

The background features a dark blue field on the left and a grey field on the right, separated by a diagonal line. In the upper right, there are white waveforms. In the lower right, there is a colorful 3D wireframe mesh transitioning from yellow to green to blue, and a faint blue circuit board pattern.

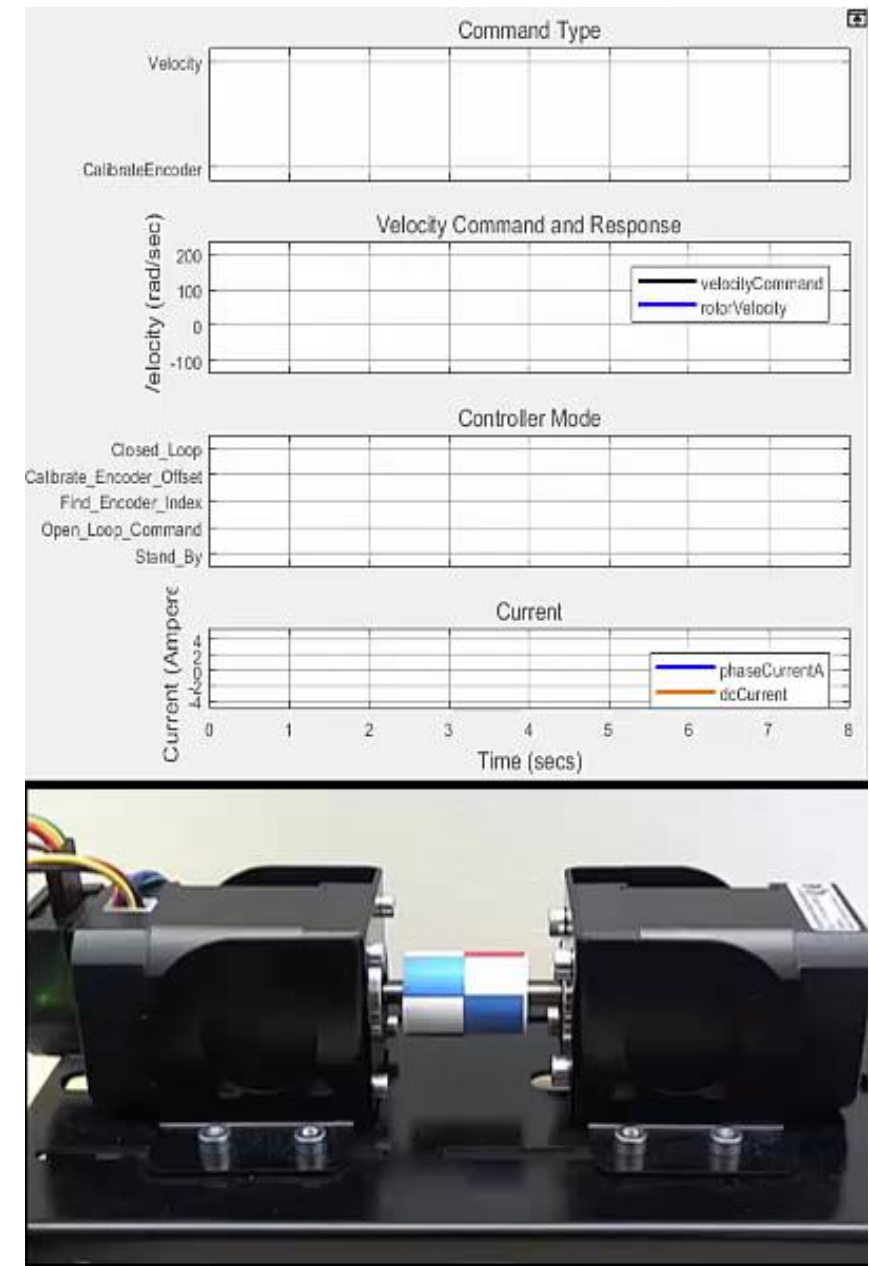
MATLAB EXPO 2017

Hardware and Software Co-Design for Motor Control Applications

GianCarlo Pacitti
Senior Application Engineer, MathWorks

Agenda

- Why use Hardware and Software for motor control?
- Why use Model-Based Design for motor control?
- How to use Model-Based Design for motor control?



ZedBoard

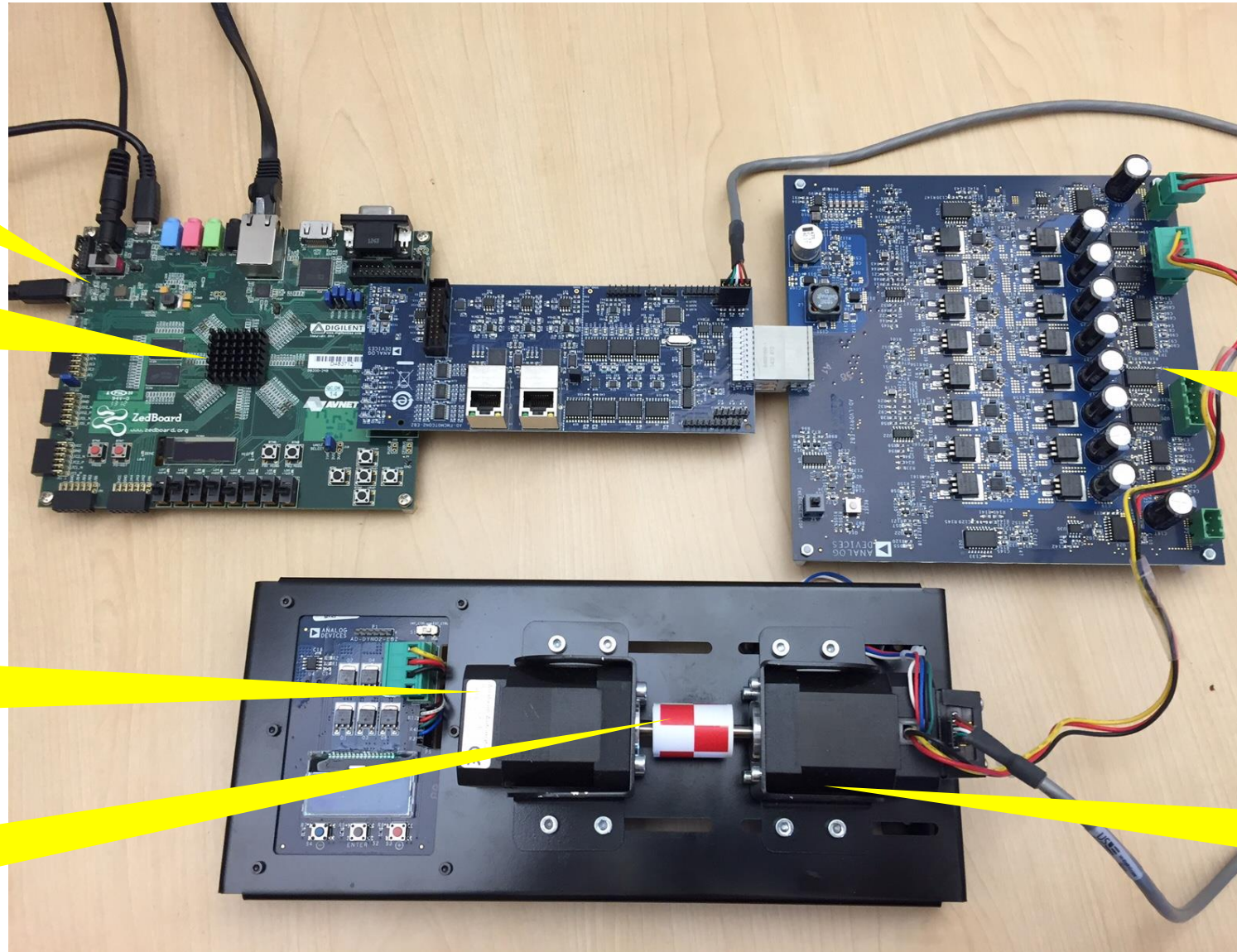
**Zynq SoC
(XC7Z020)**

Load motor

**Mechanical
coupler**

**FMC module:
control board +
low-voltage board**

**Motor under test
(with encoder)**



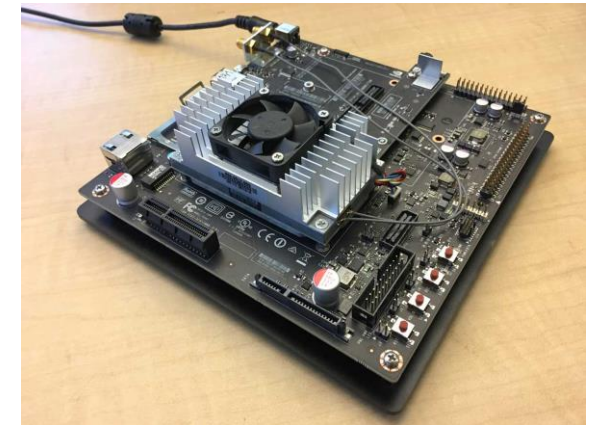
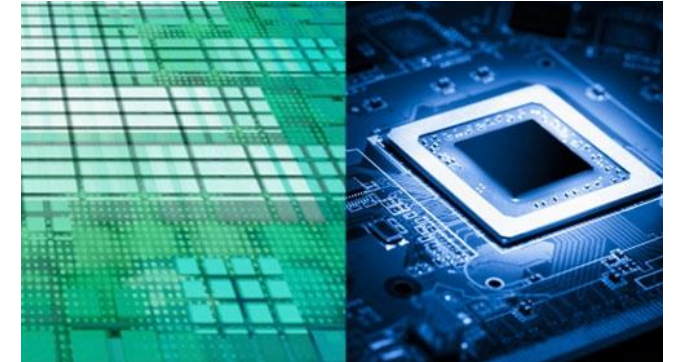
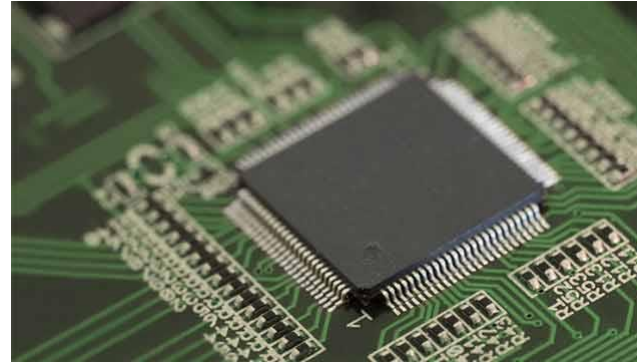
Key trend: Increasing demands from motor drives

- Increased performance targets require advanced algorithms
- Advanced algorithms require faster computing performance.
 - Field-Oriented Control
 - Sensorless motor control
 - Vibration detection and suppression
 - Multi-axis control



Where are algorithms being run to gain performance?

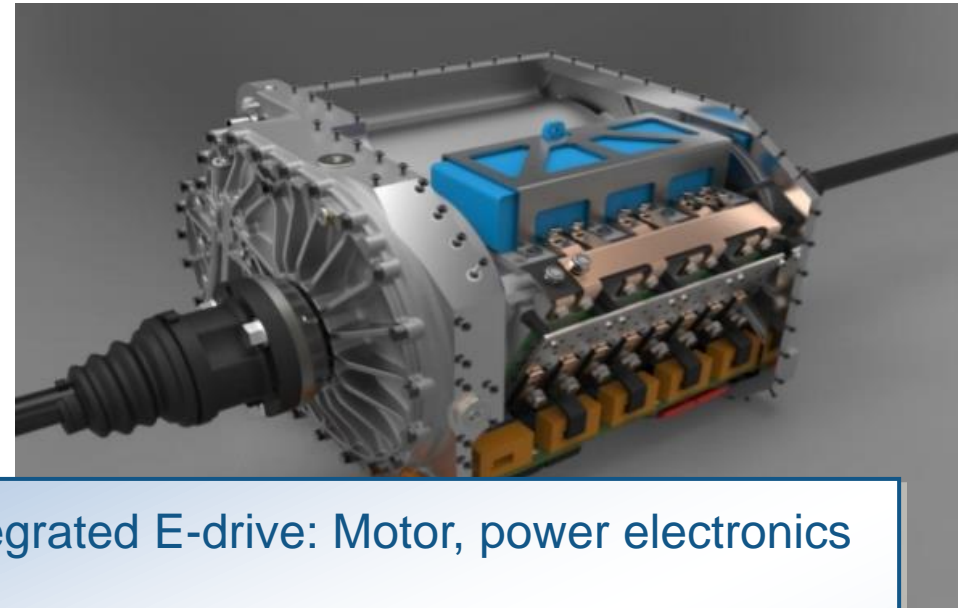
- Multi-core microprocessors
- Multi-processor systems
- FPGAs
- ASICs
- System-on-Chip devices (SoCs)
- GPUs*



- **Hardware and Software algorithms must be designed together**

Punch Powertrain develops complex SoC-based motor control

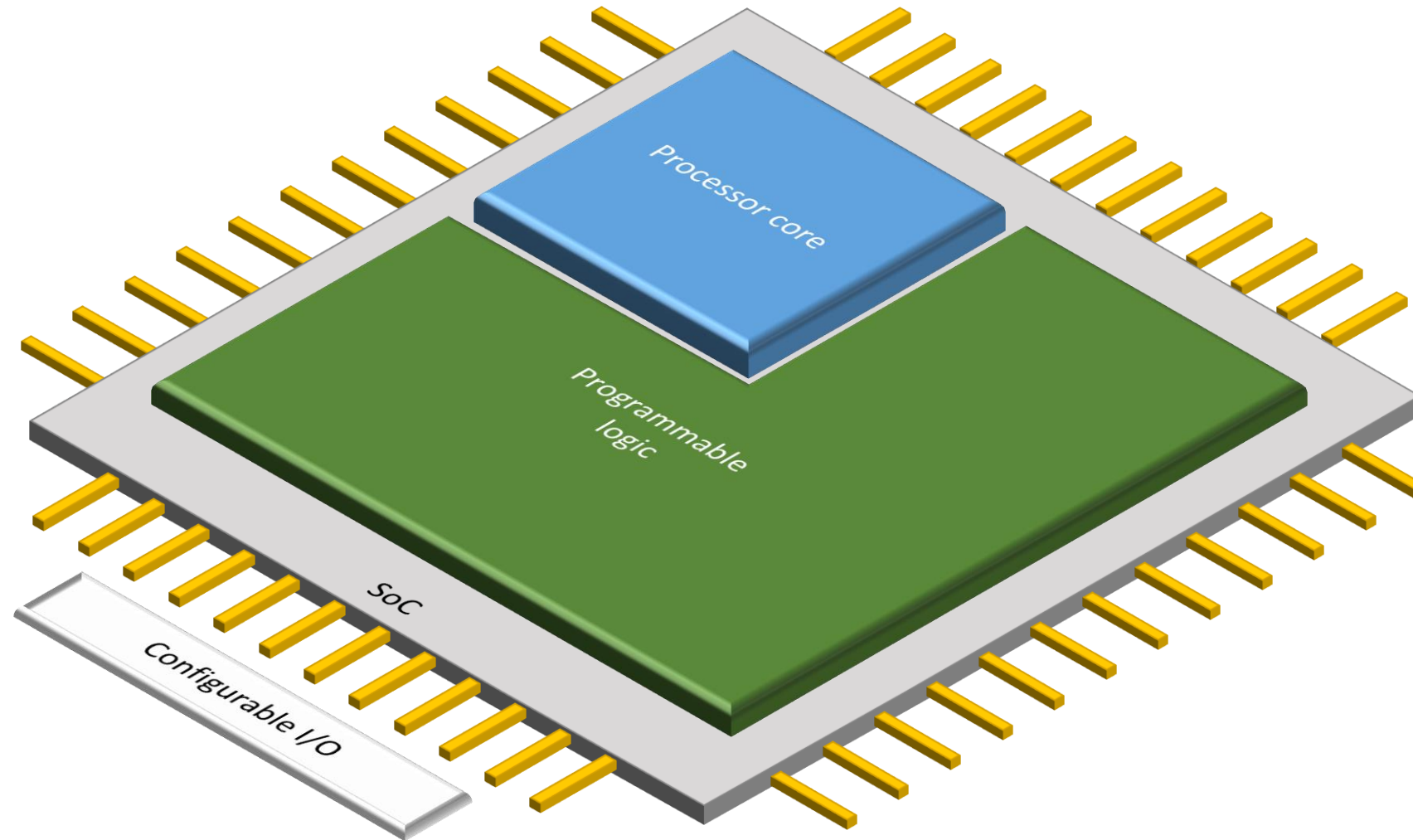
- Powertrains for hybrid and electric vehicles
- Need to increase power density and efficiency at a reduced cost
 - Integrate motor and power electronics in the transmission
- New switched reluctance motor
 - Fast: 2x the speed of their previous motor
 - Target to a Xilinx® Zynq® SoC 7045 device
 - Complex: 4 different control strategies
- Needed to get to market quickly
- No experience designing FPGAs!



- ✓ Designed integrated E-drive: Motor, power electronics and software
- ✓ 4 different control strategies implemented
- ✓ Done in 1.5 years with 2FTE's
- ✓ Models reusable for production
- ✓ Smooth integration and validation due to development process – thorough validation before electronics are produced and put in the testbench

[Link to video](#)

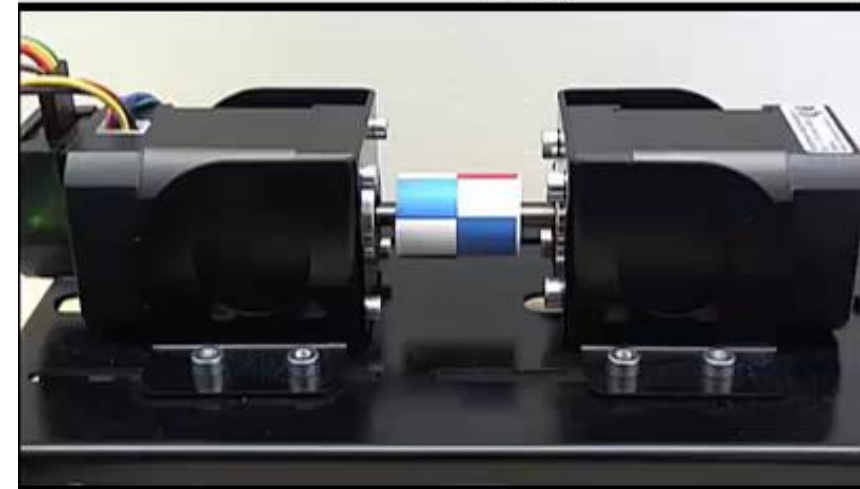
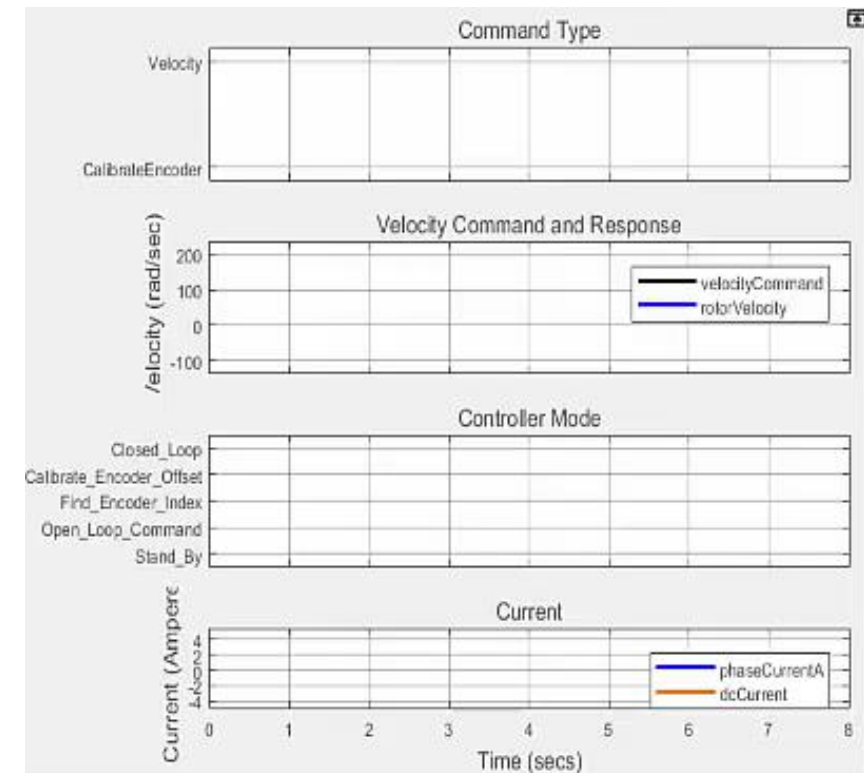
What's Inside an FPGA SoC?



Why use Hardware and Software for Motor Control?

- In order to meet increased performance
- You need more complex algorithms
- Running on the right hardware

- Why use Hardware and Software for motor control?
- Why use Model-Based Design for motor control?
- How to use Model-Based Design for motor control?



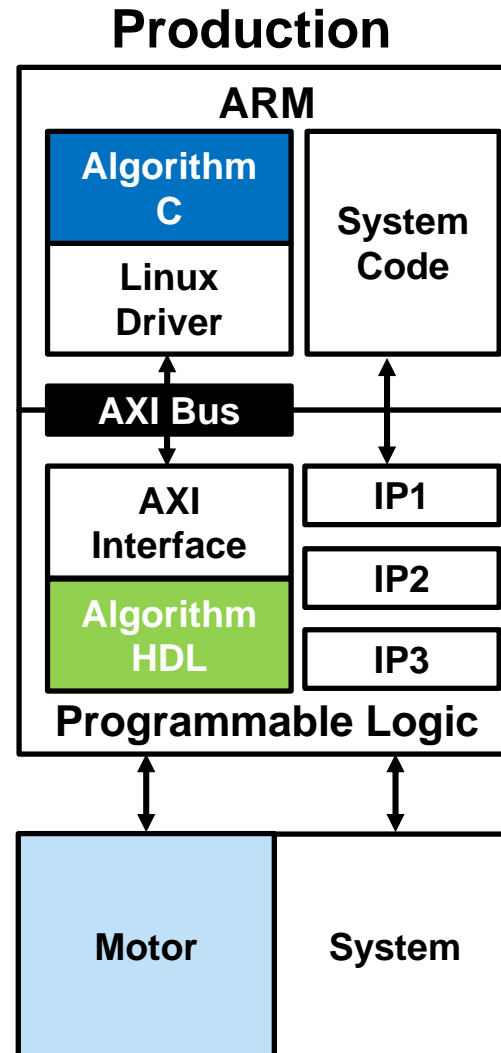
Challenges in Developing Advanced Motor Control Algorithms

- Integration requires collaboration
- Validation of design specifications with limits on access to test hardware
- How to make design decisions?

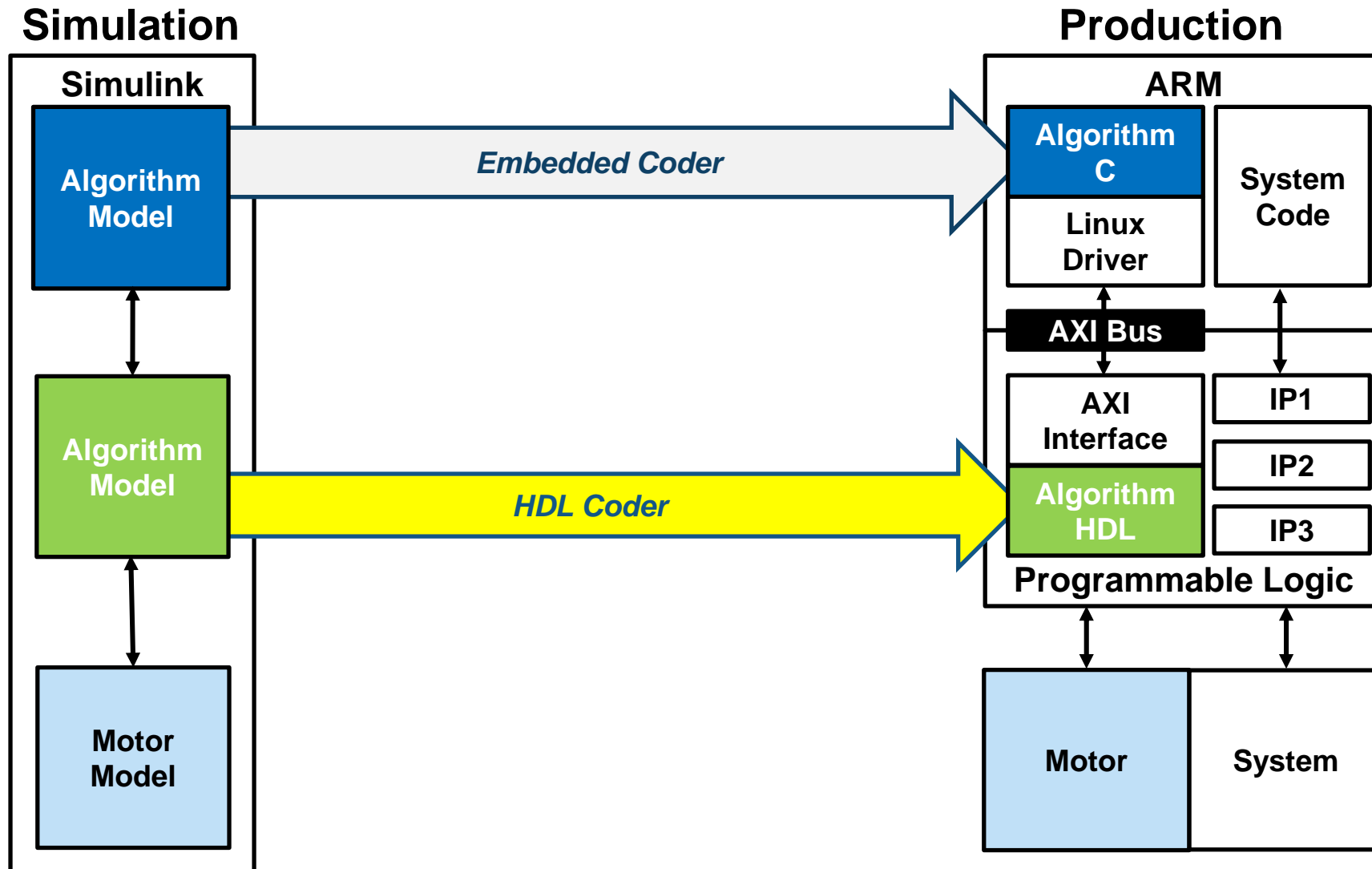
Why use Model-Based Design to Develop Motor Control Applications?

- Enables early validation of specifications using simulation months before hardware is available.
- Dramatically improves design team collaboration and designer productivity by using a single design environment.
- Reduces hardware testing time by 5x by shifting design from lab to the desktop

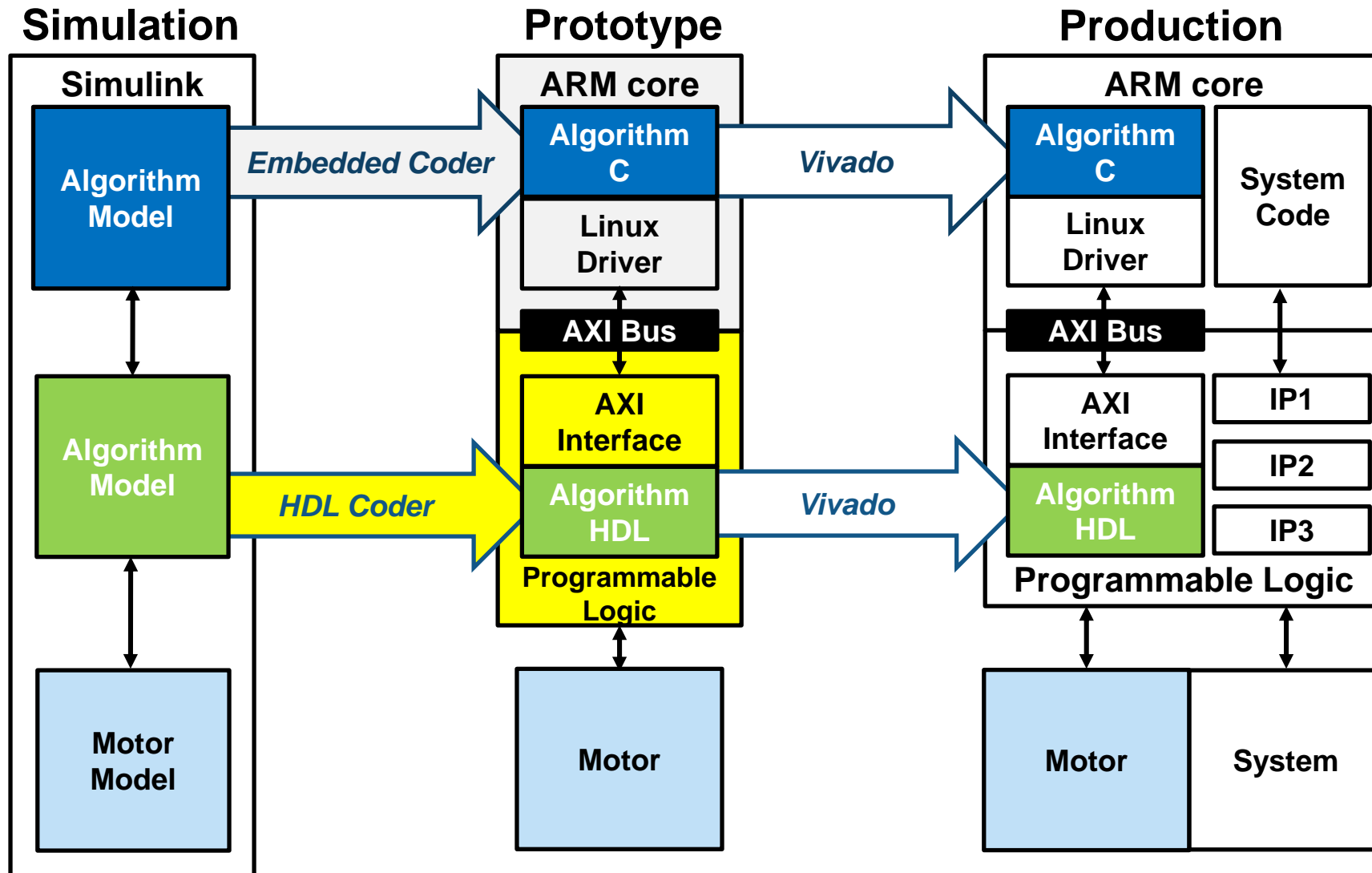
Components of Motor Control Production Applications



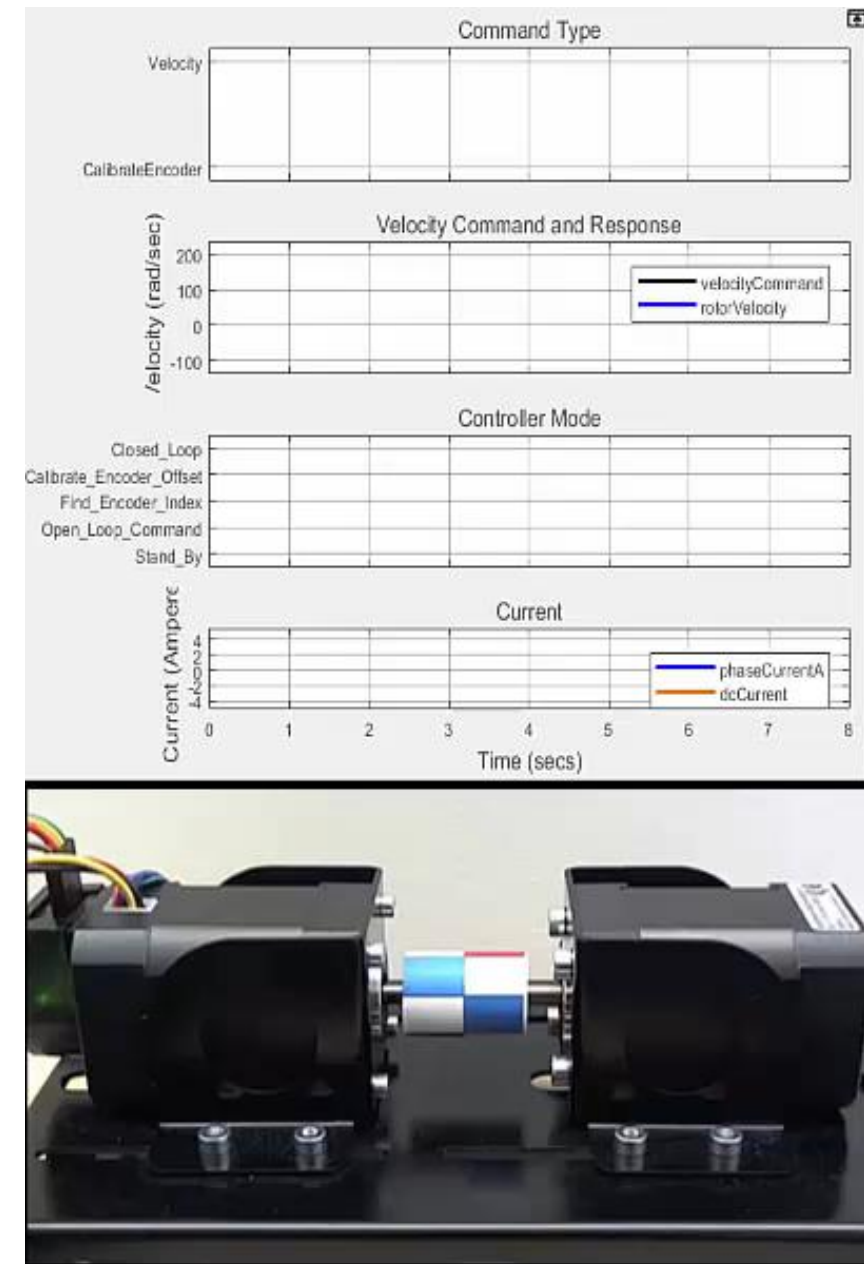
From Simulation to Production



From Simulation to Prototype to Production



- Why use Hardware and Software for motor control?
- Why use Model-Based Design for motor control?
- How to use Model-Based Design for motor control?



ZedBoard

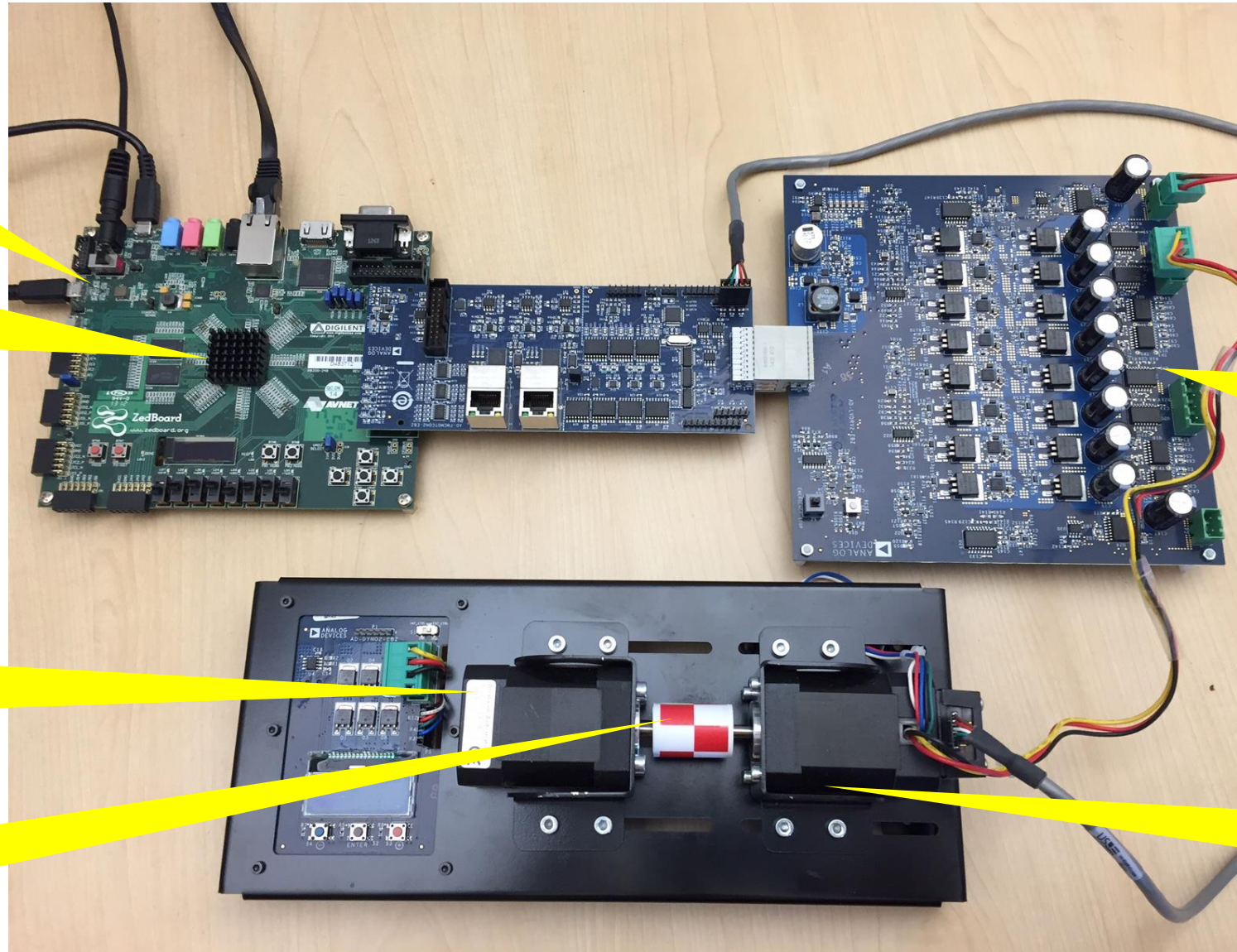
Zynq SoC
(XC7Z020)

Load motor

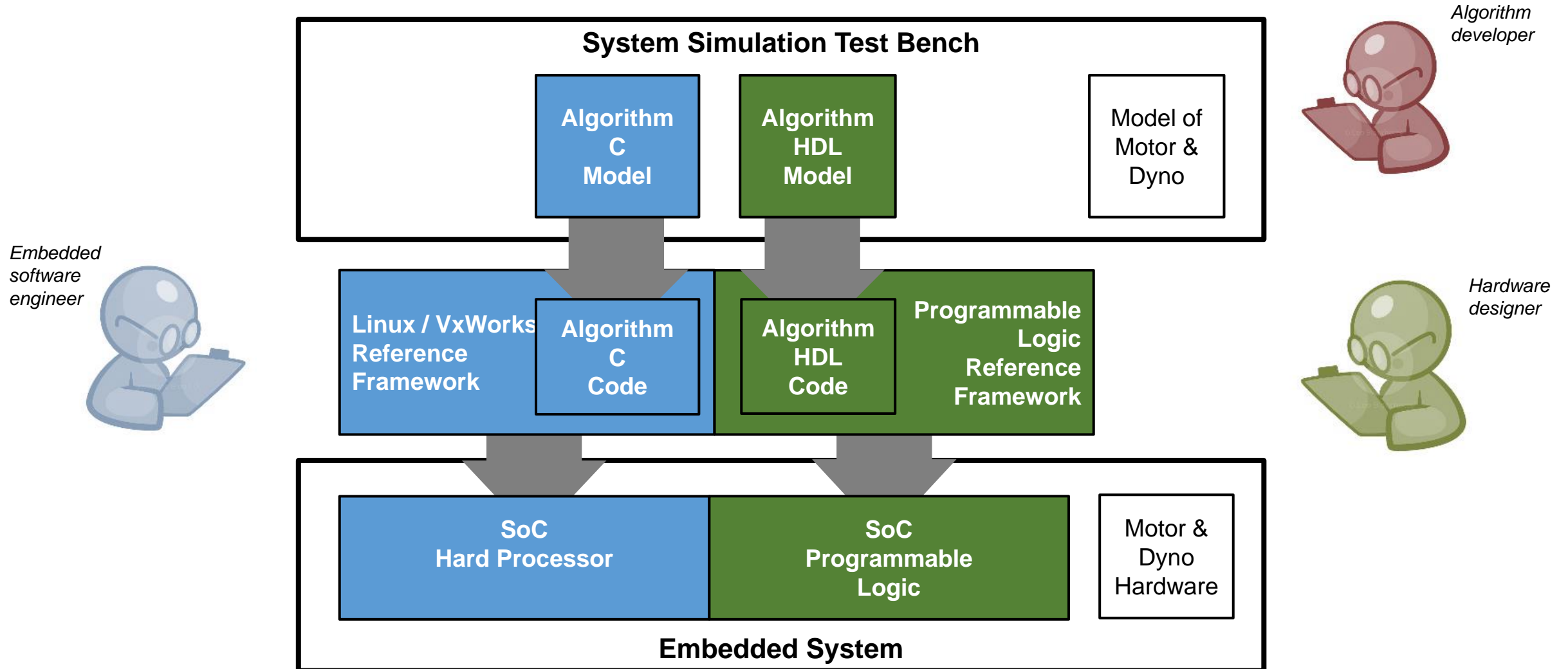
Mechanical
coupler

FMC module:
control board +
low-voltage board

Motor under test
(with encoder)



Conceptual workflow targeting hardware and software



focZynqTestBench - Simulink

File Edit View Display Diagram Simulation Analysis Code Tools Help

focZynqTestBench

Field-Oriented Control of Velocity Hardware/Software Test Bench

Copyright 2015-2017 The MathWorks, Inc.

System_Inputs

C/D

Controller_Algorithm

Motor_And_Load

D/C

Verify_Outputs

Verify_Outputs

Ready

View 1 warning 68%

VariableStepAuto

System_Response

File Tools View Simulation Help

Command Type

Velocity Command and Response

Controller Mode

Current

Velocity (rad/sec)

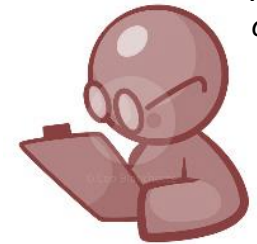
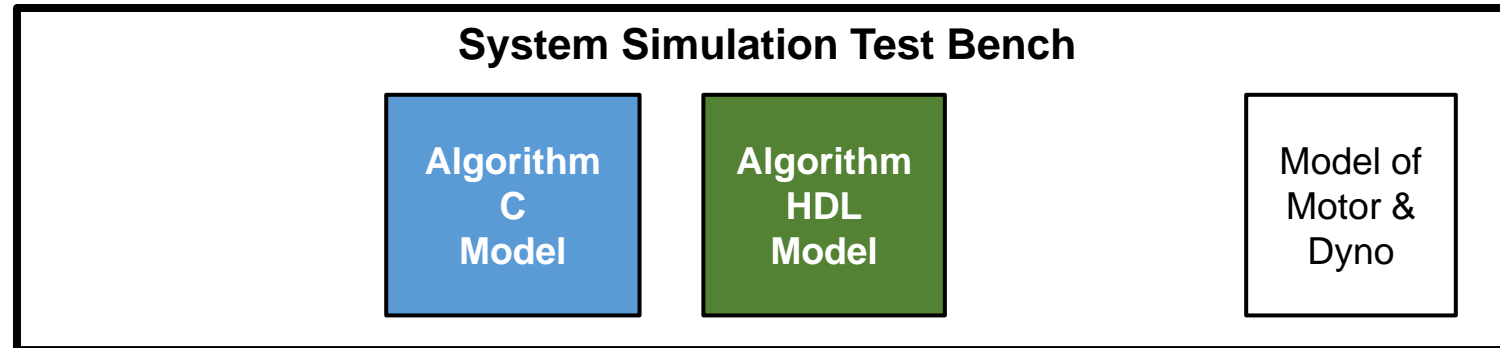
Current (Ampere)

Time (secs)

Ready

Sample based

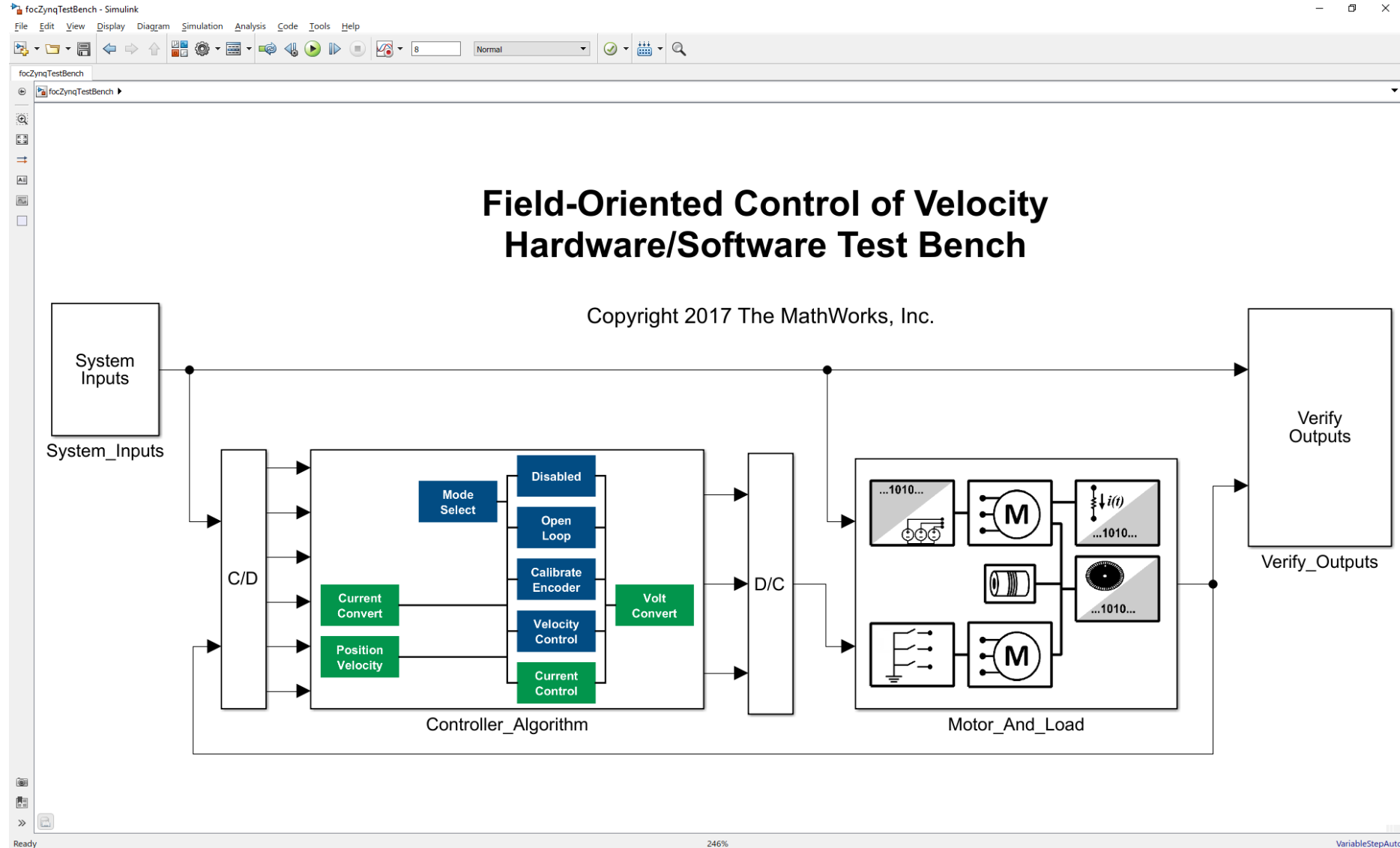
Building a System Simulation Test Bench



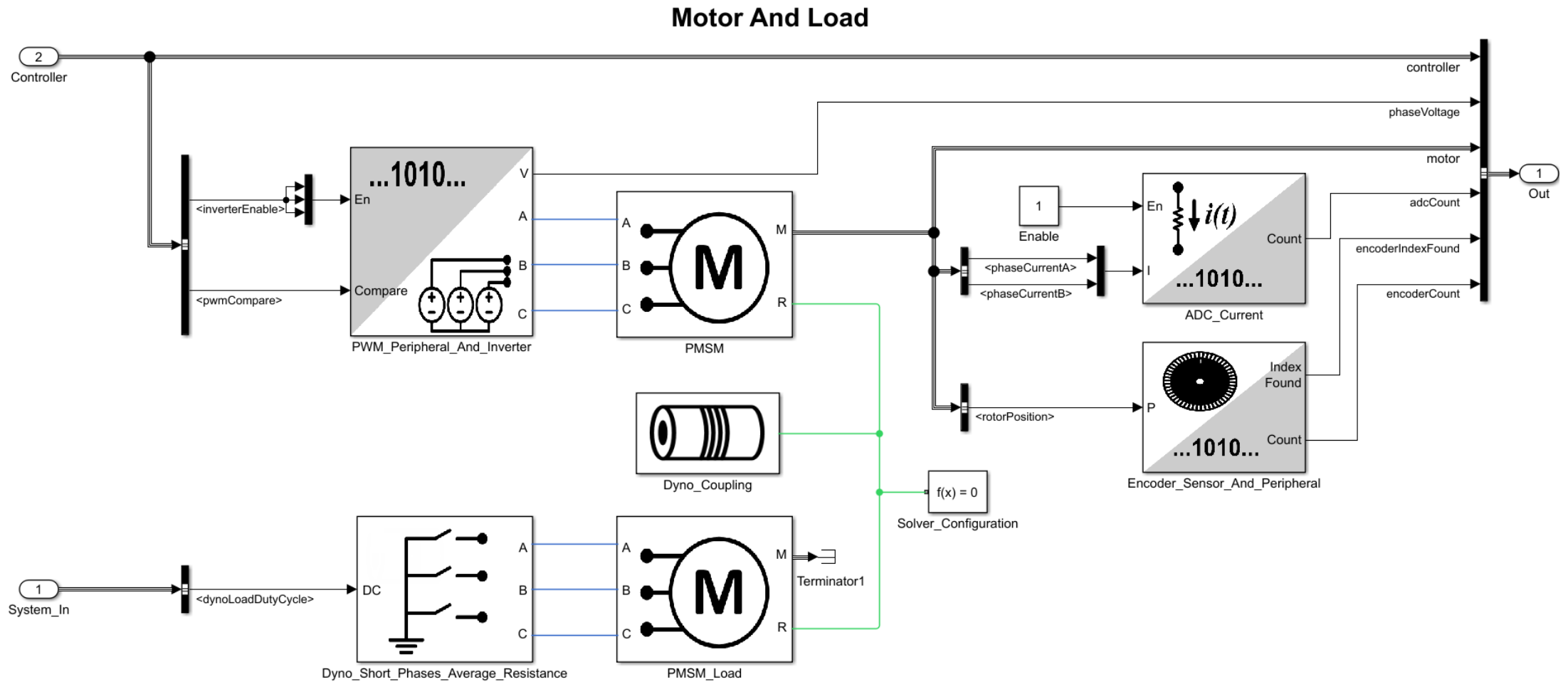
*Algorithm
developer*

- How do I get a good model of the motor?
- How can I make sure it matches real-world behaviour?

What's Inside a Motor Model?

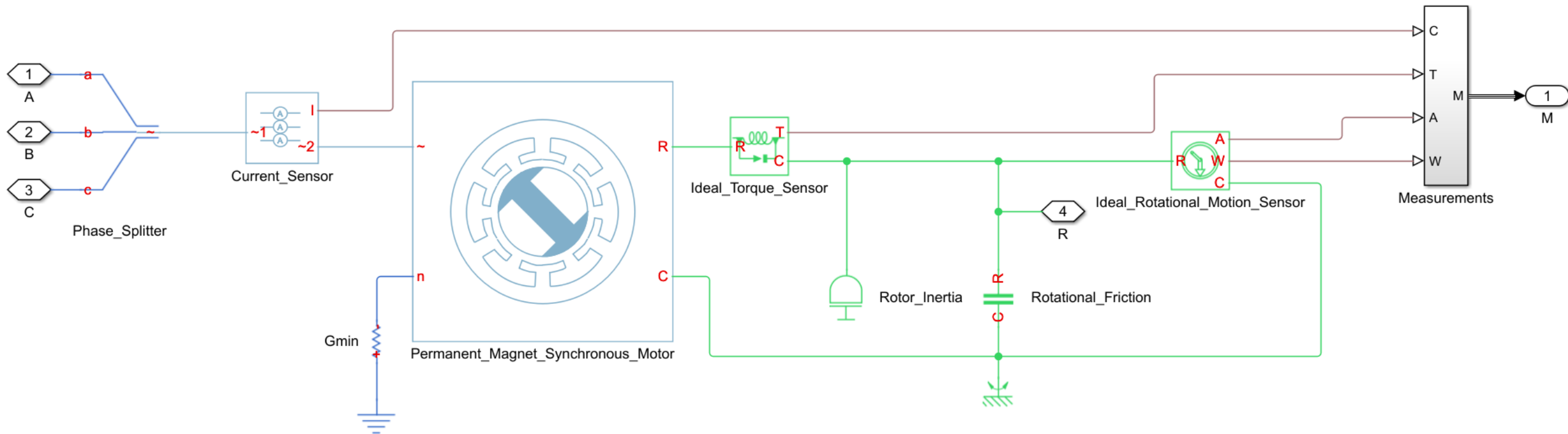


What's Inside a Motor Model?

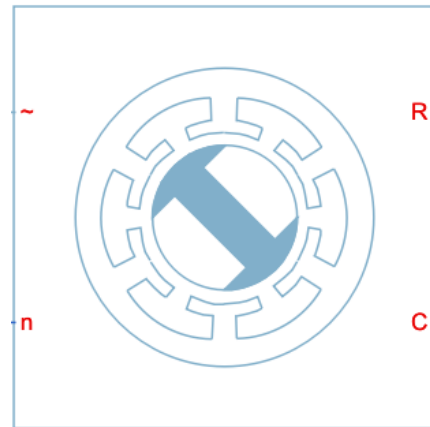


What's Inside a Motor Model?

PMSM



What's Inside a Motor Model?



Permanent_Magnet_Synchronous_Motor

What's Inside a Motor Model?

Block Parameters: Rotational_Friction

Rotational Friction

The block represents friction in the contact between rotating bodies. The friction force is simulated as a function of relative velocity and assumed to be the sum of Stribeck, Coulomb, and viscous components. The sum of the Coulomb and Stribeck frictions at zero velocity is often referred to as the breakaway friction.

Connections R and C are mechanical rotational conserving ports. The block positive direction is from port R to port C. This means that if port R velocity is greater than that of port C, the block transmits torque from port R to port C.

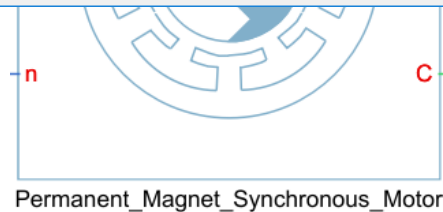
[Source code](#)

Settings

Parameters Variables

Breakaway friction torque:	<input type="text" value="pmsm.BreakawayFrictionTorque"/>	<input type="text" value="N*m"/>	<input type="text" value="Compile-time"/>
Breakaway friction velocity:	<input type="text" value="0.060623"/>	<input type="text" value="rad/s"/>	<input type="text" value="Compile-time"/>
Coulomb friction torque:	<input type="text" value="pmsm.CoulombFrictionTorque"/>	<input type="text" value="N*m"/>	<input type="text" value="Compile-time"/>
Viscous friction coefficient:	<input type="text" value="pmsm.ViscousFrictionCoefficient"/>	<input type="text" value="N*m/(rad/s)"/>	<input type="text" value="Compile-time"/>

OK Cancel Help Apply



Block Parameters: Permanent_Magnet_Synchronous_Motor

Permanent Magnet Synchronous Motor

This block represents a permanent magnet synchronous motor with sinusoidal flux distribution.

Right-click on the block and select Simscape block choices to access variant implementations of this block.

Settings

Main Initial Conditions

Number of pole pairs:	<input type="text" value="pmsm.PolePairs"/>	<input type="text" value=""/>	<input type="text" value="Compile-time"/>
Permanent magnet flux linkage:	<input type="text" value="pmsm.FluxLinkage"/>	<input type="text" value="Wb"/>	<input type="text" value="Compile-time"/>
Stator parameterization: Specify Ld, Lq, and L0			
Stator d-axis inductance, Ld:	<input type="text" value="pmsm.InductanceLd"/>	<input type="text" value="H"/>	<input type="text" value="Compile-time"/>
Stator q-axis inductance, Lq:	<input type="text" value="pmsm.InductanceLq"/>	<input type="text" value="H"/>	<input type="text" value="Compile-time"/>
Stator zero-sequence inductance, L0:	<input type="text" value="pmsm.InductanceL0"/>	<input type="text" value="H"/>	<input type="text" value="Compile-time"/>
Stator resistance per phase, Rs:	<input type="text" value="pmsm.StatorPhaseResistance"/>	<input type="text" value="Ohm"/>	<input type="text" value="Compile-time"/>

OK Cancel Help Apply

Block Parameters: Rotor_Inertia

Inertia

The block represents an ideal mechanical rotational inertia.

The block has one mechanical rotational conserving port. The block positive direction is from its port to the reference point. This means that the inertia torque is positive if the inertia is accelerated in the positive direction.

[Source code](#)

Settings

Parameters Variables

Inertia:	<input type="text" value="pmsm.Inertia"/>	<input type="text" value="kg*m^2"/>	<input type="text" value="Compile-time"/>
----------	---	-------------------------------------	---

OK Cancel Help Apply

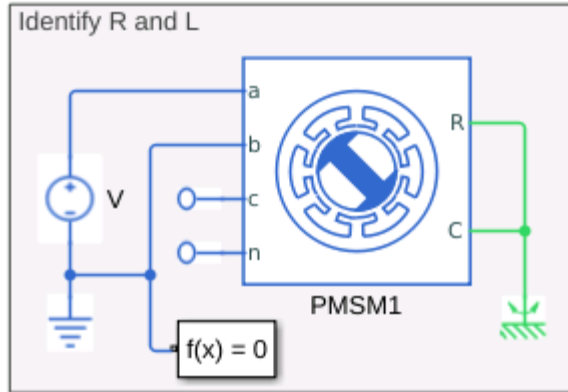
- How can we find the parameters we need for the model?

How to Find the Right Motor Parameters?

- Ask the motor designer
- From manufacturer's data sheets
- From direct bench-top measurements or test data

Modelling a PMSM with limited supplier data

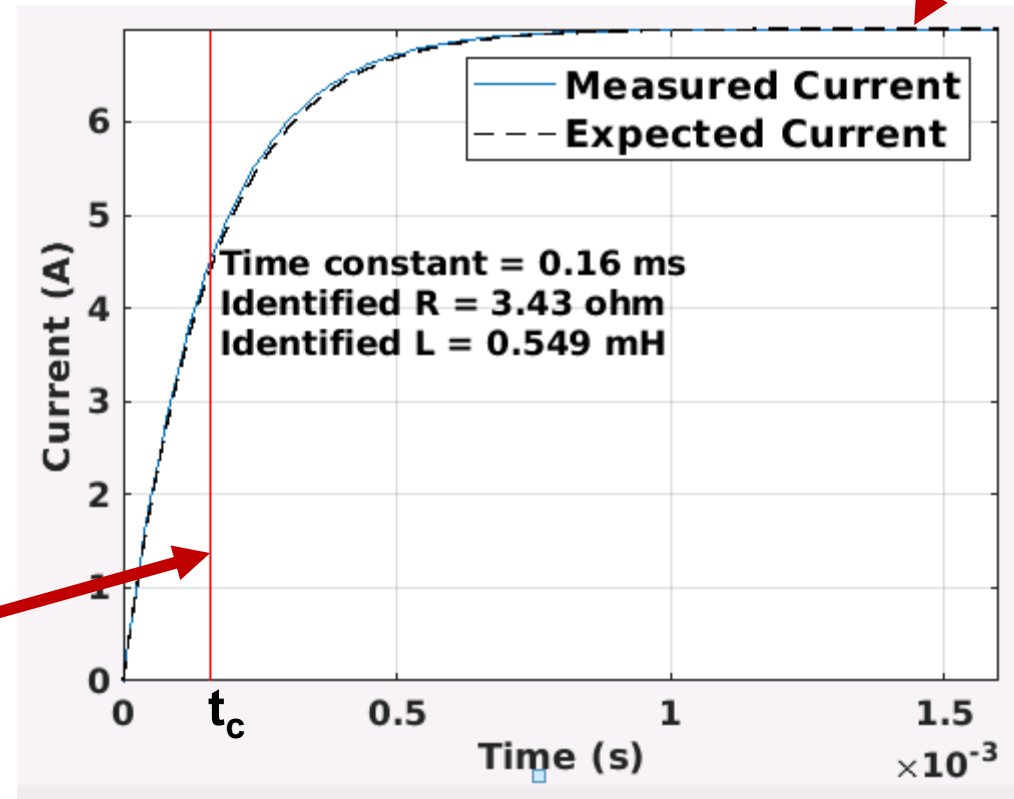
Tune to measurement data – Step 1



Locked rotor
Step voltage test

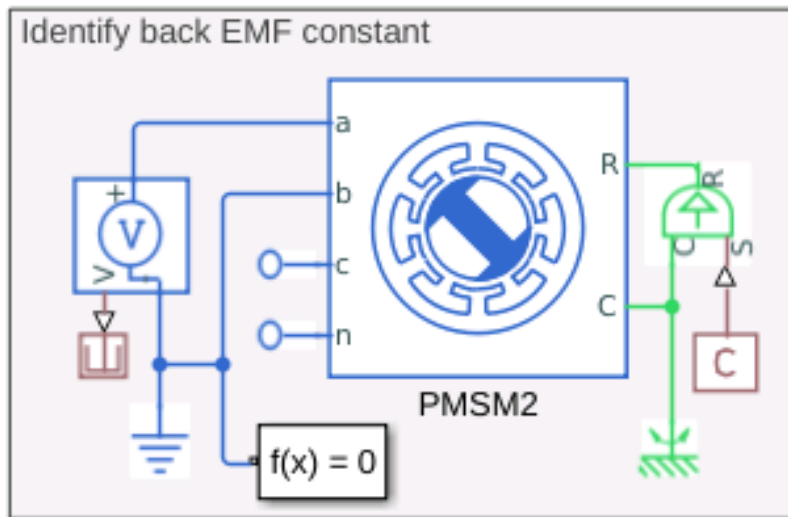
$$R = V/I$$

$$L = R * t_c$$

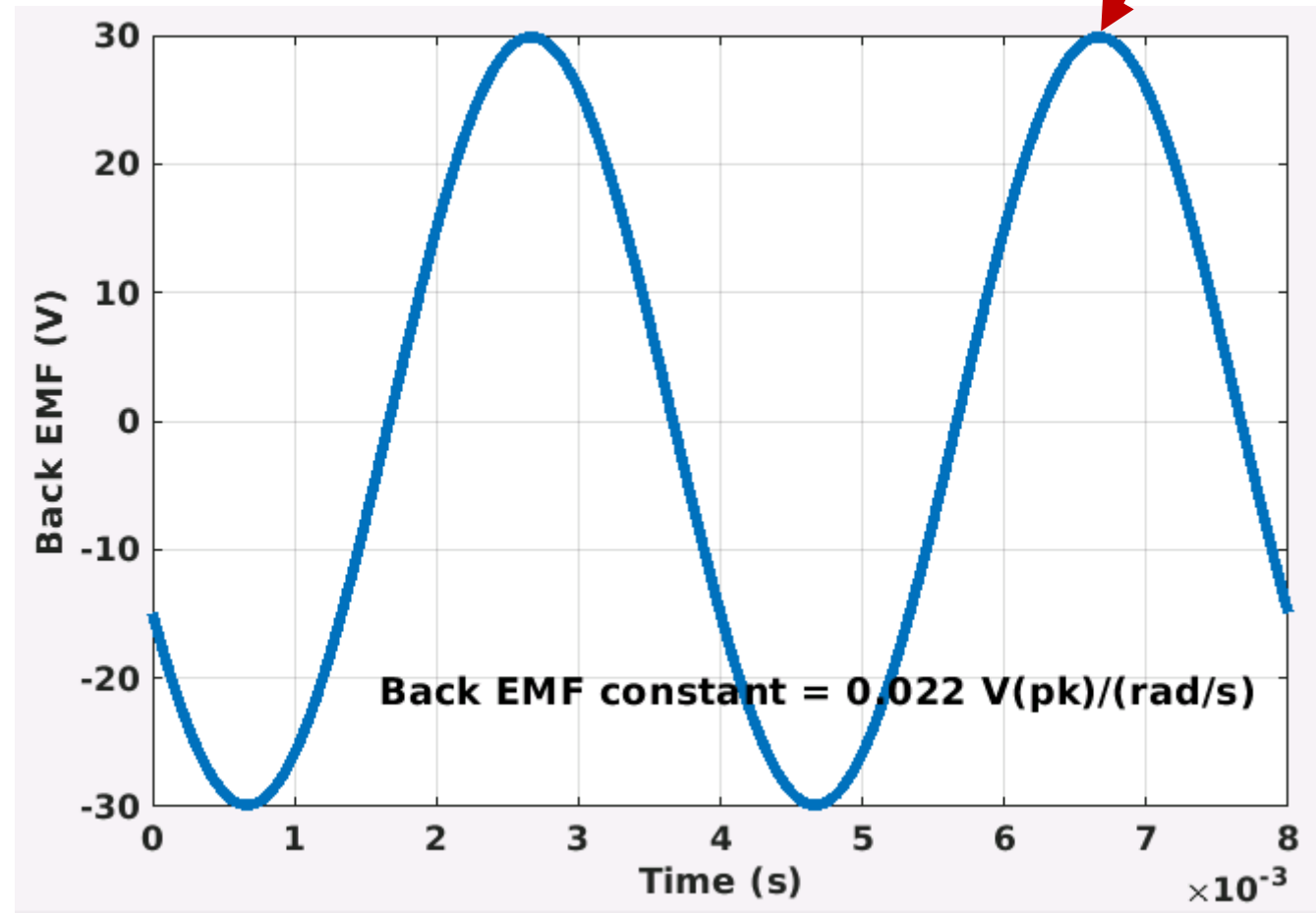


Modelling a PMSM with limited supplier data

Tune to measurement data – Step 2

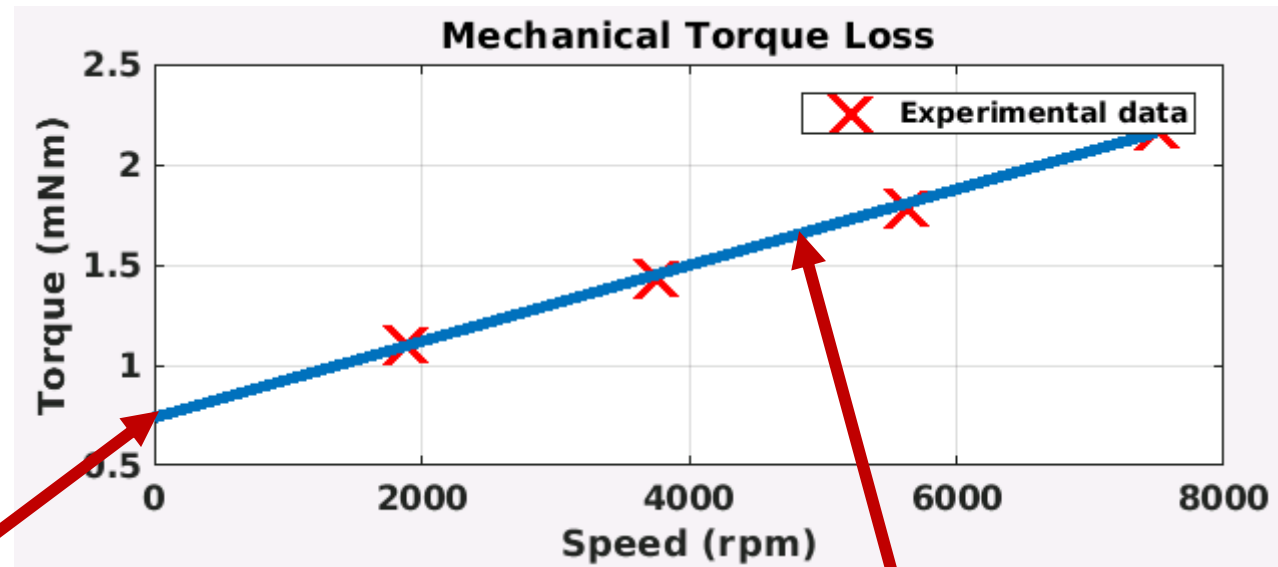
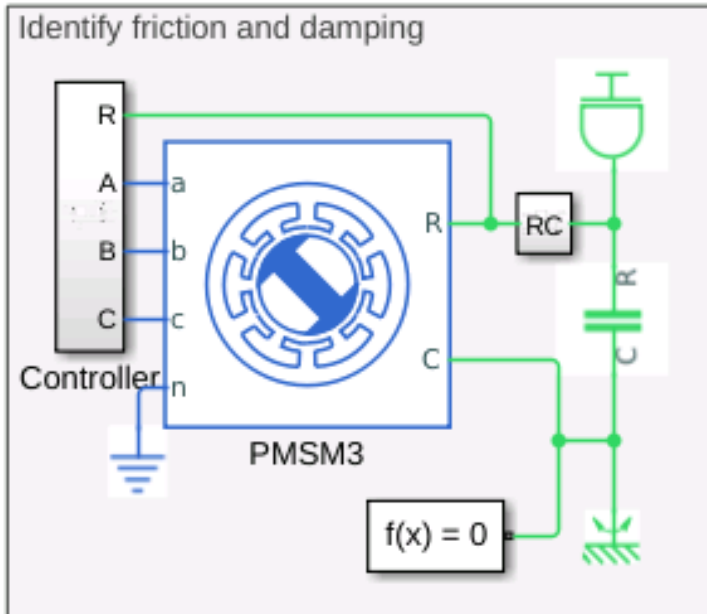


Back EMF test



Modelling a PMSM with limited supplier data

Tune to measurement data – Step 3

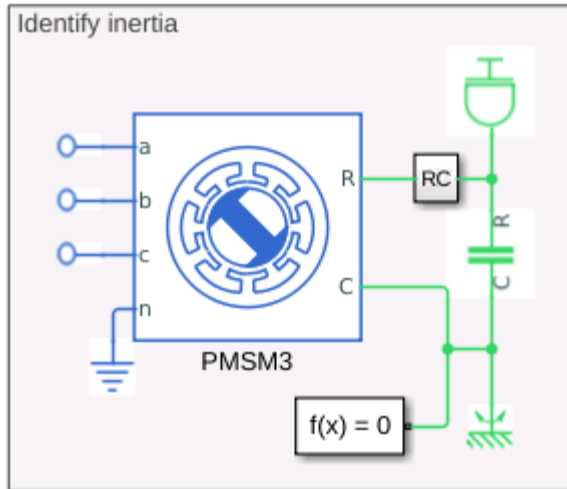


Friction

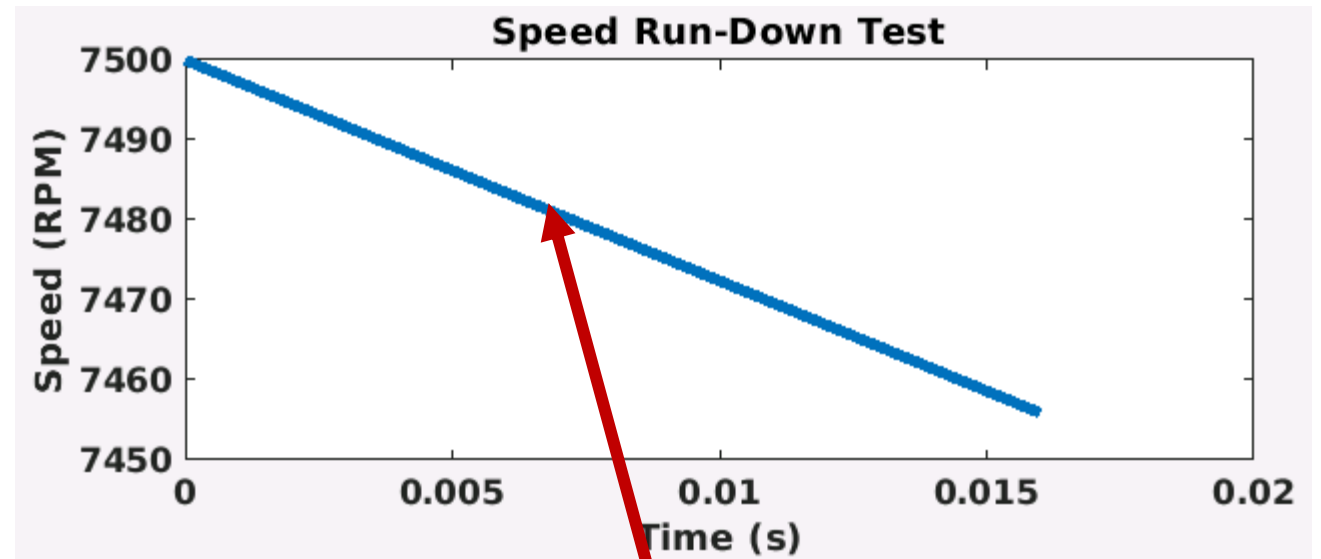
**Damping
coefficient
= gradient**

Modelling a PMSM with limited supplier data

Tune to measurement data – Step 4



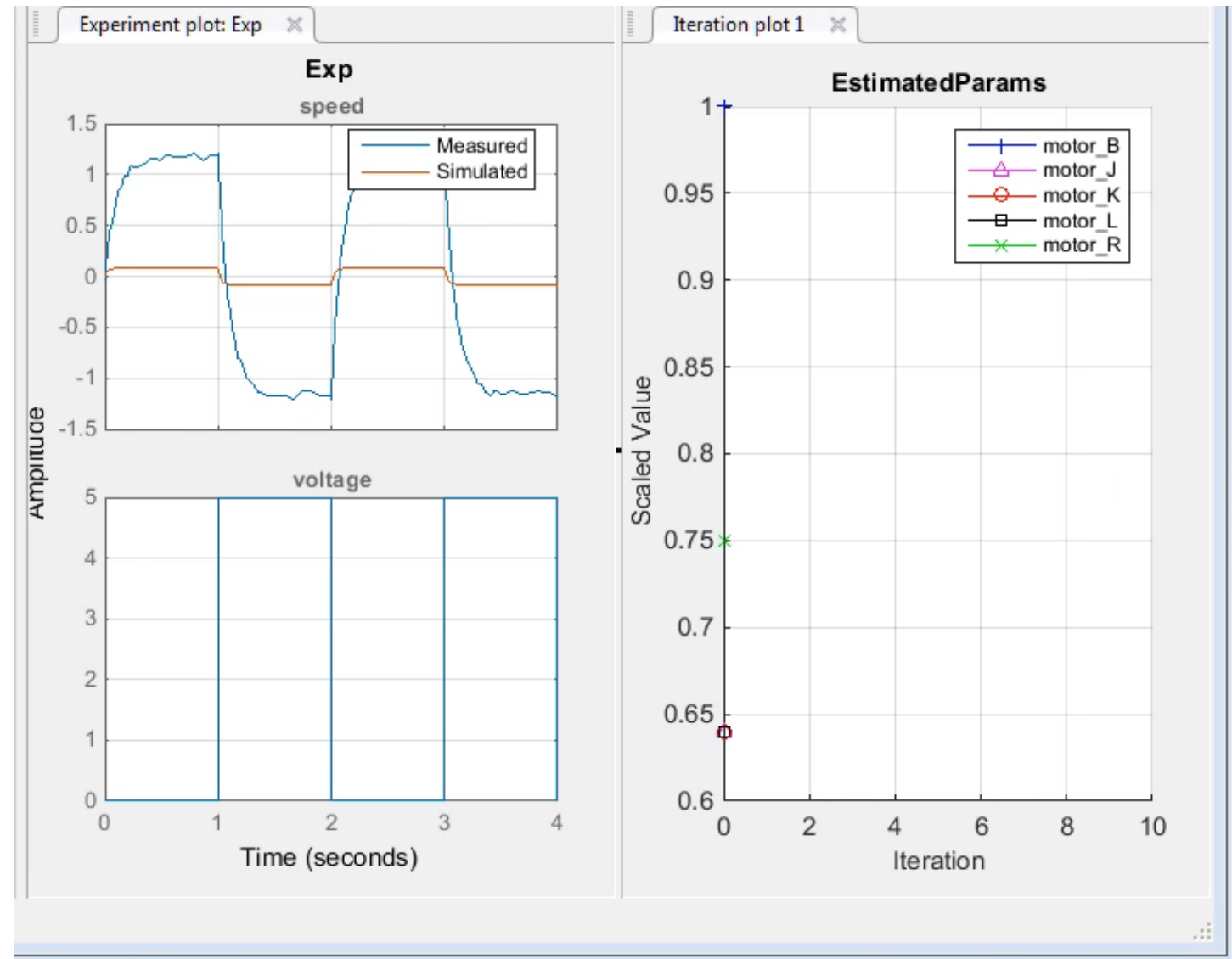
Speed run-down test



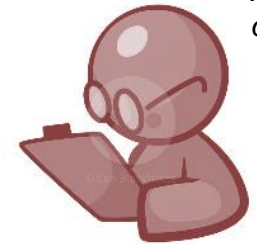
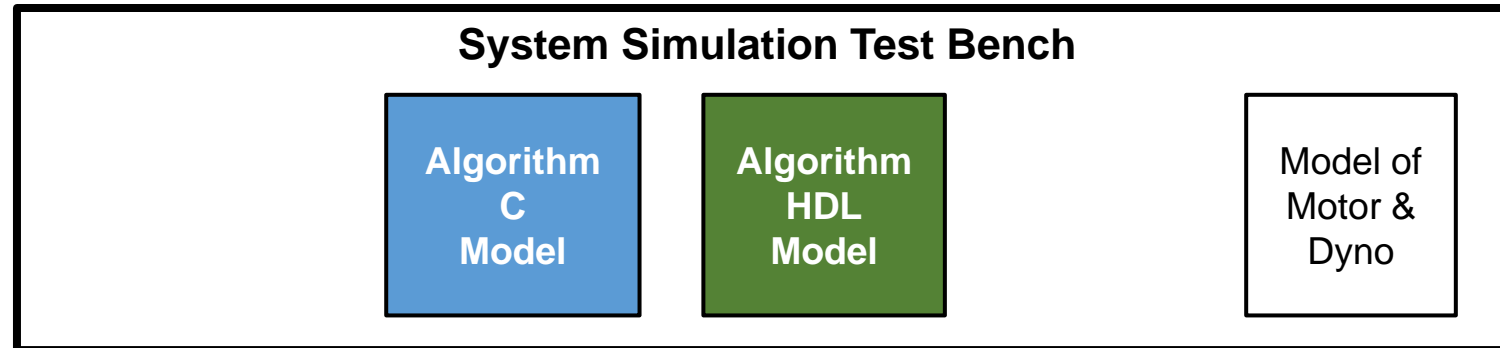
**Inertia =
Torque/gradient**

Estimating Parameters from measured data using Simulink Design Optimization

- Use simulation-based optimisation
 - match model parameters to real-world data



Adding Implementation Detail to Algorithms



*Algorithm
developer*

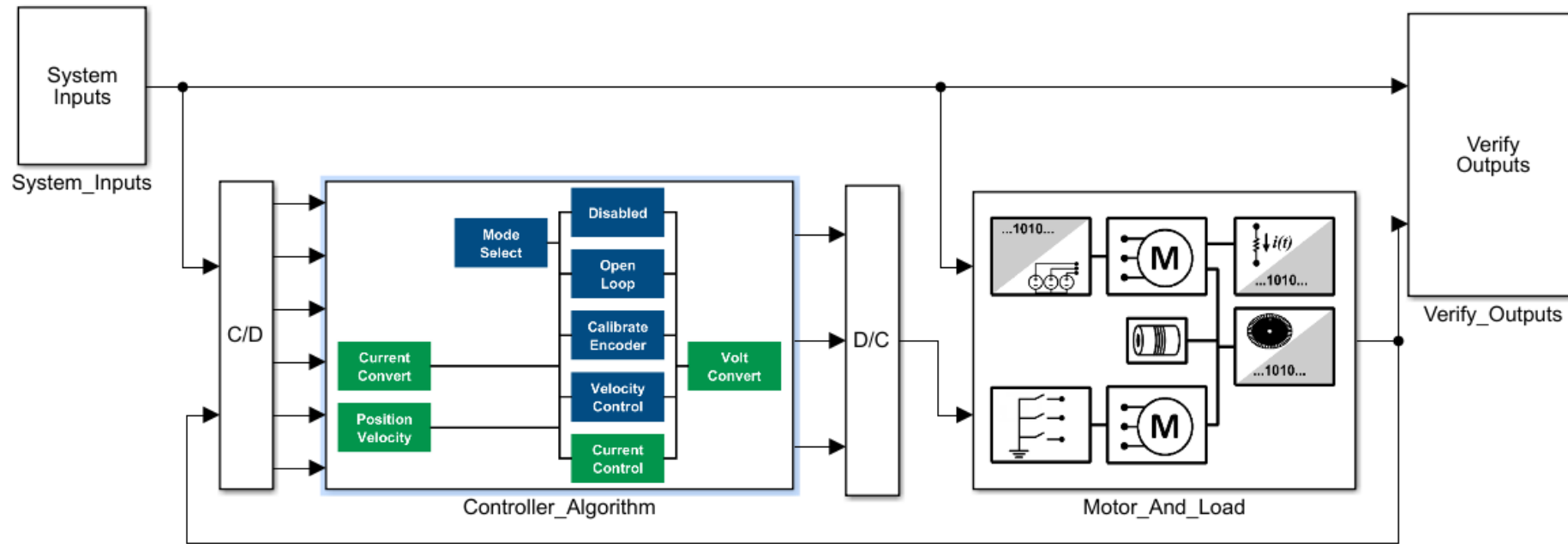
- Which parts of my algorithm should be implemented in C, and which in HDL?

Strategies for Partitioning an Algorithm Between Hardware and Software

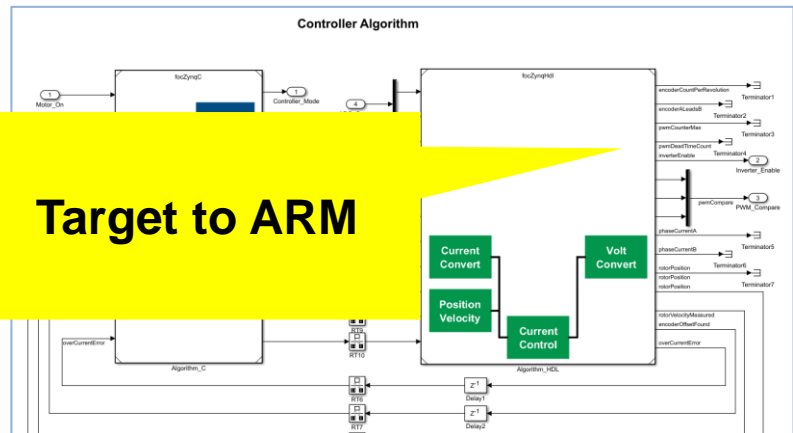
- Use experience
 - some timing requirements are known e.g. current control @25kHz
- Put everything on the software core and profile it
 - where are the bottlenecks?
 - Can these be moved to hardware?
- Put algorithms where the data comes in
 - minimise data transfer
- Monitor resource usage and move things when you are near the limit



Hardware/Software Partitioning

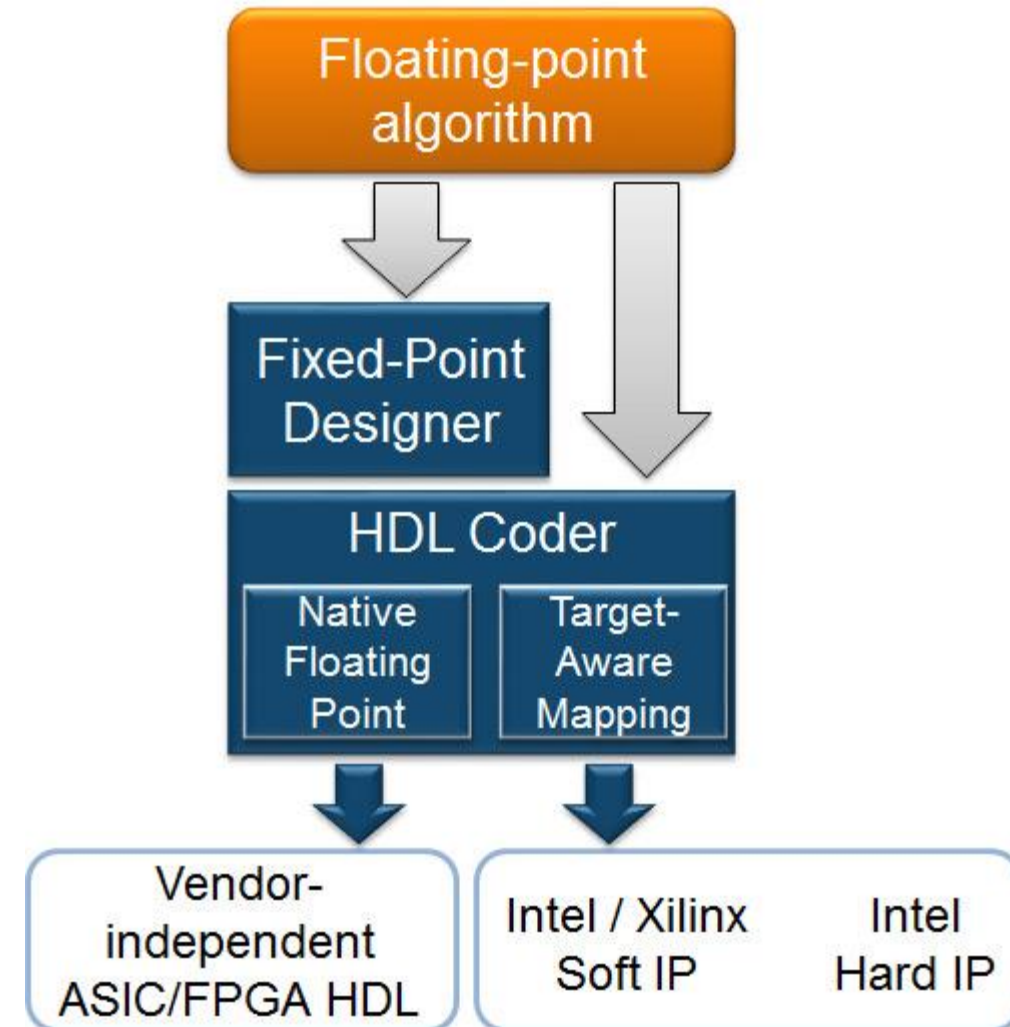


Target to Programmable Logic

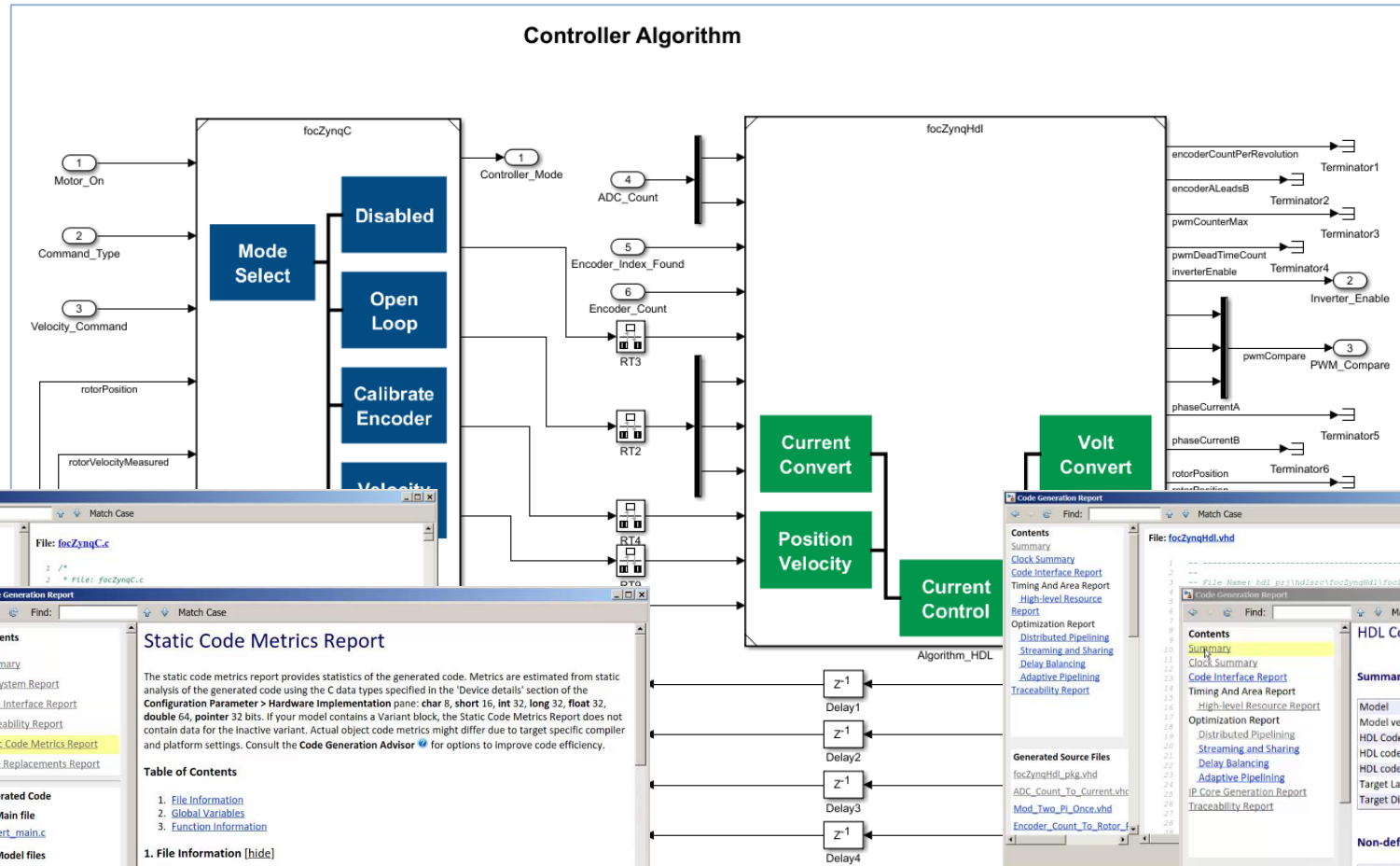


Floating-point to fixed-point conversion

- Is it always necessary?
- Possibly, to meet resource constraints on the hardware
- Fixed-Point Designer™ helps automate the conversion process
- HDL Coder™ native floating-point technology can generate HDL code from your floating-point design



Code Generation



Static Code Metrics Report

The static code metrics report provides statistics of the generated code. Metrics are estimated from static analysis of the generated code using the C data types specified in the 'Device details' section of the Configuration Parameter > Hardware Implementation pane: char 8, short 16, int 32, long 32, float 32, double 64, pointer 32 bits. If your model contains a Variant block, the Static Code Metrics Report does not contain data for the inactive variant. Actual object code metrics might differ due to target specific compiler and platform settings. Consult the Code Generation Advisor for options to improve code efficiency.

Table of Contents

- File Information
- Global Variables
- Function Information

1. File Information [hide]

(-) Summary (excludes ert_main.c)

Number of .c files : 5
 Number of .h files : 9
 Lines of code : 901
 Lines : 2,116

(-) File details

File Name	Lines of Code	Lines	Generated On
focZynqC.c	417	883	04/19/2017 9:48 PM
focZynqC.h	130	347	04/19/2017 9:48 PM
focZynqC_data.c	66	318	04/19/2017 9:48 PM

HDL Code Generation Report Summary for focZynqHdl

Summary

Model	focZynqHdl
Model version	1.368
HDL Code version	3.10
HDL code generated on	2017-04-21 14:19:09
HDL code generated for	focZynqHdl
Target Language	VHDL
Target Directory	hdl_prj\hdlsrc

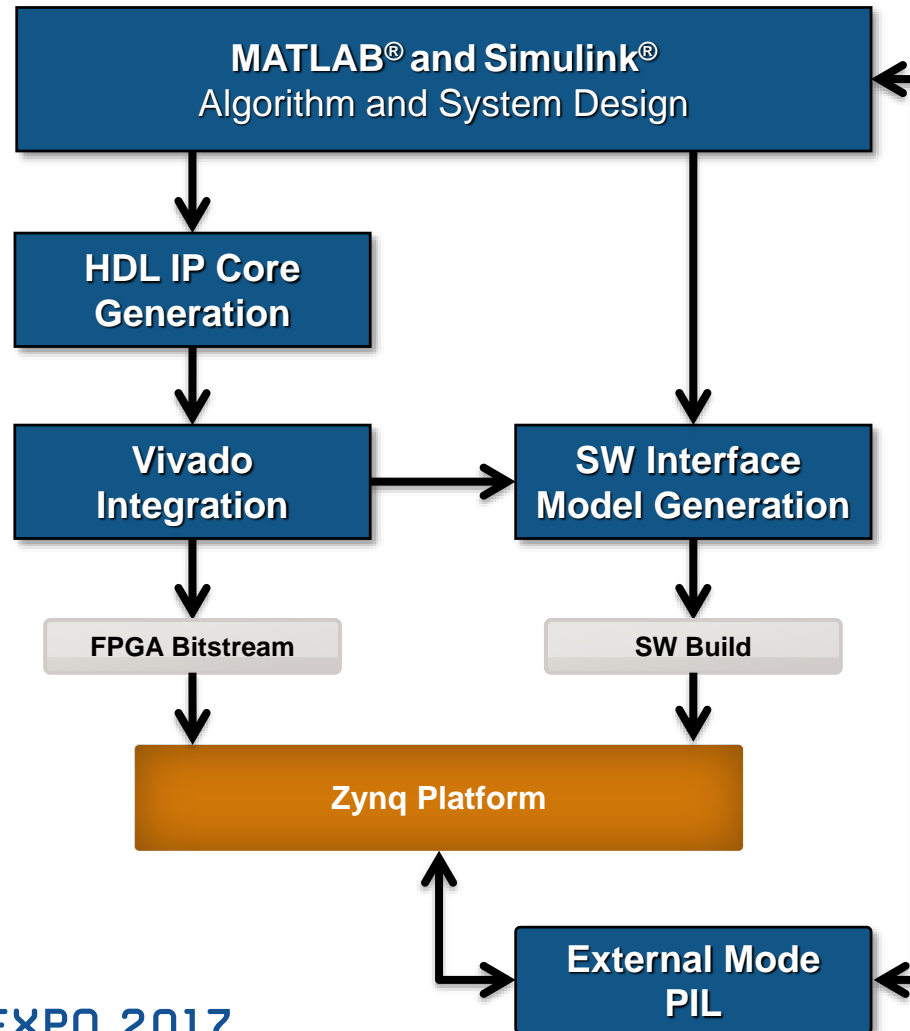
Non-default model properties

ClockRatePipelining	off
EnablePrefix	oversampledClockEnable
HDLSubsystem	focZynqHdl
ModulePrefix	focZynqHdl_ip_src_
OptimizationReport	on
Oversampling	2000
ReferenceDesign	Motor Control Reference Design
ResetType	Synchronous
ResourceReport	on
ScalarizePorts	on
SynthesisTool	Xilinx Vivado
SynthesisToolChainFamily	7ynn

Generated Source Files

- focZynqHdl_ip_src_focZynqHdl_ip
- focZynqHdl_ip_src_ADC_Count_T
- focZynqHdl_ip_src_Mod_Two_PI
- focZynqHdl_ip_src_Encoder_Cour
- focZynqHdl_ip_src_Wrap_Neg_PI
- focZynqHdl_ip_src_Rotor_Position
- focZynqHdl_ip_src_Rotor_Position
- focZynqHdl_ip_src_Rotor_Position

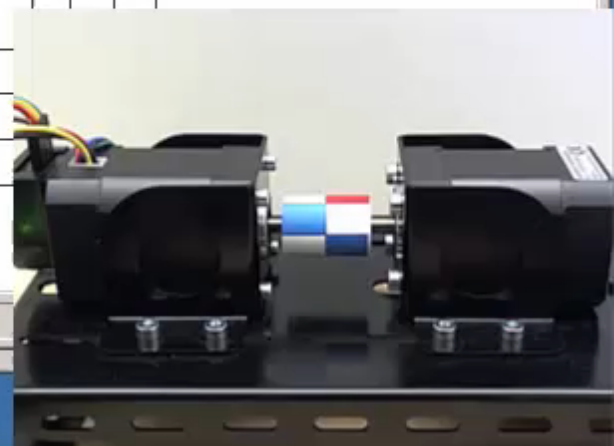
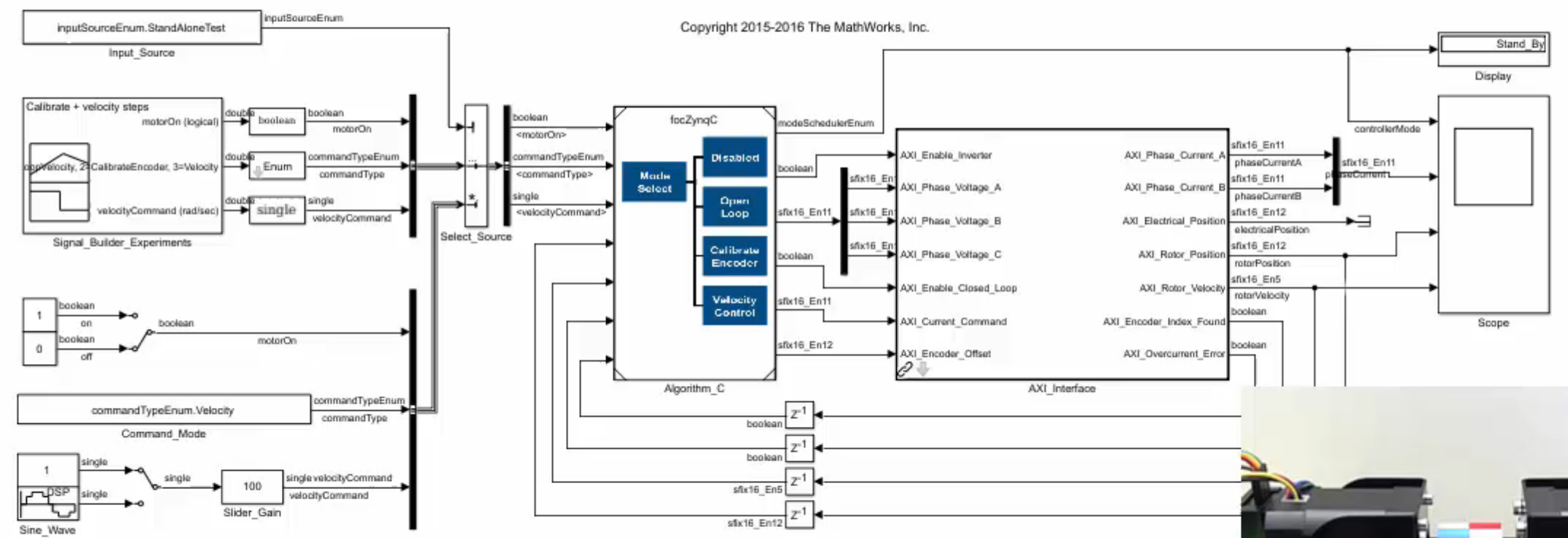
Zynq Model-Based Design Workflow



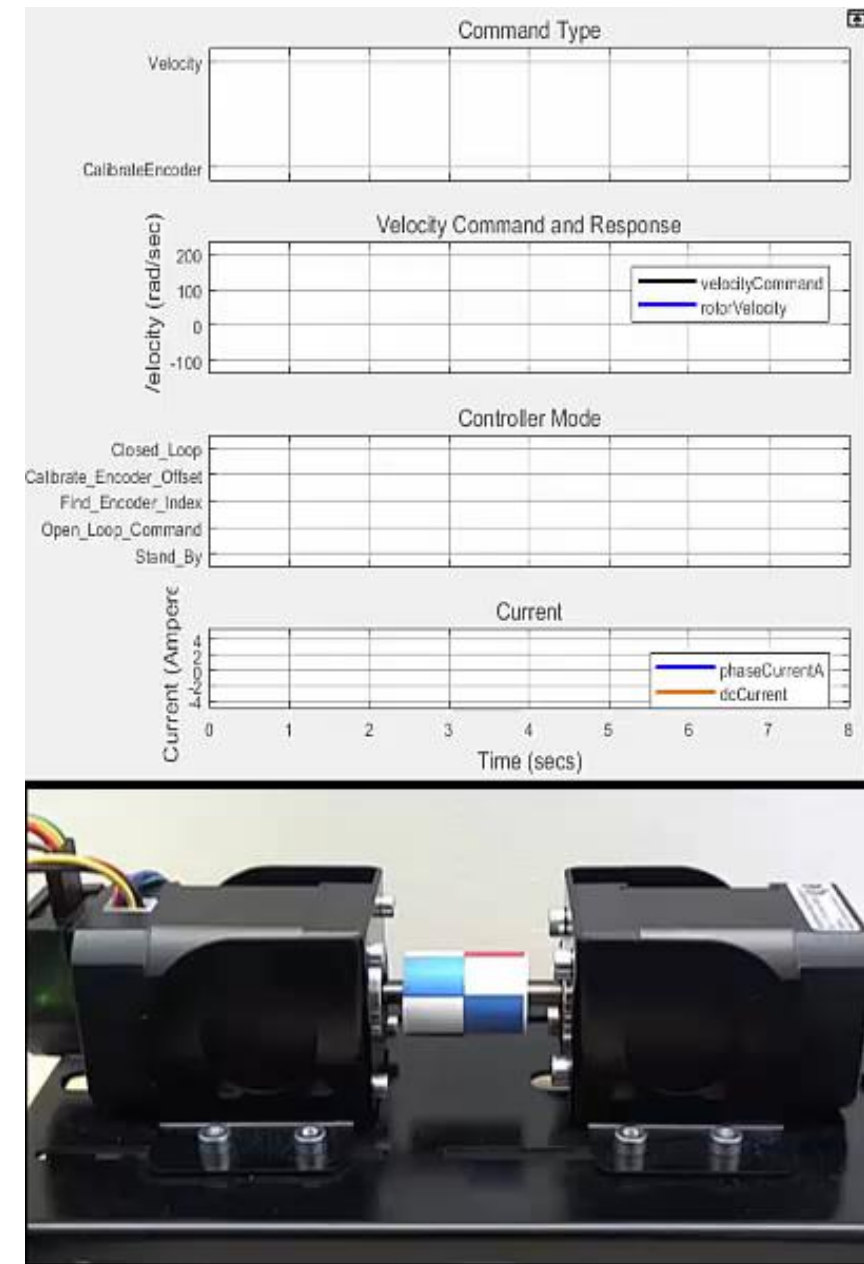
- Real-time Parameter Tuning and Verification
 - External Mode
 - Processor-in-the-loop
- More probe and debug capability in the future

Field-Oriented Control of Velocity Zynq ARM Deployment for AD-FMCMOTCON2

Copyright 2015-2016 The MathWorks, Inc.



- Why use Hardware and Software for motor control?
- Why use Model-Based Design for motor control?
- How to use Model-Based Design for motor control?



Why use Model-Based Design to develop motor control applications?

- Enables early validation of specifications using simulation months before hardware is available.
- Dramatically improves design team collaboration and designer productivity by using a single design environment.
- Reduces hardware testing time by 5x by shifting design from lab to the desktop