



Advanced Model-Based-System Design

Marc Herniter

EE - Rose-Hulman Institute of Technology

Zac Chambers

ME - Rose-Hulman Institute of Technology



Course Outline

2

- Modeling a series hybrid-electric vehicle
 - Introduction to Simulink and SimDriveline
 - Models for the Driver, Battery, and Electric Motors.
 - Creating and Running Drive Cycles
 - Models for Engines.
 - Developing the hybrid-electric vehicle controller.
 - Measuring and predicting vehicle performance.





Course Outline

3

- Real-Time Simulations (xPC)
 - Stand-Alone Simulations
 - Verify logical operation
 - Give user feel of controls and vehicle operation
 - Plant and controller on same target
- Introduction to CAN
 - Message IDs
 - Scaling and Offset
 - Big Endian and Little Endian
 - CAN Message Database
 - Cabling, isolation, and termination



Course Outline

4

- Introduction to MotoHawk MotoTune tools.
- HIL Simulations (Real-Time)
 - Separate the Plant from the Controller.
 - Controller on real-time target.
 - Plant on real-time target.
 - V&V Using HIL RT Model
 - Set up a standard set of tests for the series controller.
 - Run standard set of tests, record and report results, indicate faults.
 - Verify communications interfaces and A/D inputs and outputs.
 - Verify that controller can execute control algorithm in specified time step.
 - Verify Communication data rates.





Advanced Model-Based-System Design

Building a Large System Model



Part 1

6

- Develop a basic model for a series hybrid electric vehicle with models for
 - Engine
 - Motor/Generator
 - Battery
 - Driver
 - Powertrain
- Develop a controller for the vehicle



7

- 



Electric Vehicle

9

- We will begin by creating a model for a rear-wheel vehicle.
- In this model, an electric motor is coupled directly to the rear wheels through a differential.
- The specs of the vehicle are:
 - Vehicle Mass: 3600 lbs
 - Tire Radius: 16 inches
 - Rear-Differential Ratio 3.73

MotoTron

 **The MathWorks**

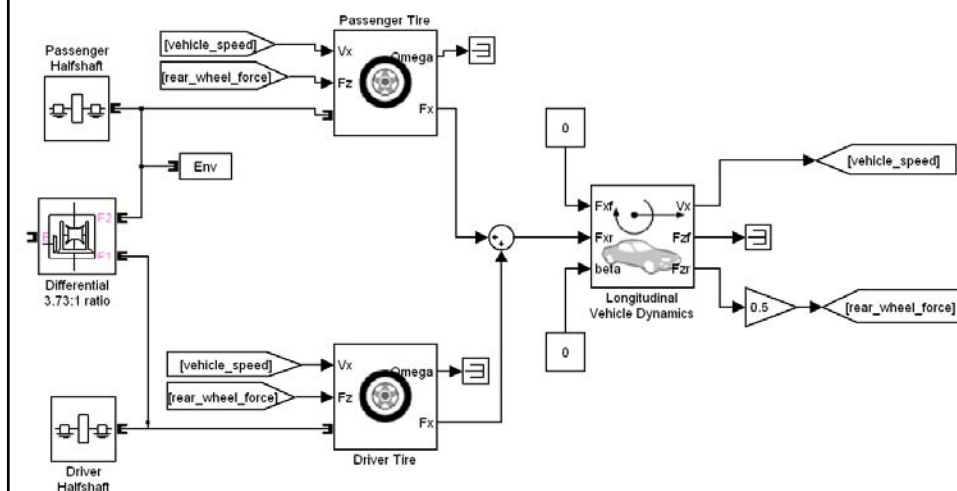
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Drive Train

10

- The drive train we will create is shown below:





Longitudinal Vehicle Dynamics

11

- The Longitudinal Vehicle Dynamics block solves for the speed of the vehicle given:
 - An input force F_{xr} .
 - A specified road grade (beta) in degrees.
- The calculation includes aerodynamic drag.
- The block also calculates the normal force on each wheel of the vehicle, which is needed by the tire model.

MotoTron

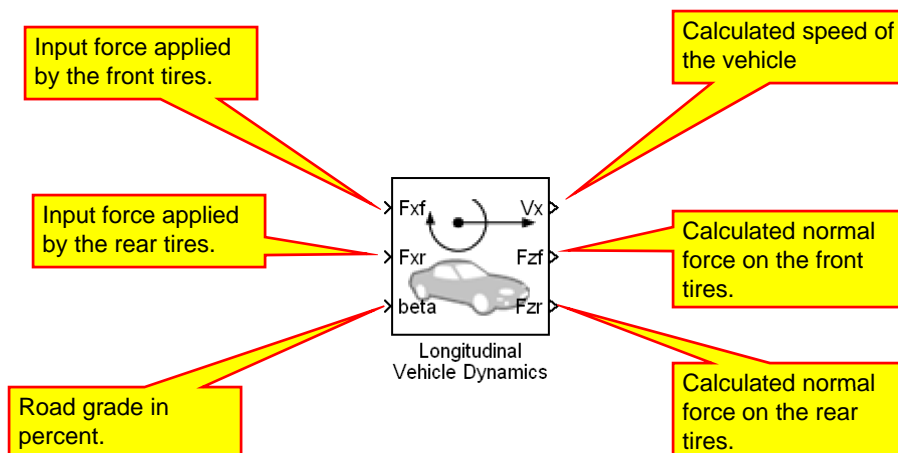
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Longitudinal Vehicle Dynamics

12



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Longitudinal Vehicle Dynamics

13

- The Longitudinal Vehicle Dynamics block is located in the **Simscape / SimDriveline / Vehicle Components** library.
- Place a block in your model and then double-click on it to set its parameters.
- We would like to understand how this block works and what is inside this block.
- Set the parameters as shown next:



Function Block Parameters: Longitudinal Vehicle Dynamics

Longitudinal Vehicle Dynamics (mask) (link)

Vehicle of two axes riding on an incline. The CG parameters specify the positions of the vehicle's center of gravity and axes. The aerodynamic parameters include the vehicle's frontal area and drag coefficient. The vehicle does not pitch or move vertically. See block reference page for details.

The input signals F_{xf} , F_{xr} , and β specify the longitudinal forward and rear contact forces (N) and the incline angle (rad), respectively. The output signals V_x , F_{af} , and F_{ar} indicate the vehicle velocity (m/s) and forward and rear vertical load forces (N), respectively. The loads are positive downwards.

Parameters

Mass [kg]:

Horizontal distance from CG to front axle [m]:

Horizontal distance from CG to rear axle [m]:

CG height from ground [m]:

Frontal area [m²]:

Drag coefficient:

Initial longitudinal velocity [m/s]:

OK Cancel Help Apply

14

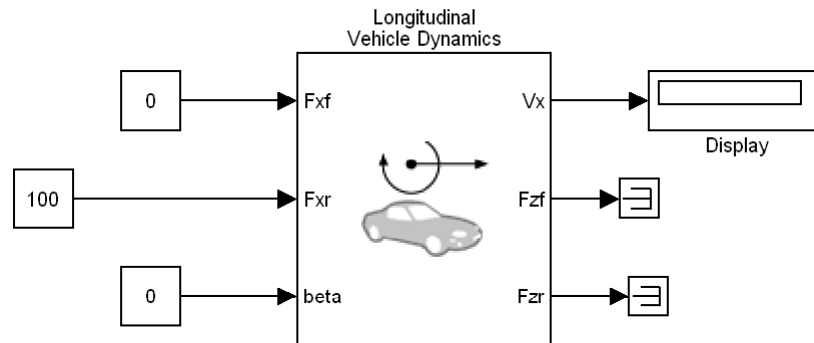
- We have set:
 - the mass of the vehicle to 1000 kg.
 - The frontal area to zero.
 - The drag coefficient to 0.
- With this model, the model reduces to a force accelerating a mass.





Create the following model

15



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Longitudinal Vehicle Dynamics

16

- All we are doing applying a 100 N force to a 1000 kg mass.
- The drag was set to zero.
- The road elevation was set to zero.
- Thus, if we divide the applied force by the vehicle mass and integrate, we should be able to calculate the vehicle's speed.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Lecture 1 Exercise 1

17

- Calculate by hand the speed of the vehicle after 100 seconds assuming a vehicle mass of 100 kg and an applied force of 100 N
- Use basic Simulink blocks to make the same calculation.
- Compare the vehicle speed using all three methods:
 - Longitudinal Vehicle Dynamics Block _____
 - Simulink Basic Blocks _____
 - Hand Calculations _____

Demo _____

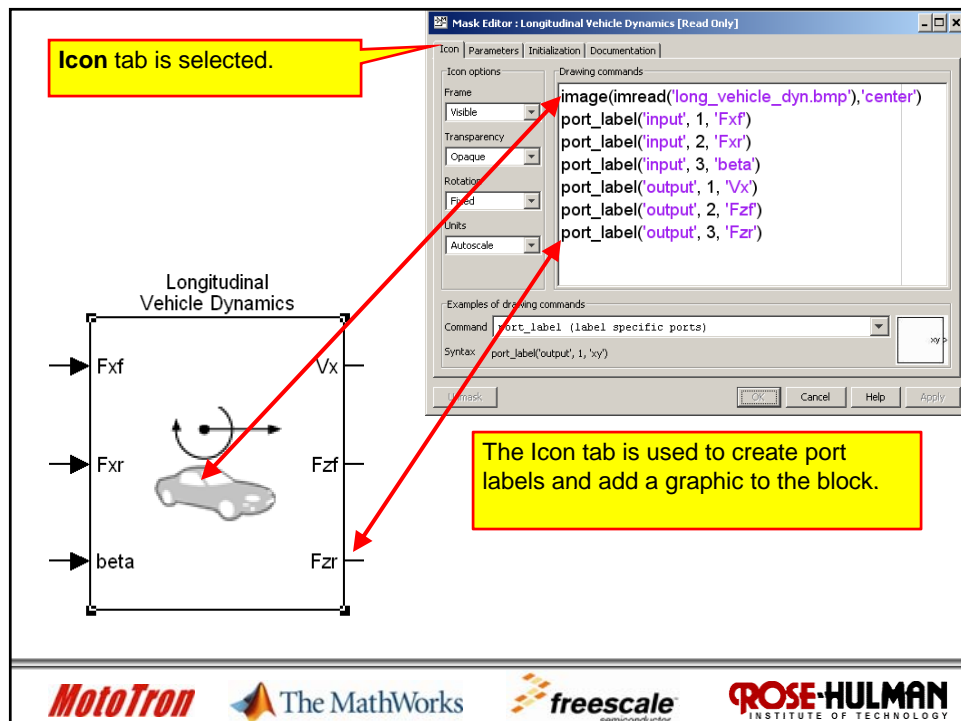
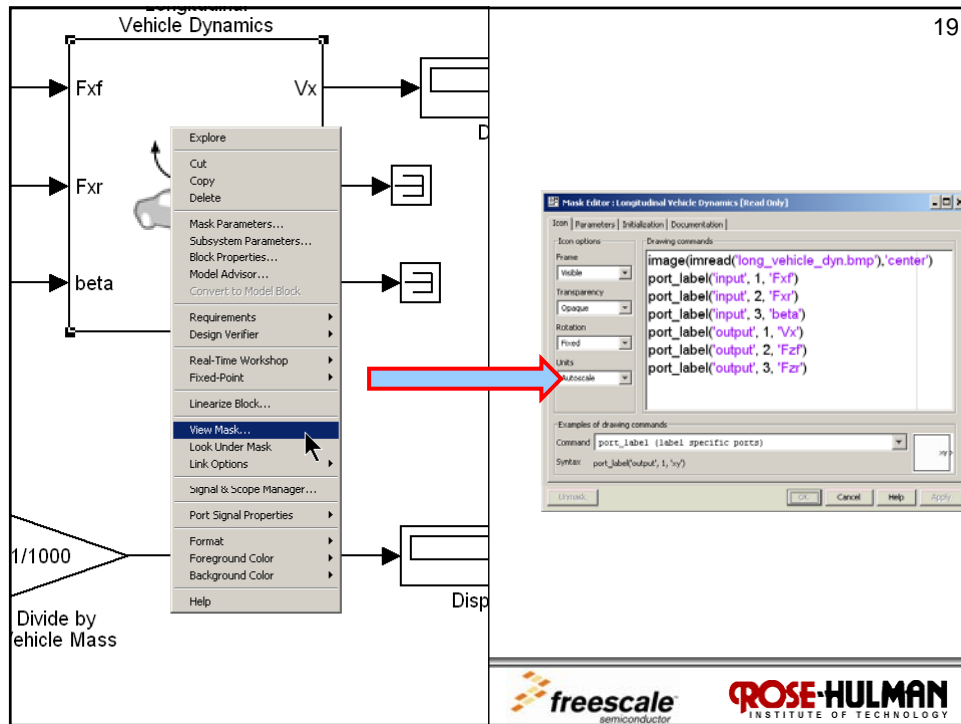


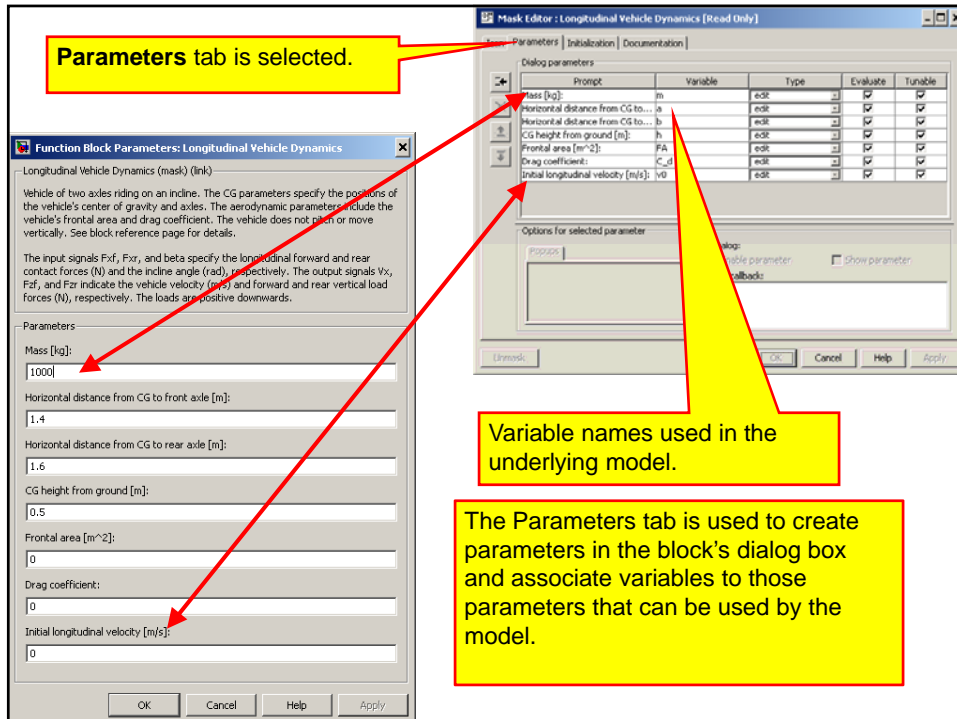
Longitudinal Vehicle Dynamics

18

- Next, we would like to look inside the block. We will do this in two steps:
 - First look at the mask to see how parameters are passed to the underlying model.
 - Second, look under the mask to see the Simulink used to implement this block.
- Right-click on the Longitudinal Dynamics Block and select **View Mask** from the menu:







Parameters tab is selected.

Variable names used in the underlying model.

The Parameters tab is used to create parameters in the block's dialog box and associate variables to those parameters that can be used by the model.

Function Block Parameters: Longitudinal Vehicle Dynamics

Longitudinal Vehicle Dynamics (mask) (link)

Vehicle of two axes riding on an incline. The CG parameters specify the positions of the vehicle's center of gravity and axes. The aerodynamic parameters include the vehicle's frontal area and drag coefficient. The vehicle does not pitch or move vertically. See block reference page for details.

The input signals F_{x1} , F_{x2} , and β specify the longitudinal forward and rear contact forces (N) and the incline angle (rad), respectively. The output signals y_x , F_{z1} , and F_{z2} indicate the vehicle velocity (m/s) and forward and rear vertical load forces (N), respectively. The loads are positive downwards.

Parameters

Mass [kg]:

1000

Horizontal distance from CG to front axle [m]:

1.4

Horizontal distance from CG to rear axle [m]:

1.6

CG height from ground [m]:

0.5

Frontal area [m²]:

0

Drag coefficient:

0

Initial longitudinal velocity [m/s]:

0

OK Cancel Help Apply

Mask Editor: Longitudinal Vehicle Dynamics [Read Only]

Parameters Initialization Documentation

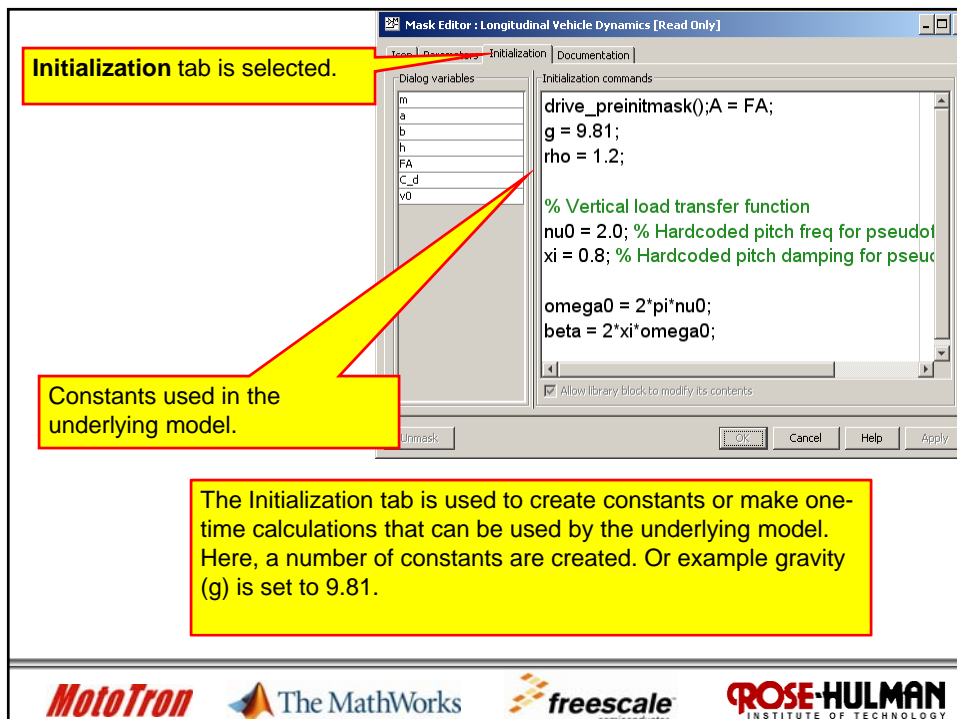
Dialog parameters

Prompt	Variable	Type	Evaluate	Tunable
Mass [kg]:	m	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Horizontal distance from CG to front axle [m]:	a	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Horizontal distance from CG to rear axle [m]:	b	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
CG height from ground [m]:	h	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Frontal area [m ²]:	FA	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Drag coefficient:	C_d	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Initial longitudinal velocity [m/s]:	v0	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Options for selected parameter

Position: ☐ Show parameter

Unmask OK Cancel Help Apply



Initialization tab is selected.

Constants used in the underlying model.

The Initialization tab is used to create constants or make one-time calculations that can be used by the underlying model. Here, a number of constants are created. For example gravity (g) is set to 9.81.

Mask Editor: Longitudinal Vehicle Dynamics [Read Only]

Parameters Initialization Documentation

Dialog variables

m
a
b
h
FA
C_d
v0

Initialization commands

```

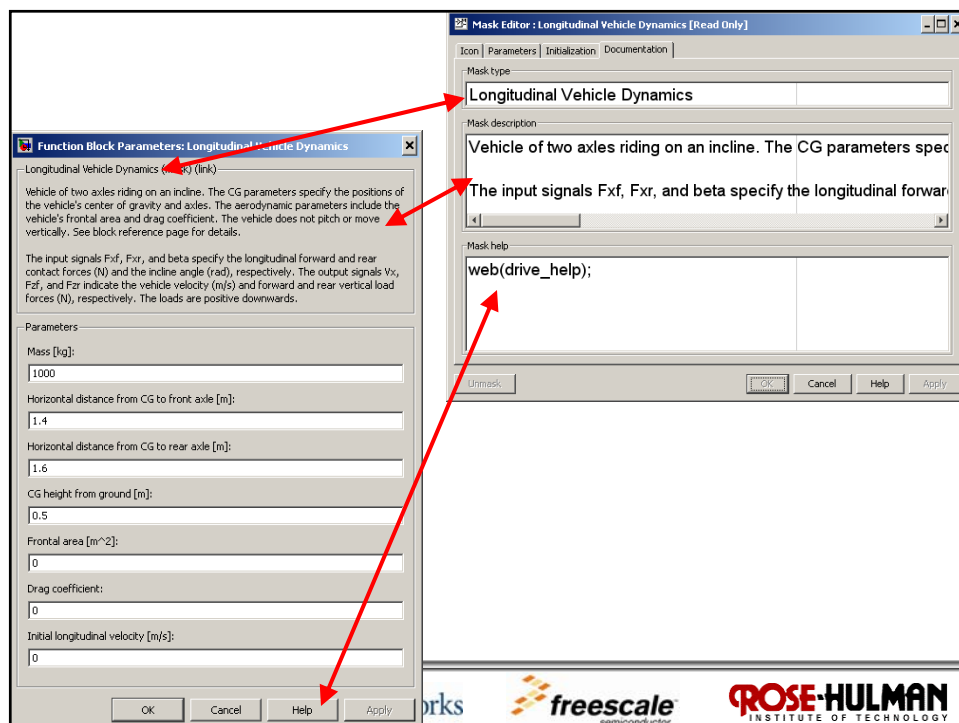
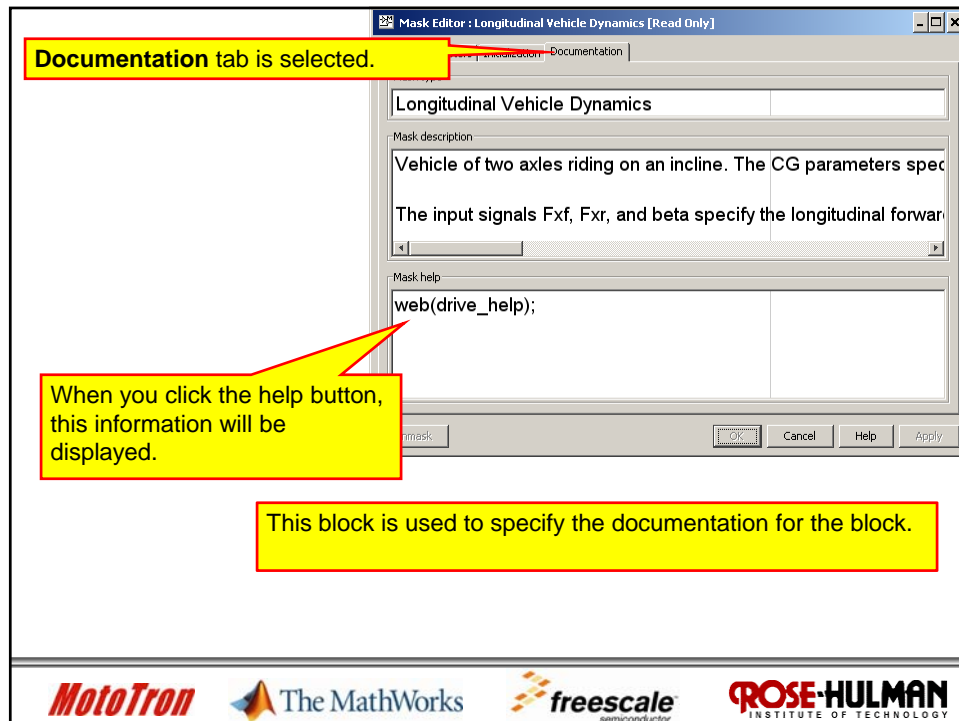
drive_preinitmask(); A = FA;
g = 9.81;
rho = 1.2;

% Vertical load transfer function
nu0 = 2.0; % Hardcoded pitch freq for pseudof
xi = 0.8; % Hardcoded pitch damping for pseudof

omega0 = 2*pi*nu0;
beta = 2*xi*omega0;
    
```

☒ Allow library block to modify its contents

Unmask OK Cancel Help Apply



Longitudinal Vehicle Dynamics

25

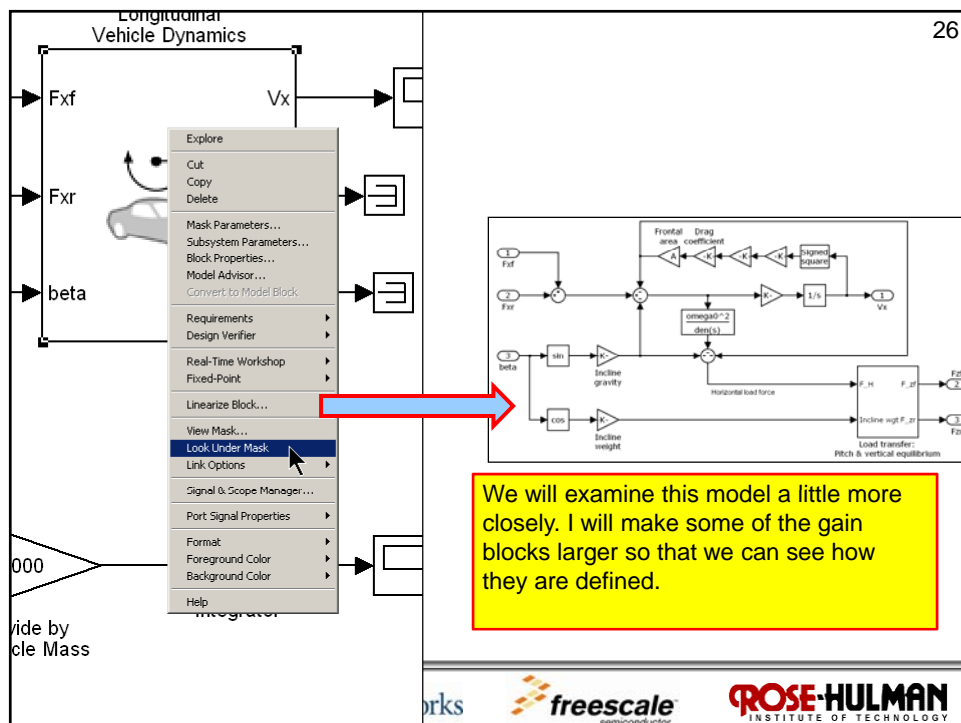
- We will not change the mask, so click the **Cancel** button.
- Next, right-click on the Longitudinal Dynamics Block and select **Look Under Mask** from the menu:
- You will see the underlying Simulink model:

MotoTron

The MathWorks

freescale
semiconductor

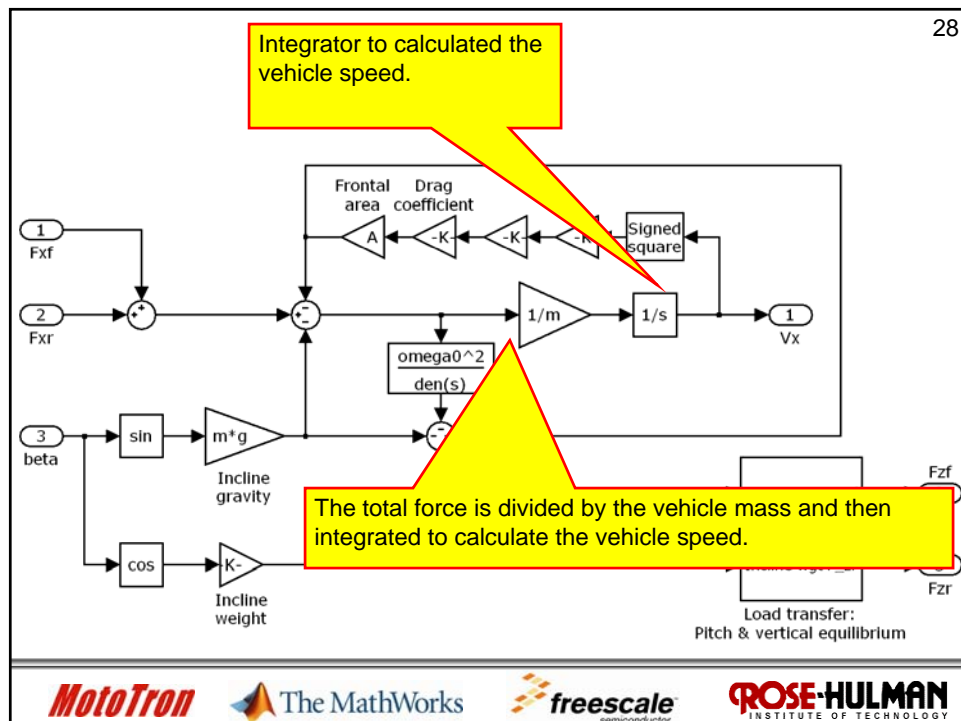
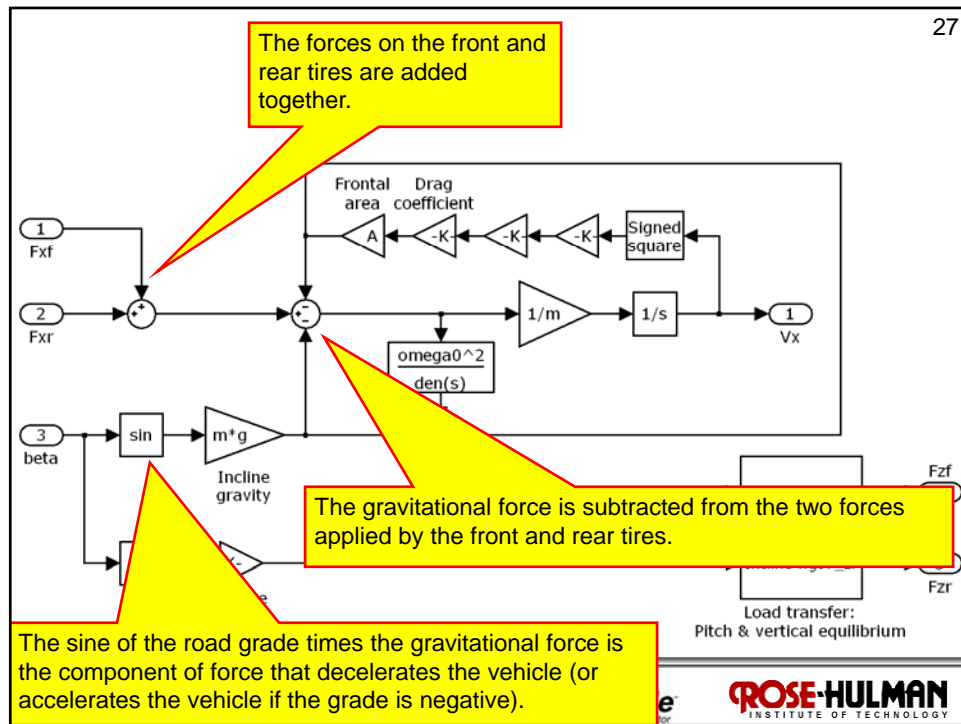
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

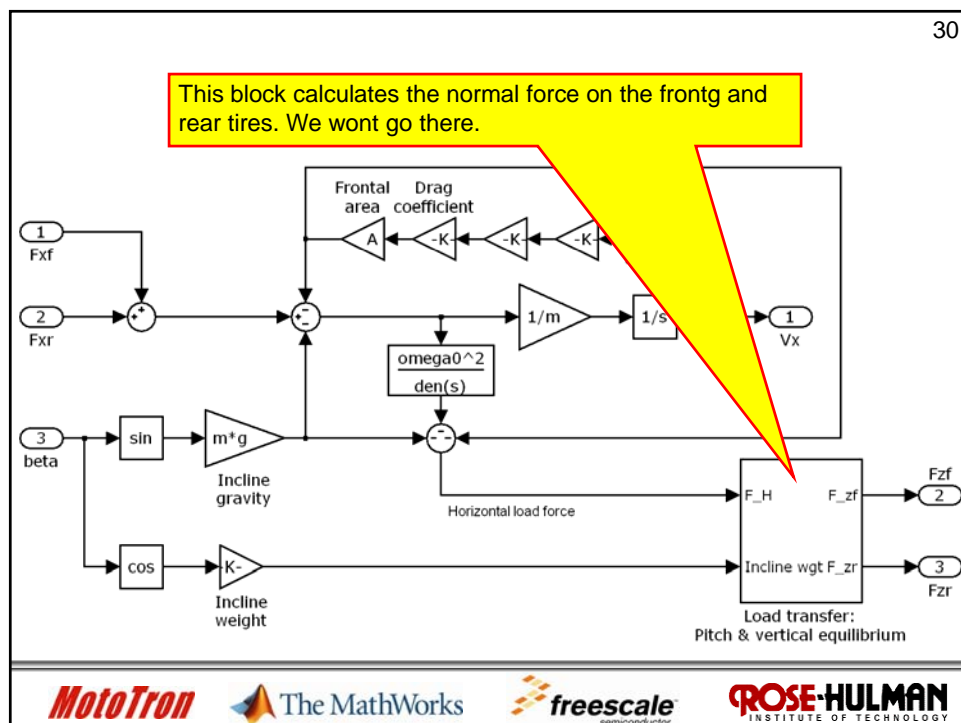
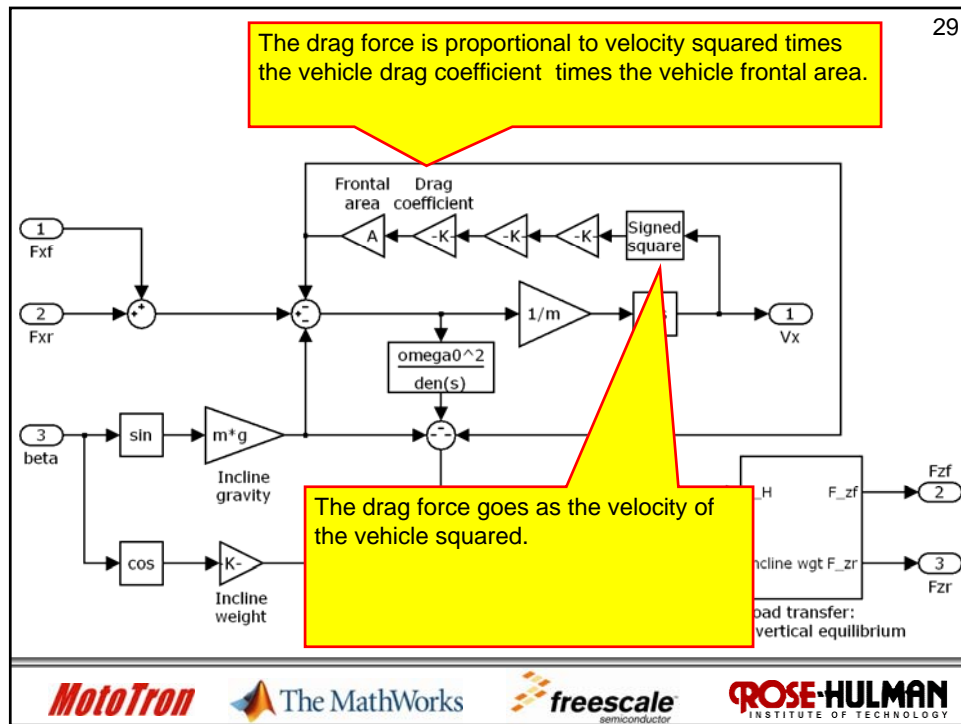


orks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY







Subsystems Blocks and Masks

31

- We kind of understand what is inside the Longitudinal Vehicle Dynamics block.
- We know how to mask a subsystem.
- We know how to look under a mask.
- Many “blocks” in Simulink are actually Simulink subsystems that have been masked.

MotoTron

The MathWorks

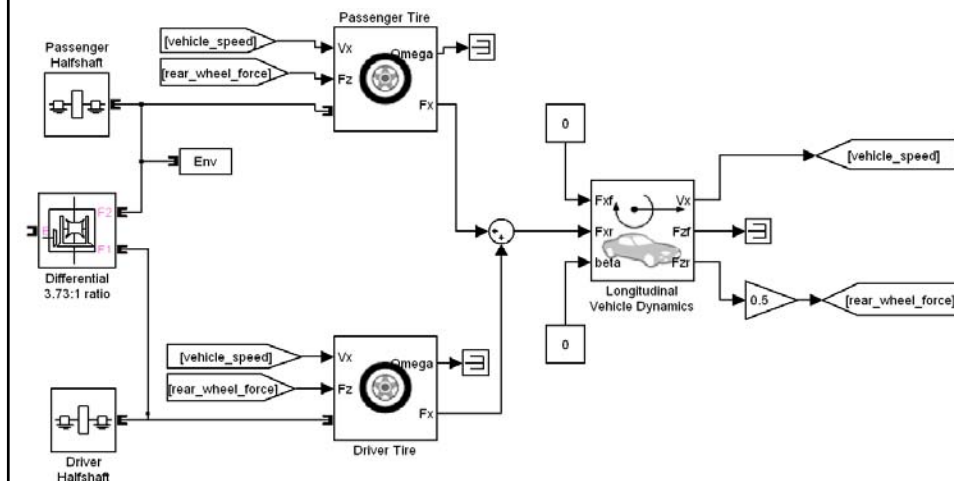
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Drive Train and Solver

32

- We will now create our the rear drive train and vehicle solver for our system.
- We will begin creating the system below:





Initialization File

33

- The model that we will be creating will contain hundreds of blocks.
- Most blocks will have numerical values.
- To give these numerical values meanings, we will define them in a MATLAB scrip file, and add documentation to the script file.
- We will define a number of MATLAB variables in this m-file, use the variables to specify the values of various blocks, and then run the m-file before every simulation.
- Name the file vehicle_Init_File.m



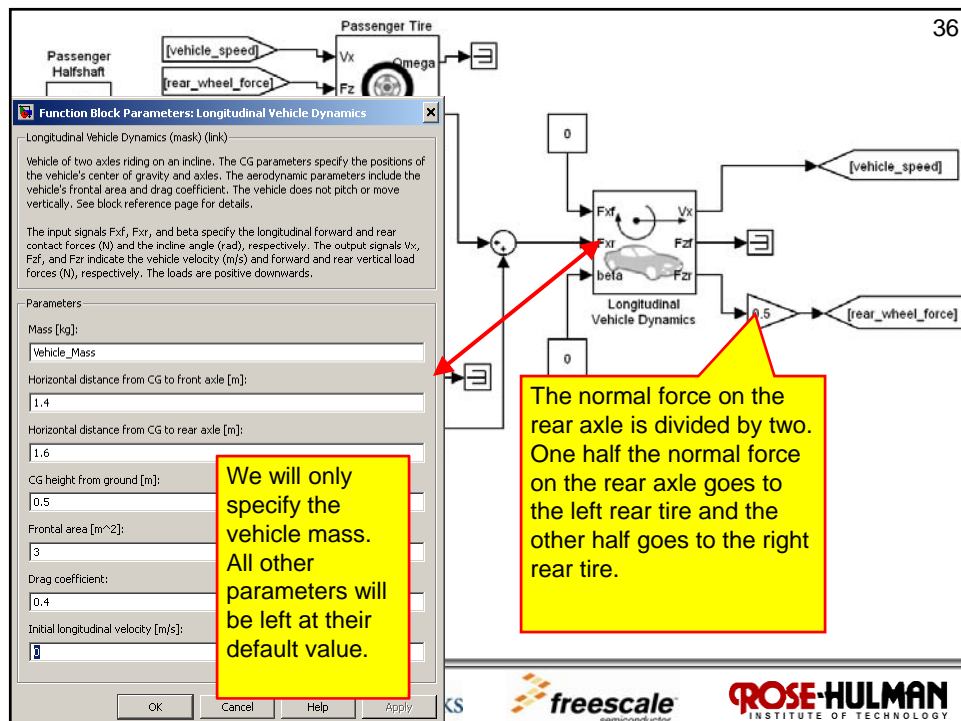
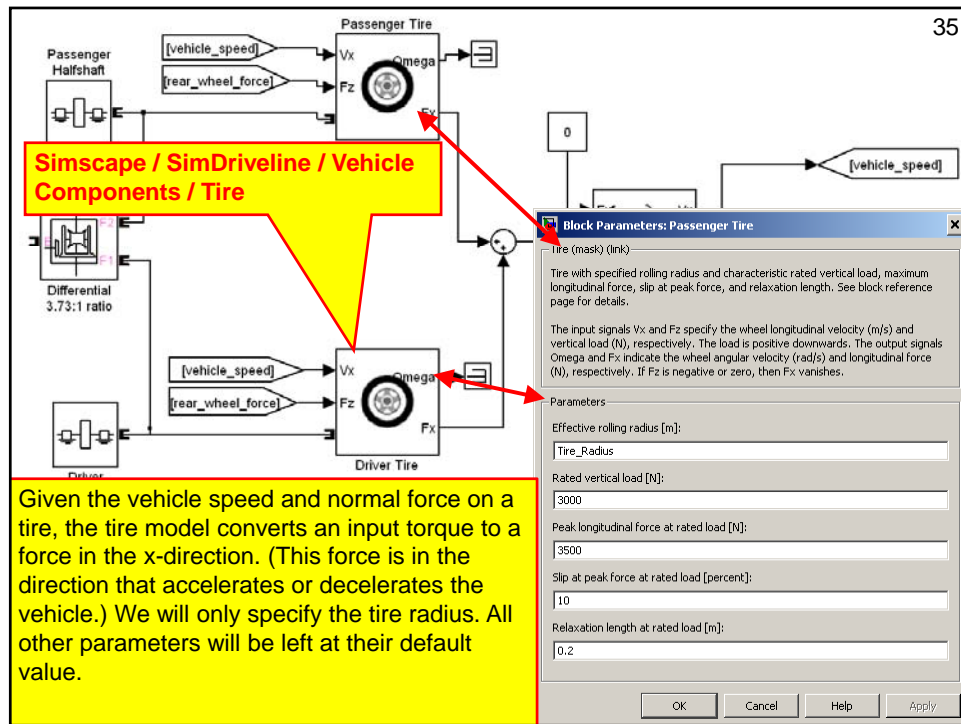
Vehicle Init File

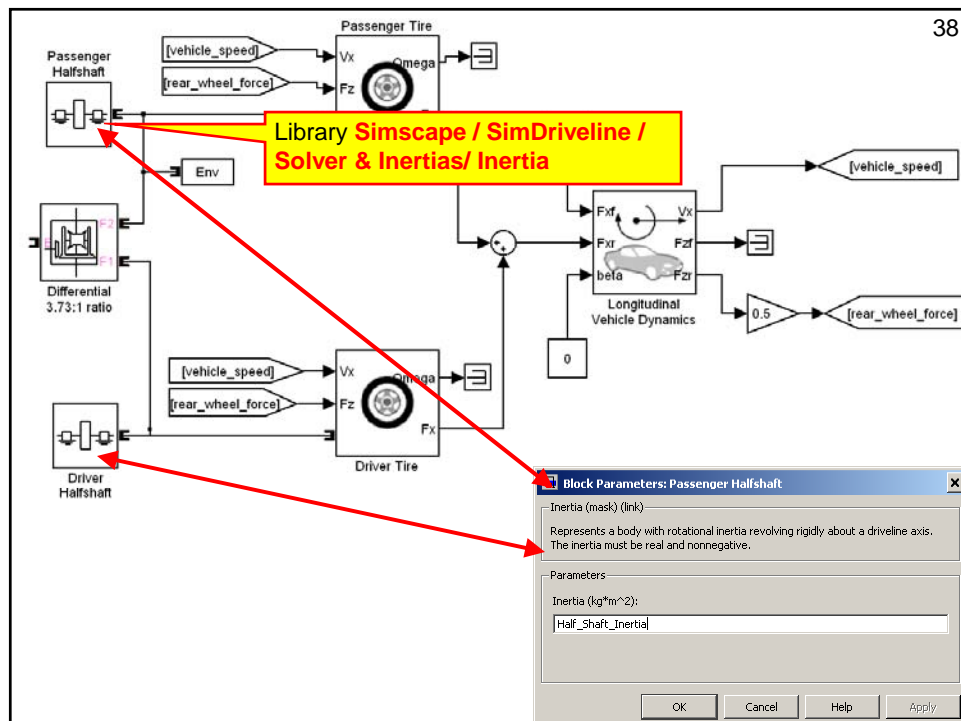
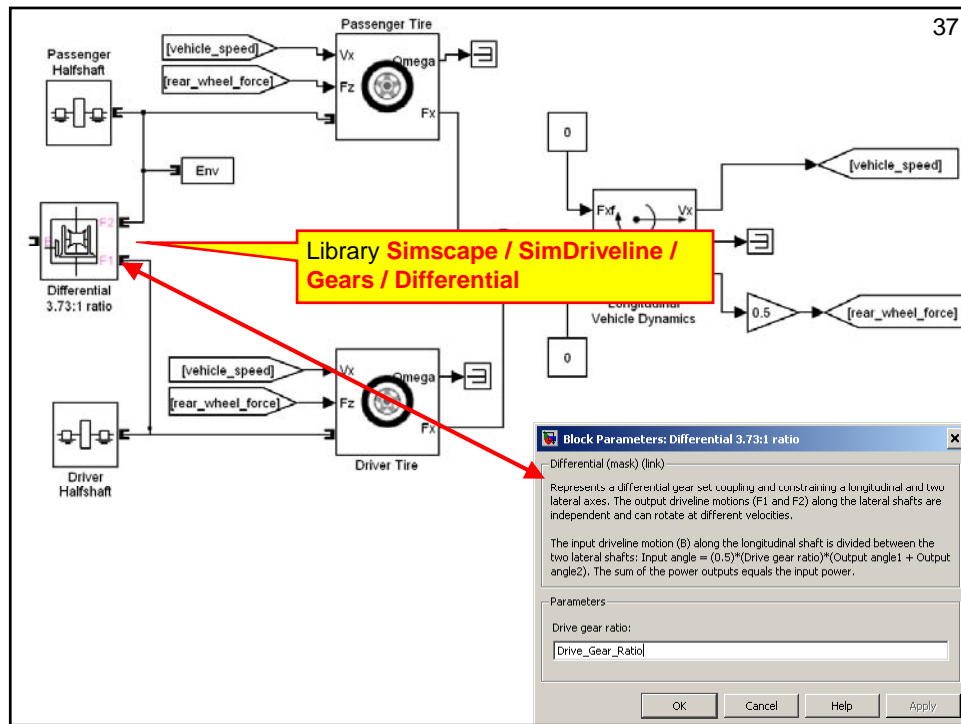
34

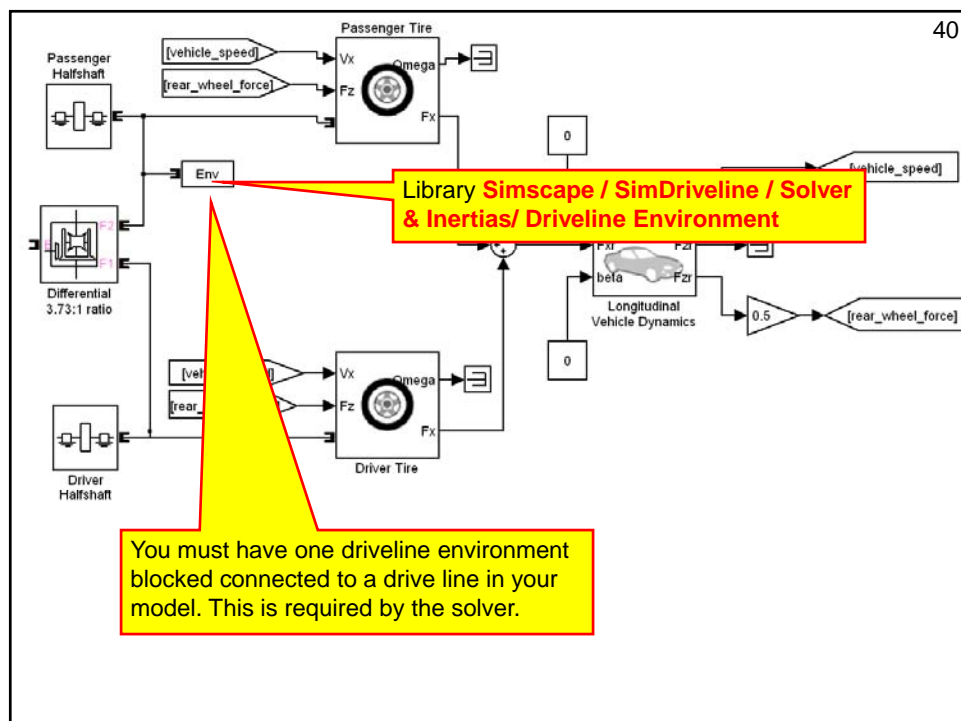
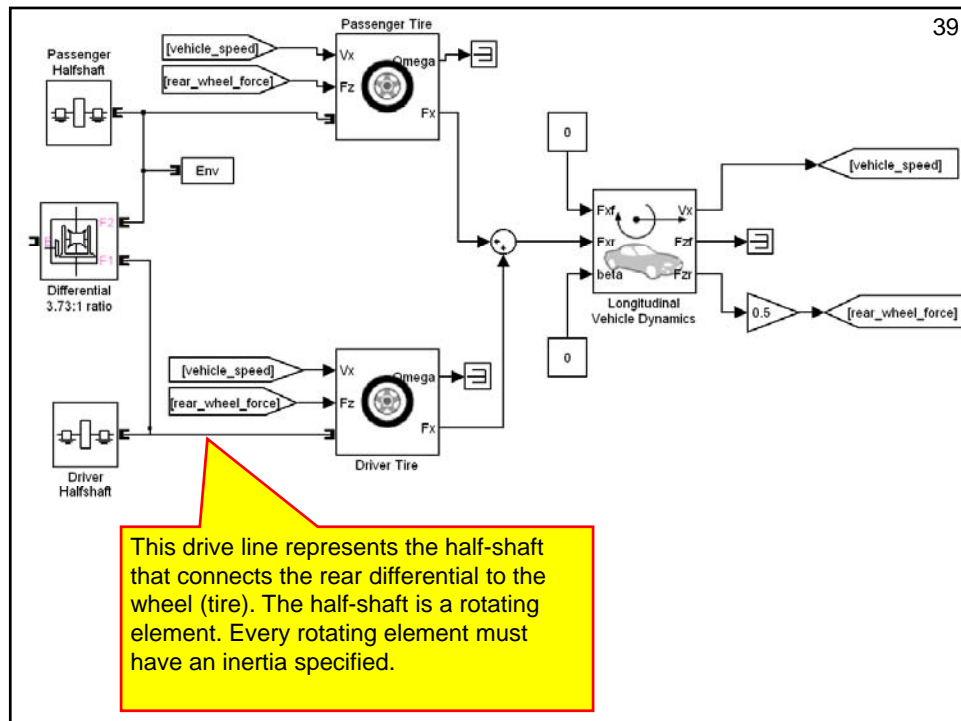
```
1  
2  
3 %Vehicle Constants  
4 Vehicle_Mass = 1633; %in kg (3600 pounds)  
5 Tire_Radius = 0.406; %in meters (16 inches)  
6  
7 % Rear Differential Constants  
8 Drive_Gear_Ratio = 3.73;  
9 Half_Shaft_Inertia = 1.5e-3; %kg*m^2  
10 Drive_Shaft_Inertia = 0.02; %kg*m^2  
11
```

- The numerical values of the blocks are defined in the following few slides:









Powertrain and Vehicle Solver

41

- We have now created a model of a rear-wheel drive vehicle that requires a torque input.
- We will test the system with a “motor” that outputs a constant torque.
- The motor on the next slide was created with a **Constant** and a **Torque Actuator** (library **Simscape / SimDriveline / Sensors and Actuators**.)

MotoTron

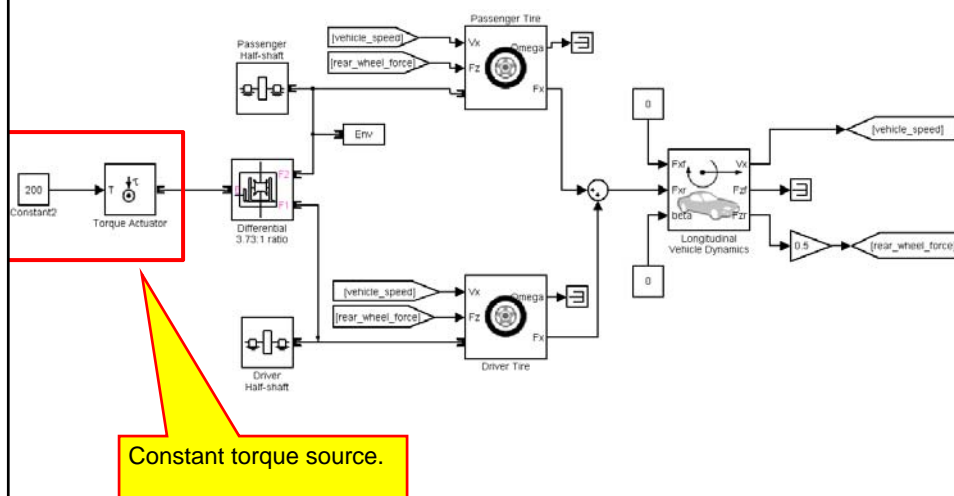
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Constant Torque Source Model

42



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



System Testing

43

- To verify that all of our models work are the same, we will simulate the models for 200 seconds.
- Calculate the vehicle's velocity at 200 seconds and generate a plot of the vehicle's speed versus time.
- Remember to remove the limitation on the number of points a scope can display.
- (Continued on next slide...)

MotoTron

 The MathWorks

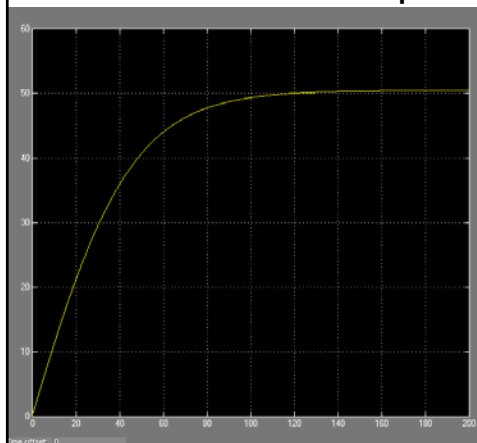
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Lecture 1 Exercise 2

44

- System Testing
- Vehicle Terminal Velocity
- Plot of vehicle speed versus time.



→
Display

Demo



Lecture 1 Exercise 3

45

- Most motors and engines have torque curves.
- Implement a motor that has the following torque curve:
 - From 0 to 2000 rpm, the torque is constant at 200 Nm
 - From 2000 to 7000 rpm, the torque decreases linearly to zero.
- The torque is in Nm and the speed is in rpm.

MotoTron

 The MathWorks

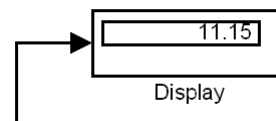
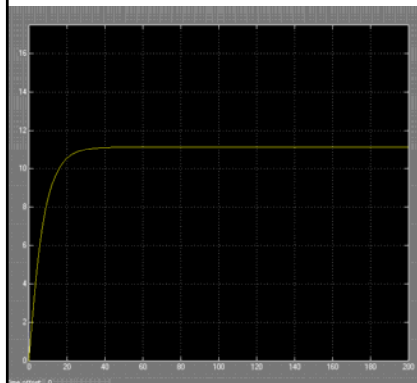
 freescaler
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Lecture 1 Exercise 3

46

- You will need to sense the “motor” speed and convert it to rpm.
- You can do this using a 1-D lookup table.
- Plot the vehicle velocity and determine the vehicles velocity after 200 seconds.



Demo _____



Model Hierarchy Battery Model

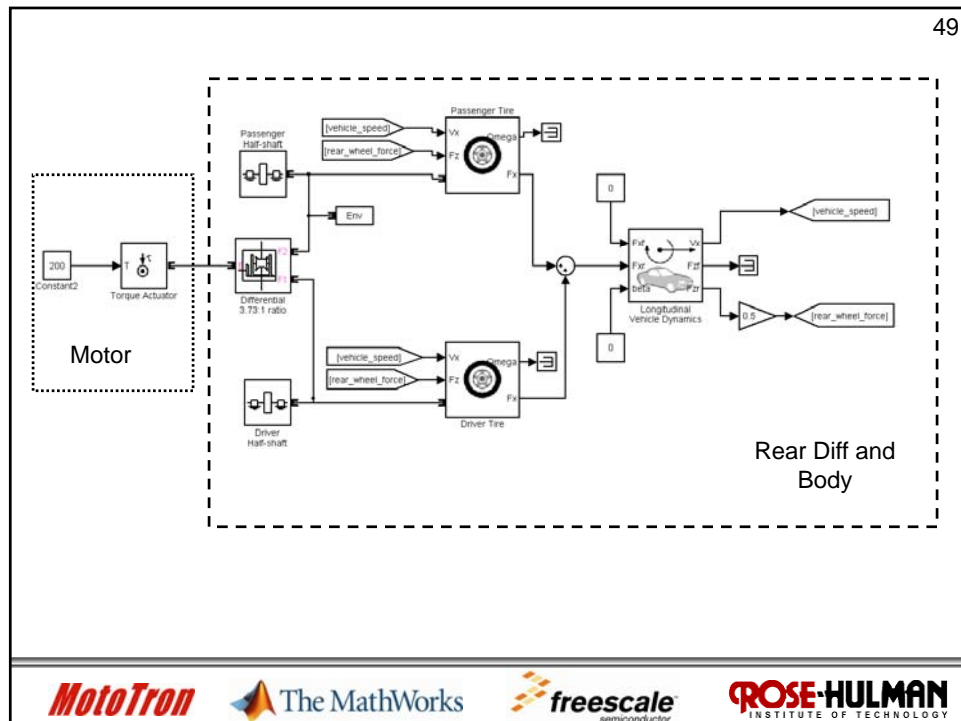


Model Hierarchy

48

- Although the model we have developed is still fairly simple, as we add models for the battery, engine, and motors, the model will become quite large and cumbersome.
- We will break the model into subsystems that represent specific vehicle components.
- Our present model has blocks that represent the motor, the rear differential, and body.

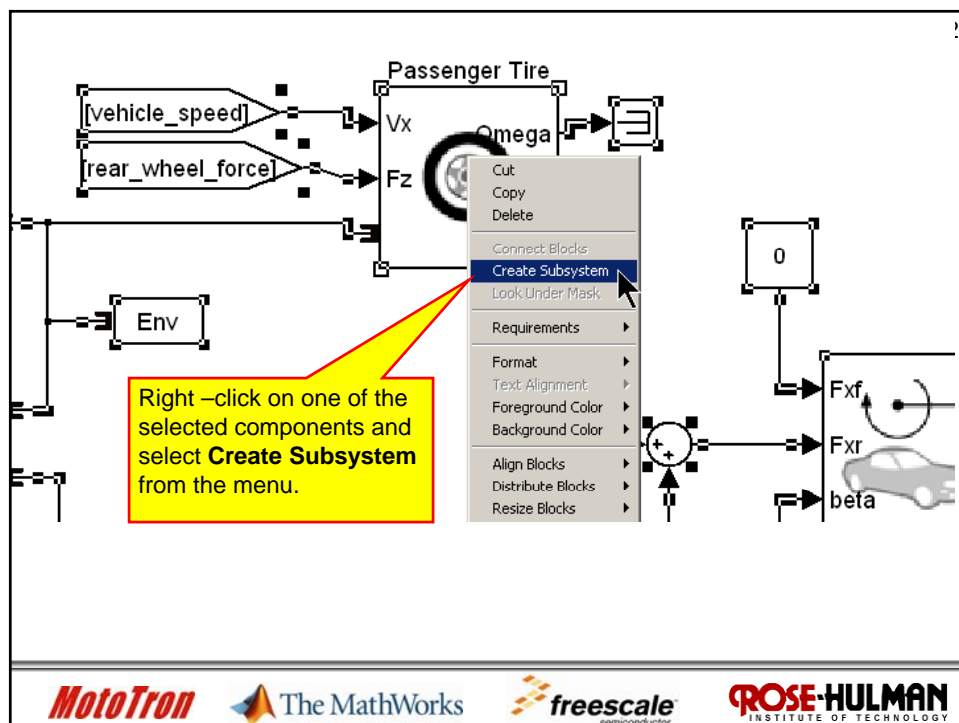
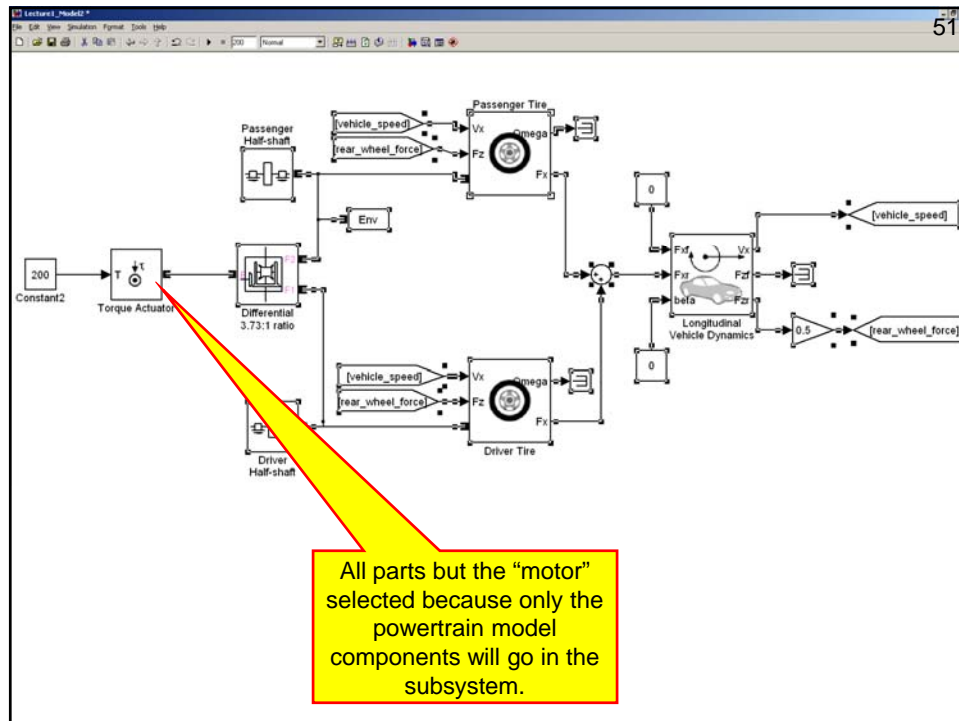




Model Hierarchy

50

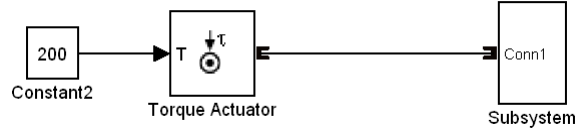
- We will make a subsystem out of the powertrain.
- Select all of the components you wish to place in the subsystem by dragging a selection box around the components.
- Right-click on the selected components and select the **Create Subsystem** menu selection.



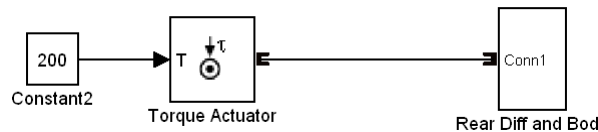


53

After creating the subsystem we have the top level block diagram as shown:



Click on the text “Subsystem” and change the text to “Rear Diff and Body”



Double-click on the Rear Diff and Body subsystem block to open it:

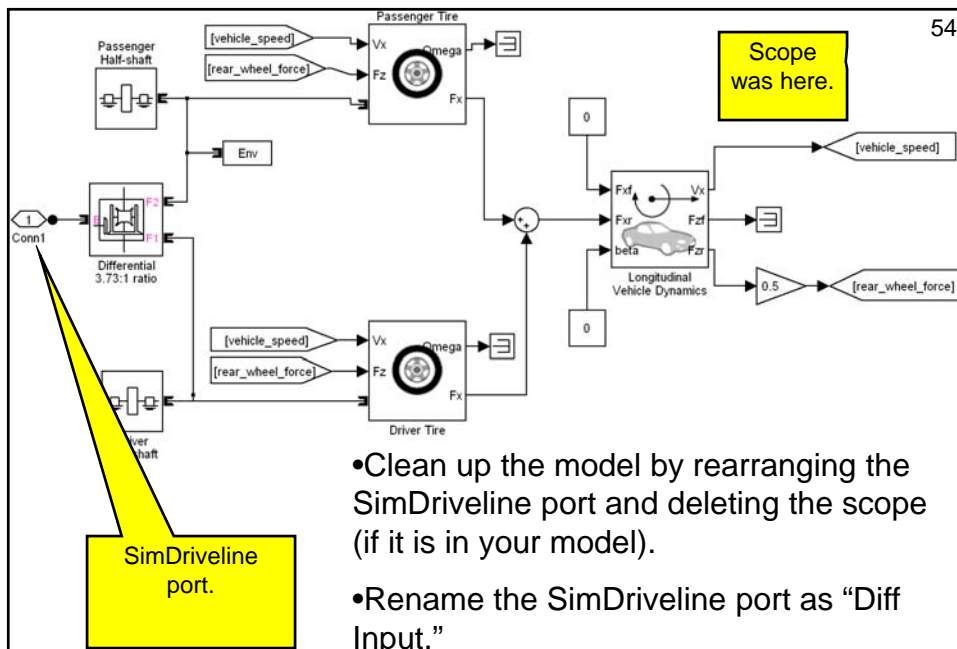
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

54



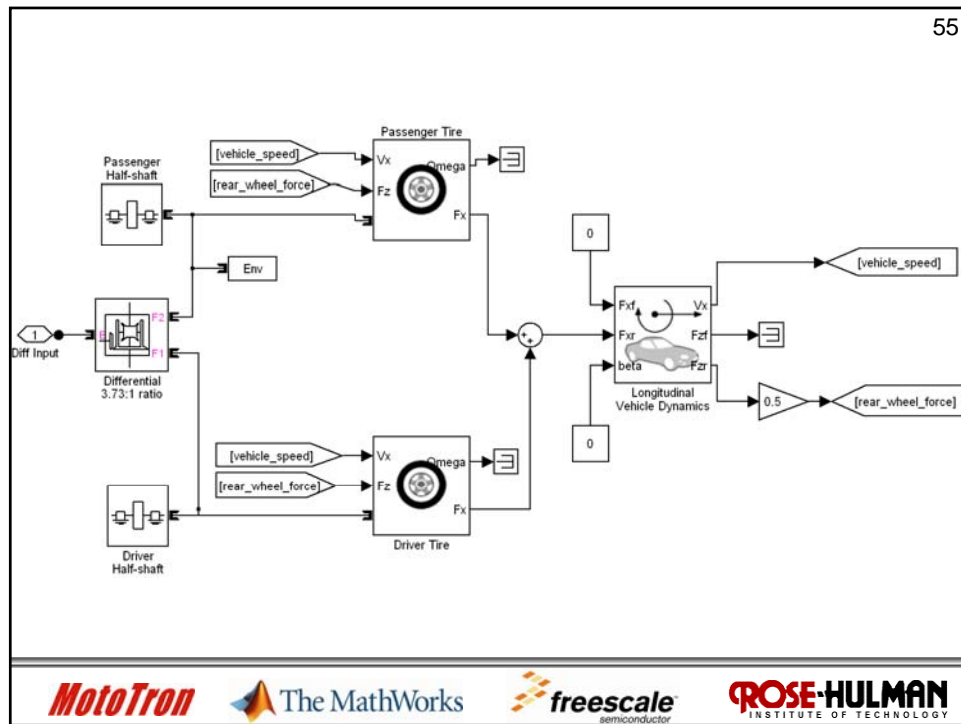
- Clean up the model by rearranging the SimDriveline port and deleting the scope (if it is in your model).
- Rename the SimDriveline port as “Diff Input.”

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Rear Diff and Body Subsystem

- This block models the physics in which we are interested.
- We would like to create a status bus that contains the vehicle speed, and tire speeds.
- We need to convert the tire speed from r/s to mph.
- We need to convert the vehicle speed from m/s to mph.



Conversions

57

$$Speed(mph) = speed(m/s) \times \left(\frac{3600 \text{ sec}}{1 \text{ Hour}} \right) \times \left(\frac{1 \text{ mile}}{1609 \text{ meters}} \right)$$

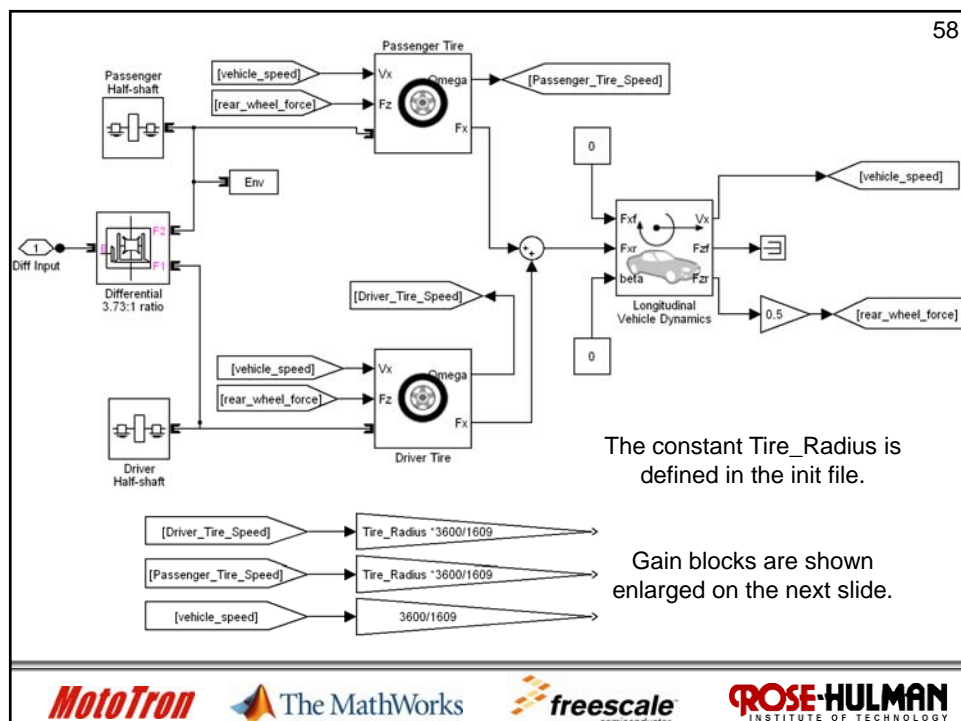
Linear Speed = angular speed (rad/sec)
times Tire Radius

MotoTron

The MathWorks

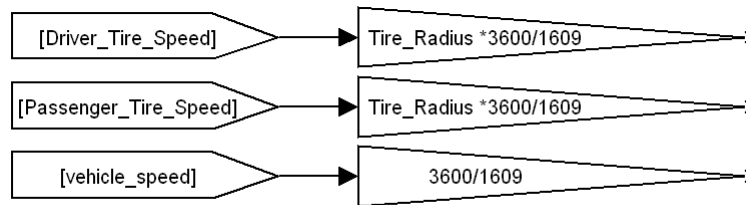
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





59



MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

60

Rear Diff and Body Diagnostics

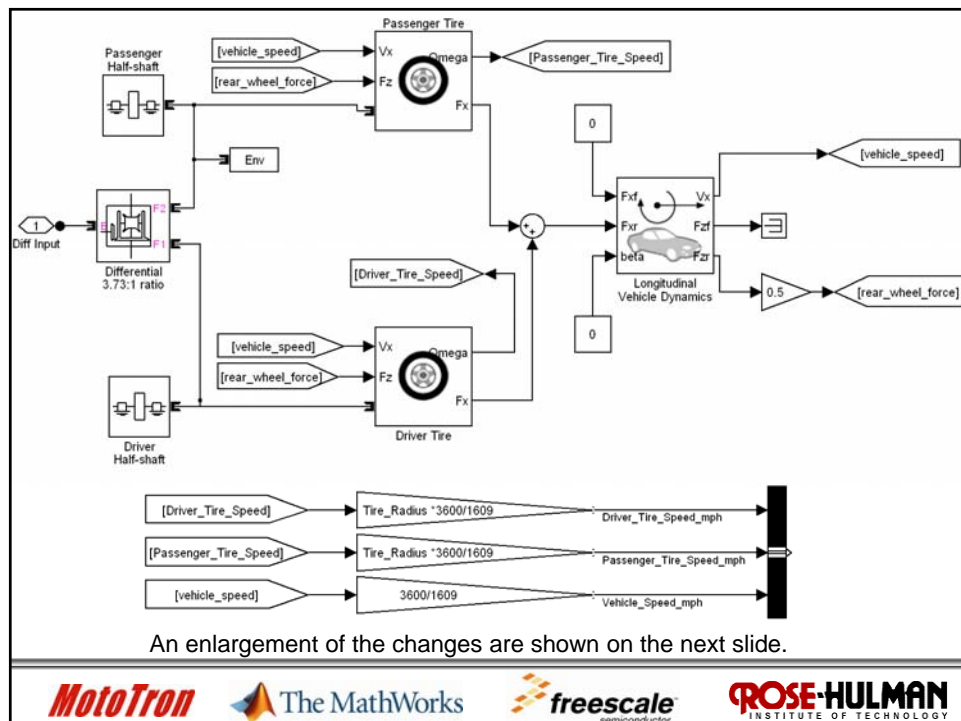
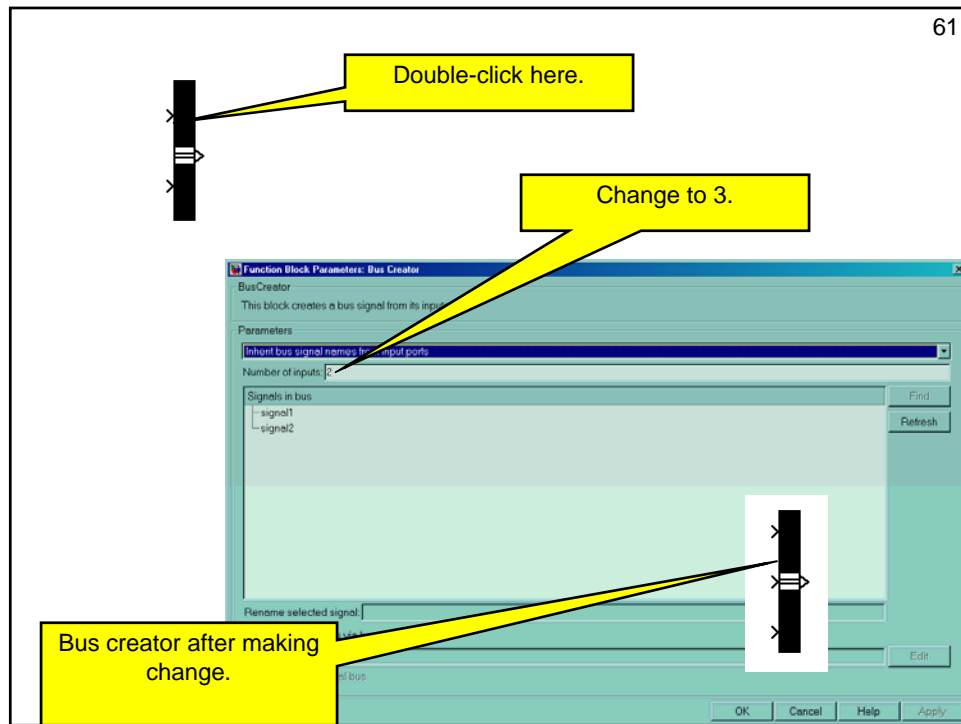
- The last thing we need to do is create the diagnostics bus for this block.
- Use the bus creator part
(**Simulink/Commonly Used Blocks**)
- Add the following signals
 - Passenger_Tire_Speed_mph
 - Driver_Tire_Speed_mph
 - Vehicle_Speed_mph

MotoTron

 The MathWorks

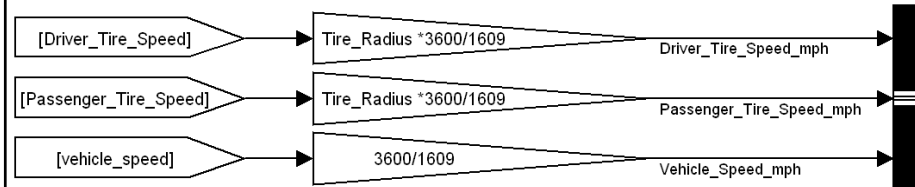
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





63



MotoTron


The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

64

Diagnostic Bus

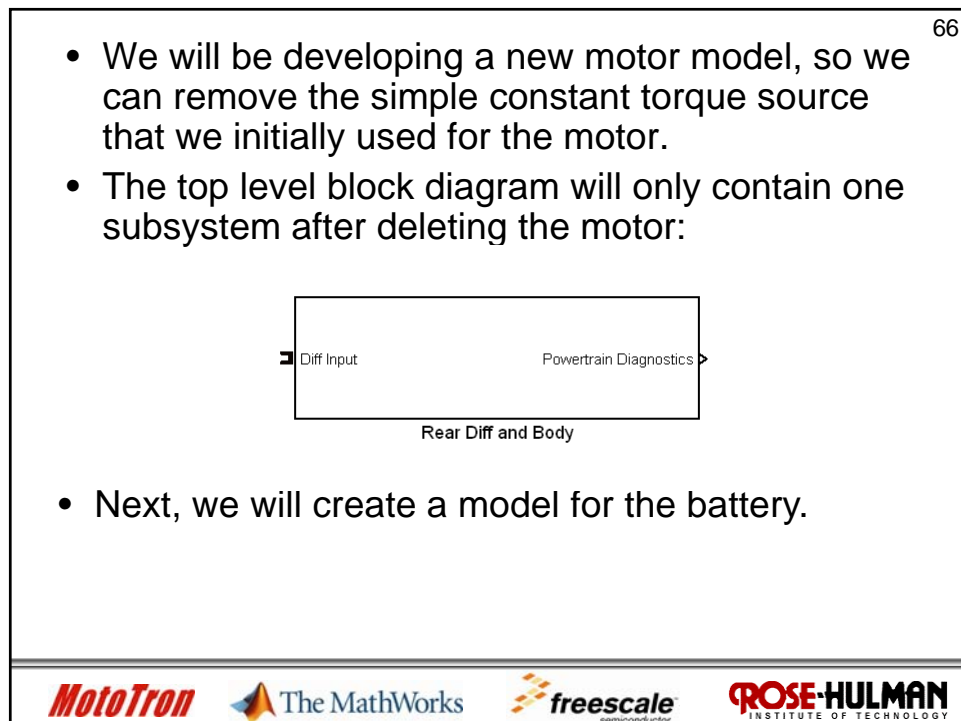
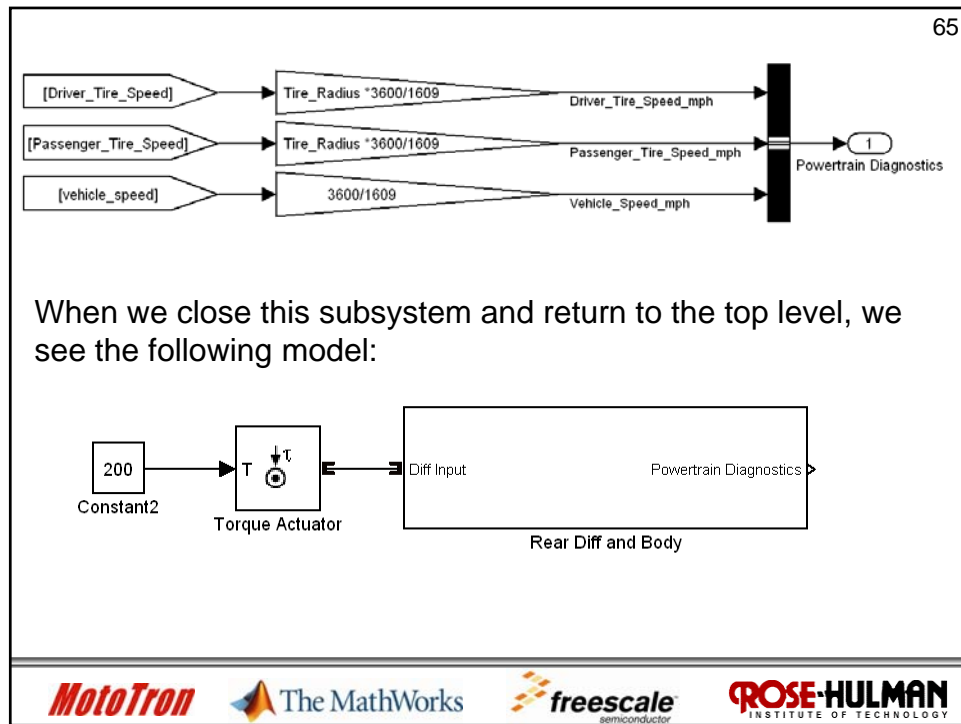
- We need to make the information available on this diagnostic bus available outside this block.
- Add an “Out1” port  (Simulink/Commonly Used Blocks) to the diagram and label it as “Powertrain Diagnostics.”

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Battery Model

67

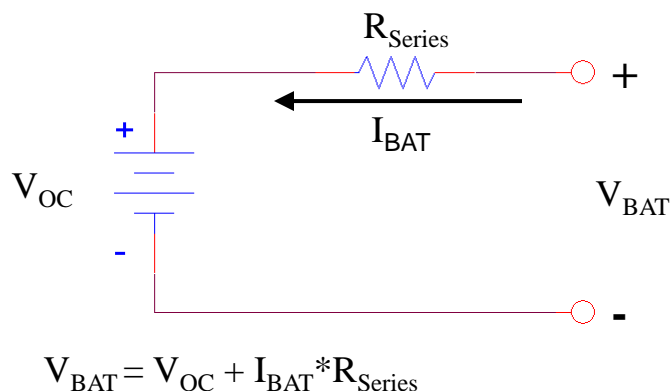
- We will now create a battery model that calculates the battery terminal voltage and battery state of charge (SOC).
- The inputs to the model are the two motor currents that we will have in the vehicle. (You can also add an input for the vehicle hotel loads.)
- The outputs of this block are the battery voltage and a diagnostic bus that contains battery signals of interest.



Battery Terminal Voltage

68

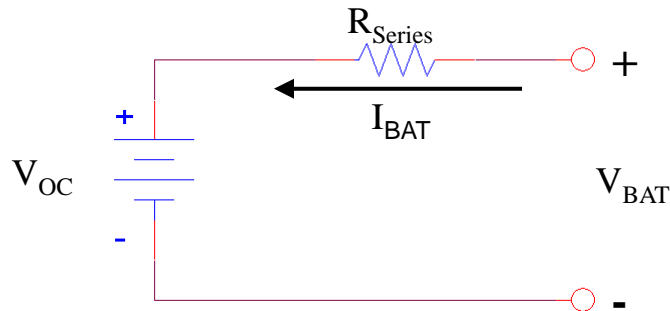
- We will use the first order model below:





Battery Terminal Voltage

69



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

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Battery Model

70

- 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
 - Making R_{Series} a function of the SOC and temperature.
 - Having a different charge and discharge series resistances.

MotoTron

The MathWorks

freescale
semiconductor

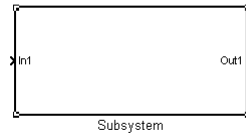
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Battery Model

71

- We will now create a subsystem and implement the equation for the battery voltage.
- Place a Subsystem block in the top level of your model (**Simulink\Commonly Used Blocks**)



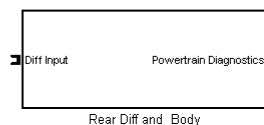
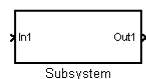
MotoTron

 The MathWorks

 **freescaler**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

72



Highlight the text “Subsystem” and change it to “Battery.” This will name the subsystem.

MotoTron

 The MathWorks

 **freescaler**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



73

Double-click here.

Subsystem name changed.

Next, double-click on the subsystem to open it.

MotoTron **The MathWorks** **freescale** **ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY

74

1
In1

1
Out1

- This subsystem has a single input and a single output.
- The output is just equal to the input.
- Delete the connection between the input and output. (Click on the wire and press the delete key.)
- Duplicate In1 by
 - Holding down the control key and then dragging In1 to a new location.
 - Right-click on In1 and drag it to a new location.

MotoTron **The MathWorks** **freescale** **ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY

75

1
In1

1
Out1

2
In2

- In1 and In2 will be the motor and generator currents. The total battery current will be the sum of these two inputs.
- Click on the text In1 and change it to Motor_Current_A.
- Click on the text In2 and change it to Generator_Current_A.

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

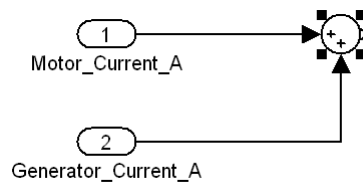
76

1
Motor_Current_A

1
Out1

2
Generator_Current_A

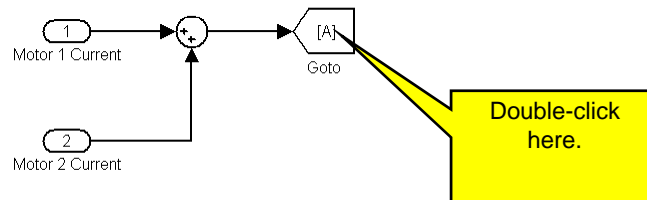
- Next, we would like to form the battery current as the sum of the motor and generator currents.
- Place the sum part (**Simulink/Commonly Used Blocks**) in your model and connect the two inputs as shown:





77

- We would like to use the battery current in a few places so we will add a “Goto” part to our model. This part is located in the **Simulink/Signal Routing** library.



- Double-click on the Goto part to change the label.

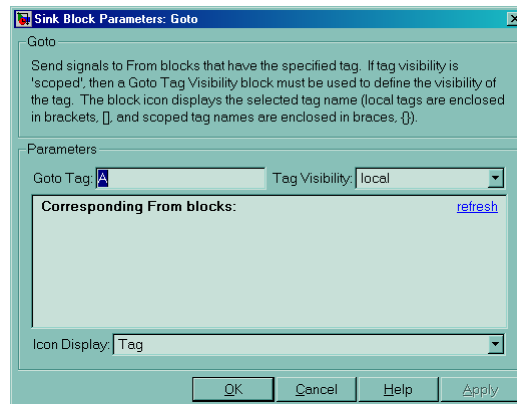
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

78



- Since the text “A” is highlighted, we can just type in a new tag for the Goto part.
- Enter the text, “Battery_Current_A” and click the **OK** button.
- You may need to change the size of the Goto part to see the label.

MotoTron

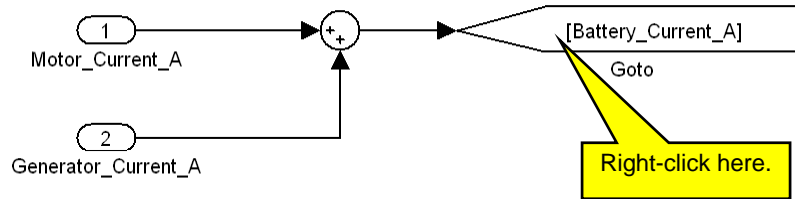
The MathWorks

freescale
semiconductor

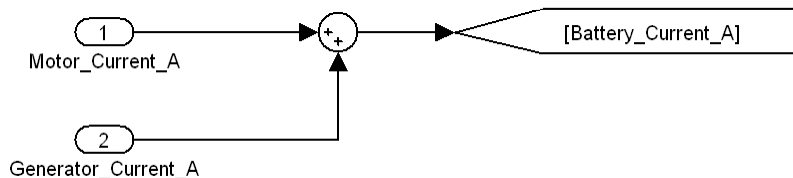
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



79



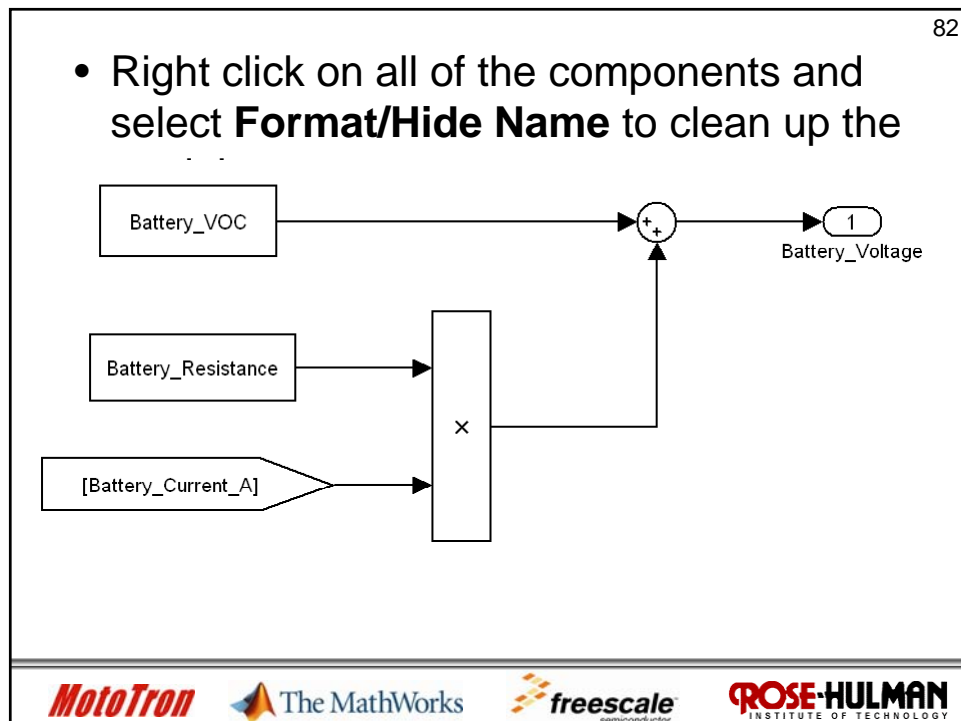
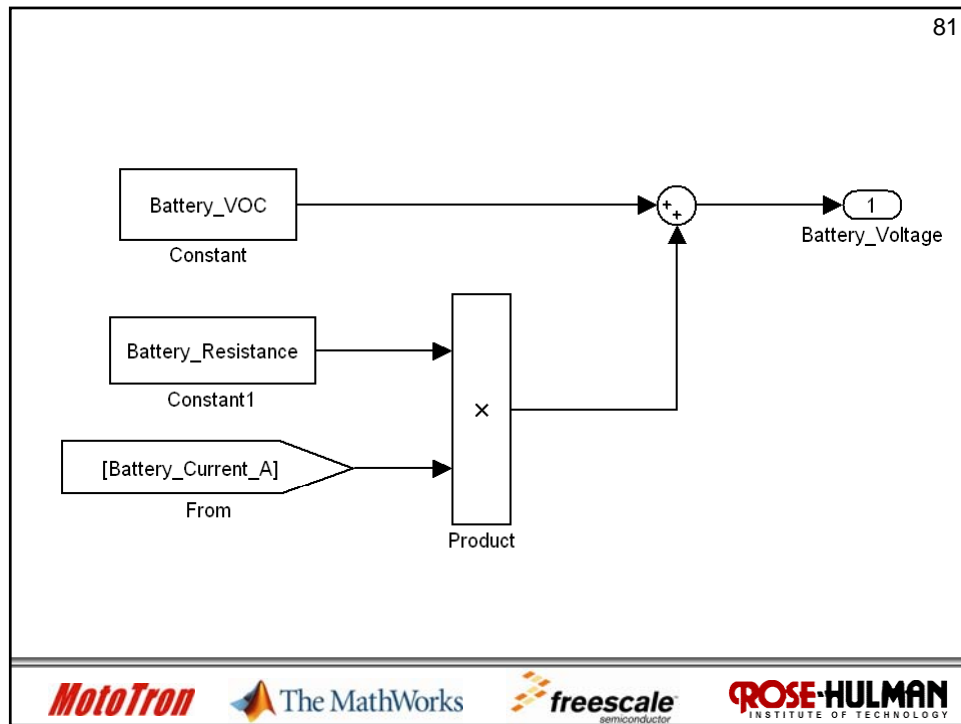
Right-click on the Battery_Current_A Goto part and select **Format/Hide Name** from the menus to hide the text "Goto."



80

Battery Model

- Next, we will calculate the battery terminal voltage as $V_{BAT} = V_{OC} + I_{BAT} * R_{Series}$
- V_{OC} and R_{Series} are constants defined with the init file and read from the workspace.
- Create the model shown next.
- Use Parts:
 - Constant (**Simulink/Commonly Used Blocks**)
 - Sum (**Simulink/Commonly Used Blocks**)
 - Product (**Simulink/Commonly Used Blocks**)
 - From (**Simulink/Signal Routing**)



83

- 



ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Battery State of Charge

85

- The unit of an Amp-Hour is an amount of charge in coulombs.
- 1 Amp = 1 coulomb / 1 second.
- 1 Hour = 3600 seconds.
- 1 Amp-Hour = 1 amp * 1 Hour = 3600 coulombs.



Battery State of Charge (SOC)

86

- The battery amp-hour rating is a measure of how much charge the battery stores.
- The battery SOC is a measure in percent (0% to 100%, or 0 to 1) of how much charge is stored in the battery relative to the full AH rating.





Battery SOC


87

- To calculate the SOC, we need to know the initial SOC and then calculate how much charge has been added or removed from the battery.
- The charge added or removed is calculated by integrating the battery current.
- We then divide the battery charge by the amp-hour rating of the battery to obtain the SOC.



Battery SOC

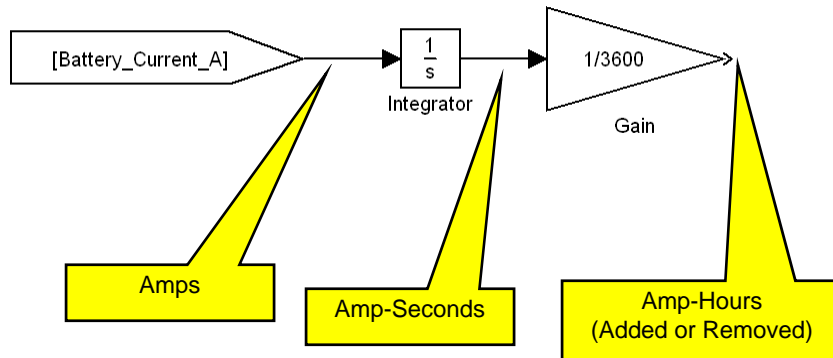
88

- The initial SOC is a constant defined in the vehicle model init file, and read from the workspace.
- Use an integrator  to integrate the battery current. (**Simulink/Commonly Used Blocks**)
- Scale the integrated current by 3600 to convert charge to amp-hours. (Use the gain block. **Simulink/Commonly Used Blocks**)





89



- Next, divide the Amp-Hours added or removed by the battery Amp-Hour Rating to calculate the SOC added or removed.
- Use the **Divide** part (**Simulink/Math Operations**).
- The battery Amp-Hour rating is a constant defined in the init file.

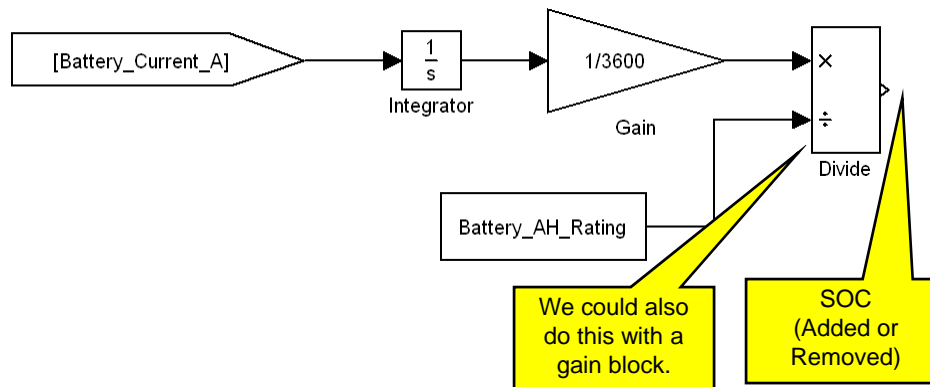
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

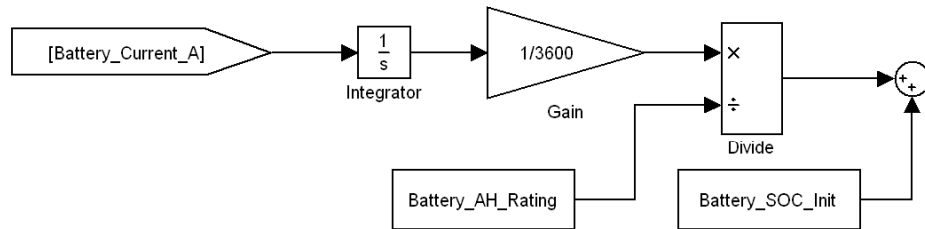
90



- Next, we add the SOC added or removed to the battery initial SOC to calculate the battery's current SOC.
- Use the Sum part (**Simulink/Commonly Used Blocks**).
- The battery initial SOC is a constant defined in the init file.

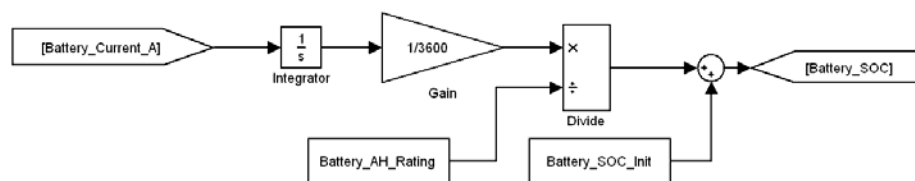


91

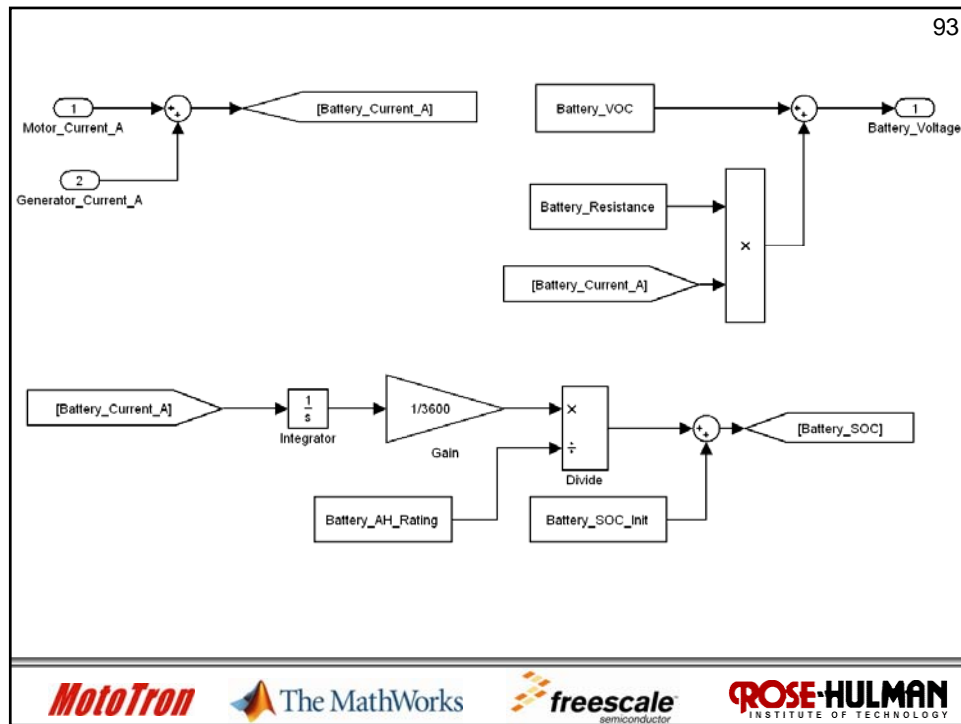


- The battery SOC will not be used by other components in the model but it will be used by our supervisory controller.
- We do not need an output port for this parameter.
- We will add a Goto port to this parameter.
- Later, we will add this signal to our status bus.

92



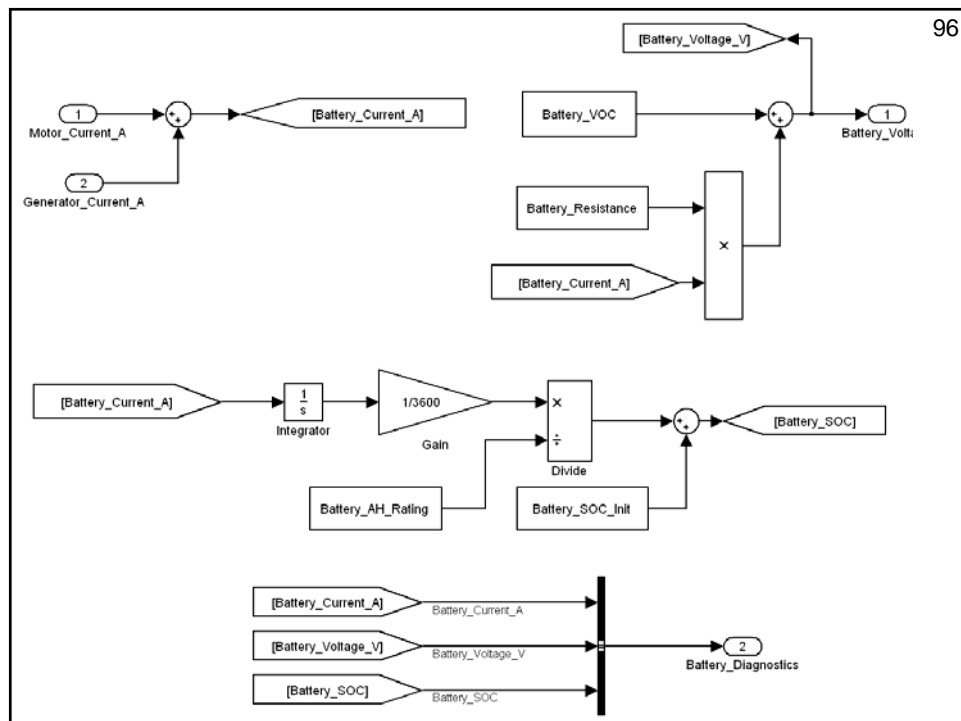
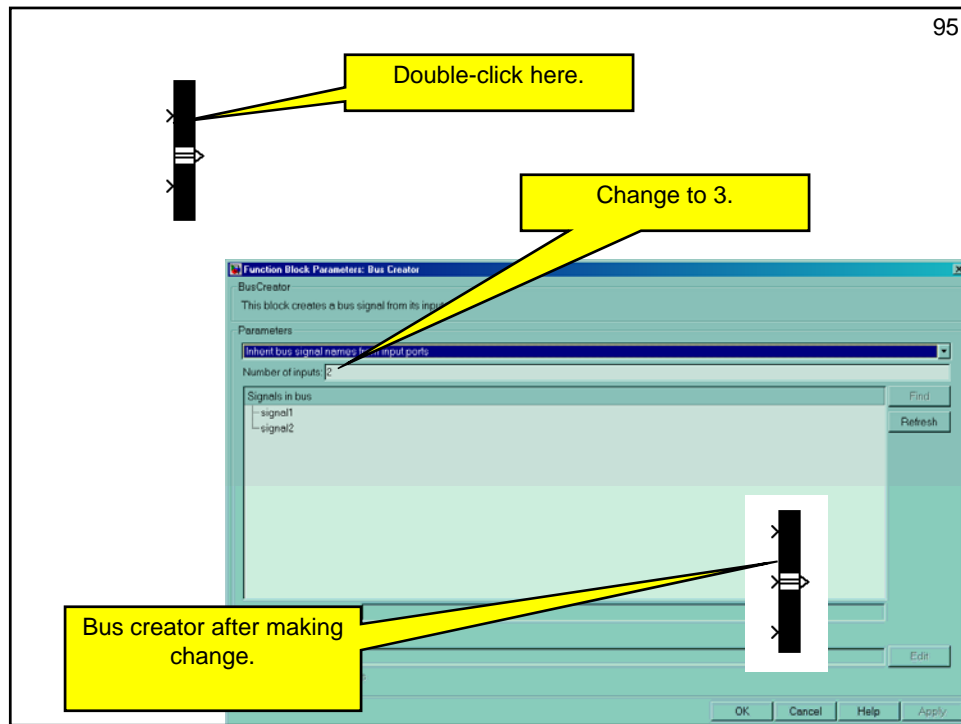
- We will clean up the model slightly.
- The entire model is shown on the next slide.



Battery Diagnostics

94

- The last thing we need to do is create the battery diagnostics bus.
- Use the bus creator part (Simulink/Commonly Used Blocks)
- Add the following signals
 - Battery Voltage
 - Battery Current
 - Battery SOC



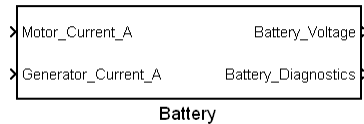
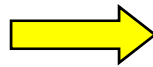


Initialization File Changes

97

```
13 %Battery Constants
14 Battery_VOC = 365; %Open circuit voltage in Volts.
15 Battery_Resistance = 0.4; %Charge and discharge resistance approximation in Ohms.
16 Battery_AH_Rating = 8; %Battery name plate Amp-Hour rating in AH.
17 Battery_SOC_Init = 0.7; %Battery initial state of charge.
18
```

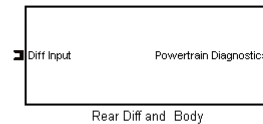
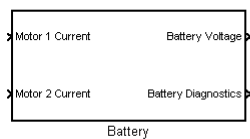
When we go to the top level block diagram, the battery subsystem should show the inputs and outputs that we defined.



Top Level Block Diagram

98

- The top level block diagram now contains two blocks.





Lecture 1 Exercise 4

99

- Show your completed Battery and Rear Diff and Body subsystems.
- Run a simulation that shows that your Battery:
 - Produces the correct output voltage for a given input current.
 - Calculates the correct battery state of charge for a given input current.

Demo_____





Advanced Model-Based-System Design

Lecture 2: Motor Model and Display Subsystem



Electric Motor Model

2

- Create an ideal motor that converts electrical power to mechanical power with 100 percent efficiency.
- This model will work for both regen and motoring modes. The conversion equation is:

$$-I_{Motor} V_{Battery} = \tau_{Motor} \omega_{Motor}$$





Electric Motor Model

3

- The battery voltage is always positive.
- When the motor torque is in the same direction as the motor shaft velocity, the motor accelerates the vehicle. (Motoring mode.)
- In this mode, the motor will draw current from the battery. → The motor current should be negative to discharge the battery.

$$-I_{Motor} V_{Battery} = \tau_{Motor} \omega_{Motor}$$



Electric Motor Model

4

- When the motor torque is in the opposite direction of the motor shaft velocity, the motor is decelerating the vehicle. (Regenerative braking mode).
- In this mode, the motor will force current into the battery. → The motor current should be positive to charge the battery.

$$-I_{Motor} V_{Battery} = \tau_{Motor} \omega_{Motor}$$





Electric Motor Model

5

- This motor model has a flat torque curve.
- The motor has the same available torque at any rpm.
- The motor is 100% efficient.
- This is a simple model to get started. We can always make the model more complicated as our understanding of the system increases.



Electric Motor Model

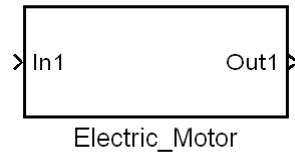
6

- Model Inputs (Simulink):
 - Battery Voltage
 - Torque Request (-1 to 1);
- Model Outputs (Simulink Signals):
 - Motor Current
 - Motor Diagnostics
 - Motor rpm
 - Motor Torque
 - Motor Current
- Model Outputs (SimDriveline):
 - Motor Torque

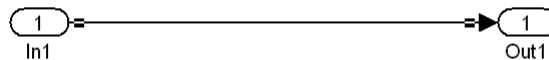


7

Place a new subsystem part (**Simulink/Commonly Used Blocks**) in your circuit and change the name to “Electric_Motor.”



Double-click on the subsystem block to open it:



8

Motor Model

- We have two Simulink inputs and two Simulink outputs.
 - Delete the line between the input and output ports.
 - Duplicate the input and output ports
 - Rename the ports:
 - Battery_Voltage
 - Torque_Request
 - Motor_Current
 - Motor_Diagnostics

9

1

Battery_Voltage

1

Motor_Current

2

Torque_Request

2





Motor_Diagnostics

We also have one SimDriveline connection port (SimDriveline/Utilities).

1

Connection Port

Place this part in your model and rename it as "Motor_Port."

10

1

Battery_Voltage

1

Motor_Current

2

Torque_Request





2

Motor_Diagnostics

1

Motor_Port

We now have all of the input and output ports for our motor model. All that is left is for us to build the actual model.

11

- We will assume that the motor output torque can range from 0 to the maximum output torque.
- The maximum torque is a constant defined in the init file.
- Remember that the torque request signal is a number from -1 to 1.
- The **Torque Actuator** part is located in the **SimDriveline/Sensors & Actuators** library.
- The inertia part is located in the **SimDriveline/Solver & Inertias** library.

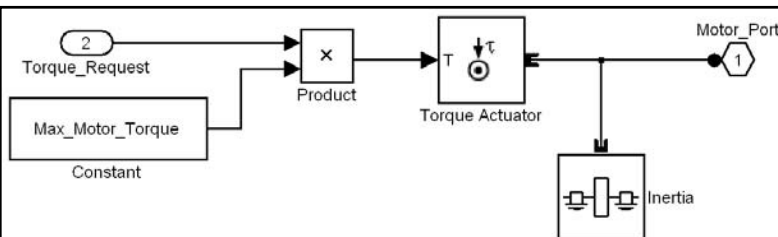
MotoTron

 **The MathWorks**

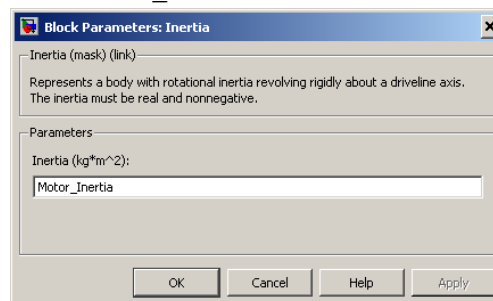
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

12



The inertia part specifies the inertia for all rotating parts of the motor. We will define this inertia in the init file. Double-click on the inertia part and change the value to “Motor_Inertia.”



MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

13

We should find the inertia from the motor specifications or measure the inertia. Once a value is obtained, we will specify it in the init file:

```

33 %Motor Model Constants
34 - Max_Motor_Torque = 240; % Nm
35 - Motor_Inertia = 0.1; % kg*m^2
    
```

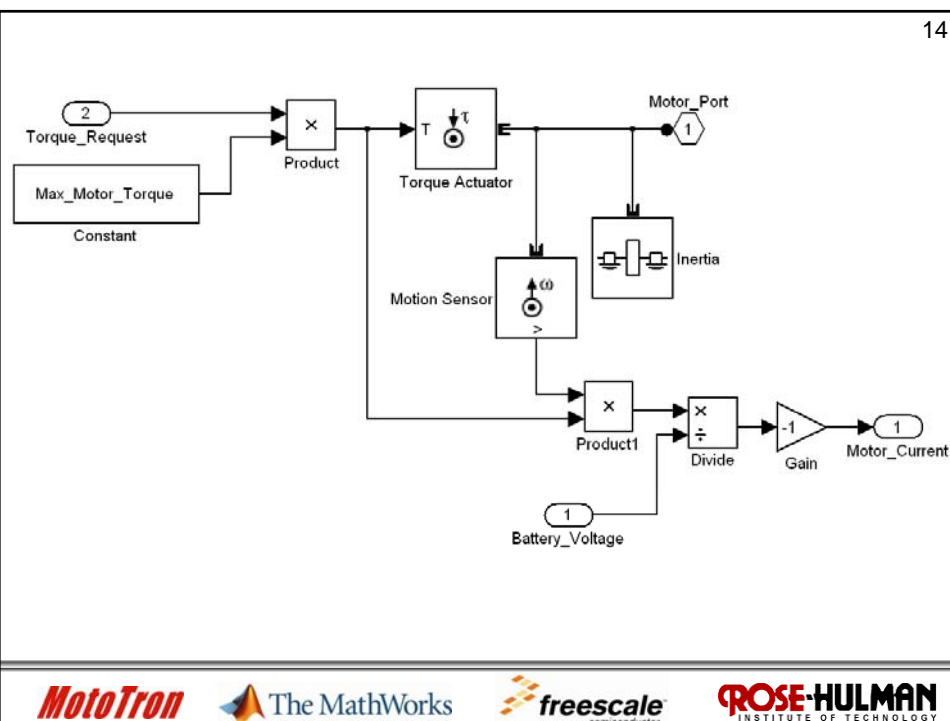
Next, we can calculate the motor current as

$$I_{Motor} = - \frac{\tau_{Motor} \omega_{Motor}}{V_{Battery}}$$

Use a **Motion Sensor** (**SimDriveline/Sensors & Actuators**) to measure the motor shaft speed in r/s.



14





Motor Diagnostics

15

- The last thing we need to do is create the motor diagnostics bus.
- Use the **BUS Creator** part (**Simulink/Commonly Used Blocks**)
- Add the following signals
 - Motor_rpm
 - Motor_torque_Nm
 - Motor_Current_A

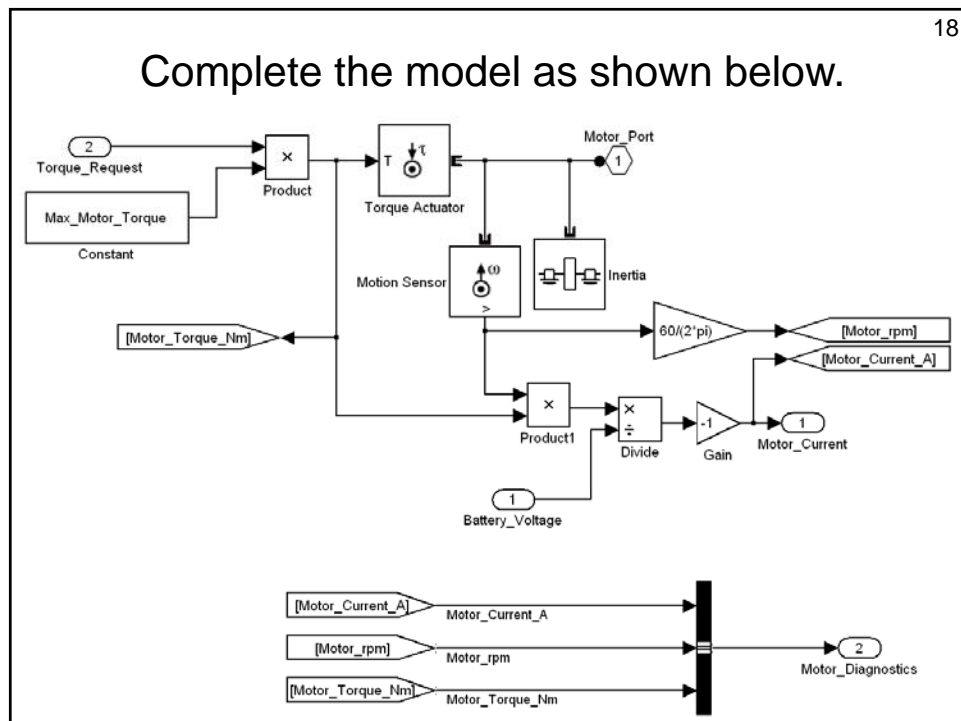
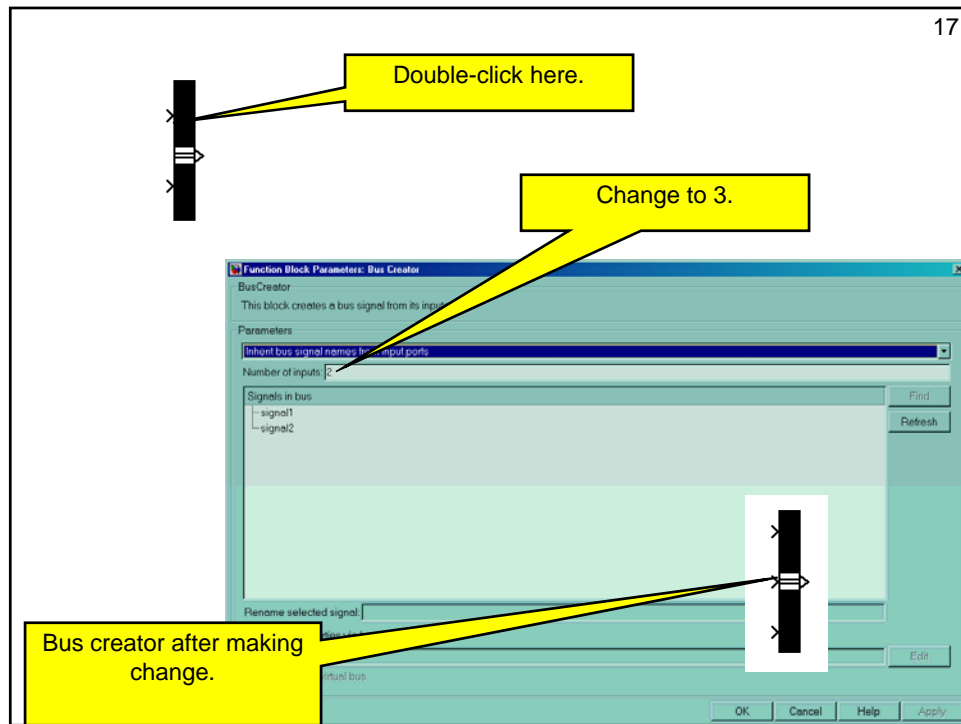


Motor Diagnostics

16

- The Motor torque and motor current are already available.
- Motor speed is available, but is in radians per second. To convert r/s to rpm, multiply by $60/(2\pi)$.
- To change the number of bus inputs, double-click on the bus creator part and change the number from 2 to 3.

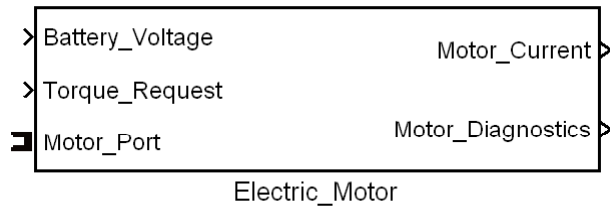




Vehicle Model

19

- When you close the motor model, the motor model subsystem should have the input and outputs as shown:



MotoTron

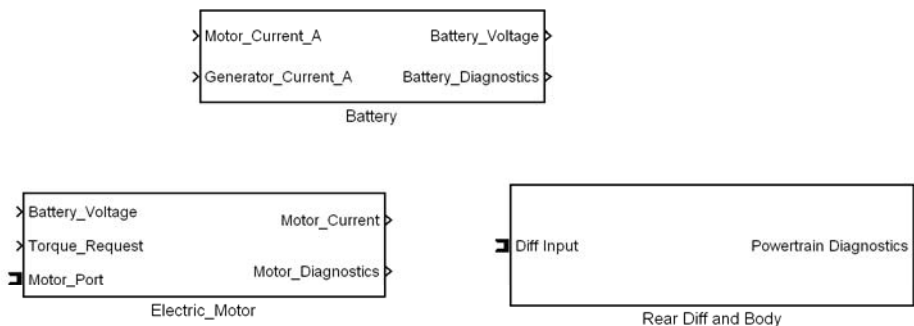
 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Top Level Block Diagram

20



MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

21

Vehicle System

- We now have enough subsystem components to create a simple vehicle. We will create the beginning of a series hybrid electric vehicle by using
 - The electric motor drive the rear diff.
 - The electric motor draw power from the battery.
- Connect the blocks as shown:

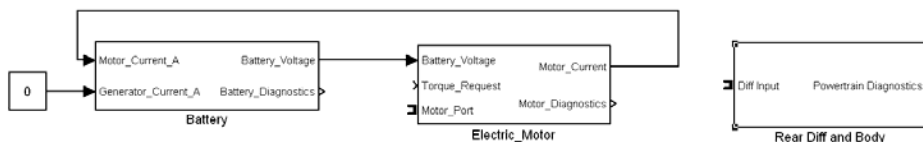
MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

22



The motor port is not in a convenient location to connect it to the rear diff. We can fix this problem by:

- Double-clicking on the Electric_Motor subsystem to open it.
- Double-clicking on the “Motor_Port” part to open its dialog box.

You will see the dialog box shown next:

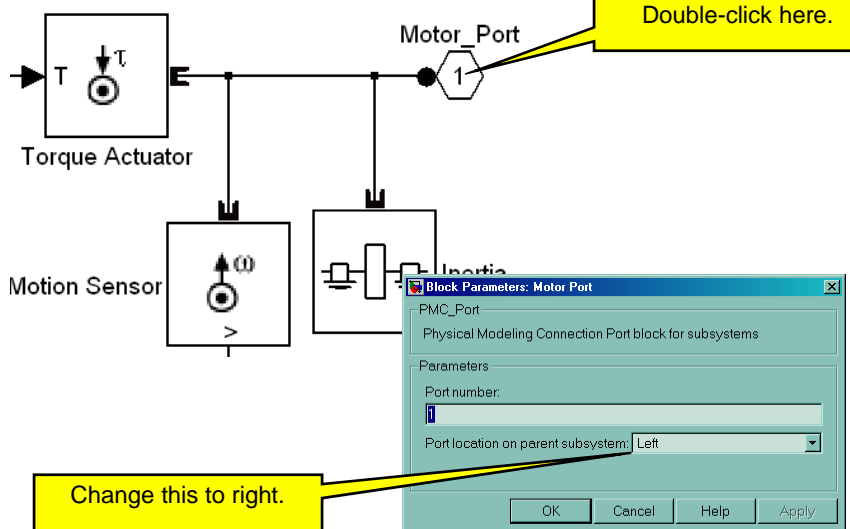
MotoTron

 **The MathWorks**





 **freescale**
semiconductor

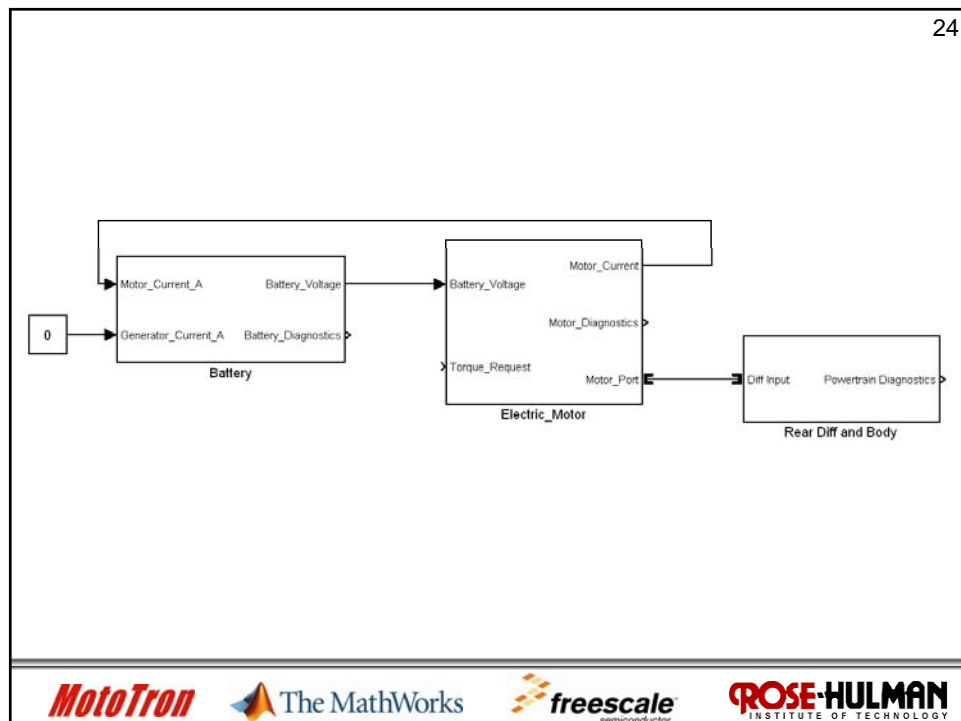
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

23



After making this change, we can easily connect the motor and rear diff.





Vehicle System Diagnostic Bus

25

- Before continuing, we would like to create the system diagnostic bus that contains every diagnostic signal in the model.
- Each subsystem block already has its own diagnostic bus.
- Creating the vehicle system diagnostic bus is just a matter of merging the individual busses using the bus creator part (**Simulink/Commonly Used Blocks**).

MotoTron

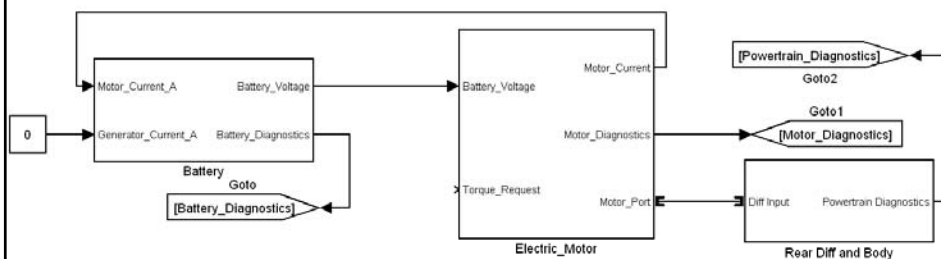
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

First, use the **Goto** part (**Simulink/Signal Routing**) to make connections to the subsystem busses.

26



MotoTron

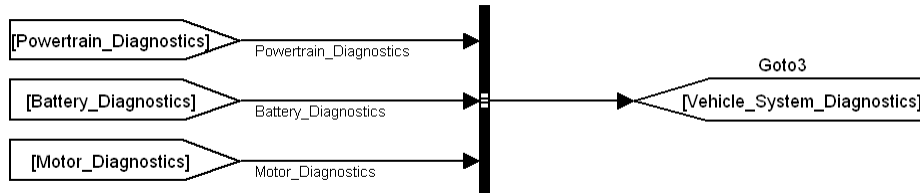
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

27

Create the Vehicle System Diagnostic Bus using the **From** part (**Simulink/Signal Routing**) and the **Bus Creator** part (**Simulink/Commonly Used Blocks**)



MotoTron

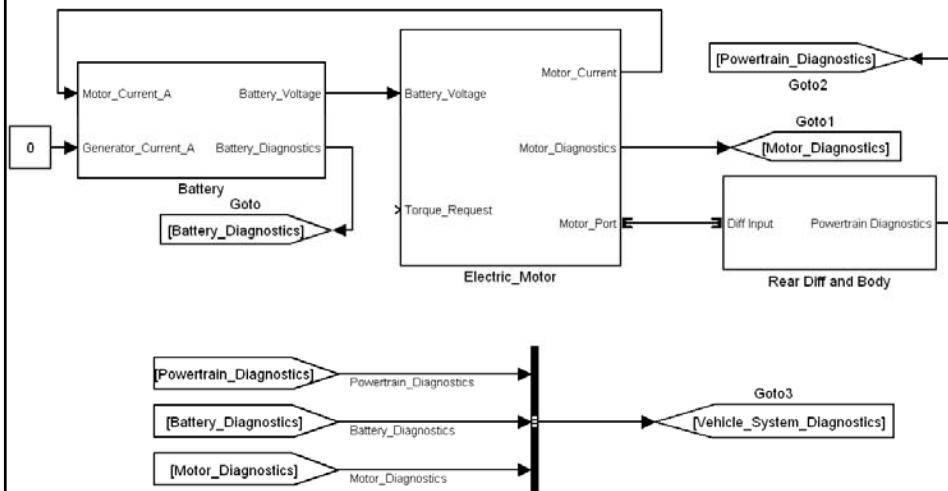
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

28

Top Level Block Diagram



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



29

Vehicle Driver Block

- This block allows us to follow a drive cycle.
- The input to this block is the vehicle's present speed.
- The output is a torque request (-1 to +1) that tells the car to speed up or slow down.
- Note that the driver block is not part of the physical system. It is for simulation purposes only and generates a torque request that would normally come from the vehicle's accelerator and brake pedals.



30

Vehicle Driver Block

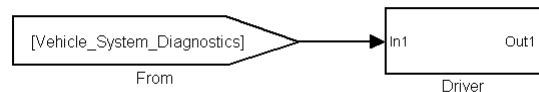
- This is a classic feedback system where:
 - we compare the desired speed to the actual speed
 - create an error signal
 - amplify the error signal,
 - pass that signal to the plant (which is our vehicle).
- The desired speed will be stored in a variable.



Vehicle Driver Block

31

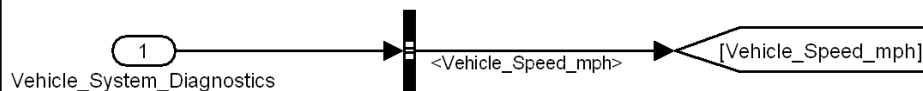
- Create a subsystem with one input and one output by placing the Subsystem block in your top level block diagram.
- The input to this block will be the Vehicle System Diagnostics bus which will contain the speed of the vehicle.
- Rename the subsystem “Driver.”



Driver Block

32

- Open the driver block.
- Rename the input port to “Vehicle_System_Diagnostics.”
- Use the **Bus Selector** part (**Simulink/Commonly Used Blocks**) to extract the Vehicle_Speed_mph signal.

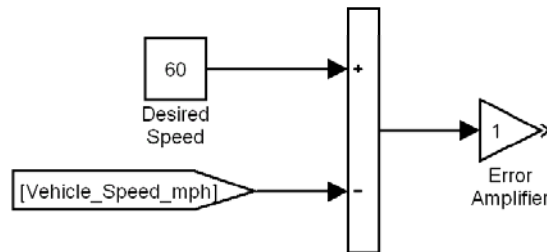




Driver Block

33

- We will now create the feedback system that compares the actual vehicle speed to the desired speed and creates an error signal.
- For the moment, the desired speed will be a constant of 60 mph.



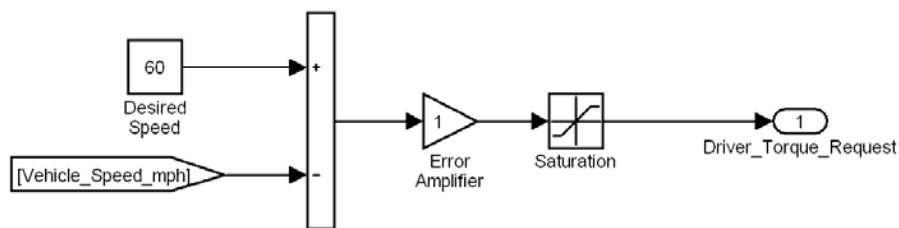
Driver Block

34

- The output of the error amplifier determines the torque.
- Depending on the gain and how far off the speed is, the torque signal can be from $-(\text{big number})$ to $+(\text{big number})$.
- We would like to limit the torque signal to ± 1 .
- Use the **Saturation** part located in the **Simulink/Commonly Used Blocks** library.

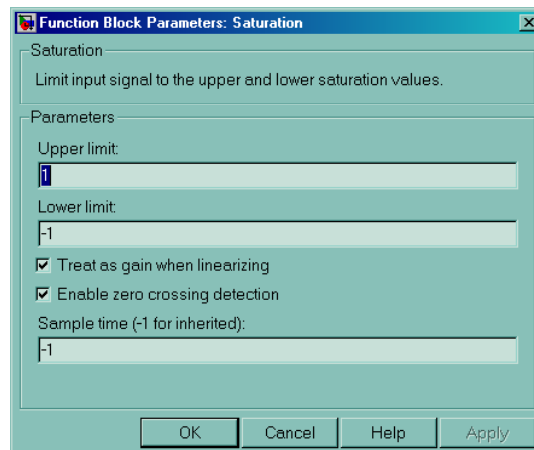
35

- A driver torque request of -1 means full braking.
- A driver torque request of +1 means full acceleration.
- Specify the limits of the saturation part as +1 and -1.
- Rename the output terminal to “Driver Torque Request.”



36

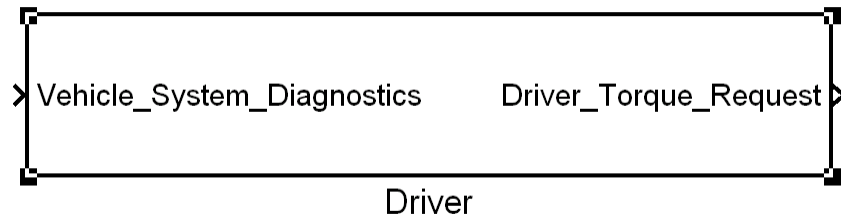
- The saturation limits are specified as shown:



37

Driver Block

- The top-level view of the driver block looks as shown:



MotoTron

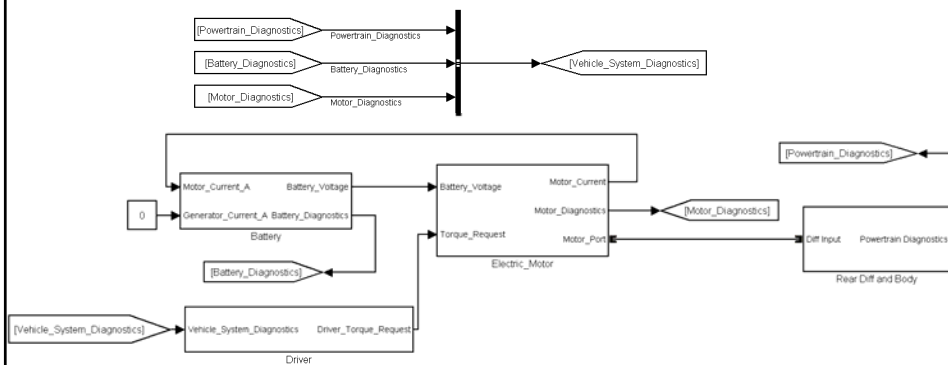
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

38

Top Level Block Diagram



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Display Subsystem

39

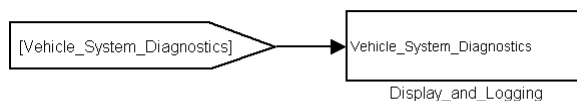
- We are now ready to run a simulation.
- The current driver block uses a constant for the vehicle speed, so the vehicle should accelerate up to a constant speed of 60 mph and hold the speed constant.
- For diagnostic purposes, we may want to create several plots to display various signals in different configurations.
- We will also use this subsystem to log data.



Display and Logging Subsystem

40

- We will create a subsystem with a single input (the vehicle system diagnostic bus) and no outputs:



- We will show two methods of creating a display. The first method will use a Scope block. The second will be shown later and use the Signal and Scope Manager.

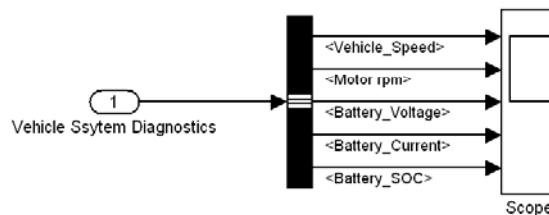




Display Subsystem

41

- Inside the display subsystem, we will use the **Bus Selector** part (**Simulink/Commonly Used Blocks**) and the **Scope** (**Simulink/Commonly Used Blocks**) to create a display of the important signals.



MotoTron

The MathWorks

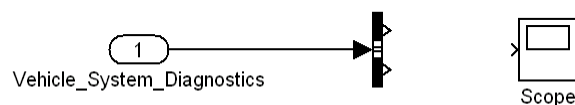
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Display Subsystem

42

- Creating the previous model took several steps.
- When you place the bus selector and scope parts, the two parts do not have the desired number of inputs and outputs:



MotoTron

The MathWorks

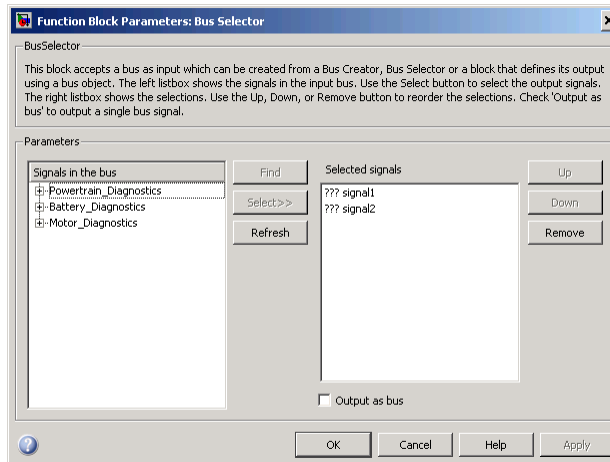
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Creating the Display

43

- Double-click on the **Bus Selector** part 



MotoTron

 **The MathWorks**

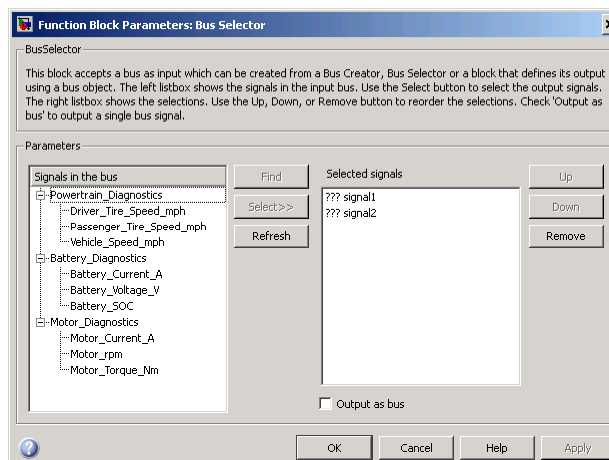
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Creating the Display

44

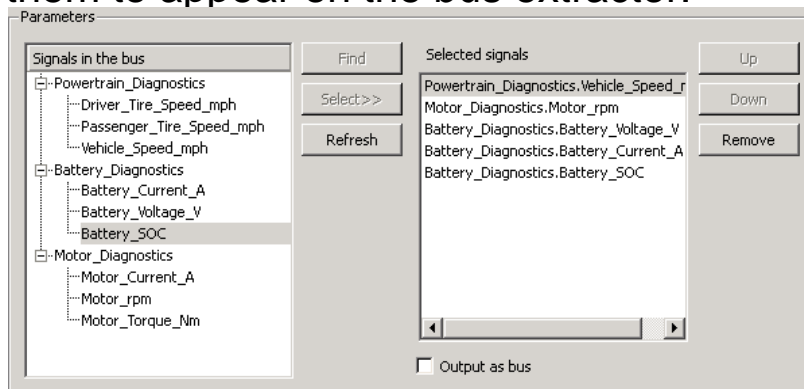
- Remove the signals ??? Signal1 and ??? Signal2.
- Click on the + signs in the “Signals in the Bus” window to view the available signals.



Creating the Display

45

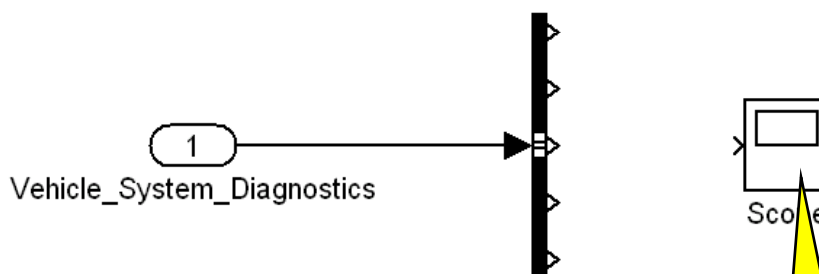
- Select the signal you want to extract and then click the **Select** button.
- Select the signals in the order you want them to appear on the bus extractor.



Creating the Display

46

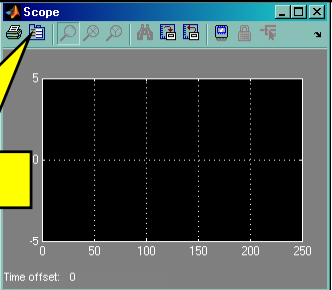
- Click the **OK** button and resize the **Bus Selector** part.




- Next, double-click on the **Scope** part:

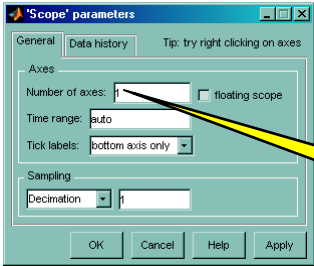
Double-click here.

47



Click on this button.

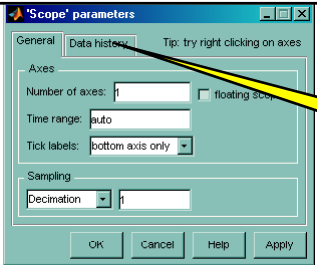
- Click on the Parameters button 



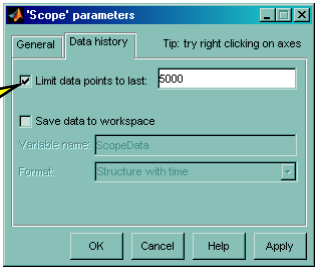
Change this to 5.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

48



Click on the Data history Tab.

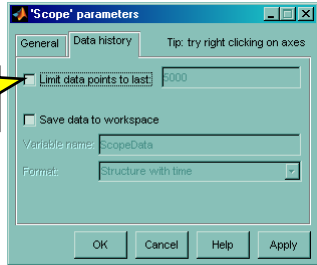


Click here to uncheck this box. Deselecting this box will display all data in our simulation.

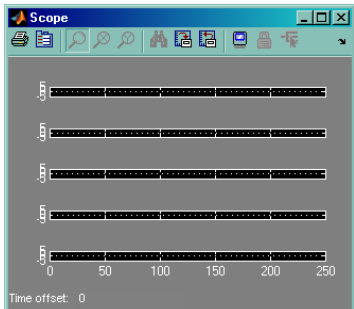
MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

49

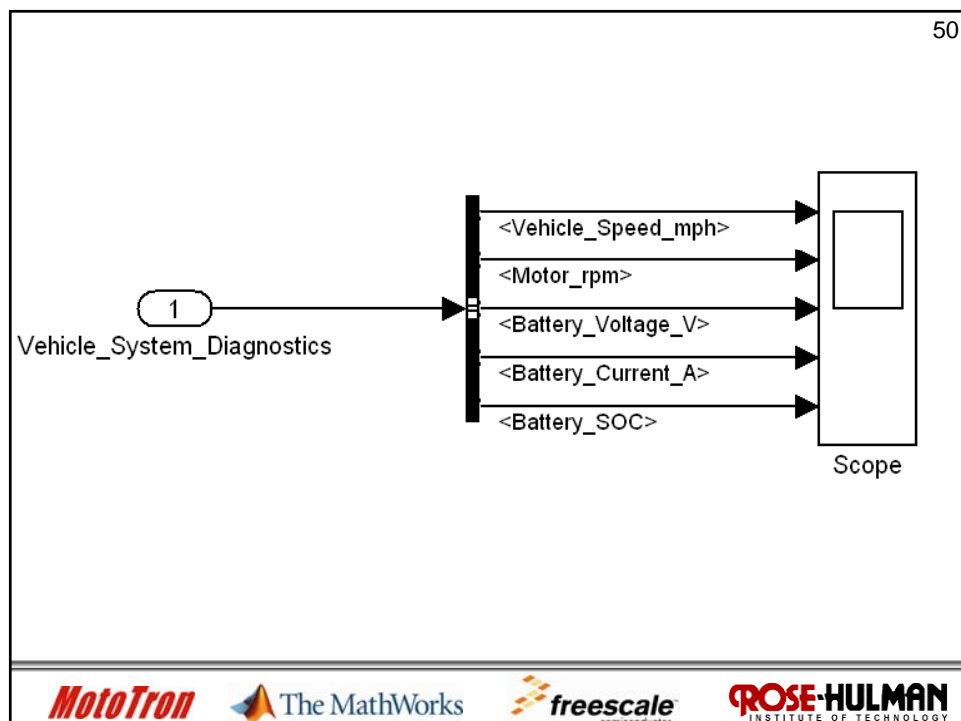
Box unchecked.



Click the OK Button.



Close the scope window and return to the model. Resize the scope part and connect the signals.

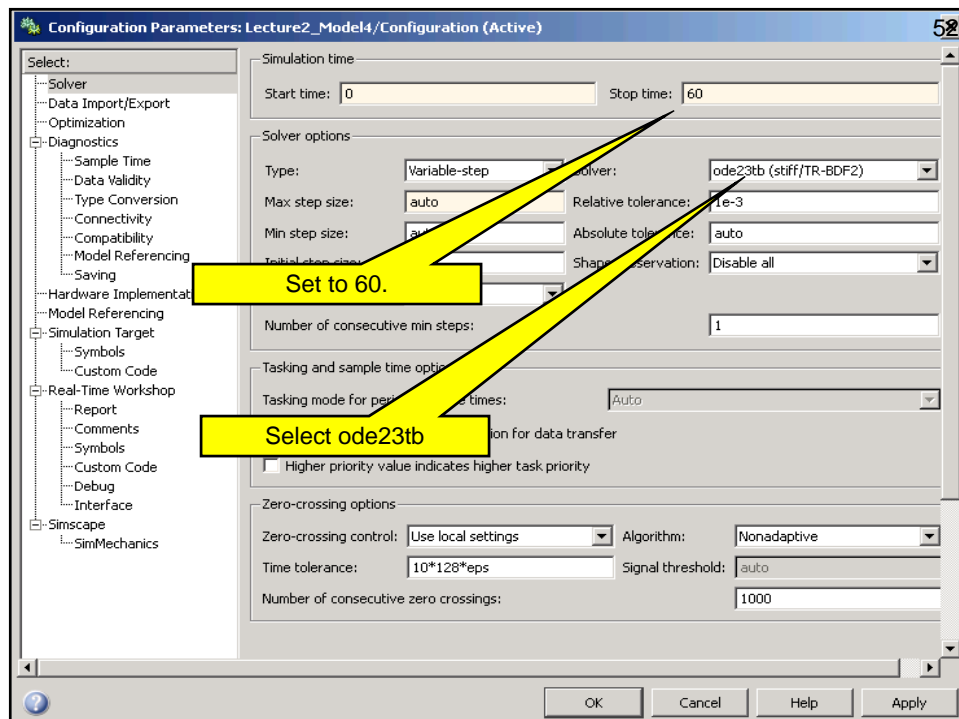




Running a Simulation

51

- We are now ready to set up and run a simulation.
- Select **Simulation** and then **Configuration Parameters** from the Simulink menus.
- Specify the **Stop time** as 60 (seconds).
- Specify ode23tb as the **Solver**.



Simulation Diagnostics

53

- We would like to enable some diagnostics to help us identify potential errors or problems in our model.
- Select **Diagnostics** and specify Algebraic Loops to generate an error.

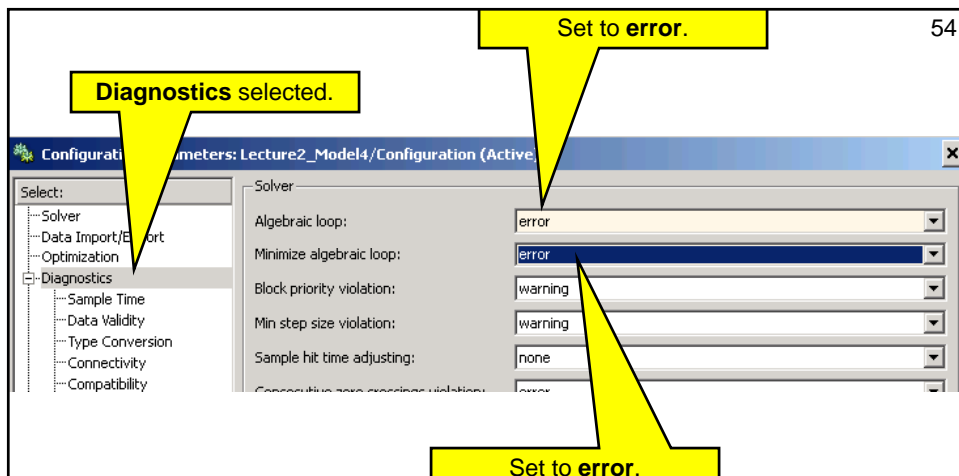
MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

54





Diagnostics selected.

Set to error.

Set to error.


Set to error.

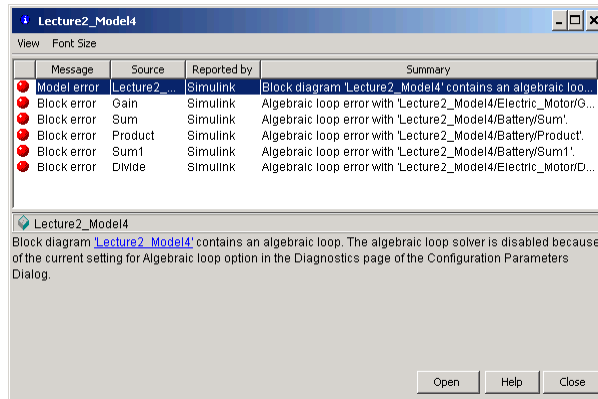
- These selections will highlight an algebraic if one is detected.

MotoTron  **The MathWorks**  **freescale**
semiconductor **ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY

Running the Simulation

55

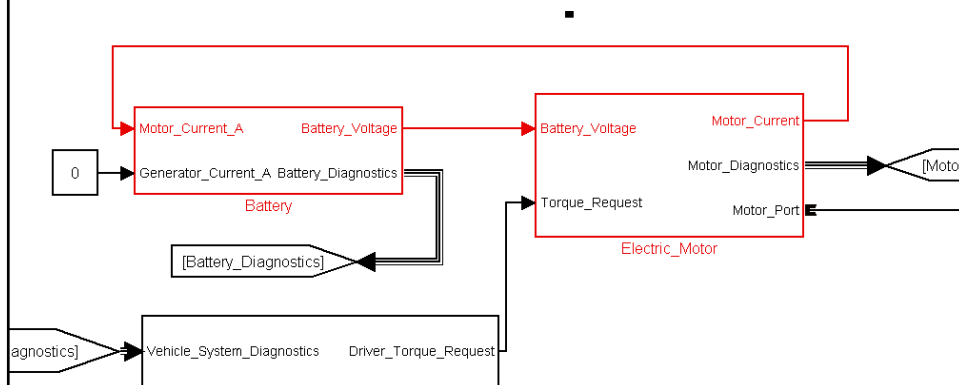
- Click the **OK** button.
- Select **Simulation** and then **Start** from the Simulink menus or click the play button. 
- It appears that we have an error.



Algebraic Loop

56

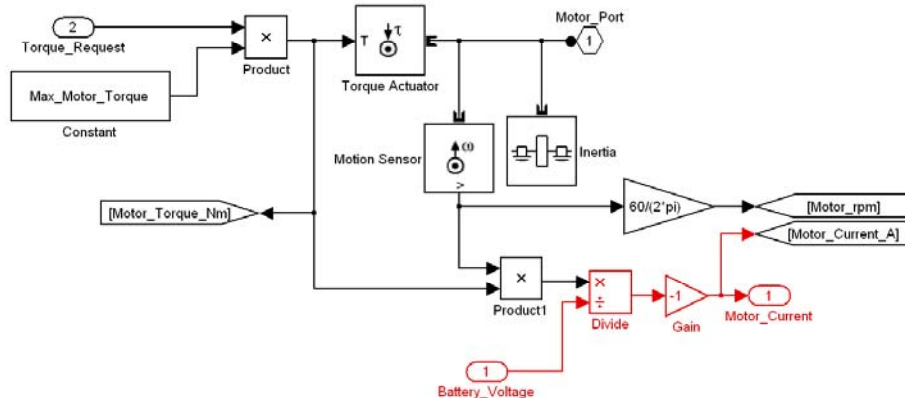
- The top-level block diagram shows that there is an algebraic loop formed between two of our subsystems. The Algebraic loop is highlighted in red:



Algebraic Loop

57

- If you look inside the Electric_Motor Subsystem, you will see the offending components in that subsystem:

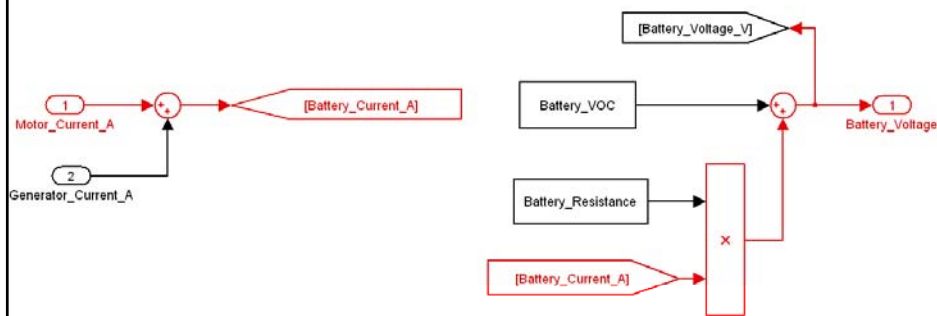


In order for Simulink to calculate the motor current, it needs to know the battery voltage.

Algebraic Loop

58

- If you look inside the Battery Subsystem, you will see the offending components in that subsystem:



In order to calculate the battery voltage, Simulink needs to know the motor current?



Algebraic Loop – The Loop


59

- In the motor model, to calculate the motor current, Simulink needs to know the battery voltage.
- In the battery model, to calculate the battery voltage, Simulink needs to know the motor current?
- Thus, Simulink does not have the information it needs to make this calculation.



Aglebraic Loop

60

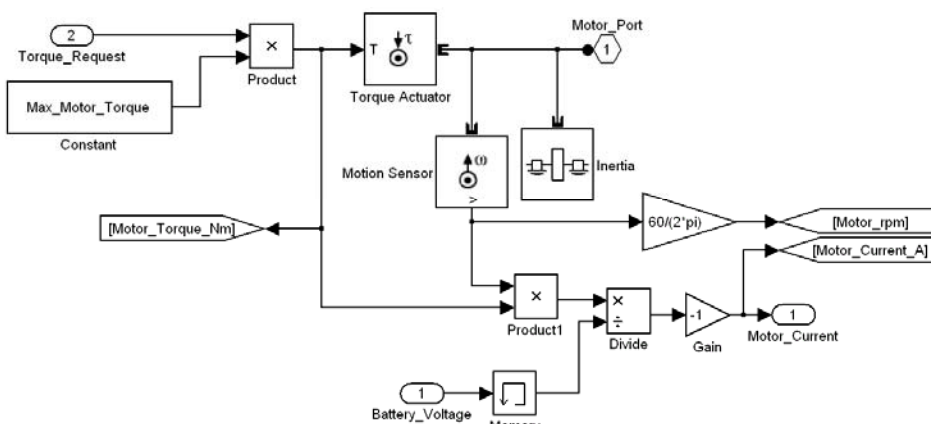
- We can break this loop by adding a **Memory** part  (Simulink/Discrete) to

either the Battery model or the motor model.

- We will add it to the Electric_Motor model as shown:



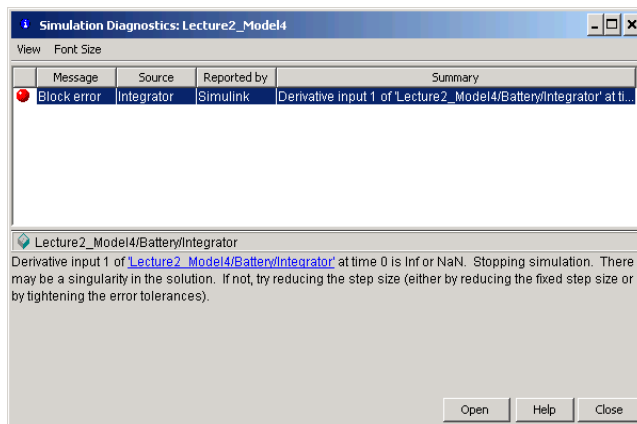
61



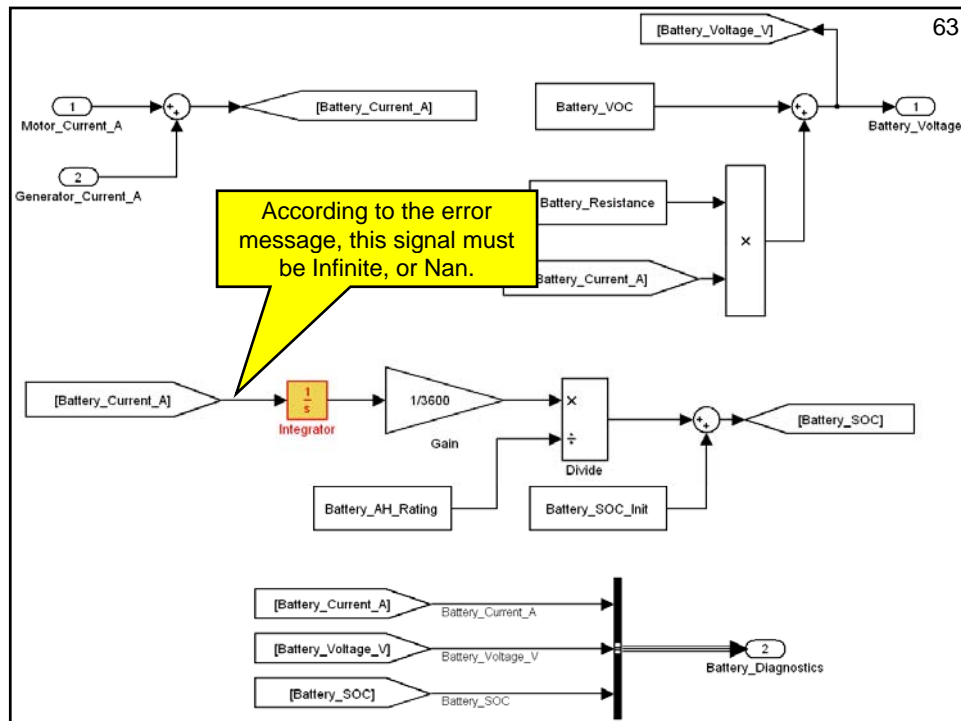
- The memory block is a one time step delay. In the division calculation, Simulink will use the battery voltage from the previous time step. Thus the battery voltage is know, and Simulink can calculate the motor current.
- Rerun the simulation and see if this fixes the problem.

62

- The problem is fixed, but we get another error.



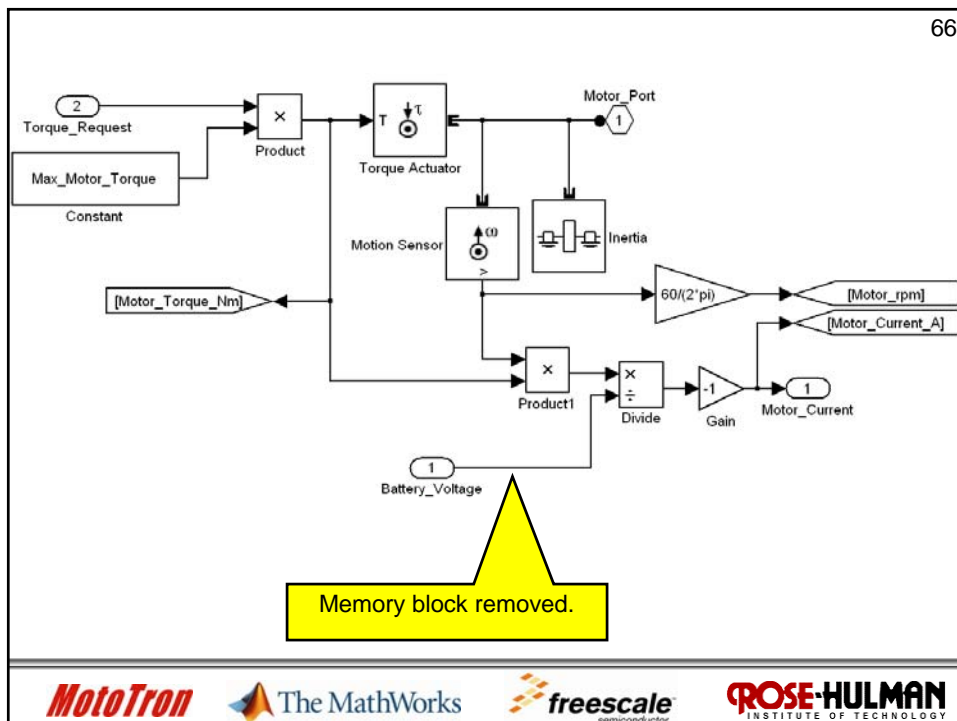
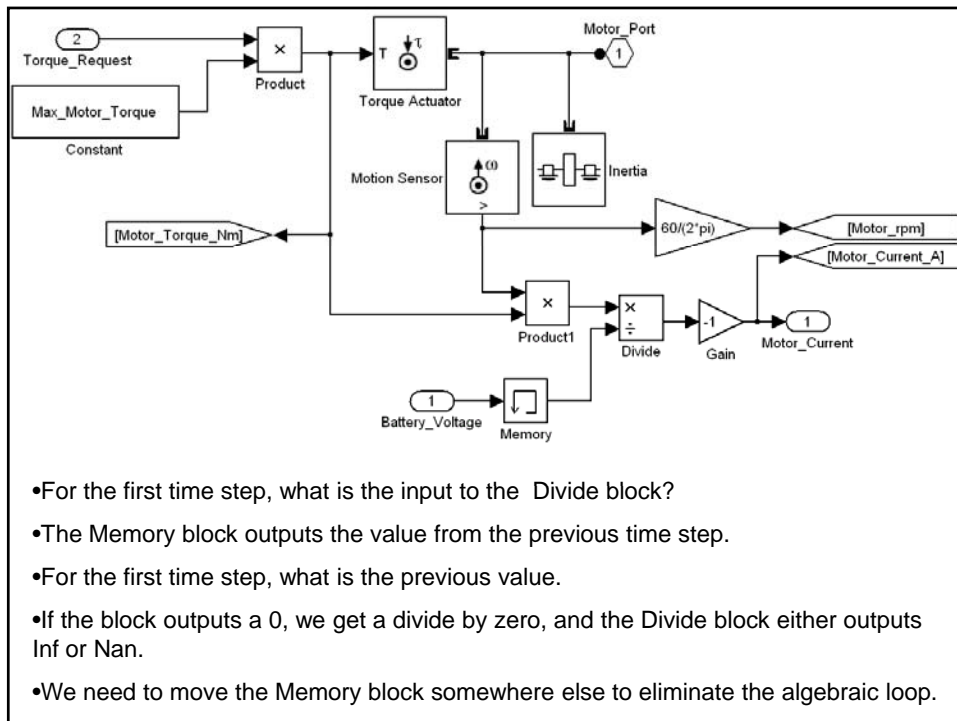
- Click on the link to jump to the error.

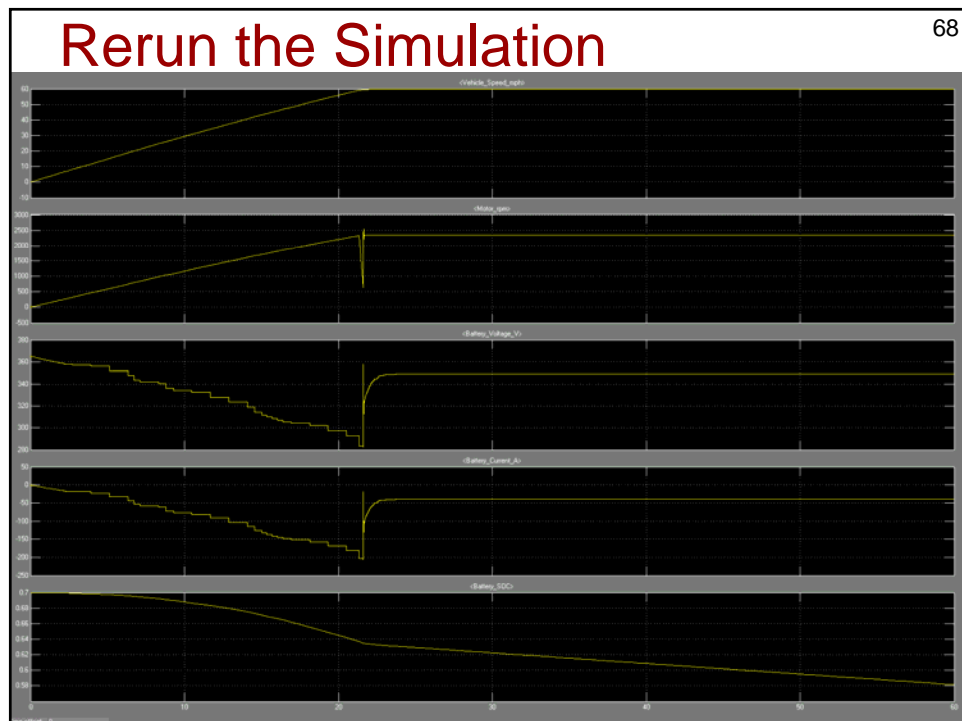
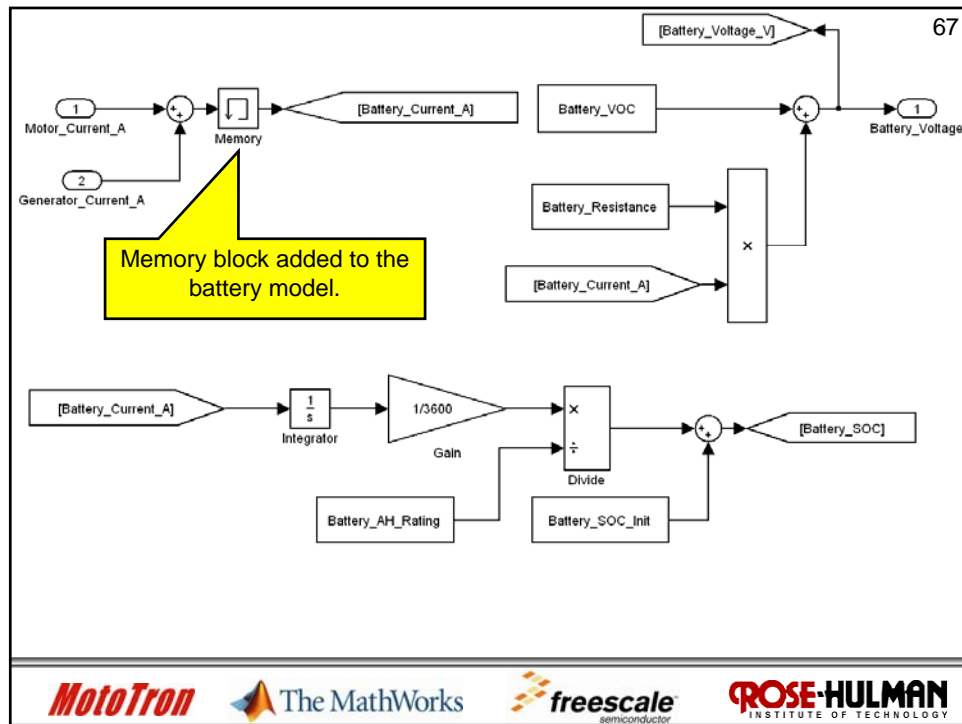


Error

64

- How can the battery current become Infinite?
- Looking at the motor model, we calculate the motor current (which becomes the battery current) as the requested motor power divided by the battery voltage.







Lecture 2 Exercise 1

69

- The Simulation Runs!
- There appear to be some problems with our physical model.
- Demonstrate the operation of your model.
- The Speed Approaches 60 mph, but there are problems in the rpm signal.

Demo_____



Advanced Model-Based-System Design

Debugging the Model



71

Debugging the Model

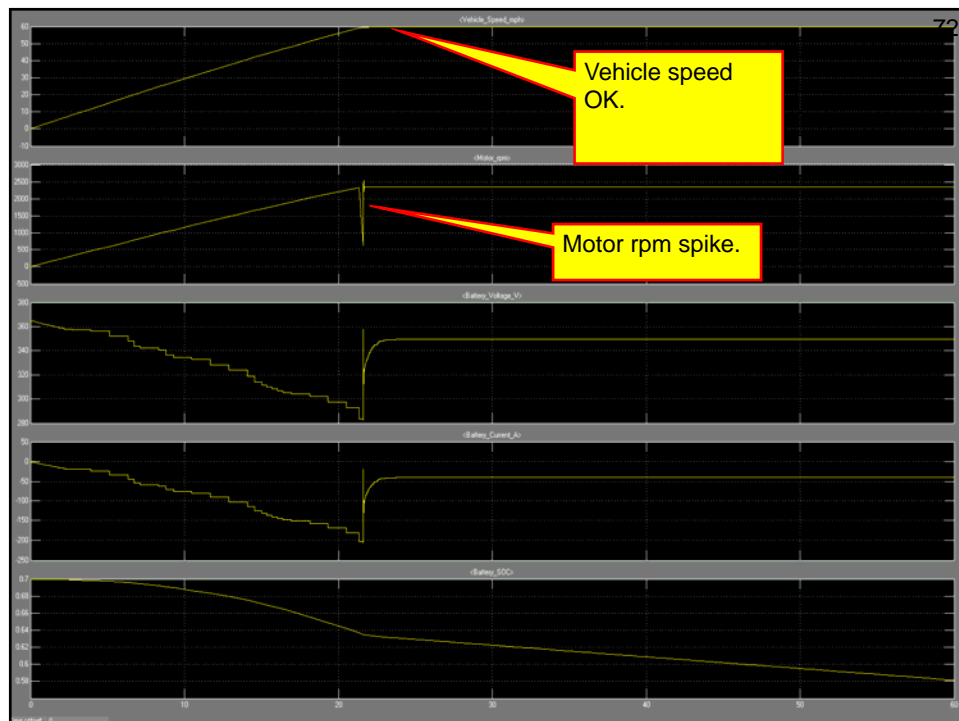
- The previous slide had a problem with the motor rpm, battery voltage, and battery current:

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Model Debugging

73

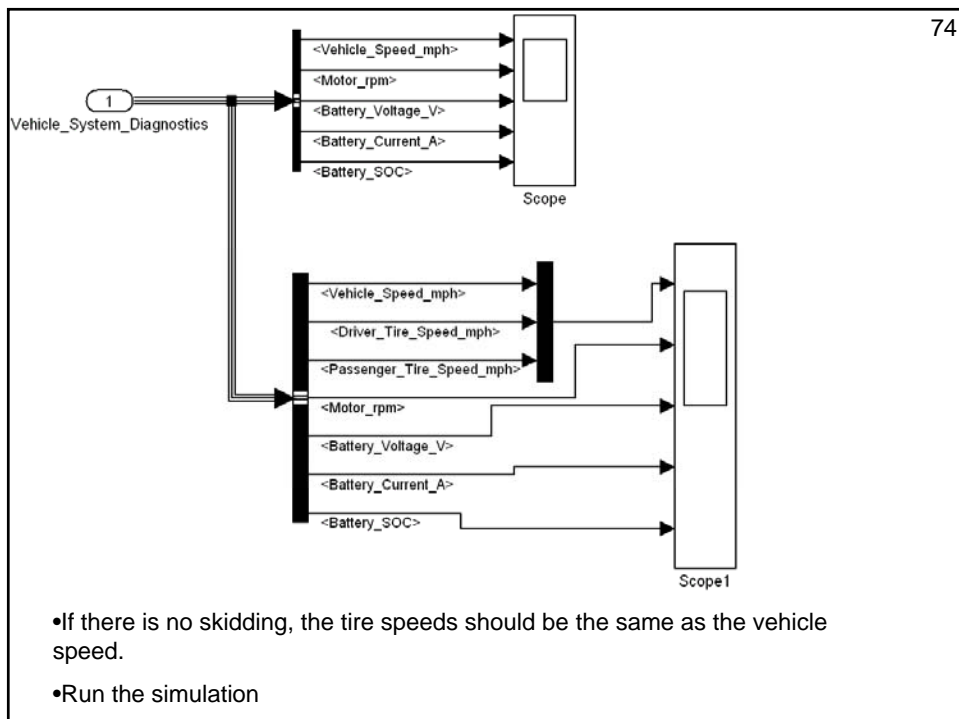
- With a direct drive system, how can the vehicle speed follow a different curve than the motor speed?
- ➔ The wheels must be skidding.
- We will verify this with another plot which we will add to the Display subsystem.

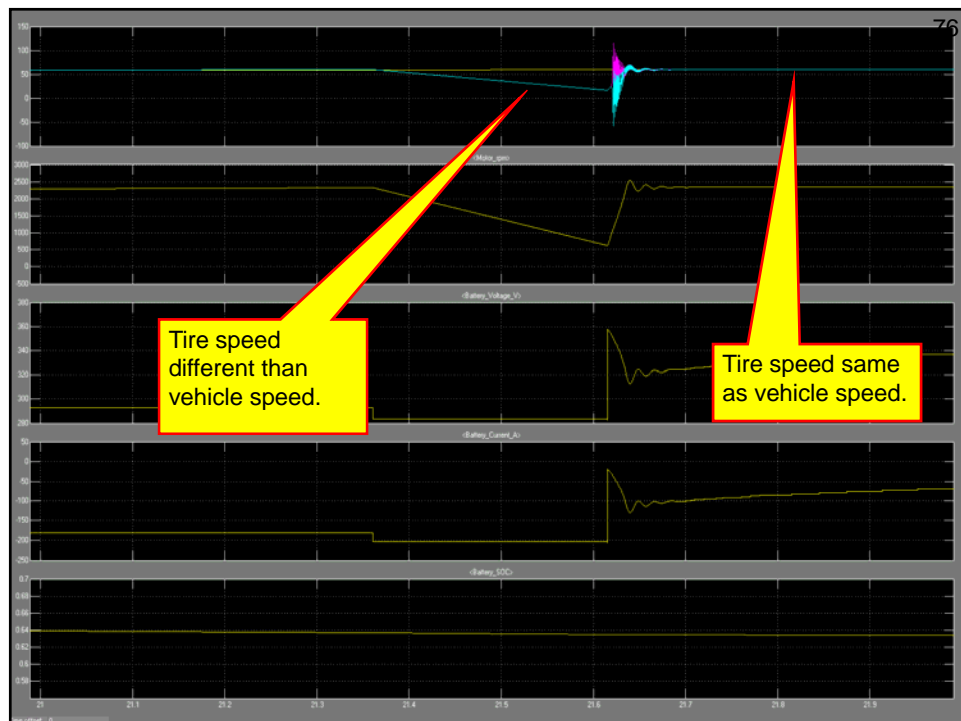
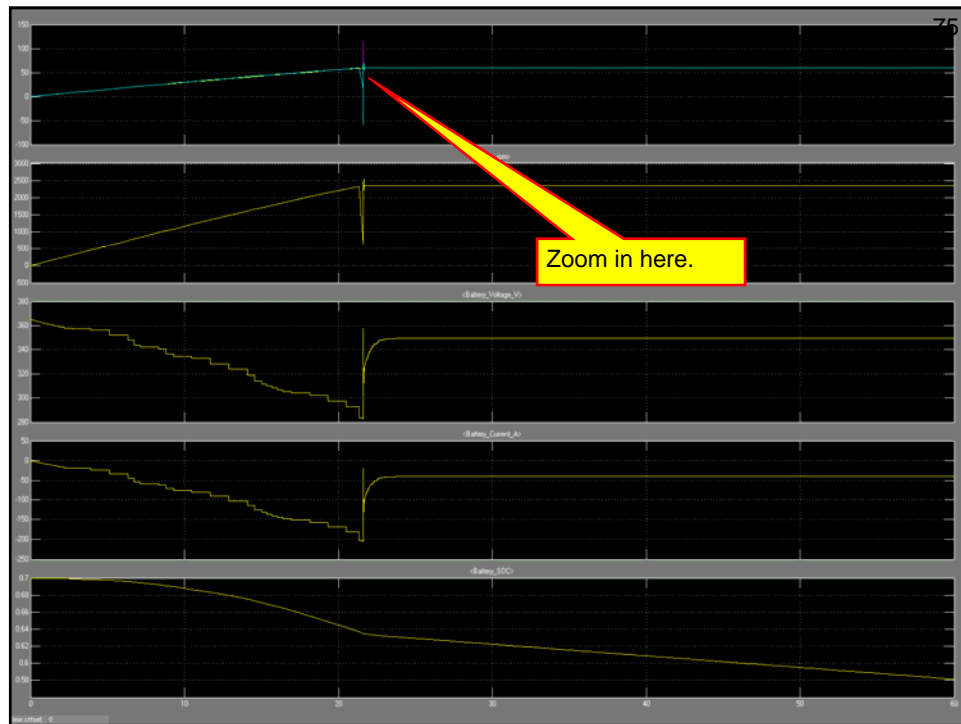
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

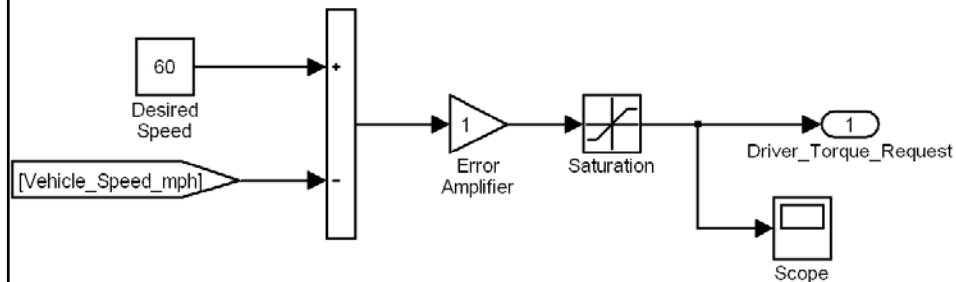




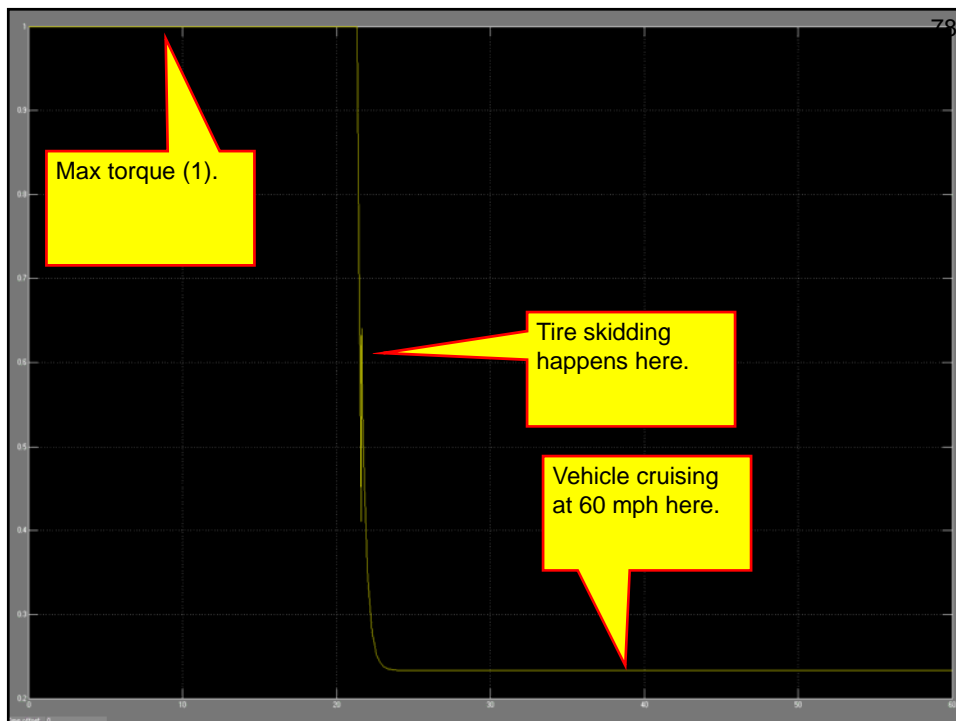
Debugging the Model

77

- Lets look at the torque request coming from the driver request block.
- Open the driver block and add a scope to the driver torque request signal.



Rerun the simulation and view the scope.





Model Debugging

79

- It looks like the tire skidding occurs when we have a large change in the driver torque request.
- We can fix this by adding a **Rate Limiter** to the driver torque request.
(**Simulink/Discontinuities**)

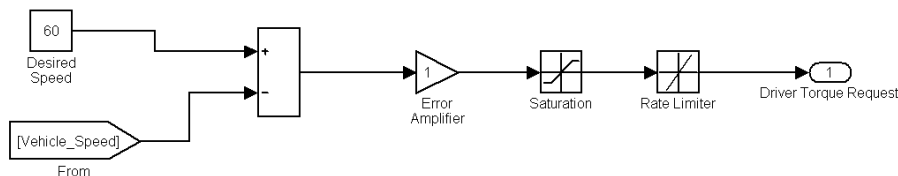


MotoTron

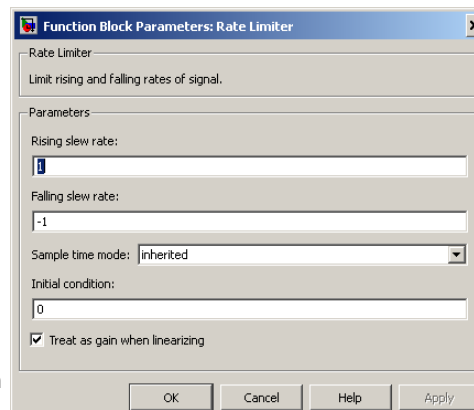
The MathWorks

freescale
semiconductor

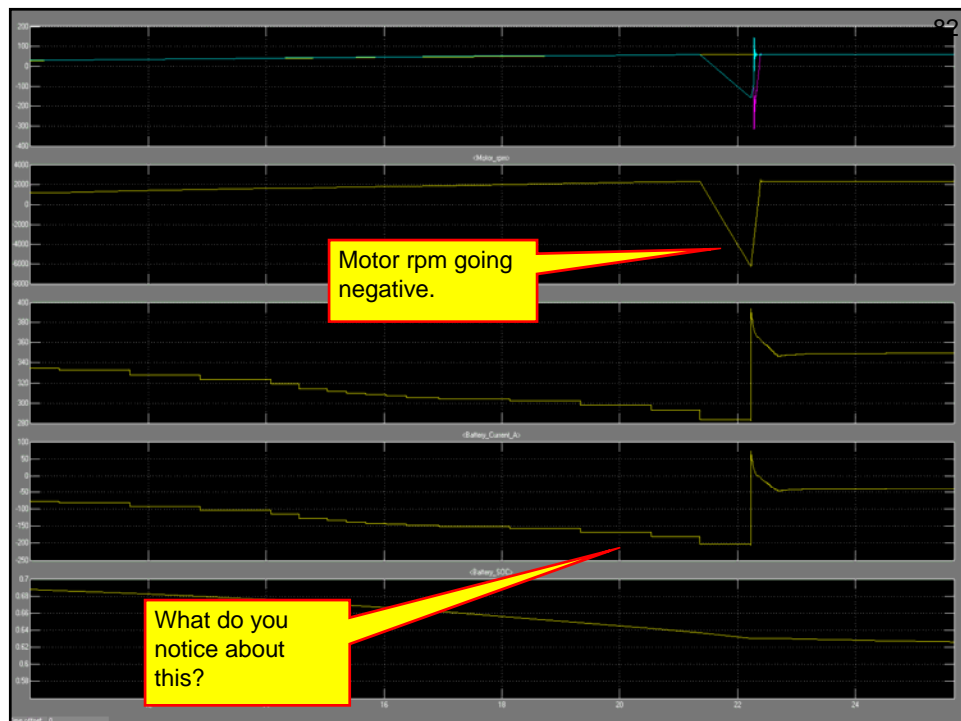
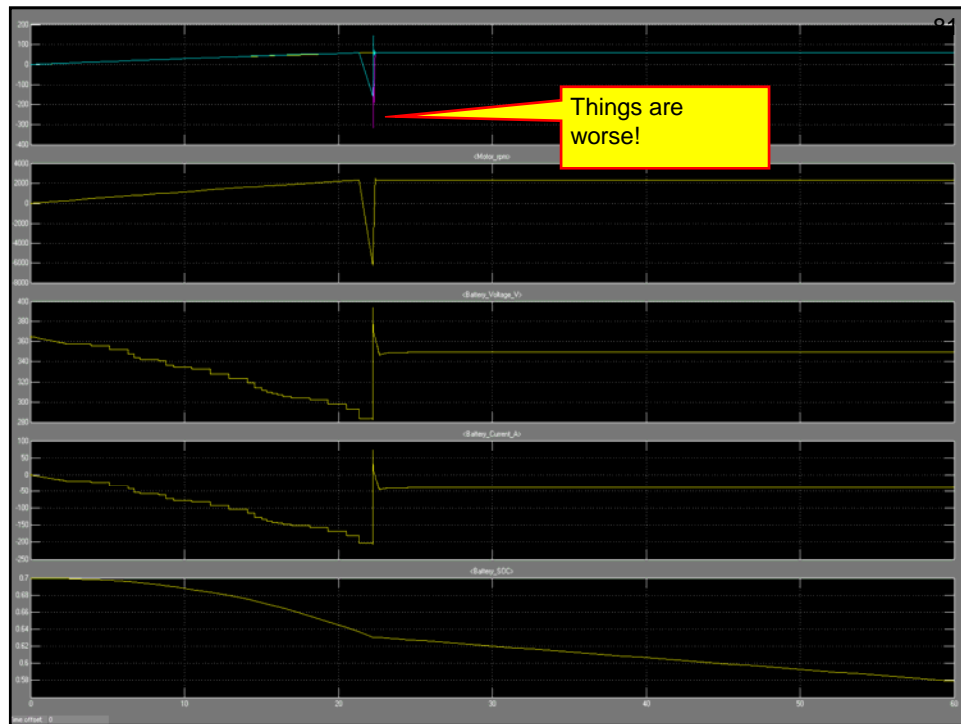
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



- The rate limiter places a limit on how fast a signal can increase or decrease.
- The default rising and falling rates are 1/second.
- We will use the default values.
- Since the driver torque request signal is between -1 and +1
 - Our motor will go from no torque (0) to full torque (1) in one second.
 - Our motor will go from full forward torque (1) to full reverse torque (-1) in two seconds.



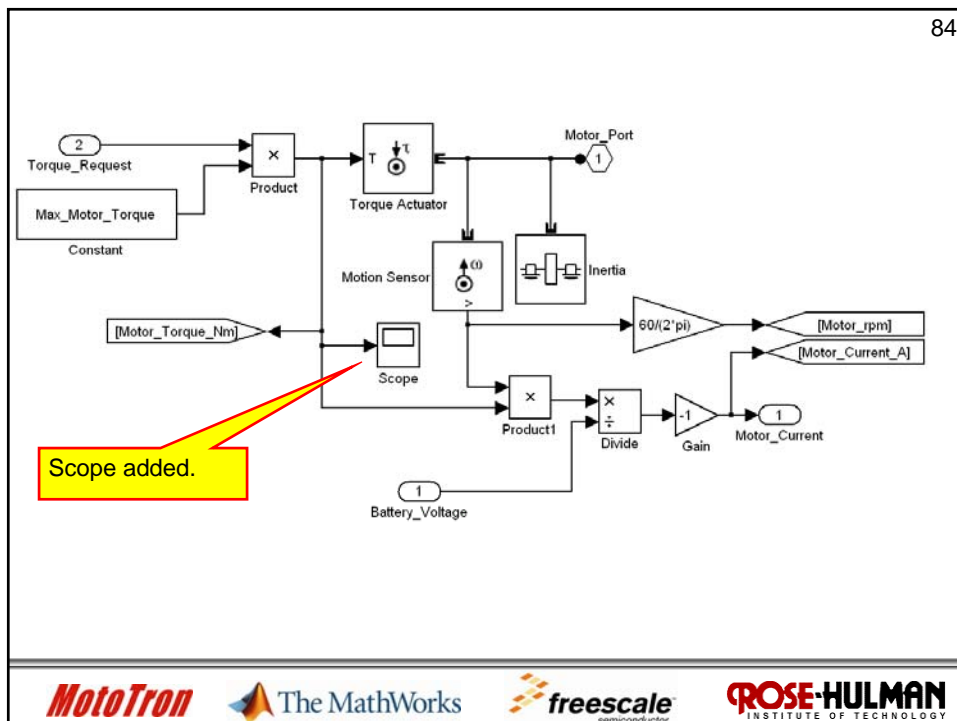
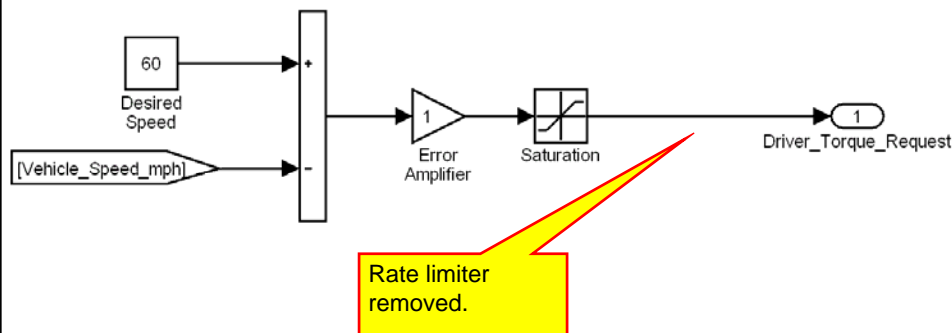
Run a simulation with the rate limiter.



Model Debugging

83

- Let's remove the rate limiter since it made things worse.
- Let's plot the motor torque using a scope.

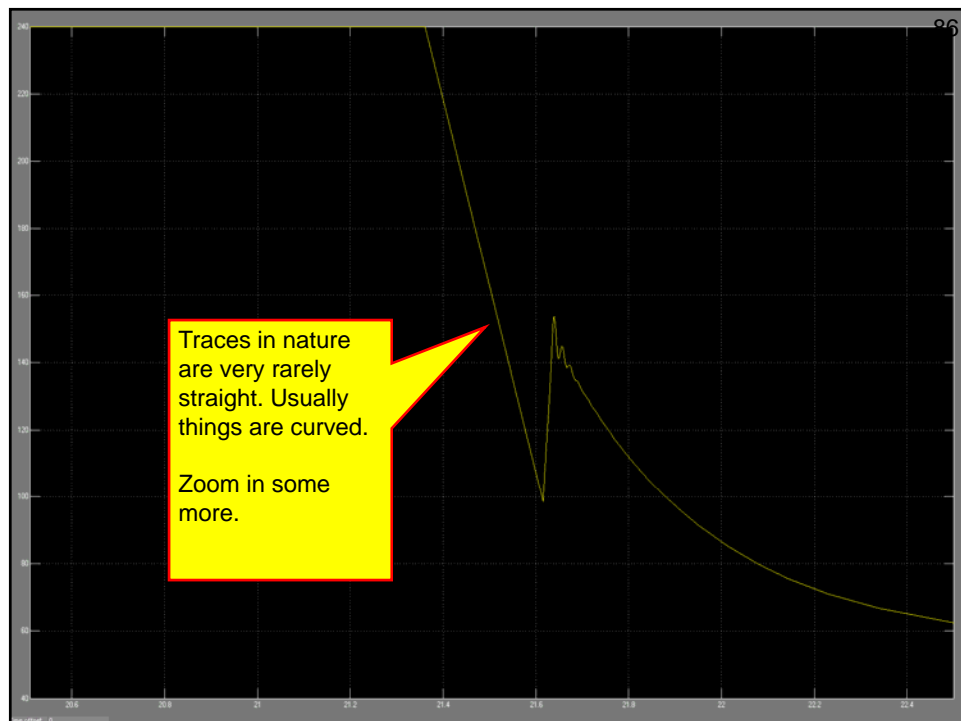
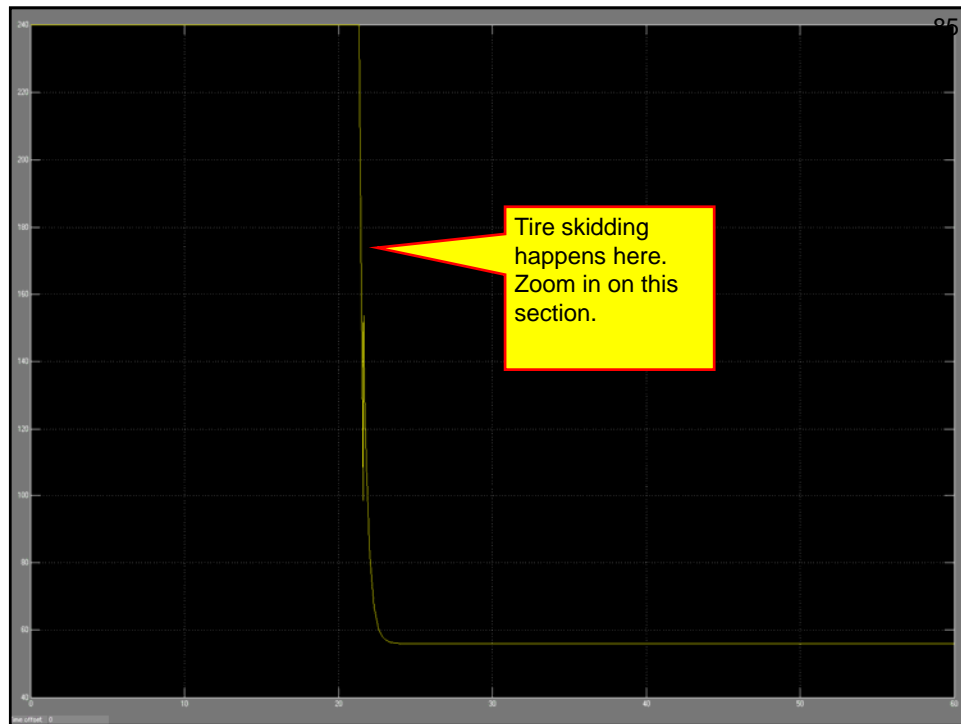


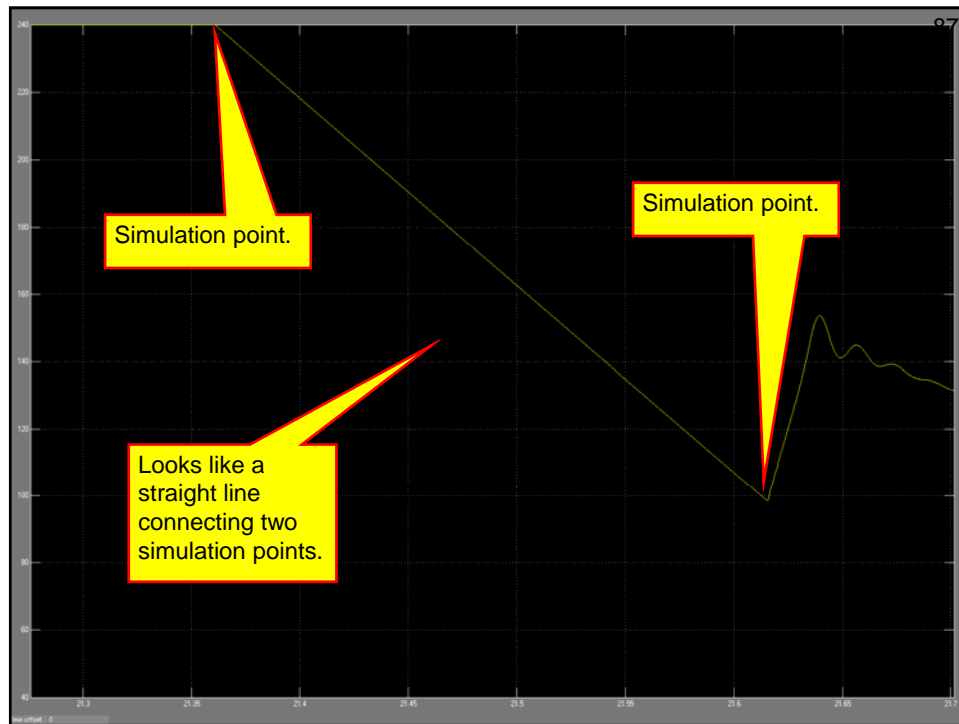
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

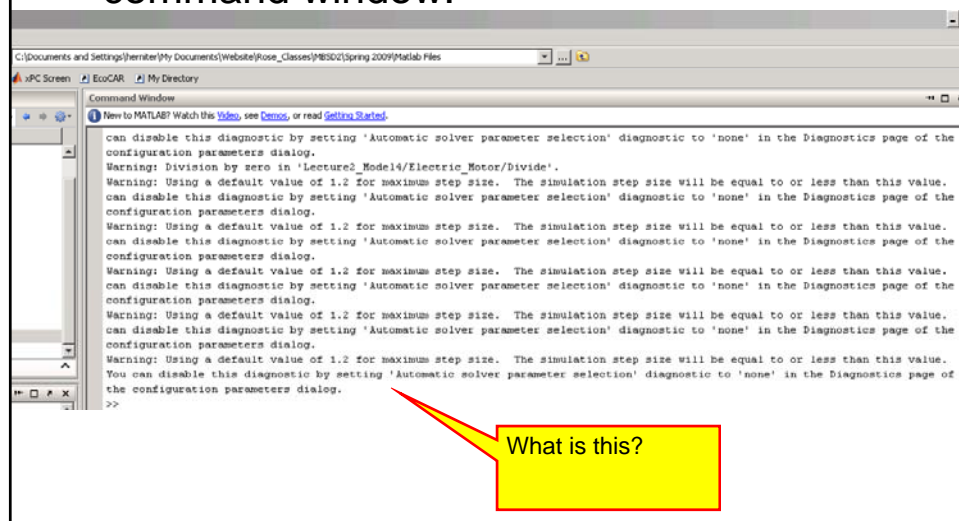




Model Debugging

88

- Another hint comes from the MATLAB command window.



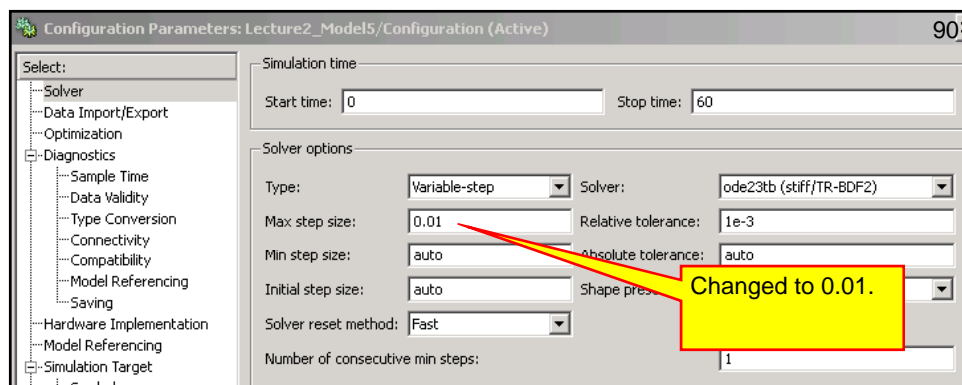


89

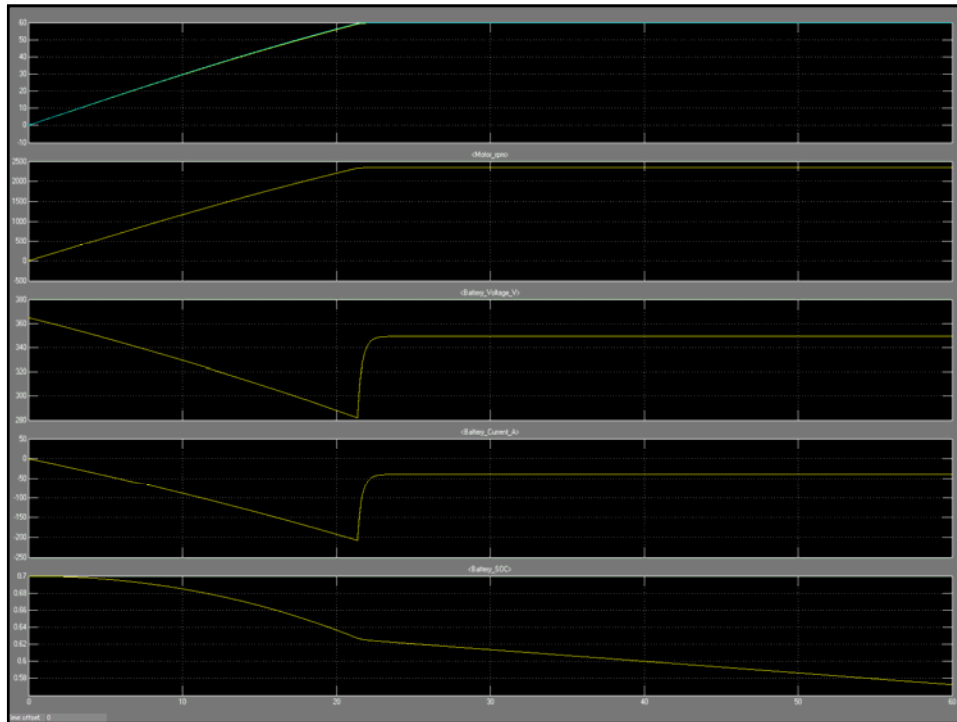
Warning: Using a default value of 1.2 for maximum step size. The simulation step size will be equal to or less than this value.

You can disable this diagnostic by setting 'Automatic solver parameter selection' diagnostic to 'none' in the Diagnostics page of the configuration parameters dialog.

- It looks like Simulink can take a step size as large as 1.2 seconds between simulation points.
- This is a large time step for our simulation. From the previous slides, it looked like we had a step of 0.35 seconds that was too large.
- Let's try a smaller step size.
- Select **Simulation** and then **Configuration Parameters** from the Simulink menus.
- Change the **Max Step size** from Auto to 0.01 (seconds)



- This parameter specifies that the maximum time between simulation points will be 0.01 seconds. (It can be smaller if necessary.)
- Rerun the simulation with this change.



Model Debugging

92

- Everything looks great.
- Our error was numerical rather than something nonphysical or too ideal in our model.
- Later we will have problems because our system is too ideal, such as torque spikes and negative battery voltages.



Lecture 2 Exercise 2

93

- We would like our vehicle to be able to accelerate from 0 to 60 mph in 9 seconds.
- Determine:
 - Required motor torque _____ (Nm)
 - Required battery current _____ (A)
 - The peak motor power _____ (kW)
- How does our model breakdown if the motor current is too large?

Demo _____





Advanced Model-Based-System Design

Lecture 2: Drive Cycles and Advanced Models



2

Following a Drive Cycle

- Next, we will have the vehicle follow a simple drive cycle.
- We will use a part called **From Workspace** (Simulink/Sources) to read in a 2-D variable.
- Using the init file, define a variable called Sch_Cycle. This is two dimensional matrix.
- The first column contains the time values and the second column contains the speed values:





Following a Drive Cycle

3

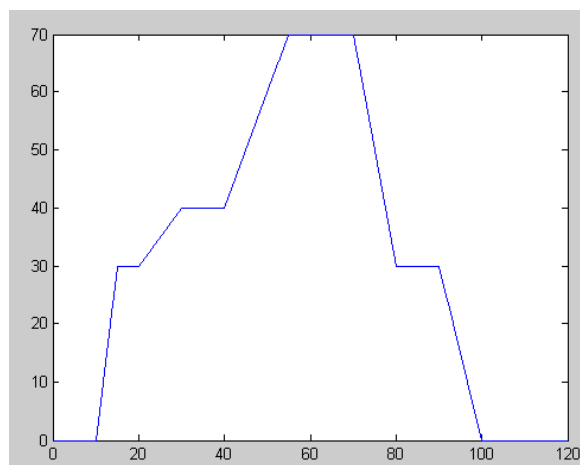
- Sch_Cycle =
 - 0 0
 - 10 0
 - 15 30
 - 20 30
 - 30 40
 - 40 40
 - 55 70
 - 70 70
 - 80 30
 - 90 30
 - 100 0
 - 120 0
- >> Your init file should look like the following:



Following a Drive Cycle

4

```
%Define our driving cycle  
Sch_Cycle = [0 0; 10 0; 15 30; 20 30; 30 40; 40 40; 55 70; ...  
             70 70; 80 30; 90 30; 100 0; 120 0];
```



Following a Drive Cycle

5

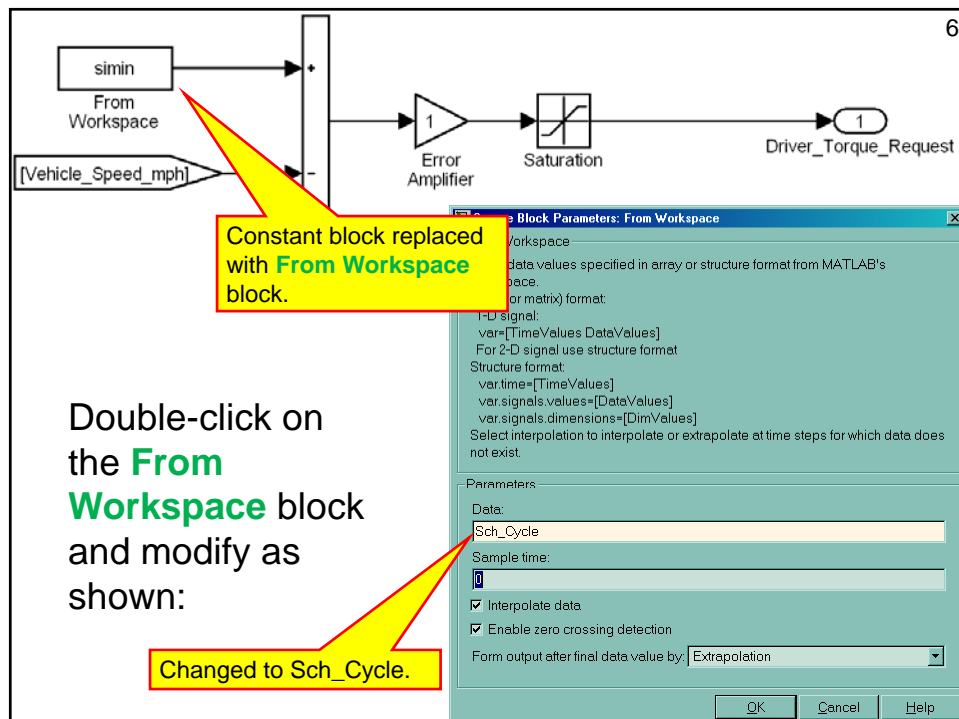
- Run the init file to load the variables into memory.
- In the Driver block, add the **From Workspace** part which is located in the **Simulink/Sources** library.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Following a Drive Cycle

7

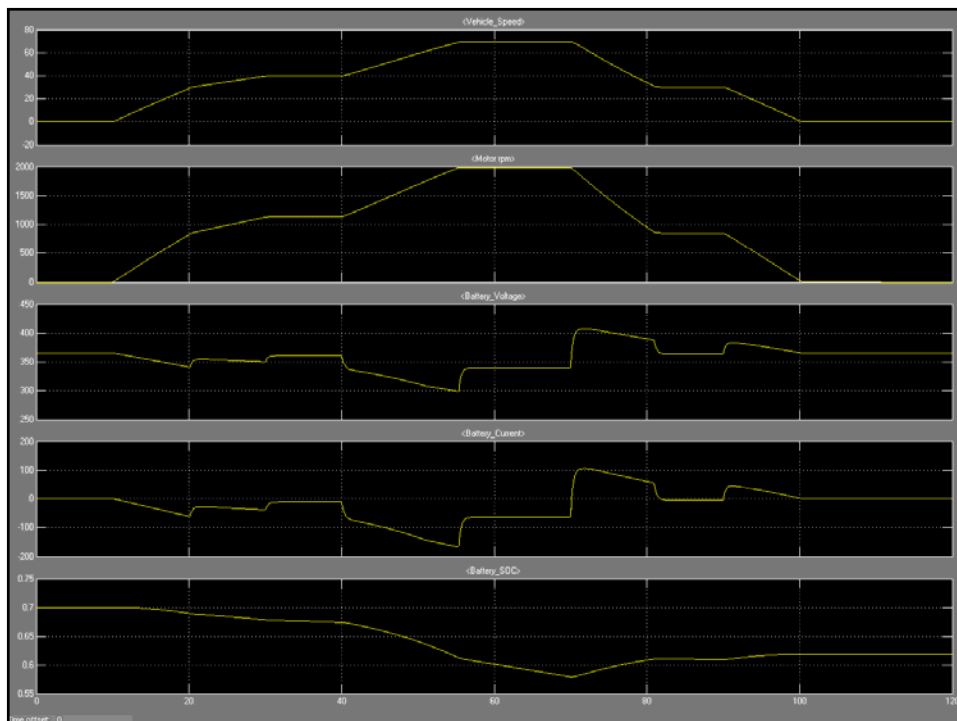
- Click the **OK** button to save the changes.
- Set the simulation time to run for 120 seconds.
- Run the simulation and view the plot:

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Following a Drive Cycle

9

- We see that the vehicle does follow the specified profile.
- Since there are no mechanical brakes, the motor is responsible for slowing the vehicle. → Regen braking is working.
- We see the battery discharge as the vehicle accelerates.
- We see the battery charge as the vehicle decelerates.

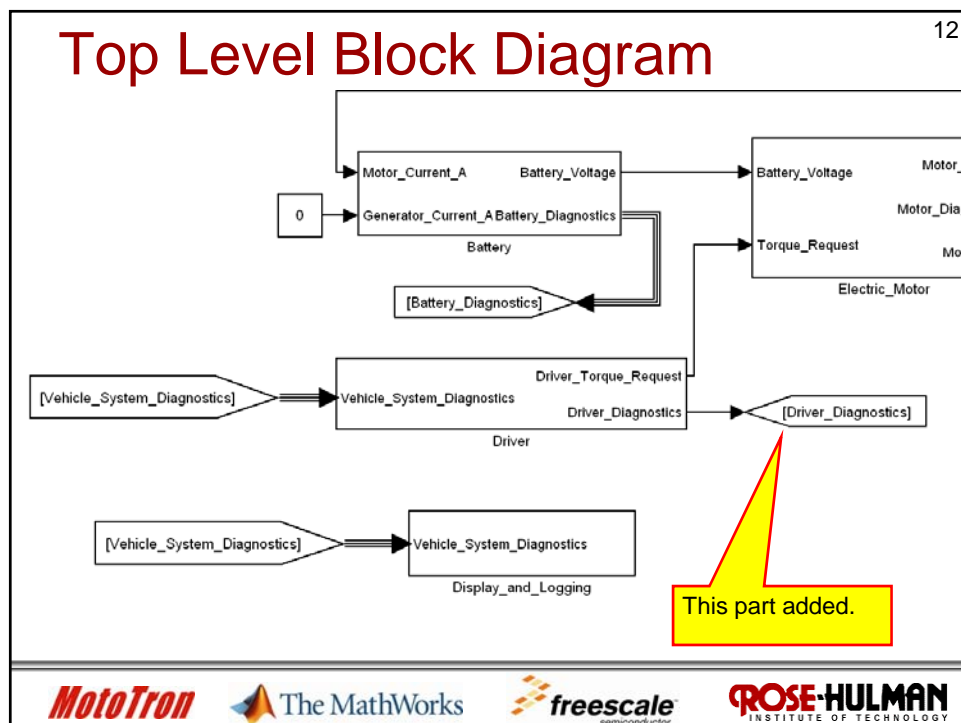
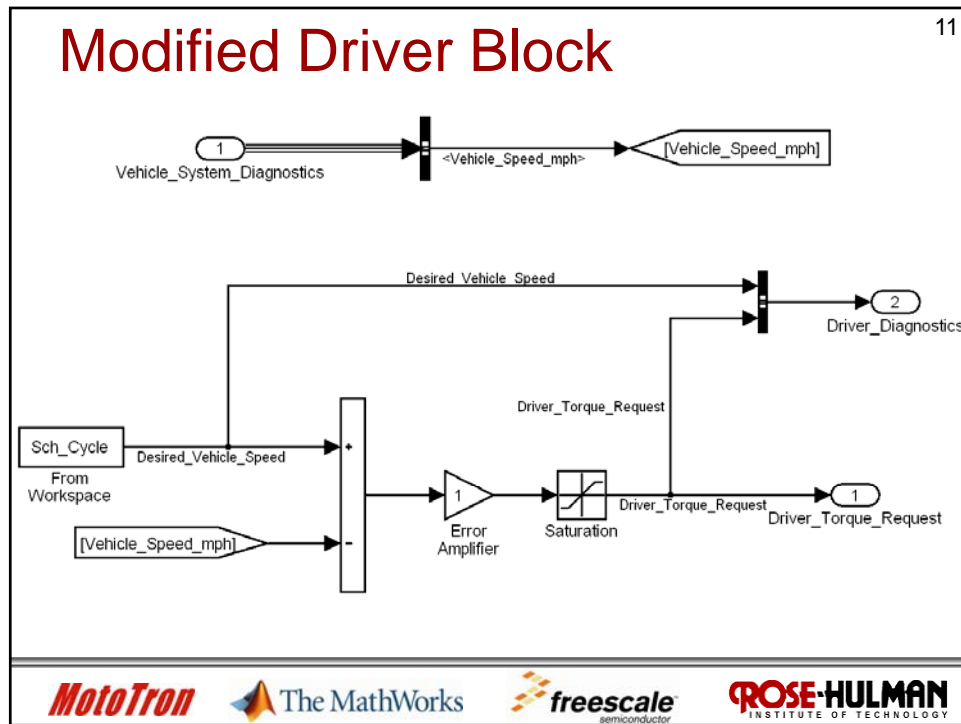


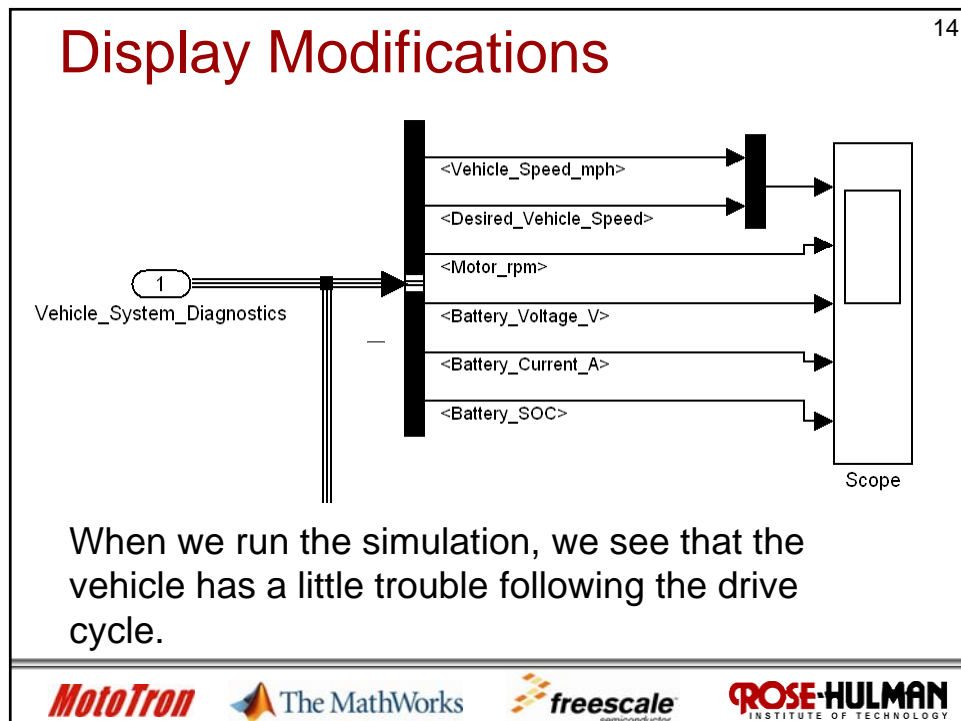
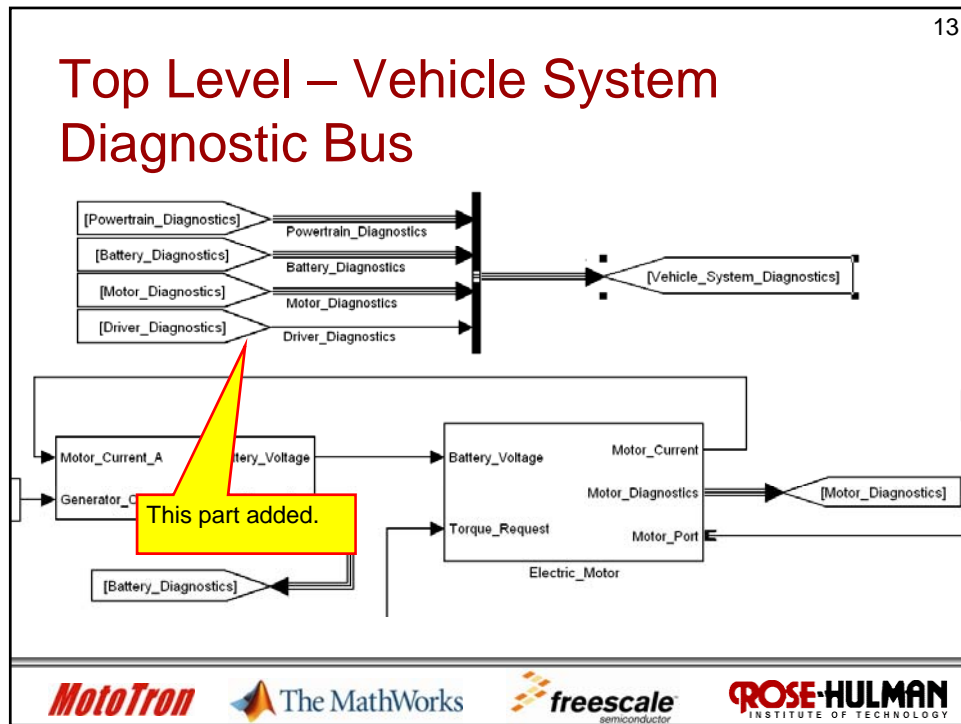
Following a Drive Cycle

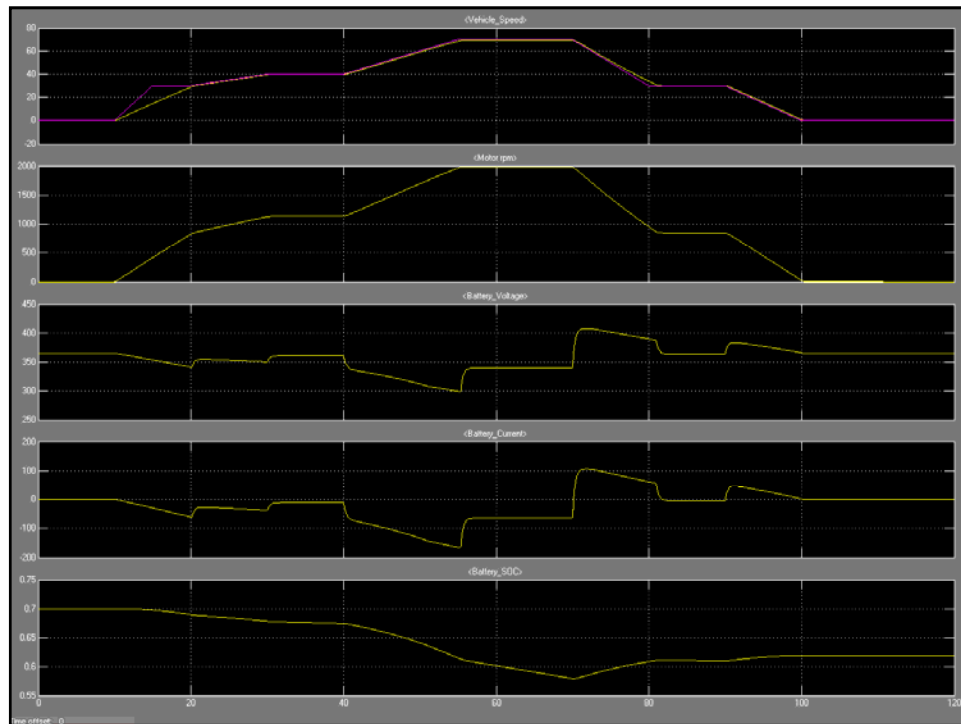
10

- One question we have is how close does the vehicle follow the drive cycle.
- We need to add a diagnostic output to the driver block to display the Sch_Cycle signal.
- We need to add this signal to the Vehicle System Diagnostic bus.
- We need to display this signal on the same plot as the vehicle speed.









Advanced Model-Based-System Design

Reading Drive Cycles in Excel

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Drive Cycles

17

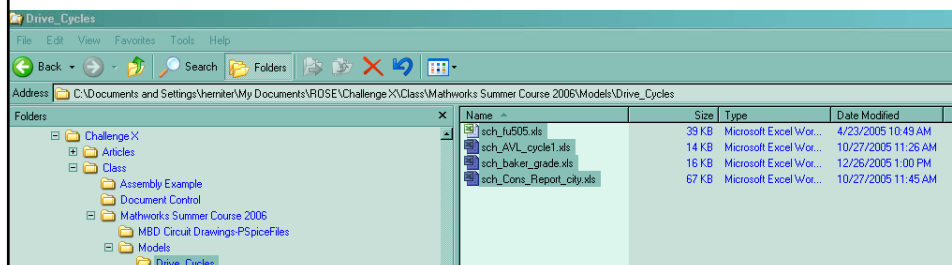
- Drive Cycles are available from several sources and in several different formats.
- We will use Excel to store our drive cycles since Excel can be used to easily modify the cycles.



Drive Cycles

18

- Create a directory called Drive_Cycles in your current working directory.
- We will keep all of our drive cycles in this directory.
- Copy the drive cycles that were provided for this class to this directory





19

AVL Drive Cycle

The Sch_Cycle tab contains the time and speed coordinates for the cycle.

	A	B	C
1	0	0	
2	5	0	
3	15	25	
4	35	25	
5	37	18	
6	57	18	
7	72	46	
8	92	46	
9	107	56	
10	127	56	
11	148	32	
12	167	0	
13	180	0	
14			

Drive Cycles

20

- The speed in the Sch_Cycle worksheet is in mph.
- For the moment, all we will use is the Sch_Cycle information for vehicle speed.
- The Excel file contains information for the brake pedal, gear selection, grade, and key on.



Drive Cycles

21

- As your model progresses, you may use some of the other signals.
- For now, we will read in all of the information, but we will only use the vehicle speed information.
- We will place the following code in our init file to read in the excel drive cycle file and store it in MATLAB variables.



22

```
%Read a drive cycle contained in an excel spreadsheet.
if (exist('fn') == 0)|(fn==0)
    fn='Drive_Cycles\sch_fu505.xls';
else
    fn=['Drive_Cycles\',fn];
end
[fn,pn]=uigetfile('Drive_Cycles\sch*.xls','Specify an Excel Schedule File Name',fn);
name=[pn,fn];
Sch_Cycle = xlsread(name, 'Sch Cycle');
Sch_Brake_on = xlsread(name, 'Sch Brake On');
Sch_Gear_on = xlsread(name, 'Sch Gear On');
Sch_Grade = xlsread(name, 'Sch Grade');
Sch_Key_on = xlsread(name, 'Sch Key On');
%Convert the grade from percent to radians.
Sch_Grade(:,2)=atan(Sch_Grade(:,2)/100);
```





23

```
if (exist('fn') == 0)|(fn==0)
    fn='Drive_Cycles\sch_fu505.xls';
else
    fn=['Drive_Cycles',fn];
end
```

If variable **fn** exists (the file name was previous selected), use the old name as the default file name.

If variable **fn** has not yet been defined, use file name sch_fu505.xls as the default file name.



24

```
[fn,pn]=uigetfile('Drive_Cycles\sch*.xls','Specify an Excel  
Schedule File Name',fn);  
name=[pn,fn];
```

Open a Windows style file name selection box. This function returns the file name and the path.

Concatenate the path and file name into one string.





25

```
Sch_Cycle = xlsread(name, 'Sch Cycle');  
Sch_Brake_on = xlsread(name, 'Sch Brake On');  
Sch_Gear_on = xlsread(name, 'Sch Gear On');  
Sch_Grade = xlsread(name, 'Sch Grade');  
Sch_Key_on = xlsread(name, 'Sch Key On');
```

Read individual worksheets into separate variables.



26

Drive Cycles

- Run this section of code.
- Select the AVL drive cycle.
- Display contents of variable Sch_Cycle.





27

Specify an Excel Schedule File Name

Look in: Drive_Cycles

- sch_AVL_cycle1.xls
- sch_baker_grade.xls
- sch_Cons_Report_city.xls
- sch_fu505.xls

File name: sch_fu505.xls

Files of type: sch*.xls

Specify an Excel Schedule File Name

Look in: Drive_Cycles

- sch_AVL_cycle1.xls
- sch_baker_grade.xls
- sch_Cons_Report_city.xls
- sch_fu505.xls

File name: sch_AVL_cycle1.xls

Files of type: sch*.xls

MotoTron **The MathWorks** **freescaler** **ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY

28

Contents of Variable Sch_Cycle

```
>> Sch_Cycle

Sch_Cycle =

    0    0
    5    0
   15   25
   35   25
   37   18
   57   18
   72   46
   92   46
  107   56
  127   56
  148   32
  167    0
  180    0

>>
```

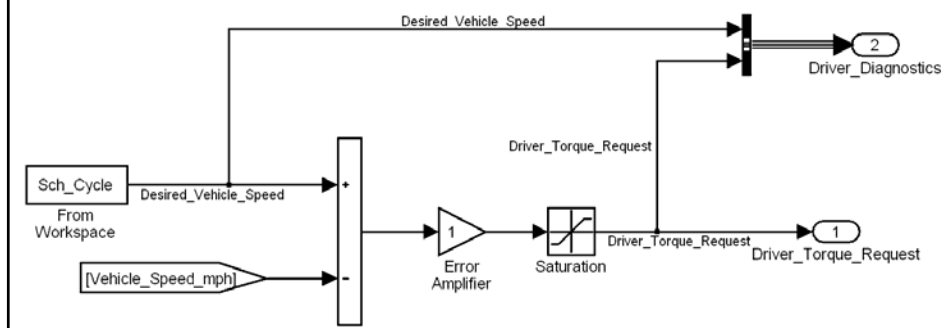
MotoTron **The MathWorks** **freescaler** **ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY



29

Drive Cycles

- If you remember, the Driver block uses the contents of variable Sch_Cycle as the drive cycle.
- We can now easily run predefined drive cycles.



30

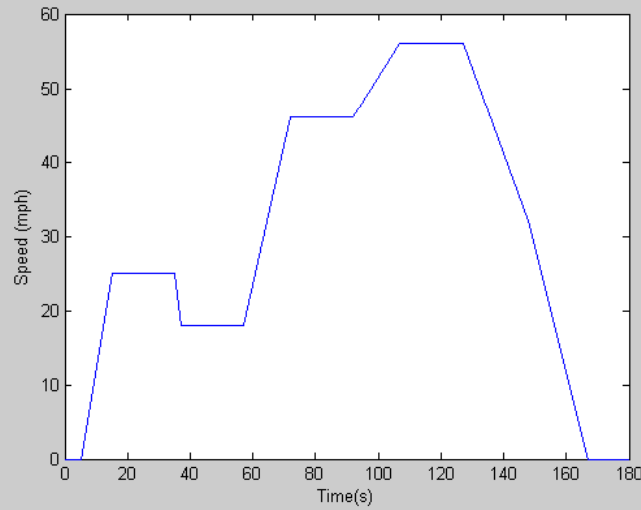
Drive Cycles

- Let's plot the drive cycles to see what they look like.
- Use the code below to plot the drive cycle.

```
plot(Sch_Cycle(:,1),Sch_Cycle(:,2));
xlabel('Time(s)');
ylabel('Speed (mph)');
```

AVL Drive Cycle

31



MotoTron

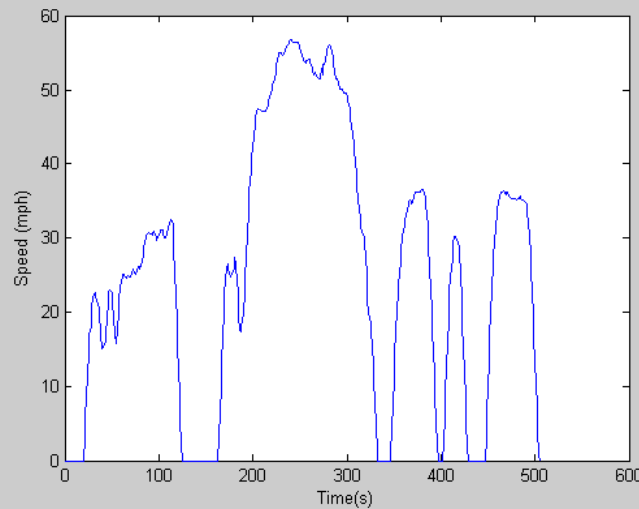
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

FU505 Drive Cycle

32



MotoTron

The MathWorks

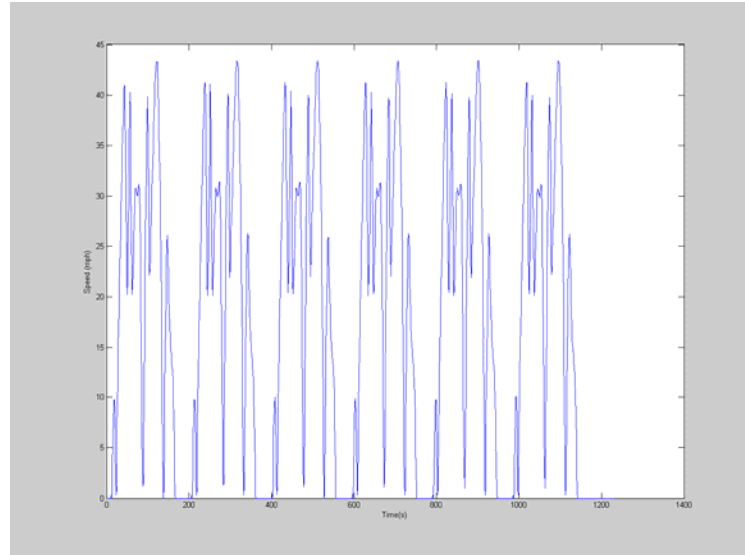
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



33

Consumer Reports City



34

Drive Cycles

- Each cycle runs for a different length of time.
- We would like to automatically specify that a simulation runs for the length of the cycle.
- Use the command below to obtain the last time point in the cycle:

```
Sch_Cycle(end,1);
```





Drive Cycles

35

- This command can be used as the stop time in the Configuration Parameters dialog box.
- Select **Simulation** and then **Configuration Parameters** from the menus, and enter the command as shown:

MotoTron

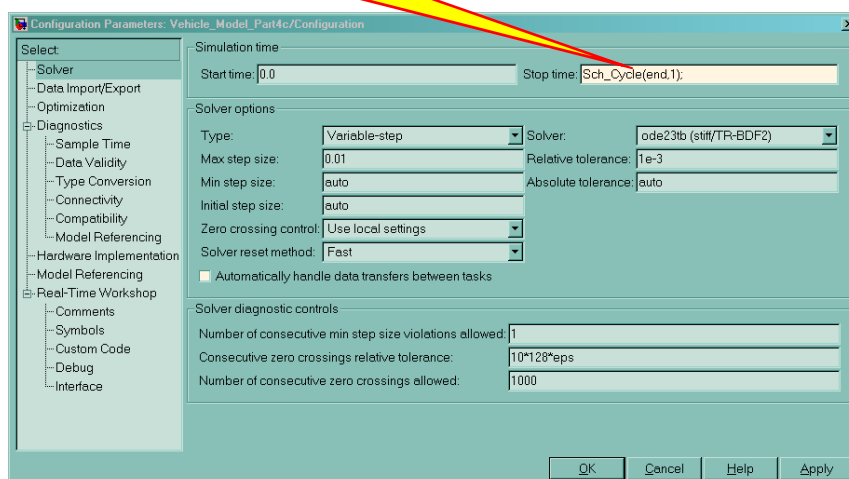
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

36

The last time point in the drive cycle will be used as the ending time for the simulation.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Automatically Run Init File

37

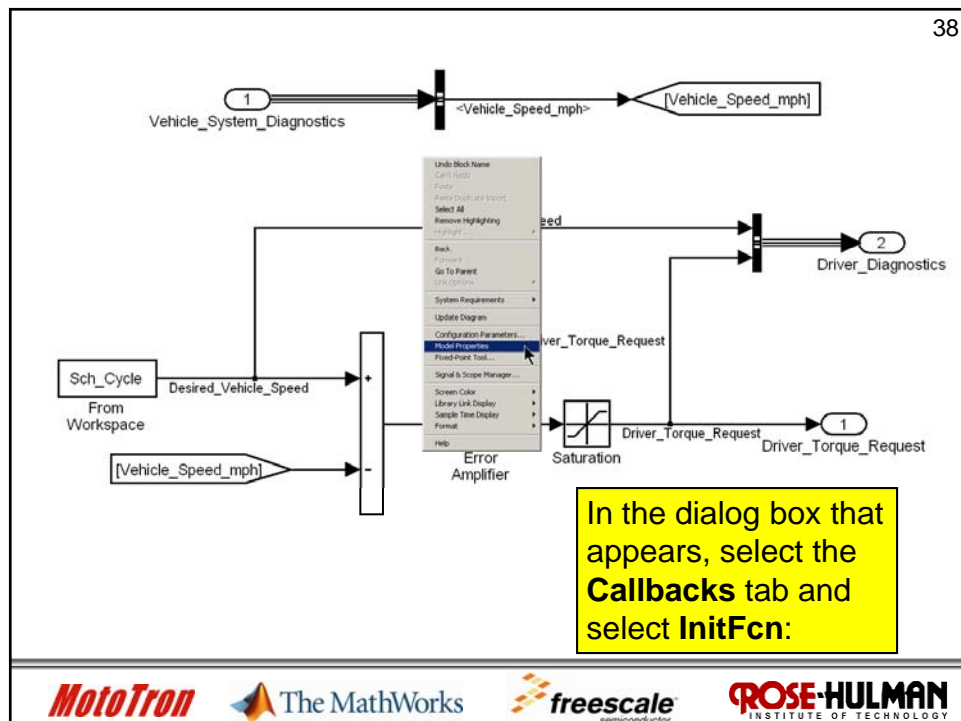
- We want to run the init file every time we run our model.
- We can do this in Simulink using a Call Back function.
- Open the model and right-click on some empty space in the model and select **Model Properties** from the menu:

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTron

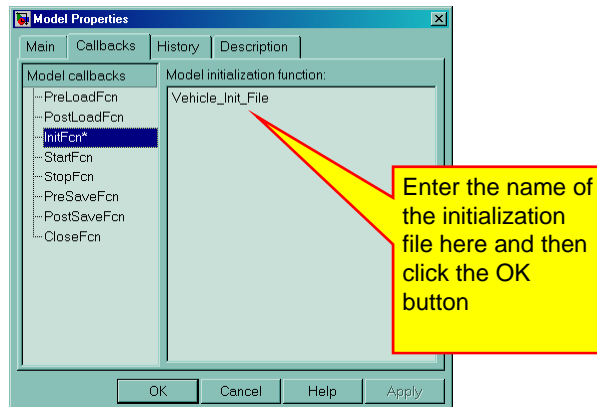
 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



39



With this setting, our init file will run each time we run a simulation.



40

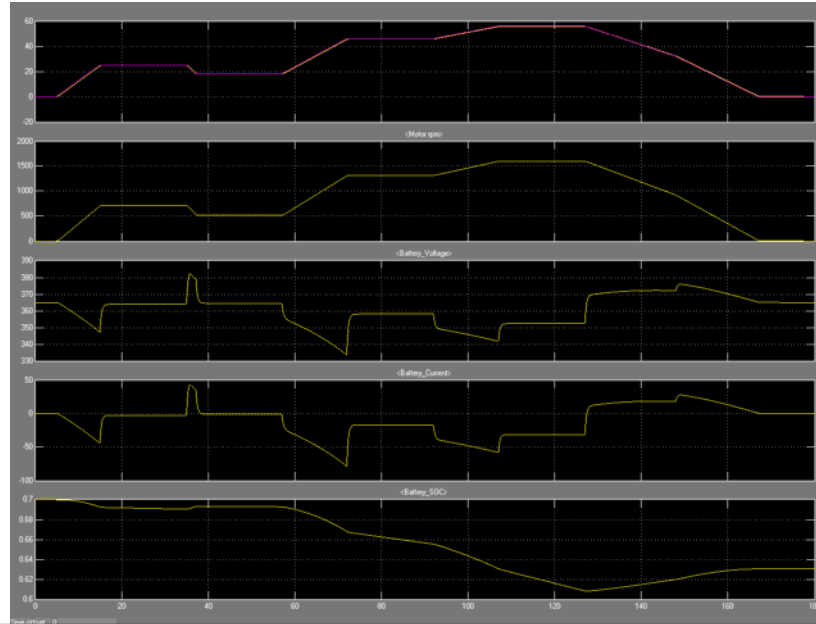
Running a Simulation

- When we run a simulation, it will automatically run the init file which will
 - Define all of our model parameters.
 - Read in a drive cycle.
- Run a simulation and make sure that everything works



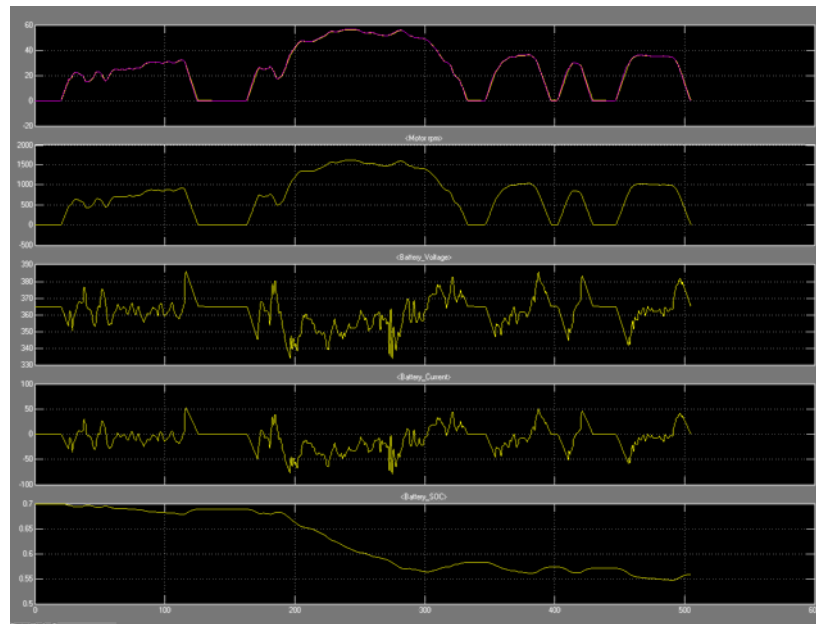
AVL Drive Cycle Results

41



FU505 Drive Cycle Results

42





Lecture 3 Exercise 1

43

- You may notice that it takes a long time to read Excel files.
- To shorten the time it takes to run the initialization file, we would like to store and read drive cycles as MATLAB .mat files.
- Part 1
 - Write an .m file that asks the user to select an excel drive cycle, reads the variables from that file, and saves the variables in a .mat file with the same name.



Lecture 3 Exercise 1

44

- Part 1: Write an .m file that:
 - Clears all variables from the MATLAB workspace.
 - Asks the user to select an excel drive cycle and reads information in the file and stores the data with the same names as used when reading drive cycles with Excel.
 - Clears variables fn and pn from the workspace.
 - Saves the drive cycle variables in a .mat file with the same name as the excel file except with a .mat extension..





Lecture 3 Exercise 1

45

- Part 2:
 - Create a new init file that is the same as the original init file except that the drive cycle is read as a .mat file rather than an excel file.
 - Using drive cycle “sch_fu505 ten times.xls”, compare the time it takes MATLAB to complete each of your init files.
 - Use the tic and toc functions to see how long it takes to run each script file.

Demo_____



Vehicle Modeling

46

- At this point we have
 - An electric vehicle model.
 - The structure to make a more complicated, detailed, and accurate model.
- We could head off in several directions
 - Add an engine.
 - Charge the battery.
 - Other





Model Based Design

47

- The important thing is to add detail slowly and verify the accuracy of our models as we add detail.
- We will do the following:
 - Add detail to the motor model.
 - Add detail to the battery model.



Electric Motor

48

- Add a torque curve.
- Make the motor less than 100% efficient.





Torque Curve

49

- Obtain the torque versus rpm curve for your motor from the manufacturer or use measured data.
- Place the data in an excel file.
- Read the excel files into the MATLAB workspace.
- Use a table lookup to use the data in your model.



Vendor Supplied Torque Curve

50

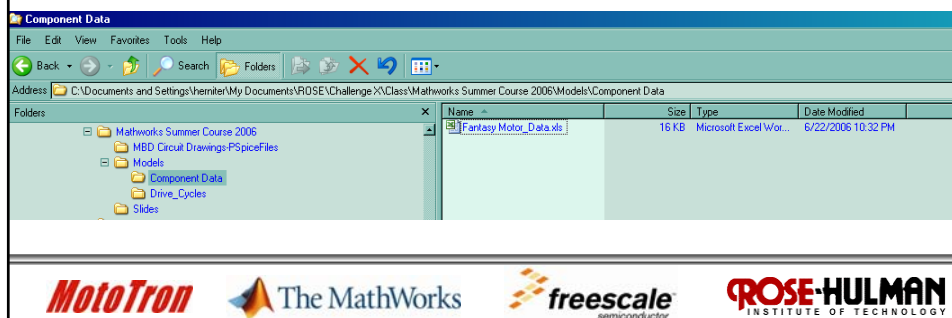
Rpm	Torque (Nm)
0	370
1200	361
1400	319
1600	260
1700	242
2000	190
2200	170
2400	128
2600	120
3000	79
3500	60
4000	51
4500	40
5000	30
6000	10
7000	0



51

Component Data

- We will place this data in an excel file and read it using the xlsread function.
- We will place all component data files in a directory called “Component Data.”



52

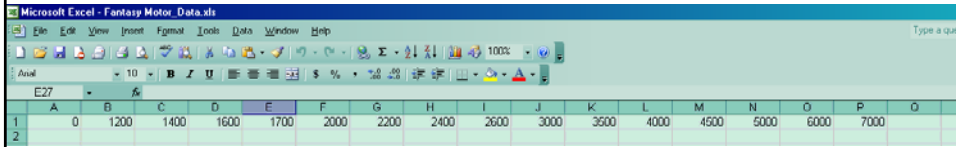
Motor rpm Data

- We will place the rpm data and the torque data in separate worksheets.
- This is not required. It just makes it easier to split up the two parts of the table.
- The two worksheets are shown next:



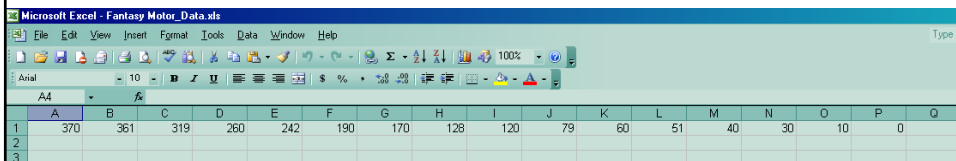
53

Rpm data. This worksheet was named Max_Torque_rpm_Axis.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0	1200	1400	1600	1700	2000	2200	2400	2600	3000	3500	4000	4500	5000	6000	7000	
2																	

Torque data. This worksheet was named Max_Torque.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	370	361	319	260	242	190	170	128	120	79	60	51	40	30	10	0	
2																	
3																	

We will read this data using the xlsread function in the init file.

Add the following lines to you init file.

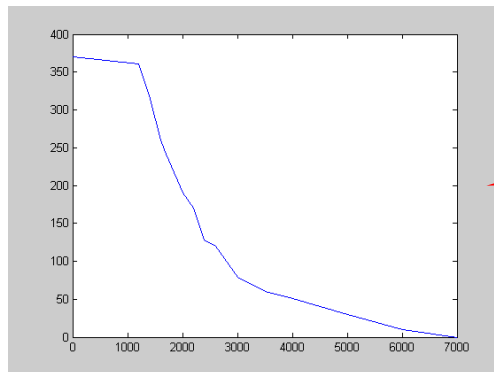
54

```

Motor_Name = 'Fantasy Motor_Data.xls';
Motor_PN = ['Component Data\',Motor_Name];
motor_max_torque = xlsread(Motor_PN,'Max_Torque');
motor_max_torque_rpm_axis =xlsread(Motor_PN,'Max_Torque_rpm_Axis');
    
```

You can plot the torque curve using the command

```
plot(motor_max_torque_rpm_axis, motor_max_torque)
```



Generate this plot to verify that you can read your excel file.

Motor Model

55

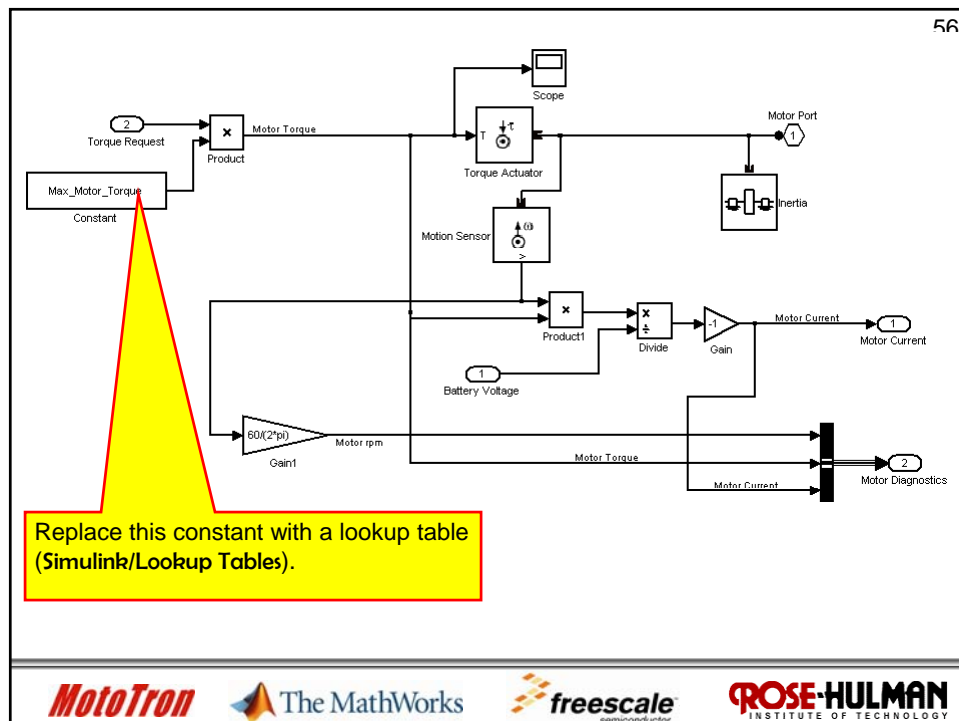
- Now that we have our motor torque curve available as workspace variables, we can use that data in our model.
- We will use a 1-D look up table.
- Our motor model presently has a constant torque curve:

MotoTron

The MathWorks

freescale
semiconductor

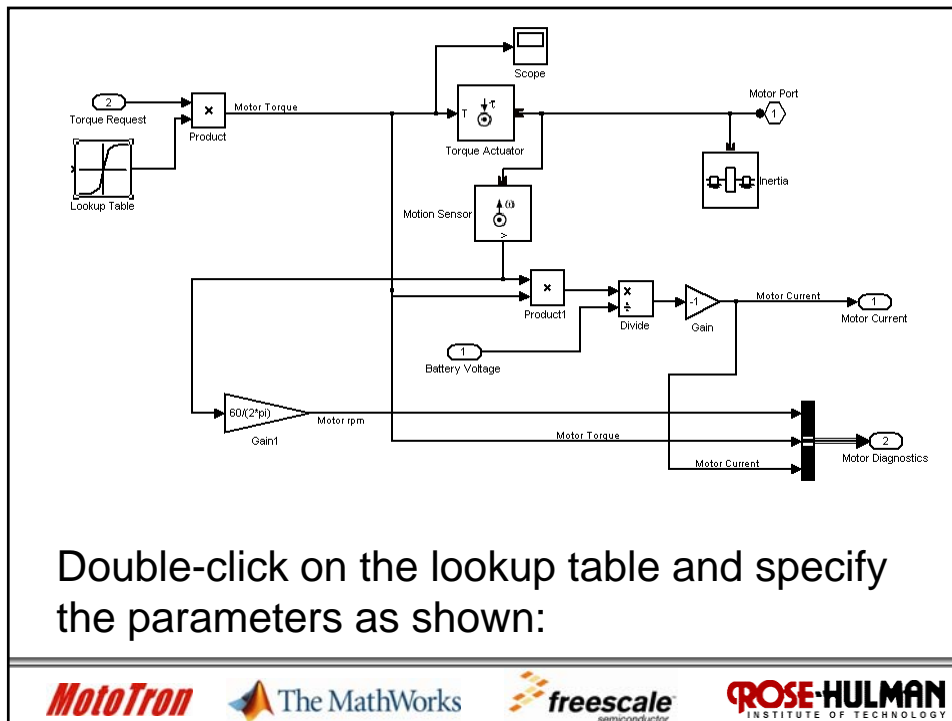
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Except where otherwise noted, this work is licensed under

<http://creativecommons.org/licenses/by/3.0/>



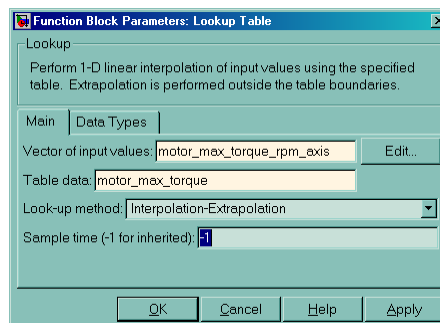
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

58



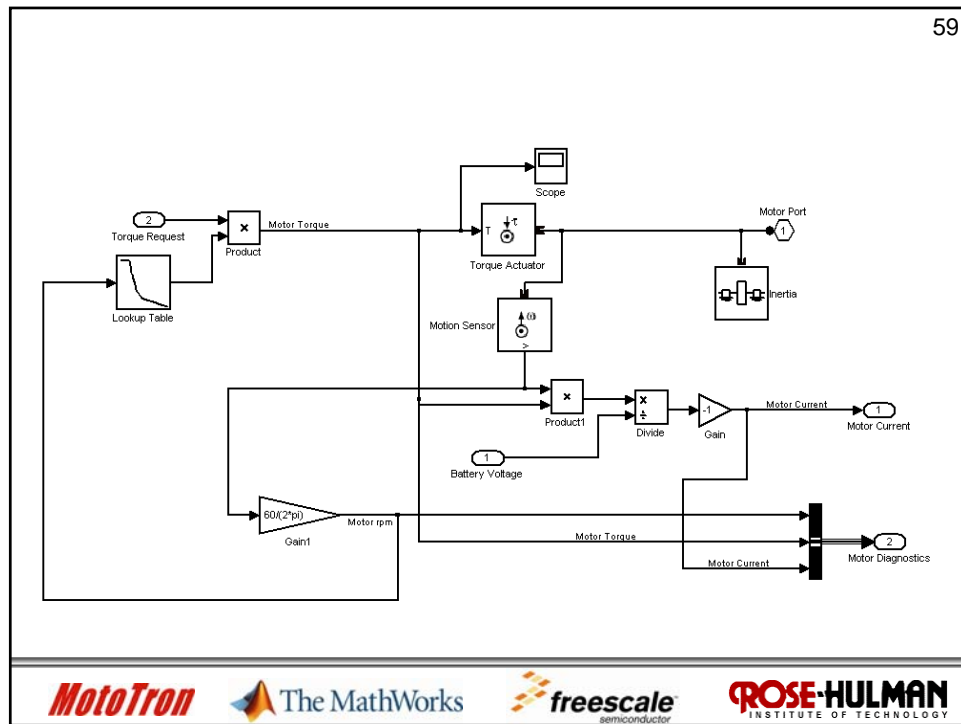
- The output of the part (Table data) is the motor torque.
- The input to this part is the motor rpm. We already calculate the rpm in the model, so the connection is easy to make.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Lecture 3 Exercise 2

60

- The motor model has a problem in that the effective torque we request changes as the available torque goes down. The effectively changes the loop gain of our system and the motor rpm changes.
- We want to modify the model so that the torque request is always the maximum motor torque times the driver torque request. If the available motor torque is less than the torque request, then the available motor torque is used. If the available motor torque is greater than the torque request, then the torque requested is used.

Demo_____



Advanced Model-Based-System Design

Lecture 4: Advanced Models



Vehicle Modeling

2

- At this point we have
 - An electric vehicle model.
 - The structure to make a more complicated, detailed, and accurate model.
- We could head off in several directions
 - Add an engine.
 - Charge the battery.
 - Other





Model Based Design

3

- The important thing is to add detail slowly and verify the accuracy of our models as we add detail.
- We will do the following:
 - Add detail to the motor model.
 - Add detail to the battery model.



Electric Motor

4

- Add a torque curve.
- Make the motor less than 100% efficient.





Torque Curve

5

- Obtain the torque versus rpm curve for your motor from the manufacturer or use measured data.
- Place the data in an excel file.
- Read the excel files into the MATLAB workspace.
- Use a table lookup to use the data in your model.



Vendor Supplied Torque Curve

6

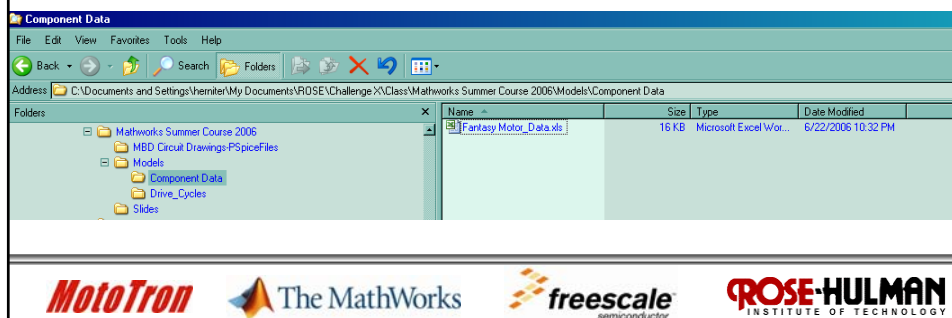
Rpm	Torque (Nm)
0	370
1200	361
1400	319
1600	260
1700	242
2000	190
2200	170
2400	128
2600	120
3000	79
3500	60
4000	51
4500	40
5000	30
6000	10
7000	0



7

Component Data

- We will place this data in an excel file and read it using the xlsread function.
- We will place all component data files in a directory called “Component Data.”



MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

8

Motor rpm Data

- We will place the rpm data and the torque data in separate worksheets.
- This is not required. It just makes it easier to split up the two parts of the table.
- The two worksheets are shown next:

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



9

Rpm data. This worksheet was named Max_Torque_rpm_Axis.

RPM	Torque
0	370
1200	361
1400	319
1600	260
1700	242
2000	190
2200	170
2400	128
2600	120
3000	79
3500	60
4000	51
4500	40
5000	30
6000	10
7000	0

Torque data. This worksheet was named Max_Torque.

RPM	Torque
0	370
1200	361
1400	319
1600	260
1700	242
2000	190
2200	170
2400	128
2600	120
3000	79
3500	60
4000	51
4500	40
5000	30
6000	10
7000	0

We will read this data using the xlsread function in the init file.

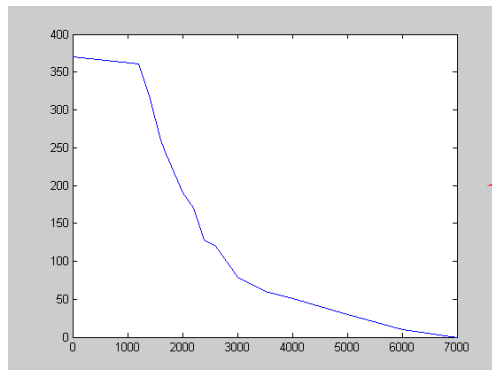
Add the following lines to you init file.

10

```
Motor_Name = 'Fantasy Motor_Data.xls';  
Motor_PN = ['Component Data\',Motor_Name];  
motor_max_torque = xlsread(Motor_PN,'Max_Torque');  
motor_max_torque_rpm_axis =xlsread(Motor_PN,'Max_Torque_rpm_Axis');
```

You can plot the torque curve using the command

```
plot(motor_max_torque_rpm_axis, motor_max_torque)
```



Generate this plot to verify that you can read your excel file.

Motor Model

11

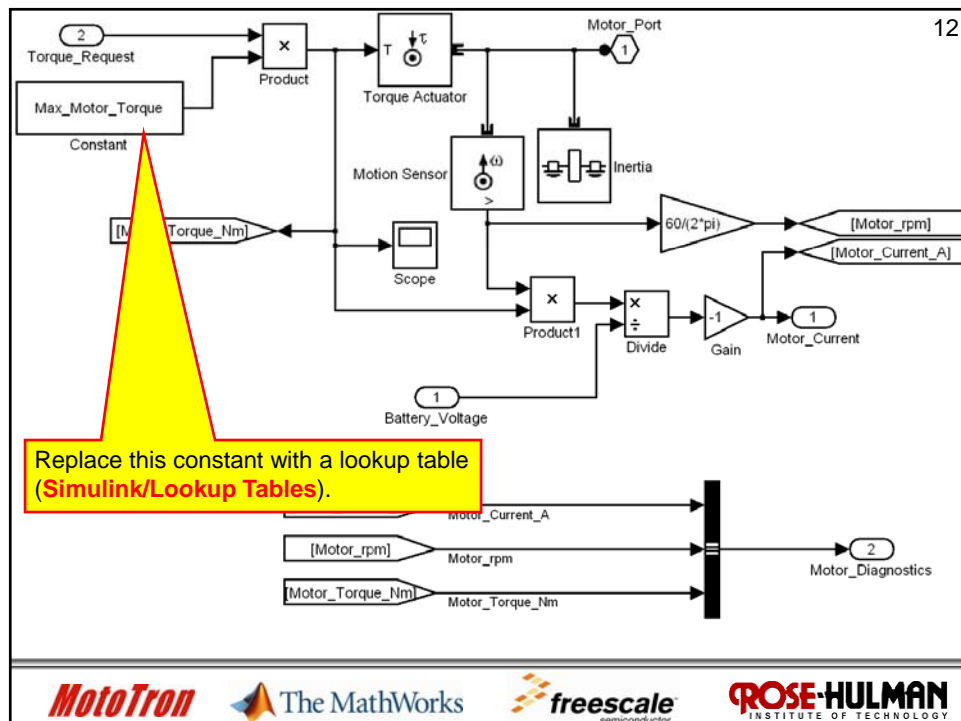
- Now that we have our motor torque curve available as workspace variables, we can use that data in our model.
- We will use a 1-D look up table.
- Our motor model presently has a constant torque curve:

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

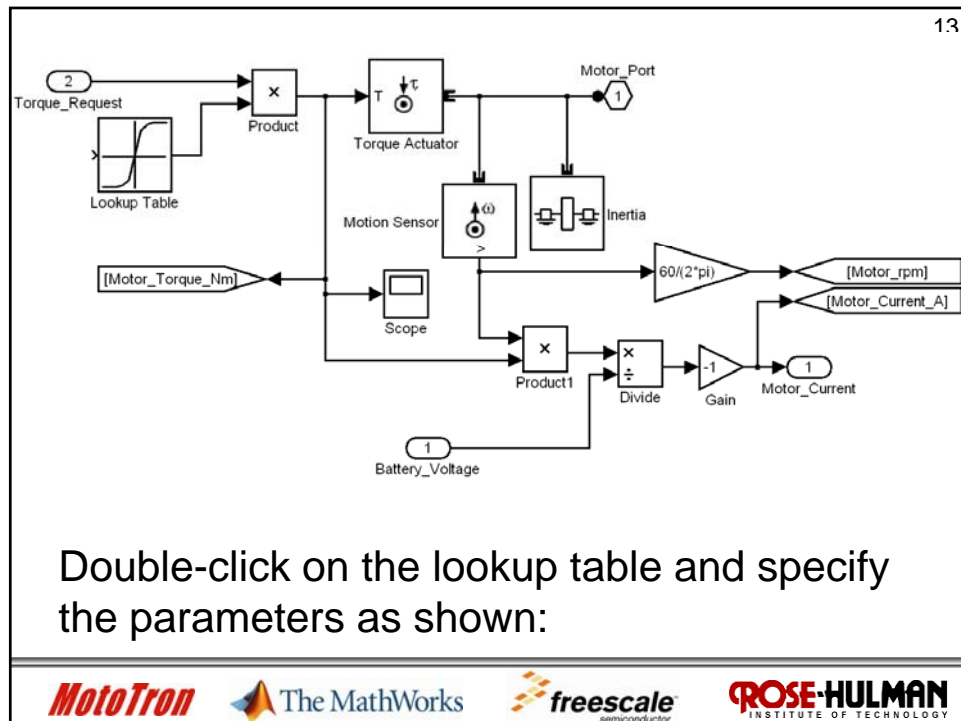


MotoTron

The MathWorks





freescale
semiconductor

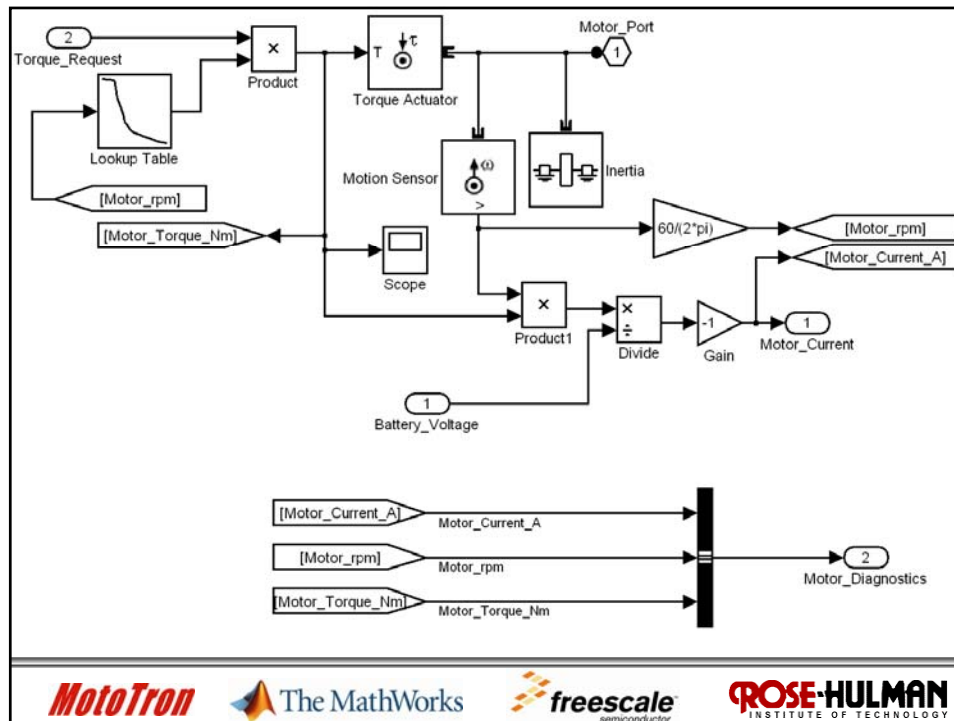
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



14

- The output of the part (Table data) is the motor torque.
- The input to this part is the motor rpm. We already calculate the rpm in the model, so the connection is easy to make.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Lecture 4 Exercise 1

16

- The motor model has a problem in that the effective torque we request changes as the available torque goes down. The effectively changes the loop gain of our system and the motor rpm changes.
- We want to modify the model so that the torque request is always the maximum motor torque times the driver torque request. If the available motor torque is less than the torque request, then the available motor torque is used. If the available motor torque is greater than the torque request, then the torque requested is used.

Demo_____



Lecture 4 Exercise 2

17

- This motor has other problems:
 - What happens if the motor rpm is negative? Surely we want to use the vehicle in reverse.
 - What happens if the motor rpm exceeds the max rpm specified in the table?
- Fix the model so that the model works for negative values of rpm and that nothing catastrophic happens if the input rpm exceeds the max specified in the data file.
- You are not allowed to modify the data in the Excel file.

Demo _____

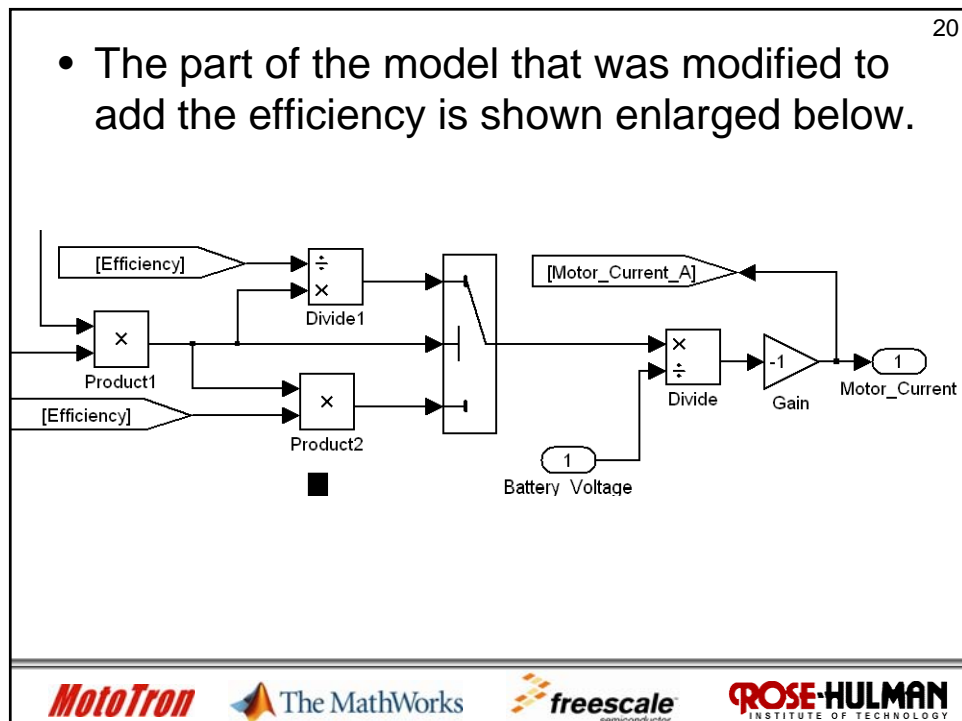
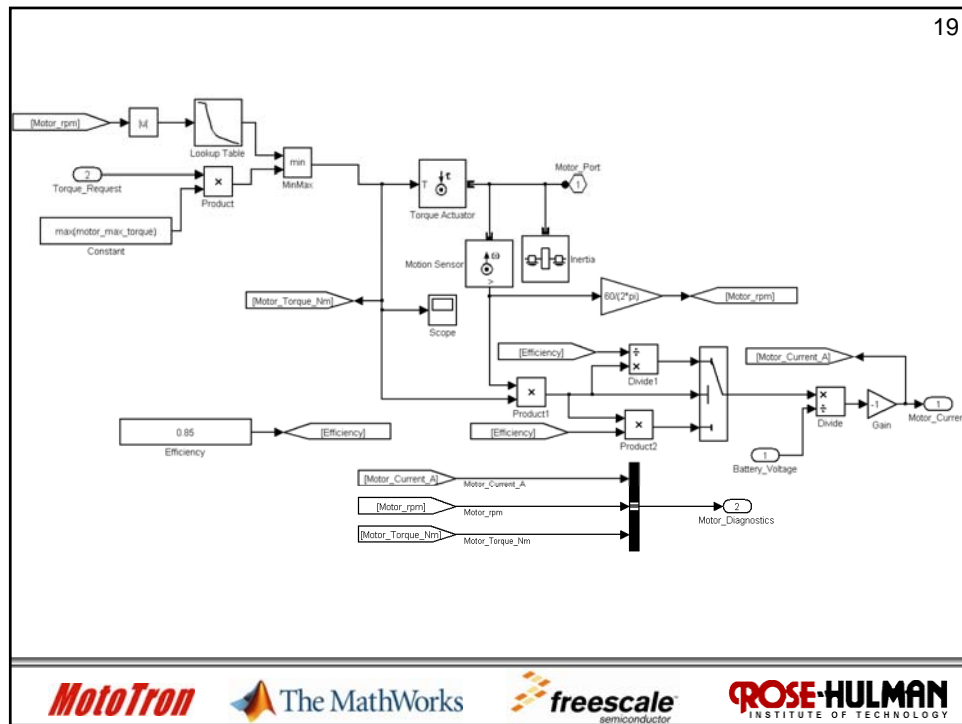


Motor Efficiency

18

- Next we will add efficiency to the motor.
- When acting as a motor, the mechanical power output is less than the electrical power input.
- When acting as a generator, the electrical power output is less than the mechanical power input.
- In our implementation, the output torque will be specified, and the corresponding electrical power will be calculated including efficiency.
- We will start with a constant efficiency of 85%.





Model Verification

21

- To see that this model behaves the way we think it should, we need to add a few diagnostic signals.
- Create signals for the mechanical and electrical power. These two signals should be related by the efficiency.
- Add the electrical and mechanical power signals to the diagnostic bus.
- Modify the model as shown.

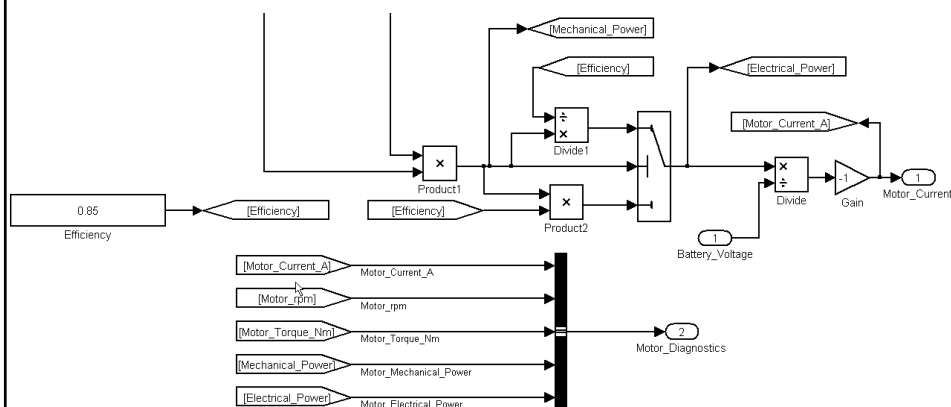
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

22



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Model Verification

23

- Run a simulation and plot
 - Vehicle Speed
 - Mechanical Power
 - Electrical Power
- Verify that when
 - The vehicle speeds up, the mechanical power is less than the electrical power.
 - When the vehicle slows down, the electrical power is less than the mechanical power.
 - We will create this plot using the signal and scope manager.

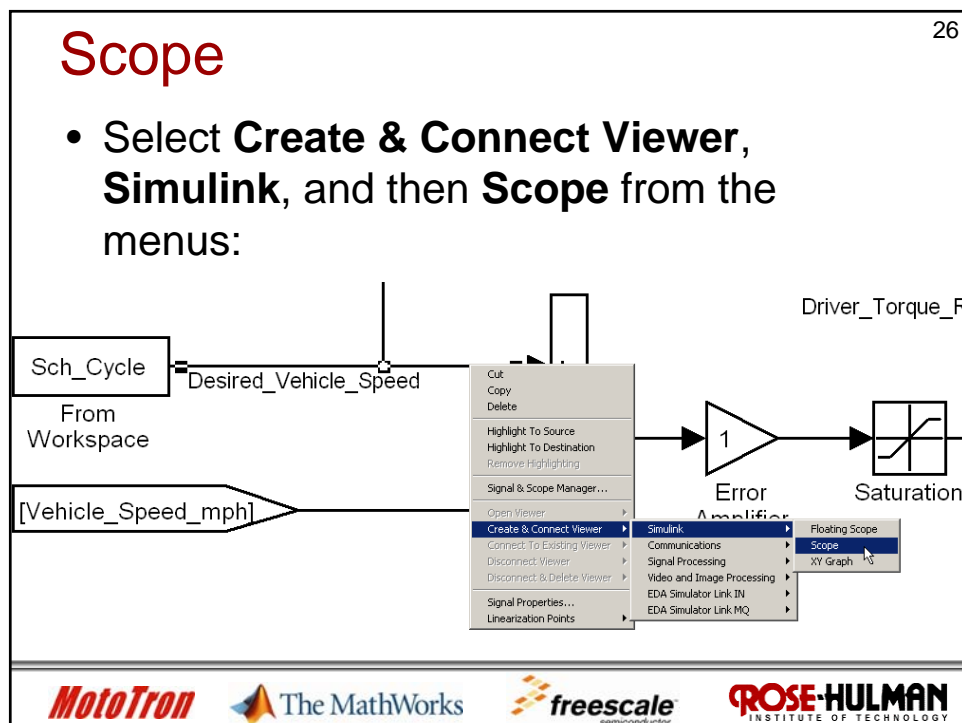
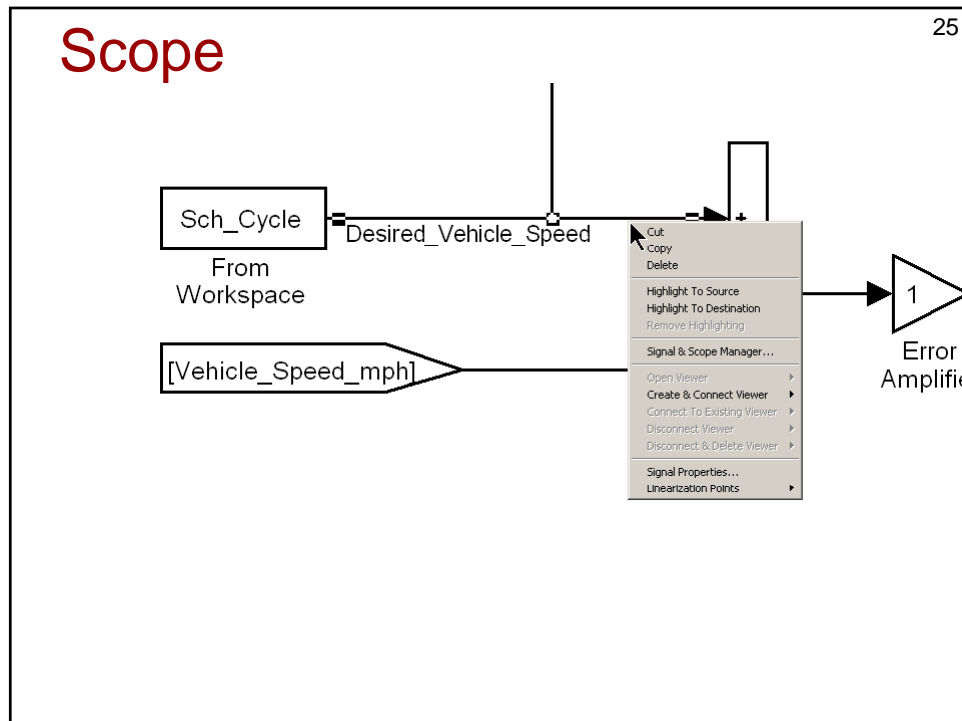


Scope

24

- We need to create a viewer and attach the signals we wish to display.
- We want the vehicle speed to be displayed on the top plot.
- Open the driver block and right click on the “Desired_Vehicle_Speed” signal line:
- Right click on the Desired_rpm signal line.

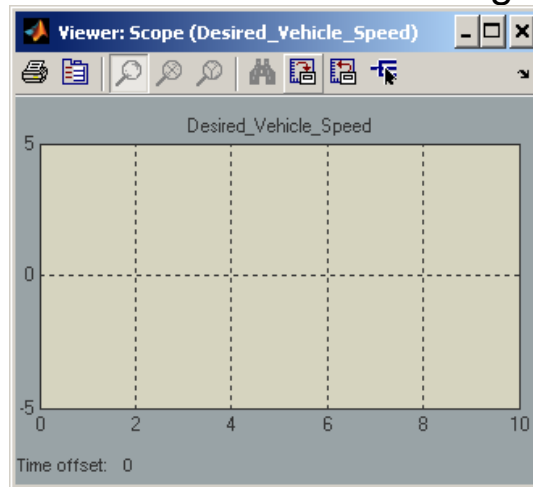




Scope

27

- A scope will be created with a single plot:



MotoTron

 The MathWorks

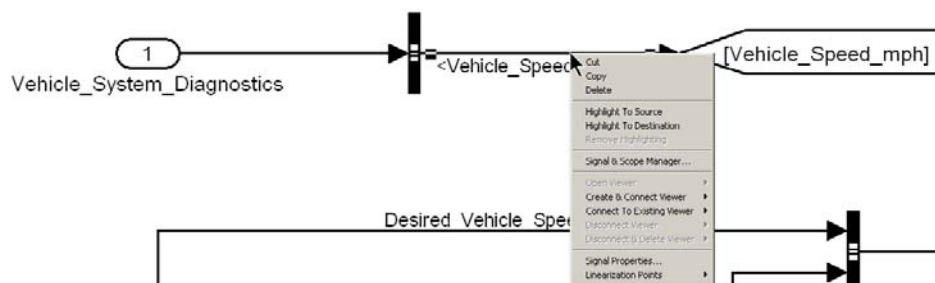
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Scope

28

- Now we need to display the actual rpm on the same scope and on the same axis.
- Right click on the Vehicle_Speed_mph signal line:



MotoTron

 The MathWorks

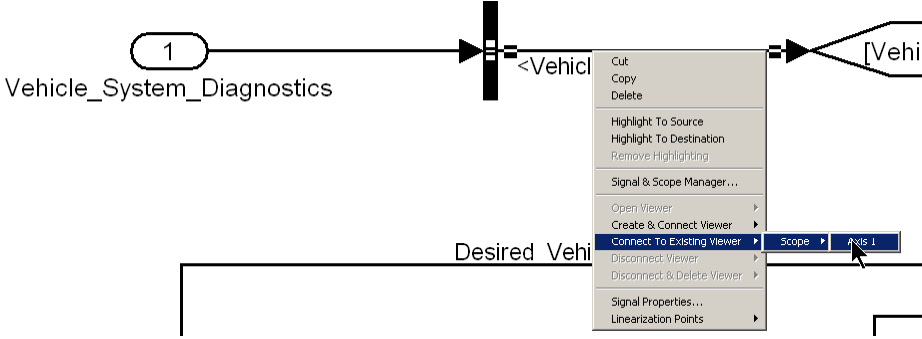
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Scope

29

- Select **Connect to Existing Viewer**, then select **Scope**, then select **Axis1**.



Vehicle_System_Diagnostics

Desired Vehi

<Vehicle

[Vehicle

Scope

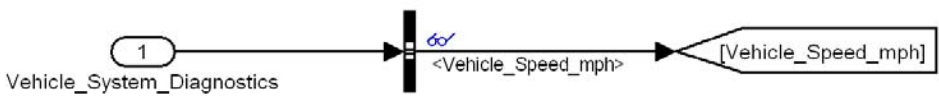
Axis 1

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Scope

30

- You will notice little glasses on your model.
- These glasses indicate that the associated signal is being display on a scope.



Vehicle_System_Diagnostics

<Vehicle_Speed_mph>

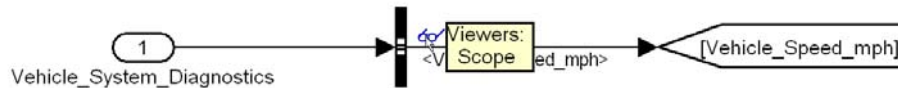
[Vehicle_Speed_mph]

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Scope

31

- If you hover the mouse pointer over the glasses, a box will appear and display the name of the viewer and the axis to which the signal is connected:



- In this case, the name of the viewer is "Scope."

MotoTron

The MathWorks

freescale
semiconductor

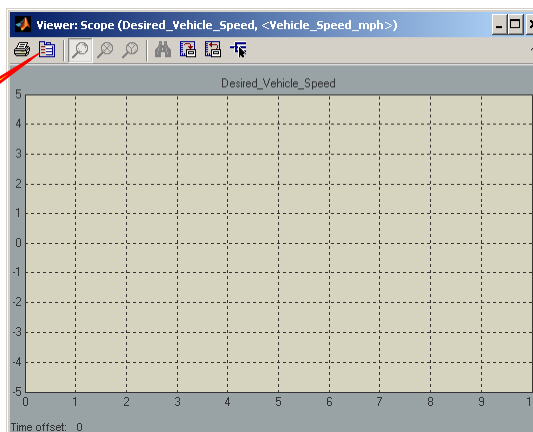
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Scope

32

- Next, we want to display two plots on the scope window.
- Left-click on the Parameters button in the scope menu:

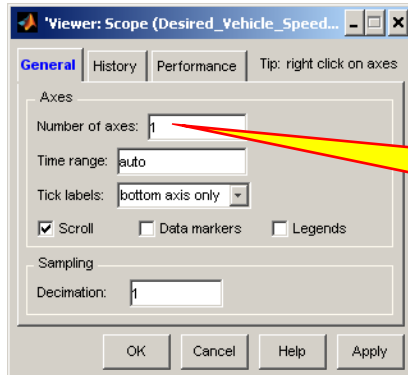
Click here.





Scope

33



Change this to 2 to display 2 plots in the Scope window.



Scope

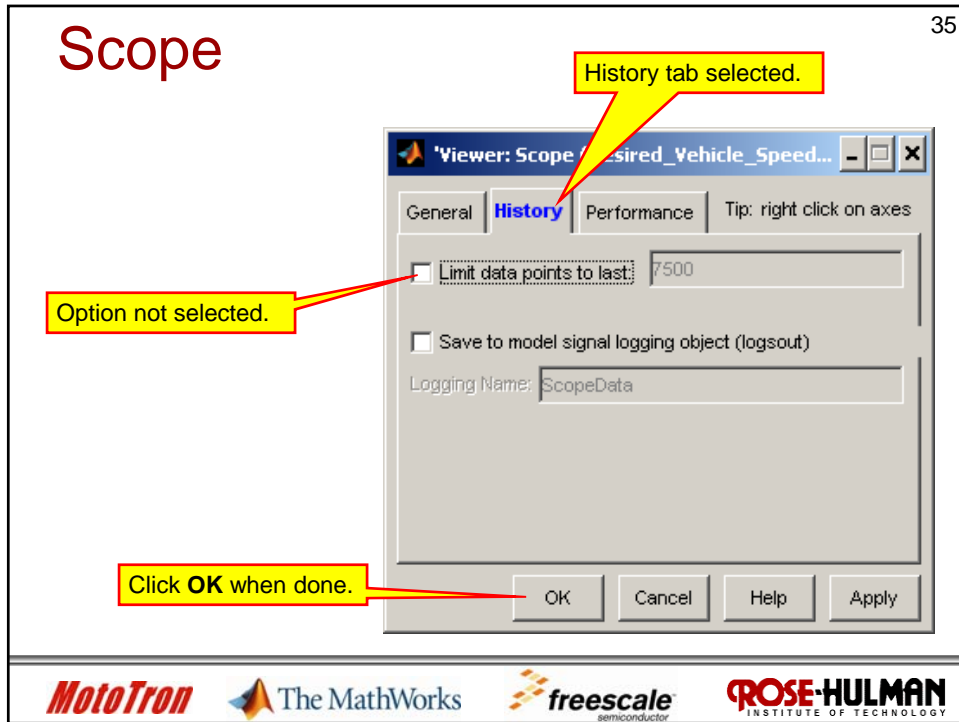
34

- Typically a scope will only display the last 7500 points of a simulation.
- We are not sure how many points our simulation will have, so we will change this setting.
- Select the **History** tab and uncheck the option as shown.



Scope

35



History tab selected.

Option not selected.

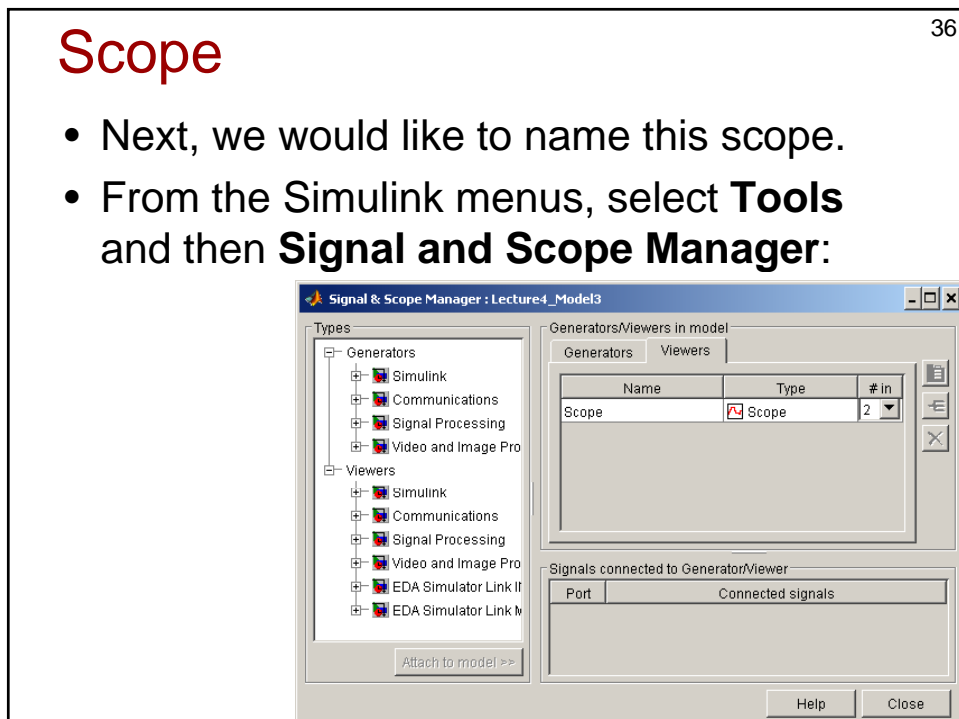
Click OK when done.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Scope

36

- Next, we would like to name this scope.
- From the Simulink menus, select **Tools** and then **Signal and Scope Manager**:



Name	Type	# In
Scope	Scope	2

Attach to model >>

Help Close

Signal & Scope Manager

37

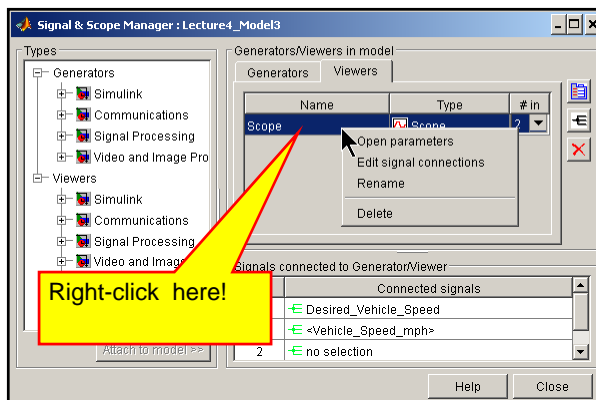
- The Signal and Scope Manager allows us to:
 - Rename scopes
 - Change the number of plots on a scope.
 - Add and delete signals displayed on a scope.
 - Delete Scopes
- Right-click on the text Scope to see the options you have in manipulating scopes

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



38

- Open parameters – opens the scope window.
- Edit signal connects allows you to select signals to display on the scope.

MotoTron

 The MathWorks

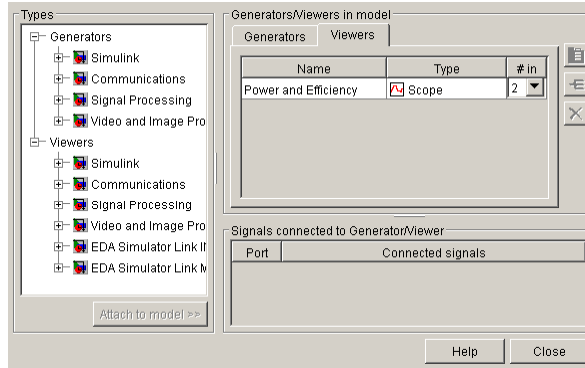
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Signals & Scope Manager

39

- We just want to rename the scope, so select Rename and change the name to “Power and Efficiency.”
- This is all we will do with the Signal and Scope Manager, so click the **Close** button.



MotoTron

The MathWorks

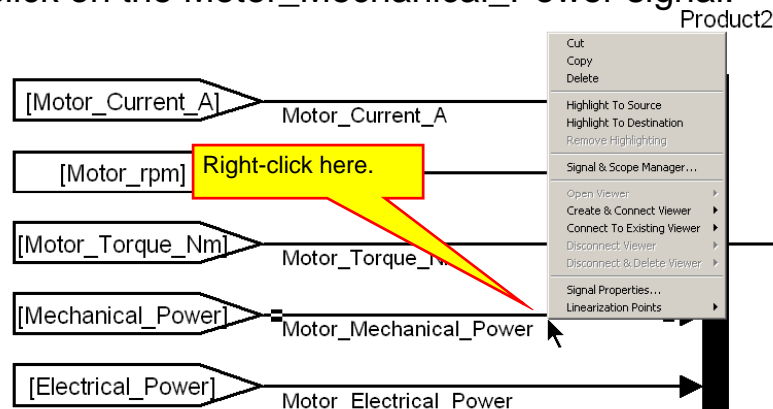
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Scope

40

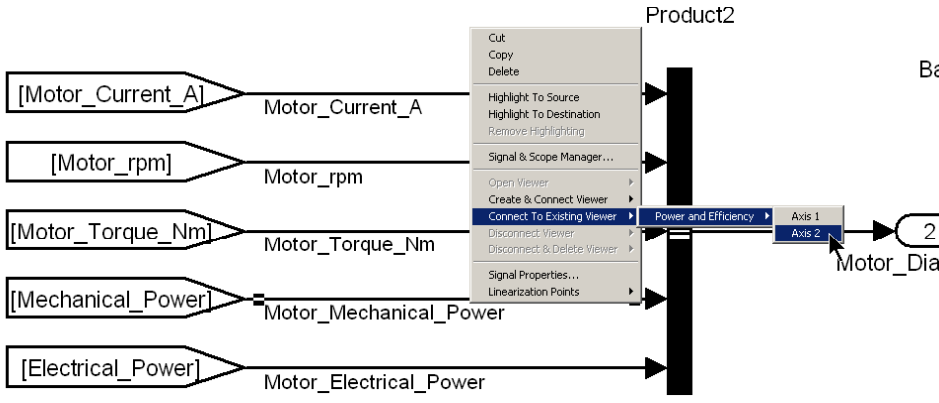
- Next, we want to display the electrical and mechanical power signals from the motor.
- Open the Electric_Motor subsystem.
- Right-click on the Motor_Mechanical_Power signal:







41

Scope

- Select **Connect to Existing Viewer**, then **Power and Efficiency**, and then **Axis 2**:







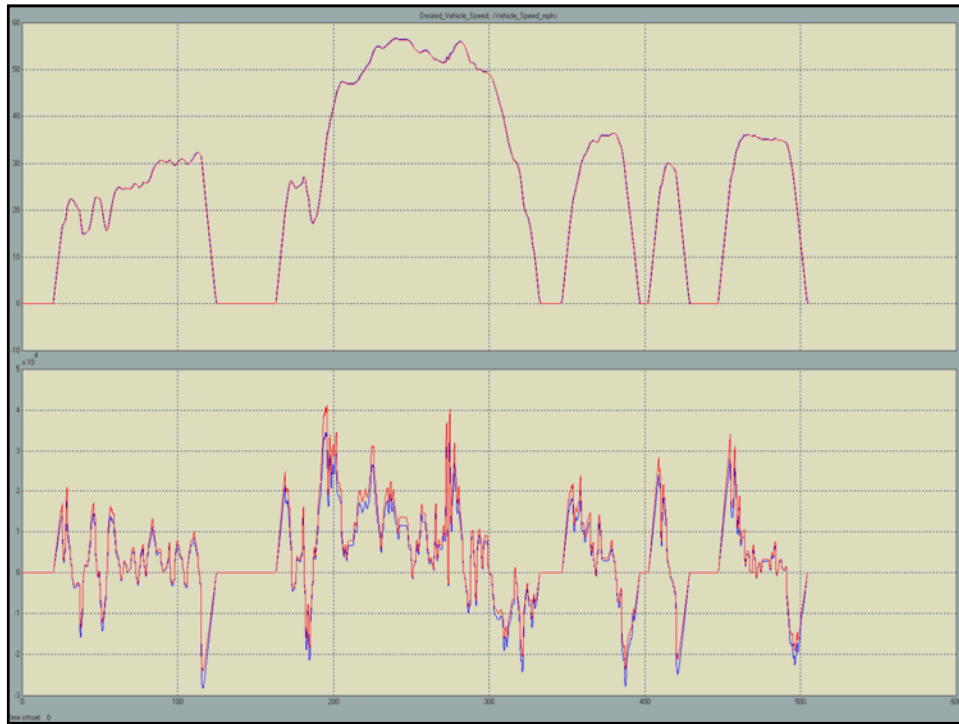





42

Scope

- The Motor_Mechanical_Power will now be displayed on the second plot in the Power and Efficiency viewer.
- Repeat the process to display the Electrical power on the same axis as the mechanical power.
- Run the FU505 drive cycle and display the results.





Lecture 4 Exercise 3

45

- In our plot, we see that the label for the electrical and mechanical power are not displayed in the Scope window.
- Fix this problem.

Demo _____



Model Verification

46

- The model appears to behave correctly for motoring and regen.
- Next, instead of having a constant efficiency, we will make the efficiency a function of motor rpm and motor current.
- We will do this with a 2-D look up table.
- First, obtain the efficiency data from the manufacturer or measure the efficiency.





47

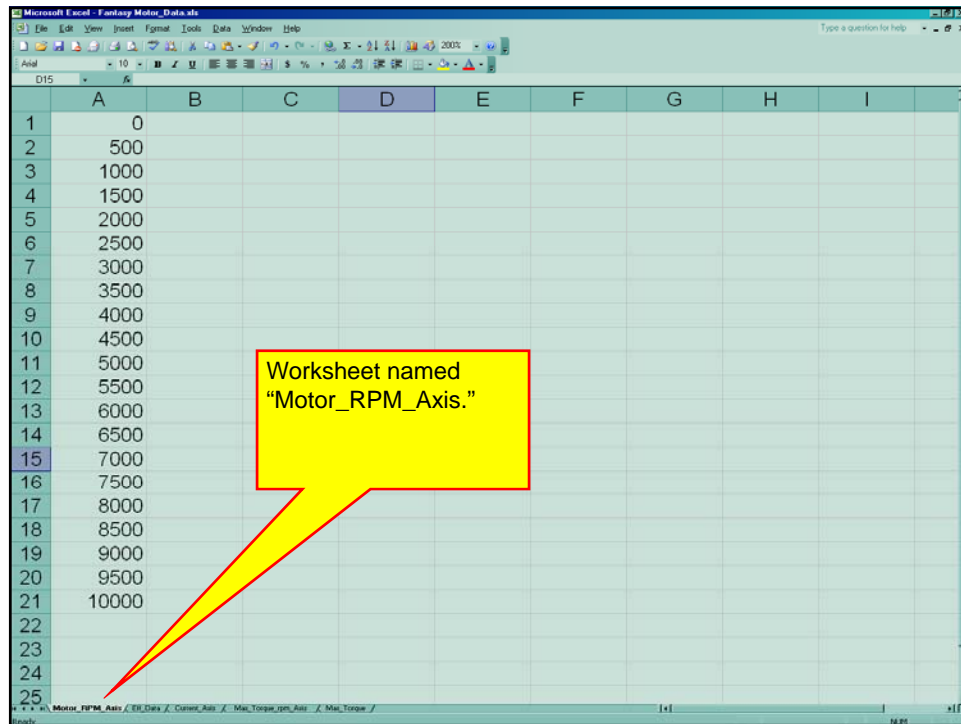
Manufacturer Supplied Efficiency Data

		Current (Amps)									
		6	50	94	138	182	226	270	314	358	402
r p m	0	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
	500	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
	1000	0.82	0.92	0.91	0.89	0.88	0.86	0.84	0.83	0.81	0.80
	1500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	2000	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	2500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	3000	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	3500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	4000	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	4500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	5000	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	5500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	6000	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	6500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	7000	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	7500	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
	8000	0.74	0.92	0.92	0.92	0.91	0.90	0.89	0.88	0.87	0.86
	8500	0.71	0.92	0.92	0.92	0.91	0.90	0.90	0.89	0.88	0.87
	9000	0.68	0.92	0.92	0.92	0.91	0.91	0.90	0.89	0.88	0.88
	9500	0.65	0.91	0.92	0.92	0.91	0.91	0.90	0.89	0.89	0.88
	10000	0.63	0.91	0.92	0.92	0.91	0.91	0.90	0.89	0.89	0.88

48

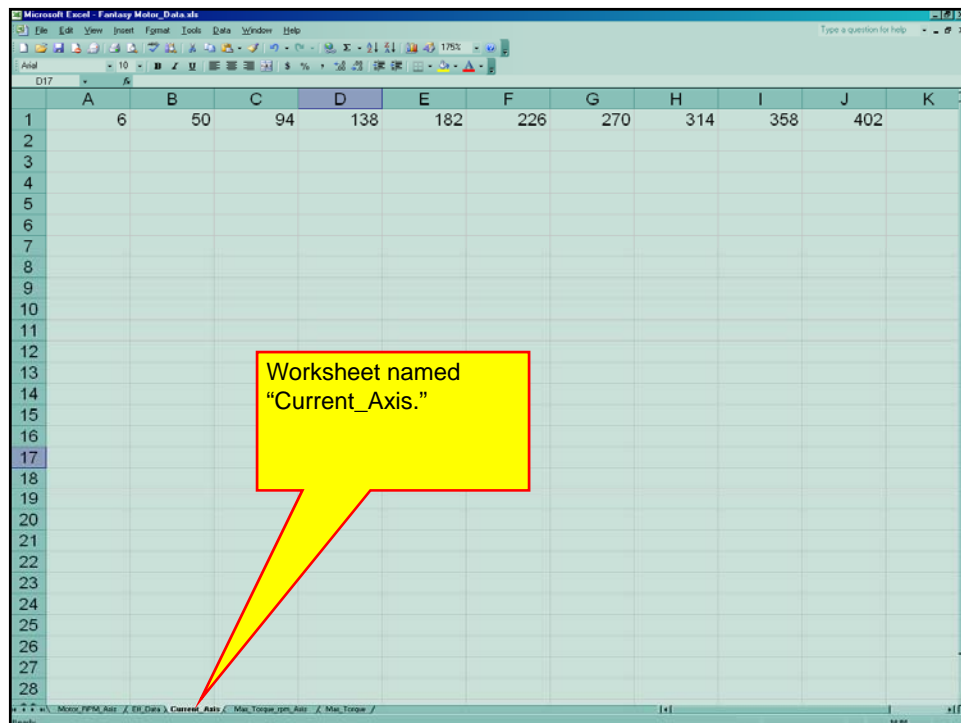
Save Data in Excel

- One worksheet for the rpm axis.
- One worksheet for the current axis.
- One worksheet for the efficiency table.



Worksheet named "Motor_RPM_Axis."

	A	B	C	D	E	F	G	H	I
1	0								
2	500								
3	1000								
4	1500								
5	2000								
6	2500								
7	3000								
8	3500								
9	4000								
10	4500								
11	5000								
12	5500								
13	6000								
14	6500								
15	7000								
16	7500								
17	8000								
18	8500								
19	9000								
20	9500								
21	10000								
22									
23									
24									
25									



Worksheet named "Current_Axis."

	A	B	C	D	E	F	G	H	I	J	K
1	6	50	94	138	182	226	270	314	358	402	
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											
28											



Worksheet named "Eff_Data."

	A	B	C	D	E	F	G	H	I	J
1	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
2	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
3	0.82	0.92	0.91	0.89	0.88	0.86	0.84	0.83	0.81	0.80
4	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
5	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
6	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
7	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
8	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
9	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
10	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
11	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
12	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
13	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
14	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
15	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
16	0.78	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
17	0.74	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
18	0.71	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
19	0.68	0.92	0.92	0.91	0.90	0.89	0.87	0.86	0.85	0.84
20	0.65	0.91	0.92	0.91	0.91	0.90	0.89	0.89	0.89	0.88
21	0.63	0.91	0.92	0.91	0.91	0.90	0.89	0.89	0.89	0.88
22										
23										
24										
25										
26										
27										

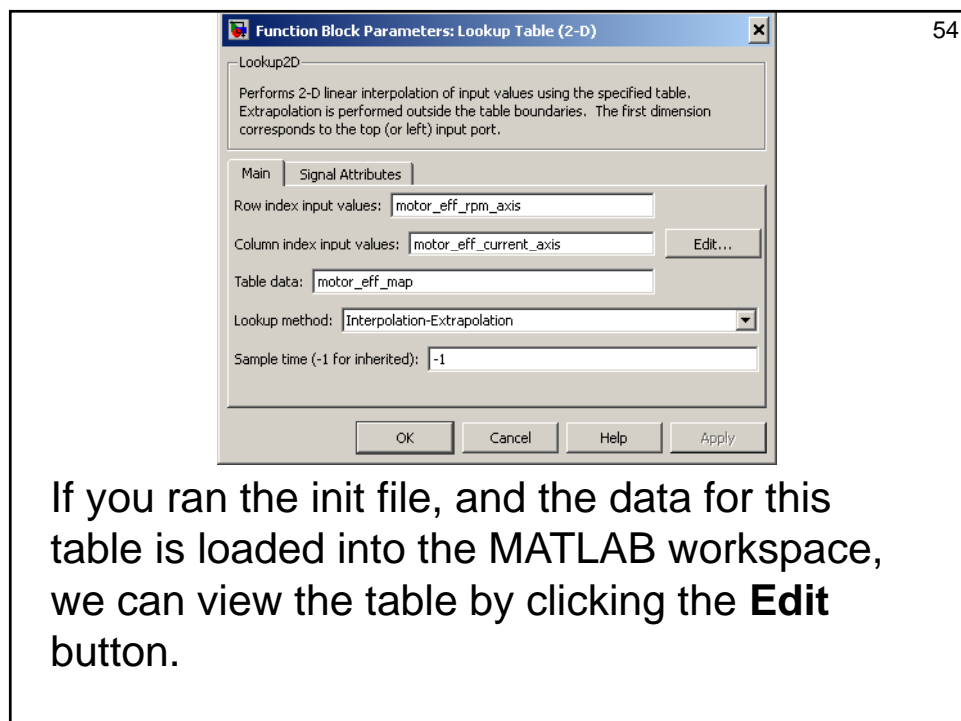
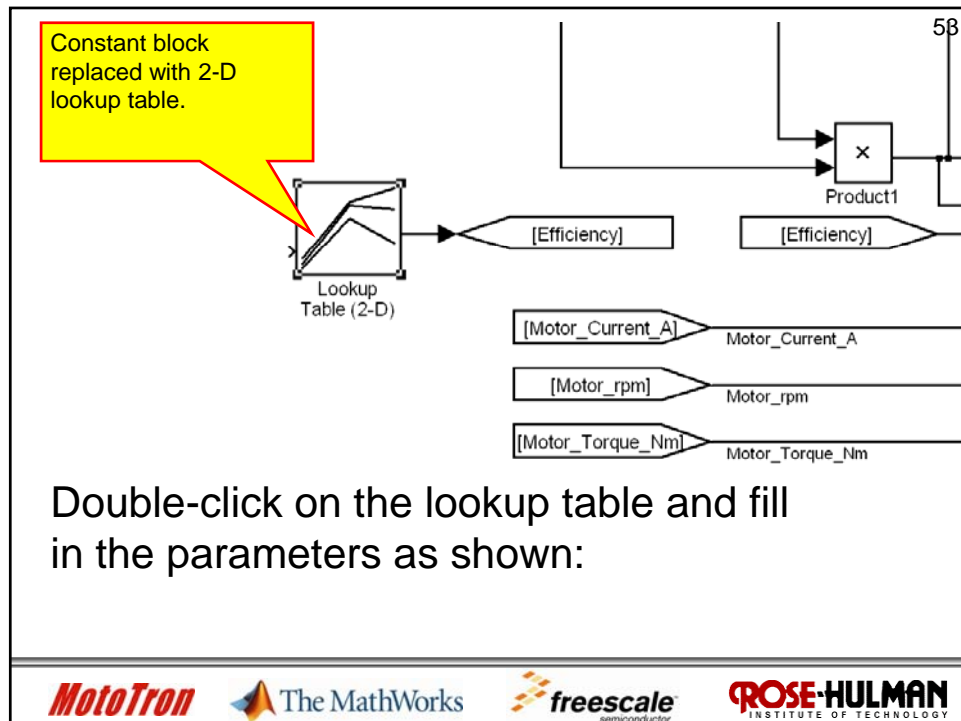
Excel

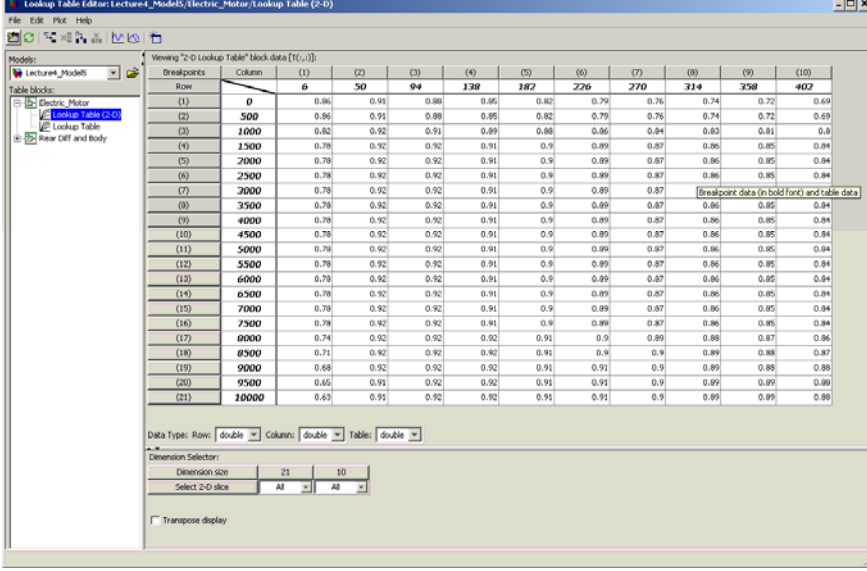
52

- Next, we will read this data using the xlsread function in MATLAB. Place these commands in your init file.

```
motor_eff_map = xlsread(Motor_PN,'Eff_Data');  
motor_eff_rpm_axis = xlsread(Motor_PN,'Motor RPM Axis');  
motor_eff_current_axis = xlsread(Motor_PN,'Current Axis');
```

- In our motor model, we will replace the constant efficiency with a Lookup Table (2-D)
(**Simulink/Lookup Tables**)
- Before continuing, run your init file to read in the table data.



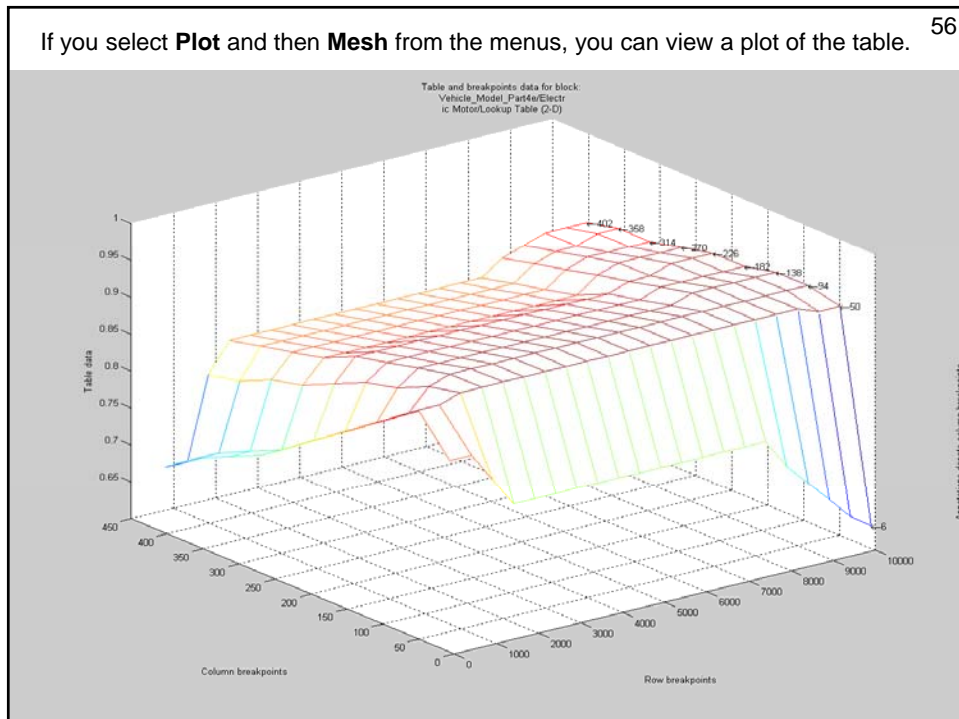


Viewing "2-D Lookup Table" block data [1(1,1)]:

Row	Column	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
0	6	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
500	6	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
1000	6	0.82	0.92	0.91	0.89	0.86	0.84	0.84	0.83	0.81	0.8
1500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
2000	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
2500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
3000	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
3500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
4000	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
4500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
5000	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
5500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
6000	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
6500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
7000	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
7500	6	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
8000	6	0.74	0.92	0.92	0.92	0.91	0.9	0.89	0.88	0.87	0.86
8500	6	0.71	0.92	0.92	0.92	0.91	0.9	0.89	0.88	0.88	0.87
9000	6	0.68	0.92	0.92	0.92	0.91	0.91	0.9	0.89	0.88	0.88
9500	6	0.65	0.91	0.92	0.92	0.91	0.91	0.9	0.89	0.89	0.89
10000	6	0.63	0.91	0.92	0.92	0.91	0.91	0.9	0.89	0.89	0.89

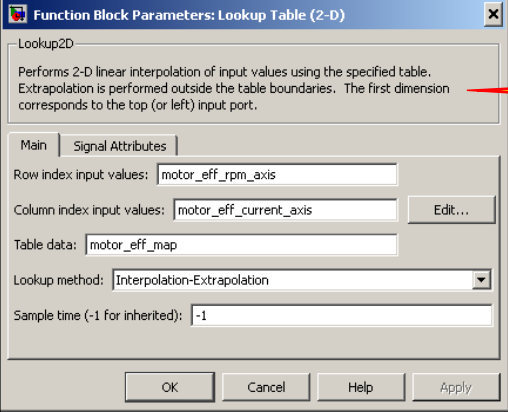
55

This editor shows us that we have defined the table correctly.






57

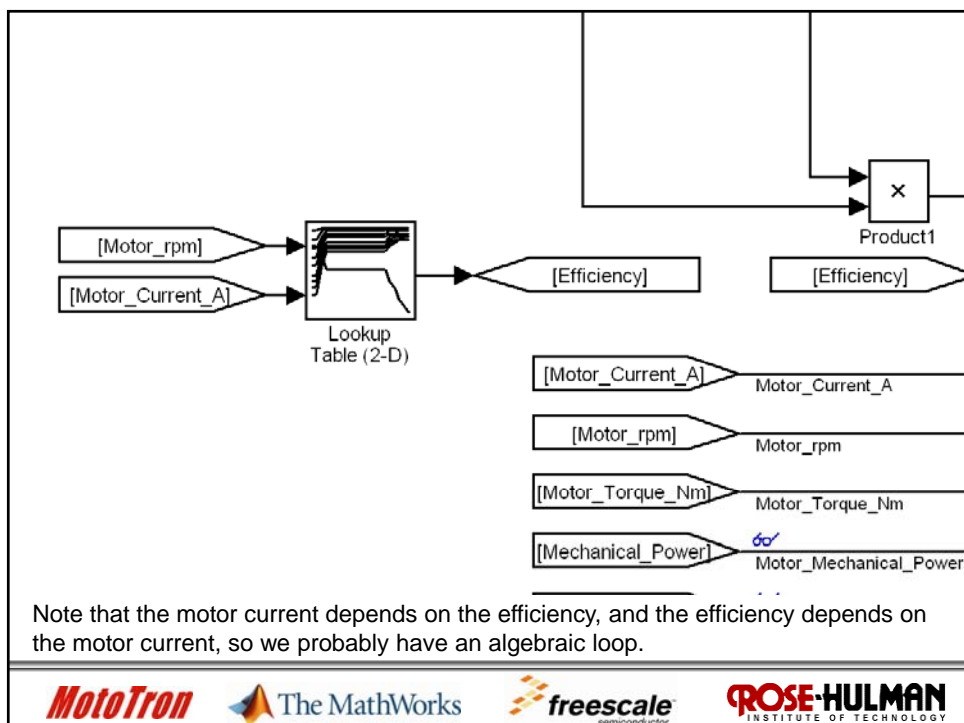


This dialog box also tells us:

- That the top input on the icon is the rpm axis
- The bottom input is the current axis.

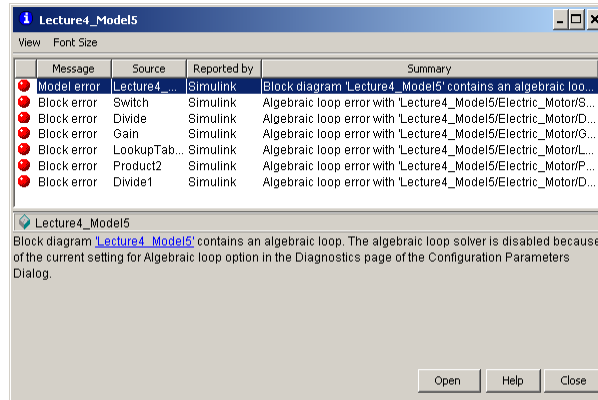
Click the **OK** button and connect the part as shown:





59

When we run the simulation, we are told that we do indeed have an algebraic loop.



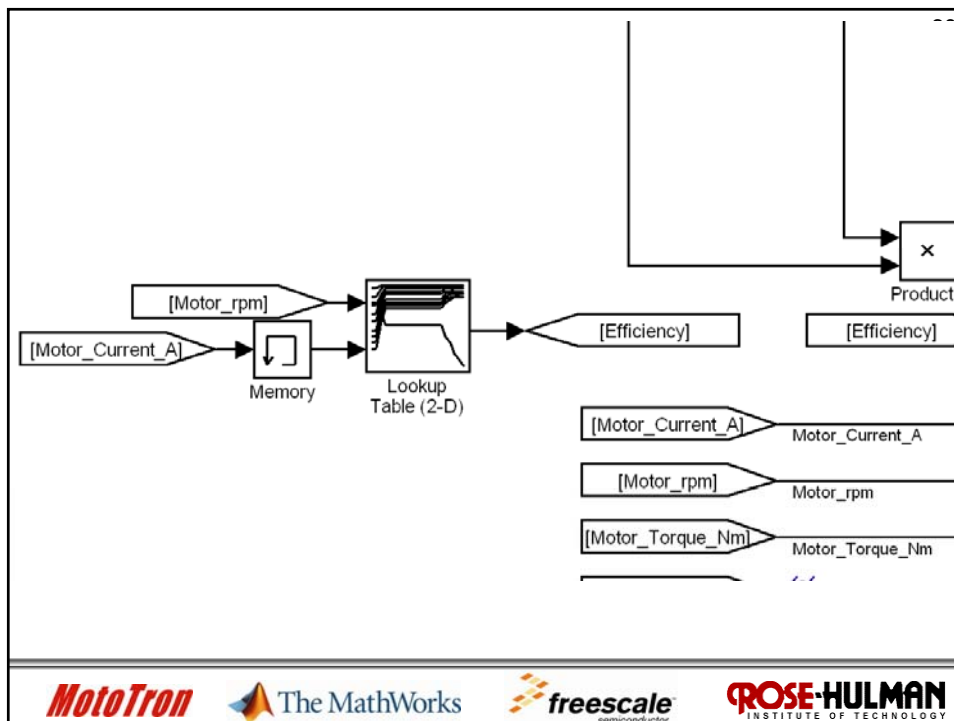
Eliminate the algebraic loop by adding a memory part (Simulink/Discrete) to the model as shown.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

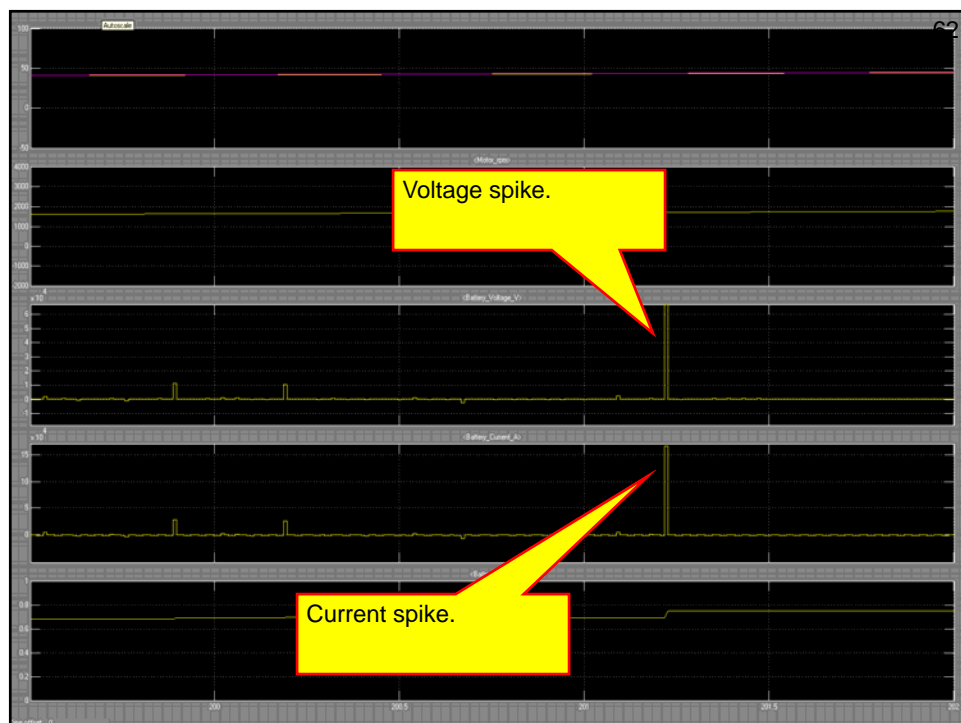
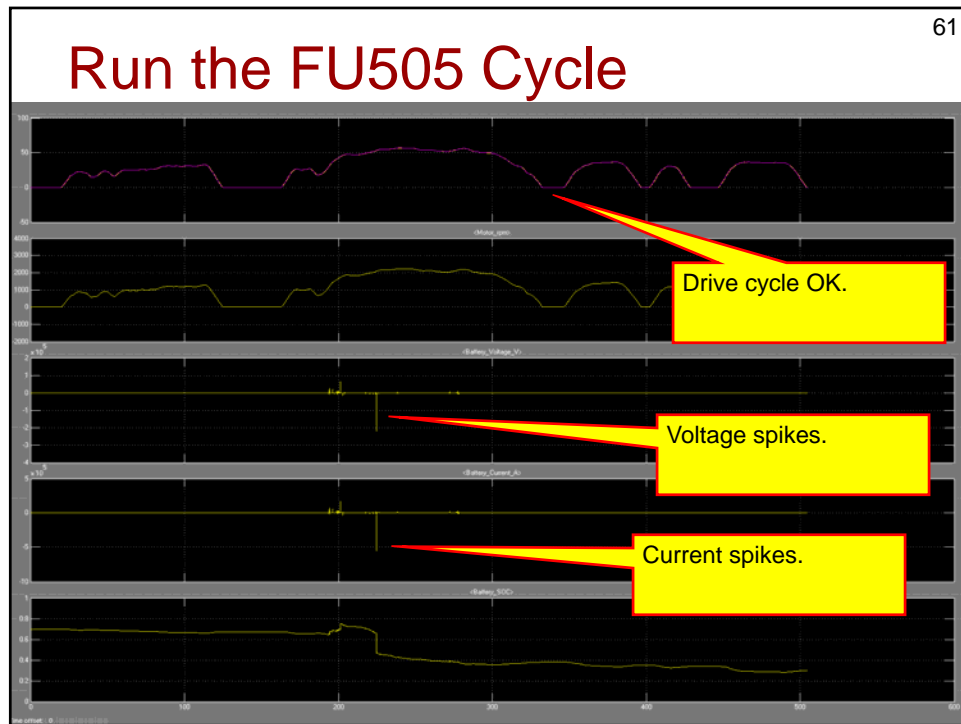


MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

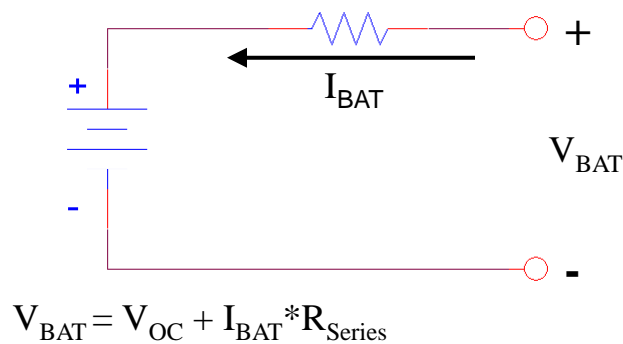




Problems

63

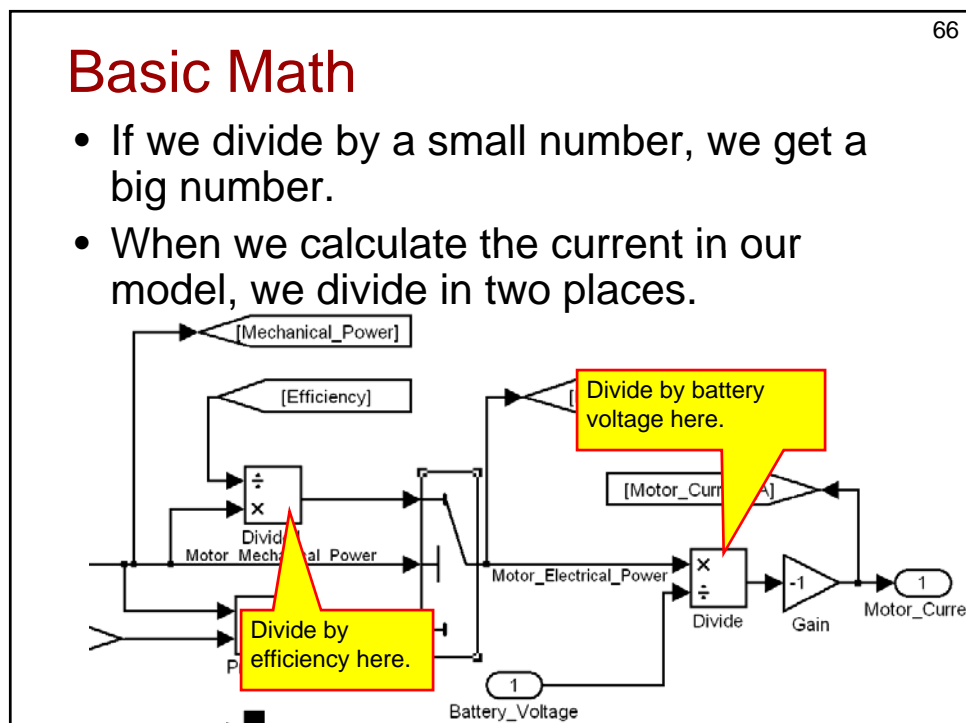
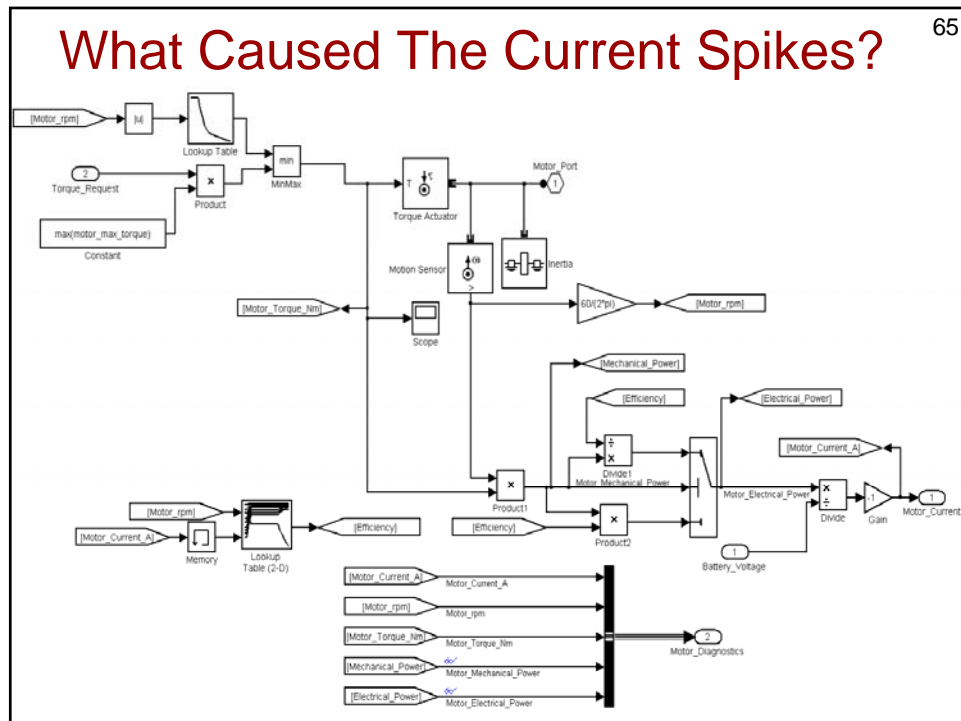
- What causes the voltage spike?
- Look at the battery model:



Problems

64

- From our model, we realize that current spikes cause the battery voltage spikes.
- If we can control the current, we can control the voltage fluctuations.
- Future needs:
 - Current limits (hard).
 - Over/under voltage detection.



67

Problems

- The only time the battery voltage will become small is when there is a large current spike.
- So the current spike causes the low battery voltage.
- Without the current spike, the battery voltage would remain high, and would not cause the current spike.
- This sounds like an algebraic loop of reasoning.

MotoTron

 **The MathWorks**

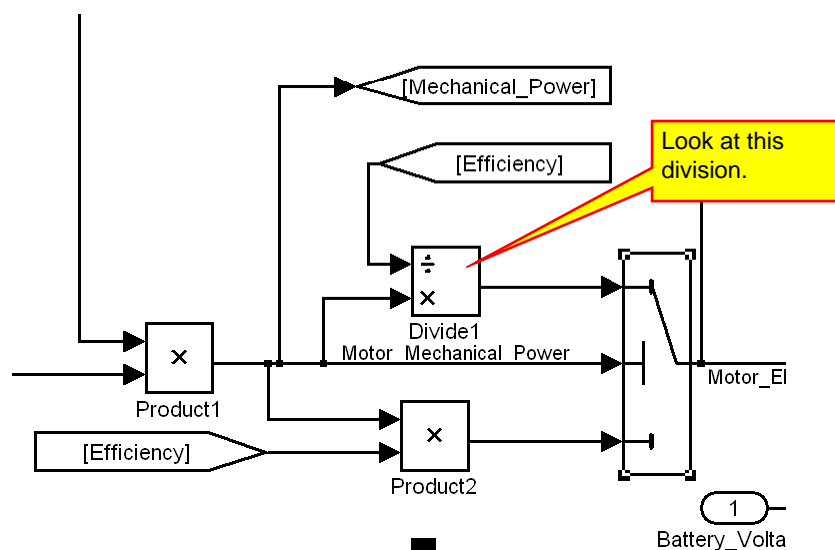
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

68

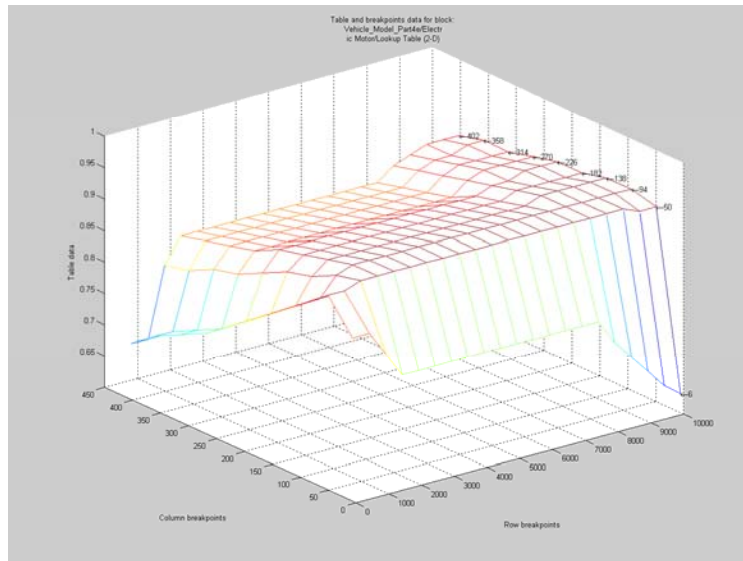
Problems

- Focus on the other division.



69

Can the Efficiency Ever Become 0?



70

Can the Efficiency Ever Become 0?

Lookup Table Editor: Vehicle_Model_Part4e/Electric Motor/Lookup Table [2-D]

File Edit Plot Help

Models: Vehicle_Model_Part4e

Table blocks: Electric Motor, **Lookup Table (2-D)**, Lookup Table, Rear Diff and Body

Viewing "2-D Lookup Table" block data [TC,]:

Breakpoints	Column	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Row		6	50	94	138	182	226	270	314	358	402
(1)	0	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
(2)	500	0.86	0.91	0.88	0.85	0.82	0.79	0.76	0.74	0.72	0.69
(3)	1000	0.82	0.92	0.91	0.89	0.88	0.86	0.84	0.83	0.81	0.8
(4)	1500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(5)	2000	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(6)	2500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(7)	3000	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(8)	3500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(9)	4000	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(10)	4500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(11)	5000	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(12)	5500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(13)	6000	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(14)	6500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(15)	7000	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(16)	7500	0.78	0.92	0.92	0.91	0.9	0.89	0.87	0.86	0.85	0.84
(17)	8000	0.74	0.92	0.92	0.92	0.91	0.9	0.89	0.88	0.87	0.86
(18)	8500	0.71	0.92	0.92	0.92	0.91	0.9	0.9	0.89	0.88	0.87
(19)	9000	0.68	0.92	0.92	0.92	0.91	0.91	0.9	0.89	0.88	0.88
(20)	9500	0.65	0.91	0.92	0.92	0.91	0.91	0.9	0.89	0.89	0.88
(21)	10000	0.63	0.91	0.92	0.92	0.91	0.91	0.9	0.89	0.89	0.88

Data Type: Row: double Column: double Table: double

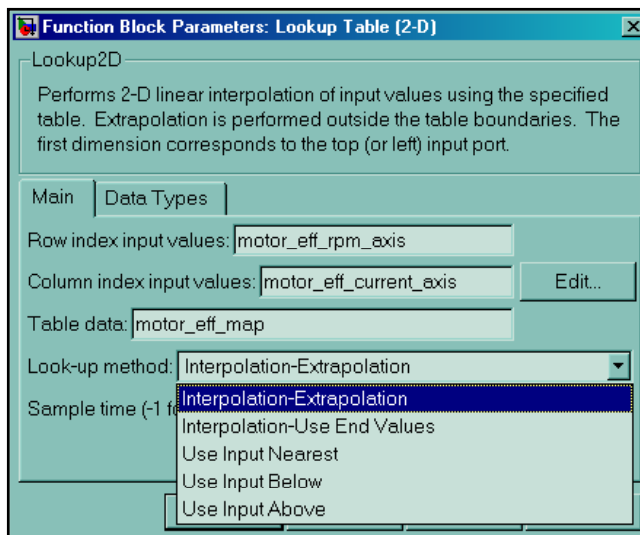
Dimension Selector: Dimension size: 21 Select 2-D slice: All Transpose display: ☐



Table-Lookup

71

- What happens when the current is zero?
- What happens when the current is greater than 402 A?
- What happens when the rpm is greater than 10000?
- ➔ We specify this in the lookup table dialog box:



72

- So, what happens when the index is outside of the range of the table?
- I'm not sure, but you better make sure that you know, or limit the inputs to be within a specific range defined in the table.

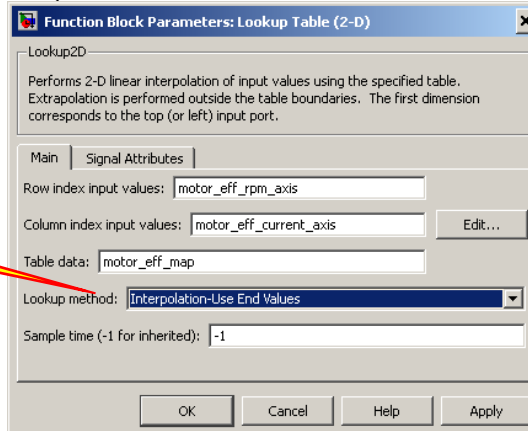


Lookup Tables

73

- To fix one problem, select **Interpolation – Use End Values** as the Lookup method.
- With this method, when we go outside of the range of data specified in the table, the values at the ends are used.

Interpolation – Use End Values selected.



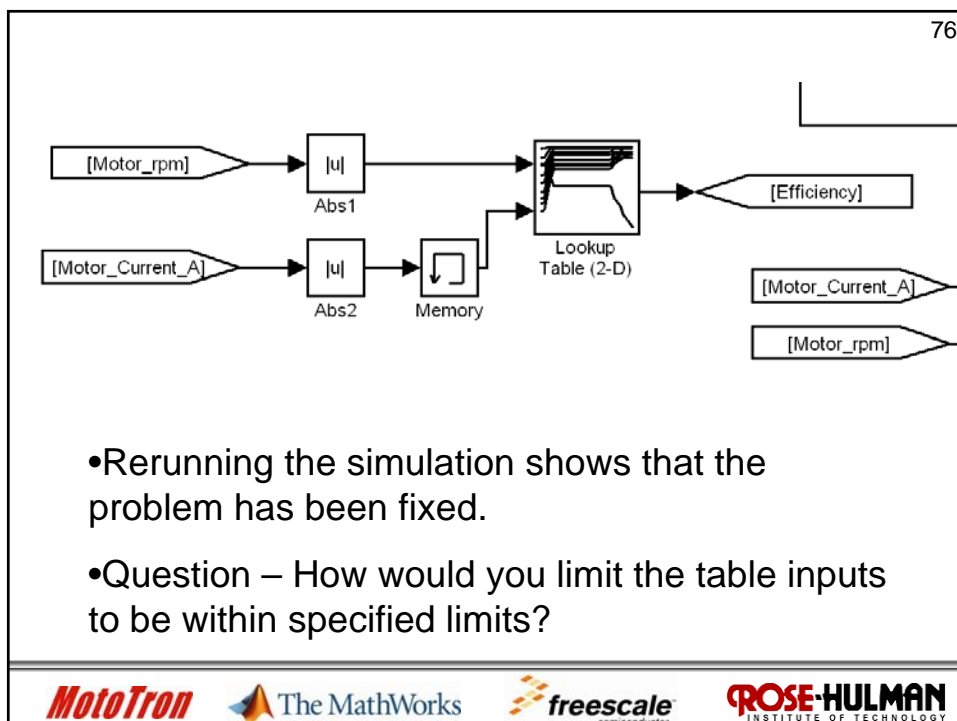
Problem

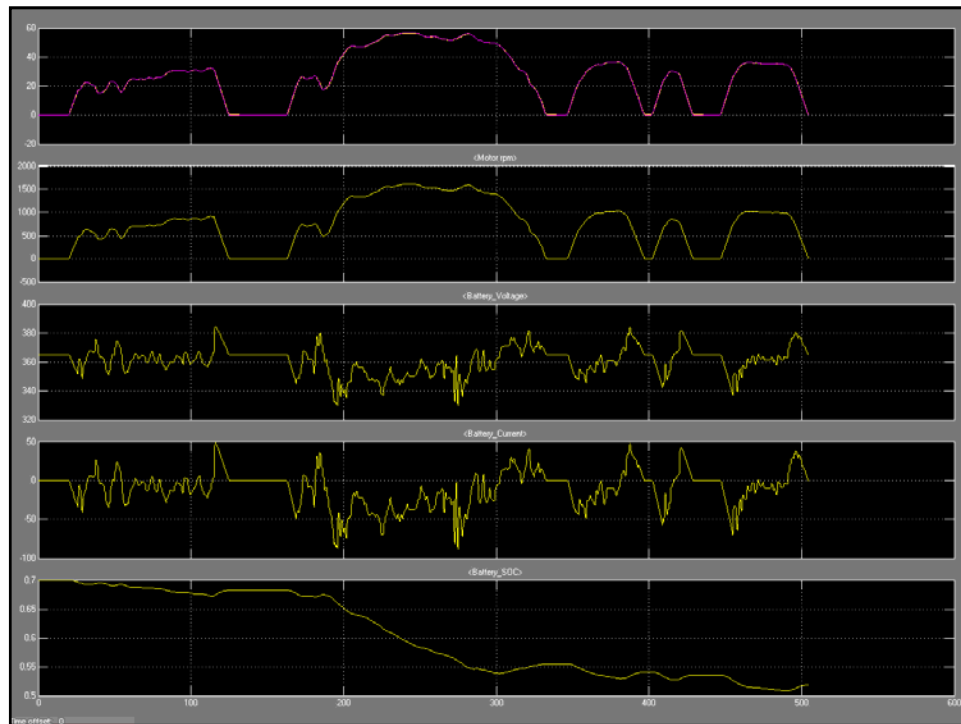
74

- Looking at the table of data reminds me of a problem.
- Hmm. I remember something a while ago...
- Can't remember exactly what it was, but something I saw before doesn't quite jive with the table data...
- What was it.



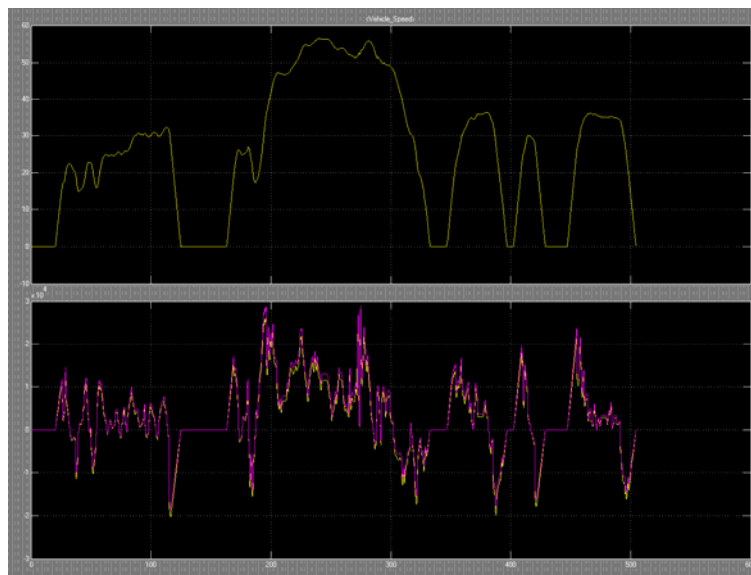
- The battery current is negative.
- Is this OK? Yes – our motor normally acts as a motor or as a generator during regen braking.
- Our table is only defined for positive currents, but we are using it for both motoring and regen modes.
- The table does apply when the motor is used as a generator, but we did not account for the negative current values in our model.
- Also note that the table applies for both positive and negative motor speed.
- Add the abs part (**Simulink/Math Operators**) as shown.





78

Look at the efficiency plot to verify our model. How can we improve this plot to make it more useful?





Motor Model

79

- We could add much more detail to the model, and you should as you make it more realistic. We could add:
 - Different regen and motoring efficiencies.
 - Add separate current limits for regenerative braking and motoring. These limits are usually different due to battery limits.
 - Add a torque map that is based on battery voltage and motor rpm.



Lecture 4 Exercise 4

80

- Demo of model working with motor efficiency and motor torque curve.

Demo_____





Lecture 4 Exercise 5

Battery Model Improvements

- Different charge and discharge resistances.
- Resistance a function of battery SOC and temperature.
- Battery open circuit voltage is a function of battery SOC and temperature.
- Data contained in file “Battery Data.xls.”

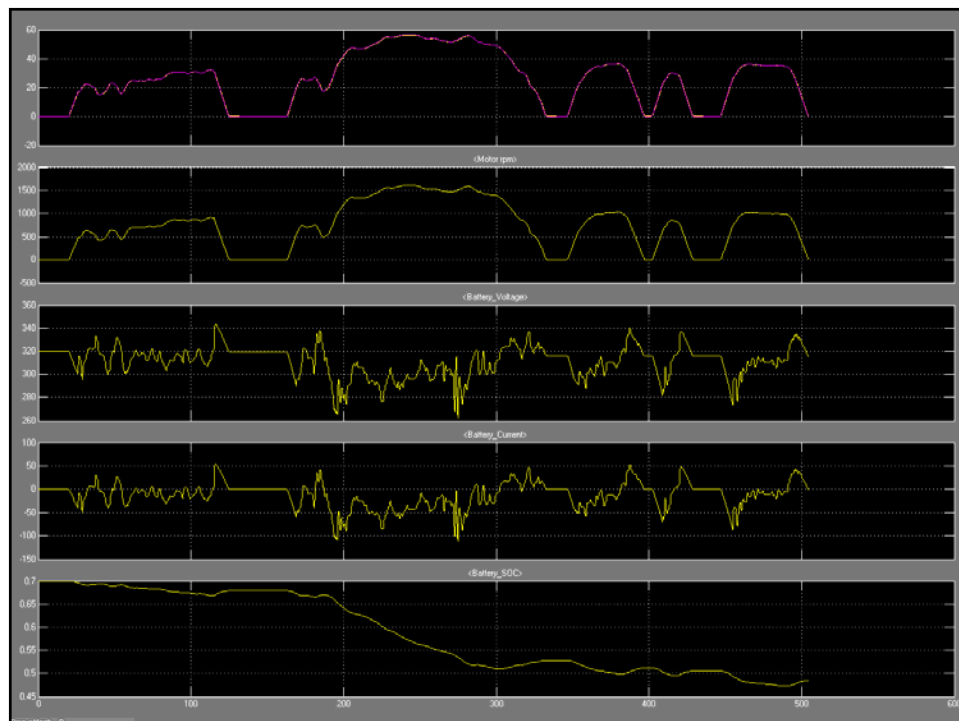
Demo_____

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>



Advanced Model-Based-System Design

Lecture 5: Engine Modeling

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Outline

2

- Reasons for an IC Engine
- Intro to SimDriveline Engine Block
- Fuel Consumption
- Torque Curve
- Defueling
- Braking Torque

MotoTron

 The MathWorks

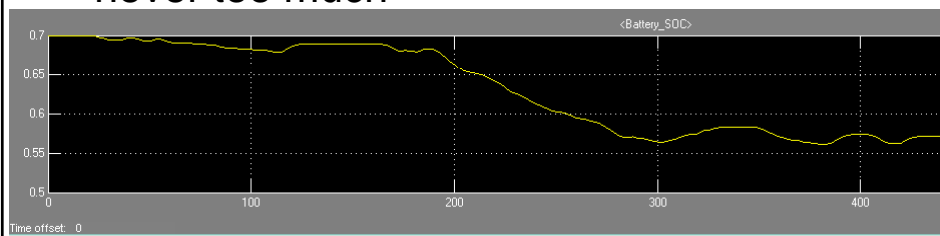
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

The Need

3

- Thus far our vehicle has been able to follow various drive cycles using the motor and battery
- The battery SOC has decreased, but never too much



MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

The Need

4

- Eventually, however, the battery will reach a SOC where it needs to be recharged.
- There is also a critical low SOC which, if exceeded, will decrease the battery lifetime or cause unsafe operation.
- We can pull the vehicle over and plug it in
– Or
- Have an on-board power generation system using an internal combustion engine and a generator.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



The Need

5

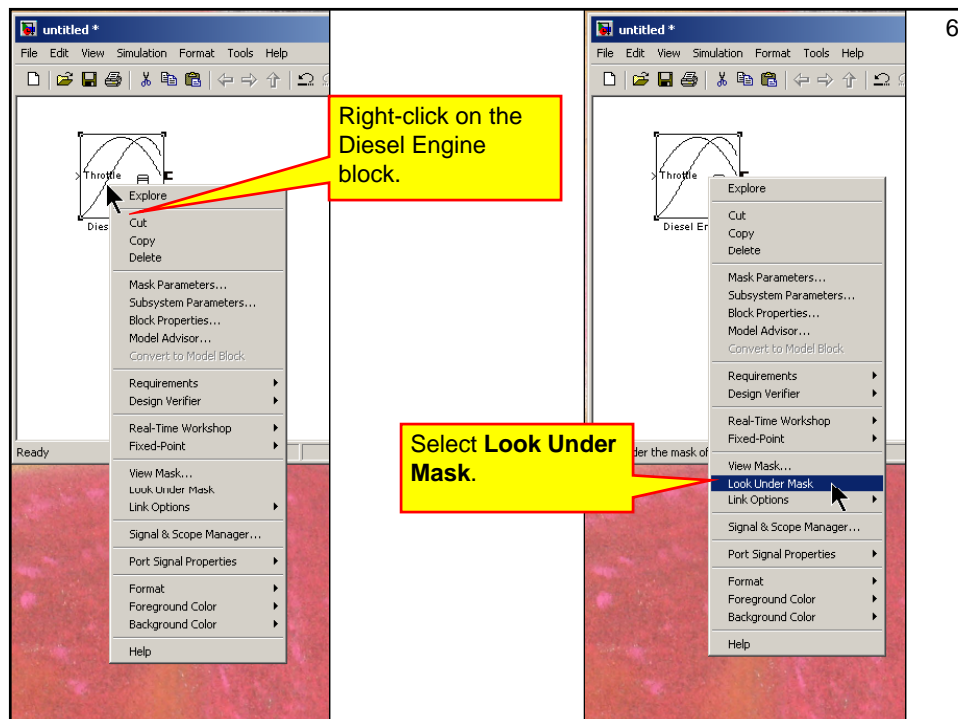
- For this module we will develop our own model of a Diesel engine.
- First, let's see take a look at the SimDriveline Diesel engine.
- Open a new empty model and place a **Diesel Engine** block in your model (library **Simscape / SimDriveline / Vehicle Components**).

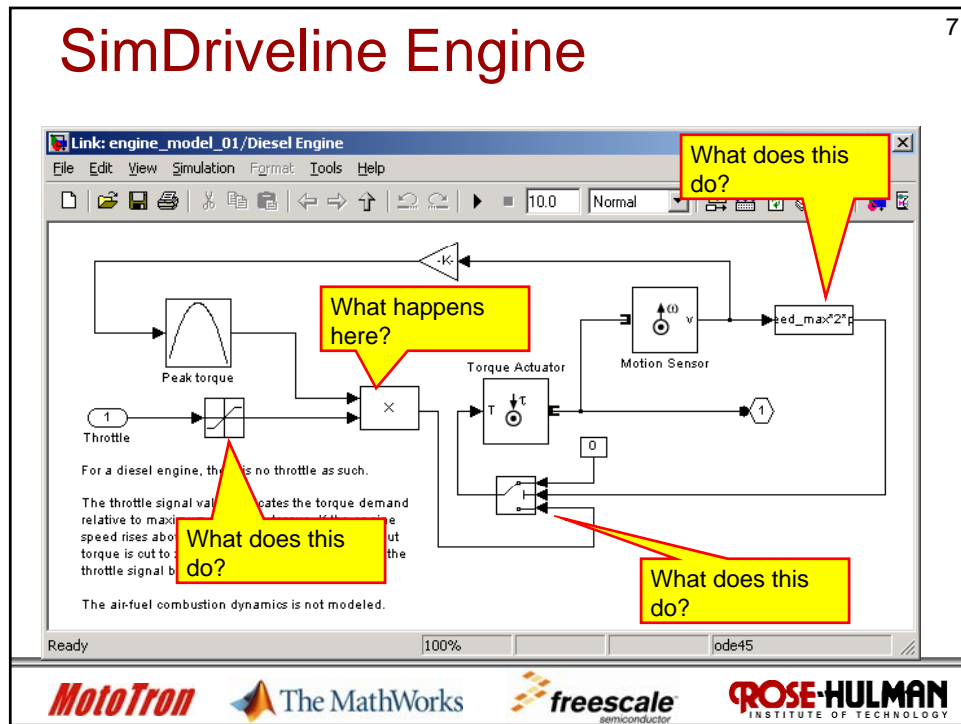
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Lecture 5 Exercise 1

8

- What is the purpose of the switch?
- What is the purpose of the compare to constant block?
- What is the value of speed_max and where does it come from?

Answers_____

Answers_____

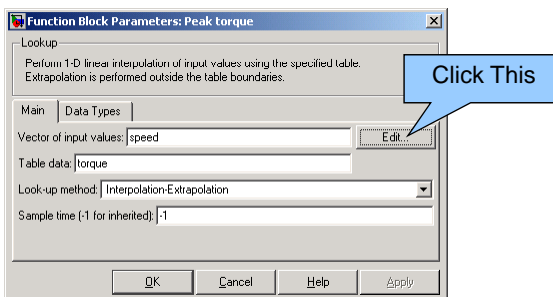
Answers_____

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

SimDriveline Engine

9

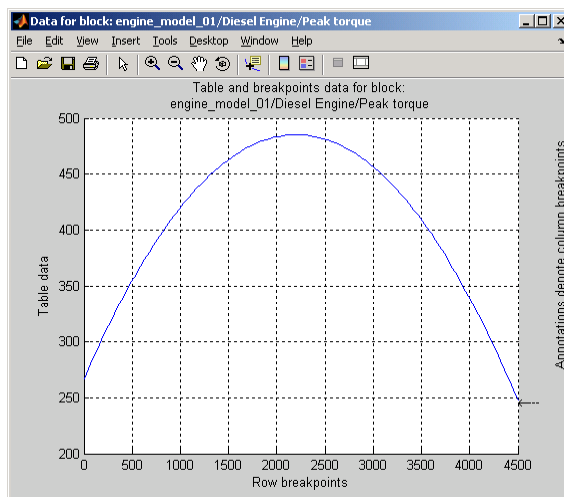
- Question
 - Does an engine produce torque at RPMs BELOW idle?
- Problem
 - Double-click on the Peak torque lookup table.



SimDriveline Engine

10

- This model has some problems...



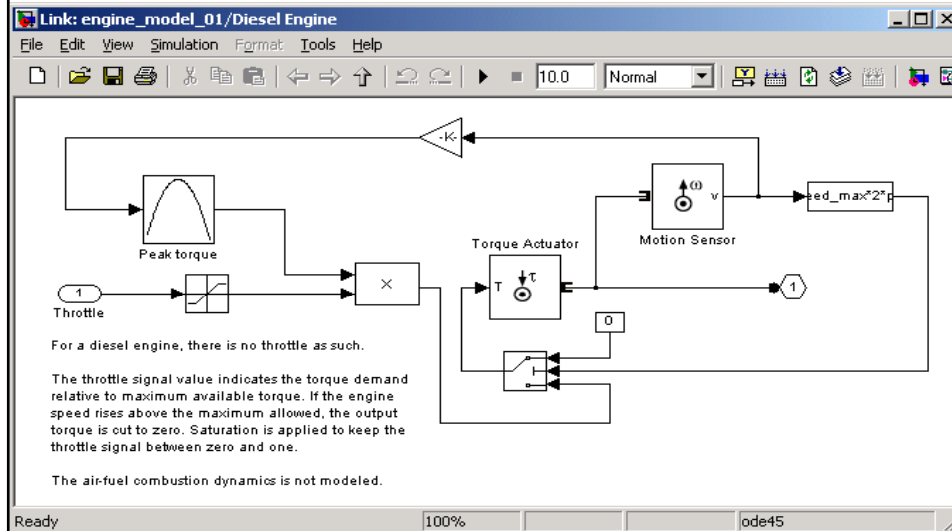
- The model produces torque below idle and at zero rpm.
- What happens above 4500 rpm and below 0 rpm?



SimDriveline Engine

11

- Looking at the model we notice a few other things are missing that we need for our model.



SimDriveline Model

12

- We see that the model does not have an inertia for the engine.
- The model does not calculate fuel consumption. (Necessary since we are building a hybrid vehicle and our main motivation is reduced petroleum consumption.)
- The SimDriveline engine gives us insight as to how to build our own model. We shall use it as a starting point and then enhance the model.



Fuel Curve

13

- At a given RPM we can inject various amounts of fuel, resulting at various consumption rates
- Check out the FantasyEngine_Data.xls file for the engine RPM and “throttle” axes as well as the fuel consumption data
- Using the skill gained with the motor, update the Vehicle_Init file accordingly



Engine Model

14

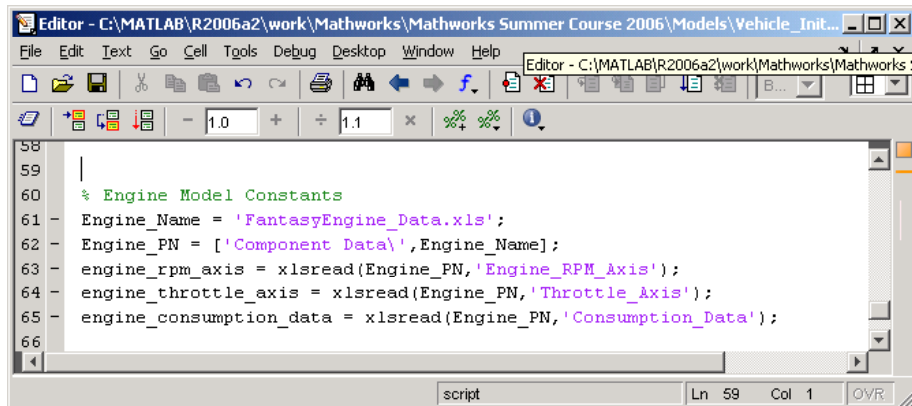
- We will create our own model.
- We will start by looking at the fuel consumption of a Diesel engine.
- We will assume that you have measured or obtained the fuel map for your engine.
- Fuel consumption and torque maps for our fictitious engine have been measured and saved in an excel file in the Component Data directory.



Fuel Curve

15

- We can read this data with the xlsread function



```
58  
59  
60 % Engine Model Constants  
61 Engine_Name = 'FantasyEngine_Data.xls';  
62 Engine_PN = ['Component Data', Engine_Name];  
63 engine_rpm_axis = xlsread(Engine_PN, 'Engine_RPM_Axis');  
64 engine_throttle_axis = xlsread(Engine_PN, 'Throttle_Axis');  
65 engine_consumption_data = xlsread(Engine_PN, 'Consumption_Data');  
66
```

MotoTron

 The MathWorks

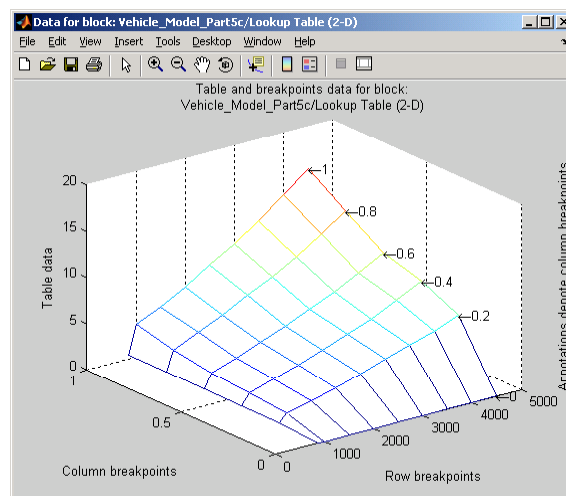
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Fuel Curve

16

- Create a new model and add a 2-D look up table.
- Awesome!



MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Torque Curve

17

- By burning fuel, a Diesel engine produces torque.
- The torque is a function of RPM and throttle.
- Read in the torque data from the FantasyEngine_Data.xls file
 - call it *engine_torque_data*

MotoTron

The MathWorks

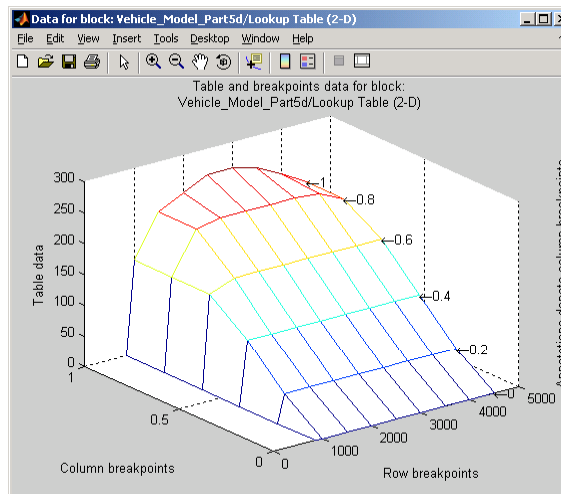
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Torque Curve

18

- Pretty Cool!
- At a given RPM we can adjust the throttle to get a desired torque



MotoTron

The MathWorks

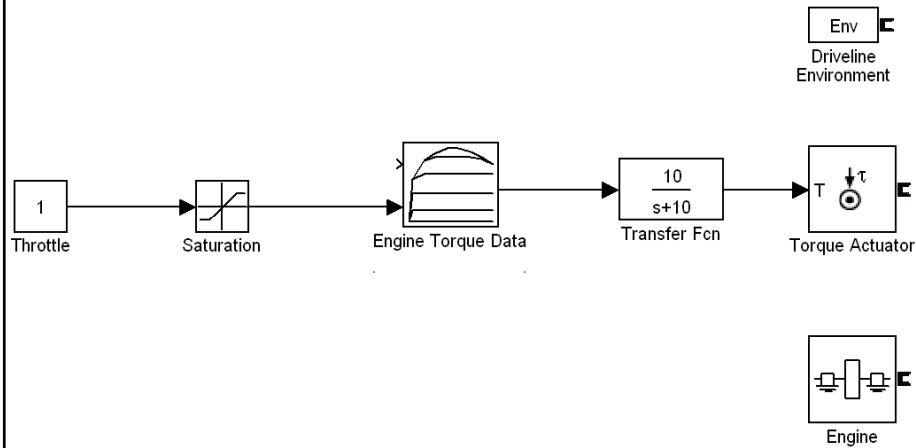
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Engine Model

19

- Create the model below with the following parts:
 - Constant, Saturation, Transfer Fnc, Torque Actuator, Driveline Environment, Inertia



Engine Model

20

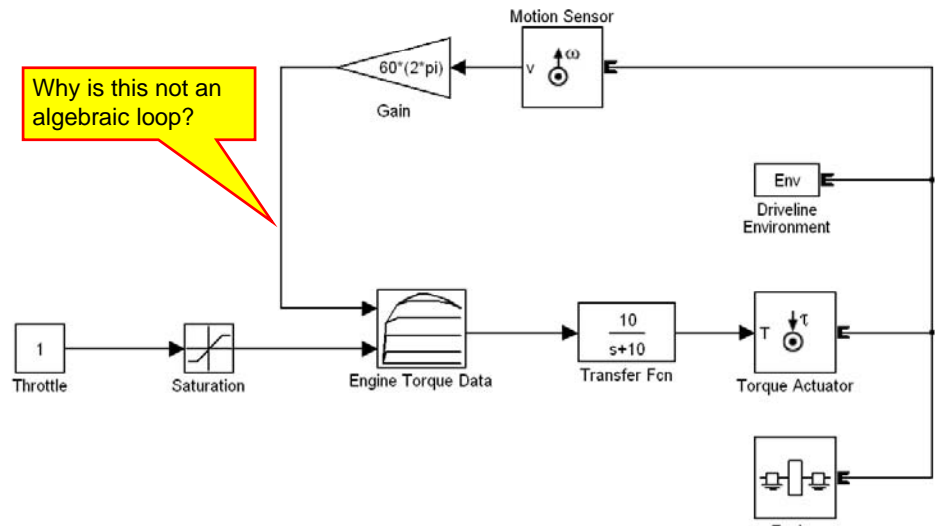
- The Laplace (continuous) transfer function delays the engine response with a time constant of 100 ms
- The limits on the saturation part are 0 and 1.
- The engine inertia is specified in the init file as "engine_inertia."

Engine Model

21

- Now add an RPM feedback loop to the data table

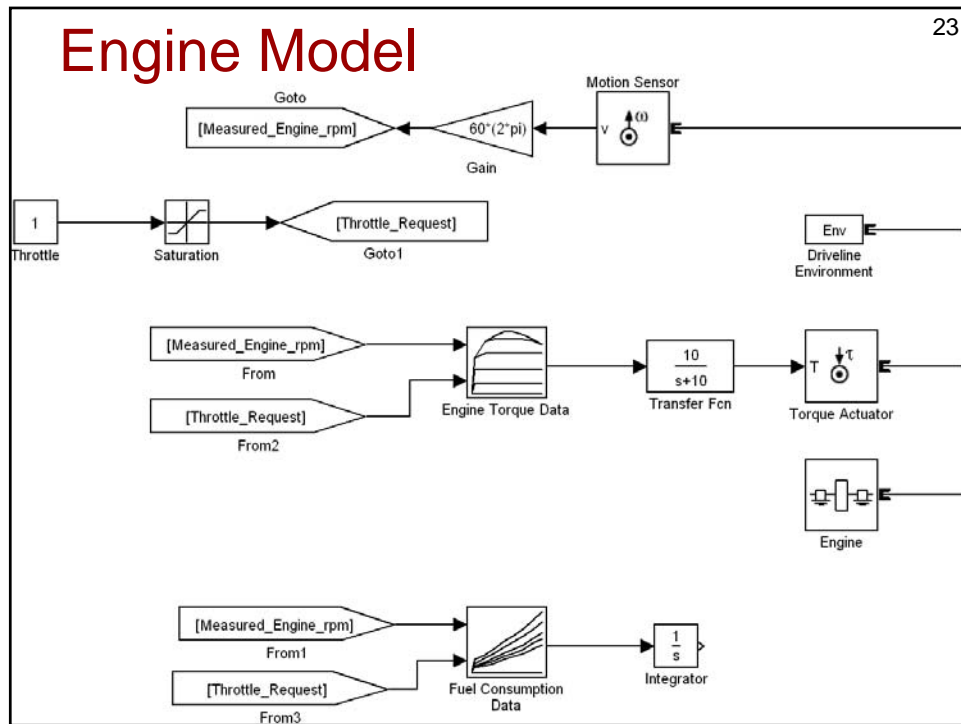
Why is this not an algebraic loop?



Engine Model

22





- Next we will add fuel consumption to the model.
- First, we will clean up the model by adding some signal routing From and Goto parts.
- Note that in our model, the fuel consumption data is in grams of fuel consumed per second.

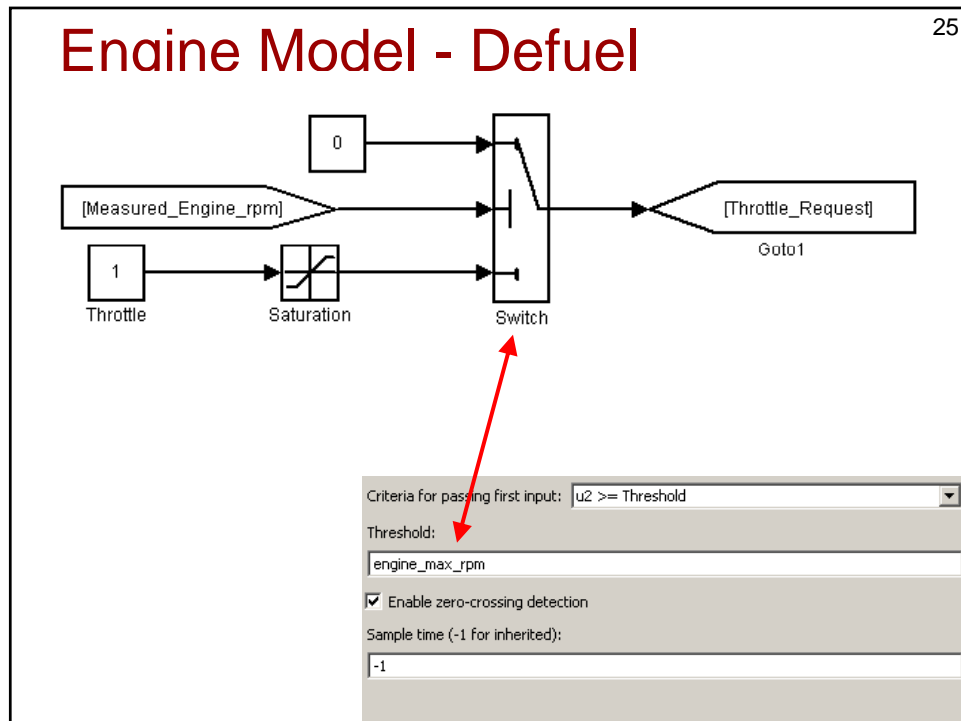


Engine Model - Defuel

24

- Next, we will build an emergency defuel if we exceed the max engine speed
- This sets the throttle to zero, shuts off fuel and kills the torque
- Add the constant **engine_max_rpm**
- **= 4500** to the init file
- Use a switch to set the throttle to zero if the engine speed exceeds 4500 rpm



Engine Model - Jake

26

- If the engine throttle is zero, engine braking will occur until idle is reached
- A constant torque will be applied opposite the direction of crankshaft rotation
- We'll estimate the engine braking torque to be 10 Nm – add this to the init file
 - *engine_brake_torque*

MotoTron **The MathWorks** **freescale** **ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY



Engine Model - Jake

27

- Now, we could do some fancy stuff with switches in our model
 - Or
- Modify our torque data by changing the zero throttle column from 0 Nm to -10 Nm for rows two through the last one
- Uncomment the last line in the engine section of the init file



Engine Model - Jake

28

```
60 % Engine Model Constants
61 - engine_max_rpm = 4500;
62 - engine_brake_torque = 10;
63 - Engine_Name = 'FantasyEngine_Data.xls';
64 - Engine_PN = ['Component Data\', Engine_Name];
65 - engine_rpm_axis = xlsread(Engine_PN, 'Engine_RPM_Axis');
66 - engine_throttle_axis = xlsread(Engine_PN, 'Throttle_Axis');
67 - engine_consumption_data = xlsread(Engine_PN, 'Consumption_Data');
68 - engine_torque_data = xlsread(Engine_PN, 'Torque_Data');
69 - engine_torque_data(2:end, 1) = -engine_brake_torque;
70
71
```

This line
uncommented.

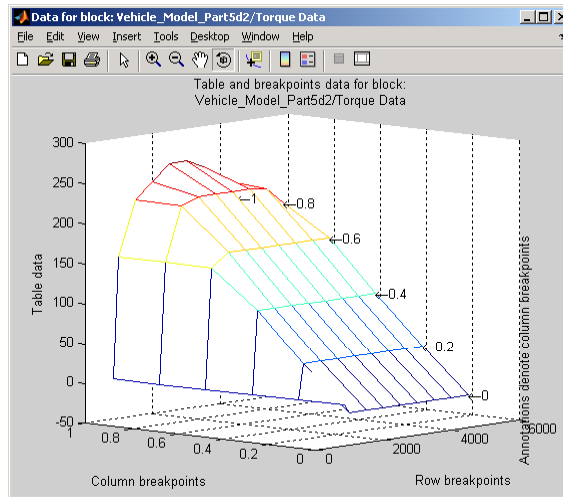
When the throttle is zero, the torque will be equal to
-engine_brake_torque.



Engine Model - Jake

29

- The torque definitely goes negative
- The output torque will be zero for a throttle slightly greater than 0.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Engine Model - Testing

30

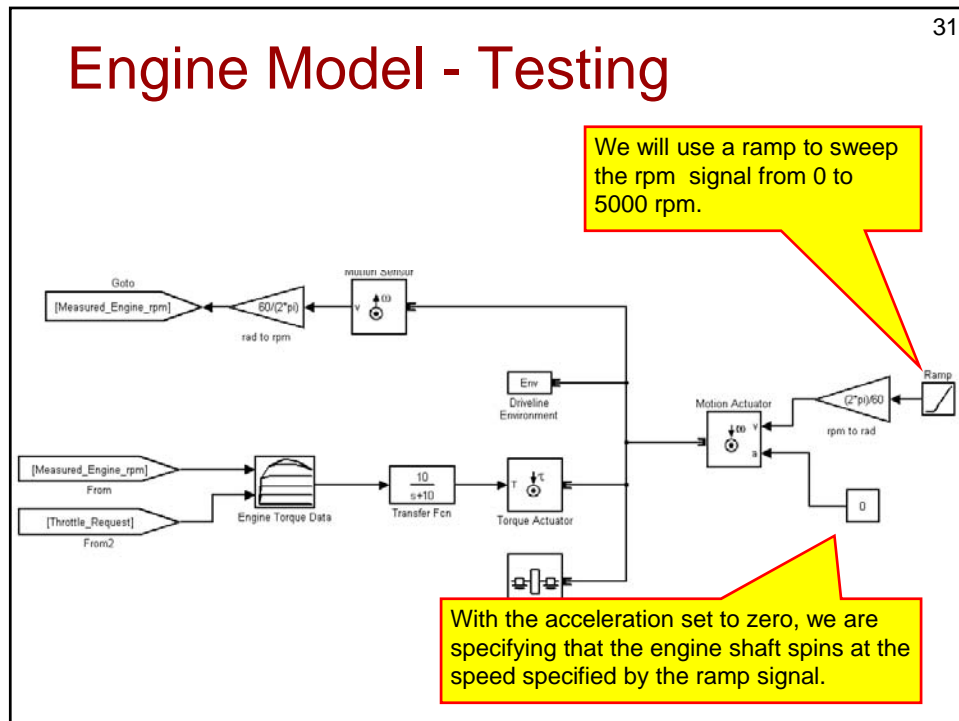
- Let's test our model by forcing the engine to spin using a motion actuator.
- Add in a
 - Motion actuator
 - Ramp
 - Gain
 - Constant
- Put scopes on the rpm, fuel consumption rate and the torque output from the look-up table

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Engine Model - Testing

32

- The properties of the ramp specify a slope of 1.
- If we run the simulation for 5000 seconds, the rpm will ramp from 0 to 5000.

Source Block Parameters: Ramp

Ramp (mask) (link)
Output a ramp signal starting at the specified time.

Parameters

Slope:

Start time:

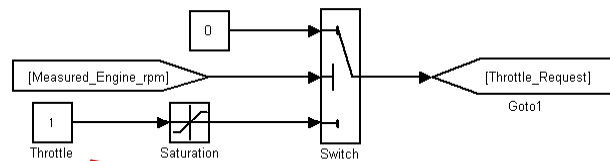
Initial output:

☒ Interpret vector parameters as 1-D

Engine Model - Testing

33

- Since we are asking for maximum torque, and the rpm is swept from 0 to 5000, we can generate a plot of maximum torque versus rpm.

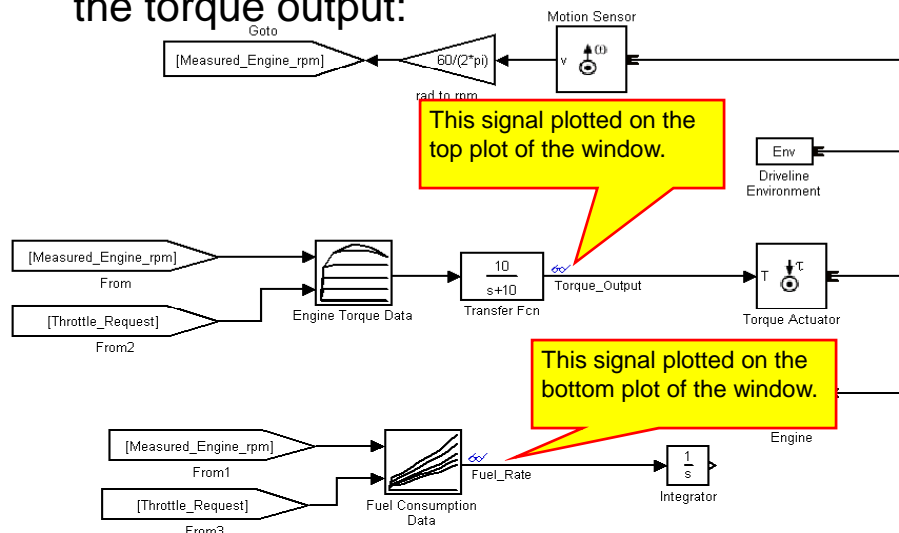


Throttle specified a 1. We are asking for maximum torque.

Engine Model - Testing

34

- We will generate plots of the fuel rate and the torque output:



35

Engine Model - Testing

- Run the model for 5000 seconds of simulation time.

MotoTron

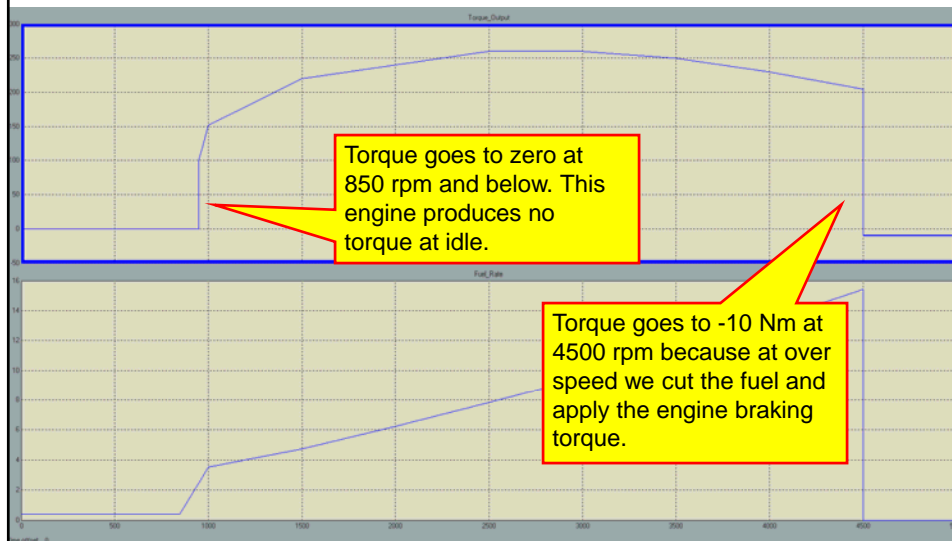
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

36

Engine Testing



MotoTron

The MathWorks

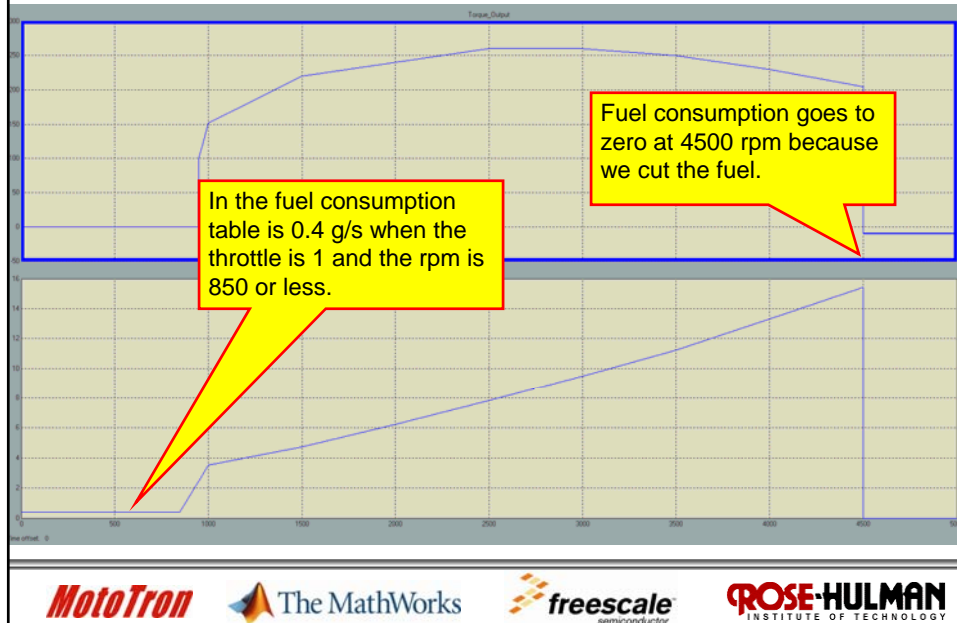
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Engine Testing

37



Engine Testing

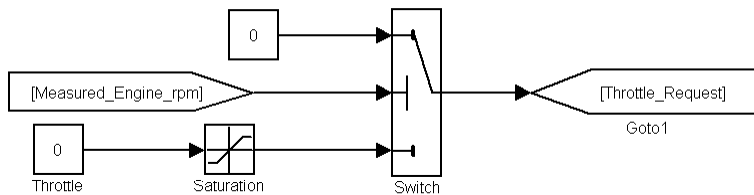
38

- We see that the fuel consumption is non zero even when the engine rpm is less than 800.
- We will need to fix this later when we make an engine starting algorithm.
- We will have an engine on signal that turns on or off fuel consumption below 800 rpm when the engine is off and at low rpm.

Engine Testing

39

- Next, we will generate the same plot with the throttle set to 0 rather than 1:



MotoTron

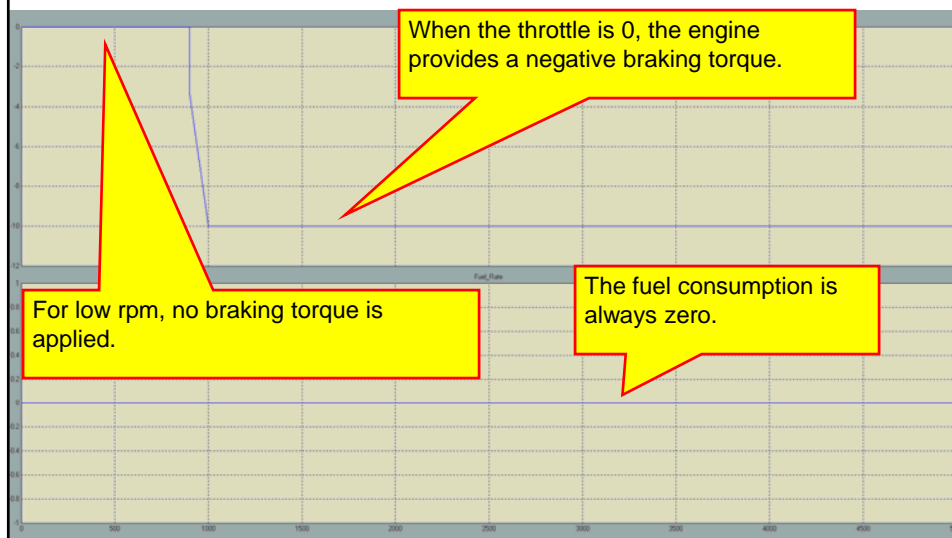
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Engine Testing

40



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Engine Testing

41

- We do notice a few issues that we need to fix later.
- The engine uses no fuel when the throttle is zero. This is because we are spinning the engine with an external source. Normally to spin the engine at this speed, a throttle will be required, and fuel will be consumed. Also, we may add in an idle controller that holds the throttle slightly above zero to keep the engine moving.



Engine Testing

42

- We also notice that the for an engine rpm less than 850, the torque is zero.
- We assume that the engine is off for this range of rpm.
- Later, when we add an engine on signal, we will also add an engine off torque, that will oppose the starter when we need to start the engine.





Lecture 5 Demo 1

43

- Engine demo at full throttle.

Demo_____

- Engine demo at zero throttle.

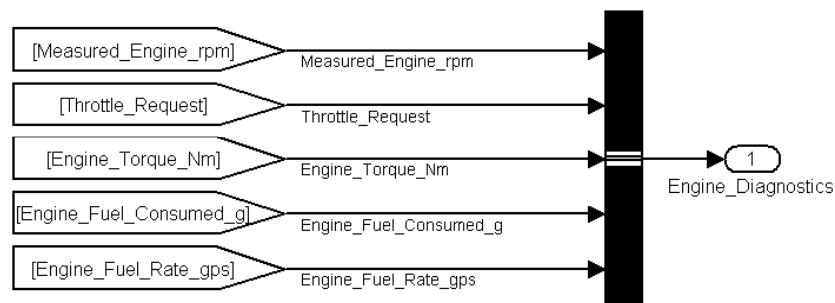
Demo_____



Engine Model

44

- We now wish to make a subsystem model out of our engine
- Add in an engine diagnostics bus and connect it to an output port.



Engine Model

45

- Get rid of the Env block. (We already have an Env block in our vehicle model. We only need one.)
- Replace the motion actuator, and associated ramp, constant, and gain block with an SimDriveline **Connection Port**.
- Replace the constant block used to specify the throttle with an inport:

MotoTron

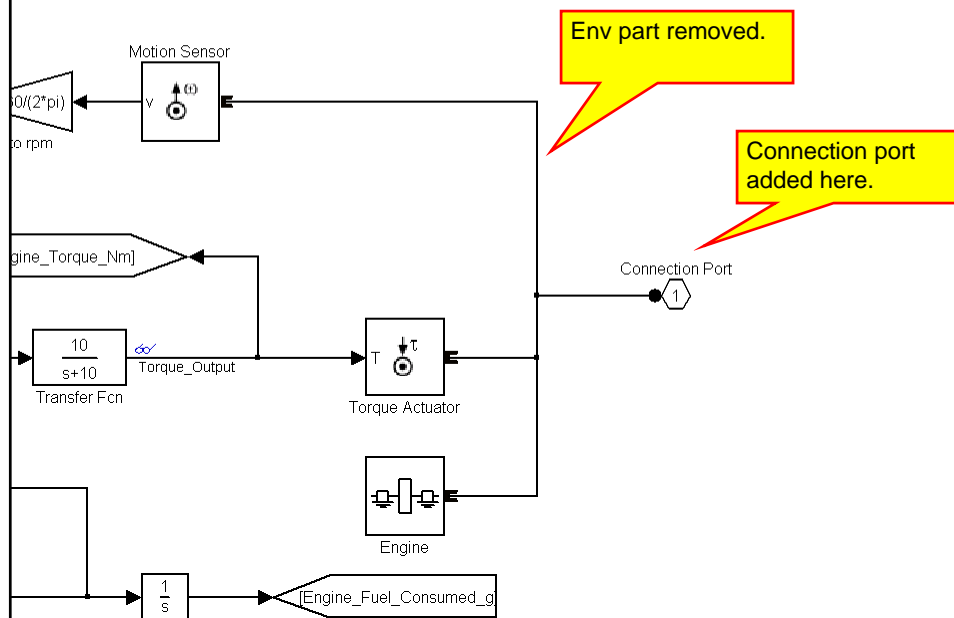
 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

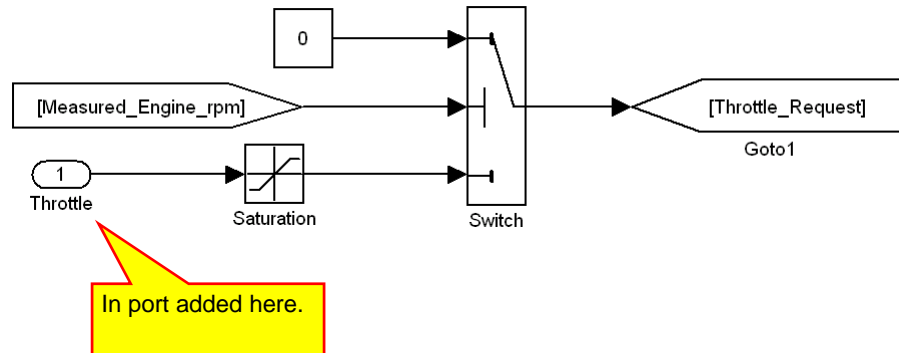
Engine Model

46



Engine Model

47



MotoTron

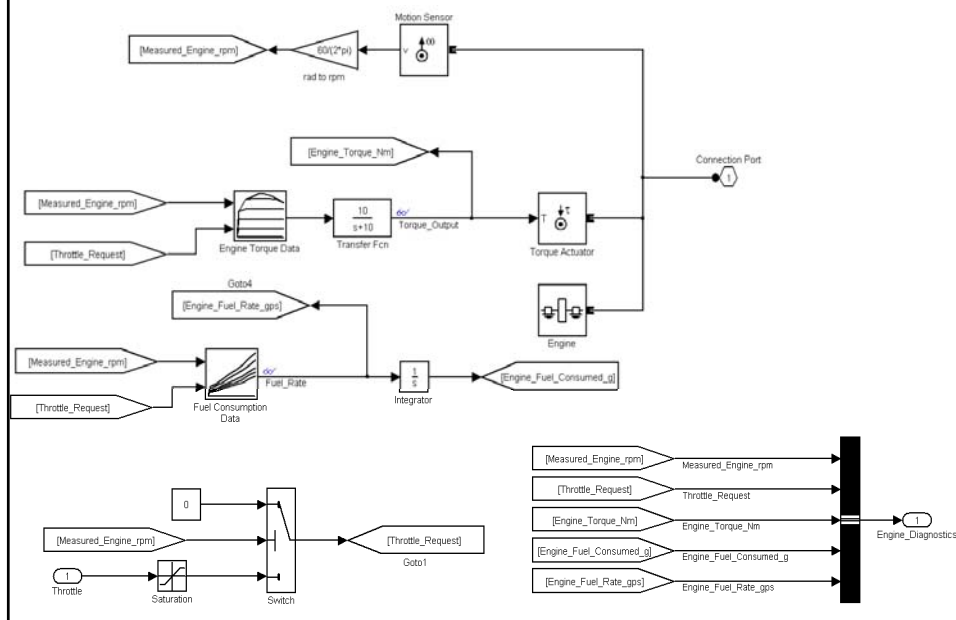
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Complete Engine Model

48

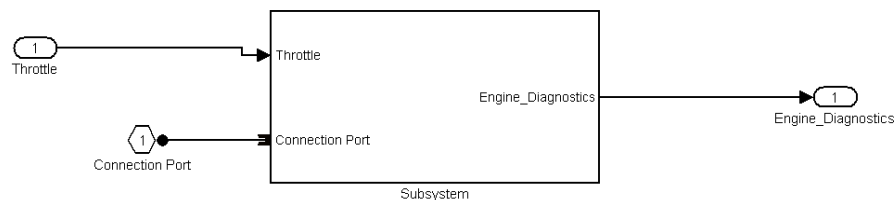




Engine Subsystem Model

49

- Type ctrl-A to select all of the components.
- Right-click on one of the components and select Create Subsystem from the menus.
- Go up one level to view the subsystem:



MotoTron

The MathWorks

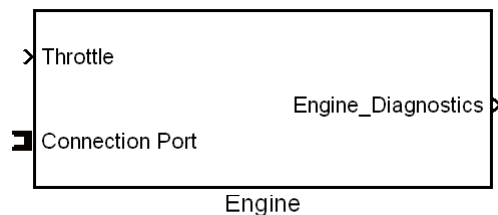
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Engine Subsystem Model

50

- Delete all of the ports and rename the subsystem as “Engine.”



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Lecture 5 Exercise 1

Engine Model Improvements

- Add an “Engine on” signal to the engine model. When the engine is on, the fuel rate and the torque are determined by the lookup tables of the throttle cutoff.
- When the engine is off, the fuel rate is zero and the engine torque is a negative constant equal to -15 Nm .
- Prove that your design works.
- Abrupt step changes in the output torque is not allowed.

Demo_____



Advanced Model-Based-System Design

Elementary Control using
Stateflow



Presentation Outline

53

- Simple charging logic
- Introduction to Stateflow
- Completing the model

MotoTron

 **The MathWorks**

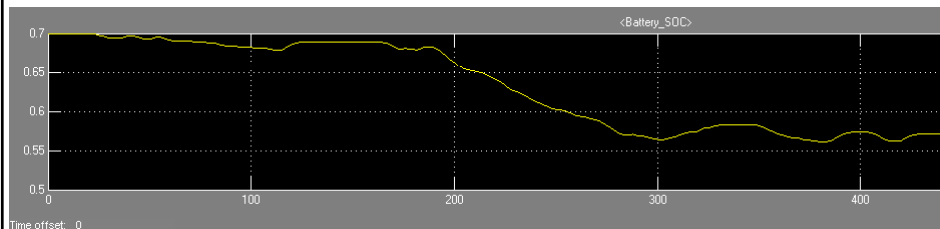
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

The Need

54

- Thus far our vehicle has been able to follow various drive cycles using the motor and battery.
- The battery SOC has decreased, but never too much.



MotoTron

 **The MathWorks**

 **freescale**
semiconductor

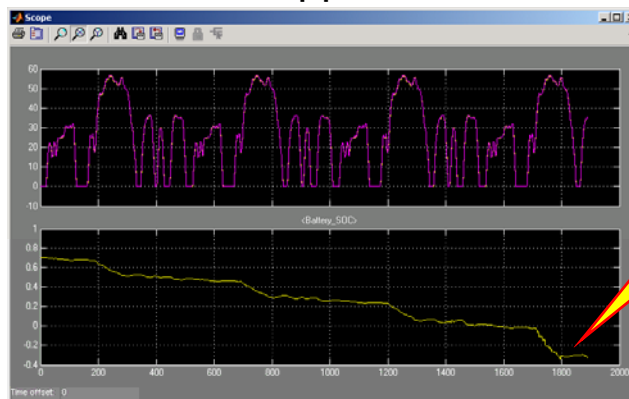
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



55

The Need

- Lets follow the 505 cycle a few times and see what happens.



Negative battery state of charge. Either we need a larger battery or we need a way to put charge into the battery.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

56

The Need

- We need some way to recharge the battery and some logic to determine when we do it.
- Our battery block has two motor ports – one for the motor and one for a generator.
- The motor does charge the battery, but since we have aerodynamic drag and the motor is less than 100% efficient, charging through the motor only delays the time at which the battery will run out of energy.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Simple Model

57

- Eventually we will create a “genset” where an engine will drive a generator that charges the battery.
- For now, we will model the “genset” as a current source that charges the battery.
- Open your latest vehicle model and rename it Lecture5_Model1.

MotoTron

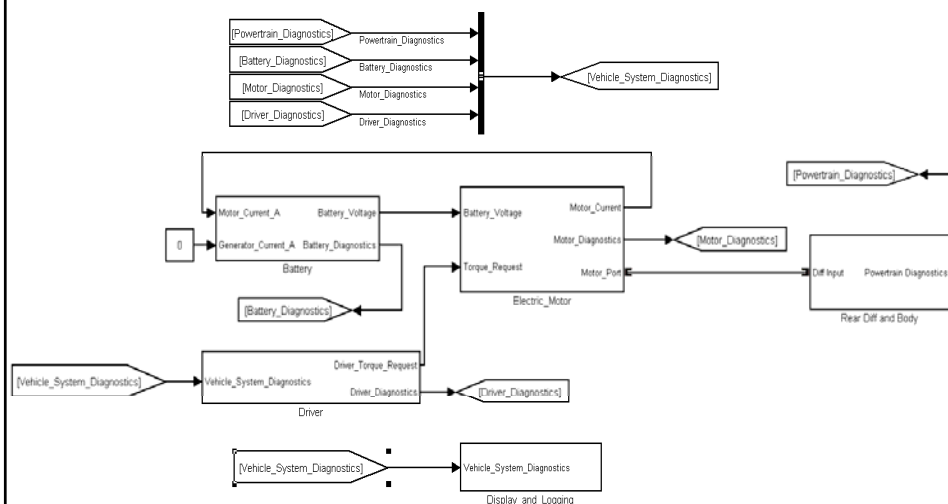
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Vehicle Model

58



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Simple Model

59

- We want to keep the SOC in a specified range.
 - No over-charging.
 - No over-discharging.
- This will require some logic to switch between a “No_Charge” state and a “Charge” state.
- We’ll build a Stateflow Controller in a new subsystem called “Controller.”



Controller Subsystem

60

- We will add a new block to our Model called “Controller.”
- Eventually, all of the functions contained in the Controller subsystem will be implemented on a real-time target.
- This target will be the supervisory controller for our vehicle.





Controller Subsystem

61

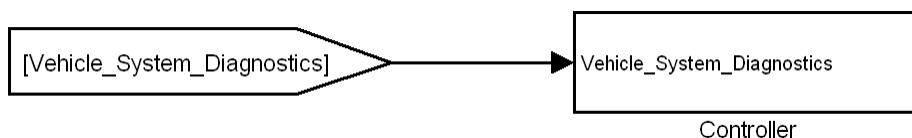
- Inputs to the control subsystem are status signals from the various vehicle subsystems (motor, battery, engine, vehicle body, etc.)
- Outputs of the control subsystem tell each subsystem what to do.
- Inputs come over the vehicle system diagnostics bus (which will be the CAN bus in a vehicle).



Control Subsystem

62

- Add a subsystem to your model called “Controller.”
- The input to this system is the vehicle system diagnostics bus.
- Hint: You may want to copy and paste the Display subsystem and rename it “Controller.”

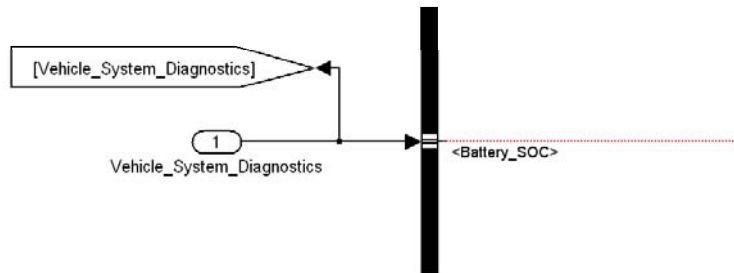




Controller Subsystem

63

- Inside the Controller subsystem, use a Bus Selector to extract battery SOC signal on the Vehicle System Diagnostic bus.
- (If you copied the Display subsystem, you already have a head start.)
- The contents of the Controller subsystem are:



Introduction to Stateflow

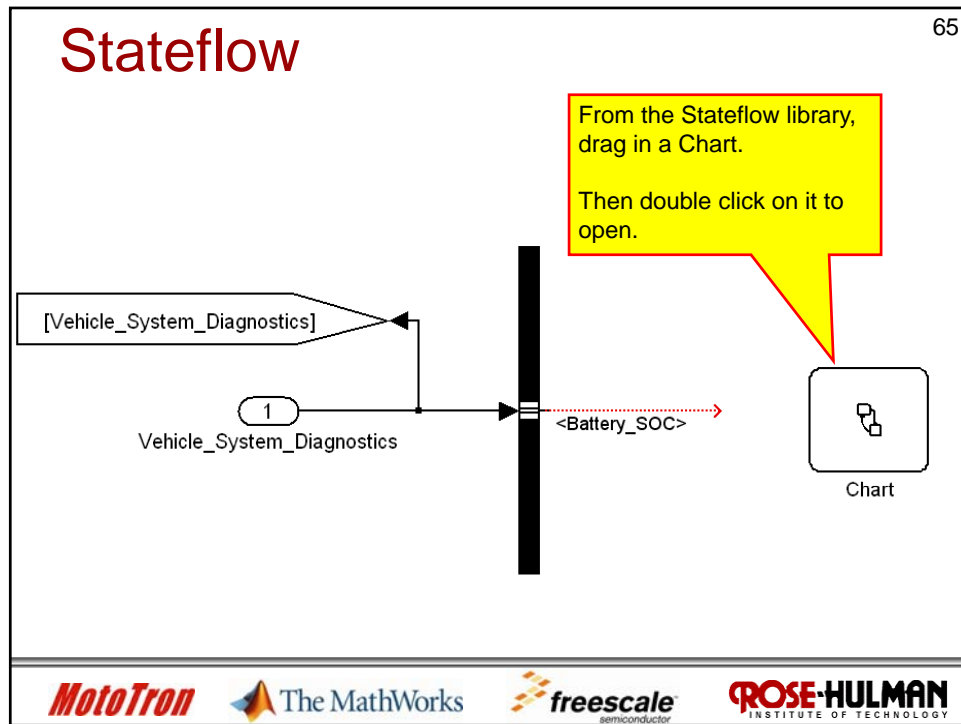
64

- We will add a Stateflow chart to the Controller subsystem.
- This chart will be used to turn on and off charging current for our system.
- Initially, the Controller output will be a fictitious current that magically charges the battery.
- Later, the controller output will turn on an Engine/Generator genset that charges the battery.



Stateflow

65



From the Stateflow library, drag in a Chart.

Then double click on it to open.

[Vehicle_System_Diagnostics]

1

Vehicle_System_Diagnostics

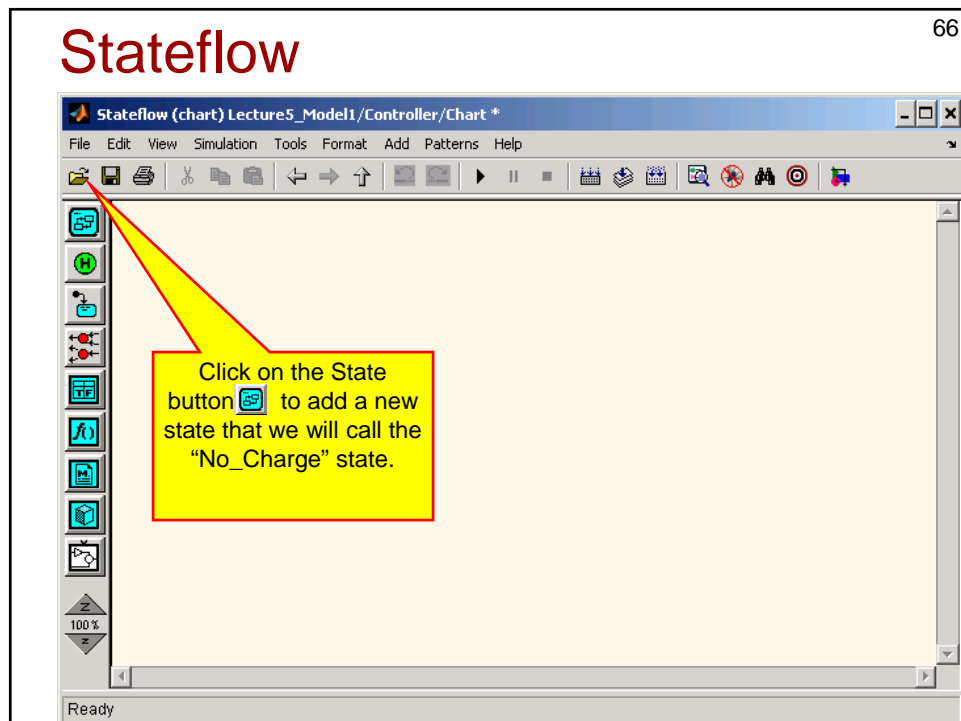
<Battery_SOC>


Chart

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Stateflow

66



Click on the State button  to add a new state that we will call the "No_Charge" state.

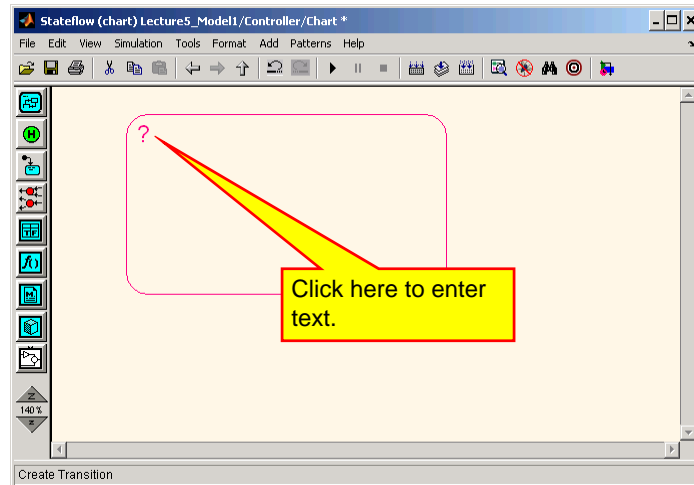
Stateflow (chart) Lecture5_Model1/Controller/Chart *

File Edit View Simulation Tools Format Add Patterns Help

Ready

Stateflow

67



MotoTron

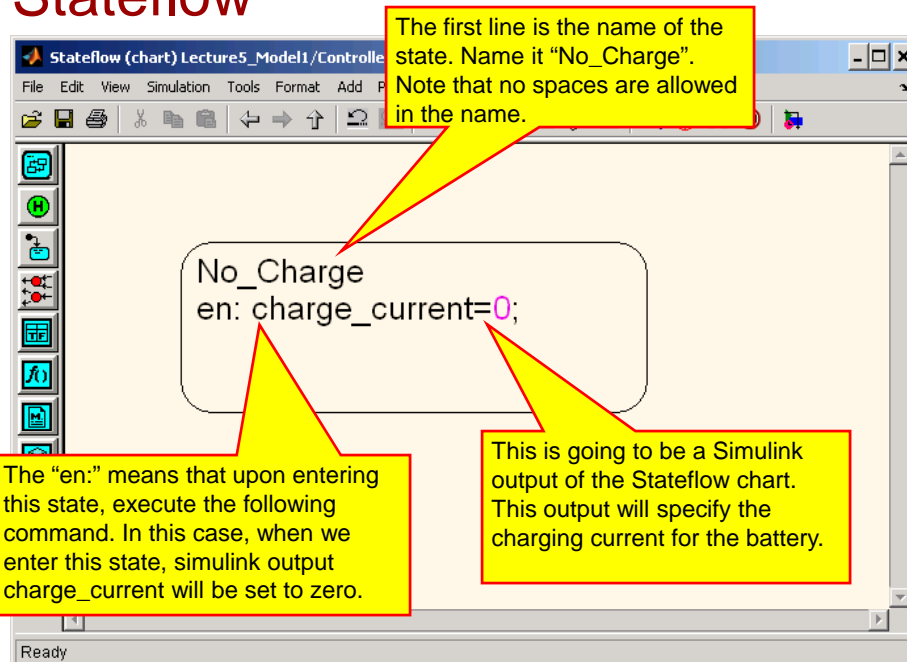
 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

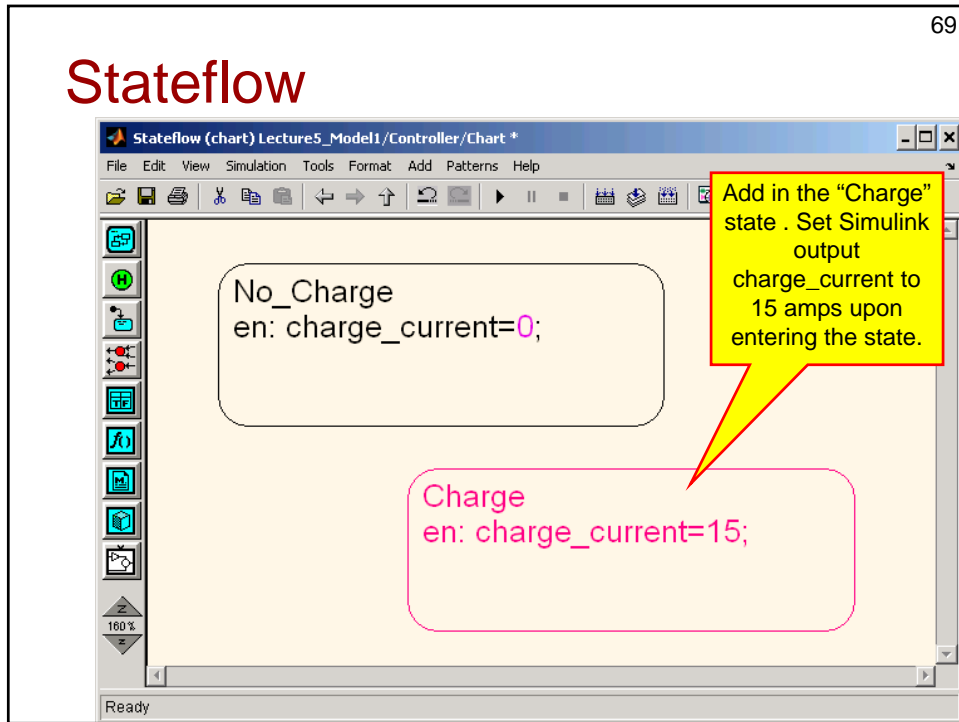
Stateflow

68



69

Stateflow



Stateflow (chart) Lecture5_Model1/Controller/Chart *

File Edit View Simulation Tools Format Add Patterns Help

No_Charge
en: charge_current=0;

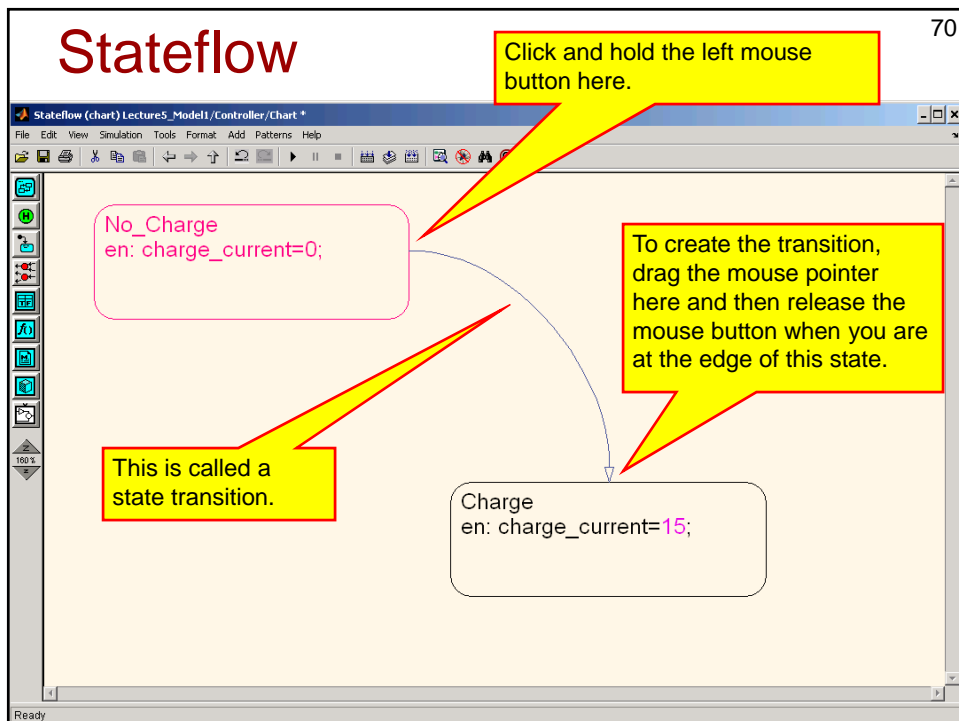
Charge
en: charge_current=15;

Add in the "Charge" state . Set Simulink output charge_current to 15 amps upon entering the state.

Ready

70

Stateflow



Stateflow (chart) Lecture5_Model1/Controller/Chart *

File Edit View Simulation Tools Format Add Patterns Help

No_Charge
en: charge_current=0;

Charge
en: charge_current=15;

Click and hold the left mouse button here.

To create the transition, drag the mouse pointer here and then release the mouse button when you are at the edge of this state.

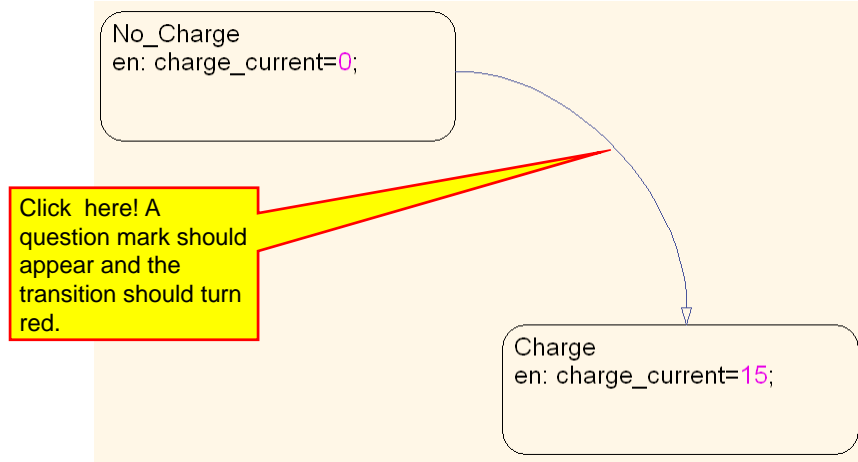
This is called a state transition.

Ready

Stateflow

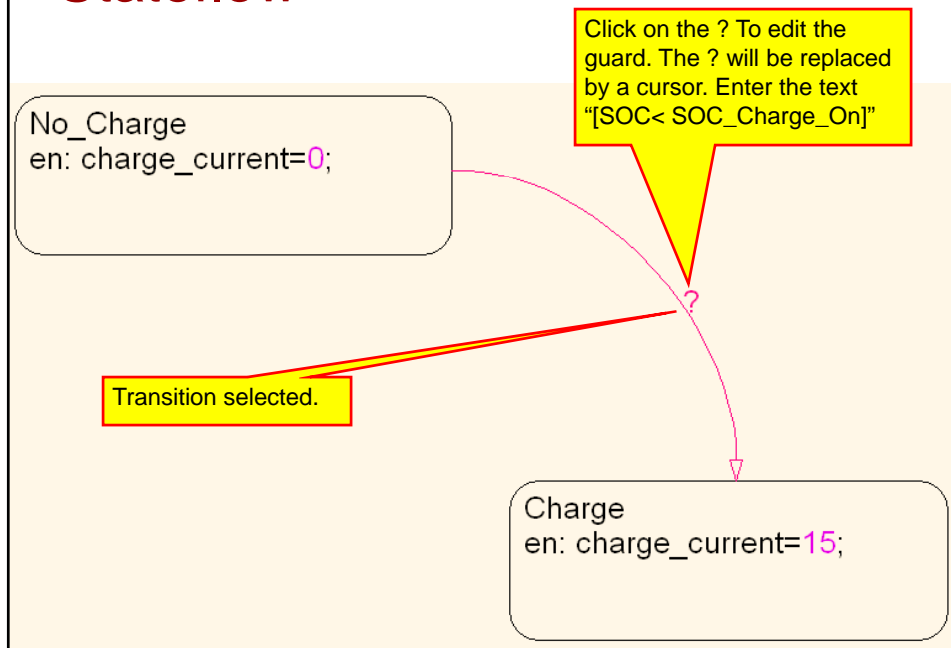
71

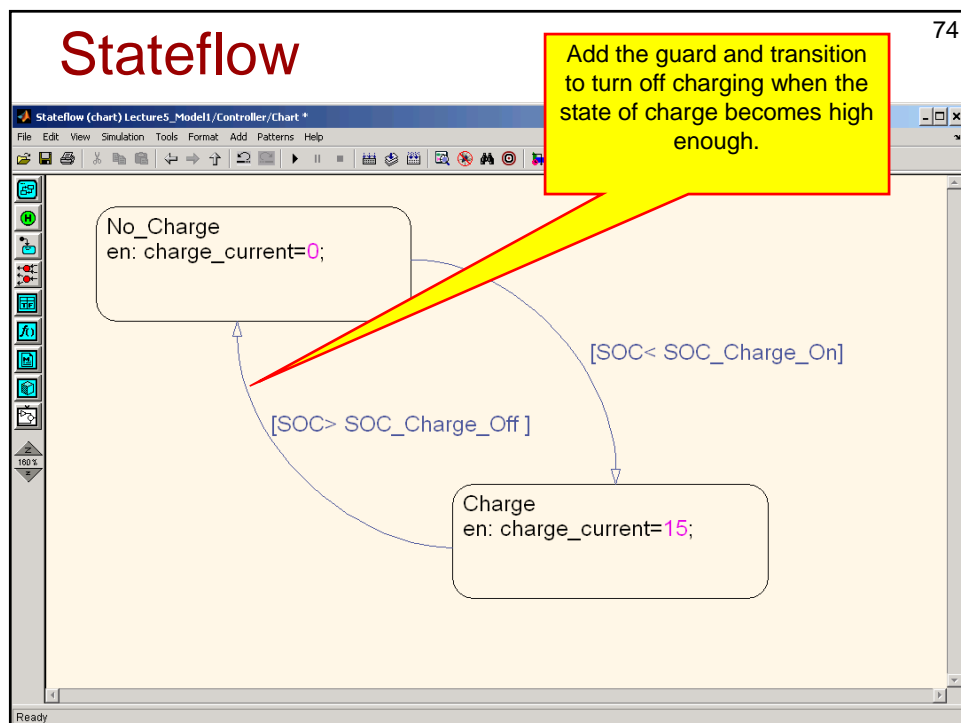
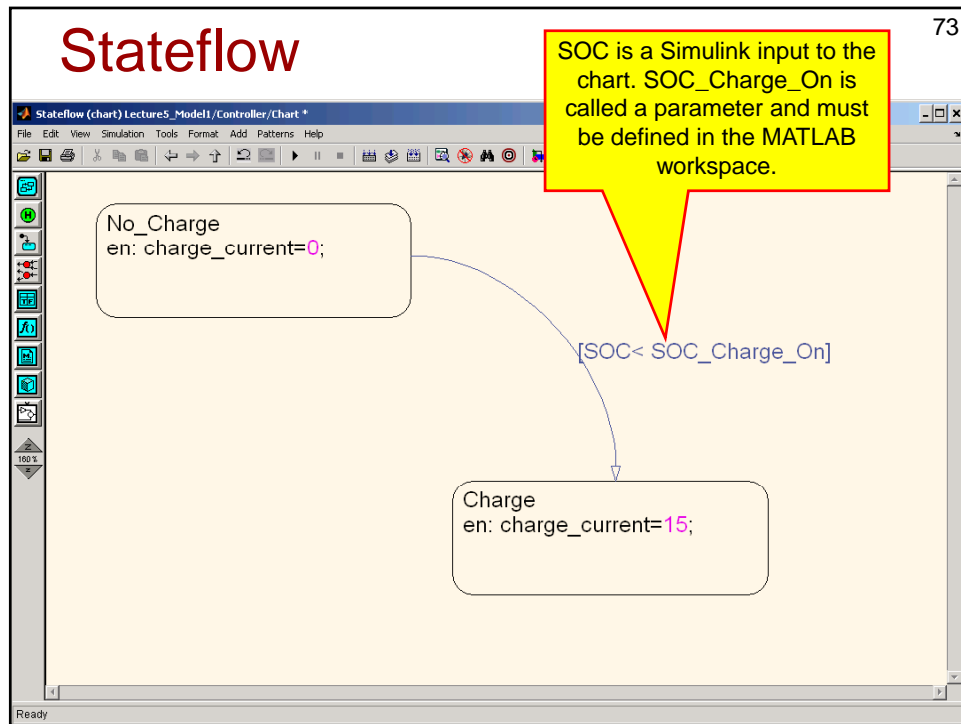
- We need to specify the “guard” for the transition. Click the left mouse button on the transition:



Stateflow

72









Stateflow

75

- The last thing we need to do is to specify in which state the chart wakes up when we first start the model.
- This is done by clicking on the Default Transition button  and connecting the transition to the desired initial state.
- When you place the Default Transition part, it will be connected to a circle. 
- Delete the circuit and connect the transition to the No_Charge state:

MotoTron

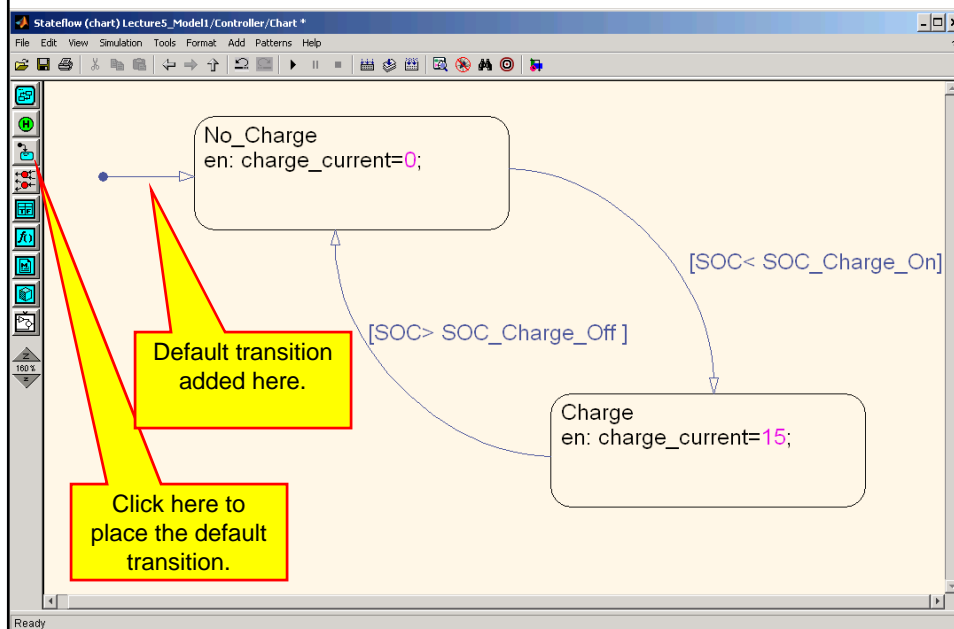
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Stateflow

76

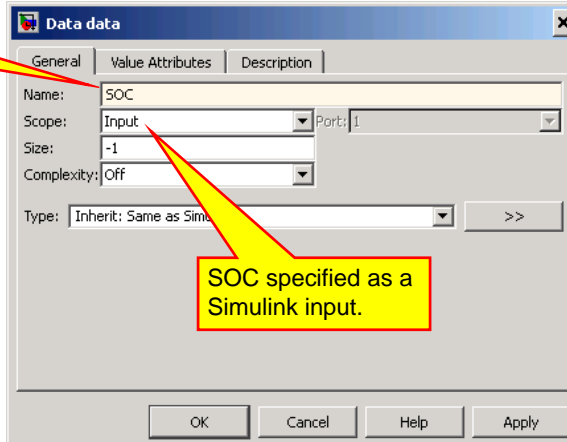


Stateflow

77

- Next, we need to specify SOC as an input from Simulink
- Select **Add, Data**, and then **Input from Simulink** from the Stateflow menus

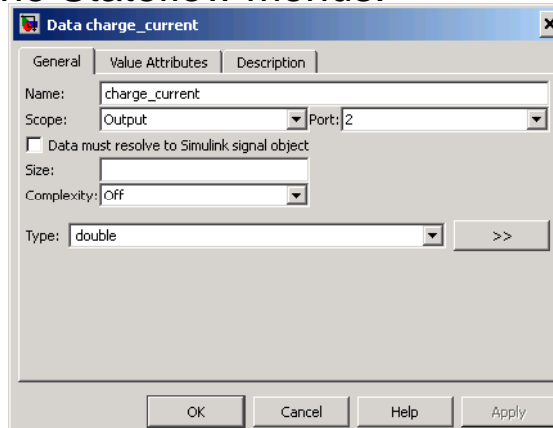
Name of the data is "SOC."



Stateflow

78

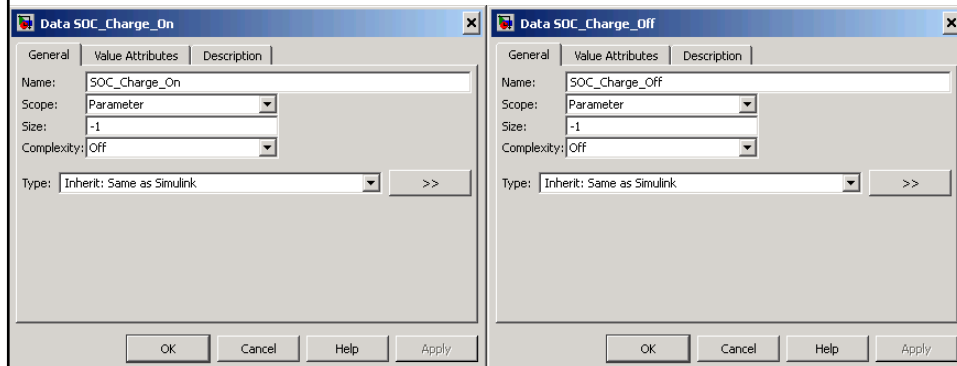
- We need to specify charge_current as a Simulink output
- Select **Add, Data**, and then **Output to Simulink** from the Stateflow menus:



Stateflow

79

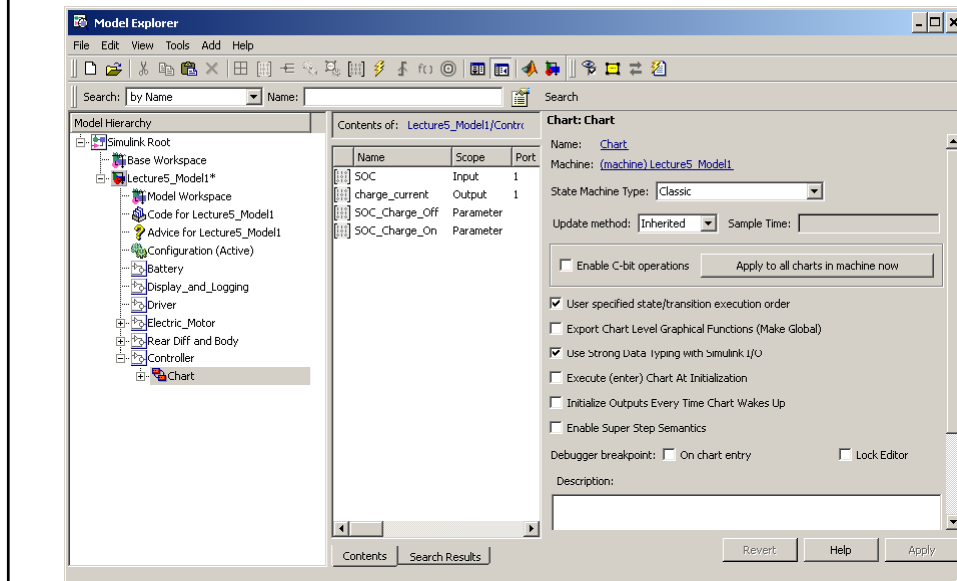
- We need to specify SOC_Charge_On and SOC_Charge_Off as parameters (which will be read in from the MATLAB workspace).
- Select **Add, Data, Parameter** from the Simulink menus twice:



Stateflow

80

- Use the Model Explorer to check your work:





Stateflow

81

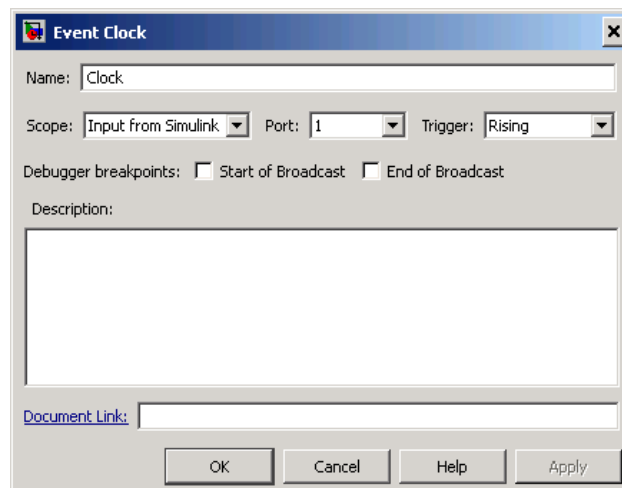
- Stateflow will only check guards and follow transitions when an event occurs.
- We will use the zero-crossing of a sinec wave source as the event and call it a “Clock.”
- Select **Add, Event, Input From Simulink** from the Stateflow menus to add an event input to your chart.
- This input will come from Simulink.



Stateflow

82

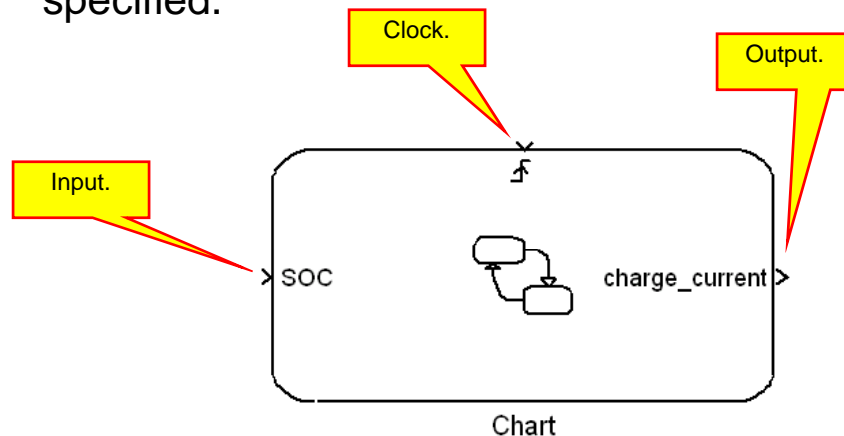
- Name the event clock:



Stateflow

83

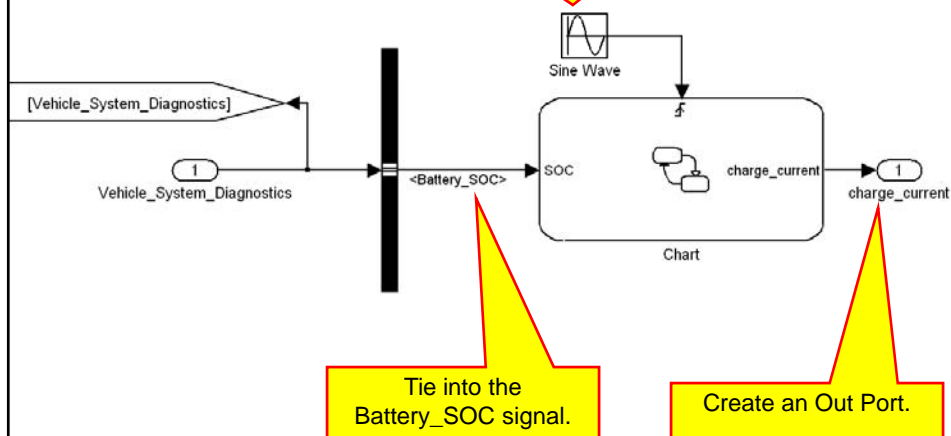
- When you close the chart, you will see the Simulink inputs and outputs that we specified.

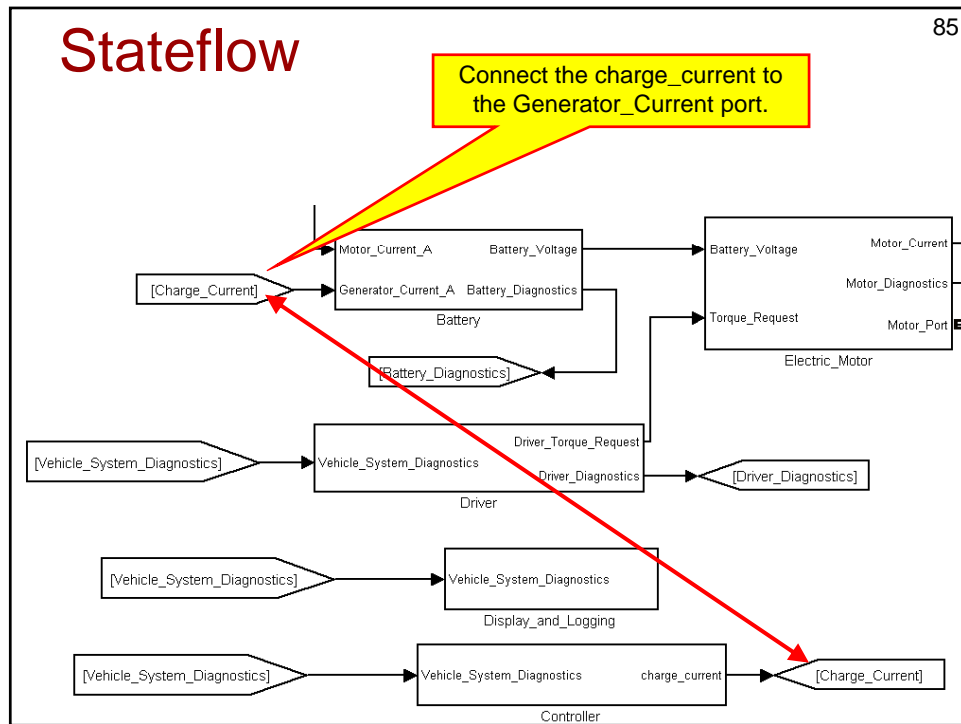


Stateflow

84

Use a sine wave for the clock with an amplitude of 1 and a frequency of $100 \times (2\pi)$ to get a 10 ms frequency.





Stateflow

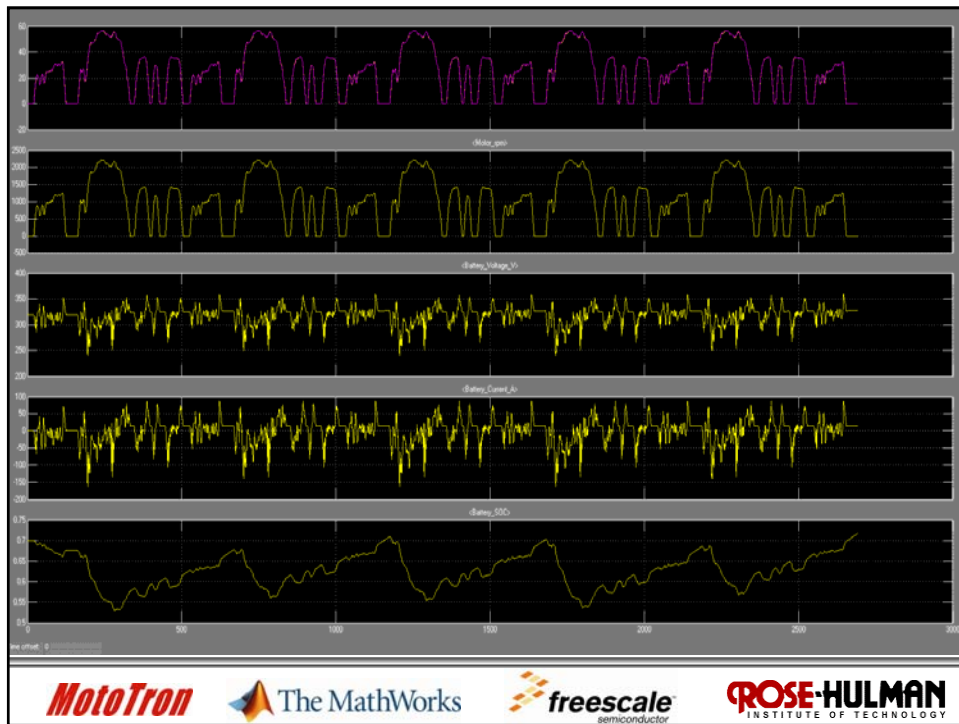
86

- Update the Init file with the guard values of 0.6 to start charging and 0.7 to stop.

```

%Controller Constants
Charging_Engine_rpm = 1800; %Constant engine rpm for charging.
SOC_Charge_On = 0.6; %Value of SOC where we start charging.
SOC_Charge_Off = 0.7; %Value of SOC where we stop charging.
    
```

- Run the FU505 five times:



Lecture 5 Demo 2

88

- Demo of Stateflow controller charging the battery.

Demo_____



Except where otherwise noted, this work is licensed under

<http://creativecommons.org/licenses/by/3.0/>



Advanced Model-Based-System Design

Lecture 6: Creating a Genset and Building a Formal Controller

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

2

Building a Formal Controller

- Create a Engine/Generator “genset” to charge the battery.
- Add engine speed control.
- Engine starting and stopping.
- Start with the previous model.

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Complete Model

3

- Copy the electric motor model and rename it “Generator.”
- We will use the same model for the motor and generator.
- If we want different properties for the motor and generator, we can use the same “motor” block but use different lookup tables to give the motor different properties.

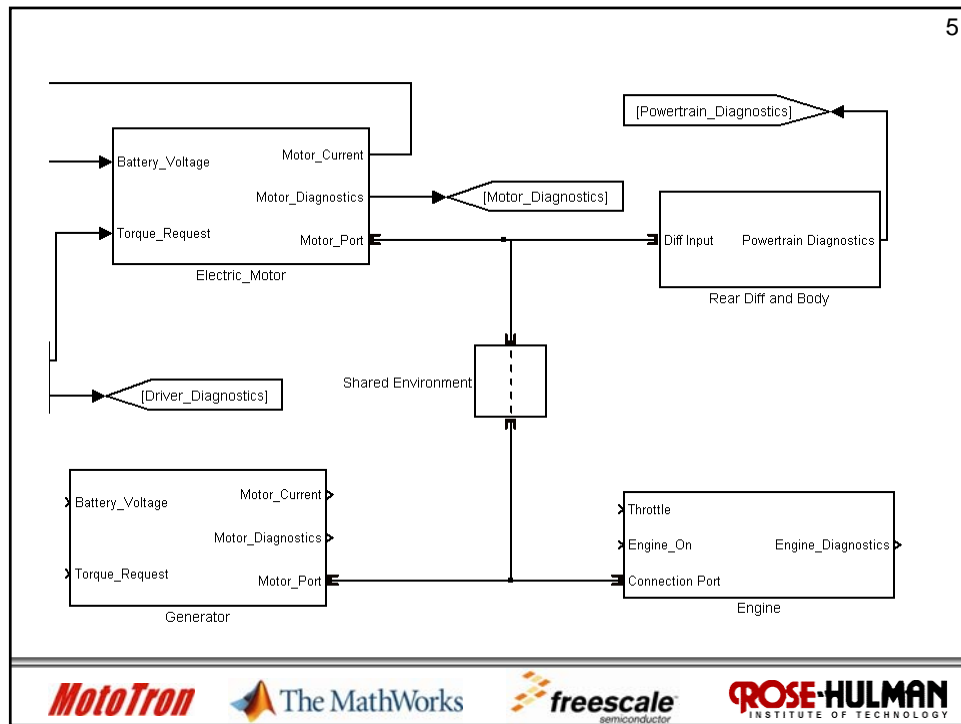


Complete Model

4

- Copy the engine model subsystem we created earlier and place it in the model.
- Make connections as shown.
- Connect the drive lines with the shared environment block.



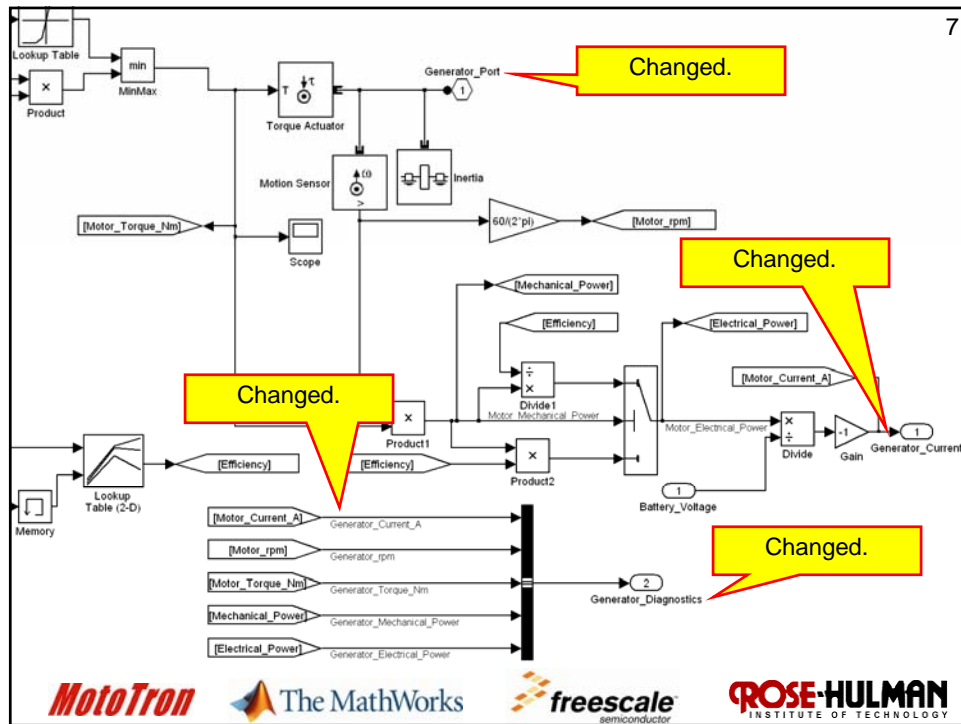


6

Modify the Generator Subsystem

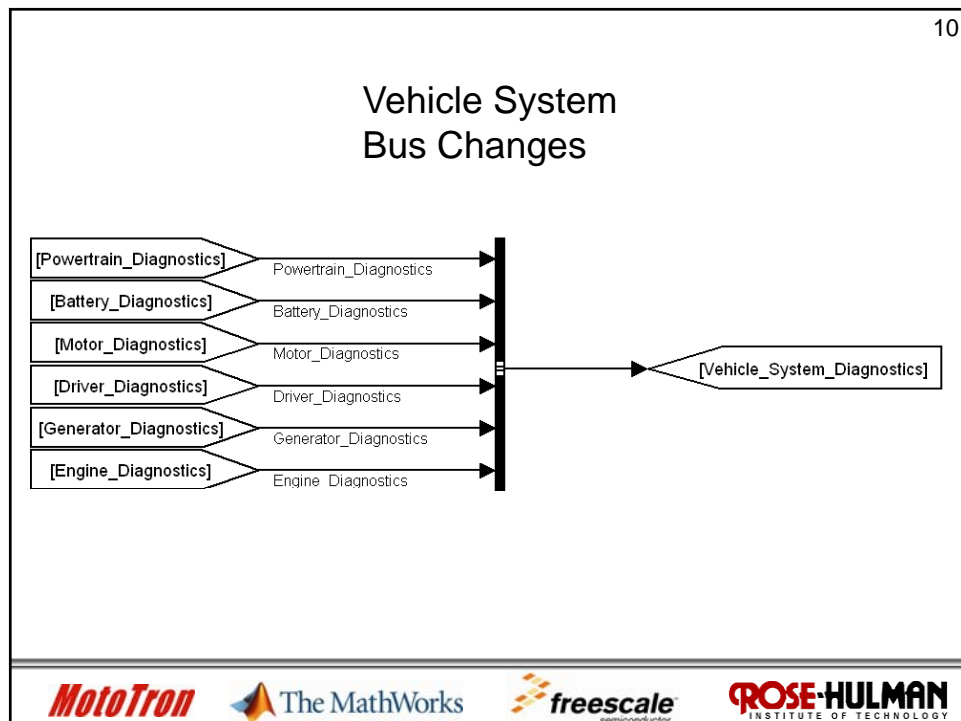
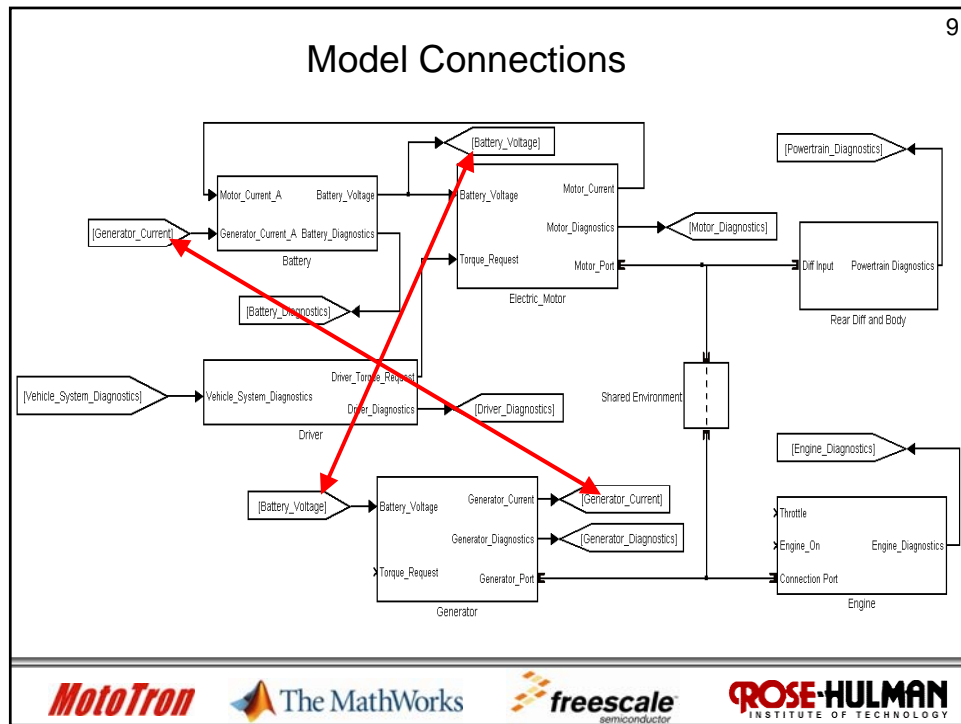
- Change the names of ports from “Motor” to “Generator.”
- Change the name of signals on the bus from “Motor” to “Generator.”

MotoTron The MathWorks freescale ROSE-HULMAN
 INSTITUTE OF TECHNOLOGY



Model Connections

- Connect the Generator to the Battery voltage.
- Connect “Generator” current on the Battery to the Generator Current
- Connect the Engine and Generator Diagnostic Ports to the Vehicle_System_Diagnostics bus.





Control System Design

11

- We now have the system to a point where we can start to build our control system.
- The electric motor drives the vehicle.
- When necessary, the engine can be started with the generator and then charge the battery.



Engine Speed Control

12

- We will first develop a method to start the engine and control the engine speed.
- We have a generator directly connected to the engine. This generator can act either as a motor or as a generator.
- We can use motor/generator to spin up the engine to start the engine.
- We can use the motor/generator to apply a torque in the opposite direction to the engine torque to regulate the engine speed.





Engine Speed Control

13

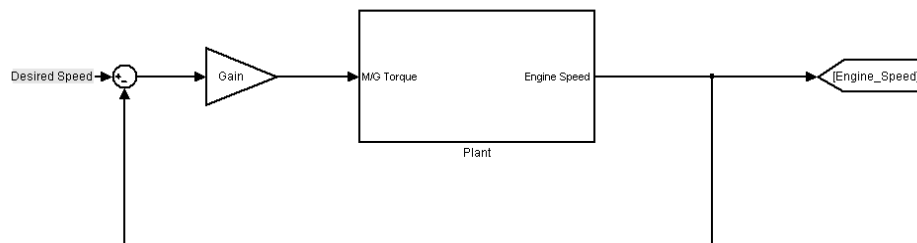
- We will use classical feedback control with proportional feedback.
- The engine throttle will be held constant.
- Monitor the engine speed.
 - If the engine speed is too slow, reduce the opposing M/G torque.
 - If the engine speed is too high, increase the opposing M/G torque.



Engine Speed Control

14

- This is a classical feedback system.



- In our case, the plant is the system comprised of the Engine coupled to the Motor/Generator.



Engine Speed Control

15

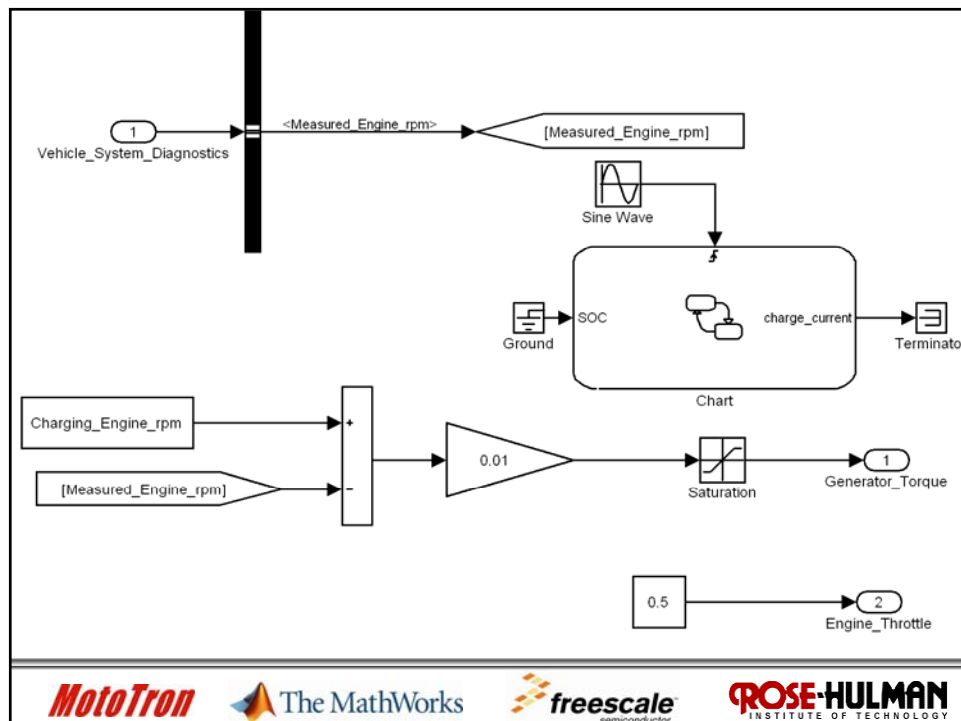
- Modify the controller as shown next.
- For the moment, we will not use the Stateflow chart.
- Note that our torque request to the M/G is constrained between -1 and +1.
- We will pick an arbitrary value for the engine throttle.
- Charging_Engine_rpm is a constant defined in the init file and is 1800 rpm.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

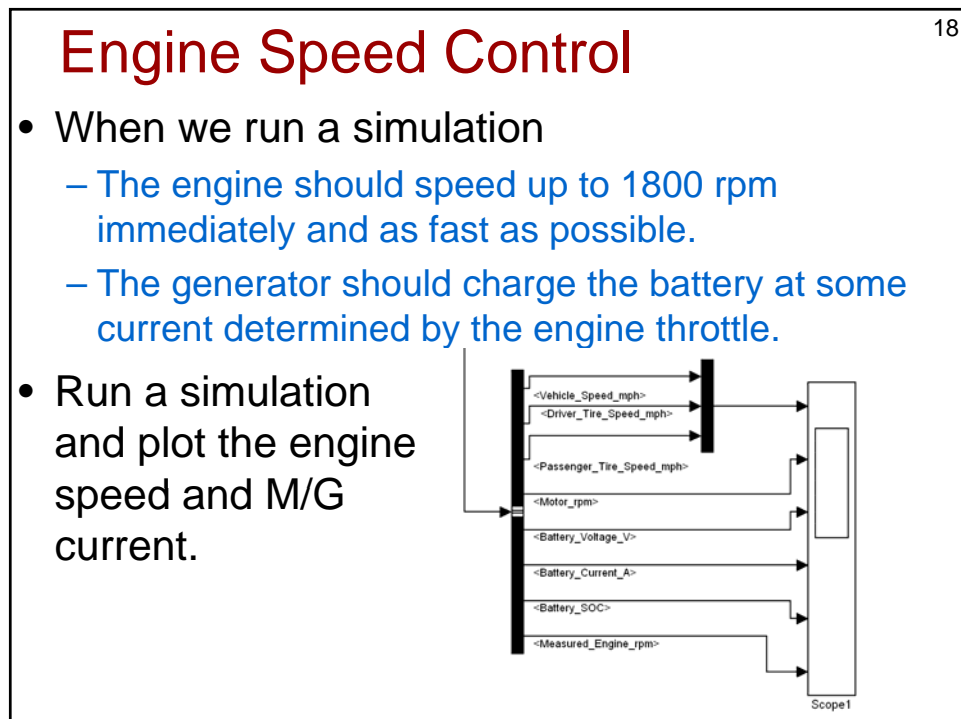
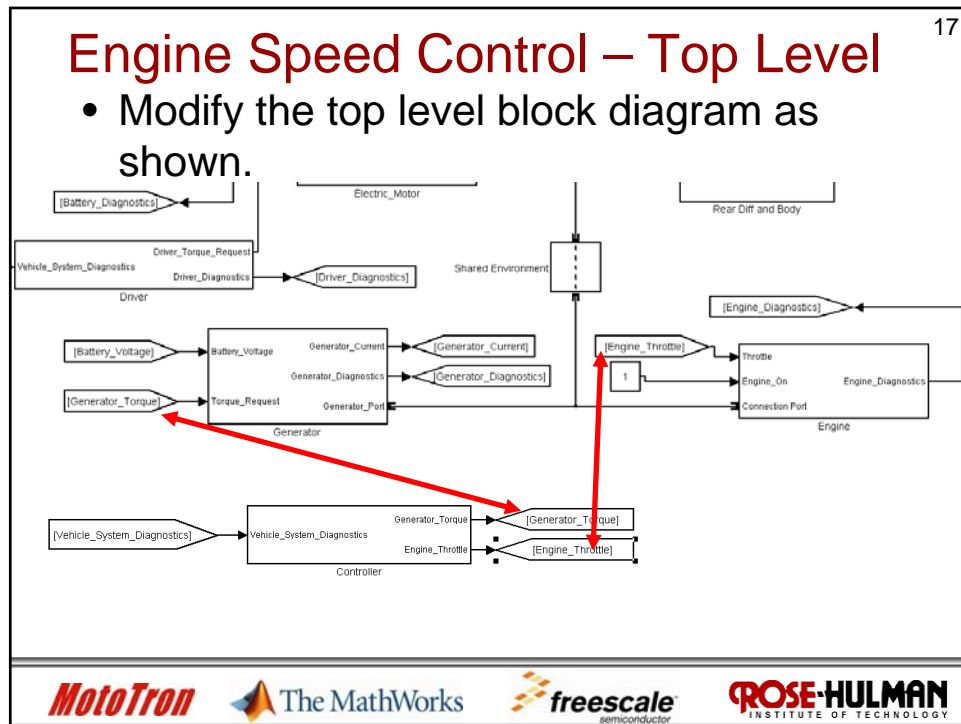


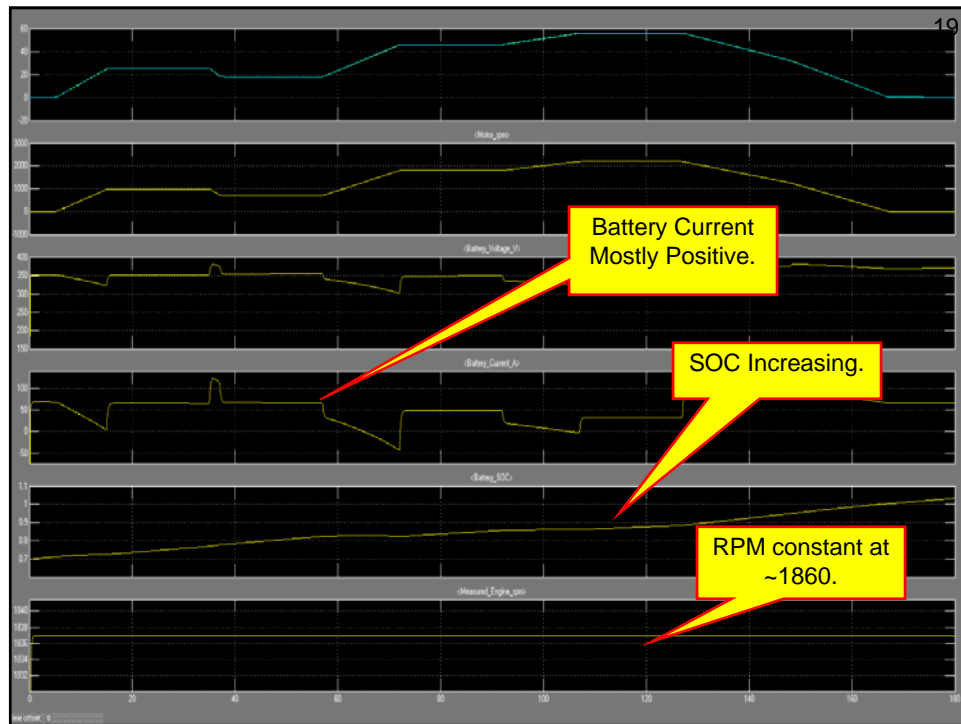
MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

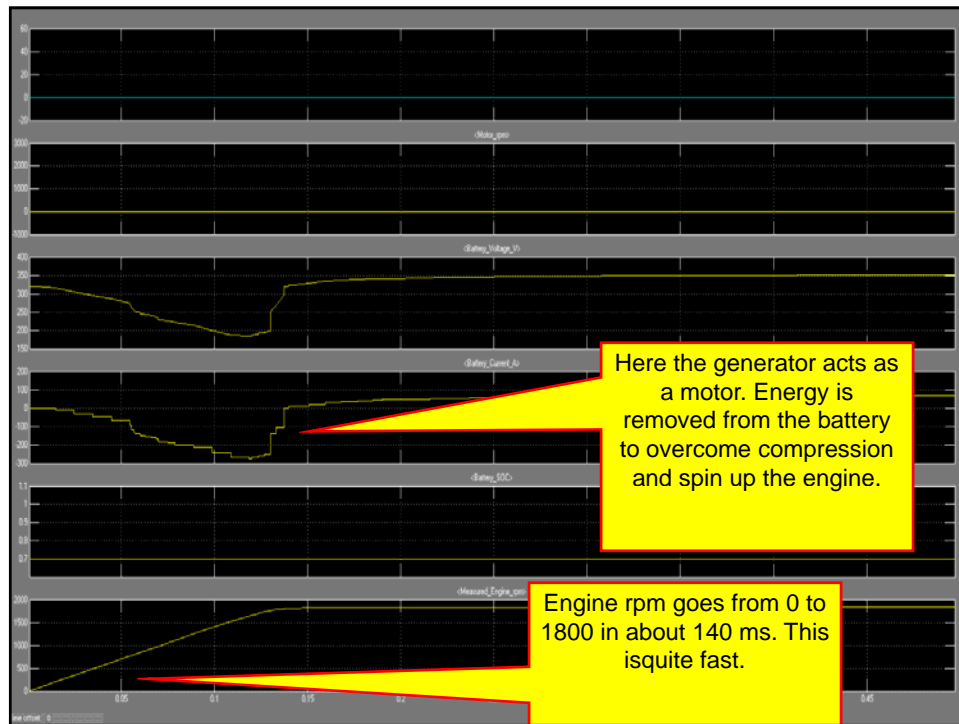




Engine Speed Control

20

- We see that the feedback loop does control the engine rpm very well, and that the generator charges that battery.
- If we zoom in on the engine rpm at the beginning of the simulation, we see that the rpm ramps up from 0 to 1800 rpm very quickly.



Engine Speed Control

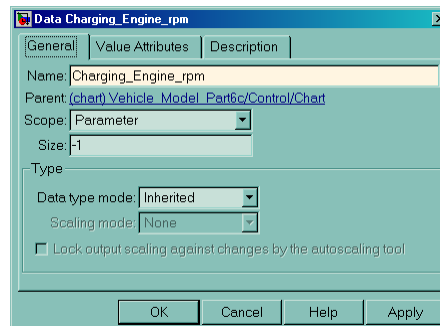
22

- There are a few issues with the controller that we must fix.
- We need to modify our control scheme to:
 - Turn on the engine only when necessary.
 - Ramp up engine speed in a controlled ramp.
 - Turn on the engine when it reaches the appropriate speed.
 - Ramp down the engine when we no longer need to charge.

Stateflow Engine Control

23

- Read Charging_Engine_rpm from the workspace. From Stateflow select **Add, Data,** and then **Parameter**



MotoTron

 **The MathWorks**

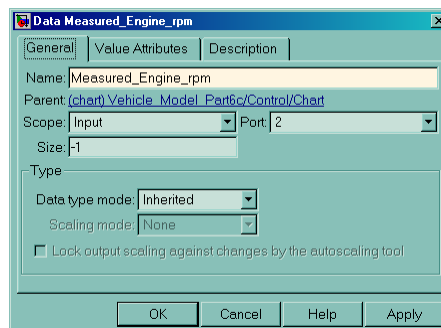
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Stateflow Engine Control

24

- We will need to know the measured Engine rpm from the Simulink model. From Stateflow select **Add, Data,** and then **Input from Simulink**

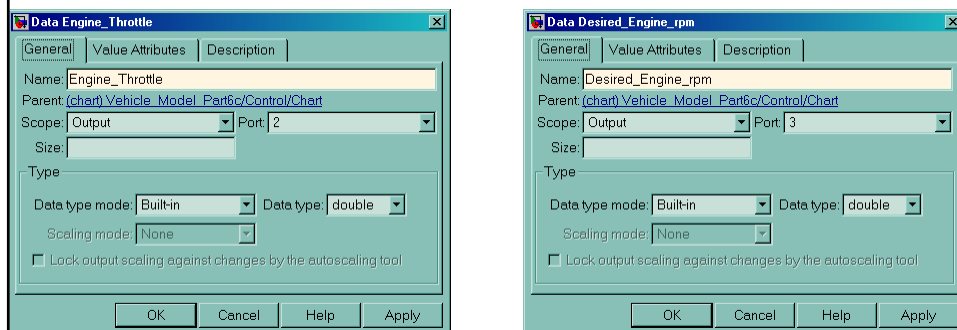




Stateflow Engine Control

25

- Stateflow will need to output the Engine Throttle and Desired Engine rpm to our Simulink controller. From Stateflow, select **Add, Data**, and then **Output to Simulink**



Stateflow Engine Control

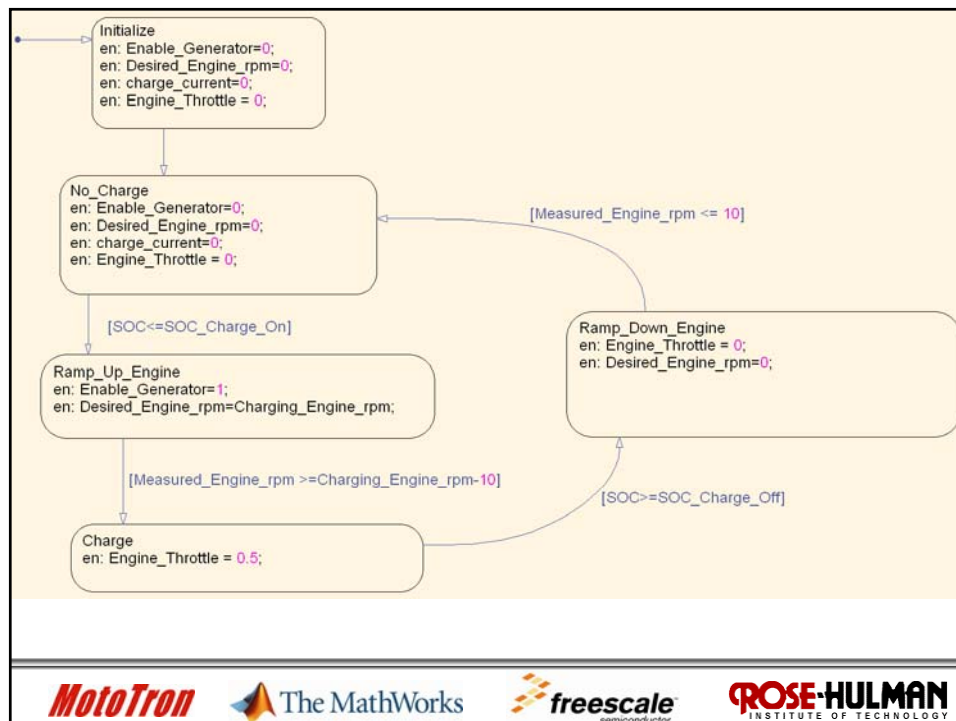
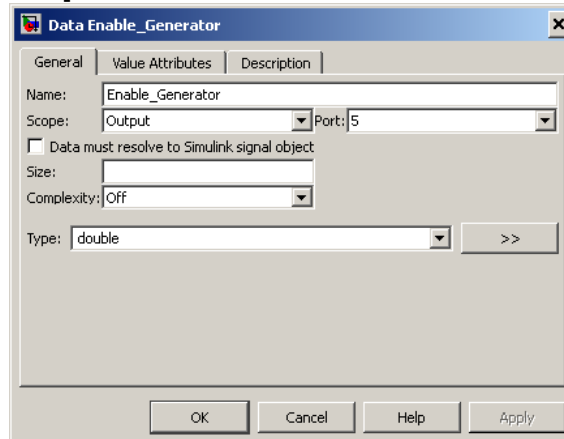
26

- Next, modify the Stateflow chart.
- When we need to charge
 - Enable Motor/Generator.
 - Change the Desired Engine rpm to the value of constant Charging_Engine_rpm.
 - When the engine reaches this rpm, change the throttle from 0 to a specified value.
- When we need to stop charging
 - Change the throttle to 0.
 - Change the Desired Engine rpm to 0.
 - When rpm reaches 10 rpm, disable motor/generator.

Stateflow Engine Control

27

- Oops, we need another output to enable the Motor/Generator. From Stateflow select **Add, Data**, and then **Output to Simulink**.

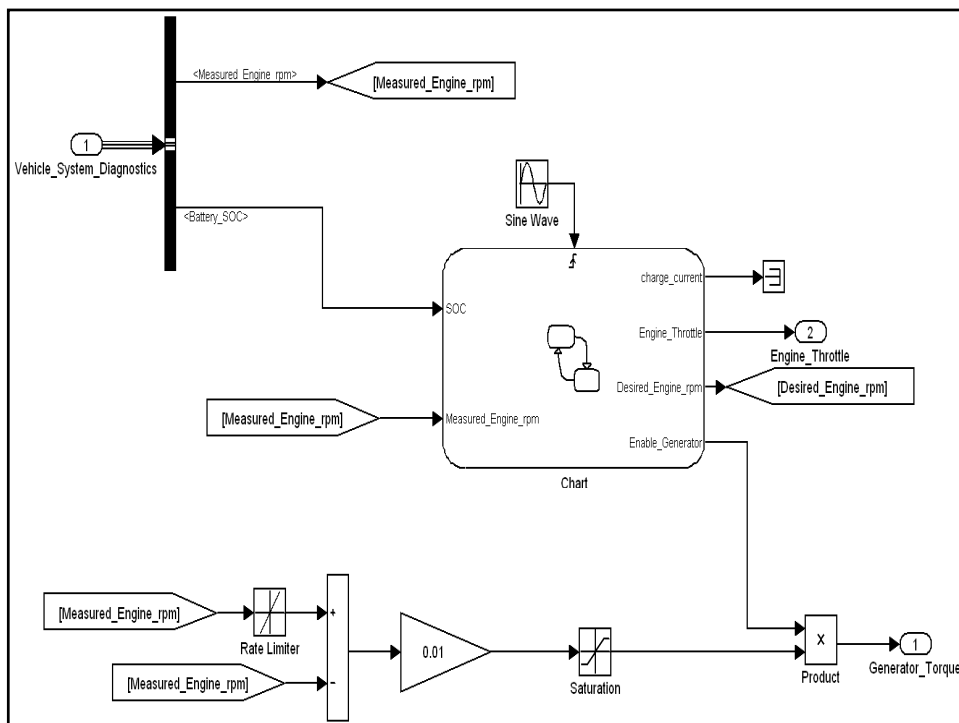


MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

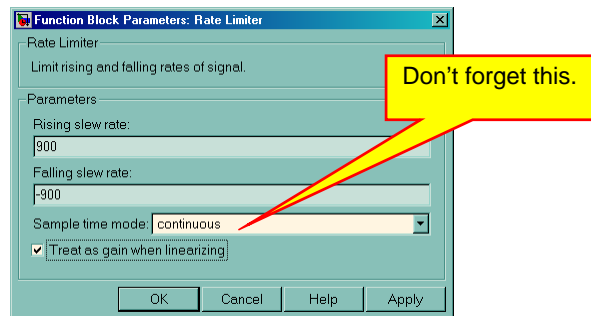


Rate Limiter

- The rate limiter in the previous slide is used to generate a ramp from the stepped signal coming from Stateflow.
- A rate limiter specifies the maximum positive and maximum negative rates at which a signal can change.
- Specifying rates of ± 900 will cause the engine rpm to ramp up from 0 to 1800 or down from 1800 to zero in 2 seconds.
- ➔ Our engine turn-on time will be 2 seconds.
- Part located in the **Simulink/Discontinuities** library.

Rate Limiter Settings

31



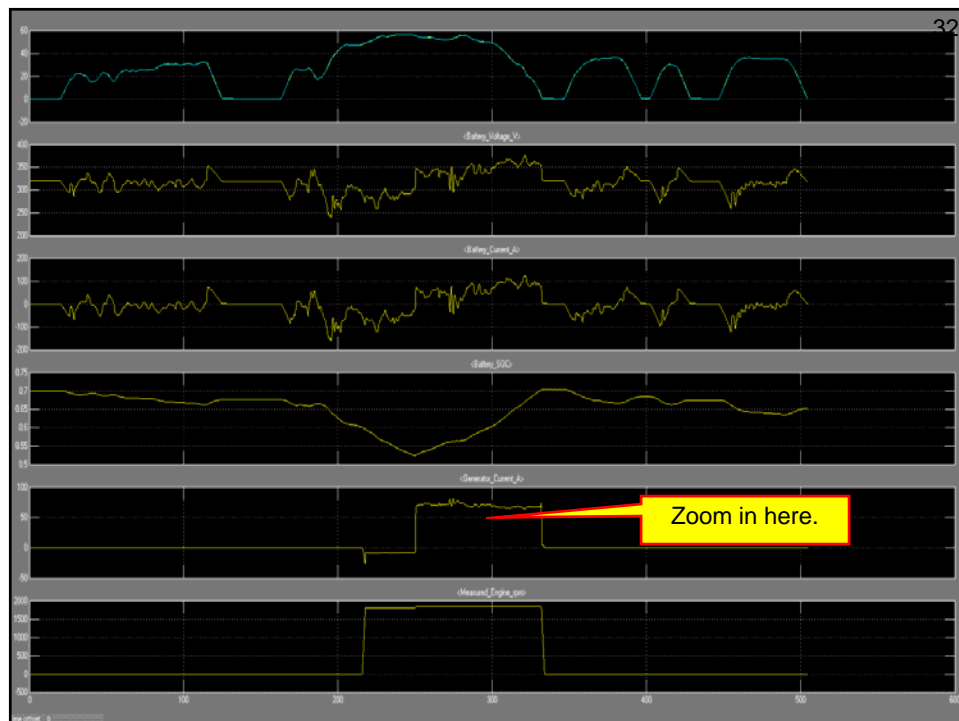
- Run a simulation first for the AVL drive cycle, then the FU505.
- Plot both the Engine rpm and the M/G current.

MotoTron

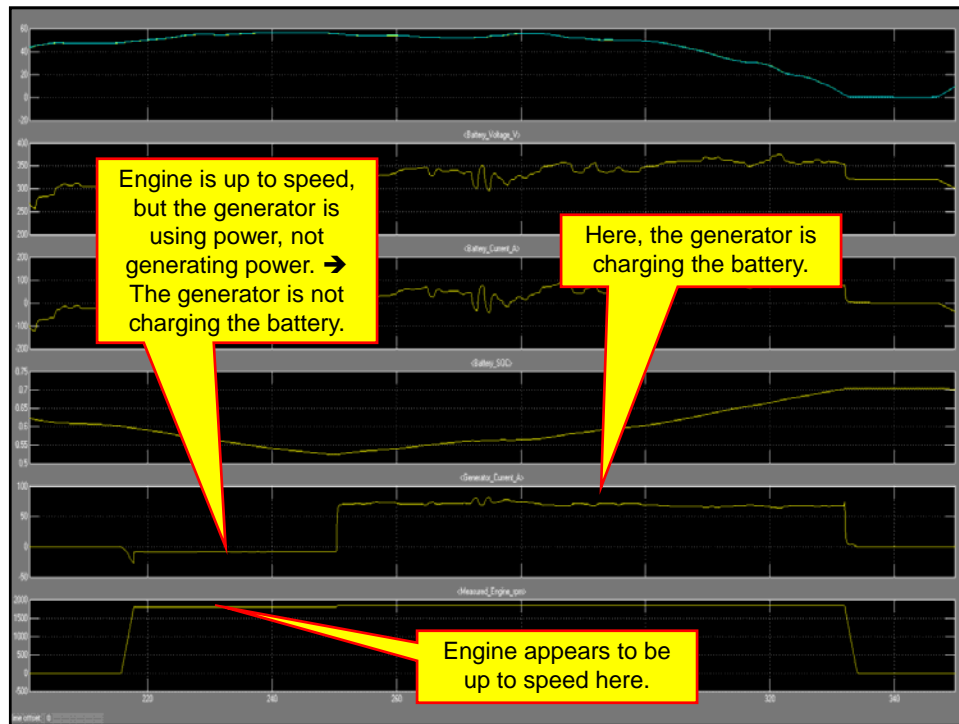
 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



32



Lecture 6 Exercise 1

Demo_____

34

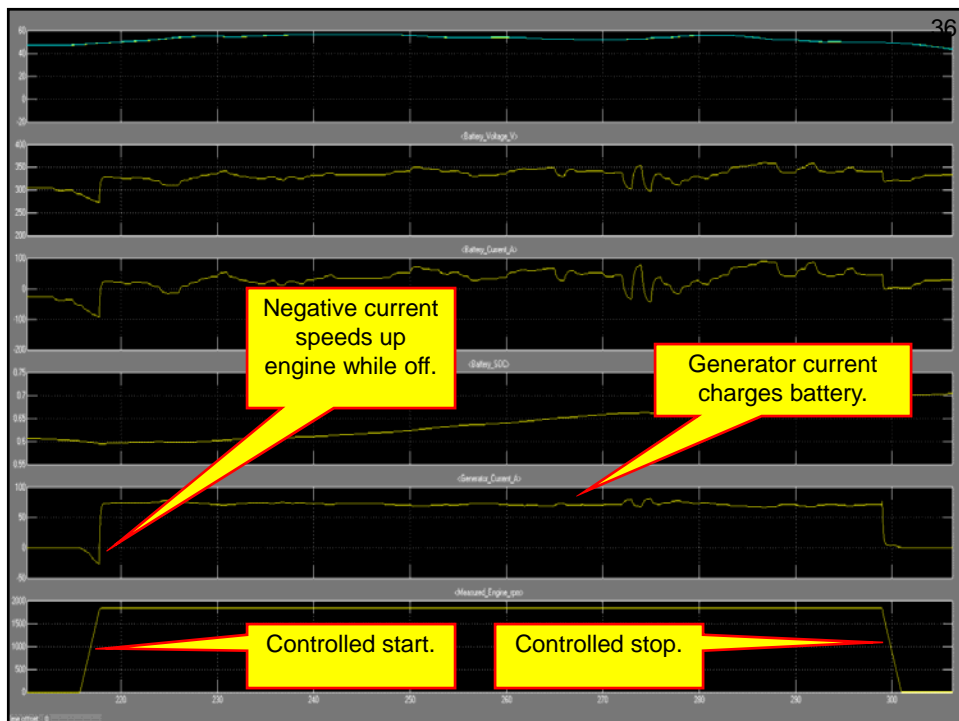
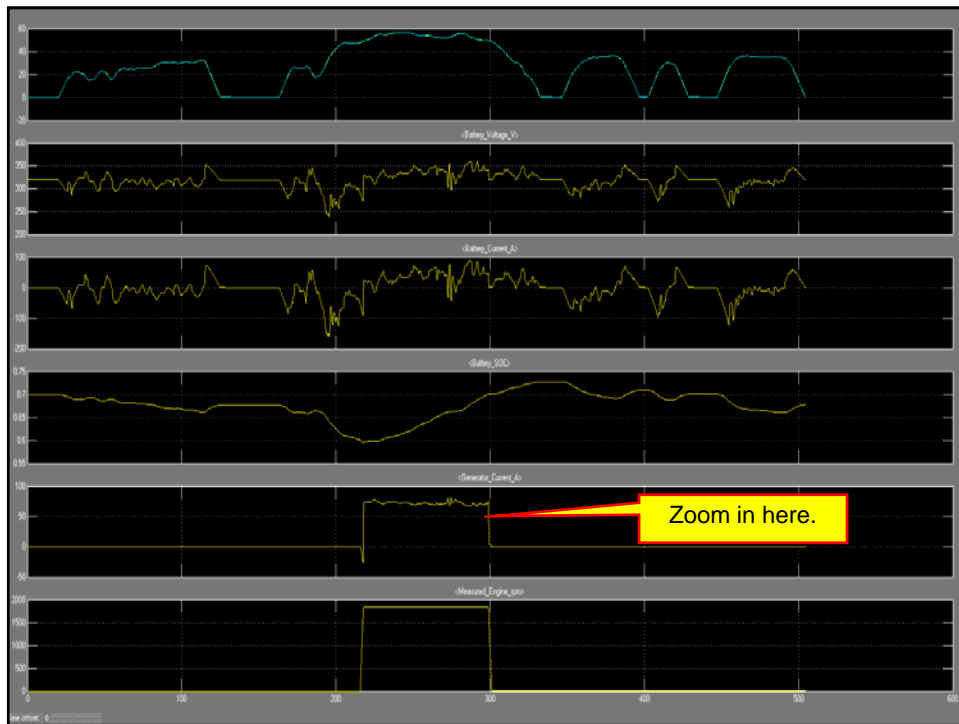
- The generator appears to spin up the engine to the appropriate speed, the generator does not immediately start charging the battery. Instead there is a long delay before charging starts. This is an error.
- Fix the error so that charging starts as soon as the generator is up to speed.
- Your fixed model should have a plot as shown next.

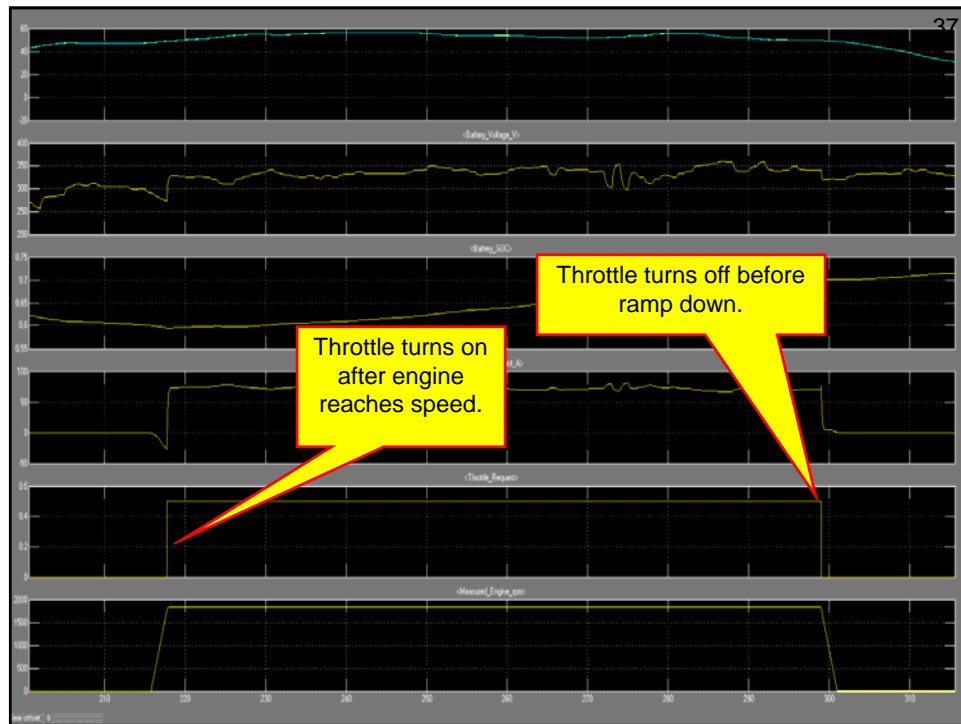
MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Lecture 6 Demo 1

38

- Demo the working controller.

Demo_____



Advanced Model-Based-System Design

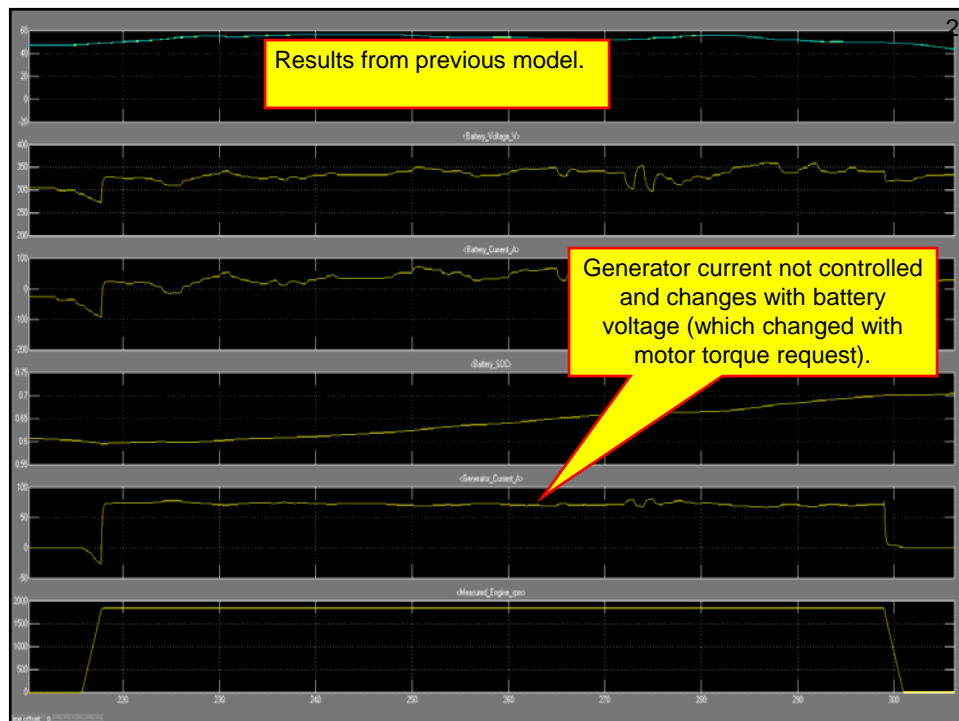
Lecture 7: Multi-Loop Control Post Processing

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





More Control Fun

3

- From the last simulation, we notice that the generator current is whatever it is, and it changes as the battery voltage changes.
- Let's add another feedback loop to control the current.
- Note that the throttle increases or decreases the engine power.
- More engine power produces more generator current.
- Less engine power produces less generator current.



Classic Feedback - Again

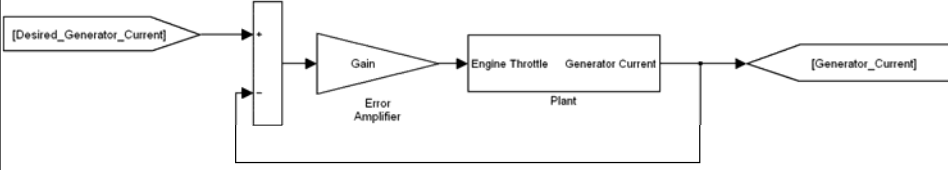
4

- We can think of the result of the system created in lecture 6 as a plant where the input signal is the engine throttle and the output is the generator current.
- By changing the throttle signal, we can control the generator current.







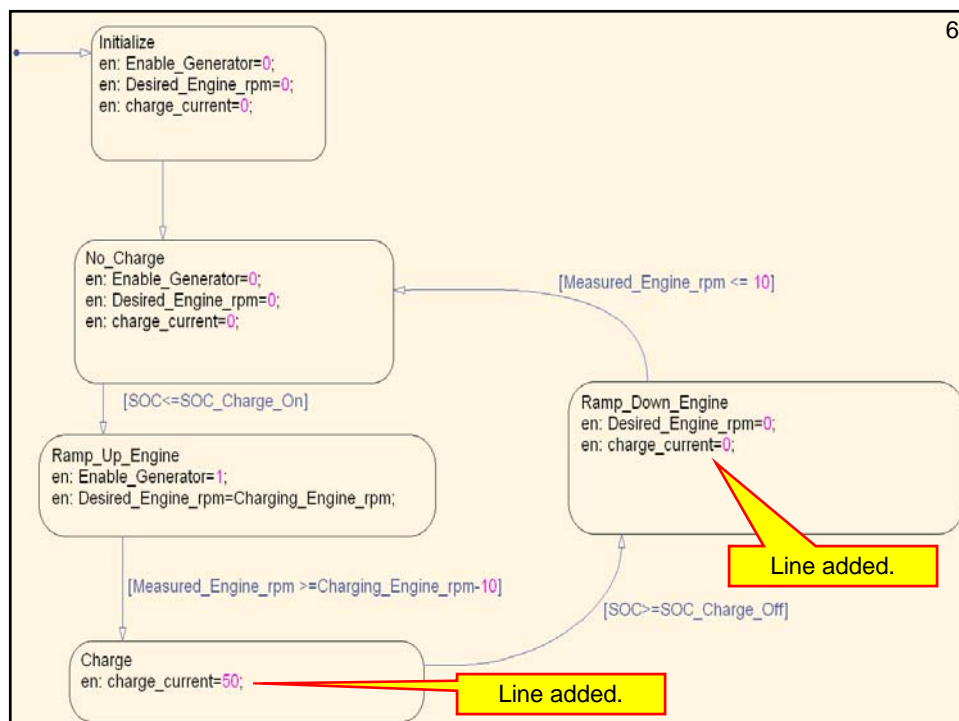
5

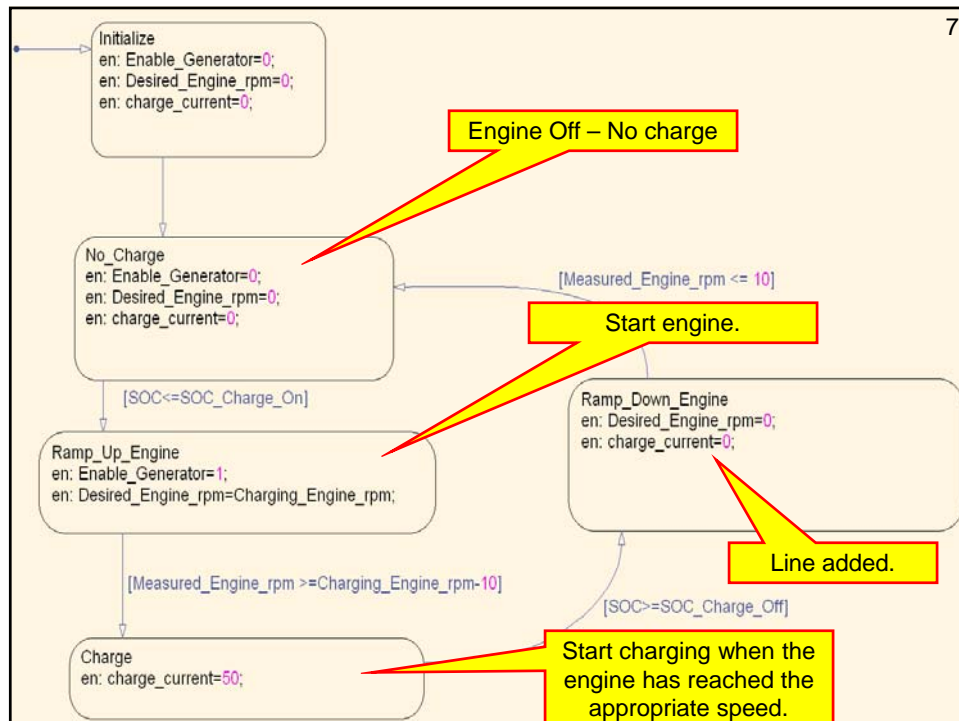
Charging Current Control



- Modify the Stateflow chart as shown next.
- Note that the Engine Throttle output has been deleted from Stateflow.
- To delete a Stateflow output, you will need to use the Model Explorer. (**Tools** and then **Explore** from the Stateflow menus.)

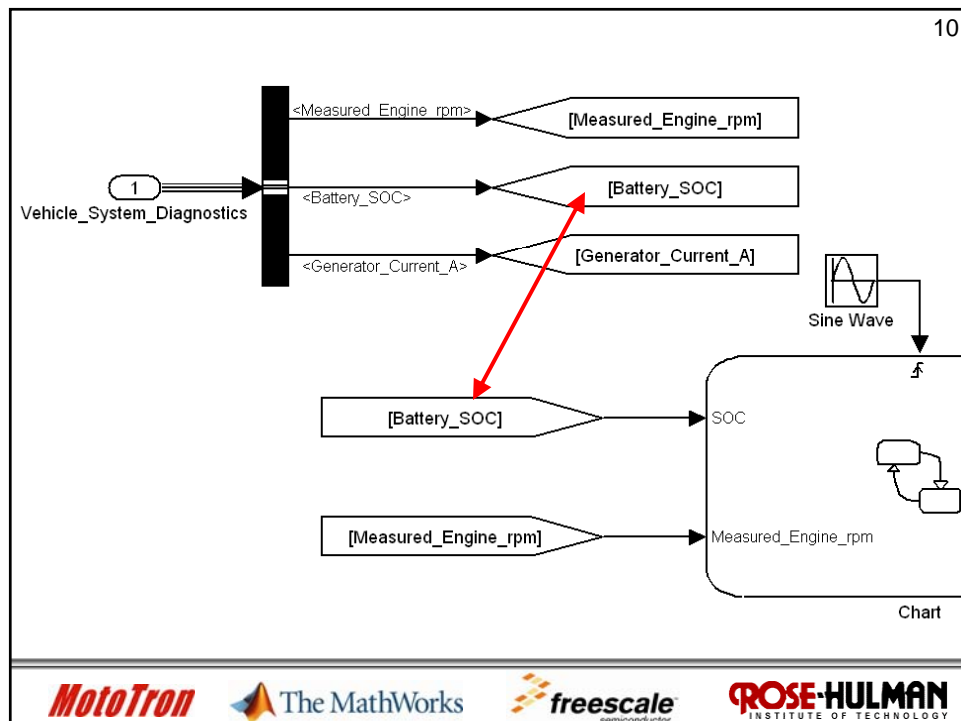
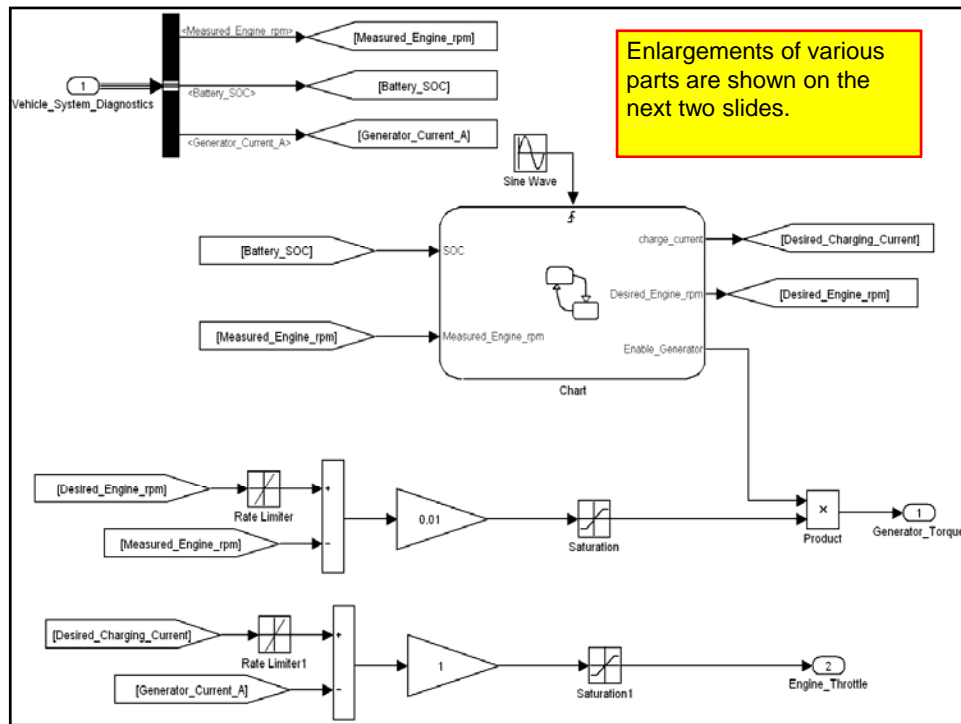


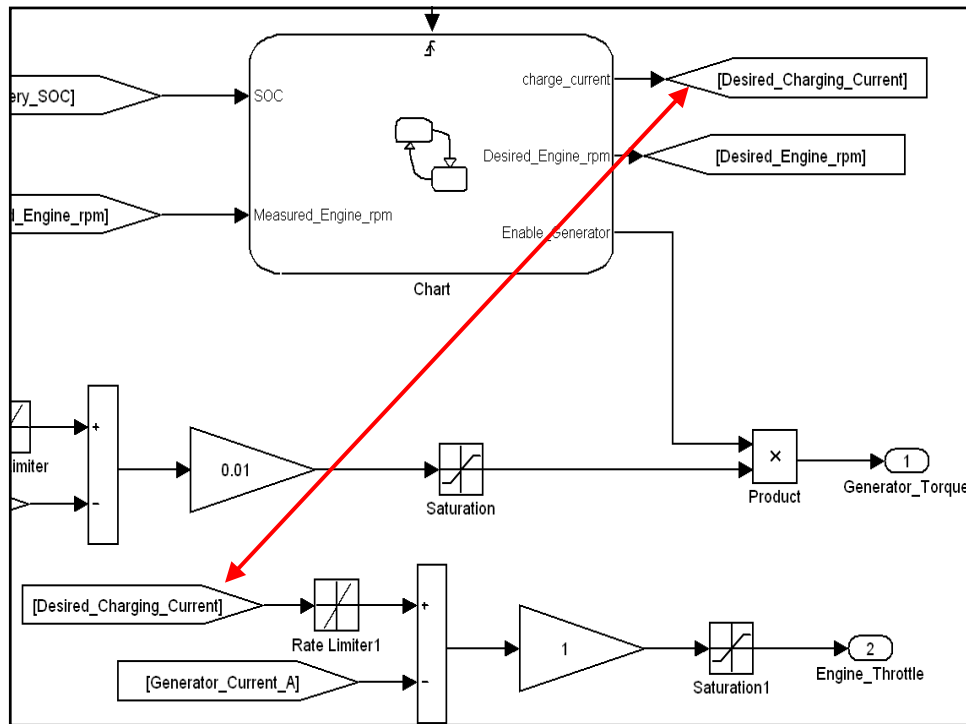


Controller Modifications

8

- Modifications to the Simulink portion controller are shown on the next slide.
- Note that we are keeping the proportional feedback loop that holds the engine speed constant at 1800 rpm.
- We are adding a second proportional feedback loop that holds the generator current constant.
- This second loop assumes that the two loops are independent and that the first loop does hold the engine speed constant.





Charging Current

12

- The signal for the charging current coming out of the Stateflow chart is a step function.
- To produce a controlled rate of change, the desired current signal is passed through a rate limiter that has a slew rate ± 50 .
- The signal out of Stateflow is a steep that goes from 0 to 50 A.
- The signal out of the rate limiter is a signal that goes from 0 to 50 A in 1 second.
- We will use the rate limiter to control the rate at which the desired charging current changes.



Saturation Block

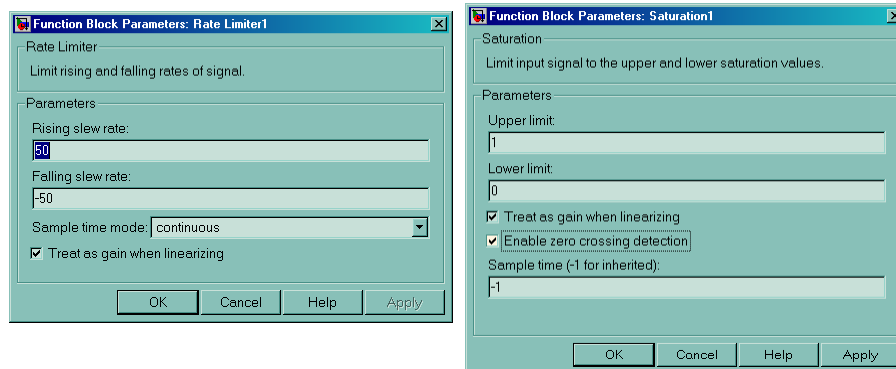
13

- The output of the error amplifier can be a large signal and either positive or negative.
- We have defined our throttle signal to be between 0 (no throttle) and 1 (full throttle).
- The saturation block is used to limit the throttle signal to appropriate values.
- The properties of the saturation and rate limiter block are shown on the next slide.



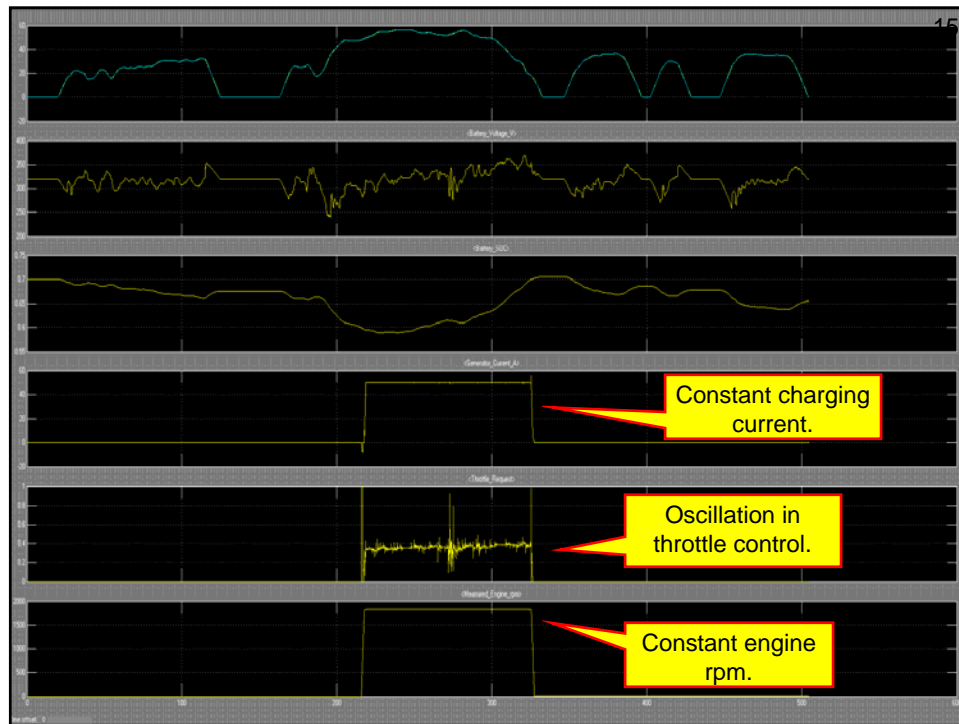
Component Settings

14



- Run the simulation and observe the throttle and generator current signals.





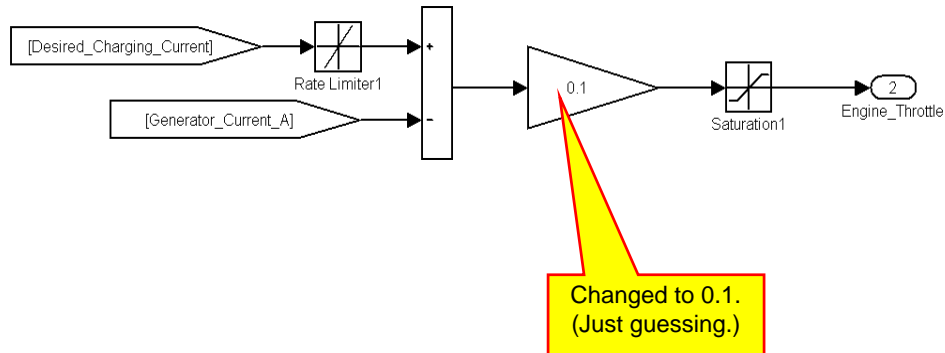
Current Control

16

- We see a lot of problems with the engine throttle signal.
- We will address them one at a time and see if we can fix the problems, or if a change is required in the design of the physical system.
- First, we will address problems in the throttle signal. The generator current is constant, but we notice noise and oscillations on the engine throttle signal.
- We will reduce the gain of the current control feedback loop.

Reduce the Gain

17

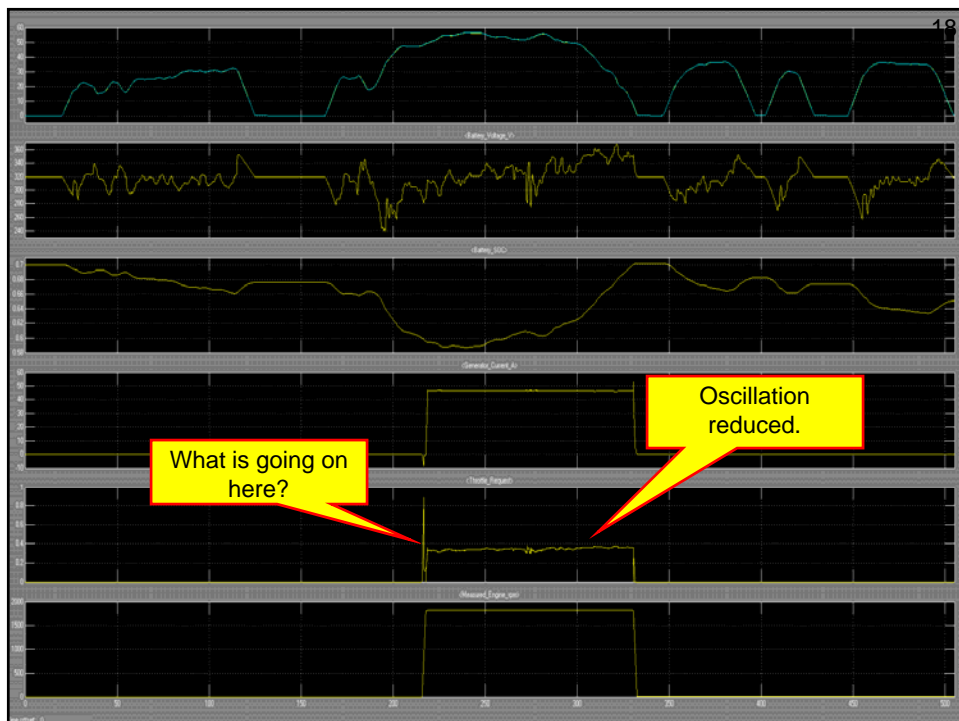


MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Current Control

19

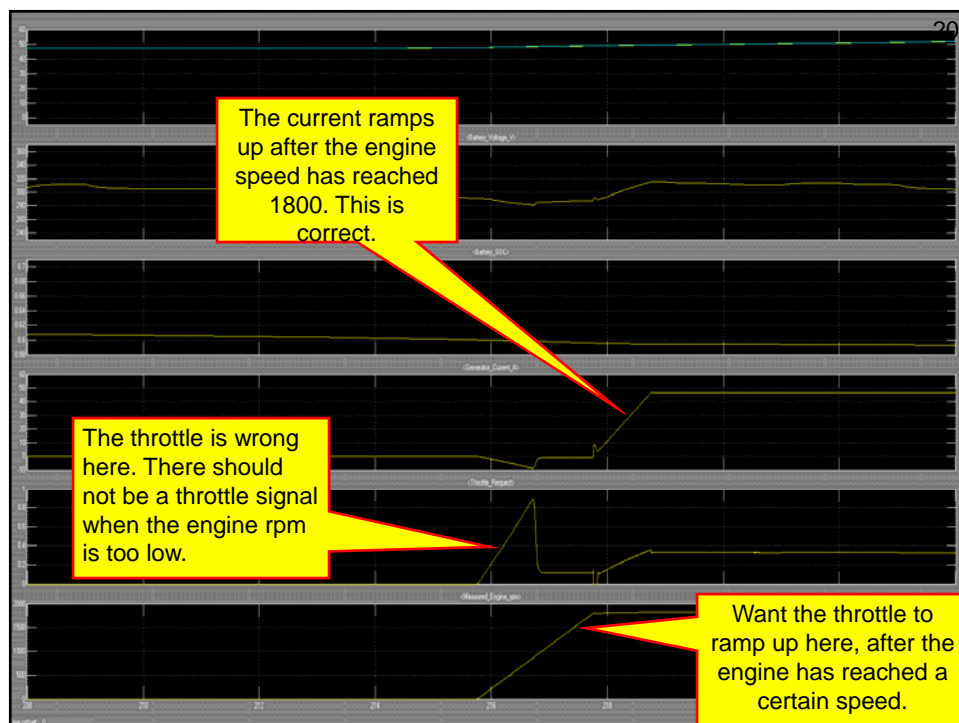
- We have reduced the oscillations in the throttle signal a little bit. We will call it good for now and fix a few other problems first.
- If the problem still persists, we will address it later.
- We will zoom in on the spike in the throttle signal that occurs when we first start the engine and commence charging.

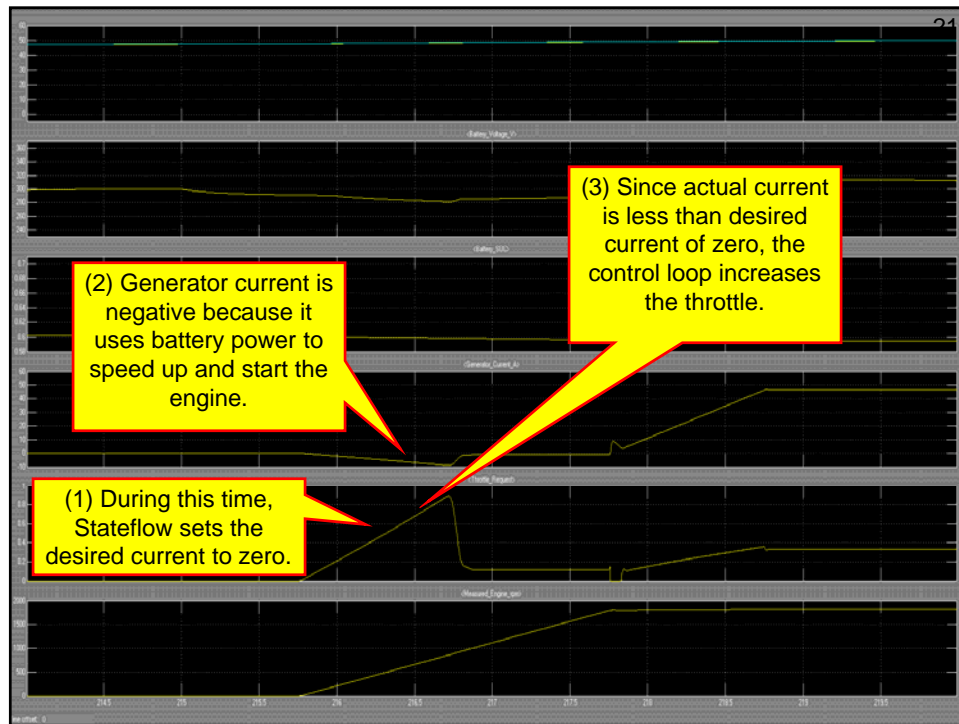
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



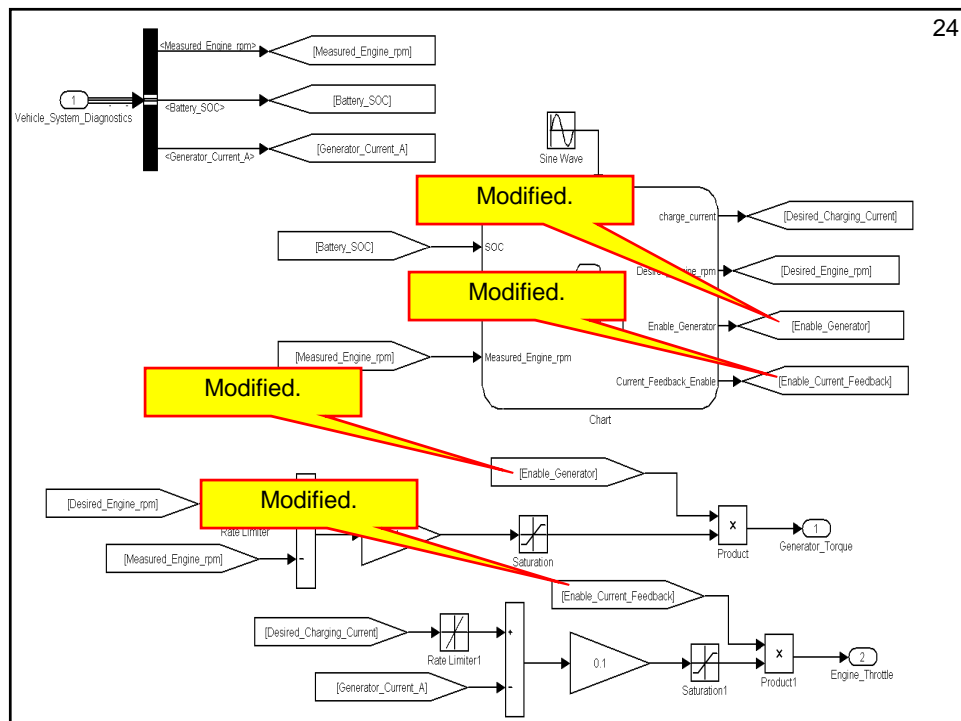
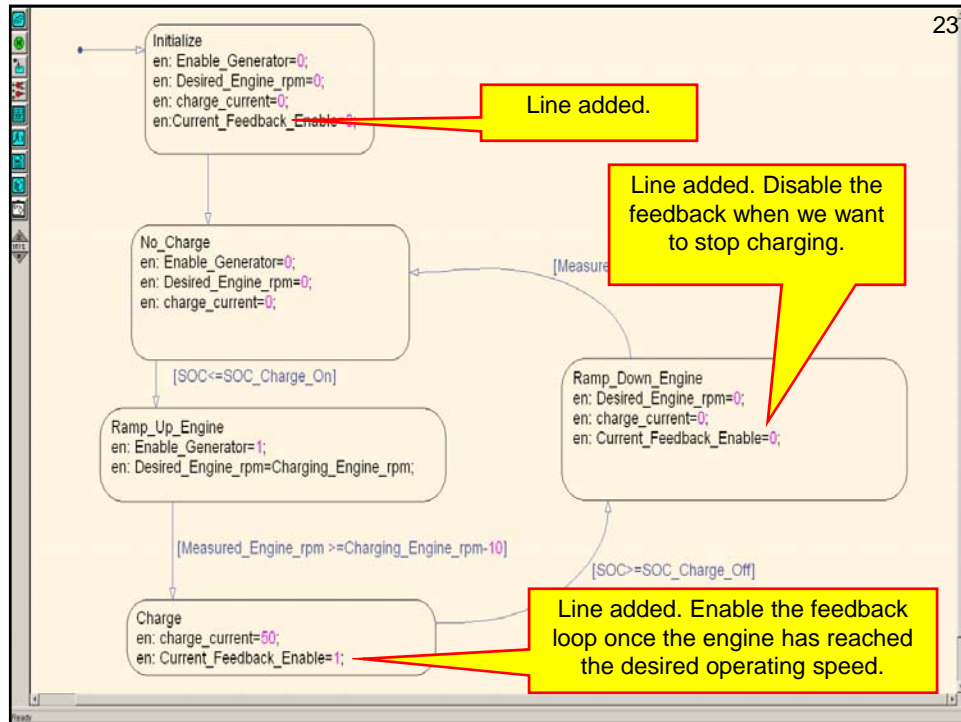


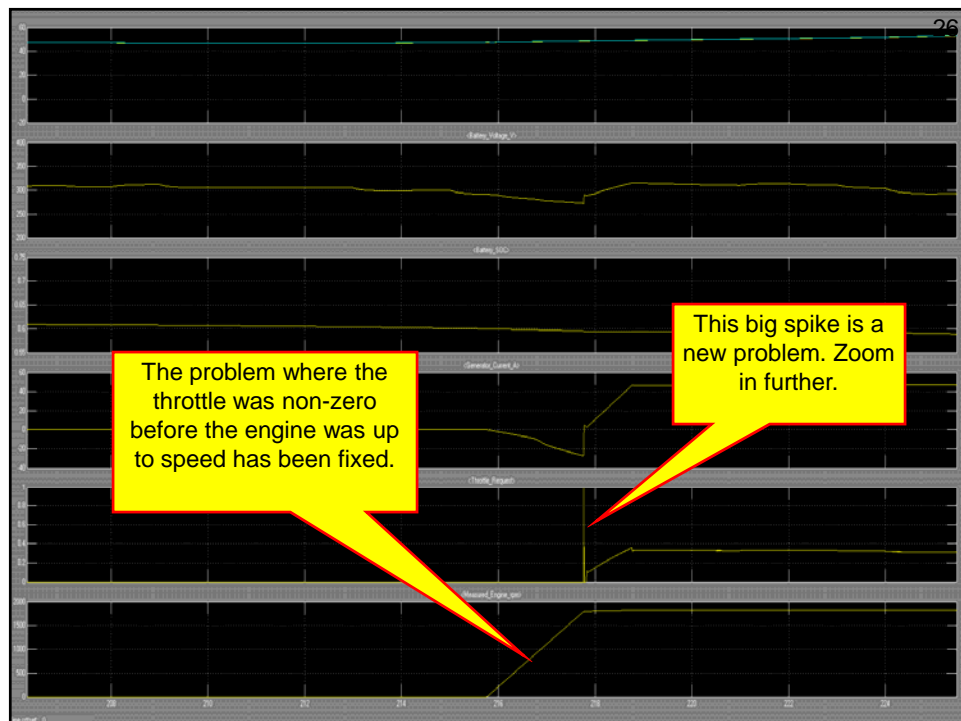
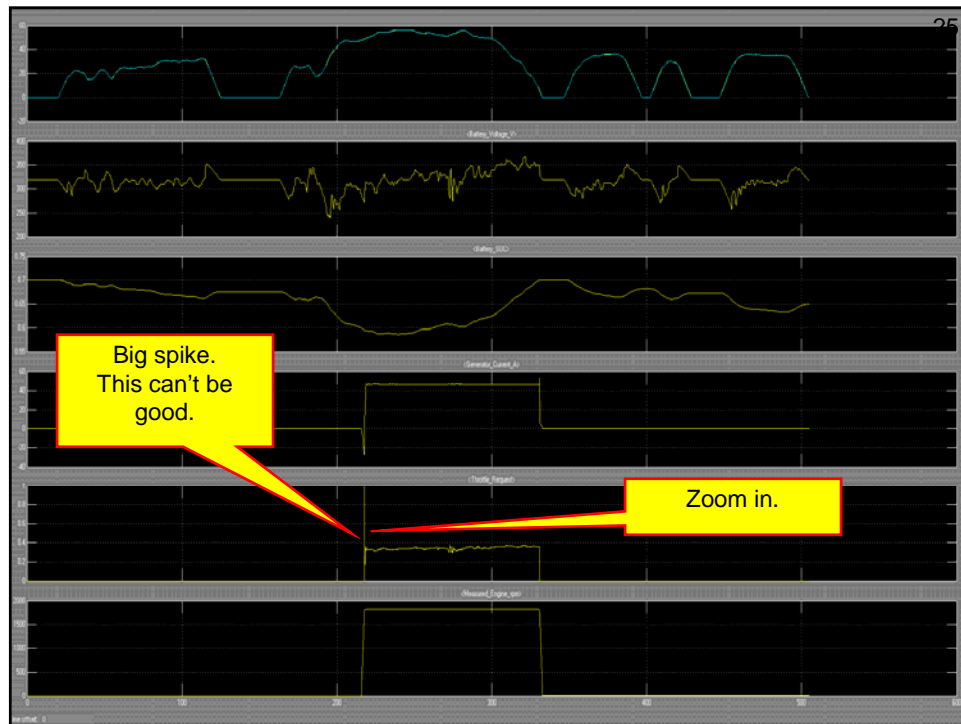
22

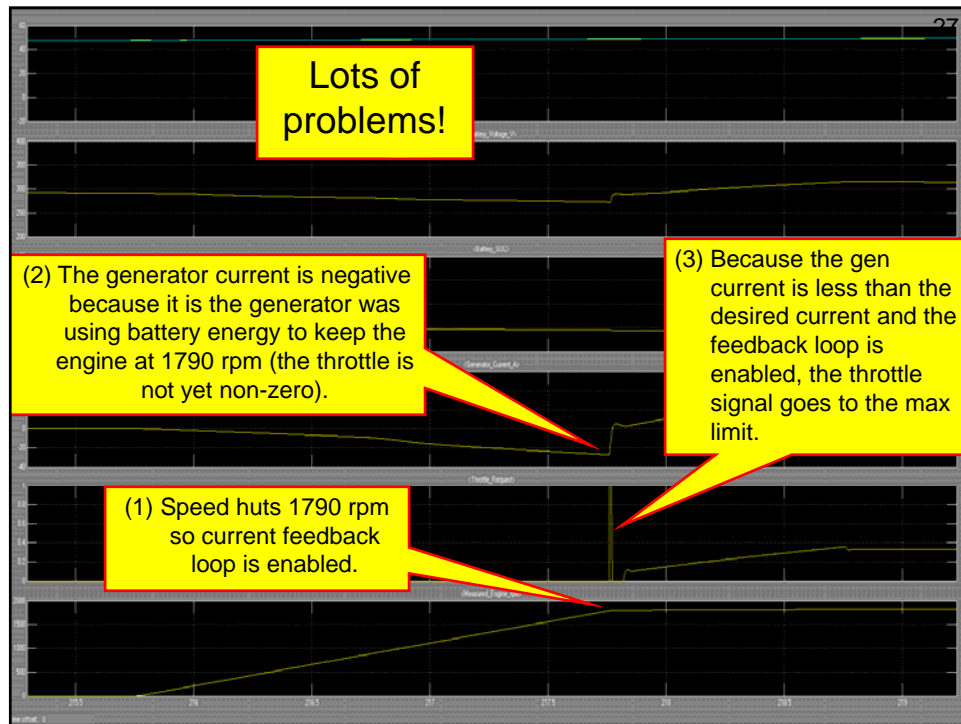
Charging Current Control

- We see that the desired generator current is zero but the actual generator current is negative.
- Because the actual current is less than desired, the current control feedback loop increases the throttle. This is incorrect because the engine is not yet on.
- We need to disable the throttle until the engine reaches the desired speed.
- Create another Stateflow output called Throttle_Enable.
- Modify the Stateflow diagram as shown next.
- Modify the Controller as shown next.









Current Control

28

- Since the generator uses power to speed up the engine, when we close the loop the throttle signal maxes out because it tries to change the generator current from a negative current to the desired current, which is zero or positive.
- We can fix this problem by using the throttle to assist the generator in speeding up the engine.
- We are not yet using the engine on signal.





Current Control

29

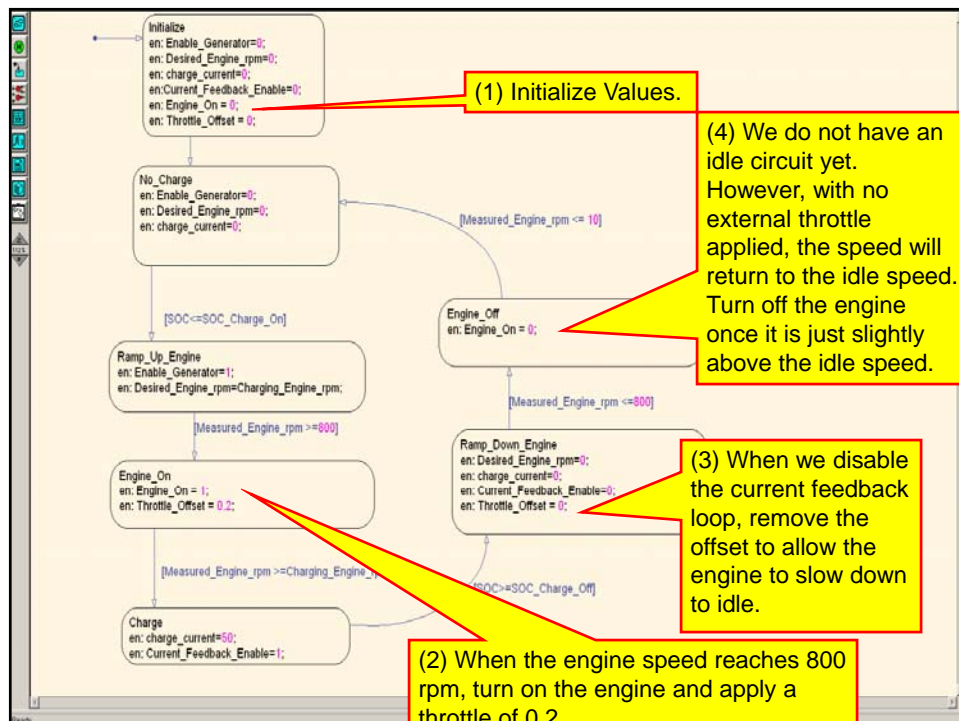
- We will modify the control to do the following.
- For engine speeds below 800 rpm, the engine is off.
- When the engine speed hits 800 rpm, turn on the engine.
- Apply a constant throttle to speed up the engine.
- We will determine this constant experimentally.
- We will need to add “Engine_On” and “Throttle_Offset” Signals to the Stateflow chart.
- Modify the Stateflow chart as shown:

MotoTron

The MathWorks

freescale
semiconductor

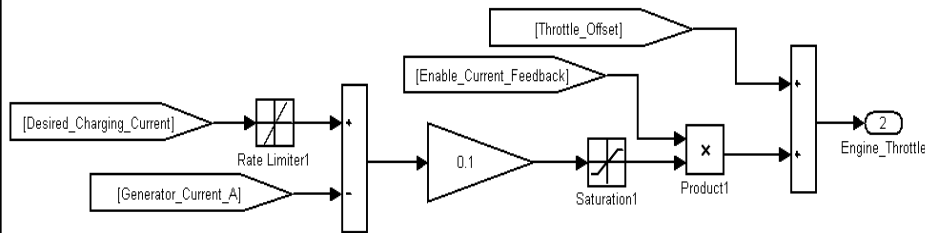
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Controller Modifications

31

- The current feedback loop has been modified as shown:



MotoTron

The MathWorks

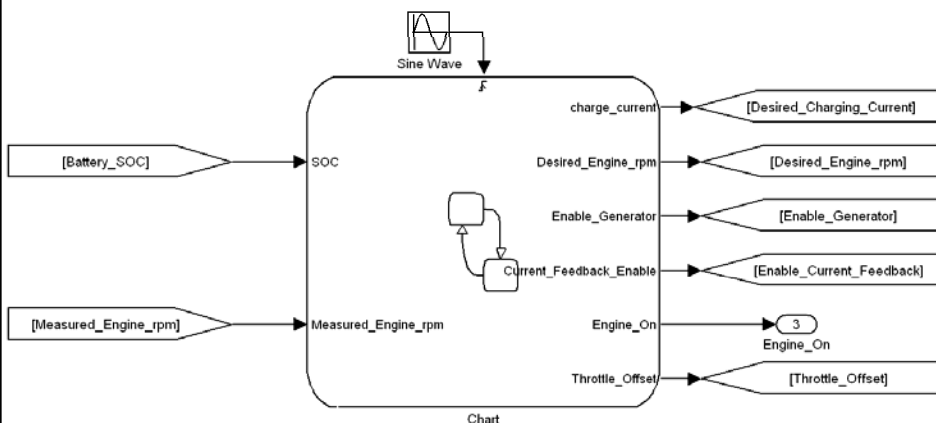
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller Modifications

32

- The Stateflow chart has been modified as shown:



MotoTron

The MathWorks

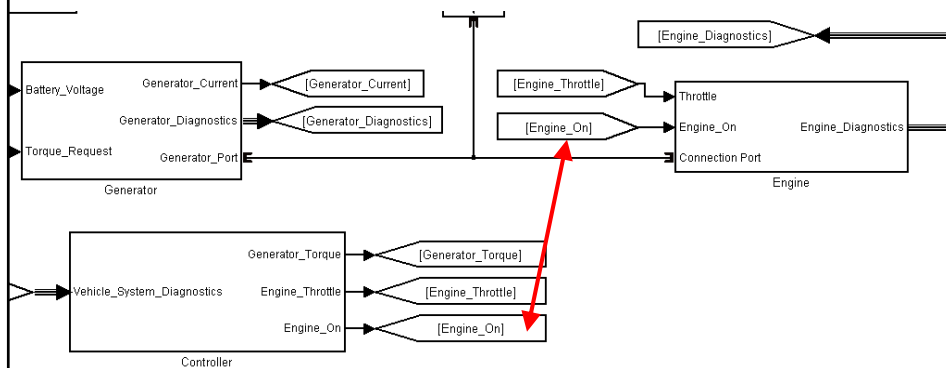
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Top-Level Model Changes

33

- The opt level model has been modified to add the connection for the Engine_On signal.

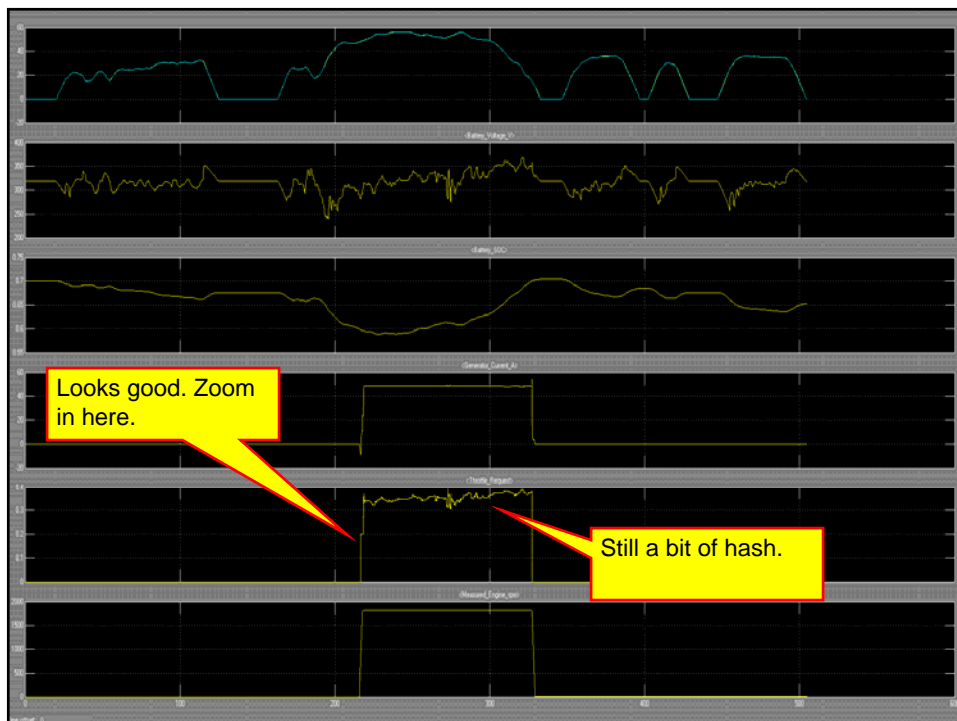


MotoTron

The MathWorks

freescale
semiconductor

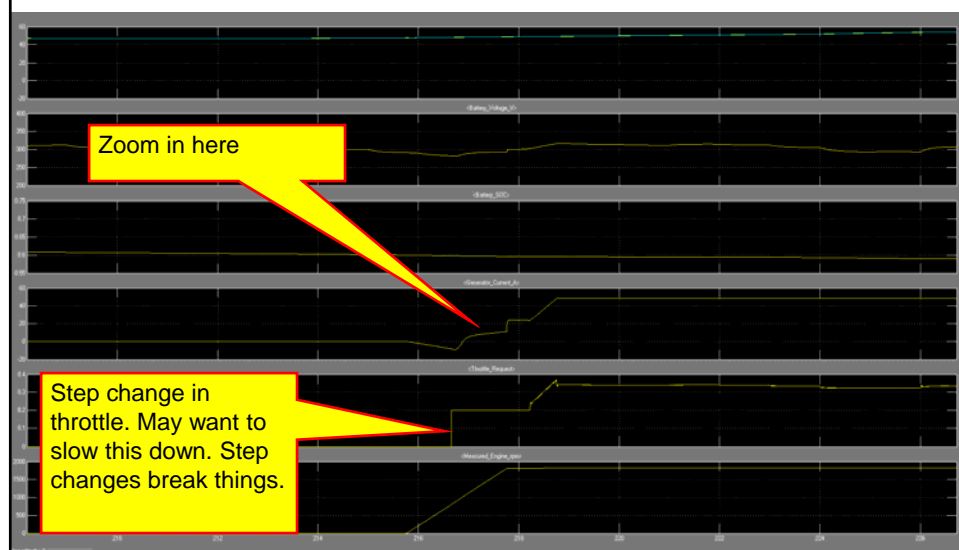
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

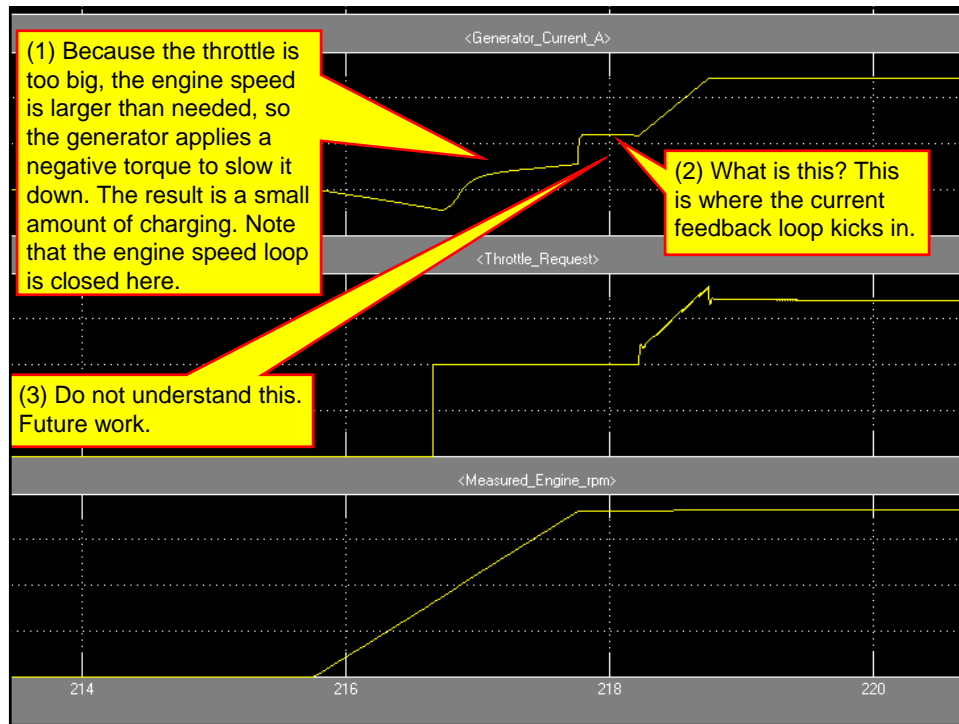




- This looks good and is acceptable. But there are a few interesting glitches that we wish to understand or examine.

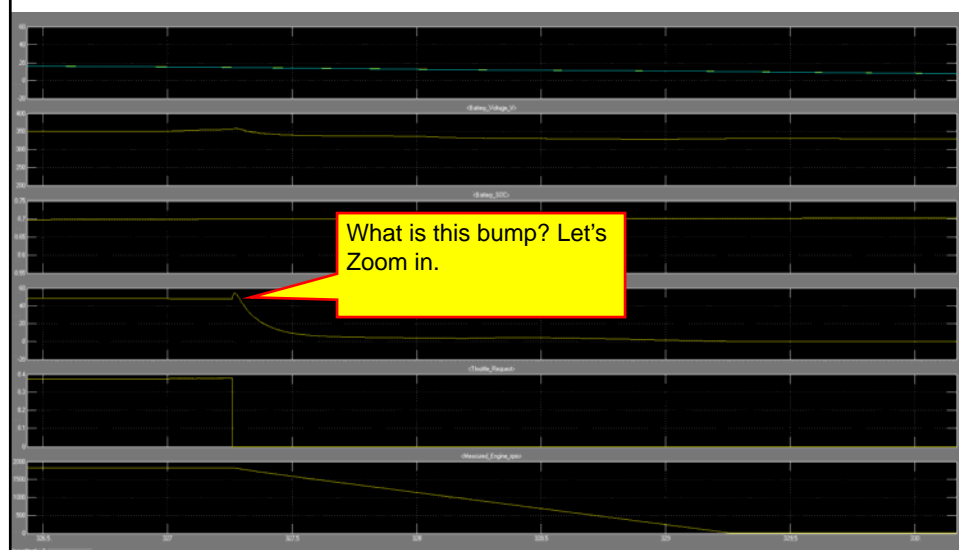
36

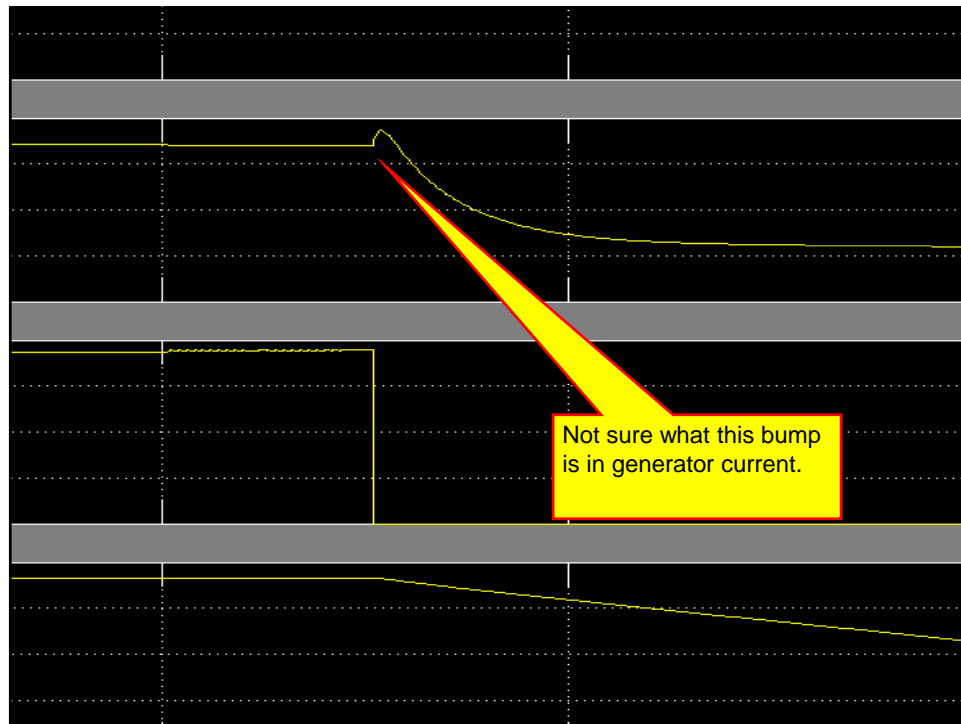




- The engine turnoff transient is shown below.
- There is a slight bump in charging current when we turn off the engine.

38





40



Controller

41

- Overall the current feedback and engine speed loops are working well.
- There are a few things that we do not understand, and need to understand.
- Even though this appears to be a fairly simple system, we see that it does require a bit of attention to control even a simple system.
- We will add more later in the course.



Controller

42

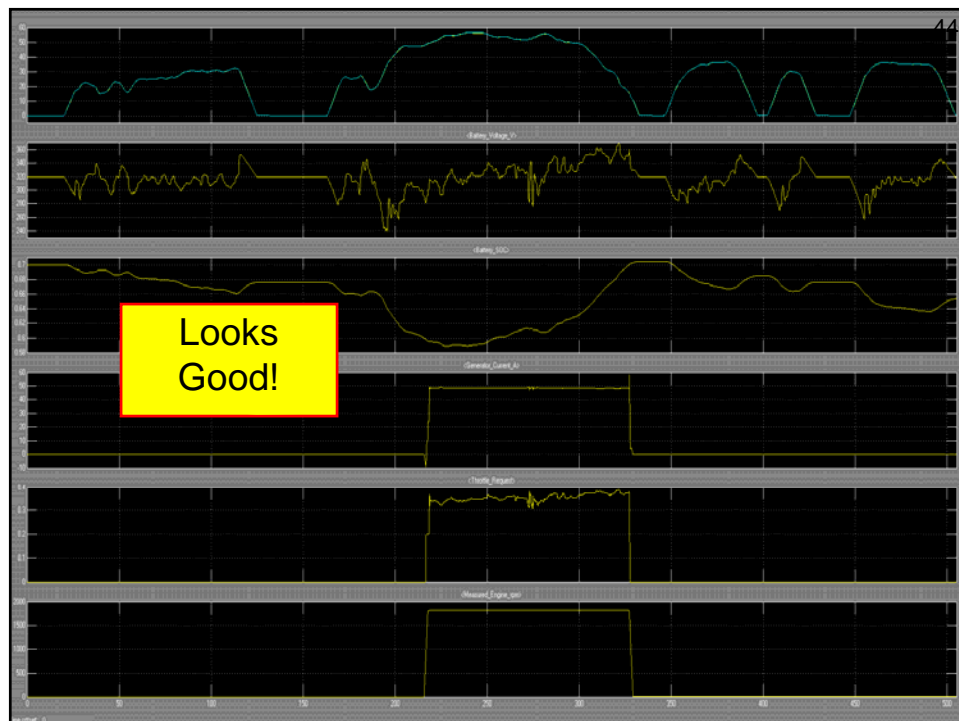
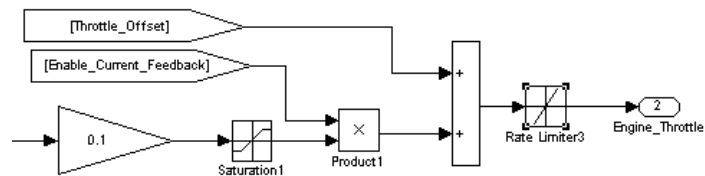
- The last thing we will do is add a rate limiter to the throttle signal.
- At the moment, this does not appear to cause a problem. However, step changes in any power or energy source can cause problems, in this case a torque spike.
- So, we will eliminate this step change with a rate limiter.



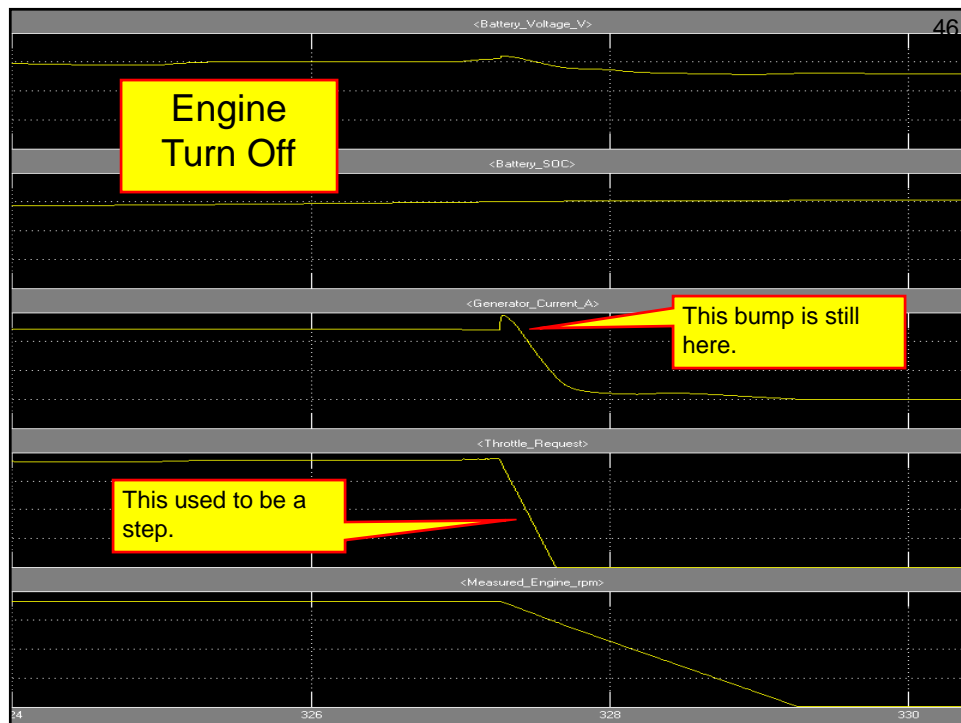
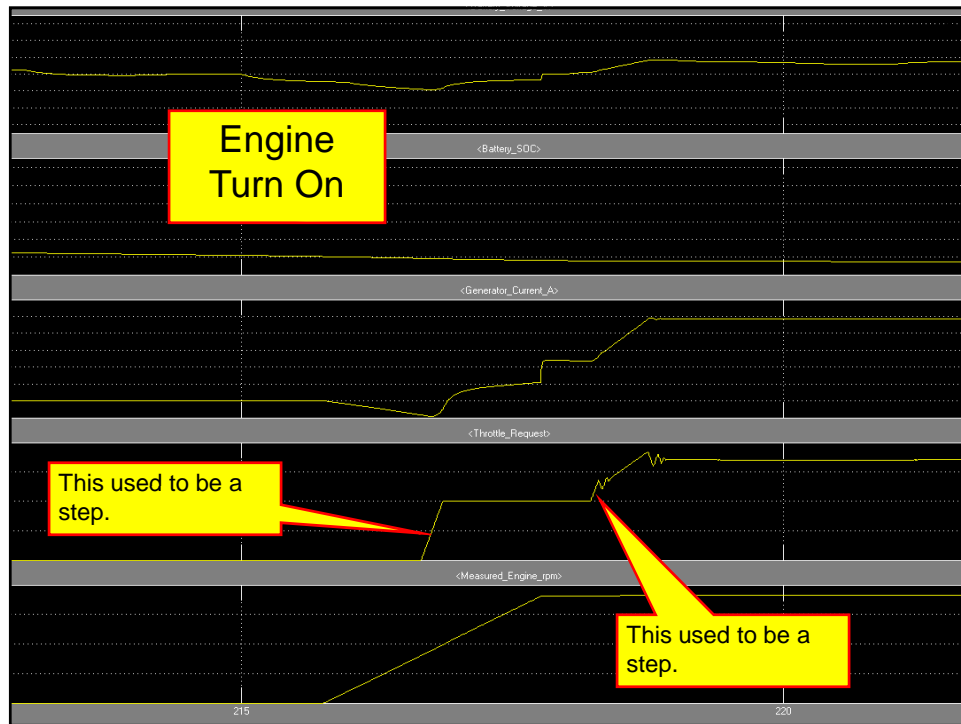
43

Current Control Problems

- Add a rate limiter to the throttle signal.
- This will add delay to our current control circuit that may cause an instability.
- Specify the rate as ± 1 per second. (Guess)
- Too fast of a slew rate will pass the spike.
- Too slow of a rate will cause instabilities.



44

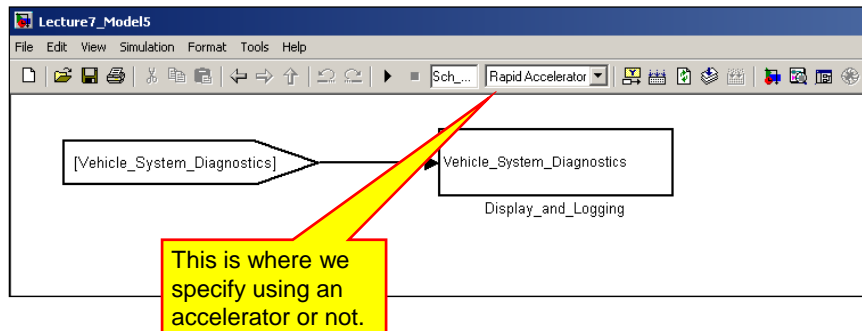




Run Simulations and Verify

47

- You might want to use the accelerator or rapid accelerator to speed up the longer runs.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Run Simulations and Verify

48

- For simulations with long runs and short sample times, the scope plots will display a large amount of data, and on occasion, MATLAB will run out of memory.
- To prevent this problem, we can add a sample time to the scopes.
- Note that this will not work for scopes set up with Signal & Scope Manager.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



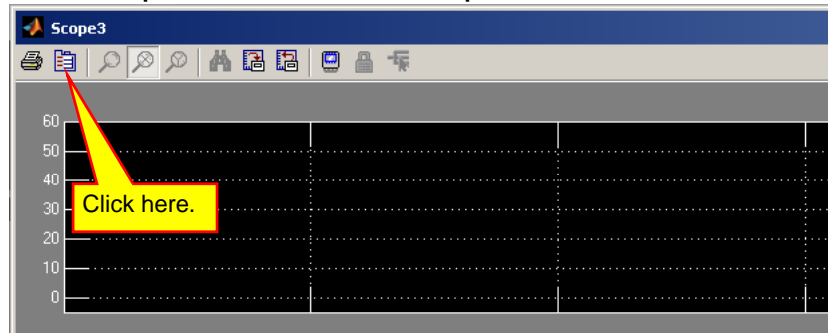
Sample Time

49

- In the Init File, define a variable

```
75  
76  
77 - Sample_Time = 0.1;  
78
```

- Open a scope and click on the parameters icon:



MotoTron

The MathWorks

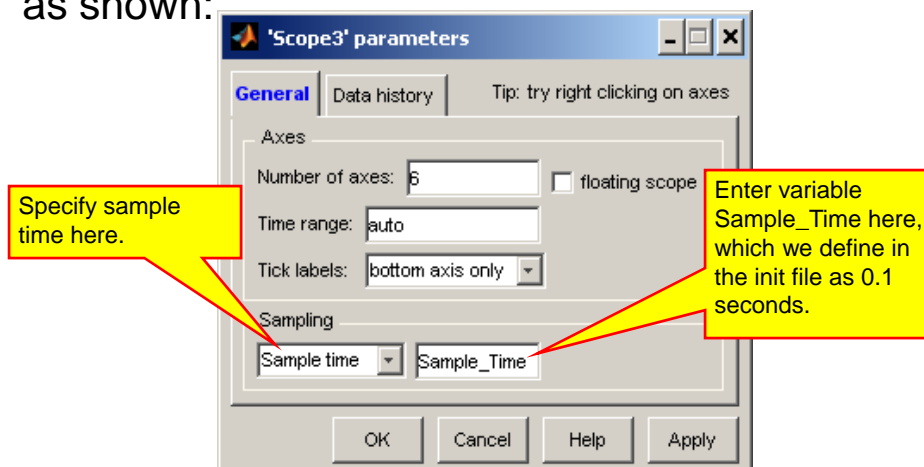
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Sample Time

50

- Set the Sample Time to variable Sample_Time as shown:



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Sample Time

51

- By setting the sample time to 0.1 seconds, we greatly limit the amount of data required to display a plot.
- Without this setting, the plot will use a huge amount of data, especially if step sizes become very small.
- Set the sample time for all scopes in your system.



Lecture 7 Demo 1

52

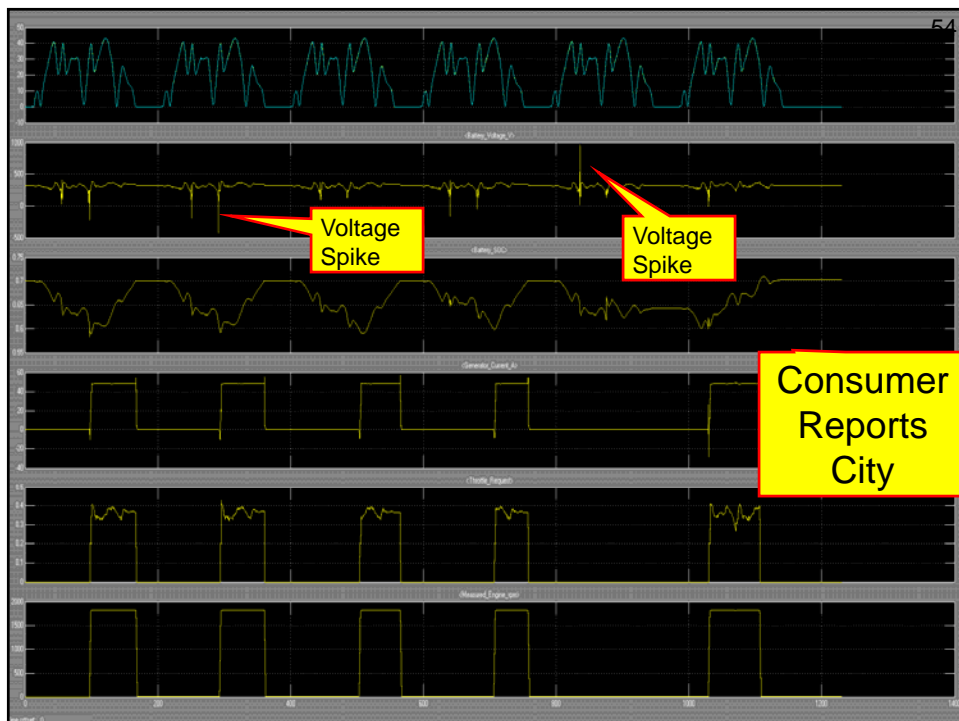
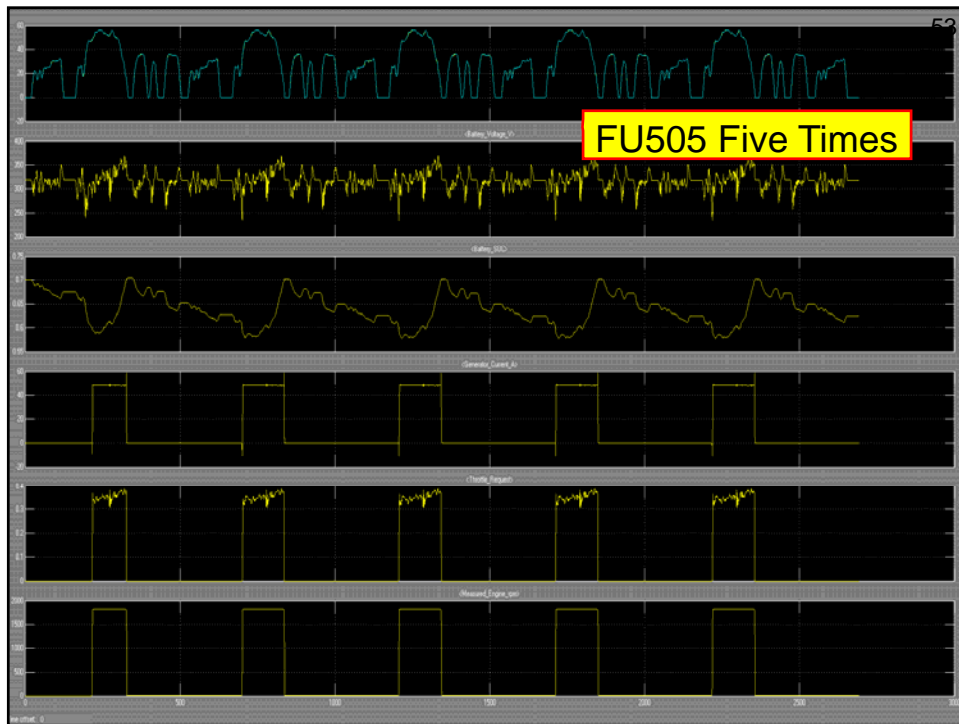
- Demo of System running the FU505 drive cycle five times
- Demo of model running the Consumer Reports drive cycle
- Demo of model running the Trip EPA Combined drive cycle.

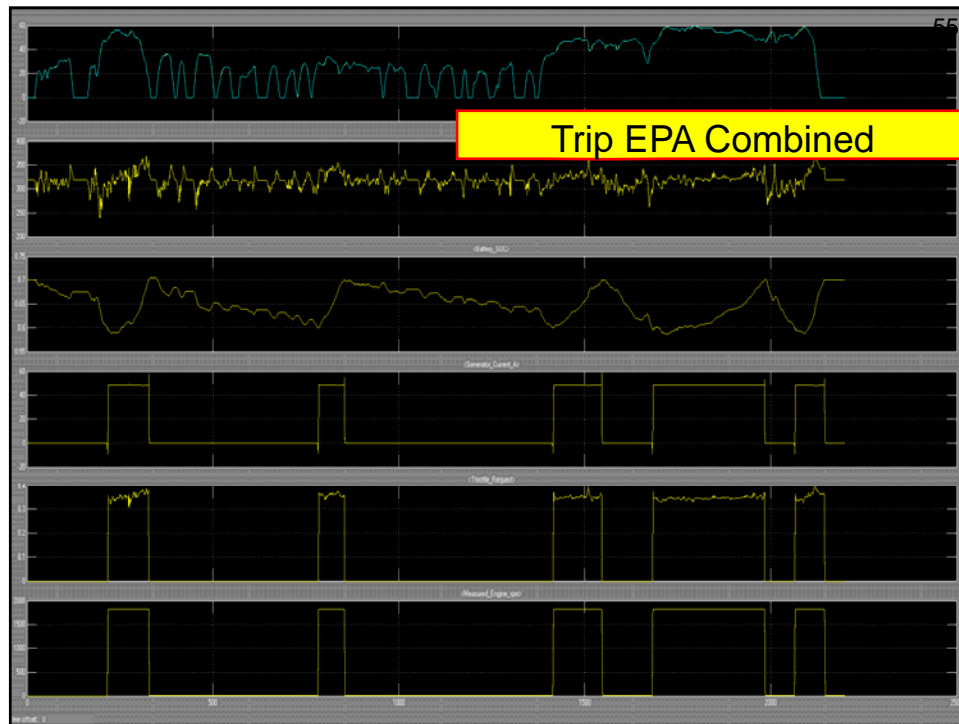
Demo_____

Demo_____

Demo_____



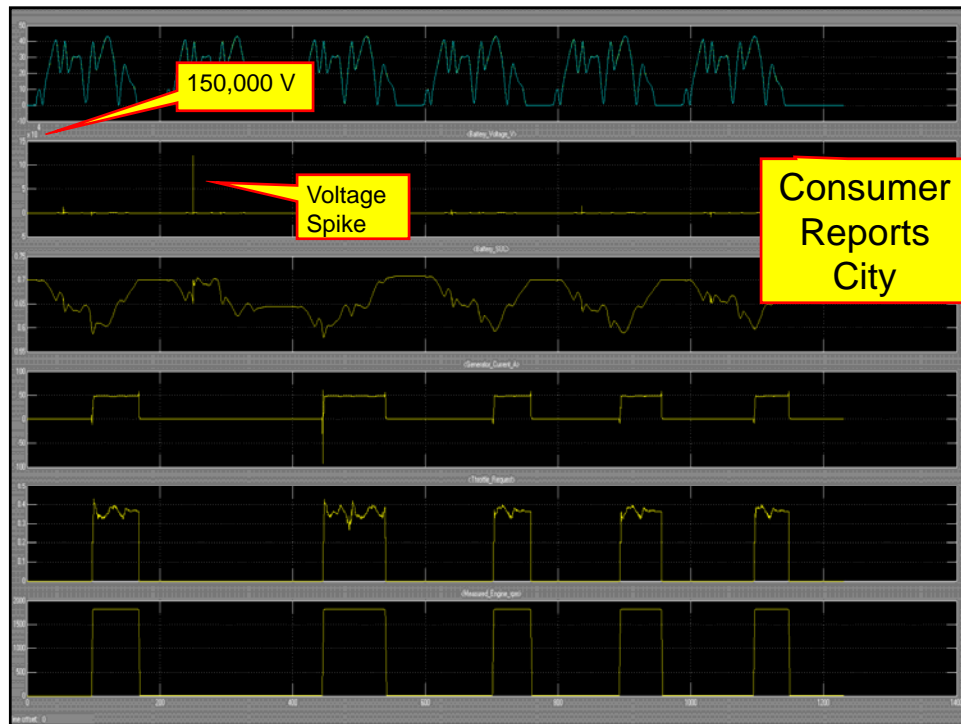




Lecture 7 Exercise 1

- In the consumer reports drive cycle, we noticed large voltage spikes. The spikes we observed in the previous Consumer Reports slide are reduced because we used a large sample time and since the spikes were so fast.
- The next slide sets the Sample_Time to 0.01, and we see huge voltage spikes.
- Figure out the reason for the battery voltage spikes and update the model to prevent the problem, and show the drive cycle with the problem eliminated.

Demo_____





Advanced Model-Based-System Design

Lecture 8: Post Processing



Post Processing

2

- Now that we have a model for the entire car, and a simple controller, we would like to make some efficiency calculations for the vehicle.
- These calculations will be done after the model runs.
- We will need to collect data from the simulation, and then use MATLAB to perform calculations on the data.





Post Processing

3

- We would like to calculate the fuel efficiency of our vehicle.
- If our engine used gas, this would be easy: efficiency (mile per gallon) is the distance travelled divided by the amount of fuel used in gallons.
- You might then ask, what about the electrical energy used to move the car? If we ran off the battery only, does the car get an infinite value of efficiency (mpg) since no gas is used?



Post Processing

4

- Furthermore, we will be using a fuel other than gasoline.
- In the future, vehicles will run off of a variety of different fuels, and we would like to be able to compare the efficiency of all of these vehicle in an apples-to-apples comparison.
- We will use a method called miles per gallon, gas equivalent (mpgge) where the mileage of our vehicle is converted to the equivalent miles per gallon had reformulated gasoline (RFG) been the fuel stock.
- We will also use state of charge correction to include the energy used from the battery.





Post Processing

5

- For our vehicle, we will assume that we are using 20% biodiesel fuel (B-20).
- This is 20 percent biodiesel, 80 percent petroleum diesel.
- We will first calculate conventional mpg for B-20.
- This is easy: $\text{mpg} = \text{distance travelled} / \text{the amount of B-20 fuel consumed (in gallons)}$
 - We already know the amount of fuel consumed in grams.
 - We do not know how far the vehicle has travelled in miles.



Post Processing

6

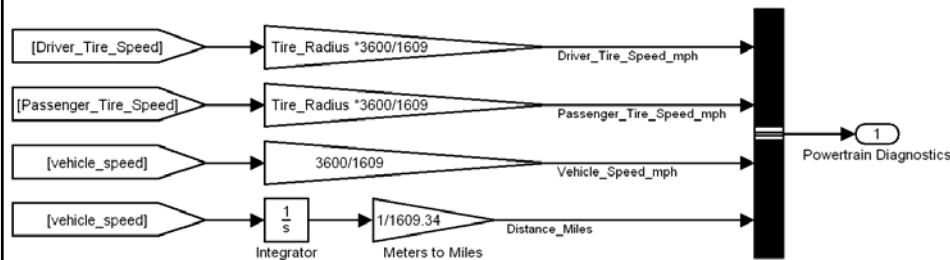
- We can have the model calculate the distance travelled by integrating the vehicle speed.
- We will convert the fuel consumed in grams to fuel consumed in gallons in the post processing file.
- Add an integrator and gain block to the Rear Diff and Body subsystem as shown.
- Add the distance signal to the diagnostics bus as shown.



Post Processing

7

- To convert from meters to miles per hour, divide meters by 1609.34.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Post Processing

8

- We are already calculating fuel consumed in the model, so we do not need to add a signal for that.
- In order to use the data calculated during the simulation in a MATLAB file, we need to save the calculated data in the MATLAB workspace.
- This is done with the MATLAB **To Workspace** part (**Simulink / Sinks library**).
- We will place the To Workspace parts in the Display_and_Logging subsystem:

MotoTron

The MathWorks

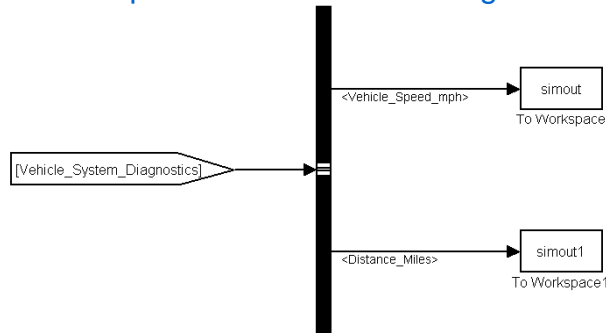
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Display_and_Logging Subsystem

9

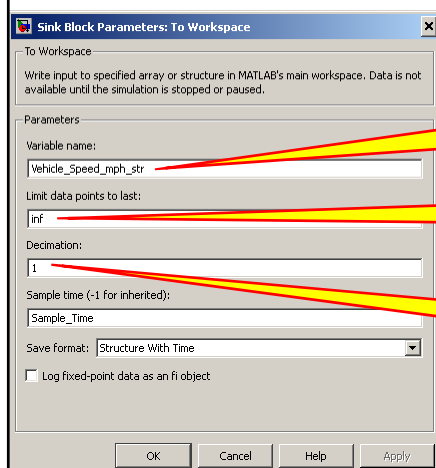
- For the moment, we will only place two To Workspace parts.
 - One will be used to log the vehicle speed, and simulation time.
 - The second will only be used to log vehicle distance.
 - The second one will be copied for all of the other signals we want to log.



Speed and Time

10

- Double-Click on the To Workspace block for the Vehicle_Speed_mph signal. Fill in the parameters as shown:



The data will be saved in the MATLAB workspace with the variable name Vehicle_Speed_mph_str.

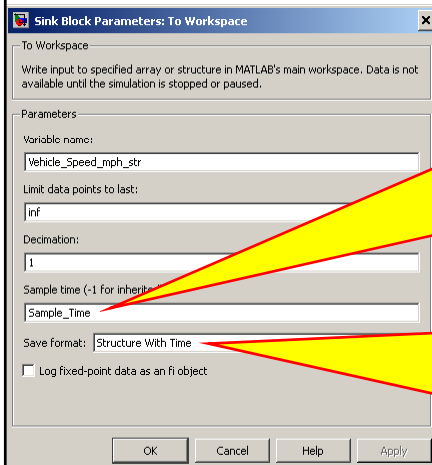
Date for the entire simulation will be saved in the structure.

No decimation. Every data point will be saved.



Speed and Time

11



To reduce the amount of data we save in the workspace, we will use the sample time to specify that we only save a data at a fixed and specified sample rate. Variable Sample_Time was specified previously in the init file as 0.1 seconds. Since all of our scopes and To Workspace blocks will use the Sample_Time variable, we can change the data collection sample rate for all of our blocks at the same time.

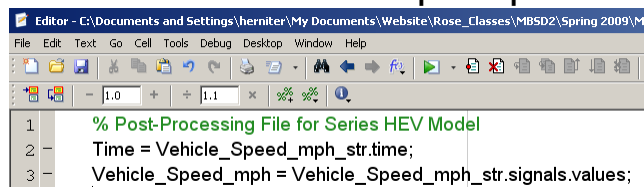
We will save this variable as a structure with time. This will save both the data for the signal and the time vector at which the data points were collected.

Although we will not be using the time vector in the mpgge calculation, we will use it later for more involved post processing.

Post Processing File

12

- We can extract the time and the vehicle speed data from the structure by using the lines below in the post processing file:



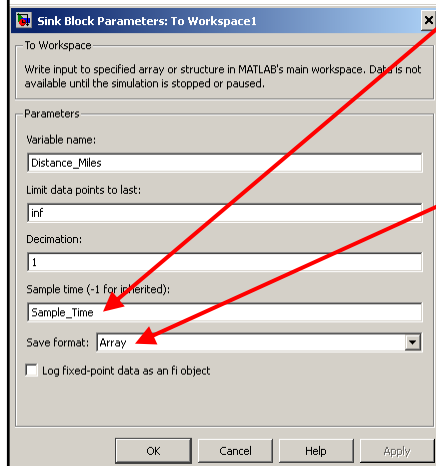
- Note that the post processing file is just an m-file that we run after the simulation has been completed.
- Save the file as Vehicle_Post_File.m



Distance_Miles

13

- The Distance_Miles signal will be saved as an array:

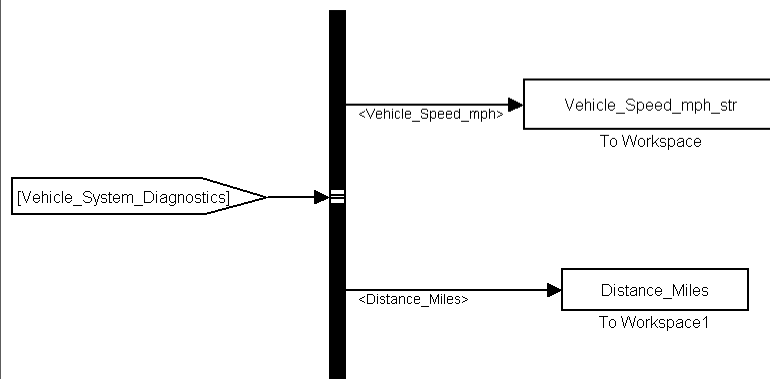


- Notice that we are using the Sample_Time variable to reduce the size of the array.
- The format is specified as an array. A 1-dimensional array will be created that contains only the data for the specified signal.

To Workspace

14

- When you close the dialog boxes, you will notice that the To Workspace blocks no display the variable name under which the data will be saved in the MATLAB workspace.





Post Processing

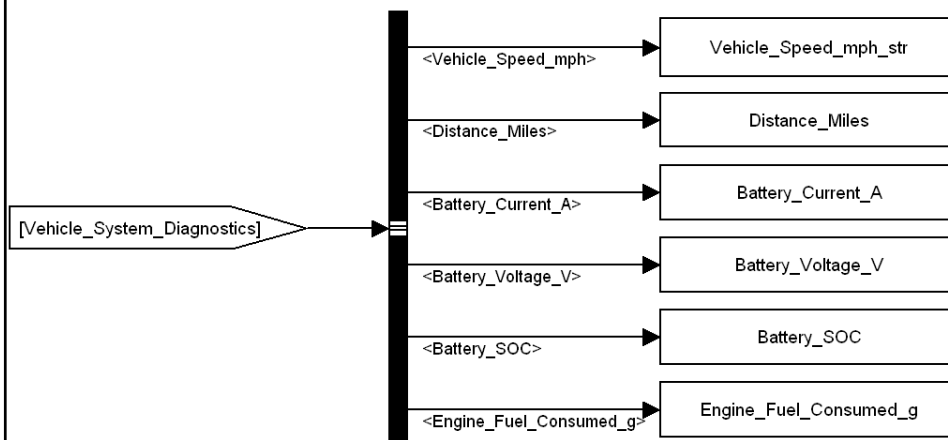
15

- Eventually we will add To Workspace blocks for all signals in our model.
- For the MPGGE calculation, we need the following signals:
 - Battery_SOC
 - Battery_Voltage_V
 - Battery_Current_A
 - Engine_Fuel_Consumed_g
- Copy the Distance_Miles To Workspace block for all of the other signals (they all will be saved as an array and use the Sample_Time).



Display and Logging Subsystem

16





Post Processing File

17

- We will now create the post processing file for our model.
- The first thing we will do is define a few constants for the conversion:

%MPGGE Calculations using Fuel heating values provided

% by Argonne National Lab (ANL).

Fuel_Heating_Value_B20 = 133393.1102; % BTU/gal (Provided by ANL)

Fuel_Heating_Value_RFG = 114871.7446; % BTU/gal (Provided by ANL)

Battery_OCV = 366; %Volts - Approximation for this model

Conversion_Efficiency = 0.25; % (Provided by ANL)



Constants

18

- Fuel heating values are the amount of energy contained in the fuel. Note that a gallon of diesel fuel contains more energy than a gallon of reformulated gasoline. This is one of the reasons why diesel costs more than gasoline.
- The heating values are in BTU/gal. (Sorry about that...)

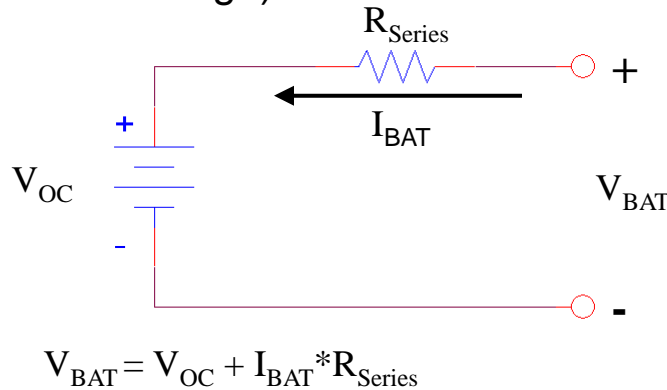




Constants

19

- Remember that our battery model was a voltage source (referred to as the open circuit voltage) in series with a resistor:



MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Constants

20

- To calculate the energy stored in the battery or the energy supplied by the battery, we need to know the battery open circuit voltage.
- In our model, the open circuit voltage is a function of the battery state of charge, which changes during the simulation.
- Later we will calculate the battery open circuit voltage from an V-I plot generated from the model.
- For now, we will assume that the open circuit voltage is constant equal to 366 V.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Constants

21

- We cannot just take energy from the battery and assume that it is free.
- Any energy take from the battery must be replaced or the battery will eventually become discharged.
- If we do use some of the energy from the battery to move the vehicle, we need to know the amount of fuel that it will take replace that energy.
- A conversion efficiency of fuel energy to electrical energy of 25% is assumed. (Later on, we may run simulations to verify this calculation).



Fuel and Distance

22

- The amount of B20 fuel consumed for the entire simulation is the last value saved in the Engine_Fuel_Consumed_g array.
- The fuel consumed is converted from grams to gallons by dividing by 3215. This conversion constant was provided by Argonne National Labs for a specific type of B-20 used in the Challenge X competitions.
- The total distance travelled in the simulation is the last value saved in the Distance_Miles array.

```

12
13 %Fuel Calculations using ANL values for fuel weight per gallon
14 Fuel_Consumed = Engine_Fuel_Consumed_g(end); % Grams
15 Fuel_Consumed_B20_Gallons = Fuel_Consumed/3215; % Based
16 % on 7.088591254 lbs per gallon provided by ANL.
17 Distance_Traveled = Distance_Miles(end); % Miles

```





Battery State of Charge (SOC)

23

- If we remove 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.



Battery State of Charge (SOC)

24

- The amount of charge added or removed from the battery is equal to the change in battery SOC times the battery amp-hour rating.
- We will assume that the ending battery SOC is less than the initial SOC, so that charge has been removed from the battery.
- Thus, we will call this amount,
Amp_Hours_Consumed:

%Calculate the electrical Energy

$\text{delta_SOC} = \text{Battery_SOC}(\text{end}) - \text{Battery_SOC_Init};$

$\text{Amp_Hours_Consumed} = \text{delta_SOC} * \text{Battery_AH_Rating};$





Battery State of Charge (SOC)

25

- The energy removed from the battery is equal to the Amp-hours consumed times the battery open circuit voltage. (Note that Amp-Hours times volts is equal to Watt-Hours.)
- Note that in our battery model, the open-circuit voltage source is the energy storage portion of the model. The series resistance accounts for battery losses.

Electrical_Energy = Amp_Hours_Consumed * Battery_OCV; %(Watt-Hours)



Battery State of Charge (SOC)

26

- We now know how much electrical energy was used in moving the vehicle. To calculate the amount of fuel the vehicle would use to replace that energy, we divide by the the Conversion_Efficiency.
- Finally, we will be using BTUs as the unit of energy for all post processing MPGGE calculations. To convert Watt-Hours to BTUs, multiply by 3.412.

Electrical_Energy_BTU = (Electrical_Energy/Conversion_Efficiency)*3.412; %W-Hr to BTU





Battery State of Charge (SOC)

27

- Note that if, at the end of a simulation, the battery final SOC is less than the battery initial SOC, electrical energy will have been removed from the battery.
- In our calculation, this will result in a negative value for the Electrical_Energy_BTU.
- Since this energy was removed from the battery and used to move the vehicle, we need to add this amount of energy to the actual amount of fuel consumed by the vehicle.
- Since energy removed from the battery is calculated as negative energy, we have to subtract this amount of energy from the fuel consumed energy so that the two actually add. (This is done later when we calculate the total energy consumed.)



Fuel Energy

28

- Next we calculate the amount of energy contained in the fuel that was consumed by the vehicle.
- We will assume B-20. If we are using a different value, we use the fuel heating value for that fuel.
- We know the amount of fuel consumed in gallons.
- The fuel heating value is the amount of energy in BTUs a fuel contains in BTU per gallon:

%Calculate the energy contained in the fuel consumed.

Fuel_Energy_BTU = Fuel_Consumed_B20_Gallons * Fuel_Heating_Value_B20;





Total Energy

29

- The total energy is the electrical energy consumed plus the fuel energy consumed.
- As mentioned earlier, if electrical energy is consumed, it is calculated as a negative quantity, so to add it to the total fuel consumed, we need to subtract:

% Calculate the total energy consumed in BTUs.

% Energy out of pack is negative, so we add it to the total fuel consumed.

Total_Energy_Consumed_BTU = Fuel_Energy_BTU - Electrical_Energy_BTU;



RFG Consumed

30

- We now know the total energy consumed by the vehicle for it to complete the drive cycle.
- Next, calculate the amount of reformulated gasoline (RFG) that would be required to supply the same amount of energy.
- Divide the total energy consumed (in BTUs) by the fuel heating value of RFG in BTU per gallon:

% Calculate the total energy consumed in BTUs.

% Energy out of pack is negative, so we add it to the total fuel consumed.

Total_Energy_Consumed_BTU = Fuel_Energy_BTU - Electrical_Energy_BTU;





MPGGE

31

- Finally, to calculate the efficiency, divide the distance travelled by the amount of RFG consumed:

%Calculate the efficiency, miles per gallon gas equivalent, with state of charge correction.

$MPGGE_RFG_wSOC = \text{Distance_Traveled} / \text{Fuel_Consumed_RFG_Gal};$

- Note that this is “gas equivalent” because we converted the energy into the amount of RFG required.
- The number is state of charge corrected because we included the amount of energy removed from the battery.



MPGGE

32

- 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.
- Awesome – We now know the total energy consumed by our vehicle over the drive cycle.





Vehicle_Post_File.m File

33

```

Editor - C:\Documents and Settings\herniter\My Documents\Website\Rose_Classes\MBSD2\Spring 2009\Matlab Files\Vehicle_Post
File Edit Text Go Cell Tools Debug Desktop Window Help
1 % Post-Processing File for Series HEV Model
2 Time = Vehicle_Speed_mph_str.time;
3 Vehicle_Speed_mph = Vehicle_Speed_mph_str.signals.values;
4
5
6 %MPGGE Calculations using Fuel heating values provided
7 % by Argonne National Lab (ANL).
8 Fuel_Heating_Value_B20 = 133393.1102; % BTU/gal (Provided by ANL)
9 Fuel_Heating_Value_RFG = 114871.7446; % BTU/gal (Provided by ANL)
10 Battery_OCV = 366; %Volts - Approximation for this model
11 Conversion_Efficiency = 0.25; % (Provided by ANL)
12
13 %Fuel Calculations using ANL values for fuel weight per gallon
14 Fuel_Consumed = Engine_Fuel_Consumed_g(end); % Grams
15 Fuel_Consumed_B20_Gallons = Fuel_Consumed/3215; % Based
16 % on 7.088591254 lbs per gallon provided by ANL.
17 Distance_Traveled = Distance_Miles(end);% Miles
18

```

Vehicle_Post_File.m File

34

```

%Calculate the electrical Energy
delta_SOC = Battery_SOC(end)-Battery_SOC_Init;
Amp_Hours_Consumed = delta_SOC*Battery_AH_Rating;
Electrical_Energy = Amp_Hours_Consumed*Battery_OCV; %(Watt-Hours)
Electrical_Energy_BTU = (Electrical_Energy/Conversion_Efficiency)*3.412; %W-Hr to BTU

%Calcualte the energy contained in the fuel consumed.
Fuel_Energy_BTU = Fuel_Consumed_B20_Gallons*Fuel_Heating_Value_B20;

% Calcualte the total energy consumed in BTUs.
%Energy out of pack is negative, so we add it to the total fuel consumed.
Total_Energy_Consumed_BTU = Fuel_Energy_BTU-Electrical_Energy_BTU;

% Calculatethe amount of RFG that would be necessary to supply
% the same amount of energy consumed by the vehicle.
Fuel_Consumed_RFG_Gal = Total_Energy_Consumed_BTU/Fuel_Heating_Value_RFG;

%Calculate the feeiciency, miles per gallon gas equivalent, with state of
%charge correction.
MPGGE_RFG_wSOC = Distance_Traveled/Fuel_Consumed_RFG_Gal;

```



Msgbox

35

- The last thing we need to do is create a spiffy display for our calculations:

```
msg1=sprintf('          Results of Vehicle Simulation\n\n');
msg2=sprintf('-----\n');
msg3=sprintf('Cycle name: %s\n-----\n\n',fn);

msg4=sprintf('Total distance traveled = %g miles.\n', Distance_Traveled);
msg5=sprintf('Amount of fuel consumed = %g grams of B20.\n', Fuel_Consumed);
msg6=sprintf('\nMPG Calculations based on ANL Calculations\n-----\n');
msg7=sprintf('MPGGE with SOC Correction = %g.\n', MPGGE_RFG_wSOC);

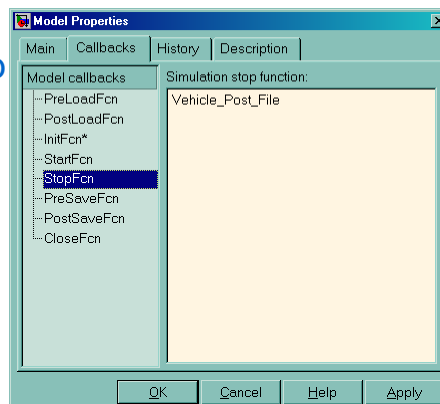
msg=[msg1,msg2, msg3, msg4,msg5, msg6, msg7];
Title=['Schedule File: ', fn];
h=msgbox(msg, Title);
```



Post Processing File

36

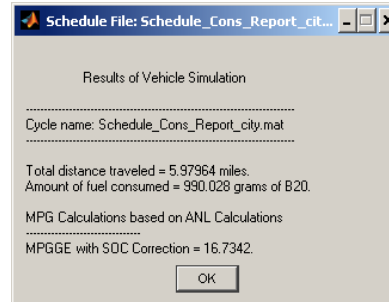
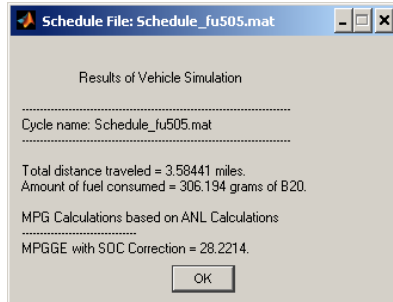
- If you would like this file to run after every simulation, we can specify it as a callback that runs when the model stops..
- Right click on the model and
 - Select **Model Properties**
 - Select the **Callbacks** Tab
 - Select **StopFcn**
 - Enter the name of the post processing file without the .m.





Simulation Results

37



Demo_____

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Open Circuit Voltage

38

- In our previous calculation, we assumed a value for the open circuit voltage.
- This value can be calculated from the battery voltage and current data collected during the simulation.
- If you remember, our model for the battery, the terminal voltage is equal to the open circuit voltage plus the battery current times the series resistance:

MotoTron

 **The MathWorks**

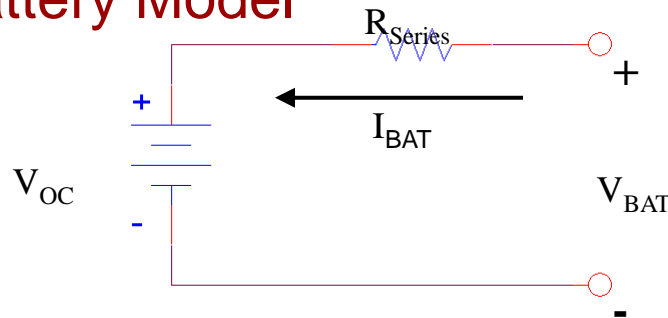
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Battery Model

39



$$V_{BAT} = V_{OC} + I_{BAT} * R_{Series}$$

- The battery terminal voltage is a linear function of the battery current.
- The equation is a straight line where the y-intercept is the battery open-circuit voltage.

MotoTron

 **The MathWorks**

 **freescaler**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Battery Open-Circuit Voltage

40

- We can determine the battery open-circuit voltage by plotting the battery voltage versus battery current (a V-I plot).
- We can fit a first order polynomial to the measured data.
- The y-intercept of the fitted curve is the battery open-circuit voltage.

MotoTron

 **The MathWorks**

 **freescaler**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Battery Open-Circuit Voltage

41

- The lines below plot the battery V-I curve:

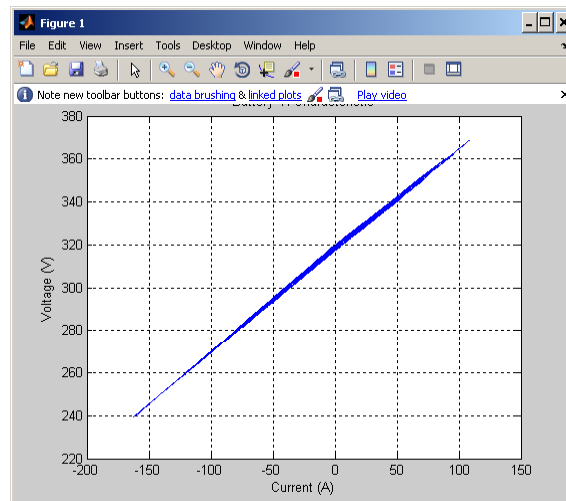
```
%Plot the measured battery V-I curve  
plot( Battery_Current_A, Battery_Voltage_V);  
xlabel('Current (A)');  
title('Battery VI Characteristic');  
ylabel('Voltage (V)');  
grid on;
```



Battery Open-Circuit Voltage

42

- The measured V-I curve for the FU505 drive cycle is:





Battery Open-Circuit Voltage

43

- We could eye-ball the curve and estimate that the open-circuit voltage is around 320 V.
- Instead, we will use a first-order polynomial curve fit to the measured data.
- We will use the MATLAB polyfit command:

```
%Calculate the battery open circuit voltage from the battery IV plot.  
poly=polyfit(Battery_Current_A, Battery_Voltage_V,1);  
Battery_OCV = poly(2); %in Volts
```



Polyfit

44

- The polyfit function calculates a polynomial fit to the measured data.
- The value is returned in a polynomial.
- Since we asked for a first order fit, the polynomial is in the form $\text{poly} = [a1 \ a2]$.
- The 1st order polynomial equation would be in the form $y = a1 \cdot x + a2$.
- Thus, the second coefficient of the returned polynomial is the y-intercept, and in this example, the battery open circuit voltage.
- For the FU505 drive cycle, the open circuit voltage is calculated as 318.441. A bit off from our estimate of 366 V.

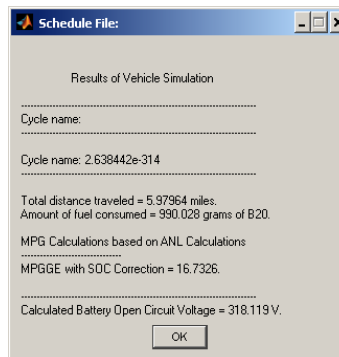
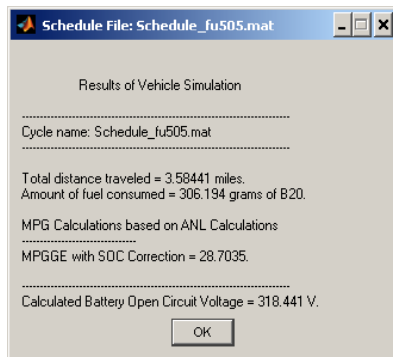




Updated MPGGE Results

45

- With the updated Battery SOC, the MPGGE calculations are as shown:



Demo_____

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Advanced Model-Based-System Design

Lecture 9: Improving Simulation Speed Brake Models



Simulation Speed

2

- By now you should have noticed that our model has the annoying problem that at zero speed, the simulation becomes exceedingly slow.
- This is bothersome because:
 - It takes so long to run.
 - The simulation appears to run slowly when the desired vehicle speed is zero. (For some reason, this seems wrong. We expect the simulation to require more computation when the vehicle is performing maneuvers rather than sitting still and not moving.





Simulation Speed

3

- The problem becomes obvious when we run the Consumer Reports City drive cycle which has high acceleration and deceleration portions, and several portions at zero speed.
- The simulation runs fast during the acceleration and deceleration portions of the drive cycle, but slows to a crawl when the vehicle is at zero speed.



Simulation Speed

4

- We have been dealing with this problem with several different vehicle models for a long time.
- The method presented here to discover the problem may not present a method of how the solution to the problem was discovered.
- The solution was discovered as a result of the model used in this course, however, the realization of the solution took several models and several years (yes, we are slow).
- We suggest a method to discover the reason for the problem, but the method presented to discover the solution may not really describe the troubleshooting method that uncovered the solution.





Simulation speed

5

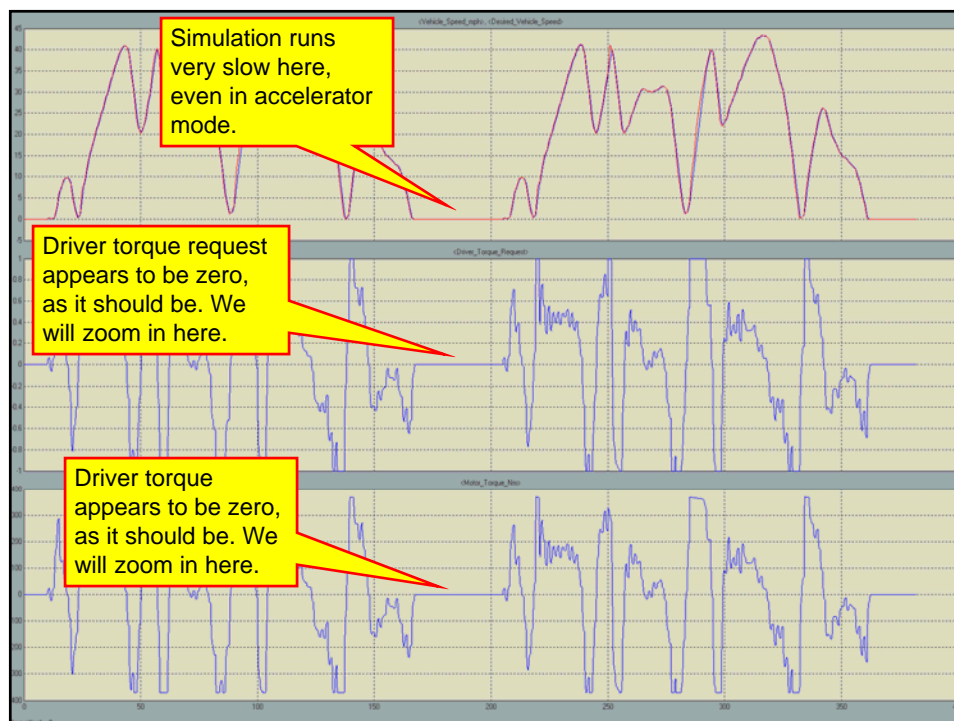
- If we run the simulation with the Consumer Reports City Cycle (not the whole thing, it takes too long) and plot the
 - Desired and actual vehicle speed
 - Driver torque request
 - Motor torque
- We might get some insight into the problem (or maybe not).

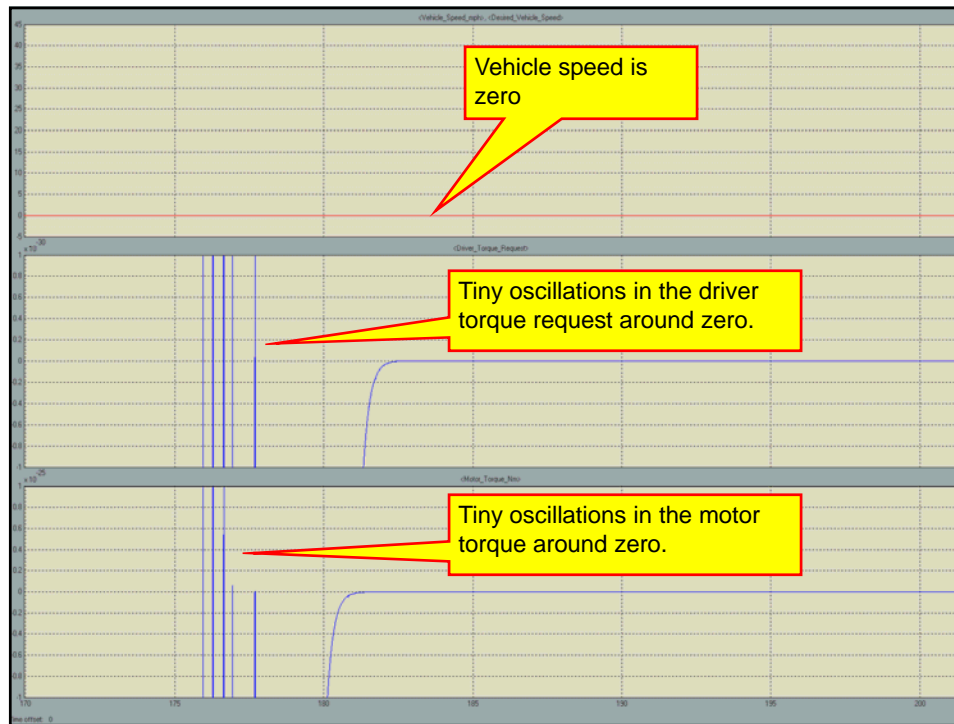
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Simulation Speed

8

- We see that there are small oscillations (10^{-30} size oscillations) around zero.
- We conclude that in order to keep the vehicle speed at zero, the proportional feedback system (the driver block feedback loop) gives the vehicle a small negative torque bump to slow the vehicle down.
- This bump, however causes the vehicle speed to become slightly negative. The feedback controller detects this and gives the motor a small positive torque bump to correct this.



Simulation Speed

9

- The vehicle speed has tiny oscillations around zero speed, and the system and the system feedback system and motor use tiny positive and negative torque pulses to push the vehicle speed slightly positive or slightly negative.
- Because there is no damping, the vehicle never settles down to zero.
- Tiny time steps are required because the MATLAB solver tries to get close to zero within a specified tolerance.
- These tiny time steps make the simulation run slowly.



Simulation Speed

10

- You might ask, if this is a proportional feedback system, why do we not see the same problem when the vehicle attempts to maintain a constant speed at say 30 mph?
- A possible explanation is that the vehicle model has aerodynamic drag built into the vehicle solver.
- Drag increases as the square of the velocity. At 30 mph the amount of drag is significant.
- To increase speed, a torque bump from the motor is necessary. If the vehicle speed is too high, aero drag will slow it down.





Simulation Speed

11

- Thus, to maintain a constant speed only positive motor torque requests are needed. Drag ends up decreasing vehicle speed.
- Thus, the torque request signal will not oscillate around some bias.



Solution 1

12

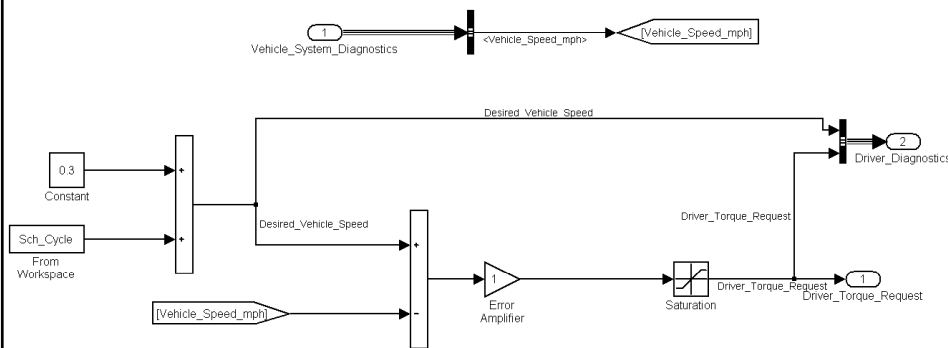
- So, we think we know the reason for the simulation to slow down when the vehicle speed is zero.
- A solution to this problem is to never allow the vehicle to reach zero.
- We will add a small constant offset to the desired speed in the driver block.



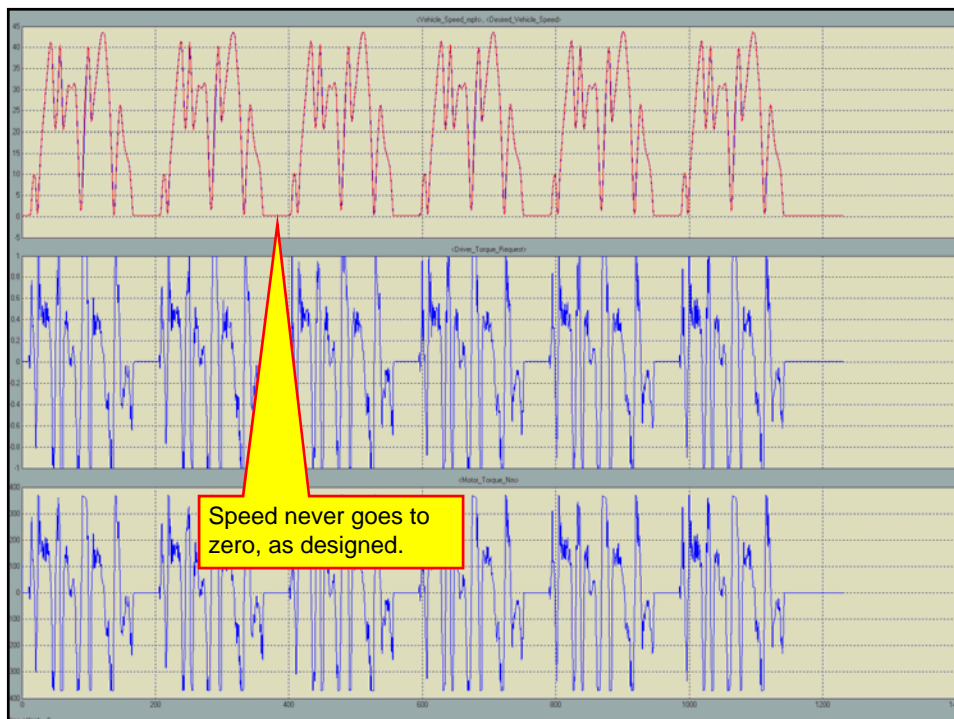
Solution 1

13

- We see that we have added a 0.3 mph constant offset to the desired speed signal.



- With this modification, the simulation now runs very fast:





Solution 1 – Speed Offset

15

- The simulation now takes about a minute to run, where it used to take about 9 minutes.
- We do notice the offset as we see that the vehicle speed never reaches 0.
- The solution is acceptable, but could cause problems when we add a shifter to our system and have the vehicle go forward and reverse.
- The solution does support our hypothesis that the driver feedback loop may be the cause of the problem.



Solution 2

16

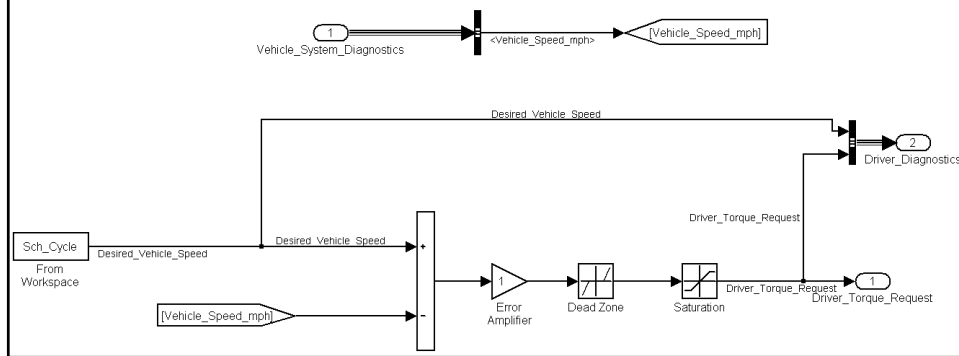
- Since we are hypothesizing that the driver block is emitting small positive and negative torque requests to keep the vehicle at zero speed, we come up with a new idea:
 - Since tiny torque requests will not significantly move the vehicle, why not just prevent torque requests below a certain threshold from being emitted by the controller.
 - We can do this with a dead zone block.



Solution 2: Dead Zone Clipper

17

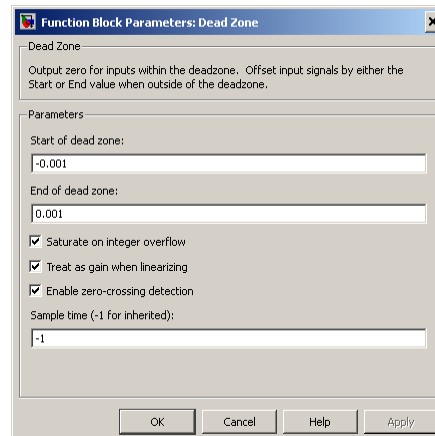
- Remove the offset we added previously.
- Add a **Dead Zone** block (**Simulink / Discontinuities**) to the controller as shown.



Solution 2: Dead Zone Clipper

18

- The properties of the Dead Zone Clipper are:





Solution 2: Dead Zone Clipper

19

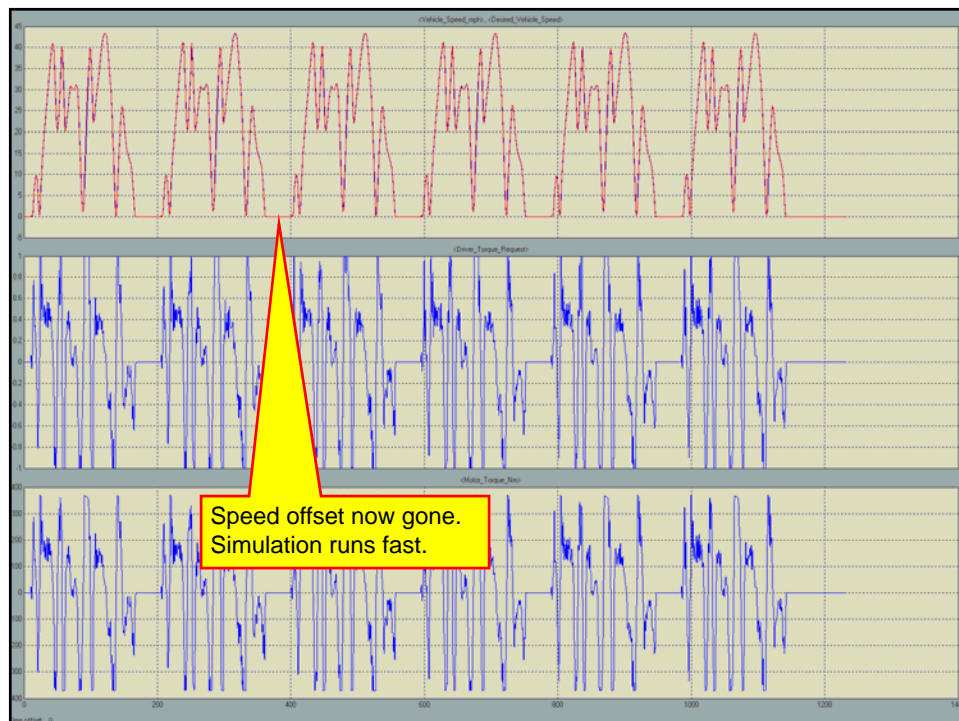
- The Dead Zone clipper is set to eliminate any torque requests between -0.001 and +0.001.
- These limits were chosen as a first guess.
- The values are small enough to not significantly affect the torque request.
- However, tiny torque requests will be blocked, and hopefully solve the problem.
- When we run this solution, the model appears to run as fast as the first solution.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Solution 2 – Dead Zone Clipper

21

- The Dead Zone solution fixes the simulation speed problem, and does not give the vehicle speed an arbitrary constant offset.
- Simulation speed tests were run using the Consumer Reports City cycle. The time to complete the simulation for the various methods were:
 - No Solution – Original Model: 529 Seconds
 - Constant 0.3 mph Offset: 70 Seconds
 - Dead Zone Clipper: 67 Seconds
- We will use the Dead Zone Solution because it does not add an arbitrary offset to the vehicle speed.



Brake Models





Brake Models

23

- At this point the only method of slowing the vehicle is to use the motor to apply a negative torque to the rear wheels.
- The benefit is that we recapture all of the possible regenerative braking energy available.
- There are two problems with this method, one fairly obvious, the other a bit subtle.
- The first problem is that if the motor, motor controller, or powertrain fail, there is no way to slow the vehicle. Thus, we need a backup.



Brake Models

24

- The second, less obvious problem, is that the motor slows the vehicle by applying a negative torque to the powertrain. This raises two issues:
 - If a tire breaks free, the wheel will actually spin at high speed in the opposite direction of vehicle movement. (With mechanical friction brakes, when there is too much braking torque, the wheel just locks).
 - At low vehicle speeds, since braking applies a negative torque, if we are not careful, pressing the brake could cause the vehicle to move backwards. (To prevent this problem, we do not allow regenerative braking below certain vehicle speeds.)





Braking Methods

25

- We will show two brake models:
 - The first method uses a torque actuator and applies a torque to the half shaft in the opposite direction of tire rotational velocity. (This method has the problem of causing the tire to spin backwards during hard braking).
 - The second method uses a friction clutch to apply a torque between the half shaft and an immovable object. (This method locks the wheel during hard braking.)



Braking Methods

26

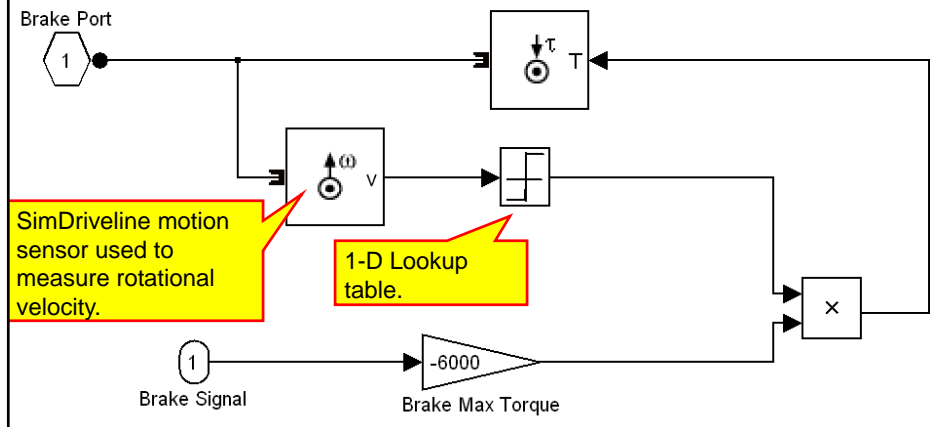
- Both methods will cause problems around vehicle speed.
 - This causes the simulation to run slowly just as it did with the problem in the driver feedback loop around zero vehicle speed.
 - We will employ a similar solution to mitigate this problem.



Method 1 – Opposing Torque

27

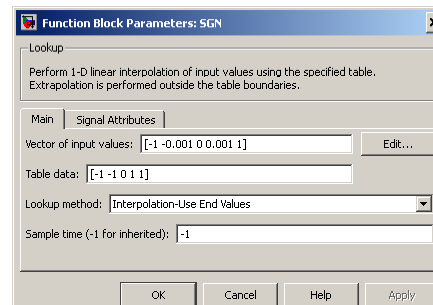
- The first model will sense the direction of rotation and apply a torque in the opposite direction. The basic model is shown below:

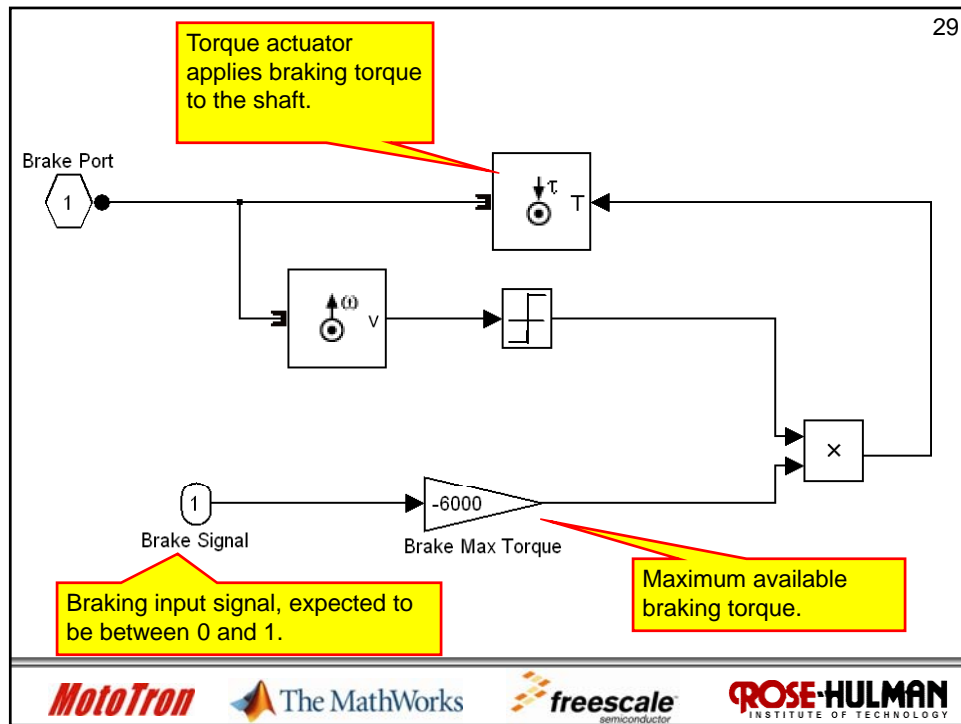


Method 1 – Opposing Torque

28

- The 1-D lookup table is used to output the direction of rotation.
- If the rotational velocity is negative, the table outputs a -1. If the rotational velocity is positive, the table outputs a +1.
- This lookup table is used to switch the direction of applied torque based on the direction of shaft rotation.





Method 1 – Opposing Torque

30

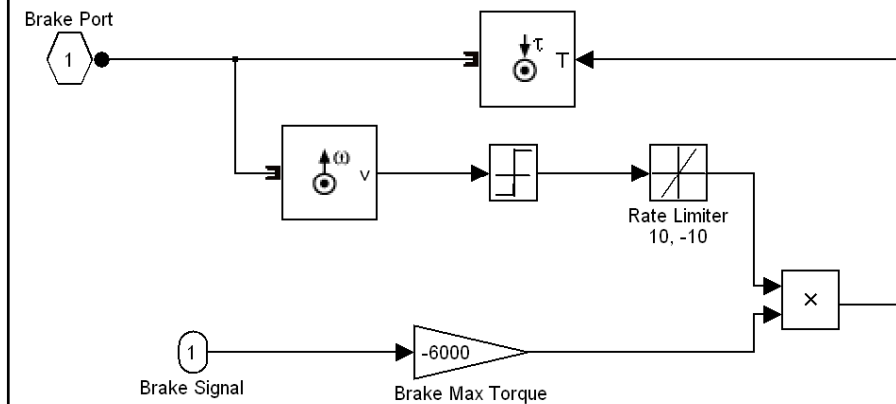
- There are some issues with this model, some of which might be obvious.
- First, when the shaft is near zero speed, we can foresee the output of the motion detector flipping back and forth around zero. As it flips, the output of the lookup table will flip causing the braking torque to flip in the opposite direction.
- We should expect that the braking torque will oscillate rapidly between positive and negative torques as the tire speed reaches zero.



Method 1 – Opposing Torque

31

- To prevent wild oscillations, we will slow down the output of the lookup table with a rate limiter.



Method 1 – Opposing Torque

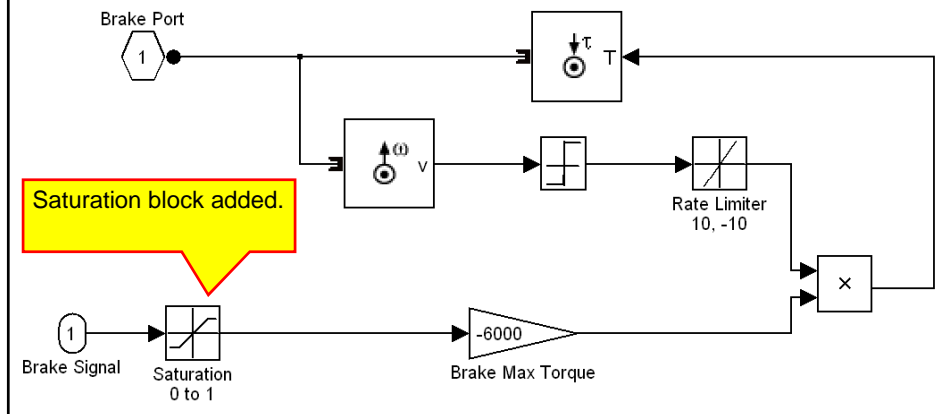
32

- The rate limiter has positive and negative slew rates of 10 per second.
- Since the output of the lookup table flips between -1 and +1, the brake can swing from full positive torque to full negative torque (or vice versa) in 0.2 seconds (2/10).
- This reduce slew rate prevents wild oscillations in the applied braking torque when the tire speed is zero.

Method 1 – Opposing Torque

33

- Next, we need to limit the controlling input signal to be between 0 and 1.
- We expect this range of inputs, but we will add a saturation block just in case the user of the brake subsystem makes a mistake:



Method 1 – Opposing Torque

34

- If the user slams on the brakes, we do not want to apply a step change in torque to the brakes.
- Large step changes in torque brake shafts.
- To prevent this, we will add a rate limiter that reduces the rate at which we can apply torque with the brake.
- We will use a positive slew rate of 4 and a negative slew rate of -10.

Method 1 – Opposing Torque

35

- The control signal goes from 0 to 1. Having a positive slew rate of 4 means that the signal will travel from 0 to 1 in $\frac{1}{4}$ seconds, or 250 ms.
- The auto industry says that humans interpret a response time of 250 ms as instantaneous.
- Thus a slew rate of +4 on the brake signal will not appear to have a delay to a human operator.
- A negative going slew rate of 10 was chosen so that the brake releases quickly, as this is not a change in torque that will break a shaft. (Removal of torque rather than applying a step increase in torque.)

MotoTron

The MathWorks

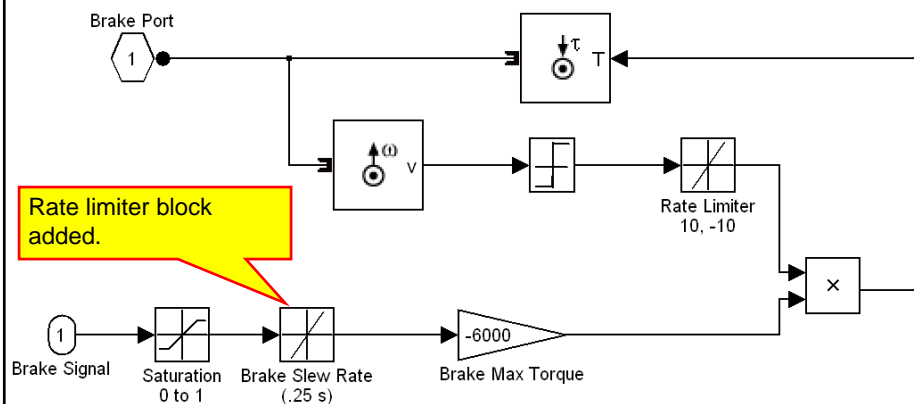
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Method 1 – Opposing Torque

36

- The rate limiter is added as shown below:



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Method 1 – Opposing Torque

37

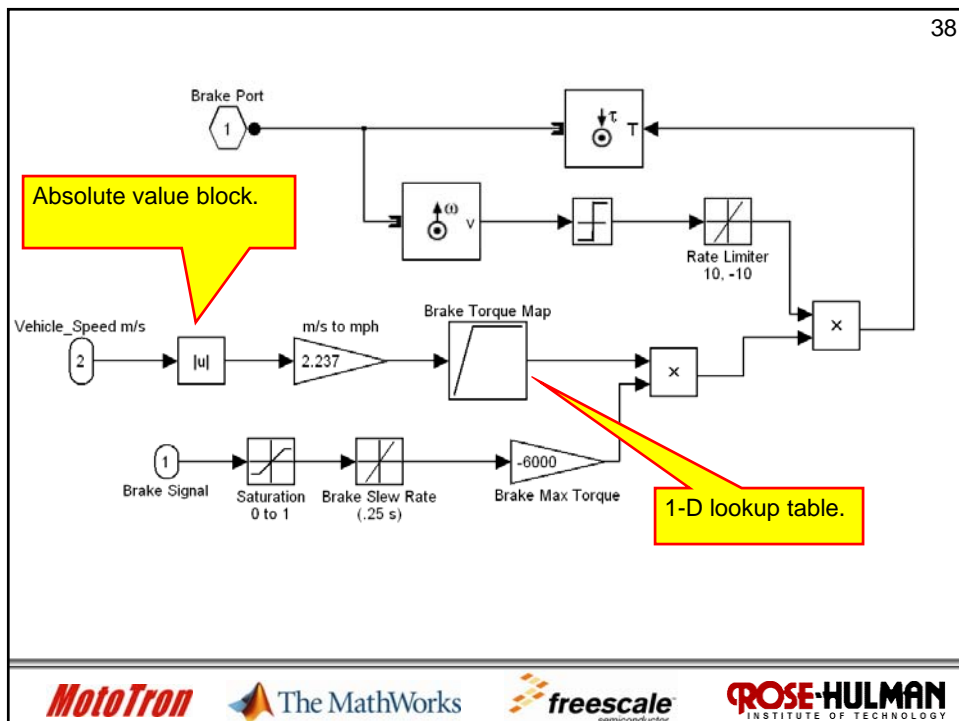
- The last problem we need to address is the same problem we observed with the driver feedback loop.
- When the vehicle speed is zero, the previously shown brake model would cause the simulation to run extremely slowly.
- The cause is the same. The brake would attempt to hold the vehicle at zero speed by applying a torque to the shaft that would oscillate between positive to negative values.
- The solution is the same. We will use a lookup table to reduce the torque to zero when the vehicle speed is zero.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTron

The MathWorks

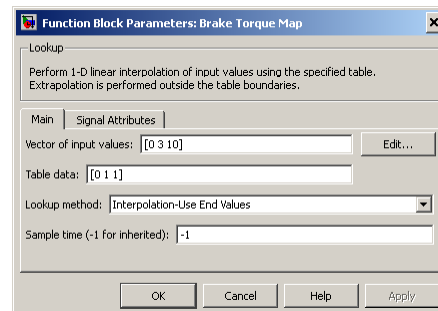
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Method 1 – Opposing Torque

39

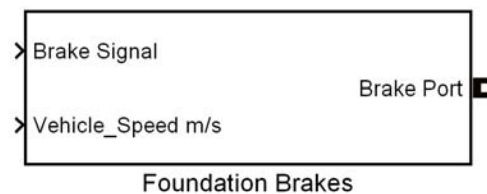
- The 1-D lookup table reduces the braking torque to zero at low speeds.
 - For speeds 3 mph and higher, the output of the table is 1 and the braking torque is the torque requested by the driver.
 - As the vehicle speed drops from 3 mph to 1 mph, the braking torque is linearly reduced to zero.



Method 1 – Opposing Torque

40

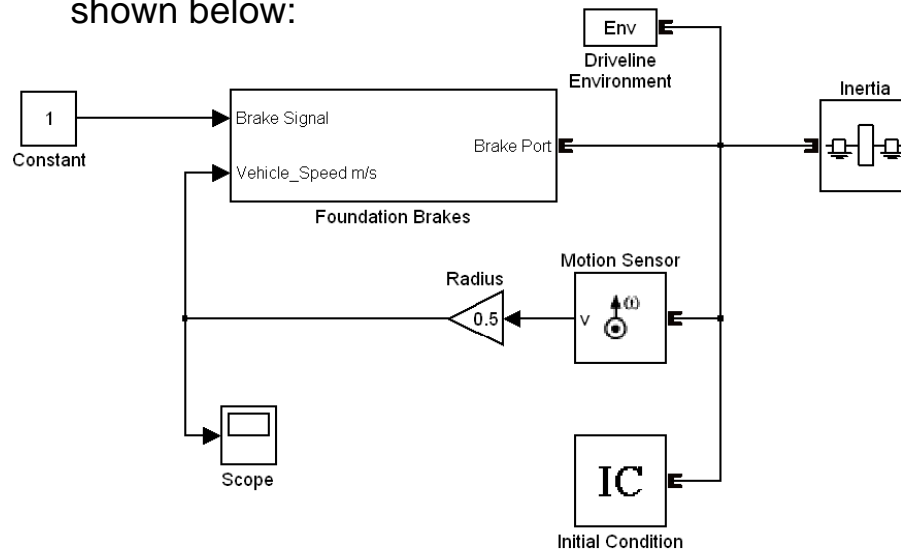
- We will place the brake model within a subsystem so that we can use it several times within the same model.



Brake Testing

41

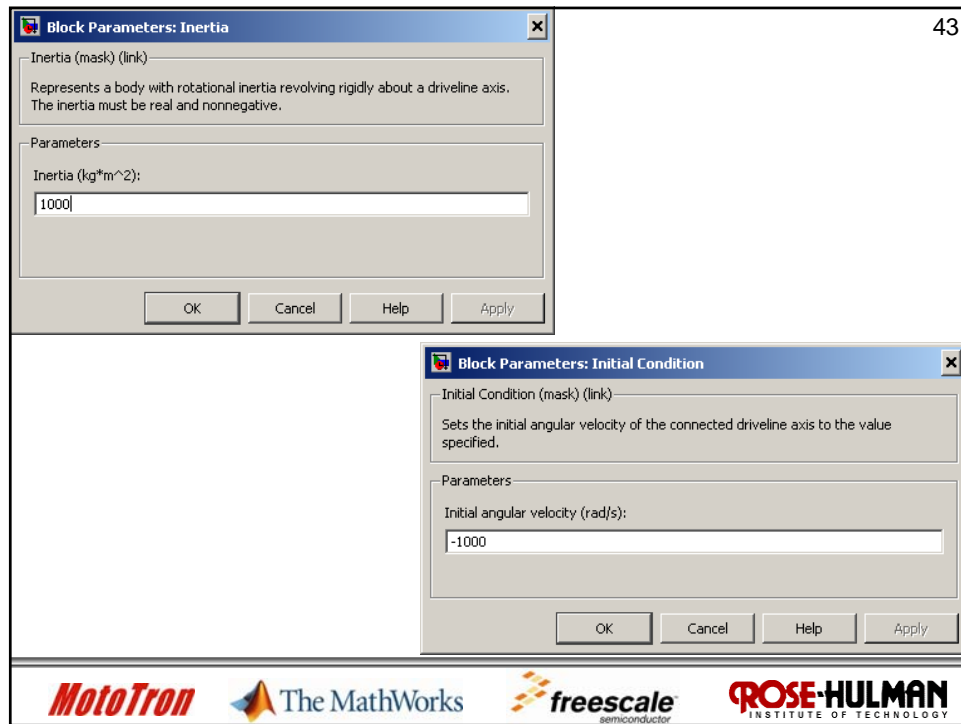
- We will test the brake with the testing harness shown below:



Brake Testing

42

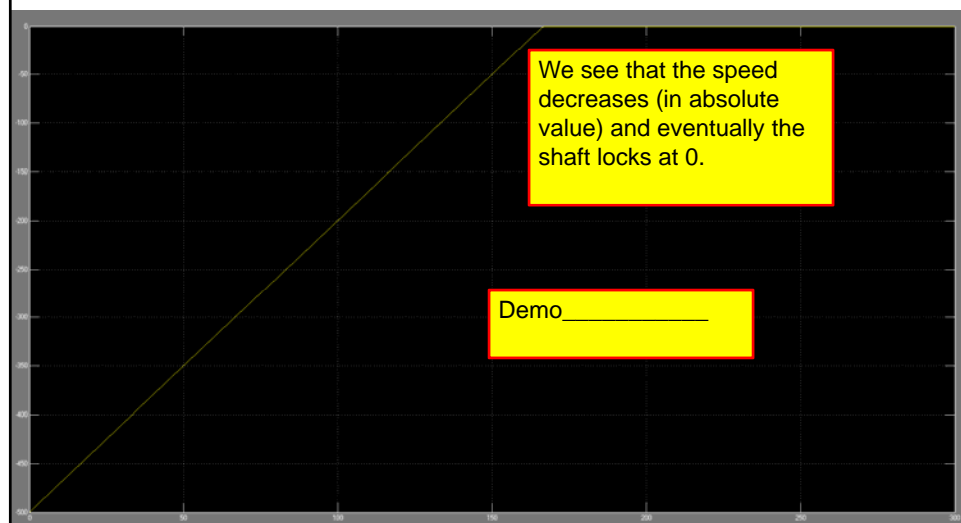
- The basic idea is to give a large inertia an initial velocity with the IC block and then use the brake to slow the speed down to zero.
- We will also do a lot of testing with the brakes in the vehicle model. However, it will take a lot of work to incorporate the brake into our model, so we will use the testing harness as a preliminary test.
- The blocks have the values shown on the next slide.



43

Brake Testing

- For an Initial condition of -1000 rad/sec, the plot of the speed is shown below:

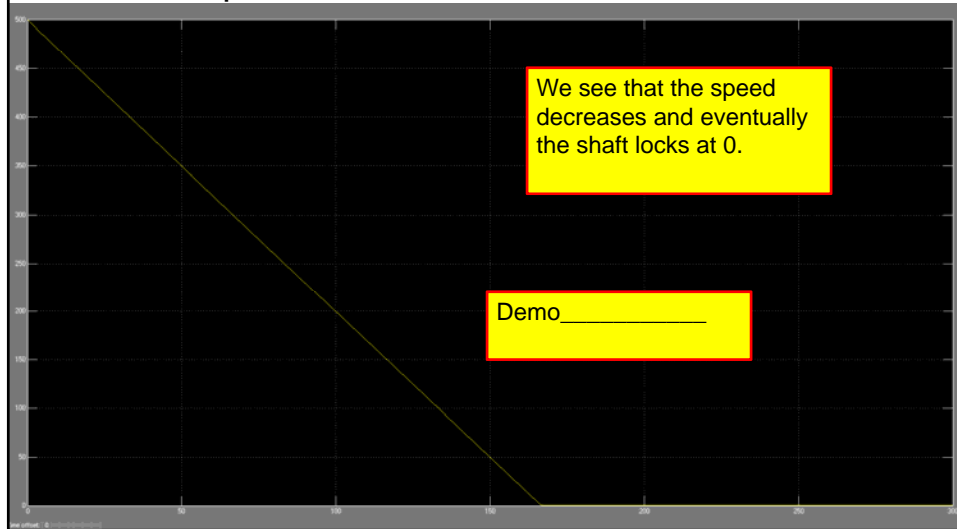


44

Brake Testing

45

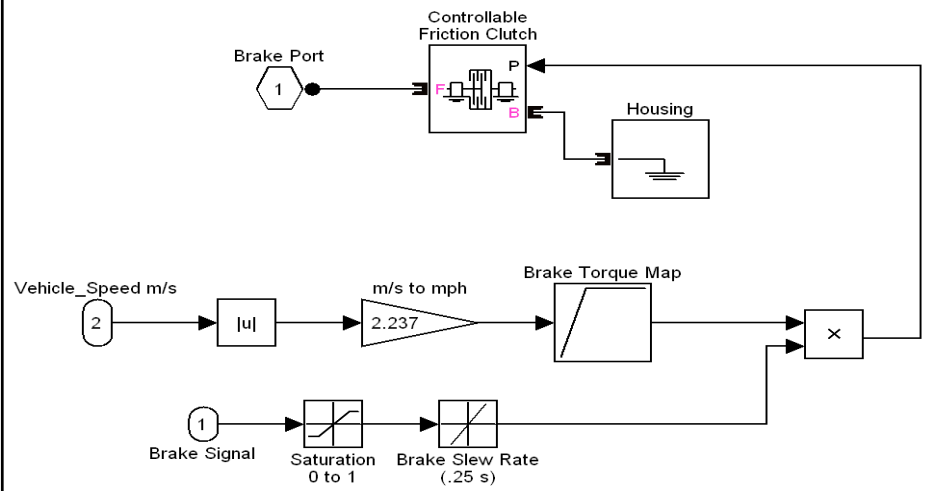
- For an Initial condition of +1000 rad/sec, the plot of the speed is shown below:



Brake Model – Method 2

46

- The second method uses many of the same components as the first method except that we use a clutch rather than a torque actuator:





Method 2

47

- This model uses a friction clutch to apply torque in the form of friction between the rotating shaft (input port of the brake) and an immovable object (in this case the housing).
- The housing can be thought of an infinite inertia. It takes an infinite amount of torque to make it spin.
- The properties of the friction clutch are shown on the next slide.



Brake Method 2

48

Parameters

Directionality:

Number of friction surfaces:

Effective torque radius (m):

Peak normal force (N):

Coefficient of friction table:

Static friction peak factor:

Engagement threshold for normalized pressure:

☒ Use default clutch velocity tolerance from the Driveline Environment block

Clutch velocity tolerance (rad/s):

OK Cancel Help Apply

- The applied torque is equal to the normal force times the times the radius of the friction surface times the number of friction surfaces. In our case the peak torque we can apply is 6000 Nm, the same as in method 1.



Clutch Model

49

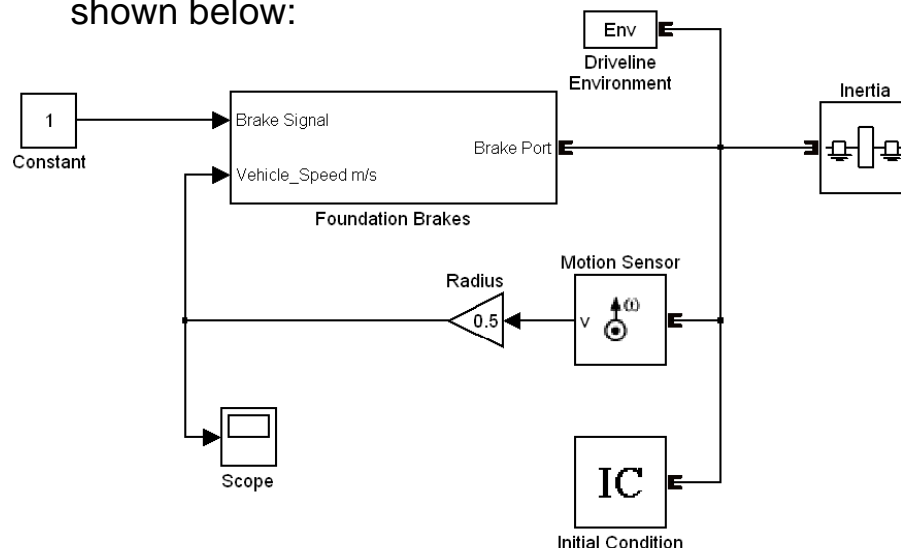
- The clutch has a pressure input (P).
- Allowable values are 0 to 1.
- As the pressure signal goes from 0 to 1, the applied frictional torque goes from 0 to 6000 Nm (for our settings).
- Next, we will test this brake using the same harness we developed earlier.

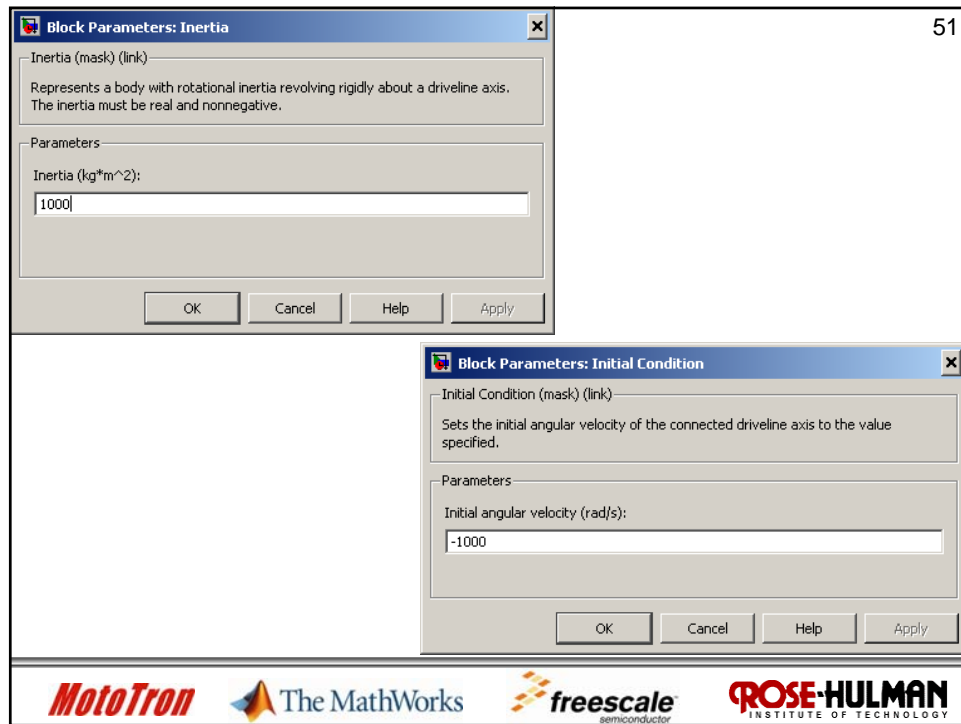


Brake Testing – Method 2

50

- We will test the brake with the testing harness shown below:



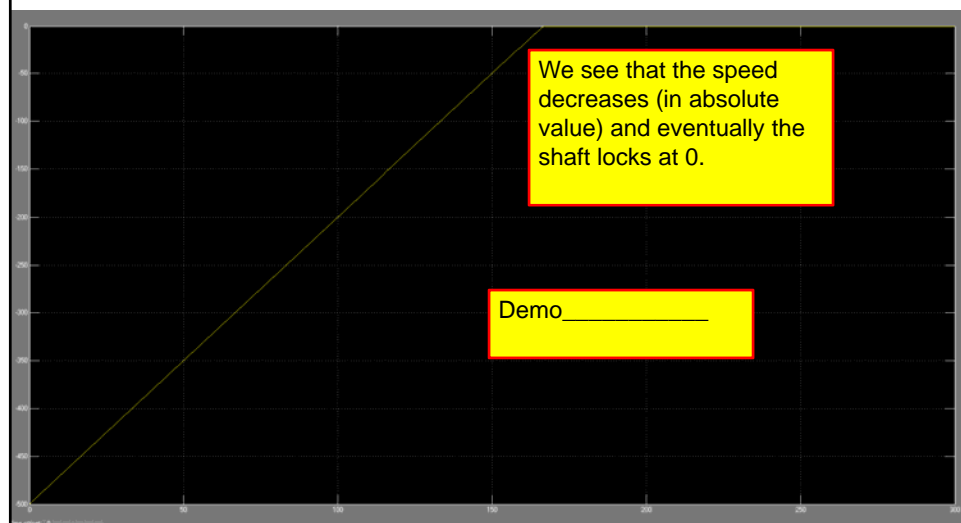


51

Brake Testing

52

- For an Initial condition of -1000 rad/sec, the plot of the speed is shown below:





Brakes

53

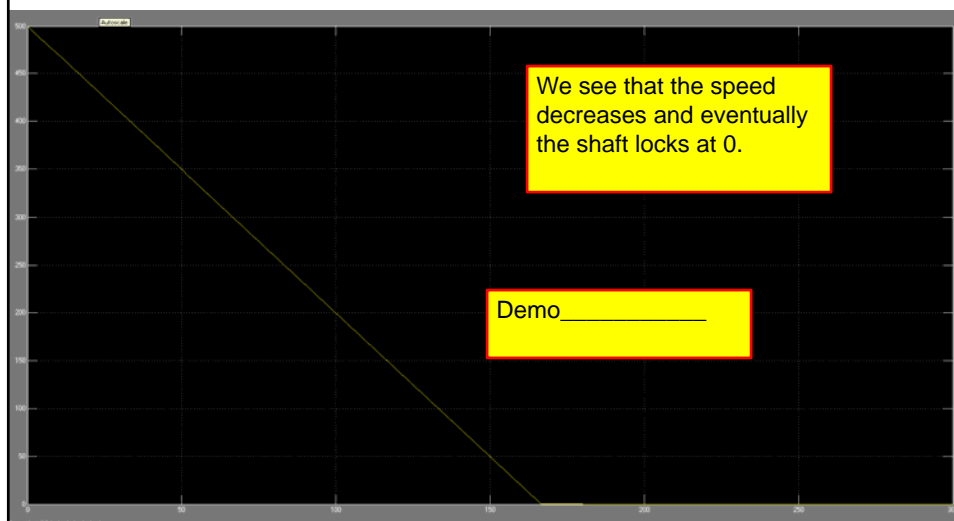
- The two models appear to behave in a similar fashion.
- We will use the clutch model in the vehicle.
- Don't ask why.....



Brake Testing

54

- For an Initial condition of +1000 rad/sec, the plot of the speed is shown below:





Advanced Model-Based-System Design

Lecture 9: Brakes Part 2: Brake Controller



Incorporating the Brake Model

2

- Now that we have a mechanical friction brake model, we need to incorporate the brake model into our vehicle. This will require three steps:
 - Connecting the brakes to the half-shafts
 - Modifying the driver model to emit acceleration and braking signals.
 - Designing a brake controller so choose between the foundation brakes (mechanical friction brakes) and the regen braking (the electric motor).



Connecting the Brakes

3

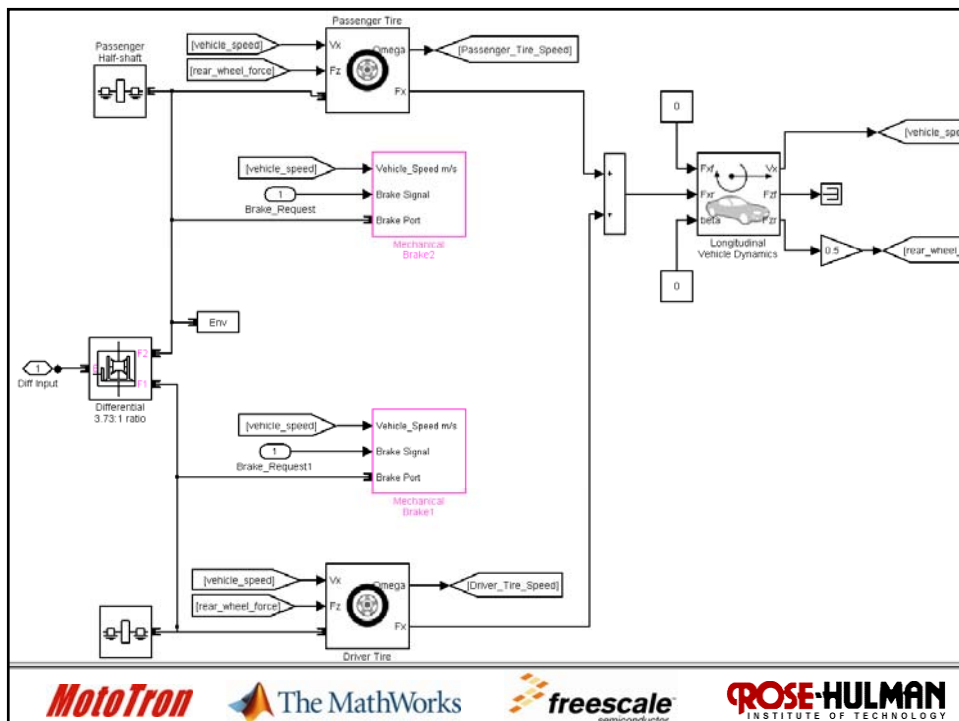
- The brakes are connected to the rear half-shafts (the shafts between the rear differential and the wheels).
- We will start with model Lectue9_Model3 that we modified in Lecture 9 and improved the simulation speed.
- Save this mode as Lecture10_Model1.
- Add the mechanical brakes as shown:

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTron

 **The MathWorks**

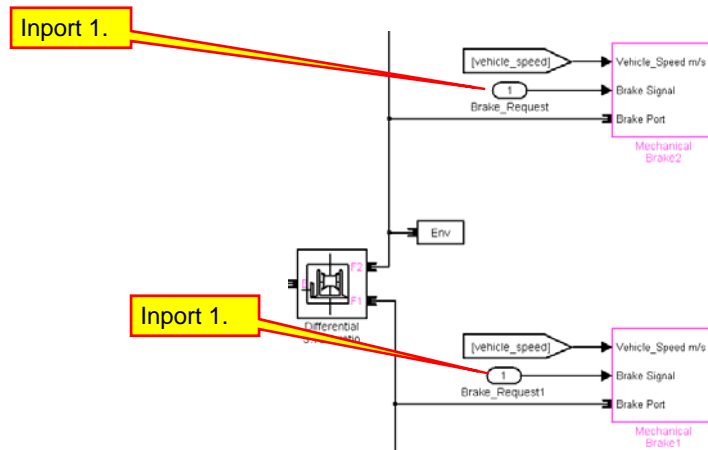
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Connecting the Brakes

5

- The brake model has:
 - Been resized to fit in a small space.
 - Two Inports that are labeled as “1”:



Inports

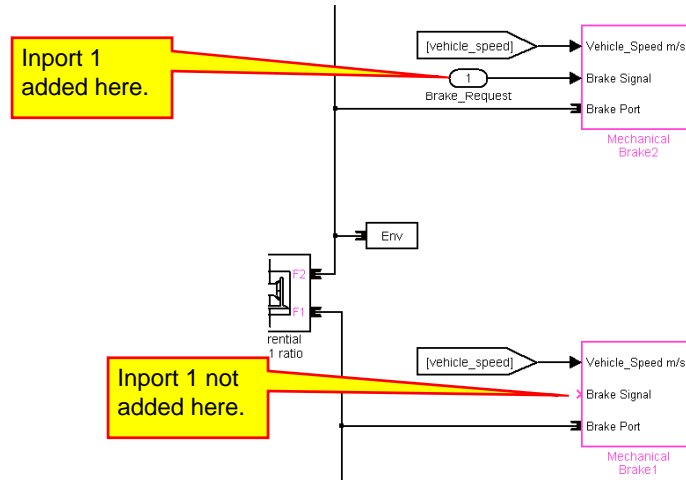
6

- We notice that both Inports are labeled as port 1.
 - This means that the ports are the same port.
 - ➔ The Rear Diff and Body subsystem will only have a single Simulink (not two).
 - We can use these duplicated Inports instead of using From and Goto blocks.
 - The duplicated port was created using the following procedure.

Inports

7

- Place a single Inport in your drawing:



MotoTron

The MathWorks

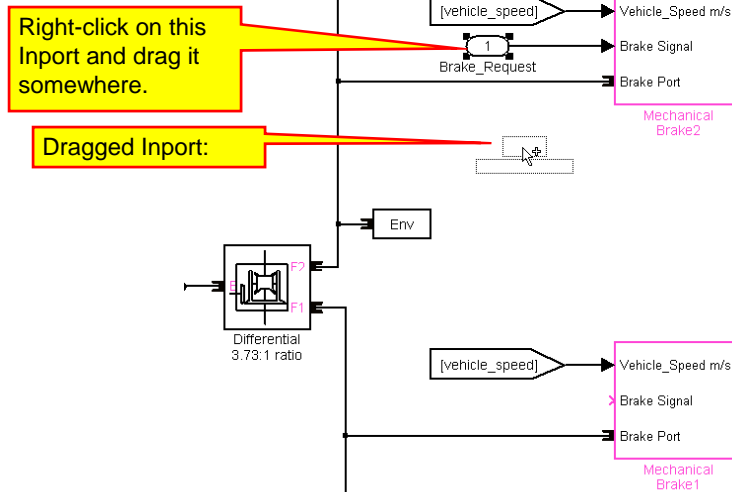
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Inports

8

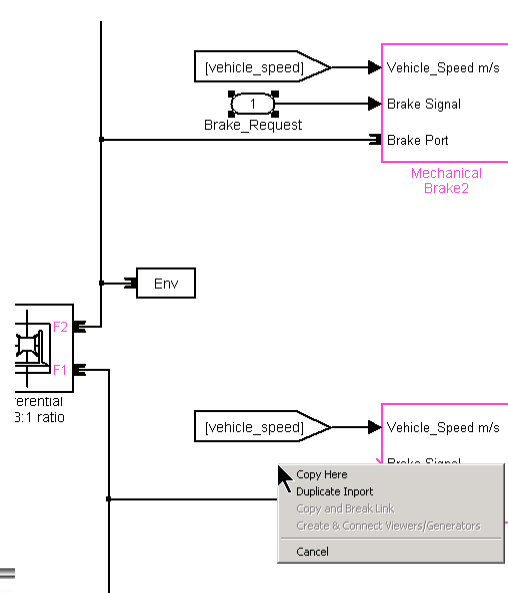
- Right-click and drag the Inport you want to duplicate:




9

Inports

- Drag the port next to the lower brake subsystem and release the mouse button.
- Simulink will ask if you want to Copy or Duplicate the port.






MotoTron 

10

Inports

- Selecting **Copy** will create a new port (number 2 in this case) and will name it with the same name as the original port and append a 1 to the name.
- Selecting **Duplicate Inport** will create a port with the same number (1 in this case) and will name it with the same name as the original port and append a 1 to the name.
- Duplicating a port does not create a new port. It creates a connection to an existing port without using From and Goto blocks.

MotoTron   



Driver Model

13

- We now need to update the driver subsystem.
- The driver block knows what the desired and actual vehicle speeds are and emits the driver torque request to try to make the actual vehicle speed equal the desired speed:
 - The emitted driver torque request is a number between -1 and +1.
 - If the vehicle speed is too slow, the driver block emits a positive signal to accelerate the vehicle.
 - If the vehicle speed is too high, the driver block emits a negative signal to decelerate the vehicle.



Driver Model

14

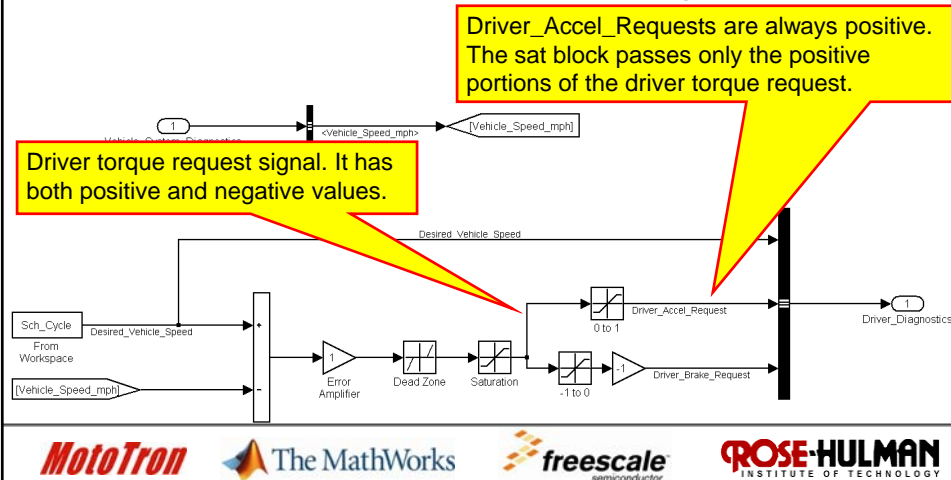
- Thus, we see that positive driver torque requests are acceleration requests and negative torque requests are braking requests.
- We can easily split the driver torque request into two separate signals:
 - The driver accelerator request which are the positive values of the driver torque request signal.
 - Driver brake request which are the negative values of the driver torque request.



Driver Model

15

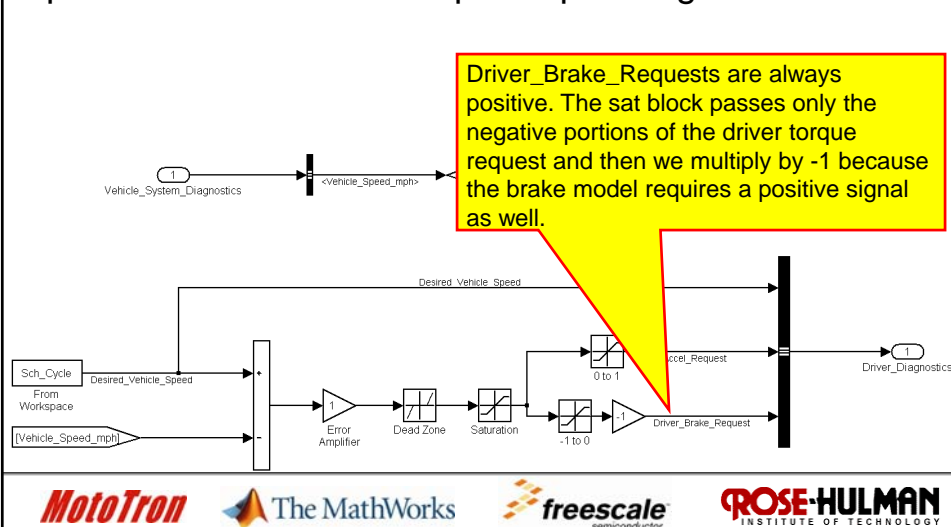
- We will use saturation blocks to split the signal in half. The Driver_Accel_Request is the positive portions of the driver torque request signal.



Driver Model

16

- The Driver_Brake_Request signal is the negative portions of the driver torque request signal.





Driver Signals

17

- Note that:
 - A positive value in the Driver_Brake_Request means slow down the vehicle.
 - A positive value in the Driver_Accel_Request means speed up the vehicle.
- These two signals are one step in preparing our model for the Hardware-in-the-Loop (HIL) simulations we will do later where these two inputs will come from actual brake and acceleration pedals.



Driver Block

18

- Also note that our driver block now has a single output.
 - The Driver_Accel_Request and Driver_Brake_Request signals are not contained in the Vehicle_System_Diagnostics bus.





Brake Controller

19

- The meat of this section is the development of a brake controller.
- We have both regen brakes (using the electric motor to slow down the vehicle) and foundation brakes (mechanical friction brakes).
- Foundation brakes result in 100% energy loss → all of the braking energy goes into generating heat.
- Regen braking allows you to recapture some of the vehicles kinetic energy and store it in the battery.



Brake Controller

20

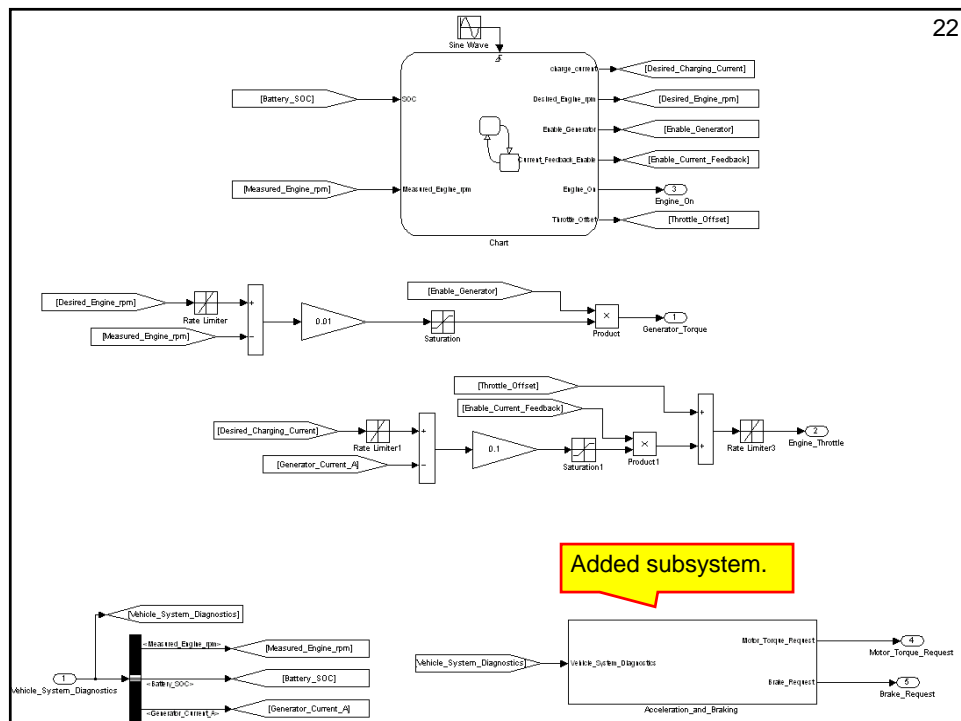
- Several different concerns will be included in our braking control strategy:
 - Fuel efficiency – We will create a strategy that uses regen braking before foundation brakes so that we maximize the energy we recapture.
 - Safety
 - Both regen and foundation brakes are required in case one method fails.
 - We need to prohibit regen braking when the battery state of charge is too high (When the battery is charged, it cannot accept charge).
 - Driver feel – We will be switching between regen and foundation brakes as the vehicle changes speed. We do not want the driver to notice a difference as we switch between the two methods.



Brake Controller

21

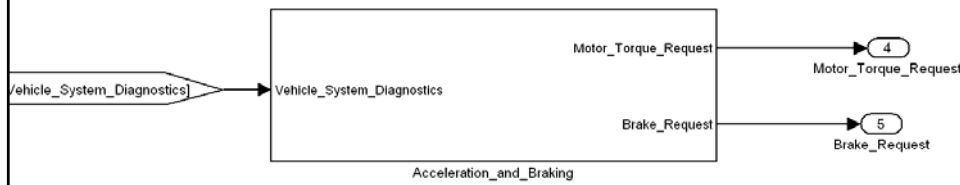
- We will add a new subsystem to the Controller called Acceleration_and_Braking
- The input to this subsystem will be the Vehicle_System_Diagnostic bus.
- The Outputs of the system will be:
 - Motor_Torque_Request (responsible for acceleration and regen braking).
 - Brake_Request (foundation brake signal)
- The Controller is shown on the next slide:



Accel and Brake Controller

23

- An enlargement is shown below.
- Note that the outputs of the Acceleration_and_Braking subsystem are also the outputs of the Controller subsystem.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Accel and Brake Controller

24

- Before we build the subsystem, we note that the motor is responsible for both acceleration and regenerative braking.
- Thus, the motor torque request signal is a combination of the acceleration request and the braking request.
 - A positive acceleration request will be passed to the motor as a positive torque request.
 - A positive braking request will be passed to the motor as a negative torque request.

MotoTron

The MathWorks

freescale
semiconductor

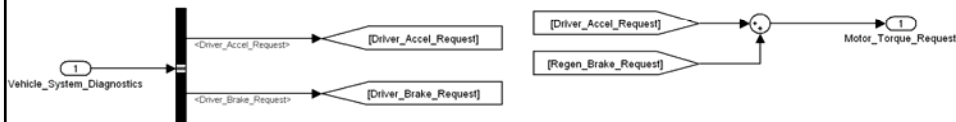
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Acceleration Signal

25

- We will first look at extracting the appropriate signals from the bus and forming the Motor Torque Request Signal. The model is shown below:



- The Driver_Accel_Trequest is passed directly on to the motor as the motor is the only component responsible for accelerating the vehicle. Thus, we will not modify the signal (yet... Later we may do some signal conditioning on the signal to prevent damage to the vehicle to do driver enthusiasm...)

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Acceleration Signal

26



- Note that we can add the accel request to the brake request because when there is a brake request the accel request is zero, and when there is an accel request the brake request is zero.
- Even though motor accel requests are positive and motor regen requests are negative, they will not cancel when we add them because one is always zero.
- Thus, we can form the complete motor torque request signal by adding together acceleration and braking requests.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Braking Method

27

- We will now create a simple braking method to increase energy recovery.
 - When the driver braking request goes from 0 to 50% pedal position, the regen braking request will go from 0 to 100%. That is, we will ask for full regen braking when the brake pedal is 50% depressed.
 - When the driver braking request goes from 0 to 25% pedal position, No Mechanical brake request will be generated.
 - When the driver braking request goes from 25 to 100% pedal position, the foundation braking request will go from 0 to 100%.



Braking Method

28

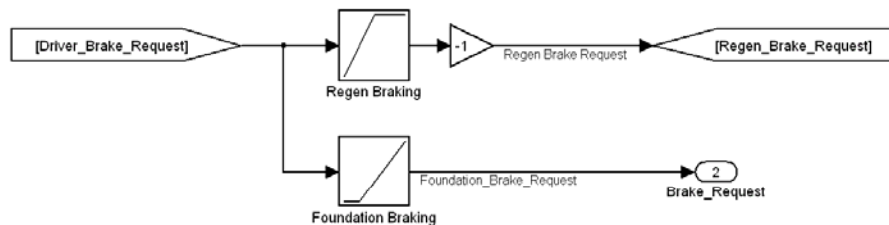
- This method:
 - Allows us to use 100% regenerative braking for light braking requests. (No foundation brakes.)
 - Uses both the foundation and regen brakes for safety through the use of both braking systems at the same time.
 - Creates a dead spot in the brake pedal with no braking regen braking is disabled (which will happen quite frequently. → Need to fix this.)



Braking Method

29

- We will use look-up tables to implement this method. The Simulink block diagram is shown below:



MotoTron

The MathWorks

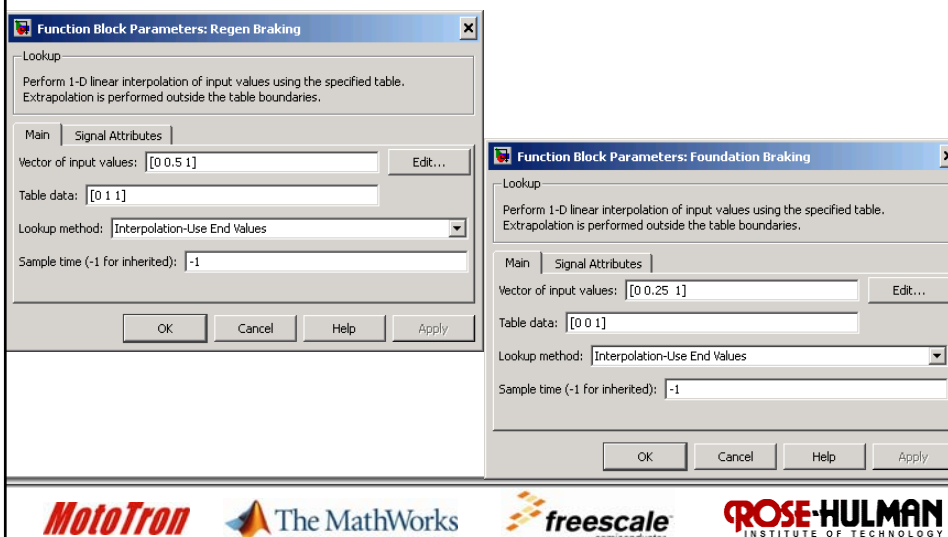
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Braking Method

30

- The parameters for the two look-up tables are:



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Braking Method

31

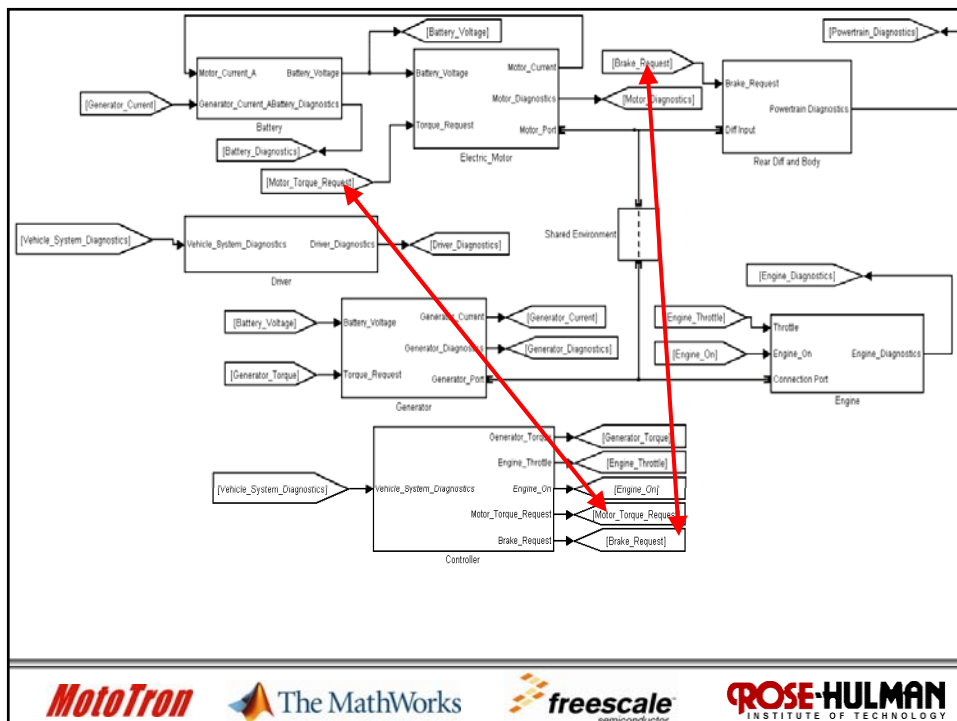
- Note that a negative brake request is passed to the motor as a negative torque request.
- We have now created a simple braking controller.
- The last thing we need to do is connect the Brake_Request and Motor_Torque_Request signals at the top-level block diagram:

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Brake Model Test

33

- To test the braking method we will run the FU505 model and plot:
 - Vehicle, Passenger Tire, and Driver Tire Speeds
 - The Driver_Accel_Request and Driver_Brake_Request
 - Battery Voltage
 - Motor and Generator Currents
 - Battery SOC
 - Regen Brake Request and Foundation Brake Request
- Show tat your model uses both regen abd foundation braking.



Lecture 10 Demo 1

Demo_____

34





Braking Results

35

- Note the following from the results of the previous simulation:
 - We successfully split the driver torque request into acceleration and Braking requests.
 - For the FU505, most of the braking is done through regen braking (which suggests that a 25% pedal throw is only needed for this drive cycle).
- Just for fun, run the Consumer Reports City cycle and observe the amount of regen and foundation braking:

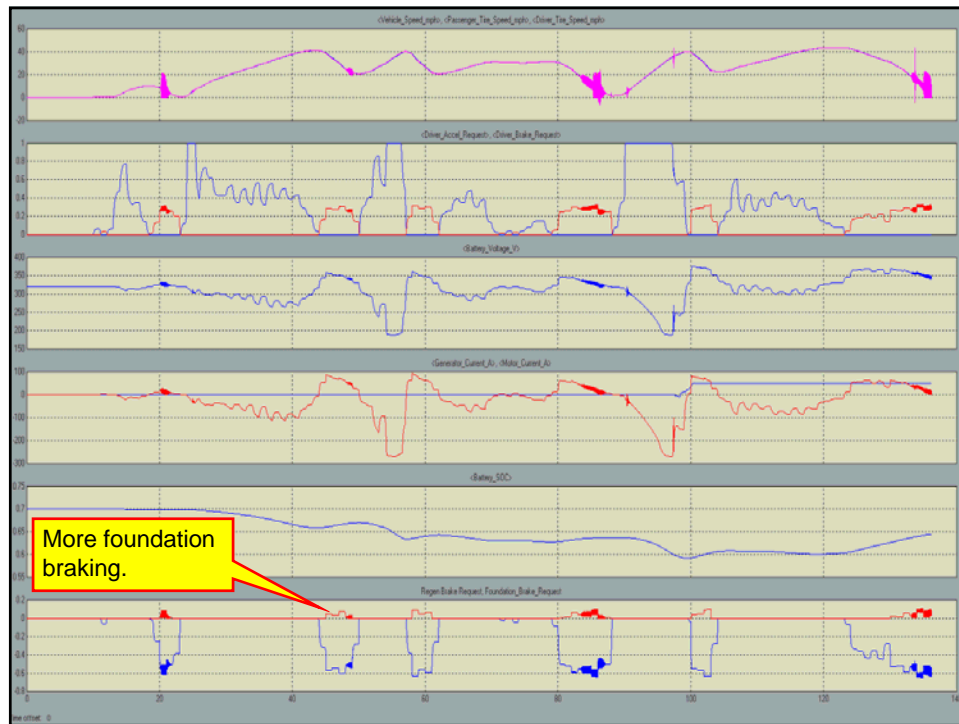


Braking Results

36

- The Consumer Reports Drive Cycle has instances of much harder acceleration and braking than does the FU505.
- The simulation crashes after a while due to too many zero crossings, which suggests that the solver is having trouble simulating the system.
- We do see that there are more foundation braking requests than in the FU505 cycle because of the heavier braking.





Braking Problem

38

- We do notice a problem with the simulation now.
- We see this hash that we did no see earlier.





Lecture 10 Exercise 1

39

- What is the “hash” that you are seeing in the previous slide?
- What is the cause of the problem?
- Determine a way to fix the problem and demo the Consumer Reports City drive cycle showing that the problem has been eliminated.

Demo_____



Lecture 10 Exercise 2

40

- In Lecture 8 we calculated the fuel efficiency of a model that used only regen braking to slow the vehicle.
- In this lecture, we have now added foundation brakes, which can only reduce the efficiency of our vehicle.
- Compare the efficiency of your vehicle from lecture 8 to the efficiency of this vehicle using of the FU505 and Consumer Reports City drive cycles.

Demo_____





Regen Braking and SOC

41

- As the battery SOC increases, the battery's capability of accepting charge decreases. (A fully charged battery cannot accept charge.)
- There are two things we need to consider:
 - Our control strategy needs to be designed (optimized) so that the battery remains somewhat discharged so that we can always take advantage of regen braking (we will not address this here).
 - If the battery SOC becomes too high, we need to disable regen braking.
- We will now reduce and, if necessary, disable regen braking if the Battery SOC becomes too high.





Advanced Model-Based-System Design

Lecture 11: Partitioning the System High-Level Control



Incorporating the Brake Model

2

- We now have a fairly large and complex system.
- We would like to arrange it more like a classical control system and also arrange it to easily facilitate deploying the models on our Hardware-In-the-Loop (HIL) system.
- We would like to set up the system in the form of a controller and a plant. In this case the plant is everything in our vehicle model except for the controller and driver block.
- The controller receives commands from the driver block and coordinates the subsystem components inside the plant.



Partitioning the System

3

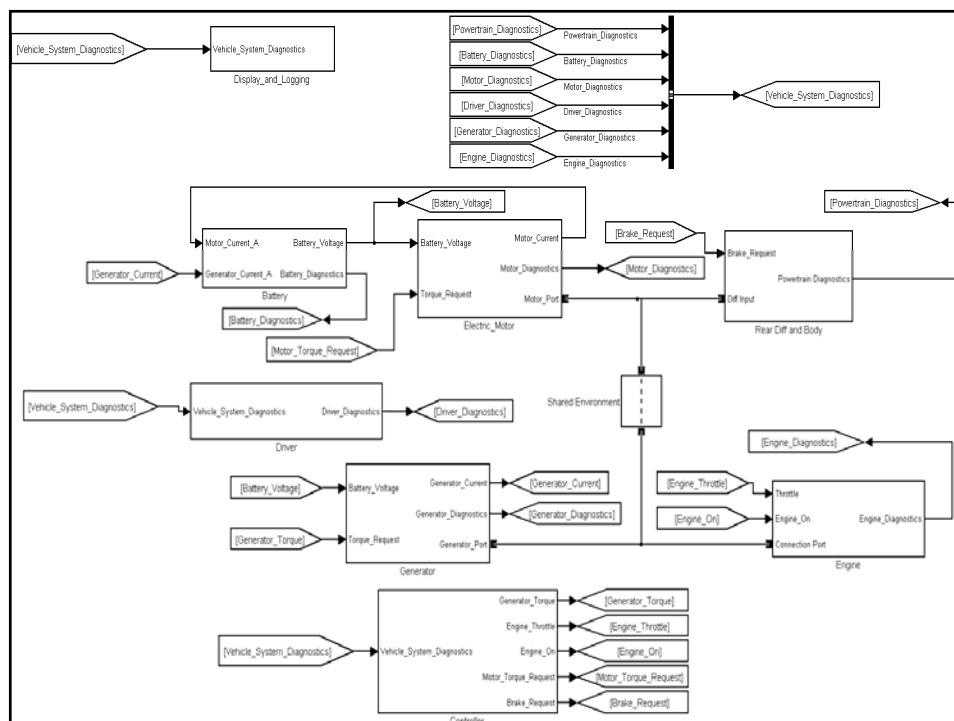
- We would like to set up a classic-looking control system.
- We will have the driver subsystem interface to the controller, and then the controller interface to the plant which contains all of the other subsystem components of the vehicle except the logging and visualization subsystem.
- We will start with the last model developed in Lecture 10 and rename it as Lecture11_Model1.
- This model is shown next:

MotoTron

The MathWorks

freescale
semiconductor

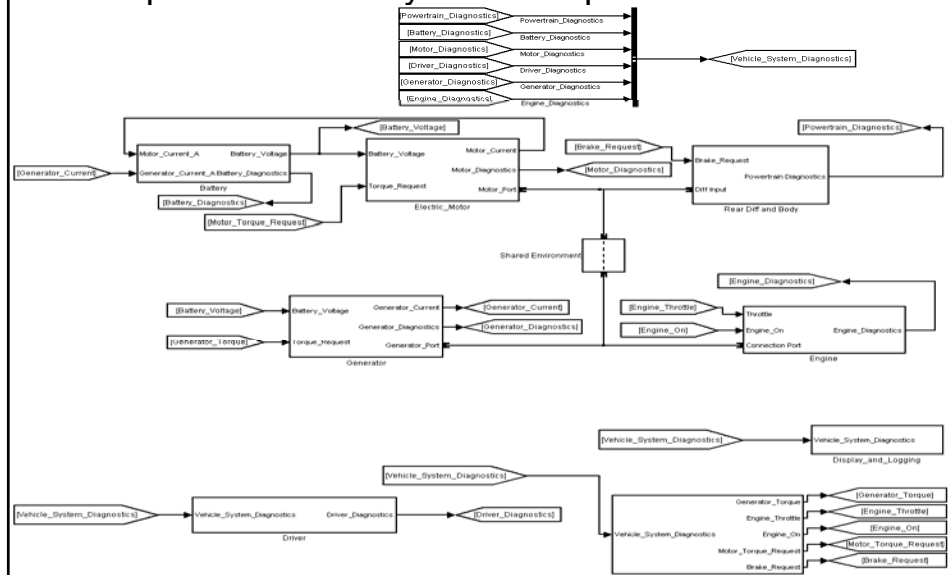
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Partitioning the System

5

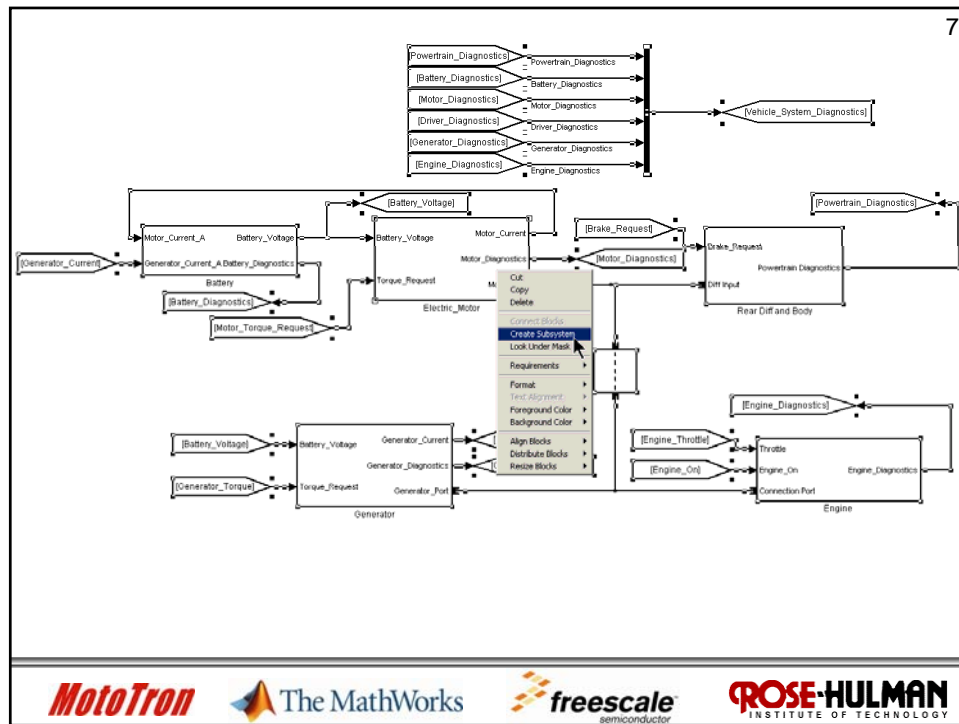
- Separate the subsystem components as shown:



Partitioning the System

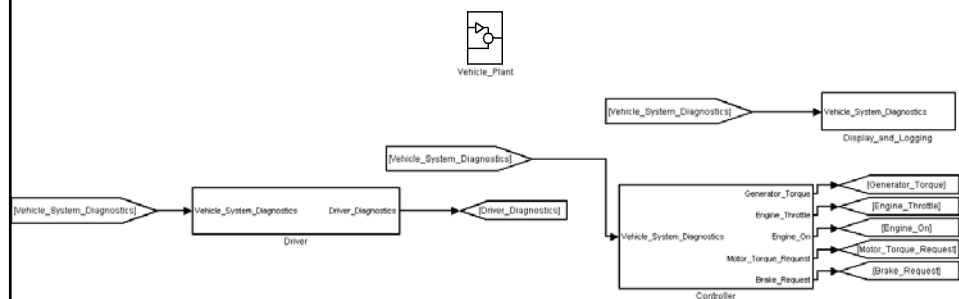
6

- We want to make a subsystem out of the Diagnostic bus, Battery, Electric Motor, Rear Diff and Body, Generator, Engine subsystems.
- Select all of these components and then right-click on one of the selected components and select **Create Subsystem**:



Partitioning the System

- When you select Create Subsystem, all of the selected subsystems will be grouped into a single subsystem.
- Rename that subsystem “Vehicle_Plant.”



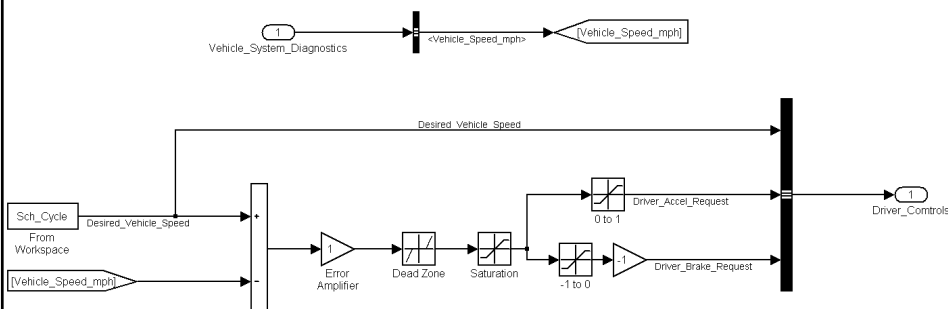
Partitioning the System

- We now need to do a fair amount of reorganization and rerouting of some control signals.
- We will start with the driver. The output signals will be grouped together in what we will now call the “Driver_Controls_Bus.”
- The model does not need to be modified, but is shown next:



Driver Block

- Note that the only reason the driver block needs the Vehicle_System_Diagnostics is because the feedback system that tracks a drive cycle needs to know the vehicle speed. In a future lecture, the driver block will have only outputs unless we implement a cruise control.



Controller Modifications

11

- The Driver_Diagnostics will no longer be part of the Vehicle_System_Diagnostics bus.
- Instead, the driver control signal go directly to the controller, and the controller issues the appropriate commands to the subsystem components based on the inputs provided.
- Modify the controller as shown next. The only change on the top level of the Controller is that the driver controls connect to the controller through a separate port.

MotoTron

The MathWorks

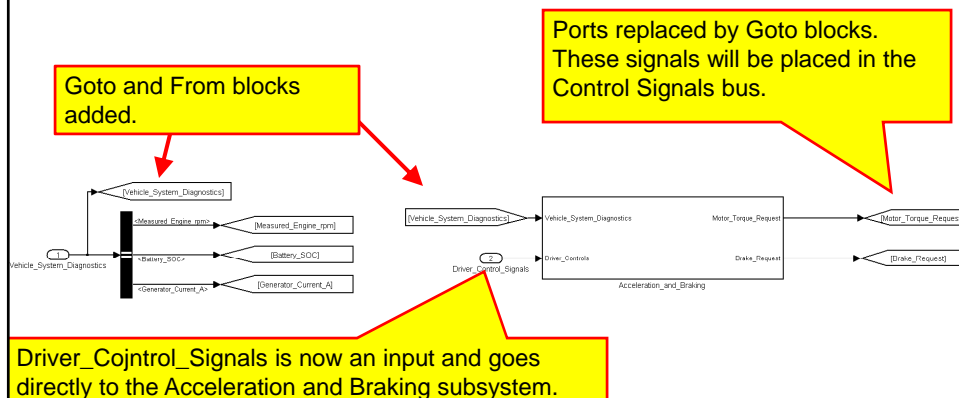
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller Top Level Changes

12

- The changes to the top level of the controller are shown next:

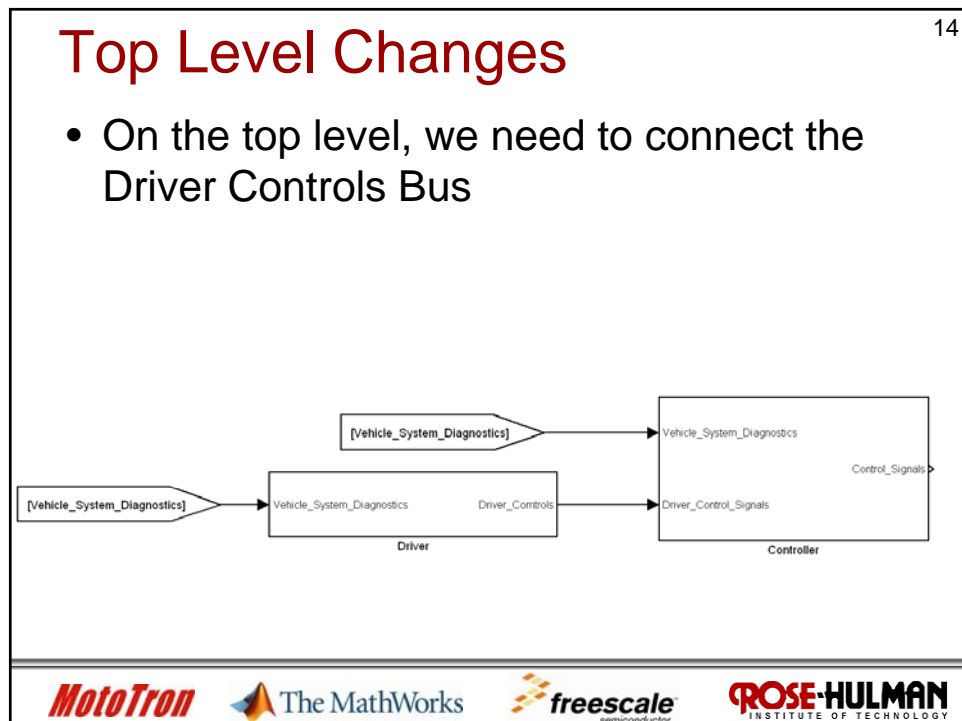
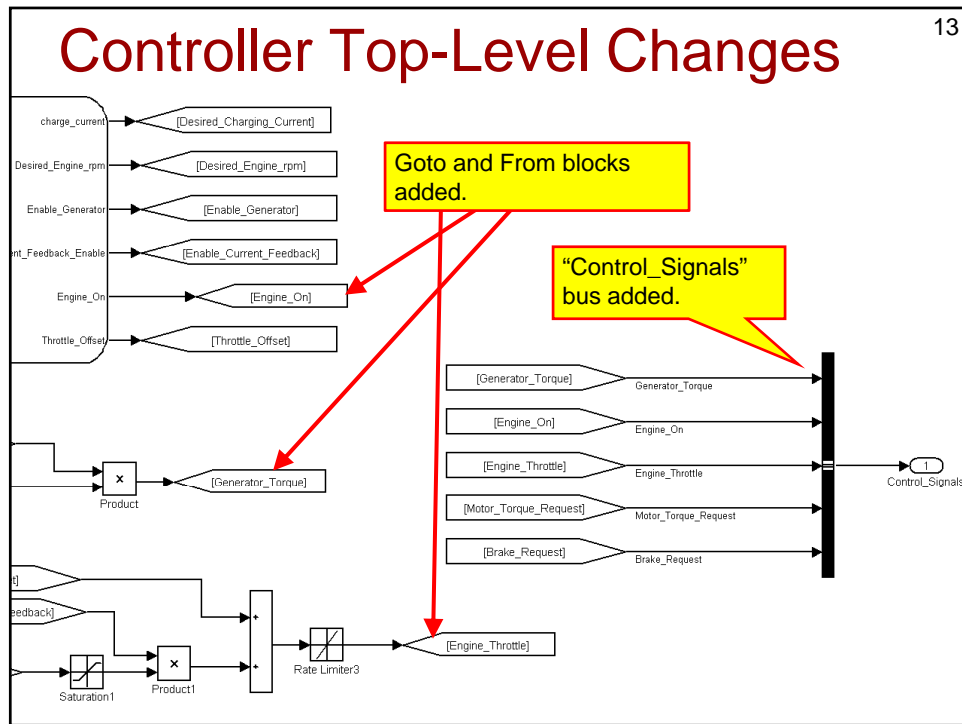


MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Vehicle_Plant Changes

15

- We now need to make some changes in the plant.
- The only input to the Plant is the Vehicle_Control_Signals bus.
- We will leave this signal as a bus because it will go to every subsystem in the Vehicle_Plant.
- Plus, we will be adding many more control signals in the future. Using the Control_Signals but will clean things up a bit.
- The Vehicle_Plant has been modified as shown next:

MotoTron

The MathWorks

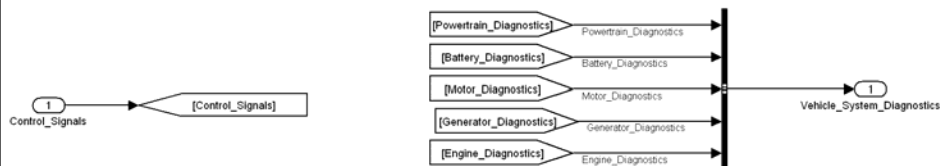
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Vehicle_Plant

16

- Note that we are immediately assign a Goto block to the Control_Signal input.
- Note that we have also removed the Driver_Diagnostics from the Vehicle_System_Diagnostics bus.



- We will be adding additional signals to the Control_Signals bus, so we will leave it as a bus and then it will be an input to all subsystem blocks in the plant.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Vehicle Plant – Battery Subsystem

17

- We will modify one plant subsystem at a time.
- At the moment, the battery has now control signals. This will change when we do our wake-up sequence for the vehicle.
- For now, we will add that Control_Signals bus as an input to the battery, and then terminate that signal inside the battery subsystem. This way, the control signals will be ready to use within the subsystem once the appropriate logic has been discovered.

MotoTron

 **The MathWorks**

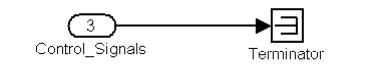
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

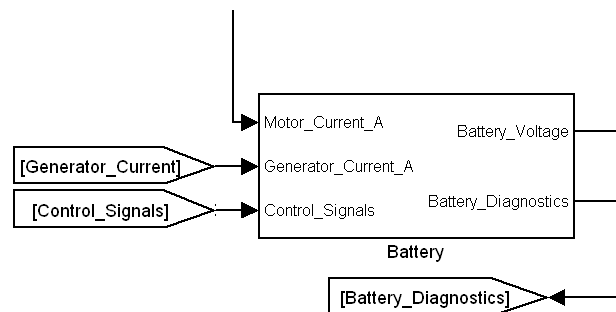
Battery Model

18

- The battery has been modified by adding a new port to connect the Control_Signals bus to:



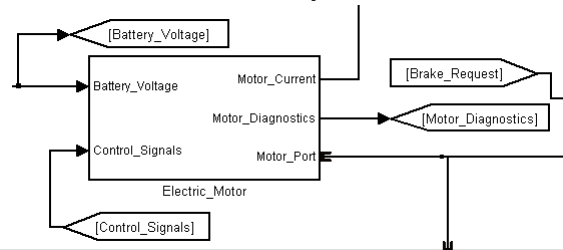
- At the plant level, we connect the control signal bus to the battery Controls_Signal input:



Plant Mods – Electric Motor

19

- The Torque Request for the electric motor will come from the Motor_Torque_Request bus.
- We will change the input of the Torque_Request port to a Control_Signals port. Inside the model, we will need to extract the motor torque from the Bus.



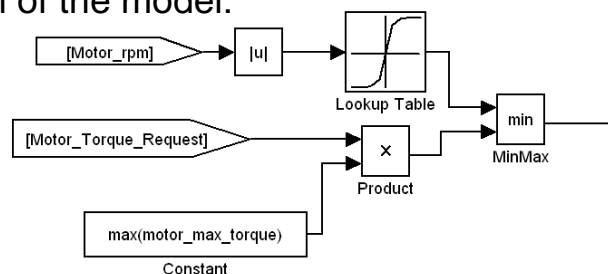
Electric Motor

20

- Inside the electric motor, we need to extract the Motor_Torque_Request signal from the bus:



- And then connect this signal to the torque request portion of the model:



More Changes

21

- We can make similar changes to all remaining subsystems within the Vehicle plant:
 - For the Rear Diff and Body: replace the Brake Request input by the Control_Signals input and extract the Brake_Request signal inside the subsystem.
 - For the Generator : replace the Generator_Torque input by the Control_Signals input and extract the Generator_Torque signal inside the subsystem.
 - For the Engine: replace the Throttle and Engine_On inputs by a single Control_Signals input and extract the Throttle and Engine_On signals inside the subsystem.

MotoTron

The MathWorks

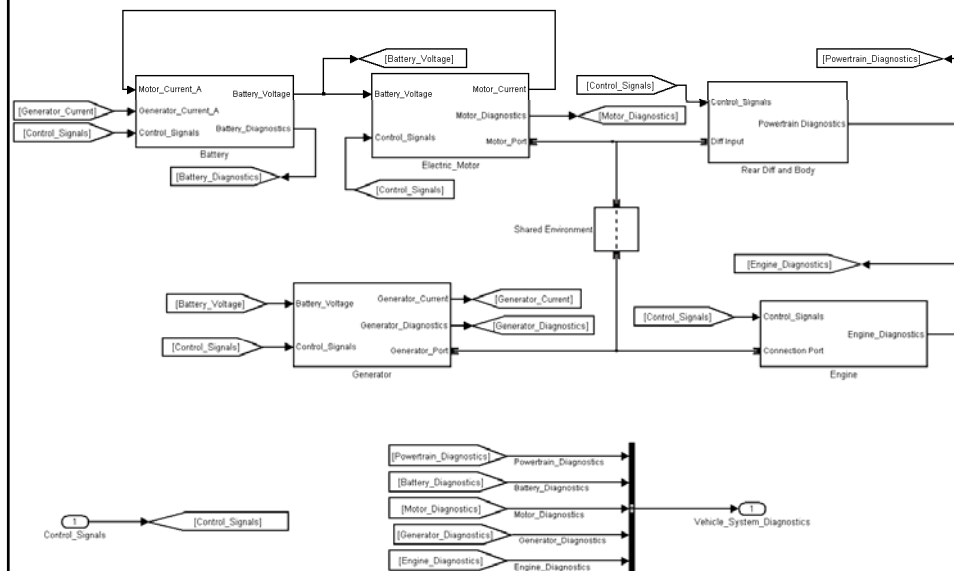
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Plant Model

22

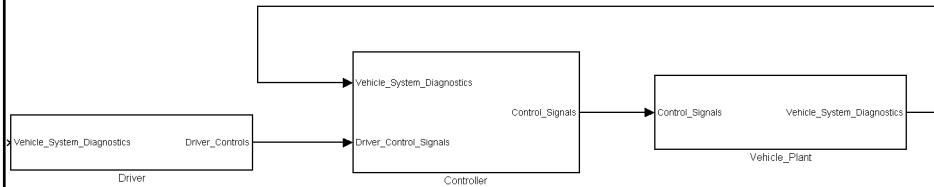
- We now have a pretty clean plant model:



Top Level

23

- We can now connect the system together at the top level:



- This now looks like a classical control system top-level block diagram:
 - The controller gets its commands from the driver block and then issues commands to the plant.
 - The controller monitors the plant output and modifies its output so that the plant achieves certain performance criteria.

MotoTron

The MathWorks

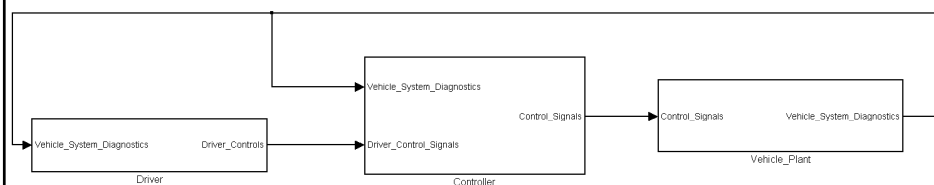
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Top Level

24

- For the moment, the Driver block needs the vehicle speed as well, and this signal is contained within the Vehicle_Systems_Diagnostics bus, so we need to connect this bus to the driver block as well:



MotoTron

The MathWorks

freescale
semiconductor

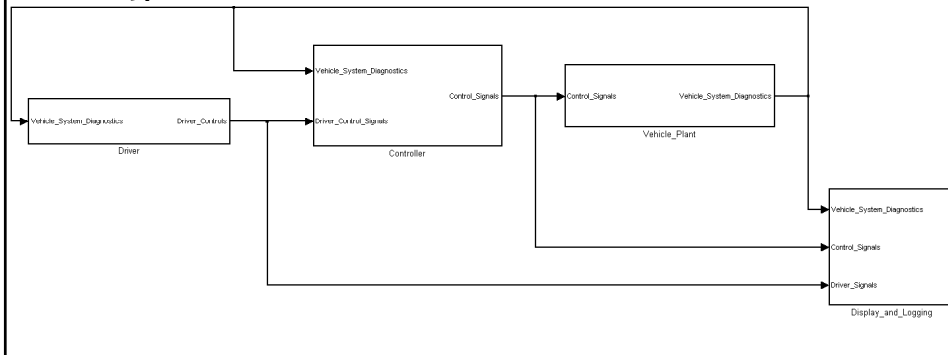
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Top Level

25

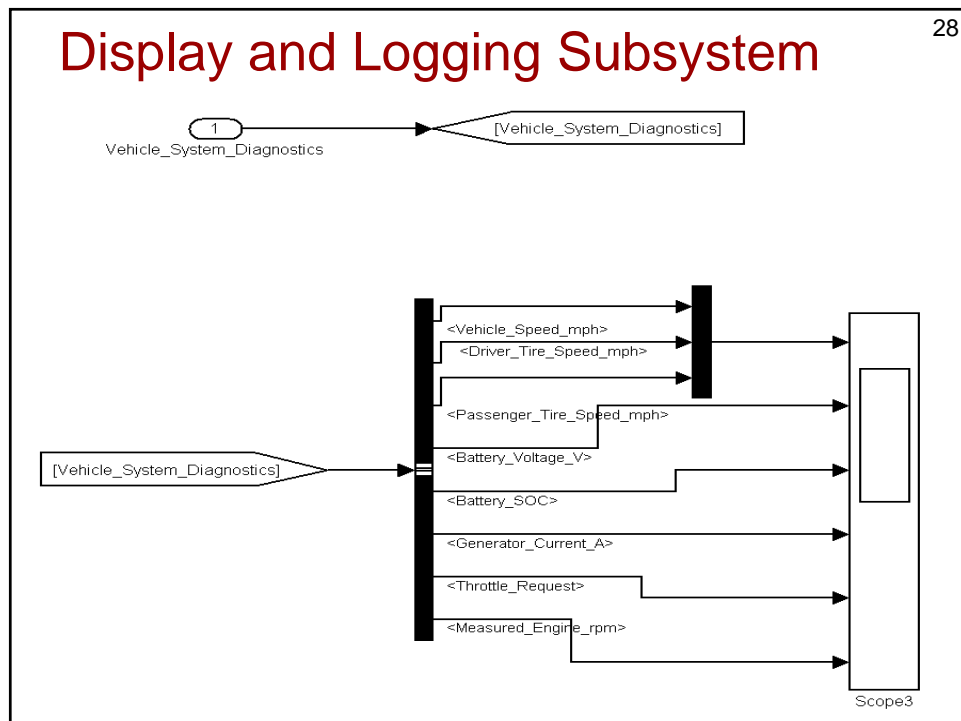
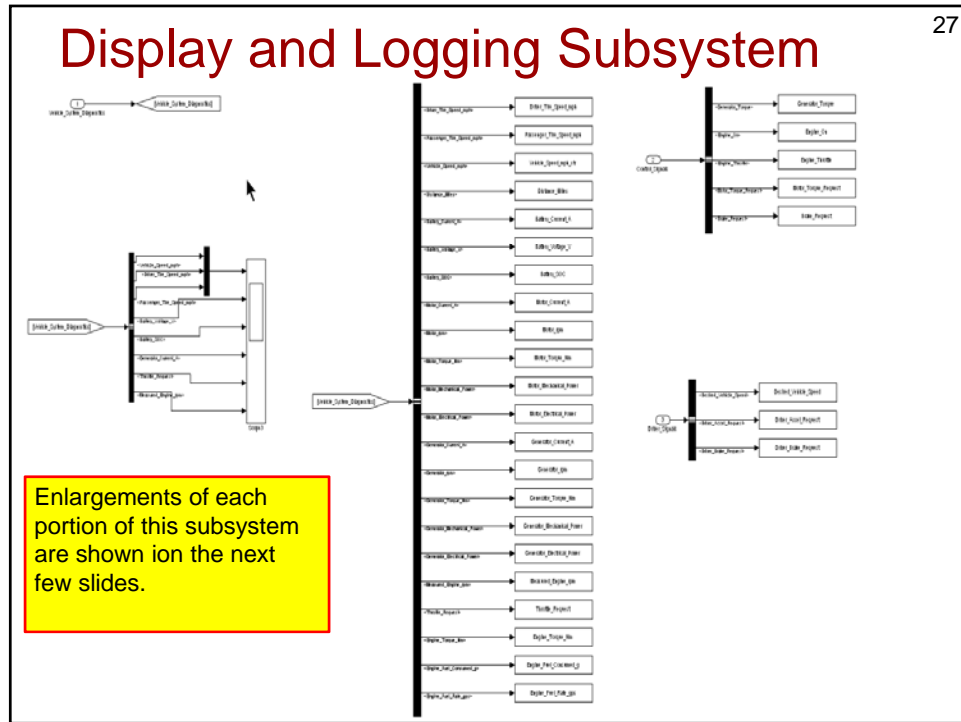
- The last thing we need to do is modify the Display_and_Logging subsystem.
- We will place this subsystem at the top level so that we can display and log signals in all three busses.



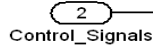
Display and Logging Subsystem

26

- This system now has three bus inputs.
- We will log every signal in all three busses.
- You will need to rearrange some of the signals from one bus to another as signals that were originally in the Vehicle_System_Diagnostics bus have been moved to one of the other busses.



29



30



You cannot see any of the signals in this slide. This portion displays all of the signals in the bus.

System Test

31

- We will now run a test to see that we connected everything correctly.
- We have not changed the model physically, so the results should be the same as in Lecture 10.
- If there are differences, it is because we made a mistake in connecting signals.

MotoTron

 **The MathWorks**

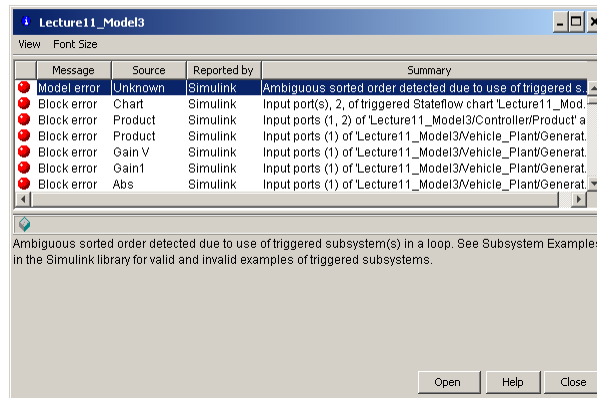
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

System Test

32

- When we run a simulation, we get an error that says that we have an algebraic loop!



MotoTron


 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

System Test

33

- Upon further investigation, we find that the algebraic loop includes the SimDriveline Env, and clutch, and tire models.
- It turns out that using the clutch can cause an algebraic loop.
- There are three ways to fix this:
 - **First Method:** Do not use a clutch. The only place we are using a clutch is for the brakes. We can use the brake model that used the torque actuator rather than the clutch.
 - **Method 2:** Add a memory block  somewhere in the mode to break the algebraic loop.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

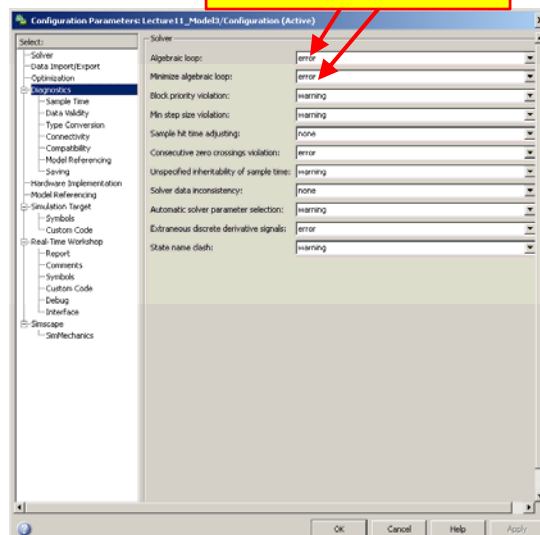
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Clutch Algebraic Loop

34

- In the **Simulation / Configuration Parameters** dialog box we can change the diagnostics for Algebraic loops from error to warning. (This did not work in this case...)

Change these to items to "warning."



System Test

35

- Pick the method you would like to use and run a simulation and verify that it produces the same results for the FU505 as in lecture 10.

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

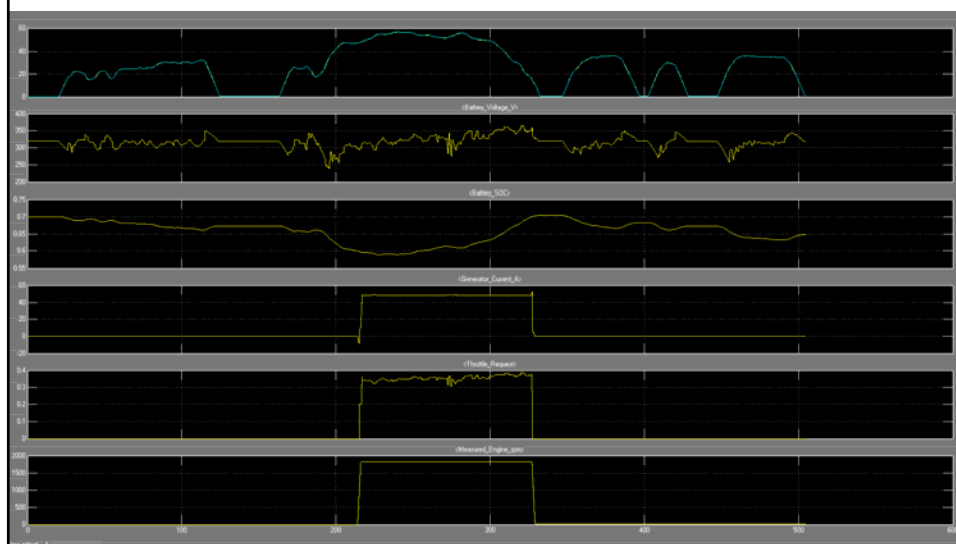
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Lecture 11 Demo 1

Demo_____

36

- Demo the working model of the rearranged system.





Manual Controls

37

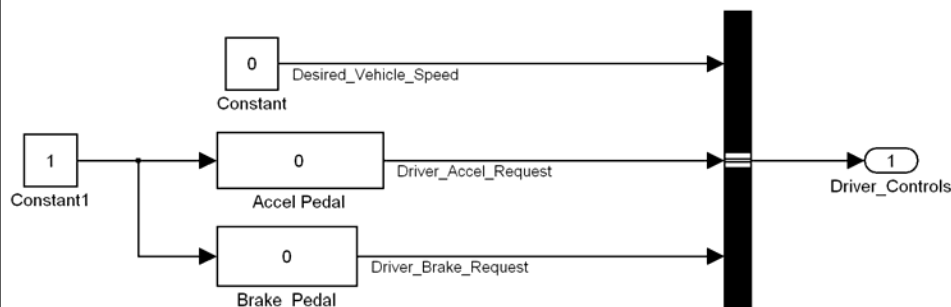
- Next, we would like to replace the driver block with manual controls.
- Eventually, we will drive the vehicle with these controls in real-time.
- For now, we will just add acceleration and braking signals.
- Later, we will add signals for turning on the vehicle and the gear shift.



Driver Block

38

- We will replace the contents of the driver block with the manual controls shown below:





Driver Block

39

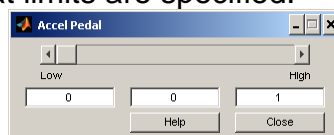
- We did not delete the Desired_Vehicle_Speed signal because it would require extra work on our part to search through the model and locate all instances of where this signal is used.
- Instead, we set the driver speed to a constant of 0 so that we can continue to use all of our plots and post processing files without modification.
- The new blocks used in the driver subsystem are **Slider Gain** blocks located in the **Simulink / Math Operations** library.



Slider Gain

40

- If you double-click on one of the slider gain blocks, you will notice that limits are specified:



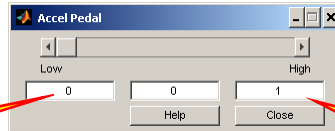
- This part is a gain block that is controlled by the slider shown above. The limits on the slider are specified in the block.
- This is a gain block. The output is the input times the gain, where the gain is determined by the slider.
- Note that the slider can be changed during a simulation, allowing you to change the gain on-the-fly.





Slider Gain

41



Lower limit of slider gain.

Upper limit of slider gain.

- Since the limits on the slider gain are 0 and 1, and the input to the slider is 1, the output of the slider varies between 0 and 1 as we move the slider.
- We will use these sliders as the accelerator and brake pedals for driving our vehicle.

MotoTron

The MathWorks

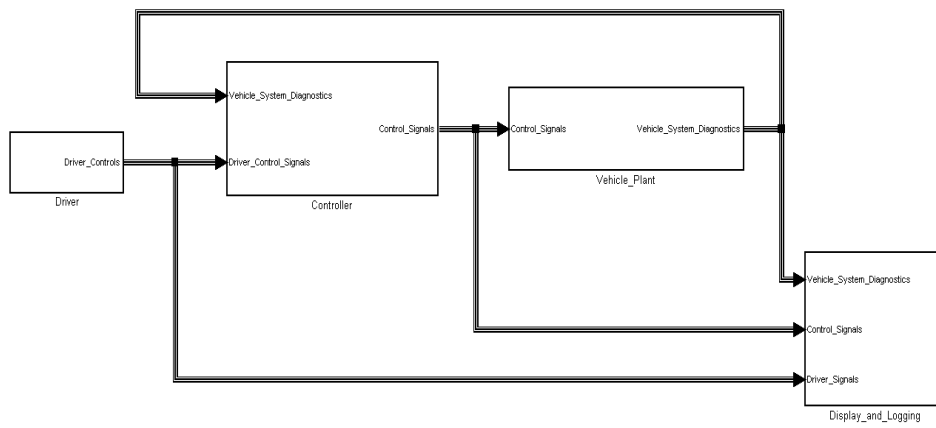
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Driver Block

42

- Note that the driver block no longer needs the vehicle speed information, so we can delete the Vehicle_Systems_Diagnostics bus:





Accel and Brake Pedals

43

- We would like to add some logic to protect the driver from himself and protect the vehicle from a rogue driver.
- We will add some logic that prevents the accelerator pedal and brake pedal from being depressed at the same time.



Accel and Brake Pedals

44

- We will implement the following rules:
 - If the brake pedal is depressed, the accelerator pedal signal is set to zero, even if the accelerator pedal is being depressed. (We will always brake and disable the accelerator pedal whenever the brake pedal is depressed.)
 - If the brake pedal is not depressed, the value of the accelerator pedal is passed to the system.



Controller Modifications

45

- The accelerator and brake signals are passed directly to the Acceleration_and_Braking subsystem that is contained within the controller.
- We will add a signal conditioning block where the signals first enter the Acceleration_and_Braking Subsystem:

MotoTron

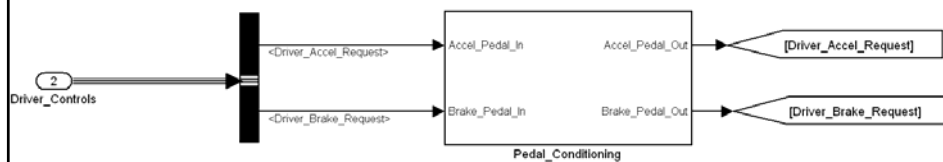
The MathWorks

freescale
semiconductor

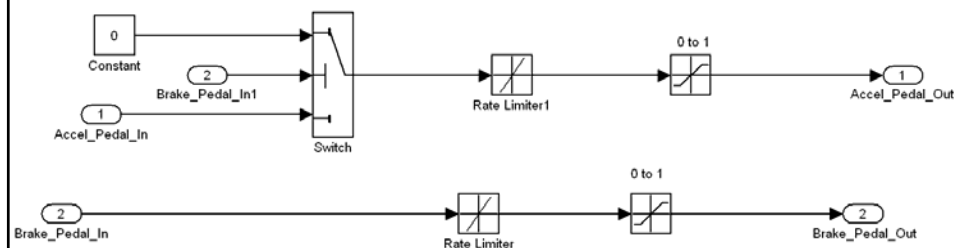
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Acceleration and Braking Subsystem

46



- The contents of the Pedal_Conditioning subsystem are





Acceleration and Braking Subsystem

47

- We see that the Brake_Pedal signal is basically passed to the system with little modifications.
- Saturation blocks are used to limit the signals from 0 to 1 in case we make a mistake and put in too large of a signal.
- The Slew rate of the limiter is set to +/-10 per second to eliminate very sharp braking and acceleration requests.
- The threshold of the switch is set to 0.05. The switch sets the acceleration pedal signal to zero if the Brake pedal is depressed more than 5% (the 0.05 threshold).

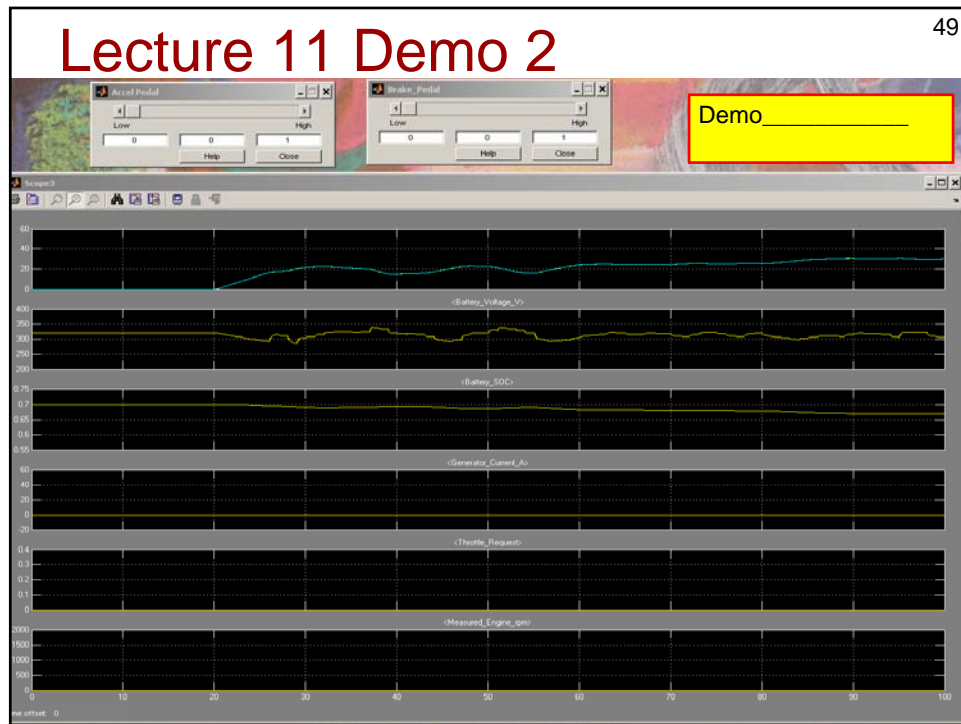


Test Drive

48

- We are now ready to drive the vehicle with the manual controls.
 - Set the simulation time to inf.
 - Set the simulation to “Normal.” (Not the accelerator or rapid accelerator.
 - In the scope plot you which to use, se the Tiome range to 100.
- Set up your windows as shown next and drive the vehicle and verify the logic of the accel and brake pedals:







Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>



Advanced Model-Based-System Design

Lecture 12: System Initialization Shifting Logic



System Initialization



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- We now have a vehicle that we can drive with manual accelerator and brake pedals.
- With most vehicles, you cannot just hump in and press the accelerator pedal and drive away. The following usually occurs
 - You turn the key to start the vehicle.
 - The vehicle goes through a component check.
 - The vehicle systems are enabled.
 - You must then shift the vehicle out of park into forward or reverse.

System Initialization



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- We will add a parallel Stateflow chart to go through the vehicle startup and shifting procedures.
- This Stateflow chart will enable the charge-control Stateflow chart that controls the engine-generator charging system.
- We will start with model Lecture12_Model0, which will be passed out in class.

System Initialization



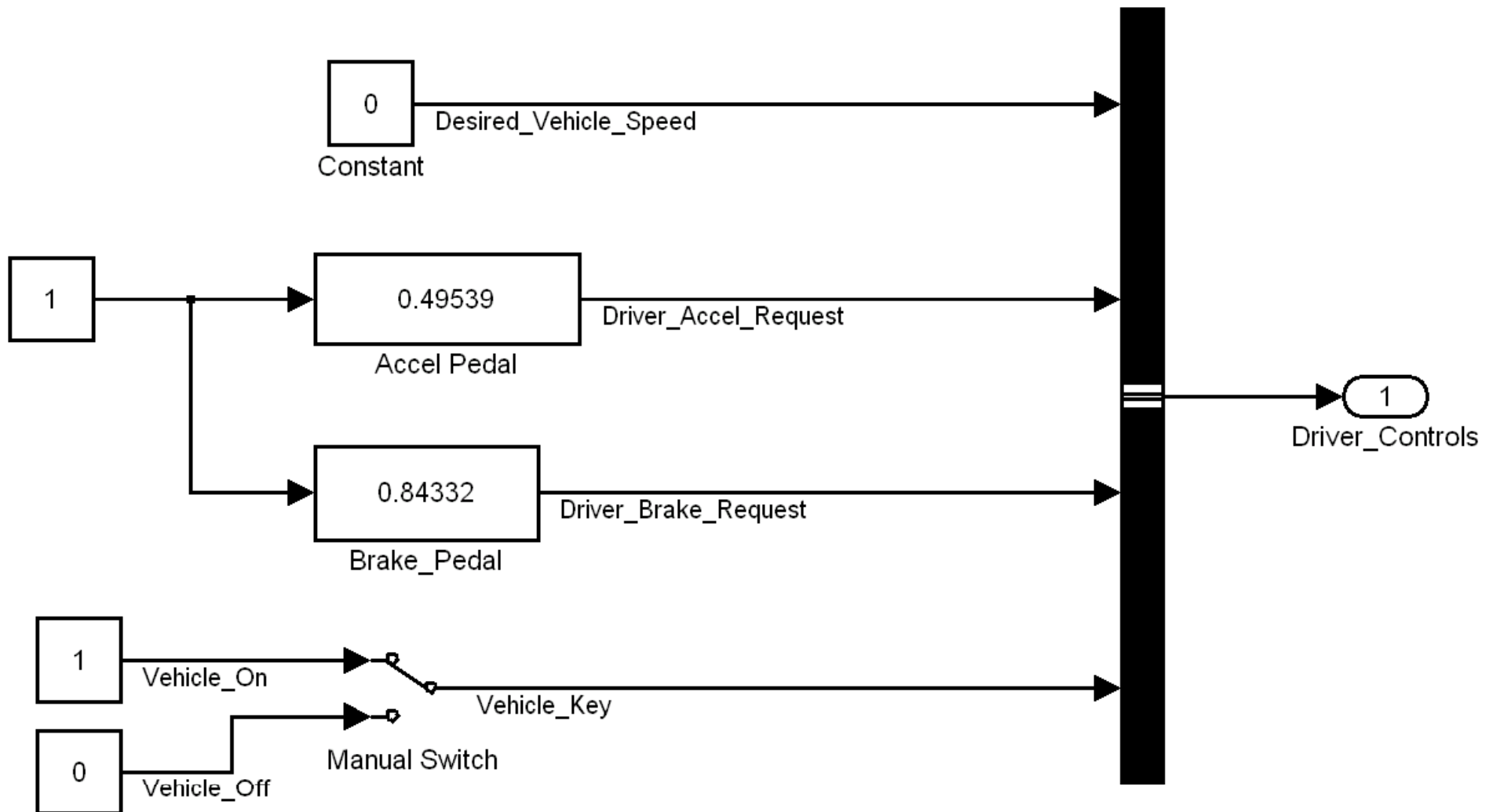
Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- First, we will add a switch to the driver block that simulates the key switch of a conventional vehicle.
- Our vehicle does not have a starter, so all we need is an off-on switch.
- We will use a manual switch to switch the signal between 0 and 1.
- We will add this to the Driver_Controls bus, and this signal will go directly to the controller.
- We will Name the signal Vehicle_Key.

Driver Subsystem



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



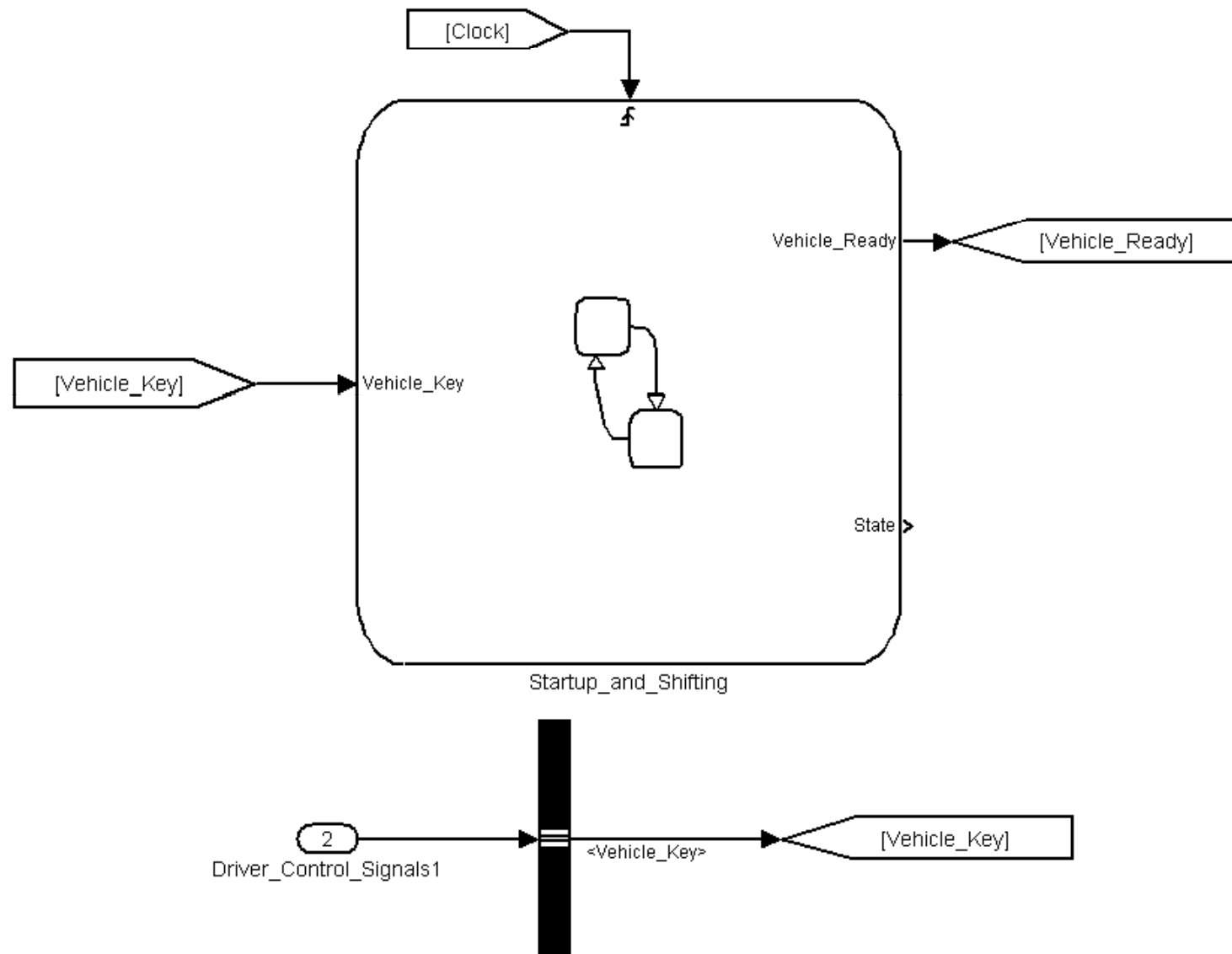
Controller   Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Next, we will add a second Stateflow chart.
- The only inputs to this chart will be the Vehicle_Key signal and a clock, which is the same clock as used for the charge controller.
- This chart has a single output, which is the Vehicle_Ready signal.
- The value of this signal is initialized to zero and will remain zero until we check the status of the battery, motor, generator, and engine.
- We will also add a variable called State for debugging purposes.
- Add a Stateflow chart as shown:

Controller Modifications



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>



Controller



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

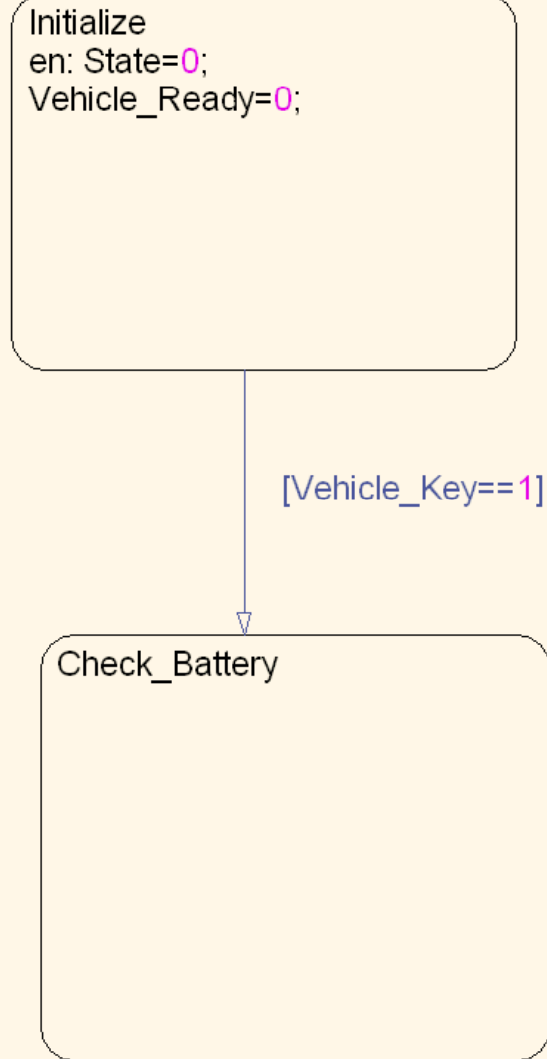
- The contents of the chart only check to see the status of the Vehicle_Key switch.
- Note that, for the moment, the vehicle_Key switch only tells the vehicle to turn on. It is not capable of turning the vehicle off.
- (We will implement this later because we need to do a controlled shut-down procedure.)
- The beginning of the Startup_and_Shifting chart are shown next:

Controller



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

There is an error in this diagram. Something was left out. You may find it later...



Controller

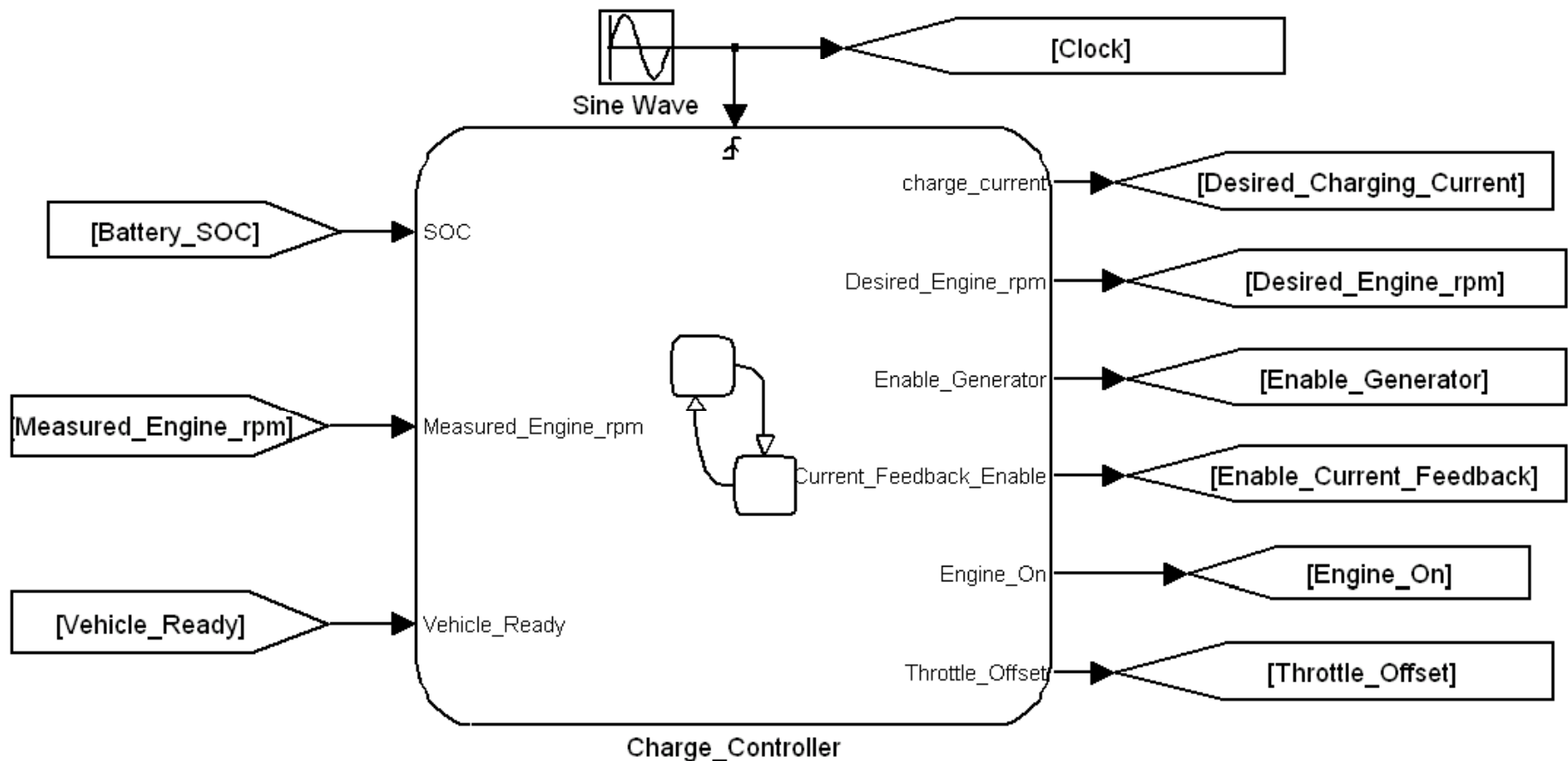


Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The Charge_Controller Stateflow chart must also be modified.
- We want to specify that the engine and generator not be enabled until the Startup_and_Shifting chart check out all of the components and signals that the vehicle is ready.
- We will need to add the Vehicle_Ready signal as an input to this chart.

Controller  Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Modify the Charge_Controller chart as shown:



Charge_Controller

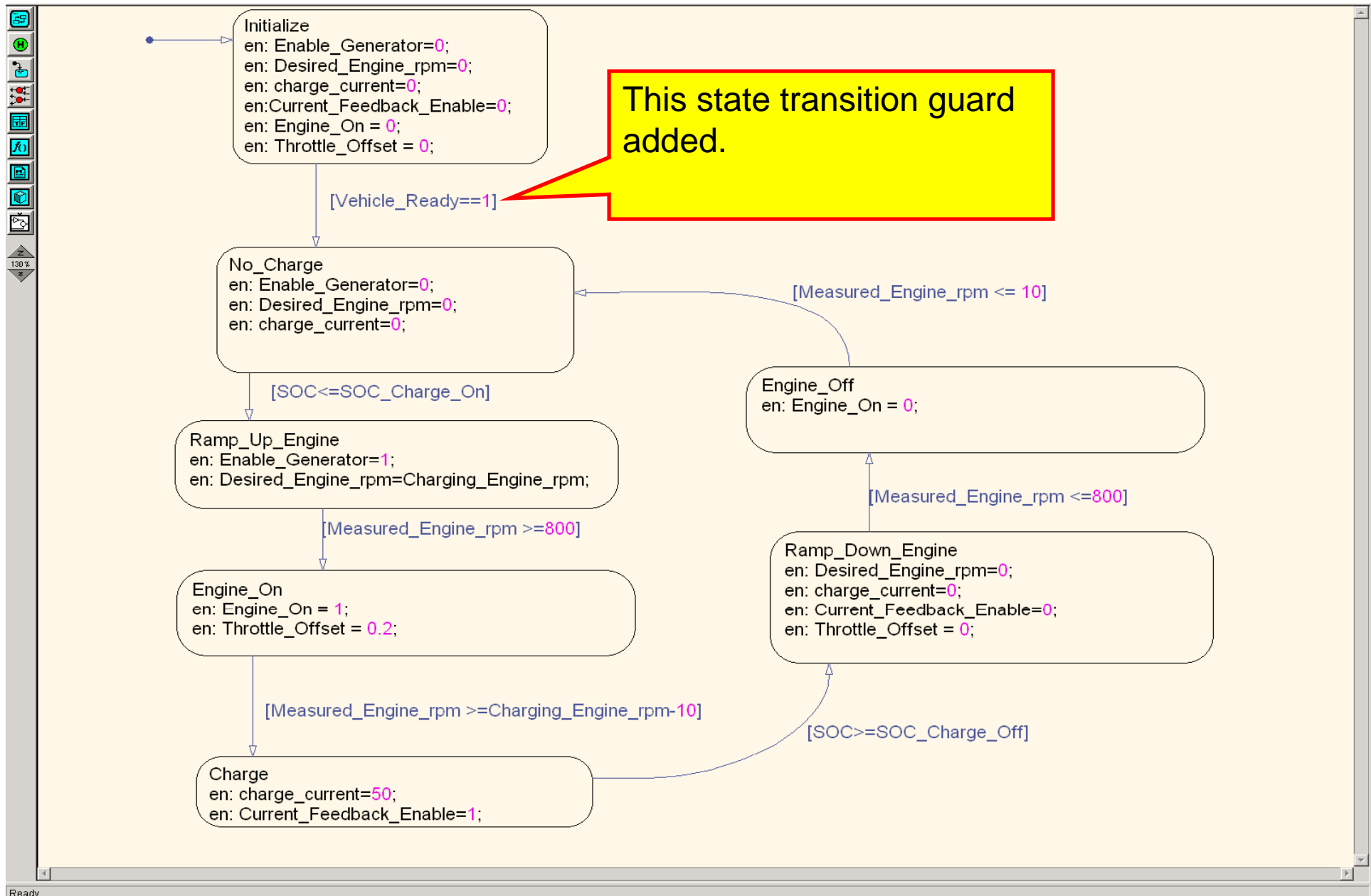


Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- We will add an exit condition to the initialization state of the Charge-Controller Stateflow chart that will not allow the vehicle to enter the normal charge/discharge cycle until the vehicle is ready.
- Note that this modification does not address the issue of a graceful shutdown.

Charge Controller

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Battery Model



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- Next we will add a startup handshake for the battery and some operational details.
- The battery has a control signal that must be sent to it and a status signal that it sends to indicate its state.
- The battery has an internal contactor. When the contactor is open, the battery voltage is zero and the pack is disconnected from the system.
- When the contactor is closed, the battery is connected to the system (motor and generator in our case) and the output voltage is that indicated by our model.

Battery Model



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- Connect Command – Input signal received by the battery
 - 1 – Close the contactor.
 - 0 – Open Contactor.
- Pack State – output signal sent by the Battery
 - 0 – Unavailable
 - 1 – Idle
 - 2 – Disconnected (Contactor Open)
 - 3 – Precharging
 - 4 – Connected (Contactor Closed).

Battery Model

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- The battery model needs to be modified as follows
 - The Connect Command must be added as part of the Control_Signals bus.
 - A switch is added to the model so that the battery voltage is switched between zero and the model voltage depending on the value of signal “Connect Command.”
 - When the battery receives the a connect command of 1, after a 3 second delay, the contactor switch will close and the battery voltage will be equal to the model voltage.
 - If the connect command goes to zero, the contactor should open immediately.
- A diagnostic output should be added that contains the Pack state signal. The pack state signal should have the following values depending on the state of the battery model:
 - 2 – The contactor is open, the battery is disconnected, and the battery voltage is zero.
 - 3 – During the three second delay when the battery is connecting.
 - 4 – The contactor is closed. The battery is connected, and the voltage is equal to the modeled voltage.

Battery Model



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The Startup_and_Shifting Controller must be modified as follows:
 - After Vehicle_Key = 1, the controller must issue the connect command.
 - Before proceeding to the next state, the controller waits for the Pack state to Equal 4.
 - If the pack state does not go to 4, do not proceed and do not allow the driver to use the vehicle. (Indicate an error and hold in an error state.)
 - If the pack state changes to 4, allow the controller to proceed to the next state.

Lecture 12 — Exercise 1

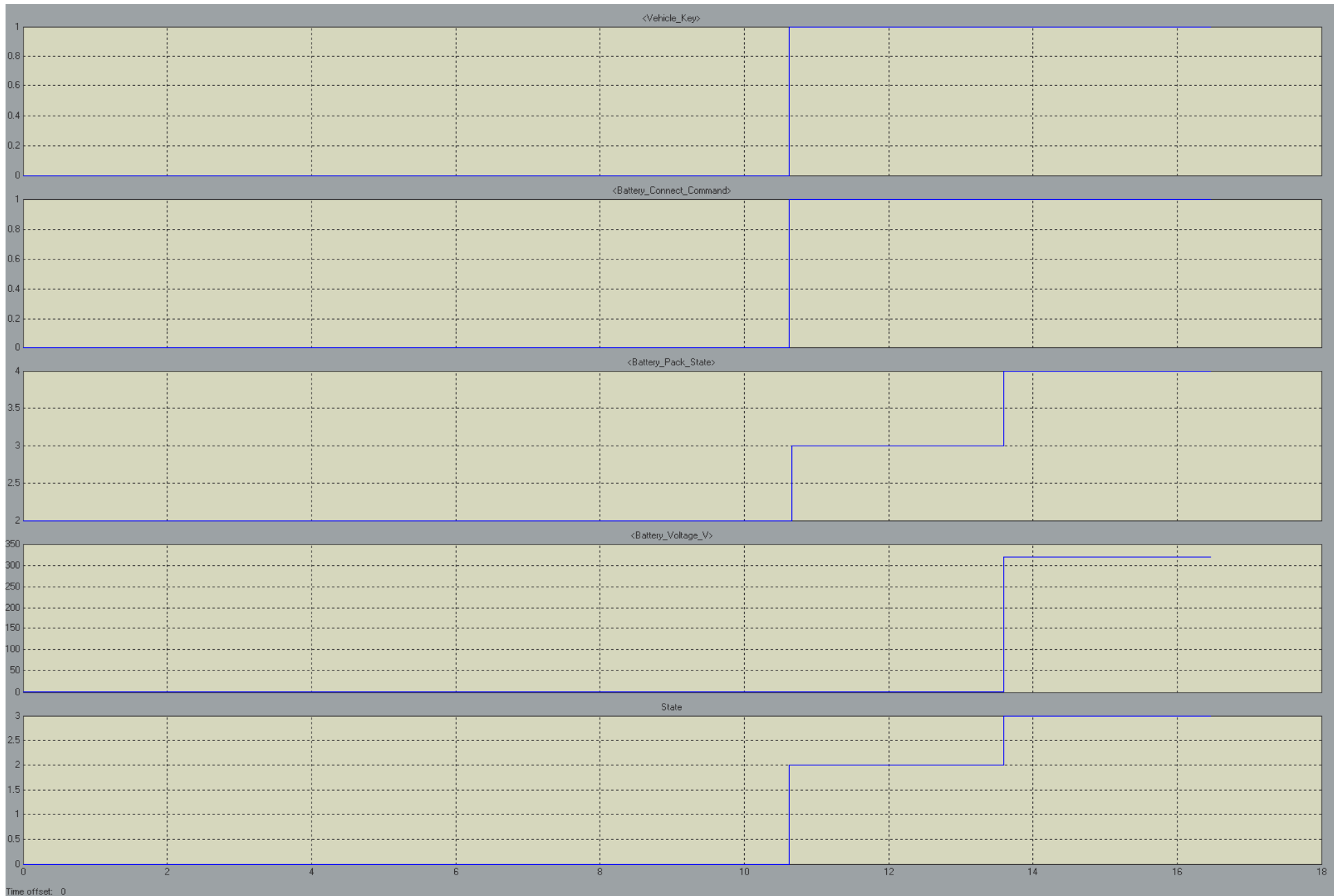
Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Demonstrate the operation of the Vehicle Key Switch and the Battery Connect command. Show the following:
 - Startup_and_Shifting chart as it walks through the startup procedure.
 - The battery voltage showing the operation of the contactor.
 - The battery status signal showing the various states.
 - Show a plot similar to the one shown on the next two slides.

Demo_____

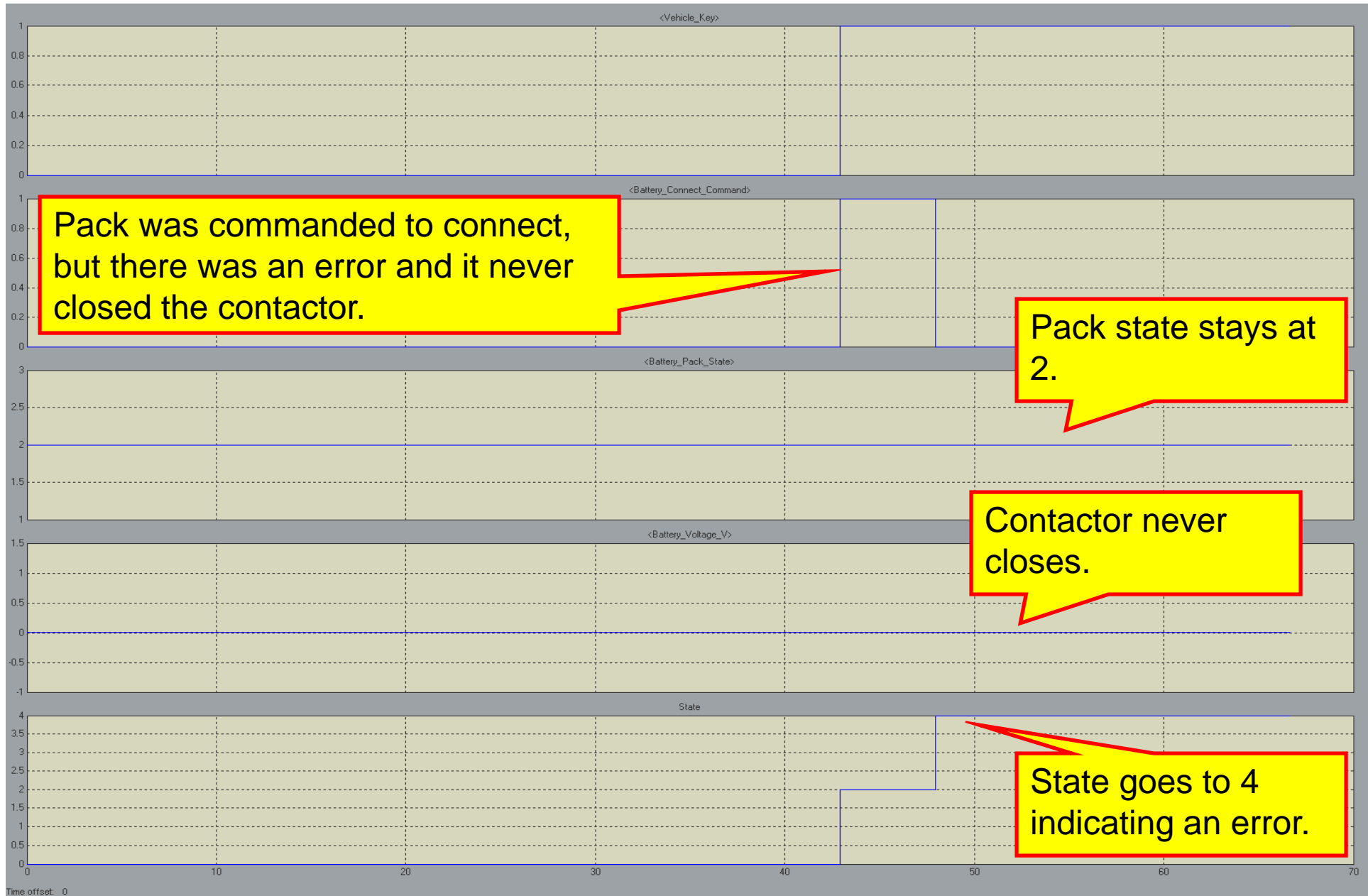
Lecture 12 — Exercise 1

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Battery Model — Exercise 1

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Motor and Generator Models



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>.

- Next, we need to add the handshake and status signals for the Motor and generator models.
- The Motor and Generator models have a control signal that must be sent to it and a status signal that it sends to indicate its state.
- Enable Command – Received by Motor or Generator
 - 1 – Enable Component.
 - 0 – Disable Component.
- Motor/Generator State – Sent by Motor or Generator
 - 0 – Motor/generator Not Ready
 - 1 – Motor/generator Ready

Motor and Generator Models



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>.

- The Motor and Generator models need to be modified as follows
 - The Enable Command must be added as part of the vehicle Control_Signals bus.
 - The motor/ or generator will only be ready after it receives the enable command and the battery voltage is greater than 200 V.
 - The motor or generator can produce no torque if they are not enabled or the battery voltage is below 200 V.
 - A diagnostic output should be added for the motor or generator state signal. This signal should have the following values:
 - 1 – Component OK and enabled.
 - 0 – The component is disabled. This occurs if the enable signal goes to zero or the battery voltage drops below 200 V.

Controller Modifications



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The start-up sequence must be modified:
 - After the battery has passed its start-up test the enable command should be sent to the motor. The controller then waits to receive the status of the motor.
 - If the status is 1, proceed to the generator check.
 - If the status does not change to 1 after 1.5 seconds, enter an error state and hold.

Controller Modifications



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The start-up sequence must be modified:
 - After the motor has passed its start-up test the enable command should be sent to the generator. The controller then waits to receive the status of the generator.
 - If the status is 1, set Vehicle_Ready to 1 and go to the Park State (for now do nothing in this state except set Vehicle_Ready to 1).
 - If the status does not change to 1 after 1.5 seconds, enter an error state and hold.
- You might want to use the After command in Stateflow.

Lecture 12 – Exercise 2



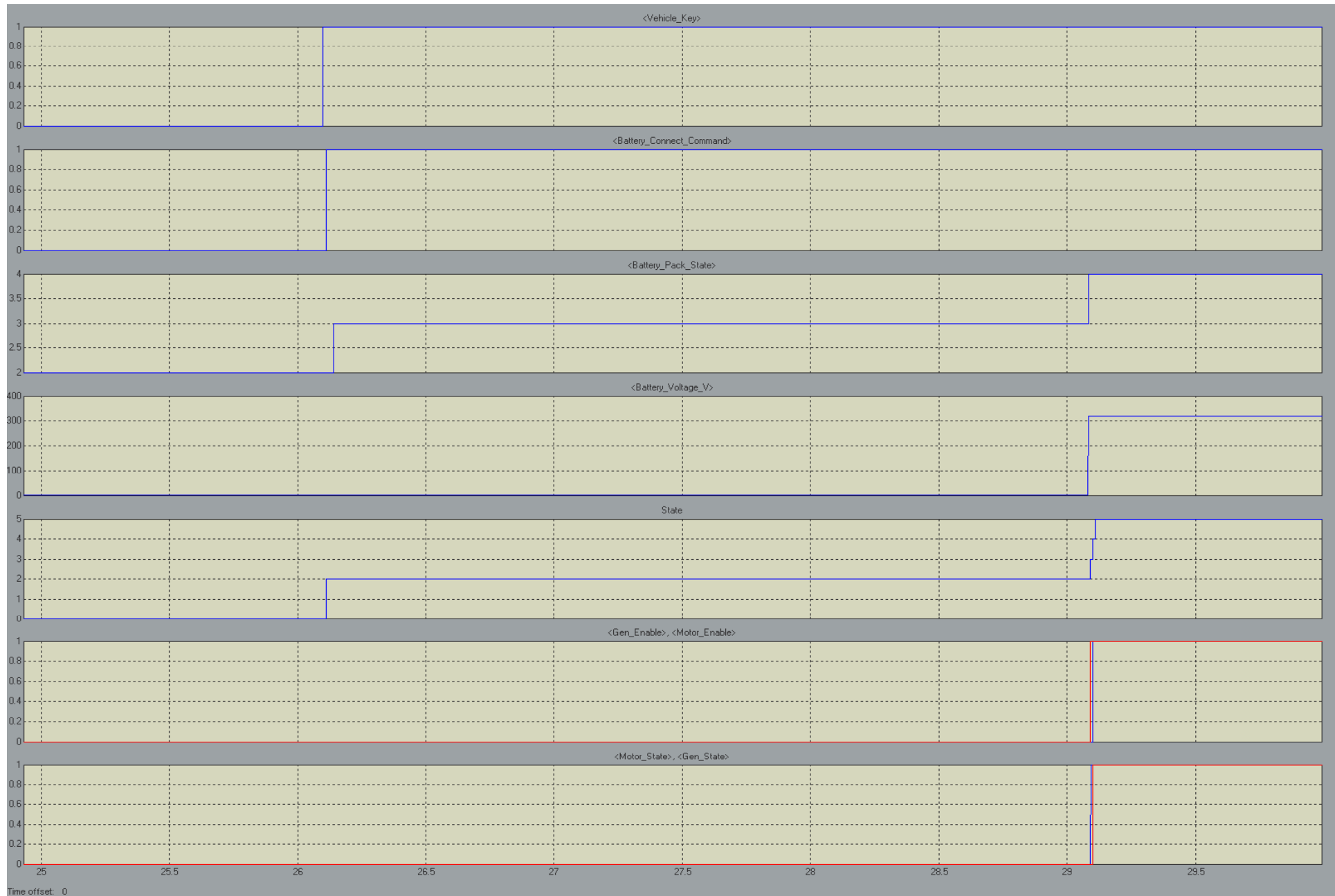
Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Demonstrate the operation of the Vehicle Key Switch, Battery Connect, Motor Enable, and Generator Enable Commands. Show the following:
 - Startup_and_Shifting chart as it walks through the startup procedure.
 - The status signals showing the various states of each component.
 - Show a plot similar to the one shown on the next two slides.
 - Plots are shown on the next two slides

Demo_____

Lecture 12 — Exercise 2

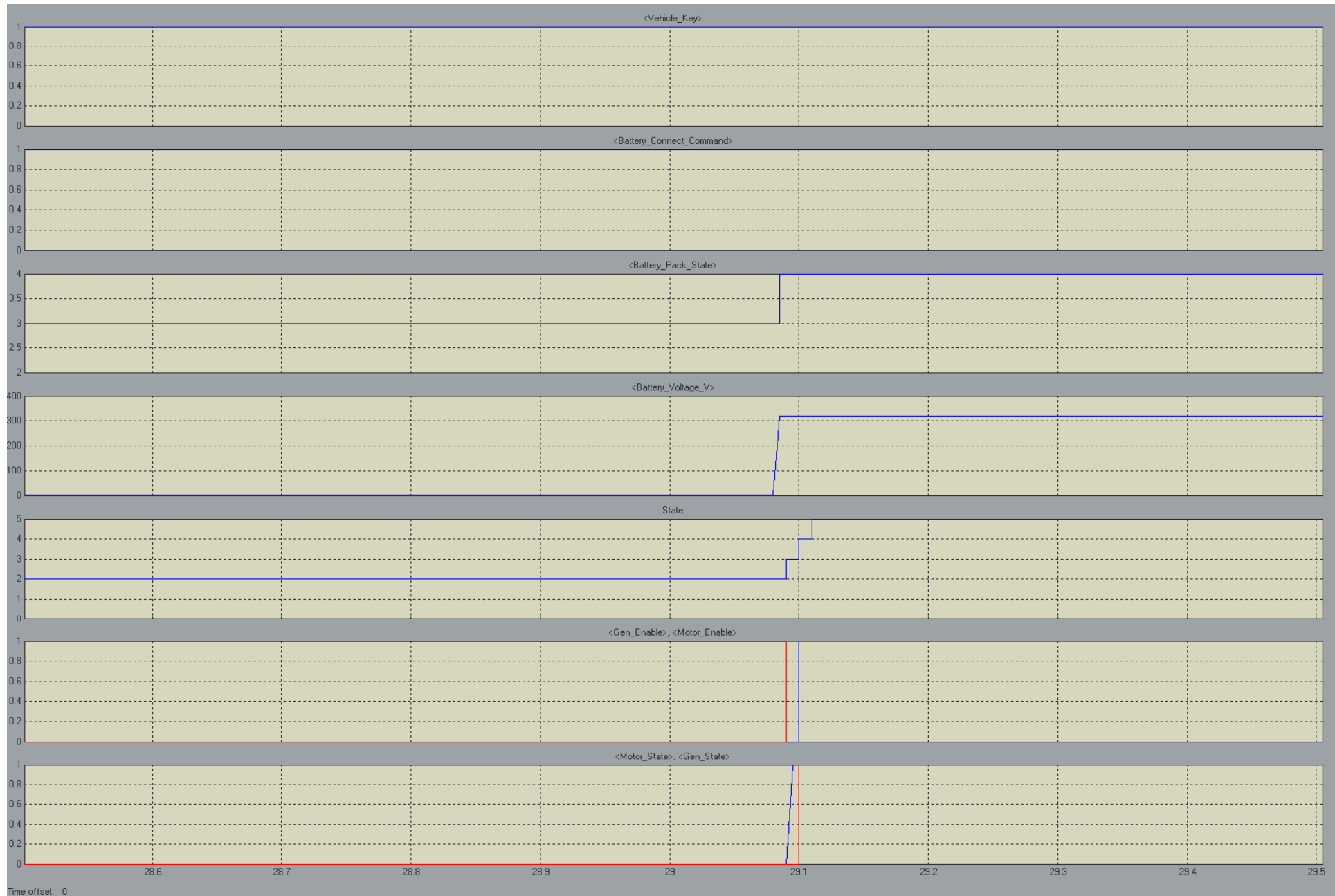
Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Lecture 12 — Exercise 2



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Lecture 12 Exercise 3

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- We now wish to add shifting logic to our model.
- We will need to add three pushbuttons for forward, reverse, and park. These switches go in the Driver block.
 - Park – When the driver presses this push-button, it indicates that the driver wants to have the transmission enter the park state. In this state, the accelerator pedal input is ignored.
 - Forward - When the driver presses this push-button, it indicates that the driver wants the vehicle to move forward when the accelerator pedal is pressed.
 - Reverse - When the driver presses this push-button, it indicates that the driver wants the vehicle to move backwards when the accelerator pedal is pressed.

Lecture 12 Exercise 3

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- We also need some dashboard indicator lights for the driver.
 - Park LED – This LED is illuminated when the vehicle is in Park.
 - Forward LED – This LED is illuminated when the vehicle is in Forward.
 - Reverse LED – This LED is illuminated when the vehicle is in Reverse.
 - Error LED – This LED illuminates to alert the driver if there is a problem.
 - Vehicle Ready LED – This light illuminates when the vehicle has passed all component checks.

Lecture 12 Exercise 3

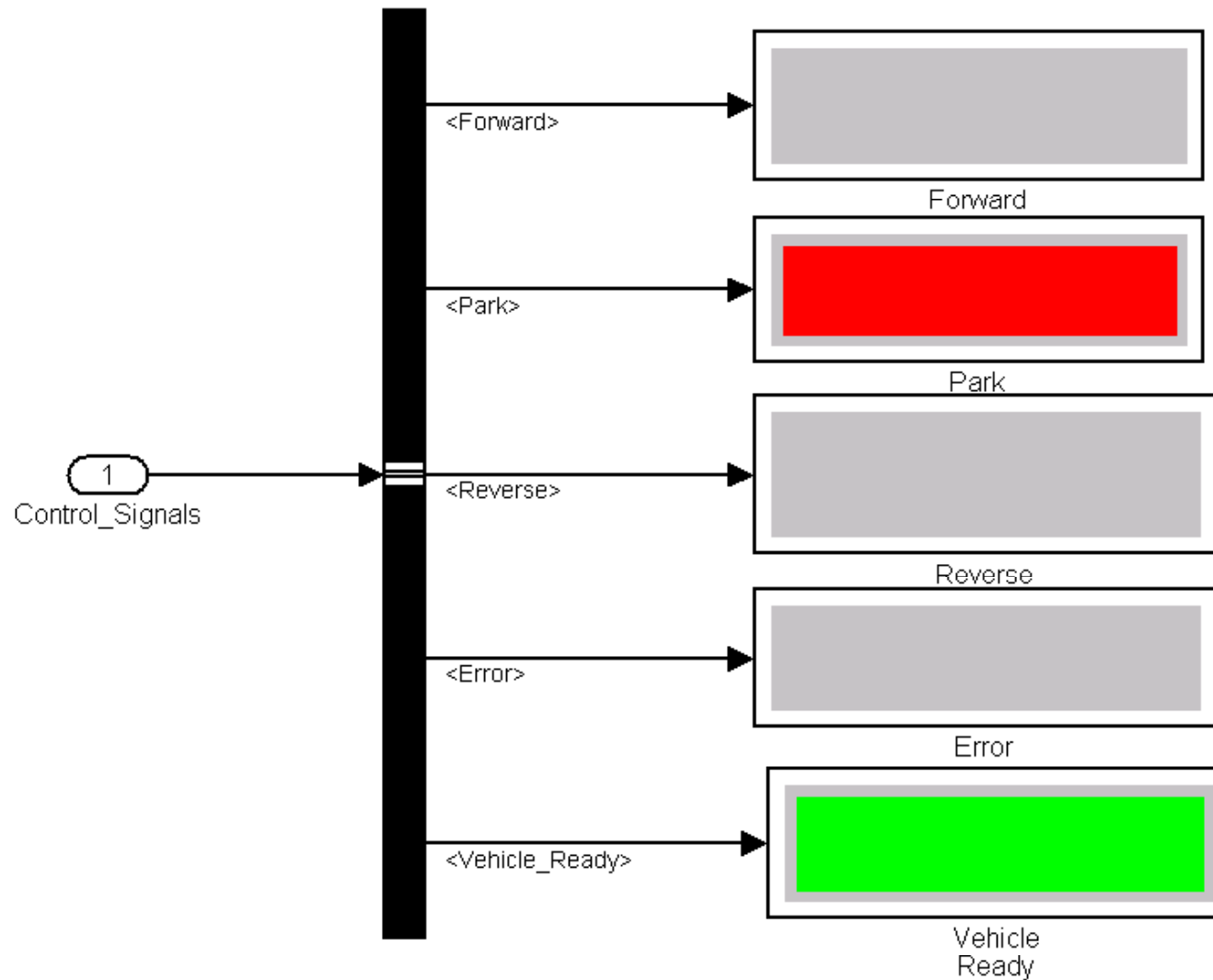


Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- Note that the controller determines when we are in park, forward or reverse.
- We will place these indicators in a new subsystem within the Vehicle_Plant called Dashboard.

Lecture 12 Exercise 3

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>





Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- All logic for the accelerator pedal, brake pedal and push-buttons resides in the controller. The controller emits signals for the traction motor torque and foundation brakes. The driver block just contains the driver input sensors.
- At vehicle startup, after checking that the status of each component is correct, the vehicle should enter the park state.
- To shift out of park into either forward or reverse, the following sequence must occur:
 - The driver must depress the brake pedal by 50% or more.
 - The driver can then press the forward or reverse buttons.
 - When the button is pressed, the appropriate indicator light is illuminated and the vehicle shifts into either forward or reverse.
 - If the vehicle is in park and the brake pedal is depressed less than 50% and the forward or reverse buttons are pressed, nothing happens and the vehicle remains in Park.

• Forward State

Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The controller takes the accelerator pedal and brake pedal signals and modifies them appropriately so that
 - The traction motor increases the vehicle speed in the forward direction when the accelerator pedal is depressed.
 - The traction motor and foundation brakes decrease speed in the forward direction when the brake pedal is depressed.
 - To exit the forward state, the following items must be true simultaneously:
 - The brake pedal must be depressed by more 50% or more.
 - The vehicle speed should be less than 1 mph.
 - The driver must press the Park or Reverse buttons.
 - If the Park, Reverse, or Forward buttons are pressed while the above conditions are not true, the buttons are ignored.

• Reverse State

Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The controller takes the accelerator pedal and brake pedal signals and modifies them appropriately so that
 - The traction motor increases the vehicle speed in the reverse direction when the accelerator pedal is depressed.
 - The traction motor and foundation brakes decrease speed in the reverse direction when the brake pedal is depressed.
 - To exit the reverse state, the following items must be true simultaneously:
 - The brake pedal must be depressed by more 50% or more.
 - The vehicle speed should be less than 1 mph.
 - The driver must press the Park or Forward buttons.
 - If the Park, Reverse, or Forward buttons are pressed while the above conditions are not true, the buttons are ignored.

Lecture 12 – Exercise 3



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- Demo of everything working.
- I will try to brake your model.

Demo_____





Introduction to Model-Based Systems Design

Lecture HIL1: Introduction to Hardware-in-the-Loop



HIL

2

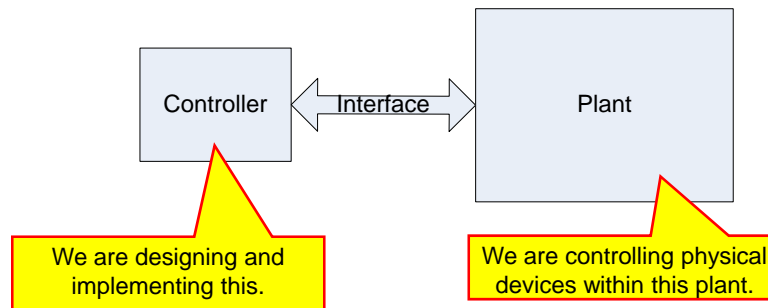
- HIL is a simulation method that allows us to test our controller, controller hardware, and the wiring interface on a virtual plant before testing the controller, controller hardware, and wiring interface on the real plant.
- This is yet another level of testing we can do before deploying our design in the real world.



HIL

3

- Remember that our goal is to design and implement a controller for real and complicated physical system.



HIL

4

- The plant is very complicated and difficult to understand, expensive to fix if we break something, and could be dangerous if controlled improperly.
- HIL allows us to test our controller on the target hardware we will be using, with the wiring interface we will be using.





HIL

5

- Instead of taking our controller and seeing if it works by connecting it to the actual plant to be controlled, we will connect it to a computer running a model of the plant.
- The model of the plant will have the same interface as the physical plant. → It will have the same physical inputs and outputs as the physical plant.



HIL

6

- So, instead of connecting our controller to the real plant, we will connect it to a computer running a model of the plant (a computer pretending to be the plant).
- We can then test our controller without worry of damaging the physical plant.





HIL

7

- HIL will allow us to test:
 - The control algorithm.
 - The controller hardware (the target computer on which the controller is deployed).
 - The control algorithm as it runs on the target hardware.
 - The physical interface between the Controller and the plant. (Assuming that the harness we use in the test is the same harness we will use when we connect to the physical plant.)



HIL

8

- Up to now we tested the control algorithm on:
 - A Windows system running on a PC - essentially using a continuous system with a time step that is variable and can become very small when needed, and floating point calculations.
 - Using xPC target – Using a fixed time step and possibly discrete control blocks, but using a more powerful computer than will be used for the target.





HIL

9

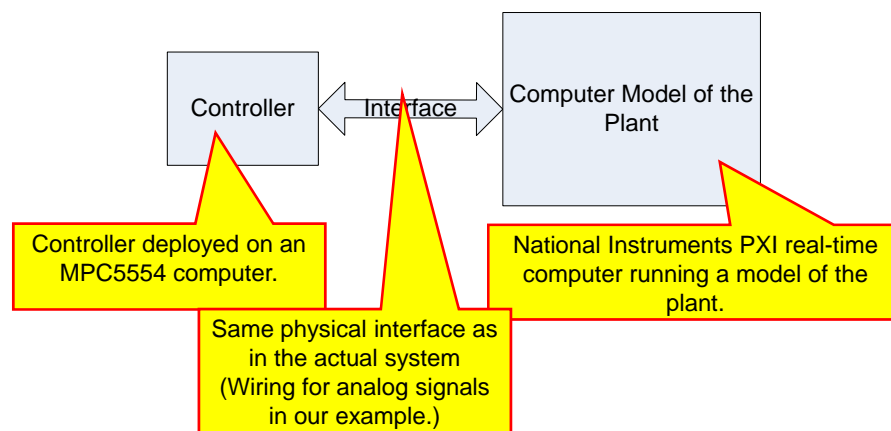
- HIL simulations allow us to test the control algorithm on the hardware that will be used to implement the controller.
- The benefit is that we can test the controller using a model of the plant so that there is no danger of personal injury or physical damage to the plant.



HIL

10

- We will be using the test platform below:





HIL

11

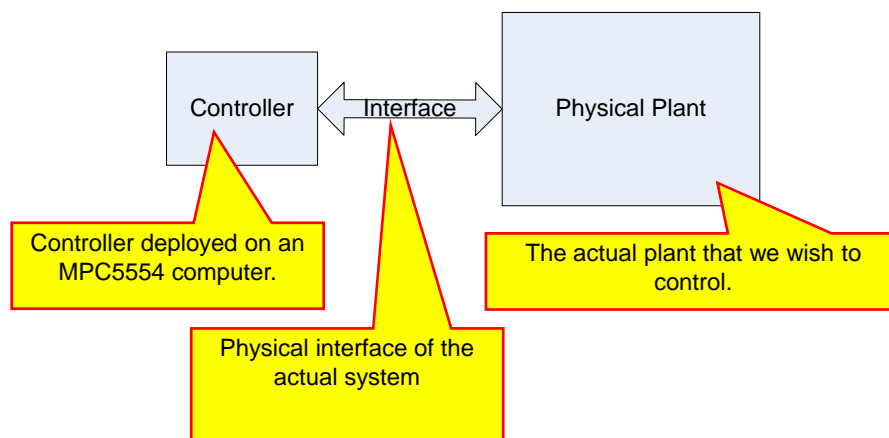
- Once controller testing is complete and the controller passes all tests and satisfied all requirements, we just plug the controller into the physical plant.
- If the models are accurate and the interface is the same, the controller should be able to control the physical plant as well as it controlled the model of the plant.



HIL

12

- Completed System:





HIL

13

- We will deploy the controller on an MPC5554 computer and the plant on a National Instruments PXI real-time computer.
- This requires us to learn National Instruments hardware and software.
- We will break the process into two steps.
- First, we will simulate the entire model in real-time on a National Instruments PXI computer.



HIL

14

- This model will be similar to what we did with xPC, except we will use National Instruments tools.
- We will learn how to use:
 - National Instruments Simulation Interface toolkit.
 - National Instruments LabVIEW
 - National Instruments Measurement and Automation Explorer (MAX)





HIL

15

- In this step, we will not split apart the plant and controller. Both will remain in the same model and be simulated on the same real-time target (National Instruments PXI Computer).
- This will allow us to learn the National Instruments tools and how to use MathWorks models with hardware and software from other vendors.



HIL

16

- In the Second step, we will split the controller and plant into separate models.
- The Plant will run on a National Instruments PXI target.
- The controller will run a Freescale MPC5554 target.
- The Plant will run on a National Instruments PXI target.
- We will connect the two targets with a wiring harness and some interface electronics (a low pass filter).



HIL

17

- We will use the RAppID toolbox and associated Freescale tools to deploy the controller on the Freescale target.
- We will use the Simulation Interface Toolkit and associated National Instruments tools to deploy the plant on the National Instruments target.
- The plant and controller models are mature and have been developed with MathWorks tools.



HIL

18

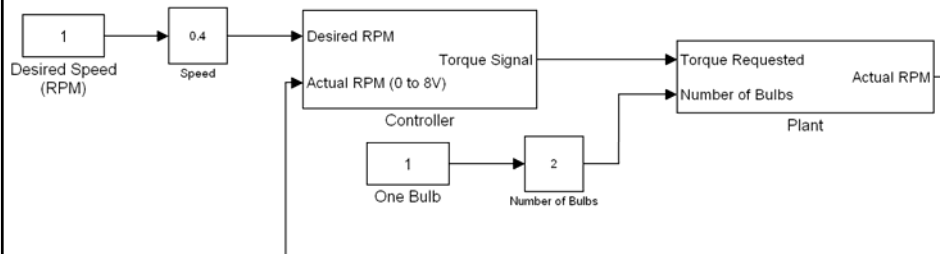
- To learn the National Instruments tools We will deploy a small model developed in a previous class on a PXI Target.
- Create a new folder called Lecture13.
- Copy file Lecture13_Model0.mdl into this directory and rename the model Lecture13_Model1.mdl.
- Copy the init file as well.



Stand-Alone Model

19

- This model was created to run on an xPC target, so we need to make a few changes.
- The starting model is shown below:



MotoTron

The MathWorks

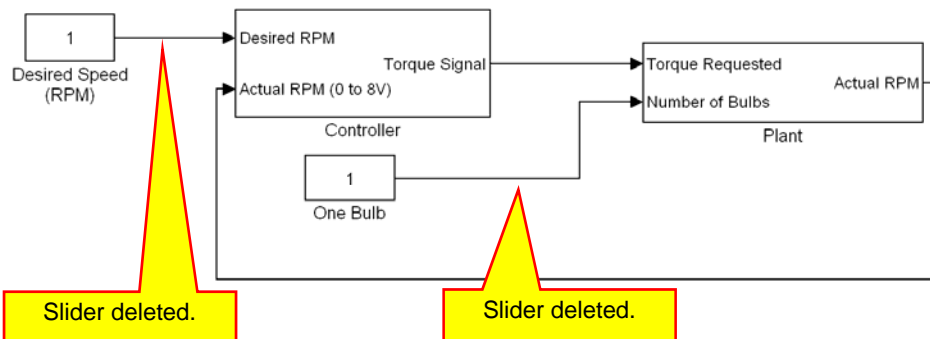
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Model Modifications

20

- The controls for the desired speed and number of bulbs will be front panel LabVIEW controls, so we can eliminate the Simulink sliders for Speed and Number of Bulbs:



MotoTron

The MathWorks

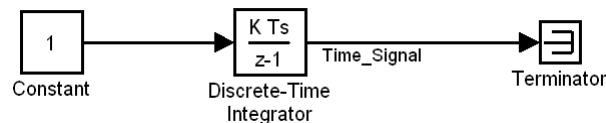
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Top-Level Modifications

21

- We would like to display the real time in our front panel. To do this, we will create an integrator at the top level of our model and integrate a constant (equal to 1).
- The value of the integral will be the elapsed time since the simulation started.
- Add the blocks below to your top-level model:



MotoTron

The MathWorks

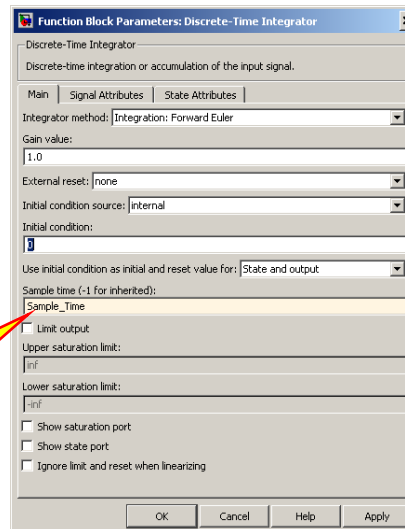
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Top-Level Modifications

22

- The sample time of the integrator should be set to the fixed step size specified in the simulation configuration parameters.
- This was a value that we specified in the init file. Set the parameters for the integrator as shown:

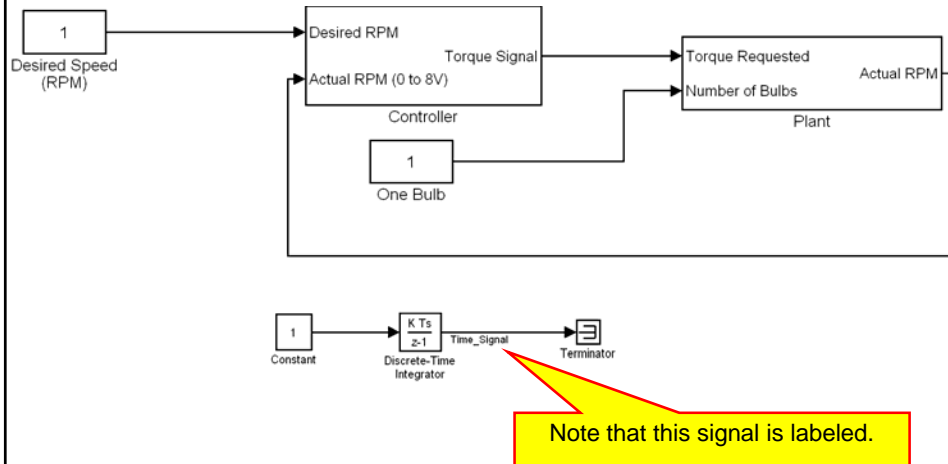


Sample_Time specified here.

Top-Level Model

23

- Your top-level Model should be as shown:



MotoTron

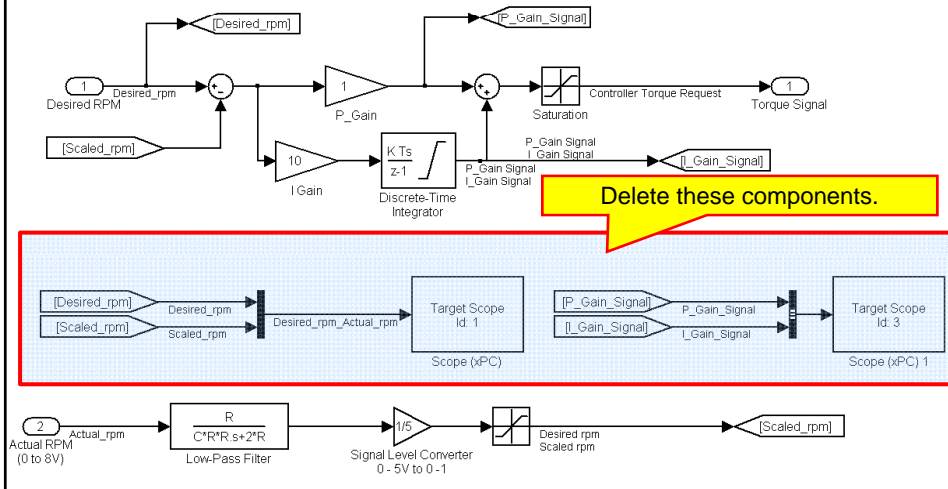
The MathWorks

freescale
semiconductorROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller Modifications

24

- The controller contains a number of blocks that were added to display information for xPC. These blocks are no longer needed and can be deleted:



Controller Modifications

25

- To make connections between the LabVIEW front panel and the Simulink model easier, we will add two gain blocks to the model as shown next.
- These gain blocks will make it easy to identify signals and label front panel controls:

MotoTron

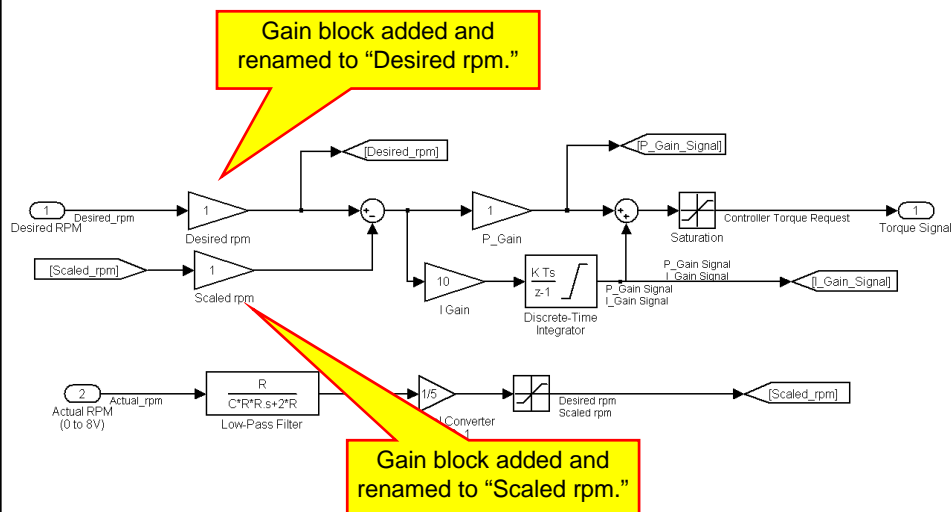
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller Modifications

26



MotoTron

The MathWorks

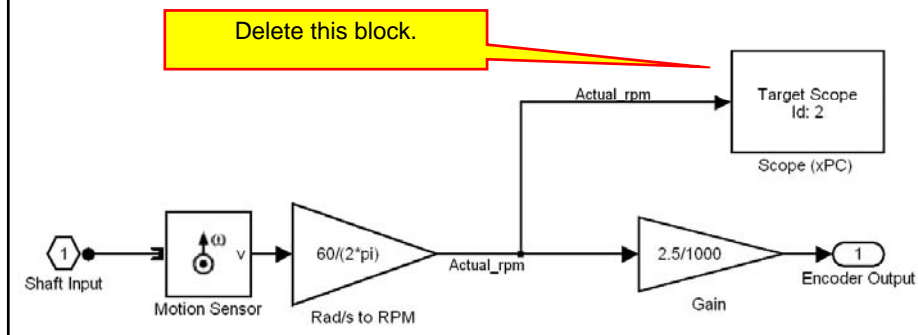
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Plant Model Modifications

27

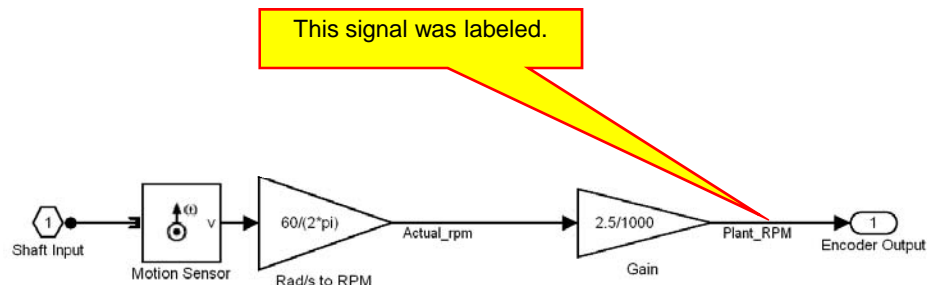
- The only changes we need to make to the plant are in the encoder model.
- This model contains an xPC Target Scope block that must be deleted:



Plant Model Modifications

28

- Label the signal at the encoder output "Plant_RPM."

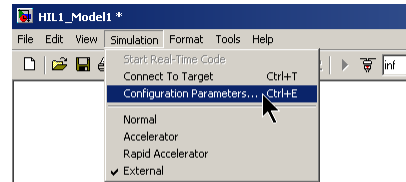




Model Modifications

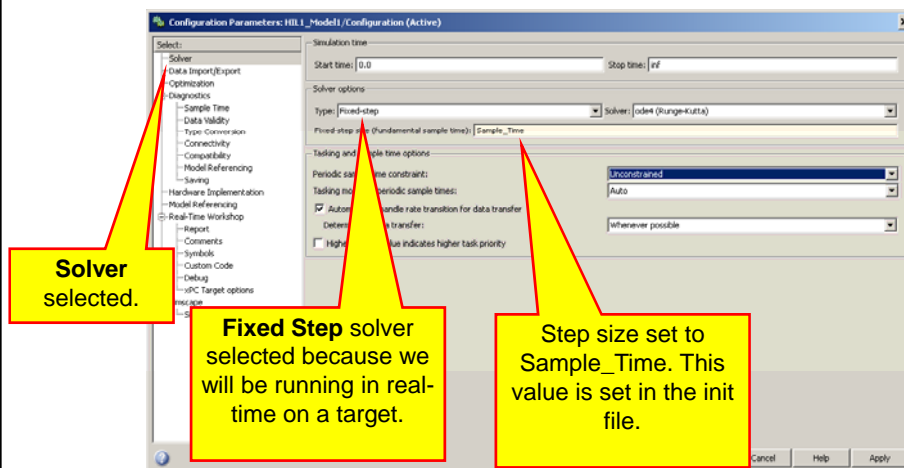
29

- We are now done with modifications to the plant model.
- We need to make some changes to specify that the MathWorks Real-Time Workshop creates a DLL for the National Instruments PXI target rather than the xPC Target.
- From the Simulink menus, select **Simulation** and the **Configuration Parameters** (or type Ctrl e.)



Model Modifications

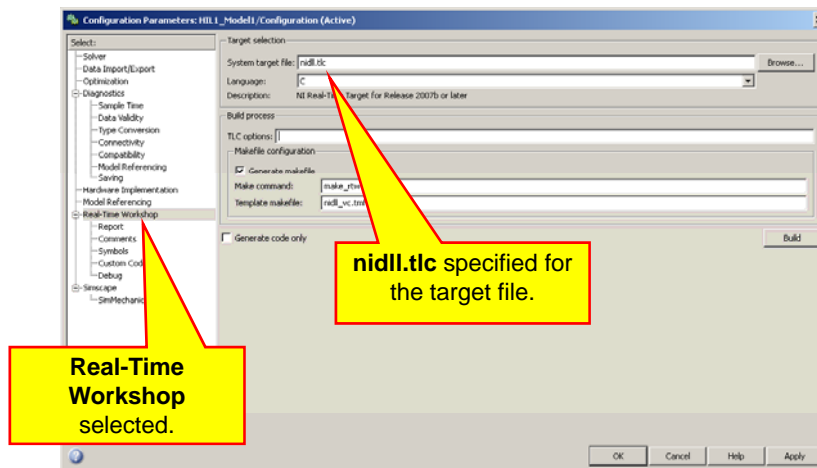
30





Model Modifications

31

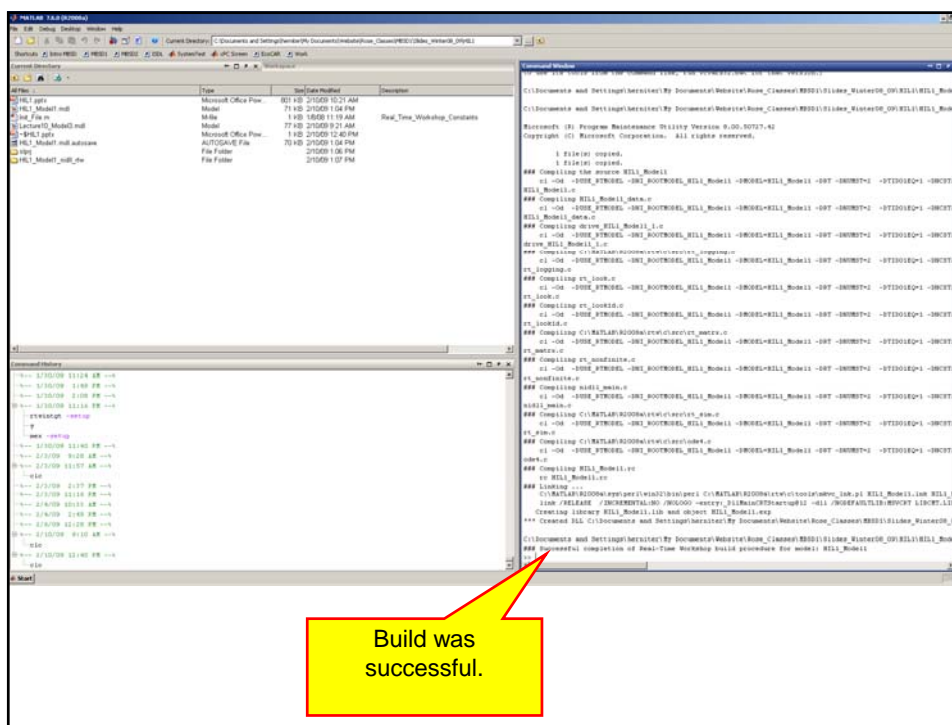


- Nidll.tlc specified that when we build the model, a dll file that can run on an National Instruments target will be created.

Creating the DLL File

32

- Click the **OK** button.
- We are now ready to create the Dynamic-Link Library (dll).
- A dll file is a compiled version of the model that can be executed on the target. The dll has no inputs or outputs, and we will need to create a LabVIEW shell to connect to the inputs and outputs.
- Type ctrl-d to check your model for errors.
- If there are no errors, type ctrl-b to build the model and create the dll file.
- You will see the dialog shown in the next 2 screen captures.





Creating a DLL file

35

- If the build was successful, a dll file will be created in a new sub-directory. My model was named Lecture13_Model1.
 - The new directory that was created is named Lecture13_Model1_nidll_rtw.
 - The dll is located in this directory and is called Lecture13_Model1.dll.
- We will need to point to this file when we set up our LabVIEW front panel.



National Instruments Targets

36

- Now that we have created the DLL for our model, we need to create a LabVIEW shell to specify the inputs to the model and display the model output.
- We can also use the LabVIEW shell to tune the controller parameters.
- Before we create our LabVIEW shell, we need to identify the PXI target on which we will run the model.





National Instruments Targets

37

- We will assume that you have already set up your National Instruments PXI target, it is running properly, it has the correct software installed, and it is plugged into the network.
- For this exercise, we need to discover the IP address of your PXI target.
- There are three ways to discover the IP address of a PXI target connected to the network.



National Instruments Targets

38

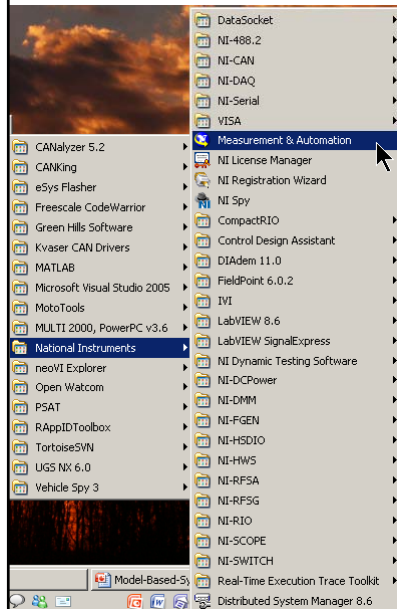
- Ask someone that knows the IP address.
- Some PXI controllers have a VGA output for a monitor. When the system starts up, it will display the IP address.
- Use the National Instruments Measurement & Automation Explorer (MAX)
 - MAX can be used to discover all PXI targets on the local subnet.



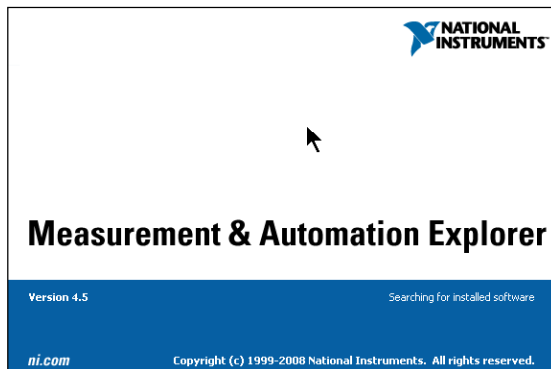


National Instruments MAX

39



- Run the National Instruments Measurement and Automation Explorer



National Instruments MAX

40

- MAX is the software that allows National Instruments hardware and software to work together seamlessly.
- We will only show how to use MAX to discover systems on your local subnet and to add remote systems for which you already know the IP address.



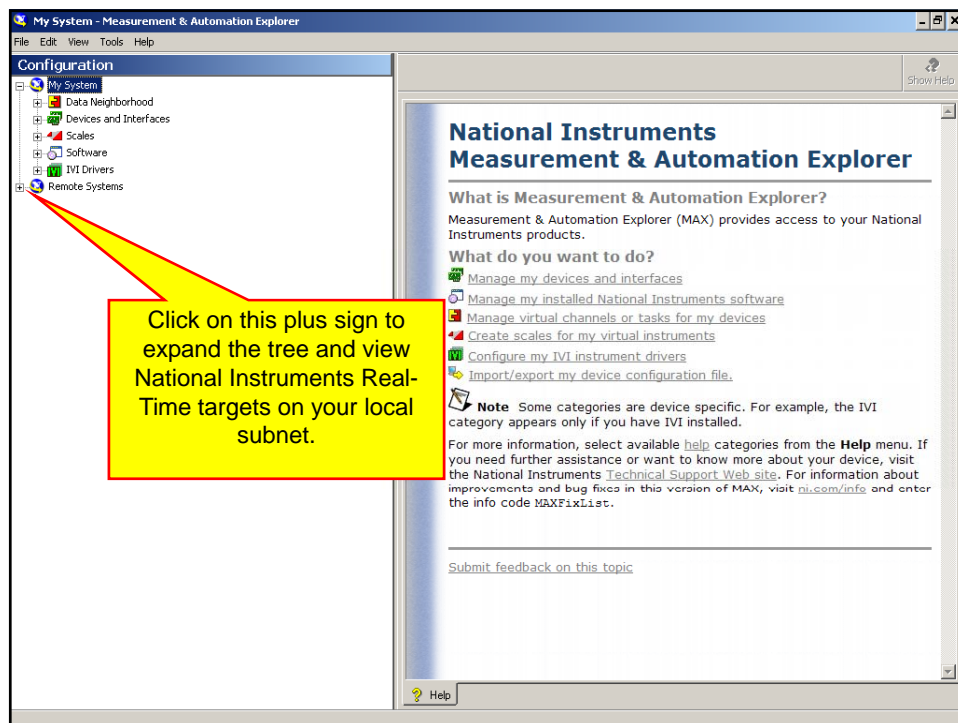
National Instruments MAX

41

- We will assume that this is the first time that you have run MAX.
- If this is the case, MAX will only show you Real-Time targets on your local Subnet.
- When MAX runs, you will see the next screen:





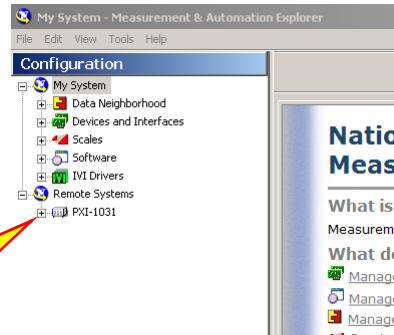


National Instruments MAX

43

- My MAX shows a single system on the local subnet:

Click on this plus sign to expand the tree and properties of this target.



MotoTron

The MathWorks

freescale
semiconductor

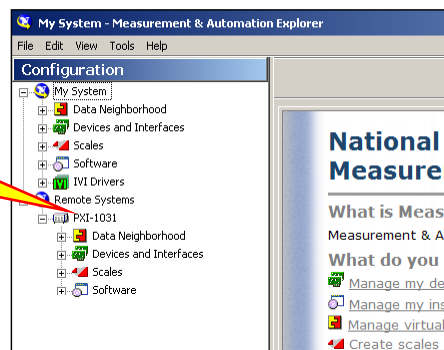
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

National Instruments MAX

44

- We would like to view the network settings for this target:

Click on this text to see the network properties of the target.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

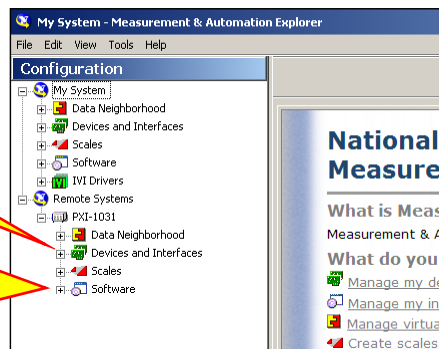
National Instruments MAX

45

- The properties of our target are shown below:

In a future exercise, we will use part of the tree to configure the cards installed on the target.

You can view and install software on the Target in this part of the tree. The versions of software on your PC must match the versions of software installed on the target. We will assume that the software has been set up correctly.

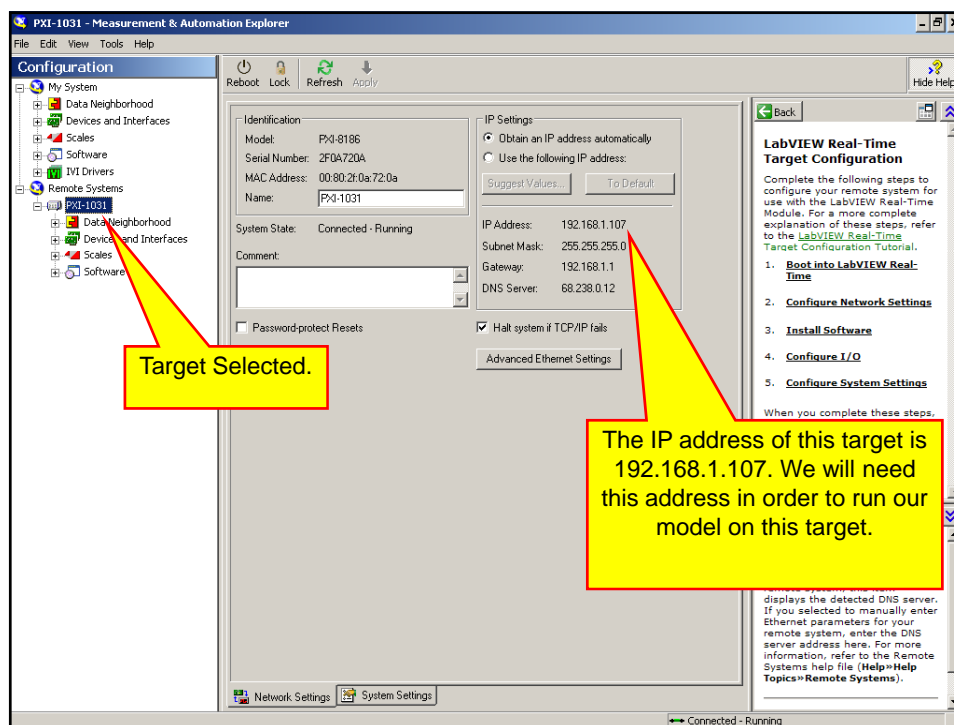


MotoTron

The MathWorks

freescale
semiconductor

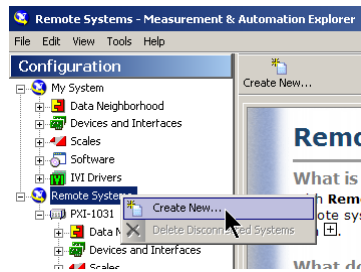
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



National Instruments MAX

47

- If you know the IP address of a target that is not on your local subnet, you can view the target by adding it as a remote target.
- In MAX, right click on **Remote Systems** and then select **Create New**:



MotoTron

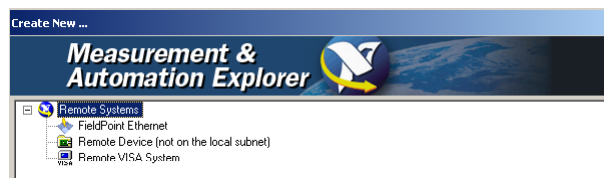
The MathWorks

freescale
semiconductor

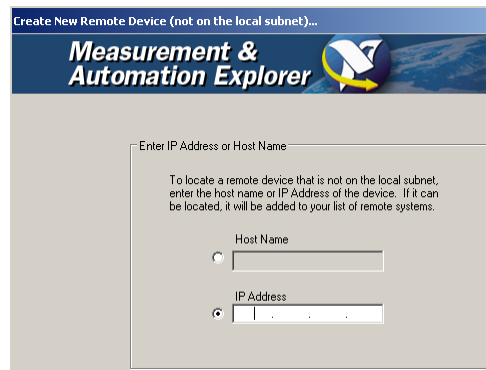
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

National Instruments MAX

48



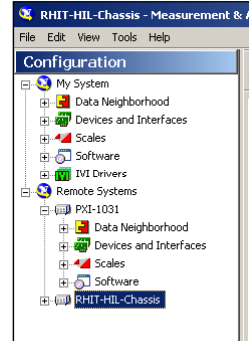
- Select **Remote Device** and then click the **Next** button.
- Enter the IP address of the known system and click the **Finish** button.



National Instruments MAX

49

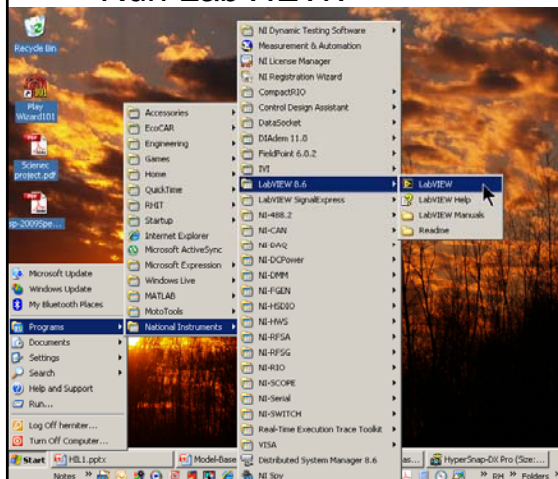
- If you have access to the system (and there is not a firewall set up to block access) you can view your remote target.
- Once you add remote systems, MAX will remember those systems the next time you run it.
- For our example, all we need to know is the IP address of our target.



National Instruments LabVIEW

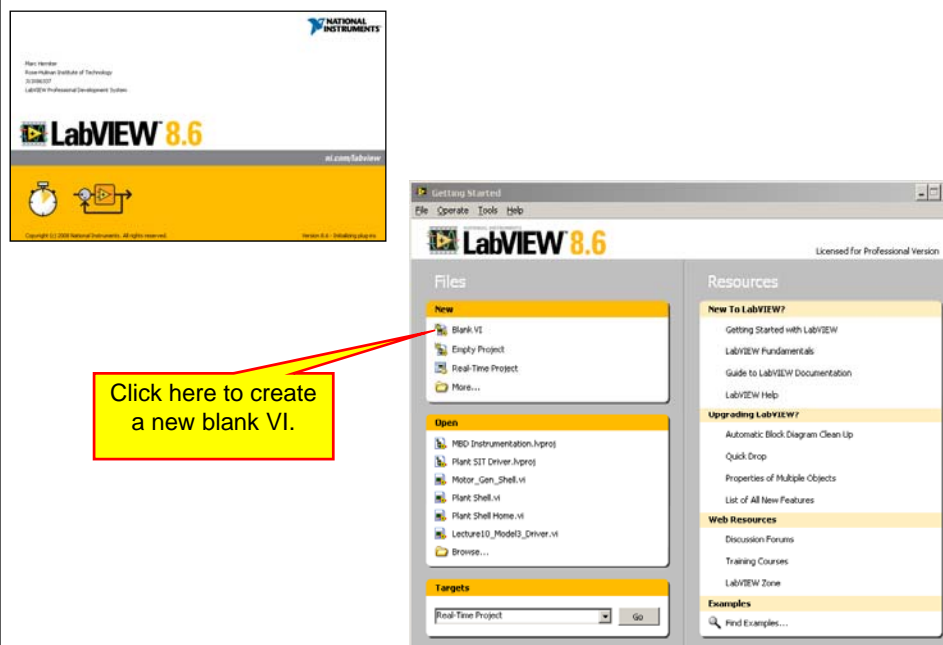
50

- Next, we want to run LabVIEW and create a shell to communicate with our model DLL.
- Run LabVIEW:



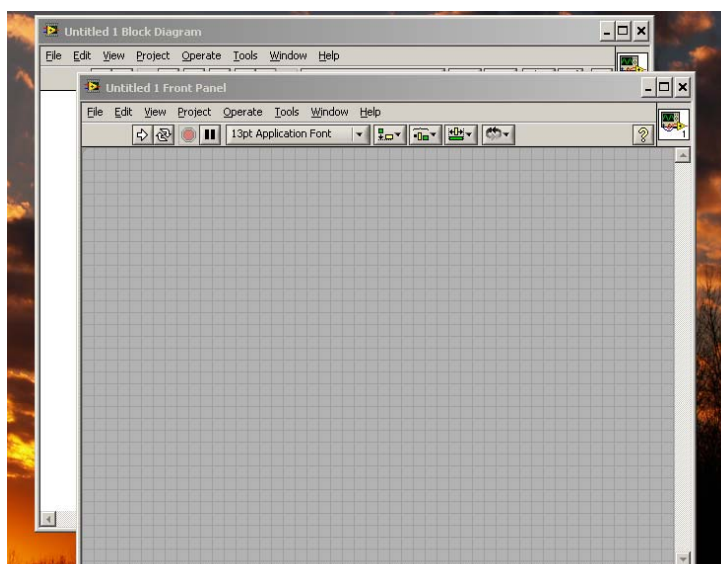
National Instruments LabVIEW

51



National Instruments LabVIEW

52



LabVIEW

53

- Before we begin, we would like to save the VI.
- Select **File** and then **Save** from the LabVIEW menus.
- Save the file in the directory where you saved your Simulink model.
- Save the VI as Motor Generator Shell.VI

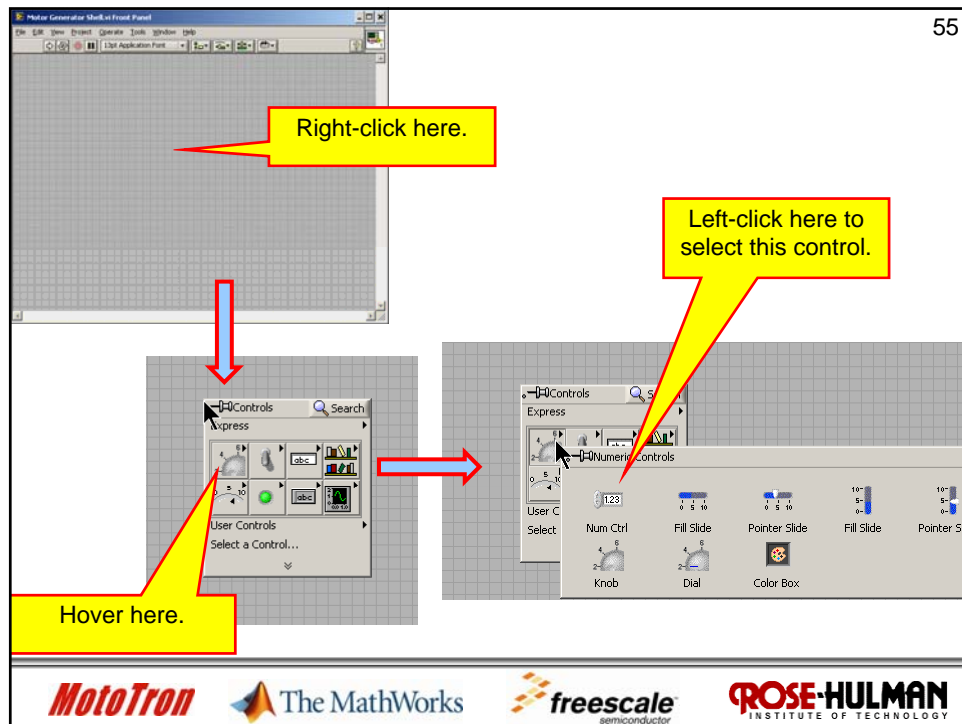


Motor Generator Shell

54

- Our motor-generator system has two user inputs (desired speed and number of light bulbs) and one output (motor speed at 2.5 V per 1000 rpm).
- We will place two controls and one chart in the LabVIEW front panel and connect them with the Simulink model signals.

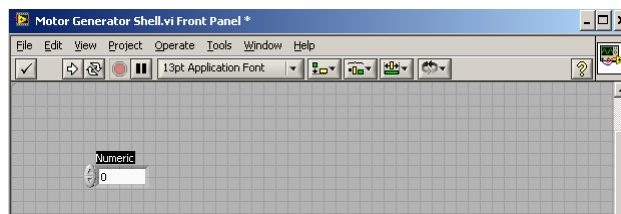




Motor-Controller Shell

56

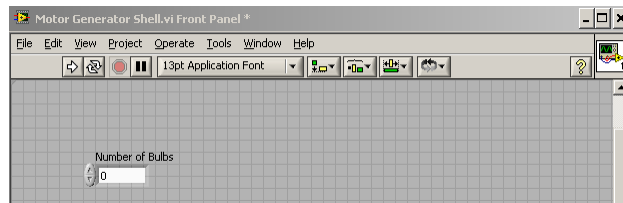
- When you left click on the control, it becomes attached to the mouse and the menu disappears.
- Place the control in your front panel by left clicking on the location you wish to place it:



Motor Generator Shell

57

- Note that the name is highlighted, so that we can easily rename it.
- Change the name to Number of Bulbs.



MotoTron

 **The MathWorks**

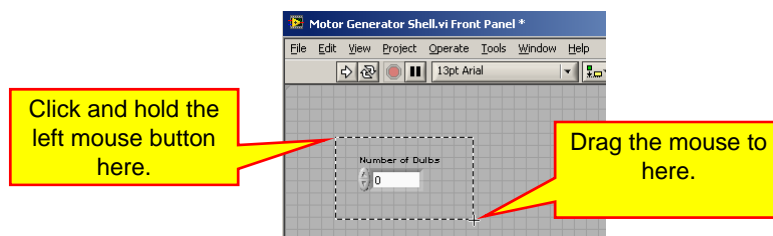
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Motor Generator Shell

58

- Next, we want to select the entire indicator.
- Click and drag the left mouse as shown



MotoTron

 **The MathWorks**

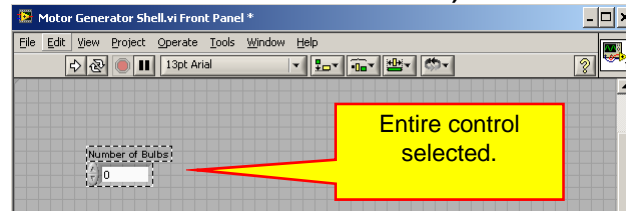
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Motor Generator Shell

59

- When you release the mouse button, the entire indicator will be selected (both the label and the indicator field).



- Click on the font pull-down menu as shown next:

MotoTron

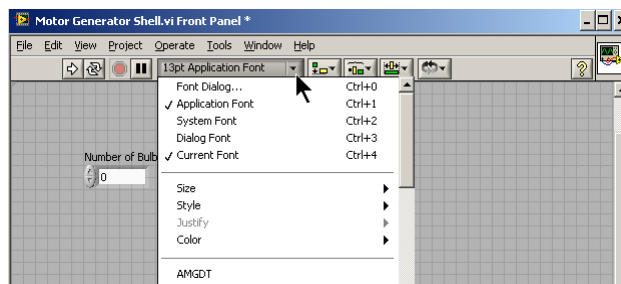
The MathWorks

freescale
semiconductor

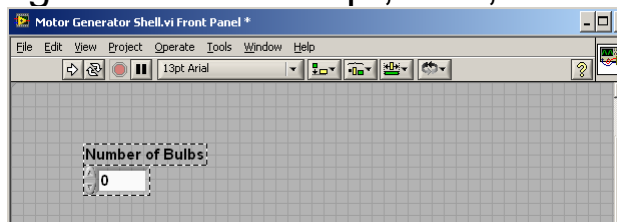
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Motor Generator Shell

60



- Change the font to 18 pt, bold, and Arial.



MotoTron

The MathWorks

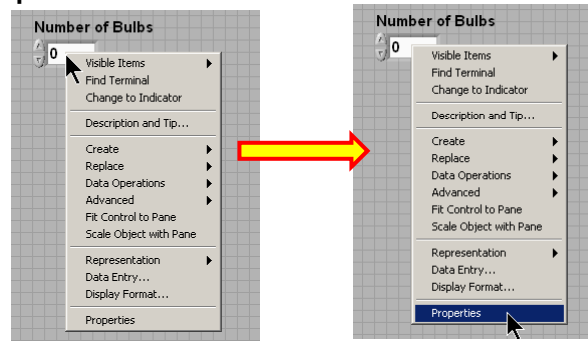
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Motor Generator Shell

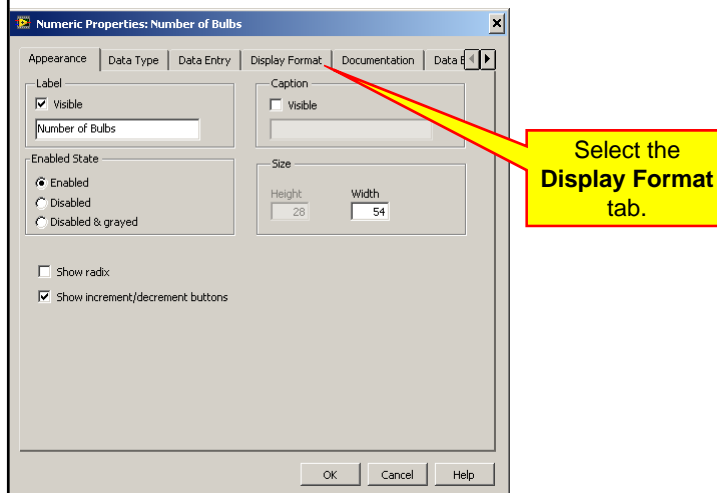
61

- Next, we want to change the control to select and display an integer between 0 and 6.
- Right click on the control and select properties:



- You are encouraged to play with the settings in this tab to see their effect on the display of the control.

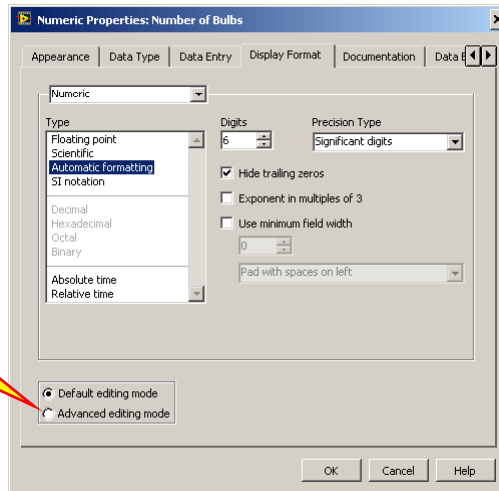
62



63

- There are several ways we could display a single digit. We will only show one here.

Click here to select the **Advanced editing mode**.



MotoTron

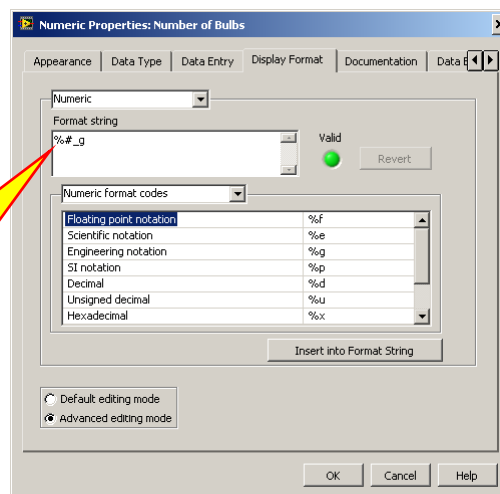
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

64

Replace this text with `%2.0f`. This means display a floating point number with a total of 2 digits and 0 digits displayed after the decimal point.



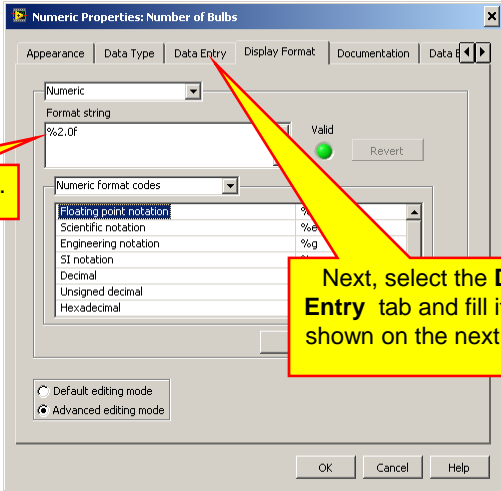
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

65



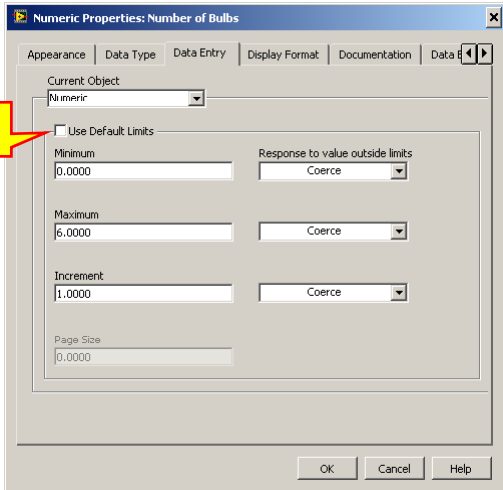
Format string changed.

Next, select the Data Entry tab and fill it in as shown on the next slide.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

66

- The selected settings for the control force the choice of an integer between 0 and 6.



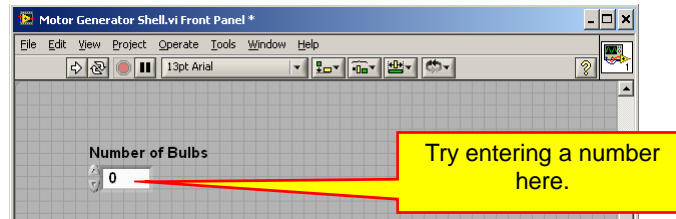
Option is not selected.

- Click the **OK** button to apply the changes and return to the front panel.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

LabVIEW Front Panel

67



- The display may look unchanged, but if you enter a number into the field, you will notice that you can only specify an integer between 0 and 6.

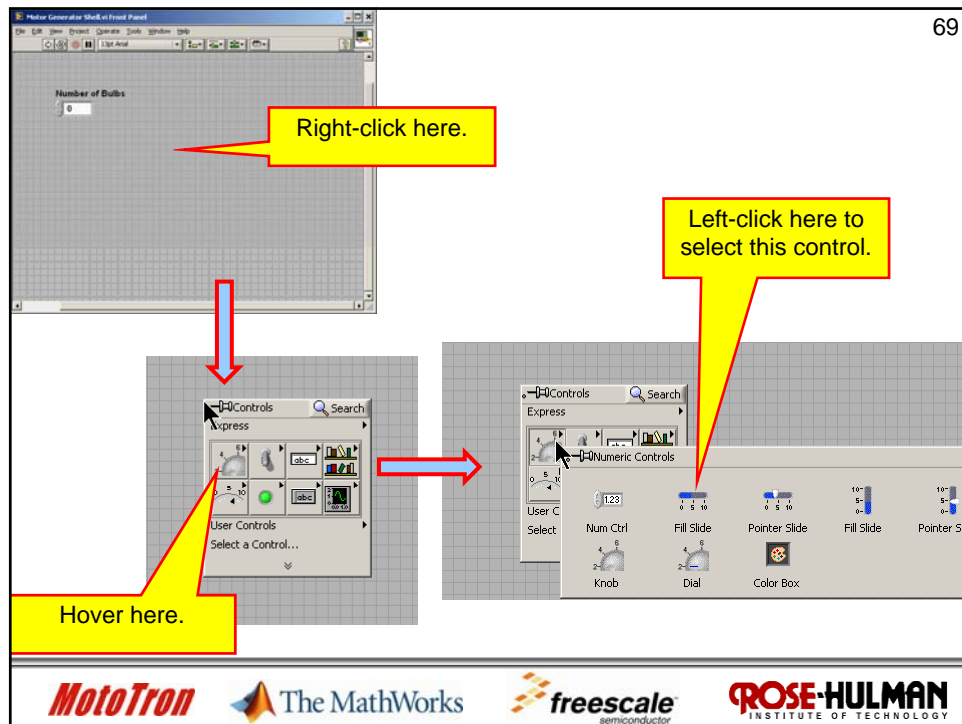
MotoTron The MathWorks freescale
semiconductor**ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY

LabVIEW Front Panel

68

- Next, we would like to create a control for the desired speed.
- We will use a fill slide and constrain it to have a value between 0 and 1 with 0.1 step increments.

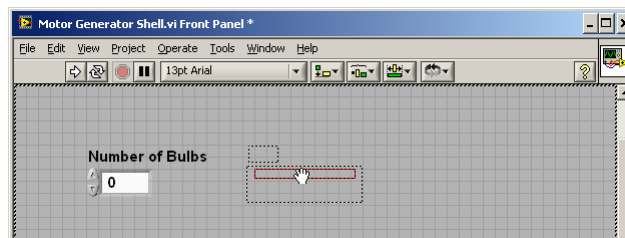
MotoTron The MathWorks freescale
semiconductor**ROSE-HULMAN**
INSTITUTE OF TECHNOLOGY



Motor-Controller Front Panel

70

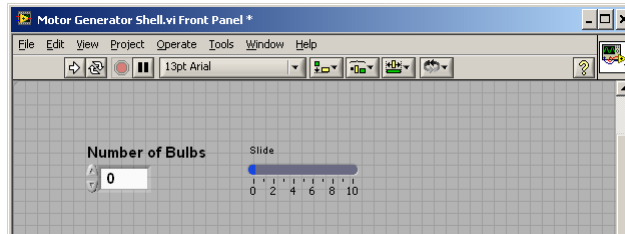
- When you left click on the control, it becomes attached to the mouse and the menu disappears.



Motor-Controller Front Panel

71

- Place the control in your front panel by left clicking on the location you wish to place it:



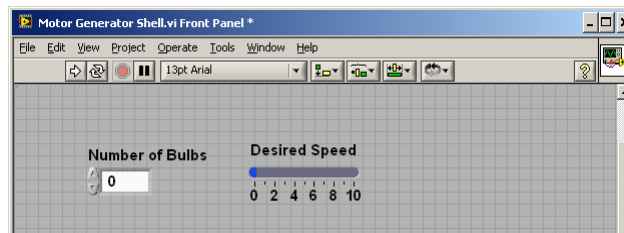




Motor-Generator Front Panel

72

- Change the name of the control to "Desired Speed."
- Change the font to 18 pt Arial bold using the same techniques we used for the numerical control.



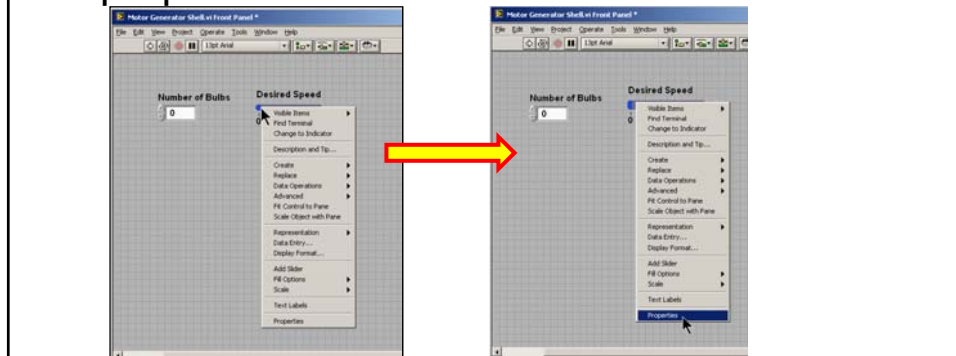




Motor-Generator Shell

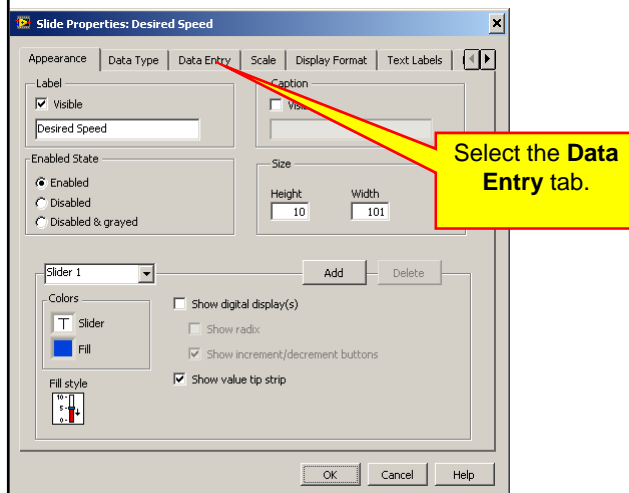
73

- Next, we want to change the control to select a number between 0 and 1 with 0.1 steps.
- Right click on the control and select properties:



- You are encouraged to play with the settings in this tab to see their effect on the display of the control.

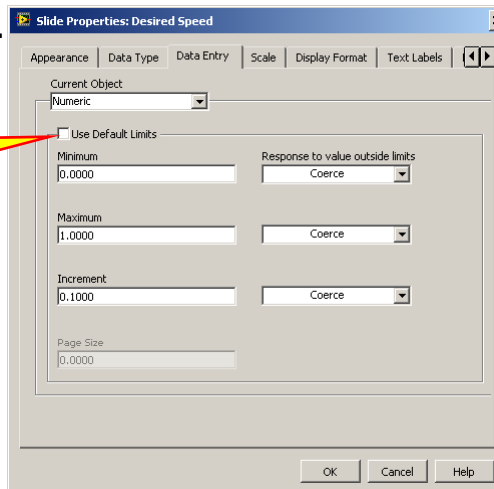
74



- The selected settings for the control force the selection of a value between 0 and 1 with 0.1 size steps.

75

Option is not selected.



The dialog box 'Slide Properties: Desired Speed' has tabs for Appearance, Data Type, Data Entry, Scale, Display Format, and Text Labels. The 'Scale' tab is active. It shows 'Current Object' as 'Numeric'. The 'Use Default Limits' checkbox is unchecked. The 'Minimum' is 0.0000, 'Maximum' is 1.0000, and 'Increment' is 0.1000. The 'Response to value outside limits' dropdown is set to 'Coerce'. The 'Page Size' is 0.0000. Buttons for OK, Cancel, and Help are at the bottom.



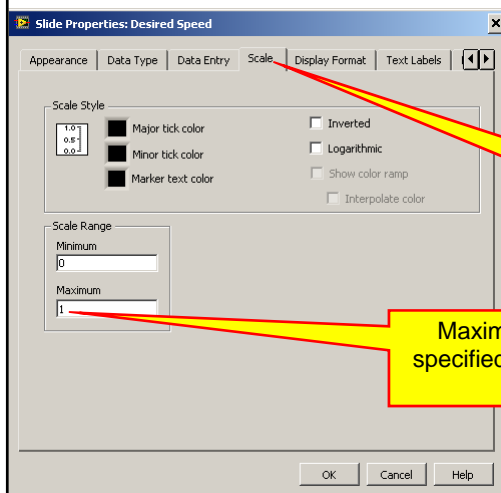



- Select the **Scale** tab and change the maximum value to 1.
- Click the **OK** button when done.

76

Select the **Scale** tab.

Maximum specified as 1.



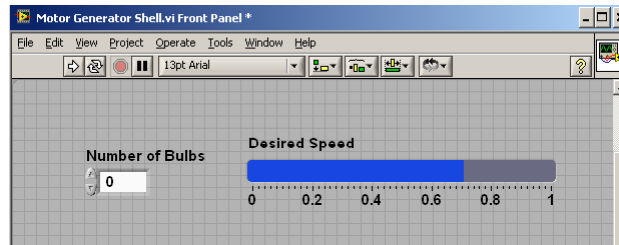
The dialog box 'Slide Properties: Desired Speed' has tabs for Appearance, Data Type, Data Entry, Scale, Display Format, and Text Labels. The 'Scale' tab is active. It shows 'Scale Style' with a numeric scale from 0.0 to 1.0. The 'Scale Range' section shows 'Minimum' as 0 and 'Maximum' as 1. Buttons for OK, Cancel, and Help are at the bottom.

Front Panel

77



- Click on the control and use the handlebars to resize the fill slide:



MotoTron

The MathWorks

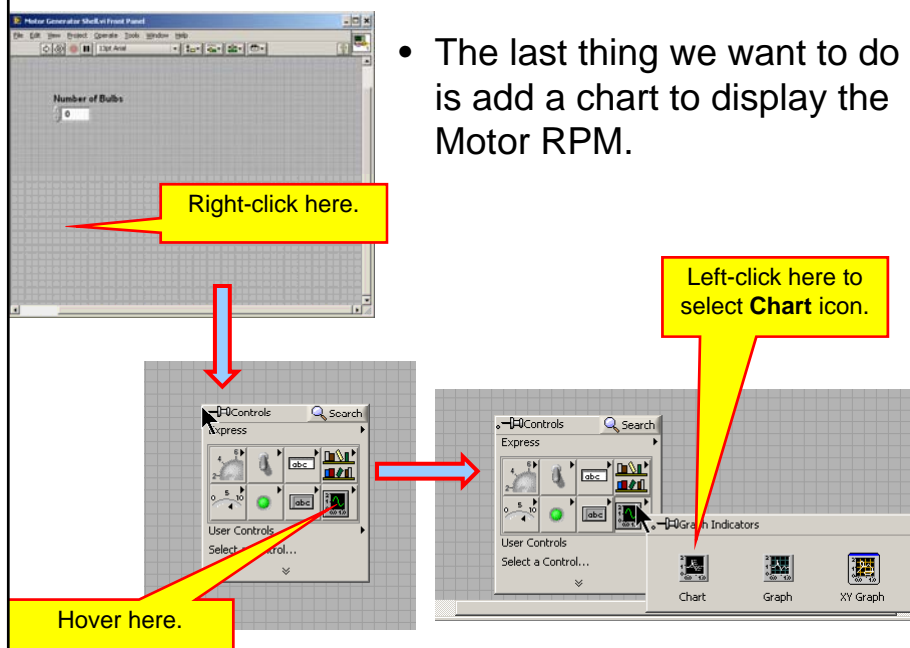
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Motor-Generator Front Panel

78

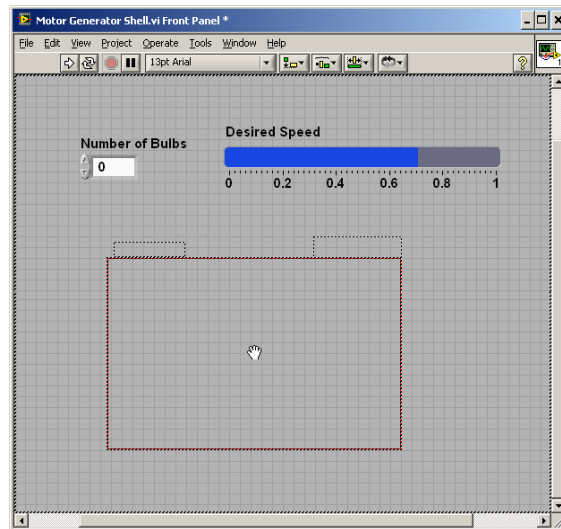
- The last thing we want to do is add a chart to display the Motor RPM.



Motor-Controller Front Panel

79

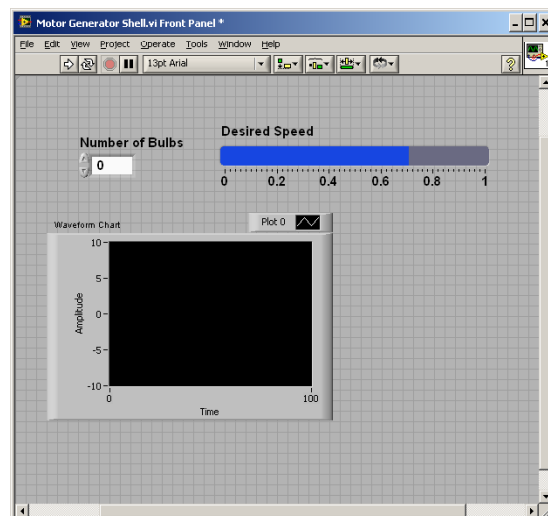
- When you left click on the chart, it becomes attached to the mouse and the menu disappears.



Motor-Controller Front Panel

80

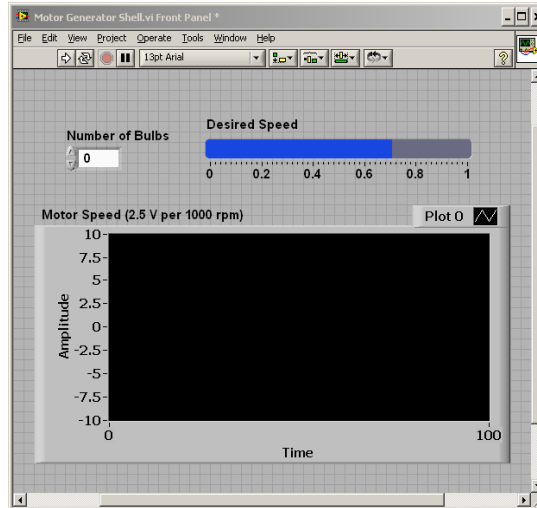
- Place the control in your front panel by left clicking on the location you wish to place it:



Motor-Generator Front Panel

81

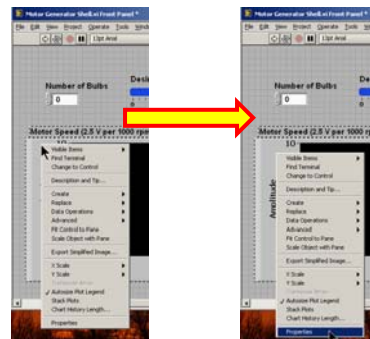
- Change the name of the chart to “Motor Speed (2.5 V per 1000 rpm).”
- Change the font to 18 pt Arial bold using the same techniques we used for the numerical controls.
- Resize the chart to fit the available space in the window:



Motor-Generator Shell

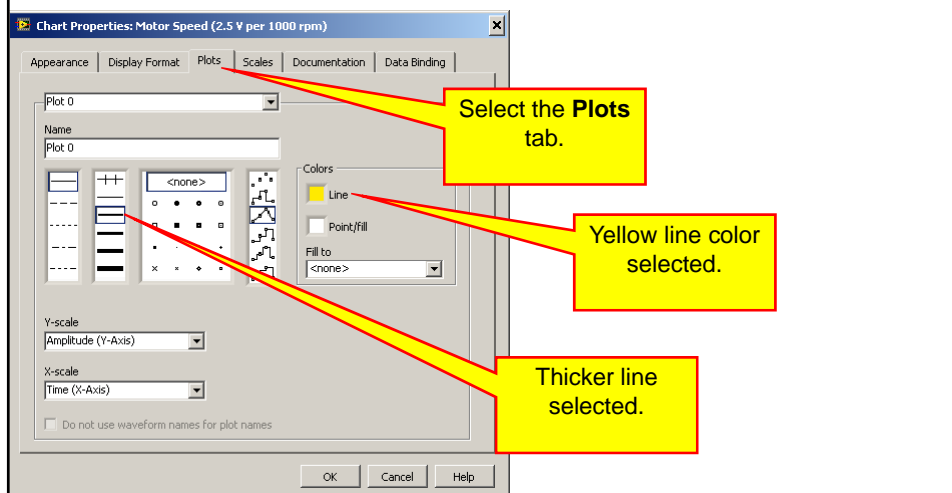
82

- Next, we want to change the chart to display 20 seconds of data and change the y-axis to have a scale of 0 to 8.
- Right click on the chart and select properties:



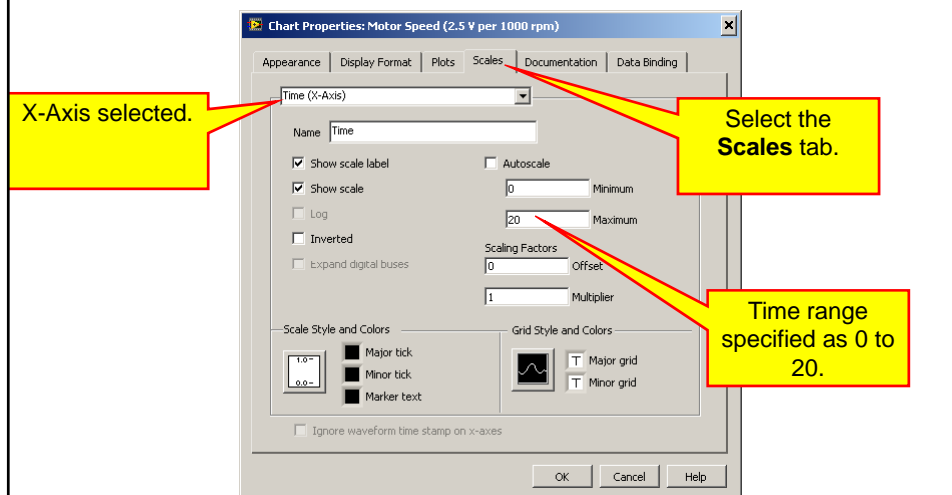
83

- The first thing we will do is change the line thickness and color of the trace. Select the **Plots** tab and make the selections shown below:



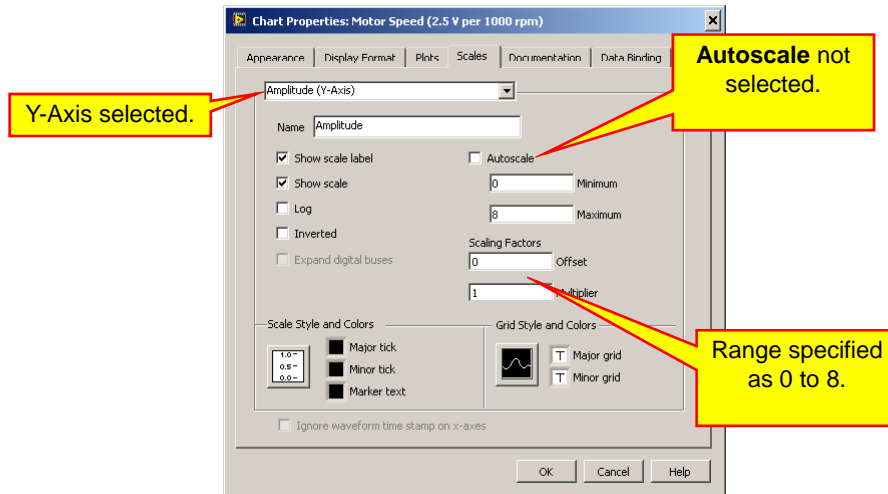
84

- Select the **Scales** tab and make the changes shown below to specify that the time axis have a range of 20 seconds:



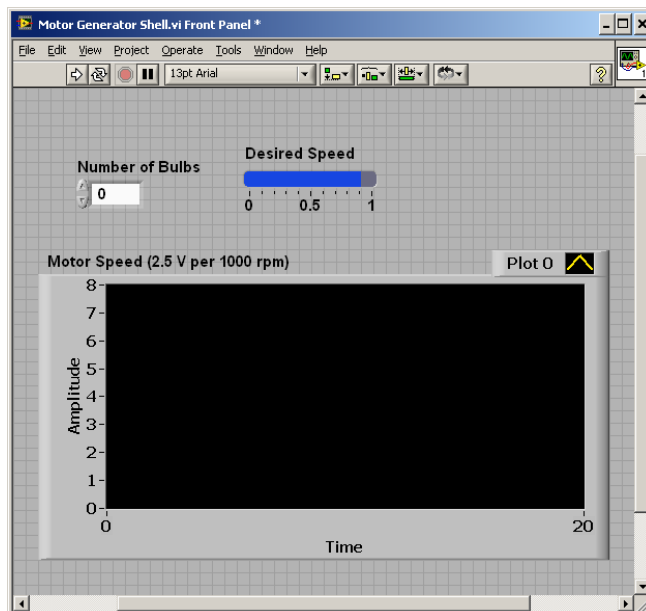
85

- Next, make the changes shown below to specify that the Y-axis have a range of 0 to 8:



86

- We are done with the changes for the chart..
- Click the **OK** button when done.



Motor-Generator Shell

87

- We have now constructed a basic front panel for our Motor-Generator system.
- The next step is to use the National Instruments Simulation Interface Toolkit (SIT) to connect the front panel controls and chart to the DLL we created with Simulink.
- Save your LabVIEW model before continuing.

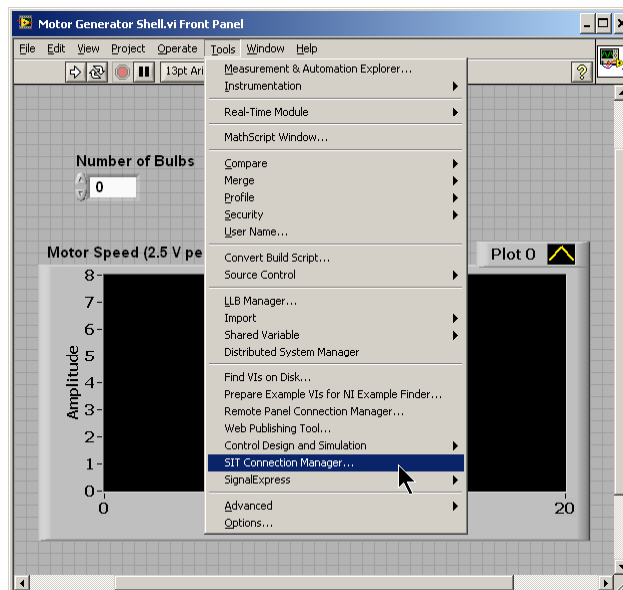





Simulation Interface Toolkit

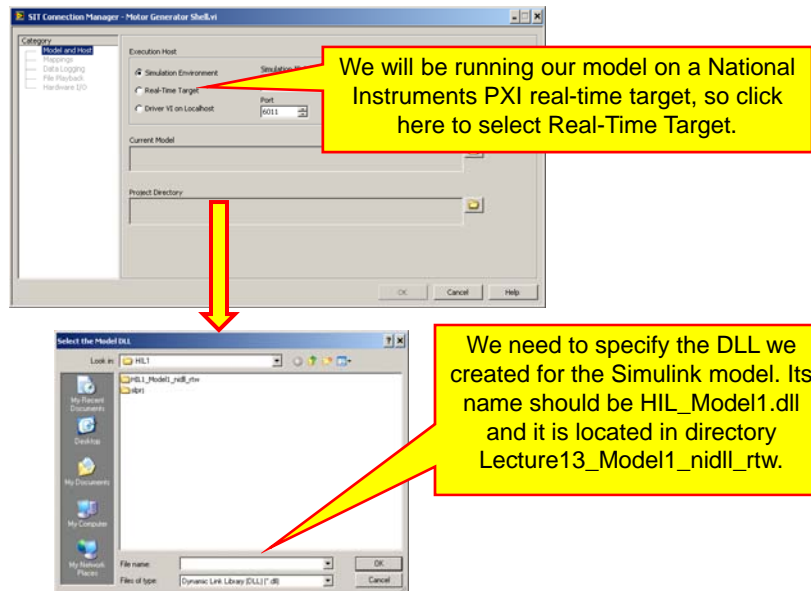
88

- We are now ready to connect our front panel to the Simulink model compiled into a DLL.
- From the LabVIEW menus, select **Tools** and then **SIT Connection Manager**:



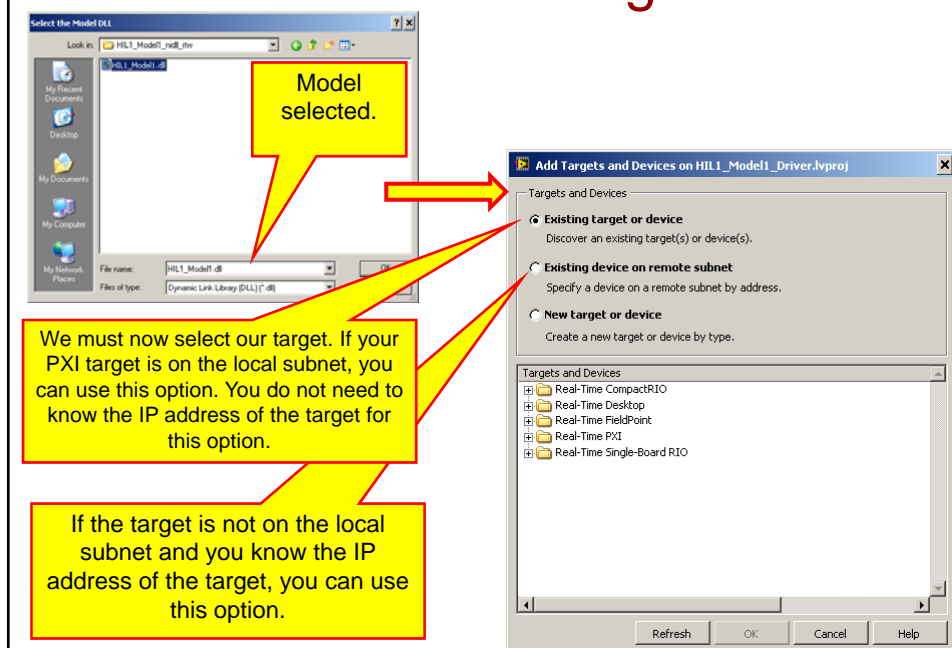
SIT Connection Manager

89



SIT Connection Manager

90



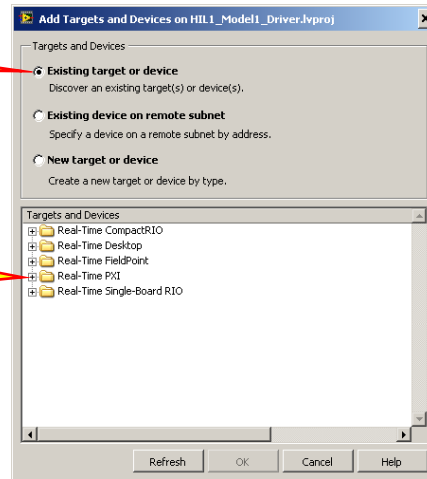
SIT Connection Manager

91

- My target is on the local subnet:

I will select this option.

To detect PXI systems on the local subnet, click on this + sign. (Make sure that your local PXI systems are plugged in to the network and that the power is turned on.)



MotoTron

The MathWorks

freescale
semiconductor

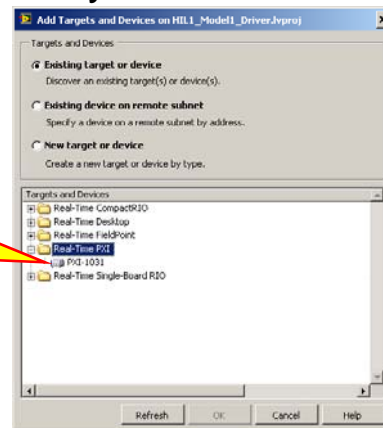
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Sit Connection Manager

92

- In my system, the SIT Connection Manager located one PXI system:

Click on the PXI target that you want to use and then click the **OK** button. If you do not have any PXI systems listed here, you may need use the option for remote systems and specify the IP address of the target manually.



MotoTron

The MathWorks

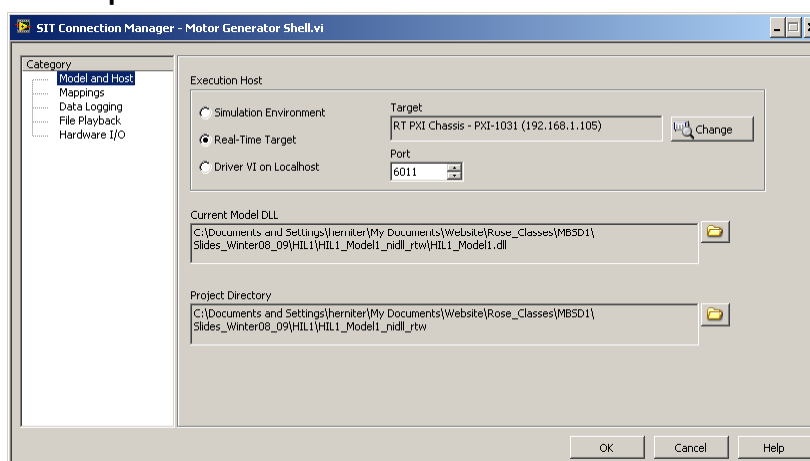
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

SIT Connection Manager

93

- After selecting the target, we return to the SIT main dialog box with the target and model specified:

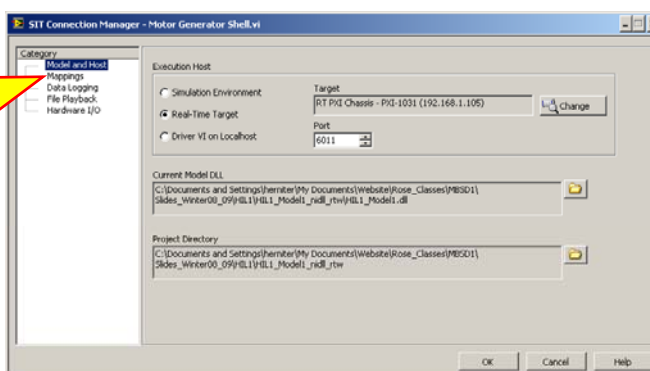


SIT Connection Manager

94

- The last step we need to take is to specify connections between our front panel and the Simulink model.

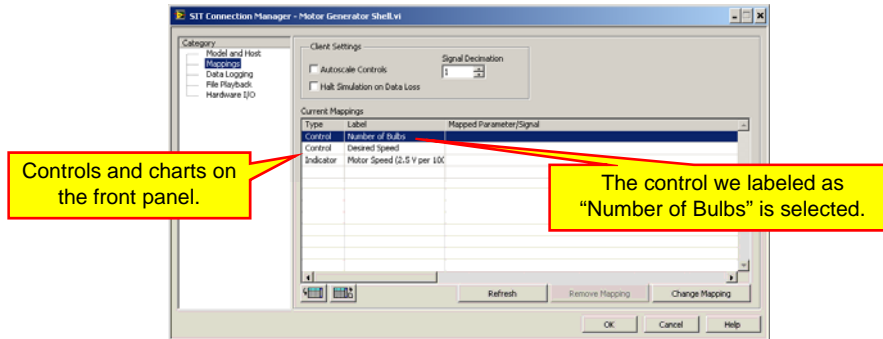
Click on the **Mappings** category to specify connections between the Simulink model and the front panel.



SIT Connection Manager

95

- The dialog box now lists the LabVIEW controls and charts in the front panel. We need to manually connect each to a signal in the Simulink model.



MotoTron

The MathWorks

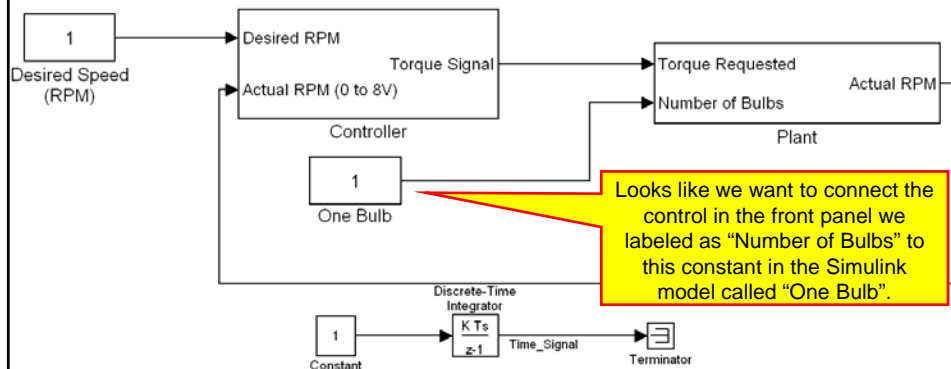
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Simulink Model

96

- We would like to connect this to the constant block in the top level of our Simulink model. The model is shown



MotoTron

The MathWorks

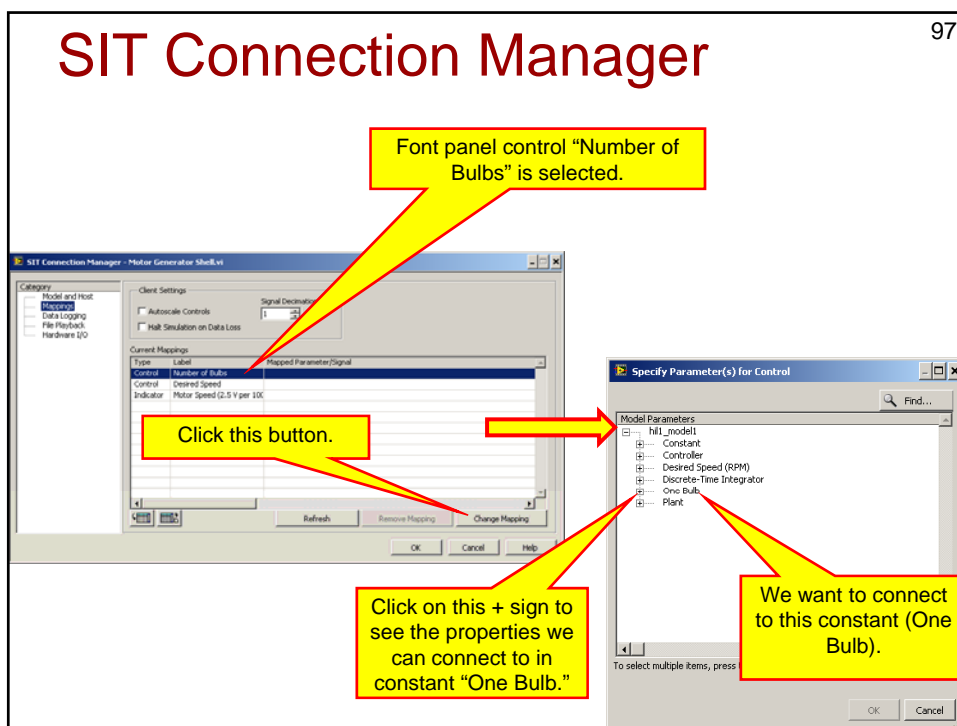
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



SIT Connection Manager

97



Font panel control "Number of Bulbs" is selected.

Click this button.

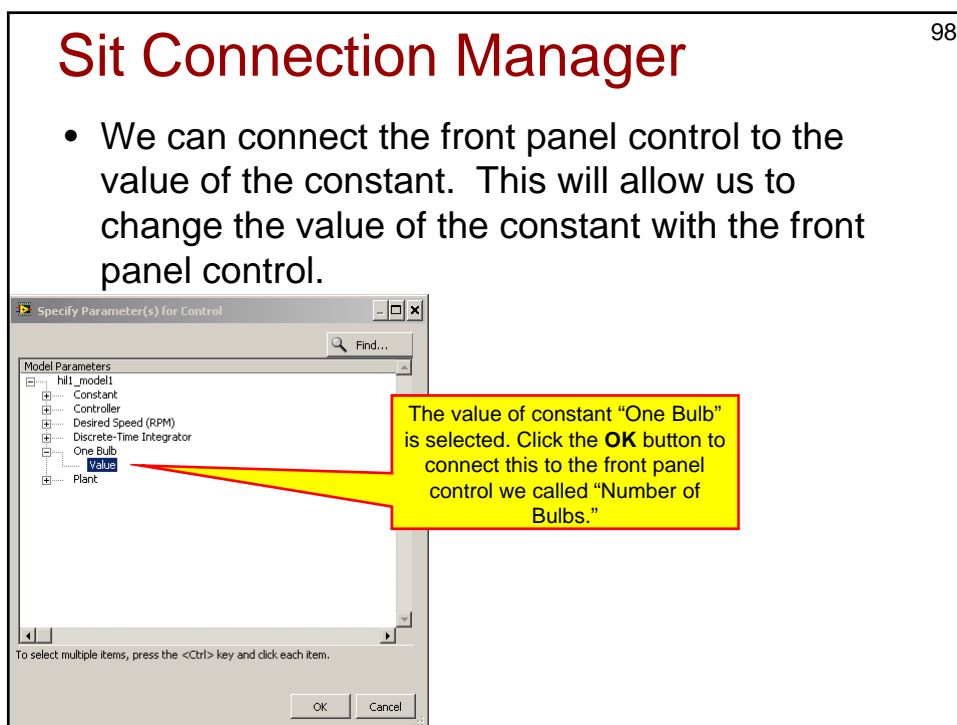
Click on this + sign to see the properties we can connect to in constant "One Bulb."

We want to connect to this constant (One Bulb).

Sit Connection Manager

98

- We can connect the front panel control to the value of the constant. This will allow us to change the value of the constant with the front panel control.

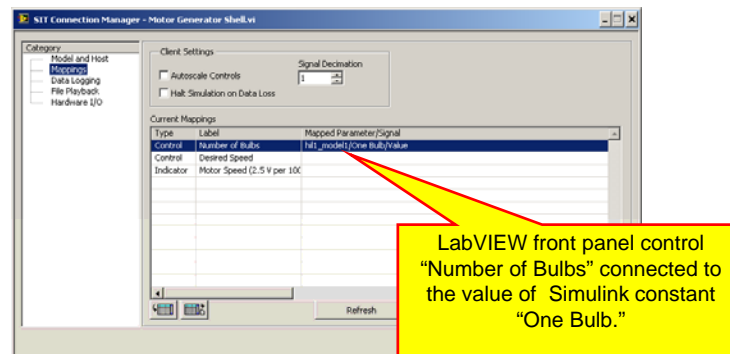


The value of constant "One Bulb" is selected. Click the **OK** button to connect this to the front panel control we called "Number of Bulbs."

SIT Connection Manager

99

- The Dialog box now shows that we have connected the front panel control to the Simulink constant.



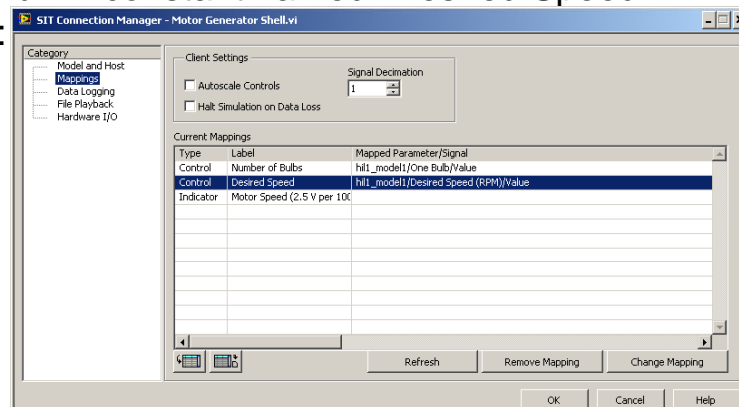




SIT Connection Manager

100

- Repeat the process to connect the front panel control called "Desired Speed" to the value of the Simulink constant named "Desired Speed (RPM):





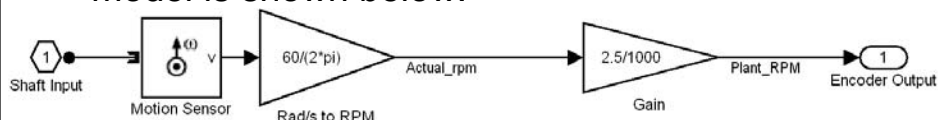




SIT Connection Manager

101

- The last thing we need to do is connect the Motor Speed chart to the encoder output signal in the Simulink model. The Simulink encoder model is shown below:

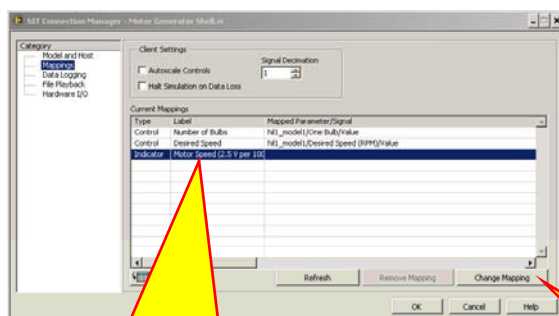


- This shows us that we need to connect the chart to the output of the gain block labeled “Gain” in the “Encoder” subsystem of the Simulink model.

SIT Connection Manager

102

- Select the Motor Speed chart and click the **Change Mapping** button:



Motor Speed chart (it is classified as an indicator) is selected.

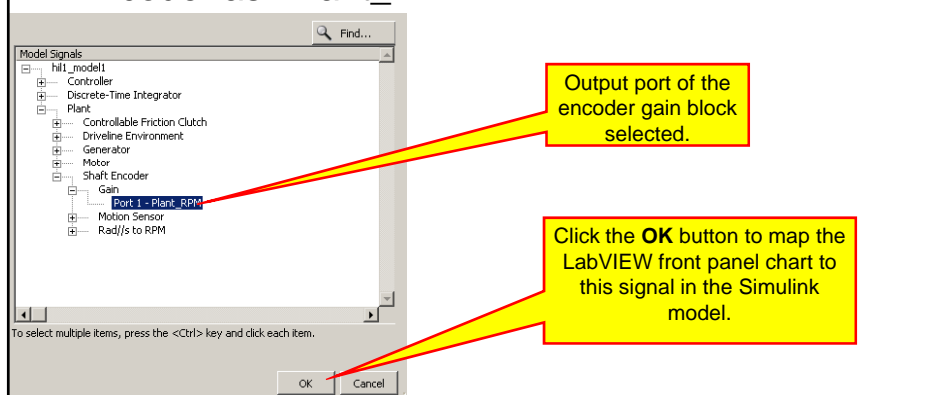
Click here.



Encoder Mapping

103

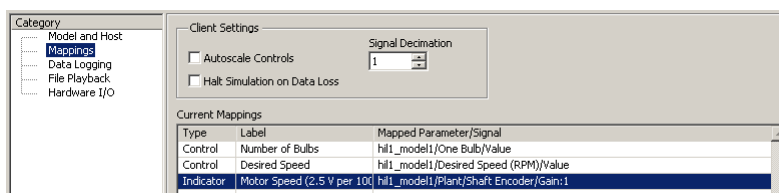
- Expand the Plant subsystem and then the Encoder subsystem until you see the gain with the output port labeled as “Plant_RPM.” If you recall, we labeled the signal out of the gain block in the Encoder as “Plant_RPM.”



SIT Connection Manager

104

- We have now mapped all of the LabVIEW front panel objects to the Simulink DLL.



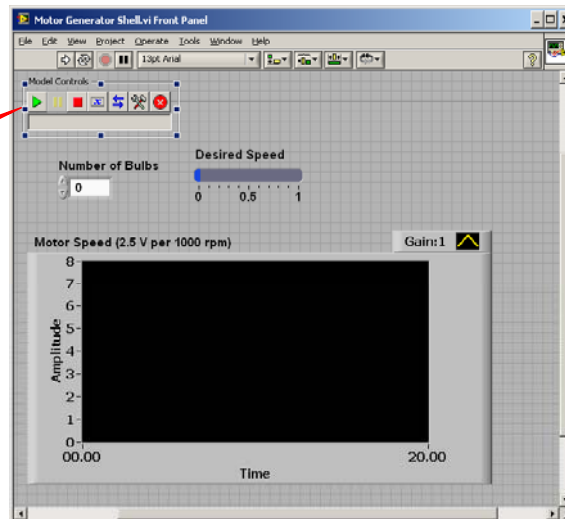
- When we click the **OK** button, the SIT Connection Manager will write the LabVIEW VI to run the Simulink DLL on the remote target and establish communication between the front panel and the real-time target.

LabVIEW Front Panel

105

- The front panel has been modified with controls to run the model.

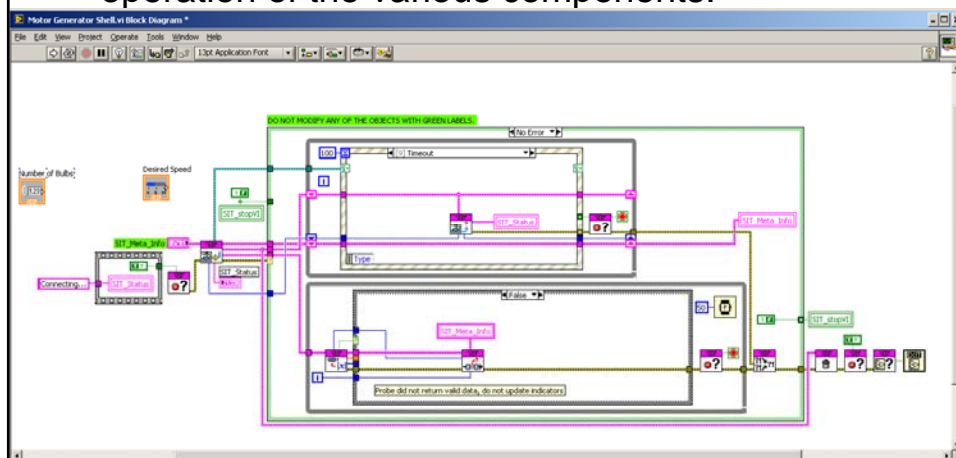
Controls added by the SIT Connection Manager.



Block Diagram








106

- If you open the block diagram window, you will see the block diagram created by the SIT Connection Manager. We will not discuss the operation of the various components.



Front Panel Controls

107


- The front panel controls added by the SIT Connection Manager have the following functions:
 -  Run the Simulation.
 -  Pause the simulation.
 -  Stop the Simulation.
 -  Edit parameters.
 -  Remap front panel controls and indicators.
 -  Show simulation details
 -  Stop the front panel VI but allow the DLL to keep running on the remote target.



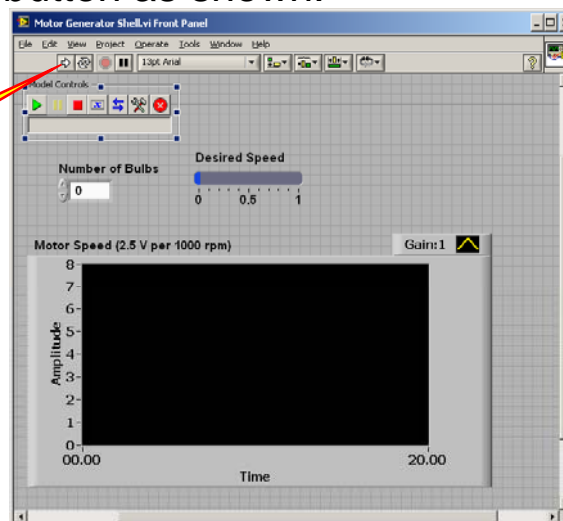



Running the Model

108

- We are now ready to run the model. Click on the **RUN**  button as shown:

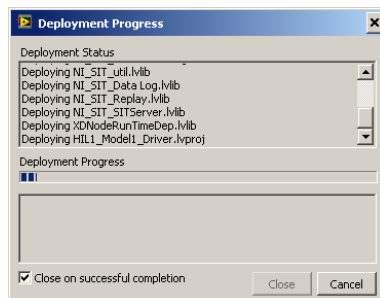
Click here.



Running the Model

109

- When you click the **Run** button, LabVIEW will connect to the target and begin downloading the model and related VIs:



MotoTron

The MathWorks

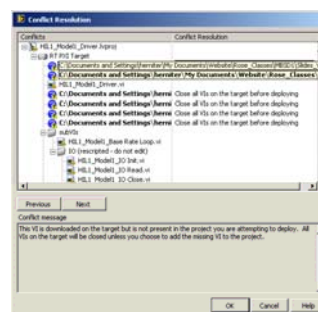
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Running the Model

110

- You might get the notification:



- This message is notifying us that there are already some VIs that are present on the target that are not needed for the project we want to run. This is not a problem, so click the **OK** button.

MotoTron

The MathWorks

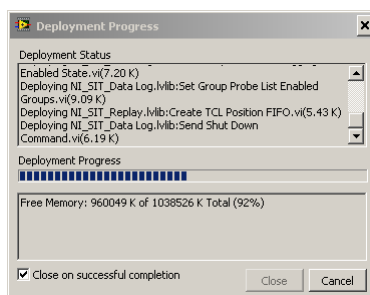
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Running the VI

111

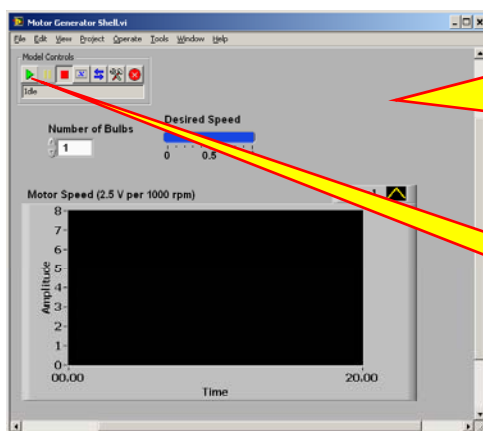
- The model will continue to download:



Running the Model

112

- When the download is complete, the front panel will look a little different:



The grid is gone. This indicates that the VI is running on our local PC.

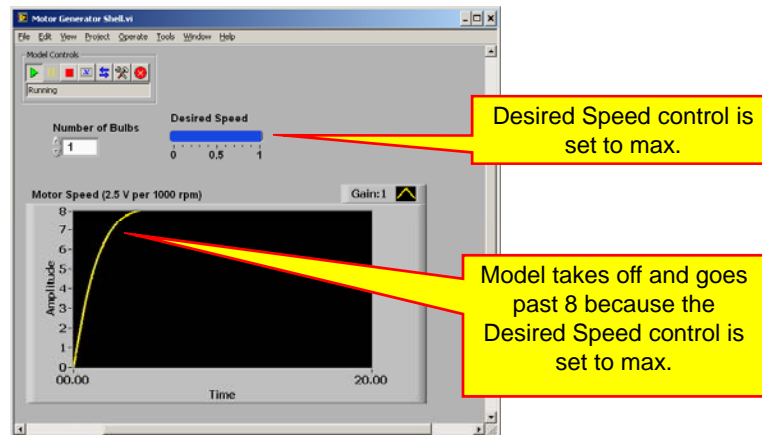
The Simulink DLL is not, however, running on the remote target.

Click here to run the Simulink DLL on the remote PXI target.

Running the Model

113

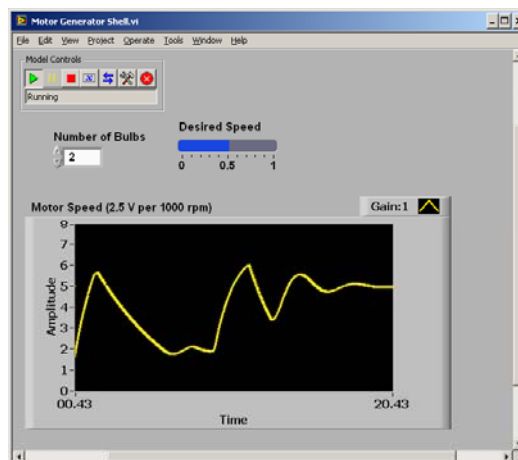
- When you click the **Run** button, the Simulink DLL will run in real-time on the remote target.



Running the Model


114

- You can now change the number of bulbs and the desired speed and watch the system respond in real-time.

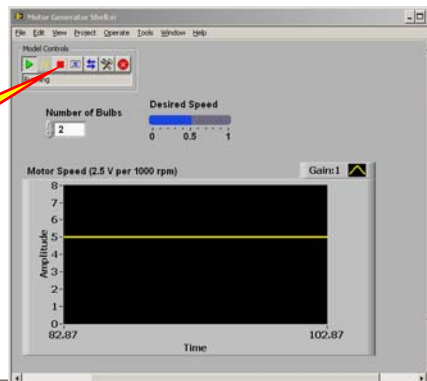


Stopping the Simulation

115

- To stop the simulation on both the remote target and your local computer (which is running the display panel), click the Stop Simulation button :

Click here to stop the simulation on both the remote target and your local PC.



MotoTron

The MathWorks

freescale
semiconductor

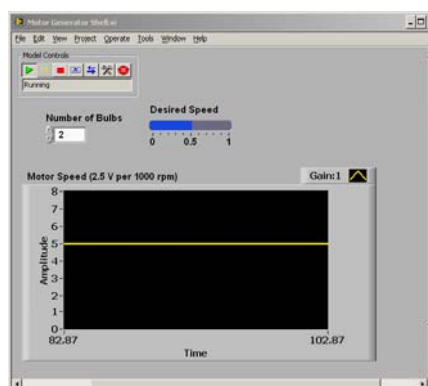
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Demo

116

- Demonstrate the working model running in real-time.

Demo_____



MotoTron

The MathWorks

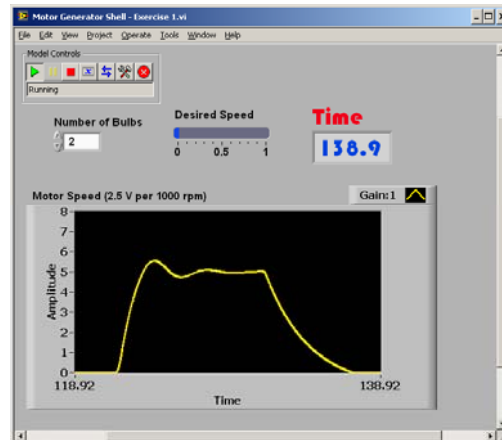
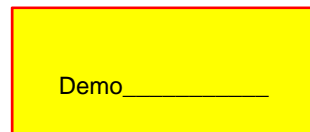
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Exercise 1

117

- Modify the front panel so that the current simulation time is displayed as shown. The time should display a total of 4 digits, one of which shows time to the tenth of a second. Note that Bauhaus font is being used.



Exercise 2

118

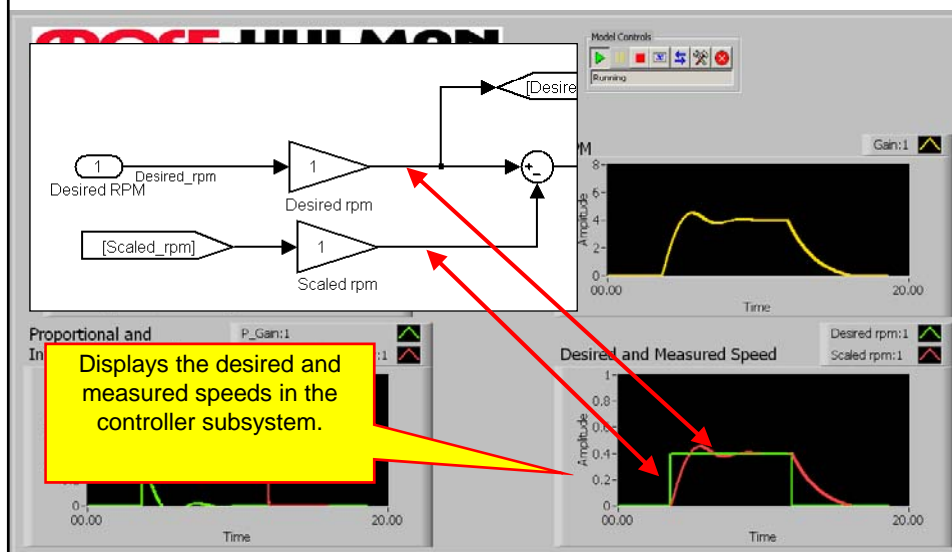
- Create the front panel shown below. The controls created earlier are unchanged. Added items are listed in the following few slides.





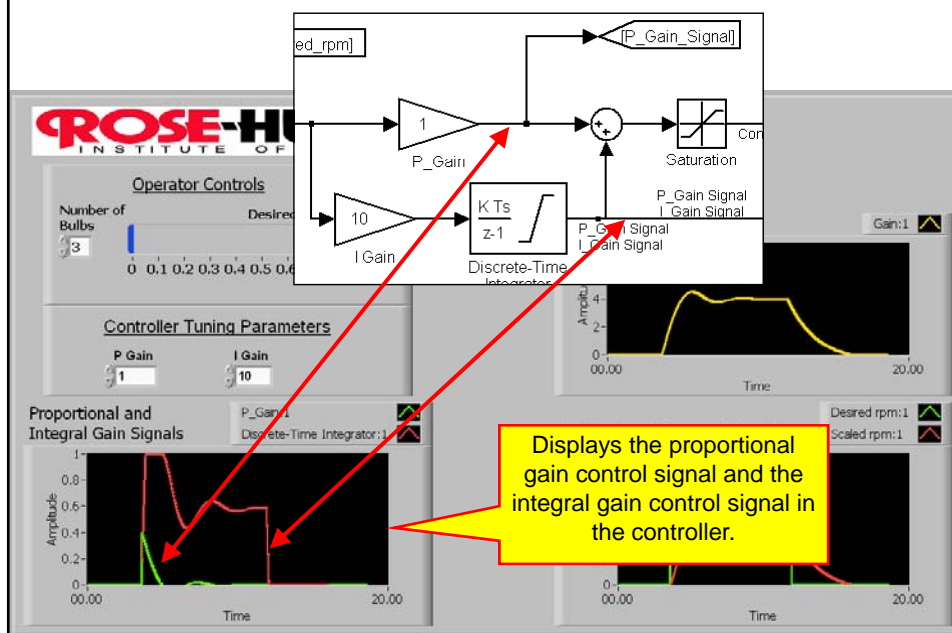
Exercise 2

119



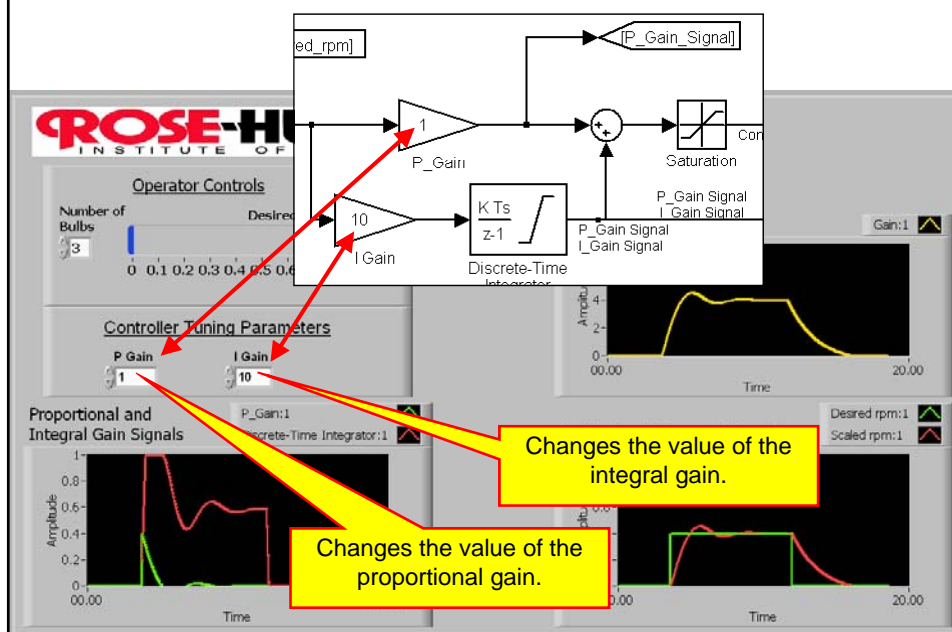
Exercise 2

120



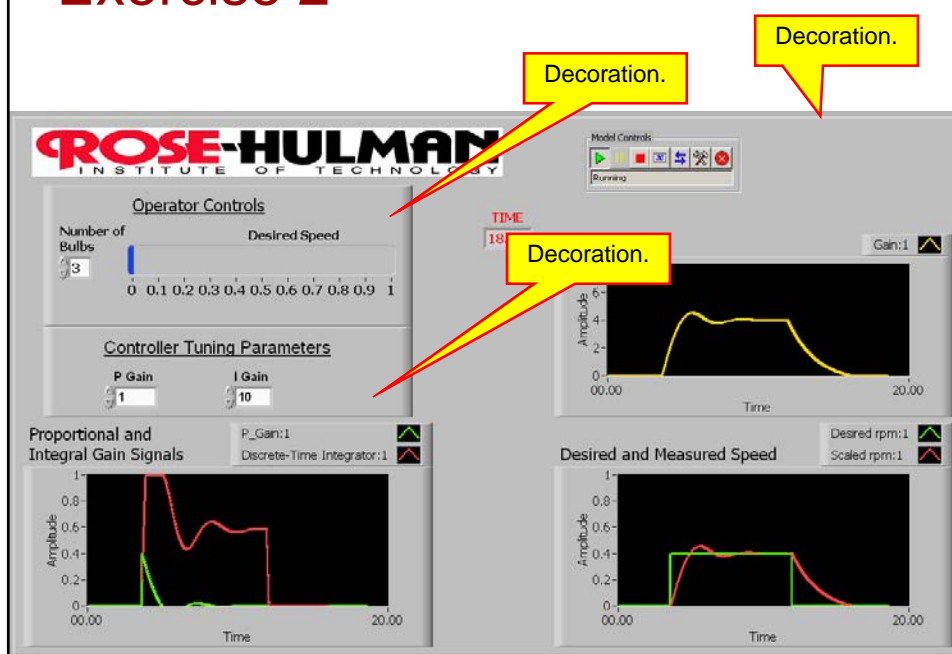
Exercise 2

121



Exercise 2

122



Exercise 2

123

Demo_____





Advanced Model-Based-System Design

Lecture 14: Real-Time Vehicle Model

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Models

2

- We now have a good model of the vehicle that runs in the Simulink environment.
- We know how to use the National Instruments Simulation Interface Toolkit to run Simulink models in real-time.
- Next, we will put the two together and run our vehicle model in real-time on the PXI platform and have a cool display to show vehicle performance.

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Foreshadowing

3

- We will need to make some modifications to the model.
- We will need to create the LabVIEW front panel from scratch.
- We will run into some numerical solver issues that will make the system behave badly.



Models

4

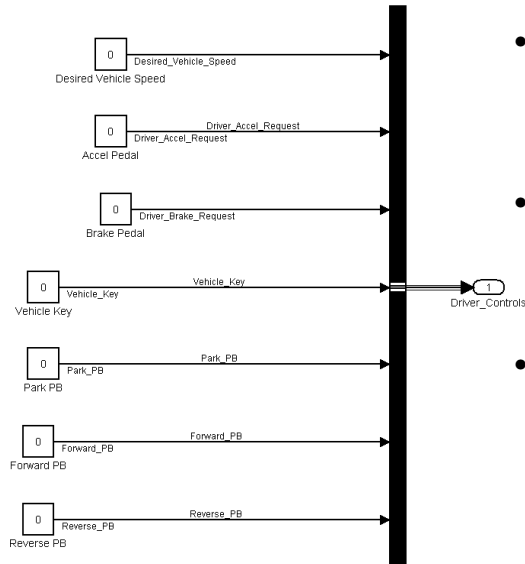
- We will start with model Lecture_14_Model0.mdl that will be passed out to the class.
- This is a Simulink model with push-buttons and slider gain blocks in the driver subsystem.
- The driver block can be used to drive the vehicle, but it is kind of clunky.
- We will replace the Simulink driver controls with LabVIEW controls to make the vehicle easier to drive.
- We will use the Display_and_Logging subsystem to monitor the model outputs in which we are interested.
- Save the model as Lecture14_Model1 and make the changes to the driver block as a show on the next slide.



Driver Block Modifications

5

- We have replaced all of the controls by constants.



- Note that the names of the constant blocks have been changed to the name of the signal in the bus.
- This makes the constants easy to identify when connecting signals with the LabVIEW SIT.
- Note that the constants have a value of 0. This sets the default value of the lv controls when the simulations starts.

Driver Block

6

- Even though we are setting the driver controls to a constant value of zero, we will be able to drive the vehicle because we will connect the constant blocks to LabVIEW controls.
- The LabVIEW controls will specify the value of the constants and thus change the driver input to the vehicle.

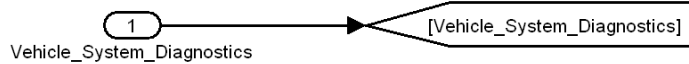
Display_and_Logging

7

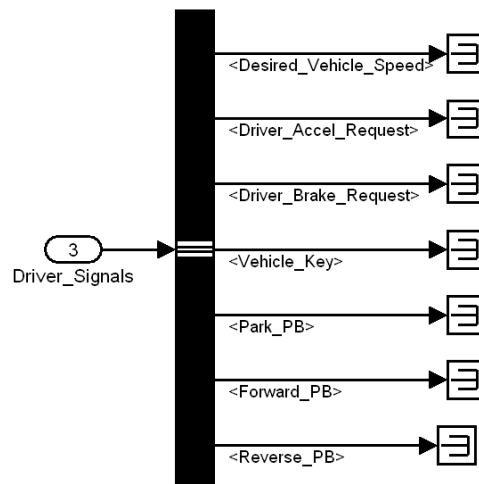
- The other subsystem that has to change is the Display_and_Logging subsystem.
- We need to:
 - Remove all scopes.
 - Delete all “To Workspace Blocks”
 - Terminate all outputs in which we have no interest.
 - Add gain blocks and signal probes to the signals that we wish to monitor.
- The next few slides show the modifications to the Logging_and_Display Subsystem.





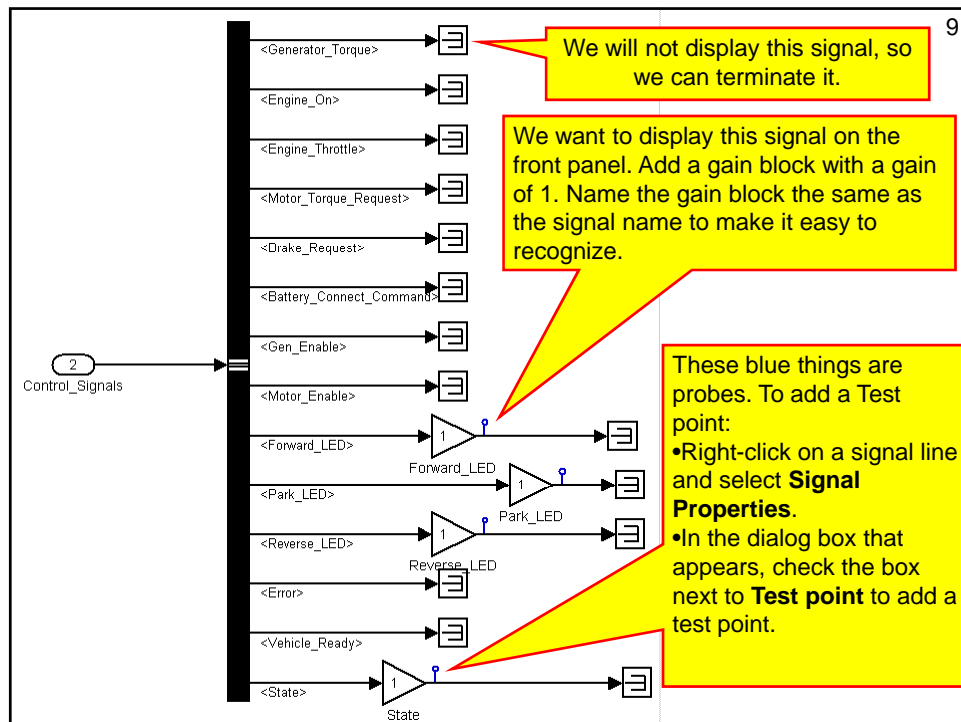


8





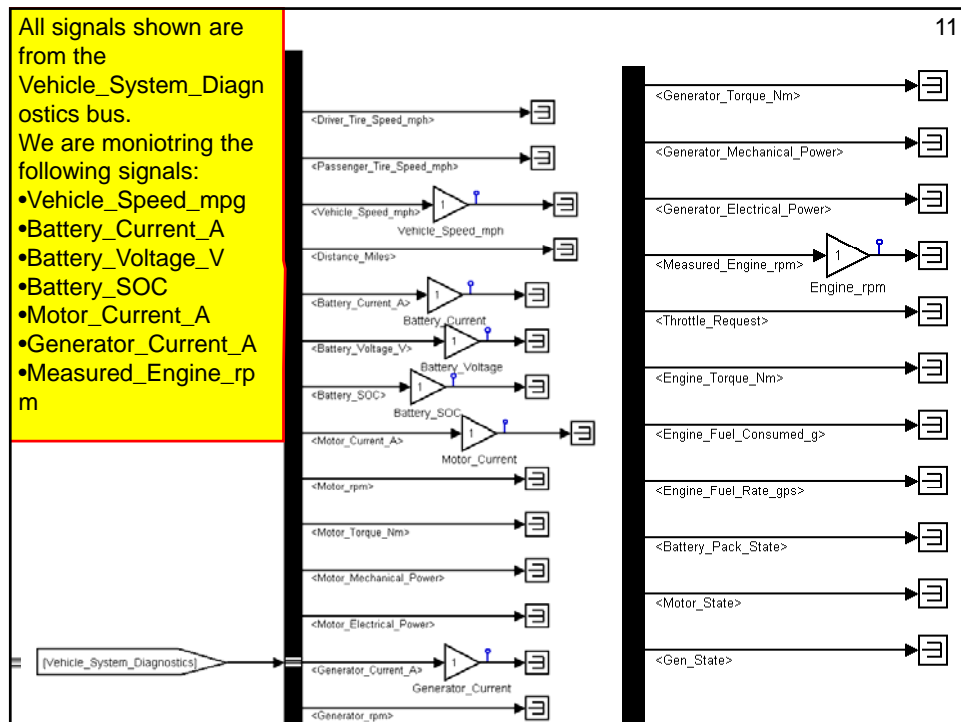




Logging and Display Block

10

- The only signals from the controller that we will be display are the forward, reverse, and park LED indicators.
- We will connect these signals to front panel indicators so show when the vehicle is actually in park, forward, or reverse.
- The State signal is displayed for debugging purposes so that we can tell which state we are in on the Startup_and_Shifting Stateflow chart.



Solver

12

- The last things we need to do are setup the simulation configuration parameters and:
 - Specify a fixed step solver type.
 - Pick a fixed step solver.
 - Choose the fixed step size.
 - In the Real-Time Workshop section, specify NIDLL.tlc as the System target file.

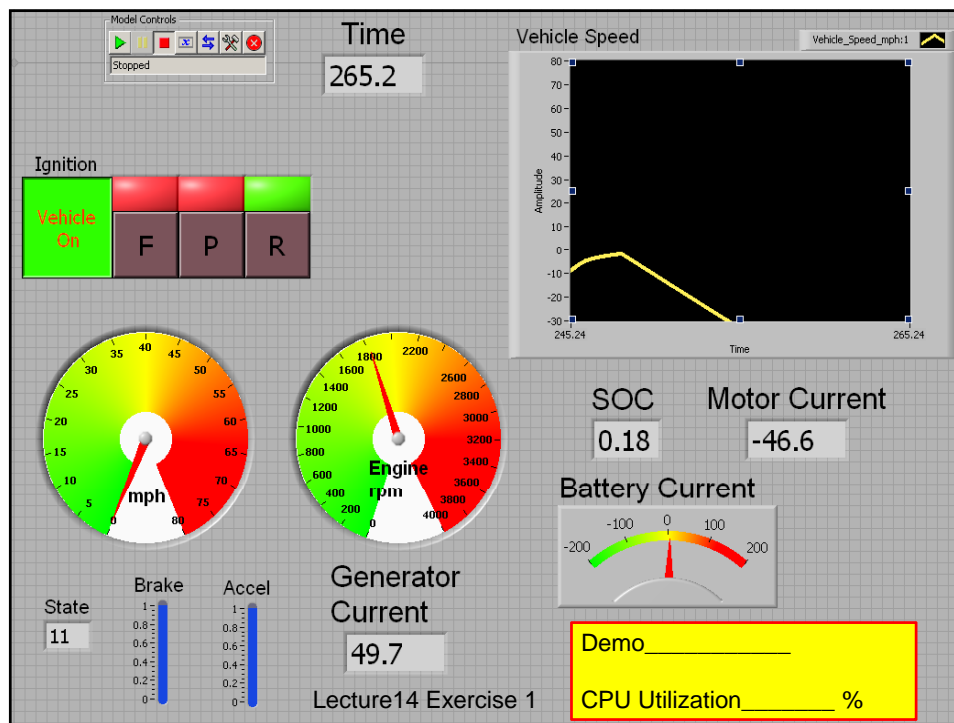
Lecture14 Exercise 1

13

- Build the dll for the model.
- Create the front panel shown on the next slide.
- Use National Instruments SIT to connect the front panel controls and indicators to the Simulink model.
- Determine the solver and fixed step size, so that the model runs in real-time.
- For real-time operation, the cpu processor utilization should be less than 70%.





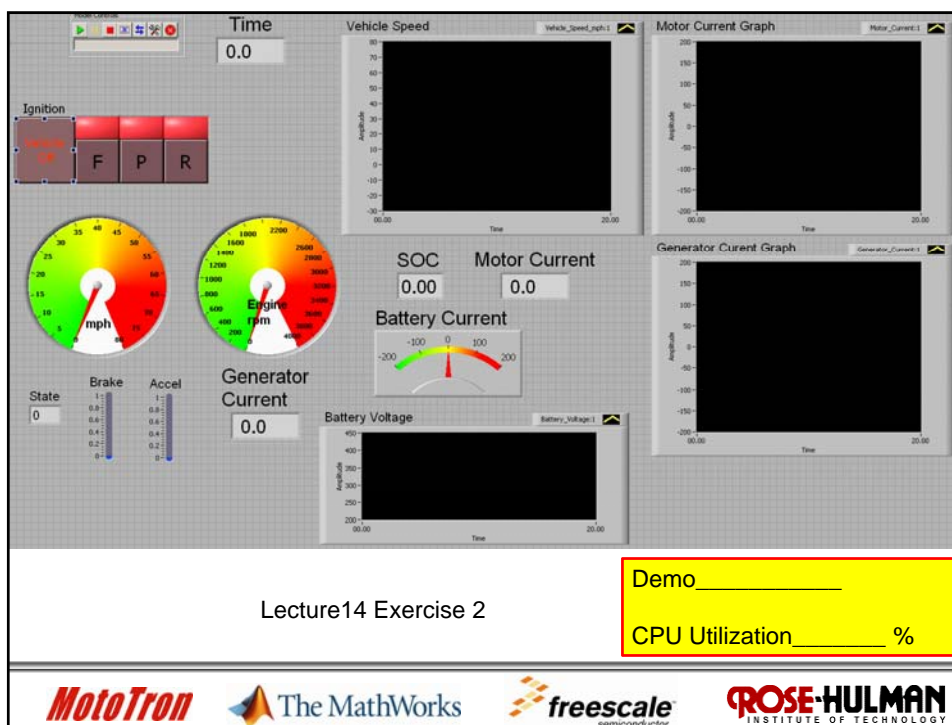




Lecture 14 Exercise 2

15

- We notice that the generator current bounces all around, independent of the step size and integration methods.
- To get a better idea of the system behavior, add the graphs to the front panel as shown next:





Model Problems

17

- We notice one obvious problem in that the motor current bounces all around even though we are asking for a constant acceleration.
- Occasionally the model goes wild and the vehicle speed goes to infinity and some of the displays show a value of NaN rather than a real numerical value. (NaN stands for Not a Number.)
- Both of these problems can be solved by using the ODE14x solver with a small time step.



Model Problems

18

- The problem with using a small time step is that the model must be executed once each time step. It does not matter what our time step is, the same calculations occur whether we use a time step of 0.1 or a time step of 0.001.
- The problem with a smaller time step is that all of the model calculations must be completed by the next time step.
- Complex models with small time steps will require a fast real-time computer to complete the required calculations in the specified time step.



Lecture 14 Exercise 3

19

- As a first fix, we will use the ODE14x solver and a time step of 0.001 seconds.
- The problem with this solution is that it may take too much CPU power to run the model.

Demo _____
 CPU Utilization _____ %



Model Problems

20

- The ODE14x solver was designed for stiff systems that require small time steps.
- Most of our model is pretty simple. Either we are integrating a slowly changing force with a large mass, or a slowly changing torque with a large inertia. In either case, the time scale is such that a small time step is not required.
- The elements in our model that require a small time step are the tires.
- The tires model tire slip. Under hard braking it is possible that the wheels will lock and the tires will skid.



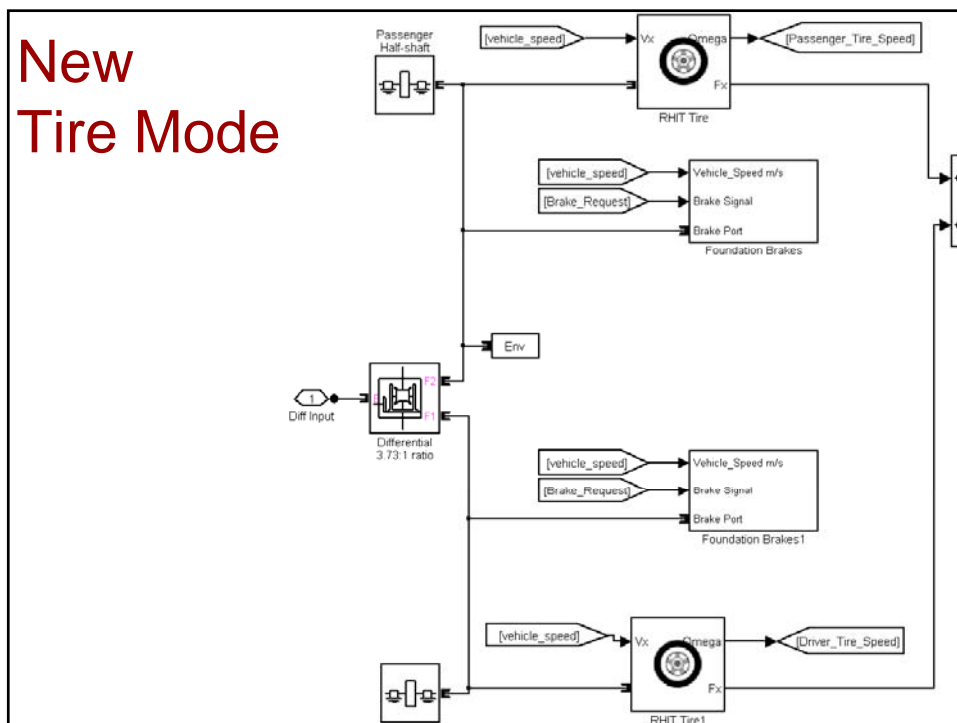


Model Problems

21

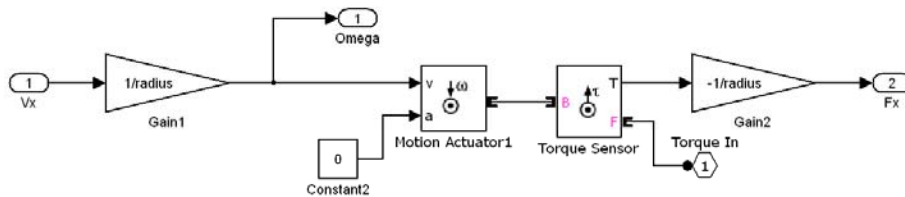
- We noticed under hard braking that occasionally the model blew up and the vehicle speed went to infinity.
- Determining the boundary between when the tire is skidding and rolling can take a very small time step.
- To get a small enough time step to simulate the tires, we go past 100% processor utilization on the real-time target and bad things happen.
- A fix is to replace the SimDriveline tire with a simplified tire model that has no slip.
- Create the tire model shown next and use it for both tires on the vehicle

New Tire Mode



New Tire Model

23



- Given an input torque, the model calculates the force applied to the vehicle by dividing by the radius of the tire.
- Given an input linear velocity of the vehicle, the model calculates the rotational velocity of the tire by dividing by the radius.
- This rotational velocity is then imposed on the driveline by the motion actuator.

New Tire Model

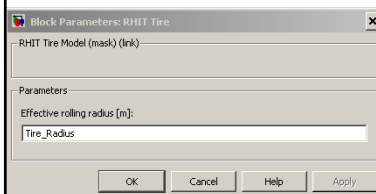
24

- This model has a problem in that it imposes a dynamic constraint on the drive line. ➔ It fixes the velocity of the drive line to which the tire is connected.
- If we have another block in the model that also fixes the speed of the same drive line, we will get an error that states that we have a redundant dynamic constraint. (We cannot have two blocks that specify the velocity of the same drive line.)

Lecture 14 Exercise 4a

25

- Create a subsystem for the new tire model.
- Create a mask for the subsystem that:
 - Specifies the tire radius as an input parameter to the model.
 - Displays the tire picture as shown.
 - Displays the name “RHIT Tire Model” rather than “Subsystem.”



Tire Masked Subsystem

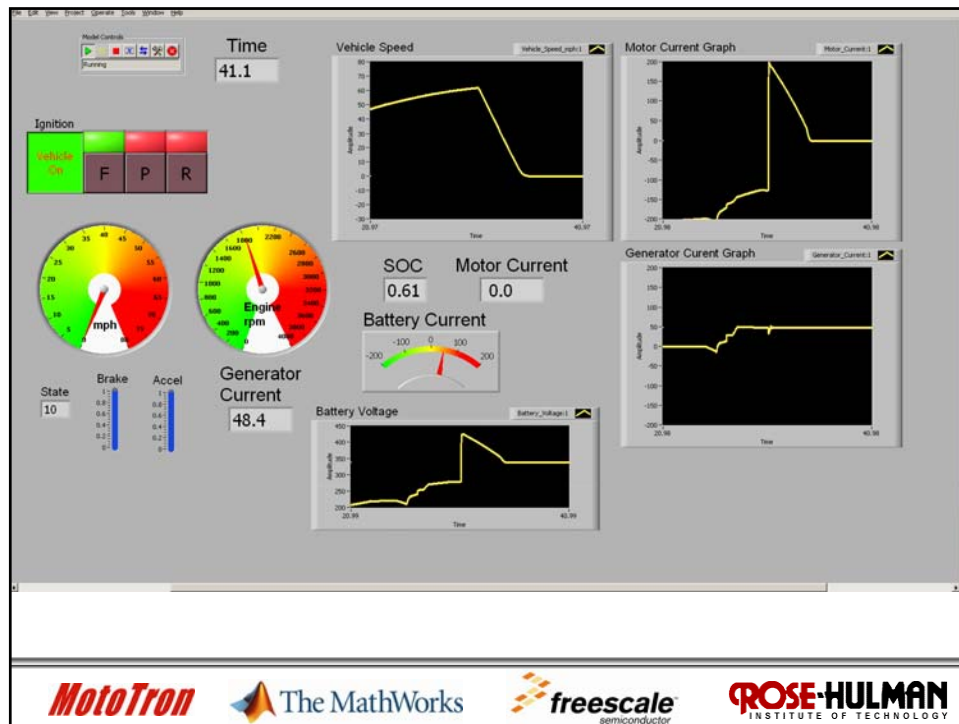
Lecture 14 Exercise 4b

26

- With this new tire model, the model should behave well using a large step size (0.01 or larger) and the ODE14x solver is not needed.
- We will notice
 - Low processor utilization.
 - The model does not blow up any more.
 - The motor current now behaves.
- A screen capture of this model running in real-time is shown on the next slide.

Demo_____

CPU Utilization_____ %



Model Debugging

28

- Now that we have the model running in real time, we can run the model and find out any other problems that it may have.
- One problem that we notice is that under hard acceleration, the motor will draw more than 50 amps.
- In our present scheme the generator only puts out 50 amps constant when it charges.
- Thus, under extended hard acceleration, we will discharge the battery completely leading to safety and lifetime issues.
- We need to modify our control scheme.



Lecture 14 Exercise 5

29

- Modify the control algorithm as follows:
 - Under normal conditions, the generator will charge the battery at 50 A.
 - If the battery SOC goes below a value of 0.58, a new algorithm kicks in where the battery charges at the motor current plus 20% with some averaging. (What should happen if regen braking kicks in and the motor current flips?)
 - The minimum charging current in this mode is 50 A.
 - This method continues until the battery is charged back up to 0.7.
 - Once the battery is charged to 0.7, the normal charging algorithm kicks back in.

Demo_____

CPU Utilization_____ %



Lecture 14 Exercise 6

30

- One of the modifications we made to the model is that we reduce the braking torque at vehicle speeds below 3 mph.
- As we drive the vehicle in real-time, this becomes very annoying because the vehicle takes a long time to slow down below 3 mph.
- We made this modification because we had numerical problems with the brakes at low speed.
- With the new tire model, the braking problem may no longer be an issue.
- Modify the brakes so that we can apply full braking torque at speeds of 0.25 mph and higher.
- Verify your design and prove that it works.

Demo_____

CPU Utilization_____ %



Except where otherwise noted, this work is licensed under

<http://creativecommons.org/licenses/by/3.0/>



Advanced Model-Based-System Design

Lecture 15: Number Systems

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Outline

2

- Binary
- Hexadecimal
- Matlab Functions
- Unsigned Integers
- Signed Integers
- Floating Point Numbers

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Number Systems

3

- There are two kinds of engineers in this world
 - Those who know binary and those who don't.
 - That was a joke.
 - If you don't know binary, you probably didn't get it.
- This section is for the engineers that didn't get it.



Base 10

4

- Most of us are familiar with base 10 number systems.
- Valid digits are 0 through 9 (Hey! There are 10 values!)
- The base is also referred to as the radix.
- An example is:

$$7384 = 7 \times 10^3 + 3 \times 10^2 + 8 \times 10^1 + 4 \times 10^0$$

Radix = 10

Radix = 10





Binary

5

- Binary uses a radix of 2.
- Valid values of a digit are 0 and 1.

$$10110 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

Radix = 2

$$10110 = (1 \times 16) + (0 \times 8) + (1 \times 4) + (1 \times 2) + 0 = 22$$

MotoTron

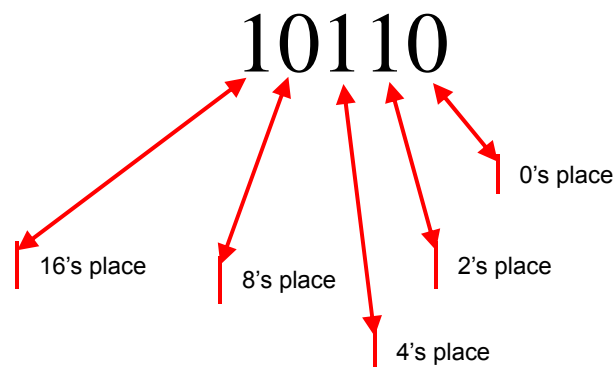
 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Binary

6



MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Binary

7

- Historically and physically our choice of 0 and 1 for a binary digit comes from:
 - Switches which can be on or off.
 - Digital logic circuits that produce either a low voltage or a high voltage.
 - Typical 5 V logic circuits
 - Low = logic 0 → Voltage from 0 to 0.8 Volts.
 - High = Logic 1 → Voltage from 3.4 to 5 Volts.
- Synonyms
 - 1 = logic 1 = “high” = “True”
 - 0 = logic 0 = “low” = “False”



Terminology

8

- A single binary digit is referred to as a bit.
- A group of 4 binary digits is referred to as a nibble. (1011 1110) is two nibbles.
- A group of 8 binary digits is referred to as a byte (10111110) is one byte.
- 1k (for digital guys) is $2^{10} = 1024$
- 1M (for digital guys) is $1k * 1k = 2^{10} * 2^{10} = 1048576$.





Hexadecimal – Radix = 16

9

- We will be dealing with long strings of bits.
- It is convenient to group those bits in groups of 4.

Binary	Hex	Decimal	Binary	Hex	Decimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15



Hexadecimal

10

- In decimal every digit can have ten values, 0 through 9.
- In hexadecimal each digit can have 16 values ranging from 0 to 15.
- Hey, we need a single symbol for each digit!
- How do we do this with only 10 numeric symbols in our mathematical vernacular.
 - For numbers 0 through 9, use 0 through 9.
 - For numbers 10 through 15, use letters A through F.



Hexadecimal

11

$$\begin{array}{ccccc} \underbrace{1011}_B & \underbrace{0110}_6 & \underbrace{1101}_D & \underbrace{0111}_7 & \underbrace{0011}_3 \end{array}$$

$$B6D73 = (11 \times 16^4) + (6 \times 16^3) + (13 \times 16^2) + (7 \times 16^1) + (3 \times 16^0)$$

$$10110110110101110011_2 = B6D73_{16} = 748915_{10}$$



Useful Matlab Functions

12

- Bin2dec – Converts a binary text string to a decimal number:

```
>> bin2dec('10110110110101110011')
ans =
    748915
```

- Dec2bin – Converts a decimal number to a binary text string.

```
>> dec2bin(748915)
ans =
10110110110101110011
```





Useful Matlab Functions

13

- Hex2dec – Converts a hexadecimal string to a decimal number:

```
>> hex2dec('B6D73')  
ans =  
748915
```

- Dec2hex – Converts a decimal number to a hexadecimal text string.

```
>> dec2hex(748915)  
ans =  
B6D73
```



Matlab

14

- How do we convert from binary to hex?

```
>> dec2hex(bin2dec('10110110110101110011'))  
ans =  
B6D73
```

- How do we convert from hex to binary?

```
>> dec2bin(hex2dec('B6D73'))  
ans =  
10110110110101110011
```





15

Hexadecimal Numbers

- If we see a number like 123, how do we know if it is a hexadecimal or decimal number? (It could actually be any base greater than 3, but we won't go there.)
- Ways of indicating a number is a hexadecimal number
 - hex 123 - saying it.
 - \$123 - preceding the number with a \$ sign.
 - x123 - preceding the number with an x which is short for "hex."
 - 123₁₆ – Indicating the base explicitly.

16

Basic Data Types in Simulink

- Boolean – True or False (not 0 or 1 numerically)
- Uint8 – Unsigned 8-bit integer. Can represent values from 0 to 255.
 - $\square 11111111_2$
 - $= 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$
 - $= 2^8 - 1 = 255_{10}$
- Uint16 – Unsigned 16-bit integer. Can represent values from 0 to 65535.
 - $\square 1111111111111111_2$
 - $= 2^{15} + 2^{14} + 2^{13} + \dots + 2^2 + 2^1 + 2^0$
 - $= 2^{16} - 1 = 65535_{10}$





Basic Data Types in Simulink

17

- Uint32 – Unsigned 32-bit integer. Can represent values from 0 to 4294967295.

$$\begin{aligned} & \square 11111111111111111111111111111111_2 \\ & = 2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 2^1 + 2^0 \\ & = 2^{32} - 1 = 4294967295_{10} \end{aligned}$$



Signed Integers

18

- There are three common ways of representing signed numbers
 - Sign and magnitude: The most significant bit represents the sign. (1 is negative, 0 is positive)
 - 10001 would represent the number -1.
 - 11111 would represent the number -15.
 - 00001 would represent the number 1.
 - 01111 would represent the number +15.





19

Signed Integers Sign and Magnitude

- With sign and magnitude representation:
 - There are an equal number of positive and negative values that can be represented.
 - There are two ways to represent 0:
 - 1000000
 - 0000000
- We will not be using this method to represent signed integers.



20

Signed Integers Biased Values

- With biased values, to calculate the numerical value of the code, calculate the magnitude of the code and then subtract off a fixed bias.
- Example: 5-bit codes, bias = 15.
 - 00000 \rightarrow value = $0 - 15 = -15$
 - 00001 \rightarrow value = $1 - 15 = -14$
 - 01111 \rightarrow value = $15 - 15 = 0$
 - 10000 \rightarrow value = $16 - 15 = 1$
 - 11111 \rightarrow value = $31 - 15 = 16$





Signed Integers – 2's Complement 21

- We will be using a method called two's complement to represent positive and negative integers.
- With 2's complement, the most significant bit has a negative weight.

Note this (-) sign.

$$\begin{aligned}
 10110 &= (-1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) \\
 &= -16 + 6 \\
 &= -10
 \end{aligned}$$



2's complement Numbers 22

$$\begin{aligned}
 00110 &= (-0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) \\
 &= -0 + 6 \\
 &= 6
 \end{aligned}$$

- With 2's complement
 - If the most significant bit is a 1, the number is negative.
 - If the most significant bit is a 0, the number will be positive.





2's complement Numbers

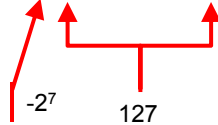
23

- We will do some 8-bit examples.
- The most negative number is
- The most positive number is
- The code for -1 is

$$10000000 = -2^7 = -128$$

$$01111111 = 127$$

$$11111111 = -2^7 + 127$$



MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

2's complement

24

- There is only one representation for 0
– 00000000

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



25

Signed Integer Types in Simulink

- Int8 – 2's complement signed 8-bit integer. Can represent values from -128 to 127.
 $10000000 = -2^7 = -128$
 $01111111 = 127$
- Int16 – 2's complement signed 16-bit integer. Can represent values from -32768 to 32767.
 $1000000000000000 = -2^{15} = -32768$
 $0111111111111111 = 32767$
- Int32 – 2's complement signed 32-bit integer. Represents values from -2147483648 to 2147483647.
 - $10000000000000000000000000000000 = -2^{31} = -2147483648$
 - $01111111111111111111111111111111 = 2147483647$



26

Floating Point Numbers

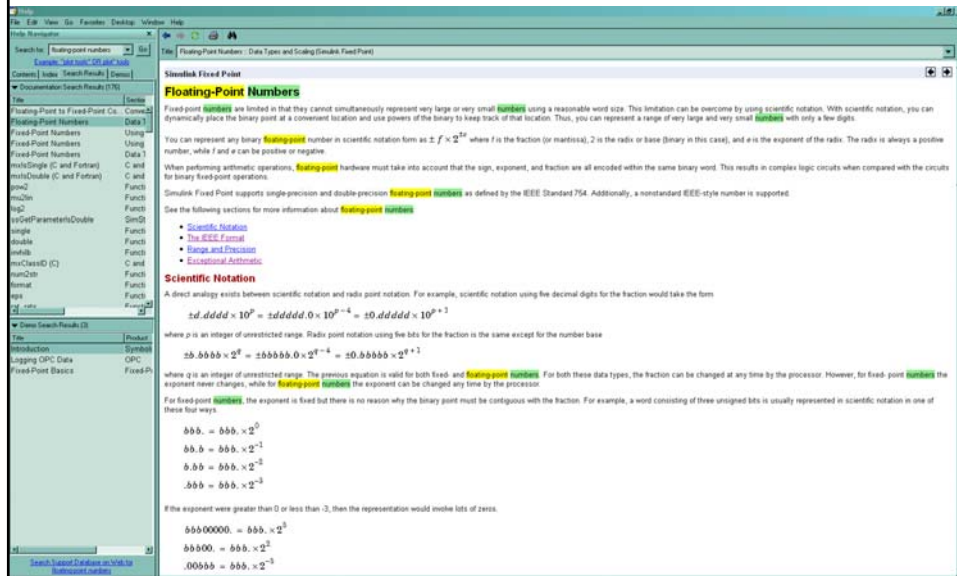
- The MathWorks help facility has a good section on floating point numbers.
- The following few slides were generated from the information contained in the MathWorks help facility.
- Search for the topic, "floating-point numbers" in the MathWorks help facility to find more in-depth information.





Floating Point Numbers

27



Floating Point Numbers

28

- Fixed-point numbers are limited in that they cannot simultaneously represent very large or very small numbers using a reasonable word size.
- This limitation can be overcome by using scientific notation.
- With scientific notation, you can dynamically place the binary point at a convenient location and use powers of the binary to keep track of that location.



Scientific Notation (Decimal)

29

- Most of us are familiar with scientific notation.
 - d is a decimal digit with values from 0 to 9.
 - We can move the decimal point right or left by decreasing or increasing the power of 10 by which we multiply.

$$\begin{aligned}\pm d.dddd \times 10^p &= \pm dddd.d \times 10^{p-4} \\ &= \pm 0.ddddd \times 10^{p+1}\end{aligned}$$

Decimal point.



Binary Point

30

- Binary numbers can have a fractional part just like decimal numbers:

$$73.84 = 7 \times 10^1 + 3 \times 10^0 + 8 \times 10^{-1} + 4 \times 10^{-2}$$

Decimal point.

$$101.11 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$$

$$101.11 = (1 \times 4) + (0 \times 2) + (1 \times 1) + (1 \times \frac{1}{2}) + (1 \times \frac{1}{4}) = 5.75$$

Binary point.





Radix Point Notation (Binary)

31

- Radix point notation is similar. Here we show radix notation for binary (radix = 2).
 - b is a binary digit with values of 0 or 1.
 - We can move the binary point right or left by decreasing or increasing the power of 2 by which we multiply.

$$\begin{aligned} \pm b.bbbb \times 2^p &= \pm bbbbbb.0 \times 2^{p-4} \\ &= \pm 0.bbbbb \times 2^{p+1} \end{aligned}$$

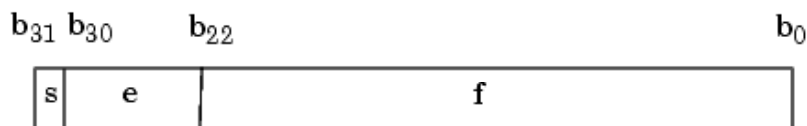
Binary point.



IEEE Floating Point Standard 754

32

- Single Precision – 32 bits



$$\text{Value} = \begin{cases} (-1)^s \cdot (2^{e-127}) \cdot (1.f) & ; \text{normalised}, 0 \leq e \leq 255, f \geq 0 \\ (-1)^s \cdot (2^{e-126}) \cdot (0.f) & ; \text{denormalised}, e = 0, f > 0 \\ \text{exceptional value} & \end{cases}$$

- Exceptional values: NaN, inf.





32-bit Floating point example

33

- $\underbrace{1}_{s} \underbrace{10111101}_{e} \underbrace{10100000000000000000000000000000}_{f}$
- $s = 1 \rightarrow$ we have a negative number.
- $e = 10111101 = 189_{10}$
- $\rightarrow (2^{e-127}) = (2^{189-127}) = (2^{62})$
- $f = 10100000000000000000000000000000$
- $1.f = 1.10100000000000000000000000000000$
 $= (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) + (0 \times 2^{-4}) + (0 \times 2^{-5}) + \dots$
 $= 1.625$



32-bit Floating Point Example

34

- Our number is
- $-1.625_{10} + 2^{62}$
- $= -7.493989779944505 \times 10^{18}$ (decimal)



IEEE Floating Point Standard 754

35

- Double Precision – 64 bits



$$\text{Value} = \begin{cases} (-1)^s \cdot (2^{e-1023}) \cdot (1.f) & ; \text{normalised}, 0 \leq e \leq 2047, f \geq 0 \\ (-1)^s \cdot (2^{e-1022}) \cdot (0.f) & ; \text{denormalised}, e = 0, f > 0 \\ \text{exceptional value} & \end{cases}$$

- Exceptional values: NaN, inf.



Floating Point Numbers

36

Data Type	Low Limit	High Limit	Exponent Bias	Precision
Single	$2^{-126} \approx 10^{-38}$	$2^{128} \approx 3 \times 10^{38}$	127	$2^{-23} \approx 10^{-7}$
Double	$2^{-1022} \approx 2 \times 10^{-308}$	$2^{1024} \approx 2 \times 10^{308}$	1023	$2^{-52} \approx 10^{-16}$

- Inf - Defined as those values outside the range of representable numbers.
- Any arithmetic operation involving Inf yields Inf.





Floating Point Numbers

37

- NaN – Not a number.
- There are two types of NaN:
 - A signaling NaN signals an invalid operation exception.
 - A quiet NaN propagates through almost every arithmetic operation without signaling an exception.
- The following operations result in a NaN:

$$\infty - \infty$$

$$\infty + \infty$$

$$0 \times \infty$$

$$0/0$$

$$\infty / \infty$$

Questions?





Advanced Model-Based-System Design

Lecture 16: Introduction to MotoTron ECUs and MotoHawk



The MathWorks



freescale
semiconductor



2

Outline

- Physical Connections
- Creating a MotoTron MotoHawk Project
- Flashing LED Project (Digital Output)
- Changing CAN Speed
- Motor Speed Controller
 - Analog Input
 - Digital Output
- Using MotoTune
 - MotoHawk Probes
 - MotoHawk Calibration



The MathWorks



freescale
semiconductor



3

Connecting MotoTron Hardware

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

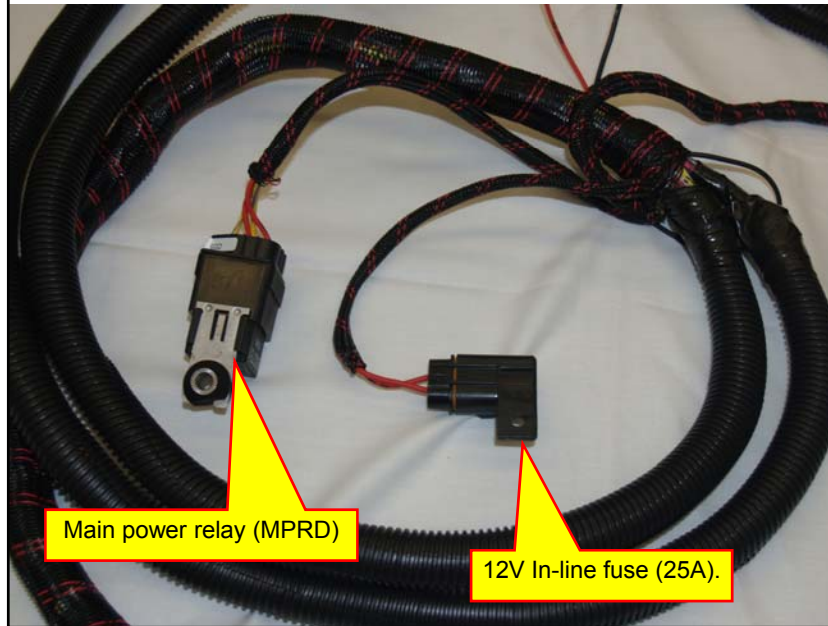
4

80-Pin Development Cable



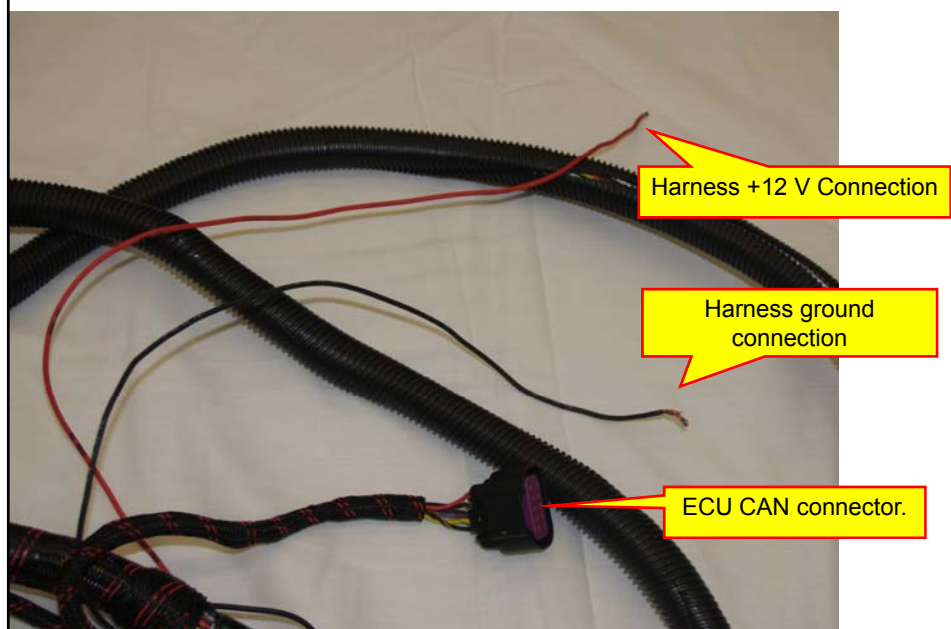
80-Pin Development Cable

5



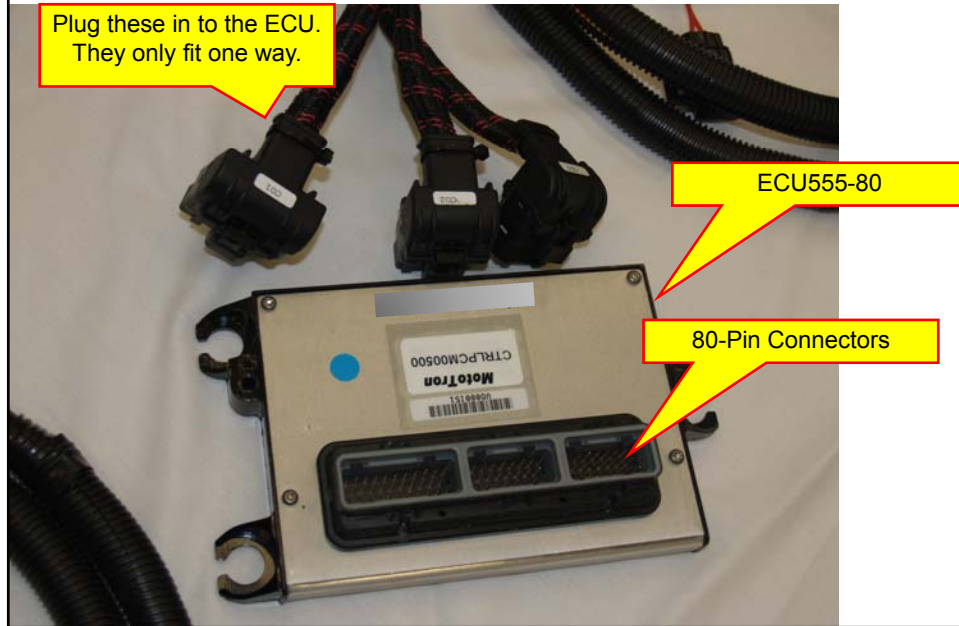
80-Pin Development Cable

6



ECU555-80

7



ECU555-80 Connections

8



12V Power Connection

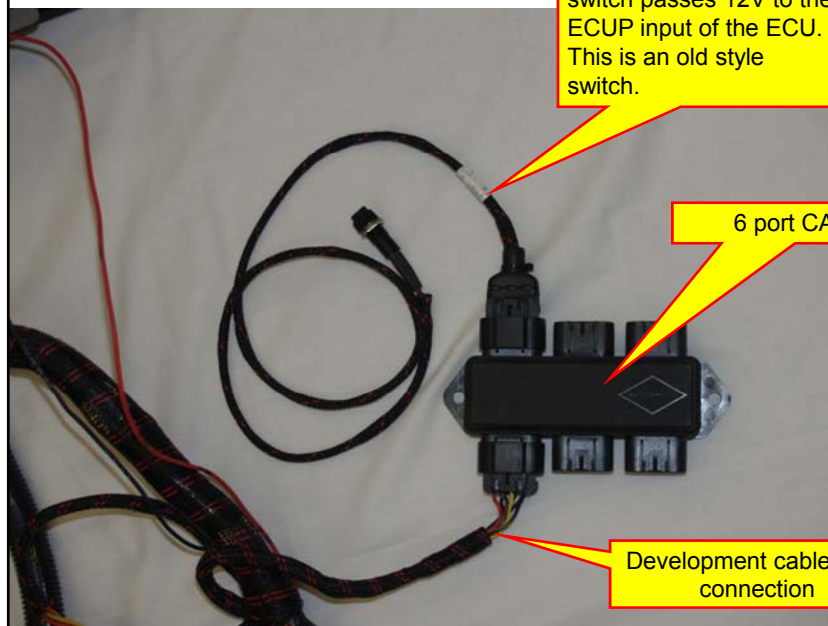
9



Connect the power connectors. For our application we only need a 1 A 12V power supply.

CAN Hub

10



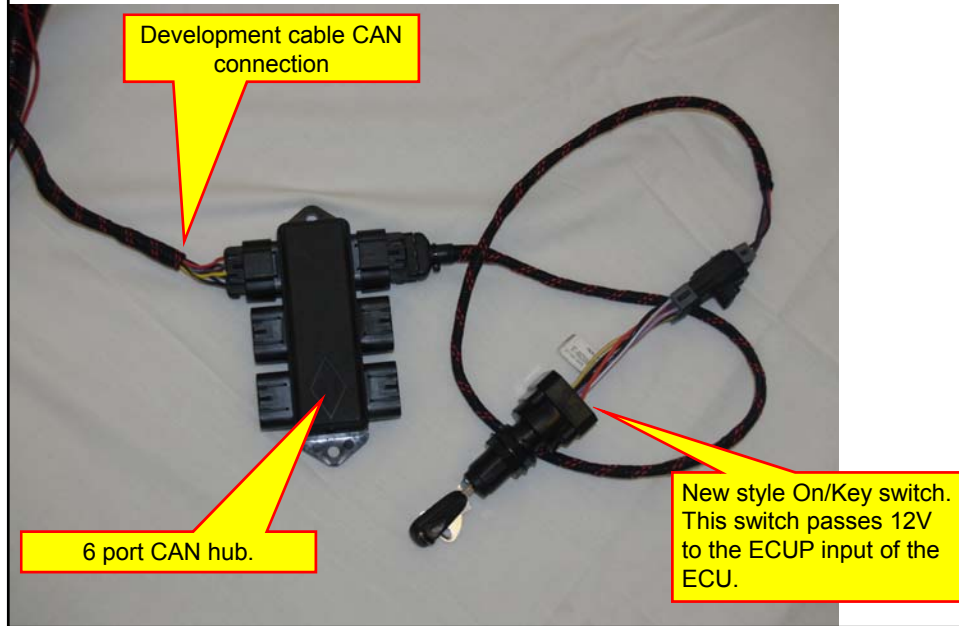
On/Key switch. This switch passes 12V to the ECUP input of the ECU. This is an old style switch.

6 port CAN hub.

Development cable CAN connection

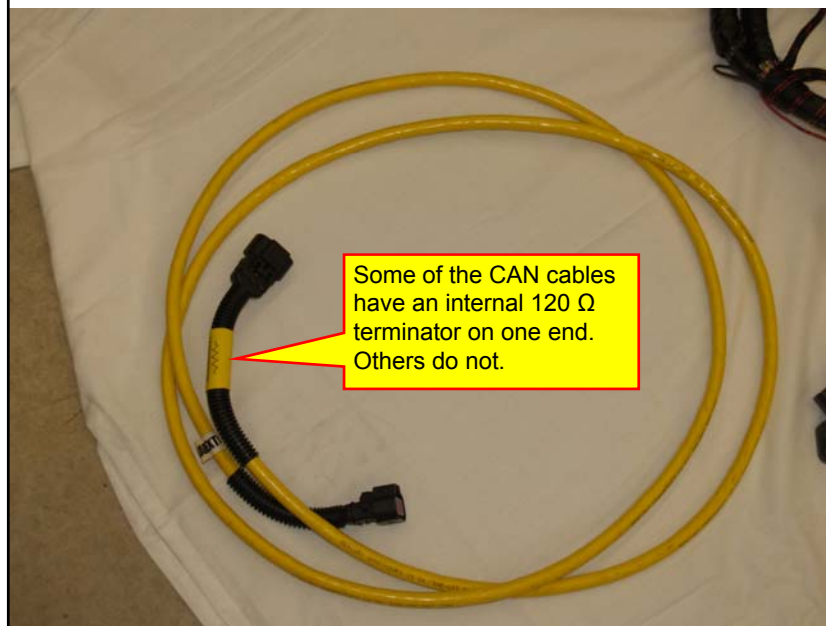
CAN Hub

11



MotoTron Yellow CAN Cables

12



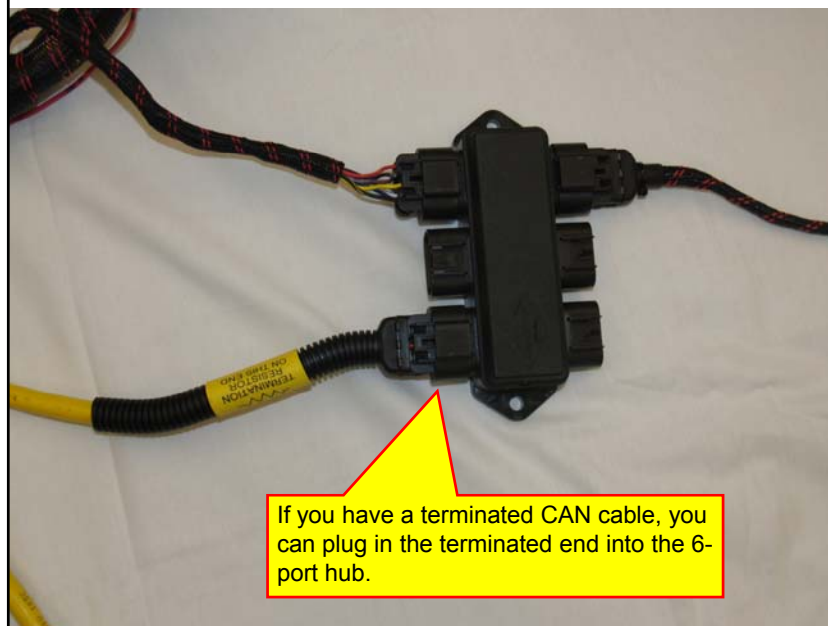
MotoTron Yellow CAN Cables

13



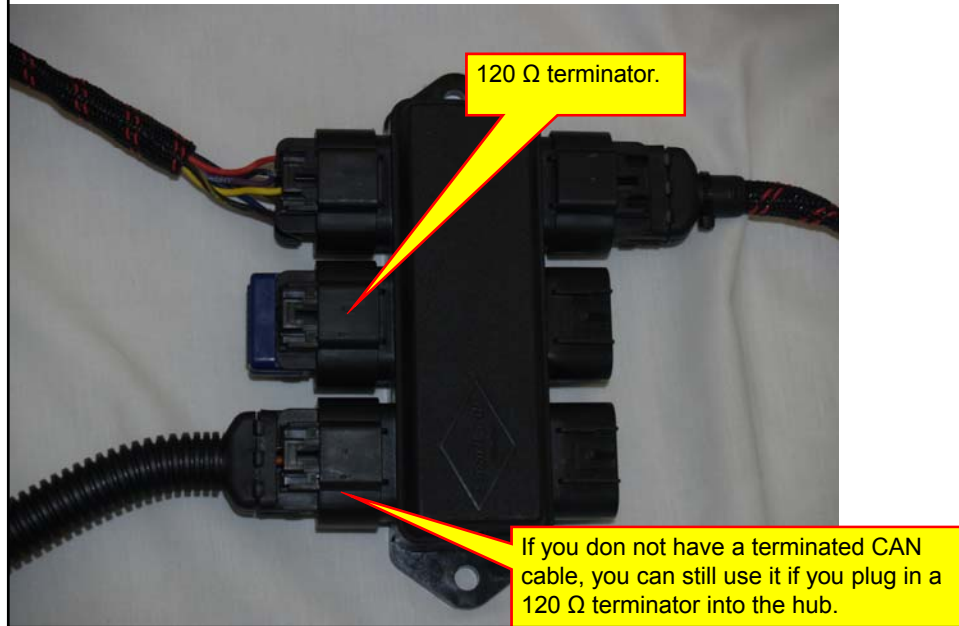
Can Connections

14



Can Connections

15



120 Ω Terminator

16



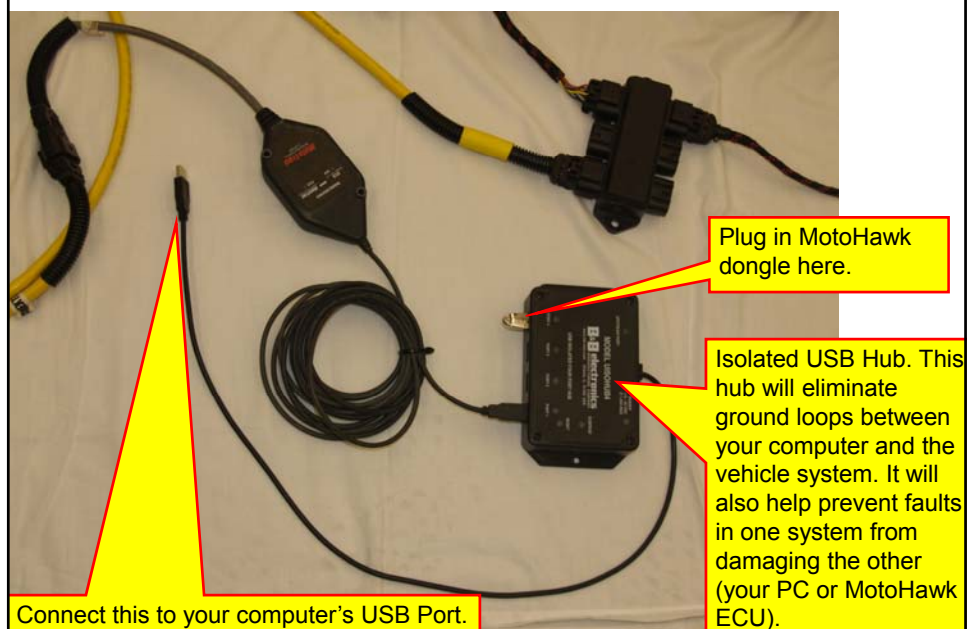
Kavaser CAN to USB

17



Isolated USB Hub

18





Complete Setup

19



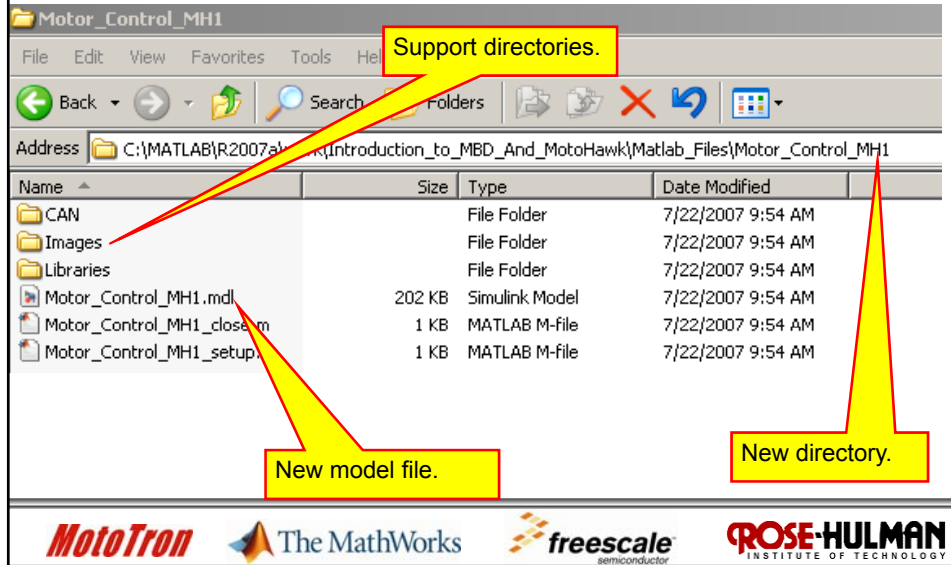
Creating a MotoHawk Project

20

- Run Matlab.
- Change to your working directory.
- At the Matlab prompt enter the command
 - `motohawk_project('Motor_Control_MH1')`
- This will:
 - Create a directory called `Motor_Control_MH1`
 - Change to that directory.
 - Create support sub directories.
 - Create a model file called `Motor_Control_MH1.mdl`
 - Open the model named `Motor_Control_MH1.mdl`

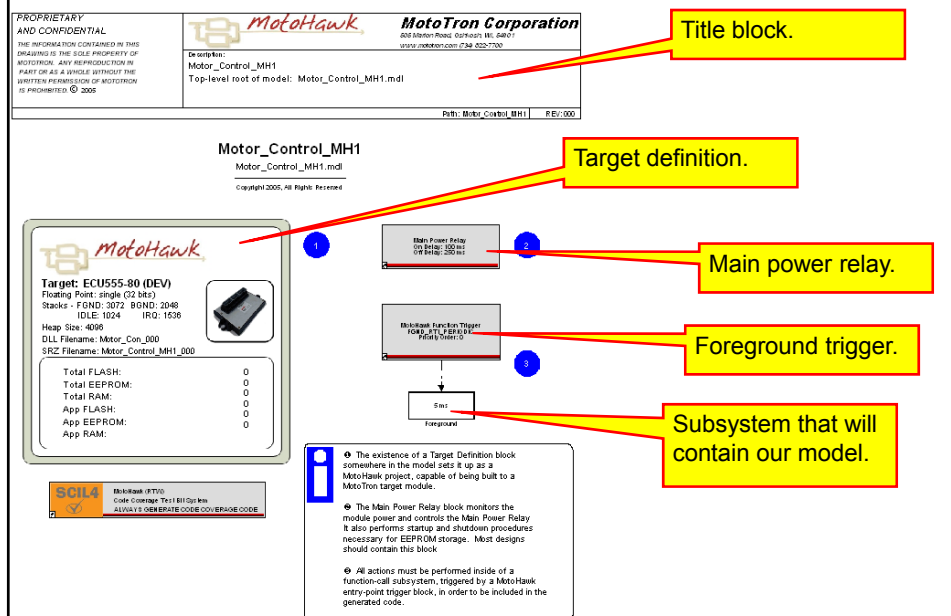
Directory Structure

21




New Simulink Model Created

22







Title Block

23


PROPRIETARY AND CONFIDENTIAL <small>THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF MOTOTRON. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MOTOTRON IS PROHIBITED. © 2005</small>	 MotoTron Corporation <small>505 Marion Road, Oshkosh, WI, 54901 www.mototron.com (734) 822-7700</small>	Description: Motor_Control_MH1 Top-level root of model: Motor_Control_MH1.mdl
		<small>Path: Motor_Control_MH1</small> REV-000

Revision number. This number will increment every time we build this model (automatically generate code).


   

MotoHawk Target Definition Block

24



Target: ECU555-80 (DEV)
Floating Point: single (32 bits)
Stacks - FGND: 3072 BGND: 2048
IDLE: 1024 IRQ: 1536
Heap Size: 4096
DLL Filename: Motor_Con_000
SRZ Filename: Motor_Control_MH1_000







Total FLASH:	0
Total EEPROM:	0
Total RAM:	0
App FLASH:	0
App EEPROM:	0
App RAM:	0

Selected ECU

Double-click on this block to change the ECU.

These numbers will fill in when we build the project.



25

Block Parameters: motohawk_target_def1

MotoHawk Target Definition (mask) (link)

This block must be located somewhere in the code-generation model.

Choose the target MotoHawk hardware.

The heap size must be a multiple of 2K, and must be at least 2K.

Each of the stack sizes must be a multiple of 8 bytes.

The DLL Name and SRZ Name are optional. If unspecified, the model name is used for both fields. These fields must not contain the file extension (i.e. .dll or .srz).

The Build Directory is optional. If empty, the current directory will be used. If specified, it may be an absolute path, or be relative to the current model location by starting with ".".

Copyright 2005 MotoTron Corp. All Rights Reserved.

Parameters:

Target: ECU555-80

Memory Layout: DEV

Floating Point Type: single (32 bits)

Foreground Stack Size (Time-Based): 3*1024

Foreground Stack Size (Angle-Based): 0

Background Stack Size: 2*1024

Idle Stack Size: 1024

Interrupt Stack Size: 1536

Heap Size: 4*1024

☐ Show Explicit Target Filenames

OK Cancel Help Apply

We will be using a ECU555-80 target for this example. If you have a different target, specify it here.

Click the OK button after making any changes. You will return to the Simulink model.

26

MotoHawk Blocks

Main Power Relay
On Delay: 100 ms
Off Delay: 250 ms

MotoHawk Function Trigger
FGND_RTI_PERIODIC
Priority Order: 0

5 ms
Foreground

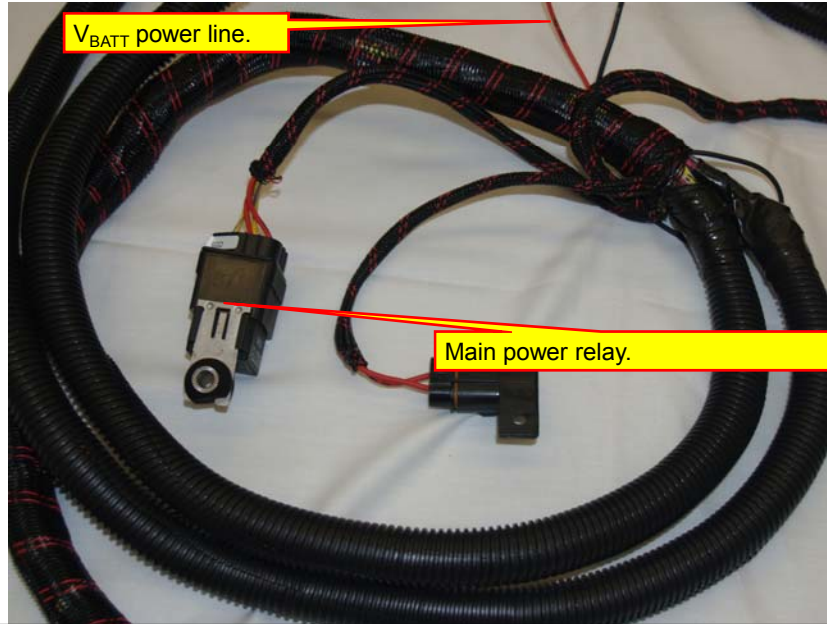
This block controls the main power relay. (See picture on next slide.) The main power relay will close 100 ms after power is applied to the ECUP line of the ECU. The relay will open 250 ms after 12 V is removed from the ECUP signal. See next slide for more information.

Foreground real-time interrupt trigger. By default, this block generates a trigger signal every 5 ms. We will show how to change the time later.

Triggered subsystem. We will put our controller in here. Our controller will execute once every 5 ms.

MotoHawk Development Harness

27



ECU555-80 Data Sheet

28

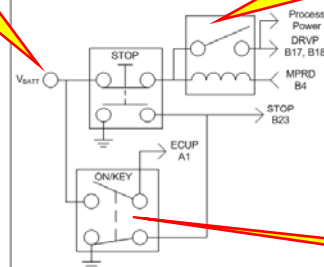
1 Input Signal Conditioning

1.1 ECUP (A1), DRVP (B17, B18), DRVG (C15, C16, C24)

Power (Key) switch input ECUP supplies module power to initiate the process under control. The DRVP inputs are wired to the Main Power Relay which will supply the process while the MPRD signal is asserted and will turn the module power on after the ECUP signal is released until the MPRD signal is released by the process (see below). Inputs are monitored by the

Main power relay. Closes 100 ms after key switch closes. Opens 250 ms after key switch opens. These delays allow for graceful starting and shutdown procedures.

Red wire in development harness.



Key switch.

The DRVG inputs are the system (battery) ground connections.



Simulation Configuration Parameters

29

- We have one last item to look at before we start creating our model.
- From the Simulink menus, select **Simulation** and then **Configuration Parameters**.
- You will see the following dialog box:

MotoTron

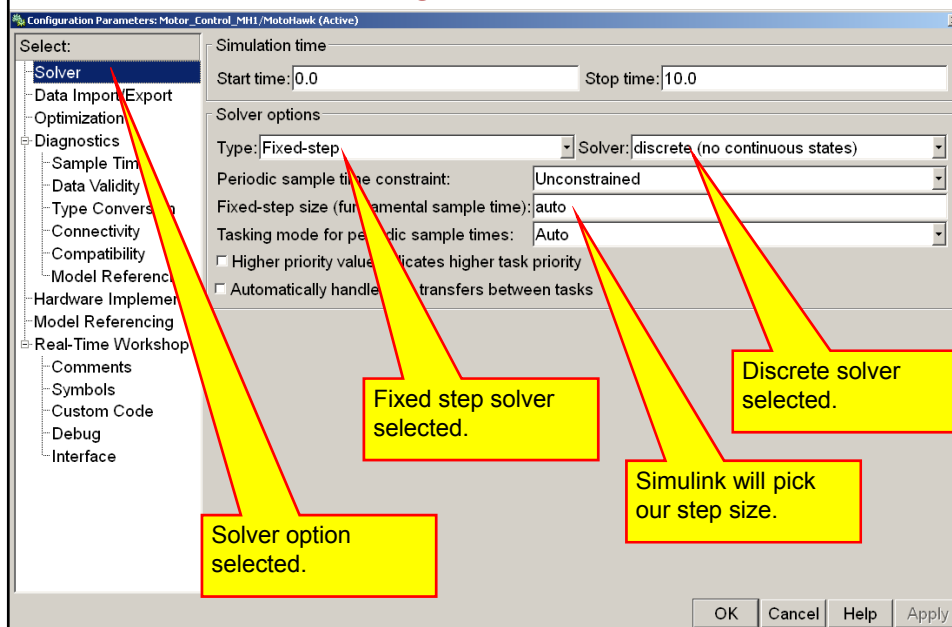
The MathWorks

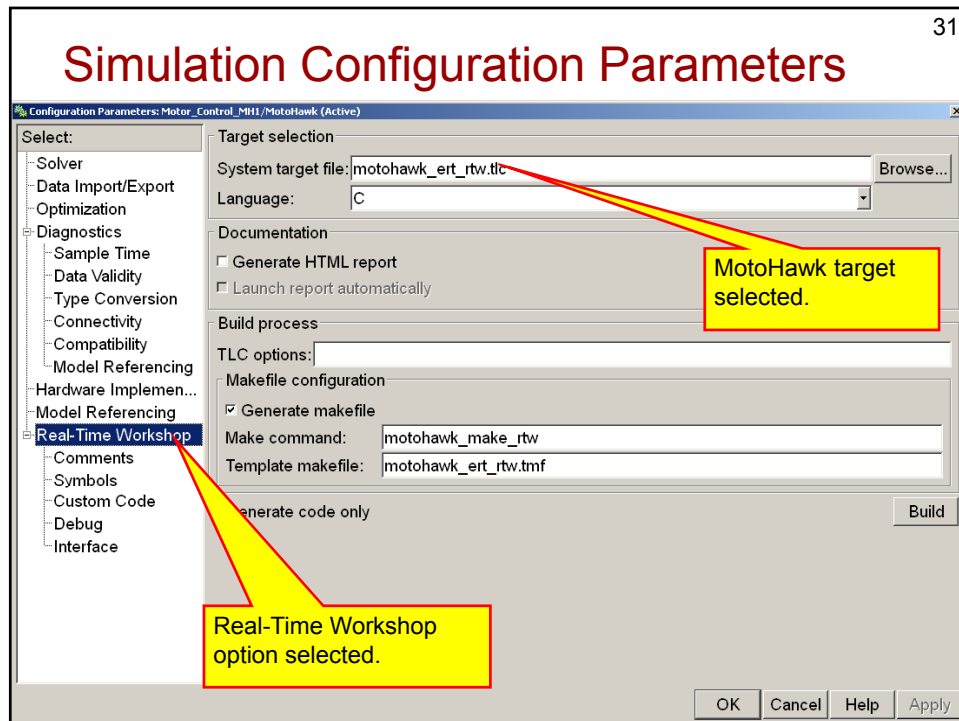
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Simulation Configuration Parameters

30





32

Simulation Configuration Parameters

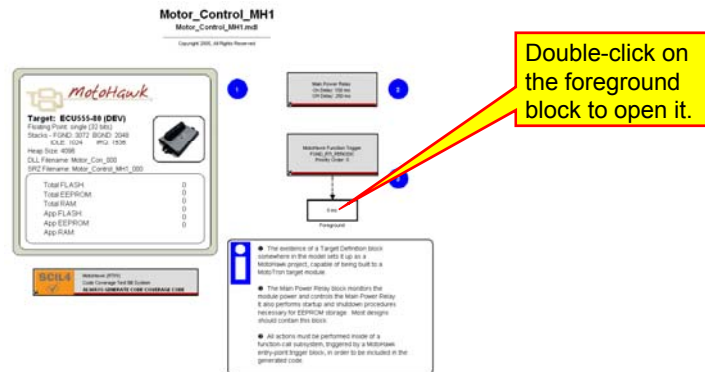
- We notice that the Simulation Configuration Parameters are set up correctly for our project.
- The parameters are set up correctly because we initially created the project using the command `motohawk_project('Motor_Control_MH1')`.
- We do not need to make any changes.
- Click the **OK** or **Cancel** button.

MotoTron The MathWorks freescale semiconductor ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Foreground Process

33

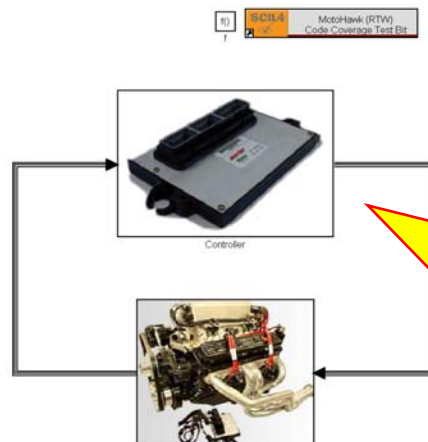
- We are now ready to create our controller.
- Our first controller will just turn on and off the LEDs.
- These LEDs tell us that the ECU is alive.



Default Foreground Block

34

PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF MOTOTRON. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MOTOTRON IS PROHIBITED. © 2005	 MotoTron Corporation 505 Manon Road, Oshkosh, WI, 54901 www.mototron.com (734) 622-7700 <hr/> Description: Motor_Control_MH1 Foreground <hr/> Path: Motor_Control_MH1\Foreground REV:000
--	--



- Default foreground process.
- We will not use this stuff.
- Delete the controller and plant.
- Your foreground subsystem should look as shown on the next slide.



New Foreground Subsystem

35

PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF MOTOTRON. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MOTOTRON IS PROHIBITED. © 2005	 Mototron Description: Motor_Control_MH1 Foreground	MotoTron Corporation 505 Marion Road, Oshkosh, WI, 54901 www.mototron.com (734) 622-7700
Path: Motor_Control_MH1\Foreground		REV:000

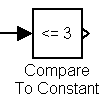
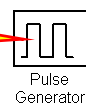
1)  SCALE: Mototron (RTW)
Code Coverage Test Bit

Flashing Light Controller

36

- We will now create a controller that turns on and off two LEDs at a one second rate.
- Place a part called **Pulse Generator** (located in the **Simulink / Sources** library).
- Place a part called **Compare to Constant** located in **Simulink \ Logic and Bit Operations** library

Double-click on this block to open it. Change it as shown next.





37

The screenshot shows the 'Source Block Parameters: Pulse Generator' dialog box. The 'Pulse Generator' section contains a code snippet: `if (t >= PhaseDelay) && Pulse is on
Y(t) = Amplitude
else
Y(t) = 0
end`. The 'Parameters' section has the following settings: 'Pulse type' is 'Sample based', 'Time (t)' is 'Use simulation time', 'Amplitude' is '1', 'Period (number of samples)' is '200', 'Pulse width (number of samples)' is '100', 'Phase delay (number of samples)' is '0', and 'Sample time' is '-1'. The 'Interpret vector parameters as 1-D' checkbox is checked. Red arrows point from yellow callout boxes to these settings.

Select sample based.
The sample time is 5 ms since we are in a triggered subsystem that is triggered every 5 ms.

The output pulse goes from 0 to 1.

The period is 200 samples. Since the sample time is 5 ms, the period is 200 times 5 ms, or 1 second.

Pulse width is 100 samples of 0.5 seconds. This is the time the pulse is 1.

Inherit the sample time. This will be 5 ms since we are in a triggered subsystem.

Click OK when done.

OK Cancel Help

Compare To Constant

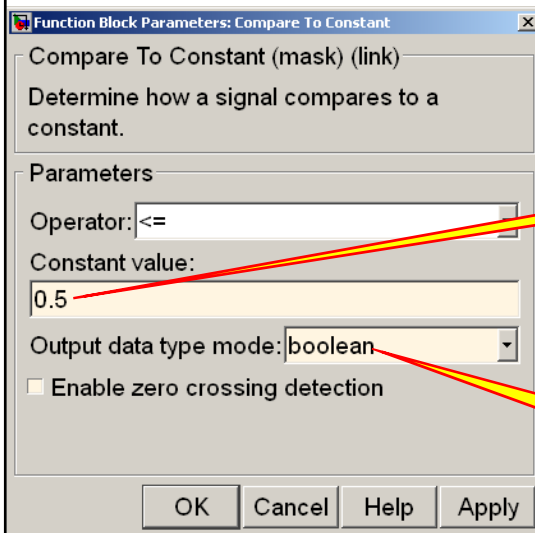
38

- We want the Compare To Constant block to perform a logical inversion function.
- To do this, we can change the comparison to ≤ 0.5 .
- When the pulse output is 1, the comparison will be false and the block will output false.
- When the pulse output is 0, the comparison will be true and the block will output true.
- Double-click on the block and change the dialog box as shown:



Compare To Constant

39



Change to 0.5.

Change to boolean. We'll see why in a few slides.



MotoHawk Digital Output

40

- We now need to add blocks to access the digital outputs of the MotoTron ECU.
- Place two blocks called **MotoHawk_Dout** in your model. These blocks are located in the **MotoHawk \ Digital I/O Blocks** library.
- Connect the blocks as shown:

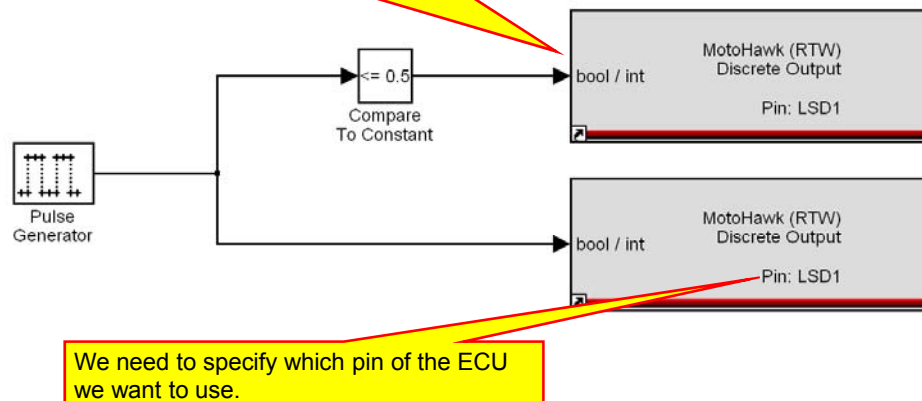




MotoHawk Digital Output

41

This block requires a Boolean data type for the input. This is why we specified that the output data type of the Compare To Constant block as Boolean.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoTron ECU Resources

42

- The MotoTron ECU has several digital outputs.
- We will use the data sheet to select one.
- Our LED will draw about 50 mA of current.
- The digital outputs of the MotoTron ECU have high current drivers, so this should not be an issue.
- Open the data sheet for your ECU.
- We will use the FUELP and TACH outputs.
- The data sheet detailing these outputs is shown next:

MotoTron

The MathWorks

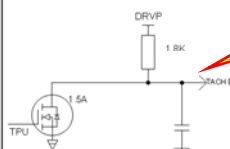
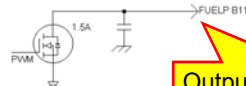
freescale
semiconductor





ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



43

FUELP





2.8 TACH (B12) This output is capable of sinking 1.5A max.	
2.9 FUELP (B11) This output is capable of sinking 1.5A max. Note: The STOP signal (1.13) will disable the FUELP output when asserted.	



44

FUELP and TACH Outputs

- To determine which wires in the development harness we should use, we need to scroll down further in the data sheet.





ECU555-80 Family Datasheet

Pin #	ControlCore	Function	Notes	Wire Number
PCM	Resource Name	Name		Color Code
B1	EST_RTN	Electronic Spark Timing Return	Low Current Return from Spark Coils	33 Black/Green
B2	EST1	Electronic Spark Timing	SmartCoil Driver	34 Green/Black
B3	DG4	Discrete Switch, Frequency, IRQ	1K Pull Up	35 Gray/Dark Blue
B4	MPRD	Main Power Relay Driver	Wire to Main Power Relay Coil	36 Yellow/Purple
B5	CNK-	Crank Position LO	Variable Reluctance Sensor Compatible with NSC LM1815	37 White
B6	CAM_DG	Hall Effect Cam Sensor	150K Pull Up	38 White/Purple
B7	OILP	Oil Pump		39 Light Blue/Black
B8	START	Starter Solenoid Relay	High Current (10A)	40 Yellow/Black
B9	EST5	Electronic Spark Timing	SmartCoil Driver	41 Green/Purple
B10	EST3	Electronic Spark Timing	SmartCoil Driver	42 Green/Brown
B11	FUELP	FUEI Pump	PWM	43 Orange
B12	TACH	Tachometer Output	1.8K Pull Up	44 Gray
B13	CNK+	Crank Position HI	Variable Reluctance Sensor Compatible with NSC LM1815	45 Red

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

FUELP and TACH

46

The FUELP output is an Orange wire and is numbered 43.

B11	FUELP	FUEI Pump	PWM	43 Orange
B12	TACH	Tachometer Output	1.8K Pull Up	44 Gray

The TACH output is an Orange wire and is numbered 44.

MotoTron

The MathWorks

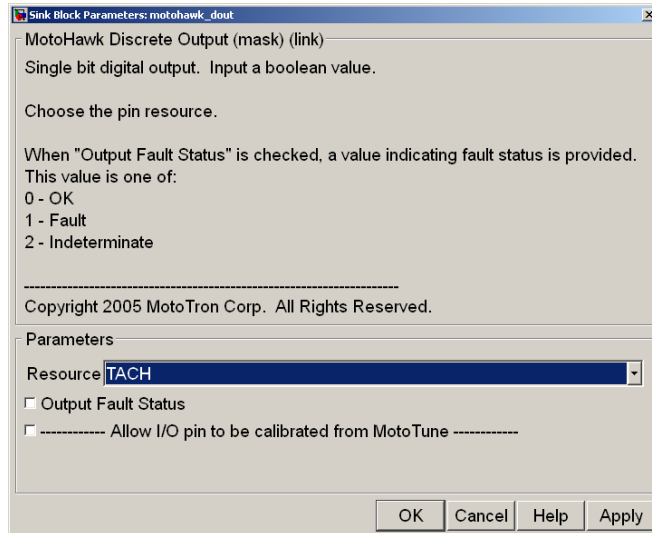
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Digital Output

47

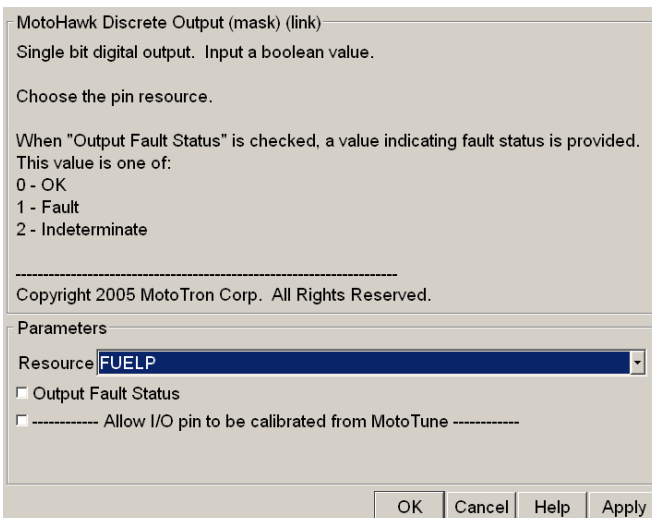
- In the Simulink model, double-click on the Discrete Output block and change the **Resource** to **TACH**:



Digital Output

48

- Change the **Resource** of the other digital output block to **FUELP**:

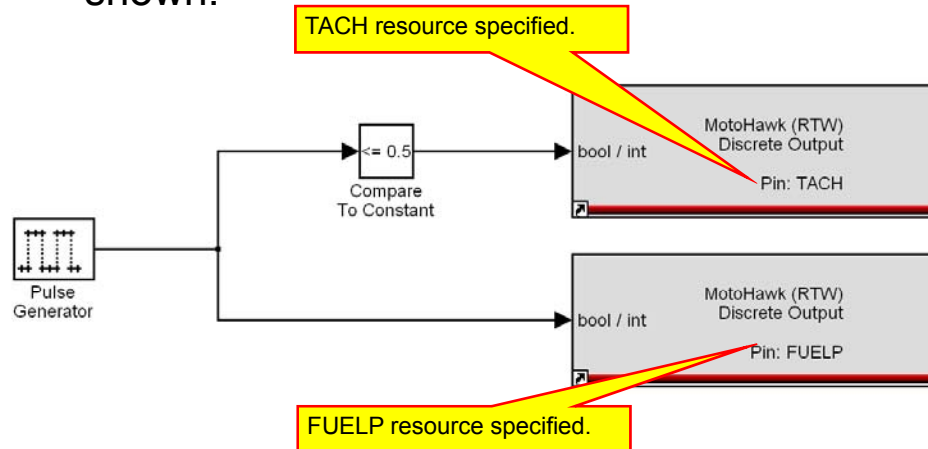




Blinky Lights Model

49

- Your Simulink model should appear as shown:



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Data Type Conversion

50

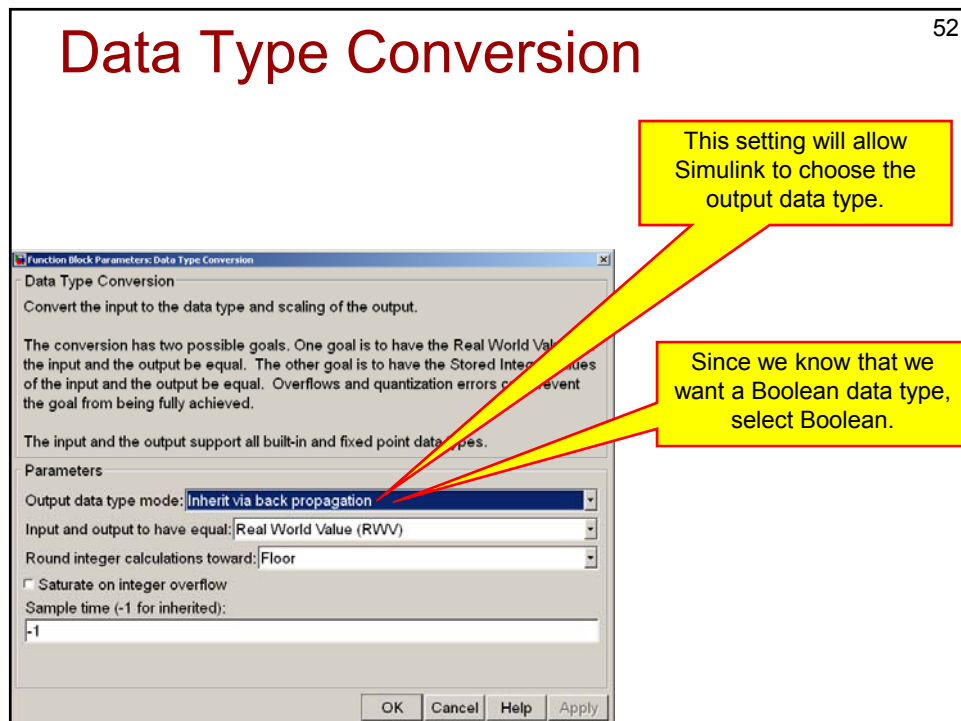
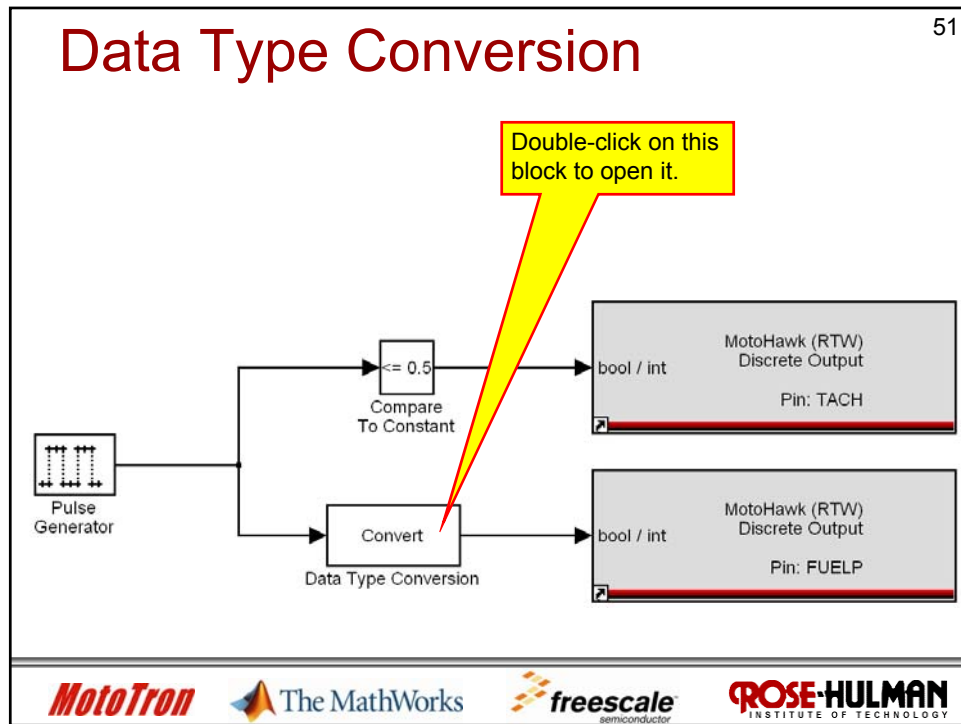
- There is one last step we need to take.
- The output of the Pulse Generator block is a double precision data type (even though it only outputs a value of 0 or 1).
- The input data type for the Discrete Output block is Boolean.
- We need to convert the double precision data type to a Boolean.
- Place a **Data Type Conversion** block as shown. This block is in the **Simulink \ Commonly Used Blocks** library.

MotoTron

The MathWorks

freescale
semiconductor

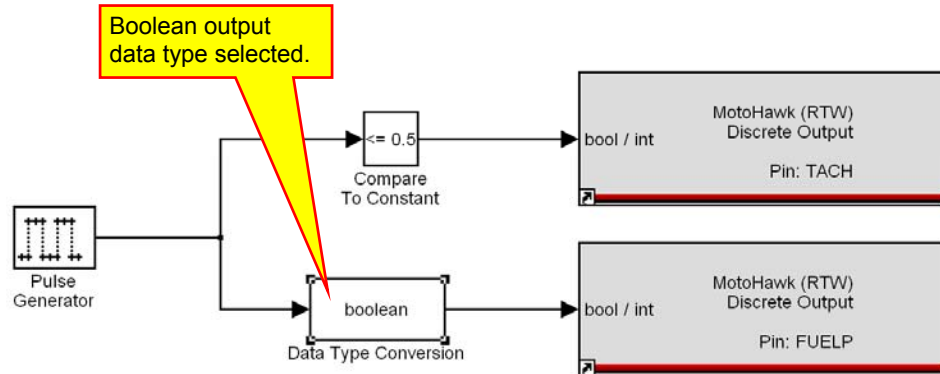
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





Data Type Conversion

53



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Data Types

54

- It is always a good idea to check your model for errors periodically.
- We would also like to display data types on the signal lines to help spot any errors.
- Data type mismatches can cause erratic behavior of your controller and be difficult to diagnose.
- To display data types on your schematic, select **Format, Port/Signal Display**, and then **Port Data Types** from the Simulink menus.

MotoTron

The MathWorks

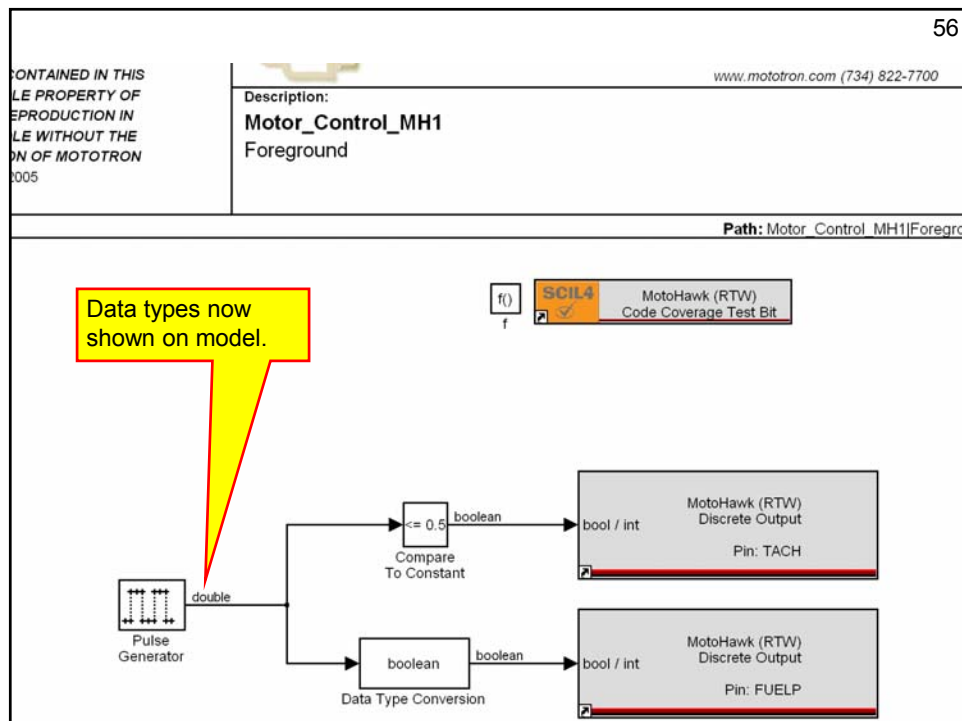
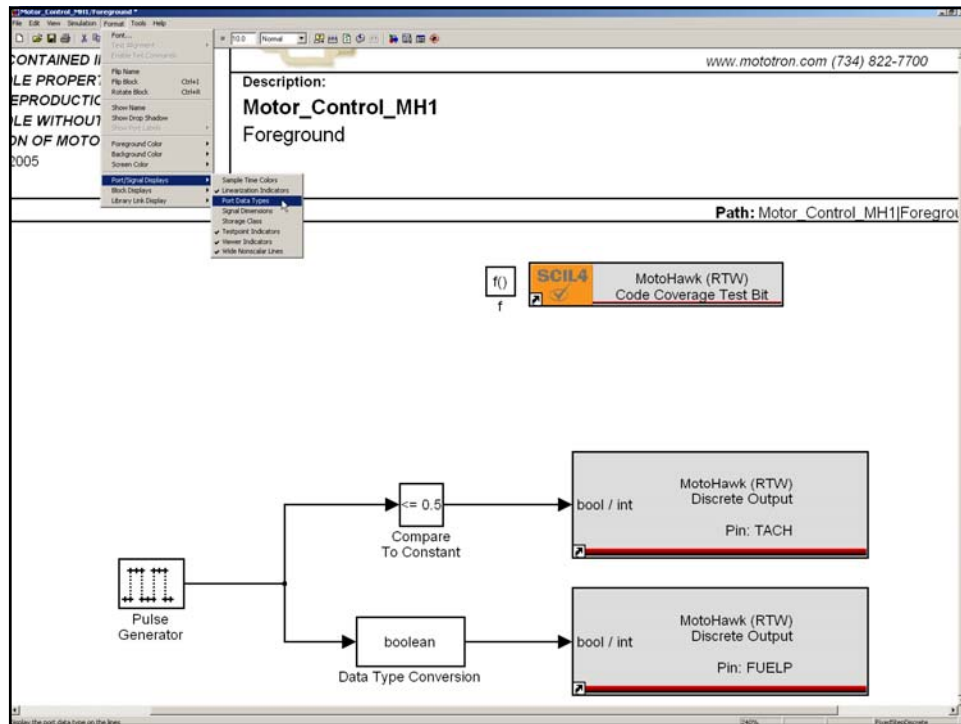
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



Except where otherwise noted, this work is licensed under

<http://creativecommons.org/licenses/by/3.0/>





Updating the Model

57

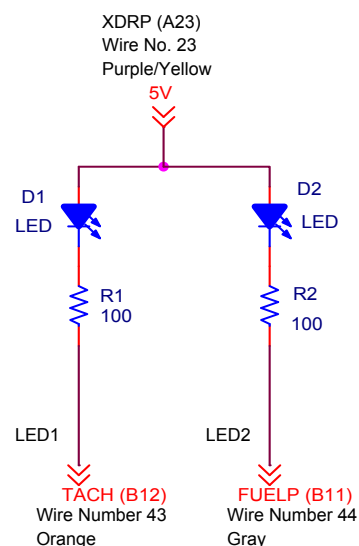
- When you make changes to your model, the data types are not automatically updated on the schematic.
- We would also like to check our model for errors.
- To accomplish both of the above, type **ctrl-d** in the Simulink window.
- Any errors will be listed and the data types will be updated.
- My model was already up to date and there were no errors, so there is nothing to show.
- However, you should use the **ctrl-d** command frequently.



Wiring

58

- The next thing we need to do is wire up our circuit.
- The LED circuit is shown to the right.
- We have already identified the wires for the TACH and FUELP Signals.
- We need to find a 5V reference.



Wiring

59

- MotoTron ECU provide a +5V reference for use in sensors.
- Two signal lines are available called XDRP and XDRP_B (Pins A23 and B24)

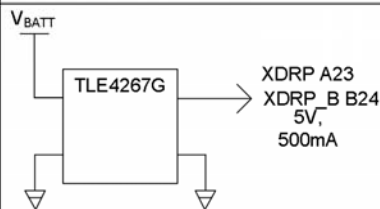


ECU555-80 Family Datasheet

2 Output Signal Conditioning

2.1 XDRP, XDRP_B (A23, B24)

These outputs are for powering sensor transducers. 5V 500mA max.



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Wiring

60

- Looking further down the datasheet for our ECU, we find
 - XDRP (A23) is wire number 23 and is Purple/Yellow

A23	XDRP	Transducer Power (5V)	500mA Source for Transducers	23 Purple/ Yellow
-----	------	-----------------------	------------------------------	-------------------

- XDRP_B (B24) is wire number 56 and is Purple/Pink

B24	XDRP_B	Transducer Power B (5V)	500mA Source for Transducers	56 Purple/Pink
-----	--------	-------------------------	------------------------------	----------------

MotoTron

The MathWorks

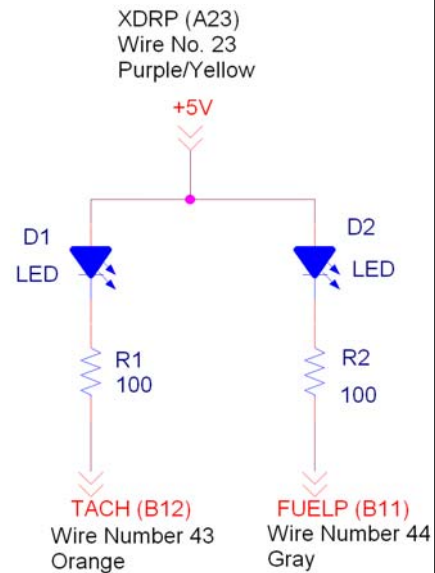
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Wiring

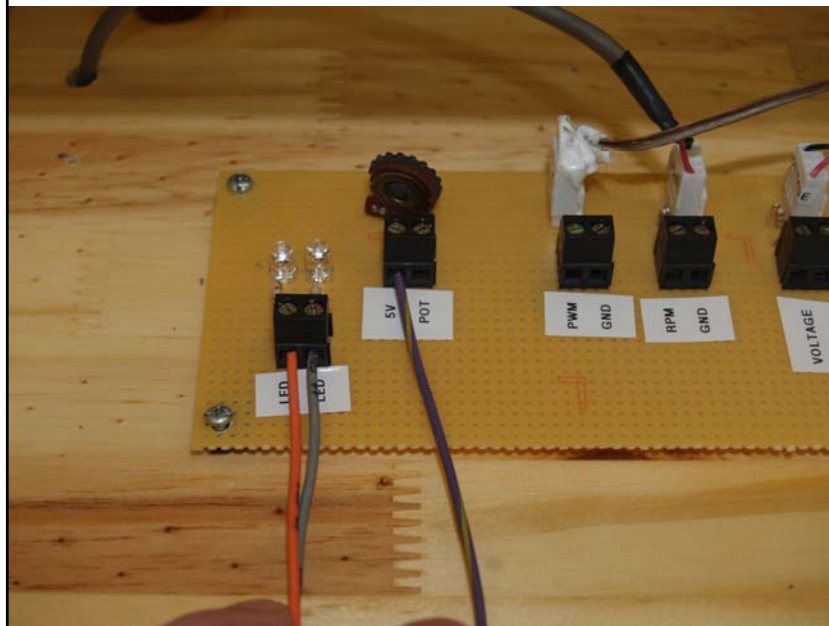
61

- We need to make connections as shown.
- Physical Connections are Shown on the next slide.



Wiring

62






Project Build

63

- We are now ready to build the Project.
- If you typed ctrl-d in the Simulink window and received no errors, you should be able to build the project without any errors.
- Before we build, notice in the MotoHawk title block that we are at revision 000.

PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF MOTOTRON. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MOTOTRON IS PROHIBITED. © 2005	 MotoHawk	MotoTron Corporation 505 Marion Road, Oshkosh, WI, 54901 www.mototron.com (734) 822-7700
	Description: Motor_Control_MH1 Top-level root of model: Motor_Control_MH1.mdl	
		Path: Motor_Control_MH1 REV:000

Revision 000.

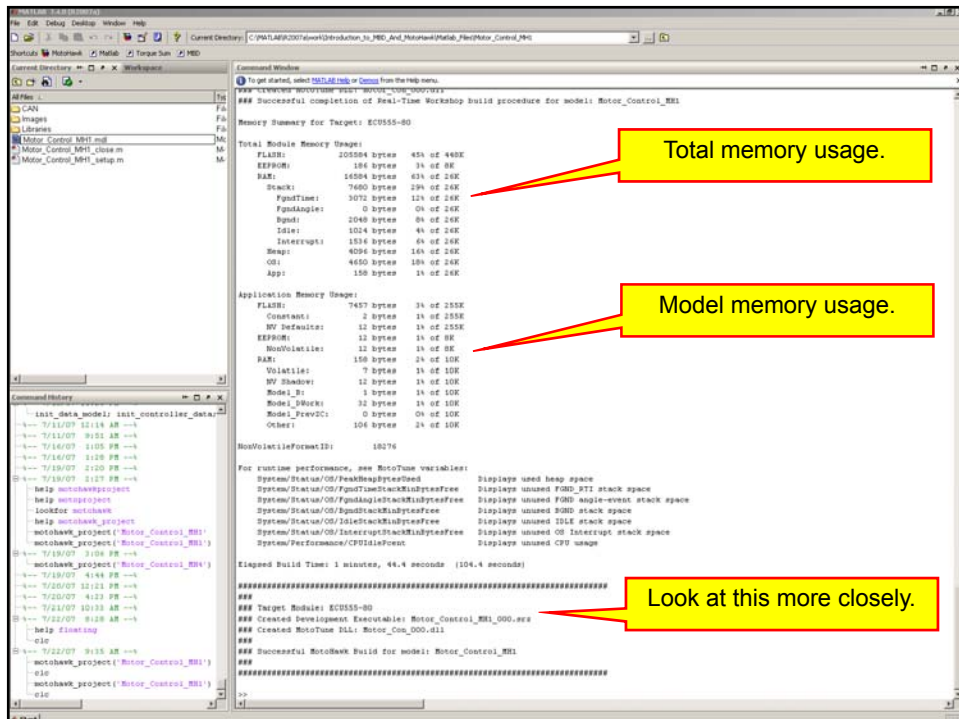
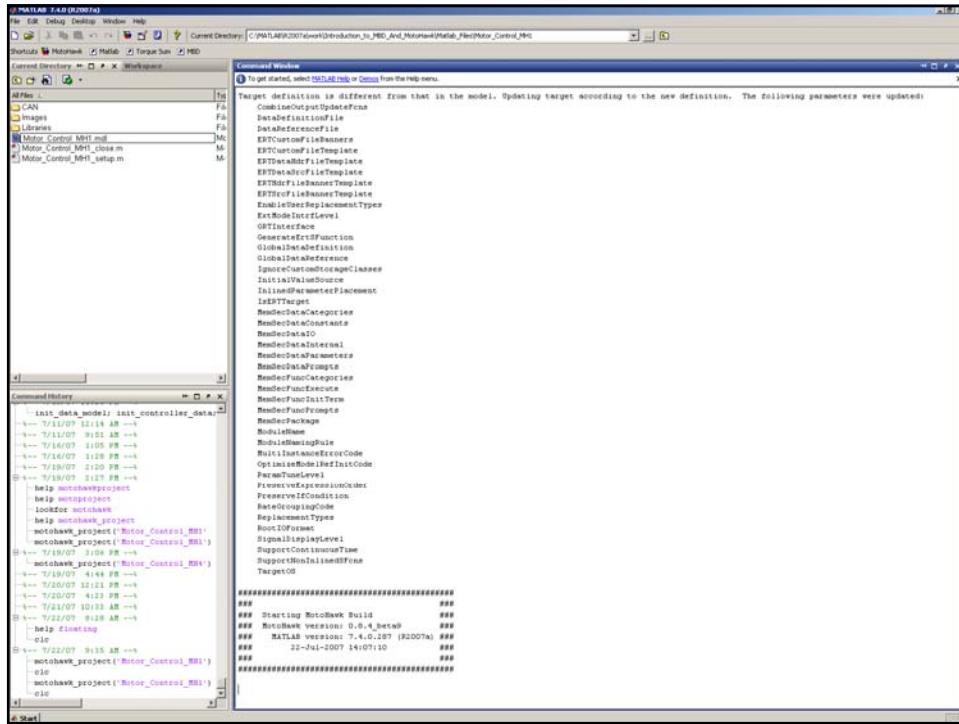


Project Build

64

- In the Simulink window, type ctrl-b to build the project.
- You can also select **Tools, Real-Time Workshop**, and then **Build Model** from the Simulink menus.
- After typing **ctrl-b**, switch to the Matlab command window to observe the progress.







Project Build

67

This is the file we will
download to the ECU.

```
#####
###
### Target Module: ECU555-80
### Created Development Executable: Motor_Control_MH1_000.srz
### Created MotoTune DLL: Motor_Con_000.dll
###
### Successful MotoHawk Build for model: Motor_Control_MH1
###
#####
```

This build was created
with rev 000 of the model.

MotoTron

 **The MathWorks**

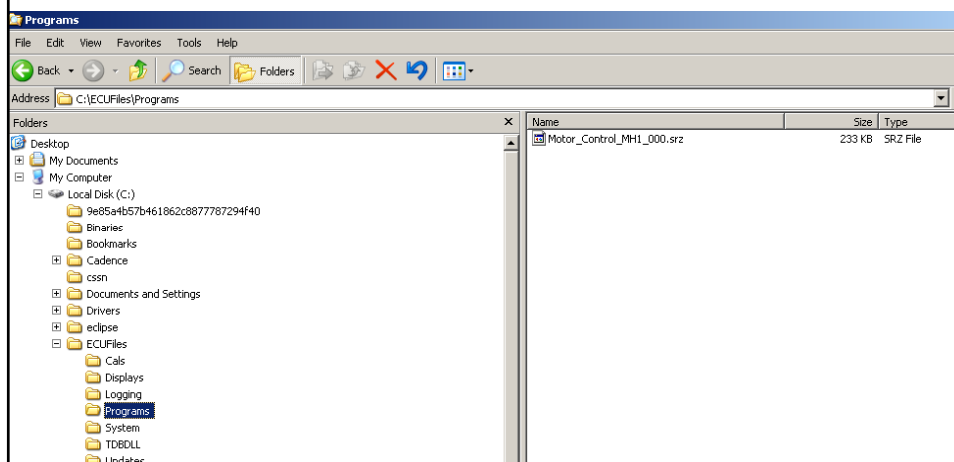
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Project Build

68

- The build files we just created are stored in directory C:\ECUFiles.
- The .srz files are located in directory C:\ECUFiles\Programs










Project Build

69

- When we switch back to our model, we notice that the revision number in the header has been incremented.

PROPRIETARY AND CONFIDENTIAL <small>THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF MOTOTRON. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MOTOTRON IS PROHIBITED. © 2005</small>		MotoTron Corporation <small>505 Marion Road, Oshkosh, WI, 54901 www.mototron.com (734) 822-7700</small>
	Description: Motor_Control_MH1 Top-level root of model: Motor_Control_MH1.mdl	
Path: Motor_Control_MH1		REV: 001

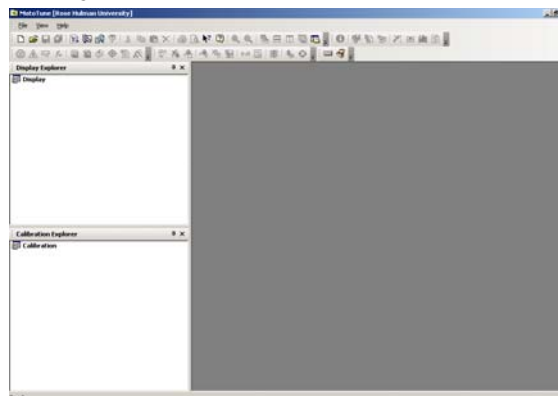
Revision 001.

MotoTune


70

- To download our controller to the MotoTron ECU we will use MotoTune.
- Run MotoTune. It is located in the MotoTools folder in your Start menu.



MotoServer

71

- Before we download our model, we will check the CAN port settings in MotoServer.
- The MotoServer icon  should be located in your windows tray.

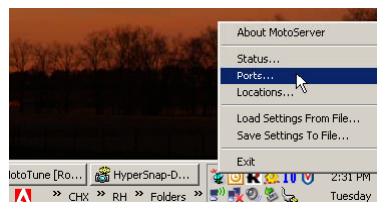
MotoServer icon.



MotorServer

72

- Right-click on the MotoServer icon and select **Ports**



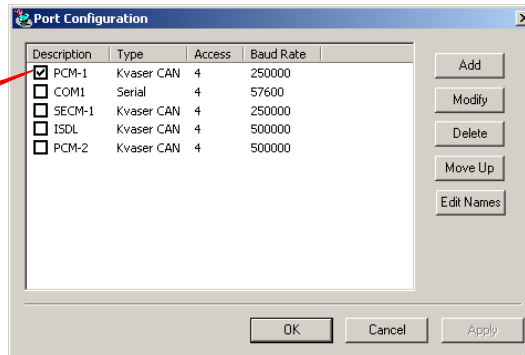


MotorServer

73

- You should have a port labeled PCM-1. The correct settings are:
 - Kvaser CAN
 - Access 4
 - Baud Rate 250000

This port should be enabled.



MotoServer

74

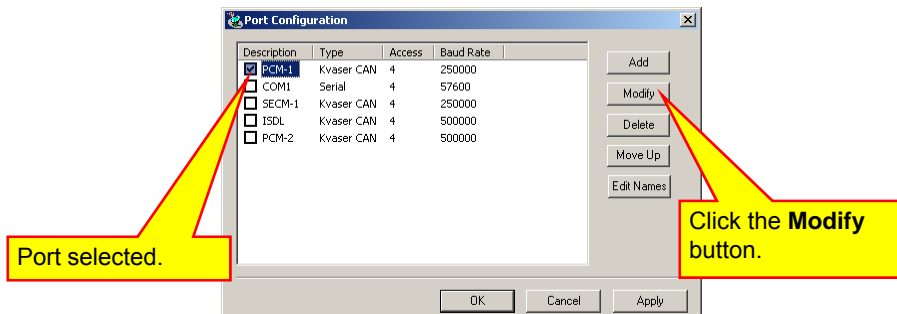
- The CAN Baud rate default setting of the ECU modules is 250k.
- If you have never changed the baud rate of your ECU, then it is probably set to 250 k.
- If your port settings are not as shown on the previous slide, or you know that your ECU was programmed previously with a different Baud rate, we will need to change the port settings.



MotoServer – Only if Necessary

75

- Skip to slide 84 if your port settings are correct as shown in slide 74.
- If you need to change the port settings, select PCM-1 and click the **Modify** button.



MotoTron

The MathWorks

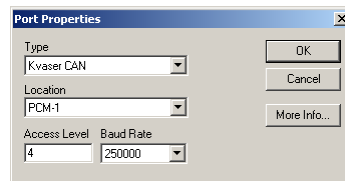
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoServer – Only if Necessary

76

- Change the settings as needed to match your CAN baud rate or the settings shown:



- Click the **OK** button to accept the changes.

MotoTron

The MathWorks

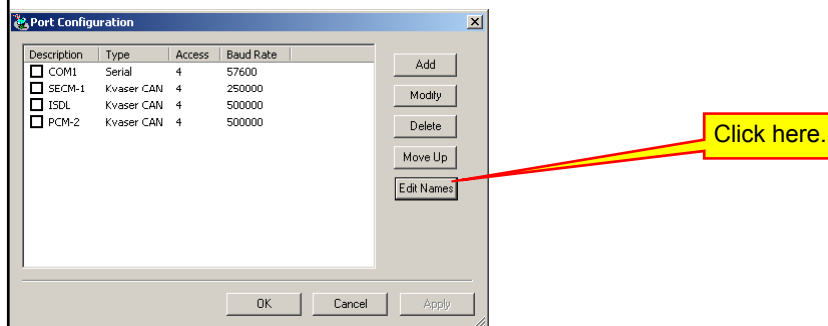
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoServer – – Only if Necessary

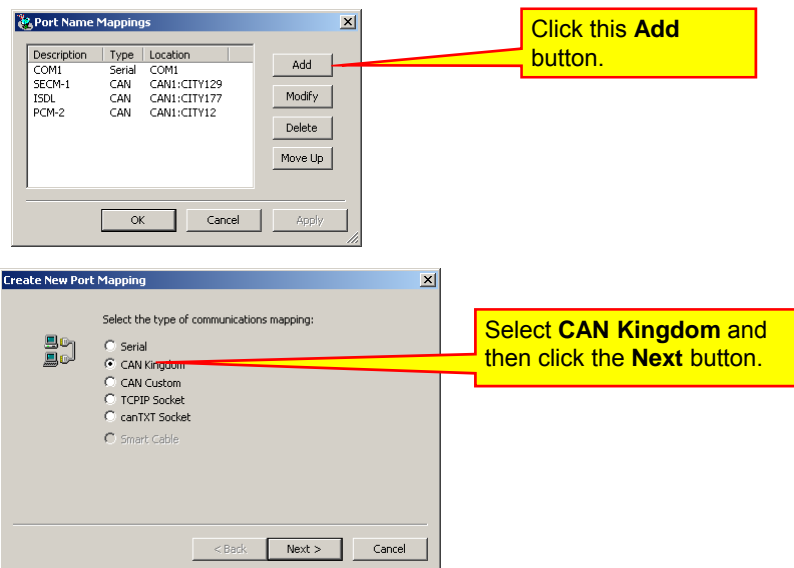
77

- If your port settings are now correct, skip to slide 84.
- If your window does not show PCM-1, you must do the following:
- Click the **Edit Names** button



MotoServer – Only if Necessary

78





MotoServer – Only if Necessary

79

- Fill in the dialog box as shown and click the **Finish** button.

The dialog box is titled "CAN Kingdom Mapping". It contains the following fields and buttons:

- Port Name: PCM-1 (Annotated: Name is PCM-1.)
- Port Can Bus: 1 (Annotated: CAN Bus 1.)
- CityID: 11 (Annotated: CityID is 11.)
- Buttons: < Back, Finish, Cancel

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoServer – Only if Necessary

80

- PCM-1 should be added to the CAN Bus Mappings.

The dialog box is titled "Port Name Mappings". It contains a table with the following data:

Description	Type	Location
COM1	Serial	COM1
SECM-1	CAN	CAN1:CITY129
ISOL	CAN	CAN1:CITY177
PCM-2	CAN	CAN1:CITY12
PCM-1	CAN	CAN1:CITY11

An annotation points to the "PCM-1" row in the table: Name listed here.

Buttons: Add, Modify, Delete, Move Up, OK, Cancel, Apply

- Click the **OK** button.

MotoTron

The MathWorks

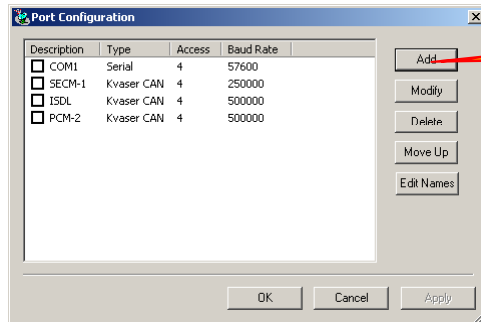
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

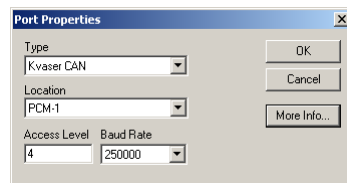


MotoServer – Only if Necessary

81



Click this **Add** button.



Fill in properties as shown.

- Type – Kvaser CAN
- Location PCM-1
- Access Level 4
- Baud Rate 250000
- Click the **OK** button when done.

MotoTron

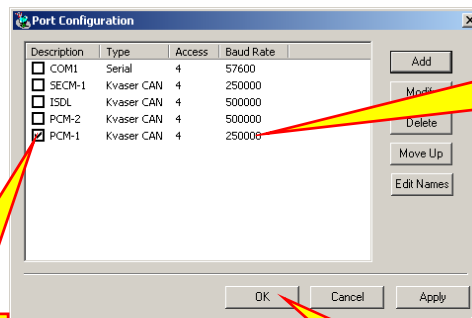
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoServer – Only if Necessary

82



The port should be added with the proper settings.

Port is selected.

Click the **OK** button. We are ready to go.

MotoTron

The MathWorks


freescale
semiconductor

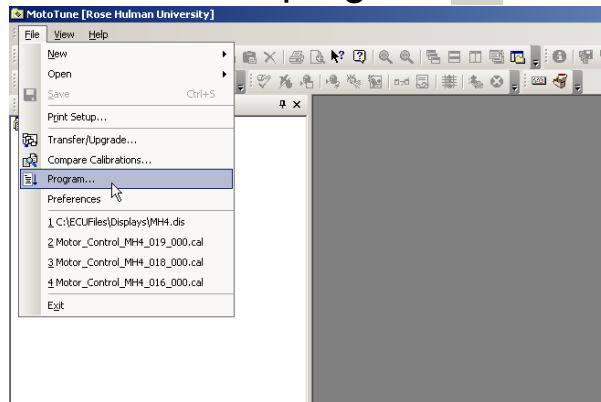
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTune

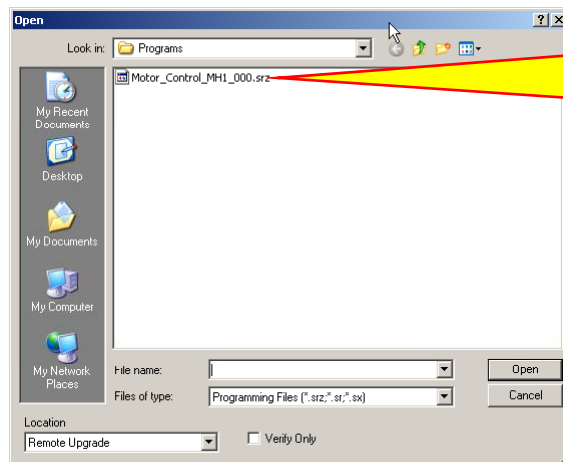
83

- We now have the ports set correctly and can program our ECU!
- In the MotoTune window select **File** and then **Program**, or select the program  button:



MotoTune

84

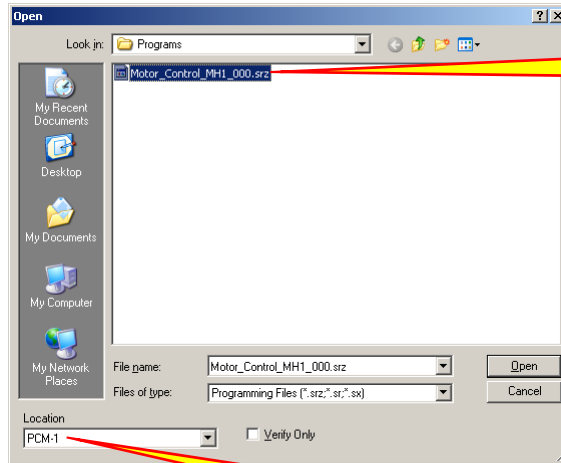


Select the most recent .srz file for your model (or the .srz file you want to download to your ECU).



MotoTune

85



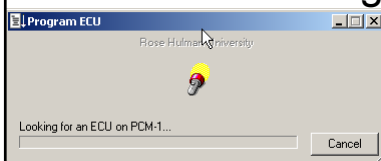
File selected. Click the **Open** button.

Note that we are using port PCM-1. If this port is not selected, use the pull-down menu to choose port PCM-1.

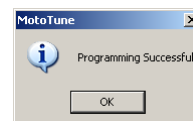
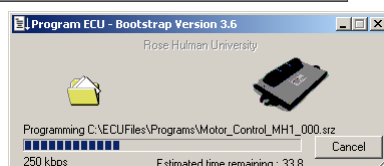
MotoTune

86

- If you are successful, you will see the series of dialog boxes below:



If you do not see this box, you may need to turn on your key switch and/or turn on your 12 V power supply.





MotoTune

87

- Our model should now be programmed on the ECU.
- You should see the LEDs flashing on and off.
- As you cycle the key switch, the ECU should automatically power off and on and start the program (which flashes the LEDs).
- MotoTune has many more capabilities which we will show in our next example.
- Here, we only show how to program the ECU with MotoTune.



Build Process

88

- If you make any changes to the model, you will need to go through the following steps to run the new model on the ECU:
 - Build the model in Simulink (**ctrl-b**)
 - Program the ECU with MotoTune
 - That is it!



CAN Baud Rate

89

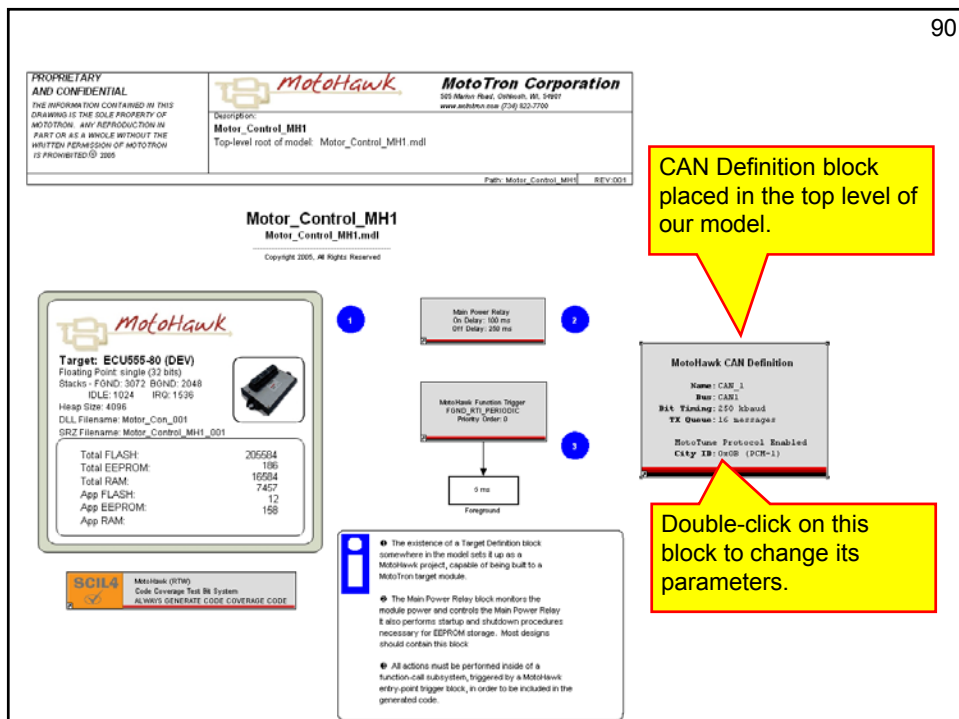
- As a last part of this example, we will show how to change the CAN Baud rate of our ECU.
- We will increase the Baud rate to 500 K.
- All future examples will use a Baud rate of 500 k.
- Place a part called **CAN Definition** in your model.
- This block is located in the **MotoHawk / CAN** library.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY





91

Block Parameters: CAN Definition

MotoHawk CAN Bus Definition (mask) (link)

CAN Resource selects which bus to initialize. CAN_1 is available on most modules, but CAN_2 and CAN_3 are only available on some. Please check the datasheet for the desired module.

Bit Timing allows a preset baud rate to be selected, or lets user-defined attributes be used to setup the CAN baud rate.

If a user-defined baud rate is selected, the following fields may be filled in:

- Prescaler
- Propagation Segment
- Phase Segment 1 and Phase Segment 2
- Resynchronization Jump Width

These are specific to the CAN hardware

Copyright 2005 MotoTron Corp. All Rights Reserved.

Parameters

Name: CAN_1

Resource: CAN1

Bit Timing: 250 kbaud

Transmit Queue Size: 16

☒ Install MotoTune Protocol

City ID: hex2dec('B')

City ID Access Level: 4

MotoTune Group String: System | System Config | Communication Config

OK Cancel Help Apply

MotoTron ECUs have two CAN ports available. We can change the properties of each CAN port independently.

Default CAN Baud rate is 250K. If we do not change the Baud rate with a CAN Definition block, the baud rate will be 250k.

Note that the City ID is 11 (hex B). Remember that the City ID for PCM-1 was also set to 11. This enables MotoTune to communicate with your ECU.

92

Block Parameters: CAN Definition

MotoHawk CAN Bus Definition (mask) (link)

CAN Resource selects which bus to initialize. CAN_1 is available on most modules, but CAN_2 and CAN_3 are only available on some. Please check the datasheet for the desired module.

Bit Timing allows a preset baud rate to be selected, or lets user-defined attributes be used to setup the CAN baud rate.

If a user-defined baud rate is selected, the following fields may be filled in:

- Prescaler
- Propagation Segment
- Phase Segment 1 and Phase Segment 2
- Resynchronization Jump Width

These are specific to the CAN hardware

Copyright 2005 MotoTron Corp. All Rights Reserved.

Parameters

Name: CAN_1

Resource: CAN1

Bit Timing: 500 kbaud

Transmit Queue Size: 16

☒ Install MotoTune Protocol

City ID: hex2dec('B')

City ID Access Level: 4

MotoTune Group String: System | System Config | Communication Config

OK Cancel Help Apply

The only change we will make is to change the Baud rate to 500 k.

Make this change and click the OK button.



CAN Baud Rate

93

Main Power Relay
On Delay: 100 ms
Off Delay: 250 ms

2

MotoHawk Function Trigger
FGND_RTL_PERIODIC
Priority Order: 0

3

5 ms
Foreground

MotoHawk CAN Definition

Name: CAN_1
Bus: CAN1
Bit Timing: 500 kbaud
TX Queue: 16 messages
MotoTune Protocol Enabled
City ID: 0x0B (PCM-1)

- Note that the Baud rate change is reflected here.
- Save your model and the type ctrl-b to build it.
- View the Matlab window to observe the build progress.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

CAN Baud Rate

94

- Since this is the second time we have built the model, the executable file is now called ...001.srz. Each time we build the model, we will get a new .srz file.

```
For runtime performance, see MotoTune variables:
System/Status/OS/PeakHeapBytesUsed           Displays used heap
System/Status/OS/FgndTimeStackMinBytesFree     Displays unused FgndTime stack space
System/Status/OS/FgndAngleStackMinBytesFree    Displays unused FgndAngle stack space
System/Status/OS/BgndStackMinBytesFree         Displays unused Bgnd stack space
System/Status/OS/IdleStackMinBytesFree         Displays unused Idle stack space
System/Status/OS/InterruptStackMinBytesFree    Displays unused Interrupt stack space
System/Performance/CPUIdlePcent                Displays unused CPU usage
```

Elapsed Build Time: 1 minutes, 56.4 seconds (116.4 seconds)

```
#####
###
### Target Module: ECU555-80
### Created Development Executable: Motor_Control_MH1_001.srz
### Created MotoTune DLL: Motor_Con_001.dll
###
### Successful MotoHawk Build for model: Motor_Control_MH1
###
#####
```



CAN Baud Rate

95

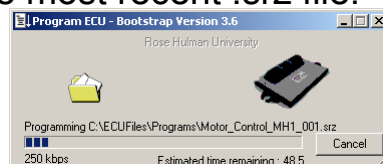
- We now need to program the ECU with the new executable (.srz file).
- Note that the ECU is still running at 250 K.
- The ECU will not run at the changed Baud rate until we program it.
- Thus, we will leave the MotoServer ports at 250k until the ECU is programmed.



CAN Baud Rate

96

- We will now program the ECU.
- Select **File** and then **Program** from the MotoTune menus.
- Select the most recent .srz file.




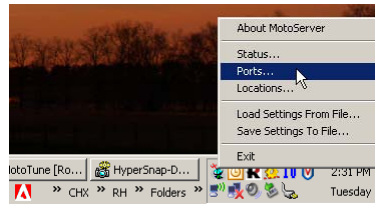
- When the programming is complete, the ECU will use a Baud rate of 500k.
- Note that your two LEDs should still be flashing. All we changed in the model was the CAN rate.



CAN Baud Rate

97

- Now that the ECU is running at a different CAN Baud rate, we need to change the port settings on MotoServer.
- Right-click on the MotoServer icon  and select **Ports**.



MotoTron

The MathWorks

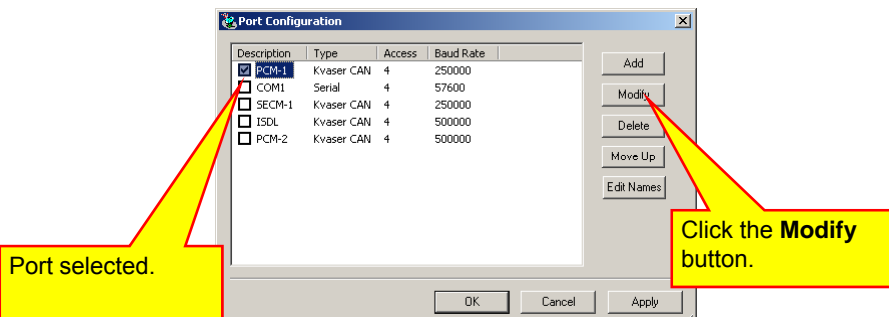
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

CAN Baud Rate

98

- To change the port settings, select PCM-1 and click the **Modify** button.



MotoTron

The MathWorks

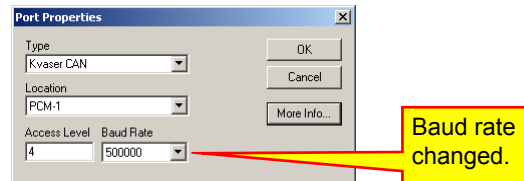
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

CAN Baud Rate

99

- Change the settings as shown:



- Click the **OK** button to accept the changes.

MotoTron

The MathWorks

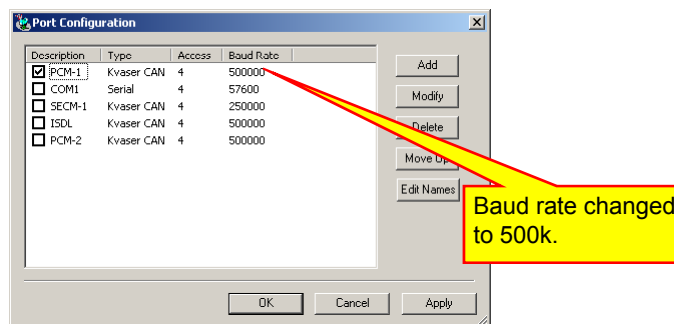
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

CAN Baud Rate

100

- Your port configuration should be as shown.



- Click the **OK** button if your settings match.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



CAN Baud Rate

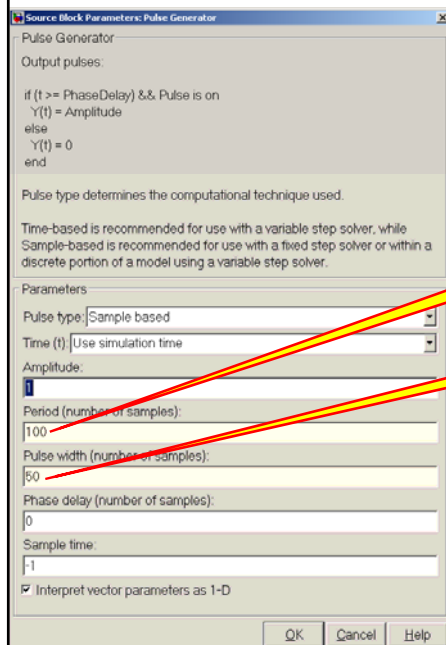
101

- MotoTune can once again communicate with our ECU.
- To show this, we will change the rate at which the LEDs flash and then reprogram the ECU.
- Double-click on the Pulse Generator block inside your model.
- Make the following changes:



CAN Baud Rate

102



Period set to 100.

Pulse width set to 50.



CAN Baud Rate

103

- Click **OK** to close the dialog box.
- Type ctrl-b in Simulink to build the model.
- Use MotoTune to down load the newly created .srz file
 - Select **File/Program** from the MotoTune menus.
 - Select the most recent .srz file.



CAN Baud Rate

104

- If the Baud rates were changed in both the ECU and MotoServer ports and they match, programming of the ECU should start.



- When programming is complete, you the new model should run, and the lights should flash faster.



The Last Slide

105

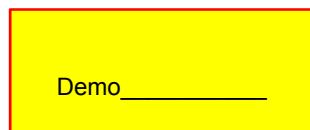
- We will leave the CAN Baud rate at 500k for the remaining examples in this workshop.



Lecture 16 Demo 1

106

- Demo the Operation of the MotoTron ECU blinking lights. The CAN baud rate should be 500 k.





Advanced Model-Based-System Design

Lecture 17: MotoTron MotoHawk Projects



Outline

2

- Implement the controller for the Motor Generator system that we modeled earlier.
- Use the MotoHawk PWM block.
- Use the MotoHawk analog input block.
- Use MotoHawk Probes to view signals internal to the ECU in real-time.
- Use MotoHawk Calibration blocks to change the feedback gains in real-time.
- View real-time signals using MotoTune charts.





Motor Control Project

3

- We will now create a new MotoHawk project in our working directory.
- Enter the command
`motohawk_project('Motor_Control_MH2')`
at the Matlab command prompt.
- Simulink will run and open the new model.



Motor Control Project

4

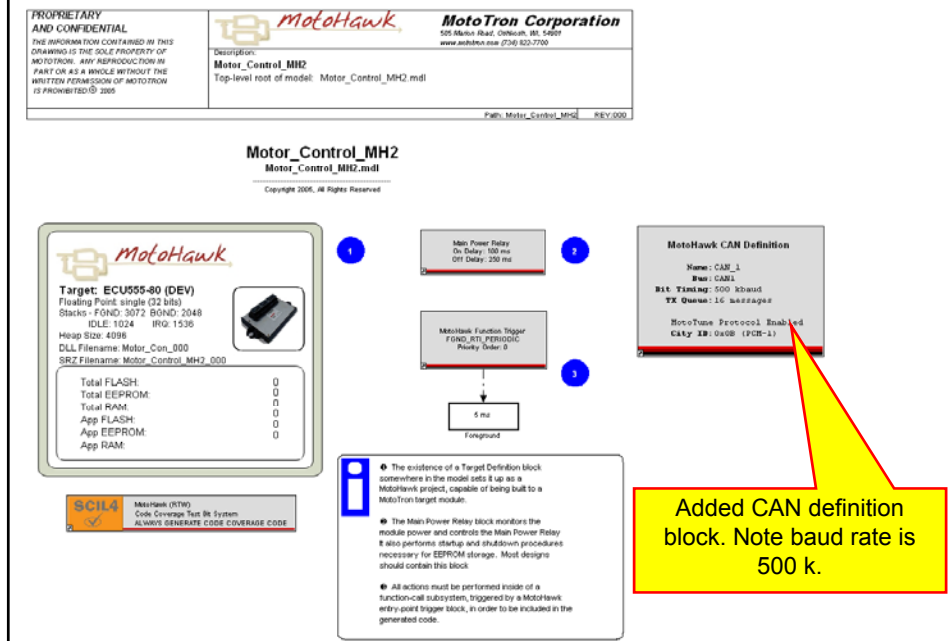
- We need to make a few modifications:
 - Change the CAN Baud rate to 500k (copy the can block from our first model)
 - Delete the controller and plant subsystems inside the Foreground subsystem.
 - Copy the Flashy lights portion of the first model and place it in the foreground subsystem of our new model. (These lights will tell us that the ECU is working.)
 - The following two slides show what you should have:





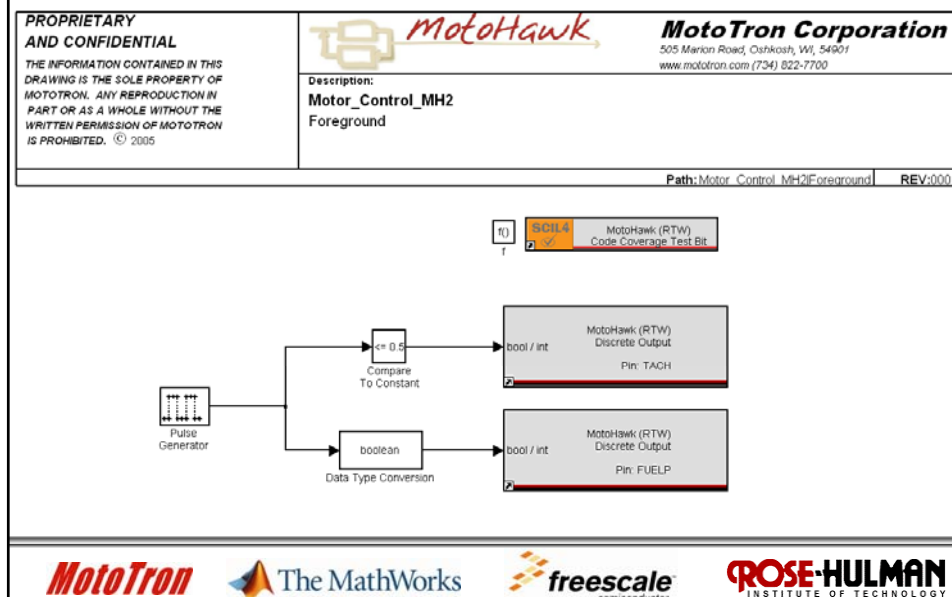
Top Level Model

5



Foreground Subsystem

6





Analog Inputs

7

- In our model, we are going to read three analog inputs:
 - The potentiometer (POT) – This is a 0 to 5V analog voltage.
 - The motor rpm output signal. This is approximately a 0 to 8V signal coming out of the motor. This signal is divided in half by two 10 k resistors on the circuit board. Thus, the input is approximately a 0 to 4 V signal.
 - The generator output voltage. This is approximately a 0 to 24V signal coming out of the motor. This signal is divided in half by a resistor network on the circuit board to produce a 0 to 5V signal.



Analog Inputs

8

- We have scaled all of our analog signals to be between 0 and 5V.
- The MotoTron ECU has several analog input channels.
- Most channels are designed for a 0 to 5 V input.
- A portion of the available inputs are shown on the next slide.
- This information is contained in the data sheet for your MotoHawk ECU.





ECU555-80 Data Sheet

9

1.2 AN1M...AN3M (A3, A4, A5) These inputs are 10bit 0-5V ADC's, $\tau = 1\text{ms}$. They are intended for pressure sensors.	
1.3 AN4M...AN8M (A6, A7, A8, A9, A10) These inputs are 10bit 0-5V ADC's, $\tau = 1\text{ms}$. They are intended for potentiometers.	
1.4 AN9M...AN12M (A14, A15, A16, A17) These inputs are 10bit 0-5V ADC's, $\tau = 1\text{ms}$. They are intended for variable resistance sensors such as thermistors.	

Analog Inputs

10

- We will use analog inputs AN4M, AN5M, and AN6M because:
 - The do not have an internal pull-up resistor.
 - They have the highest input impedance:
 - AN1M through AN3m have 51.1k resistors to ground.
 - AN4M through AN8M have 200k resistors to ground.
 - All inputs have built-in filters. Note that we did not include these filters in our model.



Analog Inputs

11

- The ECU555-80 data sheets tells us that the analog inputs have a 0-5V range, 10-bit resolution, and a filter time constant of 1 ms.
 - A 0 V input is converted to a numeric value of 0.
 - A 5 V input is converted to a numeric value of $2^{10}-1$ or 1023.
- The 1 ms time constant comes from the low pass filter (the 33k resistor and the 0.033 μ F capacitor).

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

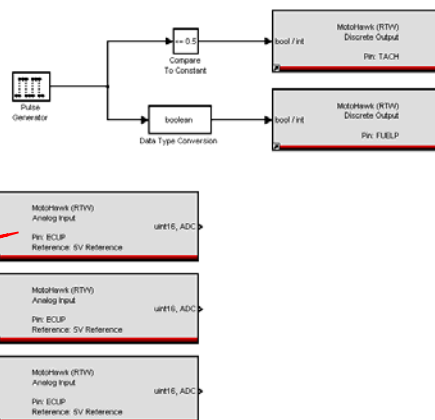
Analog Inputs

12

- Place three instances of the block `motohawk_ain` in your model.
- This block is in the **MotoHawk / Analog I/O Blocks** library

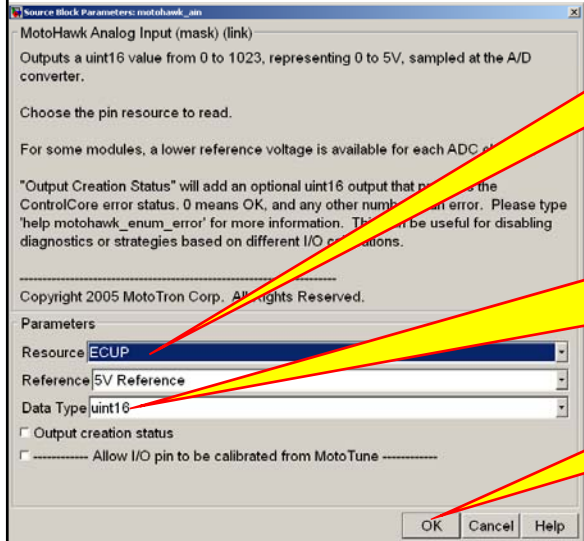
Added analog input blocks.

Double-click on this block to open it.



Analog Inputs

13



MotoHawk Analog Input (mask) (link)

Outputs a uint16 value from 0 to 1023, representing 0 to 5V, sampled at the A/D converter.

Choose the pin resource to read.

For some modules, a lower reference voltage is available for each ADC channel.

"Output Creation Status" will add an optional uint16 output that provides the ControlCore error status. 0 means OK, and any other number means an error. Please type 'help motohawk_enum_error' for more information. This can be useful for disabling diagnostics or strategies based on different I/O operations.

Copyright 2005 MotoTron Corp. All rights Reserved.

Parameters

Resource: **ECUP**

Reference: 5V Reference

Data Type: uint16

☐ Output creation status

☐ Allow I/O pin to be calibrated from MotoTune

OK Cancel Help

Change this to AN4M.

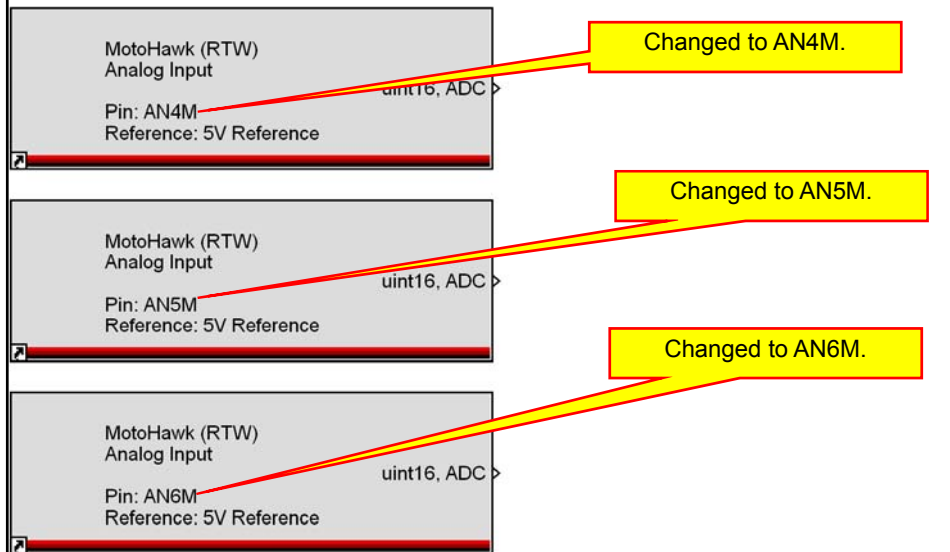
Note that the output data type is uint16. This is an unsigned 16-bit integer. The 10-bit conversion will fit in a uint16 without any problems.

Click the OK button after changing the resource to AN4M.

Analog Inputs

14

- Modify the three Analog Input Blocks as shown.



MotoHawk (RTW) Analog Input

Pin: AN4M

Reference: 5V Reference

uint16, ADC

Changed to AN4M.

MotoHawk (RTW) Analog Input

Pin: AN5M

Reference: 5V Reference

uint16, ADC

Changed to AN5M.

MotoHawk (RTW) Analog Input

Pin: AN6M

Reference: 5V Reference

uint16, ADC

Changed to AN6M.



Analog Inputs

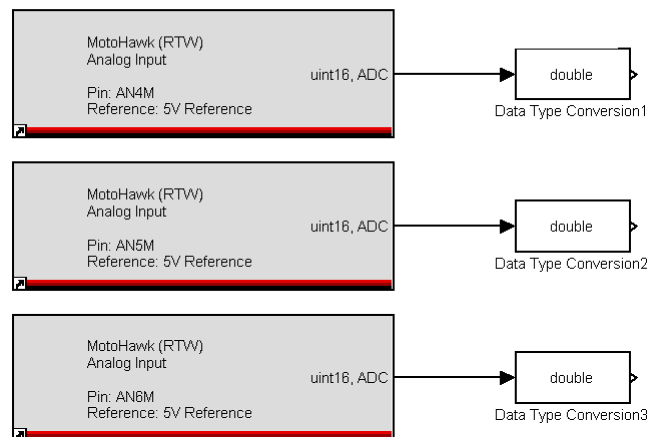
15

- The output of the analog blocks is a 16-bit unsigned integer data type (uint16).
- The rest of our model uses double precision floating point data types.
- Add three data type conversion blocks to the model (library **Simulink / Commonly Used Blocks**):
 - Double-click on the Convert block and specify the type as double.
- You should have the model shown on the next slide.



Analog Inputs

16



Analog Inputs

17

- The output of the analog input blocks is a number from 0 to 1023.
- We want to scale this value to a number between 0 and 1.
- We will do this by using a gain block with a gain of $1/1023$:
- Add Gain blocks and Goto blocks as shown next:

MotoTron

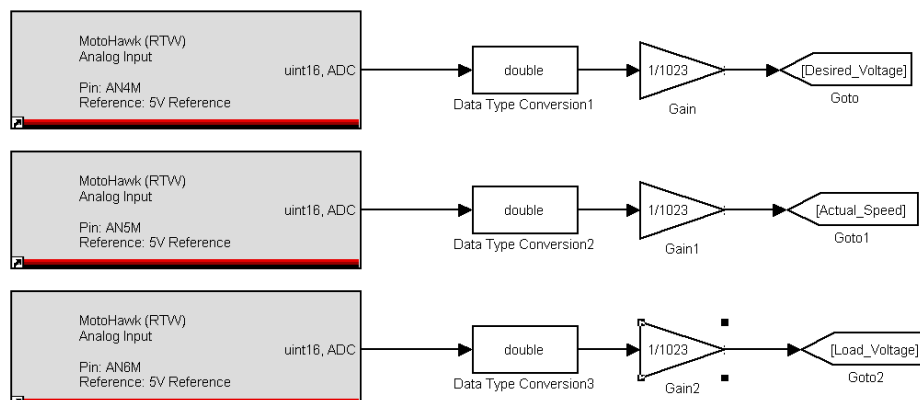
 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Analog Inputs

18



MotoTron

 **The MathWorks**

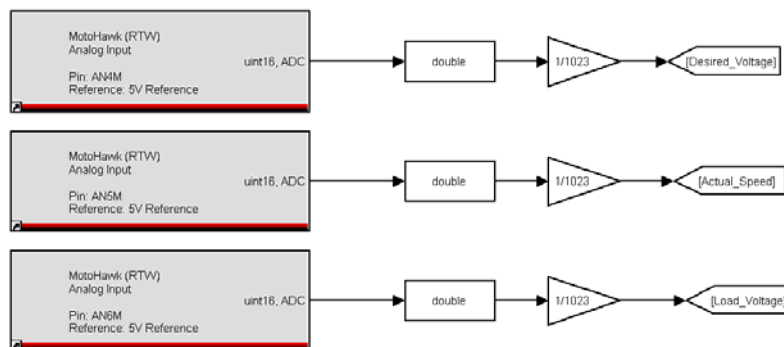
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Analog Inputs

19

- I will hide the block names on the model:
 - Select the part
 - Right click on the part
 - Select **Format** and then **Hide Name** from the menus



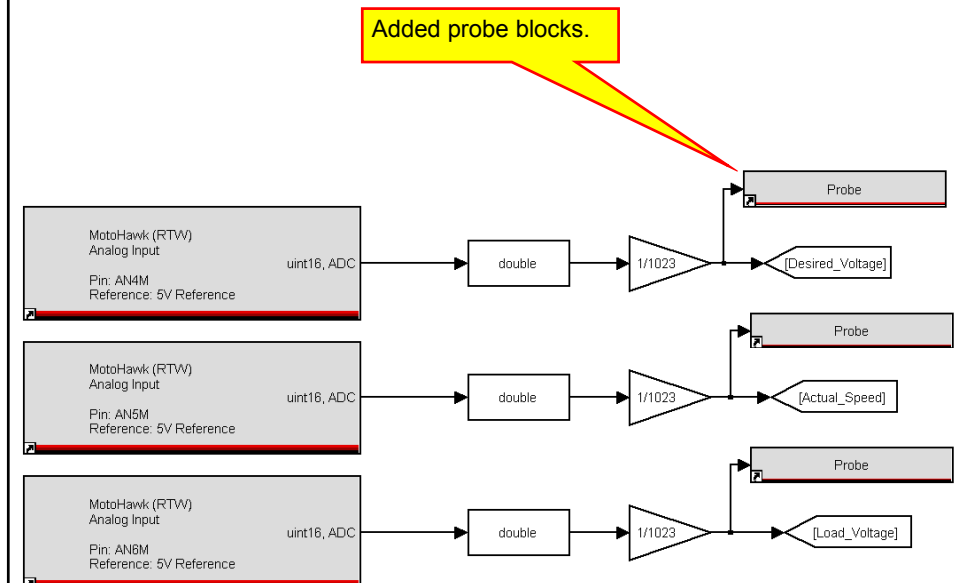
Analog Inputs

20

- We would like to observe these three signals in real-time as the controller is running on our ECU.
- We can observe the signals with MotoTune if we place MotoHawk Probes in our model.
- From the **MotoHawk / Probing & Calibration Blocks** library, place three instances of the block called `motohawk_probe` in your mdoel.

Analog Input

21



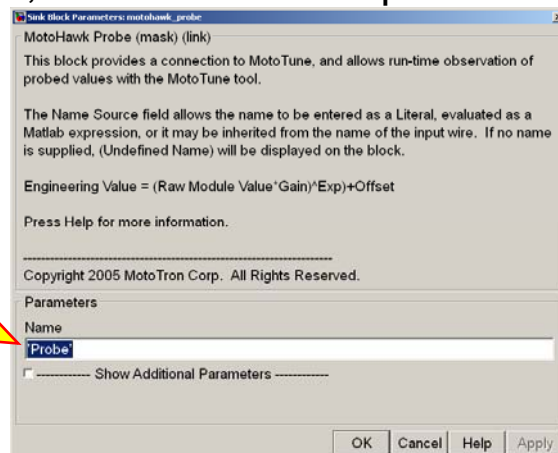
MotoHawk Probes

22

- We need to name each of the probes with a unique name.
- To name a probe, double-click on a probe block.

Change the Probe name here. Note:

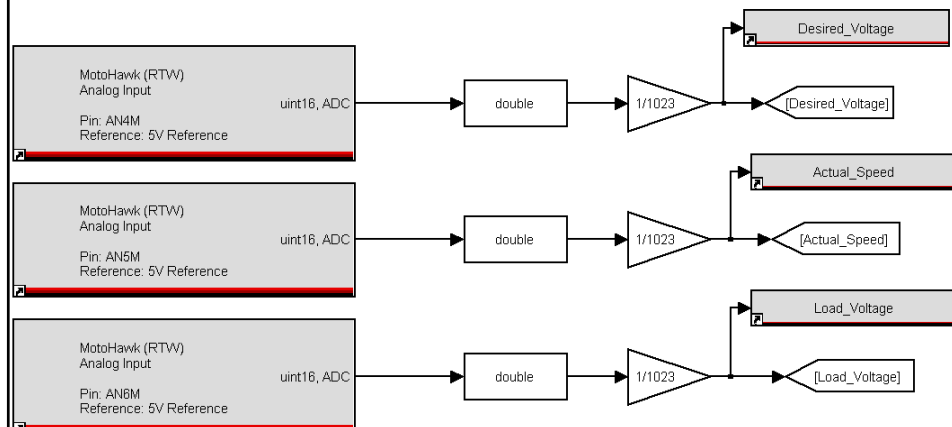
- The name must be enclosed in single quotes.
- There must be no spaces in the name.
- Underscores are OK.



MotoHawk Probes

23

- Change the Probe names as shown:



PWM Output

24

- We now have the analog inputs we need to run our system.
- Next, we need a PWM output to drive our system.
- MotoTron ECUs have several high current PWM drivers that we can use.
- For example, the ETC (electronic throttle control) output can drive a 5 A load.
- The OILP output can drive a 10 A load.



PWM Output

25

- Our system already has high current drivers, so all we need to do is supply a low current pwm signal.
- We could use a high current PWM output such as ETC, but we'll save that in case we need it.
- We will use the EST outputs because they have active pull-up and pull-down drive transistors. (Some of the outputs only have a low side driver and would need a pull-up resistor for our application.)
- Also, the EST outputs produce 5V amplitude PWM signals, and this is compatible with our hardware.
- The portion of the 555-80 data sheet is shown next for the EST outputs:



PWM Output

26



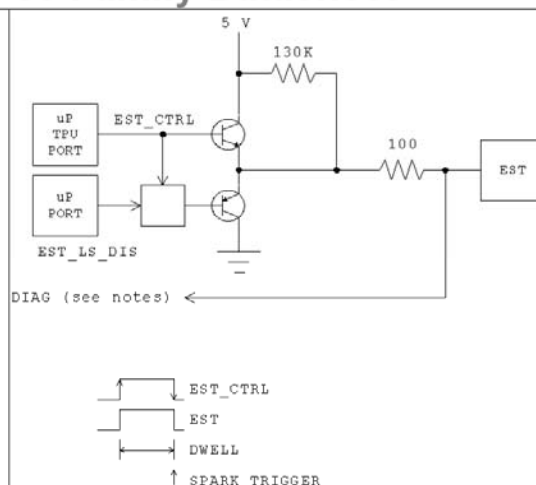
ECU555-80 Family Datasheet

2.12 EST1...EST8

(B2, C8, B10, C7, B9, C12, C13, C14)

These are TTL level outputs.

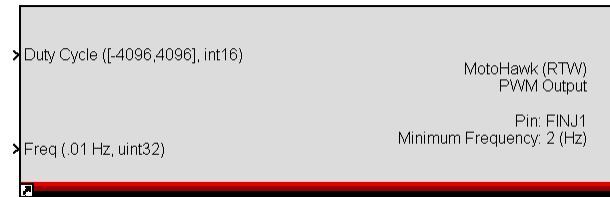
Note: Since EST_RTIN (2.13) is a direct path to the ECU ground care must be taken not to introduce ground loops. EST_RTIN is not designed to carry any significant current, it is a reference only. It should be open circuit unless the smart coil electronics provides an isolated logic ground reference. Care must also be taken not to introduce noise on EST_RTIN. Electrical transients on EST_RTIN can cause module upsets. The STOP signal (1.13) will disable these outputs when asserted.



PWM Outputs

27

- Eight EST outputs are available.
- We will use EST1 for this application.
- To access this resource, place a part called `motohawk_pwm` in your model.
- This part is located in the **MotoHawk \ Analog I/O Blocks** library.



PWM Output

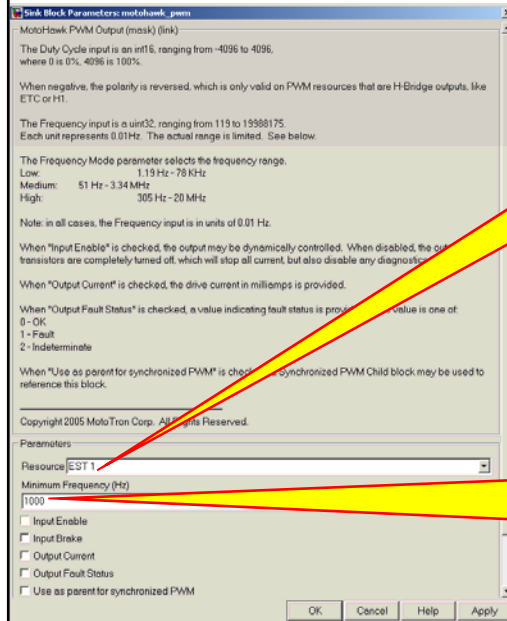
28

- The first thing we need to do is specify EST1 as the output pin we want to use.
- Double-click on the PWM block and change the parameters as shown:



PWM Output

29



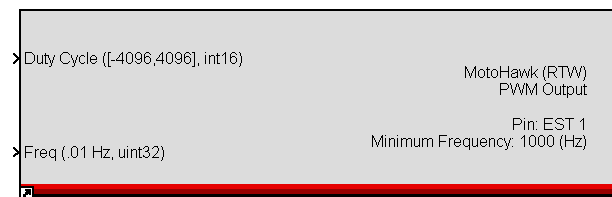
Specify the Resource as EST1.

Specify the minimum frequency as 1000. (If you are not using MotoHawk 084 Beta 9 or higher, you will get a different dialog box here.)

PWM Output

30

- We notice a few things about the PWM block.
 - The duty cycle input is a number from -4096 to 4096. Its type is a signed 16-bit integers (int16)
 - For full bridge outputs (like ETC), we can have a negative or positive output. A duty cycle of -4096 means full negative output.





PWM Output

31

- Duty Cycle Input Continued
 - Since we are using a half bridge, we will only use the positive half of the duty cycle input range.
 - A duty cycle input of -4096 to 0 produces an output that is always low.
 - A duty cycle input from 1 to 4096 produces a PWM output with a duty cycle from 0 to 100%



PWM Output

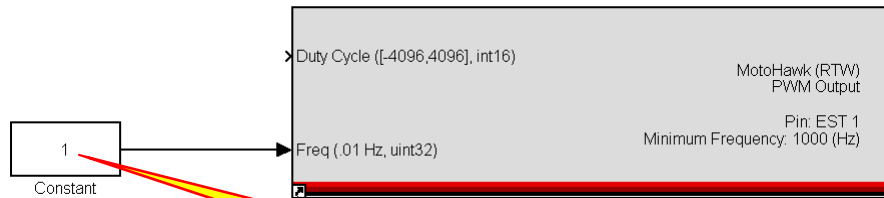
32

- The frequency input is a 32-bit unsigned integer (uint32).
- An input of 100 produces a frequency of 1 Hz.
- For this input, we need to multiply the desired PWM frequency by 100.
- We would like to use a PWM frequency of 20 kHz because it is inaudible to humans.
- We will use a constant block to specify the PWM Frequency as shown:



PWM Frequency

33



Double-click here to open this block.

Specify the value as 20000*100 to specify a PWM frequency of 20 kHz.

MotoTron

The MathWorks

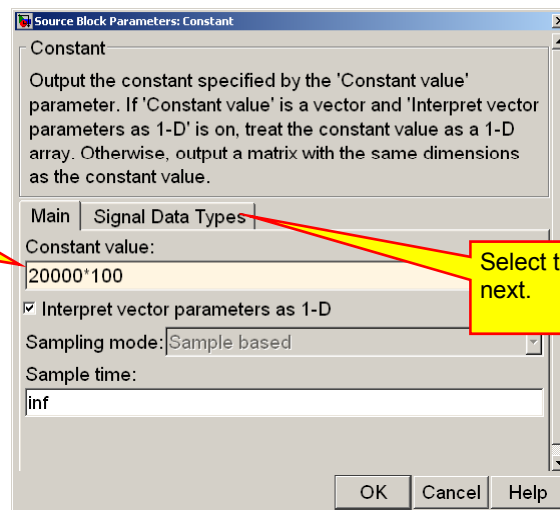
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

PWM Frequency

34

Frequency value specified here.



Select this tab next.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



PWM Frequency

35

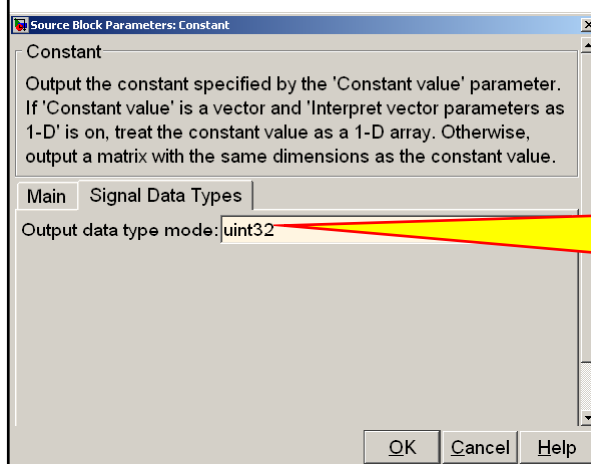
- The frequency input needs a uint32 data type.
- We could do this with a data type conversion block.
- Instead we will do it with the Signal Data Types tab in the constant block:



PWM Frequency

36

- Select the Signal Data Types tab and select the uint32 data type:



Uint32 specified.

Click the **OK** button when done.



PWM Duty Cycle

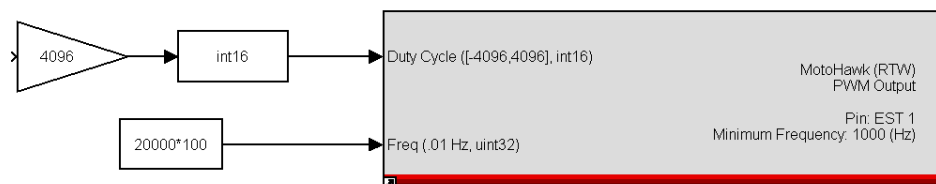
37

- The signal from our controller will have the following properties:
 - Numerical value from 0 to 1.
 - Data type: floating point double precision
- Add a gain block to scale values of 0 to 1 to values from 0 to 4096.
- Add a data conversion block to convert a double precision number to a type of int16.



PWM Duty Cycle Input

38





MotoHawk Override

39

- Eventually, the PWM signal will come from our controller.
- For now, we would like to manually control the duty cycle through MotoTune.
- Place a constant with a value of 0 and a part called `motohawk_override_abs` from the **MotoHawk / Calibration & Probing Blocks** library
- Wire your model as shown:

MotoTron

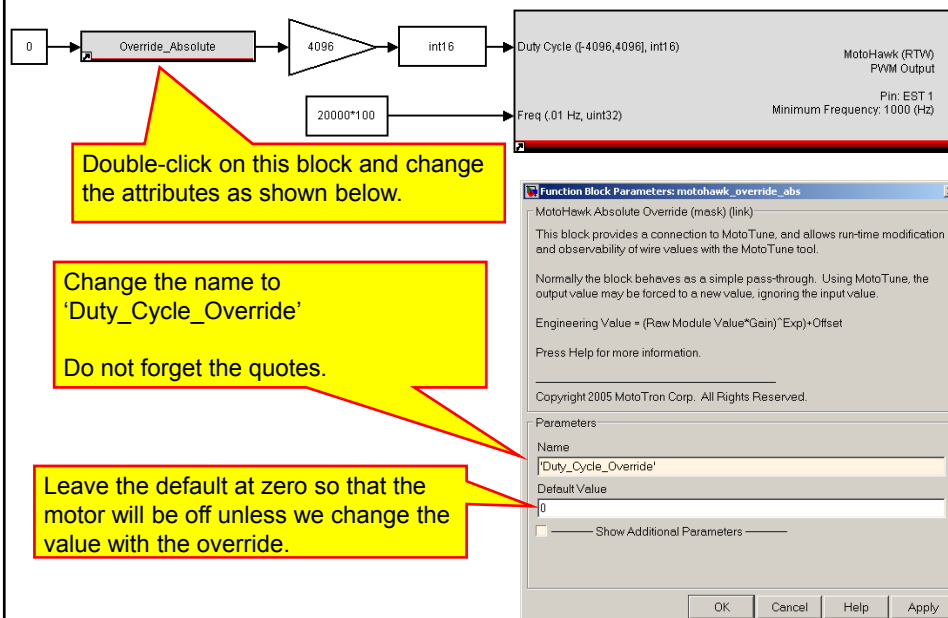
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

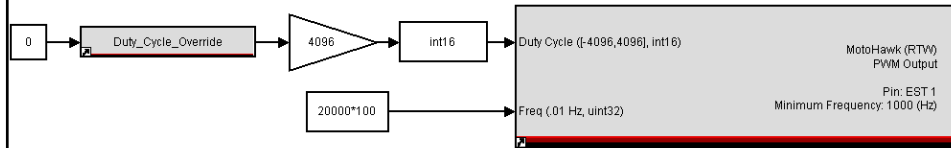
MotoHawk Override

40



MotoHawk Override

41



- Before we build the model we need to check it for errors.
- Select **Format**, **Port/Signal Display**, and then **Port Data Types** from the Simulink menus to display data types on the signal lines.
- Type ctrl-d to check the model for errors.

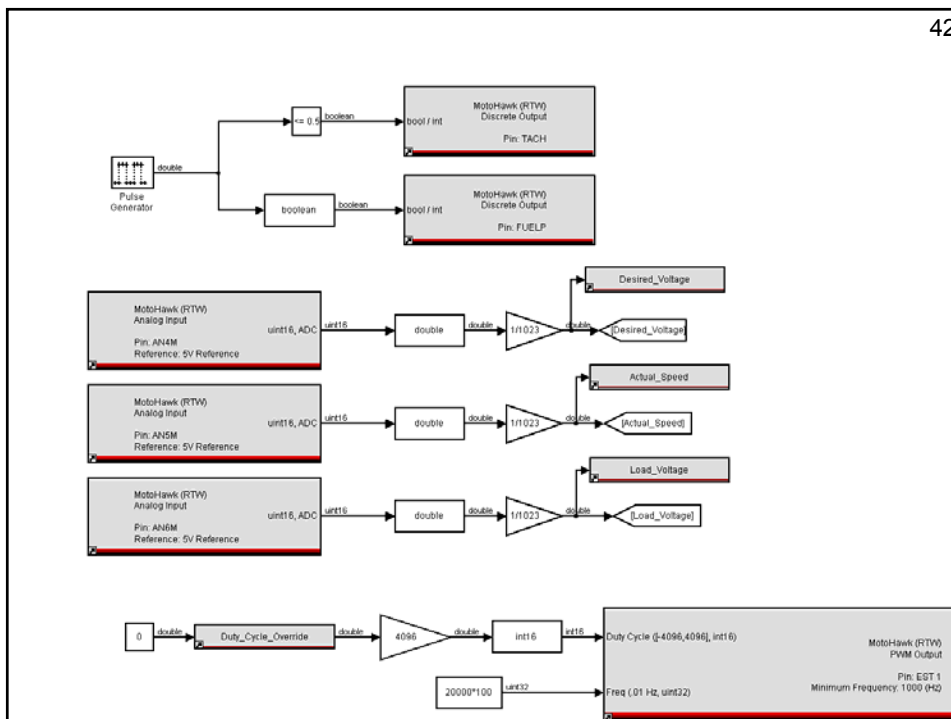
MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

42



Model Build

43

- If:
 - you did not receive any errors after using the ctrl-d command, and
 - Your data types match the ones shown
- Type ctrl-b to build your model.
- Switch to the Matlab command window to view the progress.

```
#####
###
### Target Module: ECU555-80
### Created Development Executable: Motor_Control_MH2_000.srz
### Created MotoTune DLL: Motor_Con_000.dll
###
### Successful MotoHawk Build for model: Motor_Control_MH2
###
#####
```

Wiring Connections

44

- The final step we need to take is physically connect the ECU to our motor-generator system with wires in the development harness.
- The first wire we need to connect is our transducer ground reference. This wire is called XDRG and is pin A22, wire 22 and is Black/Orange.

A22	XDRG	Transducer Ground	Ground Return for Transducers	22 Black/ Orange
-----	------	-------------------	-------------------------------	------------------

- Connect this wire to one of the GND connections on the motor-generator circuit board





PWM Connection

45

- From the portion of the ECU555-80 datasheet we saw earlier, we chose EST1 which was pin B2.
- Looking further down the datasheet, we see the following information

B2	EST1	Electronic Spark Timing	SmartCoil Driver	34 Green/Black
----	------	-------------------------	------------------	----------------

- We see that this is a green/black wire and is numbered 34.
- Connect this wire to the one labeled “PWM” on the motor/generator PC board.



POT Connection

46

- From the portion of the ECU555-80 datasheet we saw earlier, we chose AN4M which was pin A6.
- Looking further down the datasheet, we see the following information

A6	AN4	Potentiometer Input	220K Pull Down	6 Light Blue/White
----	-----	---------------------	----------------	--------------------

- We see that this is a light blue/white wire and has the number 6.
- Connect this wire to the one labeled “POT” on the motor/generator PC board.





RPM Connection

47

- From the portion of the ECU555-80 datasheet we saw earlier, we chose AN5M which was pin A7.
- Looking further down the datasheet, we see the following information

A7	AN5	Potentiometer Input	220K Pull Down	7 White/Yellow
----	-----	---------------------	----------------	----------------

- We see that this is a white/yellow wire and has the number 7.
- Connect this wire to the one labeled “RPM” on the motor/generator PC board.



Voltage Connection

48

- From the portion of the ECU555-80 datasheet we saw earlier, we chose AN6M which was pin A8.
- Looking further down the datasheet, we see the following information

A8	AN6	Potentiometer Input	220K Pull Down	8 Brown/White
----	-----	---------------------	----------------	---------------

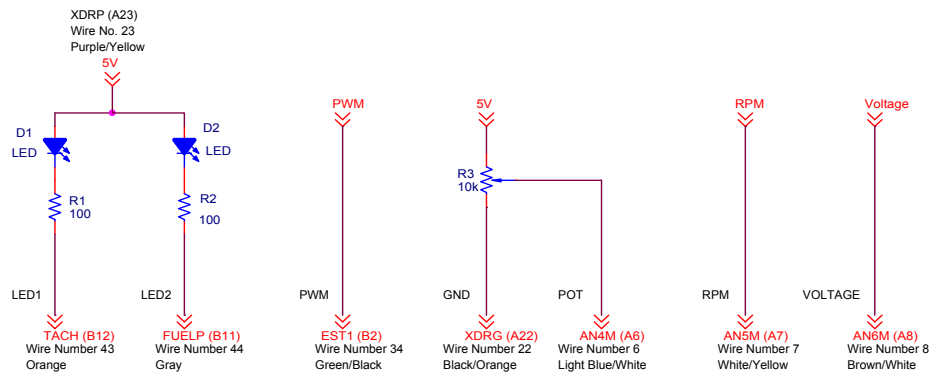
- We see that this is a brown/white wire and has the number 8.
- Connect this wire to the one labeled “VOLTAGE” on the motor/generator PC board.



Circuit Diagram

49

- A complete circuit diagram for our connections is shown below.



MotoTron

The MathWorks

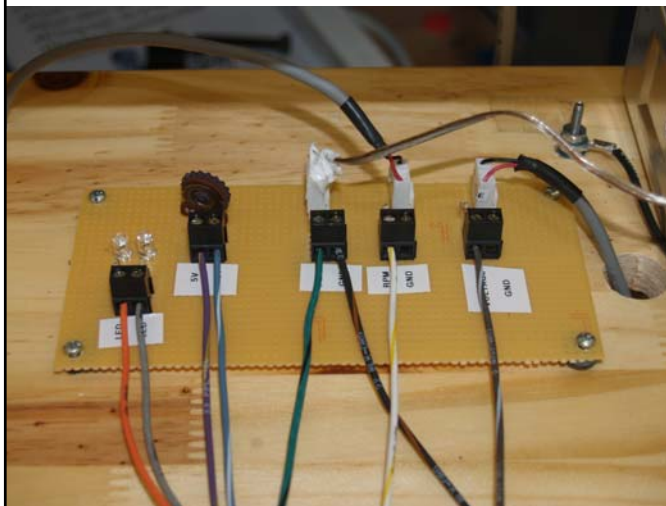
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Wiring Connections

50

- A picture of the wiring connections is shown below:





MotoTune

51

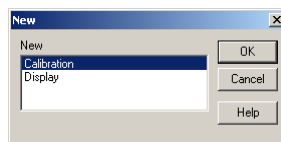
- We are now ready to program the ECU with MotoTune.
- Use the procedure covered earlier
 - Run MotoTune
 - Select **File** and then **Program** from the MotoTune menus
 - Select the most recent executable version of our MH2 model.
- When the programming is complete, click the **OK** button to close the dialog box.



MotoTune Display

52

- You should have an empty MotoTune window.
- Select **File**, **New**, and then **Online Display/Calibration** or from the menus.



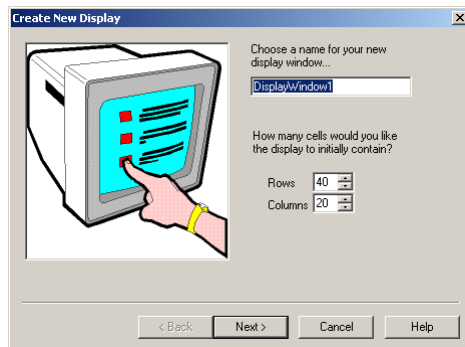
- Select **Display** and click the **OK** button.



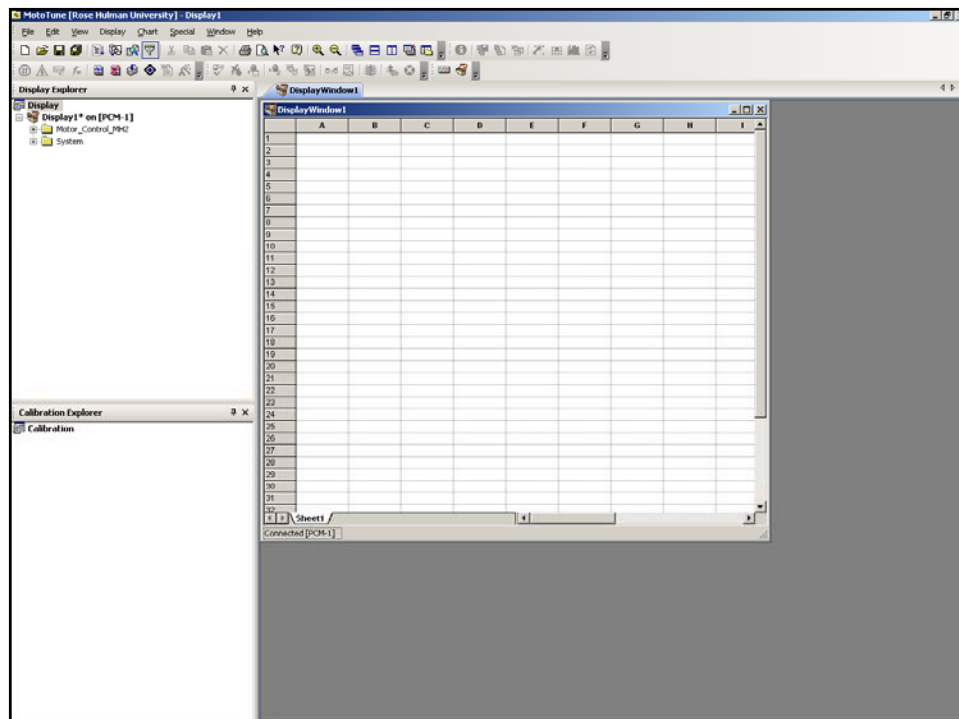


MotoTune Display

53



- Click the **Next** button twice and then click the **Finish** button.
- You should have the screen shown next:

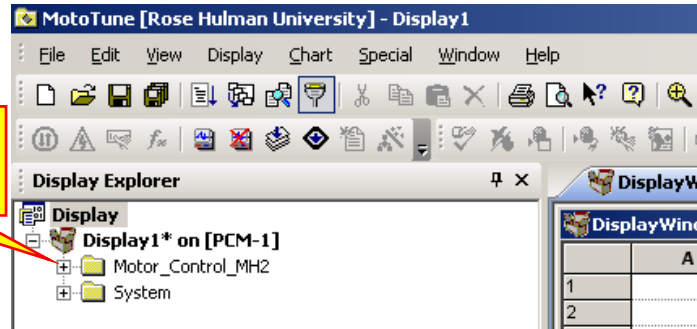


MotoTune

55

- Click on the plus sign(s) next to Motor_Control_MH2 to expand the tree.

Click on this + sign and then expand the tree fully.



MotoTron

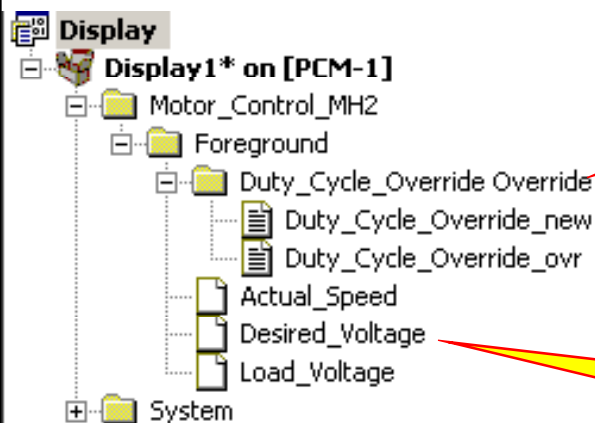
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoTune Display

56



MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTune Display

57

- You can display as many or as few of the probes and overrides as you want.
- To place an item in your display, drag the item into the display window.
- If you drag an individual item, only that item will be displayed.
- If you drag a folder, all probes and overrides in the folder will be displayed.

MotoTron

 **The MathWorks**

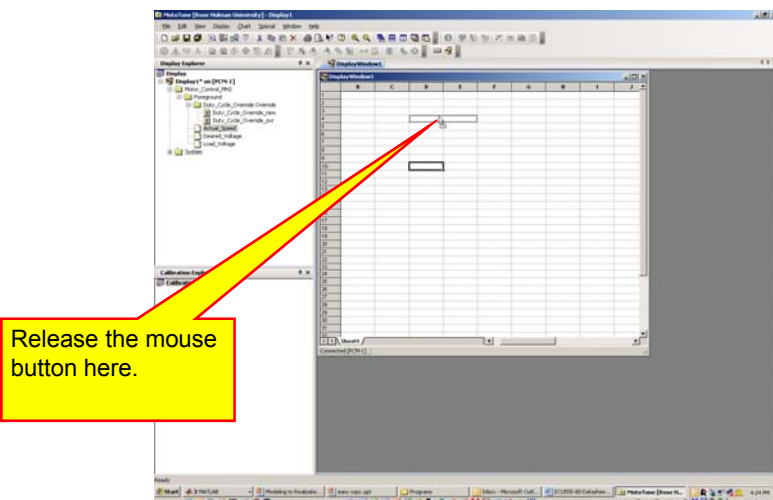
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoTune Display

58

- Drag the item Actual_Speed to the display window as shown below:





MotoTune Display

59

- When you release the mouse button, the item and its value will be displayed.
- Resize the cells to show the entire text labels.

	B	C	D	E	F	G	H
1							
2							
3							
4			Actual_Speed	0.00			
5							
6							
7							

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

MotoTune Display

60

- Drag the probes Desired_Voltage and Actual_Voltage to the display window.

	B	C	D	E	F	G	H
1							
2							
3							
4			Actual_Speed	0.00			
5			Desired_Voltage	0.00			
6			Load_Voltage	0.00			
7							

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY



MotoTune Display

61

- For the override, we need both items.
- The easiest way to do this is by dragging the folder Duty_Cycle_Override Override to the display window. Both items will appear in the window:

	B	C	D	E	F	G	H
1							
2							
3							
4			Actual_Speed	0.00			
5			Desired_Voltage	0.00			
6			Load_Voltage	0.00			
7							
8							
9			Duty_Cycle_Override_new	0.00			
10			Duty_Cycle_Override_ovr	Pass-Through			
11							
12							

MotoTune Display

62

- The next thing we need to do is change the speed at which the displayed values are updated.
- Right click on the cell as shown below:

Actual_Speed	0.00
Desired_Voltage	0.00
Load_Voltage	0.00
Duty_Cycle_Override_new	0.00
Duty_Cycle_Override_ovr	Pass-Through

Right-click here and then select **Properties** from the menu.

MotoTron

The MathWorks

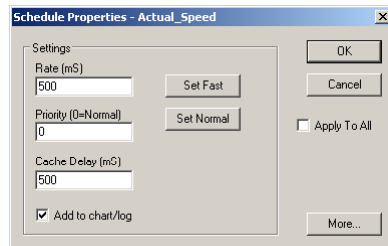
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



MotoTune Display

63

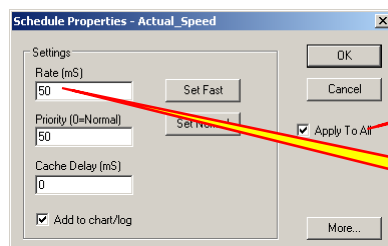


- By default, MotoTune updates all of the values at a 500ms rate.
- This is too slow for our example.
- Click the **Set Fast** button and select the **Apply To All** option



MotoTune Display

64



Option selected.

Values will be displayed at a 50 ms rate.

- We have specified that all values in the display window will be updated every 50 ms.
- The downside is that this information is communicated over the CAN bus.
- By displaying a lot of items and updating them at a high rate, we are increasing the amount of CAN traffic on the bus.





MotoTune Display

65

- The model is running on our ECU.
- If you rotate the pot, you should see the value on the Desired_Voltage probe change between 0 and 1.

DisplayWindow1			
	A	B	C
1			
2			
3			
4			
5		Actual_Speed	0.00
6		Desired_Voltage	0.37
7		Load_Voltage	0.00
8			
9			
10		Duty_Cycle_Override_new	0.00
11		Duty_Cycle_Override_ovr	Pass-Throu

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoTune Override

66

- Right now, the override is set to Pass-Through.
- From the model, the value we specified is 0, so the motor will be off.

DisplayWindow1			
	A	B	C
1			
2			
3			
4			
5		Actual_Speed	0.00
6		Desired_Voltage	0.37
7		Load_Voltage	0.00
8			
9			
10		Duty_Cycle_Override_new	0.00
11		Duty_Cycle_Override_ovr	Pass-Throu

Click here to reveal the options.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoTune Override

67





Actual_Speed	0.00
Desired_Voltage	0.43
Load_Voltage	0.00
Duty_Cycle_Override_new	0.00
Duty_Cycle_Override_ovr	Pass-Through

After selecting the cell, options are now available.

Select **Override** and press the enter key.

Actual_Speed	0.00
Desired_Voltage	0.43
Load_Voltage	0.00
Duty_Cycle_Override_new	0.00
Duty_Cycle_Override_ovr	Override

Override now selected.

MotoTune Override

68

Actual_Speed	0.00
Desired_Voltage	0.43
Load_Voltage	0.00
Duty_Cycle_Override_new	0.00
Duty_Cycle_Override_ovr	Override

Enter a value here.





When you press the enter key, the output of the override block will be changed to the value you enter.

Actual_Speed	0.00
Desired_Voltage	0.43
Load_Voltage	0.00
Duty_Cycle_Override_new	0.00
Duty_Cycle_Override_ovr	Override

Enter a value between 0 and 1. Zero is full off, 1 is full on.

The motor should spin and you should see all of the other probe values change.

If you have an oscilloscope, you can observe the PWM output and see the PWM waveform as the duty cycle changes.



MotoTune Override

69

Actual_Speed	0.28
Desired_Voltage	0.64
Load_Voltage	0.30
Duty_Cycle_Override_new	0.20
Duty_Cycle_Override_ovr	Override

- We now know how to use probes and overrides.
- The last thing we will do is save our display.



MotoTune Display

70

- Select **File** and then **Save** from the MotoTune menus.
- Specify a name for your display, like MH2 and click the **Save** button.
- The next time we use this model, we can open the display we saved.
- We can also use the display in future models if we have the same probes and overrides.





Lecture 17 Demo 1

71

- Demo of MotoTune with the Motor/generator system
 - MotoHawk Probes display measurements
 - MotoHawk override controls motor speed.

Demo _____

Controller Implementation

72

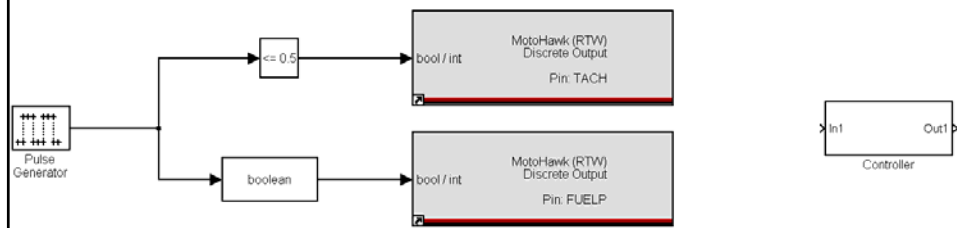
- We now have a MotoHawk shell that has the inputs and outputs that we need to control our system.
- We will now place the controller we designed with simulations in our ECU model.
- We will do this in a subsystem so that we can easily separate our control subsystem from the MotoHawk I/O shell.



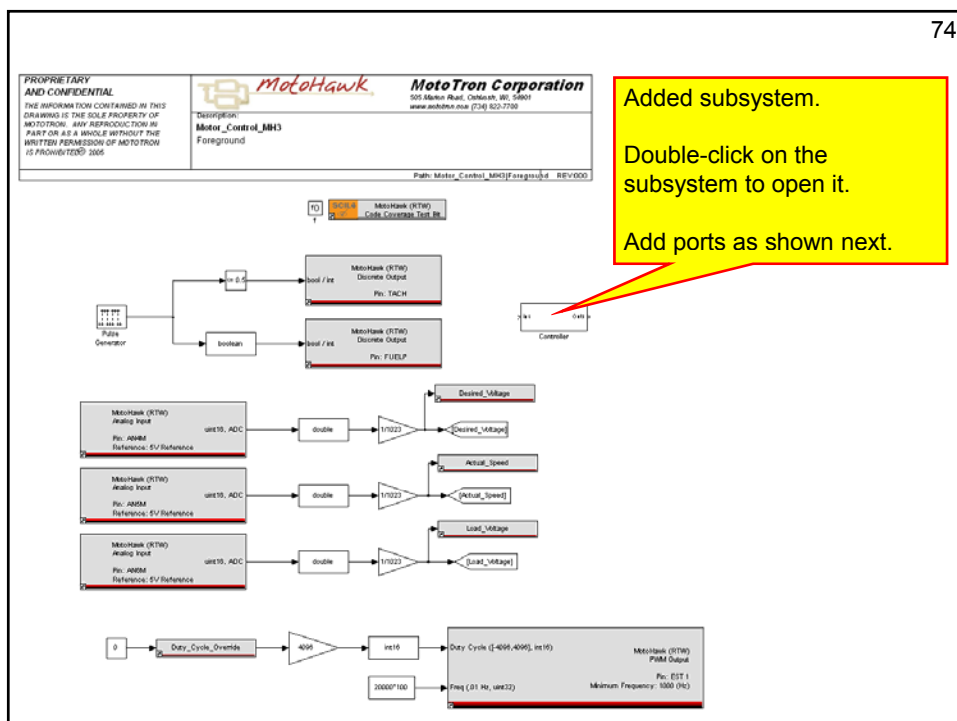
Controller Implementation

73

- Place a subsystem block in the foreground block of our model.
- Subsystem blocks are located in the **Simulink / Commonly Used Blocks** library.
- Rename the subsystem “Controller.”

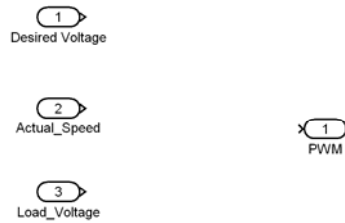


74



Controller Subsystem

75



- In the Foreground subsystem, make the connections shown next to the Controller subsystem.
- Use From and Goto blocks located in the **Simulink / Signal Routing** library.

MotoTron

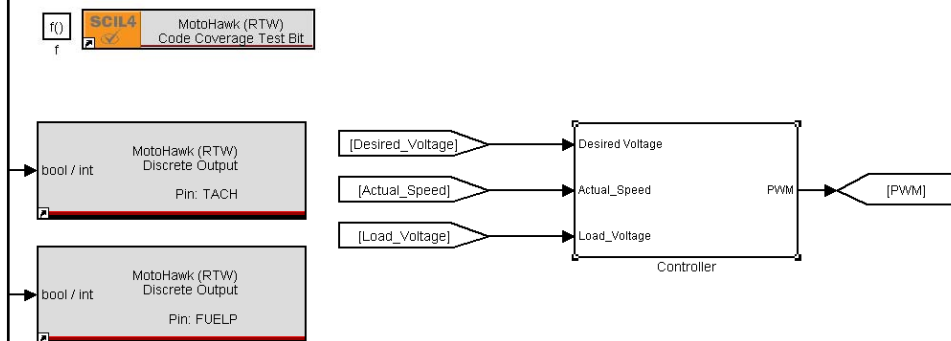
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller

76



MotoTron

The MathWorks

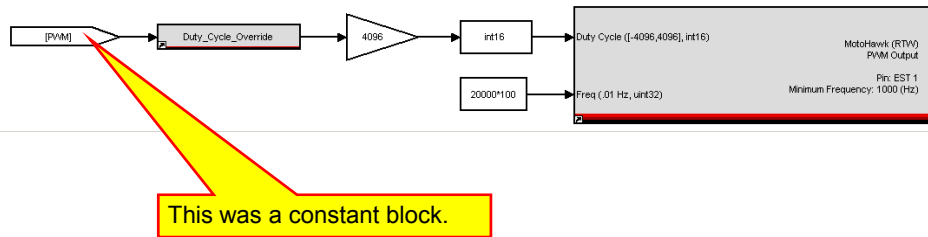
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller

77

- To connect the controller PWM output to the ECU PWM output, replace the constant block with a From block as shown:



MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller

78

- We now have a shell for utilizing the ECU resources, and we have a convenient place to put all of our control logic.
- If we make a change to our control algorithm, we only need to modify the contents of the controller subsystem. The only time we will need to modify our MotoHawk shell is if we need to create additional inputs or outputs.

MotoTron

 **The MathWorks**

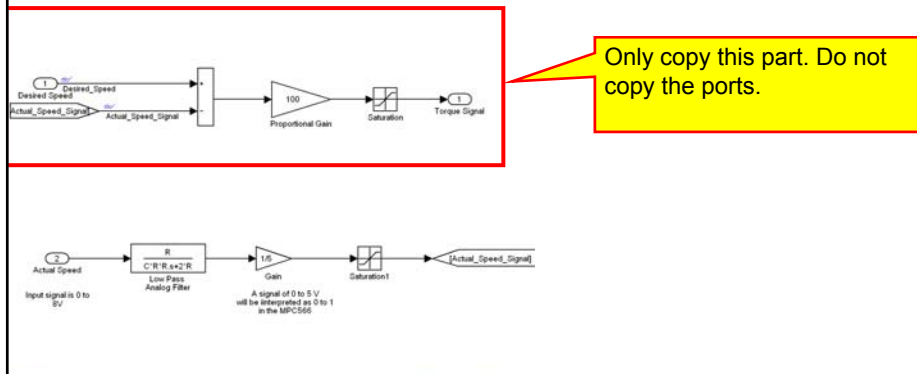
 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Controller

79

- We will start with the proportional feedback method we developed in model Motor_Control_Sim1.mdl.
- The controller portion of the model is shown below.



Controller.

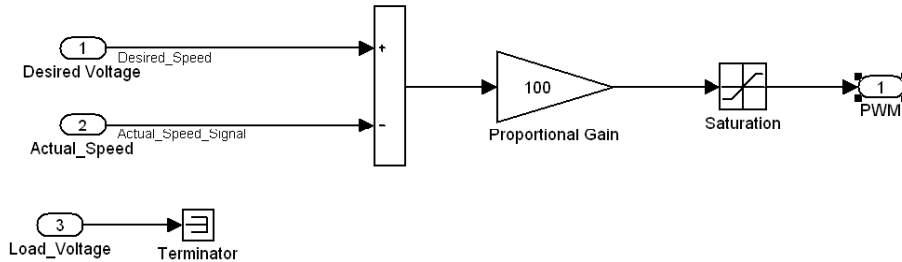
80

- We only need to copy the proportional control portion of the model.
- The low-pass filter is implemented in hardware and is a circuit on the PC board.
- Copy and paste the controller from the simulation into the MotoHawk model as shown:



MotoHawk Controller

81



- The load voltage signal is not being used, so we will connect it to a terminator.
- This controller had a fixed gain of 100.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoHawk Controller

82

- We would like to add the capability of changing the proportional gain while the controller is running.
- This will allow us to tune the controller to our physical system in real time.
- We can do this by adding a MotoHawk Calibration block to our controller.
- The calibration block is located in the **MotoHawk / Calibration & Probing Blocks** library.
- Make the changes shown next.

MotoTron

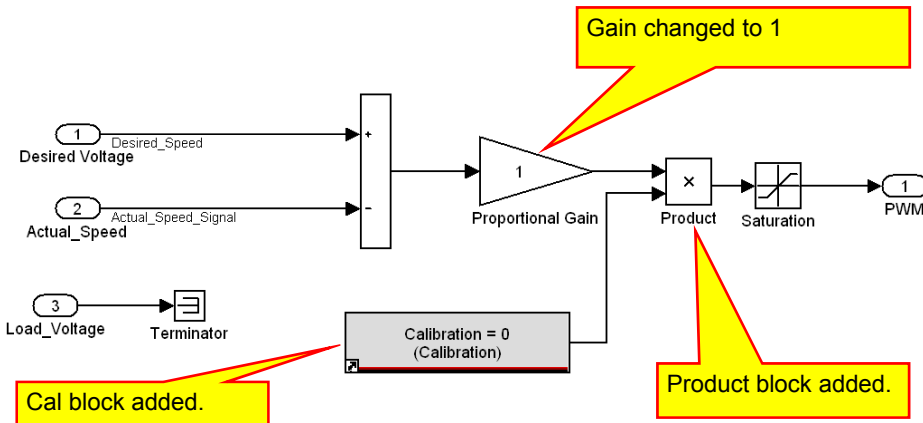
The MathWorks

freescale
semiconductor





ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoHawk Controller

83

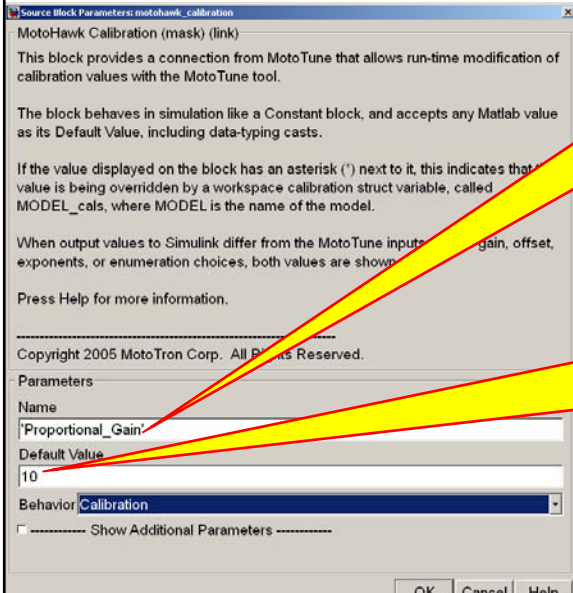


- Double-click on the calibration block and make the changes shown next.

Calibration Block

84



Name changed to 'Proportional_Gain'

Do not forget the quotes. No spaces allowed.

Default value set to 10.

We can change this value while the model is running on the ECU.



Calibrations

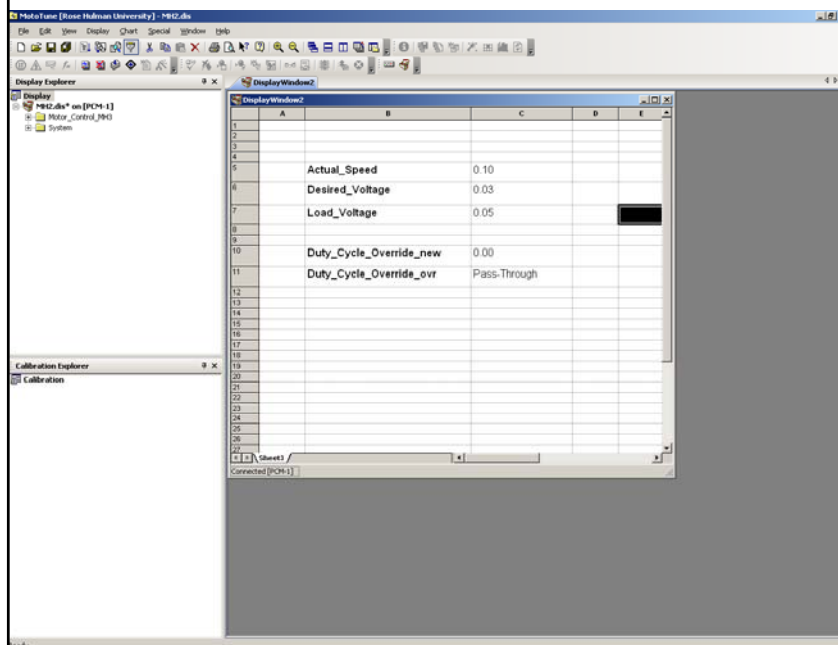
85

- Click the **OK** button.
- Build the model.
- Download the model to your ECU with MotoTune.
- Display all of the probes and overrides as we did in the previous example.
- You can load a saved display from the MotoTune File menu if you want.
- You should have the following MotoTune display.



MotoTune

86





Motor-Generator Operation

87

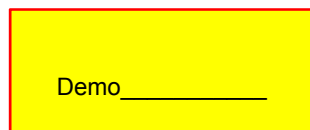
- We can now run our controller on the actual motor-generator system.
- We notice the following:
 - The motor speed follows the setting on the speed pot.
 - The motor speed is held relatively constant as we change the load on the generator. (As it should with a proportional feedback system.)
 - The Motor generator system makes a ton of noise.



Lecture 17 Demo 2

88

- Demo of Motor/generator with the proportional feedback control system.





Motor-Generator Operation

89

- We notice that the motor speed signal is bouncing all around. (It might not do this. We added a flywheel to the motor generator system to fix this problem.)
- If we looked at the motor speed on an oscilloscope, we would notice that the speed signal is oscillating.



Motor-Generator Operation

90

- In our MotoHawk realization, the foreground process only executes every 5 ms.
- This means that the control algorithm executes only once every 5 ms, which may be too slow for the motor/generator system.





MotoHawk – Trigger Definition

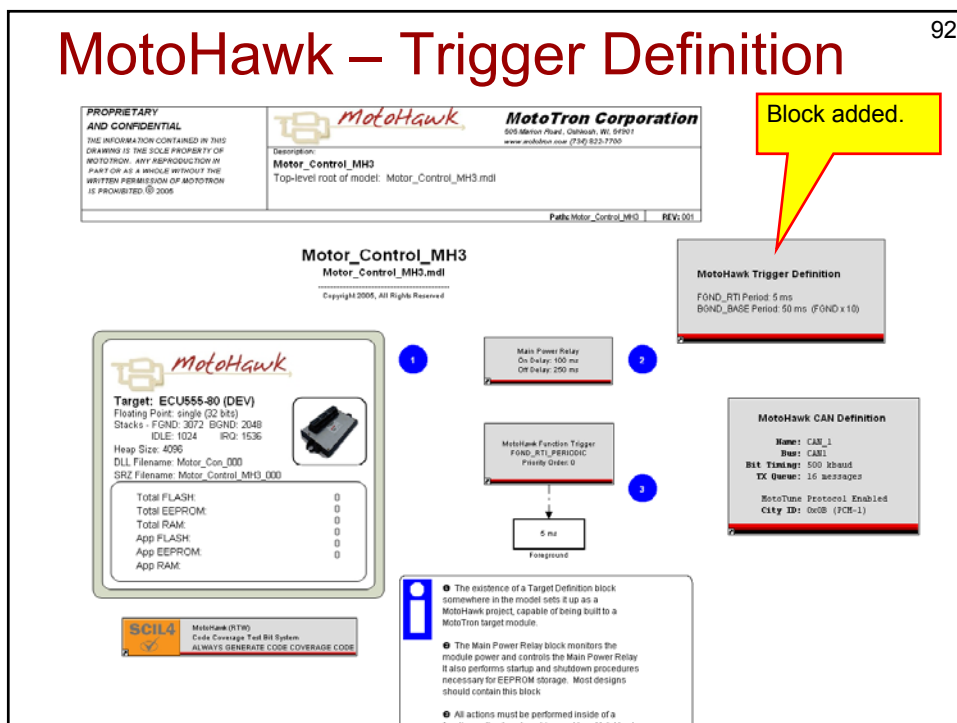
91

- To change the rate at which the foreground process is triggered, we need to place a `motohawk_trigger_def` block in the top level of our model.
- This block is located in the **MotoHawk / Trigger Blocks** library.
- Place the block in the top level as shown next:



MotoHawk – Trigger Definition

92





MotoHawk – Trigger Definition

93

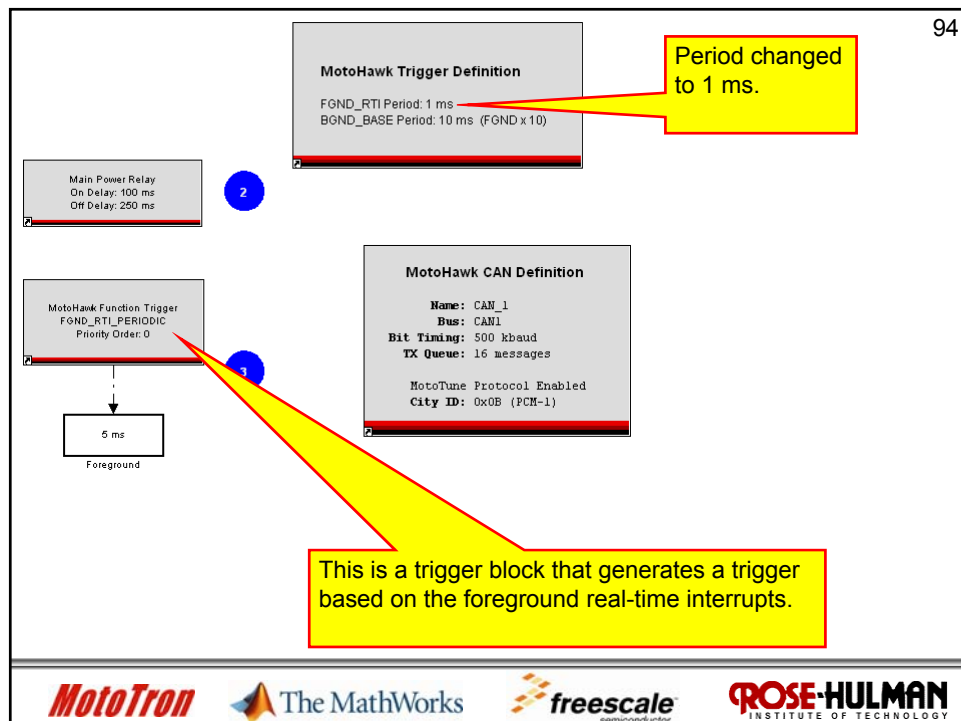
- When we open the block, we see
 - Trigger FGND RTI period is set to 5000 μ s, or 5 ms.
 - FGND RTI stands for foreground real-time interrupt.
 - This trigger will be generated every 5 ms.
 - Change this value to 1000 to generate a trigger every 1 ms.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

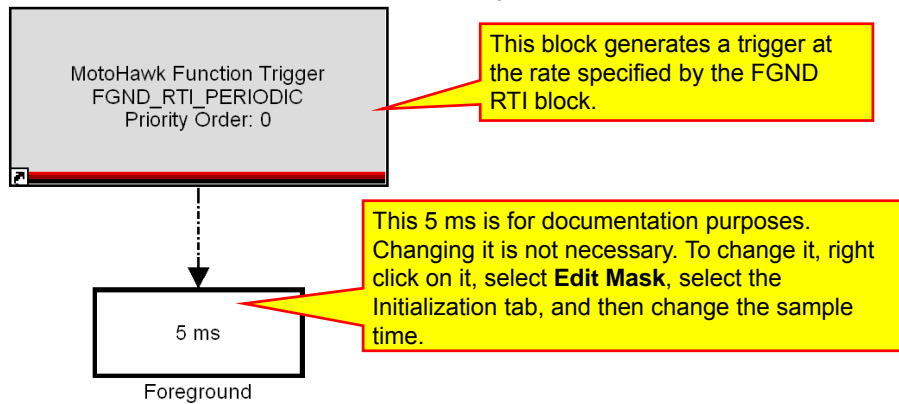




FGND RTI

95

- We see that our foreground subsystem is triggered by the FGND RTI. Since we changed this value to 1 ms, our controller will now execute every 1 ms.



Motor-Generator Operation

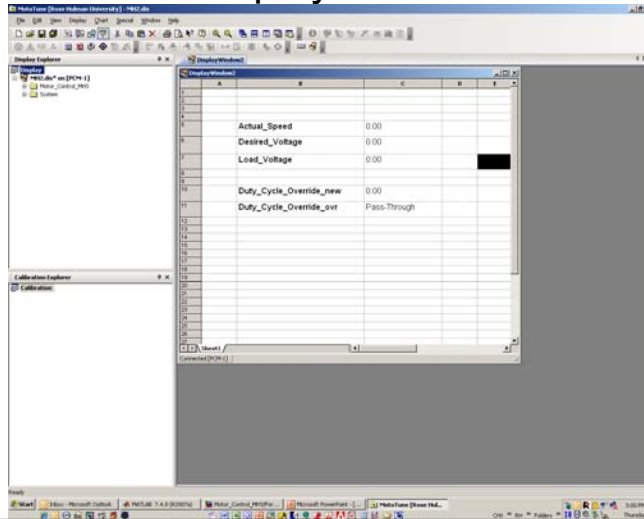
96

- Build the model and download it to the ECU.
- Lets see if the bad behavior has been fixed.
- The oscillation has been eliminated with no load.
- There is still a grinding sound at higher speeds and higher loads.
- We will look at changing the feedback gain to see if it fixes this oscillation.

MotoTune Calibration

97

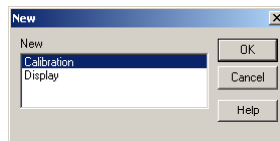
- In the MotoTune window we have all of the probes and overrides displayed.



MotoTune Calibration

98

- We can change the gain by changing the calibration we added in the previous model.
- Select **File**, **New**, and then **Online Display & Calibration** from the MotoTune menus.

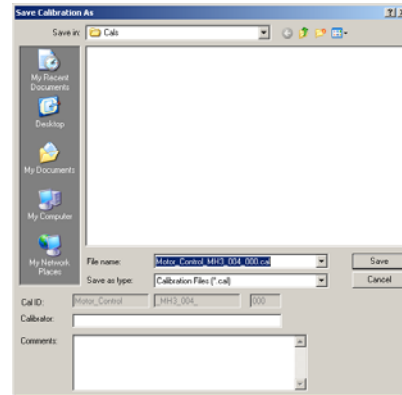


- Select **Calibration** and click the **OK** button

MotoTune Calibration

99

- You can specify a new name or use the one given.
- Click the **Save** button.
- In the future, we will be able to load calibrations that we made earlier.
- We can also back load calibrations into our model (although this process is not recommended).



MotoTron

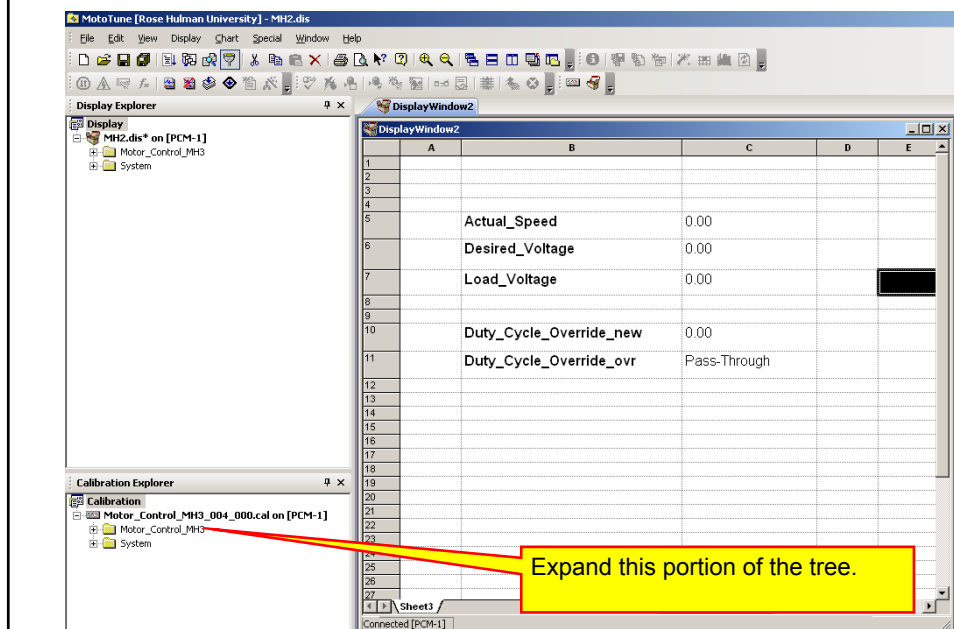
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

MotoTune Calibration

100





MotoTune Calibration

101

Calibration Explorer

Motor_Control_MH3_004_000.cal on [PCM-1]

- This is the calibration part that we added to our model.
- Double-click on it to open a window for this calibration.

Controller

Name	Value	Units	Help
Proportional_Gain	10.00		

- Calibration window that allows us to change the value of the Proportional_Gain.
- Enter a new value in the cell.

When you press the Enter key, the new value will be sent to the ECU.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Calibrations

102

- The Calibrations allow us to try different values of the parameter we are changing.
- We can save the calibrations to a file for later use.
- We can backload the values into our model. (Not recommended.)



Motor-Generator Operation

103

- My system appears to have the following behavior:
- For gains of 1 to 5, the noise disappears.
- For a gain around 10, we hear some noise.
- For gains of 100 and 1000, the oscillation is noticeable for all speeds of operation.



Proportional Feedback

104

- The error signal is the difference between the desired quantity and the actual measured quantity.
 - In our case the error is the difference between the desired voltage signal and the voltage signal representing the rpm.
- A system with proportional feedback will have the following properties:
 - As we increase the gain, the error signal will decrease.
 - For a fixed gain, as we increase the load, the error will increase.





Proportional Feedback

105

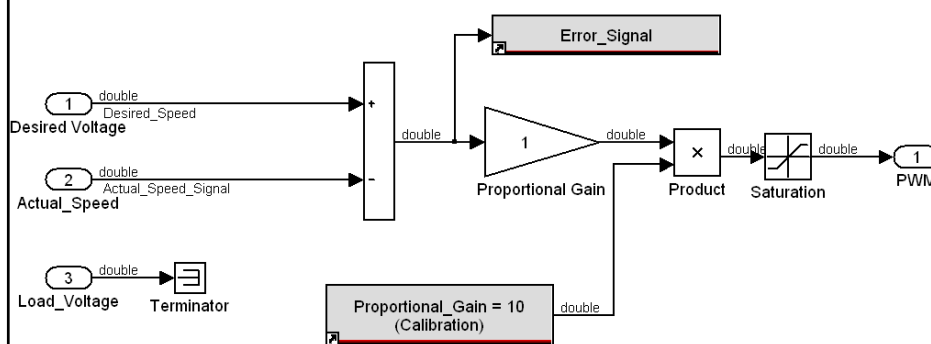
- To make the actual rpm follow the desired rpm (make the error as small as possible) we want to make the gain as large as possible.
- Increasing the gain can make the system unstable.
- We can observe all of these properties in our system.



Proportional Gain

106

- To make the observation of our system a little easier we will add a probe on the error signal as shown below:





Motor-Generator System

107

- Download our change to the ECU.
- Observe that the system exhibits the behaviors discussed.



Lecture 17 Demo 3

108

- Demo of Motor/generator with the proportional feedback control system with the following
 - Foreground process set to a trigger rate of 1 ms
 - Modifying the feedback gain with MotoHawk calibrations.
 - Viewing the Error Signal

Demo_____



Further Investigations

109

- At this point, we know how to use several of the tools available from MotoTron.
- We will investigate the effects of different control methods.



Integral Control

110

- The proportional control method causes the measured signal to follow the desired control signal with a constant amount of error.
- Integral will cause this error to go to zero in steady state.
 - We are controlling rpm.
 - The rpm will equal the command rpm in steady state.
 - When we change the load, the rpm will return to the steady state value after a slight (hopefully) disturbance.



Integral Control

111

- We explored the effects of adding an integrator in our SIL simulations, and in our real-time simulations.
- We will not use the built-in MathWorks discrete integrator because:
 - The integration time step is determined at compilation time. (For our system, it is 1 ms.)
 - This predetermined value is used to calculate the integral even if the actual time between integration steps varies slightly while the model is running.

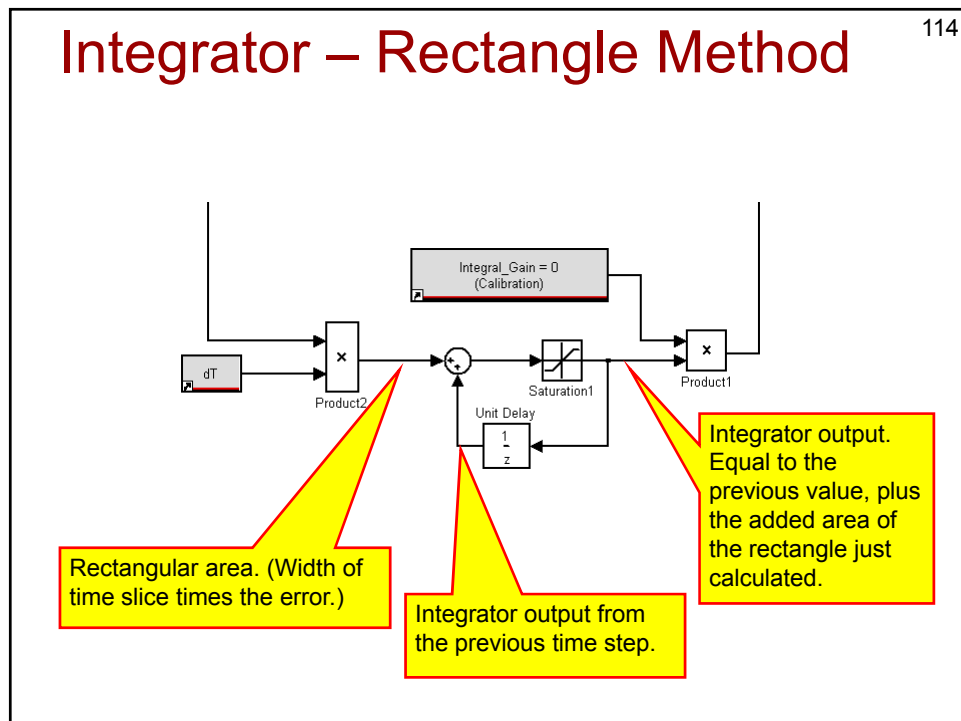
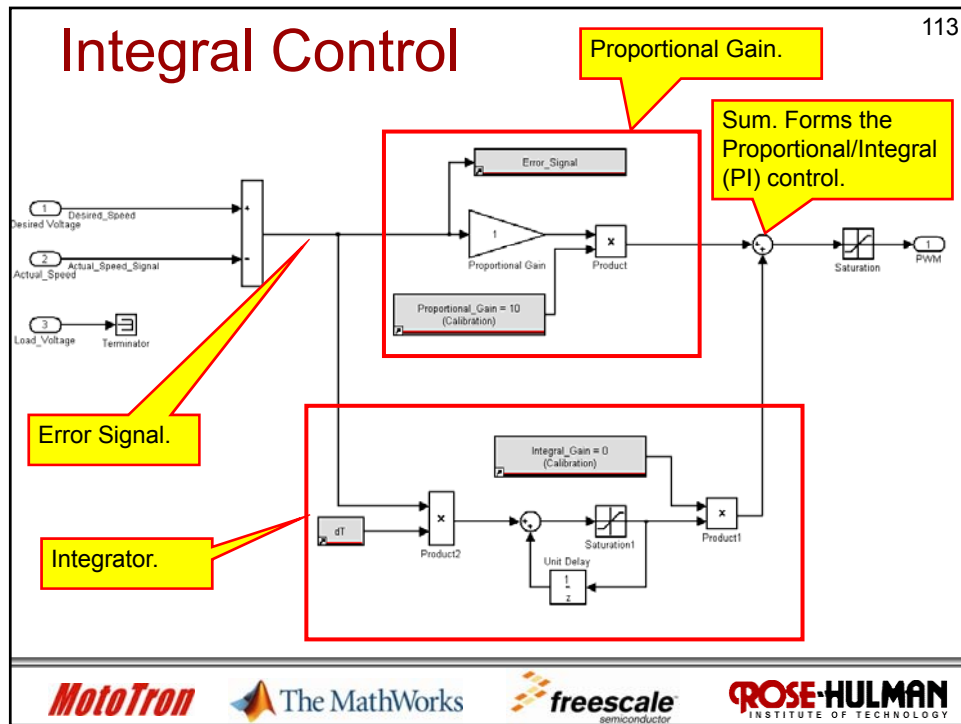


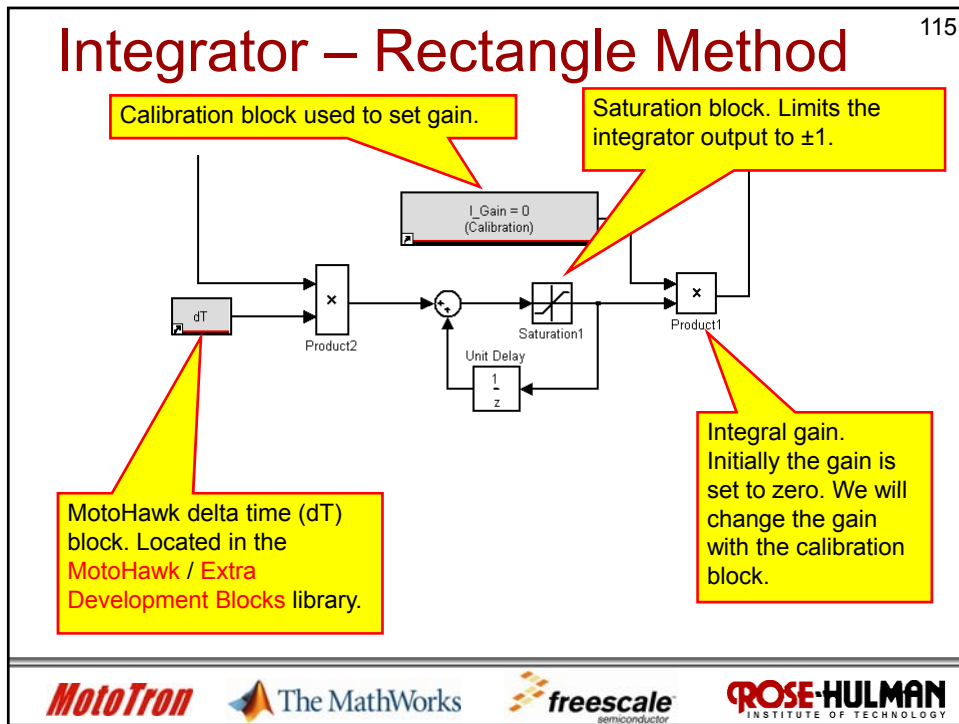
Integral Control

112

- MotoHawk has a block called dT.
- This block returns the actual time difference between when the block was previously executed to the time is currently being executed.
- This block allows us to use the actual time difference in a calculation.
- We will use a rectangular integration method with saturation limits.
- Open your controller and modify as shown:





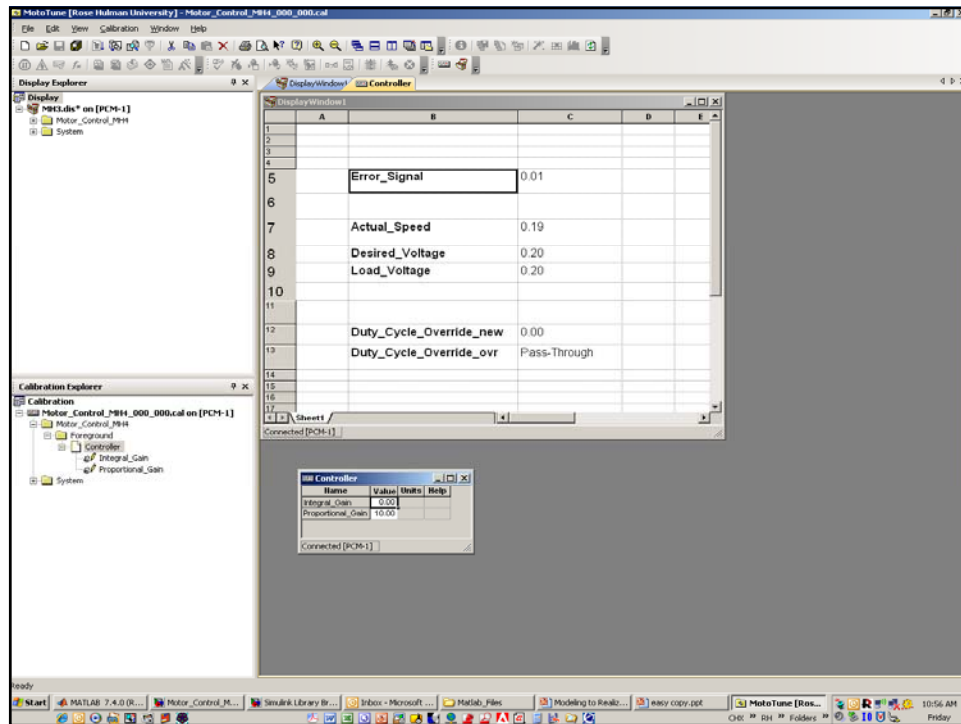


116

Model Build

- Build and download the model to your ECU.
- Display all MotoHawk Probes.
- Display the calibration spreadsheet for the proportional and integral gains.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY



System Behavior

118

- You should notice the following properties of your PI System.
 - The proportional gain behaves similar to our last example.
 - High proportional gain causes the system to oscillate and emit an grinding noise.
 - Low proportional gain produces a large error under high load conditions.
 - The Integral gain
 - Drives the error signal to zero.
 - Causes over and undershoot.
 - Larger values of integral gain cause more over and undershoot, but drive the error to zero faster.



Lecture 17 Demo 4

119

- Demo of Motor/generator with the PI feedback controller.

Demo _____

Calibration Problem

120

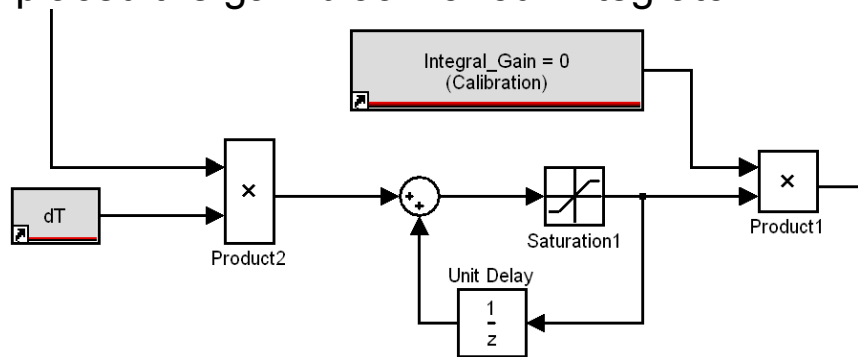
- One thing you might have noticed is that when you changed the value of the integral gain calibration, the motor speed jumped (sometimes extremely high) and then slowly game back to the rpm it was set at.
- Imagine if you were in a vehicle and you changed the gain with a calibration and a 50 hp engine or electric motor made a similar jump in speed and torque.



Calibration Problem

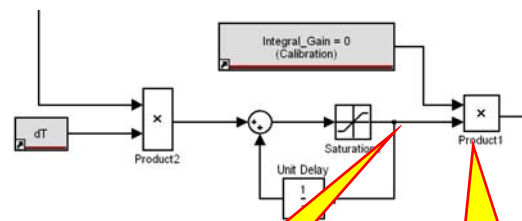
121

- A step change like the one we see in our motor-generator system could easily break a shaft.
- The reason for this problem is where we placed the gain block for our integrator.



Calibration Problem

122



Integrator output: This value hold constant or changes relatively slowly when a step change occurs in the gain. (It will eventually change in response to the step change in gain.)

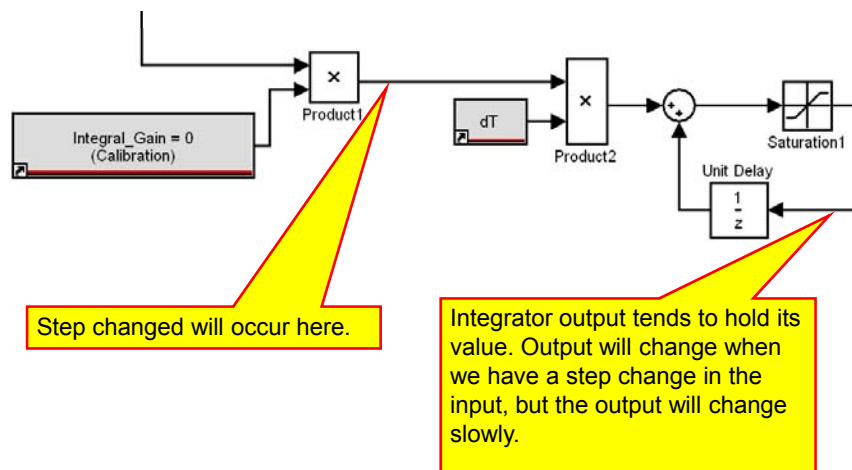
Integral gain (the calibration together with the product block): This value can change instantaneously.

Output to sum block: Even though the integrator holds it value, this signal will have step changes because the gain has step changes.

Calibration Problem

123

- We can fix this problem by placing the gain before integrator:



Calibration Problem

124

- Even though the input to the integrator will have step changes, the output of the integrator can only change slowly.
- Thus, when we change the integral gain, we will not get wild changes in the motor speed and torque that we saw earlier.
- We expect changes, but only slow changes.
- Note that the change in the placement of the gain does change the effect of the limiter. (How?)



Controller Improvements

125

- One thing we notice in the operation of our motor-generator system is that even though we are keeping the motor speed constant, as we turn on more light bulbs, the light bulbs become dimmer.
- This is because of the output impedance of the generator.
- We are effectively modeling the generator as an ideal voltage source:
 - The output voltage is constant no matter how much current we draw.
 - The output voltage is a function of the generator rpm (which we are keeping constant)



Generator Model

126

- With this model, if we keep the rpm constant, the output voltage should be constant, independent of load.
- This is obviously not the case since we observed that the lights become dimmer as we turn on more lights.
- A better model of the generator is an ideal voltage source with a series output resistance.





Generator Model

127

- What we should do:
 - Set up a series of measurements using the MathWorks Model Based Calibration tool.
 - Measure the generator characteristics.
 - Generate a new model.
 - Simulate the new model.
 - Generate a new controller for the model.
 - Download the new model on to our ECU.



Constant Voltage Controller

128

- We will skip the process of improving the model and simulating a controller.
- **Note that this is a dangerous step if you are working with a complicated and expensive system.**
- **You should always simulate a new control method before deploying it on hardware.**
- **We don't have time in this short workshop to do this.**





Constant Voltage Controller

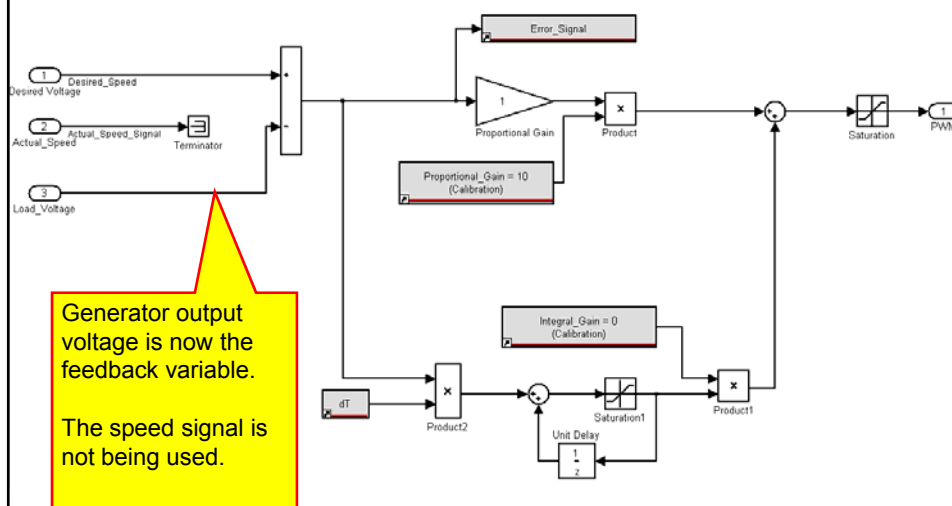
129

- Instead we will use our knowledge of control systems.
- Looking at our controller, we note that we are comparing the control signal (called Desired Voltage) to the measured rpm signal.
- The feedback tries to make the measured signal equal to the control signal.
- To regulate the generator output voltage, all we need to do is change the signal we are feeding back.
- Make the changes shown on the next slide.



Constant Voltage Controller

130





Constant Voltage Controller

131

- The feedback now attempts to keep the generator output voltage equal to the control signal.
- As the load changes, the controller will try to keep the generator voltage constant.
- When you run this model, you will notice that, in order to keep the generator voltage constant, the motor will speed up as more light bulbs are turned on.



Constant Voltage Controller

132

- Build this model.
- Download it to your ECU.
- Test it.
- Observe the effects of changing the gains.
- Note that this system may require different feedback gains than our constant motor speed system.
- You should be able to visually see the effects of over and undershoot in the brightness of the bulbs.





Lecture 17 Demo 5

- Demo of Motor/generator with the PI feedback controller and constant voltage feedback.

Demo _____



Except where otherwise noted, this work is licensed under

<http://creativecommons.org/licenses/by/3.0/>



Advanced Model-Based-System Design

Lecture 18: CAN Communication

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Outline

2

- Can code calculator
- CAN message data base with Vector CANdb+
- CAN message m-files for MotoHawk
- MotoHawk Read CAN Message block
- MotoHawk Send CAN message block
- We will create a system where we have two MotoTron ECUs communicate over a CAN link.

MotoTron

 The MathWorks

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY



CAN Signals - Apology

3

- (The author apologizes for the repetitive nature of the following 70 slides.)



ECU1 CAN Output

4

- We will generate 4 sine waves on ECU1 and send the values over CAN to ECU2.
 - Sine Wave 1: Period = 1 s, Amplitude = 1V
 - Sine Wave 2: Period = 2 s, Amplitude = 3V
 - Sine Wave 3: Period = 3 s, Amplitude = 5V
 - Sine Wave 4: Period = 4 s, Amplitude = 7V
- We will send all 4 values in one CAN message with CAN ID x123





Vector CANdb++

5

- We will use a program named CANdb+ from Vector Informatik to visualize CAN messages and maintain a CAN database.
- All signals will be saved in a database named MBD_MotoHawk.dbc
- We will show a variety of signals to illustrate CAN scaling and offsets.



CAN Message ECU1_Message1

6

- Sent by ECU1
- CAN ID x123 (hex)
- CAN Standard 11-bit ID
- DLC – 6 bytes in length
- Signals
 - SW1 → values -1 to 1
 - SW2 → values -3 to 3
 - SW3 → values -5 to 5



SW1

7

- Sine Wave 1 is a signal with values from -1 to +1.
- Send as an unsigned 8-bit code.

CAN Code Calculator

Unsigned Codes

$n := 8$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 255$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



8

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 1 \quad Y_{\min} := -1$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

Given

$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 7.843137 \times 10^{-3} \quad \text{offset} = -1$$

Name	Message	Startbit	Leng...	Byte Order	Value Type	Factor	Offset
SW1	ECU1_Message1	0	8	Intel	Unsigned	0.00784314	-1

SW2

9

- Sine Wave 2 is a signal with values from -3 to +3.
- Send as a signed 8-bit code.

CAN Code Calculator

2's compliment signed codes

$$n := 8$$

$$X_{\max} := 2^{n-1} - 1 \quad X_{\min} := -2^{n-1}$$

$$X_{\max} = 127 \quad X_{\min} = -128$$

X is the binary value of the code in the field. We are assuming a signed codes from -2^{n-1} to $2^{n-1}-1$.



10

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 3 \quad Y_{\min} := -3$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

Given

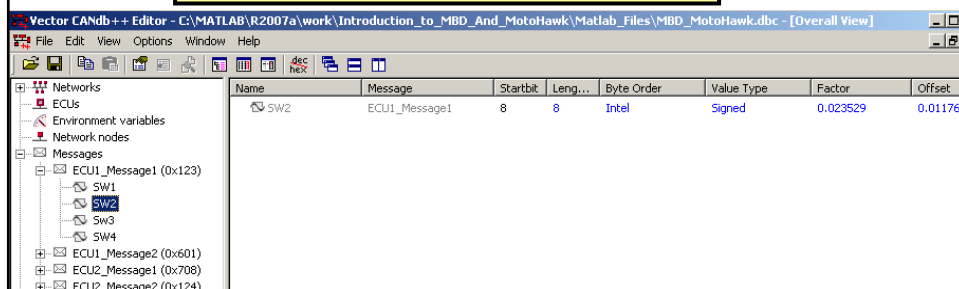
$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 0.023529$$

$$\text{offset} = 0.0117647$$





SW2

11

- Sine Wave 3 is a signal with values from -5 to +5.
- Send as an unsigned 16-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes

$n := 16$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 6.553 \times 10^4$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 5 \quad Y_{\min} := -5$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

Given

$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 1.525902 \times 10^{-4} \quad \text{offset} = -5$$

12

Vector CANdb++ Editor - C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\MBD_MotoHawk.dbc - [Overall View]

File Edit View Options Window Help

Networks
ECUs
Environment variables
Network nodes
Messages
ECU1_Message1 (0x123)
SW1
SW2
SW3
SW4
ECU1_Message2 (0x601)
ECU2_Message1 (0x708)
ECU2_Message2 (0x124)

Name	Message	Startbit	Leng...	Byte Order	Value Type	Factor	Offset
Sw3	ECU1_Message1	16	16	Intel	Unsigned	0.00015259	-5

SW4

13

- Sine Wave 4 is a signal with values from -7 to +7.
- Send as a signed 16-bit code.
- Use big endian (Motorola) format.

CAN Code Calculator 2's compliment signed codes

$n := 16$

$$X_{\max} := 2^{n-1} - 1 \quad X_{\min} := -2^{n-1}$$

$$X_{\max} = 32767 \quad X_{\min} = -32768$$

X is the binary value of the code in the field. We are assuming a signed codes from -2^{n-1} to $2^{n-1}-1$.



14

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 7 \quad Y_{\min} := -7$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

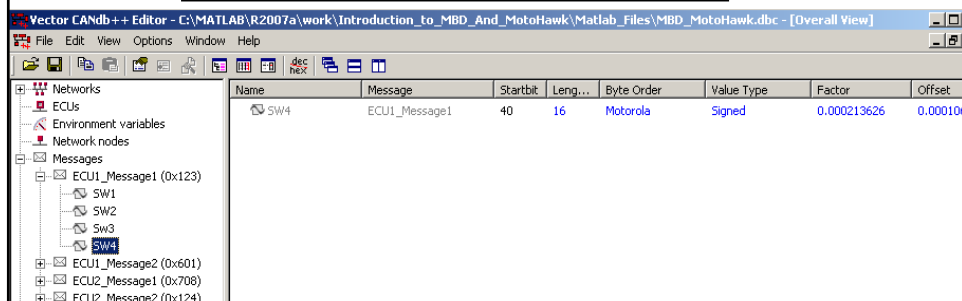
Given

$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

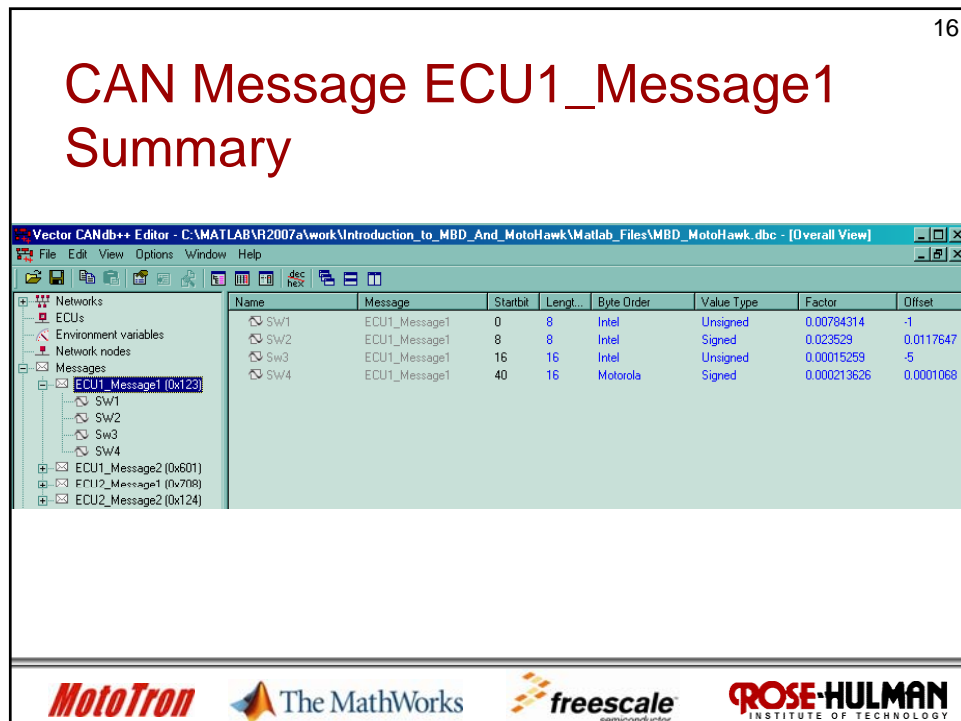
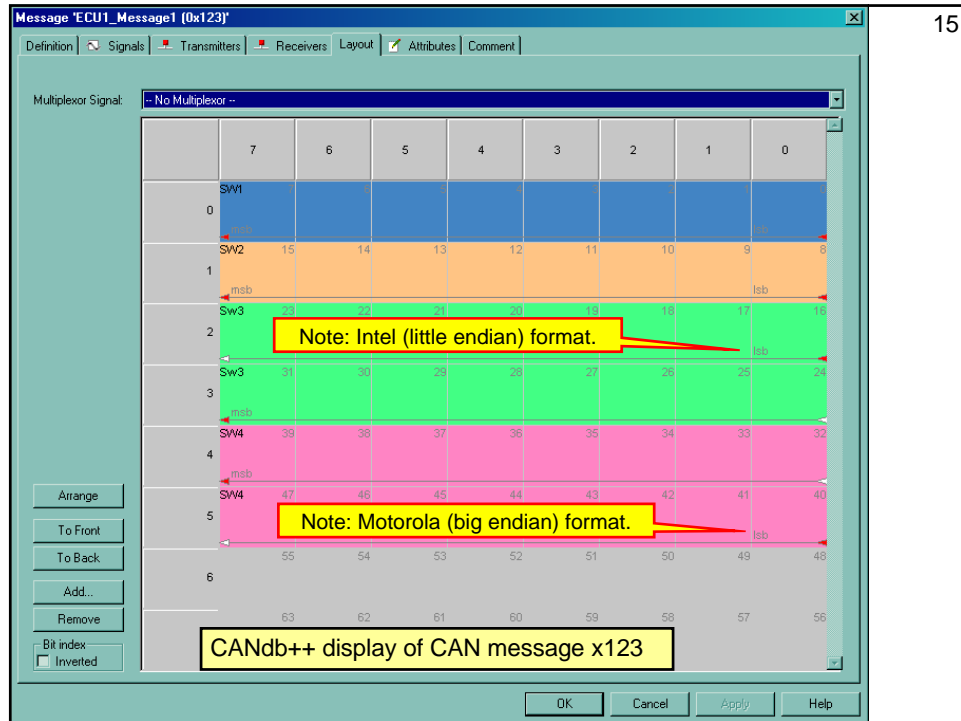
$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 2.136263 \times 10^{-4} \quad \text{offset} = 0.0001068$$



Name	Message	Startbit	Leng...	Byte Order	Value Type	Factor	Offset
SW4	ECU1_Message1	40	16	Motorola	Signed	0.000213626	0.0001068





MotoHawk CAN M-files

17

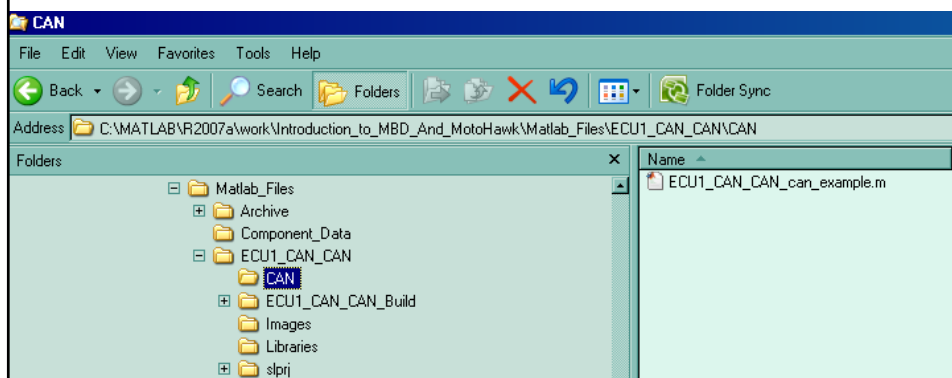
- The MotoHawk CAN block does not read CANdb+ files.
- We will create one m-file for each CAN message we want to send.
- When you create a model using the command `motohawk_project`, you may have noticed a subdirectory called CAN.
- This directory:
 - Is where we will place our CAN m-files.
 - Contains an example CAN message m-file.



Example CAN Message M-file

18

- If you look in the CAN directory, you should see an m-file with the name of your project and a .m extension.





CAN Message m-files

19

- You can open this file and take a look at the file.
- Here, we will go over the m-files we created for this example.
- These files are provided for you.
- First, take a look at file called ECU1_Message1.m.
- Edit this file with the Matlab editor:



ECU1_Message1.m

20

```

function msg = ECU1_Message1()

%%
%% In MotoHawk (version 0.8.2 and later), the format of a CAN message is described using a MATLAB
%% struct (structure array) containing required and optional fields. Optional field .idcontent()
%% is a MATLAB cell array of structs used to describe/access bit fields within the message ID (such
%% as a Node ID in this example, or the Priority and Source Address fields of an SAE J1939 message).
%% Optional field .payloads() is a MATLAB cell array of structs used to describe/access fields within
%% the message payload.
%%
%% *** message name and description ***
%% .name           : Name displayed on block                               (default: empty string)
%% .description     : Brief text used to document the message              (default: empty string)
%% .protocol        : Name of the protocol used                           (default: empty string)
%% .module          : Name of the source module                           (default: empty string)
%% .channel         : Name of the source CAN channel                       (default: 1)
%%
%% *** CAN ID ***
%% .id              : CAN ID (29-bit)                                     (if undefined, uses .idinherit = 1)
%% .id_extended     : Extended CAN ID (29-bit)                           (if undefined, uses .idinherit = 1)
%% .id_for_receive  : ID for a receive slot (default: 0xffffffff)
  
```

Note that our m-file was named ECU1_Message1.m.





ECU1_message1

21

```
Broadcast = 20; %The broadcast rate in ms

msg.name           = 'ECU1_Message1';
msg.description    = 'Example Message for Model-Based Design Intro Course';
msg.protocol       = 'CAN';
msg.module         = 'ECU1';
msg.channel        = 1; %CAN bus being used

msg.idext          = 'STANDARD';
msg.id             = hex2dec('123');
msg.idmask         = hex2dec('7ff'); %don't worry about masking now
msg.idinherit      = 0; %don't want to inherit
msg.payload_size   = 6; %number of bytes in the payload
msg.payload_value  = []; %don't do payload filtering, just a huge pain
msg.payload_mask   = [];
msg.interval       = Broadcast; %defined above
```

The message will be sent at a 50 Hz rate.

Information for documentation.

The physical CAN channel we will be using.



ECU1_message1

22

```
Broadcast = 20; %The broadcast rate in ms

msg.name           = 'ECU1_Message1';
msg.description    = 'Example Message for Model-Based Design Intro Course';
msg.protocol       = 'CAN';
msg.module         = 'ECU1';
msg.channel        = 1; %CAN bus being used

msg.idext          = 'STANDARD';
msg.id             = hex2dec('123');
msg.idmask         = hex2dec('7ff'); %don't worry about masking now
msg.idinherit      = 0; %don't want to inherit
msg.payload_size   = 6; %number of bytes in the payload
msg.payload_value  = []; %don't do payload filtering, just a huge pain
msg.payload_mask   = [];
msg.interval       = Broadcast; %defined above
```

We will be using an 11-bit ID.

The ID for this CAN message is Hex 123.

This is a 6-byte message.





ECU_Message1

23

```
%=====
% Sine Waves created on ECU1 sent to ECU2
%=====
i = 1;
%Sine Wave Signal SW1
msg.fields(i).name = 'SW1';
msg.fields(i).units = '';
msg.fields(i).start_bit = 56;
msg.fields(i).bit_length = 8;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.007843137;
msg.fields(i).offset = -1;
i = i+1;

%Sine Wave Signal SW2
msg.fields(i).name = 'SW2';
msg.fields(i).units = '';
msg.fields(i).start_bit = 40;
msg.fields(i).bit_length = 8;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'SIGNED';
msg.fields(i).scale = 0.023529;
msg.fields(i).offset = 0.0117647;
i = i+1;

%Sine Wave Signal SW3
msg.fields(i).name = 'SW3';
msg.fields(i).units = '';
msg.fields(i).start_bit = 40;
msg.fields(i).bit_length = 16;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.0001528902;
msg.fields(i).offset = -5;
i = i+1;

%Sine Wave Signal SW4
msg.fields(i).name = 'SW4';
msg.fields(i).units = '';
msg.fields(i).start_bit = 16;
msg.fields(i).bit_length = 16;
msg.fields(i).byte_order = 'BIG_ENDIAN';
msg.fields(i).data_type = 'SIGNED';
msg.fields(i).scale = 0.0002132643;
msg.fields(i).offset = 0.0001068;
```

Here is where the signals
contained in the message
are defined.

ECU1_message1 – SW1

24

```
%=====
% Sine Waves created on ECU1 sent to ECU2
%=====
i = 1;
%Sine Wave Signal SW1
msg.fields(i).name = 'SW1';
msg.fields(i).units = '';
msg.fields(i).start_bit = 56;
msg.fields(i).bit_length = 8;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.007843137;
msg.fields(i).offset = -1;
i = i+1;
```

Signal name. Same as in CANdb+.

Units for documentation only.

Same as in CANdb+.



ECU1_message1 – SW1

25

```

=====
% Sine Waves created on ECU1 sent to ECU2
=====
i = 1;
%Sine Wave Signal SW1
msg.fields(i).name = 'SW1';
msg.fields(i).units = '';
msg.fields(i).start_bit = 56;
msg.fields(i).bit_length = 8;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.007843137;
msg.fields(i).offset = -1;
i = i+1;

```

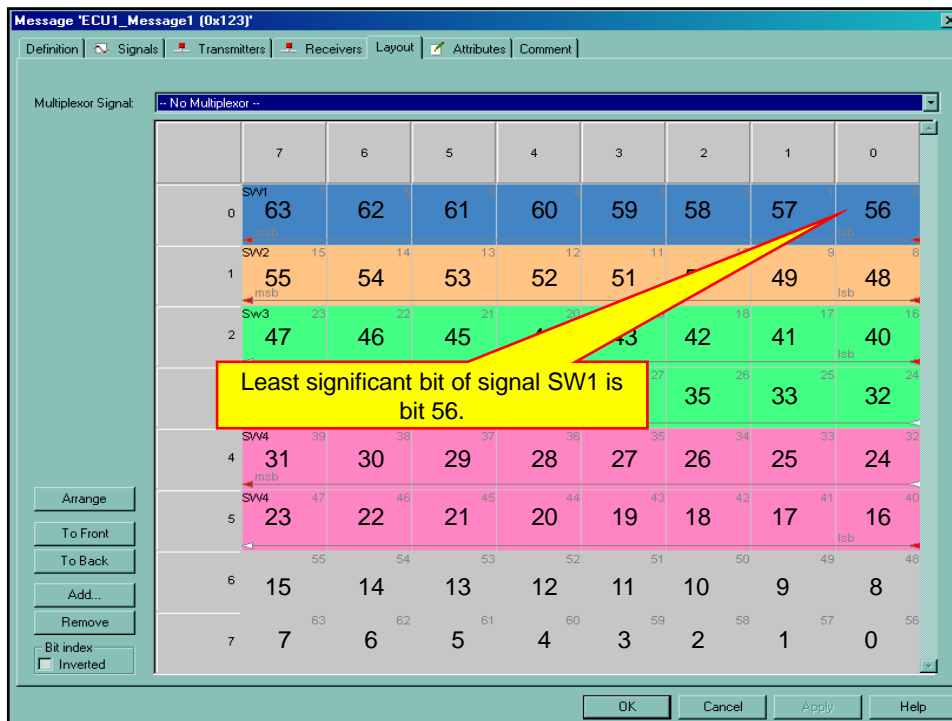
Location of the least significant bit of the signal.

Independent of:

- Big endian, little endian.

- Number of bytes in the message.

Bit numbering is different than that shown in CANdb+.

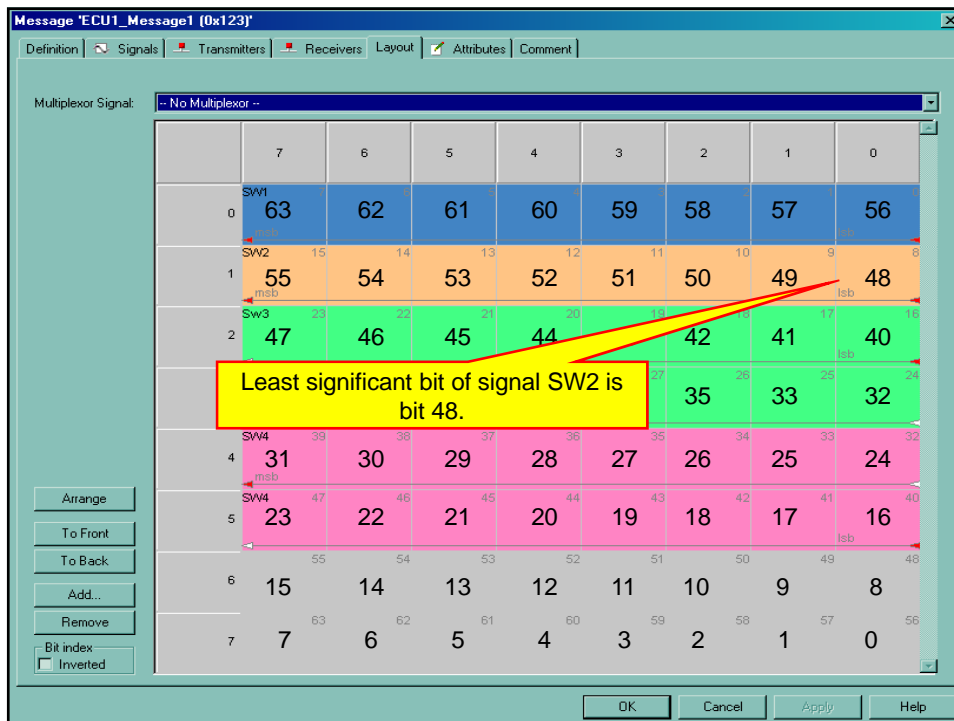


ECU1_message1 – SW2

27

```
%Sine Wave Signal SW2
msg.fields(i).name = 'SW2';
msg.fields(i).units = '';
msg.fields(i).start_bit = 48;
msg.fields(i).bit_length = 8;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'SIGNED';
msg.fields(i).scale = 0.023529;
msg.fields(i).offset = 0.0117647;
i = i+1;
```

Location of the least significant bit of this signal is 48.



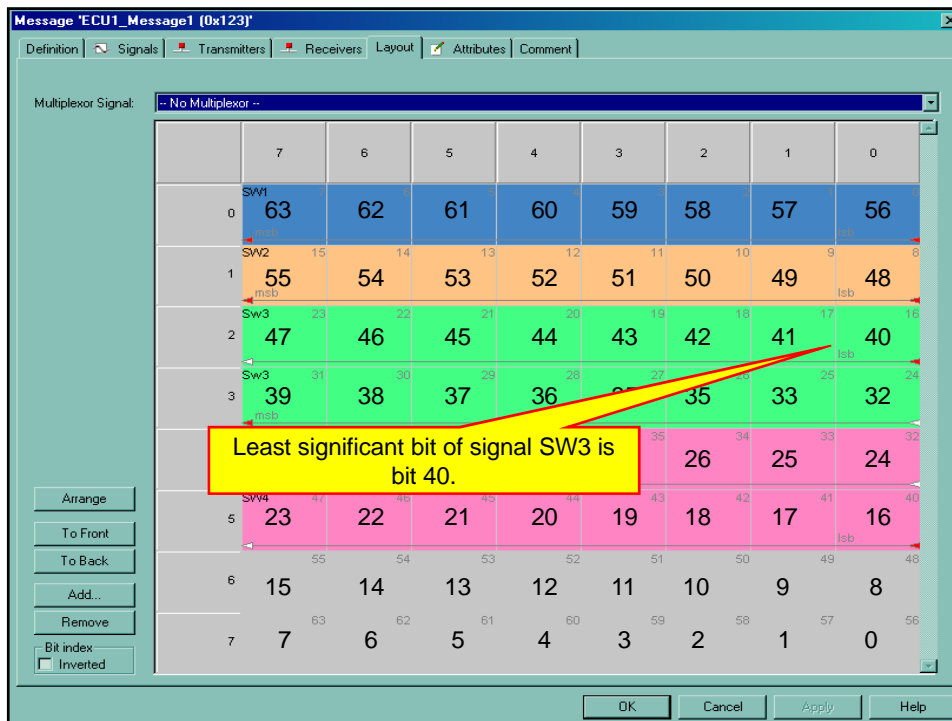


ECU1_message1 – SW3

29

```
%Sine Wave Signal SW3
msg.fields(i).name = 'SW3';
msg.fields(i).units = '';
msg.fields(i).start_bit = 40;
msg.fields(i).bit_length = 16;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.0001525902;
msg.fields(i).offset = -5;
i = i+1;
```

Location of the least significant bit of this signal is 40.

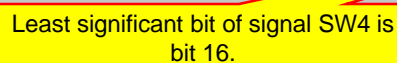


31

```

57 %Sine Wave Signal SW4
58 - msg.fields(i).name = 'SW4';
59 - msg.fields(i).units = '';
60 - msg.fields(i).start_bit = 16;
61 - msg.fields(i).bit_length = 16;
62 - msg.fields(i).byte_order = 'BIG_ENDIAN';
63 - msg.fields(i).data_type = 'SIGNED';
64 - msg.fields(i).scale = 0.0002136263;
65 - msg.fields(i).offset = 0.0001068;
66

```

MotoTron**freescale**



33

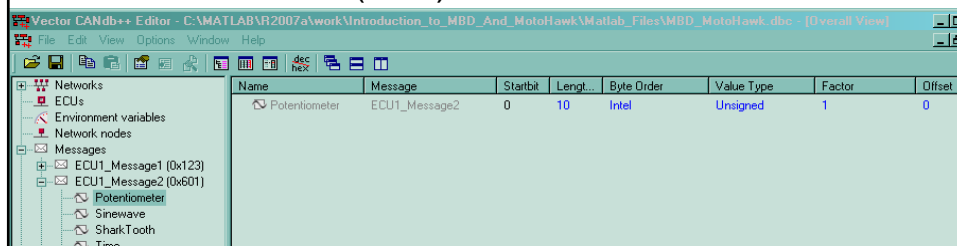
ECU1_Message2

- Sent by ECU1
- CAN ID x601 (hex)
- CAN Standard 11-bit ID
- DLC – 6 bytes in length
- Signals
 - Potentiometer Value → 0 to 1023.
 - Sine Wave → Period = 5.0 s, Amplitude = 5V
 - Shark Tooth Waveform (Ramp) → -100 to 100.

34

Potentiometer

- Analog signal measured with analog input. Use the POT on the Motor-Generator system.
- Value is read on ECU1 as a 10 bit code with values from 0 to $2^{10}-1$ (1023 for non ECE people).
- Send over CAN as an unsigned 10-bit code.
- Factor is 1.
- Use little endian (Intel) format.



Sinewave (not misspelled!)

35

- Signal with values from -5 to +5.
- Send as an unsigned 14-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes

$n := 14$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 1.638 \times 10^4$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to 2^n-1 .



36

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 5 \quad Y_{\min} := -5$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

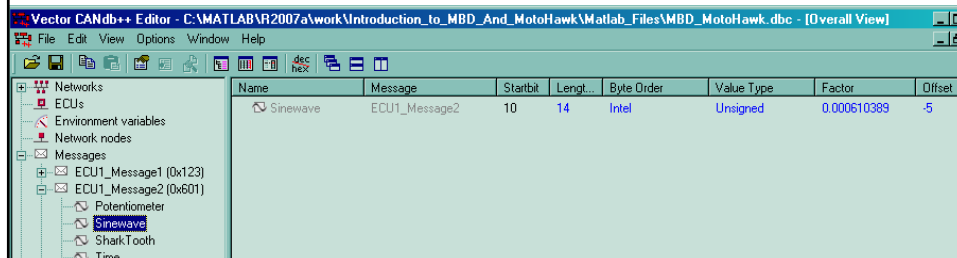
Given

$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset} \quad +$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 6.1038881768 \times 10^{-4} \quad \text{offset} = -5$$



SharkTooth

37

- Signal with values from -100 to +100.
- Send as an unsigned 12-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes

$n := 12$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 4.095 \times 10^3$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



Y_{\max} and Y_{\min} are the values of the data signal being sent via CAN.

$$Y_{\max} := 100 \quad Y_{\min} := -100$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

Given

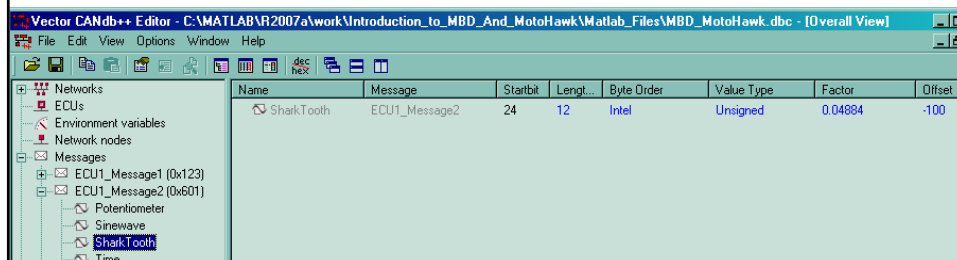
$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 0.0488400488 \quad \text{offset} = -100$$

38



Time

39

- Signal with values from 0 to 3600.
- Send as an unsigned 12-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes

$n := 12$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 4.095 \times 10^3$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



40

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 3600 \quad Y_{\min} := 0$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

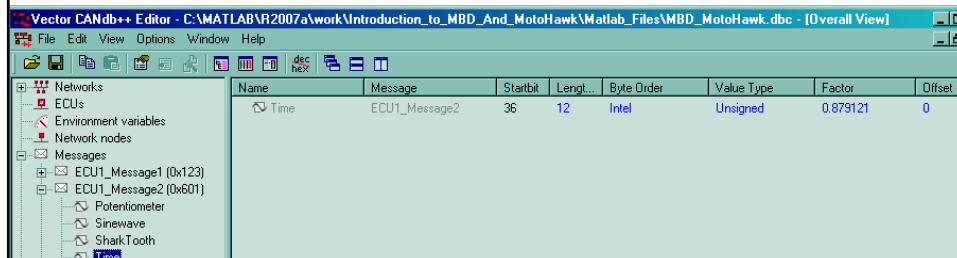
Given

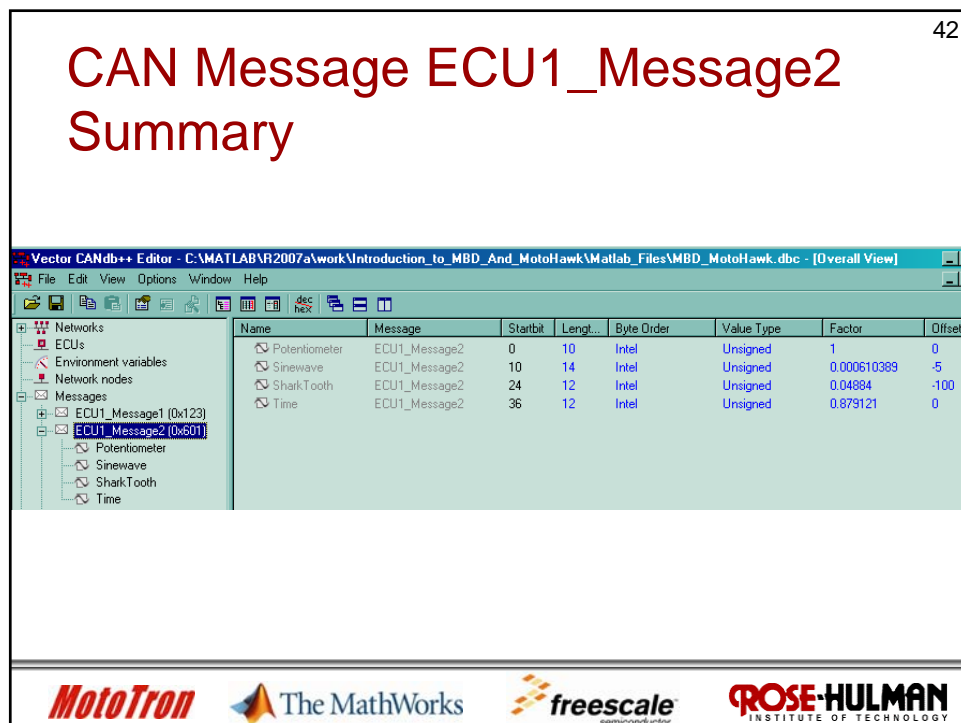
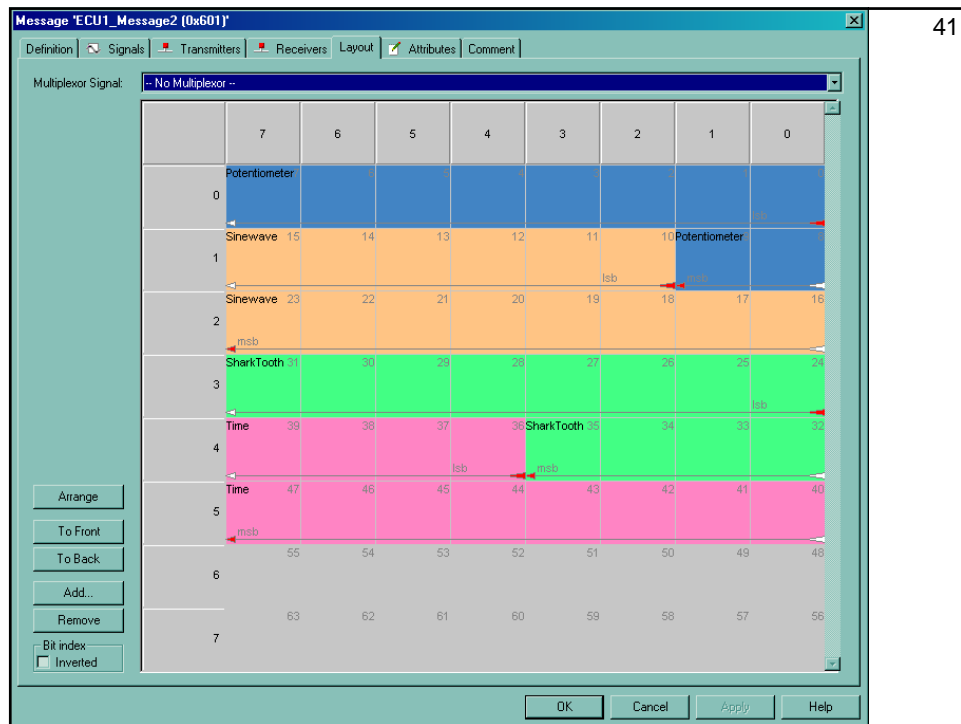
$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 0.8791208791 \quad \text{offset} = 0$$







ECU1_Message2.m

43

Editor - C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\ECU1_CAN_CAN\ECU1_Message2.m

File Edit Text Go Cell Tools Debug Desktop Window Help

Stack Base

```

1 function msg = ECU1_Message2()
2
3
4 %%
5 %% In MotoHawk (version 0.8.2 and later), the format of a CAN message is described using a MATLAB
6 %% struct (structure array) containing required and optional fields. Optional field .idcontent()
7 %% is a MATLAB cell array of structs used to describe/access bit fields within the message ID (such
8 %% as a Node ID in the example, or the Priority and Source Address fields of an SAE J1939 message).
9 %% Optional field .ids() is a MATLAB cell array of structs used to describe/access fields within
10 %% the message payload.
11
12 %% *** message name and description ***
13 %% .name           : name displayed on block (default: empty string)
14 %% .description     : text used to document the message (default: empty string)
15 %% .protocol        : of the protocol used (default: empty string)
16 %% .module          : of the source module (default: empty string)
17 %% .channel         : number of the source CAN channel (default: 1)
18
19 %% *** CAN ID ***
20 %% .id              : CAN ID (29-bit) (if undefined, uses .idinherit = 1)

```

Note that our m-file was named ECU1_Message2.m.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

ECU1_message2

44

```

Broadcast = 20;

msg.name           = 'ECU1_Message2';
msg.description    = 'Example Message for Model-Based Design Intro Course';
msg.protocol       = 'C\$';
msg.module         = 'ECU1';
msg.channel        = 1; %CAN bus being used

msg.idext          = 'STANDARD';
msg.id             = hex2dec('601');
msg.idmask         = hex2dec('7ff'); %don't worry about masking now
msg.idinherit      = 0; %don't want to inherit
msg.payload_size   = 6; %number of bytes in the payload
msg.payload_value   = []; %don't do payload filtering, just a huge pain
msg.payload_mask    = [];
msg.interval       = Broadcast; %defined above

```

The message will be sent at a 50 Hz rate.

Information for documentation.

The physical CAN channel we will be using.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY



ECU1_message2

45

```
Broadcast = 20; %The broadcast rate in ms

msg.name           = 'ECU1_Message2';
msg.description    = 'Example Message for Motor Vehicle Design Intro Course';
msg.protocol       = 'C&';
msg.module         = 'ECU1';
msg.channel        = 1; %CAN bus being used

msg.idext          = 'STANDARD';
msg.id             = hex2dec('601');
msg.idmask         = hex2dec('7ff'); %don't worry about masking now
msg.idinherit      = 0; %don't want to inherit
msg.payload_size   = 6; %number of bytes in the payload
msg.payload_value   = []; %don't do payload filtering, just a huge pain
msg.payload_mask    = [];
msg.interval       = Broadcast; %defined above
```

We will be using an 11-bit ID.

The ID for this CAN message is Hex 601.

This is a 6-byte message.



ECU_Message2

46

```
%=====
% Signals created on ECU1 sent to ECU2
%=====
i = 1;

%Potentiometer Signal
msg.fields(i).name = 'Potentiometer';
msg.fields(i).units = '';
msg.fields(i).start_bit = 56;
msg.fields(i).bit_length = 10;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 1;
msg.fields(i).offset = 0;
i = i+1;

%Sinewave Signal
msg.fields(i).name = 'Sinewave';
msg.fields(i).units = '';
msg.fields(i).start_bit = 50;
msg.fields(i).bit_length = 14;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.0006103880;
msg.fields(i).offset = -5;
i = i+1;

%SharkTooth Signal
msg.fields(i).name = 'SharkTooth';
msg.fields(i).units = '';
msg.fields(i).start_bit = 40;
msg.fields(i).bit_length = 12;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.0488400488;
msg.fields(i).offset = -100;
i = i+1;

%Time Signal
msg.fields(i).name = 'Time';
msg.fields(i).units = 's';
msg.fields(i).start_bit = 20;
msg.fields(i).bit_length = 12;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.0791200791;
msg.fields(i).offset = 0;
```

Here is where the signals contained in the message are defined.

ECU1_message2 – Sinewave

47

```
%Sinewave Signal
msg.fields(i).name = 'Sinewave';
msg.fields(i).units = '';
msg.fields(i).start_bit = 50;
msg.fields(i).bit_length = 14;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.0006103888;
msg.fields(i).offset = -5;
i = i+1;
```

Location of the least significant bit of signal Sinewave is 50.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Message 'ECU1_Message2 (0x601)'

Definition | Signals | Transmitters | Receivers | Layout | Attributes | Comment

Multiplexor Signal: -- No Multiplexor --

	7	6	5	4	3	2	1	0
Potentiometer	63	62	61	60	59	58	57	56
Sinewave	55	54	53	52	51	50	49	48
Potentiometer	47	46	45	44	43	42	41	40
SharkTooth	39	38	37	36	35	34	33	32
Time	31	30	29	28	27	26	25	24
Time	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8
	7	6	5	4	3	2	1	0

Least significant bit of signal Sinewave is bit 50.

Arrange
To Front
To Back
Add...
Remove
Bit index:
☐ Inverted

OK Cancel Apply Help

48



ECU1_message2 – Time

49

```
%Time Signal
msg.fields(i).name = 'Time';
msg.fields(i).units = '';
msg.fields(i).start_bit = 28;
msg.fields(i).bit_length = 12;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.8791208791;
msg.fields(i).offset = 0;
```

Location of the least significant bit of the signal.



Message 'ECU1_Message2 (0x601)'

Definition | Signals | Transmitters | Receivers | Layout | Attributes | Comment

Multiplexor Signal: -- No Multiplexor --

	7	6	5	4	3	2	1	0
Potentiometer	63	62	61	60	59	58	57	56
Sinewave	55	54	53	52	51	50	49	48
SharkTooth	39	38	37	36	35	34	33	32
Time	31	30	29	28	27	26	25	24
Time	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8
	7	6	5	4	3	2	1	0

Least significant bit of signal Sinewave is bit 28.

Arrange
To Front
To Back
Add...
Remove
Bit index
☐ Inverted

OK Cancel Apply Help

50

51

CAN Message ECU2_Message1

- Sent by ECU2
- CAN ID x708 (hex)
- CAN Standard 11-bit ID
- DLC – 3 bytes in length
- Signals
 - Temperature → 0 to 100
 - Fred → -3150 to -3120
 - LED3 → 0 to 1

52

Temperature

- Signal with values from 0 to 100.
- Send as an unsigned 7-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes +

$n := 7$ Number of bits in the code

$X_{\max} := 2^n - 1$ $X_{\max} = 127$

$X_{\min} := 0$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



53

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 100 \quad Y_{\min} := 0$$

factor := 1 offset := 0 Initial Guesses

Given

$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 0.7874015748 \quad \text{offset} = 0$$

54

Fred

- Signal values from -3150 to -3120.
- Send as an unsigned 5-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes

$n := 5$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 31$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



55

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := -3120 \quad Y_{\min} := -3150$$

factor := 1 offset := 0 Initial Guesses

Given

$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

$$\text{factor} = 0.9677419355 \quad \text{offset} = -3150$$

Name	Message	Startbit	Length	Byte Order	Value Type	Factor	Offset
Fred	ECU2_Message1	7	5	Intel	Unsigned	1	-3150

56

LED3

- Signal value 0 or 1.
- Send over CAN as an unsigned 1-bit code.
- Factor is 1.
- Use little endian (Intel) format.

Name	Message	Startbit	Length	Byte Order	Value Type	Factor	Offset
LED3	ECU2_Message1	12	1	Intel	Unsigned	1	0

Pulsewidth

57

- Signal values from 0 to 100.
- Send as an unsigned 10-bit code.
- Use little endian (Intel) format.

CAN Code Calculator

Unsigned Codes

$n := 10$ Number of bits in the code

$$X_{\max} := 2^n - 1 \quad X_{\max} = 1.023 \times 10^3$$

$$X_{\min} := 0$$

X is the binary value of the code in the field. We are assuming a unsigned codes from 0 to $2^n - 1$.



58

Ymax and Ymin are the values of the data signal being sent via CAN.

$$Y_{\max} := 100 \quad Y_{\min} := 0$$

$$\text{factor} := 1 \quad \text{offset} := 0 \quad \text{Initial Guesses}$$

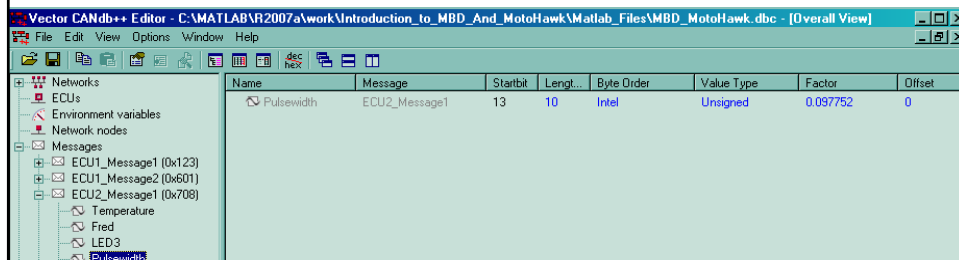
Given

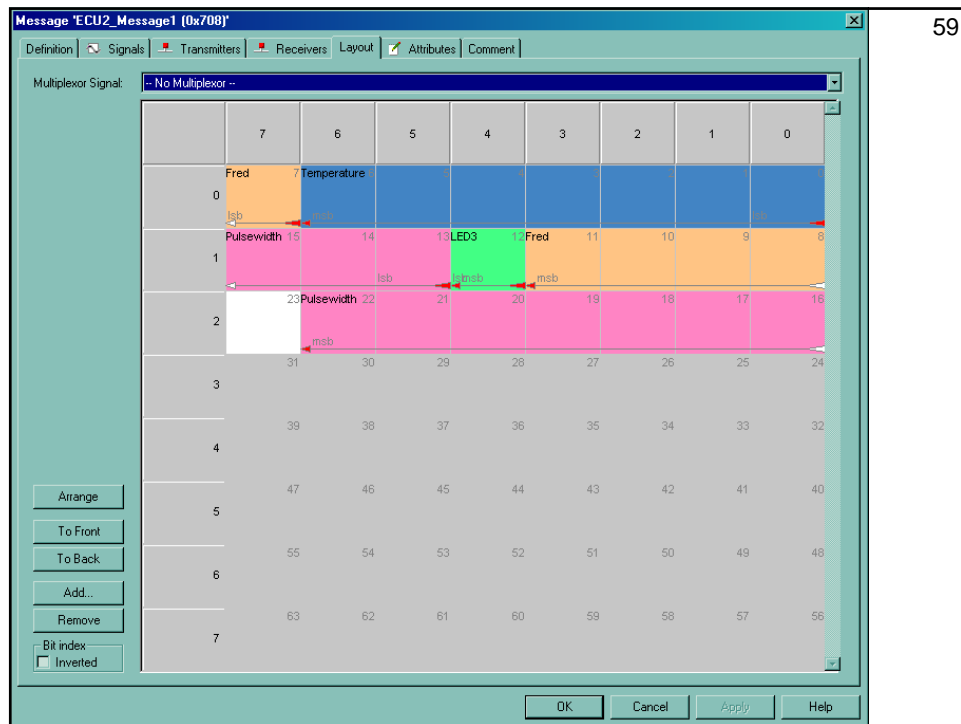
$$Y_{\max} = \text{factor} \cdot X_{\max} + \text{offset}$$

$$Y_{\min} = \text{factor} \cdot X_{\min} + \text{offset}$$

$$\begin{pmatrix} \text{factor} \\ \text{offset} \end{pmatrix} := \text{Find}(\text{factor}, \text{offset})$$

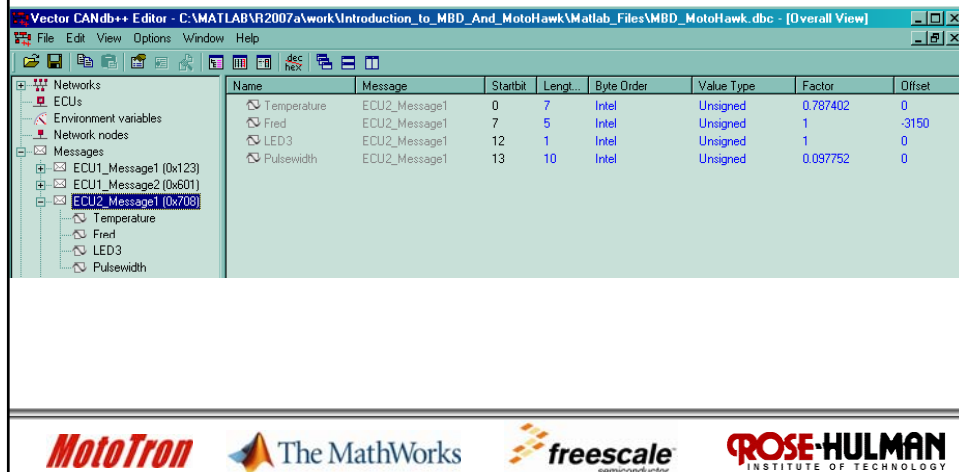
$$\text{factor} = 0.0977517107 \quad \text{offset} = 0$$









CAN Message ECU2_Message1 Summary

60



Name	Message	Startbit	Length	Byte Order	Value Type	Factor	Offset
Temperature	ECU2_Message1	0	7	Intel	Unsigned	0.787402	0
Fred	ECU2_Message1	7	5	Intel	Unsigned	1	-3150
LED3	ECU2_Message1	12	1	Intel	Unsigned	1	0
Pulsewidth	ECU2_Message1	13	10	Intel	Unsigned	0.097752	0



ECU2_Message1.m

61

Editor - C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\ECU1_CAN_CAN\ECU2_Message1.m

File Edit Text Go Cell Tools Debug Desktop Window Help

```

1 function msg = ECU2_Message1()
2
3
4 %%
5 %% In MotoHawk (versions 0.8.2 and later), the format of a CAN message is described using a MATLAB
6 %% struct (structure array) containing required and optional fields. Optional field .idcontent()
7 %% is a MATLAB cell array of structs used to describe/access bit fields within the message ID (such
8 %% as a Node ID in this example, or the Priority and Source Address fields of an SAE J1939 message).
9 %% Optional field .fields() is a MATLAB cell array of structs used to describe/access fields within
10 %% the message.
11 %%
12 %% *** message description ***
13 %% .name (default: empty string)
14 %% .description (default: empty string)
15 %% .protocol (default: empty string)
16 %% .channel (default: empty string)
17 %%

```

Note that our m-file was named ECU2_Message1.m.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

ECU2_Message1

62

```

Broadcast = 20; %The broadcast rate in ms

msg.name = 'ECU2_Message1';
msg.description = 'Example Message for Model-Based Design Intro Course';
msg.protocol = 'C$';
msg.module = 'ECU2';
msg.channel = 1; %CAN bus being used

msg.idext = 'STANDARD';
msg.id = hex2dec('708');
msg.idmask = hex2dec('7ff'); %don't worry about masking now
msg.idinherit = 0; %don't want to inherit
msg.payload_size = 3; %number of bytes in the payload
msg.payload_value = []; %don't do payload filtering, just a huge pain
msg.payload_mask = [];
msg.interval = Broadcast; %defined above

```

The message will be sent at a 50 Hz rate.

Information for documentation.

The physical CAN channel we will be using.





ECU2_Message1

63

Broadcast = 20; %The broadcast rate in ms

```
msg.name           = 'ECU2_Message1';
msg.description    = 'Example Message for Model-Based Design Intro Course';
msg.protocol       = 'C$';
msg.module         = 'ECU2';
msg.channel        = 1; %CAN bus being used

msg.idext          = 'STANDARD';
msg.id             = hex2dec('708');
msg.idmask         = hex2dec('7ff'); %don't worry about masking now
msg.idinherit      = 0; %don't want to inherit
msg.payload_size   = 3; %number of bytes in the payload
msg.payload_value  = []; %don't do payload filtering, just a huge pain
msg.payload_mask   = [];
msg.interval       = Broadcast; %defined above
```

We will be using an 11-bit ID.

The ID for this CAN message is Hex 708.

This is a 3-byte message.



ECU2_Message1

64

```
%-----
% Misc Signals on ECU2 sent to ECU1
%-----
i = 1;

%Temperature Signal
msg.fields(i).name = 'Temperature';
msg.fields(i).units = '';
msg.fields(i).start_bit = 56;
msg.fields(i).bit_length = 7;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.78125;
msg.fields(i).offset = 0;
i = i+1;

%Feed Signal
msg.fields(i).name = 'Feed';
msg.fields(i).units = '';
msg.fields(i).start_bit = 63;
msg.fields(i).bit_length = 5;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 1;
msg.fields(i).offset = -3150;
i = i+1;

%LED3 Signal
msg.fields(i).name = 'LED3';
msg.fields(i).units = '';
msg.fields(i).start_bit = 20;
msg.fields(i).bit_length = 1;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 1;
msg.fields(i).offset = 0;
i = i+1;

%Pulsewidth Signal
msg.fields(i).name = 'Pulsewidth';
msg.fields(i).units = '';
msg.fields(i).start_bit = 53;
msg.fields(i).bit_length = 10;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 0.097752;
msg.fields(i).offset = 0;
```

Here is where the signals contained in the message are defined.



ECU2_Message1 – Fred

65

```
%Fred Signal
msg.fields(i).name = 'Fred';
msg.fields(i).units = '';
msg.fields(i).start_bit = 63;
msg.fields(i).bit_length = 5;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 1;
msg.fields(i).offset = -3150;
i = i+1;
```

Location of the least significant bit of signal Fred is 63.



Message 'ECU2_Message1 (0x708)'

Definition | Signals | Transmitters | Receivers | Layout | Attributes | Comment

Multiplexor Signal: -- No Multiplexor --

	7	6	5	4	3	2	1	0
0	Fred 63	Temperature 62	61	60	59	58	57	56
1	Pulsewidth 55	54	53	LED3 52	Fred 51	50	49	48
2	47	46	45	44	43	42	41	40
3								
4	31	30	29	28	27	26	25	24
5	23	22	21	20	19	18	17	16
6	15	14	13	12	11	10	9	8
7	7	6	5	4	3	2	1	0

Least significant bit of signal Fred is bit 63.

Arrange
To Front
To Back
Add...
Remove
Bit index:
☐ Inverted

OK Cancel Apply Help

66



ECU2_Message1 – LED3

67

%LED3 Signal

```
msg.fields(i).name = 'LED3';
msg.fields(i).units = '';
msg.fields(i).start_bit = 52;
msg.fields(i).bit_length = 1;
msg.fields(i).byte_order = 'LITTLE_ENDIAN';
msg.fields(i).data_type = 'UNSIGNED';
msg.fields(i).scale = 1;
msg.fields(i).offset = 0;
i = i+1;
```

Location of the least significant bit of signal LED3 is 52.



Message 'ECU2_Message1 (0x708)'

Definition | Signals | Transmitters | Receivers | Layout | Attributes | Comment

Multiplexor Signal: -- No Multiplexor --

	7	6	5	4	3	2	1	0
0	63	62	61	60	59	58	57	56
1	55	54	53	52	51	50	49	48
2	47	46	45	44	43	42	41	40
3	39	38	37	36	35	34	33	32
4	31	30	29	28	27	26	25	24
5	23	22	21	20	19	18	17	16
6	15	14	13	12	11	10	9	8
7	7	6	5	4	3	2	1	0

Least significant bit of signal LED3 is bit 52.

Arrange
To Front
To Back
Add...
Remove
Bit index:
☐ Inverted

OK Cancel Apply Help

68



69

CAN Message ECU2_Message2

- Sent by ECU2
- CAN ID x124 (hex)
- CAN Standard 11-bit ID
- DLC – 1 byte in length
- Signals
 - Cooling_Fan → 0 to 1



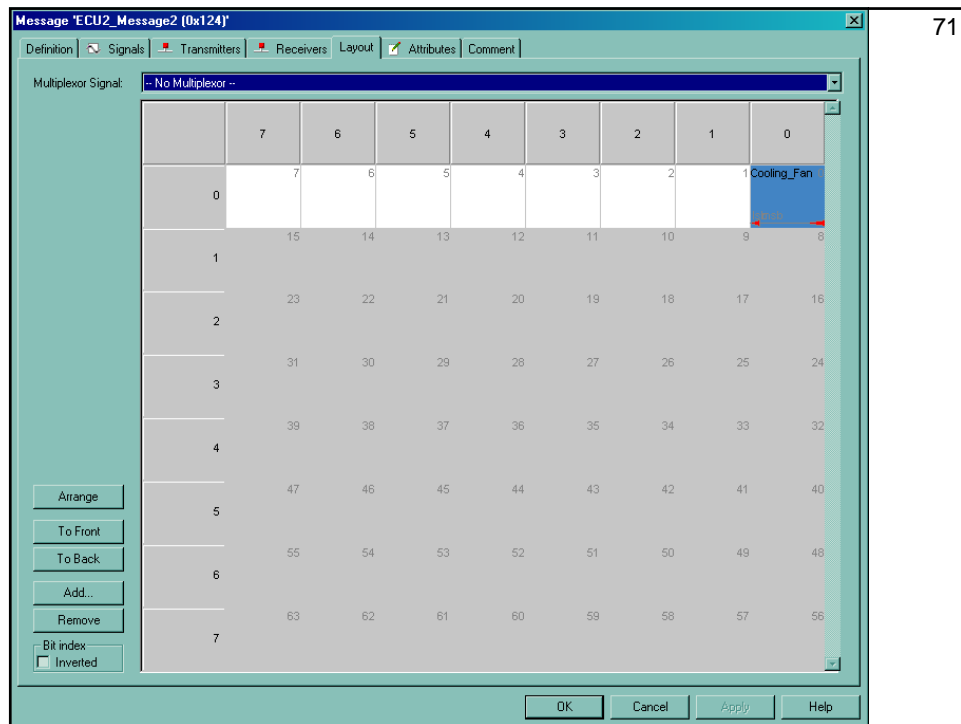
70

Cooling_Fan

- Signal value 0 or 1.
- Send over CAN as an unsigned 1-bit code.
- Factor is 1.
- Use little endian (Intel) format.

Name	Message	Startbit	Length	Byte Order	Value Type	Factor	Offset
Cooling_Fan	ECU2_Message2	0	1	Intel	Unsigned	1	0





71

CAN Project Preview

72

- We will connect two MotoTron ECUs together on a single CAN bus.
- Each ECU will control a motor-generator setup.
- ECU1 will send commands to ECU2, some of which are commands to spin the motor connected to ECU2.
- ECU2 will send commands to ECU1, some of which are commands to spin the motor connected to ECU1.





CAN Project Preview

73

- We will use two PCs to communicate with the individual ECUs.
- Each PC will use its own copy of MotoTune.
- To use MotoTron with two ECUs on the same CAN bus, we will have to change the City ID of one of the ECUs.



Can Project Preview

74

- We will use the same hardware setup as we used in the previous examples.
- No wiring changes are needed.
- All we need to do is connect two of the CAN hubs together with one of the yellow MotoTron CAN cables (non-terminated at both ends). **(Do not do this yet!)**
- We can remove one of the key switches since the switches are effectively in parallel. **(Do not do this yet!)**





Work in Groups of Two

75

- One person should choose to be ECU1.
Follow the slides with the header “ECU1.”
- These are slides that follow this slide.
- One person should choose to be ECU2.
Follow the slides with the header “ECU2.”
- ECU2 should skip the following slides and
jump to the appropriate slides.
(Approximately slide number??)



ECU1 Slides

76

ECU2 Group skip ahead to slides
labeled with header ECU2.
(Approximately slide number 116)





ECU1

77

- ECU1 will do the following:
 - Generate 4 sine waves and send the values over the CAN bus using message ECU1_Message1.
 - Read the potentiometer from the motor-generator system and send the binary value (0 to 1023) over the CAN bus using ECU1_Message2.
 - Generate signals for Time, a ramp (shark tooth), and a sine wave, and send the values over the CAN bus using message ECU1_Message2.

ECU1

78

- ECU1 will receive the following information over the CAN bus in message ECU2_Message1:
 - Temperature: Value will be displayed with a MotoHawk probe.
 - Fred and LED signals: Light up LED on motor-generator system. Display value with MotoHawk probe.
 - Receive the pulse width signal (0 to 100) and spin the motor with the given pulse width.





ECU1

79

- ECU1 will receive the following information over the CAN bus in message ECU2_Message1:
 - Cooling Fan signal. Display the value with a MotoHawk probe.

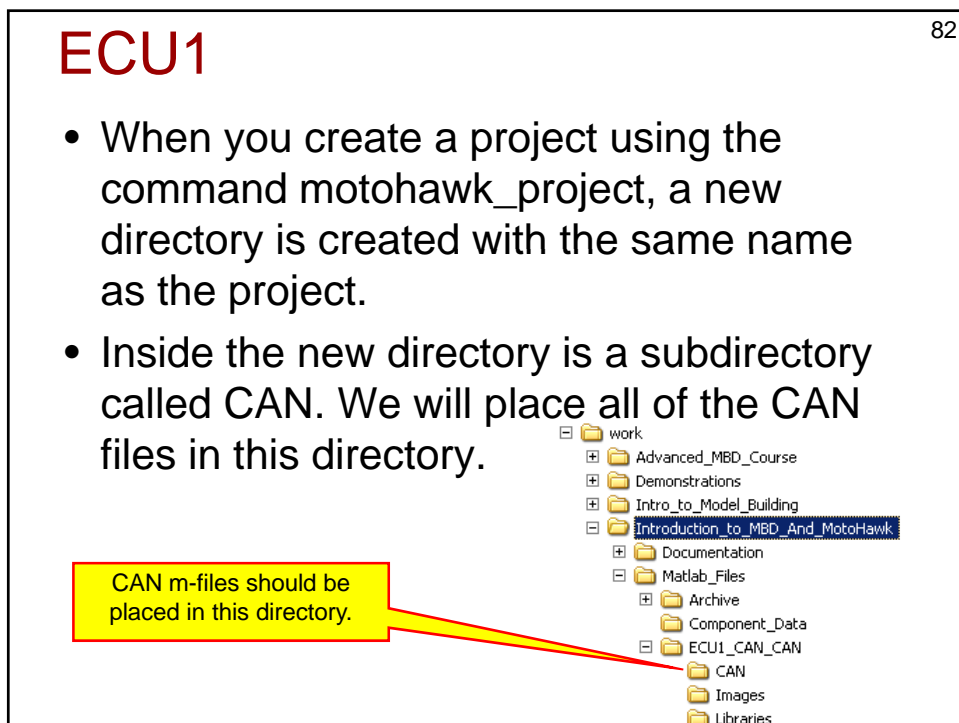
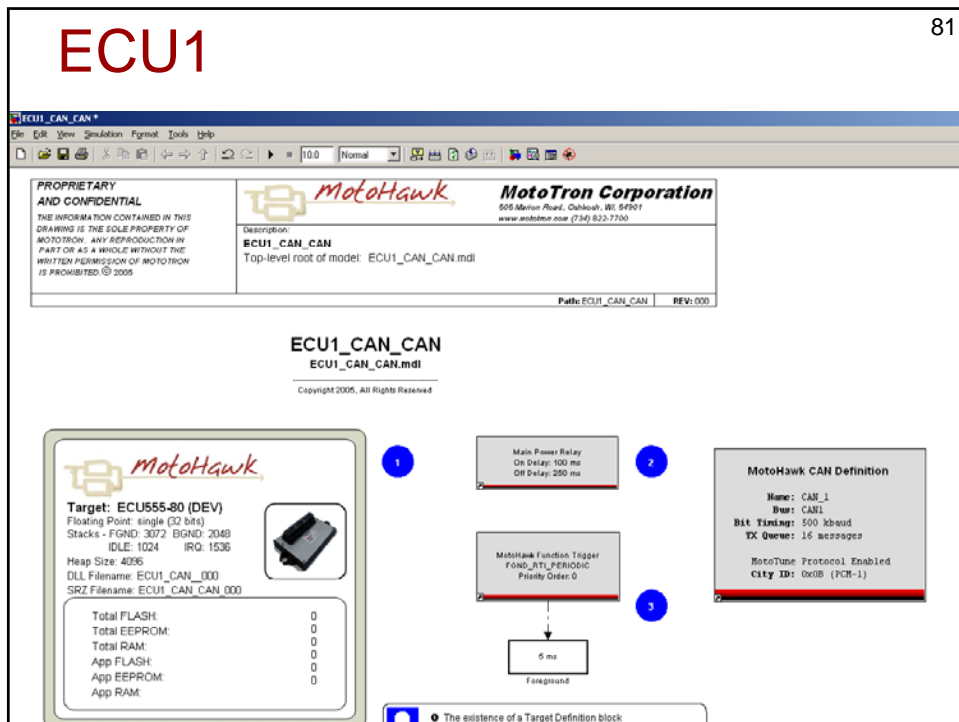


ECU1

80

- At the Matlab command prompt, enter the command:
`motohawk_project('ECU1_CAN_CAN')` to create a new model.
- In the top level of the model, place a MotoHawk CAN Definition block:
 - Part in library **MotoHawk /CAN Blocks**
 - Change the CAN rate to 500 k baud
 - Leave the CityID at 11 (hex B).

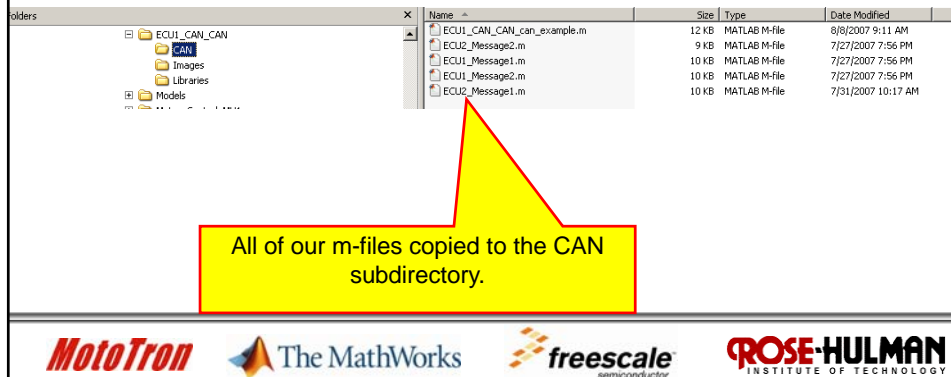




ECU1

83

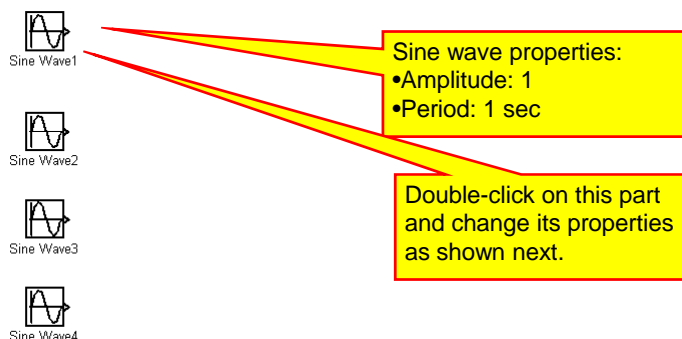
- Copy all of the CAN files to the CAN directory.
- By default, the MotoHawk CAN blocks will look in this directory for the m-files.



ECU1

84

- Next, open the foreground subsystem.
- Delete the controller and plant models.
- Place 4 Sine Wave parts in this subsystem (library **Simulink/Sources**).





85

ECU1

Time based chosen. We cannot use a sample based sine wave inside a triggered subsystem.





Frequency (rad/sec) = $2\pi(\text{Frequency in Hz})$
= $2\pi / (\text{Period in seconds})$

Since this part is inside a triggered subsystem, the sample time must be set to inherited (-1).

86

ECU1

- Use the same settings for the other sine waves except:
 - Sine Wave 2 should have a period of 2 seconds and amplitude of 3.
 - Sine Wave 3 should have a period of 3 seconds and amplitude of 5.
 - Sine Wave 4 should have a period of 4 seconds and amplitude of 7.





ECU1

87

- We want to send the values of the sine waves over the CAN bus.
- Place a part called **Send CAN Messages** in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the part and change the settings as shown:

MotoTron

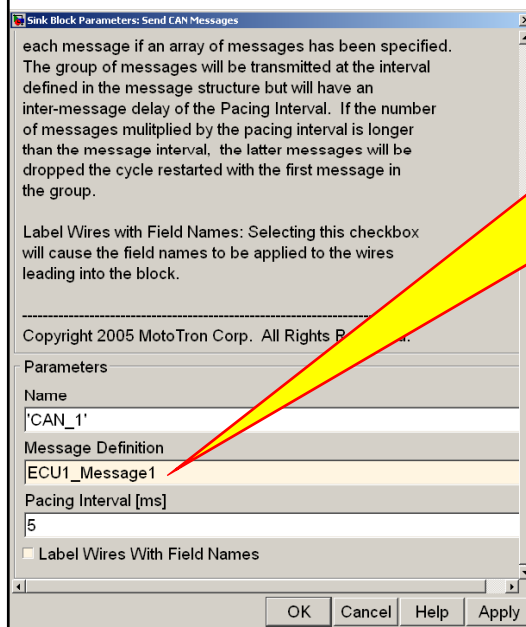
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU1

88



We created this m-file earlier.

- The name of the file was ECU1_Message1.m.
- We placed this file in the subdirectory named CAN.
- This m-file contains the signal definitions for this message.

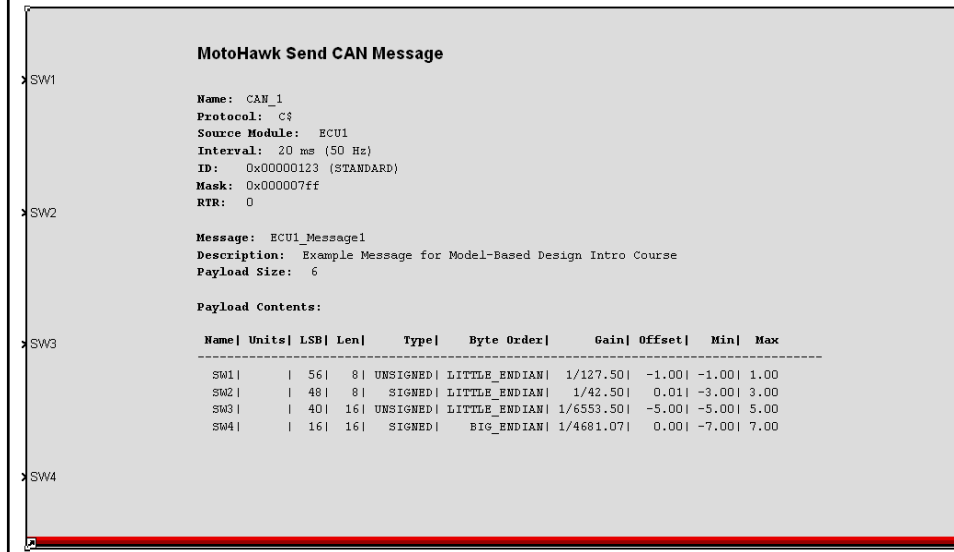
When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink input for every signal in the message.
- The block will display the properties of each signal and the CAN transmit rate.

ECU1

89

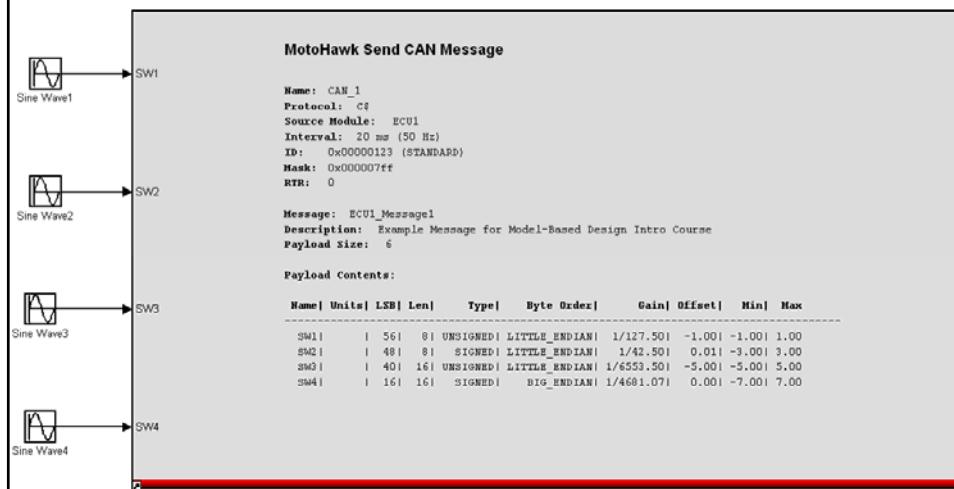
- Your block should look as shown.



ECU1

90

- Connect the sine wave sources to the CAN block.





ECU1

91

- ECU1 will also send out the following information that will be contained in message ECU1_Message2:
 - The potentiometer reading from the motor-generator.
 - A sine wave of amplitude 5 and period 5 seconds.
 - A periodic ramp signal of amplitude 100 and period of 5 seconds (mistakenly called a shark tooth).
 - The time since the ECU was last started.



ECU1

92

- For the potentiometer signal, we will use the same analog input as we used in the earlier project.
- Copy the potentiometer analog input from our previous project.
- Use a convert block to change the data type to double.
- Add an override so that we can change the value while debugging.

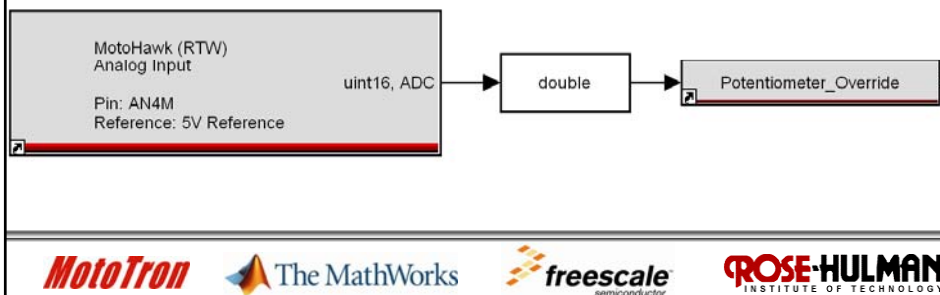




ECU1

93

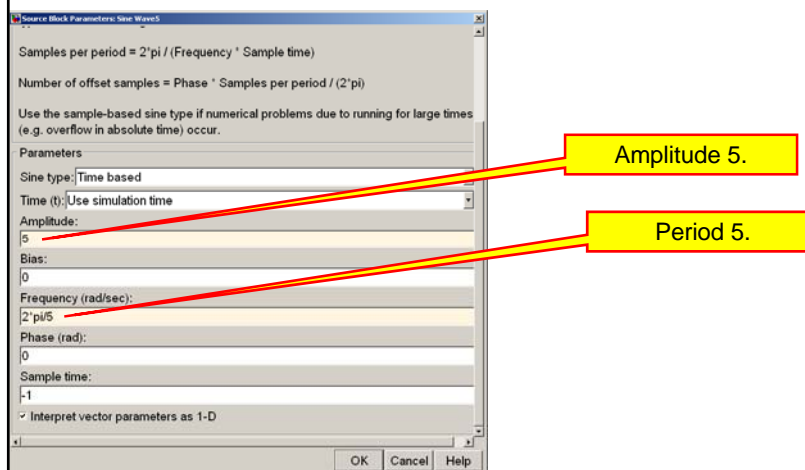
- As a reminder, the potentiometer used analog input AN4M.
- Do not scale the value. (We are transmitting the raw value in the range of 0 to 1023.)



ECU1

94

- For the sine wave, use the same part as we used for the first 4 sine waves and set the amplitude to 5 and period to 5:

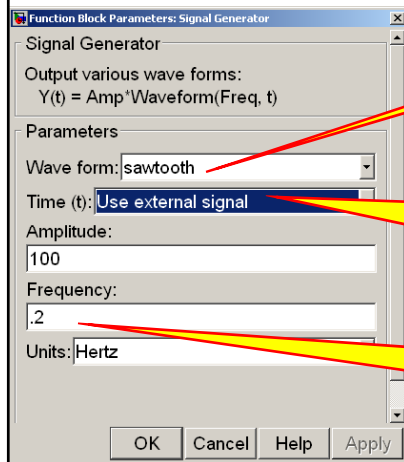




ECU1

95

- For our ramp, we will use a part called Signal Generator. (Located in library **Simulink/Sources**.)



Sawtooth chosen.

Note that we have chosen that the time reference for this block should be an external signal.

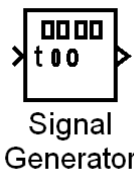
We will need to provide a signal for this block that corresponds to time.

Amplitude set to 100, Frequency set to 0.2 Hz (corresponding to a period of 5 seconds.)

ECU1

96

- When you click OK in the dialog box for the signal generator, you will notice that the Signal Generator has an input.
- This input is the time input for the block.

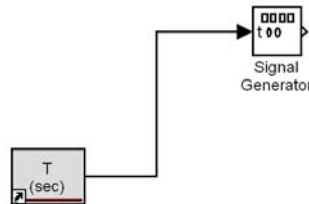




ECU1

97

- The last signal we need is a signal corresponding to time.
- MotoHawk provides a block called `motohawk_abs_time` (located in library **MotoHawk/Extra Development Blocks**).
- The output of this block is the time since the MotoTron ECU was last restarted.
- Place the block in your model and connect it to the input of the signal generator block.



ECU1

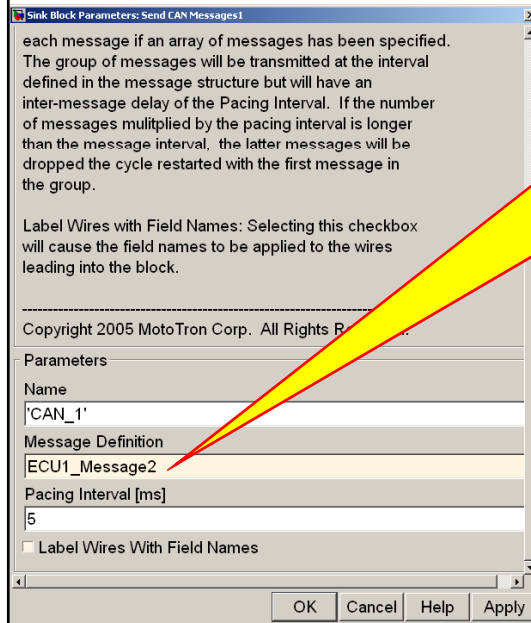
98

- We want to send the values of the signals just created over the CAN bus.
- Place a part called Send CAN Messages in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the part and change the settings as shown:



ECU1

99



We created this m-file earlier.

- The name of the file was ECU1_Message2.m.
- We placed this file in the subdirectory named CAN.
- This m-file contains the signal definitions for this message.

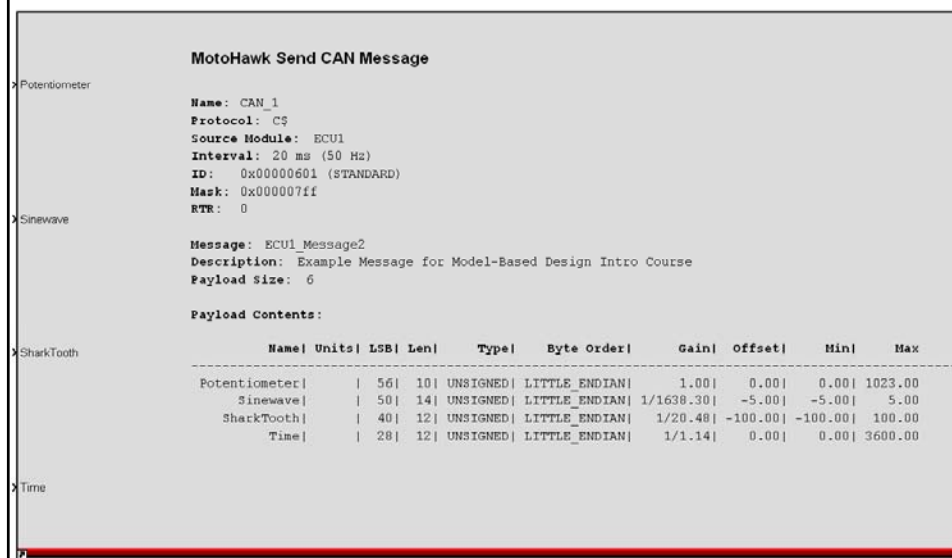
When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink input for every signal in the message.
- The block will display the properties of each signal and the CAN transmit rate.

ECU1

100

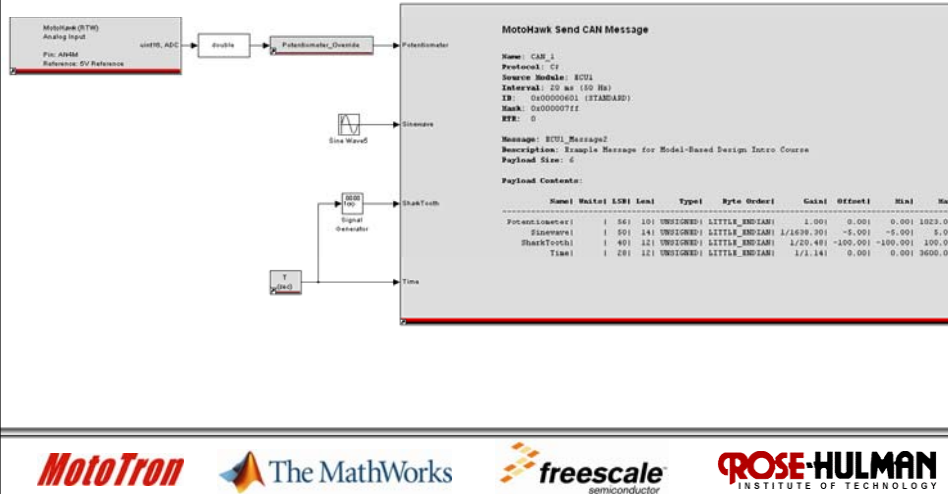
- Your block should look as shown.



ECU1

101

- Connect the CAN block as shown. (An enlargement is shown on the next slide.)



MotoTron

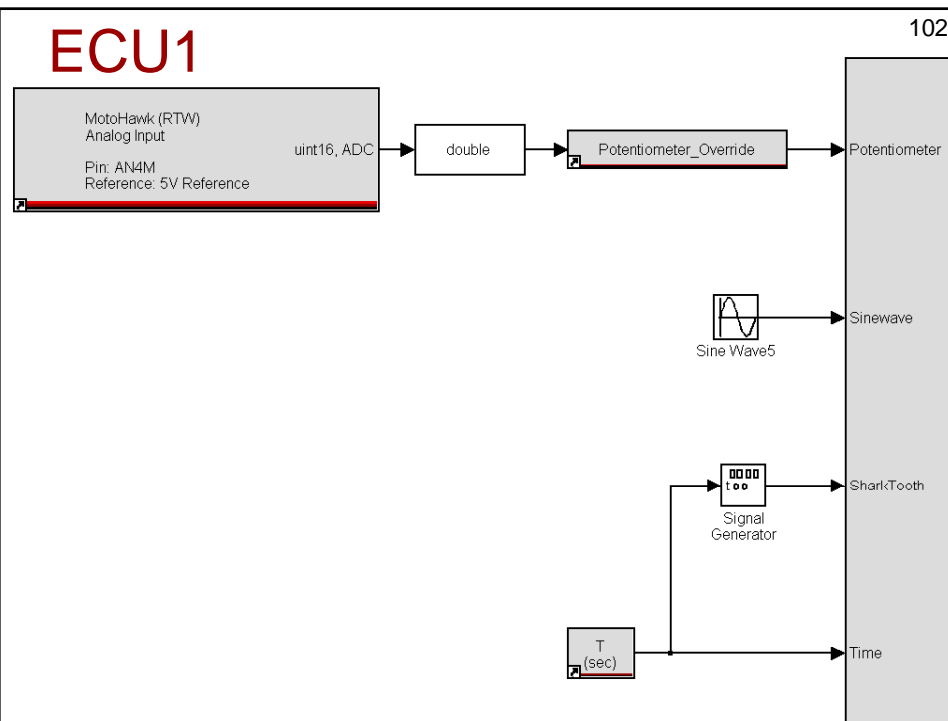
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU1

102





ECU1

103

- ECU1 will be receiving two can messages.
- We need to add a CAN receive block for each message we are receiving.
- Place a part called Read CAN Message in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the block and change the settings as shown:

MotoTron

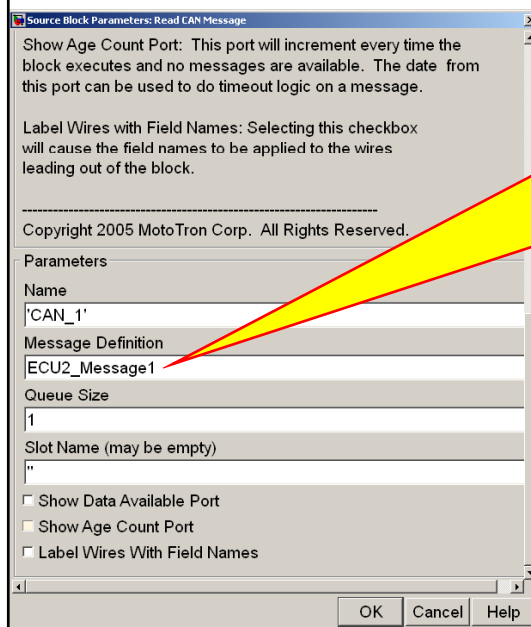
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU1

104



We created this m-file earlier.

- The name of the file was ECU2_Message1.m.

- We placed this file in the subdirectory named CAN.

- This m-file contains the signal definitions for this message.

When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink output for every signal in the message.

- The block will display the properties of each signal and the CAN transmit rate.



ECU1

105

MotoHawk Read CAN Message

Name: CAN_1
 Protocol: C\$
 Source Module: ECU2
 Interval: 20 ms (50 Hz)
 Queue Size: 1
 ID: 0x00000708 (STANDARD)
 Mask: 0x000007ff
 RTR: 0

Message: ECU2_Message1
 Description: Example Message for Model-Based Design Intro Course
 Payload Size: 3

Payload Contents:

Name	Units	LSB	Len	Type	Byte Order	Gain	Offset
Temperature		56	7	UNSIGNED	LITTLE_ENDIAN	1/1.270	0.000
Fred		63	5	UNSIGNED	LITTLE_ENDIAN	1.000	-3150.000
LED3		52	1	UNSIGNED	LITTLE_ENDIAN	1.000	0.000
Pulsewidth		53	10	UNSIGNED	LITTLE_ENDIAN	1/10.230	0.000

Temperature >

Fred >

LED3 >

Pulsewidth >

ECU1

106

- All signals will be connected to probes so we can observe their values.
- In addition, we will do the following:
- Fred:
 - If the value equals -3127, turn on one of the LEDs in the motor-generator system.
 - Use digital output part with pin FUELP. (Same as in our previous motor control exercise. – Should already be wired up correctly.)





ECU1

107

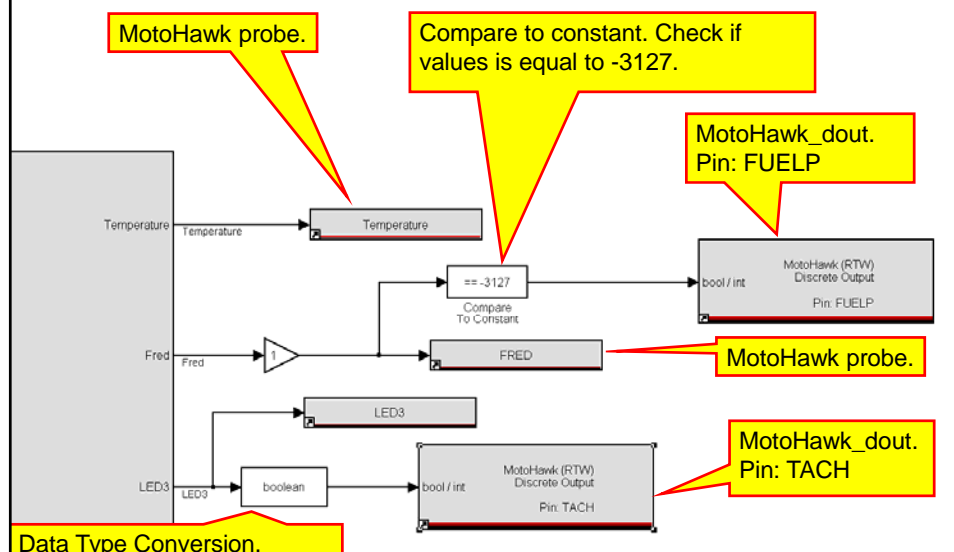
- LED3:
 - Convert to a Boolean type and turn on and off an LED in the motor-generator system.
 - Use digital output part with pin TACH. (Same as in our previous motor control exercise. – Should already be wired up correctly.)
- Pulsewidth:
 - The received signal has values from 0 to 100.
 - Convert to values from 0 to 4096.
 - Convert to type int16 and send out a PWM signal using the PWM output block with pin EST1.
 - (Same as in our previous motor control exercise. – Should already be wired up correctly.)

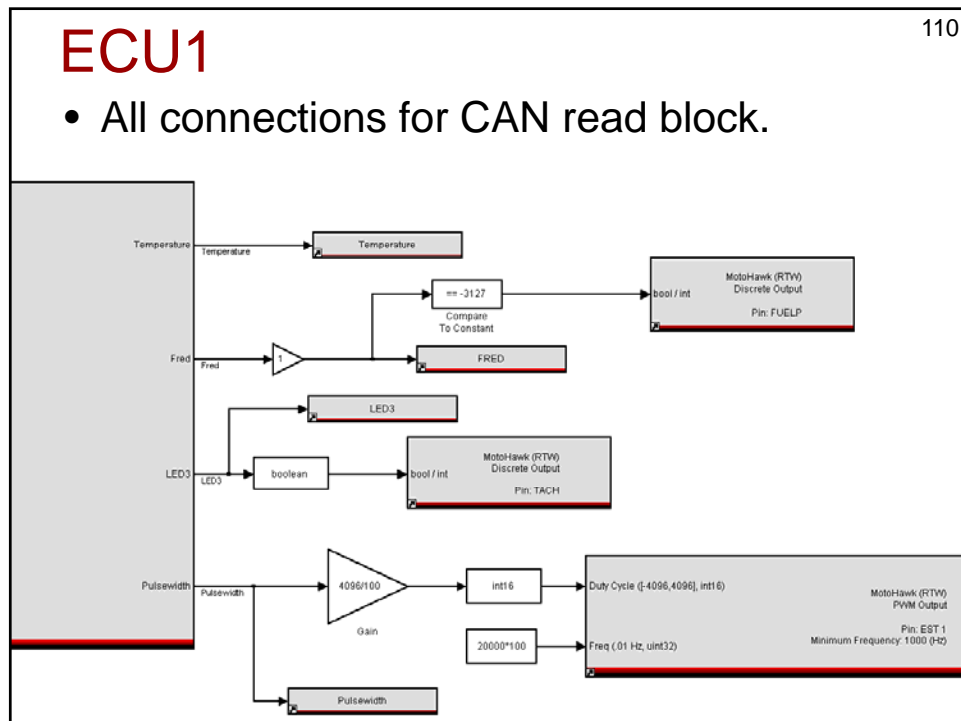
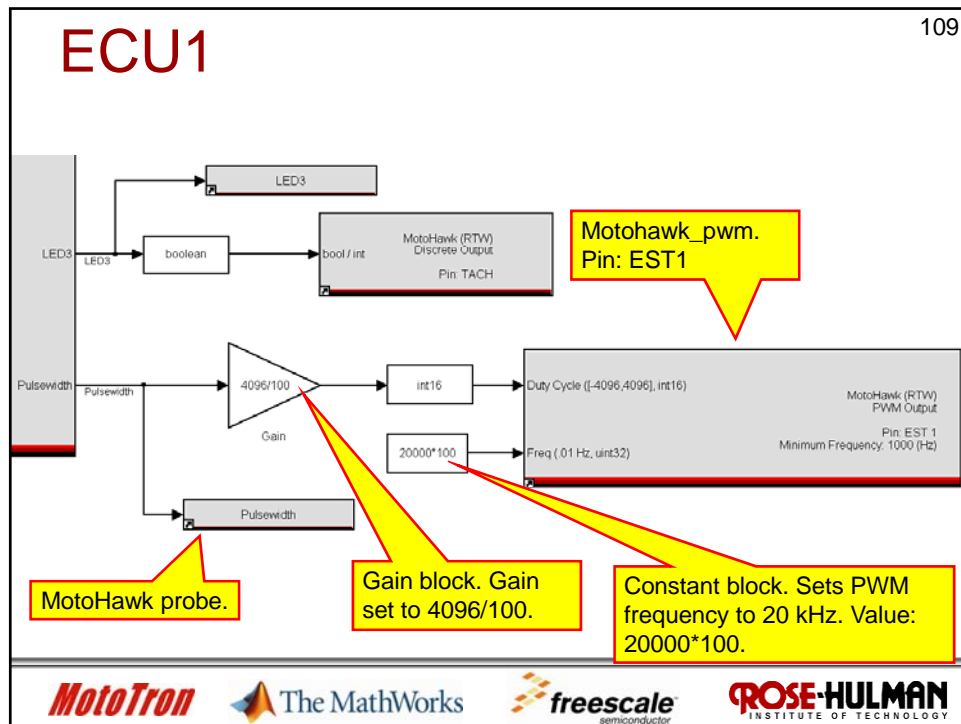


ECU1

108

- The next few slides show the connections.







ECU1

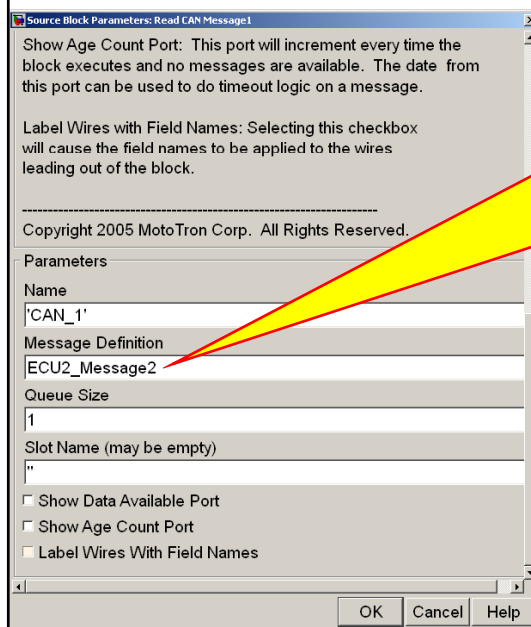
111

- The second CAN message that ECU1 will receive is ECU2_message2.
- This message has one signal called Cooling_Fan.
- We will display this signal with a probe.
- Place a part called Read CAN Message in your model. (Library **MotoHawk/CAN Blocks.**)
- Double-click on the block and change the settings as shown:



ECU1

112



We created this m-file earlier.

- The name of the file was ECU2_Message2.m.

- We placed this file in the subdirectory named CAN.

- This m-file contains the signal definitions for this message.

When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink output for every signal in the message.

- The block will display the properties of each signal and the CAN transmit rate.



ECU1

113

MotoHawk Read CAN Message

Name: CAN_1
Protocol: C5
Source Module: ECU2
Interval: 20 ms (50 Hz)
Queue Size: 1
ID: 0x00000124 (STANDARD)
Mask: 0x000007ff
RTR: 0

Message: ECU2_Message2
Description: Example Message for Model-Based Design Intro Course
Payload Size: 1

Cooling_Fan

Payload Contents:

Name	Units	LSB	Len	Type	Byte Order	Gain	Offset
Cooling_Fan		56	1	UNSIGNED	LITTLE_ENDIAN	1.000	0.000

ECU1

114

- Connect the Cooling_Fan signal to a MotoHawk probe.

MotoHawk Read CAN Message

Name: CAN_1
Protocol: C5
Source Module: ECU2
Interval: 20 ms (50 Hz)
Queue Size: 1
ID: 0x00000124 (STANDARD)
Mask: 0x000007ff
RTR: 0

Message: ECU2_Message2
Description: Example Message for Model-Based Design Intro Course
Payload Size: 1

Payload Contents:

Name	Units	LSB	Len	Type	Byte Order	Gain	Offset
Cooling_Fan		56	1	UNSIGNED	LITTLE_ENDIAN	1.000	0.000

Cooling_Fan

Cooling_Fan



ECU1

115

- We are now done building the model.
- Use the techniques covered previously to:
 - Check for consistency in data types:
 - Select **Format, Port/Signal Displays**, and then **Port Data Types** to display data types.
 - Type **ctrl-D** to evaluate your model for errors.
 - Build the Model (type **ctrl-b**)
 - Use MotoTune to download your model to your ECU.
 - Note: Do not connect both ECUs to the same CAN network yet.



ECU2 Slides

116

ECU1 Group skip ahead to slides
labeled with header ECU1/ECU2.
(Approximately slide number 166)





ECU2

117

- ECU2 will do the following:
 - Receive 4 sine waves over the CAN bus using message ECU1_Message1. The values will be displayed with probes.
 - Receive the potentiometer over the CAN bus using ECU1_Message2. Scale the signal from 0 to 4096 and spin the motor in the motor-generator system using the given duty cycle.
 - Receive signals for Time, a ramp (shark tooth), and a sine wave over the CAN bus using message ECU1_Message2. The values will be displayed with probes.

ECU2

118

- ECU2 will send the following information over the CAN bus in message ECU2_Message1:
 - Temperature. The value will be set with a MotoHawk override.
 - Fred and LED signals. The values will be set with a MotoHawk overrides.
 - Pulsewidth. The value will be read from the potentiometer on the motor-generator system and scaled to values from 0 to 100. An override will also be used for debugging.





ECU2

119

- ECU2 will send the following information over the CAN bus in message ECU2_Message1:
 - Cooling Fan signal. The value will be set with a MotoHawk override.

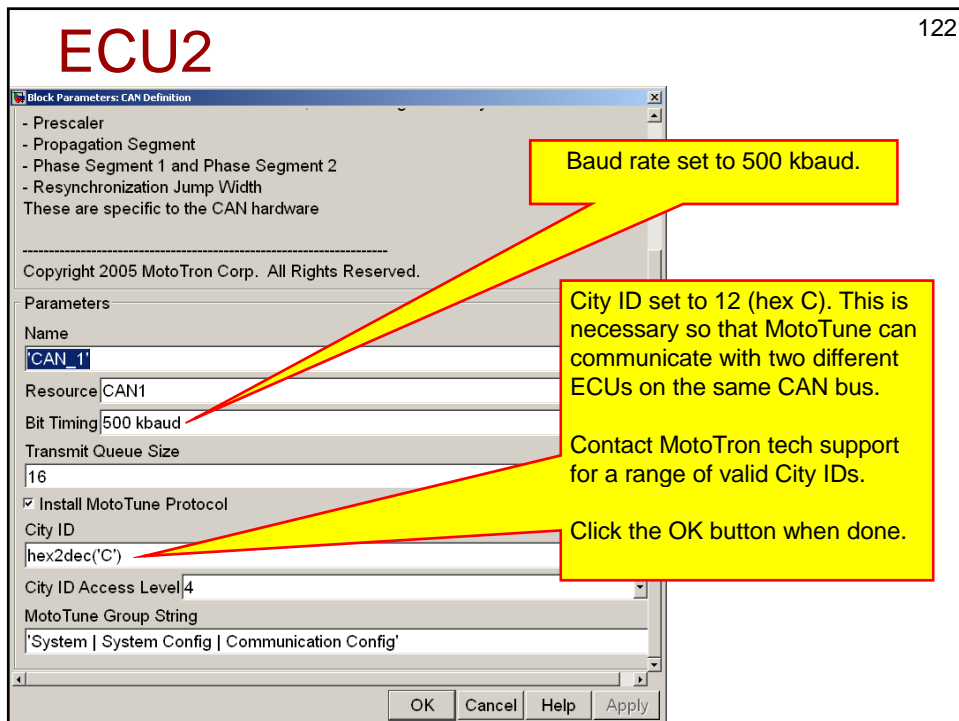
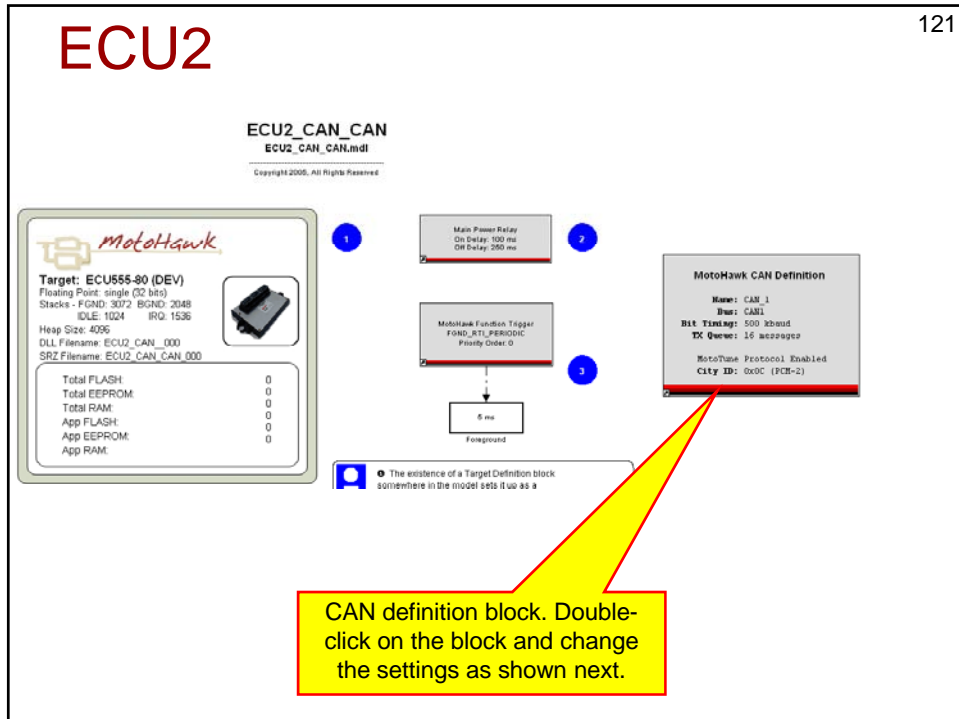


ECU2

120

- At the Matlab command prompt, enter the command:
`motohawk_project('ECU2_CAN_CAN')` to create a new model.
- In the top level of the model, place a MotoHawk CAN Definition block:
 - Part in library **MotoHawk /CAN Blocks**
 - Change the CAN rate to 500 k baud
 - Change the CityID to 12 (hex C).

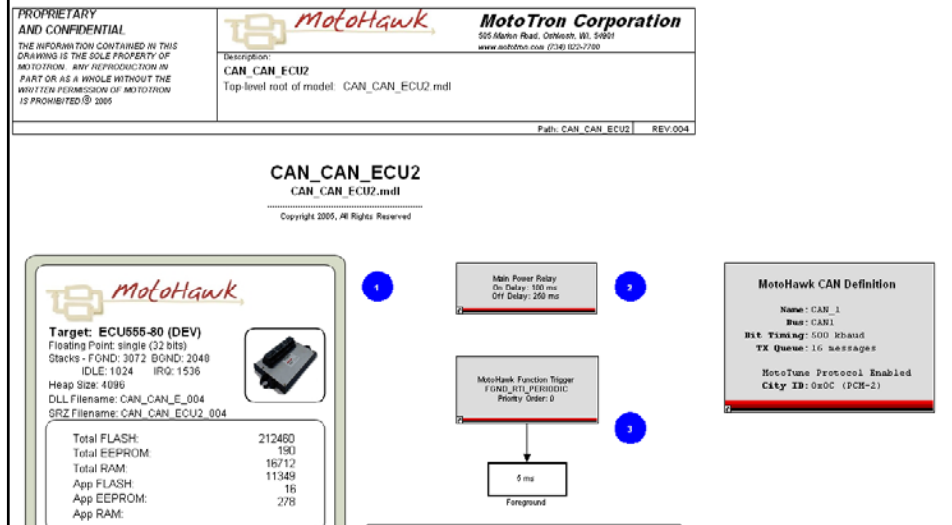






ECU2

123

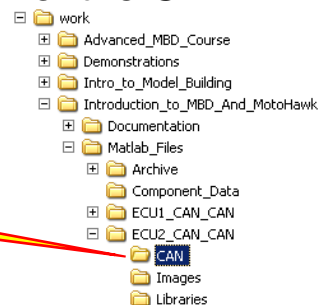


ECU2

124

- When you create a project using the command `motohawk_project`, a new directory is created with the same name as the project.
- Inside the new directory is a subdirectory called CAN. We will place all of the CAN files in this directory.

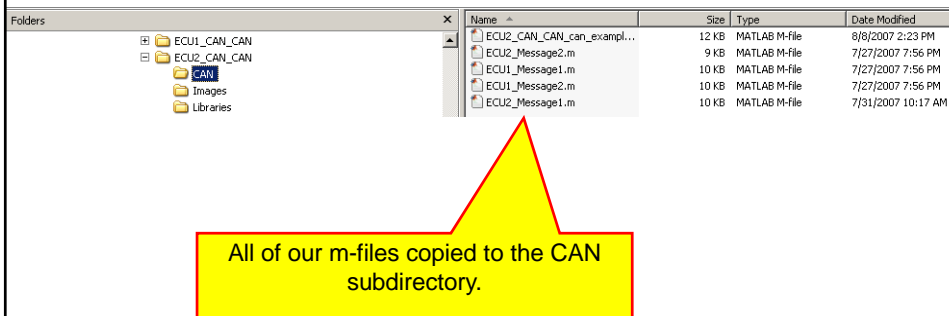
CAN m-files should be placed in this directory.



ECU2

125

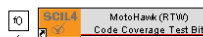
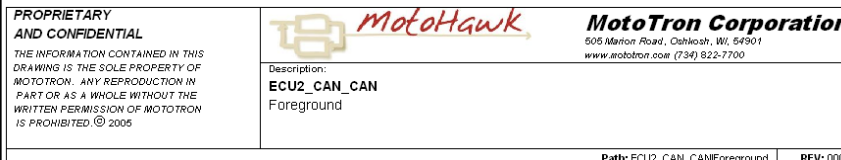
- Copy all of the CAN files to the CAN directory.
- By default, the MotoHawk CAN blocks will look in this directory for the m-files.



ECU2

126

- Next, open the foreground subsystem.
- Delete the controller and plant models.
- The foreground subsystem should be empty except for the two blocks shown:





ECU2

127

- ECU2 will be receiving two can messages.
- We need to add a CAN receive block for each message we are receiving.
- Place a part called Read CAN Message in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the block and change the settings as shown:

MotoTron

 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2

128

Source Block Parameters: Read CAN Message

Label Wires with Field Names: Selecting this checkbox will cause the field names to be applied to the wires leading out of the block.

Copyright 2005 MotoTron Corp. All Rights Reserved.

Parameters

Name
CAN_1

Message Definition
ECU1_Message1

Queue Size
1

Slot Name (may be empty)
"

☐ Show Data Available Port

☐ Show Age Count Port

☐ Label Wires With Field Names

OK Cancel

We created this m-file earlier.

- The name of the file was ECU1_Message1.m.

- We placed this file in the subdirectory named CAN.

- This m-file contains the signal definitions for this message.

When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink output for every signal in the message.

- The block will display the properties of each signal and the CAN transmit rate.



ECU2

129

MotoHawk Read CAN Message

Name: CAN_1
Protocol: C\$
Source Module: ECU1
Interval: 20 ms (50 Hz)
Queue Size: 1
ID: 0x00000123 (STANDARD)
Mask: 0x000007ff
RTR: 0

Message: ECU1_Message1
Description: Example Message for Model-Based Design Intro Course
Payload Size: 6

Payload Contents:

Name	Units	LSB	Len	Type	Byte Order	Gain	Offset
SW1		56	8	UNSIGNED	LITTLE_ENDIAN	1/127.500	-1.000
SW2		48	8	SIGNED	LITTLE_ENDIAN	1/42.501	0.012
SW3		40	16	UNSIGNED	LITTLE_ENDIAN	1/6553.501	-5.000
SW4		16	16	SIGNED	BIG_ENDIAN	1/4681.072	0.000

SW1

SW2

SW3

SW4

ECU2

130

- All signals will be connected to probes so we can observe their values.

MotoHawk Read CAN Message

Name: CAN_1
Protocol: C\$
Source Module: ECU1
Interval: 20 ms (50 Hz)
Queue Size: 1
ID: 0x00000123 (STANDARD)
Mask: 0x000007ff
RTR: 0

Message: ECU1_Message1
Description: Example Message for Model-Based Design Intro Course
Payload Size: 6

Payload Contents:

Name	Units	LSB	Len	Type	Byte Order	Gain	Offset
SW1		56	8	UNSIGNED	LITTLE_ENDIAN	1/127.500	-1.000
SW2		48	8	SIGNED	LITTLE_ENDIAN	1/42.501	0.012
SW3		40	16	UNSIGNED	LITTLE_ENDIAN	1/6553.501	-5.000
SW4		16	16	SIGNED	BIG_ENDIAN	1/4681.072	0.000

SW1

SW2

SW3

SW4

SW1

SW2

SW3

SW4



ECU2

131

- The second CAN message that ECU2 receives is ECU1_Message2.
- Place a part called Read CAN Message in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the block and change the settings as shown:

MotoTron

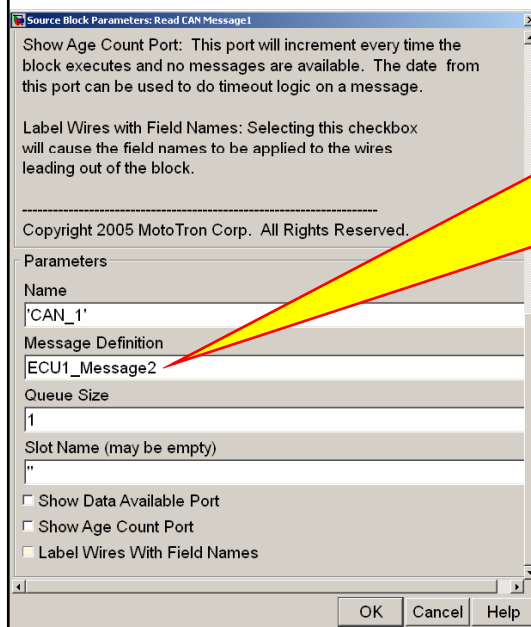
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2

132



We created this m-file earlier.

- The name of the file was ECU1_Message2.m.

- We placed this file in the subdirectory named CAN.

- This m-file contains the signal definitions for this message.

When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink output for every signal in the message.

- The block will display the properties of each signal and the CAN transmit rate.



ECU2

133

MotoHawk Read CAN Message

Name: CAN_1
 Protocol: CS
 Source Module: ECU1
 Interval: 20 ms (50 Hz)
 Queue Size: 1
 ID: 0x00000601 (STANDARD)
 Mask: 0x000007ff
 RTR: 0

Message: ECU1_Message2
 Description: Example Message for Model-Based Design Intro Course
 Payload Size: 6

Payload Contents:

Name	Units	LSB	Len	Type	Byte Order	Gain	Offset
Potentiometer		56	10	UNSIGNED	LITTLE_ENDIAN	1.000	0.000
Sinewave		50	14	UNSIGNED	LITTLE_ENDIAN	1/1638.300	-5.000
SharkTooth		40	12	UNSIGNED	LITTLE_ENDIAN	1/20.475	-100.000
Time		28	12	UNSIGNED	LITTLE_ENDIAN	1/1.138	0.000

Potentiometer >

Sinewave >

SharkTooth >

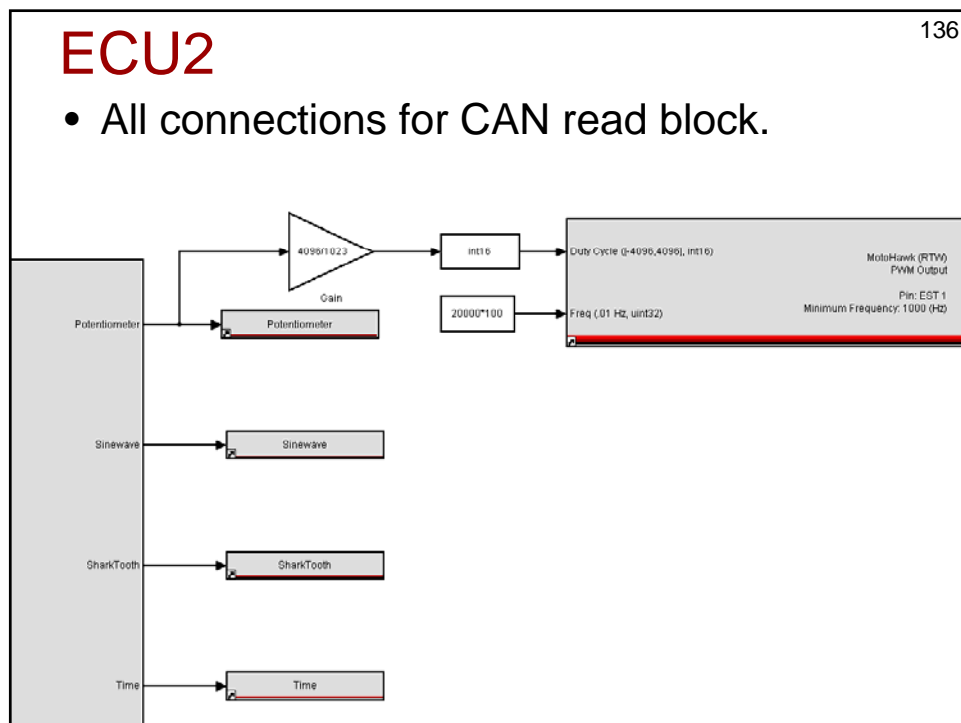
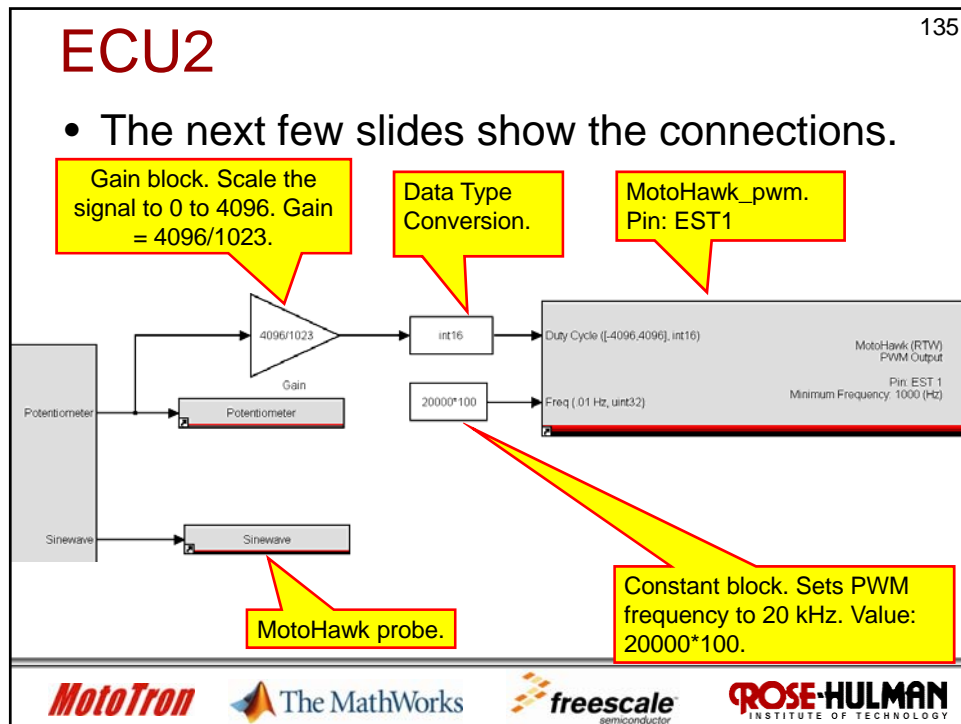
Time >

ECU2

134

- All signals will be connected to probes so we can observe their values.
- In addition, we will do the following to the Potentiometer signal:
 - Scale the signal from 0 to 1023, to 0 to 4096.
 - Convert the signal to an int16 data type.
 - Output the signal with a MotoHawk_pwm block on EST1 and spin the motor with the specified duty cycle.







ECU2

137

- ECU2 will transmit the following information that will be contained in message ECU2_Message1:
 - Temperature. A value from 0 to 100. This value will be set with a MotoHawk override.
 - Fred. A value from -3150 to -3120. This value will be set with a MotoHawk override.
 - LED3. A value from 0 to 1. This value will be set with a MotoHawk override.
 - Pulsewidth. A value from 0 to 100. The value will be obtained from the potentiometer on the motor-generator system. An override will also be used for debugging purposes.



ECU2

138

- For the pulsewidth signal, we will use the same analog input as we used in the earlier project.
- Copy the potentiometer analog input from our previous project.
- Use convert block to change the data type to double.
- Use a gain block to scale the signal from 0 to 100.
- Add an override so that we can change the value while debugging.

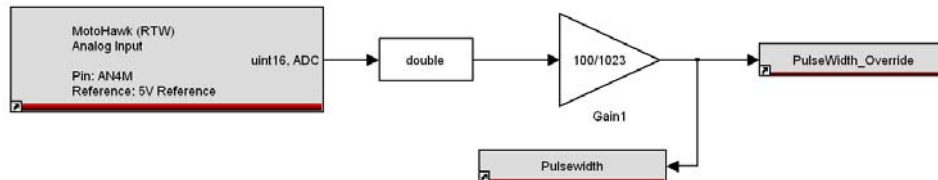




ECU2

139

- As a reminder, the potentiometer used analog input AN4M.



MotoTron

The MathWorks

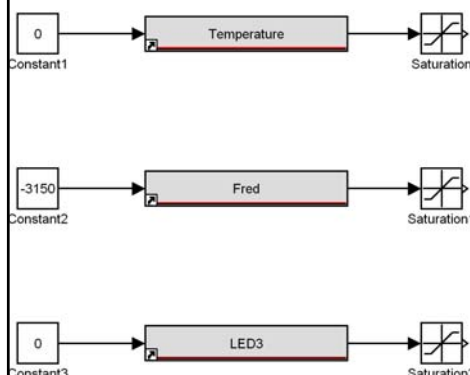
freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2

140

- The three other signals in this message use overrides to set the values.
- To protect against the user making an error when setting an override, we will add saturation blocks to limit the signals.



Temperature Saturation limits: 0 to 100.

FRED Saturation Limits: -3150 to -3120.

LED3 Saturation limits: 0 to 1.



ECU2

141

- We want to send the values of the signals just created over the CAN bus.
- Place a part called Send CAN Messages in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the part and change the settings as shown:

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2

142

Sink Block Parameters: Send CAN Messages

inter-message delay of the Pacing Interval. If the number of messages multiplied by the pacing interval is longer than the message interval, the latter messages will be dropped the cycle restarted with the first message in the group.

Label Wires with Field Names: Selecting this checkbox will cause the field names to be applied to the wires leading into the block.

Copyright 2005 MotoTron Corp. All Rights Reserved

Parameters

Name
CAN_1

Message Definition
ECU2_Message1

Pacing Interval [ms]
5

☐ Label Wires With Field Names

OK Cancel Help

We created this m-file earlier.

- The name of the file was ECU2_Message1.m.

- We placed this file in the subdirectory named CAN.

- This m-file contains the signal definitions for this message.

When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

- There will be one Simulink input for every signal in the message.

- The block will display the properties of each signal and the CAN transmit rate.

ECU2

143

- Your block should look as shown.

MotoHawk Send CAN Message

Name: CAN_1
 Protocol: C5
 Source Module: ECU2
 Interval: 20 ms (50 Hz)
 ID: 0x00000708 (STANDARD)
 Mask: 0x000007ff
 RTR: 0

Message: ECU2_Message1
 Description: Example Message for Model-Based Design Intro Course
 Payload Size: 3

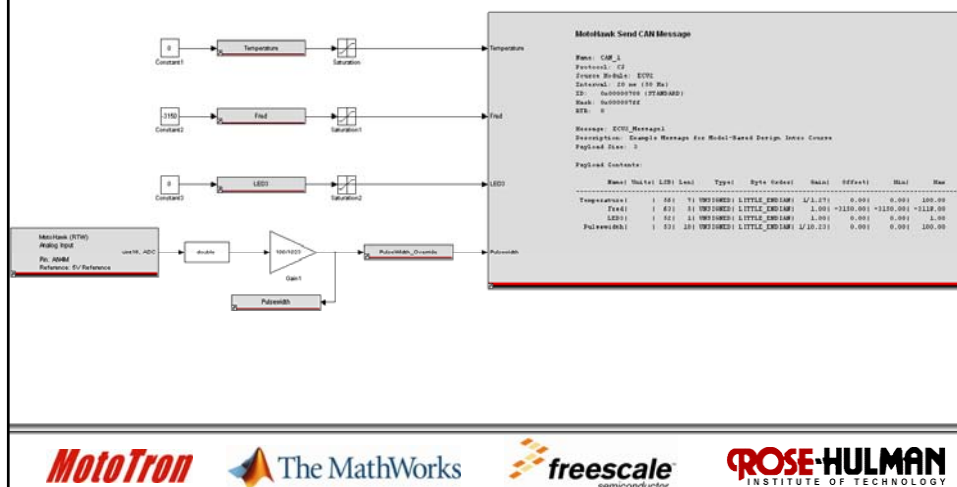
Payload Contents:

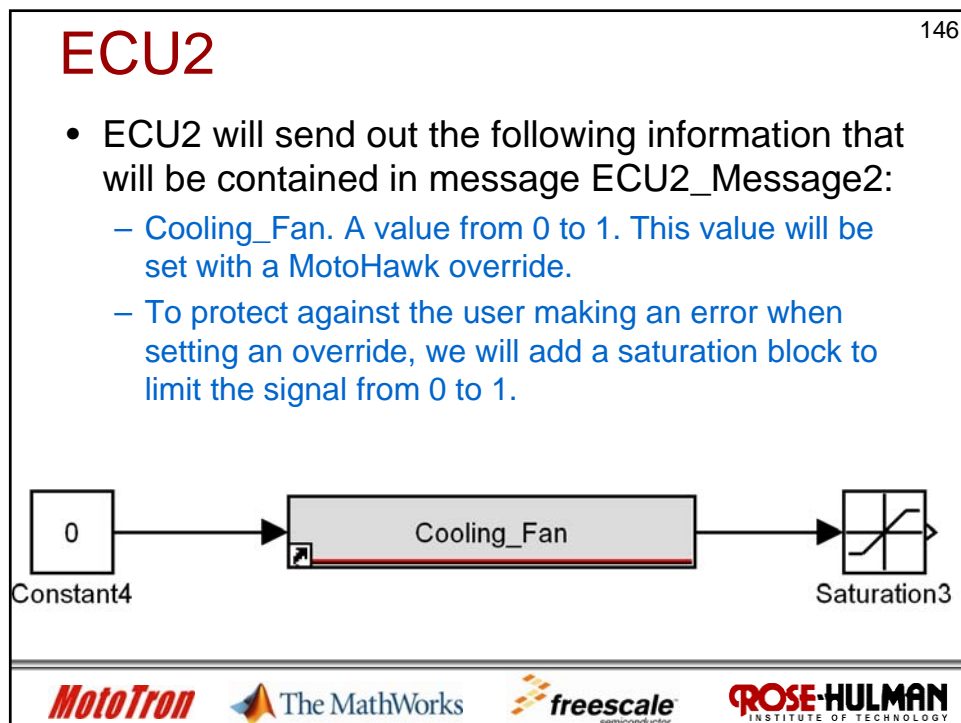
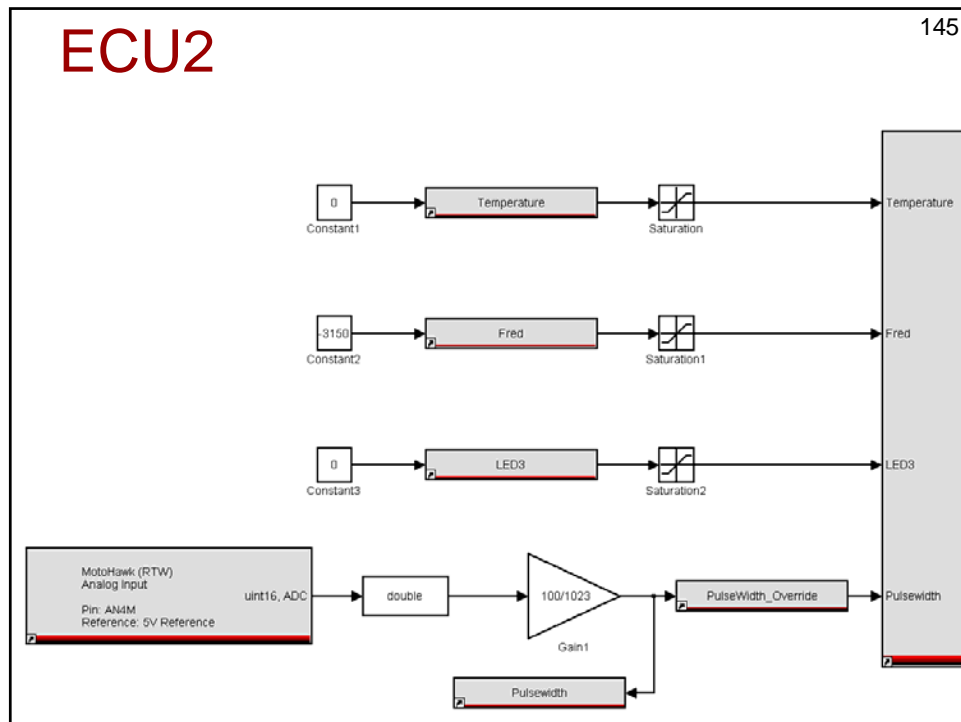
Name	Units	LSB	Len	Type	Byte Order	Gain	Offset	Min	Max
Temperature		56	7	UNSIGNED	LITTLE_ENDIAN	1/1.27	0.00	0.00	100.00
Fred		63	5	UNSIGNED	LITTLE_ENDIAN	1.00	-3150.00	-3150.00	-3119.00
LED3		52	1	UNSIGNED	LITTLE_ENDIAN	1.00	0.00	0.00	1.00
Pulsewidth		53	10	UNSIGNED	LITTLE_ENDIAN	1/10.23	0.00	0.00	100.00

ECU2

144

- Connect the CAN block as shown. (An enlargement is shown on the next slide.)







ECU2

147

- We want to send the value of the Cooling_Fan signal just created over the CAN bus.
- Place a part called Send CAN Messages in your model. (Library **MotoHawk/CAN Blocks**.)
- Double-click on the part and change the settings as shown:

MotoTron

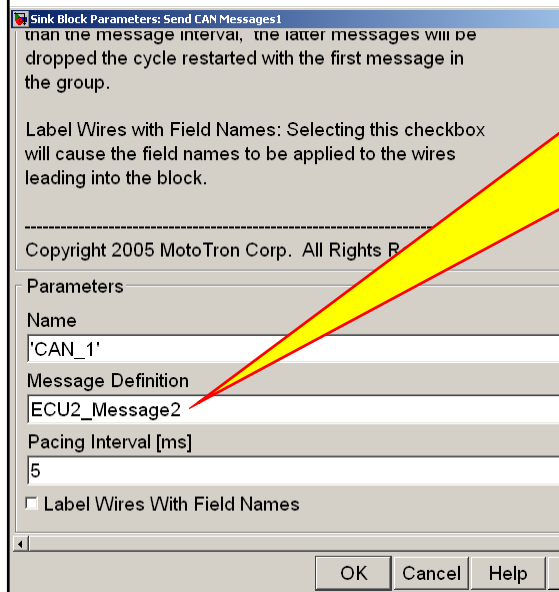
 **The MathWorks**

 **freescale**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2

148



We created this m-file earlier.

- The name of the file was ECU2_Message2.m.

- We placed this file in the subdirectory named CAN.

- This m-file contains the signal definitions for this message.

When you click the OK button, if MotoHawk can find the m-file, the block properties will change:

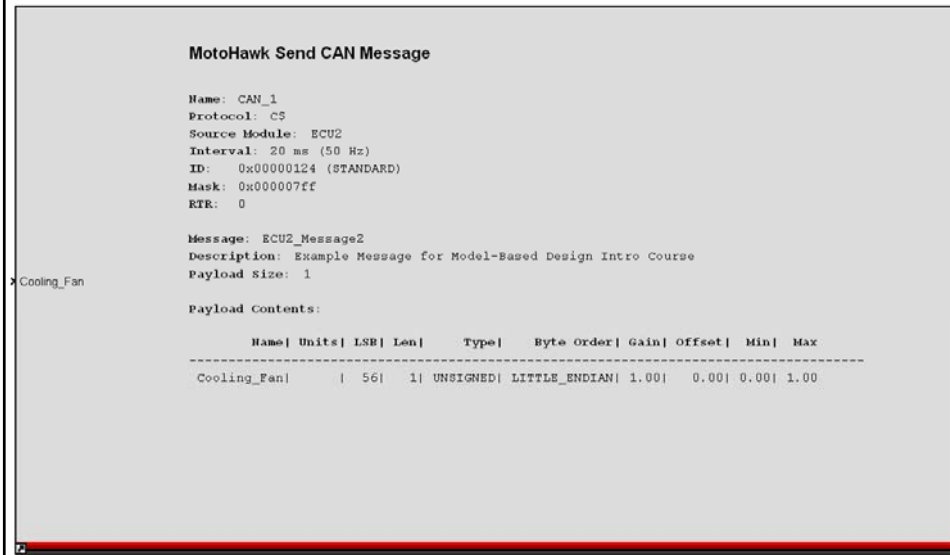
- There will be one Simulink input for every signal in the message.

- The block will display the properties of each signal and the CAN transmit rate.

ECU2

149

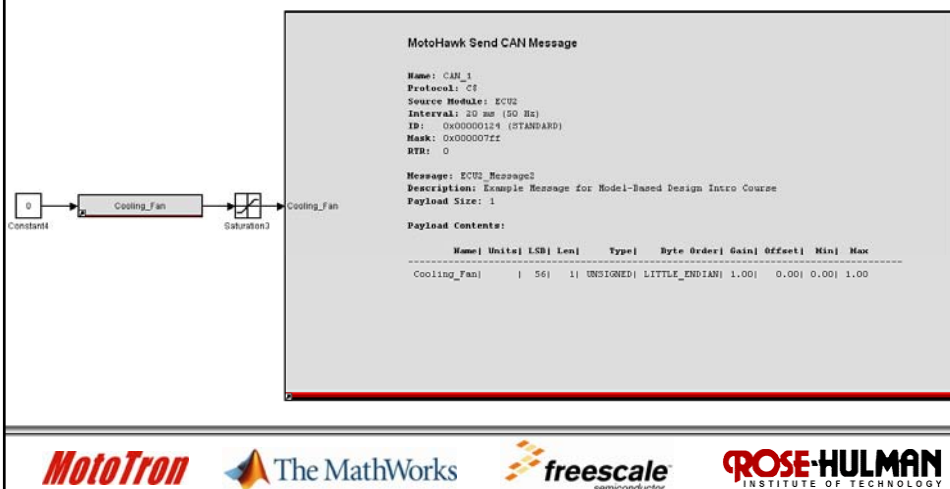
- Your block should look as shown.



ECU2

150

- Connect the CAN block as shown.





ECU2

151

- We are finished building the model.
- Use the techniques covered previously to:
 - Check for consistency in data types:
 - Select **Format, Port/Signal Displays**, and then **Port Data Types** to display data types.
 - Type **ctrl-D** to evaluate your model for errors.
 - Build the Model (type **ctrl-b**)
 - Use MotoTune to download your model to your ECU. (See Next Slide!!!)
 - Note: Do not connect both ECUs to the same CAN network yet.



ECU2

152

- If you recall, in the top level of model ECU2_CAN_CAN, we changed the City ID of ECU2 to 12 (hex C).
- Since we have not yet programmed ECU2 with the new model, ECU2 still has a City ID of 11.
- Thus, we program ECU2 with MotoTune the same as we did in our last example.
- Once we program ECU2 with the new model, we will need to make some changes in the MotoTune ports.
- (Program your ECU with the new model if you have not yet done so.)



ECU2


153

- Once ECU2 has been programmed with the new model, we need to change (or add) a port for PCM-2 (City ID 12).
- We used a procedure earlier to change/verify the properties of the MotoServer ports.
- We will repeat a similar procedure here.



ECU2 - MotoServer

154

- We now need to add a port for City ID 12 (PCM-2) using MotoServer.
- The MotoServer icon  should be located in your windows tray.

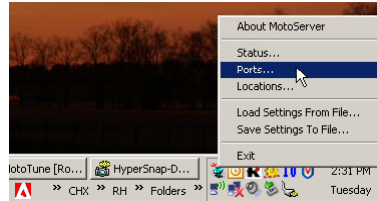
MotoServer icon.



ECU2 - MotorServer

155

- Right-click on the MotoServer icon and select **Ports**



MotoTron

 The MathWorks

 freescale
semiconductor

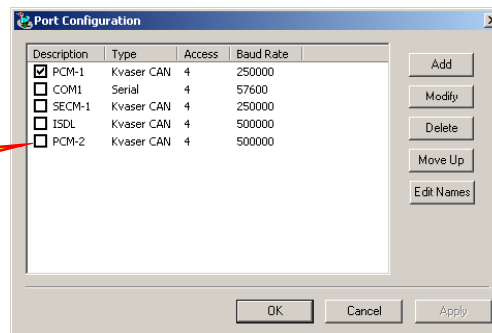
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2 - MotorServer

156

- You may or may not have a port called PCM-2.
- If you have a PCM-2 port, the settings should be:
 - Kvaser CAN
 - Access 4
 - Baud Rate 500000

If you have this port, enable it and disable the PCM-1 port.





ECU2 - MotoServer

157

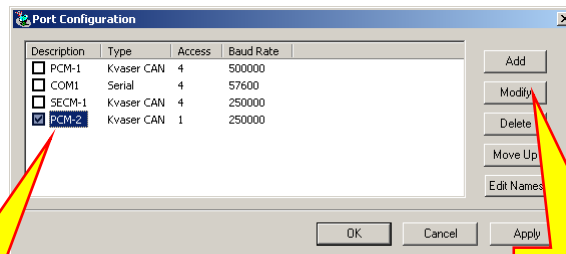
- If PCM-2 port settings are not:
 - Kvaser CAN
 - Access 4
 - Baud Rate 500000
- Then you will need to change the port settings.
- Skip to slide 166 if your port settings are correct.



ECU2 - MotorServer

158

- If you need to change the port settings, select PCM-2 and click the **Modify** button.

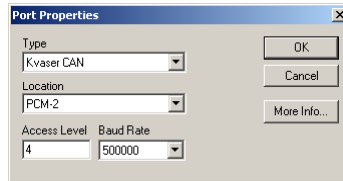




ECU2 - MotorServer

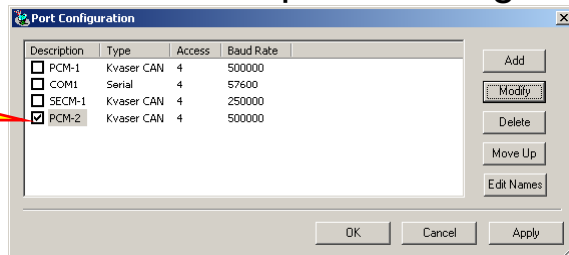
159

- Change the settings as shown:



- Click the **OK** button to accept the changes.

Make sure that only port PCM-2 is enabled.



MotoTron

The MathWorks

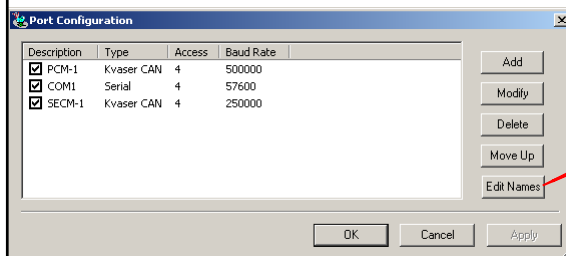
freescale semiconductor

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

ECU2 - MotoServer

160

- If your port settings are correct, skip to slide 166.
- If your window does not have a port named PCM-2, you must do the following:
- Click the **Edit Names** button

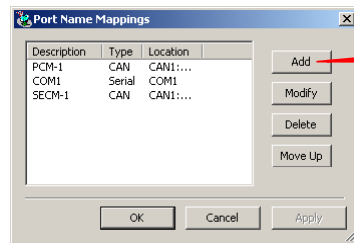


Click here.

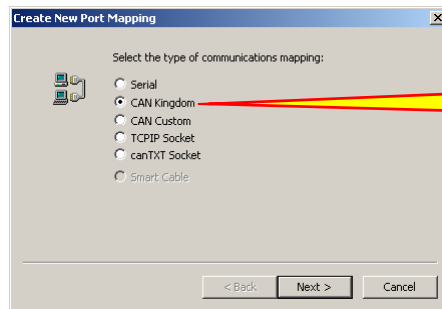


ECU2 - MotoServer

161



Click this **Add** button.

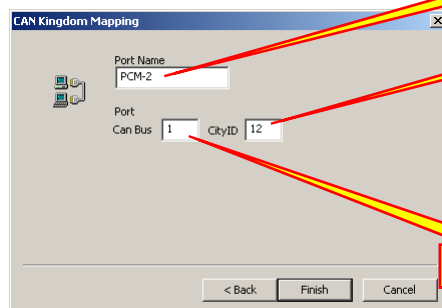


Select **CAN Kingdom** and then click the **Next** button.

ECU2 - MotoServer

162

- Fill in the dialog box as shown and click the **Finish** button.



Name is **PCM-2**.

CityID is **12**.

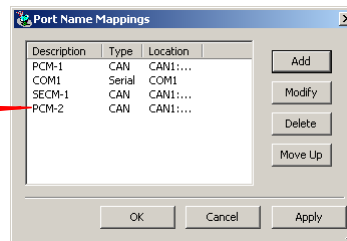
CAN Bus 1.

ECU2 - MotoServer

163

- PCM-2 should be added to the CAN Bus Mappings.

Name listed here.



- Click the **OK** button.

MotoTron

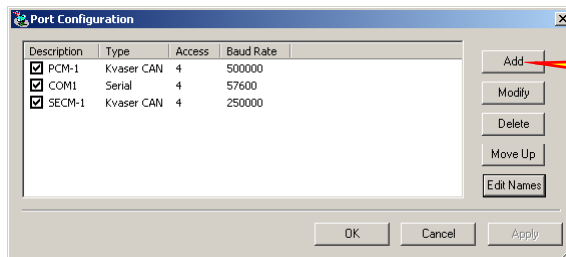
 The MathWorks

 freescale
semiconductor

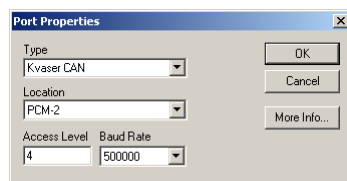
ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

ECU2 - MotoServer

164



Click this **Add** button.



Fill in properties as shown:

- Type – Kvaser CAN
- Location PCM-2
- Access Level 4
- Baud Rate 500000
- Click the **OK** button when done.

MotoTron

 The MathWorks

 freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

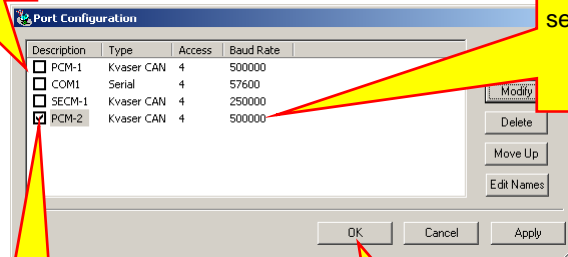


ECU2 - MotoServer

165

Disable all other ports.

The port should be added with the proper settings.



Port is enabled.

Click the **OK** button. We are ready to go.



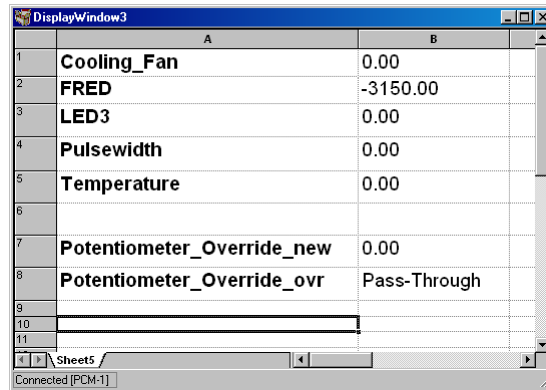
ECU1 / ECU2 CAN

166

- We can now connect the two 6-port hubs together with a CAN Cable. (Non-terminated at both ends.)
- Remove one of the key switches. (One key switch will turn on both ECUs.)
- Run MotoTune on each PC and open a display to your ECU.
 - PC1 connect to ECU1 (PCM-1 port)
 - PC2 connect to ECU2 (PCM-2 port)
- Both PC1 and PC2 show all probes and overrides on the display.

PC1 – ECU1 Display

167



	A	B
1	Cooling_Fan	0.00
2	FRED	-3150.00
3	LED3	0.00
4	Pulsewidth	0.00
5	Temperature	0.00
6		
7	Potentiometer_Override_new	0.00
8	Potentiometer_Override_ovr	Pass-Through
9		
10		
11		

Connected [PCM-1]

MotoTron

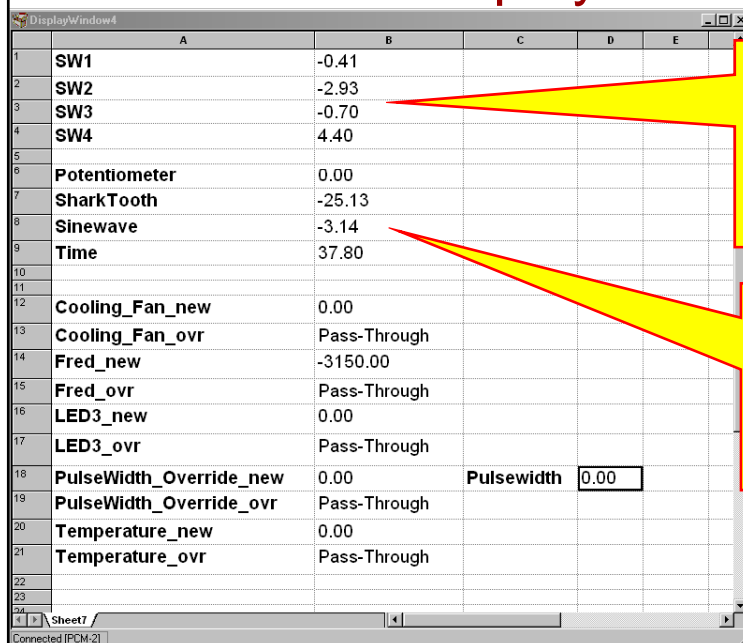
The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

PC2 – ECU2 Display

168



	A	B	C	D	E
1	SW1	-0.41			
2	SW2	-2.93			
3	SW3	-0.70			
4	SW4	4.40			
5					
6	Potentiometer	0.00			
7	SharkTooth	-25.13			
8	Sinewave	-3.14			
9	Time	37.80			
10					
11					
12	Cooling_Fan_new	0.00			
13	Cooling_Fan_ovr	Pass-Through			
14	Fred_new	-3150.00			
15	Fred_ovr	Pass-Through			
16	LED3_new	0.00			
17	LED3_ovr	Pass-Through			
18	PulseWidth_Override_new	0.00	Pulsewidth	0.00	
19	PulseWidth_Override_ovr	Pass-Through			
20	Temperature_new	0.00			
21	Temperature_ovr	Pass-Through			
22					
23					
24					

Connected [PCM-2]

Quick check:
Signals SW1
through SW4
should be
changing
continuously.

Quick check: All
of these signals
should be
changing except
the Potentiometer
signal.



Testing

169

- Next, we will change values of the overrides on PC2/ECU2 and we should see the probe values change on PC1/ECU1.
- Test each override for several values within each signal's range.
- I will show the display for both PCs on the same slide. You will have the displays shown on two different PC screens.

MotoTron

The MathWorks

freescale
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Testing

170

DisplayWindow1

	A	B
1	Cooling_Fan	1.00
2	FRED	-3150.00
3	LED3	0.00
4	Pulsewidth	0.00
5	Temperature	0.00
6	Potentiometer Override_new	0.00
7	Potentiometer Override_ovr	Pass-Through

DisplayWindow4

	A	B	C	D	E
1	SW1	-0.02			
2	SW2	0.01			
3	SW3	4.35			
4	SW4	-7.00			
5	Potentiometer	0.00			
6	SharkTooth	21.66			
7	Sinewave	2.31			
8	Time	487.03			
9					
10					
11					
12	Cooling_Fan_new	1.00			
13	Cooling_Fan_ovr	Override			
14	Fred_new	-3150.00			
15	Fred_ovr	Pass-Through			
16	LED3_new	0.00			
17	LED3_ovr	Pass-Through			
18	PulseWidth Override_new	0.00	Pulsewidth	0.00	
19	PulseWidth Override_ovr	Pass-Through			
20	Temperature_new	0.00			
21	Temperature_ovr	Pass-Through			

Callout 1 (Yellow box): To see a faster response, you may want to set the update rate of a cell to fast. (Right-click on a cell and select **Properties**. Next, click the **Set Fast** button and then click the **OK** button.)

Callout 2 (Yellow box): This value should follow a change in the indicated override. Valid range is 0 to 1.



Testing

171

This value should follow a change in the indicated override. Valid range is -3150 to -3120.

Sheet1	Sheet2
Cooling_Fan	SW1
FRED	SW2
LED3	SW3
Pulsewidth	SW4
Temperature	Potentiometer
Potentiometer_Override_new	SharkTooth
Potentiometer_Override_ovr	Sinewave
	Time
	Cooling_Fan_new
	Cooling_Fan_ovr
	Fred_new
	Fred_ovr
	LED3_new
	LED3_ovr
	PulseWidth_Override_new
	PulseWidth_Override_ovr
	Temperature_new
	Temperature_ovr

Testing

172

Note a significant amount of error? For a temperature range of 0 to 100 degrees, we only used 7 bits. One bit is equal to 0.79 degrees.

With signals like the Pulsewidth and temperature, we are representing a continuous signal by a binary code with a finite number of bits. You will notice that the value transmitted over the CAN bus is an approximation that results in round off error.

Sheet1	Sheet2
Cooling_Fan	SW1
FRED	SW2
LED3	SW3
Pulsewidth	SW4
Temperature	Potentiometer
Potentiometer_Override_new	SharkTooth
Potentiometer_Override_ovr	Sinewave
	Time
	Cooling_Fan_new
	Cooling_Fan_ovr
	Fred_new
	Fred_ovr
	LED3_new
	LED3_ovr
	PulseWidth_Override_new
	PulseWidth_Override_ovr
	Temperature_new
	Temperature_ovr



Testing

173

- Using the overrides, verify that all signals being sent from one ECU are received accurately by the other ECU.



MotoTune – Displaying Charts

174

- To verify the waveforms that are being sent over CAN are working, we will plot the signals with MotoTune.
- We will first plot signal SW1 on ECU2.
- Right-click on the SW1 value cell and select **Properties**.

DisplayWindow4			
	A	B	
1	SW1	-0.99	
2	SW2	-2.27	
3	SW3	4.99	
4		2.93	

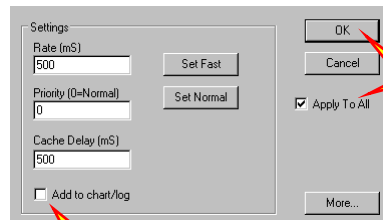
Right-click here and select Properties from the menu.



MotoTune – Displaying Charts

175

- Fill in the dialog box as shown:



We will apply the changes to all signals in the display.

Click the OK button when done.

This option not selected combined with the **Apply To All** option will result in all signals being removed from the chart.

MotoTune – Displaying Charts

176

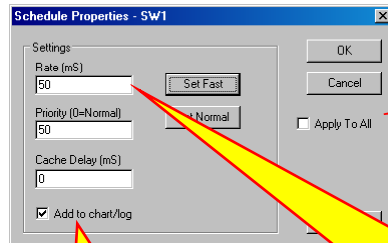
- After clicking the OK button, the selections to not plot anything will take effect.
- Right-click again on the SW1 value cell and select **Properties**.
- This time:
 - Click the **Set Fast** button.
 - Do not select the **Apply to All** button.
 - Select the **Add to Chart/Log** button.



MotoTune – Displaying Charts

177

- Fill in the dialog box as shown:



Option not selected.

The value of the signal will be updated every 50 ms.

This signal will be displayed on the chart.



MotoTune – Displaying Charts

178

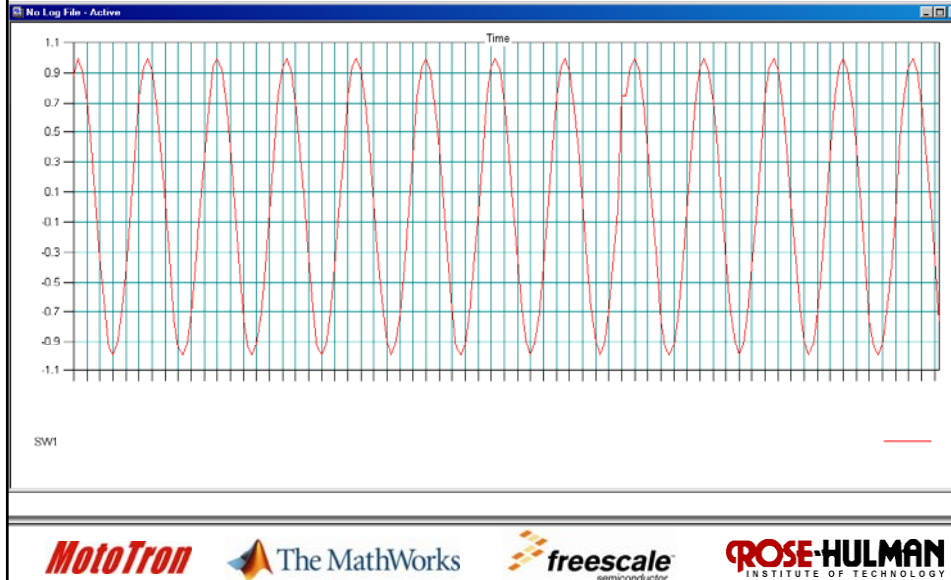
- Click the **OK** button to accept the changes.
- With our settings, only signal SW1 will be displayed on our chart.
- To display the chart, select **Chart** and then **Open Chart** from the MotoTune menus.
- You should see the following chart:





MotoTune – Displaying Charts

179



MotoTune – Displaying Charts

180

- To add another signal to the chart:
 - Right-click on the value cell and select **Properties**.
 - In the dialog box that appears:
 - Click the **Set Fast** button.
 - Select the **Add to chart/log** option.
 - Do not select the **Apply To All** option.
 - Click the **OK** button.
 - Close the chart that is presently open.
 - Open a new chart by selecting **Chart** and then **Open Chart** from the MotoTune menus.

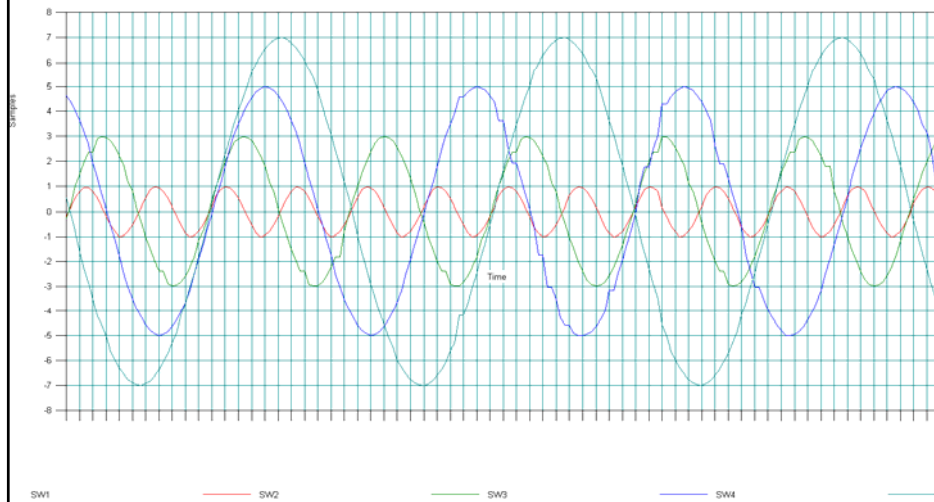




MotoTune – Displaying Charts

181

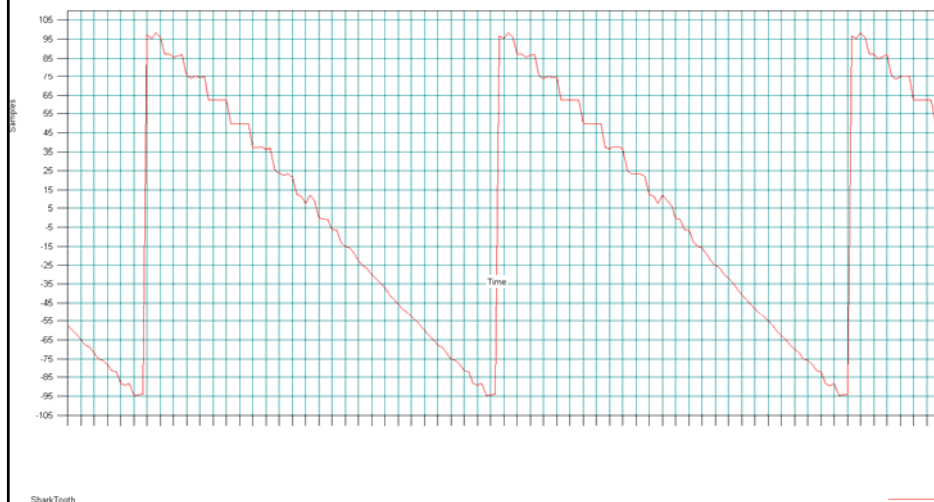
- Display all 4 Sine waves on the same



MotoTune – Displaying Charts

182

- Display the SharkTooth waveform.





CAN_CAN Project

183

- You should now be able to control the motors and LEDs connected from one ECU by CAN messages sent from the other ECU.
- We are done. Whew...
- Any Questions?



Lecture 18 Demo ECU1

184

- ECU1: CAN Communication
 - Fred – ECU 1 Turns on LED when it receives value of -3127. Value sent with MotoHawk Probe. _____
 - LED3 – Turn on and off an LED connected to ECU1. One bit signal sent by ECU2. _____
 - Pulsewidth – Receive signal of 0 to 100 from ECU2. ECU1 Receives the signal and emits a PWM signal that controls the motor speed. _____
 - Temperature – ECU1 Receive CAN signal from ECU2 and display value with probe. _____
 - Cooling FAN – Receive 10bit Signal from ECU2. Display with MotoHawk Probe. _____



Lecture 18 Demo ECU2

185

- ECU2: CAN Communication
 - Receive four sine waves from ECU1. Display values on a chart . _____
 - Potentiometer– Receive potentiometer signal from ECU1. Scale signal and emit a PWM signal that controls the motor speed. _____
 - ECU2 receives signals for time, a ramp (shark tooth), and a sine wave over the CAN bus. The values will be displayed with probes and a chart. _____



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>



Advanced Model-Based Systems Design

Lecture 19: Hardware In The loop Simulations (HIL)



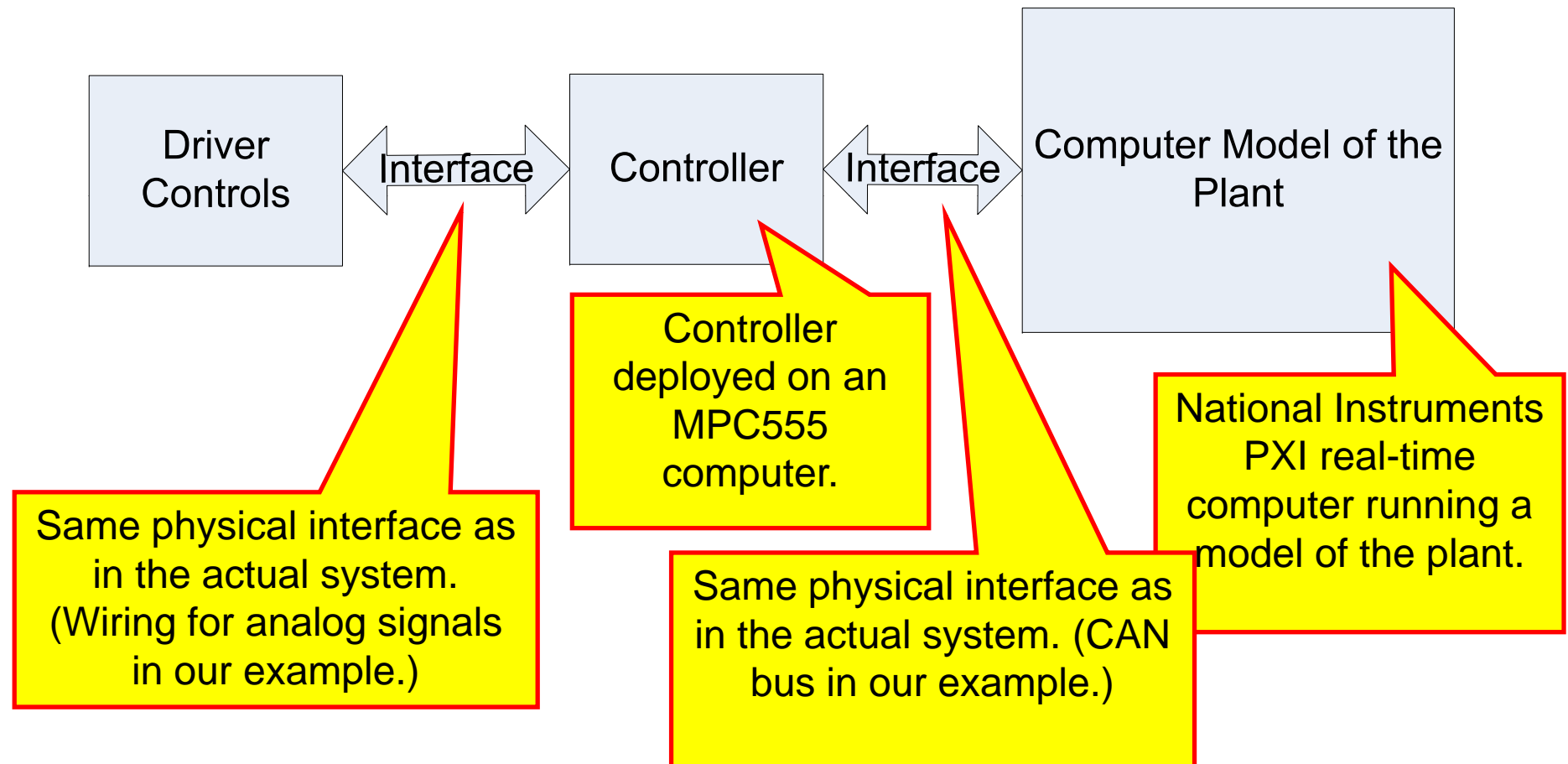
- Up to this point we have:
 - Learned several levels of simulations: PC and Real-Time.
 - Learned several software packages: MATLAB, Simulink, MotoHawk, LabVIEW
 - Used many platforms: PC, LabVIEW RT, PXI
 - Used several different hardware targets: PXI, MPC555.

- It is now time to put it all together and perform Hardware-in-the-loop (HIL) simulations.
- We will start with the full vehicle model developed in Lecture 14 exercise 6 and split the model so that:
 - The controller runs in the MPC555 target.
 - The plant runs on the PXI target.
- The two targets will be connected with a CAN bus, the same harness that will be used in the final product.
- The Controller will be connected to driver controls through a wiring harness.
- The models will run in real time.

- This is a test of the controller:
 - Hardware - It is running on the target we will use in the final implementation.
 - Speed - It is running in real time.
 - Wiring Interface - It is connected to the plant and driver controls using the same interface that will be used in the final implementation.
 - This tests both the wiring as well as the effect of network latency as control messages are sent through the CAN bus.
- If the controller works when hooked to our virtual plant, we have confidence that it will work when we hook it to the physical plant (the real vehicle).

- We will start with Lecture14_Exercise6.mdl and split it into two models, the plant and the controller. (The model has been renamed Lecture19_Model0.mdl and passed out.)
- The plant will:
 - Run on the PXI Target.
 - Use LabVIEW and SIT to create a shell to interface between the model and the controller.
 - The inputs and outputs will be CAN signals.
- The controller will
 - Run on an MPC555 target.
 - Use MotoHawk to interface between the model and physical world.
 - The inputs and outputs will be analog voltages and CAN signals.

- We will be using the test platform below:



HIL Simulations

Part 1: Implementing the Controller on the MPC5554 Target



Motor Controller Deployment



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- From lectures 16 through 18, we now know how to use the hardware resources of the MPC555 well enough to use it as the target for the controller of our motor-generator system.
- We will use the control method we proved, tested, and verified in the SIL and real-time portions of the class.
- First, we will create a shell that accesses the hardware resources of our target (MPC555).

Motor Controller Deployment



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- The controller will have the following driver inputs that come from the driver board:
 - Brake Pedal – 0 to 5 V analog input.
 - Accelerator Pedal – 0 to 5 V analog input.
 - Park push-button - 0 to 5 V analog input.
 - Forward push-button - 0 to 5 V analog input.
 - Reverse push-button - 0 to 5 V analog input.
- The driver board has LEDs that indicate Park, Forward, Reverse, Error, and Vehicle Ready:

Driver Board Schematic



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>



Motor Controller Deployment



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- You will need to use MotoHawk analog input channels to read the analog inputs and then scale the signal appropriately for the signals required by the controller.
- You will use MotoHawk high current digital outputs to drive the LEDs. Note that a low output will illuminate the LEDs on the driver board.
- All other inputs and outputs for the controller will use the CAN bus.



Hardware Shell



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- We will now create a top-level “shell” for our controller that:
 - Initializes the MPC555 and MotoTune
 - Reads and scales the inputs and provides the outputs
 - Passes the information to a subsystem that contains the controller.



Hardware Shell



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The basic idea is that our interface to the hardware will not change that much.
- Given the same interface, we can make significant changes to our control method.
- All of these changes will be implemented in the controller subsystem.
- The hardware shell will remain relatively unchanged. (Occasionally, a new control method will require new inputs or outputs. In this case, we will need to modify the hardware shell.)


Hardware Model — Top Level

Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

Specify CAN 1 and a baud rate of 500 K

Controller Controller.mdl

Copyright 2005, All Rights Reserved

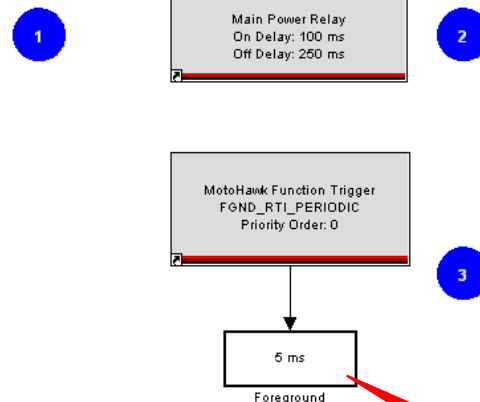


Target: ECU555-80 (DEV)
 Floating Point: single (32 bits)
 Stacks - FGND: 3072 BGND: 2048
 IDLE: 1024 IRQ: 1536
 Heap Size: 4096
 DLL Filename: Controlle_018
 SRZ Filename: Controller1_018



Total FLASH:	177060
Total EEPROM:	190
Total RAM:	17008
App FLASH:	18675
App EEPROM:	16
App RAM:	562

SCIL4 MotoHawk (RTW)
 Code Coverage Test Bit System
 ALWAYS GENERATE CODE COVERAGE CODE



MotoHawk CAN Definition

Name: CAN_1
 Bus: CAN1
 Bit Timing: 500 kbaud
 TX Queue: 16 messages
 RX Queue: 16 messages
 MotoTune Protocol Enabled
 City ID: 0x0C (PCM-2)

MotoHawk (RTW) Tool Chain Definition

Name: MyToolChain
 Tool Chain: Custom
 Build Template: GreenHills 3.6
 Install Directory: C:\GHS\ppc424\

Specify the GHS compiler.

- 1 The existence of a Target Definition block somewhere in the model sets it up as a MotoHawk project, capable of being built to a MotoTron target module.
- 2 The Main Power Relay block monitors the module power and controls the Main Power Relay. It also performs startup and shutdown procedures necessary for EEPROM storage. Most designs should contain this block.
- 3 All actions must be performed inside of a function-call subsystem, triggered by a MotoHawk entry-point trigger block, in order to be included in the generated code.

Controller and shell inside here. Run once every 5 ms.

MotoTron

 **The MathWorks**

 **freescalse**
semiconductor

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Hardware Shell - Foreground Subsystem

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

All input signals bussed together here and passed to the controller.

Analog inputs and outputs.

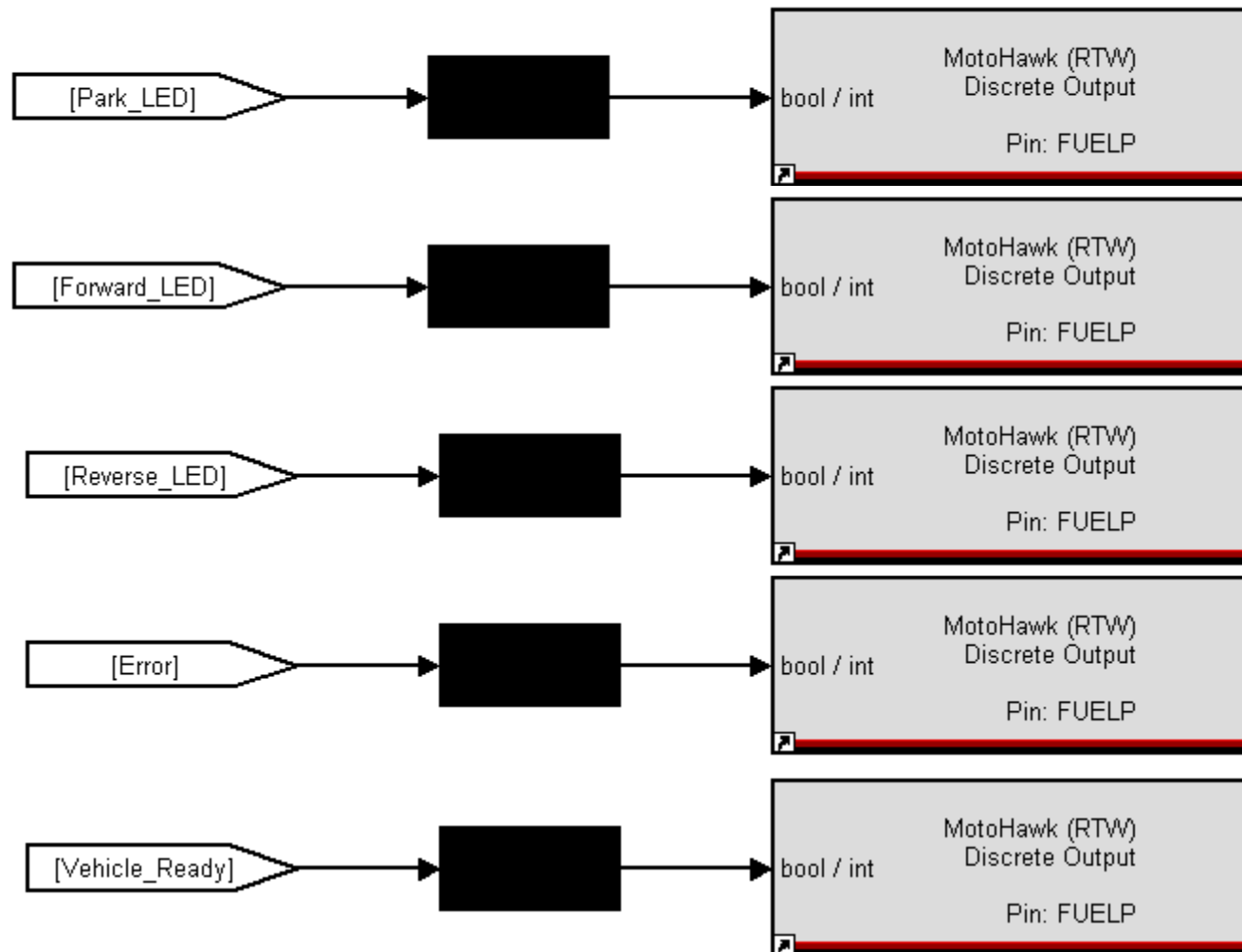
Controller outputs extracted with a bus selector and passed to analog and CAN output blocks.

CAN Outputs

CAN Inputs

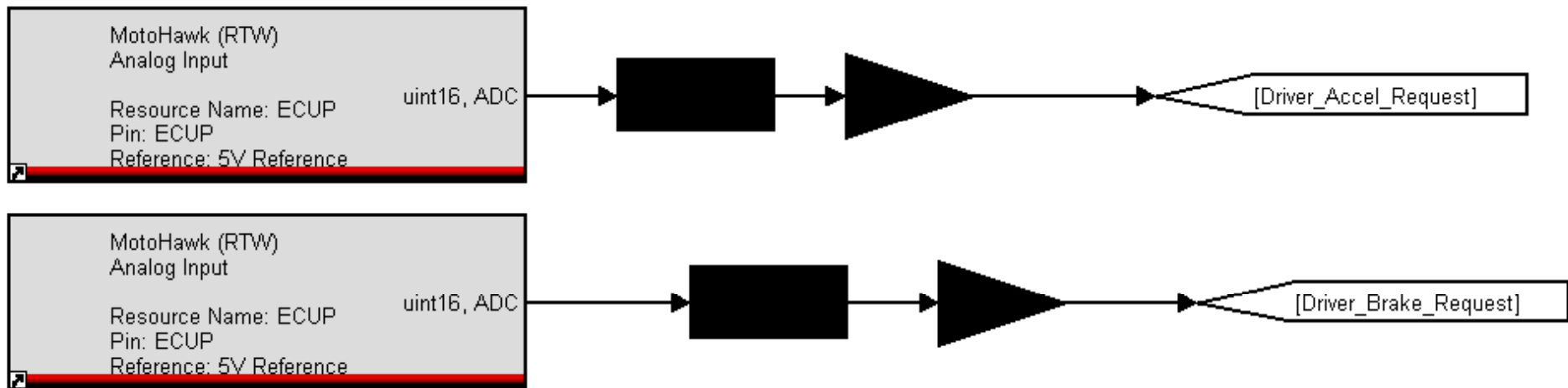
Hardware Shell – LED Outputs

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Hardware Shell — LED Outputs

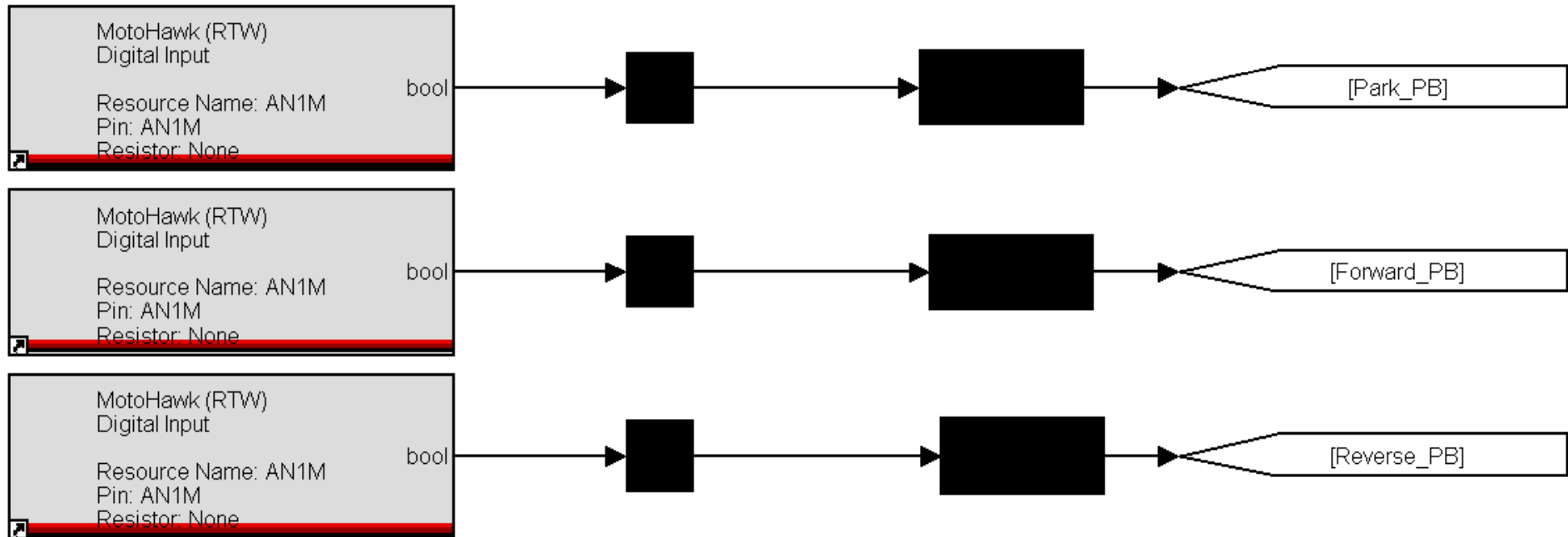
Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Hardware Shell — Push-button Inputs



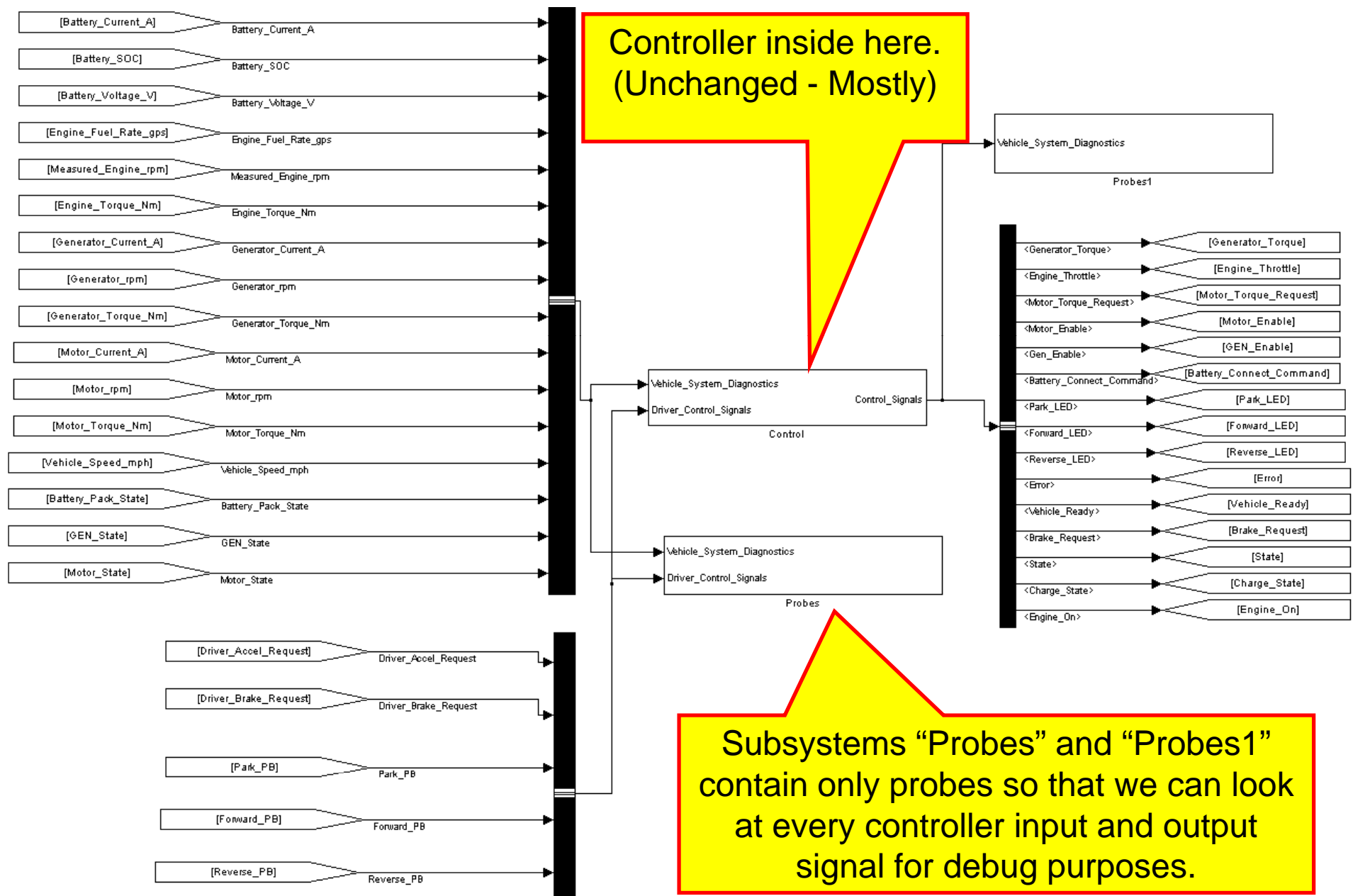
Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



Hardware Shell — Bus interface to Controller



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



HIL Simulations

Part 2: Implementing the Plant on the National Instruments PXI Target



Plant Model

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Since we already implemented the entire model in real-time to run on a PXI target in the previous lecture, we will reuse some of the work we did in that model.
- This Model was resaved as Lecture19_Model0.
- Open the model and resave it as Vehicle_Plant.mdl.

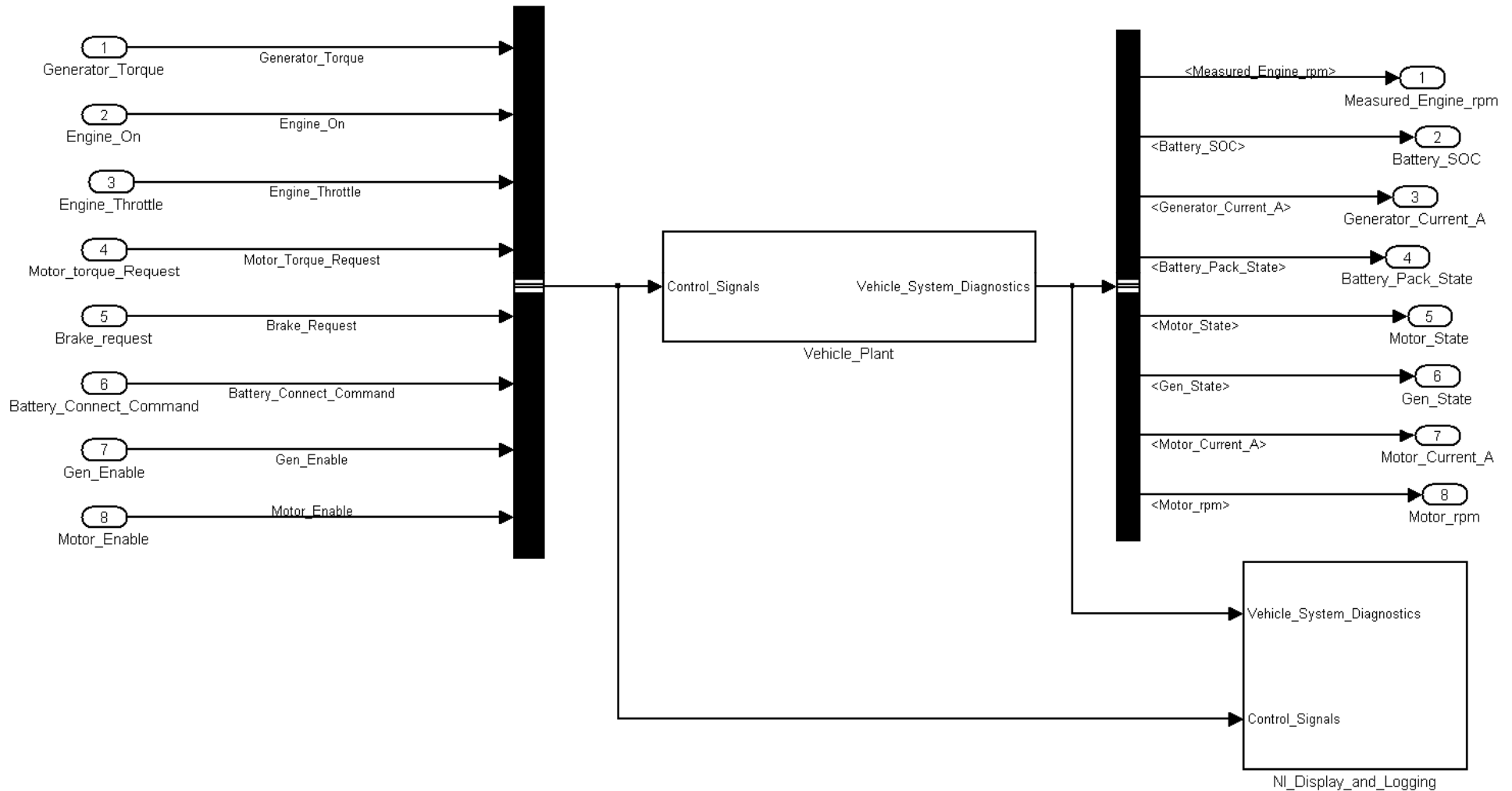
Plant Model

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Delete the Controller subsystem and NI_Driver subsystem.
- Leave the NI_Display_and_Logging subsystem in the model as we will display most of the same signals in the LabVIEW front panel as we did in lecture 14.
- The inputs that came from the controller are now connected to In ports and the plant outputs that went to the controller are now connected to Out ports.
 - We will associate these with CAN signal inputs and outputs using the Simulation interface toolkit.

Plant Model

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



NI_Display_and_Logging



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- The NI_Display_and_Logging subsysyem was slightly modified because we needed to remove the driver signals.

LabVIEW Front Panel

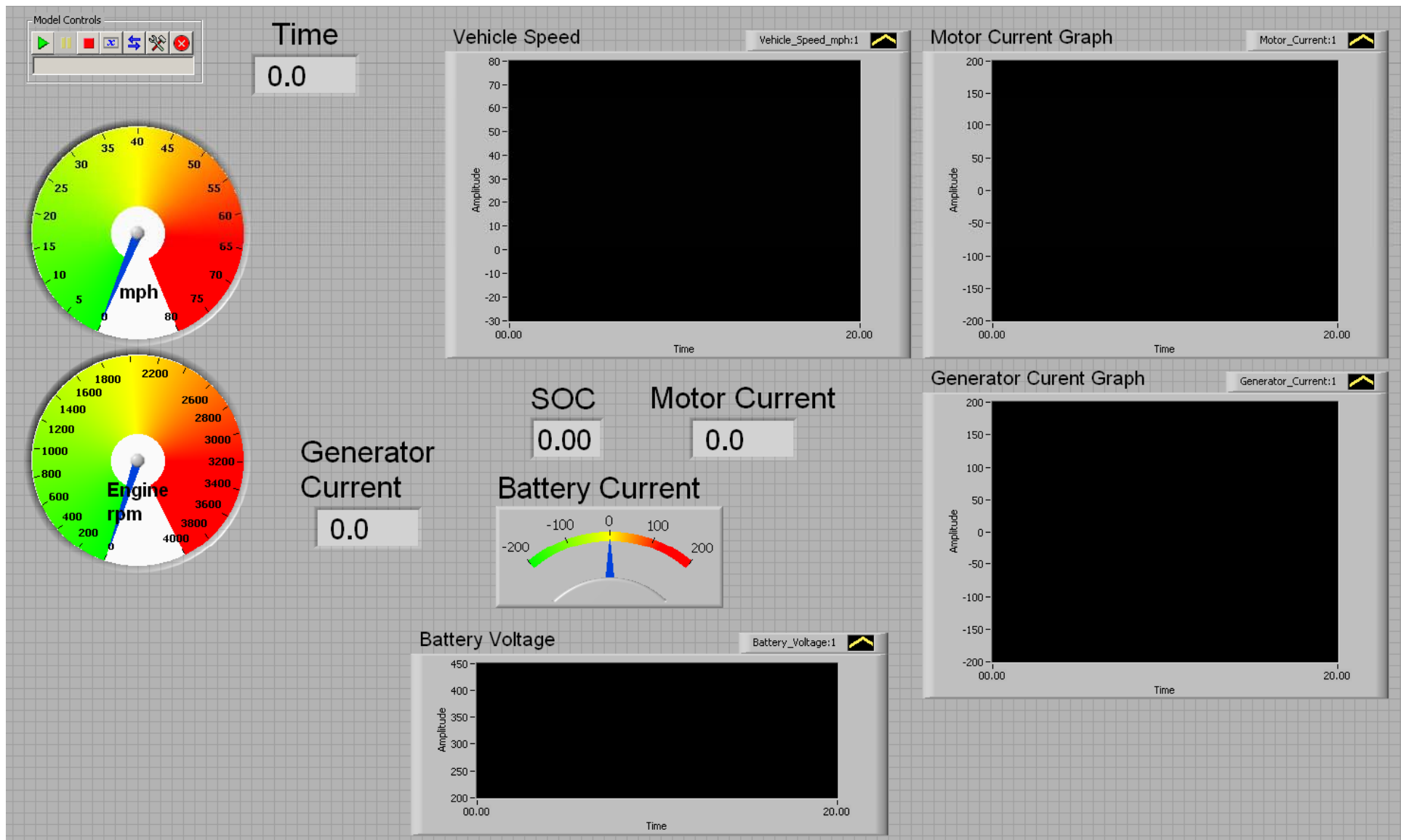


Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- We will use the front panel we created in Lecture 14 in this example.
- We will remove the driver controls from display but reuse everything else from the example.
- The VI from lecture 14 has been provided for you and renamed as Lecture19_Model0.vi.
- Modify the front panel as shown next:

LabVIEW Front Panel

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>



SIT Connection Manager



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- You will need to use the SIT connection manager to:
 - Change the DLL to use the one for the plant only.
 - Connect the front panel displays to the appropriate signals in the DLL.
 - Associate CAN inputs and outputs with the In and Out ports that we placed in the plant model.

SIT Connection Manager - CAN



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

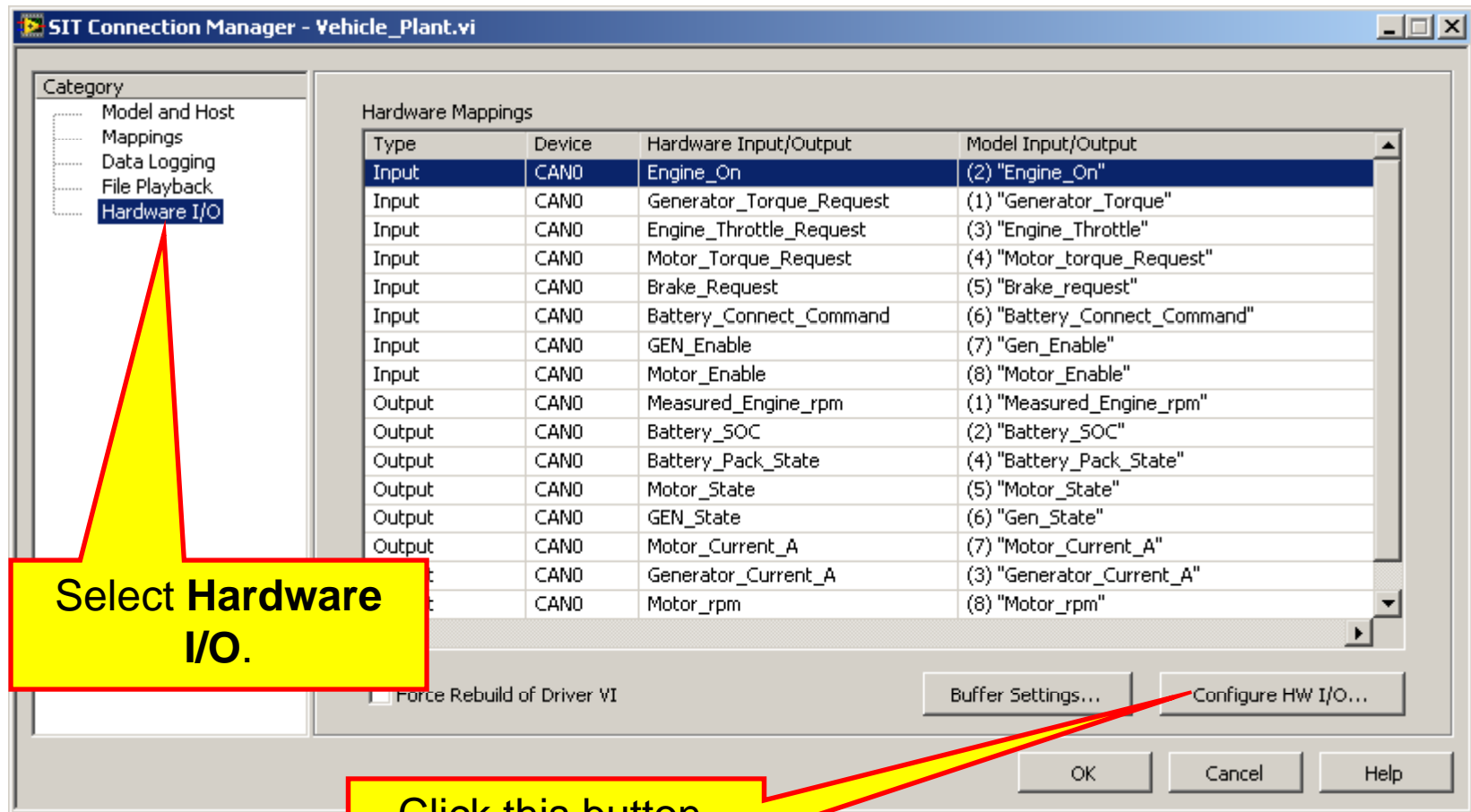
- The CAN signals are specified in a CANdb file. This file has been provided for you and is named AMBD_HIL1.dbc
- All of the CAN signals that are needed are contained in this file.
 - (Not all of the m-files needed to define the CAN signals in MotoHawk have been provided.)
- CAN signals are associated with In and Out ports by selecting the **Hardware I/O** category in the SIT Connection Manager:

SIT Connection Manager - CAN

30



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

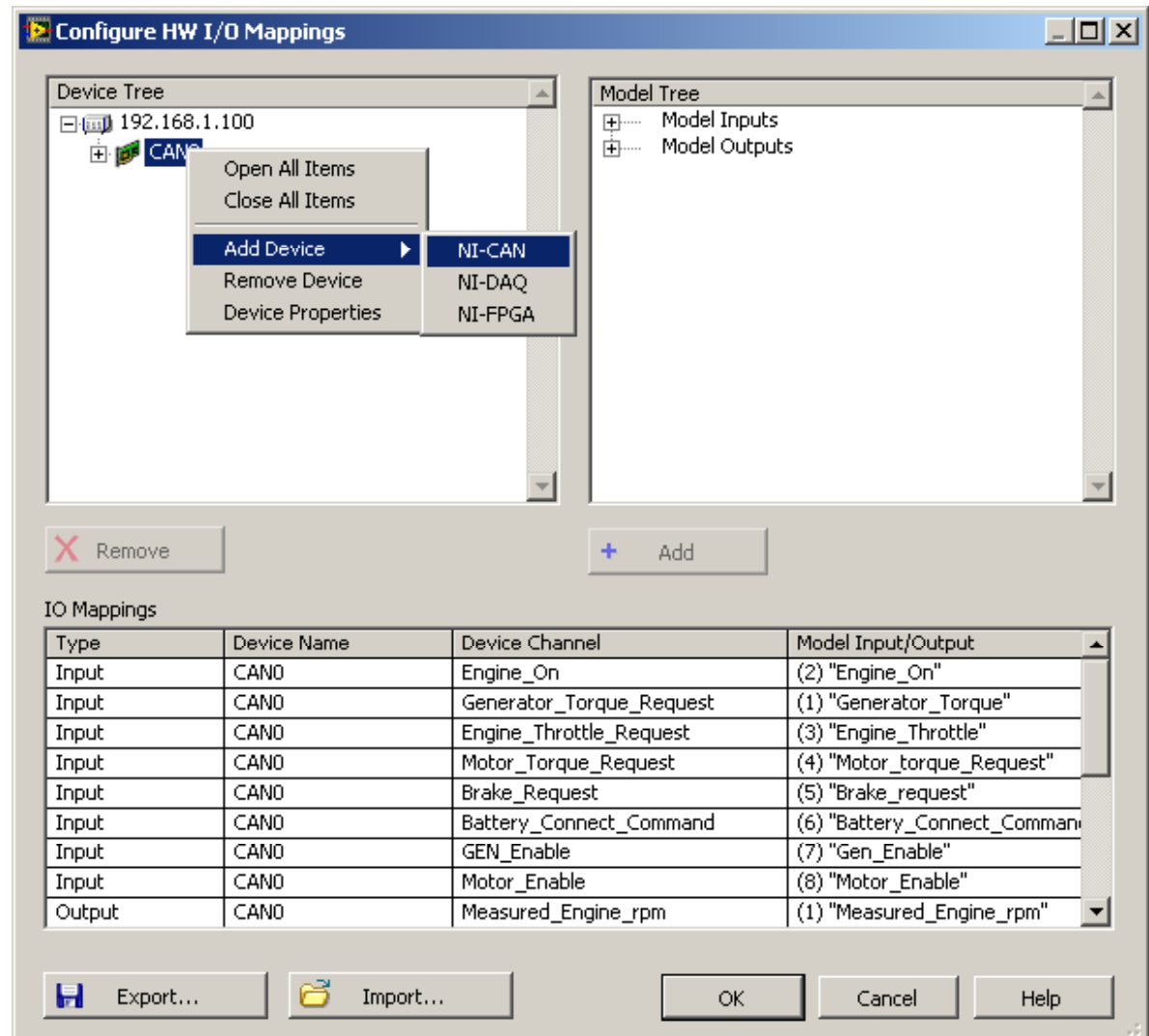


SIT Connection Manager

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- You will need to right click on your target, select Add Device and then NI-CAN.

- The ensuing screens will allow you to identify the CAN hardware on your target and specify a CANdb “.dbc” for the project.



SIT Connection Manager



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Once you identify the CAN channels and specify the CANdb file, all of the signals in the CANdb file will be displayed.
- You can then associate model inputs and outputs with (In and Out ports) with CAN signals.

CAN Baud Rate



Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- You will need to use the National Instruments Measurement and Automation Explorer to set the baud rate of the specific CAN channels.
- (Show how to do this.)

Lecture 19 Exercise 1

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- Demonstrate a working HIL system with the controller logic unchanged. You will need to:
 - Use MotoTune to debug and display many of the controller input and output signals.
 - Define some m-files for the missing CAN signals.
 - Wire the driver control board to your ECU.
 - Debug a lot of wrong connections and signal associations.
- You should be able to drive you vehicle with the driver controls.

Demo_____

Lecture 19 Exercise 2



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- You will notice that the charging current oscillates wildly when the vehicle starts charging.
- This is because the feedback signals and generator torque engine throttle command signals for the proportional feedback loop come over the CAN bus.
- CAN messages are periodic and are sent at a slower rate than needed to make the loop stable.

Lecture 19 Exercise 2a



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- Investigate an earlier model that we ran in Matlab.
- Add delay to the feedback loops to model the delay introduced by the periodicity of the CAN messages.
- Show that the control loops for maintaining constant engine rpm and constant oscillate when we add 20 ms delays to the incoming and outgoing signals.
- This should prove the theory that the reason the system is unstable is the added delay due to latency in the CAN network.

Demo_____

Lecture 19 Exercise 2b



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by/3.0/>

- In the controller model running on the MotoTron ECU, add calibration blocks that allow us to change the feedback gains of the engine speed loop and the generator current loop.
- Determine the feedback gain of both loops necessary to obtain constant and stable charging currents.

Demo_____

Lecture 19 Exercise 3

Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by/3.0/>

- In Exercise 2b, we find that the proportional gains have to be reduce so much that the engine rpm and generator currents are quite far away from the desired values.
 - The loops are stable but we have a large error because the gains are so small.
- To fix this problem, add integrators to both loops and add calibration blocks so that we can tune both the proportional and integral gains of each loop separately.
- Show that the error goes to zero and that the system is stable. (A little bit of overshoot is acceptable.)
- You will need to build your own digital integrator and make sure that it saturates (or has limits on how big the value can grow).

Demo_____