

A blue-toned photograph of the Georgia Tech Campanile, a historic red brick building with a tall, spired tower. The word "TECH" is prominently displayed in large letters on the side of the tower. The image is framed by a thin yellow border.

MBSD Lecture 0

Overview

Course Goals

- Present and emphasize MBSD approach
 - Simple models
 - Complex system
 - Basic control
 - Iterative refinement
- Utilize simulation tools
 - Matlab
 - Simulink
 - Stateflow

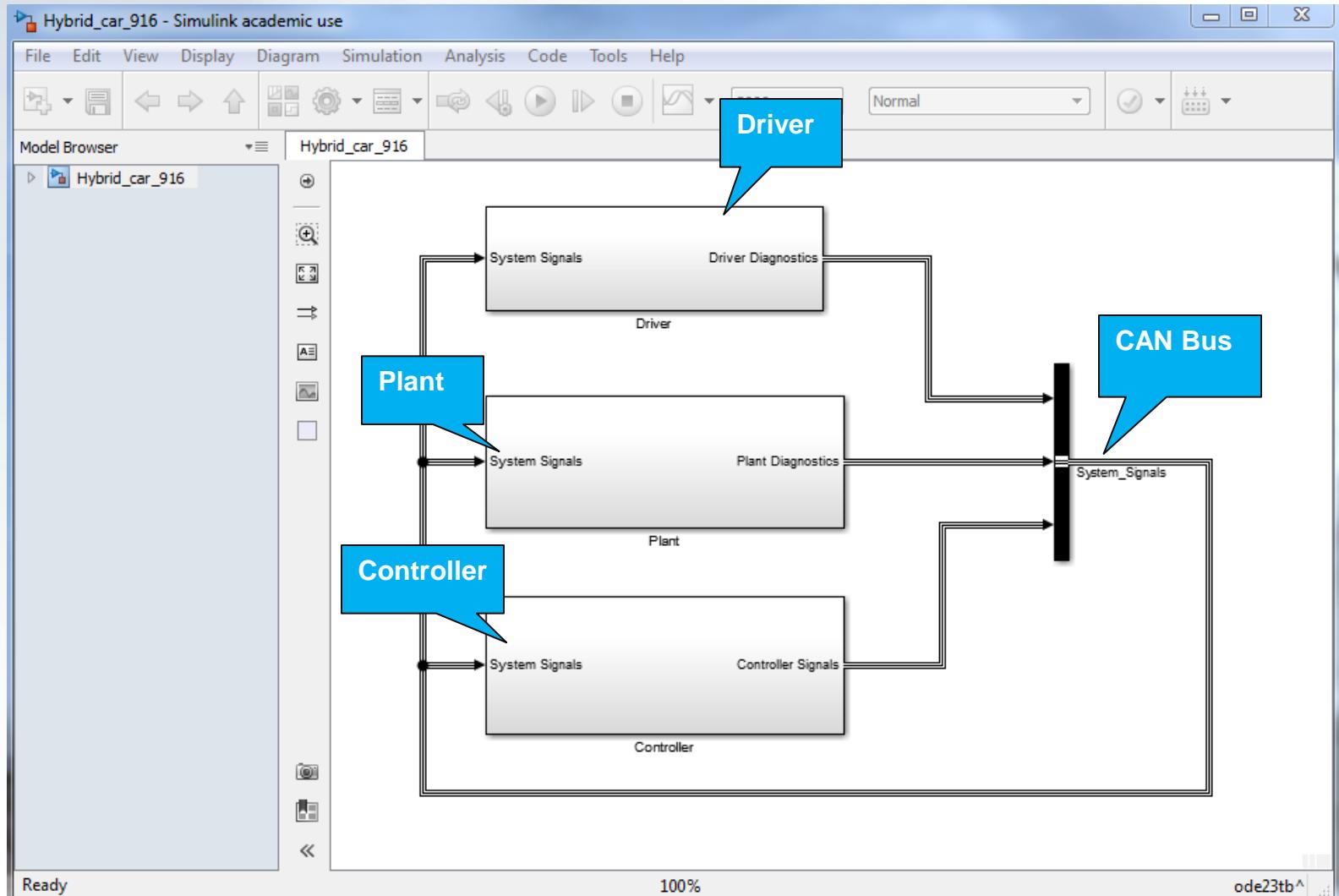


Course Outline

- Simple Series Electric Vehicle
 - Model hierarchy
 - Ideal Component Models
 - Controls
- Improved Component Models
 - Inefficiencies
 - Experimental Data
- Improved Controls

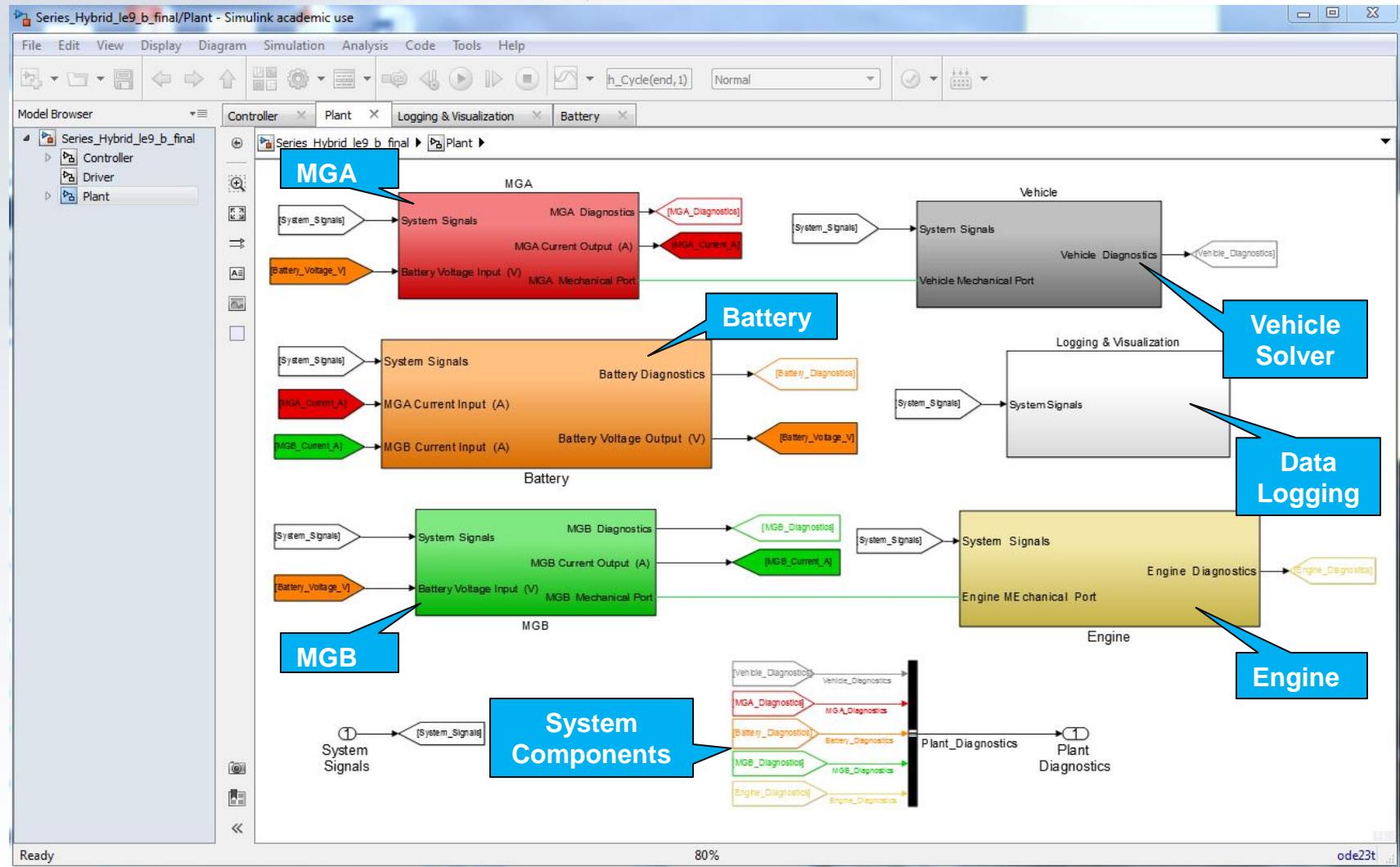


Model Overview - System

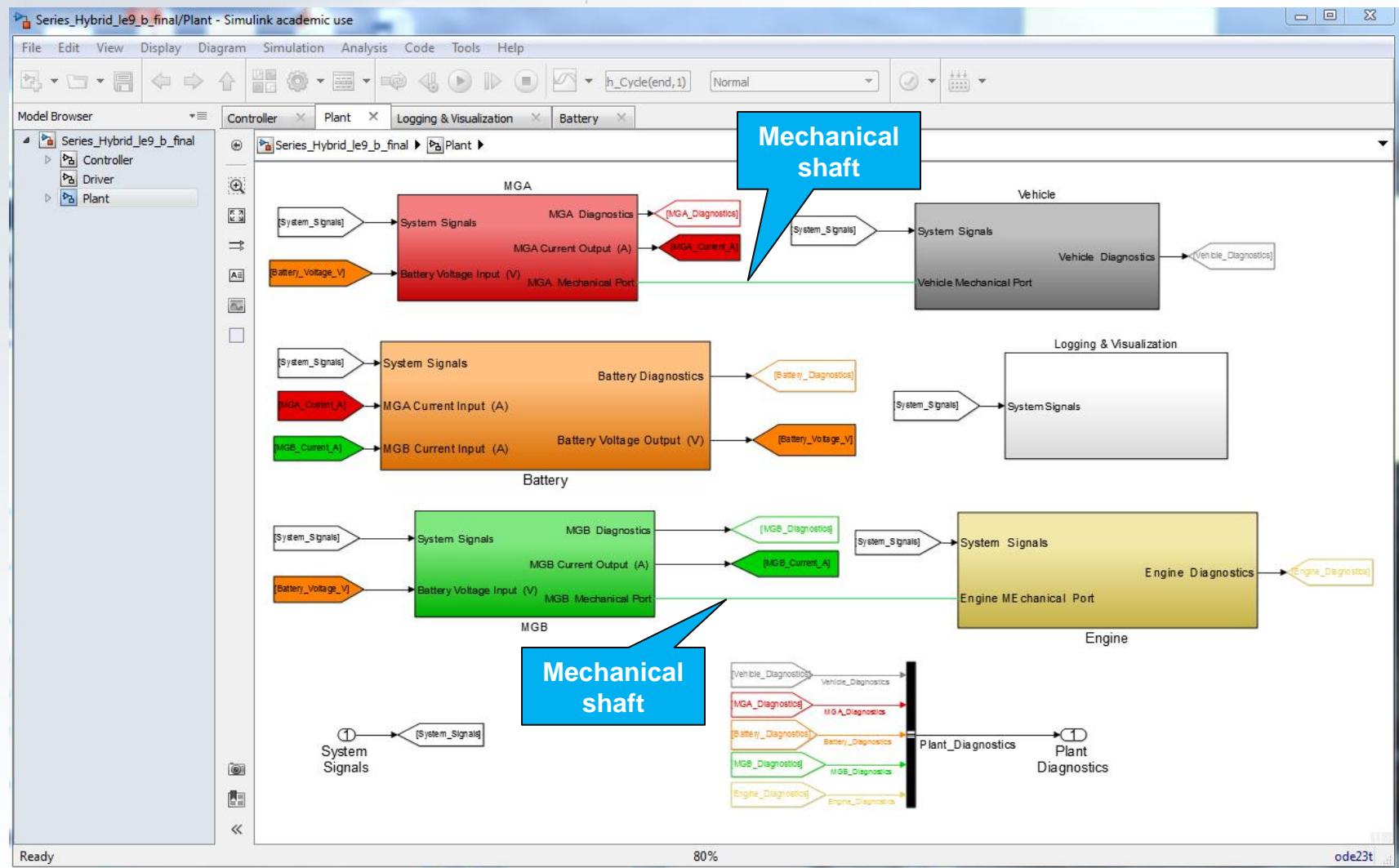




Model Overview - Plant

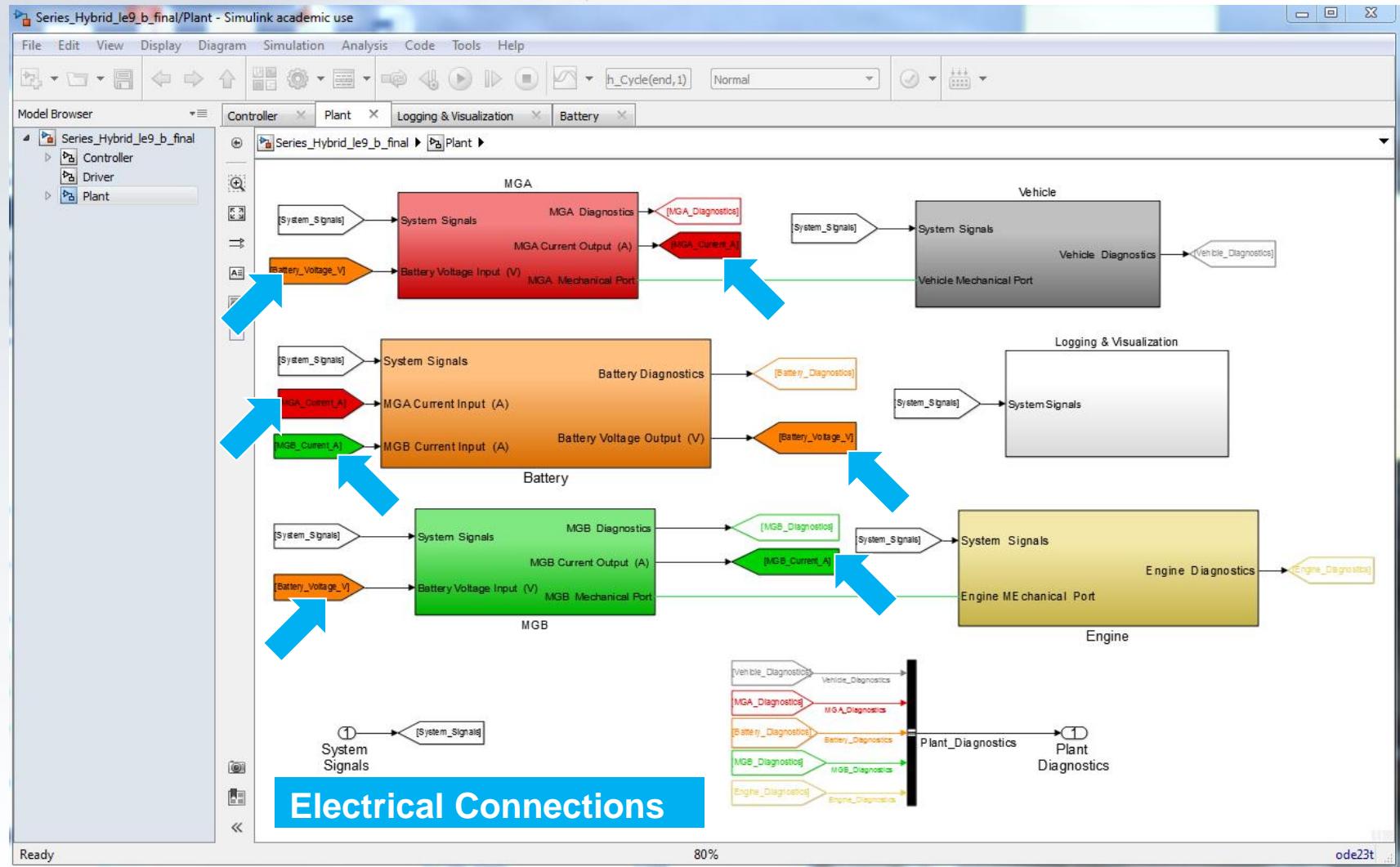


Model Overview - Plant



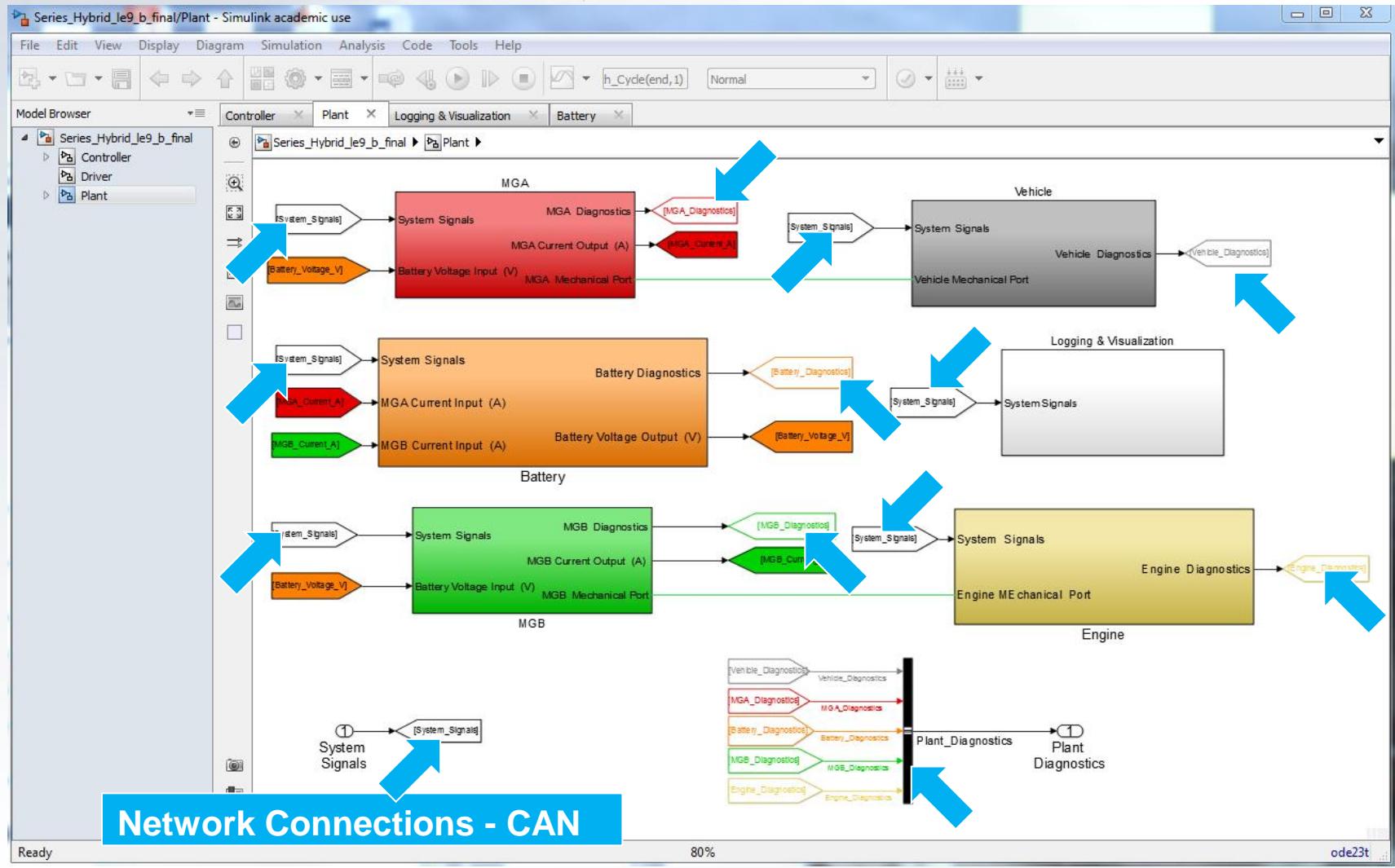


Model Overview - Plant



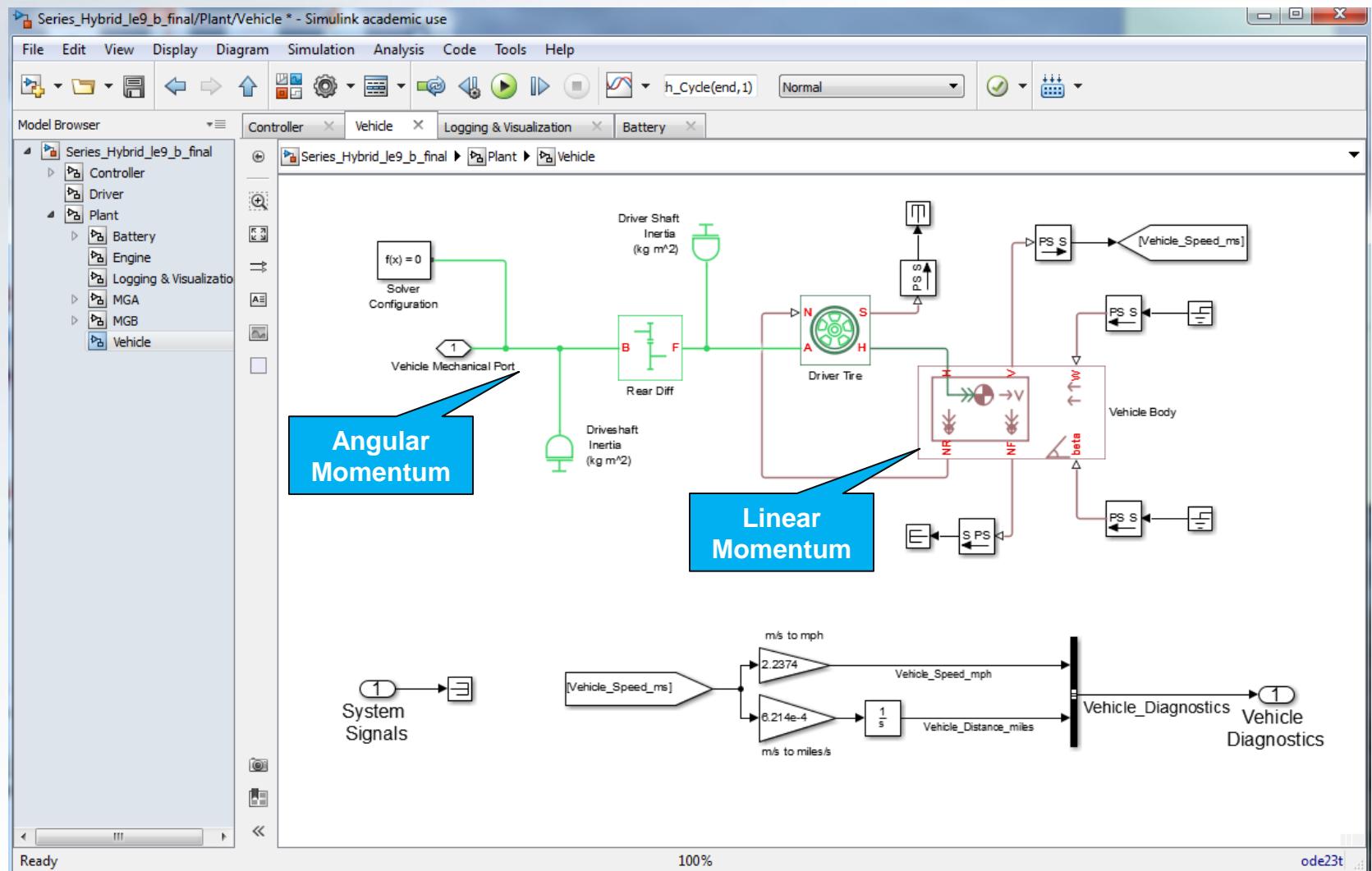


Model Overview - Plant



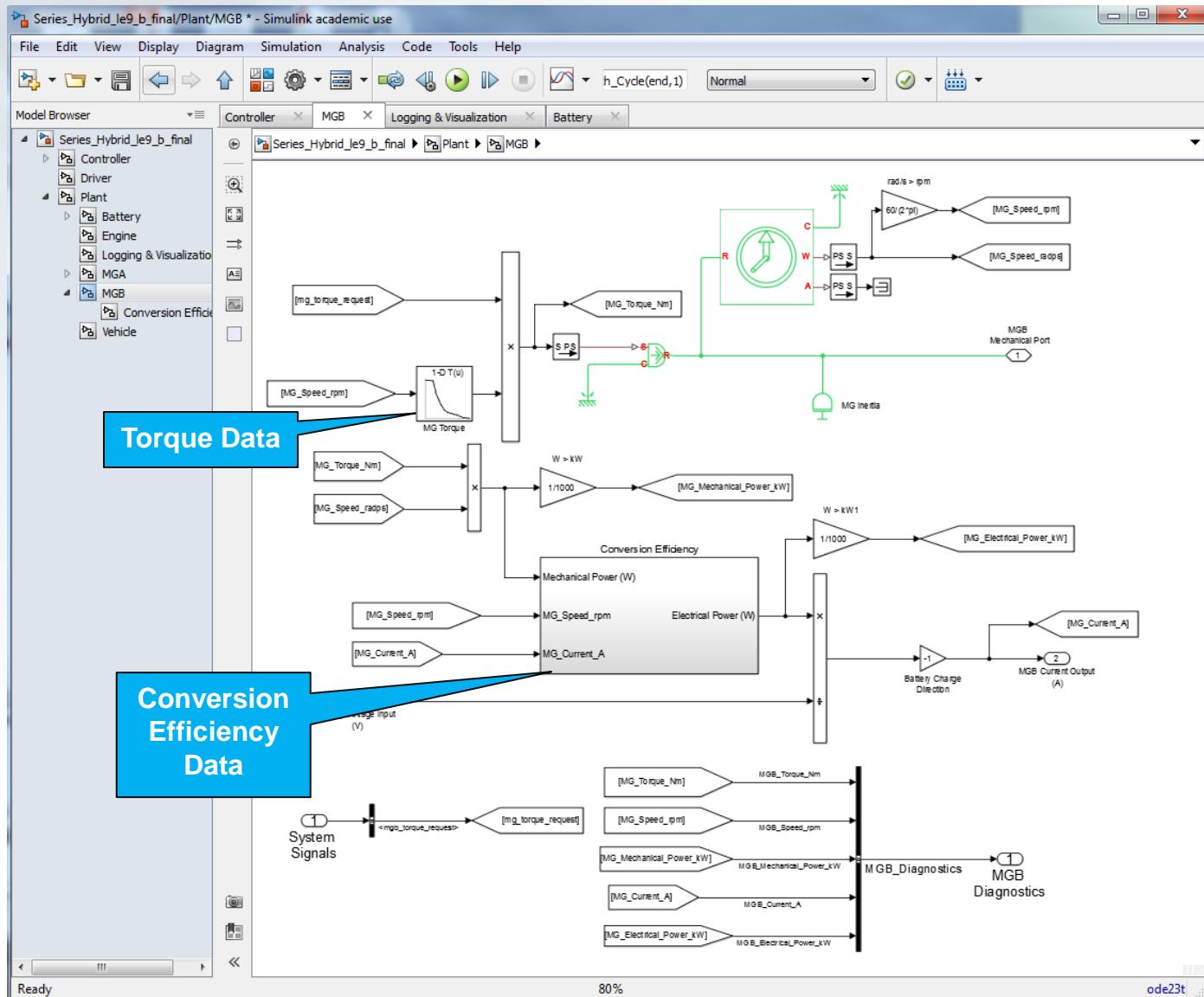


Model Overview - Vehicle

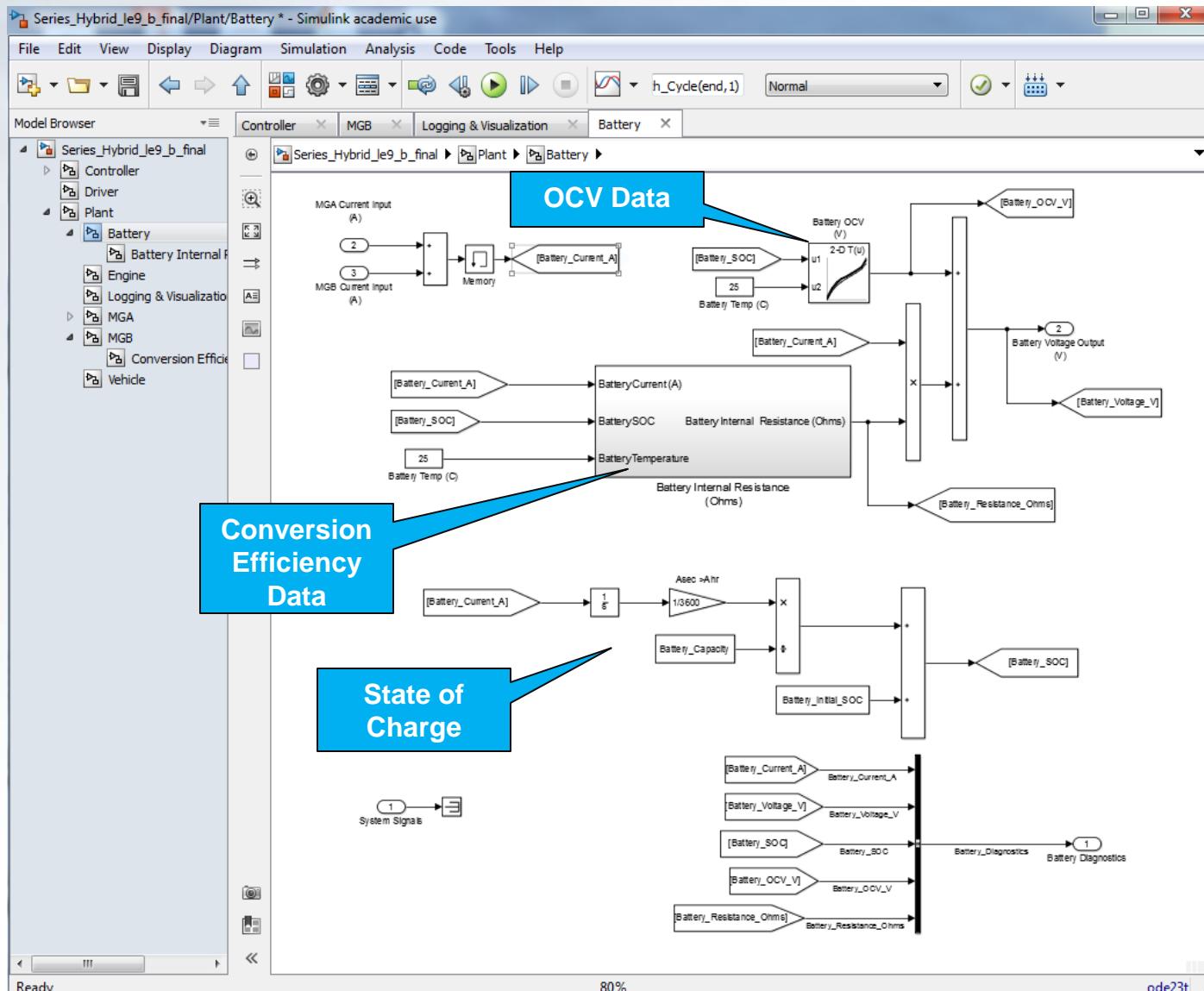




Model Overview - MG

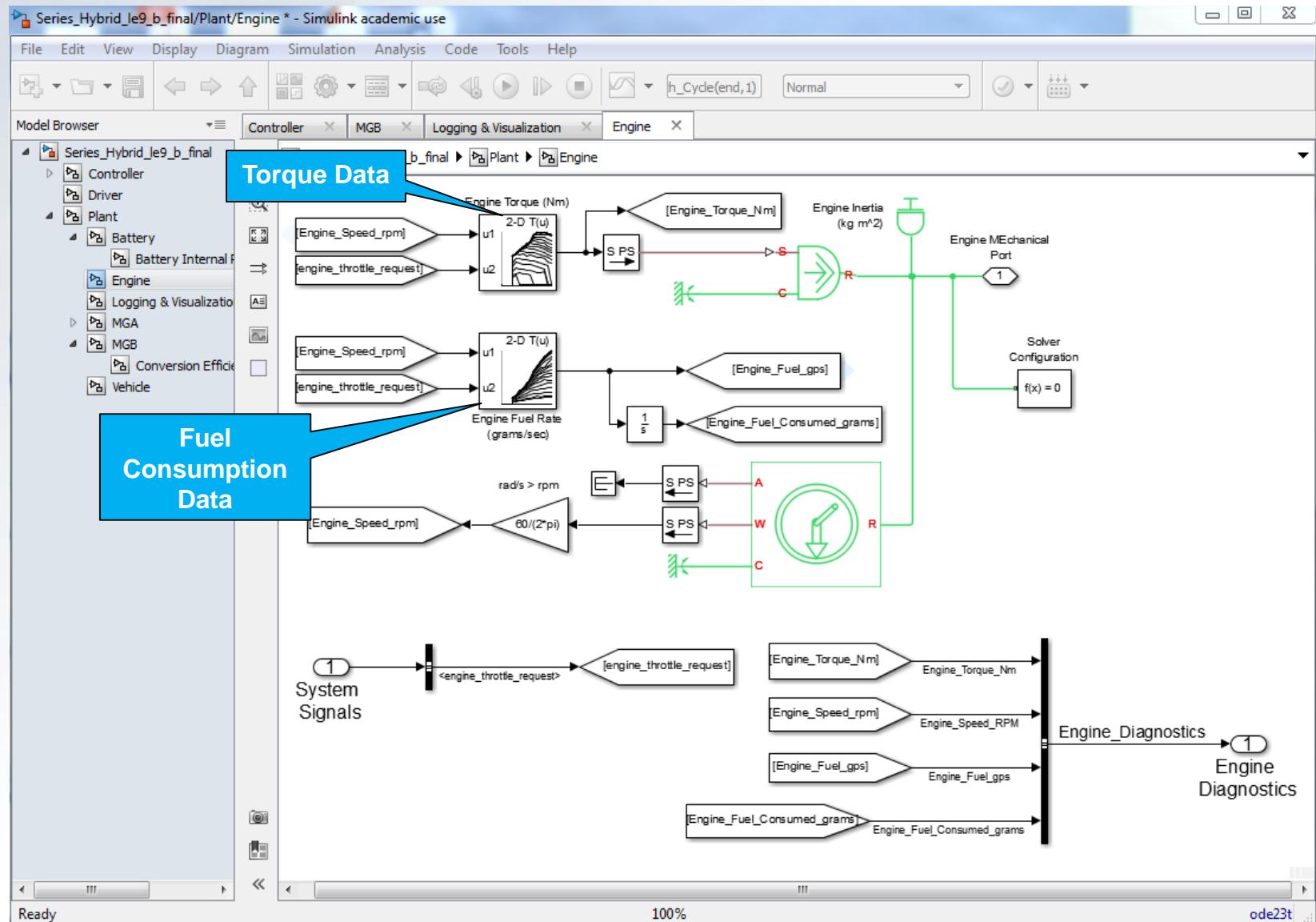


Model Overview – Battery



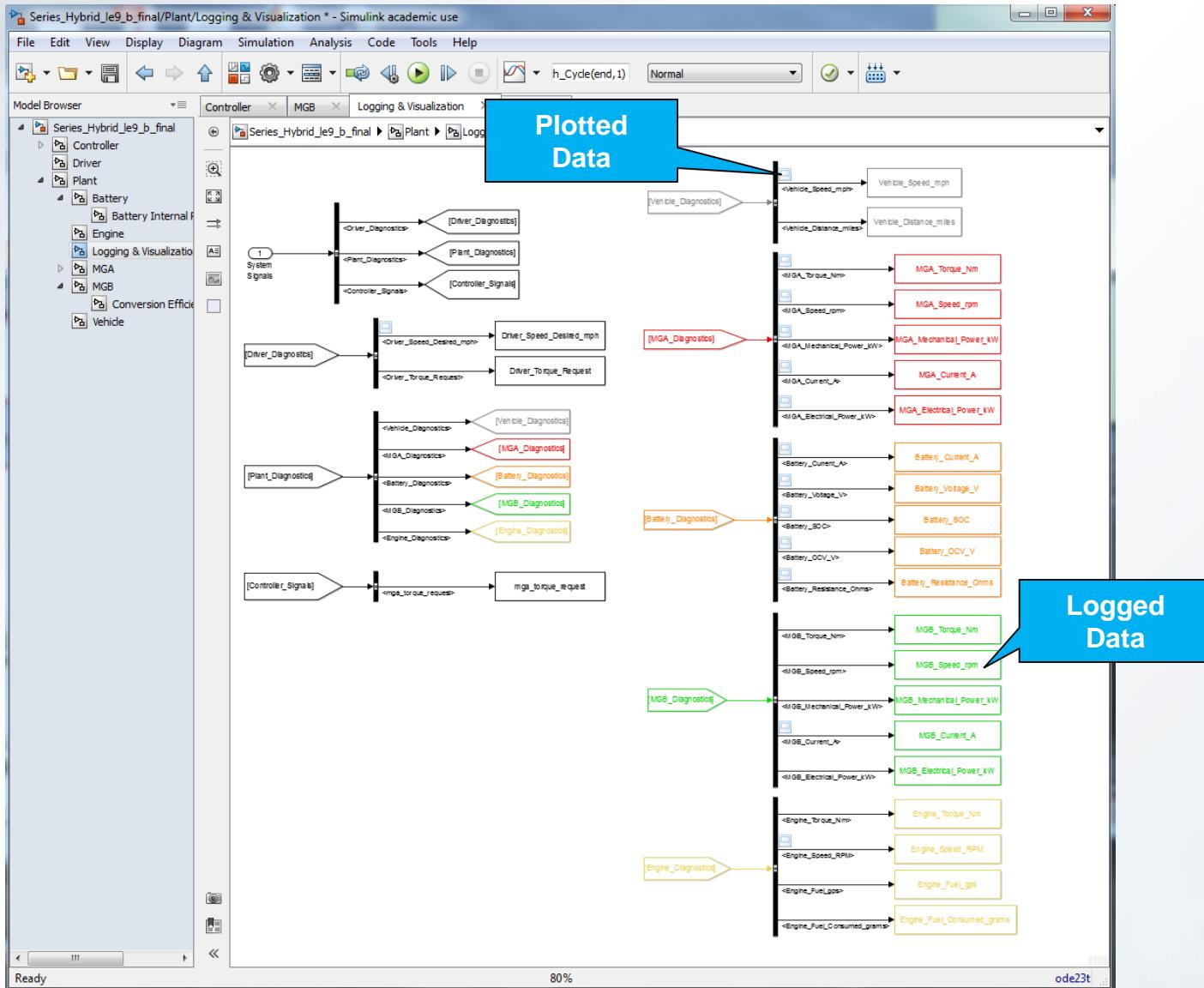


Model Overview - Engine



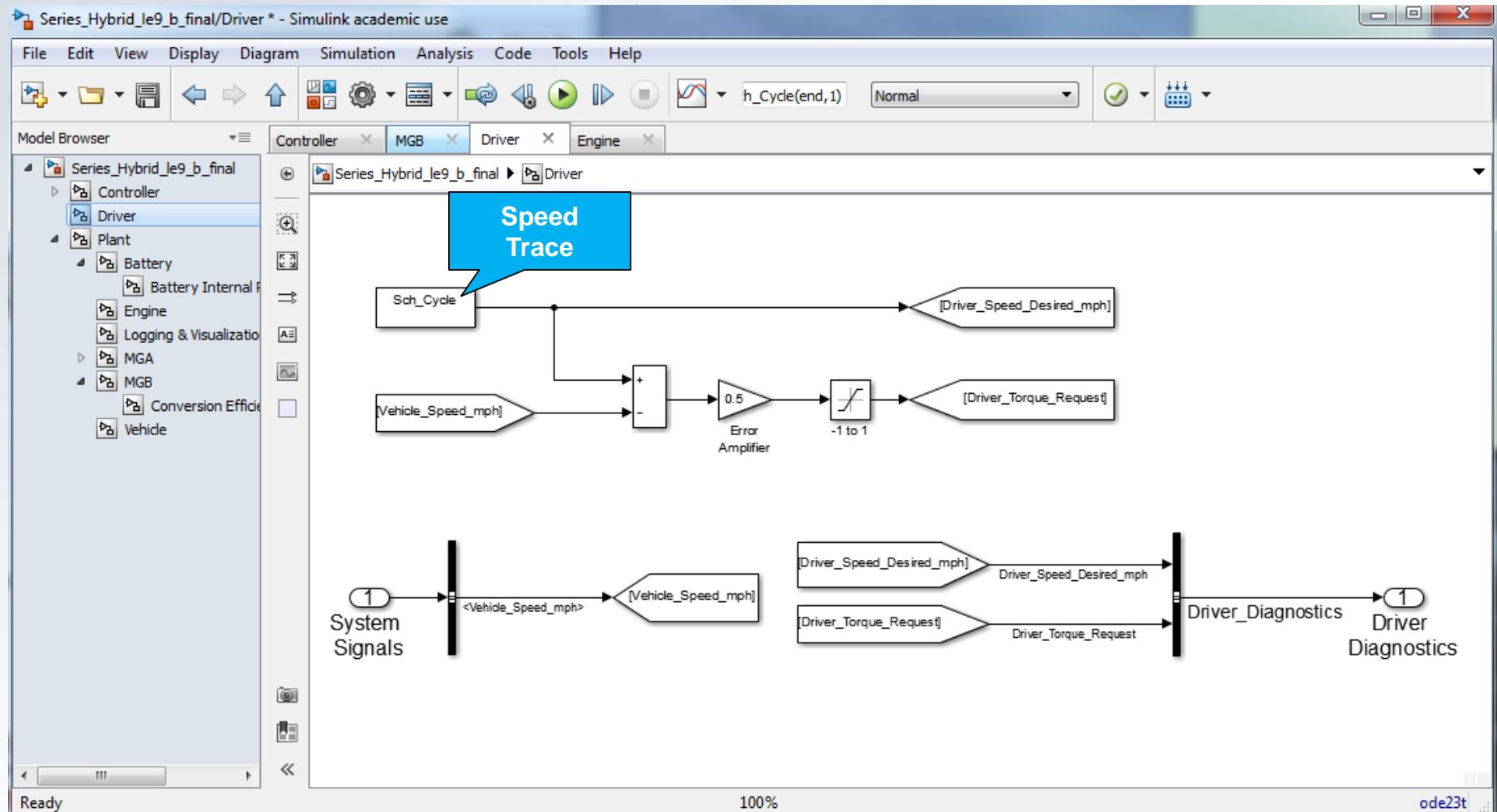


Model Overview - Logging



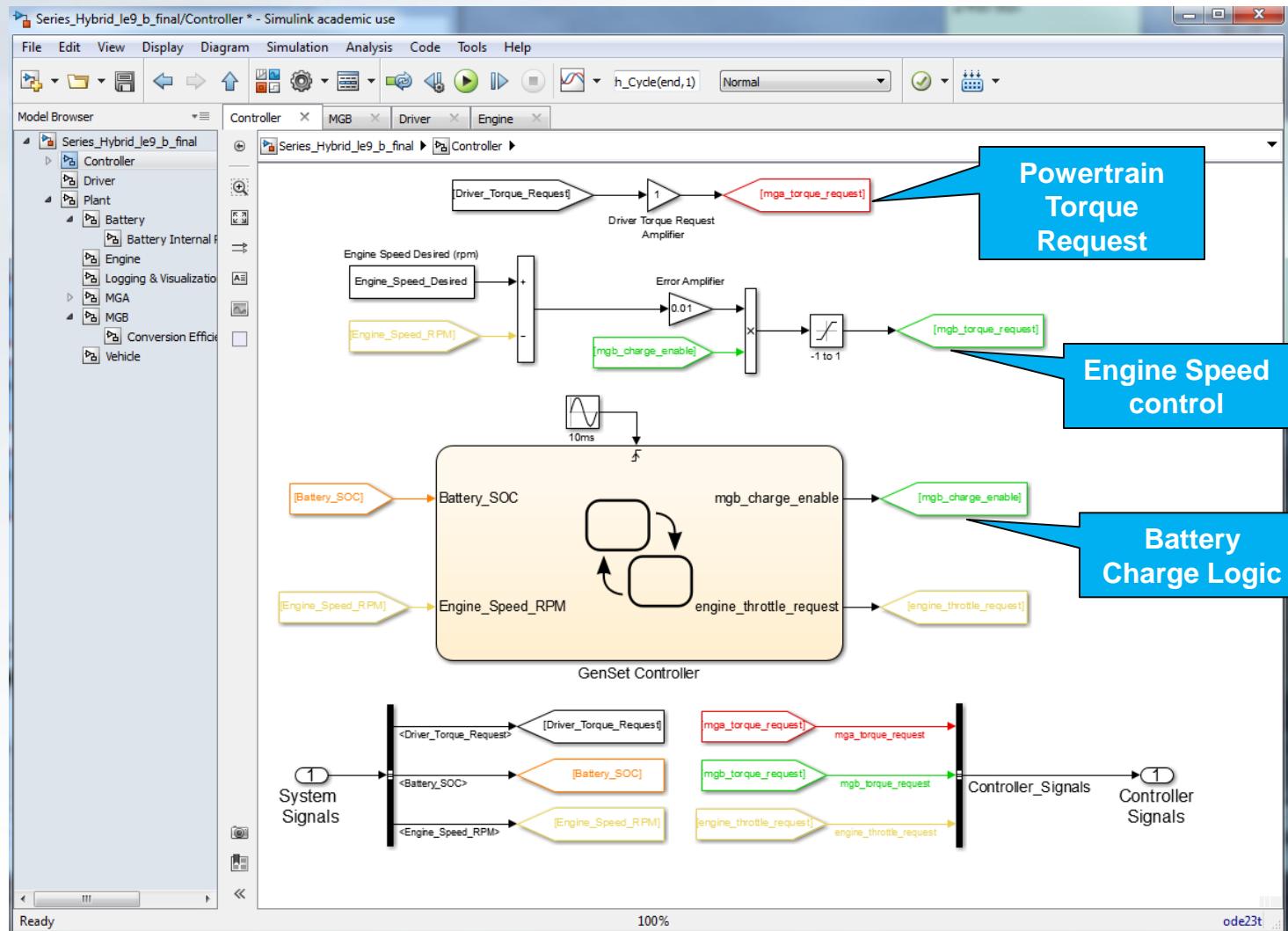


Model Overview - Driver



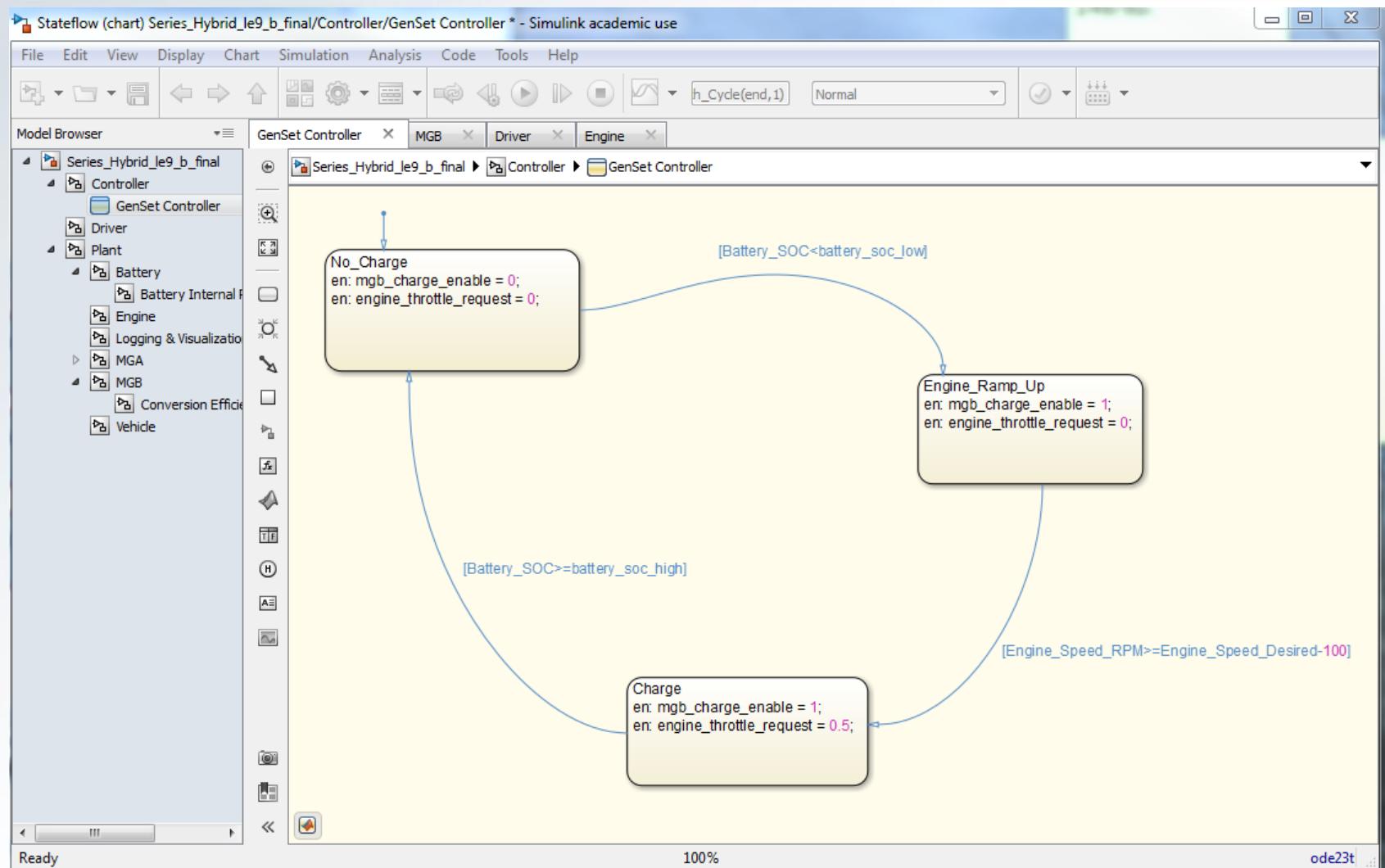


Model Overview - Controller



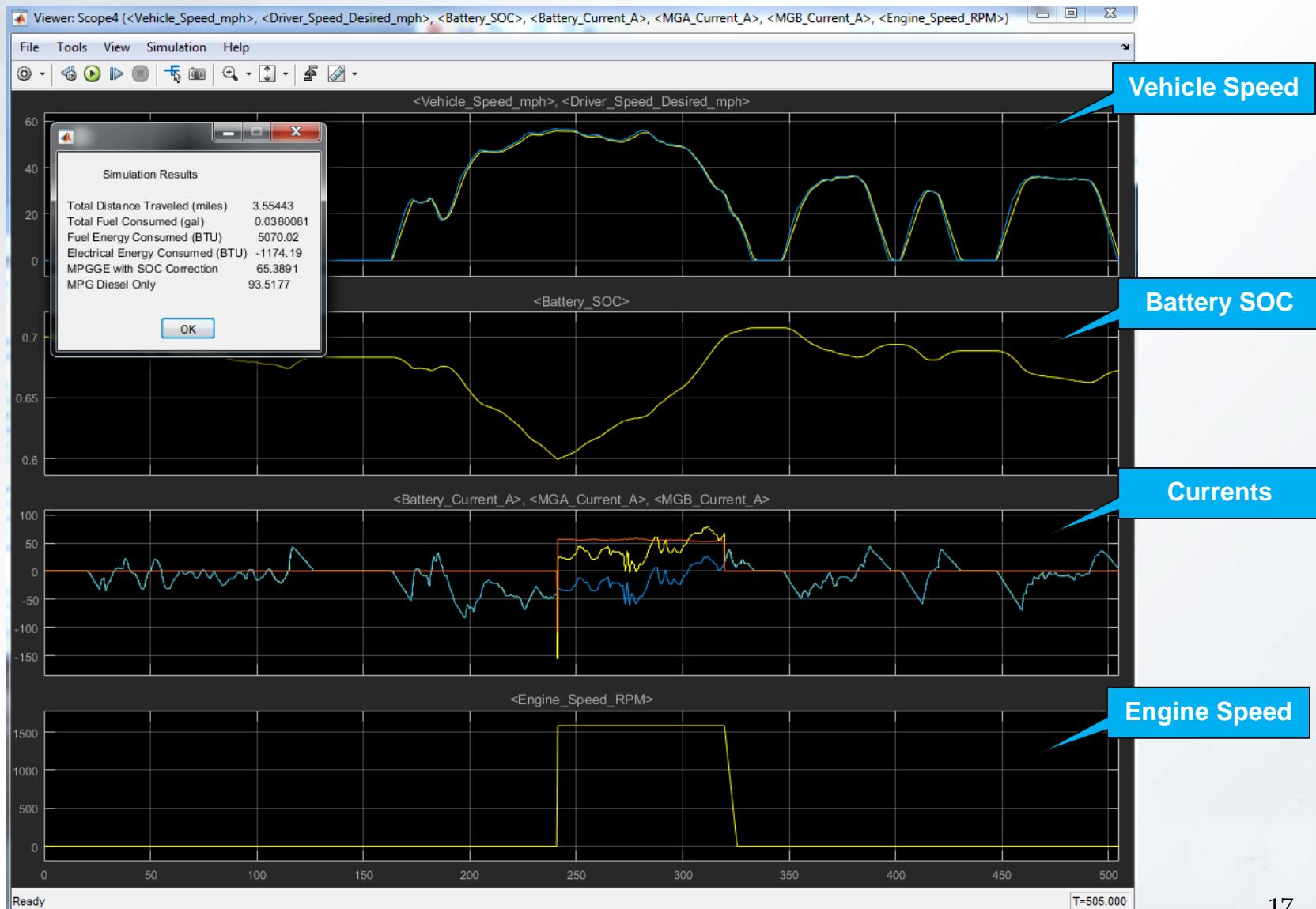


Model Overview - GentSet





Results



A blue-toned photograph of the Georgia Tech Campanile, showing its brick facade and the prominent 'TECH' sign on top.

MBSD Lecture 1

Getting Started with the High Level System



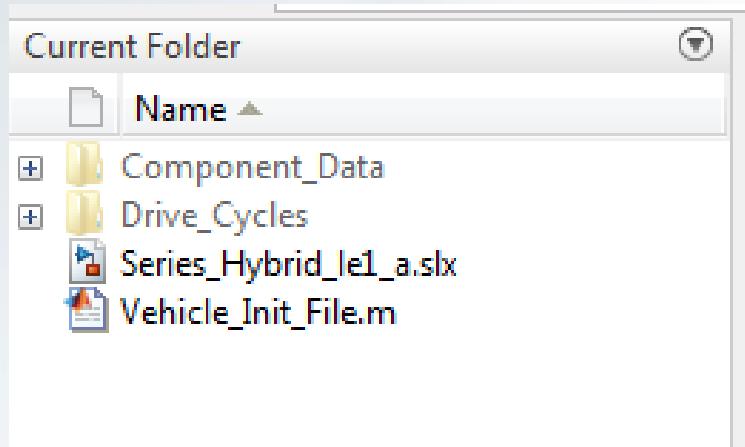
Lecture Goals

- Getting Started with Simulink
- Developing model hierarchy (subsystems)
- Establishing CAN bus



Getting Started

- Start Matlab
- Make sure you are in the correct directory



- Type **simulink** in the command window



Getting Started

The screenshot shows the MATLAB R2016a interface. The Command Window displays the command `>> simulink`. The Current Folder browser shows a directory structure under `C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model`, with files like `Component_Data`, `Drive_Cycles`, `Series_Hybrid_le1_a.slx`, and `Vehicle_Init_File.m`. A red box highlights the folder icon in the toolbar of the Current Folder browser. A blue callout points to this icon with the text: "This button allows you to choose the directory". Another blue callout points to the `Simulink` button in the toolbar with the text: "Instead of typing Simulink in the command directly you can select this button". A third blue callout points to the left edge of the workspace browser with the text: "Make sure all files and folders exist in your directory".



Getting Started

Simulink Start Page

SIMULINK®

New Examples

Open...

Recent

Series_Hybrid_le1_a.slx

Projects

Source Control...

Archive...

Select Blank model

Search All Templates

My Templates Learn More

You have not created any templates. [Learn how to create templates.](#)

Simulink

Blank Model

Blank Library

Blank Project

Code Generation

Digital Filter

Feedback Controller

Fixed-step

HDL Code Generation

Signal Processing

Simple Project

Simple Simulation

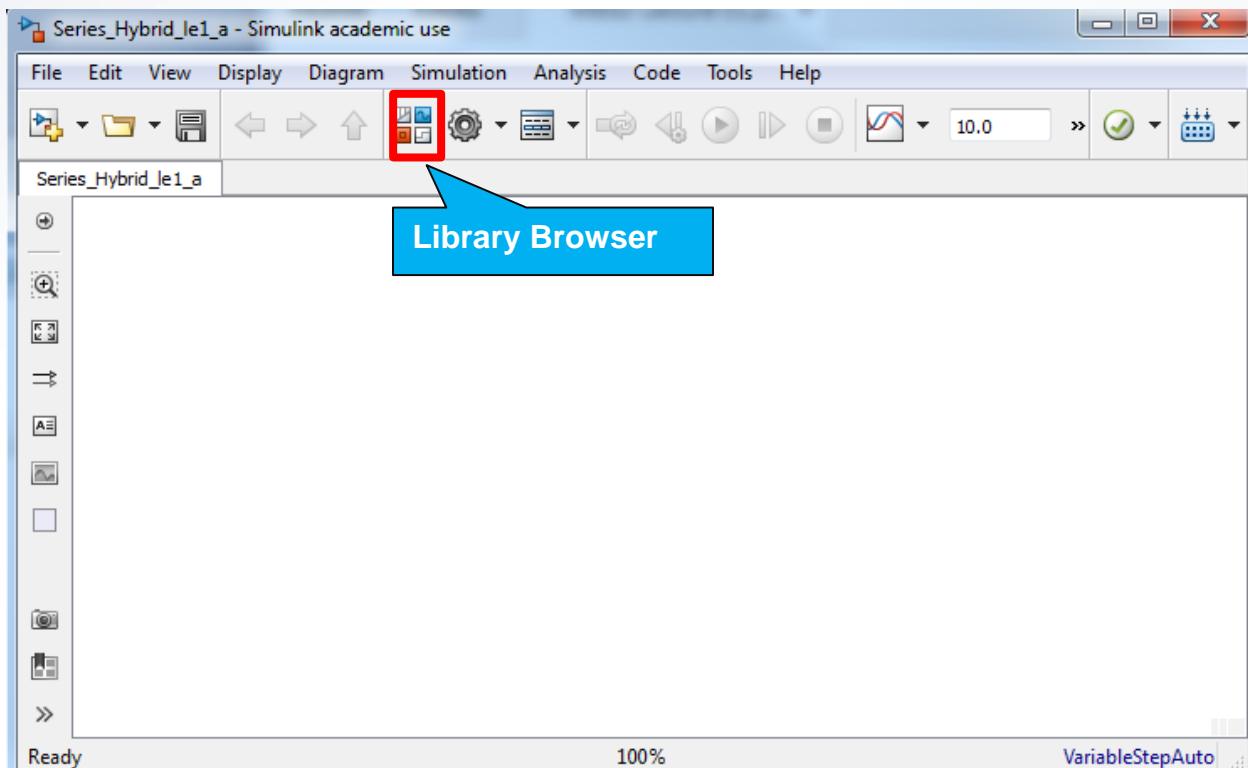
Aerospace Blockset

The screenshot shows the MATLAB Simulink Start Page. On the left, there's a sidebar with 'Recent' projects and 'Projects' like 'Source Control...' and 'Archive...'. The main area has tabs for 'New' and 'Examples'. Under 'New', there's a search bar and a dropdown for 'All Templates'. Below that are sections for 'My Templates' (which is empty) and 'Simulink'. The 'Simulink' section contains eight templates: 'Blank Model', 'Blank Library', 'Blank Project', 'Code Generation', 'Digital Filter', 'Feedback Controller', 'Fixed-step', 'HDL Code Generation', 'Signal Processing', 'Simple Project', and 'Simple Simulation'. Each template has a small preview diagram. A large blue callout box with the text 'Select Blank model' and a pointing arrow is positioned over the first template, 'Blank Model'.



Getting Started

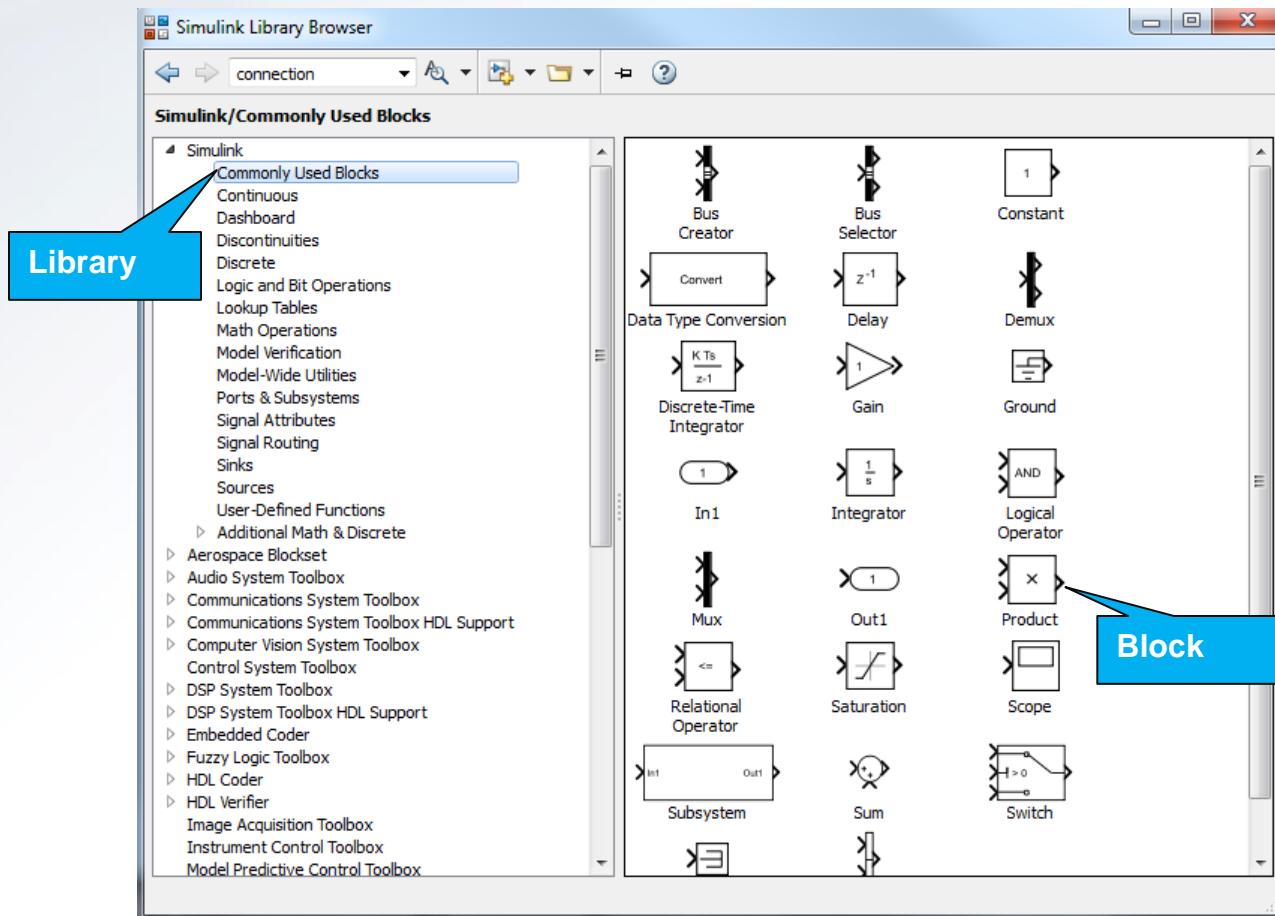
- Simulink is a graphical programming environment which enables you to link function blocks to create simulations
- In the Simulink model window, select library browser





Getting Started

- Get yourself familiar with the libraries and blocks in the library browser





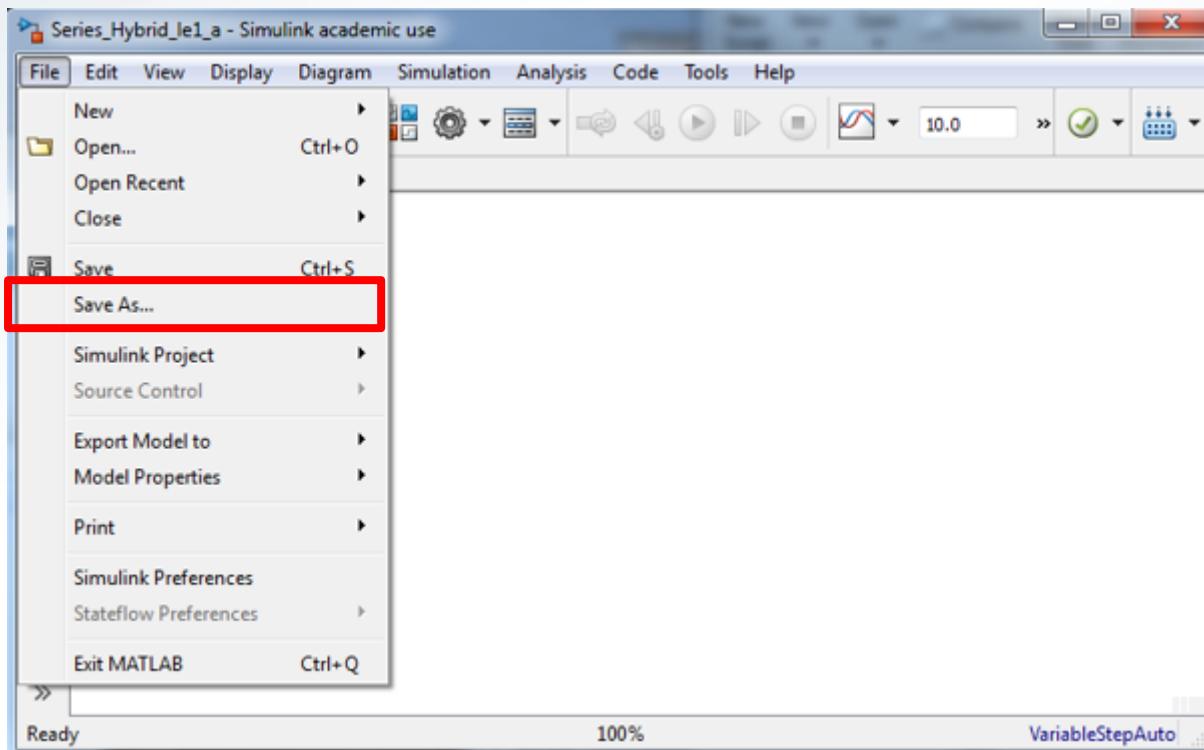
Getting Started

- We will primarily use the following libraries
 - Simulink \ Commonly Used Blocks
 - Simulink \ Signal Routing
 - Simulink \ Lookup Tables
 - Simulink \ Sinks
 - Simulink \ Sources
 - Simscape \ Driveline
 - Stateflow



Getting Started

- Save the Simulink model as :
 - [Series_Hybrid_le1_a](#)
- Do not use spaces or variable names in file names



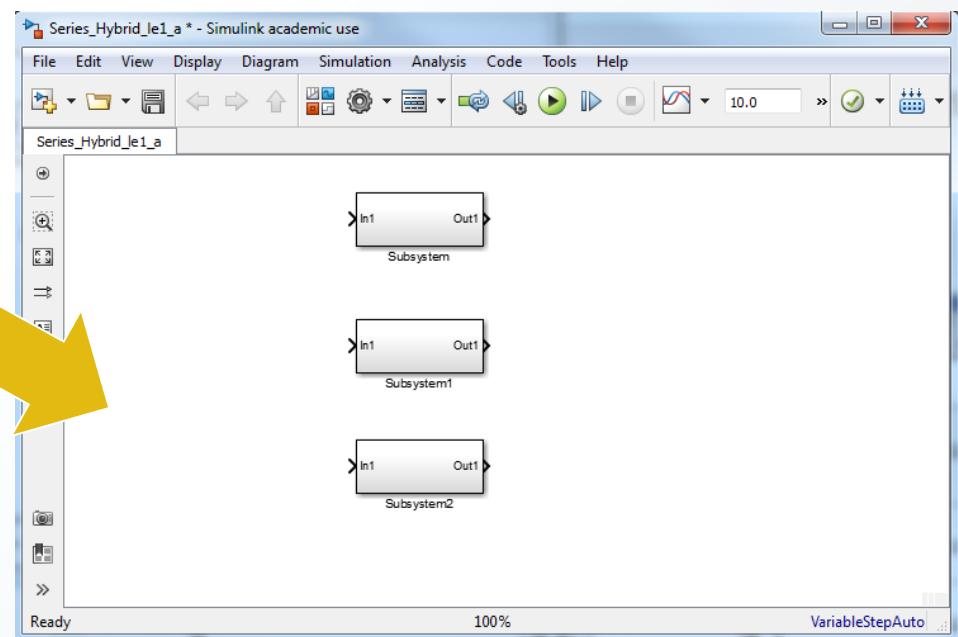
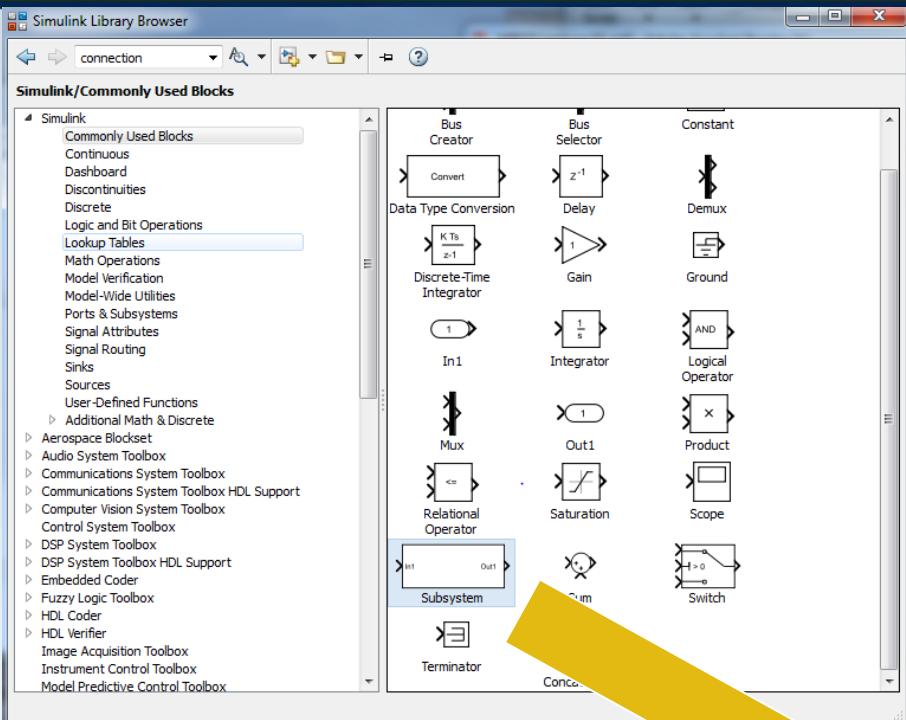


High Level Model

- We will first create the high level plant and controller model
- From
 - [Simulink \ Commonly Used Blocks](#)
- Drag three **Subsystem** blocks into the model
 - [Driver block](#)
 - [Controller block](#)
 - [Vehicle block](#)



High Level Model





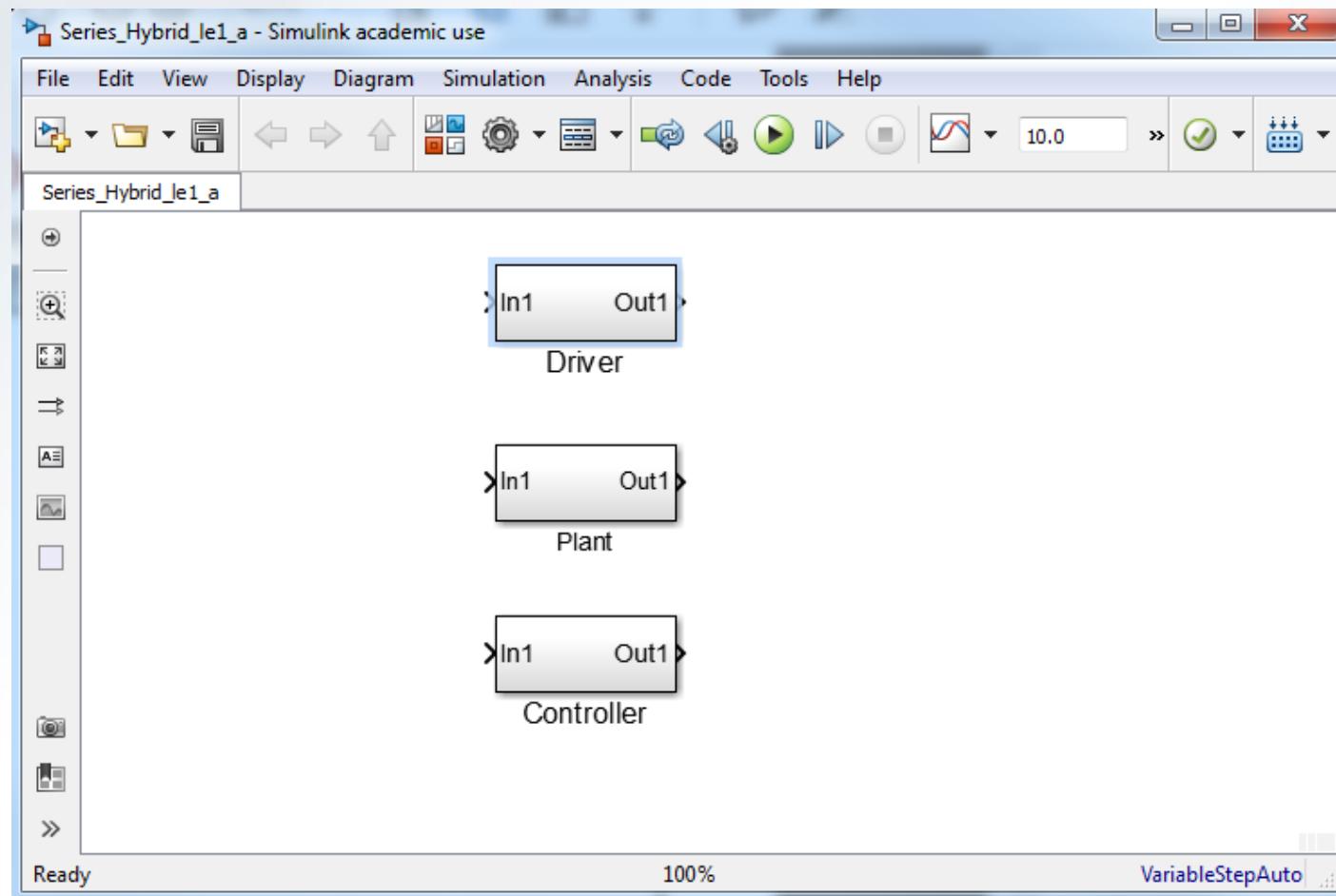
High Level Model

- Rename the upper subsystem
 - Driver
- The middle subsystem
 - Plant
- The lower subsystem
 - Controller



High Level Model

- You can change the font size by right clicking on each subsystem/Format/Font Style...





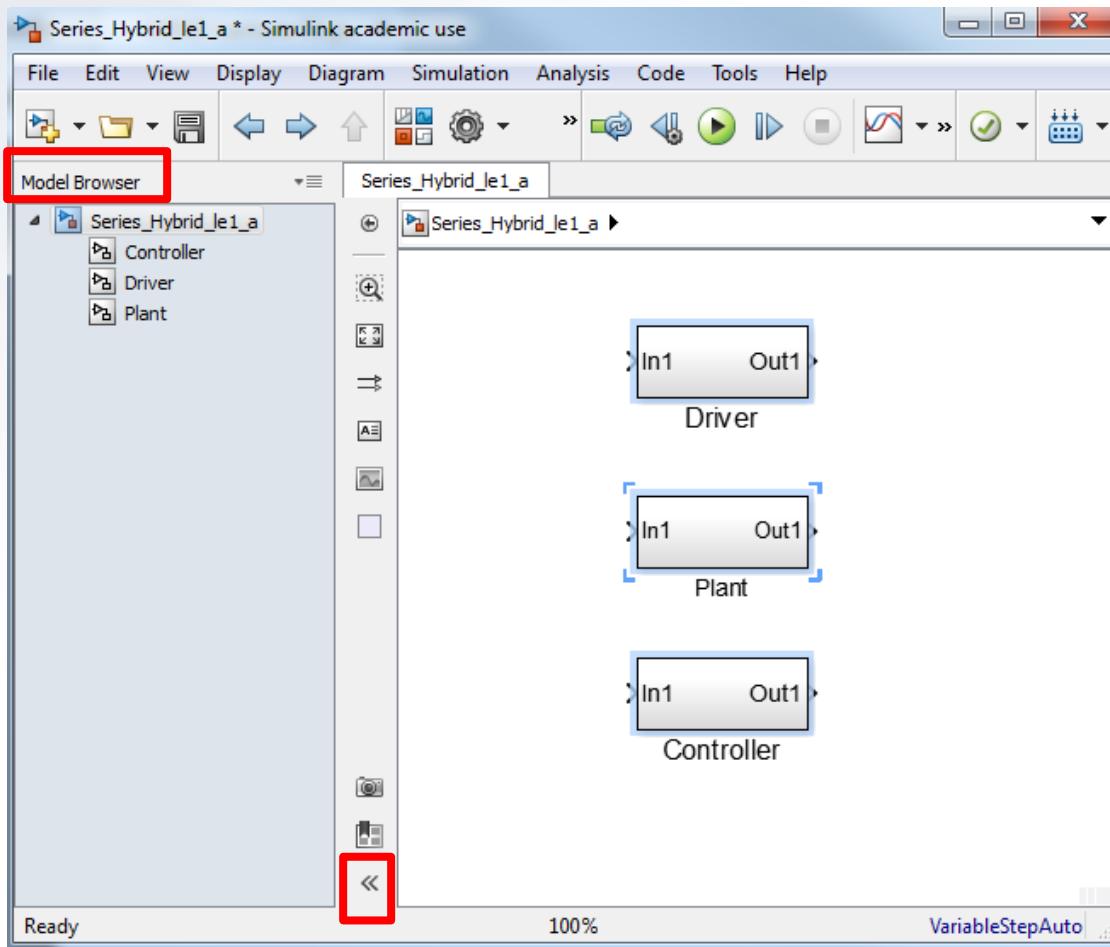
High Level Model

- At this high level, signals will be sent out by the Driver, Plant, and Controller
 - Driver Torque Request
 - Battery State of Charge
 - Engine Throttle Request
- These signals will be put onto a common bus so that all three subsystems can receive all signals
 - Only desired signals will be selected



High Level Model

- Open Model Browser by selecting the arrow
 - Shows an outline form of the model





Driver Subsystem

- The Driver receives information
 - Speedometer
 - Tachometer
 - Idiot lights (engine hot,)
- The Driver sends information
 - Torque request (gas/brake pedal position)
 - Gear selection

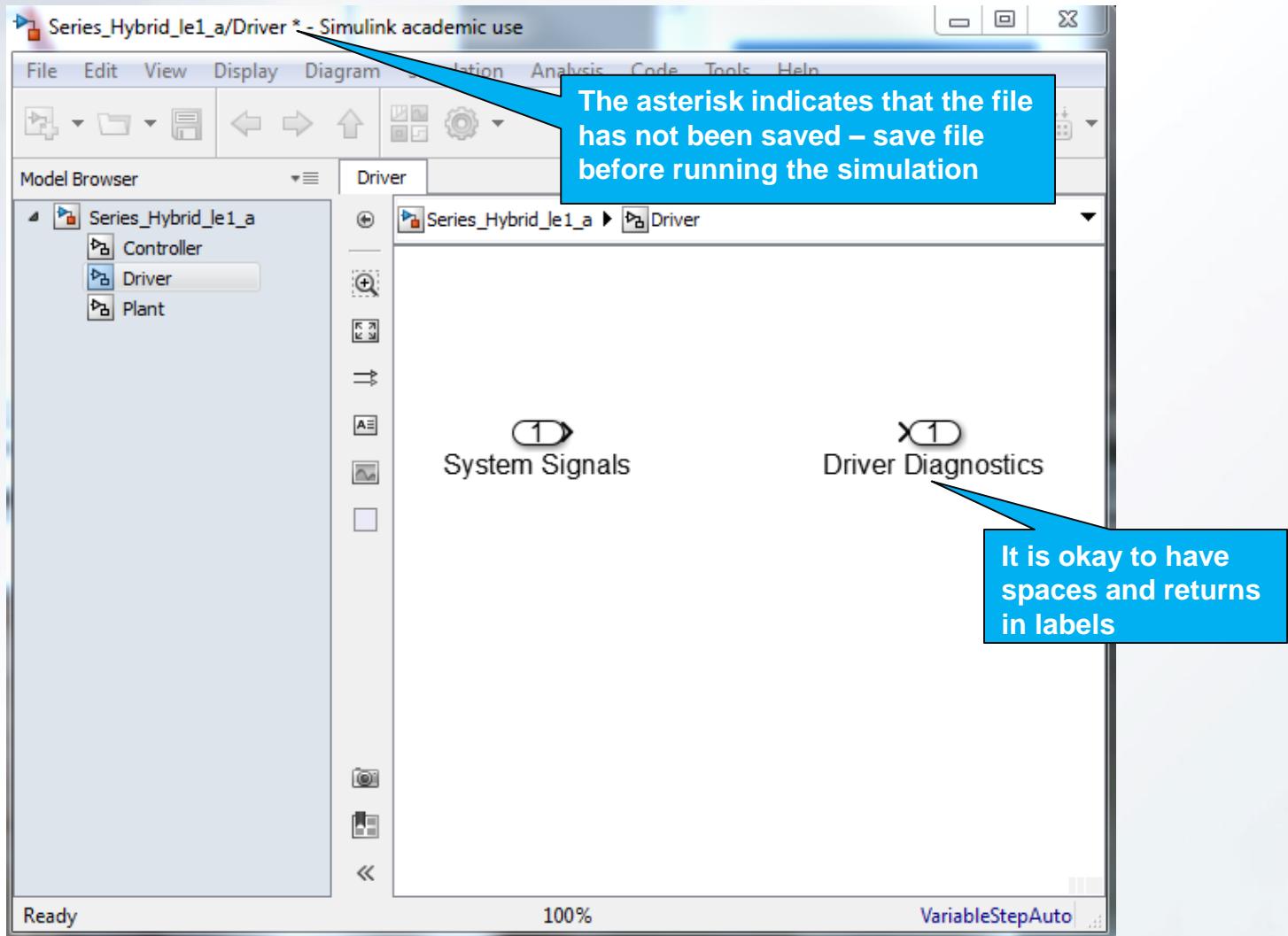


Driver Subsystem

- Open the Driver subsystem by:
 - clicking on Driver in Model Browser or
 - double click on the Driver block
- Delete the wire that connects the **In1** block to **out1** block
- Rename
 - In1 to System Signals
 - Out1 to Driver Diagnostics



Driver Subsystem



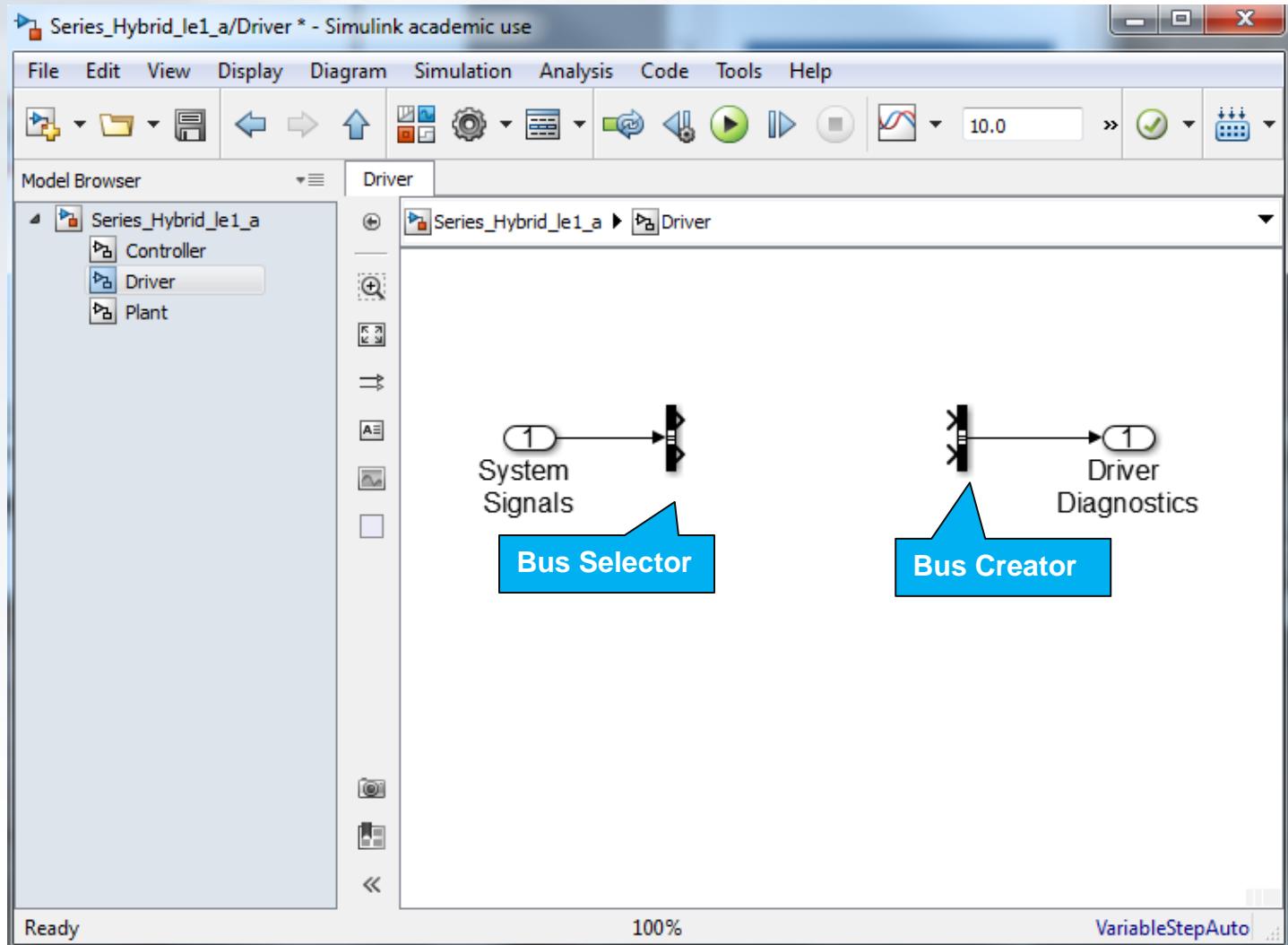


Driver Subsystem - CAN

- From
 - Simulink \ Commonly Used Blocks
- Drag in a **Bus Selector** and a **Bus Creator**
 - The Bus Creator puts signals on the bus
 - The Bus Selector extracts them from the bus
- Connect the System Signals input to the **Bus Selector**
- Connect the Driver Signals input to the **Bus Creator**



Driver Subsystem - CAN



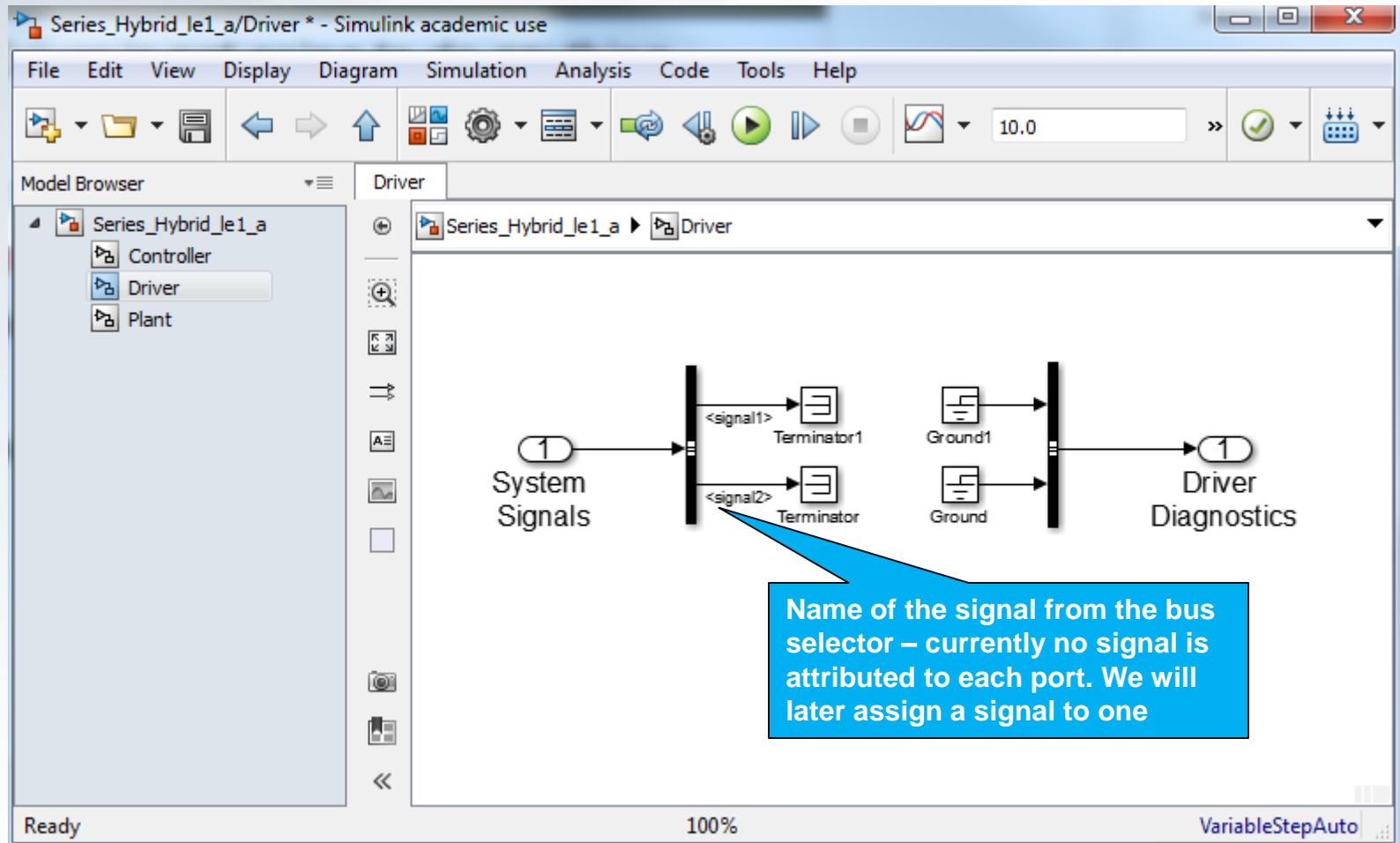


Driver Subsystem - CAN

- For now, the Driver is not going to do anything
- From
 - [Simulink \ Commonly Used Blocks](#)
- Drag in two **Terminators** and two **Grounds**
- Connect the two outputs of the Bus Selector to the **Terminators**
- Connect the two inputs of the Bus Selector to the **Grounds**



Driver Subsystem - CAN





Driver Subsystem - CAN

- The symbols for the ground and terminator are pretty obvious – there is no need for them to be labeled
- On one of the Terminators
 - Right Click
 - Select Format
 - Deselect Show Block Name
- We will do this for many of the blocks
- Hide the name of all terminator and ground blocks



Driver Subsystem - CAN

Series_Hybrid_le1_a/Driver * - Simulink academic use

File Edit View Display Diagram Simulation Analysis Code Tools

Model Browser Series_Hybrid_le1_a Driver

Driver

System Signals

<signal>

Terminator1

Ground1

<signal>

Terminator

Driver Diagnostics

Explore

- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V
- Comment Through Ctrl+Shift+Y
- Comment Out Ctrl+Shift+X
- Delete Del
- Find Referenced Variables
- Create Subsystem from Selection Ctrl+G
- Format
 - Rotate & Flip
 - Arrange
 - Mask
 - Library Link
 - Signals & Ports
 - Requirements Traceability
 - Fixed-Point Tool...
 - C/C++ Code
 - HDL Code
 - Polyspace
 - Block Parameters (Terminator)
 - Properties...
 - Help
- Font Style...
- Foreground Color
- Background Color
- Shadow
- Show Block Name

Ready 100% VariableStepAuto

```
graph LR; SS[System Signals] --> T1[Terminator1]; SS --> T2[Terminator]; T1 --> G1[Ground1]; T2 --> DD[Driver Diagnostics]
```

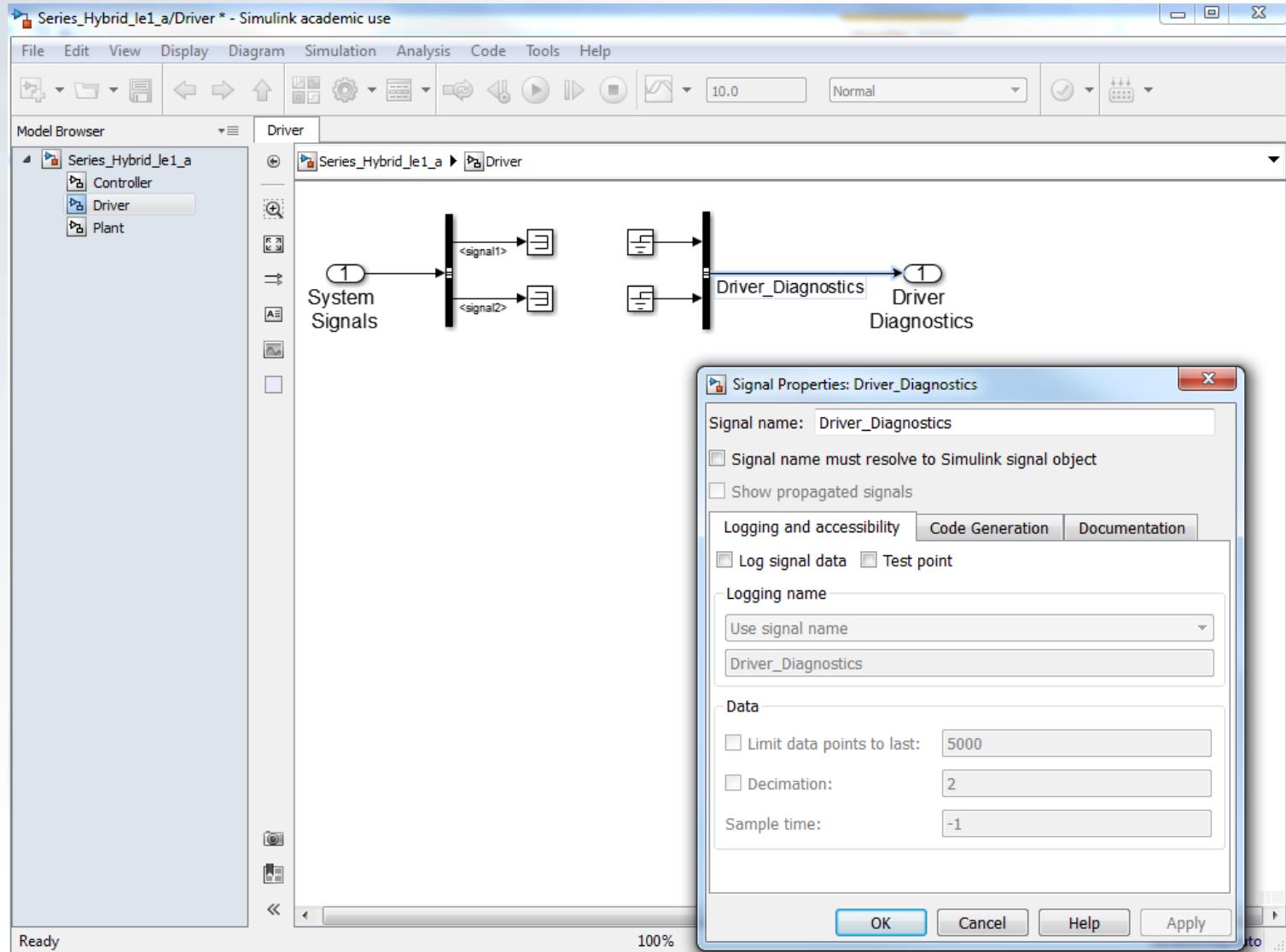


Driver Subsystem - CAN

- While no signals are being sent by the Driver, the Driver Diagnostics signal should be identified on the bus
- On the wire connecting the **Bus Creator** to the Driver Diagnostics output
 - Right Click
 - Select Properties
 - For the Signal name enter **Driver_Diagnostics**
 - Click OK



Driver Subsystem - CAN





Driver Subsystem - CAN

- Move the blocks around, enlarge if necessary, and make things look nice

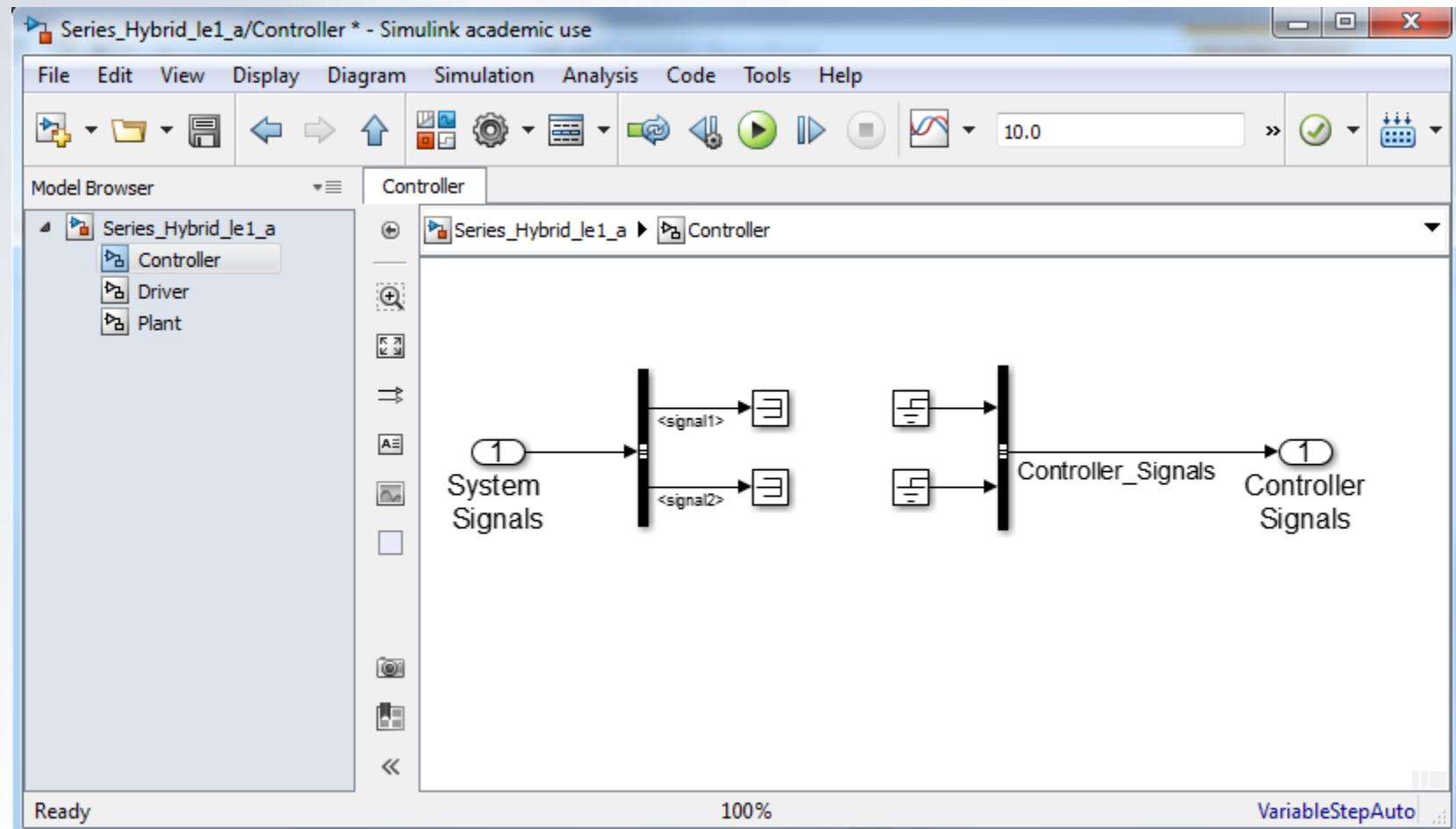


Controller Subsystem

- For now, the controller is not going to do anything
- Repeat everything done to the Driver subsystem for the Controller subsystem
 - Rename the Out1 to Controller Signals
 - Rename the CAN signal to Controller_Signals



Controller Subsystem



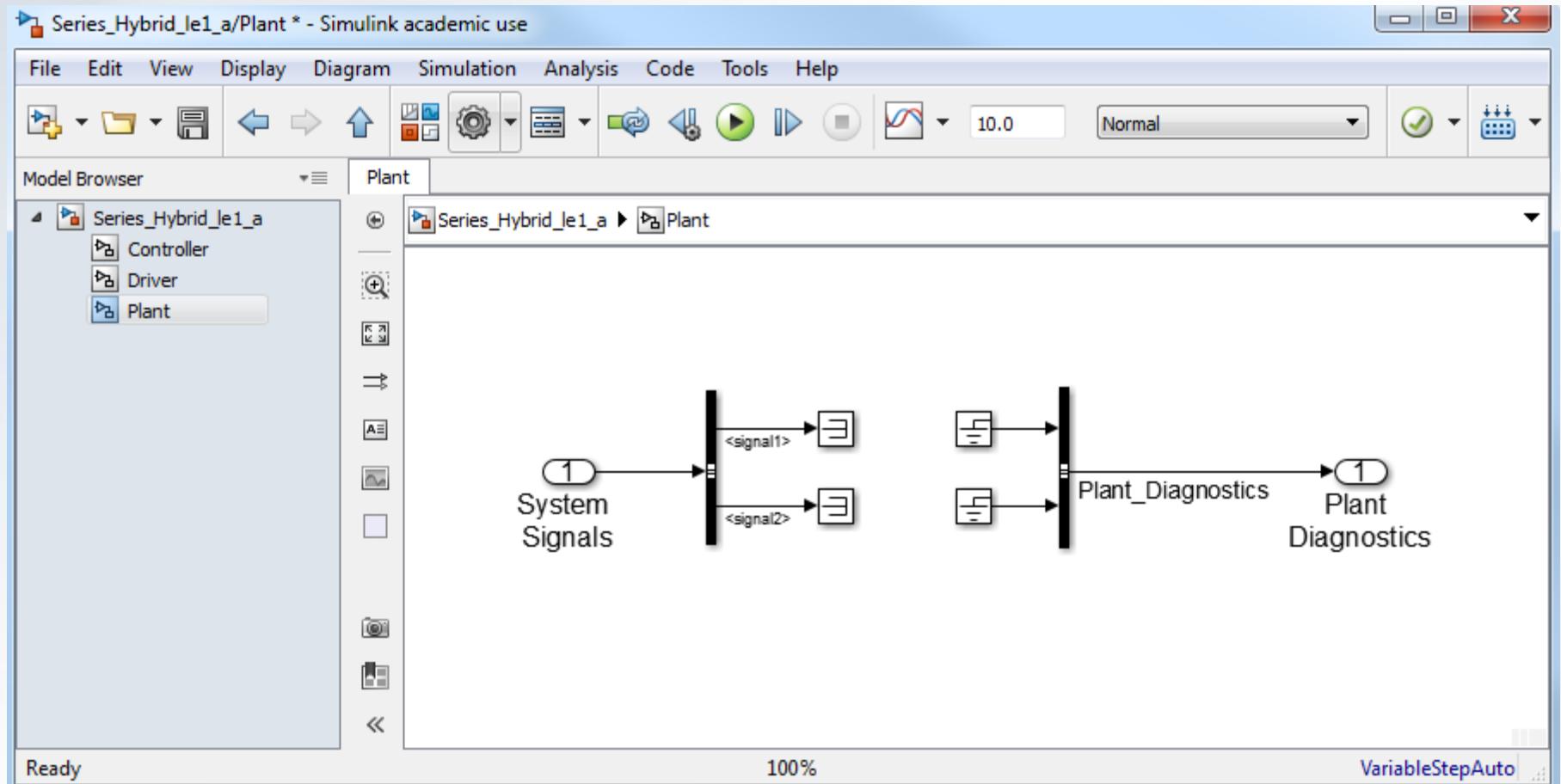


Plant Subsystem

- For now, the Plant is not going to do anything
- Repeat everything done to the Controller subsystem for the Plant subsystem
 - Rename the Out1 to Plant Diagnostics
 - Rename the CAN signal to Plant_Diagnostics



Plant Subsystem



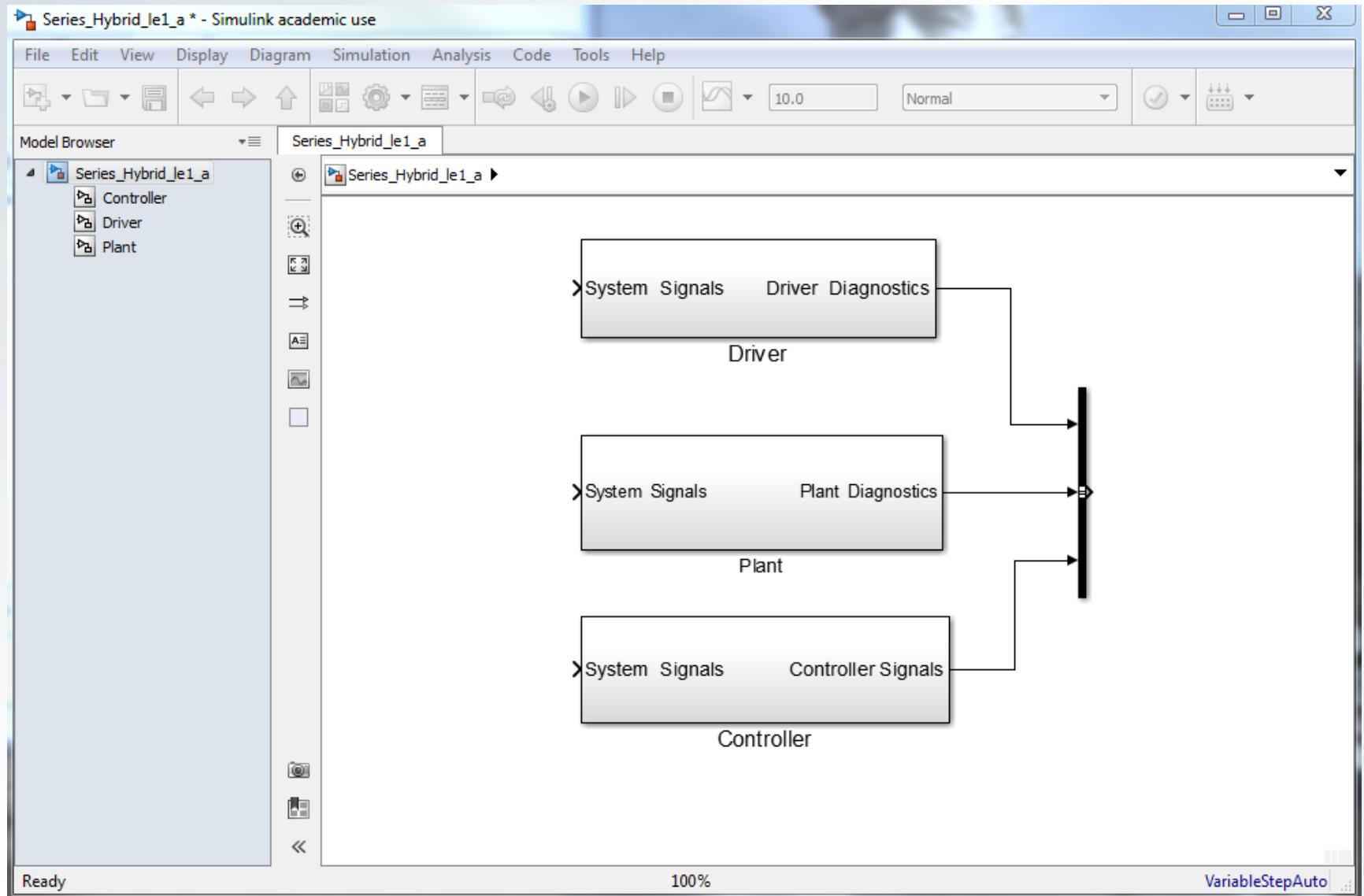


High Level

- Return to the top level of the model
- Increase the size of the subsystems to make them legible
- Drag in a **Bus Creator**
 - Double click of the Bus Creator
 - Change the number of inputs to 3
 - Click OK
- Connect each subsystem to the **Bus Creator**
- If you Double click on the **Bus Creator** after connecting the subsystem you can view the input signals



High Level



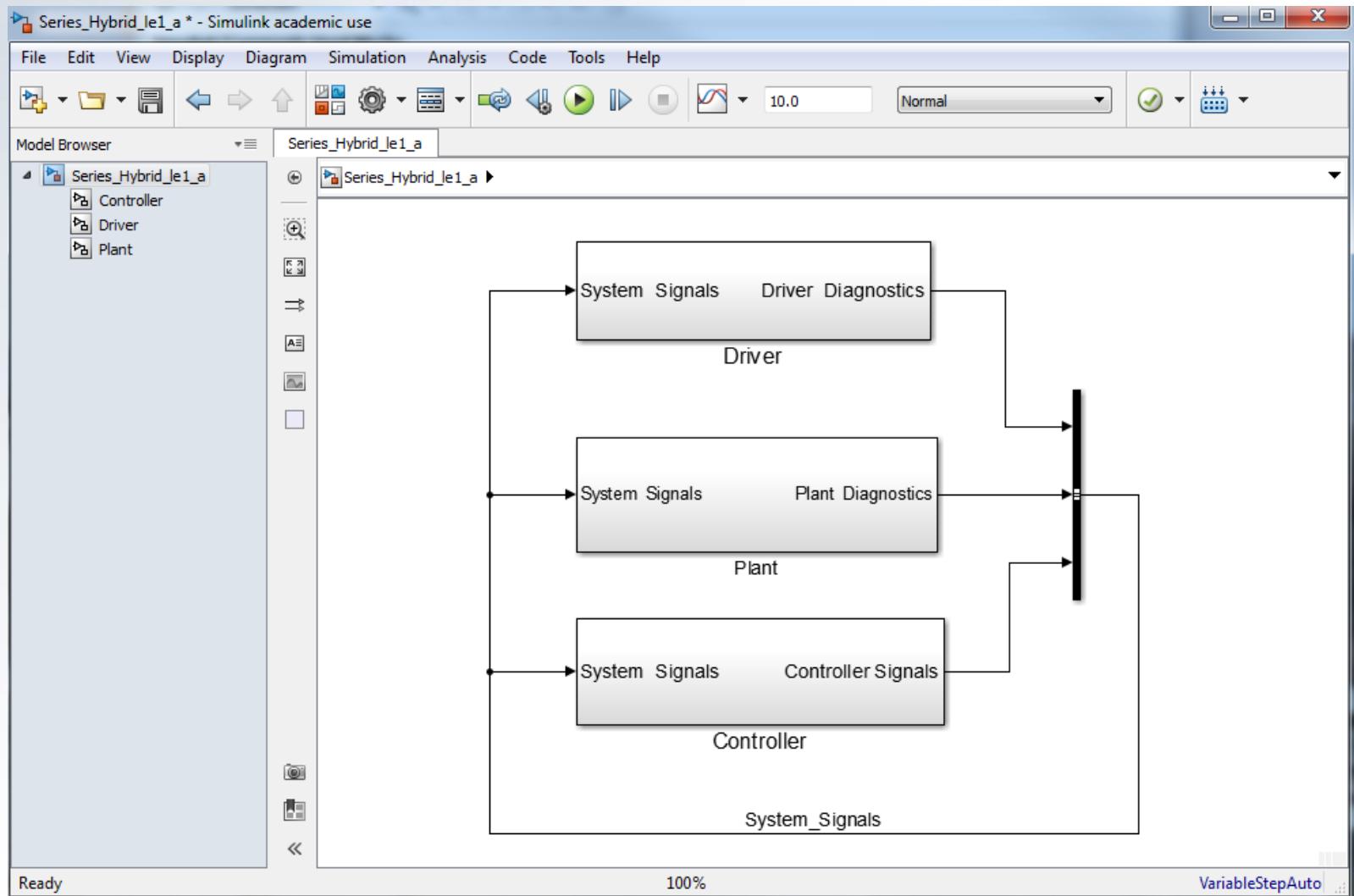


High Level

- Connect the output of the Bus Creator to each of the subsystem inputs
- Rename the output signal of the Bus Creator to System_Signals



High Level





High Level

- Congratulations! You have
 - Created a high level hierarchical system
 - Established communication between the three key subsystems
 - Gained experience with Simulink

A blue-toned photograph of the Georgia Tech Campanile, a tall brick tower with a spire and decorative stonework. The word "TECH" is prominently displayed on the side of the tower.

MBSD Lecture 2

Simple Vehicle Model



Lecture Goals

- Create the Vehicle subsystem
- Create the MGA subsystem with a preposterous motor model
- Observe and assess the results

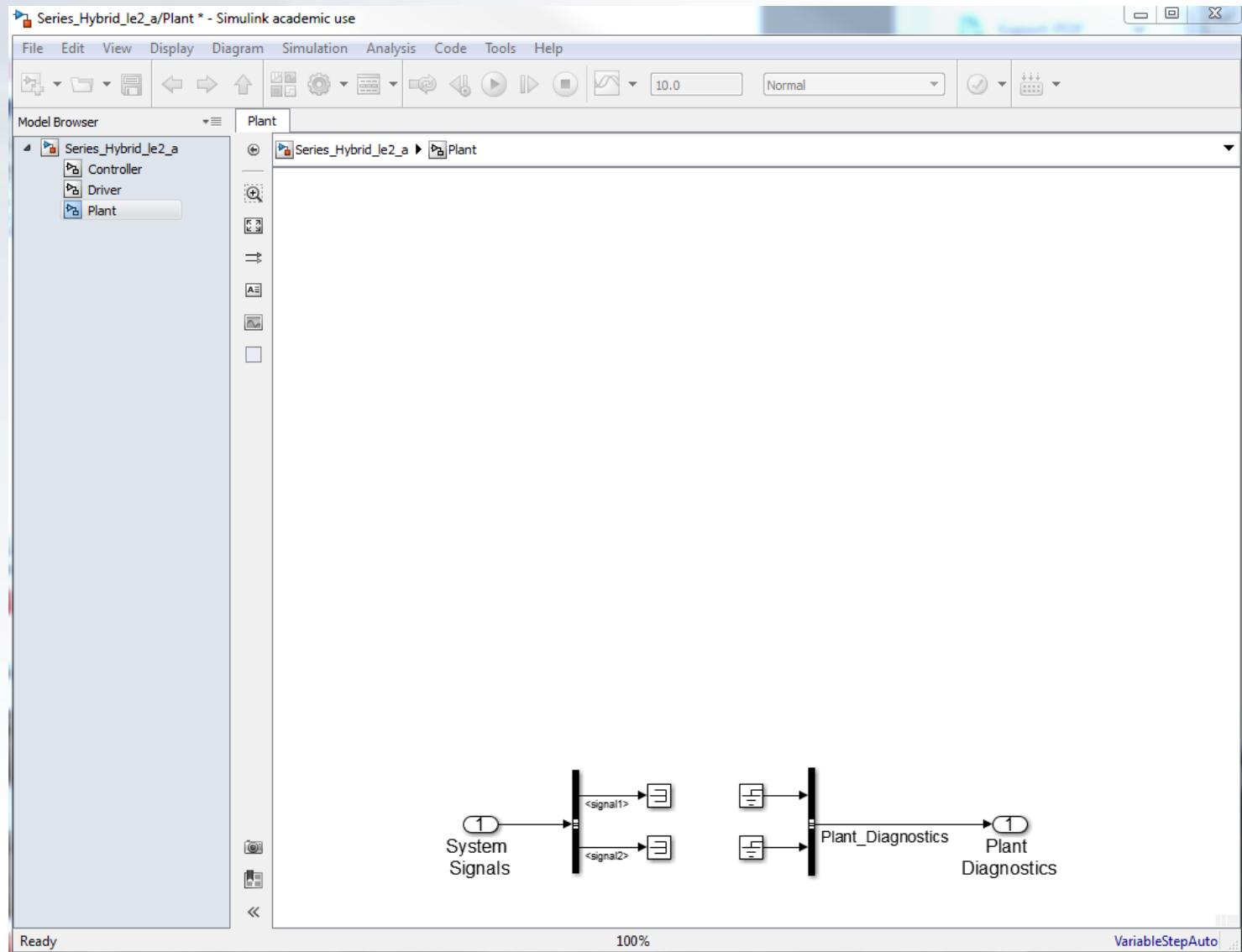


Simple Vehicle

- Save your previous model (Series_Hybrid_le1_a.slx) as Series_Hybrid_le2_a
- In the Model Browser window, click on the Plant subsystem
- Move the buses to the bottom of the window



Simple Vehicle



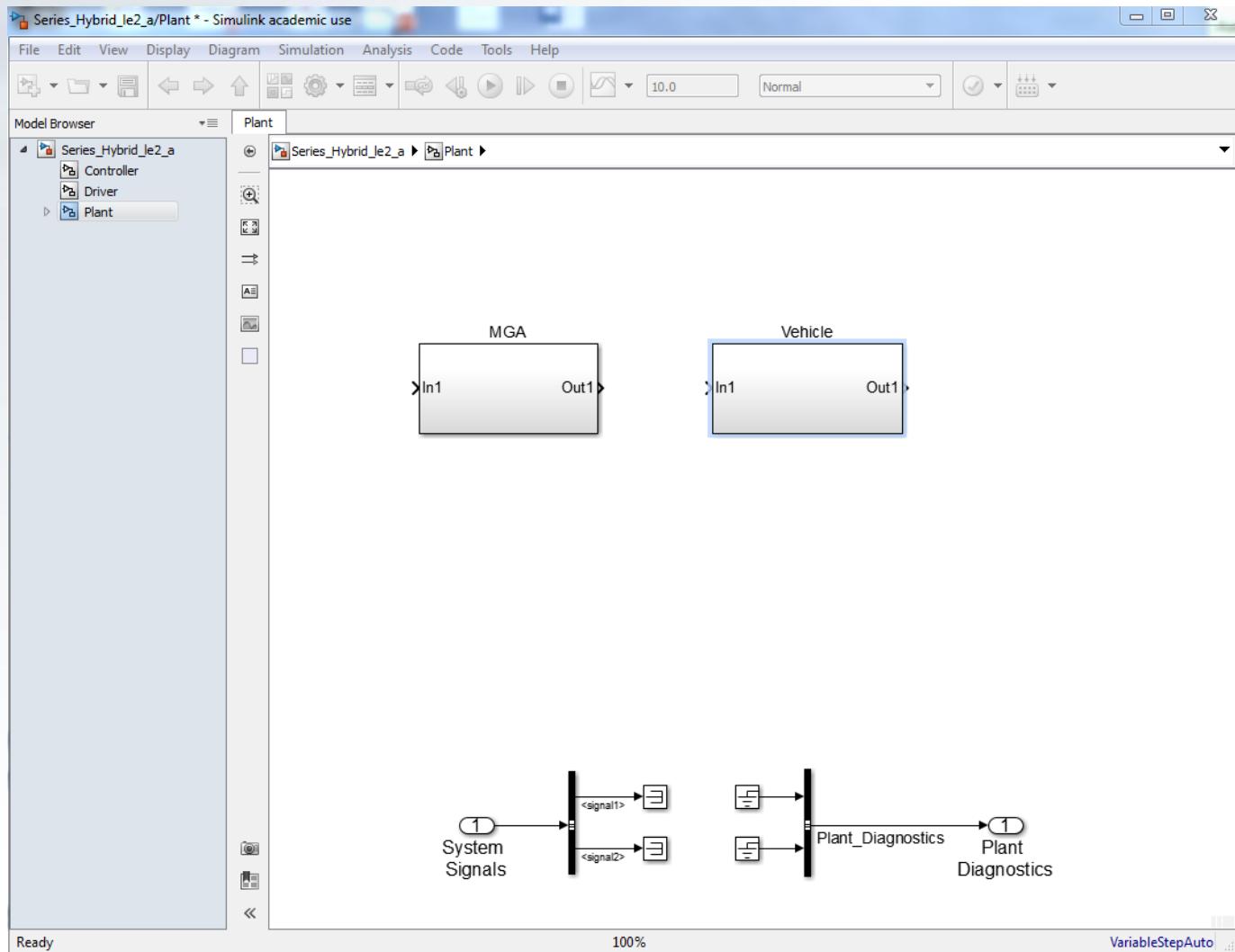


Simple Vehicle

- Drag two **Subsystems** into the Plant window
- Rename them
 - Vehicle
 - MGA
- Right click on the Vehicle subsystem
 - Select Rotate & Flip
 - Flip Block Name
- Repeat for the MGA subsystem



Simple Vehicle



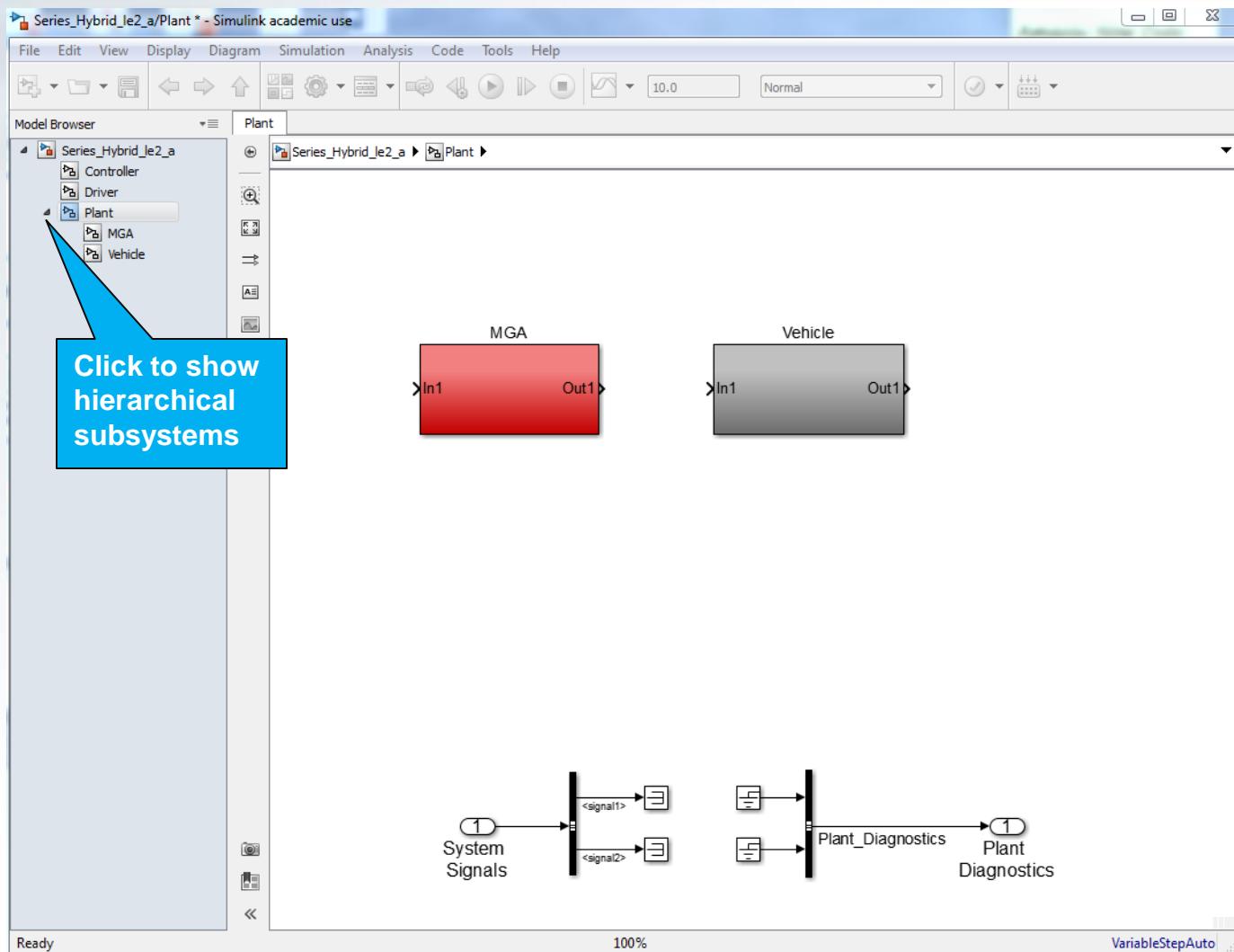


Simple Vehicle

- Let's also color code our subsystems to make things easy to identify
 - Right click on the subsystem
 - Format/Background Color
- Make the vehicle gray and MGA red



Simple Vehicle





Vehicle Subsystem

- The Vehicle subsystem will mechanically receive torque via a shaft from MGA
- We will skip mechanical brakes and use exclusively regenerative braking as applied by MGA
- The Vehicle subsystem will contain
 - Posi-trac rear axle
 - Rear tires
 - Vehicle solver
 - Inertias for the driveshaft and axle half shafts



Vehicle Subsystem

- Delete the **In1** and **Out1** blocks as well as the connecting wire
- Copy the **In1**, **Out1**, **Bus**, **Terminator**, and **Ground** blocks from the Plant subsystem
- Rename Plant to Vehicle
- From
 - Simscape / Utilities
- Drag in a **Connection Port** and rename it Vehicle Mechanical Port

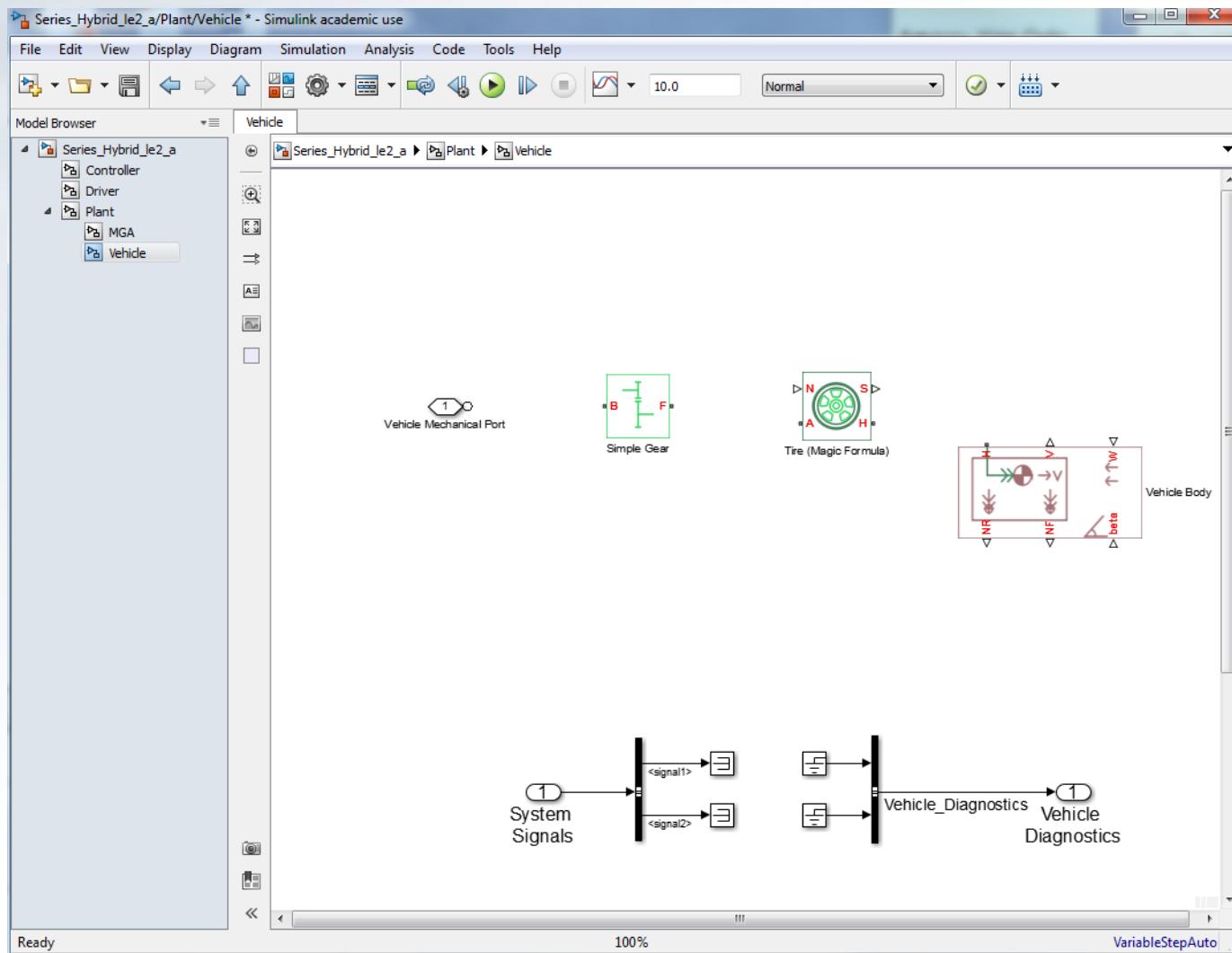


Vehicle Subsystem

- From
 - Simscape / Driveline / Tires & Vehicles
- Drag in the **Vehicle Body** and one **Tire(Magic Formula)** blocks
- From
 - Simscape / Driveline / Gears
- Drag in a **Simple Gear** block



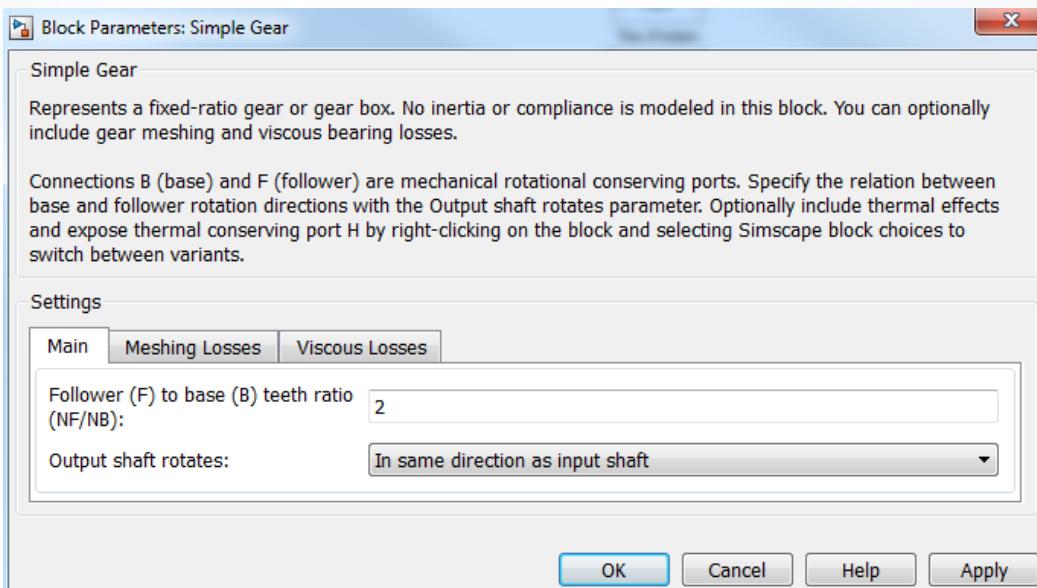
Vehicle Subsystem





Vehicle Subsystem

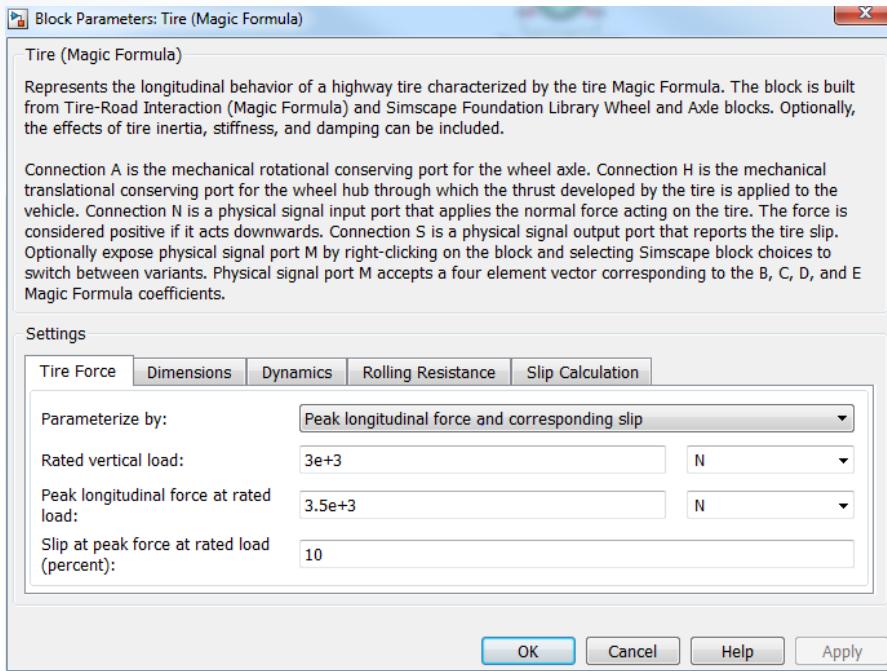
- Double click on the **Simple Gear**
- We can specify the gear ratio and relative rotational directions
 - Change the output shaft rotation to the same direction as input shaft
 - Leave the ratio as 2





Vehicle Subsystem

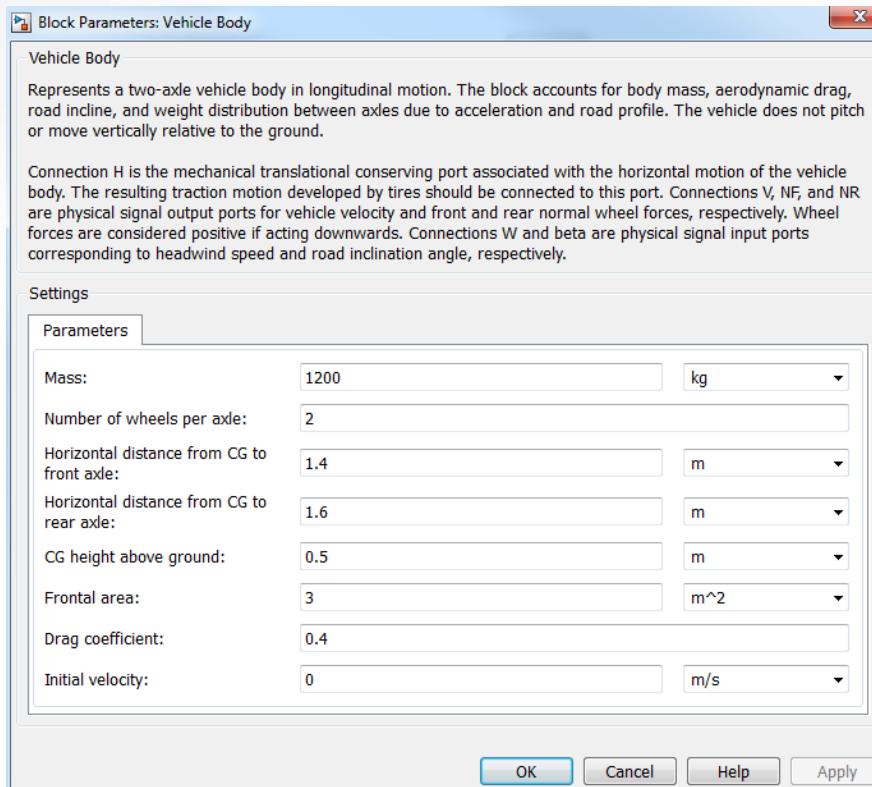
- Double click on the **Tire** block
 - This block converts the applied torque to force in the x-direction
 - It also determined whether or not the tire slips
 - Keep the default values





Vehicle Subsystem

- Double click on the **Vehicle Body** block
 - This block solves linear momentum (with drag and pitch) from the tire force
 - Keep the default values



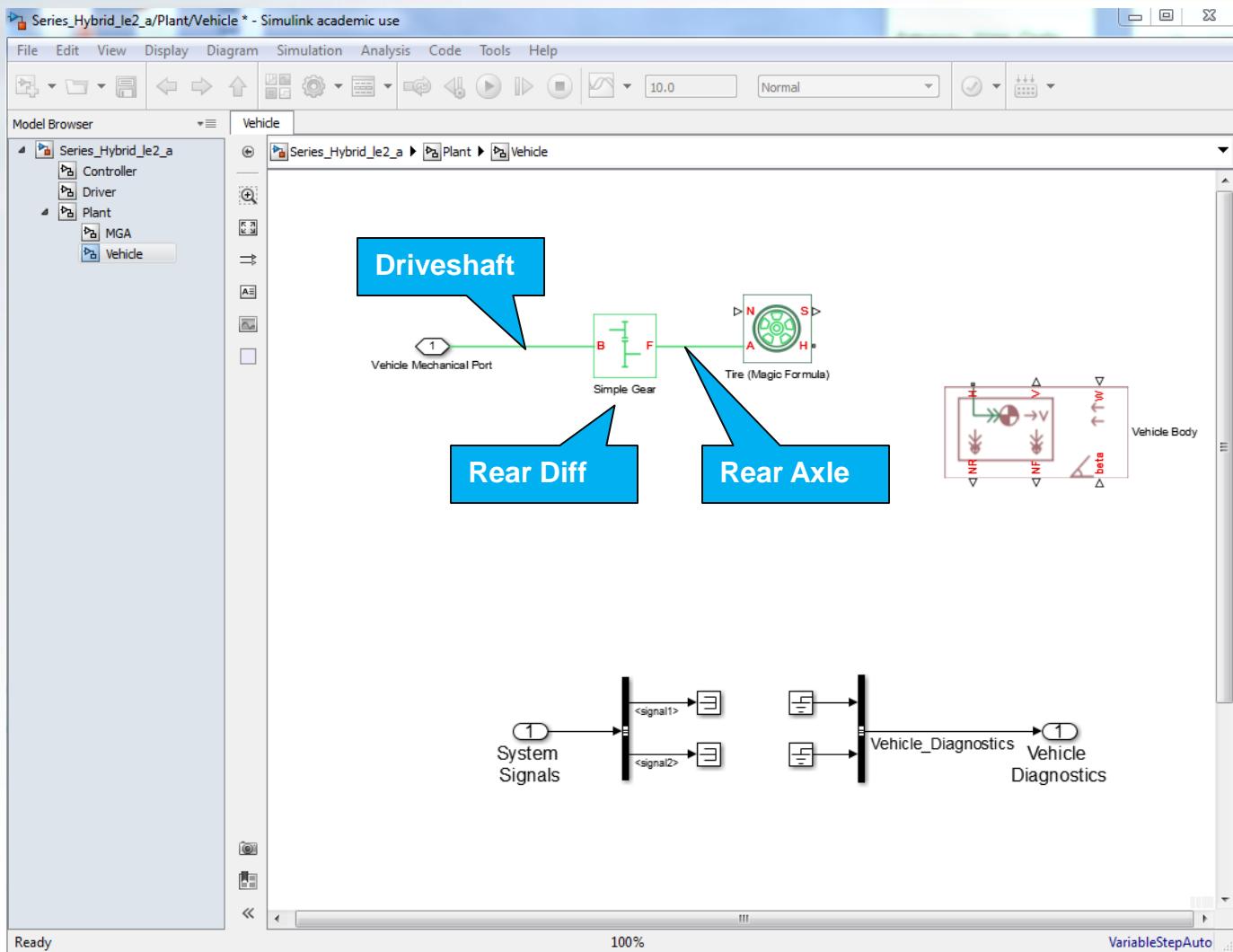


Vehicle Subsystem

- Connect the Vehicle Mechanical Port to the Base side of the **Simple Gear**
- Connect the Follower side of the **Simple Gear** to the A port on the **Tire**
- We now have a model of the driveshaft, rear differential, and axle shaft



Vehicle Subsystem





Vehicle Subsystem

- Note the difference in between the Driveline connection ports and the Simulink connection ports
- They will not connect to each other
- Driveline blocks are used to solve the Conservation of Angular Momentum principle during each time step

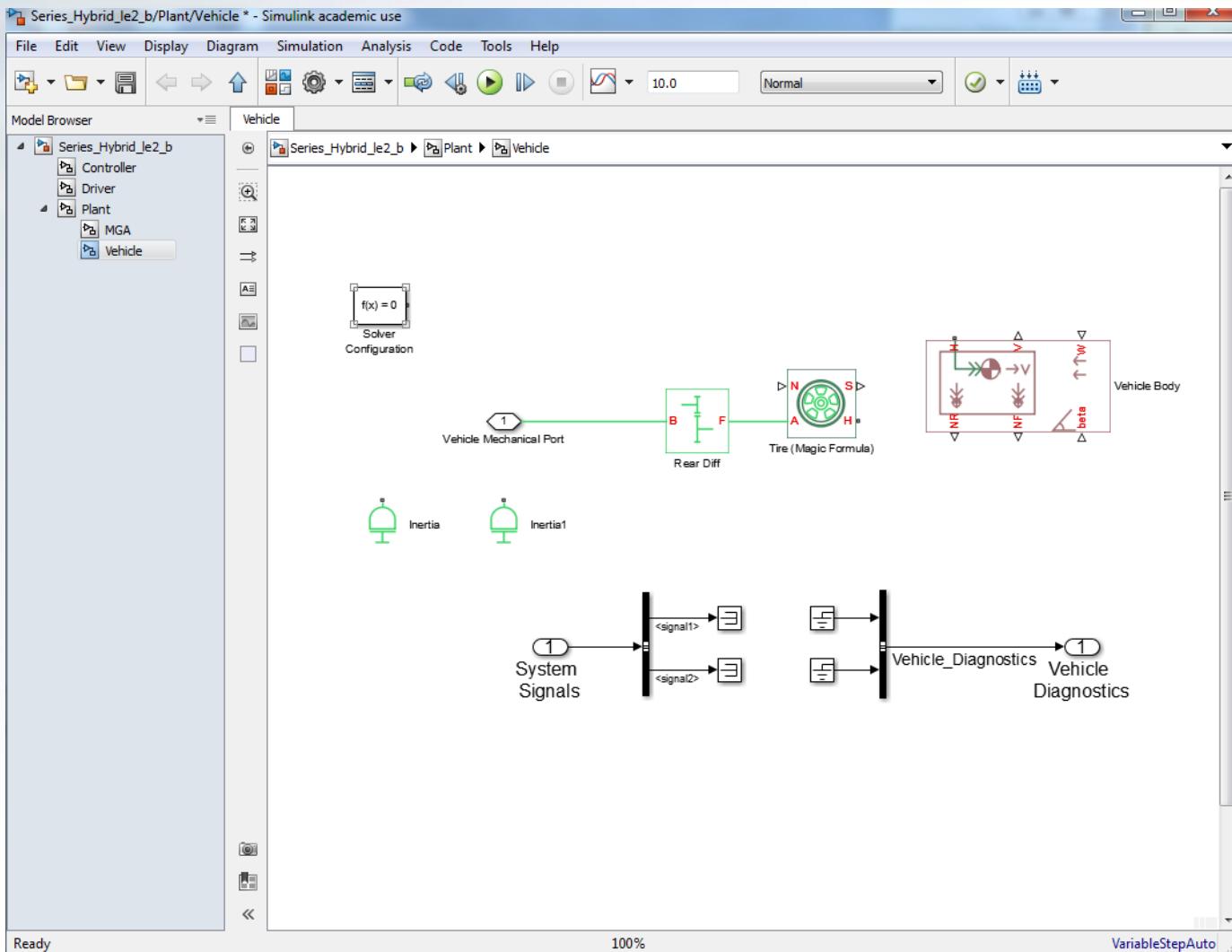


Vehicle Subsystem

- Every shaft must have a mass moment of inertia
- Also, a special solver must be used
- From
 - Simscape / Foundation Library / Mechanical / Rotational Elements
- Drag in two **Inertia** blocks
- From
 - Simscape / Utilities
- Drag in a **Solver Configuration** block



Vehicle Subsystem



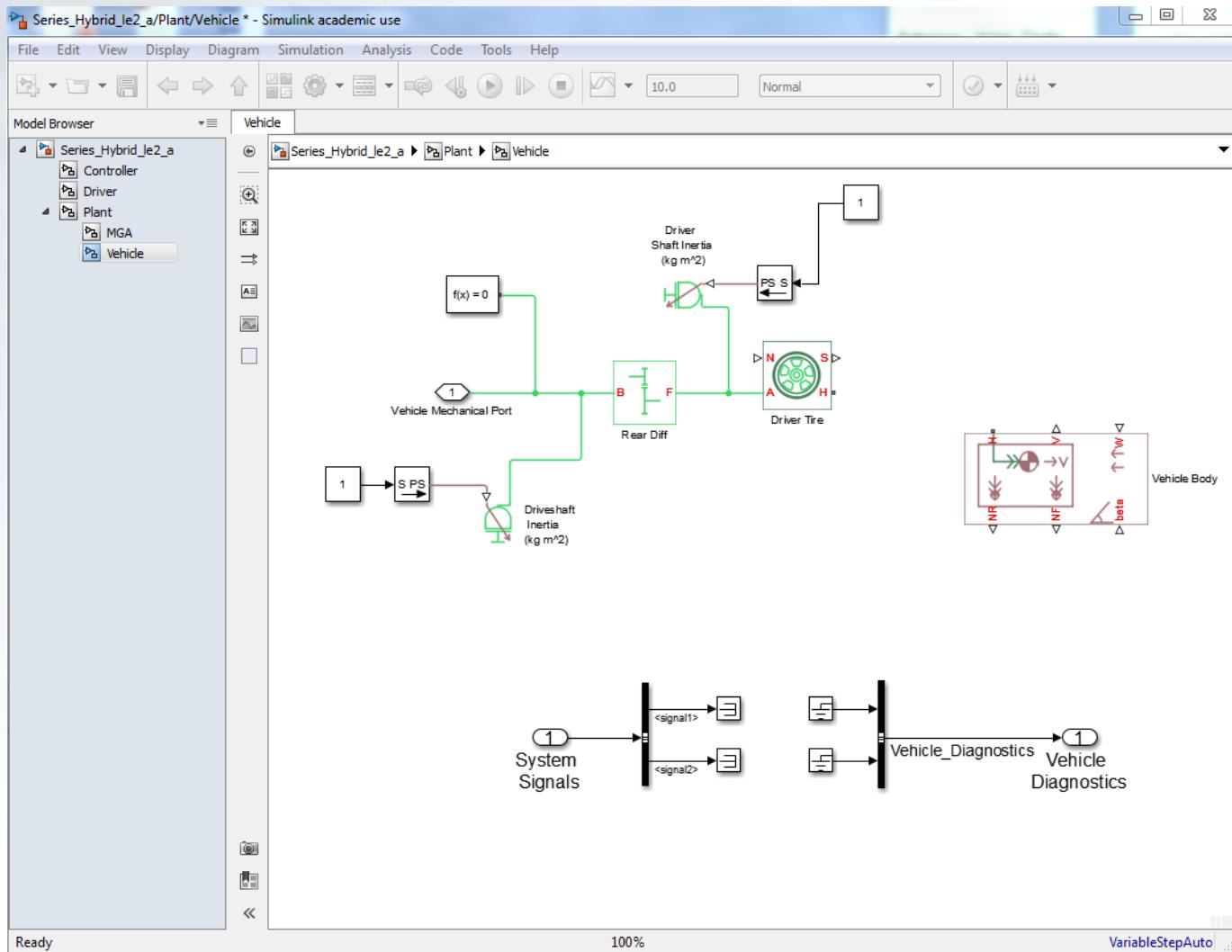


Vehicle Subsystem

- Connect the **Inertias** to the driveshaft and rear axle and Rename them:
 - Driveshaft Inertia (kg m^2)
 - Driver Halfshaft Inertia (kg m^2)
- Connect the **Solver Configuration** to the driveshaft
- Hide its name
- Rename the Tire to Driver Tire
- Rename the Simple Gear to Rear Diff



Vehicle Subsystem



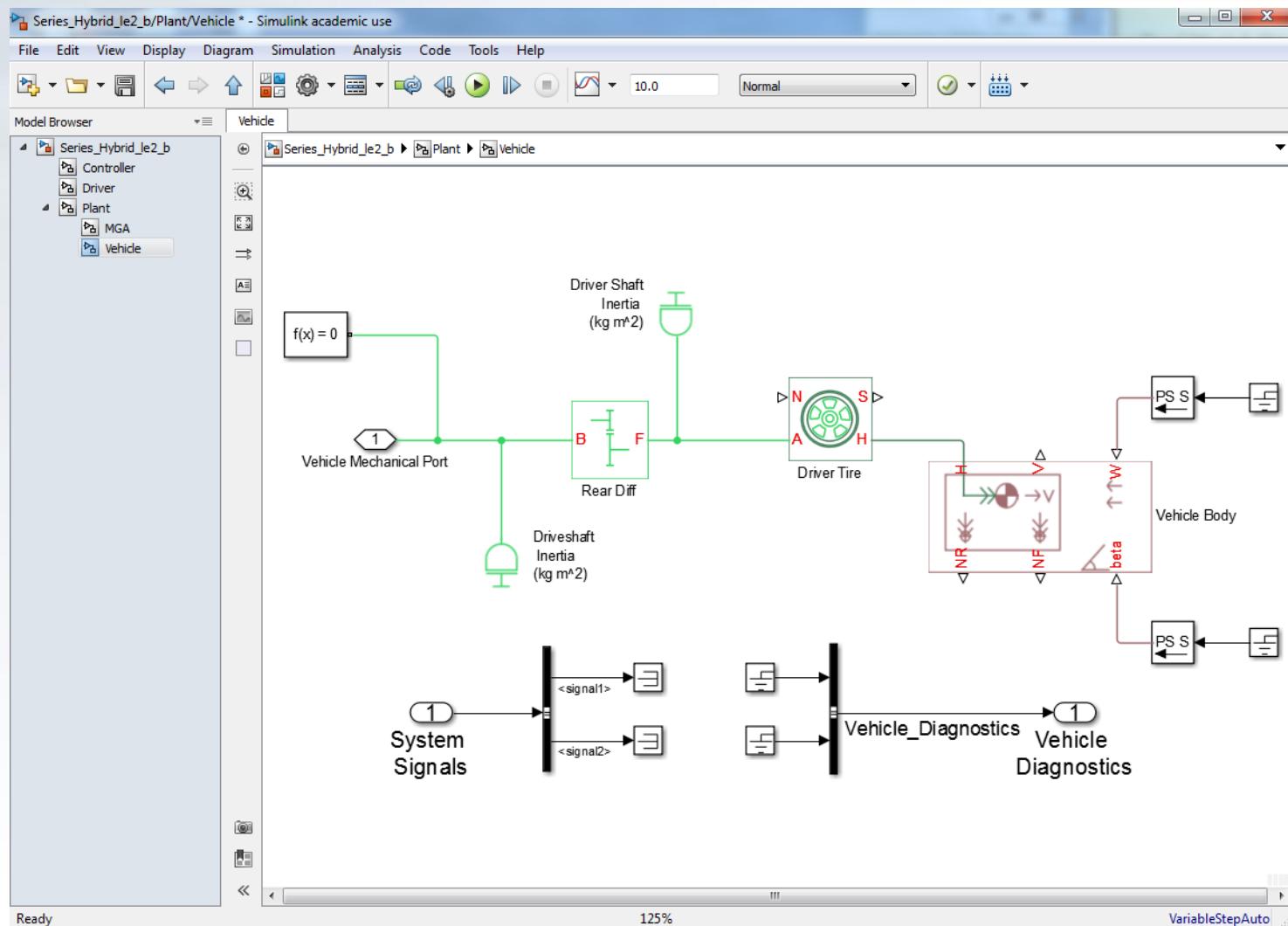


Vehicle Subsystem

- Rename and save your model as Series_Hybrid_le2_b
- Now we can connect the H port of the **Drive Tire** block to the H port of the **Vehicle Body** block
- From
 - Simscape / Utilities
- Drag in two **Simulink-PS Converter** blocks
- These blocks transfer Simulink signals to physical signals
- Drag in two **Ground** blocks
- Hide their names
- Connect the **Ground** blocks to the **Simulink-PS Converters**
- Connect the **Simulink-PS Converters** to the beta and W ports of the **Vehicle Body** block



Vehicle Subsystem



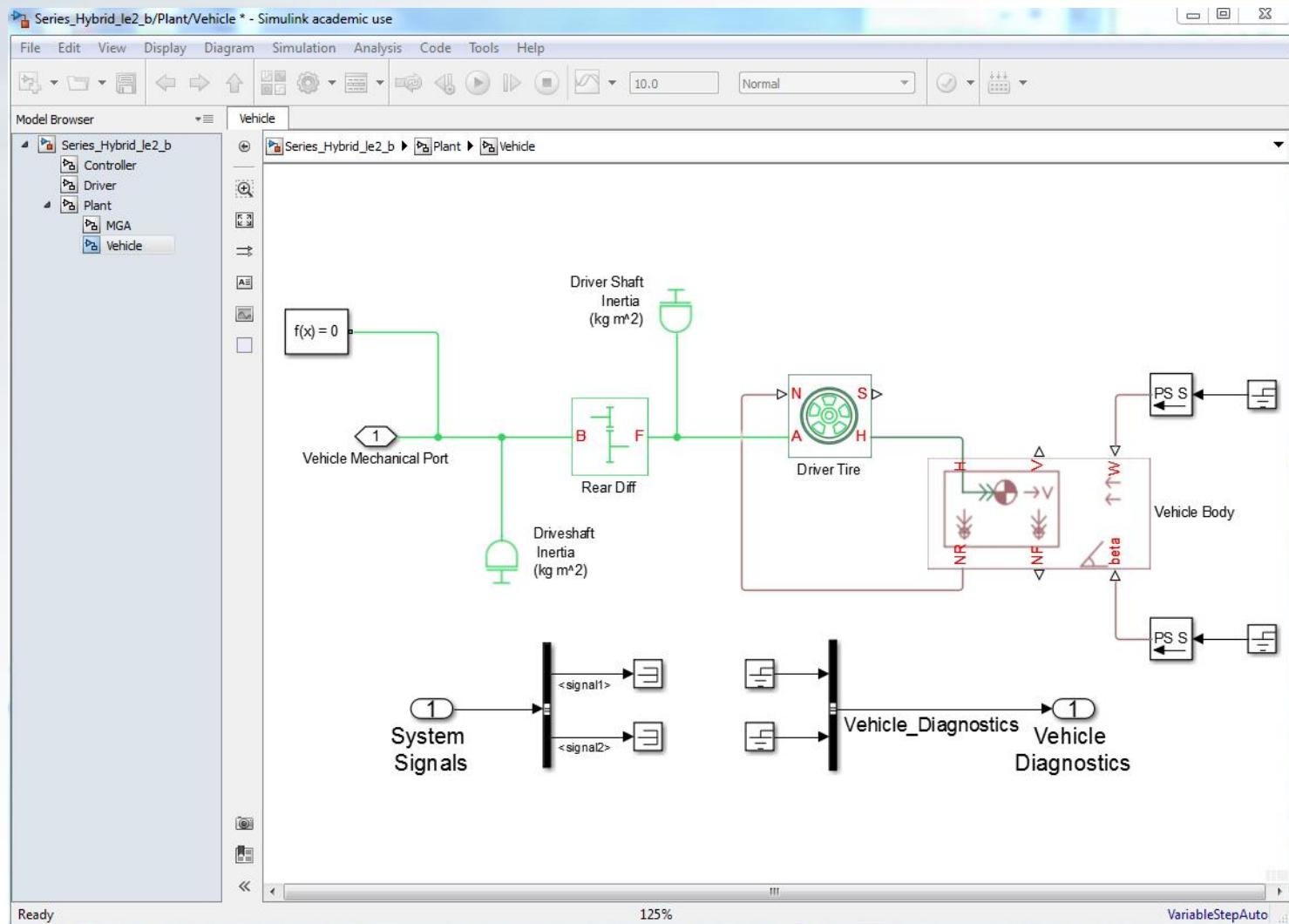


Vehicle Subsystem

- The NR port of the **Vehicle Body** block is a physical signal reporting the normal force applied on the rear wheels
- Connect the NR port of the **Vehicle Body** block to the N port of the **Driver Tire** block
- Rearrange the block and wire to make model look good
- S port of the Drive Tire block is a physical signal reporting tire slip



Vehicle Subsystem





Vehicle Subsystem

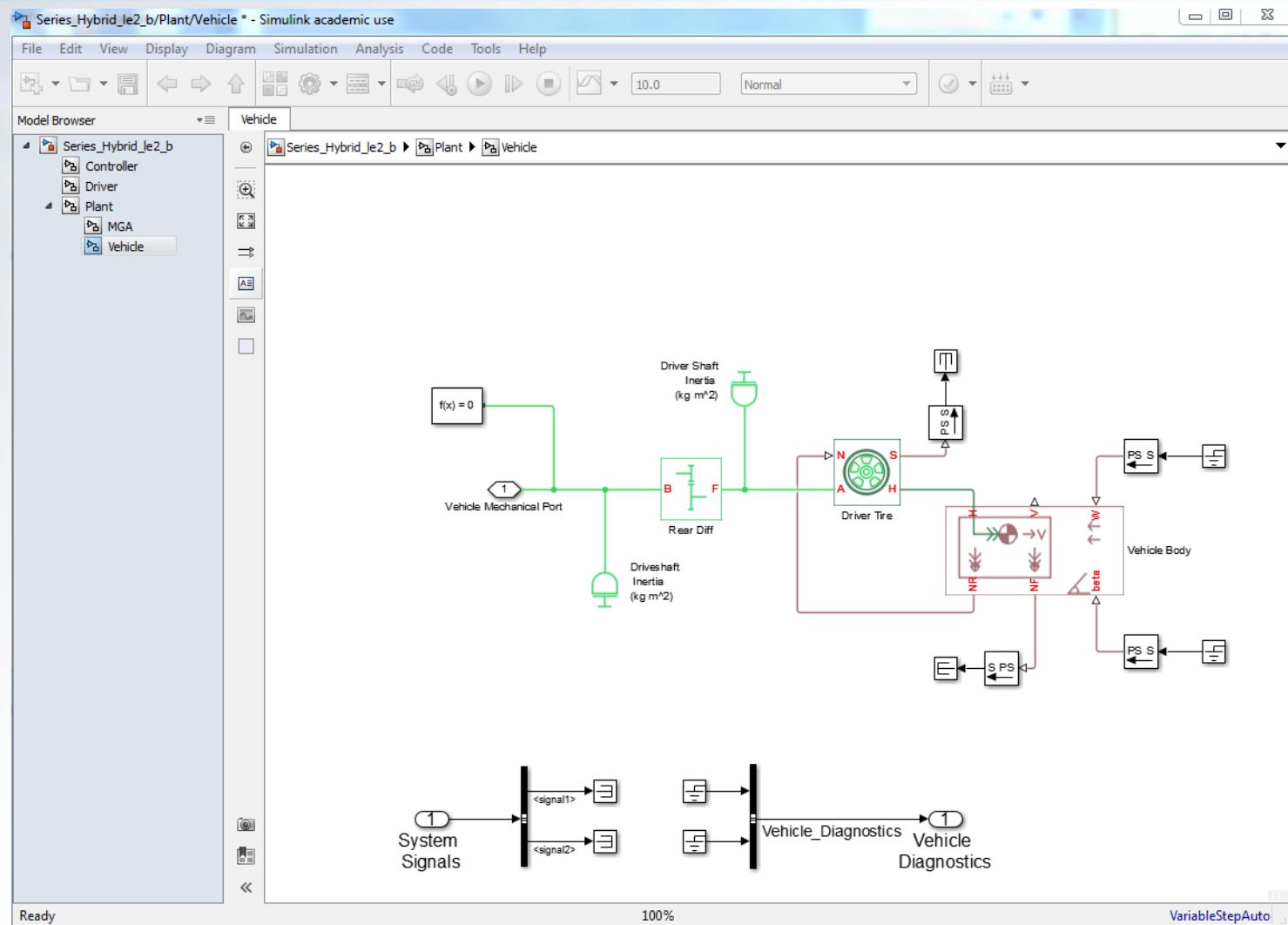
- S port of the **Drive Tire** block is a physical signal reporting tire slip
- NF port of the **Vehicle Body** block reports the normal load on the front tires
- We are not interested in tire slip or the normal force on the front tire (this is a rear wheel vehicle)
- From
 - Simscape / Utilities
- Grab two **PS-Simulink Converter** blocks (opposite of **Simulink-PS Converter**)
- Grab two **Terminator** blocks
- Hide name of **PS-Simulink Converter** and **Terminator** both blocks
- Connect the **PS-Simulink Converter** blocks to the **Terminators**
- Connect the S port of the Drive Tire block and NF port of the Vehicle Body block to the **PS-Simulink Converter** blocks
- Rearrange the block to make the model look good and compact



Vehicle Subsystem

- It is preferred for the wires not to cross as much as possible. You should place your blocks in position to avoid the wire crossing as much as possible

Vehicle Subsystem





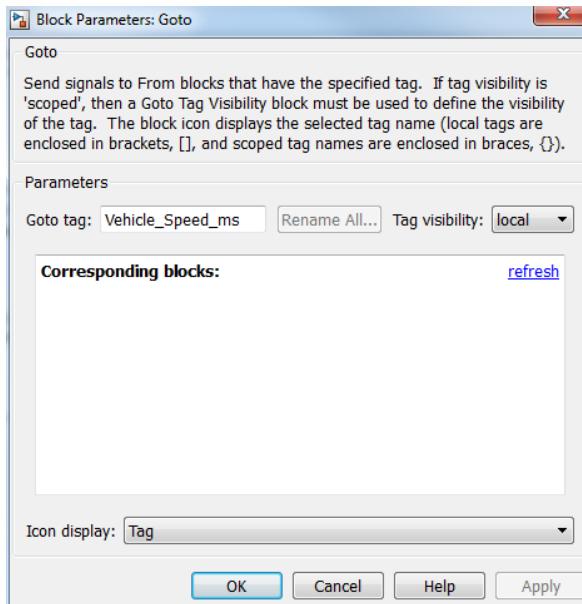
Vehicle Subsystem

- We need to send the vehicle velocity signal (from the V port of the **Vehicle Body** block) to other parts of the model
- Grab a **PS-Simulink Converter** block and connect the V port of the **Vehicle Body** block to the **PS-Simulink Converter**
- From
 - **Simulink / Signal Routing**
- Grab a **Goto** block
- Connect the **PS-Simulink Converter** to the **Goto** block

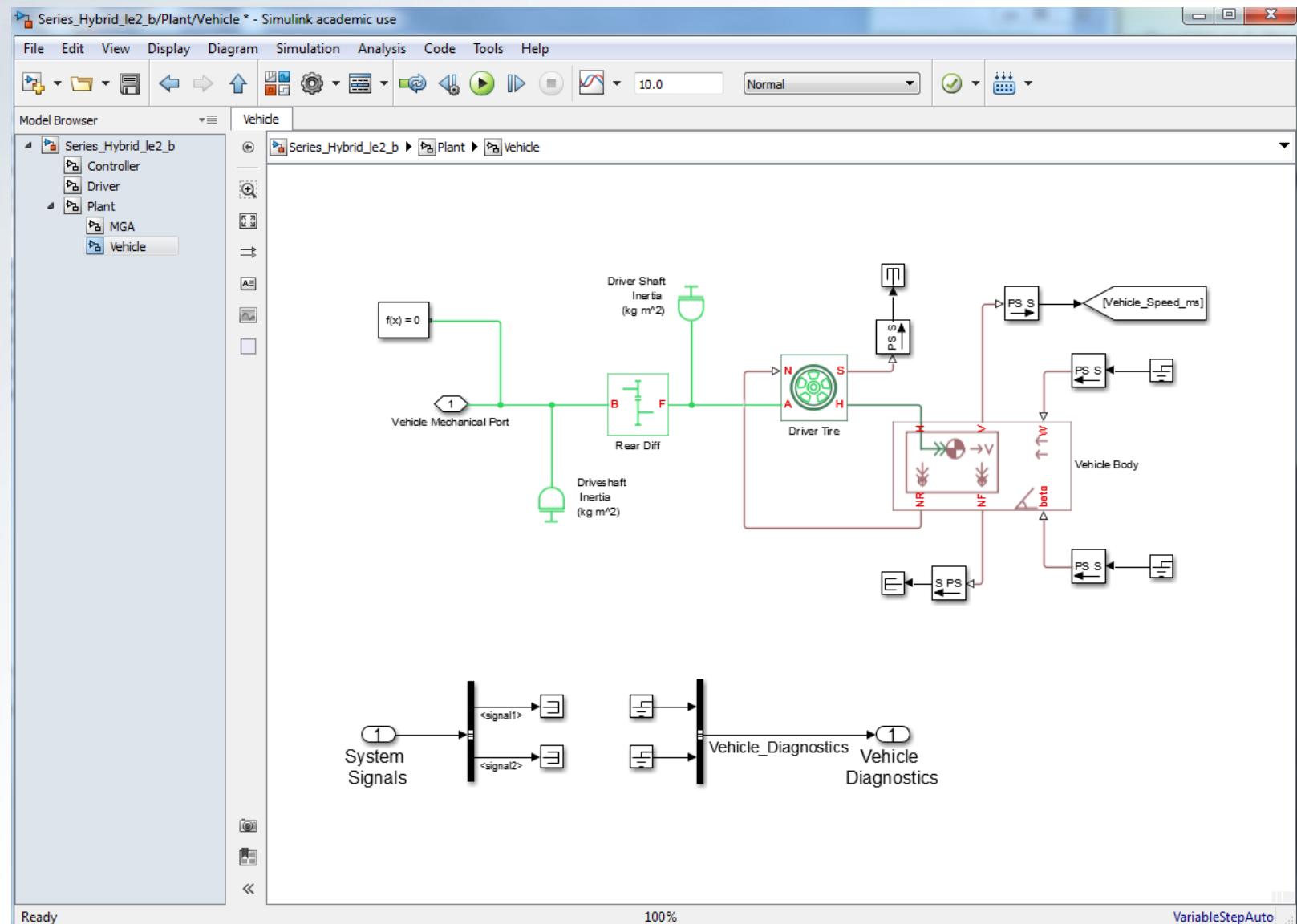


Vehicle Subsystem

- Double click on the **Goto** block and change the tag to `Vehicle_Speed_ms`
- Make sure the visibility is set to local
- Click OK
- Drag the sides of the **Goto** block to make the name visible
- Hide the name of **Goto** and **PS-Simulink Converter** blocks



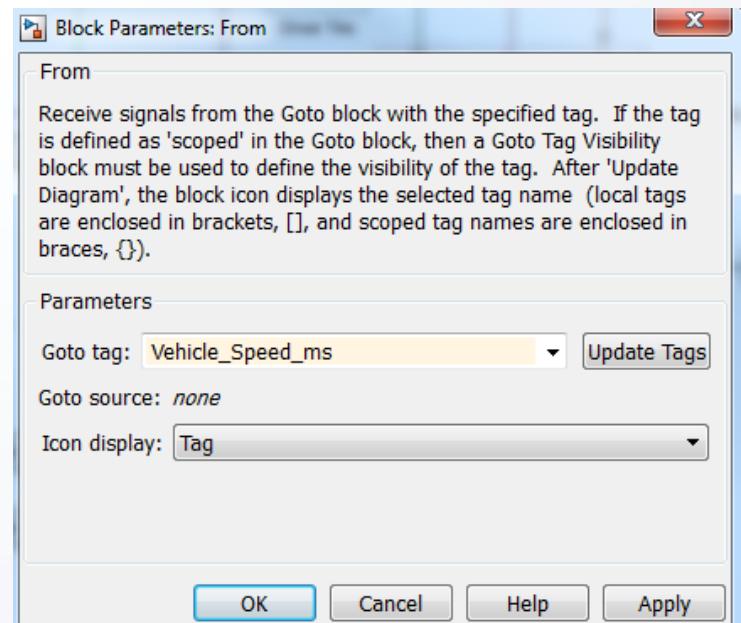
Vehicle Subsystem





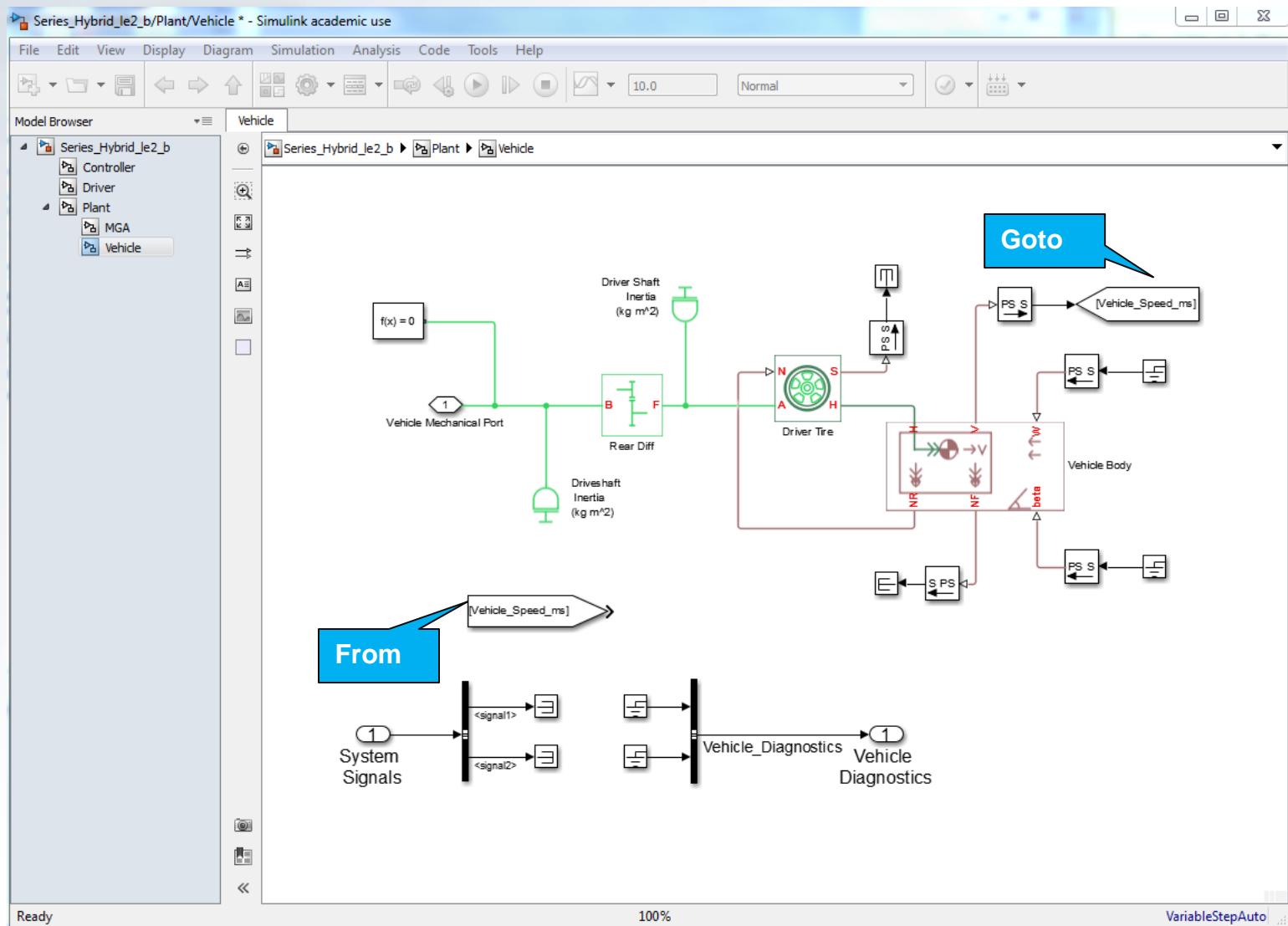
Vehicle Subsystem

- From
 - Simulink / Signal Routing
- Grab a **From** block
- Double click on the **From** block
- Open the Goto tag bar and select Vehicle_Speed_ms
 - Note: if you don't see Vehicle_Speed_ms, click on Update Tags and try again
- Click OK
- Hide the name of the **From** block





Vehicle Subsystem





Vehicle Subsystem

- Congratulations – you have a model of a vehicle that will (hopefully) move when subjected to a torque
- From a vehicle diagnostics perspective, we are interested in two things
 - Speed (mph)
 - Distance Traveled (miles)
- Move the Vehicle_Speed_ms **From** block down near the Vehicle Diagnostics Bus
- From
 - Simulink / Commonly Used Blocks
- Drag in two Gain **blocks** and one **Integrator** block
- Hide the name of the **Integrator** block



Vehicle Subsystem

- Congratulations – you have a model of a vehicle that will (hopefully) move when subjected to a torque
- From a vehicle diagnostics perspective, we are interested in two things
 - Speed (mph)
 - Distance Traveled (miles)
- Move the Vehicle_Speed_ms **From** block down near the Vehicle Diagnostics Bus
- From
 - Simulink / Commonly Used Blocks
- Drag in two Gain **blocks** and one **Integrator** block
- Hide the name of the **Integrator** block



Vehicle Subsystem

- Set the first **Gain** to 2.2374 (m/s to mph)
- Set the second **Gain** to 6.214 e-4 (m/s to miles/s)
- Name them accordingly
- Connect the output of the **From** block to the input of the **Gain** blocks
- Connect the second **Gain** block to the **Integrator** block
- Delete the **Grounds** going into the Bus
- Connect the two signals
- Name them: (Double click on the wires)
 - **Vehicle_Speed_mph**
 - **Vehicle_Distance_miles**

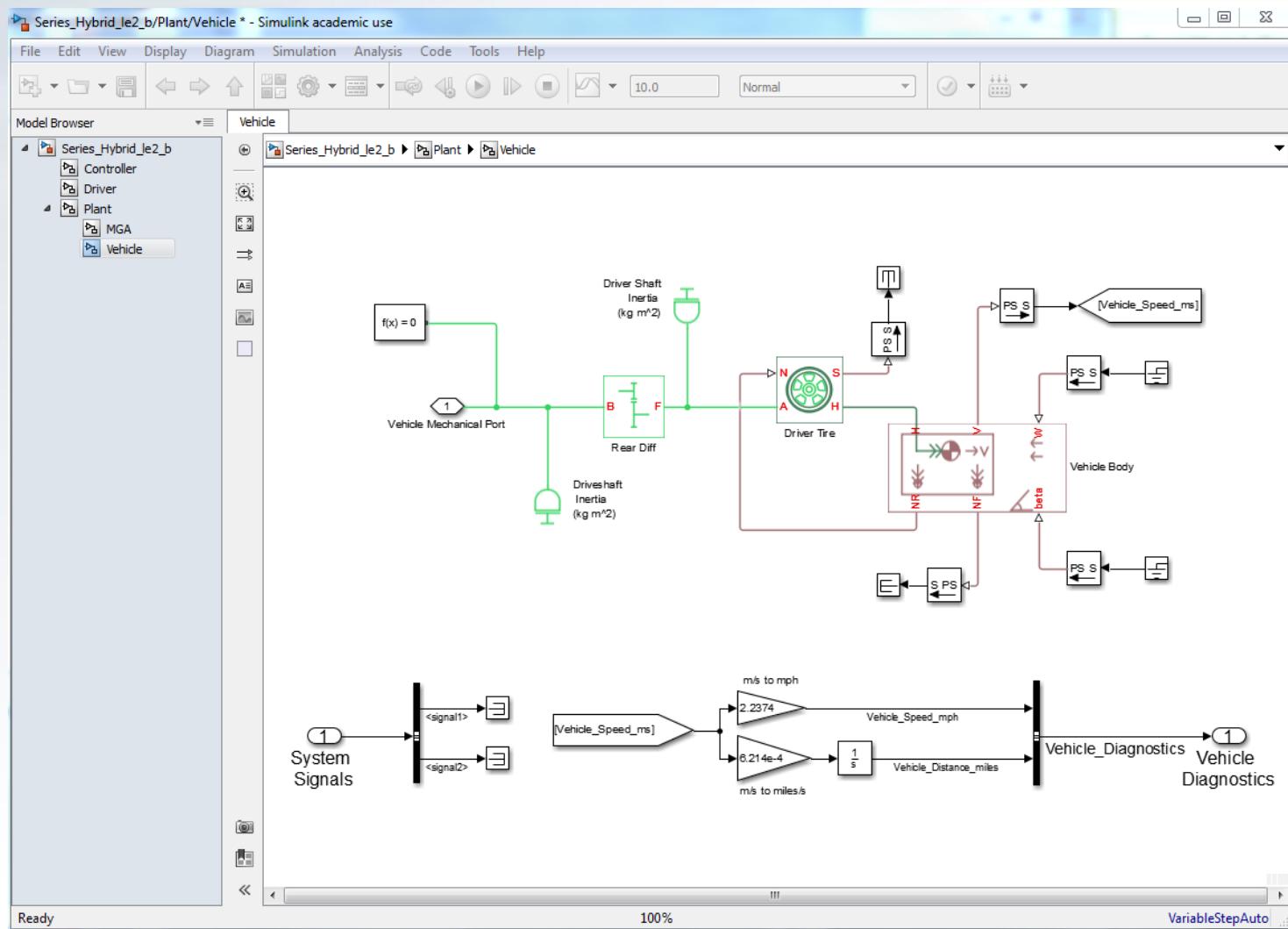


Vehicle Subsystem

- The Vehicle subsystem is done
 - Note that we did not use any of the system signals
 - When we get around to mechanical brakes we will need a brake signal
- Take some time to move things around and make it look nice



Vehicle Subsystem





MGA Subsystem

- Our motor model is going to be preposterously simple
 - Constant torque output
 - No RPM limits
 - No input power required
- Rename and save your model as:
 - Series_Hybrid_le2_c
- Use the Model Browser to go to the MGA subsystem

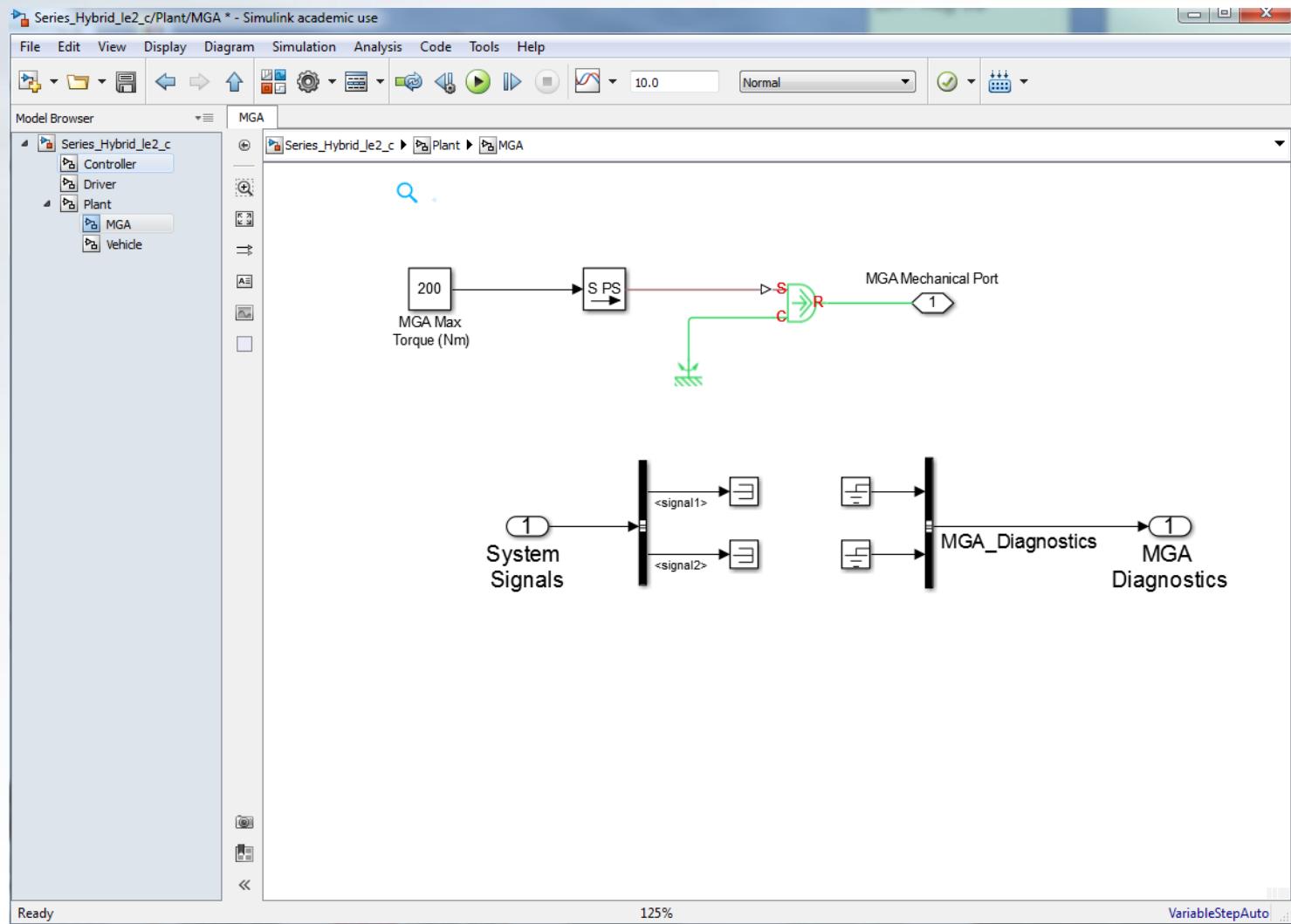


MGA Subsystem

- Delete the **In1**, **Out1**, and the connecting wire
- Copy the Bus from the Plant subsystem
 - **Rename accordingly**
- Drag in a Constant block and rename it MGA Max Torque (Nm)
- Set the constant value at 200 Nm
- From
 - **Simscape / Foundation Library / Mechanical / Mechanical Sources**
- Drag in an **Ideal Torque Source** block and hide its name
- Drag in a **Connection Port** and rename it MGA Mechanical Port
- Drag in a **Simulink-PS Converter** and hide its name
- From
 - **Simscape / Foundation Library / Mechanical / Rotational Elements**
- Grab a **Mechanical Rotational Reference** block and hide its name
- Wire everything together as shown in the next page



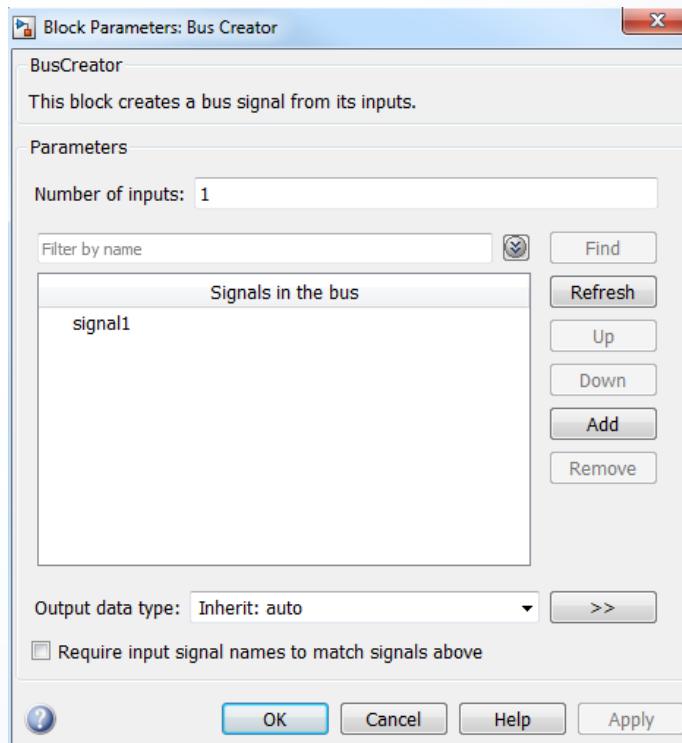
MGA Subsystem





MGA Subsystem

- The only MGA diagnostic is the torque
- Delete the **Terminators**
- Double click on the **Bus Creator** and change the Number of inputs to 1
- Click OK



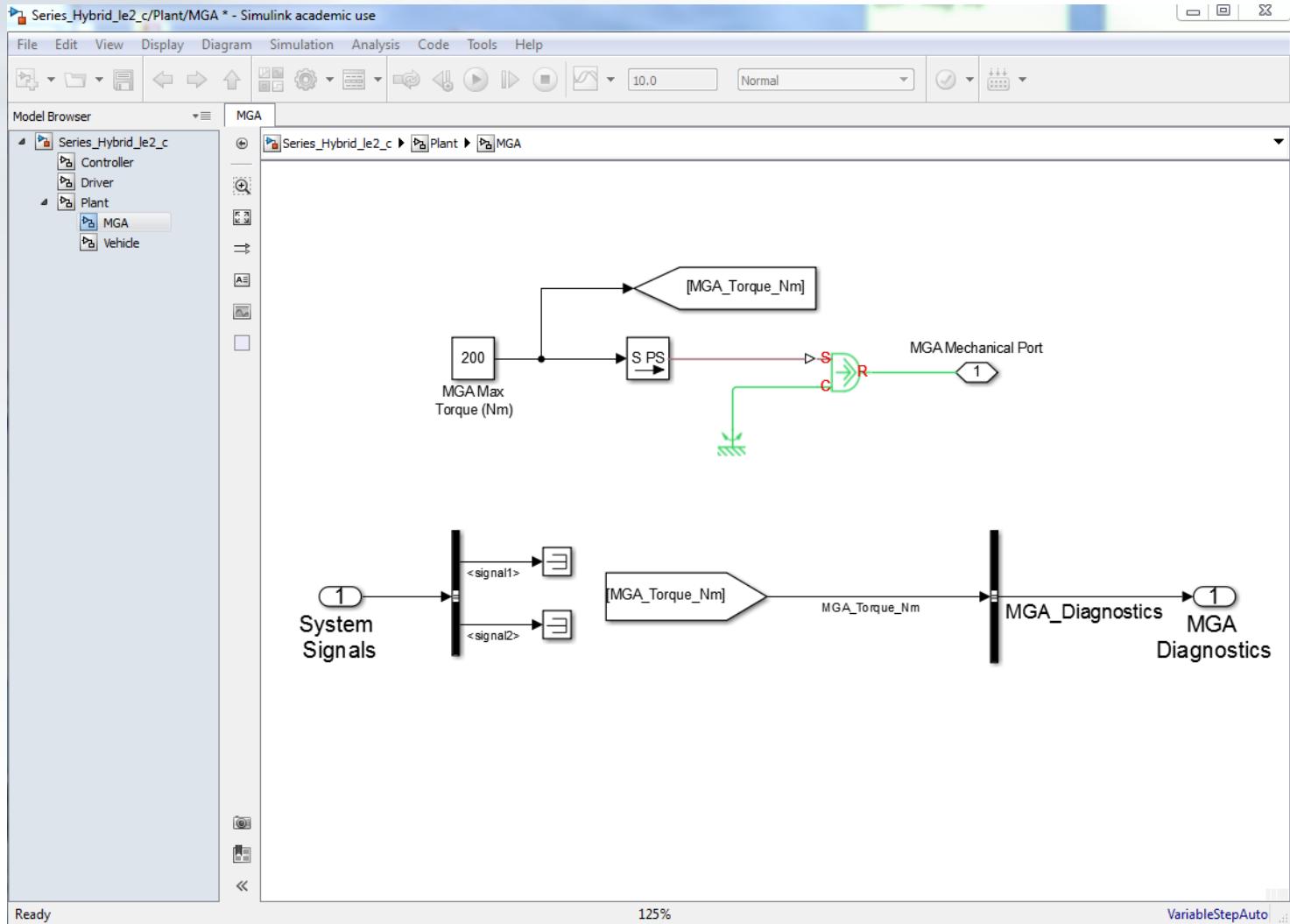


MGA Subsystem

- Drag in a pair of **Goto** and **From** blocks
- Name them accordingly
- Use them to route the MGA torque signal to the **Bus Creator** block
- Name the signal going into the Bus
- Compare with next slides



MGA Subsystem



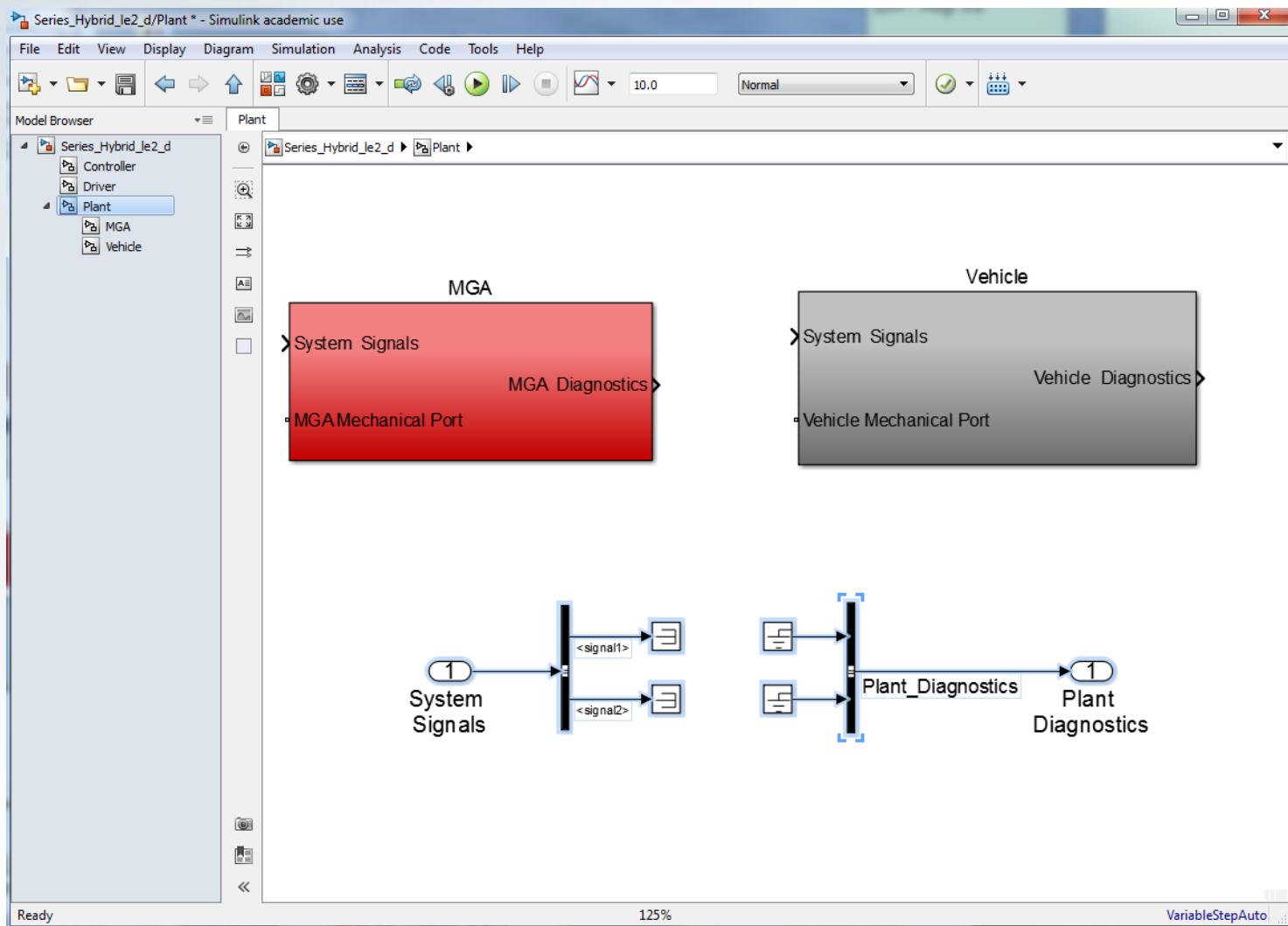


MGA Subsystem

- Congratulations – you've made a motor model
- It is now time to connect MGA to the Vehicle and see what happens
- Save your model as
 - [Series_Hybrid_le2_d](#)
- Use the Model Browser to go to the Plant level of the model
- Make the MGA and Vehicle subsystems larger



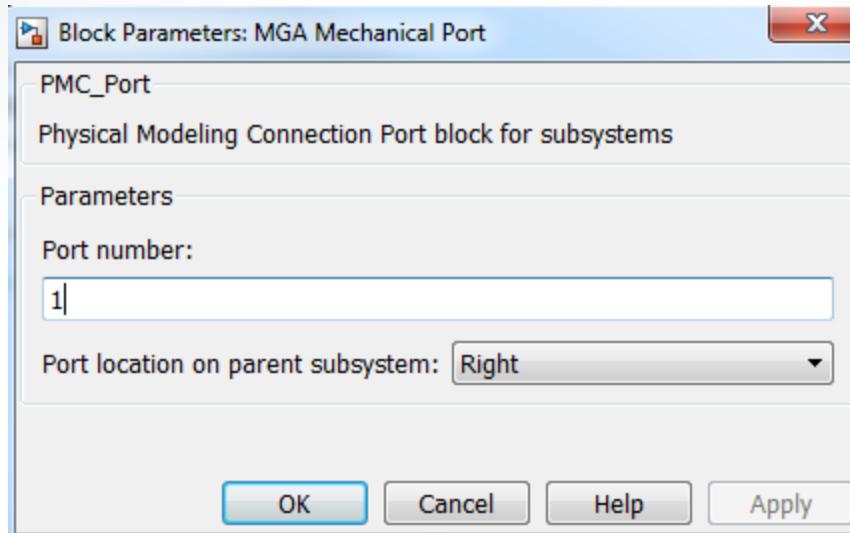
Plant Level





Plant Level

- The MGA Mechanical Port is on the wrong side of the block
- Go into the MGA subsystem and double click on the Port
- Change the location from left to right
- Click OK



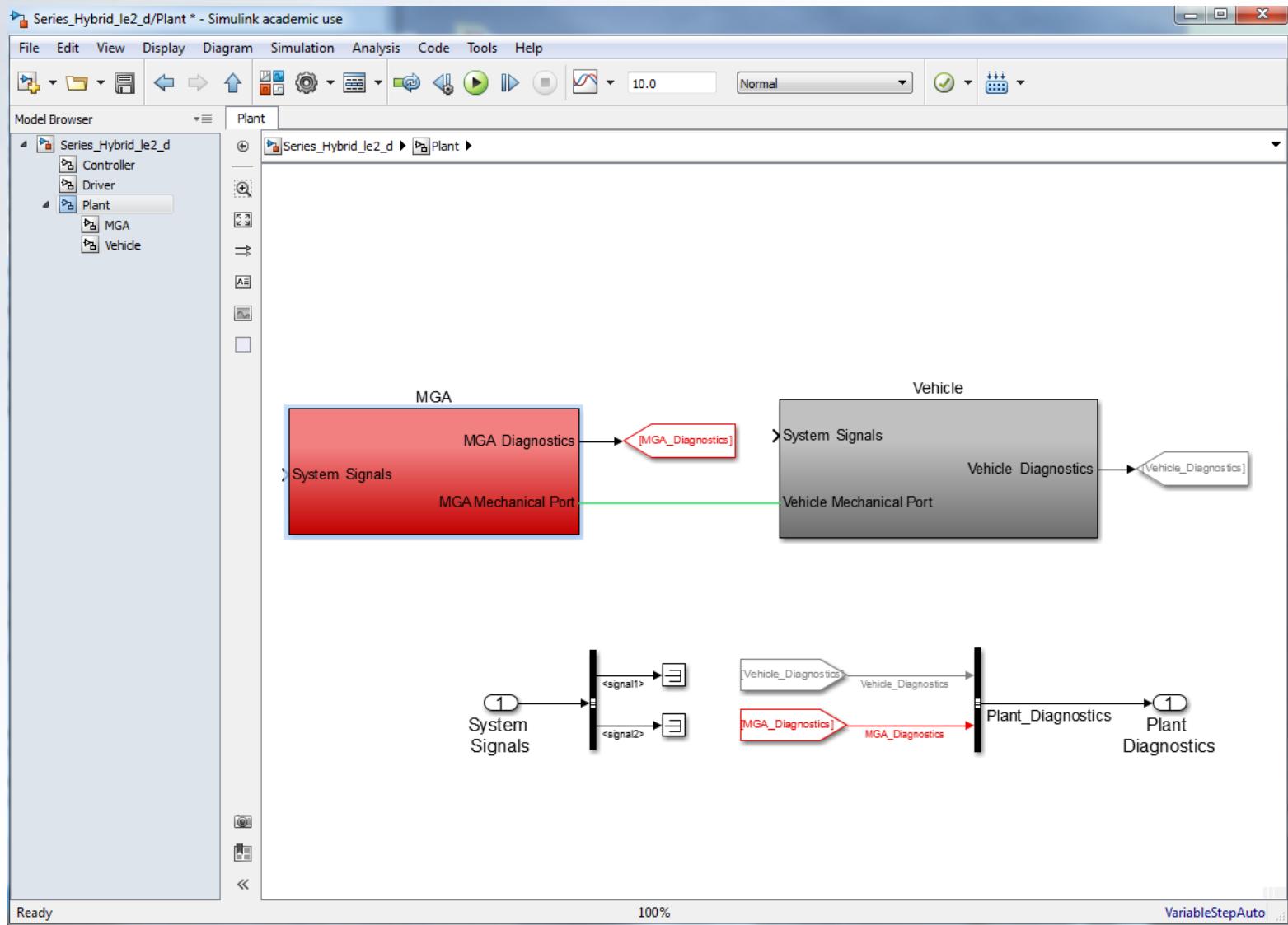


Plant Level

- Connect the two Mechanical Ports
- Route the Vehicle and MGA Diagnostics to the Plant Signal CAN Bus
 - Name the signal inputs accordingly
- Change the foreground of the Vehicle routing blocks to gray
- Change the foreground of the MGA routing blocks to red



Plant Level



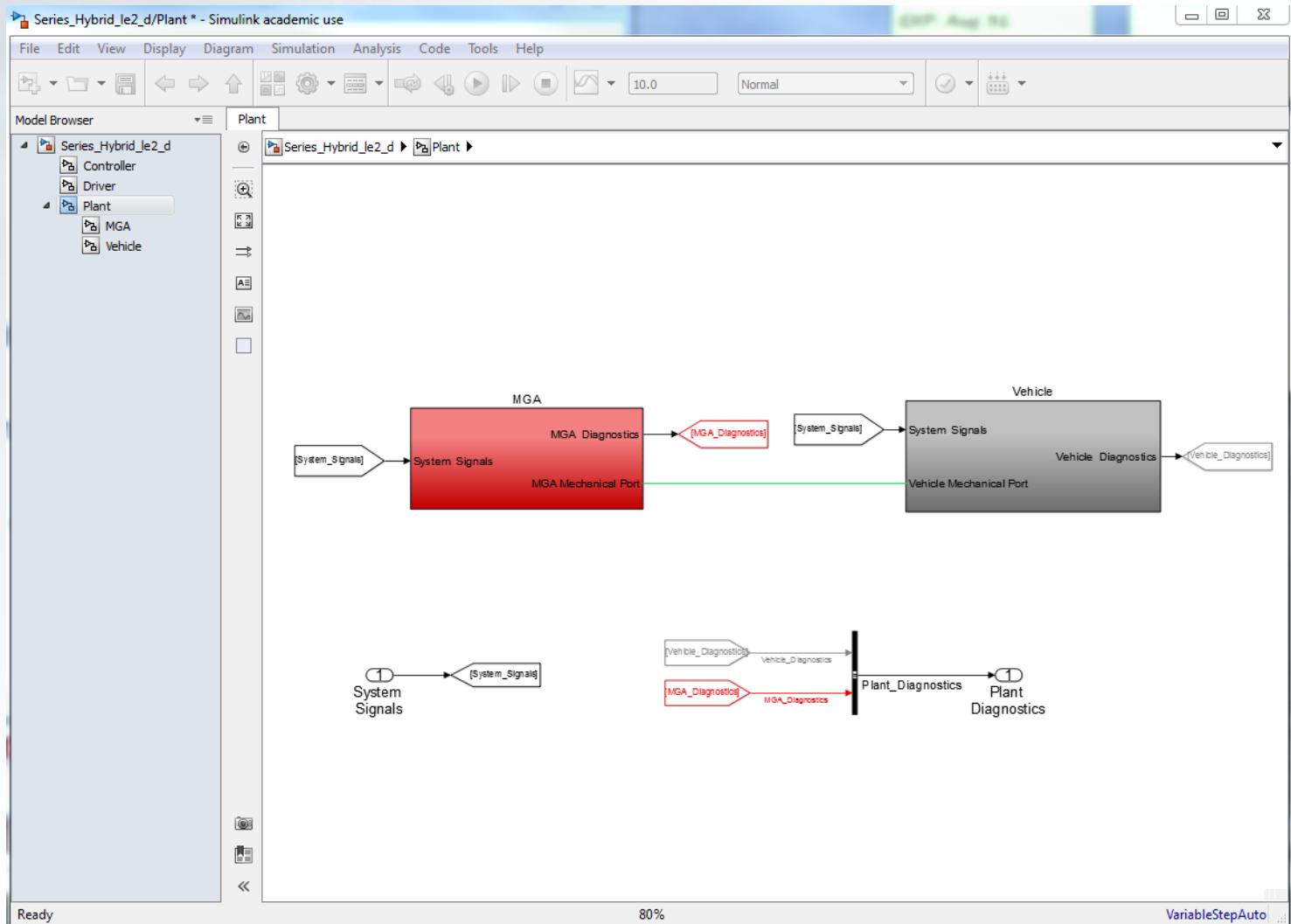


Plant Level

- All that is left is to route the System Signals to the Vehicle and MGA
- Delete the **Terminators** and the **Bus Selector**
- Use the **Goto** and **From** blocks to route the System Signals to MGA and the Vehicle



Plant Level



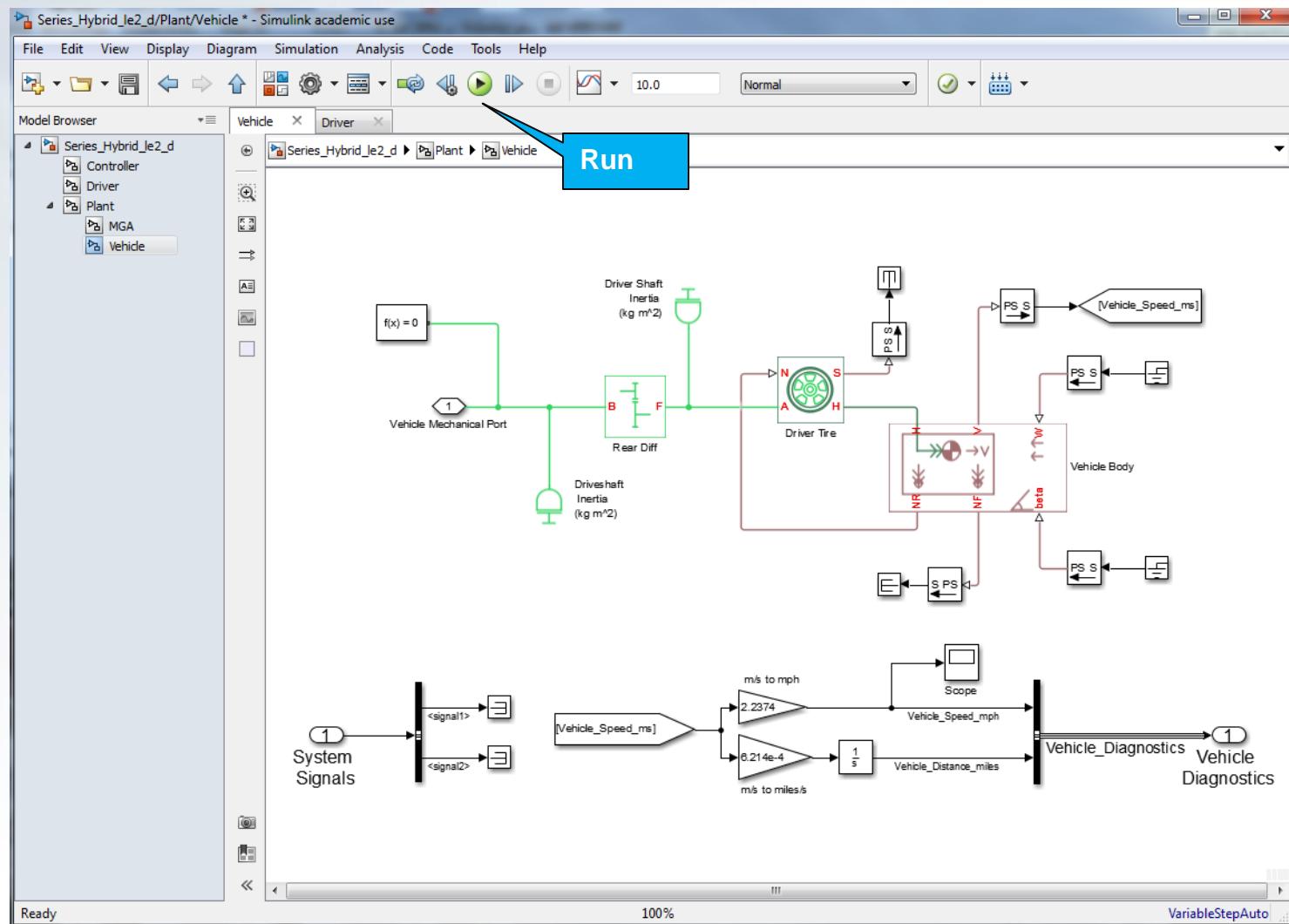


Running the Model

- It is now time to run the model
- Go to the Vehicle subsystem
- From
 - **Simulink / Sinks**
- Drag in a **Scope**
- Connect it to the Vehicle_Speed_mph signal
- Click on the Run button to start the simulation



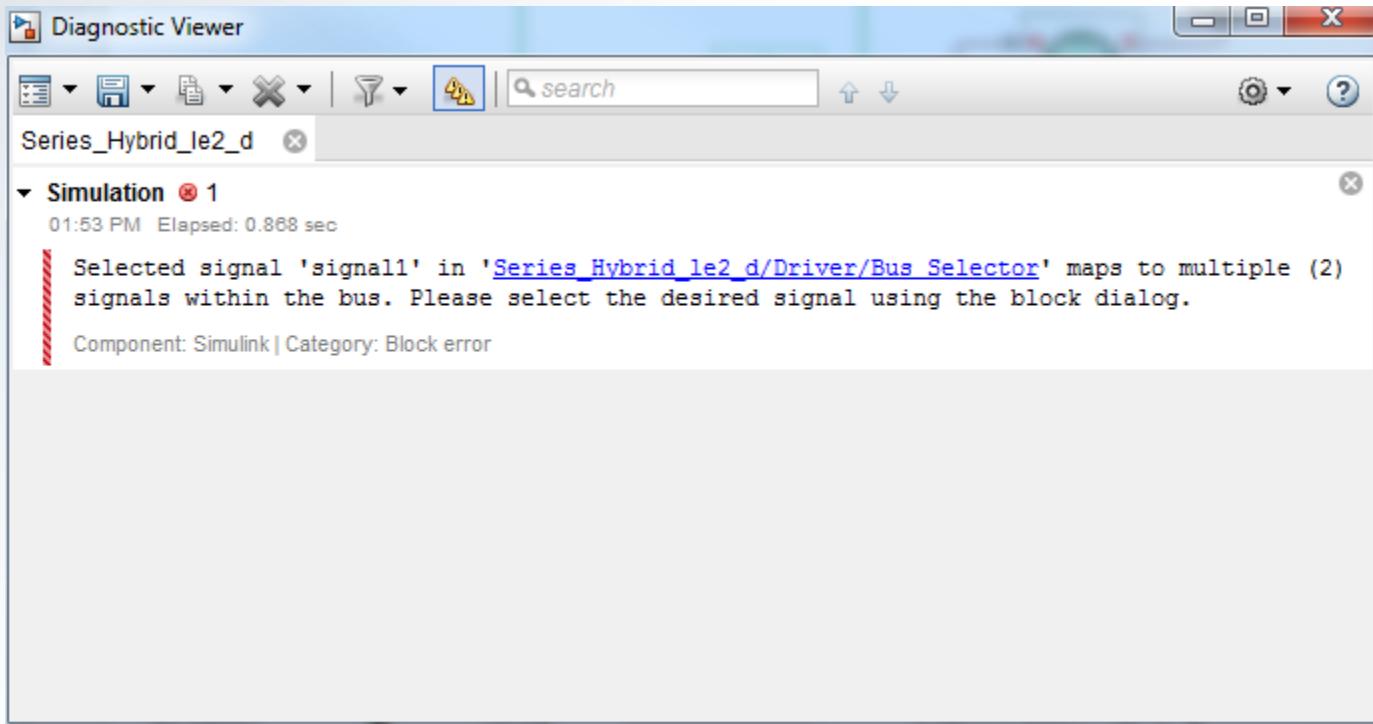
Running the Model





Running the Model

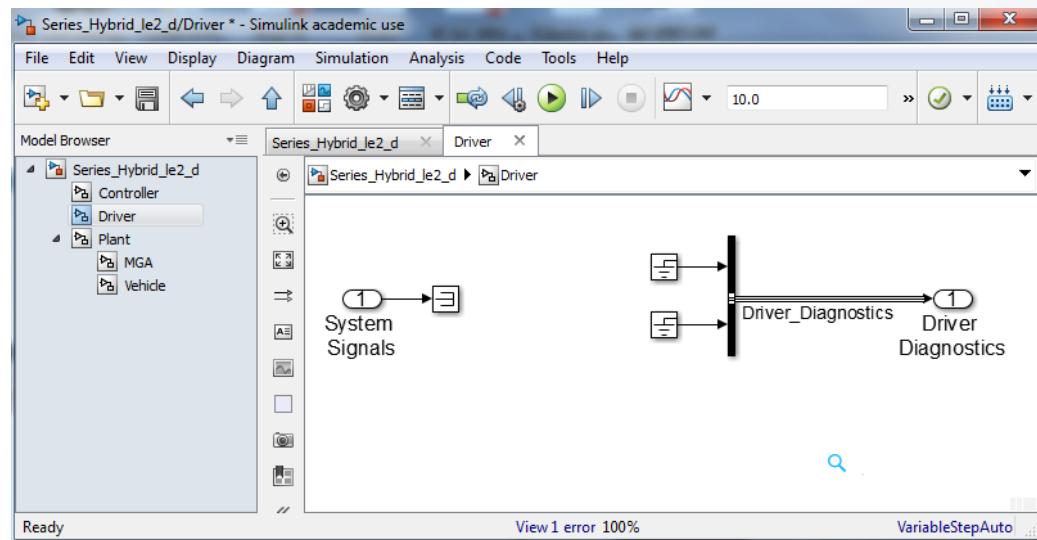
- Oh! There is an error



Running the Model



- Since we do not yet need any signals from the System Diagnostics bus, replace the **Bus Selector** with a **Terminator** in the
 - Driver
 - Controller
 - Plant / Vehicle
 - Plant / MGA
 - subsystems





Running the Model

- Rerun the model
- If the scope is not open, double click on the block
- Observe the results
 - 0 to 25 mph in 10 seconds
- You can check the results by hand

$$m \frac{dv_x}{dt} = \sum F_x$$

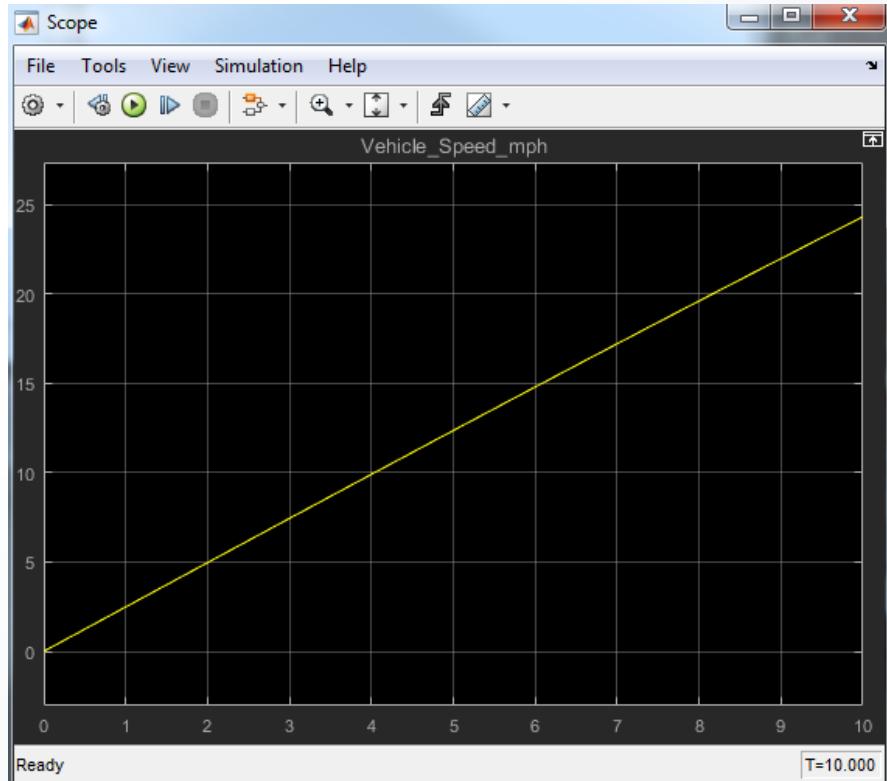
$$\frac{dv_x}{dt} = \frac{1}{m} \left(\frac{TN}{r} \right)$$

$$\int_{v_i}^{v_f} dv_x = \int_{t_i}^{t_f} \left(\frac{TN}{mr} \right) dt$$

Gear ratio

$$v_f = \frac{(200 \text{ Nm})(2)}{(1200 \text{ kg})(0.3 \text{ m})} = 11.1 \text{ m/s} = 24.9 \text{ mph}$$

Tire radius





Simple Vehicle

- Review
 - Vehicle subsystem
 - Receives torque from MGA
 - Solves vehicle velocity
 - Sends/Receives CAN messages
 - MGA subsystem
 - Provides torque to Vehicle
 - Sends/Receives CAN messages

A blue-toned photograph of the Georgia Tech Campanile, showing its brick facade and the large 'TECH' sign on top.

MBSD Lecture 3

Initialization, Solver, Logging &
Visualization, and Driver



Lecture Goals

- Create an initialization file for data management
- Select the correct ODE solver
- Create a Data Logging and Visualization subsystem
- Create a Driver subsystem
- Have the vehicle follow a drive cycle
- Experiment with feedback gain

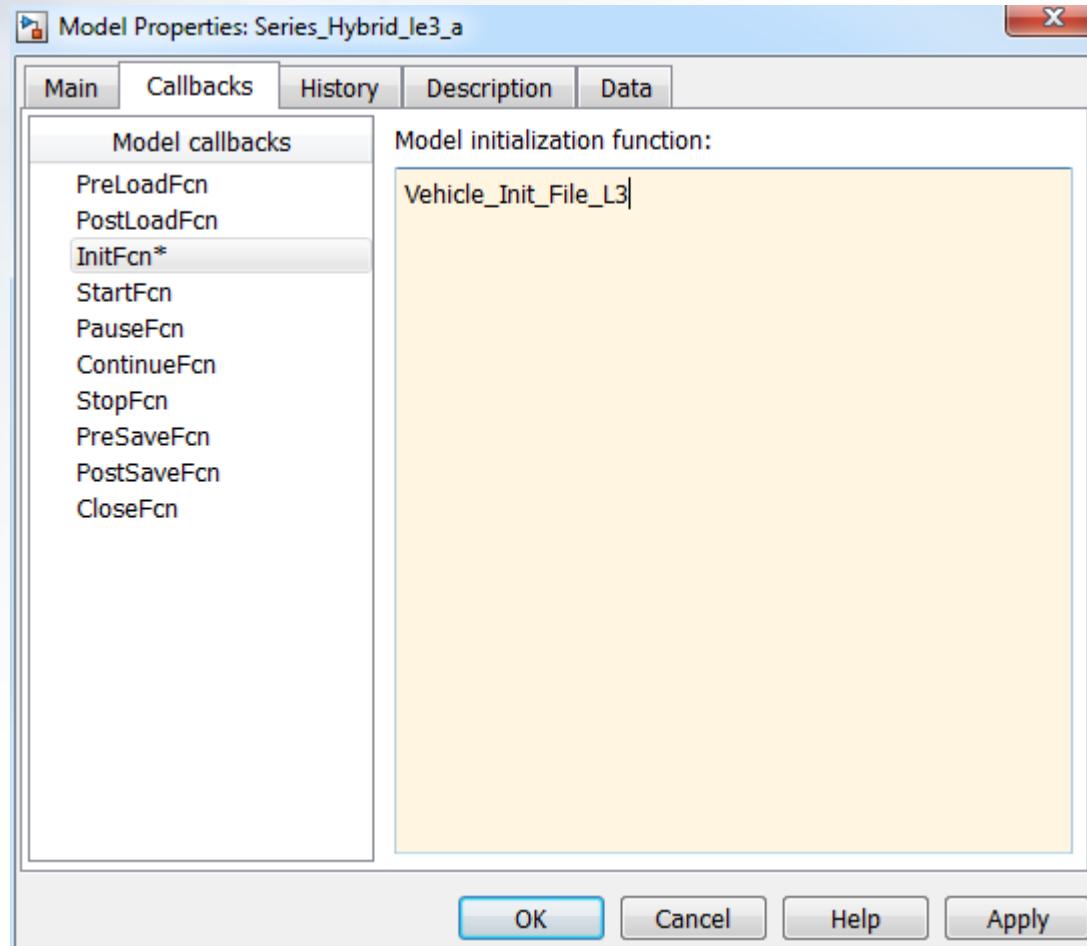


Initialization File

- Save your previous model as
 - Series_Hybrid_le3_a
- Go to the System Level of the model
- Right click on any vacant white space on the model
 - Select Model Properties
 - Select the Callbacks tab
 - Click on the InitFcn
 - Type Vehicle_Init_File_L3 in the second column



Initialization File





Initialization File

- This is going to be a Matlab M-file which will automatically execute each time the model is run
- All of our component parameters will go in this file
- Start by going to the Matlab window
- Open file
 - [Vehicle_Init_File_L3.m](#)



Initialization File

C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m

The screenshot shows the MATLAB IDE interface with the following details:

- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Go To, Find, Breakpoints, Run, Run and Advance, Run Section, Advance, and Run and Time.
- Editor Area:** Displays the MATLAB script code.
- Code Content:**

```
1 % Series Electric Vehicle Init File
2 - clc
3 - clear variables
4
5 %Vehicle Parameters
6 - Vehicle_Mass = 1200; %Vehicle mass, kg
7 - Vehicle_Tire_Radius = 0.3; %Tire radius, m
8 - Vehicle_DShaft_Inertia = 1.5e-3; %Driveshaft Inertia, kg m^2
9 - Vehicle_Dr_Shaft_Inertia = 3.0e-3; %Driver shaft Inertia, kg m^2
10 - Vehicle_Rear_Diff_Ratio = 2; %Rear Diff Ratio
11 -
12 %MGA Parameters
13 - MGA_Max_Torque = 200; %MGA Max Torque, Nm
14
15
16
```
- Status Bar:** Shows "Ln 11 Col 1".



Initialization File

- We will replace the numeric values in the blocks with these variable names
 - This consolidates all key parameters into one location
 - Help prevent making mistakes
 - Tip: copy the variable names from the M-file and paste into the block to prevent spelling errors
- Update all relevant blocks



Initialization File

Series_Hybrid_le3_a/Plant/Vehicle * - Simulink academic use

File Edit View Display Diagram Simulation Analysis Code Tools Help

Model Browser

- Series_Hybrid_le3_a
 - Controller
 - Driver
 - Plant
 - MGA
 - Vehicle

Vehicle

Series_Hybrid_le3_a > Plant > Vehicle

10.0

Driver Shaft Inertia (kg m²)

f(x) = 0

Vehicle Mechanical Port

Rear Diff

B F

Block Parameters: Rear Diff

Simple Gear

Represents a fixed-ratio gear or gear box. No inertia or compliance include gear meshing and viscous bearing losses.

Connections B (base) and F (follower) are mechanical rotational conserving ports. Specify the relation between base and follower rotation directions with the Output shaft rotates parameter. Optionally include thermal effects and expose thermal conserving port H by right-clicking on the block and selecting Simscape block choices to switch between variants.

Settings

Main Meshing Losses Viscous Losses

Follower (F) to base (B) teeth ratio (NF/NB):

Output shaft rotates:

OK Cancel Help Apply

C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m

EDITOR PUBLISH VIEW

New Open Save Compare Print EDIT Breakpoints Run Run and Advance

FILE NAVIGATE BREAKPOINTS RUN

```
% Series Electric Vehicle Init File
1 clc
2 clear variables
3
4
5 %Vehicle Parameters
6 Vehicle_Mass = 1200; %Vehicle mass, kg
7 Vehicle_Tire_Radius = 0.3; %Tire radius, m
8 Vehicle_DShaft_Inertia = 1.5e-3; %Driveshaft Inertia, kg m^2
9 Vehicle_Dr_Shft_Inertia = 3.0e-3; %Driver shaft Inertia, kg m^2
10 Vehicle_Rear_Diff_Ratio = 2; %Rear Diff Ratio
11
12 %MGA Parameters
13 MGA_Max_Torque = 200; %MGA Max Torque, Nm
14
15
16
17
```

Vehicle Diagnostics

Vehicle Diagnostics

Ready 100% VariableStepAuto



ODE Solver

- While the Solver Configuration calls the hooks for solving the driveline angular momentum, an overall ODE solver must be specified
- Our model contains a set of stiff ODEs and as recommended by The MathWorks, best solver suited for this type of ODEs is ODE23t



ODE Solver

- As with the Init file, right click on the model background
- Select
 - Model Configuration Parameters
 - Solver: ode23t (mod. stiff/Trapezoidal)
 - Click OK
- Run the model and note how much faster it runs!



ODE Solver

Configuration Parameters: Series_Hybrid_Le6_a/Configuration (Active)

★ Commonly Used Parameters All Parameters

Select: Solver

Simulation time

Start time: 0.0 Stop time: Sch_Cycle(end,1)

Solver options

Type: Variable-step Solver: ode23t (mod. stiff/Trapezoidal)

Additional options

ode23t (mod. stiff/Trapezoidal)
auto (Automatic solver selection)
discrete (no continuous states)
ode45 (Dormand-Prince)
ode23 (Bogacki-Shampine)
ode113 (Adams)
ode15s (stiff/NDF)
ode23s (stiff/Mod. Rosenbrock)
ode23tb (stiff/TR-BDF2)

OK Cancel Help Apply



Logging & Visualization

- Let's now create a subsystem to represent the on-board data logger
- This will also be a good place for having all the visualization
- First, go to the Vehicle subsystem and delete the **Scope** block
 - Having scopes in the model is bad for auto-code generation



Logging & Visualization

- Drag a new **Subsystem** into the Plant level of the model
- Rename it Logging & Visualization
- Rename the **In1** block to System Signals
- Delete the **Out1** block
 - The logger does not send any diagnostics
- Drag a **Bus Selector** into the subsystem
- Go back into the Plant level of the model
- Copy a System Signals **From** block and connect it to System Signals of Logging & Visualization subsystem



Logging & Visualization

Series_Hybrid_le3_a/Plant/Logging & Visualization * - Simulink academic use

File Edit View Display Diagram Simulation Analysis Code Tools Help

Model Browser Logging & Visualization

Series_Hybrid_le3_a ► Plant ► Logging & Visualization

System Signals

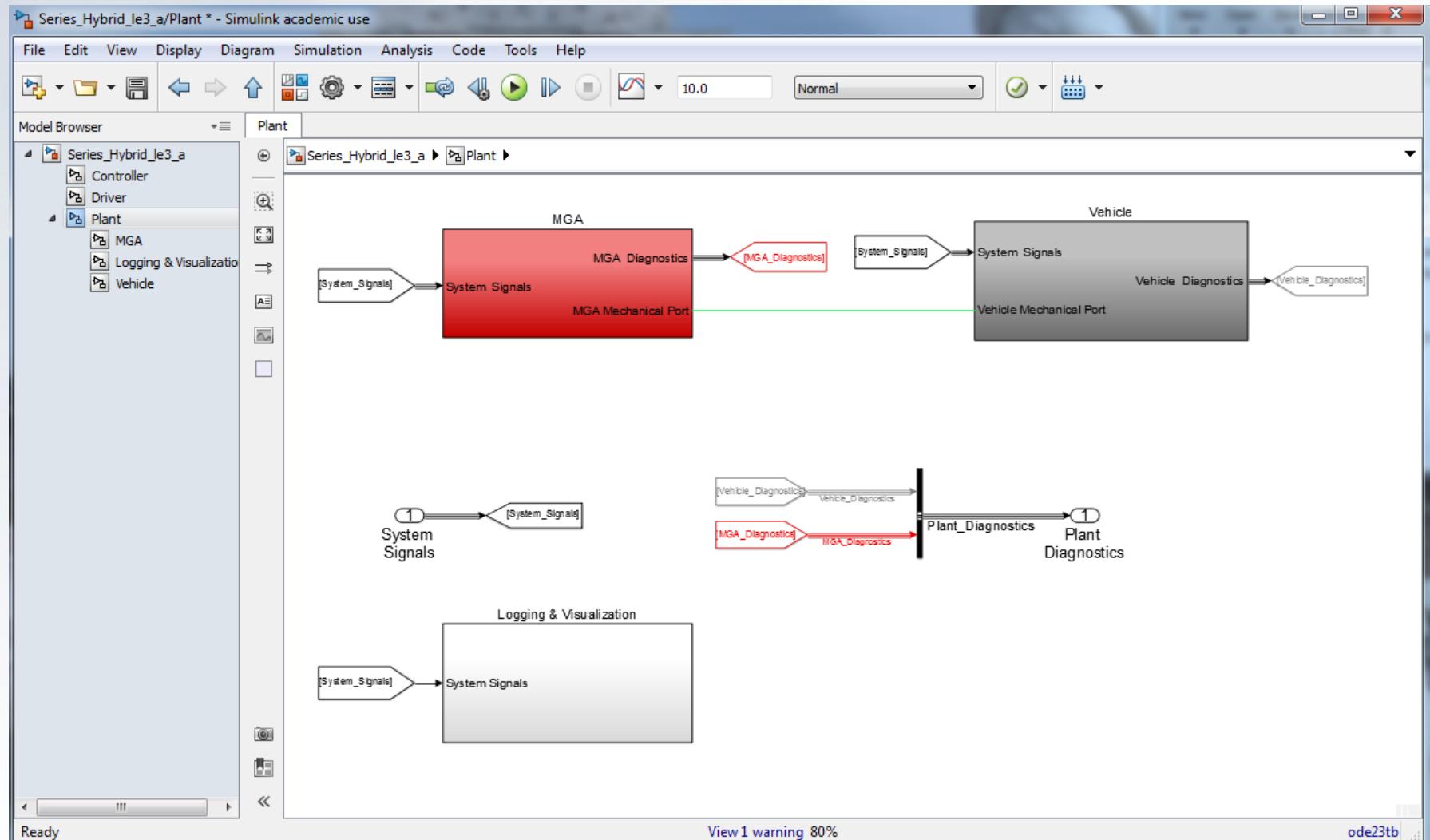
ode23tb

```
graph LR; A[1 System Signals] --> B[Scope];
```

The screenshot shows a Simulink model window titled "Series_Hybrid_le3_a/Plant/Logging & Visualization". The Model Browser on the left lists the model structure: Series_Hybrid_le3_a, Controller, Driver, Plant, MGA, Logging & Visualization (which is selected), and Vehicle. The main workspace displays a single block, a Scope, with a signal labeled "1 System Signals" entering it. The status bar at the bottom indicates "Ready", "View 1 warning 100%", and "ode23tb".



Logging & Visualization



Ready

View 1 warning 80%

ode23tb



Logging & Visualization

- In the Logging & Visualization subsystem
- Double click on the **Bus Selector** block
- In the Selected signals column
 - Click on signal1 and then Remove
 - Click on signal1 and then Remove
- In the Signals in the bus column
 - Click on Driver_Diagnostics and then Select
 - Click on Plant_Diagnostics and then Select
 - Click on Controller_Diagnostics and then select
- Click OK



Logging & Visualization

Block Parameters: Bus Selector

BusSelector

This block accepts a bus as input which can be created from a Bus Creator, Bus Selector or a block that defines its output using a bus object. The left listbox shows the signals in the input bus. Use the Select button to select the output signals. The right listbox shows the selections. Use the Up, Down, or Remove button to reorder the selections. Check 'Output as bus' to output a single bus signal.

Parameters

Filter by name

Signals in the bus

- ▷ Driver_Diagnostics
- ▷ Plant_Diagnostics
- ▷ Controller_Signals

Find

Select>>

Refresh

Selected signals

- Driver_Diagnostics
- Plant_Diagnostics
- Controller_Signals

Up

Down

Remove

Output as bus

OK Cancel Help Apply



Logging & Visualization

- Drag in three **Goto** blocks and name them Driver_Diagnostics, Plant_Diagnostics, and Controller_Signals
- Wire them to the **Bus Selector**
- Drag in three **From** blocks
- **Terminate** the Driver_Diagnostics and the Controller_Signals **From** blocks
- Drag in a **Bus Selector** and wire it to the Plant_Diagnostics **From** block

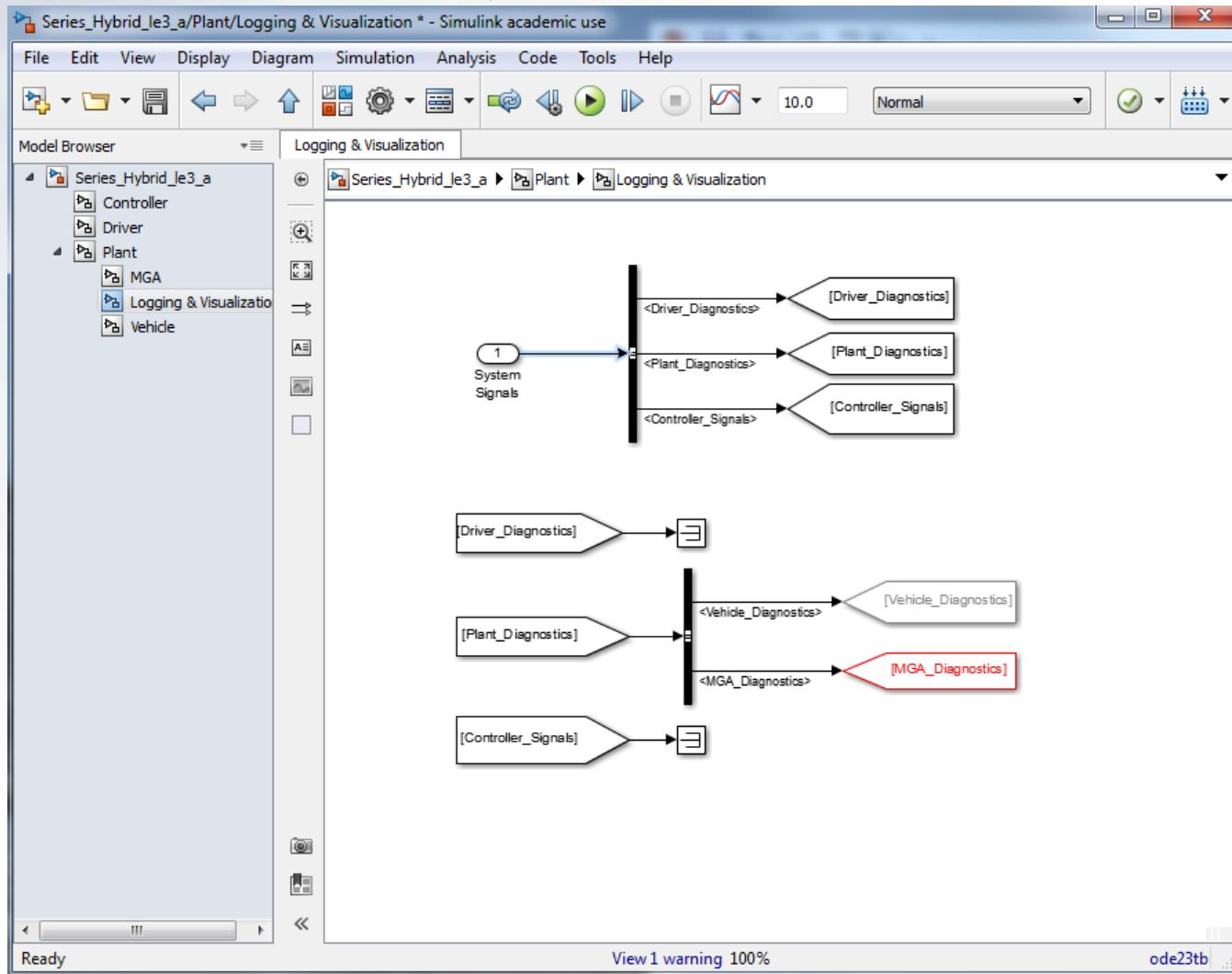


Logging & Visualization

- Double click on the **Bus Selector**
- Remove signal1 and signal2
- Select the Vehicle_Diagnostics and the MGA_Diagnostics signals
- Drag in two more **Goto** blocks
 - Connect them to the Bus Selector
 - Name them accordingly
 - Color them accordingly



Logging & Visualization



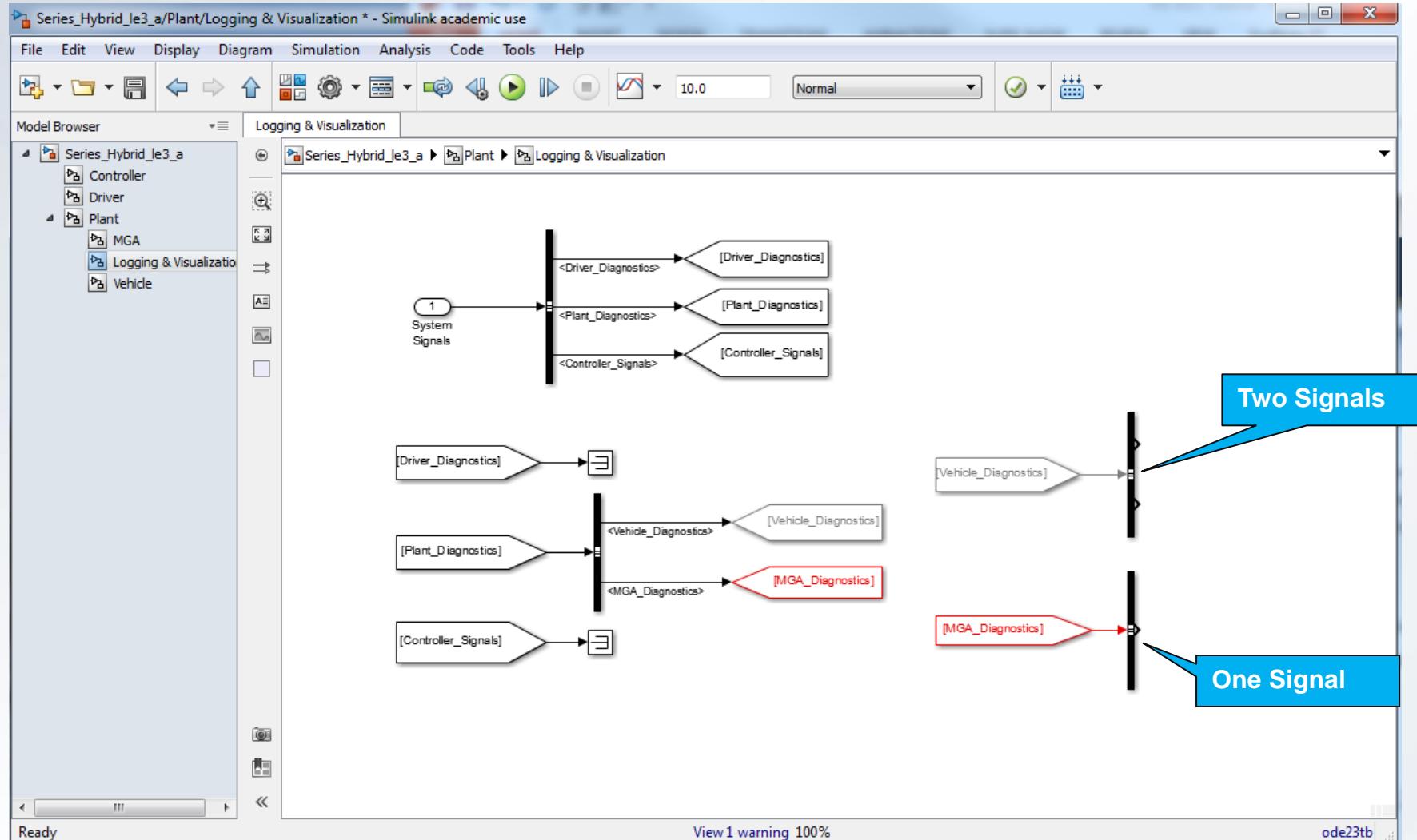


Logging & Visualization

- Drag in two more **From** and **Bus Selector** blocks
- Select
 - **Vehicle_Diagnostics** - first **From** block
 - **MGA_Diagnostics** – second **From** block
- Connect the **From** blocks to the **Bus Selector** blocks
- Use the **Bus Selector** blocks to select all signals



Logging & Visualization





Logging & Visualization

- From
 - Simulink / Sinks
- Drag in a **To Workspace** block
 - This will write the desired signal values to an array for review when the simulation is done
- Double click on the **To Workspace** block
 - Make the following changes
 - Variable name - Vehicle_Speed_mph
 - Sample time - Logging_Sample_Time_s
 - Save format – Array
- Update the input file



Logging & Visualization

The screenshot shows the MATLAB/Simulink environment. On the left, a script editor window displays the 'Vehicle_Init' script. A red box highlights the 'Logging Parameters' section, which includes the line `Logging_Sample_Time_s = 1;`. On the right, a 'Block Parameters: To Workspace' dialog box is open. A blue callout bubble points to the 'Decimation' field, which contains the value '1'. The text 'Very important: Don't forget this' is overlaid on the callout bubble. The dialog box also contains fields for 'Variable name' (set to 'Vehicle_Speed_mph'), 'Limit data points to last' (set to 'inf'), 'Save format' (set to 'Array'), 'Save 2-D signals as' (set to '3-D array (concatenate along third dimension)'), and a checked checkbox for 'Log fixed-point data as a fi object'. The 'Sample time (-1 for inherited)' field is set to 'Logging_Sample_Time_s'.

```
% Series Electric Vehicle Init File
clc
clear variables

%Vehicle Parameters
Vehicle_Mass = 1200; %Vehicle Mass
Vehicle_Tire_Radius = 0.3; %Tire radius
Vehicle_DShaft_Inertia = 1.5e-3; %Driveshaft Inertia
Vehicle_Dr_Shaft_Inertia = 3.0e-3; %Driver shaft inertia
Vehicle_Rear_Diff_Ratio = 2; %Rear Differential Ratio

%MGA Parameters
MGA_Max_Torque = 200; %MGA Max Torque

%Logging Parameters
Logging_Sample_Time_s = 1; %Logging Sample Time
```

Block Parameters: To Workspace

To Workspace

Write input to specified timeseries, array, or structure in a workspace. For menu-based simulation, data is written in the MATLAB base workspace. Data is not available until the simulation is stopped or paused.

To log a bus signal, use "Timeseries" save format.

Parameters

Variable name:

Vehicle_Speed_mph

Limit data points to last:

inf

Decimation:

1

Very important:
Don't forget this

Save format: Array

Save 2-D signals as: 3-D array (concatenate along third dimension)

Log fixed-point data as a fi object

Sample time (-1 for inherited):

Logging_Sample_Time_s

OK Cancel Help Apply

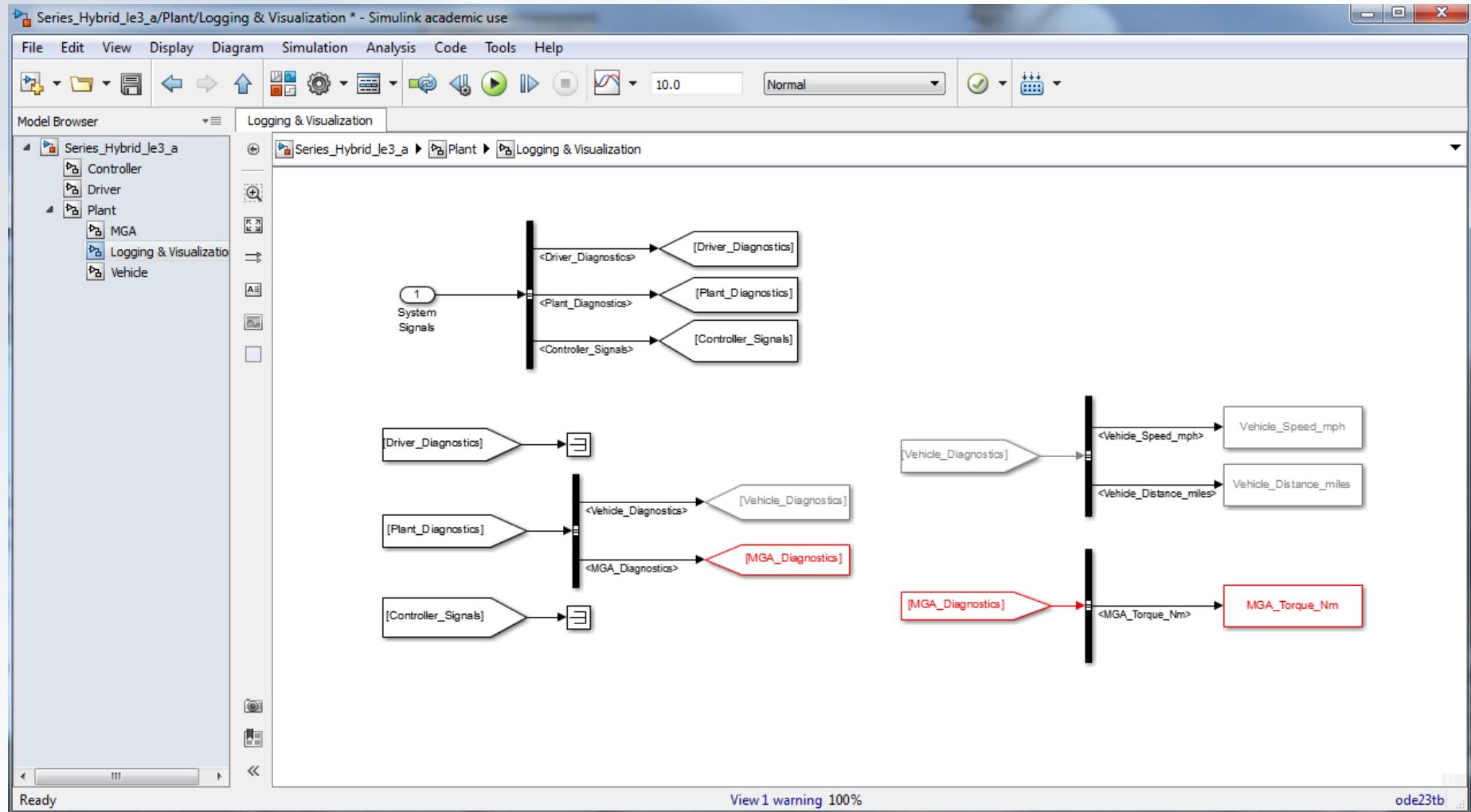


Logging & Visualization

- Connect the **To Workspace** block to the **Bus Selector**
- Repeat for the remaining signals coming from the Plant
- Color everything accordingly



Logging & Visualization



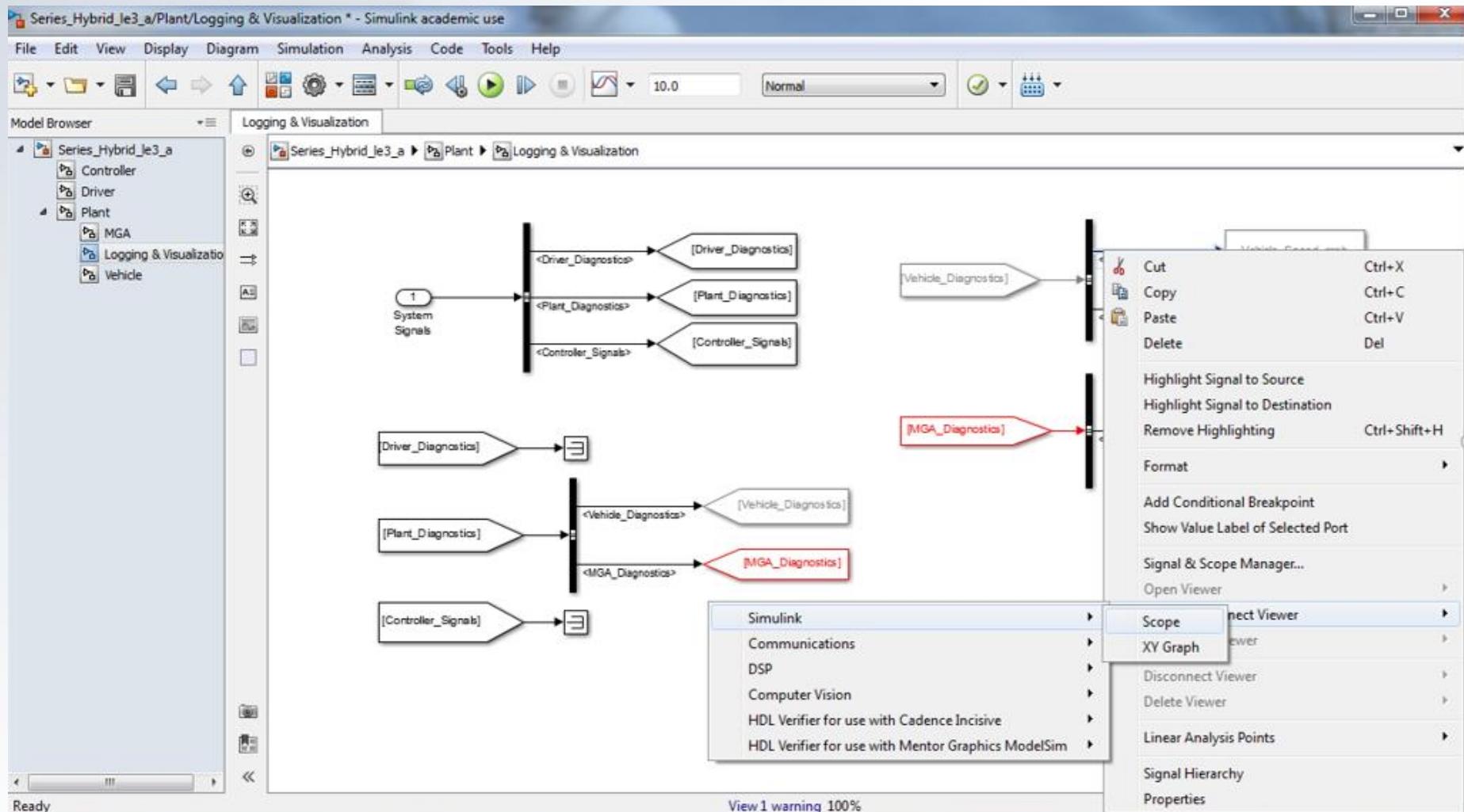


Logging & Visualization

- Rather than using Scope blocks, we will use the Signal and Scope Manager to view the signals
- Right click on the Vehicle_Speed_mph signal and select
 - Create & Connect Viewer / Simulink / Scope



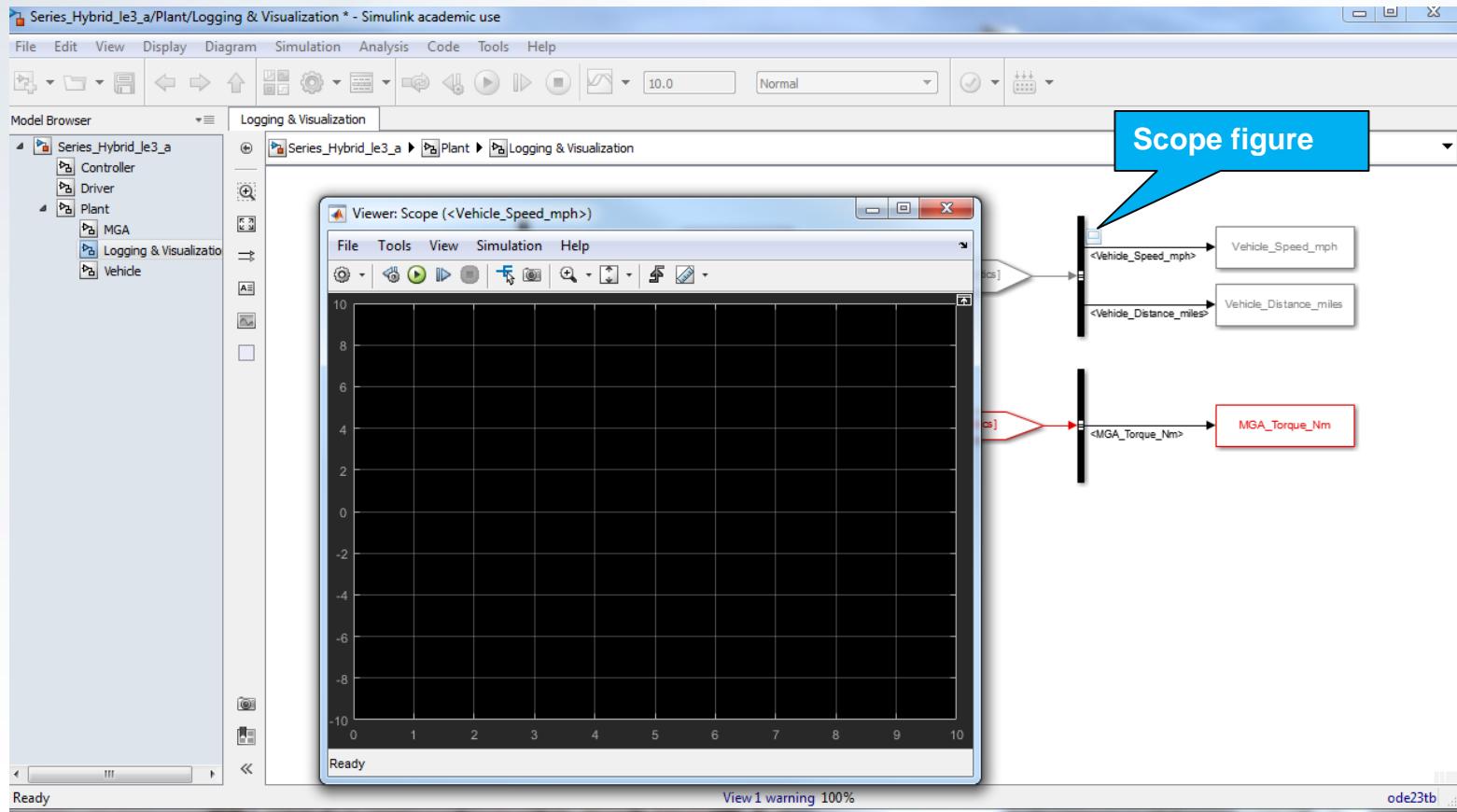
Logging & Visualization





Logging & Visualization

- Note the scope figure on the Vehicle_Speed_mph signal – that denotes that the signal is being scoped





Driver Subsystem

- The Driver subsystem will be our first proportional feedback loop
 - The vehicle will be at a certain speed
 - The driver will want to go a different speed
 - The driver will send a torque request (-1 to 1) to the controller
 - The vehicle will change speed
 - Repeat for the duration of the drive cycle

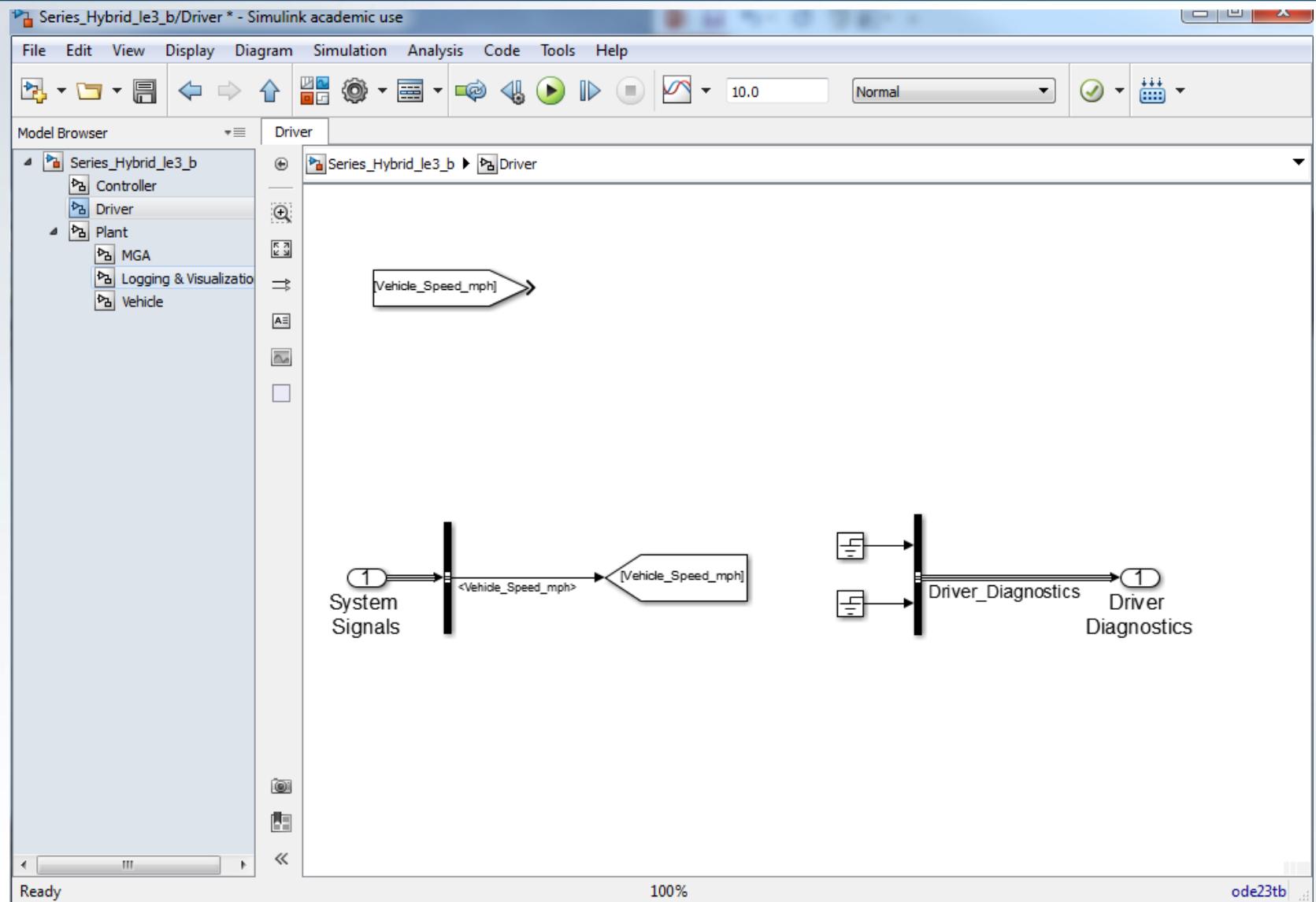


Driver Subsystem

- Save the model as Series_Hybrid_le3_b
- Use the Model Browser to go into the established Driver subsystem
- Drag in a **Bus Selector**, a **Goto**, and a **From** block
- Delete the **Terminator** and connect the **Bus Selector** block
- Select the Vehicle_Speed_mph signal
- Connect this signal to the **Goto** block



Driver Subsystem





Driver Subsystem

- The drive cycle will be read in from the workspace
 - It will contain time (s) and desired speed (mph) at that time
 - We will use the standard FU505 drive cycle
 - 505 seconds long
 - Collected data
 - The data is stored as a .mat file
 - The .mat file will be loaded with the init file

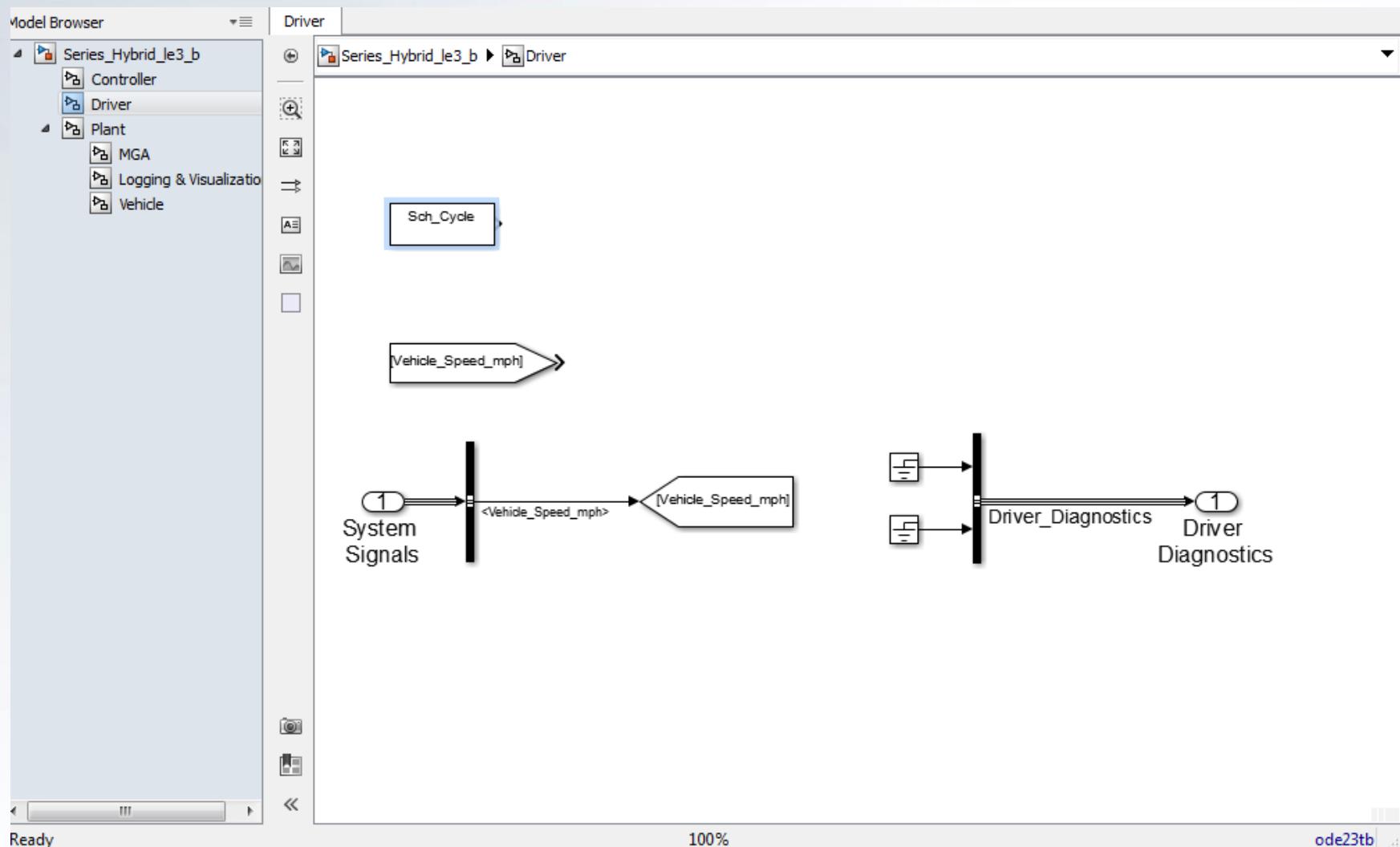


Driver Subsystem

- From
 - Simulink / Sources
- Drag in a **From Workspace** block
 - Double click on it
 - Rename the Data to: Sch_Cycle
 - Click OK



Driver Subsystem



Ready

100%

ode23tb

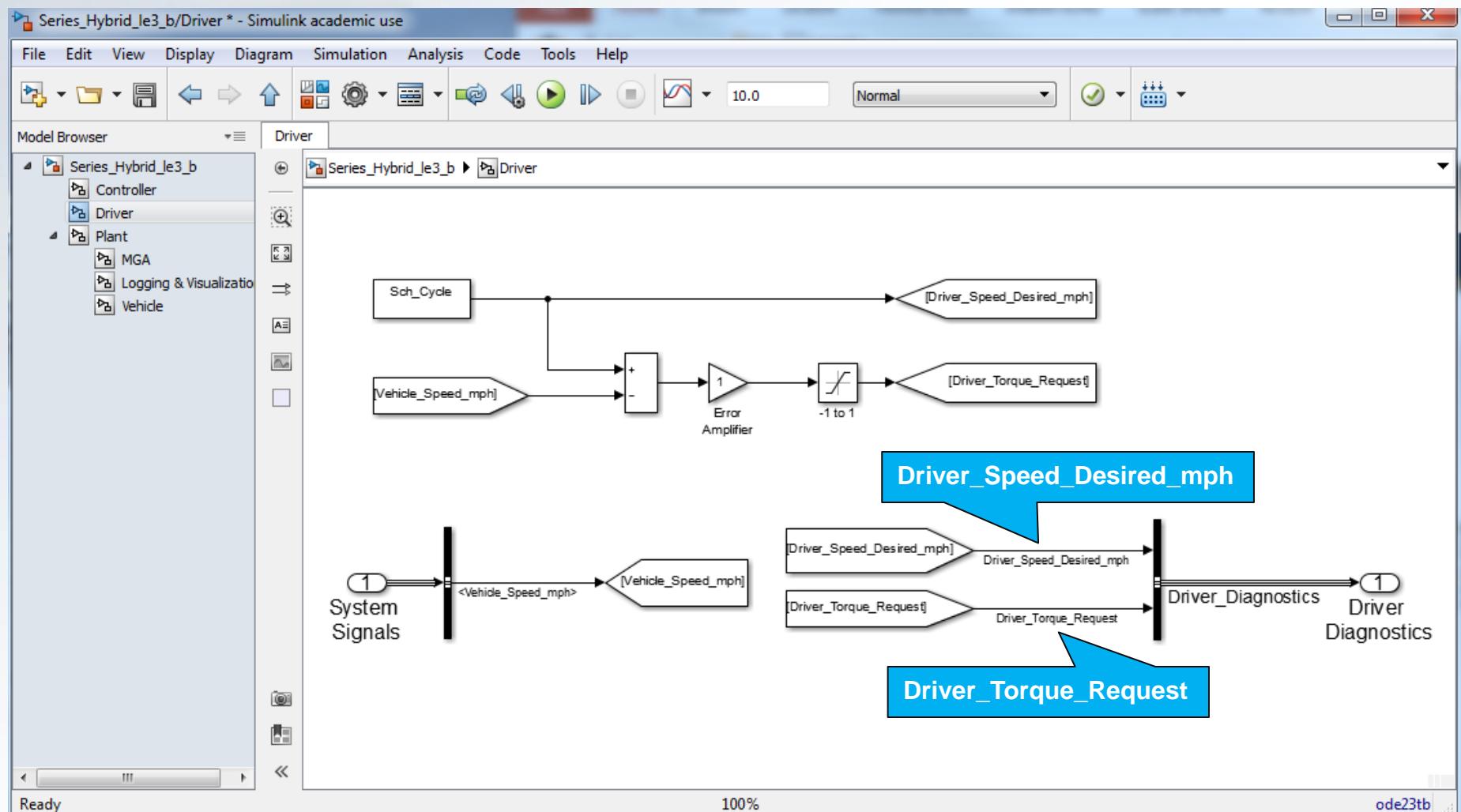


Driver Subsystem

- From
 - Simulink / Commonly Used Blocks
- Drag in a **Sum**, **Gain**, and **Saturation** block
 - Make the Sum block rectangular with a +-
- Drag in another two **Goto** and **From** blocks
- Arrange, wire, and name as shown on the next slide



Driver Subsystem





Driver Subsystem

- We must now determine the value of the error amplifier
- Classical controls theory would have us generate a transfer function and look for poles
- This system will soon become so complex that classic approaches do not work
- Instead use some experience
 - 1 mph faster
 - 10 mph faster
 - 20 mph faster
- You step on the gas (torque request)
 - Very little
 - Half way
 - To the floor



Driver Subsystem

- The torque request will vary from
 - +1
 - full positive torque requested
 - 20 mph+ too slow
 - -1
 - Full negative torque requested
 - 20 mph+ too fast
- A gain of 0.05 will send a ± 1 at 20 mph error
- A Saturation will limit the request to ± 1



Driver Subsystem

Series_Hybrid_le3_b/Driver * - Simulink academic use

File Edit View Display Diagram Simulation Analysis Code Tools Help

Model Browser Driver

Series_Hybrid_le3_b > Driver

Saturation

Limit input signal to the upper and lower saturation values.

Main Signal Attributes

Upper limit: 1

Lower limit: -1

Treat as gain when linearizing

Enable zero-crossing detection

OK Cancel Help Apply

Block Diagram:

```
graph LR; Sch_Cycle --> Sum(( )); [Vehicle_Speed_mph] --> Sum; Sum --> Vehicle; Vehicle --> Diagnostics
```

System Signals

Vehicle

Driver Diagnostics

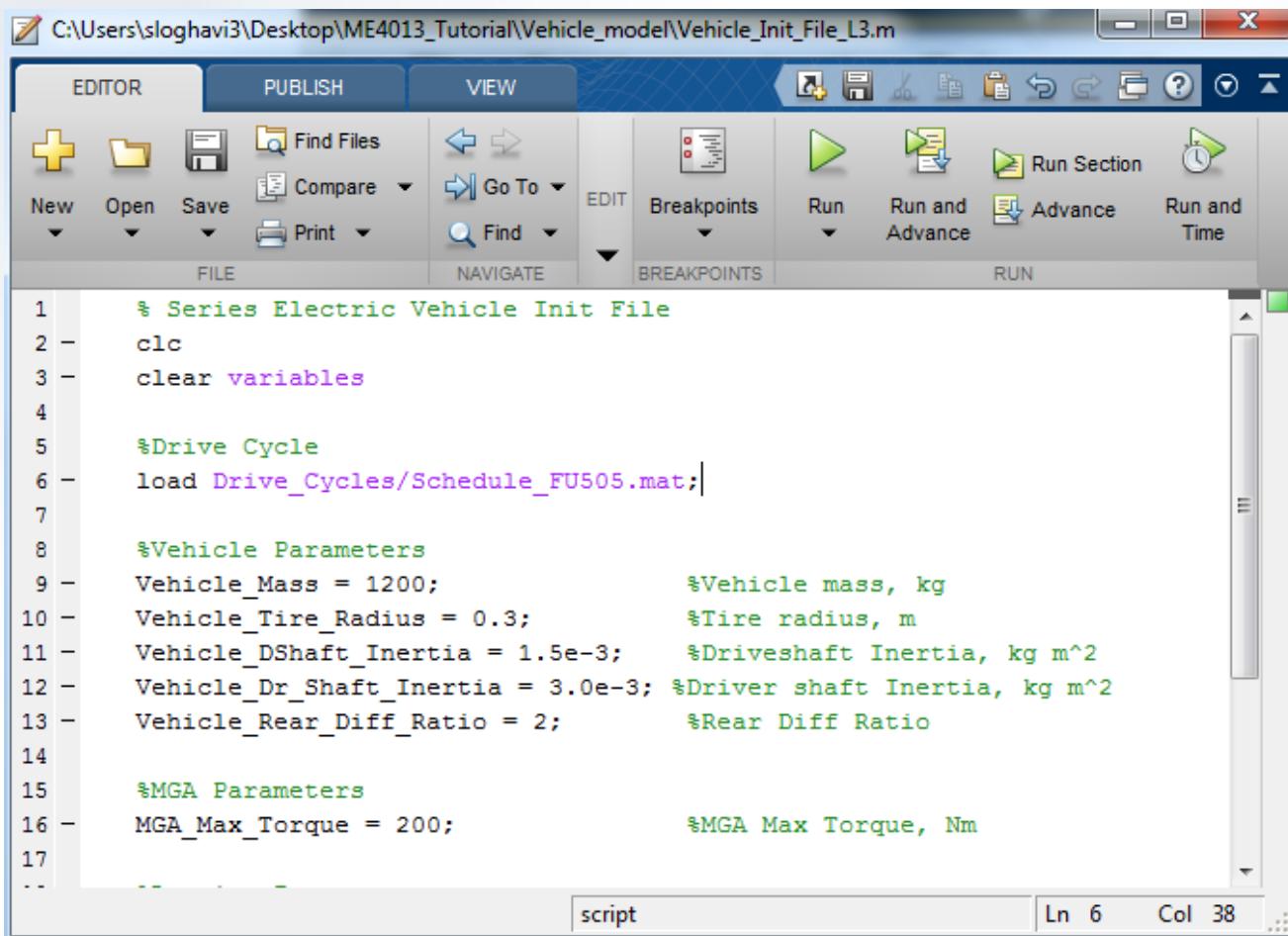
Ready 100% ode23tb

A screenshot of the MATLAB/Simulink interface showing the "Driver" subsystem. The workspace browser on the left shows the model structure: Series_Hybrid_le3_b > Driver. The main window displays a block diagram with a "Sch_Cycle" block connected to a summing junction. A signal from "Vehicle_Speed_mph" is also connected to the same summing junction. The output of this junction goes to a "Vehicle" block, which then connects to a "Driver Diagnostics" block. A "System Signals" block is also present. A "Saturation" block parameters dialog is open, showing settings for upper and lower limits, and checkboxes for linearization and zero-crossing detection.



Driver Subsystem

- The final step in the Driver subsystem is to add the desired drive cycle to the initialization file



A screenshot of a MATLAB Editor window titled "C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m". The window has tabs for "EDITOR", "PUBLISH", and "VIEW". The "EDITOR" tab is selected, showing a toolbar with icons for New, Open, Save, Find Files, Compare, Go To, Find, Breakpoints, Run, Run and Advance, Advance, and Run and Time. Below the toolbar is a menu bar with FILE, NAVIGATE, BREAKPOINTS, and RUN. The main code area contains the following MATLAB script:

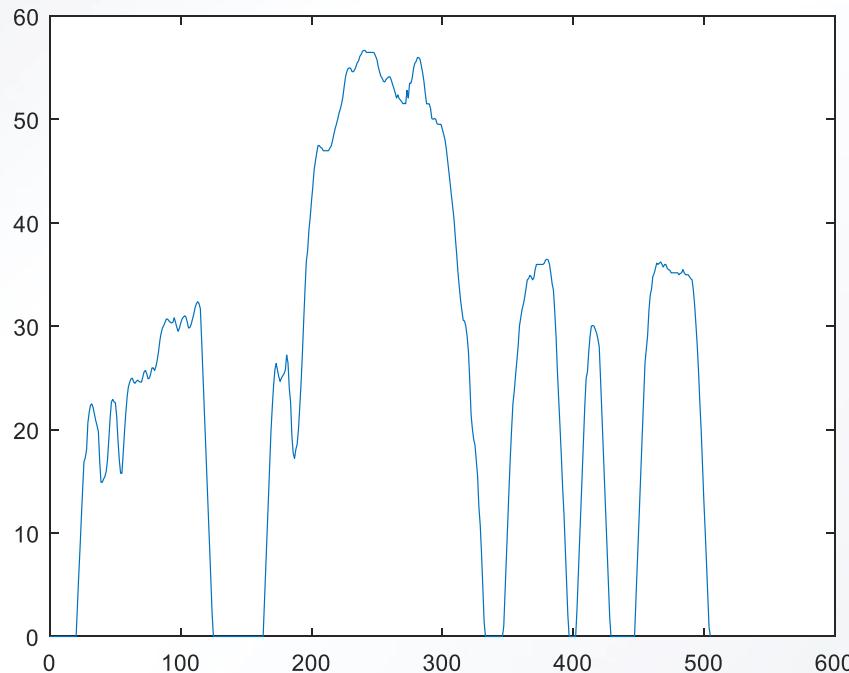
```
1 % Series Electric Vehicle Init File
2 clc
3 clear variables
4
5 %Drive Cycle
6 load Drive_Cycles/Schedule_FU505.mat;
7
8 %Vehicle Parameters
9 Vehicle_Mass = 1200; %Vehicle mass, kg
10 Vehicle_Tire_Radius = 0.3; %Tire radius, m
11 Vehicle_DShaft_Inertia = 1.5e-3; %Driveshaft Inertia, kg m^2
12 Vehicle_Dr_Shift_Inertia = 3.0e-3; %Driver shaft Inertia, kg m^2
13 Vehicle_Rear_Diff_Ratio = 2; %Rear Diff Ratio
14
15 %MGA Parameters
16 MGA_Max_Torque = 200; %MGA Max Torque, Nm
17
```

The status bar at the bottom right shows "script" in the first field, "Ln 6" in the second, and "Col 38" in the third.



Driver Subsystem

- Run the init file manually to load the drive cycle
- At the Matlab command line type
 - `plot(Sch_Cycle(:,1),Sch_Cycle(:,2))`



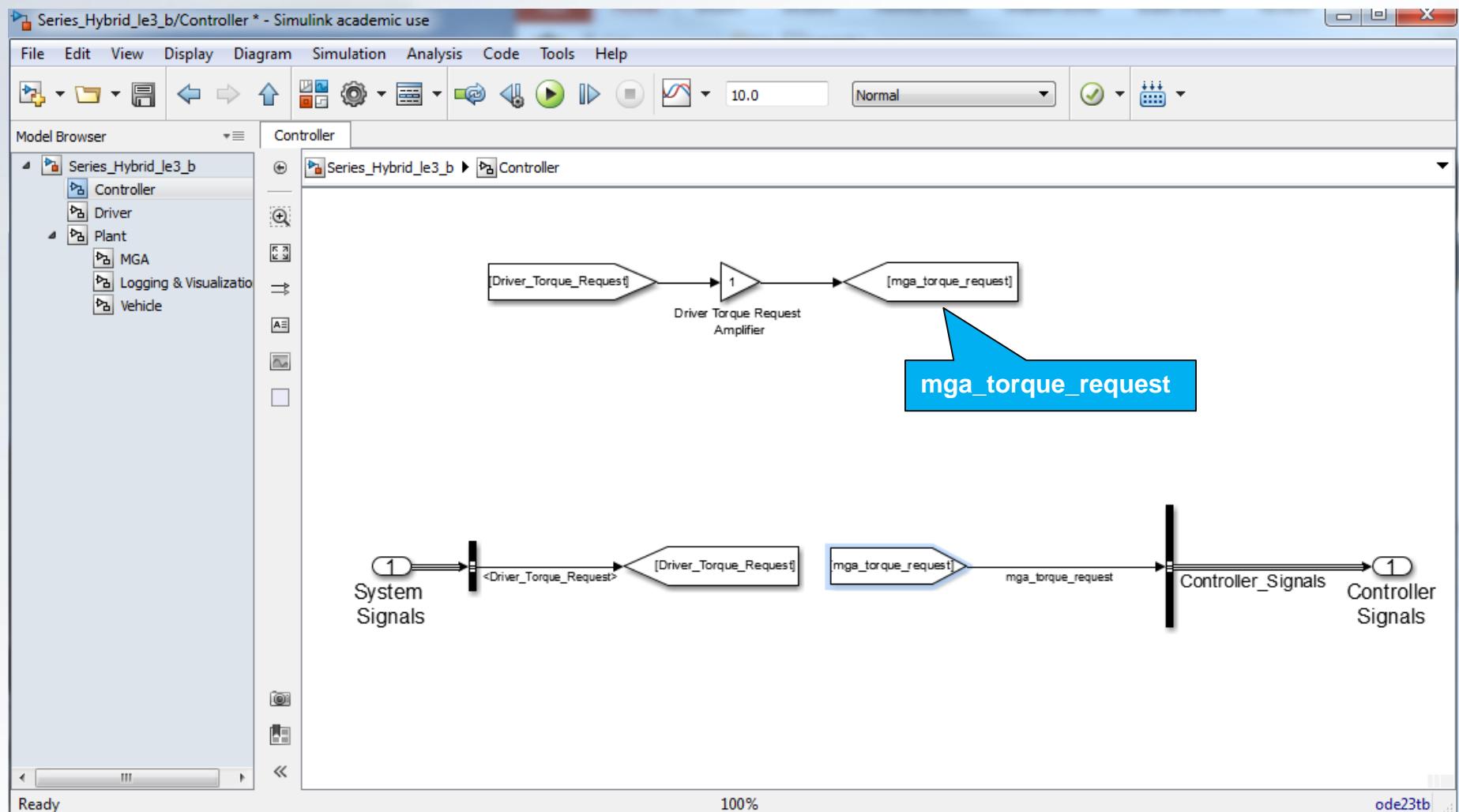


Controller Subsystem

- The Driver torque request will next be sent through the CAN bus to the Controller subsystem
- Go to the Controller subsystem, delete the Terminators and **Grounds**, and drag in a **Gain**, a **Bus Selector**, two **Goto**, and two **From** blocks
- On the **Bus Selector**, select the **Driver_Torque_Request** signal
- Wire and label as shown on the next slide



Controller Subsystem





Controller Subsystem

- The driver torque request passes through the controller where it is sent to MGA
 - Just like a real vehicle
- The **Gain** block can be modified
 - Sporty : increase request
 - Economy : decrease request
- Naming convention
 - All Control signals will be lowercase
 - End with either “request” or “enable”

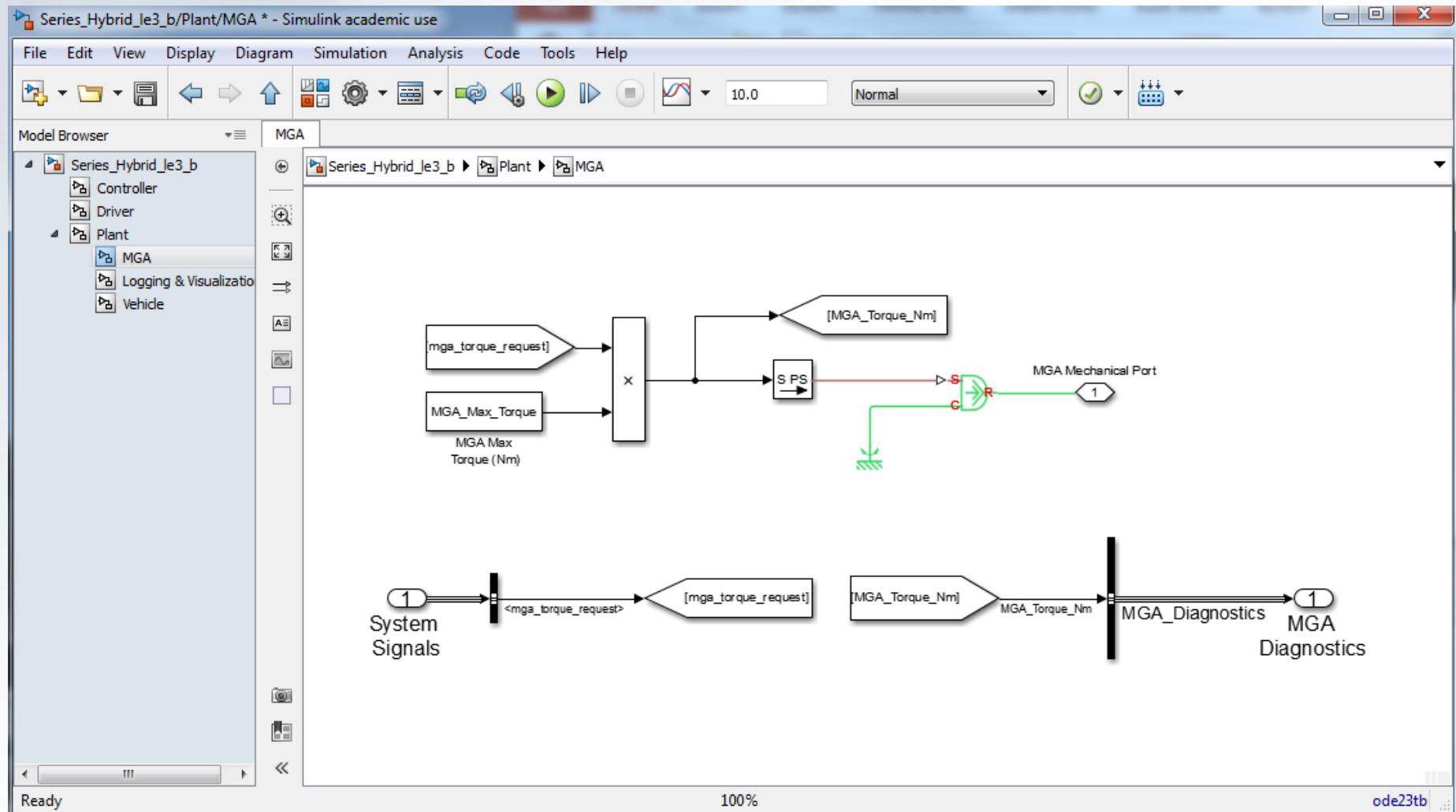


MGA Subsystem

- The last step is to pass the torque request to MGA
- Drag in a **Bus Selector**, a **Goto**, a **From**, and a **Product** block
- Delete the **Terminator**
- Select the `mga_torque_request` signal
- Wire and label as shown on the next slide



MGA Subsystem



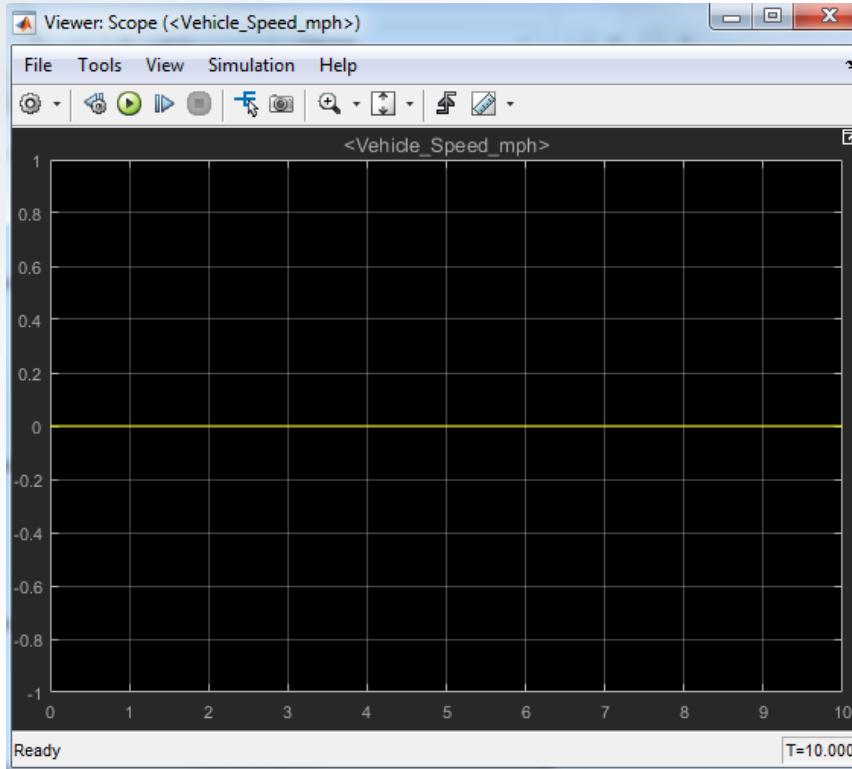


Run the simulation

- Go to the model level and Run the model

Results

- Run the model and observe the scope

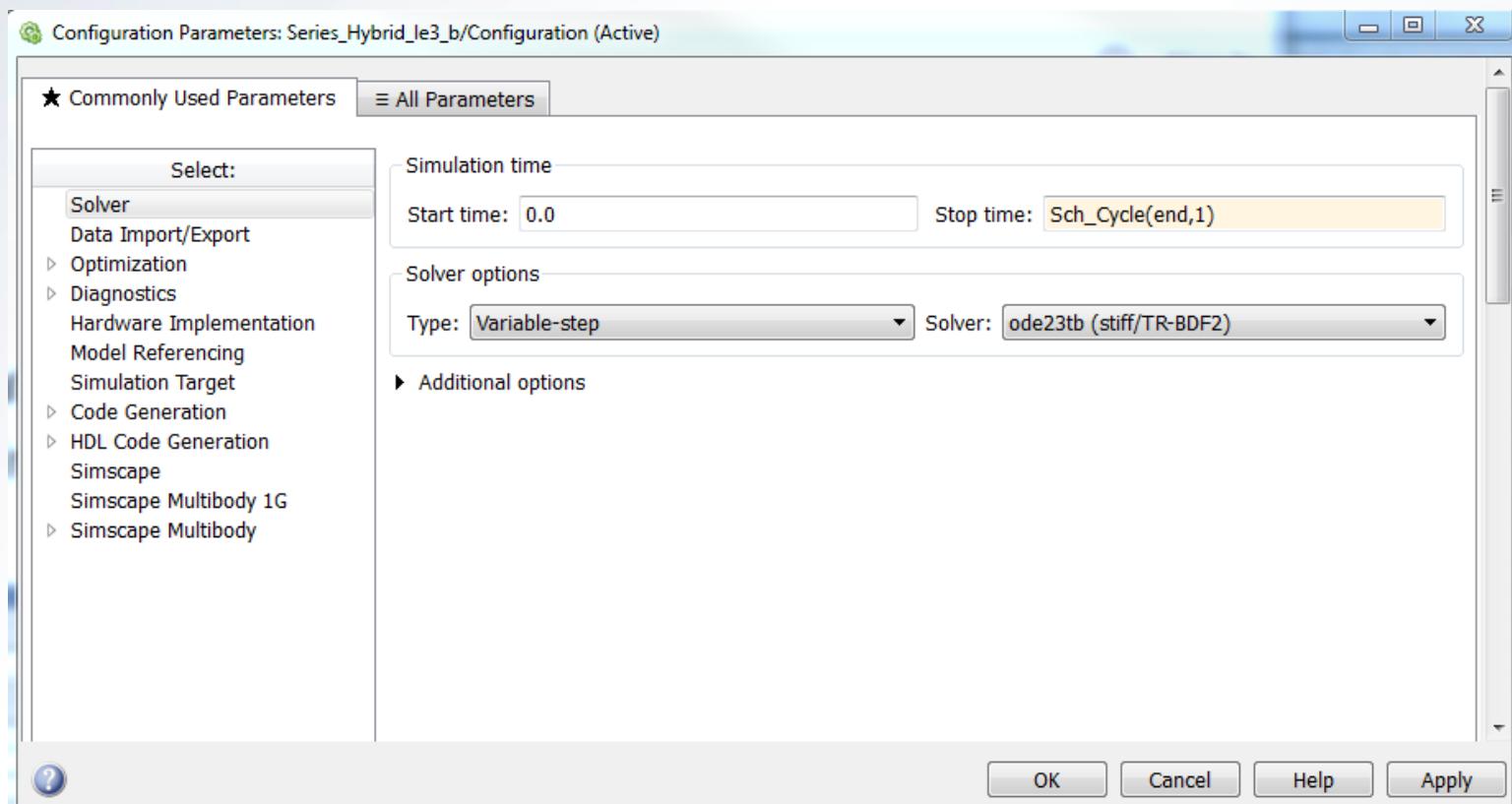


- It only ran for 10 seconds



Results

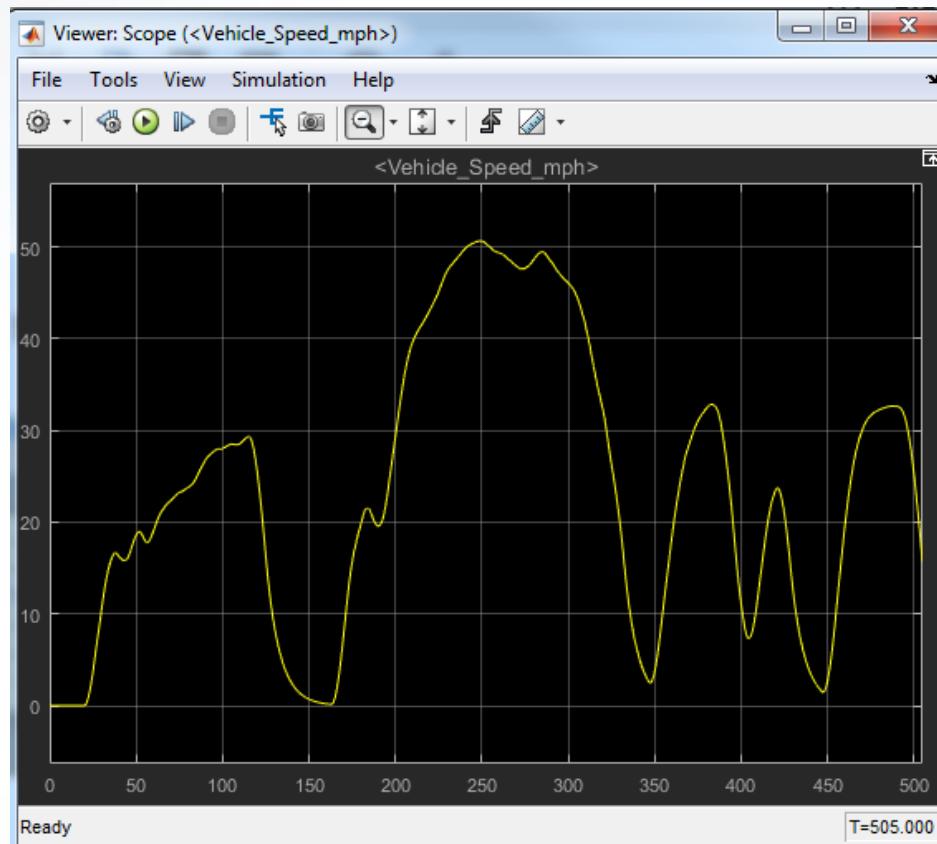
- Open the Configuration Parameters
- Change the Stop Time to
 - Sch_Cycle(end,1)





Results

- Run the simulation again
- Awesome! The vehicle is doing something!
- Note: in the Scope, go to Tools / Axes Scaling
- Select: Automatically Scale Axis Limits



- But How well?

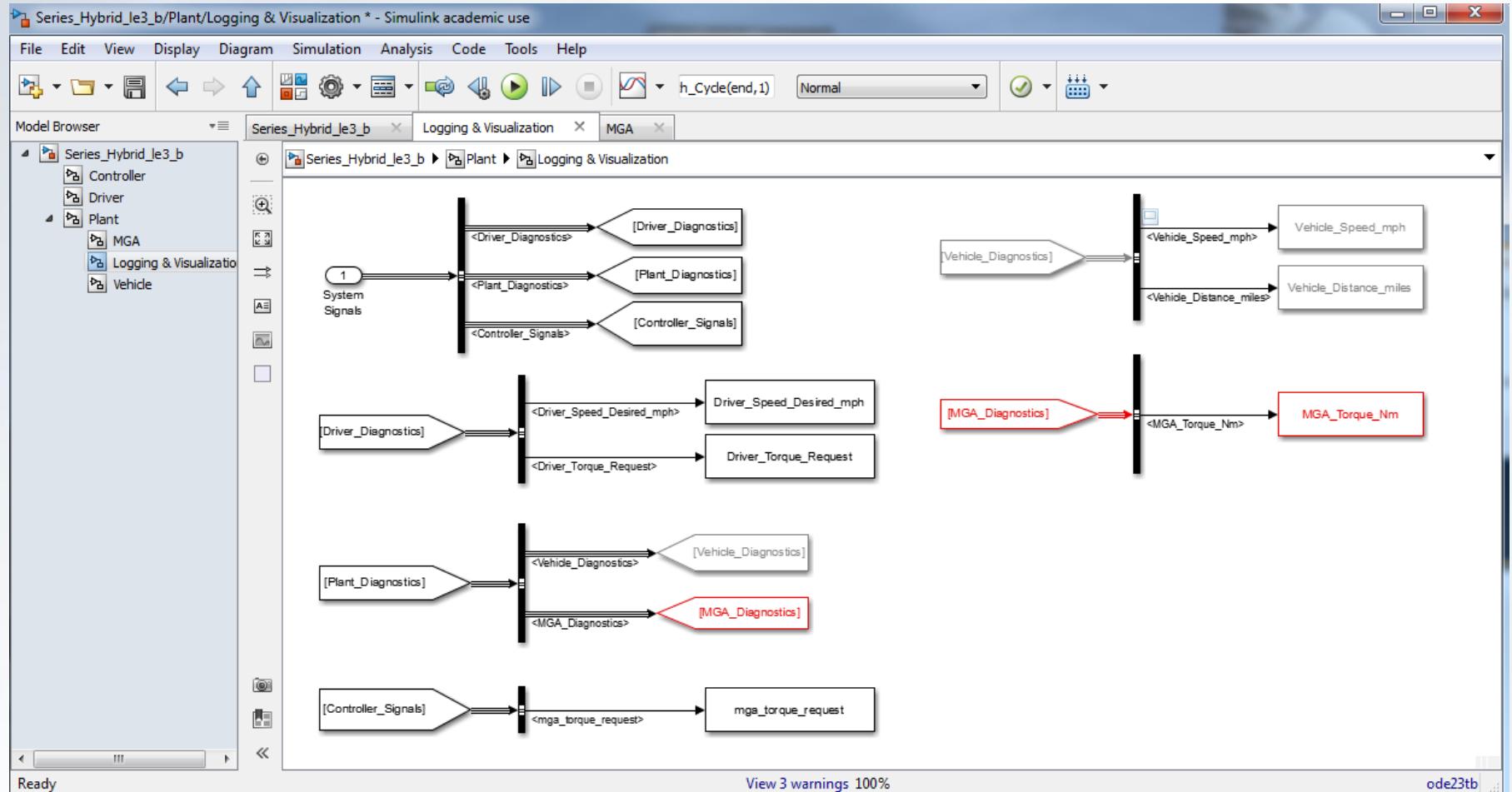


Logging & Visualization

- Go to the Logging & Visualization subsystem
- Delete the **Terminator** on the Driver_Diagnostics **From** block
- Use a **Bus Selector** to extract all Driver signals
- Drag in two **To Workspace** blocks to log the Driver Data
- Repeat for the Controller Signals



Logging & Visualization



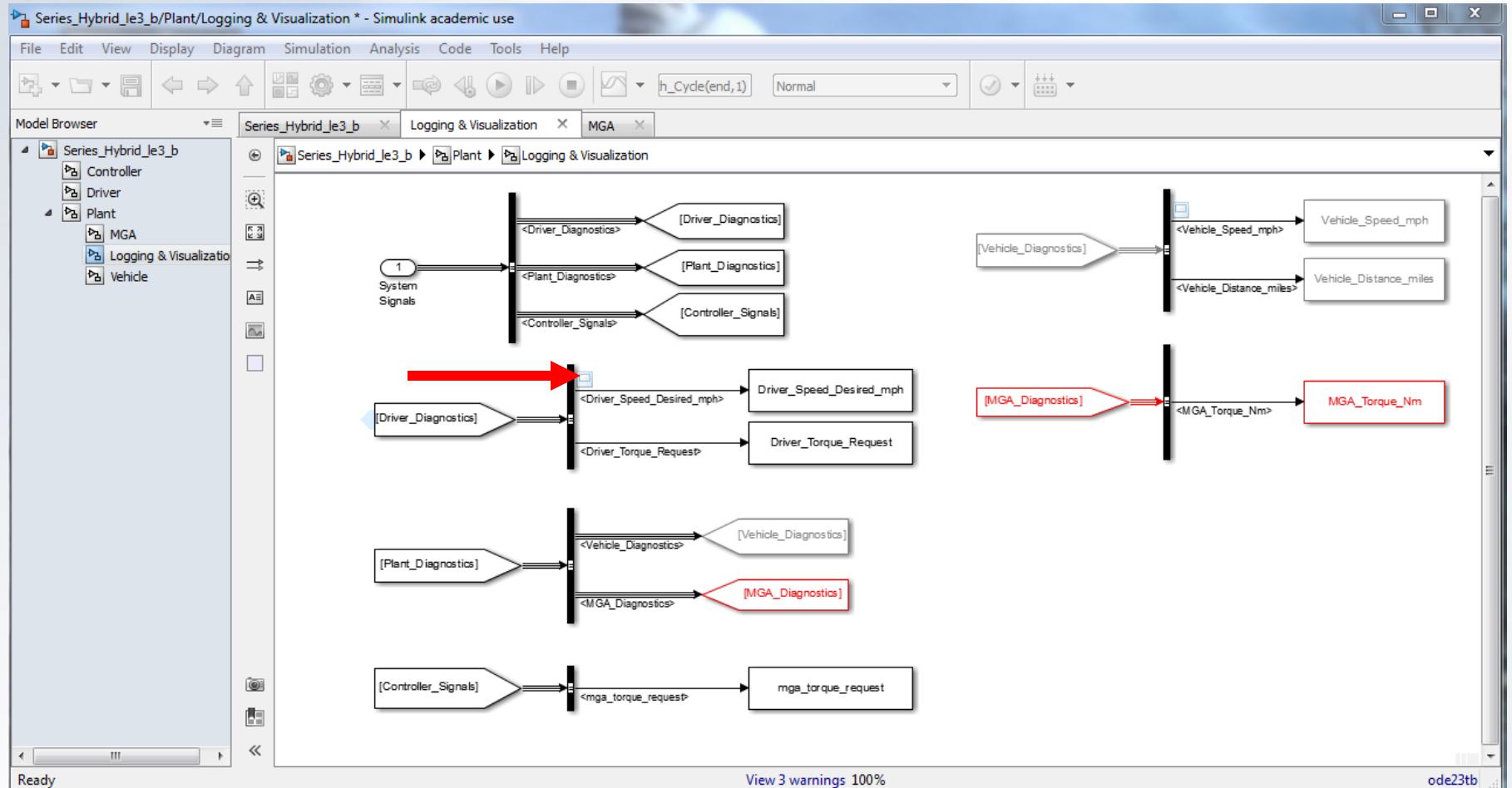


Logging & Visualization

- Right click on the Driver_Speed_Desired_mph signal
- Select
 - Connect to Existing Viewer
 - Scope
- Rerun the simulation and observe the scope

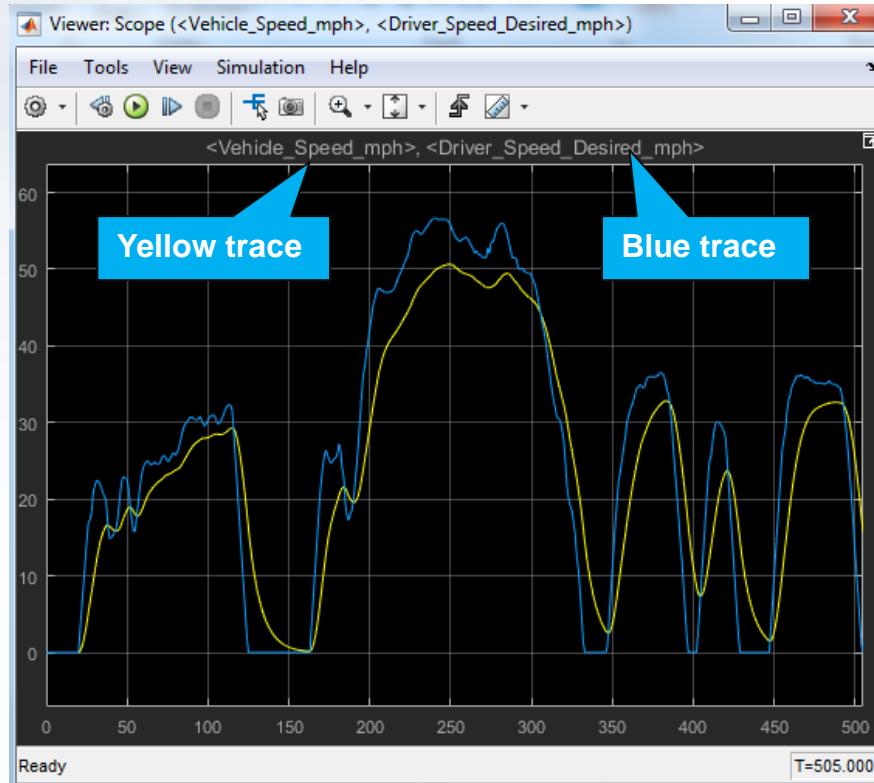


Logging & Visualization



Results

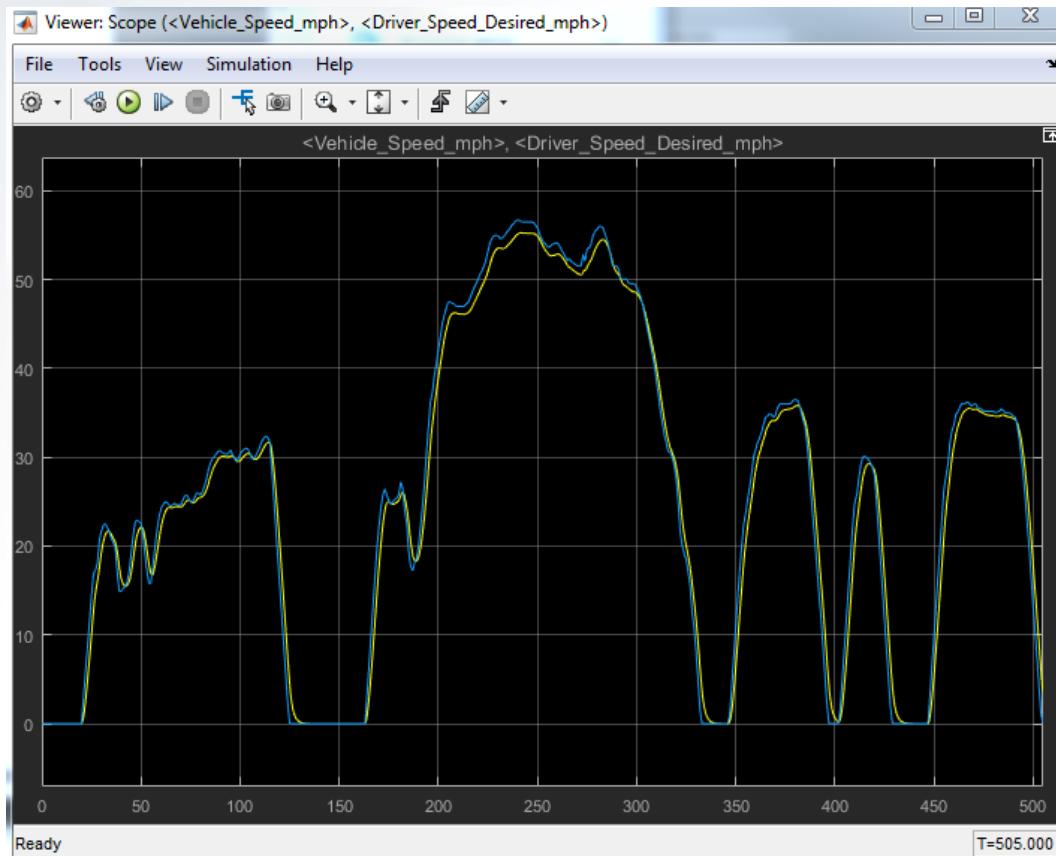
- Our vehicle does a poor job of following the drive cycle



- Increase the MGA max torque in the init file until the traces are similar

Results

- Even with 1000 Nm, the vehicle still can not follow the trace exactly

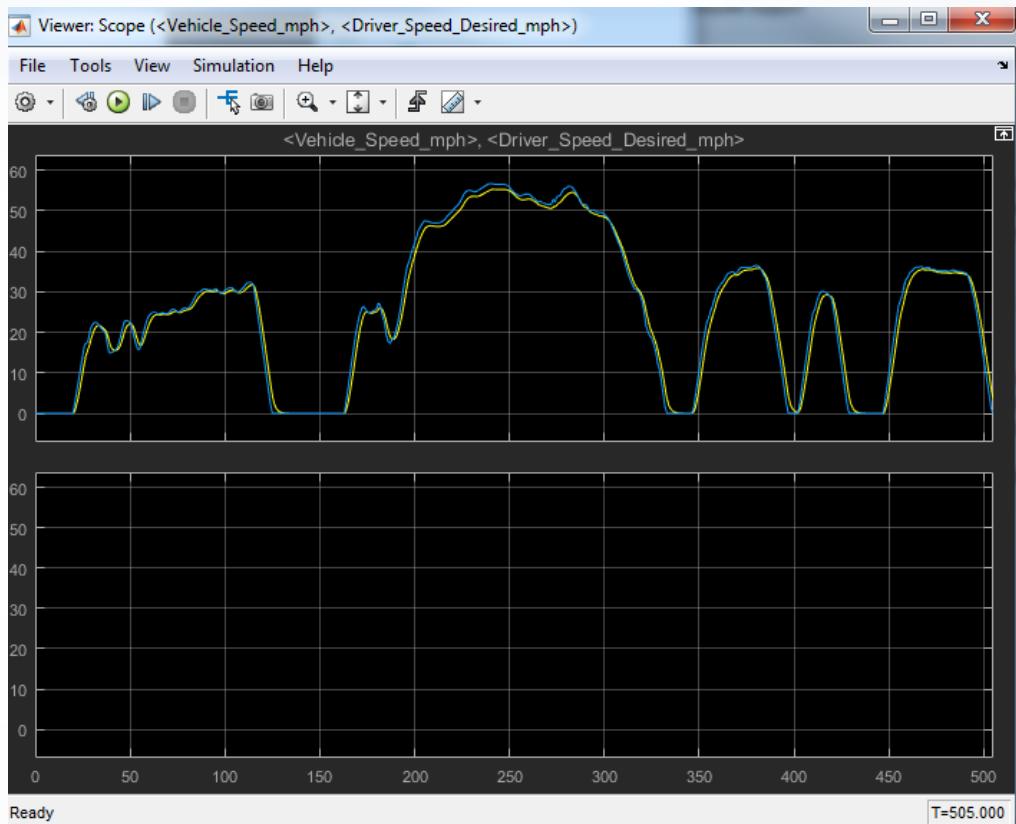


- Put the torque back to 200 Nm



Results

- Let's see what the driver is doing
- On the scope
- Select
 - [View / Layout](#)
- Choose a 2×1 layout
- Now we can add signals to both Axis 1 and Axis 2



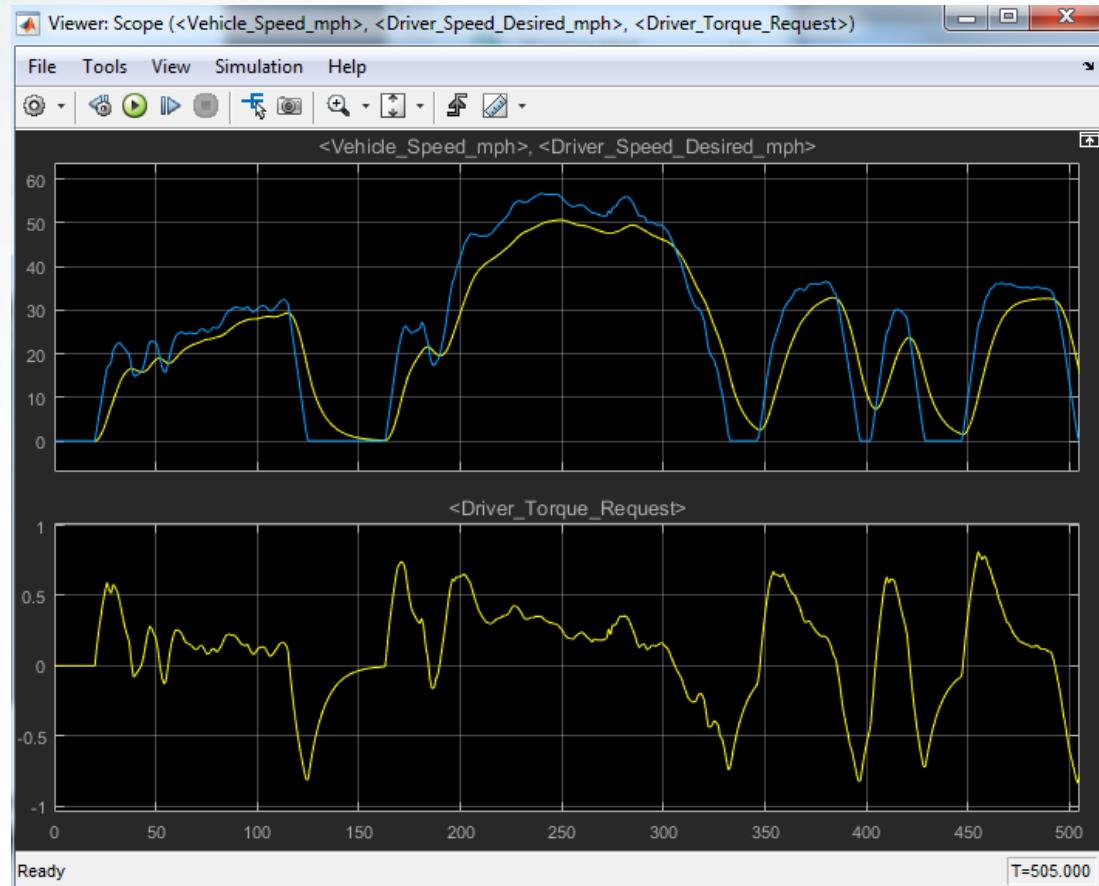


Results

- In the Logging & Visualization subsystem
 - Right click on the Driver_Torque_Request signal
 - Select
 - Connect to Viewer / Scope / Axes 2
- This will put the Driver torque request directly below the drive cycle
- Rerun the simulation

Results

- Wow – the driver torque request never goes above about 0.7



- Why?



Results

- Note that the vehicle is “chasing” a velocity and only occasionally off by more than 10 mph
- Change the error amplifier in the Driver subsystem to make the driver more aggressive
- Increasing the gain
 - Decreases the proportional error
 - Allows the system to respond faster
 - Can cause overshoot and oscillations

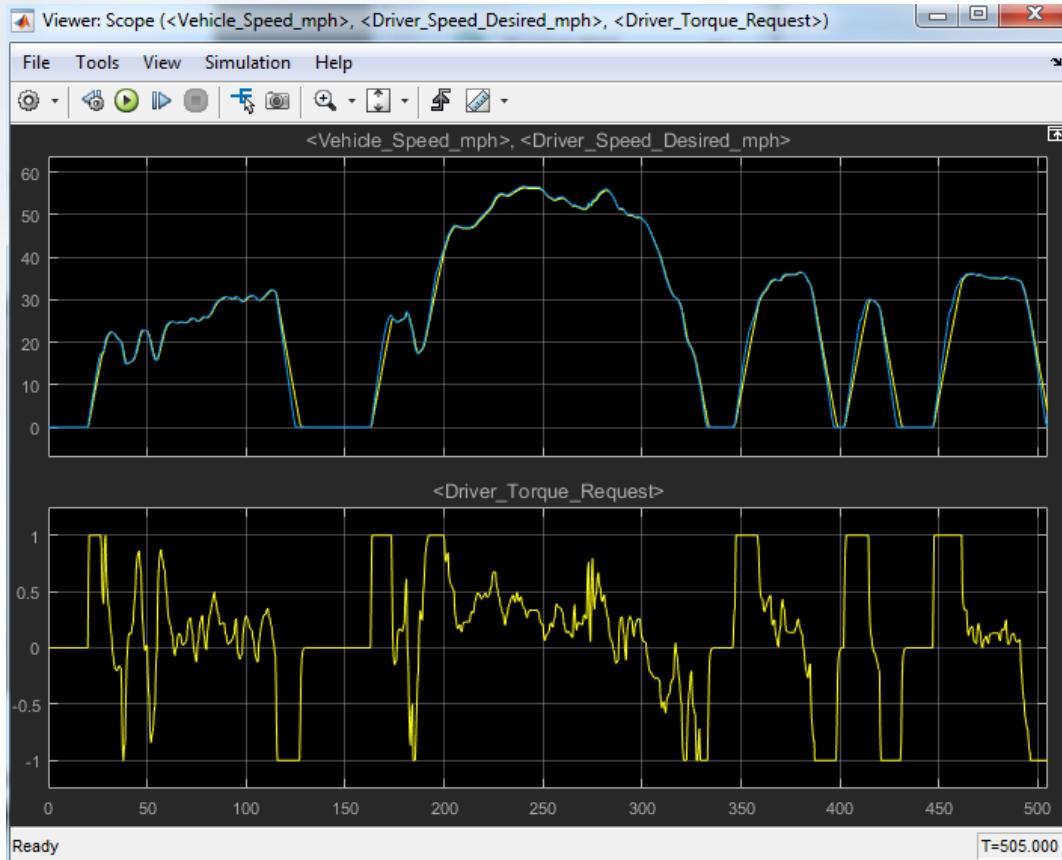


Results

- An error amplifier of 1 should work quite well

Results

- As we expected, error amplifier of 1 worked quite well for this drive cycle



- What about for a different drive cycle?



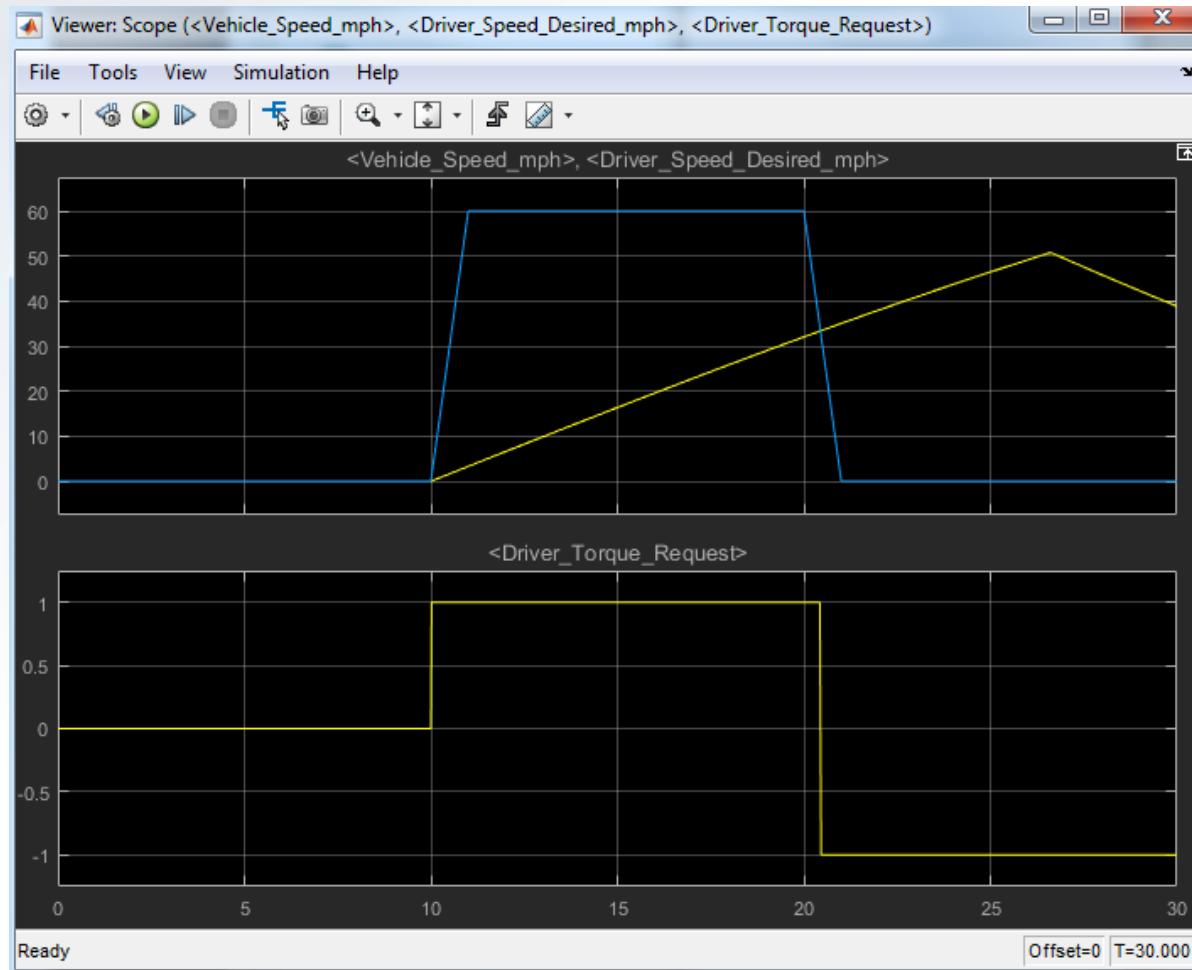
Results

- In the init file
 - Change the drive cycle to Schedule_Boston_Cab.mat
 - Increase the MGA torque to 1000 Nm
- Rerun the simulation



Results

- Definitely some overshoot



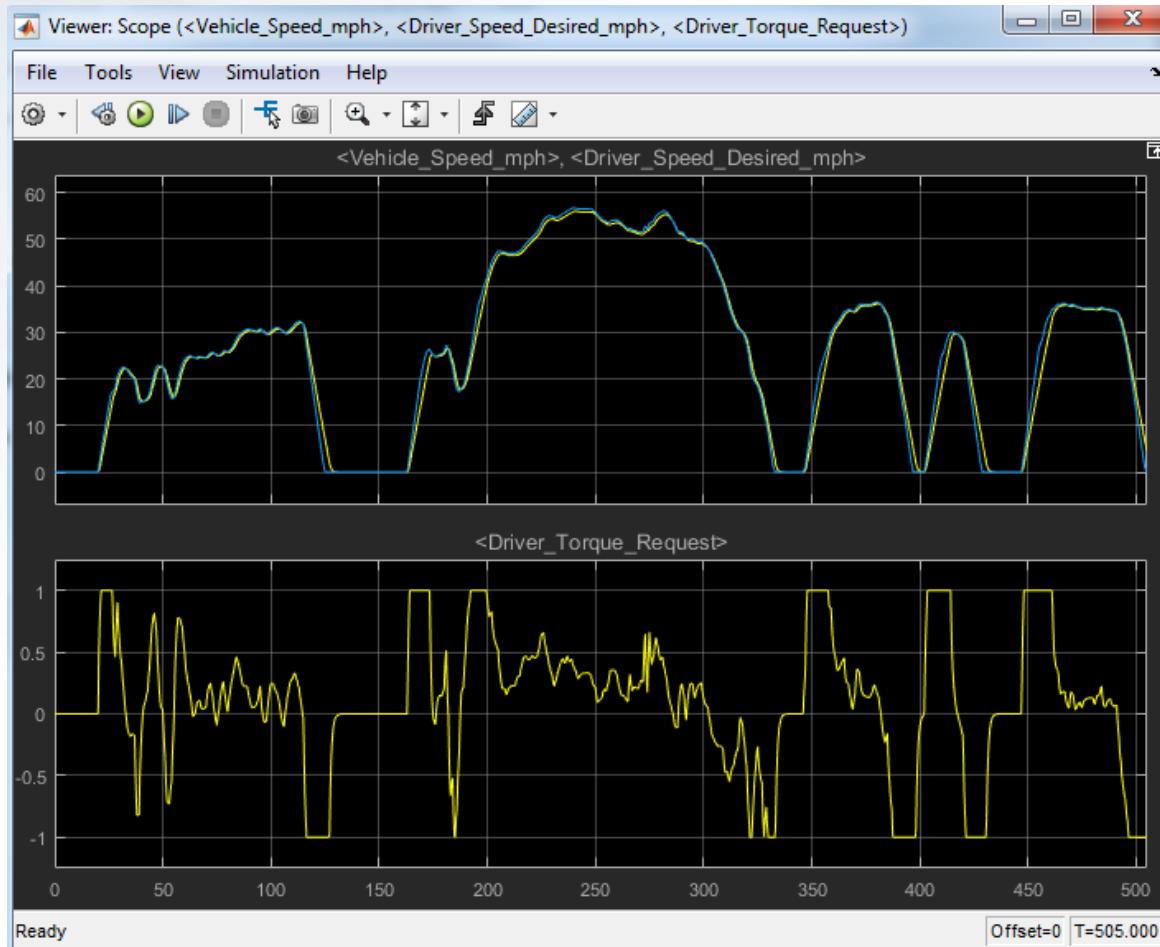


Results

- In the init file
 - Change the drive cycle back to the FU505
 - Change the MGA torque back to 200 Nm
- In the Driver subsystem
 - Change the error amplifier to 0.5
- Rerun the simulation

Results

- This provides a good balance of trace correlation and driver torque request





Initialization, Solver, and Driver

- Review
 - Created an initialization file to manage data
 - Changed to appropriate ODE solver
 - Created a logging and graphics subsystem
 - Developed driver proportional feedback loop
 - Velocity mismatch generates an error signal
 - Error signal converted to a driver torque request
 - Driver torque request goes to controller
 - Controller requests MGA torque
 - System response very sensitive to feedback gain value

A blue-toned photograph of the Georgia Tech Campanile, a historic brick building with a tall, spired tower. The word "THE TECH" is prominently displayed in white letters on the side of the tower. The image is framed by a thin yellow border.

MBSD Lecture 4

Improved MGA and Simple Battery



Outline

- Make MGA an ideal electromechanical device
- Create a preposterous battery to power MGA
- Track the battery state of charge (SOC)

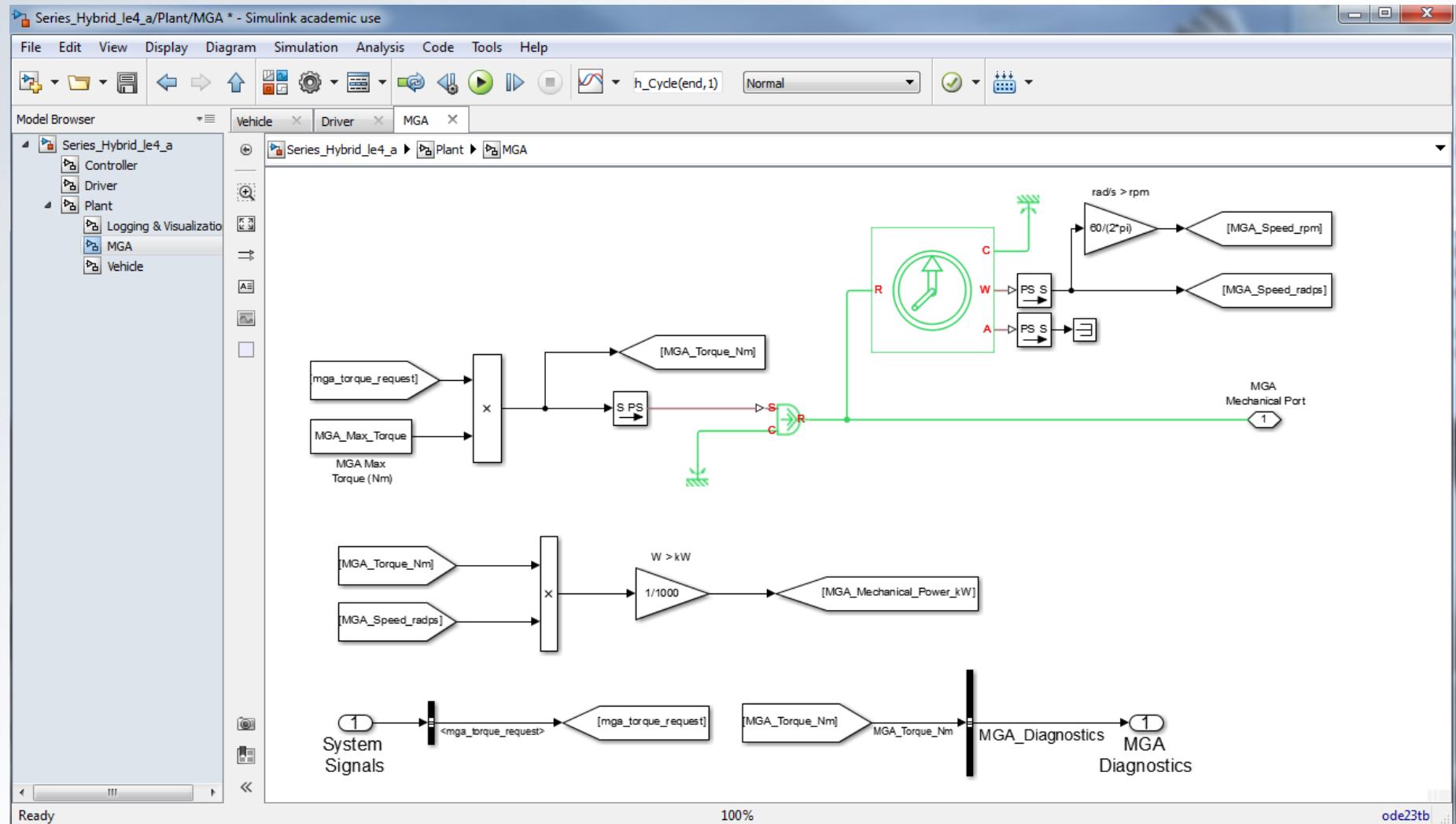


- Save your model as [Series_Hybrid_le4_a.slx](#)
- Go into the MGA subsystem
- MGA will still
 - Have a constant maximum torque
 - Have no rpm limits
 - Be modulated by the Controller
- By providing torque at a given RPM, power is transferred to the Vehicle

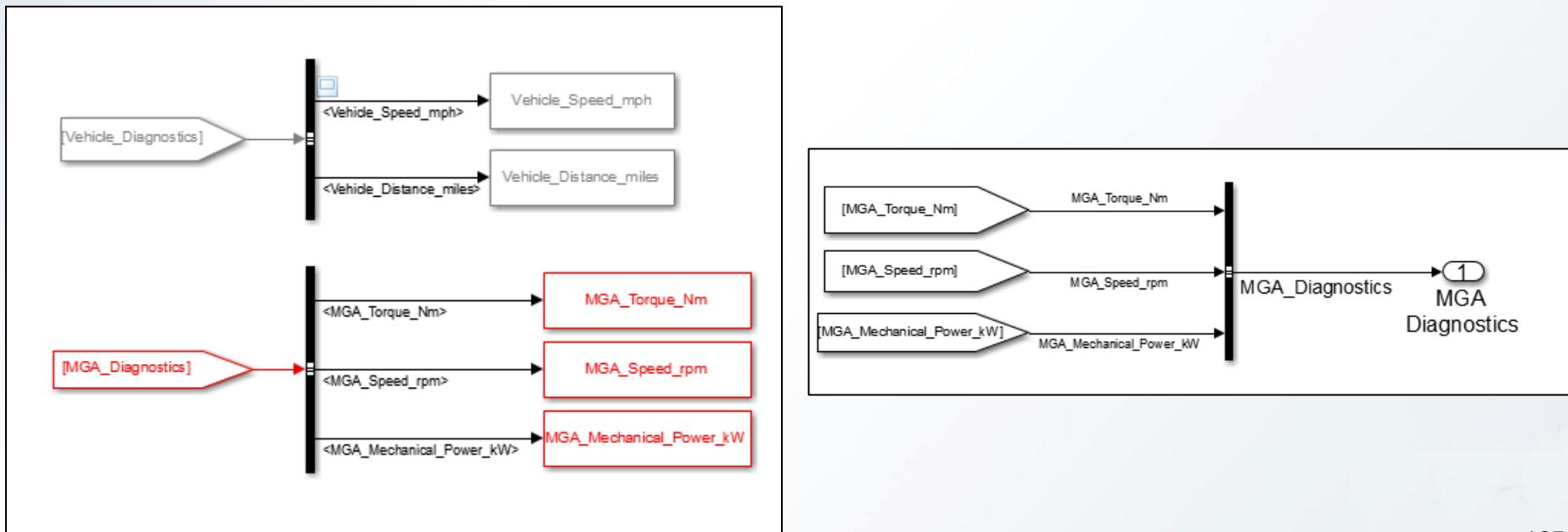


- From
 - Simscape / Foundation Library / Mechanical / Mechanical Sensors
- Drag in an **Ideal Rotational Motion Sensor** block
- From
 - Simscape / Foundation Library / Mechanical / Rotational Elements
- Drag in a **Mechanical Rotational Reference** block
- Also drag in three **Gotos**, two **Froms**, two **Gains**, two **PS-Simulink Converters**, One **Terminator** and a **Product** block
- Arrange and wire as shown on the next slide

MGA



- Add the MGA_Speed_rpm and MGA_Mechanical_Power_kW to the MGA diagnostics bus
- In the Logging & Visualization subsystem, extract and log these signals





MGA & Battery

- To make the mechanical power, the motor will need to receive electrical power from the battery
- This power will be the battery voltage times the current drawn
- Let's now add an ideal battery to the Plant
- Drag in a **Subsystem** and rename it Battery
- Make the background orange



Battery

- The Battery will receive the system signals and broadcast its diagnostics
 - Delete the wire connecting the **In1** and **Out1** blocks and rename them **System Signals** and **Battery Diagnostics**
 - Terminate the **System Signals**
 - Drag in a **Bus Creator**, connect it to the **Battery Diagnostics**, and ground the inputs

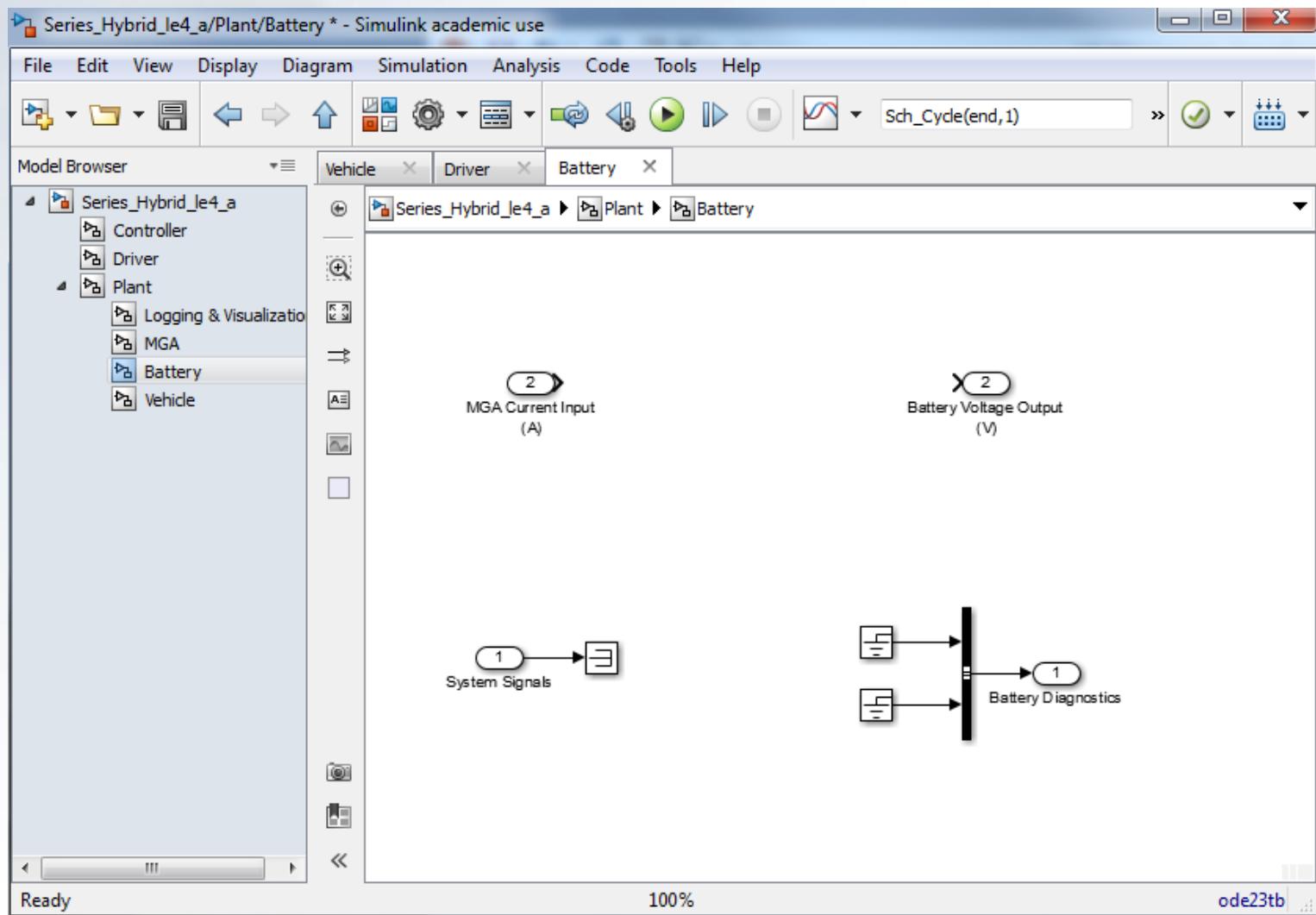


Battery

- Electrically, the battery will
 - Tell MGA its voltage
 - Receive a current demand from MGA
- From
 - Simulink / Commonly Used Blocks
- Drag in an **In1** and an **Out1** block
- Rename them
 - MGA Current Input (A)
 - Battery Voltage Output (V)



Battery





Battery

- For now, the battery voltage will be constant and defined in the init file
- Drag in a **Constant** block
 - Rename it **Battery Voltage (V)**
 - Give it a variable name of **Battery_Voltage**
 - Connect it to the **Battery Voltage Output**
- Update the init file for the battery voltage to be 336V



Battery

Series_Hybrid_le4_a/Plant/Battery * - Simulink academic use

File Edit View Diagram Simulation Analysis Code Tools Help

Model Browser Vehicle Driver Battery

Series_Hybrid_le4_a
Controller
Driver
Plant
Logging & Visualization
MGA

Sch_Cycle(end,1)

MGA Current Input (A) → Battery_Voltage (V) → Battery Voltage Output (V)

C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m

EDITOR PUBLISH VIEW

New Open Save Find Files Compare Go To Print Find Breakpoints Run Run Section Run and Advance Run and Time

```
14 %MGA Parameters
15 MGA_Max_Torque = 200; %MGA Max Torque, Nm
16 -
17 %Battery Parameters
18 Battery_Voltage = 336; %Battery Voltage, V
19 -
20 -
21 -
22 %Logging Parameters
```

script Ln 19 Col 56 ode23tb

Ready

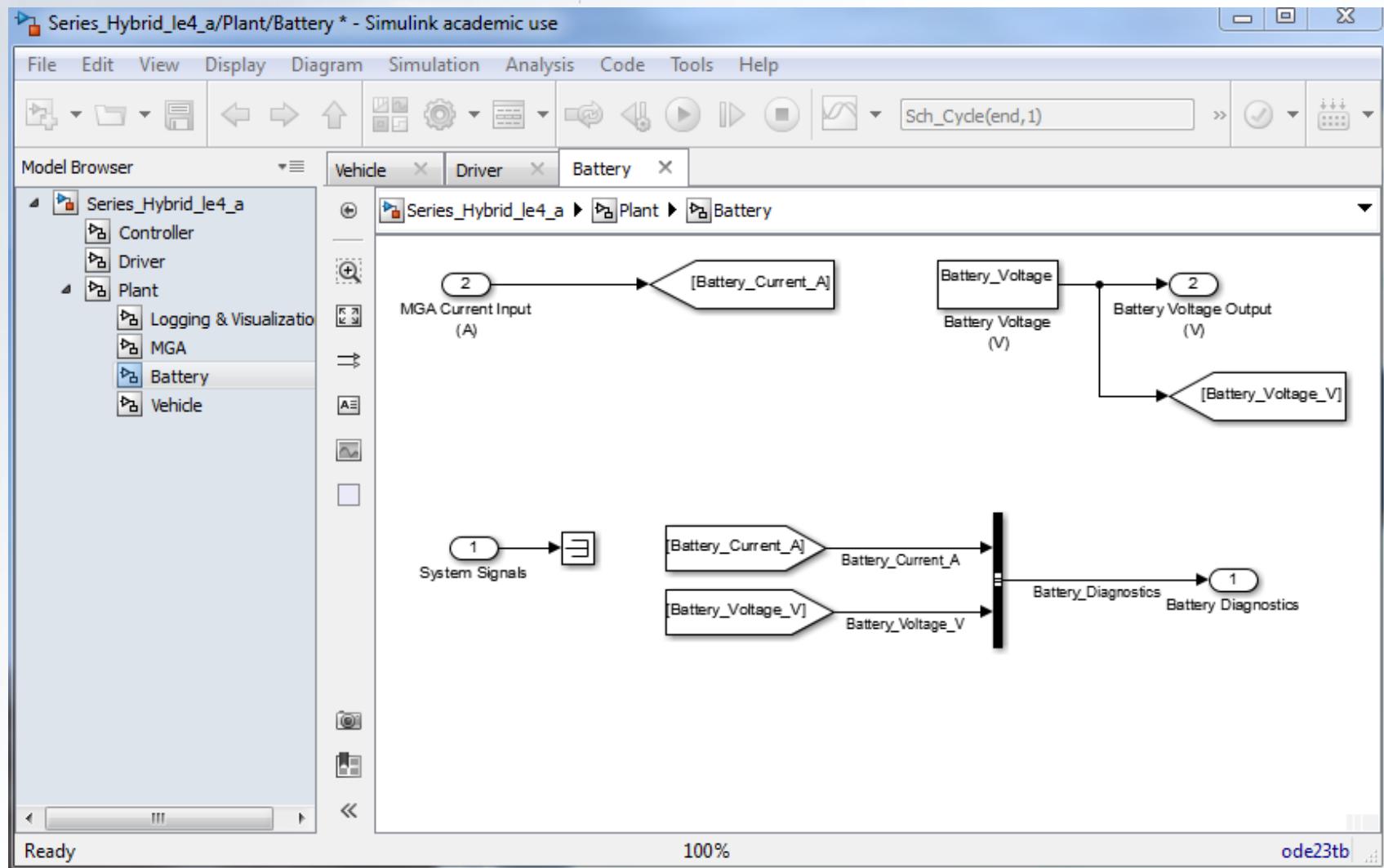


Battery

- Our magic battery can provide any amount of current required by the motor
- Put the battery current and the battery voltage on the Battery Diagnostics bus



Battery

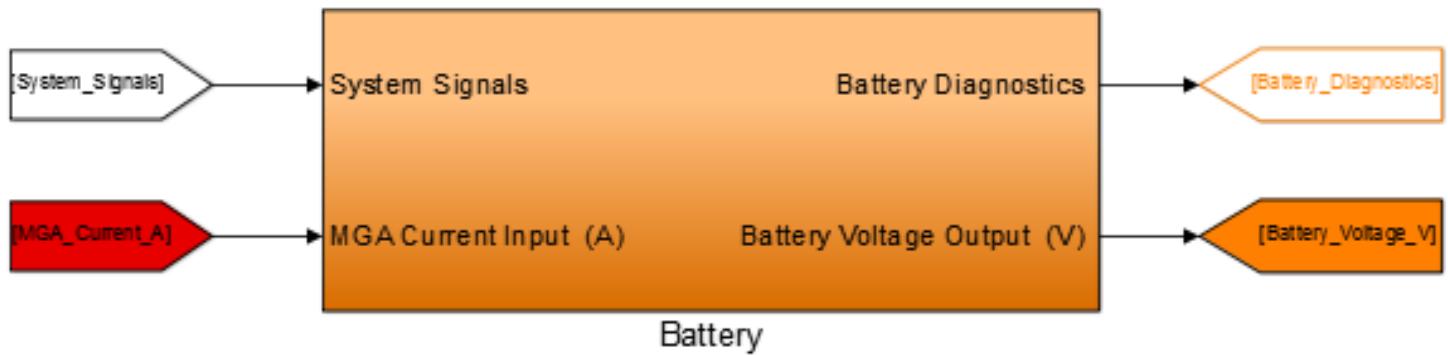
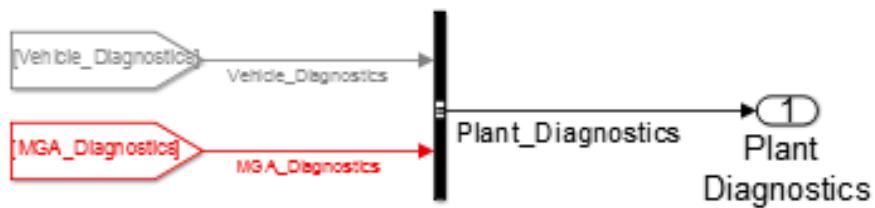




Battery

- Outstanding, we now have a constant voltage power source with unlimited amperage and capacity
- Return to the Plant level of the model
 - Make the Battery subsystem larger
 - Drag in three **From** and two **Goto** blocks
 - Wire and label as shown on the next slide
- Convention
 - Physical signals get colored background
 - CAN signals get colored foreground

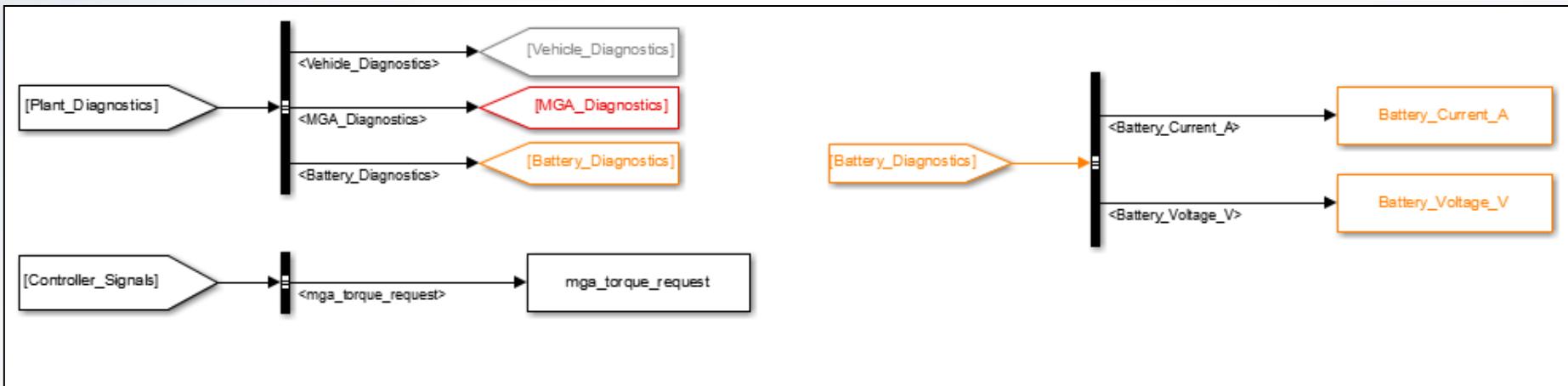
Battery





Battery

- Place the Battery Diagnostics onto the Plant CAN bus
- Extract the Battery signals in the Logging & Visualization subsystem

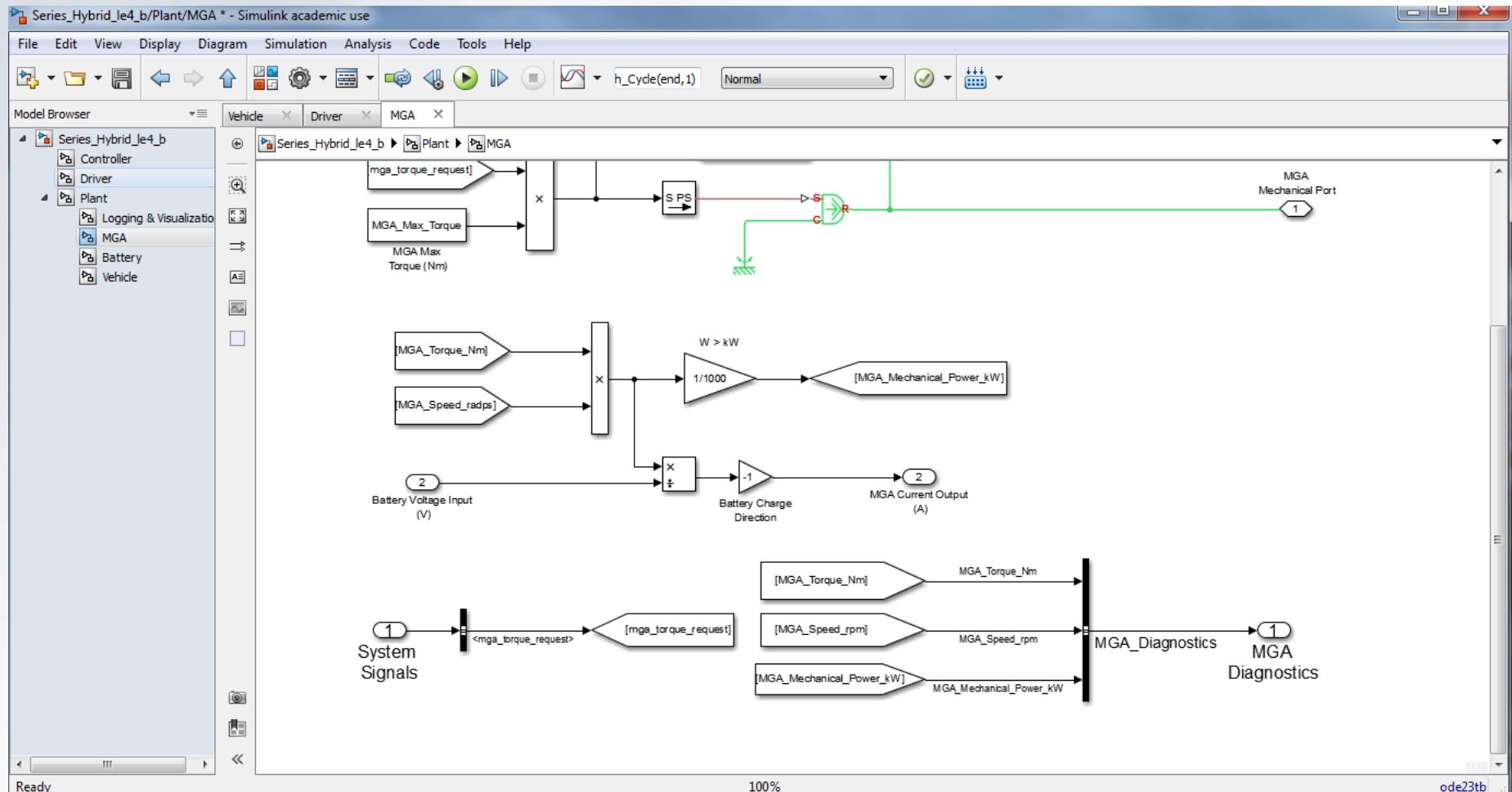




- Save the model as **Series_Hybrid_le4_b.slx**
- In the MGA subsystem, drag in an **In1** and **Out1** block
- Rename them
 - **Battery Voltage Input (V)**
 - **MGA Current Output (A)**
- For now, the electrical power from the battery will equal the mechanical power from the motor



- Thus the current required will be the mechanical power divided by the battery voltage
- From
 - [Simulink / Math Operations](#)
- Drag in a **Divide** block
- When MGA is drawing current, it will come out of the battery
- Drag in a Gain block and give it a value of -1

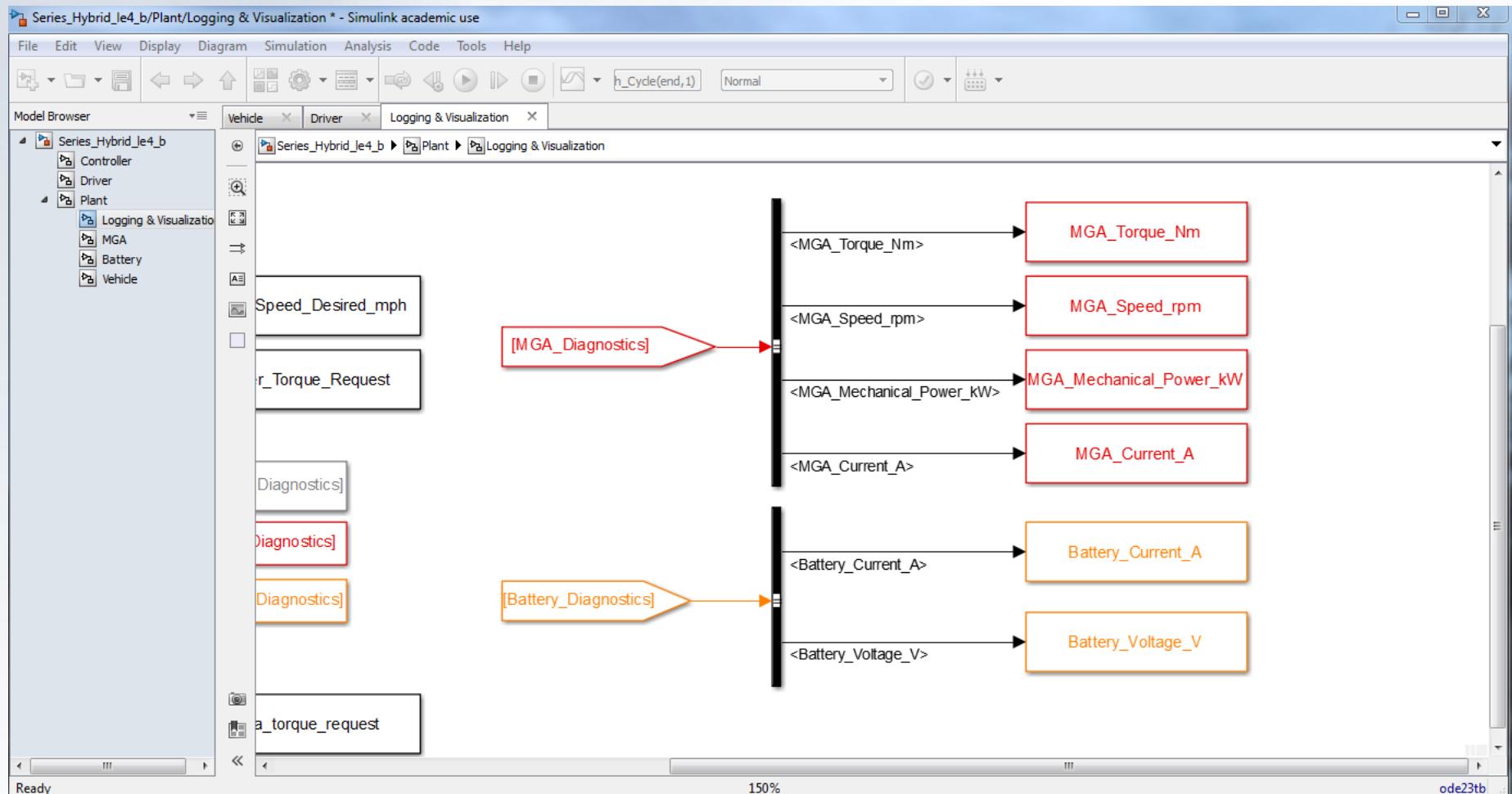




- Place the MGA Current signal onto the diagnostics bus
 - Label it **MGA_Current_A**
- Extract the signal in the Logging & Visualization subsystem

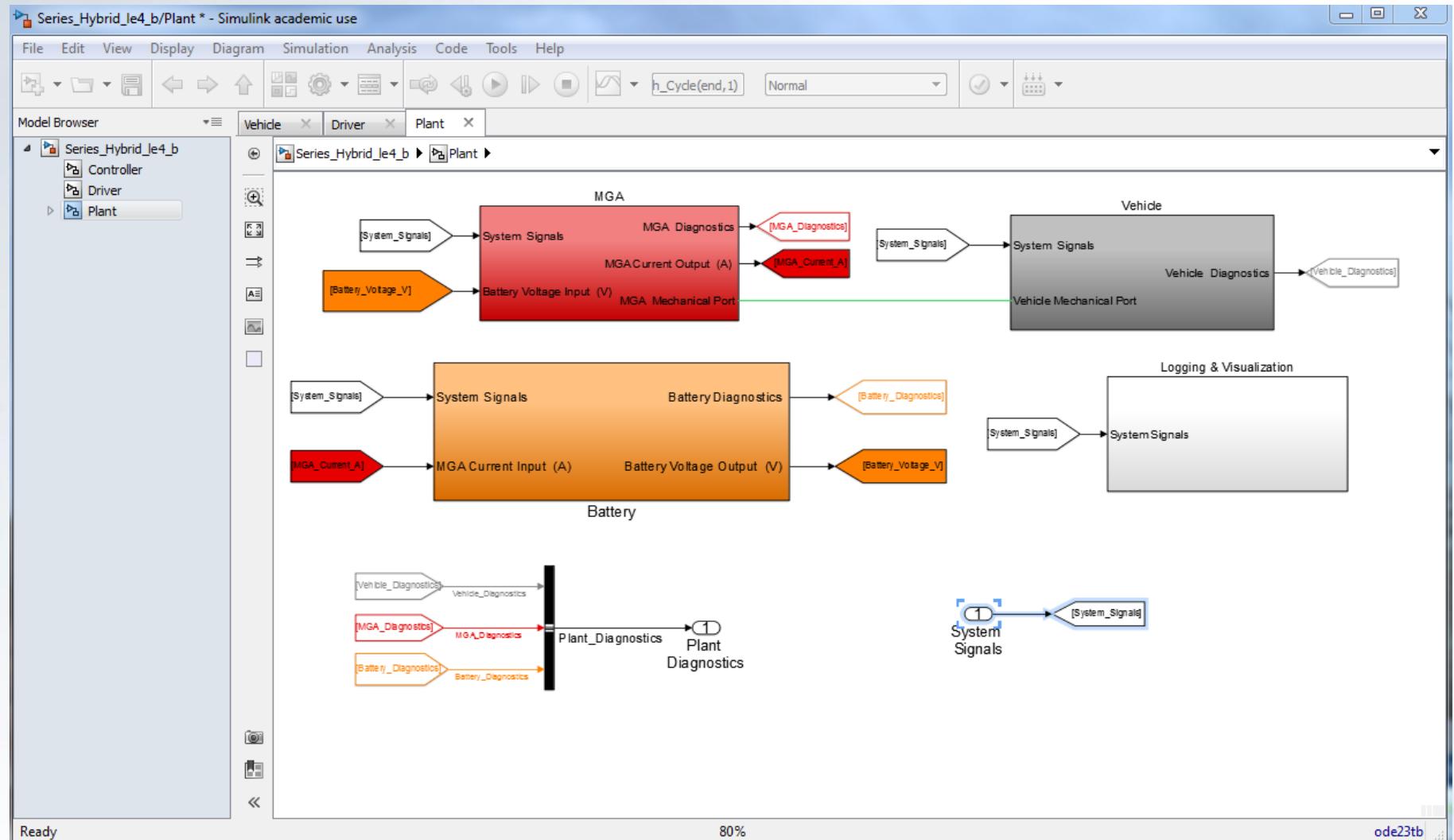


MGA / Logging & Visualization





- Return to the Plant level of the model
- Make the MGA subsystem larger
- Use a **Goto** and a **From** block to connect MGA to the Battery
 - Color accordingly



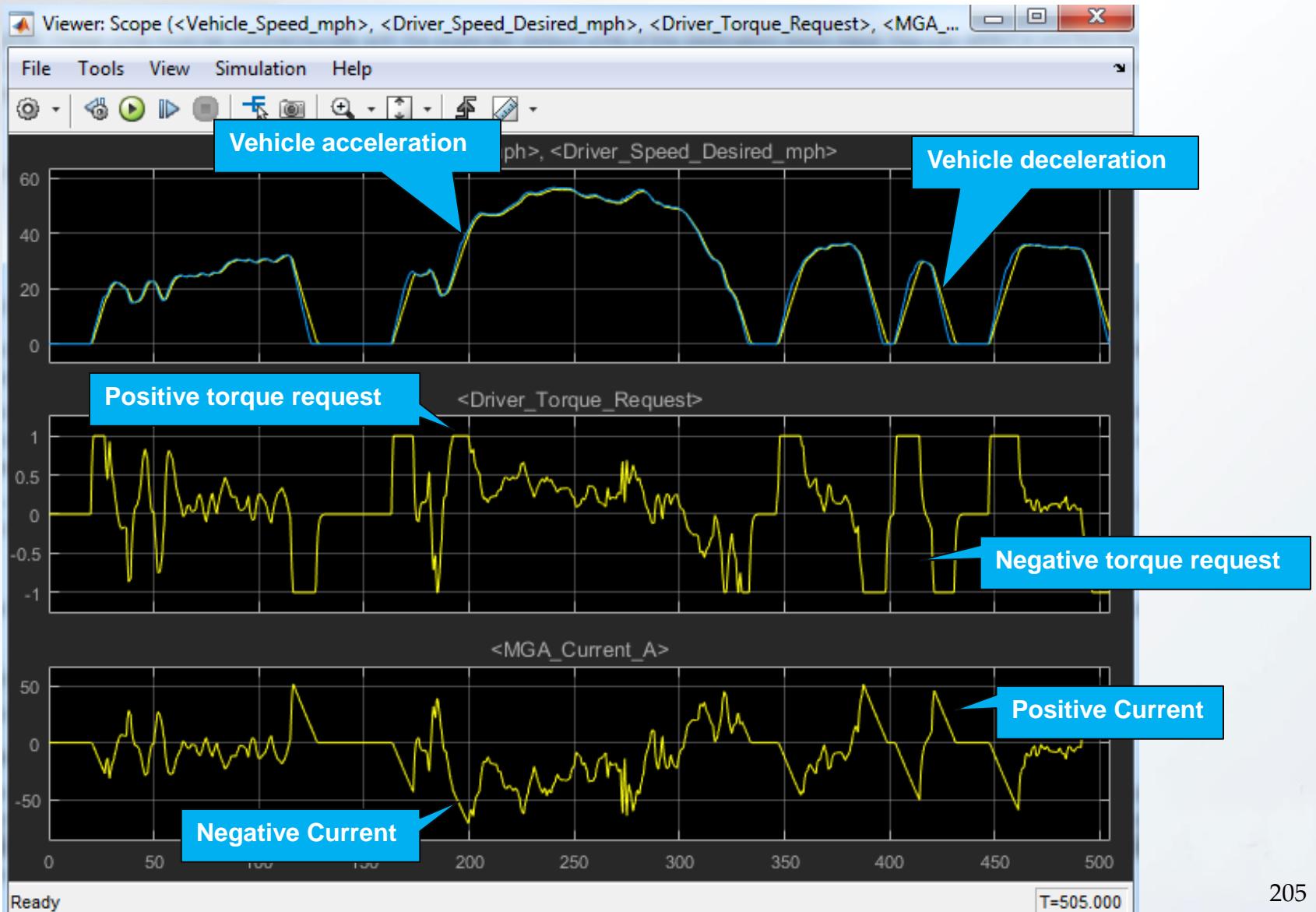


Logging & Visualization

- Let's now observe the MGA current as it follows the FU505 drive cycle
- Go to the Logging & Visualization subsystem
 - Add a third axis to the Scope
 - Place the MGA_Current_A signal on the third axis
 - Run the simulation



Results





Battery SOC

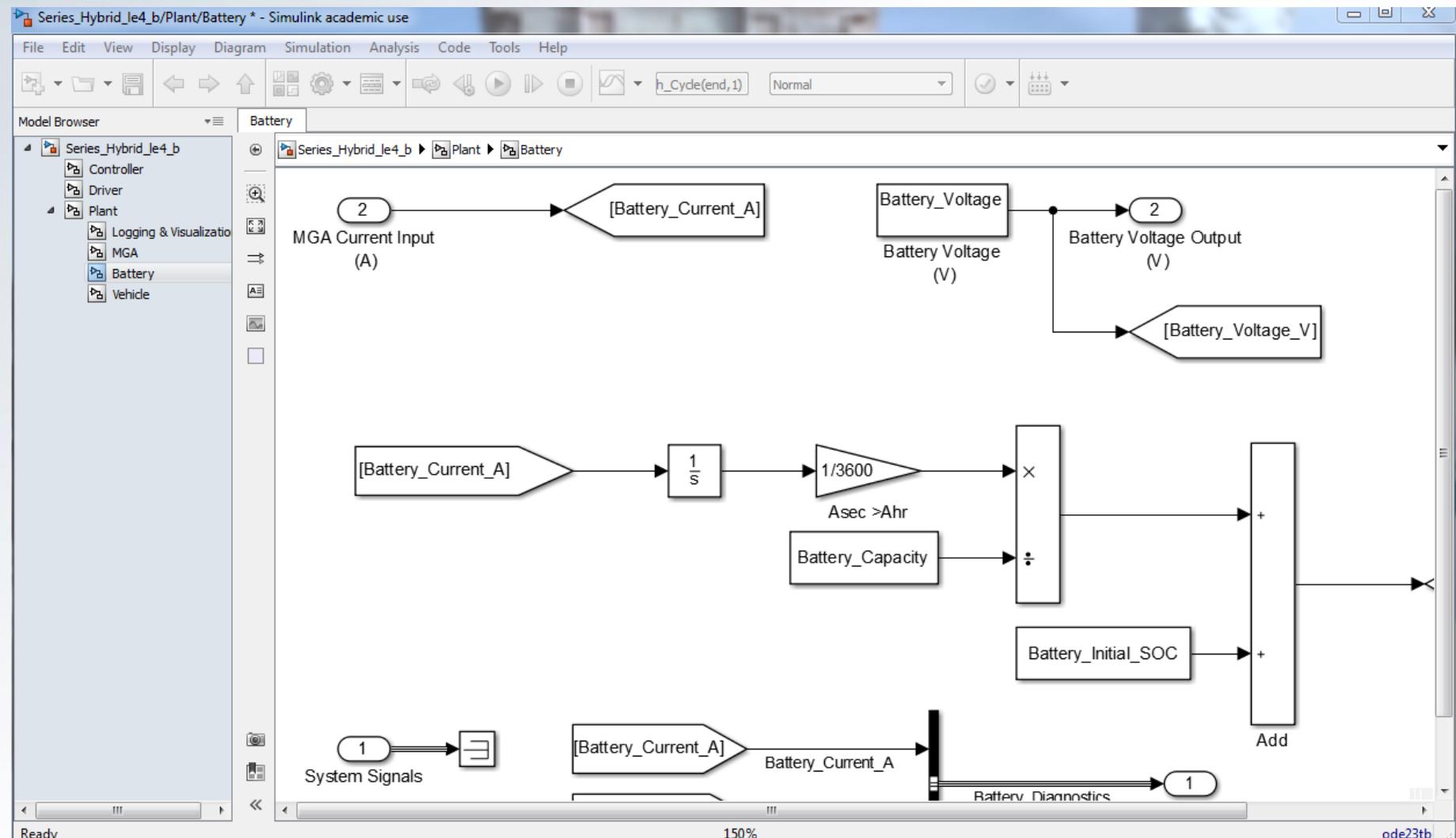
- Let's add battery charge to our model
- Battery Capacity is in units of Amp hours
 - The amount of charge transferred by a constant current of one amp for one hour
 - Thus we can integrate the battery current to find out how much charge has been removed
- This value can be non-dimensionalized by dividing by the maximum capacity of the battery to get the State of Charge (SOC)
 - 1 fully charged
 - 0 fully discharged

Battery SOC





Battery SOC



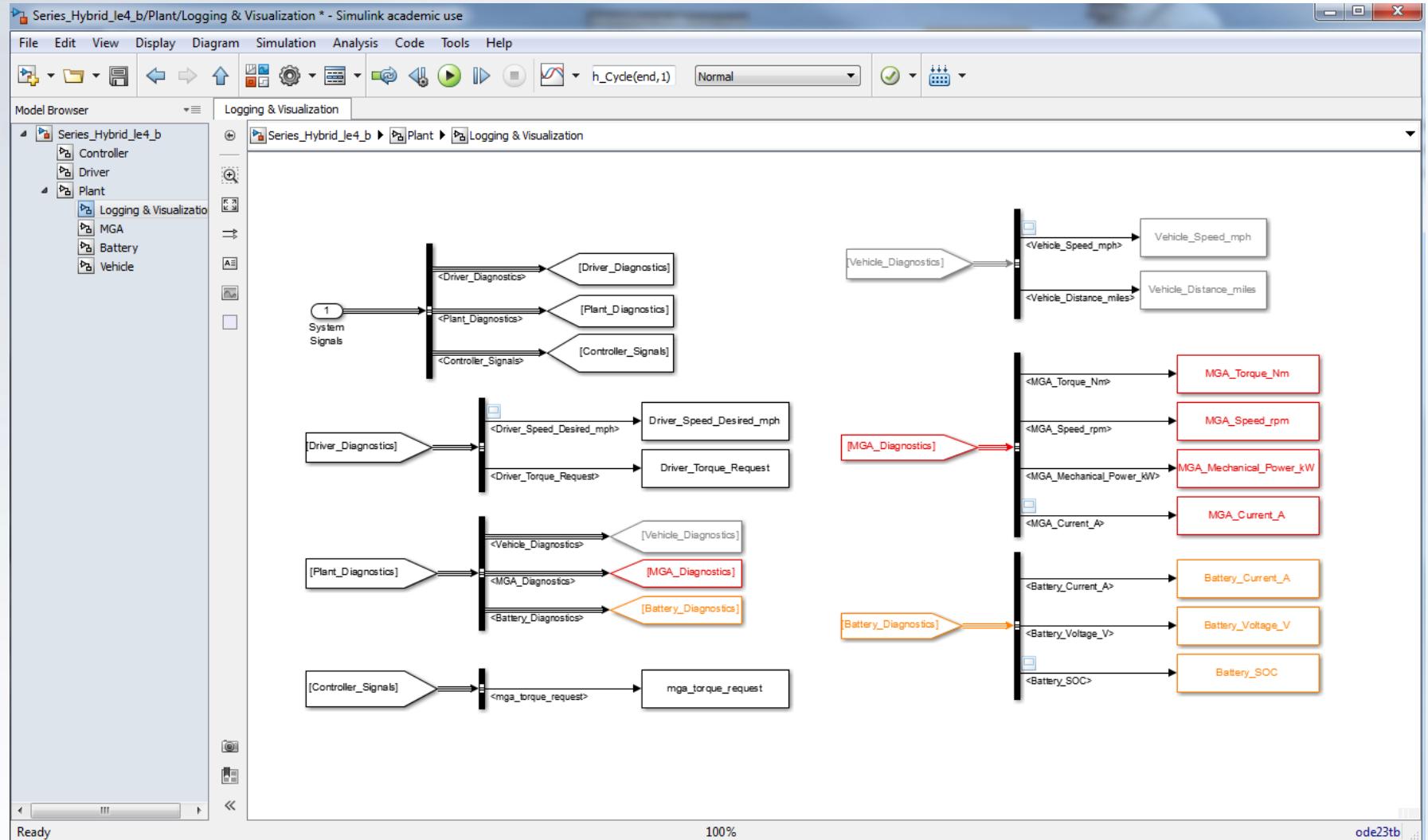


Battery SOC

- Add the Battery_SOC signal to the Battery Diagnostics bus
- Extract the signal in the Logging & Visualization subsystem
- Right click on the Driver_Torque_Request signal
 - Select Disconnect Viewer
 - Select Scope
 - Select Axes 2
- Add the SOC signal to Axes 2



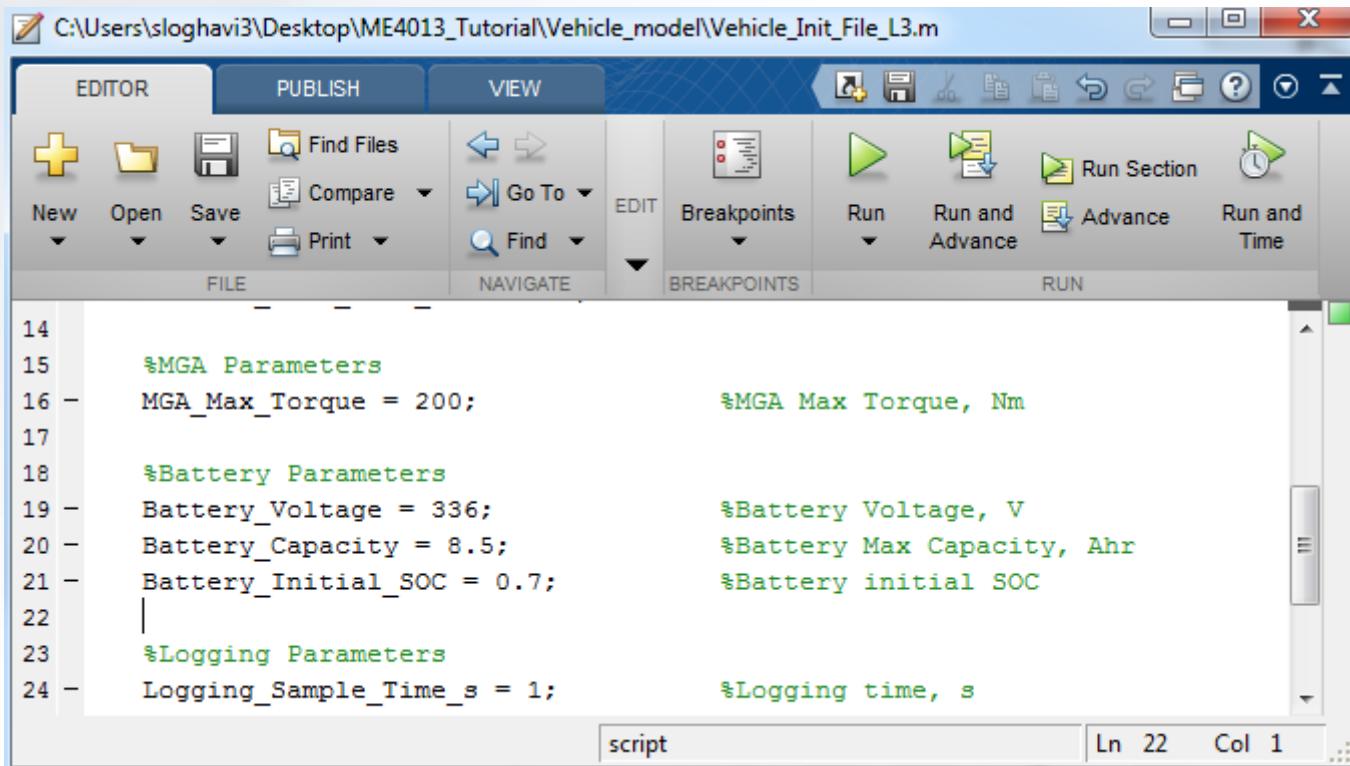
Logging & Visualization





Initialization

- The last step is to add the two new battery parameters to the init file



A screenshot of the MATLAB Editor window titled "Vehicle_Init_File_L3.m". The window has tabs for "EDITOR", "PUBLISH", and "VIEW". The "EDITOR" tab is selected. The toolbar includes buttons for New, Open, Save, Find Files, Compare, Print, Breakpoints, Run, Run and Advance, Advance, and Run and Time. The code in the editor is as follows:

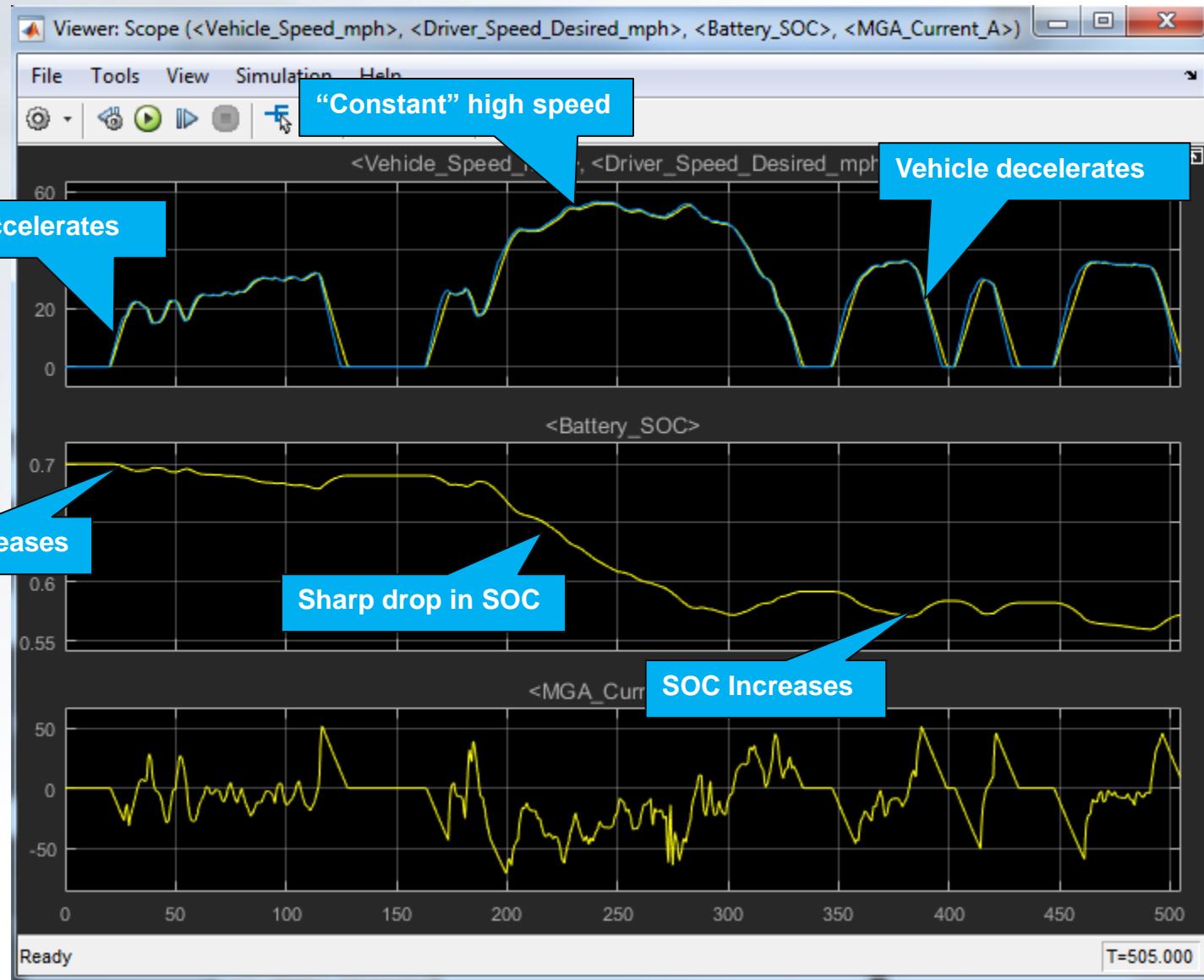
```
14
15 %MGA Parameters
16 - MGA_Max_Torque = 200; %MGA Max Torque, Nm
17
18 %Battery Parameters
19 - Battery_Voltage = 336; %Battery Voltage, V
20 - Battery_Capacity = 8.5; %Battery Max Capacity, Ahr
21 - Battery_Initial_SOC = 0.7; %Battery initial SOC
22 |
23 %Logging Parameters
24 - Logging_Sample_Time_s = 1; %Logging time, s
```

The status bar at the bottom shows "script" in the first field, "Ln 22 Col 1" in the second field, and a zoom icon in the third field.

- Run the Simulation



Results





Review

- Developed an electromechanical model of MGA
 - 100% efficient
 - Constant torque capable
 - No rpm limits
- Developed an electrical battery model
 - 100% efficient
 - Constant voltage capable
 - No current limits
 - SOC ready

A blue-toned photograph of the Georgia Tech Campanile, a historic brick building with a tall, spired tower. The word "THE TECH" is prominently displayed in white letters on the side of the tower. The image is framed by a thin yellow border.

MBSD Lecture 5

Simple Charging logic



Outline

- Simple Battery charging
- Simple GenSet charge logic



Battery SOC

- While having our vehicle follow a drive cycle, the Battery SOC decreased
- Eventually, the Battery SOC will get too low
 - Permanent damage
 - Stranded motorist
- This will require an on-board power generating unit
 - Diesel engine and MGB



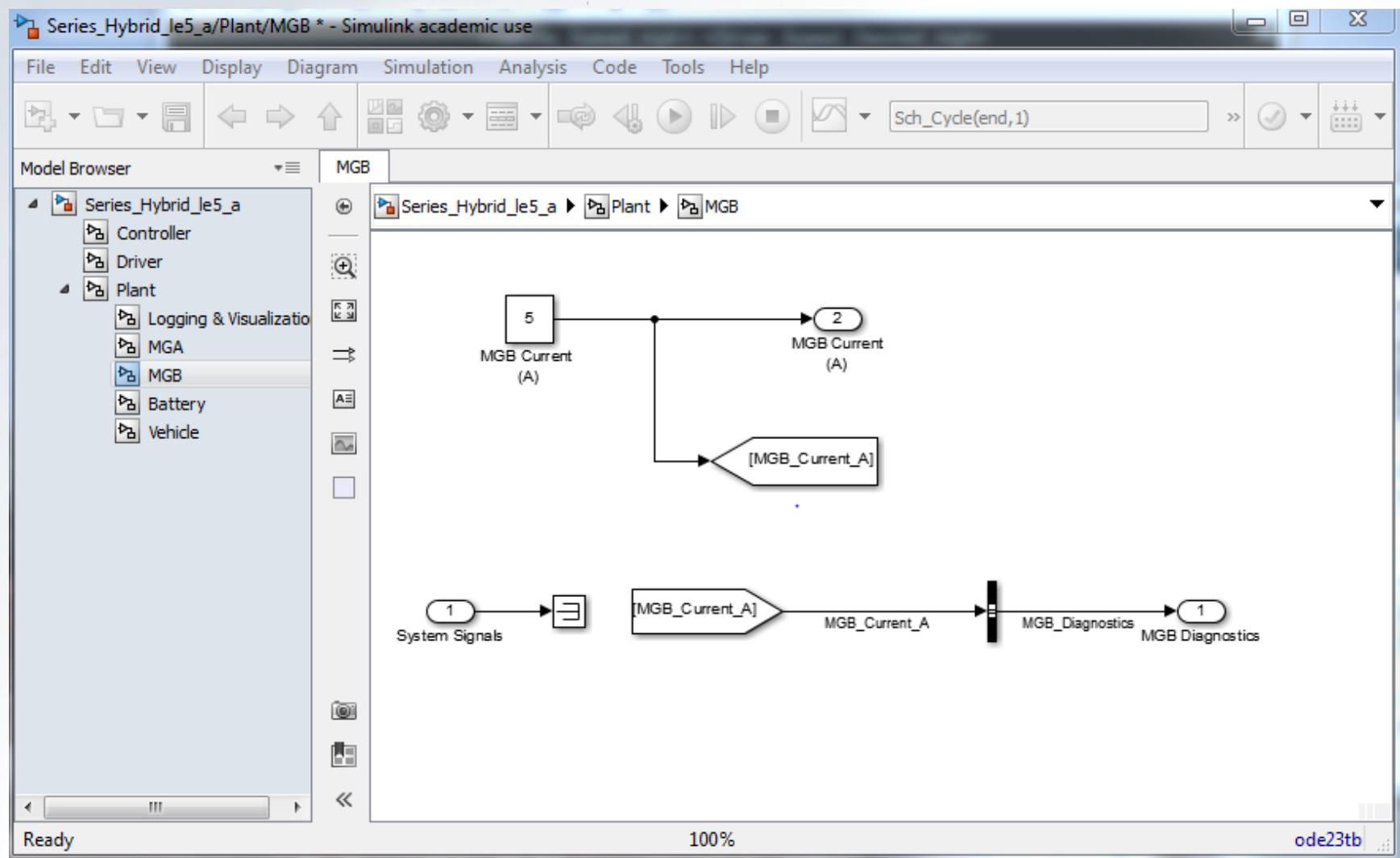
Battery SOC / GenSet

- The net effect of the GenSet is to provide power to the battery
 - MGB current at Battery voltage
- Let's make a ridiculously simple MGB
 - Constant current output
- Rename the model **Series_Hybrid_le5_a.slx**
- In the plant level of the model, drag in a new **Subsystem** and rename it MGB
 - Make the background green



- In the MGB subsystem
 - Delete the connecting wire
 - Create the System Signals and MGB Diagnostics bus
 - Drag in a Constant block
 - Rename it MGB Current (A)
 - Give it a value of 5
 - Drag in an Out1 block
 - Rename it MGB Current Output (A)
 - Put the current on the Diagnostics Bus

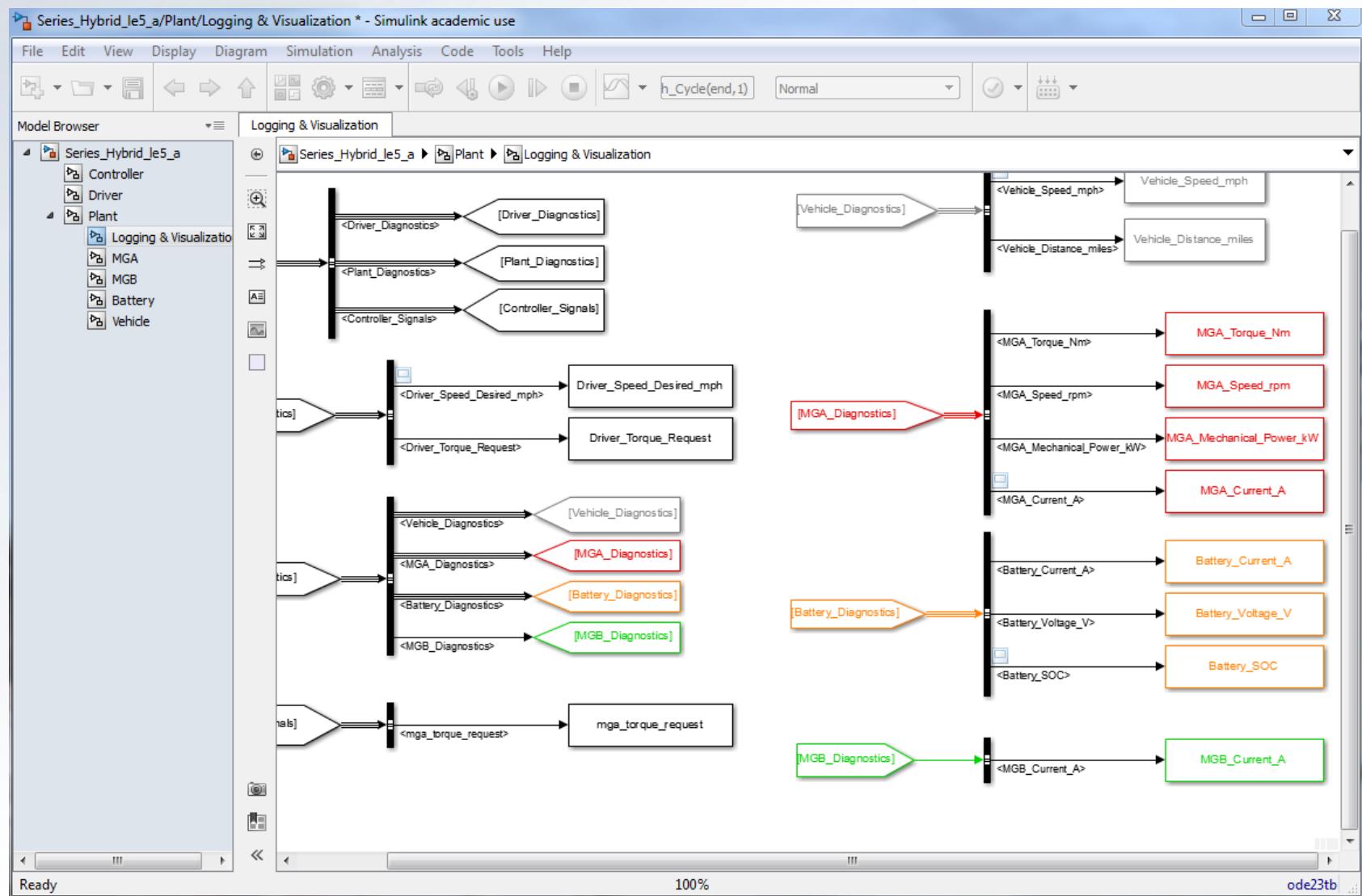
MGB





- Return to the Plant level and put the MGB Diagnostics on the Plant Diagnostics Bus
- Connect a System Signals **From** block
- Go to the Logging & Visualization subsystem and extract the MGB diagnostics
- Color accordingly!

MGB



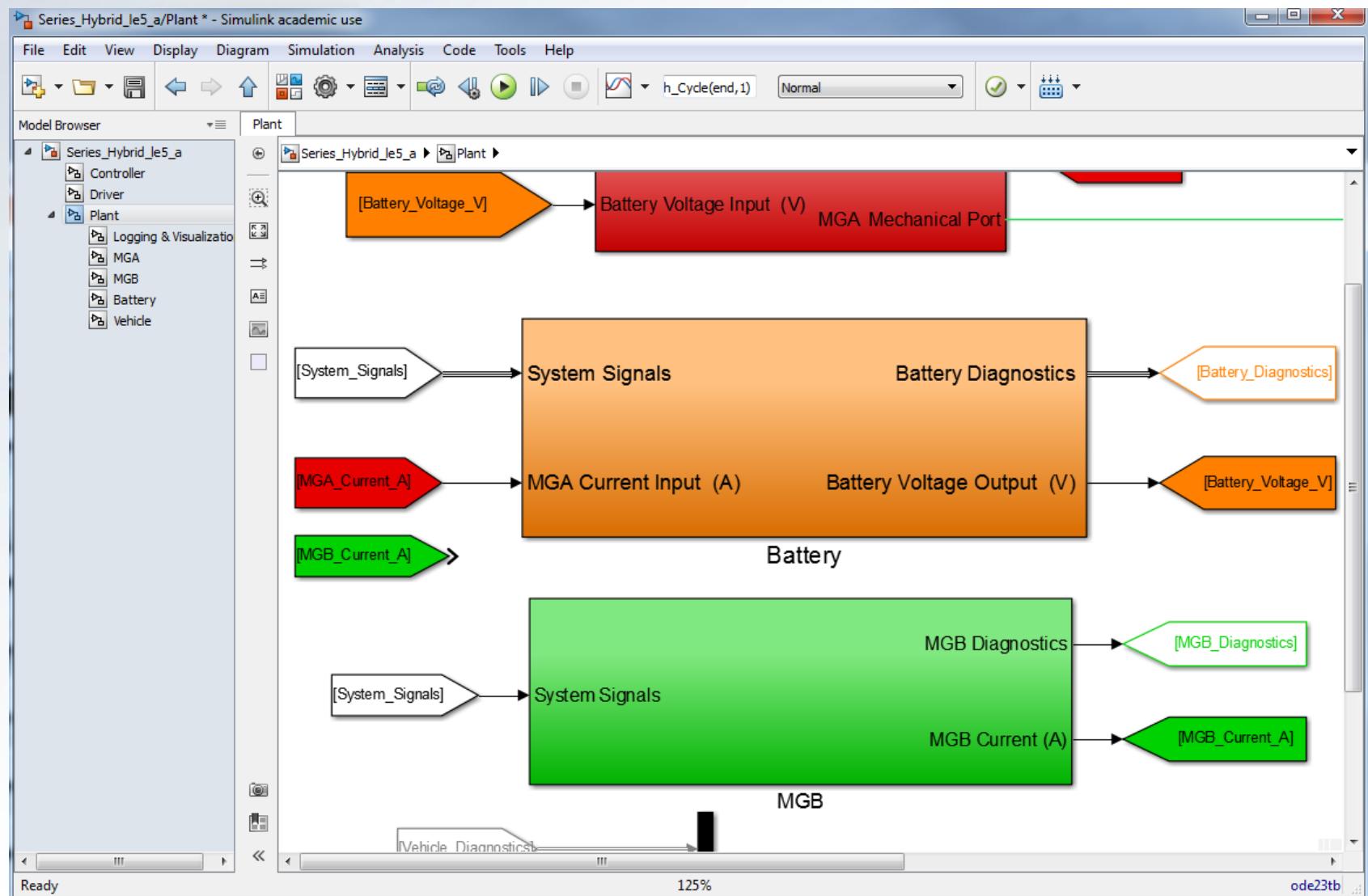


MGB / Battery

- Return to the Plant level of the model and use a **Goto** and a **From** block to make the electrical connection to the Battery



MGB / Battery



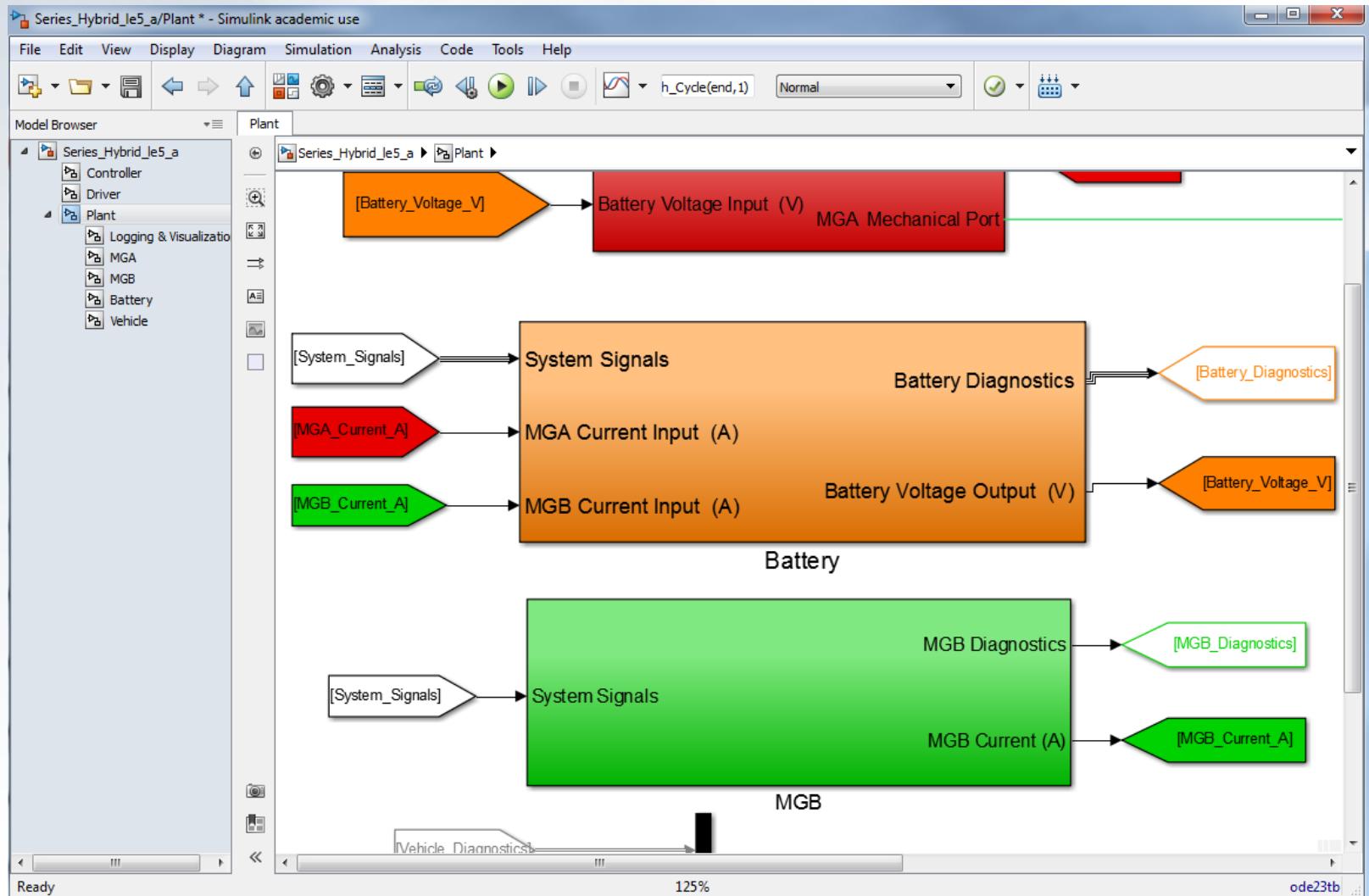


Battery

- Drag in an **In1** block and rename it MGB Current Input (A)
- Drag in a **Sum** block and add the two Current Inputs together
- Connect the output to the Battery_Current **Goto** block
- Excellent – both MGs can remove/provide current from/to the Battery
- Return to the Plant level and connect the MGB_Current_A **From** block



Battery



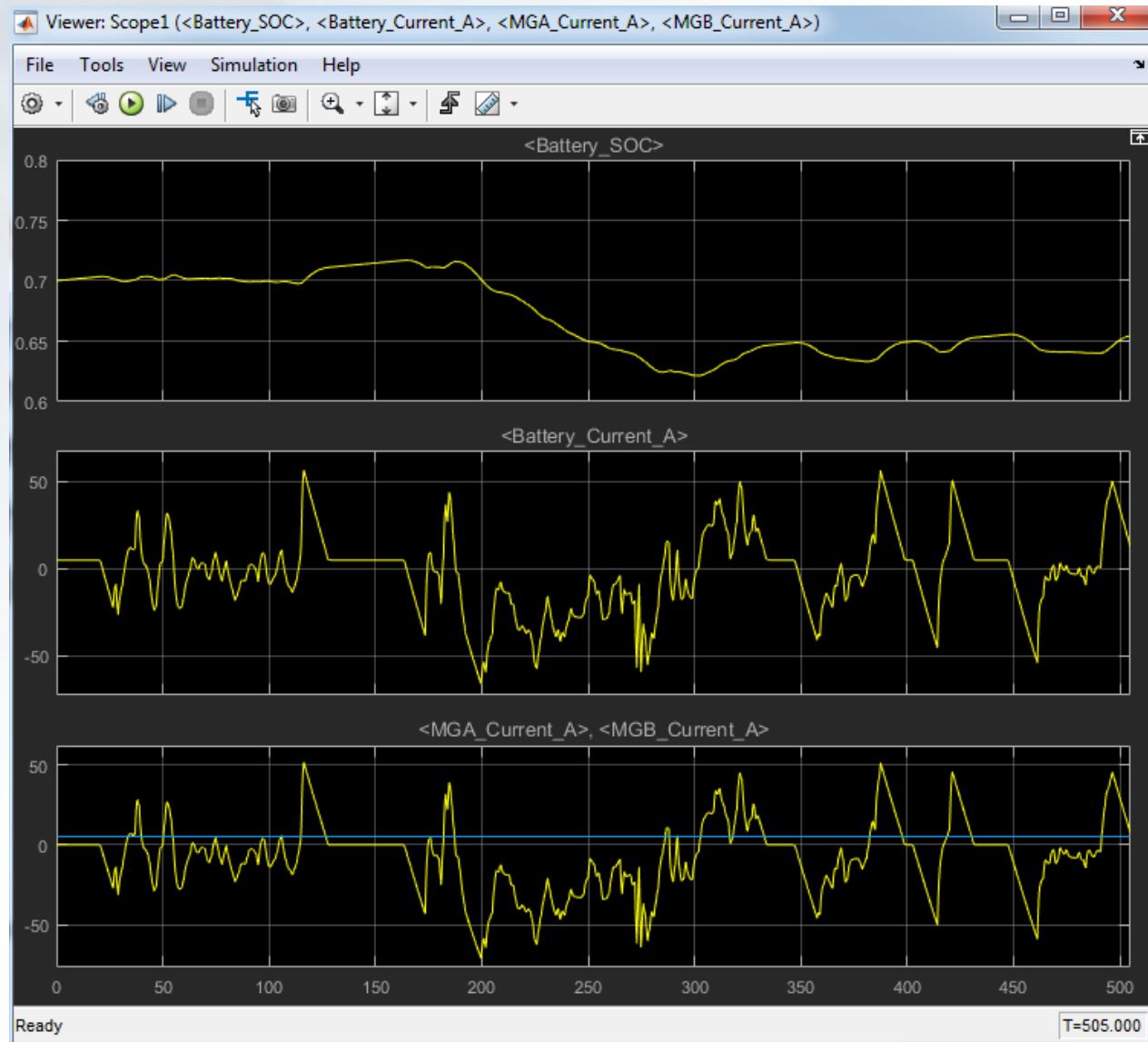


Battery

- In the Logging & Visualization subsystem
 - Right click on the Battery_SOC signal
 - Create a new scope
 - Give it three axis
 - Axis 1 – Battery SOC
 - Axis 2 – Battery Current
 - Axis 3 – MGA and MGB Current
- Run the simulation



Results



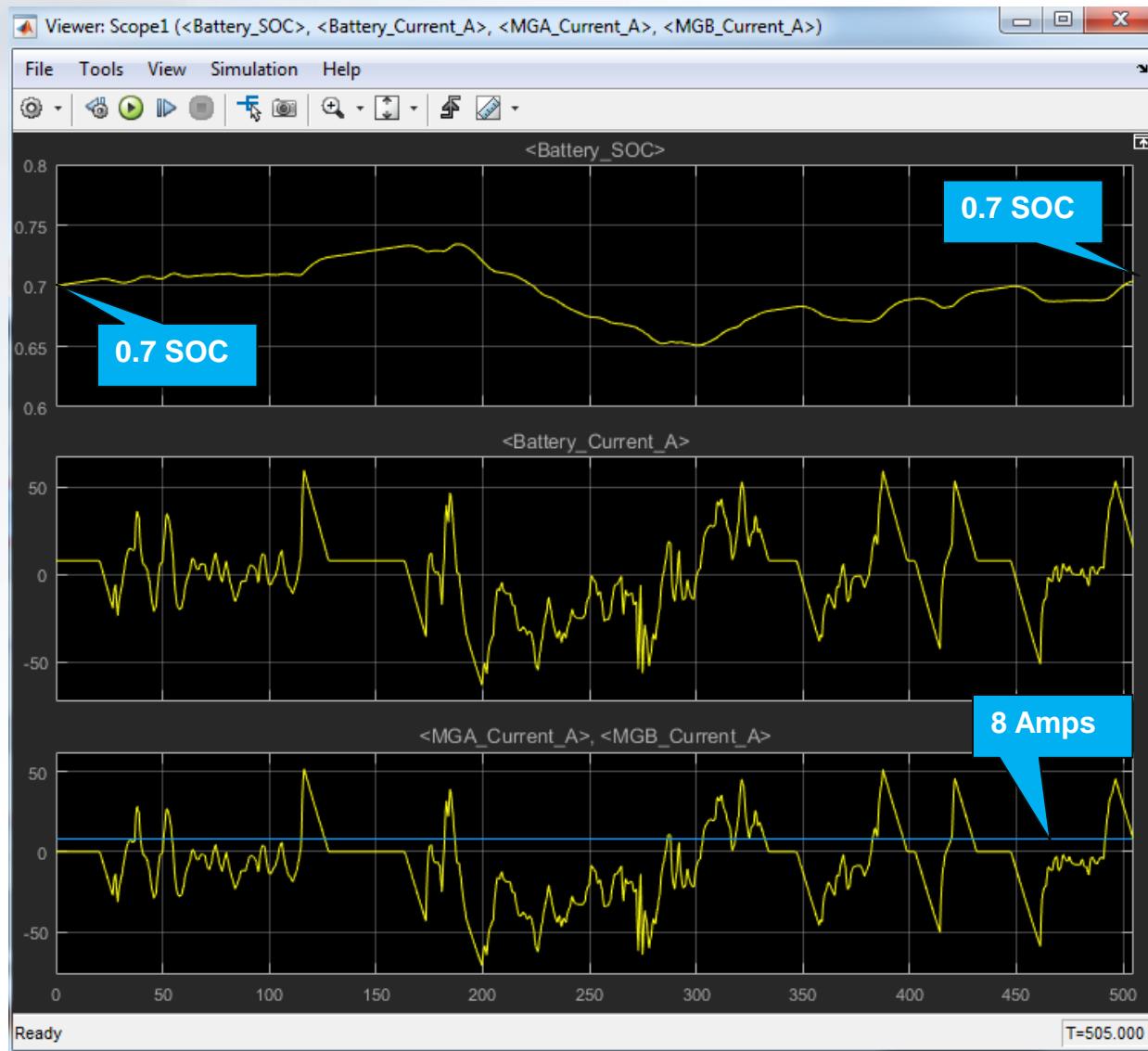


Results

- The “trickle charge” of a constant 5 amps kept the SOC up but the Battery was still *charge depleting*
- Vary the value of the MGB current until the Battery is relatively *charge sustaining*



Results





GenSet Controls

- Our series electric vehicle will use a genset which will be activated intermittently
 - SOC too low : start charging battery
 - SOC too high : stop charging battery
- Let's make the idealized genset do exactly that via MGB and a state machine
- Go to the Controller level of the model

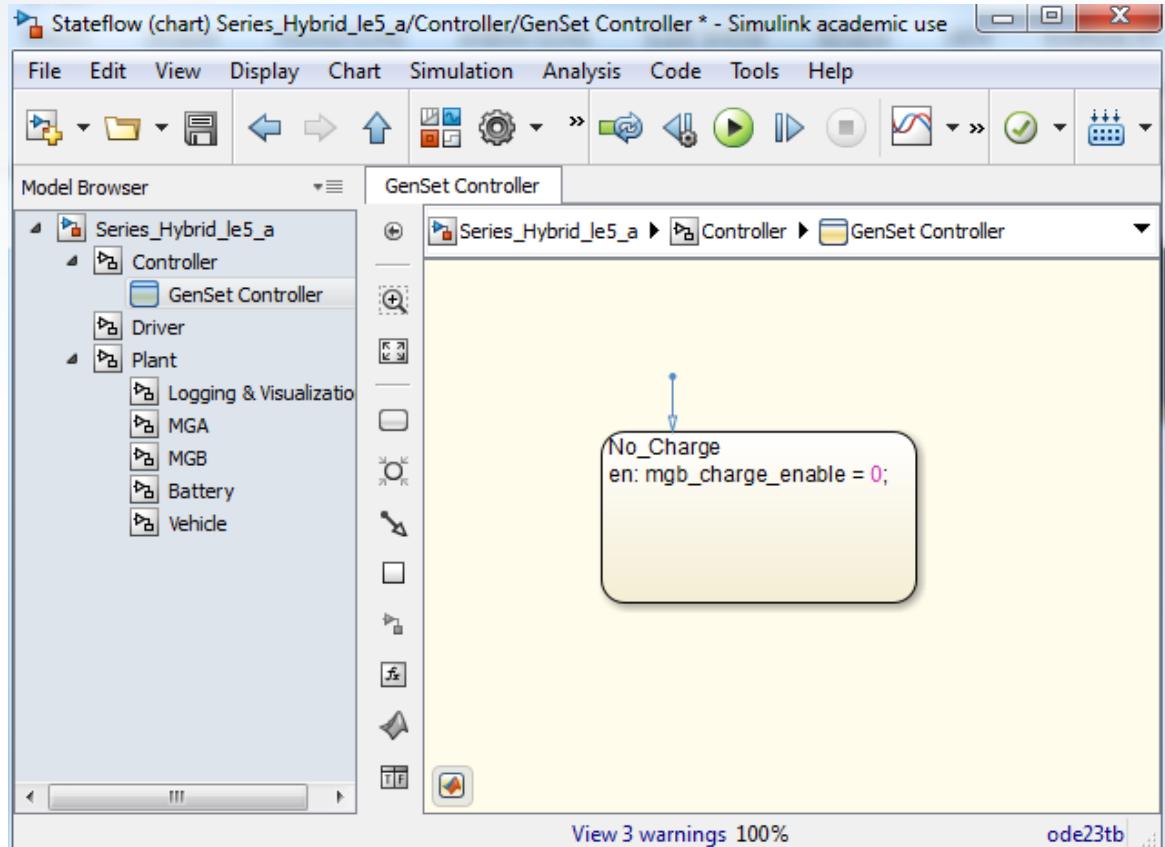


GenSet Controls

- From
 - Simulink / Stateflow
- Drag in a **Chart** block
 - Rename it GenSet Controller
 - Double click on it
- The first logical state will be the “no charge” state
 - The Battery SOC is above the lower limit

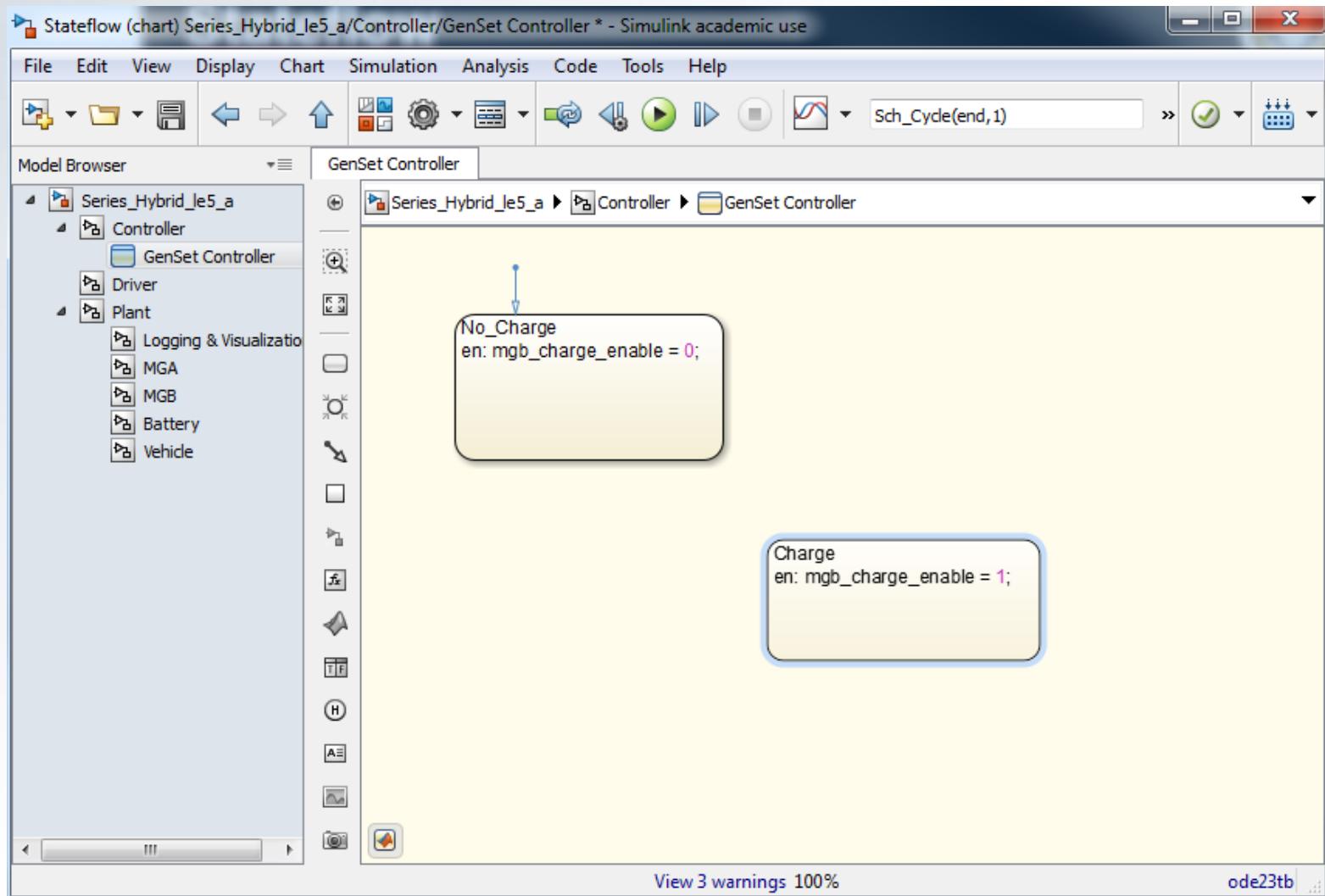
Stateflow

- Left click on the State icon
- Left click on the window
- Name the state
 - No_Charge
- On entry, set the MGB charge enable to 0
- Create a charge state





Stateflow



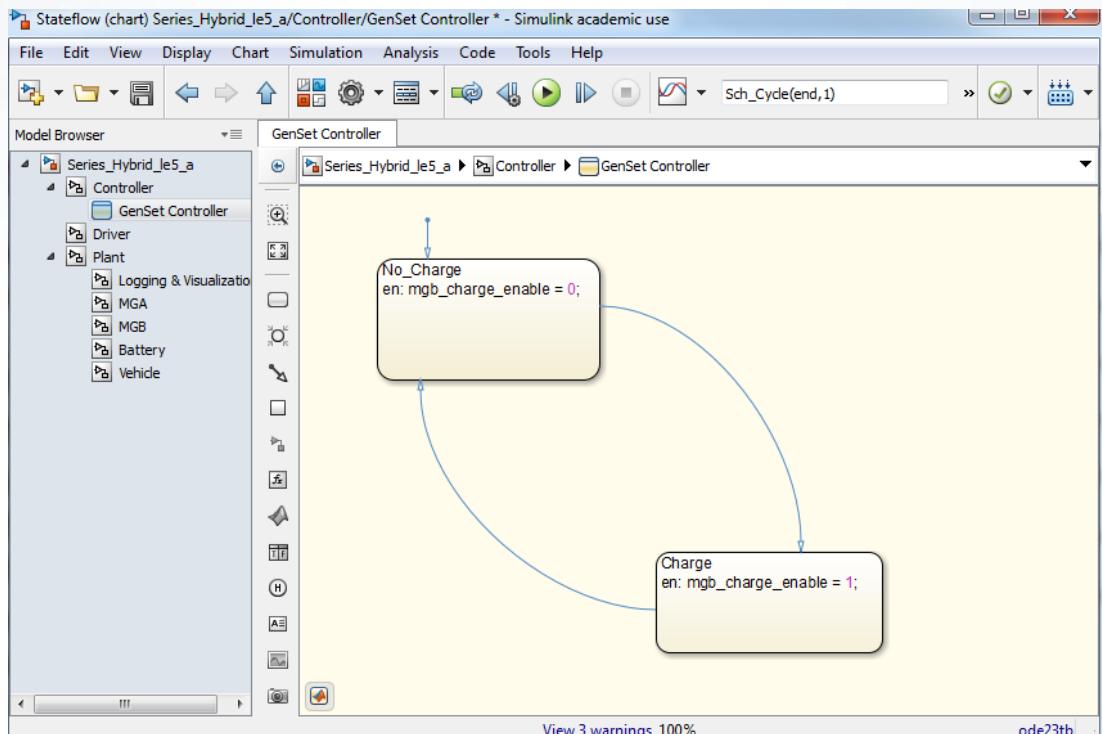


Stateflow

- To go from one state to another, a *transition* must occur
- These transitions are controlled by parameters called *guards*
- If, at the instance the guard is queried, the parameters are true then the transition occurs
- Let's now create the transitions followed by the guards

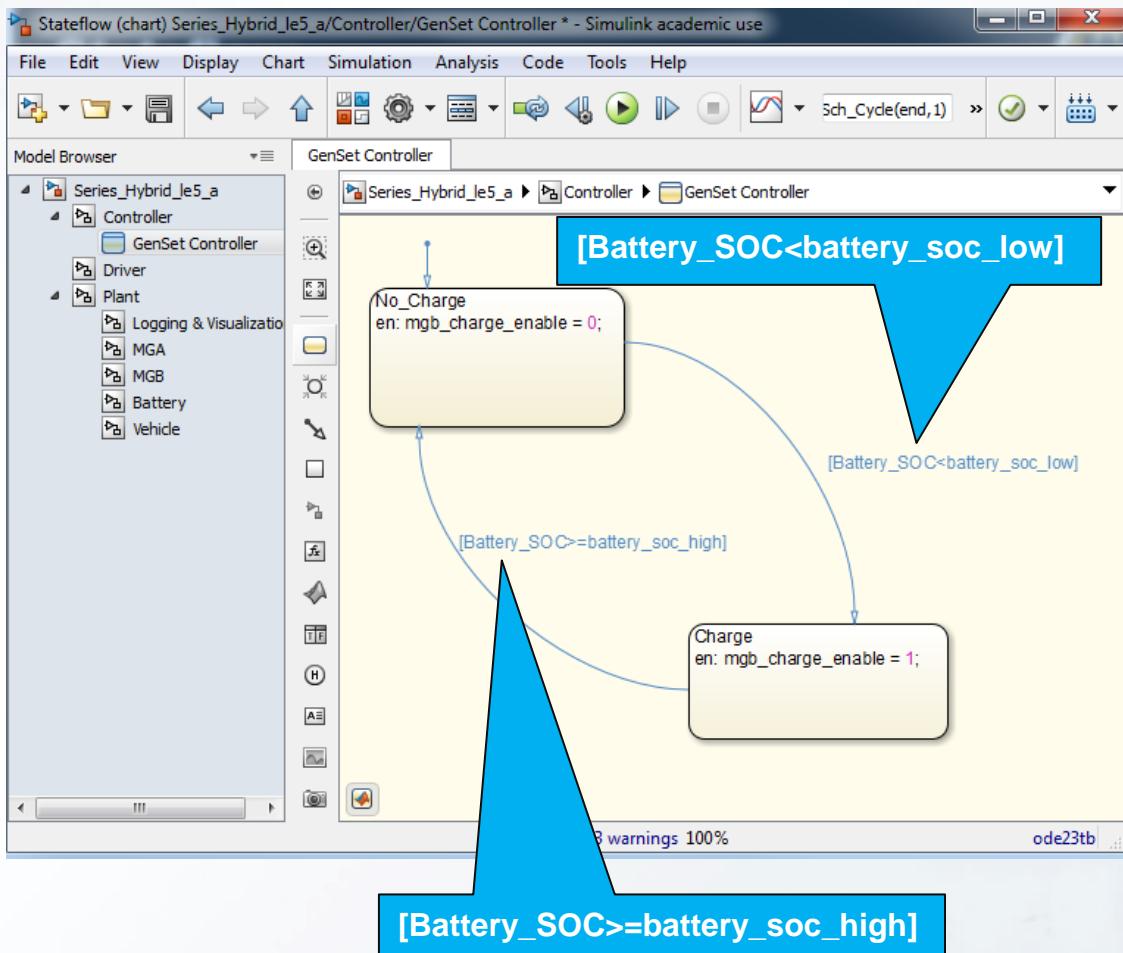
Stateflow

- Place the mouse over the No_Charge state
- Left Click
- Hold and drag to the Charge state
- Connect the Charge back to the No_Charge



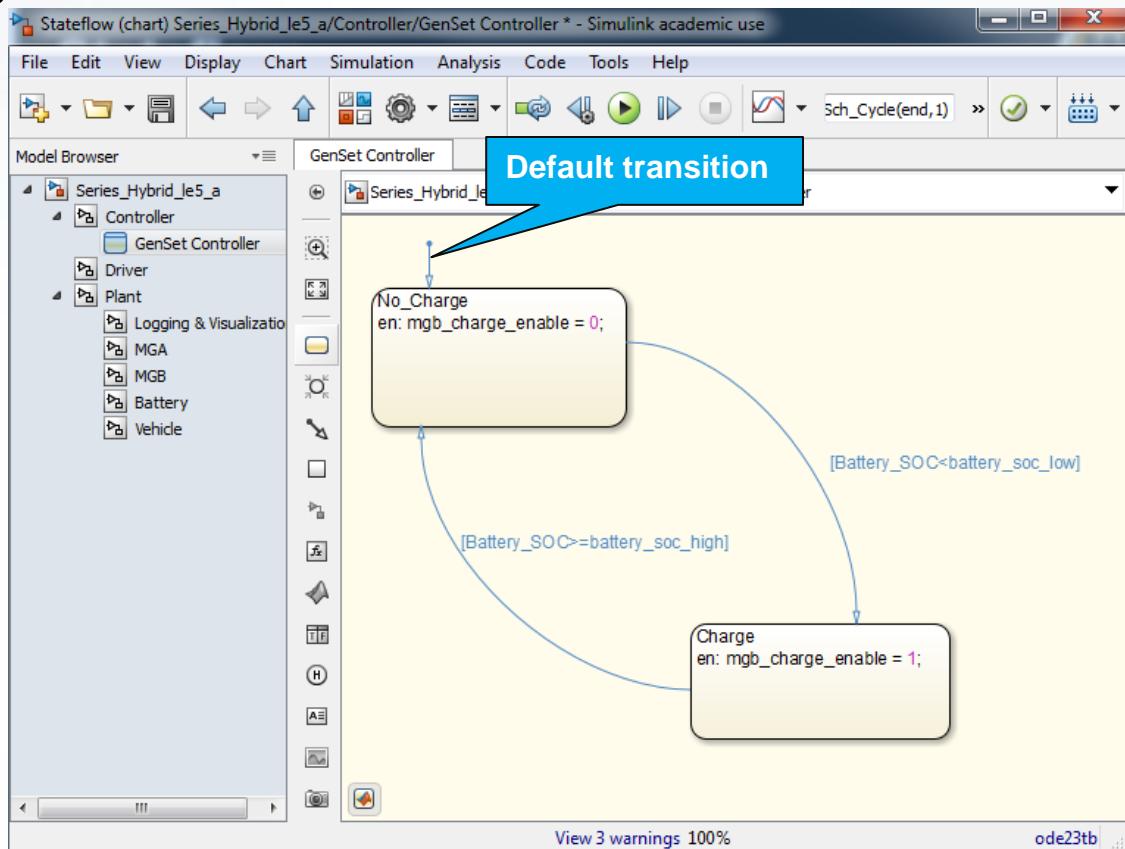
Stateflow

- Left click on the right transition
- Replace the question mark with the code at right
 - Enclosed in square brackets
- Repeat for the second transition



Stateflow

- Stateflow needs to know which state to “wake up” in
- The default transition was created when the first state was placed





Stateflow

- We need to add the Battery_SOC input from Simulink into Stateflow
 - Select
 - Chart / Add Inputs & Outputs / Data Input From Simulink
 - Change the Name to Battery_SOC
 - Click OK
- If you select
 - Tools / Model Explorer
- You can see that Battery_SOC has been added
- Close Model Explorer



Stateflow

Stateflow (chart) Series_Hybrid_le5_a/Controller/GenSet Controller * - Simulink academic use

File Edit View Display Chart Simulation Analysis Code Tools Help

Model Browser

- Series_Hybrid_le5_a
 - Controller
 - GenSet Controller
 - Driver
 - Plant
 - Logging & Visual
 - MGA
 - MGB
 - Battery
 - Vehicle

Parse Chart Refresh Blocks Group & Subchart Add Inputs & Outputs Add Other Elements Add Pattern In Chart Insert Pattern On Selection Save Pattern Create Container Format Arrange Library Link Decomposition Execution Order Subchart mappings... Properties...

Data Input From Simulink Data Output To Simulink Event Input From Simulink Event Output To Simulink Message Input From Simulink Message Output To Simulink

```
graph TD; A[Battery_SOC < battery_soc_low] --> Charge[Charge  
en: mgb_charge_enable = 1;]
```

[Battery_SOC < battery_soc_low]

Charge
en: mgb_charge_enable = 1;

View 3 warnings 100% code23tb



Stateflow

- Similarly we need to add the value of mgb_charge_enable to Simulink
 - Select
 - Chart / Add Inputs & Outputs / Data Output to Simulink
 - Change the Name to mgb_charge_enable
 - Click OK
- We also need to add the guards as parameters
 - Select
 - Chart / Add Other Elements / Parameter...
 - Change the Name to battery_soc_low
 - Click OK
- Repeat for battery_soc_high



Stateflow

- The final step is to set the frequency at which the guards are queried
 - Select
 - Chart / Add Inputs & Outputs / Event Input From Simulink
 - Change the Name to Clock
 - Click OK
- After this step, Model Explorer should be as shown in the next slide



Model Explorer

Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy

- Simulink Root
 - Base Workspace
 - Series_Hybrid_le5_a*
 - Model Workspace
 - Configuration (Active)
 - Code for Series_Hybrid_le5_a
 - Simulink Design Verifier results
 - Advice for Series_Hybrid_le5_a
 - Controller
 - GenSet Controller
 - Driver
 - Plant

Contents of: Series_Hybrid_le5_a/Controller/GenSet Controller (only)

Column View: Stateflow Show Details 5 of 10 object(s)

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	Comment
Battery_SOC	Input	1		Inherit: Same as Simulink	-1		un
mgb_charge_enable	Output	1		Inherit: Same as Simulink	-1		un
battery_soc_high	Parameter			Inherit: Same as Simulink	-1		un
battery_soc_low	Parameter			Inherit: Same as Simulink	-1		un
Clock	Input	1					

Chart: GenSet Controller

General Fixed-point properties Documentation

Name: GenSet Controller
Machine: (machine) Series_Hybrid_le5_a
Action Language: MATLAB
State Machine Type: Classic
Update method: Inherited Sample Time:

User specified state/transition execution order
 Export Chart Level Functions (Make Global)
 Execute (enter) Chart At Initialization
 Initialize Outputs Every Time Chart Wakes Up
 Enable Super Step Semantics
 Support variable-size arrays
 Saturate on integer overflow
 Create data for monitoring: Child activity
 Lock Editor

Revert Help Apply

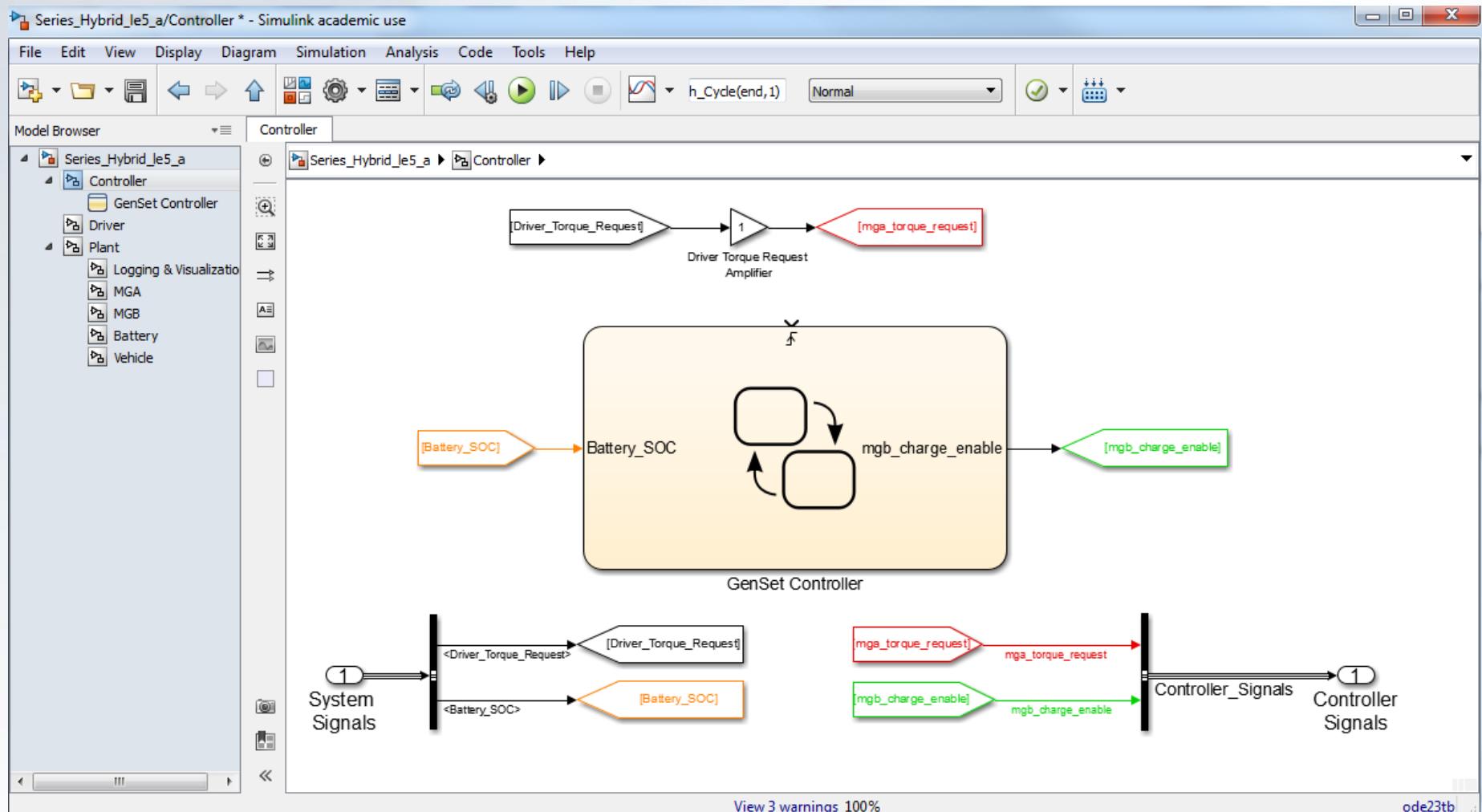


GenSet Controller

- Return to the Controller level of the model
- Make the GenSet Controller subsystem larger
- Extract the Battery_SOC from the System Signals bus and route it to the GenSet Controller
- Route the mga_charge_enable signal to the Control Signals bus and add it



GenSet Controller



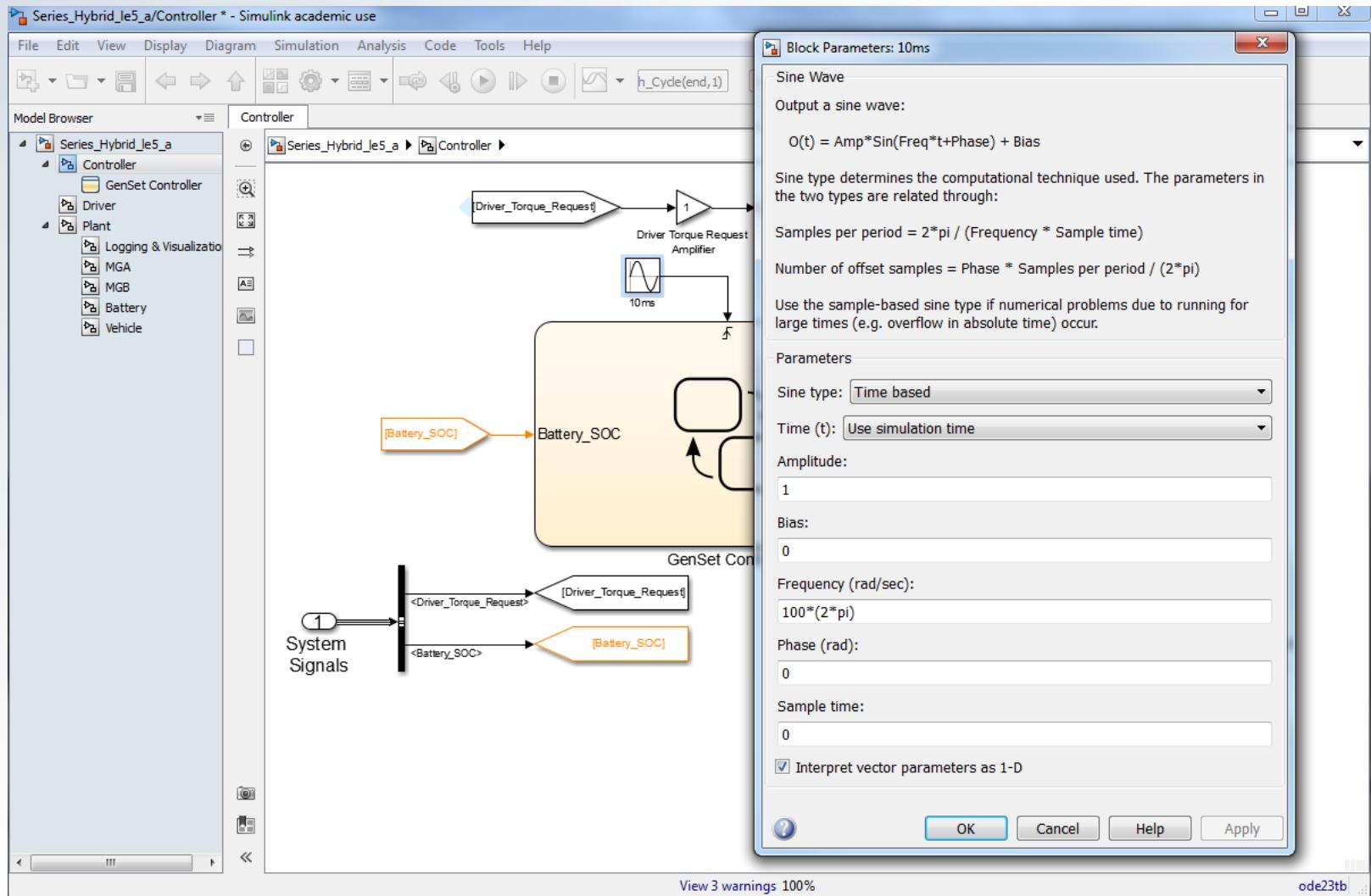


GenSet Controller

- From
 - Simulink / Sources
- Drag in a **Sine Wave** block
 - Rename it 10ms
- Connect it to the Clock Port
- Double Click on it
 - Change the Frequency to $100*(2*\pi)$
 - Click OK



GenSet Controller





GenSet Controller

- Excellent, we have built a simple two state controller which, based on the Battery SOC, will determine whether or not it should be charged
- This check will be performed every 10 ms
- In the init file, we need to add guards
 - Low SOC : 0.6
 - High SOC : 0.7
- Add to the init file



GenSet Controller

C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m

The image shows a MATLAB/Simulink interface with a script editor window open. The window title is 'C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m'. The menu bar includes 'EDITOR', 'PUBLISH', and 'VIEW'. The toolbar has icons for New, Open, Save, Find Files, Compare, Print, Go To, Breakpoints, Run, Run and Advance, Advance, and Run and Time. The script content is as follows:

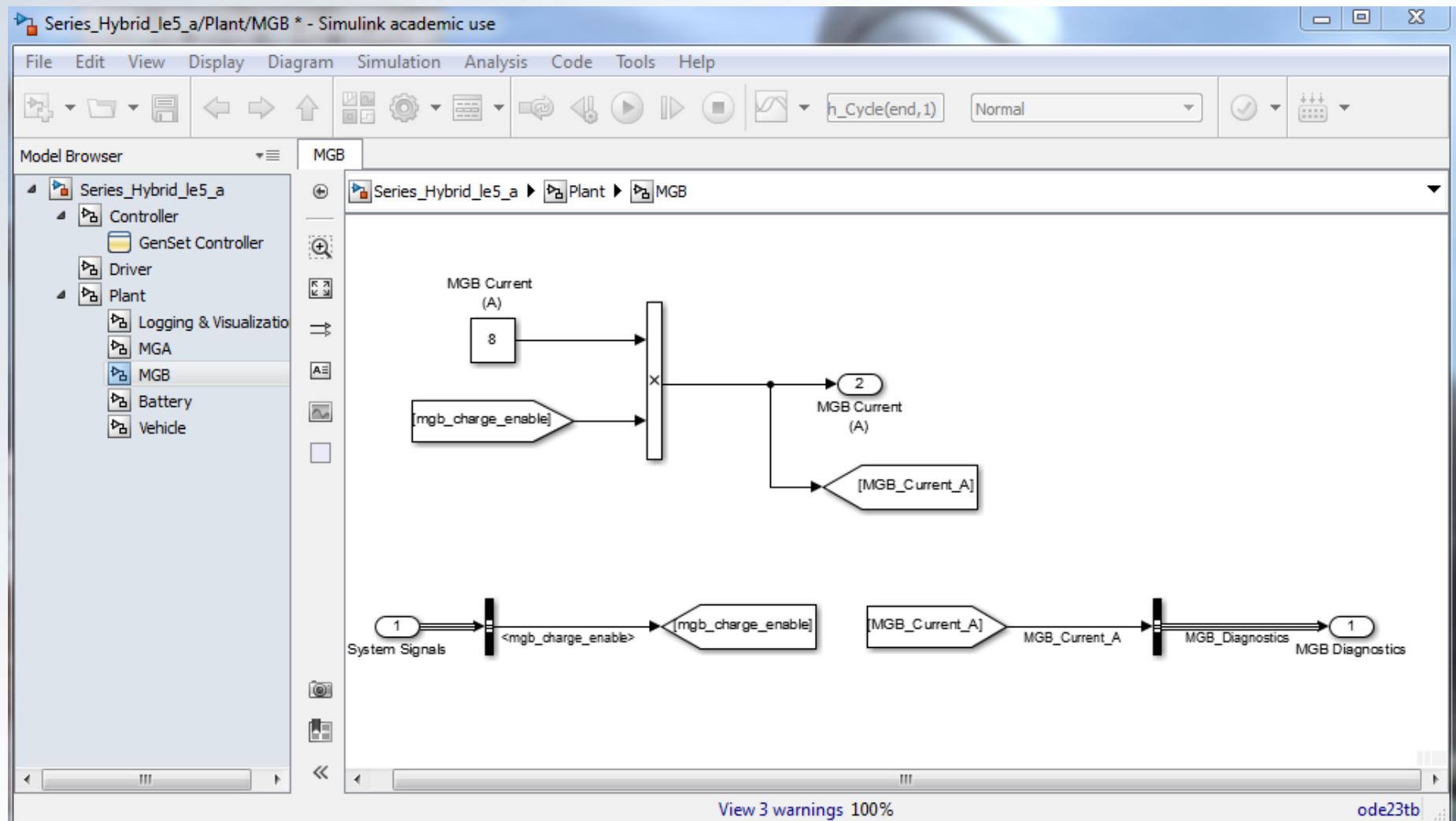
```
17 %Battery Parameters
18 Battery_Voltage = 336; %Battery Voltage, V
19 - Battery_Capacity = 8.5; %Battery Max Capacity, Ahr
20 - Battery_Initial_SOC = 0.7; %Battery initial SOC
22
23 %Controller Parameters
24 - battery_soc_low = 0.6; %Battery SOC low value
25 - battery_soc_high = 0.7; %Battery SOC high value
26
27 %Logging Parameters
```

The status bar at the bottom shows 'script' in the first field, 'Ln 25 Col 54' in the second field, and three small icons in the third field.

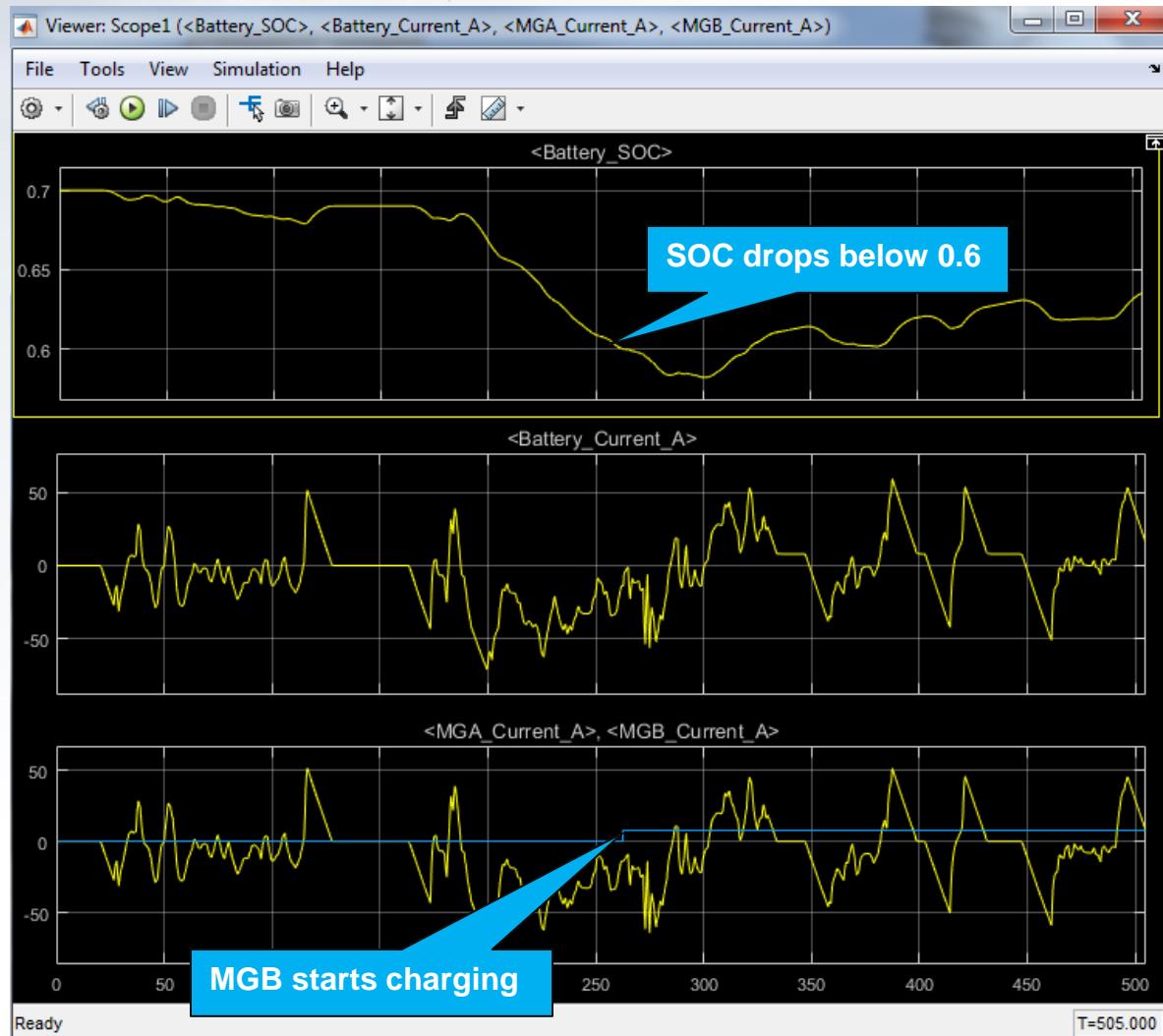


- Return to the MGB subsystem
- Extract the mgb_charge_enable signal from the System Signals
- Drag in a **Product** block and multiply the mgb_charge_enable signal with the MGB Current block
- Run the simulation

MGB



Results



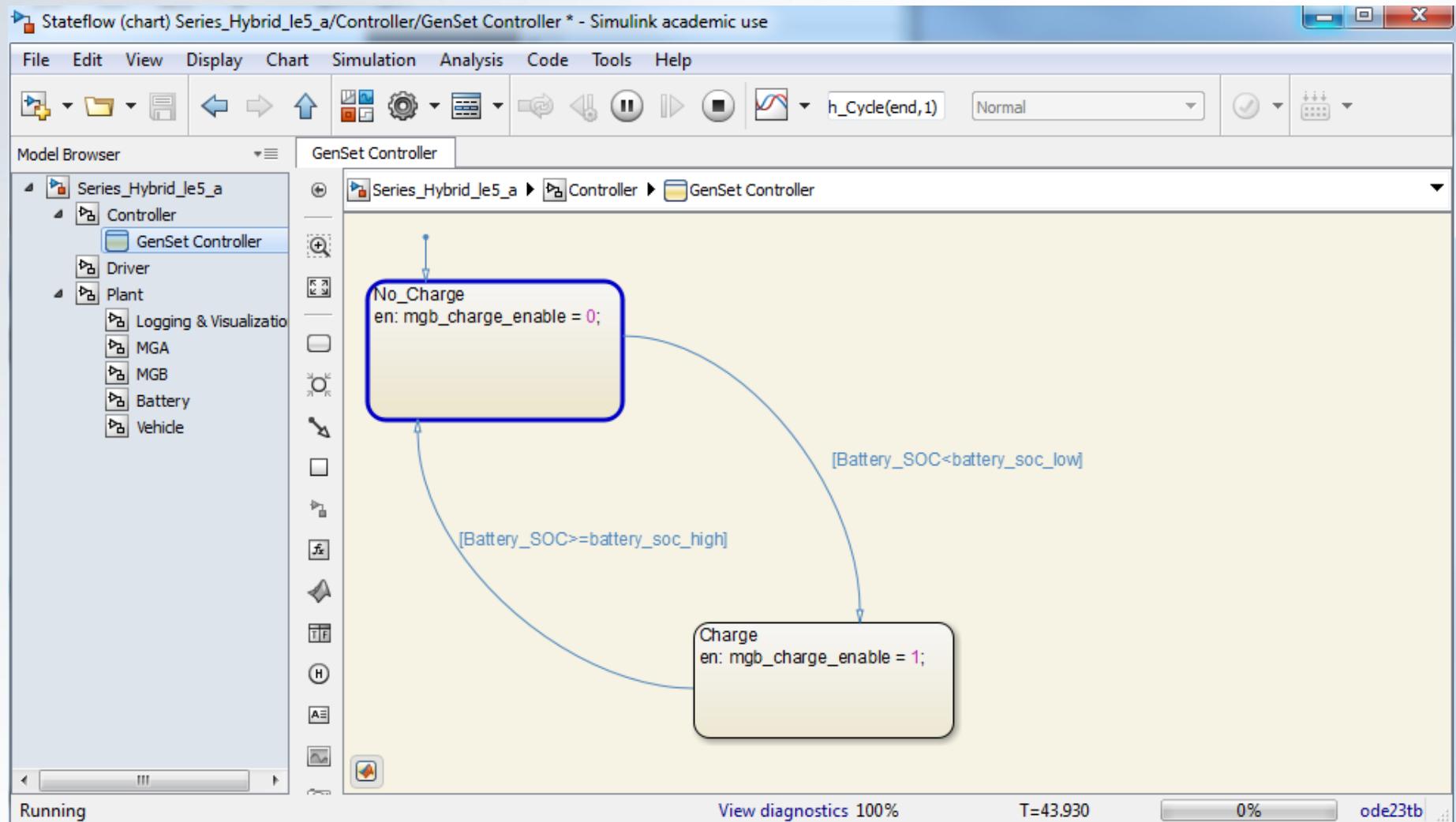


Results

- In the init file, change the drive cycle to
 - Schedule_FU505_Ten_Times.mat
- Increase the MGB current to 50 amps
- Run the simulation
 - MGB now cycles on and off
 - Watch the Stateflow chart
 - The current state is highlighted during the simulation

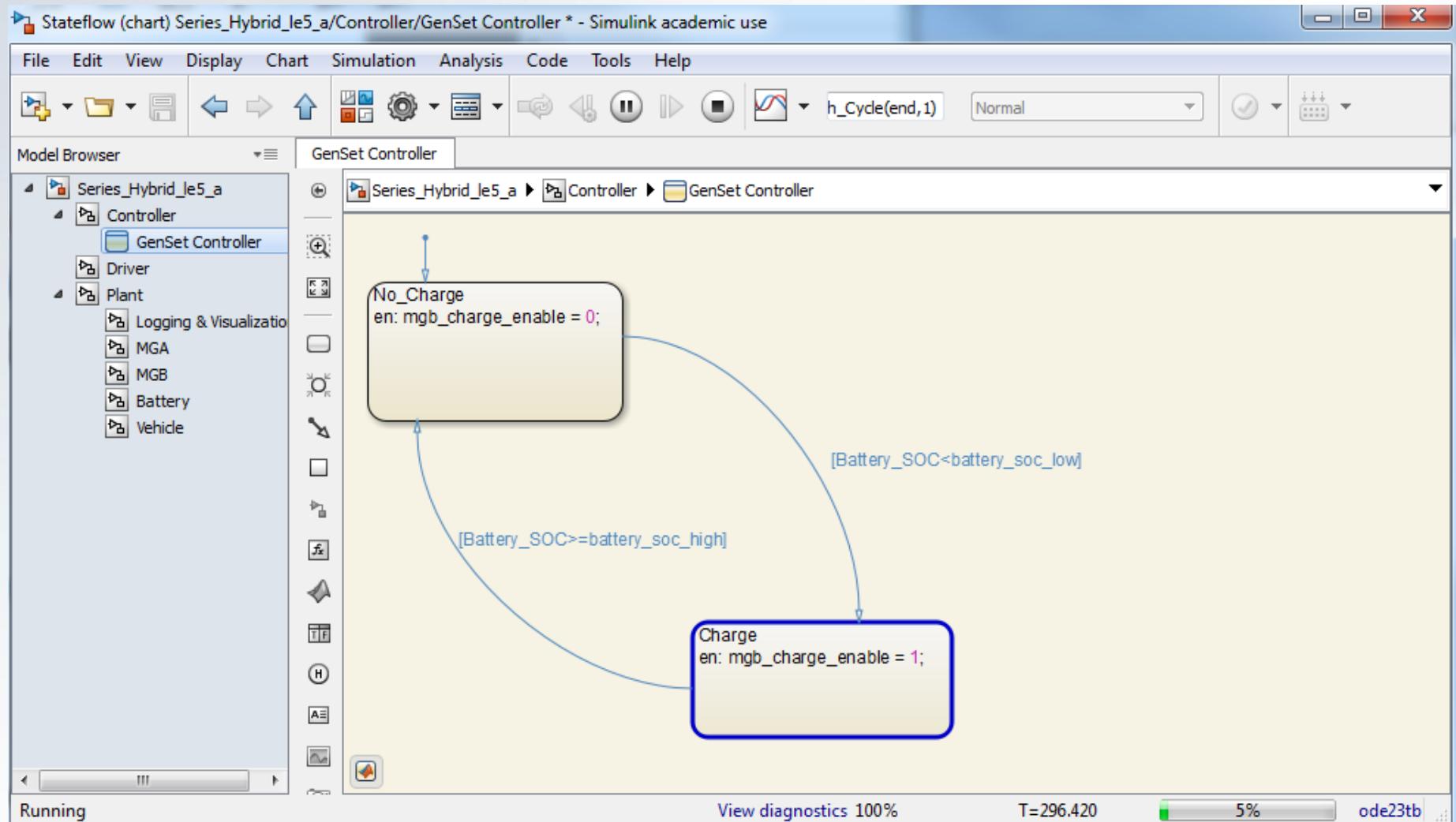


Results: MGB not charging





Results: MGB charging





GenSet and Stateflow

- Review
 - Created a simple GenSet which provides current to recharge the battery
 - Tweaked the trickle charge current for charge sustaining
 - Created a state controller to turn the charge on and off

A blue-toned photograph of the Georgia Tech Campanile, a historic brick building with a tall, spired tower. The words "THE TECH" are prominently displayed on the side of the tower. The image is framed by a thin yellow border.

MBSD Lecture 6

Improved GenSet and Controls



Outline

- Improved MGB model
- Simple diesel engine model
- Charging logic



Improved GenSet

- Previously, the GenSet simply provided a constant current
- Improvements
 - Diesel Engine
 - Constant torque
 - Speed control
 - MGB
 - Recycle MGA model
- Rename your model
 - Series_Hybrid_le6_a.slx

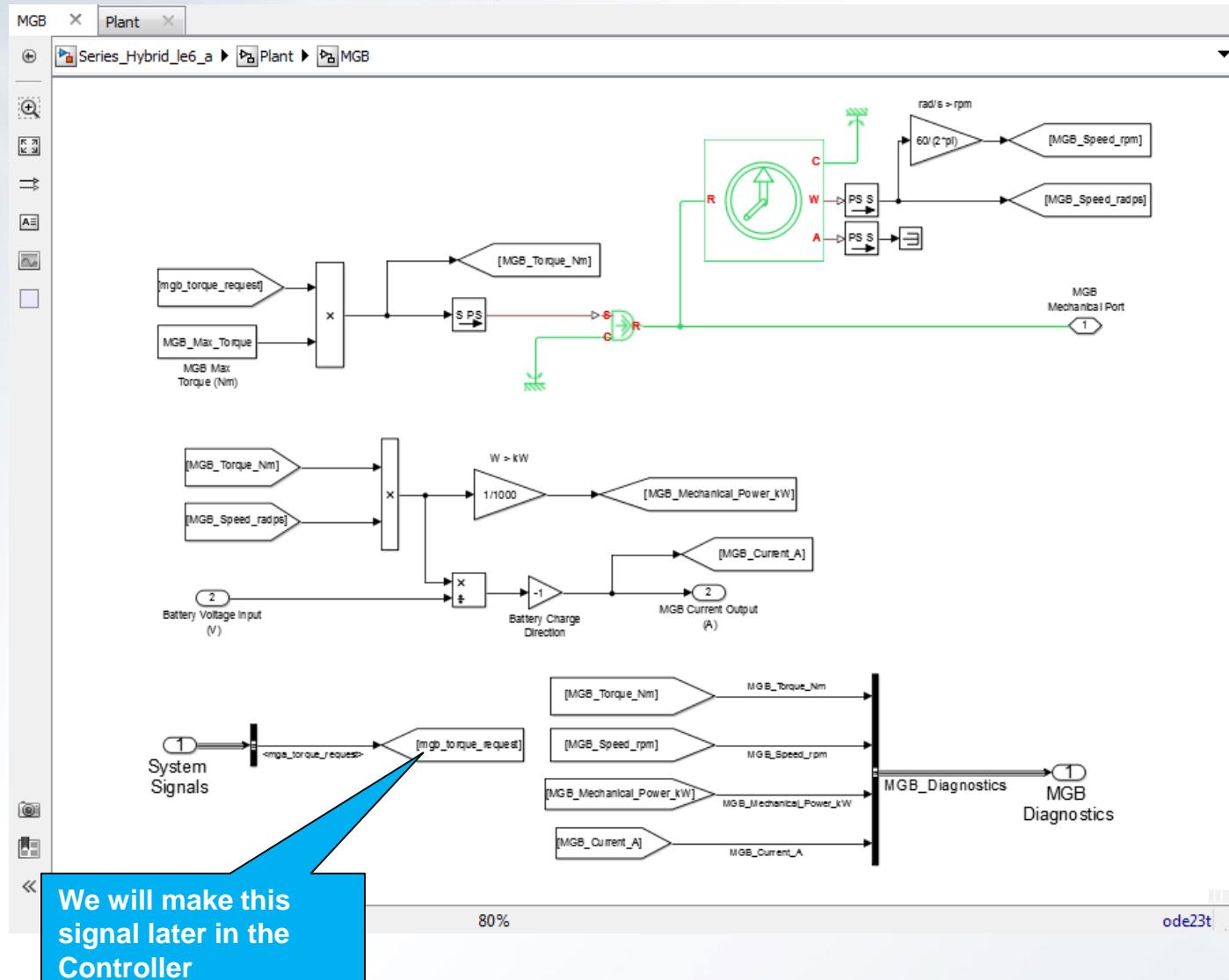


Improved MGB

- Delete the MGB model you just got done making
- Copy the MGA model
 - Make it green
 - Rename EVERYTHING from MGA to MGB
- Add the MGB max torque to the init file
 - 200 Nm
- Update the Logging & Visualization for MGB
 - Note: We will create the signal for mgb_torque_request later in the controller subsystem

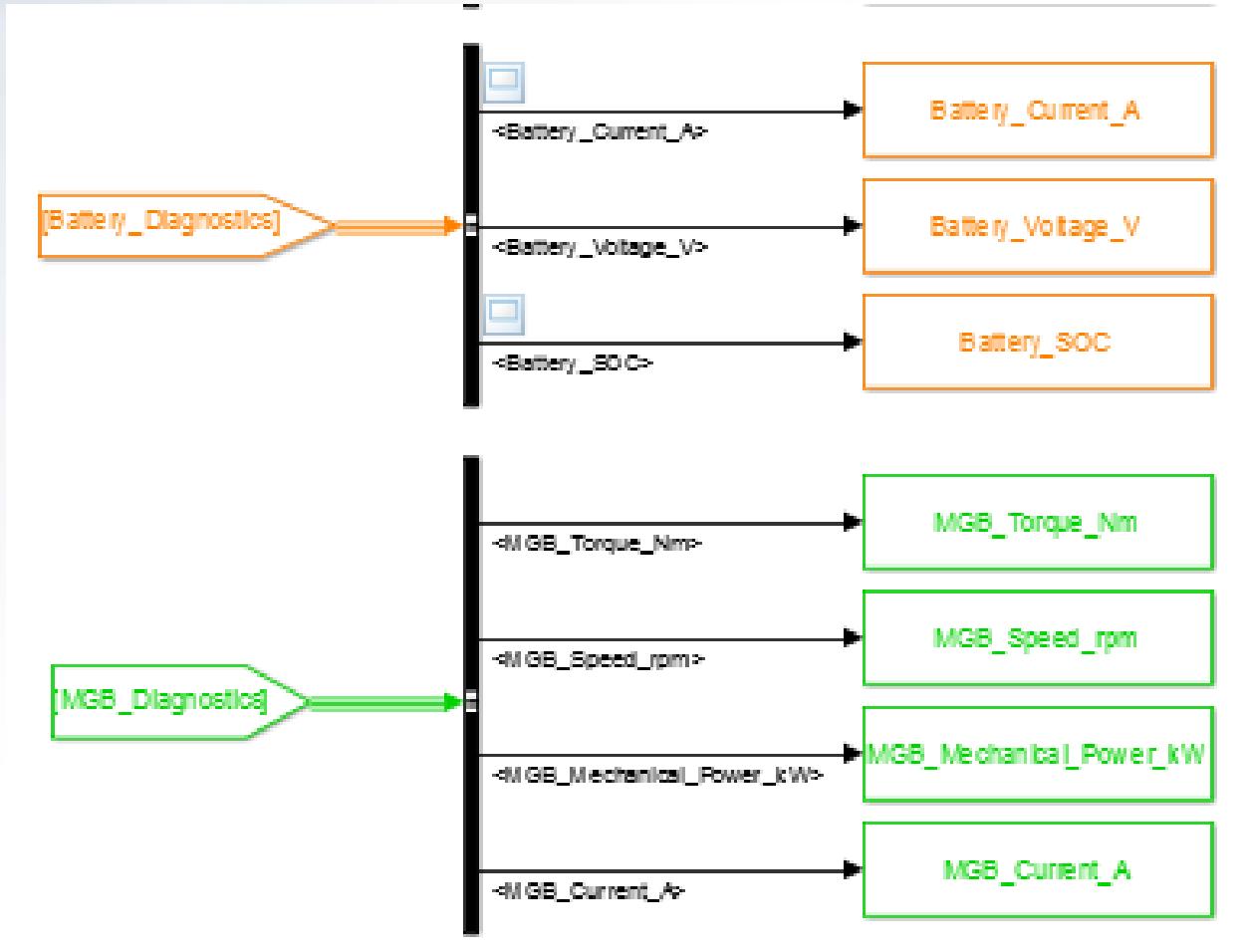


Improved MGB





Logging & Visualization



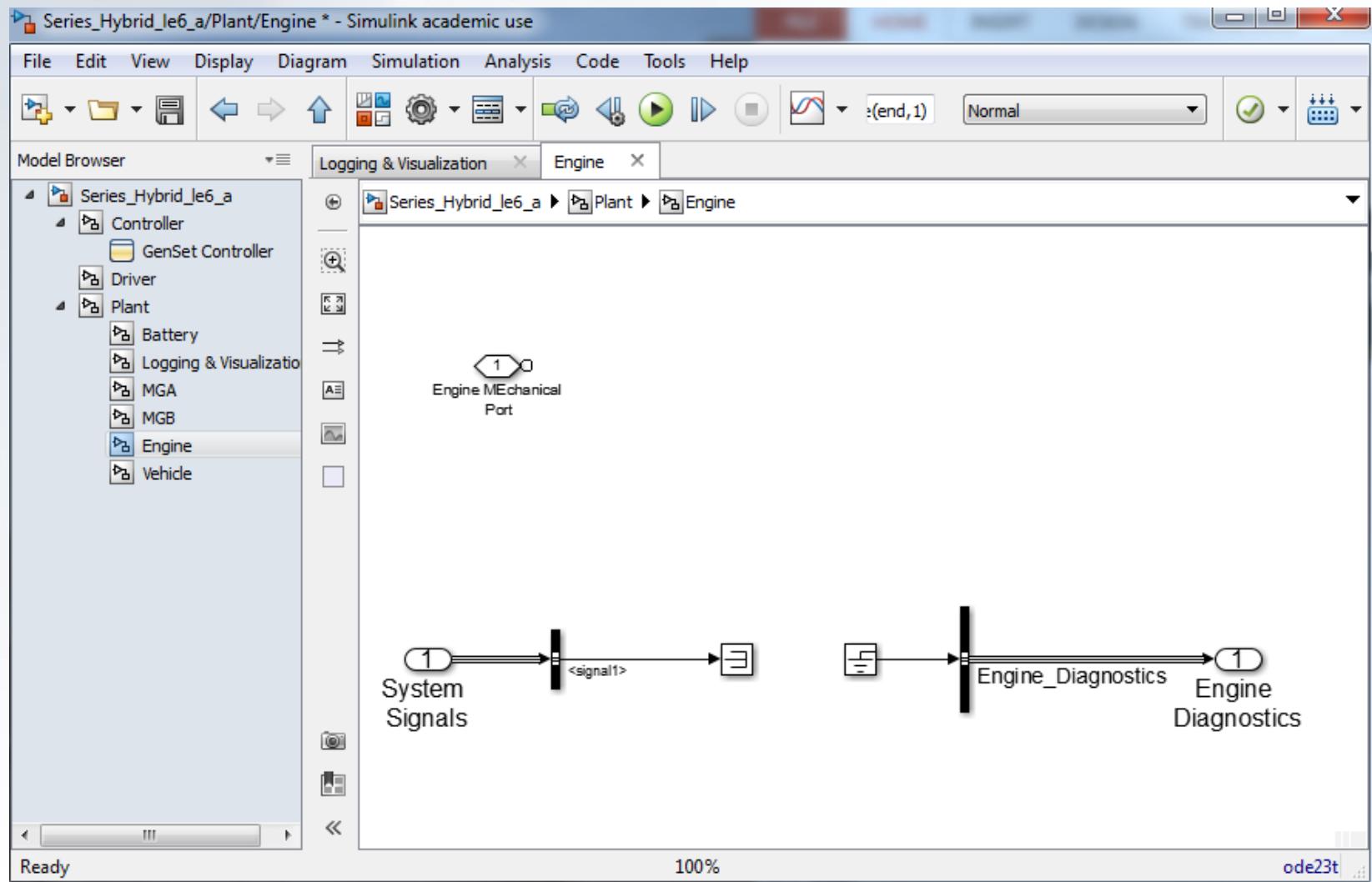


Engine

- Return to the Plant level, drag in a **Subsystem**, and rename it Engine
 - Make it yellow
- In the Engine subsystem, create the System Signals and Engine Diagnostics bus
- Drag in a **Connection Port** for the mechanical connection to MGB



Engine



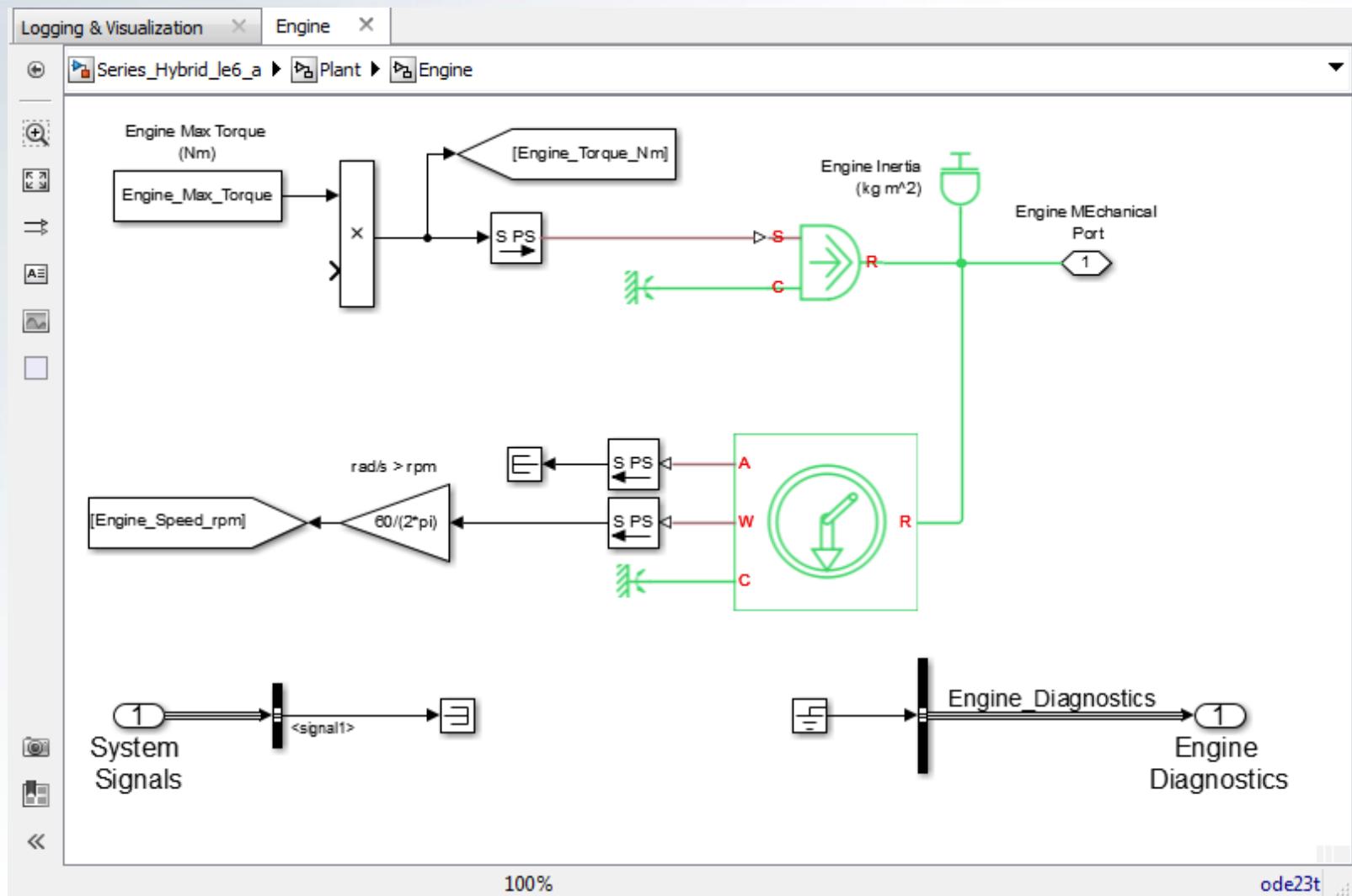


Engine

- Drag in
 - One Constant
 - One Simulink-PS Converter
 - One Ideal Torque Source
 - One Inertia
 - One Ideal Rotational Motion Sensor
 - One Gain
 - One Terminator
 - One Product
 - Two Goto
 - Two PS-Simulink Converters
 - Two Mechanical Rotational References
- Arrange and label as shown on the next slide



Engine



100%

ode23t

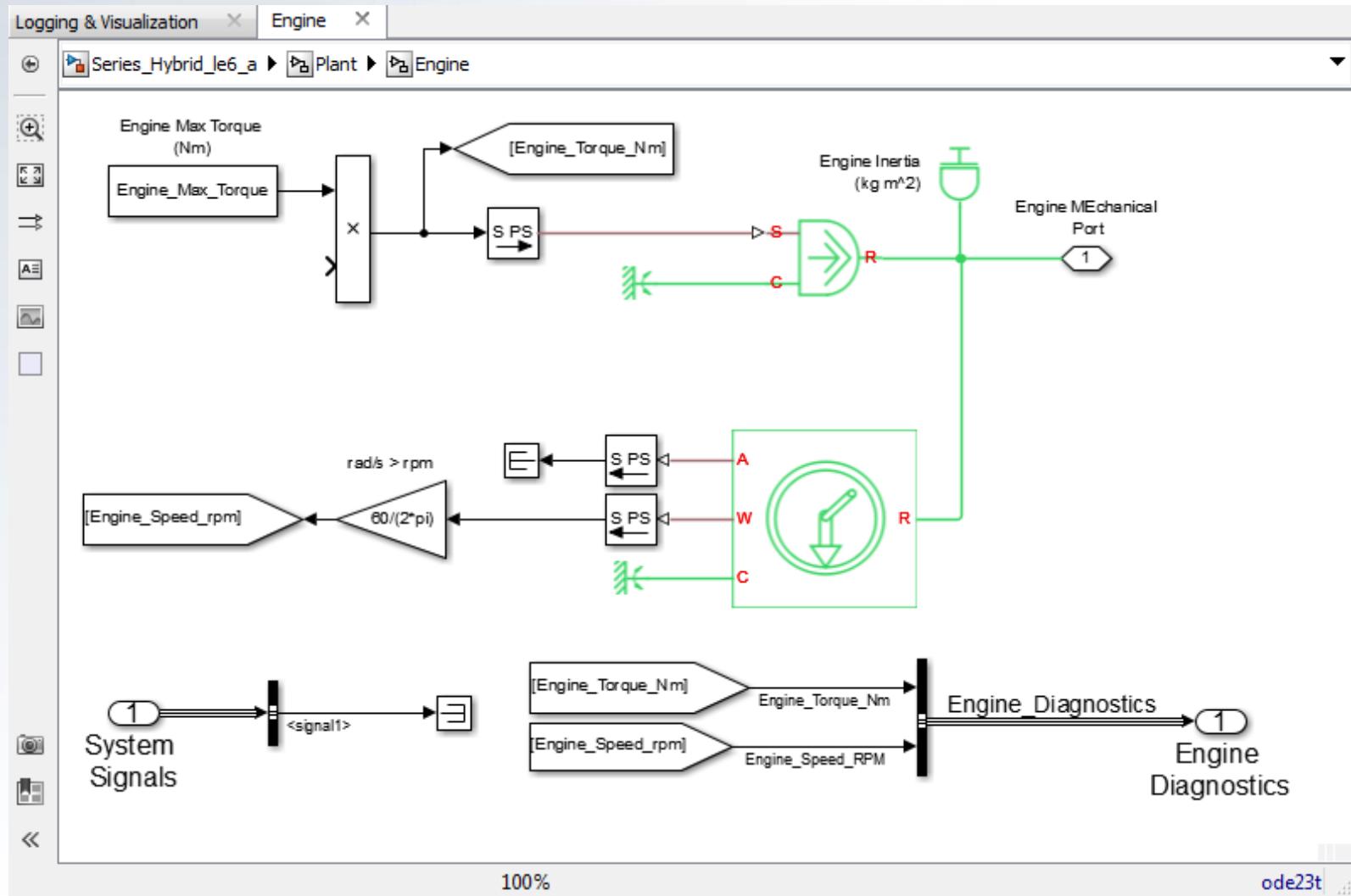


Engine

- Our engine torque will be modulated with a throttle request from the Controller
 - Range of 0 to 1
 - Analogous to a driver stepping on the accelerator pedal
- Place the engine torque and speed on the Diagnostic bus



Engine



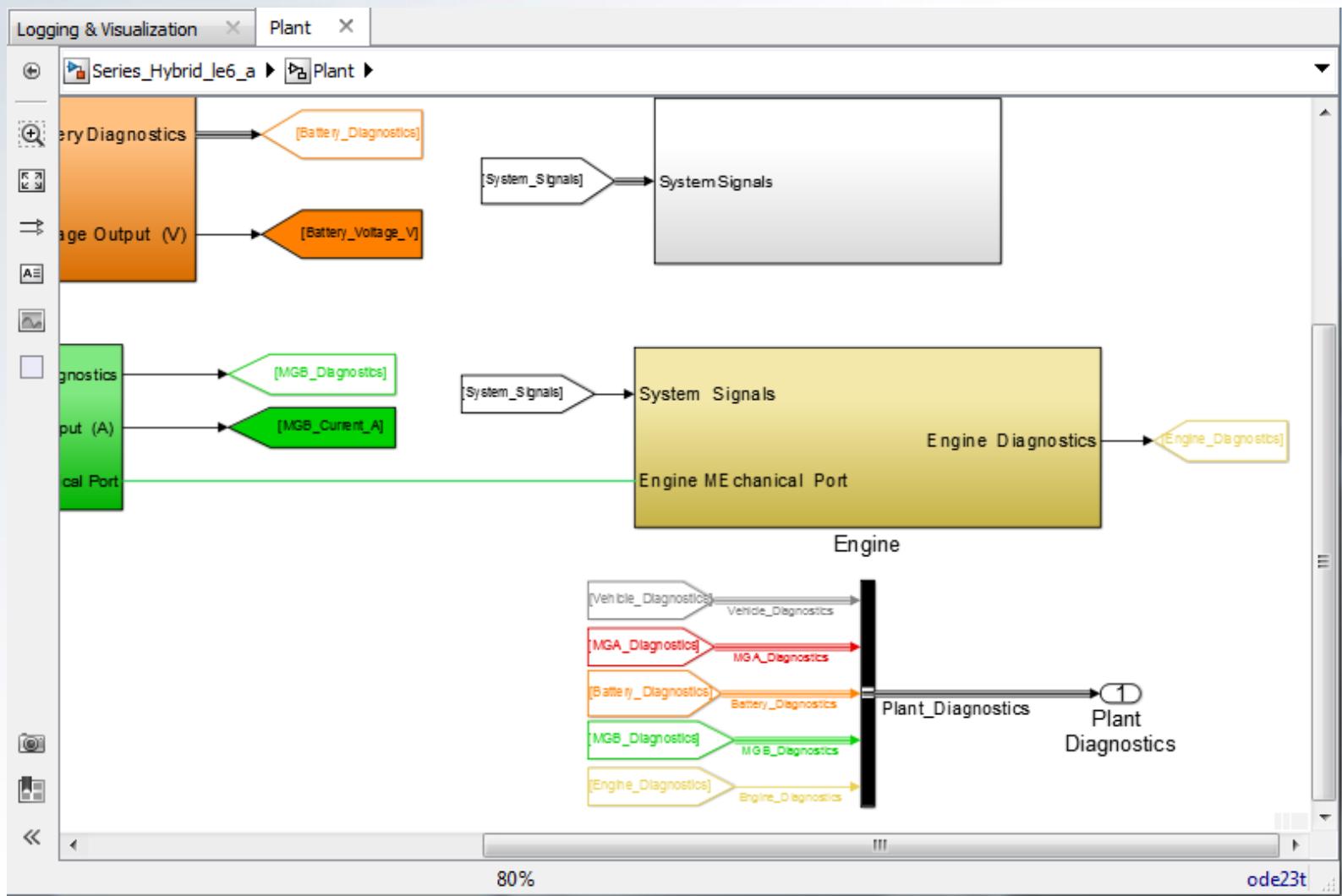


Engine

- Return to the Plant level of the model
- Connect a System Signals **From** block
- Place the Engine Diagnostics on the CAN Bus
- Connect the Engine mechanical port to the MGB mechanical port
- Go to the Logging & Visualization and extract the Engine diagnostics

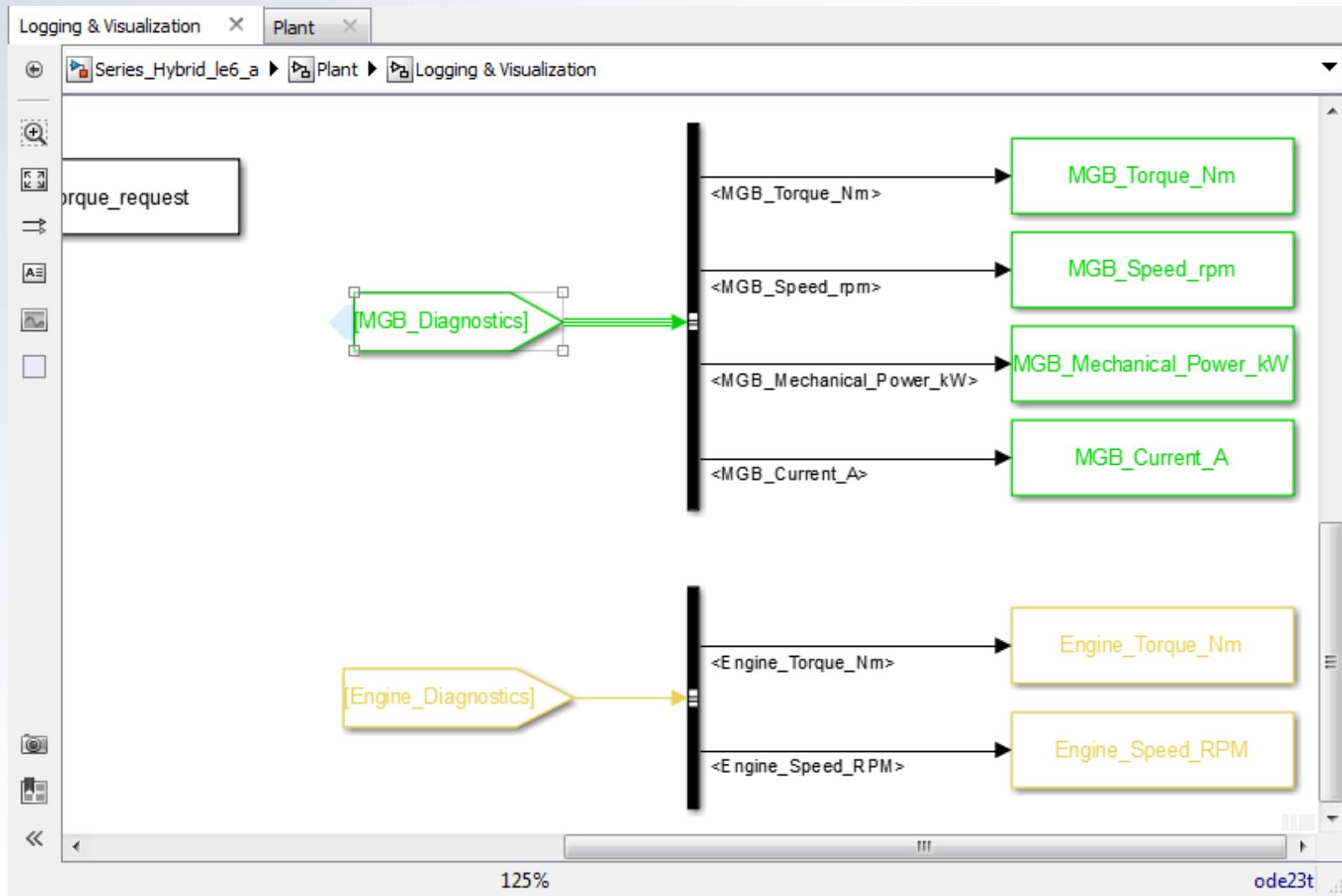


Engine





Logging & Visualization





Engine

- Now place the Engine parameters into the init file
 - $\text{Engine_Max_Torque} = 200 \text{ Nm}$
 - $\text{Engine_Inertia} = 1.5e-3 \text{ kg m}^2$

A screenshot of a software interface, likely MATLAB or a similar environment, showing a script editor window titled "Vehicle_Init_File_L3.m". The window has tabs for "EDITOR", "PUBLISH", and "VIEW", with "EDITOR" selected. The toolbar includes buttons for New, Open, Save, Compare, Print, Go To, Find, Breakpoints, Run, Run and Advance, Advance, and Run and Time. The script code is as follows:

```
18
19 %Battery Parameters
20 - Battery_Voltage = 336; %Battery Voltage, V
21 - Battery_Capacity = 8.5; %Battery Max Capacity, Ahr
22 - Battery_Initial_SOC = 0.7; %Battery initial SOC
23
24 %Engine Parameters
25 - Engine_Max_Torque = 200; %Engine Max Torque, Nm
26 - Engine_Inertia = 1.5e-3; %Engine Inertia, kg m^2
27 -
28 %Controller Parameters
29 - battery_soc_low = 0.6; %Battery SOC low value
```

The status bar at the bottom shows "script" in the first field, "Ln 27 Col 1" in the second, and three small icons in the third.

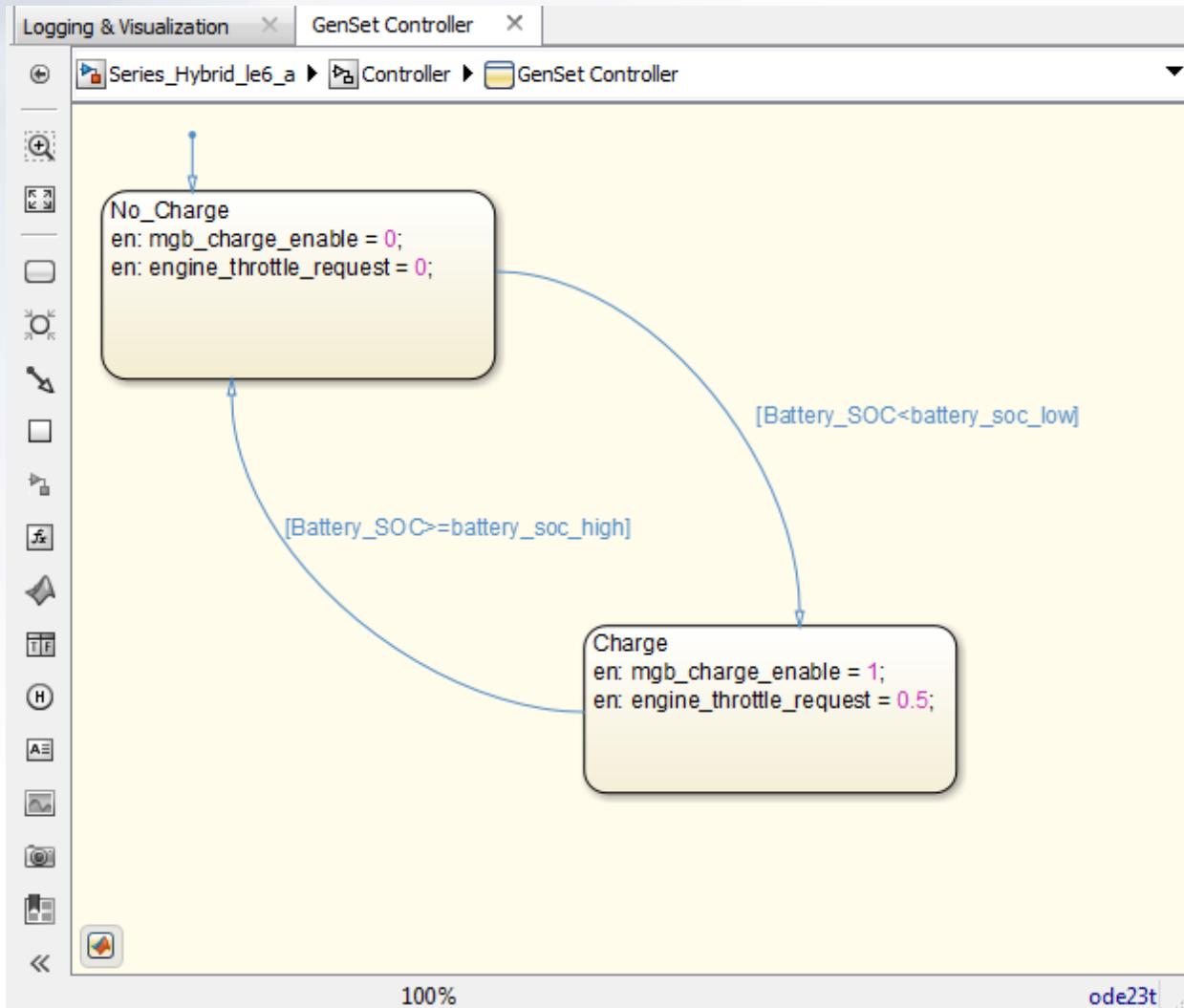


Engine / Controller

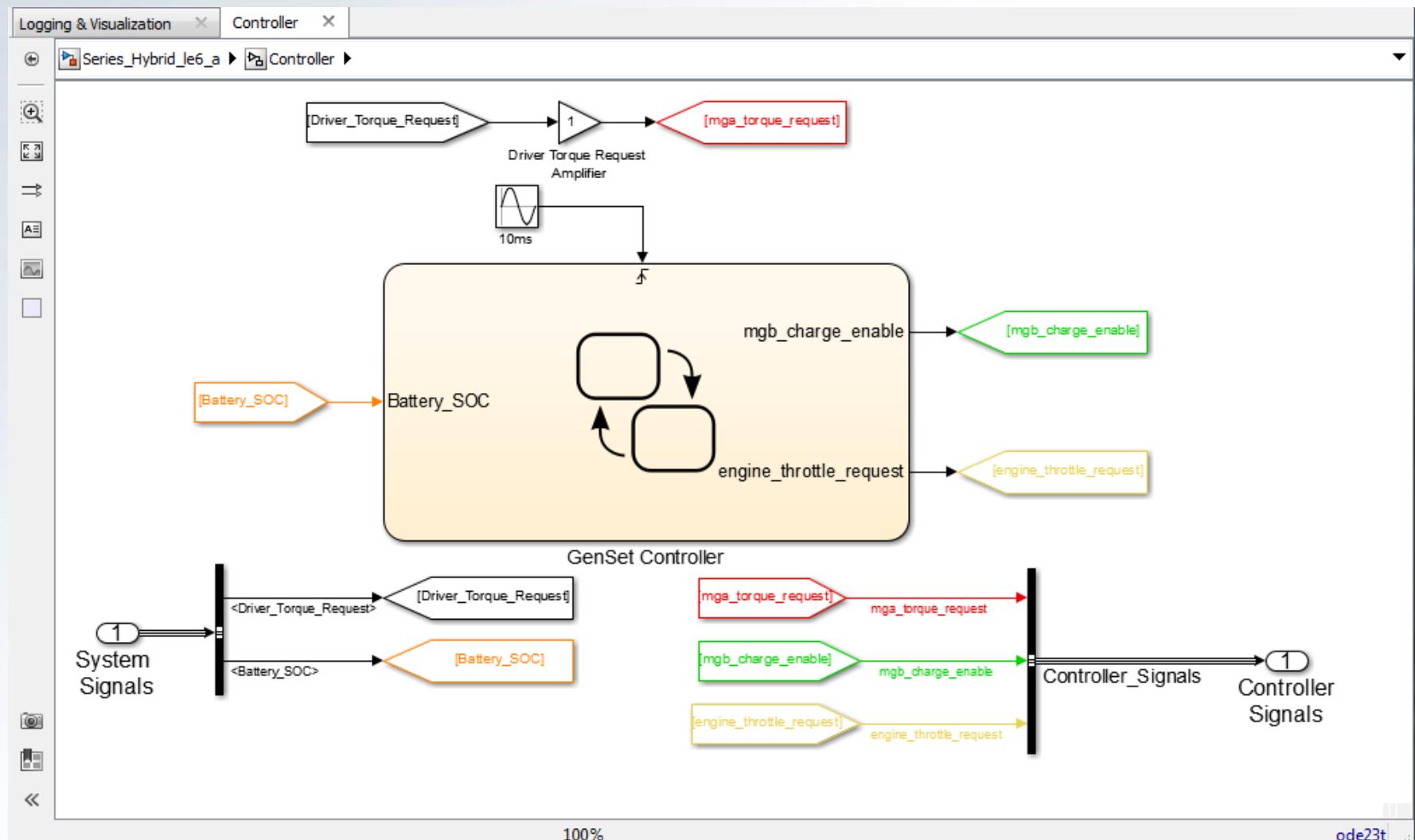
- To keep things simple
 - No Charge State
 - MGB charge enable = 0
 - Engine throttle request = 0
 - Charge State
 - MGB charge enable = 1
 - Engine throttle request = 0.5
- Add the new engine throttle data to the Stateflow Controller
- Place the engine throttle signal on the bus



Controller



Controller



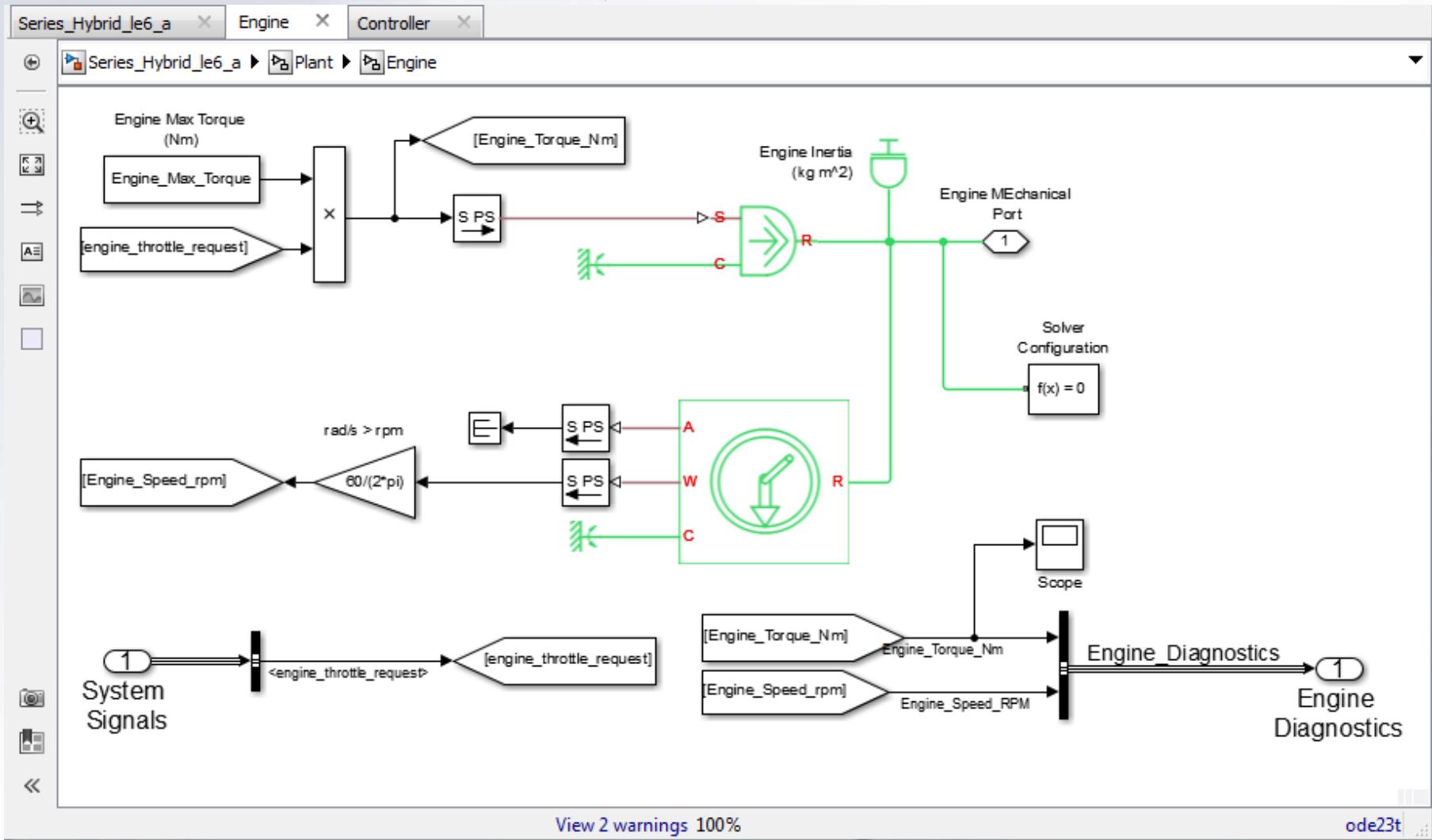


Engine / GenSet

- Return to the Engine subsystem and extract the throttle request signal
- Excellent, the first pass Engine and associated improved GenSet is complete
- Drag in a **Solver Configuration** block and connect the block as shown in the next slide



Engine



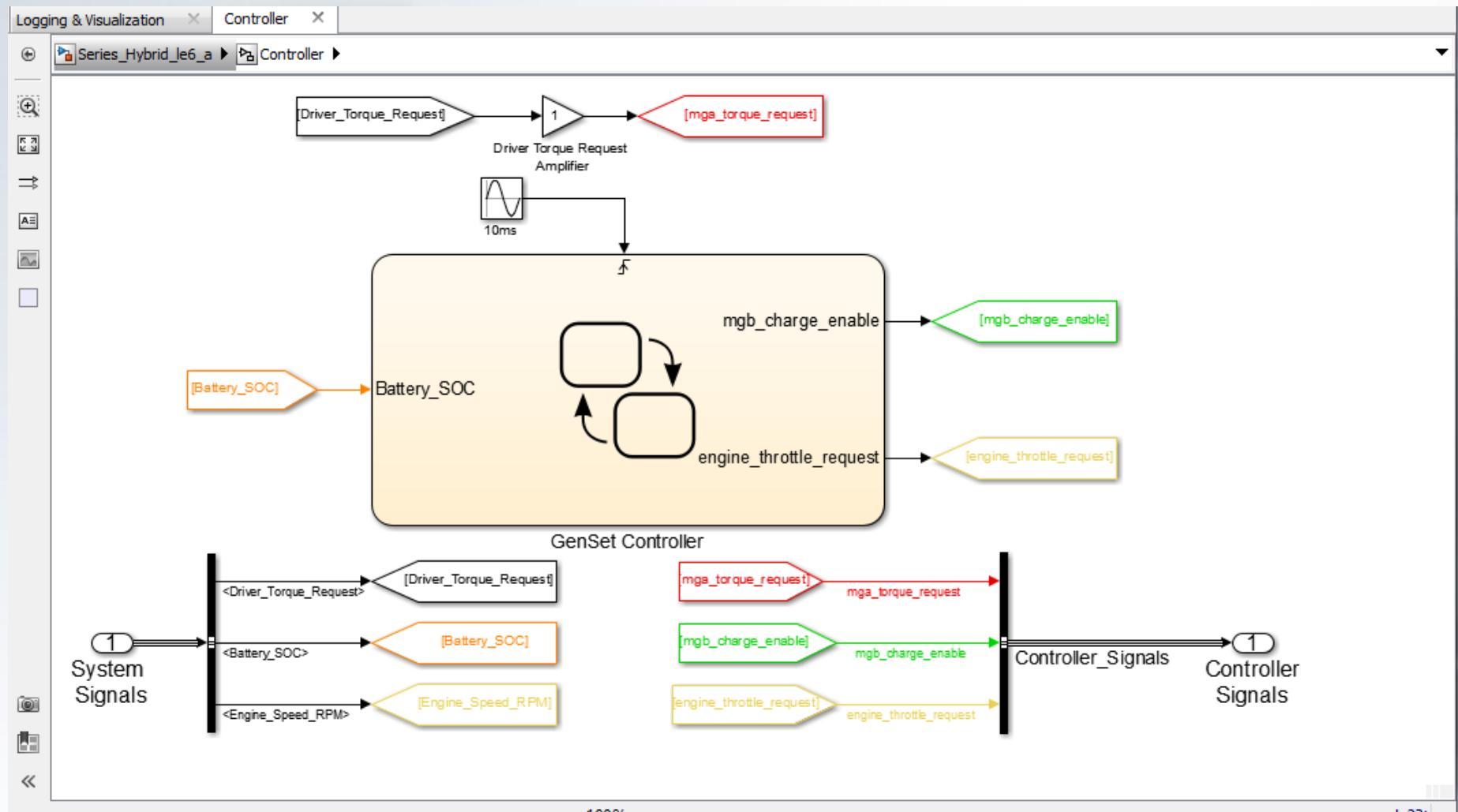


Controller

- The controller will need to send a torque request to MGB to hold the engine at a constant desired speed
 - Engine too fast – increase torque
 - Engine too slow – decrease torque
- Go to the Controller level of the model
- Extract the engine speed signal



Controller



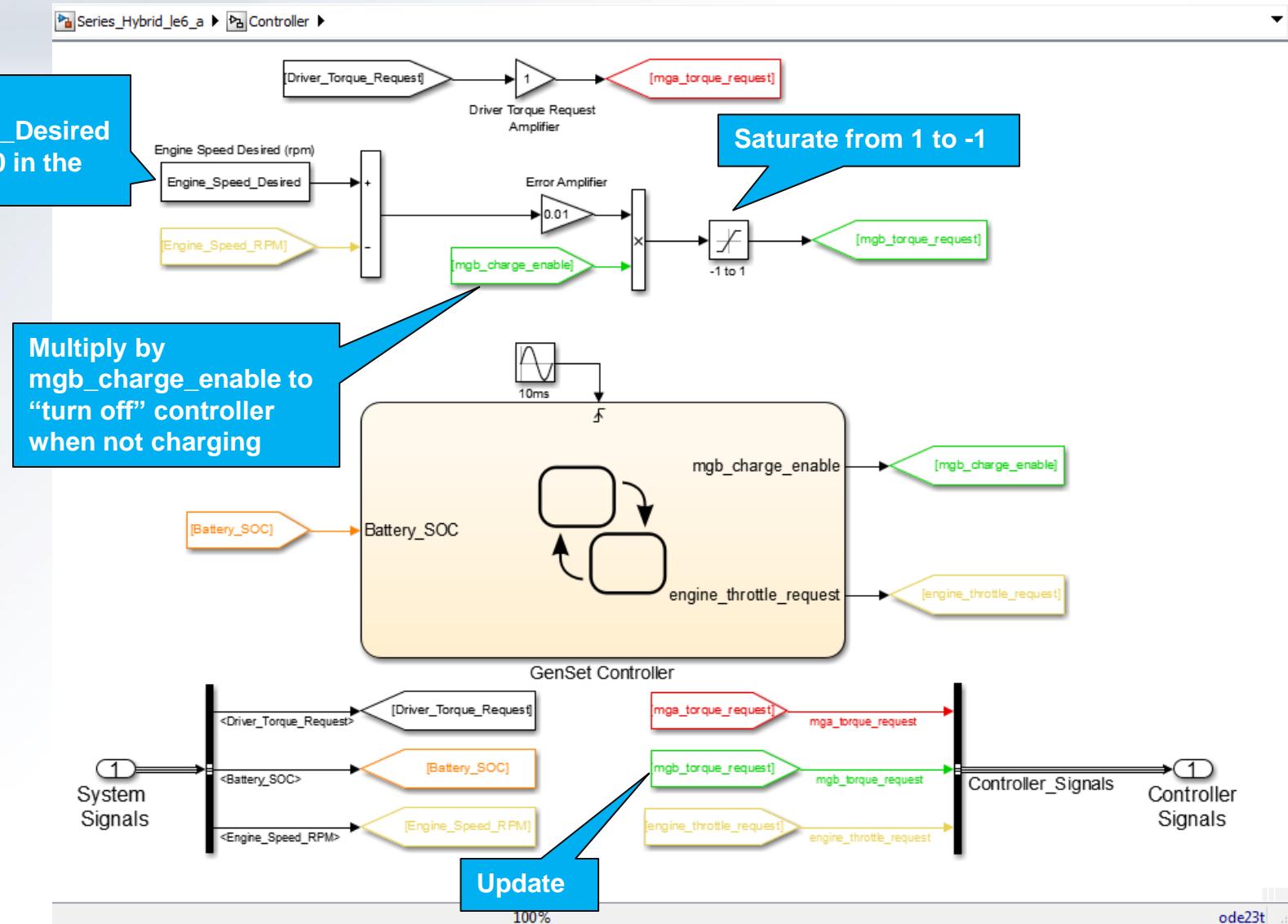


Controller

- As done with the Driver subsystem
 - The actual engine speed will be subtracted from the desired speed
 - This will create an error signal
 - The error signal will be amplified
 - The amplified signal will be saturated
 - The resulting torque request will be sent to MGB
- Implement this feedback controller
- Update the Control Signals bus

Controller

Give the
Engine_Speed_Desired
a value of 1800 in the
init file





MGB and Visualization

- Go to the MGB subsystem and extract the mgb_torque_request signal
- Go to the Logging & Visualization subsystem
 - Right click on the Vehicle_Speed_mph signal
 - Open the Scope
 - Add the MGB_Current_A signal to the third axis
 - Create a fourth axis and put the Engine_Speed_RPM signal on it

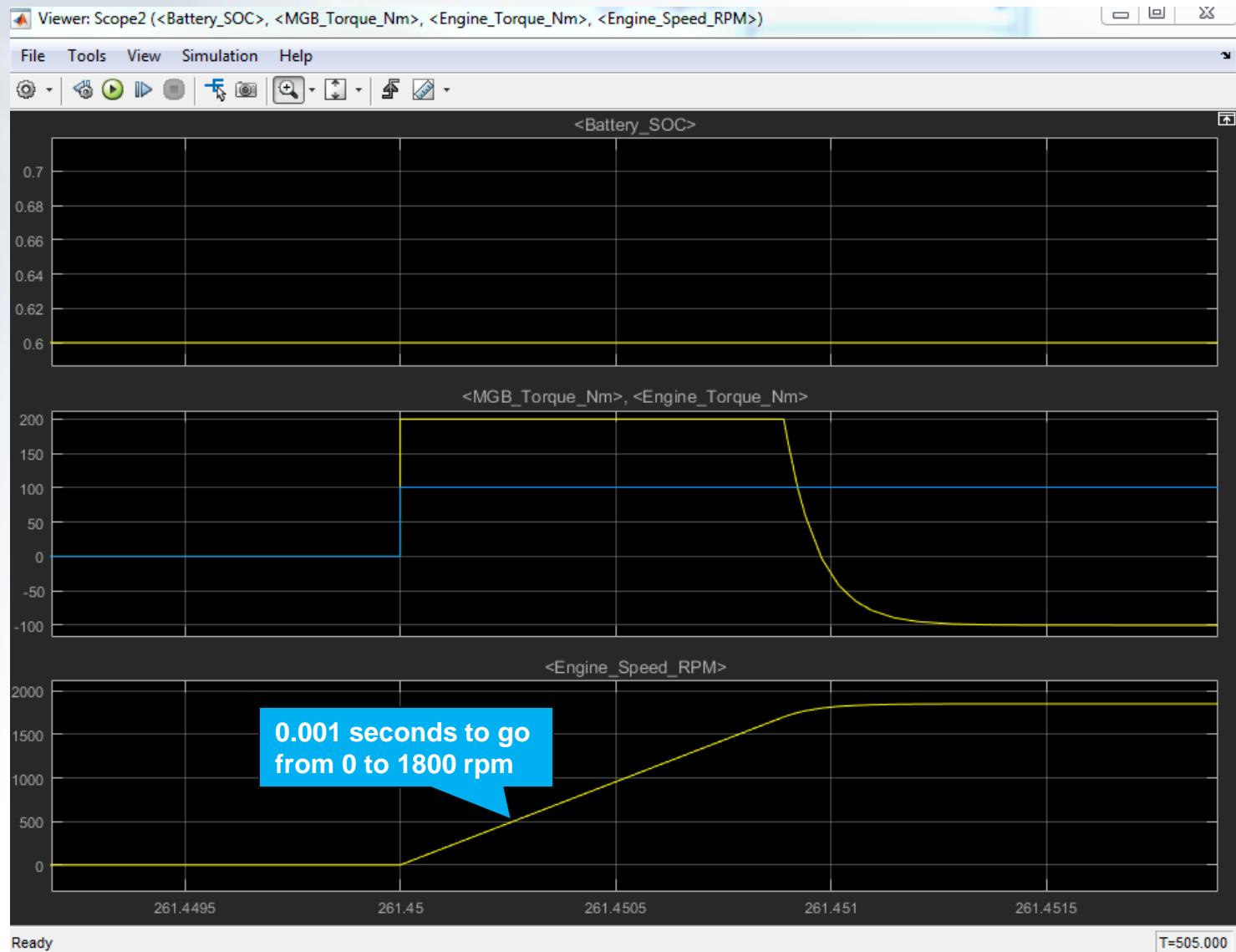


Visualization

- Create a new scope
 - Axis 1 : Battery SOC
 - Axis 2 : Engine Torque and MGB Torque
 - Axis 3 : Engine Speed
- Run the FU505 cycle to save time



Results





Improved GenSet & Controls

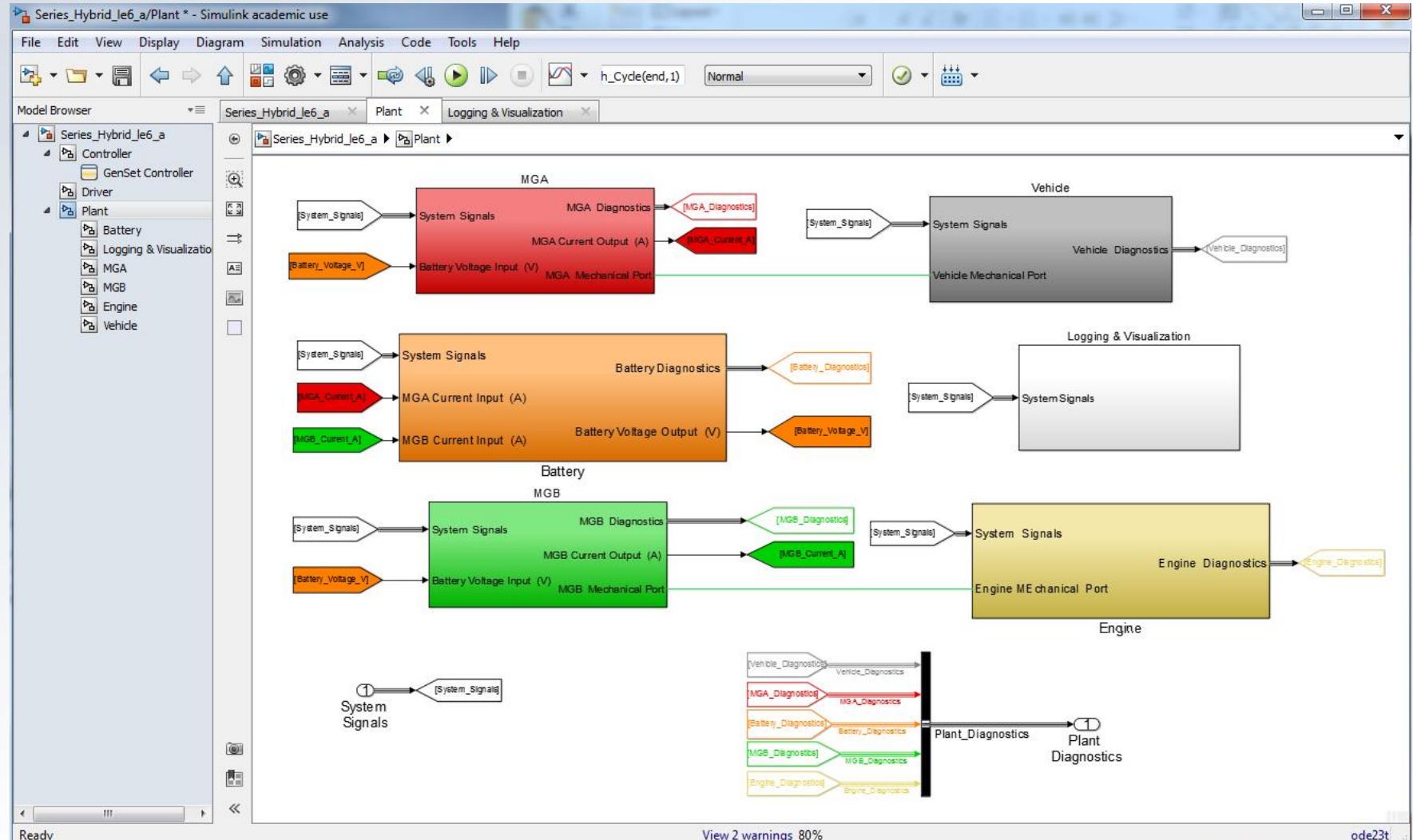
- Review
 - Made MGB a duplicate of MGA
 - Created a very simple diesel engine
 - Updated the controller to intermittently obtain current from the GenSet



Lectures 0 - 6

- We built a model of a series electric vehicle
 - Plant
 - Driver
 - Controller
- All of the plant components are crude
 - System response could be intuited
 - Complex system was established
- Component refinement is next
 - Controller will need to be updated

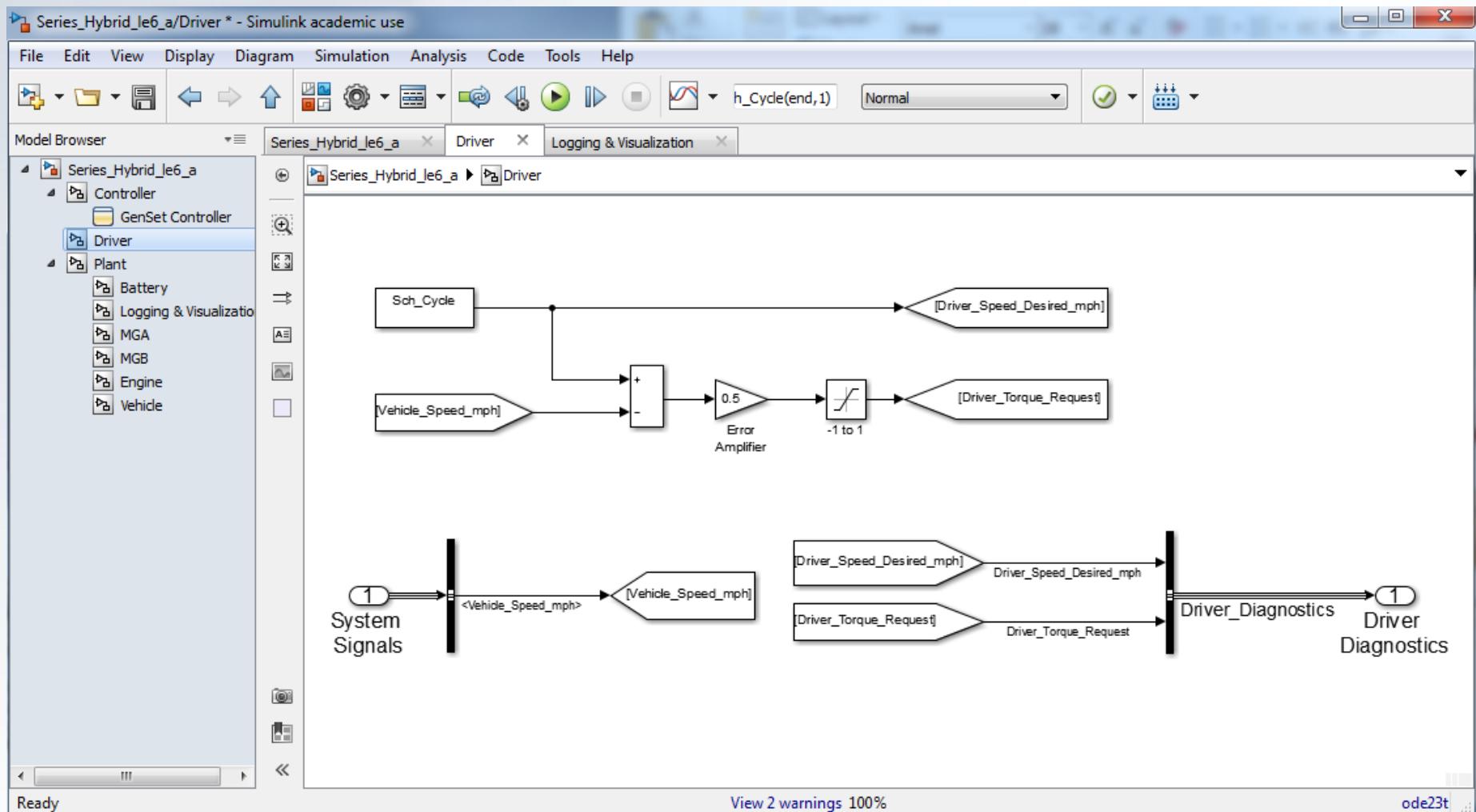
Lectures 0 - 6



The high level plant model will not change!

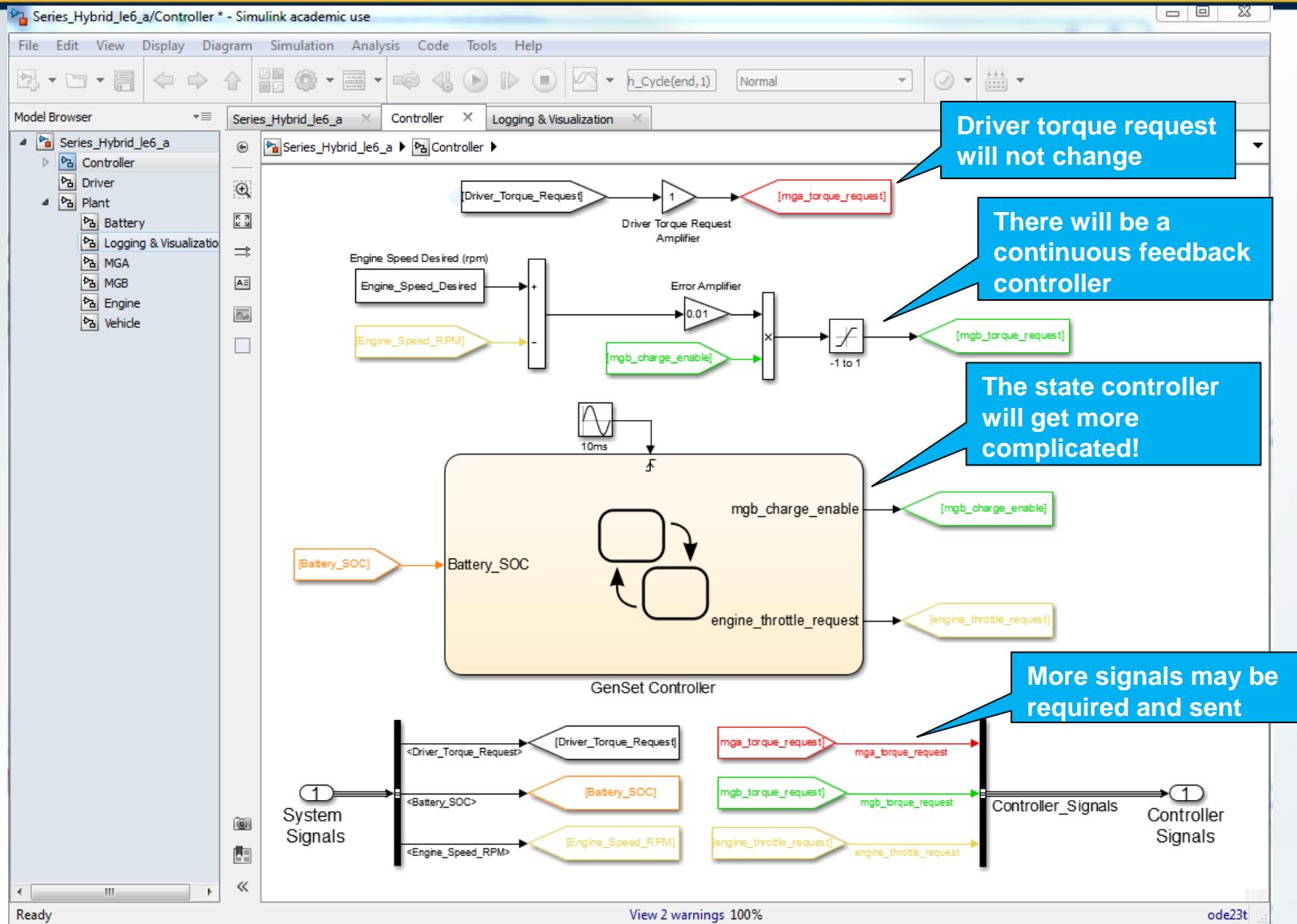


Lectures 0 - 6



The Driver will not change!

Lectures 0 - 6



A blue-toned photograph of the Georgia Tech Campanile, a tall brick tower with a spire and the words "THE TECH" on its side.

MBSD Lecture 7

Improved Engine Model and
MPGGE SOC Fuel Efficiency



Outline

- Improve Engine model
 - Experimental data for torque curves
 - Experimental data for fuel consumption
 - Controller refinements
- Fuel Efficiency
 - Calculate MPG, MPGGE, and MPGGE with SOC correction



Experimental Data

- Using the engine dynamometer at Rose-Hulman, a small diesel engine was tested at various engine speeds and throttle positions
- Data for torque (Nm) and fuel consumption (grams/sec) were collected
- Let's add this to our model
- Rename your model
 - [Series_Hybrid_le7_a.slx](#)



Experimental Data

- In the Matlab Window, go to
 - Current Directory window
 - Double click on the Component_Data folder
- At the command line, type
 - clear variables
 - load Engine_Diesel_Data
- The loaded variables will appear in the Workspace window



Experimental Data

MATLAB R2016a - academic use

HOME PLOTS APPS

New Script New Open Compare Import Data Save Workspace Clear Workspace New Variable Open Variable Run and Time Clear Commands Analyze Code Simulink Preferences Set Path Parallel Add-Ons Layout ENVIRONMENT

FILE VARIABLE CODE SIMULINK RESOURCES

C: > Users > sologhavi3 > Desktop > ME4013_Tutorial > Vehicle_model > Component_Data

Current Folder

Name

- Battery_Data.mat
- Battery_Data.xls
- Engine_Diesel_Data.mat
- Fantasy Motor_Data.xls
- MG_Data.mat
- Scaled_Diesel_Engine.xls

Command Window

```
>> clear variables
>> load Engine_Diesel_Data
fx >>
```

Workspace

Name	Value
Engine_Fuel_Data	15x21 double
Engine_Fuel_RPM_Axis	15x1 double
Engine_Fuel_Throttle_Axis	1x21 double
Engine_Torque_Data	15x21 double
Engine_Torque_RPM_Axis	15x1 double
Engine_Torque_Throttle_Axis	1x21 double

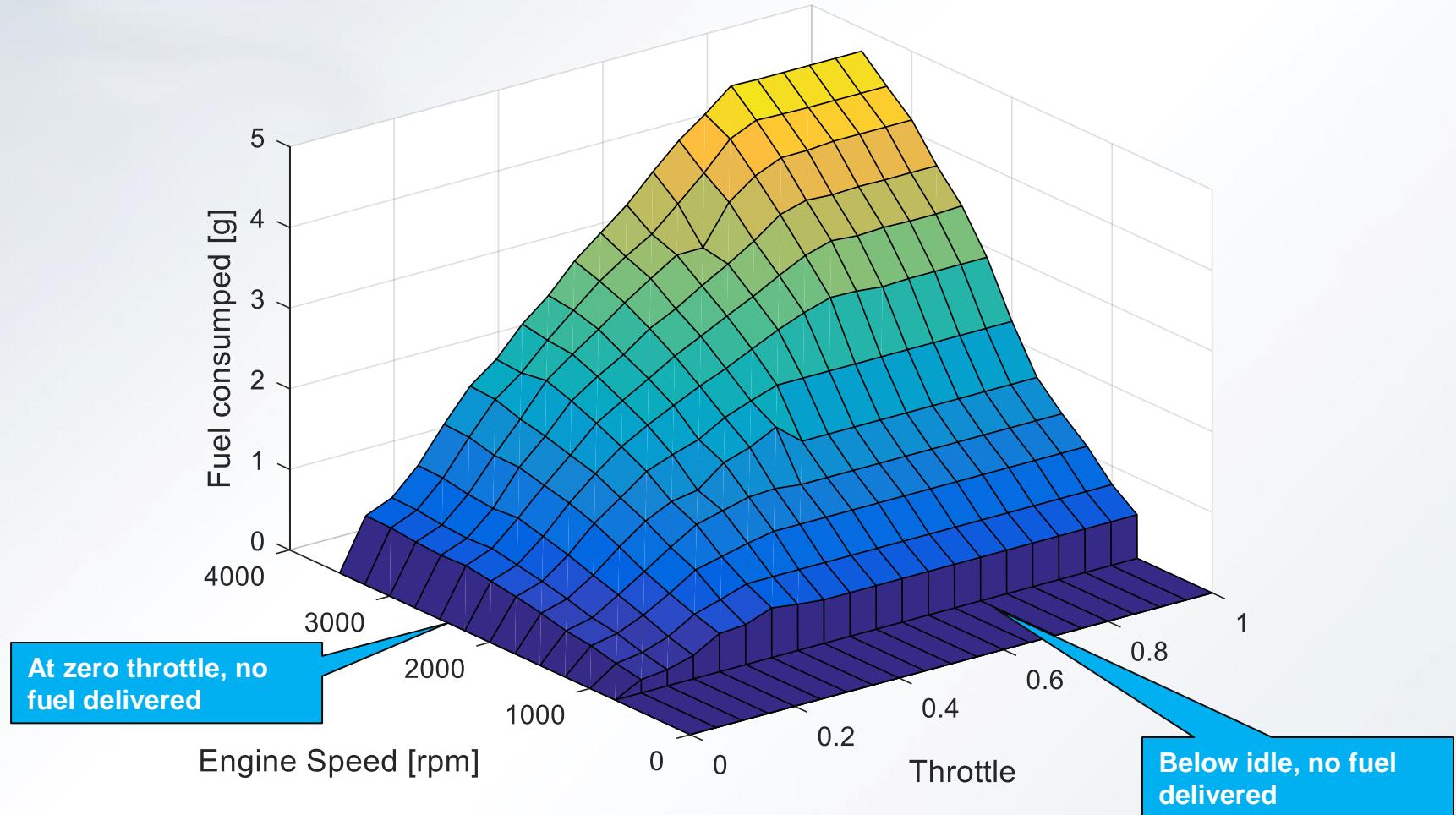
Experimental Data



- Note the size of the matrices
 - Engine_Fuel_Data : 15x21
 - Engine_Fuel_RPM_Axis: 15x1
 - Engine_Fuel_Throttle_Axis: 1x21
 - Thus
 - The data rows are for given rpms
 - The data columns are for given throttle positions
 - At the command line type
 - `surf(Engine_Fuel_Throttle_Axis, Engine_Fuel_RPM_Axis, Engine_Fuel_Data)`



Experimental Data

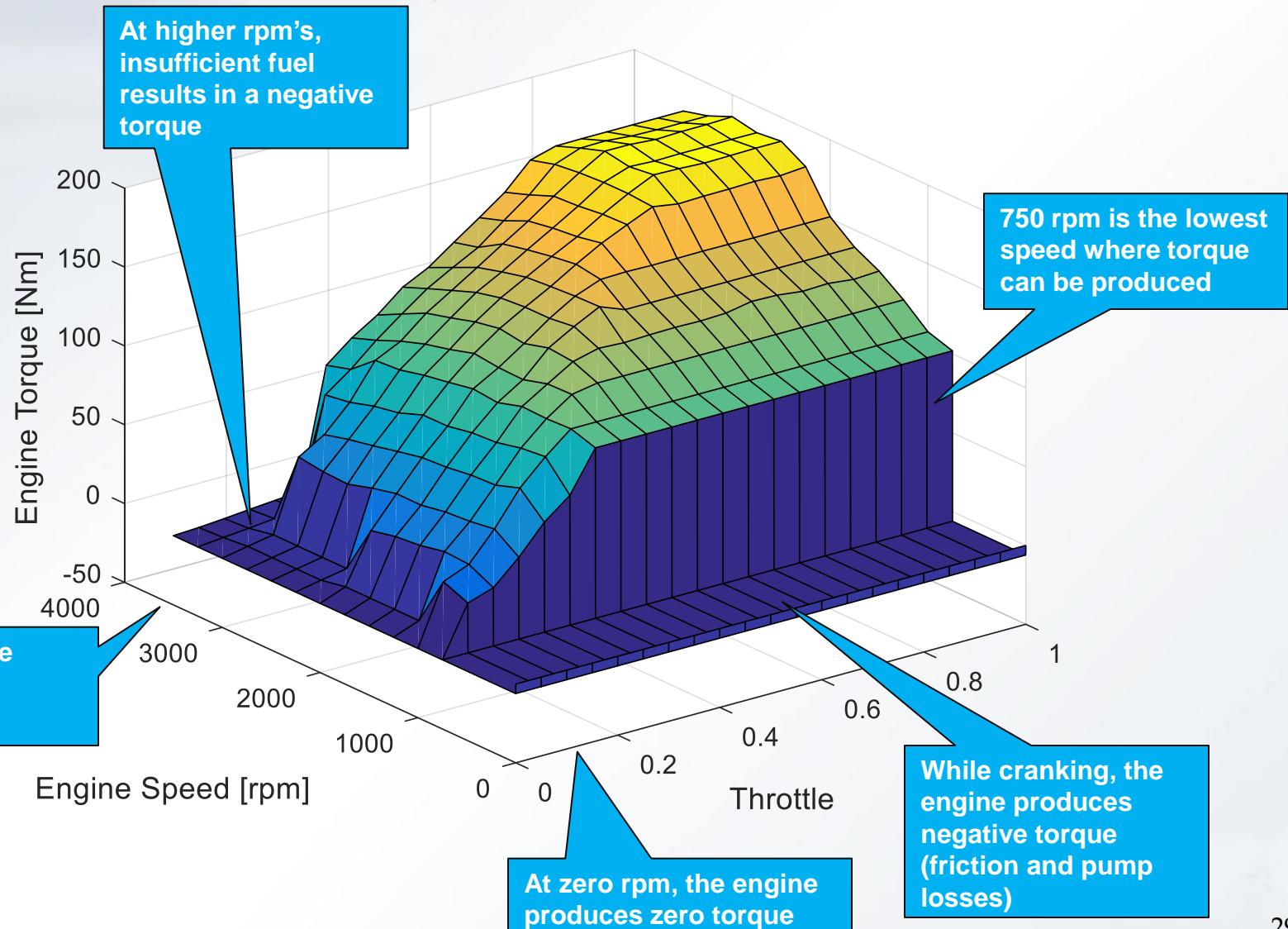




Experimental Data

- This engine does not have an idle circuit
 - 750 rpm is the lowest it can run
 - A throttle signal of zero will send no fuel
 - It will stall out
- This is great!
 - Our controller completely rules the throttle signal
- Modify the previous surf command to view the torque data

Experimental Data

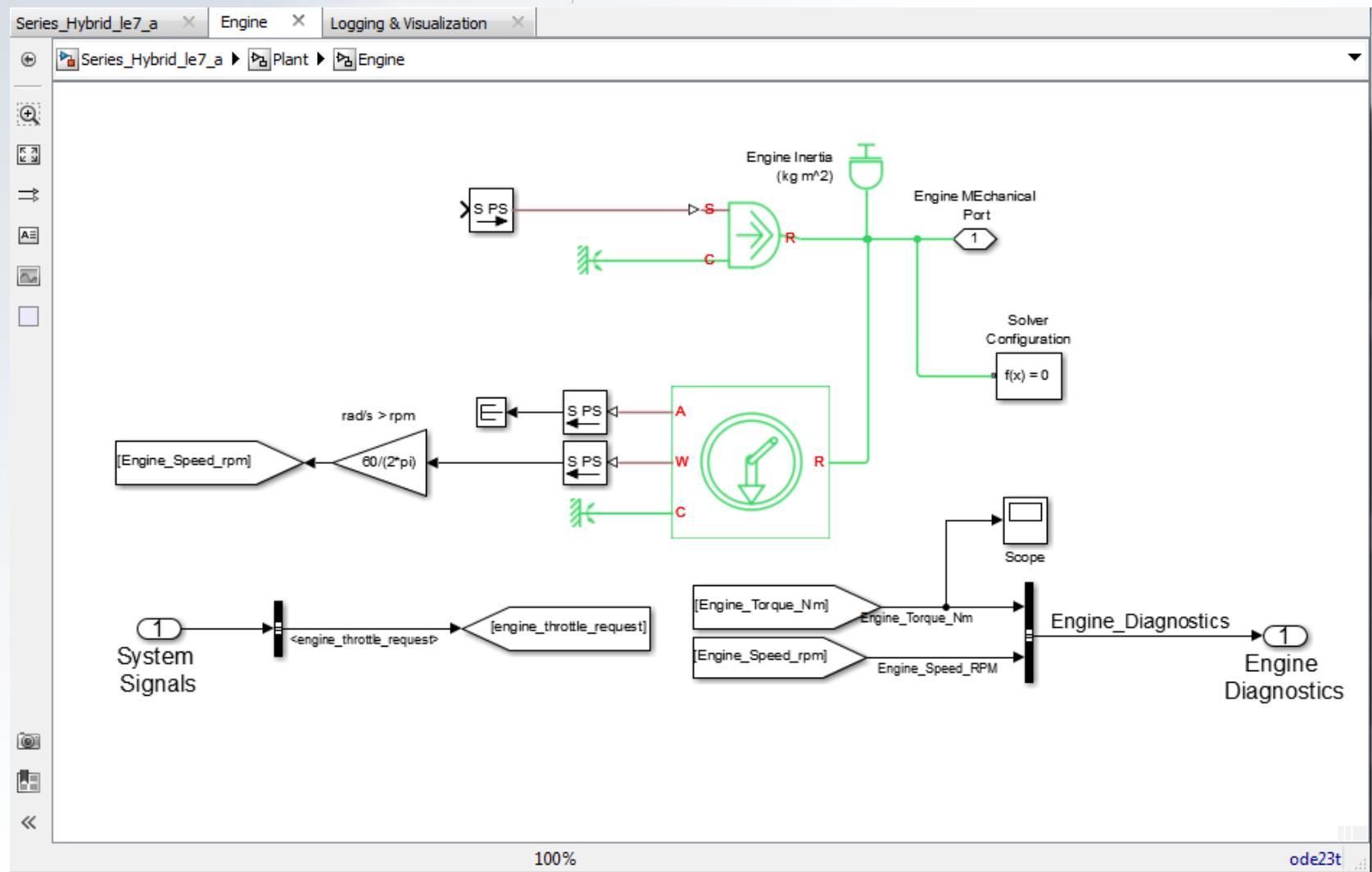




Engine

- We will need to read in this data with a 2-d lookup table
- Go to the Engine subsystem and delete the
 - Engine_Max_Torque Constant
 - engine_throttle_request From
 - Product block
 - Engine_Torque_Nm Goto

Engine



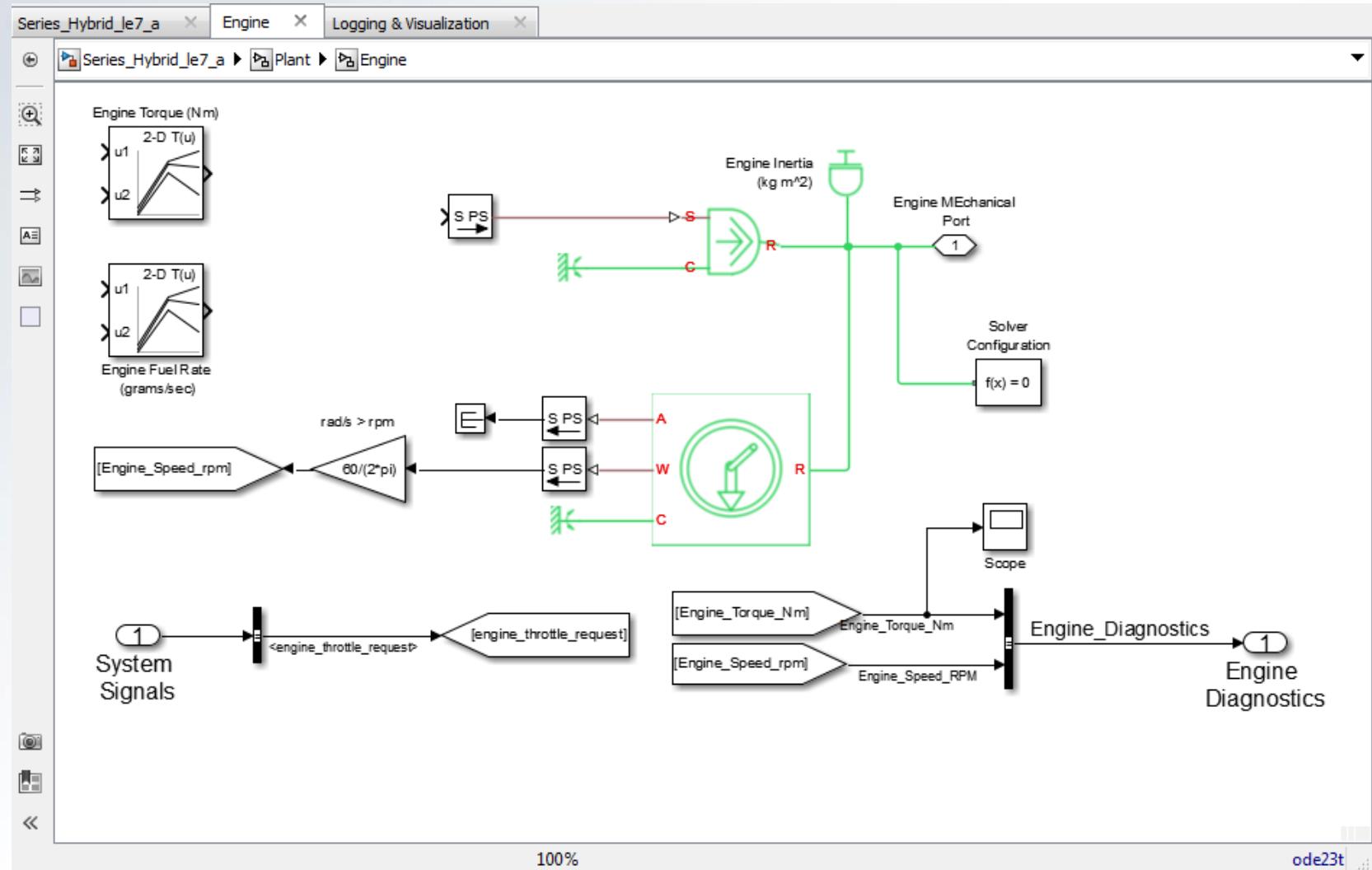


Engine

- From
 - Simulink / Lookup Tables
- drag in two
 - 2-D Lookup Table blocks
- Rename them
 - Engine Torque (Nm)
 - Engine Fuel Rate (grams/sec)



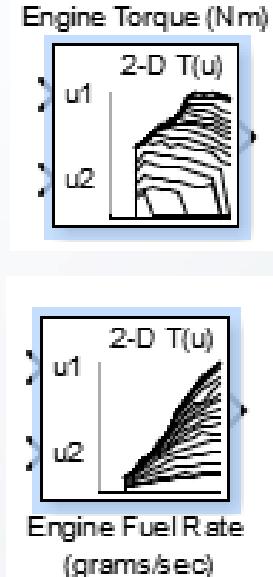
Engine





Engine

- Double click on the Engine Torque table
- In the dialogue box, change
 - Table data: Engine_Torque_Data
 - Breakpoints 1: Engine_Torque_RPM_Axis
 - Breakpoints 2: Engine_Torque_Throttle_Axis
- Click OK
- Note the shape of the table
 - It looks like the data!
- Repeat for the Fuel Rate table





Engine

Block Parameters: Engine Torque (Nm)

Lookup Table (n-D)

Perform n-dimensional interpolated table lookup including index searches. The table is a sampled representation of a function in N variables. Breakpoint sets relate the input values to positions in the table. The first dimension corresponds to the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 2

Table data: Engine_Torque_Data

Breakpoints specification: Explicit values

Breakpoints 1: Engine_Torque_RPM_Axis

Breakpoints 2: Engine_Torque_Throttle_Axis

Edit table and breakpoints...

?

OK Cancel Help Apply

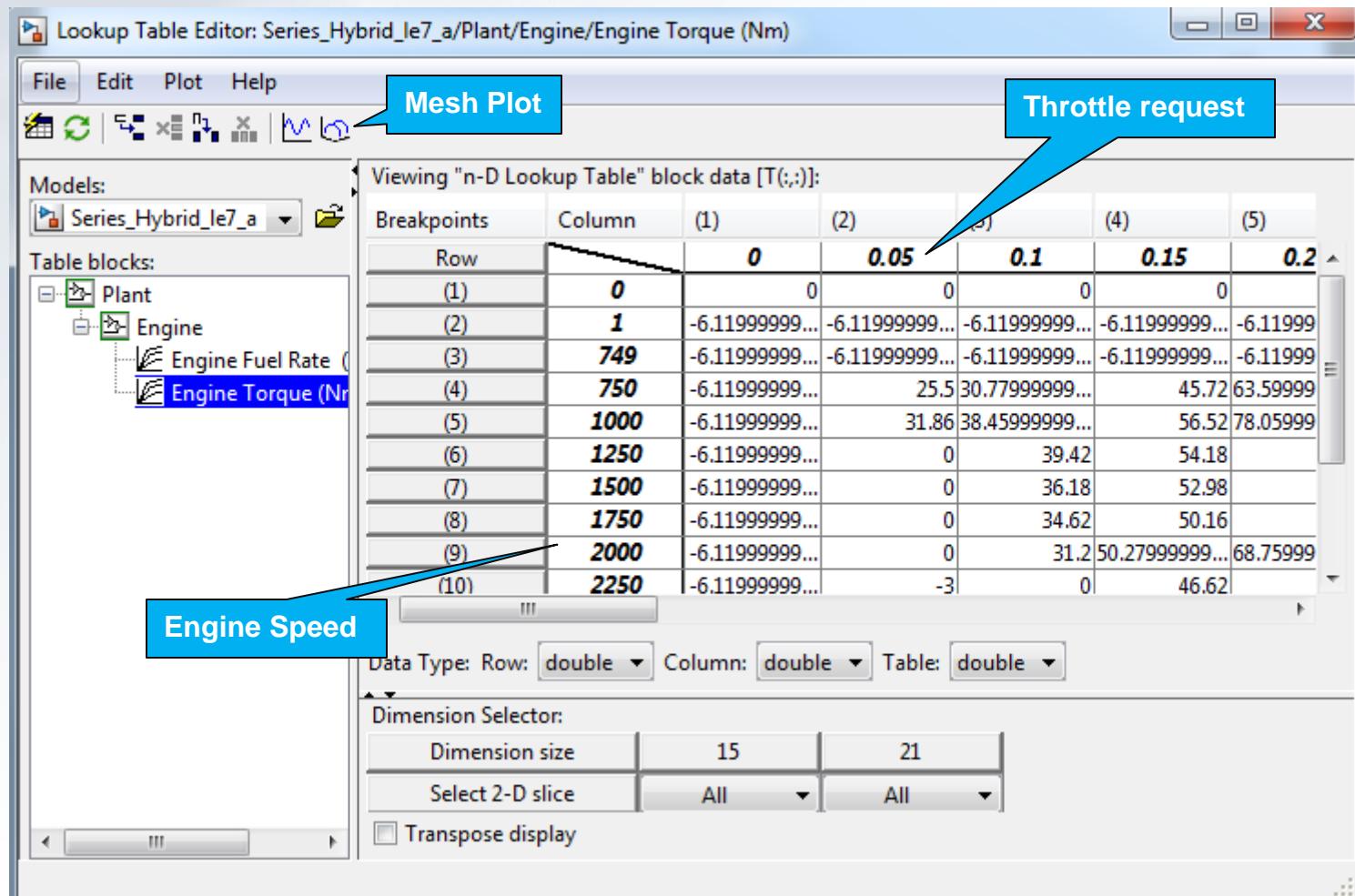
Engine



- Double click on the Engine Torque table
- Click the Edit table and breakpoints... button
 - Note that all the data shows up
- Click on the Mesh Plot icon
- Verify that the shape is correct
 - Easy to make a mistake, especially with square matrix data!
- Close the mesh plot and the Editor
- Click Cancel on the dialogue box



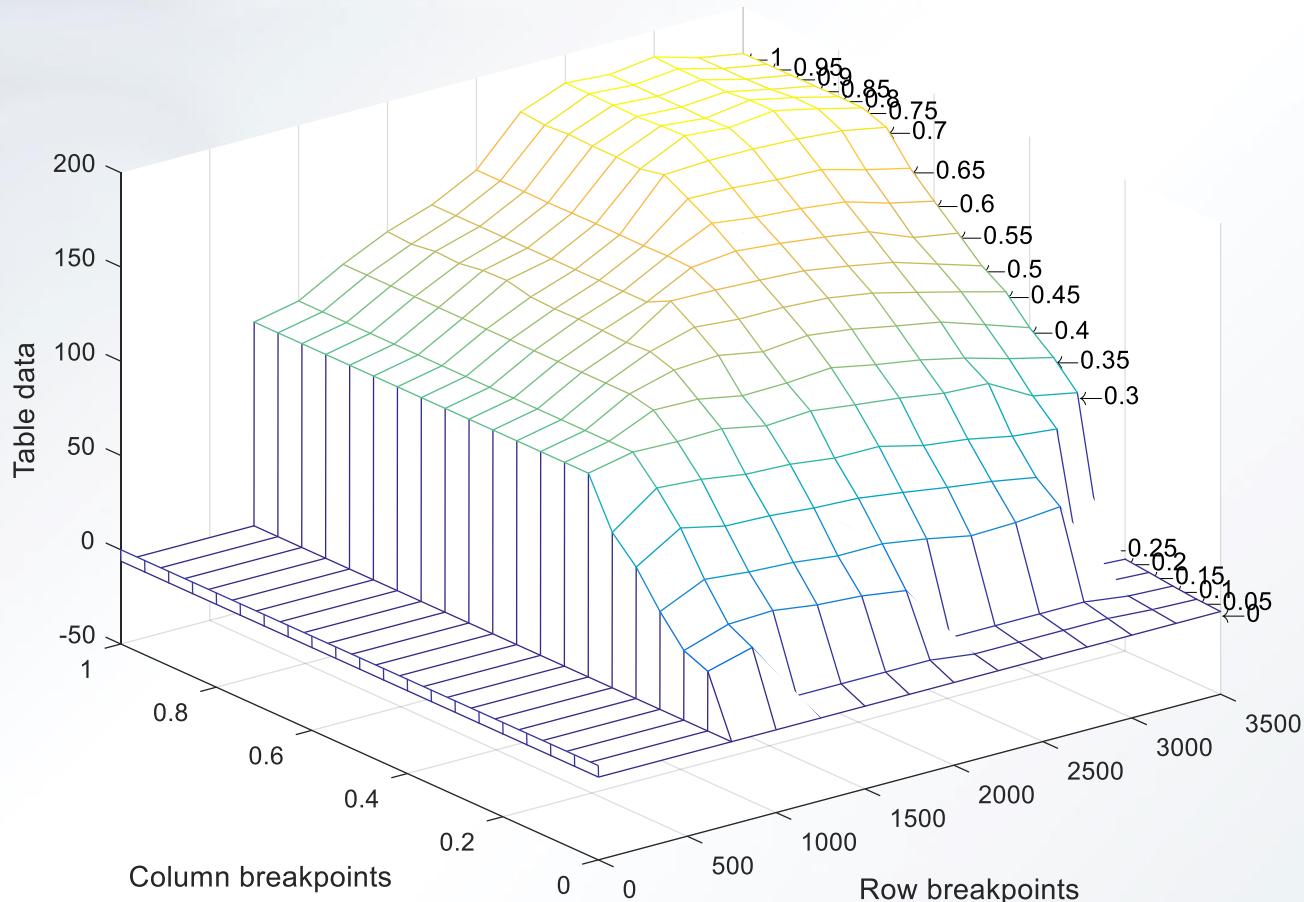
Engine



Engine



Table and breakpoints data for block: Series_Hybrid_le7_a/Plant/Engine/Engine Torque (Nm)

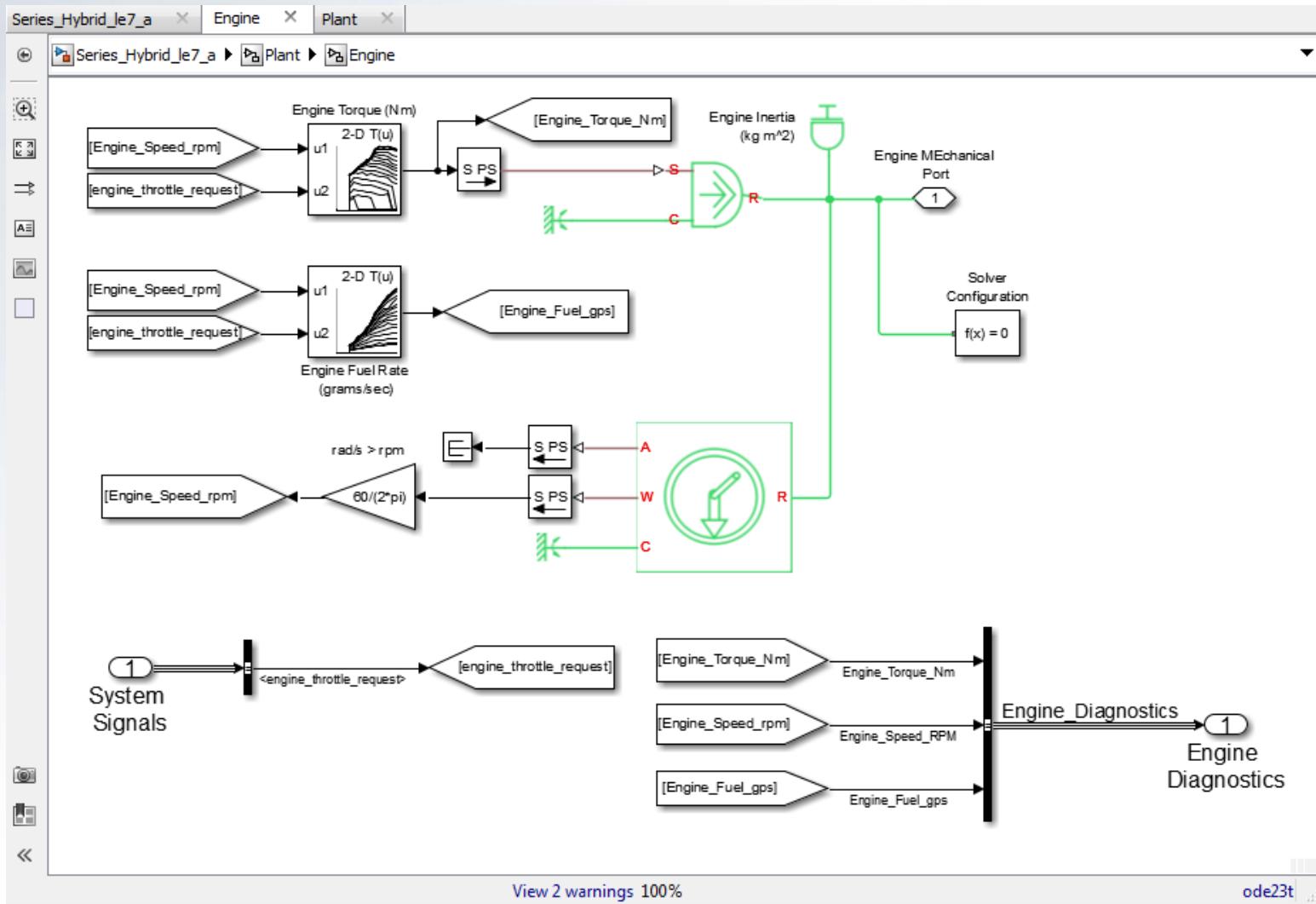




Engine & Logging

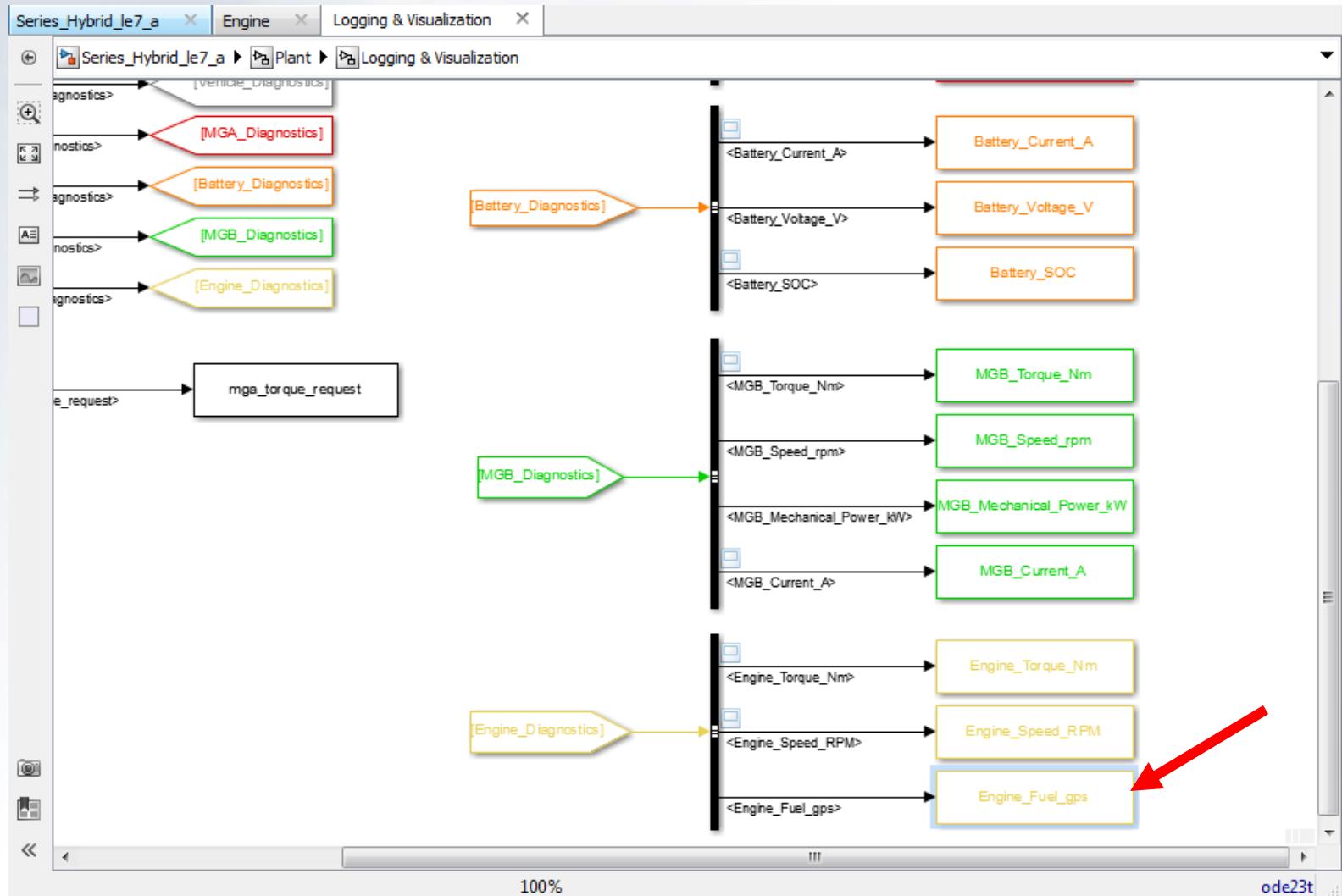
- In the Engine subsystem, route
 - An engine rpm signal to each table
 - Top input
 - A throttle request signal to each table
 - Bottom input
 - The torque output to the **Simulink-PS Converter**
 - The fuel rate output to the Engine Diagnostics bus
 - Put a **Goto** block on the Engine Torque signal:
Engine_Torque_Nm
- In the Logging & Visualization subsystem
 - Extract the engine fuel rate signal

Engine & Logging





Engine & Logging





Engine & Initialization

- In the init file, add the line
 - load Component_Data/Engine_Diesel_Data.mat
- Update the engine inertia to 0.12
 - The crankshaft and flywheel

The screenshot shows a MATLAB script editor window titled "Vehicle_Init_File_L3.m". The window has tabs for "EDITOR", "PUBLISH", and "VIEW". The "EDITOR" tab is selected. The toolbar contains icons for New, Open, Save, Find Files, Compare, Print, Go To, Find, Breakpoints, Run, Run and Advance, Run and Time, and Run Section.

```
%Battery Parameters
Battery_Voltage = 336; %Battery Voltage, V
Battery_Capacity = 8.5; %Battery Max Capacity, Ahr
Battery_Initial_SOC = 0.7; %Battery initial SOC

%Engine Parameters
load Component_Data/Engine_Diesel_Data.mat %Engine Data File
Engine_Inertia = 0.12; %Engine Inertia, kg m^2
Engine_Speed_Desired = 1800; %Desired Engine Speed, rpm

%Controller Parameters
```

The status bar at the bottom indicates "script" in the first field, "Ln 25" in the second, and "Col 43" in the third.



Directory

- Recall that we used the Matlab window to browse into the Component_Data directory
- Matlab MUST be in the same working directory as the Simulink model
- In the Matlab window
 - Click on the Go up one level icon
- Run the Simulink model

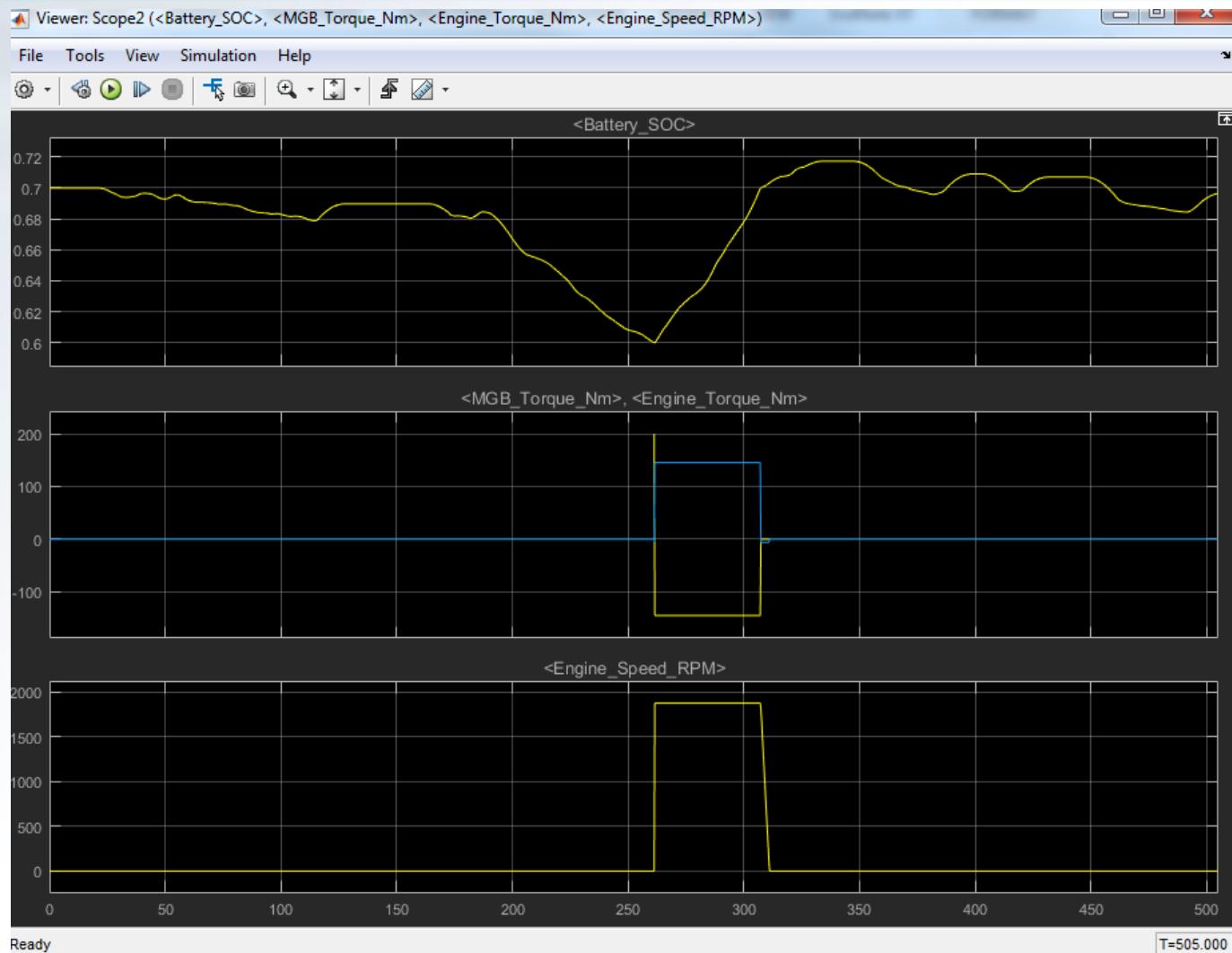


Results

- The results look very similar to those previous
 - The model still appears to be responding correctly
 - At 1800 rpm and 0.5 throttle, the engine torque should be about 150 Nm
 - Verify

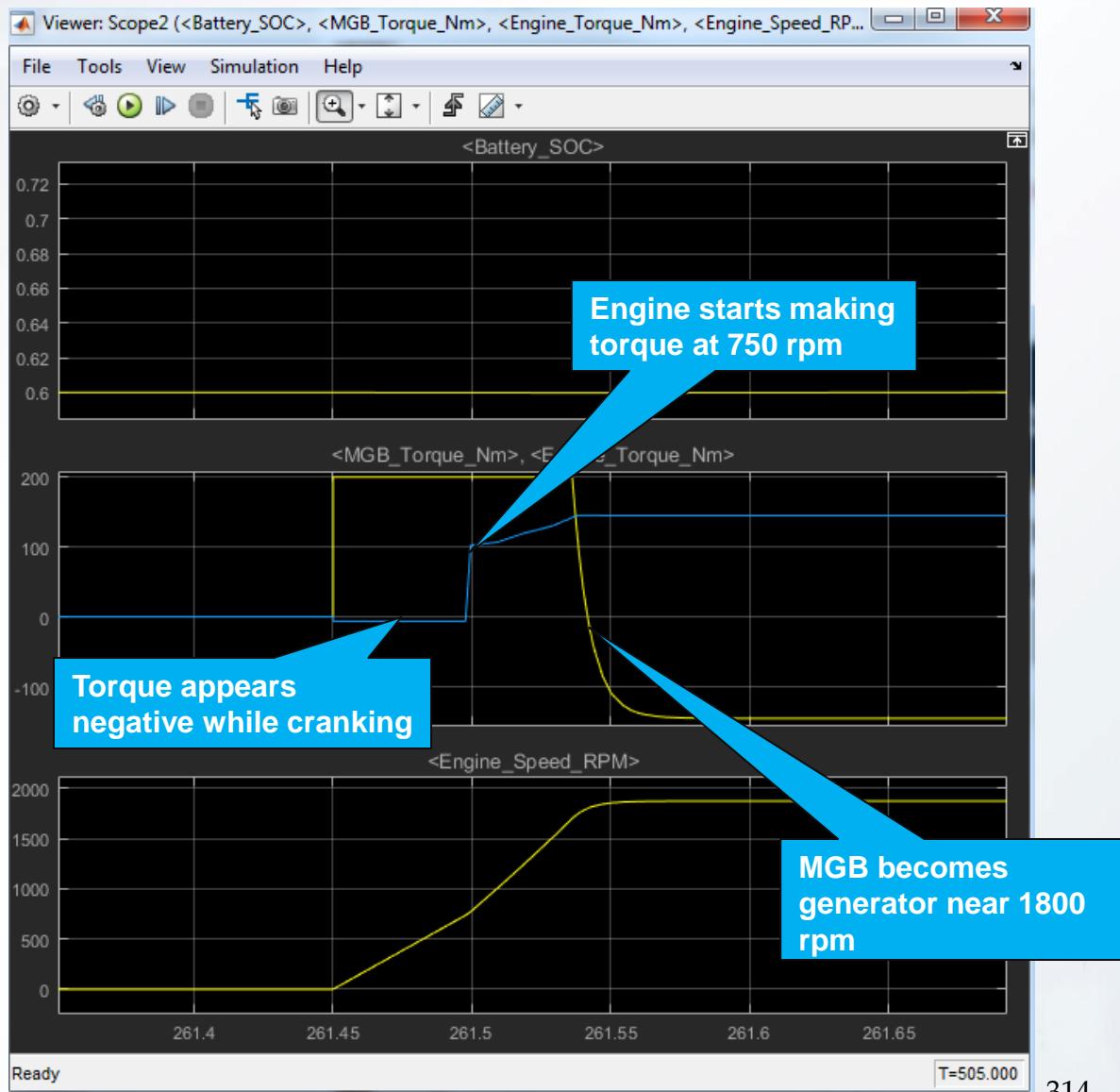


Results



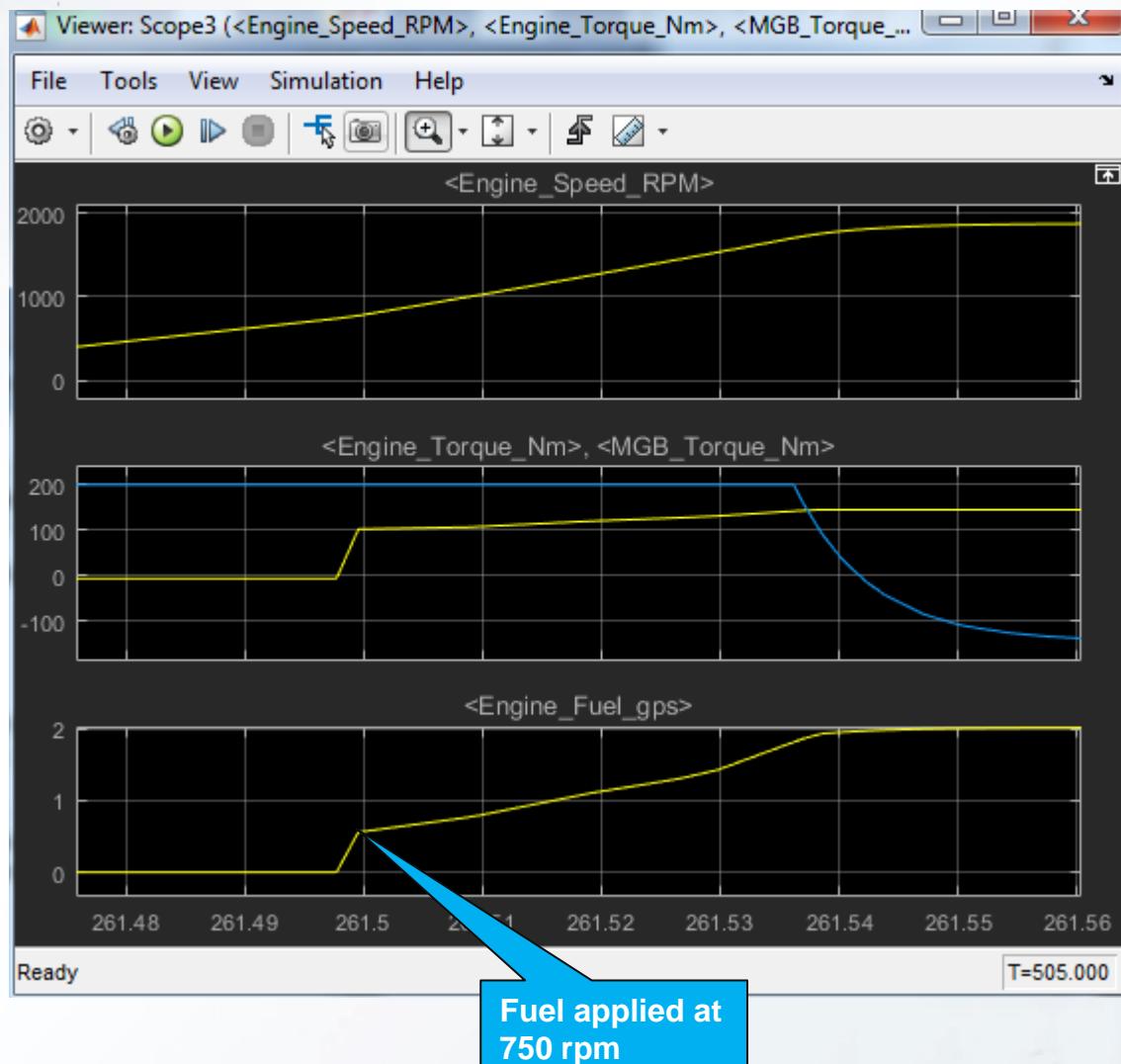
Results

- Use the Zoom icon to zoom in on the time when the engine is starting
 - 261 seconds
- Let's make a new scope for further investigation



Results

- For the new scope
 - Axis 1
 - Engine speed
 - Axis 2
 - Engine torque
 - MGB torque
 - Axis 3
 - Engine fuel rate
- Rerun the simulation





Results & Controller

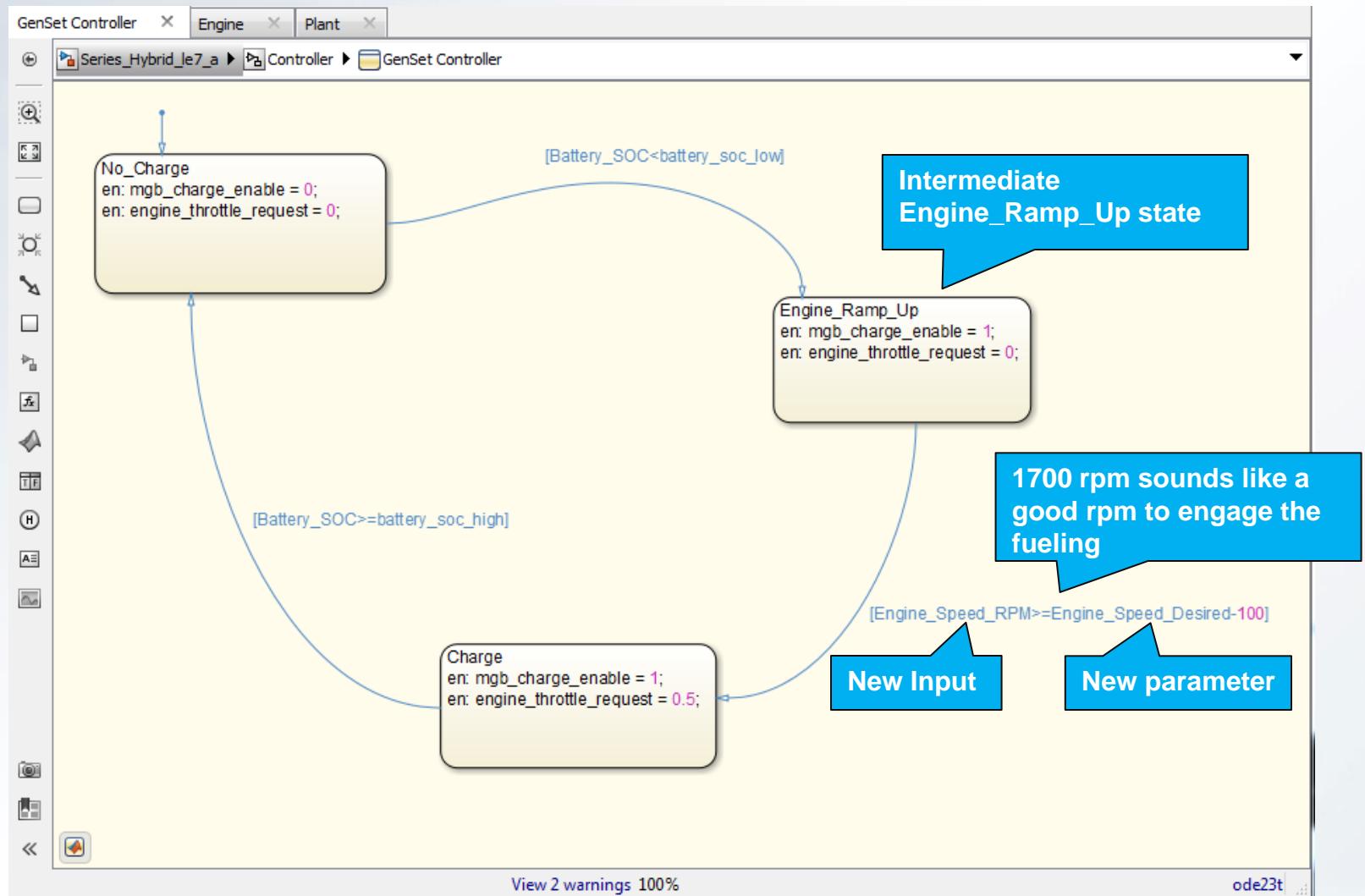
- We don't want the engine to start making power until it is closer to the operating point
- An additional logic state will be needed
 - Set `mgb_charge_enable` to 1 to start engine
 - Set `engine_throttle_request` to 0 until “close” to operating rpm
- Then go into the Charge state



Controller

- Go to the Controller subsystem
 - Open the Stateflow GenSet Controller
 - Add this new state
 - Engine_Speed_RPM is a Simulink input
 - Engine_Speed_Desired is a parameter
 - What is a good value for “close” to 1800?

Controller





Controller

Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy

- Simulink Root
 - Base Workspace
 - Series_Hybrid_le7_a*
 - Model Workspace
 - Configuration (Active)
 - Code for Series_Hybrid_le7_a
 - Simulink Design Verifier results
 - Advice for Series_Hybrid_le7_a
 - Controller
 - GenSet Controller
 - Driver
 - Plant

Contents of: Series_Hybrid_le7_a/Controller/GenSet Controller (only) Filter Contents

Column View: Stateflow Show Details 8 of 15 object(s)

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue
Battery_SOC	Input	1		Inherit: Same as Simulink	-1	
mgb_charge_enable	Output	1	<input type="checkbox"/>	Inherit: Same as Simulink	-1	
battery_soc_high	Parameter			Inherit: Same as Simulink	-1	
battery_soc_low	Parameter			Inherit: Same as Simulink	-1	
engine_throttle_request	Output	2	<input type="checkbox"/>	Inherit: Same as Simulink	-1	
Clock	Input	1				
Engine_Speed_RPM	Input	2		Inherit: Same as Simulink	-1	
Engine_Speed_Desired	Parameter			Inherit: Same as Simulink	-1	

Chart: GenSet Controller

General Fixed-point properties Documentation

Name: GenSet Controller

Machine: (machine) Series_Hybrid_le7_a

Action Language: MATLAB

State Machine Type: Classic

Update method: Inherited Sample Time:

User specified state/transition execution order

Export Chart Level Functions (Make Global)

Execute (enter) Chart At Initialization

Initialize Outputs Every Time Chart Wakes Up

Enable Super Step Semantics

Support variable-size arrays

Saturate on integer overflow

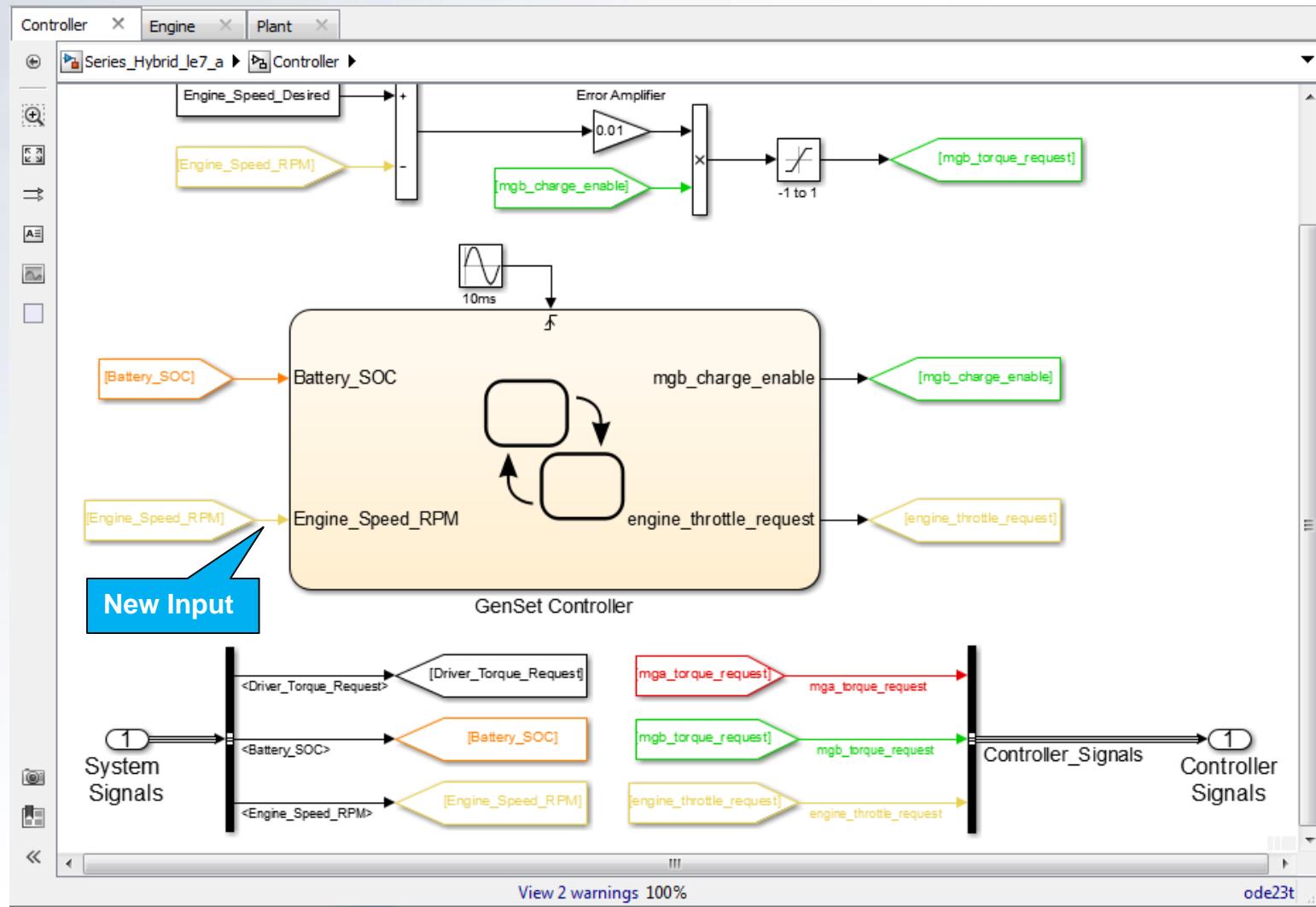
States When Enabling: Held

Create data for monitoring: Child activity

Lock Editor

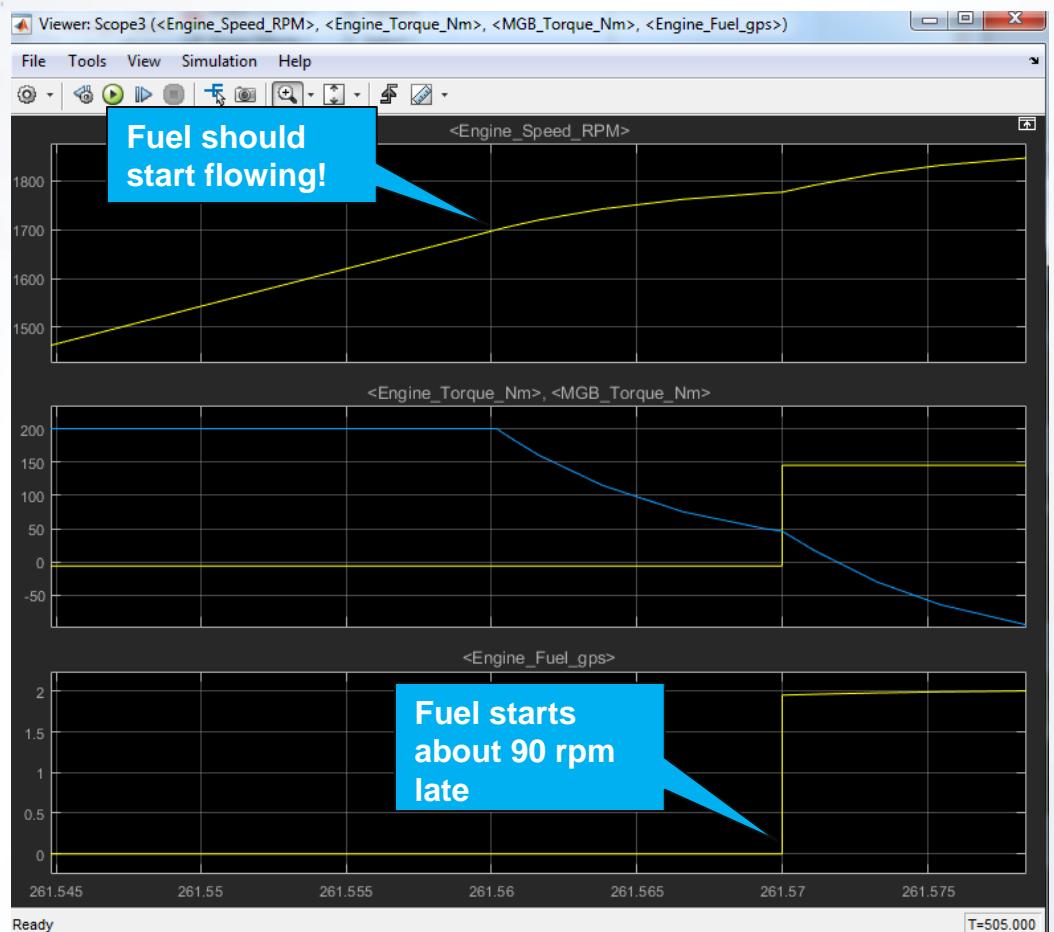
Revert Help Apply

Controller



Results

- X-axis zoom in on the engine ramp up
- Y-axis zoom in on the engine speed
- The fuel rate is about 90 rpm late
 - Why?





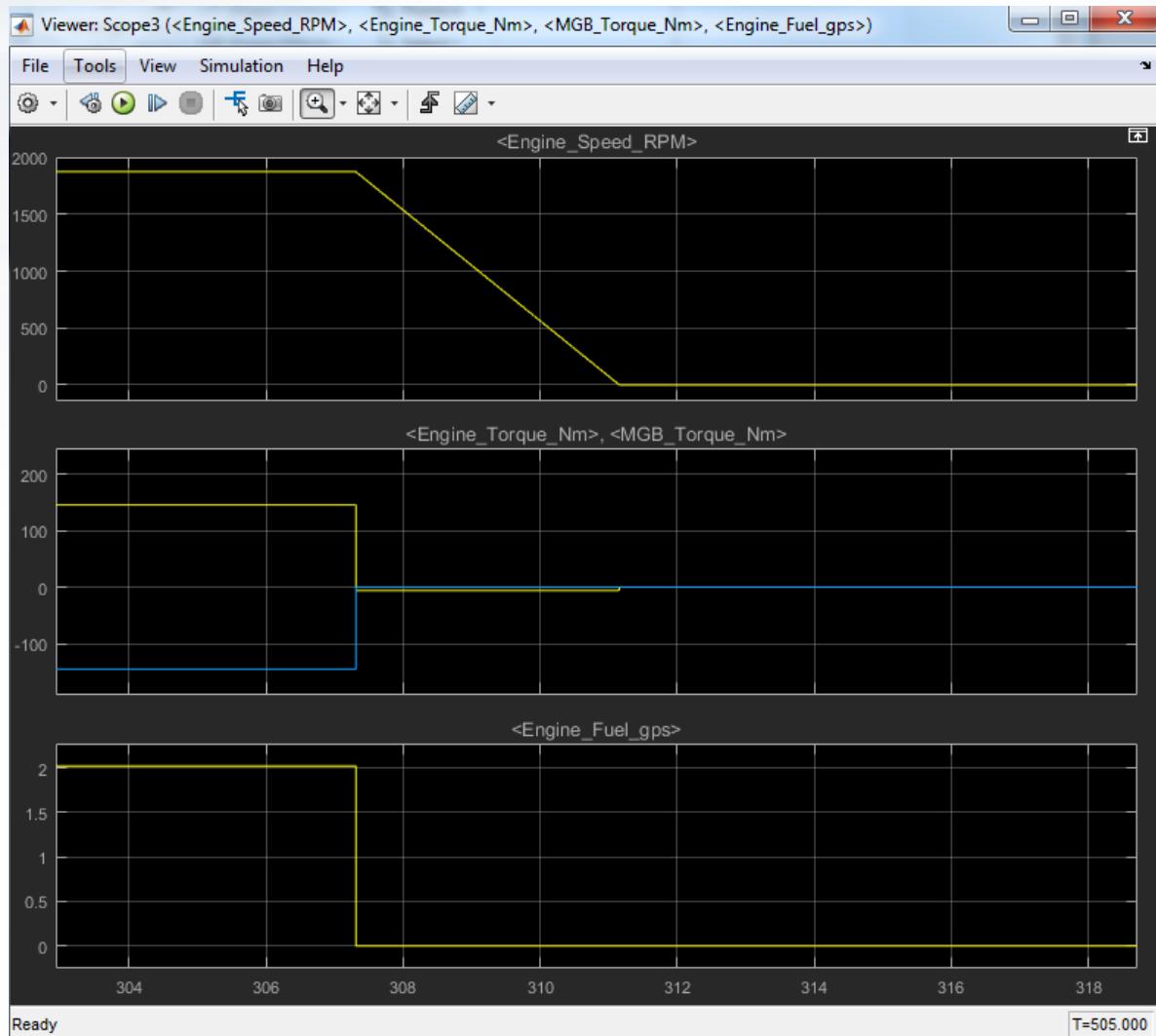
Results

- The ridiculous MGB starts the engine far too quickly
 - The transition guards are checked every 10 ms
 - In 10ms the engine speed goes from 1500 to almost 1800 rpm
 - The transition point was almost missed!
- Engine ramp-up rate will be addressed after we improve the MGB model



Results

- X-axis zoom in on the engine coast down
- Everything here looks fine
 - Fuel shuts off
 - Engine coasts down in about 4 seconds





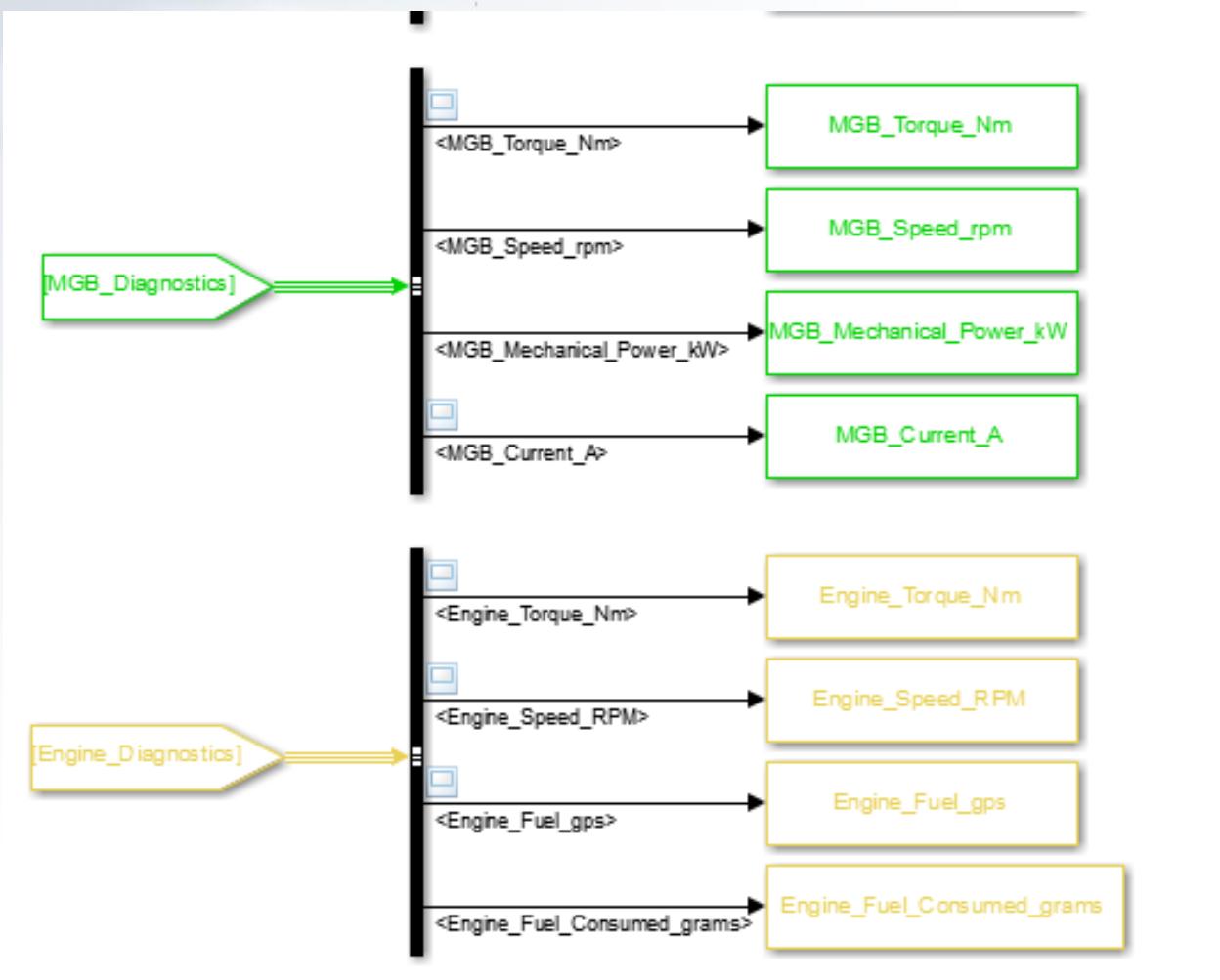
- Let's now calculate the vehicle fuel efficiency
 - MPG
 - SOC correction
 - GE (gas equivalent)
- We will do this with a post-processing file
 - Simply for experience
 - Could be done in the Logging & Visualization subsystem



- Calculating MPG is easy!
 - Distance traveled / gallons consumed
- We already have a signal for distance traveled in miles
- We need a fuel consumed signal
 - Add an **Integrator** into the Engine subsystem to calculate the amount of fuel consumed in grams
 - Add the signal to the Engine Diagnostics bus
 - Extract it in the Logging & Visualization
- All of the remaining calculations are done in post file



Engine & Logging





- If we removed energy from the battery during the drive cycle, we must calculate the additional amount of fuel required to replace it
- If we added energy to the battery, we must calculate the amount of fuel required to produce that energy and subtract the fuel from our total



- Step 1 – Amp Hours Consumed
 - $\text{Ah}_{\text{Consumed}} = (\text{Initial_SOC} - \text{Final_SOC}) * \text{Battery_Capacity}$
 - Battery capacity is in Ah (8.5 Ah for our model)
- Step 2 – Convert to Energy
 - $\text{Electrical_Energy} = \text{Ah}_{\text{Consumed}} * \text{Battery_OCV}$
 - OCV (open circuit voltage) is determined experimentally from V-I plots (366 V in our model) and changes with the SOC. Energy is in Watt-hr.



- Step 3 – Conversion Efficiency
 - $\text{Electrical_Energy_BTU} = (\text{3.412} * \text{Electrical_Energy}) / \text{Efficiency}$
 - Conversion efficiency is also determined experimentally by ANL (25% approximation generous)
 - The 3.412 converts Watt-hr to BTU
- Excellent – we now know the energy in BTU needed to account for the charge added to or removed from the battery



- As a standard for comparison, we employ the MPGGE with SOC correction
- This gives the equivalent amount of RFG required to produce the same energy used in the drive cycle
- We already know the BTUs of energy needed to correct the SOC



- Step 4 – Fuel Energy BTU
 - $\text{Fuel_Energy_BTU} = \text{Fuel_Consumed_Gallons}^* 133393.1102$
- Step 5 – Total Energy BTU
 - $\text{Total_Energy_BTU} = \text{Electrical_Energy_BTU} + \text{Fuel_Energy_BTU}$
- 133393.1102 BTU/Gal is the fuel heating value of B-20
- Excellent – we now know the total energy consumed by our vehicle over the drive cycle



- Step 6 – RFG Required
 - $\text{Fuel}_{\text{Consumed}}_{\text{RFG}} = \text{Total}_{\text{Energy}}_{\text{BTU}} / 114871.7446$
 - This gives us the gallons of RFG required to produce the same energy
 - 114871.7446 BTU/gal is the fuel heating value of RFG
- Step 7 – MPGGE with SOC Correction
 - $\text{MPGGE} = \text{Distance}_{\text{Travelled}} / \text{Fuel}_{\text{Consumed}}_{\text{RFG}}$



Vehicle_Post_File.m File

C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Post_File.m

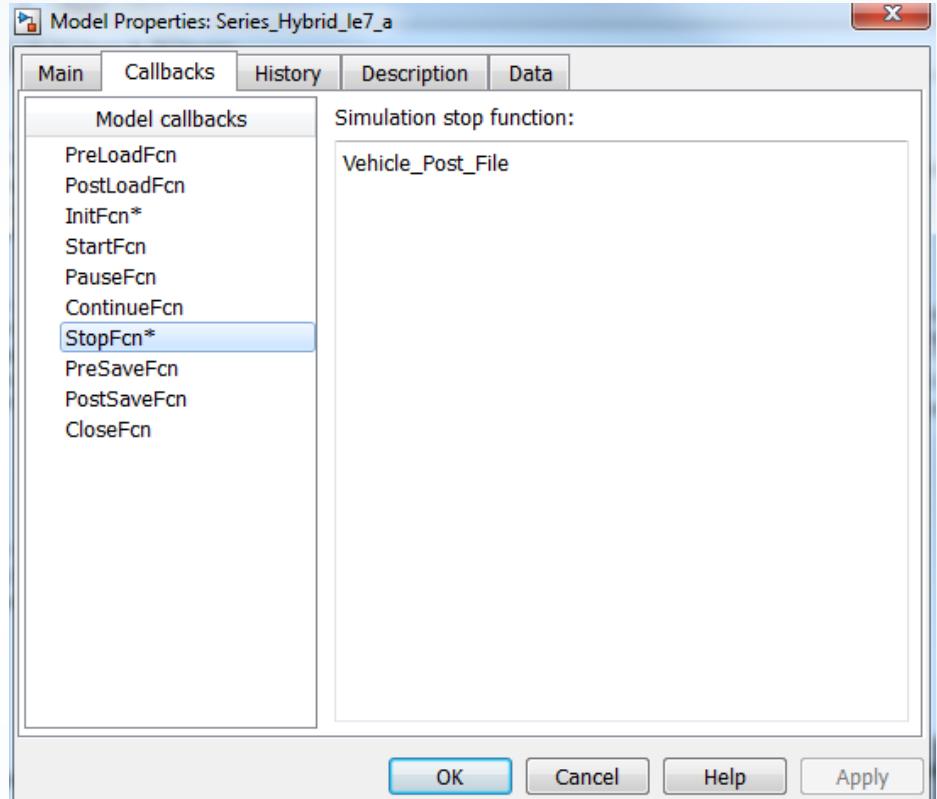
FILE	NAVIGATE	EDIT	BREAKPOINTS	RUN
1 % Post Processing				
2				
3 % Constants				
4 Fuel_Heating_Value_B20 = 133393.1102; %BTU/gal				
5 Fuel_Heating_Value_RFG = 114871.7446; %BTU/gal				
6				
7 % Calculations				
8 Fuel_Consumed = Engine_Fuel_Consumed_grams(end); %grams				
9 Distance_Traveled = Vehicle_Distance_miles(end); %miles				
10				
11 Battery_OCV = 366; %Volts				
12 delta_SOC = Battery_SOC(end)-Battery_Initial_SOC; %non-d				
13 Amp_Hours_Consumed = delta_SOC*Battery_Capacity; %amp hours				
14 Electrical_Energy = Amp_Hours_Consumed*Battery_OCV; %watt hours				
15 Conversion_Efficiency = 0.25; %ANL data				
16 Electrical_Energy_BTU = (Electrical_Energy*3.412)/Conversion_Efficiency;				
17				
18 Fuel_Consumed_B20_Gallons = Fuel_Consumed/3215; %gallons				
19 Fuel_Energy_BTU = Fuel_Consumed_B20_Gallons*Fuel_Heating_Value_B20;				
20				
21 Total_Energy_Consumed = Fuel_Energy_BTU-Electrical_Energy_BTU;				
22 Fuel_Consumed_RFG_Gal = Total_Energy_Consumed/Fuel_Heating_Value_RFG;				
23 MPGGE_RFG_wSOC = Distance_Traveled/Fuel_Consumed_RFG_Gal;				
24 MPG_Diesel = Distance_Traveled/Fuel_Consumed_B20_Gallons;				
25				
26 %Dialogbox				
27 msg1=sprintf(' Simulation Results \n\n');				
28 msg2=sprintf('Total Distance Traveled (miles) %g\n',Distance_Traveled);				
29 msg3=sprintf('Total Fuel Consumed (gal) %g\n',Fuel_Consumed_B20_Gallons);				
30 msg4=sprintf('Fuel Energy Consumed (BTU) %g\n',Fuel_Energy_BTU);				
31 msg5=sprintf('Electrical Energy Consumed (BTU) %g\n',Electrical_Energy_BTU);				
32 msg6=sprintf('MPGGE with SOC Correction %g\n',MPGGE_RFG_wSOC);				
33 msg7=sprintf('MPG Diesel Only %g\n',MPG_Diesel);				
34 msg=[msg1,msg2,msg3,msg4,msg5,msg6,msg7];				
35 h=msgbox(msg);				

script Ln 2 Col 1



Post Processing File

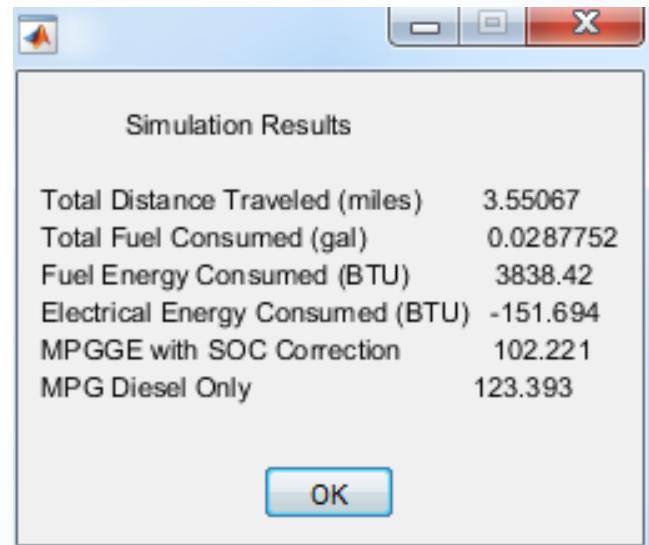
- Like the init file, the post file should run after every simulation
- Right click on the model and
 - Select Model Properties
 - Select Callbacks
 - Select StopFcn
 - Enter the name of the post processing file without the .m





Results

- Rerun the simulation
- Not bad
 - 102 MPGGE w/SOC
 - 123 MPG straight
- Unrealistic?
 - 100% regen braking
 - 100% efficient battery
 - 100% efficient MGs
- We'll watch these numbers go down!





Review

- Improve Engine model
 - Experimental data for torque curves
 - Experimental data for fuel consumption
 - Controller refinements
- Fuel Efficiency
 - Calculate MPG, MPGGE, and MPGGE with SOC correction

A blue-toned photograph of the Georgia Tech Campanile, a tall brick tower with a spire. The word "TECH" is prominently displayed in white letters on the side of the tower.

MBSD Lecture 8

Improved Battery Model



Outline

- Improve Battery model
 - First order math model
 - Variable battery voltage
 - Algebraic loops
 - Variable open circuit voltage
 - Variable charge/discharge resistance
 - Numeric error



Simple Battery

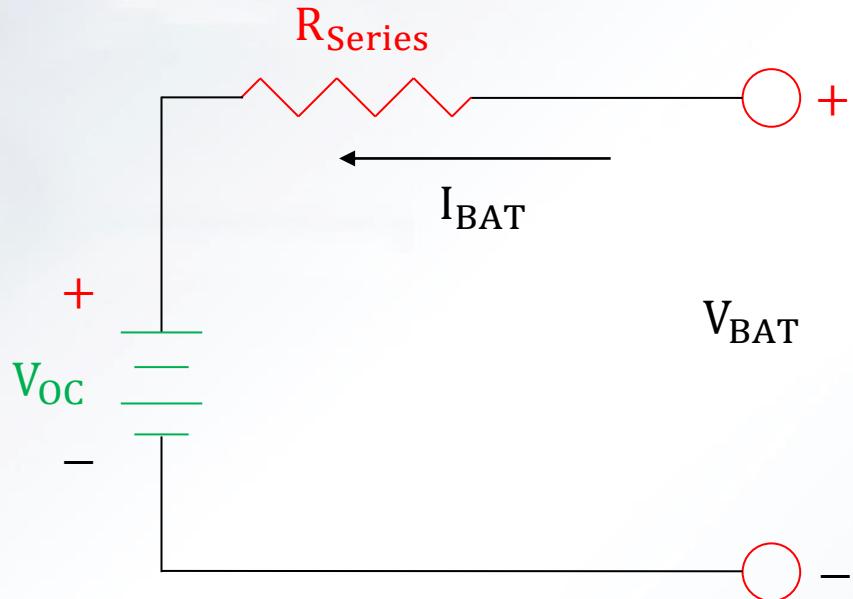
- Thus far, the battery model
 - Has had a constant voltage of 336V
 - Has had a constant OCV of 366V
 - Been able to provide infinite current
 - Has had no energy conversion losses
- Let's now improve the battery model by including the internal resistance

$$V_{BAT} = V_{OC} + I_{BAT} \times R_{Series}$$

- The battery voltage is now a function of the current



Battery Terminal Voltage



- Current is defined as positive into the battery
- Positive current charges the battery and increases the battery SOC



Future Improvements

- For now, the open circuit voltage V_{OC} and the battery series resistance R_{Series} are a constant
- As our understanding of the model increases, we can make the battery model less ideal by:
 - Making V_{OC} a function of SOC and temperature
 - Having a different charge and discharge series resistances, both of which are a function of SOC and temperature
- These will all involve experimental (or manufacturer) data and 2-d lookup tables

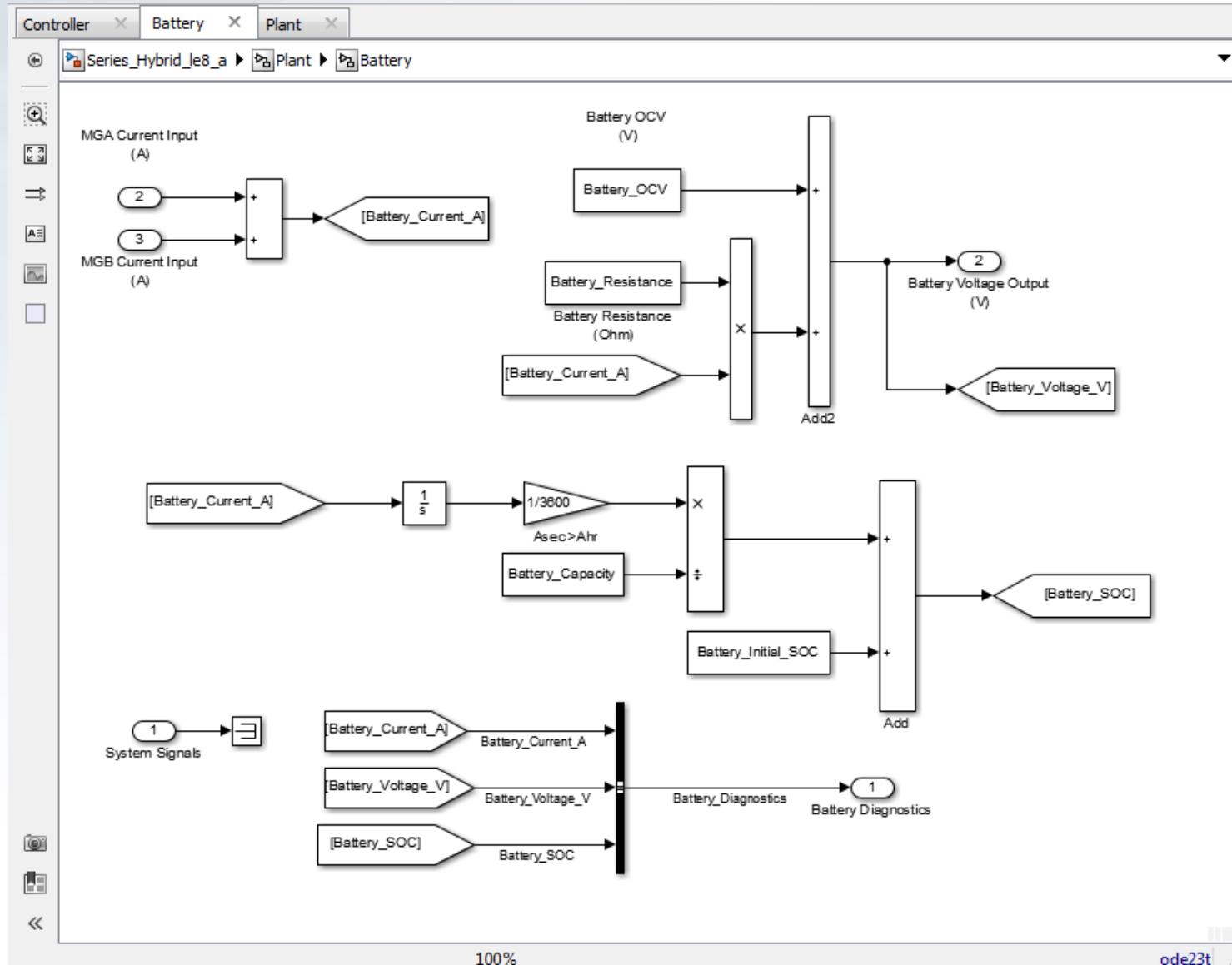


First Order Model

- The battery OCV and series resistance will be constants read from the init file
 - `Battery_OCV = 366`
 - `Battery_Resistance = 0.4`
- Add to the init file
- Remove the `Battery_Voltage`
- Remove the OCV from the post file
- Rename your model `Series_Hybrid_le8_a.slx` and implement the battery voltage equation



First Order Model





Results

- Run the Simulation – Dooh!

A screenshot of the MATLAB Diagnostic Viewer window titled "Diagnostic Viewer". The window displays error messages for a model named "Series_Hybrid_le8_a".

```
Diagnostic Viewer

Series_Hybrid_le8_a
Component: Simulink | Category: Block
'Series_Hybrid_le8_a/Plant/Battery/Add2' (algebraic variable)

Component: Simulink | Category: Block
Discontinuities detected within algebraic loop(s), may have trouble solving

Division by zero in 'Series_Hybrid_le8_a/Plant/MGB/Divide'

Component: Simulink | Category: Block warning
An error occurred while running the simulation and the simulation was terminated

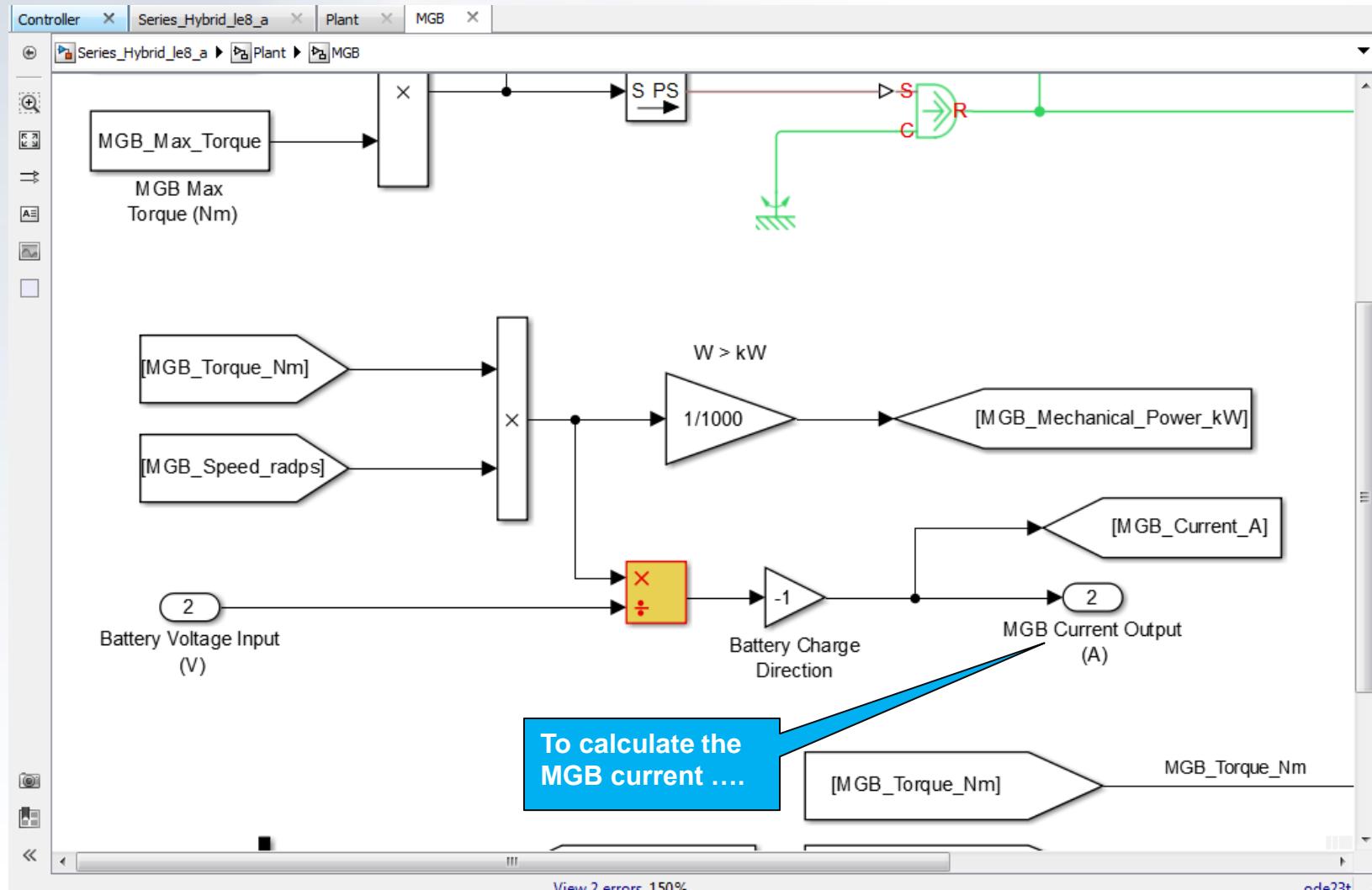
Caused by:
• Algebraic state in algebraic loop containing 'Series_Hybrid_le8_a/Plant/MGB/Divide'
computed at time 0.0 is Inf or NaN. There may be a singularity in the solution. If
the model is correct, try reducing the step size (either by reducing the fixed step
size or by increasing the error tolerances)

Component: Simulink | Category: Block error
```

Algebraic Loop: This means a variable is a function of itself!

Click here to go to the problem

Algebraic Loop





Algebraic Loop

- Aha! The loop involves the battery voltage which is a function of the battery current which is a function of the battery voltage ...
- To break the algebraic loop a delay will be used to have either
 - The battery voltage signal lag the battery current calculation or
 - The battery current signal lag the battery voltage calculation
- Close the Simulations Diagnostics box

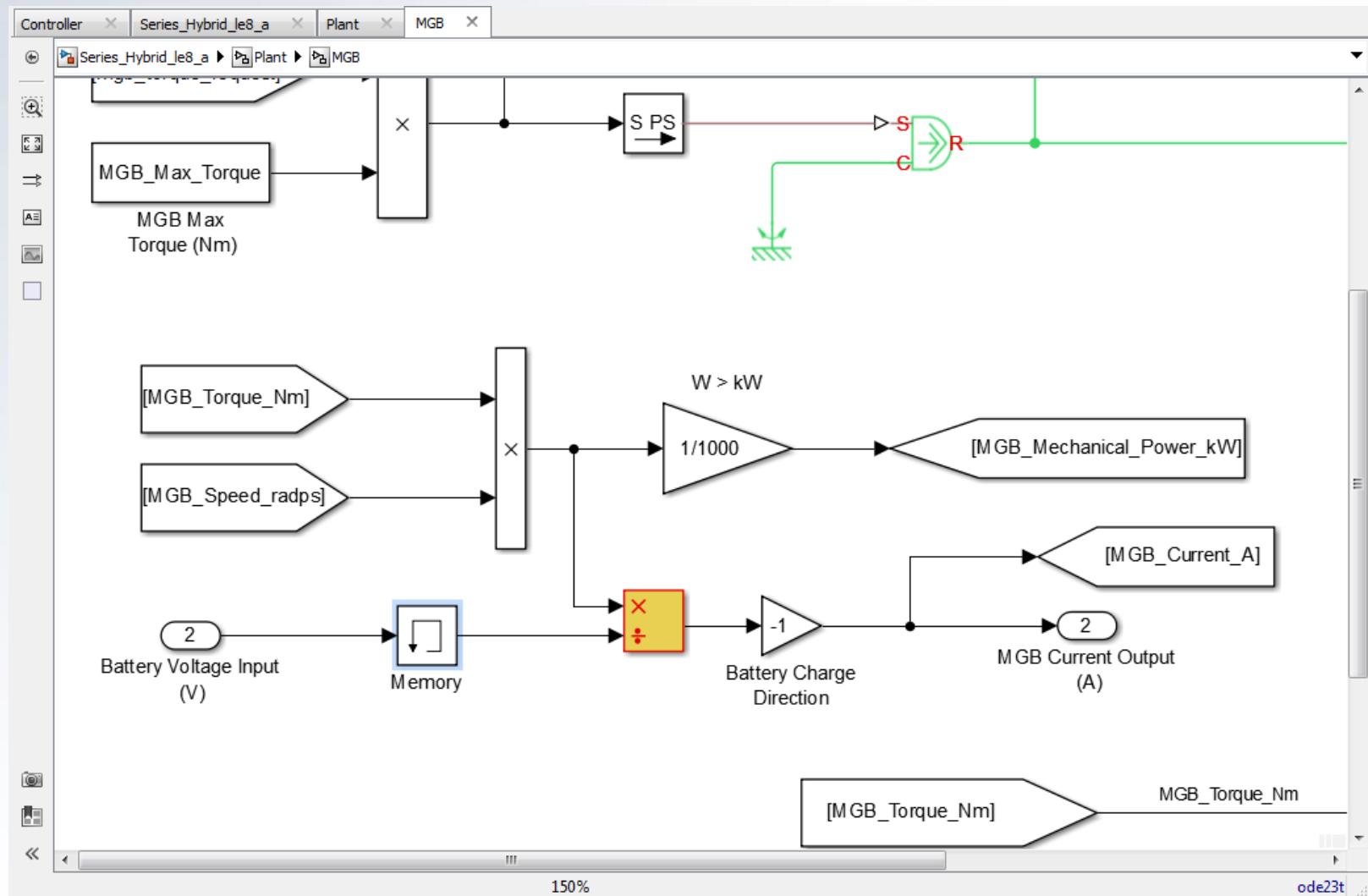


Algebraic Loop

- Go to the MGB subsystem where the algebraic loop was first identified
- From
 - Simulink / Discrete
- Drag in a **Memory** block
- Insert it just after the Voltage Input block
- Thinking ahead, MGA is also used to calculate the battery current
- Put a **Memory** block there too



MGA/MGB Voltage Delay





Results

- Run the simulation and...

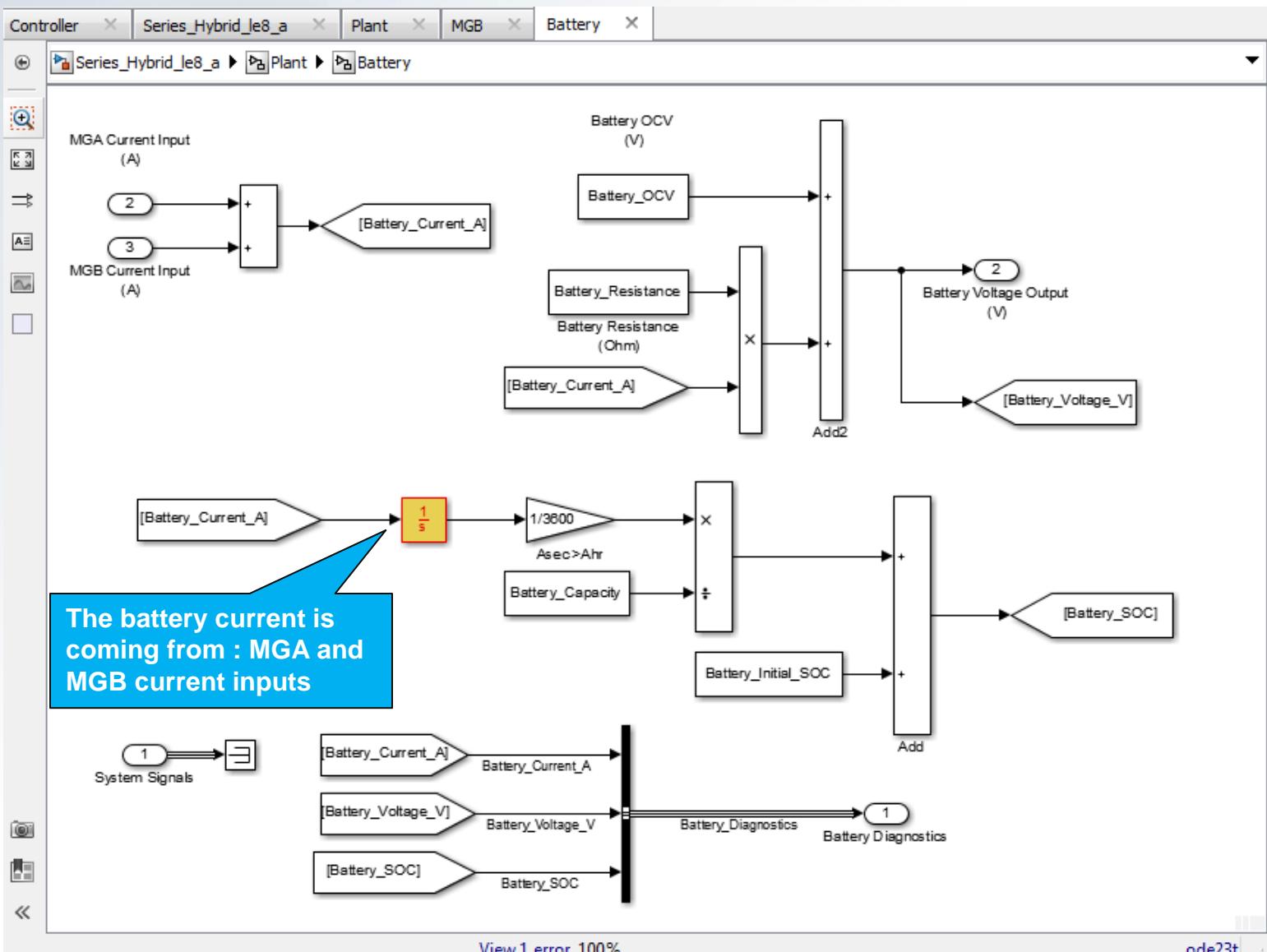
The screenshot shows the Diagnostic Viewer window with the title "Diagnostic Viewer". The window displays a list of errors and warnings from a simulation run named "Series_Hybrid_le8_a".

- Component: Simulink | Category: Block
['Series_Hybrid_le8_a/Plant/MGA/Product'](#) (algebraic variable)
- Component: Simulink | Category: Block
Discontinuities detected within algebraic loop(s), may have trouble solving
- Division by zero in ['Series_Hybrid_le8_a/Plant/MGA/Divide'](#) [2 similar]
- Component: Simulink | Category: Block warning
An error occurred while running the simulation and the simulation was terminated
- Caused by:
 - Derivative of state '1' in block ['Series_Hybrid_le8_a/Plant/Battery/Integrator'](#) at time 0.0 is not finite. The simulation will be stopped. There may be a singularity in the solution. If not, try reducing the step size (either by reducing the fixed step size or by increasing the error tolerances)

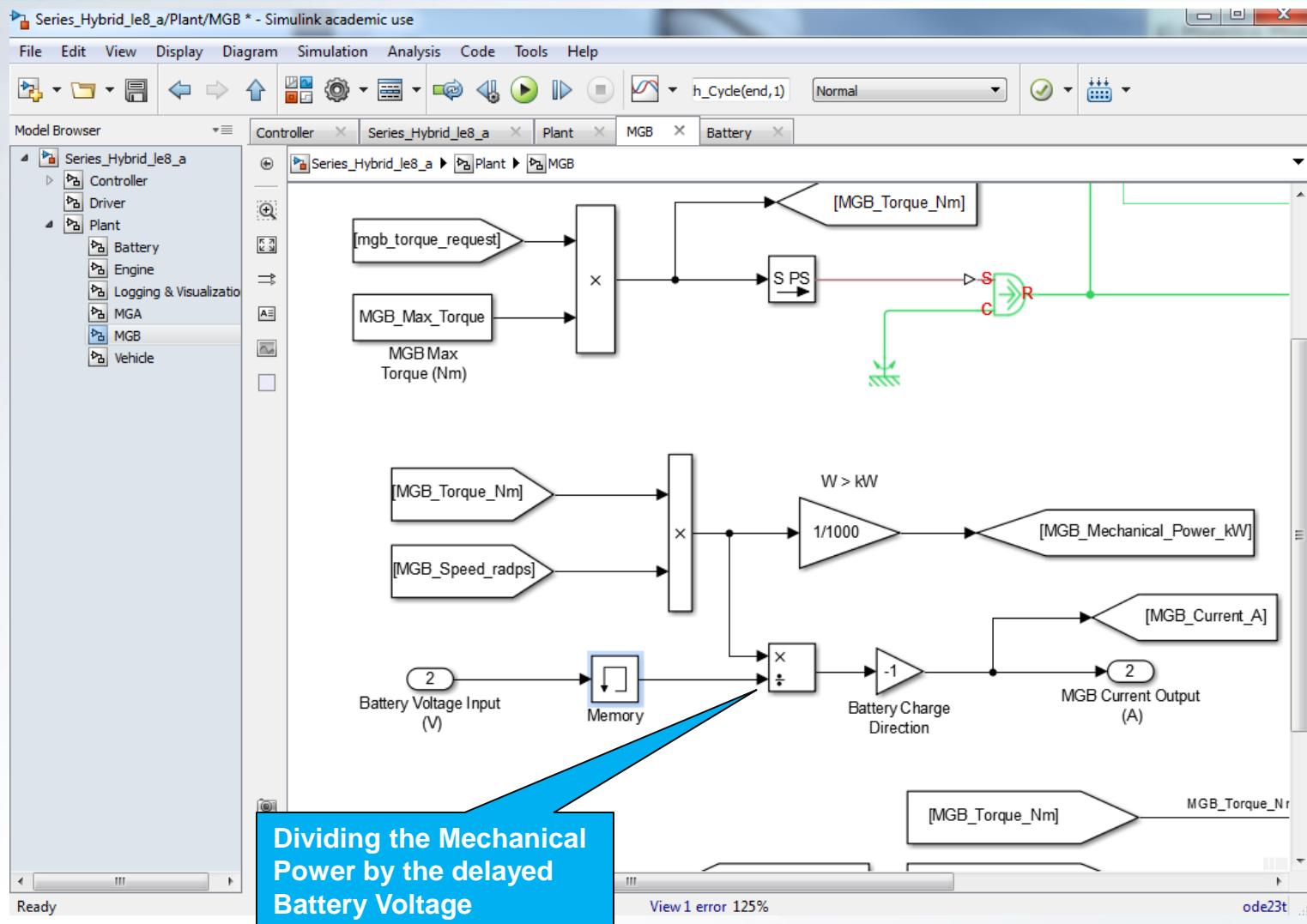
A value that is not finite means we are dividing by zero somewhere in the model ... (Note that it occurs at time 0)

Click here to go to the Battery subsystem

Not finite (infinity)



Not finite (infinity)





Not finite (infinity)

- At the start of the simulation (time zero), there is no delayed value of voltage
- What does the memory block do?
 - Double click on it
 - Aha! It uses zero as its first value...
 - which results in a divide by zero error at time zero
- The first strategy of delaying the Battery voltage signal was a bad one
- Undo the MGA/MGB improvements

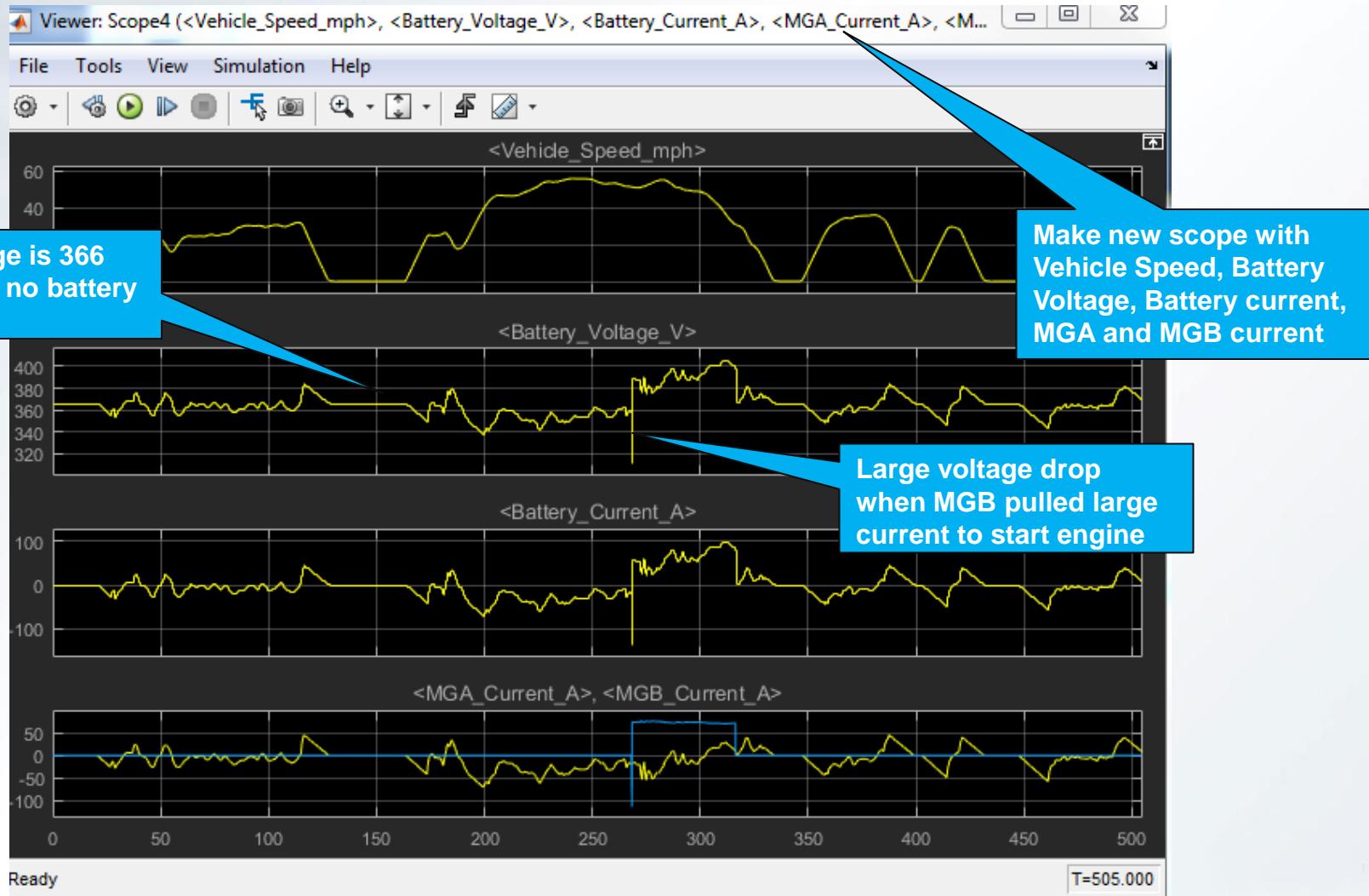


Algebraic Loop

- Let's try strategy two, delaying the Battery current in calculating the Battery Voltage
- Close the Simulation Diagnostics box
- Close the MPGGE dialogue boxes
- Go to the Battery subsystem
 - Insert the Memory block into the Battery voltage calculation
 - Verify that there are no downstream divides that could result in division by zero
- Run the simulation

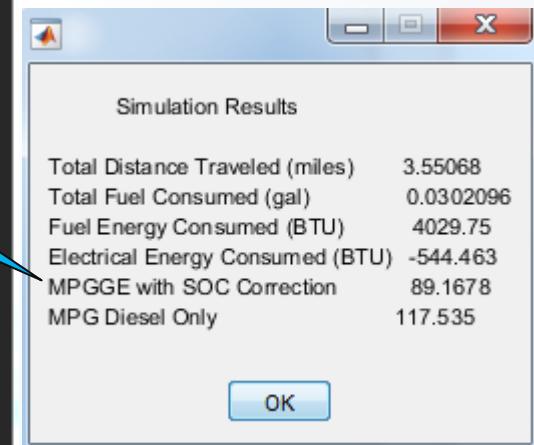
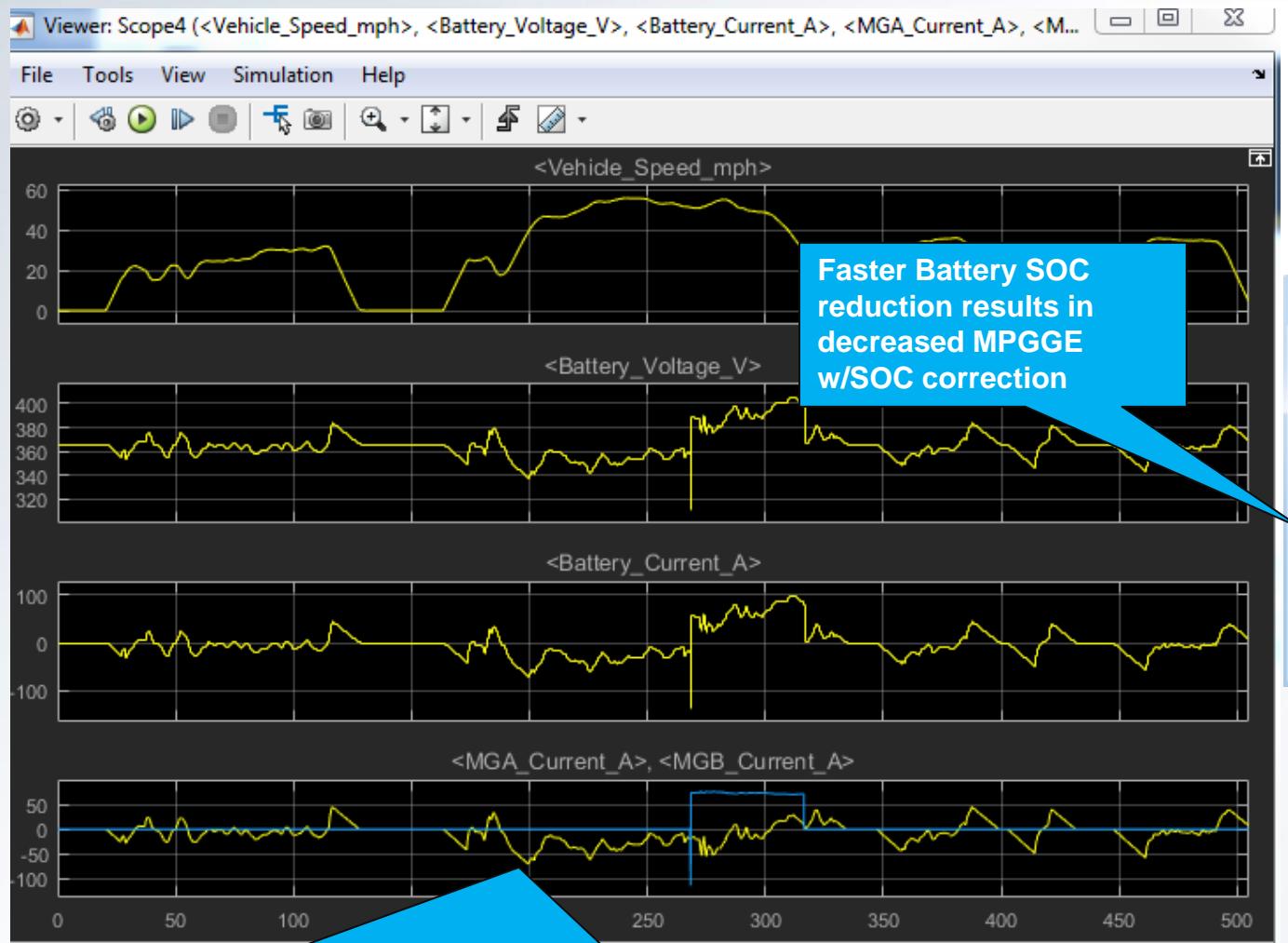


Results





Results



Negative MGA current results in Negative Battery current

Negative Battery current results in Decreased Battery Voltage

Decreased Battery voltage results in Increased MGA current for torque request

Increased MGA current results in Faster reduction in Battery SOC



Results

- This simple improvement has substantially decreased our SOC corrected MPGGE
- Let's improve the model further and see how the fuel economy is affected
- First, we will add a variable OCV as a function of SOC and temperature
- At the Matlab command line type
 - `clear variables`
 - `load Component_Data/Battery_Data.mat;`
 - Click on the Workspace tab



Battery Data

MATLAB R2016a - academic use

HOME PLOTS APPS

New Script New Open Find Files Import Data New Variable Analyze Code Simulink Preferences

Open Compare Save Workspace Open Variable Run and Time Set Path Add-Ons

Clear Workspace Clear Commands Parallel RESOURCES

FILE VARIABLE CODE SIMULINK ENVIRONMENT

Current Folder C: > Users > sologhavi3 > Desktop > ME4013_Tutorial > Vehicle_model >

Command Window

```
>> clear variables
>> load Component_Data/Battery_Data.mat;
fx >> |
```

Workspace

Name	Type	Value
Battery_OCV_Data	9x5 double	[0.1000;0.2000;0.3000;0.4000;0.5000]
Battery_OCV_SOC_Axis		[0,10,20,35,55]
Battery_OCV_Temp_Axis		
Battery_R_Charge_Data	9x5 double	
Battery_R_Discharge_Data	9x5 double	
Battery_R_SOC_Axis		[0.1000;0.2000;0.3000;0.4000;0.5000]
Battery_R_Temp_Axis		[-10,10,20,35,55]

Row vector

Data matrix

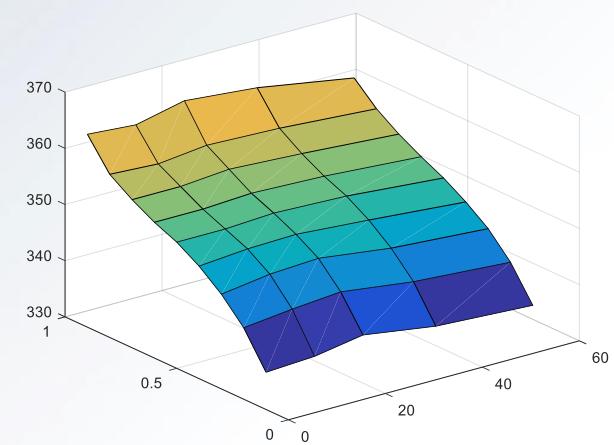
Column vector



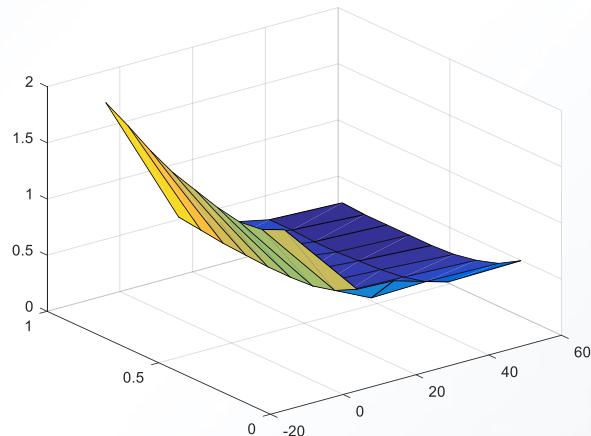
Battery Data

- At the command line, type
 - `surf(Battery_OCV_Temp_Axis, Battery_OCV_SOC_Axis, Battery_OCV_Data)`
- Repeat for the resistance data

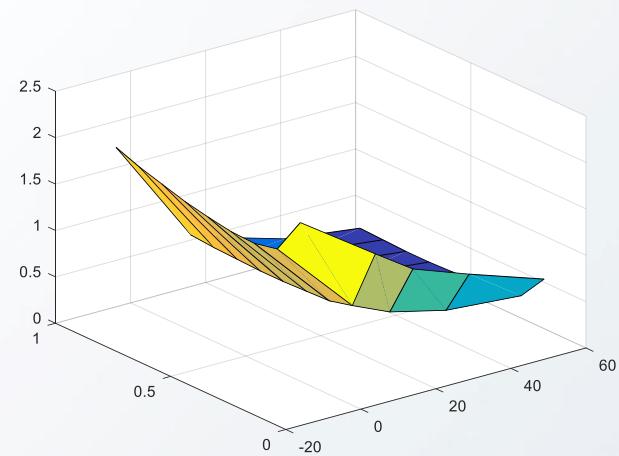
OCV



Charge



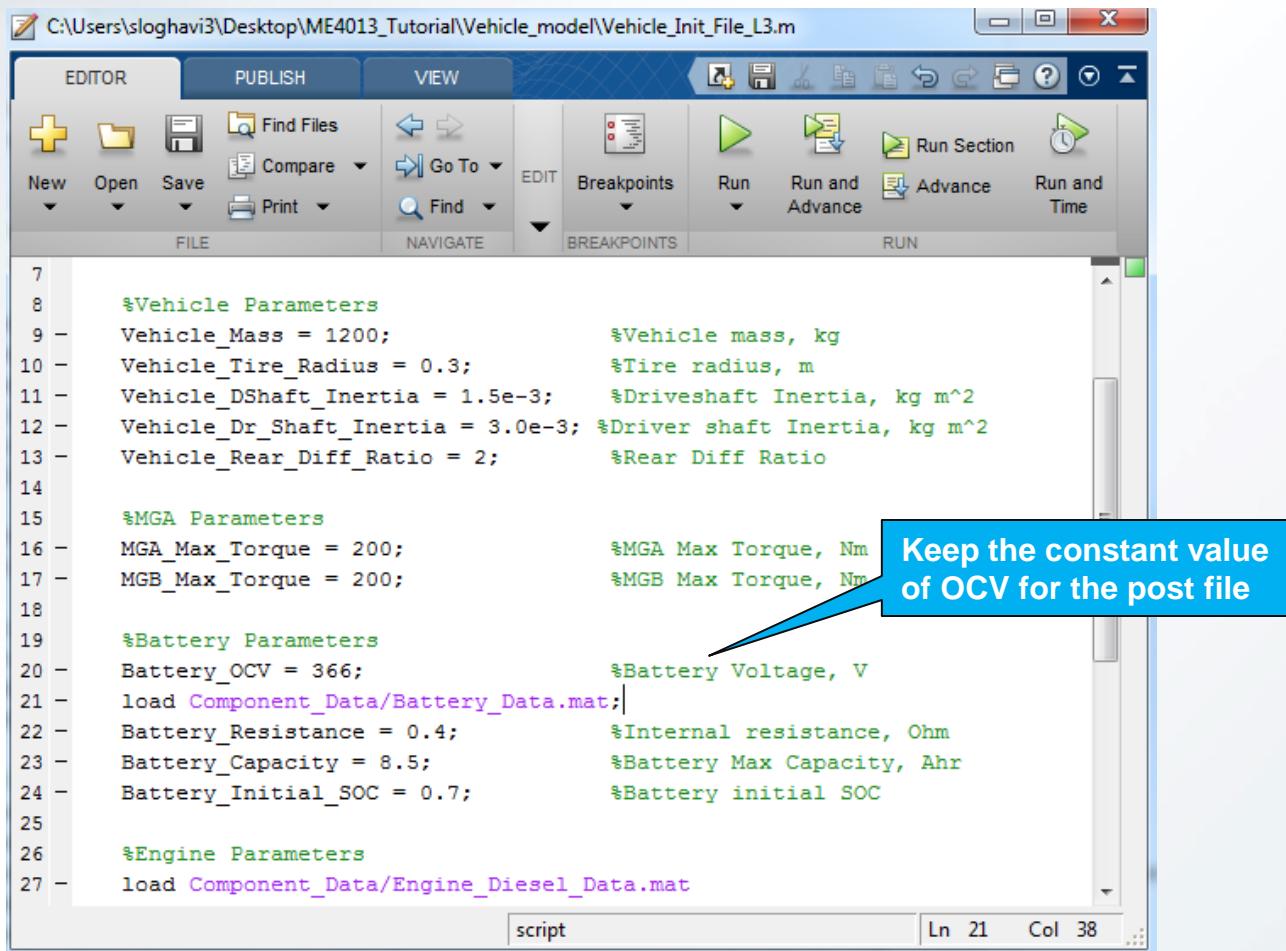
Discharge





Battery Data

- Add the loading of the battery data to the init file



The screenshot shows the MATLAB Editor window with the title bar "C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m". The code in the editor is as follows:

```
7 %Vehicle Parameters
8 Vehicle_Mass = 1200; %Vehicle mass, kg
9 Vehicle_Tire_Radius = 0.3; %Tire radius, m
10 Vehicle_DShaft_Inertia = 1.5e-3; %Driveshaft Inertia, kg m^2
11 Vehicle_Dr_Shaft_Inertia = 3.0e-3; %Driver shaft Inertia, kg m^2
12 Vehicle_Rear_Diff_Ratio = 2; %Rear Diff Ratio
13
14 %MGA Parameters
15 MGA_Max_Torque = 200; %MGA Max Torque, Nm
16 MGB_Max_Torque = 200; %MGB Max Torque, Nm
17
18 %Battery Parameters
19 Battery_OCV = 366; %Battery Voltage, V
20 load Component_Data/Battery_Data.mat;
21 Battery_Resistance = 0.4; %Internal resistance, Ohm
22 Battery_Capacity = 8.5; %Battery Max Capacity, Ahr
23 Battery_Initial_SOC = 0.7; %Battery initial SOC
24
25 %Engine Parameters
26 load Component_Data/Engine_Diesel_Data.mat
```

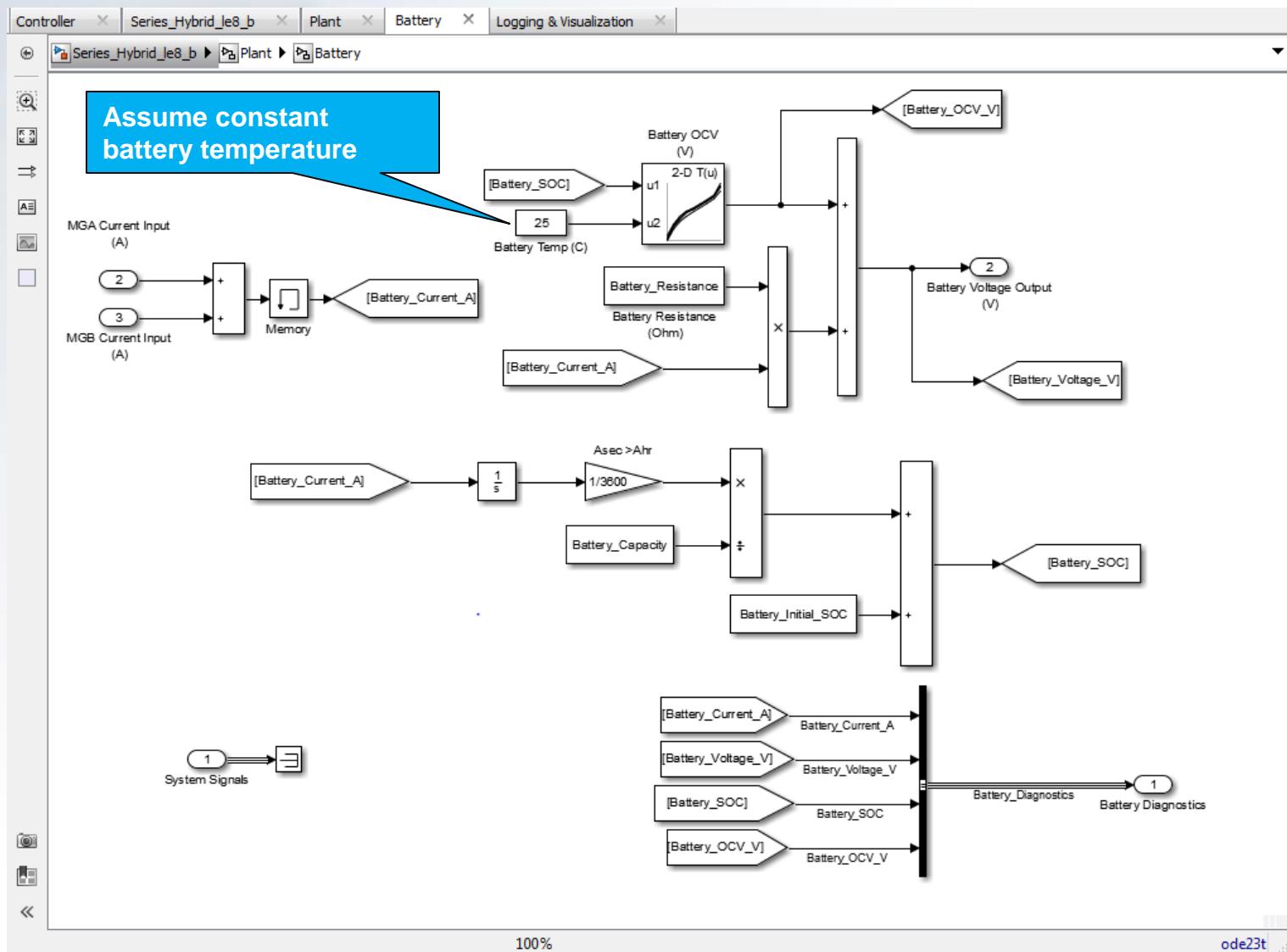
A blue callout box with the text "Keep the constant value of OCV for the post file" points to the line "Battery_OCV = 366;".



Variable OCV

- Save your model as
 - Series_Hybrid_le8_b.slx
- Go to the Battery subsystem
- Delete the Battery OCV Constant block
- Drag in a
 - 2-D Look up Table block
 - Constant block
- Add the OCV signal to the Battery Diagnostics Bus and extract it in the Logger

Variable OCV





Variable OCV

Block Parameters: Battery OCV (V)

Lookup Table (n-D)
Perform n-dimensional interpolated table lookup including index searches. The table is a sampled representation of a function in N variables. Breakpoint sets relate the input values to positions in the table. The first dimension corresponds to the top (or left) input port.

Table and Breakpoints Algorithm Data Types

Number of table dimensions: 2

Table data: Battery_OCV_Data

Breakpoints specification: Explicit values

Breakpoints 1: Battery_OCV_SOC_Axis

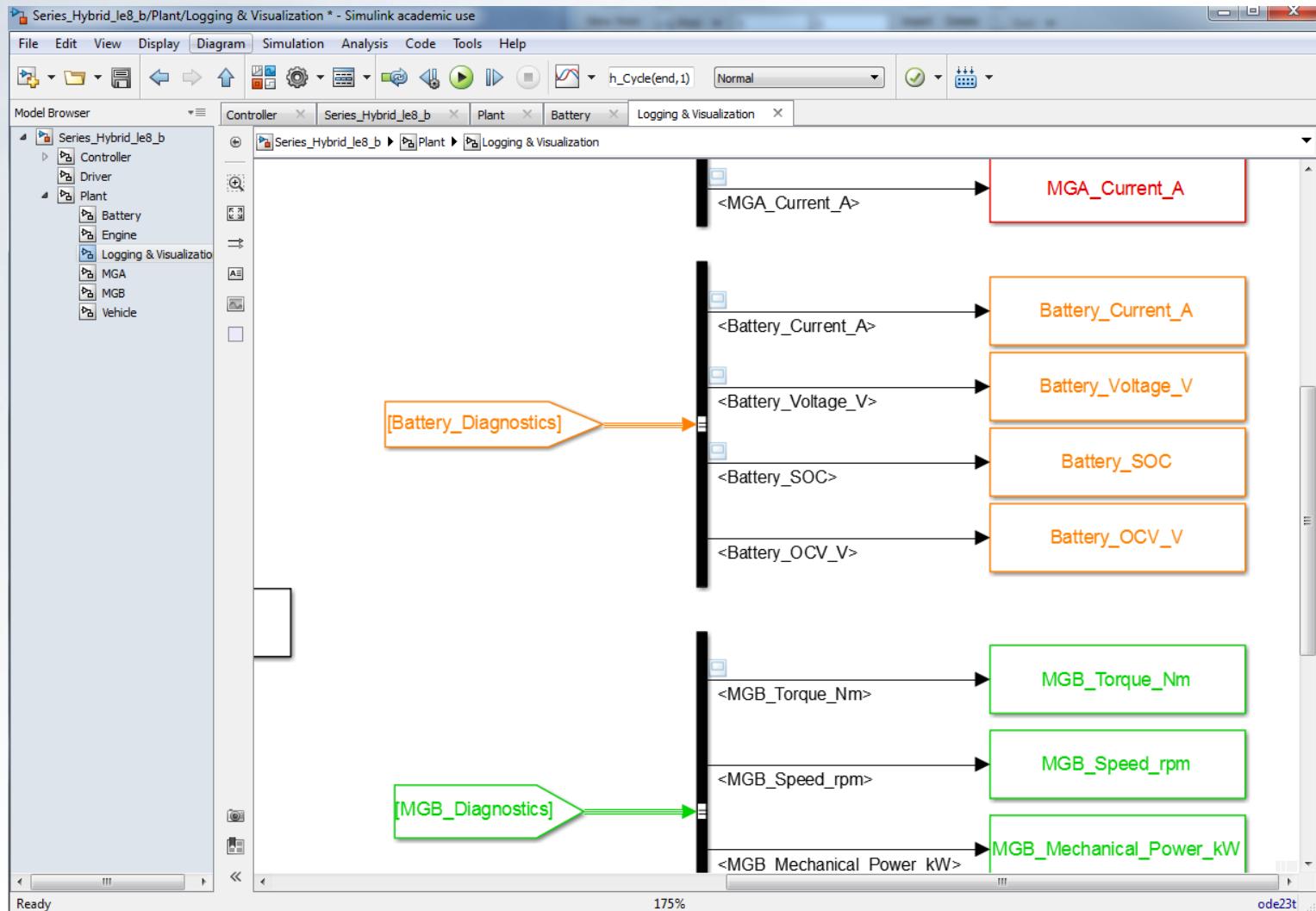
Breakpoints 2: Battery_OCV_Temp_Axis

Edit table and breakpoints...

OK Cancel Help Apply



Logging & Visualization



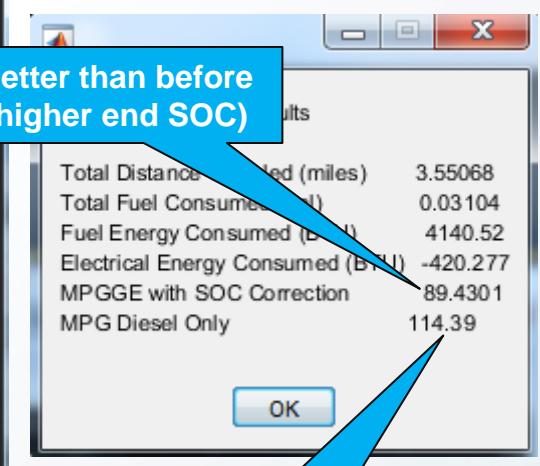
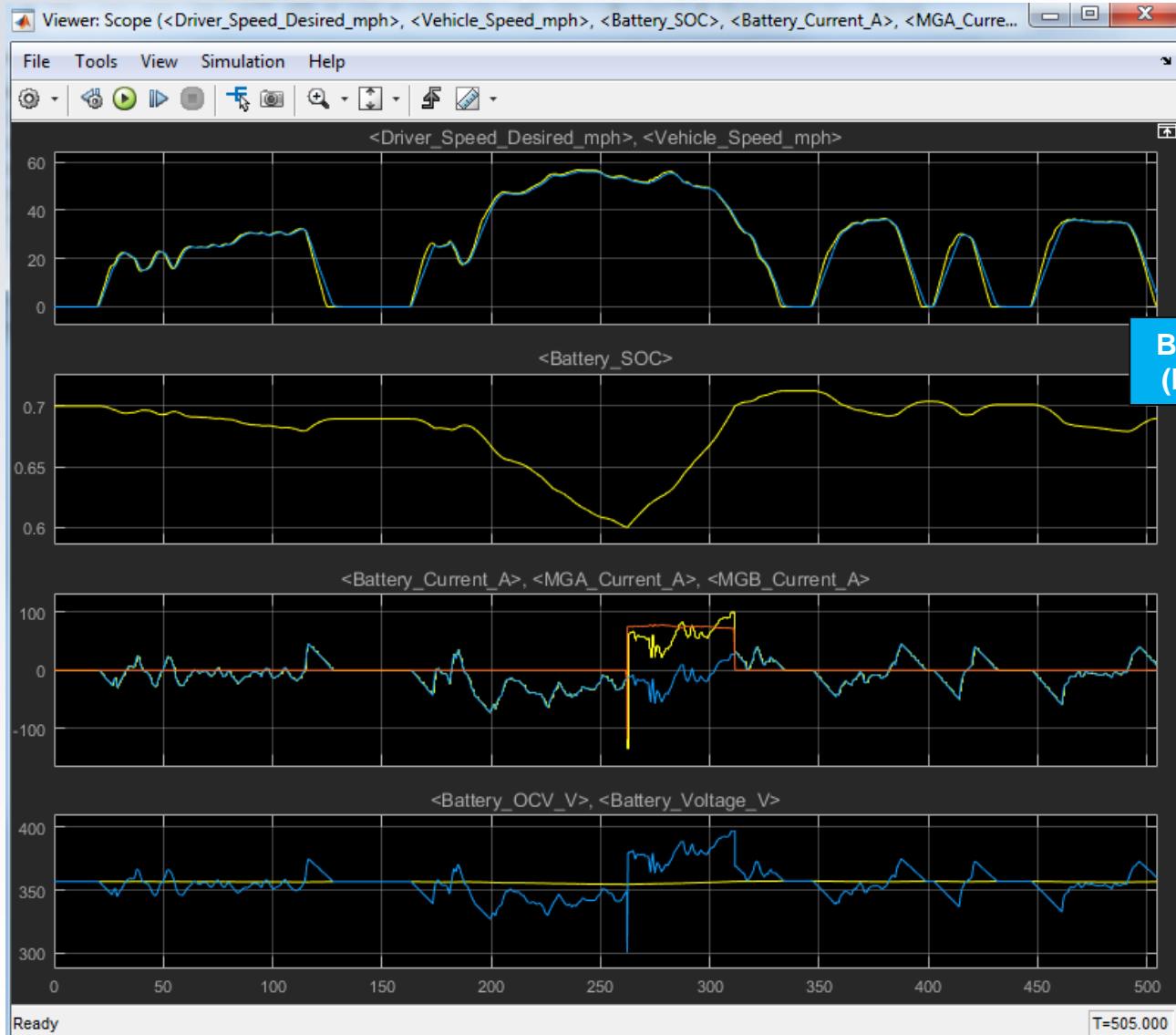


Variable OCV

- We've got a lot of scopes – delete them all
- Create a new one
 - Axis 1 – Vehicle Speed (Desired & Actual)
 - Axis 2 – Battery SOC
 - Axis 3 – Battery, MGA, & MGB current
 - Axis 4 – Battery OCV and Voltage
- Run the simulation



Variable OCV



Better than before
(higher end SOC)



Variable Internal Resistance

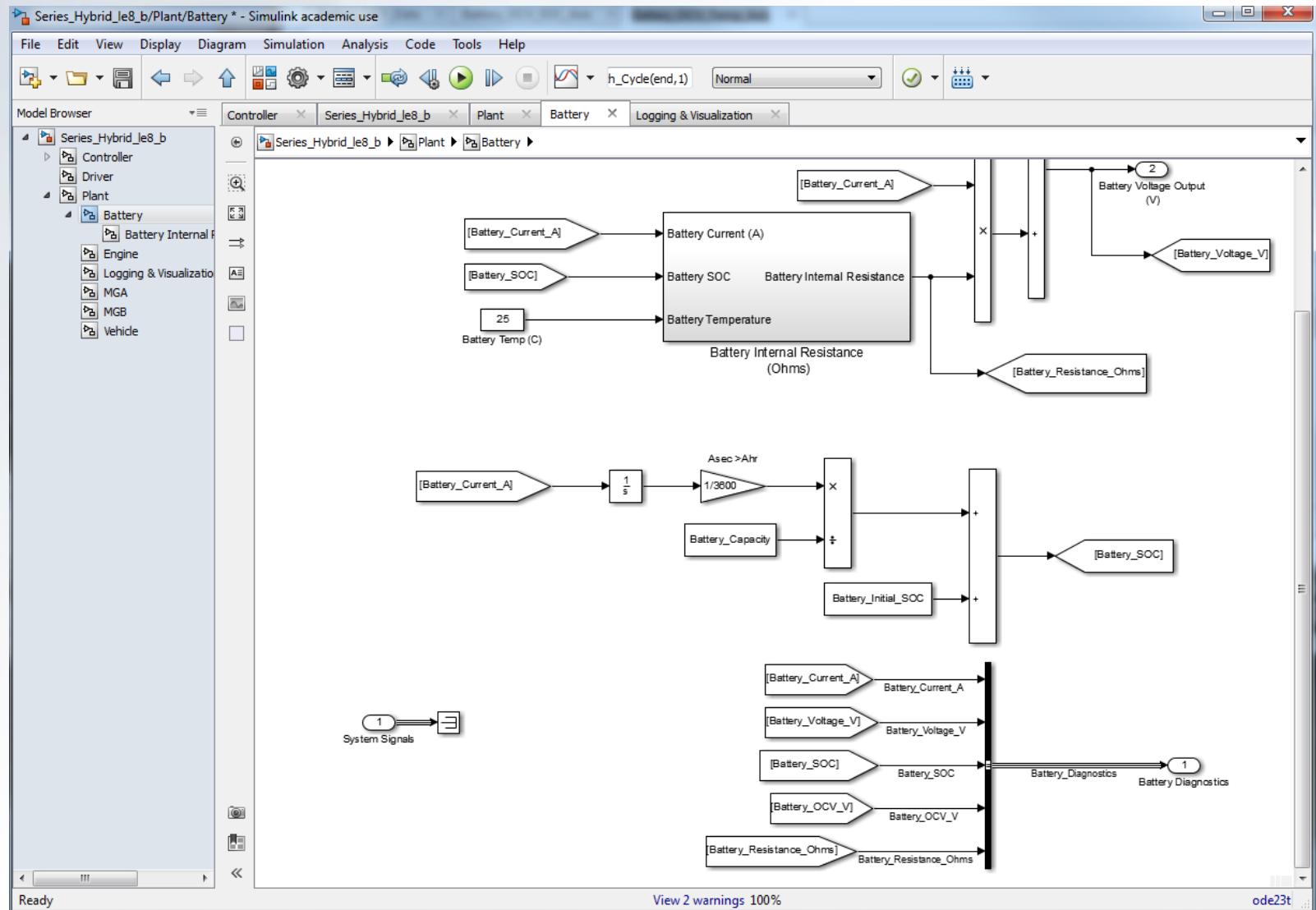
- The internal resistance of the battery is also a function of the temperature and SOC
- It also varies for charging and discharging
- Drag in
 - A Subsystem block
- And rename it
 - Battery Internal Resistance (Ohms)



Variable Internal Resistance

- It will have three inputs
 - Battery Current
 - Battery SOC
 - Battery Temperature
- Add one output
 - Battery Internal Resistance
- Put the Battery Resistance on the Diagnostics Bus and extract it in the Logger

Variable Internal Resistance



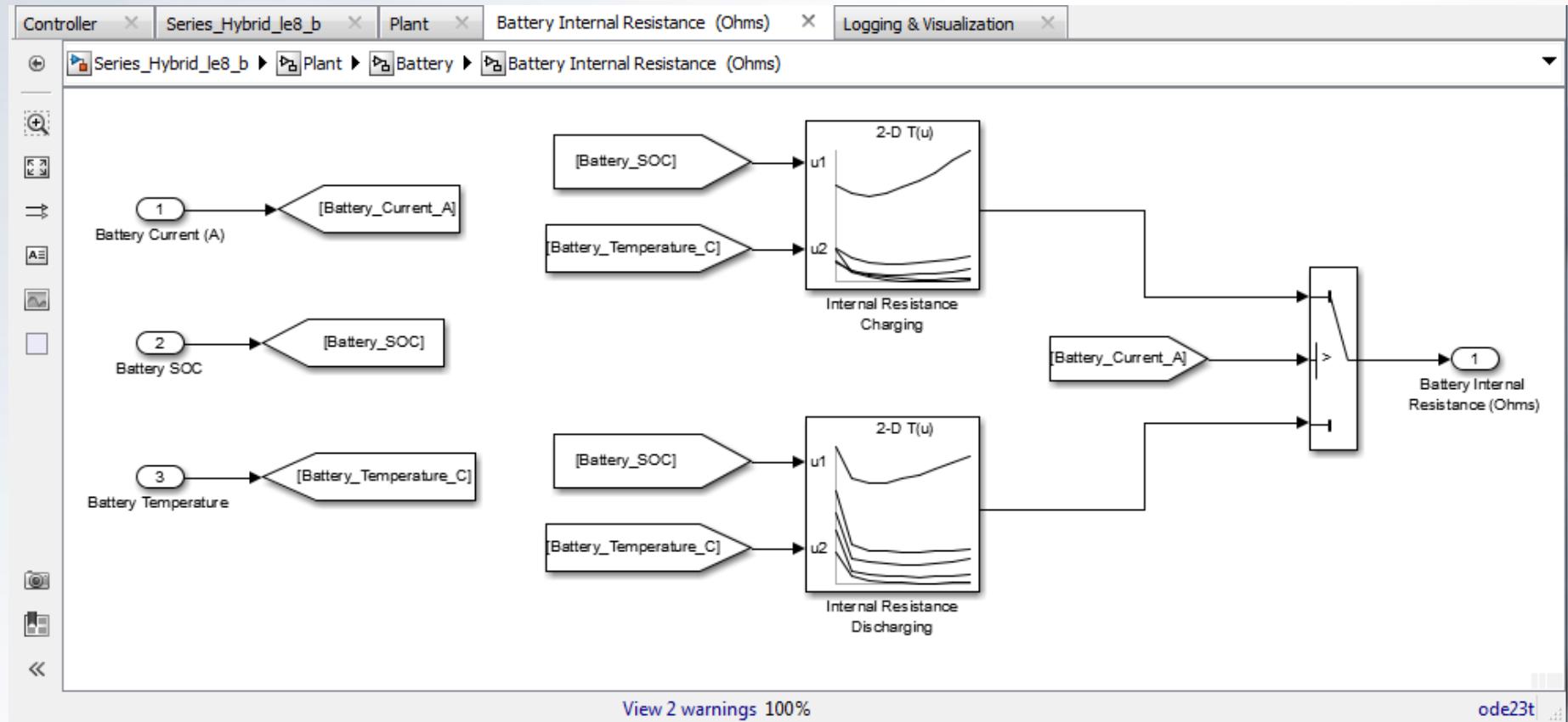


Variable Internal Resistance

- We will use the current to tell if we are charging or discharging
- From
 - [Simulink / Commonly Used Blocks](#)
- Drag a **Switch** block into the Battery Resistance subsystem
- Also drag in
 - two **2-D Lookup Table** blocks
 - three **Goto** and five **From** blocks



Variable Internal Resistance





Variable Internal Resistance

Block Parameters: Switch

Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The first and third input ports are data ports, and the second input port is the control port. The criteria for control port 2 are $u2 \geq \text{Threshold}$, $u2 > \text{Threshold}$ or $u2 \approx 0$.

Main Signal Attributes

Criteria for passing first input: $u2 > \text{Threshold}$

Threshold: 0

Enable zero-crossing detection

When the current is positive (charging) the data from the top table is passed

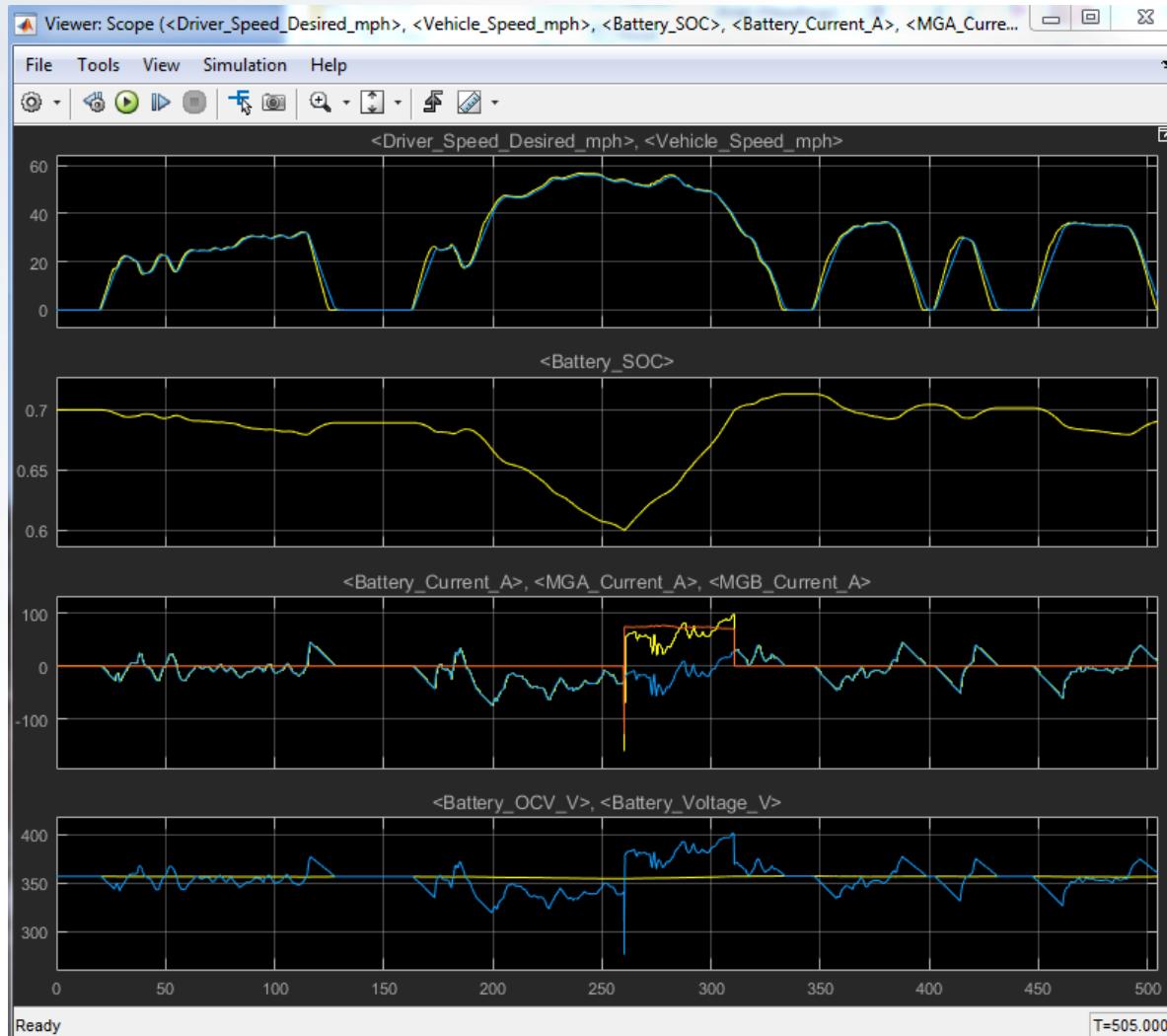
```
graph LR; BC[Battery Current_A] --> S(( )); S --> P1[1]; S --> P2[2]; S --> P3[3];
```

ode23t



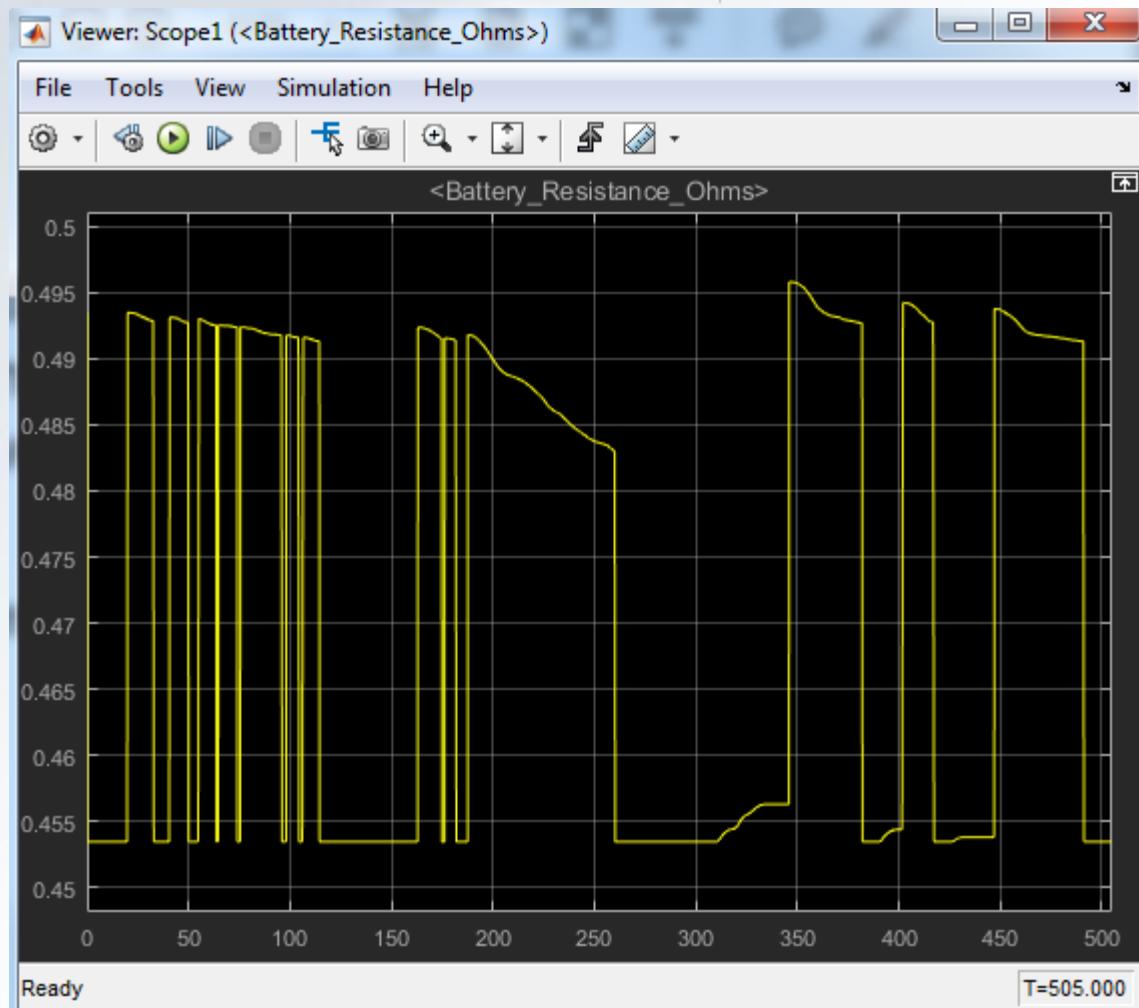
Variable Internal Resistance

- Put the resistance on a new scope and run the simulation





Results



Simulation Results

Total Distance Traveled (miles)	3.55068
Total Fuel Consumed (gal)	0.031732
Fuel Energy Consumed (BTU)	4232.83
Electrical Energy Consumed (BTU)	-411.518
MPGGE with SOC Correction	87.8212
MPG Diesel Only	111.896

OK

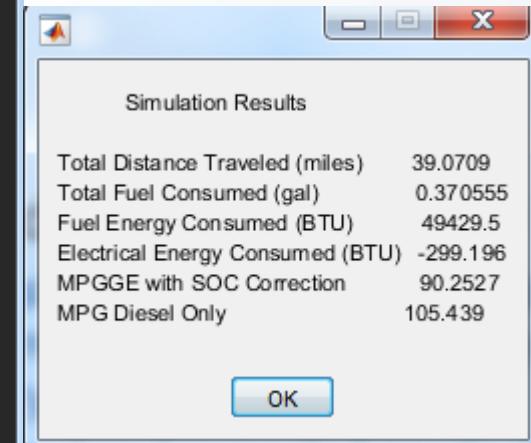
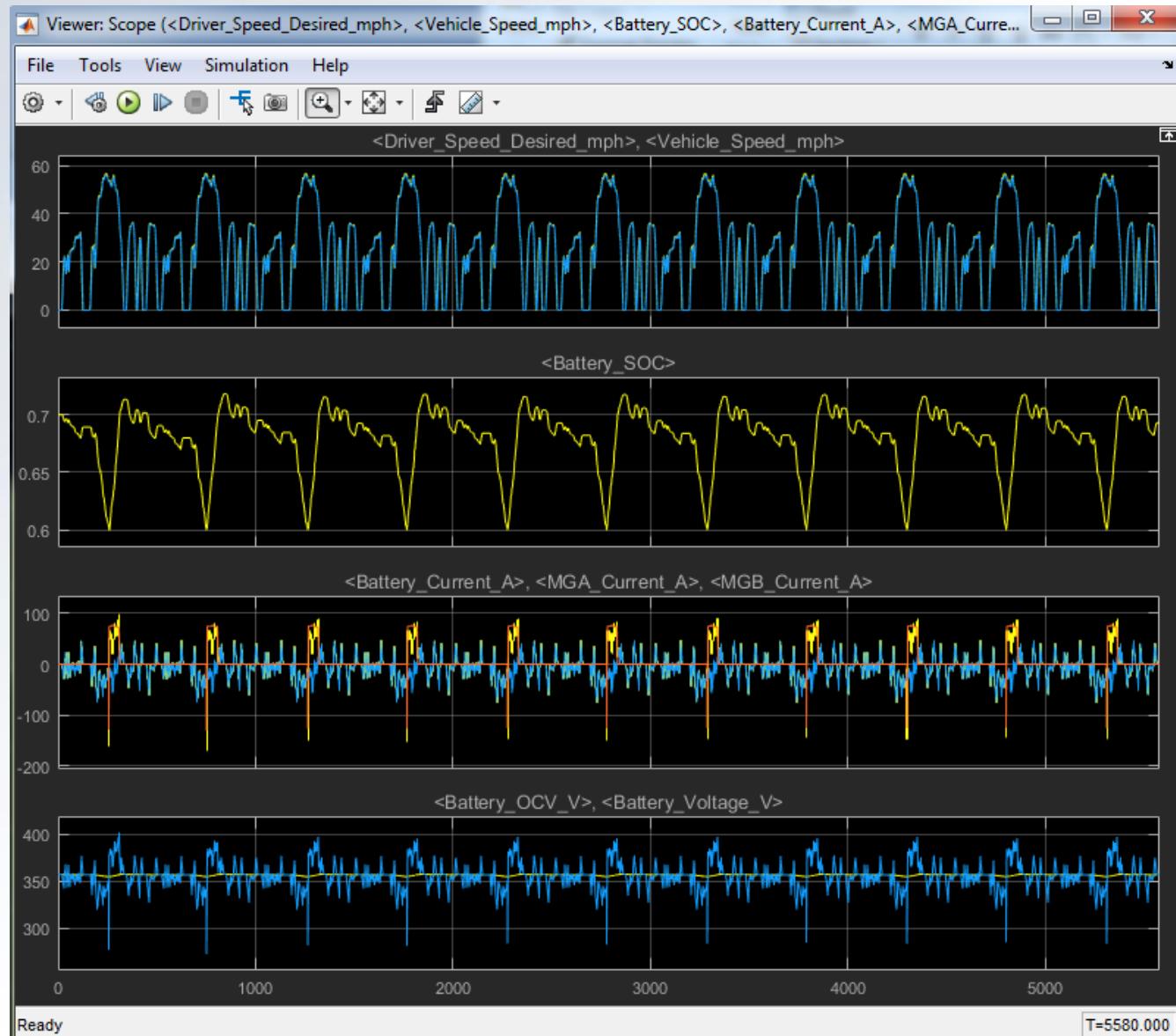


Results

- Higher internal resistance lowered the fuel economy
- Run the FU505_Ten_Times.mat drive cycle



Results



Review



- Improved the battery model
 - Used first order model battery model to incorporate internal resistance, making battery voltage a function of battery current
 - Discovered and corrected an algebraic loop with a memory block
 - Improved the model with open circuit voltage data based on SOC and battery temperature
 - Improved the model with variable charge and discharge internal resistances based on SOC and battery temperature

A blue-toned photograph of the Georgia Tech Campanile, a historic red brick building with a tall, spired tower. The word "TECH" is prominently displayed in large letters on the side of the tower. The image is framed by a thin yellow border.

MBSD Lecture 9

Improved MG Model



Outline

- Improved Motor/Generator model
 - Variable torque
 - Constant conversion efficiency
 - Variable conversion efficiency

MG Data



- At the Matlab command line type
 - clear variables
 - load Component_Data/MG_Data.mat

The screenshot shows the Matlab environment with two main windows: the Command Window and the Workspace browser.

Command Window:

```
>> clear variables
>> load Component_Data/MG_Data.mat
fx >> |
```

Workspace Browser:

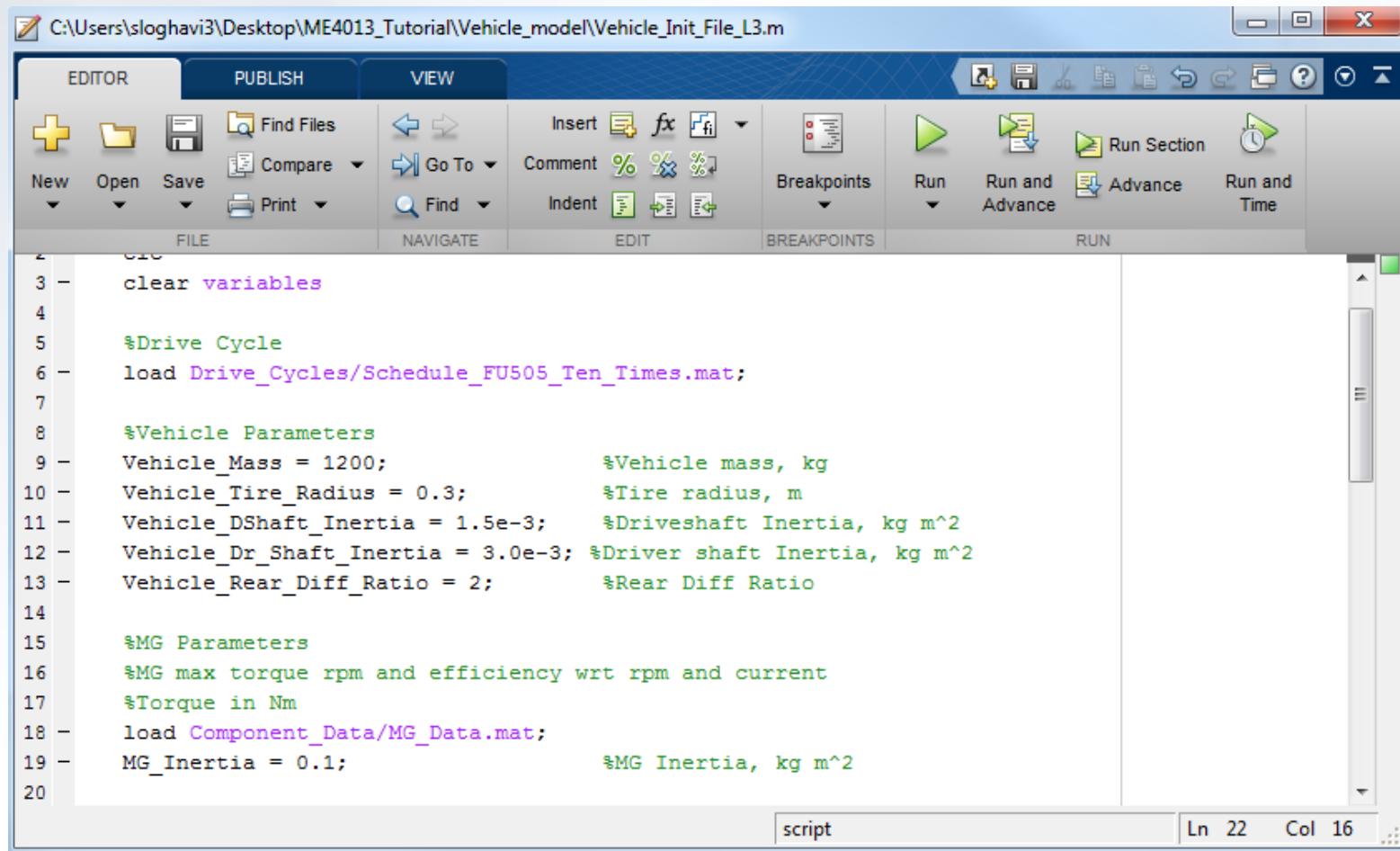
Name	Value
MG_Eff_Current_Axis	[6,50,94,138,182,226,270,314,358,402] 21x10 double
MG_Eff_Data	21x1 double
MG_Eff_RPM_Axis	1x16 double
MG_Torque_Data	1x16 double
MG_Torque_RPM_Axis	1x16 double

A blue callout bubble points to the 'MG_Eff_Current_Axis' entry in the workspace table, labeled "Efficiency 2-D Lookup Table". Another blue callout bubble points to the 'MG_Torque_Data' entry, labeled "Torque 1-D Lookup Table".



Initialization

- Add the MG data loading to the init file
- Also add an MG inertia of 0.1 kg m^2



A screenshot of a MATLAB IDE window titled "C:\Users\sloghavi3\Desktop\ME4013_Tutorial\Vehicle_model\Vehicle_Init_File_L3.m". The window has tabs for "EDITOR", "PUBLISH", and "VIEW". The "EDITOR" tab is selected, showing a script file with the following code:

```
1 %<<%
2 %>>%
3 clear variables
4
5 %Drive Cycle
6 load Drive_Cycles/Schedule_FU505_Ten_Times.mat;
7
8 %Vehicle Parameters
9 Vehicle_Mass = 1200; %Vehicle mass, kg
10 Vehicle_Tire_Radius = 0.3; %Tire radius, m
11 Vehicle_DShaft_Inertia = 1.5e-3; %Driveshaft Inertia, kg m^2
12 Vehicle_Dr_Shaft_Inertia = 3.0e-3; %Driver shaft Inertia, kg m^2
13 Vehicle_Rear_Diff_Ratio = 2; %Rear Diff Ratio
14
15 %MG Parameters
16 %MG max torque rpm and efficiency wrt rpm and current
17 %Torque in Nm
18 load Component_Data/MG_Data.mat;
19 MG_Inertia = 0.1; %MG Inertia, kg m^2
20
```

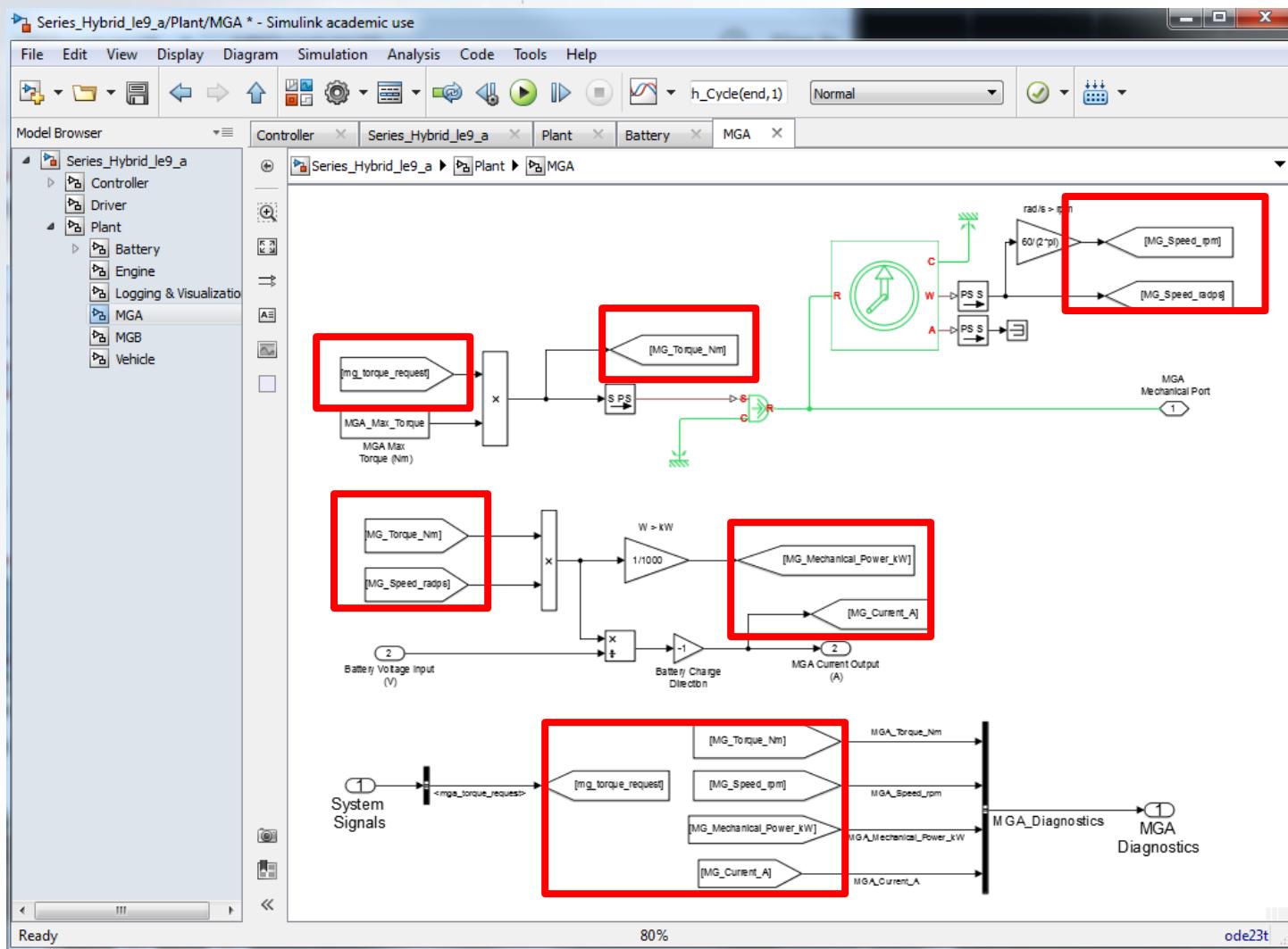
The status bar at the bottom right shows "script" in the active tab, "Ln 22 Col 16" for the current line and column, and "380" for the page number.



Variable Torque

- We will start with the MGA subsystem
 - Once we have it refined, we will copy it to MGB
- Rename your model
 - Series_Hybrid_le9_a.slx
- Go to the MGA subsystem
 - Change the name of all Goto and From blocks to MG

Variable Torque



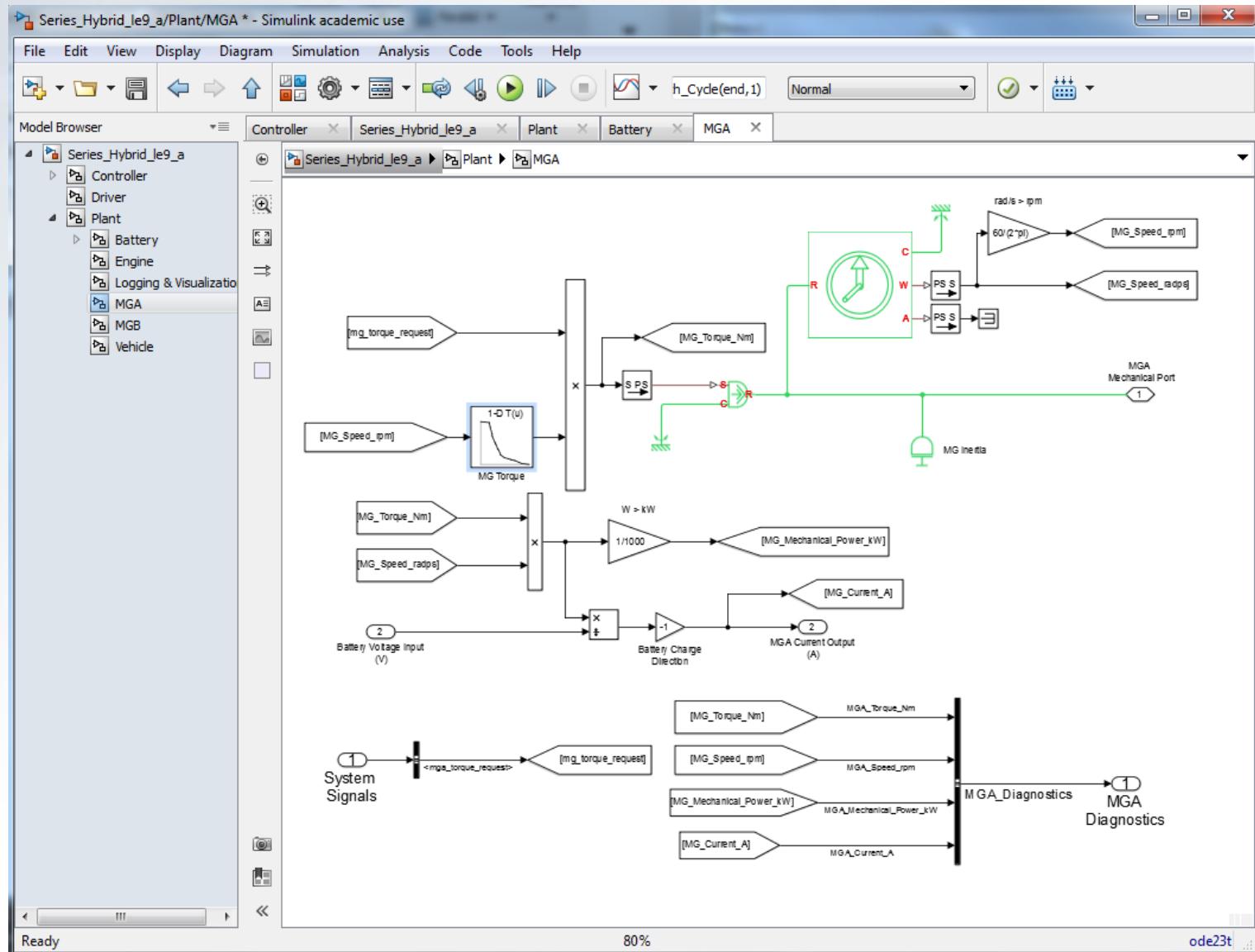


Variable Torque

- Drag in
 - An Inertia block
 - A 1-D Lookup Table block
- Delete the MGA_Max_Torque block
- Arrange as shown on the next slide
- Add the
 - MG_Torque_RPM_Axis
 - MG_Torque_Data
- to the Lookup Table



Variable Torque



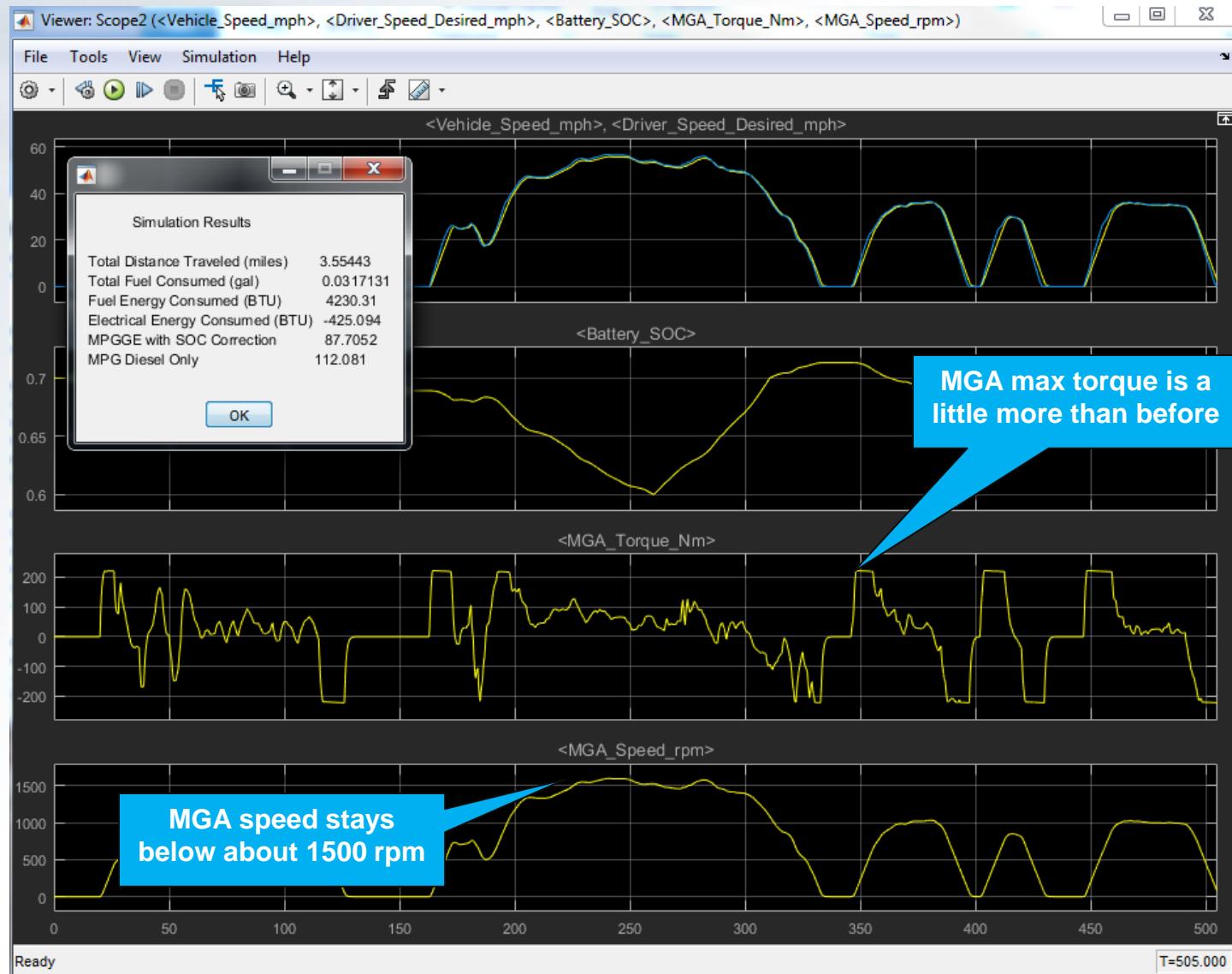


Results

- Let's create a new scope
 - Axis 1 – Vehicle speed (desired and actual)
 - Axis 2 – Battery SOC
 - Axis 3 – MGA Torque
 - Axis 4 – MGA Speed rpm
- Run the simulation for the FU505 drive cycle



Results





Efficiency

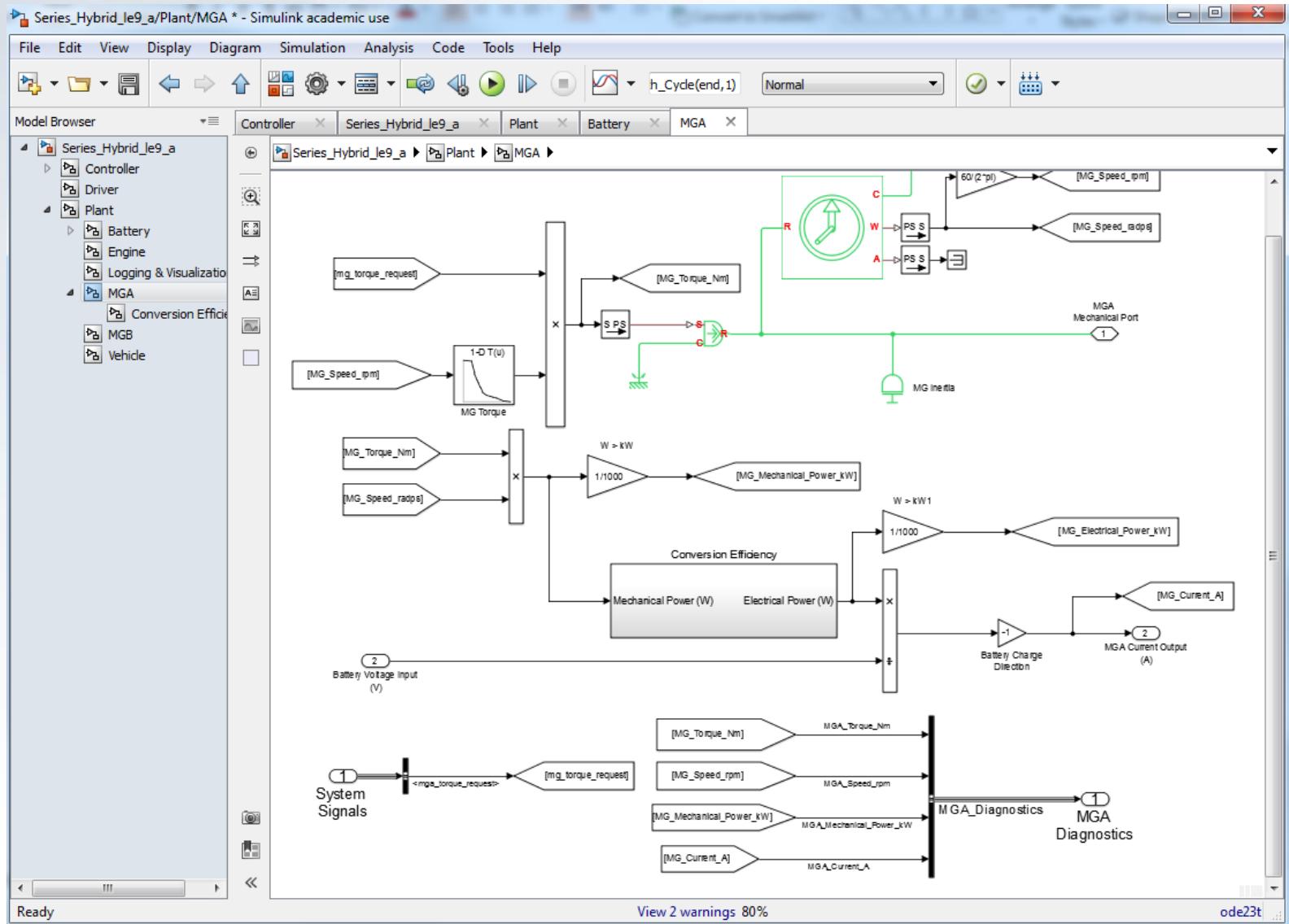
- Electromechanical energy conversion can go two ways
 - Electrical to mechanical
 - Accelerating the vehicle
 - Mechanical to Electrical
 - Regenerative braking
- This is a little more challenging than the battery efficiency
- We will start by assuming a constant conversion efficiency of 0.85



Efficiency

- Drag a **Subsystem** block into MGA and
- Rename
 - **Subsystem** to Conversion Efficiency
 - In1 to Mechanical Power (W)
 - Out1 to Electrical Power (W)
- Add a **Gain** and a **Goto** for the Electrical Power

Efficiency





Efficiency

- Go to the Conversion Efficiency subsystem
- Drag in
 - A Constant block
 - A Switch block
 - Two Product blocks
 - A Goto block
 - Two From blocks

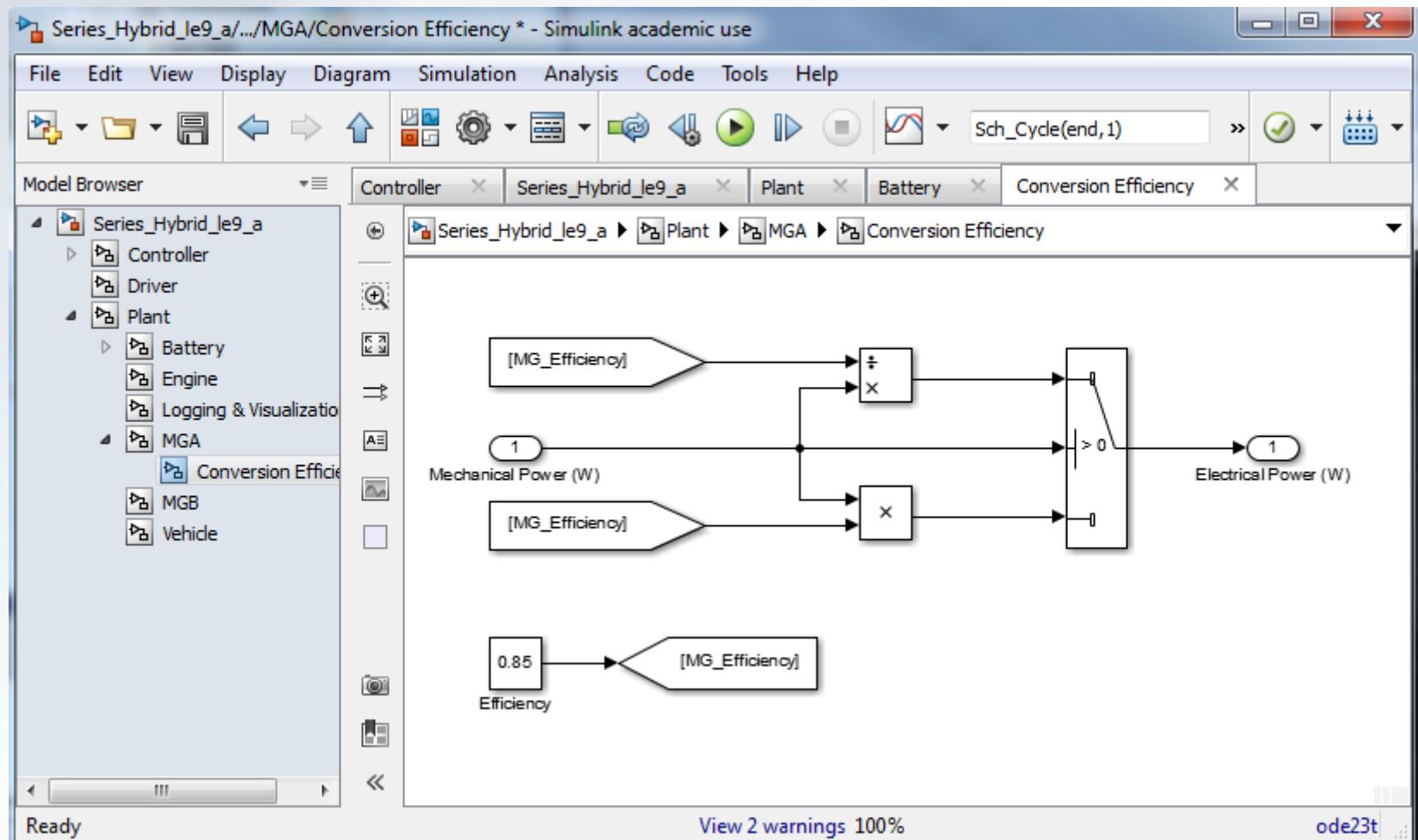


Efficiency

- If the mechanical power is positive
 - Top of switch
 - The motor is accelerating the vehicle
 - A higher electrical power is required
 - Therefore divide MP by the efficiency
- If the mechanical power is negative
 - Bottom of switch
 - The generator is making power
 - A lower electrical power is made
 - Therefore multiply MP by the efficiency
- Arrange as shown on the next slide



Efficiency



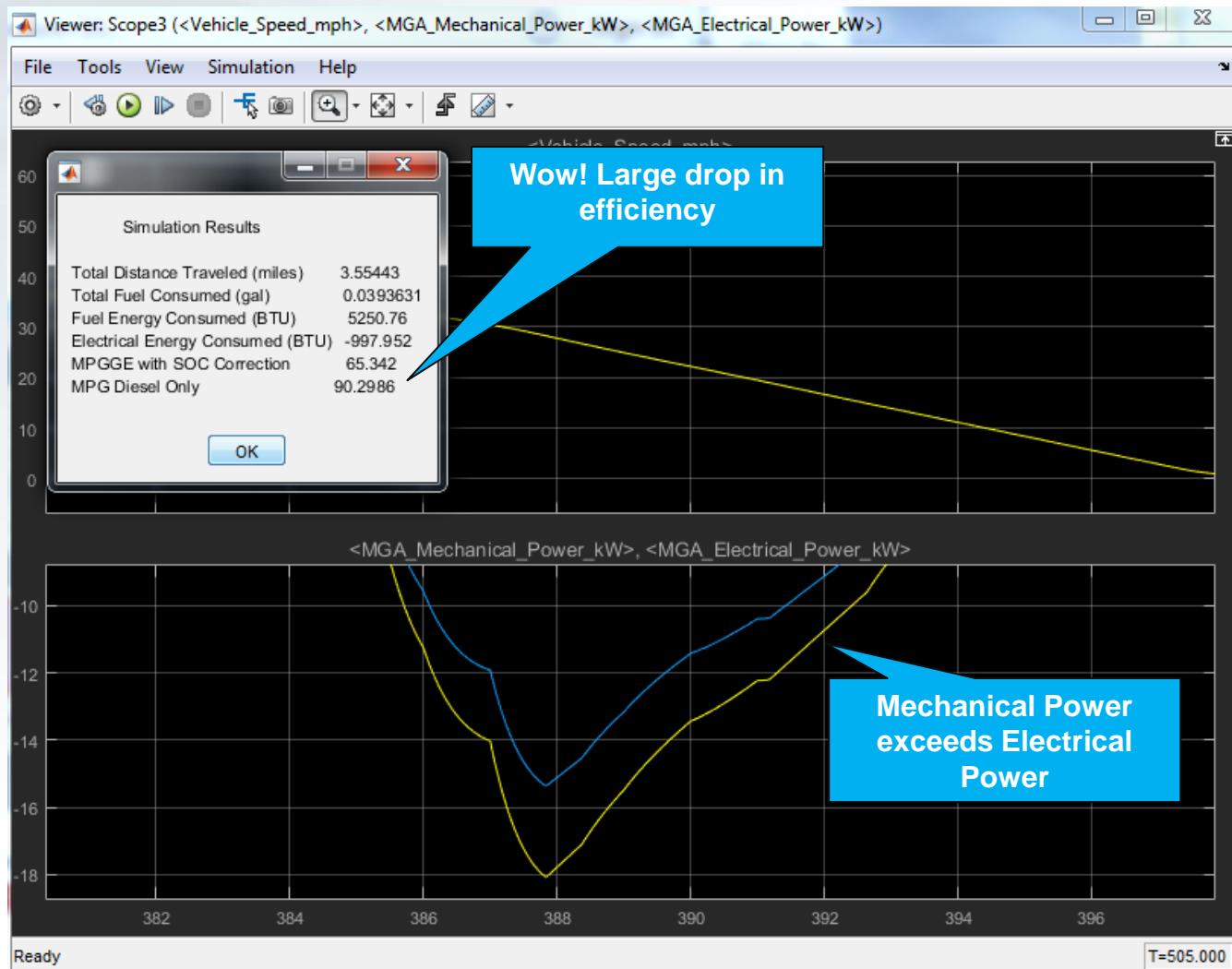


Efficiency

- Add the Electrical Power to the Diagnostics bus
- Extract it in the Logging & Visualization subsystem
- Create a scope
 - Axis 1 – Vehicle Speed (actual)
 - Axis 2 – Mechanical and Electrical Power



Results





Variable Efficiency

- Conversion efficiency is a function of both MG speed and current
- Let's add that in
- Rename your model
 - Series_Hybrid_le9_b.slx
- In the Conversion Efficiency subsystem
 - Delete the Efficiency block and Goto/From blocks

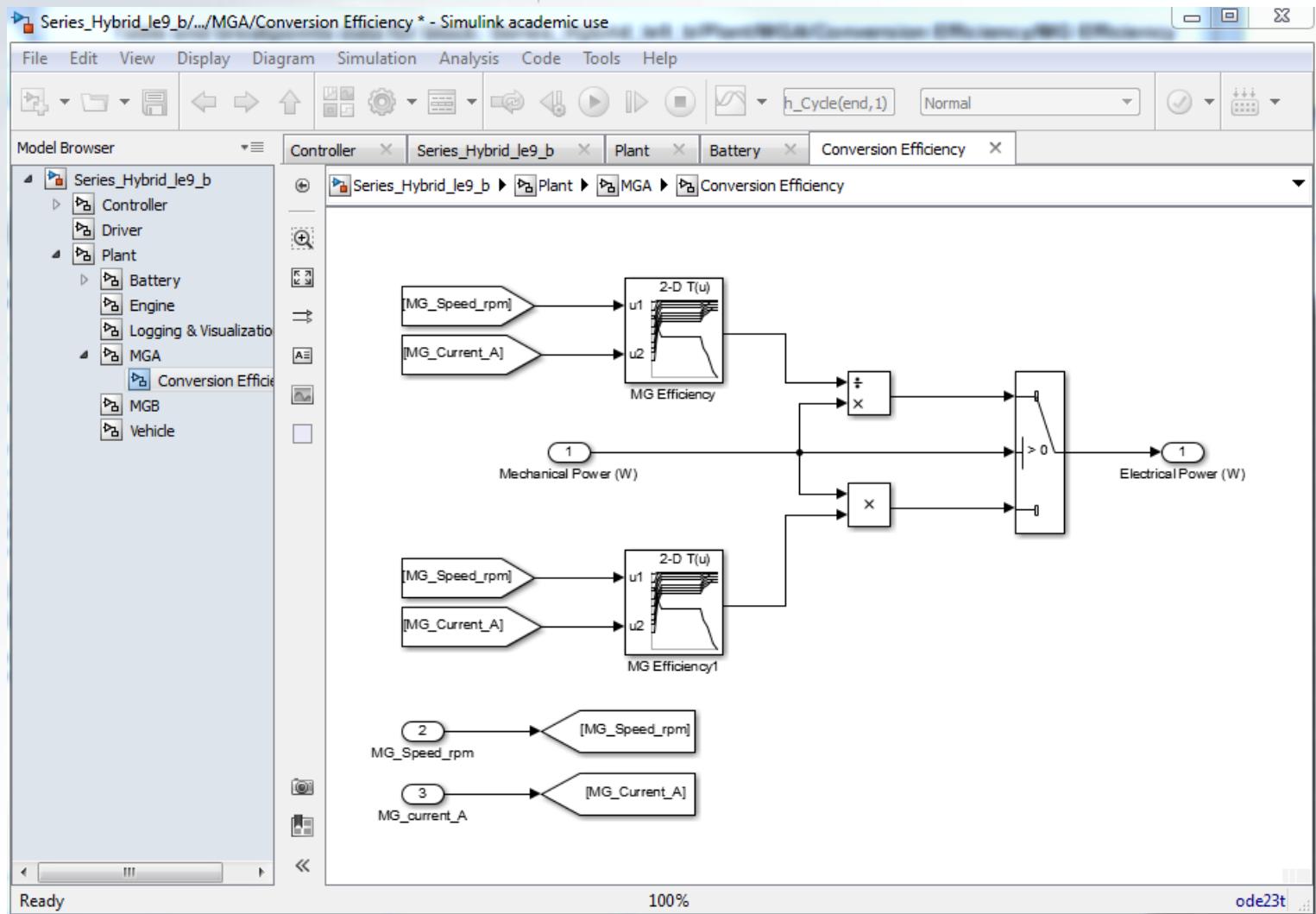


Variable Efficiency

- Drag in
 - Two 2-D Lookup Table blocks
 - Two In1 blocks
 - Two Goto blocks
 - Four From blocks
- Arrange and label as shown on the next slide



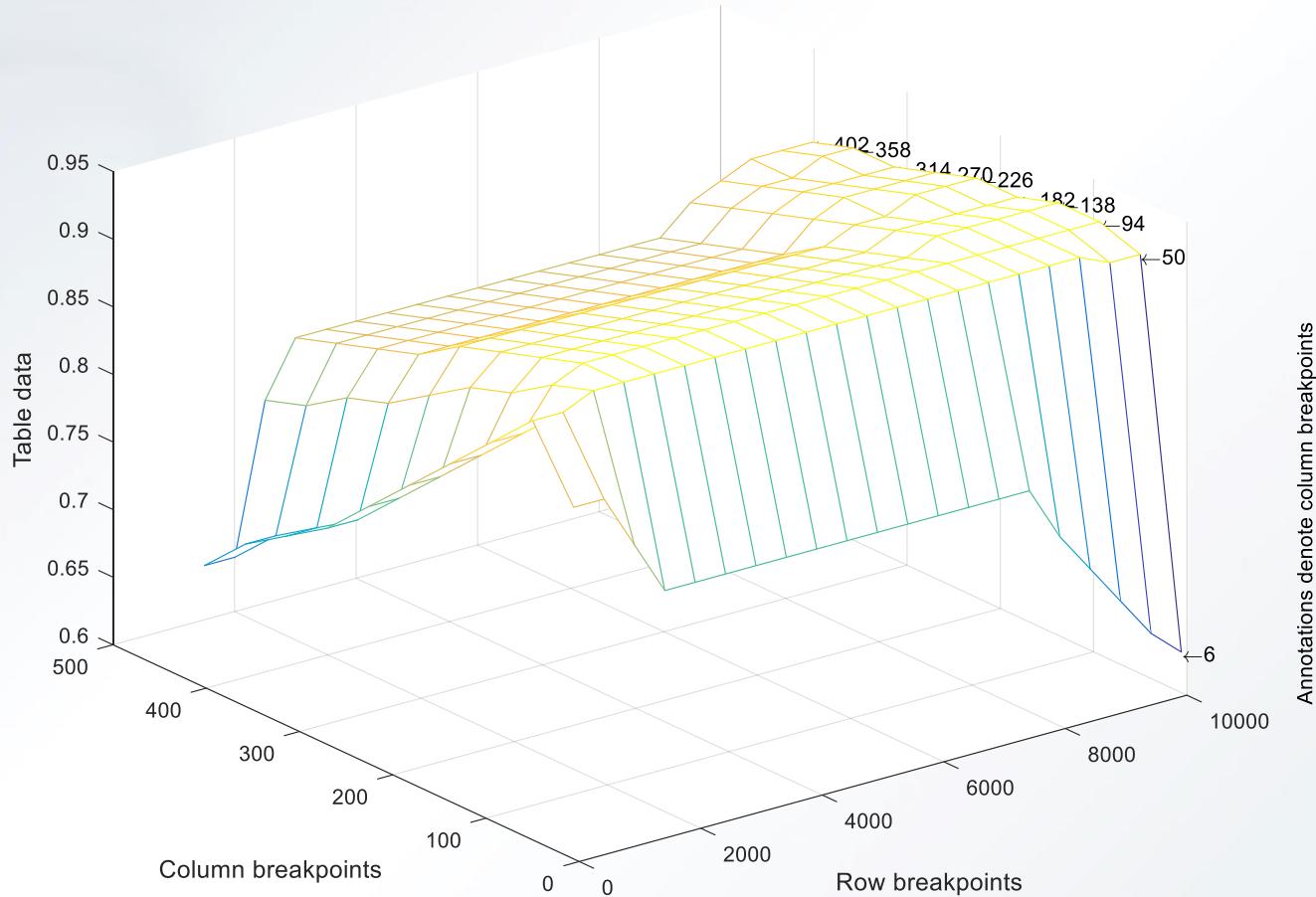
Variable Efficiency





Variable Efficiency

Table and breakpoints data for block: Series_Hybrid_le9_b/Plant/MGA/Conversion Efficiency/MG Efficiency



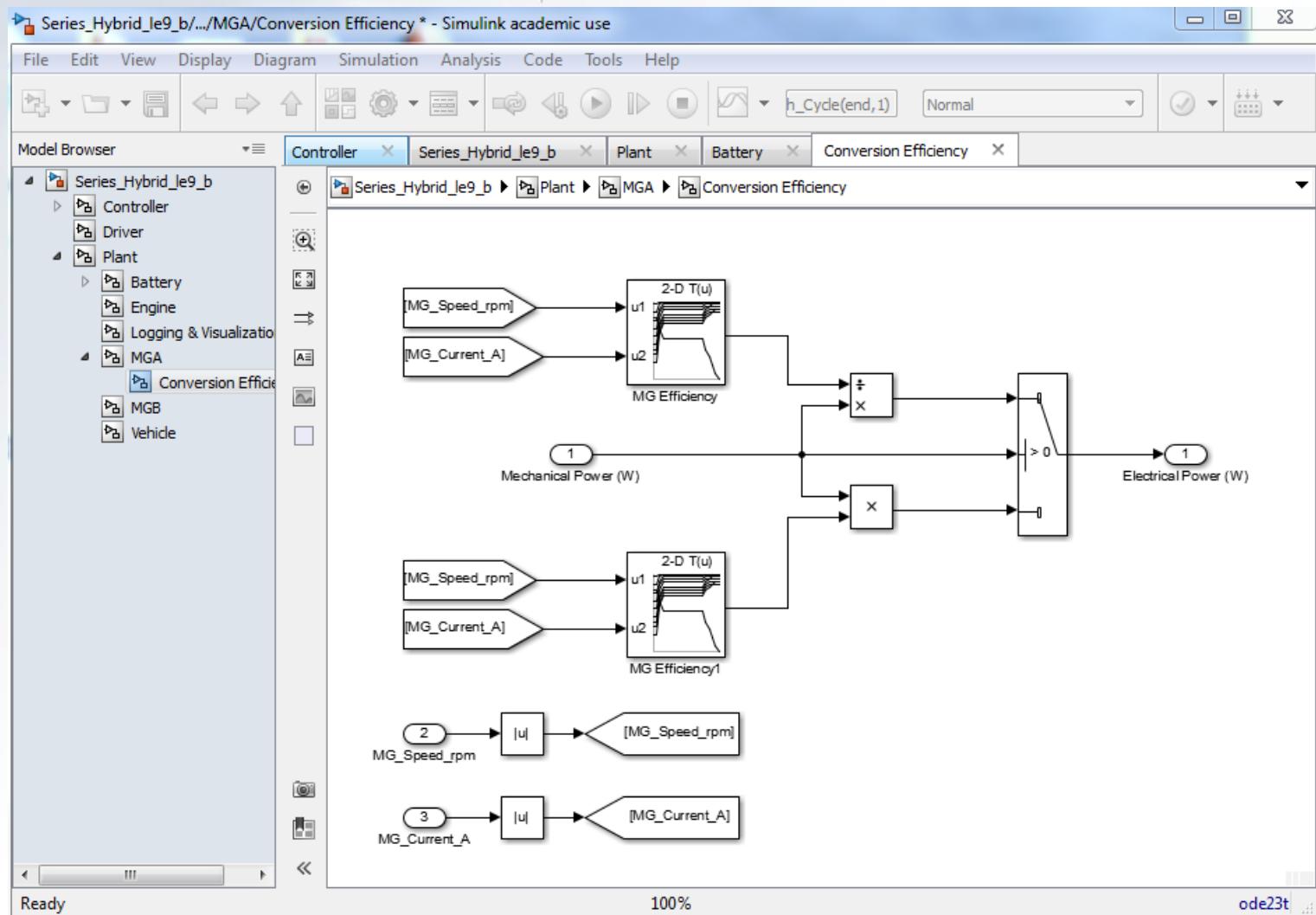


Variable Efficiency

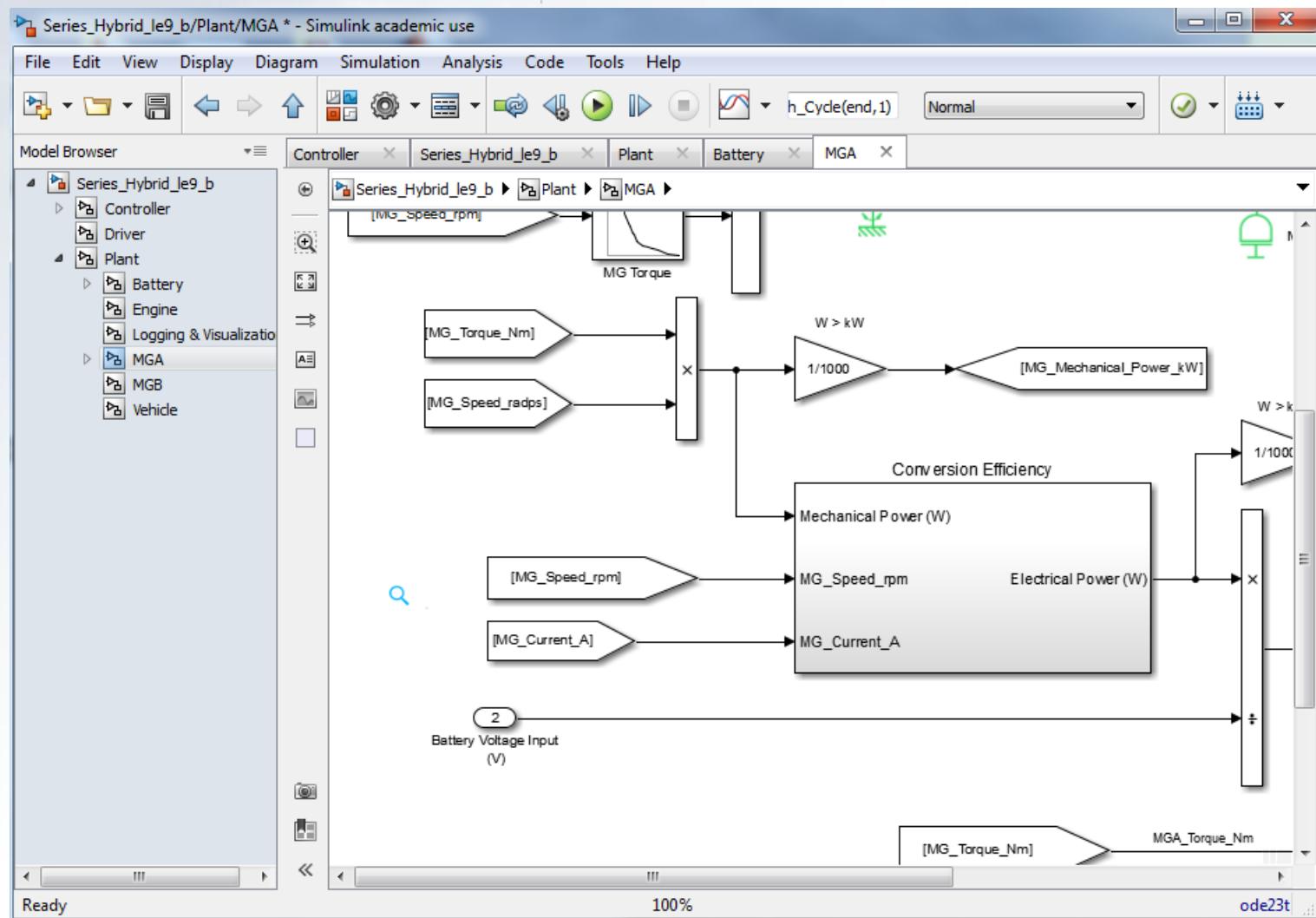
- Hmm, our efficiency data is for positive values of current and MG speed
 - Can either of these be negative?
- From
 - Simulink / Math Operations
- Drag in two **Absolute Value** blocks
- Return to the MGA subsystem
 - Connect the necessary **From** blocks
- Run the simulation



Variable Efficiency

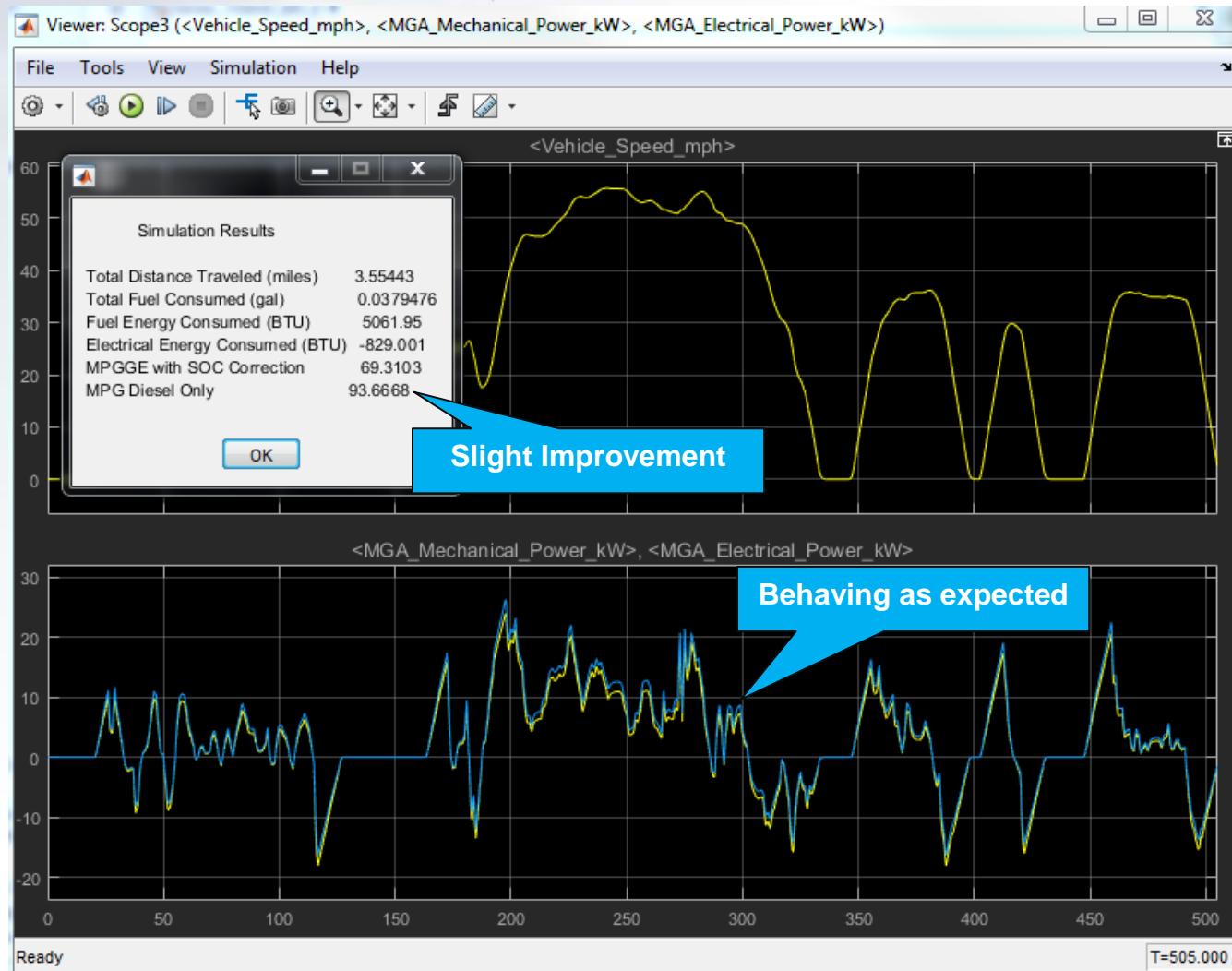


Variable Efficiency





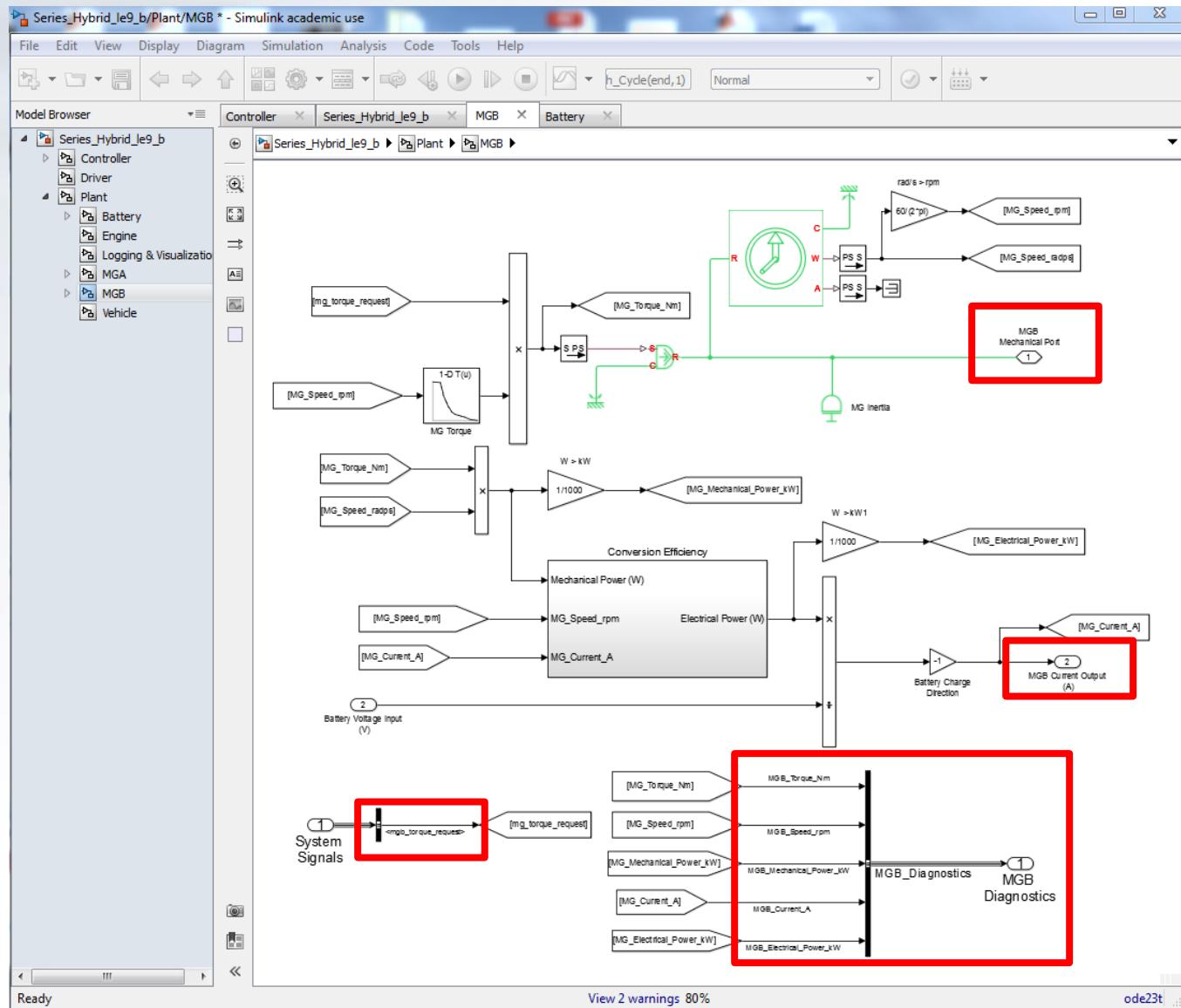
Variable Efficiency





Variable Efficiency

- Outstanding, we now have a pretty good model of a motor/generator
- Go to the Plant level of the model
 - Delete MGB
 - Copy MGA
 - Update the names accordingly
- Extract the MGB electrical power in the Logging & Visualization subsystem





Results

- One last new scope
 - Axis 1 – Vehicle speed (actual and desired)
 - Axis 2 – SOC
 - Axis 3 – Battery, MGA, & MGB current
 - Axis 4 – Engine rpm
- Run the Simulation



Results





Results

- Aha – our desired engine speed is 1800 rpm
 - Engine torque at 0.5 throttle 145 Nm
 - MGB max torque 140 Nm
- MGB can't control the engine!
- Decrease the desired engine speed to 1500 rpm



Results





Review

- Improved MG Model
 - Variable maximum torque wrt MG speed
 - Constant efficiency to explore electromechanical conversion losses
 - Variable efficient wrt MG speed and current



Future Direction

- We blew up our battery
 - Need to add current limits
- We blew up our engine
 - Need to add an emergency defuel at redline engine speeds
- We are destroying the engine
 - Need to limit MGB speed for starting engine
- We haven't done that much control work
 - Other ways to control the system?