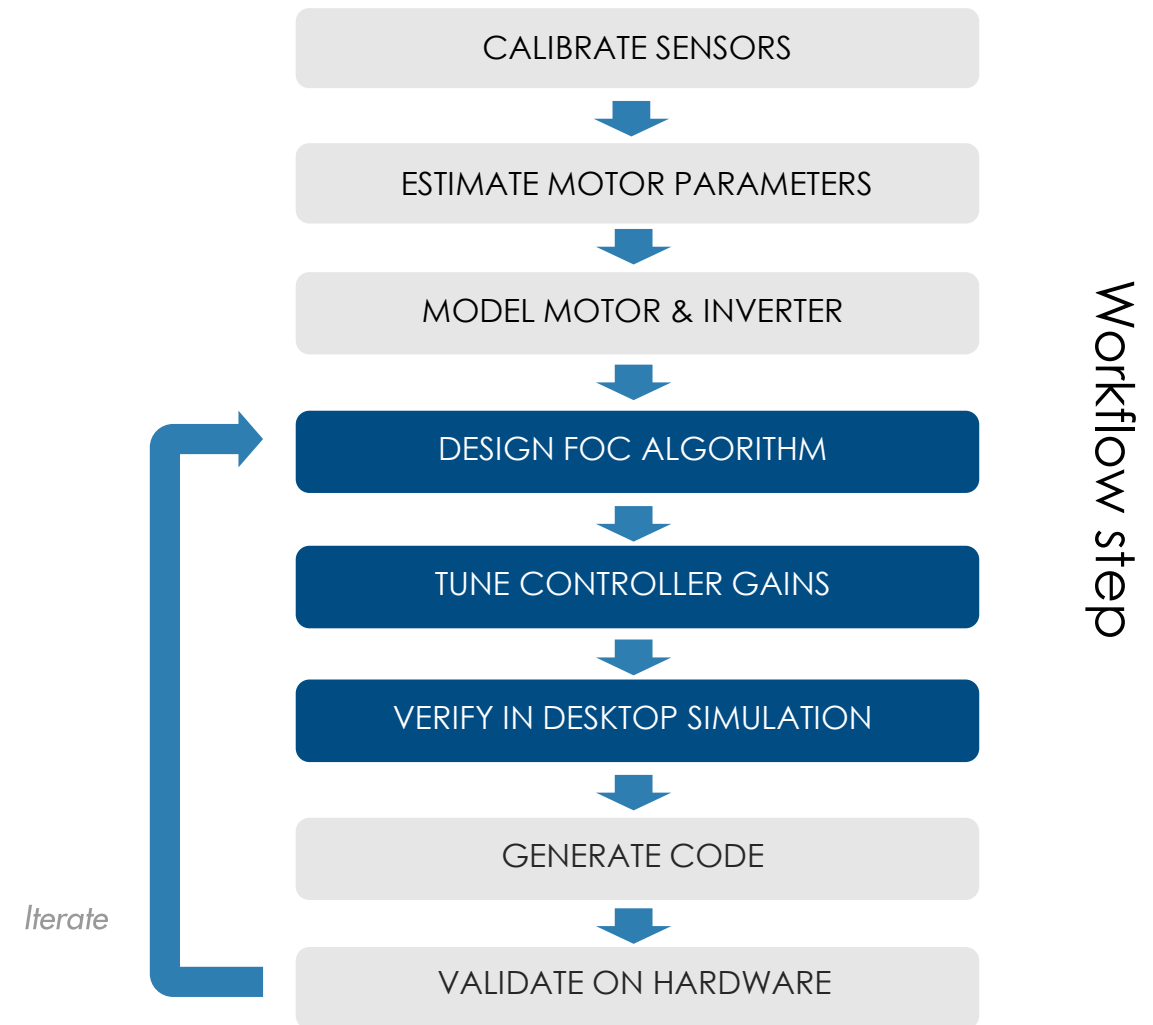




# Agenda

## From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



# Field-Oriented control

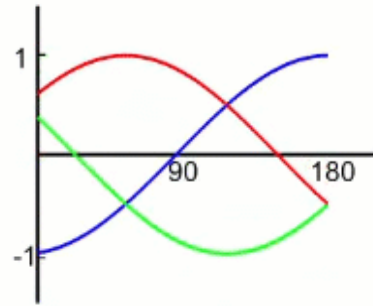
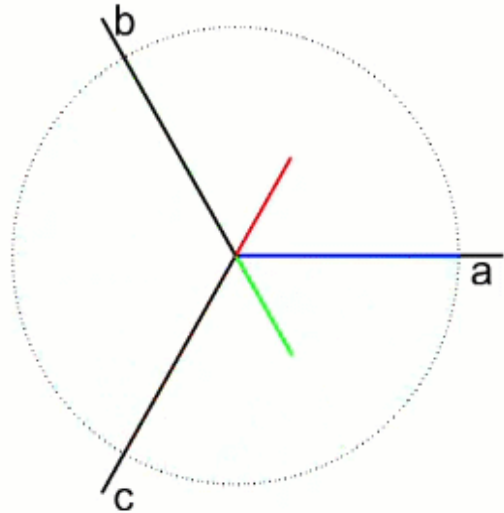
Controller tuning

Calibration

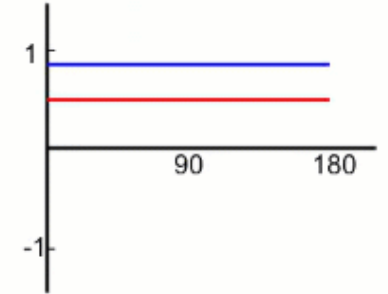
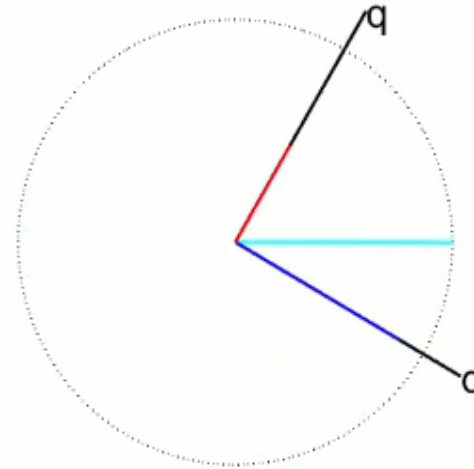
# Modeling Field-Oriented Control (FOC)

## A word about transforms

Measured current (A,B,C)  
in time domain



Current control (d, q)

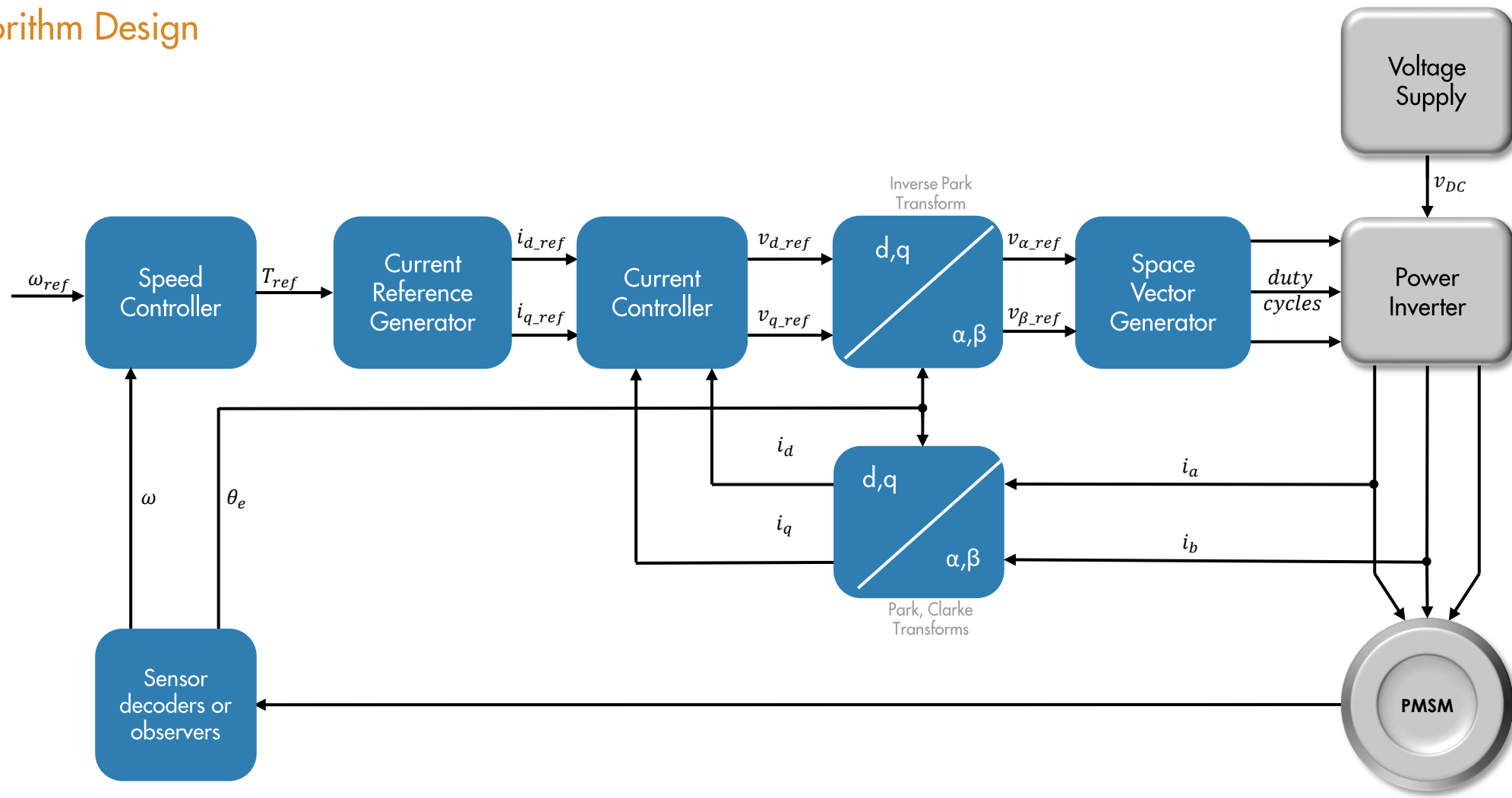


Clarke transform ( $abc \leftrightarrow \alpha\beta$ )

Park transform ( $\alpha\beta \leftrightarrow dq$ )

# Modeling Field-Oriented Control (FOC)

## Algorithm Design

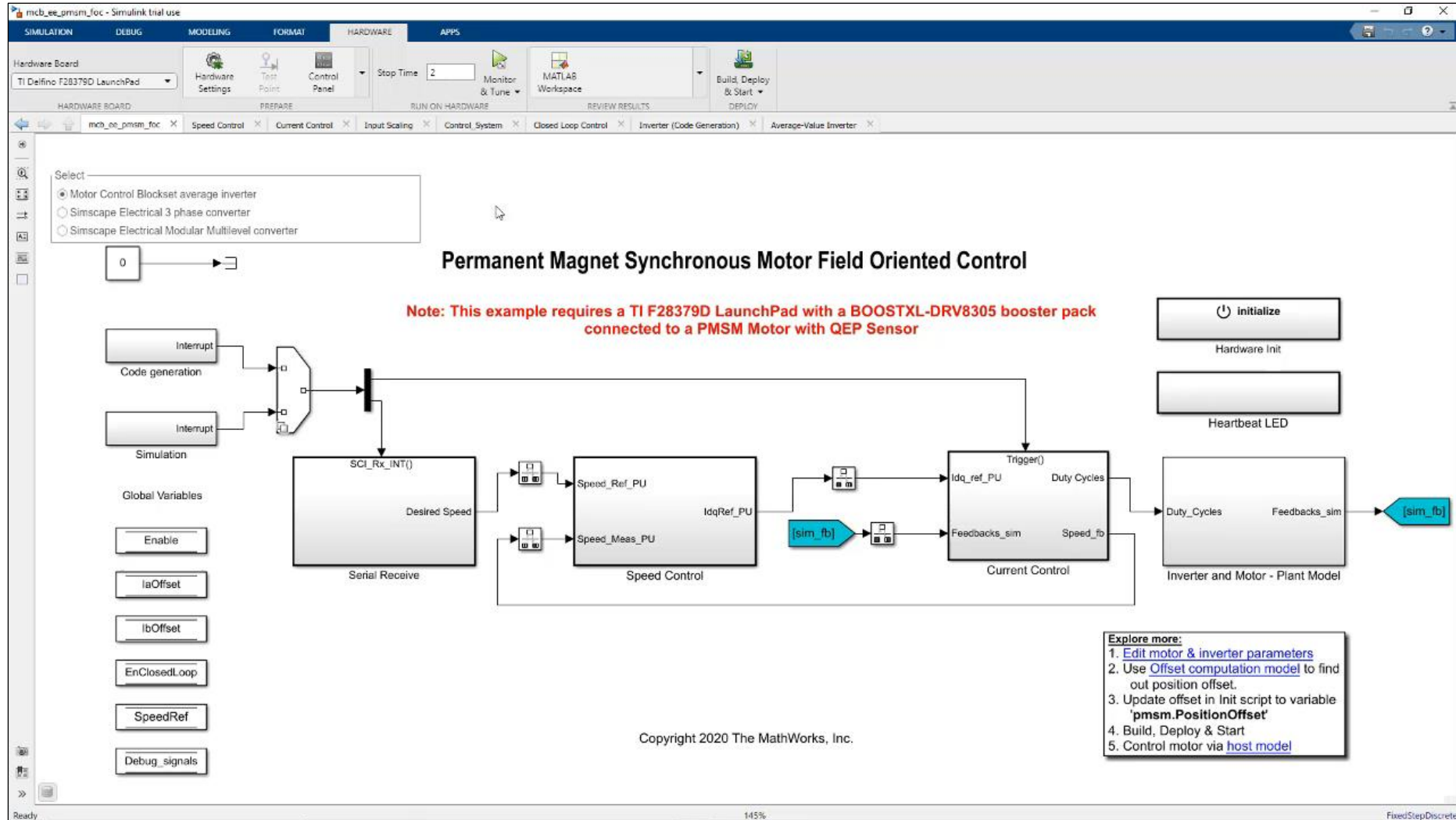


Control algorithm

Physical system

# Modeling Field-Oriented Control (FOC)

## Overview of the model



Field-Oriented control

**Controller tuning**

Calibration

# Modeling Field-Oriented Control (FOC)

```

%% 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

**Tune PI controllers by Using Field Oriented Control (FOC) Autotuner**

Computes the gain values of the PI controllers within the speed and current controllers by using the Field Oriented Control Autotuner block.

[Open Example](#)

FOC Autotuner

Motor Control Blockset

**Tune Field-Oriented Controllers Using SYSTEME**

Tune a field-oriented controller for an asynchronous machine in one simulation.

[Open Script](#)

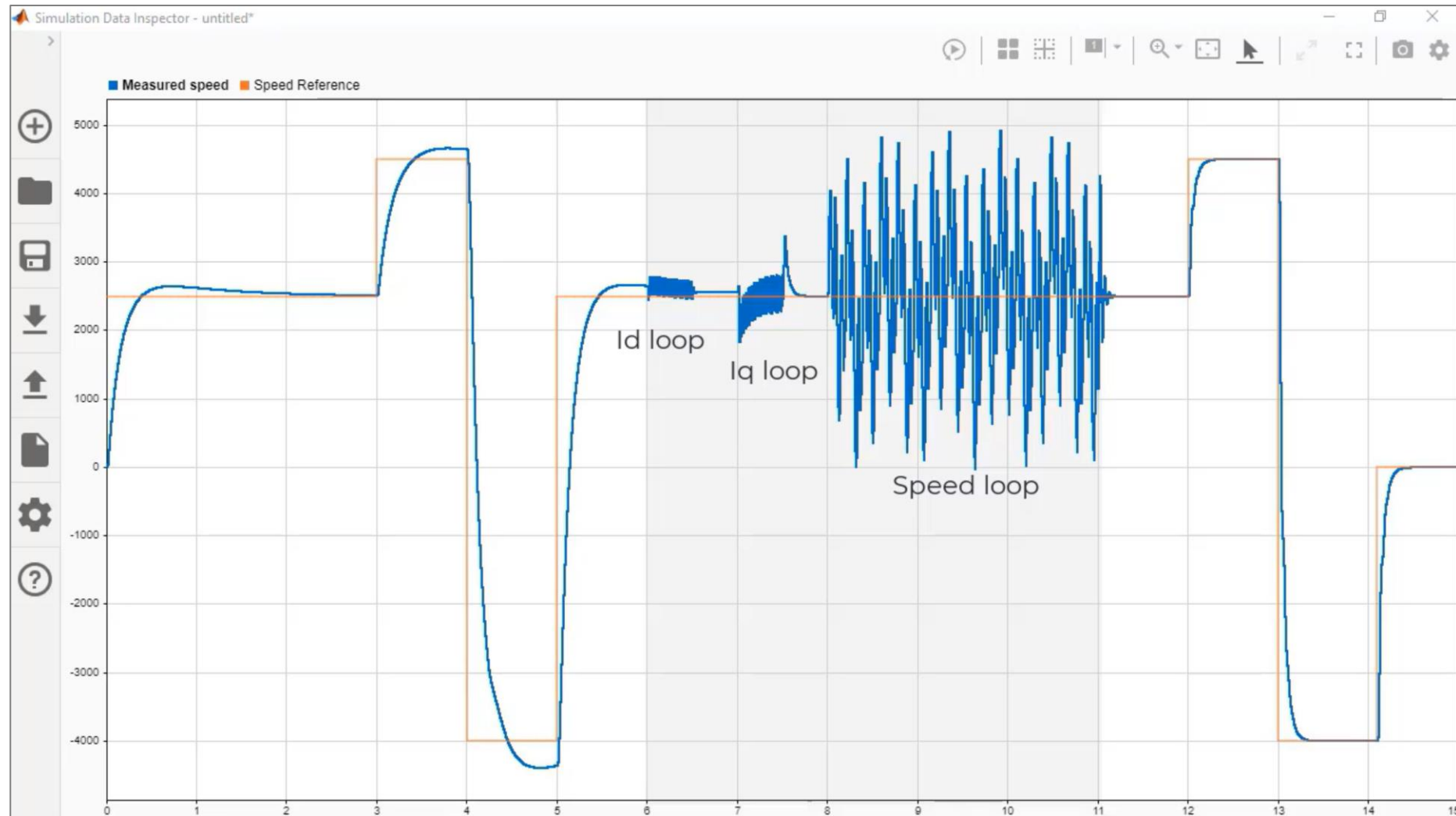
Classic Control Theory

Simulink Control Design



# Autotuning controller gains

## Overview of the workflow



Field-Oriented control

Controller tuning

**Calibration**

# FOC Circles tracing

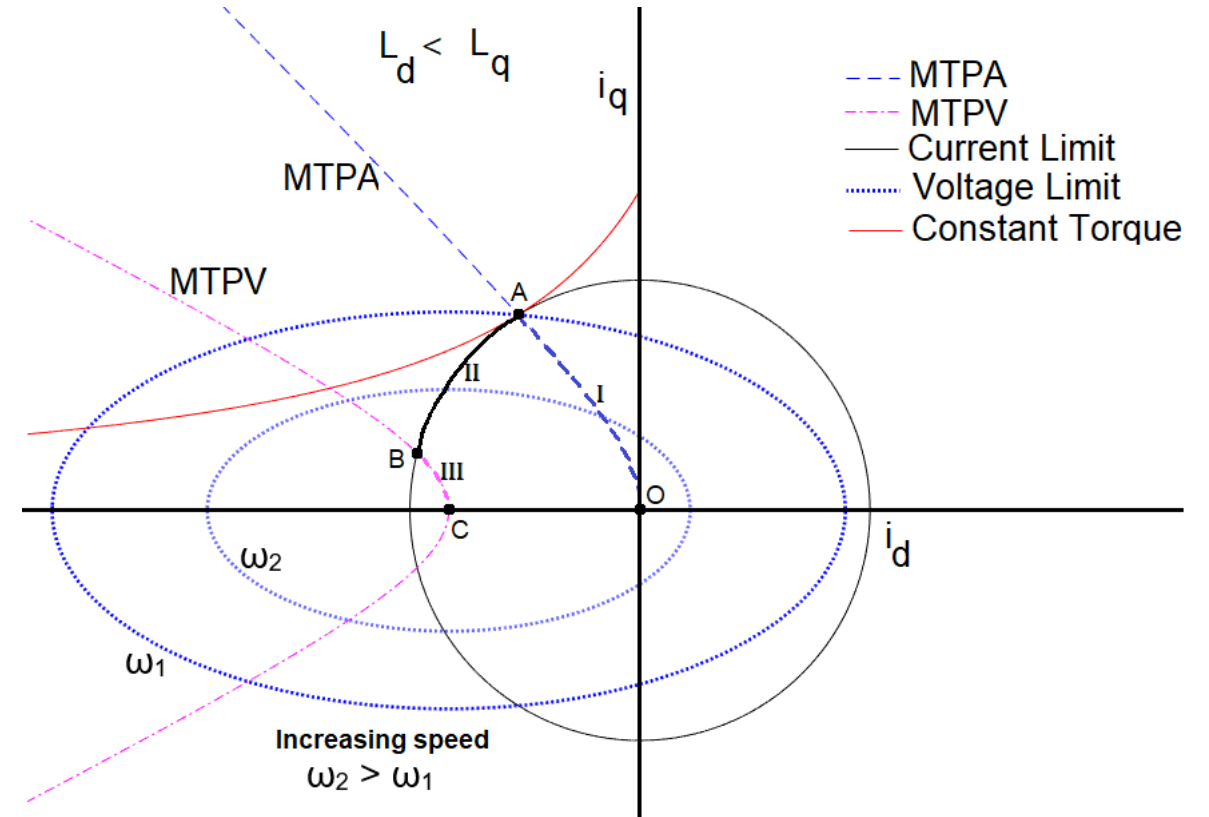
## mcbPMSMConstraintCurves

```

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

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

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



## Calibrating Optimal PMSM Torque Control with Field-Weakening Using Model-Based Calibration

By Dakai Hu, MathWorks

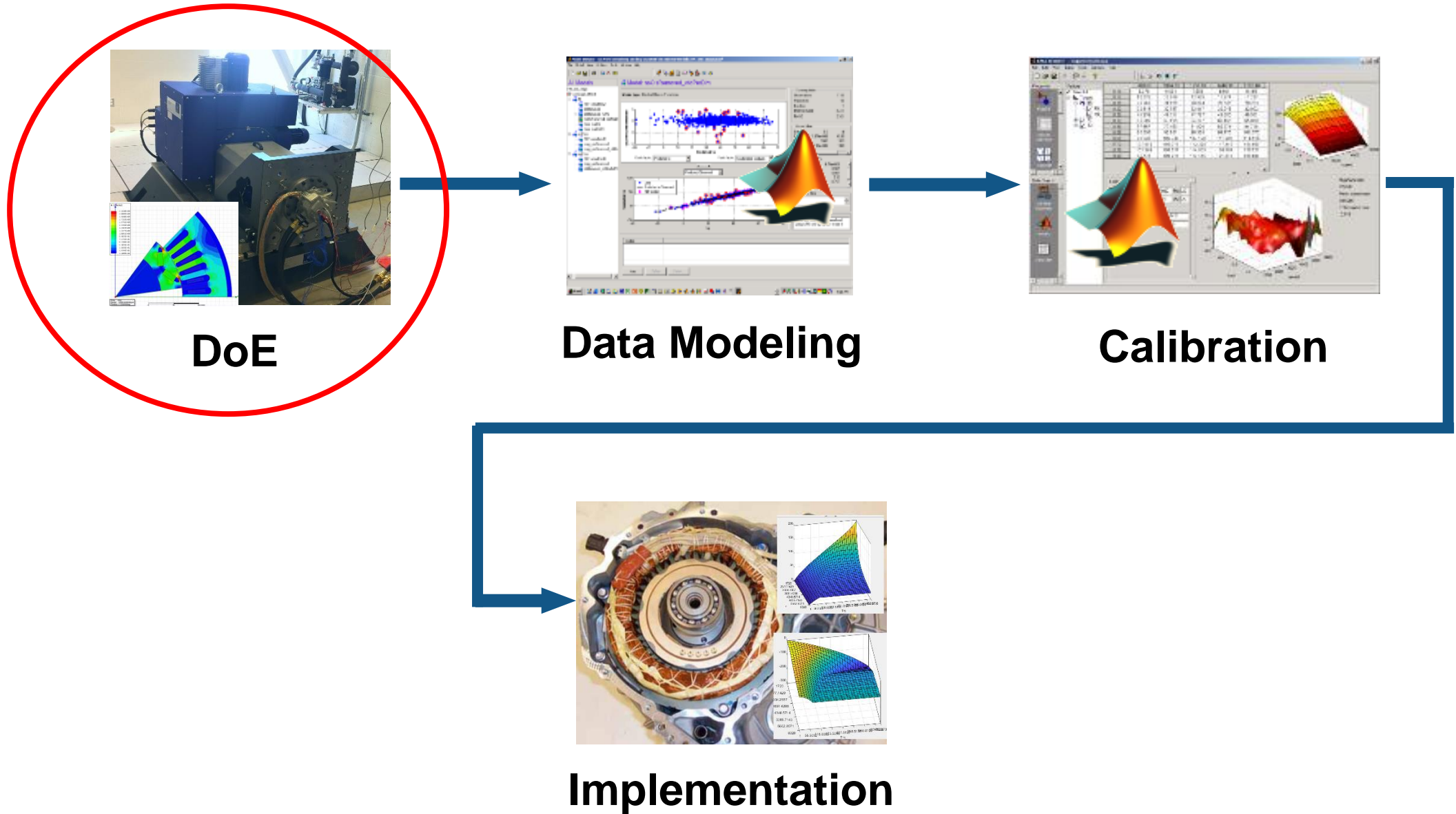
Permanent magnet synchronous motor (PMSM) calibration is an indispensable step in the design of high-performance electric traction drive controls. Traditionally, the calibration process involves extensive hardware dynamometer (dyno) testing and data processing, and its accuracy depends largely on the expertise of the calibration engineer.

Model-based calibration standardizes the PMSM calibration process, reduces unnecessary testing, and generates consistent results. It is an industry-proven, automated workflow that uses statistical modeling and numeric optimization to optimally calibrate complex nonlinear systems. It can be used in a wide range of applications and is well known for being adopted in internal combustion engine control calibration. When applied to e-motor control calibration, the model-based calibration workflow can help motor control engineers achieve optimal torque and field-weakening control for PMSMs.

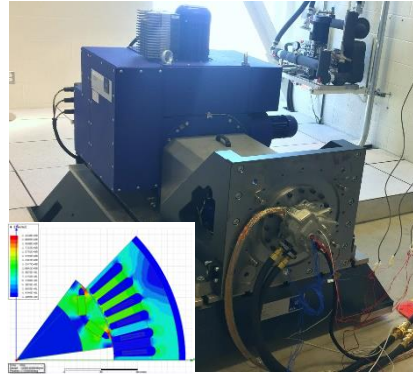
### PMSM Characterization and Calibration: Challenges and Requirements

PMSMs stand out from other types of e-motors because of their high efficiency and torque density. This is because the permanent magnets inside the machine can generate substantial air gap magnetic flux without external excitation. This special trait makes a PMSM an excellent candidate for both non-traction and traction motor drive applications.

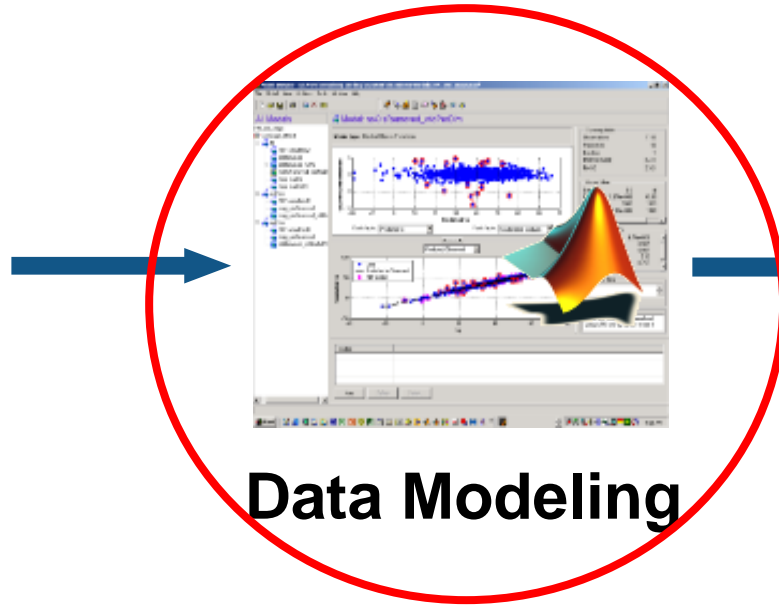
# Model-Based Calibration Workflow



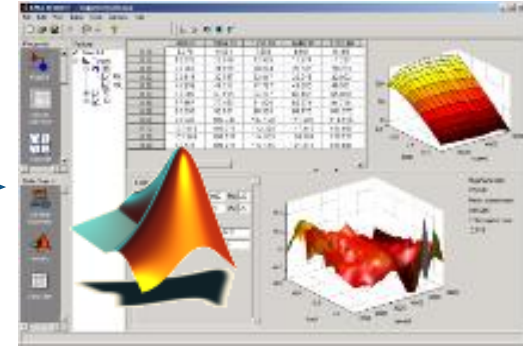
# Model-Based Calibration Workflow



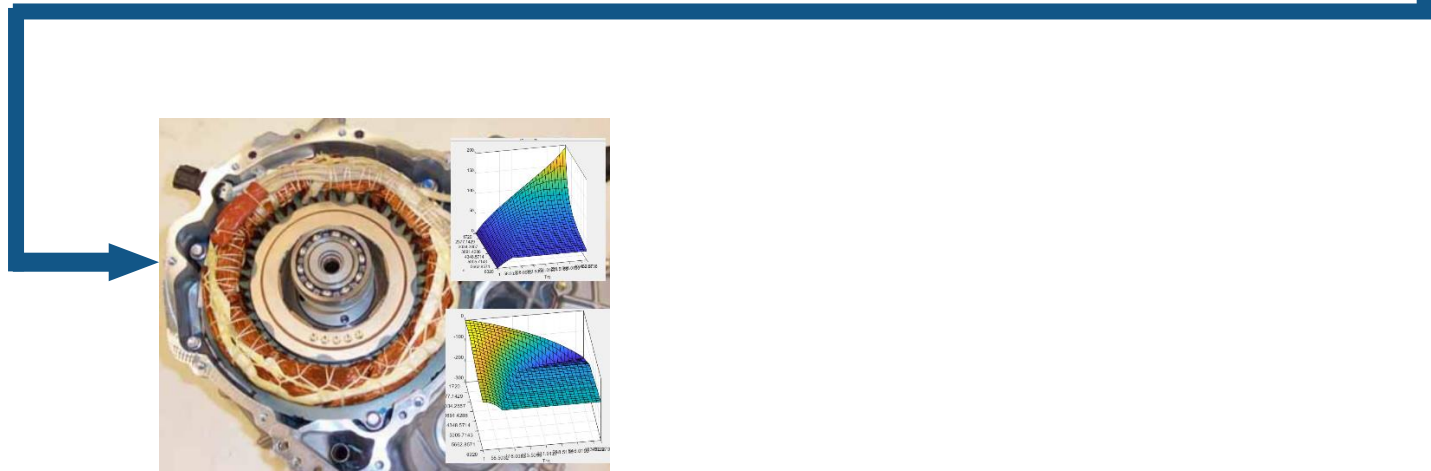
**DoE**



**Data Modeling**



**Calibration**



**Implementation**



# Fitting Models

Model Browser - C:\Temp\Work\Untitled

File Model View Outliers Window Help

Response Model: Vs

Te... 26 Select Test...

Model type: Gaussian Process Model (ARDSquaredExpon)

Alternative Local Models

Name	Observ...	Param...	Box-Cox	PRESS R...	RMSE	Best Model
GPM-ARDSqu...	262	141.098	1	0.042	0.013	✓

Response Surface

Plot: Surface

X-axis: Id

Y-axis: Trq

Name	Value	Tolerance
Id	-565.685424	Linked to X...
Trq	1.4.9683.24	Linked to Y-A...

Select Data Point...

Diagnostic Statistics

Residuals [-]

X-axis factor: Predicted ...

Vs - Test 26

Vs [-]

Trq [-]

Id [-]

RMSE Plots

RMSE Plots - double-click a point to change test.

RMSE

Test Number

X-axis factor: Test Number

Y-axis factor: RMSE

Validation

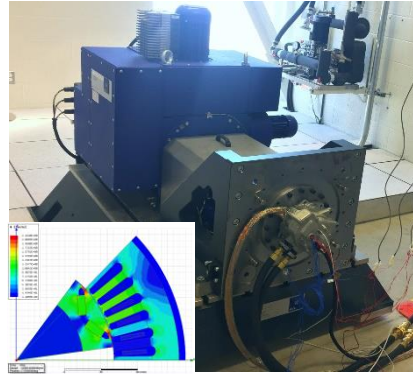
Pooled statistics

Local RMSE	8.616e...
Validation ...	

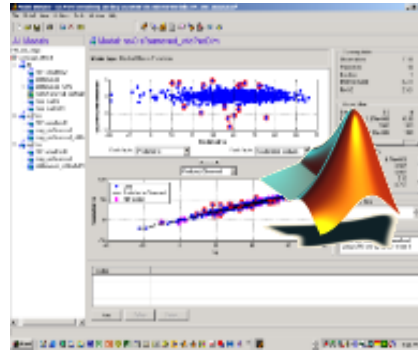
Test notes

Test number color: █ Set Color...

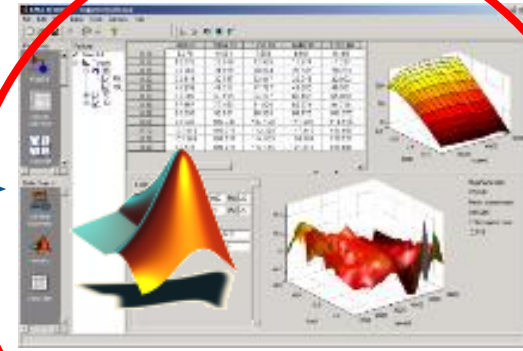
# Model-Based Calibration Workflow



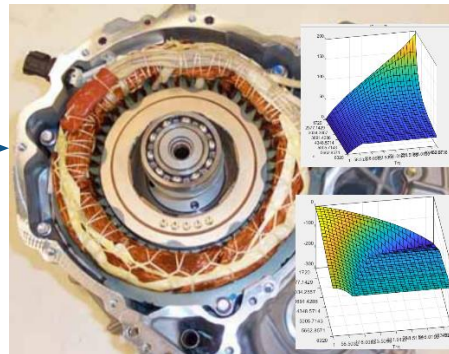
**DoE**



**Data Modeling**



**Calibration**

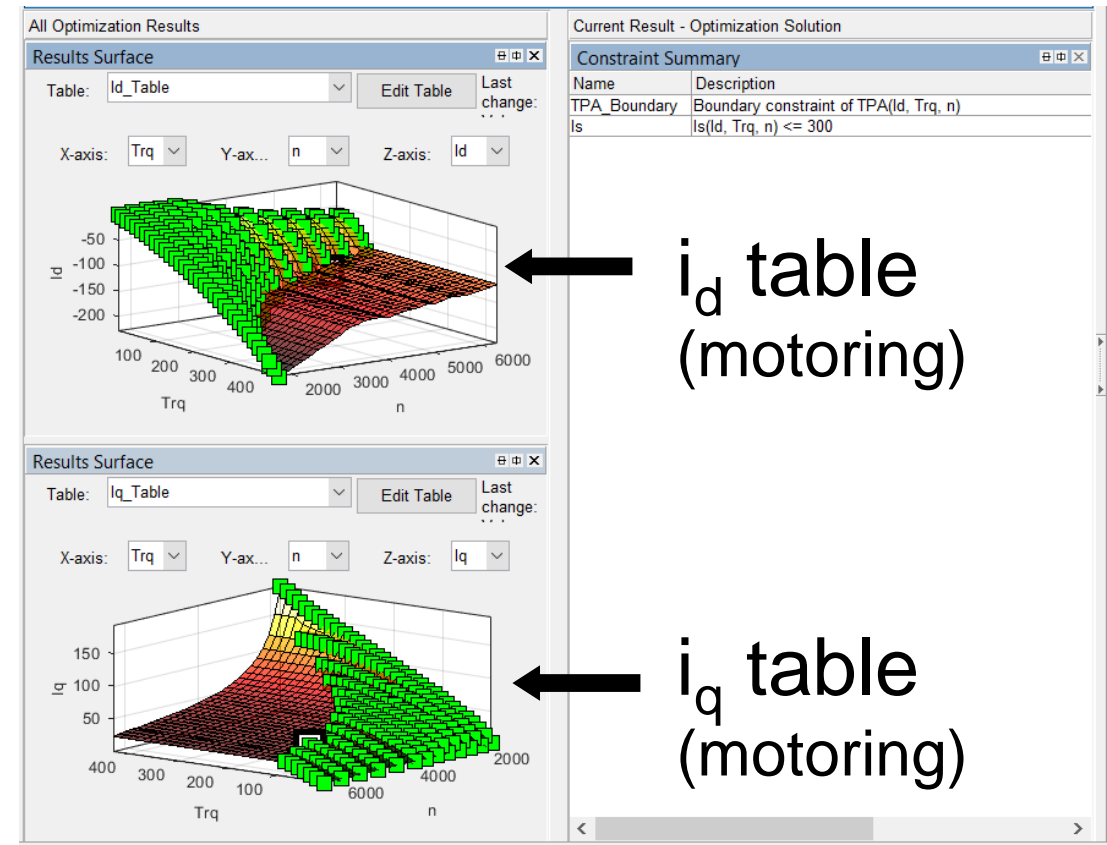
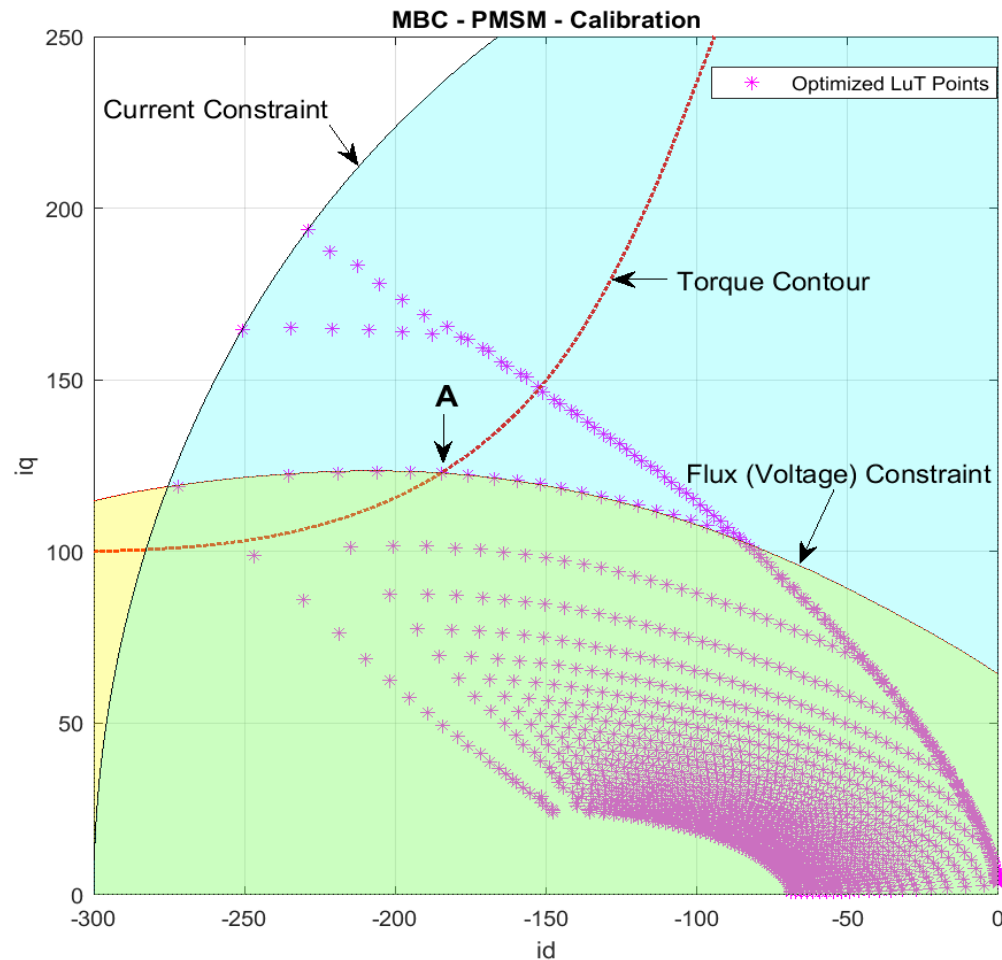


**Implementation**

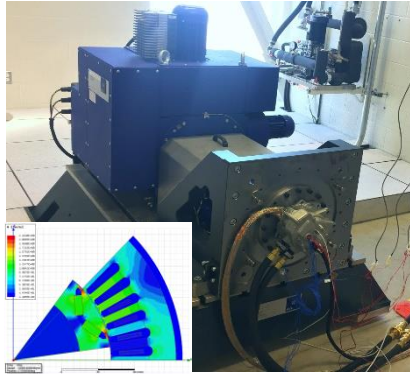


# Calibration

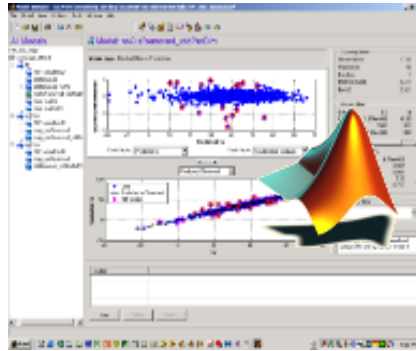
Goal : find the best ( $i_d$ ,  $i_q$ ) operating points that can achieve pre-set optimization objective while satisfying certain physical constraints.



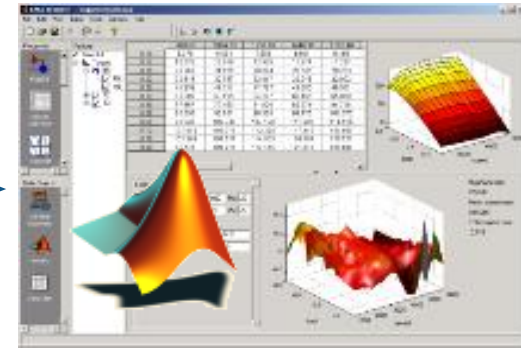
# Model-Based Calibration Workflow



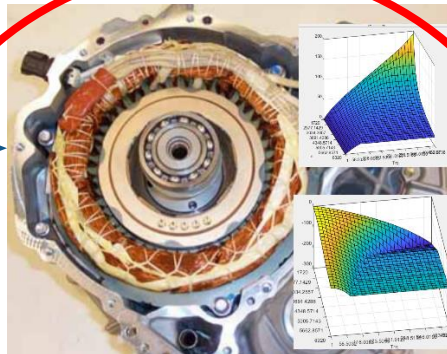
**DoE**



**Data Modeling**

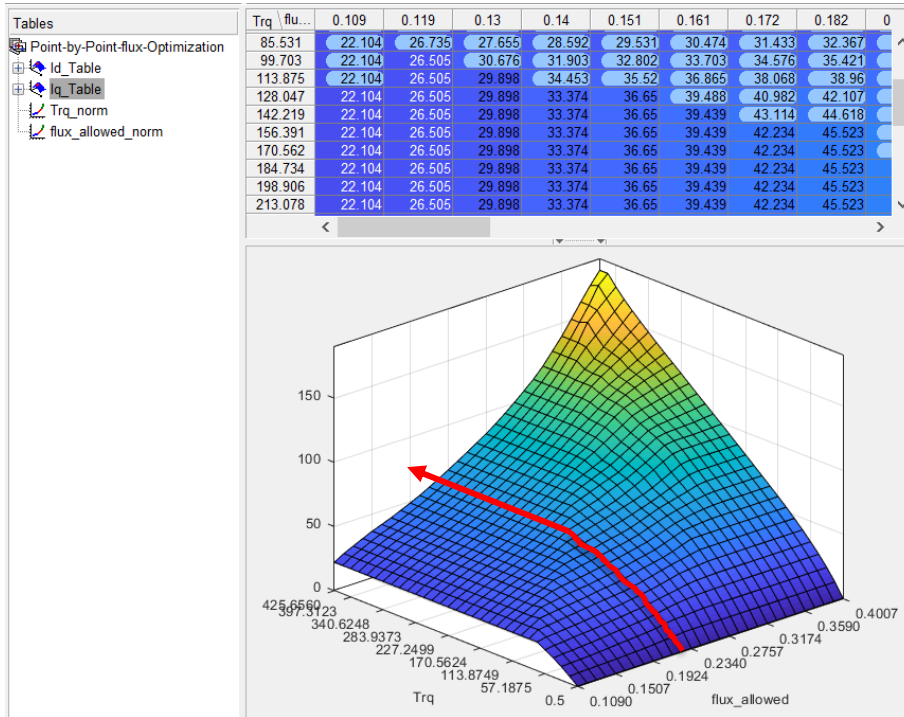


**Calibration**

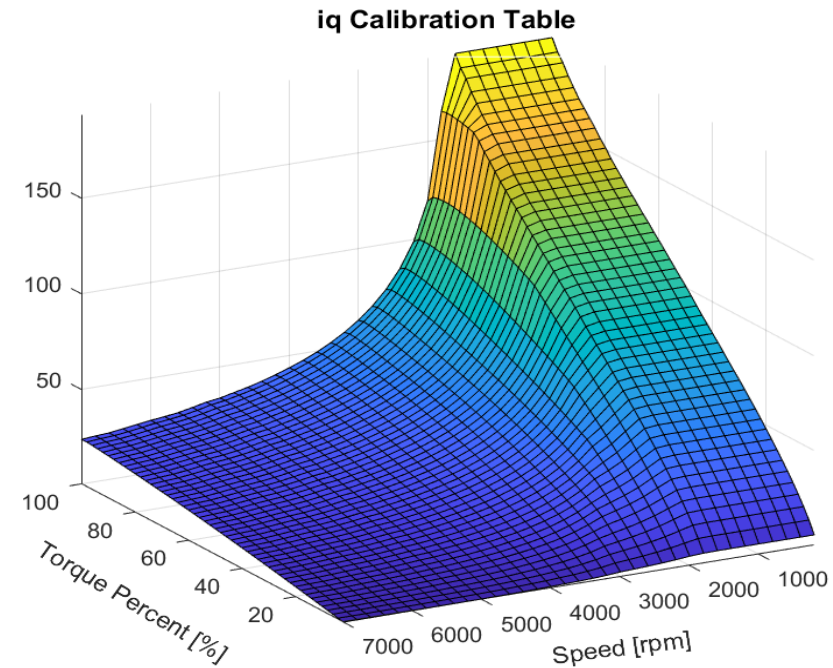


**Implementation**

# Calibration Results – Fill Calibration Tables



$i_q$  table (after clipping max torque)

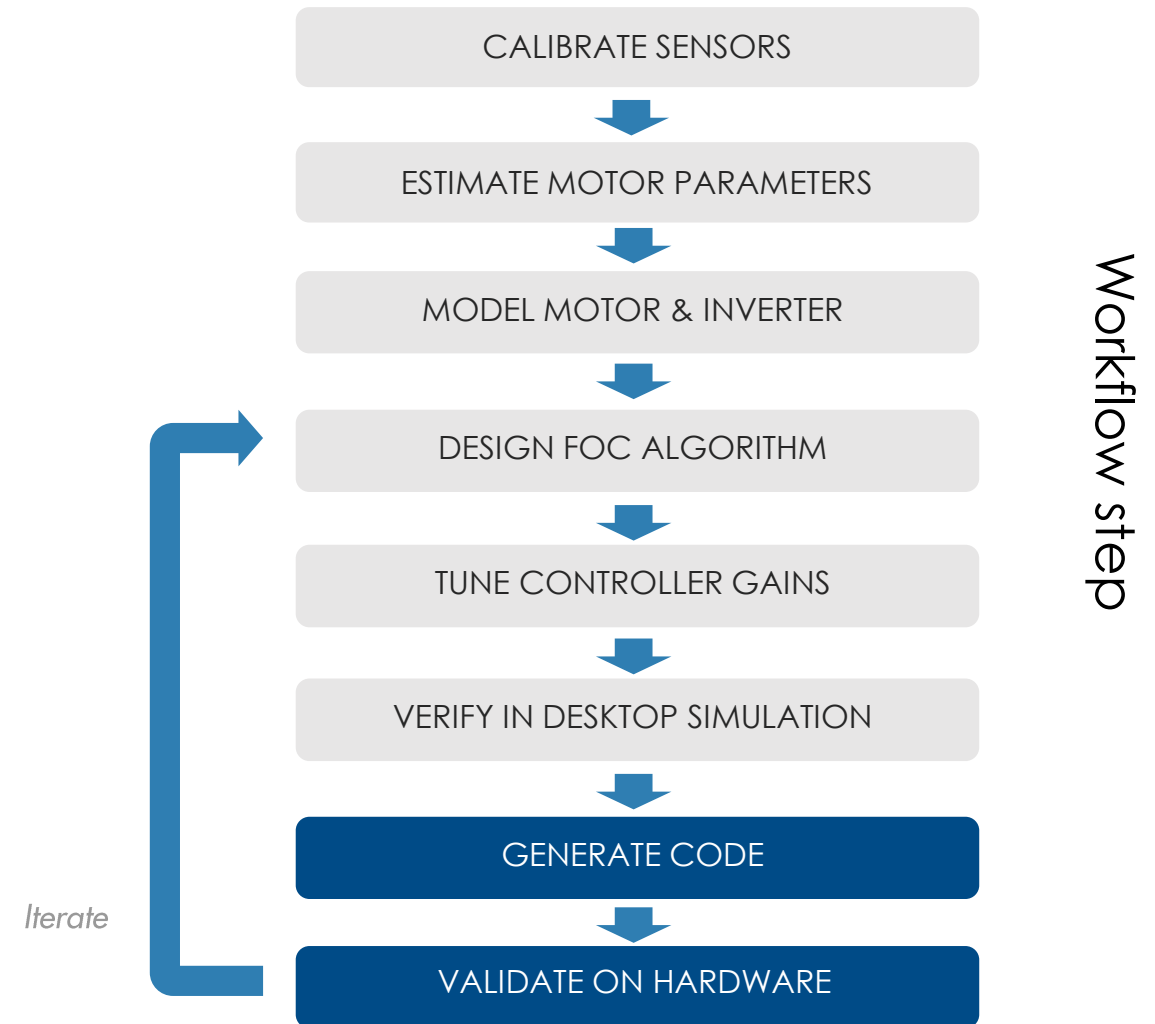


$i_q$  table (based on percentage of max torque)

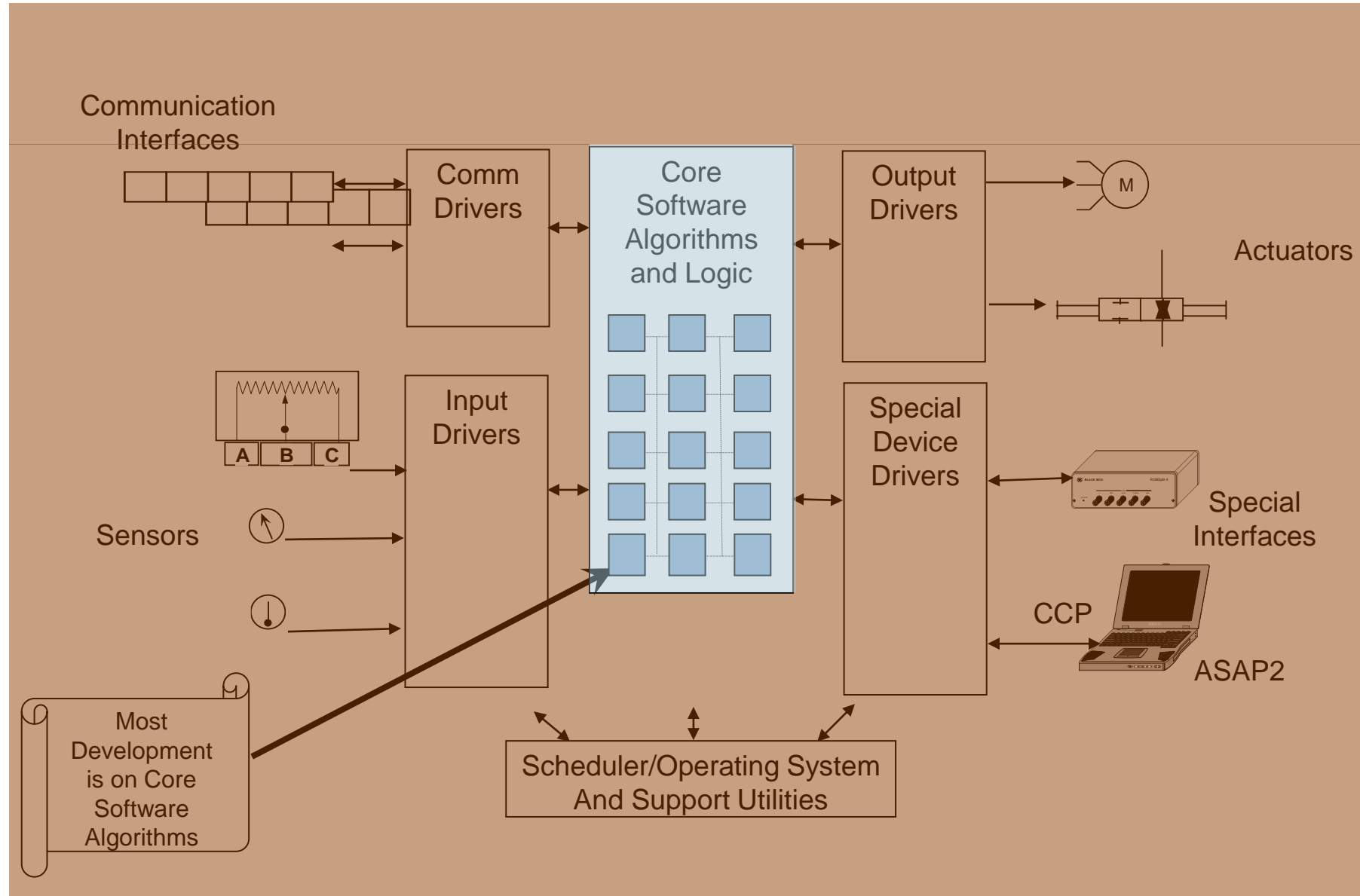
# Agenda

## From Desktop Simulation to Software Deployment

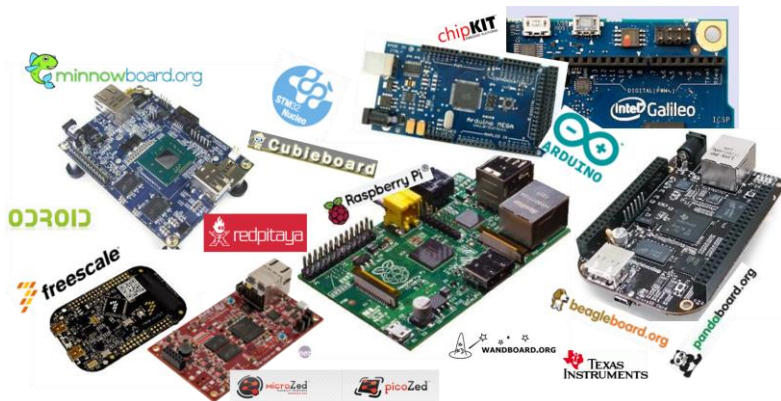
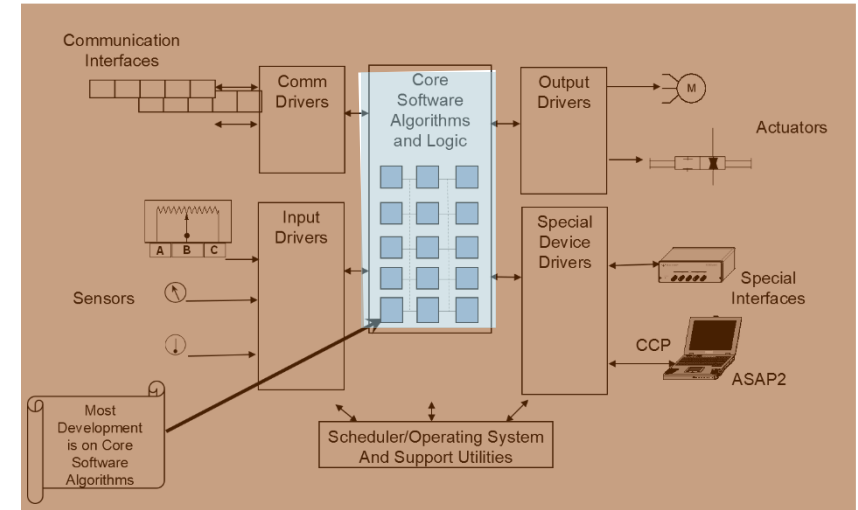
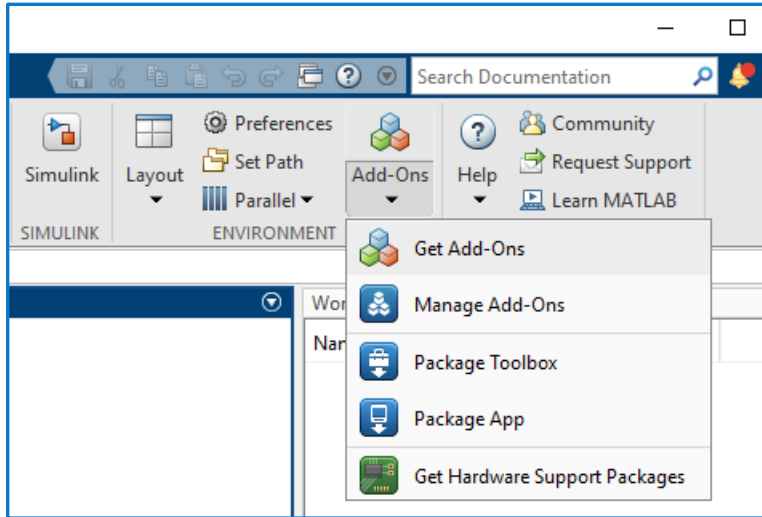
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Controller tuning
  - Calibration
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test



# Simple Embedded Software Architecture



# Custom Target, Hardware Support Package or Specialized toolbox



MATLAB Central ▾ Files Authors My File Exchange ▾ Publish About



## Embedded Coder Support Package for STMicroelectronics STM32 Processors

by MathWorks Embedded Coder Team **STAFF**

Generate code optimized for STMicroelectronics STM32 Processor based boards

Overview Reviews (24) Discussions (105)

Embedded Coder® Support Package for STMicroelectronics STM32 Processors enables users to build, load, and run Simulink models on STM32 devices using two separate workflows included in this support package. Any STM32F4xx, STM32F7xx, STM32G4xx and single core STM32H7xx family processor based boards are supported using STM32CubeMX-generated peripheral configurations and few ST Discovery boards are supported using built-in peripheral configurations.

Hardware Support Packages: <https://www.mathworks.com/hardware-support/home.html>



# C2000 Microcontroller Blockset

## Supported devices:

- F2802x/3x/5x/6x/07x/004x
- F2833x/32x/37xS/37xD/38xS/38xD
- Fixed-point F280x/1x
- F280013x / F280015x



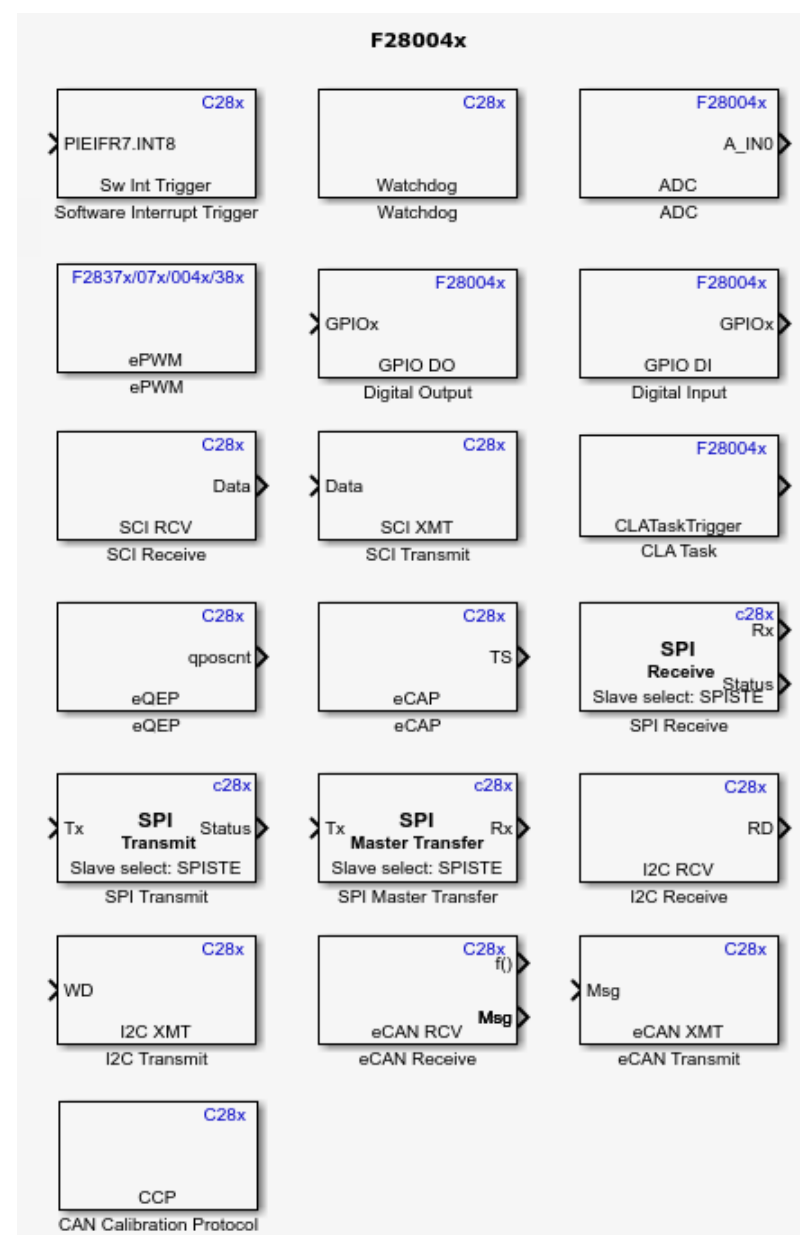
F28379D LaunchPad

## Scheduling the generated code:

- Periodic tasks
- Idle tasks
- Interrupts (Hardware, Software)
- Advanced concepts:
  - Pre-emptive rate-monotonic scheduler
  - Base rate interrupt replacement
  - Peripheral triggers (launch A/D conversion from PWM)
  - Running on the CLA
  - Loading in Flash, running in RAM
  - Using DMA

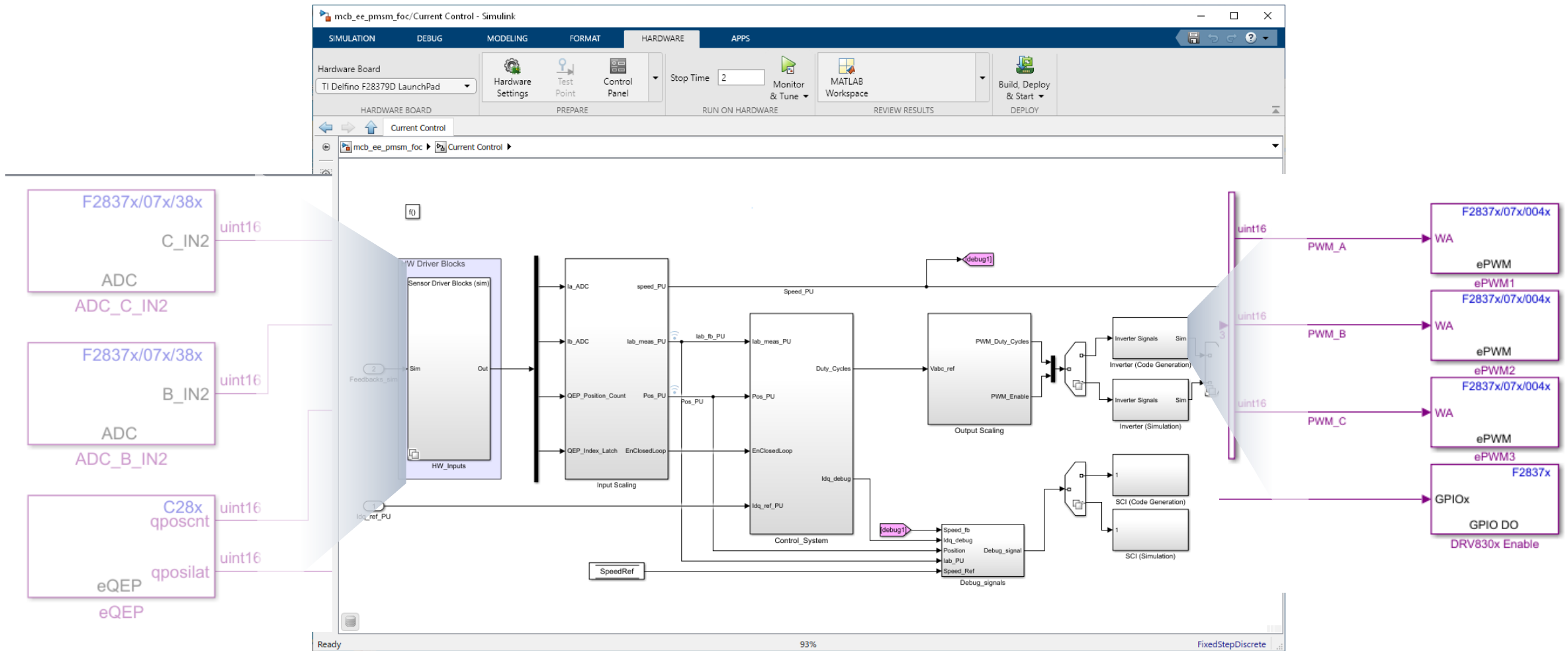
# Supported TI C2000 drivers

- ADC, AIO, Comparator,
- GPIO, eQEP, ePWM, eCAP,
- eCAN, I2C, SCI, SPI, LIN
- Watchdog, DMA
  
- Motor control position sensing
  - Optical encoder (using eQEP)
  - Hall sensors (using eCAP)
  - Sensorless (using SMO)





# Prepare the Model for Code Generation Using Supported TI C2000 Drivers Blocks



# Deployment on the Target

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

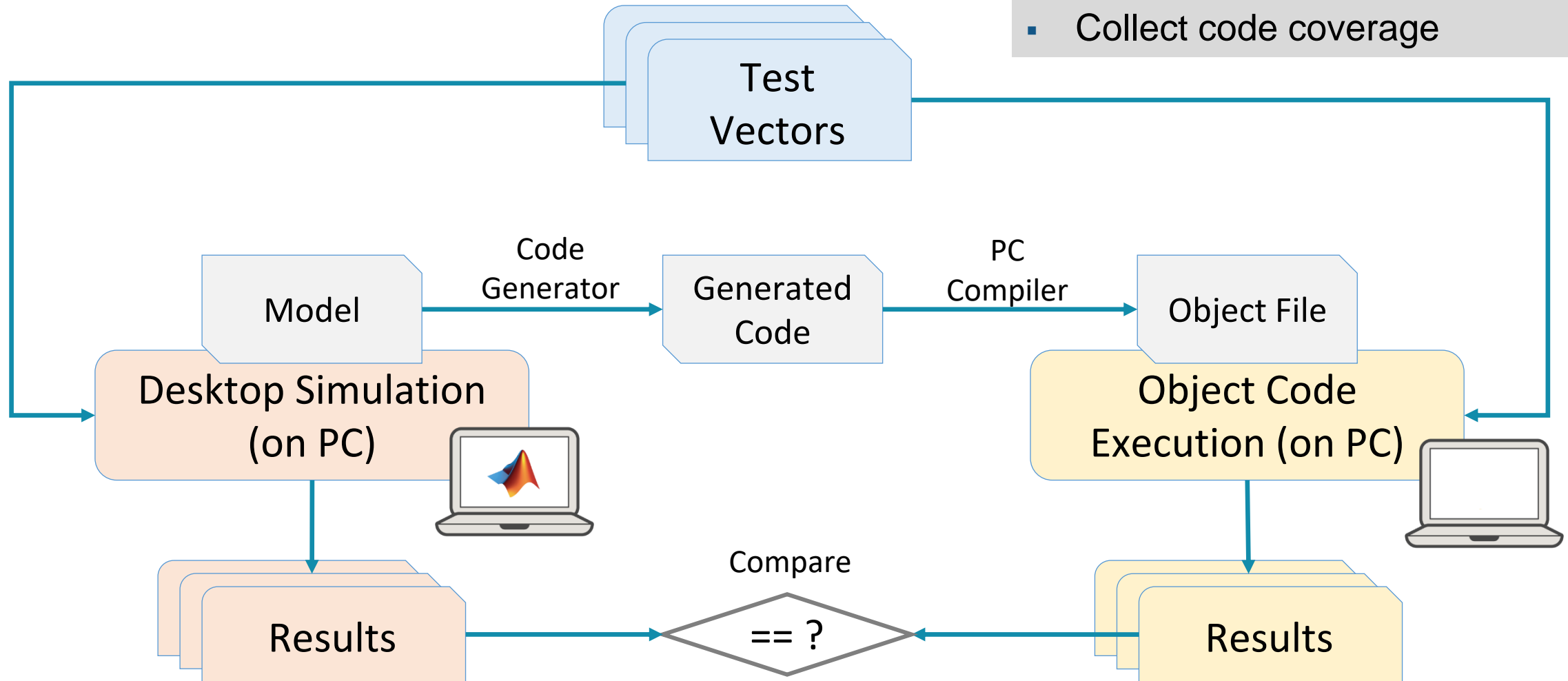
The screenshot displays the Simulink environment for a 'Permanent Magnet Synchronous Motor Field Oriented Control' model. The model is divided into two main sections: 'Speed Control' and 'Current Control'. The 'Speed Control' section includes blocks for 'Speed\_Ref\_PU' and 'Speed\_Meas\_PU', which feed into an 'IdqRef\_PU' block. The 'Current Control' section includes 'Idq\_ref\_PU', 'Feedbacks\_sim', and 'Speed\_fb' blocks. A 'Trigger()' block is also present, which outputs 'Duty Cycles' and 'Speed\_fb'. The model is connected to a hardware target, as indicated by the 'Hardware' tab in the top toolbar.

Below the model, a 'Timing Legend' window is visible, showing the following data:

Discrete	Period
	25.0000e-006
	50.0000e-006
	500.0000e-006

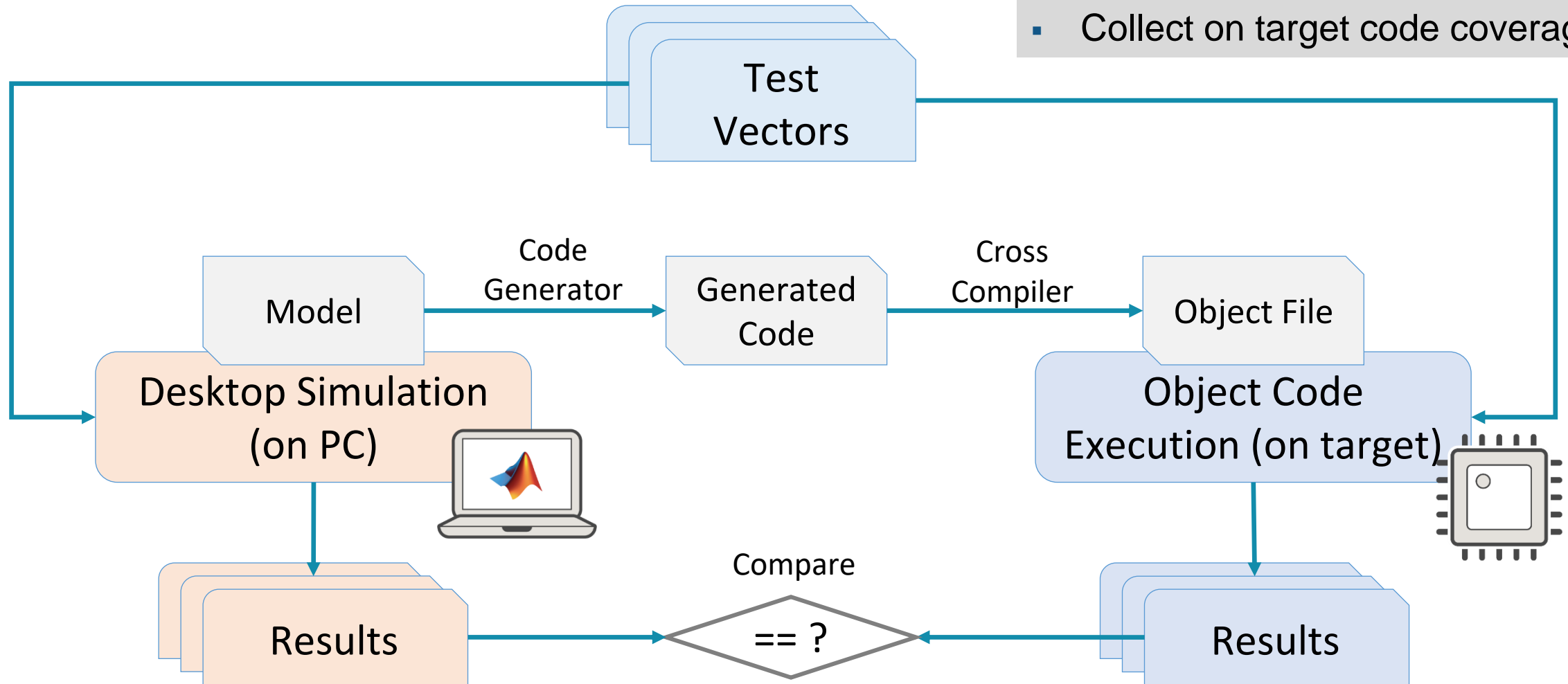
# Software-In-the-Loop (SIL) Testing

- Show equivalence, model to code
- Assess code execution time
- Collect code coverage



# Processor-In-the-Loop (PIL) Testing

- Verify numerical equivalence
- Assess target execution time
- Collect on target code coverage



# 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 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	
[+] <a href="#">Current_initialize</a>	2260	2260	1365	1365	1	
<a href="#">Current_step [5e-05 0]</a>	5135	5067	5135	5067	10001	
<a href="#">Current_terminate</a>	540	540	540	540	1	

### 3. CPU Utilization

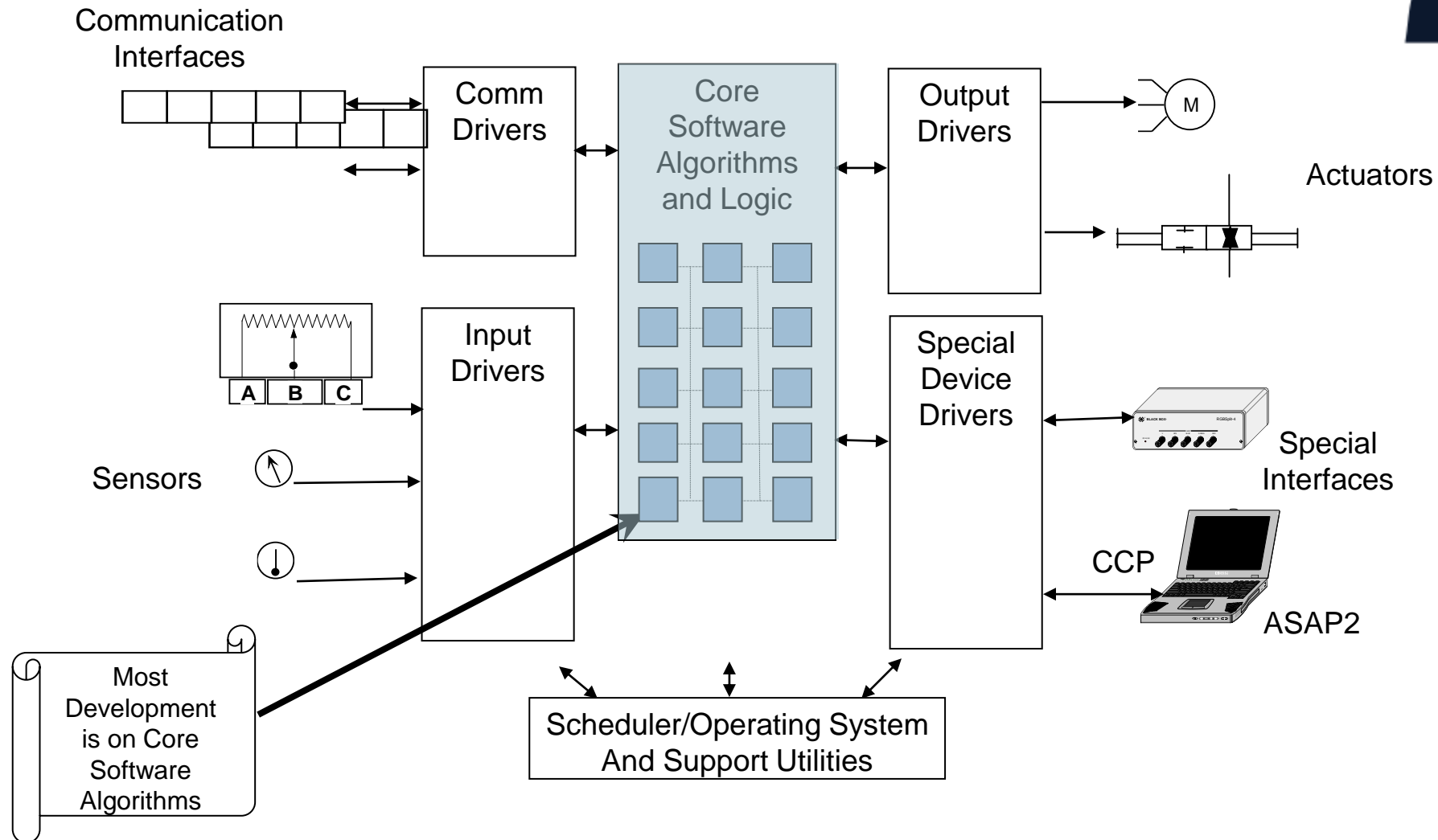
Task	Average CPU Utilization	Maximum CPU Utilization
<a href="#">Current_step [5e-05 0]</a>	10.13%	10.27%
Overall CPU Utilization	10.13%	10.27%

# Deployment on the Target

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

Copyright 2020 The MathWorks, Inc.

# Simple Embedded Software Architecture



# Integrating Generated Controller Code with an Embedded Software Project

Model

Hand

## Embedded Software Project

Execute at 20kHz

Command

ADC

Encoder

**Controller**

PWM



# Integrate Generated Controller Code with Your Hand-Coded Software Project

Model

Hand

## Embedded Software Project Pseudo-Code

```
main()
{
    adcInit();
    encoderInit();
    pwmInit();

    controllerInit();

    while(1) {
    }
}
```

```
interruptServiceRoutine()
{
    AdcStruct = readAdcCountFromDriver();
    EncoderStruct = readEncoderCountFromDriver();

    PwmStruct = controllerStep(AdcStruct, EncoderStruct);

    writePwmCountToDriver(PwmStruct);
}
```

# Customize Generated Code

Copyright 2021 The MathWorks, Inc.

**Explore more:**

1. [Edit motor & inverter parameters](#)
2. Generate c code using the 'Embedded Coder' app
3. Integrate generated code with driver code
4. Control motor via [host model](#)

```

1 /*
2  * File: speed_control_algorithm.c
3  *
4  * Code generated for Simulink model 'speed_control_algorithm'.
5  *
6  * Model version          : 3.2
7  * Simulink Coder version : 9.8 (R2022b) 13-May-2022
8  * C/C++ source code generated on : Tue Nov  8 14:45:10 2022
9  *
10 * Target selection: ert.tlc
11 * Embedded hardware selection: ARM Compatible->ARM Cortex-M
12 * Code generation objectives: Unspecified
13 * Validation result: Not run
14 */
15
16 #include "speed_control_algorithm.h"
17 #include "rtwtypes.h"
18
19 /* Exported block parameters */
20 real32_T PI_Speed = 0.589872479f; /* Variable: PI_Speed
21                                     * Referenced by: '<S2>/K11'
22                                     */
23
24 /* Block states (default storage) */
25 DW_speed_control_algorithm_T speed_control_algorithm_DW;
26
27 /* Model step function */
28 boolean_T My_Special_StepName(real32_T *arg_speed_ref_pu, real32_T
29 arg_speed_meas_pu, boolean_T My_arg_enable, boolean_T arg_en_closed_loop,
30 real32_T arg_IdqRef_PU[2])
31 {
32     /* local block i/o variables */
33     real32_T My_Signal_Name;
34     real32_T rtb_DeadZone;
35     real32_T rtb_IProduct;
36     real32_T rtb_Sum;
37     int8_T tmp;
38     int8_T tmp_0;
39
40     /* specified return value */
41     boolean_T arg_PID_Enable;
42
43     /* Outputs for Atomic SubSystem: '<root>/Speed Control' */
44     /* Switch: '<S3>/Switch' incorporates:
45      * Import: '<root>/en_closed_loop'
46      * Import: '<root>/enable'
47      * Import: '<root>/speed_meas_pu'
48      * Import: '<root>/speed_ref_pu'
49      * Logic: '<S3>/AND'
50      */
51     if (arg_en_closed_loop && My_arg_enable) {
52         rtb_IProduct = *arg_speed_ref_pu;
53     } else {
54         rtb_IProduct = arg_speed_meas_pu;
55     }
56
57     /* Sum: '<S5B>/Add1' incorporates:
58      * Constant: '<S5B>/filter_constant'
59      * Constant: '<S5B>/one'
60      * Product: '<S5B>/Product'
61      * Product: '<S5B>/Product1'
62      * Switch: '<S3>/Switch'

```

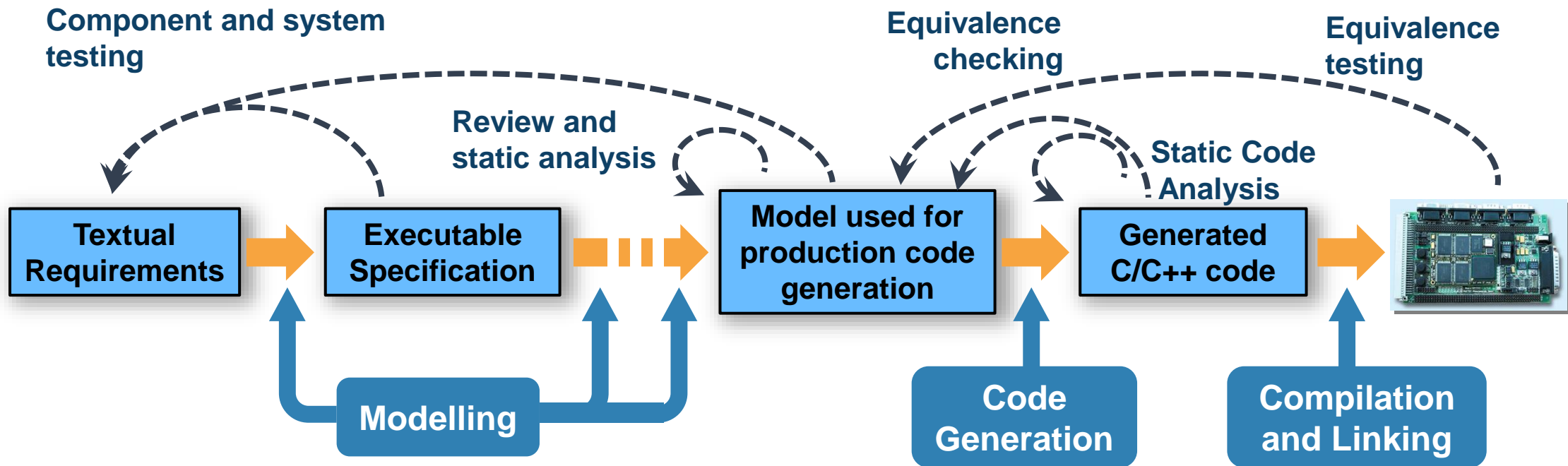
Source	Function Customization Template	Function Name	Function Preview
Initialize	Model default	speed_control_algorithm_initialize	void speed_control_algorithm_initialize[* self]
PeriodicD1 [Sample Time: 0.0005s]	Model default	My_Special_StepName	arg_PID_Enable = My_Special_StepName[* self], * arg_s...
Terminate	Model default		void speed_control_algorithm_terminate[* self]

# Basic generated code architecture & interface

- Basic generated functions format:
  - `void modelname_initialize(void)` : to call at system initialization
  - `Void modelname_step(void)` : to call each processing step (on timer or interrupt)
  - `Void modelname_terminate(void)` : to call at system shutdown.
  
- Several possible customisation using Embedded coder
  - Functions / files names
  - Function interface (return & argument passing mode).
  - Several step functions for concurrent tasking.
  - ...
    - example : `int MyModel_step(int inputs, *int params, *int dworks)`

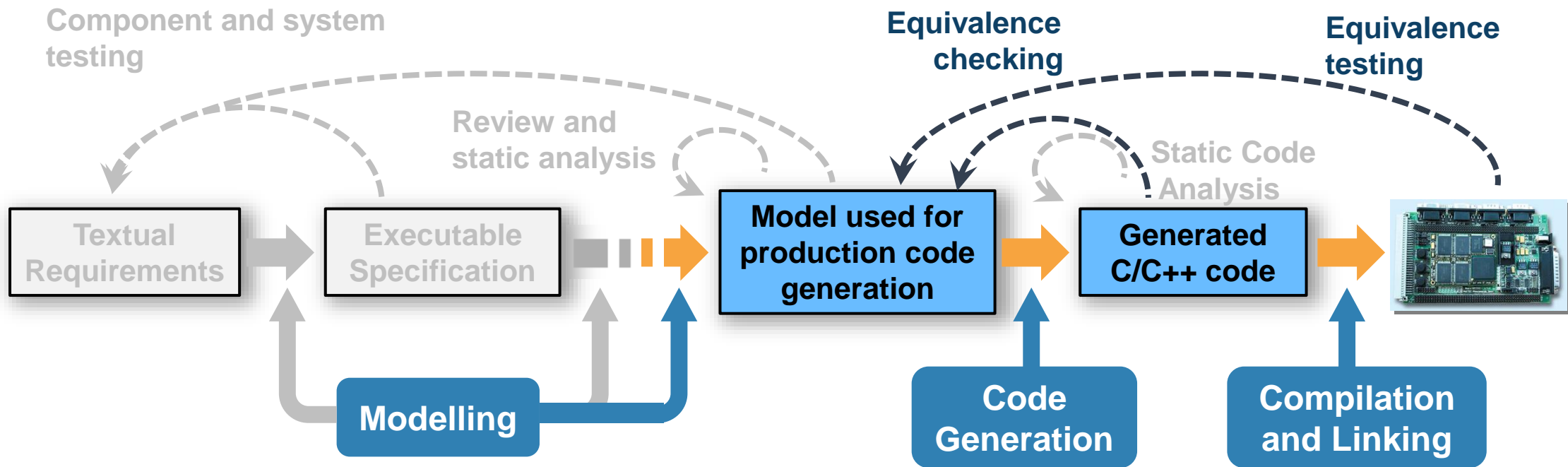
# Complete Model-Based Design Workflow

Get the complete confidence in your design



# Complete Model-Based Design Workflow

Get the complete confidence in your design



# Key Takeaways

- Model-based design for motor control enables you
  - To make faster time to market.
  - To be more resilient to external disturbances
  - Increase collaboration
  - Share knowledge
  
- Motor Control Blockset, using reference examples & built-in blocks, enables you
  - To minimize even more development time
  - Upskill yourself
  - To experiment easily new control technics

# Q&A