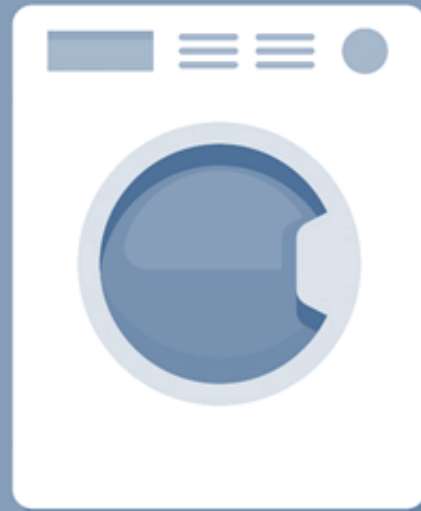
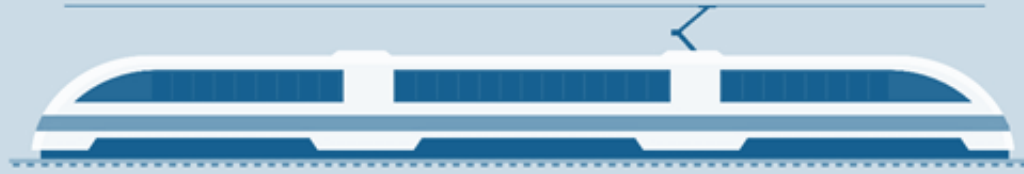


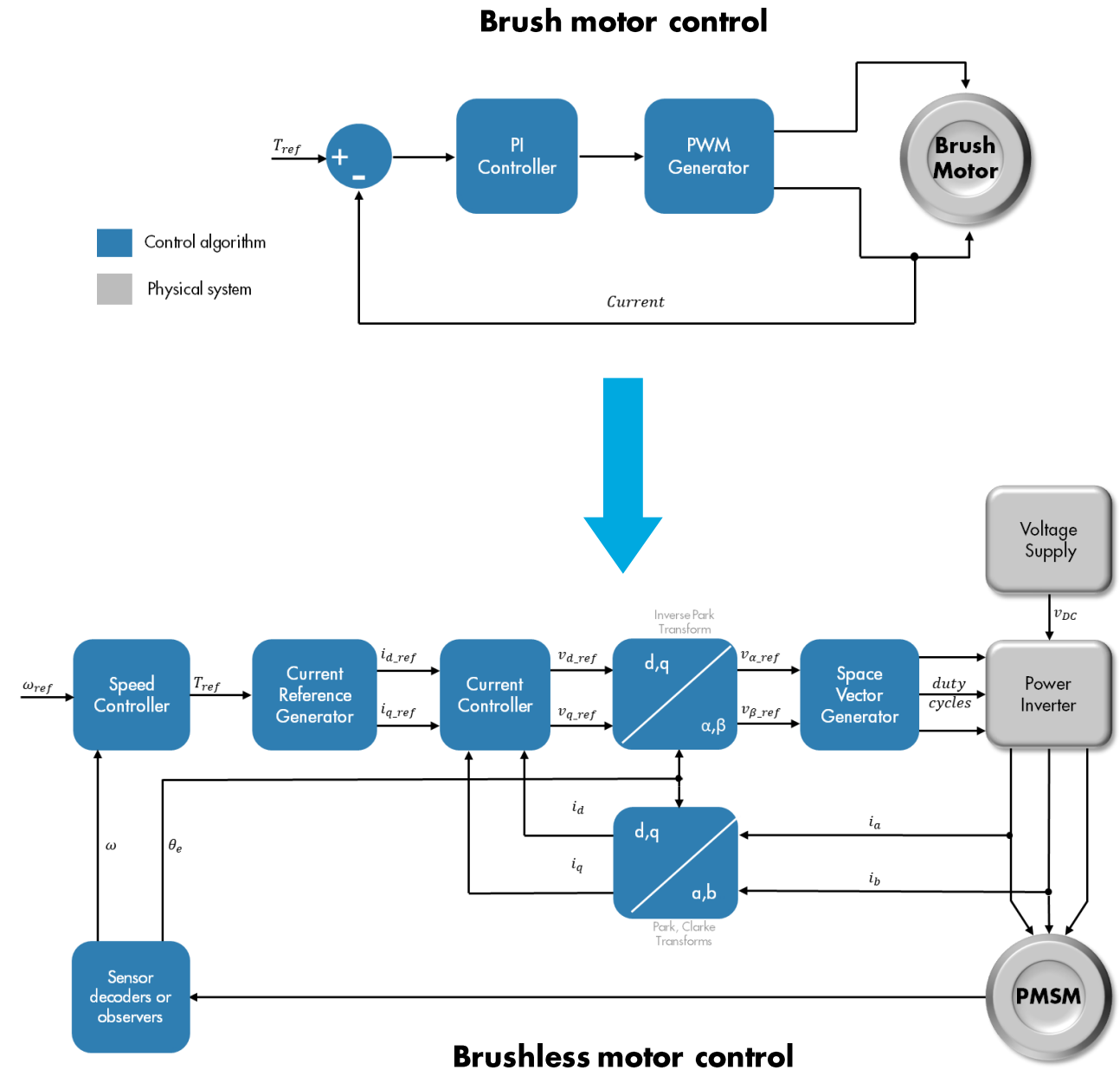
# Développement d'un FOC pour PMSM

Accélérer l'électrification



# More Complex Algorithms

- Increasing motor control algorithm **complexity**
  - Field-Oriented Control (FOC)
  - Field-weakening control
  - Sensorless
  - Space vector PWM
  
- Increasing need to run these algorithms **faster**
  - Wide bandgap semiconductors
  - Increasing popularity of motor types such as switched reluctance motors



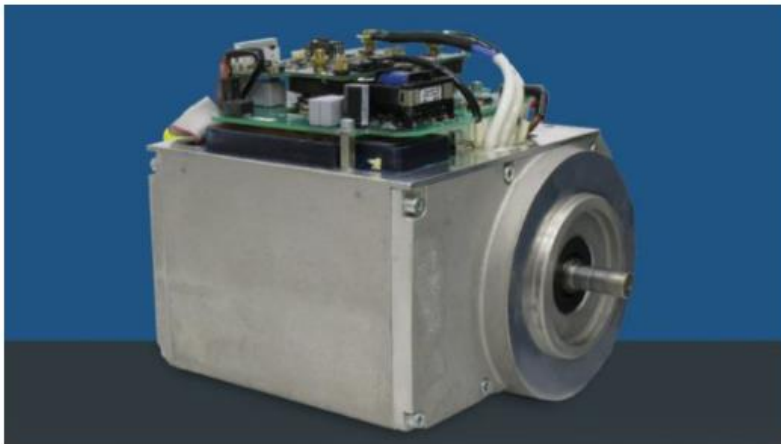
# Model-Based Design

Answer to complex design

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

"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



ATB Technologies permanent magnet synchronous motor.

### 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

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

"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



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

# Why Simulink for motor control?

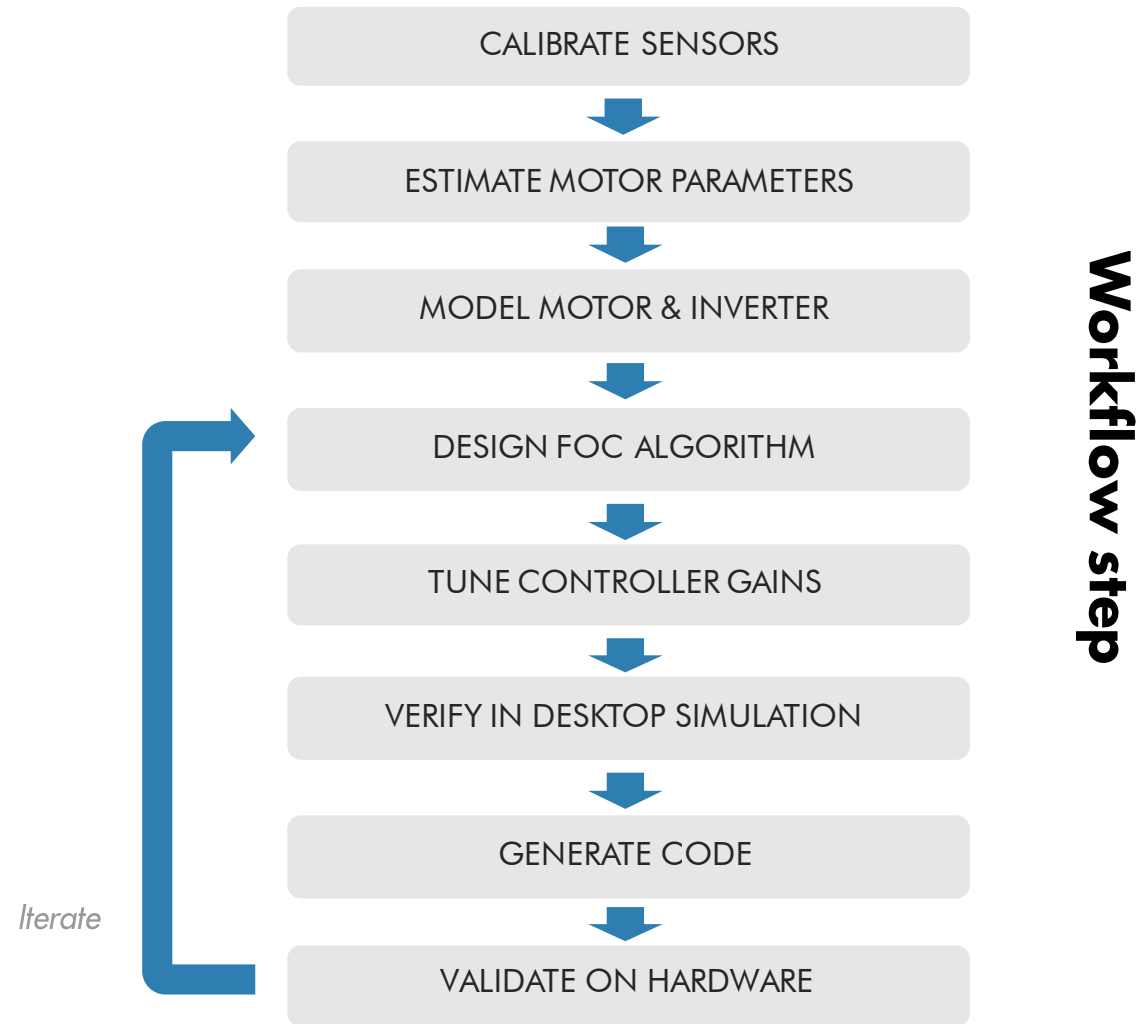
## Motor Control Blockset

- Verify control algorithm with desktop simulation
- Generate compact and fast code from models
- Minimize development time using reference examples



→ Customers routinely report 50% faster time to market

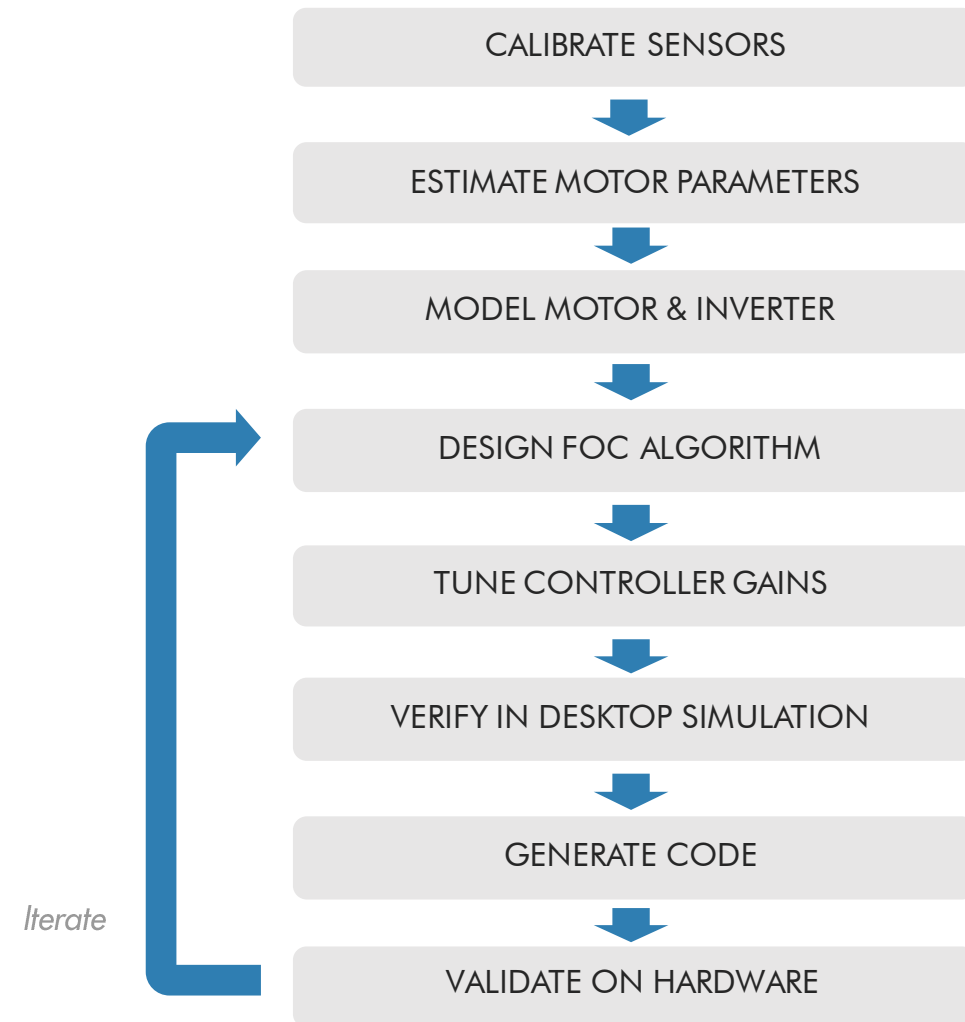




# Agenda

## From Desktop Simulation to Software Deployment

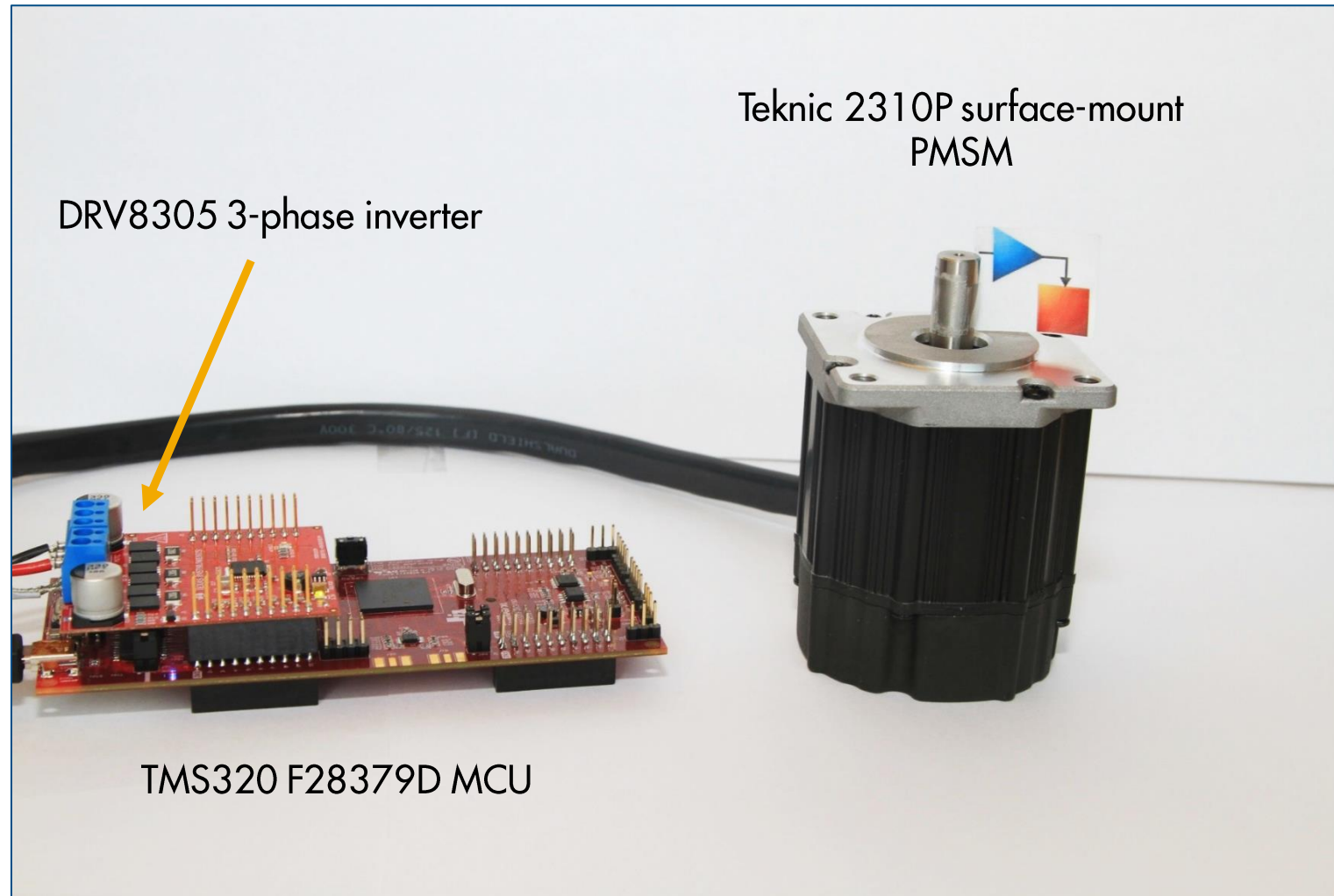
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Code generation



**Workflow step**

# Texas Instruments Motor Control Kit

## Hardware configuration

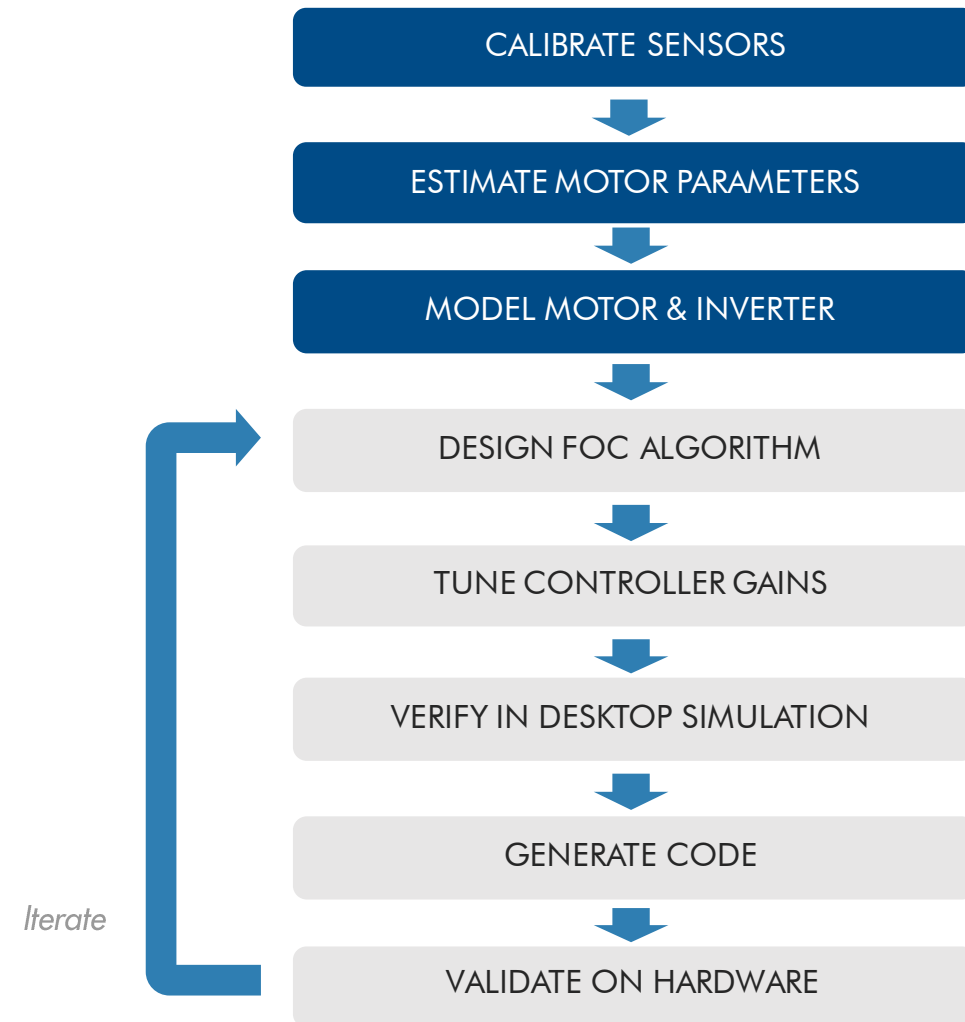




# Agenda

## From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Code generation

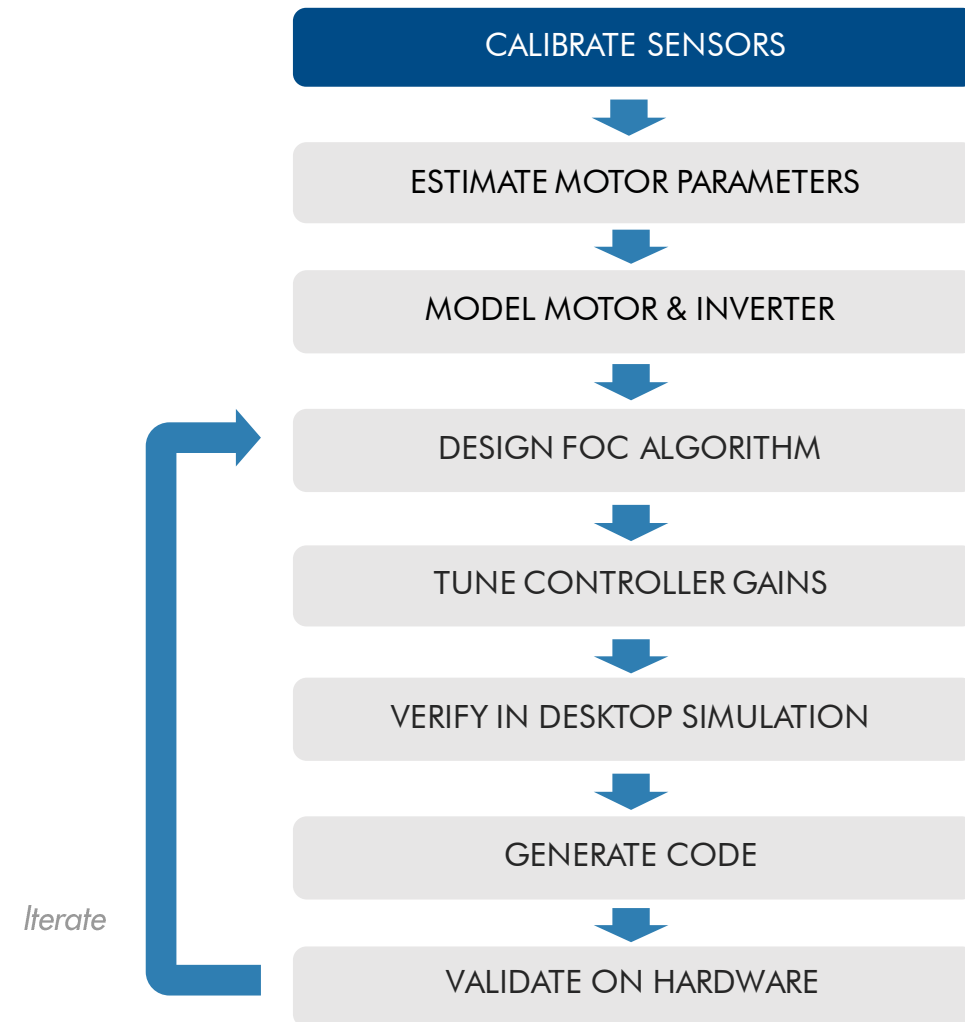


**Workflow step**

# 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
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test

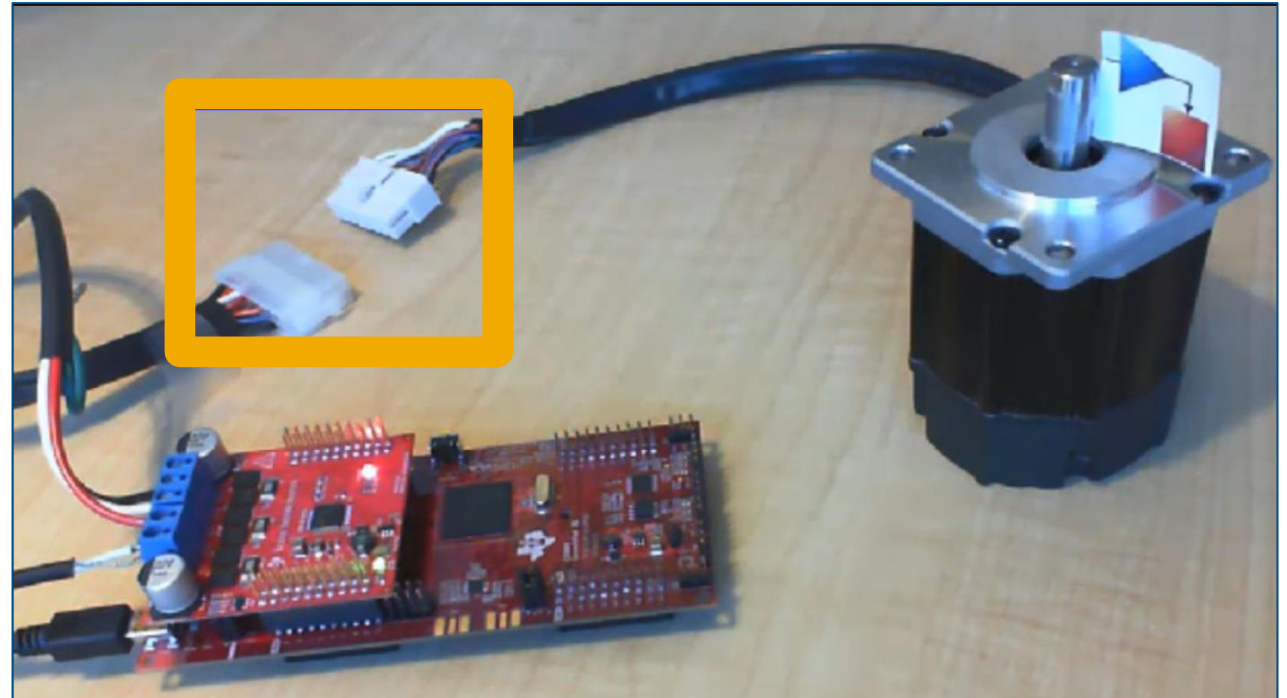


**Workflow step**

# Sensor Calibration

## Plant Modeling

- ADC offsets
- Position Sensor Offset

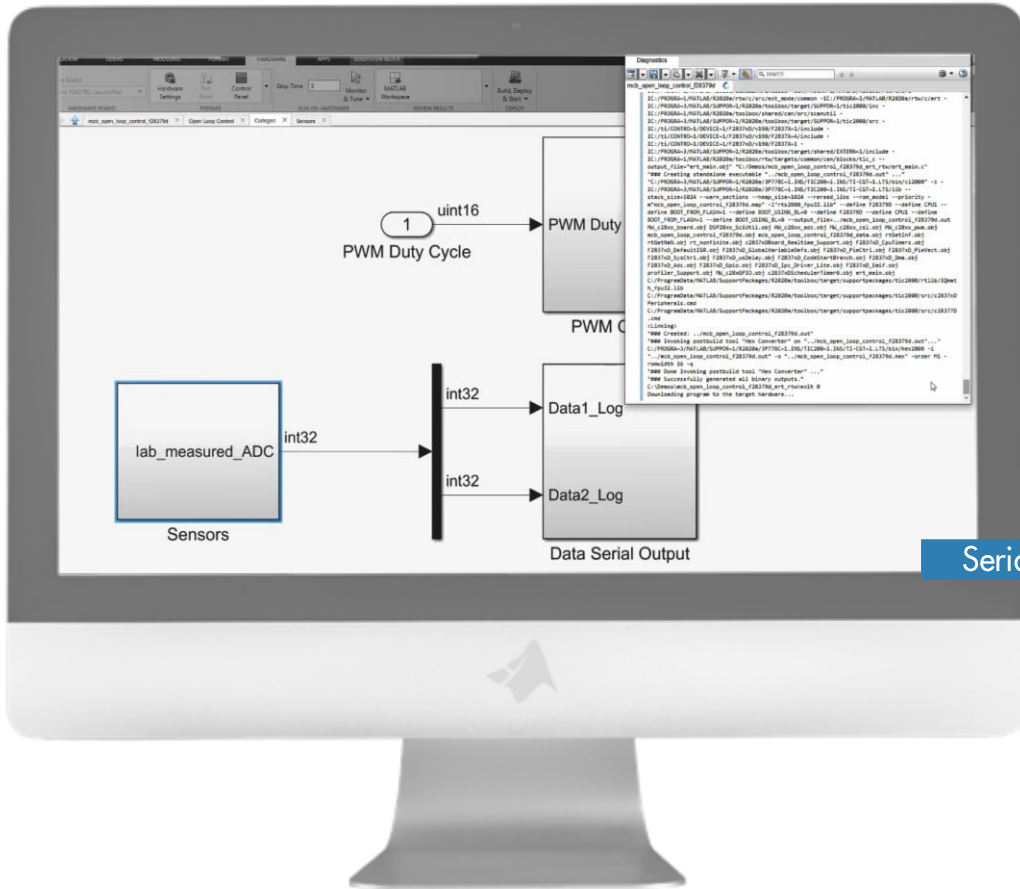


# ADC Offsets

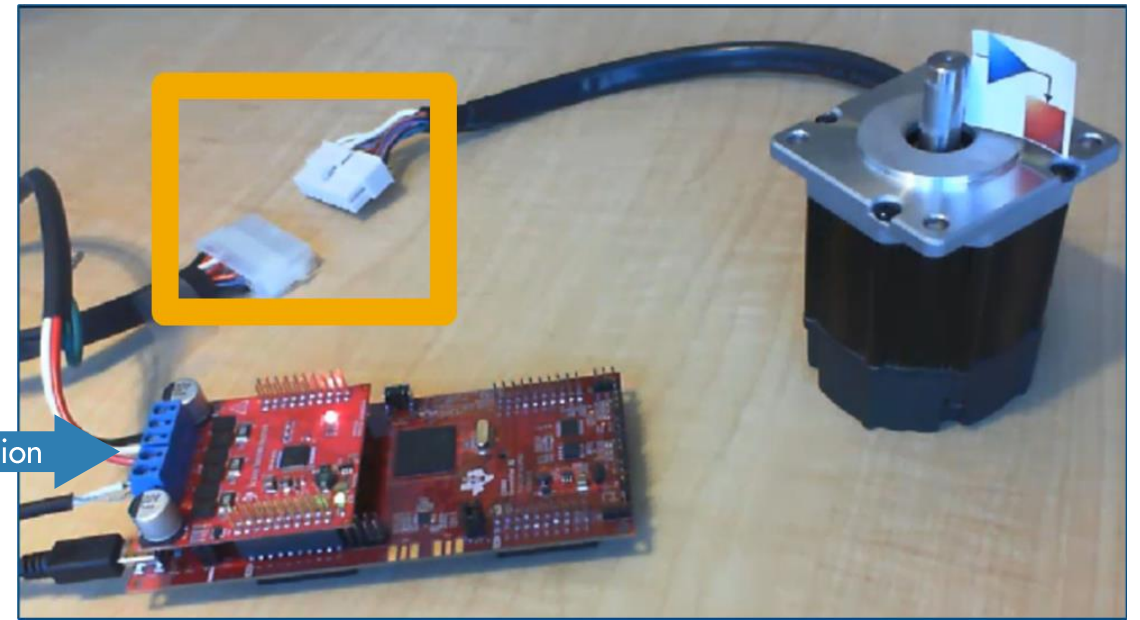
## Plant Modeling

3

Completed and Ready to Run



Serial connection



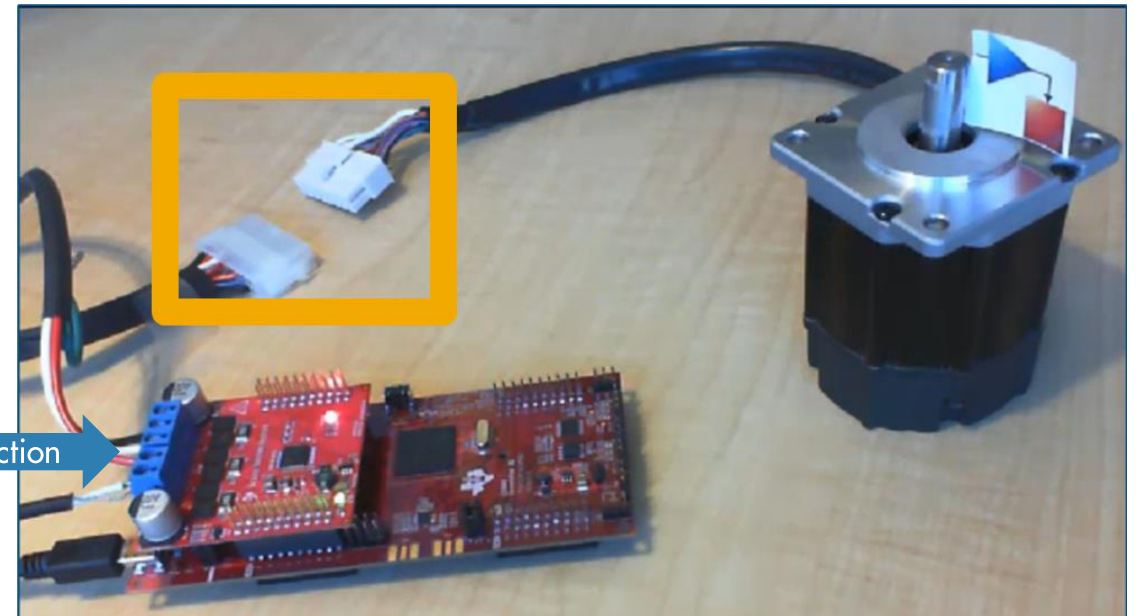
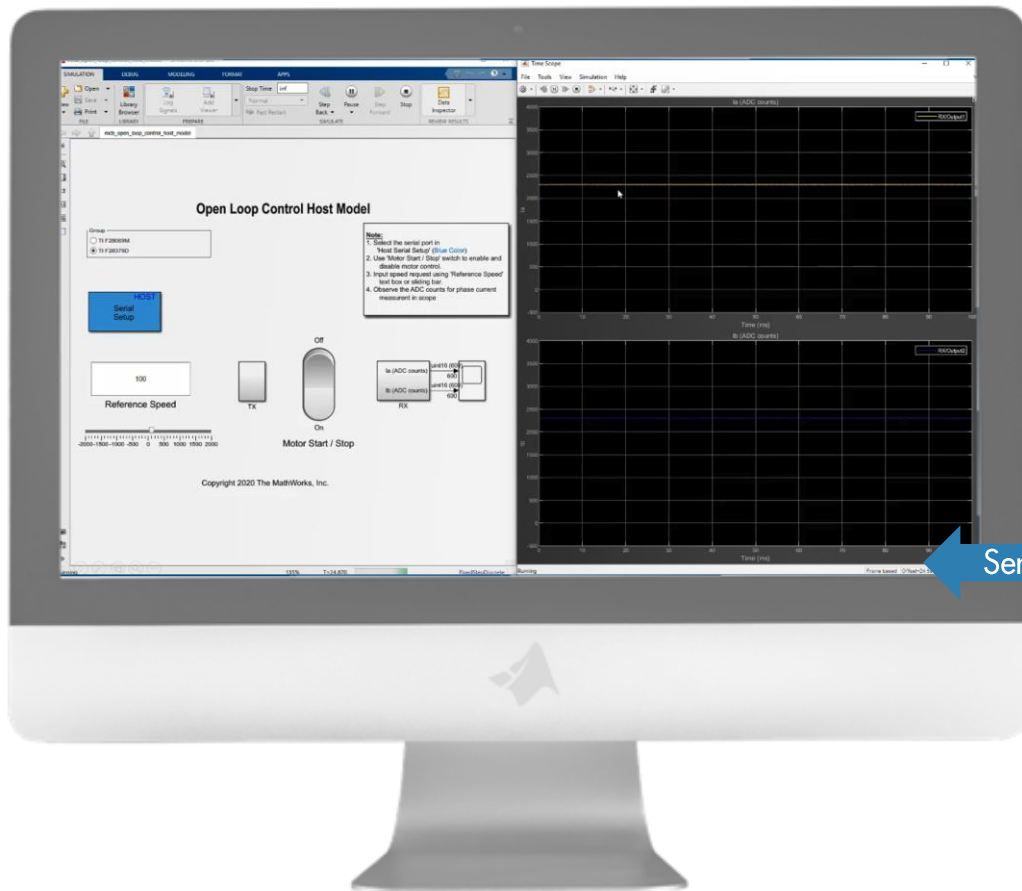
Development Computer with ADC offset calibration model

Target Hardware

# ADC Offsets

## Plant Modeling

### 4 External Mode Simulation



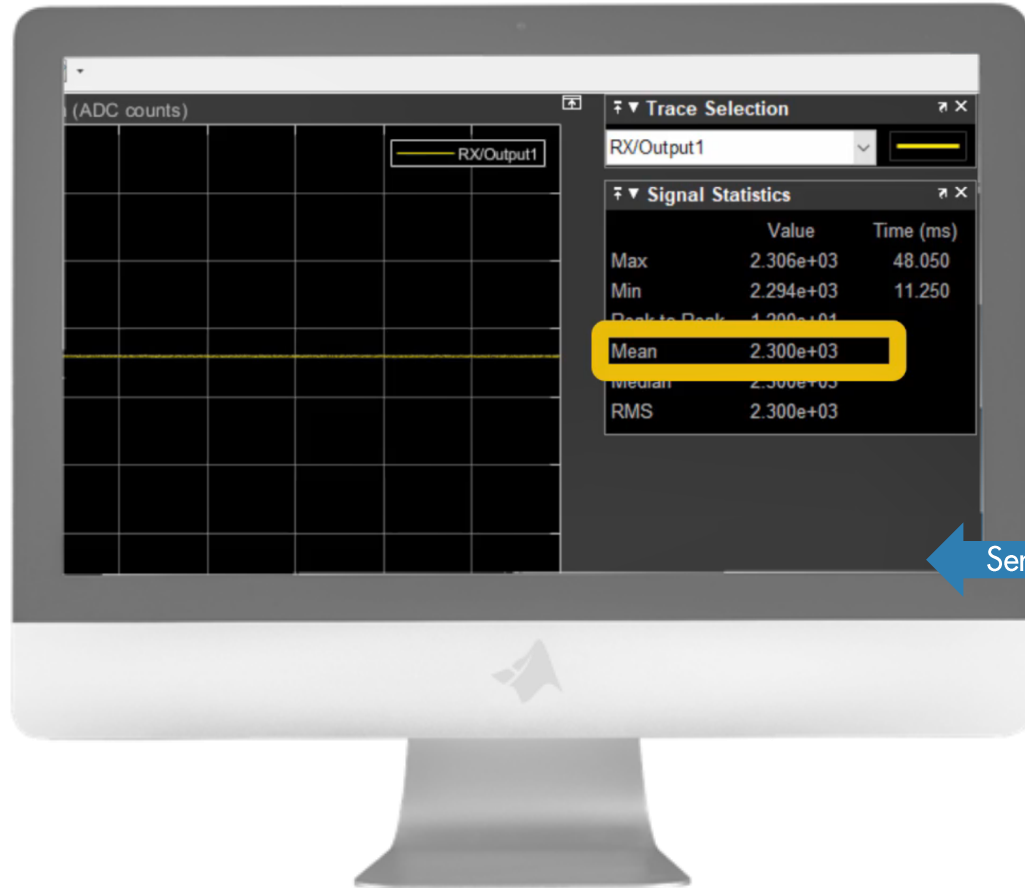
Development Computer with ADC offset calibration model

Target Hardware

# ADC Offsets

## Plant Modeling

- 5 Get Offset for Phase A and B



Serial connection

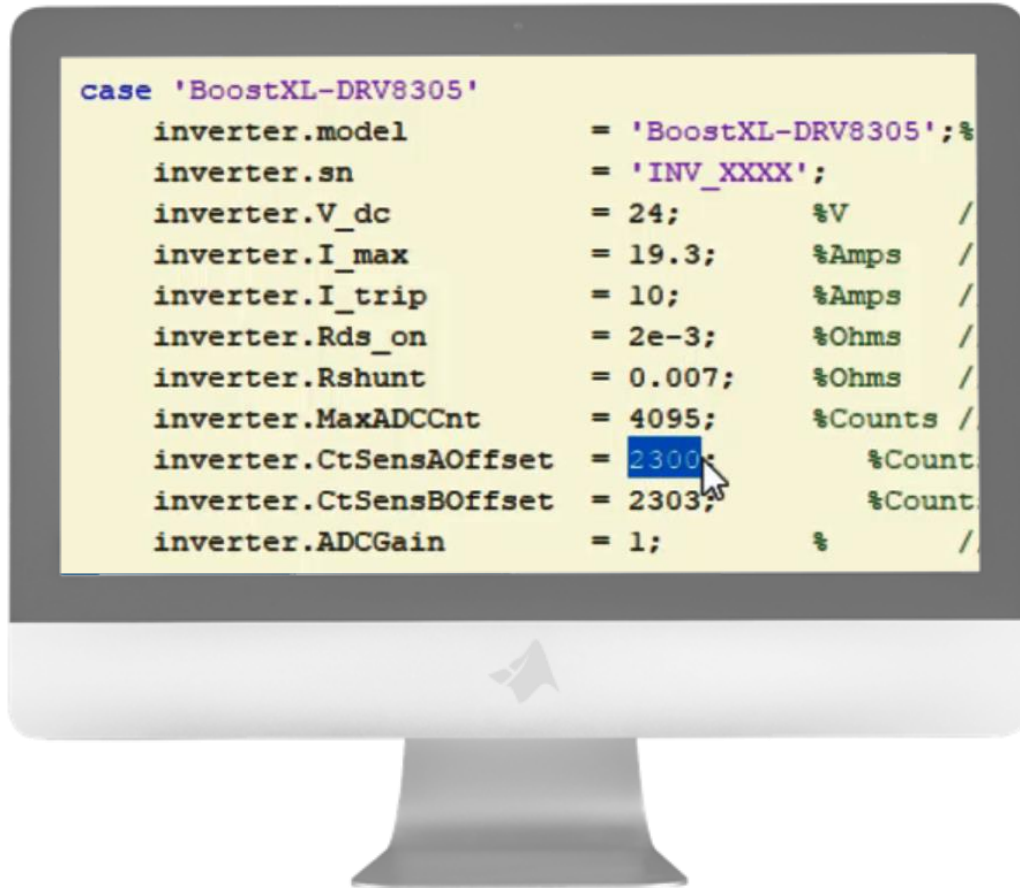




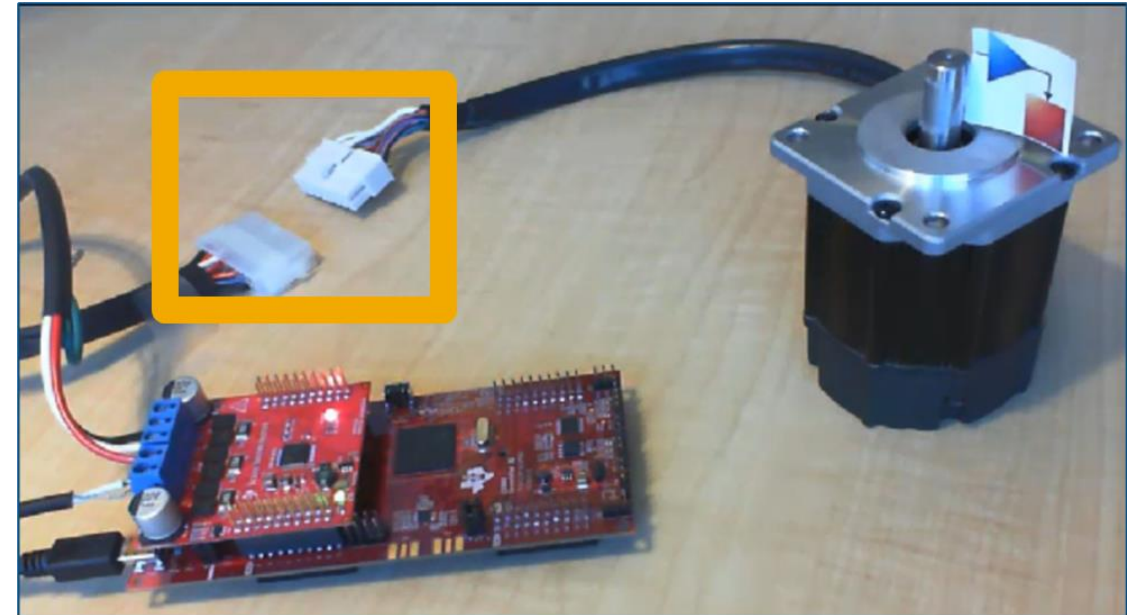
# ADC Offsets

## Plant Modeling

### 6 Parameterization script



Development Computer with ADC offset calibration model



Target Hardware

# Position Sensor Offset

## Plant Modeling

### Hall Offset Calibration for PMSM Motor

R2021a

This example calculates the offset between the rotor direct axis (d-axis) and position detected by the Hall sensor. The field-oriented control (FOC) algorithm needs this position offset to run the permanent magnet synchronous motor (PMSM) correctly. To compute the offset, the target model runs the motor in the open-loop condition. The model uses a constant  $V_d$  (voltage along the stator's d-axis) and a zero  $V_q$  (voltage along the stator's q-axis) to run the motor (at a low constant speed) by using a position or ramp generator. When the position or ramp value reaches zero, the corresponding rotor position is the offset value for the Hall sensors.

[Open Example](#)

The control algorithm (available in the field-oriented control and parameter estimation examples) uses this offset value to compute an accurate position of d-axis of the rotor. The controller needs this offset to optimally run the PMSM.

#### Models

This example includes these models:

- [mcb\\_pmsm\\_hall\\_offset\\_f28069m](#)
- [mcb\\_pmsm\\_hall\\_offset\\_f28379d](#)

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

```
open_system('mcb_pmsm_hall_offset_f28069m.slx');
```

### Offset Computation with Hall sensor

**Note:** This example requires a TI F28069m controller card mounted on DRV8312 inverter connected to a PMSM Motor with Hall Sensor

#### Steps:

1. Enter parameters in the Configuration panel.
2. Click **Build, Deploy & Start** in the **Hardware** tab.
3. Perform calibration by using [host model](#).
4. If the motor does not start or rotate smoothly, increase **Vd Ref in Per Unit voltage** (that can have a maximum value of 1) in the Configuration panel.
5. If the current drawn by the connected motor is too high, reduce the value mentioned in step 4.
6. [Learn more](#) about this example.

Configuration

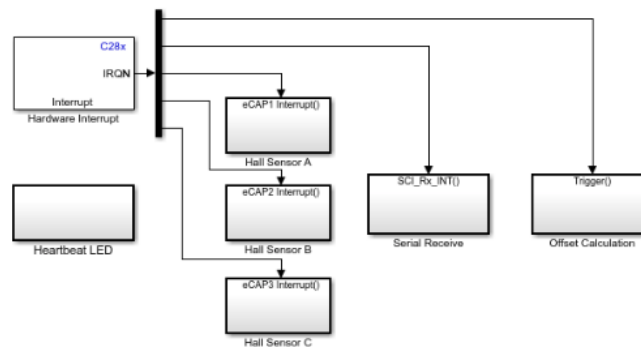
Number of Pole Pairs:

PWM Frequency [Hz]:

Data type for control algorithm:

Vd Ref in Per Unit voltage:

- Global variables
- GlobalHallState
  - HallStateChangeFlag
  - GlobalSpeedCount
  - GlobalSpeedValidity
  - GlobalDirection
  - Enable



### Documentation

Search R2021a Documentation

CONTENTS

All Examples Blocks

### Sensor Calibration

**Offset Computation with Hall sensor**

Note: This example requires a TI F28069m controller card mounted on DRV8312 inverter connected to a PMSM Motor with Hall Sensor

**Offset Computation for QEP**

Note: This example requires a TI F28069m controller card mounted on DRV8312 inverter connected to a PMSM Motor with Hall Sensor

### Hall Offset Calibration for PMSM Motor

Calculates the offset between the rotor direct axis (d-axis) and position detected by the Hall sensor. The field-oriented control (FOC)

[Open Example](#)

### Quadrature Encoder Offset Calibration for PMSM Motor

Calculates the offset between the d-axis of the rotor and encoder index pulse position as detected by the quadrature encoder sensor. The

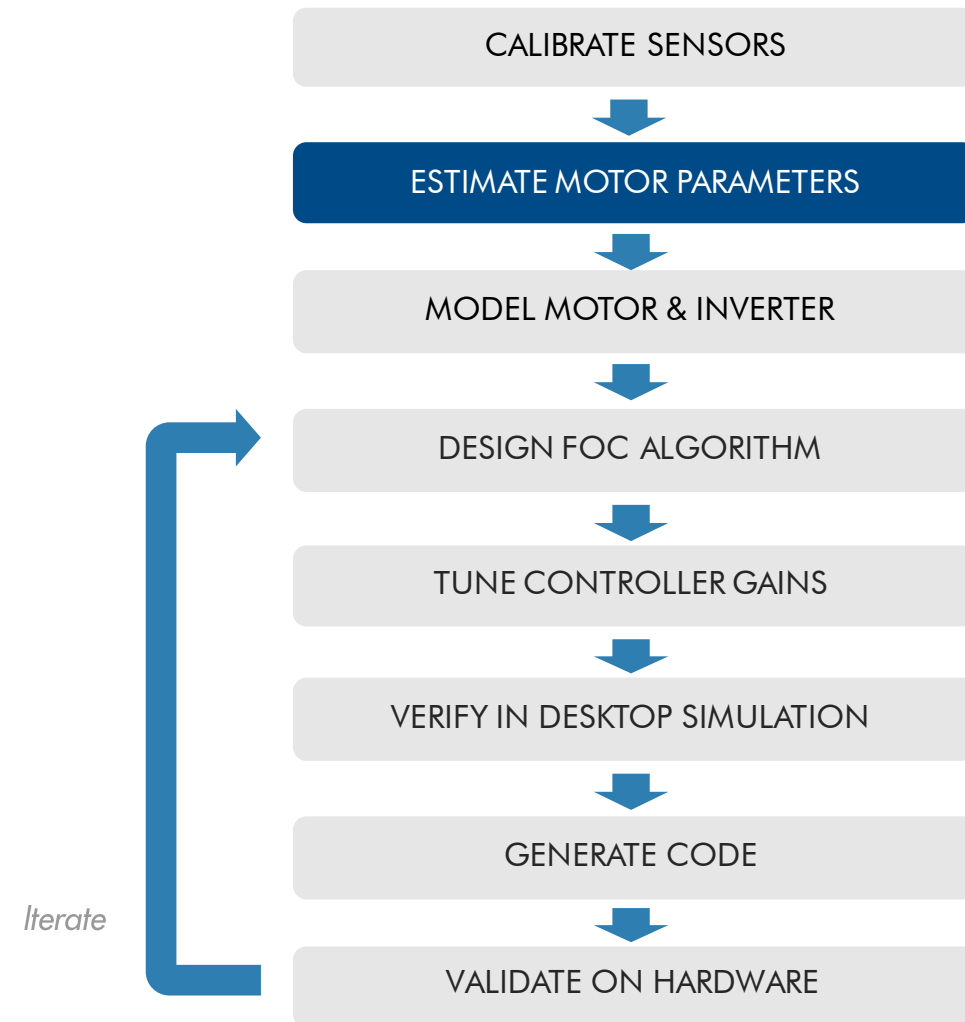
[Open Example](#)



# 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
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test

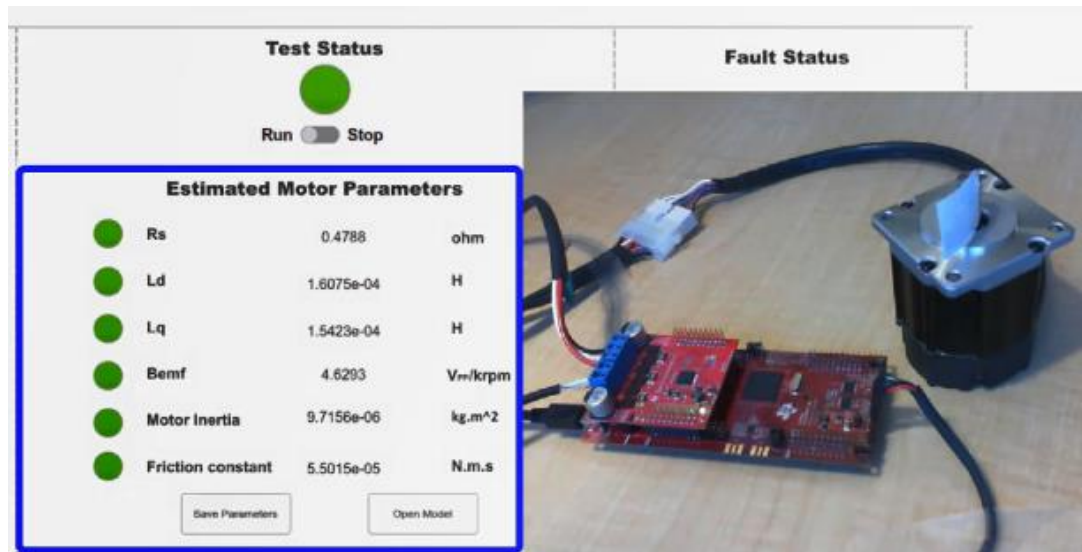


Workflow step

# Motor Parameters Estimation

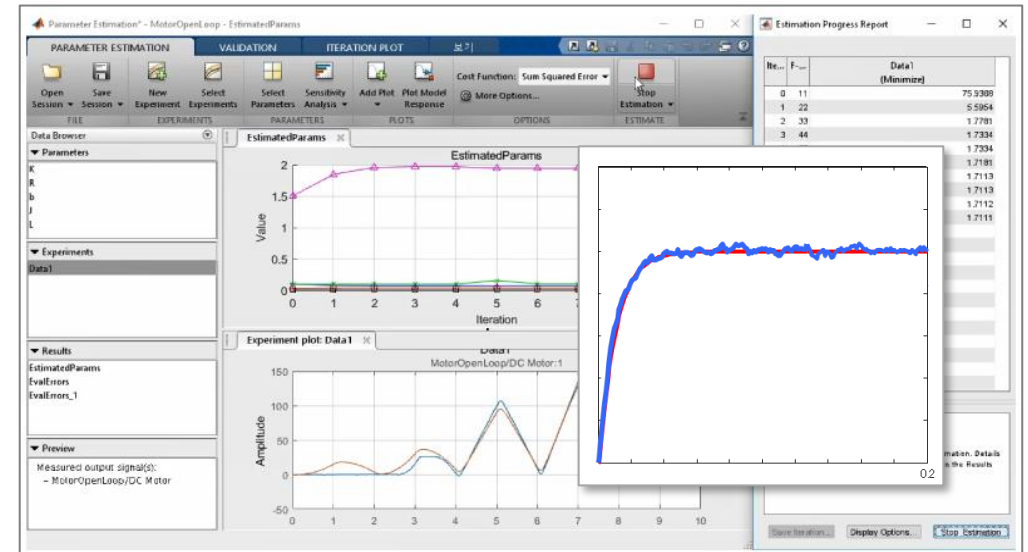
## Plant Modeling

Two types of parameter estimation methods:



Parameter Estimation with Instrumented Test

or



Parameter Estimation using Operation Data

# Motor Parameters Estimation - Instrumented Test

## Plant Modeling

**Board Selection**  
DRV8305 and F28379D Launchpad

**Communication Port**  
HOST  
Serial Setup

The COM port has to match your board  
For F28379D Launchpad, set Baudrate to 115200  
For F28379D Launchpad, set Baud rate to 56k

**Required Inputs**

Nominal Voltage:  V

Nominal Current:  A (rms value)

Nominal Speed:  rpm

Pole pairs:

Input DC Voltage:  V

Hall Offset:  Per Unit Position

Note: Press Ctrl+D to update the workspace

**Hall Offset:** For Hall offset calculation open required model for the hardware  
[mcb\\_pmsm\\_hall\\_offset\\_f28069m](#)  
[mcb\\_pmsm\\_hall\\_offset\\_f28379d](#)

**Target Models:** Click **Build load and Run** in required model for loading the target  
[mcb\\_param\\_est\\_f28069\\_DRV8312](#)  
[mcb\\_param\\_est\\_f28379D\\_DRV8305](#)

**Test Status**  
Run  Stop

**Estimated Motor Parameters**

<input type="checkbox"/>	Rs	--	ohm
<input type="checkbox"/>	Ld	--	H
<input type="checkbox"/>	Lq	--	H
<input type="checkbox"/>	Bemf	--	Vpp/krpm
<input type="checkbox"/>	Motor Inertia	--	kg.m <sup>2</sup>
<input type="checkbox"/>	Friction constant	--	N.m.s

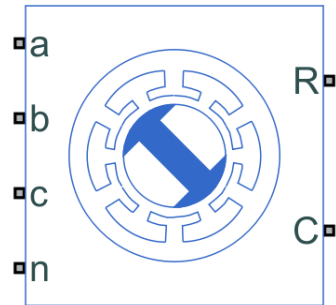
Save Parameters    Open Model

**Signal Conditioning and Scaling**

Signal from Target  
Speed

Algorithm  
Copyright 2020 The MathWorks, Inc.

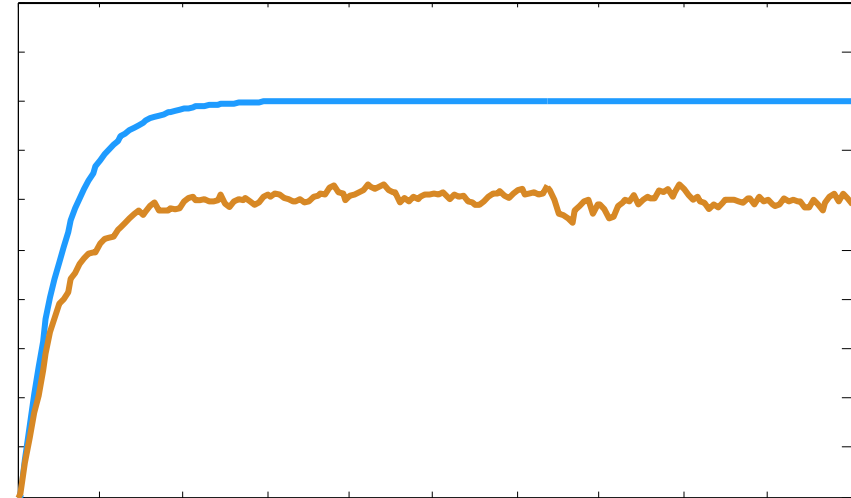
# Parameter Estimation Using Operation Data



PMSM model



Real PMSM



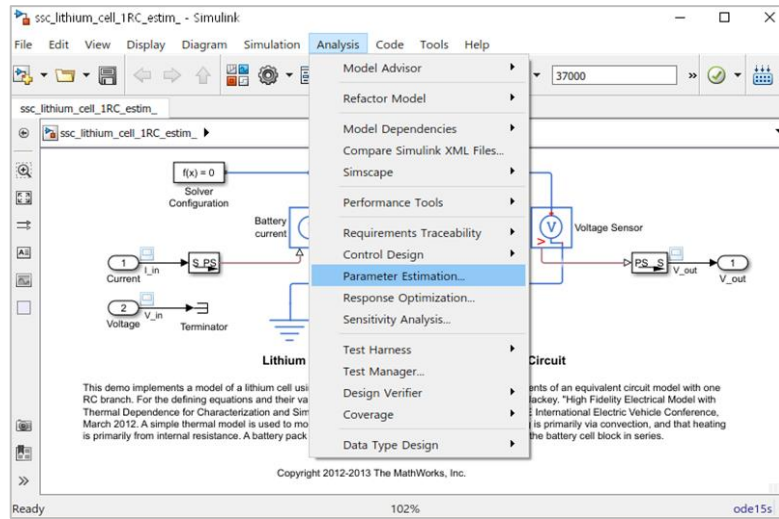
**Problem:** Simulation data does not match measured data because the parameters are incorrect

**Solution:** Use **Simulink Design Optimization** to automatically tune model parameters



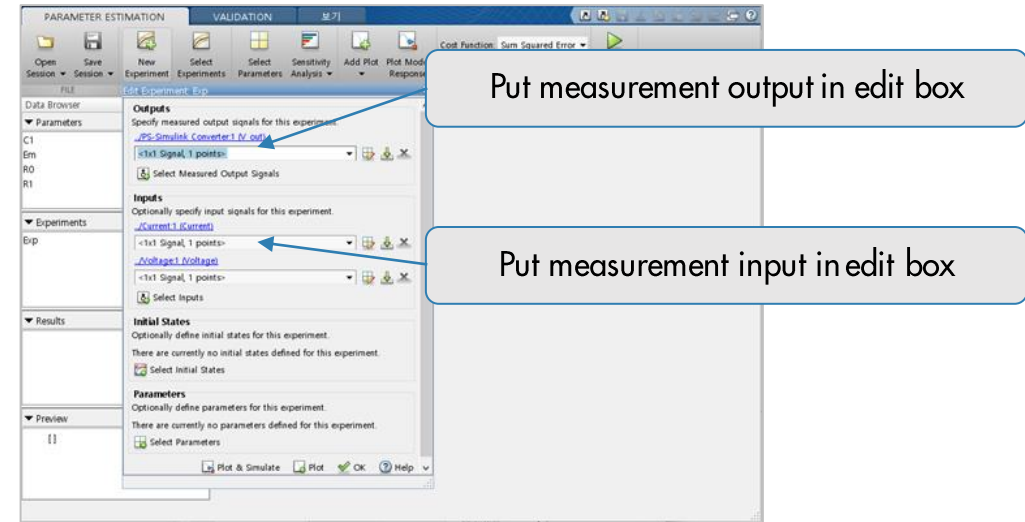
1

## Open Parameter Estimation



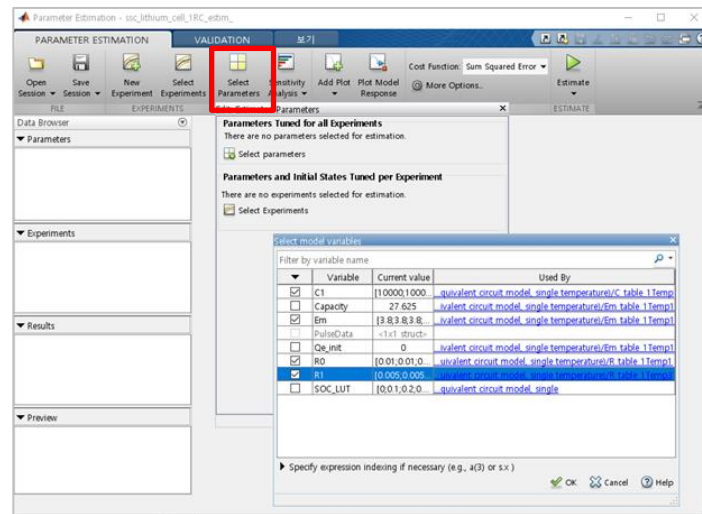
2

## Make New Experiment



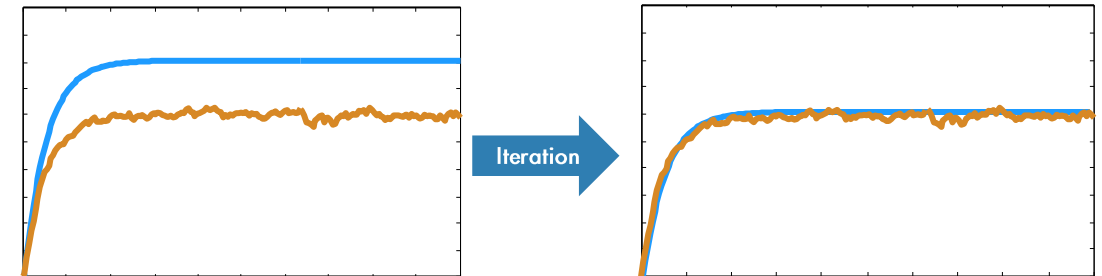
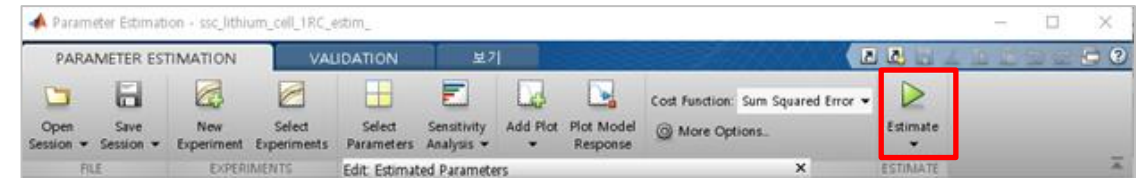
3

## Select Tuning Parameters



4

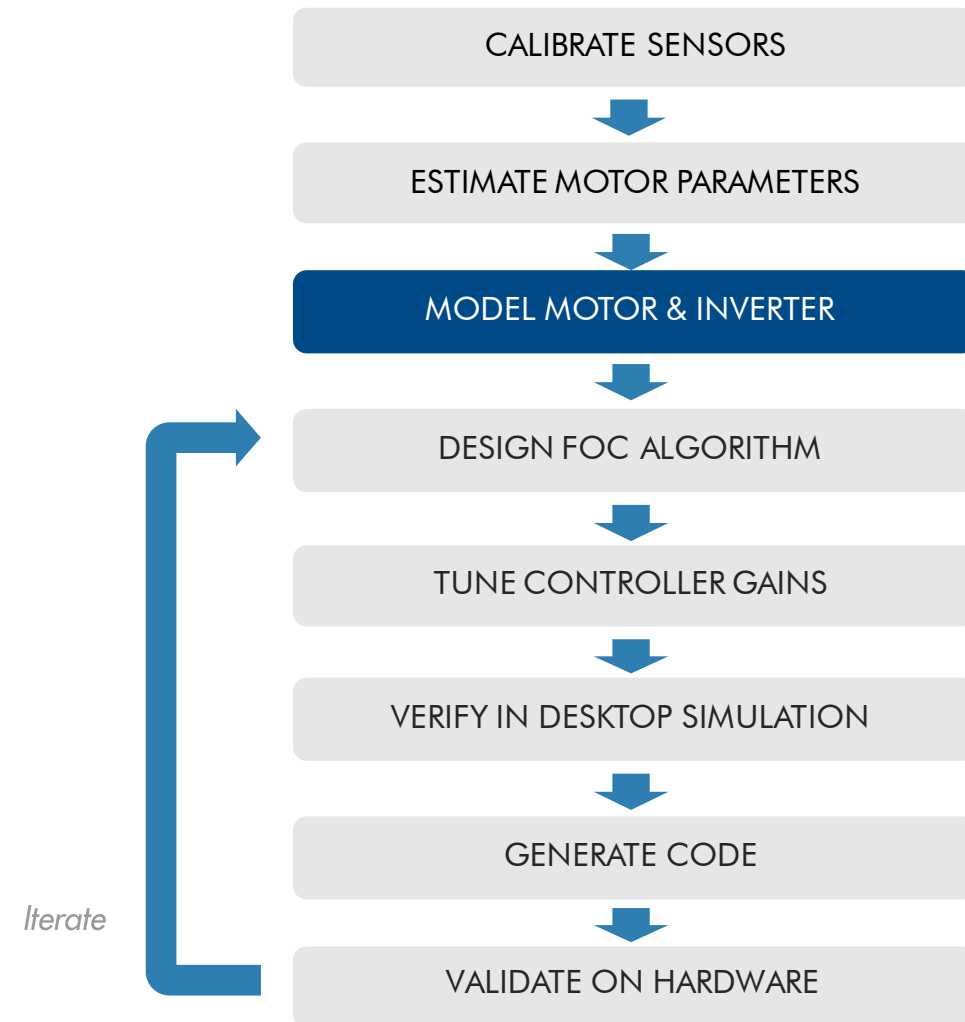
## Run Optimization



# 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
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Rapid control prototyping
  - Code generation
  - Hardware-In-The-Loop (HIL) test

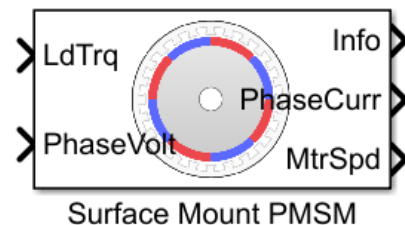
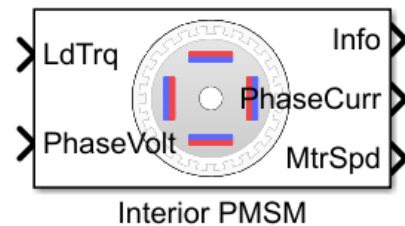
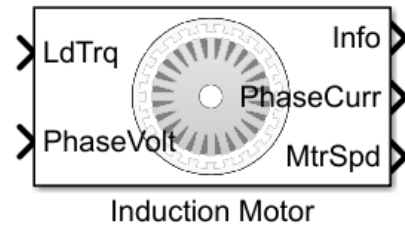


**Workflow step**

# Motor and Inverter Modeling

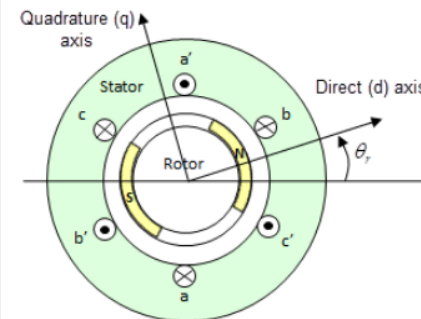
Choose the right level of fidelity

- 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

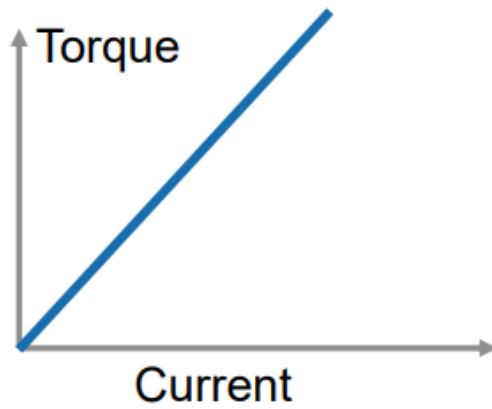
The block implements these equations, expressed in the motor flux reference frame (dq frame). All quantities in the motor reference frame are referred to the stator.

$$\omega_e = P\omega_m$$

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

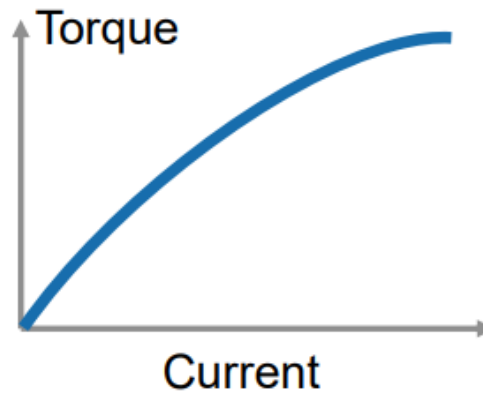
# Model Fidelity

## Plant Modeling



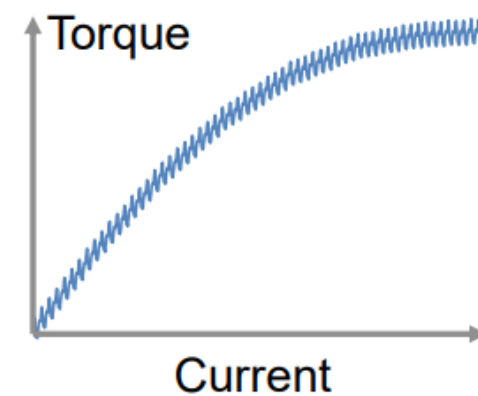
Linear Lumped Parameter

**Motor Control Blockset  
Simscape Electrical**



Saturation

**Simscape Electrical**



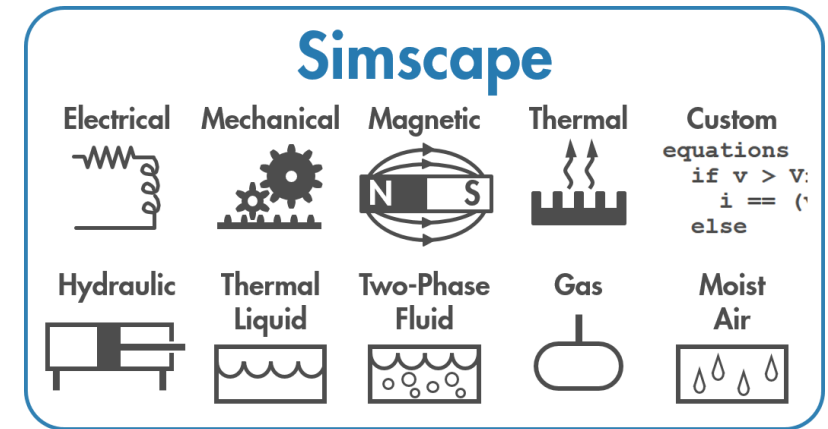
Saturation &  
Spatial Harmonics

**Simscape Electrical**

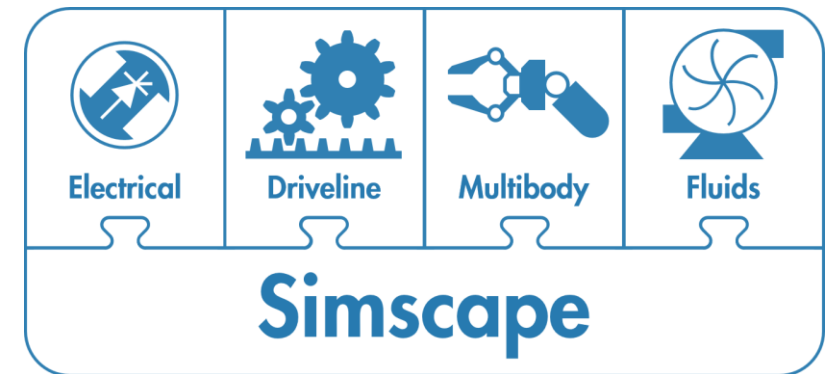
# Simscape Products

## 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



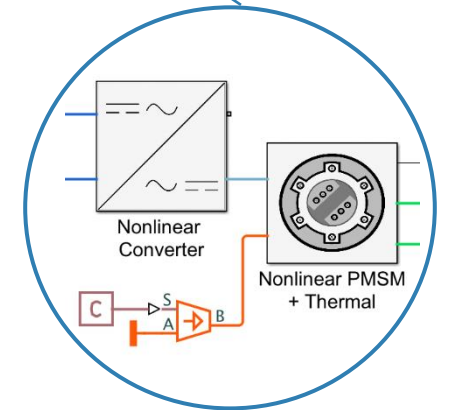
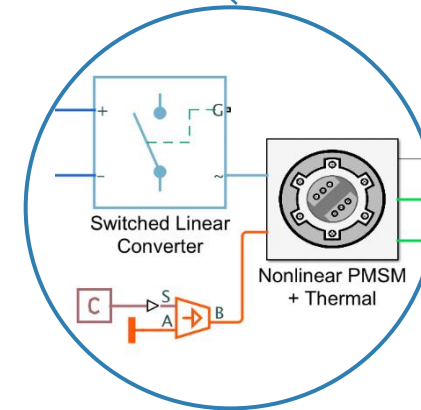
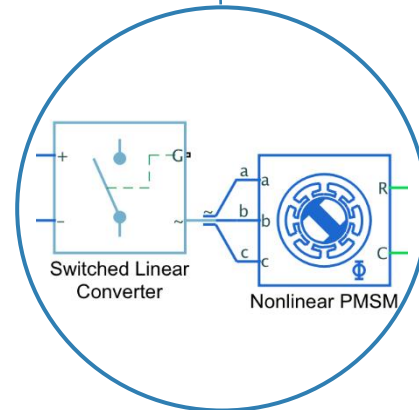
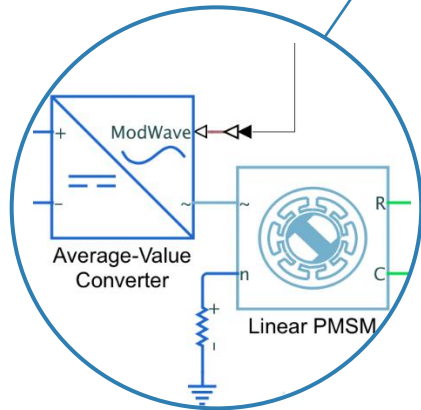
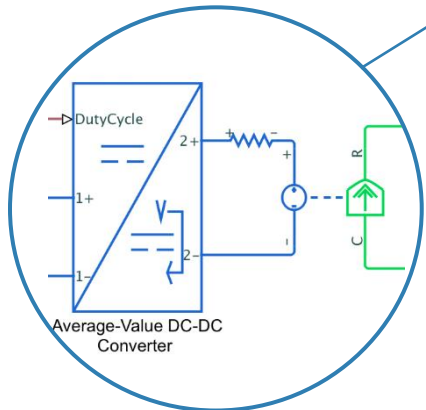
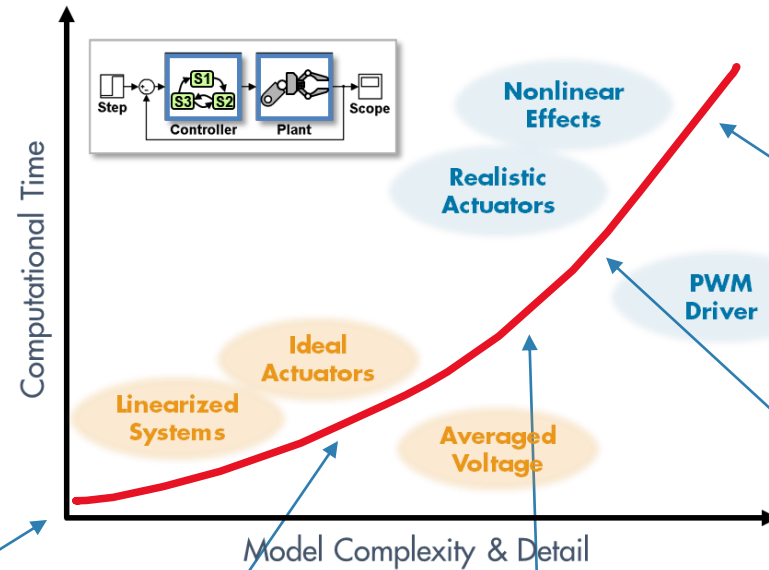
Simscape Foundation



Simscape Add-Ons

# Trade Off - Balance Model Fidelity vs Simulation Speed

## Plant Modeling

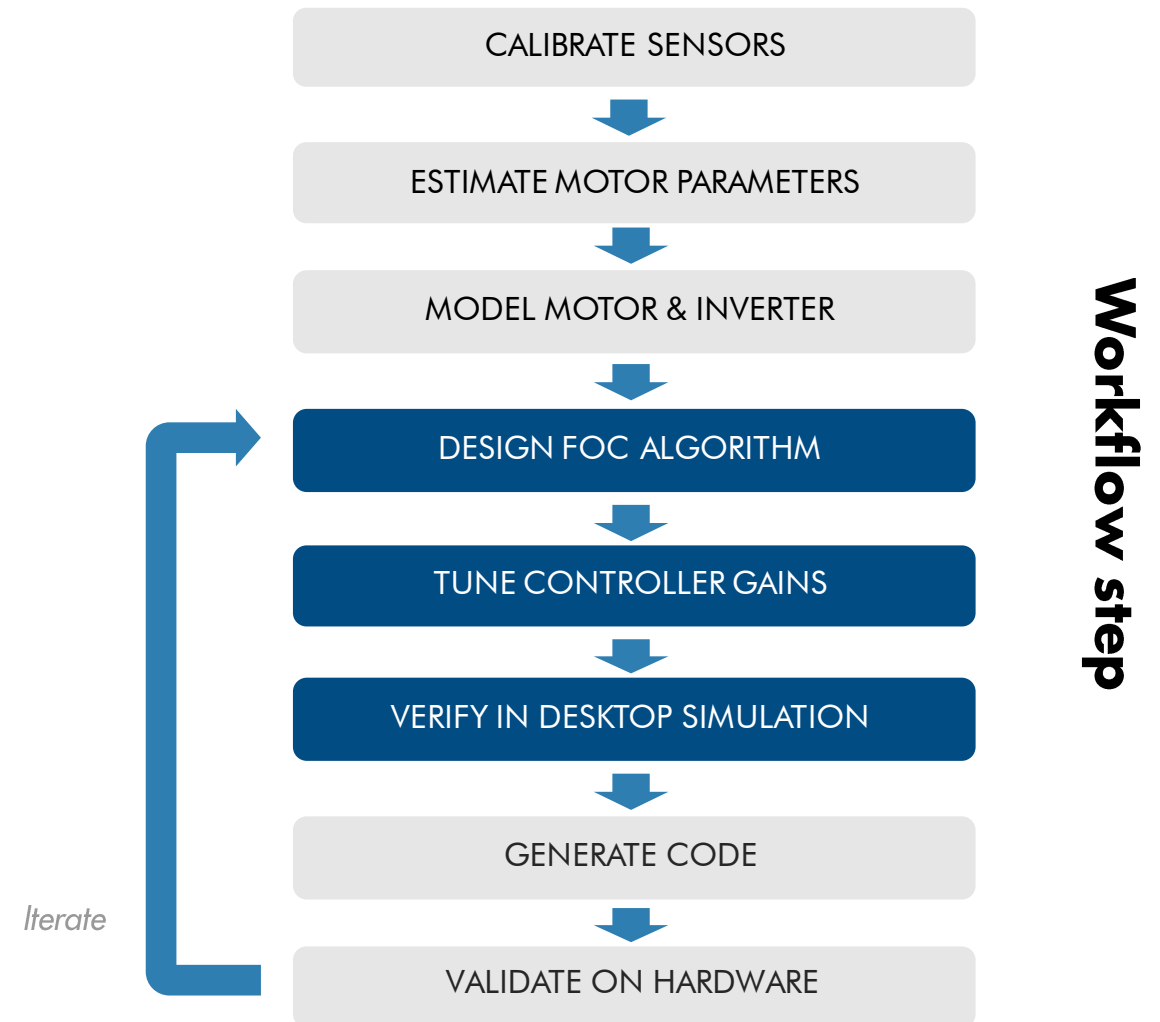




# Agenda

## From Desktop Simulation to Software Deployment

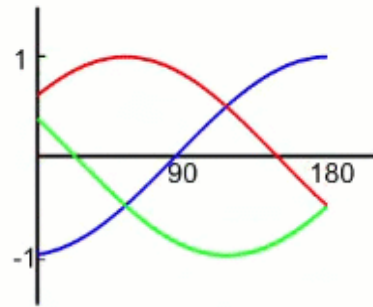
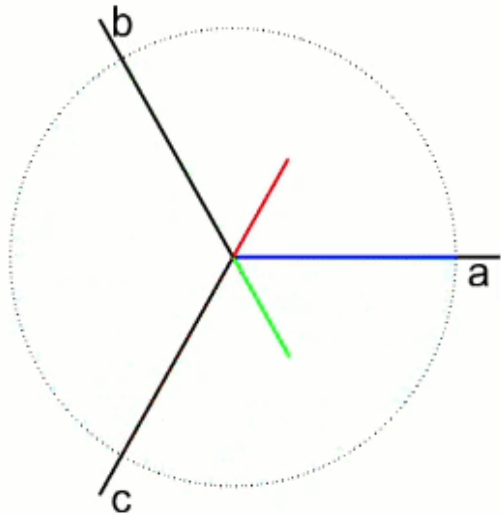
- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Code generation



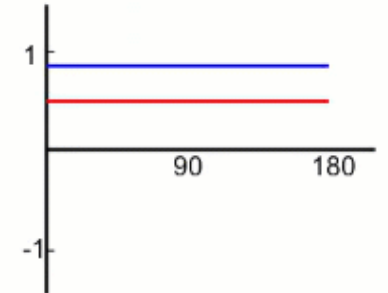
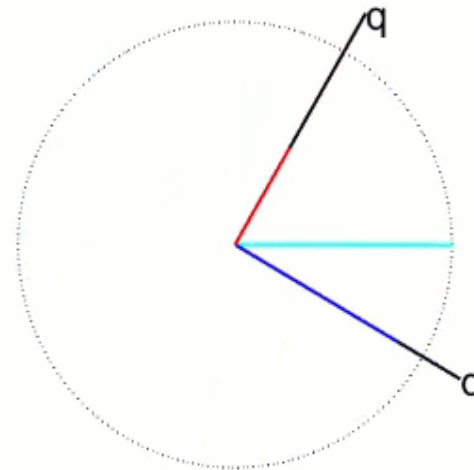
# Modeling Field-Oriented Control (FOC)

## A word about transforms

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



Current control (d, q)

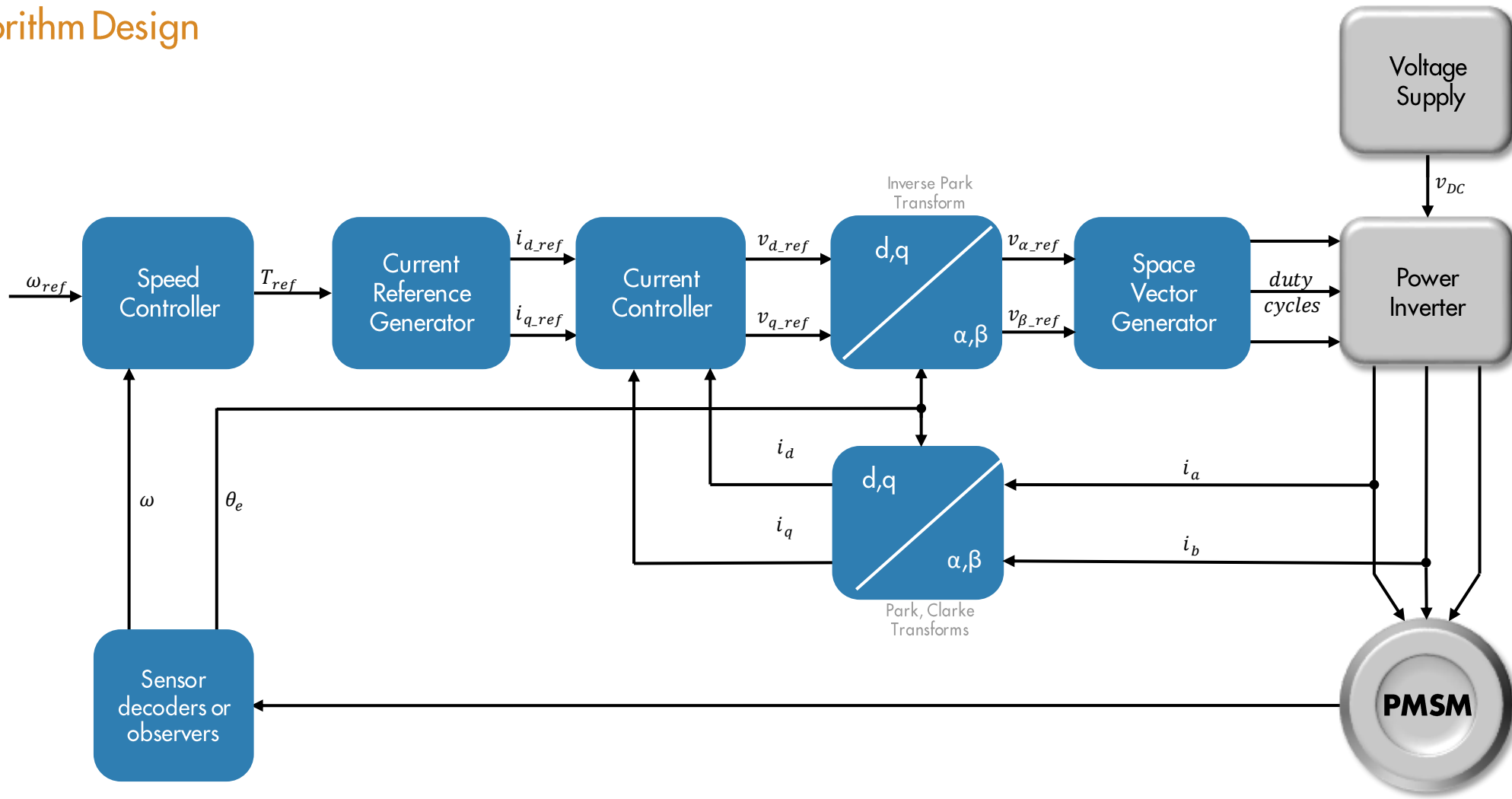


Clarke transform ( $abc \leftrightarrow \alpha\beta$ )

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

# Modeling Field-Oriented Control (FOC)

## Algorithm Design

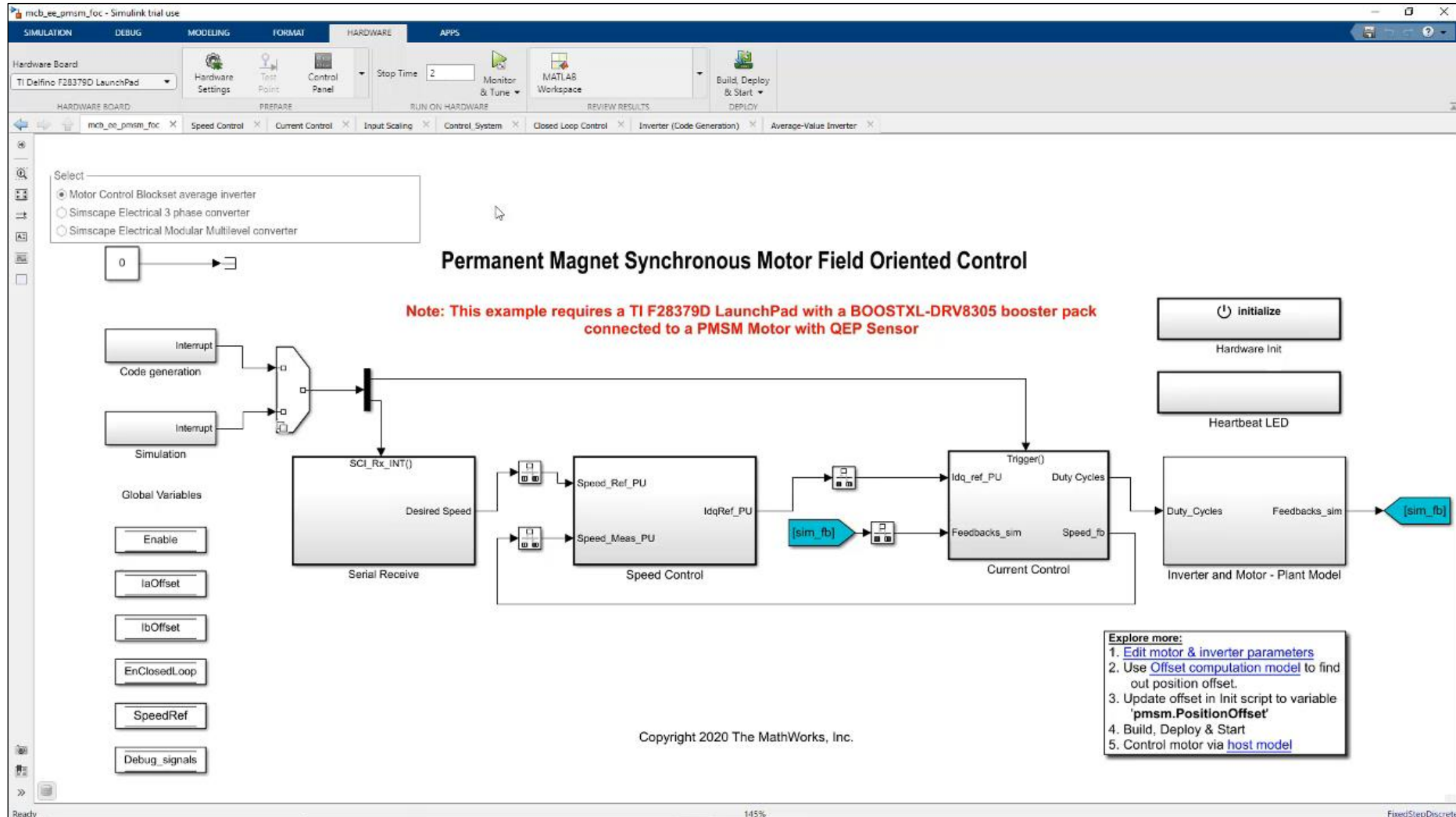


Control algorithm

Physical system

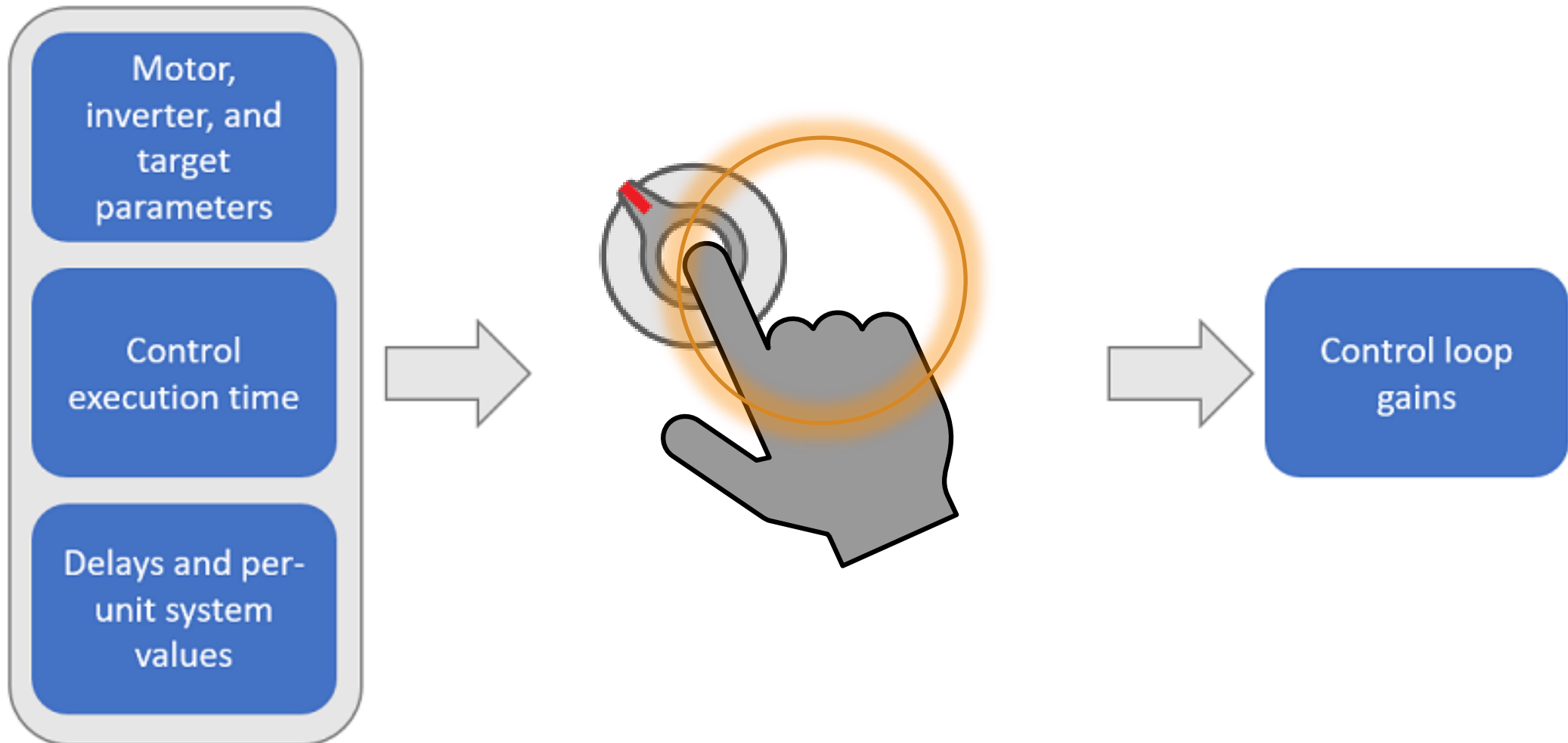
# Modeling Field-Oriented Control (FOC)

## Overview of the model



# Autotuning controller gains

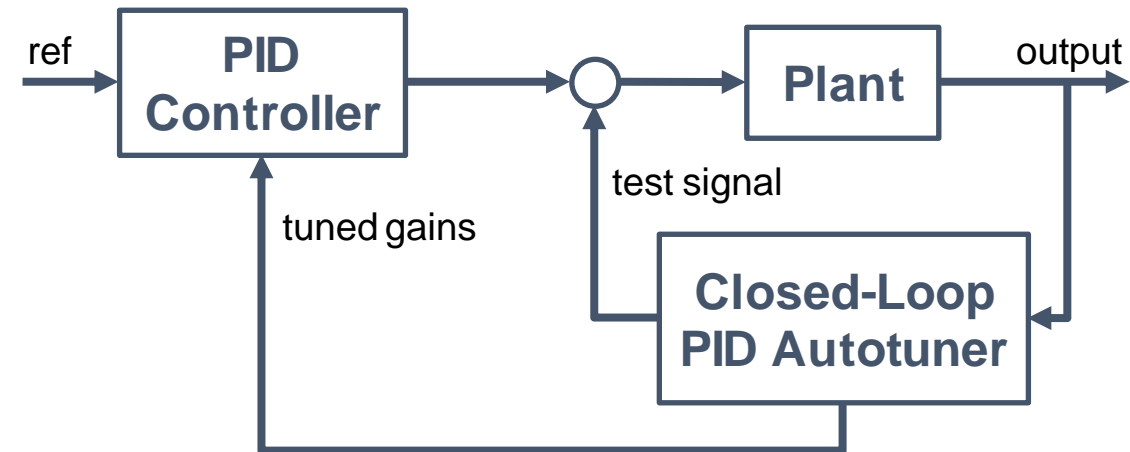
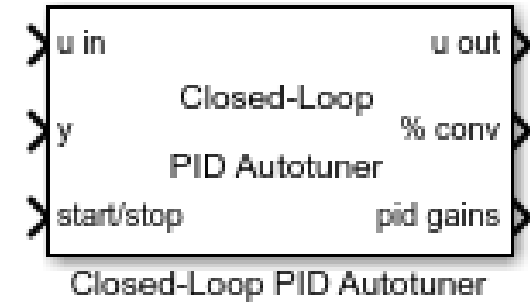
Multiple tuning methods available



# Autotuning controller gains

## Closed-Loop PID Autotuner Block

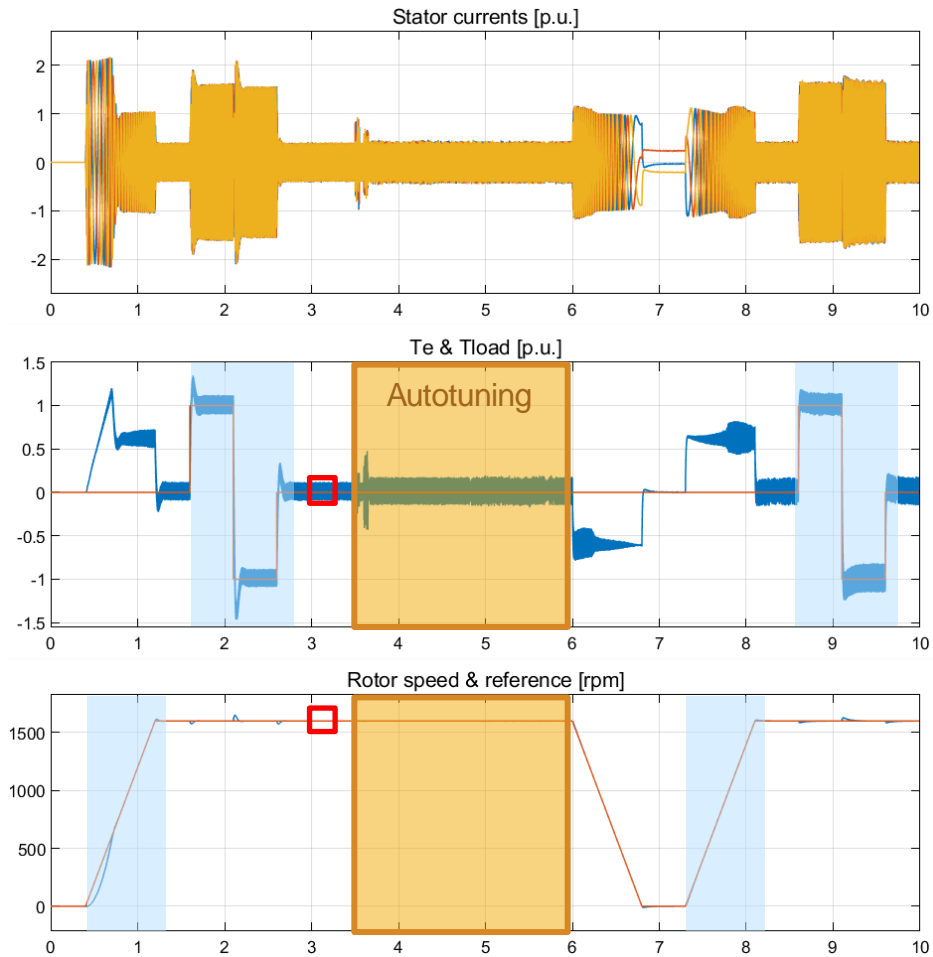
- Algorithm details:
  - Injects a few superimposed sine waves, while maintaining closed-loop operation
  - Collects plant input-output data
  - Estimates frequency response in real-time
  - Tunes PID parameters to satisfy desired bandwidth and phase margin
- Initial stable PID controller is required
- Option to deploy autotuning to embedded processor using Simulink Coder™





# Autotuning controller gains

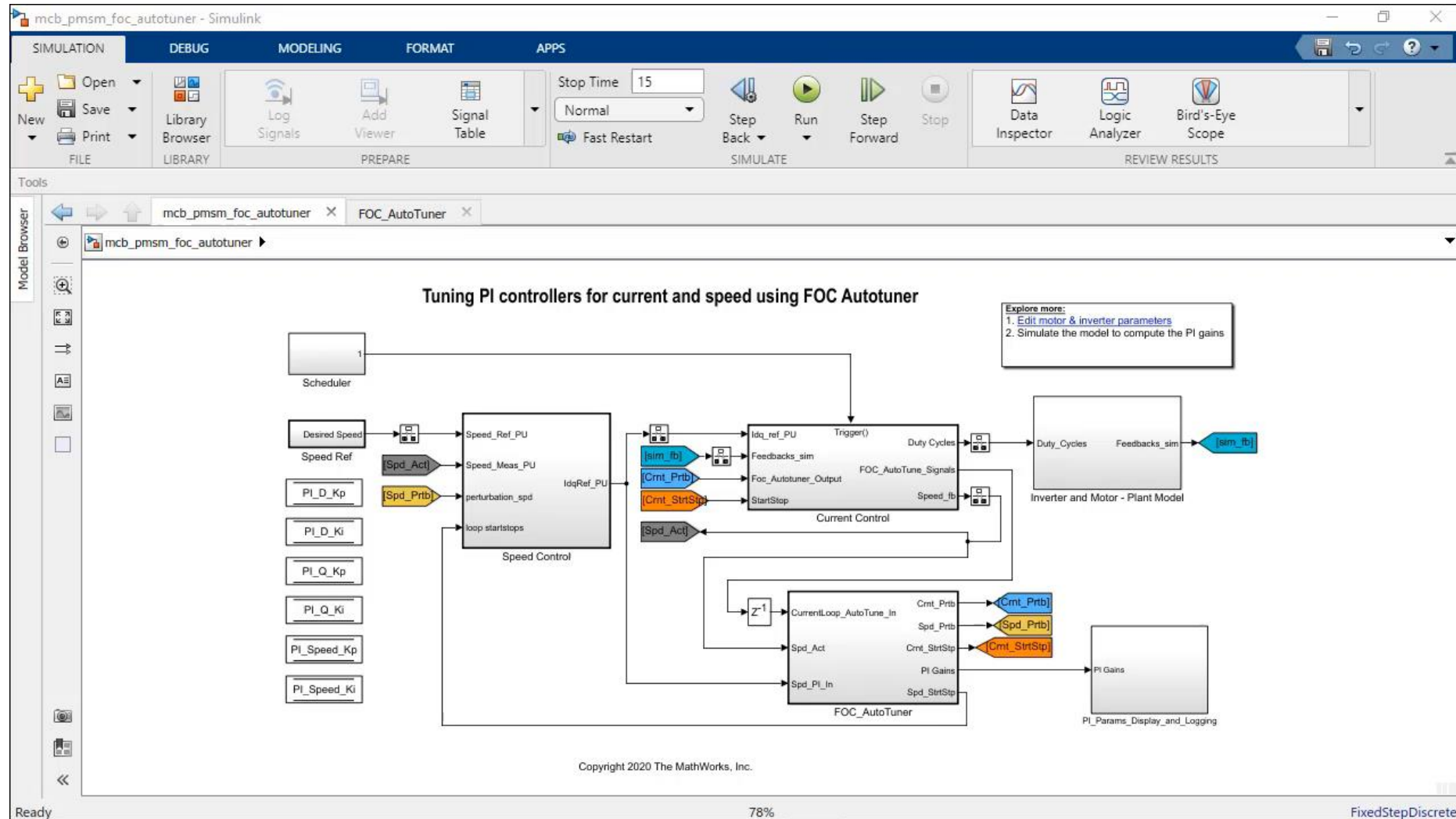
## Tuning All Controller Gains in One Simulation



Motor speed is close to the nominal value while tuning

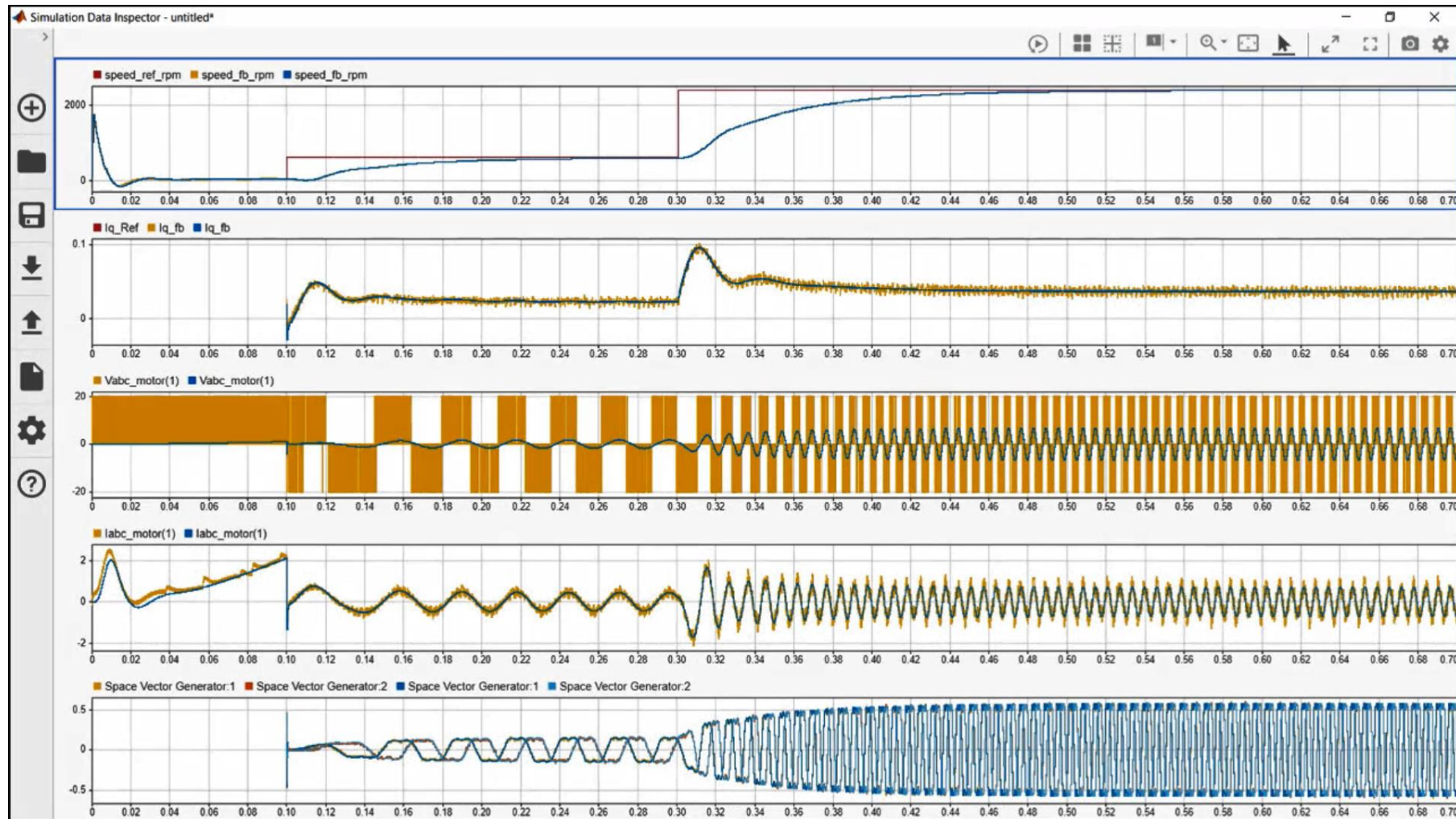
# Autotuning controller gains

## Overview of the workflow



# Autotuning controller gains

## Verify Controller Using Desktop Simulation



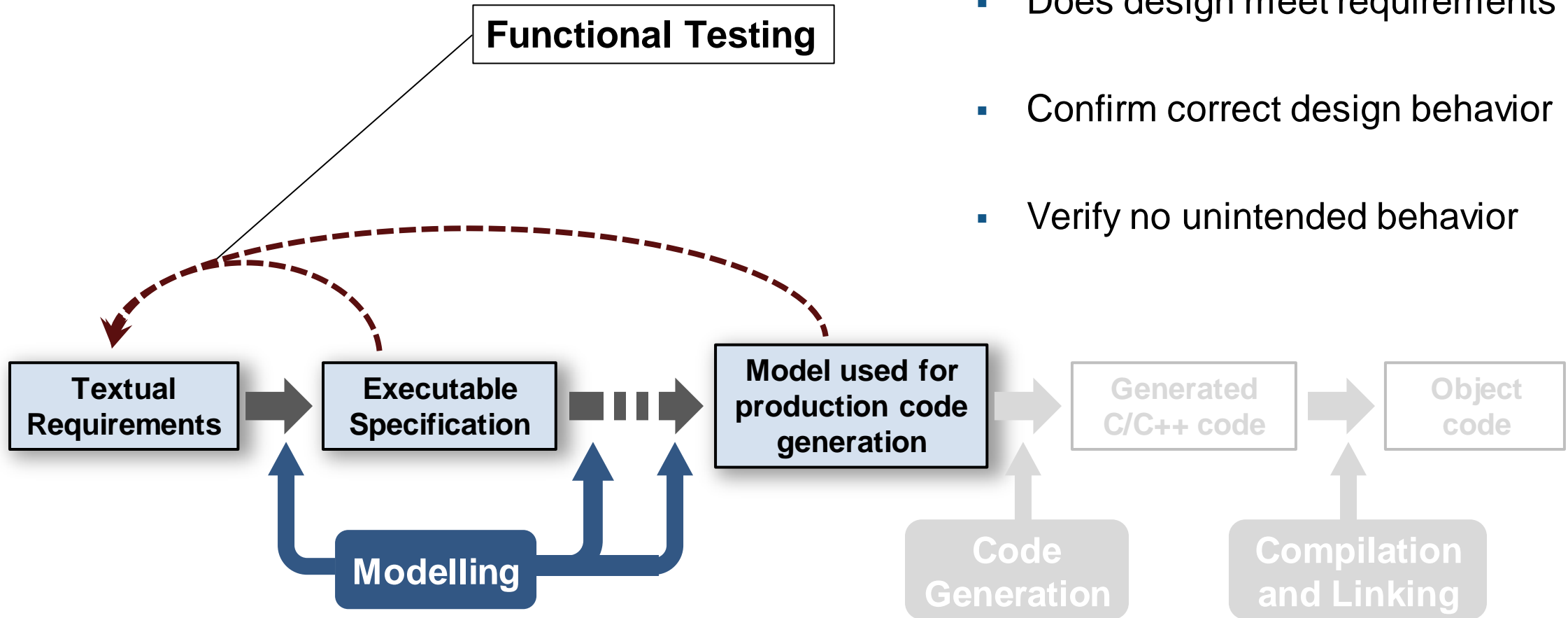
# Verifying Controller

Key take-away

**Simulation models are primary meant  
to support V&V activities**

# Verifying Controller

## Overview



# Verifying Controller

## Functional Testing Process

Author test-cases that are derived from requirements

- Use test harness to isolate component under test
- Test Sequence to create complex test scenarios

Manage tests, execution, results

- Re-use tests for regression
- Automate in Continuous Integration systems such as Jenkins

## Test Sequence

Step	Transition	Next Step
init_step speed = ramp (1); throttle = ramp (1);	1 after (2, sec)	step_2
step_2 speed = 2* ramp (1); throttle = 2* ramp (1);	1 gear == 3	step_3
step_3 peak_speed = speed; peak_throttle = throttle;		

Visual representation of Test Sequence1: A box with 'gear' on the left and 'speed' and 'throttle' on the right. Inside the box, there are three numbered steps (1, 2, 3) and a large green checkmark.

## Test Harness

The screenshot shows a Simulink model titled 'Test Harness basic\_report\_outport - Simulink'. It features a central car model with various input and output blocks. A blue arrow points from the 'Test Harness' title to the car model. A red dashed box highlights a specific component within the harness.

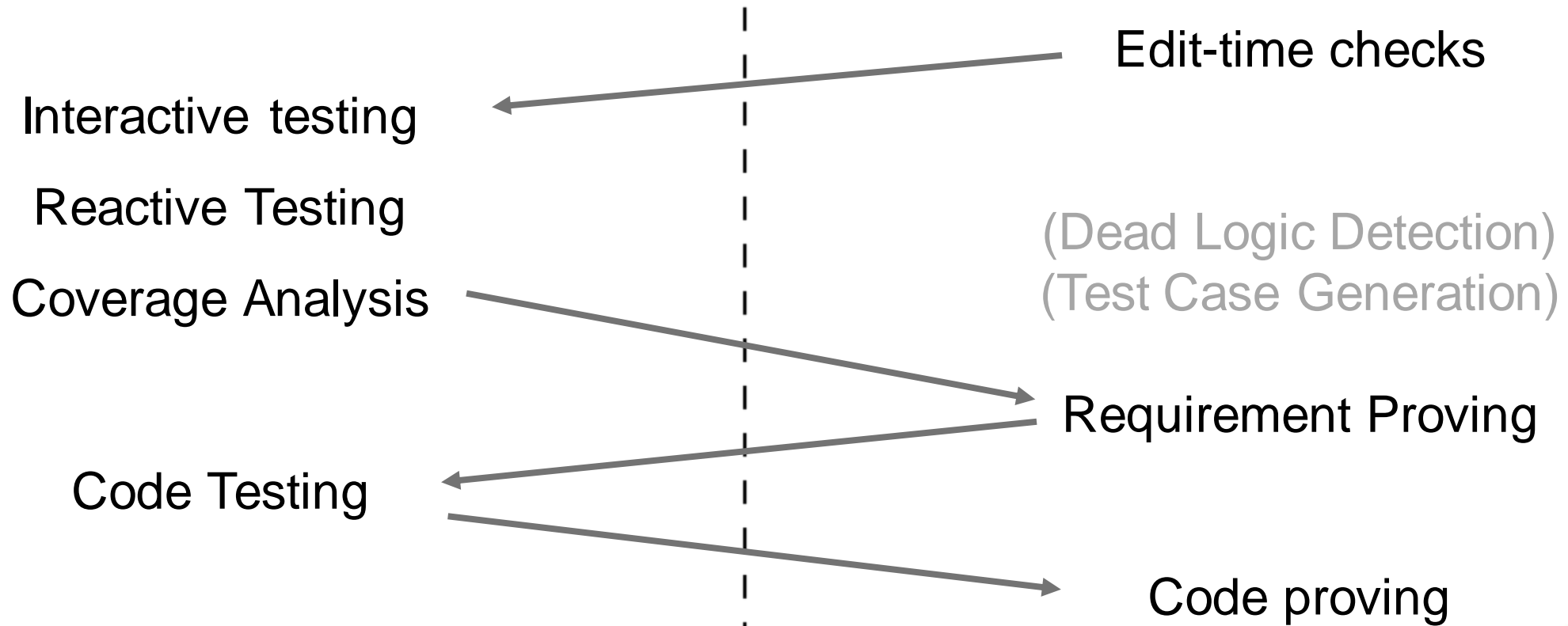
## Test Manager

The screenshot shows the Test Manager interface. It displays a list of test results on the left, including 'Signal Builder Baseline examples', 'Slow Accel', and 'Fast Accel'. On the right, there are two graphs: a 'Comparison' graph showing 'Baseline' (yellow) and 'Compare To' (red) over time, and a 'Tolerance' graph showing 'Tolerance' (green) and 'Difference' (red) over time.



# Verifying Controller

V&V journey



**Dynamic Testing**

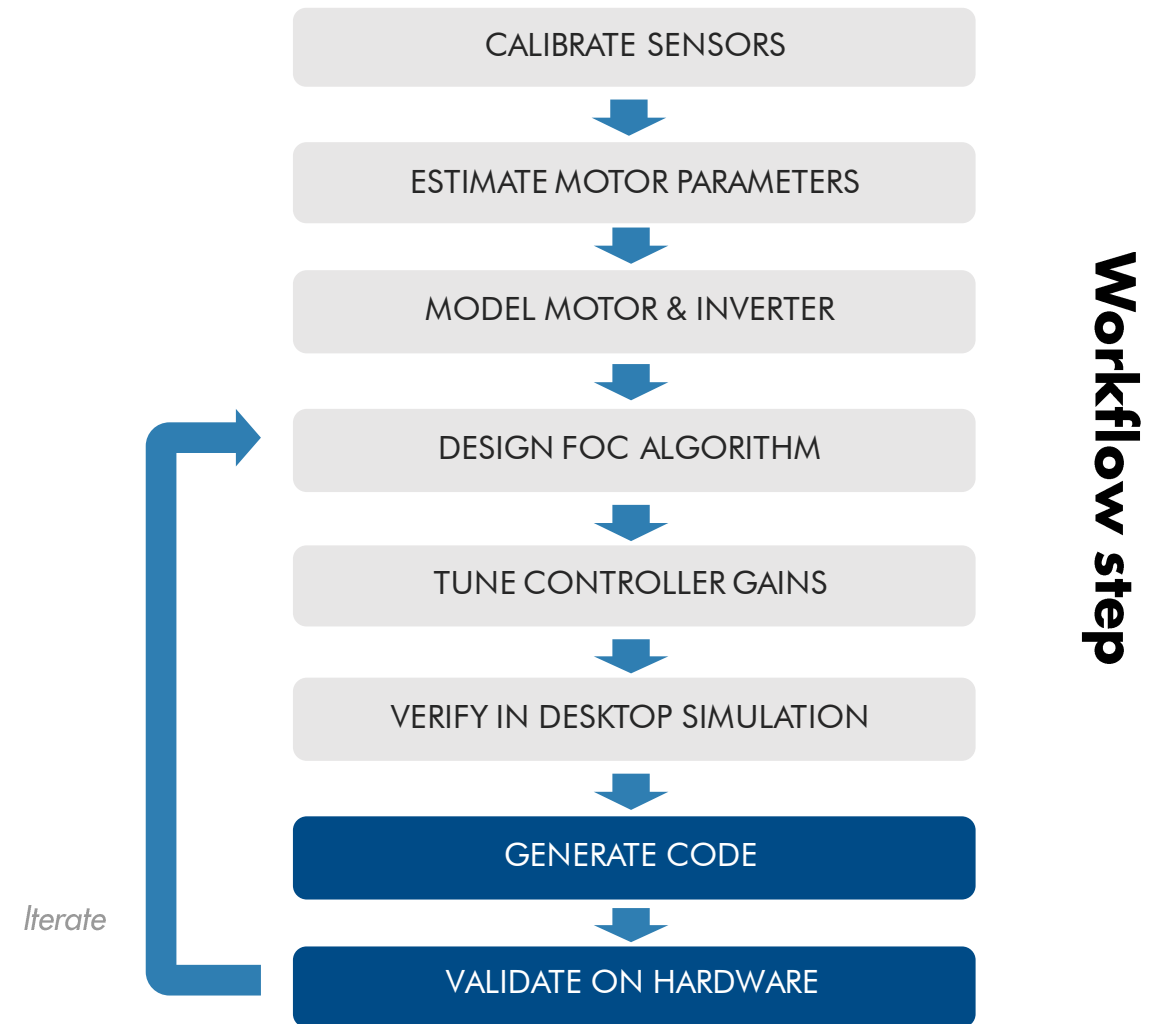
**Static Analysis**



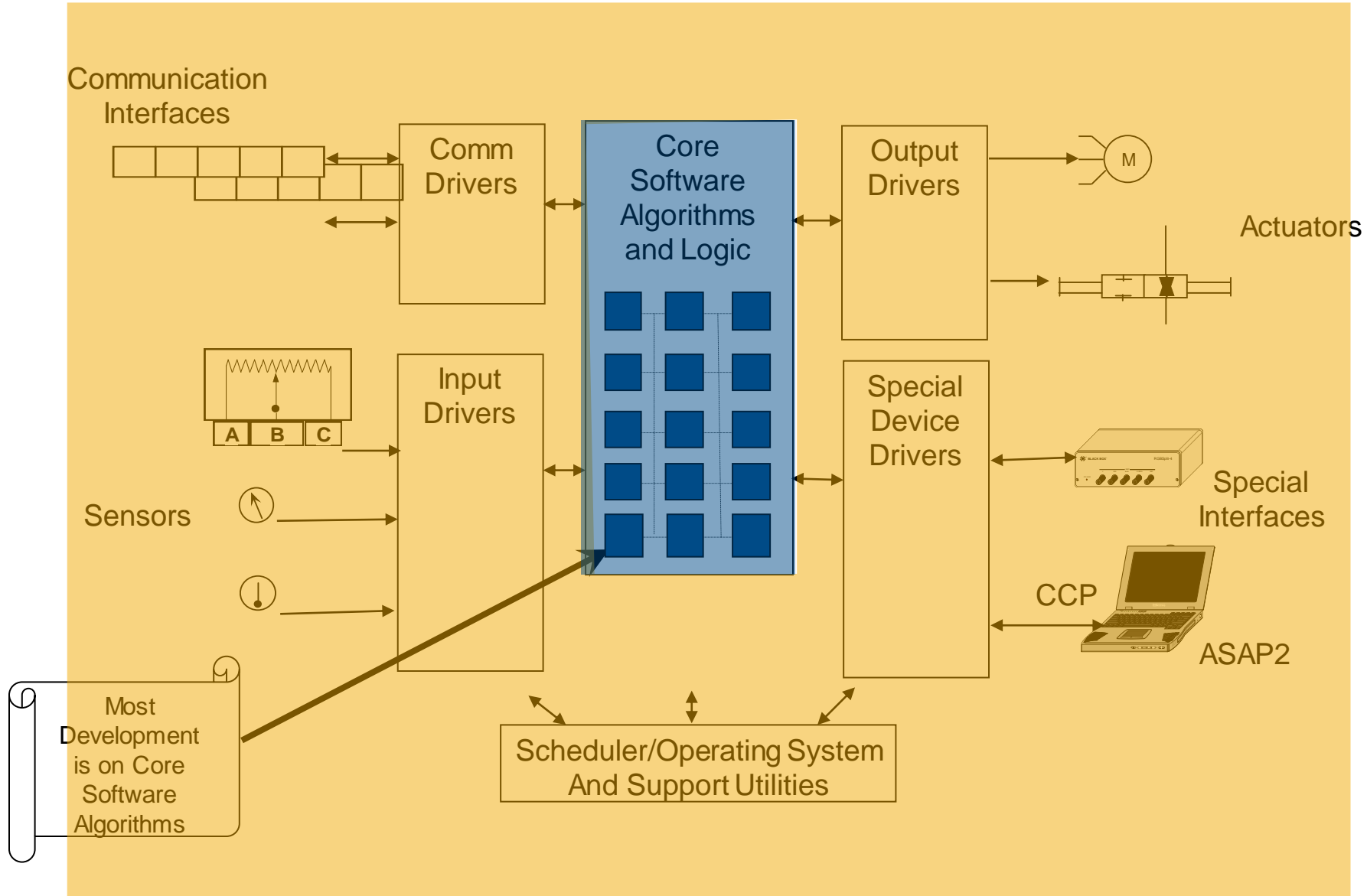
# Agenda

## From Desktop Simulation to Software Deployment

- Plant modeling
  - Sensors Calibration
  - Motor Parameters Estimation
  - Motor and Inverter Model
- Algorithm design with simulation
  - Field-Oriented control
  - Autotuning control gain
  - Verifying controller
- Software deployment
  - Code generation



# 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();

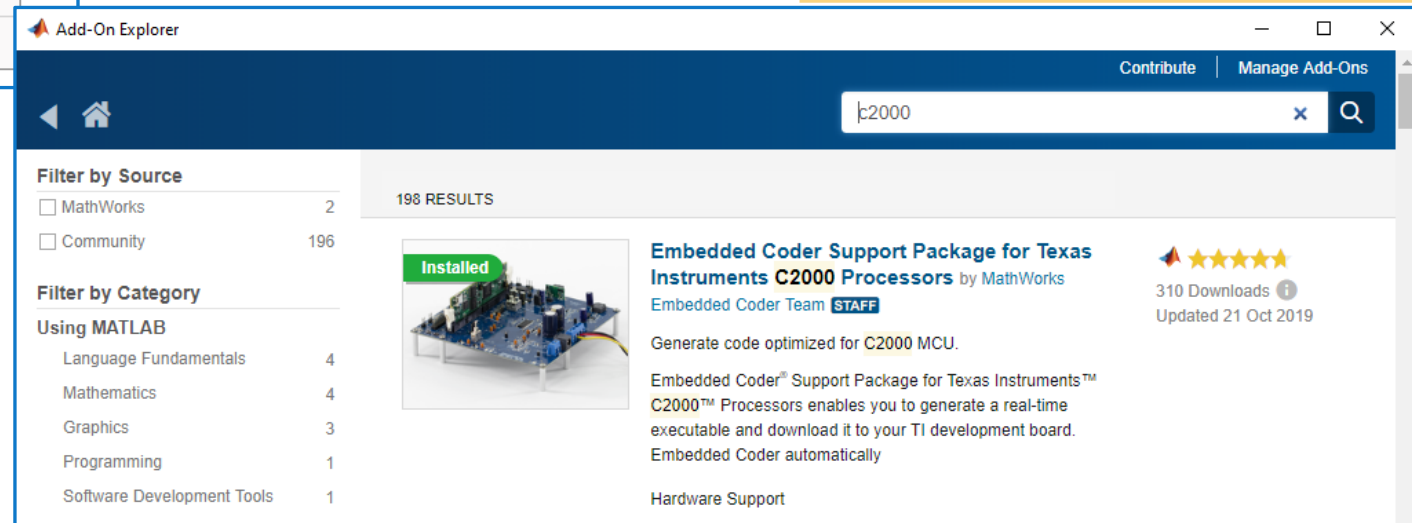
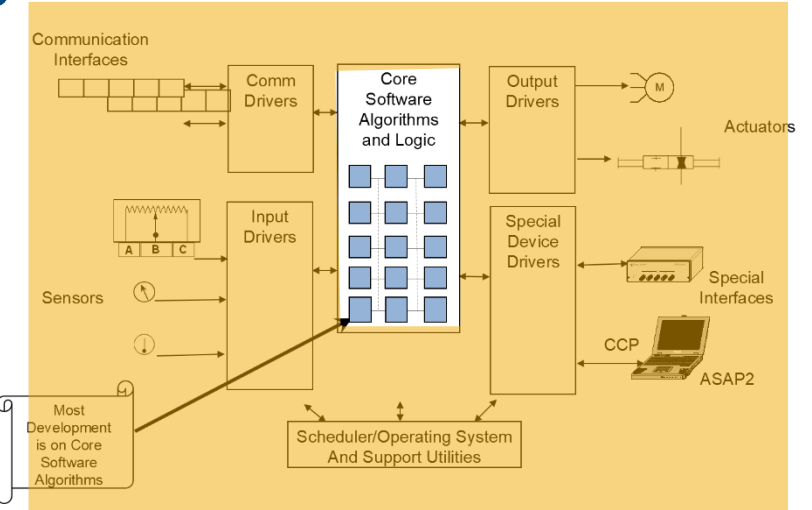
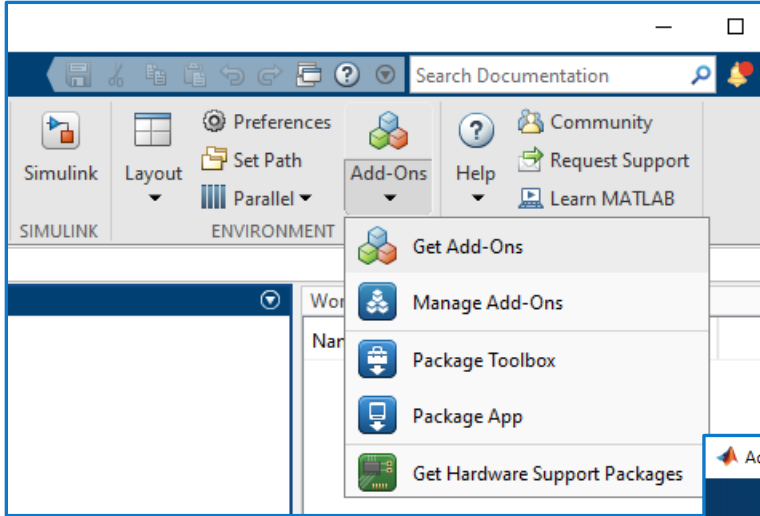
    while(1) {
    }
}
```

```
interruptServiceRoutine()
{
    readAdcCountFromDriver();
    readEncoderCountFromDriver();

    controller();

    writePwmCountToDriver();
}
```

# Embedded Coder Hardware Support Packages



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



# MathWorks TI C2000 Support Package for Embedded Coder

## Supported devices:

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



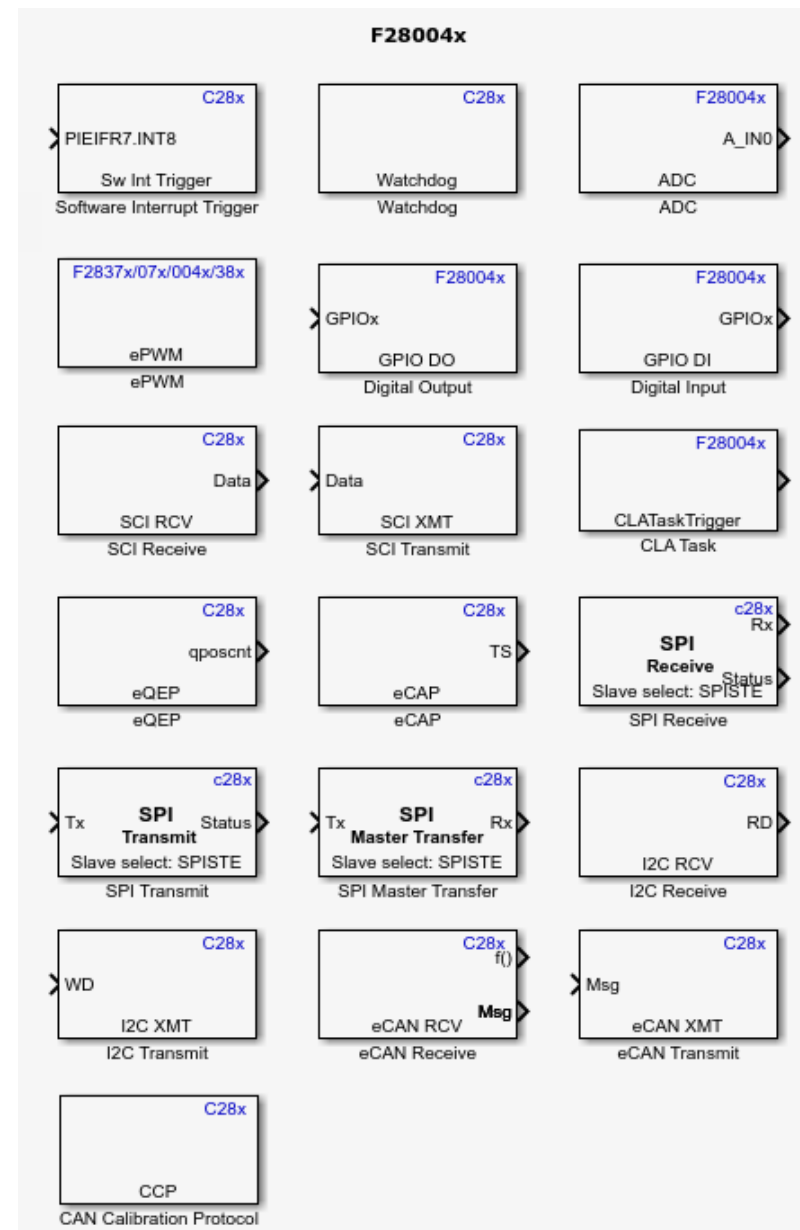
F28379D LaunchPad

## Scheduling the generated code:

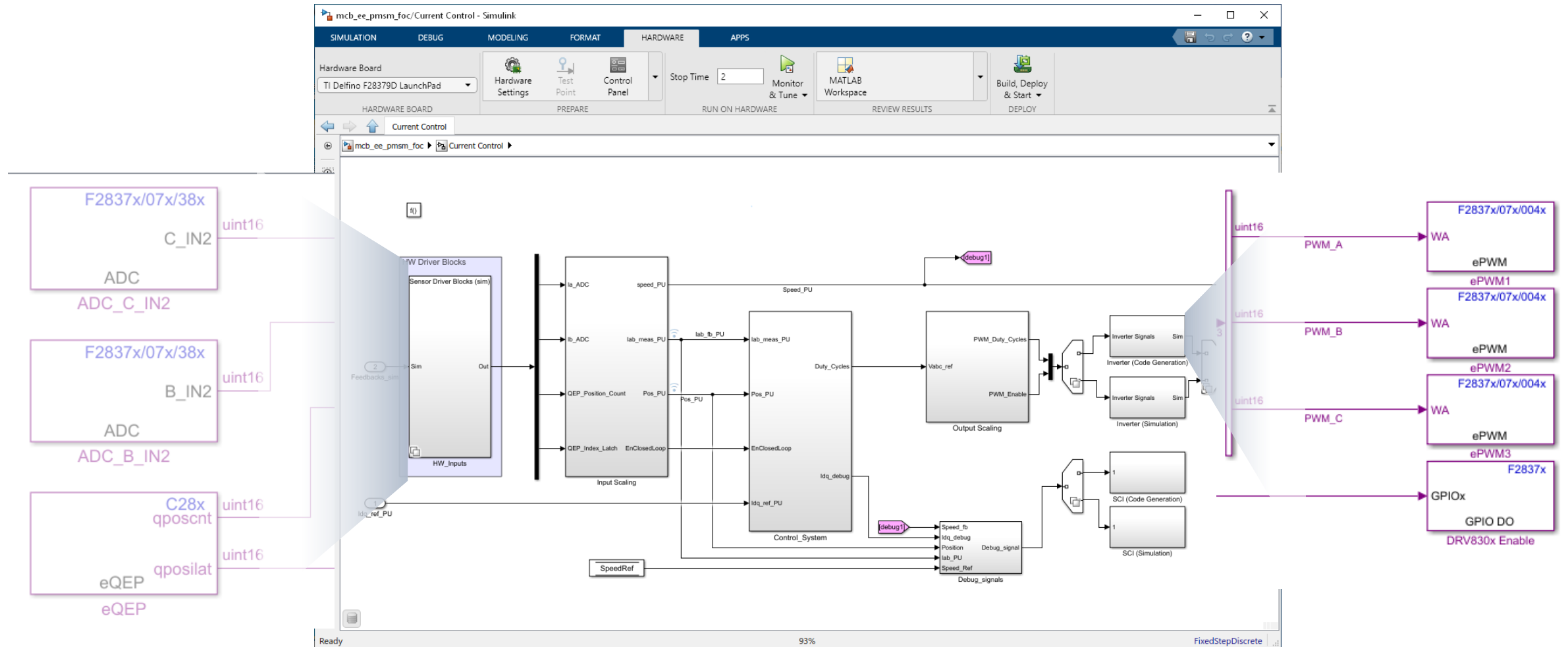
- Periodic tasks
- Idle tasks
- Interrupts (Hardware, Software)
- Advanced concepts:
  - Pre-emptive rate-monotonic scheduler
  - Base rate interrupt replacement
  - Peripheral triggers (launch A/D conversion from PWM)
  - Running on the CLA
  - Loading in Flash, running in RAM
  - Using DMA

# 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



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

Configuration Parameters: mcb\_ee\_pmsm\_foc/Configuration (Active)

Search

Solver  
Data Import/Export  
Math and Data Types  
▶ Diagnostics  
Hardware Implementation  
Model Referencing  
Simulation Target  
▶ Code Generation  
Coverage  
Simscape  
▶ Simscape Multibody

Hardware board: TI Delfino F28379D LaunchPad

Code Generation system target file: [url.tlc](#)

Device vendor: Texas Instruments Device type: C2000

▶ Device details

Feature set for selected hardware board:

Embedded Coder Hardware Support Package  
 SoC Blockset

Hardware board settings

▼ Target hardware resources

Groups

Build options Communication interface: serial

Clocking SCI module: SCI\_A

ADC\_A

ADC\_B Serial port in MATLAB preferences: COM3 Refresh

ADC\_C

ADC\_D

CMPSS

DAC

ePWM

eCAP

eQEP

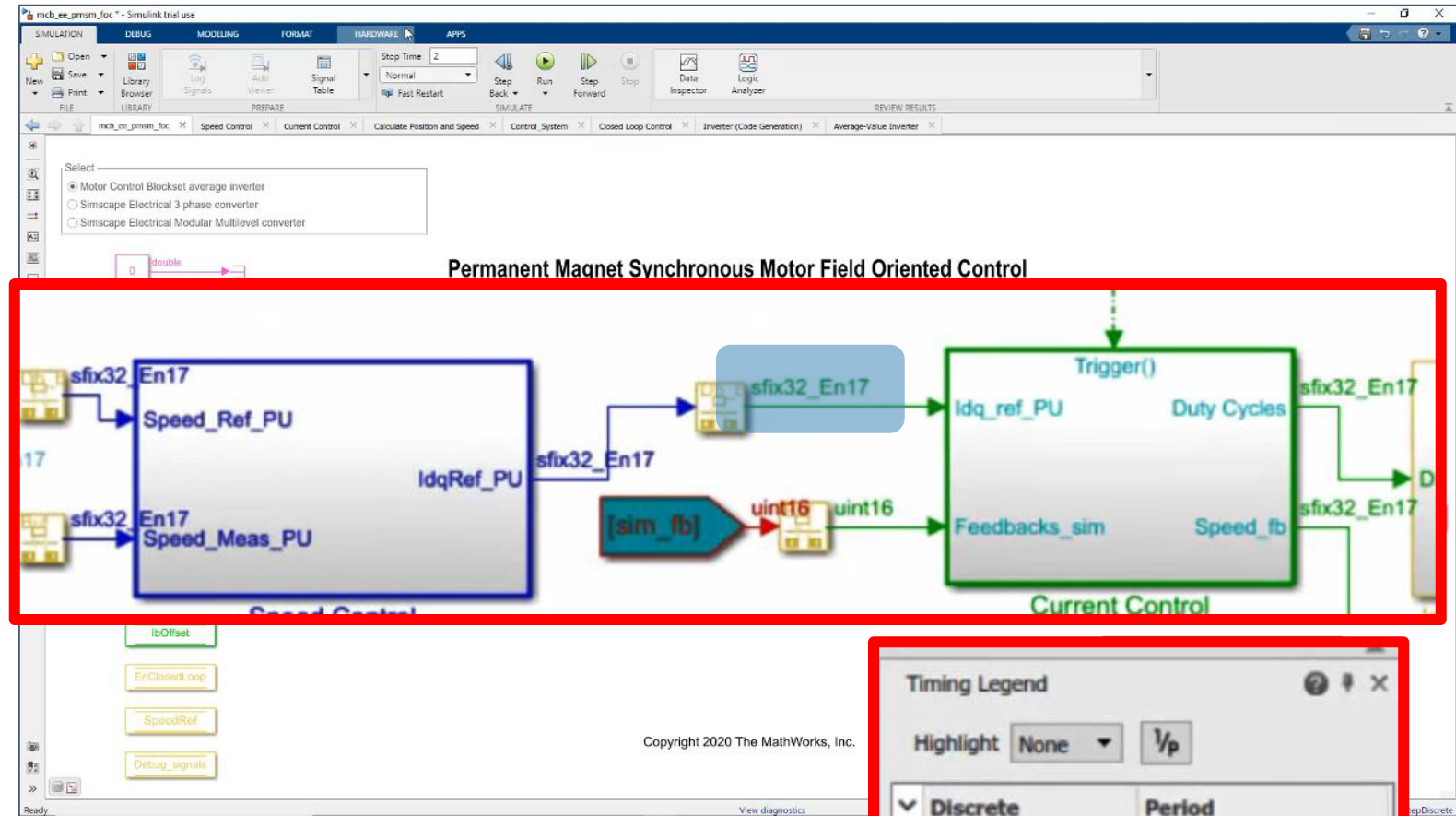
I2C\_A

I2C\_B

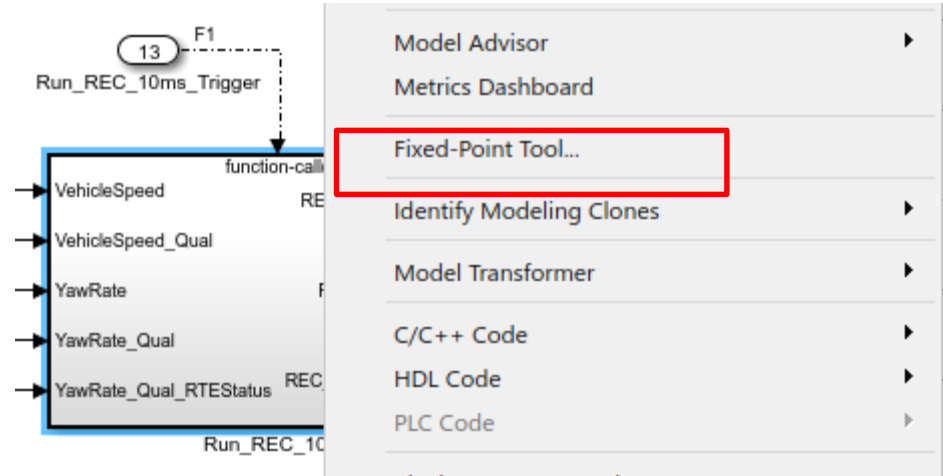
SCI\_A

# Deployment on the Target

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



# Fixed-Point conversion



- Run the tool on the system to convert
- Chose your conversion method

**ITERATIVE FIXED-POINT CONVERSION**

Setup

**System Under Design (SUD)**

Select the system to analyze or convert.

Selected system under design: Swc\_REC\_step\_4\_Fixed\_Point\_With\_ToolsTest/Run\_REC\_10ms

- Simulink Root
  - Swc\_REC\_step\_4\_Fixed\_Point\_With\_ToolsTest
    - Run\_REC\_10ms

**Range Collection Mode**

Select whether to collect ranges for objects in the model through simulation, or through static analysis that derives the ranges.

Simulation ranges

Derived ranges

Simulation with Range Analysis

**Simulation Inputs**

Specify inputs for simulations. You can choose to use the current model inputs, or select a Simulink.SimulationInput object from the base

Simulation inputs: Use default model inputs Refresh

**Signal Tolerances**

Specify tolerances for signals in your model that have signal logging enabled. After simulating with embedded types, the Workflow Browser specified signal tolerances.

Filter signal list:

Signal Name	Abs Tol	Rel Tol
LIB_Mdl_Saturation_Opp:1		
Rescaler5:1		
Rescaler6:1		
COMP_REC_YawRate_Angle:1		
COMP_REC_YawRate_Angle:2		
COMP_REC_YawRate_Angle:3		
REC_VSpeed_map_X_kf	0	
REC_YawRate_Weight	0.0048828125	
REC_YawRate_SteerAngle	0.01953125	



# Fixed-Point conversion

- Prepare the environment
- Configure your options
- Accept or modify the datatype proposition

**EXPLORE**

Settings

**PROPOSE**

Propose:

Propose signedness:

Safety margin for simulation min/max (%):

**CONVERT TO FIXED POINT**

Convert double/single types:

Convert inherited types:

Default word length:

Default fraction length:

Original Data Type	Word Length	Fraction Length
Double/Single	→ 16	Will propose
Inherited	→ 16	Will propose
Fixed point	→ No change	Will propose

WORKFLOW BROWSER

- Setup
- Preparation Results

Selected system under design: Swc\_REC\_step\_4\_Fixed\_Point\_With\_ToolsTest/Run\_REC\_10ms

Select a result below for more information

Selection	Check	Status
<input checked="" type="radio"/>	Create Restore Point	✓
<input type="radio"/>	Hardware Implementation Consistency	✓
<input type="radio"/>	Diagnostic Settings	✓
<input type="radio"/>	Unsupported Constructs	✓
<input type="radio"/>	System Under Design Boundary	✓

Progress

100%

Preparation is complete for the selected system under design

**PREPARATION DETAILS**

**Check Details**

To ensure your original design is saved before making fixed-point data type changes, create a restore point for the model.

**Check Status**

A restore point has previously been created for this model. To restore the model to this state, click the Restore Original Model button.

WORKFLOW BROWSER

- Setup
- Preparation Results
- BaselineRun

Results

Name	CompiledDT	SpecifiedDT	ProposedDT	Accept	SimM
COMP_REC_YawRate_Angle/Rescaler6	double	Inherit: Inherit ...	fixdt(0,32,15)	✓	0
COMP_REC_YawRate_Angle/Sum : Accumulator	double	Inherit: Inherit ...	n/a	✓	0
COMP_REC_YawRate_Angle/Sum : Output	double	Inherit: Inherit ...	fixdt(0,16,15)	✓	0
COMP_REC_YawRate_Angle/Switch1	double	Inherit: Inherit ...	fixdt(0,16,15)	✓	0
COMP_REC_YawRate_Angle/VehicleSpeed		Inherit: auto	fixdt(0,16,7)	✓	
COMP_REC_YawRate_Angle/VehicleSpeed_Qual		Inherit: auto	fixdt(0,16,14)	✓	
COMP_REC_YawRate_Angle/YawRate		Inherit: auto	fixdt(1,16,11)	✓	
COMP_REC_YawRate_Angle/YawRate_Qual		Inherit: auto	fixdt(0,16,14)	✓	
COMP_REC_YawRate_Angle/YawRate_Qual_RTETStatus		Inherit: auto	fixdt(1,16,4)	✓	
REC_VSpeed_map_X_kf		fixdt(0,32,0)	fixdt(0,32,15)	✓	
REC_YawRate_SteerAngle		Inherit: auto	n/a		
REC_YawRate_Weight		Inherit: auto	n/a		
VehicleSpeed		Inherit: auto	fixdt(0,16,7)	✓	
VehicleSpeed_Qual		Inherit: auto	fixdt(0,16,14)	✓	

MODEL HIERARCHY

- Simulink Root
  - Data Objects
  - Swc\_REC\_step\_4\_Fixed
    - Run\_REC\_10ms

Visualization of Simulation Data

Histograms of all results in the model

**RESULT DETAILS**

Swc\_REC\_step\_4\_Fixed\_Point\_With\_ToolsTest/Run\_REC\_10ms/COMP\_REC\_YawRate\_Angle/Rescaler6

**Proposed Data Type Summary**

Property	Proposed Data Type	Specified DT
Data Type	fixdt(0,32,15)	Inherit: Inherit
Minimum	0	
Maximum	131071.99996948242	
Precision	3.0517578125e-05	

**Ranges used for proposal**

Property	Minimum	Maximum
Shared Simul...	0	65536
Simulation	0	65536

**Visualization of Simulation Data using fixdt(0,32,15)**

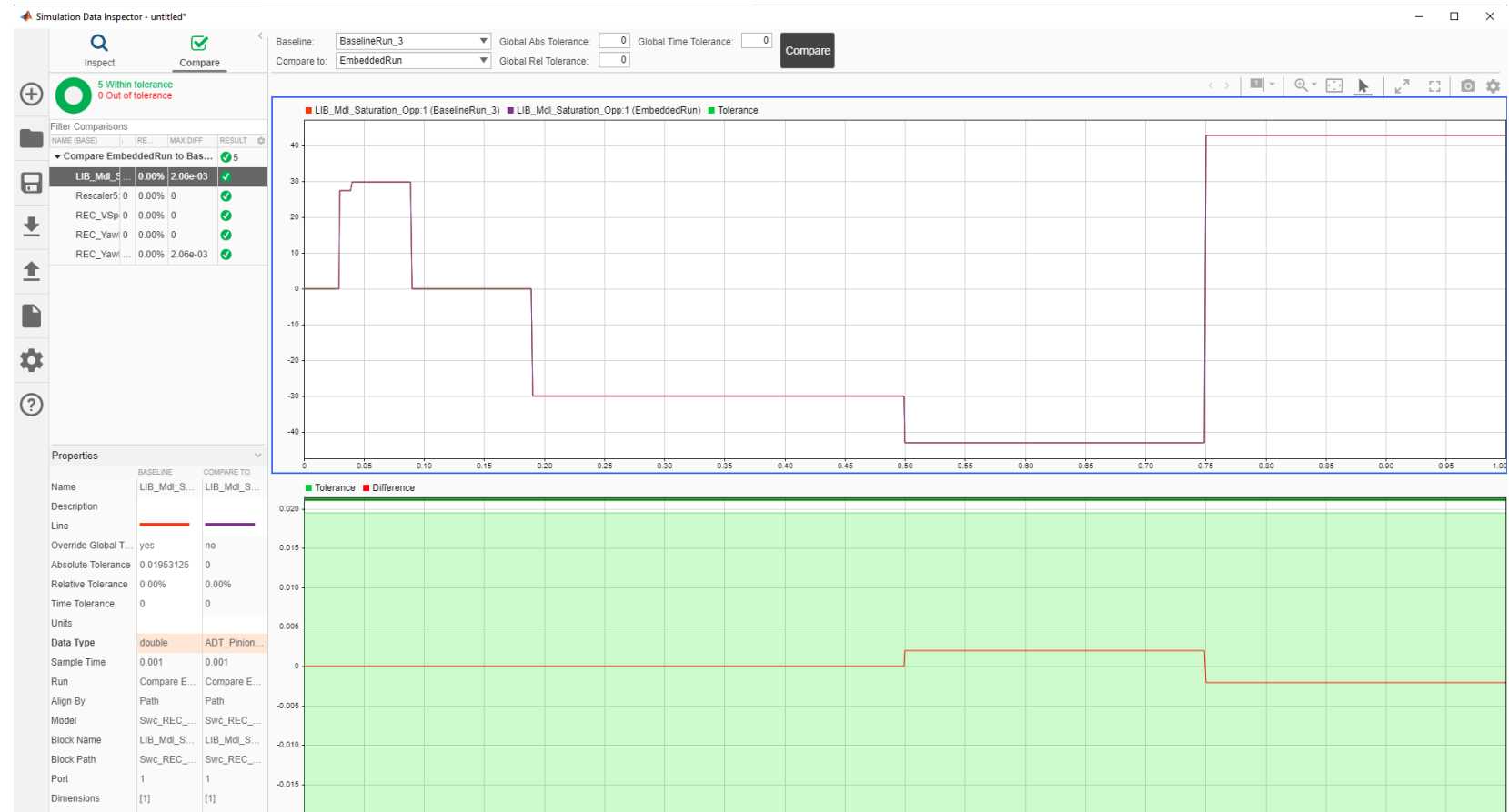
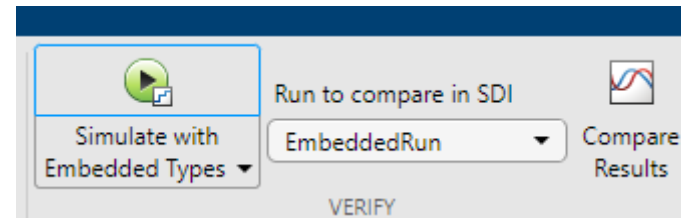
Values	Potential Overflows	In-Range	Potential Underflows
Positive	0	71	0
Negative	0	0	0
Zero	0	230	0

**Proposal Details**

- There is a requirement for the data type of this result to match the data type of other results.
  - [Highlight Elements Sharing Same Data Type](#)

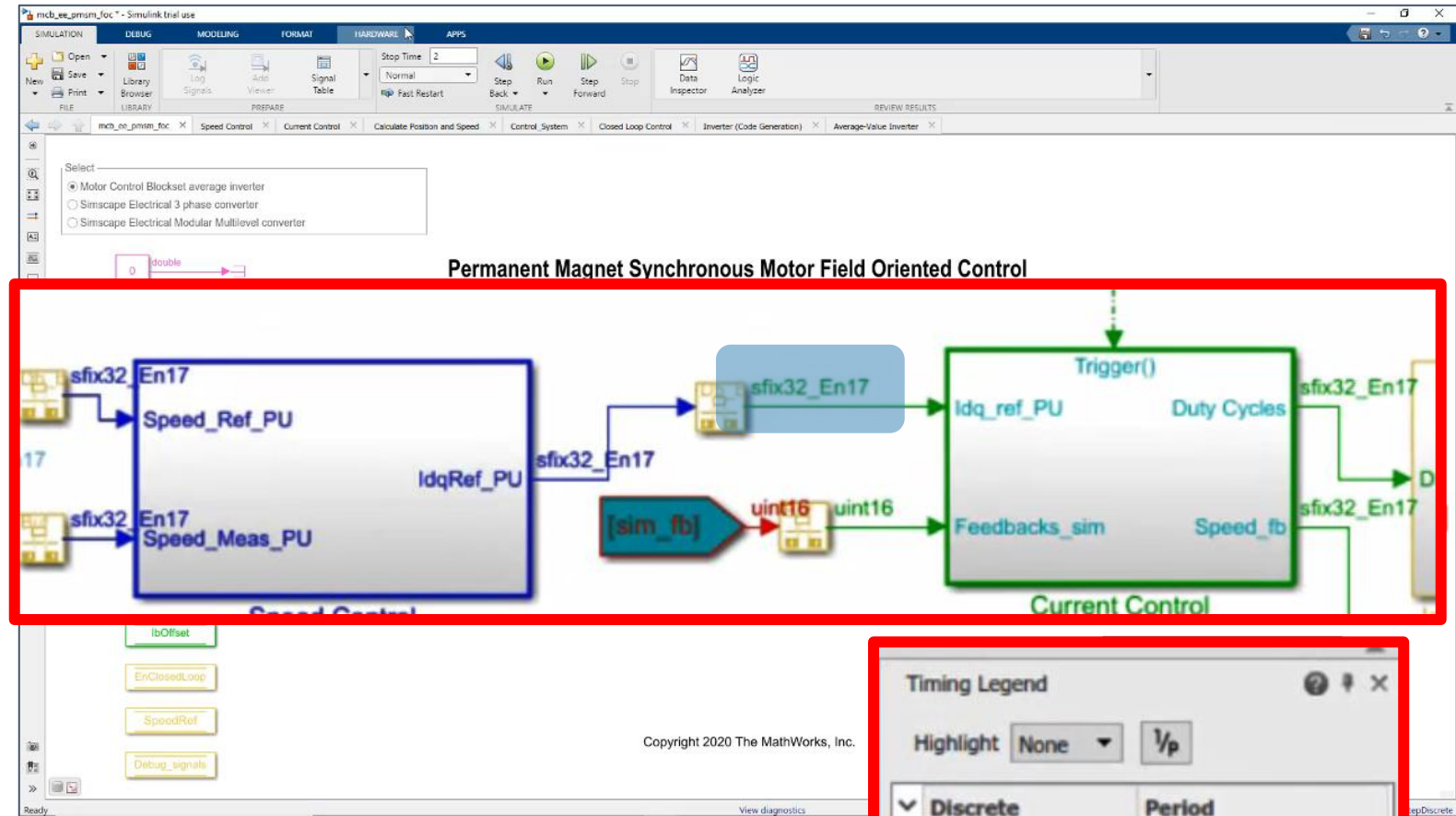
# Fixed-Point conversion

- Run again with your new datatype
- Compare automatically with floating point results



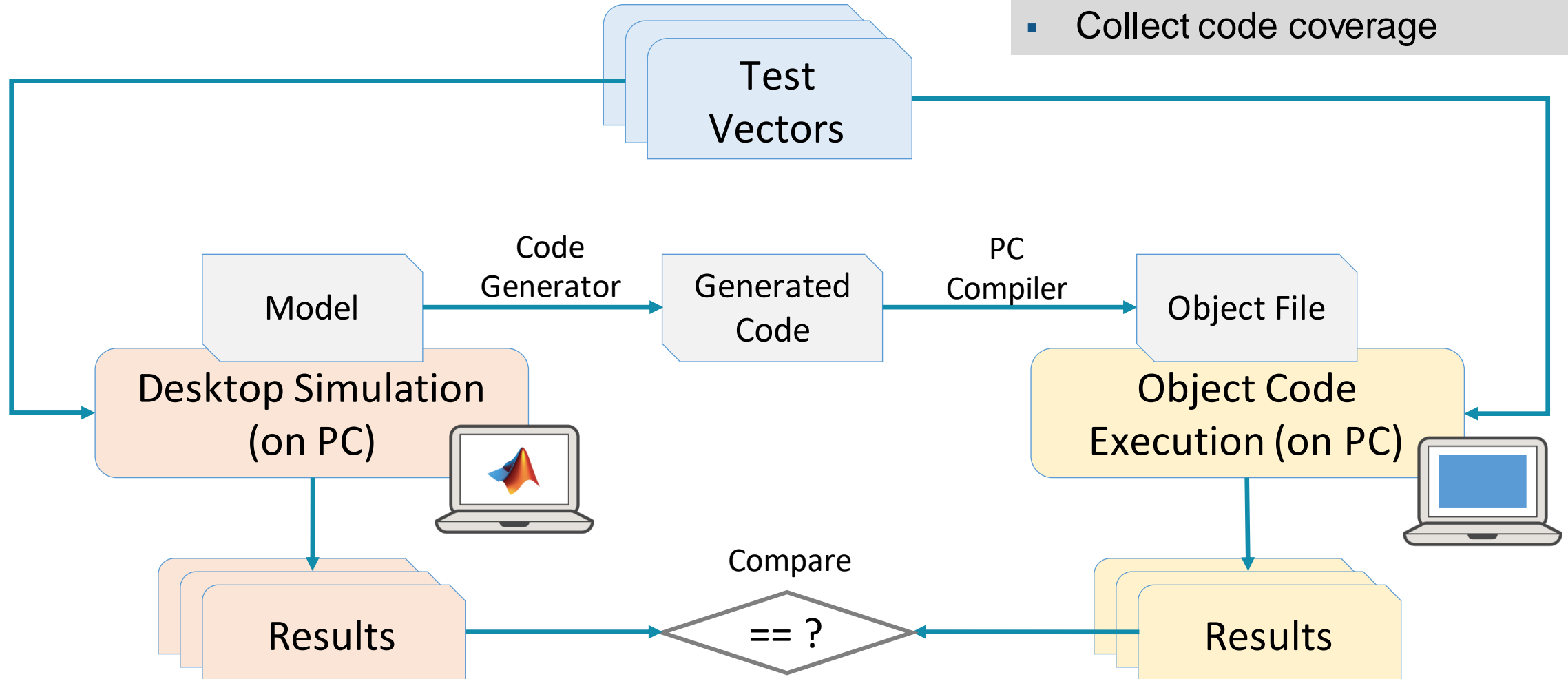
# Deployment on the Target

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

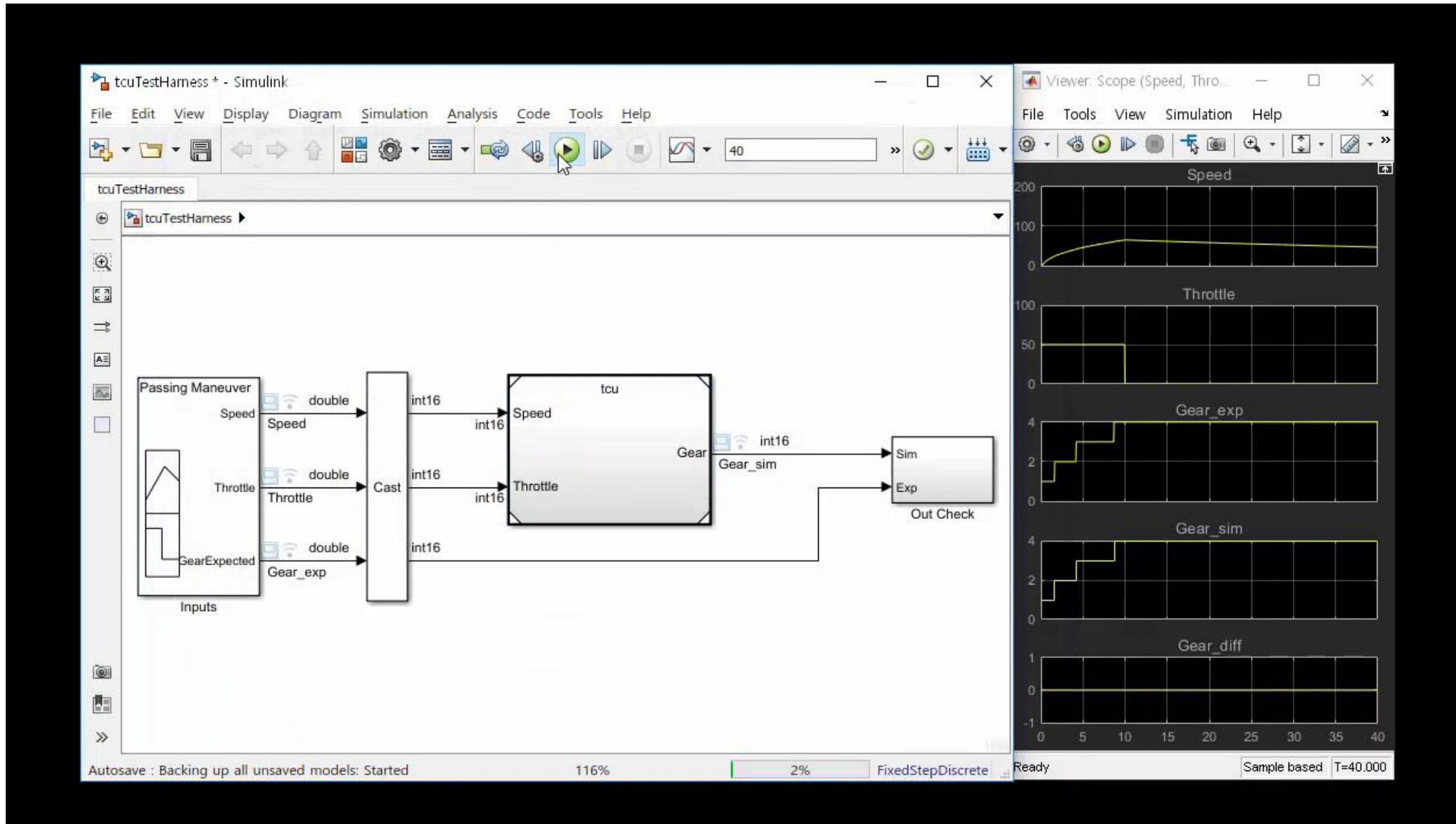


# Software-In-the-Loop (SIL) Testing

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

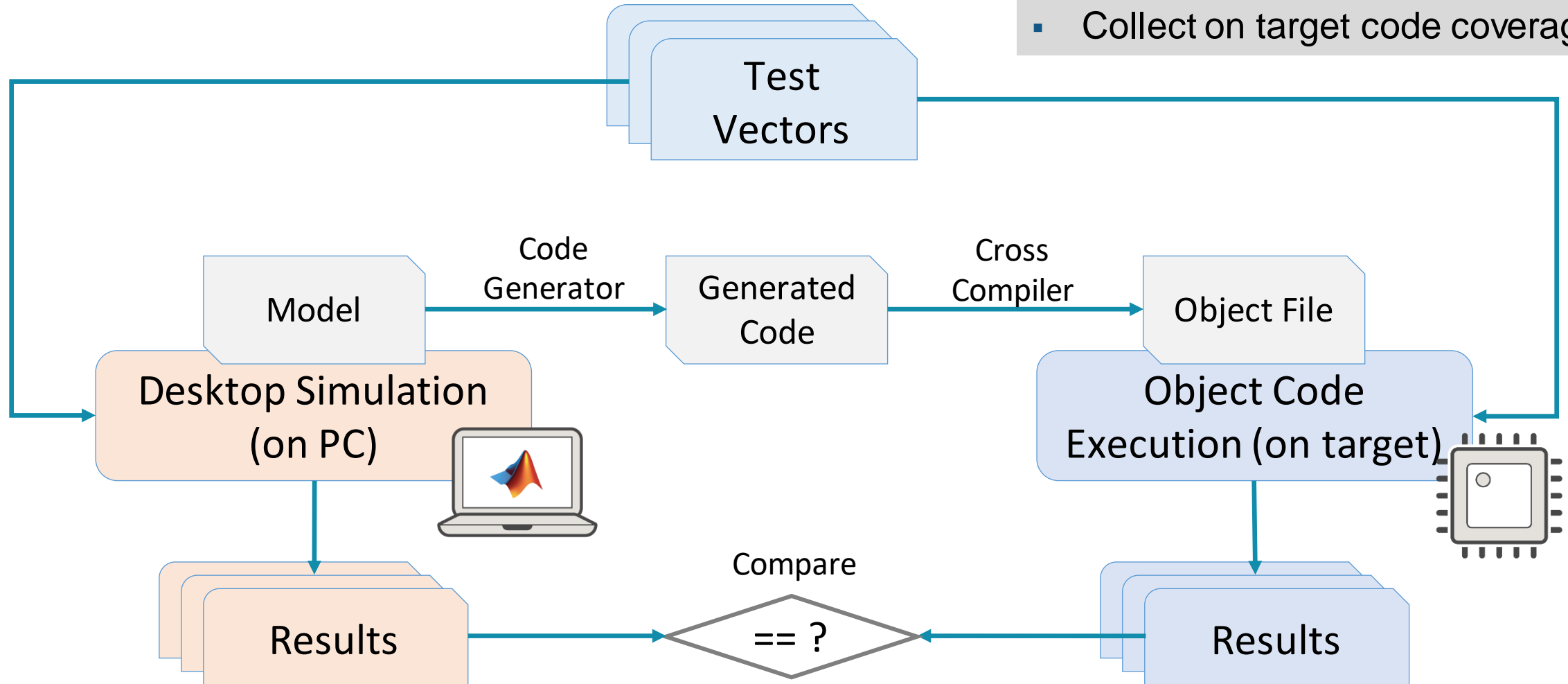


# Software-In-the-Loop Test with Model Reference



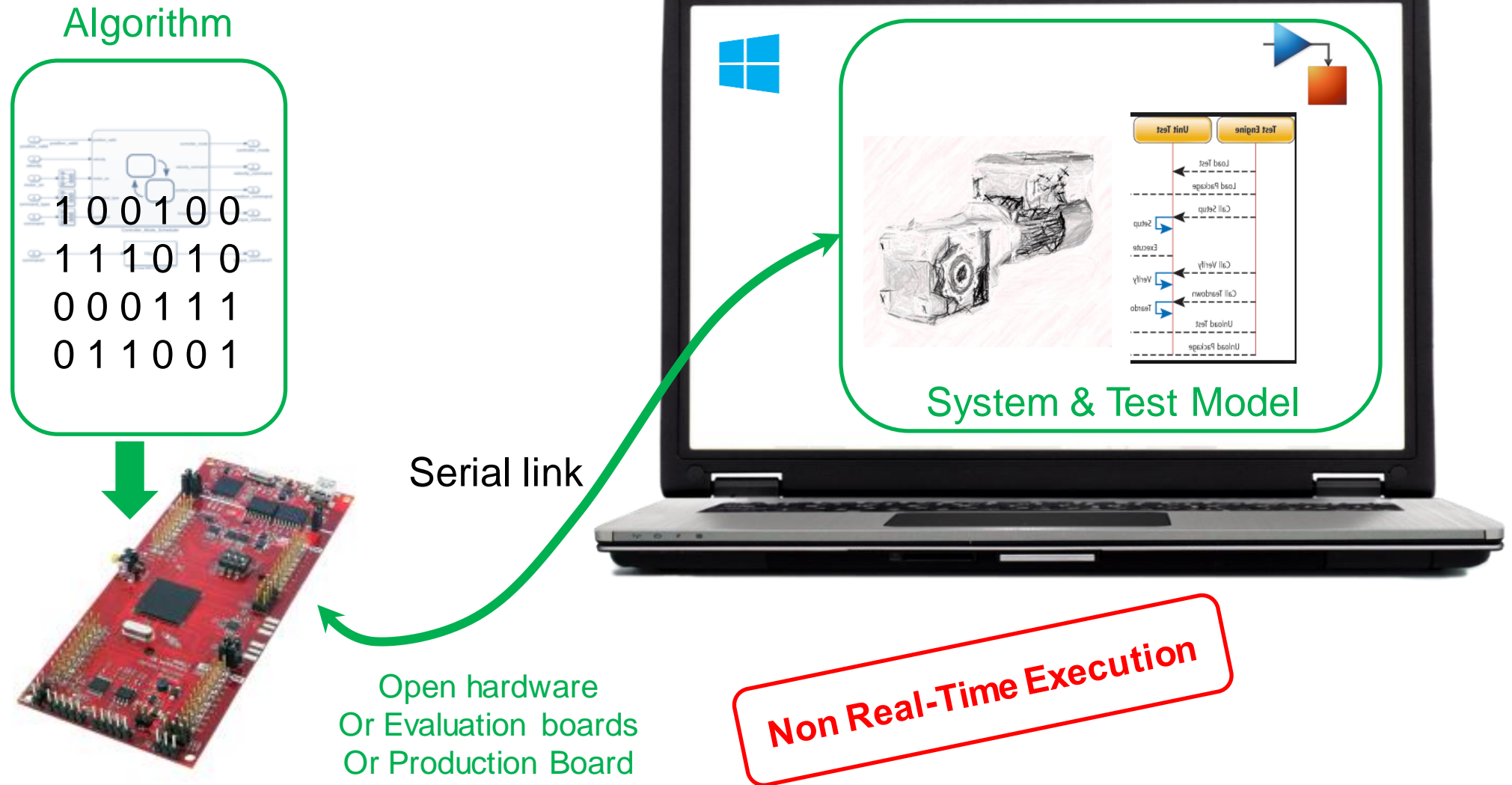
# Processor-In-the-Loop (PIL) Testing

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



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

## Verify Production Controller with Processor-in-the-loop





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

## Code Execution Profiling Report for mcb\_pmsm\_foc\_sim\_v2/Current Controll

The code execution profiling report is recorded by instrumentation probe. For more information, see [Profiling](#).

### 1. Summary

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

### 2. Profiled Sections of Code

Section
[+] <a href="#">Current_initialize</a>
<a href="#">Current_step [5e-05 0]</a>
<a href="#">Current_terminate</a>

### 3. CPU Utilization

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

Section	Maximum Execution Time in ns	Average Execution Time in ns	Max
[+] <a href="#">Current_initialize</a>	2260	2260	
<a href="#">Current_step [5e-05 0]</a>	5135	5067	
<a href="#">Current_terminate</a>	540	540	

### 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%

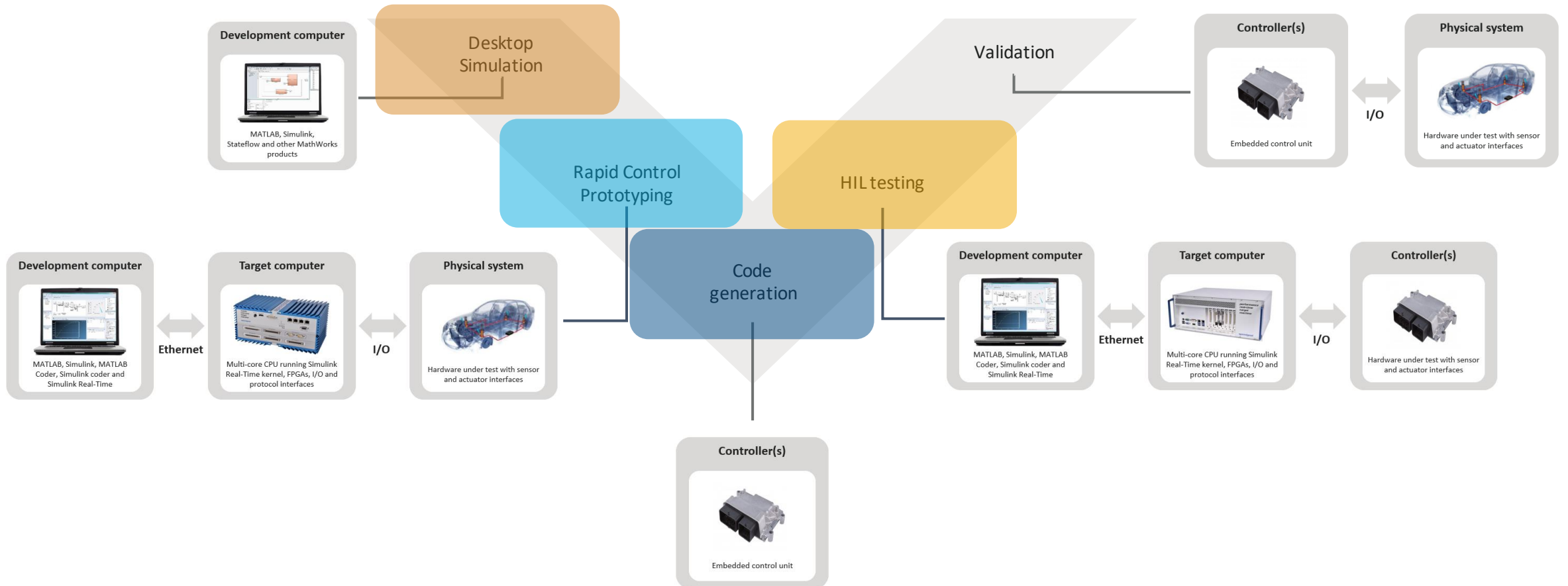
<a href="#">Current_step [5e-05 0]</a>	10.13%	10.27%
Overall CPU Utilization	10.13%	10.27%

# Deployment on the Target

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

The screenshot displays the MATLAB/Simulink environment. The main window shows a Simulink model titled "PMSM Control Host". The model includes a "Serial Setup" block, a "Reference Speed (RPM)" dial, a "Start / Stop Motor" switch, and a "Scope (FixedStepDiscrete)" block. A "Debug signals" panel is visible, with "Speed Control" selected. Below the model is a photograph of the physical hardware, which includes a red PCB with a microcontroller and a black PMSM motor. The "SelectSignals" window on the right shows a grid for signal selection, with the x-axis ranging from 0 to 0.1 and the y-axis from -0.03 to 0.08. The status bar at the bottom indicates "Running" and "FixedStepDiscrete".

# Code Generation and Real-Time Testing in Model-Based Design



## Key Takeaways

- Model-based design for motor control enables you to make 50% faster time to market.
  - Various fidelity modeling of motor and inverter using Simscape Electrical
  - Autotuning PI controller gains using optimization algorithm
- Motor Control Toolbox, a new product in R2020a, enables you to minimize development time using reference examples
  - Sensor calibration, built-in algorithmic blocks, automated parameter estimation, and gain-tuning
- Generate, deploy and validate production code

# Q&A