



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



# Introduction to Model-Based System Design

## Lecture 1



The MathWorks



## MBD

2

- Model-Based-System Design re-engineers the traditional development process from one which is paper-based to one that uses an executable model that is the repository for all information about the concept, design, and implementation. The model is used throughout the four stages of development: Research, Design, Implementation, and Verification & Validation. At each stage of development the model is updated and elaborated ensuring continuity and traceability throughout the evolution of the design.



The MathWorks





3

## What is Model-Based-System Design?

- In its most basic form, Model-Based-System Design is the use of models to describe the specifications, operation, and performance of a component or a system of components.
- Instead of listing specification in a text document, a model is used that implements the specifications, operation, and performance of components.



The MathWorks



freescale  
semiconductor



4

## What is Model-Based-System Design?

- Models can be shared with other engineers:
  - Engineers do not have to generate their own models from text specifications.
  - Same model can be used by several engineers at several different levels in the design process.
  - Component models can be used in larger systems.
  - Models supplied by manufacturers accurately reflect the performance of their components.



The MathWorks



freescale  
semiconductor





5

## Old Method – Text Documents

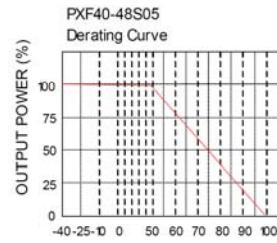
- Text documents are used to describe the performance of a component.
- Example: 12V to 5 V DC-DC Converter
  - 12 V power is readily available in vehicle environments.
  - 5V supplies are needed to power the Navigation/Infotainment system.
  - Power conversion is less than 100% efficient.



6

## Old Method – Text Documents

- Electrical Specifications
  - Input Voltage: 9V to 18 V
  - Output Voltage: 5V
  - Output Ripple: 50 mV
  - Output Current: 0A to 8A
  - Efficiency:  $\geq 86\%$
  - Derating:



The MathWorks





7

## Old Method – Text Documents

- Thermal Specifications
  - Case to Ambient Thermal Resistance:  
1°C/Watt
  - Case to Sink Thermal Resistance  
0.03°C/Watt
  - Heat Capacity: 300 J/°C
  - Heat Sink Thermal Resistance:
    - 0.91 °C/Watt (Natural Convection)
    - 0.150 °C/Watt (50 CFM Forced Air Convection)
    - 0.105 °C/Watt (100 CFM Forced Air Convection)
    - 0.068 °C/Watt (200 CFM Forced Air Convection)



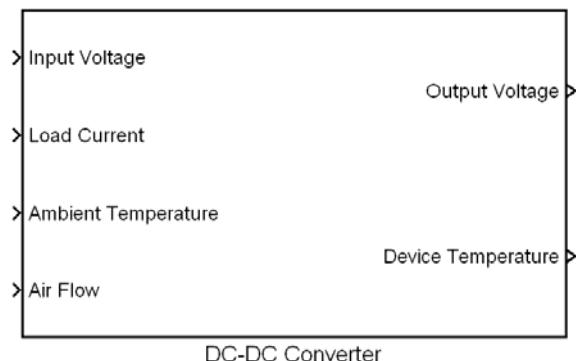
The MathWorks



8

## Model-Based-System Design

- Create a model that implements all of these specifications:



The MathWorks





9

## Model-Based-System Design

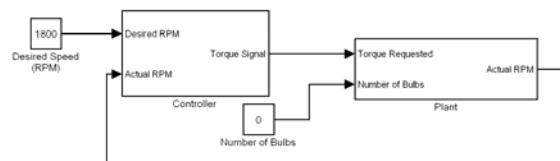
- This model can:
  - Contain much more detail than written specifications.
  - Contain behavioral characteristics such as:
    - Voltage cutback during over current event.
    - Thermal cutback and power derating.
  - Can be shared among engineers working in different areas and at different levels of the design process.



10

## What is Model-Based-System Design?

- Model-Based-System Design allows us to use models throughout the entire design process:
  - Simulation using Simulink (SIL)
    - Not real-time
    - Develop detailed plant model.
    - Develop a system controller.





11

## What is Model-Based-System Design?

- Real-Time Simulations
  - Determine how the systems responds in real-time.
  - Good for human-system interaction.
  - Additional debugging.



12

## What is Model-Based-System Design?

- Targeting
  - Implement the controller developed and debugged in SIL and Real-Time simulations on a hardware target (embedded controller).
    - Most logic errors have been removed.
    - Errors occur if model is inaccurate.
  - Use automatic Code Generation (Skip the computer engineering guy!)
    - Still need to know something about digital systems and microcontrollers.
    - Eliminates the errors introduced when involving an engineer to manually program the controller.
    - Allows the engineers with the physical understanding of the system to do the logic development and “programming.”

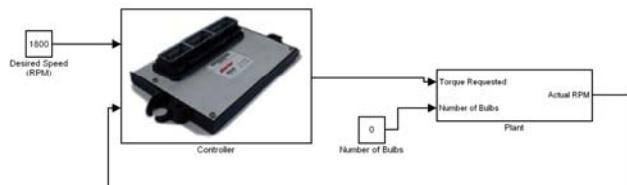




13

## What is Model-Based-System Design?

- Hardware in the Loop (HIL) Simulations
  - Real-time.
  - Controller implemented on our Target.
  - Plant implemented on a real-time system.



**MotoTron**

The MathWorks

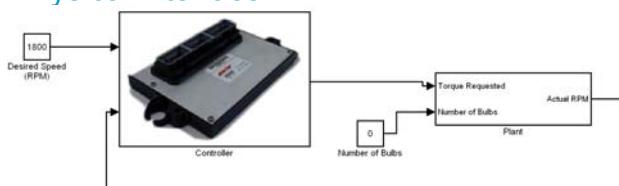
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

14

## What is Model-Based-System Design?

- Hardware in the Loop (HIL) Simulations
  - Same physical interface as in actual system.
  - Smoke-free testing of the controller.
  - Tests:
    - Controller logic.
    - Controller speed and processing power.
    - Physical interface.



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



15

## What is Model-Based-System Design?

- Controller deployment
  - Given accurate models and a consistent interface, we can just plug in the controller to our plant.
  - It should work perfectly the first time. (Not)
  - It should work reasonable well, but we will notice that the plant model may have inaccuracies.
  - Typically we will need to modify the plant and controller to account for the differences.



16

## What is Model-Based-System Design?

- Verification and Validation
  - SIL, HIL, PIL are forms of V&V
  - Design a set of repeatable tests.
    - Run tests, measure results.
    - Determine pass/fail status.
    - Collect Data
    - Improve models.
  - Repeat the same tests each time we make a change to the controller or plant.





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

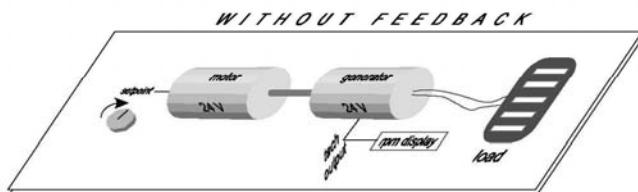
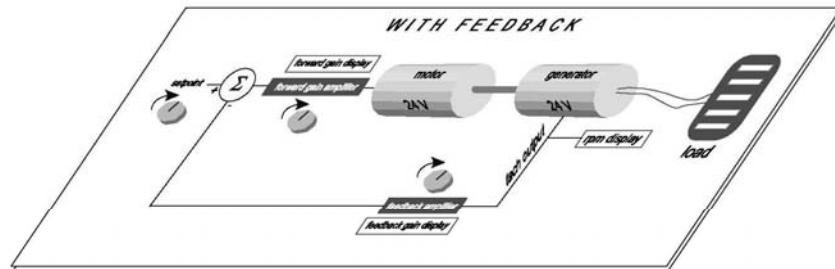


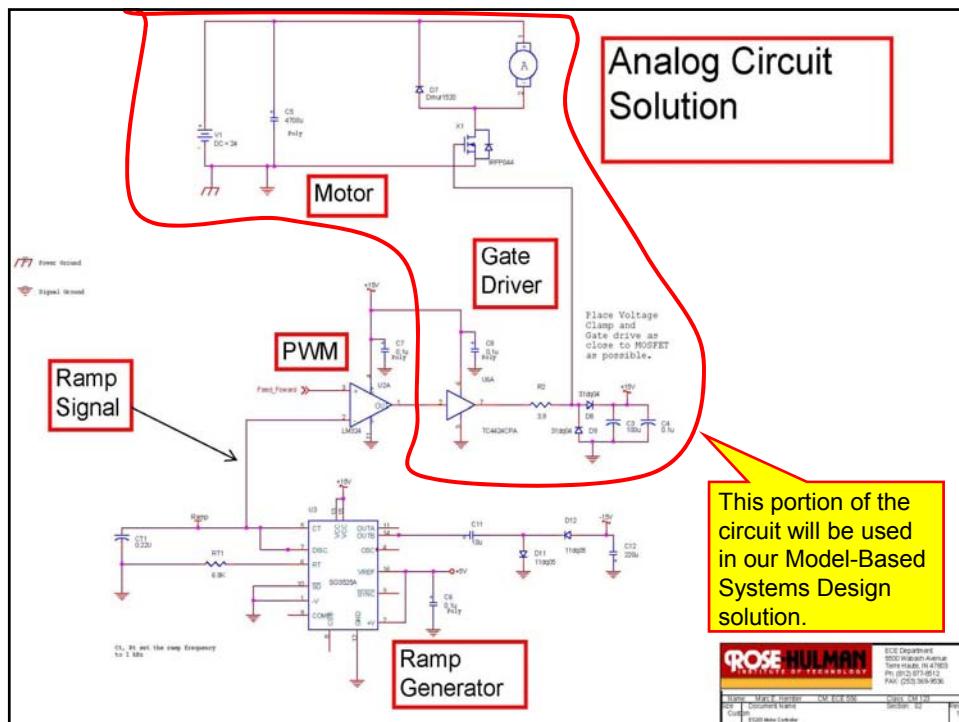
## Introduction to Model-Based Systems Design

### Model-Based Design of a Motor-Generator System



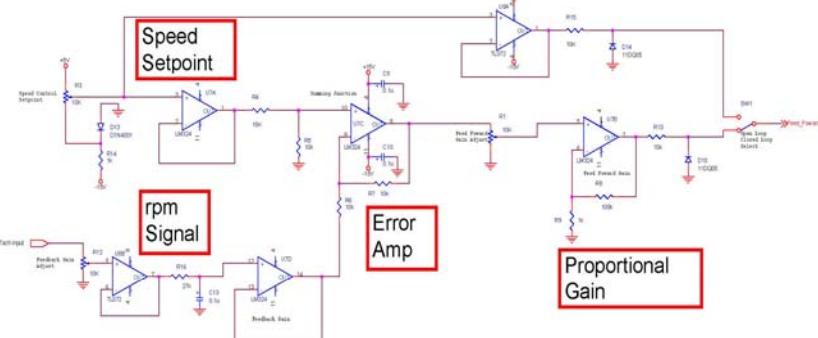
18





## Proportional Feedback

20

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

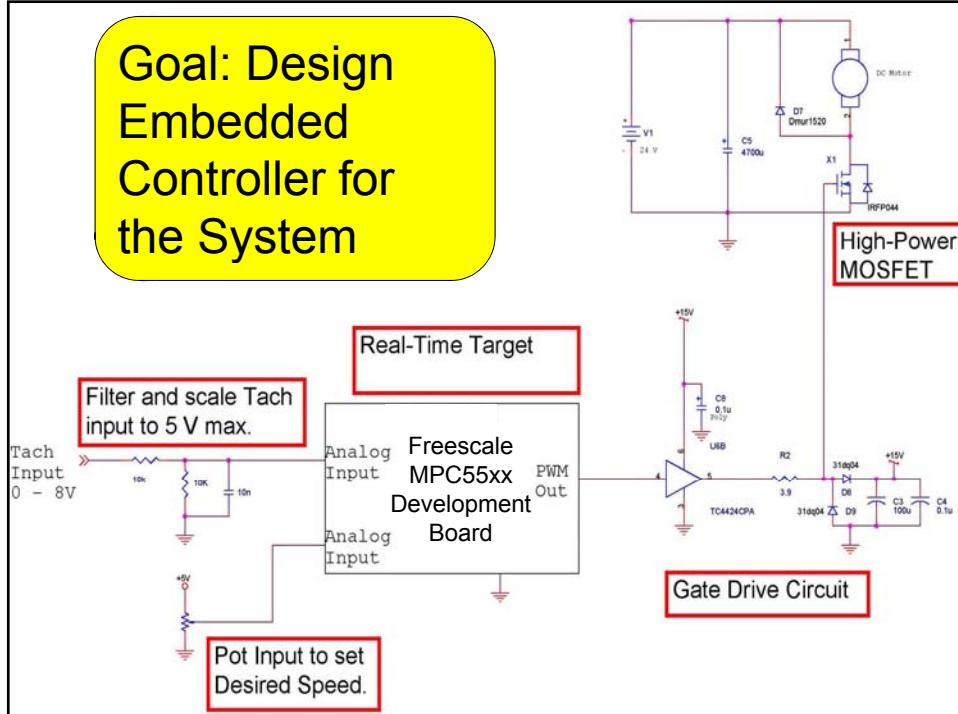


21

## Model-Based Design Solution

- Create Plant and Controller Models
  - Motor and Generator Models
  - P and PI Controllers
- Simulate with Simulink
- Real-Time Simulations with xPC
- Implement Controller on Freescale MPC55xx Real-Time Target
- Test
- Improve Model and Controller
- Repeat

**Goal: Design Embedded Controller for the System**





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

## Simulation Fallacy

- Simple system models are worthless
- You must start with:
  - All component nonlinearities and limitations
  - Every function the model will perform



## Simulation Fallacy

- The first time we run a simulation of this type the following *will* occur:
  - It does not work
    - Users have no idea why
  - It has convergence errors, logic errors, or improper component behavior
    - Users have no idea why
  - It runs but the output is obviously wrong
    - Users have no idea why





## Modeling Building Philosophy

- Start with simple component models
- Understand simple component operation
- Develop a simple controller
- Anticipate expected system output
- Verify that output matches expectation



## Modeling Building Philosophy

- Add a single function to the model and
  - Understand the effect on the
    - Component
    - System
    - Controller
  - Anticipate the expected system output
  - Verify that output matches expectation
- Repeat as needed...





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

27

## Course Overview

- Model-Based Design for a small system
- Simulink Simulations
- Real-time simulations with xPC
- Implement controller on MPC5553 or MPC5554 target
- Hardware In The Loop Real-Time Simulations
- Test controller on real system
- Model Verification and Validation
- Design of Experiments to Collect Experimental Data on Motor and Generator
- Model Refinement and Re-Verification
- Further Exploration of Alternate Control Methods as Time Permits





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



# Introduction to Model-Based Systems Design

## Lecture 2 : Model Building with Simulink



The MathWorks



freescale<sup>®</sup>  
semiconductor



## Presentation Outline

2

- Intro to Simulink via Unit Converter
- Model-Based System Design
- Model Hierarchy
- Plant Model – Motor
- Plant Model – Generator
- Plant Model – Shaft Encoder



The MathWorks



freescale<sup>®</sup>  
semiconductor





3

## Intro to SimXYZ

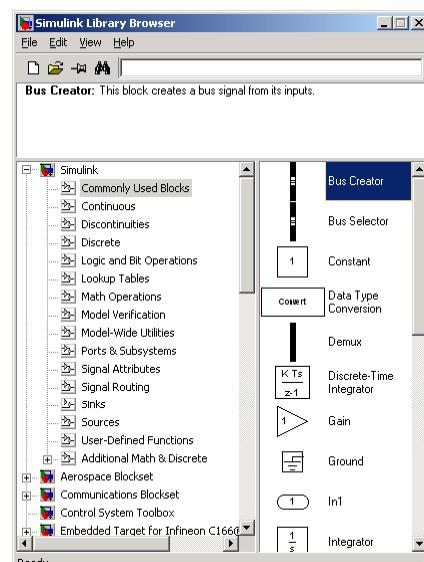
- **Simulink** is an extensive collection of libraries which enable you to *link* specific blocks to develop *simulations*
- Libraries include
  - Mathematical operations
  - Differential equations
  - Continuous/discrete functions
  - Scopes
  - Many more!



4

## Intro to SimXYZ

- Fire up Matlab and at the command line type **simulink**
- You should get a new window similar to that at right.
- Take two minutes to explore all the different libraries and associated blocks!



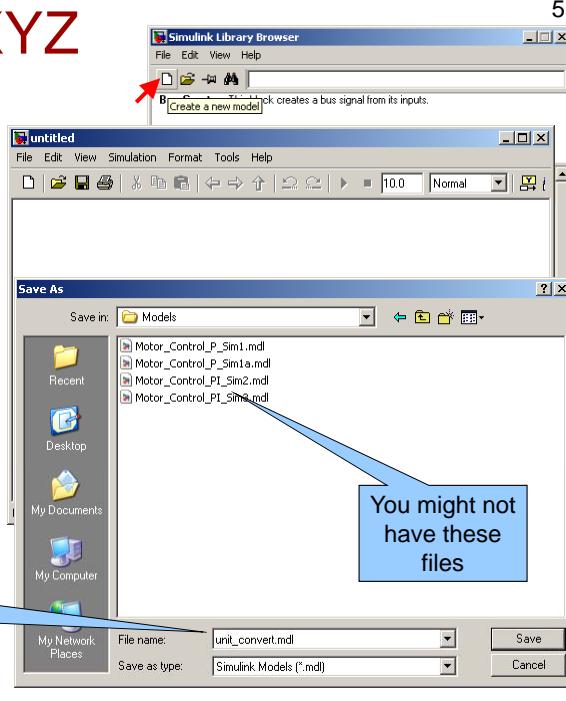


## Intro to SimXYZ

- Click on the Create New Model Icon
- You should get a new model window.
- Save it as unit\_convert.mdl

Model name specified here.

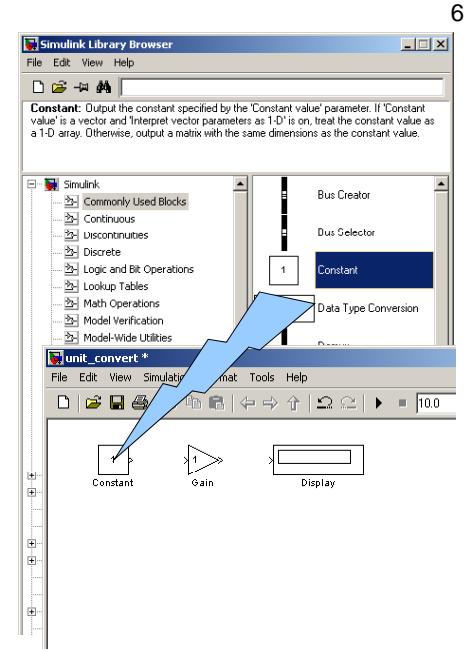
You might not have these files



## Intro to SimXYZ

### Commonly Used Blocks

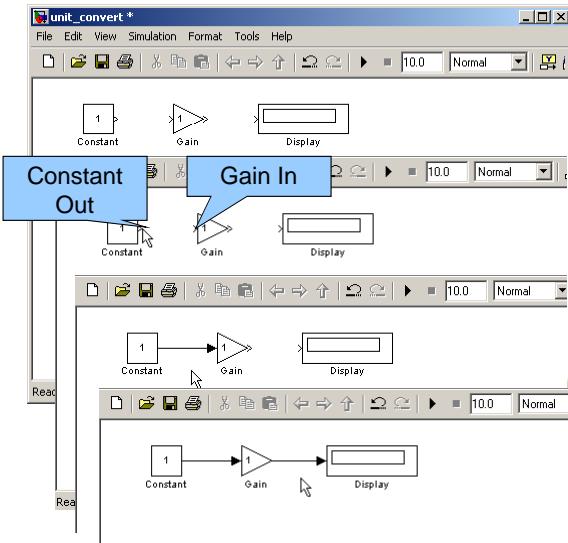
- Constant
- Gain
- Sinks
- Display
- Left click on the Constant and drag it into the model window
- Repeat for the Gain and the Display





## Intro to SimXYZ

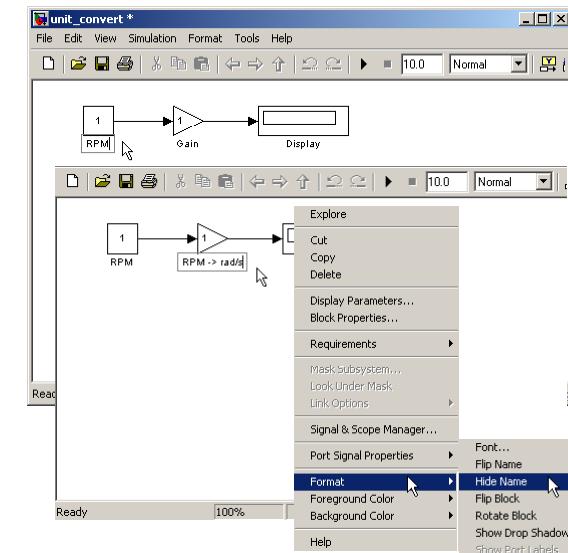
- Place mouse over the **Constant** output port ( $>$ ), left click, and drag to the **Gain** input port ( $>$ )
- A wire with an arrow will link the two blocks
- Link the **Gain** and **Display** blocks using the same method.



7

## Intro to SimXYZ

- Click on the text **Constant** and change it to *rpm*
- Change the text **Gain** to *rpm -> rad/s*
- Right click on the text **Display**, select **Format**, and **Hide Name**

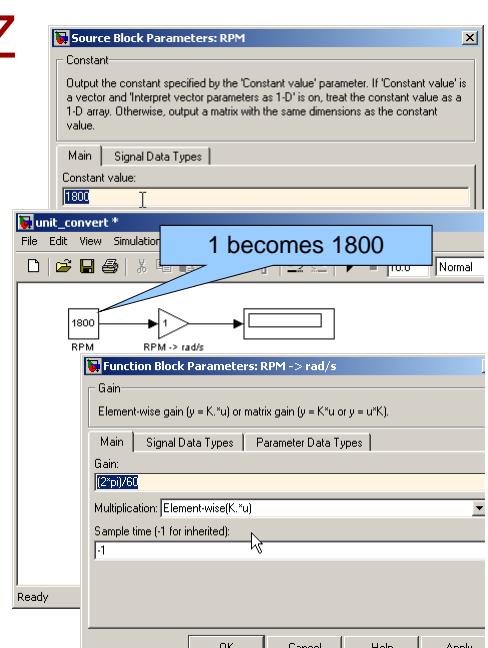


8



## Intro to SimXYZ

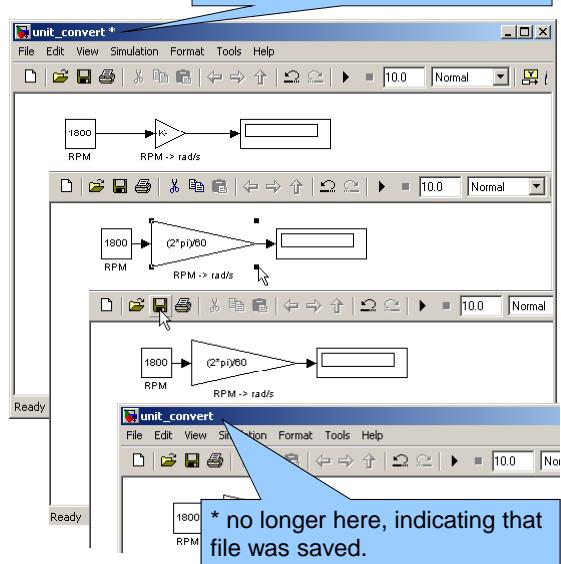
- Double click on the **Constant (rpm)** block and change the value from 1 to 1800.
- Click the **OK** button.
- Double click on the **Gain (rpm -> rad/s)** block and change the value from 1 to  $(2\pi)/60$ .
- Click the **OK** button.



9

## Intro to SimXYZ

- The value of the gain is now **K** → the block is not big enough to display the actual value.
- Left click on the block, then grab a handle and drag it to make the block larger.
- Save your file.
- Note that the asterisk on the filename goes away when it is saved.



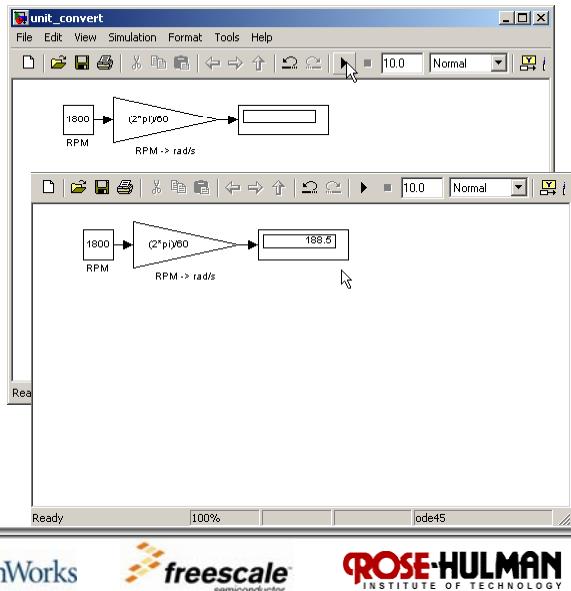
10



11

## Intro to SimXYZ

- Run the model by left clicking the **Run** button.
- A value of 1800 rpm is converted to 188.5 rad/s.
- You have mastered the basics of Simulink!

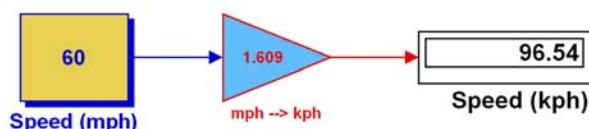
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

12

## Lecture 2 Exercise 1a

- Design a converter that converts a speed of 60 miles per hour to kilometers per hour. Make the following format changes to your blocks:
- All text should be displayed in bold.
- Constant Block Properties:
  - Block text changed to “Speed (mph)”
  - Font: Arial 16.
  - Text and block outline color is blue.
  - The block has a drop shadow.
  - Block fill color is yellow.
- Gain block properties
  - Block text changed to “mph → kph”
  - Font: Arial 12.
  - Text and block outline color is red.
  - Block fill color is light blue.
- Display block properties:
  - Block text changed to “Speed (kph)”
  - Font: Arial 18.
- Your solution should look as shown:
- 

Demo \_\_\_\_\_





13

## Lecture 2 Exercise 1b

- Use Simulink to calculate how far a vehicle moving at 60 mph would travel in 100 seconds. Provide an answer in both miles and in kilometers.
- Answers: 1.667 miles, 2.682 km.

Demo \_\_\_\_\_

14

## High-Level System

- Our plan is now to use Simulink to build a model of the plant and controller
  - Plant : Motor, Generator, Shaft Encoder
  - Controller: P or PI
  - Plant Input : Light bulb load
  - Controller Input : Generator speed
- We will build a very simple plant first.
- We are aiming for the following system:



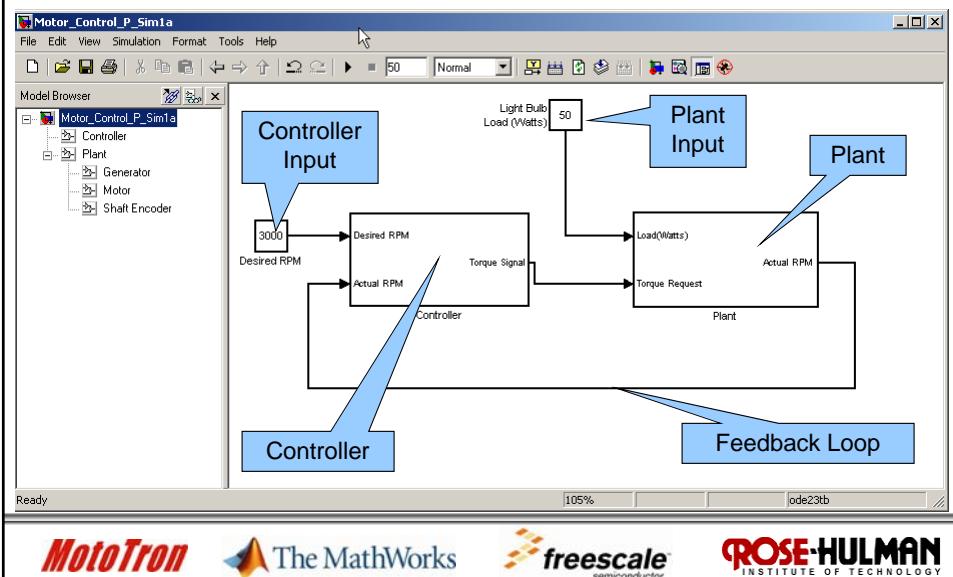
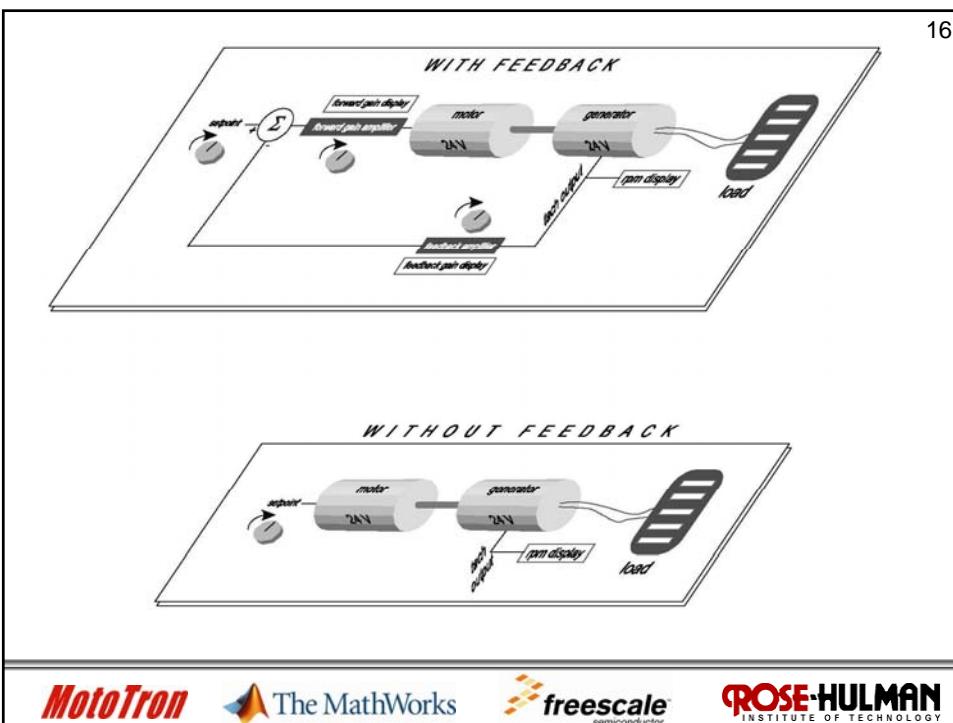
The MathWorks





## High Level System

15

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



17

## Model-Based System Design

- To build our plant (and controller), we will use MBSD
  - Start with simple component models
  - Anticipate the appropriate system responses
  - Verify the model works correctly
  - Make ONE improvement
  - Understand effect on model
  - Make ONE improvement
  - Understand effect on model
  - Repeat



18

## Model-Based System Design

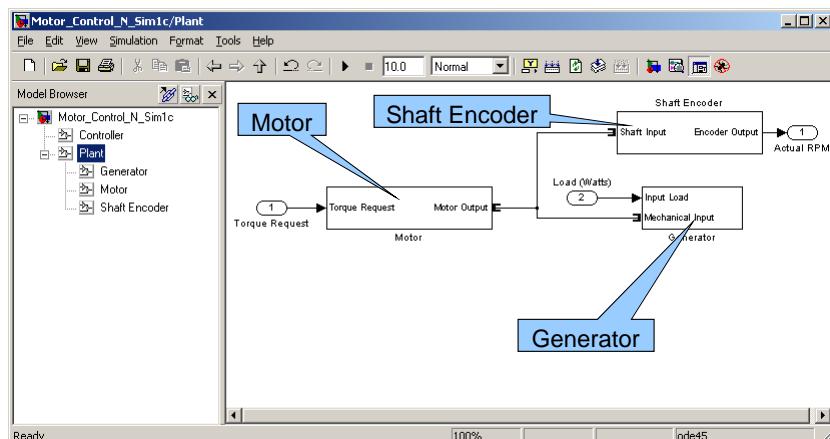
- We shall employ **Model Hierarchy** to represent plant components
  - Subsystems will contain the physical model for each component
  - Subsystems may contain subsystems that represent functionality
- We will also use SimDriveline to handle the kinetics and kinematics of our system





19

## MBSD – Model Hierarchy



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## MBSD

20

- Before we tackle the plant, let's build the top-level model found on slide 15
- Open a new model file and save it as Lecture2\_Model1.mdl

**MotoTron**

The MathWorks

**freescale**  
semiconductor

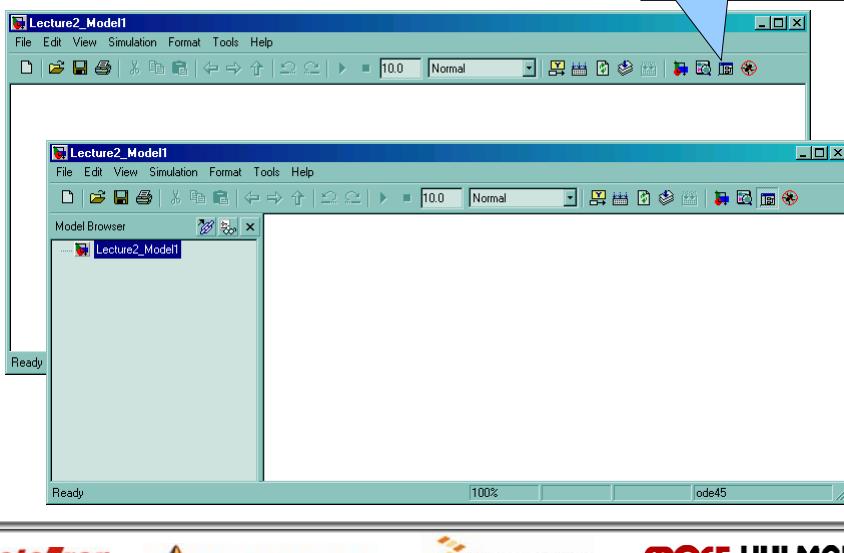
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## System Model

21

Toggle the model browser to show the subsystems



**MotoTron**

The MathWorks

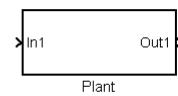
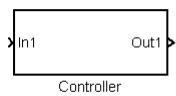
freescale<sup>®</sup>  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## System Model

22

- From **Commonly Used Blocks**, drag two **Subsystem** Blocks into your model
- Rename the one subsystem “Controller,” and the other “Plant”



- In the Model Browser window, left click on the Plant subsystem.

**MotoTron**

The MathWorks

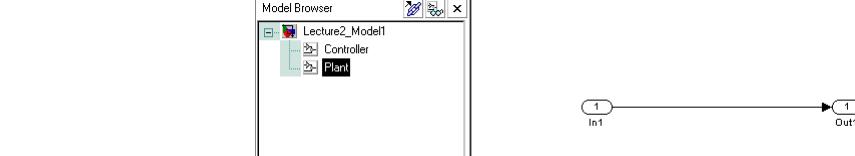
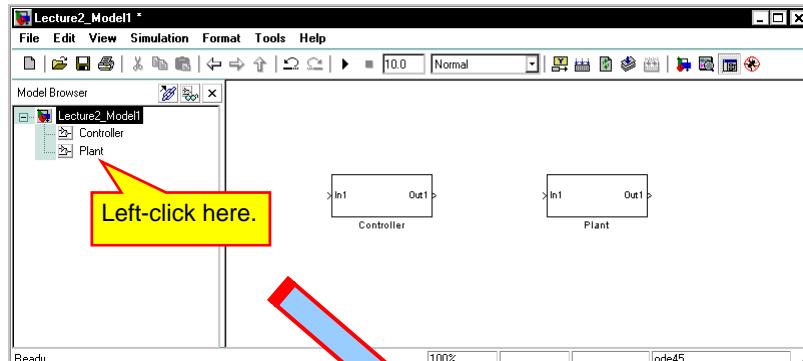
freescale<sup>®</sup>  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



23

## System Model



24

## Plant Model

- Our plant is not very exciting – it has an input and an output with nothing in between
- Delete the arrow connecting the input and output and add three more subsystems to represent the generator, motor, and shaft encoder
- Make your model look like the next slide





## Plant Model

Right click  
Select Format  
Select Flip Name

- Fantastic, you have got the pieces to build the plant
- Let's build a simple motor – left click on Motor in the Model Browser

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

## Motor

- The motor we are modeling is a Pacific Scientific model 4VM62-020-4 with the following name plate specs:
  - Torque Constant: 30 Oz-in per Amp
  - Max Current: 8 Amps
- Our driver will limit the motor to about 4 amps maximum.
- The max torque our motor will produce is 120 Oz-in.

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



## Motor

27

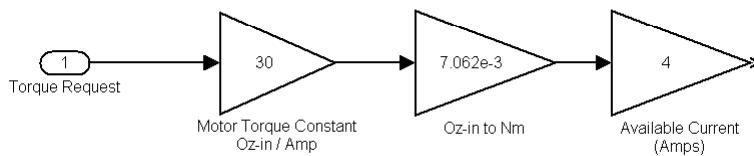
- SimDriveline uses MKS units so we need to convert our torque to Nm.
- The conversion factor is one Oz-In = 0.007062 Nm.
- Note that the maximum torque of our motor is 120 Oz-in = 0.85 Nm.



## Plant Model - Motor

28

- Our motor will operate by
  - Receiving a 0-100% max torque request from the controller
  - Multiplying this request with the motor's max rated torque (0.85 Nm)
- Drag a gain into the motor subsystem
  - Rename the In1 to Torque Request





29

## Plant Model - Motor

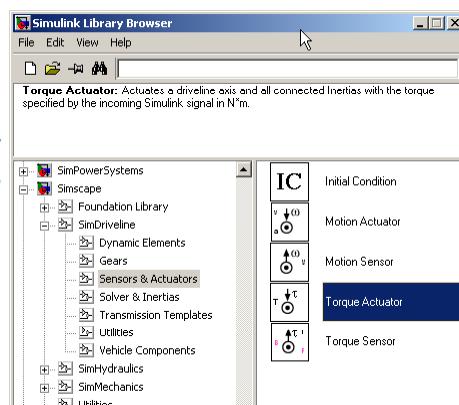
- We have created an ideal motor with
  - Variable torque from 0 to max rated value.
  - No rpm limits.
  - No energy conversion inefficiencies.
  - No frictional losses.
  - Torque is independent of rpm.
- This is a terribly inaccurate model, but it is an easy to understand first step.



30

## Plant Model - Motor

- Now we need to apply the motor torque to a drive line.
- Use SimDriveline
  - A Simulink physical library set for modeling dynamics
  - Kinetics and kinematics are built in
  - Plug & Play!
- From the Simscape library, go to **SimDriveline / Sensors & Actuators** and select **Torque Actuator**

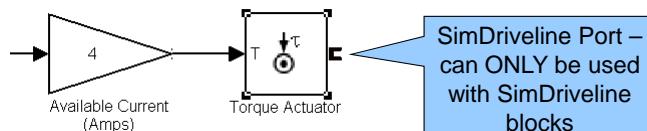




31

## Plant Model - Motor

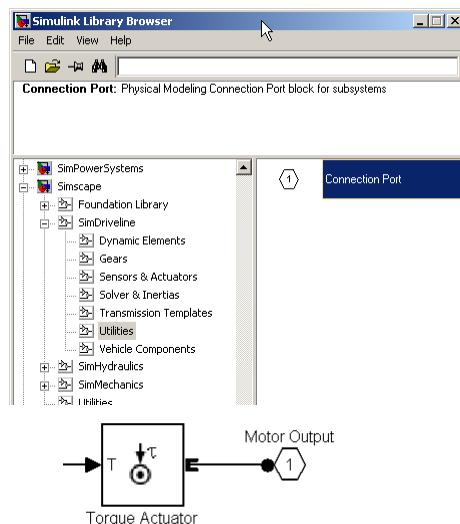
- Drag a Torque Actuator into the motor model
  - It converts a Simulink signal to a SimDriveline signal
- Link the output of the **Available Current** gain block to the input of the **Torque Actuator**



## Plant Model - Motor

32

- Since the SimDriveline port can only be used with SimDriveline blocks
  - Delete the Out1 port in the Motor subsystem
  - Drag in a **Connection Port** from the **SimDriveline/Utilities** library
  - Right click on the **Port**, select **Format**, and then **Flip Block**
  - Connect the Actuator to the Port
  - Rename the **Port** to **Motor Output**

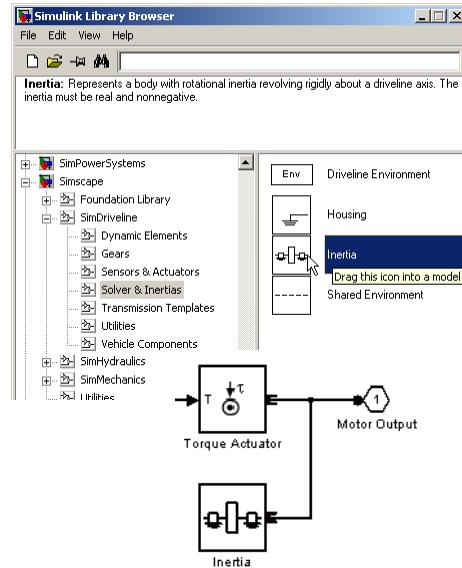




## Plant Model - Motor

33

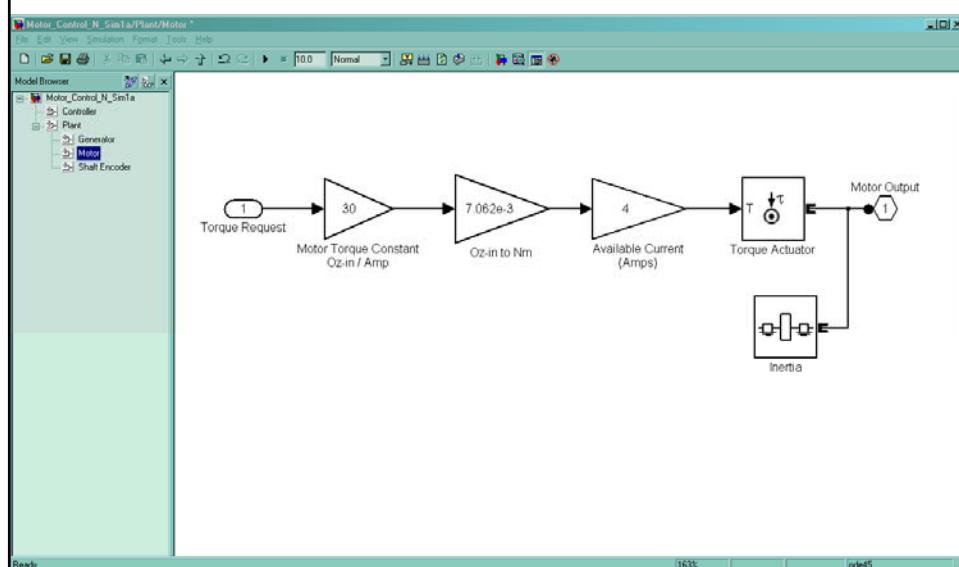
- All rotating devices have a mass moment of inertia
  - Drag in an **Inertia** from the **Solver & Inertias** library
  - Double click on it to give it the actual value of  $1e-3 \text{ kg}^*\text{m}^2$  for the motor
  - Connect the Inertia to the driveline



## Plant Model - Motor

34

- Our motor model is done!





35

## Lecture 2 Exercise 2

Modify the motor model to use the name plate specifications for the motors and generators used in the lab. Use the following items listed on the nameplate:

- Torque constant.
- Rotor Inertia.
- Max motor current is 6.3 amps. (Due to current limits on the DC power supply.)

Demo \_\_\_\_\_

**MotoTron**

The MathWorks

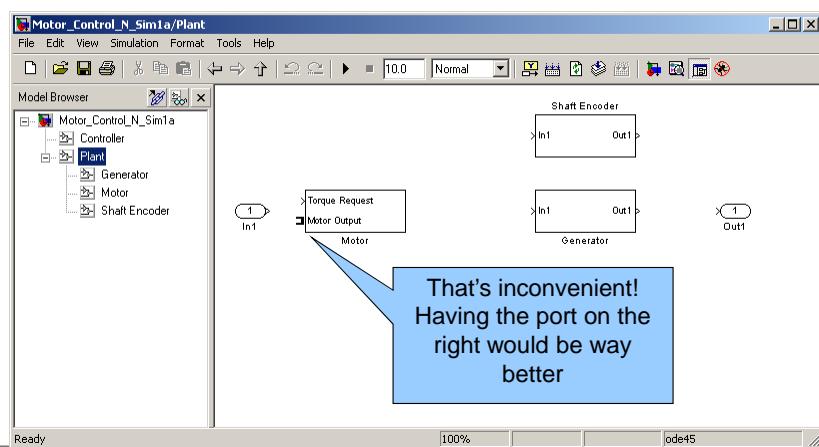
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

36

## Plant Model

- Return to the Plant level of the model



**MotoTron**

The MathWorks

**freescale**  
semiconductor

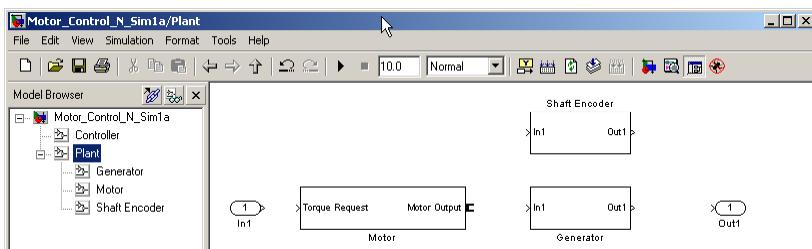
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



37

## Plant Model - Motor

- Use the **Model Browser** to return to the Motor subsystem and double click on the Connection Port (Motor Output)
- Change the **Port Location** from left to right
- Return to the Plant subsystem and resize the Motor subsystem block



38

## Plant Model - Generator

- Okay, let's now build a generator to be powered by the motor
- Now is a great time to do a version change
  - [Save your model.](#)
  - [Next, save the model as Lecture2\\_Model2.mdl](#)
- In the Model Browser, click on the Generator
- We know that there will need to be a SimDriveline port to connect to the Motor – add it and rename it Mechanical Input
- Delete the Simulink input and output ports.

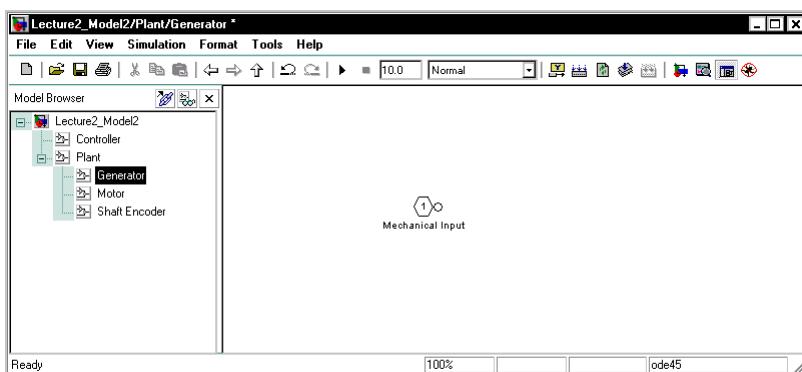




Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

39

## Plant Model - Generator



40

## Plant Model - Generator

- Our generator will operate as follows:
  - The Generator output is connected to a bank of 12 V light bulbs connected in parallel.
  - The bulbs are an electrical load.
  - As the generator spins, it generates a voltage that is proportional to generator speed.
  - The voltage is applied to the load and the load draws current.
  - To produce this current, the generator applies a torque to the shaft in opposition to the direction of rotation.





41

## Plant Model - Generator

- Effectively, the generator is a mechanical load to the motor that is controlled by the electrical load.
- Since:
  - The generator voltage is proportional to speed, and
  - The resistor current is proportional to voltage, and
  - The generator torque is proportional to current
- → The generator mechanical load is proportional to the generator speed.



42

## Plant Model - Generator

- Our first generator model will model the generator as a mechanical load that is proportional to speed.
- We will model the generator so that it achieves a maximum load torque of 1 Nm at 3000 rpm.
- There will be no user input for the load torque. We will assume that a fixed number of bulbs are turned on.





43

## Plant Model - Generator

- Drag the following into the generator
  - Gain
  - Divide
  - Torque Actuator
  - Motion Sensor
  - Inertia
  - Constant
  - Saturation
- and arrange according to the next slide
  - You'll have to flip some blocks around

**MotoTron**

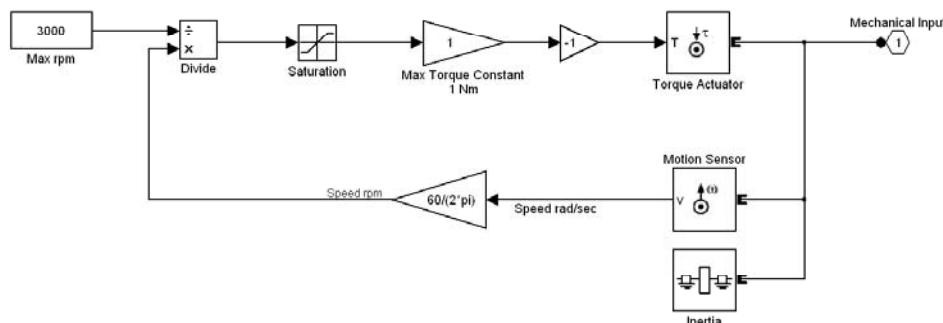
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

44

## Plant Model - Generator



**MotoTron**

The MathWorks

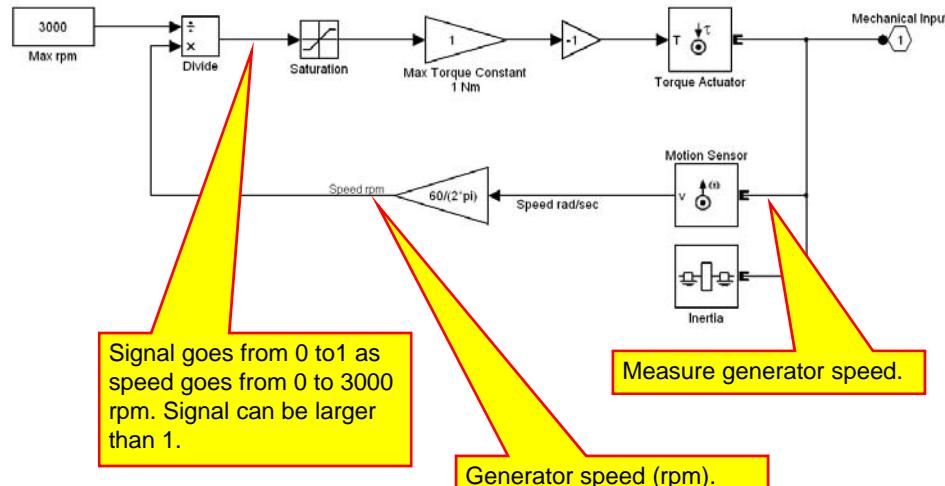
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



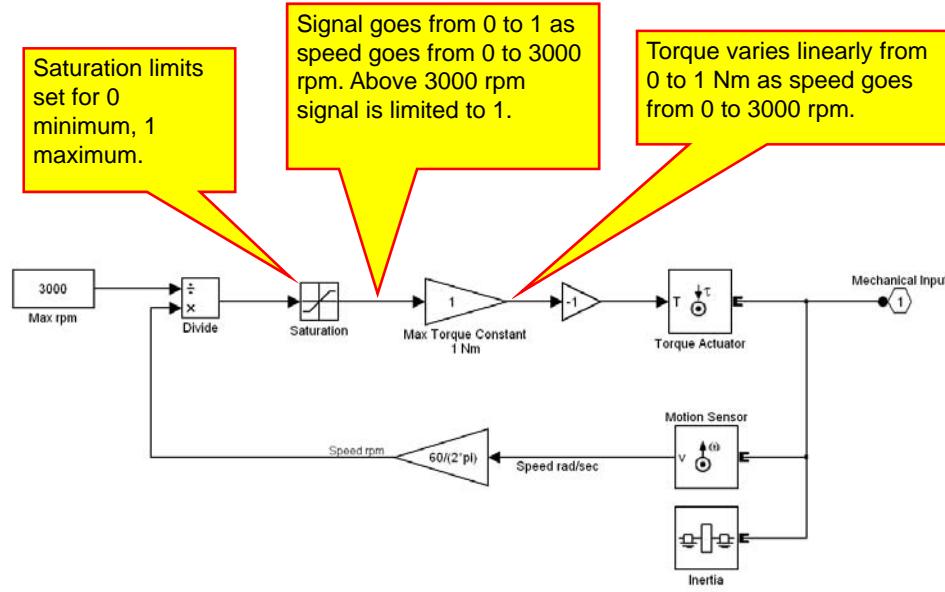
45

## Plant Model - Generator

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

46

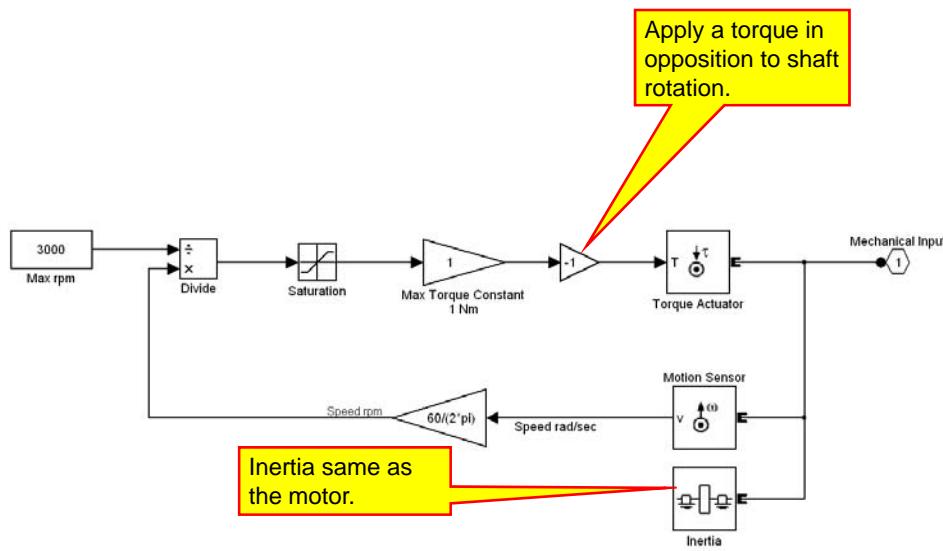
## Plant Model - Generator





47

## Plant Model - Generator

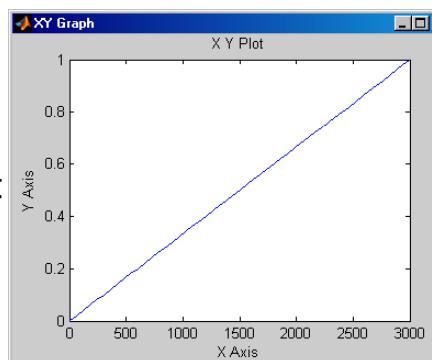


48

## Lecture 2 Exercise 3

- Verify the operation of your generator model.
- Copy the Generator subsystem to a new model.
- Generate a plot of generator load torque versus shaft rpm.
- You should obtain a plot as shown below:
- You may need to add a part called SimDriveline Env to get your simulation to run.

Demo\_\_\_\_\_

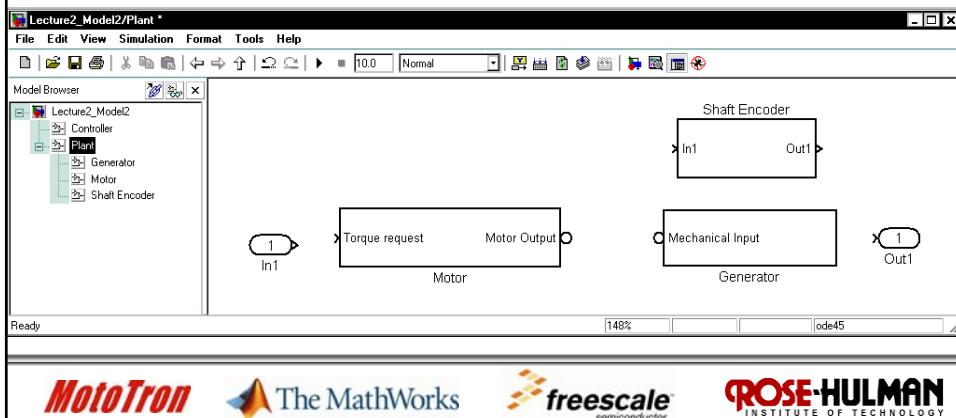




49

## Plant Model - Generator

- Click on the Plant in the Model Browser
- 2/3 of the way done!



50

## Plant Model - Encoder

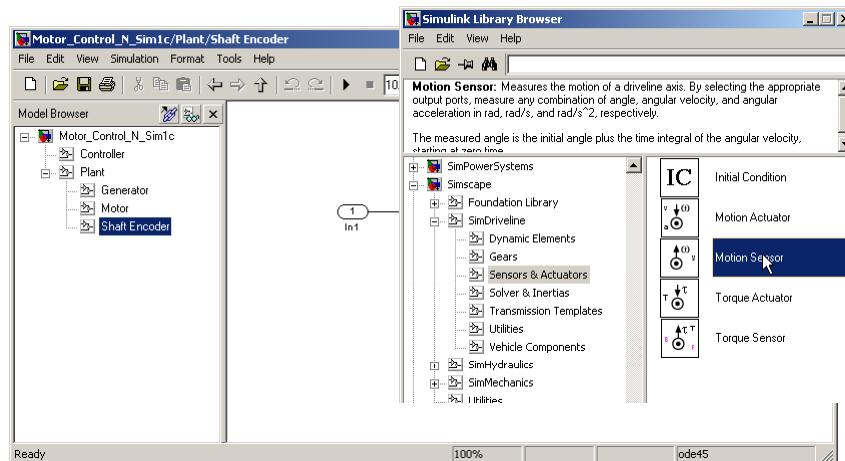
- Our final step is to read the shaft speed through a shaft encoder
- We will use the SimDriveline Motion Sensor to convert the SimDriveline signal to a Simulink signal
- Save your model
- Resave your model as Lecture2\_Model3.mdl
- Click on the Shaft Encoder in the Model Browser
  - Drag in a **Connection Port**
  - Drag in the **Motion Sensor** from the **Sensors & Actuators** library





51

## Plant Model - Encoder



**MotoTron**

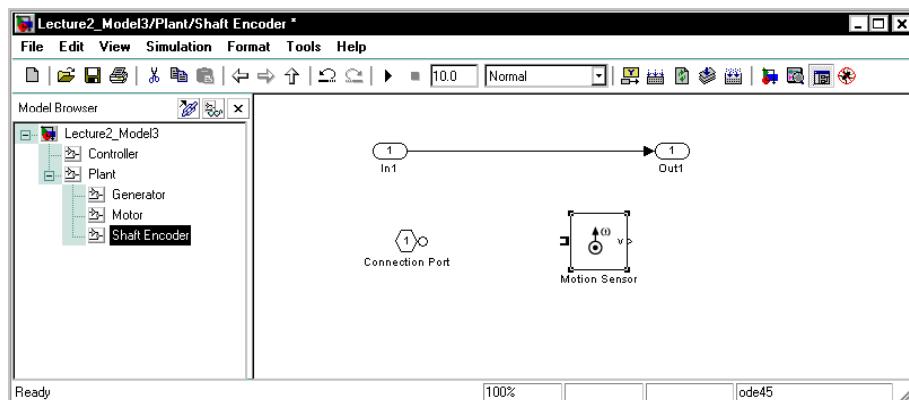
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

52

## Plant Model - Encoder



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



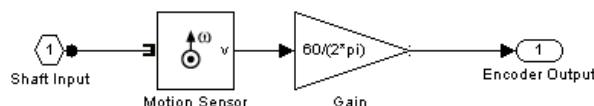
## Plant Model - Encoder

53

- Delete the In1 port
- Rename the Connection Port to Shaft Input
- Drag in a **Gain** to convert rad/s to rpm
- Rename the Out1 port to Encoder Output
- Link everything together

## Plant Model - Encoder

54



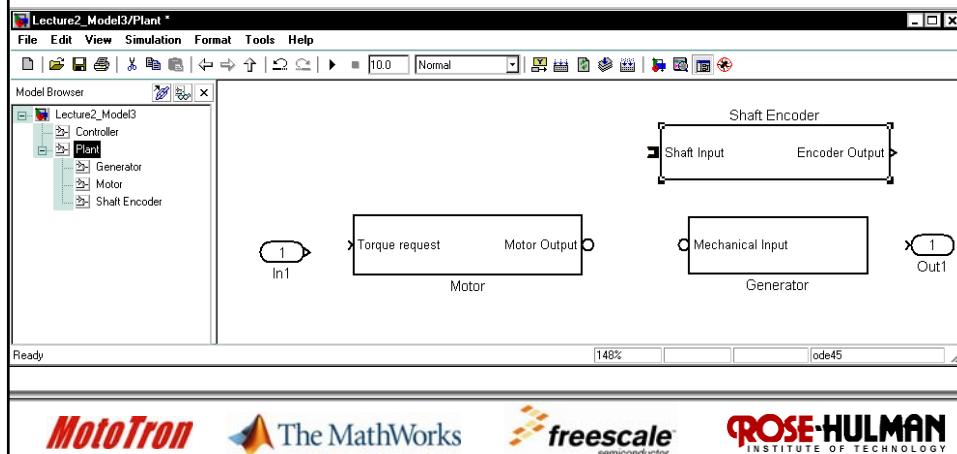
- Outstanding – the encoder is done!



55

## Plant Model – Whole Plant

- In the Model Browser, click on Plant
- Adjust the size of the Encoder



56

## Plant Model – Whole Plant

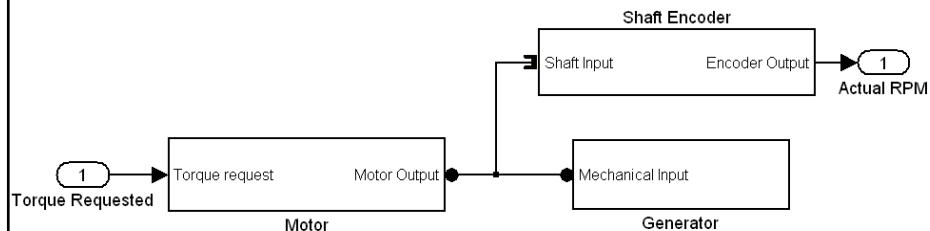
- Link the Motor Output to the Mechanical Input of the Generator
- Rename In1 to Torque Request and link to the Motor Torque Request port
- Rename Out2 to Actual rpm and link to the Encoder Output port
- Link the Shaft Encoder Shaft Input to the driveline





57

## Plant Model – Entire Plant



## Environment

58

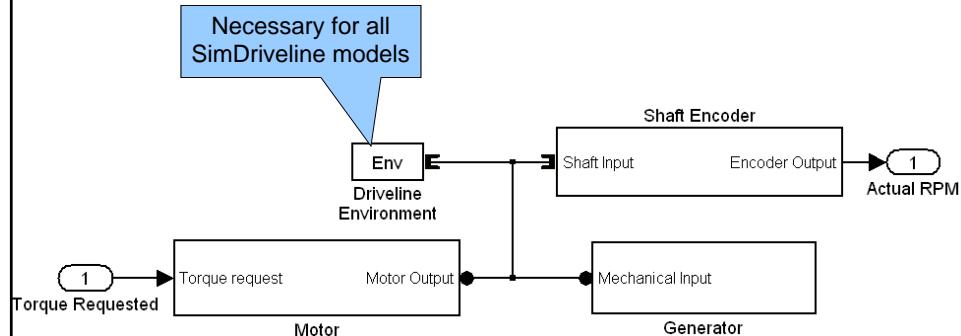
- We are just about ready to go but are missing one thing
  - **Driveline Environment**
- This enables Simulink to utilize the SimDriveline blocks
- In the SimDriveline **Solvers & Inertias** library, drag a **Driveline Environment** block into your plant and connect to the driveline





59

## Environment



Completed Model Demo \_\_\_\_\_

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

Any Questions?

**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/>



## Introduction to Model-Based Systems Design

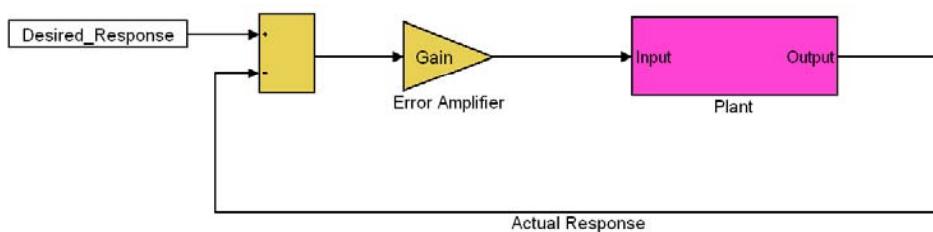
### Lecture 3: Controller Design and Testing



## Presentation Outline

2

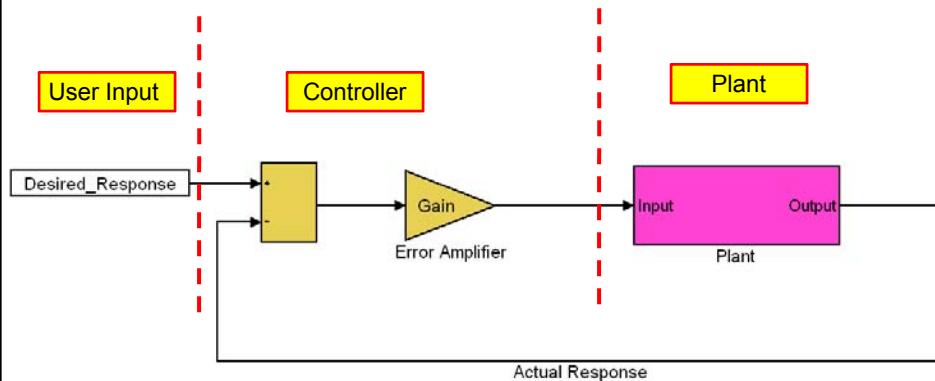
- Proportional Control
- System Diagnostics
- Testing
- Understand the operation of a proportional controller





## Proportional Control

3

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

## Proportional Control

4

Error – Difference between desired and actual response.

Amplifier error signal. Gain  $\geq 0$ .

Desired\_Response

Error Amplifier

Input

Output

Plant

Actual Response

Input to the plant is proportional to the difference between the desired plant response and actual plant response.

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



## Getting Started

5

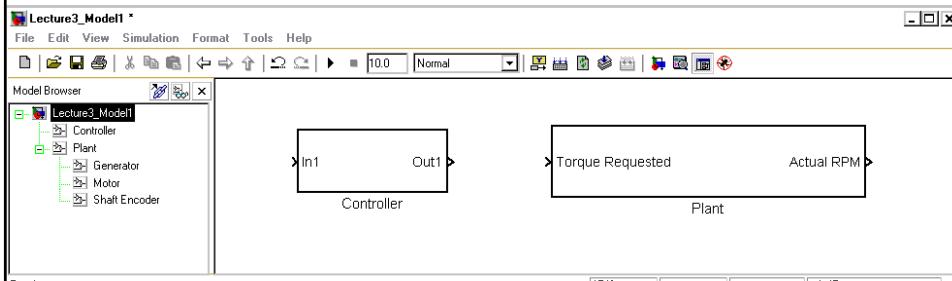
- It's time to do a version switch
- Save your model.
- Save your model again as
  - [Lecture3\\_Model1.mdl](#)



## Getting Started

6

- Use the Model Browser to go to the top level of your model





7

## Control Strategy

- Our plan is to implement a proportional (P) controller based on system speed
  - We would like the motor rpm to be held constant relatively independent of the mechanical load.
  - This will be done by modifying the amount of torque produced by the motor based on the difference between the desired and actual motor speed.



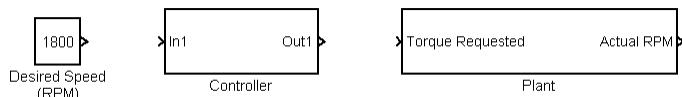
The MathWorks



## Whole System

8

- At the top level of your model, drag in a **Constant** from the **Commonly Used Blocks** library
  - It will be the desired system speed
    - Double click and set its value to 1800 rpm
    - This will be a user input in the physical realization (a knob).



The MathWorks





9

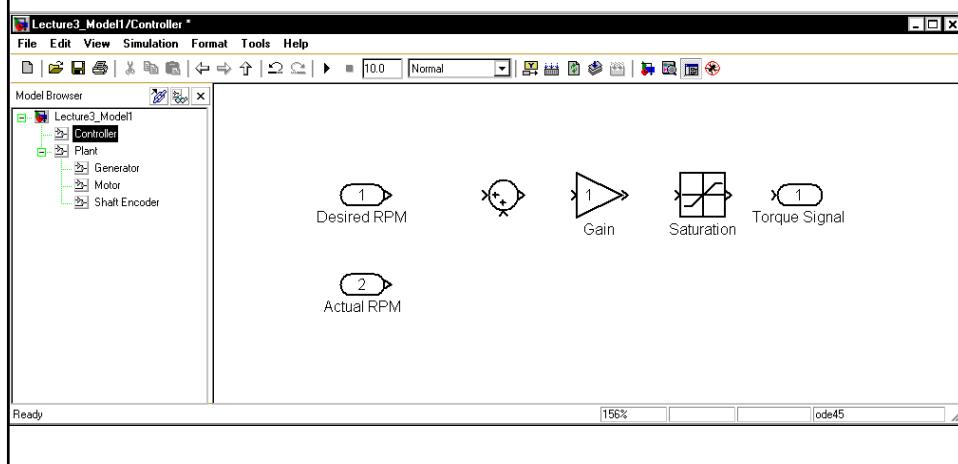
## Controller

- Open the Controller subsystem within the Model Browser
  - Rename Out1 to “Torque Signal”
  - Rename In1 to “Desired rpm”
  - Add a second In1 and rename it “Actual rpm”
  - From **Commonly Used Blocks**, drag in a
    - Sum
    - Gain
    - Saturation



10

## Controller

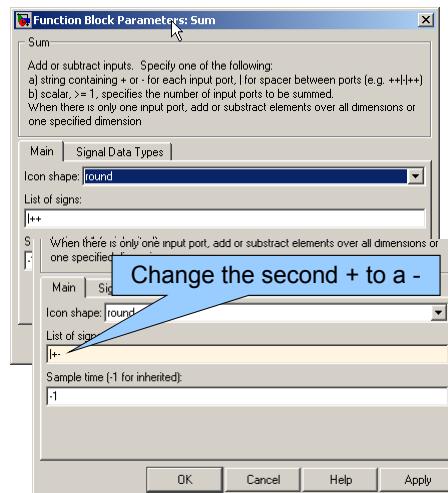




11

## Controller

- Double click on the **Sum** block
- Change the List of signs: to **|+-** so that the second signal is subtracted
- Click **OK**



**MotoTron**

**The MathWorks**

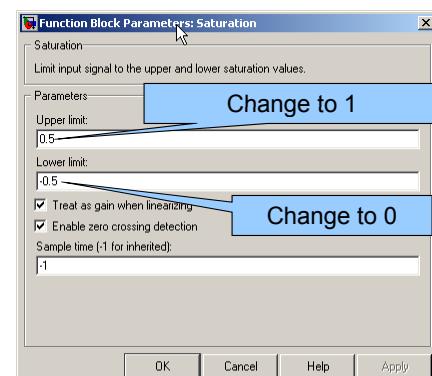
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

12

## Controller

- Double click on the **Saturation** block
- Change the **Upper** and **Lower** limits to 1 and 0 respectively
  - This allows our torque request to go from zero to max positive torque
  - Our motor cannot produce a negative torque.
- Click **OK**

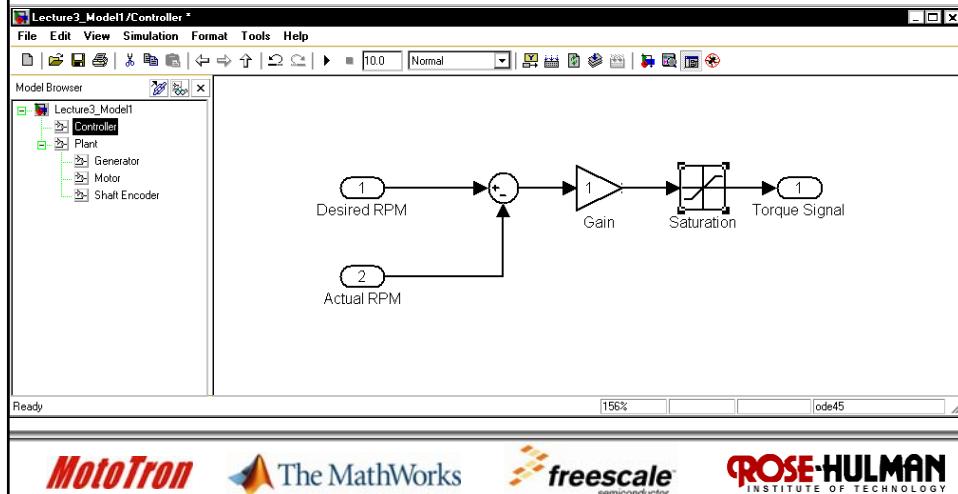




13

## Controller

- Now link everything together



14

## Controller

- The last thing to do is estimate the value for the **Gain** block
    - Usually classic control theory tells us this
      - We're not going to do this here.
    - In many systems where we use Model-Based-System Design, the systems are so complicated that we do not have sufficient models to do the classical analysis.
- Use common sense, then tune



The MathWorks

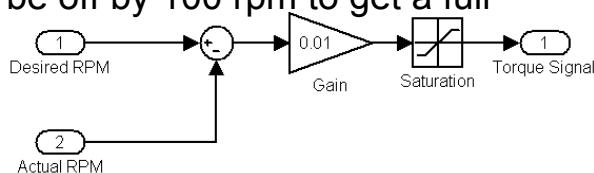




15

## Controller

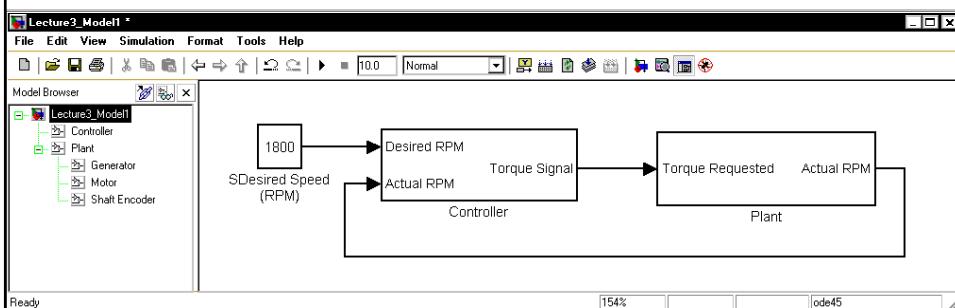
- We are looking at the difference in between desired and actual rpm
  - For a gain of 1, if we are below the desired rpm by 1 rpm, the controller will ask the motor for full torque.
    - That's a bit excessive!
  - For a gain of 1, if we are slightly above the desired rpm, the torque will drop to zero.
    - That's a bit excessive!
    - Could yield oscillations and broken shafts!
  - As a starting point, let's set the gain to 0.01.
- The rpm must be off by 100 rpm to get a full torque signal



16

## Top-Level Block Diagram

- Go back to the top level of your model and connect everything.



- Looks like a classical feedback system.





17

## Initial Condition

- We are just about ready to run our model.
- Left click on the Plant in the model browser
- Drag in an **Initial Condition** block from the **SimDriveline / Sensors & Actuators** library
- Connect it to the driveline.
- Double click and give it a value of 0 rad/s (0 rpm)
- Click **OK**
- Note that without this block the initial speed always start at zero. Using this initial condition allows us to start the motor at a high rpm and then see how it settles down to the desired rpm.

**MotoTron**

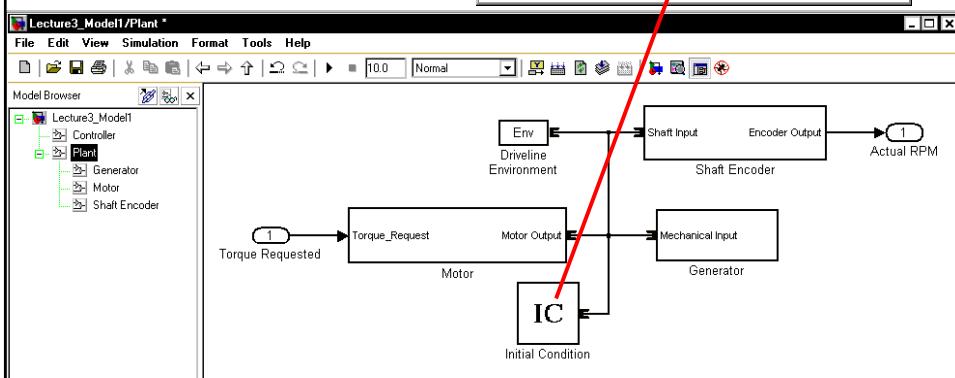
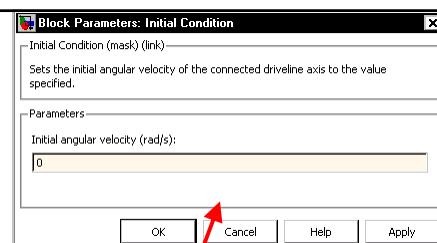
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

18

## Initial Condition



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



19

## System Diagnostics

- Before we run our model, we may want to review the results in graph form.
- We will utilize a scope
- Process:
  - Name all signals of interest
  - Attach them to the scope
- Preliminary signals of interest
  - Desired and actual rpm

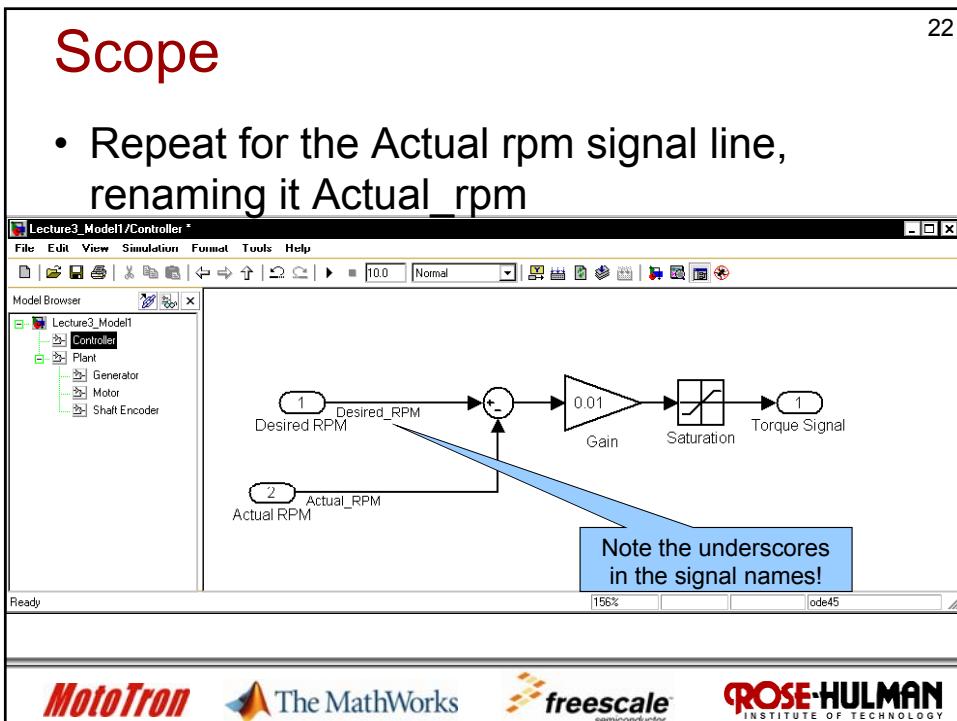
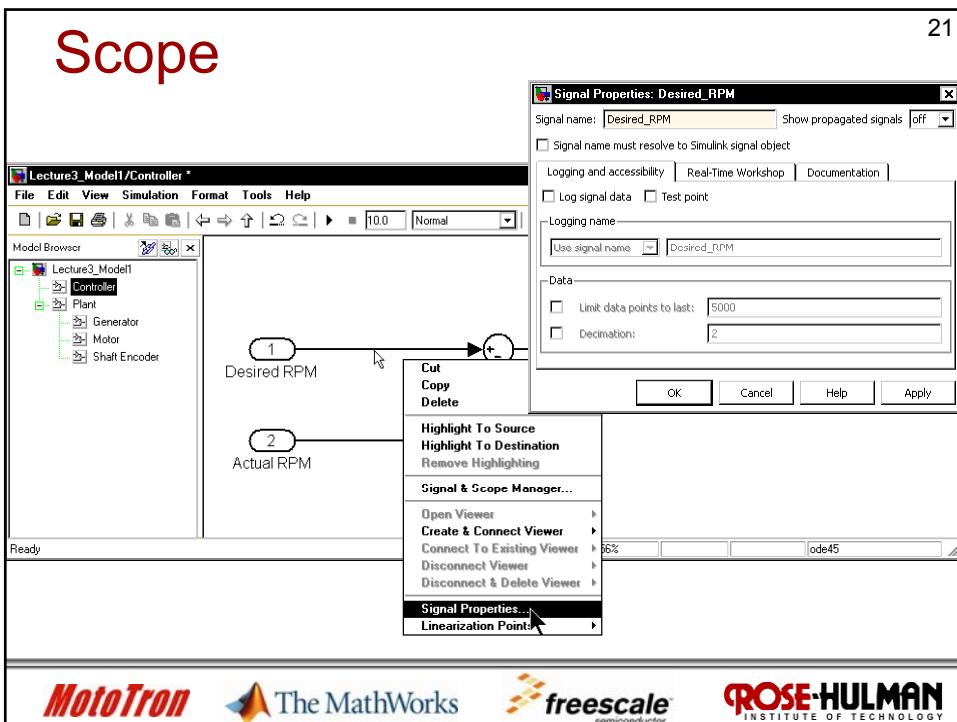


## Scope

20

- In the **Model Browser**, left click on the Controller
- Right click on the signal line which connects the Desired rpm port to the sum
- Select **Signal Properties** from the menu.
- Change the **Signal Name** to Desired\_rpm
- Click the **OK** button.



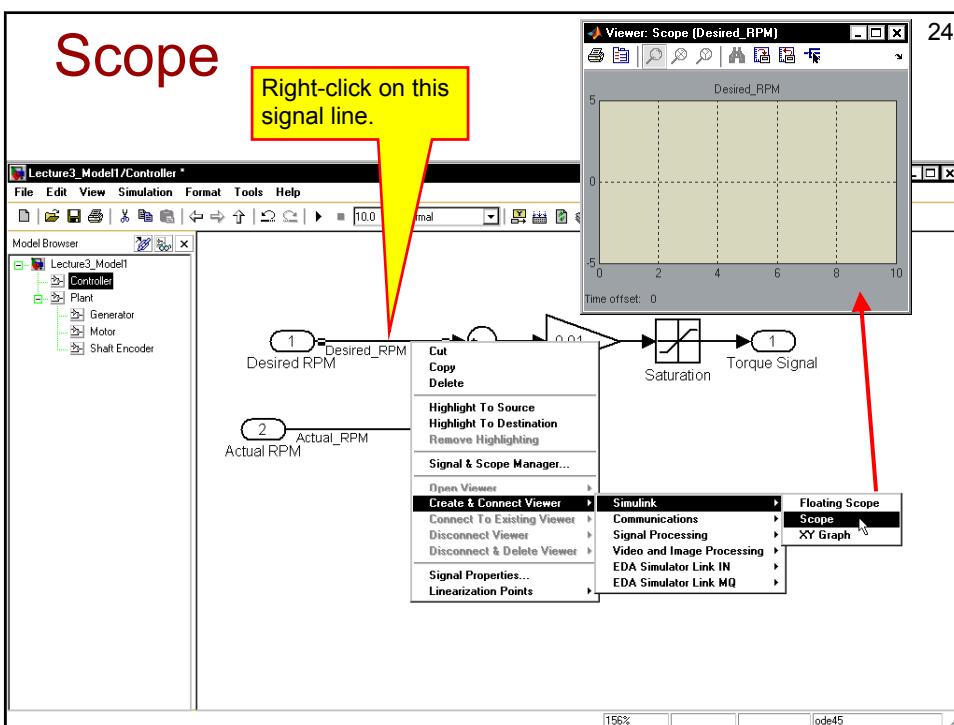




23

## Scope

- Now we need to create a viewer and attach the signals.
- Right click on the Desired\_rpm signal line.
- (1) Select **Create & Connect Viewer**, (2) select **Simulink**, then (3) select **Scope**, from the menus.
- Congratulations – you have created a scope that displays a single signal, Desired\_rpm.





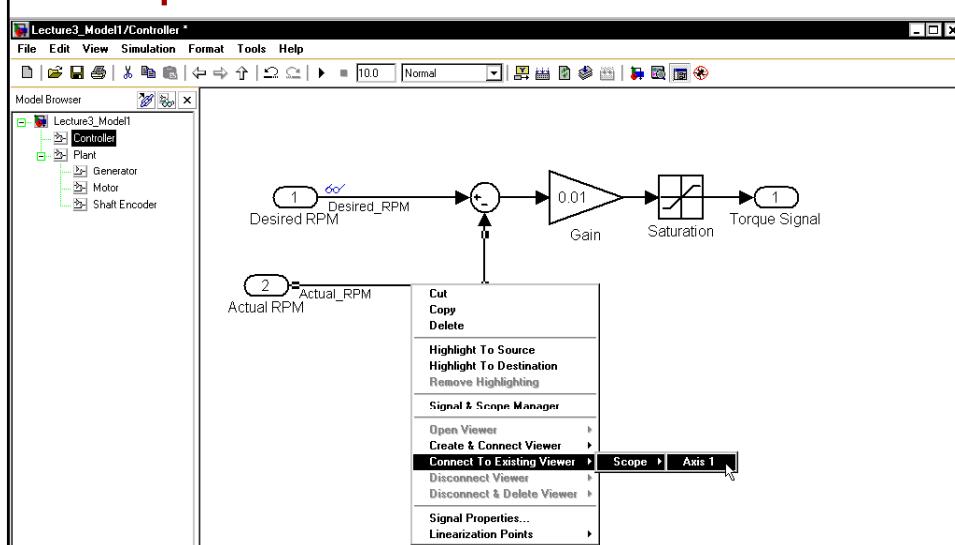
25

## Scope

- Now we need to display the actual rpm on the same scope and on the same axis.
- Right click on the Actual\_rpm signal line and select **Connect to Existing Viewer**, then select **Scope**, then select **Axis1**.

26

## Scope

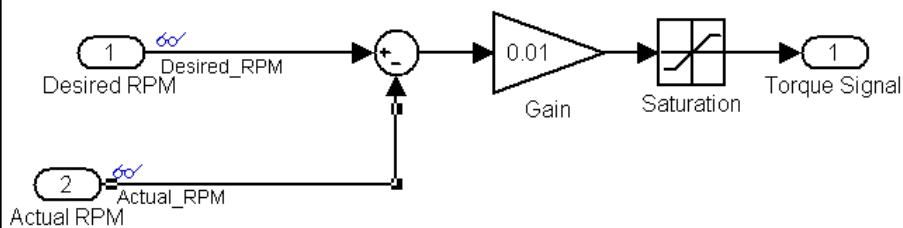




27

## Scope

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



28

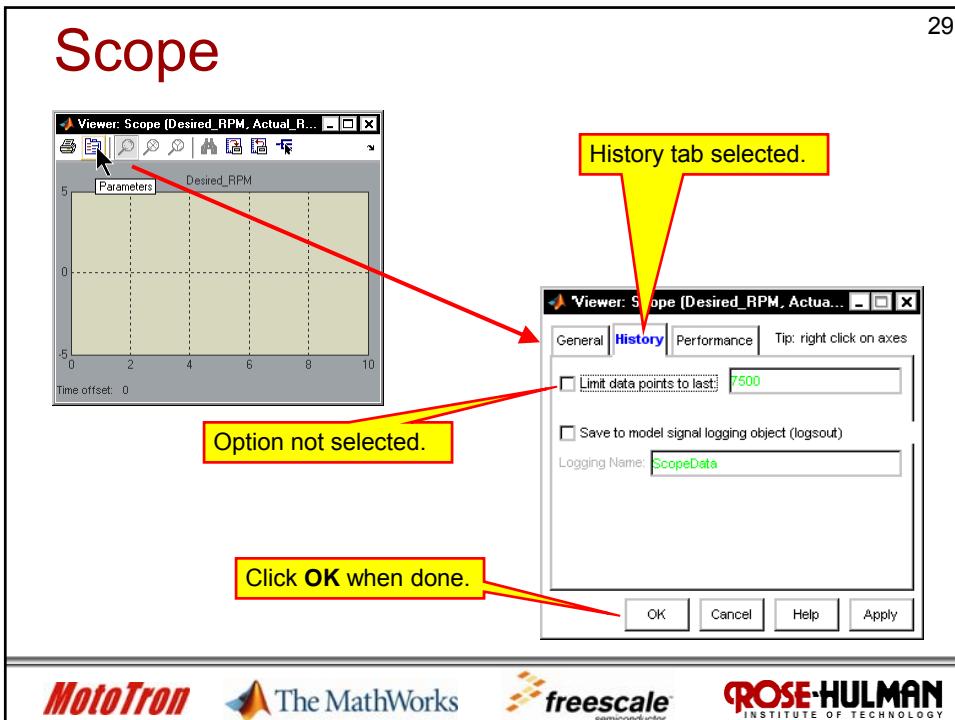
## Scope

- 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.
- Click on the **Parameters** button in the scope window as shown next.
- Select the **History** tab and uncheck the option as shown.



## Scope

29

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

## Execution

30

- We can now run our model.
- Before doing so, how should our system respond?
  - When the rpm is well below 1800 rpm, the motor should apply a large torque to the system.
    - The motor should quickly approach 1800 rpm.
  - At close to 1800, the motor torque is reduced and the rpm should hold close to 1800 rpm.
  - Let's see if our simple system and simple controller behave as expected.

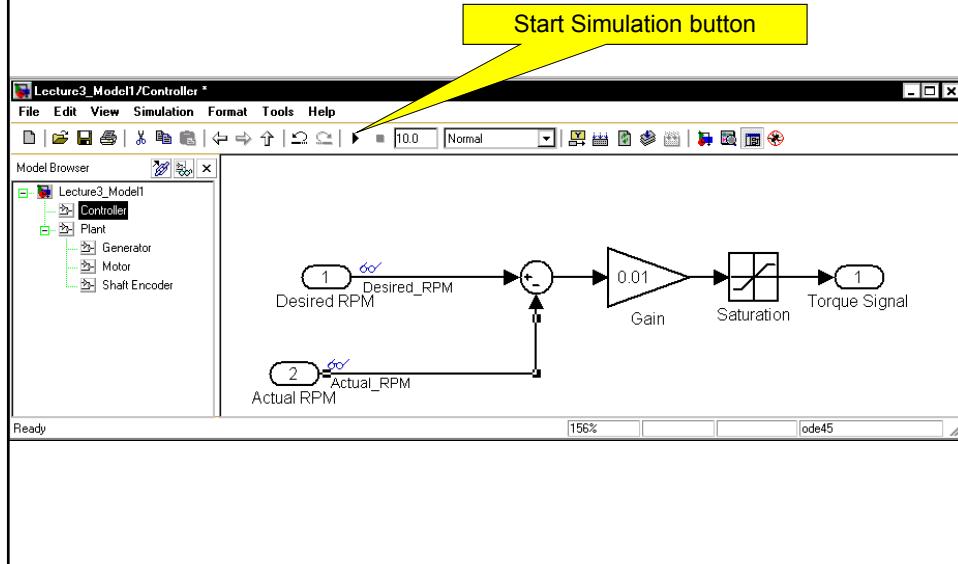
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



31

## Execution

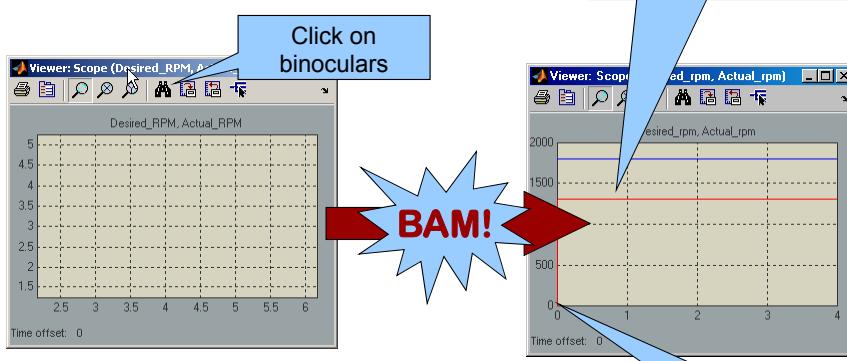
- Click the Start Simulation button



32

## Execution

- Observe the Scope...



- It does not work!





33

## Model Problems

- The motor speeds up very quickly. Why?
- The motor speed never reaches the desired rpm. Why?



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Lecture 3 Exercise 1

34

The motor cannot reach the desired speed of 1800 rpm.

We would like to find out why.

1. Change the scope y-axis so that the range is 0 to 2000 by default.
2. Modify the scope so that it displays 4 plots in the same window
3. Display following signals:
  1. Desired and actual rpm (already done).
  2. Controller torque requested.
  3. Motor Torque
  4. Generator Torque
4. Determine why the motor cannot reach the desired speed and fix it.

Demo \_\_\_\_\_



The MathWorks

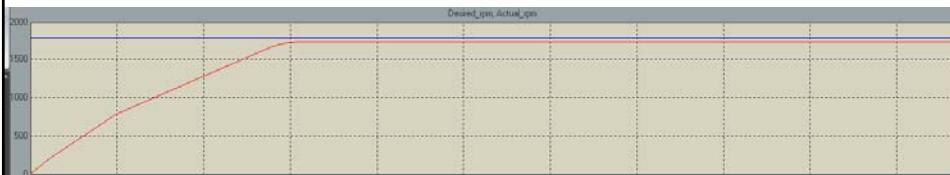
freescale<sup>®</sup>  
semiconductor



35

## System Observations

- The system goes from 0 to 1800 rpm in about 6 ms.



- Does this make sense?
- Yes, because the inertia of the system is so small.



36

## System Observations

- A 6 ms step response seems quite fast for this system.
- The motors are specified as ultra-low inertia motors, so this might be reasonable.
- We should figure out a way to measure the inertia.
- Such a small inertia will make the system hard to control.





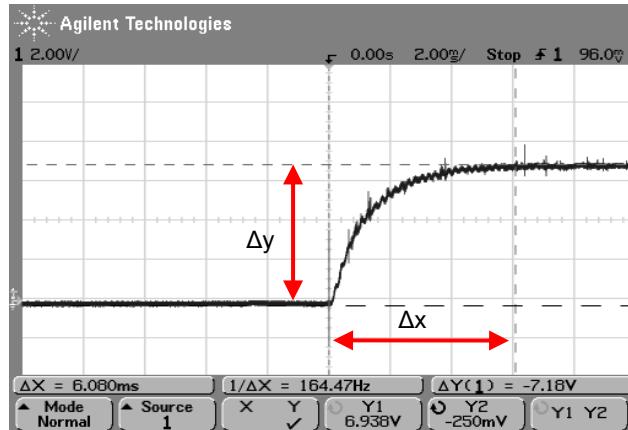
## Motor Step Response

37

- 0 to 99% takes about 6 ms ( $\Delta x = 6.080$  ms)

- Tach scaling = 2.5 V per 1000 rpm

$$\begin{aligned}\Delta rpm &= \Delta y \left( \frac{1000 rpm}{2.5V} \right) \\ &= (7.18 V) \left( \frac{1000 rpm}{2.5V} \right) \\ &= 2872 rpm\end{aligned}$$



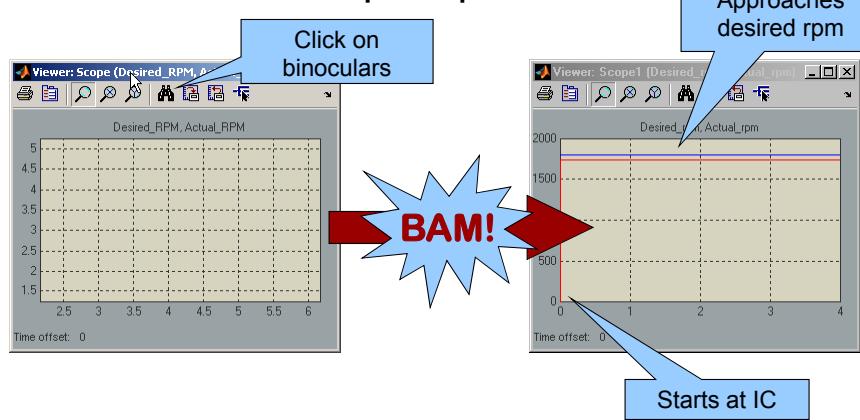
With this fast step response, the response of the motor on slide 34 seems reasonable.



## Model Testing and System Comprehension

38

- Observe the step response...





39

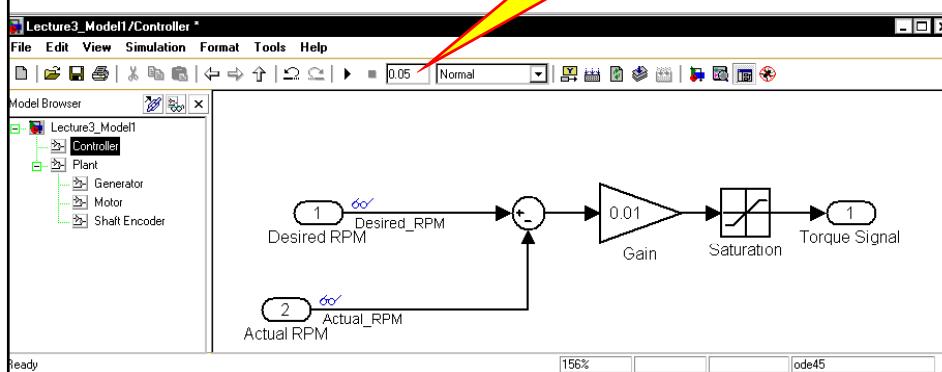
## Issues

- Note that we have but approached the desired rpm but we don't seem to achieve it – why?
- Because this is a proportional control method and the error signal times the gain yields the torque necessary to drive the motor!
- Also note that 4 seconds of simulation time is not needed.
- Change the simulation end time to 0.05 seconds as shown next:

40

## Simulation Time

Changed to 0.05 seconds.

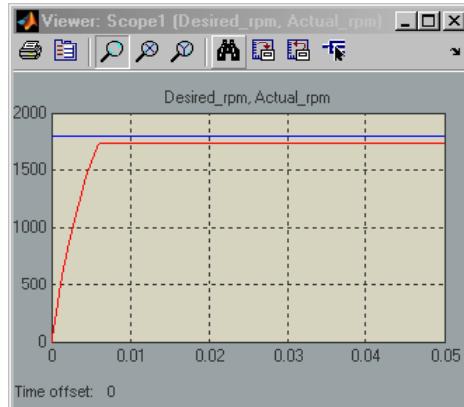




41

## Simulation Time

- Rerun the simulation and observe the results:



## Verification

42

- Let's now make sure our system behaves as expected
- Perform the following tests, write down what you think should happen, and verify that it does





43

## Verification – Initial Conditions

- Test 1 - Set IC to 250 rad/s, all else constant
  - Expected \_\_\_\_\_
  - Simulation \_\_\_\_\_



The MathWorks

freescale<sup>®</sup>  
semiconductor

44

## Verification – Controller Gain

- For the next tests, hold the IC at 0 rad/s, the desired rpm at 1800
- We shall experiment with the controller gain.



The MathWorks

freescale<sup>®</sup>  
semiconductor



45

## Lecture 3 Demo 1

### Verification – Controller Gain

- Test 3 - Set controller gain to 0.001
  - Steady state error (rpm) \_\_\_\_\_
  - Steady state stability \_\_\_\_\_
- Test 4 – Set controller gain to 0.01
  - Steady state error (rpm) \_\_\_\_\_
  - Steady state stability \_\_\_\_\_
- Test 5 – Set controller gain to 0.1
  - Steady state error (rpm) \_\_\_\_\_
  - Steady state stability \_\_\_\_\_
- Test 6 – Set controller gain to 1
  - Steady state error (rpm) \_\_\_\_\_
  - Steady state stability \_\_\_\_\_

Demo \_\_\_\_\_



46

### Verification – Controller Gain

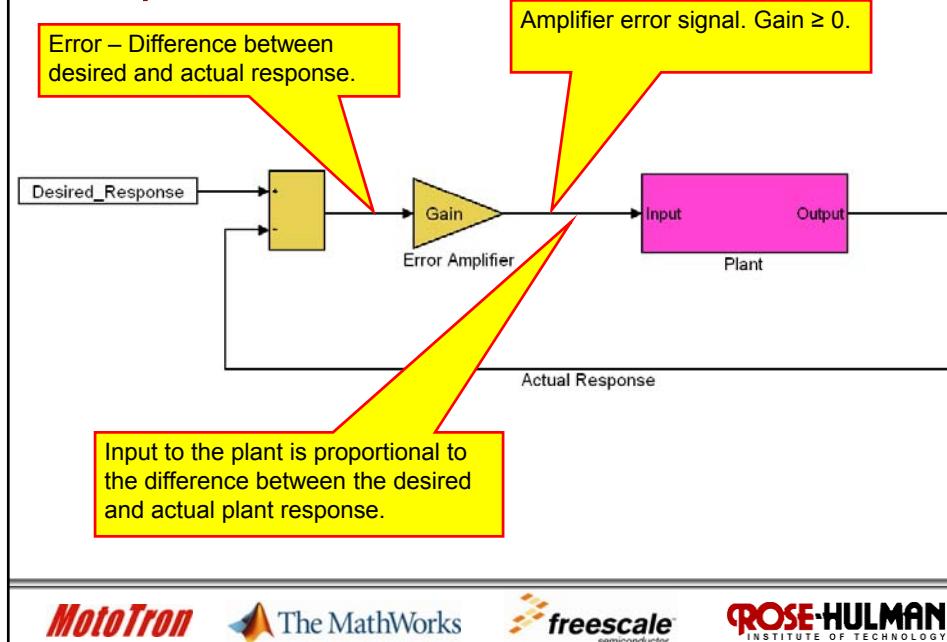
- Test 3 - Set controller gain to 0.001
  - Steady state error (rpm) 500 rpm
  - Steady state stability stable – Did not reach 1800
- Test 4 – Set controller gain to 0.01
  - Steady state error (rpm) 70 rpm
  - Steady state stability stable - Basically
- Test 5 – Set controller gain to 0.1
  - Steady state error (rpm) 7 rpm
  - Steady state stability Slight Oscillation 2 rpm p-p Mid Freq
- Test 6 – Set controller gain to 1
  - Steady state error (rpm) <1 rpm
  - Steady state stability oscillates - 1 rpm p-p, High Freq





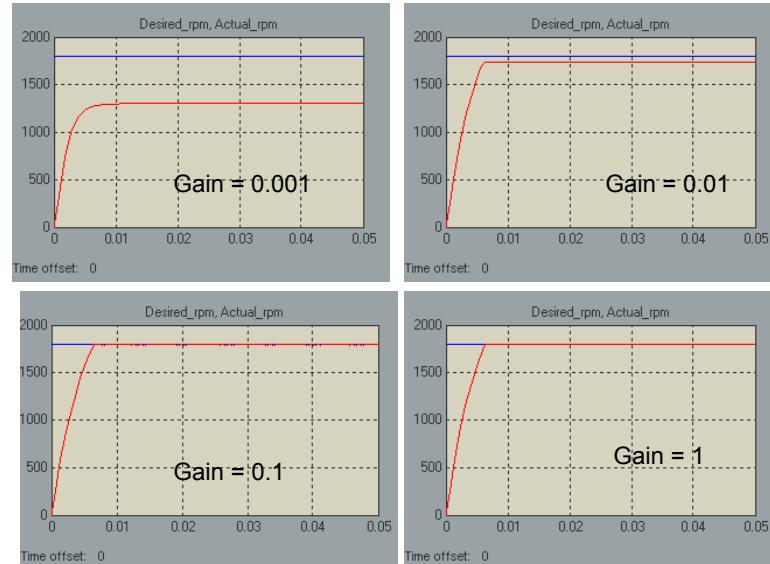
47

## Proportional Control

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

48

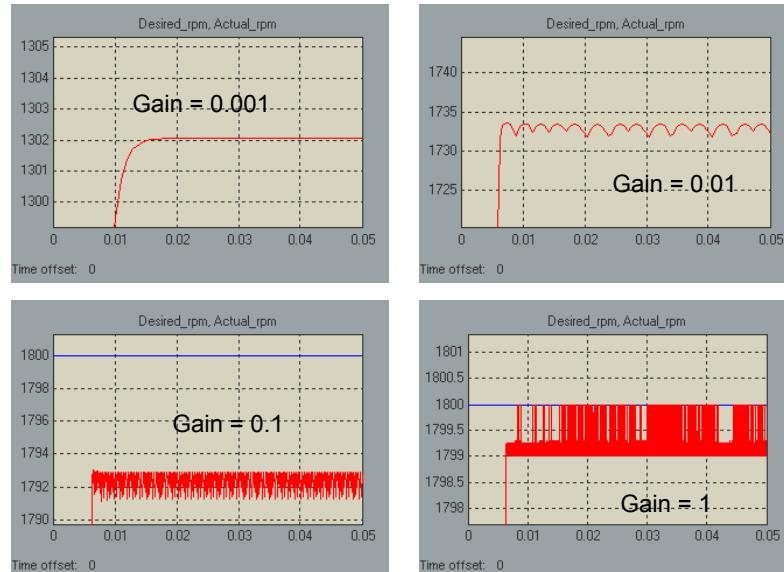
## Verification – Controller Gain

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



49

## Verification – Controller Gain



50

## Verification – Controller Gain

- Increasing the gain
  - Reduces the steady state error
  - Can lead to oscillation



51

## MBSD

- This is why we do MBSD
  - Understand a complex system using simple components
  - Understand effect of controller and components / loads on system response
  - Size components to within the ballpark
  - Identify model shortcomings for future improvement
- Good job!



## Controller Design and Testing

Incremental Improvement to the Model





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

53

## Outline

- Motor
  - Torque will be a function of rpm
- Generator
  - Load torque can be changed by changing the number of light bulbs.
- Friction
  - Will be added via a clutch
- MPC555x
  - Signals will be scaled accordingly



54

## Getting Started

- It's time to do a version switch.
- Save your model.
- Save your model again as
  - [Lecture3\\_Model2.mdl](#)





55

## Improved Motor

- We will improve the motor model by making the motor torque a function of motor rpm.
- Use the same method used previously to measure the motor rpm.
- Use a lookup table to implement the rpm dependence of the motor.
- Obtain a lookup table from the manufacturer.
- Later on we will design an experiment and measure the actual torque curve of the motor.



56

## Improved Motor Model

- The following torque curve was obtained from the manufacturer. (Not really, but we'll pretend.)

Rpm	Torque (Nm)
0	0.4
1000	0.4
2000	0.3
3000	0.2
4000	0



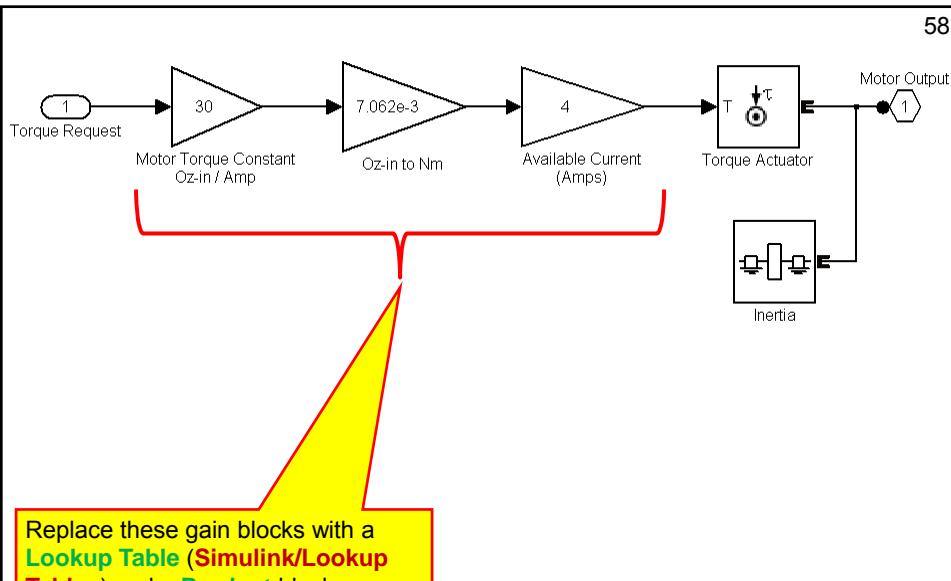


57

## Motor Model

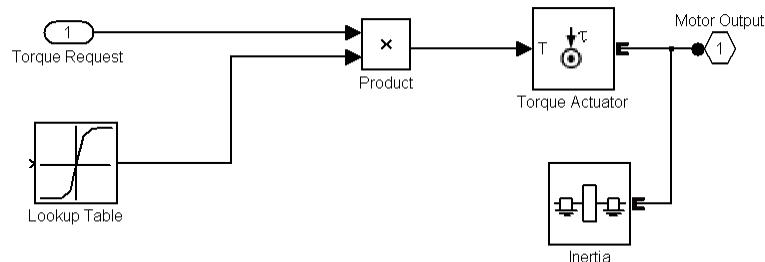
- Now that we have our motor torque curve data, we can use it in our model.
- We will use a 1-D look up table.
- Our motor model presently has a constant torque curve:

58





59



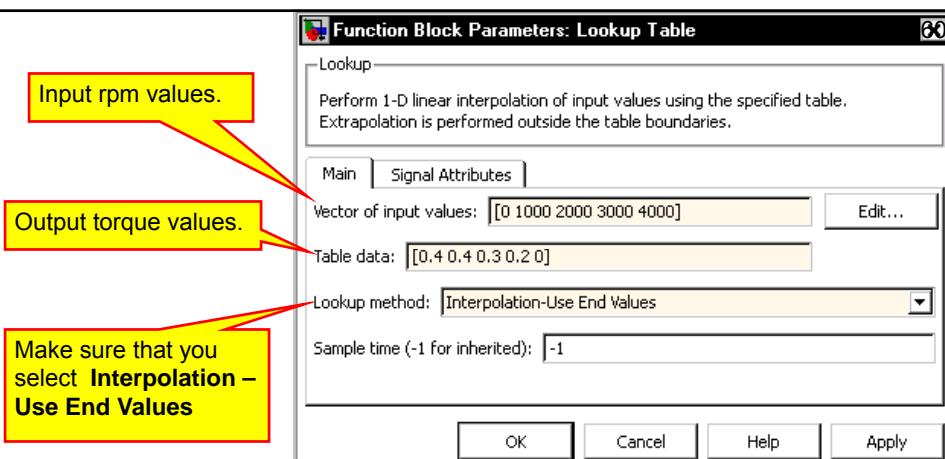
Double-click on the lookup table and specify the parameters as shown:

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



- The output of the part (Table data) is the motor torque.
- The input to this part is the motor rpm.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

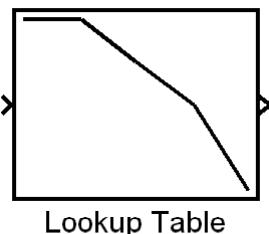
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



61

## Motor Model

- When you click the OK button, the torque curve will be displayed on the lookup table block.



62

## Motor Model

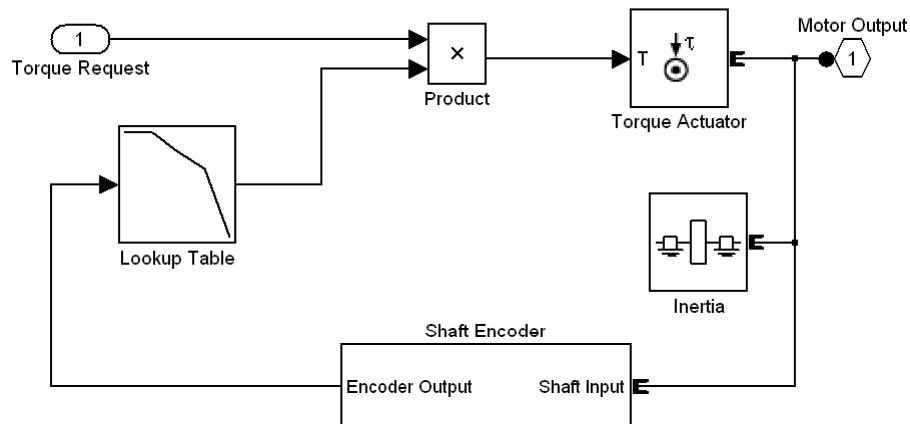
- We need to measure the motor speed and convert the units from rad/sec to rpm.
- We have already done this with a shaft encoder subsystem.
- We can copy the shaft encoder subsystem and then paste it inside the motor model.
- Finally, we will feedback the rpm value to the table.





63

## Motor Model



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

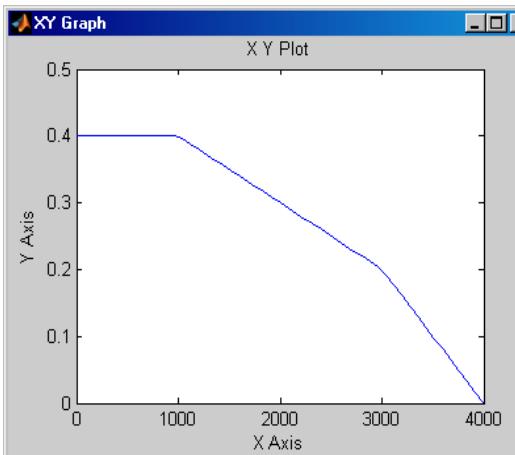
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Lecture 3 Exercise 2

64

- Verify the operation of your motor model.
- Copy the Motor subsystem to a new model.
- Generate a plot of motor torque versus shaft rpm.
- You should obtain a plot as shown:
- You may need to add a part called SimDriveline Env to get your simulation to run.

Demo\_\_\_\_\_





65

## Verification - Motor

- We have made a simple change to our model – let's make sure it operates as we expected.
- Run the following experiments and observe the behavior. We should see similar results as in the previous simulation.
- Set the shaft rpm initial condition back to zero.



66

## Verification - Motor

- To verify the motor, we should really observe the torque versus speed.
- We need to add a torque sensor and then plot it on our graph.
- We will create a new scope with two plots.
- To create a new scope, we use the same procedure we used earlier





67

## Adding a Scope

- In the first plot, we wish to display the desired and actual rpm.
- To create a new scope, right click on the signal you display (Desired\_rpm), and then select **Create & Connect Viewer**, then **Simulink**, and then **Scope** from the menu. This will display signal Desired\_rpm on the scope.
- To add a second trace to the graph, right click on the second signal you wish to display (Actual\_rpm), select **Connect to Existing Viewer**, then **Scope1**, then **Axis1** from the menus.



68

## Verification - Motor

- We now have two scopes for our model called “Scope” and “Scope 1.”
- “Scope” has 4 plots (probably) that we created in exercise 1.
- “Scope1” has a single plot. We would like to add a second plot.
- Click on the **Parameters** button in the Scope1 window as shown next.
- Select the **General** tab and set the number of axes to 2.





69

## Scope Settings

The screenshot shows two windows side-by-side. On the left is a scope window titled 'Viewer: Scope1 (Desired\_rpm, Actual\_rpm)'. It displays a plot of two signals over time, with the y-axis ranging from 0 to 2000 and the x-axis from 0 to 0.05. A red arrow points from the 'Number of axes' text box in the scope's parameter settings to the 'Number of axes: 2' field in the dialog. Another yellow callout box labeled 'General tab selected.' points to the 'General' tab in the dialog. The dialog also includes tabs for History, Performance, and Tip: right click on axes. The 'Axes' section contains fields for Number of axes (set to 2), Time range (auto), Tick labels (bottom axis only), and checkboxes for Scroll (checked), Data markers (unchecked), and Legends (unchecked). Sampling and Decimation sections are also present. Buttons at the bottom include OK, Cancel, Help, and Apply.

Number of axes specified as 2.

General tab selected.

**MotoTron** **freescale**

70

## Motor Model

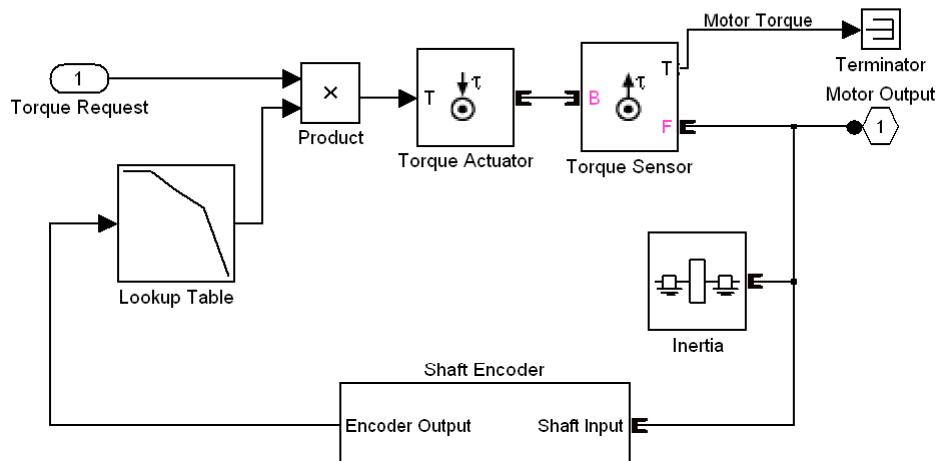
- We must now add a torque sensor.
- Find a part called **Torque Sensor** located in the **Simscape / SimDriveline / Sensors & Actuators** library.
- Use a terminator block as shown (**Simulink / Commonly Used Blocks** library) to terminate the signal.
- Label the Signal as “Motor Torque”

**MotoTron** **freescale**



71

## Motor Model



**MotoTron**

**The MathWorks**

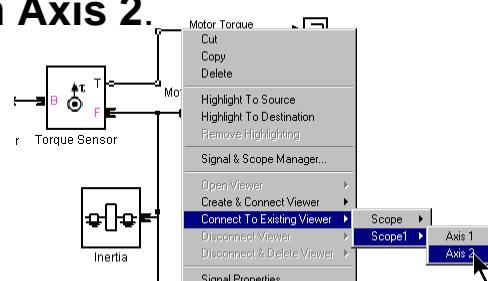
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

72

## Motor Model

- We now want to add this trace to Axis 2 of Scope1.
- Right-click on the Motor Torque signal line and select **Connect to Existing Viewer**, **Scope1**, and then **Axis 2**.

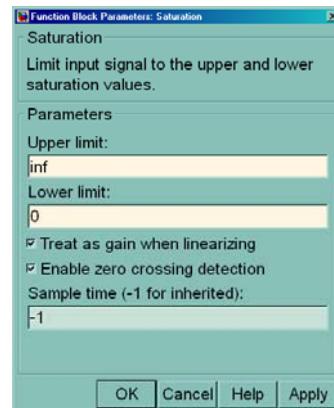




73

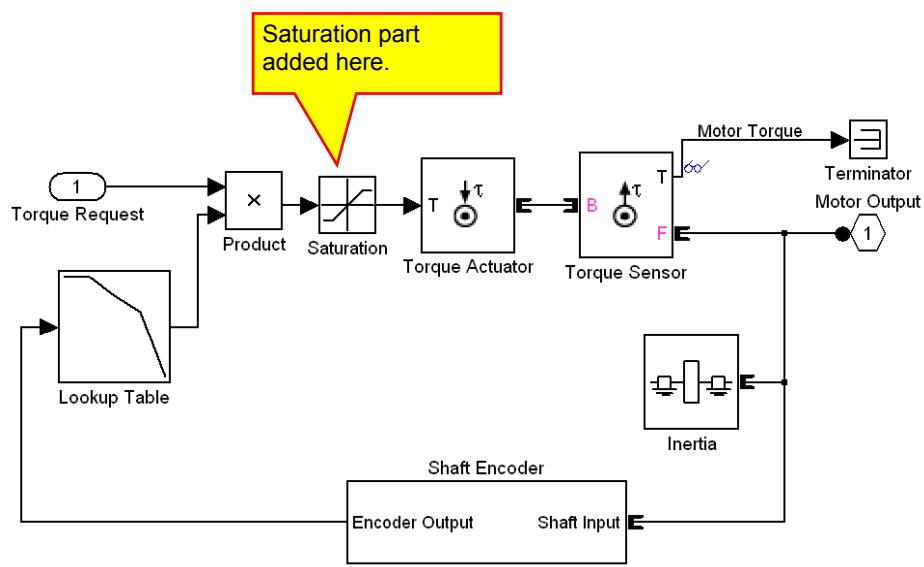
## Motor Model

- One last improvement we will make is to guarantee that the motor cannot output a negative torque. (Because our DC motor controller does not have a brake.)
- Add a saturation part as shown and set its limits to 0 and inf.



74

## Motor Model





75

## Motor Model

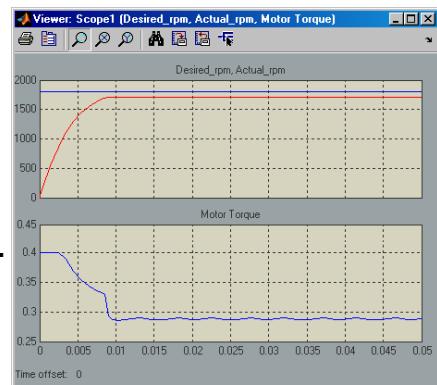
- We now have a slightly better motor model than before.
- This motor follows a torque curve and cannot output a negative torque.
- If we had more time, we would continue to make improvements to the model, but not here.
- We will now test the model to make sure the improvements we have made work as expected.



76

## Motor Testing

- When we run a simulation, we should see the torque fall off as the motor speeds up.
- Run the following tests.
- Compare to previous results.
- See if our model behaves as expected.
- For all tests, set the feedback gain to 0.01.





77

## Verification – Motor IC

- Test 1 - Set IC to 0 rad/s, all else constant
  - Old Model rpm starts below, ramps up
  - New Model
- Test 2 – Set IC to 250 rad/s, all else constant
  - Old Model rpm starts above, ramps down
  - New Model
- Make sure you observe the torque and see that the motor behaves correctly.
- Observe that the motor torque is zero if the motor speed is too high.



78

## Verification – Motor Gain

- For the next tests, set the IC at 0 rad/s, the desired rpm at 1800





79

## Lecture 3 Demo 2

### Verification – Feedback Gain

- |   | Old Model | New Model |
|---|-----------|-----------|
| • Test 4 - Set controller gain to 0.001 | _____     | _____     |
| – Steady state error (rpm)              | _____     | _____     |
| – Torque Signal                         | _____     | _____     |
| – Steady state stability                | _____     | _____     |
| • Test 5 – Set controller gain to 0.01  | _____     | _____     |
| – Steady state error (rpm)              | _____     | _____     |
| – Torque Signal                         | _____     | _____     |
| – Steady state stability                | _____     | _____     |
| • Test 6 – Set controller gain to 0.1   | _____     | _____     |
| – Steady state error (rpm)              | _____     | _____     |
| – Torque Signal                         | _____     | _____     |
| – Steady state stability                | _____     | _____     |
| • Test 7 – Set controller gain to 1     | _____     | _____     |
| – Steady state error (rpm)              | _____     | _____     |
| – Torque Signal                         | _____     | _____     |
| – Steady state stability                | _____     | _____     |

Demo \_\_\_\_\_

80

## Verification - Motor

- Our model seems to be operating as expected (or else do not continue)
  - When we observe the torque signal, it is easy to see that the system oscillates at high gain!
- Let's move on to the generator
  - Version control
  - Save it.
  - Save it again as Lecture3\_Model3.mdl



The MathWorks





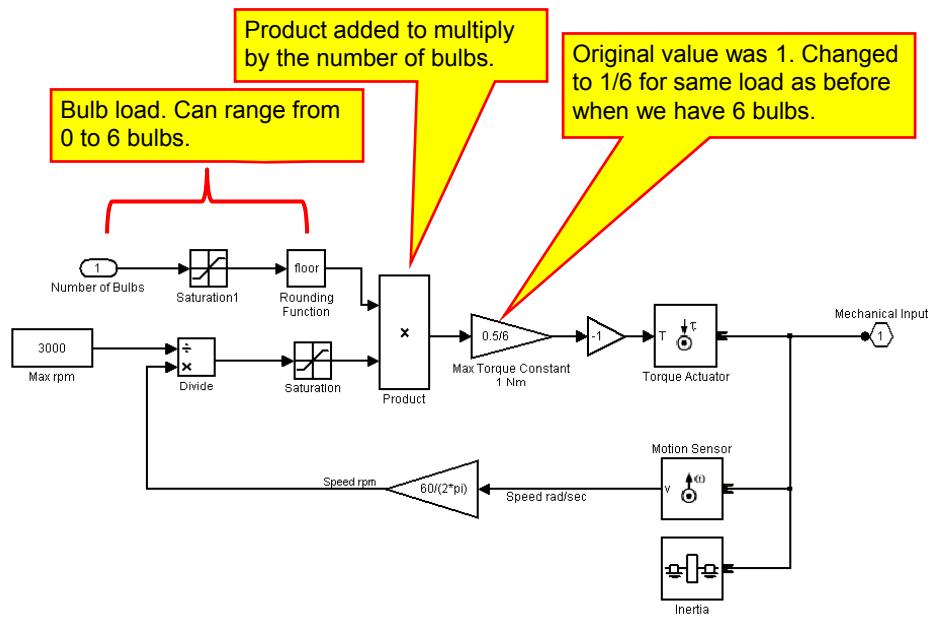
81

## Improved Model - Generator

- Our previous generator model applied a load torque that varied linearly with speed.
- The max load was equivalent to all of the light bulbs being turned on.
- We could never vary the load by changing the number of light bulbs.
- We will modify the load torque by multiplying the torque constant by the number of bulbs.

82

## Improved Generator



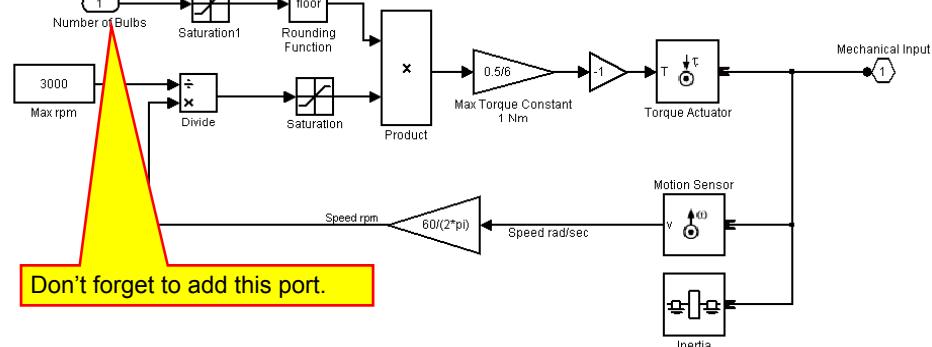


83

## Improved Generator

Saturation part with limits set from 0 to 6.

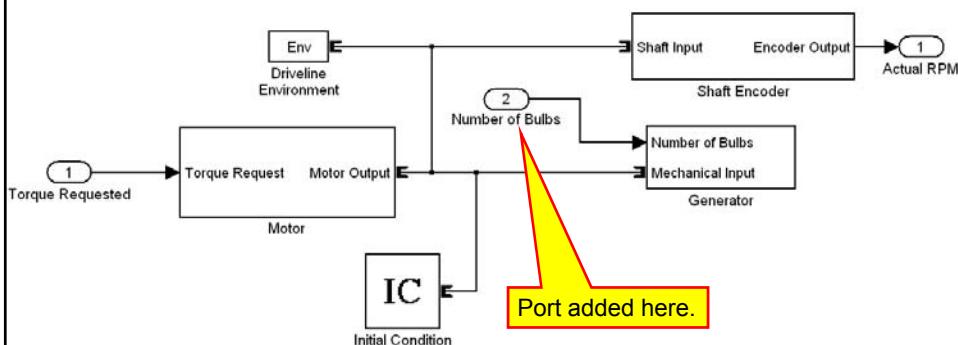
Part added so that we can only have integer values of the number of bulbs. Part name **Rounding Function**. Library **Simulink / Math Operations**.



84

## Improved Generator

- Since we added a new Simulink port to the generator model, we will need to add a port to the plant model as well.

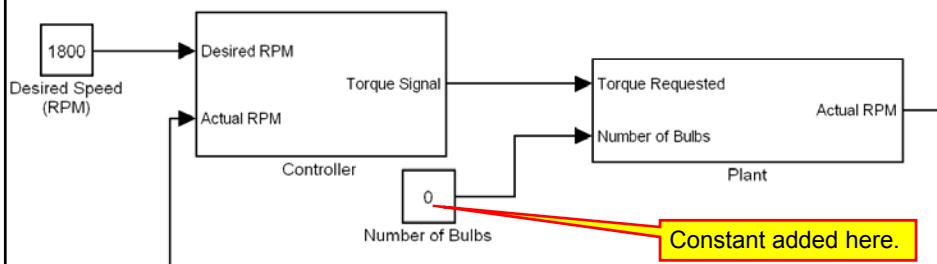




85

## Improved Generator

- At the top level we need to add a constant that will be used to specify the number of bulbs.



86

## Verification - Generator

- We have made a simple change to our model – let's make sure it operates as we expected
- Run the following experiments and ensure that we get the same results as with our previous motor improvement
- Set the light bulb load to 6 bulbs. This should be the same as our previous experiments.
- Set the desired rpm is 1800 and the proportional gain to 0.01.





87

## Lecture 3 Demo 3

### Verification – Generator IC

- Test 1 - Set IC to 0 rad/s, all else constant
  - Old Model \_\_\_\_\_
  - New Model \_\_\_\_\_
- Test 2 – Set IC to 250 rad/s, all else constant
  - Old Model \_\_\_\_\_
  - New Model \_\_\_\_\_
- These results should match our previous simulations since 6 bulbs is equivalent to what we did previously.

Demo\_\_\_\_\_

88

### Verification – Feedback Gain

- For the next tests, hold the IC at 0 rad/s, the desired rpm at 1800, and the load at 6 bulbs – we shall verify the controller gain.
- The results should be the same as our previous simulations.



The MathWorks

freescale<sup>®</sup>  
semiconductorROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



89

## Lecture 3 Demo 4

### Verification – Feedback Gain

- |   | Old Model         | New Model         |
|---|-------------------|-------------------|
| • Test 4 - Set controller gain to 0.001 | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state error (rpm)              | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Torque Signal                         | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state stability                | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| • Test 5 – Set controller gain to 0.01  | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state error (rpm)              | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Torque Signal                         | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state stability                | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| • Test 6 – Set controller gain to 0.1   | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state error (rpm)              | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Torque Signal                         | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state stability                | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| • Test 7 – Set controller gain to 1     | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state error (rpm)              | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Torque Signal                         | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |
| – Steady state stability                | <hr/> <hr/> <hr/> | <hr/> <hr/> <hr/> |

Demo \_\_\_\_\_

## Verification

90

- Our generator model works as expected for 6 bulbs. (If not, fix it.)
- Since we have a variable load, we can now observe the affect of changing the load on the simulation.
- Next, we will run some tests while varying the load.
- For all tests set the gain to 0.01, the desired rpm to 1800, and the initial condition for the rpm to 0.



The MathWorks





91

## Lecture 3 Demo 5

### Verification – Feedback Gain

- Test 1 - Set number of bulbs to 0
  - Steady state error (rpm)
  - Torque Signal
- Test 2 - Set number of bulbs to 1
  - Steady state error (rpm)
  - Torque Signal
- Test 3 - Set number of bulbs to 2
  - Steady state error (rpm)
  - Torque Signal
- Test 4 - Set number of bulbs to 3
  - Steady state error (rpm)
  - Torque Signal
- Test 5 - Set number of bulbs to 4
  - Steady state error (rpm)
  - Torque Signal

---



---



---



---



---



---



---



---



---



---

Demo \_\_\_\_\_

92

## Lecture 3 Demo 6

### Verification – Feedback Gain

- Test 1 - Set number of bulbs to 5
  - Steady state error (rpm)
  - Torque Signal
- Test 2 - Set number of bulbs to 6
  - Steady state error (rpm)
  - Torque Signal
- The results should show that the steady state error increases as we increase the number of bulbs.
- With no load (0 bulbs) the error should be very close to zero.

---



---



---



---



---

Demo \_\_\_\_\_



93

## Improved Model - Generator

- Right now our generator is pretty much a black box that creates a load torque that is proportional to motor rpm.
- Lets try to do a similar model that relates a bit more to the physical components in our system.
  - The generator produces a voltage that is proportional to the generator speed.
  - This voltage is applied to the light bulbs, and the bulbs draw current proportional to the voltage applied across them (Ohm's Law).
  - The current is supplied by the generator.
  - To supply the current, the generator produces a torque that opposes its direction of motion.



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Improved Generator Model

94

- This model is actually a combination of many of the previous models we have already built.
- Before continuing:
  - Save your model.
  - Save your model again as [Lecture3\\_Model4.mdl](#)



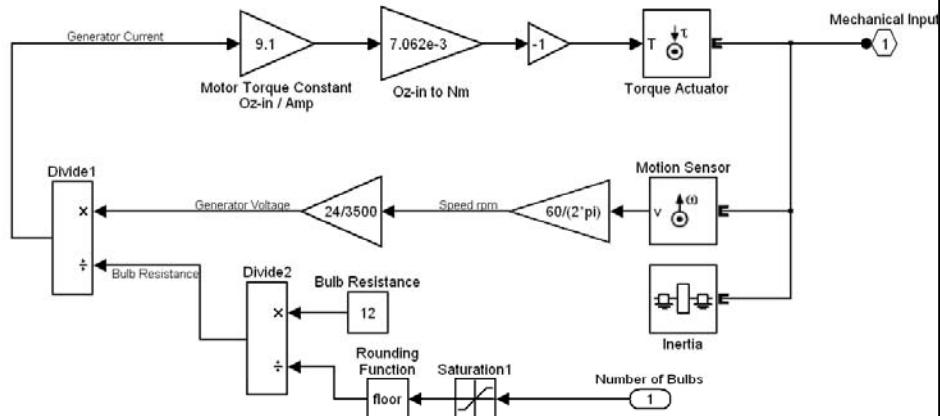
The MathWorks

freescale<sup>®</sup>  
semiconductor



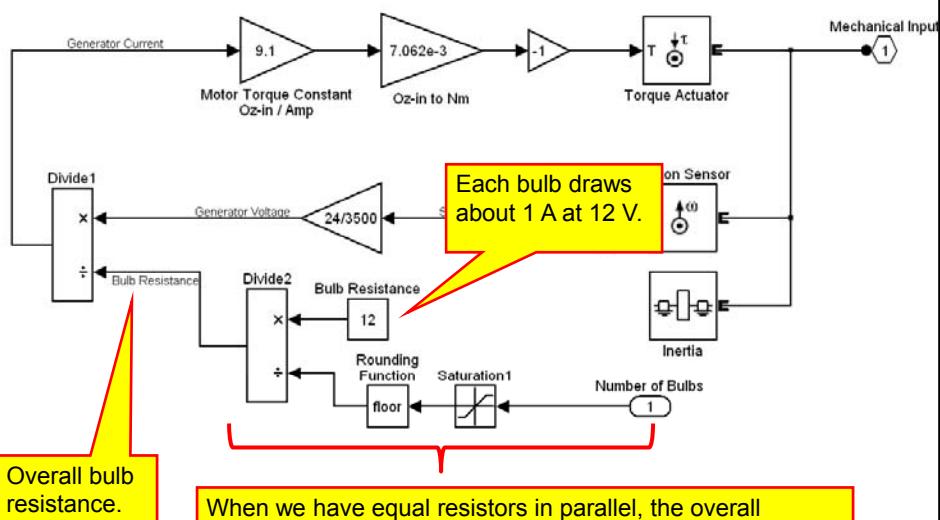
95

## Improved Generator Model

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

96

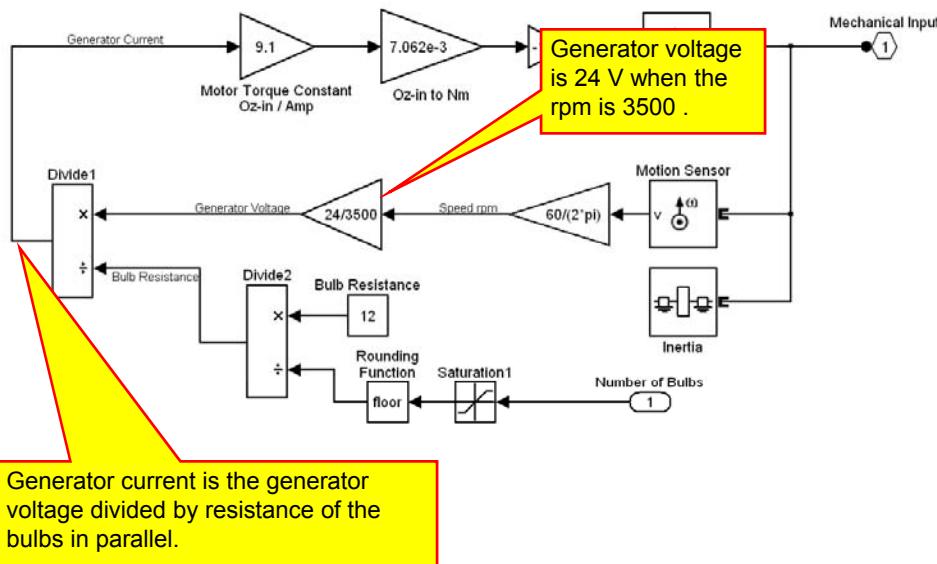
## Improved Generator Model





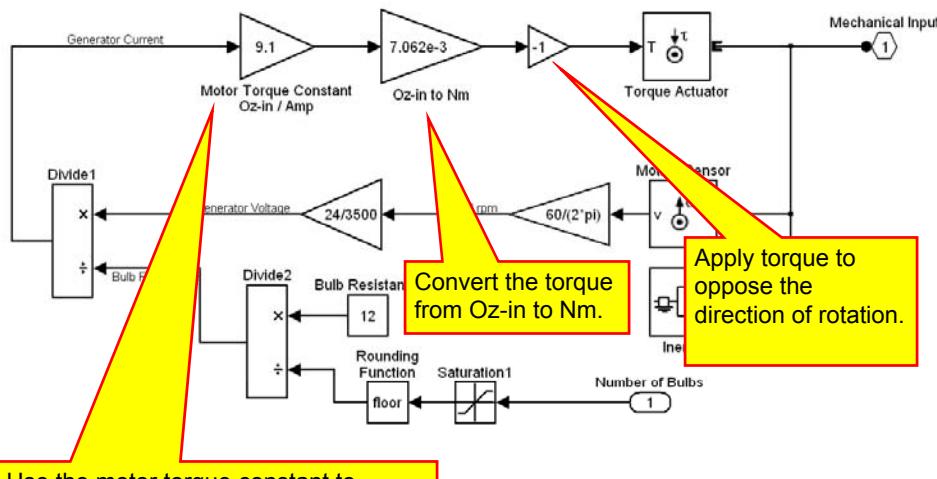
97

## Improved Generator Model



## Improved Generator Model

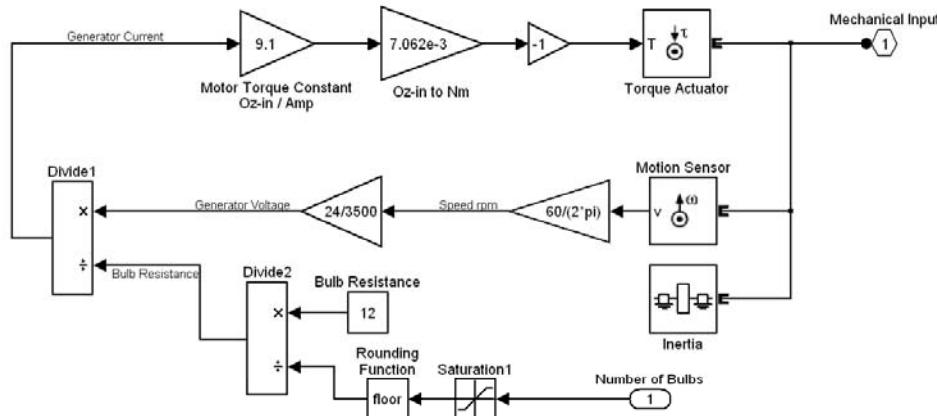
98





99

## Improved Generator Model



This generator model has a bad flaw. Can you find it and fix it.  
You might not notice until you run a few simulations.

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

100

## Improved Load - Verification

- To keep things moving along, we'll do just a couple quick verifications
- For all tests set the gain to 0.01, the desired rpm to 1800, and the initial condition for the rpm to 0.
- You should notice that this model produces a higher torque load than previously. For 6 bulbs, there is a fairly large error. (We do see this in the lab.)

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



101

## Lecture 3 Demo 7

### Verification – Feedback Gain

- Test 1 - Set number of bulbs to 0
  - Steady state error (rpm)
  - Torque Signal
- Test 2 - Set number of bulbs to 1
  - Steady state error (rpm)
  - Torque Signal
- Test 3 - Set number of bulbs to 2
  - Steady state error (rpm)
  - Torque Signal
- Test 4 - Set number of bulbs to 3
  - Steady state error (rpm)
  - Torque Signal
- Test 5 - Set number of bulbs to 4
  - Steady state error (rpm)
  - Torque Signal

---



---



---



---



---



---



---



---



---



---

Demo \_\_\_\_\_

102

## Lecture 3 Demo 8

### Verification – Feedback Gain

- Test 1 - Set number of bulbs to 5
  - Steady state error (rpm)
  - Torque Signal
- Test 2 - Set number of bulbs to 6
  - Steady state error (rpm)
  - Torque Signal
- The results should show that the steady state error increases as we increase the number of bulbs.
- With no load (0 bulbs) the error should be very close to zero.
- With 6 bulbs, the generator loads down the motor, and then motor cannot achieve the desired speed.

---



---



---



---



---

Demo \_\_\_\_\_



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

103

## Verification - Generator

- Conclusions

- We have slowly improved the models to make them more physically realistic.
- Simple models can produce good results and help us understand our physical system and help control strategy development.



The MathWorks





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

105

## Lecture 3 Problem 2

- Capacitor Car – Description on next page.

Grade \_\_\_\_\_



# MBSD1 - Introduction to Model-Based System Design

## Lecture 3 Problem 2

One of the systems we will be modeling in this course is a capacitor-powered electric car. (We may also look at a spring powered car as well.) In this homework, we will begin developing the models for the capacitor and the electric motor. Create models for the following components:

### Capacitor:

The amount of charge that a capacitor stores in coulombs is  $q$ . The equation that governs the amount of charge stored by the capacitor is  $q = CV$ , where  $C$  is the capacitance in Farads, and  $V$  is the capacitor voltage. We add or remove charge from the capacitor with electric current. If the current is positive, we will add charge to the capacitor and increase the stored charge and the capacitor voltage. If the current is negative, we remove charge and decrease the stored charge and capacitor voltage. The charge added or removed,  $\Delta q$ , is equal to the integral of the current,  $\Delta q = \int I(t) dt$ . Your model should have the following parameters:

Inputs: Capacitor current.

Outputs: Capacitor voltage.

Parameters: Initial capacitor voltage, capacitance in farads. (Constants)

Test your model by showing that the capacitor obeys the equation  $I_C = C \frac{dV}{dt}$  using a sine wave input current of 1 amp amplitude and a 1 Farad capacitor.

### Electric Motor:

A lossless DC electric motor obeys the following equations.

- The torque is equal to the torque constant  $K$  times the motor current,  $\tau = KI$  where the torque is in Newton-meters, the current is in amps, and the torque constant has units of Nm/amp.
- The motor back emf (voltage) is equal to the motor speed  $\omega$  in rad/sec times the back emf constant  $K$  which has units volts/(rad/sec),  $V_{emf} = K\omega$ .

Note that if the value of  $K$  has the same numerical value in both equations, we obtain the equation for a lossless electromechanical energy converter,  $V_{emf}I = \tau\omega$ , where electrical power equals mechanical power. To calculate the motor current, we take the applied voltage minus the back emf, and divide it by the motor winding resistance,  $I = \frac{V_{app} - V_{emf}}{R}$ .

Create a motor model with the following parameters:

Input: Applied motor voltage in volts.

Output: Motor torque in Nm (SimDriveLine port)

Output: Motor current in amps. (Simulink signal.)

Parameters: Torque constant and back EMF constant ( $K$ ), winding resistance, and motor inertia.

Test your motor model by applying a constant voltage of 24 V and no mechanical load (you will still need the ENV part from the SimDriveline libraries. You should observe a spike of current at startup when the motor speed is zero. As the motor speeds up, you will see the current go to zero as the motor reaches steady state. Use the following motor parameters.

$$K=0.01 \text{ V}/(\text{rad/sec}) = 0.01 \text{ Nm/Amp.}$$



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

$$\text{Inertia} = 1.5 \times 10^{-3} \text{ kg}\cdot\text{m}^2$$

Winding resistance = 2 Ohms.

Applied Voltage = 24 V

You should see the current spike up to 12 Amps and the motor should speed up to

**System test:** Hook up your capacitor model to your motor model. Give the capacitor an initial voltage of 12 V. Use the SimDriveline clutch to apply a constant frictional force to the motor. Run the system until the capacitor discharges to zero. Show that the energy stored in the capacitor equals the energy dissipated by the friction clutch plus the energy dissipated by the series resistance. Use the controllable friction clutch in the SimDriveLine libraries. Specify a clutch pressure of 1 and use the settings below for the clutch:

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



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



## Introduction to Model-Based Systems Design

### Lecture 3A: Step Response, Data Collection, and xPC



## System Observations

2

- In Lecture 3, we noticed that the motor had a 0% to 99% step response time of about 6 ms.
- This will cause us difficulties in the future.
- Problem 1:
  - It will require a small integration step size during Simulink simulations. You may have noticed this already.
  - A small step size will take more time to complete a simulation.
  - For us, this is not that great of a limitation because our model is fairly simple.





3

## System Observations

- Problem 2:

- When we simulate a fast system in real time, a small fixed integration step size is required.
- Even though we have a simple model, it will require such a small fixed step size that our real-time targets are not fast enough to run the model.



4

## System Observations

- Problem 3:

- The system responds so quickly to a step change, that we can not observe system behavior visually.
- This is mostly an educational restriction because we want you to see overshoot, undershoot, and ringing as the controller gains are changed.
- If the system responds in 6 ms, students will not be able to see the system's dynamic response. It would be possible to see the response with a scope, but we want to see the response on our real-time system.





5

## System Observations

- Problem 4:
  - It will be difficult to control a plant that responds this quickly with the MPC555x microcontrollers available in the lab.
  - Our controllers are limited to a minimum fixed step size of 1 ms.
  - ➔ It is difficult to control a plant that has a time constant of about 2 ms with a controller that has a response time of 1 ms.
  - This is our greatest limitation.



6

## System Limitations

- If the motors do indeed respond as fast as our simulations indicate, the controller hardware available in the lab will not be fast enough to the feedback control method.
- We will either need to modify the plant or purchase new and more expensive microcontroller targets.
- Both of these solutions are expensive and time consuming.





7

## System Limitations

- Before we go to the expense of modifying the plant or purchasing new equipment, we will verify that the motors do indeed have this fast of a response.
- We will measure the step response of the motors.
- In lecture 3, we presented a slide that showed the step response measured using an oscilloscope.
- We never actually set up and ran this experiment. Some one just told us they measured the step response and the results were presented.



8

## System Limitations

- Here we will set up an experiment to:
  - Send the motor a step input.
  - Measure the system step response using a real-time computer system running xPC.
  - Write a Matlab post processing function to calculate the time constant from the measured data.
  - Determine a course of action to mitigate the problem.





9

## Lecture Outline

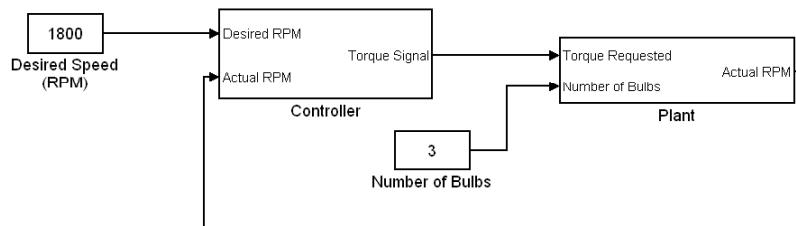
- Review system step response and observe problems in the model.
- Setup a real-time target using xpceexplr
- Learn how to run xPC on a real-time target.
- Setup A/D and D/A blocks using xPC on a Speedgoat real-time target.
- Run the Step response and collect data.
- Transfer the data to the host PC and analyze the data.



10

## Model Review

- We will start with the last model generated in lecture 3. This has been provided to you as Lecture3A\_Model0.mdl.
- Save the model as Lecture3A\_Model1.mdl.
- The model is shown below:

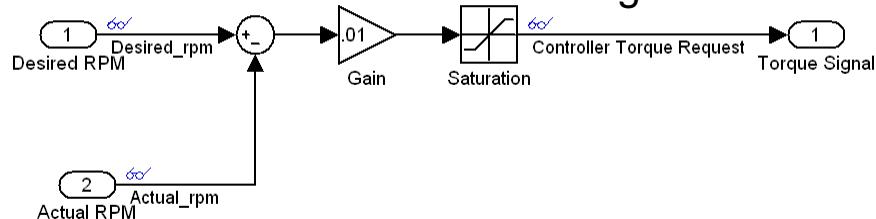




## Model Review

11

- The controller has a feedback gain of 0.01:

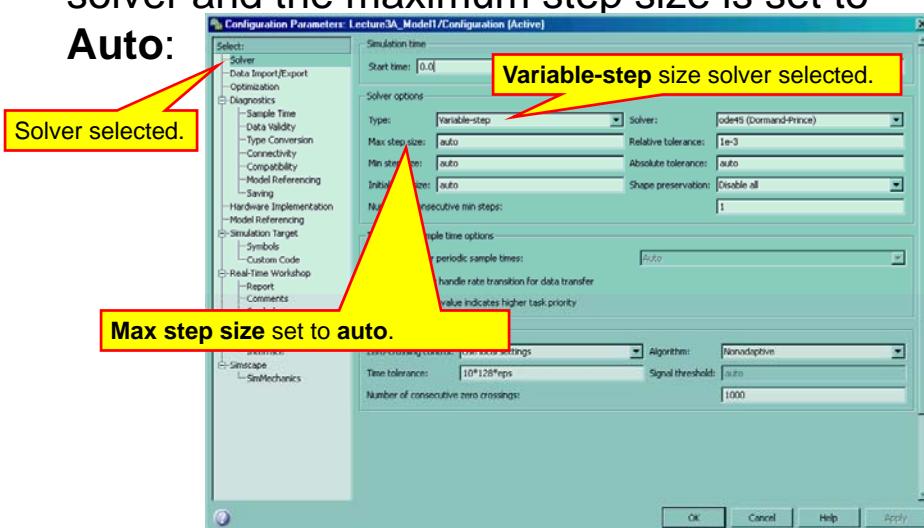


- Select **Simulation** and then **Configuration Parameters** from the Simulink menus to view the simulation setup:

## Model Review

12

- We see that we are using a variable step solver and the maximum step size is set to **Auto**:





13

## Solvers

- A variable step solver uses a variable size time step to perform the integration and solve the differential equations.
- When the system response changes slowly with time, a variable-step size solver increases the integration step size so that fewer integration steps are required to complete the simulation.
  - This requires less computation and allows the simulation to run faster.
- When the system encounters an area where things change quickly, the solver reduces the time step so that faster dynamics are simulated accurately.



14

## Solvers

- Our motor-generator system is a good application for a variable-step solver.
- Usually we change the speed or number of bulbs infrequently.
- During the times when the speed does not change or the bulb load is not changed, the solver can take a large time step.
- When we change the requested speed or the bulb load, we usually ask for a step change. → When we change the number of bulbs, we introduce a step change in the load. We cannot gradually increase the number of bulbs from say 1 to 2 because we must turn on the bulbs in discrete amounts. (We cannot turn on 1.5 bulbs.)
- A step change in the request speed or bulb load introduces a step change into the system. The variable step solver will reduce the time step to accurately simulate the system.





15

## Solvers

- We also note that the simulation parameters specify that the **max time step** is set to **auto**.
- This means the solver can increase the time step to as large a value as it desires.
- We will see later that the discrete intervals at which a microcontroller can change its output effectively adds phase delay into the system. This delay can cause the system to oscillate.
- The larger the simulation time step, the larger the delay.



The MathWorks

freescale<sup>®</sup>  
semiconductor

16

## Solvers

- We also note that the simulation parameters specify that the **min time step** is set to **auto**.
- This means that the solver can make the time step as small as it wants to.
- This could result in very long simulation times.
- When the step size becomes small, you will notice:
  - The simulation appears to hang at a certain spot because the time step is so small. The simulation is still running, but the time step is so small that the simulation appears to not be making any progress.
  - Simulink will stop and indicate that the minimum time step had been reached and that there is a convergence error.
- A small time step is a result of a system that changes its properties over a very short time.



The MathWorks

freescale<sup>®</sup>  
semiconductor



17

## Solvers

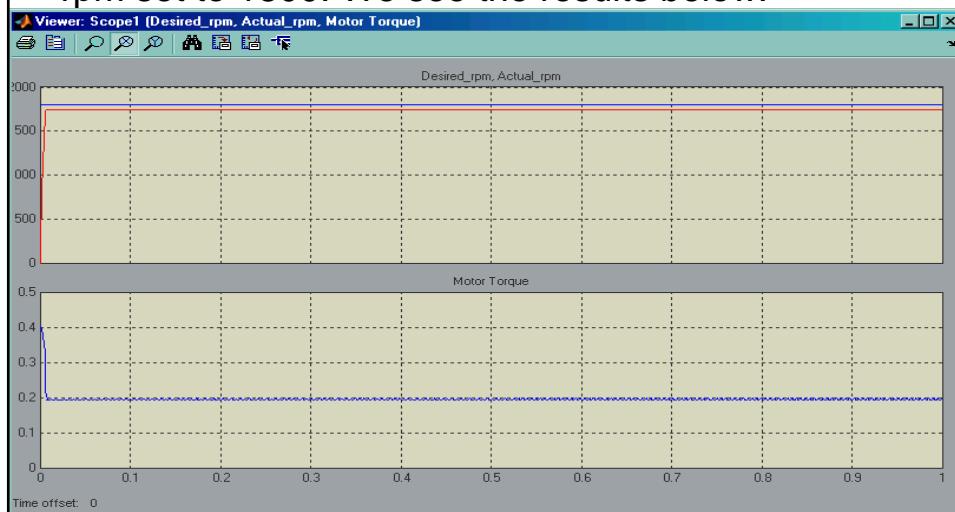
- Examples of systems that require small time steps are:
  - Clutches – during the locked or unlocked state, the system is easy to solve. However, when the system transitions between the locked and unlocked state, an extremely small time step is required to determine just when the mode switches between the locked and unlocked states.
  - Power-electronic switches – a switch goes from a high impedance state to a low impedance state in a matter of microseconds. This transition take a small time step to simulate accurately.



18

## System Simulation

- Run the simulation with the max time step set to auto, the feedback gain set to 0.01, and the desired rpm set to 1800. We see the results below:

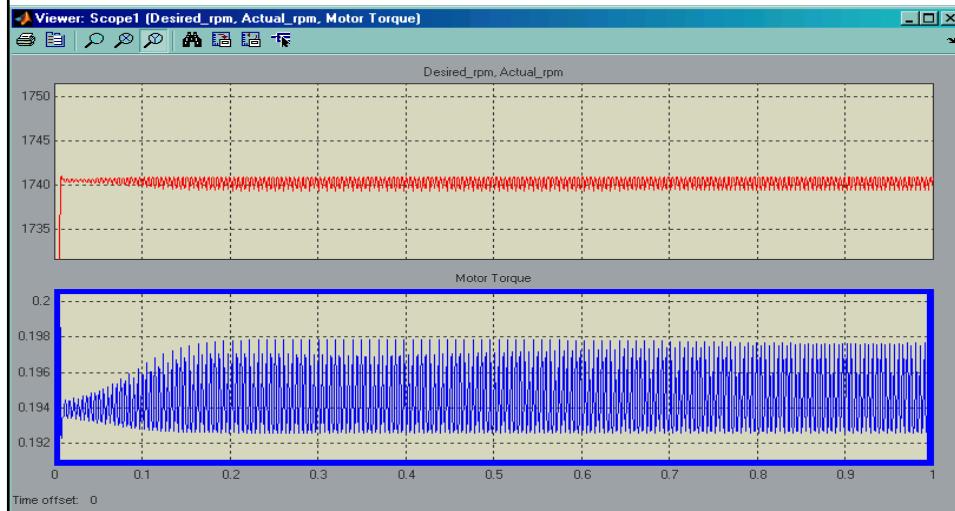




19

## System Simulation

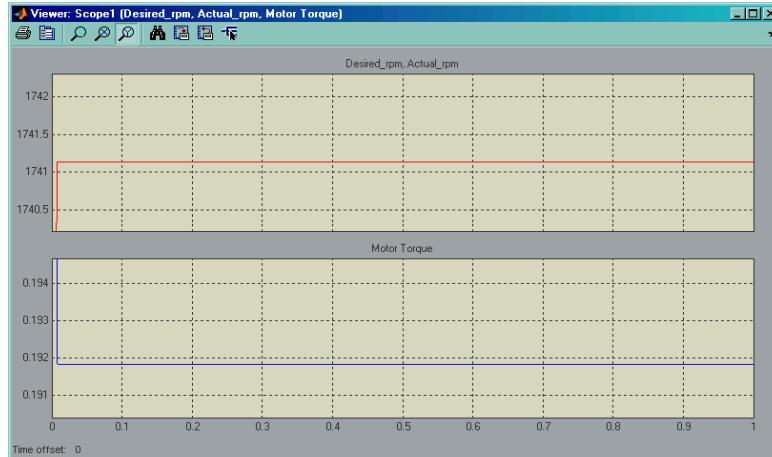
- If you zoom in on the rpm and the torque signals, you will notice an oscillation:



20

## System Simulation

- The oscillation could be a result of the feedback gain being too high, the simulation time step being too large, or a combination of the two.
- We will fix the problem by setting the max time step to 0.0001 seconds:





## System Simulation

21

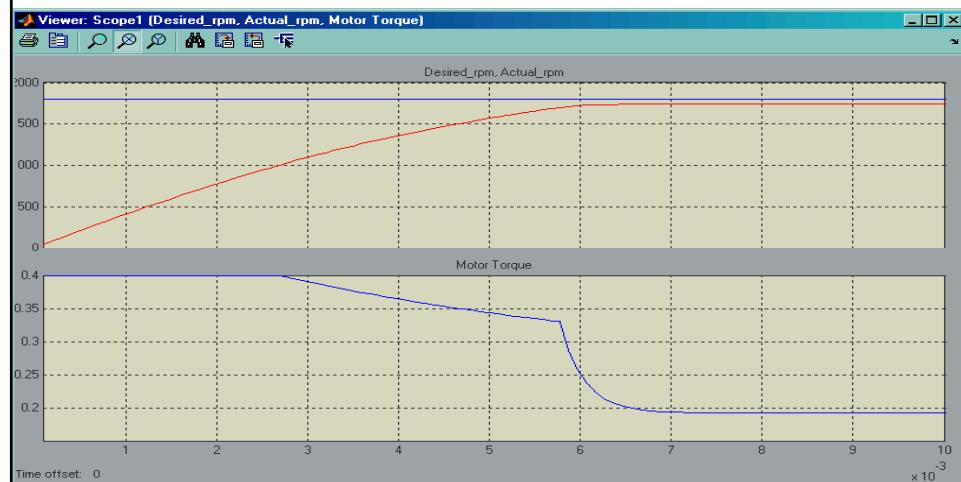
- A max simulation time step of 0.001 seconds did not eliminate the oscillation.
- A max time step of 0.0001 was needed.
- The smallest time step we can run our controller at is 1 ms, meaning that this physical system is too fast to be controlled by our microcontroller targets.



## System Simulation

22

- Zooming in at the beginning of the simulation, we see that the system step response reaches steady state in about 6 ms:





## System Response

23

- This is indeed a very fast response.
- If the motor does respond at this fast of a rate, we will need to slow the system down or choose new controller targets.
- Before we take one of these drastic steps to correct the problem, we will measure the actual step response of the motors to verify that they are indeed this fast.
- We will use xPC and a real-time target to run a step response and collect the data.



## Lecture 3A Exercise 1

24

- Use the fixed-step ode3 (Bokacki-Shampine) solver and determine the largest fixed step size that can be used to simulate the system without yielding an oscillation in the motor torque or plant rpm.
- The desired rpm should be set to 1800, number of bulbs to 3, and the feedback gain to 0.01 (same as used in the previous slides.)
- Note that when we deploy the controller on the MPC555x target, a fixed step size will be used. Thus, this is a good test to determine the required step size needed for our controller.
- Find the max step size to the nearest 100  $\mu$ s.

Demo \_\_\_\_\_





## xPC

25

- We will be using real-time target computers from Speedgoat.
- These targets are industrial strength personal computers designed to survive in rugged environments.
- They are preconfigured with xPC and have analog and digital I/O capabilities.
- In Lecture 8, we will show how to setup and use an old personal computer as a real-time computer using xPC target.
- Since these targets are preconfigured, we will just go through the steps necessary to communicate with and run models on the target.
- In Lectures 8 and 10, we will show more detail on using `xpcexplr` and target setup.



## xPC Explorer

26

- From the Matlab prompt, type the command `>>xpcexplr`
- We will use `xpcexplr` to:
  - specify settings to allow us to communicate with the target.
  - Specify a compiler to allow us to build models.





## Visual Studio Compiler

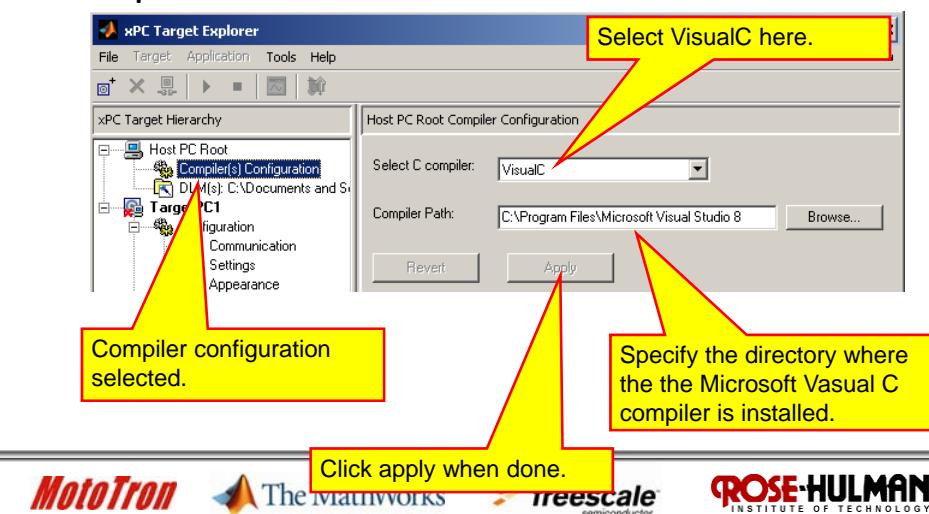
27

- First, we will select the compiler. We will assume that you are using the Microsoft Visual Studio 2005.
- We will assume that you have already installed the Microsoft compiler and that it is installed in directory C:\Program Files\Microsoft Visual Studio 8.

## Visual Studio Compiler Setup

28

- The compiler is specified using the xPC Explorer.

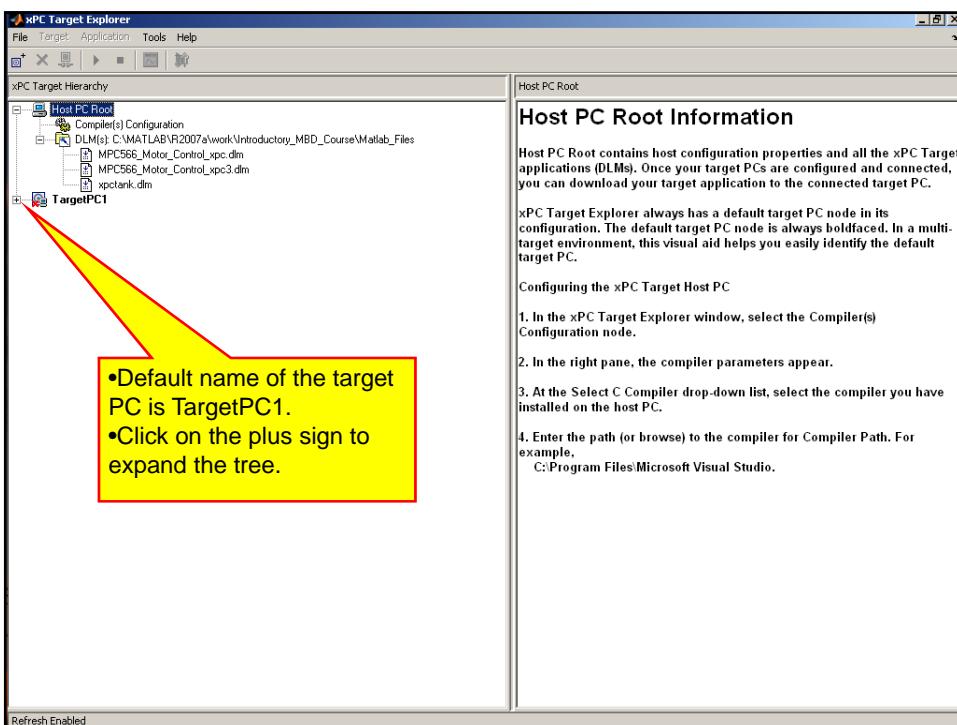
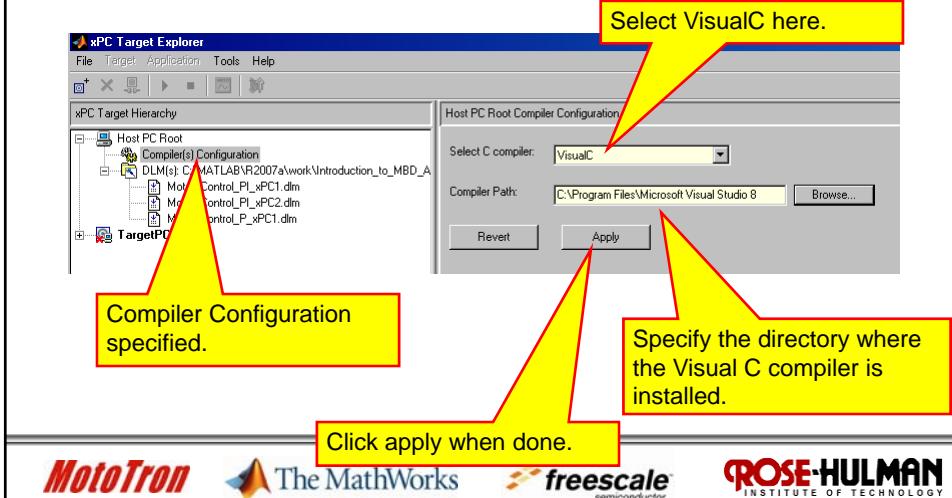




29

## Visual Studio Compiler Setup

- If you have Microsoft Visual Studio 2005 installed, your settings will be as shown:





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

31

- The first thing we will do is specify the communication options for our target.
- Click on the **Communication** selection in the tree.

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

32

- We will look briefly at some of these options.

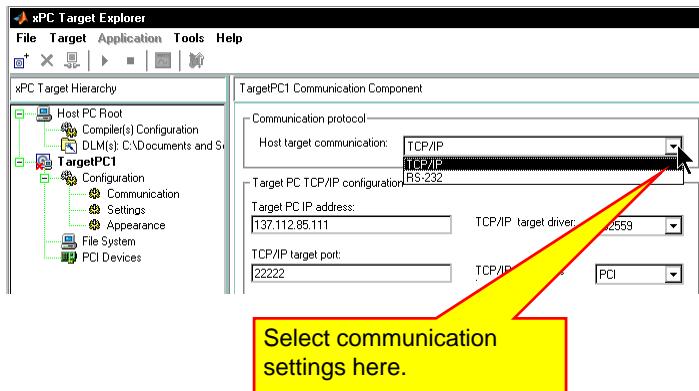
**MotoTron** **The MathWorks** **freescale** **ROSE-HULMAN** INSTITUTE OF TECHNOLOGY



## xPC Setup

33

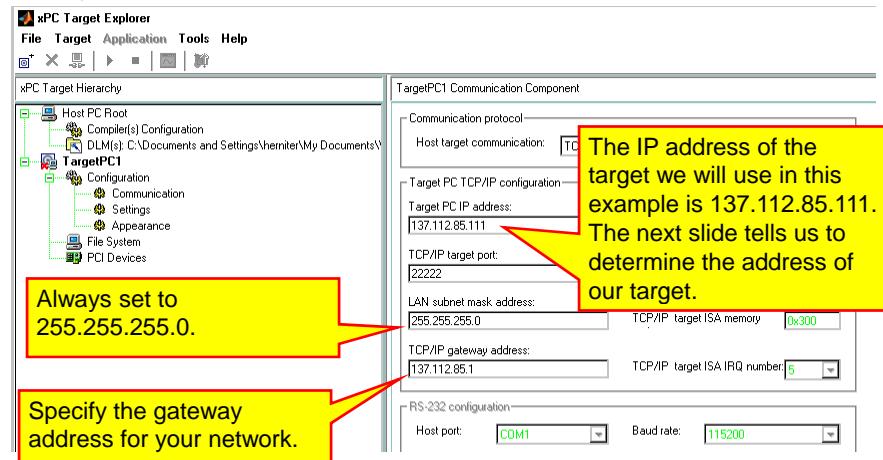
- We will be communicating with our target through the network.
- Select the **TCP/IP** connection



## xPC Setup

34

- Specify the IP addresses for your target system.





35

## xPC Target

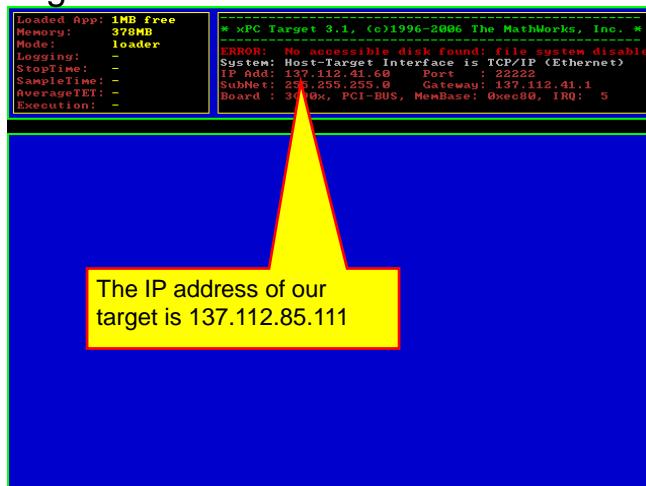
- We will be using targets preconfigured with xPC.
- These targets have an IP address specified in their setup file/configuration.
- When you boot up this target, a screen will appear showing you the IP address.
- This is a static address. The address is specified in xpceexplr in the configuration section we are currently working on.
- When you specify an IP address in xpceexplr, you need to make sure that no other computer is using that IP address.



36

## xPC Target

- When you boot up the xPC target, if a monitor is connected, it should display status information about the target, including its IP address.

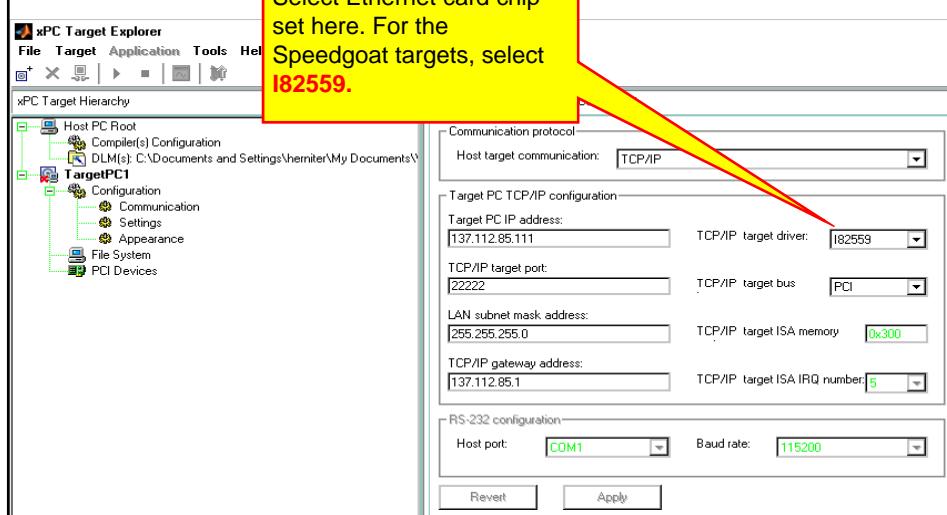




## xPC Setup

37

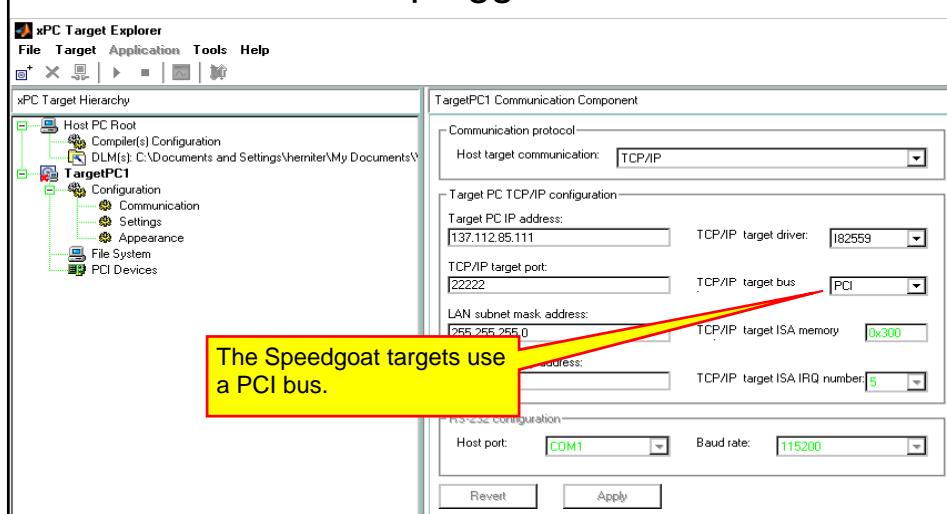
- Select the driver/chip set for your Ethernet card:



## xPC Setup

38

- Select the type of bus into which the Ethernet card is plugged:

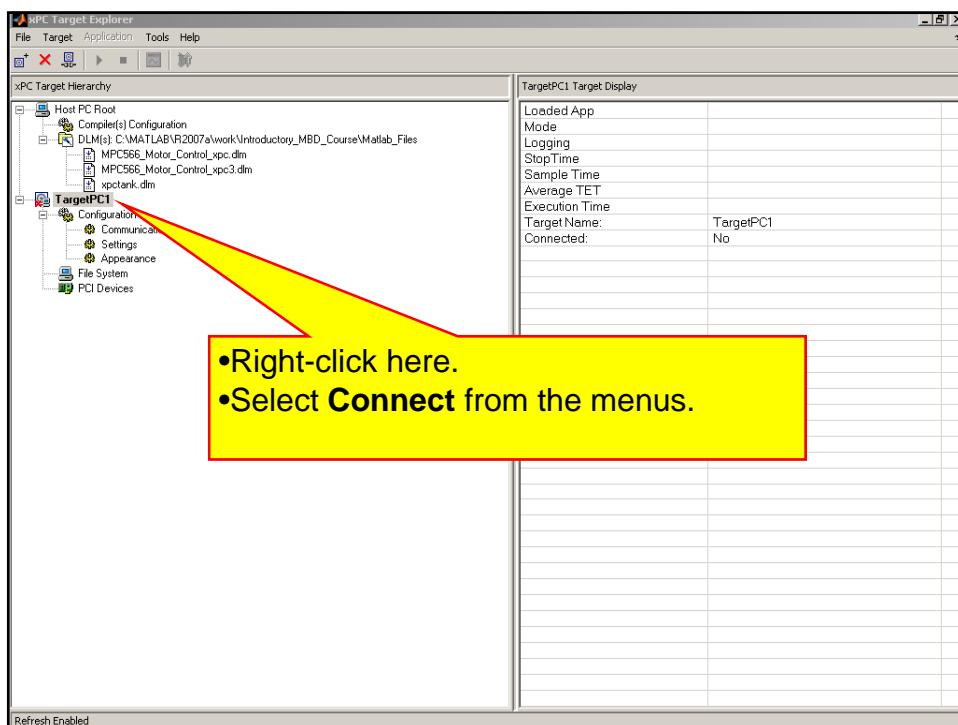
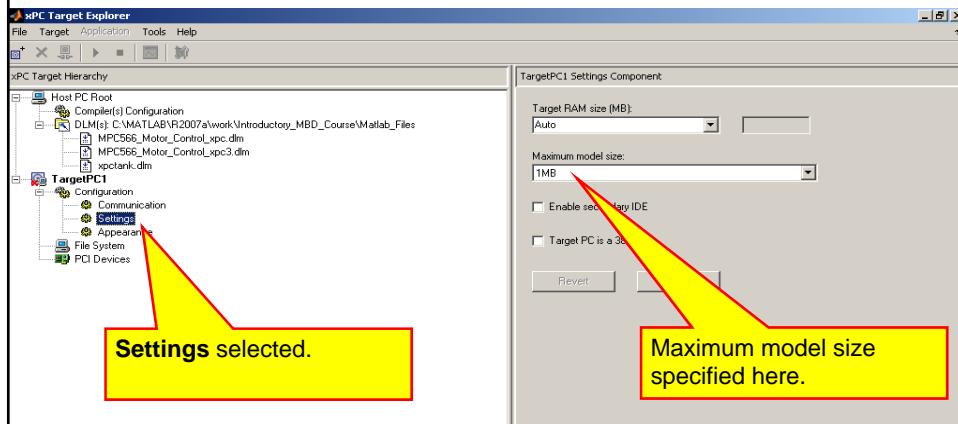




## xPC Setup

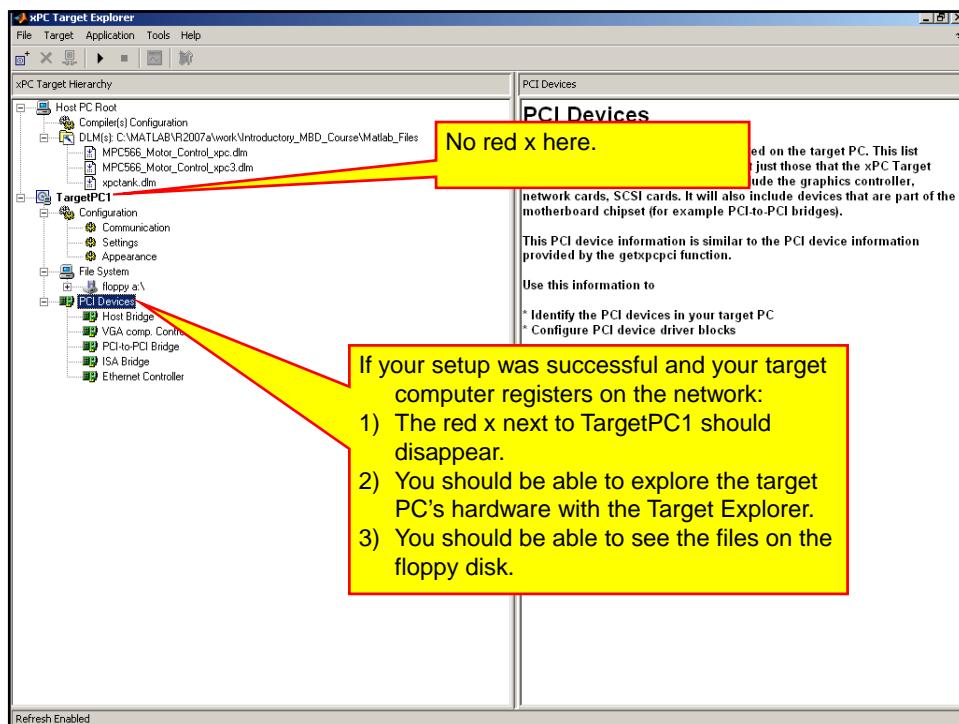
39

- Next, select **Settings**.
- Since our model is small, we will use the default settings for model size:





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



## xPC Setup

42

- Communication with our target is now complete.
- We are now ready to create a model to run on the xPC target computer.



The MathWorks





43

## Step Response

- We will now use the Speedgoat target to run a test on the motor and collect data.
- To generate a step, we will use an analog output channel to generate a 0 to 5 V step.
  - A 5 V input to the motor driver will tell the driver to run the motor at full power.
- We will then measure the motor rpm using an analog input channel from the Speedgoat target.
- We will write the data to a file on the target machine.
- We will then run a Matlab script on the host computer to read the data file, perform some data conversion, and then determine the step response.



The MathWorks

freescale<sup>®</sup>  
semiconductor

44

## Speedgoat Target

- The Speedgoat real-time target has a PMC730 Multi-function I/O target installed.
- This card has the following facilities:
  - 32 Analog input channels with 16-bit resolution.
    - Can be configured as 32 single-ended input channels (single input referenced to ground).
    - Can be configured as 16 differential input channels (signal difference between the two inputs).
  - 8 Single-ended analog output channels with 16-bit resolution.



The MathWorks

freescale<sup>®</sup>  
semiconductor



45

## PMC730 Multi-function I/O Card

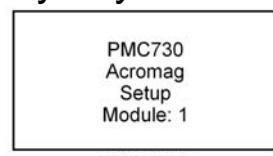
- 16 Digital I/O Channels.
  - Input/output function configured in groups of 8-channels
- One 32-bit counter/timer
- A data sheet for the PMC730 is available on-line in the DataSheets section of the MBSD1 website.



46

## Speed Goat target.

- Create a new Simulink model and name it Lecture3A\_xPC1.mdl.
- We must first configure the hardware on the PCM730 board.
- Place a Simulink block named **PMC730 5** located in the **xPC Target: Speedgoat I/O Driver Library / IO101** library in your model.





47

## PMC730 Acromag Setup

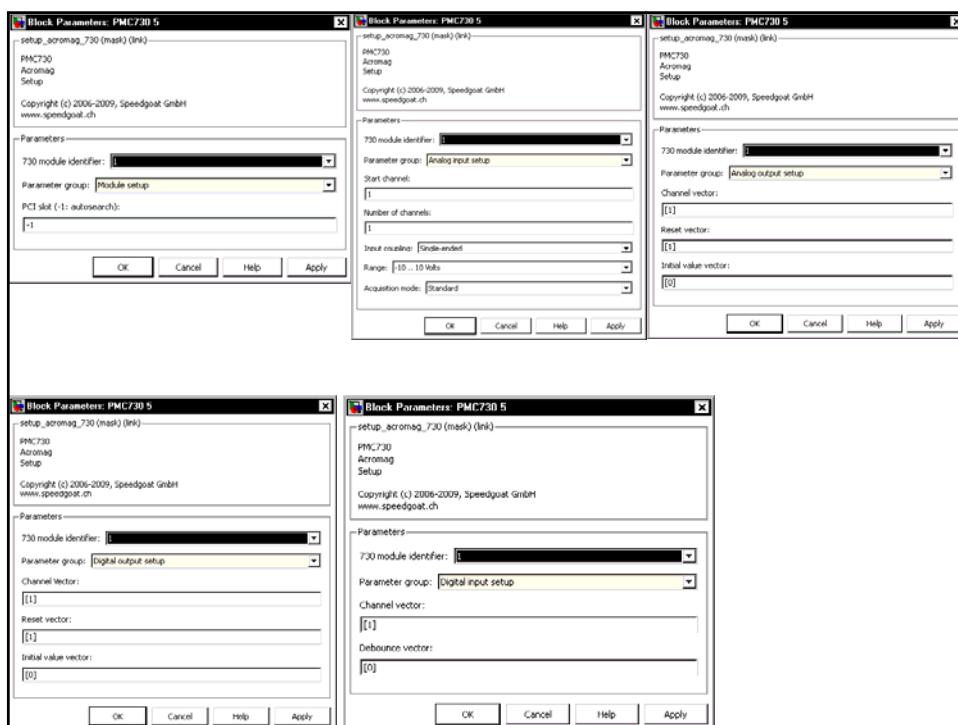
- This single block is used to setup all functions of the PMC730 card.
- Only 1 copy of this block is needed to set up the analog I/O and digital I/O facilities of the card.
- Double-click on the block to open it.
- Note that the block will appear differently depending on the parameter group selected.
- The screen captures on the next page show the same block with different Parameter groups selected.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY





49

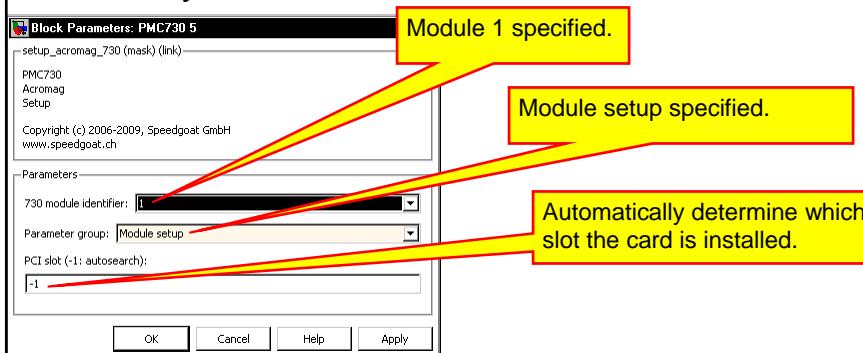
## PMC730 5 Acromag setup block

- We notice that the module identifier in all of the screen captures on the previous slide was set to 1.
- If a Speedgoat target has more than one PMC730 target, we can use this card to address each PMC730 individually.
- Our targets have a single PMC730 installed, so the module identifier will be specified as 1.
- We will now set the parameters on this block to have a single analog input and a single analog output.

50

## PMC730 5 Acromag setup block

- We first must specify the Module setup.
- Since we only have a single PMC730 module, we can specify that xPC automatically determine in which slot the PMC730 is installed. If we had more than one PMC730 installed, we would specify the slot manually:

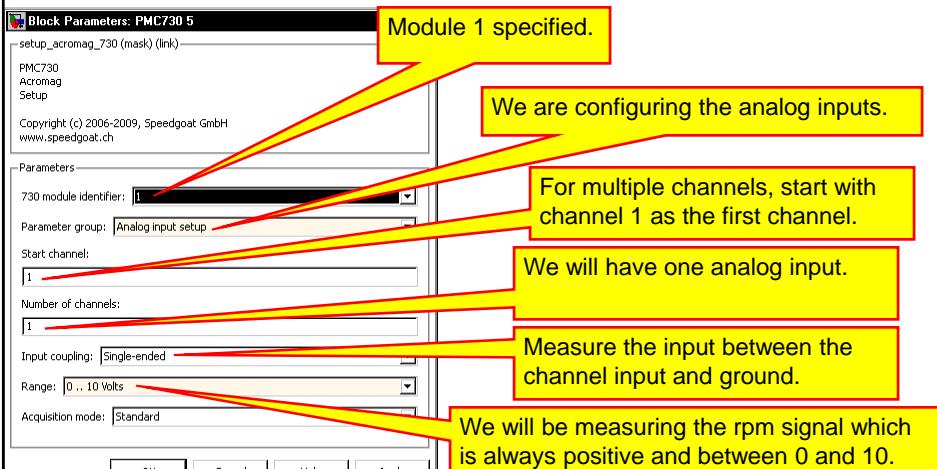




51

## PMC730 5 Acromag setup block

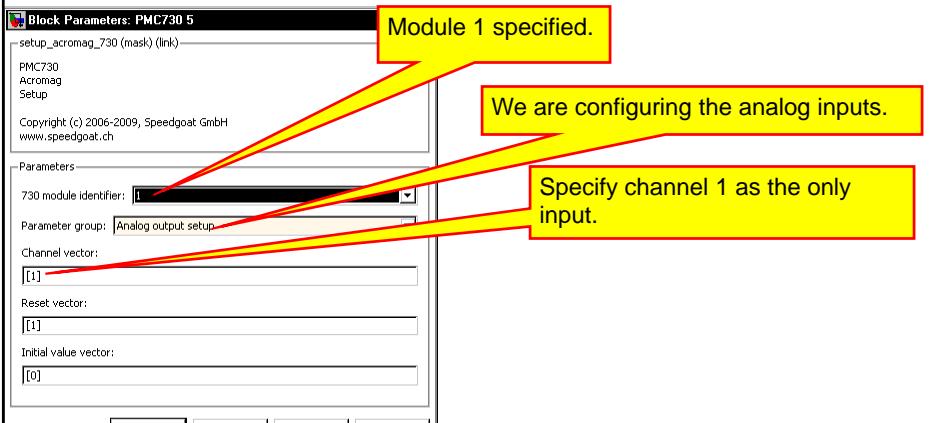
- We need to specify one channel for analog input:



52

## PMC730 5 Acromag setup block

- We need to specify one channel for analog output:

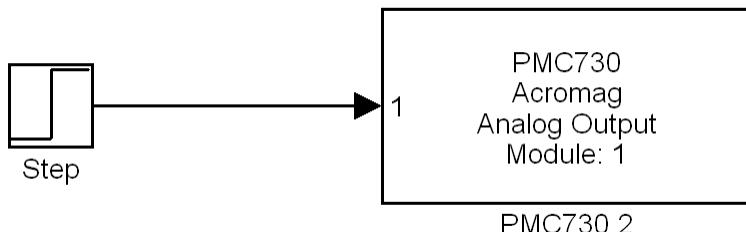




## Analog Output

53

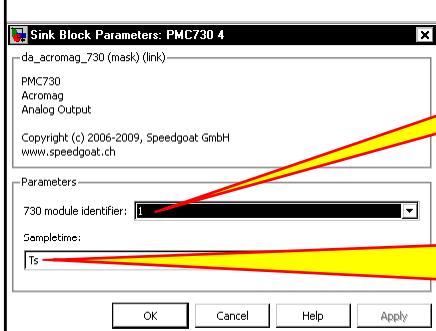
- Next, we will add an analog output block to provide a 0 to 5 V step to the motor controller.
- Place a block called **PMC730 2** located in the **xPC Target: Speedgoat I/O Driver Library / IO101** library in your model.
- Connect a **Step** block from the **Simulink / Sources** library to the Analog output block:



## Analog Output

54

- The settings for the analog output block are:



We are using the analog output channel of the PMC730 identified as module 1 in the setup block.

We will specify the sample time as a variable so that we can change the rate at which we collect data in the model. Ts will be defined in the workspace.



The MathWorks

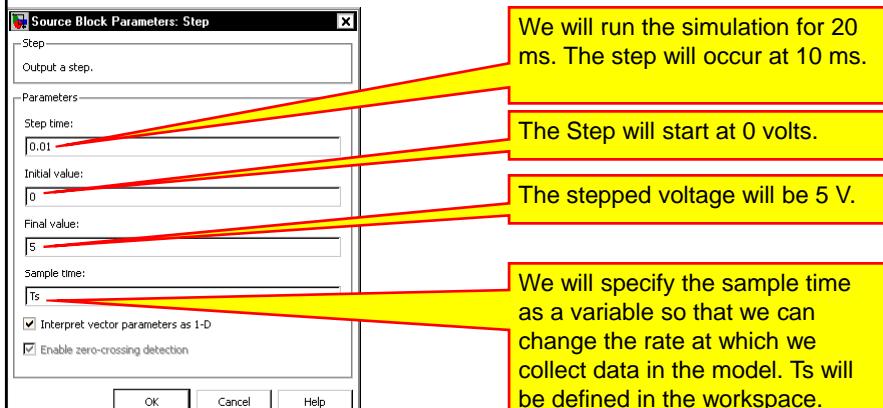

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Step Block

55

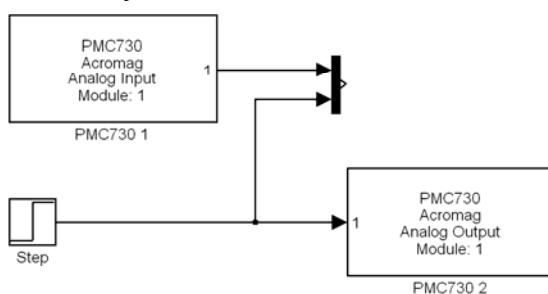
- The settings for the step block are:



## Analog Input

56

- We will use an analog input channel to measure the rpm.
- Place a block called **PMC730 1** located in the **xPC Target: Speedgoat I/O Driver Library / IO101** library in your model.
- Add a **Mux** from the **Simulink / Commonly Used Blocks** library and connect as shown:

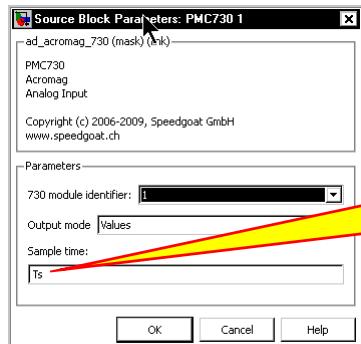




57

## Analog Input

- The properties of the analog input block are:



We will specify the sample time as a variable so that we can change the rate at which we collect data in the model.  $T_s$  will be defined in the workspace.

**MotoTron**

The MathWorks

freescale<sup>®</sup>  
semiconductor

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY

58

## Simulink xPC Target Library

- Next, we need to place some xPC scopes in our model.
- xPC Scopes can be used to generate plots or to save data in a file.
- Open the Simulink Library Browser by selecting **View** and then **Library Browser** from the Simulink menus or clicking the Library Browser button
- Expand the xPC Target library near the bottom of the list:

**MotoTron**

The MathWorks

freescale<sup>®</sup>  
semiconductor

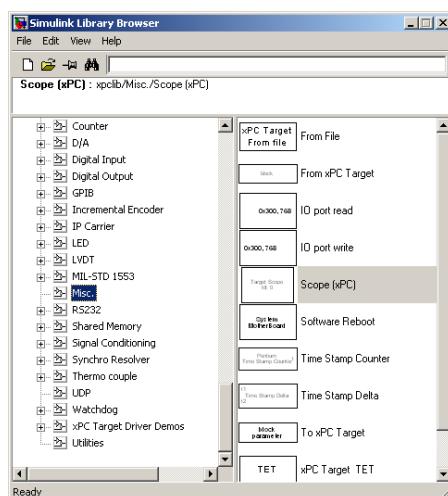
ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



## Simulink xPC Target Library

59

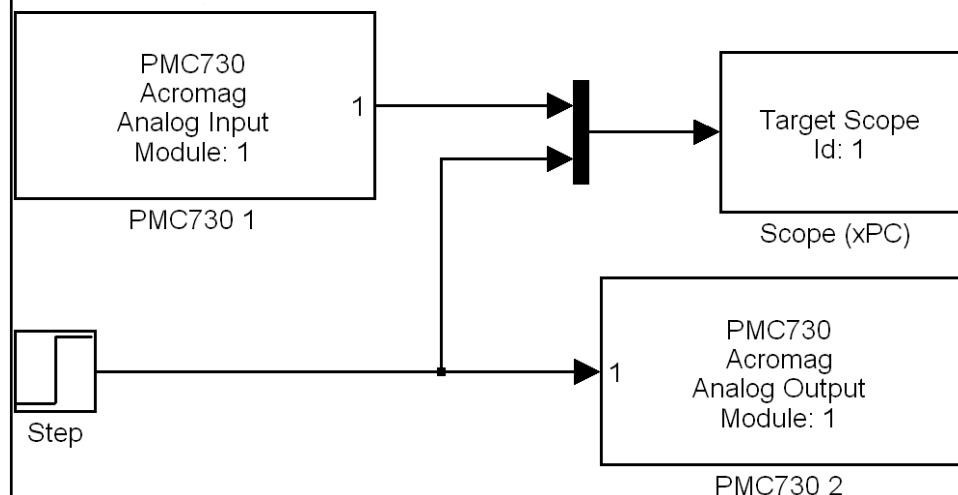
- We are looking for the xPC Scope which is in the **Misc** library:
- These blocks will display a trace on the target PC screen or can be used to record data in a file.
- Place a scope in your model as shown.



## Model Changes

60

- We want to record the applied step signal and the measured rpm.

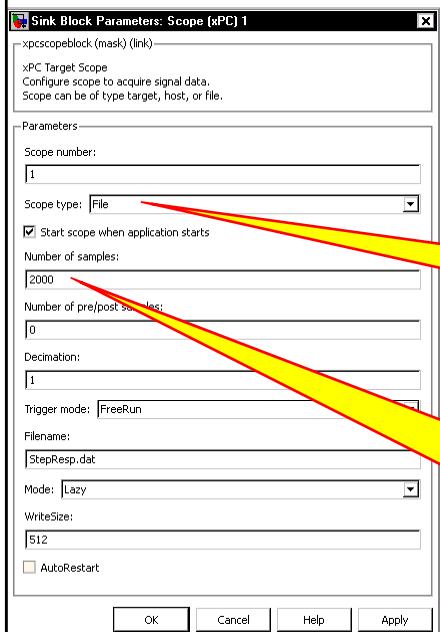




## Model Changes

61

- Next, double-click on the Target Scope block and make the changes below.



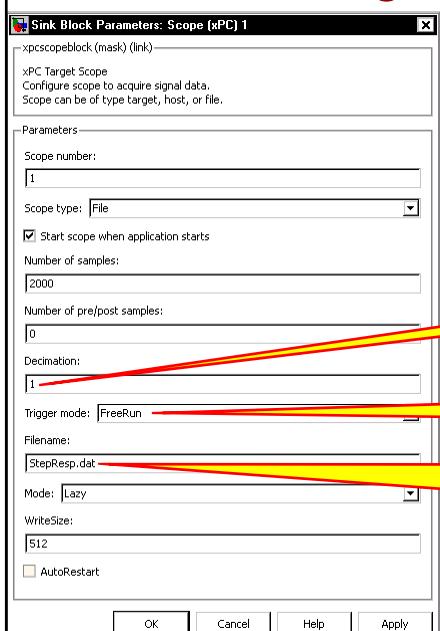
Specify the Scope type as File. This will save the input signals to a file.

Collect a total of 2000 data points. The amount of time the data will span is the number of data points times the sample time (assuming a decimation of 1). For a step time of 0.00001, we can record data for 20 ms, the entire length of our simulation.

## Model Changes

62

- Next, double-click on the Target Scope block and make the changes below.



Record every data point.

This trigger method means that the scope will be triggered automatically.

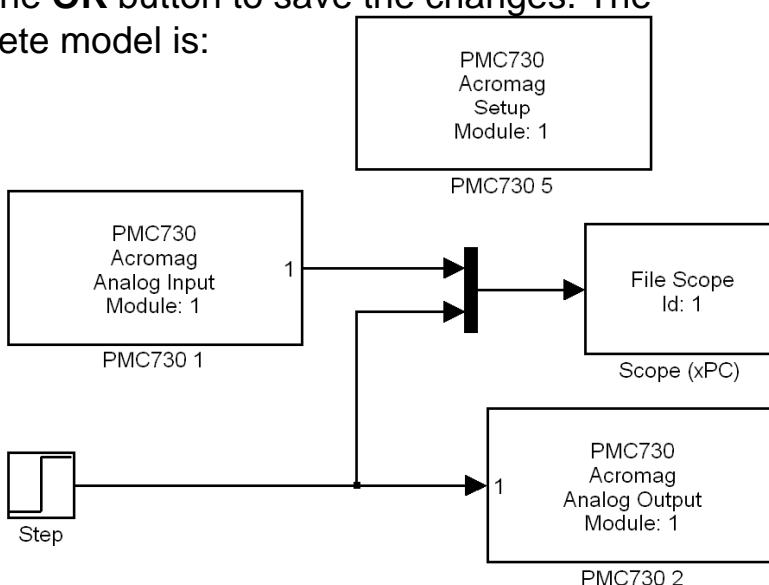
Data is saved on the real-time target in this file. Change the name of the file to Step\_xxx.dat where xxx are your initials.



63

## Model Changes

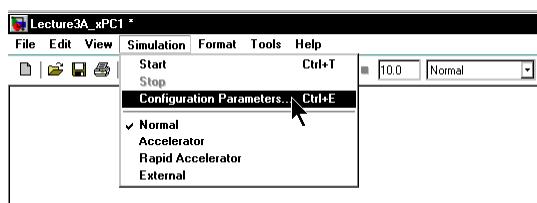
- Click the **OK** button to save the changes. The complete model is:



64

## Simulation Parameter Setup

- Next, we need to set up the simulation parameters.
- Select **Simulation** and then **Configuration Parameters** from the Simulink menus





65

## Simulation Parameter Setup

- There are a number of parameters we need to change.
- Select the **Solver** tab and specify the following parameters
  - Stop time: 0.02
  - Type: Fixed-step
  - Solver: Discrete (no continuous states)
  - Fixed-Step Size:  $T_s$
- These selections are shown on the following slide.

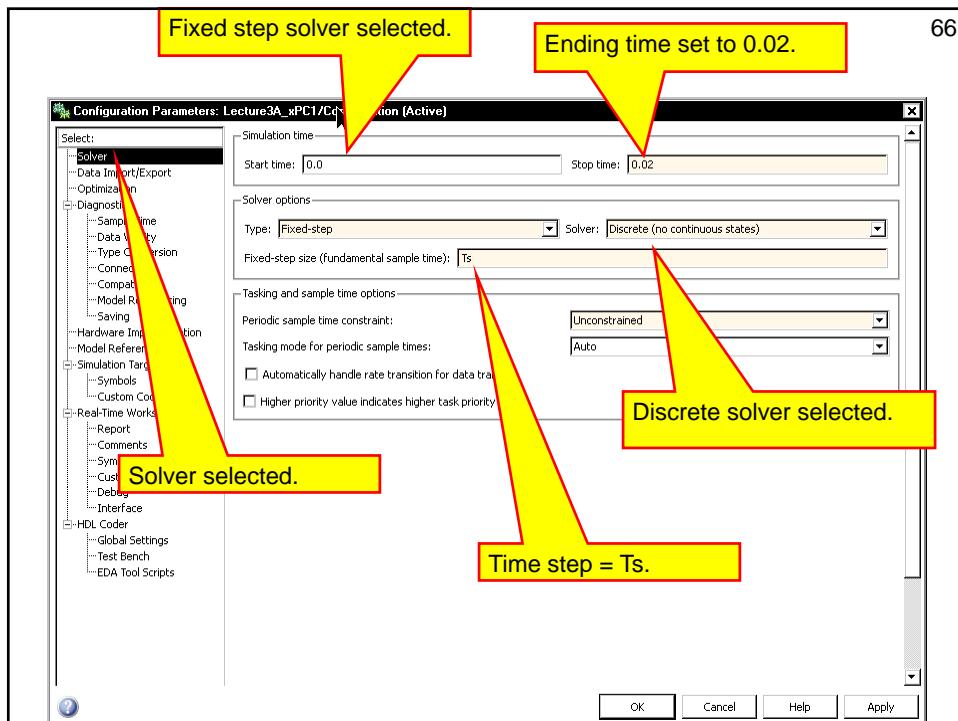
**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

66

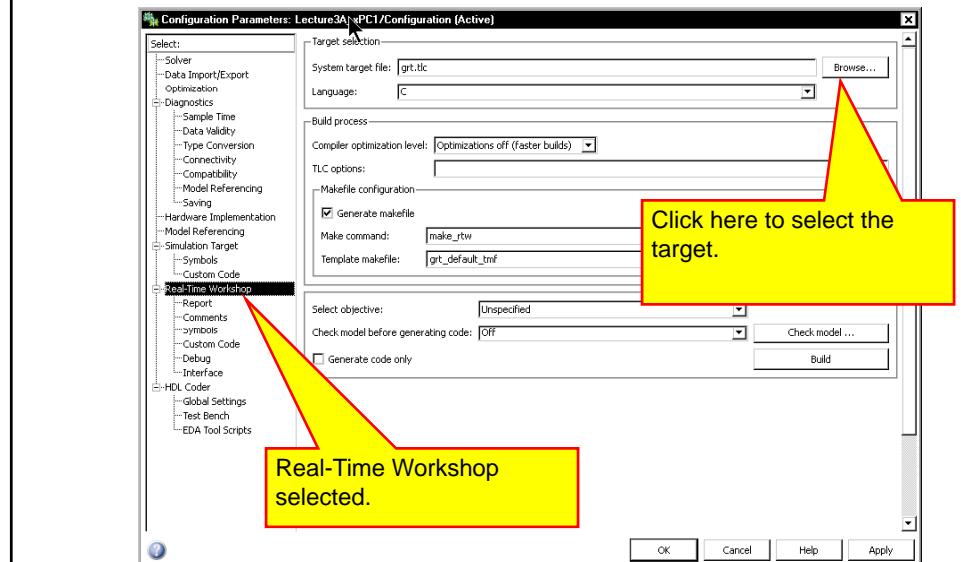




67

## Simulation Parameter Setup

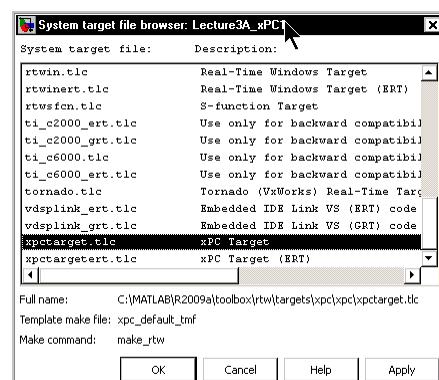
Next, select the **Real-Time Workshop** tab:



68

## Simulation Parameter Setup

- We need to specify that we will be using a xPC target as our target system.
- Click the **Browse** button, select the file named **xpctarget.tlc**, and then select **OK**.

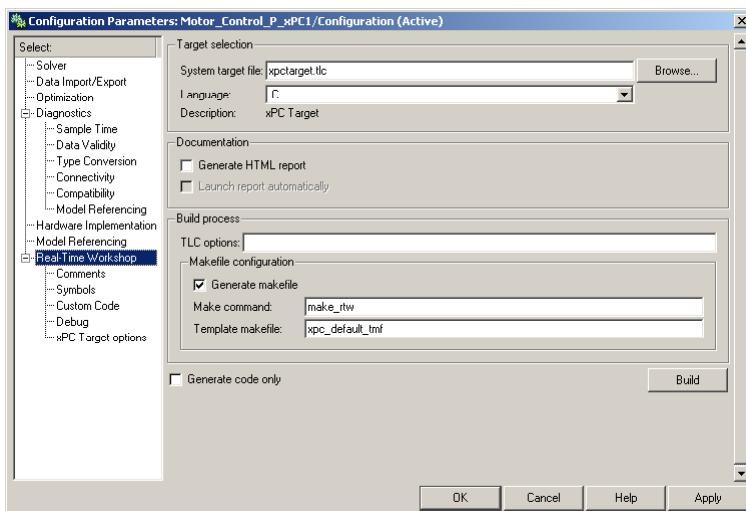




69

## Simulation Parameter Setup

- Your dialog box should look as shown.



70

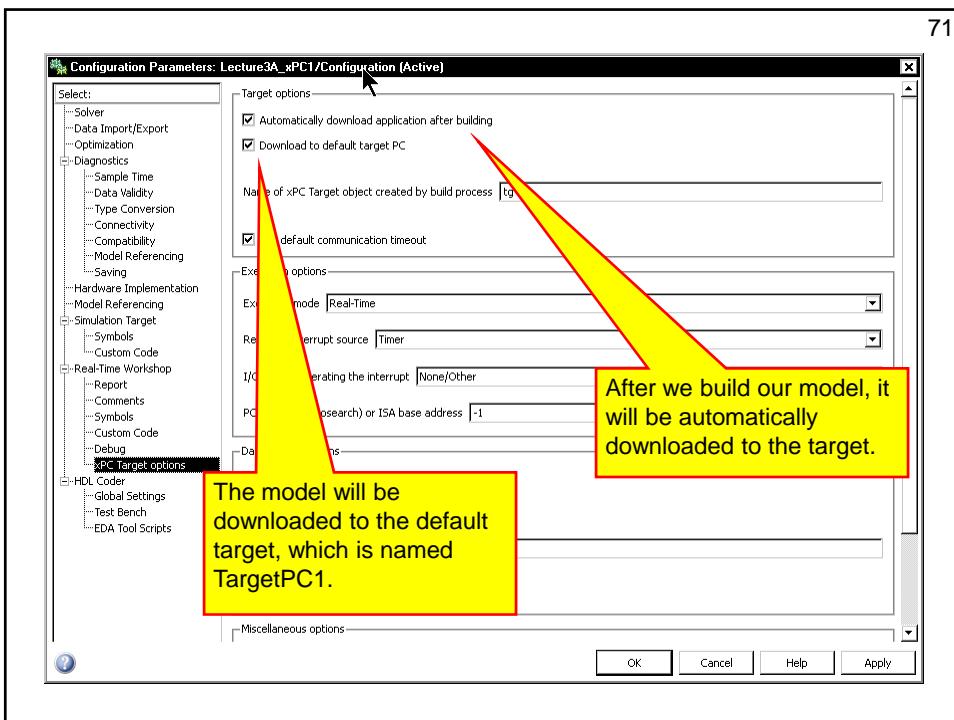
## Simulation Parameter Setup

- We do not need to make any other changes.
- However, we will look at the xPC Target options tab to see some other options that control behavior we will see.
- Select the **xPC Target options** tab.





71



72

## Simulink xPC Target Library

- The Configuration Parameters are now complete so click the **OK** button to accept the settings.
- Next, we need to define the value of the sample time,  $T_s$ .
- You can do this at the Matlab prompt or as part of the initialization process of the model, which is how we will do it.
- Select **File** and then **Model Properties** from the Simulink menus:





73

## Model Properties

The screenshot shows the MATLAB interface with a model named 'Lecture3A\_xPC1'. The 'File' menu is open, and the 'Model Properties...' option is selected. A red arrow points from this option to the 'Callbacks' tab of the 'Model Properties' dialog box. The 'Callbacks' tab displays the 'Model initialization function:' field, which contains the path to the 'InitFcn' file.

**Model Information for: Lecture3A\_xPC**

Source File: C:\Documents and Settings\herniter\My Documents\Website\Rose  
 Last Saved: Sun Dec 06 14:59:17 2009  
 Created On: Sun Dec 06 13:36:00 2009  
 Is Modified: yes  
 Model Version: 1.2

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

74

## Model Properties

- Select the **Callbacks** tab and select the **InitFnc** tab.

The screenshot shows the 'Callbacks' tab selected in the 'Model Properties' dialog box. The 'InitFnc' tab is highlighted in the left pane under 'Model callbacks'. The right pane shows the 'Model initialization function:' field, which is currently empty.

- Commands in this window will be executed every time the model is run, prior to the model execution.

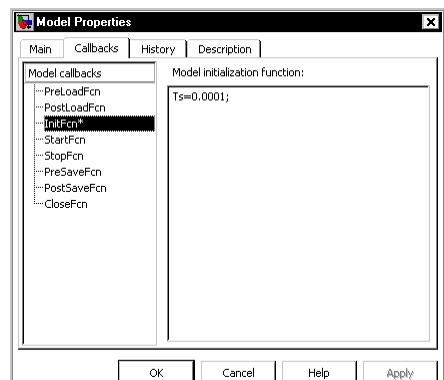
**MotoTron** **The MathWorks** **freescale** **ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



75

## Model Properties

- We can place any Matlab commands in this pane.
- We will define the sample time here as 100 microseconds:



**MotoTron**

The MathWorks

freescale  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

76

## Model Wiring

- The last thing we need to do before running the model is to connect the real-time system to the system under test.
- Pin connections for the PMC730 are defined in the users manual located on the website for MBSD1.
- We are using Analog Output 1.
- We need to also find the ground reference for this output.
- The data sheet is shown on the next slide:

**MotoTron**

The MathWorks

freescale  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



77

## Analog Output Pin Connection

Table 2.2: PMC730 Field I/O Pin Connections

Pin Description	Pin	Pin Description	Pin
Counter Output	1	COMMON	
Dig CH0/ ADC Trigger In	2	Dig CH8/ ADC Trig Out	36
Dig CH1/ DAC Trigger In	3	Dig CH9/ DAC Trig Out	37
Dig CH2/ Counter Input	4	Digital CH10/ Counter Output	38
Dig CH3/ Counter Trig In	5	Digital CH11	39
Dig CH4/ Counter Ext Clk	6	Digital CH12	40
Dig CH5/ Counter Gate off	7	Digital CH13	41
Digital CH6	8	Digital CH14	42
Digital CH7	9	Digital CH15	43
COMMON	10	Analog Out CH4	44
COMMON	11	Analog Out CH5	45
Analog Out CH0	12	COMMON	46
Analog Out CH1	13	COMMON	47
Analog Out CH2	14	COMMON	48
Analog Out CH3	15	COMMON	49
COMMON	16	Analog Out CH6	50
COMMON	17	Analog Out CH7	51
COMMON	18	SENSE	52
Analog In S15/D15+	19	Analog In S31/D15-	53
Analog In S14/D14+	20	Analog In S30/D14-	54
Analog In S13/D13+	21	Analog In S29/D13-	55
Analog In S12/D12+	22	Analog In S28/D12-	56
Analog In S11/D11+	23	Analog In S27/D11-	57
Analog In S10/D10+	24	Analog In S26/D10-	58
Analog In S9/D9+	25	Analog In S25/D9-	59
Analog In S8/D8+	26	Analog In S24/D8-	60
Analog In S7/D7+	27	Analog In S23/D7-	61
Analog In S6/D6+	28	Analog In S22/D6-	62
Analog In S5/D5+	29	Analog In S21/D5-	63
Analog In S4/D4+	30	Analog In S20/D4-	64
Analog In S3/D3+	31	Analog In S19/D3-	65
Analog In S2/D2+	32	Analog In S18/D2-	66
Analog In S1/D1+	33	Analog In S17/D1-	67

Pin connection information is contained in table 2.2.

Analog output channels listed here.

78

## Analog Output Pin Connection

We specified the first analog output channel. Thus, our output is pin 12.

COMMON	10	Analog Out CH4	44
COMMON	11	Analog Out CH5	45
Analog Out CH0	12	COMMON	46
Analog Out CH1	13	COMMON	47
Analog Out CH2	14	COMMON	48
Analog Out CH3	15	COMMON	49
COMMON	16	Analog Out CH6	50
COMMON	17	Analog Out CH7	51
COMMON	18	SENSE	52

We can use any of these common connections as our ground reference.



79

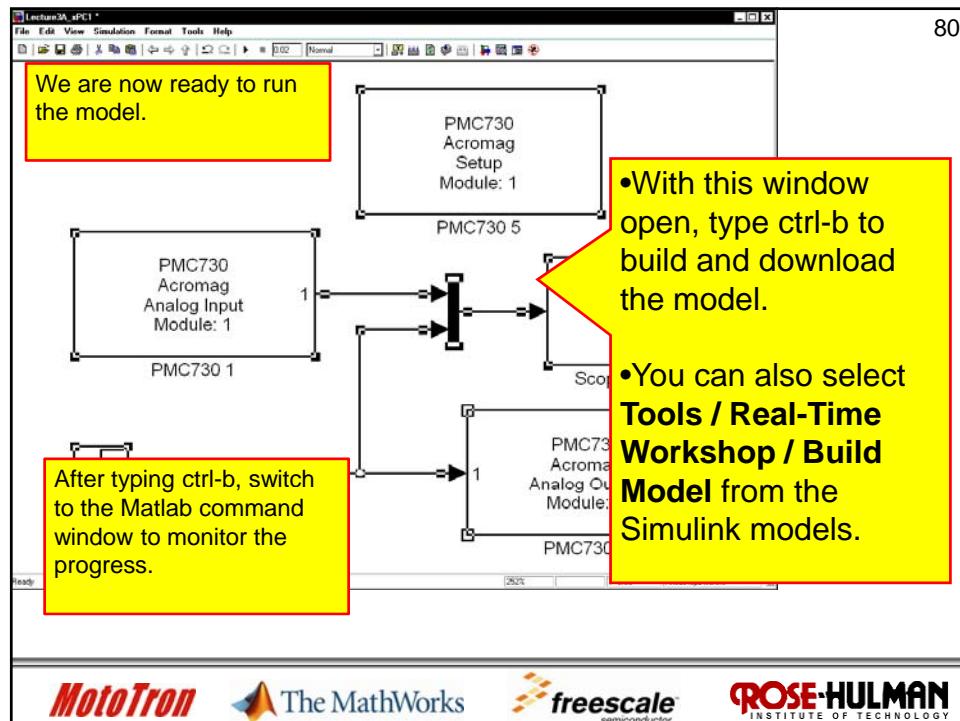
## Analog Input Pin Connection

We can use any of the common connections specified in the previous slide as our ground reference.

Analog In S9/D9+	25	Analog In S25/D9-	59
Analog In S8/D8+	26	Analog In S24/D8-	60
Analog In S7/D7+	27	Analog In S23/D7-	61
Analog In S6/D6+	28	Analog In S22/D6-	62
Analog In S5/D5+	29	Analog In S21/D5-	63
Analog In S4/D4+	30	Analog In S20/D4-	64
Analog In S3/D3+	31	Analog In S19/D3-	65
Analog In S2/D2+	32	Analog In S18/D2-	66
Analog In S1/D1+	33	Analog In S17/D1-	67
Analog In S0/D0+	34	Analog In S16/D0-	68

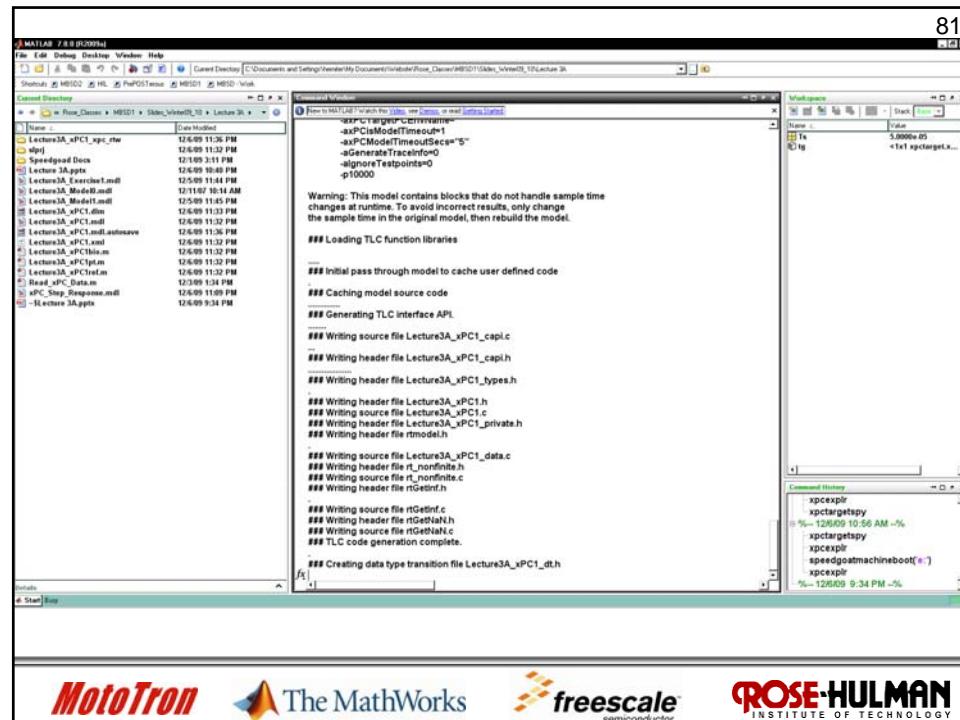
We specified the first analog input channel. Thus, our input is pin 34.

80





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

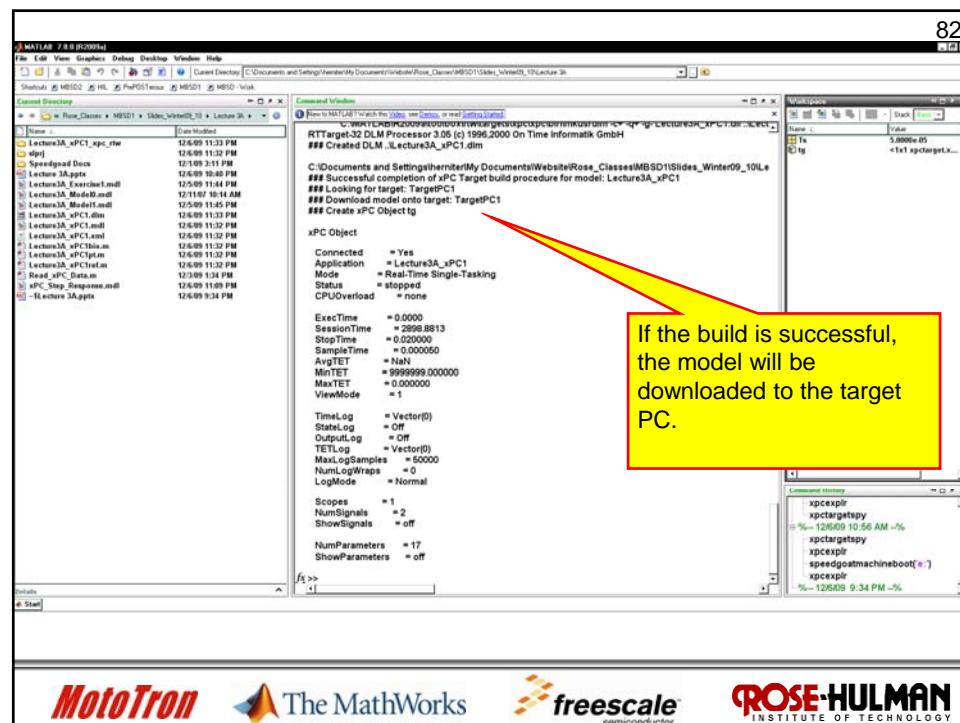


**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

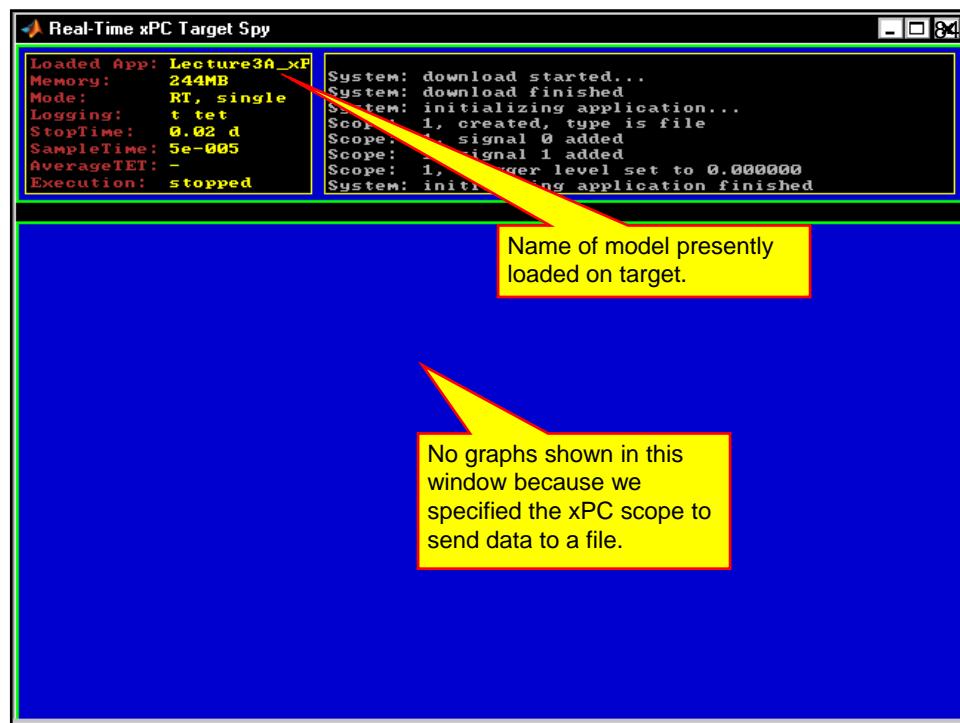
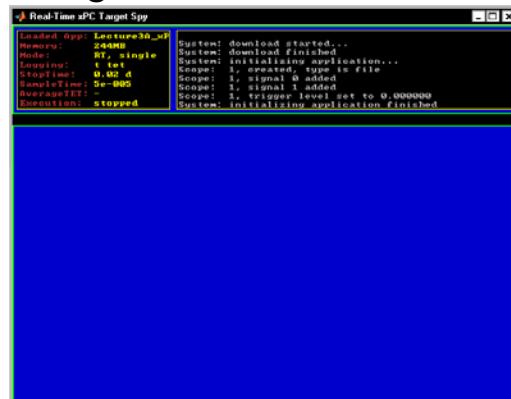




83

## Model Execution

- If you were successful, your model should be downloaded to your target PC.
- If you have a monitor on your PC, you should see the following screen.





85

## Model Execution

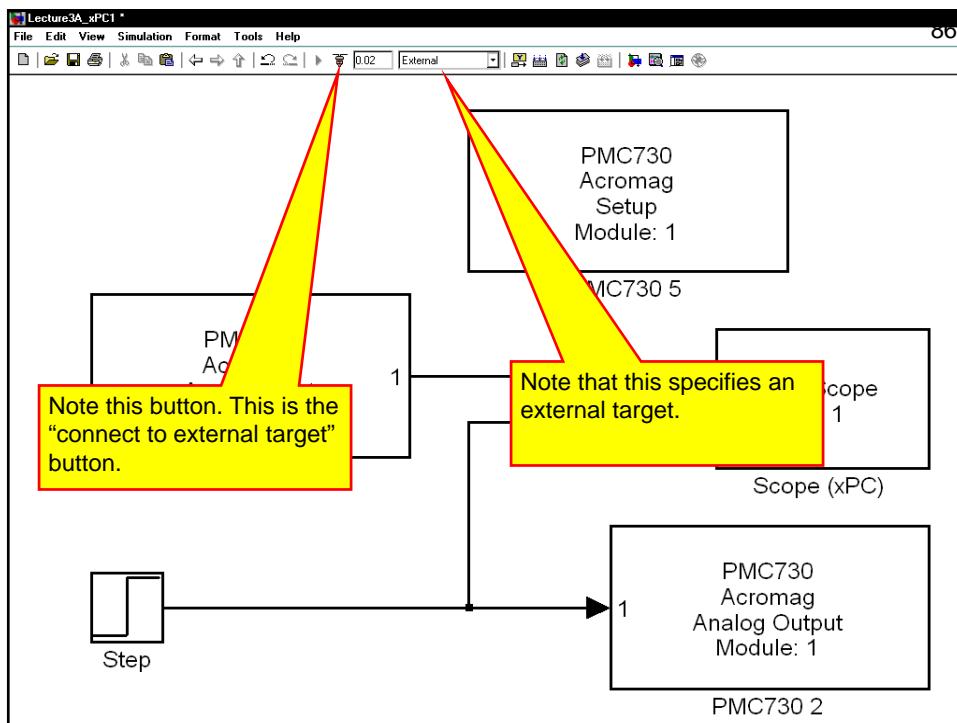
- We are now ready to run the model on our Speedgoat target.
- We must specify that the model should run on an external target.
- Select **Simulation** and then **External** from the Simulink Menus.
- The tool bar should show that you have selected an external target.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY





87

## Model Execution

- Note that a new button has appeared.
- This button means connect to external target.
- Next, we must connect to the external target.
- Click on the button, or select **Simulation** and then **Connect to Target** from the Simulink menus.



88

## Model Execution

- You might ask, how does Matlab know which target to connect to.
- In the Simulation Parameters setup, on the xPC tab we specified to automatically download the model after building, and we specified that it should use the default target.
- In the xPC Explorer, we specified TargetPC1 as the default target.
- Thus, our model will run on TargetPC1.

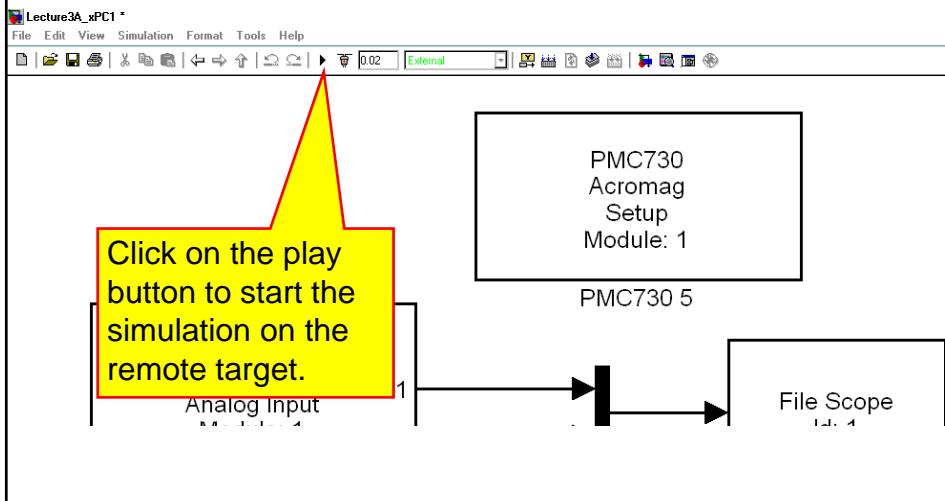




## Model Execution

89

- After connecting to the target, the play ▶ button should become available:



## Model Execution

90

- When the mode runs, you should hear the motors turn on briefly and then turn off.
- The motor just responded to the step and the model ended after 20 ms of execution.
- We now need to check to see if the data file was created on the target.
- We will use the `xpcexplr` to view the root directory of the target pc.





91

## Collected Data File

- Run the xPC Explorer (xpcexplr)
- Connect to the remote target
- Expand the File System portion of the tree.
- Select local disk C:\ to view the contents of the root directory:

92

## Collected Data File

xPC Target Explorer

File Target Application Tools Help

Host PC Root

TargetPC1

Configuration

Communication

Settings

Appearance

File System

local disk c:\

FDOS

FREEDOS

KERMIT.TLS

COMMAND.COM

AUTOEXEC.BAT

FDCONFIG.SYS

CONFIG.TEL

D10000.DSYS

E100BDI.COM

E100DDI.COM

FTPBIN.EXE

IPXODI.COM

LSL.COM

NET.CFG

ODPIKT30.COM

PASSWORD.TEL

RTTBOOT.COM

RUNFTP.BAT

TELPASS.EXE

TERMIN.COM

USBAPI.SYS

XPCRNLRTB

STEPRESP.DAT

Root directory selected.

This is our data file.



## Post Processing

93

- We now need to read the data file from the remote target and then process the data.
- The Matlab script is discussed in the next few slides:

```
f=xptarget.fs;
h=f=fopen('STEPRESP.DAT');
data=f.fread(h);
ffclose(h);
```

Open the file system on the remote target.

Obtain the handle for the data file.

Read the data from the file into variable data. The data is in raw format.

Close the data file.



## Post Processing

94

```
new=readxpcfile(data);
Step_Input=new.data(:,2);
voltage=new.data(:,1);
time=new.data(:,3);

subplot(2,1,1);
plot(time,voltage,'r', time, Step_Input, 'b');
rpm=voltage*1000/2.5;
```

Convert the data to a structure.

The step input is the second column of the data.

The measured voltage is the first column of the data.

Time is the first column of the data.

Generate two plots in one window. Plot the unscaled data in the top plot.





## Post Processing

95

```

ten_pct_point=max(rpm)^0.1;
ninety_pct_point=max(rpm)^0.9;

for i = 2:length(time)
    if (rpm(i) >= ten_pct_point) && (rpm(i-1) < ten_pct_point)
        ten_pct_index = i;
    end
    if (rpm(i) >= ninety_pct_point) && (rpm(i-1) < ninety_pct_point)
        ninety_pct_index = i;
    end
end

subplot(2,1,2);
ten_pct_time=time(ten_pct_index);
ninety_pct_time=time(ninety_pct_index);
rise_time=ninety_pct_time-ten_pct_time;
time_constant = rise_time/2.2;
Response_time=5*time_constant;

```

Determine the 10% and 90% points. Calculate the time constant as the 10 to 90% rise time divided by 2.2.

## Post Processing

96

```

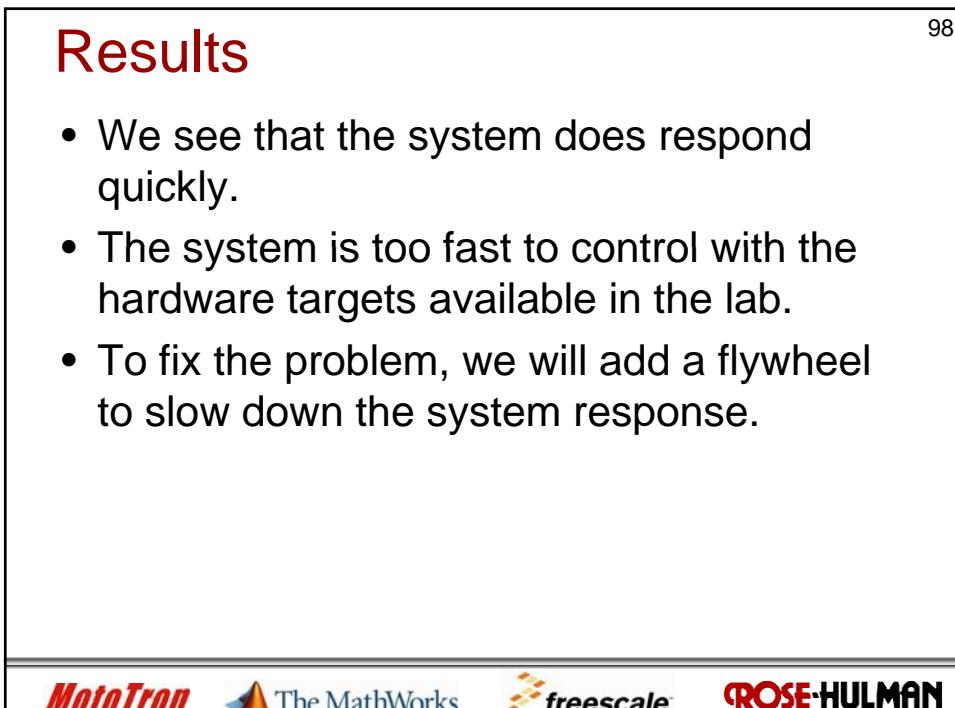
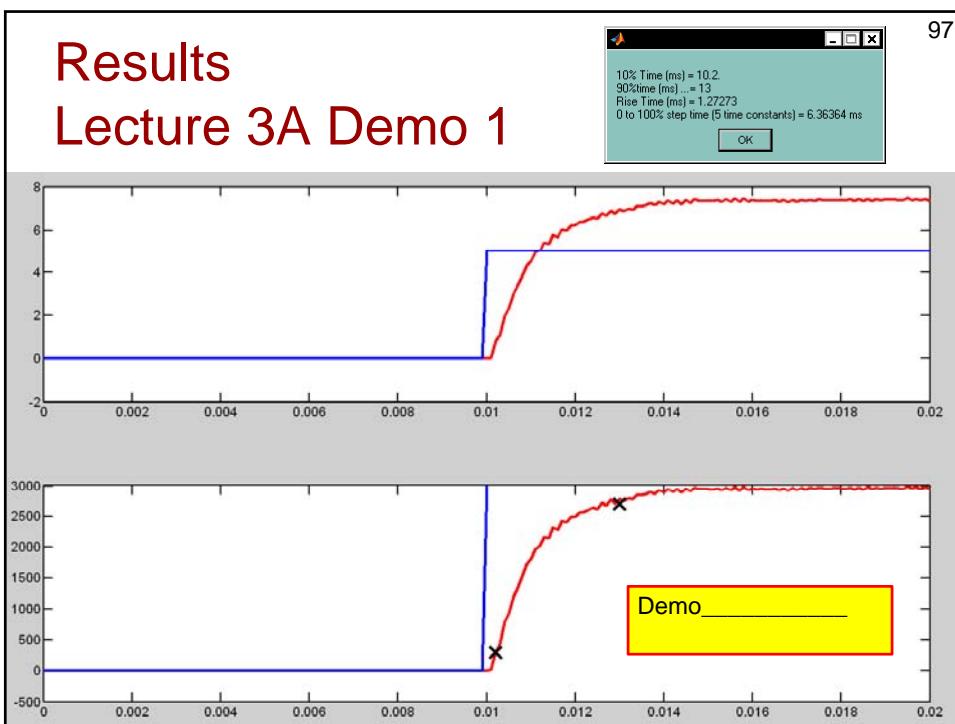
plot(time,rpm,'r',time,Step_Input*3000/5,'b', ...
ten_pct_time, ten_pct_point,'kx', ninety_pct_time, ...
ninety_pct_point, 'kx','MarkerSize',20);

msg1=sprintf('10%% Time (ms) = %g.\n90%% time (ms) ... = %g\nRise Time (ms) = %g\n',...
ten_pct_time*1000, ninety_pct_time*1000, time_constant *1000);
msg2=sprintf('0 to 100%% step time (5 time constants) = %g ms', Response_time*1000);
msgbox([msg1, msg2]);

```

Plot the scaled data and the 10% and 90% points on the second plot in the window.

Display calculated values in a message box.

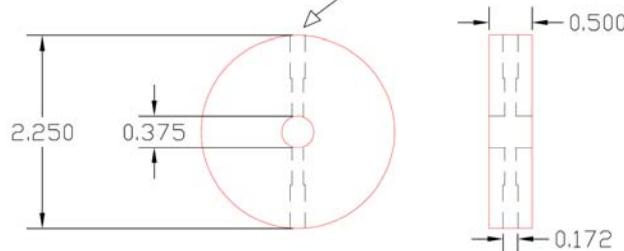




## Flywheel

Drill # 29 Thru  
Tap 8-32 Thru

99



- We will add the flywheel shown above to the motor-generator system.
- The calculated inertia of the flywheel is  $1.041 \times 10^{-4} \text{ kg}\cdot\text{m}^2$ .

**MotoTron**

**The MathWorks**

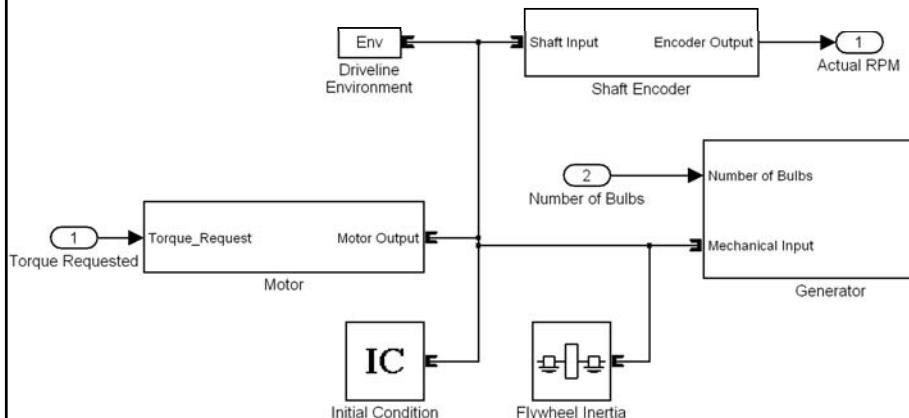
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Flywheel

100

- In the model, this change has the effect of adding more inertia to the system as shown below:



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



101

## Lecture 3A Exercise 2

- Add the flywheel inertia to your system. (Repeat exercise 1 with the added inertia.)
- Use the fixed-step ode3 (Bokacki-Shampine) solver and determine the largest fixed step size that can be used to simulate the system without yielding an oscillation in the motor torque or plant rpm.
- The desired rpm should be set to 1800, number of bulbs to 3, and the feedback gain to 0.01 (same as used in the previous slides.)
- Note that when we deploy the controller on the MPC555x target, a fixed step size will be used. Thus, this is a good test to determine the required step size needed for our controller.
- Find the max step size to the nearest 1 ms.



Demo\_\_\_\_\_

## Lecture 3A – Problem1 System Rise Time

102

Demo\_\_\_\_\_

- We would like to redo our rise time measurement using the motor generator system with the added flywheel.
- A second Speedgoat real-time system has been set up to control and measure data from the motor generator system:
  - The IP address is [xxx.xxx.85.112](http://xxx.xxx.85.112).
  - The rpm signal conversion is 1.25 V per 1000 rpm. (We added a voltage divider to cut the rpm signal in half.)
- Repeat the measurements of Lecture 3A Demo1 for the Motor-Generator system with a flywheel. Note that this system is probably 50 to 100 times slower than the motor itself.
- Generate a plot, calculate the time constant, and calculate the rise time.
- Note that this system is wired differently than the motor.





103

Demo \_\_\_\_\_

## Lecture 3A – Problem2 Measured Coast Down

- We would like to measure the coast-down response of the motor-generator system used in Problem 1.
- Let the motor speed up to its maximum speed and then allow the motor to coast to a stop.
- For the coast-down, measure:
  - The time constant.
  - The time it takes to reach zero (5 time constants)
  - Generate a plot of the coast-down response showing with the 10% and 90% points marked.
- Save the coast-down data in a file on your computer for use in a later example.



The MathWorks



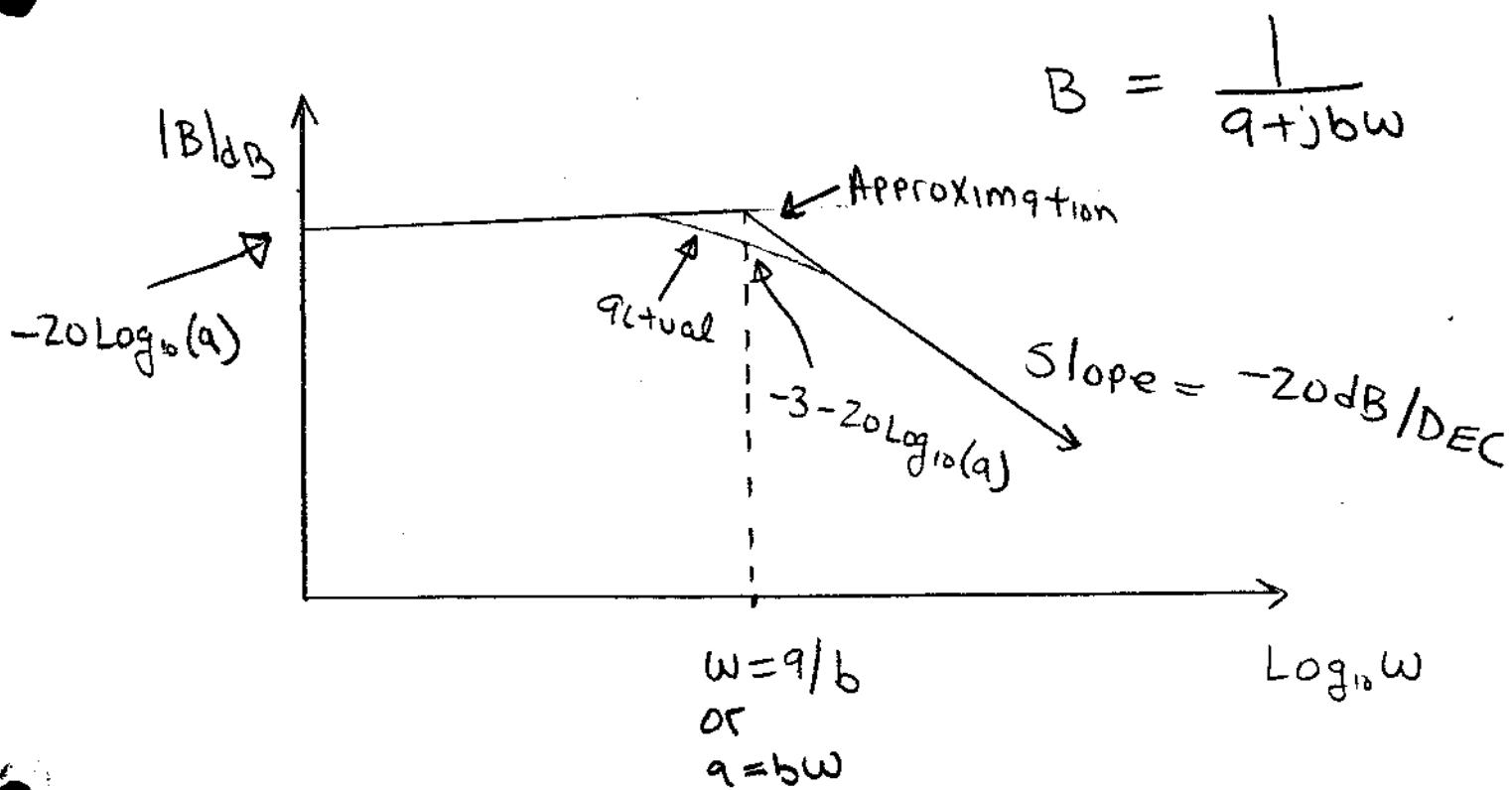
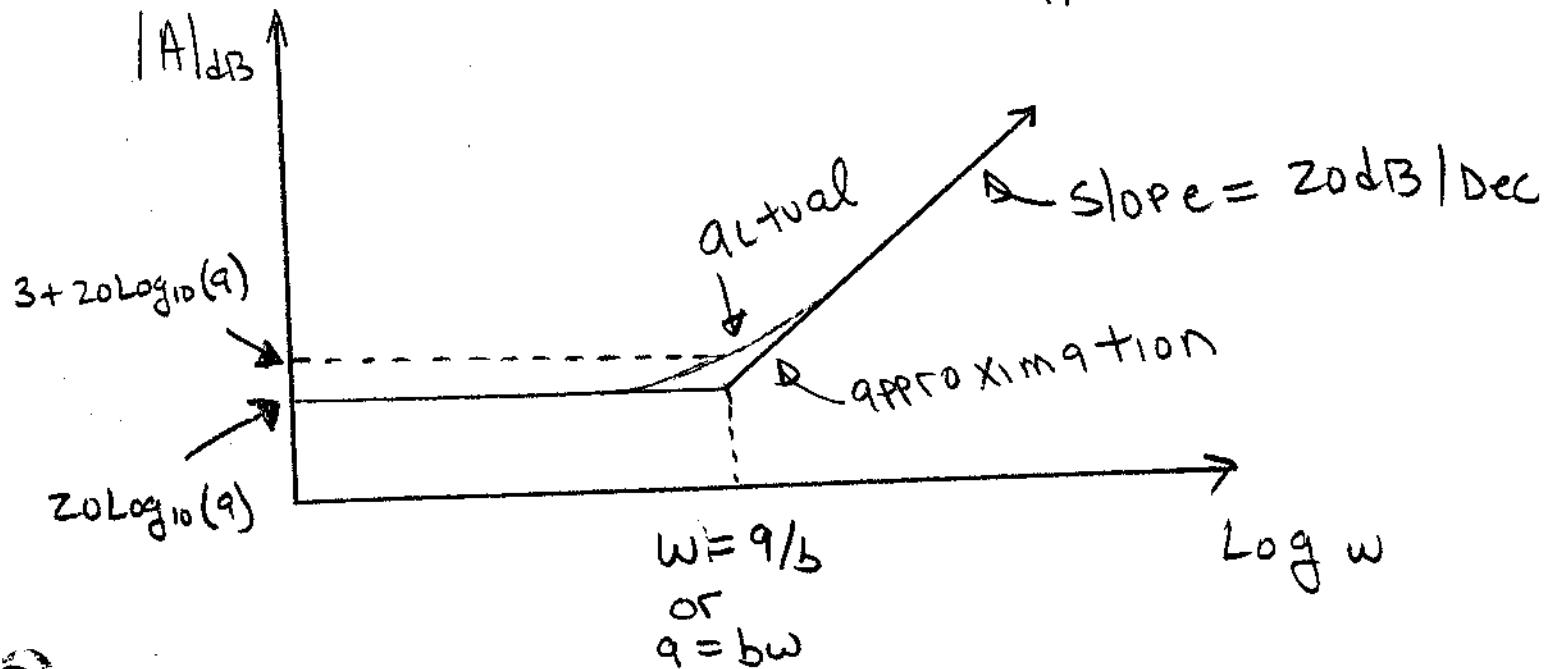
# Bode Magnitude Plot



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

## Approximation

$$A = a + jb\omega$$





$$\text{Let } A = a + j b \omega$$

$$\text{then } \angle A = \tan^{-1} \left( \frac{b\omega}{a} \right)$$

For  $a \gg b\omega$  or  $\omega \ll 9/b$

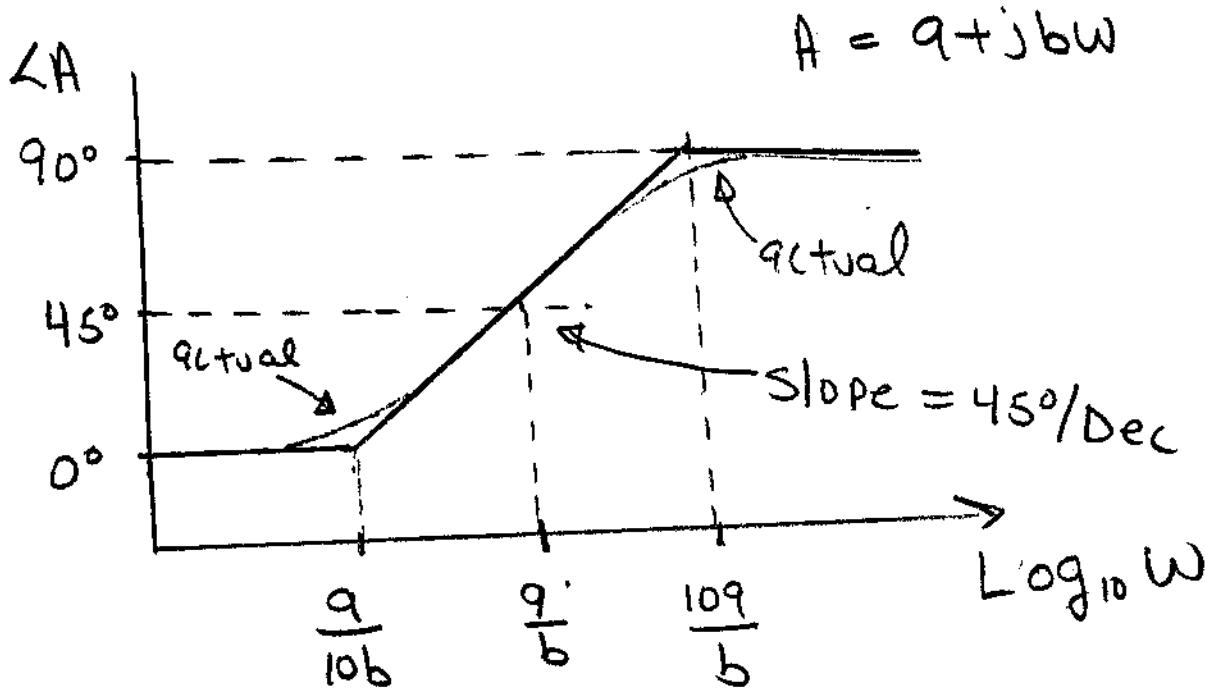
$$\begin{aligned}\angle A &= \tan^{-1} \left( \frac{b\omega}{a} \right) \approx \tan^{-1} \left( \frac{\text{Very small}}{\#} \right) \\ &= \tan^{-1}(0) = 0^\circ\end{aligned}$$

For  $a = b\omega$

$$\begin{aligned}\angle A &= \tan^{-1} \left( \frac{b\omega}{a} \right) = \tan^{-1} \left( \frac{1}{1} \right) \\ &= \tan^{-1}(1) = 45^\circ\end{aligned}$$

For  $b\omega \gg a$  or  $\omega \gg 9/b$

$$\begin{aligned}\angle A &= \tan^{-1} \left( \frac{b\omega}{a} \right) = \tan^{-1} \left( \frac{\text{Very large}}{\#} \right) \\ &= \tan^{-1}(\infty) = 90^\circ\end{aligned}$$



Now for  $B = \frac{1}{a + jb\omega}$

Remember  $\angle\left(\frac{\text{Top}}{\text{bottom}}\right) = \angle\text{Top} - \angle\text{bottom}$

So

$$\angle B = \angle\left(\frac{1}{a + jb\omega}\right) = \angle 1 - \angle(a + jb\omega)$$

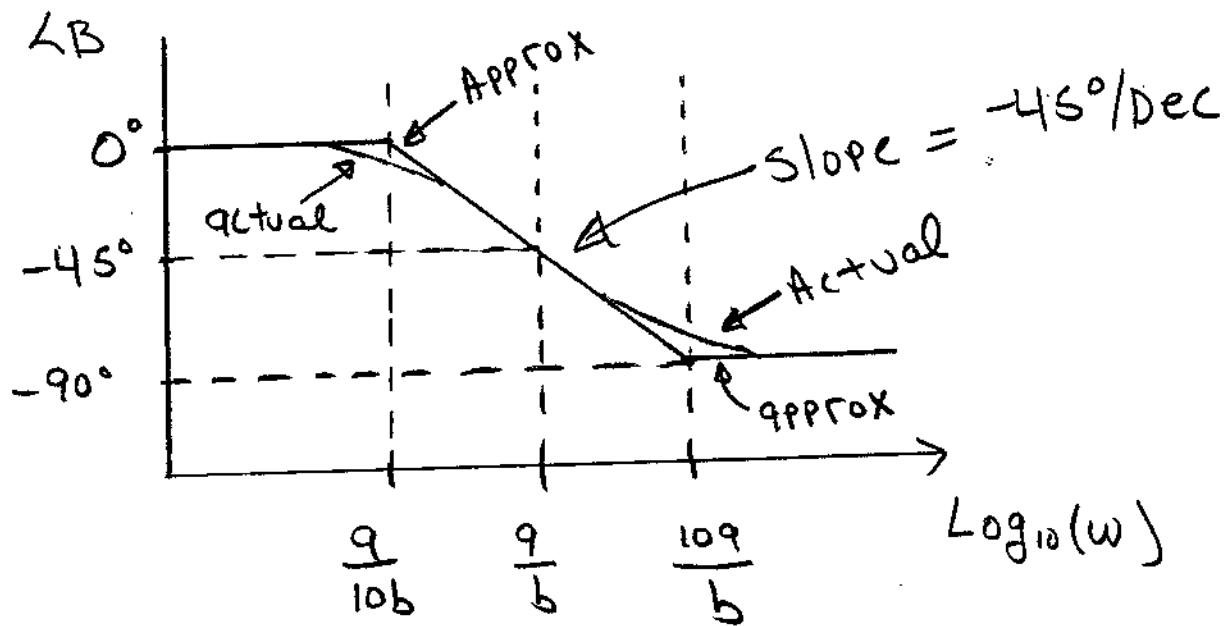
$$\angle 1 = 0^\circ \text{ so}$$

$$\angle B = -\angle(a + jb\omega) = -\angle A$$



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

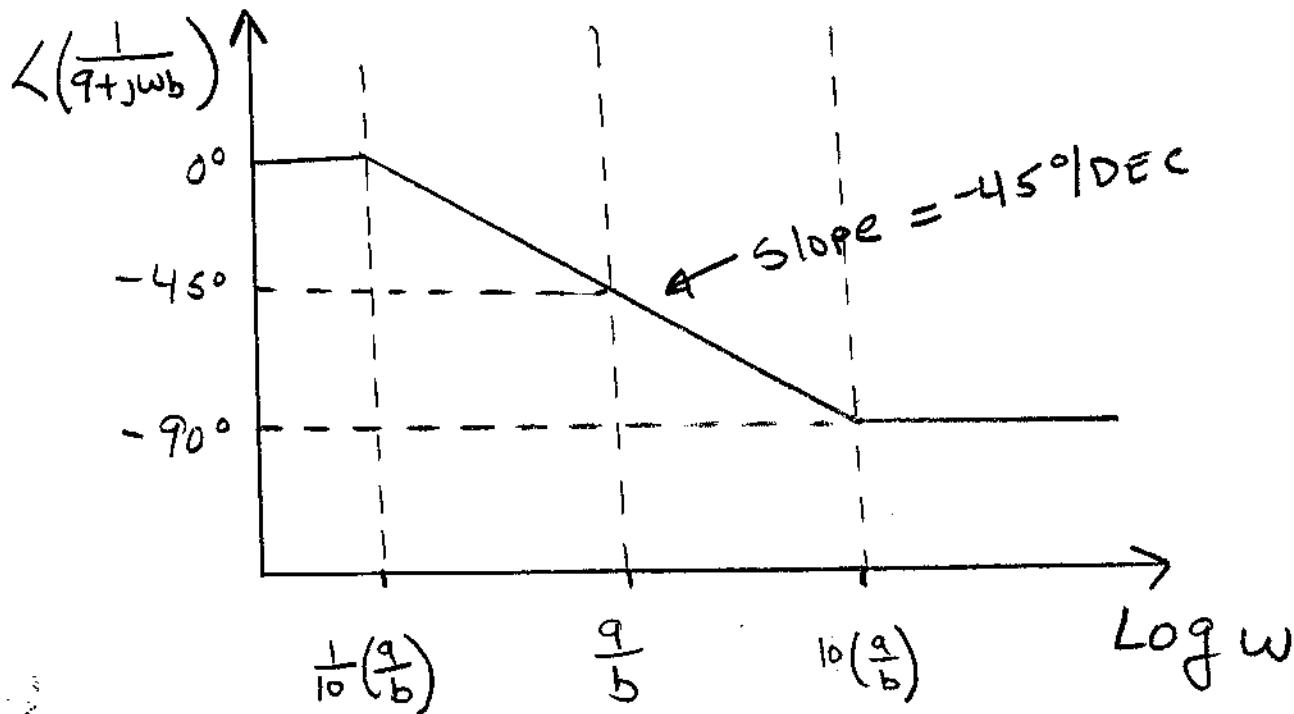
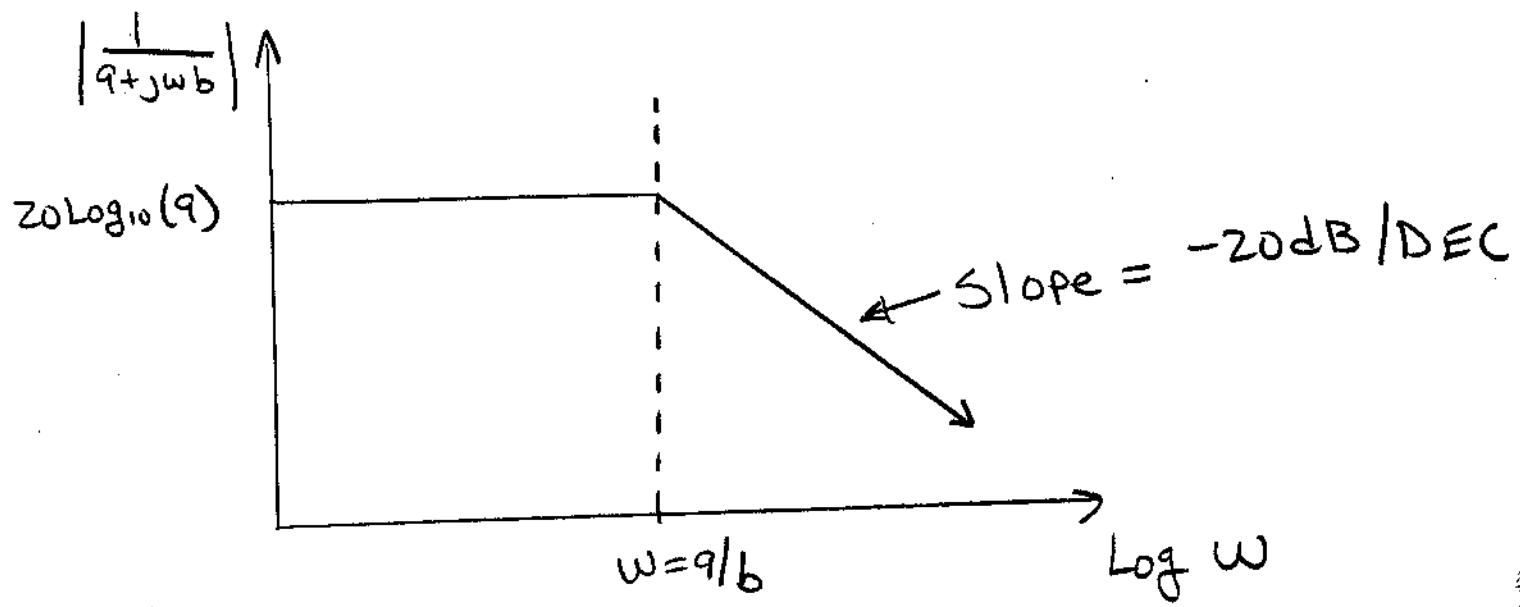
So for  $B = \frac{1}{a+jwb}$



# Pole @ On The denominator $\frac{1}{q+jwb}$

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

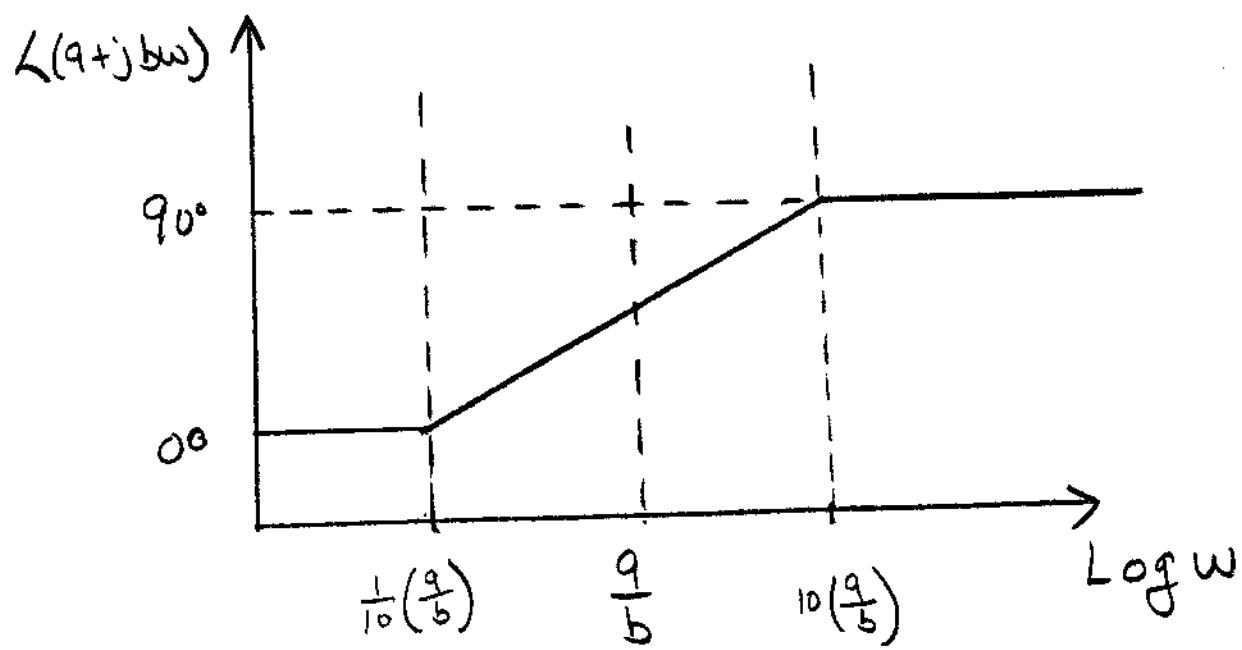
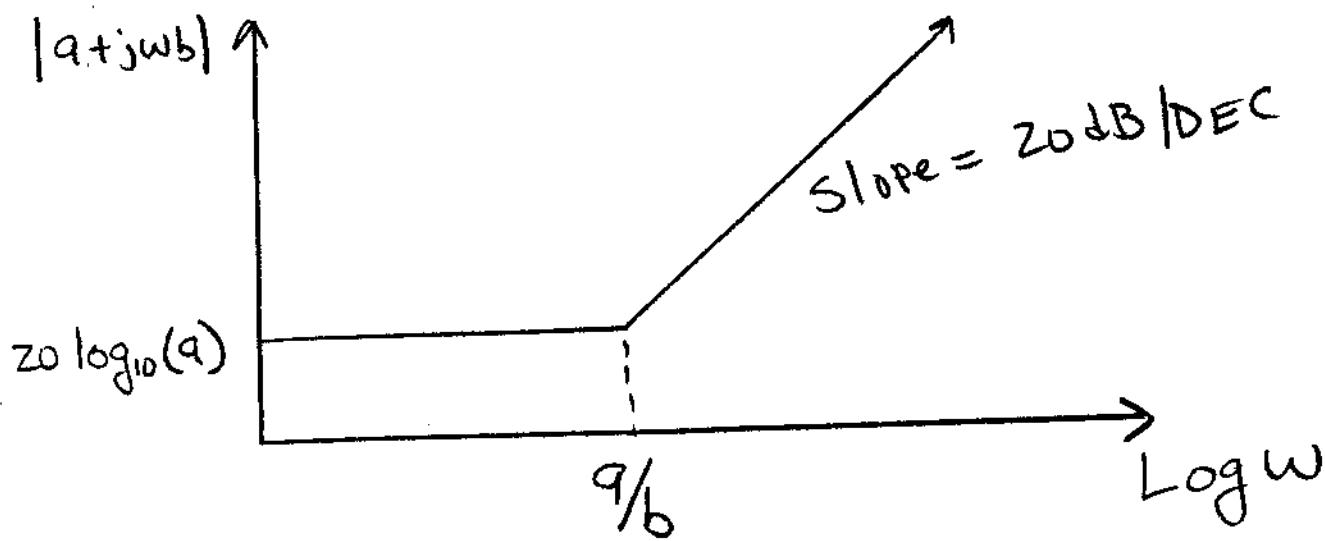
this is a pole at  $\omega = q/b$



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

Zero - in the numerator,  $a + j\omega b$

This is a zero at  $\omega = \frac{9}{b}$



# Multiple Poles and Zeros



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

$$C = \frac{(a+jbw)(c+jdw)}{(e+jfw)}$$

$$|C| = \frac{|a+jbw| |c+jdw|}{|e+jfw|}$$

$$|C|_{dB} = 20 \log_{10} \left( \frac{|a+jbw| |c+jdw|}{|e+jfw|} \right)$$

$$\begin{aligned}
 &= 20 \log_{10} |a+jbw| + \\
 &\quad 20 \log_{10} |c+jdw| - \\
 &\quad 20 \log_{10} |e+jfw|
 \end{aligned}$$

$\Rightarrow$  can do each plot separately  
 and add or subtract  
 them. This can be done for both  
 magnitude and phase.

We ~~will~~ do it all at once

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

## Magnitude plot

- each pole gives a slope of  
 $-20 \text{ dB/Dec}$
- each zero gives a slope of  
 $+20 \text{ dB/Dec}$

## Phase plot

- each pole gives a total phase of  $-90^\circ$ . gives a slope of  $-45^\circ/\text{Dec}$   
 For one decade below and one decade above the pole.
- each zero gives a total phase of  $+90^\circ$ . gives a slope of  $45^\circ/\text{DEC}$   
 For one decade below and one decade above the zero.

Frequency (r/s)

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

1000

SEMILOGARITHMIC. 5 CYCLES X 100 THE INCH  
5TH LINES ACCENTED

MAGNITUDE

-60

-80



X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

X

$$H(j\omega) = \frac{(1+j\omega)}{(10+j\omega)(100+j\omega)}$$

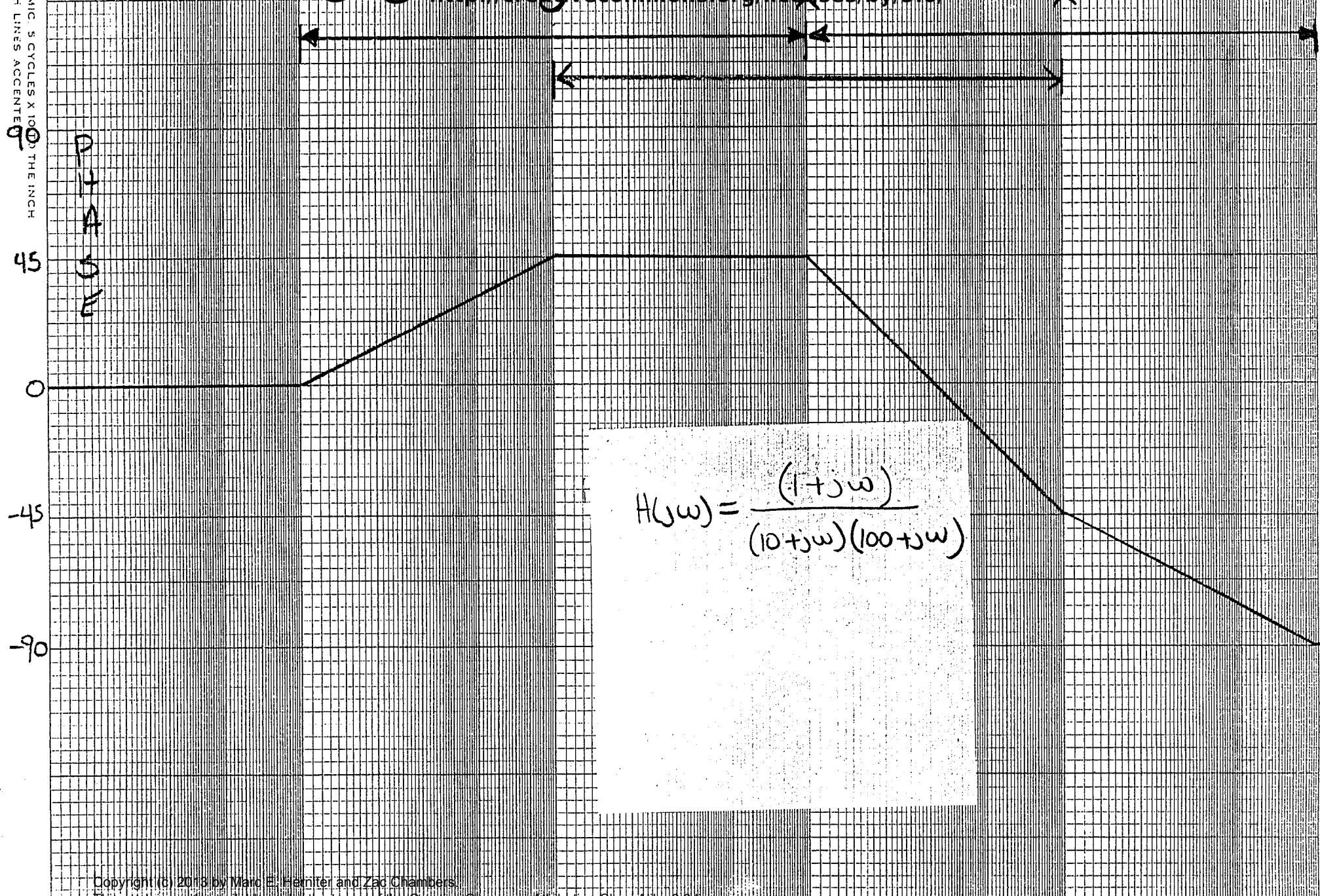
$$H(0) = \frac{1}{(10)(100)} \\ = \frac{1}{1000} = -60 \text{ dB}$$

## Frequency

161 of 783

SEM LOGARITHMIC 5 CYCLES X 100 THE INCH  
5TH LINES ACCENTED

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



# Frequency (r/s)

324C

162 of 783

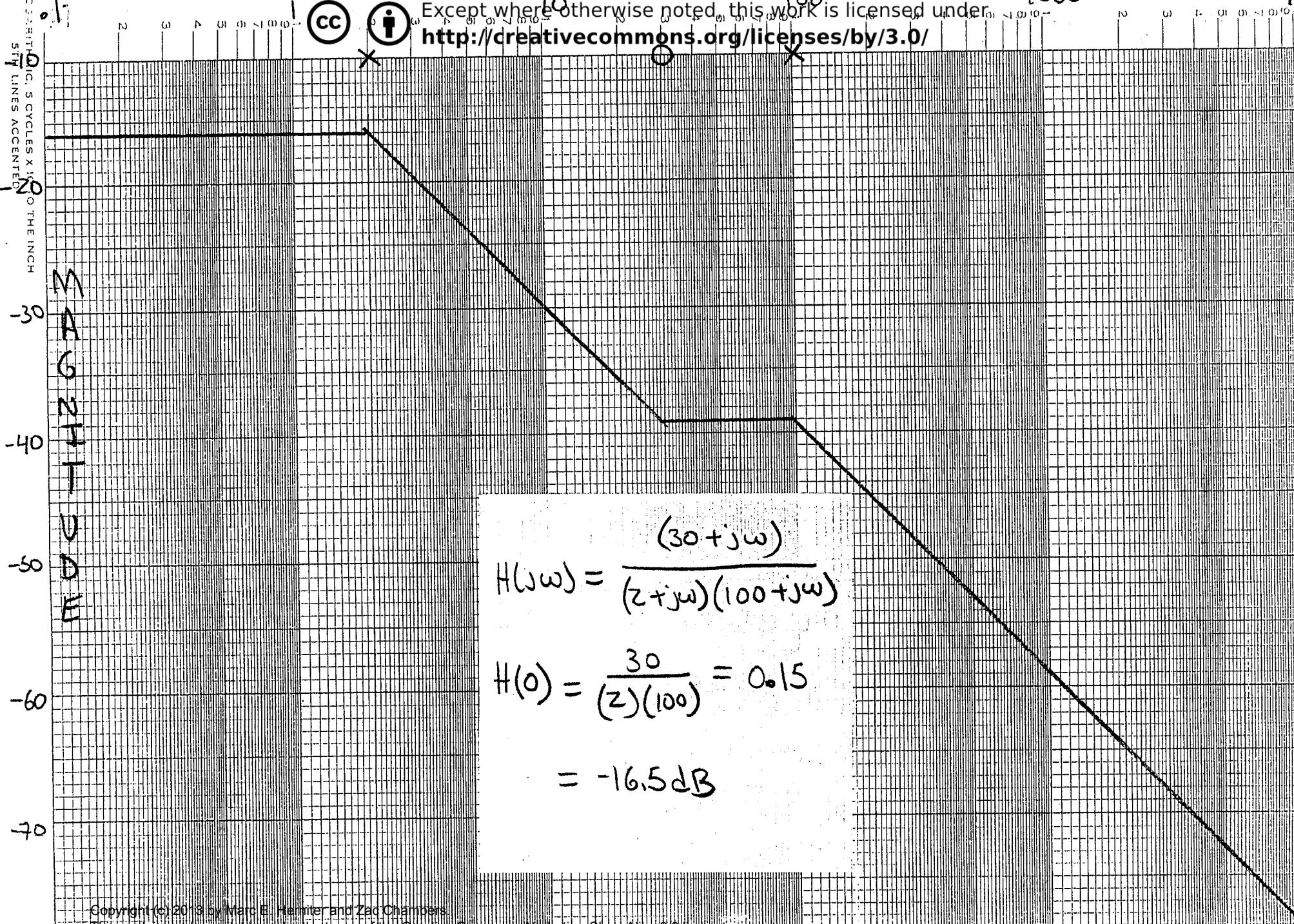
12-085  
104



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

SEM-LC JOURNALIC. 5 CYCLES X 1000 THE INCH  
STYLINES ACCENTED

M  
A  
G  
N  
I  
T  
U  
D  
E



$$H(j\omega) = \frac{(30+j\omega)}{(z+j\omega)(100+j\omega)}$$

$$H(0) = \frac{30}{(z)(100)} = 0.15$$

$$= -16.5 \text{ dB}$$

10

100

1000

104

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



-45

PHASE

-90

-135

$$H(j\omega) = \frac{(30+j\omega)}{(2+j\omega)(100+j\omega)}$$

Frequency

.01

10



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

+40

M

A

G

Z

H

T

U

D

E

PHASE

$$H(j\omega) = \frac{(1+j\omega)}{j\omega}$$

$$\text{For } \omega \ll 1 \quad H(j\omega) \approx \frac{1}{j\omega}$$

$$\Rightarrow \angle H(j\omega) = -90^\circ$$

$$\text{For } \omega = .01 \quad |H(j\omega)| \approx \left| \frac{1}{j(.01)} \right|$$

$$= 40 \text{ dB}$$

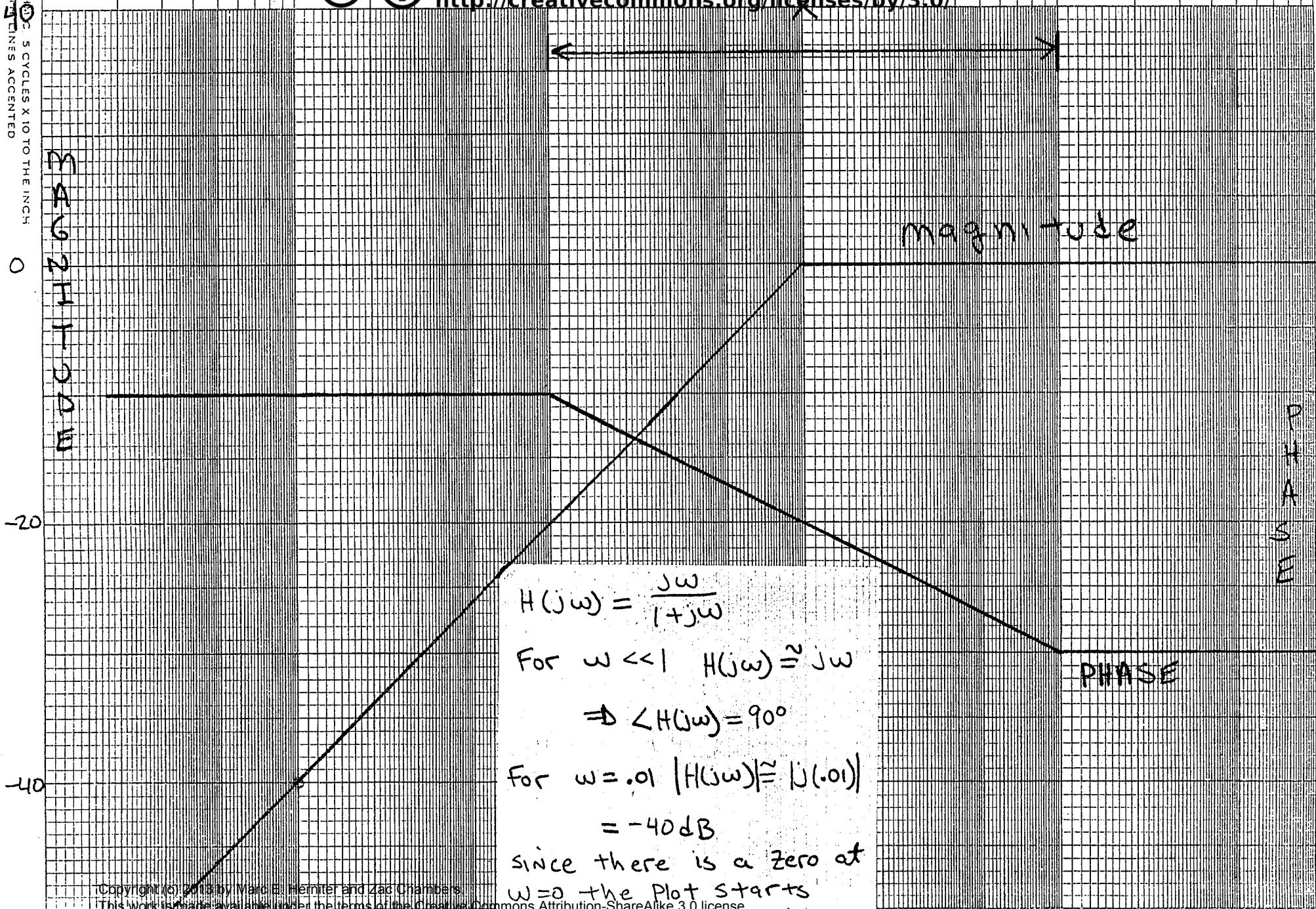
SINCE there is a pole  
at  $\omega = 0$  the plot

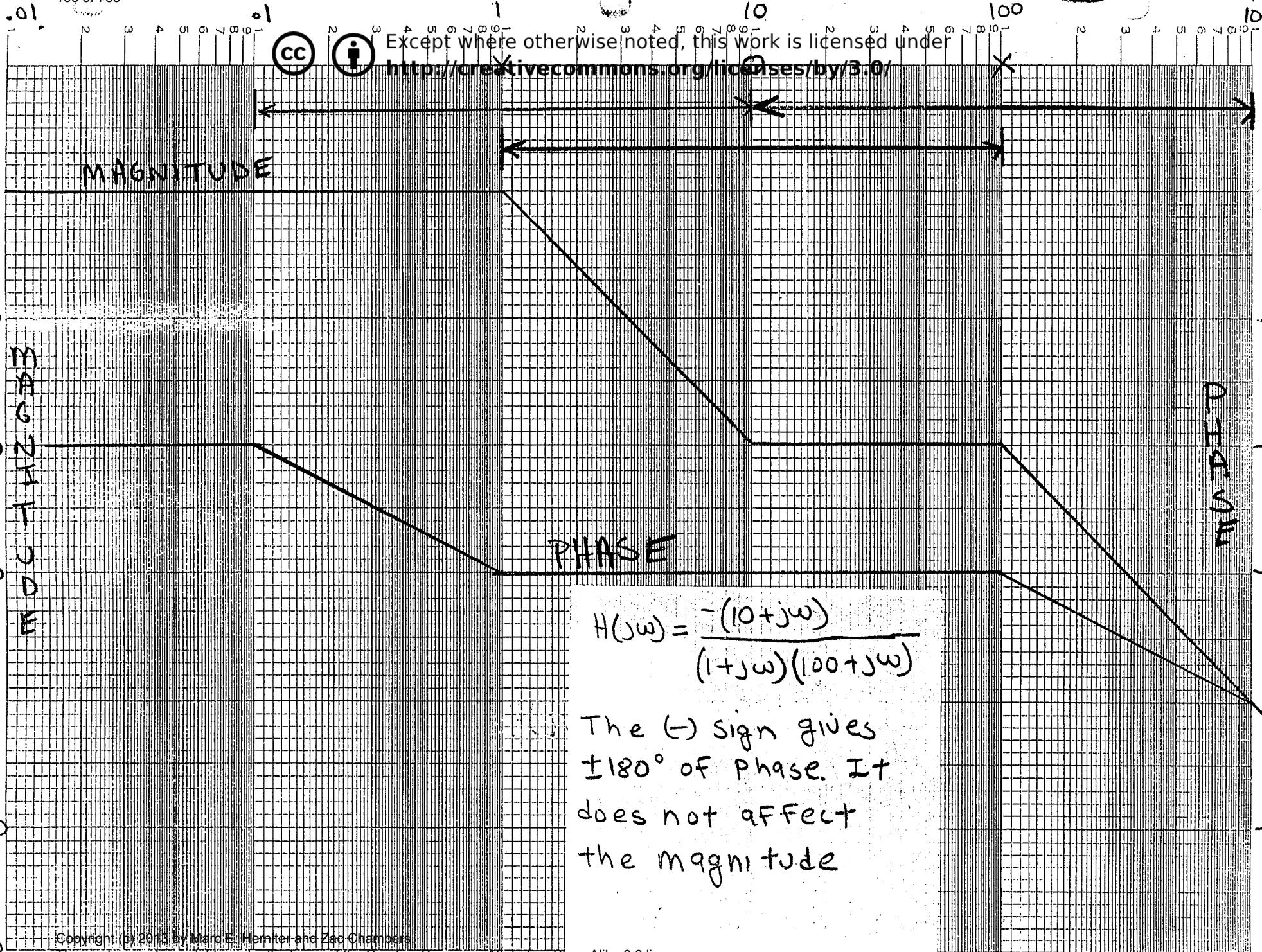
Copyright (c) 2013 by Matche Hemingway and Chambers.

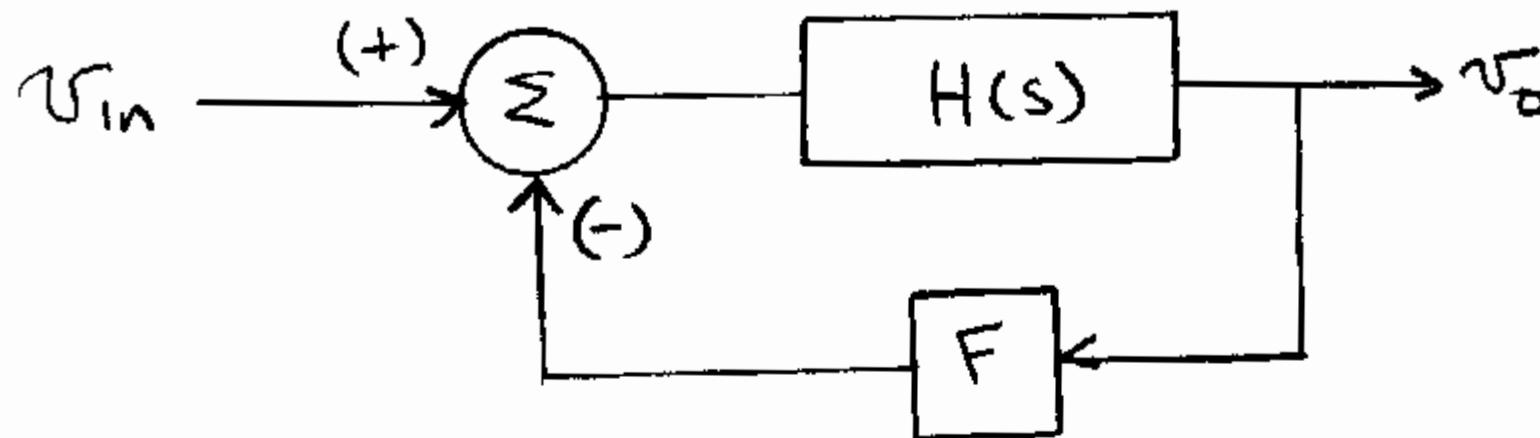
This work is made available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license,  
<http://creativecommons.org/licenses/by-sa/3.0/>.



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







The Summing junction performs the difference and accounts for the negative feedback.

$$H(s) = \frac{(+)\text{const} (z_0)(z_{e0})(\dots)}{(p_1)(p_2)(\dots)(p_n)}$$

$$0 \leq F \leq 1$$

$$\text{when } \angle FH(s) = \pm 180^\circ, \quad FH(s) = -|FH(s)|$$



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

The  $(-)$  signs will cancel and we will no longer have negative feedback. We will have positive feedback and the system will oscillate.

### For amplifier stability:

For an amplifier to be stable

$$|F H(s)| < 1 \text{ when } \angle F H(s) = \pm 180^\circ$$

### Design Rules: Gain margin

How much less than one should

$|F H(s)|$  be when  $\angle F H(s) = \pm 180^\circ$ . OR

how much greater than one should

$|F H(s)|$  be when  $\angle F H(s) = \pm 180^\circ$



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

Phase margin :

How much less than  $\pm 180^\circ$  should  $\angle F H(s)$  be when  $|F H(s)| = 1$ .

Phase margin  $\triangleq 180 - |\angle F H(s)|$  when  $|F H(s)| = 1$ .

Design rule of thumb: (this usually depends on whose thumb it is)

want gain margin  $\frac{1}{|F H(s)|} > 12 \text{ dB}$

want phase margin  $\geq 60^\circ$



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

284

2

LOGARITHMIC 5 CYCLES X 10 TO THE INCH  
576 LINES ACCENTED

$|FH(s)|_{dB}$

$OdB$

$0^\circ$

Phase  
 $FH(s)$

$180^\circ$

Gain  
margin

4 Phase =  
 $180^\circ$

(18)



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

PRINTED  
12-09-06  
BY ZAC

SEMILOGARITHMIC CYCLES X 10 TO THE INCH  
5TH LINES ACCENTED

|FH(s)|

0dB

0°

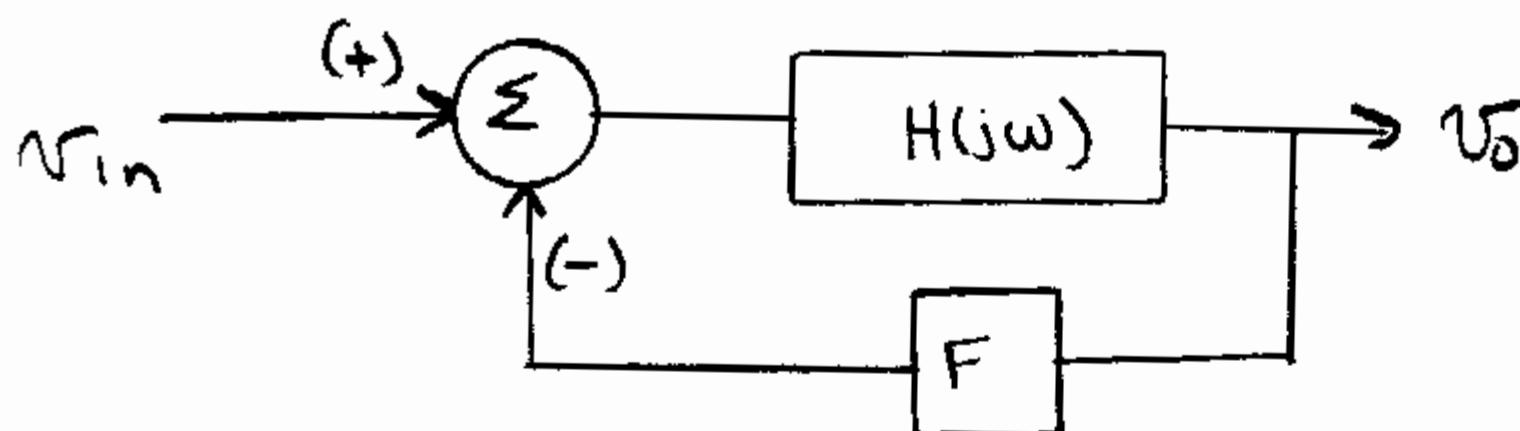
Phase  
FH(s)

180°

$\leftarrow \text{Gain} = 0$

Phase margin

(3)



$$\text{Let } H(j\omega) = \frac{10^3}{(1 + \frac{j\omega}{10^6})(1 + \frac{j\omega}{5 \times 10^6})(1 + \frac{j\omega}{10^7})}$$

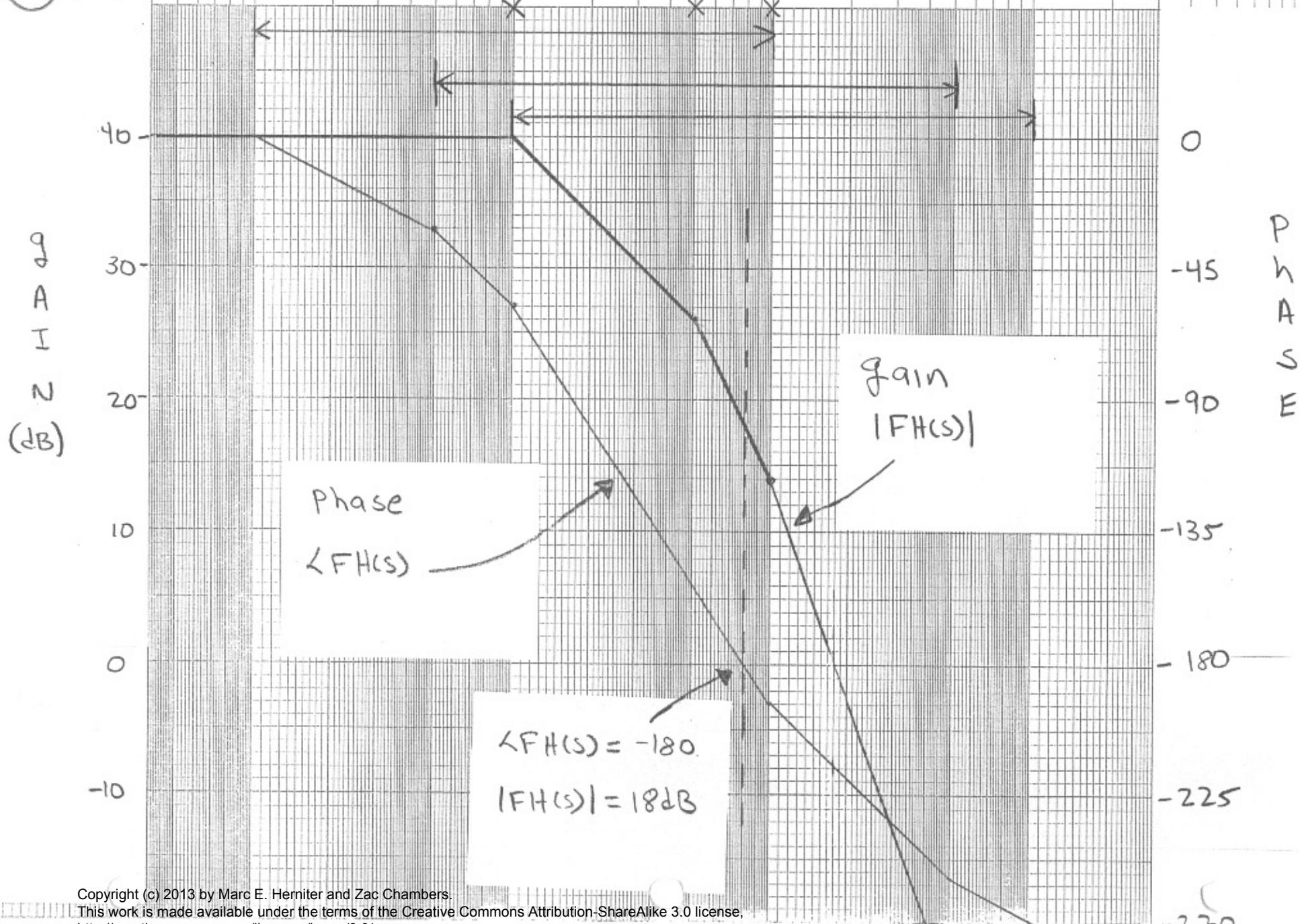
This amplifier has a low frequency gain of 1000, and poles at  $\omega = 10^6$ ,  $10^7$ , and  $5 \times 10^6$  rad/s.

$$\text{Let } F = \frac{1}{10}, \text{ Then}$$

$$FH(j\omega) = \frac{100}{\left(1 + \frac{j\omega}{10^6}\right)\left(1 + \frac{j\omega}{5 \times 10^6}\right)\left(1 + \frac{j\omega}{10^7}\right)}$$

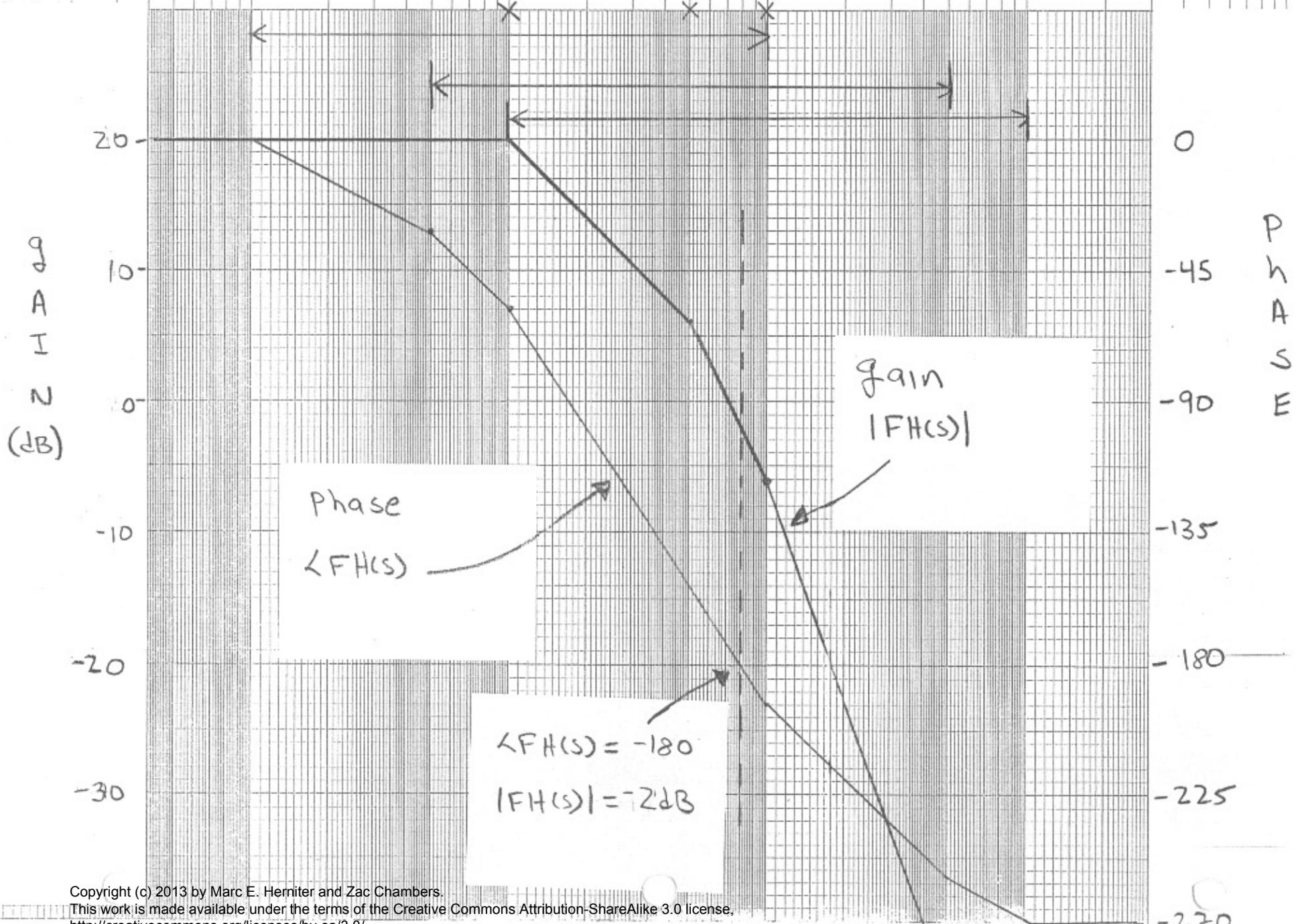
Gain and Phase for  $FH(s)$ ;  $F = 1/10$ 

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



Gain and Phase for  $FH(s)$ ;  $F = 1/100$ 

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/>

Magnitude Plotting

$$20 \log_{10} |H(s)| - 20 \log_{10} \left| \frac{1}{F} \right| =$$

$$20 \log_{10} |H(s)| + 20 \log_{10} |F| =$$

$$20 \log_{10} |FH(s)|$$

so

$$|FH(s)|_{dB} = |H(s)|_{dB} - \left| \frac{1}{F} \right|_{dB}$$

We are usually concerned with

$$|FH(s)|_{dB} = 0 \quad \text{or}$$

$$|H(s)|_{dB} - \left| \frac{1}{F} \right|_{dB} = 0$$

OR

$$|H(s)|_{dB} = \left| \frac{1}{F} \right|_{dB}$$

so when  $|FH(s)|_{dB} = 0$ ,  $|H(s)|_{dB} = \left| \frac{1}{F} \right|_{dB}$



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

So we have

$$H(s) = \frac{10^3}{\left(1 + \frac{j\omega}{10^6}\right)\left(1 + \frac{j\omega}{5 \times 10^6}\right)\left(1 + \frac{j\omega}{10^7}\right)}$$

a good question is "what range of  $\omega$  can we have such that the amplifier will be stable.

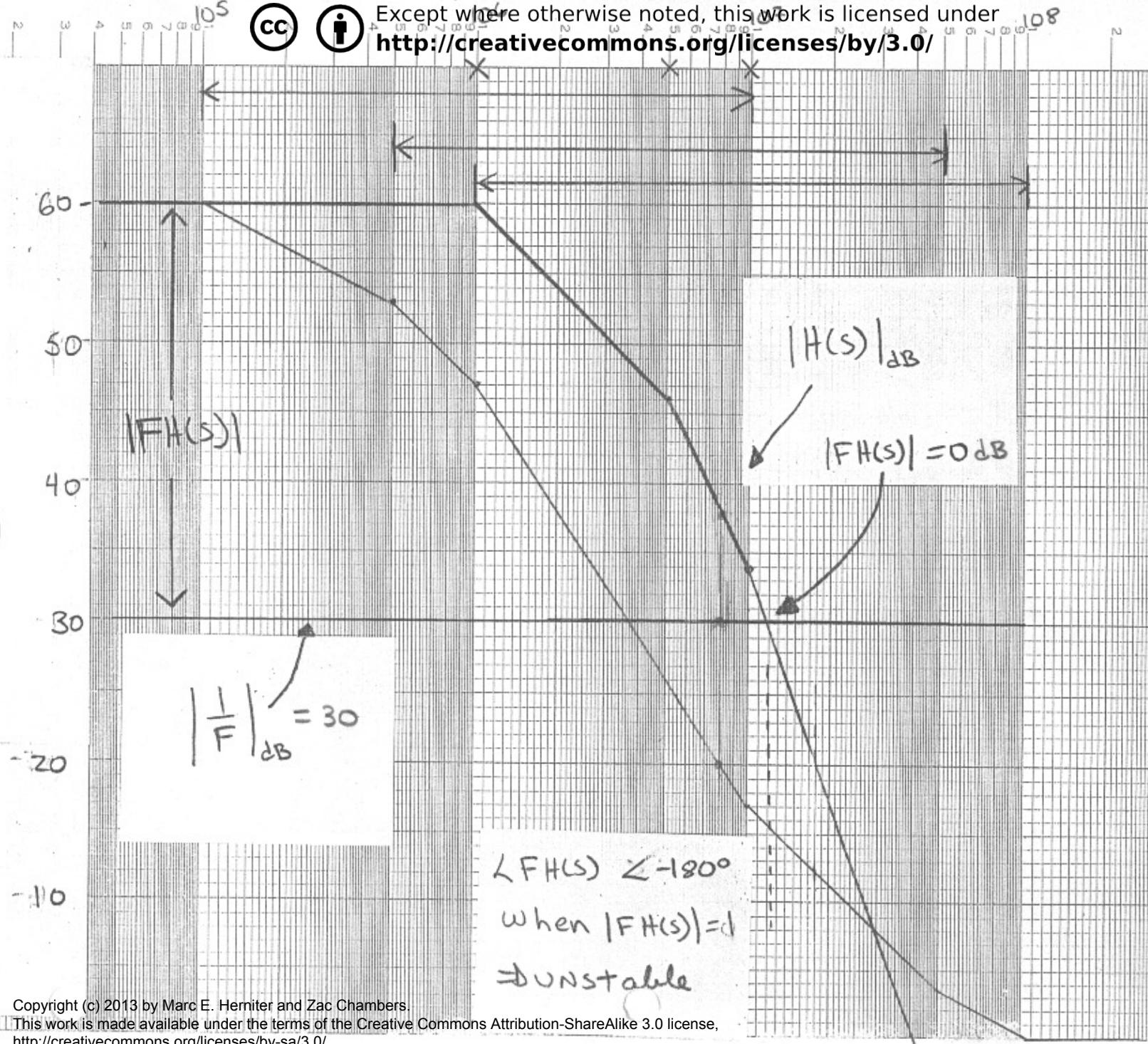
Gain and Phase for  $H(s)$ 

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

487

GAIN (dB)

-10 0 10 20 30 40 50 60

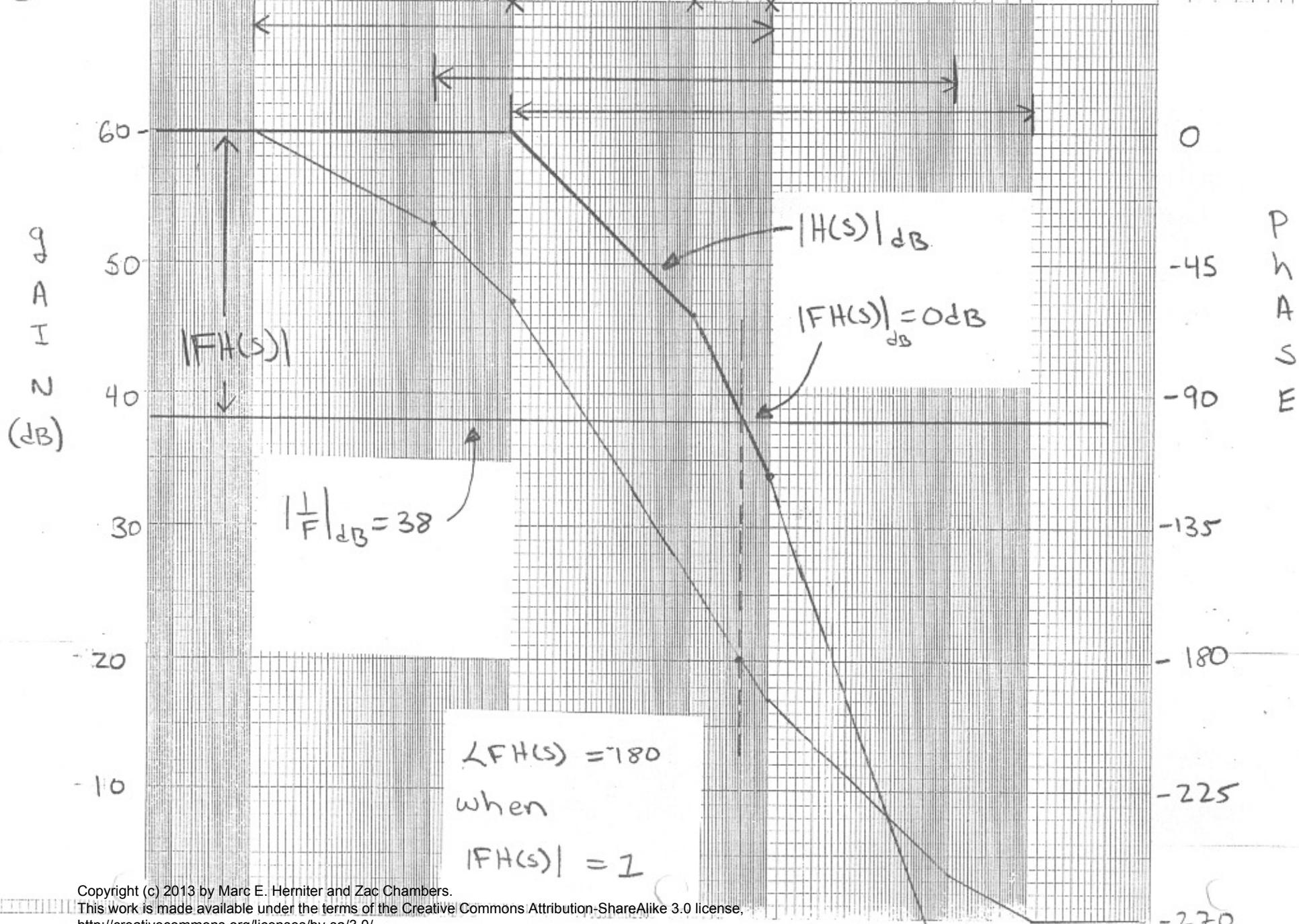


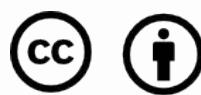
Gain and Phase for  $H(s)$ 

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

488

12.083





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

489

This amplifier will be stable

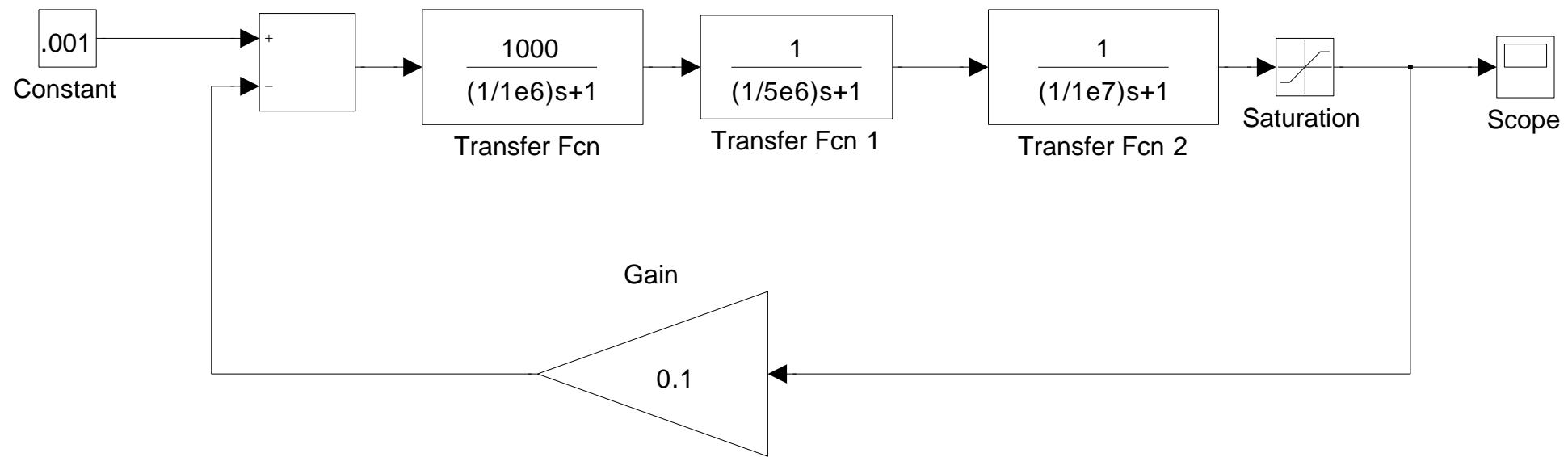
for  $| \frac{1}{F} |_{dB} > 38$

or  $| \frac{1}{F} | > 79.4$

or  $F < .0125$



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



Configuration Parameters: Stability\_Example/Configuration (Active) X

Select: 181 of 783

Solver

Data Import/Export

Optimization

Diagnostics

- Sample Time
- Data Validity
- Type Conversion
- Connectivity
- Compatibility
- Model Referencing
- Hardware Implementation
- Model Referencing

Real-Time Workshop

- Comments
- Symbols
- Custom Code
- Debug
- Interface

Simulation time

Start time: 0.0 Except where otherwise noted, this work is licensed under Stop time: End of  
<http://creativecommons.org/licenses/by/3.0/>

Solver options

Type: Variable-step Solver: ode45 (Dormand-Prince)

Max step size: 10e-9 Relative tolerance: 1e-3

Min step size: auto Absolute tolerance: auto

Initial step size: auto

Zero crossing control: Use local settings

Automatically handle data transfers between tasks

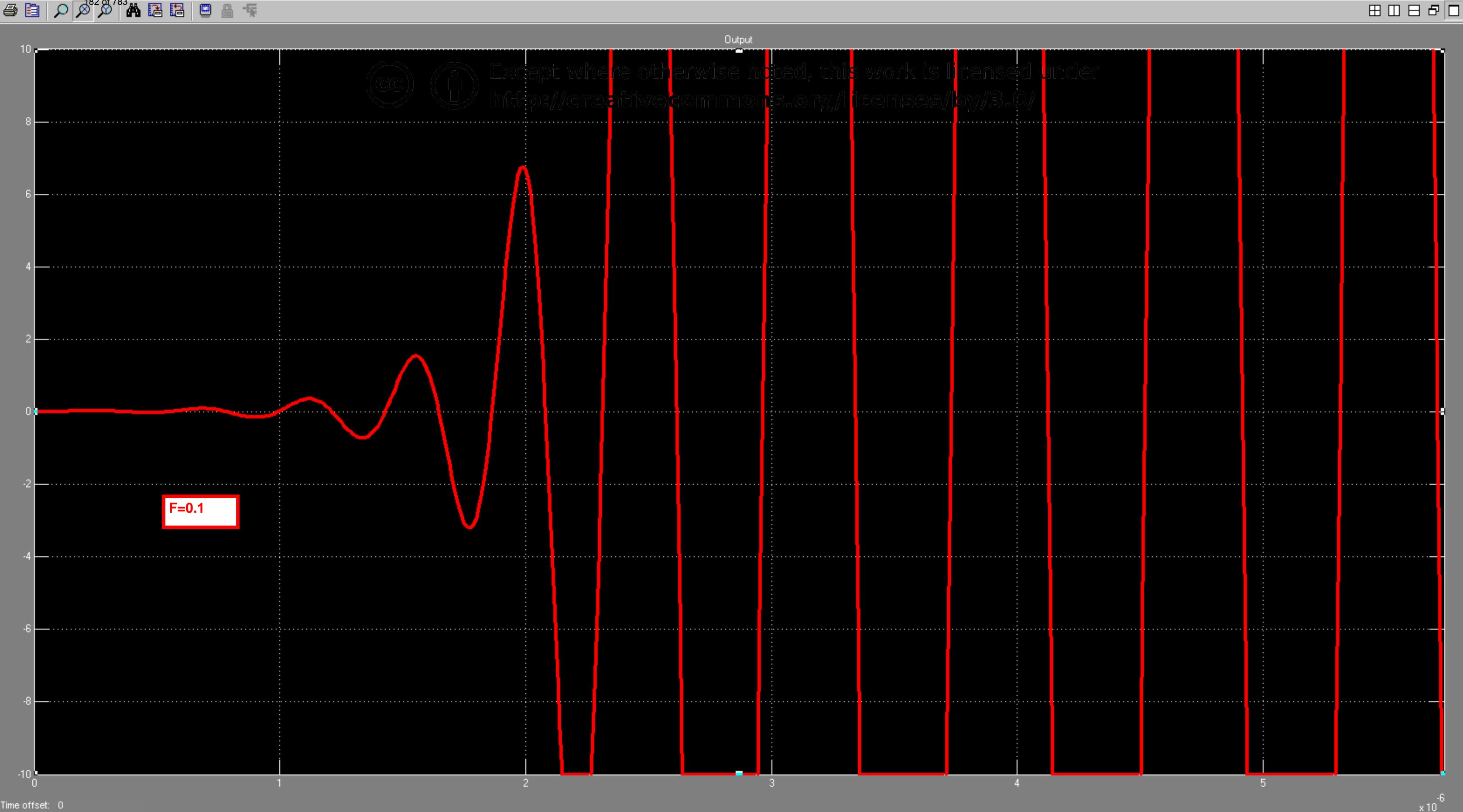
Solver diagnostic controls

Number of consecutive min step size violations allowed: 1

Consecutive zero crossings relative tolerance:  $10^{128} * \text{eps}$

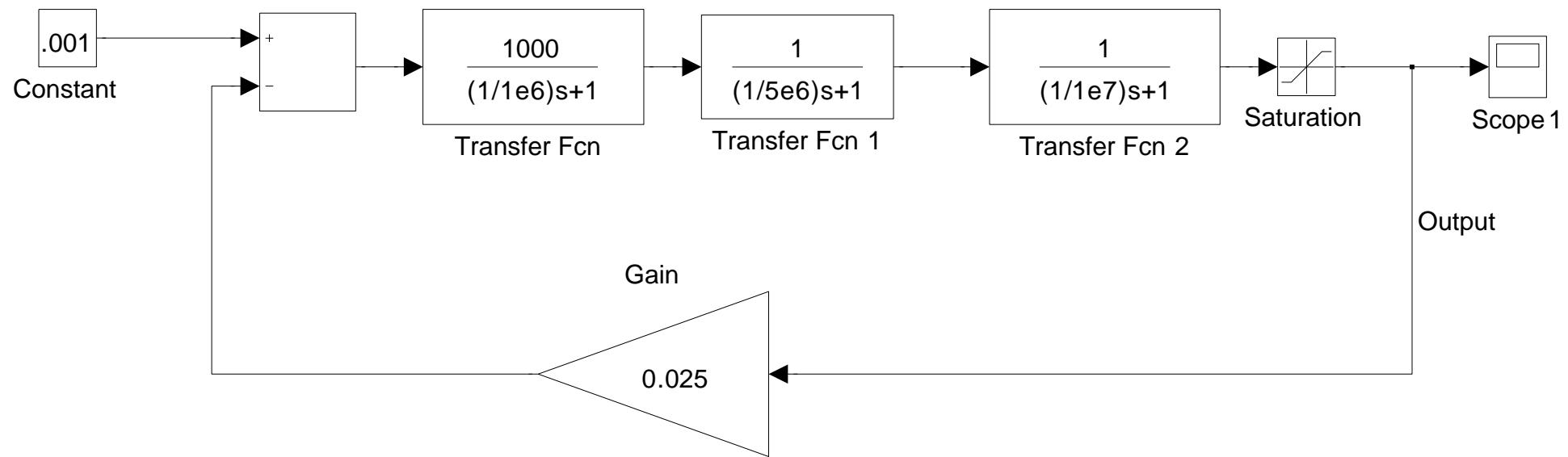
Number of consecutive zero crossings allowed: 1000

Copyright (c) 2013 by Marc E. Herniter and Zac Chambers.  
This work is made available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license,  
<http://creativecommons.org/licenses/by-sa/3.0/>.





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

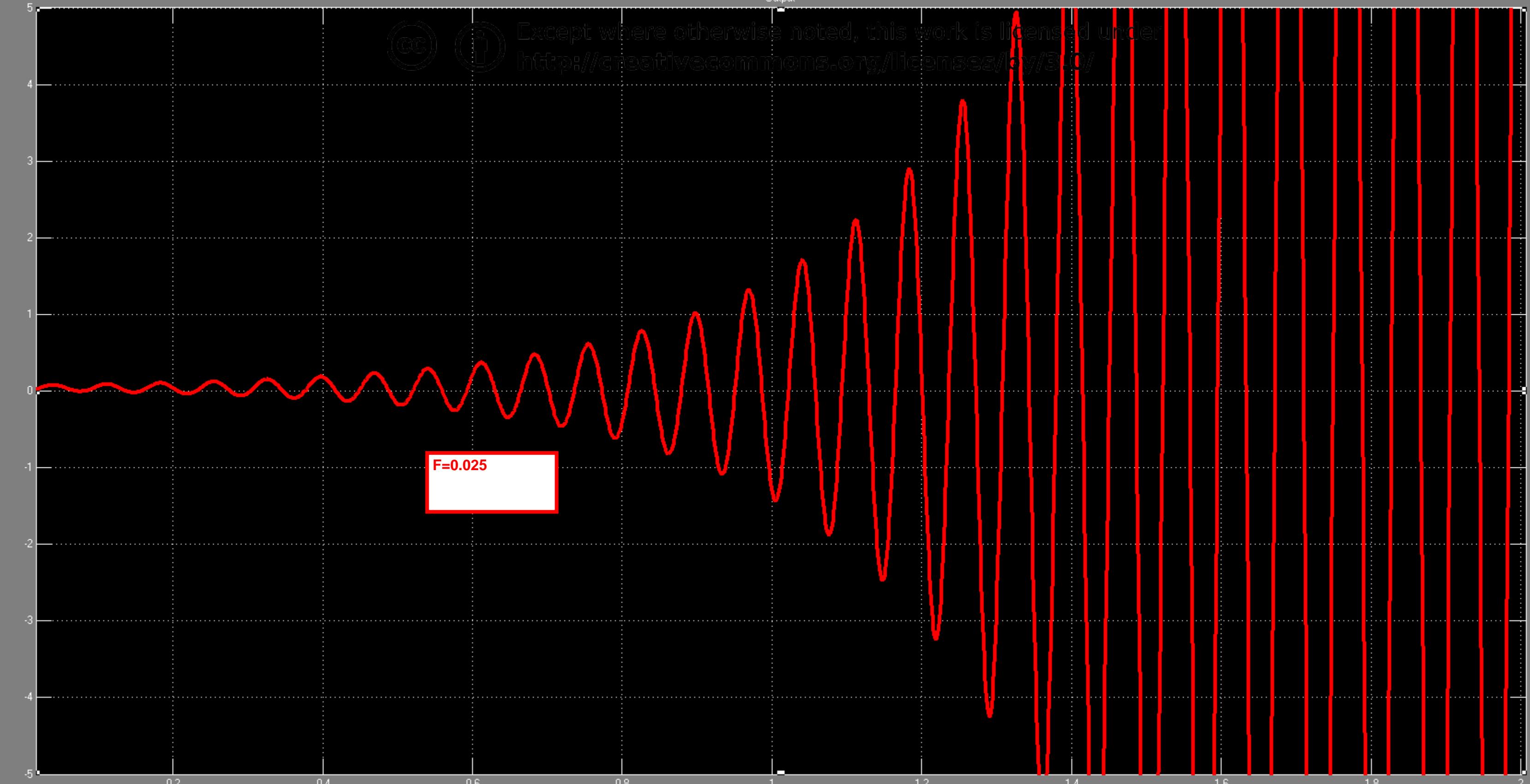




Output

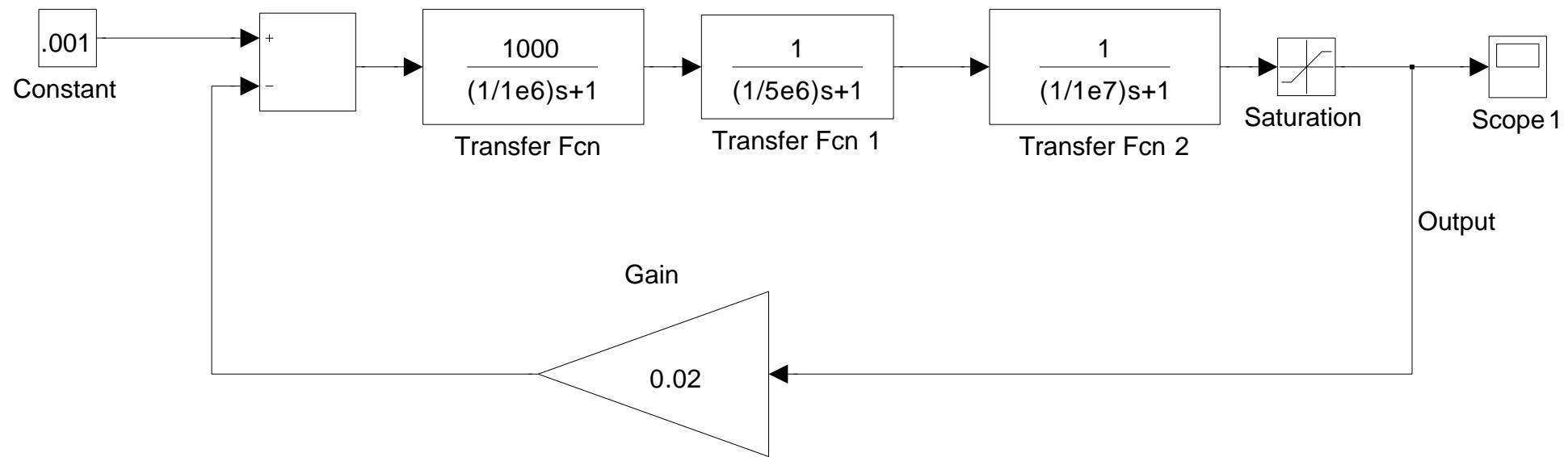


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





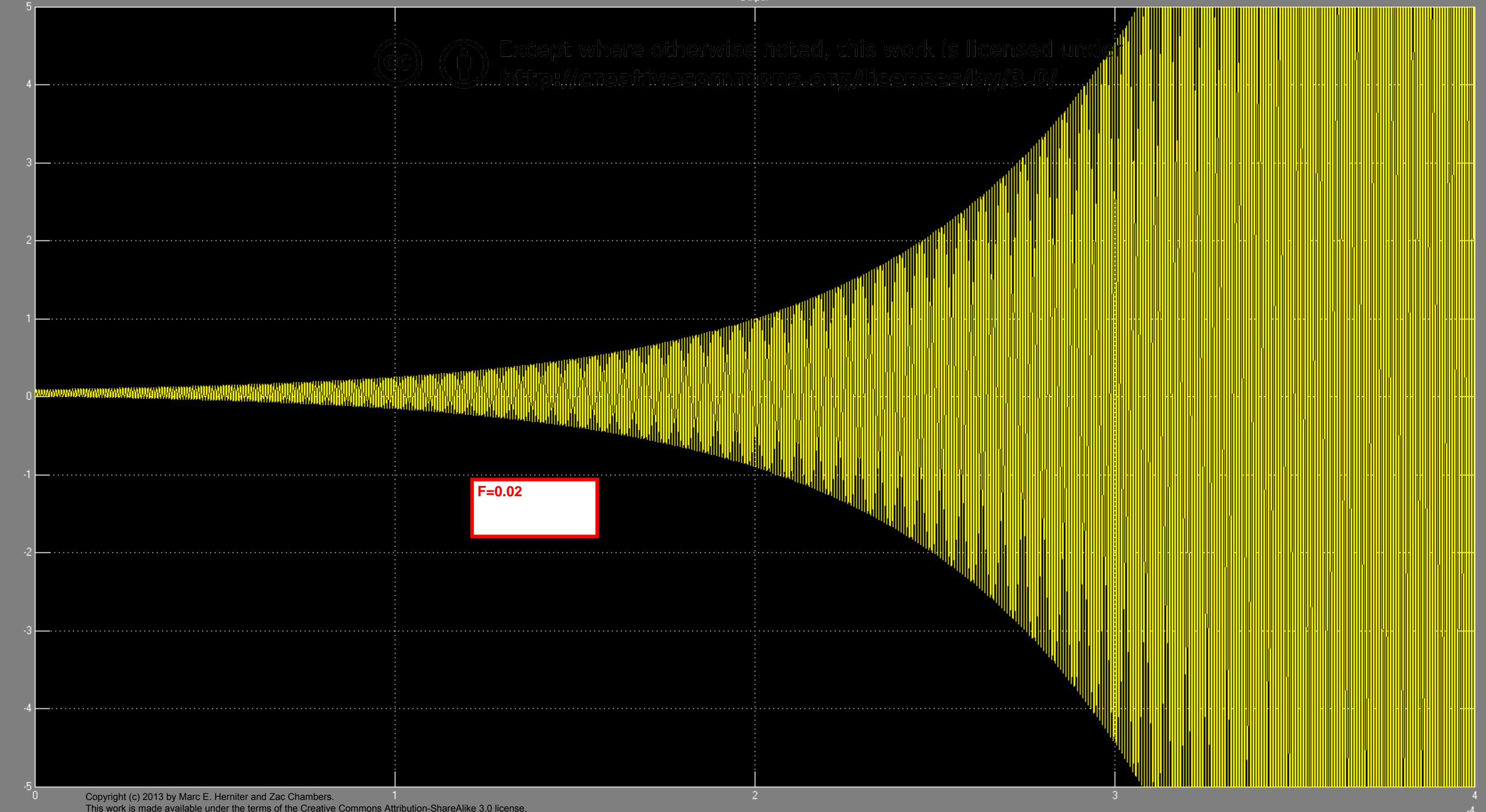
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



Output

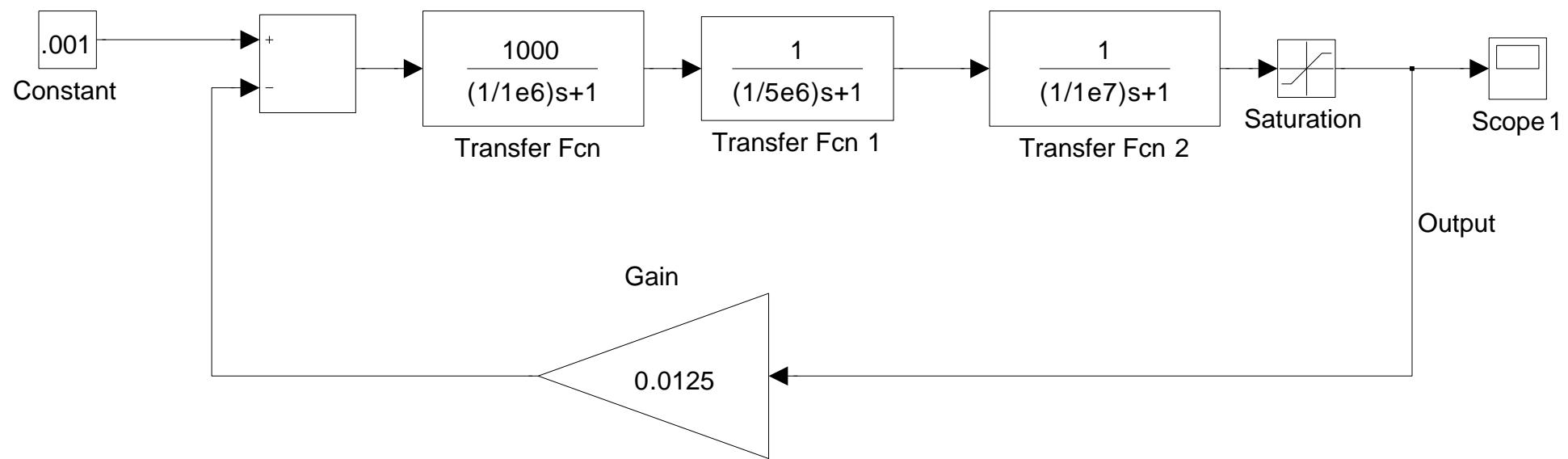


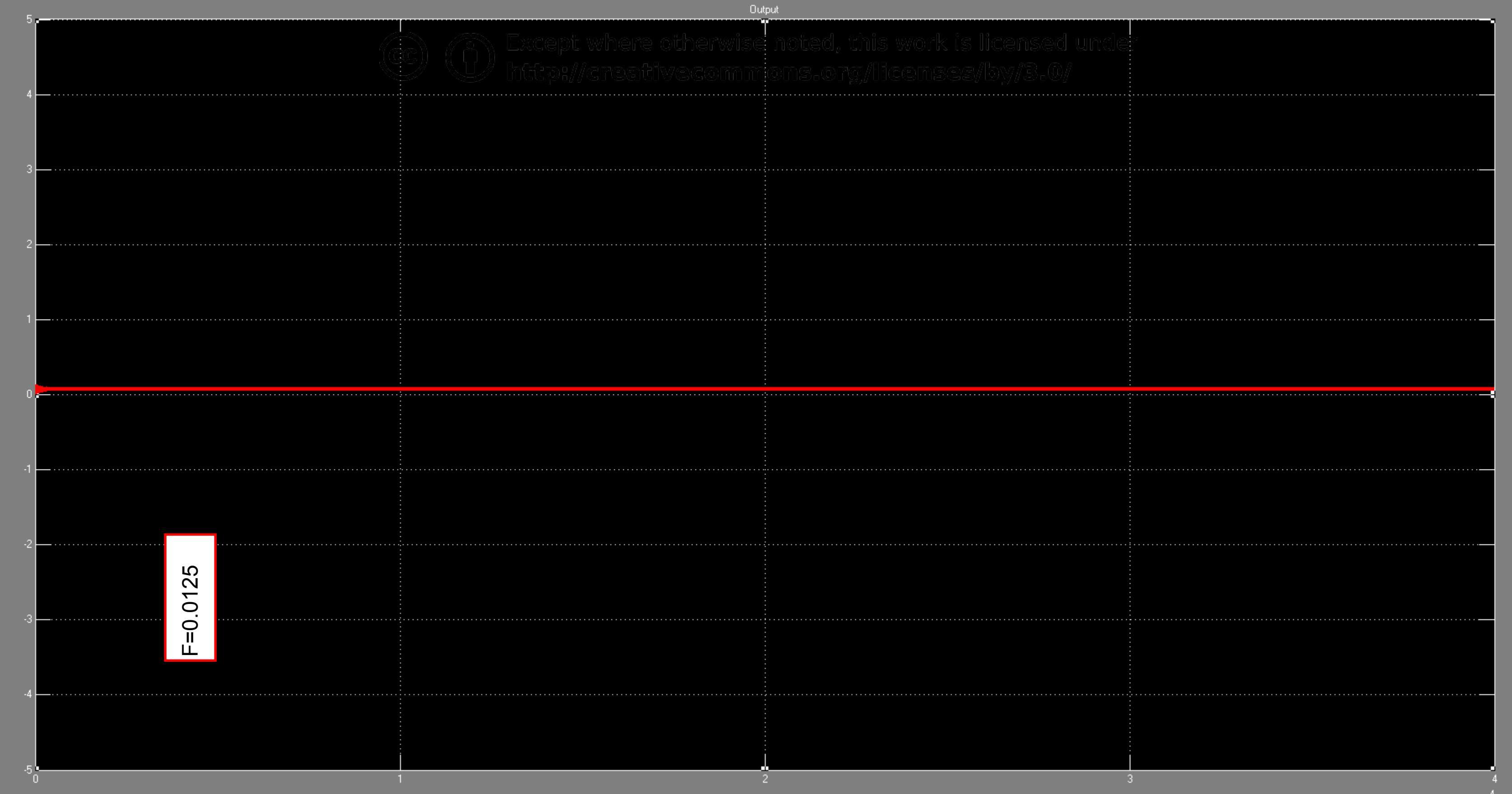
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





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







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



## Introduction to Model-Based Systems Design

### Lecture 4 : Plant Model Transfer Function



## Plant Model – Transfer Function

2

- To use the tools covered in a classical control class, we need to determine the transfer function of the plant model.
- We will do this by measuring the magnitude response versus frequency of the plant input-output transfer function. (This will be a Bode magnitude plot.)
- We could also measure the phase plot, but we will not do this.





3

## Plant Model – Transfer Function

- We must first bias the plant to operate around the desired operating point.
- In our case, we'll say that we want to use the motor-generator system near a frequency of 1800 rpm.
- We will determine the bias needed to obtain this rpm.
- We will use the plant with a fixed load and remove the controller
- The example we show will be for the original plant without the flywheel. This plant will be very fast and have a pole at a high frequency.
- For an exercise, we will repeat the analysis with the same model and add the flywheel. The bode plot should be similar with the poles located at lower frequencies.



## Plant Model – Transfer Function

4



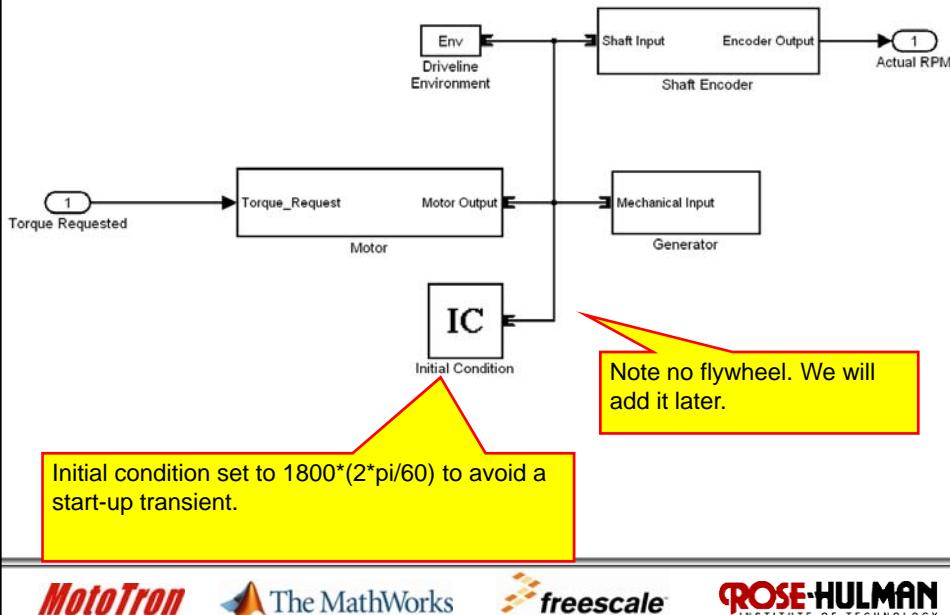
- The constant input is the value of the input needed to achieve an rpm of 1800 at the output.
- The next few screens show the models I used to generate the magnitude plot. You should use the improved models..





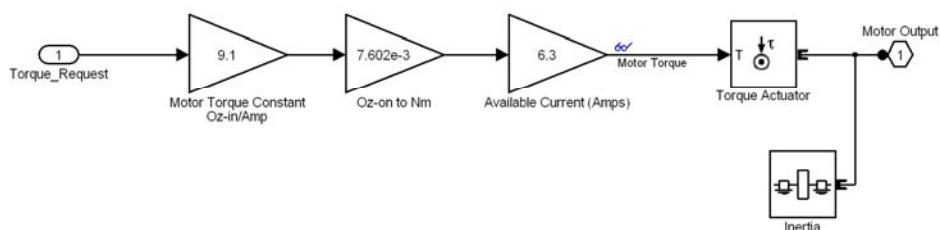
## Plant Model

5

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

## Motor Model

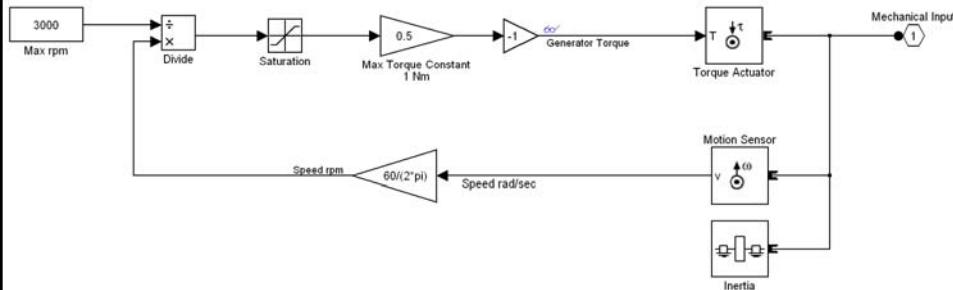
6

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



## Generator Model

7



**MotoTron**

The MathWorks

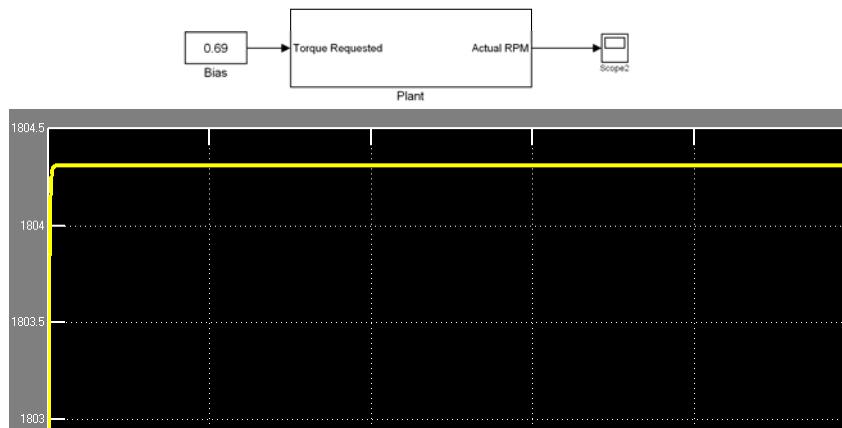
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Plant Model – Transfer Function

8

- With a constant input of 0.69, the motor-generator system runs at about 1800 rpm:



**MotoTron**

The MathWorks

**freescale**  
semiconductor

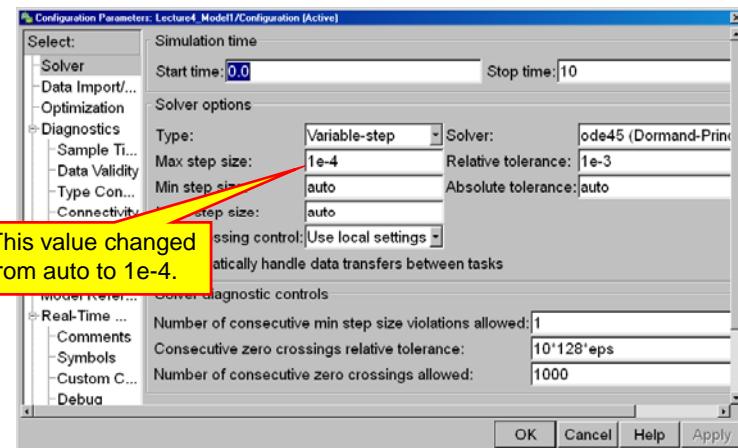
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Plant – Transfer Function

9

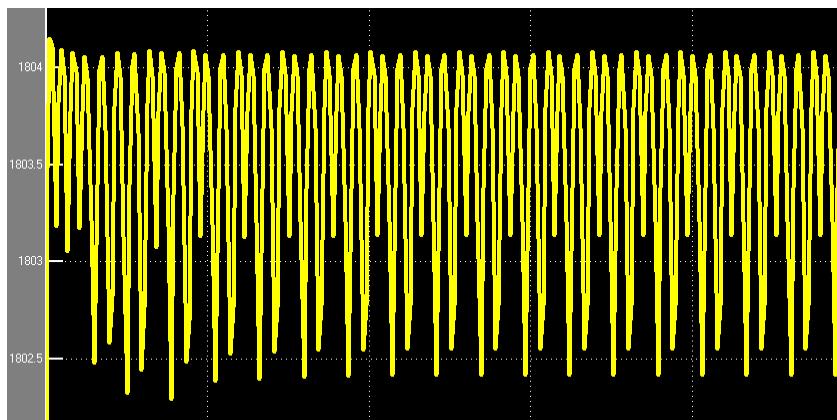
- The previous simulation required a small step size to prevent numerical oscillations in the output:



## Plant – Transfer Function

10

- Had we left the max step size at auto, we would see the following output:

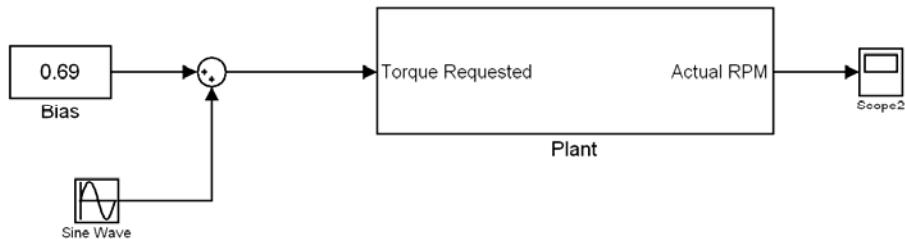




11

## Plant – Transfer Function

- Now that we have our plant biased at 1800 rpm, we will introduce a small-signal sine wave at the input and measure the magnitude of the output at various frequencies:



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

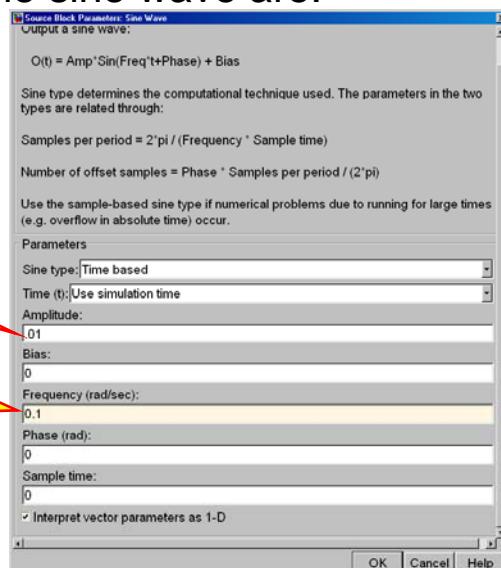
12

## Plant – Transfer Function

- The properties of the sine wave are:

Amplitude set to a small variation. This is small compared to the bias value of 0.69. We will not change the amplitude (0.01) in our analysis.

We will start with a low frequency. We will change this value each time we run a simulation.





## Plant – Transfer Function

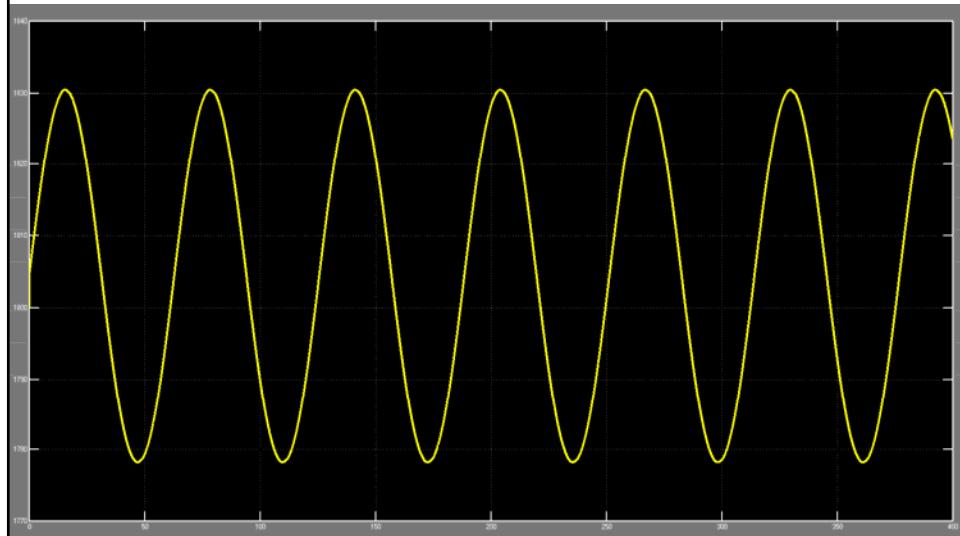
13

- We will let the simulation run for 400 seconds (since the frequency is so low).
- We will set the maximum step size to 1e-3 so that the simulation does not take too long to run (because the simulation will run for 400 seconds of simulation time).
- We see the following output waveform:



## Plant Transfer Function

14





15

## Cursors

- We can measure the waveform using cursors.
- The cursors are not normally displayed on a scope plot.
- To display standard plotting menus on a scope, run the following Matlab commands

```
shh = get(0,'ShowHiddenHandles');
set(0,'ShowHiddenHandles','On')
set(gcf,'menubar','figure')
set(gcf,'CloseRequestFcn','closereq')
set(gcf,'DefaultLineClipping','Off')
set(0,'ShowHiddenHandles',shh)
```



16

## Cursors

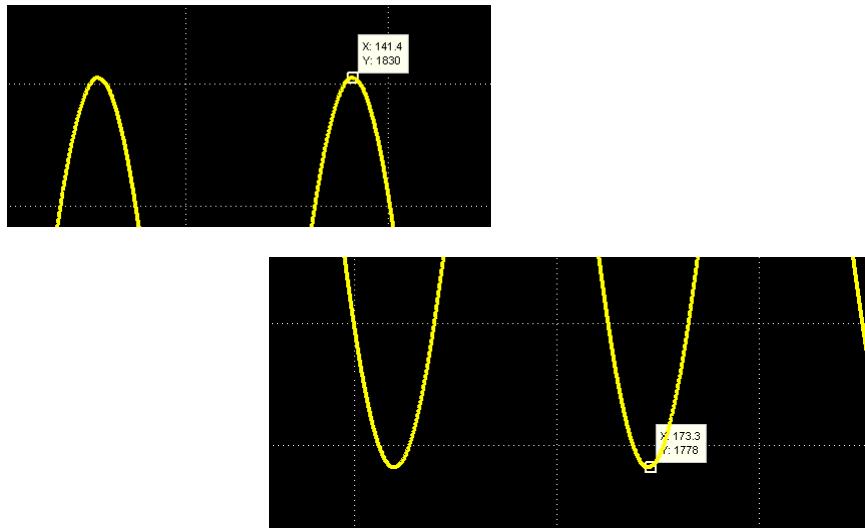
- These commands will cause the File menu to be displayed on the scope screen.
- To display the cursors, you can then select **Tools** and then **Data Cursor** to display the cursors.
- When you click the mouse on a trace, a square will be displayed with the points coordinates.





## Cursor Values

17



The MathWorks



## Plant – Transfer Function

18

- For an input frequency of 0.1 rad/sec, we see the following:
- Peak-to-peak output is 1830 rpm – 1778 rpm = 52 rpm.
- Output amplitude = 26 rpm.
- Gain = Output amplitude divided by input amplitude =  $(26 \text{ rpm})/(0.01) = 2600$ .
- $\text{Gain}_{\text{db}} = 20 * \log_{10}(\text{Gain}) = 68.3 \text{ dB}$



The MathWorks





## Plant – Transfer Function

19

- We will measure the gain at several different frequencies.
- The input amplitude will be kept constant.
- We will change the length of the simulation and the max step size as appropriate for higher frequencies.



### Frequency (rad/sec)

### Output Amplitude

### Gain

### Gain (dB)

0.1

26

2600

68.3

1

26

2600

68.3

10

26

2600

68.3

20

26

2600

68.3

40

25.5

2550

68.1

80

24

2400

67.6

100

23

2300

67.2

200

17.5

1750

64.9

300

13.5

1350

62.6

400

11

1100

60.8

1000

4.5

450

50.1

10000

0.5

50

34.0



20



21

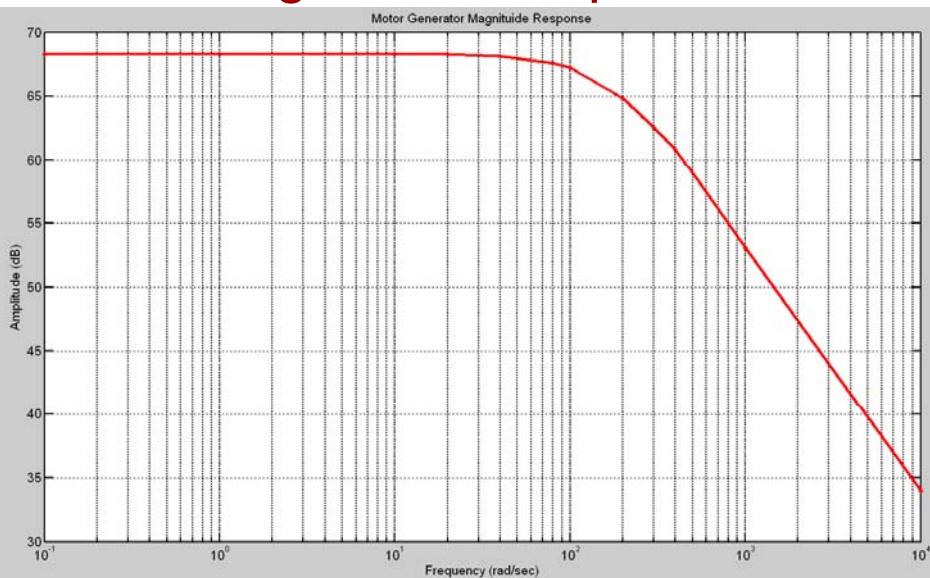
## Plot with Matlab

- Amplitude=0.01;
- freq=[.1 1 10 20 40, 80 100 200 300 400 1000 10000];
- output=[26, 26, 26, 26, 25.5, 24 23, 17.5 13.5 11 4.5, 0.5];
- mag=output/Amplitude;
- dB=20\*log10(mag);
- semilogx(freq,dB,'rx-');
- grid on
- title('Motor Generator Magnitude Response');
- ylabel('Amplitude (dB)');
- xlabel('Frequency (rad/sec)');



22

## Plant Magnitude Response





23

## Plant Transfer Function

- Our motor-generator system has a single pole between 100 and 200 rad/sec.
- Note that a single poles system cannot oscillate because we only have 90 degrees of phase (1 pole).
- So why did we see oscillations at when we had large proportional gain?
  - There must be extra delay somewhere.



The MathWorks



24

## Lecture 4 Problem 1

Demo \_\_\_\_\_

- Repeat the bode plot for our system and include the flywheel inertia of  $1.041 \times 10^{-4} \text{ kg}\cdot\text{m}^2$ .
- Fill in the table below and generate a plot.

Frequency (rad/sec)	Output Amplitude	Gain	Gain (dB)
0.1			
1			
2			
4			
8			
10			
20			
40			
80			
100			
1000			
10000			



25

## Lecture 4 Problem 2

A plant has the following transfer function

$$\frac{1000}{(1 + s/0.1)(1 + s)(1 + s/100)}$$

We will use this plant with proportional feedback as shown in class.

- a) Use gain and phase plots to find the largest value of the feedback constant F that can be used to have a stable system with zero degrees phase margin.
- b) Use Simulink to show that the system is unstable for values of F larger than this value of F.
- c) Use Simulink to show that the system is stable for values less than or equal to this value of F.
- d) Use gain and phase plots to find the largest value of the feedback constant F that can be used to have a stable system with a 60 degrees phase margin.

Grade \_\_\_\_\_



The MathWorks





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



## Introduction to Model-Based Systems Design

### Lecture 5 : Improving the Plant Model



2

## Improved Model - Friction

- To this point, our system has been frictionless.
- We could see this as a problem if we give the motor an initial velocity, set the desired velocity to 0, and set the bulb load to zero.
- The motor would never apply a torque, but the shaft would continue to spin indefinitely.
- This is obviously not correct.





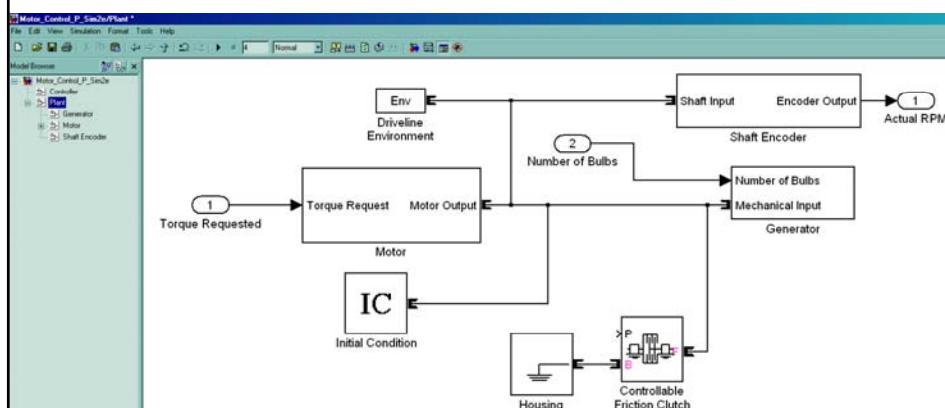
3

## Improved Model - Friction

- The easiest way to add friction is to incorporate a Friction Clutch into the plant.
- Go to the plant level and drag in a **Controllable Friction Clutch** and a **Housing**.
  - Simscape / Dynamic Elements / Controllable Friction Clutch**
  - Simscape / Solvers & Inertias / Housing**

4

## Improved Model - Friction





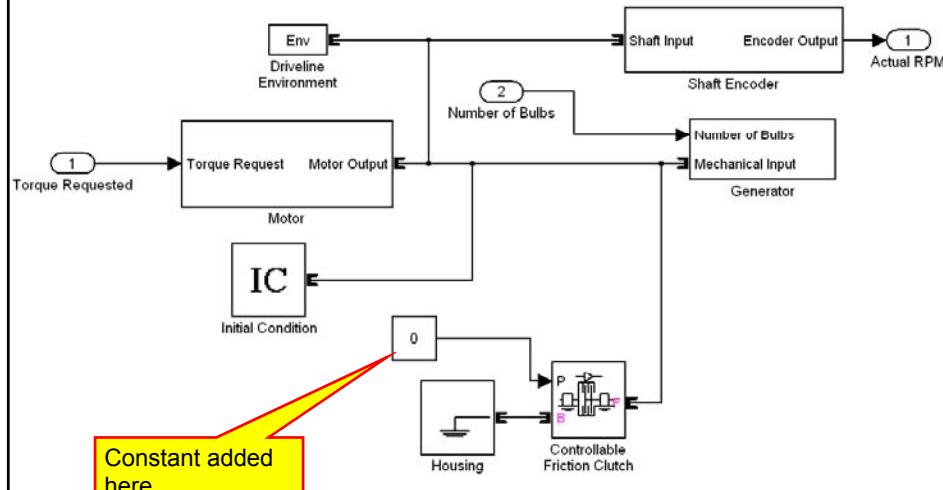
5

## Improved Model - Friction

- The base of our clutch is attached to a housing to lock it in place.
- The follower is connected to the driveline.
- The pressure port regulates how much the clutch is engaged
  - 0 : not engaged, no friction
  - 1 : fully engaged, full friction
    - System might not rotate
    - Or have the full drag force applied
- Add a constant to regulate the pressure

6

## Full Plant Model





7

## Improved Model - Friction

- Now we need to set up our clutch – double click on it and change following
  - Directionality : unidirectional
  - Number of friction surfaces : 1
  - Effective torque radius : 1
  - Peak normal force : 1
- Our clutch is unidirectional because our shafts only spin one way
- The other parameters are unchanged



## Clutch Friction

- The peak torque that the clutch can apply is equal to the **number of friction surfaces** times the **torque radius** times the **peak normal force**.
- For our case, the peak torque will be 1 Nm. (More than our motor can supply.)
- The actual torque applied is this peak torque times the pressure input (a number between 0 and 1).

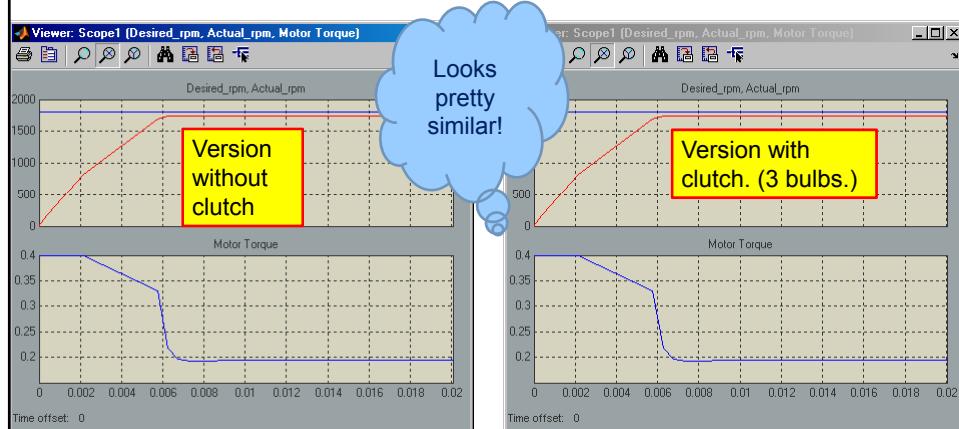




9

## Improved Friction - Verification

- For our first test, set the pressure to zero and verify that nothing changed from our last model.



10

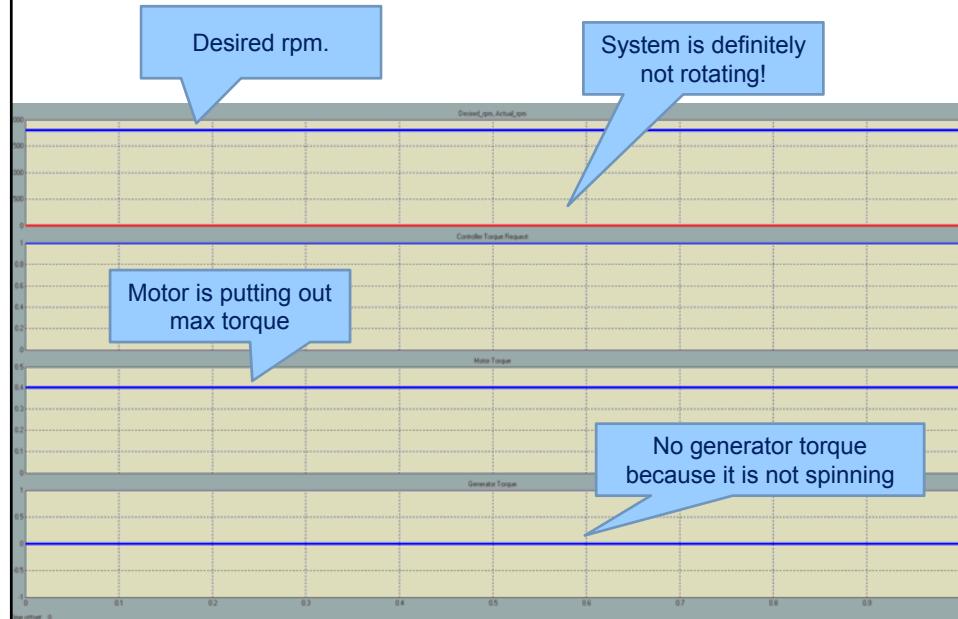
## Improved Friction - Verification

- Now set the pressure to one and the number of bulbs to zero and verify that the system does not rotate. (We expect this because the friction torque is greater than the maximum motor torque.)



11

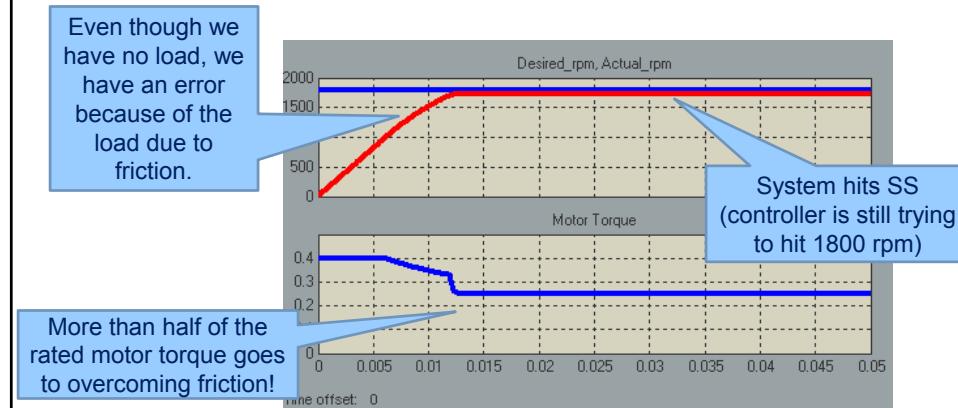
## Improved Friction - Verification



12

## Improved Friction - Verification

- Set the load to zero bulbs and the desired rpm to 1800. Zero initial condition.
- Set the pressure to 0.25 and execute.





13

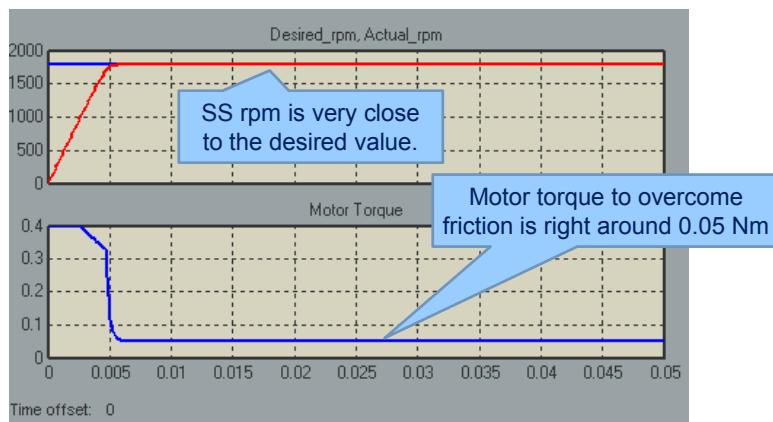
## Improved Friction - Validation

- Our friction appears to be too high
  - Over half of the motor's rated torque goes to overcoming it!
- Adjust the pressure such that about 0.05 Nm of torque are required to overcome the friction.

14

## Improved Friction - Validation

- Setting the pressure to 0.05 works well





15

## Lecture 5 Exercise 1

- Determine the value of the pressure to the friction clutch so that the system coasts down from 2700 rpm to zero in about 130 ms.

Demo\_\_\_\_\_



16

## Plant Model – Coast Down Test

- In vehicle design, a data point for verifying the accuracy of the mechanical model of the vehicle is a coast-down test.
- With zero wind, run the vehicle at 60 mph.
- Set the engine and motors to zero torque and plot the vehicle speed as it coasts down to zero.
- Tests and measures aerodynamic drag and vehicle frictional losses.
- Calibrate your model to match the measured coast-down.





## Plant Model – Coast Down

17

- For our motor-generator system, we will set the motor-generator rpm to a preset value.
- With no load, we will time how long it takes the rpm to reach zero.
- We will fit the value of the clutch friction in the model so that the model coast-down time matches the measured coast-down time.
- We will use a MathWorks tool to automatically find the correct value of clutch pressure.

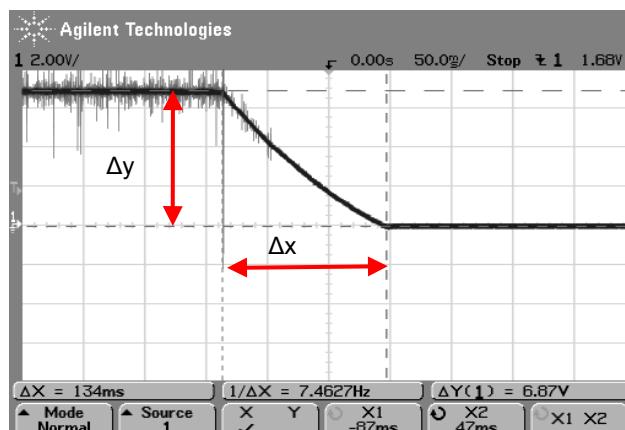


## Measured Coast-Down

18

• Tach scaling = 2.5 V per 1000 rpm

$$\begin{aligned}\Delta rpm &= \Delta y \left( \frac{1000 \text{ rpm}}{2.5 \text{ V}} \right) \\ &= (6.87 \text{ V}) \left( \frac{1000 \text{ rpm}}{2.5 \text{ V}} \right) \\ &= 2748 \text{ rpm}\end{aligned}$$





## Measured Coast-Down

19

- We will use the pressure value of the friction clutch to match the measured coast-down time to the simulated coast-down time.
- Instead of searching for the clutch pressure manually, we will use the Simulink Response Optimization toolbox.
- First, we need to measure points from the measured rpm trace.
- Measured points are shown next:



## Motor-Generator Coast Down

20

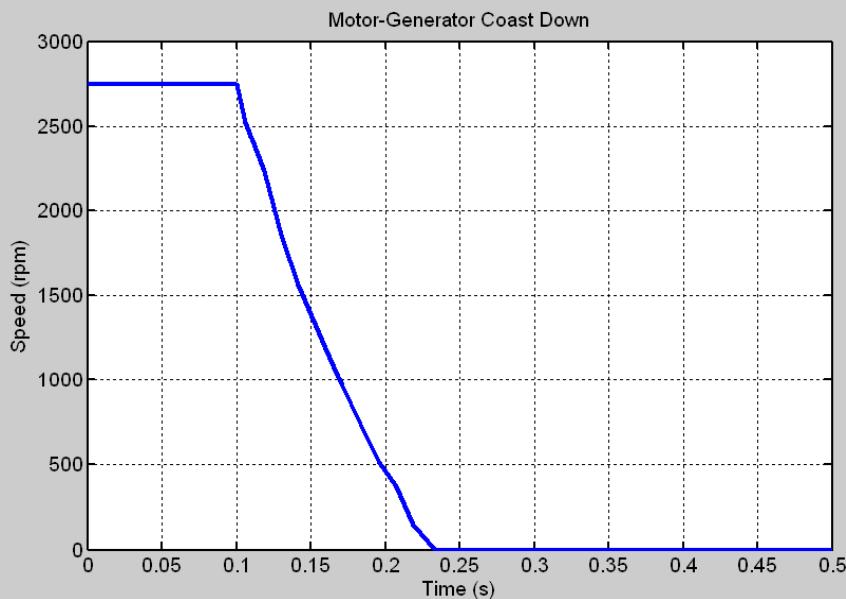
- %Measured Coast Down Data for Motor-Generator System
- time=[ 0 0.100 0.1060 0.1187 0.1298 0.1417,...
- 0.1596 0.1715 0.1834 0.1953 0.2072,...
- 0.2191 0.2340, .5];
- rpm=[2748, 2748, 2525, 2228, 1857, 1560, 1188, 966,...
- 742, 519, 371, 148,0,0];
- plot(time,rpm);
- xlabel('Time (s)');
- ylabel('Speed (rpm)');
- title('Motor-Generator Coast Down');
- grid on;





21

## Motor-Generator Coast Down



22

## Motor-Generator Coast-Down

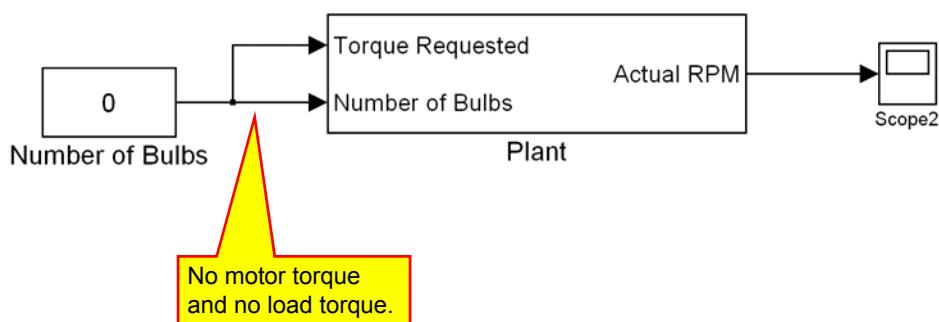
- We must now modify the plant so that:
  - It has no torque input and no load.
  - We set the initial condition to 2748 rpm.
  - Remove whatever torque maintains the speed at 2748 rpm and allow the motor-generator system to coast down to 0 rpm.
- The top-level is shown next.
- Note that the controller has been removed.





23

## Motor-Generator Coast Down



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

24

## Motor-Generator Coast-Down

- Inside the plant, we need to set the initial condition to 2748 rpm.
- We will add a new torque source that balances the torque due to friction.
- While the forces are balanced, the motor-generator system will maintain the initial rpm, whatever it is.
- When we want the system to coast-down, we set the added torque source to zero.

**MotoTron**

The MathWorks

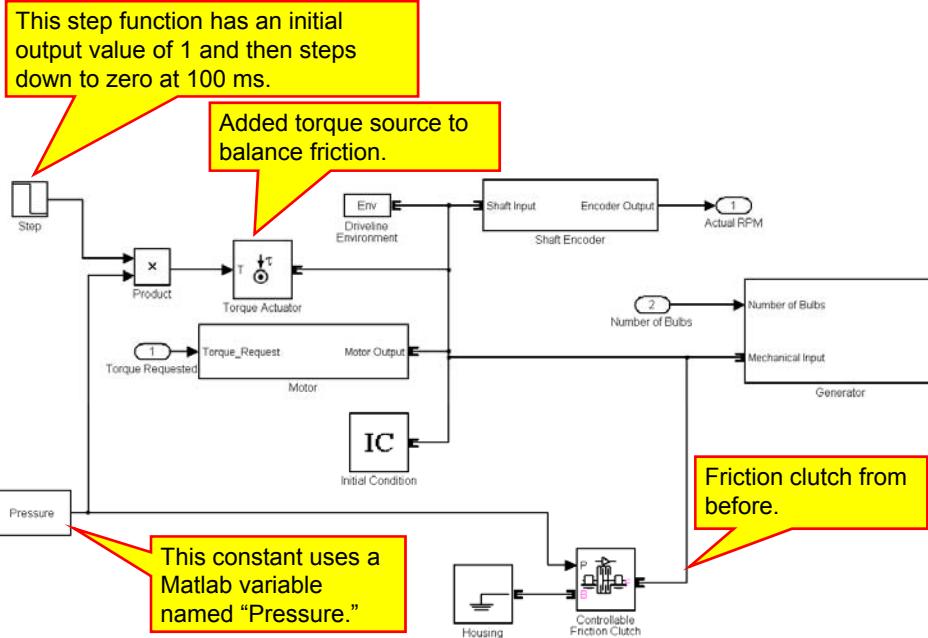
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



25

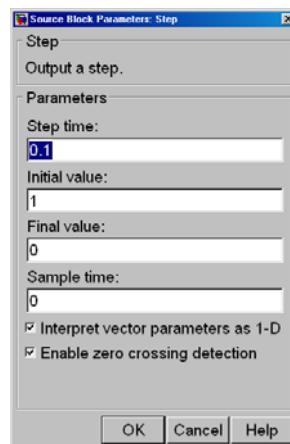
## Motor-Generator Coast Down



26

## Motor-Generator Coast Down

- The settings for the **Step** input source are shown below. This part is located in the **Simulink / Sources** toolbox.

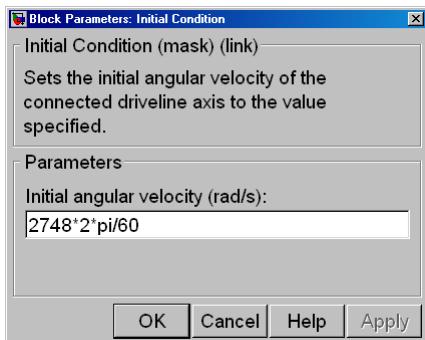




27

## Motor-Generator Coast Down

- The initial condition of the drive line is set as shown:



28

## Motor-Generator Coast Down

- Next, in the Matlab workspace set the value of the constant Pressure to 0.05, the value we found in the last section.

```
Command Window
To get started, select MATLAB Help or Demos from the Help menu.

>> Pressure=0.05

Pressure =
0.0500

>> |
```





29

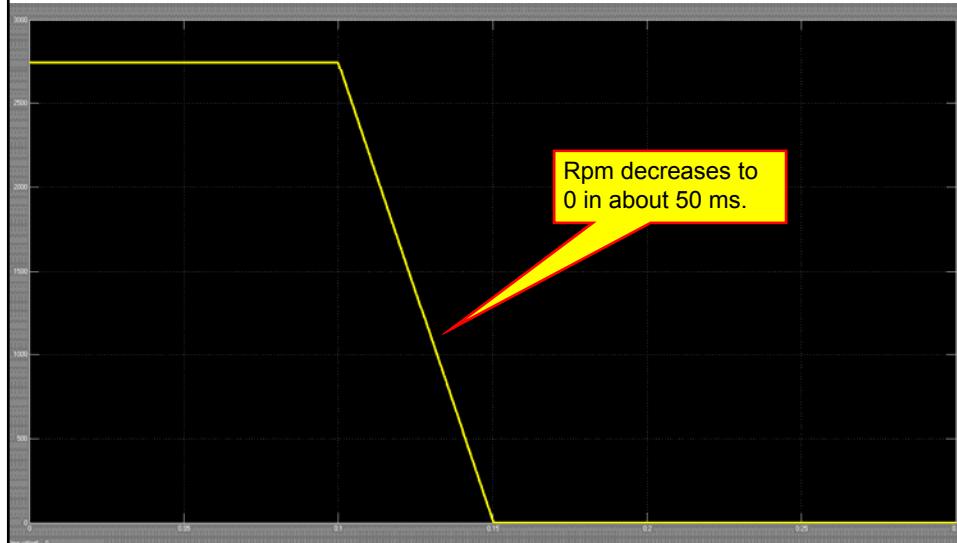
## Motor-Generator Coast Down

- Run a simulation for 300 ms and verify that:
  - The system starts with an initial rpm of 2748.
  - At 100 ms, the system coasts down to zero rpm.
  - The time it takes to coast down is not the same as the measured time.



30

## Motor-Generator Coast Down





31

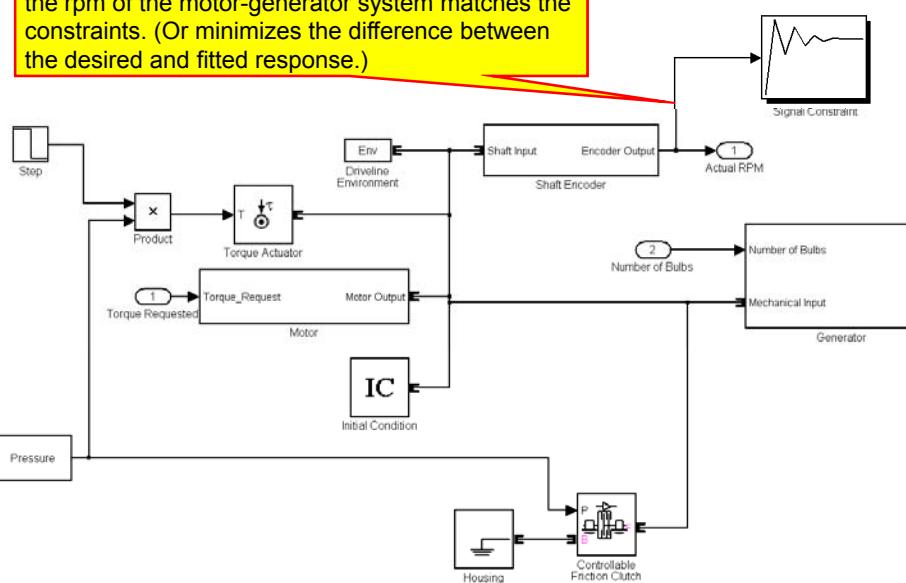
## Motor-Generator Coast Down

- We will now use the MathWorks Simulink Response Optimization toolbox to determine the optimum value of the Pressure so that the response of our model matches the measured response.
- Add a part called **Signal Constraint** from the **Simulink Design Optimization** toolbox.

32

## Motor-Generator Coast Down

We are going to choose the Pressure signal so that the rpm of the motor-generator system matches the constraints. (Or minimizes the difference between the desired and fitted response.)

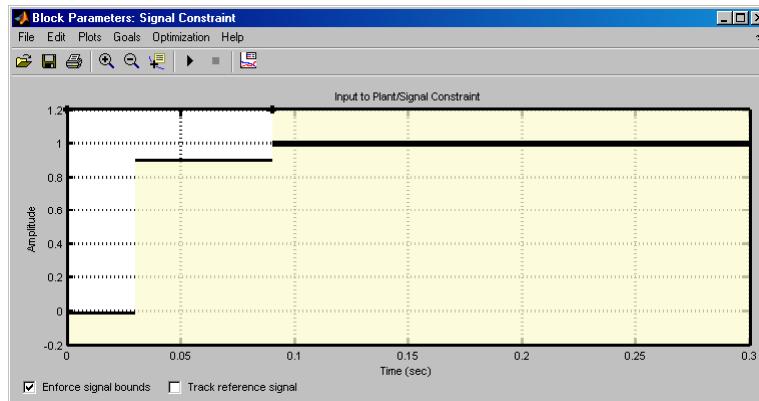




33

## Response Optimization

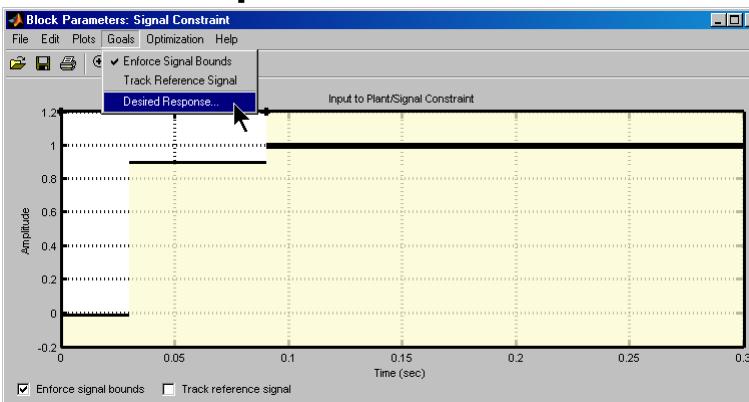
- Open the **Signal Constraint** Block
- The plot shown is for having a response fitted to a set of constraints.



34

## Response Optimization

- We have some measured data that we would like to use for the response.
- To use the data, select **Goals** and then **Desired Response**:

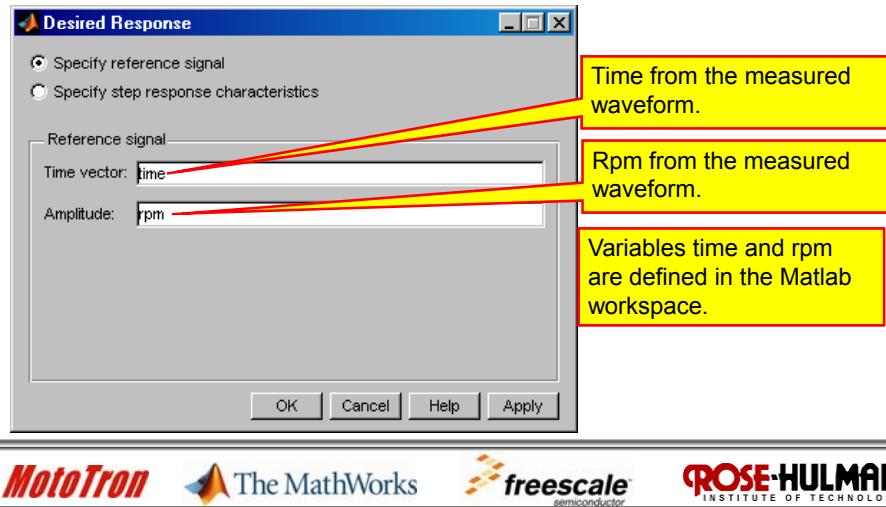




35

## Response Optimization

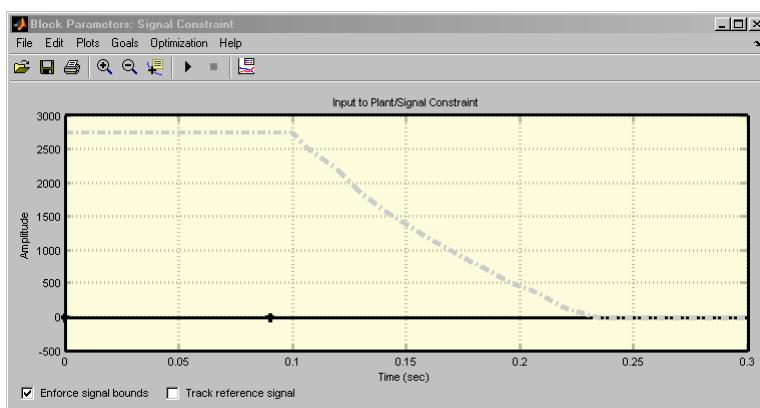
- Set the Reference Signal to the measured time and rpm data:

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

36

## Response Optimization

- When you click the **OK** button, the measured response will be displayed:

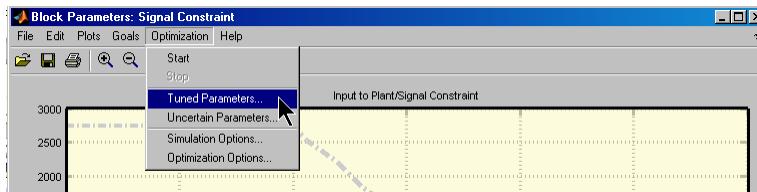
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Response Optimization

37

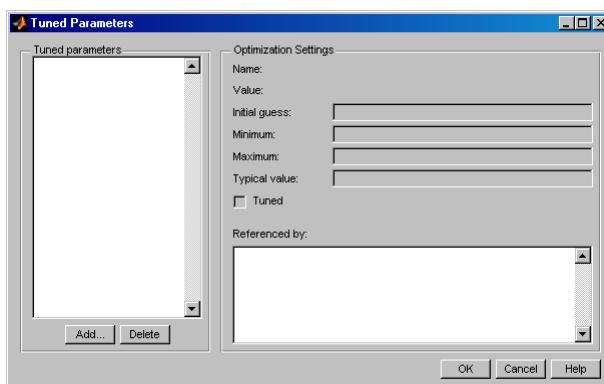
- Next, we have to select the parameters that can be varied to achieve the desired response.
- We have previously defined variable Pressure in the workspace and gave it a value of 0.05.
- Select **Optimization** and then **Tuned Parameters** from the menus:



## Response Optimization

38

- Presently no parameters are selected.

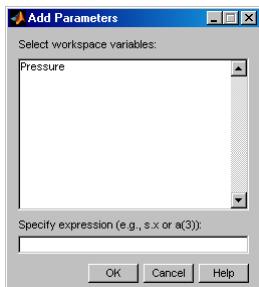


When you click the **Add** button, the list of available parameters defined in the workspace will be listed.



## Response Optimization

39

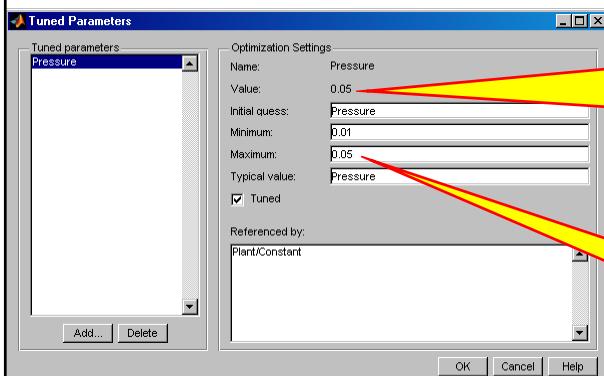


- Select the parameter Pressure and click the **OK** button.
- Specify the minimum and maximum values as shown next.

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

## Response Optimization

40



The initial value of the Pressure constant is set to 0.05, the value we gave it at the Matlab command prompt.

We constrain the Pressure input to be between 0.01 and 0.05.

Click the **OK** button when done.

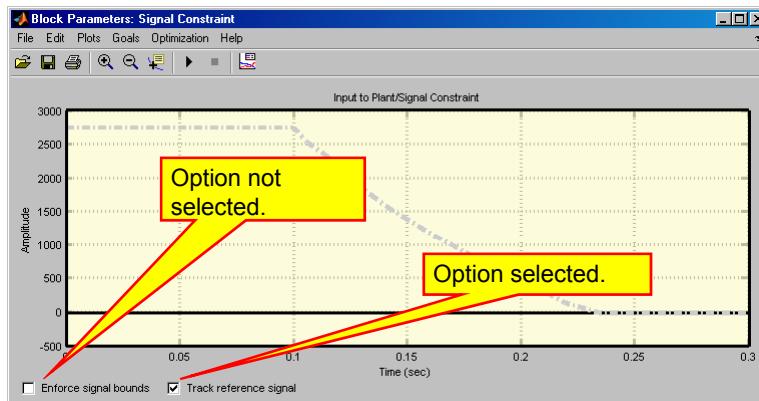
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



41

## Response Optimization

- Select the **Track reference signal** option so that we optimize to the provided data.



**MotoTron**

**The MathWorks**

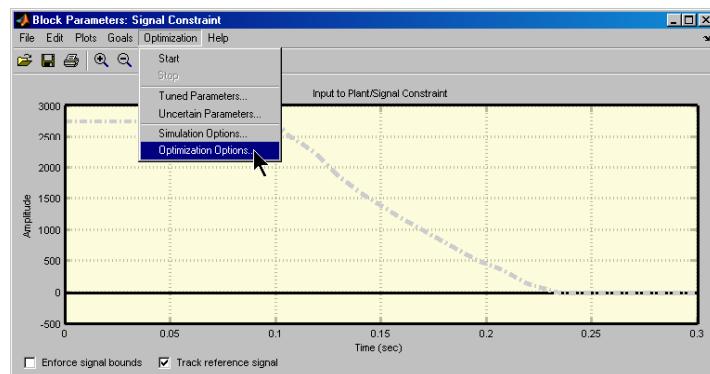
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

42

## Response Optimization

- To make the system track the desired response more closely, we need to reduce the tolerance slightly. Select **Optimization** and then **Optimization Options** from the menus:

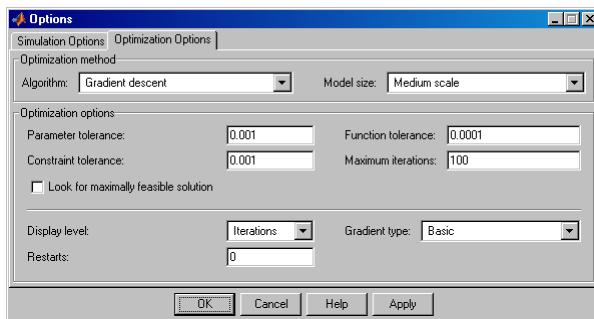




43

## Response Optimization

- Set the function tolerance to 0.0001.
- Click the **OK** button when done.

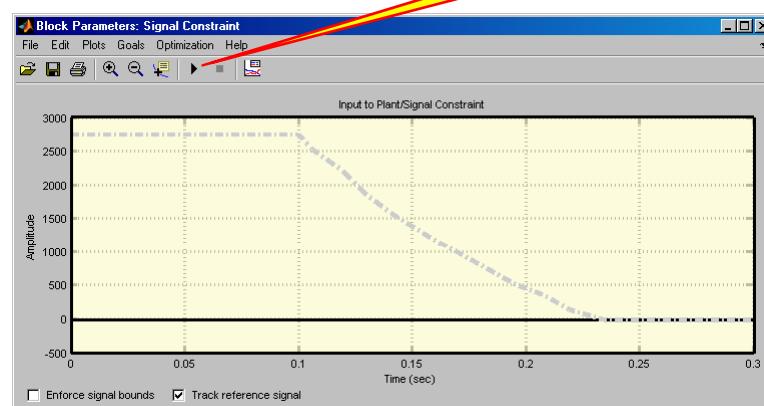


## Response Optimization

44

- Click the **Play** button and watch the optimization proceed:

**Click here.**

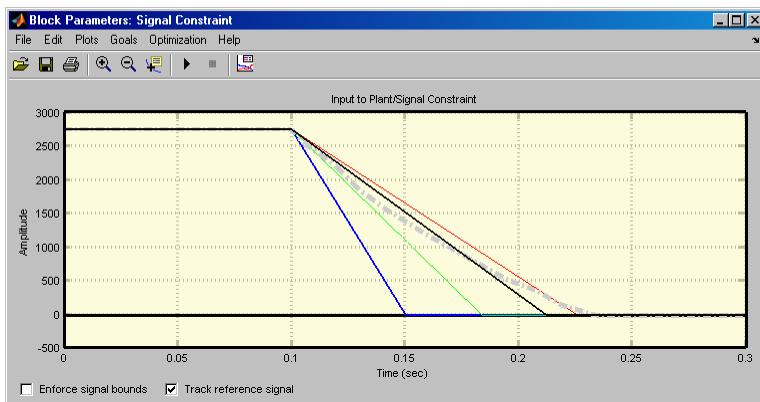




# Response Optimization

45

- Results



**MotoTron**

**The MathWorks**

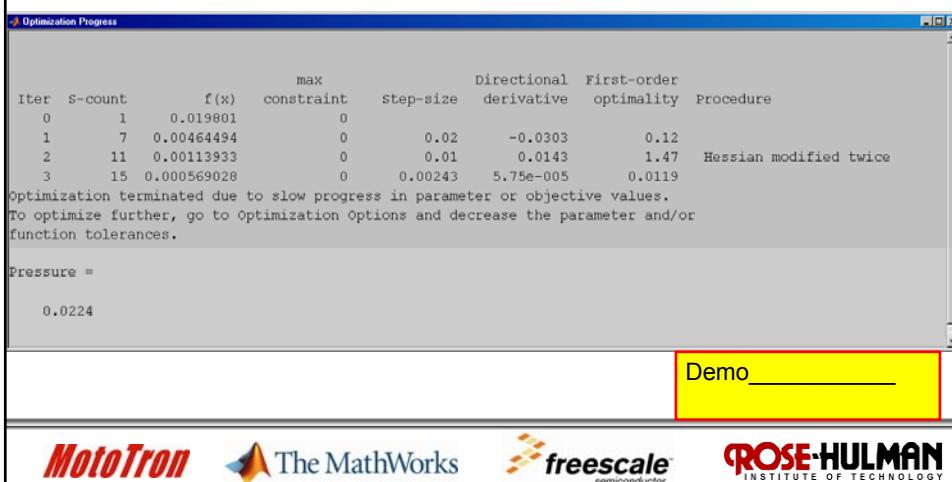
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Lecture 5 Demo 1 Response Optimization

- Optimum value of Pressure is 0.0224.

46



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



47

## Friction Model Notes

- Our model has a linear change from high rpm to zero. The actual response looks like an exponential decay – somewhat.
- Our model of a constant frictional torque is probably too simple.
- Friction is probably a function of rotational speed.
- → Stuff to add in the future.



48

## Lecture 5 Problem 1

- In Lecture 3A problem 2, we measured the coast-down of the motor/generator system with the flywheel.
- Add the flywheel to your model
- Use the measured coast-down data from Lecture 3A as the reference signal.
- Use the Response Optimization Toolbox to determine the optimum value of clutch pressure for the system with the flywheel.

Demo \_\_\_\_\_





49

## Lecture 5 Problem 2

- In both models for this lecture (the model with the flywheel and the one without the flywheel) we noticed that the measured rpm had a slight curve while the simulated rpm was a straight line.
- The simulated response is a straight line because we are applying a constant torque which means that the angular velocity decrease at a constant rate.
- To model the curvature in the measured rpm, we will try a new model.
- Modify your model so that the friction has a constant component plus a component that is proportional to motor speed (in rad/sec).
- Make both components tunable.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

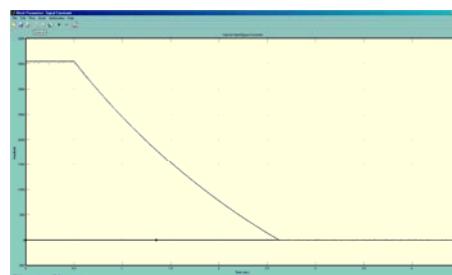
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

50

## Lecture 5 Problem 2

- Use the optimization tool to determine the coefficients for an optimal fit. You will need to determine two coefficients.
- An optimal plot is shown below. The fit is so close that you cannot see much difference between the measured data and the model results in the screen capture.
- You may want to change the optimization method to “Simplex search.”
- You may want to increase your max step size so that the simulations take less time.

Demo





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

## Any Questions?



The MathWorks





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



## Introduction to Model-Based Systems Design

### Lecture 6 : Signal Filtering



## Signal Modeling

2

- We now need to clean up our model in terms of the signals being sent to the controller.
- Starting with the shaft encoder, it does not send an rpm signal but rather a voltage proportional to rpm
  - Name plate specification: 2.5 V per 1000 rpm
  - At some point we'll need to verify this.
- Let's add that to the encoder model

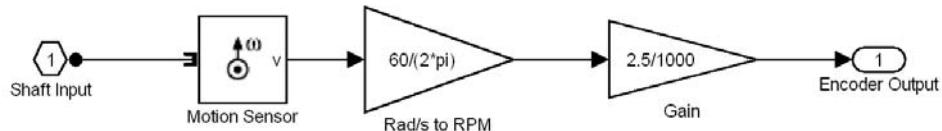




3

## Signal Modeling

- Drag a gain into the encoder and set the value to 2.5/1000.

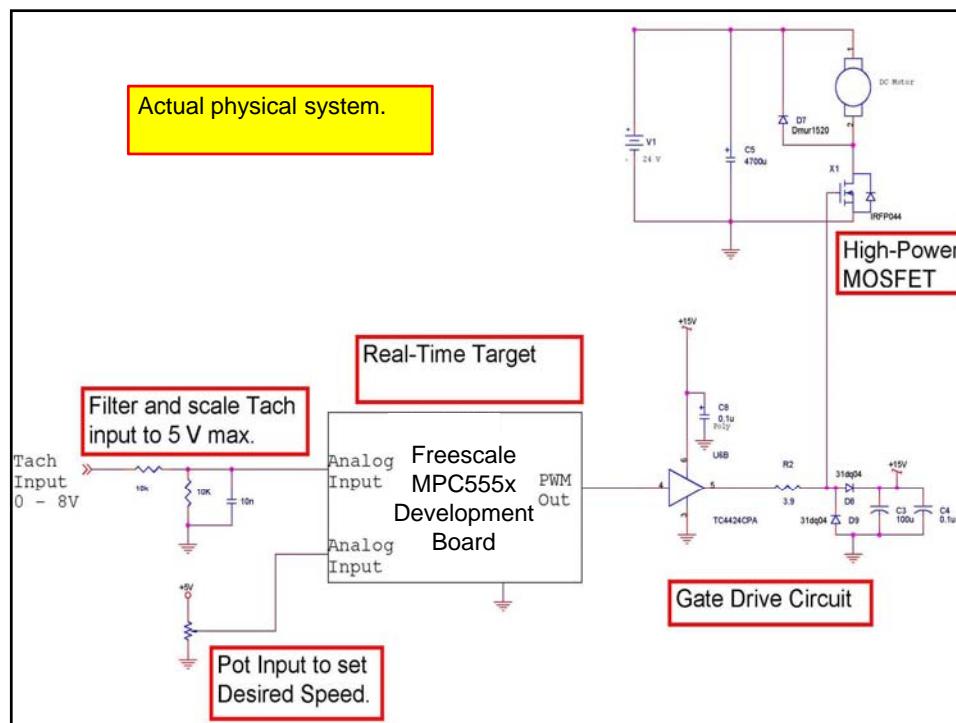


**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

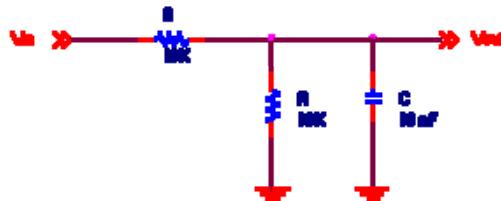




5

## Improved Signals – MPC555x

- This 0-8 V signal is next passed through a combination resistive divider and low pass filter to
  - Eliminate noise on the signal
  - Change range to 0-5 V
- The circuit diagram is



6

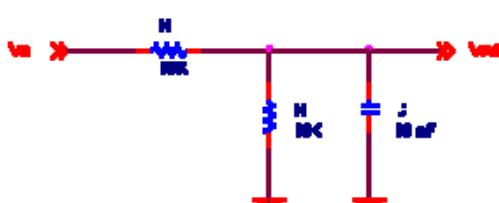
## Improved Signals - MotoTron ECU

- From circuit analysis

$$\frac{V_o}{V_{in}} = \frac{R\|Z_C}{R + R\|Z_C}$$

$$R\|Z_C = \frac{R \times \frac{1}{sC}}{R + \frac{1}{sC}}$$

$$\frac{V_o}{V_{in}} = \frac{\left(\frac{R}{1+sCR}\right)}{R + \left(\frac{R}{1+sCR}\right)} = \frac{R}{sCR^2 + 2R}$$





7

## Improved Signals - Controller

- Having figured out the appropriate transfer function to represent our filter and knowing
  - $R = 10k$
  - $C = 10 \text{ nF}$
- Drag a **Transfer Fnc** block into the controller from the **Simulink / Continuous** library.
- We'll also need a gain and a saturation
- We'll need to disconnect the Actual rpm input from the P controller as well

**MotoTron**

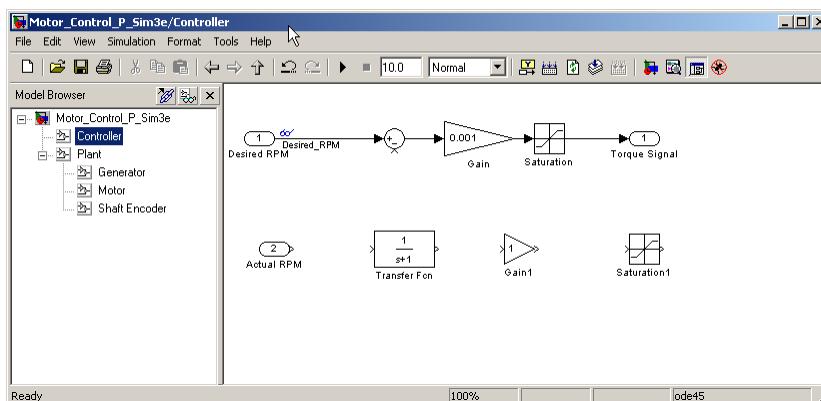
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

8

## Improved Signals - Controller



**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/>

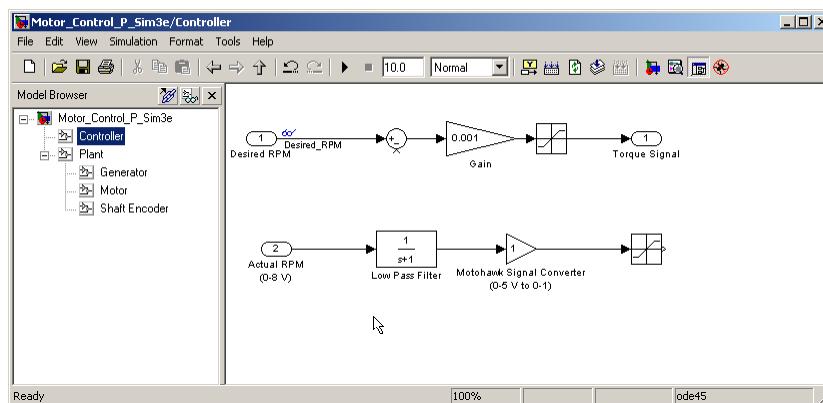
9

## Improved Signals - Controller

- Changes:
  - Actual rpm → Actual rpm (0-8V)
  - Transfer Fnc → Low Pass Filter
  - Gain 1 → Signal level converter (0-5 V to 0-1)
  - Saturation -> hide name
- Link them together

10

## Improved Signals - Controller

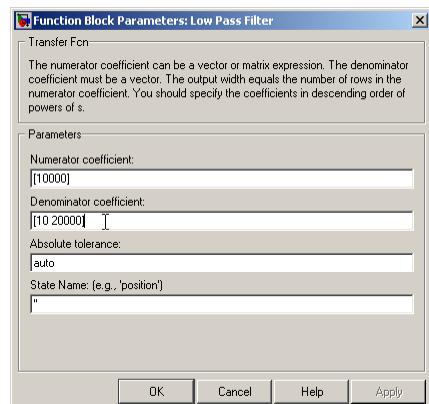




11

## Improved Signals - Controller

- Double click on the Low Pass Filter transfer function and put in the values for R and C
  - The numerator coefficient is the value of R
  - The denominator coefficients are the multipliers on s and 1 respectively  $CR^2$  and  $2R$



12

## Improved Signals - Controller

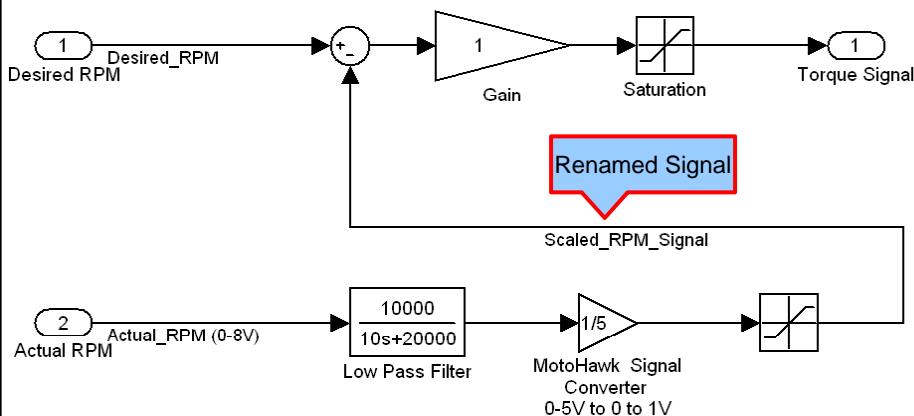
- The last step in the controller is to take the filtered signal which ranges from 0-5 V and convert it to a signal of magnitude 0-1
  - Note that our filtered rpm voltage signal has a range of 0 to 4 V.
  - We will not use the full range of the A/D converter (which has an input range of 0 to 5 V)
- Set the gain to 1/5 for the conversion
- Set the saturation to limits to 0 and 1
- Connect the output back to the summing junction.





## Improved Signals - Controller

13

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

## Improved Signals - Controller

14

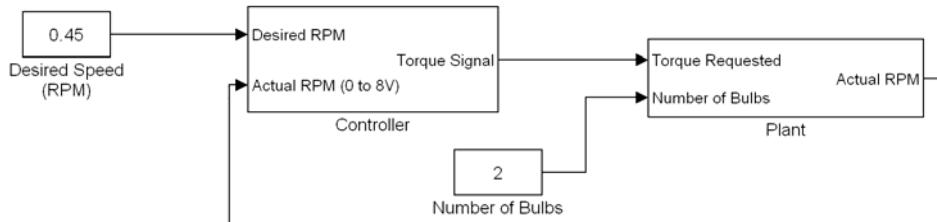
- In the controller an rpm of 0 to 4000 rpm is represented by a signal that ranges from 0 to 1.
- We need modify our desired rpm of 1800 rpm to reflect this change.
- We will change the desired rpm signal to 0.45 (equal to 1800/4000).

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



## Improved Signals - Controller

15

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

## Improved Signals - Controller

16

- Before we run our model, let's think about the controller
- We initially had our gain at 0.01 because that would yield a full torque signal if the rpm difference was 100 rpm
- Now, 100 rpm is roughly a signal of 0.025
- We should expect to increase our proportional gain

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

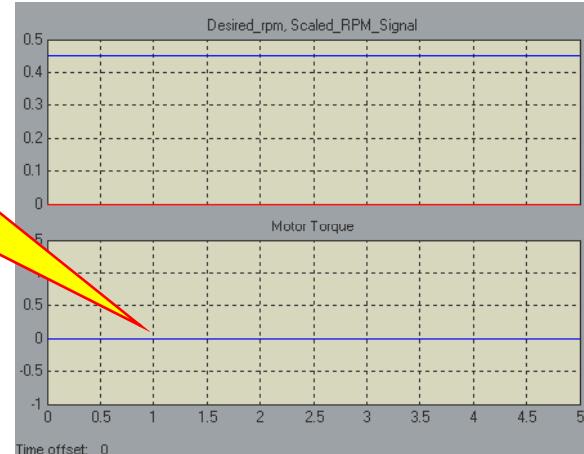


## Improved Signal - Controller

17

- Let's set our initial condition to zero, the load to 2 bulbs, the desired rpm to 0.45, and leave the gain at 0.01.

Motor torque effectively zero. Gain is way to low for the small difference between the desired and actual speed.



## Improved Signals - Controller

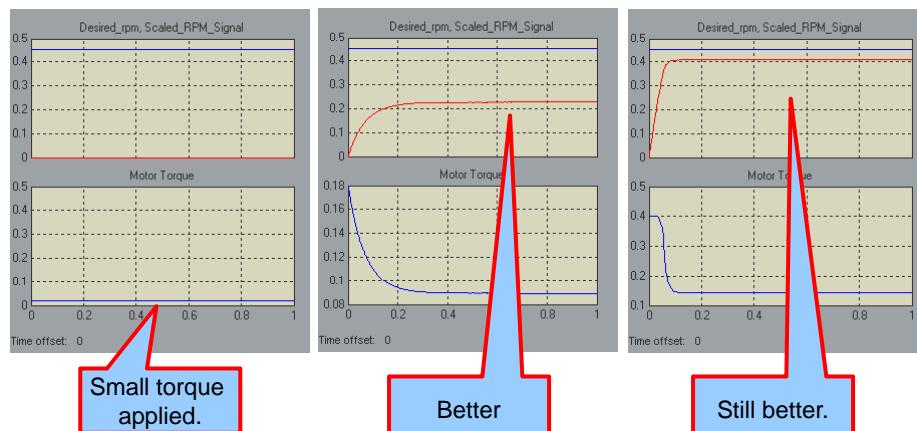
18

- Now set the gains to 0.1, 1, and 10

Gain = 0.1

Gain = 1

Gain = 10

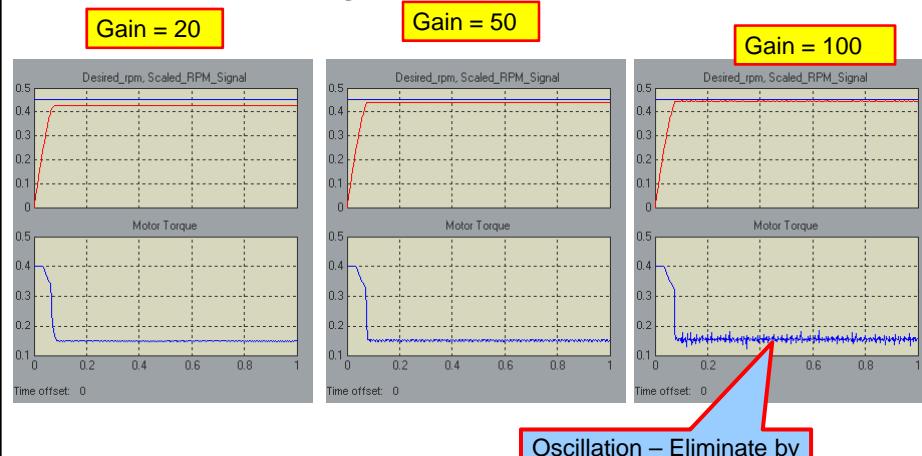




19

## Improved Signals - Controller

- Now set the gains to 20, 50, and 100



Oscillation – Eliminate by  
decreasing max step size.

**MotoTron**

**The MathWorks**

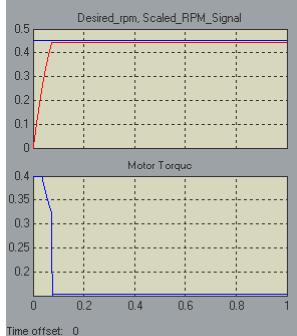
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

20

## Lecture 6 Demo 1

- Demonstrate the system response for various feedback gains.
- Demonstrate the effect of reducing the simulation maximum step size.



Demo \_\_\_\_\_

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



21

## Model Improvements

- Improved motor from constant torque source to torque dependent on rpm.
- Improved generator:
  - Linearly dependent on rpm with constant max load.
  - Linearly dependent on rpm with variable load.
  - Linearly based on rpm with physical relationship to generator voltage and load resistance.
- Added friction
  - Constant
  - Linear dependent on rpm
- Scaled signals to match levels used by Freescale target.



22

## Model-Based Design

- We made the improvements one at a time.
- We verified that the system was behaving as we expected after each improvement.
- We ran extensive tests to identify changes in system response due to the improvement.





23

## Init File

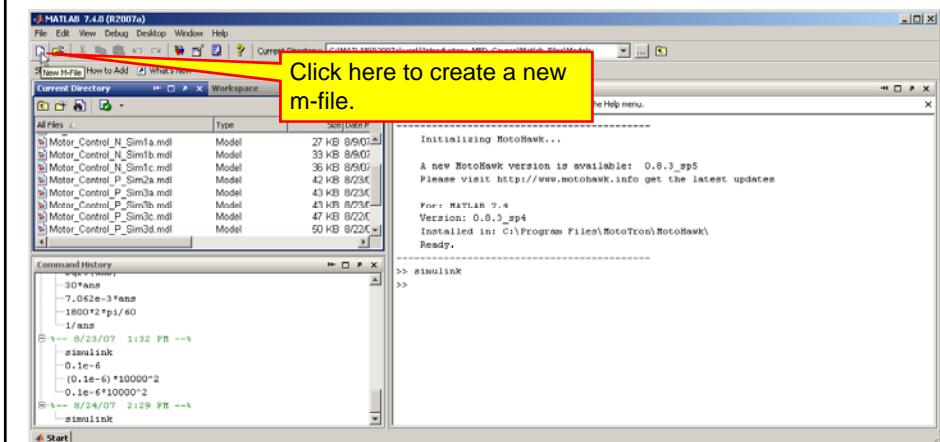
- We've got a lot of magic numbers in our model:
  - Motor and Generator Inertia
  - Motor Torque Constant
  - Etc
- In a typical model, model constants are reused in several places.
- If we choose to change any of these parameters, we might miss some of them in the model.
- Let's put these values into an initialization file.
- This file will be executed each time we run our model, before the model runs.



24

## Init File

- Open your Matlab window and open a new m-file:
- Select **File**, **New**, and then **M-File** from the menus or click on the **New M-File** icon as shown below:





25

## Init File

- Save the file in the same directory as your Simulink models.
- Save the file as “Init\_File.m”
- Enter the text shown on the following page exactly as shown.
- Save the file when done.
- This will allow us to assign variable names to parameters inside of our Simulink model and read those values from the workspace



26

## Init File

```

Editor - C:\Documents and Settings\herniter\My Documents\Website\Rose_Classes\Workshop\Init.m
File Edit Text Go Cell Tools Debug Desktop Window Help
[File Explorer] [New] [-] [1.0] [+] [÷] [1.1] [×] [%] [%] [①]
1 %Real_Time_Workshop_Constants
2 Sample_Time=.001; %Seconds
3 PWM_Period = 1/20000; %Seconds
4
5
6 %Low Pass Filter Constants
7 R=10000; %Ohms
8 C=0.1e-6; %Farads
9
10
11 %Motor and Generator Contants
12 Motor_Inertia = 4.378e-6; %kg*m^2
13 Torque_Constant = 9.1; %Oz-In per amp
14 Motor_Rated_Current = 4; % Amps
15 Generator_Inertia = 4.378e-6; %kg*m^2
16 Generator_Voltage_Gain = 24/3500; %Volts per rpm
17
18 %Load Constants
19 Bulb_R = 12; %Standard 12 V automotive bulb (Ohms)
20
21 %Encoder Constant
22 Encoder_Gain = 2.5/1000; %Volts per rpm.
23
24 %Flywheel Inertia
25 Flywheel_Inertia = 1.041e-4;

```



## Init File

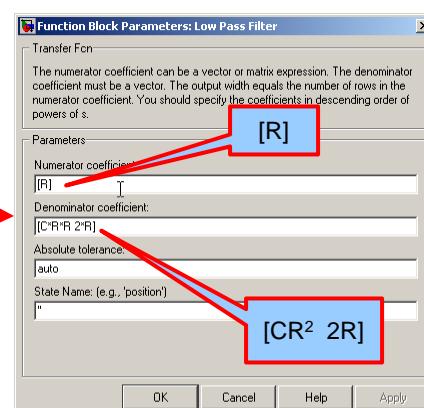
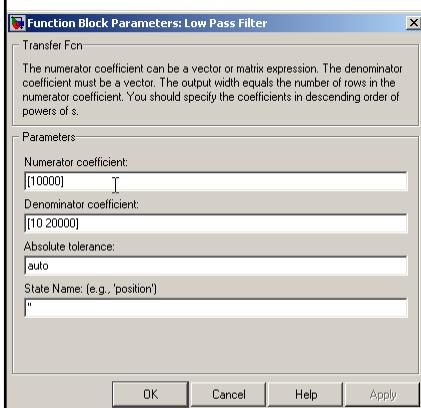
27

- Now we need to update our model with the variable names in the init file
- We'll start with the low pass filter
  - Save your model as Lecture6\_Model3
  - Open up the controller
  - Double click on your low pass filter
  - Change the numbers to variables as shown on the next page



## Init File

28

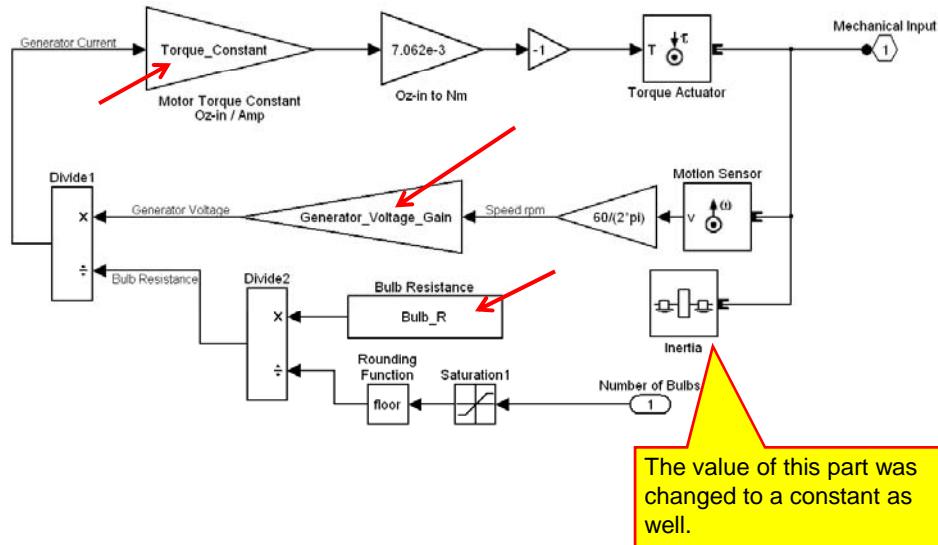




29

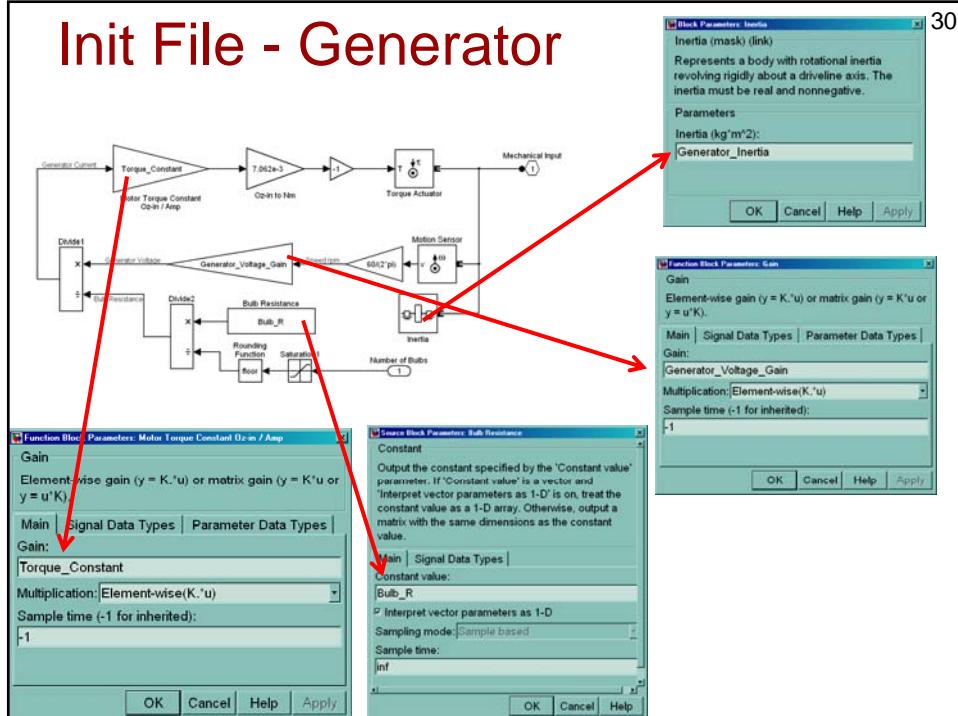
## Init File

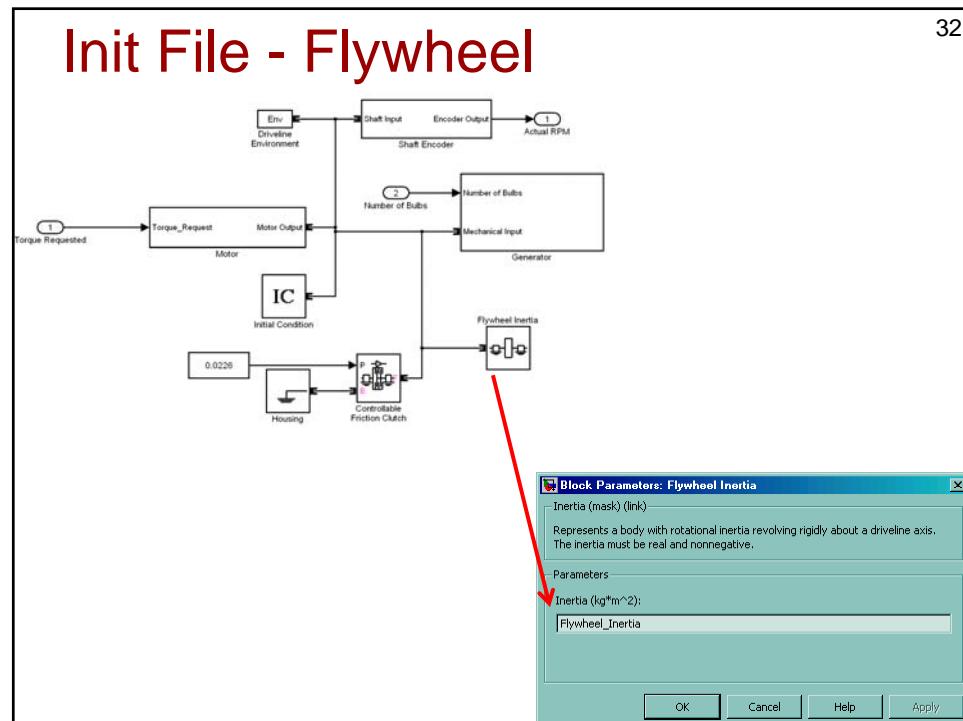
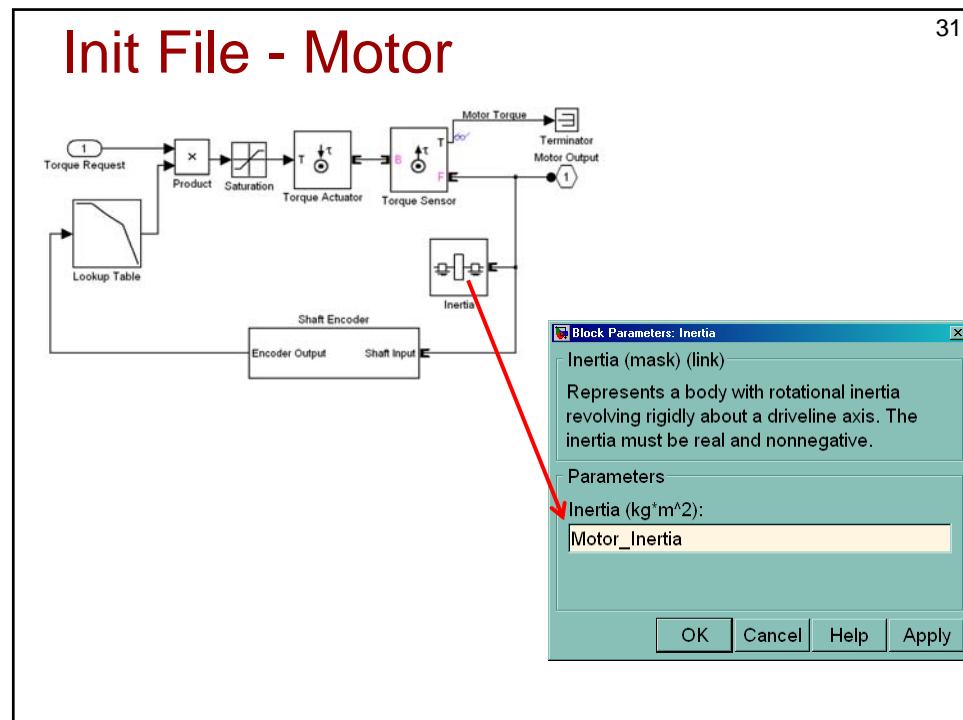
- Update the generator as shown:



30

## Init File - Generator



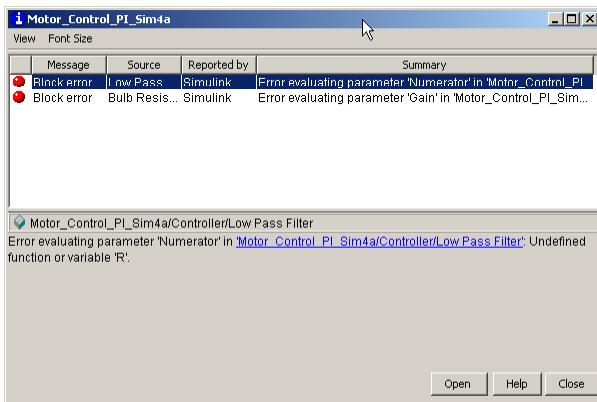




## Init File

33

- Save your file and run!



- Hmm, undefined variable...



## Init File

34

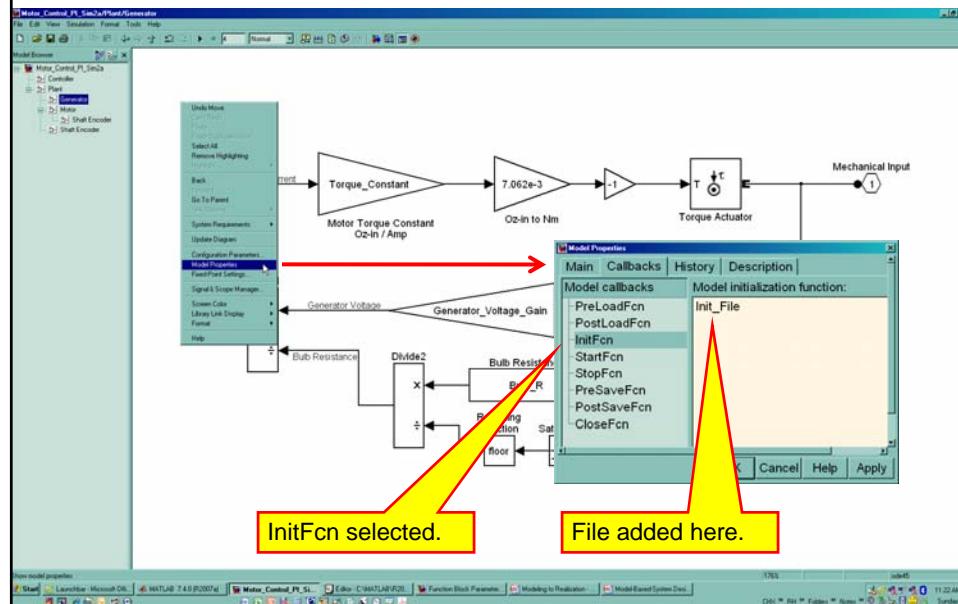
- We can either run the script file manually to place the variables into the workspace, or...
- Have Simulink run the init file automatically each time we run a simulation (which also places the variables into the workspace).
- Right click any place in your model where it is blank
- Select **Model Properties**
- Select the **Callback History** tab
- Add Init\_File to the **InitFcn Model Callback** section
- Click **OK**





## Init File

35



## Init File- Lecture 6 Demo 2

36

- Run your model now
- It works because the Init\_File runs before the model runs, and defines all of the constants needed by the mode.
- From now on, we will define all of our data and model constants in the init file.
- The model will reference the variables named in the init file rather than use magic numbers.

Demo\_\_\_\_\_





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



## Introduction to Model-Based Systems Design

### Lecture 7: Time Delays and Proportional and Integral Control



The MathWorks



freescale<sup>®</sup>



## Presentation Outline

2

- Sample Time Delay
  - Assess impact of signal delay on system performance.
- PI Control
  - Add integral control to eliminate error.



The MathWorks



freescale<sup>®</sup>





3

## Model Updates

- Before we begin, we will update our model with the coast-down friction components that we added in Lecture 5 Problem 2.
- Add the following parameters to the init file:

```

24 %Flywheel Inertia
25 Flywheel_Inertia = 1.041e-4;
26
27 %Coast Down Friction Coefficients
28 % Calculated in Lecture 5 Problem 2
29 P1 = 0.0130;
30 P2 = 0.4152;

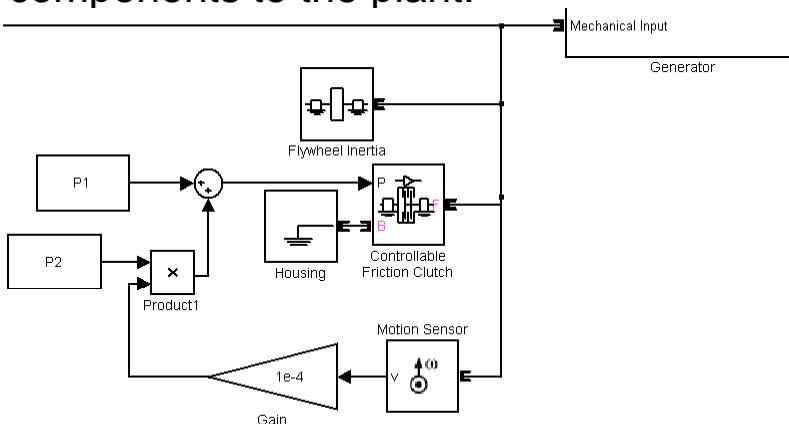
```



## Model Updates

4

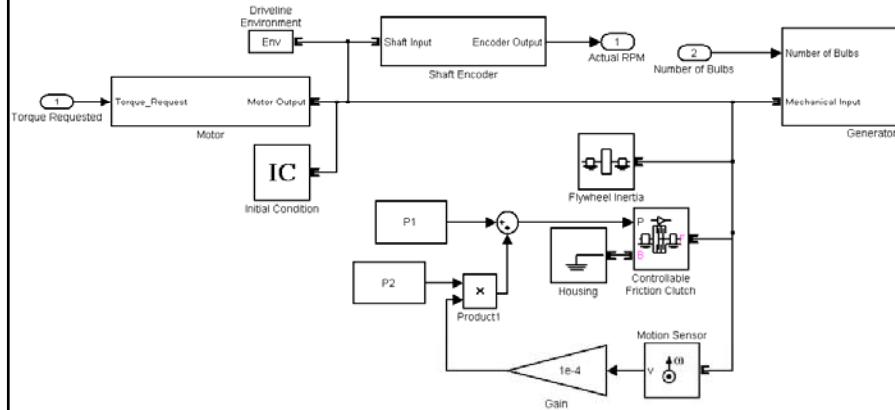
- Add the proportional and constant friction components to the plant:





## Model Updates – Complete Plant

5



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Step Response

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



7

## Step Input

- So far we have been running our model for short simulation times
  - We have changed nothing while the simulation is running
- Let's slow our system down and watch the effect of changing the load or desired rpm
- Model Name Change
  - Save your model
  - Save your model as Lecture7\_Model1



8

## Step Input

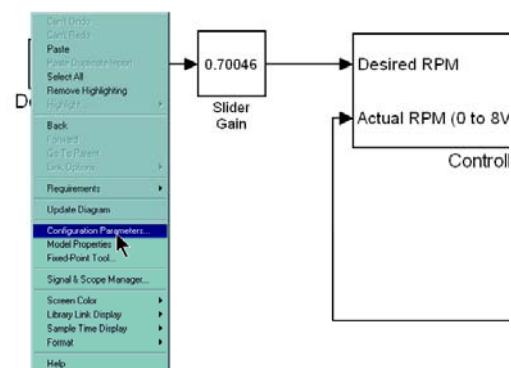
- First, let's slow down our simulation.
- On your model, again right click any place where there is a blank area.
- Select **Configuration Parameters**
- Select the **Solver** tab.
- Change the **Stop Time** to inf (it will run forever)
- Change the **Max Step Size** and **Initial Step Size** to 1e-4 seconds (this will slow it down by decreasing the maximum step size and forcing more calculations).





## Step Input

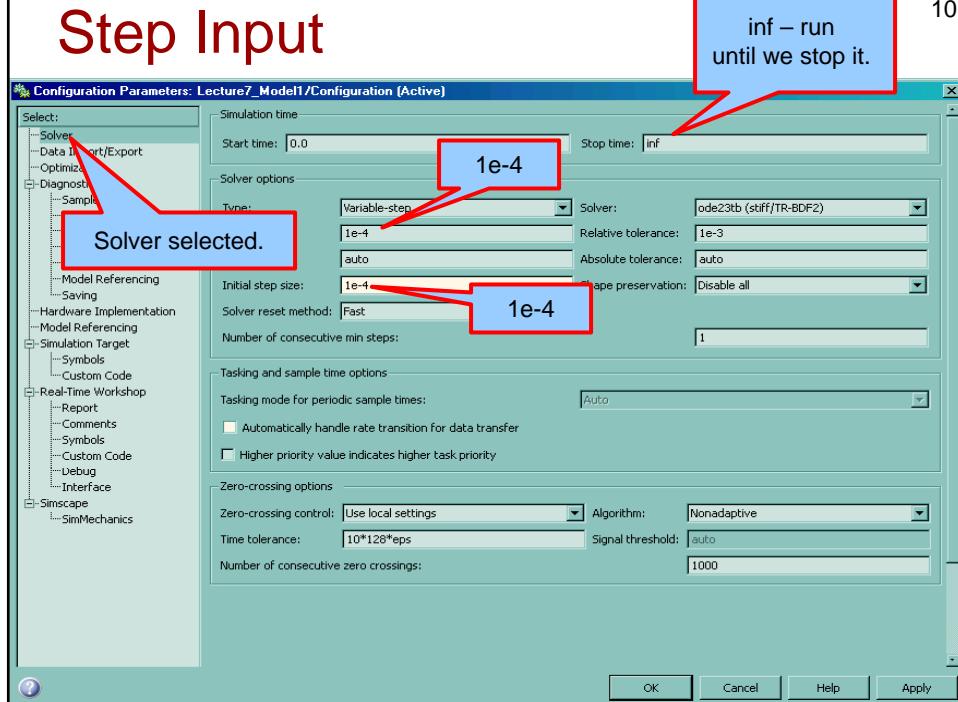
9

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

## Step Input

inf – run until we stop it.

10





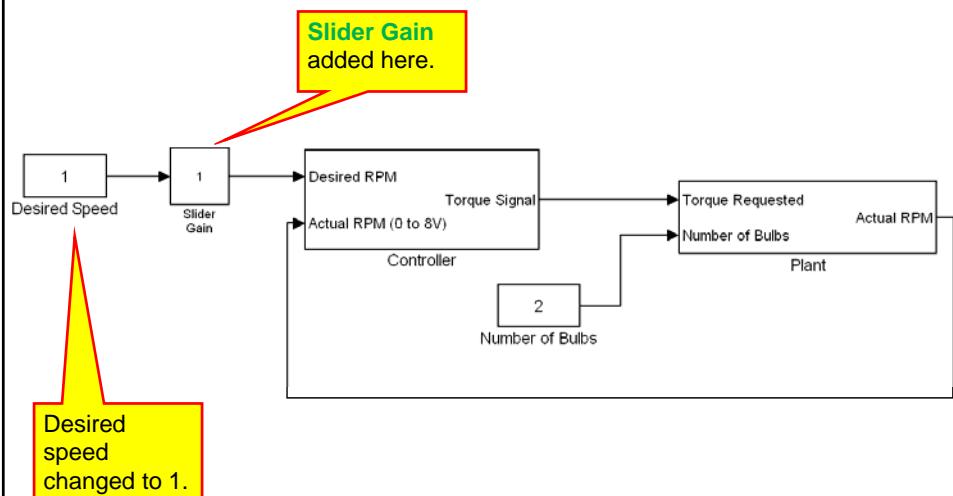
## Step Input

11

- Now let's add the ability to change our desired speed.
- Drag a **Slider Gain** from the **Math Operations** library into the top of your diagram and connect as shown.

## Step Input

12

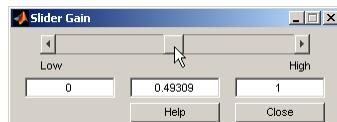




13

## Step Input

- Double click on the slider gain and set the lower and upper limits to 0 and 1 respectively and slide the slider



- Awesome – we can vary our desired rpm from 0 to 3000 rpm *while the simulation is running* by moving this slider from 0 to 1.

**MotoTron**

**The MathWorks**

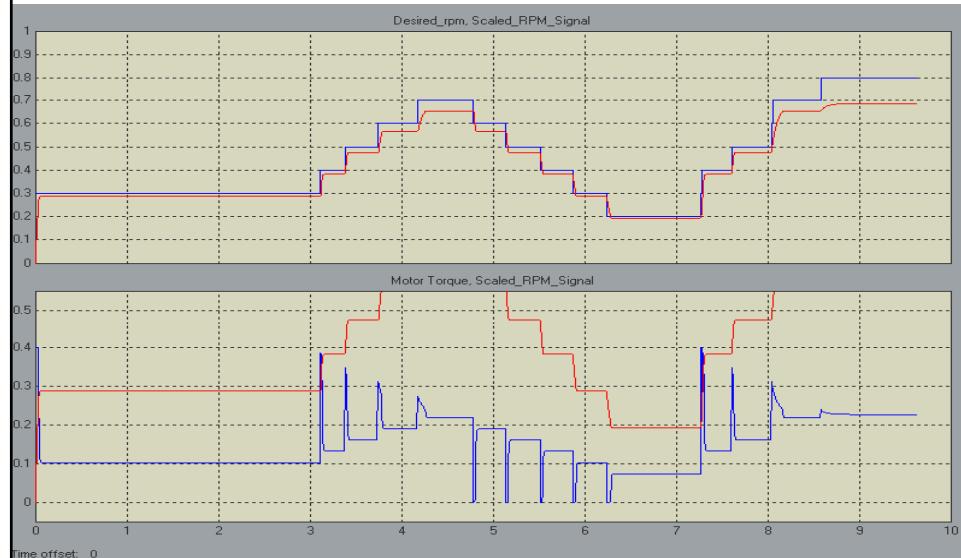
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

14

## Step Input

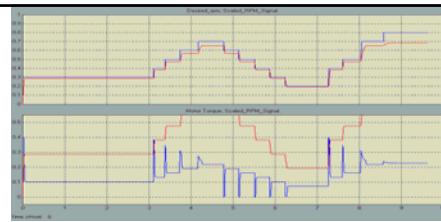
- Run your simulation.





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

## Step Input



- We notice two things:
  - The simulation runs very slowly.
  - The motor generator system responds fast making it difficult to see the step response.
- Even with the added flywheel, the system responds very quickly and makes it difficult to see a speed change request.



## Scope Settings

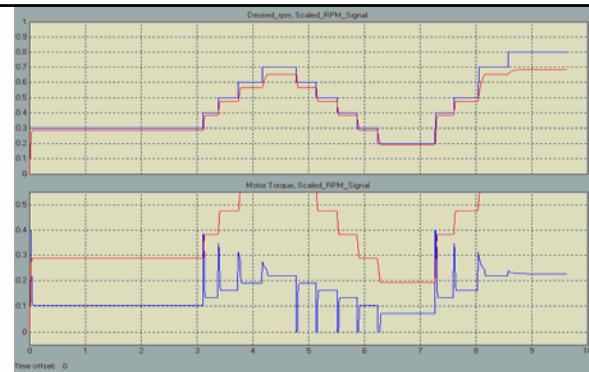
16

- To see the system response, we need to change the scope's default x-axis time range.
- The time axis on our scope is 0 to 10 seconds by default.
- We need to change the time axis on our plots.





## Step Input



- Stop the Simulation.
- To adjust the x-axis time range, right click on the scope plot and select **Scope parameters**.
- Set the **Time range** as shown:

**MotoTron**

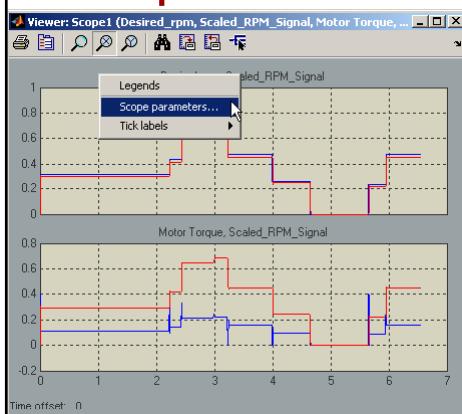
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

18

## Scope Time Range



Time range set to 0.5. This will force the x-axis to display a range of 0 to 0.5 seconds.

Rerun the simulation and move the slider.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

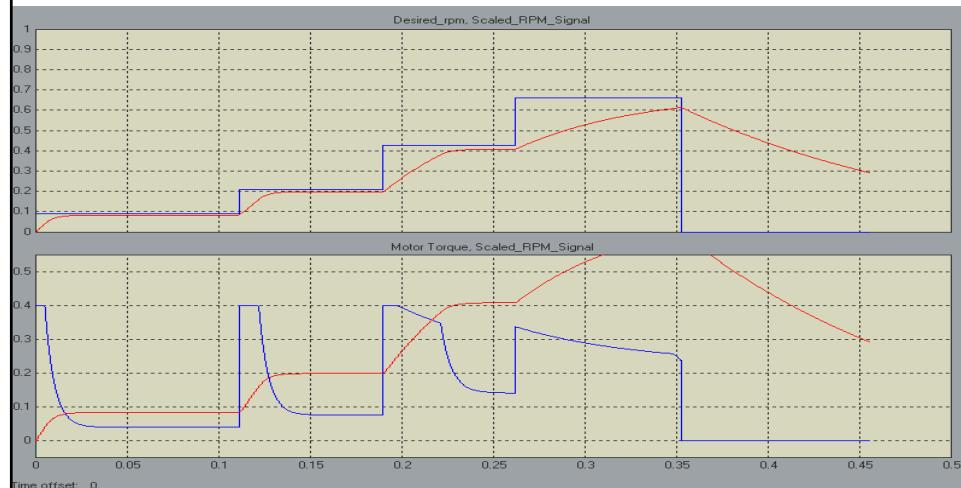
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Step Input

19

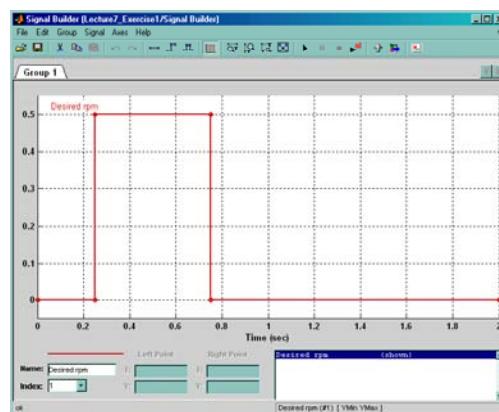
- Pretty cool – you can watch the actual rpm try to match the desired rpm (2 bulb load, Gain = 20.) You may need to slow the simulation even more by reducing the maximum step size to 1e-5.



## Lecture 7 Exercise 1

20

Use the signal builder to create the following pulsed waveform:



(Continued on next slide.)





21

## Lecture 7 Exercise 1

Use this waveform as the rpm input for the controller.  
 For this waveform, or a slight modification, observe  
 the system step response for the following  
 conditions:

Number of Bulbs	Proportional Gain	Rise Time	Fall Time
2	10		
2	20		
2	50		
2	100		

Demo \_\_\_\_\_



## Step Input

22

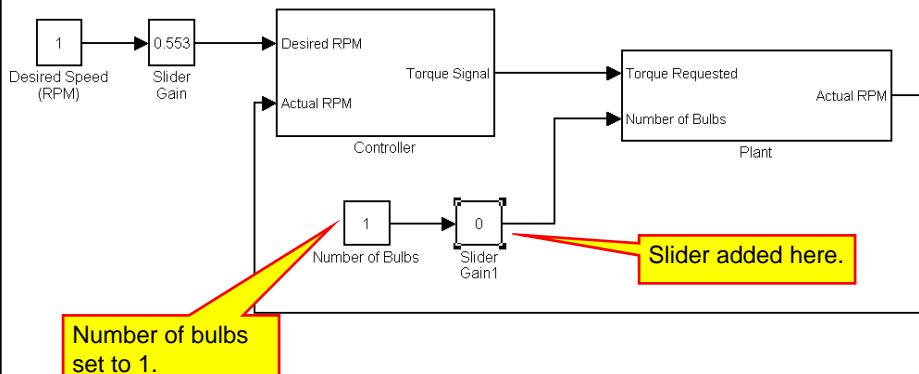
- Our physical system has 6 light bulbs.
- We would like to see how changing the load affects the system response.
- At the top level of your model
  - Change the constant for the number of bulbs to 1
  - Add a slider as shown next.
  - Set the limits on the slider from 0 to 6





## Step Input

23



Number of bulbs  
set to 1.

Slider added here.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Step Input

24

- We will now test how our system responds for changes in the load while holding the speed constant.
- Set the speed slider to 0.6 and do not change it.
- Run the simulation and observe how the system response changes as we change the load slider.
- Set the proportional feedback gain to 100.

**MotoTron**

**The MathWorks**

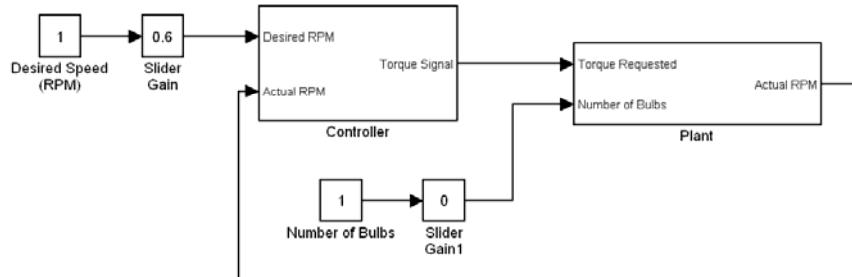
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Step Input

25



**MotoTron**

**The MathWorks**

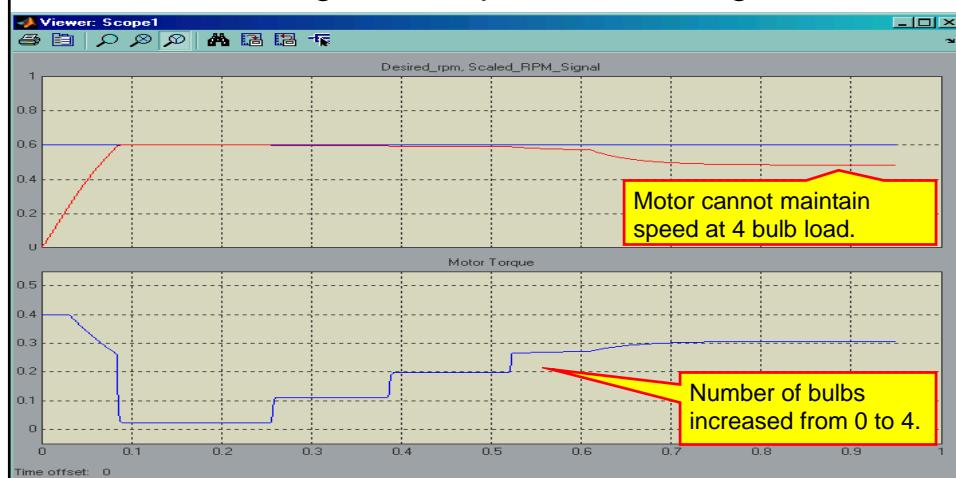
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Step Input

26

- Increasing the load increase the steady state error.
- Decreasing the load decreases the steady state error.
- Notice the changes in torque as we change the load.





## Step Input

27



## Step Input

28

- Decreasing the load.



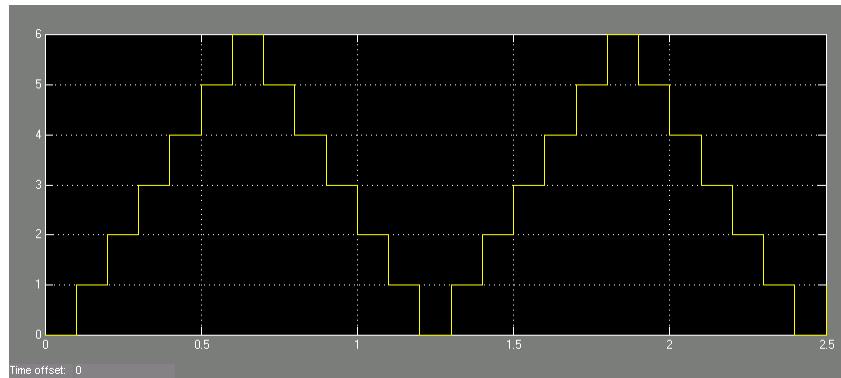


## Lecture 7 Exercise 2

Demo \_\_\_\_\_

29

Create the following waveforms using one of the Simulink sources. Use this input as the number of bulbs and plot the system response.



## Controller Cycle Time

30

- To this point, our controller measures signals and makes a change every computational step.
- Since this is a variable step size solver, this time changes as the simulation runs and can be quite small.
- This is not true in the hardware implementation.
  - In the real-time hardware realization, the controller performs all computations within a fixed time.
  - We will call this time the controller cycle time.
  - The inputs are read once in this fixed time.
  - All calculations are performed for the measured inputs once in this fixed time.
  - The outputs are changed once in this fixed time.
  - If the controller finishes early, we wait until the fixed time is up before executing the control loop again.





31

## Controller Cycle Time

- The effect of the controller cycle time is:
  - We can only sample the inputs once in the time period. If the input changes, we won't know about it until the next fixed time step.
  - We can only change the output control signal once in the fixed time period. This limits how fast we can make a change. (Or, how fast the controller can respond.)
- For control people, this time delay is a phase delay in the feedback loop.
- This delay can cause the system to become unstable.



32

## Controller Cycle Time

- As a first step in modeling this delay , we will use a sample and hold.
- The value will be held for a fixed and specified amount of time.
- We will model the controller cycle time on both the inputs and outputs of the controller.
- In the hardware realization, the fixed time can be changed at compile time. It can be reduced within limits imposed by your ECU.
- In a future lecture, we will run a Software-in-The-Loop (SiL) simulation to model this delay.





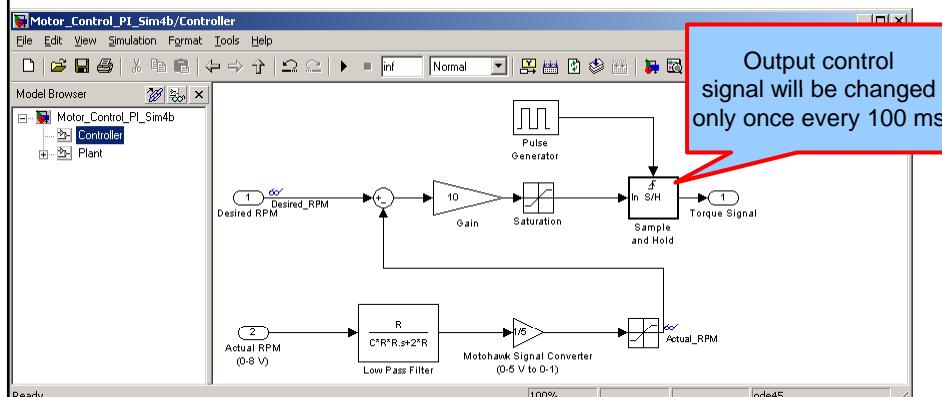
33

## Controller Cycle Time

- Save your model as Lecture7\_Model4.
- From the **Signal Processing / Signal Operations** library, drag a **Sample and Hold** block into your controller.
- From the **Simulink / Sources** library, drag a **Pulse Generator** into your controller
- To start, we will use specify a sample time of 100 ms.

34

## Controller Cycle Time

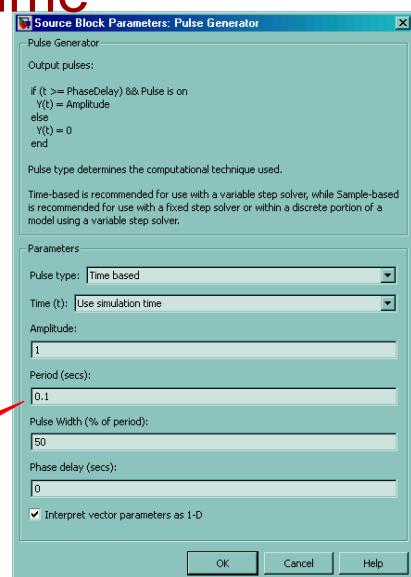




35

## Controller Cycle Time

- To set our 10 ms trigger, double click on the Pulse Generator and set the Period to 0.01 (10 ms)



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

36

## Controller Cycle Time

- Thinking back to our step input work, we have introduced a physical step into our system!
  - The torque request undergoes a step change every controller cycle time.
  - If the cycle time is too long, these steps could be large.
  - Larger step changes can be felt or heard.
- Lets simulate the system and see if a 100 ms time step on the output affects the controller's performance.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Controller Cycle Time

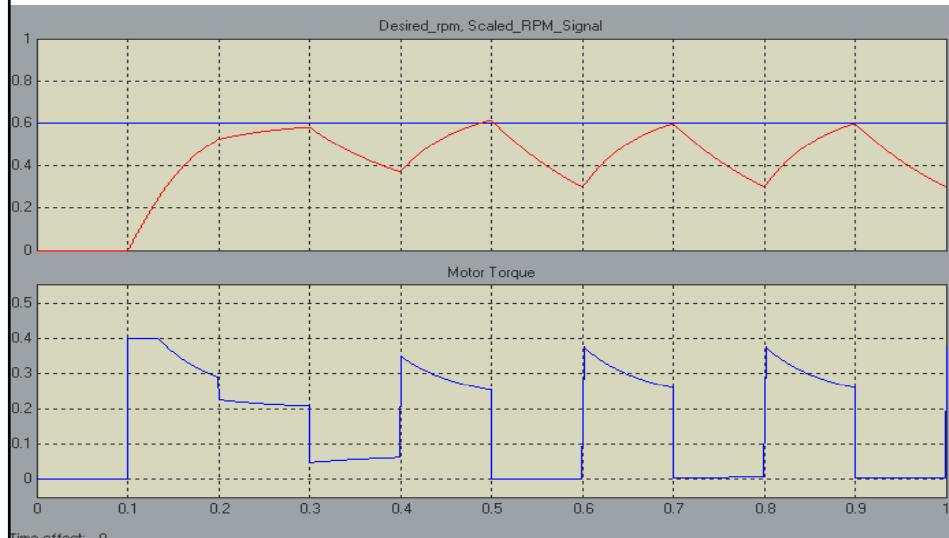
37

- Set the following on your model:
  - rpm slider to 0.6
  - Load slider to 2 bulbs
  - The IC to zero
  - The feedback gain to 10,
  - Simulation end time to 1 s
  - The controller sample time to 0.1 seconds. (A very long time.)
  - The simulation max step size to 1e-5.
  - When the simulation runs, you will notice that it oscillates at a 100 ms rate!



## Controller Cycle Time

38

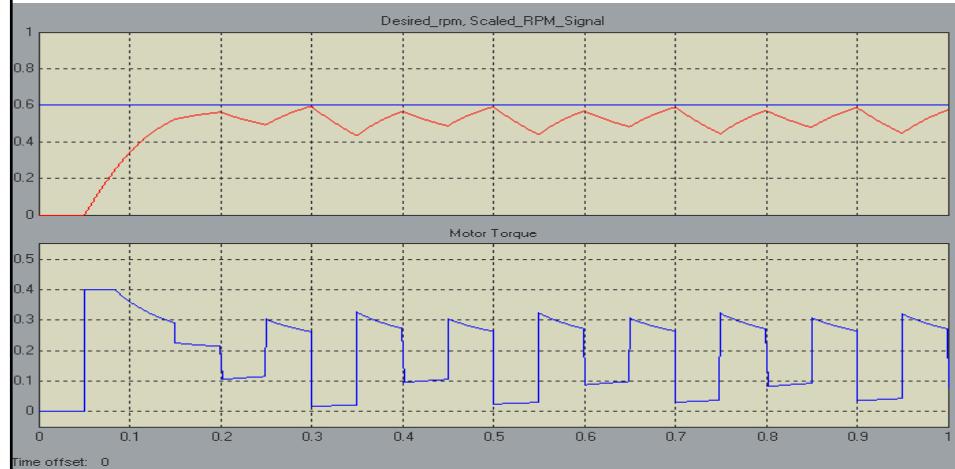




## Controller Cycle Time.

39

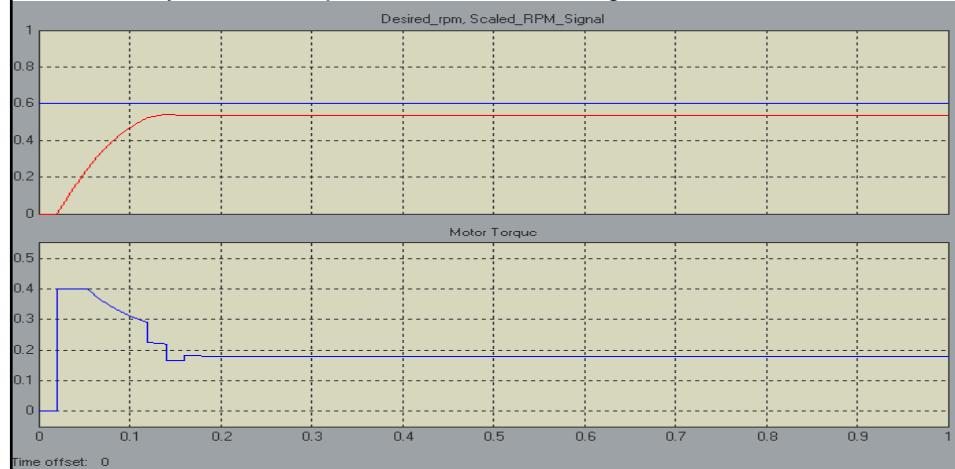
- Try reducing the controller cycle time until the oscillation disappears.
- This simulation is for a cycle time of 50 ms



## Controller Cycle Time.

40

- Try reducing the controller cycle time until the oscillation disappears.
- I found that a sample time of 20 ms is sufficient to stop the oscillation.
- A simulation of this type is necessary to determine how much computing power we will need to run our controller.
- Smaller cycle times require faster hardware targets.





## Lecture 7 Exercise 3

41

Demo \_\_\_\_\_

Determine the maximum value of Sample time that can be used to achieve a stable feedback system using the conditions specified below:

	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs
Gain = 10				
Gain = 20				
Gain = 50				
Gain = 100				



## Controller Cycle Time

42

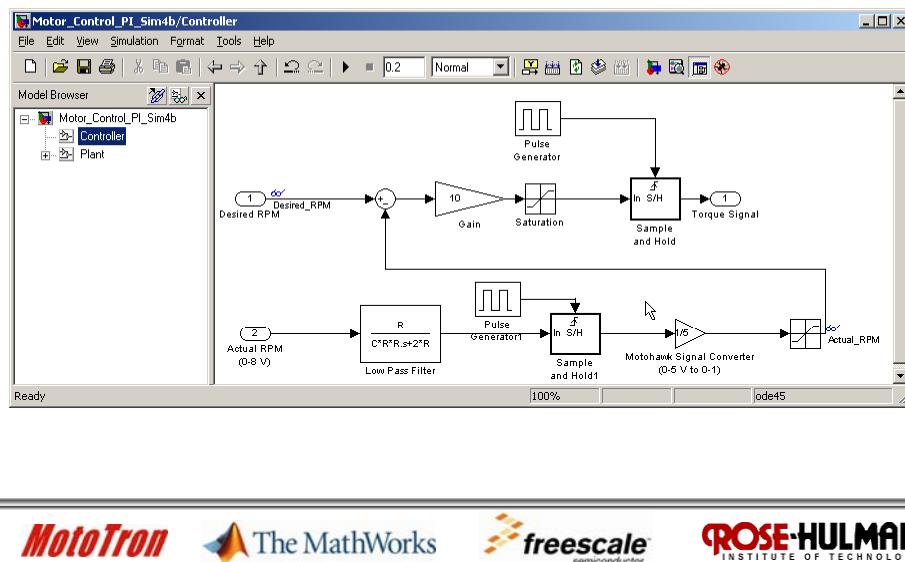
- Next, we will model the controller cycle time effect on the input signal.
- Remember that the ECU will only sample the input once every cycle time.
- Bring in another **Pulse Generator** and **Sample & Hold** and put it on the generator side just after the low pass filter.
- The pulse generator will have the same properties as the previous one we added.





43

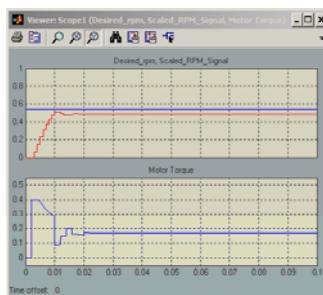
## Controller Cycle Time

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

## Controller Cycle Time

44

- Set the period on both pulse generators to 1 ms. Leave all other settings the same and run a simulation. (Gain = 10, 2 bulbs, max step size = 1e-5).
- The system is still stable because we actually have the same amount of delay even though we have two sample and holds.
- In this case the output is based on the value of the input from the previous cycle.
- This is because both sample and holds are triggered at the same time.
- In the first example the output was based on the inputs at the present sample.





45

## Controller Cycle Time Conclusion

- Controller cycle time adds delay to a feedback loop that can cause a system to oscillate or overshoot.
- We can alleviate some of these affects by changing the gains of our system.
- To model a system, we should include all sampling delays
  - Controller cycle time.
  - CAN message rates.
  - Other.



## Proportional and Integral Control





47

## PI Control

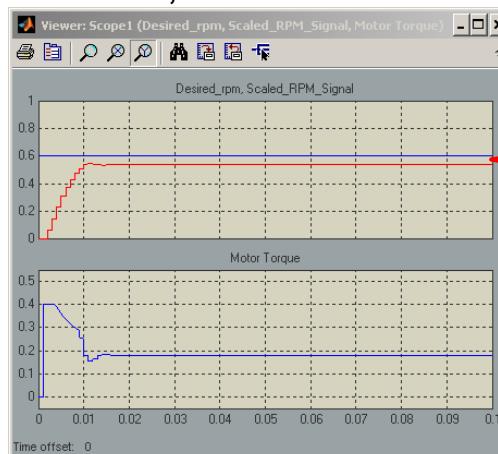
- One of our goals is to design a controller for our system.
- Now that we have a good plant model, we can use all of our knowledge of control systems to design controllers. (And simulate them!)
- We will look at adding an integrator to our system because:
  - A proportional gain controller yields an error that is inversely proportional to the gain.
  - Too high of a proportional gain can make the system oscillate.
- Update your model to version Lecture7\_Model6.



48

## PI Control

- Set your Controller Cycle Times back to 1 ms
- Set your proportional gain to 10, the desired rpm slider to 0.6, and the load slider to 2 bulbs.





49

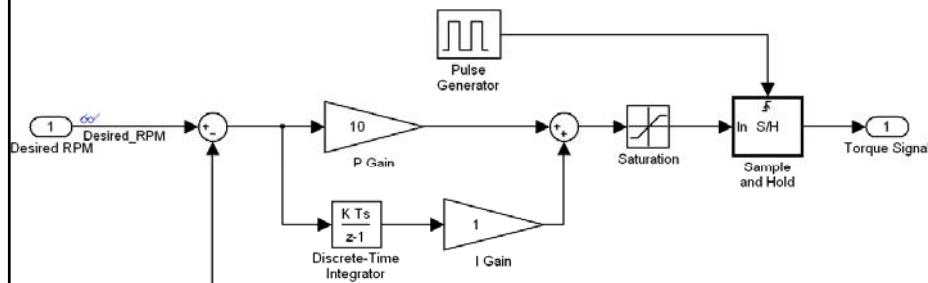
## PI Control

- To reduce this error we will add an integrator to our controller.
- We could use a continuous time integrator for Simulink simulations. 
- When we implement our controller on an ECU, a discrete solver is usually used and a continuous integrator will not work.
- We will use a discrete time integrator.

50

## PI Control

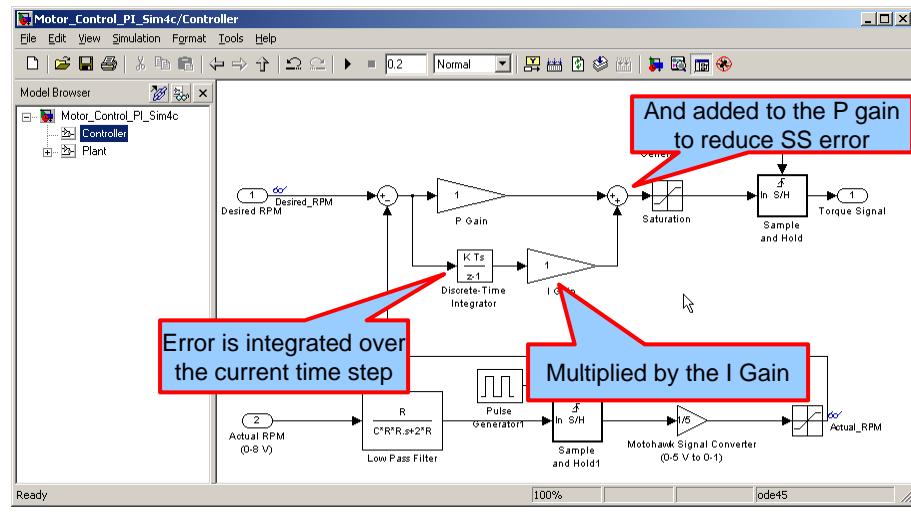
- From the **Simulink / Discrete** library drag in a **Discrete-Time Integrator**
- Drag in a gain and a sum as well
- Hook up as shown below and on the next slide





51

## PI Control



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

52

## Discrete Time Integrator

- This integrator performs a forward Euler method. (Basically a sum of rectangles.)
- The block implements the sum
$$y(n) = y(n-1) + K \cdot T \cdot u(n-1)$$
- K is the gain and is equal to 1 for our application.
- T is the sampling period (the time between samples).

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



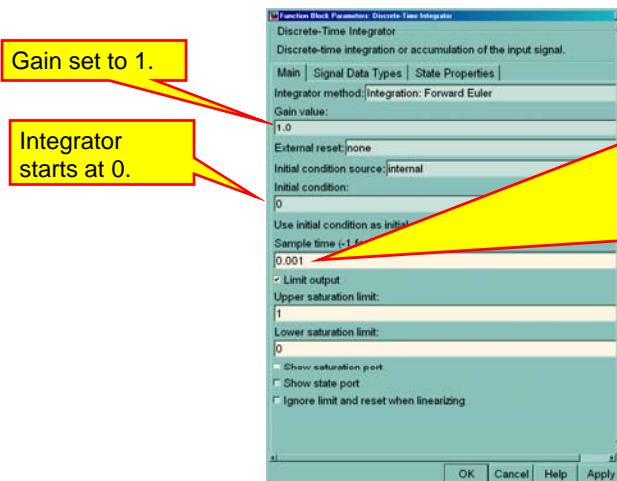
## Discrete Time Integrator

53

- With K=1 we have the equation  
 $y(n) = y(n-1) + T^*u(n-1)$
- The present output is the previous output plus the previous input times the time slice T
- Note that  $T^*u(n-1)$  is the area of a rectangle.
- We are summing up rectangles.
- Set the integrator properties as shown next.

## Discrete Tim Integrator

54



Sample time set the same as for our controller cycle time. When we deploy this controller on the ECU, the integrator sample time will be the controller cycle time.

We should probably define this in our init file and use it for the integrator and two pulse generators.

Do that!



The MathWorks

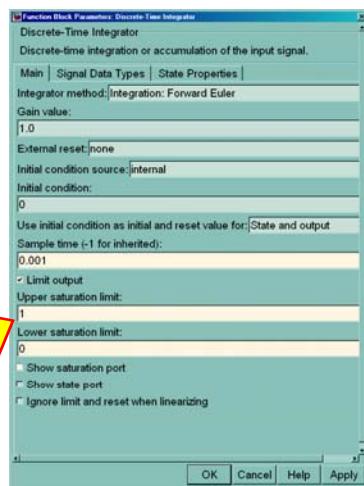




## Discrete Tim Integrator

55

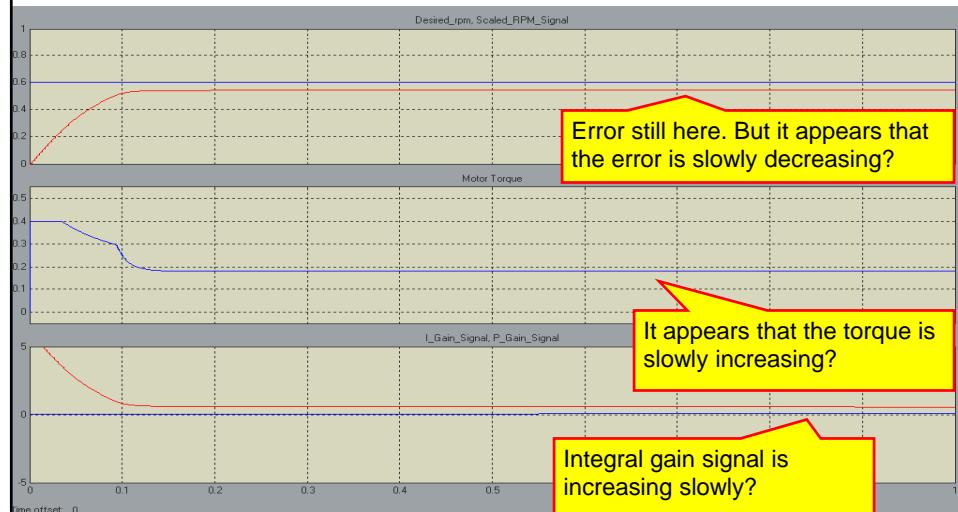
- Integrators can integrate to infinity (it takes a long time!)
- Suppose we turn on too many bulbs. The motor is not strong enough to reach the desired rpm and the error will be large.
- The integrator will integrate this error, possibly for a long time until you change the number of bulbs.
- If the integrator does not have saturation limits, the integrator will integrate to a large value.
- When you do finally turn off a bulb, the integrator value decreases slowly and causes the motor to spin wildly until the integrator has time to integrate back to an appropriate value.
- This overshoot is called integrator wind up.
- To prevent it, add saturation limits.



## Discrete Time Integrator

56

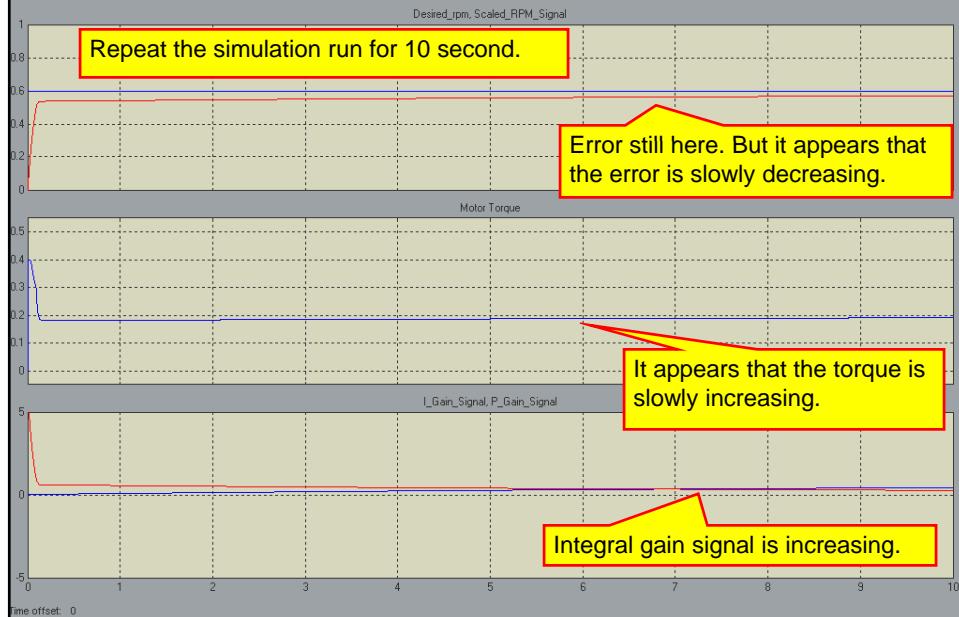
- Run a simulation with a proportional gain of 10 and integral gain of 1, load = 2 bulbs, and a desired speed of 0.6. Let the simulation run for 1 second.





## Discrete Time Integrator

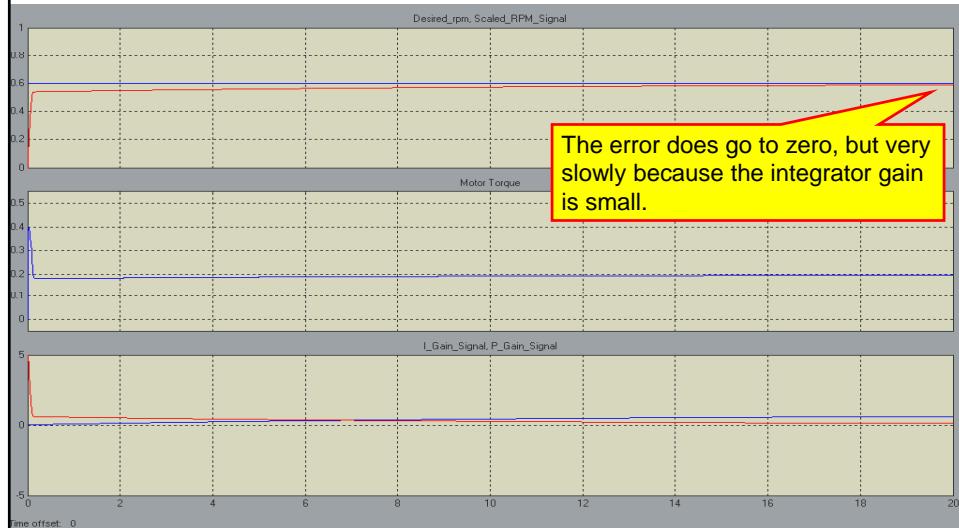
57



## Discrete Time Integrator

58

- Change the simulation to allow it to run for 20 seconds.

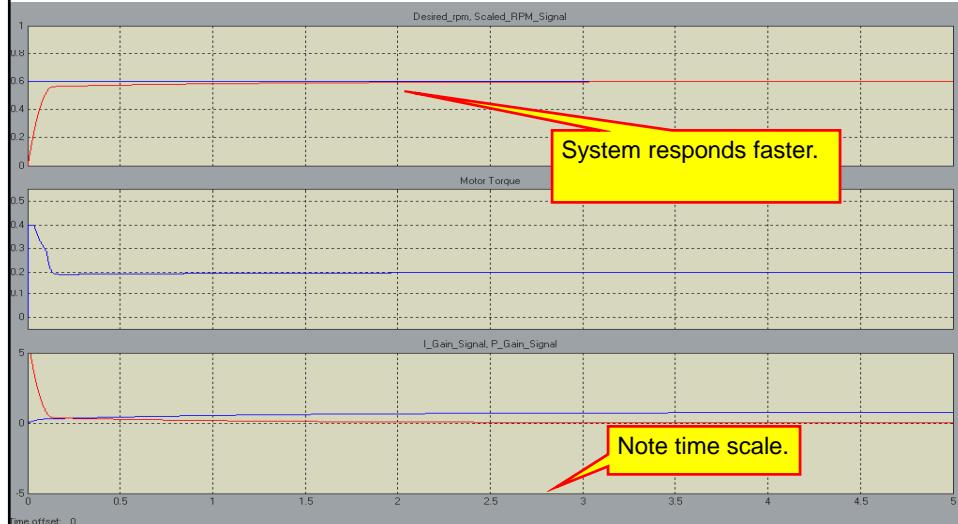




## Discrete Time Integrator

59

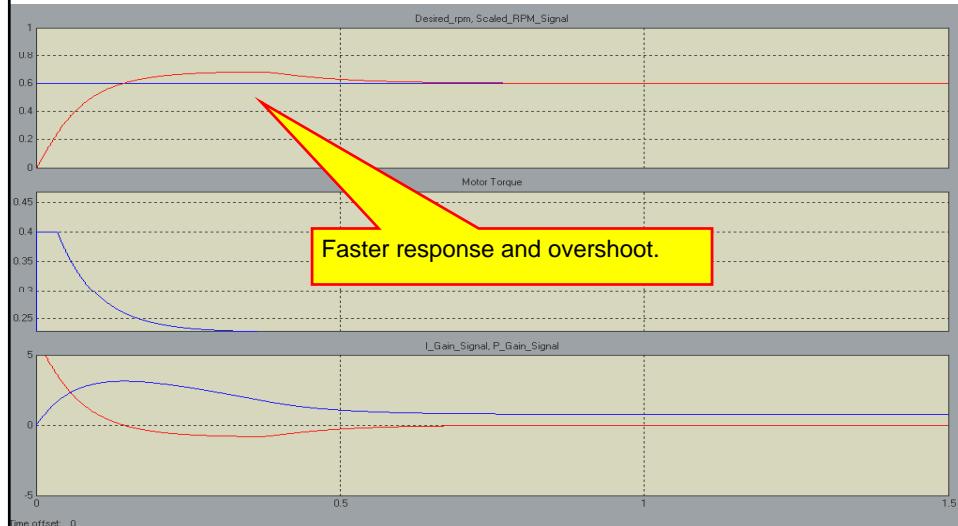
- Next, set I\_Gain = 10, and P gain = 10. This should make the integrator respond faster.



## Discrete Time Integrator

60

- Next, set I\_Gain = 100, and P gain = 10. This should make the integrator respond faster.





61

## Lecture 7 Demo 1

- Demo your PI controller.
- Show how changing the P and I gains affect the system response.

Demo \_\_\_\_\_



The MathWorks



freescale  
semiconductor



## Discrete Time Integrator

62

- For the previous simulations, we were near the power limitations of the motor.
- We will do some testing at a new set point
- Set the following:
  - Desired speed slider = 0.3
  - Number of bulbs slider = 1
  - Initial condition = 0
  - All sample times = 0.001
  - Simulation time set appropriately.



The MathWorks



freescale  
semiconductor

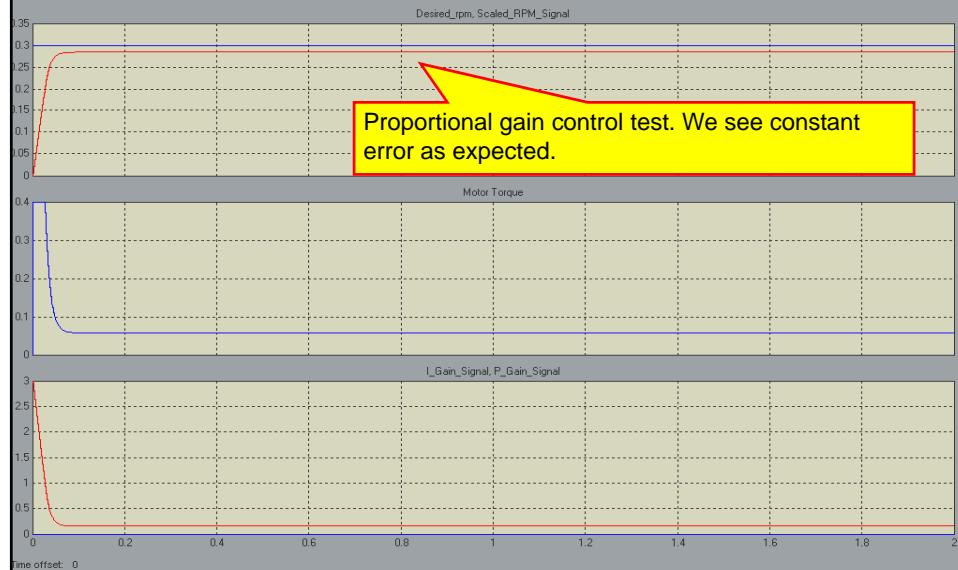




## Discrete Time Integrator

63

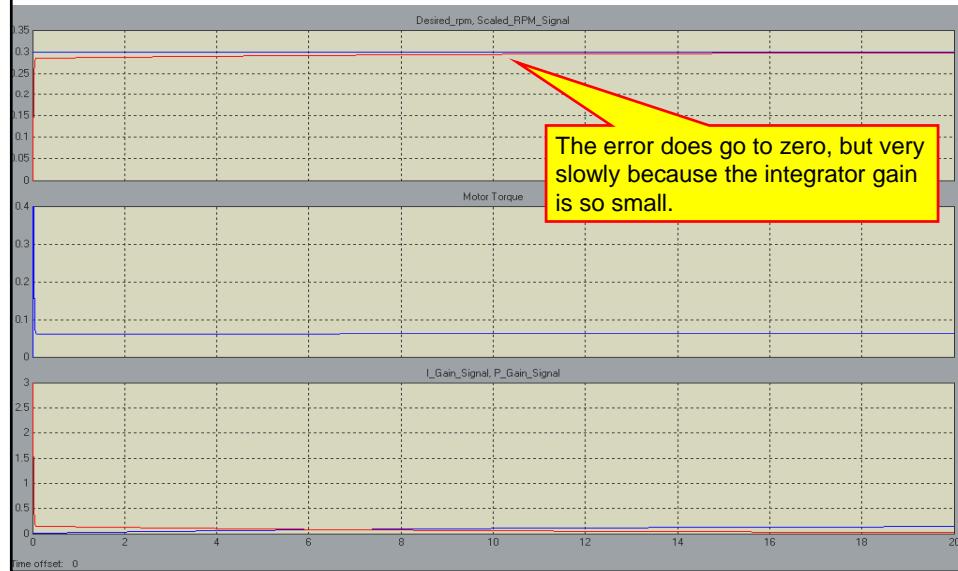
- P\_Gain = 10, I\_Gain = 0



## Discrete Time Integrator

64

- P\_Gain = 10, I\_Gain = 1.

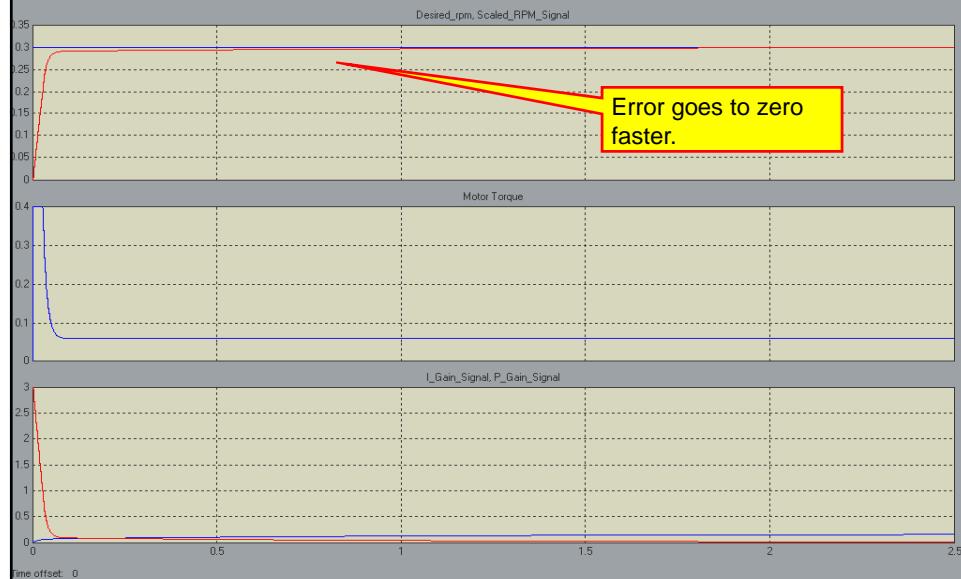




## Discrete Time Integrator

65

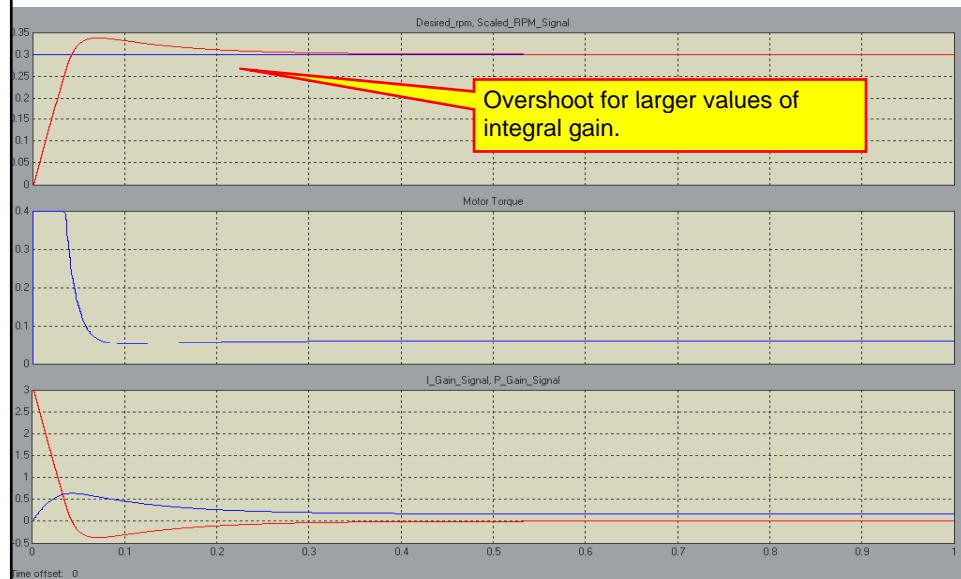
- P\_Gain = 10, I\_Gain = 10



## Discrete Time Integrator

66

- P\_Gain = 10, I\_Gain = 100

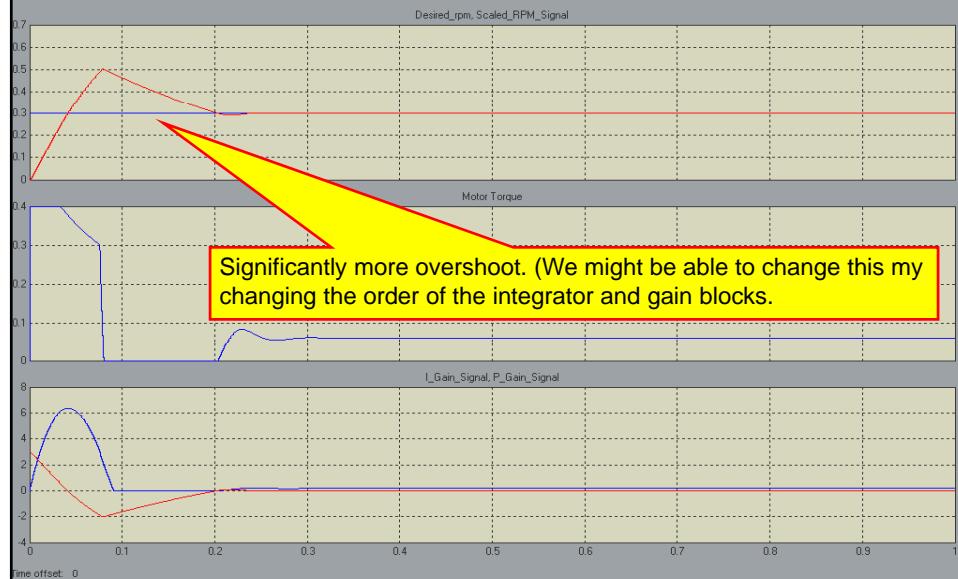




67

## Discrete Time Integrator

- P\_Gain = 10, I\_Gain = 1000

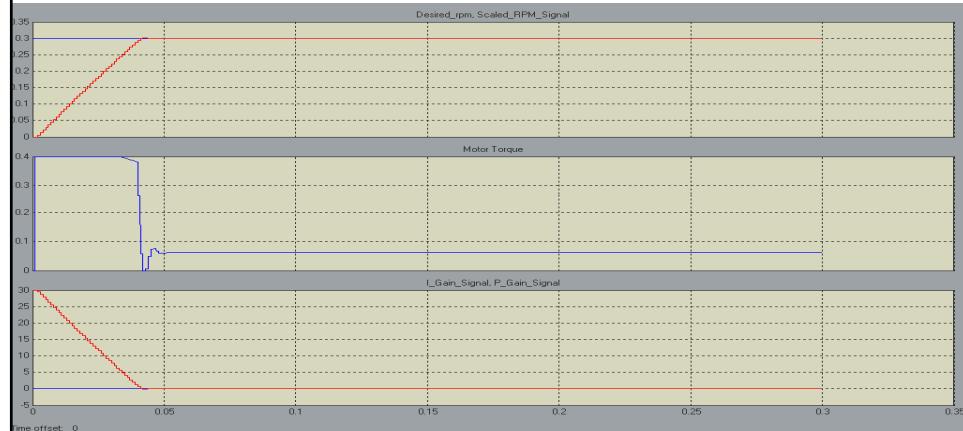


68

## Lecture 7 Exercise 4

Demo \_\_\_\_\_

- Repeat the previous procedure for different values of proportional gain.
- Determine the proportional and integral gains to achieve a plot close to the one below.





69

## Discrete Time Integrator

- I like the system response with P\_Gain = 100 and I\_Gain = 10.
- We will hold these gains constant and hold the number of bulbs constant at 1 bulb.
- We will observe the system response as we change the speed slider between 0 and 0.6.
- Change the simulation ending time to inf and decrease the max simulation step size to make the simulation run slowly.



70



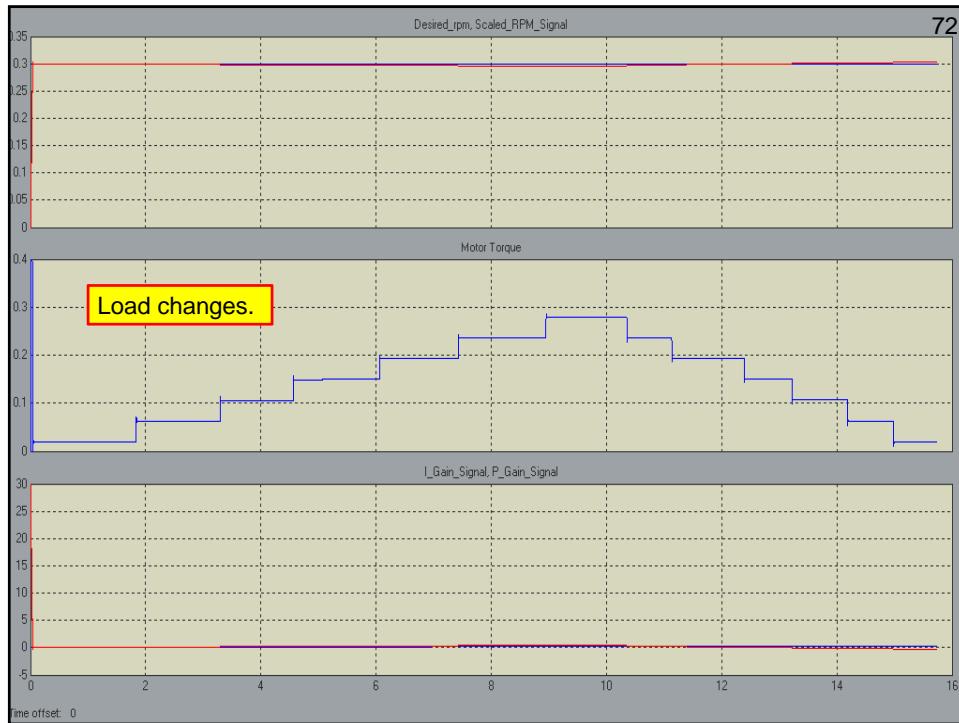


71

## Discrete Time Integrator

- Next, keep the speed slider constant at 0.3 and change the bulb load from 0 to 6.
- Vary the load up and down.

72





73

## PI Controller

- Feel free to experiment with the controller gains to see their effect.
- Tuning is the process of choosing the gains to achieve the desired response.
- With this PI controller, we have a lot of flexibility.



The MathWorks



## Lecture 7 Demo 2

74

- Demo your PI controller with 1 bulb load.
- Show how changing the P and I gains affect the system response.
- Determine values of P and I gains for optimum response.

Demo \_\_\_\_\_



The MathWorks

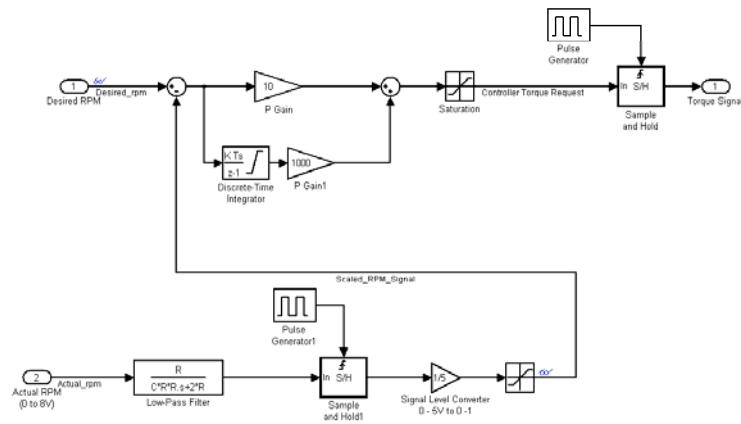




75

## Lecture 7 Exercise 5

Investigate the difference between the two controllers below. Note that the order of the integrator and gain has been switched in the two models:



**MotoTron**

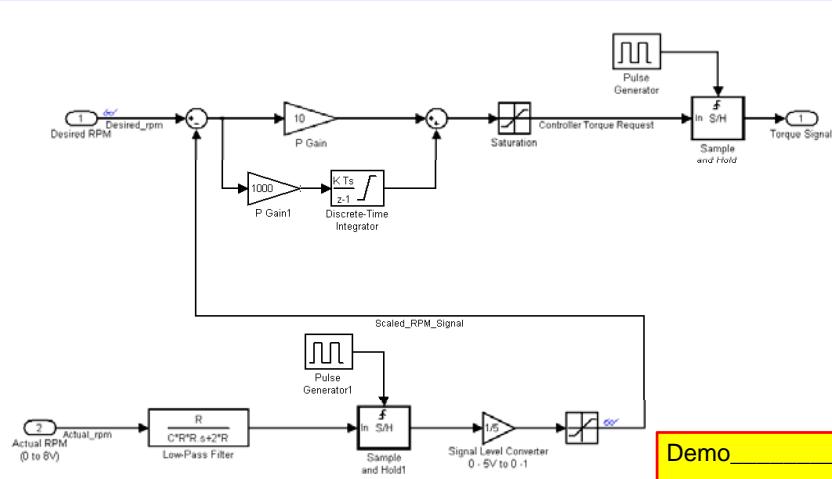
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

76

## Lecture 7 Exercise 5



Demo

Investigate the difference in these two methods for:

1. The affect of the placement on overshoot at high gain.
2. The affect of the placement on changing the integral gain while the system is running.



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

## Any Questions?

## Start Project 1



The MathWorks





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



# Introduction to Model-Based Systems Design

## Lecture 8: xPC Real-Time Systems 1



The MathWorks



## Outline

2

- Setting up an xPC Target
- Running Real-Time Simulation
  - Proportional Control Model
  - Integral Control Model
- Tuning Parameters
- Using xPC Explorer
- Blocks and Gauges



The MathWorks





3

## Real-Time Simulations

- We would like to run real-time simulations for several reasons:
  - Further debug our controller and model.
  - Obtain a feel for how the model responds in real-time.
  - Tune the control parameters for a desired response.
  - See how changing inputs and model parameters affect model and controller operation.



The MathWorks



4

## Terminology

- Host Computer – The computer where you run Matlab. The real-time model will not run on this computer. We will create our models on this computer.
- Target Computer (Platform) – The computer on which our real-time model will run.
- We will use the host computer to communicate with and monitor the target.
- The target computer can run the plant, the controller, or both depending on the type of verification we want to do.



The MathWorks





5

## What is xPC

- xPC allows you to use a personal computer as a real-time target.
  - Inexpensive
  - Most of us have an older PC we can use.
- Communication through TCP/IP or RS-232
- Can tune parameters
- Supports I/O hardware in PC such as A/D, D/A, DIO, CAN, etc



The MathWorks

freescale<sup>®</sup>  
semiconductor

6

## What You Need (For this course)

- An Old PC (can be almost any PC with an Intel 386 or 486 Pentium or AMD K5 or K6/Athlon processor)
- A floppy disk (Not required anymore)
- Microsoft Visual Studio 2005 or Open Watcom C/C++ (Version 1.3)
- An Ethernet Card
  - Card Specifics (Driver/Slot type)
  - Network Specifics
    - IP Address
    - Gateway Address
    - LAN Subnet Mask



The MathWorks

freescale<sup>®</sup>  
semiconductor



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

## Setting Up xPC

7

- For our example, we'll use a PC with a Pentium III and an Ethernet card.
- From the Matlab prompt, type the command **>>xpcexplr**
- This will allow us to both setup and monitor the target PC.



## Visual Studio Compiler

8

- First, we will select the compiler. We will assume that you are using the Microsoft Visual Studio 2005.
- We will assume that you have already installed the Microsoft compiler and that it is installed in directory C:\Program Files\Microsoft Visual Studio 8.

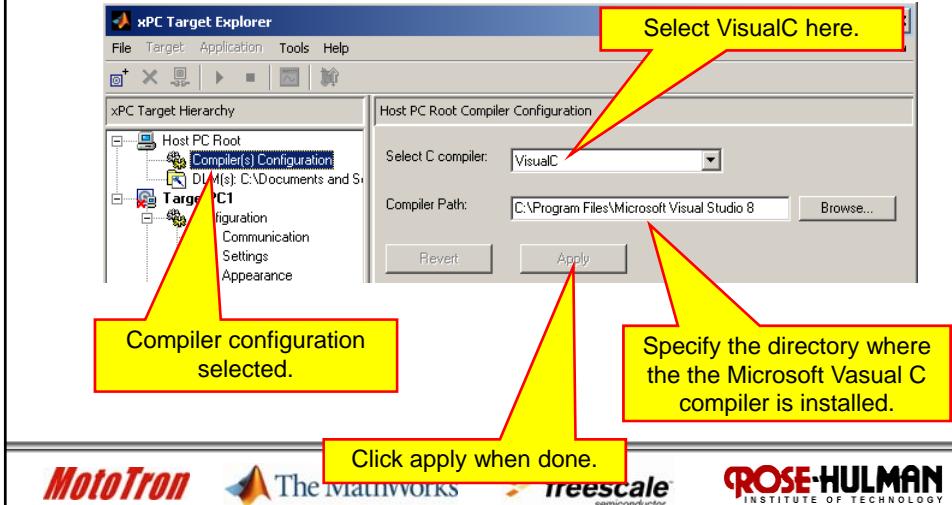




## Visual Studio Compiler Setup

9

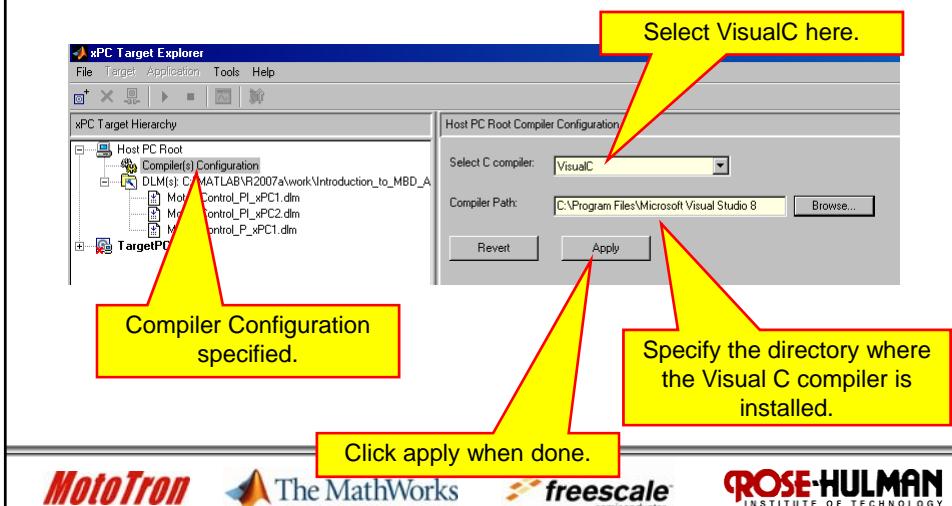
- The compiler is specified using the xPC Explorer.



## Visual Studio Compiler Setup

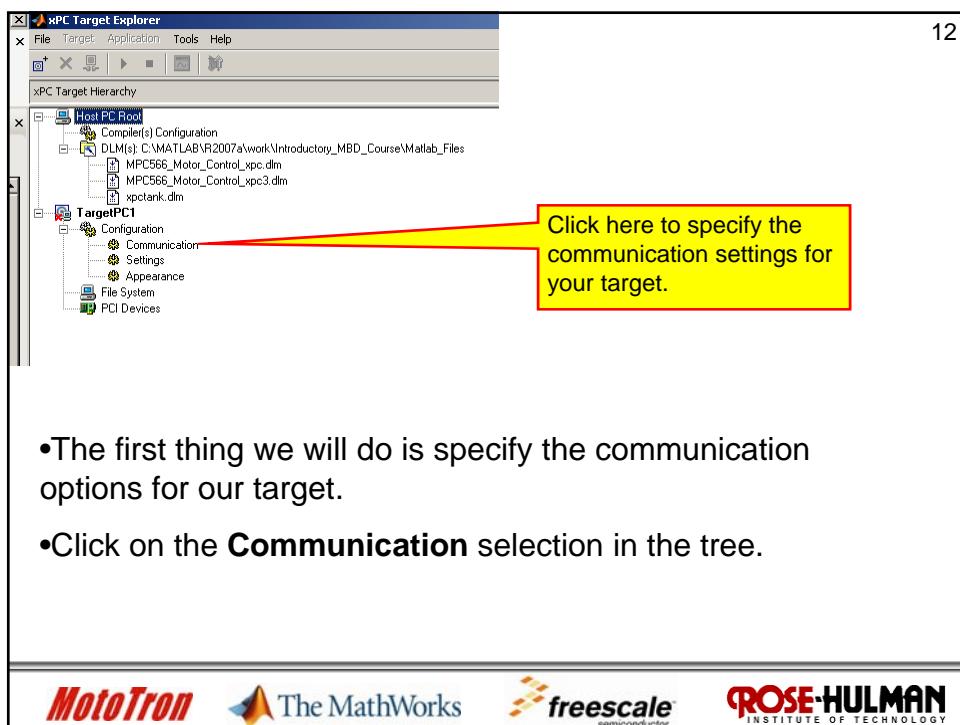
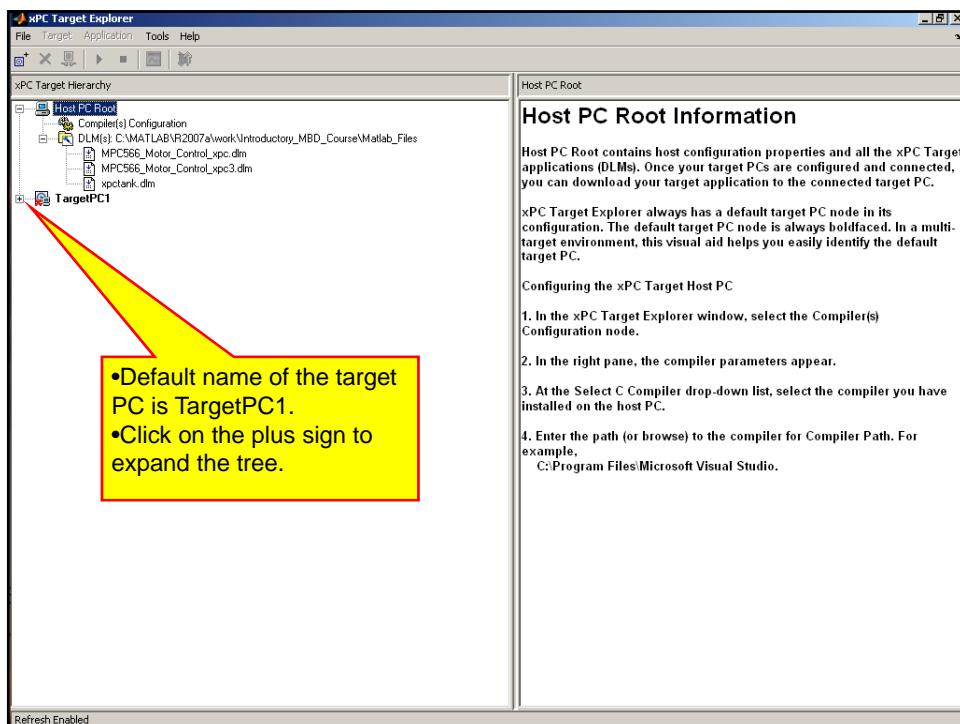
10

- If you have Microsoft Visual Studio 2005 installed, your settings will be as shown:





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



The MathWorks





13

The screenshot shows the xPC Target Explorer interface. On the left, the 'xPC Target Hierarchy' tree view shows 'Host PC Root' and 'TargetPC1' nodes. Under 'TargetPC1', there are 'Configuration' (selected), 'Communication' (highlighted with a yellow box), 'Settings', and 'Appearance' nodes. The main panel is titled 'TargetPC1 Communication Component'. It includes sections for 'Communication protocol' (set to 'TCP/IP'), 'Target PC TCP/IP configuration' (IP address: 137.112.41.60, port: 22222, subnet mask: 255.255.255.0, gateway: 137.112.41.1), 'RS-232 configuration' (port: COM1, baud rate: 115200), and buttons for 'Revert' and 'Apply'. A yellow callout box points to the 'Communication' tab in the tree view.

- Selections are hardware dependent.
- Obtain network information from your network administrator if you do not know it.
- We will look briefly at some of these options.

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

14

## xPC Setup

- We can specify communication through an RS-232 cable or a TCP/IP connection:
- We will select the **TCP/IP** connection

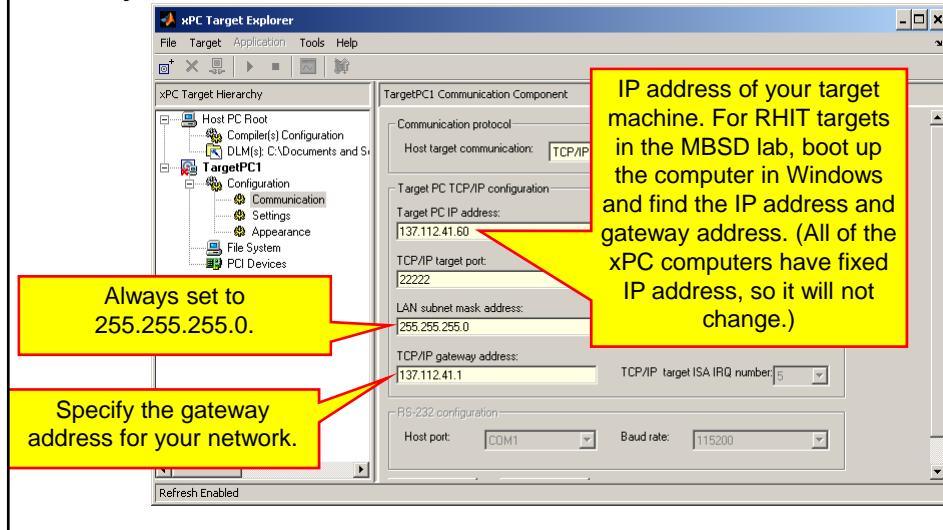
The screenshot shows the same xPC Target Explorer interface as the previous slide. The 'Communication' tab is selected in the tree view. In the main panel, the 'Communication protocol' dropdown menu is open, showing 'TCP/IP' (selected) and 'RS-232'. A yellow callout box points to this dropdown menu. The rest of the configuration panel remains the same as in the previous slide.



## xPC Setup

15

- Specify the IP addresses for your target system.



## xPC Setup

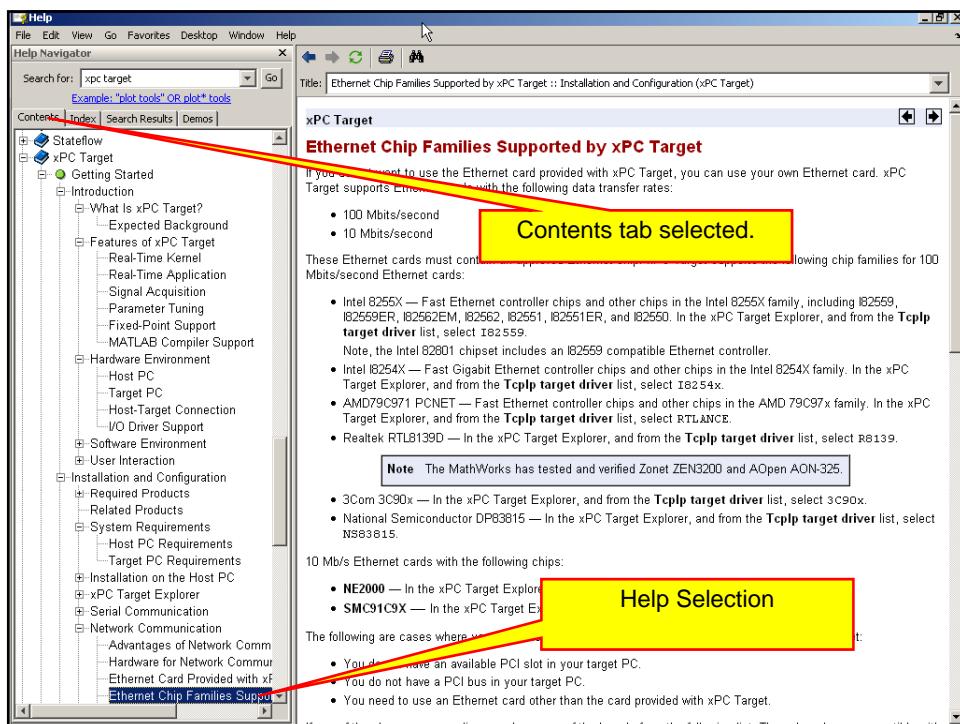
16

- For the Ethernet card, we need to know the card chip set and the bus type (ISA or PCI) into which the card is plugged .
- If you do not know this information, you might be able to use the Windows Device Manager to find it.
- A list of the supported Ethernet cards can be found in the Matlab xPC help with the following topic: **Ethernet Chip Families Supported by xPC Target**





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



## xPC Setup

18

- Select the driver/chip set for your Ethernet card:

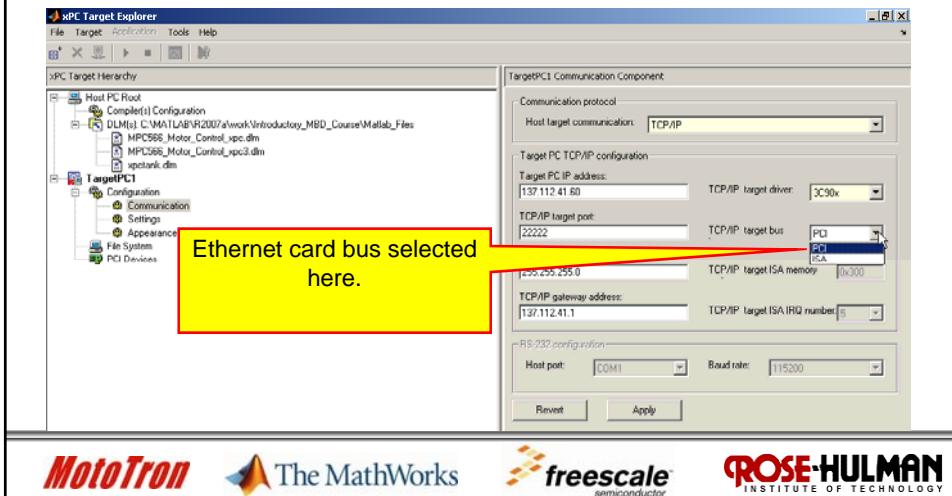
Select Ethernet card chip set here. For the RHIT MBSD lab, select **R8139**.



## xPC Setup

19

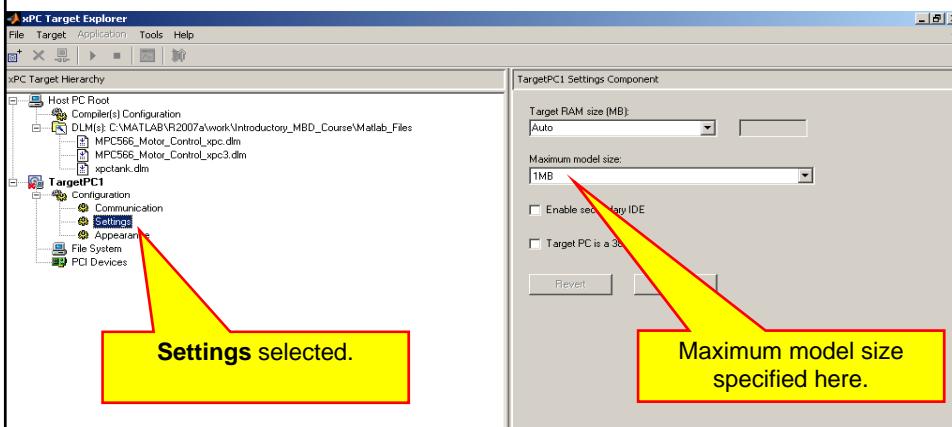
- Select the type of bus into which the Ethernet card is plugged:

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

## xPC Setup

20

- Next, select **Settings**.
- Since our model is small, we will use the default settings for model size:

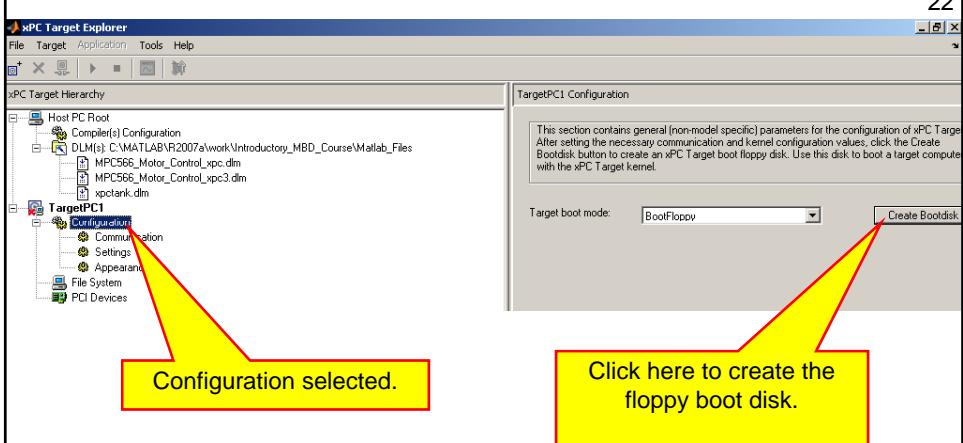




## xPC Setup

21

- Next, we will create a boot floppy for our target PC.
- Hopefully your host PC (the one running Matlab) has a floppy disk drive.
- Hopefully you can find a floppy disk.
- Select **Configuration** in the tree:



Click the **Create Bootdisk** button.

Note that everything on the disk will be erased.



## xPC Setup

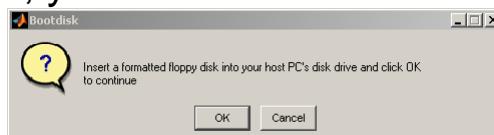
23

- Incase you haven't used a floppy disk in the past several years, remember the following:
  - You may need to format the disk before clicking the **Create Bootdisk** button.
  - Make sure that the write protect switch on the disk is set to allow writing (you should **not** be able to see through the hole.)

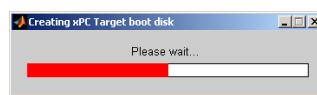
## xPC Setup

24

- When you click the **Create Bootdisk** button, you should see the dialog box:



- As the disk is created, you should see the dialog box:



When done, the dialog box will disappear.



25

## xPC Setup

- Next, place the bootdisk in your target PC and turn it on.
- Your target PC should display a screen similar to the one shown next.
- You can ignore the error about no accessible disk. We will not be using the target PC's disk.



The MathWorks

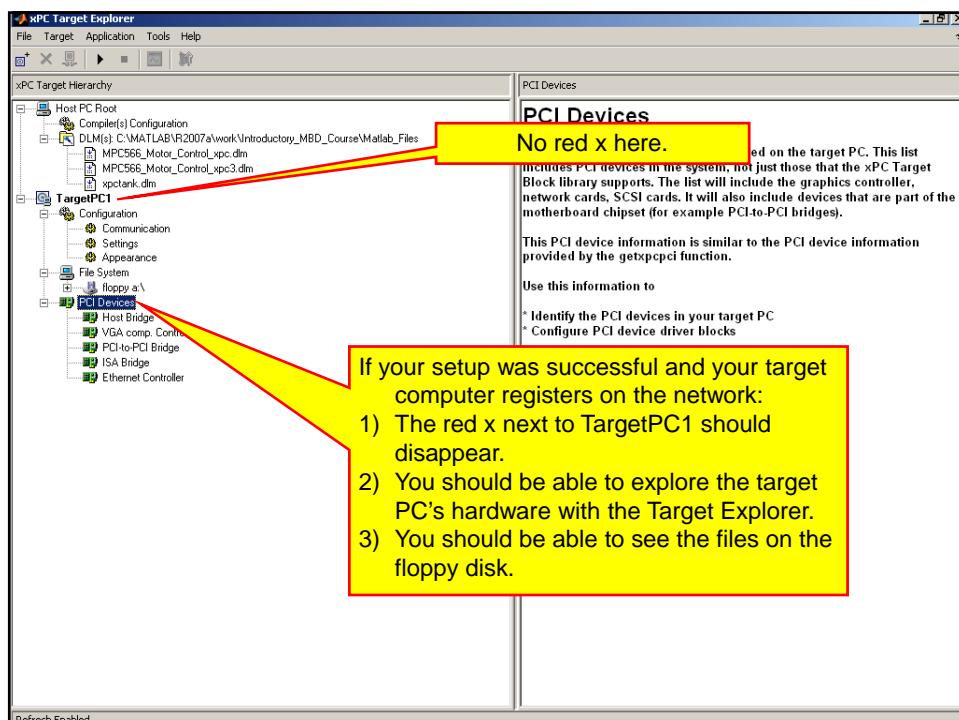
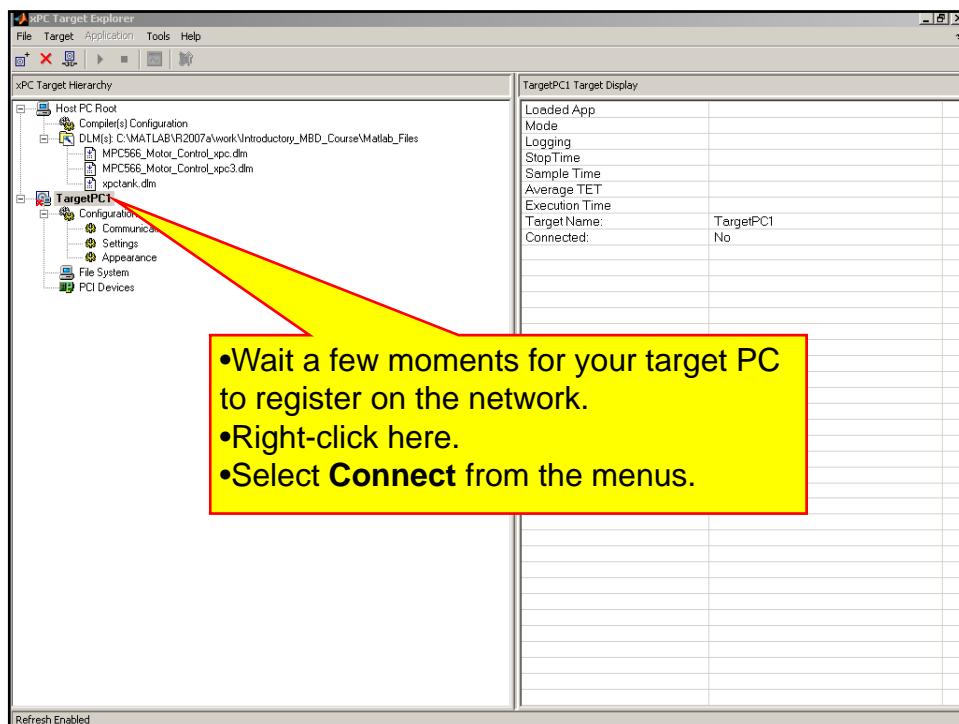


We'll ignore this.

- Next, switch back to the xPC Target Explorer window.
- Right click on the text "TargetPC1."



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





29

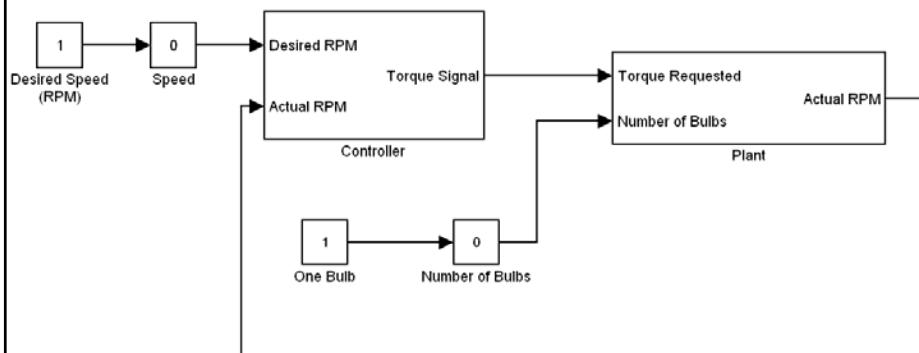
## xPC Setup

- Our target setup is now complete.
- We are now ready to modify our model to run on the target computer.
- We will use the model with proportional feedback as a starting point.(Passed out as Lecture8\_Model0.mdl)
- Later, we can use the same process to simulate our more complicated feedback models.
- Open the model with proportional feedback and then rename it Lecture8\_Model1.mdl.
- Make sure that this model has the flywheel inertia in it, with the inertia specified as  $10^{-4} \text{ m}^2\text{kg}$

30

## Model Modifications

- Add sliders for the speed and number of bulbs.





31

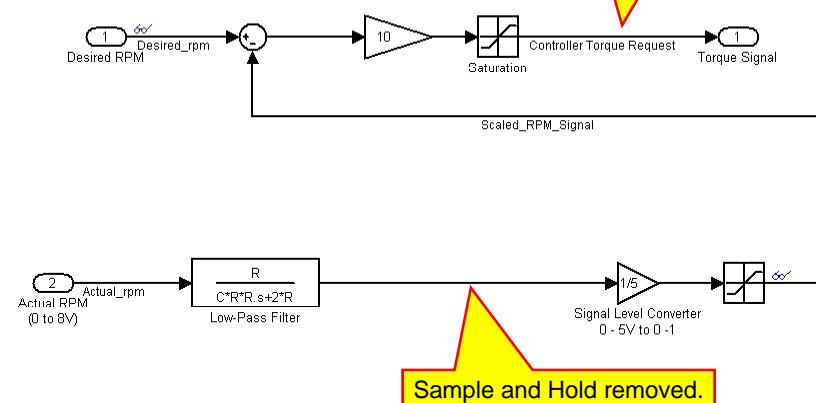
## Model Modifications

- Remove the Sample and Hold locks we added to model the controller response time.
- These delays will be modeled by the fixed step size.

## Model Modifications

32

- Controller Block

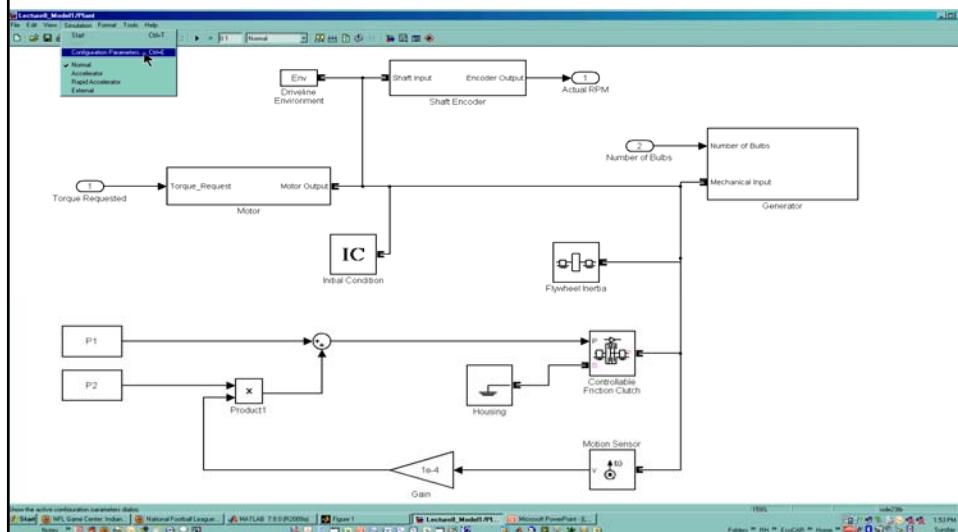




33

## Simulation Parameter Setup

- Select **Simulation** and then **Configuration Parameters** from the Simulink menus



34

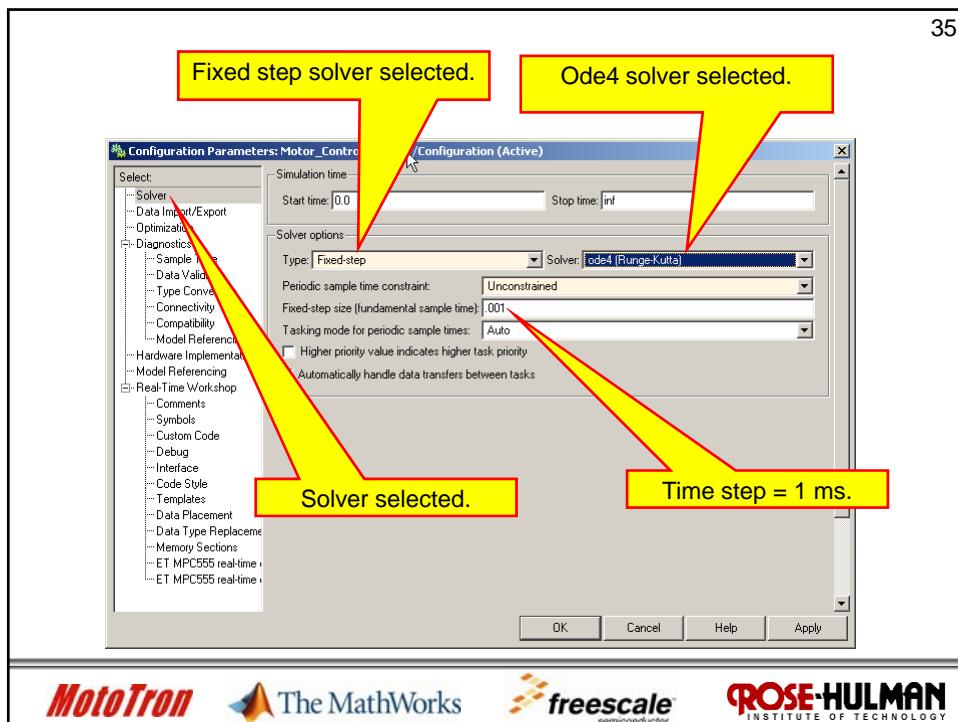
## Simulation Parameter Setup

- There are a number of parameters we need to change.
- Select the **Solver** tab and specify the following parameters
  - Type: Fixed-step
  - Solver: ode4 (Runge-Kutta)
  - Fixed-Step Size: 0.001 (1 ms time step)
- These selections are shown on the following slide.





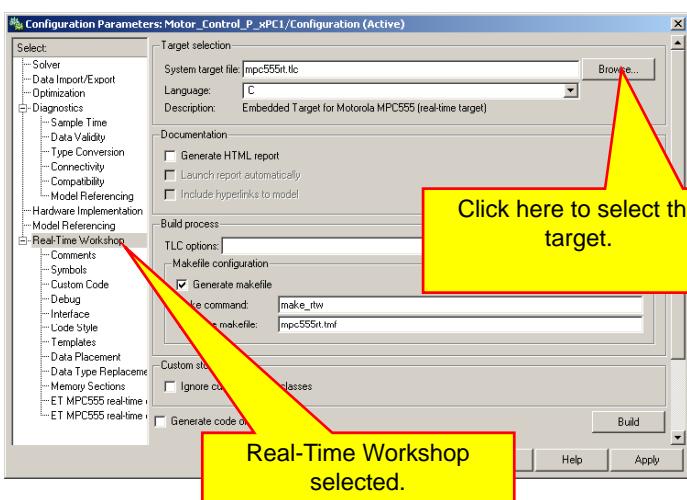
35

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

36

## Simulation Parameter Setup

Next, select the **Real-Time Workshop** tab:

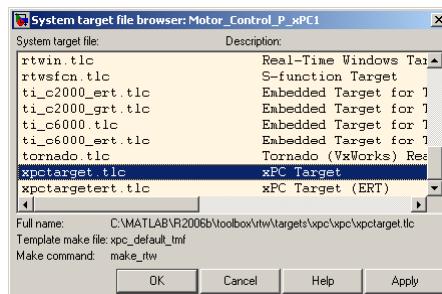




## Simulation Parameter Setup

37

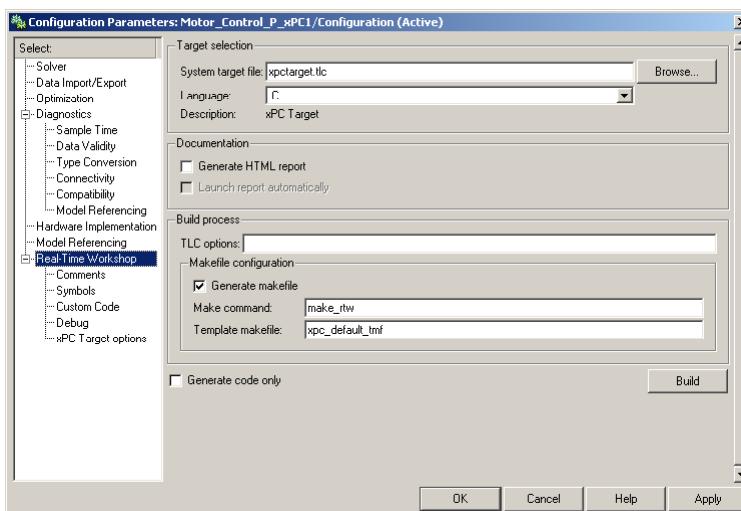
- We need to specify that we will be using a xPC target as our target system.
- Click the **Browse** button, select the file named **xpctarget.tlc**, and then select **OK**.



## Simulation Parameter Setup

38

- Your dialog box should look as shown.

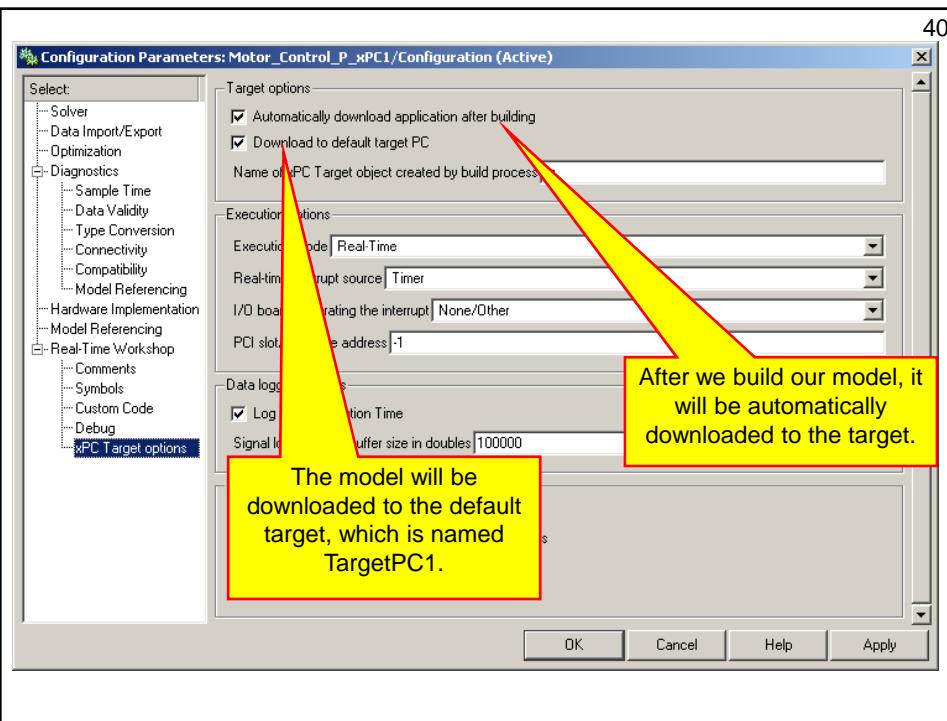




## Simulation Parameter Setup

39

- We do not need to make any other changes.
- However, we will look at the xPC Target options tab to see some other options that control behavior we will see.
- Select the **xPC Target options** tab.

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



41

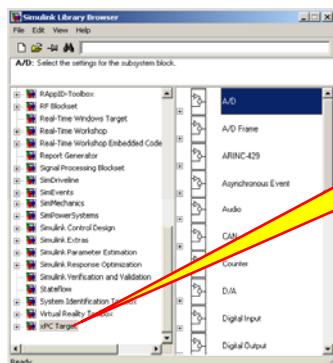
## Simulink xPC Target Library

- The Configuration Parameters are now complete so click the **OK** button to accept the settings.
- Next, we need to place some xPC scopes in our model.
- Open the Simulink Library Browser by selecting **View** and then **Library Browser** from the Simulink menus or clicking the Library Browser  button
- 

42

## Simulink xPC Target Library

- The **xPC Target** library is down at the bottom of the list.
- Click on the **+** to expand the library:



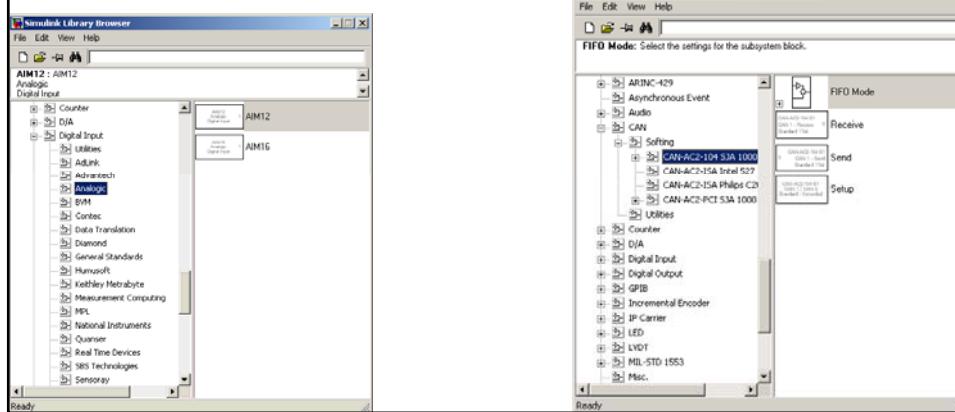
xPC Target library way down here.



## Simulink xPC Target Library

43

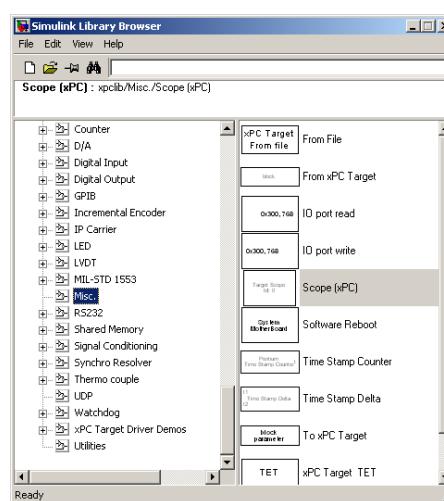
- Take a minute to explore the library.
- Notice that there are both utility blocks like scopes as well as hardware blocks from specific hardware vendors.



## Simulink xPC Target Library

44

- We are looking for the xPC Scope which is in the **Misc** library:
- These blocks will display a trace on the target PC screen.
- Place two scopes in your model as shown.

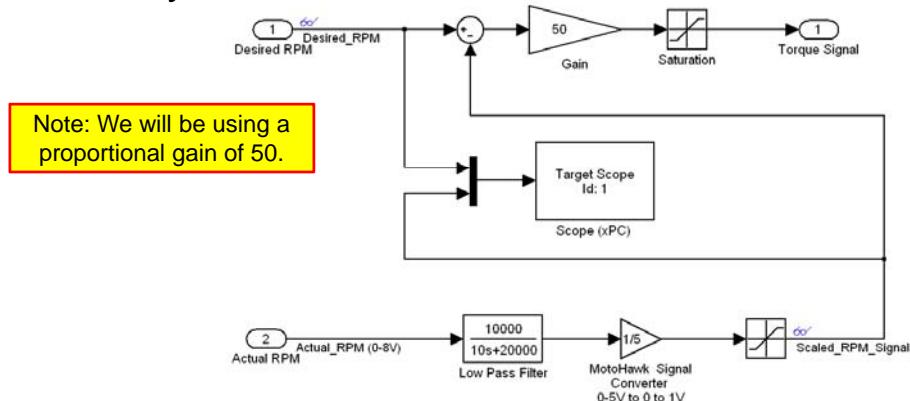




45

## Model Changes

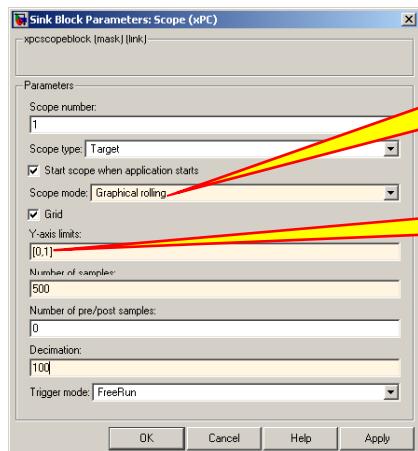
- In the controller, we want to show the desired speed and the motor actual speed on the same plot.
- Modify the controller block as shown:



46

## Model Changes

- Next, double-click on the Target Scope block and make the changes below.



This changes how the plot updates.  
You can try different settings to see  
the effect.

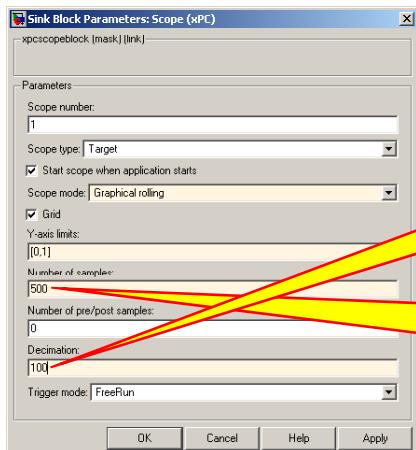
I expect these signals to be between 0  
and 1.



47

## Model Changes

- Other Target Scope block parameters:



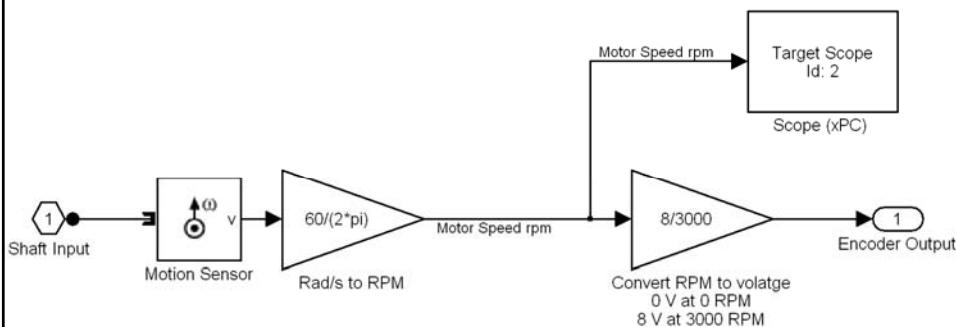
We specified a step size of 1 ms. With a decimation of 100, our scope will only record 1 point for every 100 steps. Thus, our scope will only have a point every 0.1 s.

The total number of points on the scope screen will be 500. Since we have a point every 0.1 s, the scope will display 50 seconds of data.

48

## Model Changes

- Click the **OK** button.
- Place another scope in the Shaft Encoder subsystem of the plant and display the Motor rpm

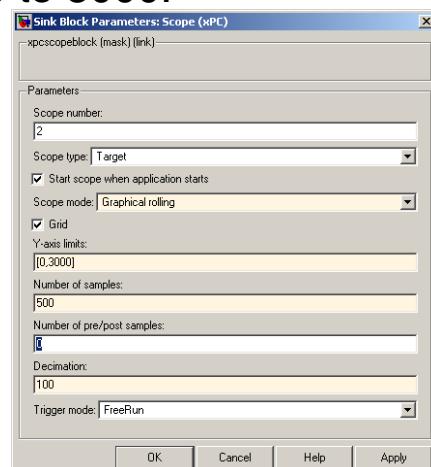




49

## Model Changes

- Make the same changes to the Target Scope block as we did before, except let the y axis range from 0 to 3000:
- Click the **OK** button.
- We are now ready to build our model.
- Type **ctrl-b** in the Simulink window to build and download the model.



50

## Model Changes

- Open The Signal & Scope Manager
- Delete all of the scopes that we created.
- (The scopes will cause Matlab errors in the command window.)
- (The simulation will still run if you do not delete the scopes.)

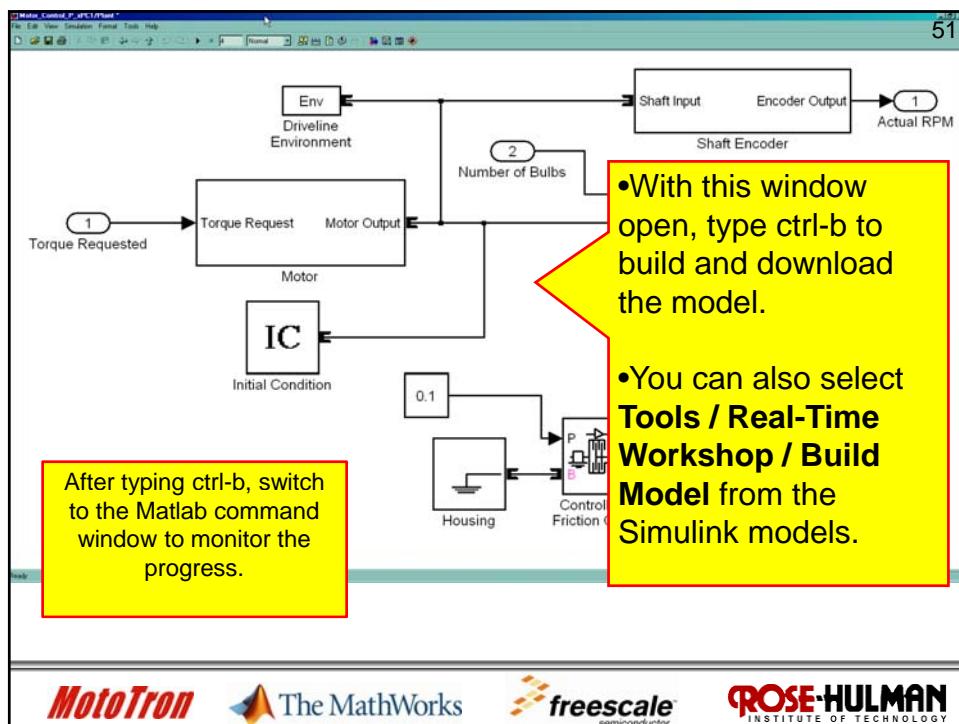


The MathWorks





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



```

## Starting xPC Target build procedure for model: Motor_Control_P_xPC1
## Generating code into build directory: C:\MATLAB\R2007a\work\Introductory_MBD_Course\Matlab_Files\Archive
## Invoking Target Language Compiler on Motor_Control_P_xPC1.rtw
tlc
r
C:\MATLAB\R2007a\work\Introductory_MBD_Course\Matlab_Files\Archive\Motor_Control_P_xPC1_xpc_r
C:\MATLAB\R2007a\toolbox\rtw\targets\xpc\xpc\xpctarget.tlc
--C:\MATLAB\R2007a\work\Introductory_MBD_Course\Matlab_Files\Archive\Motor_Control_P_xPC1_rtw
TC:\MATLAB\R2006b\toolbox\rtw\targets\xpc\xpc
TC:\MATLAB\R2006b\toolbox\rtw\targets\xpc\target\build\xpcblocks\tlc_c
TC:\MATLAB\R2006b\toolbox\physmod\pm_util\pm_util\tlc_c
IC:\MATLAB\R2007a\work\Introductory_MBD_Course\Matlab_Files\Archive\Motor_Control_P_xPC1_xpc
IC:\MATLAB\R2006b\rtw\ctlc\lib
IC:\MATLAB\R2006b\rtw\ctlc\lib
--IC:\MATLAB\R2006b\rtw\ctlc\blocks
--IC:\MATLAB\R2006b\rtw\ctlc\fixpt
--IC:\MATLAB\R2006b\stateflow\ctlc
--aForceIntegerDowncast=0
--aFoldNonRolledExpr=1
--aInlineInvariantSignals=0
--aInlineParameters=0
--aLocalBlockOutputs=0
--aRollThreshold=5
--aStructRepType=0
--aGenCodeOnly=0
--aRTWVerbose=1
--aIncludeHyperlinkInReport=0
--aLaunchReport=0
--aForceParamTraitComments=0
--aGenerateComments=1
--aIgnoreCustomStorageClasses=1
--aTwoHierarchicalIntDef=0

```



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

File Edit Debug Desktop Window Help C:\MATLAB\R2007a\work\Introductory\_MBD\_Course\Matlab\_Files\Archive\Motor\_Control\_P\_xPC1\pc\_fbw

Current Directory - c\_rtw x

All Files |

buildInfo.m  
drive\_Motor\_Control\_P\_xPC1  
drive\_Motor\_Control\_P\_xPC1.c  
drive\_Motor\_Control\_P\_xPC1.dll  
drive\_Motor\_Control\_P\_xPC1.exp  
modelsources.txt  
Motor\_Control\_P\_xPC1.h  
Motor\_Control\_P\_xPC1.c  
Motor\_Control\_P\_xPC1.dll  
Motor\_Control\_P\_xPC1.exp

Current Directory | Workspace

Command History x

```

xpctargetcopy
--> 6/12/07 1:29 PM
@ <-- 6/12/07 2:39 PM
-est'C:\MATLAB\R2007a
clic
xpcexplorer
xpctask
help xpc
xPCGAUGEDEMO
xpctaskpanel
xpctask
@ <-- 6/14/07 10:20 AM
xpcexplorer
xpctargetcopy
--> 6/14/07 1:20 PM
xpcexplorer
xpcexplorer
clic
@ <-- 6/14/07 2:07 PM
xpcexplorer
xpcexplorer
clic

```

Start

Microsoft (R) Program Maintenance Utility Version 6.00.6160.0  
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

```

### Compiling Motor_Control_P_xPC1.c
cl -Oc /CO /nologo -DMODEL=Motor_Control_P_xPC1 -DRT -DNUMST=2 -DTID01EQ=1 -DNCSTATES=2
Motor_Control_P_xPC1.c
### Compiling Motor_Control_P_xPC1_capi.c
cl -Oc /CO /nologo -DMODEL=Motor_Control_P_xPC1 -DRT -DNUMST=2 -DTID01EQ=1 -DNCSTATES=2
Motor_Control_P_xPC1_capi.c
### Compiling Motor_Control_P_xPC1_data.c
cl -Oc /CO /nologo -DMODEL=Motor_Control_P_xPC1 -DRT -DNUMST=2 -DTID01EQ=1 -DNCSTATES=2
Motor_Control_P_xPC1_data.c
### Compiling drive_Motor_Control_P_xPC1_1.c
cl -Oc /CO /nologo -DMODEL=Motor_Control_P_xPC1 -DRT -DNUMST=2 -DTID01EQ=1 -DNCSTATES=2
drive_Motor_Control_P_xPC1_1.c
### Compiling rt_nonfinite.c
cl -Oc /CO /nologo -DMODEL=Motor_Control_P_xPC1 -DRT -DNUMST=2 -DTID01EQ=1 -DNCSTATES=2
rt_nonfinite.c
### Compiling xpctarget.c
cl -Oc /CO /nologo -DMODEL=Motor_Control_P_xPC1 -DRT -DNUMST=2 -DTID01EQ=1 -DNCSTATES=2
xpctarget.c
### Linking ...
C:\MATLAB\R2007a\sys\perl\win32\bin\perl C:\MATLAB\R2007a\rtw\c\tcc\mkvc\mkvc_ink.pl Motor_Contr
link /NOLOGO /DLL /SUBSYSTEM:CONSOLE /DEF:xpcvedll.def /Include:_malloc C:\MATLAB\R2007a\to
Creating library Motor_Control_P_xPC1.lib and object Motor_Control_P_xPC1_exp
### Created DLL Motor_Control_P_xPC1.dll
C:\MATLAB\R2007a\rtw\c\tcc\mkvc\mkurdml -c+ -os -as -a= Motor_Control_P_xPC1.dll
RTTtarget-32 DLM Processor 3.05 (c) 1996,2000 On Time Informatik GmbH
### Created DLM ..\Motor_Control_P_xPC1.dlm
C:\MATLAB\R2007a\work\Introductory_MBD_Course\Matlab_F...
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

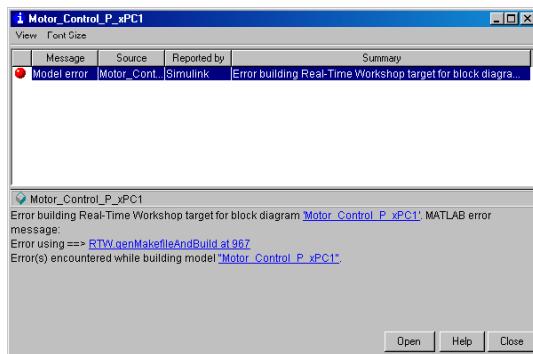
```

If the build is successful,  
the model will be  
downloaded to the target  
PC.

MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY

## Watcom Compiler Bug Fix

- If you are using the Watcom compiler, you will receive the error below:



- Click the Close button.





## Watcom Compiler Bug Fix

55

- You will need to compile and download the model to the xPC target manually.
- In the Matlab window change to directory Motor\_Control\_P\_xPC1\_xpc\_rtw and edit the file named Motor\_Control\_P\_xPC1.bat
- To edit the file, right click on the file name Motor\_Control\_P\_xPC1.bat and then select **Open** :

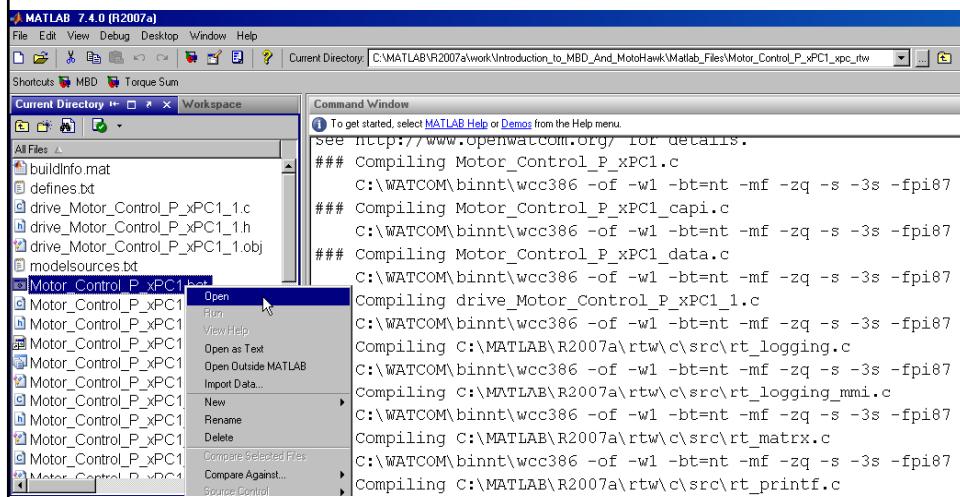


The MathWorks



## Watcom Compiler Bug Fix

56



The MathWorks





57

## Watcom Compiler Bug Fix

- A text editor should run and display the text file:

```

1 set WATCOM=C:\WATCOM
2 cd .
3 "%WATCOM%\binnt\wmake" -f Motor_Control_P_xPC1.mk GENERATE_REPORT
4 @if errorlevel 1 goto error_exit
5 exit /B 0
6
7 :error_exit
8 echo The make command returned an error
9 An_error_occurred_during_the_call_to_ma
10

```

Go to the end of this line  
and add the text  
OPT\_OPTS=



58

## Watcom Compiler Bug Fix

```

1
2
3 _TO_ANCHOR=... MODELREF_TARGET_TYPE=NONE WATCOM_VER=1.3 OPT_OPTS=
4
5
6
7
8
9
10

```

Text added here.

- Save the changes and close the batch file.





59

## Watcom Compiler Bug Fix

- Next, we need to run the batch file by entering the command **!Motor\_Control\_P\_xPC1.bat** at the Matlab command prompt.

The screenshot shows the MATLAB interface. The Command Window displays the command `>> !Motor_Control_P_xPC1.bat` and its execution results, which include the path to the batch file and a copyright notice from Sybase. The File browser (Workspace browser) on the left shows various files related to the project, including source code files like `drive_Motor_Control_P_xPC1_1.c`, header files like `drive_Motor_Control_P_xPC1_1.h`, and build-related files like `buildinfo.mat` and `defines.txt`.

```

MATLAB 7.4.0 (R2007a)
File Edit Debug Desktop Window Help
Current Directory: C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\Motor_Control_P_xPC1_xpc1_tw
Shortcuts MBD Torque Sum
Current Directory Workspace
All Files
buildinfo.mat
defines.txt
drive_Motor_Control_P_xPC1_1.c
drive_Motor_Control_P_xPC1_1.h
drive_Motor_Control_P_xPC1_1.obj
modelsources.txt
Motor_Control_P_xPC1.bat
Motor_Control_P_xPC1.c
Motor_Control_P_xPC1.h
Motor_Control_P_xPC1.mk
Motor_Control_P_xPC1.obj

Command Window
To get started, select MATLAB Help or Demo from the Help menu.
Operable program or batch file.
>> !Motor_Control_P_xPC1.bat

C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\Mot
C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\Mot
C:\MATLAB\R2007a\work\Introduction_to_MBD_And_MotoHawk\Matlab_Files\Mot
Open Watcom Make Version 1.3
Portions Copyright (c) 1988-2002 Sybase, Inc. All Rights Reserved.
Source code is available under the Sybase Open Watcom Public License.

```

60

## Watcom Compiler Bug Fix

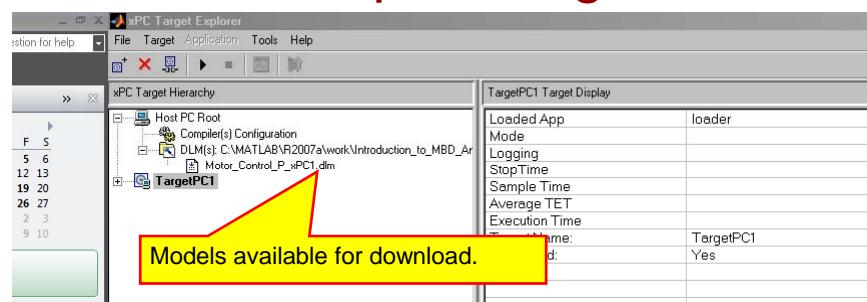
- When the compilation process is complete, you will need to download the model to your xPC target computer using the xPC Explorer.
- Change back to your working directory.
- Open up the windows explorer (if it is not already open) by typing **xpcexplr** at the Matlab command prompt.
- Connect to your xPC target.
- The available xPC models should be listed in the xPC Explorer window.





## Watcom Compiler Bug Fix

61



- To download the model to your xPC target computer, right-click on the text Motor\_Control\_P\_xPC1.dlm and select **Download**:

**MotoTron**

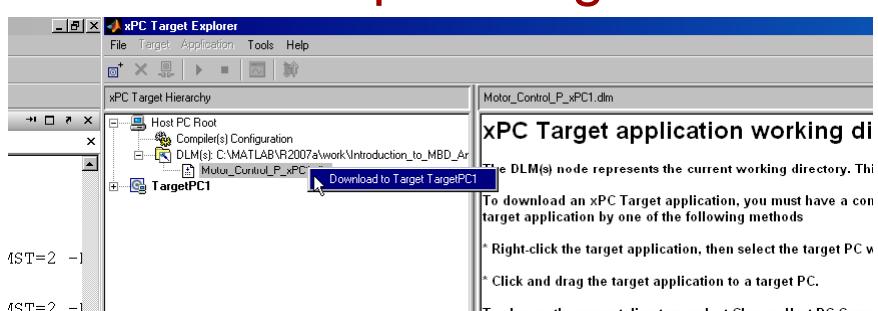
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Watcom Compiler Bug Fix

62



- The model should download and you should be at the same point as if the build process did not encounter an error.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

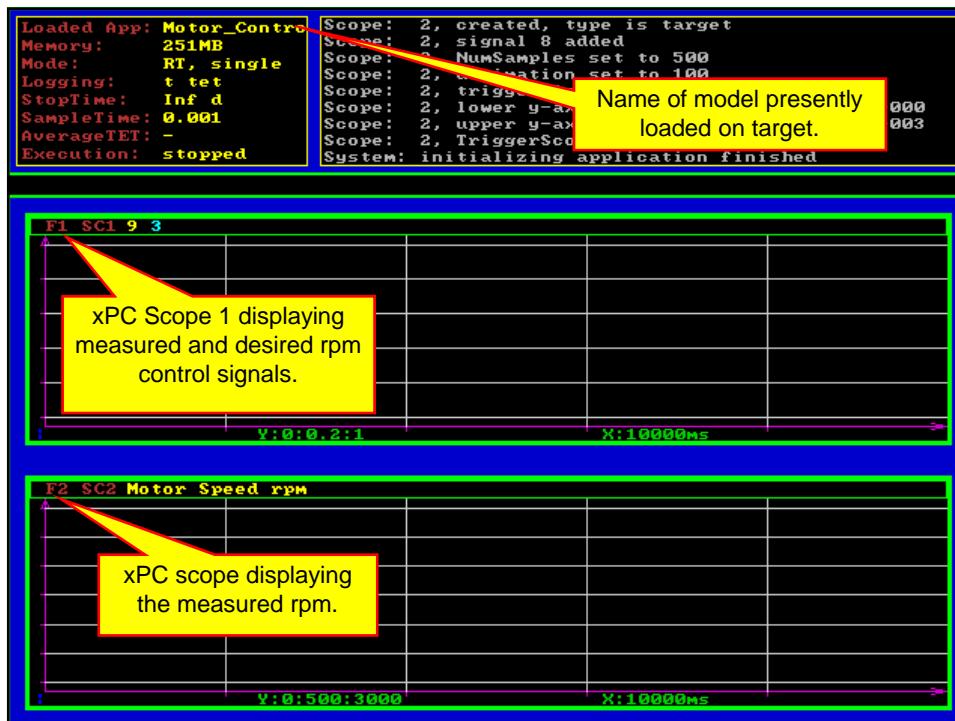
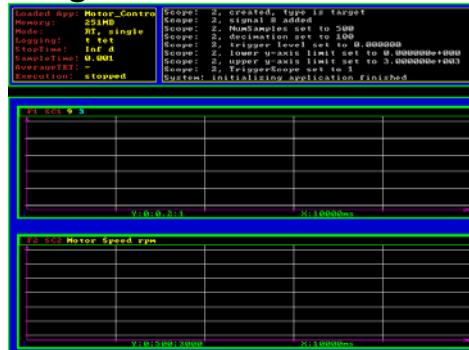
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

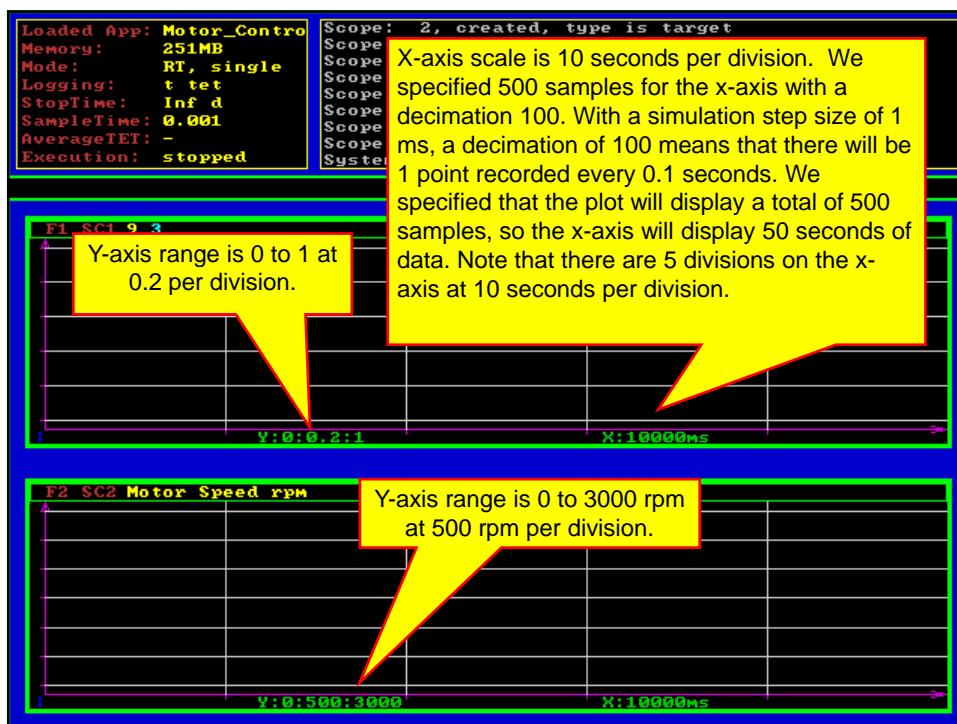


63

## Model Execution

- If you were successful, your model should be downloaded to your target PC.
- If you have a monitor on your PC, you should see the following screen.



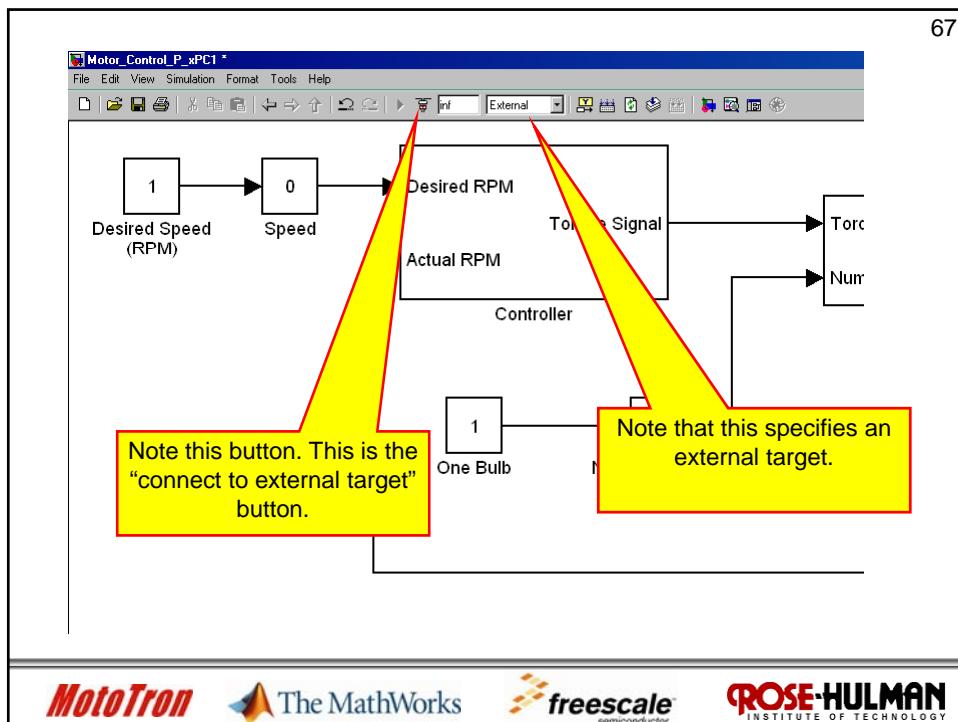


## Model Execution

66

- We are now ready to run the model on our target PC.
- We must specify that the model should run on a an external target.
- Select **Simulation** and then **External** from the Simulink Menus.
- The tool bar should show that you have selected an external target.





## Model Execution

68

- Note that a new button has appeared.
- This button means connect to external target.
- Next, we must connect to the external target.
- Click on the button, or select **Simulation** and then **Connect to Target** from the Simulink menus.





69

## Model Execution

- You might ask, how does Matlab know which target to connect to.
- In the Simulation Parameters setup, on the xPC tab we specified to automatically download the model after building, and we specified that it should use the default target.
- In the xPC Explorer, we specified TargetPC1 as the default target.
- Thus, our model will run on TargetPC1.

**MotoTron**

**The MathWorks**

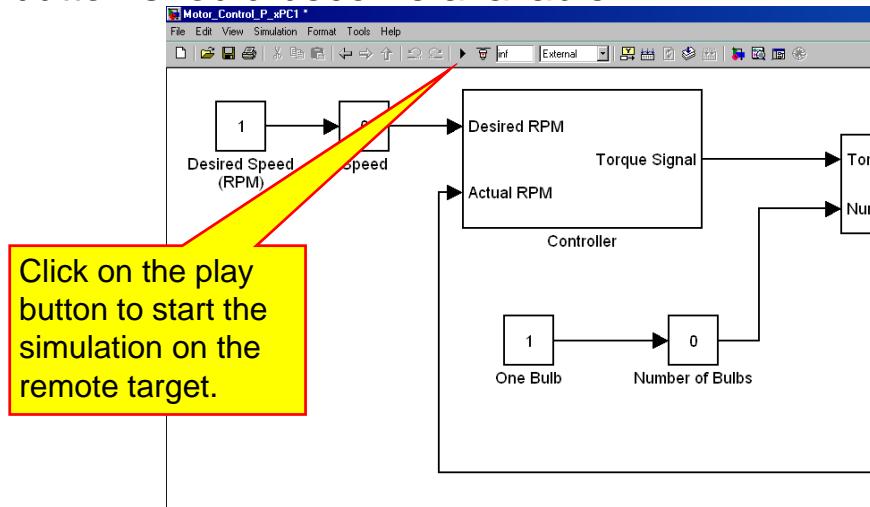
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

70

## Model Execution

- After connecting to the target, the play ▶ button should become available:





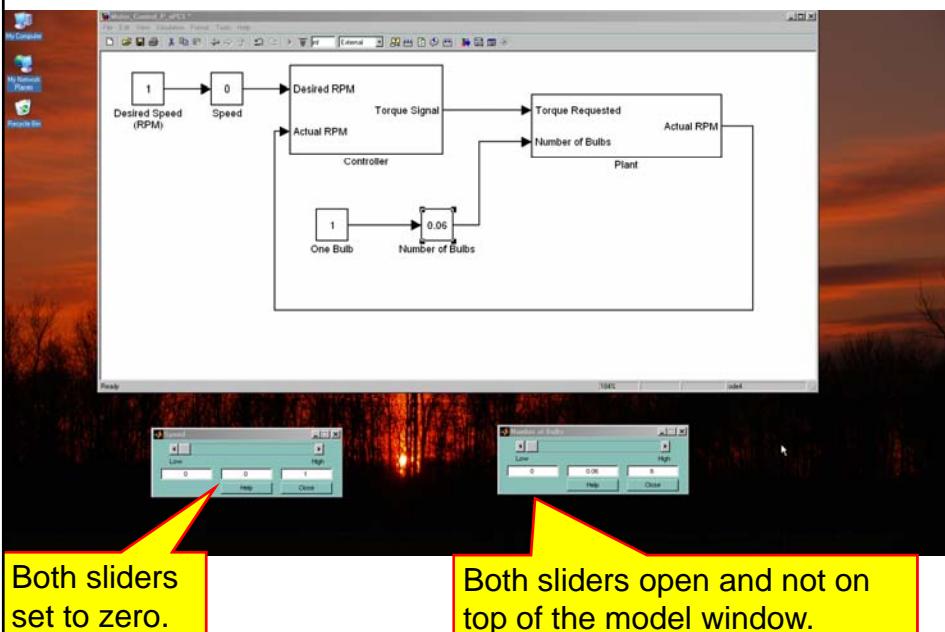
71

## Model Execution

- Before we start the model, we will open the sliders and set both to zero.
- This will make all of our simulations start at the same point.
- You may want to arrange your screen as shown on the next slide to make it easy to change the inputs to the model.

72

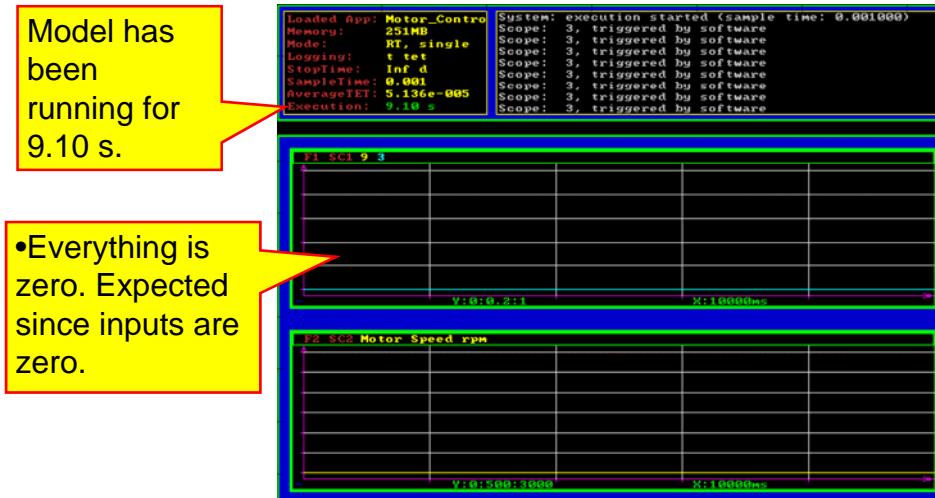
## Model Execution



## Model Execution

73

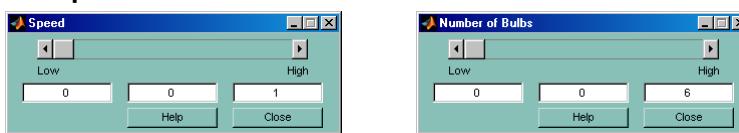
- When you click the play button, you should see the screen below:



## Model Execution

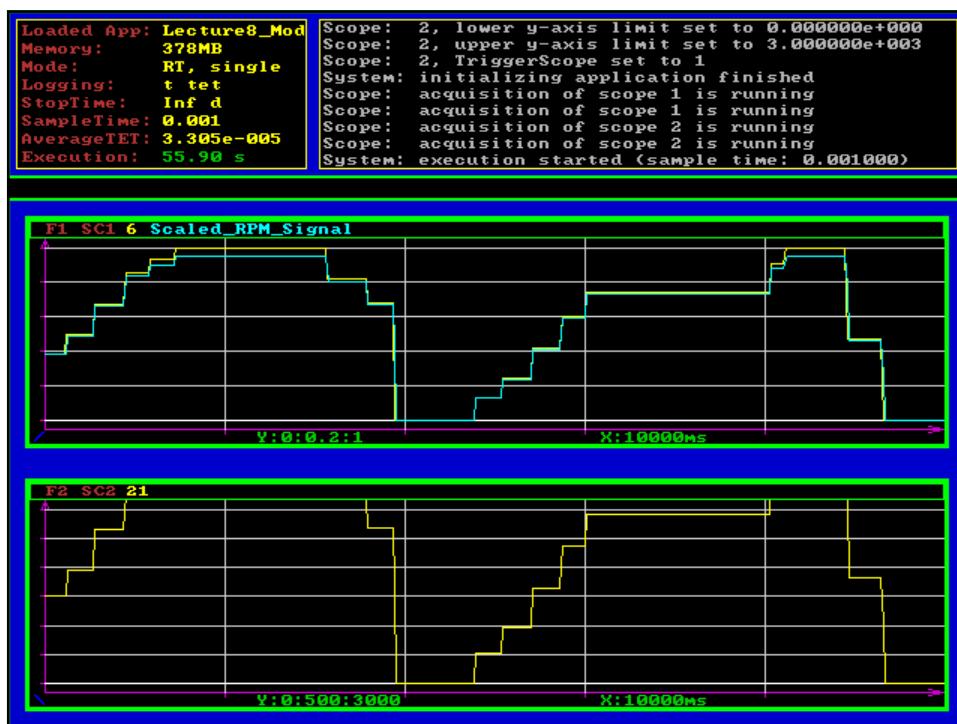
74

- Change the speed slider and observe the operation of the model.
  - Keep the number of light bulbs at zero (no load).
  - The next few slides will show the operation of my model as the speed slider changes.
  - Note that the desired speed is the value of the speed slider.
  - Proportional Gain set to 10.





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



## Lecture 8 Demo 1

76

- Initial xPC real-time system response.

Demo\_\_\_\_\_





77

## Model Execution

- In the previous slide:
  - In the top trace, the desired speed (the value of the slider) is shown in yellow.
  - We see that the actual speed follows the desired speed closely except for very high speed requests.
  - We also see that the system reacts very quickly. This is expected from our SIL simulations, and our physical understanding of the system (ultra-low inertia motors).

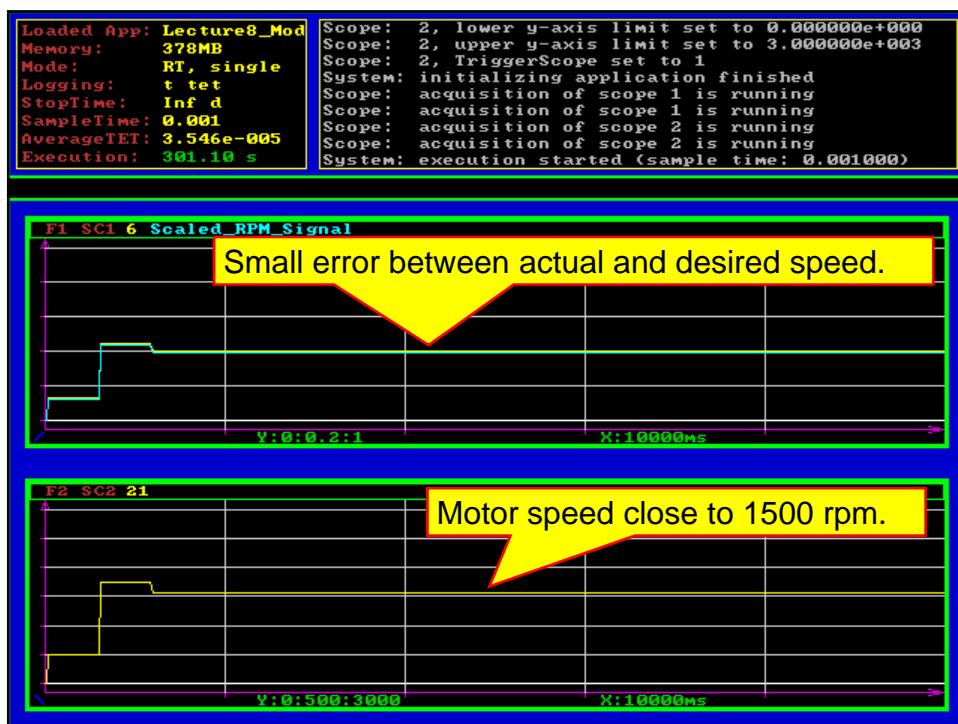


78

## Model Execution

- Next, we will show the effect of changing the load on the motor.
- Set the speed so that the motor rpm is about 1500 rpm with no load.
- On my model, this corresponds to:
  - Number of Light Bulbs = 0.
  - Speed  $\approx 0.4$



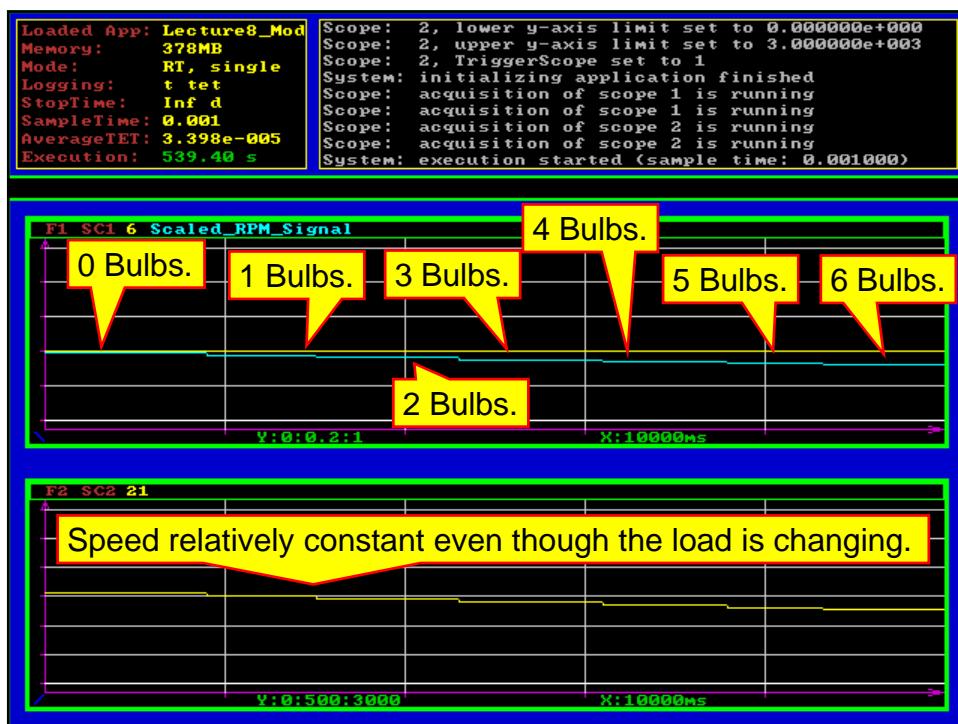


## Model Execution

80

- Next, leaving the speed constant, experiment with changing the number of light bulbs.
- Note that the speed is held relatively constant until the load becomes too large for the motor to drive.
- When the load is too high, you will see the motor rpm drop off.
- Increase the bulb load from 0 to 6 bulbs





## Lecture 8 Demo 2

82

- Constant speed, variable load testing.

Demo\_\_\_\_\_





83

## Scope Display

- For a proportional feedback system, we know that as we change the load, the error between the desired and actual speed should increase.
- A large proportional gain will reduce the error, but the error should increase as we increase the load.
- For a bulb load of 0 to 2 bulbs, we do not see much change in the rpm and because of the scale of the plots.



The MathWorks

freescale<sup>®</sup>  
semiconductor

84

## Scope Display

- If we want a better look at what is happening to the motor speeds, we need to change the y-axes of the two plots.
- There are two ways we can do this:
  - Change the properties of the target scope blocks in the Simulink model. (This requires us to rebuild and download the model.)
  - Use the xPC Target Explorer – can change the axes in real-time



The MathWorks

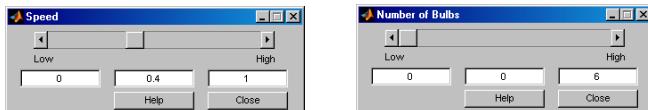
freescale<sup>®</sup>  
semiconductor



85

## Scope Display

- Set the bulb load to zero so that the rpm is close to 1500 and the actual and desired speeds are close to 0.4.



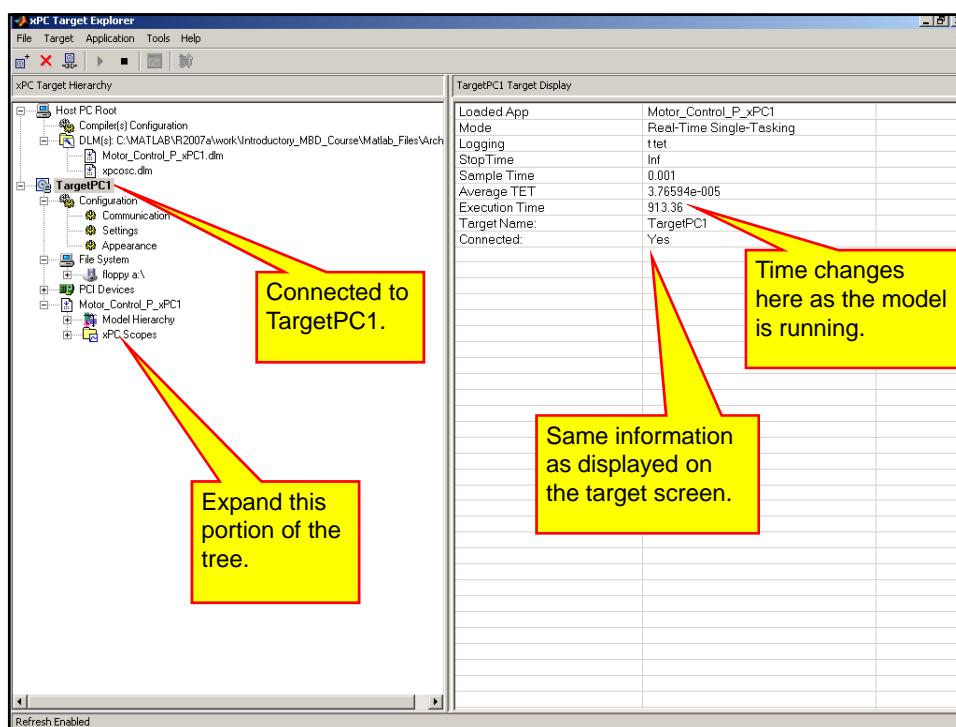
- Open the xPC Target Explorer.
- If it is not already open
  - Type `xpcexpr` at the Matlab command prompt.
  - Connect to the target named TargetPC1

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

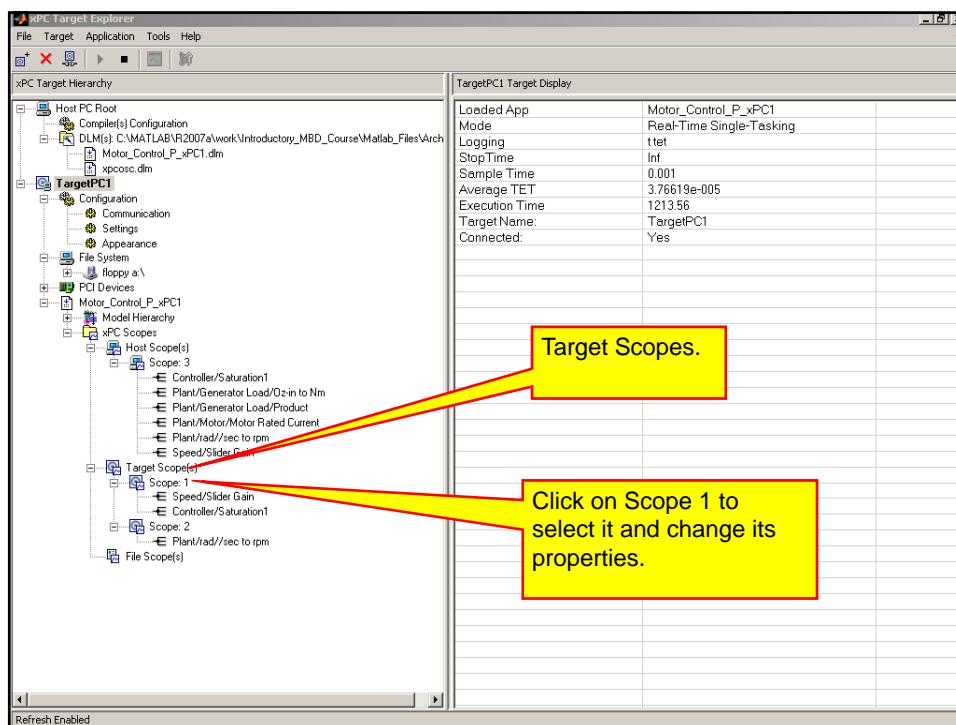


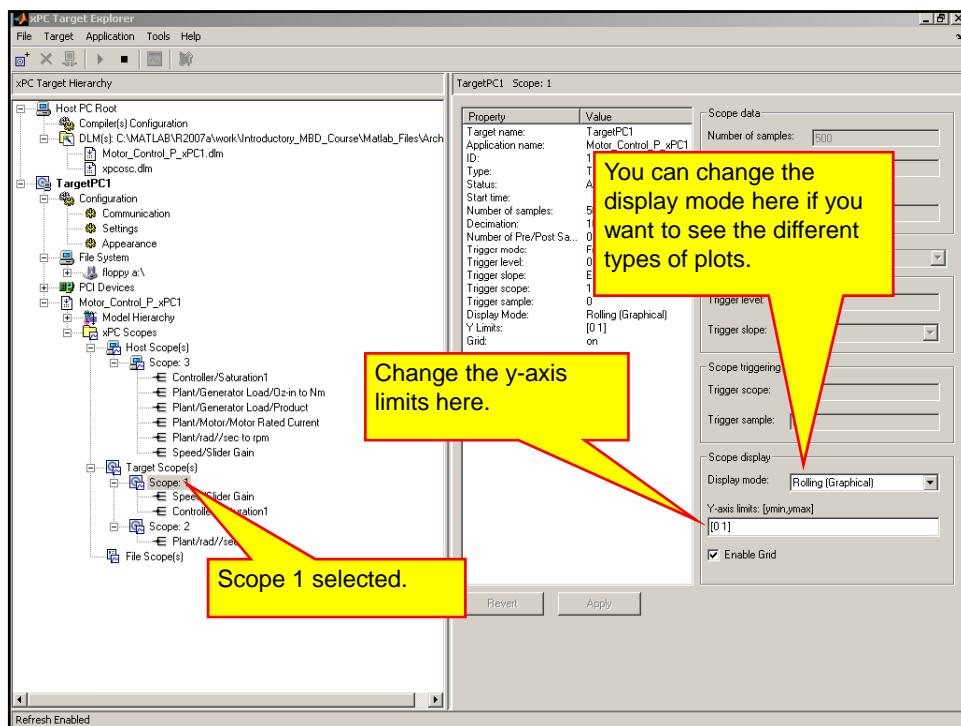


87

## Scope Display

- Expand the portion of the tree labeled **xPC Scopes**.
- Two target scopes should be shown:
  - These are the scopes displayed on the xPC Target screen.
- One Host Scope display is shown.
  - This was a Simulink scope we created with the Signal and Scope Manager.





## Scope Display

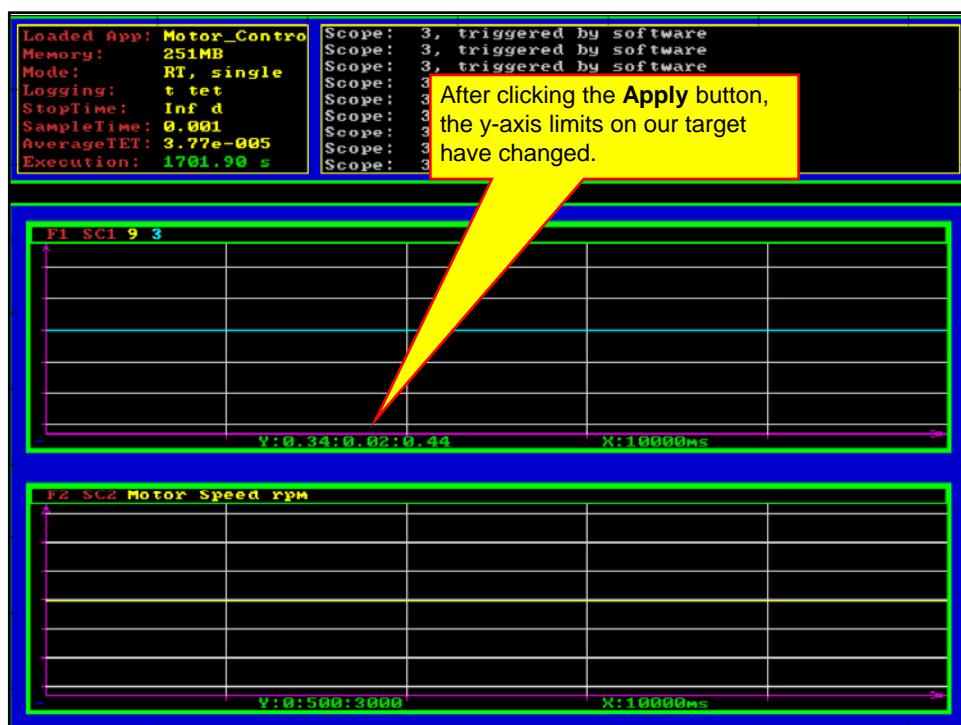
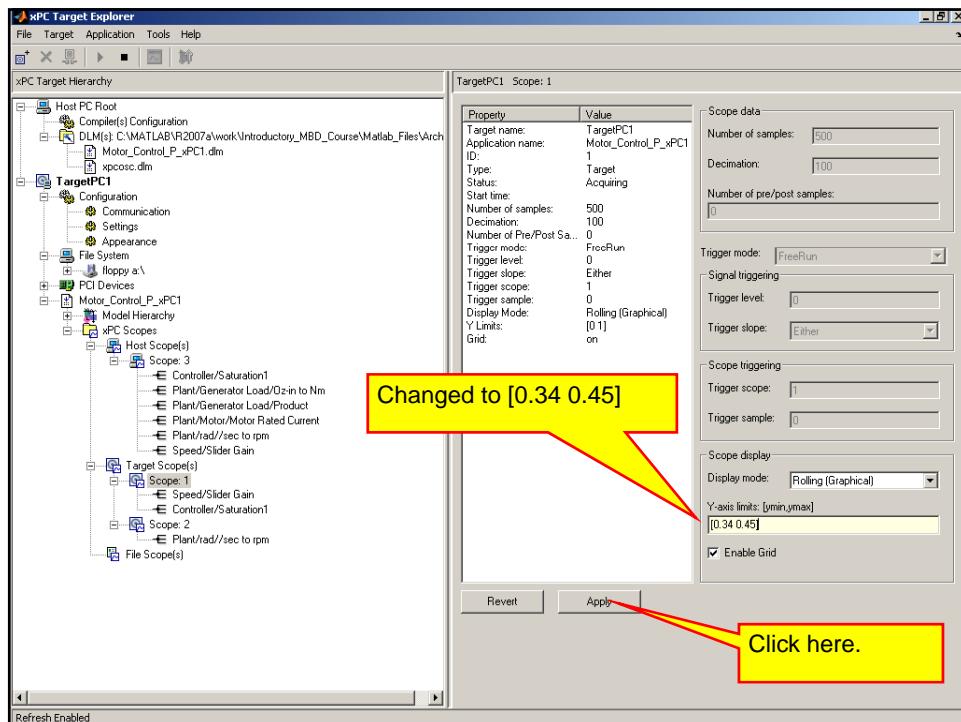
90

- For Scope 1, change the limits from [0 1] to [0.34 0.45]
- Click the **Apply** button to send the changes to TargetPC1.



The MathWorks







93

## Scope Display

- Use the same process to change the y-axis limits on Scope 2 to be from 1300 to 1700.

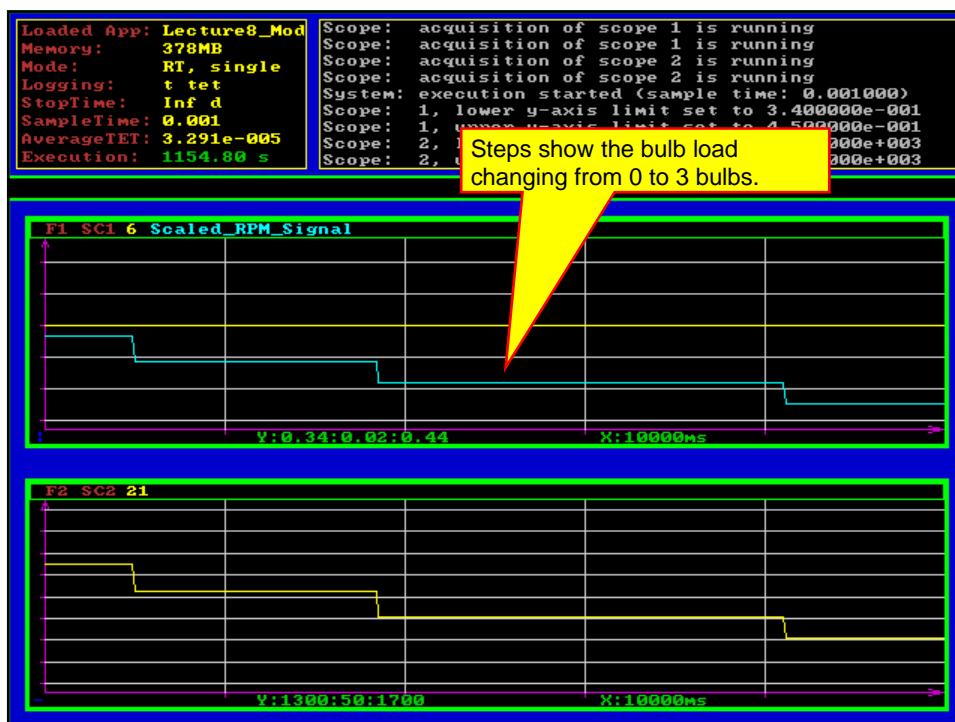


94

## Model Execution

- We can now see the performance of our controller with light loads more closely.
- We will keep the speed constant at 1500 rpm (Speed = 0.4) and vary the load from 0 to 3 light bulbs.
- The performance of my system is shown on the next slide.





## Results Summary

96

- The proportional gain controller appears to work ok with a gain of 10.
- As the load is changed from 0 to 3 bulb, the speed only changes by about 170 rpm.
- When the load becomes too high, the motor reaches its maximum power output and can no longer keep the motor speed constant. (We really need to verify this conclusion by plotting the motor torque).



The MathWorks

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



97

## Proportional Gain

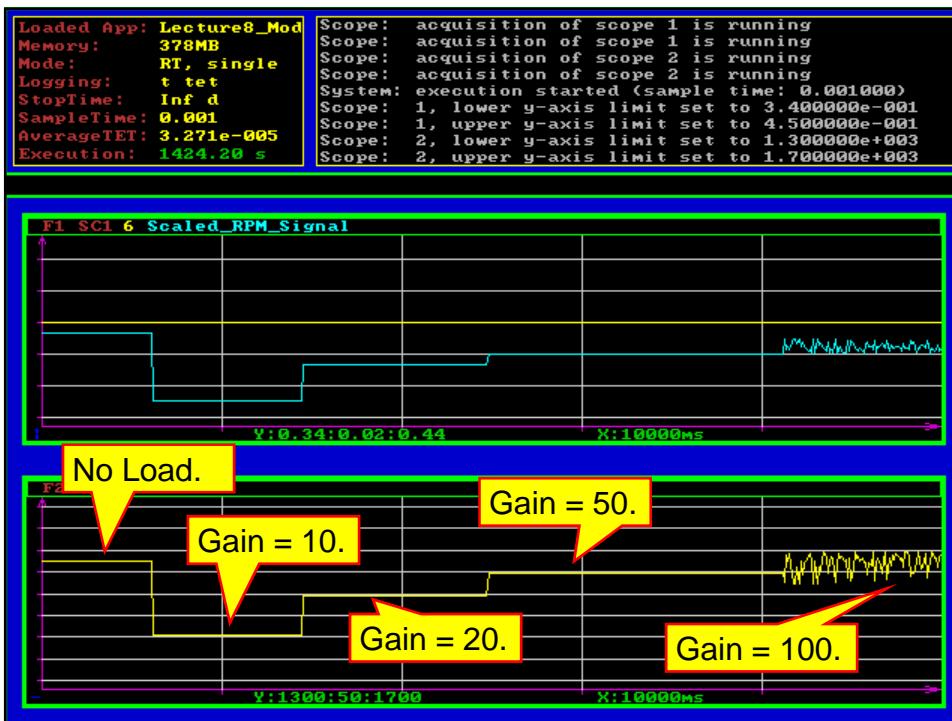
- Test Your system performance with higher Gains:
  - Proportional Gain = 20
  - Proportional Gain = 50
  - Proportional Gain = 100
- Note that you can change the gain in real-time while the simulation is running.
- The Next Slide shows the system performance for 3 bulbs at 1500 rpm and gains of 10, 20, 50, 100.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY





99

## Lecture 8 Demo 3

- System response for constant load, constant speed, variable feedback gain.

Demo\_\_\_\_\_



The MathWorks



freescale<sup>®</sup>  
semiconductor



Any Questions?



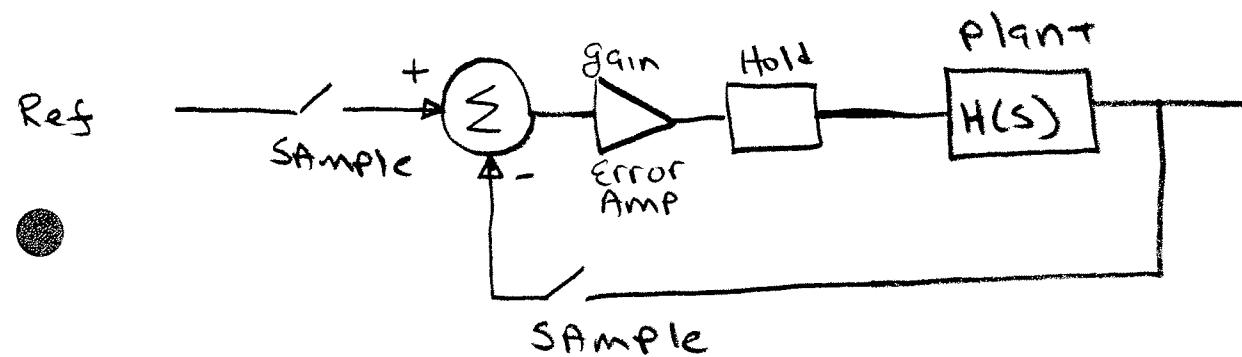
The MathWorks



freescale<sup>®</sup>  
semiconductor



- In a previous example we added a sample and hold to the output of the controller to simulate the effect of the control computer only updating its output at a periodic rate we called the controller cycle time
- The controller samples the inputs once in the cycle time
- The controller output changes once in the cycle time. In between changes, the output is held constant at its last value
- We can model our system as having a sampling circuit on the controller input and a hold on the controller output



Sampling

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

- Sampling is equivalent to multiplying an input by an impulse train

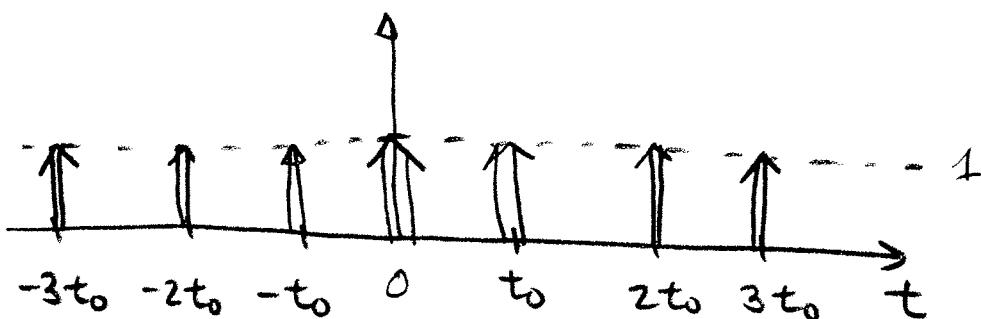
$$x(t) \xrightarrow{\quad} x^*(t)$$

$$x(t) \xrightarrow{\textcircled{X}} x^*(t)$$

$\sum_{k=-\infty}^{\infty} \delta(t - kt_0)$

- where  $t_0$  is the sampling time interval

$$\sum_{k=-\infty}^{\infty} \delta(t - kt_0)$$

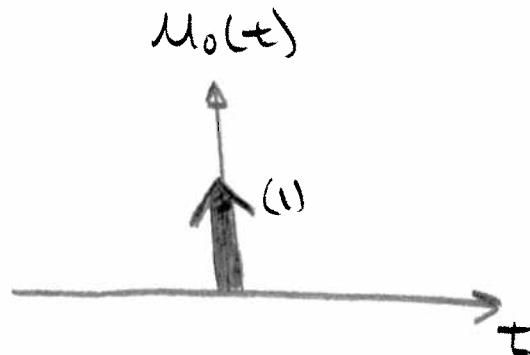
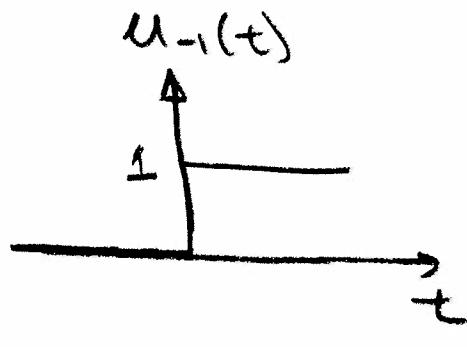


# Impulse function

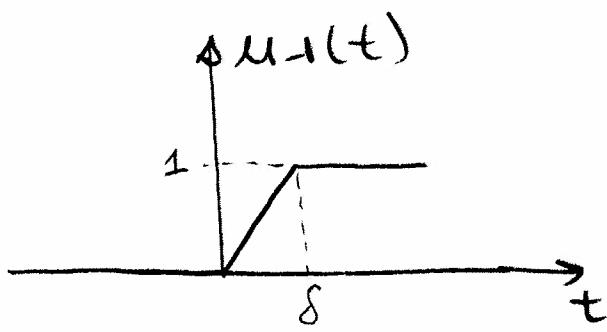


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

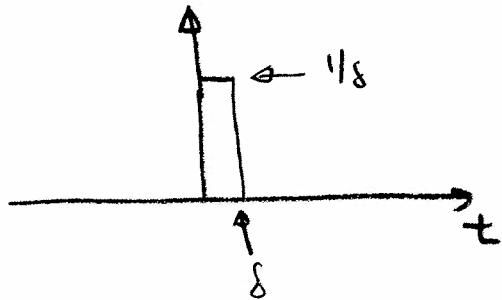
- an impulse can be thought of as the time derivative of a unit step function



Physical understanding of an impulse



$$M_0(t) = \frac{d}{dt} (u_-(t))$$



- For an Ideal impulse, we take the Limit as  $\delta \rightarrow 0$
- Notes : (1) The amplitude of an impulse is infinite.  
 (2) The area of an impulse is 1

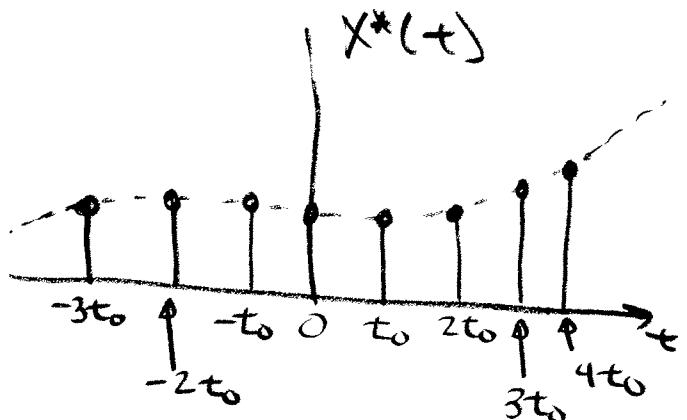
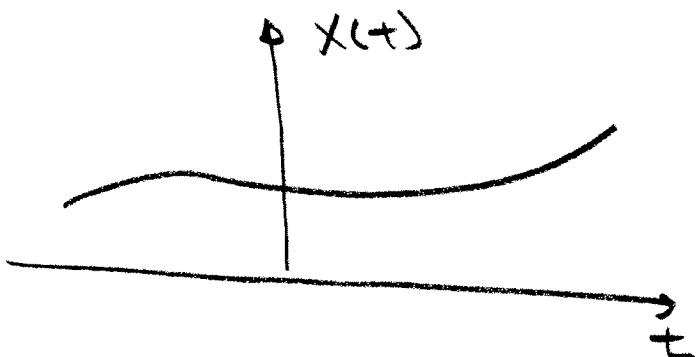
$$\int_{-\infty}^{\infty} M_0(t) dt = 1$$

**Notations in Textbooks**  $\delta(t) \equiv M_0(t)$

## Sampling

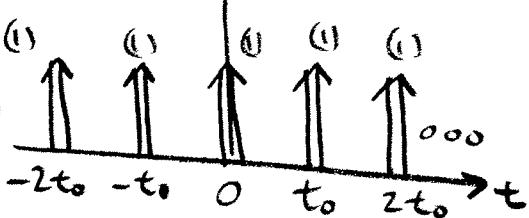
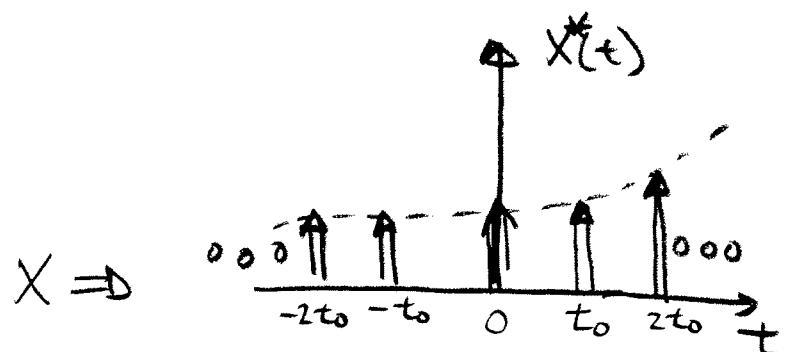
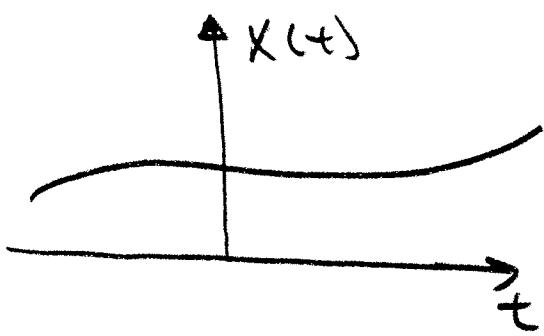
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

$$x(t) \longrightarrow x^*(t)$$



$$x(t) \xrightarrow{\text{Sampling}} x^*(t)$$

$$m(t) = \sum_{k=-\infty}^{\infty} \delta(t - kt_0)$$



$\Rightarrow$  The area of the impulses are modulated by the input signal  $x(t)$

- in order to hold the output of the sampling system to have a value between impulses, we need to integrate the impulse and then reset the integrator before integrating the next impulse.

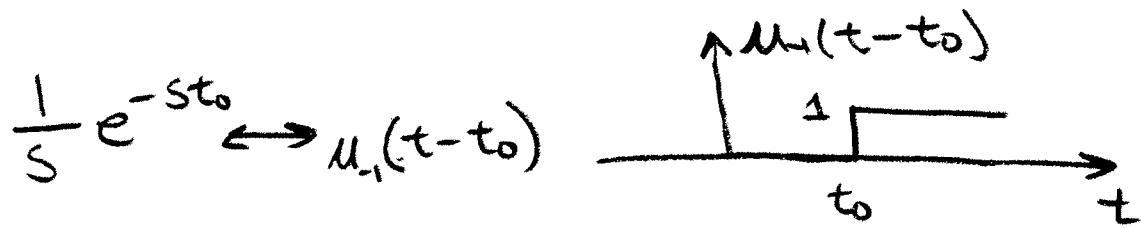
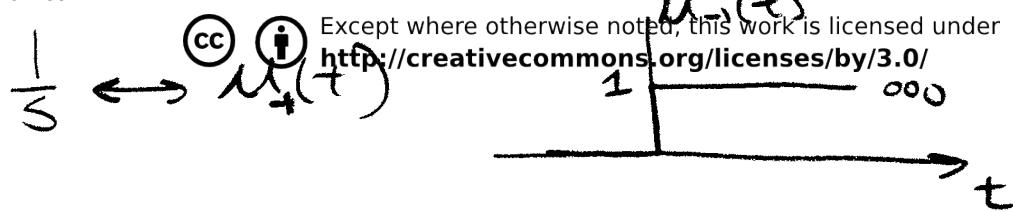
$$\text{Hold Function } G(s) = \frac{1}{s} (1 - e^{-st_0}) \\ = \frac{1}{s} - \frac{1}{s} e^{-st_0}$$

- $\frac{1}{s}$  is an integrator (or a unit step)
- $e^{-st_0}$  is a time delay

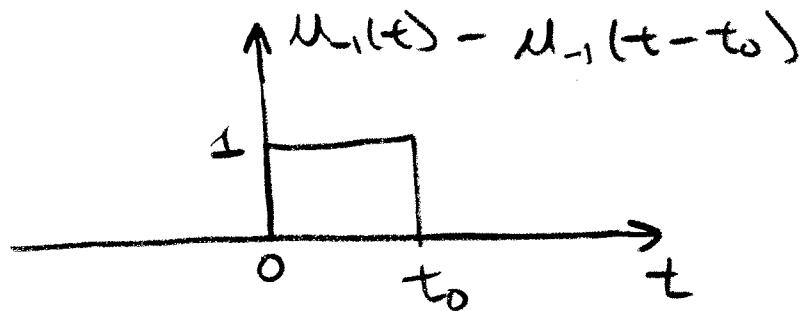
$$x(t) \Leftrightarrow X(s)$$

$$x(t-t_0) \Leftrightarrow X(s) e^{-st_0}$$

$\frac{1}{s} e^{-st_0}$  is an integrator delayed by time  $t_0$  (a unit step delayed by time  $t_0$ )



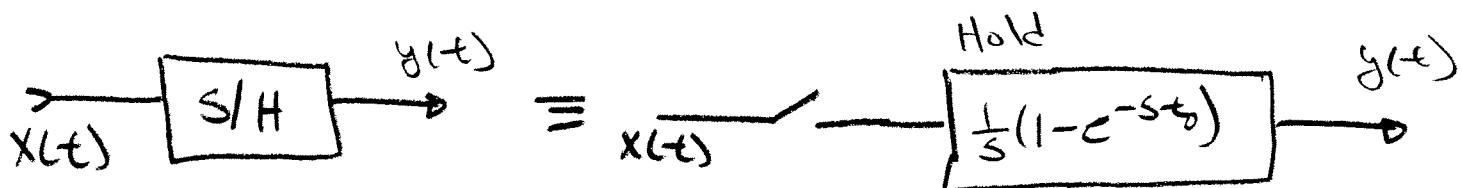
$$\frac{1}{s} - \frac{1}{s} e^{-s t_0} \leftrightarrow u_+(t) - u_+(t - t_0)$$

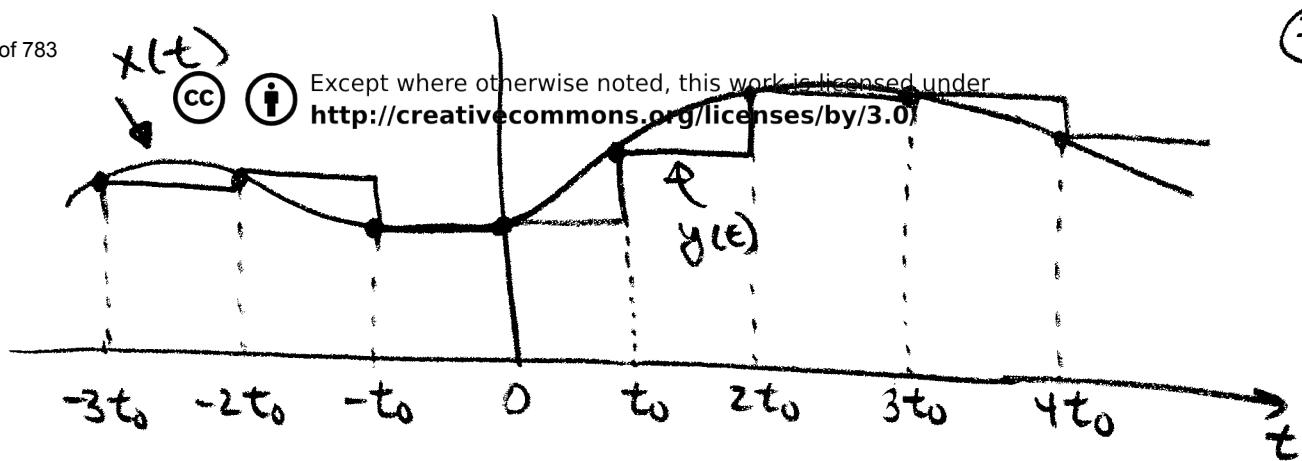


Hold  $G(s) = \frac{1}{s}(1 - e^{-s t_0})$

Sampling  $x^*(t) = x(t) m(t) = x(t) \sum_{k=-\infty}^{\infty} \delta(t - k t_0)$

Sample and hold





- we will split the sample and hold function into two separate pieces in our analysis.

- $\Rightarrow$  Sampling  $\Rightarrow$  controller input
  - $\Rightarrow$  hold  $\Rightarrow$  controller output
- 

Low frequency gain

If  $x(t) \leftrightarrow X(s)$

Then the DC gain of the system can be determined by evaluating  $X(s)$  at  $s=0$

gain of

$$\lim_{s \rightarrow 0} G(s) = \lim_{s \rightarrow 0} \frac{1 - e^{-st_0}}{s} = t_0$$

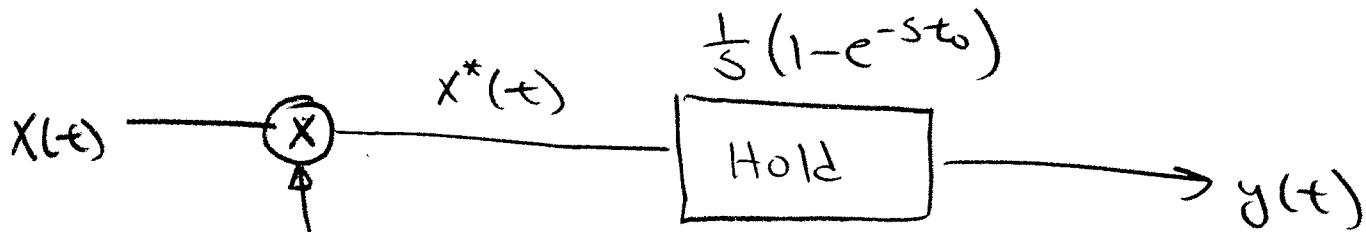
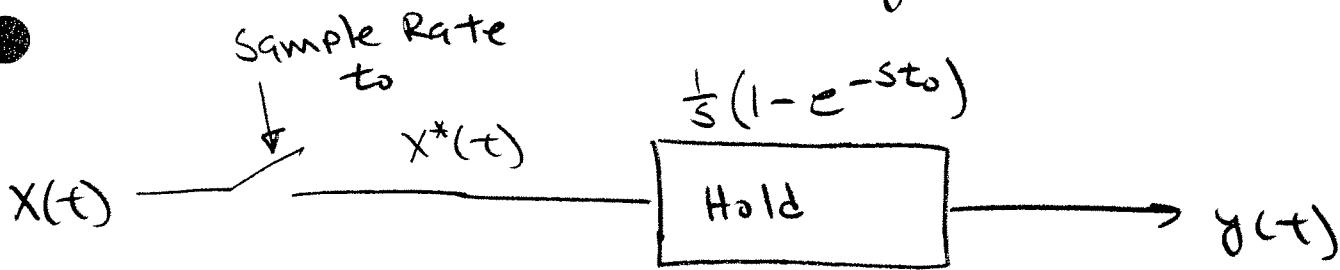
Since  $0 \leq t_0 < 1$ , for our systems, the hold by itself reduces the gain

- The Fourier transform of an impulse train is

$$\sum_{k=-\infty}^{\infty} \delta(t - kt_0) \Leftrightarrow \frac{1}{t_0} \sum_{n=-\infty}^{\infty} \delta(F - \frac{n}{t_0})$$

We can say that the DC gain of the sampling function is  $\frac{1}{t_0}$

$\Rightarrow$  The overall gain of the sample and hold is  $\left(\frac{1}{t_0}\right)t_0 = 1$

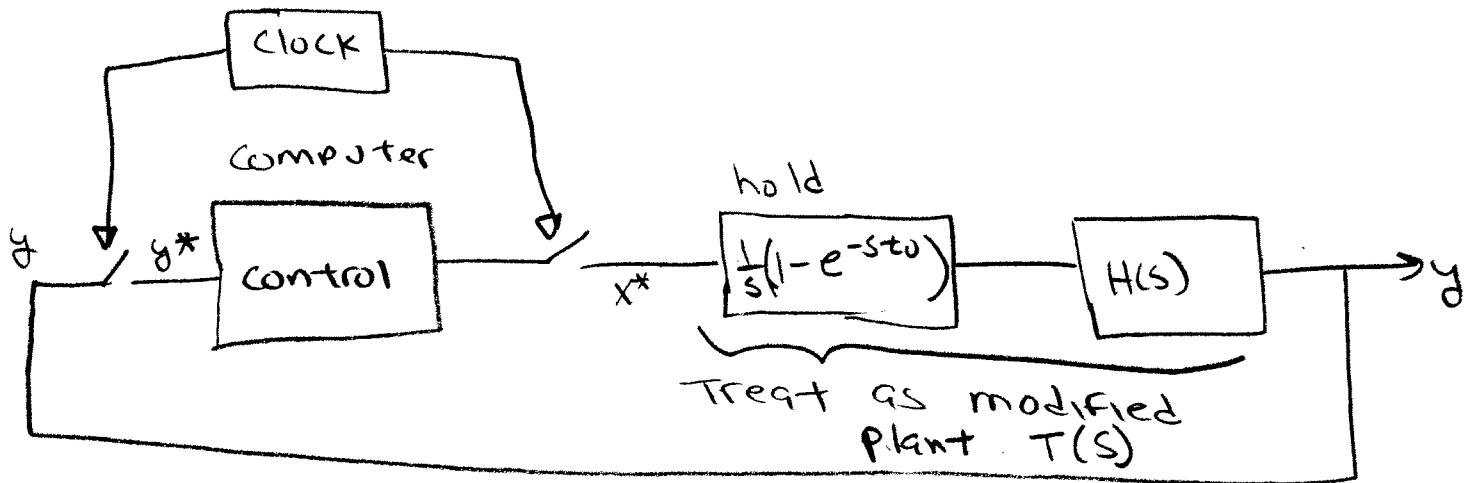
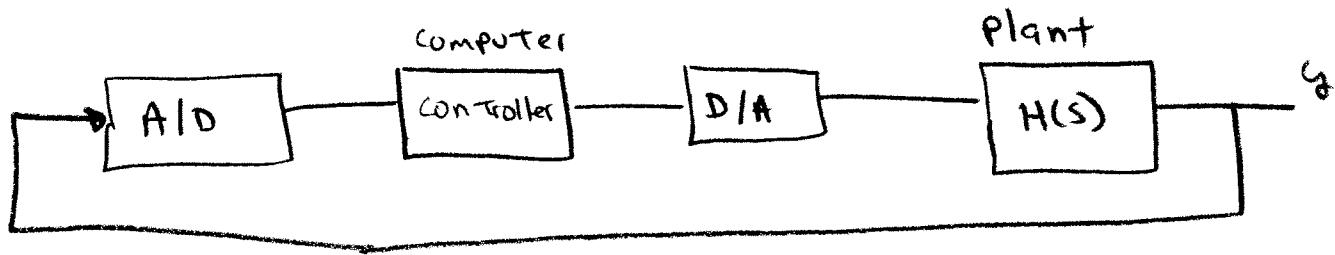


$$m(t) = \sum_{k=-\infty}^{\infty} \delta(t - kt_0)$$

$$\underbrace{\phantom{0}}_{g_{\text{ain}} = t_0}$$

$$\underbrace{\phantom{0}}_{g_{\text{ain}} = 1/t_0}$$

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

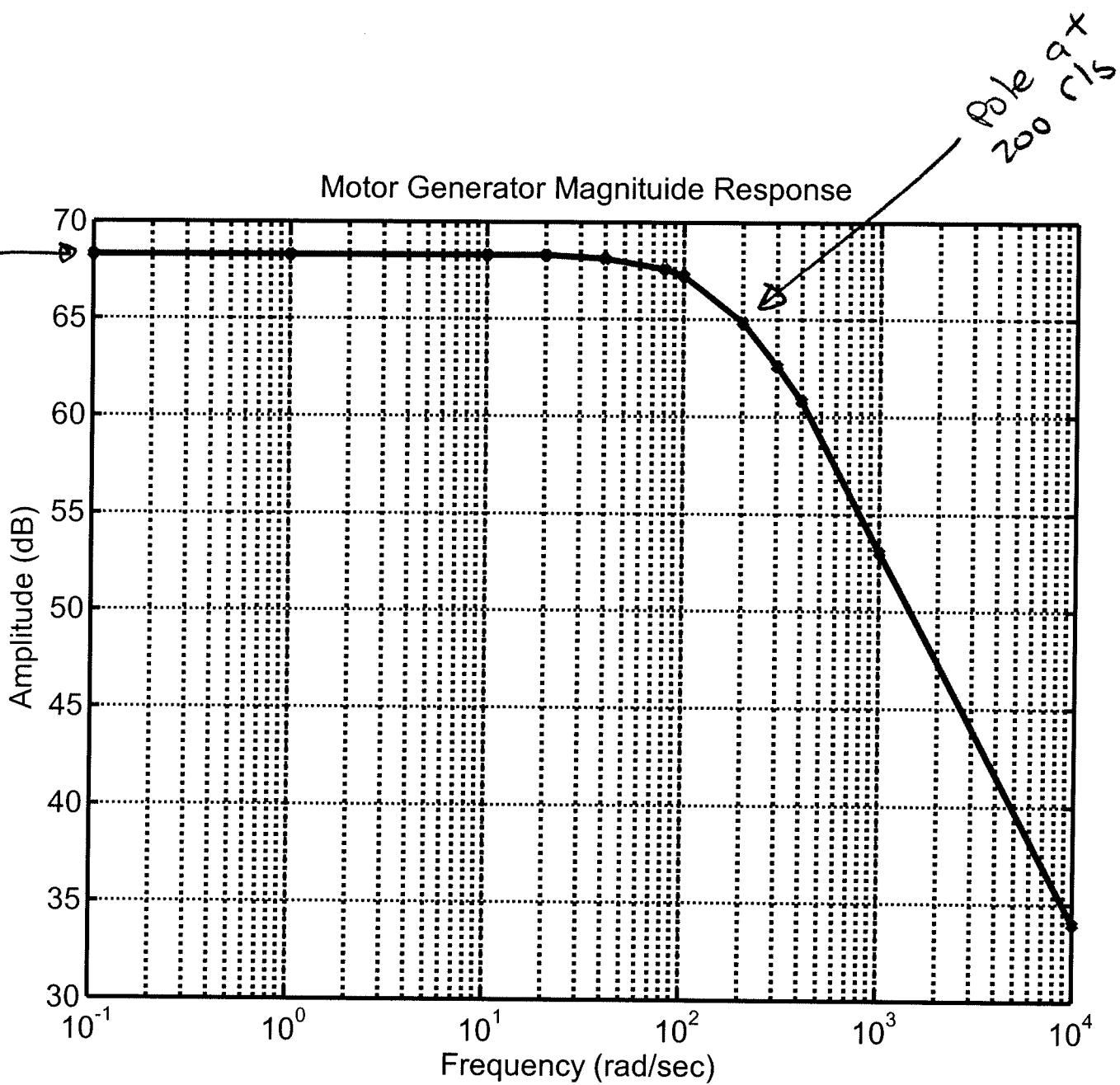


- We can now look at the magnitude and phase of the system including the sample and hold.
- Hold  $\frac{1}{s}(1 - e^{-sT_0})$  will give us a phase delay and a gain of  $T_0$ .
- We have seen that this added phase can make the system unstable with feedback.
- We will include the gain of our sampling circuit so that the overall gain of the S/H is one.

- Measured magnitude response of our motor generator system



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



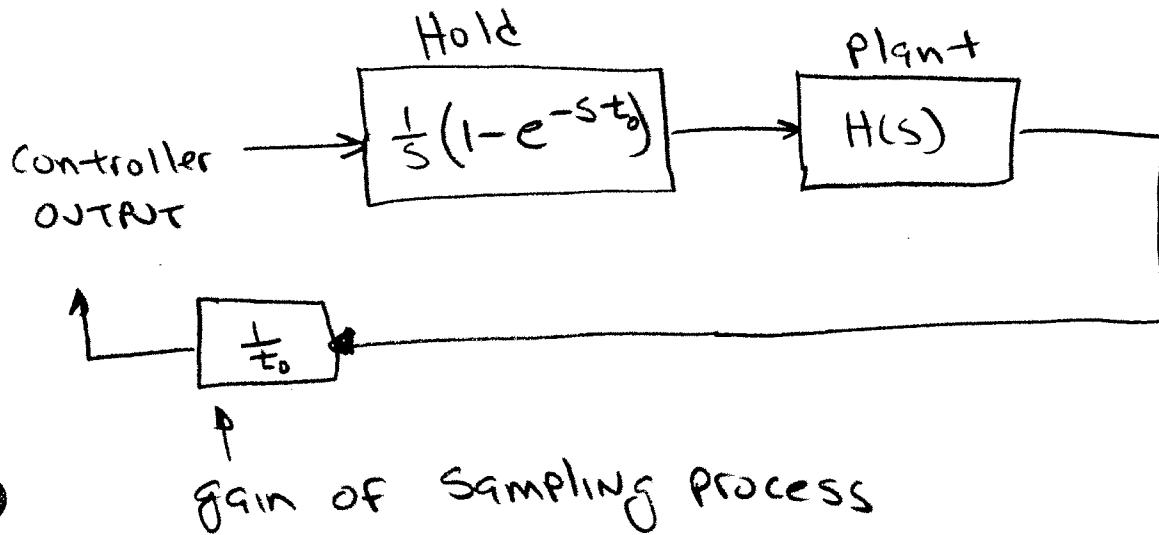
$$H(s) = \frac{2600}{1 + s/200} = \frac{2600(200)}{s + 200}$$

$$H(j\omega) = \frac{2600(200)}{j\omega + 200}$$

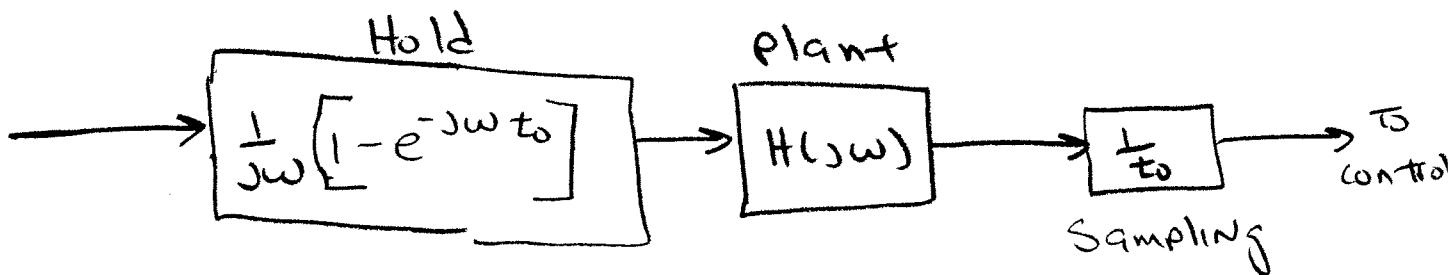
- We can  plot the magnitude

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

- and phase plot of a motor/generator system including the effects of the sample and hold.

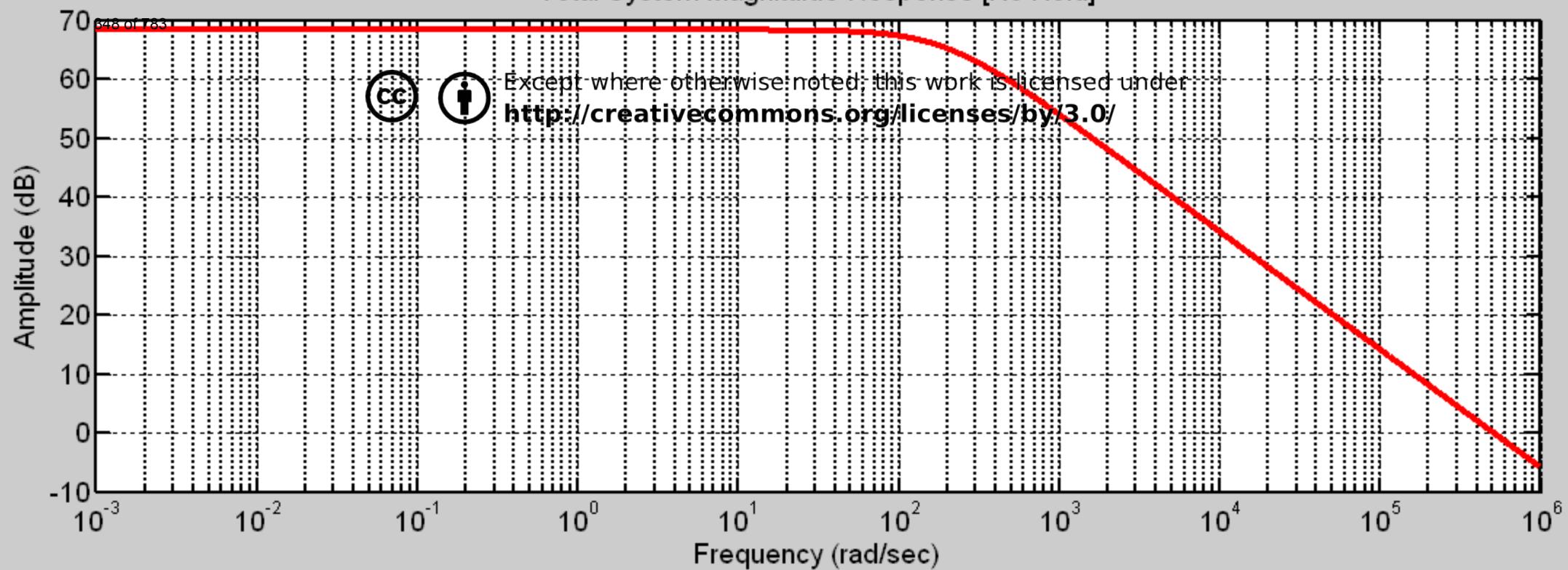


We will use Fourier transforms for everything  $\Rightarrow s = j\omega$

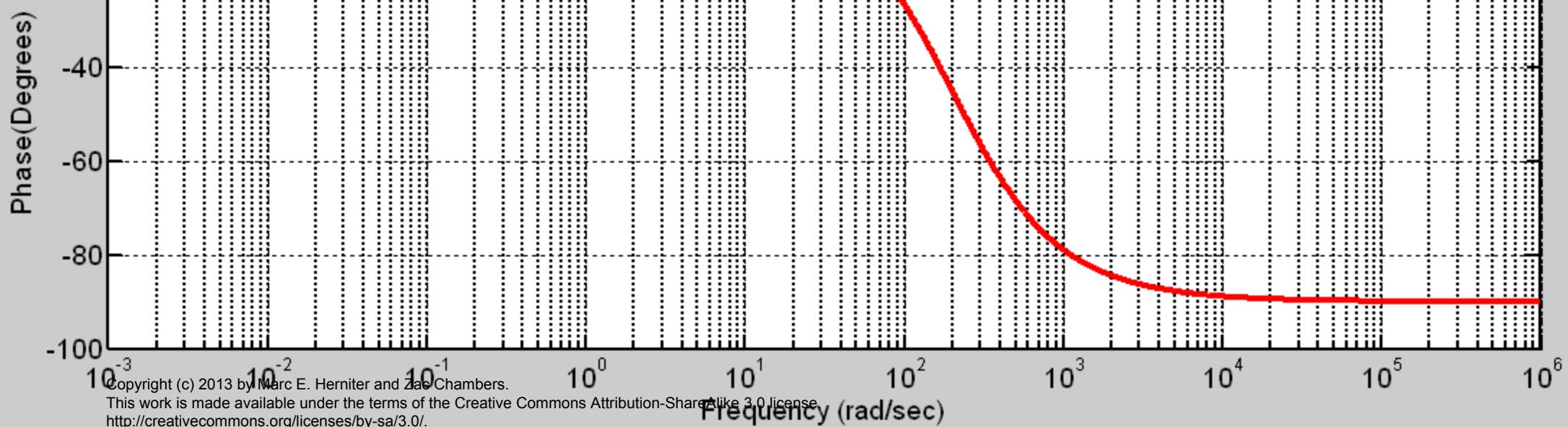


```
set(0,'defaultlinelinenwidth',3);
set(0,'defaultaxesfontname','Arial');
347 of 783
set(0,'defaultaxesfontsize',14);  Except where otherwise noted, this work is licensed under
set(0,'defaultaxeslinewidth',2);
set(0,'defaulttextfontsize',14);
set(0,'defaulttextfontname','Arial');
t0 = 0.1; %Sample time
msg=sprintf(' [No Hold]');
omega=logspace(-3,6,100000); %Generate frequencies from 10^-3 to 10^6
% Or frequencies from 0.001 rad/sec to 1,000,000 rad/sec
H=(2600.*200)./(j.*omega+200); %Evaluate the plant transfer function specified frequencies.
TF=H; %The overall transfer function.
mag=abs(TF);
dB=20*log10(mag);
phase=180*angle(TF)/pi;
subplot(2,1,1)
semilogx(omega,dB,'r-');
grid on
title(['Total System Magnitude Response',msg]);
ylabel('Amplitude (dB)');
xlabel('Frequency (rad/sec)');
subplot(2,1,2)
semilogx(omega,phase,'r-');
grid on
title(['Total System Phase Response',msg]);
ylabel('Phase(Degrees)');
xlabel('Frequency (rad/sec)');
Copyright (c) 2013 by Marc E. Herniter and Zac Chambers.
This work is made available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license,
http://creativecommons.org/licenses/by-sa/3.0/.
```

### Total System Magnitude Response [No Hold]



### Total System Phase Response [No Hold]



```
set(0,'defaultlinelinenumber',3);
set(0,'defaultaxesfontname','Arial');
set(0,'defaultaxesfontsize',14);
set(0,'defaultaxeslinewidth',2);
set(0,'defaulttextfontsize',14);
set(0,'defaulttextfontname','Arial');
t0 = 0.001; %Sample time
omega=logspace(-3,6,100000); %Generate frequencies from 10^-3 to 10^6
% Or frequencies from 0.001 rad/sec to 1,000,000 rad/sec
H=(2600.*200)./(j.*omega+200); %Evaluate the plant transfer function specified frequencies.
SH=(1/t0)*(1-exp(-j.*omega.*t0))./(j.*omega); %Evaluate the sample and hold at specified frequencies.
TF=SH.*H; %The overall transfer function.
mag=abs(TF);
dB=20*log10(mag);
phase=180*angle(TF)/pi;
subplot(2,1,1)
semilogx(omega,dB,'r-');
grid on
title('Total System Magnitude Response');
ylabel('Amplitude (dB)');
xlabel('Frequency (rad/sec)');
subplot(2,1,2)
semilogx(omega,phase,'r-');
grid on
title('Total System Phase Response');
ylabel('Phase(Degrees)');
xlabel('Frequency (rad/sec)');

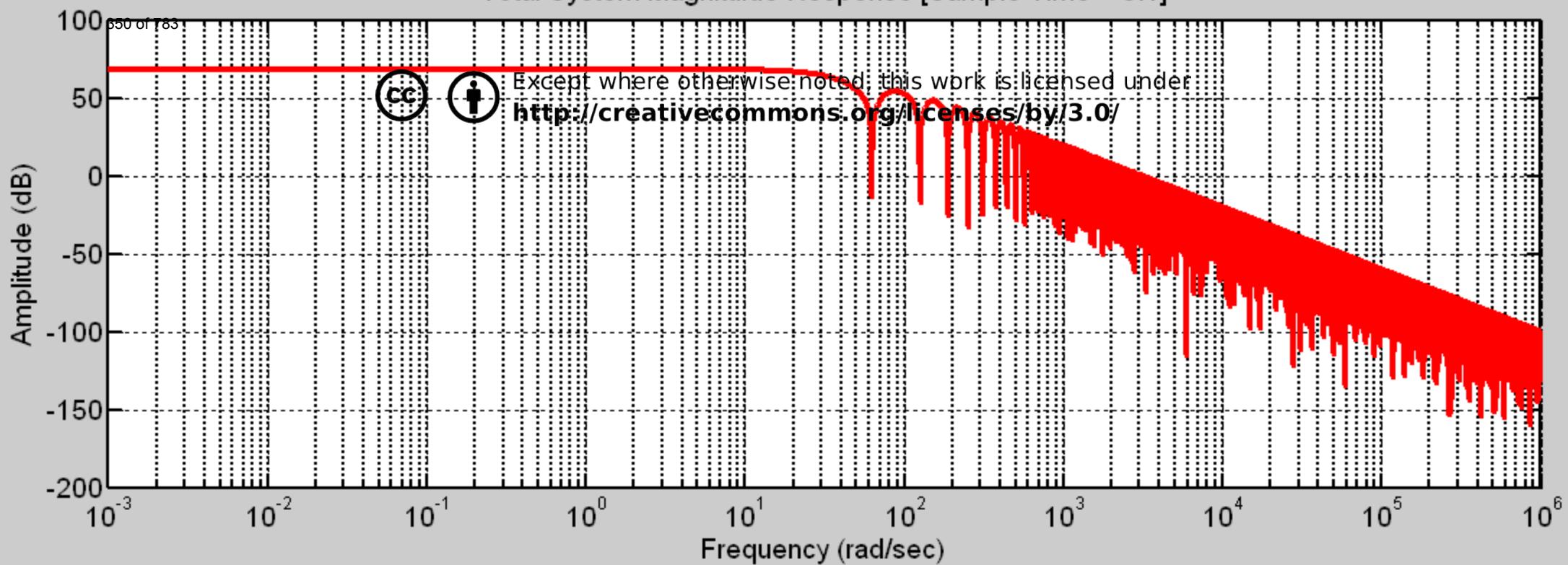
```

Copyright (c) 2013 by Marc E. Herniter and Zac Chambers.  
This work is made available under the terms of the Creative Commons Attribution-ShareAlike 3.0 license,  
<http://creativecommons.org/licenses/by-sa/3.0/>.

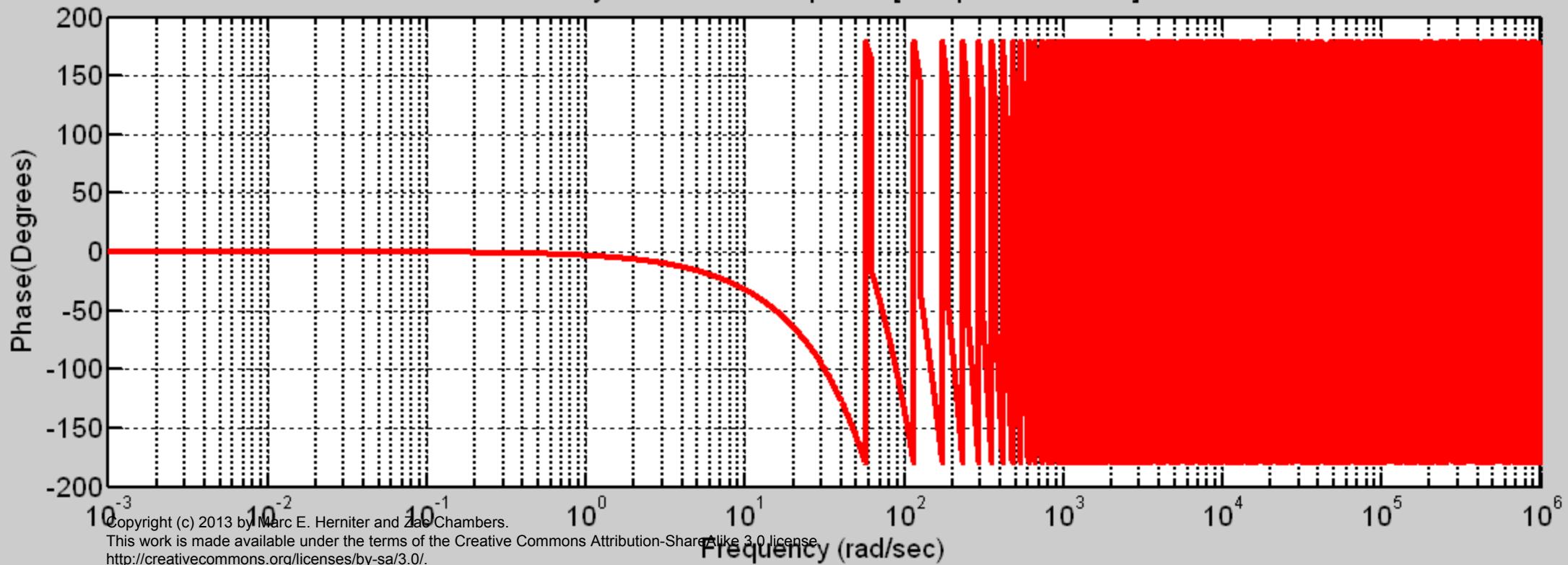


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

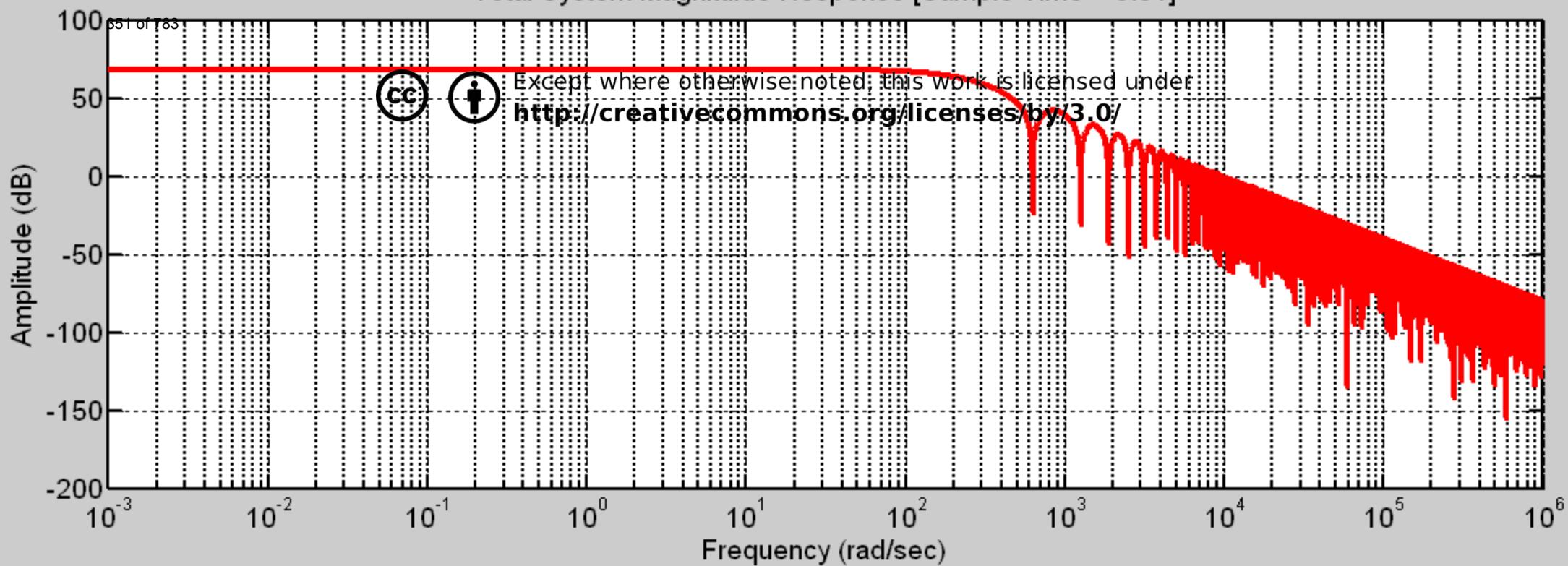
### Total System Magnitude Response [Sample Time = 0.1]



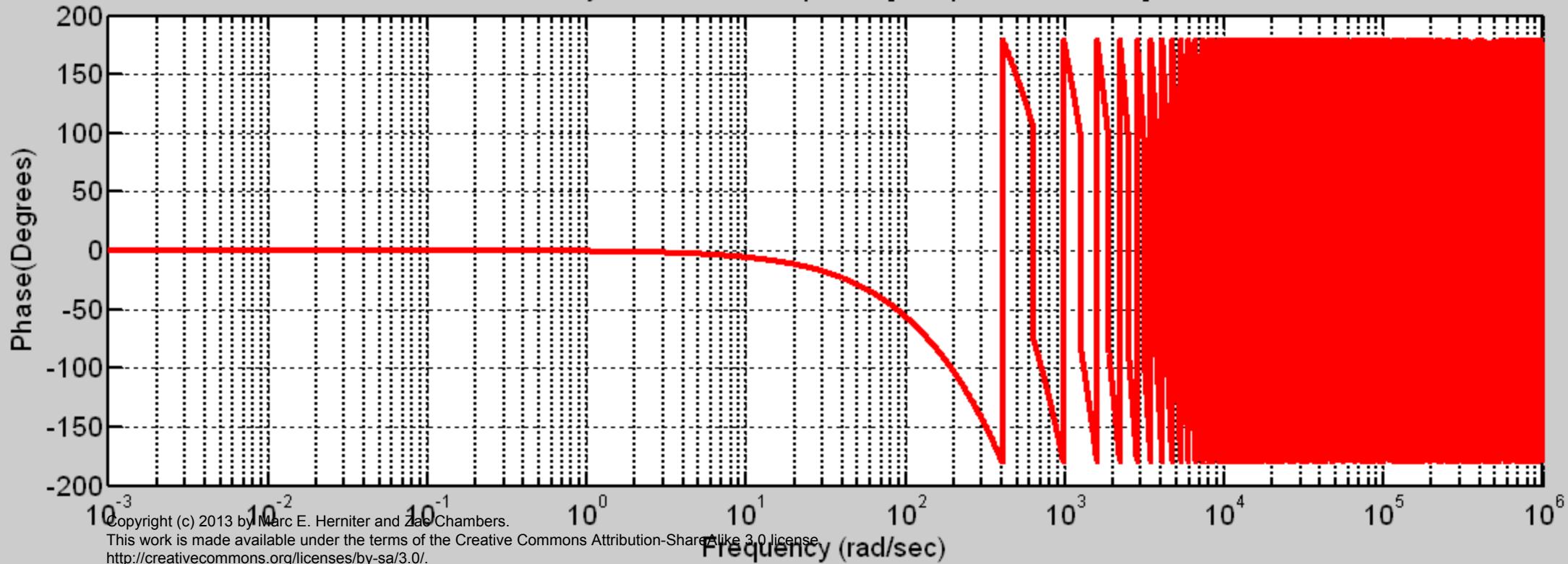
### Total System Phase Response [Sample Time = 0.1]



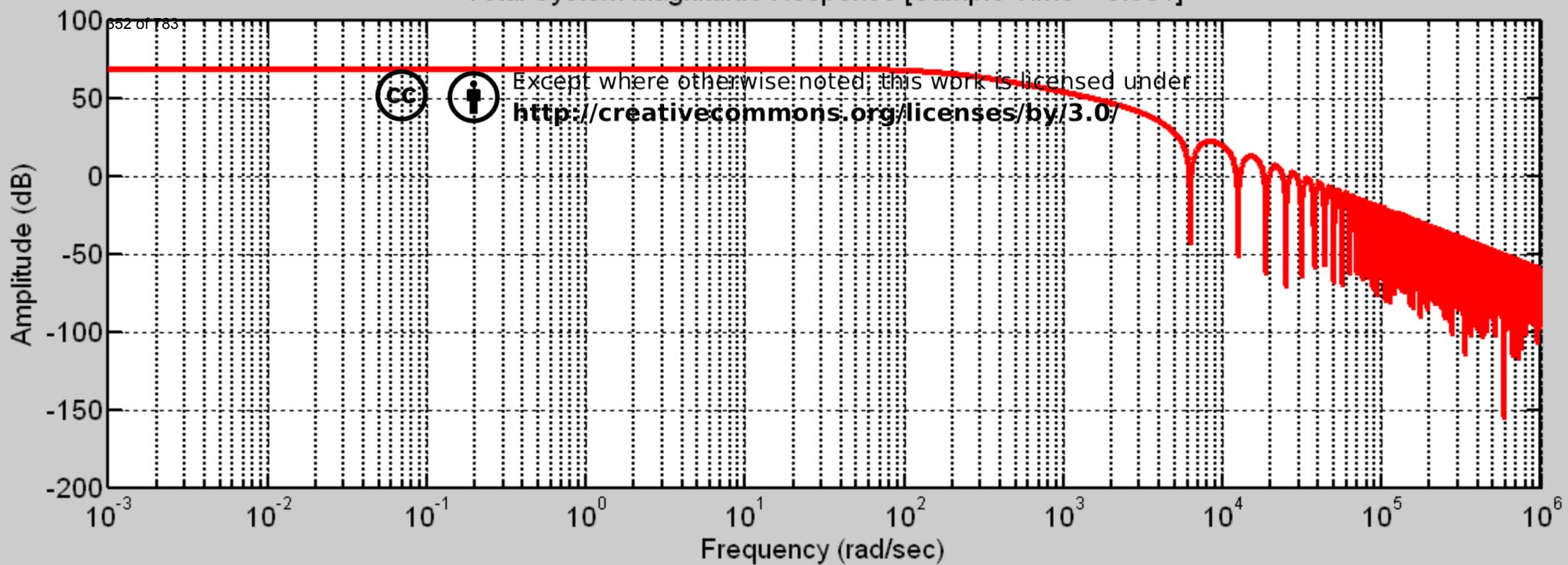
### Total System Magnitude Response [Sample Time = 0.01]



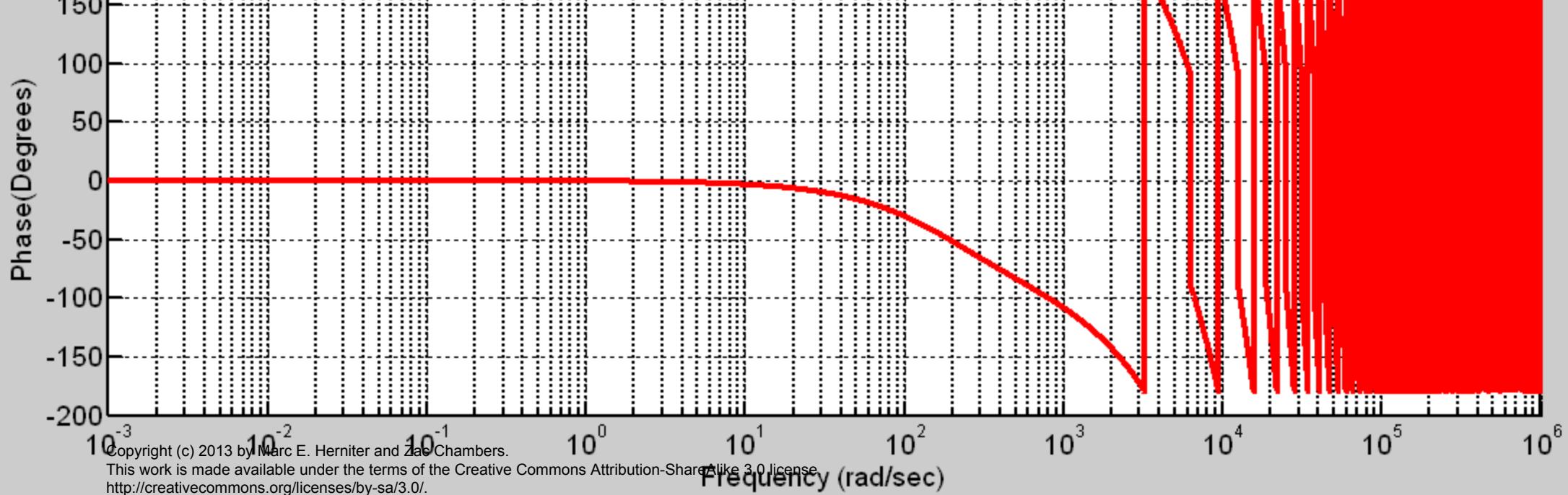
### Total System Phase Response [Sample Time = 0.01]



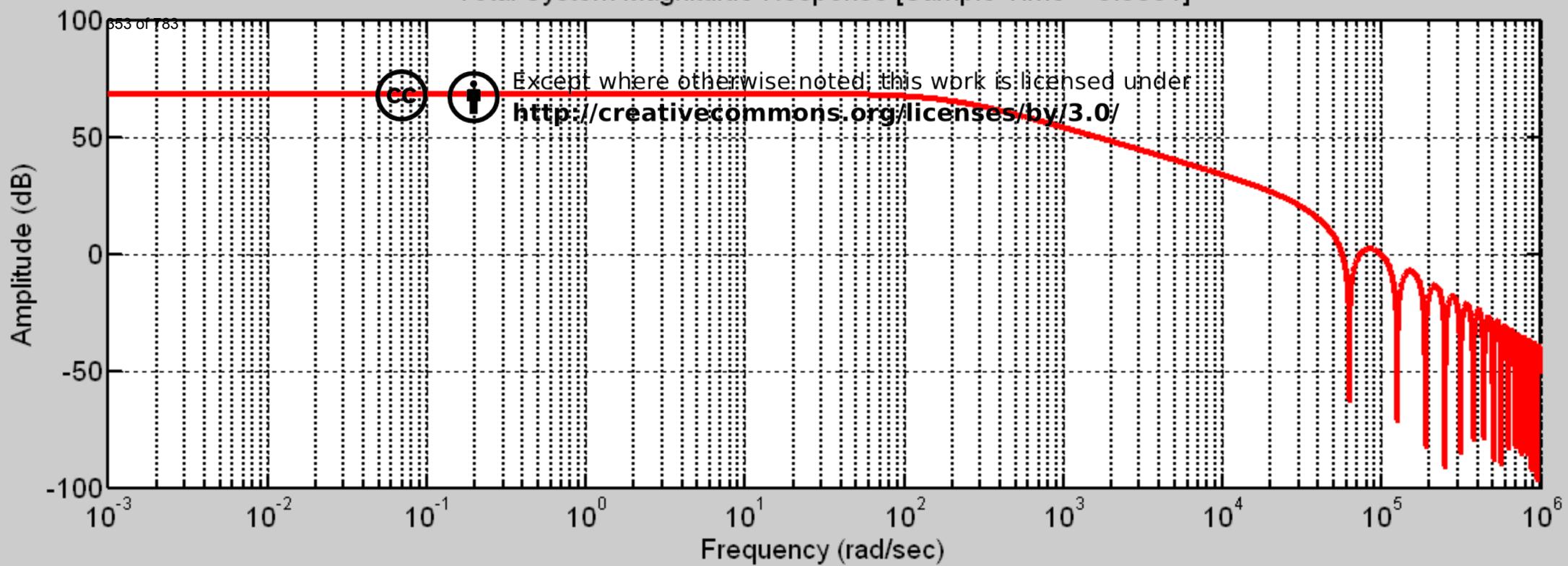
### Total System Magnitude Response [Sample Time = 0.001]



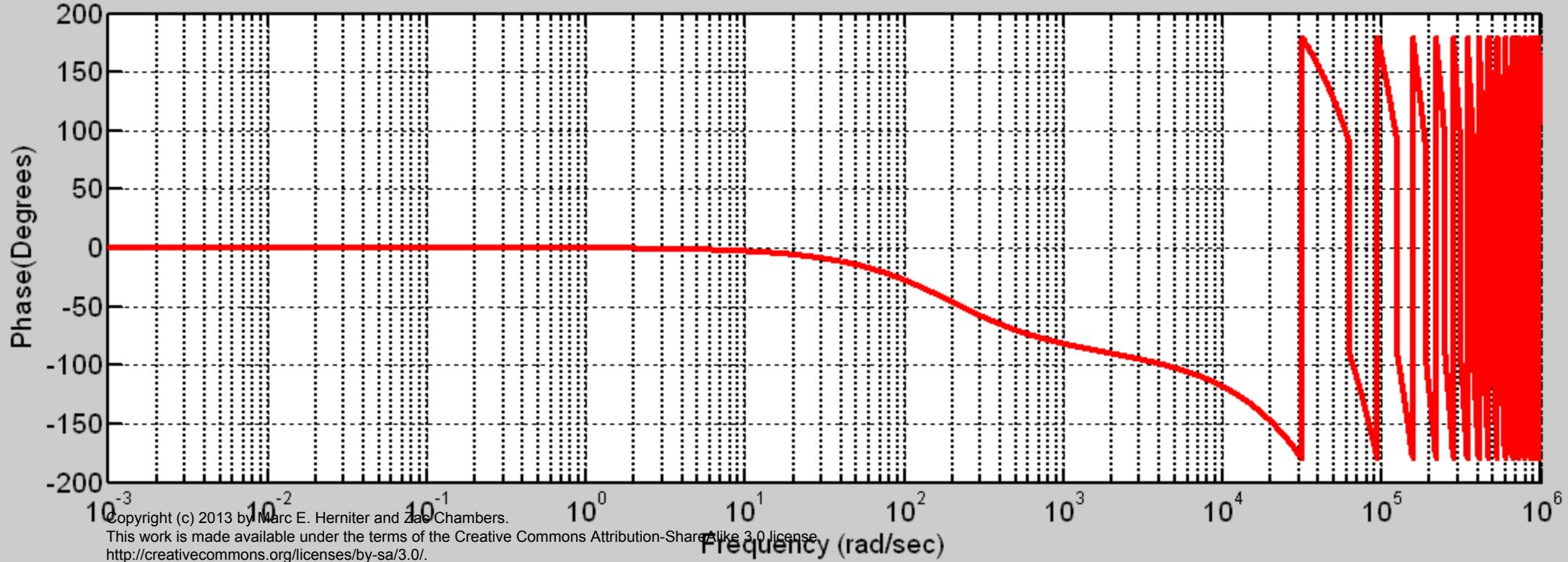
### Total System Phase Response [Sample Time = 0.001]

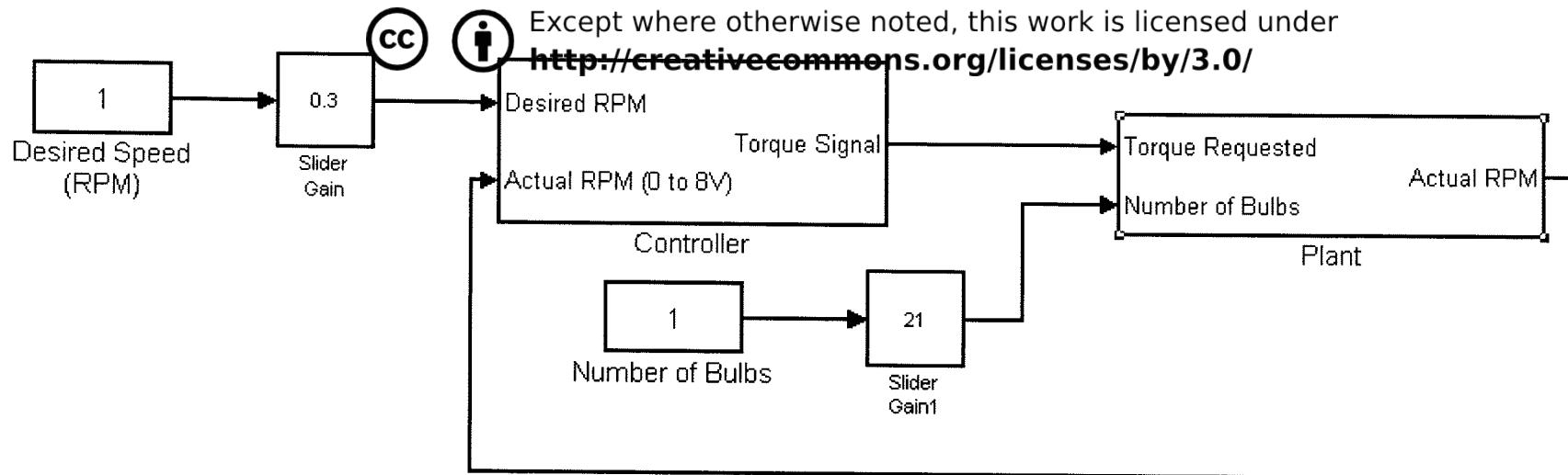


### Total System Magnitude Response [Sample Time = 0.0001]



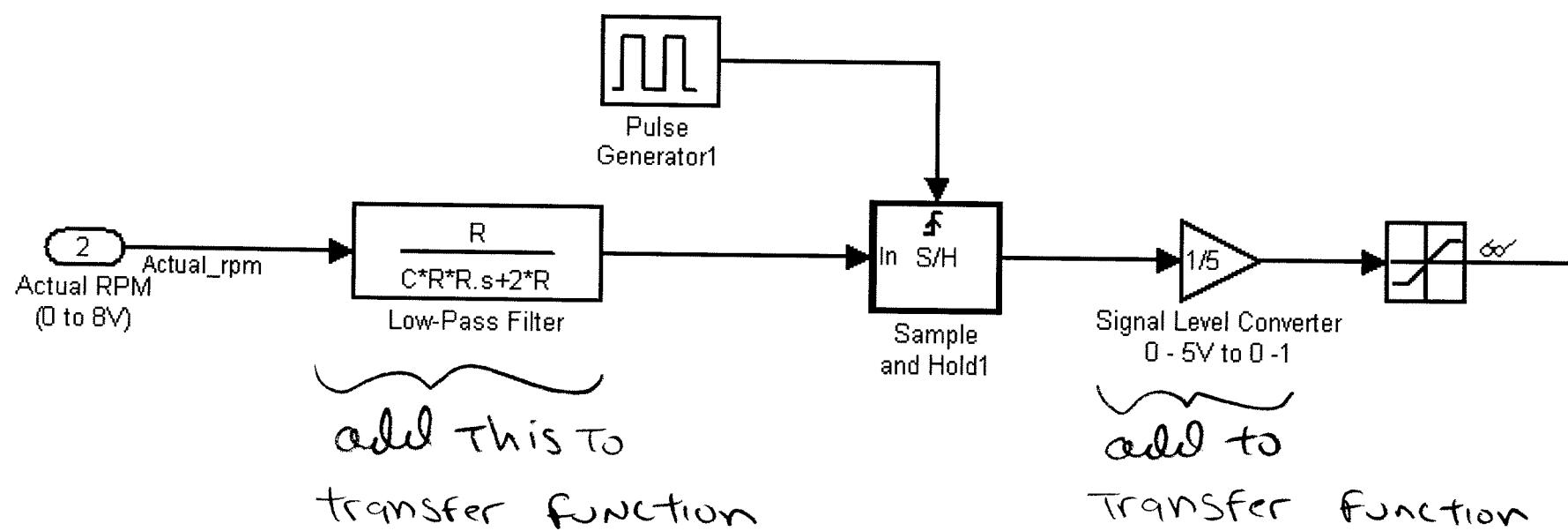
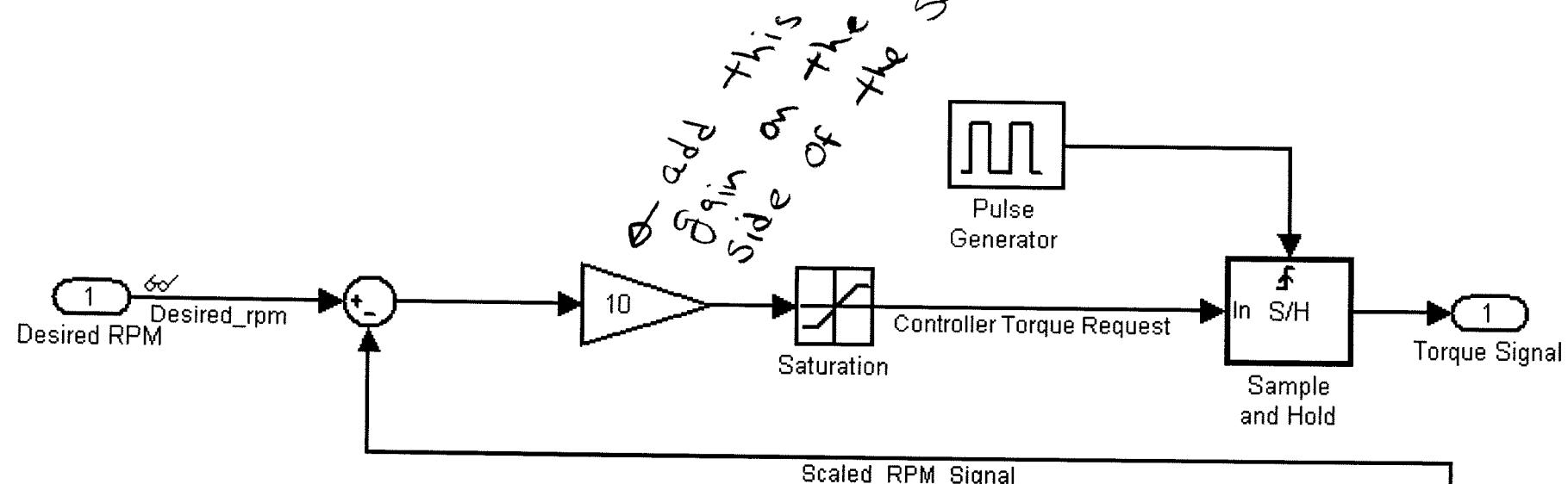
### Total System Phase Response [Sample Time = 0.0001]





- Review our latest plant and controller so that we can include all added gains & filters in our analysis

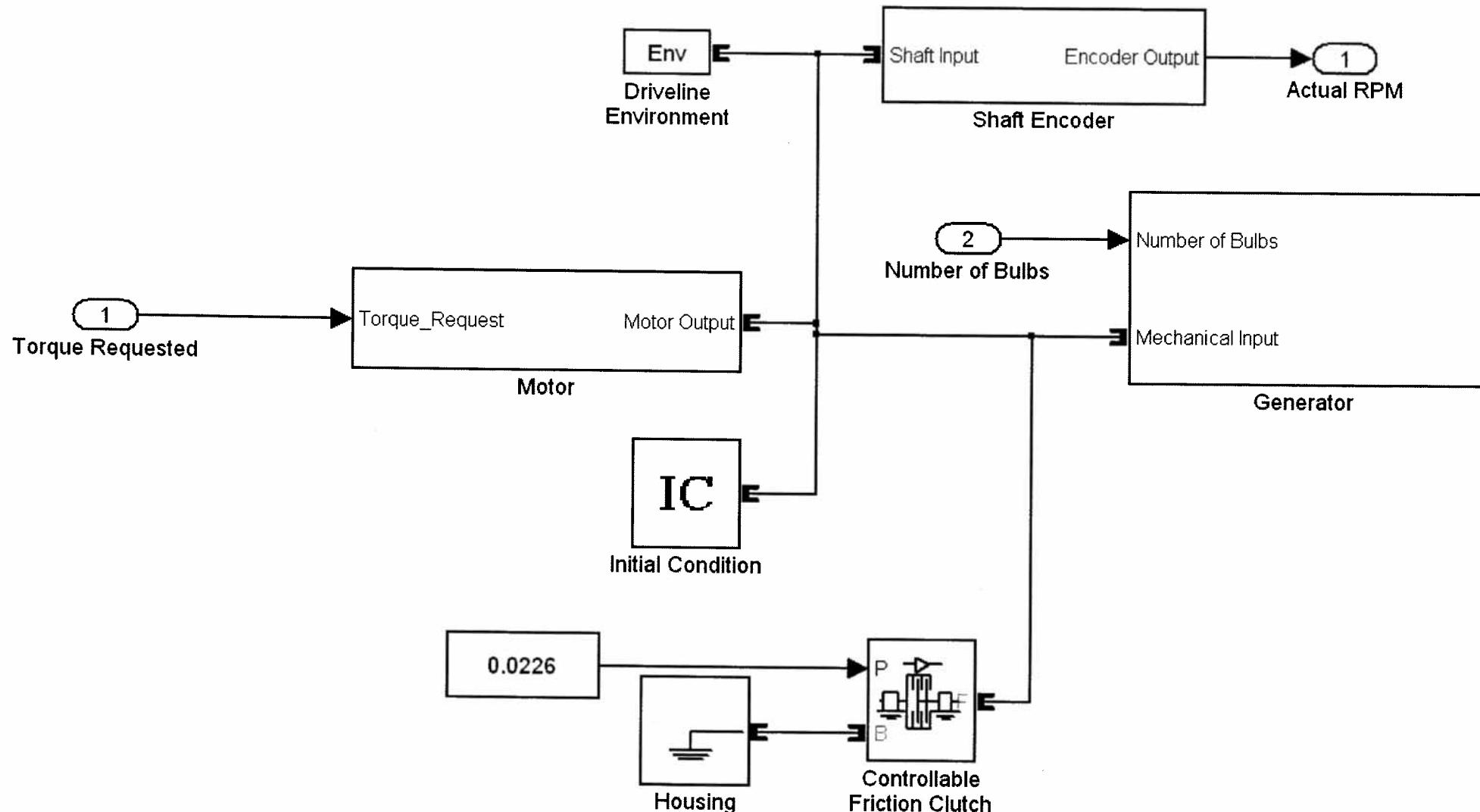
CC BY SA  
where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



# Plant



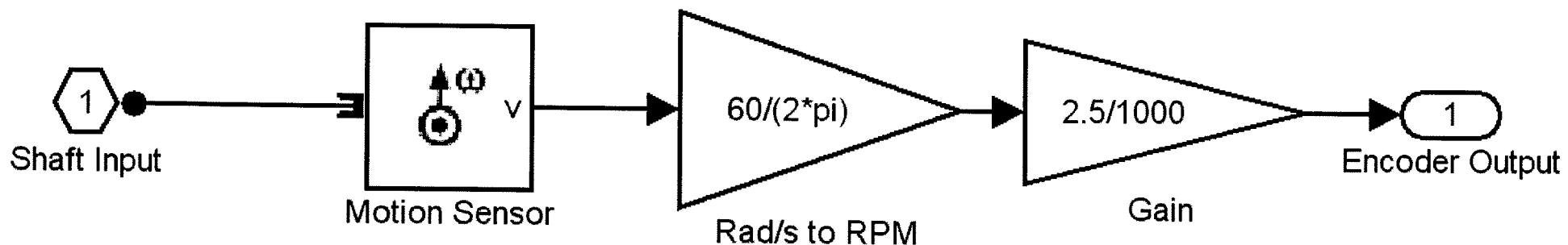
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



Not included in our measured plant transfer function



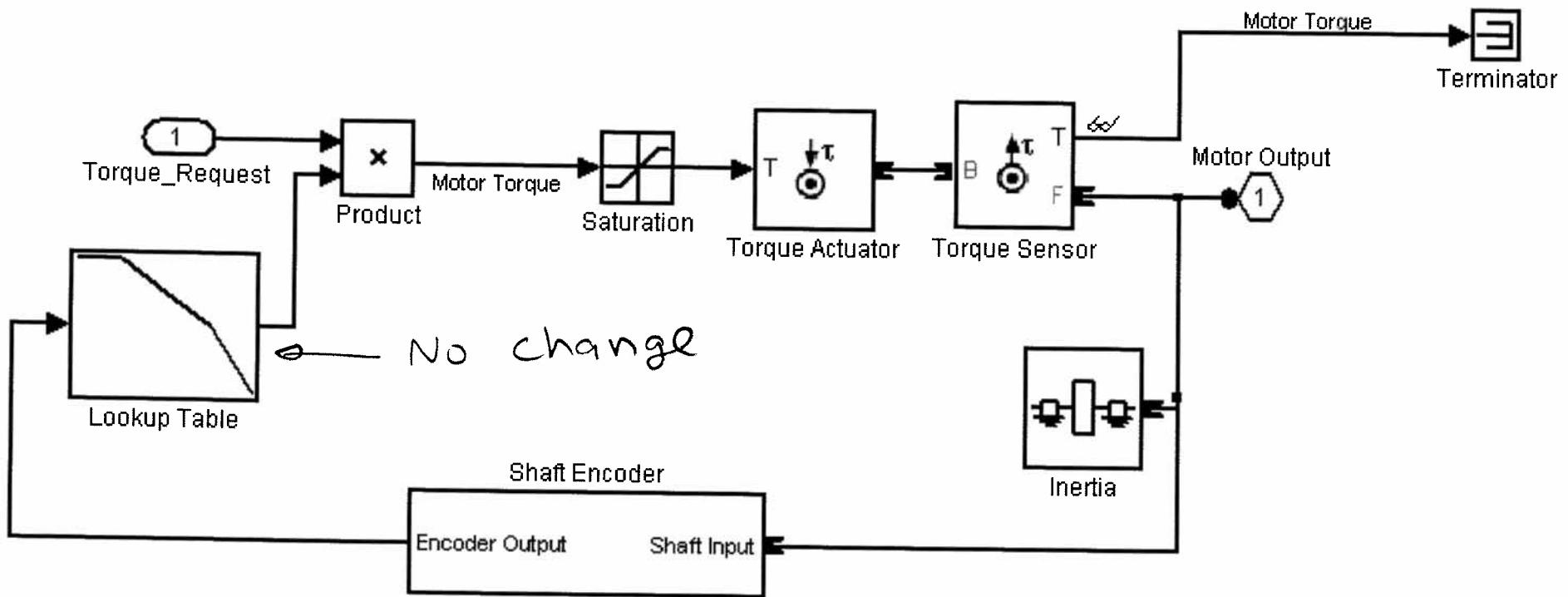
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



we will add this  
 gain to our loop  
 transfer function



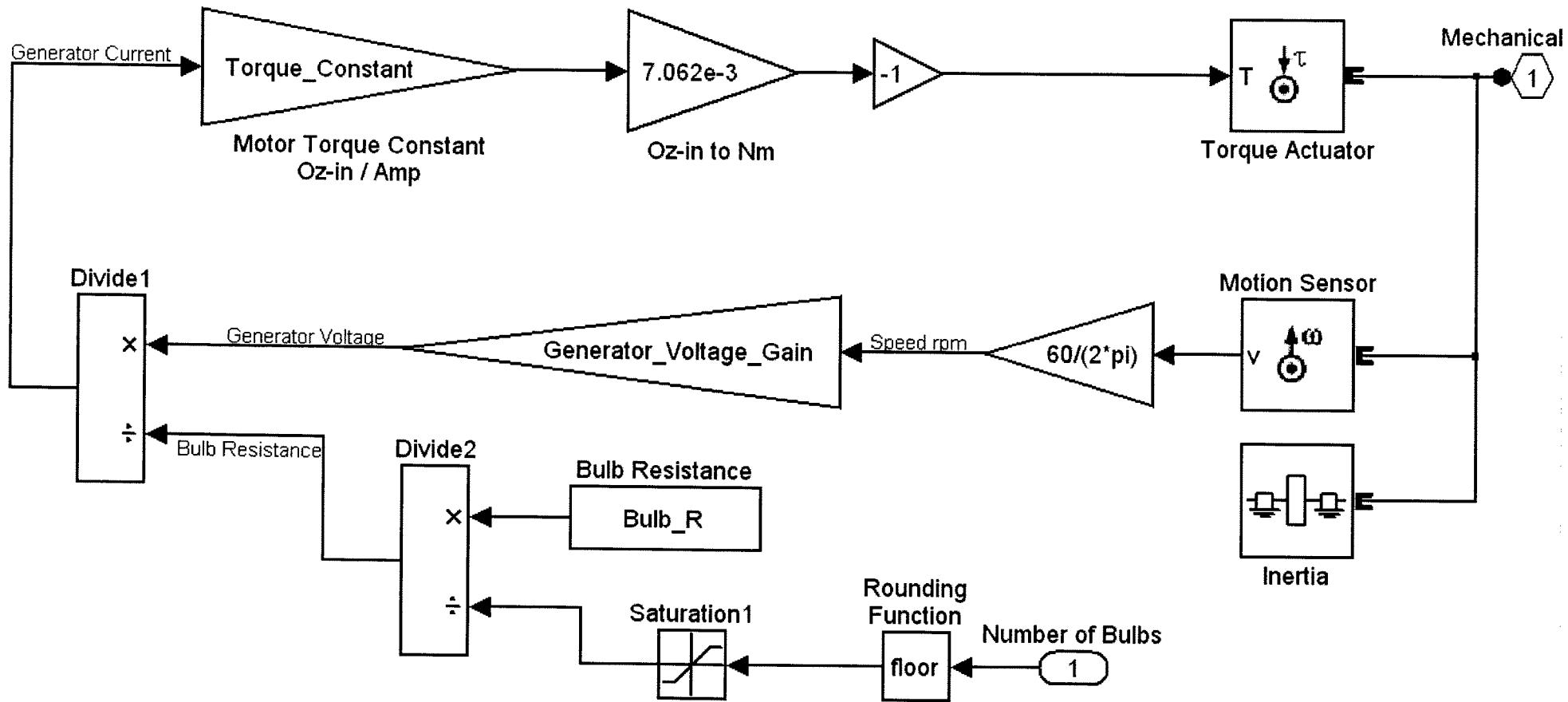
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



- measured plant transfer function was  
at 1800 rpm.



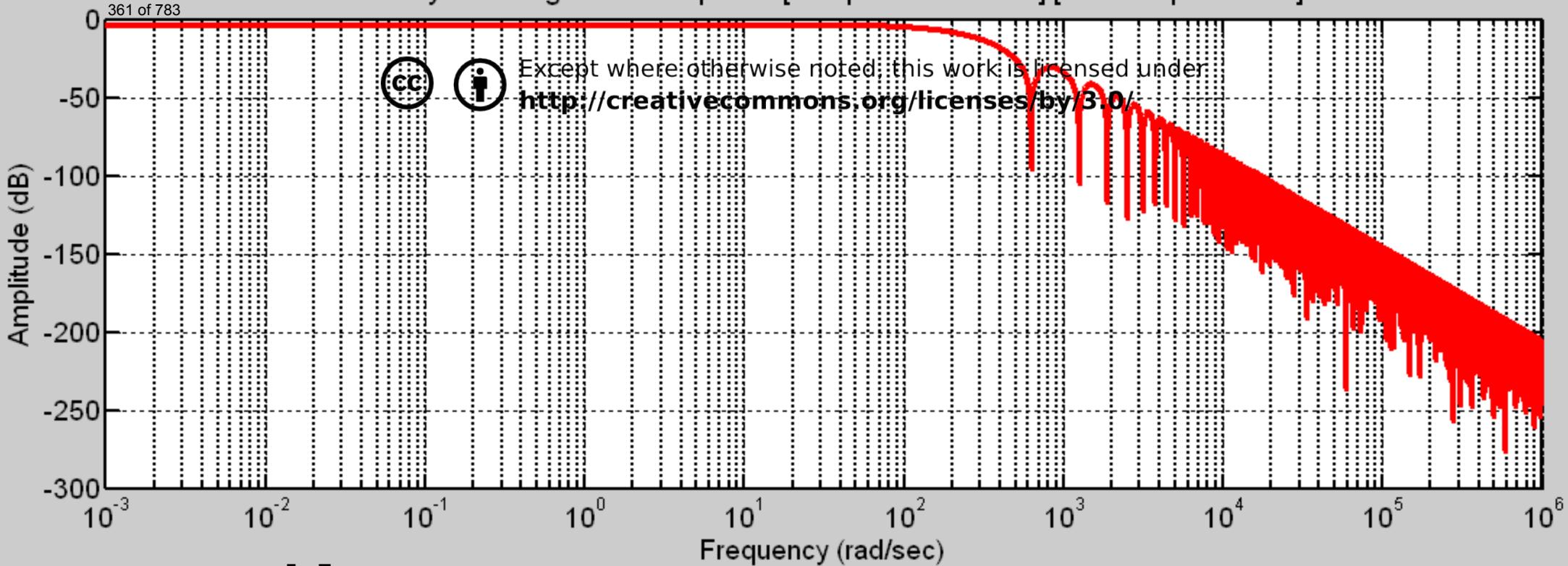
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



No changes - we measured the plant transfer function at a fixed bulb load. We will use the same load.

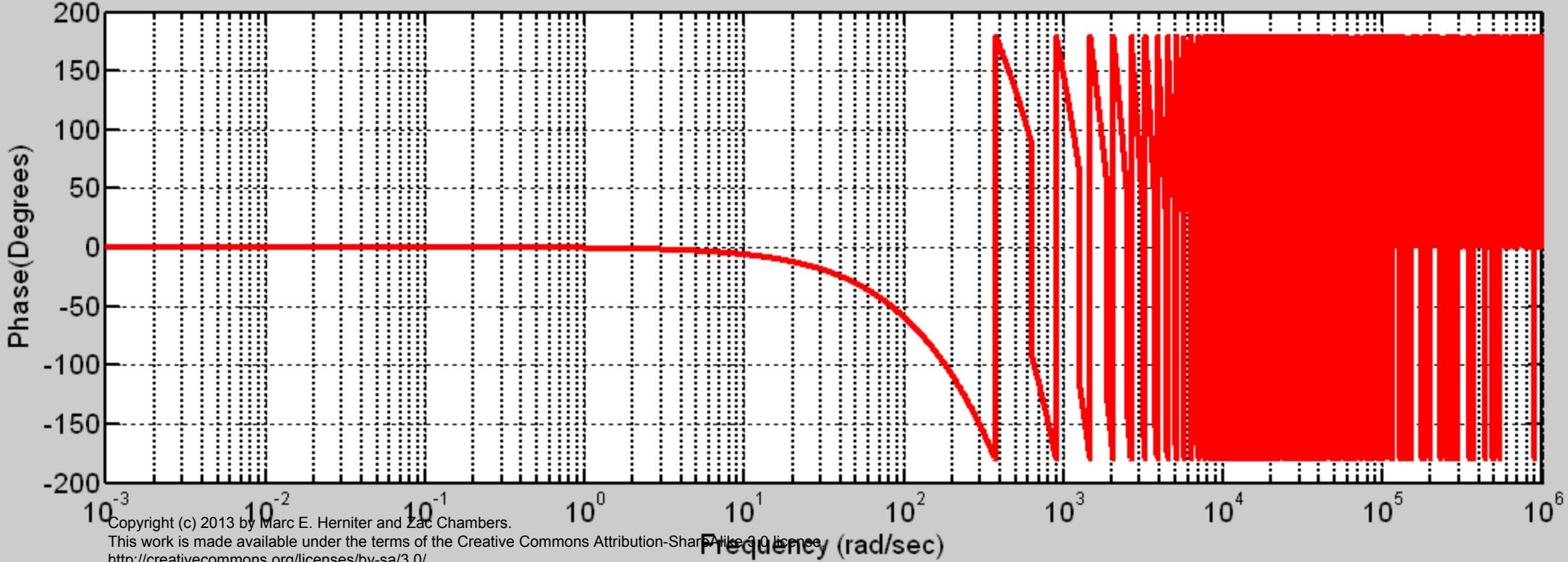
t0 = 0.01; %Sample time  
 msg=sprintf(' [Sample Time = %g]',t0);  
 omega=logspace(-3,6,100000); %Generate frequencies from  $10^{-3}$  to  $10^6$   
 % Or frequencies from 0.001 Hz/second to 1,000,000 rad/sec  
 % Except where otherwise noted, this work is licensed under  
 % Creative Commons Attribution-ShareAlike 3.0 license  
 % http://creativecommons.org/licenses/by/3.0/  
 H=(2600.\*200)./(j.\*omega+200); %Evaluate the plant transfer function specified frequencies.  
 SH=(1/t0)\*(1-exp(-j.\*omega.\*t0))./(j.\*omega); %Evaluate the sample and hold at specified frequencies.  
 % Low Pass Filter  
 R=10000; %Ohms  
 C=0.1e-6; %Farads  
 LP=R./(j.\*omega.^2\*R\*C+2\*R);  
 %Signal Level Converter  
 SLC = 1/5;  
 % Encoder Gain  
 EG=2.5/1000;  
 %Feedback Gain  
 F=1;  
 msg2=sprintf(' [Error Amp Gain = %g]',F);  
 TF=SH.\*H.\*LP.\*SLC.\*EG.\*F; %The overal transfer function.  
 mag=abs(TF);  
 dB=20\*log10(mag);  
 phase=180\*angle(TF)/pi;  
 subplot(2,1,1)  
 semilogx(omega,dB,'r-');  
 grid on  
 title(['Total System Magnitude Response',msg,msg2]);  
 ylabel('Amplitude (dB)');  
 xlabel('Frequency (rad/sec)');  
 subplot(2,1,2)  
 semilogx(omega,phase,'r-');  
 grid on  
 title(['Total System Phase Response',msg,msg2]);  
 ylabel('Phase(Degrees)');  
 xlabel('Frequency (rad/sec)');

### Total System Magnitude Response [Sample Time = 0.01] [Error Amp Gain = 1]

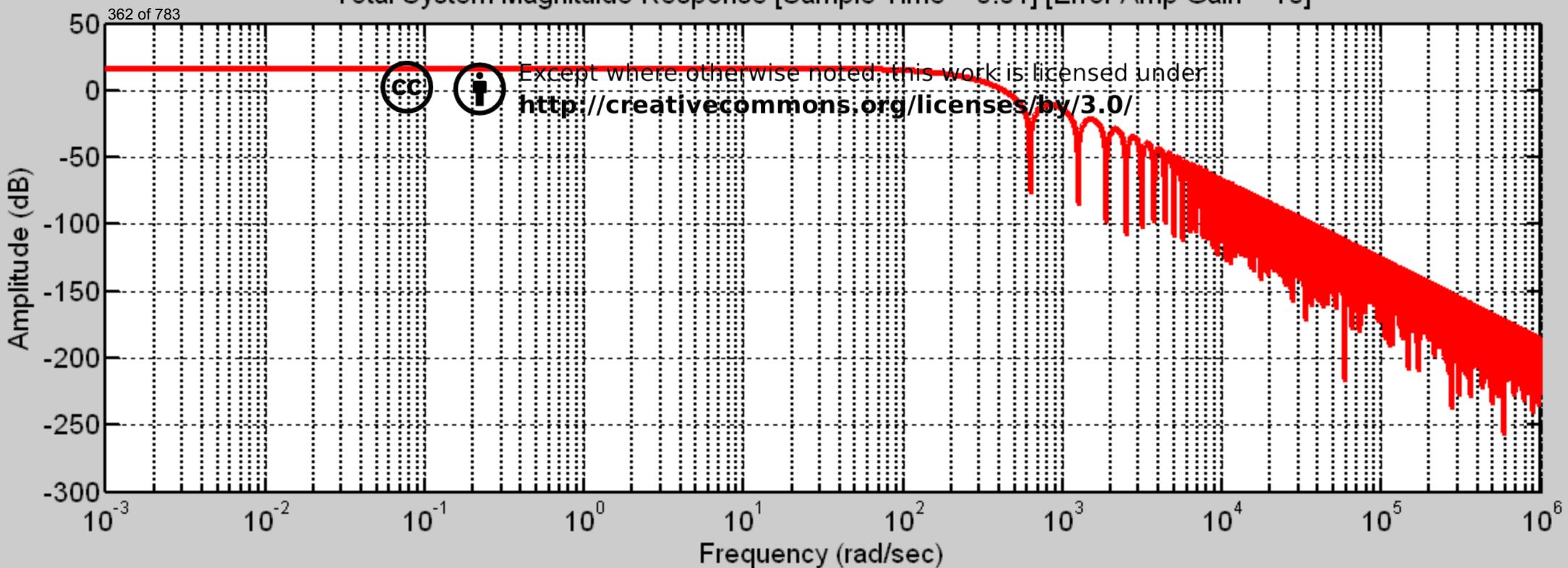


**System Stable**

### Total System Phase Response [Sample Time = 0.01] [Error Amp Gain = 1]

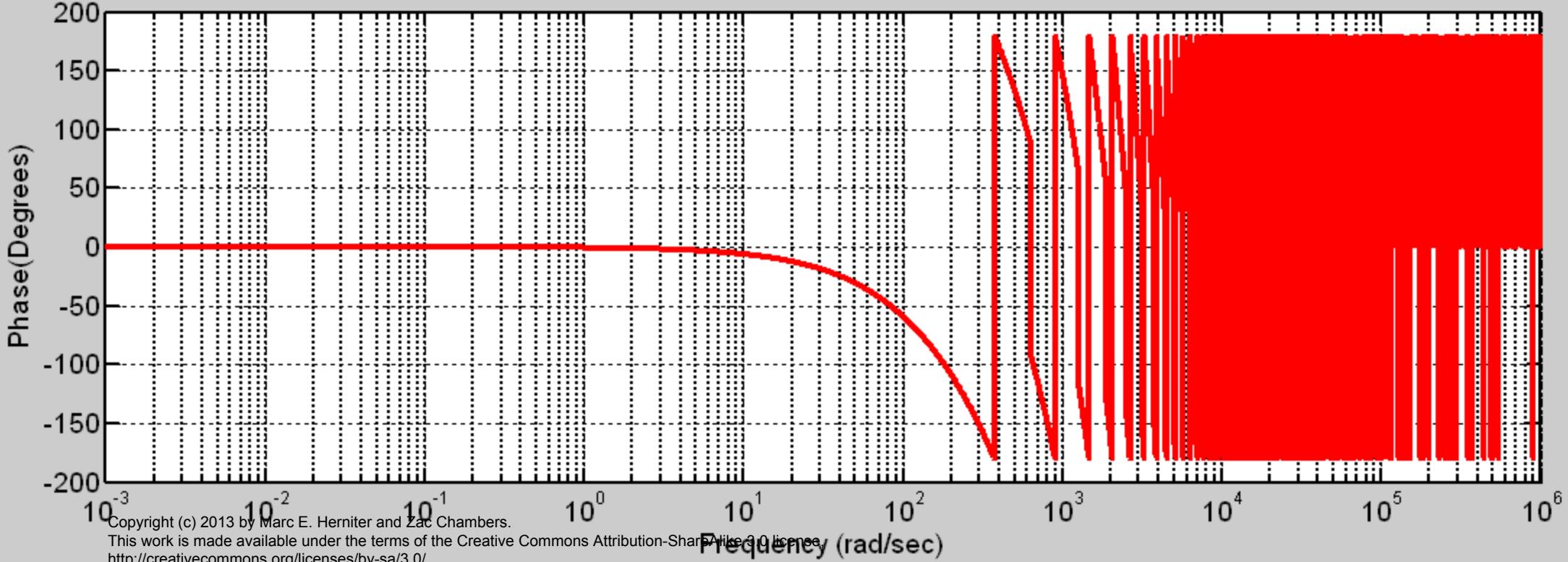


### Total System Magnitude Response [Sample Time = 0.01] [Error Amp Gain = 10]

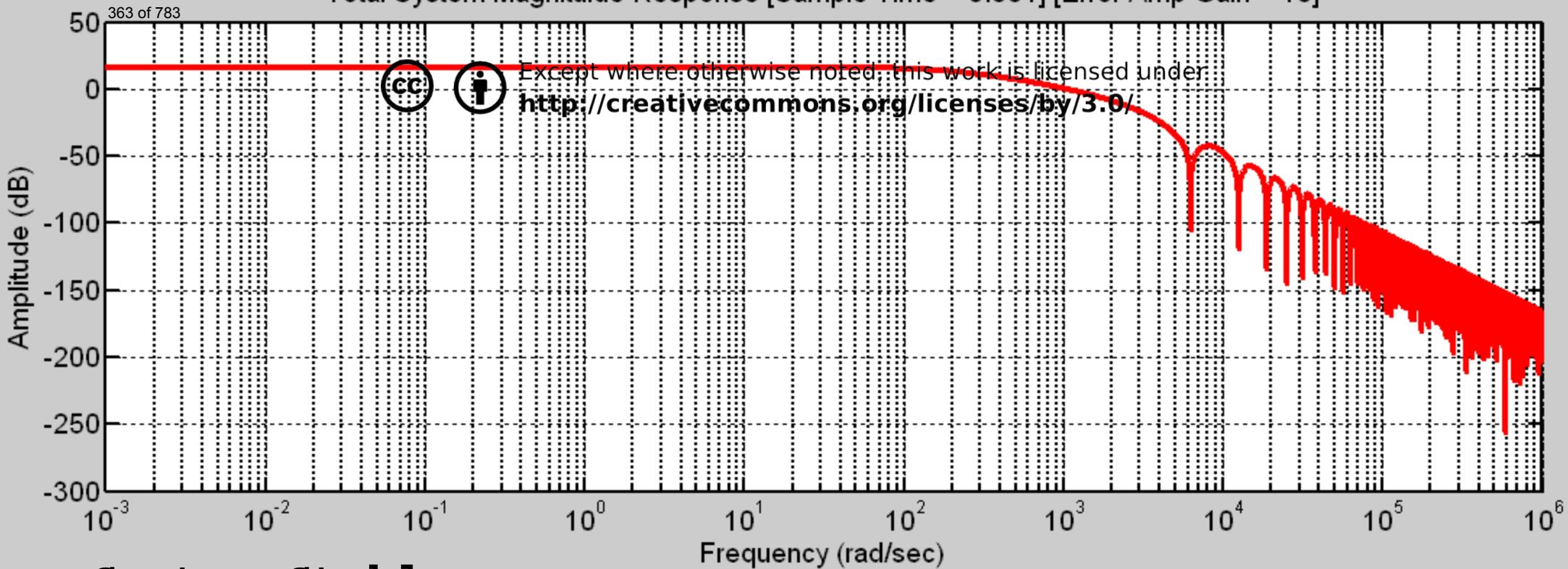


**System Probably Unstable**

### Total System Phase Response [Sample Time = 0.01] [Error Amp Gain = 10]

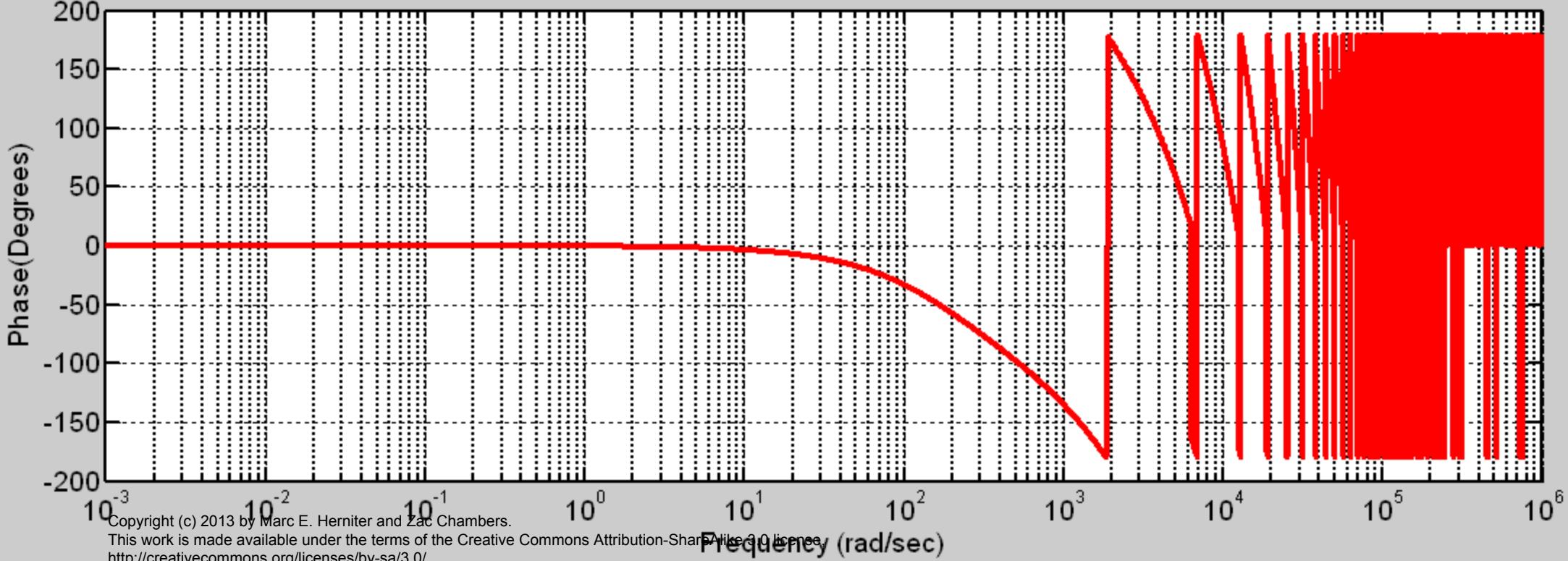


### Total System Magnitude Response [Sample Time = 0.001] [Error Amp Gain = 10]

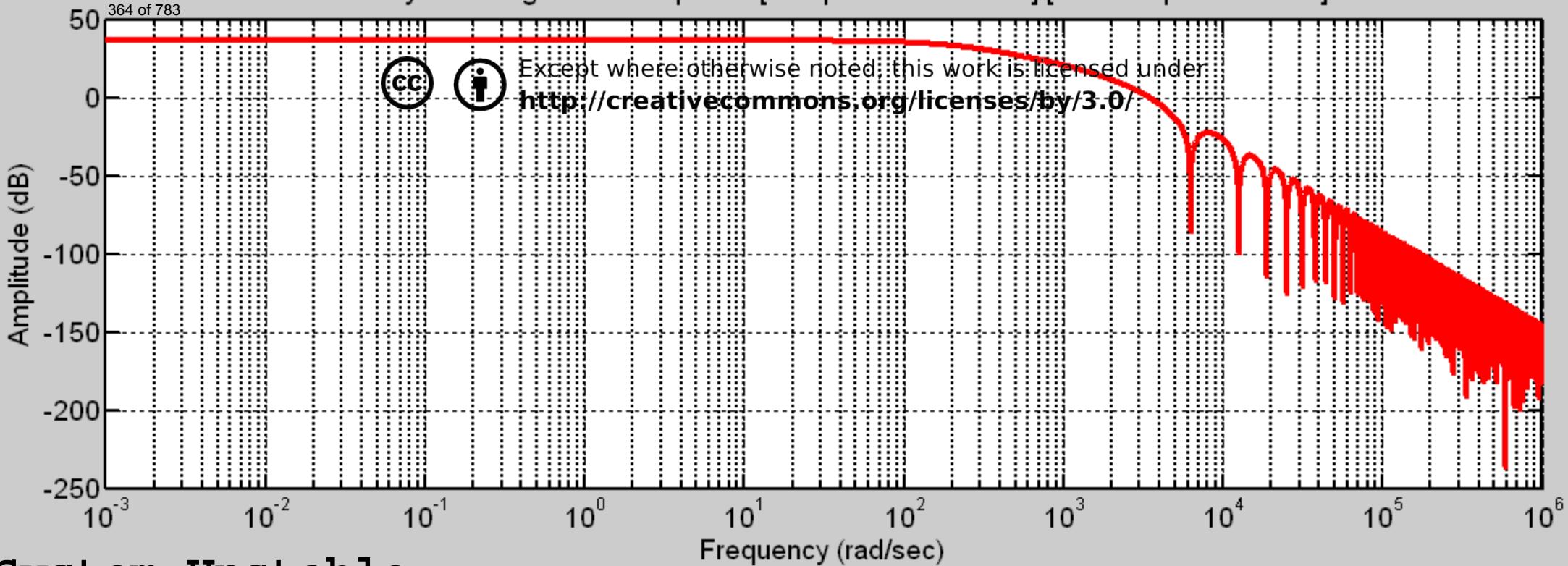


**System Stable**

### Total System Phase Response [Sample Time = 0.001] [Error Amp Gain = 10]

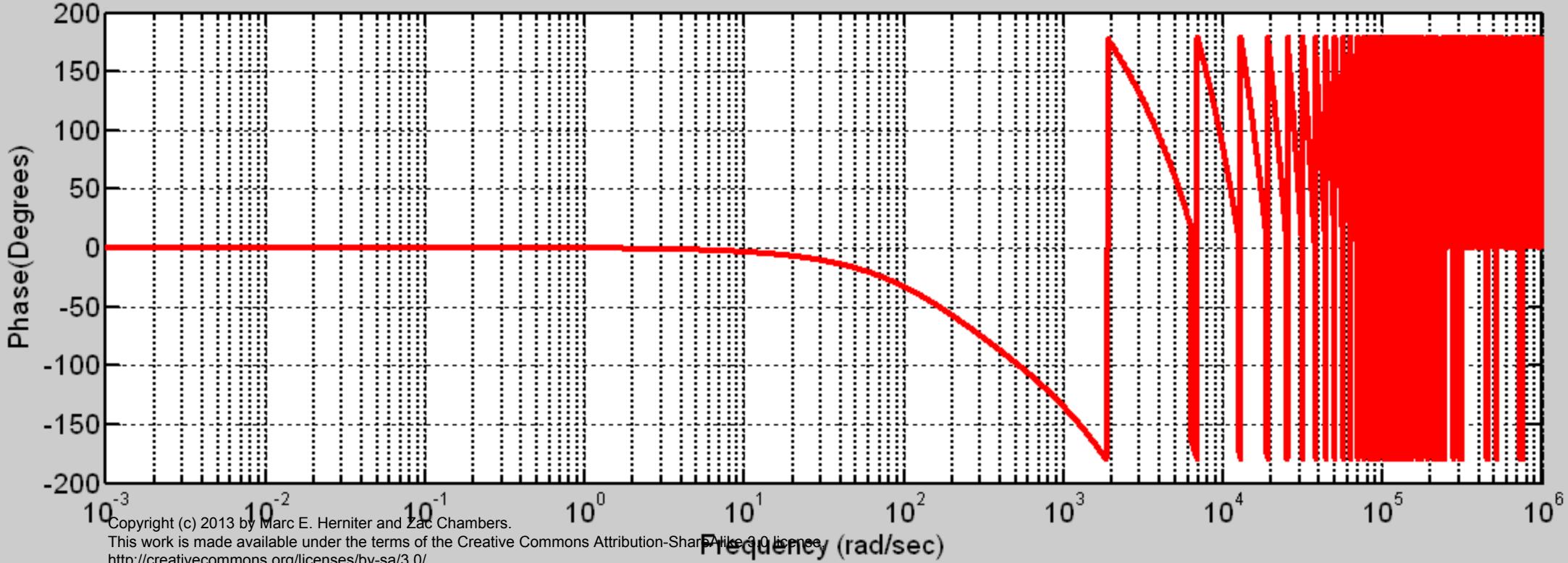


### Total System Magnitude Response [Sample Time = 0.001] [Error Amp Gain = 100]

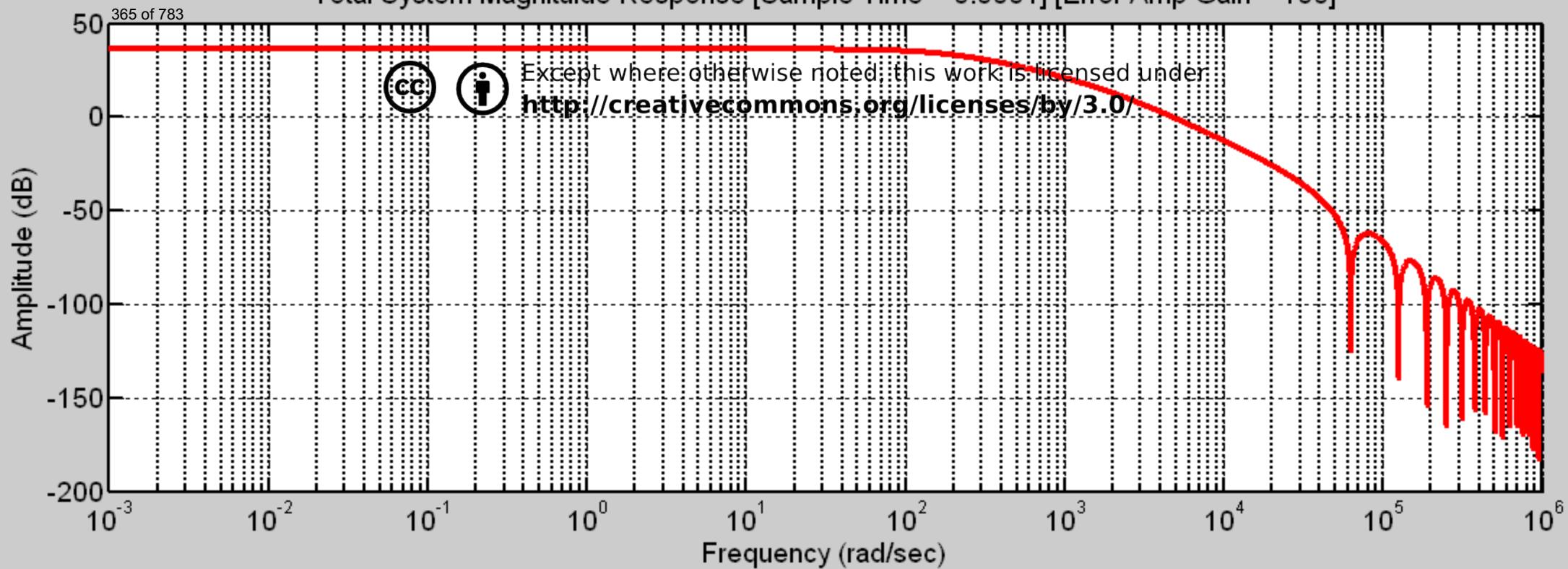


**System Unstable**

### Total System Phase Response [Sample Time = 0.001] [Error Amp Gain = 100]

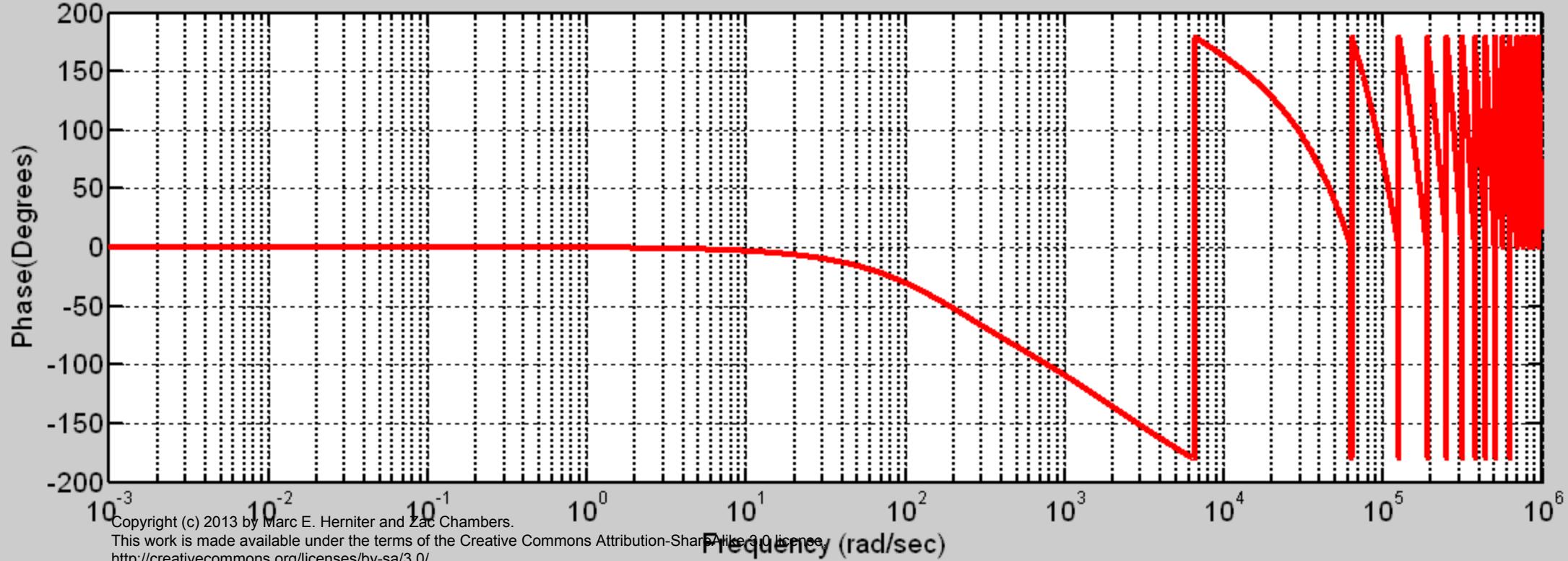


# Total System Magnitude Response [Sample Time = 0.0001] [Error Amp Gain = 100]



**System Stable**

# Total System Phase Response [Sample Time = 0.0001] [Error Amp Gain = 100]





1

## Lecture 9 Problem 1

- The plots demonstrated in this lecture were for our motor-generator system **without the flywheel**.
- For the problems in this section, we would like to use the transfer function from the system with the flywheel.
- In lecture 4 problem 1, we measured the frequency response plot of the motor-generator system including the flywheel.
- Determine the transfer function of this system from the measured frequency plot.
- Show the magnitude frequency plot whence your transfer function was generated.

Answer \_\_\_\_\_



2

## Lecture 9 Problem 2

- For a feedback gain of 1, determine the largest fixed time step where the system will be stable (0.1, 0.01, 0.001, 0.0001)
- Show gain and phase plots using Matlab.
- The plots should be displayed in the same manner as shown in the notes for lecture 9.

Gain and plots \_\_\_\_\_





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

3

## Lecture 9 Problem 3

- For a feedback gain of 10, determine the largest fixed time step where the system will be stable (0.1, 0.01, 0.001, 0.0001)
- Show gain and phase plots using Matlab.
- The plots should be displayed in the same manner as shown in the notes for lecture 9.

Gain and plots \_\_\_\_\_



4

## Lecture 9 Problem 4

- For a feedback gain of 100, determine the largest fixed time step where the system will be stable (0.1, 0.01, 0.001, 0.0001)
- Show gain and phase plots using Matlab.
- The plots should be displayed in the same manner as shown in the notes for lecture 9.

Gain and plots \_\_\_\_\_





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



## Introduction to Model-Based Systems Design

### Lecture 10: xPC Real-Time Systems 2



The MathWorks



freescale  
semiconductor



## xPC - Demo

2

- Throughout this lecture you will need to use the **xpctargetspy** command to obtain copies of the xPC target screen while your model is running.
- Print the screens out and turn them in as a single handout.

XPCTARGETSPY



The MathWorks



freescale  
semiconductor





3

## Integral Controller – Class Exercise

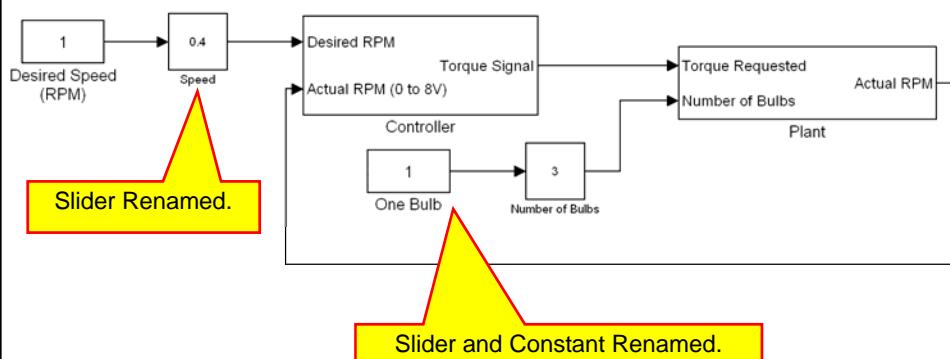
- For our next exercises, we would like to use the PI controller developed earlier and run it on the xPC real-time target.
- Open the model Lecture8\_Model1.mdl.
- Save the file as Lecture10\_Model1.mdl



4

## Integral Controller – Class Exercise

- The top level model should be as shown.

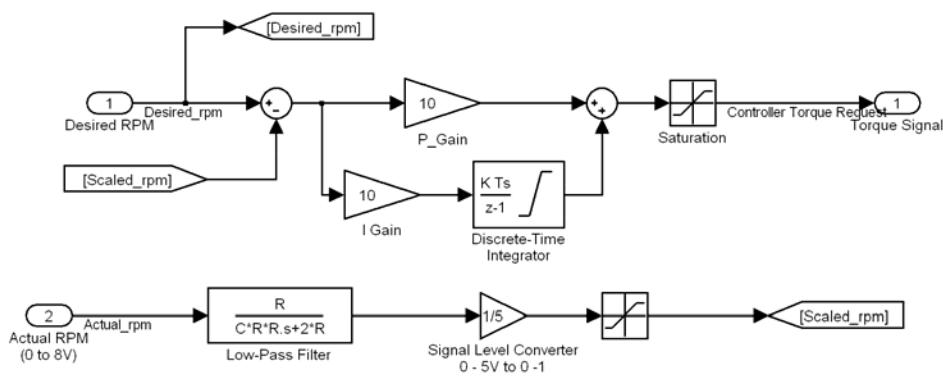




5

## Integral Controller

- Add the integrator we developed earlier in our SIL simulations.



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

6

## Integral Controller – Class Exercise

- Make the Following Modifications
  - Set the Simulation Configuration Parameters for:
    - a fixed step size of 1 ms using the ode4 solver
    - xPC Target on the default target
  - Add xPC Target scopes to display the following:
    - Motor rpm
    - Controller desired speed and actual speed
    - Proportional Gain and Integral Gain signals (Specify a scope y-axis range of -2 to 2).
    - Use same settings as in the previous simulation

**MotoTron**

The MathWorks

**freescale**  
semiconductor

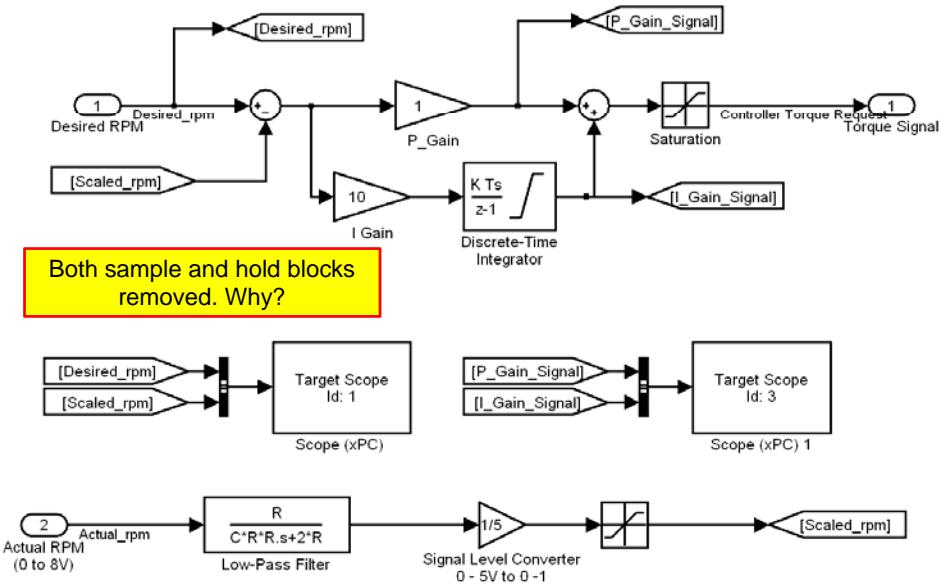
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Controller

7

P Gain =1, and I Gain = 10 and renamed P\_Gain and I\_Gain. (Note the underscore in the names.)

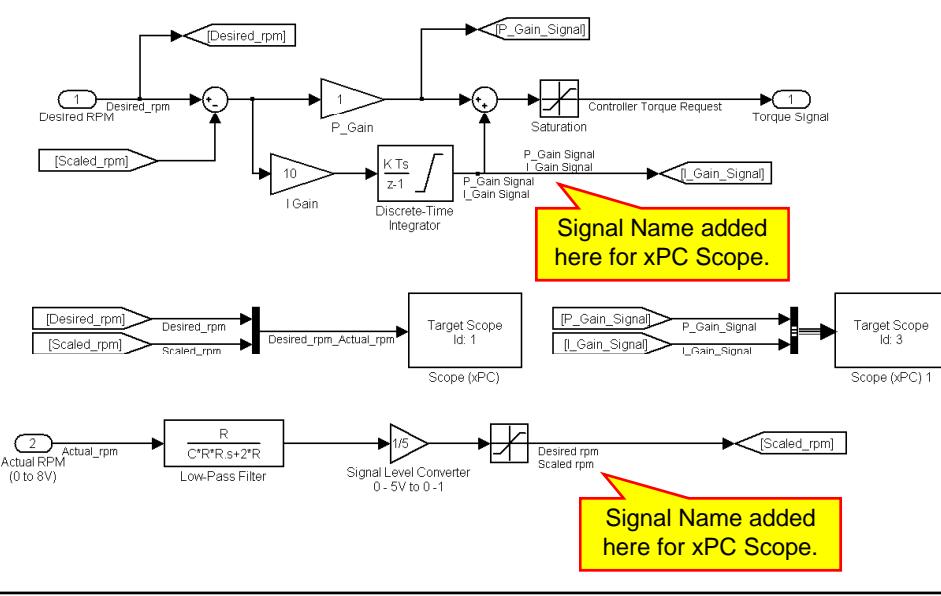


## Controller Model

8

Signal Name added here for xPC Scope.

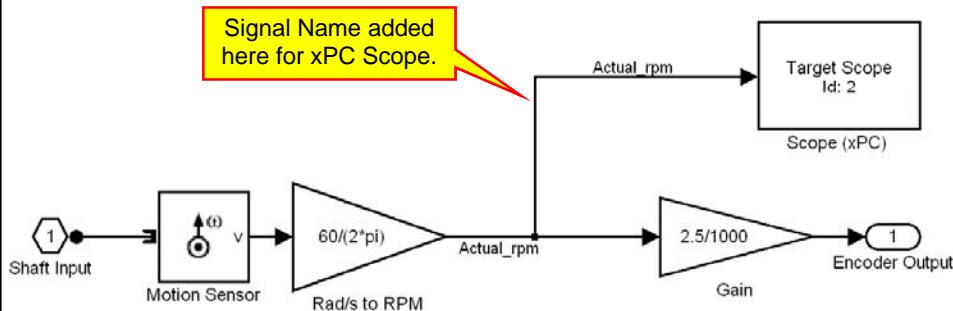
Signal Name added here for xPC Scope.





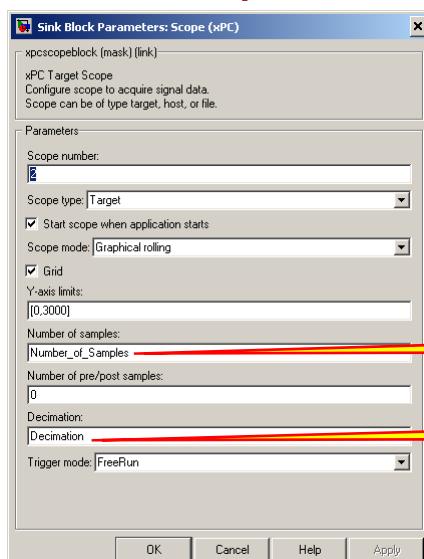
## Plant – Shaft Encoder

9

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

## xPC Scope Block Modifications

10



Make the changes below  
in all three xPC scope  
blocks in the model.

Value changed.

Value changed.

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



11

## Integral Controller – Class Exercise

- Run the simulation on your xPC Target.
- Note the following
  - The integral gives us over and undershoot.
  - The integral causes ringing.
  - The integral drives the error to zero in steady state. (Even with small proportional gain.)
- The following screens show the performance of my system.



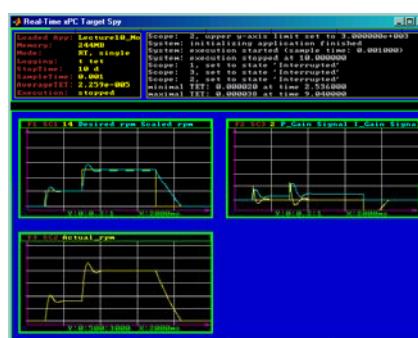
## Lecture 10 xPC 1

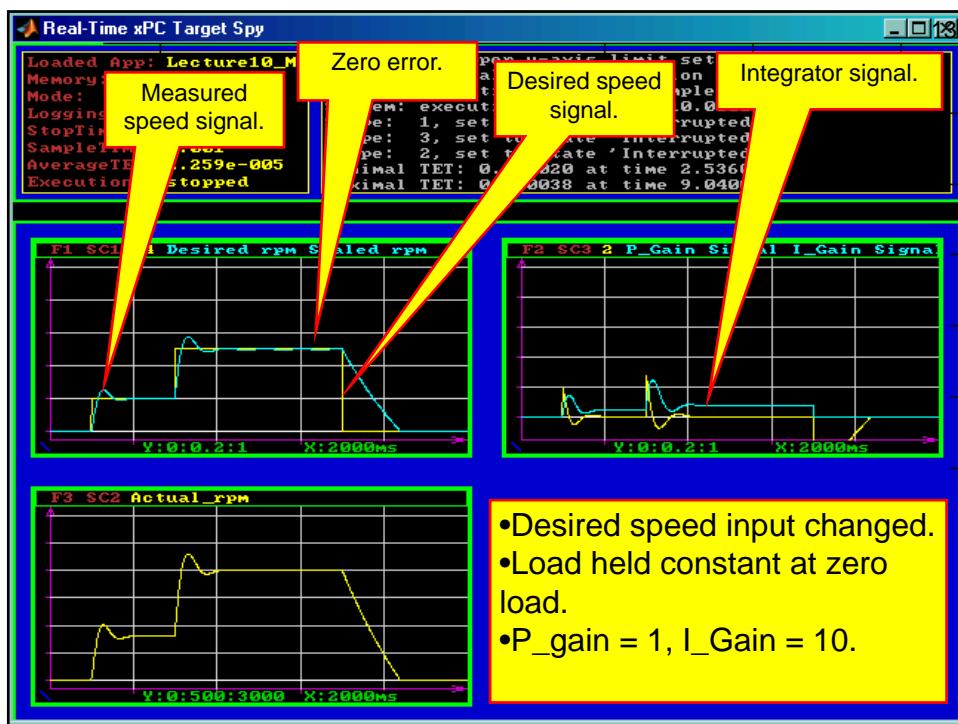
12

- Obtain a copy of the next screen using the `xpctargetspy` command.

**XPCTARGETSPY**

- Hint: Instead of using a slider, you may want to use a more repeatable input waveform.





## Lecture 10 xPC 2

14

- Obtain a copy of the next screen using the `xpctargetspy` command.

**XPCTARGETSPY**



**MotoTron**

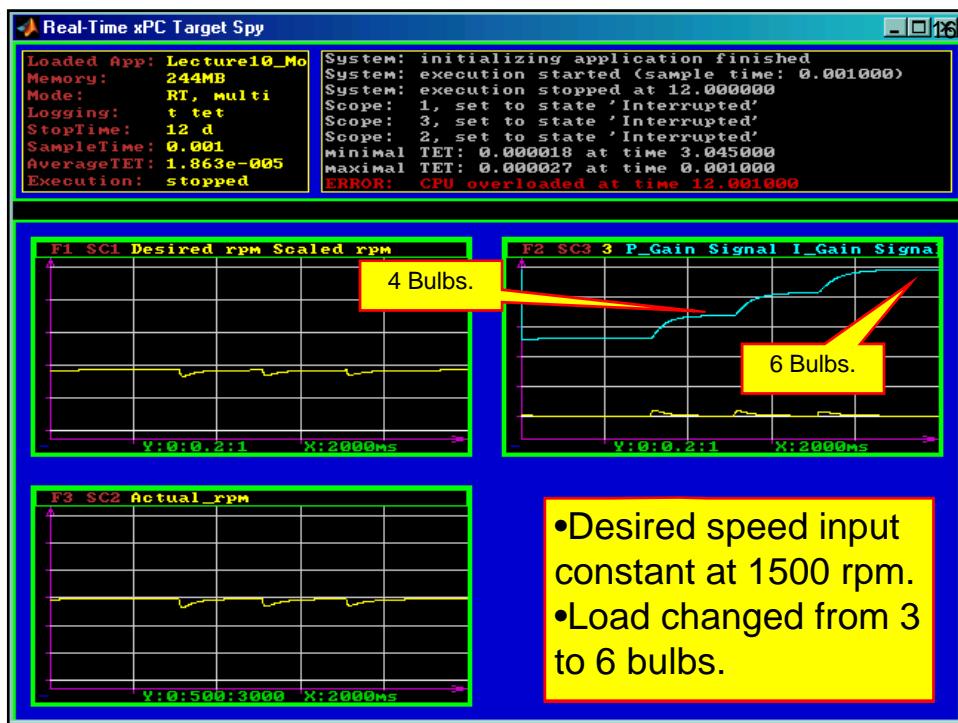
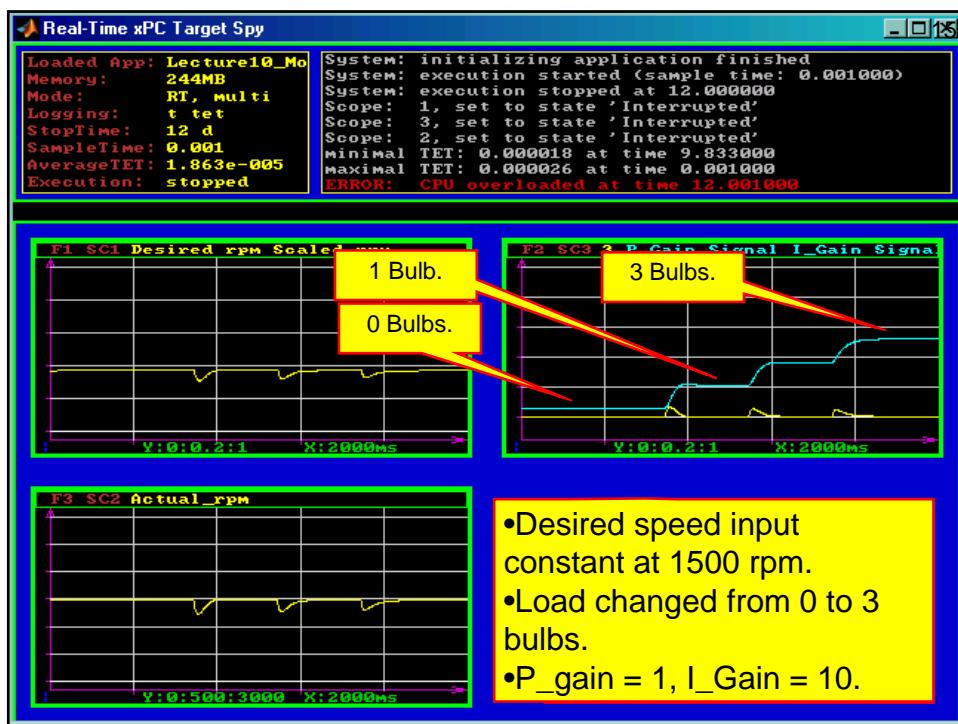
The MathWorks

freescale<sup>®</sup>  
semiconductor

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



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





17

## Parameter Tuning

- This more complicated PI controller had a number of interesting characteristics.
- We will use parameter tuning to change controller parameters (such as the proportional or integral gain).
- Parameters can be changed while the simulation is running.
- The ability to change the controller parameters on the fly gives us the ability to tune the controller to a desired performance.



The MathWorks

freescale<sup>®</sup>  
semiconductor

18

## Parameter Tuning

- There are two ways we can tune parameters:
  - Using the xPC Target Explorer
  - Connecting the Simulink model to the model running on the xPC target and changing model blocks.
- Both methods allow us to change model parameters while the model is executing in real-time on the xPC target.



The MathWorks

freescale<sup>®</sup>  
semiconductor



19

## Parameter Tuning

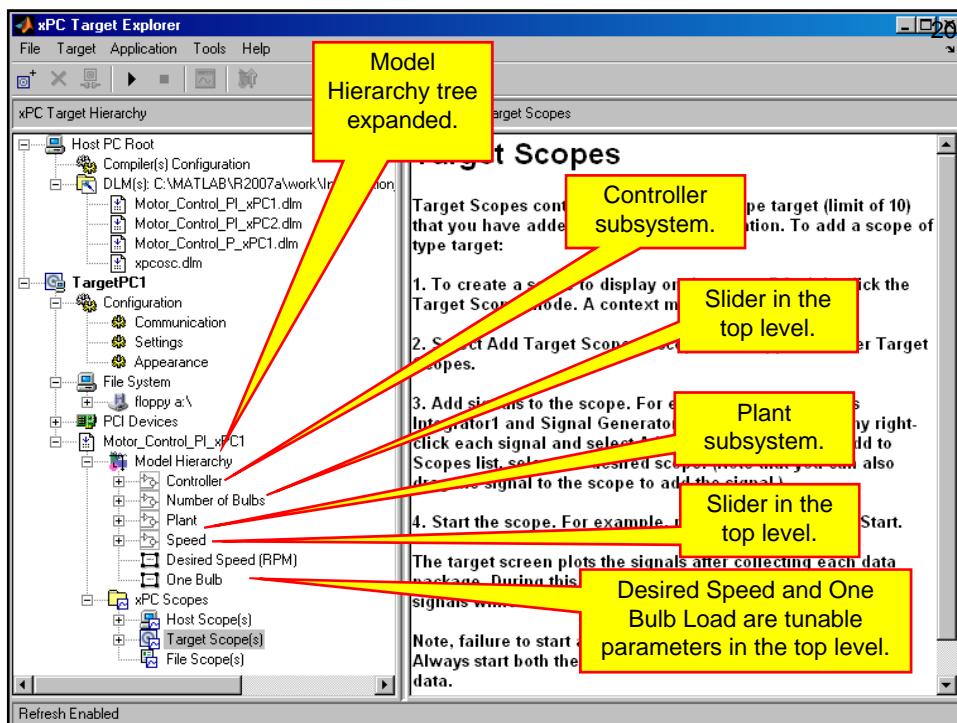
- We will use xPC Target Explorer to see what parameters are tunable in our model.
- Assuming that our PI model is loaded on the TargetPC1, connect to the model with xPC Target Explorer.
  - If your model is not loaded, build your model so that it is automatically loaded on the default target.
- Expand the tree under **Model Hierarchy**

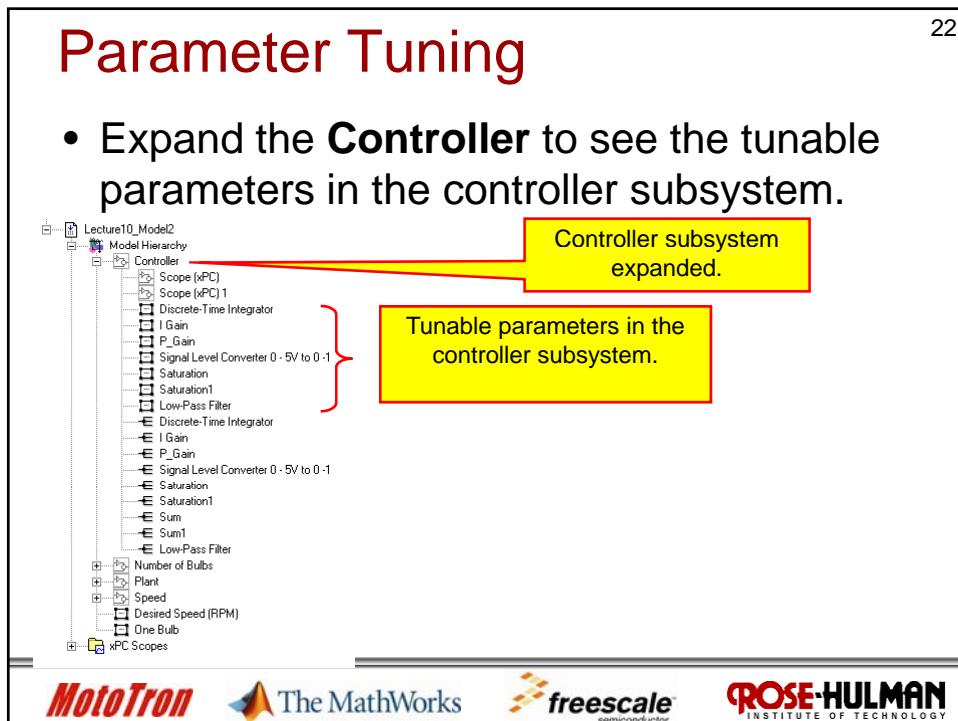
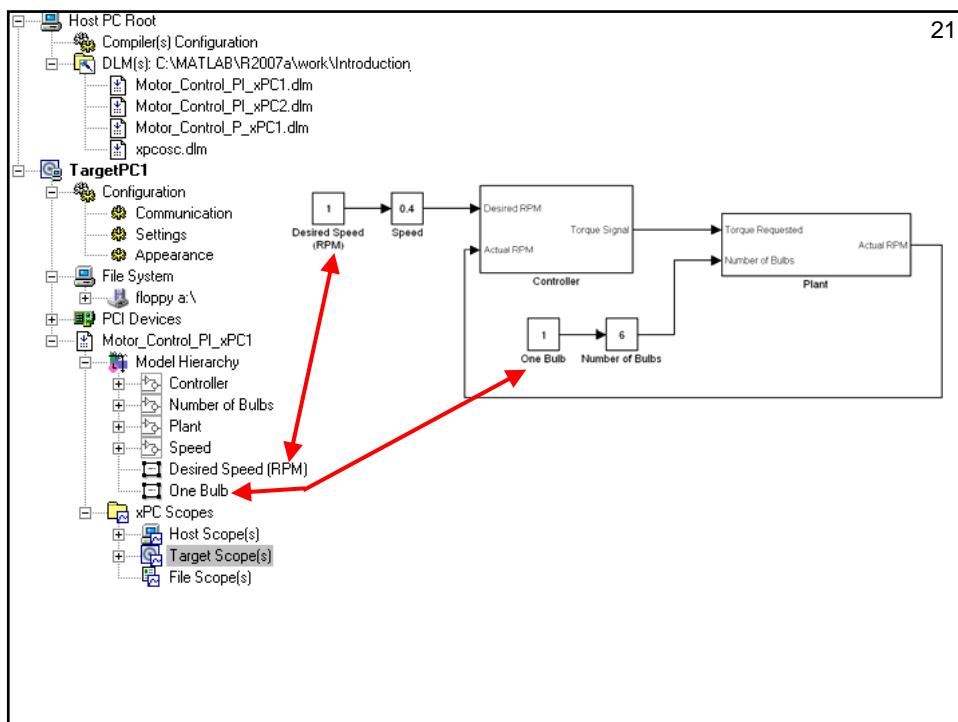
**MotoTron**

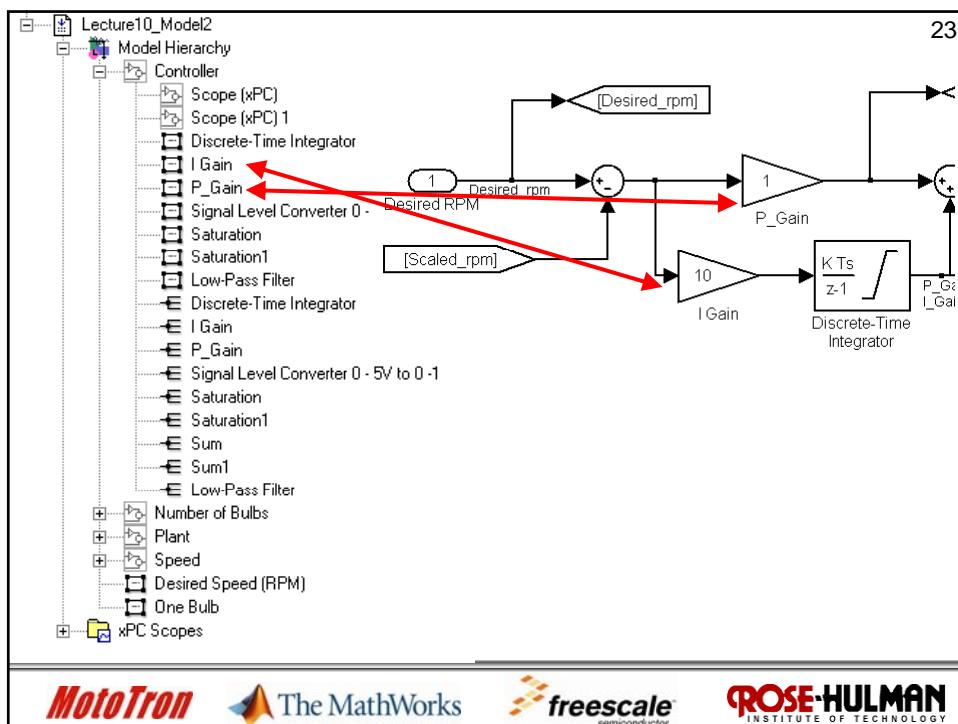
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY







## Parameter Tuning

24

- The xPC Target Explorer allows us to find out which parameters in the model we can change. (We can also use the xPC Target explorer to change the parameters, but we will do this later.)
- We will use the Simulink model to change the values of Proportional Gain and Integral Gain while the simulation is running.
- Close the xPC Target Explorer.
- Connect your Simulink model to the XPC Target model and then run the simulation.



The MathWorks

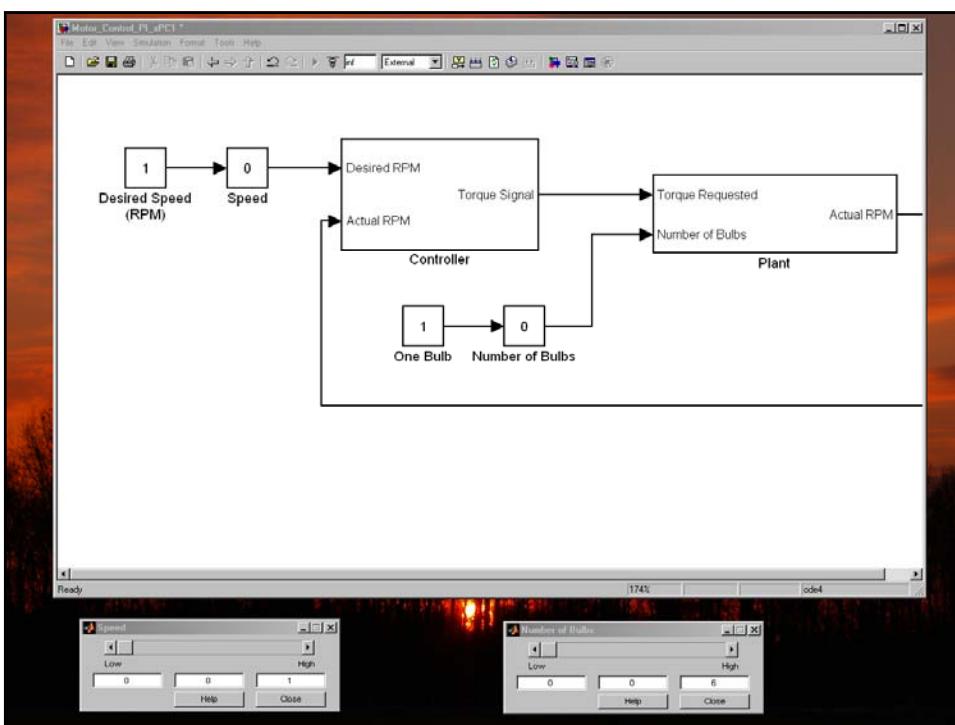




25

## Model Parameter Tuning

- While the simulation is running:
- Open up the two sliders for the load and desired speed.
- Set both sliders to zero.
- Open up the controller subsystem.
- A nice arrangement for the windows is shown on the next slide.

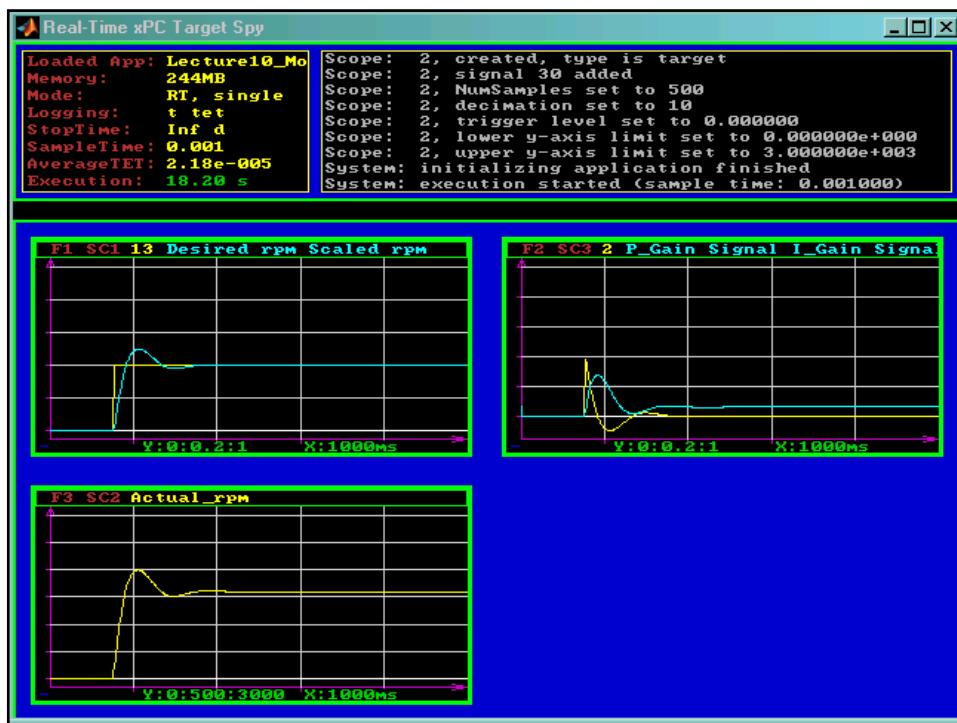




27

## Parameter Tuning

- Before we change any parameters, lets run a step response with no load.
  - Leave the load slider at 0 bulbs.
  - Change the desired speed slider to 0.4 and observe the results.



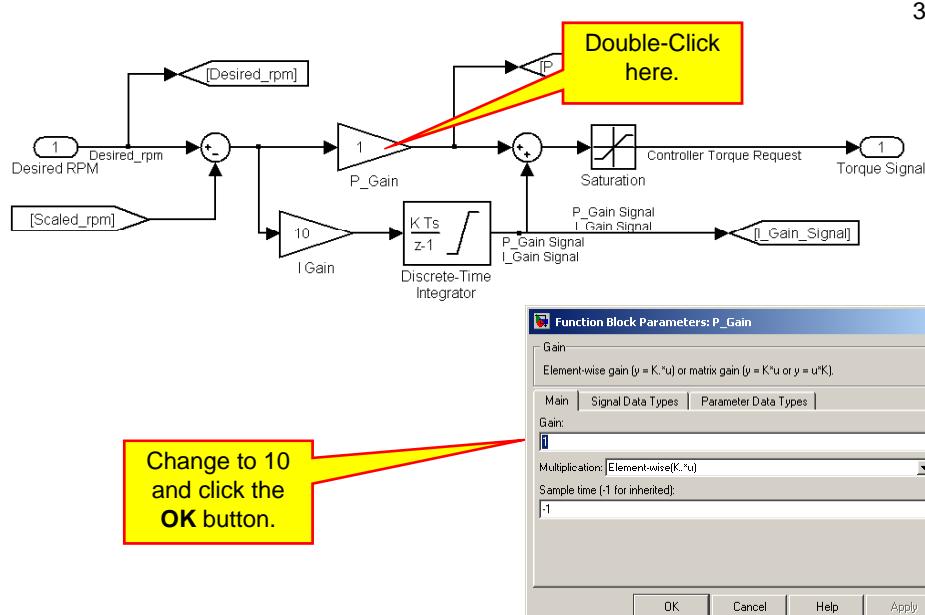


29

## Parameter Tuning

- Next, let's change the proportional gain to 10.
  - Leave the model running.
  - Set both sliders back to zero.
  - Next, double-click on the Proportional Gain block and change its value to 10.
  - Click the **OK** button to accept the changes.

30

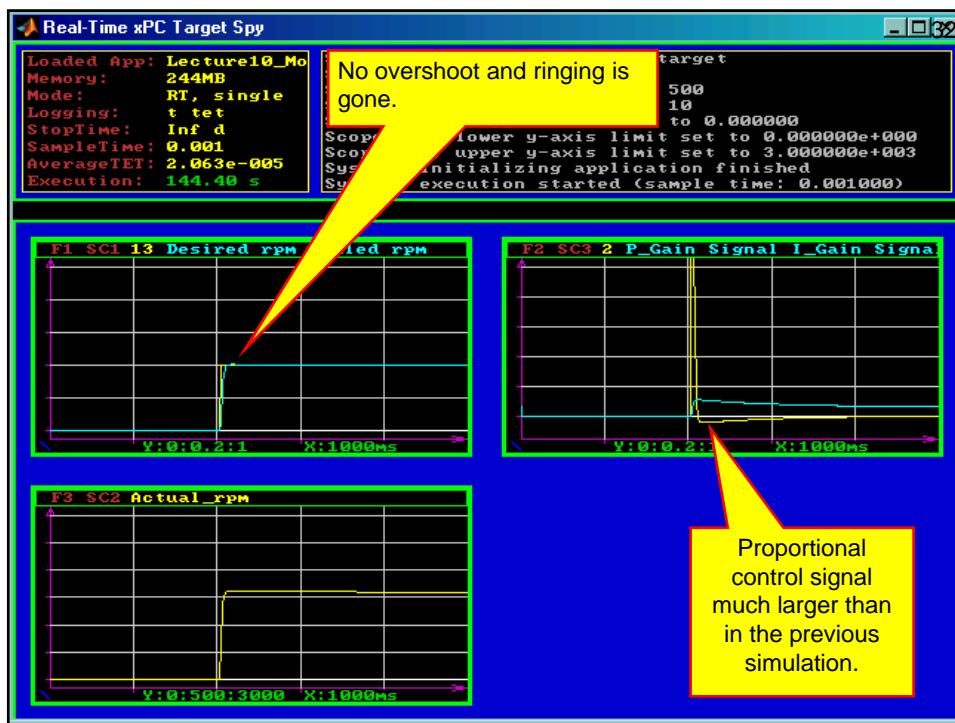




31

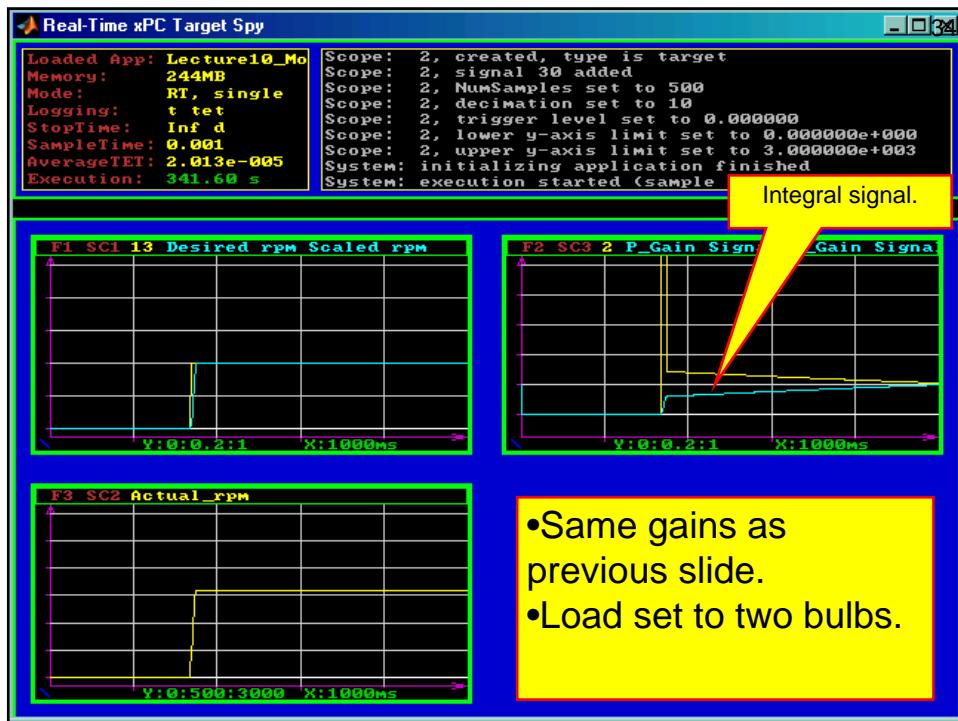
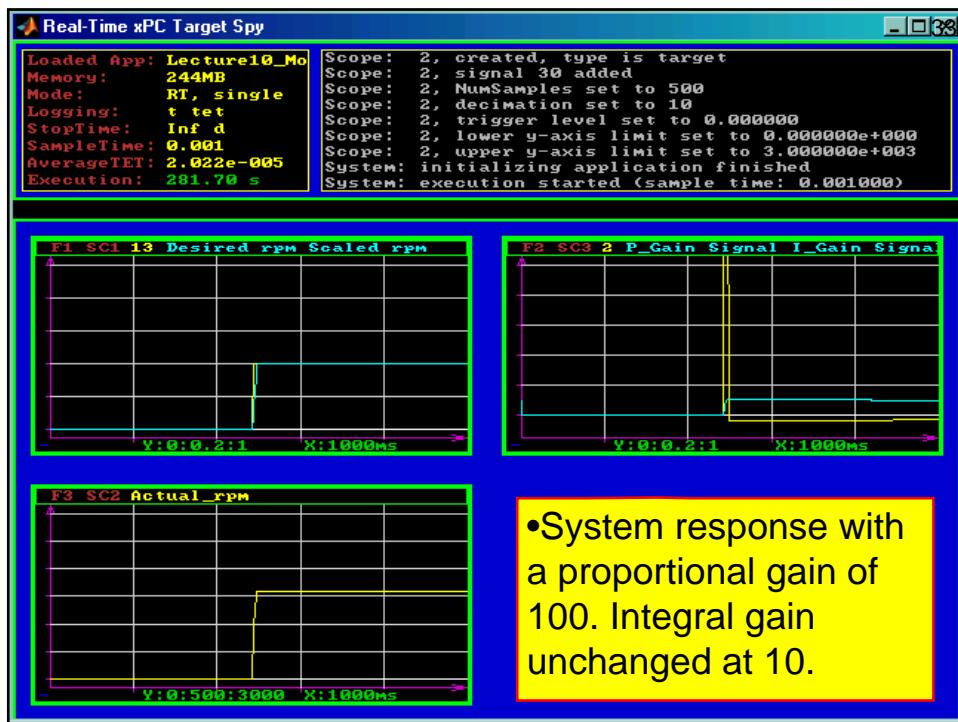
## Parameter Tuning

- Once again, run a step response with no load.
  - Leave the load slider at 0 bulbs.
  - Change the desired speed slider to 0.4 and observe the results.





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





35

## Model Parameter Tuning

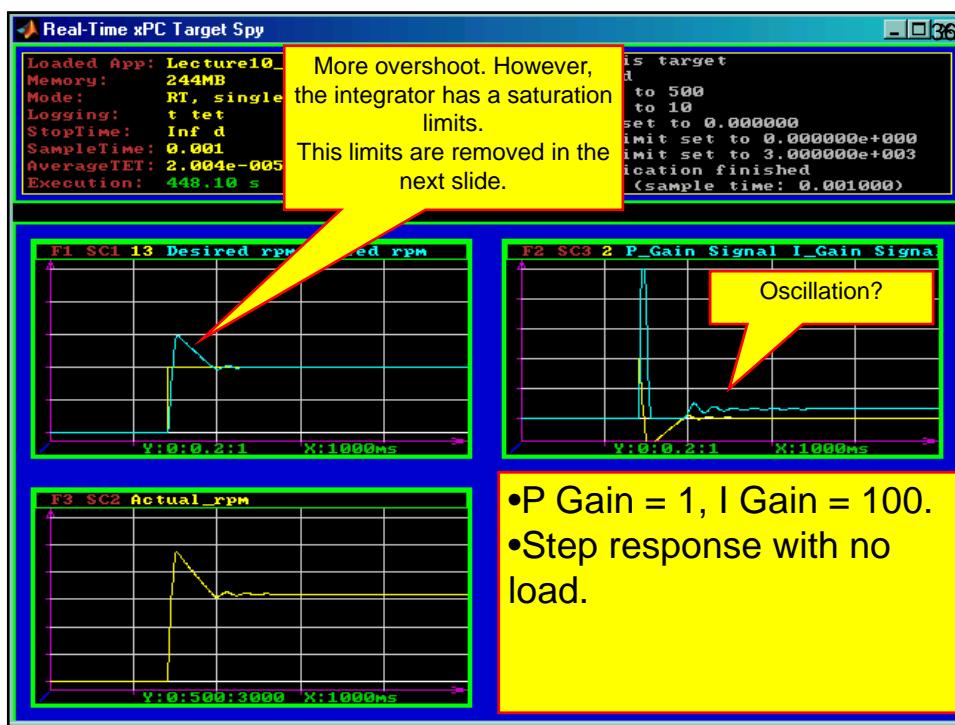
- The model appears to behave very well when we increase the proportional gain.
- For our next experiment, set:
  - Proportional Gain to 1.
  - Integral Gain to 100.
- Run the step response with no load.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



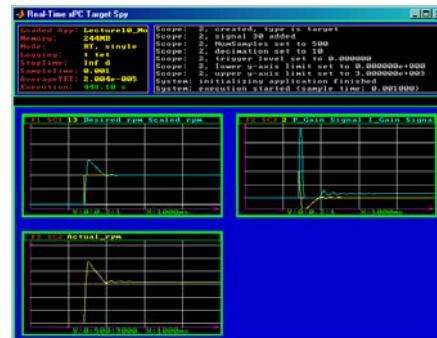


37

## Lecture 10 xPC 3

- Obtain a copy of the next screen using the xpctargetspy command.

**XPCTARGETSPY**

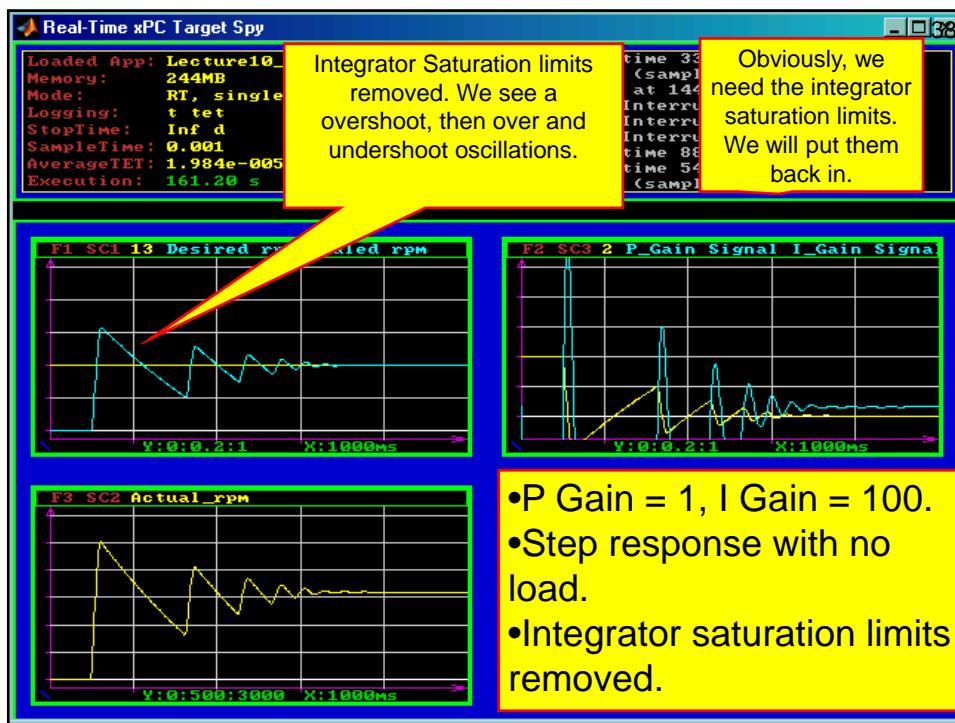


**MotoTron**

**The MathWorks**

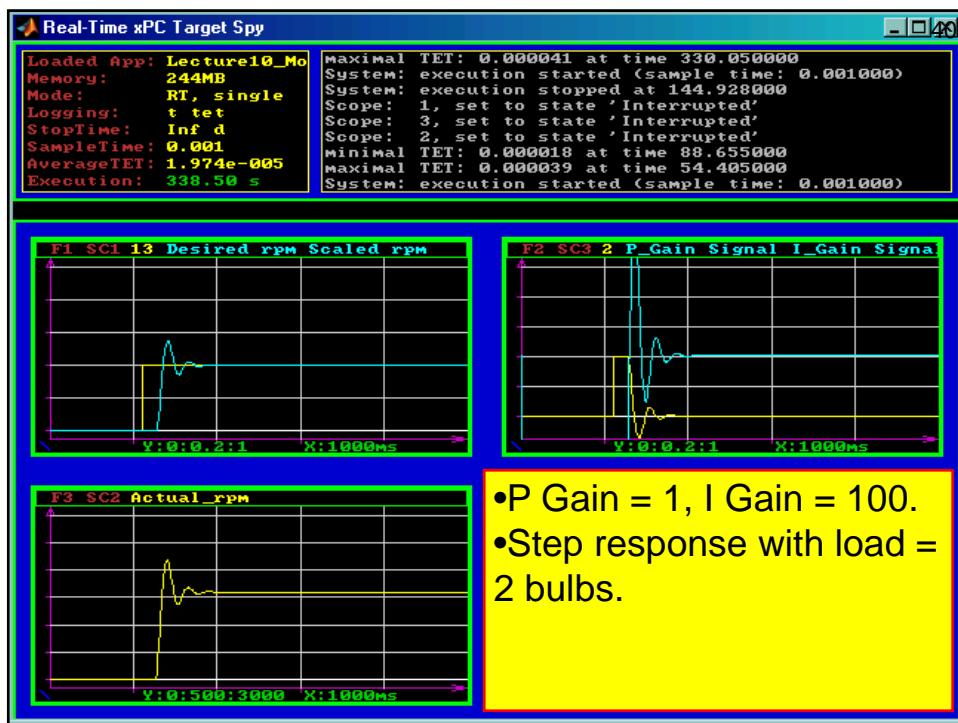
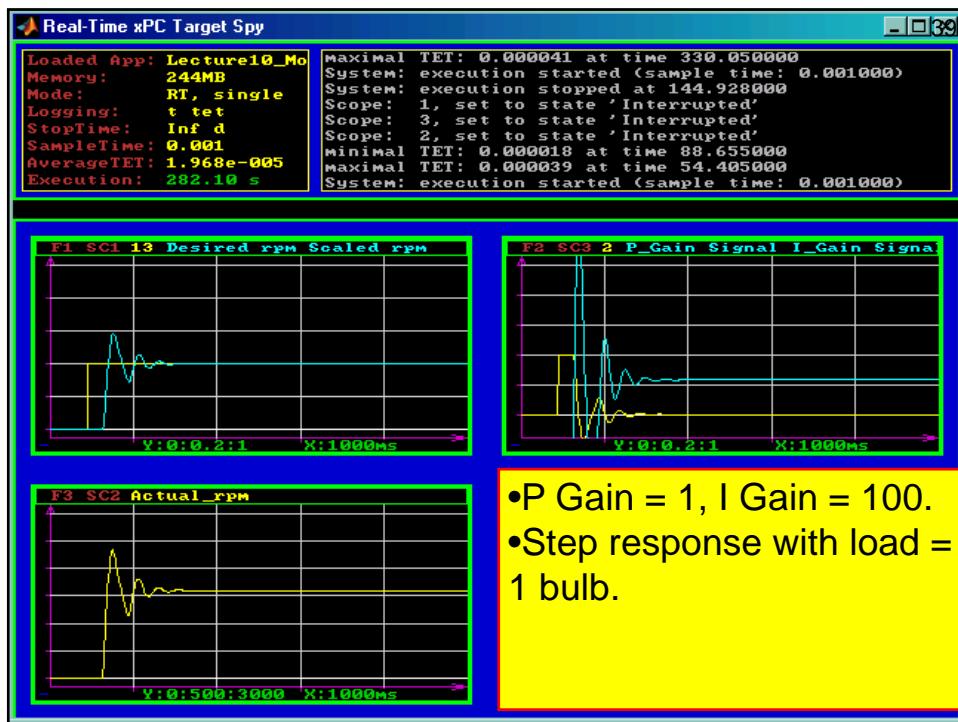
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY





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





## Model Results

41

- From our results we can make the following observations about our physical system.
  - It is a single pole system that can be controlled easily with proportional gain.
  - The integrator causes overshoot and ringing.
  - If we can increase the proportional gain to very large values:
    - The difference between desired and actual speed will become very small.
    - The integrator can be eliminated.



## Notes

42

- We only changed tuning parameters in our controller.
- We can easily use parameter tuning to observe how various plant values affect the operation of the system as well.
- Most plant parameters have tolerance. We can observe the effect of changing plant parameters using the method described here.





## xPC Explorer

43

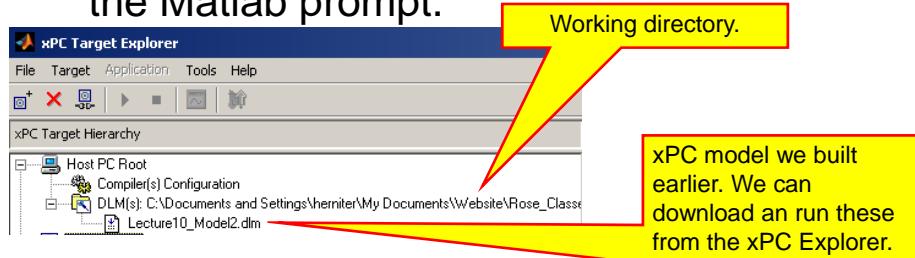
- We will now show a few techniques using the xPC Explorer.
- We will show how to
  - Load a Model
  - Run a Model
  - Tune Parameters
- We will start assuming you just started xPC and Matlab, so
  - Reboot your xPC target and restart Matlab



## xPC Explorer

44

- When Matlab restarts, change to the working directory where our motor control files are located.
- Do not run Simulink.
- Run xPC Explorer by typing **xpcexplr** at the Matlab prompt.

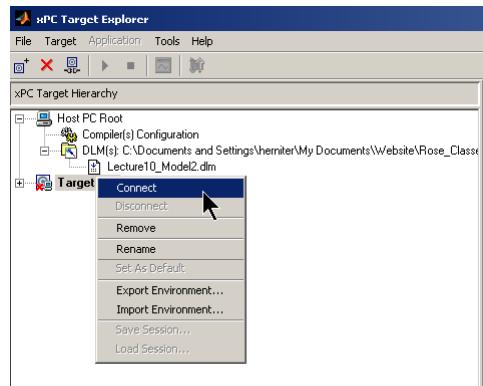




45

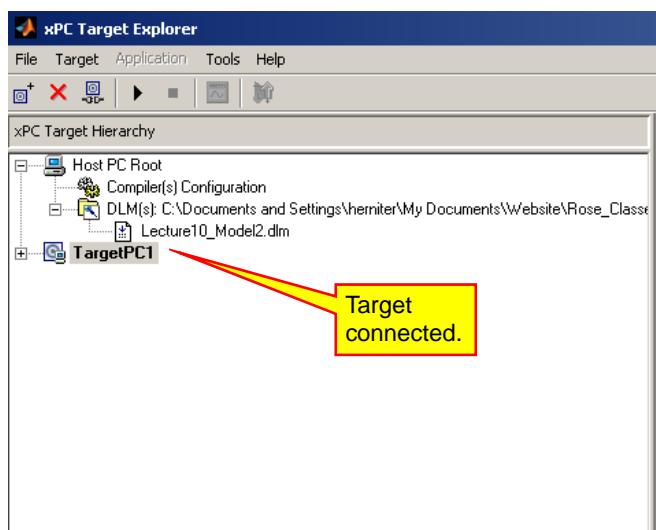
## xPC Explorer

- First, we need to connect to the xPC Target computer.
- Right click on **TargetPC1** and select **Connect**.



46

## xPC Explorer

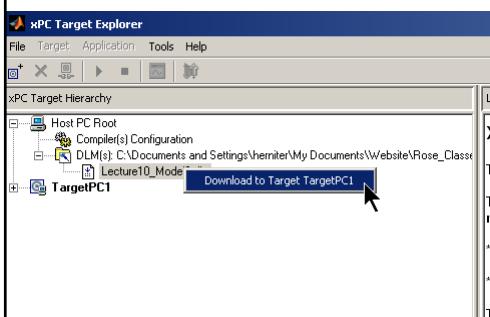




## xPC Explorer

47

- Once connected, we want to download the model with the PI controller.
- Right-click on the model you want to download (Lecture10\_Model2.dlm) and select **Download to Target TargetPC1**:



## xPC Explorer

48

- The target screen should show 3 plots as shown below.

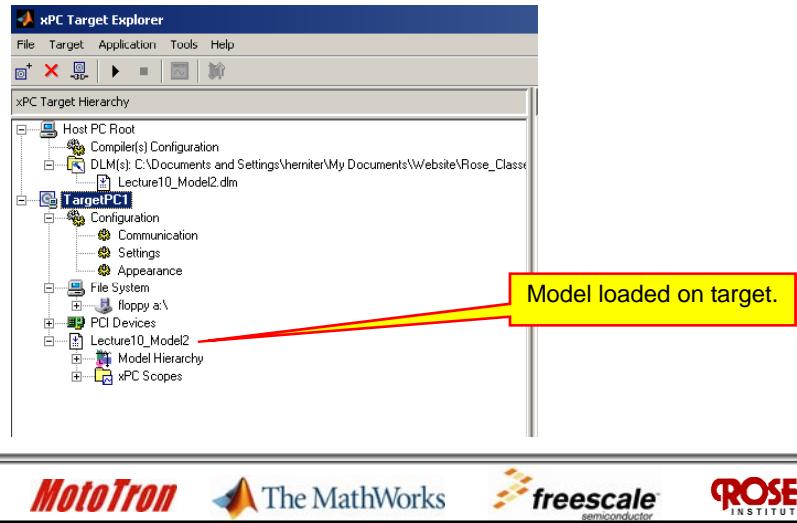




## xPC Explorer

49

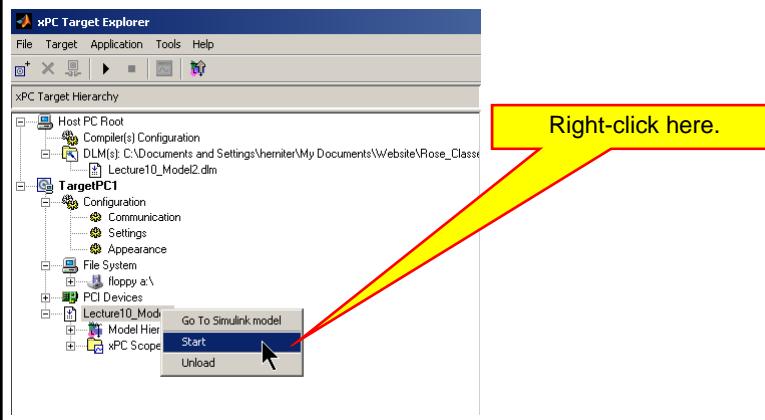
- If you expand the TargetPC1 tree, it should indicate that the model is loaded.

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

## xPC Explorer

50

- To start the model, right-click on Motor\_Control\_PI\_xPC1 in the TargetPC1 tree and select Start:





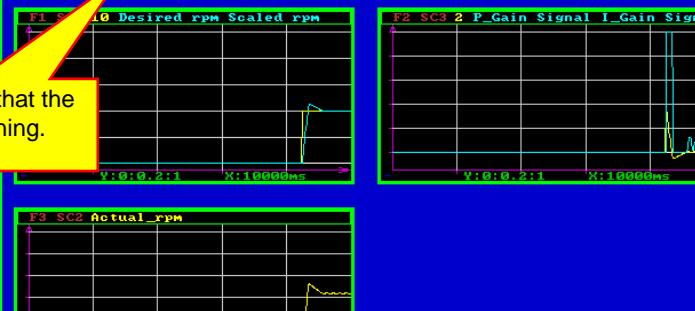
## xPC Explorer

51

- You model should now be running on the target:

```
Loaded App: Lecture10_Mo Scope: 2, TriggerScope set to 1
Memory: 379MB System: initializing application finished
Mode: RT, single Scope: acquisition of scope 1 is running
Logging: t tet Scope: acquisition of scope 1 is running
StopTime: Inf d Scope: acquisition of scope 3 is running
SampleTime: 0.001 Scope: acquisition of scope 3 is running
AverageTET: 2.966e-005 Scope: acquisition of scope 2 is running
Execution: 7.88 s System: execution started (sample time: 0.001000)
```

Time indicates that the model is running.

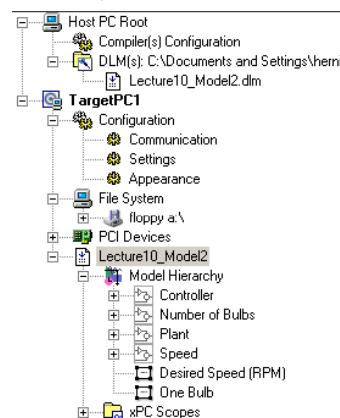


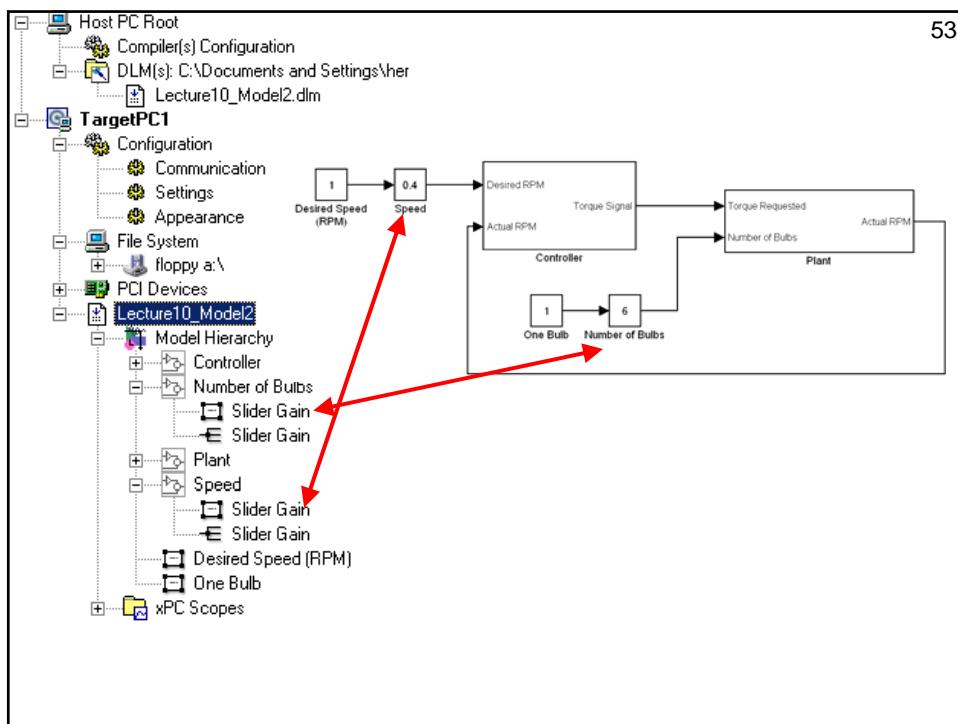
## xPC Explorer

52

- We will now use the xPC Explorer to set the inputs to zero.
- Expand the Model Hierarchy tree as shown:

- The Number of Bulbs branch and the Speed branch are the sliders in the top level of our model.





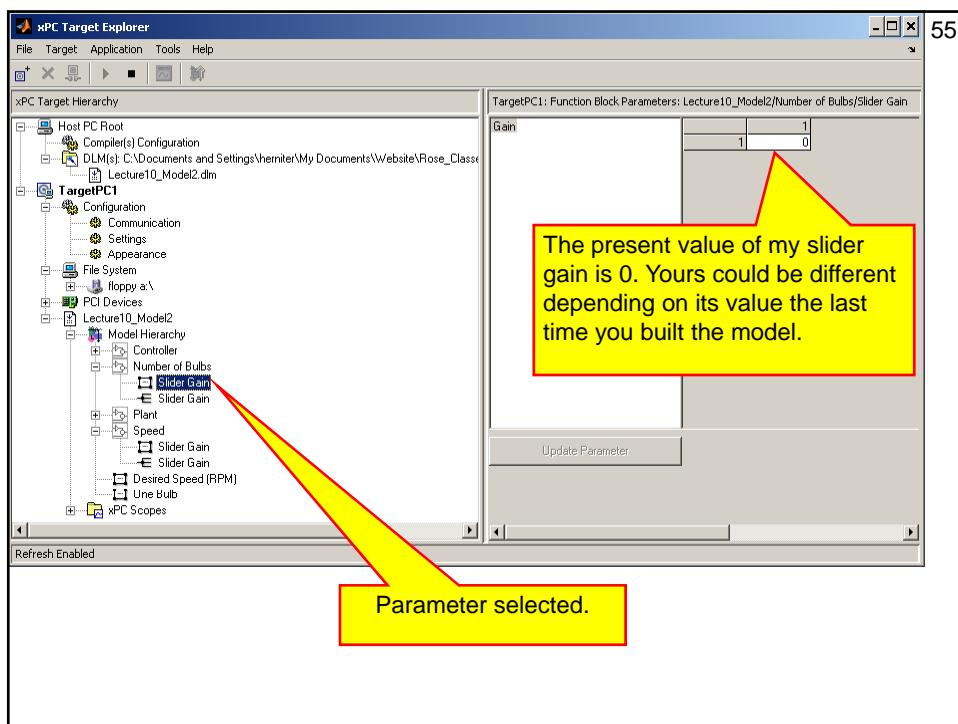
53

## xPC Target

54

- Changing the parameters in the xPC Explorer is the same thing as moving the sliders in the Simulink model when the Simulink model is connected to the target model.
- To see the value of a parameter, and possibly change it, select the parameter.
- First, select the **Slider Gain** parameter in the **Number of Bulbs** system

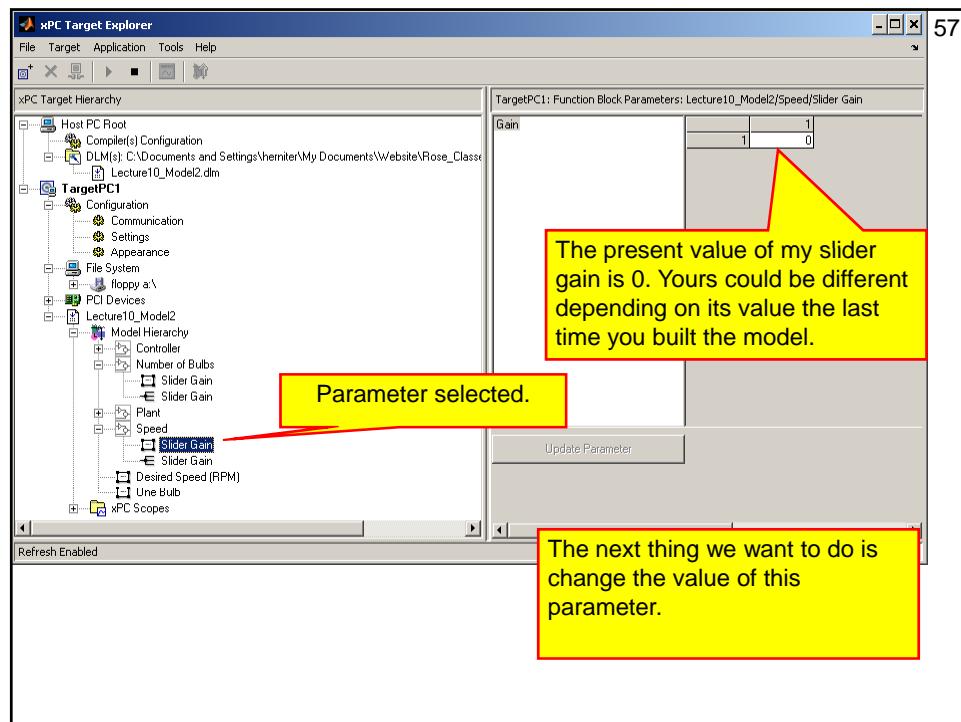




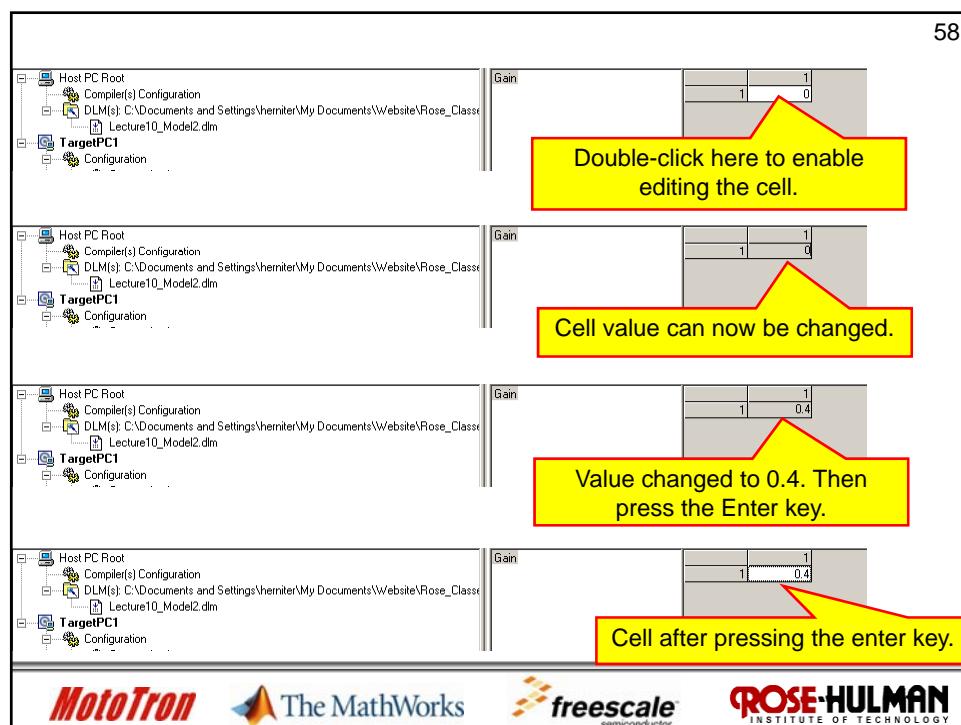
## xPC Explorer

- We will leave the load at zero. If the number of bulbs in your model is not zero, you can follow the procedure shown in the next few slides to change it to zero.
- Next, we will look at the Speed slider gain. Select the parameter to display its value.





57



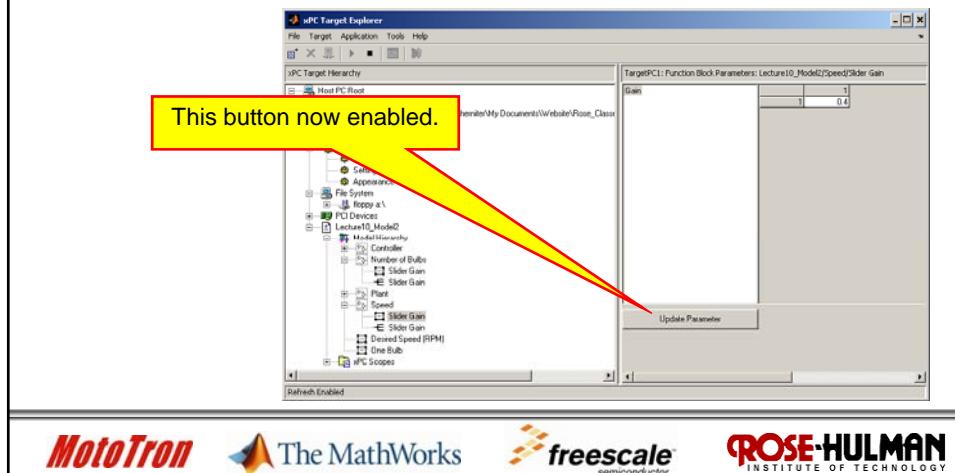
58



## xPC Explorer

59

- After pressing the Enter key, the **Update Parameter** button becomes enabled:

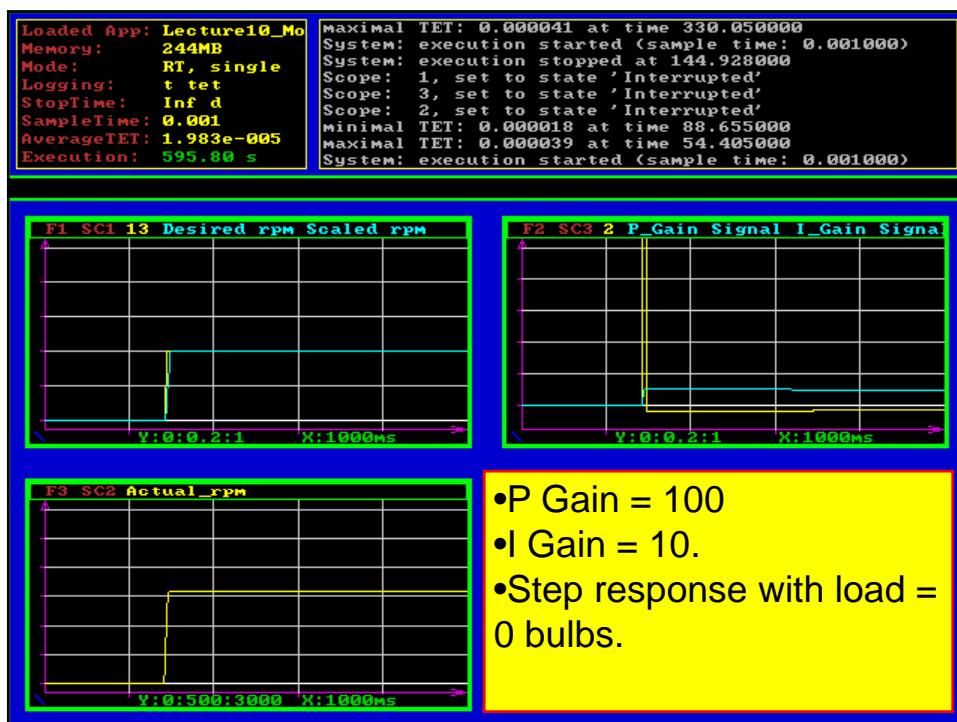
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## xPC Explorer

60

- When you click the **Update Parameter** button, the parameter change will be downloaded to the model on your xPC target.
- Since the model is running, you will see the change in real-time.
- Since the parameter we are changing is the speed, you will see the motor speed change in response to parameter change.
- My xPC Target screen is shown next.

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

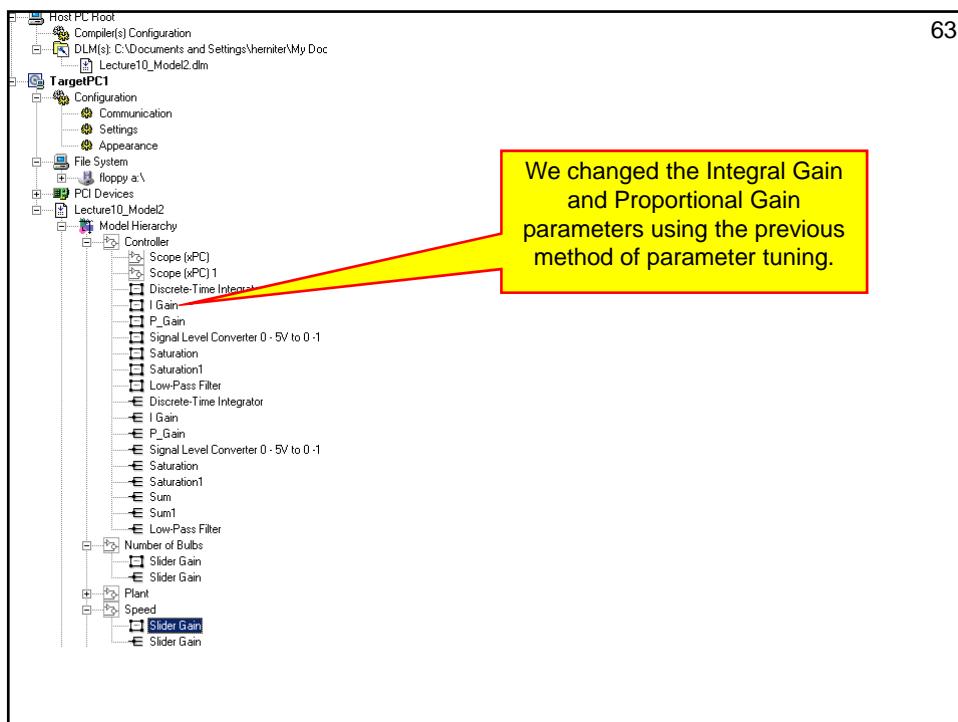


## xPC Target

62

- We can use this method to change any of the changeable parameters in the model.
- You can see all of the available parameters by expanding the **Model Hierarchy** tree.





63

## xPC Target

64

- We will show a few last commands that we can use with the xPC Target Explorer.
- To stop the model running on our xPC Target computer, right click on **Lecture10\_Model2** and select **Stop**

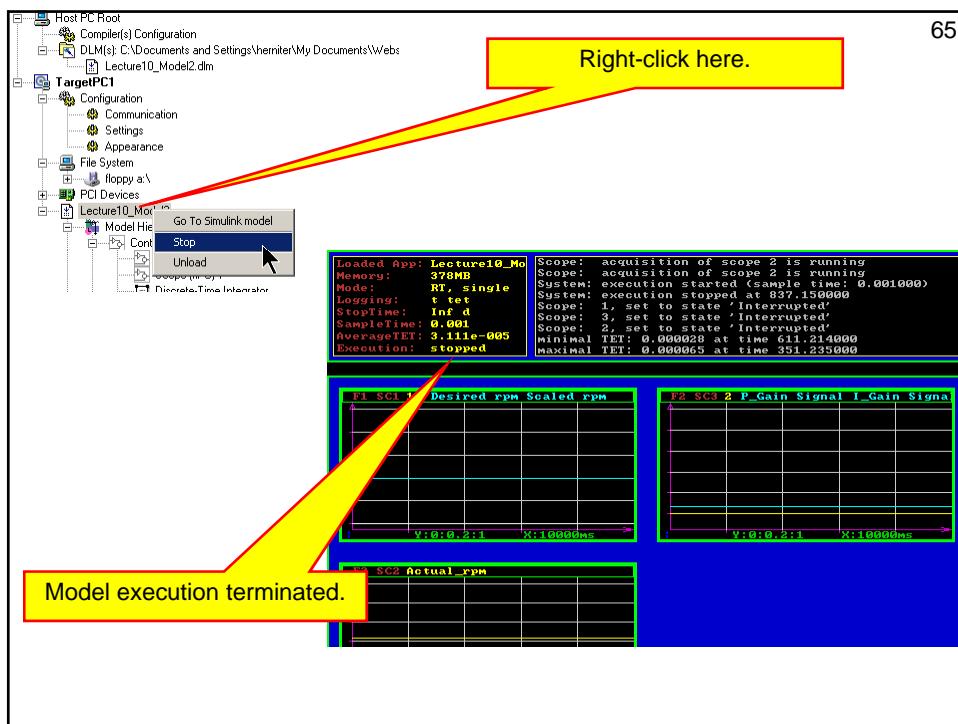


The MathWorks





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

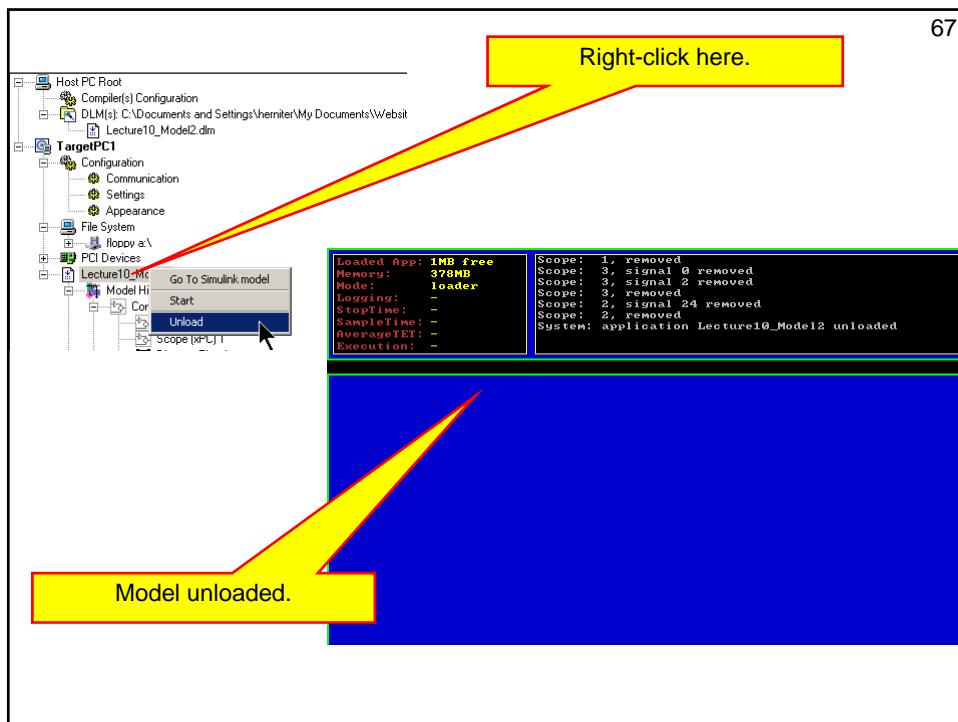


## xPC Explorer

66

- To unload the present model from our xPC Target computer, right click on **Lecture10\_Model2** and select **Unload**





67

## Gauges Blockset

68

- For our last example of this section, we will show how to use the Gauges Blockset toolbox to display the real-time performance of our model.
- Save model Lecture10\_Model2.mdl as Lecture10\_Model3.mdl.
- Make the changes to the top level as shown next.



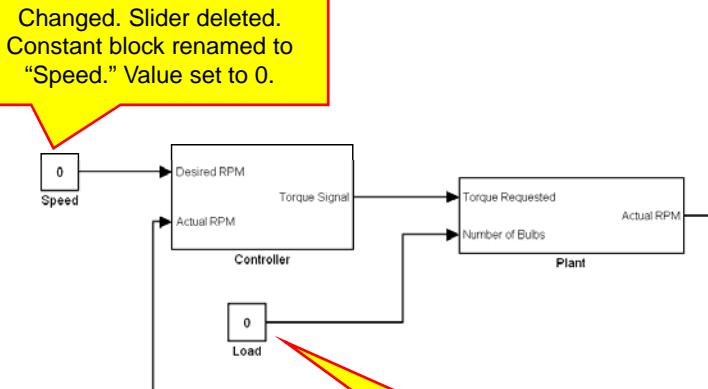
The MathWorks





## Modifications to Model

69



Changed. Slider deleted.  
Constant block renamed to "Load." Value set to 0.

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Gauges Blockset

70

- Open up a new model and name it `Motor_Control_PI_xPC2_Panel.mdl`.
- Place in two copies of the block called `To xPC Target`.
- These are located in the **xPC Target / Misc toolbox**.
- These parts allow us to send data from the front panel to the model running on our xPC target machine.
- Add constants and sliders as shown.

**MotoTron**

The MathWorks

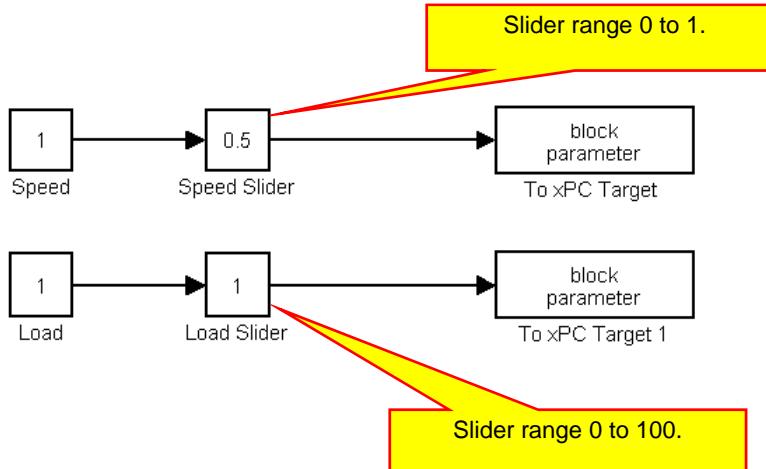
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Gauges Blockset

71



## Gauges Blockset

72

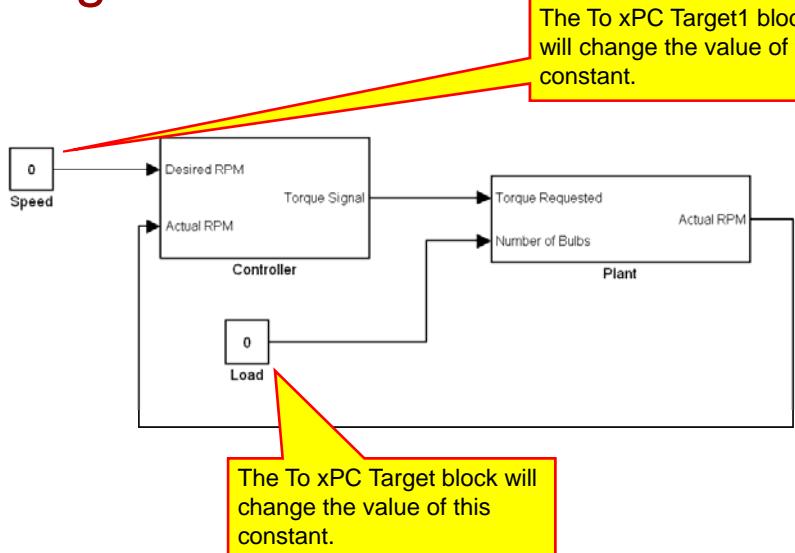
- The To xPC Target blocks allow us to change the value of blocks of a model that is running on an xPC target.
- We will use the To xPC Target blocks to change the value of constants “Load” and “Speed” in model Motor\_Control\_PI\_xPC2.





## Gauges Blockset

73

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

## Gauges Blockset

74

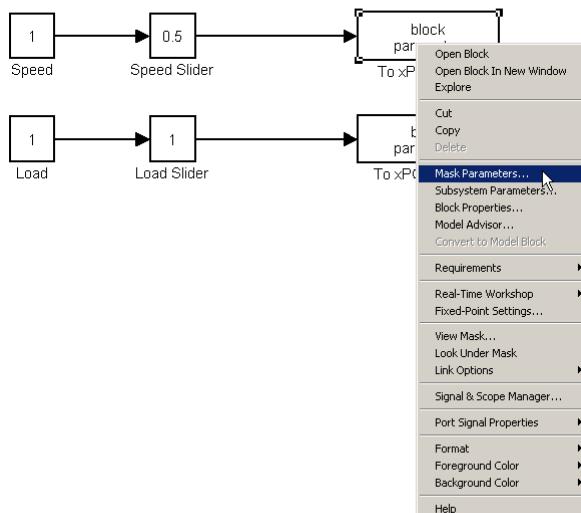
- To change a value, you must know the name of the model (Motor\_Control\_PI\_xPC2) and the name of the blocks (Speed and Load).
- Right-click on block To xPC Target and select **Mask Parameters** from the menu.
- Change the parameters as shown:

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



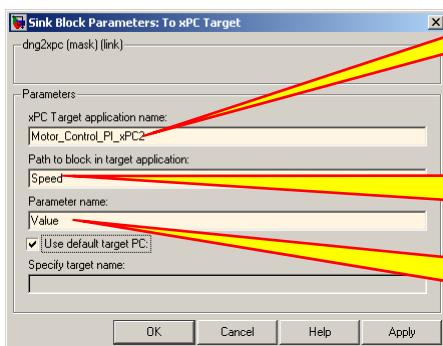
## Gauges Blockset

75



## Gauges Blockset

76



Name of model running on the xPC Target.

Name of the block of which we are going to change the value.

Specifies that we will change the Value of the constant block.



The MathWorks

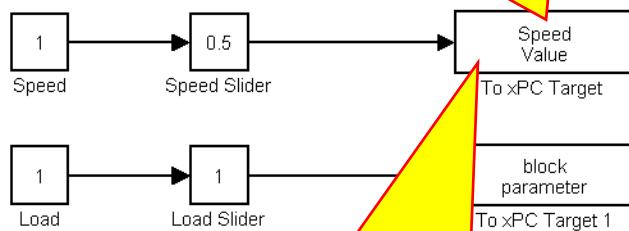




77

## Gauges Blockset

After you click the OK button, your block should appear as shown.



To test the linkage, we can double-click on this block. After double-clicking on this block, the corresponding block in the xPC model should be highlighted in cyan.

**MotoTron**

**The MathWorks**

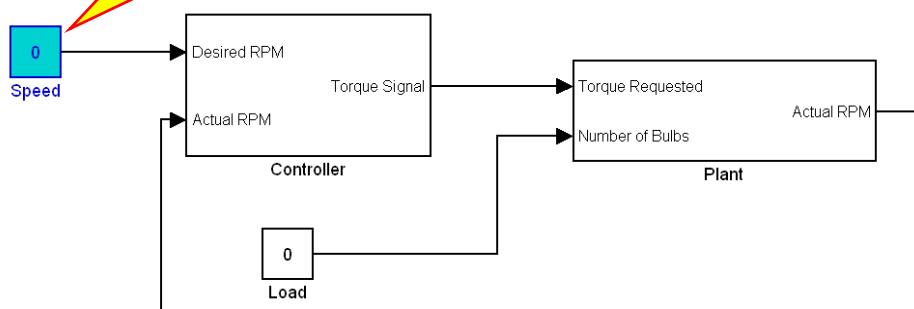
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

78

## Gauges Blockset

This block highlighted in cyan after we double-clicked on the To xPC Target block in the front panel model. This shows that we have successfully linked the panel to the xPC model.

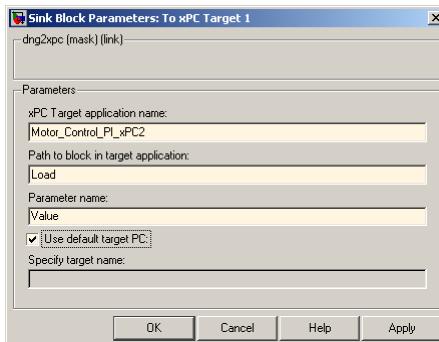




79

## Gauges Blockset

- Repeat the process for block To xPC Target1 and link it to the constant block names “Load” in the xPC Target model.



80

## Gauges Blockset

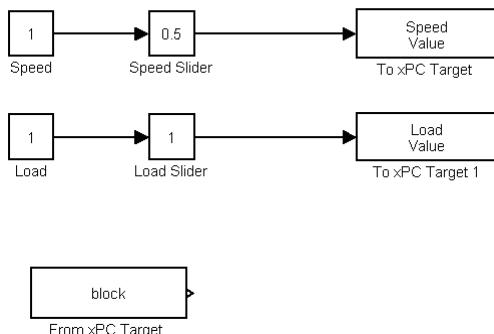
- The next thing we want to do is display the performance of the real-time model on our panel.
- We will use a gauge for this and an LED display.
- To transfer information from your xPC model to the front panel, use the part called From xPC Target, also located in the **xPC Target / Misc** library.
- Place one copy in your panel model.





## Blocks and Gauges

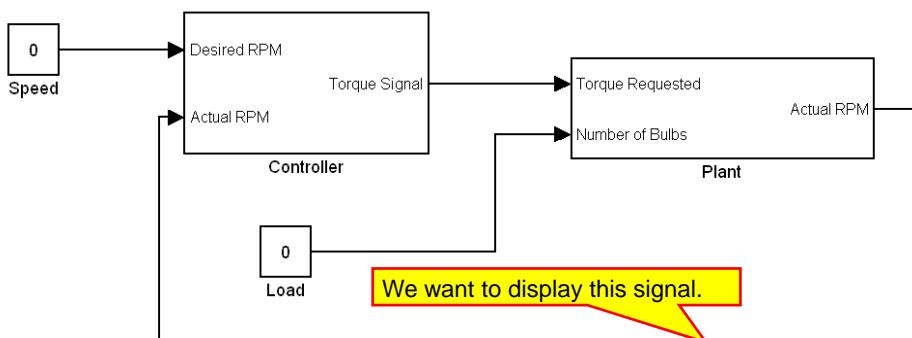
81



## Gauges Blockset

82

- We wish to display the rpm of the motor in the plant.
- This signal is located in the Plant subsystem of the xPC model.

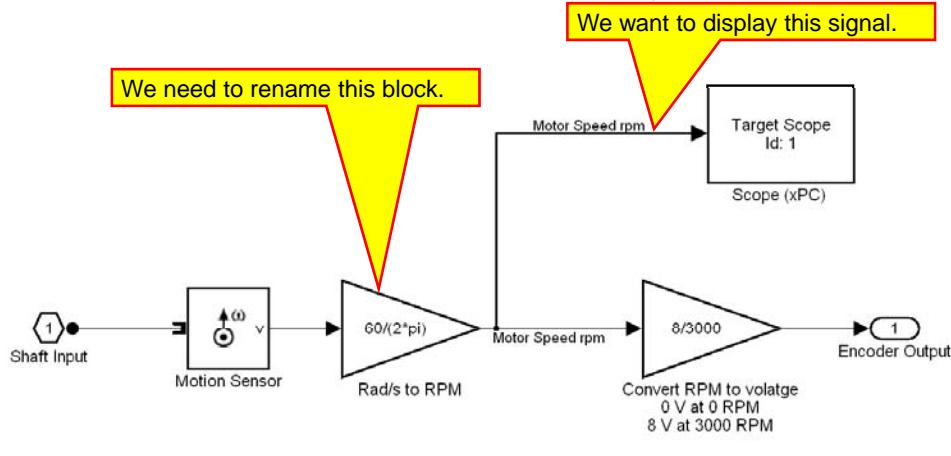




83

## Gauges Blockset

- Open the Shaft Encoder subsystem.



**MotoTron**

The MathWorks

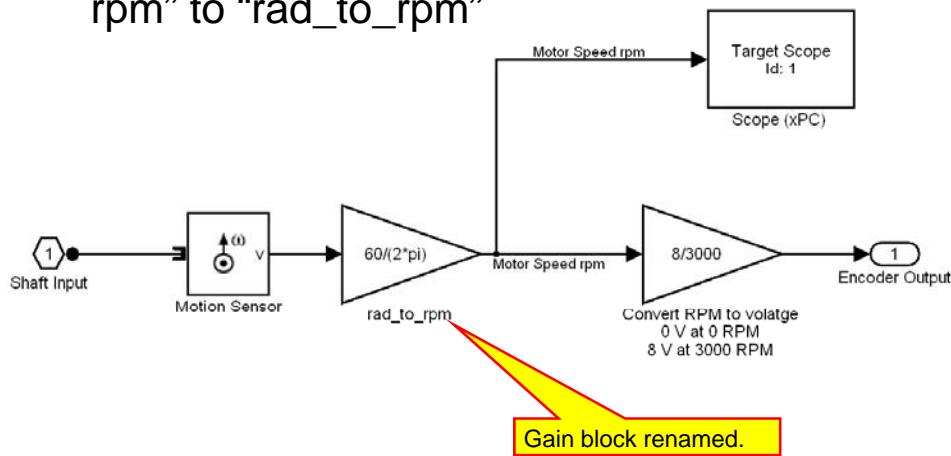
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

84

## Gauges Blockset

- Rename the gain block from “rad/sec to rpm” to “rad\_to\_rpm”



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

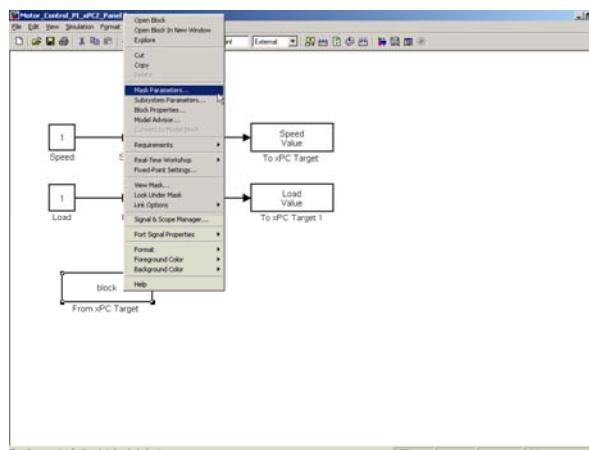


85

## Gauges Blockset

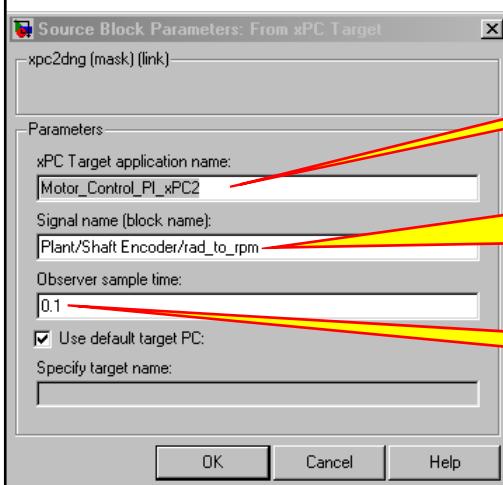
- Right-click on the From xPC Target block in the Panel model and the select **Mask Parameters** from the menu:

- Fill in the dialog box as shown next.



86

## Gauges Blockset



Model name  
Motor\_Control\_PI\_xPC2.

Block name  
Plant/Shft\_Encoder/rad\_to\_rpm.  
Note that the block rad\_to\_rpm is located inside the Shaft Encoder subsystem.

Get a data point every 0.1 seconds.

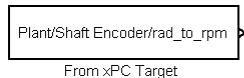
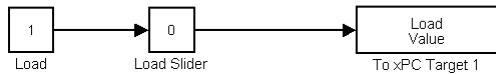
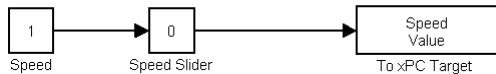
Click the OK button to accept the settings and then display the panel model.





## Gauges Blockset

87

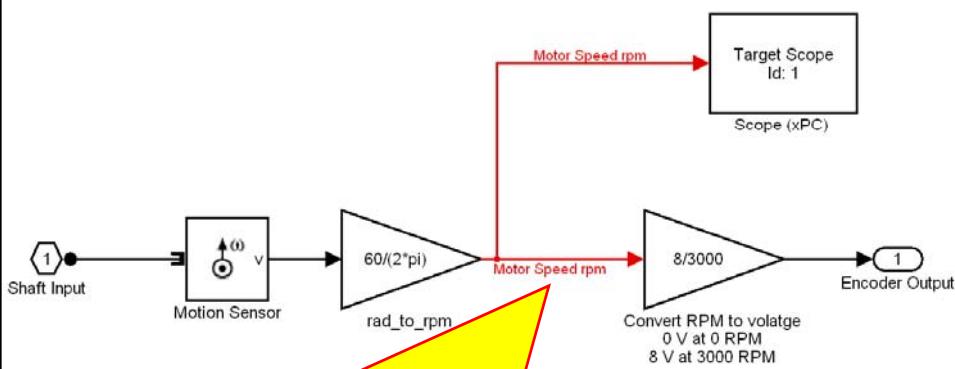


To test the linkage, double-click on this block. The corresponding signal should be highlighted in the xPC model.



## Gauges Blockset

88



This signal highlighted in red after we double-clicked on the From xPC Target block in the front panel model. This shows that we have successfully linked the panel to the xPC model.





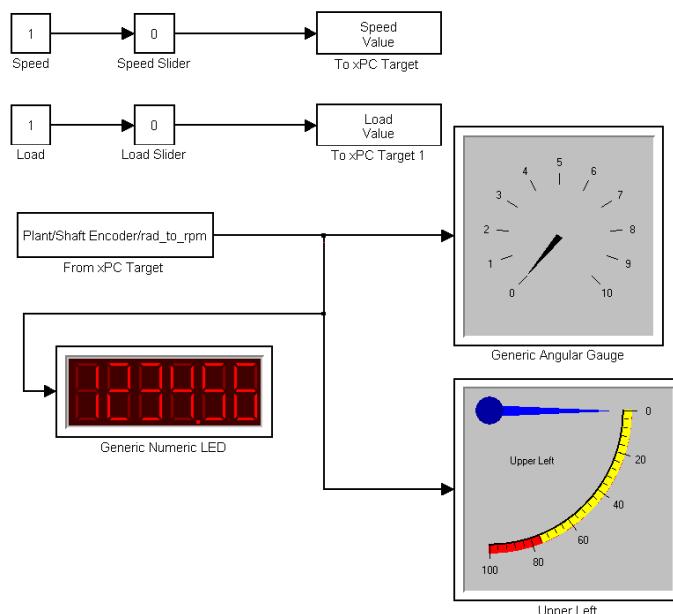
89

## Gauges Blockset

- The last thing we want to do is display the rpm using some of the blocks from the Gauges Blockset toolbox.
- Place the following blocks in your model
  - Generic Angular Gauge (Angular Gauges)
  - Upper Left (Angular Gauges)
  - Generic Numeric LED (Numeric Displays)

90

## Gauges Blockset



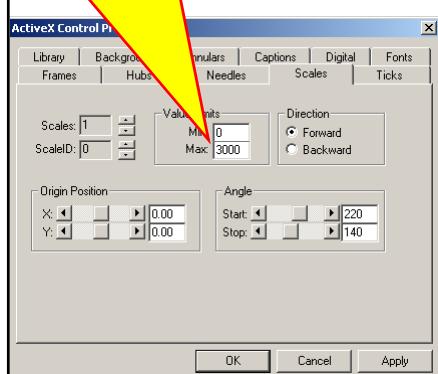


91

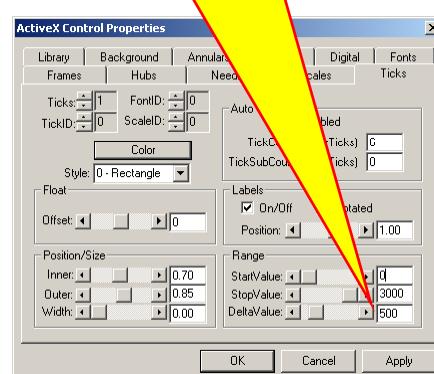
## Gauges Blockset

- Double-click on the Generic Angular Gauge block and change the values as shown:

Max value set to 3000. (Note Scales tab selected.)



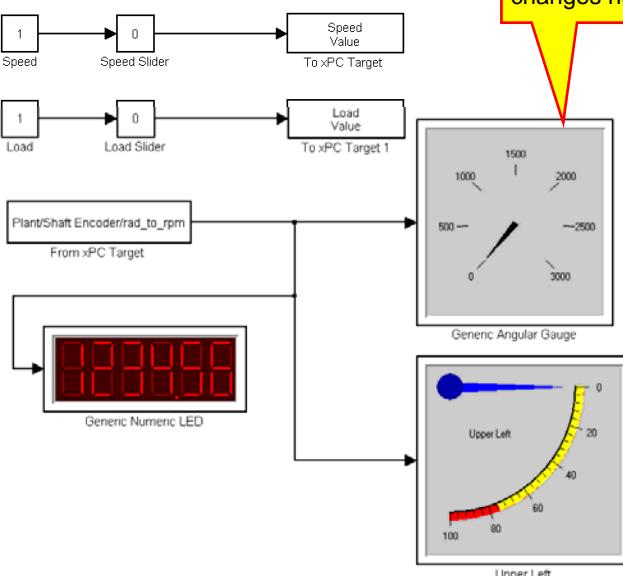
Delta value set to 500. (Note Ticks tab selected.)



## Gauges Blockset

92

Note display changes here.

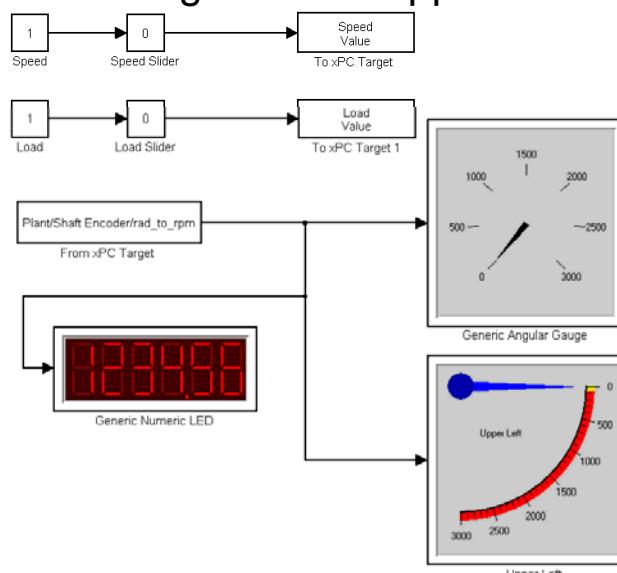




93

## Gauges Blockset

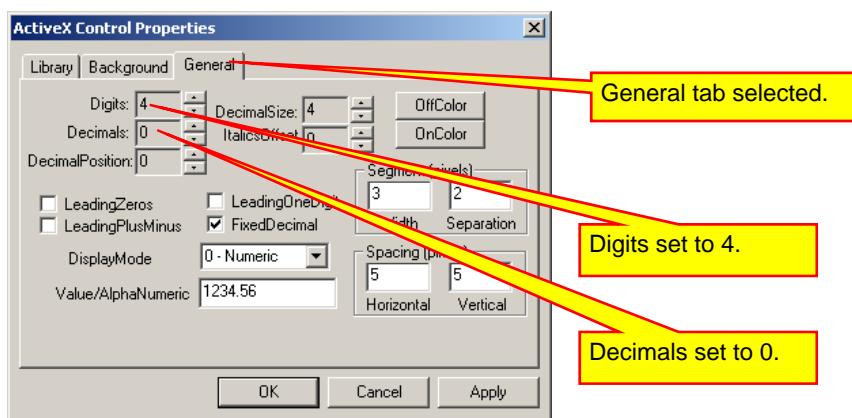
- Make the same changes to the Upper Left gauge.



94

## Gauges Blockset

- Double-click on the Generic Numeric LED block and make the following changes:

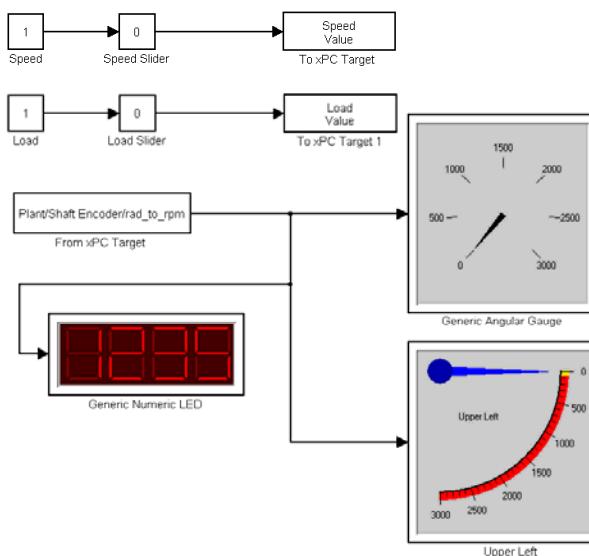




95

## Gauges Blockset

- Make the same changes to the Odometer block.



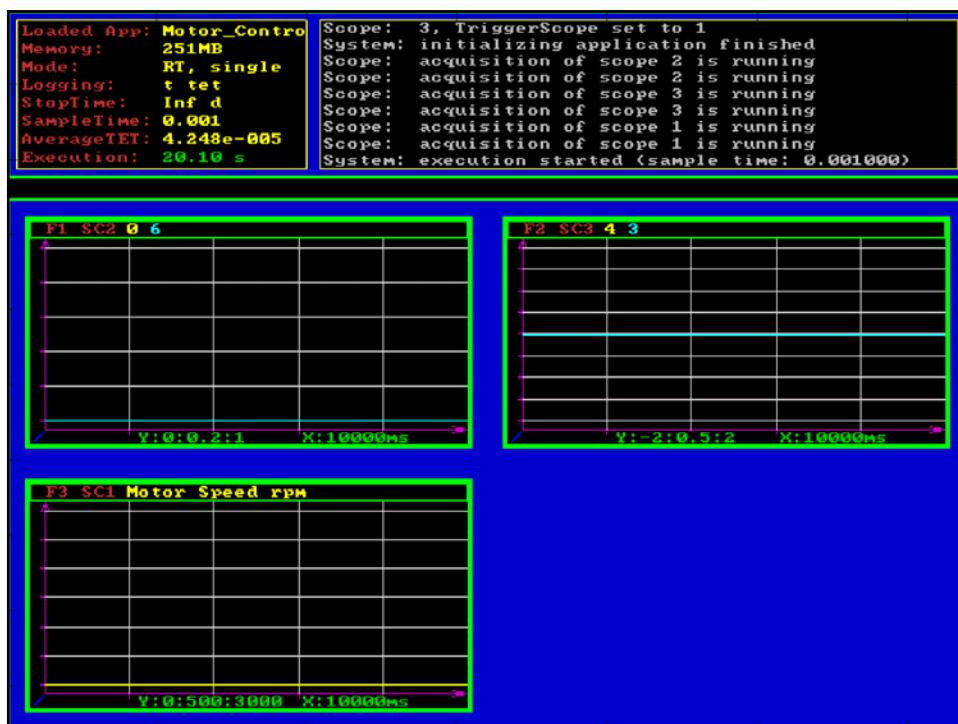
96

## Gauges Blockset

- We are now ready to run the models.
- Compile and load model  
**Motor\_Control\_PI\_xPC2** on to the remote target by typing **ctrl-b** in the model window.
- Use the xPC Explorer to run the model on the remote target.
- You should see the three simulation graphs on your remote target as before:



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

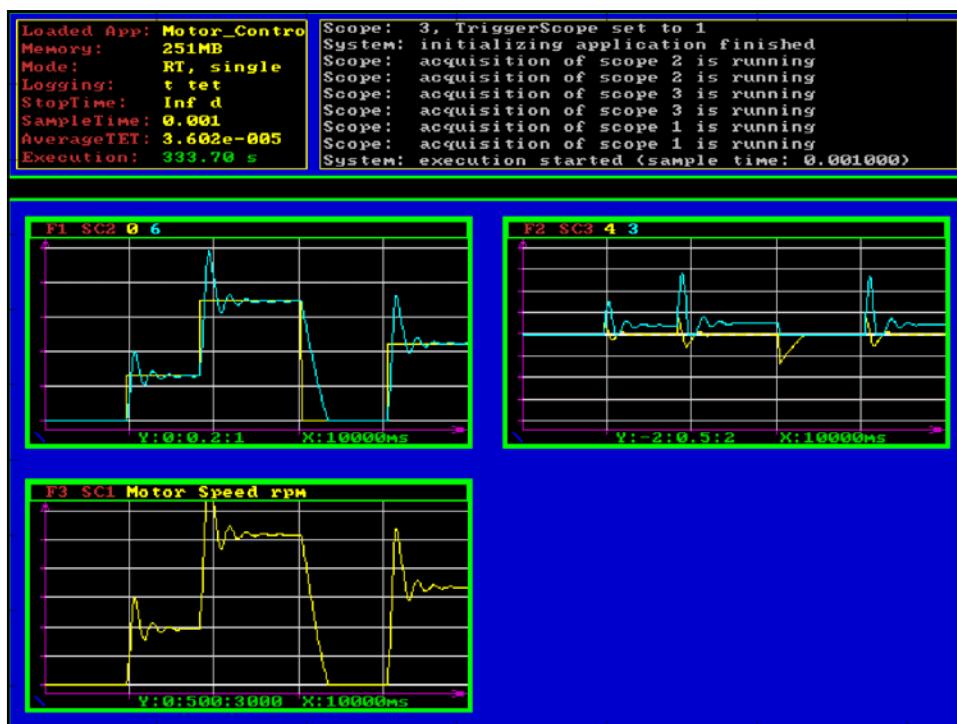


## Gauges Blockset

98

- Next, switch back to your panel model (Motor\_Control\_PI\_xPC2\_Panel.mdl) and run it.
  - Set the simulation type to **Normal**.
- As you change the sliders, you should see the plots on the remote target respond accordingly.

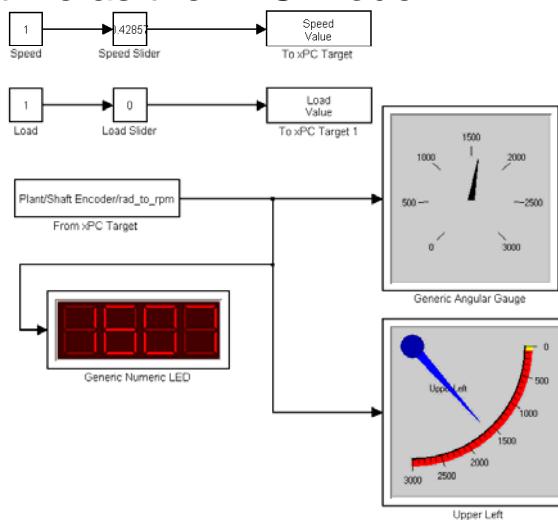




## Gauges Blockset

100

- The Gauges and LEDs should also change in real-time as the xPC model runs.





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

101

## Lecture 10 Gauges - Demo

- Demonstrate your working model to me.
  - Front Panel Display
  - xPC Real-Time Display

Demo\_\_\_\_\_



The MathWorks



freescale<sup>®</sup>  
semiconductor



Any Questions?



The MathWorks



freescale<sup>®</sup>  
semiconductor





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



## Introduction to Model-Based Systems Design

### Lecture 10a: Atomic Subsystems and Software-in-the-Loop (SIL)



## Outline

2

- In this lecture we will take a look at two techniques that will help verify the operation of the controller subsystem and bring it closer to the implementation that will be deployed on the target microcontroller.
- With an atomic subsystem, we can specify the order of execution of the blocks in the controller subsystem and give it a different simulation step size than the plant model.
- With SIL, we will compile the controller into the same c-code that will be deployed on the target, and execute the controller subsystem as C-code rather than Simulink blocks.





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

3

## Models

- We will start with Model 2 from lecture 10.
- We will remove all of the xPC target scope blocks and change the solver back to a variable step solver with a maximum step size of 0.001 seconds.
- The Real-Time Workshop System target file has been change from xpctarget.tlc to grt.tlc, the default target file for new models. (Pretending that we just created this model from scratch.....)
- The simulation mode has been set to normal.
- This model has been saved as Lecture10a\_Model0 and has been passed out as a starting point.



4

## Virtual Subsystems

- In a standard model, with non-atomic subsystems (referred to as virtual subsystems), the blocks are executed when the input data for the blocks is available.
- That is, for a given time step, a few but not all blocks in subsystem “a” could execute, then a few but not all blocks in subsystem “b” could execute, then a few but not all blocks in subsystem “c” could execute.
- Then, execution could go back to subsystem “a” and execute the remainder of the blocks in the subsystem.
- The Simulink engine sorts blocks such that data needed by a particular block is always calculated prior to the execution of the block.





5

## Virtual Subsystems

- In general, some of the blocks in the controller will execute, then some blocks in the plant, then some blocks in the controller, then some in the plant, and so on, until all the blocks have executed for a given time step.
- This is not how the controller would execute in the physical system, where all of the controller code will execute in a specific order, execute completely, and there are no blocks in the controller because it is the physical system.



6

## Atomic Subsystems

- In an atomic subsystem, when a subsystem is executed, all blocks in the subsystem run before execution continues outside the subsystem. (This is closer to how the controller will execute in the physical system.)
- Thus, specifying a subsystem as atomic causes all blocks within a subsystem to execute as a group.

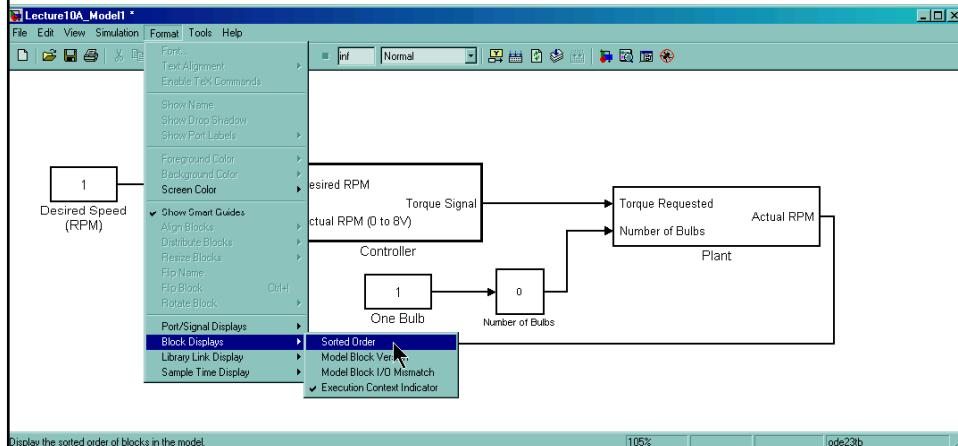




## Block Execution Order

7

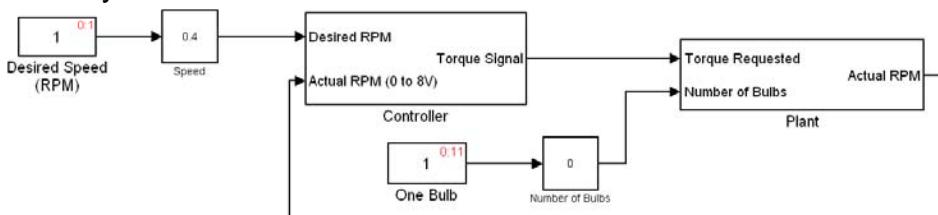
- Before we change the controller to an atomic subsystem, we will look at the execution order of our model.
- To display the block execution order, select **Format**, **Block Displays**, and then **Sorted Order** from the Simulink menus:



## Block Execution Order

8

- The block diagram now shows the execution order in the form  $x:y$  where  $x$  is the system level and  $y$  is the sorted order within the system level  $x$ .



- The Desired Speed (RPM) constant block is at level 0 and the second block executed in the system.
- Constant block One Bulb is at level 0 and is the 12<sup>th</sup> block executed.

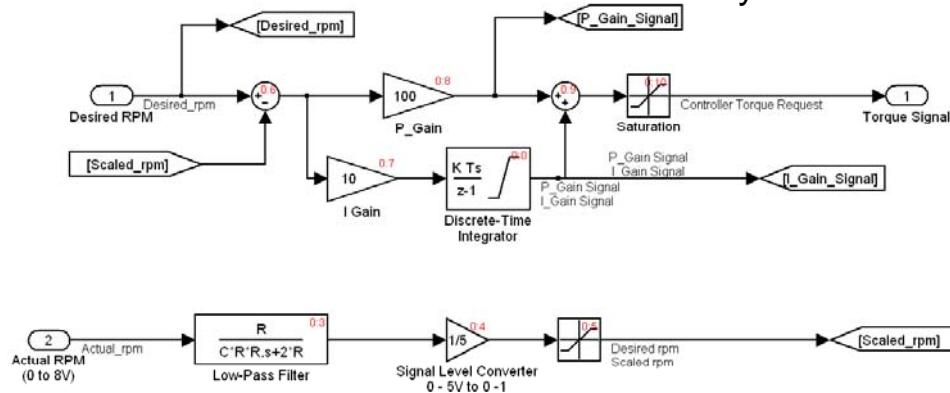




9

## Controller Subsystem

- The Sorted Order of the controller subsystem is:



- The blocks are mostly executed in order from 4<sup>th</sup> to 11<sup>th</sup>.

**MotoTron**

**The MathWorks**

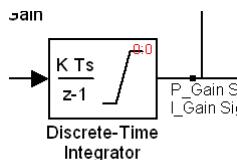
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

10

## Controller Subsystem

- We see that the integrator sorted order is 0:0 meaning that it is at level 0 and the first block executed in the model.



- This is because the discrete integrator block defaults to a forward Euler method which doesn't require the current input, only the input from the previous time step.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



11

## Controller Subsystem

- We see that the integrator in the controller is executed first.
- Then the constant Desired Speed (RPM) at the top level,
- Then another unlisted block (probably the speed slider)
- Then the remainder of the blocks in the controller subsystem.



12

## Sorted Order

- We see that a block in the controller is executed, then blocks outside the controller are executed, then the remaining blocks inside the controller are executed, then execution continues with the remainder of blocks outside the controller.
- This order of execution is not the same as would occur in the physical implementation of the controller.





13

## Sorted Order

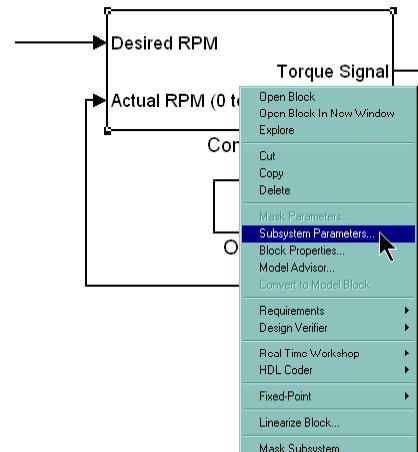
- We also see that all blocks in the model are at level zero (0:x) independent of the subsystem in which they reside.
- This is because virtual subsystems (ones that are not atomic or triggered – the ones we have been using so far) are graphical conveniences for the user.
- Simulink sees all of the virtual subsystems at the same level. (Level zero in this case.)



14

## Atomic Subsystems

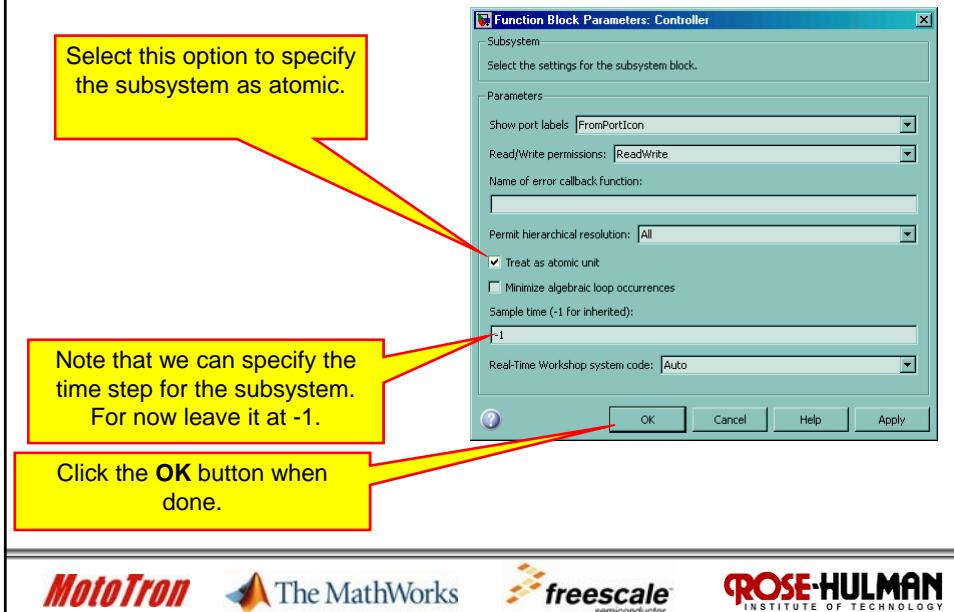
- Next, we will define the controller as an atomic subsystem.
- Return to the top-level of the model and right-click on the controller subsystem, then select **Subsystem Parameters**





## Atomic Subsystems

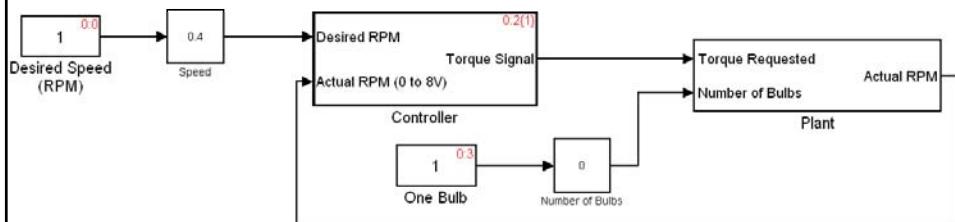
15

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

## Atomic Subsystems

16

- After you type ctrl-d to update the model, you will see the display below:



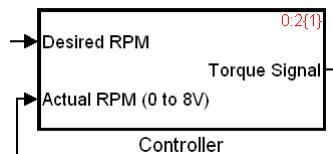
- We see that the One Bulb constant is now executed 4<sup>th</sup> where before it was executed 12<sup>th</sup>.

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



17

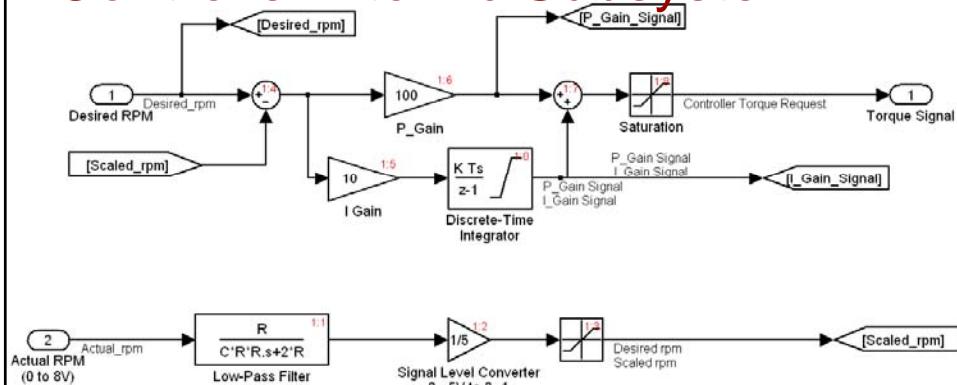
## Atomic Subsystems



- We see that the notation for the Controller subsystem is now 0:2{1}.
- This means that the subsystem block itself is at level 0 and is the 3<sup>rd</sup> block executed in the model.
- The contents of the Controller block are now designated as level 1.
- When the controller block executes as the third block in the model, all blocks within the controller will be executed before any other blocks outside the controller run.

18

## Controller Atomic Subsystem



- All blocks have notation 1:x indicating that they are at level 1.
- Blocks will execute starting with the integrator 1:0 and ending with the Saturation block (1:8)



19

## Controller Atomic Subsystem

- The blocks within the Controller atomic subsystem execute as a group.
- Some blocks outside the controller will execute before, and some will execute after.
- Once the calculations for the Controller begin, because we specified the Controller as Atomic, all blocks will execute as a group before execution continues outside the Controller.
- This order of execution is closer to how we would expect execution of code to occur when we deploy the controller on a hardware target.



20

## Atomic Subsystems

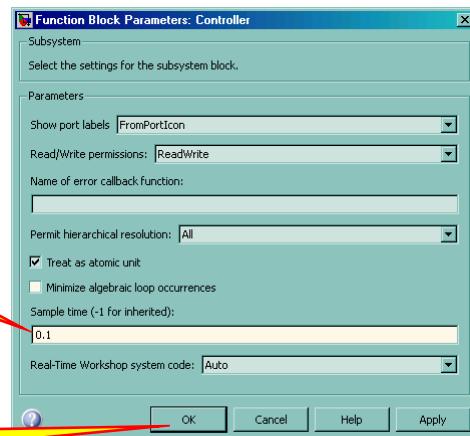
- With an atomic subsystem, we can specify that the subsystem execute with a different time step than the plant.
- For the plant, we will use a small time step so that we simulate the plant accurately.
- For the controller, we will try to see how large a time step we can use and still obtain a stable system.
- Right-click on the Controller subsystem and select **Subsystem Parameters**:





21

## Atomic Subsystems



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

22

## Simulation Configuration Parameters

- Select **Simulation** and the **Configuration Parameters** to view the simulation options.
- We see that the a variable step solver is specified with a max step size of 0.001 (1e-3).

**MotoTron**

The MathWorks

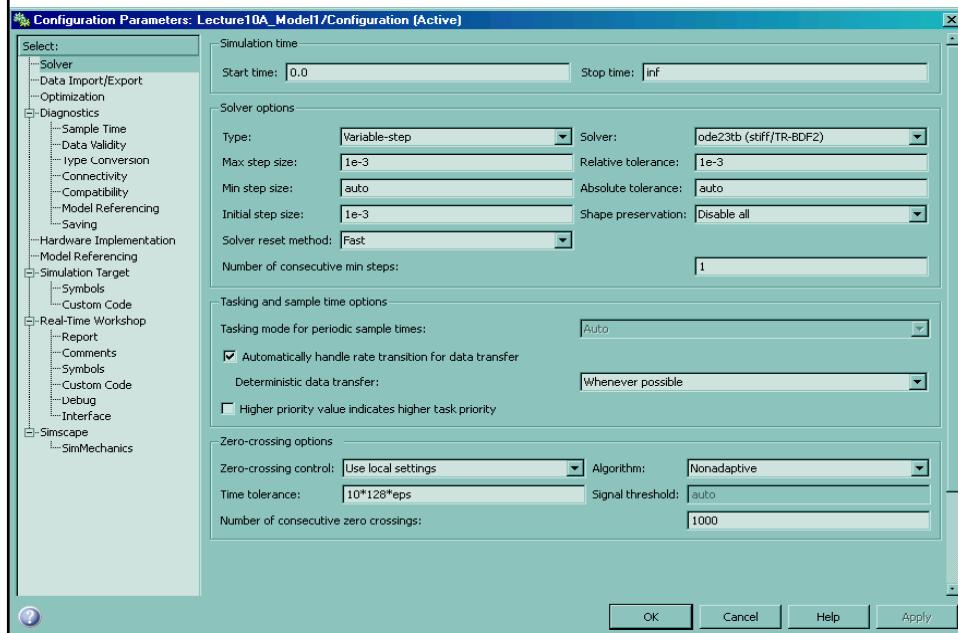
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Simulation Configuration

23



## Atomic Subsystems

24

- The atomic subsystem will allow us to run the plant with a small step size to simulate the physics of the plant accurately.
- The atomic subsystem will allow us to use a large fixed step size for the controller so that we can determine the fixed step size needed for the controller target hardware.
- Before we continue, because the controller now uses a fixed sample time, we need to make some changes to the controller.





25

## Discrete-Time Integrator

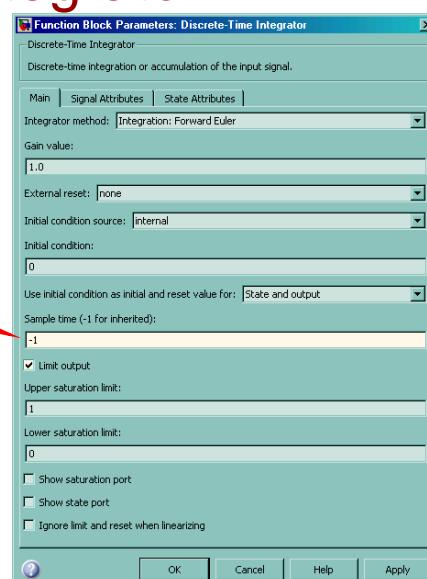
- Since the sample time of the controller is now specified as 0.1, the discrete-time integrator within the controller will use a sample time of 0.1 if we specify its Sample Time as inherited:



26

## Discrete-Time Integrator

Sample Time of the Discrete- Time integrator specified as inherited.





## Low-Pass Filter

27

- The low pass filter placed inside the controller is a continuous time block and cannot run inside an atomic subsystem with a fixed time step.
- In the physical implementation, the low-pass filter is an analog circuit that filters the input signal to the controller, and is not part of the control algorithm implemented by the microcontroller.
- The controller is not part of the plant or the controller, but a signal conditioning circuit between the plant and controller.
- We will move the low-pass filter outside of the controller and place it in the top level between the plant and controller.

## Signal Scaling

28

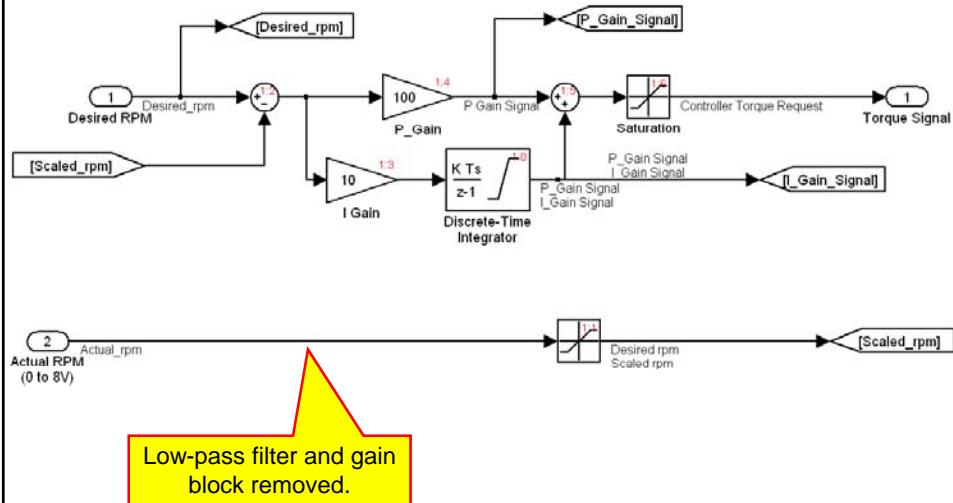
- The gain block inside the controller that scales the signal to 1 is also not part of the controller algorithm.
- In the controller, this will be accomplished with the analog to digital converter and a gain block, but this will be outside of the control algorithm.
- We will remove the gain block as well.





29

## Updated Controller



**MotoTron**

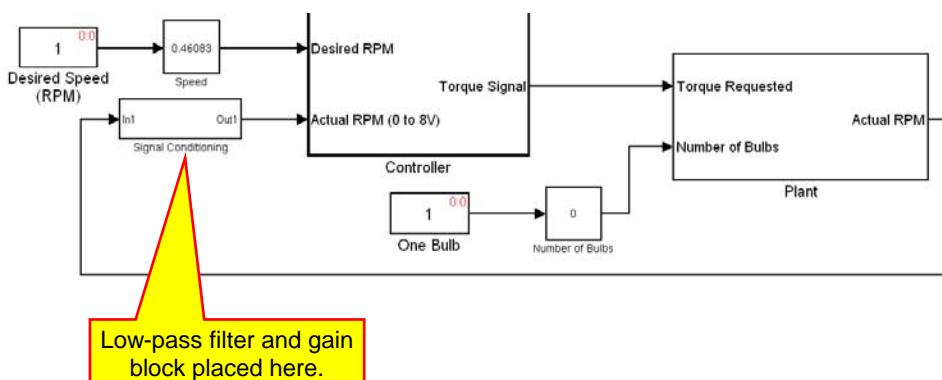
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

30

## Updated Top-Level Model



**MotoTron**

The MathWorks

**freescale**  
semiconductor

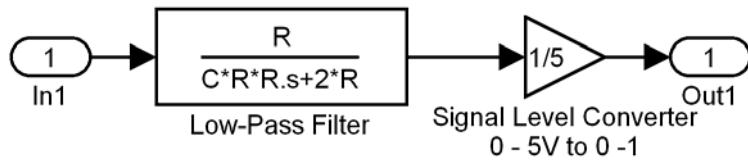
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



31

## Signal Conditioning Block

- The contents of the Signal Conditioning subsystem are shown below:



32

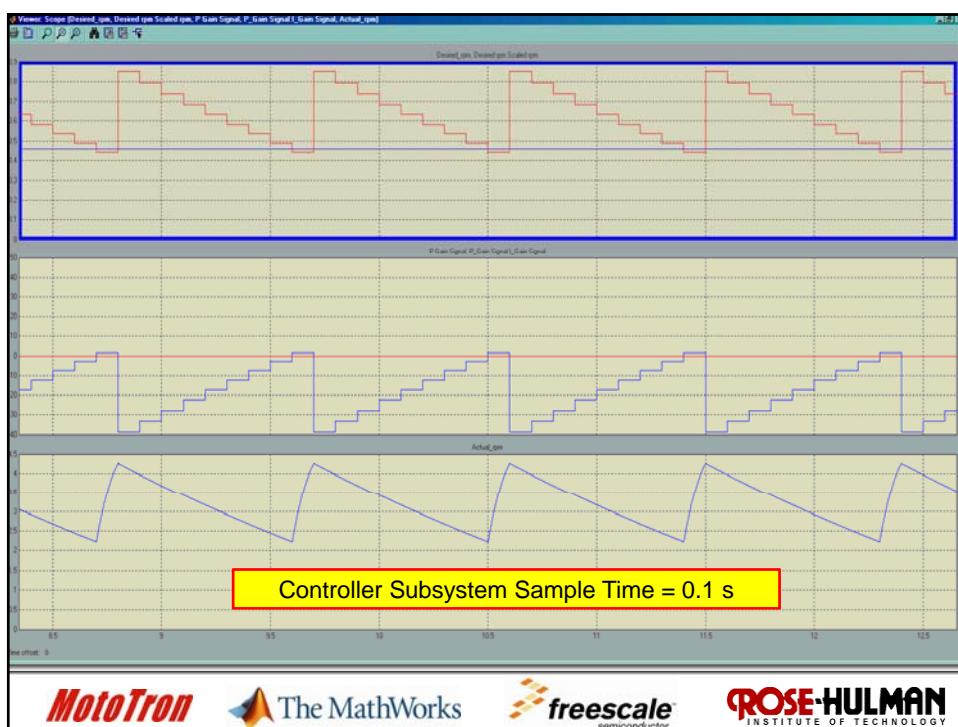
## Atomic Subsystems

- When we run the simulation, we will see that the system oscillates at the sample time specified by the controller atomic subsystem because the sample time of 0.1 seconds is too large.





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



## Controller Sample Time

34

- We can now see how small of a sample time we need for the controller so that it can accurately control the system.
- We did this previously, but there is a difference.
- When we changed the time step before using the Simulation Configuration dialog box, we changed the step size for both the controller and the plant.
- It was possible that the large step sized used in simulating the plant was the cause or part of the cause of the oscillations.
- Using this method, we know that the larger step size is used only in the controller. The plant can use an arbitrarily small step size.
- We will start reducing the Controller sample time and see how large of a sample time we can use and achieve a stable system.

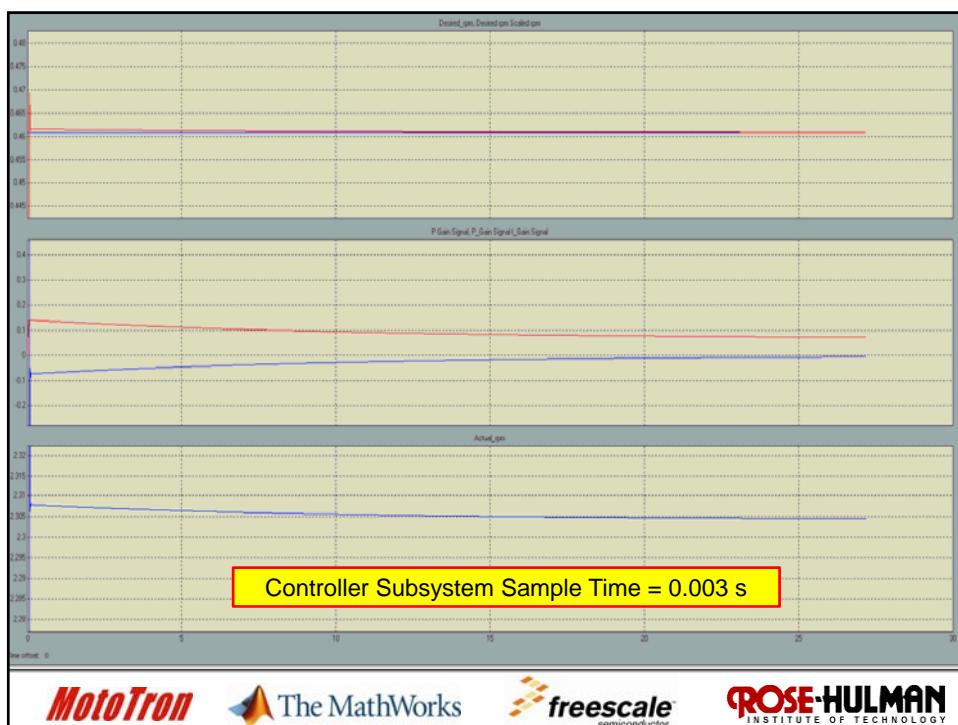
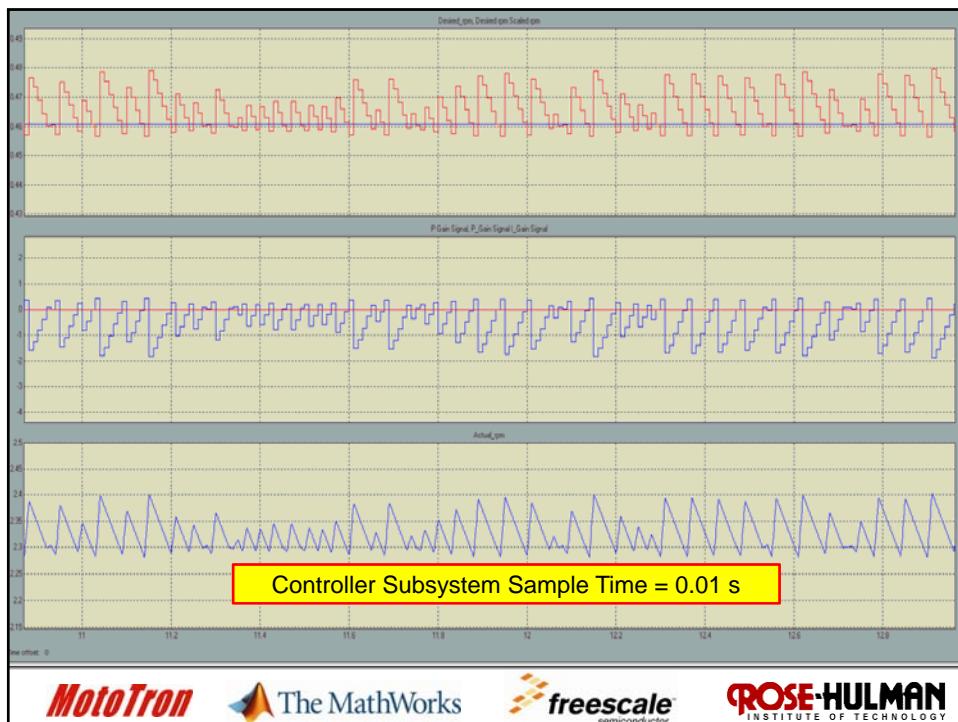


The MathWorks



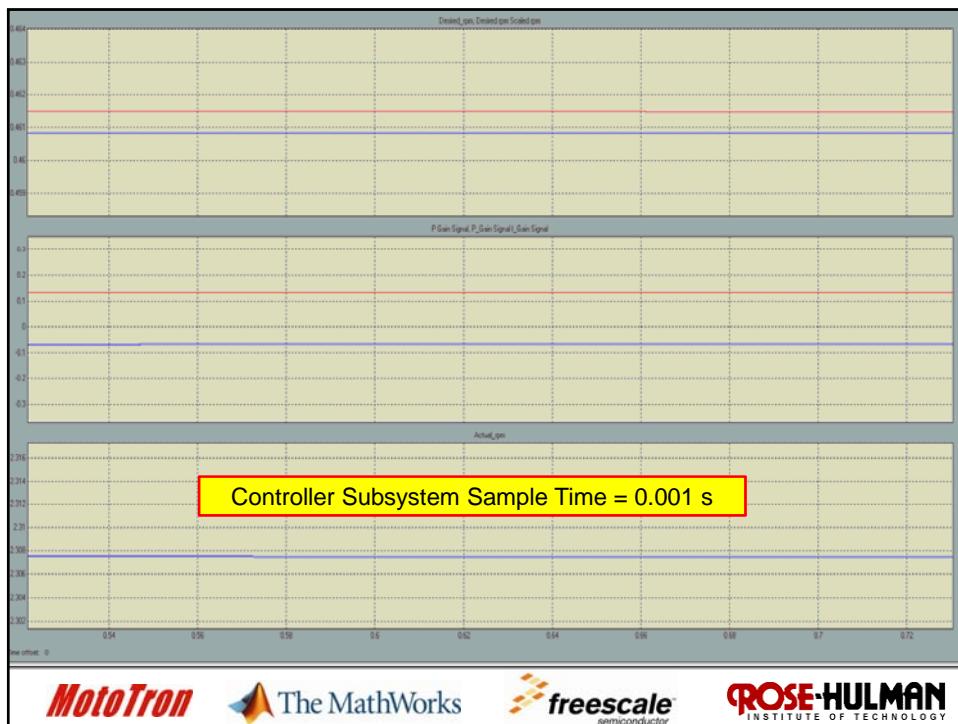


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





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



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Lecture 10A Demo 1

38

- Demonstrate the controller as an atomic subsystem with sample times of 0.1, 0.01, and 0.001 seconds.

Demo \_\_\_\_\_

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



39

## Controller Step Size

- This simulation shows that a controller step size of 0.001 to 0.003 seconds is small enough to achieve a stable system.
- This results is similar to what we found earlier, except we eliminated the possibility that the oscillation was due to the plant simulation step size.
- Now we know that the oscillations we found when making the time step too large were because of the large time step in the controller, not the plant.



40

## SIL Simulations

- We have taken the first step into seeing how the controller would behave when deployed on a controller.
- We have run the controller with a fixed time step.
- On the target, the controller will run with a fixed time step.





41

## MIL Simulations

- One problem, is that our controller is a model made out of Simulink blocks executing in the Matlab environment.
- These type of simulations are sometimes called Model-in-the-Loop or MIL simulations.
- When we deploy the controller on the target hardware, the model will be converted to C, compiled, and then the compiled code will run on the target.



42

## SIL Simulations

- So we see a slight difference.
- In our MIL simulations, the controller is implemented by executing Simulink model blocks.
- On the target hardware, the controller is implemented by executing C-code.
- How can we gain confidence that the two implementations are the same, or at least very close?





43

## SIL Simulations

- With SIL simulations, we will convert the controller to a block of C-code and run the C-code in our simulation.
- The C-code for the controller logic will be the same C-code as deployed on the target.
- The target will have additional code in the preamble to initialize the hardware, and some additional code for accessing the hardware resources (analog and digital I/O for example).
- The C-code representing the Simulink blocks in the controller will be the same.



44

## SIL Simulations

- SIL simulations will give us confidence that the C-code generated for the controller produces the same results as the Simulink block representation of the controller.
- We will accomplish this by creating an S-function block out of the controller.
- An S-function is a block of C-code with inputs and outputs to interface with the remainder of the Simulink model.





45

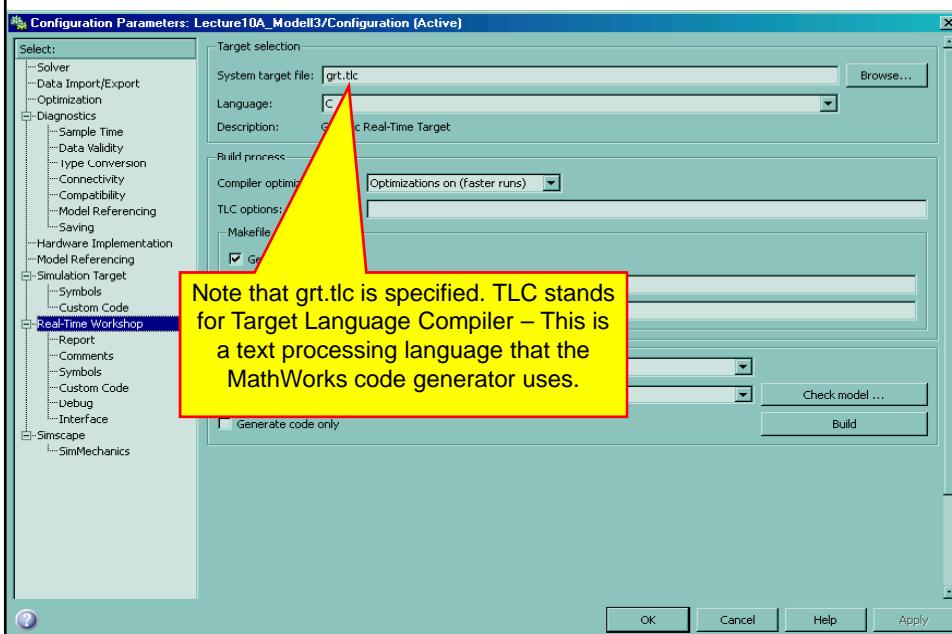
## Real-Time Workshop Embedded Coder

- Next, from the Simulink menus, select **Simulation** and then **Configuration Parameters**.
- Select the Real-Time Workshop tab.



46

## Real-Time Workshop Embedded Coder





47

## Real-Time Workshop Embedded Coder

- The default System target file is grt.tlc.
- GRT stands for generic real-time target.
- GRT engages the Real-Time Workshop – a subset of the features of the ert.tlc (mentioned on the next slide)
- It is designed for Rapid Prototyping and hardware-in-the-loop code generation.
- It is not as efficient or configurable as ert.tlc.



48

## Real-Time Workshop Embedded Coder

- ERT stands for Embedded Real-Time Target.
- Ert.tlc engages Real-Time Workshop Embedded Coder.
- We need to specify the System target file as ert.tlc which generated code for embedded real-time hardware targets.
- Click the Browse button:

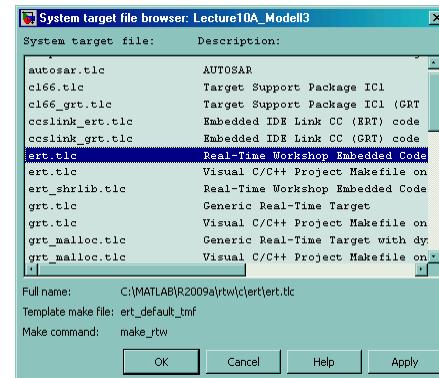




49

## Embedded Real-Time Coder

- Select the first ert.tlc file.

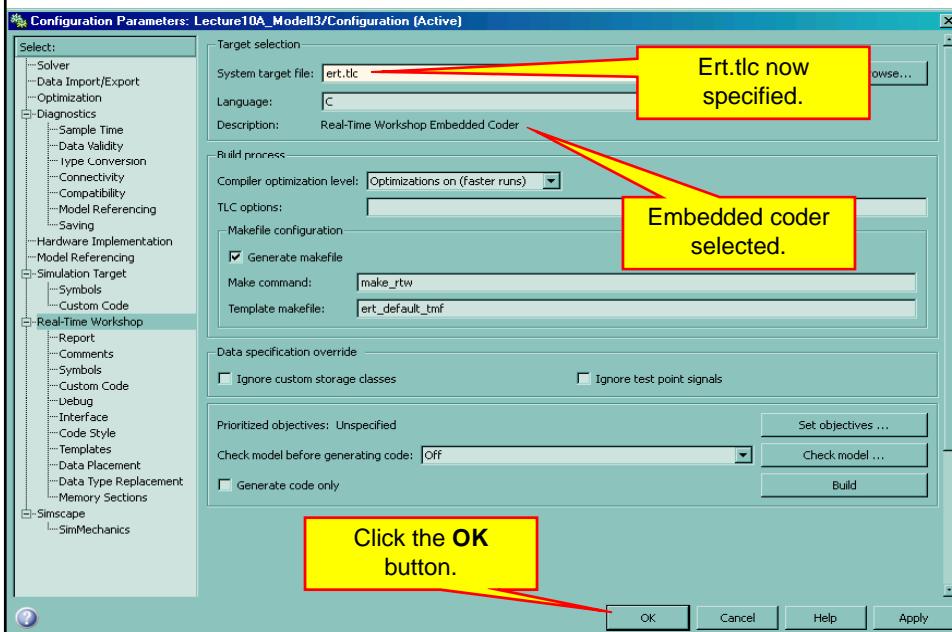


- The second ert.tlc file only generates a makefile, which you would have to run later to compile and build the generated code.



50

## Ert.tlc





51

## Ert.tlc

- It turns out that the embedded coder is used by many of the hardware vendors to generate the code for their targets.
- In this course, we will be using the rappid.tlc System target file to generate code for our MPC5554 controllers.
- If we edit the file rappid.tlc located in directory C:\Program Files\RAppID Toolbox\rappid\rappid we will see that it references ert.tlc for generating code for the Simulink blocks:



52

## Ert.tlc

```
% Inherit ERT options
rtwgensettings.DerivedFrom = 'ert.tlc';

% specify the RTW build directory name (failure to do so results
% in the code generating into the current MATLAB directory)
rtwgensettings.BuildDirSuffix = '_rappid_rtw';
END_RTW_OPTIONS
%/
```

- We see that the rappid.tlc references the ert.tlc file.





53

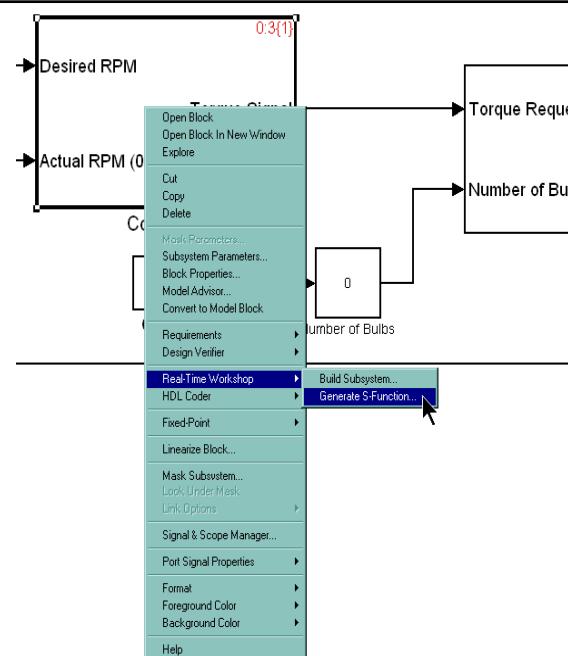
## SIL Simulations

- Because our S-function will use the ert.tlc system target file, and because the Rappid.tlc file references the ert.tlc file, the code generated for the S-function will be the same as the code generated for our targets (5554).
- Thus, we will be generating the same code for our S-function as will be generated for the target Hardware.
- This is a good way to test the code for our controller before we test it on the physical plant.
- Make sure that ert.tlc is specified as the **System target file**, save the changes, and return to the top-level of the model.



## S-Functions

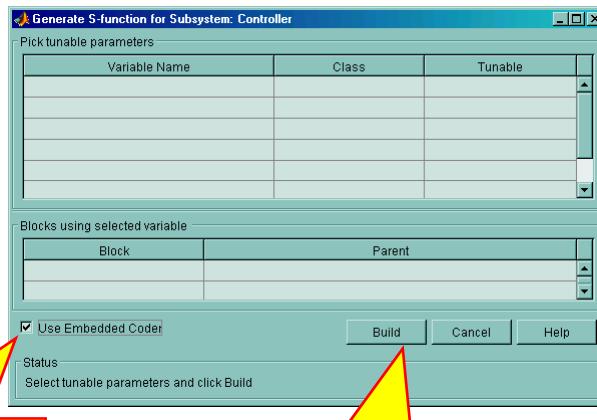
- To create an S-function block, right-click on the Controller subsystem and select **Real-Time Workshop**, and then **Generate S-Function**





## S-Functions

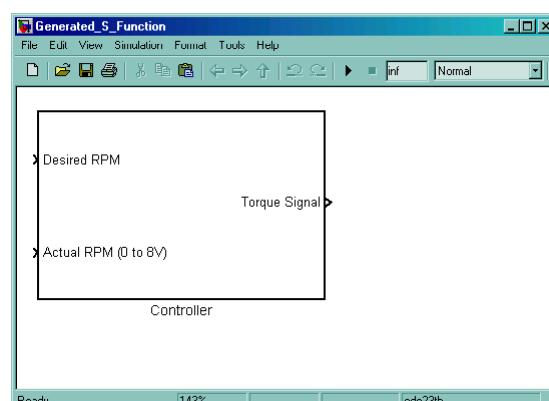
55



## S-Functions

56

- A new model will be created containing the S-function block.

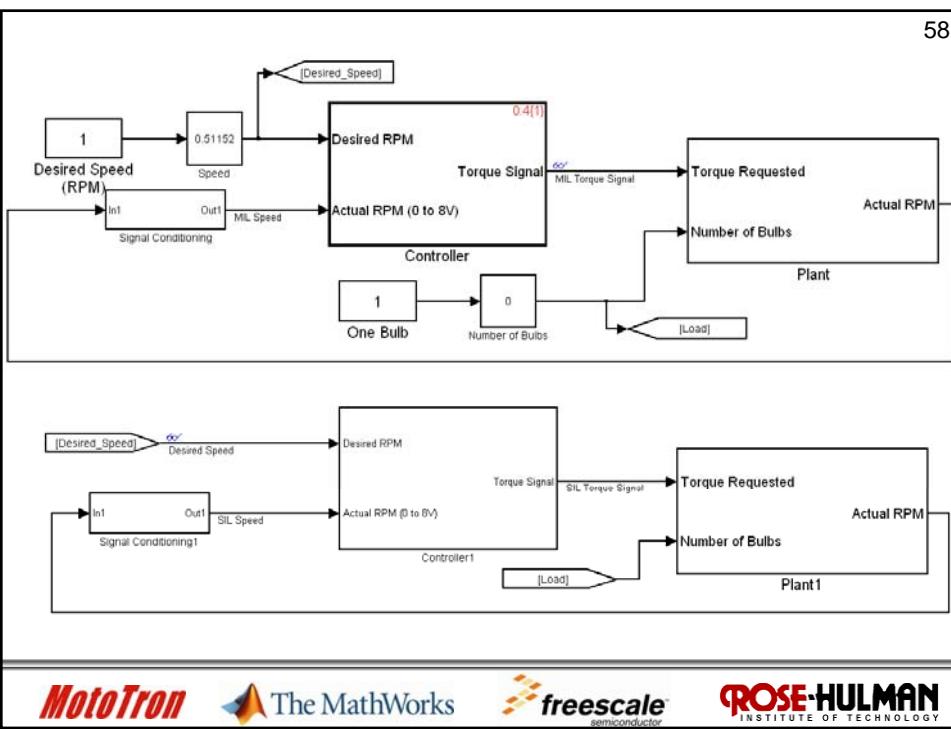




## SIL Simulations

57

- For SIL simulations, we would like to compare the performance of the system controlled with the Simulink model controller to the same system controlled with the S-function controller.
- We can do this with the system shown next.
- Copy the S-function block back into your original model, duplicate the plant and signal conditioning subsystems, and wire as shown:

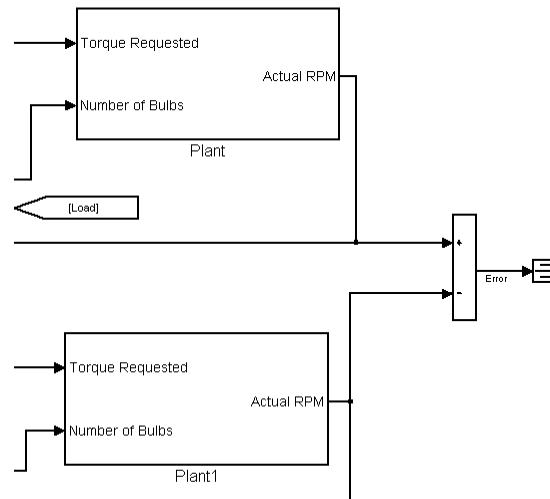




59

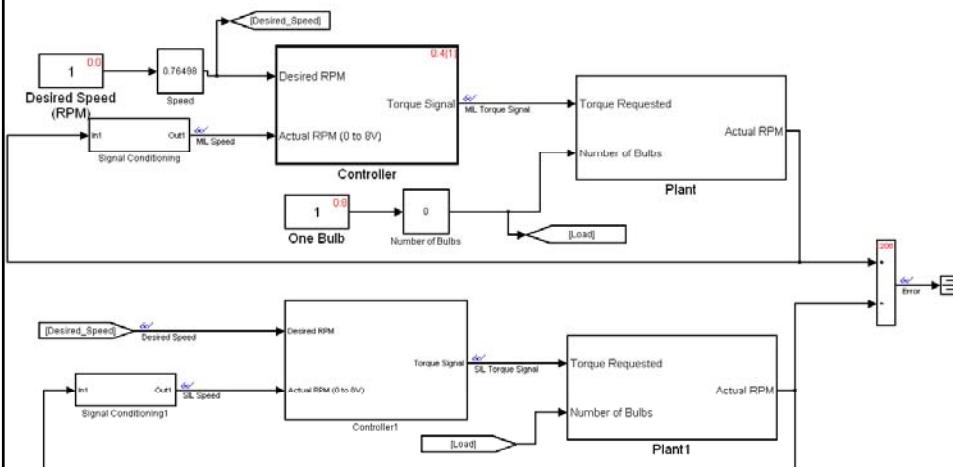
## SIL Simulations

- We would like to see the difference between the two plant outputs.
- We will take the difference between the two and plot it:



60

## Complete Top-Level Model





61

## System Performance

- The plot on the next page shows a load of zero bulbs.
- The error between the two systems is zero.
- This shows that the response of the plant to the Simulink model controller and the S-function controller are nearly identical.
- This gives us confidence that the code we generate from the Simulink model that will run on our target matches closely to the Simulink block version of the model.
- You should run several tests at different speeds and different loads to verify the S-function's operation.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

62

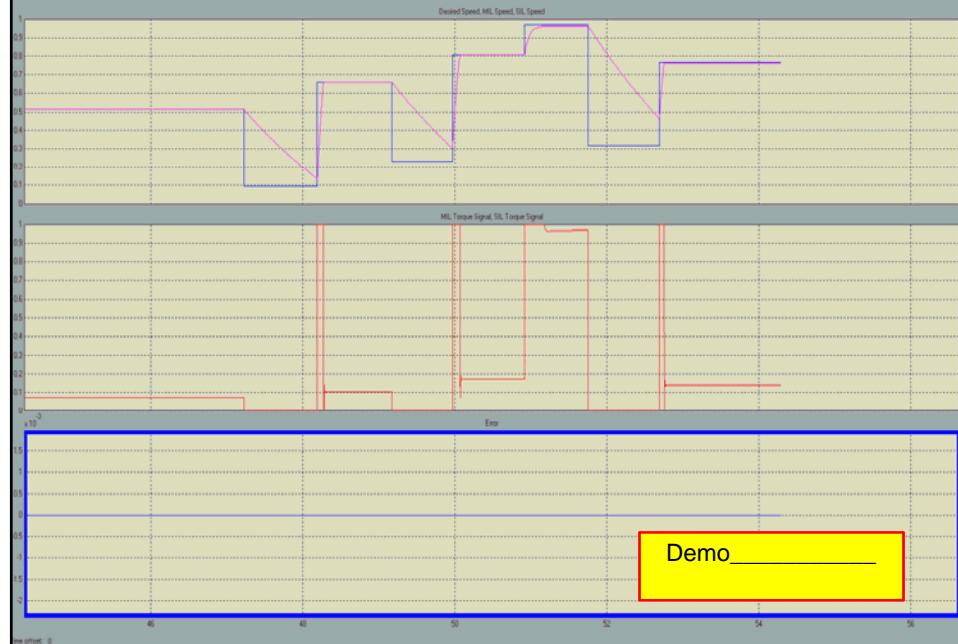
## SIL Performance





## Lecture 10A Demo 2

63





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



## Introduction to Model-Based Systems Design

### Lecture 11: Introduction to the MPC555x Target



The MathWorks



freescale<sup>®</sup>  
semiconductor



## Installing the Software

2

- The software is already installed on the computers in the Rose-Hulman Model-Based Systems Design Lab.
- Rose-Hulman students can skip to slide 15 and skip the software installation slides.



The MathWorks



freescale<sup>®</sup>  
semiconductor





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

3

## Installing the Software

- You will need the following software components
  - CodeWarrior Development Studio for MPC55xx, Build Tools Edition (To compile our Simulink Models)
  - RAppID Toolbox (Simulink blocks to access the MPC555x facilities)
  - Esys Flasher (Download models to the MPC555x development board)



4

## CodeWarrior Installation

- Choose the following options when installing CodeWarrior
  - Choose all of the default or typical settings.
  - Running the updater is not necessary.
- We will need to make the following changes:
  - Copy file C:\Program Files\Freescale\MPC55xx V2.0\bin\lmgr8c.dll
  - To directory C:\Program Files\Freescale\MPC55xx V2.0\PowerPC\_EABI\_Tools\Command\_Line\_Tools





5

## CodeWarrior License

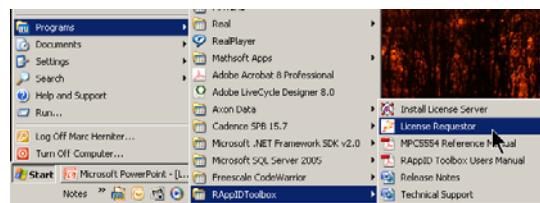
- Request a license from Freescale
- Copy the license.dat file to the following Directories:
  - C:\flexlm
  - C:\Program Files\Freescale\MPC55xx V2.0\License
  - C:\Program Files\Freescale\MPC55xx V2.0



6

## Installing RAppID Toolbox

- Unzip file RAppIDToolboxECV101Release.zip
- Run file setup.exe
  - Select the default choices.
  - Restart your system.
- Request a License File
  - In the RAppID Toolbox program group, run the License Requestor program:





## Installing RAppID

7

- Copy the license file to direcory  
C:\RAppID\_Lic
- Run Matlab.
  - Change the working directory to C:\Program Files\RAppIDToolbox\rappid
  - From the Matlab Command Line prompt, run the command **rappid\_path**

```
Command Window
To get started, select MATLAB Help or Demos from the Help menu.

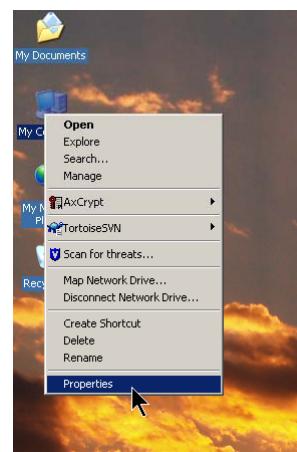
>> rappid_path
Treating 'C:\Program Files\RAppIDToolbox\rappid' as RAppID installation root.
Found SIMD Optimized DSP Blocks.
RAppID path prepended.
Successful.
>> |
```



## Installing RAppID

8

- Next We have to set up an environment variable in Windows.
- Right click on My Computer and select **Properties**:

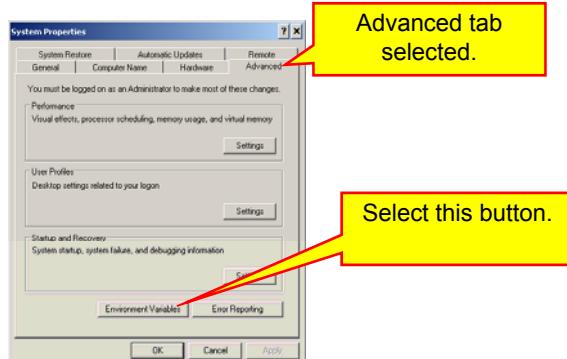




## Installing RAppID

9

- Select the **Advanced** tab
- Select the **Environment Variables** button.



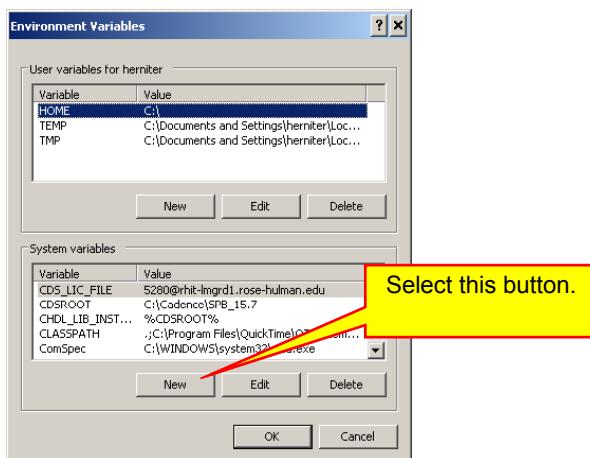
The MathWorks



## Installing RAppID

10

- Select the **New** button as shown:



The MathWorks





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

11

## Installing RAppID

- Add a new environment variable “**MW\_TOOL**” (Note all caps!)
- Give it a value **C:/Program Files/Freescale/MPC55xx V2.0**
- (Note that you might have a different version of the compiler.)



12

## Installing RAppID

- Click the OK button to define the new environment variable.
- Click the OK button two more times to exit the My Computer dialog boxes.
- You will need to restart MATLAB so that it will recognize the new variable.





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

13

## Installing eSys Flasher

- Run eSysFlasher\_v1.3\_setup.exe
  - Select the defaults.
  - Restart your system.



The MathWorks



14

## Installing eSys Flasher

- Plug in the P&E Microcomputer USB Multilink device into your USB port.
  - Select to not use the internet to find a driver.
  - It should find the driver locally.
  - If the driver installs properly, the blue light should illuminate on the USB Multilink Box.

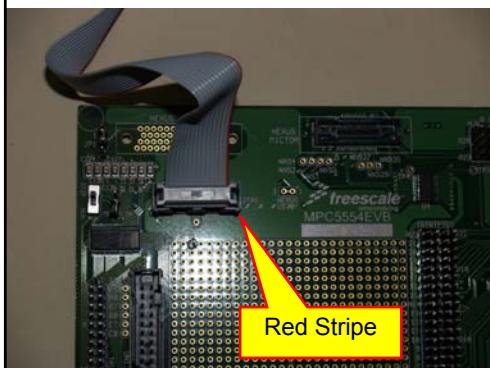




15

## Connecting the Hardware

- Plug in the ribbon cable of the USB Multilink into the port labeled JTAG on the MPC555x board.
  - Note that the red stripe should line up with pin 1 on the JTAG connector.



- Turn on the power to the MPC555x board
- We are now ready to go!

16

## MPC555x

- We will now create a simple project where we flip-flop two LEDs on the MPC555x board.
- If we can do this, we can do anything!
- Note that pin connections for the MPC555x board are documented in file MPC5554DEMO\_man\_G.pdf.





## MPC555x LED Flasher

17

- Run MATLAB, run Simulink, and then open a new Simulink Project.
- In the **RAppID** library, locate a block called **RAppID-EC** and place it in your model:

**freescale** **ROSE-HULMAN** INSTITUTE OF TECHNOLOGY

## MPC555x Setup

18

LED\_Flasher \*

RAppID MPC5554 Target Setup

System Clock : 128 MHz  
 Target : MPC5554  
 Compiler : diab  
 Target Type : IntFlash  
 Operating System : simpletarget

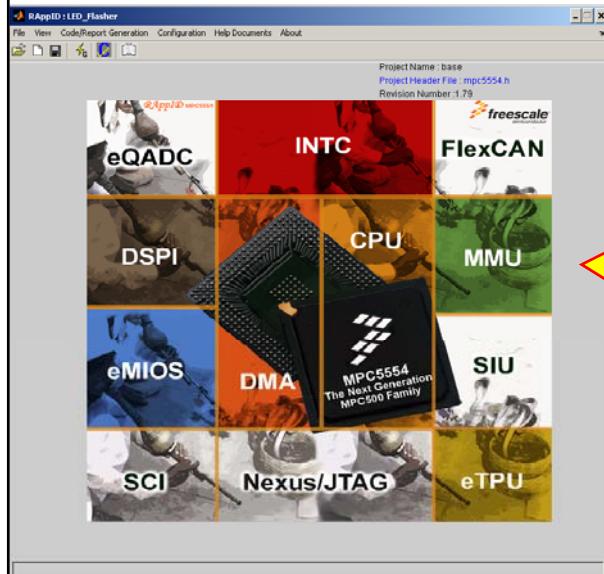
RAppID-EC

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



## MPC555x Setup

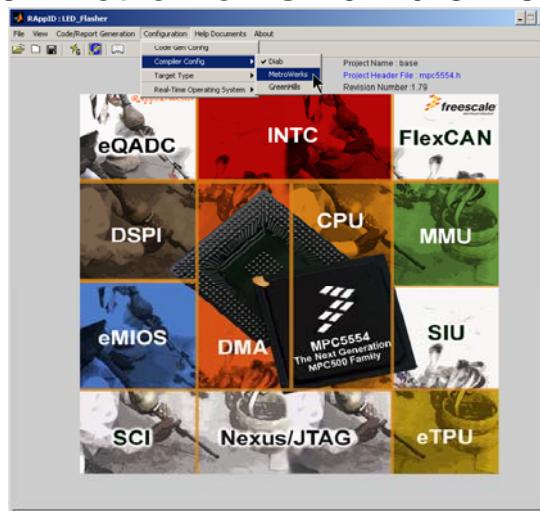
19



## MPC555x Setup

20

- Select Configuration, Compiler Config, and then MetroWerks from the menus:

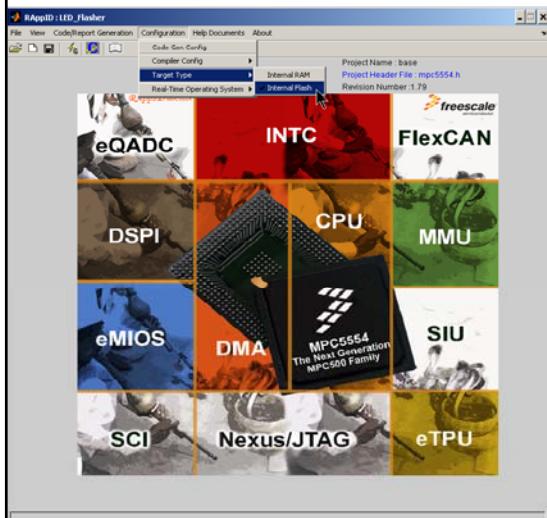




21

## MPC555x Setup

- Select Configuration, Target Type, and then Internal Flash from the menus:



- This command tells the system to use non-volatile flash memory rather than RAM.
- Select File and then Save.
- Select File and then Exit.

22

## MPC555x Setup

- The Target Setup block should show the changes to the configuration:

```
RAppID MPC5554 Target Setup
System Clock : 128 MHz
Target : MPC5554
Compiler : metrowerks
Target Type : IntFlash
Operating System : simpletarget
```

MetroWerks compiler and Internal Flash memory selected.

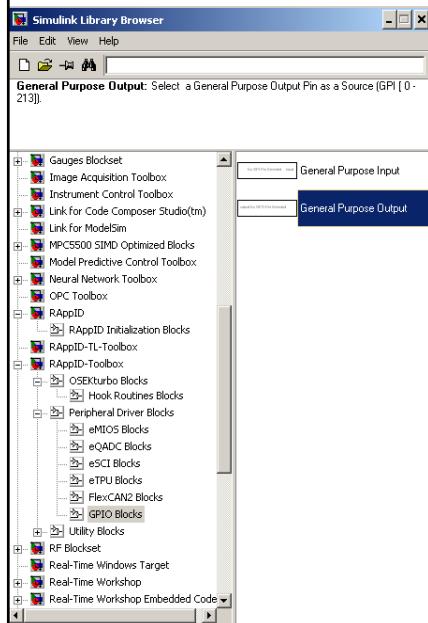
RAppID-EC





## MPC555x Digital Output

23



- We would like to turn on two LEDs.
- Place two **General Purpose Output** blocks in your model.

## MPC555x Digital Output

24

- Double click on the blocks and select an output.
- We will use:
  - Pin 203: Primary Function “GPIO”
  - Pin 204: Primary Function “GPIO”
- After making the changes, your model should look a shown:





## MPC555x Digital Output

25

RAppID MPC5554 Target Setup

System Clock : 128 MHz  
 Target : MPC5554  
 Compiler : metrowerks  
 Target Type : IntFlash  
 Operating System : simpletarget

RAppID-EC

> output GPO [203]

General Purpose Output

> output GPO [204]

General Purpose Output1

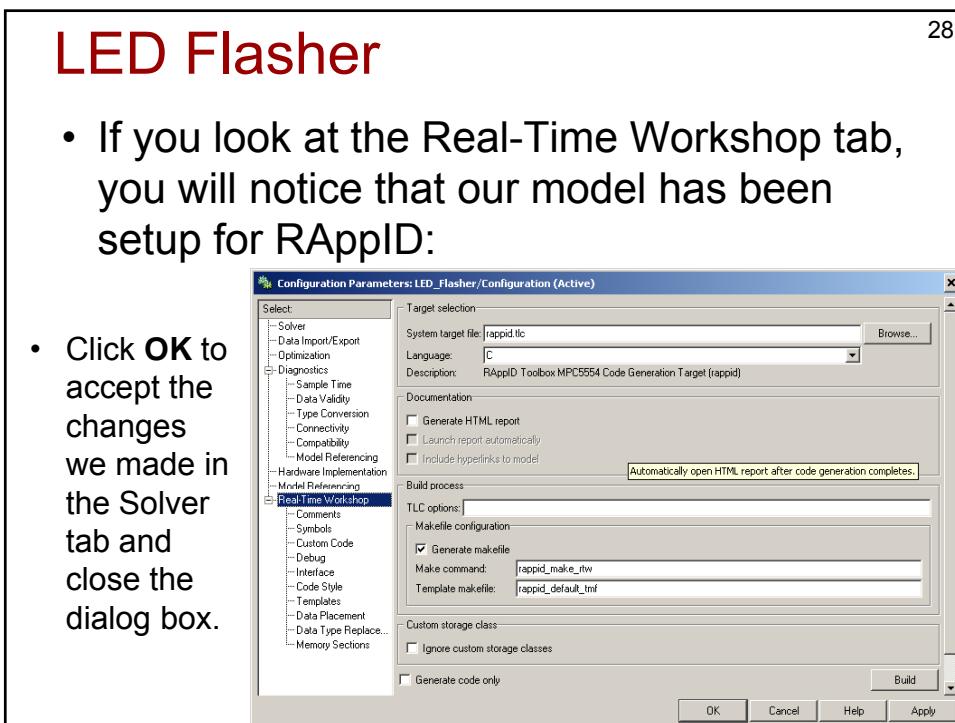
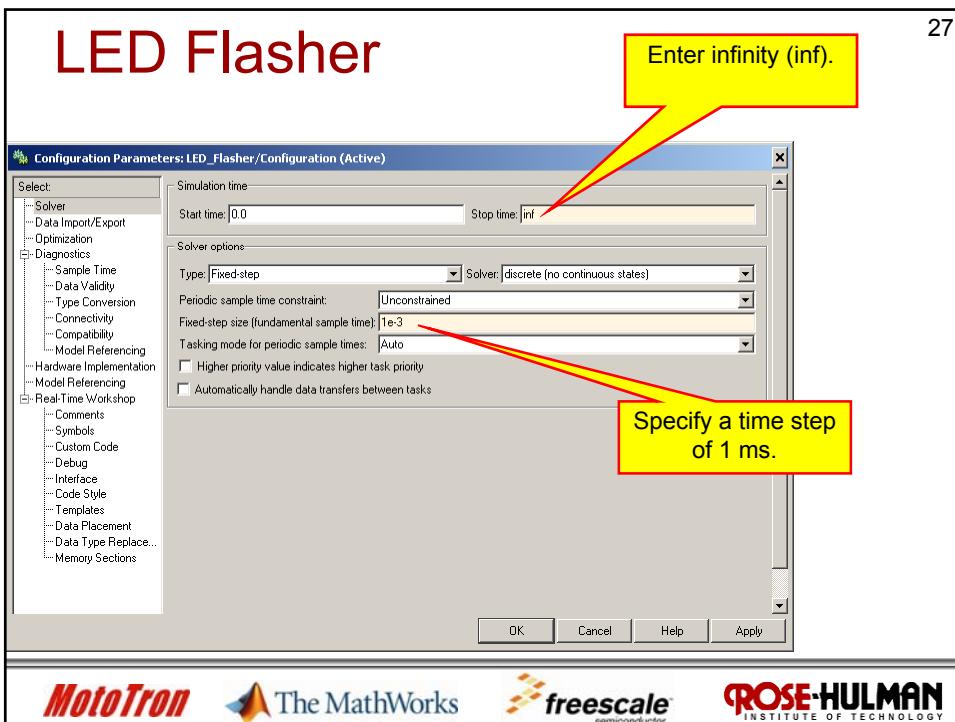


## LED Flasher

26

- We would like the LEDS to flash on and off at a 1 Hz rate.
- We need to select a discrete solver and specify a fixed time step of 1 ms. (We can actually, use a larger time step, but we'll use 1 ms.)
- Select **Simulation** and then **Configuration Parameters** from the Simulink menus.
- Select the solver tab and fill in a shown:







29

## Clock

- The next thing we need to do is to add a 1 Hz clock.
- We will do this with a **Pulse Generator** block located in the **Simulink / Sources** library:



- Place the block in your model and change the settings as shown next:

**MotoTron**

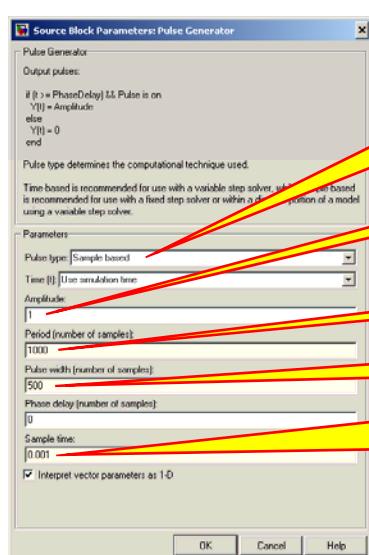
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

30

## Pulse Generator



Pulse width and period are calculated based on the number of samples.

The output flip-flops between 0 and 1.

The period is 1000 samples. Since the sample time is 1 ms, the period is 1 s.

The pulse width is 500 samples. Since the sample time is 1 ms, the period is 0.5 s.

Sample time is 1 ms. It does not have to be the same as our simulation step size. It should be an integer multiple of the simulation step size.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



31

## Compare to Constant

- We would like the LEDs to flip-flop on and off.
- When one LED is on, the other should be off.
- We could do this with a logic function.
- Instead, we will use a compare to constant block.
- The **Compare to Constant** block is located in the **Simulink / Logic and Bit Operations** library.

**MotoTron**

**The MathWorks**

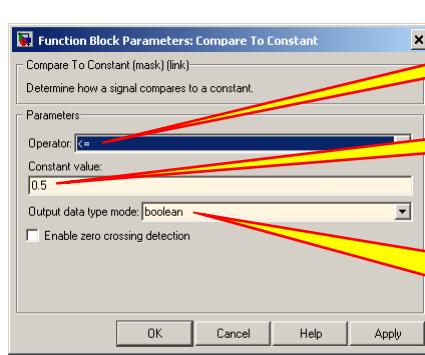
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

32

## Compare to Constant

- Specify the settings as shown:



Compare operator set to less than or equal to.

Compare the input to a numerical value of 0.5.

The output data type is Boolean (true or false). Note that True ≠ 1 and False ≠ 0.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



33

## Data Types

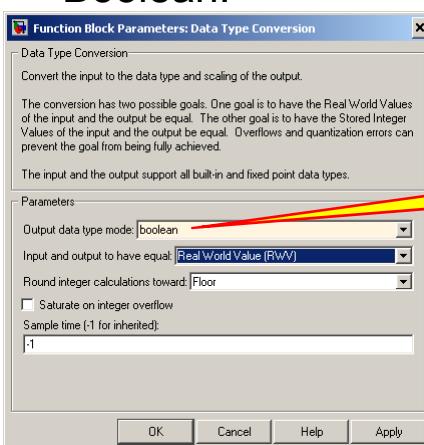
- The GPO (general purpose output) blocks require a Boolean data type for an input.
- The **Compare to Constant** block has the right output data type.
- The Pulse Generator block has a double-precision output data type.
- We need to convert from a double-precision data type to a Boolean.
- Use the **Convert** block located in the **Simulink / Commonly Used Blocks** library.



34

## Convert Block

- Specify the **Output data type mode** to be Boolean:



The output data type is Boolean (true or false).





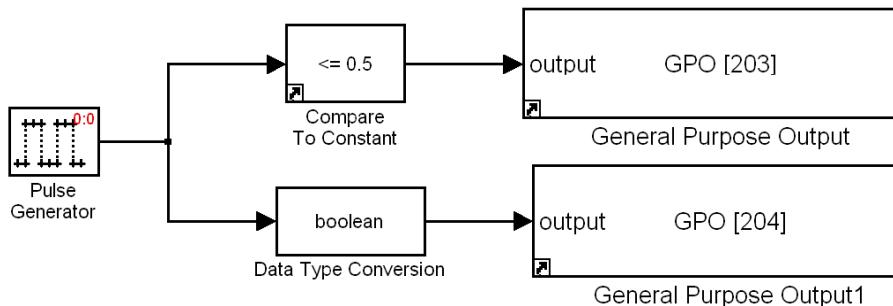
35

## Complete Model

RAppID MPC5554 Target Setup

System Clock : 128 MHz  
 Target : MPC5554  
 Compiler : metrowerks  
 Target Type : IntFlash  
 Operating System : simpletarget

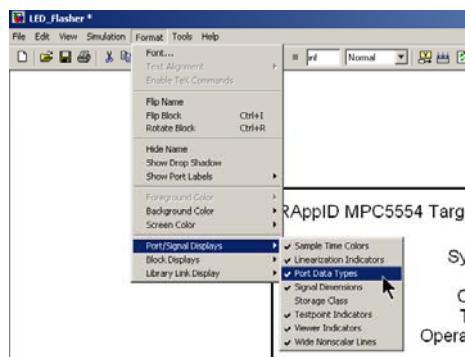
RAppID-EC



36

## Data Types

- It is important that we use the correct data types for all of the blocks.
- We can show the data types on the model.
- Select **Format**, **Port/Signal Displays**, and the **Port Data Types** from the Simulink menus to display data types on the model.

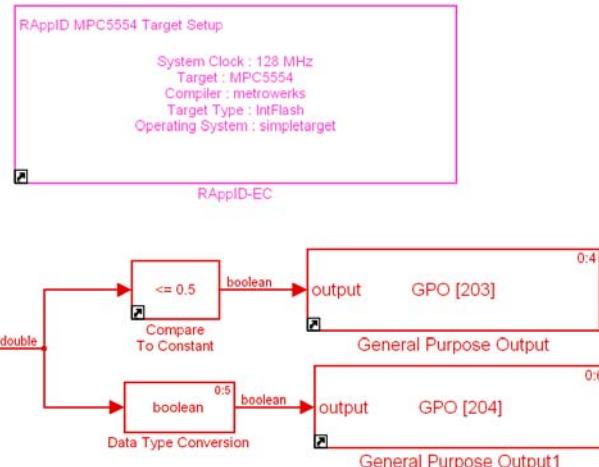




## Displaying Data Types

37

- To check the model for errors and display data types, type ctrl-d in your model window:



## Data Types

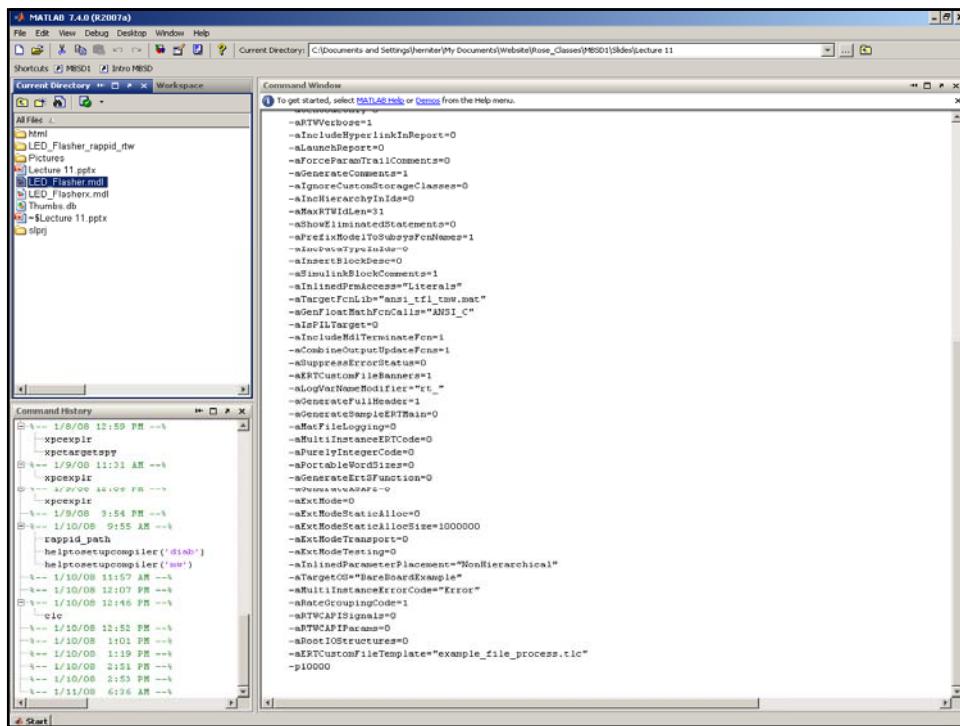
38

- For this simple model, the data types look correct.
- The GPO blocks require a Boolean input, and our input data types are Boolean.
- We can now build the model.
- Type ctrl-b to build the model.
- You should see the progress of the build process in the Matlab command window.





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



## Build

40

- If the build completes successfully, you should see the ending text blow in the command window:

```

Building file msr_init.o...
Building file rappid_main.o...
Building file siu_init.o...
Building file sys_init.o...
### Creating WR_rtlib
    1 file(s) copied.
==== Creating LED_Flasher.out from obj files ====
C:/Program Files/Freescale/"MPC55xx V2.0"/PowerPC_EABI_Tools/Command_Line_Tools/mwldeppc handlers_fit.o handlers
Created executable: LED_Flasher.elf
Building target all
*** Created executable: LED_Flasher.elf
### Successful completion of Real-Time Workshop build procedure for model: LED_Flasher
>>

```





41

## Downloading the Model

- The next thing we need to do is download the model to our board.
- Make sure that your MPC555x board has the power turned on.
- Make sure that the USB link is plugged into a USB port and that it is plugged into the JTAG port on your MPC555x board.
- Run the eSys Flasher program.



## eSys Flasher

42

- Select the **P&E Wiggler (USB)** option.



- After clicking the OK button, if you get a message below



- Then you forgot to turn on your MPC555x power, or the USB link is not plugged in.





43

## eSys Flasher

- If you have communication between your computer and MPC555x, you should see the screen below:



44

## eSys Flasher

- Select the **S-Record File** type and click the **Program** button:



- You need to locate a file named **LED\_Flasher.mot**.
- This should be located in the directory named **LED\_Flasher\_rappid\_rtw** which should be in your working directory:



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

45

## eSys Flasher

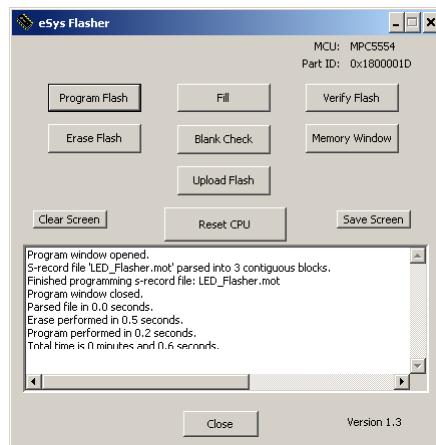


- Select file LED\_Flasher.mot and click the Open button to program your board.
- If successful, you should see the window shown next:



46

## eSys Flasher





## MPC555x Wiring

47

- Next, we need to connect the GPO outputs on the MPC555x board to the LED display on the MPC555x board.
- Information documenting the MPC555x demo board is located in file MPC5554DEMO\_man\_G.pdf.
- In our model we are using GPO 203 and GPO 204.
- Open this file and search for the text 203.



## MPC555x Output Pins

48

### A23– AF26 HEADER

BALL - SIGNAL	BALL - SIGNAL	BALL - SIGNAL	BALL - SIGNAL
A23 – MDO8	A24 – VDD	A25 – 3.3V	A26 – GND
B23 – MDO4	B24 – MD00	B25 – GND	B26 – 3.3V
C23 – MDO1	C24 – GND	C25 – 3.3V	C26 – VDD
D23 – GND	D24 – 3.3V	D25 – TCK	D26 – TDI
E23 – 3.3V	E24 – TMS	E25 – TDO	E26 – TEST*
F23 – MSE00*	F24 – JCOMP	F25 – EVTI*	F26 – GND
G23 – MSE01*	G24 – MCKO	G25 – GP1	G26 – TPU_B15
H23 – RDY*	H24 – GPIO203	H25 – TPU_B14	H26 – TPU_B13
J23 – 5V	J24 – TPU_B12	J25 – TPU_B11	J26 – TPU_B9
K23 – TPU_B10	K24 – TPU_B8	K25 – TPU_B7	K26 – TPU_B5
L23 – TPU_B6	L24 – TPU_B4	L25 – TPU_B3	L26 – TPU_B2
M23 – TCRCLK_B	M24 – TPU_B1	M25 – TPU_B0	M26 – SIN_B
N23 – SOUT_B	N24 – PCS_B3	N25 – PCS_B0	N26 – PCS_B1
P23 – PCS_A3	P24 – PCS_B4	P25 – SCK_B	P26 – PCS_B2
R23 – PCS_B5	R24 – SOUT_A	R25 – SIN_A	R26 – SCK_A
T23 – PCS_A1	T24 – PCS_A0	T25 – PCS_A2	T26 – 5V
U23 – PCS_A4	U24 – TXDA	U25 – PCS_A5	U26 – 3.3V
V23 – CNTX_C	V24 – RXDA	V25 – RSTOUT*	V26 – RSTCFG*
W23 – RXDB	W24 – CNRX_C	W25 – TXDB	W26 – RESET*
Y23 – WKPCFG	Y24 – BOOTCFG1	Y25 – GND	Y26 – GND
AA23 – 5V	AA24 – PLLCFG1	AA25 – BOOTCFG0	no connection
AB23 – VDD	no connection	AB25 – PLLCFG0	no connection
AC23 – GND	AC24 – VDD	AC25 – VRG33	no connection
AD23 – NC1	AD24 – GND	AD25 – VDD	AD26 – 3.3V
AE23 – 3.3V	AE24 – CLKOUT	AE25 – GND	AE26 – VDD
AF23 – CNRX_B	AF24 – 3.3V	AF25 – ENGCLK	AF26 – GND

We see that I/O port GPO203 is pin H24.





49

## MPC555 Output Pins

### A23- AF26 HEADER

BALL - SIGNAL	BALL - SIGNAL	BALL - SIGNAL	BALL - SIGNAL
A23 - MDO8	A24 - VDD	A25 - 3.3V	A26 - GND
B23 - MDO4	B24 - MD00	B25 - GND	B26 - 3.3V
C23 - MDO1	C24 - GND	C25 - 3.3V	C26 - VDD
D23 - GND	D24 - 3.3V	D25 - TCK	D26 - TDI
E23 - 3.3V	E24 - TMS	E25 - TDO	E26 - TEST*
F23 - MSE00*	F24 - JCOMP	F25 - EVTI*	F26 - EVTO*
G23 - MSE01*	G24 - MCKO	G25 - <b>GPIO204</b>	G26 - TPU_B15
H23 - RDY*	H24 - GPIO203	H25 - TPU_B14	H26 - TPU_B13
J23 - 5V	J24 - TPU_B12	J25 - TPU_B11	J26 - TPU_B9
K23 - TPU_B10	K24 - TPU_B8	K25 - TPU_B7	K26 - TPU_B5
L23 - TPU_B6	L24 - TPU_B4	L25 - TPU_B3	L26 - TPU_B2
M23 - TCRCLK_B	M24 - TPU_B1	M25 - TPU_B0	M26 - SIN_B
N23 - SOUT_B	N24 - PCS_B3	N25 - PCS_B0	N26 - PCS_B1
P23 - PCS_A3	P24 - PCS_B4	P25 - SCK_B	P26 - PCS_B2
R23 - PCS_B5	R24 - SOUT_A	R25 - SIN_A	R26 - SCK_A
T23 - PCS_A1	T24 - PCS_A0	T25 - PCS_A2	T26 - 5V
U23 - PCS_A4	U24 - TXDA	U25 - PCS_A5	U26 - 3.3V
V23 - CNTX_C	V24 - RXDA	V25 - RSTOUT*	V26 - RSTCFG*
W23 - RXDB	W24 - CNRX_C	W25 - TXDB	W26 - RESET*
Y23 - WKPCFG	Y24 - BOOTCFG1	Y25 - GND	Y26 - GND
AA23 - 5V	AA24 - PLLCFG1	AA25 - BOOTCFG0	no connection
AB23 - VDD	no connection	AB25 - PLLCFG0	no connection
AC23 - GND	AC24 - VDD	AC25 - VRC33	no connection
AD23 - NC1	AD24 - GND	AD25 - VDD	AD26 - 3.3V
AE23 - 3.3V	AE24 - CLKOUT	AE25 - GND	AE26 - VDD
AF23 - CNRX_B	AF24 - 3.3V	AF25 - ENGCLK	AF26 - GND

We see that I/O port GPO204 is pin G25.



## MPC555x

50

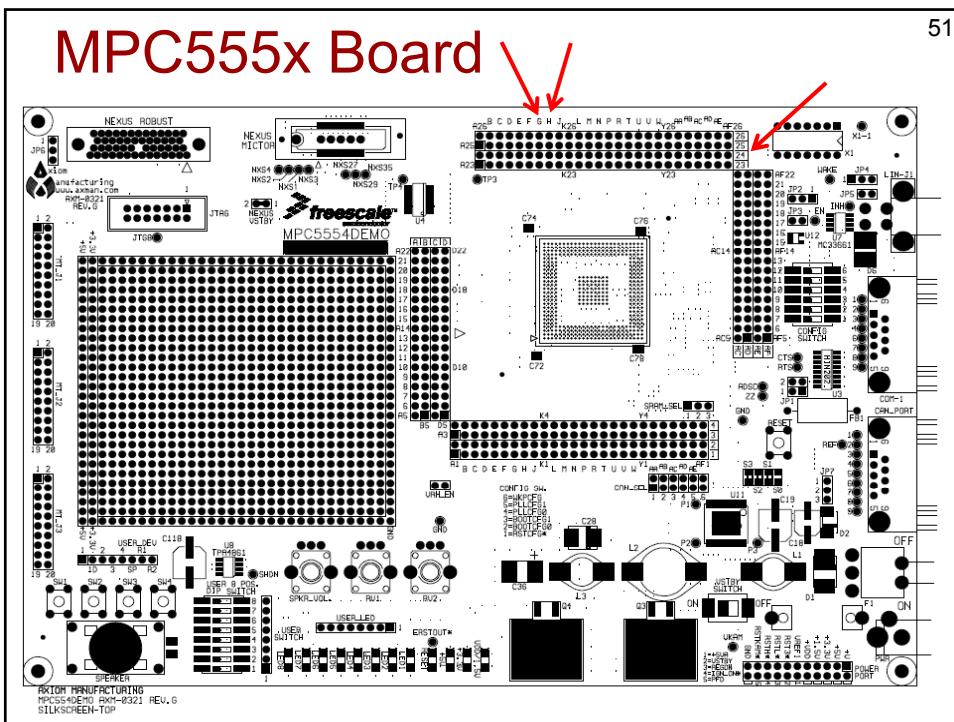
- We need to locate pins G25 and H24 on the MPC555x board.
- We can do this by looking at file MPC5554DEMO\_TOP\_F-G.pdf in the MPC555x documentation, or by looking at the board:



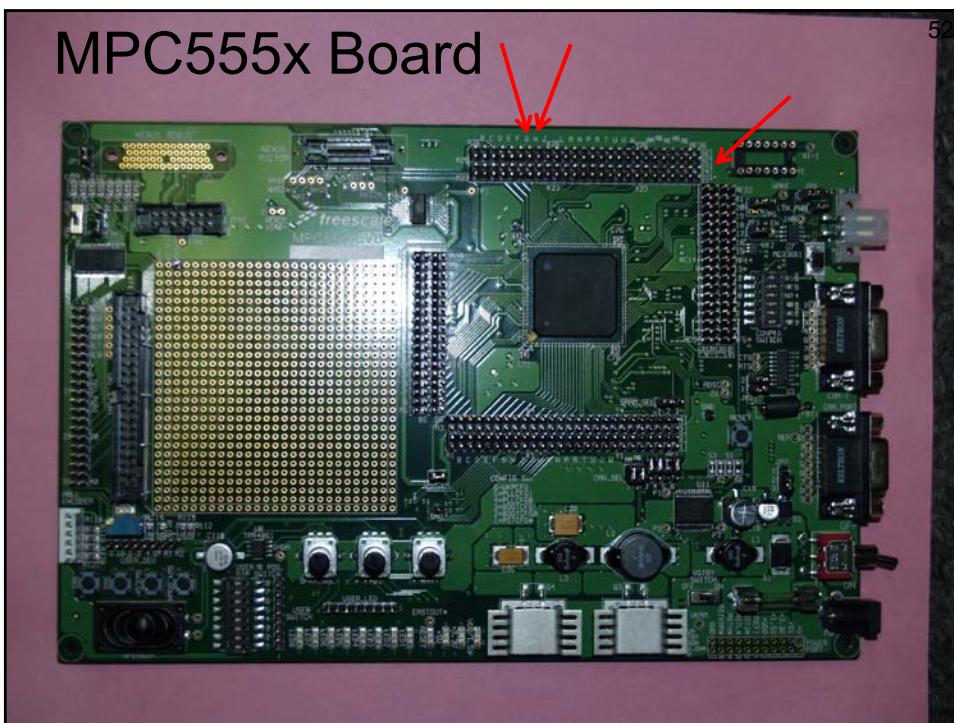


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

51



52





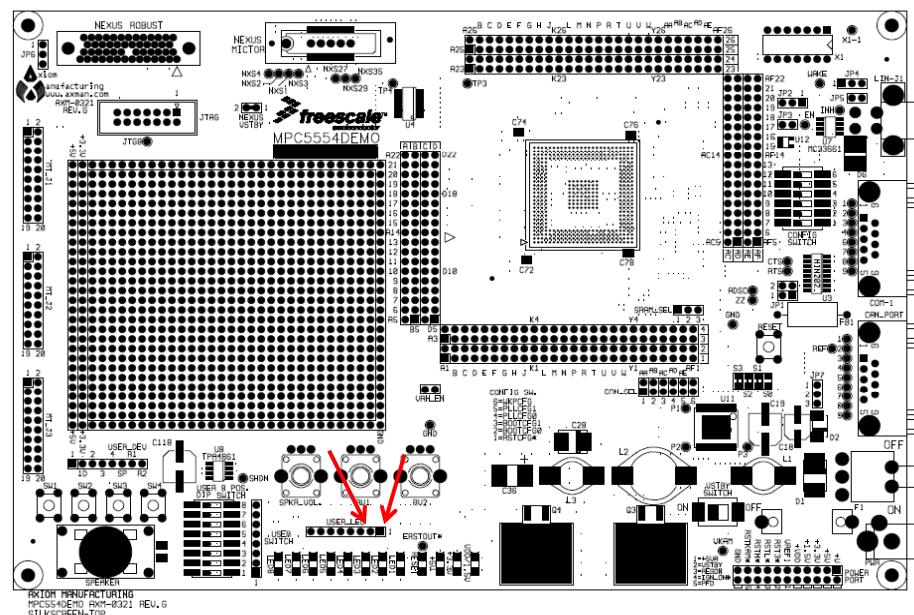
53

## MPC555x

- We would like to hook up these digital outputs to the LED display on the MPC555xDemo board.
- We will use LED1 and LED2.
- The LED connector is shown on the next slide:

54

## MPC555x Board

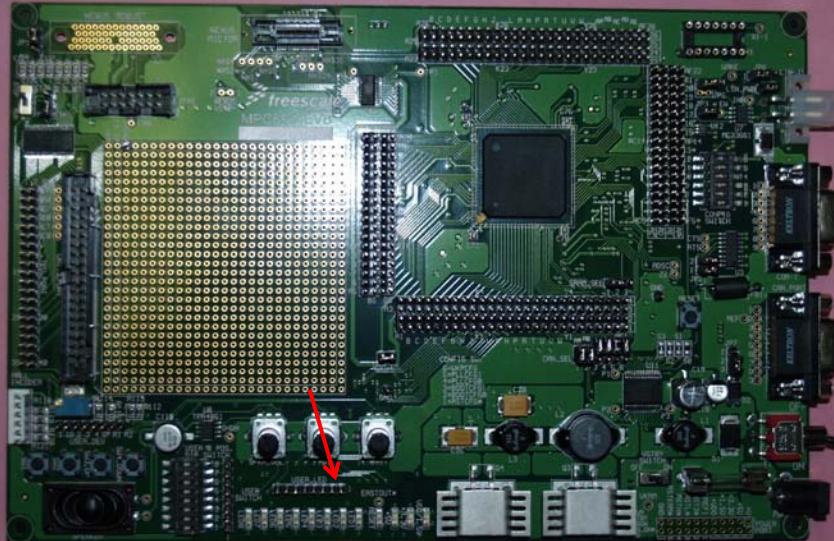




Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

55

## MPC555x Board



56

## MPC555x Wiring

- Use the patch-cord wires to connect the digital outputs to the LEDs.
- When you cycle the power to the board, or press the Reset button, the LEDs should flash on and off at a 1 Hz rate.





57

## Lecture 11 Exercise 1

- Change the LED flasher frequency to 2 Hz and demonstrate your working system.

Demo \_\_\_\_\_



58

## Lecture 11 Exercise 2

- Create an 8-Bit ring counter that changes state every  $\frac{1}{2}$  second.
- Do not use Stateflow to do this.
- The LED output sequence is shown below:

```

1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1

```

Demo \_\_\_\_\_





59

## Lecture 11 Exercise 3

- Create an 8-Bit up-down ring counter that changes state every  $\frac{1}{2}$  second.
- Do not use Stateflow to do this.
- The LED output sequence is shown below:

1 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0	0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0	0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0	0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0	0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0	1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1	0 1 0 0 0 0 0 0

Demo\_\_\_\_\_



60

## Lecture 11 Exercise 4

- Create an 8-Bit ring counter that:
  - Changes the rate at which the counter changes state.
  - For five seconds, the state should change every 0.1 seconds.
  - For 5 seconds, the state should change every 0.2 seconds.
  - The rate change continually flips between 0.1 and 0.2 seconds every 5 seconds.
- Do not use Stateflow to do this.

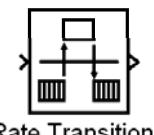




61

## Lecture 11 Exercise 4 (Cont)

- You may want to display sample times on the model. Select **Format**, **Port/Signal Displays**, and the **Sample Time Colors** from the Simulink menus.
- You may need to use a **Rate Transition** block located in the **Simulink / Signal Attributes** library.



Rate Transition

Demo \_\_\_\_\_



Any Questions?





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



## Introduction to Model-Based Systems Design

### Lecture 11A: The FreeMASTER Debugging Tool



The MathWorks



freescale  
semiconductor



## FreeMASTER

2

- We will next look at a tool called FreeMASTER from Freescale Semiconductor that allows us to observe the value of Simulink signals inside the MPC555x microcontroller in real-time.
- We will assign names to signals of interest in the Simulink model.
- FreeMASTER can then display the value of the signal:
  - As a text display that is updated at a specified rate.
  - Graphically in a plot.



The MathWorks



freescale  
semiconductor





3

## Setup

- The only additional setup required is to connect your PC to the MPC555x board using a 9-pin serial cable.
- The cable plugs into the port labeled COM-1 on the MPC555x board.
- The communication settings will be specified in your Simulink model with a configuration block and a setup dialog box in FreeMASTER.



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Two-Speed Ring Counter

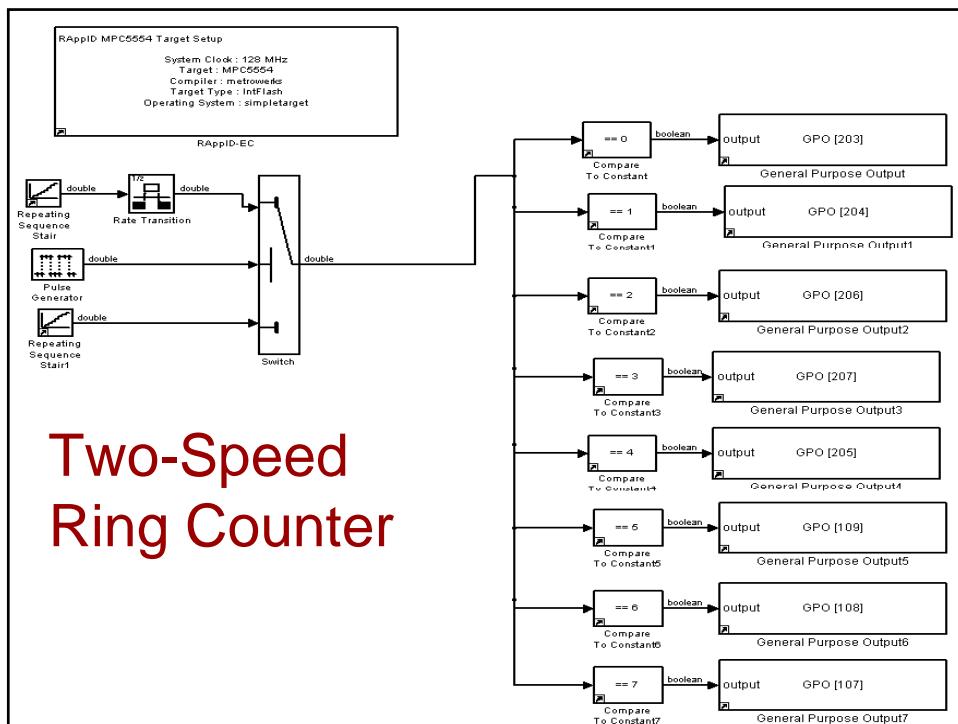
4

- We will start with the model created in Lecture 11, Exercise 4.
- As a reminder, this model did the following:
  - Create an 8-Bit ring counter that:
    - Changes the rate at which the counter changes state.
    - For five seconds, the state should change every 0.1 seconds.
    - For 5 seconds, the state should change every 0.2 seconds.
    - The rate change continually flips between 0.1 and 0.2 seconds every 5 seconds.



The MathWorks

freescale<sup>®</sup>  
semiconductor



## FreeMASTER

6

- To use the FreeMASTER tool, we need to place a block called **eSCI\_FMSTR\_Interface** in our model.
- This block is located in library **RAppID-Toolbox / Utility Blocks / FreeMaster\_Interface**
- This block sets up the serial communication on the MPC555x board.
- The only option is the baud rate, which we will leave at 57600:

FreeMASTER eSCI Interface Setup

eSCI Module: eSCI\_A  
Baud Rate: 57600  
Tx/Rx and Protocol Mode: Polling

eSCI\_FMSTR\_Interface



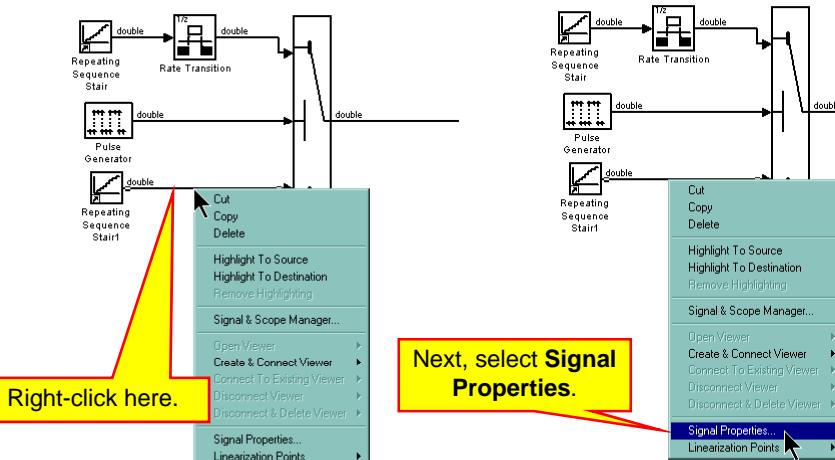
7

## FreeMASTER

- We can view any signal in the model.
- To view a signal, we need to name the signal line and then declare the storage class as ExportedGlobal.
- To name a signal line, right-click on the signal line and select **Signal Properties**.
- We will name the signal lines of the Repeating Sequence signal sources.
- Right-click as shown next:

8

## Naming Signals

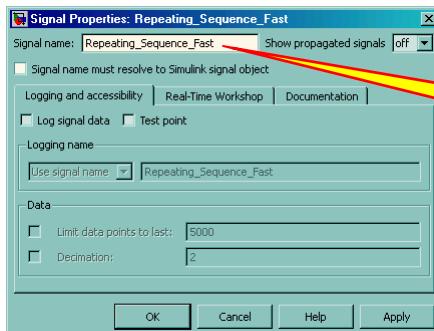




## Naming Signals

9

- Specify the name of the signal as Repeating\_Signal\_Fast



**MotoTron**

**The MathWorks**

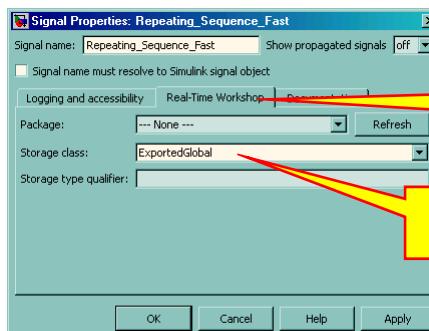
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Naming Signals

10

- Next, select the **Real-Time Workshop** tab and specify the **Storage class** as **ExportedGlobal**:

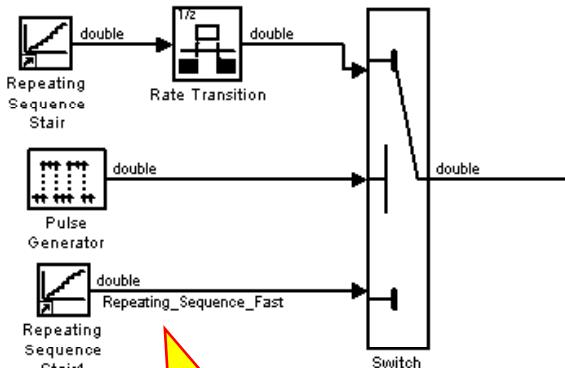


- Click the **OK** button when done. The signal name will be displayed on the block diagram.



11

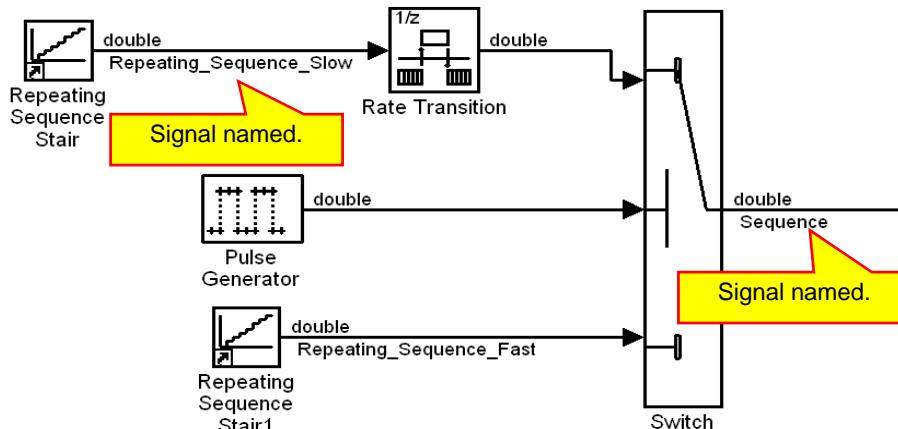
## Naming Signals



## Naming Signals

12

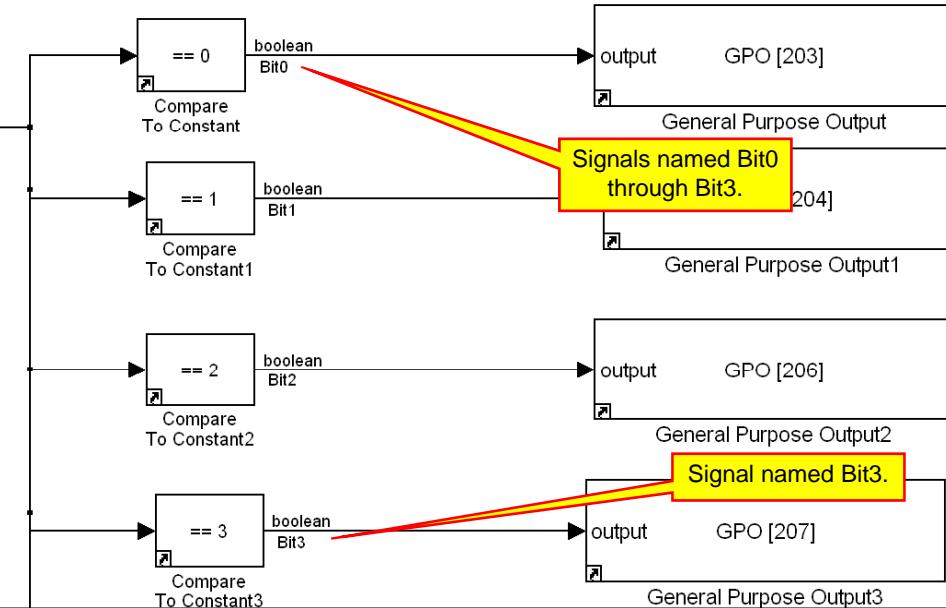
- Name the following signals shown on the next few slides.
- Be sure to declare them as ExternalGlobal.





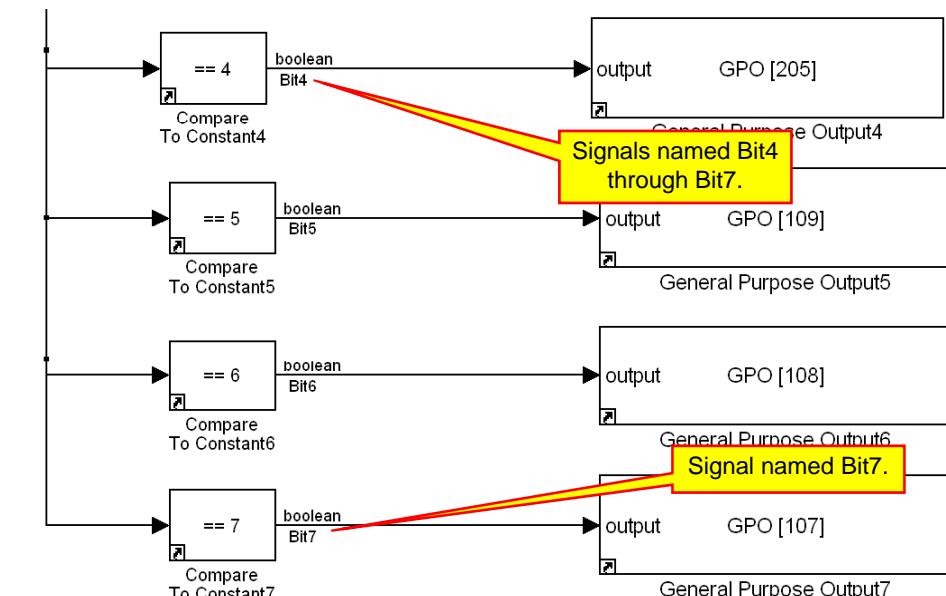
## Naming Signals

13



## Naming Signals

14





15

## Model Complete

- We have now named all of the signals of interest.
- Build your model and download it to the MPC555x target.
- Verify that the model runs correctly by displaying the correct pattern on the LEDs.
  - The purpose of learning the FreeMASTER tool is to help us debug models that are not working correctly.
  - We already know that this model works.
  - The purpose of this model is to learn how to use the FreeMASTER tool.



16

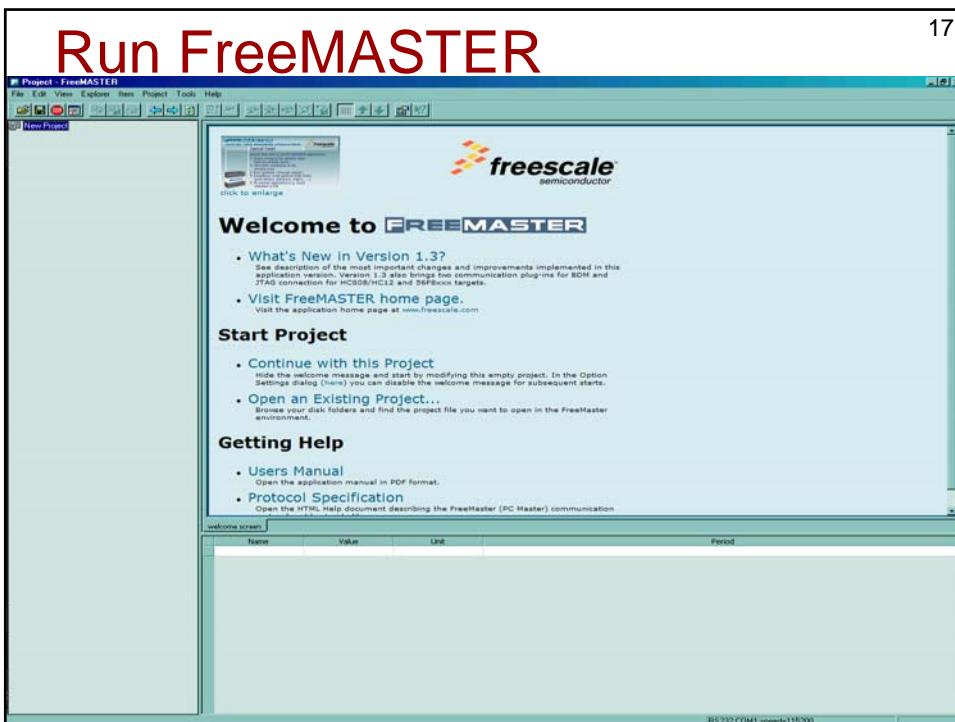
## Data Types

- Before we continue, it is good to note the data types of the signals we will be viewing:
- Double
  - Repeating\_Sequence\_Slow,  
Repeating\_Sequence\_Fast, Sequence
- Boolean
  - Bit0 through Bit7
- We will need this information later when we select signals.

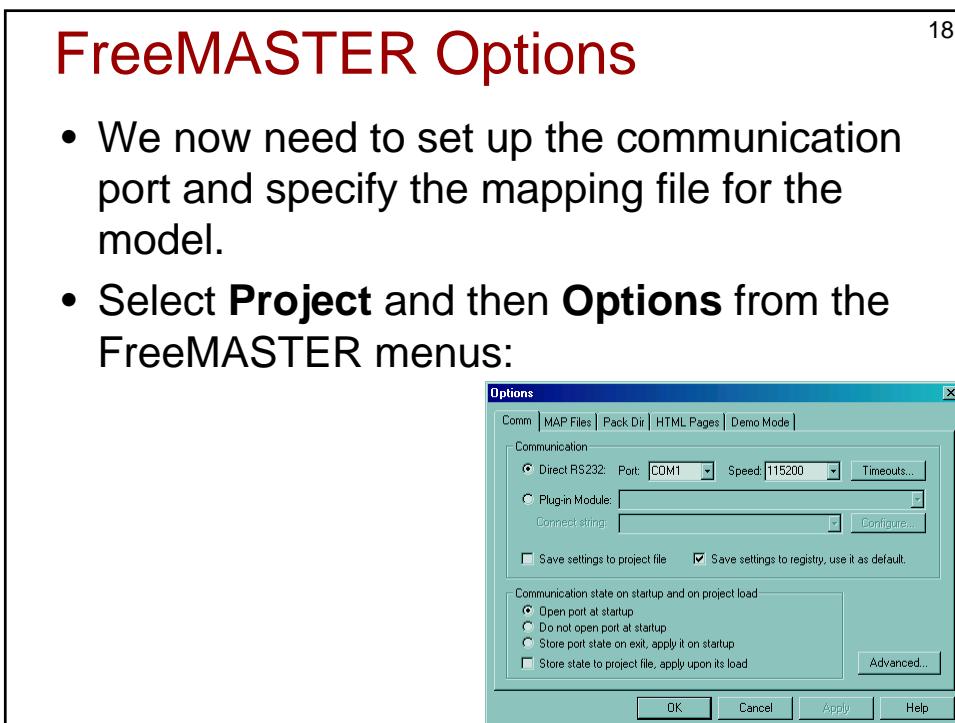




Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



17



18

## FreeMASTER Options

- We now need to set up the communication port and specify the mapping file for the model.
- Select **Project** and then **Options** from the FreeMASTER menus:

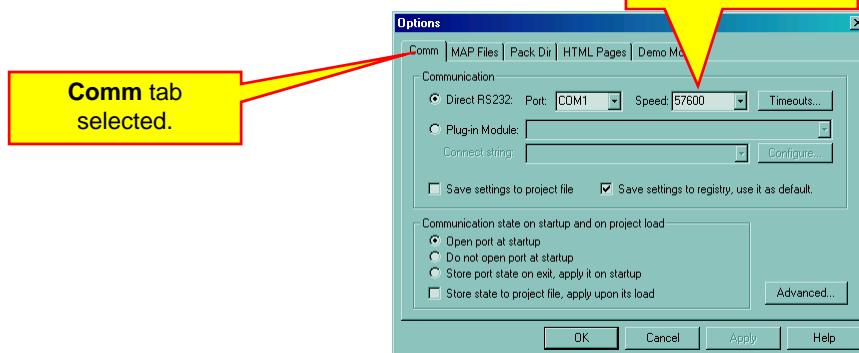


19

## FreeMASTER Setup

- The default com port is COM1 (which should work for us) and the speed is 115200.
- In the FreeMASTER setup block we placed in the model, the BAUD rate was set to 57600.
- Change the speed to 57600:

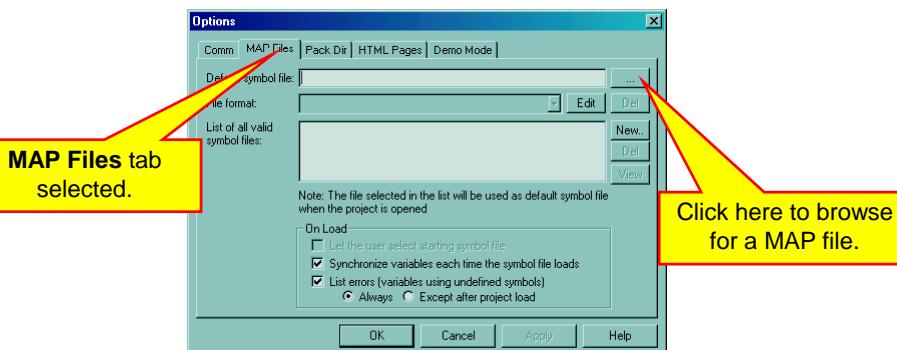
Change to 57600.



20

## FreeMASTER Setup

- Next select the MAP Files tab:





## MAP Files

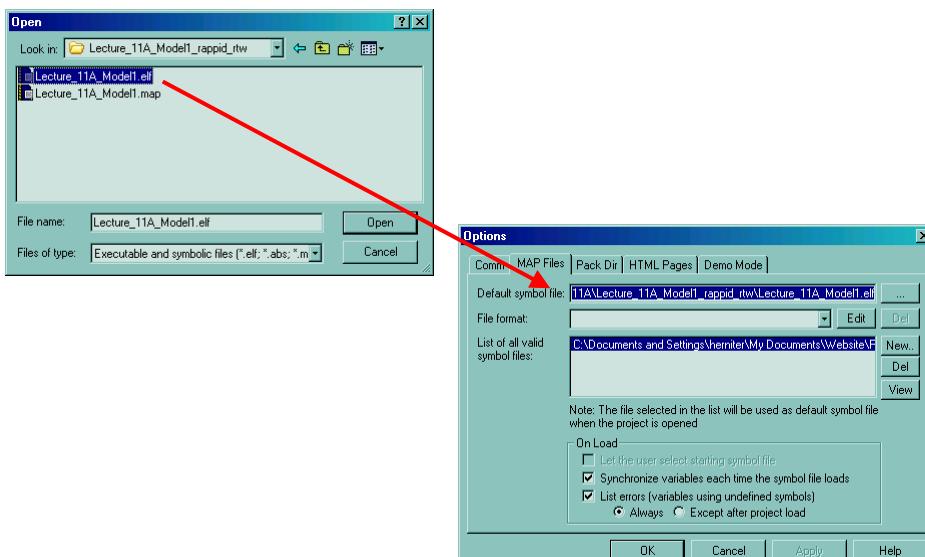
21

- The MAP files are symbol mapping files for our model.
- We are looking for a file with the extension .elf.
- The file is contained in a directory created by Simulink.
- If your model name was “x” then the file would be named “x.elf” and located in directory “x\_rapid\_id\_rtw”.
- This directory is created by Simulink and placed in the working directory.
- Select the .elf file for your model:



## MAP Files

22

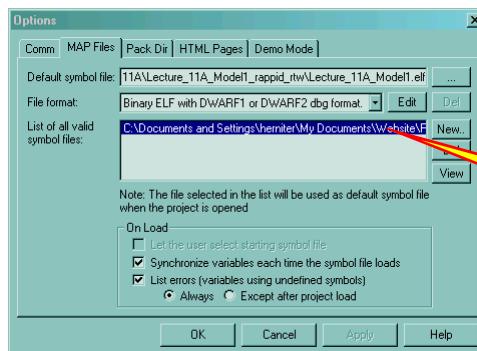




## MAP FIles

23

- Next, select the **File format as Binary ELF with DWARF1 or DWARF2 dbg format.**



File format selected.

- Click the **OK** button when done.



The MathWorks



## FreeMASTER

24

- After clicking the **OK** button, we should be able to establish communication with the MPC555x board.
- In the lower right corner of the FreeMASTER window, the text **RS232;COM1,speed=57600** should be displayed:

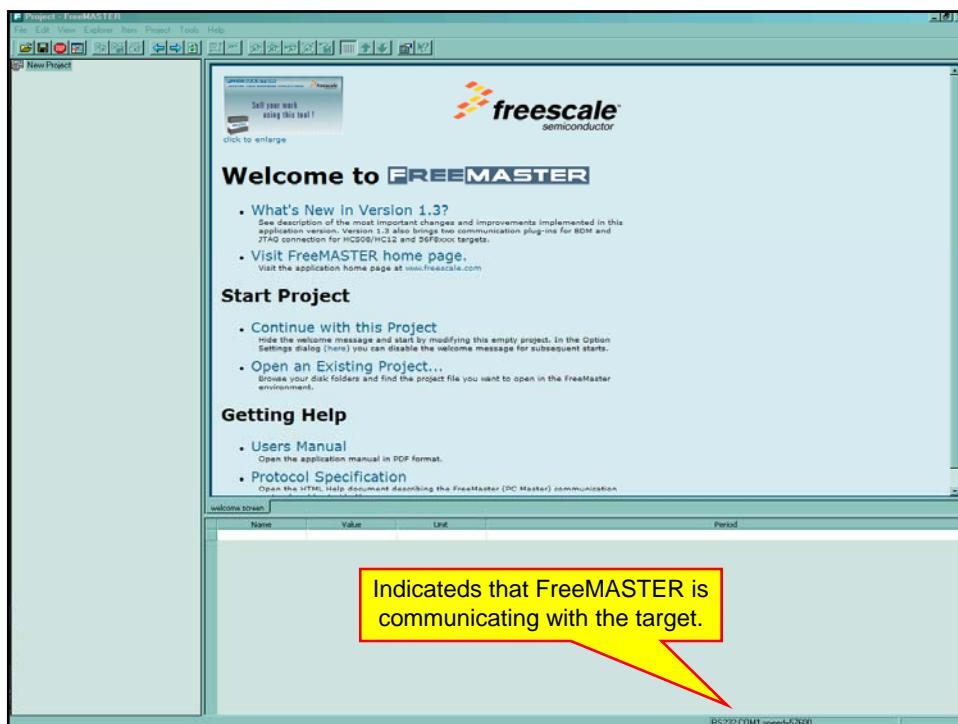


The MathWorks





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



## Communications

26

- If you did not establish communication between FreeMASTER and your MPC555x board:
  - Check the BAUD rate setting of the eSCI\_FMSTR\_Interface block in your Simulink model.
  - Check the Speed settings in FreeMASTER by selecting Project and then Options from the menus.
  - Make sure that you know which port on your computer is COM1.



The MathWorks





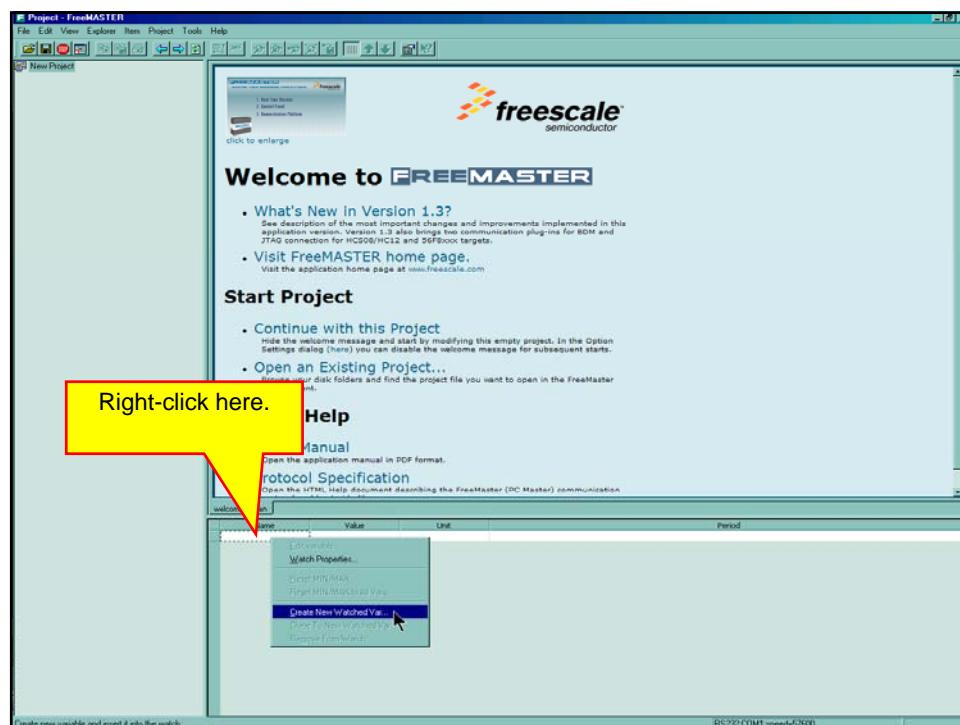
27

## Watched Variables

- Next, we will display the values of signal Repeating\_Signal\_Slow.
- Right click as shown next and select **Create New Watched Variable**:

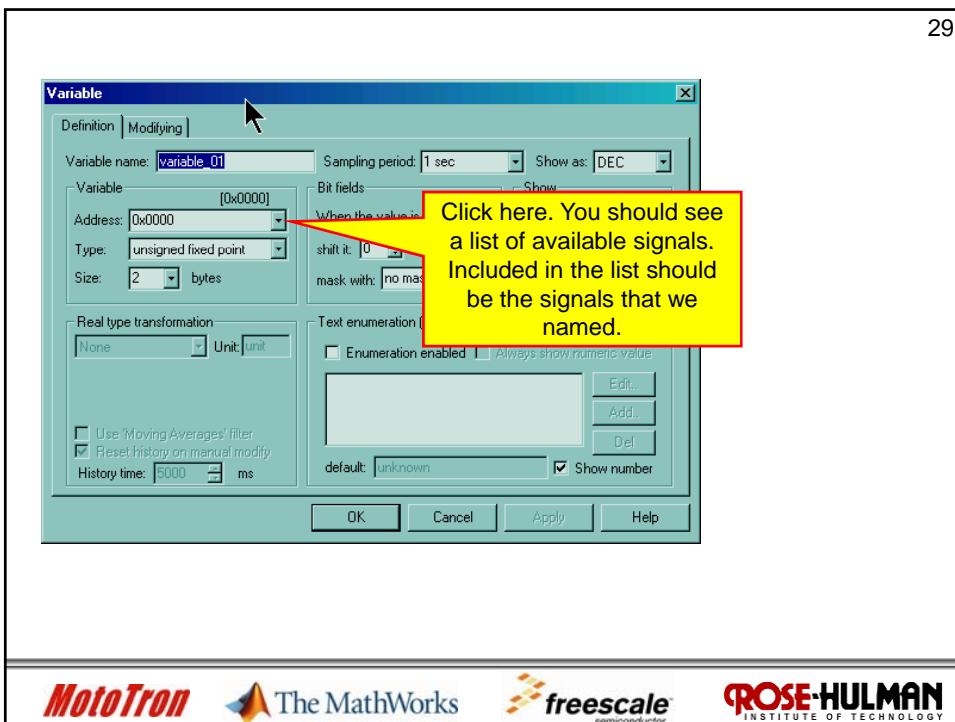


The MathWorks

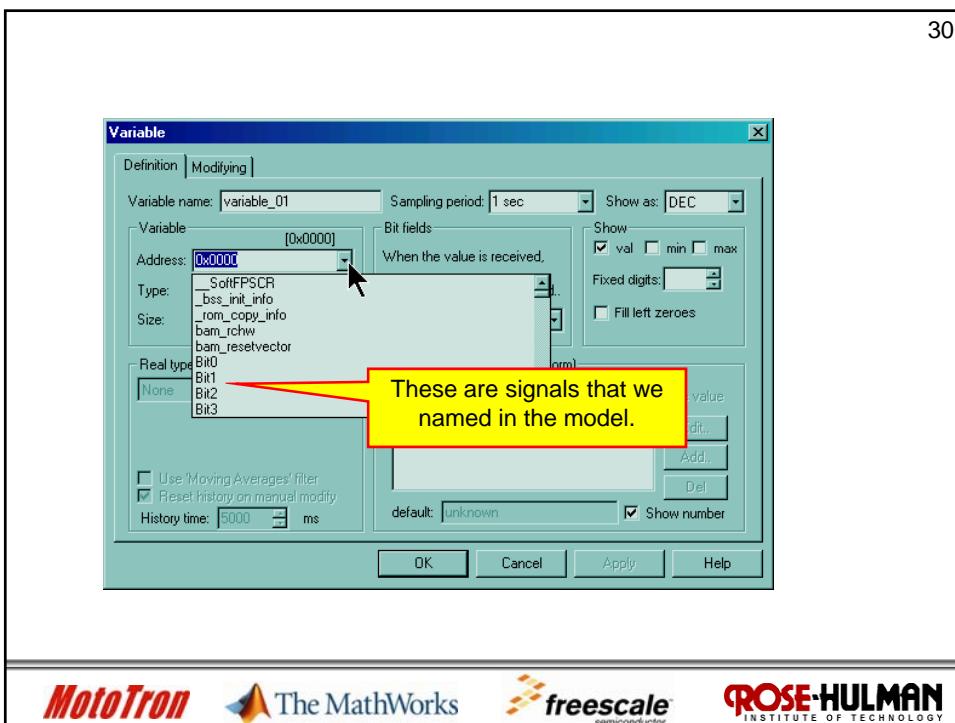





29

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

30

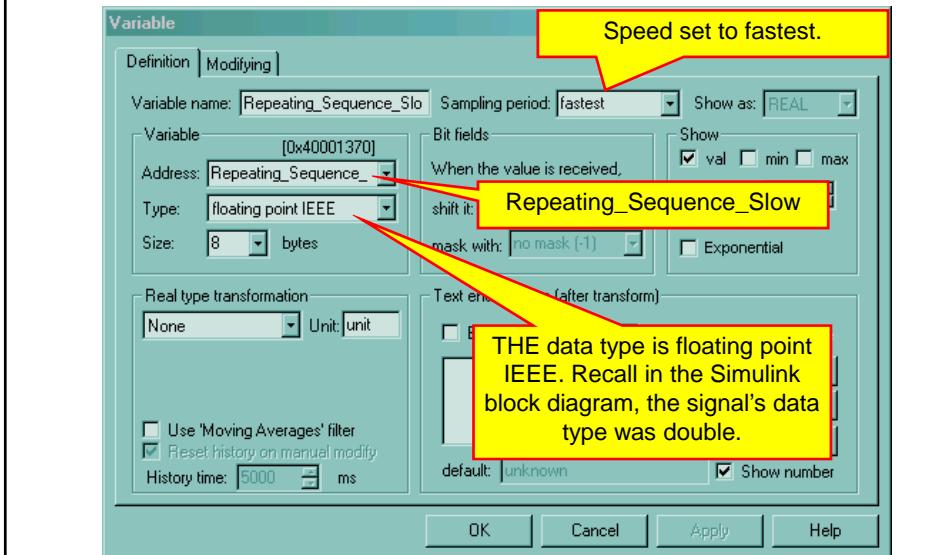
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



31

## Watched Variables

- Make the following selections



32

## Speed Note

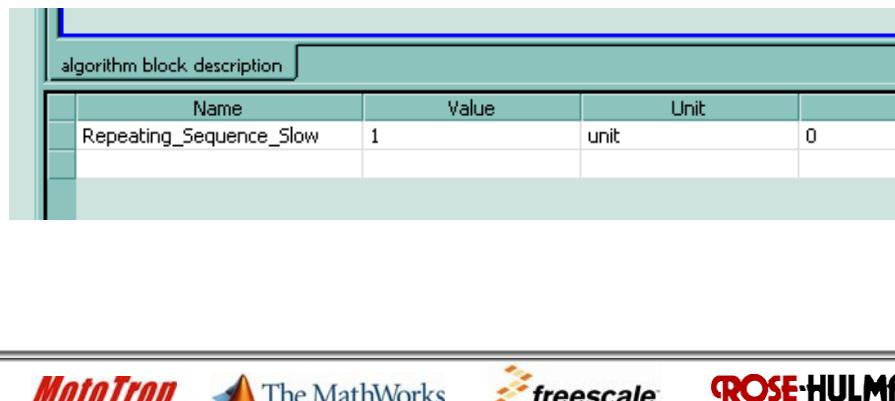
- We have selected the Sampling period to fastest.
- If you recall, the fast repeating sequence had a sampling time of 100 ms.
- To see the waveforms, we need a sampling rate that is about 10 times faster than 100 ms or about 10 ms for each signal we are displaying.
- There are two problems with requiring this fast of a sample rate:
  - The tool might not be able to sample this fast.
  - If we wish to display a lot of the signals, the faster we sample, the more data we need to communicate over the RS232 link.
  - At some point the link will become saturated and we will not be able to view some of the data.



33

## Watched Variables

- After Clicking the **OK** button, you will see the variable and its value displayed.
- The value is updated at the specified rate:

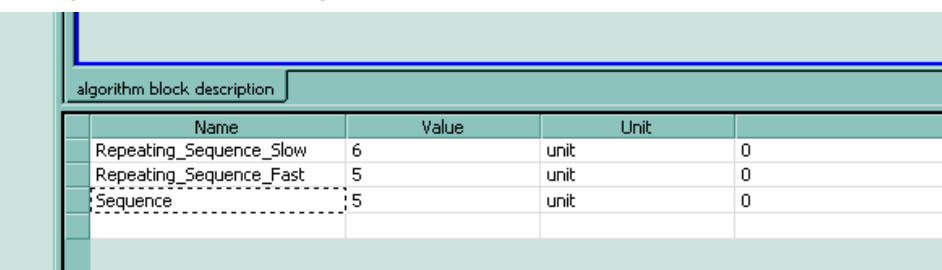


Name	Value	Unit	0
Repeating_Sequence_Slow	1	unit	0

34

## Lecture 11A Demo 1

- Repeat the Procedure to display the values of signals Repeating\_Sequence\_Fast and Sequence:



Name	Value	Unit	0
Repeating_Sequence_Slow	6	unit	0
Repeating_Sequence_Fast	5	unit	0
Sequence	5	unit	0

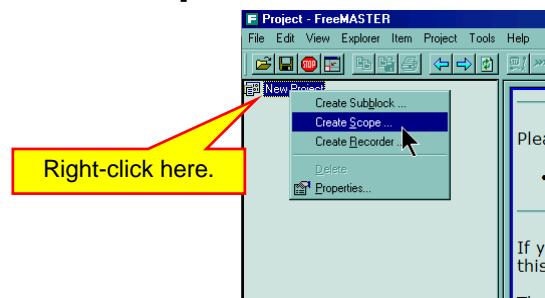
- You should see the fast and slow signals change at different rates, and the Sequence signal should flip-flop between the fast and slow rates.



35

## Scopes

- Next, we would like to generate plots of the three signals.
- Right-click on the text **New Project** and select **Create Scope**



**MotoTron**

The MathWorks

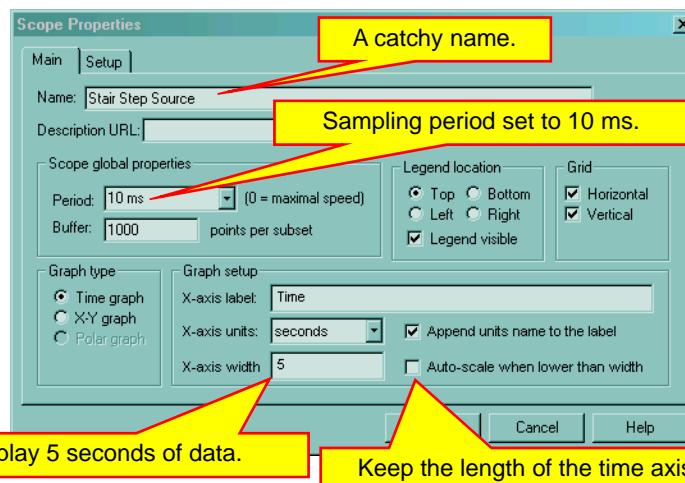
freescale  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

36

## Scopes

- Fill in the **Main** tab of the dialog box as shown:



**MotoTron**

The MathWorks

freescale  
semiconductor

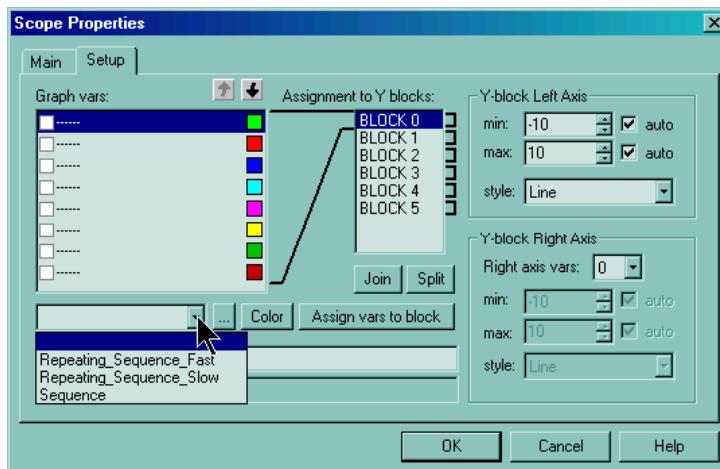
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Scope Properties – Setup Tab

37

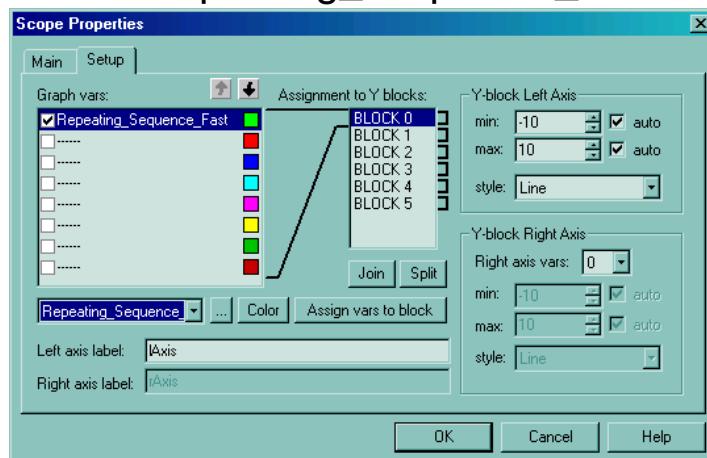
- The setup tab is used to assign watched variables to the plot.
- Click on the down arrow as shown:



## Scope Properties – Setup Tab

38

- Select Repeating\_Sequence\_Fast:



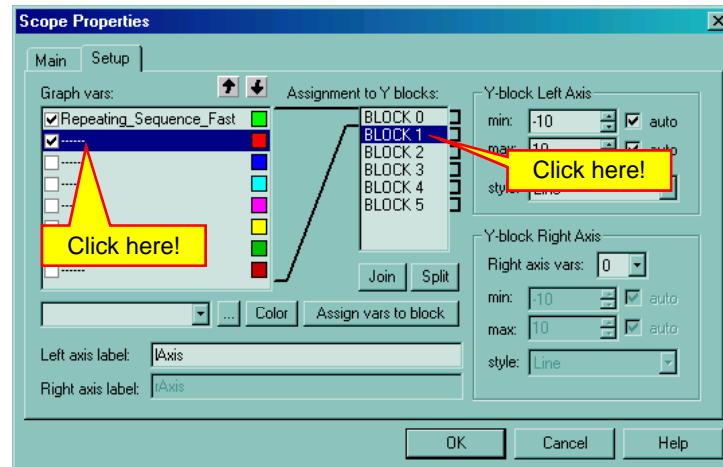
- Repeating\_Sequence\_Fast is displayed in Block 0.



39

## Scope Properties – Setup Tab

- Next, click as shown to select the second trace and Block 1:



**MotoTron**

The MathWorks

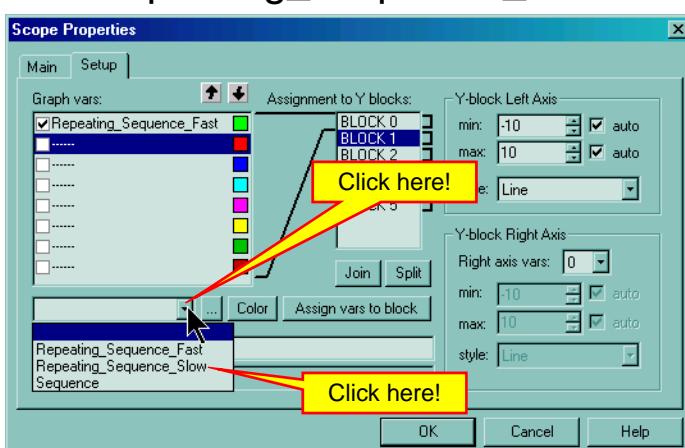
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

40

## Scope Properties – Setup Tab

- Click on the down arrow as shown and then select Repeating\_Sequence\_Slow



**MotoTron**

The MathWorks

**freescale**  
semiconductor

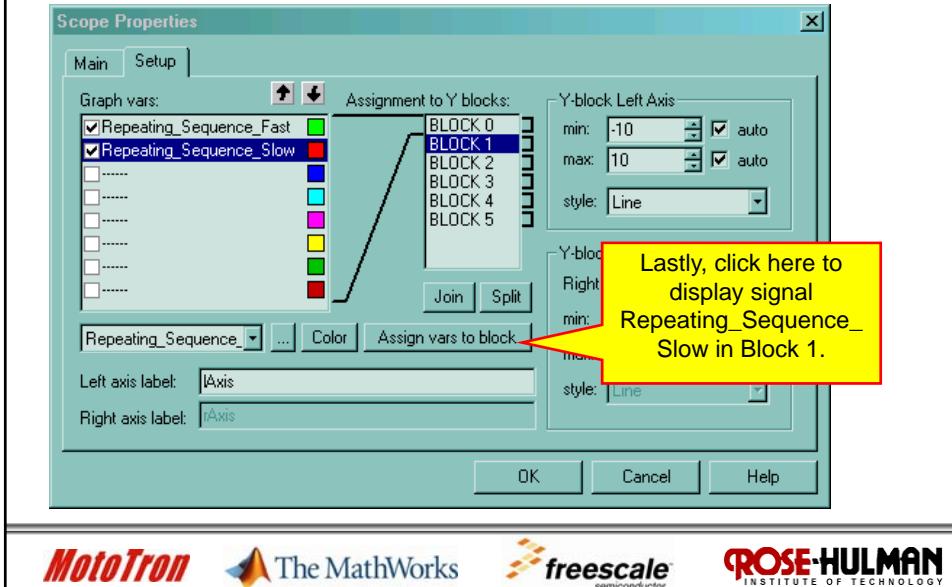
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



41

## Scope Properties – Setup Tab

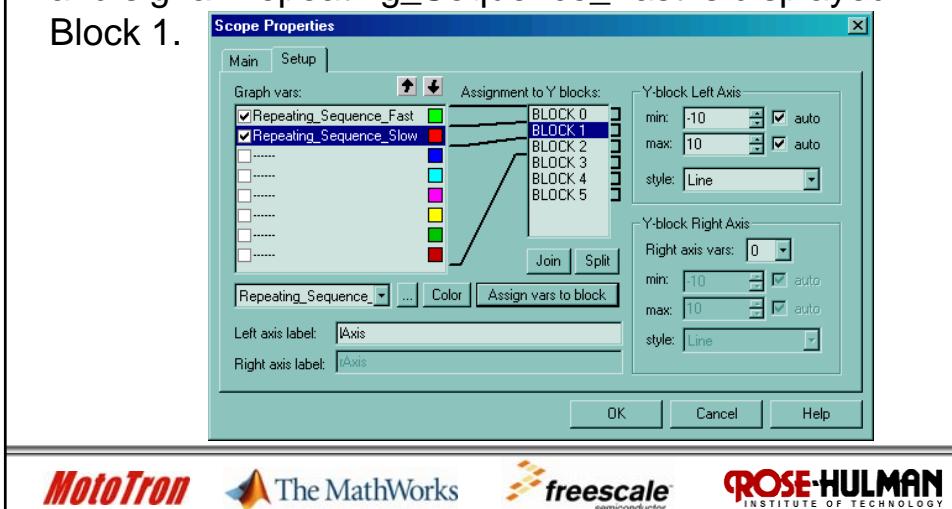
- You should see the screen below:



42

## Scope Properties – Setup Tab

- This screen shows that signal Repeating\_Sequence\_Slow is displayed in Block 0 and signal Repeating\_Sequence\_Fast is displayed in Block 1.





43

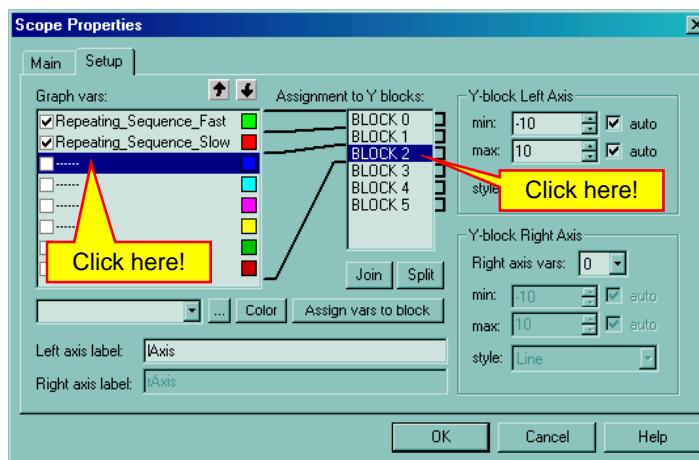
## Scope Properties – Setup Tab

- We will now repeat the procedure to display signal Sequence in Block 3.

## Scope Properties – Setup Tab

44

- Click as shown to select the third trace and Block 2:

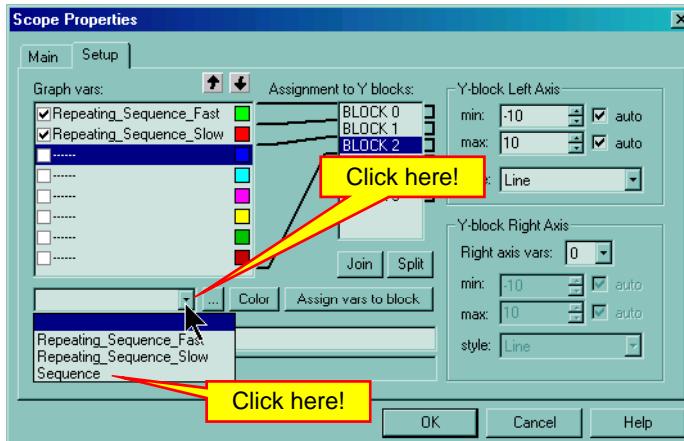




45

## Scope Properties – Setup Tab

- Click on the down arrow as shown and then select Sequence



**MotoTron**

**The MathWorks**

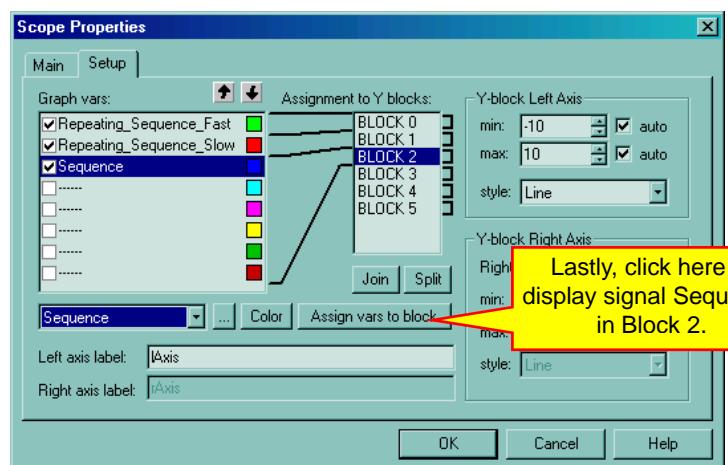
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

46

## Scope Properties – Setup Tab

- You should see the screen below:



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

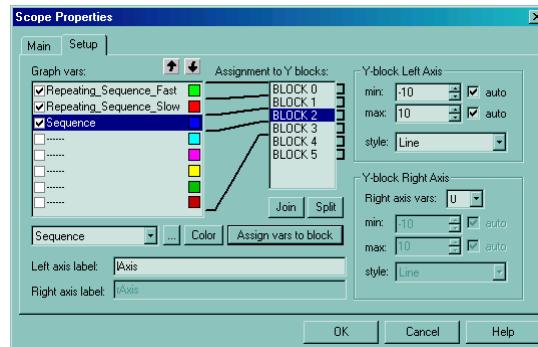
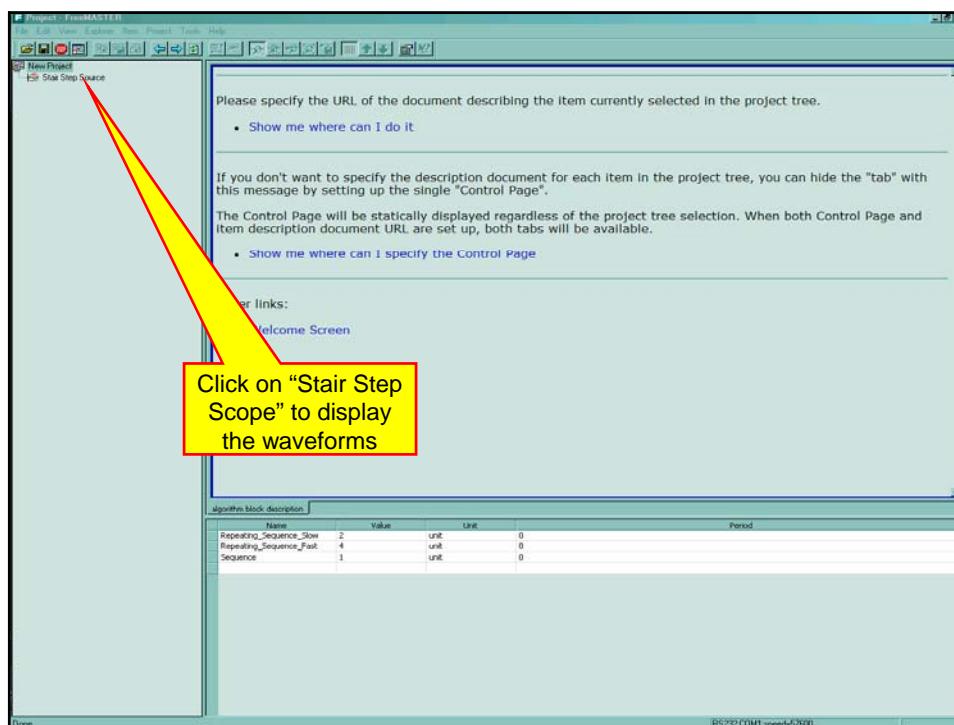
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

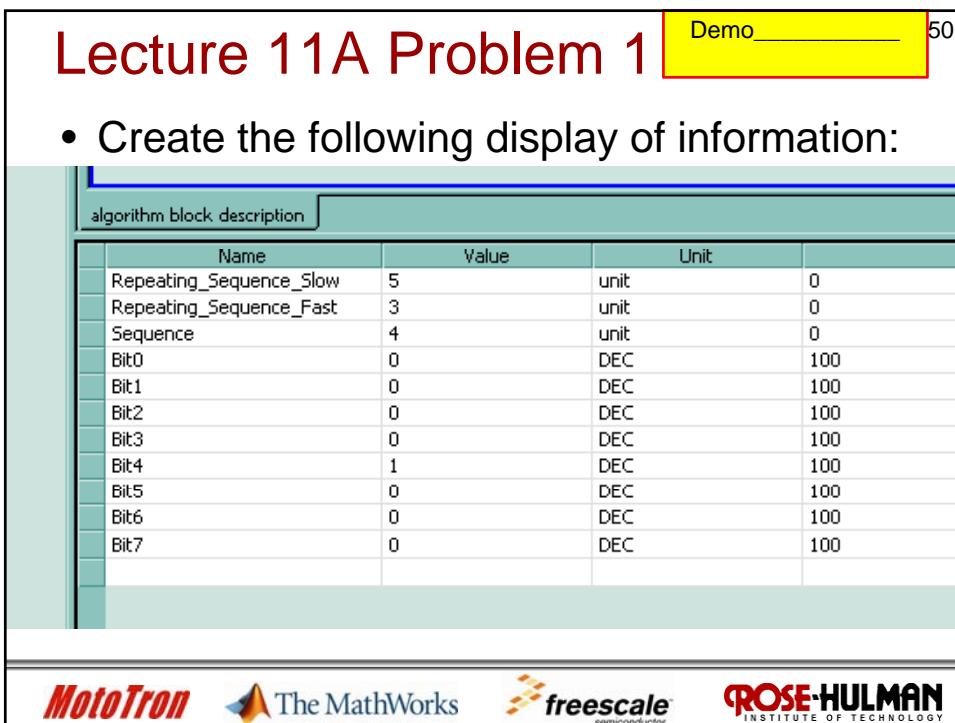
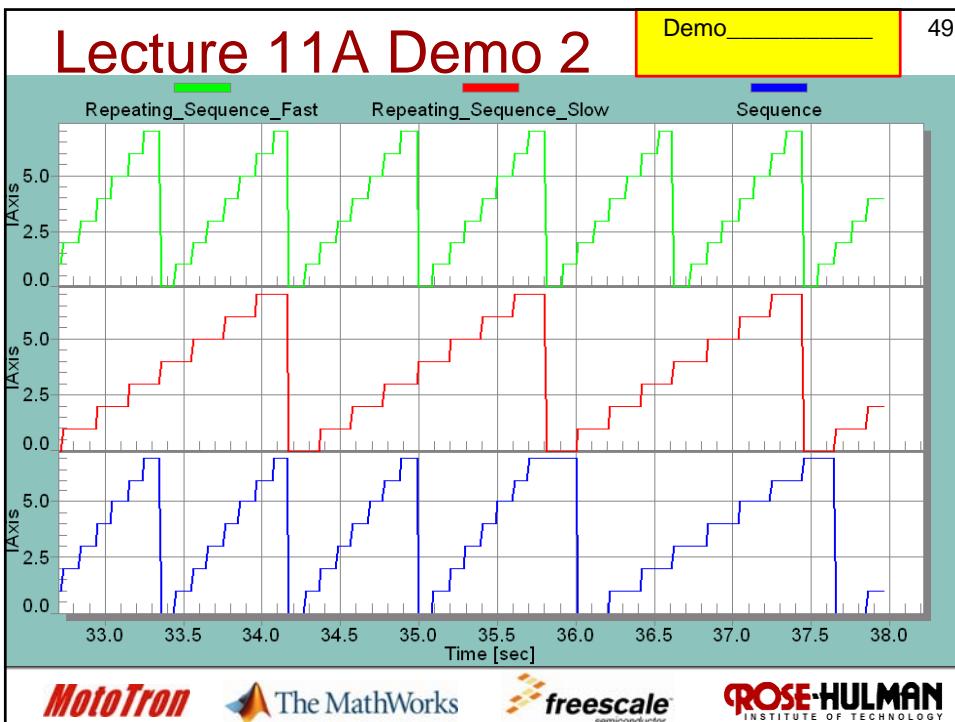


47

## Scope Properties – Setup Tab

- This screen shows that signal Repeating\_Sequence\_Slow is displayed in Block 0, signal Repeating\_Sequence\_Fast is displayed in Block 1, and signal Sequence in Block 2.
- When you click the **OK** button, you will return to the screen shown next:

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



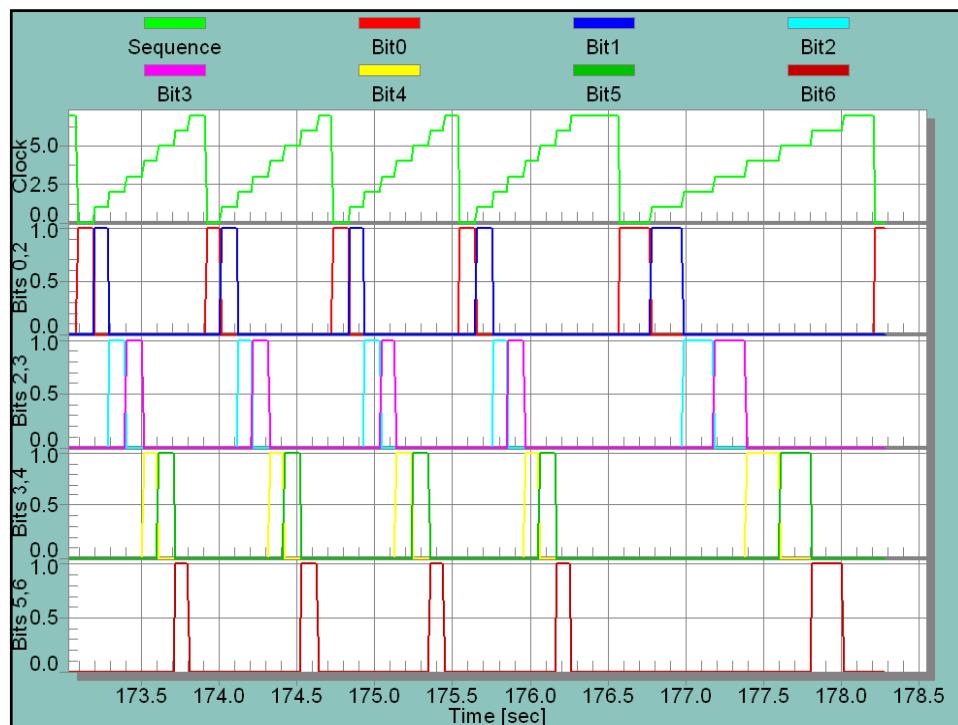
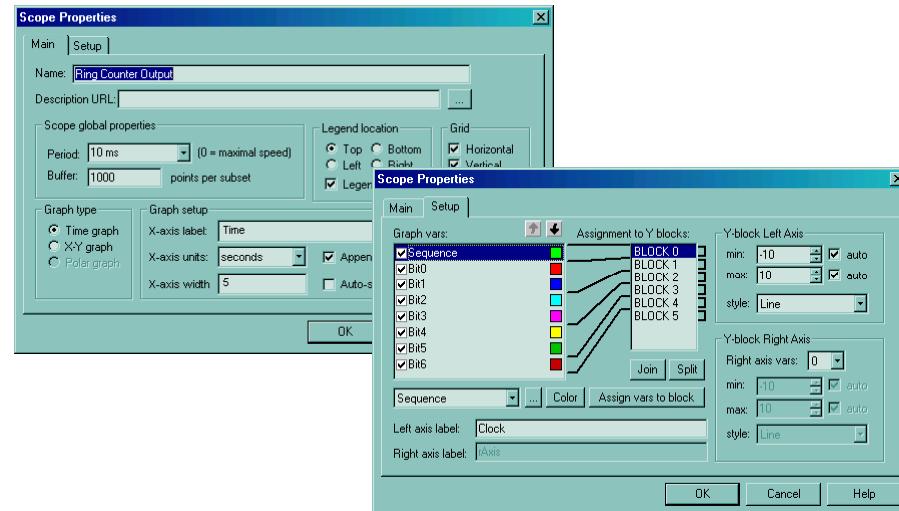


## Lecture 11A Problem 2

Demo\_\_\_\_\_

51

- Create the plot shown on the next page.  
 Note that only bits 0 through 6 are shown.





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

## Any Questions?



The MathWorks





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



# Introduction to Model-Based Systems Design

## Lecture 12: Number Systems



The MathWorks



## Outline

2

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



The MathWorks





3

## Number Systems

- 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.



The MathWorks



## 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



The MathWorks





5

## Binary

- 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**

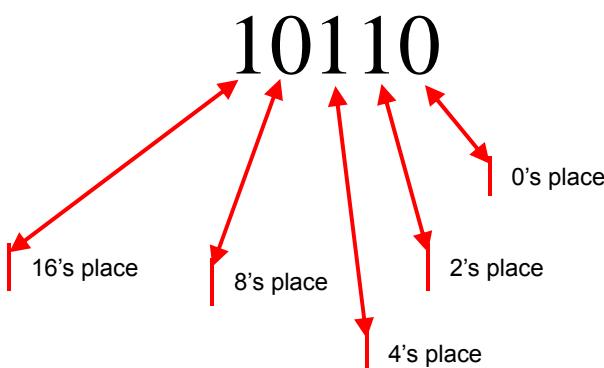
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

6

## Binary



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



7

## Binary

- 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”



8

## Terminology

- 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.



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

## Hexadecimal

11

$$\begin{array}{ccccc} 1 & 0 & 1 & 1 & 0 \\ \underbrace{\quad}_{B} & \underbrace{0 & 1 & 1 & 0}_{6} & \underbrace{1 & 1 & 0 & 1}_{D} & \underbrace{0 & 1 & 1 & 1}_{7} & \underbrace{0 & 0 & 1 & 1}_{3} \end{array}$$

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

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



## 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
```





13

## Useful Matlab Functions

- 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."
  - $123_{16}$  – 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.
 
$$\begin{aligned} &\square 11111111_2 \\ &= 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 \\ &= 2^8 - 1 = 255_{10} \end{aligned}$$
- Uint16 – Unsigned 16-bit integer. Can represent values from 0 to 65535.
 
$$\begin{aligned} &\square 1111111111111111_2 \\ &= 2^{15} + 2^{14} + 2^{13} + \dots + 2^2 + 2^1 + 2^0 \\ &= 2^{16} - 1 = 65535_{10} \end{aligned}$$



The MathWorks





17

## Basic Data Types in Simulink

- Uint32 – Unsigned 32-bit integer. Can represent values from 0 to 4294967295.  

$$\begin{aligned} & \square 11111111111111111111111111_2 \\ & = 2^{31} + 2^{30} + 2^{29} + \dots + 2^2 + 2^1 + 2^0 \\ & = 2^{32} - 1 = 4294967295_{10} \end{aligned}$$



18

## Signed Integers

- 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.
  - $00000 \rightarrow \text{value} = 0 - 15 = -15$
  - $00001 \rightarrow \text{value} = 1 - 15 = -14$
  - $01111 \rightarrow \text{value} = 15 - 15 = 0$
  - $10000 \rightarrow \text{value} = 16 - 15 = 1$
  - $11111 \rightarrow \text{value} = 31 - 15 = 16$





21

## Signed Integers – 2's Complement

- 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  
 $10000000 = -2^7 = -128$
- The most positive number is  
 $01111111 = 127$
- The code for -1 is

$$11111111 = -2^7 + 127$$



The MathWorks



## 2's complement

24

- There is only one representation for 0  
 $- 00000000$



The MathWorks





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.
  - $10000000000000000000000000000000 = -2^{31} = -2147483648$
  - $01111111111111111111111111111111 = 2147483647$



The MathWorks

freescale  
semiconductor

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.



The MathWorks

freescale  
semiconductor



## Floating Point Numbers

27

The screenshot shows the MATLAB help browser with the title 'Floating-Point Numbers'. The content discusses the limitations of fixed-point numbers and how floating-point numbers overcome these by using scientific notation ( $\pm f \times 2^e$ ). It explains the IEEE Standard 754 for single and double precision, including special values like NaN and Inf, and exceptional arithmetic operations. It also covers scientific notation and radix point notation.

**Fixed-Point Numbers**  
 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. Thus, you can represent a range of very large and very small numbers with only a few digits.

You can represent any binary floating-point number in scientific notation form as  $\pm f \times 2^e$ , where  $f$  is the fraction (or mantissa),  $2$  is the radix or base (binary in this case), and  $e$  is the exponent of the radix. The radix is always a positive number, while  $f$  and  $e$  can be positive or negative.

Simulink Fixed Point supports single-precision and double-precision floating-point numbers as defined by the IEEE (Standard 754). Additionally, a nonstandard IEEE-style number is supported.

See the following sections for more information about floating-point numbers:

- [Scientific Notation](#)
- [The IEEE Standard](#)
- [Special Values](#)
- [Exceptional Arithmetic](#)

**Scientific Notation**  
 A direct analogy exists between scientific notation and radix point notation. For example, scientific notation using five decimal digits for the fraction would take the form

$$\pm dd.ddd \times 10^p = \pm dddd.d \times 10^{p+3}$$

where  $p$  is an integer of unrestricted range. Radix point notation using five bits for the fraction is the same except for the number base

$$\pm bb.bbb \times 2^q = \pm bbbb.b \times 2^{q+4}$$

where  $q$  is an integer of unrestricted range. The previous equation is valid for both fixed- and floating-point numbers. For both these data types, the fraction can be changed at any time by the processor. However, for fixed-point numbers the exponent never changes, while for floating-point numbers the exponent can be changed any time by the processor.

For floating-point numbers, the exponent is fixed but there is no reason why the binary point must be contiguous with the fraction. For example, a word consisting of three unsigned bits is usually represented in scientific notation in one of these four ways

$$\begin{aligned} bb.b &= bb.b \times 2^0 \\ bb.b &= bb.b \times 2^{-1} \\ .bb.b &= bb.b \times 2^{-2} \\ .bbb &= bb.b \times 2^{-3} \end{aligned}$$

If the exponent were greater than 0 or less than -3, then the representation would involve lots of zeros.

$$\begin{aligned} bbbb0000. &= bb.b \times 2^3 \\ bbb0. &= bb.b \times 2^1 \\ .000bb &= bb.b \times 2^{-3} \end{aligned}$$

## 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.



The MathWorks





29

## Scientific Notation (Decimal)

- 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.

$$\begin{aligned}\pm d.dddd \times 10^p &= \pm dddd.0 \times 10^{p-4} \\ &= \pm 0.dddd \times 10^{p+1}\end{aligned}$$

Decimal point.



The MathWorks

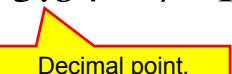


## Binary Point

30

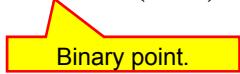
- Binary numbers can have a fractional part just like decimal numbers:

$$73.84 = 7 \times 10^1 + 3 \times 10^0 + 8 \times 10^{-1} + 4 \times 10^{-2}$$



$$101.11 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$$

$$101.11 = (1 \times 4) + (0 \times 2) + (1 \times 1) + (1 \times \frac{1}{2}) + (1 \times \frac{1}{4}) = 5.75$$



The MathWorks



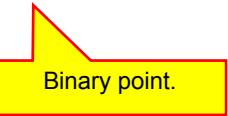


31

## Radix Point Notation (Binary)

- 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.

$$\pm b.bbbb \times 2^p = \pm bbbbb.0 \times 2^{p-4}$$



Binary point.

$$= \pm 0.bbbbb \times 2^{p+1}$$

## IEEE Floating Point Standard 754

32

- Single Precision – 32 bits



$$Value = \begin{cases} (-1)^s \cdot (2^{e-127}) \cdot (1.f) & ; \text{normalised}, 0 \leq e \leq 255, f \geq 0 \\ (-1)^s \cdot (2^{e-126}) \cdot (0.f) & ; \text{denormalised}, e = 0, f > 0 \\ \text{exceptional value} & \end{cases}$$

- Exceptional values: NaN, inf.



33

## 32-bit Floating point example

- $\begin{array}{cccccc} 1 & \underline{10111101} & \underline{10100000000000000000000000000000} \\ s & e & f \end{array}$
- $s = 1 \rightarrow$  we have a negative number.
- $e = 10111101 = 189_{10}$
- $\rightarrow (2^{e-127}) = (2^{189-127}) = (2^{62})$
- $f = 10100000000000000000000000000000$
- $1.f = 1.10100000000000000000000000000000$   
 $= (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) + (0 \times 2^{-4}) + (0 \times 2^{-5}) + \dots$   
 $= 1.625$



34

## 32-bit Floating Point Example

- Our number is
- $-1.625_{10} * 2^{62}$
- $= -7.493989779944505 \times 10^{18}$  (decimal)





35

## IEEE Floating Point Standard 754

- 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.



36

## Floating Point Numbers

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.





37

## Floating Point Numbers

- 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:

$$\infty - \infty$$

$$\infty + \infty$$

$$0 \times \infty$$

$$0/0$$

$$\infty / \infty$$

38

## Lecture 12 Problem 1

- Determine the decimal value of the following bit string (it is 32 bits in length.)
- 1001110011101100110111000000011
- Assuming the following data types
  - UInt32 (magnitude)
  - Int32 (2's complement)
  - Sign and magnitude
  - Single precision floating point
- Create an m-file that displays all four results in an mbox.



The MathWorks





39

## Lecture 12 Problem 2

- Determine the decimal value of the following bit string (it is 32 bits in length.)
- 010000011101000011010000000011
- Assuming the following data types
  - Uint32 (magnitude)
  - Int32 (2's complement)
  - Sign and magnitude
  - Single precision floating point
- Create an m-file that displays all four results in an mbox.



40

## Lecture 12 Problem 3

- Create an m-file that defines a arbitrary 32 character text string, the contents of which are zeros and ones. For example,  
 '010000011101000011010000000011'
- The script then displays the value of the bit string in an mbox, assuming the four data types below:
  - Uint32 (magnitude)
  - Int32 (2's complement)
  - Sign and magnitude
  - Single precision floating point
- Demonstrate your function with a binary value given by your instructor.





41

## Lecture 12 Exercise 1

- Create an counter that changes state every  $\frac{1}{2}$  second.
- Do not use Stateflow to do this.
- The repeating LED output sequence is shown below:

0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0	0 1 1 1 1 1 1 1
1 1 0 0 0 0 0 0	0 0 1 1 1 1 1 1
1 1 1 0 0 0 0 0	0 0 0 1 1 1 1 1
1 1 1 1 0 0 0 0	0 0 0 0 1 1 1 1
1 1 1 1 1 0 0 0	0 0 0 0 0 1 1 1
1 1 1 1 1 1 0 0	0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 1

Demo \_\_\_\_\_



42

## Lecture 12 Exercise 2

- Create an counter that changes state every  $\frac{1}{2}$  second.
- Do not use Stateflow to do this.
- The repeating LED output sequence is shown below:

0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0	1 1 1 1 1 1 0 0
1 1 0 0 0 0 0 0	1 1 1 1 1 0 0 0
1 1 1 0 0 0 0 0	1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0	1 1 1 0 0 0 0 0
1 1 1 1 1 0 0 0	1 1 0 0 0 0 0 0
1 1 1 1 1 1 0 0	1 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0	0 0 0 0 0 0 0 0

Demo \_\_\_\_\_





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

## Questions?



The MathWorks





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

# Introduction to Model-Based Systems Design

## Lecture 13: MPC555x Digital Input



## MPC555x Digital Output

2

- The digital output blocks work about the same as the digital input blocks.
- These blocks read a 1-bit digital input.
- The output of the block is a Boolean data type.
- Use block **General Purpose Input** located in library **RAppID-Toolbox / GPIO Blocks**.

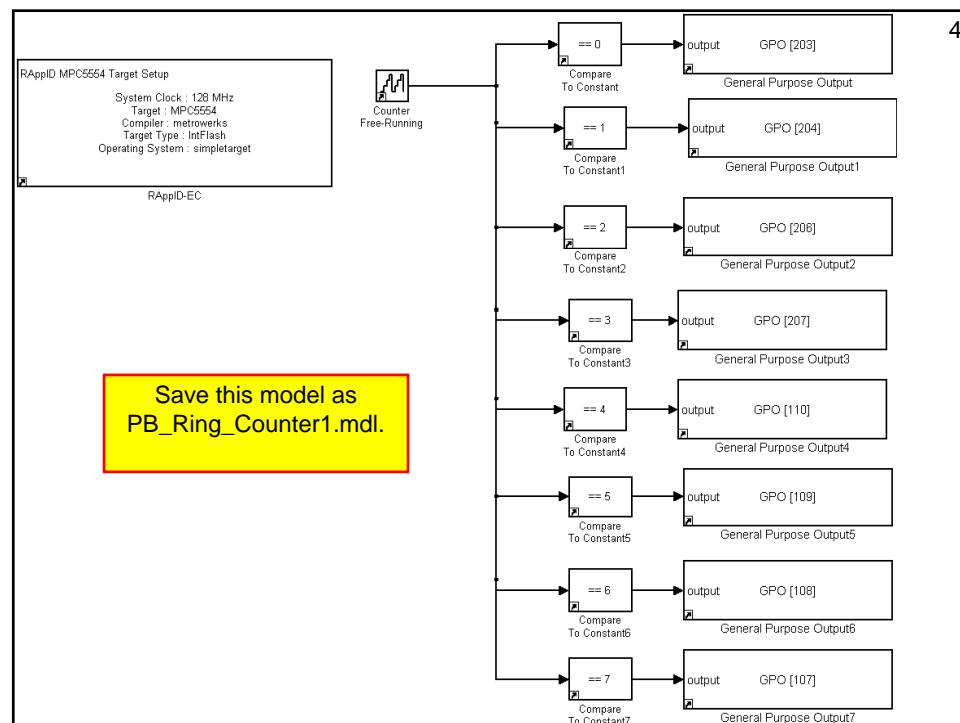




## MPC555x Digital Input

3

- We will modify the ring counter we used before.
- When a push-button is pressed, the counter will shift the bit to the right.
- When the push-button is not pressed, the counter will shift to the left.
- (Right and left are relative. At least I did not say clockwise.)
- We will start with the ring counter shown next.

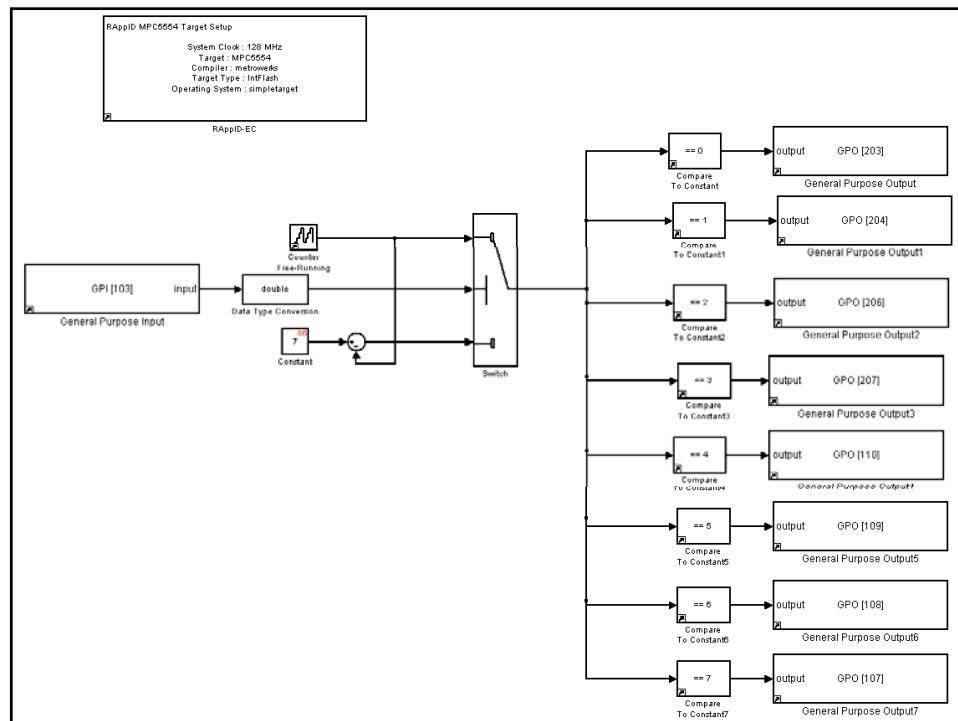




5

## MPC555x Digital Input

- Place the **General Purpose Input** block in your model.
- Select an input channel.
- Add the remainder of the blocks shown.





7

## MPC555x Digital Input

- After the convert block, the Digital Input Block will output a numerical value of either 0 or 1.
- The threshold of the switch is set to 0.5 so that there is no ambiguity in the switch state.
- Use file MPC5554DEMO\_man\_G.pdf to determine the correct pin connection for your chosen digital input.



8

## Push-Button Switches

- The MPC555x Demo boards have four push-buttons for your use.
- Page 7 (Section User Components / User\_Switch) of the User's Manual specifies the properties and pins for the push-buttons:

PIN #	USER COMPONENT CONNECTION
1	SW1 out, de-bounced CMOS drive 0 or 3.3V, active low.
2	SW1 out, de-bounced Open Drain output, active low, 10K ohm pull-up to 3.3V. Suitable for IRQ input signal drive.
3	SW2 out, active low, 10K ohm pull-up to 3.3V.
4	SW3 out, active low, 10K ohm pull-up to 3.3V.
5	SW4 out, active low, 10K ohm pull-up to 3.3V.
6	SPEAKER amp input. 0 to 5V/pp, volume adjust with SPKR_VOL.
7	RV1 center tap, 0 – 5V adjustment
8	RV2 center tap, 0 – 5V adjustment





9

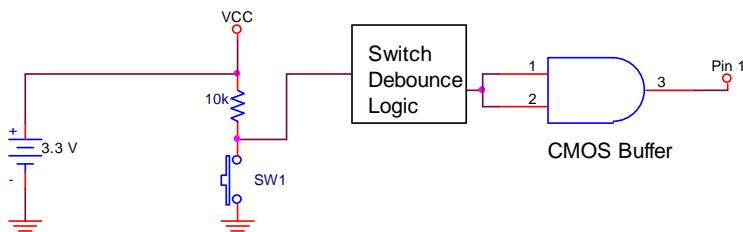
## Push-Button 1

- This switch has a de-bounced output. This means that when you press or release the button, you only get a single edge at the output pin.
- Think of a toggle switch as two metal balls being flipped back and forth by a spring. When the two metal balls collide, they will mechanically bounce apart repeatedly until the kinetic energy is dissipated. When the balls are touching, the switch is closed. When balls are separated, the voltage dips down. As the balls mechanically bounce, the voltage bounces up and down as well.

10

## Push-Button 1

- CMOS Output (Pin 1) – Can drive a load.

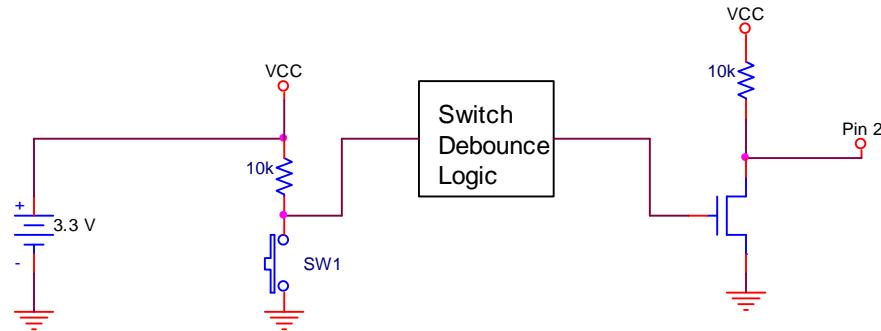




## Push-Button 1

11

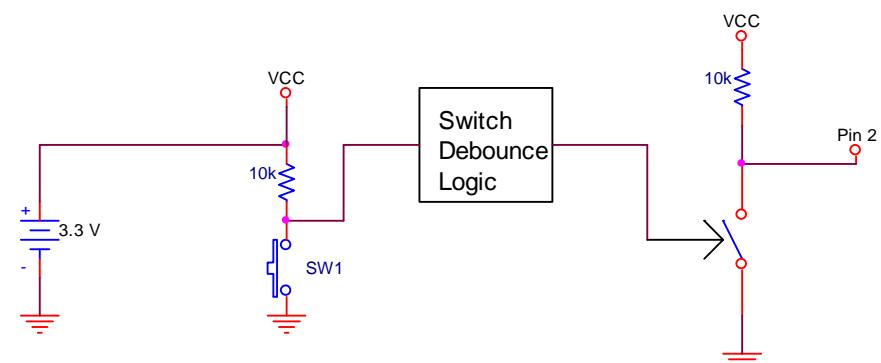
- Open Drain Output (Pin 2) – Cannot drive a large load.



## Push-Button 1

12

- The open drain output is logically equivalent to the following circuit:

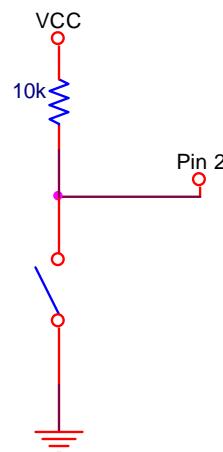




13

## Push-Button 1 (Pin 2)

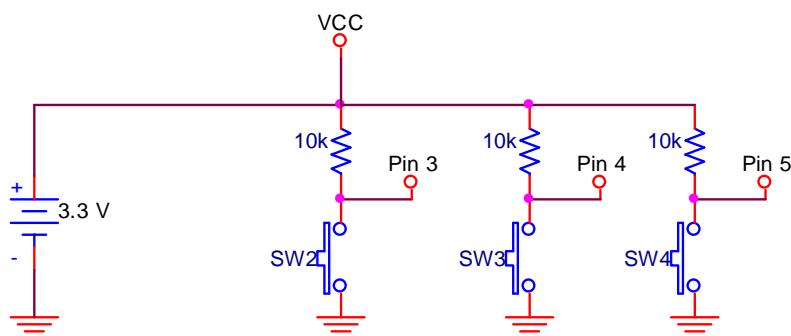
- When the switch is closed, pin 2 goes to zero volts.
- When the switch is open, pin 2 is pulled up to Vcc volts through the 10k resistor.



14

## Push-Buttons 2,3,4 (Pins 3,4,5)

- These switches are shown below.
- Note that the switches are not de-bounced.

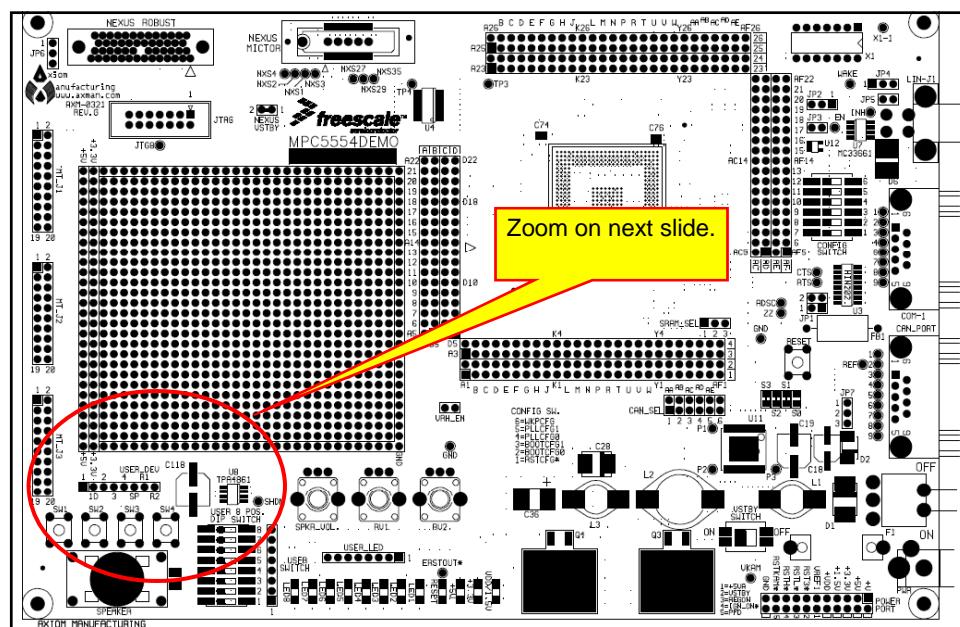


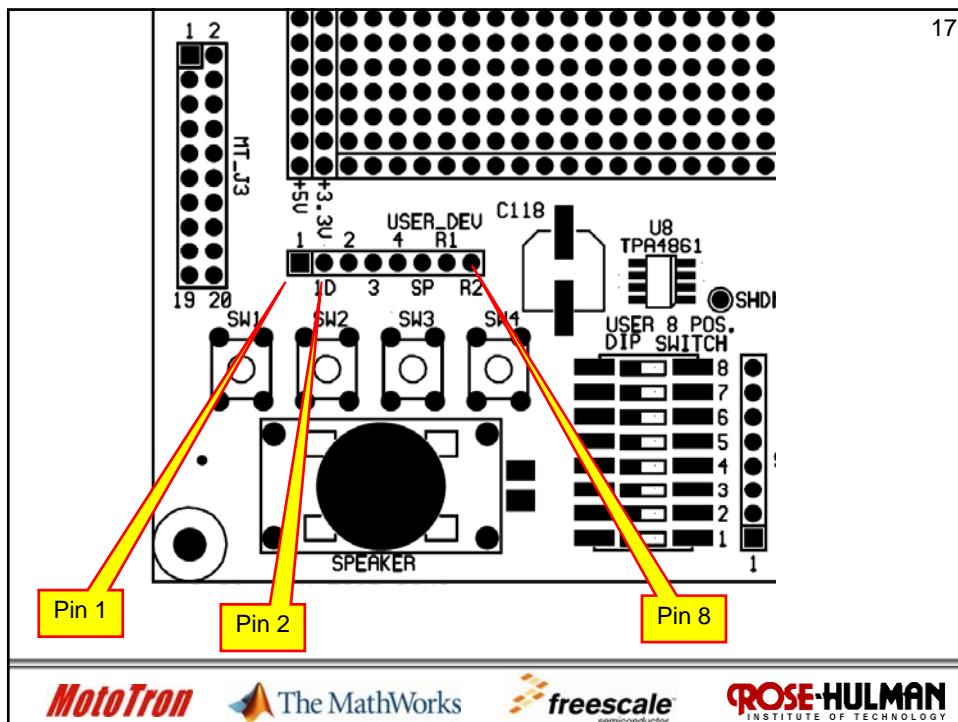


15

## Push-Button Switches

- We can use any of the 4 switches (pins 1 through 5) as our digital inputs.
- The physical location of the switch connections can be found using file MPC5554DEMO\_TOP\_F-G.pdf.
- The location of the USER\_DEV jumpers is shown next:





## Lecture 13 – Demo 1

- Wire up your circuit.
- Modify the model to display the value of the digital input and the count using the FreeMaster tool.
- Compile and download the model.
- Demo your working system.
  - The ring counter should change direction when the push-button is pressed.

Demo \_\_\_\_\_





19

## Ring Counter Problem

- The preceding example has a problem in that when you change the direction of the counter, the illuminated LED may jump several places depending on the count.
- We will correct this problem by creating our own counter.
- We will first create a counter that shifts in one direction, but holds and remembers its position when a push-button is pressed.



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Triggered Subsystems

20

- We will use a triggered subsystem.
- Triggered subsystems give us a method to allow different parts of our program to run at different rates.
- We will create the system shown next. This looks pretty close to the previous ring counter except that we added a triggered subsystem:

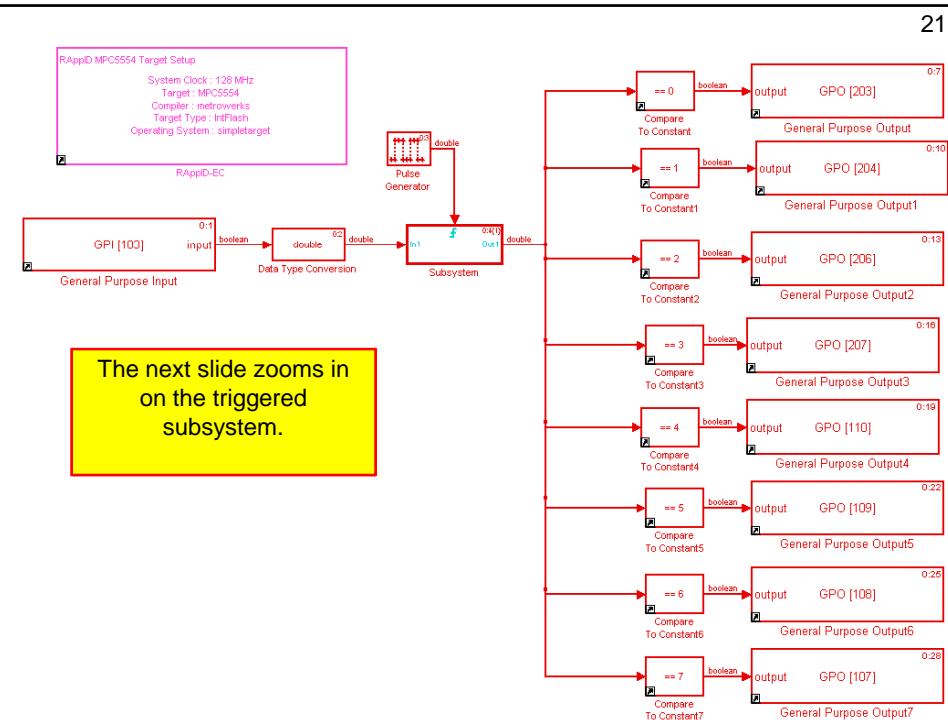


The MathWorks

freescale<sup>®</sup>  
semiconductor

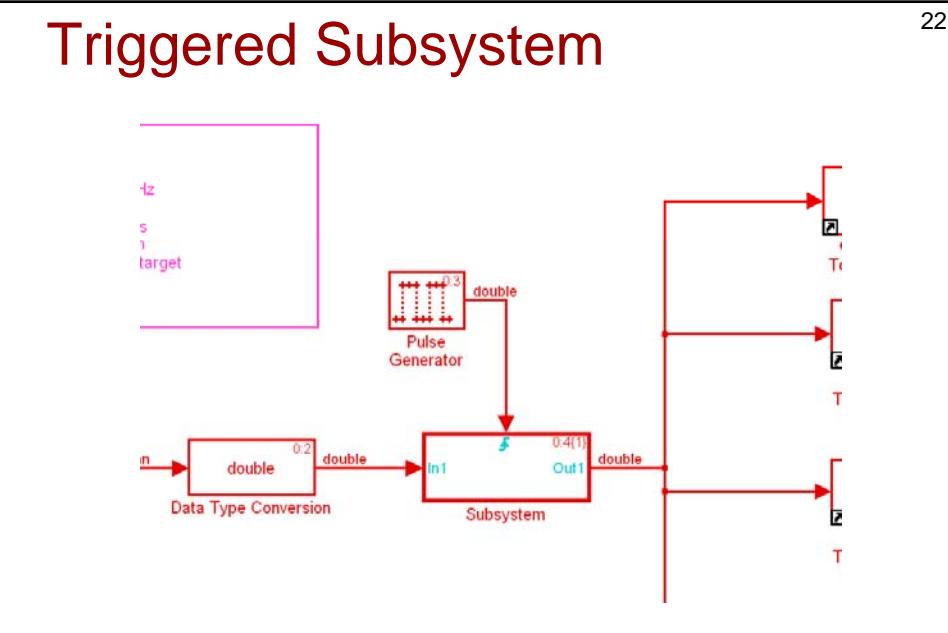


21



## Triggered Subsystem

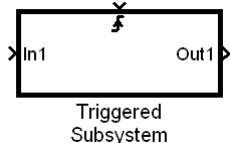
22





23

## Triggered Subsystems



- The contents of a triggered subsystem executes once every trigger.
- The above subsystem only executes when it gets positive edge.
- We can control how often the subsystem executes by controlling the trigger.



24

## Triggered Subsystem

- In our example, the entire model is executed once every 0.001 seconds, as this the fixed time step we specified in the simulation setup.
- The triggered subsystem executes once every 0.5 seconds, as this is the frequency we will specify for the pulse generator.
- (We will use the triggered subsystem to increment our counter once every 0.5 seconds.)





## Triggered Subsystems

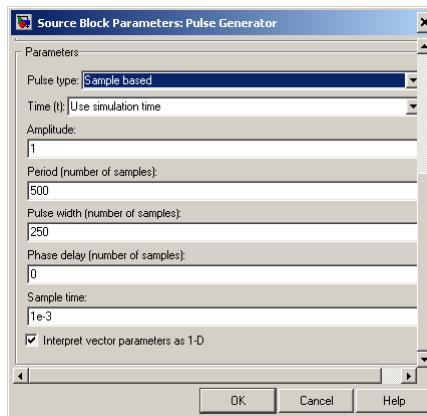
25

- There are two ways to create a triggered subsystem:
  - Use the **Triggered Subsystem** block located in the **Simulink / Ports & Subsystems** library.
  - Use a standard Subsystem block. Inside the subsystem place a **Trigger** block (also located in the **Simulink / Ports & Subsystems** library).
  - With the trigger block, you can specify a rising edge, falling edge, both rising and falling edges, or a function call trigger.

## Triggered Subsystem

26

- The pulse generator in our example is set to generate a rising edge every 0.5 seconds (period set to 500 samples).

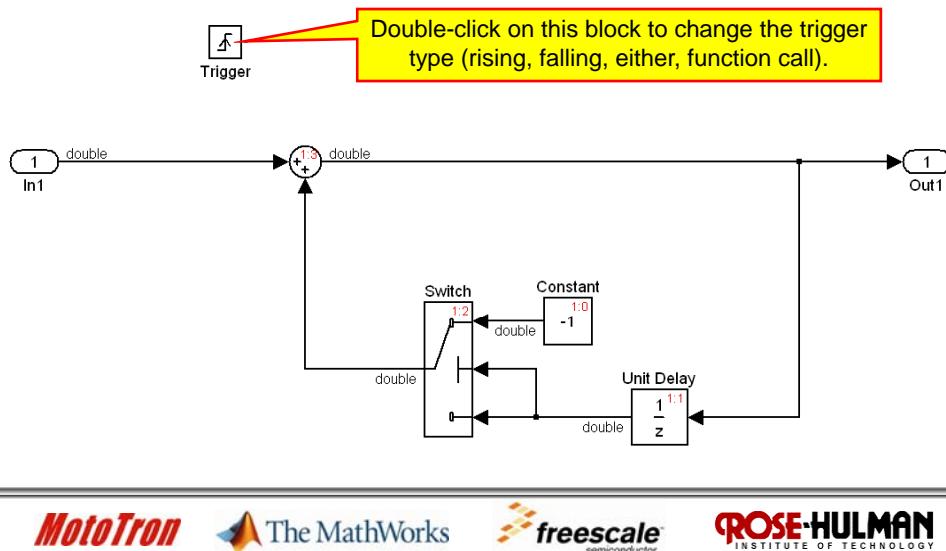




27

## Triggered Subsystem

- The contents of the subsystem are shown below:



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

28

## Subsystem Operation

- The input to the subsystem is numerically either 0 or 1.
- The  $\frac{1}{z}$  block is a one simulation time step delay. The output of the block is the input from the previous time step.
- The Simulink model below would be a counter that counts up at the rate specified by the trigger:

**MotoTron**

**The MathWorks**

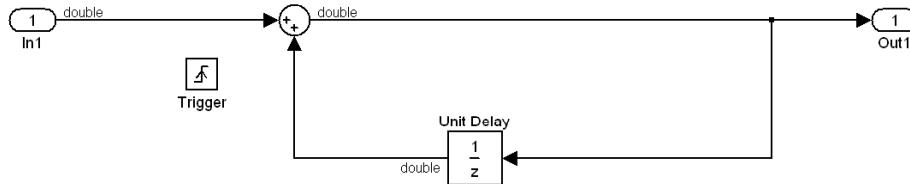
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Counter

29

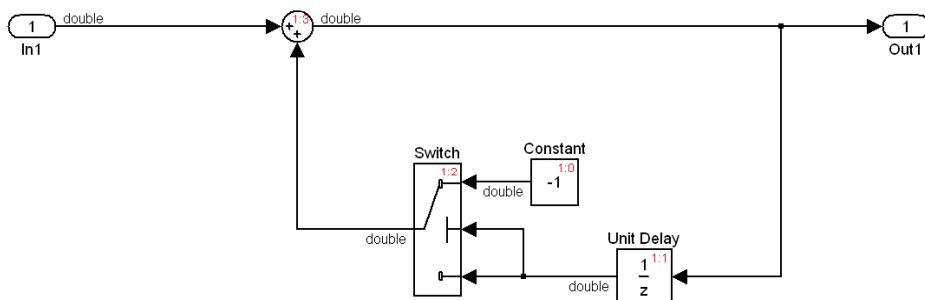


- When the input is a 1, the counter counts up every time the subsystem receives a rising edge.
- When the input is a 0, the counter holds.

## Mod 8 Counter

30

- In our system, when the count is above 6.5, we set the input to the sum block to -1 so that we get 0 when we add 1 to it:





31

## Mod 8 Counter

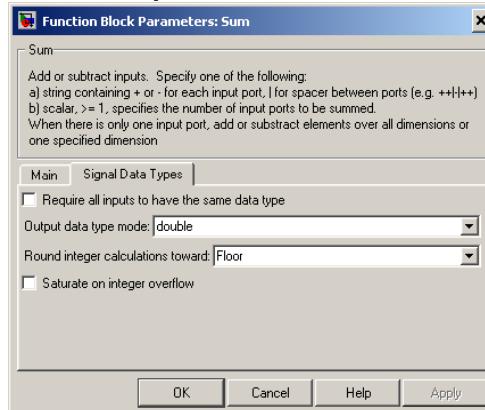
- Some details about the blocks are shown next.



32

## Sum Block

- The **Output data type mode** of the Sum block should be set to double. If you do not do this, Simulink may not be able to determine the data types due to the feedback loop:

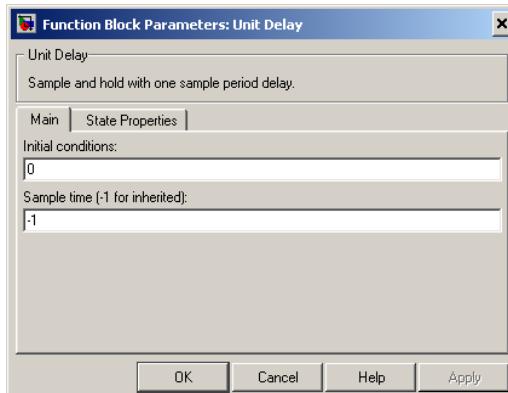




33

## Unit Delay Block

- The sample time of the block should be set to -1 (inherited) and the initial condition should be set to zero.



**MotoTron**

**The MathWorks**

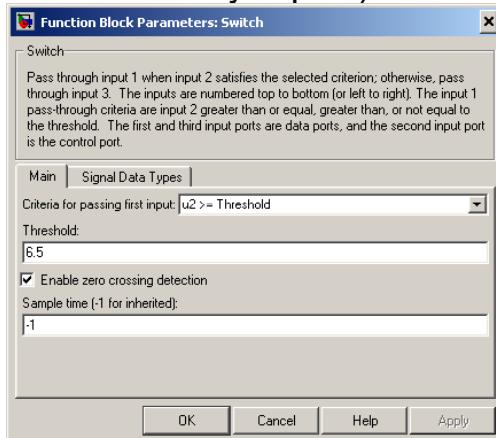
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

34

## Switch Block

- The threshold of the Switch block should be set to 6.5. (We did not choose a threshold of 7 because you can sometimes have trouble determining if two floating point numbers are numerically equal.)





## Model Sample Times

35

- If you show sample colors on your model (select **Format**, **Port/Signal Displays**, and then **Sample Time Colors** from the Simulink menus) you will see that the subsystem executes at a different rate than the main portion of the model.
- The main model is shown in red indicating that it has the fastest sample time.
- The subsystem is shown in cyan, indicating that it is a triggered subsystem:

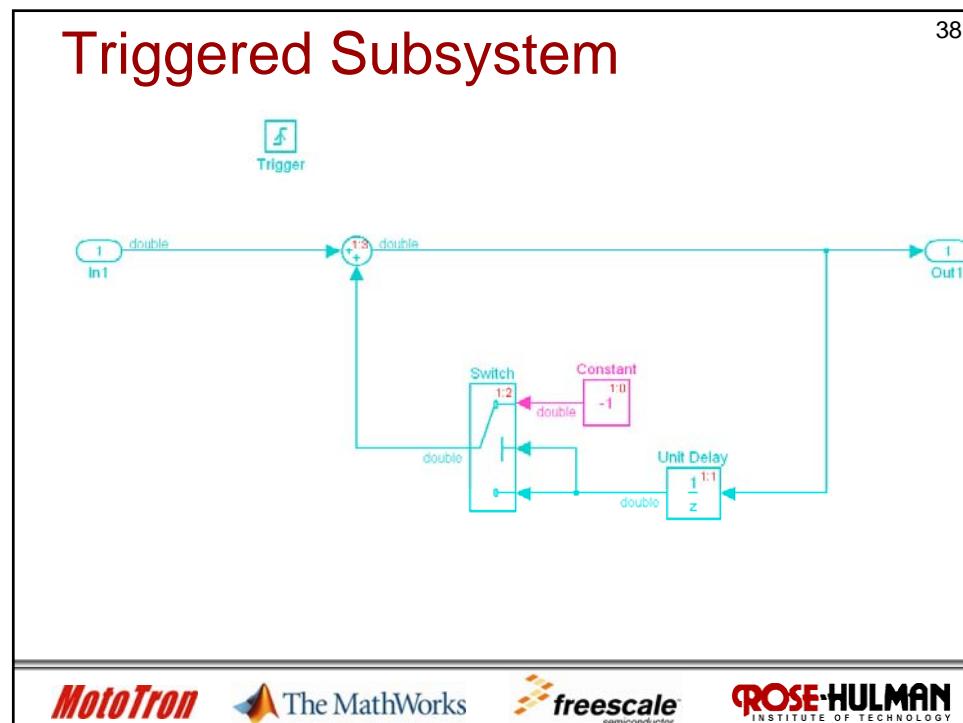
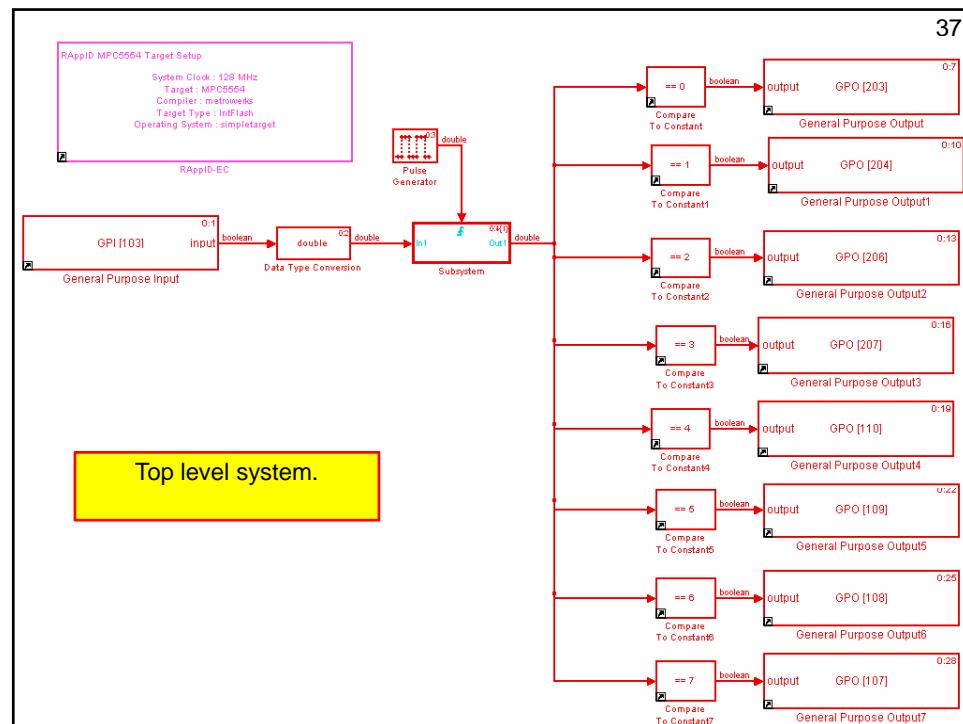


## Sample Time Colors

36

Color	Use
Black	Continuous sample time
Magenta	Constant sample time
Red	Fastest discrete sample time
Green	Second fastest discrete sample time
Blue	Third fastest discrete sample time
Light Blue	Fourth fastest discrete sample time
Dark Green	Fifth fastest discrete sample time
Orange	Sixth fastest discrete sample time
Yellow	Can indicate one of the following: <ul style="list-style-type: none"> <li>• A block with hybrid sample time, e.g., subsystems grouping blocks and Mux or Demux blocks grouping signals with different sample times, <a href="#">Data Store Memory</a> blocks updated and read by different tasks.</li> <li>• Variable sample time. See the <a href="#">Pulse Generator</a> block and <a href="#">Specifying Sample Time</a> for more information.</li> <li>• A block with the seventh, eighth, etc., sample time.</li> </ul>
Cyan	Blocks in triggered subsystems
Gray	Fixed in minor step







Demo\_\_\_\_\_

39

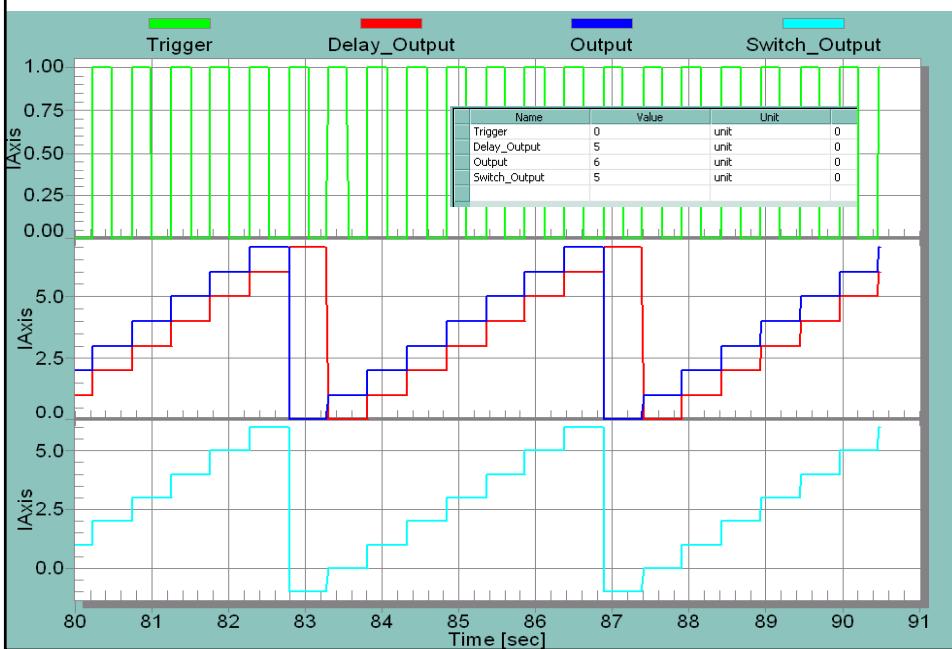
## Lecture 13 - Demo 2

Mod 8 counter using a triggered subsystem.

- Wire up your circuit.
- Use the FreeMASTER tool to monitor the following signals for the triggered subsystem:
  - The input.
  - The output
  - The switch output
  - The output of the  $1/z$  block
  - The trigger
  - An example display is shown on the next slide.
- Compile and download the model.
- Demo your working system.
- The counter should hold when you press the pushbutton.

## Lecture 13 - Demo 2

40





41

## Lecture 13 Exercise 1

- You will notice that the counter does hold when you press the bust-button. However, there is a problem:
- When the count reaches 7 and you press the hold push button, all of the LEDs go out and the count holds (actually at -1).
- Fix this problem so that the count always holds when the button is pushed, the counter always remembers the count, and there is one LED on at all times.

Demo \_\_\_\_\_



42

## Lecture 13 Exercise 2a

Demo \_\_\_\_\_

- Create an up-down ring counter that changes direction when you press a push-button. (A hold button is not required).
  - When the push-button is **not** depressed, the ring counter goes in the “normal” direction.
  - While the push-button is depressed, the ring counter goes in the opposite direction
- The counter should not skip or jump when you press the button (the count should be continuous).
- You are required to use a triggered subsystem to solve this problem.





43

## Lecture 13 Exercise 2b

Demo \_\_\_\_\_

- Create an up-down ring counter that changes direction when you press a push-button. (A hold button is not required).
- The push-button has memory (similar to a flip-flop):
  - When the push-button is pressed and released, the counter changes direction.
  - The counter does not change direction until the push-button is pressed and released again.
  - The counter changes direction every time the push-button is pressed and then released.
- The counter should not skip or jump when you press the button (the count should be continuous).
- You are required to use a triggered subsystem to solve this problem.



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Lecture 13 Exercise 3

44

- Create an up-down ring counter that changes direction when you press a push-button. (A hold button is not required).
- The counter should not skip or jump when you press the button. (The count should be continuous).
- A second push-button should be used to change the counting frequency. You should be able to change directions and speed simultaneously.

Demo \_\_\_\_\_



The MathWorks

freescale<sup>®</sup>  
semiconductor



45

## Lecture 13 Exercise 4

- Create two 4-bit ring counters with the following properties:
  - One counter shifts to the right, the other shifts to the left.
  - One counter counts at a 1 Hz rate, the other at a 10 Hz rate.
  - Your fixed step size is 1 ms.
  - Two push-buttons are available that hold the individual counters. While holding, the counters cannot lose their count. The hold functions on each counter are independent of the other counter.
- You are required to use triggered subsystem to solve this problem.

Demo \_\_\_\_\_



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Questions?



The MathWorks

freescale<sup>®</sup>  
semiconductor



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

# Introduction to Model-Based Systems Design

## Lecture 14: MPC555x Analog Input



## MPC555x Analog Input

2

- The MPC555x processor has over 40 analog inputs.
- There are two analog to digital converters on the processor, so only two channels can be sampled at the same time.
- Depending on the sample rate, the accuracy of the conversion is up to 12 bits.





3

## Analog Voltmeter - Bar Graph

- We will read a 0 to 5 V signal with an analog input.
- We will then scale the measured value to light up LEDs in a linear bar graph.
- Start with a new model and place the RAppID-EC block in it.
  - Set the compiler to Metrowerks
  - Set the target type to Internal flash.
- You should have an empty model except for the RAppID-EcoCAR block.

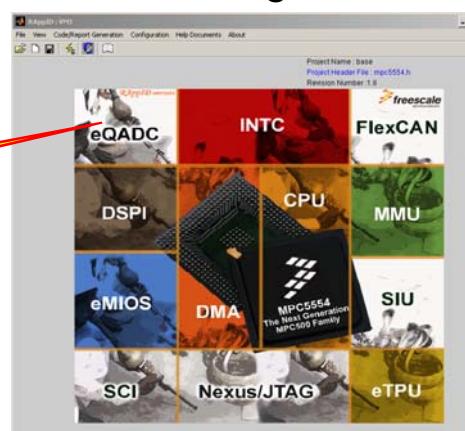


4

## Analog Input

- Double-click on the RAppID-EcoCAR block to obtain the configuration window.
- Click on the eQADC block to configure the ADC blocks.

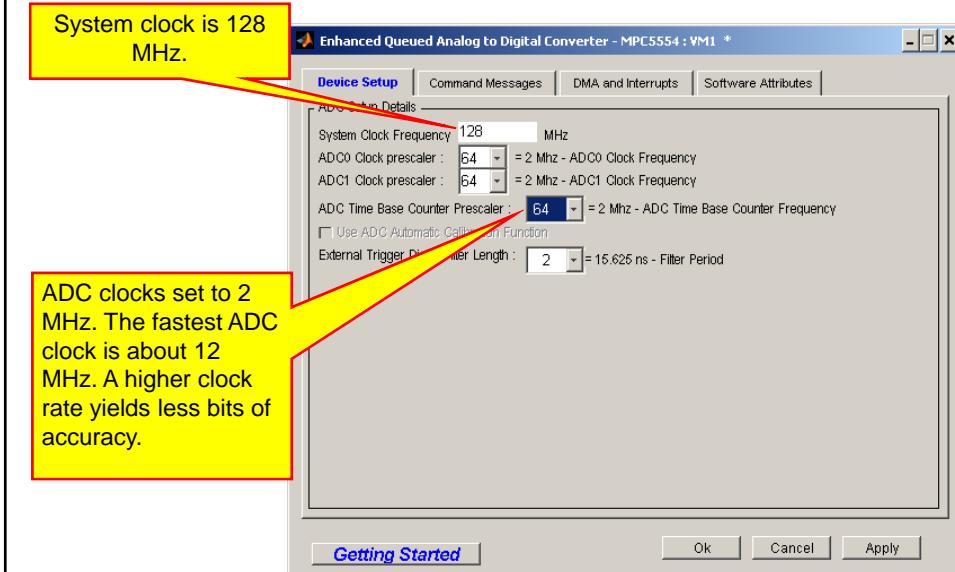
Click here.





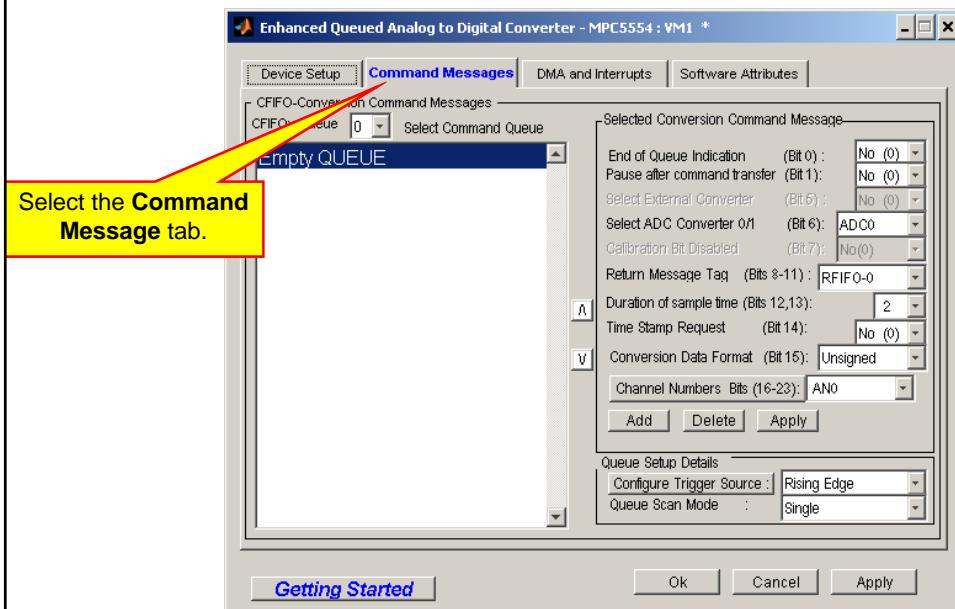
## ADC Prescaler

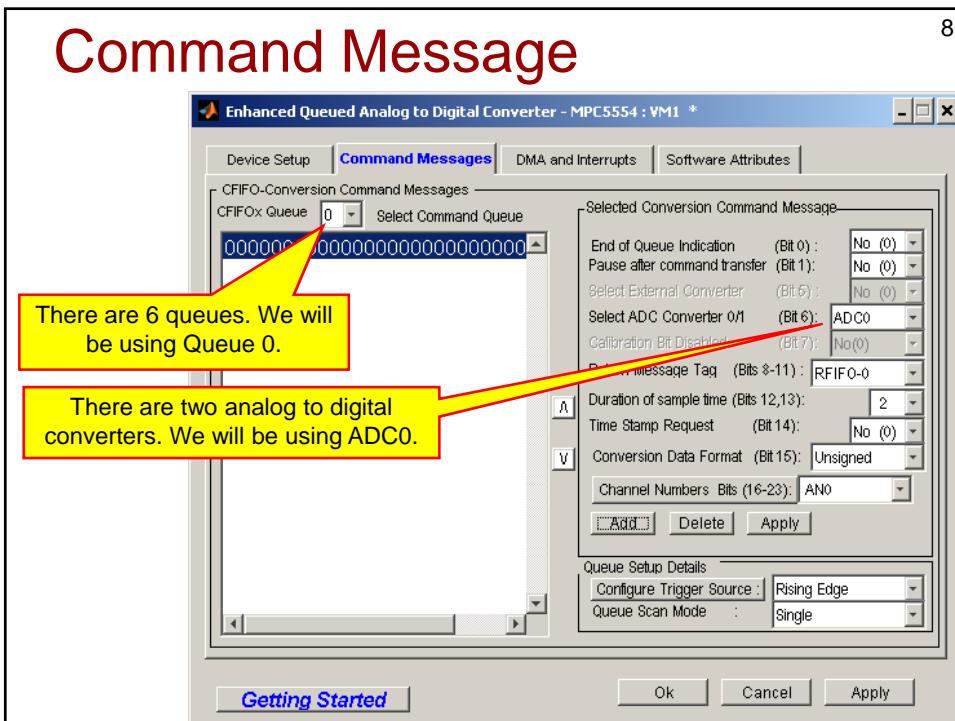
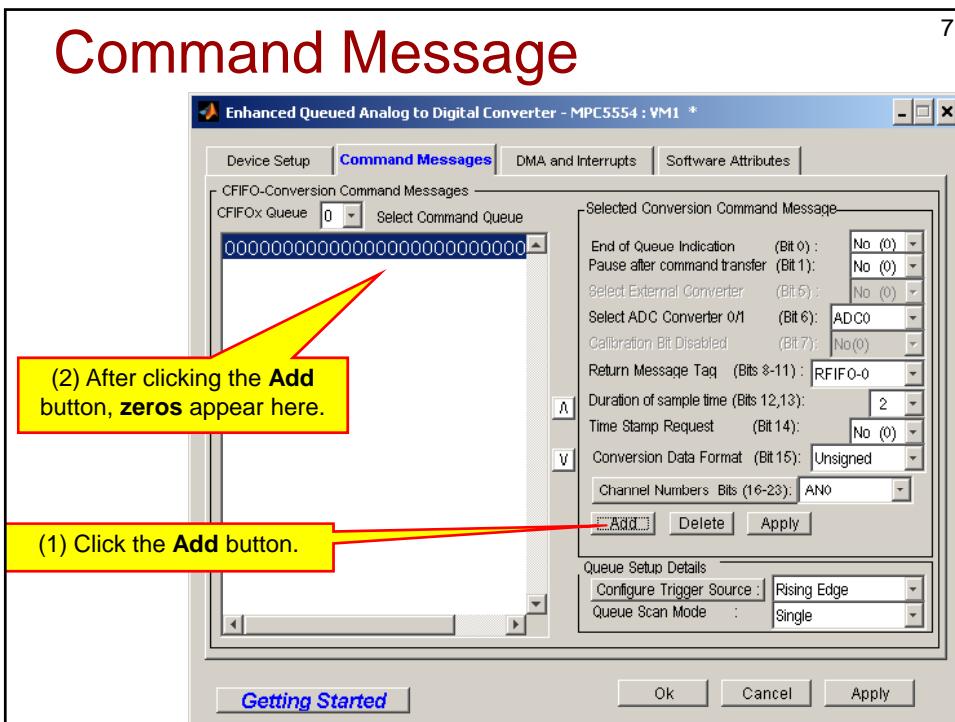
5

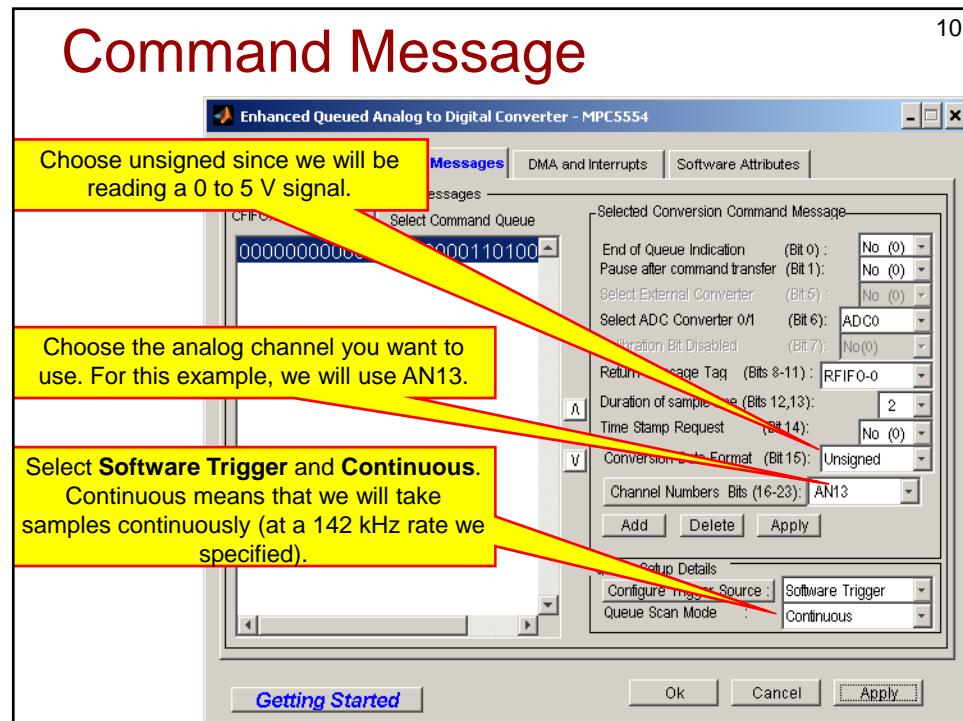
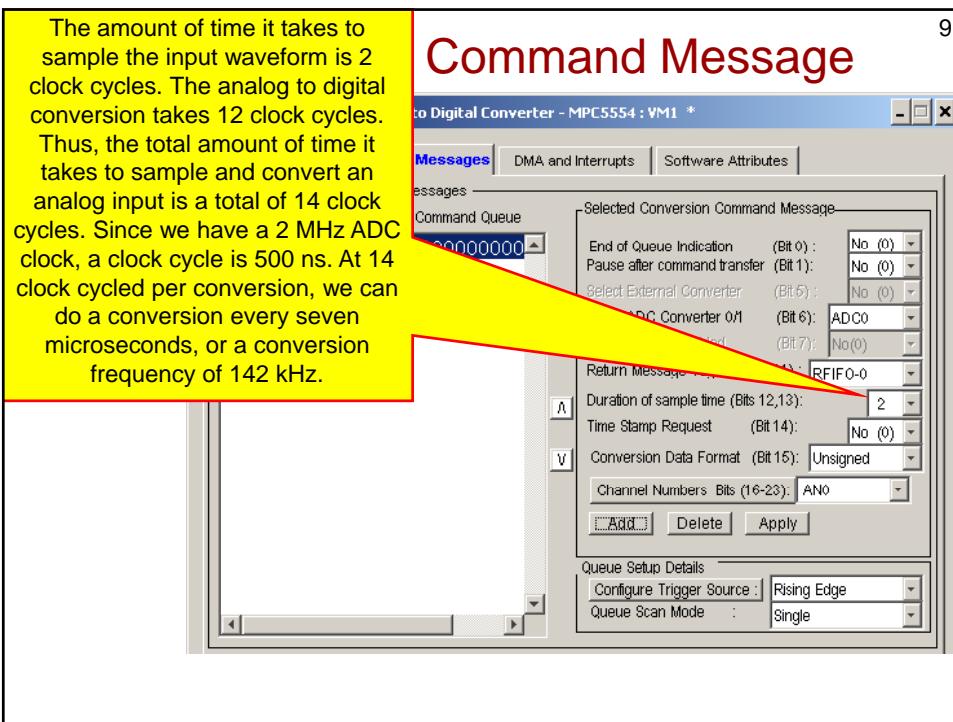


## Command Message

6



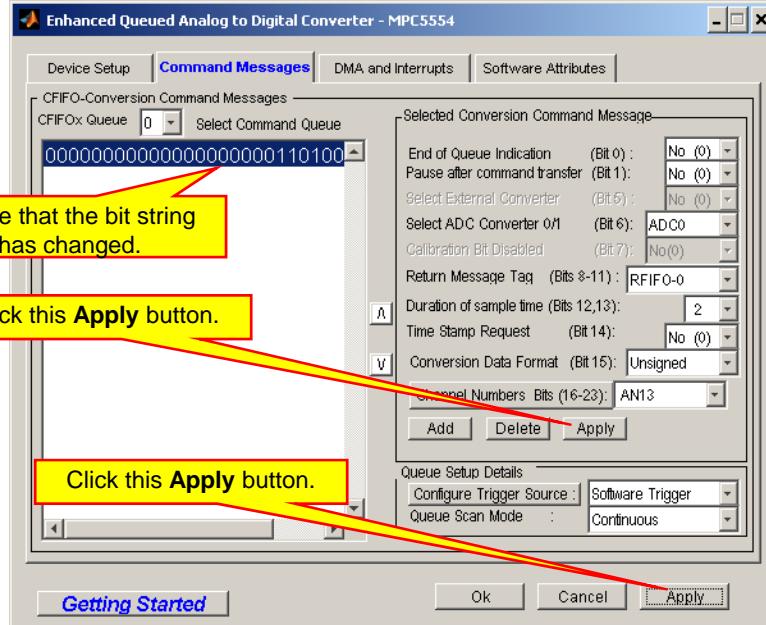






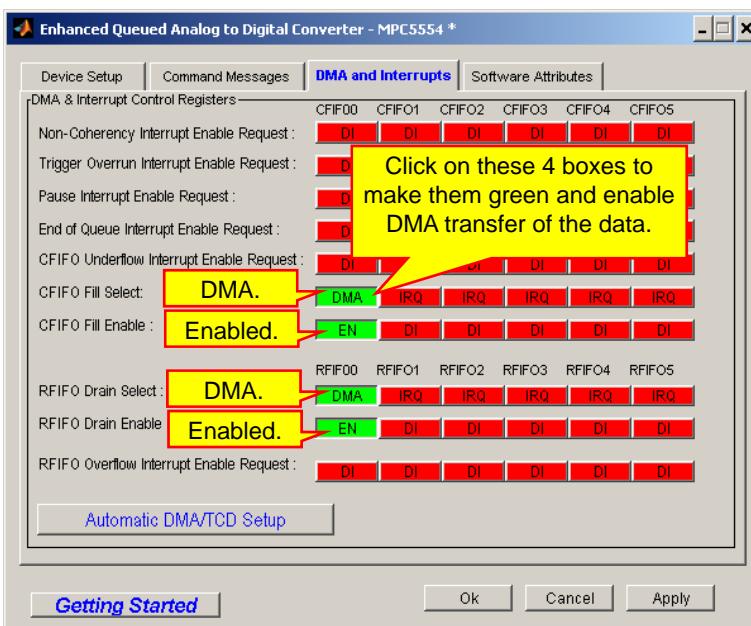
## Command Message

11



## DMA and Interrupts

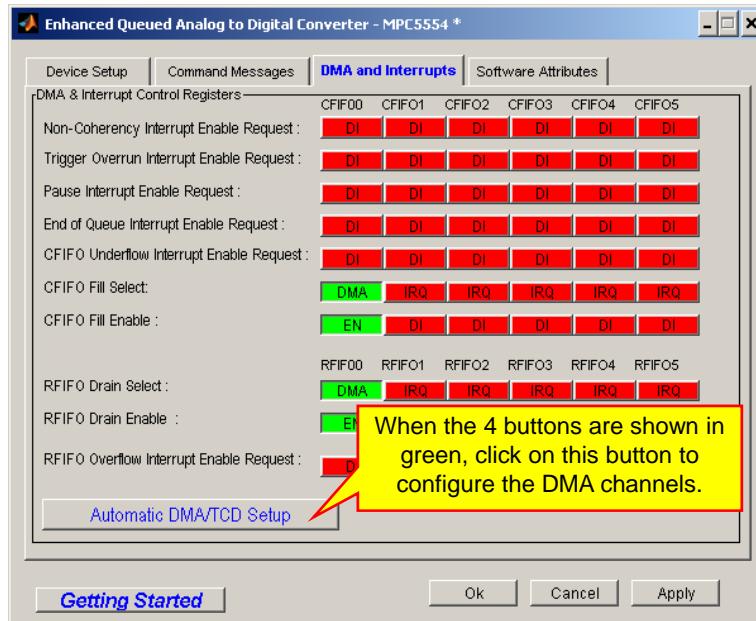
12





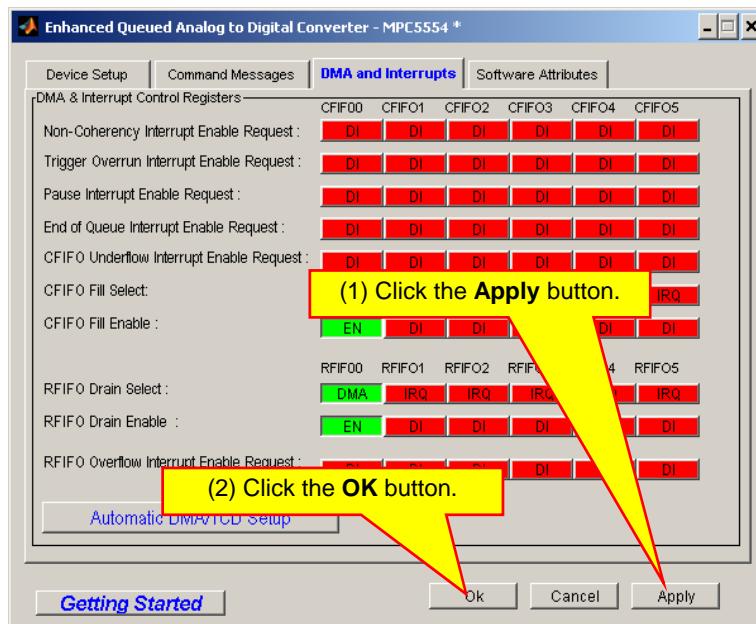
## DMA and Interrupts

13



## DMA and Interrupts

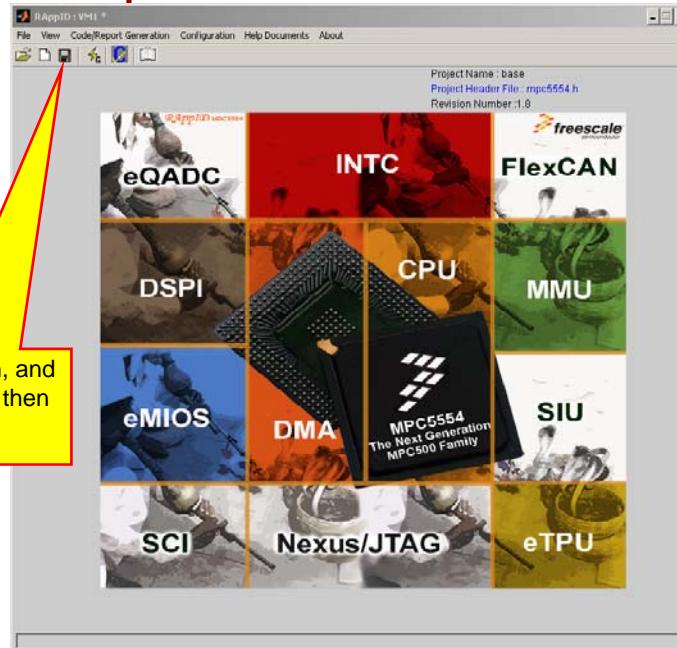
14





## RAppID Setup

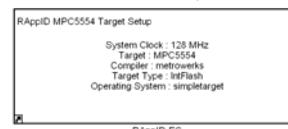
15



## eQADC Block

16

- Next, we can place an ADC block in our model.
- Place a block called **eQADC\_getdata** from library **RAppID-Toolbox / Peripheral Driver Blocks / eQADC Blocks** in your model:



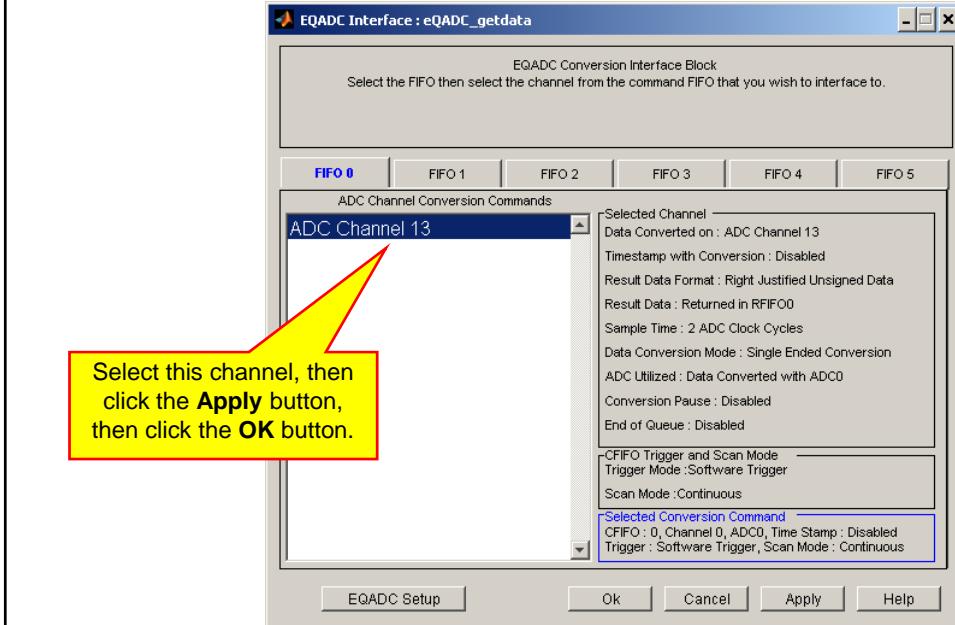
Double-click on this block.





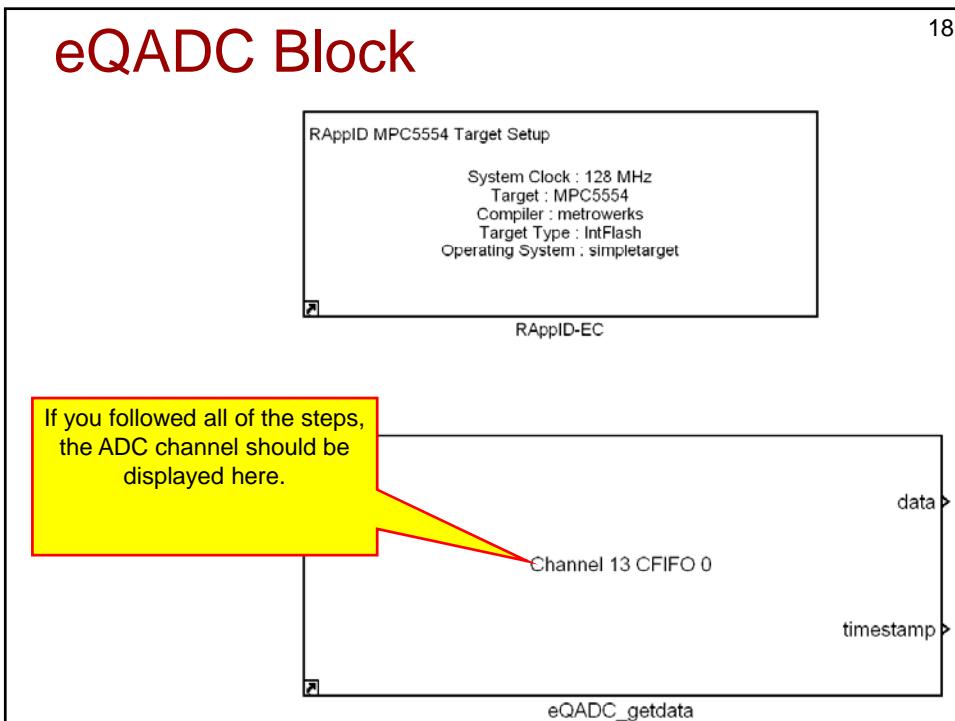
# eQADC Get Data

17



# eQADC Block

18





## eQADC

19

- We will not be using the **timestamp** output, so we will connect it to a terminator.
- The data output is:
  - Has 12 bits of resolution.
  - Returned as an unsigned 16-bit integer (UINT16)
  - Centered in the middle 12 bits of the 16 bit integer! → We need to divide by 4 to get at the 12 bit integer.



## Voltmeter Model

20

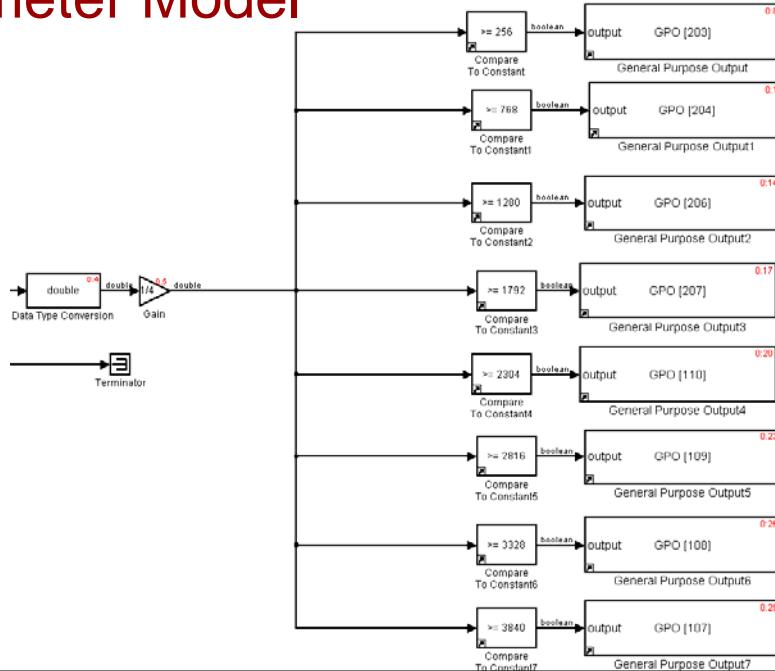
- The remaining logic of our model is shown in the next three slides:





## Voltmeter Model

21



## eQADC Initialization

22

- The last thing we need to do is initialize the ADC at system start-up.
- Place a block called **Simple Target StartupHook** in your model. (Library **RAppID / Utility Blocks / Simple TargetHookRoutuines**.)
- This block generates a function call trigger at system startup. We will use this to trigger a subsystem at startup.

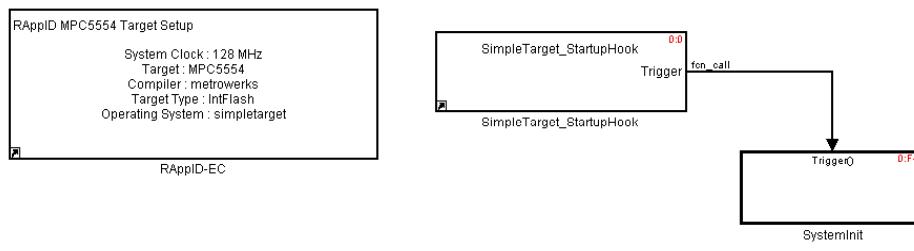




## System Init

23

- Place a triggered Subsystem in your model and change the trigger type to a function call trigger. Your top level model should look as shown:



## System Init

24

- In the triggered subsystem, place a block called **eqadc\_triggercmd** located in library **RAppID-Toolbox / Peripheral Driver Blocks / eQADC**.
- Change the FIFO Queue to Queue 0 (same as we specified in the eQADC setup slides.)
- Your subsystem should look as shown on the next slide:





## System Init

25



- This block initializes the FIFO Queue and starts the ADC converter.



## Analog Voltage Reference

26

- The last thing we need is a 0 to 5 V signal.
- The MPC555x Demo board has two potentiometers that will generate this signal.
- The voltages are available on the same jumper as we used for the push-button switches.

PIN #	USER COMPONENT CONNECTION
1	SW1 out, de-bounced CMOS drive 0 or 3.3V, active low.
2	SW1 out, de-bounced Open Drain output, active low, 10K ohm pull-up to 3.3V. Suitable for IRQ input signal drive.
3	SW2 out, active low, 10K ohm pull-up to 3.3V.
4	SW3 out, active low, 10K ohm pull-up to 3.3V.
5	SW4 out, active low, 10K ohm pull-up to 3.3V.
6	SPEAKER amp input. 0 to 5V/pp, volume adjust with SPKR_VOL.
7	RV1 center tap, 0 – 5V adjustment
8	RV2 center tap, 0 – 5V adjustment

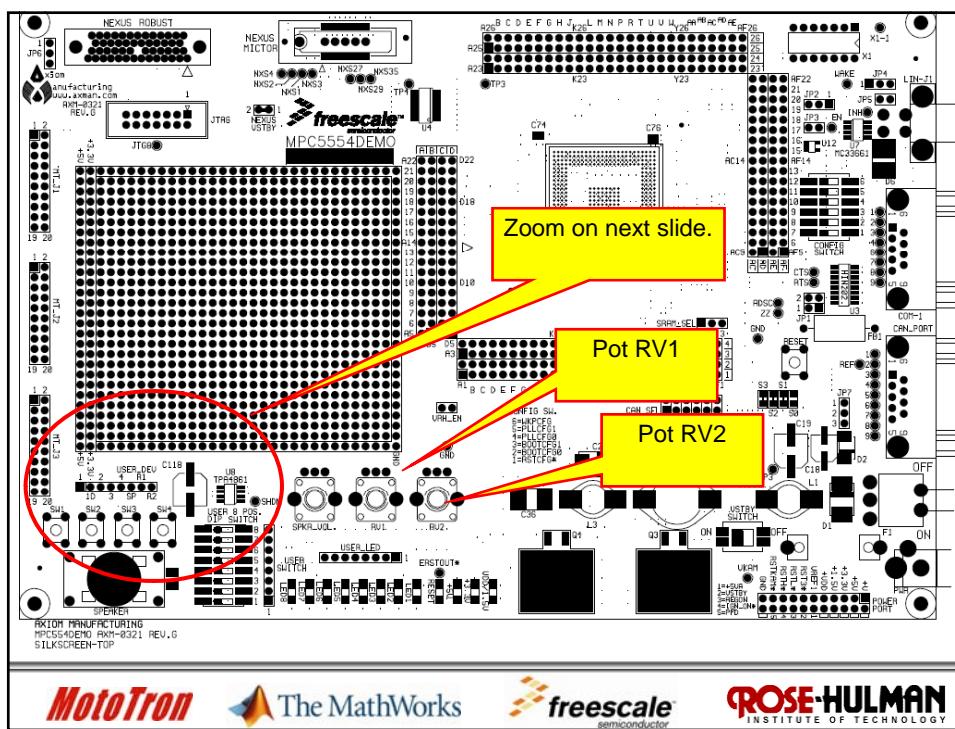


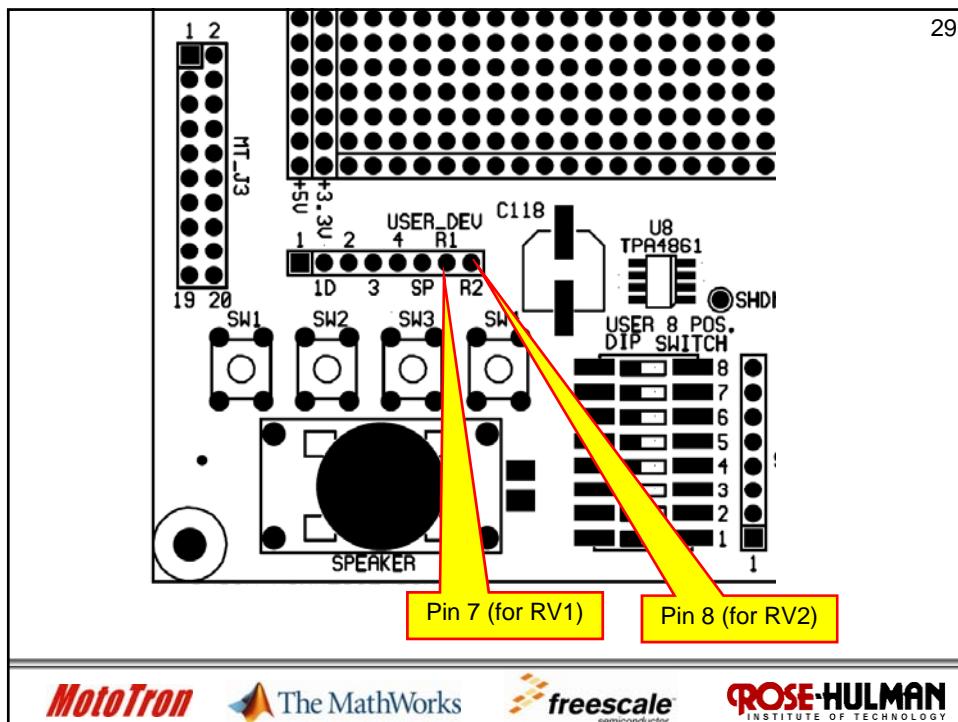
27

## Analog Voltage Reference

- Pins 7 and 8 are connected to two separate potentiometers.
- We can use either one as our input.
- The location of the pins and the potentiometers are shown on the next slide.

PIN #	USER COMPONENT CONNECTION
1	SW1 out, de-bounced CMOS drive 0 or 3.3V, active low.
2	SW1 out, de-bounced Open Drain output, active low, 10K ohm pull-up to 3.3V. Suitable for IRQ input signal drive.
3	SW2 out, active low, 10K ohm pull-up to 3.3V.
4	SW3 out, active low, 10K ohm pull-up to 3.3V.
5	SW4 out, active low, 10K ohm pull-up to 3.3V.
6	SPEAKER amp input. 0 to 5V/pp, volume adjust with SPKR_VOL.
7	RV1 center tap, 0 – 5V adjustment
8	RV2 center tap, 0 – 5V adjustment





## Lecture 14 – Demo 1

30

- Analog Input – Demo
- Wire up your circuit.
- Compile and download the model.
- Display all signal values using FreeMASTER:
- Demo your working analog voltmeter / bar graph.

algorithm block description			
Name	Value	Unit	
Raw_Value	b#10110100	BIN	100
Double_Value	184	unit	100
Scaled_Value	46	unit	100
Bit_0	0	DEC	100
Bit_1	0	DEC	100
Bit_2	0	DEC	100
Bit_3	0	DEC	100
Bit_4	0	DEC	100
Bit_5	0	DEC	100
Bit_6	0	DEC	100
Bit_7	0	DEC	100

Demo \_\_\_\_\_

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



31

## Embedded MATLAB Functions

- Before we look at our next example with analog inputs, we will look at the Embedded MATLAB Function.
- This function allows us to use MATLAB functions within a Simulink model.
- The Embedded MATLAB Function block supports a subset of the language for which it can generate efficient embeddable code.
- For details about the Embedded MATLAB subset, see *Working with Embedded MATLAB* in the Embedded MATLAB documentation.



The MathWorks

freescale<sup>®</sup>  
semiconductor

32

## Embedded MATLAB Functions

- To generate embeddable code, the Embedded MATLAB Function block relies on an analysis that determines the size and class of each variable.
- This analysis imposes the following additional restrictions on the way in which the above features may be used.
  - The first definition of a variable must define both its class and size. The class and size of a variable cannot be changed once it has been set.
  - Whether data is complex or real is determined by the first definition.



The MathWorks

freescale<sup>®</sup>  
semiconductor



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

33

## Embedded MATLAB Functions

- The preceding limitations require you to code in a certain style.
- Some common idioms to avoid are listed in *Limitations on Indexing Operations* and *Limitations with Complex Numbers* in the Embedded MATLAB documentation.



34

## Embedded MATLAB Functions

- In addition to language restrictions, Embedded MATLAB Function blocks support only a subset of the functions available in MATLAB.
- A list of supported functions is given in the *Embedded MATLAB Run-Time Function Library* in the Embedded MATLAB documentation.
- These functions include common categories
  - Arithmetic functions like plus, minus, and power
  - Matrix operations like size, and length
  - Advanced matrix operations like lu, inv, svd, and chol
  - Trigonometric functions like sin, cos, sinh, and cosh





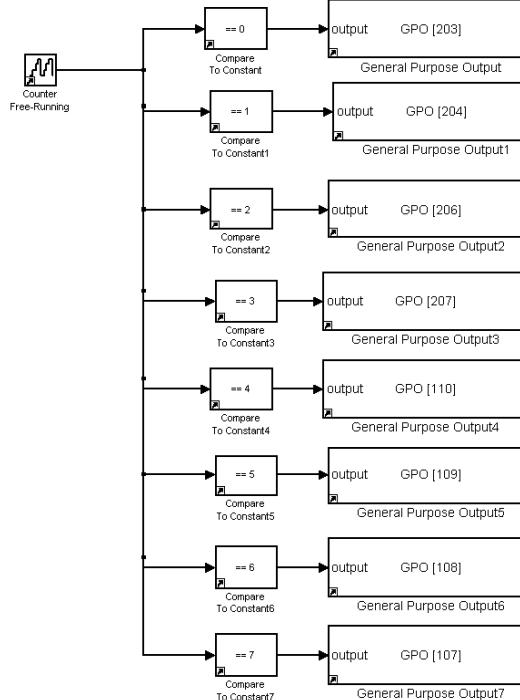
35

## Ring Counter

- As an example of using an Embedded MATLAB Function, we will modify the ring counter model we developed earlier.
- Instead of using the compare to constant blocks, we will use a single Embedded MATLAB Function block.
- The ring counter model we will start with is shown next.



36

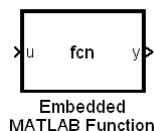




## Ring Counter

37

- Remove the Compare to constant blocks and place a single **Embedded MATLAB Function** block in your model.
- The block is located in library **Simulink / User-Defined Functions**.

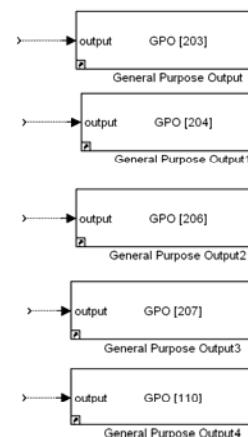


- Note that the default block has a single input and single output.

RAppID MPC5554 Target Setup  
 System Clock : 128 MHz  
 Target : MPC5554  
 Compiler : metrowerks  
 Target Type : InflFlash  
 Operating System : simpletarget

RAppID-EC

Double-click on this  
block to open it.

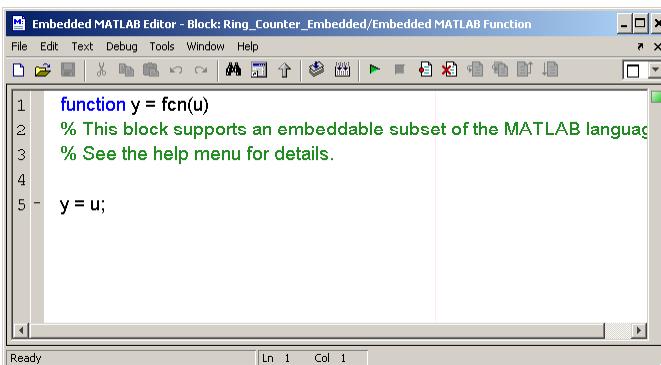




39

## Embedded MATLAB Function

- When you open the Embedded MATLAB Function block, a text editor opens and displays the function.
- Note that the block inputs and outputs are derived from the function declaration in the first line of the function.



```

1 function y = fcn(u)
2 % This block supports an embeddable subset of the MATLAB language
3 % See the help menu for details.
4
5 - y = u;

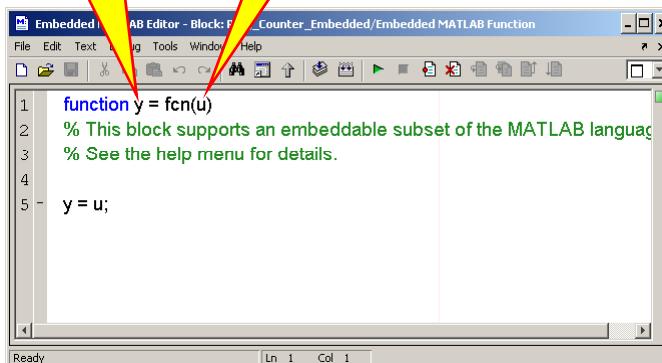
```

40

## Embedded MATLAB Function

The block will have a single output labeled y.

The block will have a single input labeled u.



```

1 function y = fcn(u)
2 % This block supports an embeddable subset of the MATLAB language
3 % See the help menu for details.
4
5 - y = u;

```



41

## Embedded MATLAB Function

- We have a single input that is the count.
- We need eight outputs, one for each LED.
- An example implementation is shown next:



The MathWorks



42

```

1 function [y0, y1, y2, y3, y4, y5, y6, y7] = LED_Out(count)
2
3     y0=False; y1=False; y2=False; y3=False; y4=False; y5=False; y6=False; y7=False;
4
5     if count == 0
6         y0=True;
7     else
8         if count == 1
9             y1=True;
10        else
11            if count == 2
12                y2=True;
13            else
14                if count == 3
15                    y3=True;
16                else
17                    if count == 4
18                        y4=True;
19                    else
20                        if count == 5
21                            y5=True;
22                        else
23                            if count == 6
24                                y6=True;
25                            else
26                                if count == 7
27                                    y7=True;
28                                end
29                            end
30                        end
31                    end
32                end
33            end
34        end
35    end

```

Eight outputs have been defined, labeled y0 through y7.

Function block named LED\_Out.

Input named count.





43

```

1 function [y0, y1, y2, y3, y4, y5, y6, y7] = LED_Out(count)
2
3 y0=False; y1=False; y2=False; y3=False; y4=False; y5=False; y6=False; y7=False;
4
5 if count == 0
6     y0=True;
7 else
8     if count == 1
9         y1=True;
10    else
11        if count == 2
12            y2=True;
13        else
14            if count == 3
15                y3=True;
16            else
17                if count == 4
18                    y4=True;
19                else
20                    if count == 5
21                        y5=True;
22                    else
23                        if count == 6
24                            y6=True;
25                        else
26                            if count == 7
27                                y7=True;
28                            end
29                        end
30                    end
31                end
32            end
33        end
34    end
35 end

```

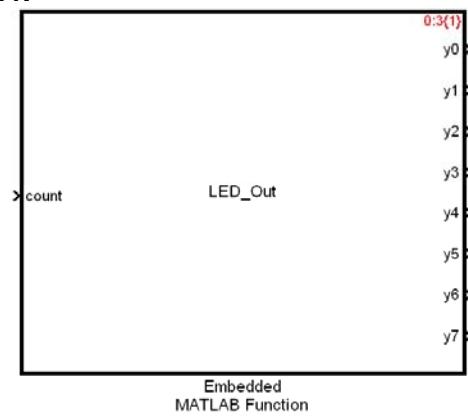
Outputs predefined as false and declared as Boolean data types. We will not need a convert block.

A pretty lame implementation.

## Embedded MATLAB Function

44

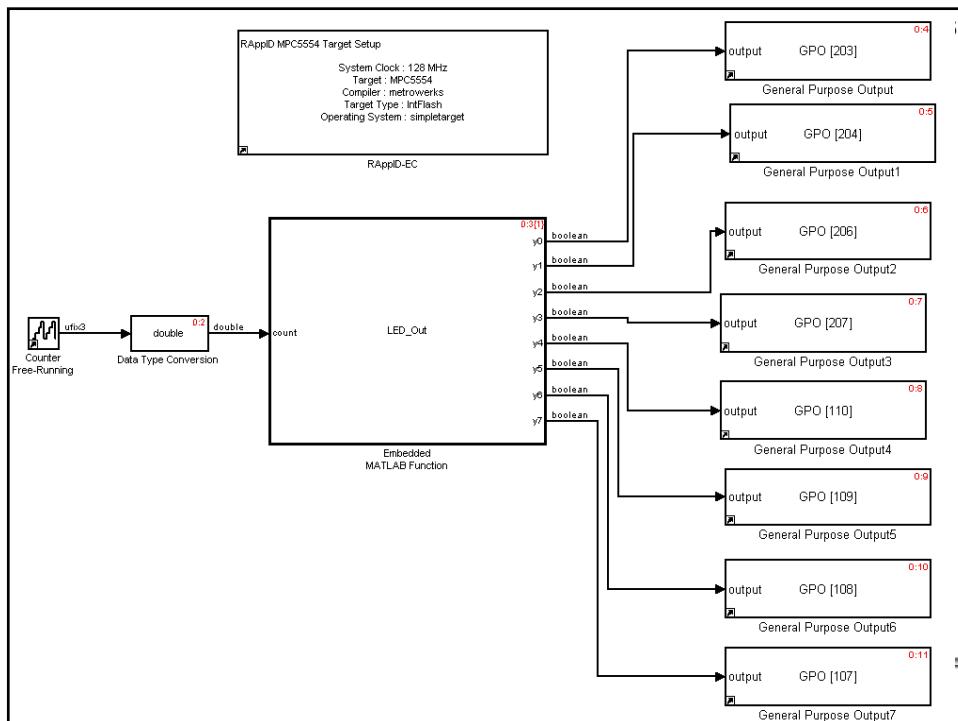
- When we save the function and look at the Simulink model, we see that the block name, input, and outputs reflect the function declaration:



## Embedded MATLAB Function

45

- Connect the function as shown.
  - We will need a convert block for the input to convert the data type of the free running counter to double.
  - When you type ctrl-d, you will notice that the output signals from the LED\_Out block are Boolean data types.





47

## Embedded MATLAB Function

- A different realization of the embedded function is shown below:

```

1 function [y0, y1, y2, y3, y4, y5, y6, y7] = LED_Out(count)
2
3 - y0=False; y1=False; y2=False; y3=False; y4=False; y5=False; y6=False; y7=False;
4
5 - switch count
6 - case 0
7 -     y0=True;
8 - case 1
9 -     y1=True;
10 - case 2
11 -     y2=True;
12 - case 3
13 -     y3=True;
14 - case 4
15 -     y4=True;
16 - case 5
17 -     y5=True;
18 - case 6
19 -     y6=True;
20 - case 7
21 -     y7=True;
22 end

```

48

## Embedded MATLAB Functions

- The last implementation we will show uses a row vector for the output.

```

function y = LED_Out(count)

y=[False, False, False, False, False, False, False, False];

y(count+1) = True;

```

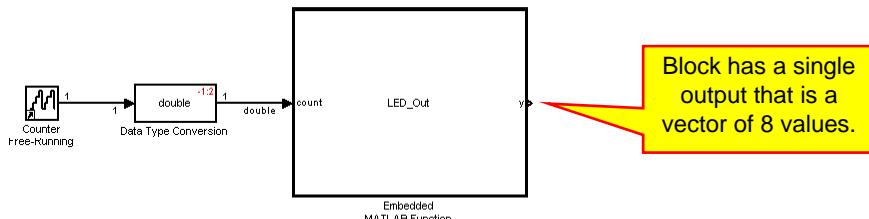
- The function has a single output, but it contains 8 values.
- When you look at the block, it will have a single output:





49

## Embedded MATLAB Functions



- We need to run the Simulink Model Explorer to specify the output as a row vector.
- Select **View** and then **Model Explorer** from the Simulink menus.
- Select **Embedded MATLAB Function** from the tree:

**MotoTron**

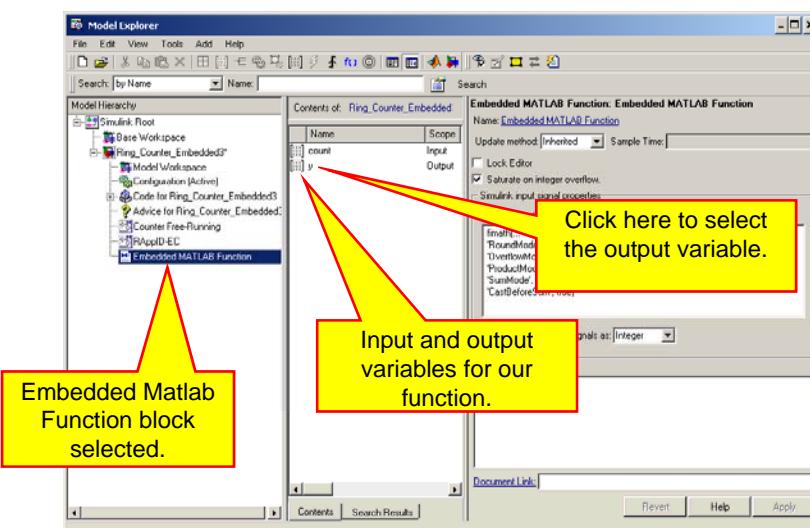
The MathWorks

freescale<sup>®</sup>  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

50

## Embedded MATLAB Functions



**MotoTron**

The MathWorks

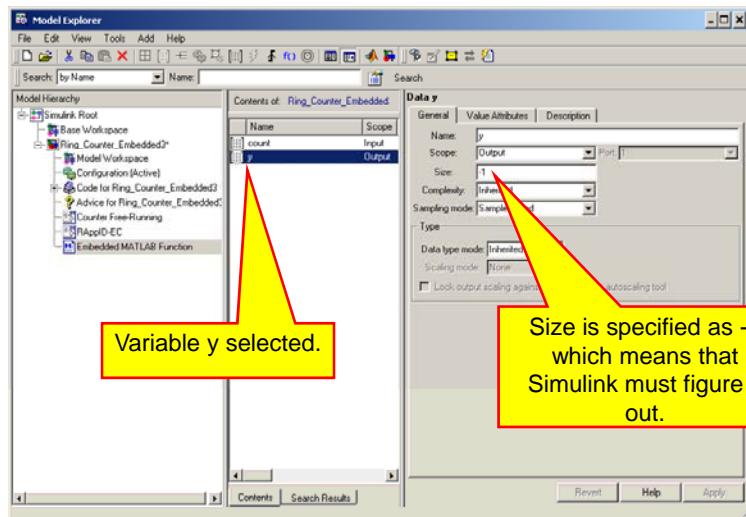
freescale<sup>®</sup>  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



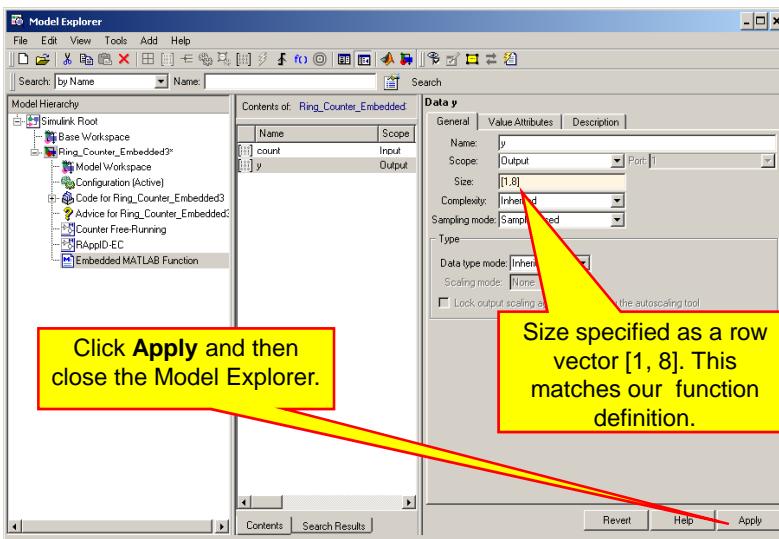
51

## Embedded MATLAB Functions

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

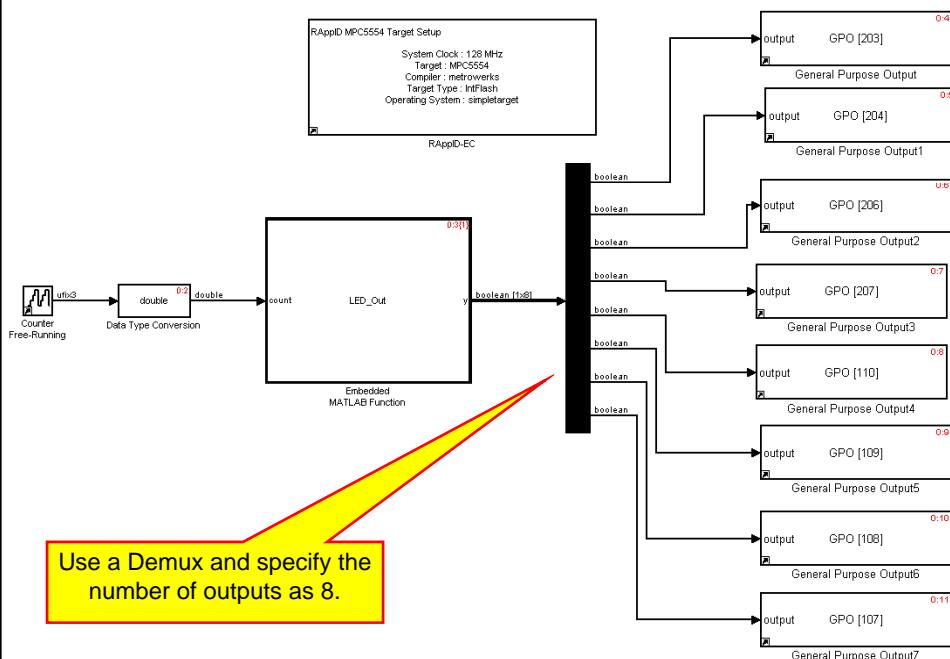
52

## Embedded MATLAB Functions

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



53



## Lecture 14 – Demo 2

54

- Wire up your circuit.
- Compile and download the model.
- Demo your working system.
  - Ring counter with embedded Matlab function.

Demo \_\_\_\_\_





55

## Lecture 14 Exercise 1

Demo \_\_\_\_\_

- Create an up-down ring counter where the up/down function and the ring counting logic is contained in an Embedded MATLAB Function block.
- As push-button is used to change directions of the counter.
- The counter is allowed to skip positions when you press the push-button.
- You may use a counter external to the embedded Matlab function if you so desire.



## Lecture 14 Exercise 2

Demo \_\_\_\_\_

56

- Create an up-down ring counter where the up/down function, the ring counting logic, and the stored count is contained in an Embedded MATLAB Function block.
- As push-button is used to change directions of the counter.
- The counter is **not** allowed to skip positions when you press the push-button.
- The embedded function must keep track of the count. You may not use an external counter to keep track of the count. (In essence, the embedded Matlab function has memory.)
  - You may want to use the Matlab persistent and isempty functions.





57

## Lecture 14 Exercise 3

- Create a ring counter using a **Truth Table**.
- **Truth Tables** are located in the **Stateflow** library.

Demo \_\_\_\_\_



58

## Serial Communication

- We will now show an example of using the Enhanced Serial Communication Interface (eSCI).
- This will show examples of:
  - Embedded MATLAB function
  - For Iterator
  - Index Vector
- We will start with the voltmeter me made earlier.





59

## Serial Communication

- We will convert a 0 to 5 V analog signal to a numerical value of 0 to 500.
- We will convert the number to an array of ASCII codes.
- We will output the codes to the serial COM port and view it with a terminal program such as HyperTerminal.
- We will use a triggered subsystem to output the text string once a second.
- The top level model is shown next:

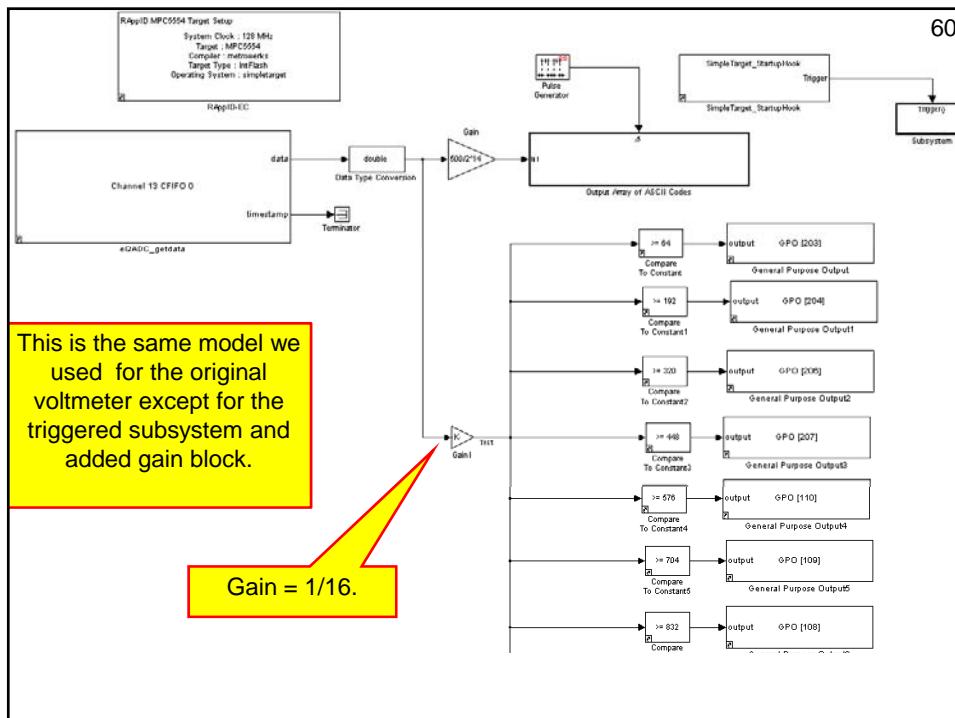
**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

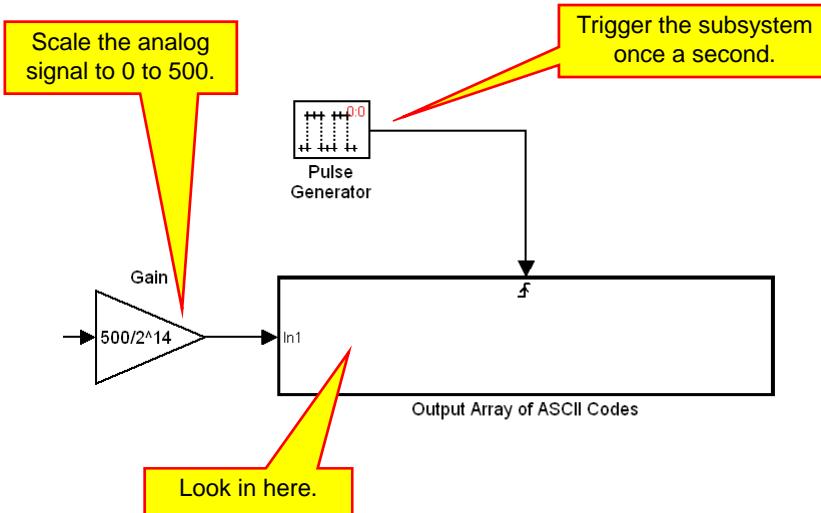
60





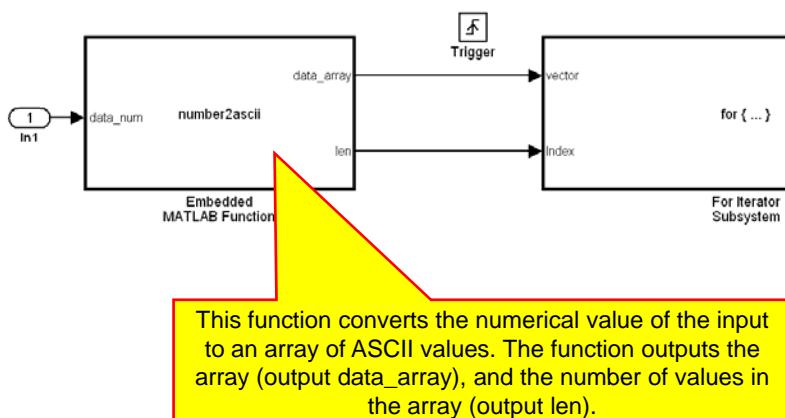
61

## Serial Communication



62

## Output ASCII Text String





```
function [data_array, len] = number2ascii(data_num) 63
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

index = 0;
cnt = 1;
data_array = [32,32,32,32,32,32,32,32,32];
temp = [32,32,32,32,32,32,32,32,32];
len = 0;
if(data_num ==0)
    index = index +1;
    data_array(index) = 48;
else
    if(data_num < 0) %negative sign
        index = index +1;
        data_array(index) =45;
    data_num = (data_num * (-1));
end
```

Declare the arrays. 32 is the ASCII code for a space.

If the numerical value is 0, set the first element in the array to 48, the ASCII code for 0.

If the numerical value is negative, set the first element in the array to 45, the ASCII code for '-'.

If the numerical value is negative, convert it to a positive number for the remainder of the conversion.

```
while(data_num ~= 0 )
    digit =mod(fix(data_num),10); %convert
    temp(cnt)=digit+48; %length
    data_num = fix(data_num/10);
    cnt = cnt+1; %length
end
```

Get the least significant digit in the number (ignore the fractional part of the number.).

The ASCII codes for 0 through 9 are 48 through 57.

Shift the number to the right by one digit.

```
% Arrange the numbers in correct order
for i=cnt-1:-1:1
    index = index +1;
    data_array(index)=temp(i); %reverse
end
```

Reverse the order of the array because we started with the least significant digit.

```
data_array(index+1) = 13;
data_array(index+2) = 10;
len =index+2;
```

Tack on a line feed and carriage return.

```
end
```



## Embedded Functions

65

- To see what the function does, we can copy and paste the function into an m-file and then test it at the command prompt.
- Some examples are shown next:



```
>> [data_array, len] = number2ascii(123.4)
```

```
data_array =
```

```
49 50 51 13 10 32 32 32 32
```

ASCII 32 is a space.

```
len =
```

```
5
```

```
>> [data_array, len] = number2ascii(123)
```

```
data_array =
```

```
49 50 51 13 10 32 32 32 32
```

ASCII 13 and 10 are a carriage return and a line feed.

```
len =
```

```
5
```

```
>> [data_array, len] = number2ascii(-123)
```

```
data_array =
```

```
45 49 50 51 13 10 32 32 32
```

ASCII 3

ASCII -

ASCII 1

ASCII 2

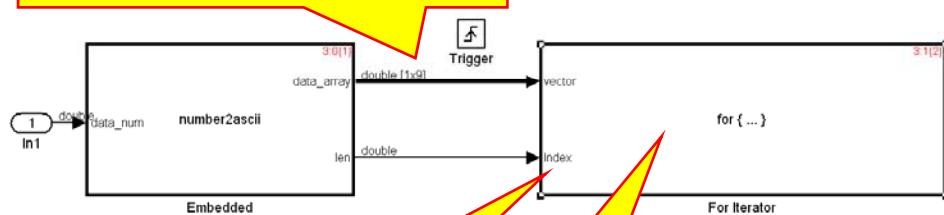




67

## Output ASCII Text String

Note that this output is an array.



Next, look inside this subsystem.

This is a **For Iterator Subsystem** block. It is located in the **Simulink / Ports & Subsystems** library.

**MotoTron**

**The MathWorks**

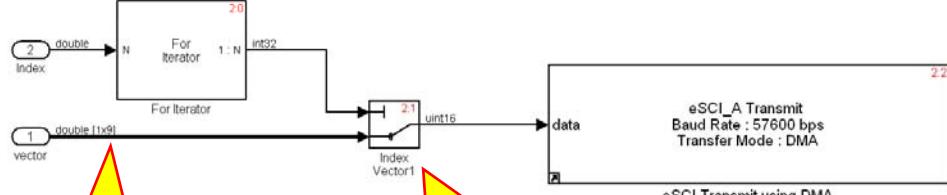
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

68

## Output ASCII Text String

Iterate from 1 to the length of the ASCII string. (The value of output 'len' in the previous slide.). Make sure you check the indexing method. **Make sure that it is ones based.**



This signal line is an array of values.

Using the index provided by the For Iterator, step through the values in the array. **Make sure that you check the indexing method. Make sure that it is ones based.**

**MotoTron**

**The MathWorks**

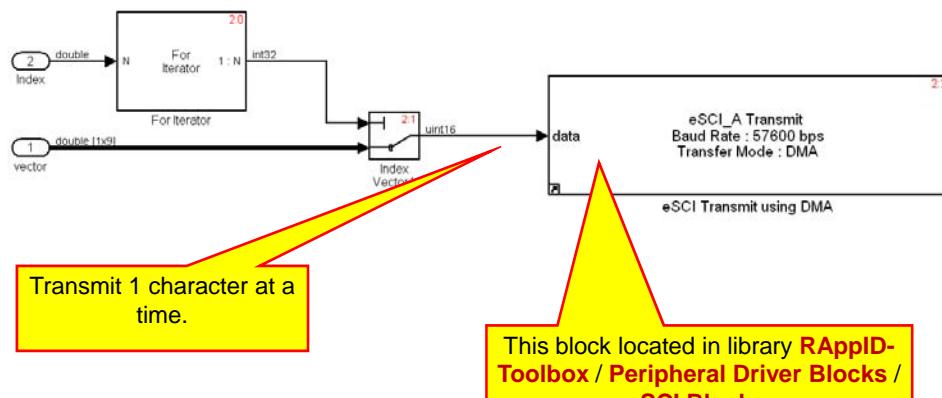
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Output ASCII Text String

69



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Serial Output

70

- Using a 9-pin serial cable, connect your computer to COM 1 of the MPC555x board.
- Use hyper terminal to view the results.
- You will need to set the baud rate appropriately. (57600, 8-N-1)

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Lecture 14 Demo 3

71

- Wire up your circuit.
- Compile and download the model.
- Demo your working system.
  - Analog voltmeter with ASCII output.

Demo\_\_\_\_\_



## Function Call Triggered Subsystems

72

- For the next example, we will show how to force a sequence of events to occur in a specific order.
- We will do this with triggered subsystems that use a function call for the trigger.
- The previous example just output a numerical value between 0 and 500.
- We would like to output the text, ‘Analog Input 1: ’.





73

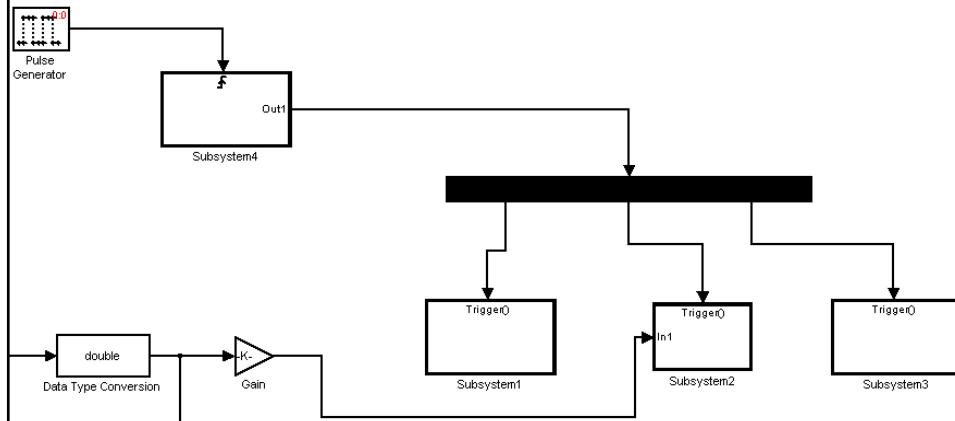
## Function Call Triggered Subsystems

- Next, we would like to output the text equivalent for the number.
- Finally, we would like to output a period, carriage return, and a line feed.
- We will break up the three sections into three triggered subsystems.
- We will show how to force the subsystems to execute in a specific order.

74

## Function Call Triggered Subsystems

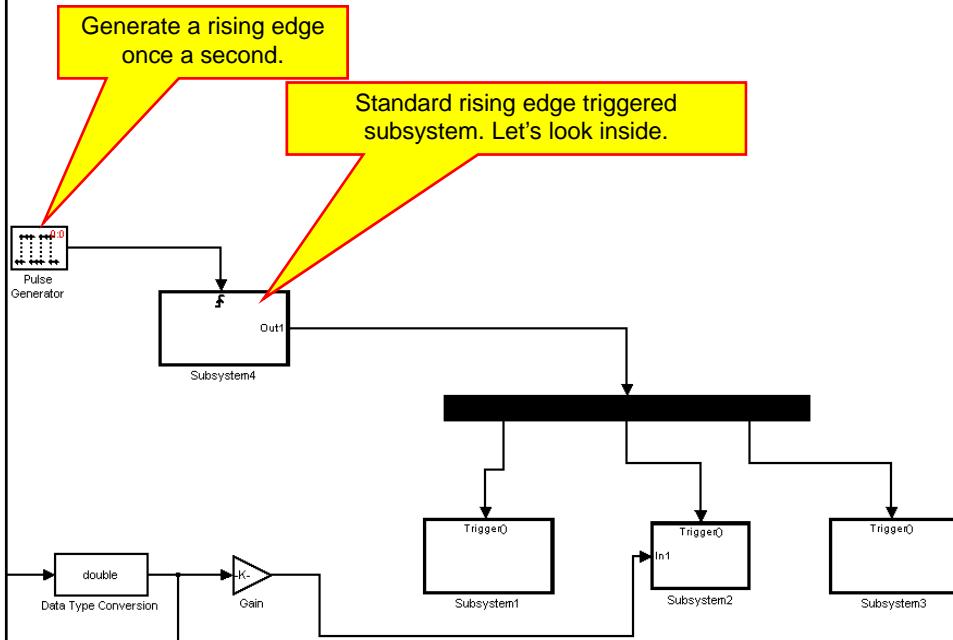
- We will use the previous Voltmeter model and make a few modifications.
- The changes are shown below:





75

## Function Call Triggered Subsystems



76

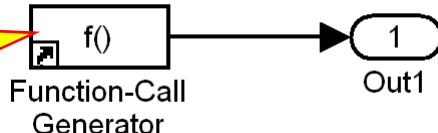
## Function Call Triggered Subsystems

All this subsystem does is generate a function call trigger. Basically, we are converting an edge trigger to a function call trigger.



Trigger

This block is located in library **Simulink / Ports & Subsystems**. Double-click on this block to view its parameters.



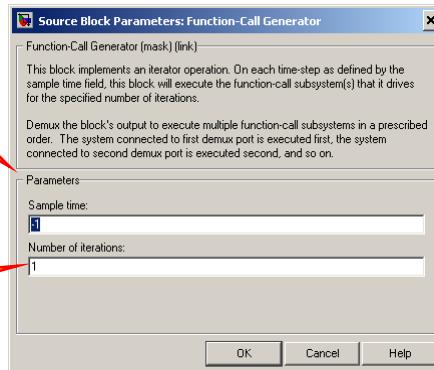


## Function-Call Generator

77

Sample time set to -1 (inherited) because we are inside a triggered subsystem.

Generate 1 function call each time this subsystem is triggered.



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Function Call Triggered Subsystems

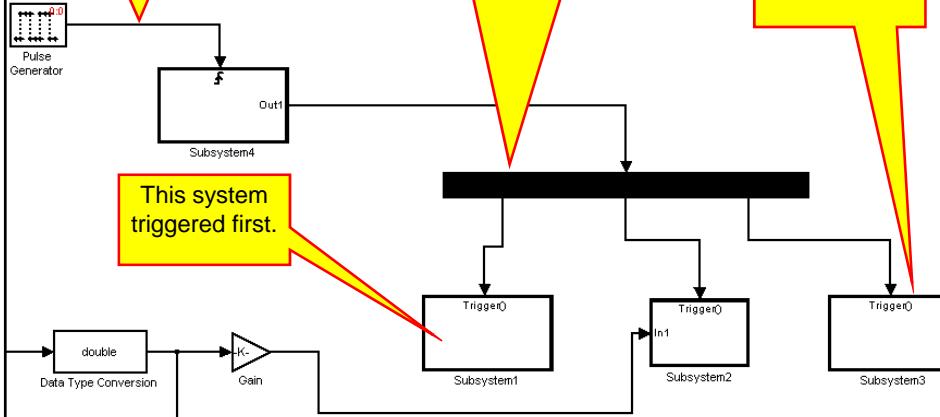
78

One edge here triggers all three subsystems once, in order from left to right.)

This is a **DEMUX** with 3 outputs.

This system triggered last.

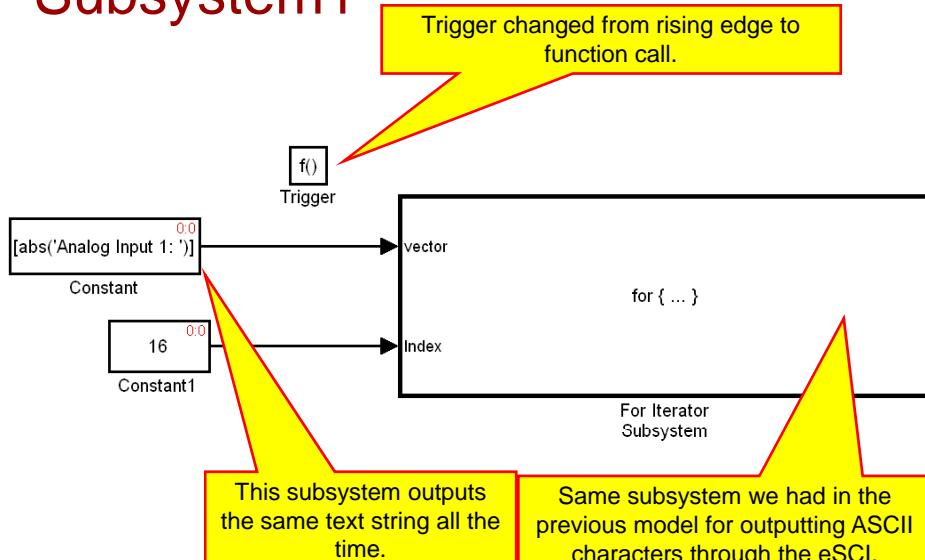
This system triggered first.





## Subsystem1

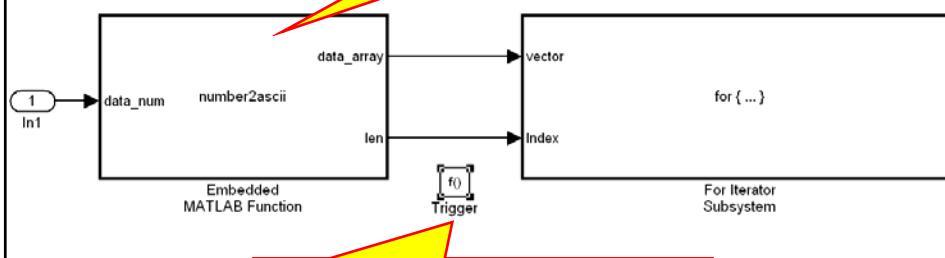
79



## Subsystem2

80

The embedded function was modified slightly.  
 The carriage return and line feed are no longer added at the end of the number. The modified function is shown next.





81

## Embedded MATLAB Function

```

while(data_num ~= 0)
    digit =mod(data_num,10);      %get the reminder
    temp(cnt)=digit+48;          %convert to ascii character
    data_num = fix(data_num/10);   %remove current ones digit
    cnt = cnt+1;                 %length of number
end

% Arrange the numbers in correct order
for i=cnt-1:-1:1
    index = index +1;
    data_array(index)=temp(i); %reorder string correctly
end

%data_array(index+1) = 13;
%data_array(index+2) = 10;
%len =index+2;
len=index;

```

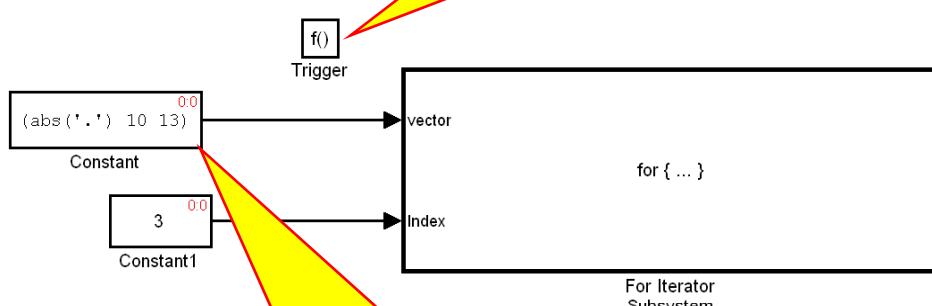
These line modified. The line feed and carriage return are not added to the array.



## Subsystem 3

82

Trigger changed from rising edge to function call.



This subsystem outputs the same text string all the time. (A period, carriage return, and a line feed.





83

## Lecture 14 Demo 4

- Compile and download the model.
- Demo your working system.
- Example output from my model is shown below:

```
Analog Input 1: 5.  

Analog Input 1: 5.  

Analog Input 1: 113.  

Analog Input 1: 302.  

Analog Input 1: 65.  

Analog Input 1: 228.  

Analog Input 1: 234.  

Analog Input 1: 202.
```

Demo\_\_\_\_\_



## Lecture 14 Exercise 4

84

- Read two analog input voltages. Use potentiometers RV1 and RV2 as your inputs.
- Use RV1 to control the output of your LED voltmeter.
- Output the following text strings every second, and in the same order (must be forced in this order using function call triggered subsystems):

Analog Input 1: xxx.

Analog Input 2: yyy.

Demo\_\_\_\_\_





85

## Lecture 14 Exercise 5

- Use the analog input and a potentiometer to control the speed at which your ring counter shifts.
- The counter frequency should be continuously variable from 1 Hz to 10 Hz.

Demo \_\_\_\_\_



The MathWorks



freescale  
semiconductor



## Questions?



The MathWorks



freescale  
semiconductor





# Introduction to Model-Based Systems Design

## Lecture 14A: RHIT Debug Blocks



## MPC555x Analog Input

2

- In lecture 14 we learned how to write ASCII text to the eSCI port and display that text with a terminal program such as MS Terminal.
- The display was somewhat limited as we could only display line after scrolling line:

```
Analog Input 1: 5.  
Analog Input 1: 5.  
Analog Input 1: 113.  
Analog Input 1: 302.  
Analog Input 1: 65.  
Analog Input 1: 228.  
Analog Input 1: 234.  
Analog Input 1: 202.
```

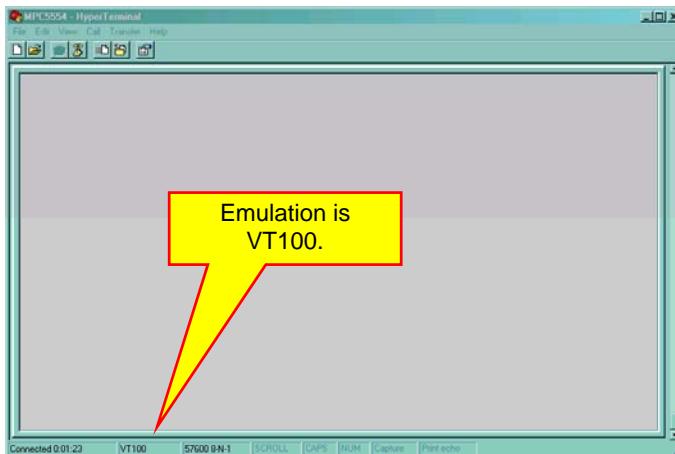




## Terminal Emulation

3

- When you ran MS terminal, you might have seen something that reminded you of days gone past (if you are old like me):



## Video Terminals

4

- Back in the days of mainframes and video terminals that printed on paper, VT100 video terminals were the terminals that dreams were made of.
- Most people used terminals that printed on paper using a ribbon and a 9-wire head. (If you could afford it, you would have a 24 wire printer.)
- Techies with toys, professors with huge research grants, and geeks that would buy hardware rather than food, used video terminals.



The MathWorks





5

## VT100

- The VT100 was introduced by Digital Equipment Corporation (DEC) in 1978.
- The screen had a resolution of 80 characters wide by 23 lines.
- Typically, in a video terminal as with terminals that printed on paper, text would print one line at a time.
- Text would move to the next line if you sent the printer a carriage return and a line feed.



6

## Escape Control Sequences

- The cool thing about the VT100 is that it had non-printing key sequences that allowed you to move the cursor around the screen, highlight and underline text, and many other screen functions.
- We will reach back into the distant past of 1978 to dredge up ESCape key sequences to make the ASCII text debugging tool more useful.
- Along the way, we will learn escape key sequences, high tech in 1978, which will impress absolutely no one when you go for a job interview.
- However, we will make some useful debugging tools.





## Terminal Emulation

7

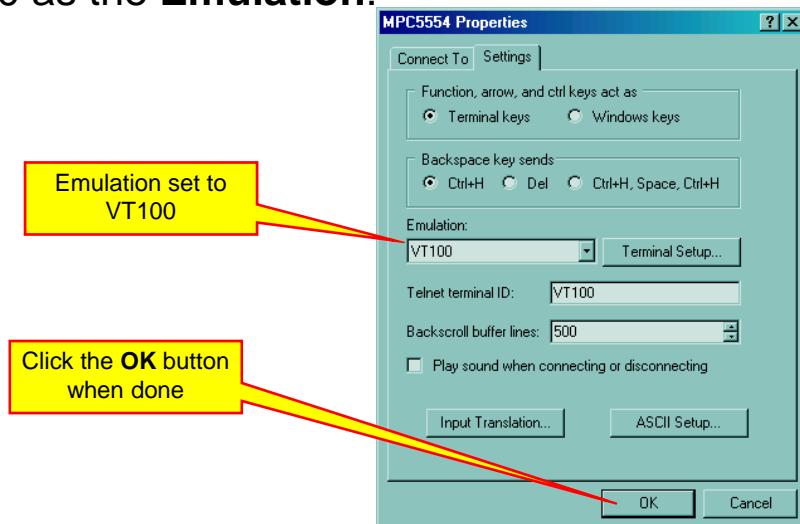
- First, we must set MS Terminal to emulate a VT100 so that it will respond to escape sequences.
- Select **File** and then **Properties** from the MS Terminal menus:



## Terminal Emulation

8

- Select the **Settings** tab and then specify VT100 as the **Emulation**:





9

## Escape Codes

- You can usually find the escape key sequences if you search on Google for “VT100 Escape Codes.”
- Some codes are
  - **<ESC>[H** - Move the cursor to the home position. (The upper left corner.)
  - **ESC>[{row};{column}H** – Move the cursor to the specified coordinates. 1,1 is the upper left corner (home). The column can go from 1 to 80, the row can go from 1 to 23.
  - **<ESC>[K** - Erase from the current cursor position to the end of the line.
  - **<ESC>[1K** - Erase from the current cursor position to the beginning of the line.
  - **<ESC>[2J** - Erase the entire screen.



The MathWorks



10

## Escape Codes

- **<ESC>[{attr1};...{attrn}m** -Set Attributes.
  - 0 Reset all attributes
  - 1 Bright
  - 2 Dim
  - 4 Underscore
  - 5 Blink
  - 7 Reverse
  - 8 Hidden
- Source: <http://www.termsys.demon.co.uk/vtansi.htm>,  
Date accessed February 1, 2010.



The MathWorks





## RHIT Library

11

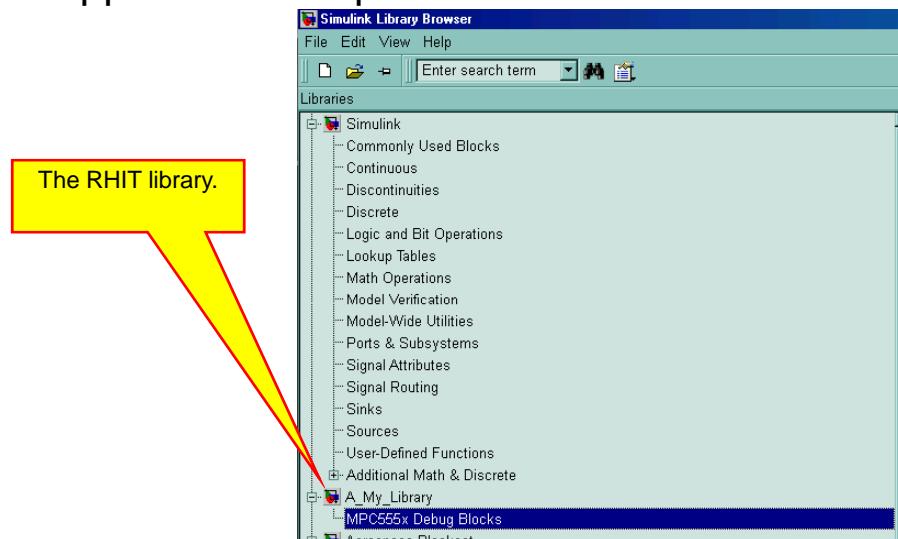
- In lecture 14, we learned how to send text strings.
- Escape key sequences are just non-printing text strings.
- We will use them to place text on the screen in specific locations and with various properties.
- We have created a library of blocks for Rose-Hulman.
- The library is named A\_My\_Library and can be found using the Simulink Library Browser.



## RHIT Library

12

- It was called A\_My\_Library so that it would appear at the top of the list:





## RHIT Library

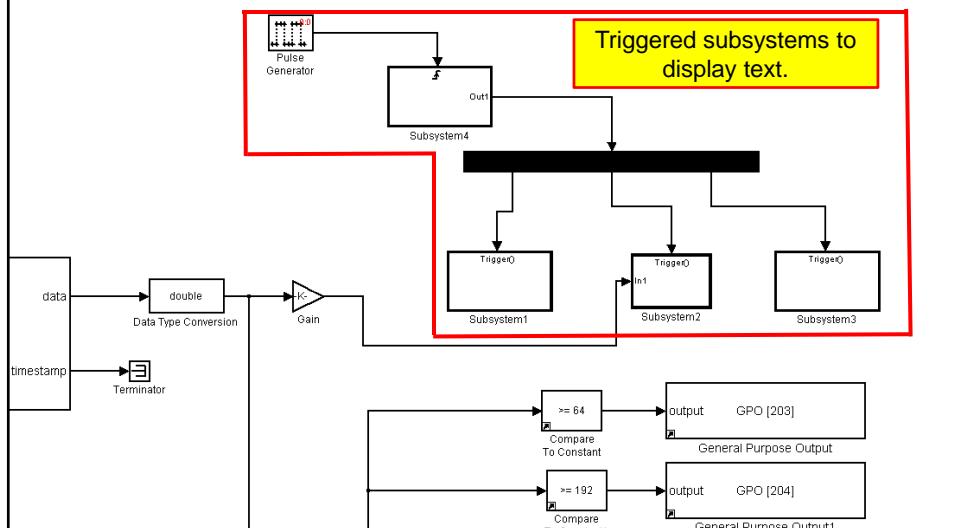
13

- The RHIT Library has a sub-library called MPC555x Debug Blocks.
- This contains a number of blocks that use the eSCI port to display text messages using the VT100 Escape codes.
- Before we start, we will open model Volt\_Meter3 (passed out as Lecture14A\_Model0).
- Save the model as Lecture14A\_Model1.mdl.

## ASCII Test String Output

14

- If you notice, this model used triggered subsystems to display a text screen on the terminal screen.

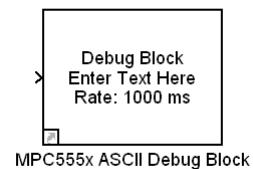




## RHIT Debug Blocks

15

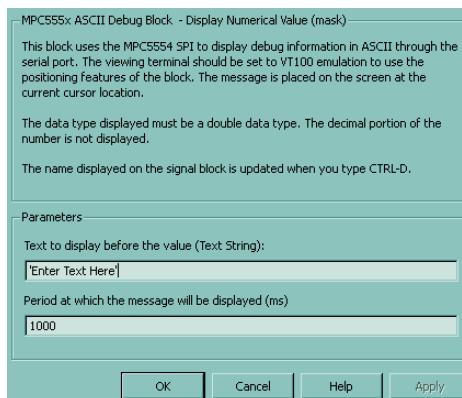
- Block **MPC555x ASCII Debug Block** in library **A\_My\_Library / MPC555x Debug Blocks** is a subsystem created from the blocks enclosed in the red box on the previous slide.
- The subsystem is masked, and passes parameters to the subsystem contained within.
- The block is shown below:



## MPC555x ASCII Debug Block

16

- The input to the block is the numerical value it will display
- If you double-click on the block, it will open as if it were any other Simulink block:
- Here you enter the text you wish to be displayed before the numeric value and the rate at which you want the value refreshed (in milliseconds).

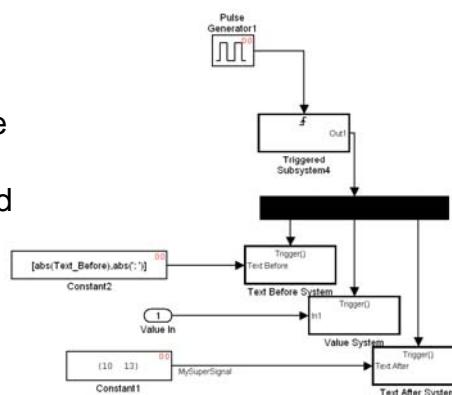




## MPC555x ASCII Debug Block

17

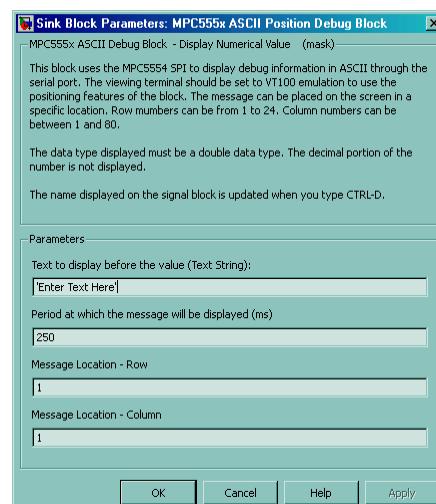
- Click the **OK** button to close the dialog box.
- If you right click on the **MPC555x ASCII Debug Block** and then select **Look Under Mask**, you will see the subsystem that implements the block:
- This is the same set of blocks that we used to display an ASCII text string in the volt meter model.
- The only difference is that the text before and pulse generator period are specified using masked subsystem parameters rather than hard coded into the constants.



## MPC555x ASCII Position Debug Block

18

- The **MPC555x ASCII Position Debug Block** is the same as the Debug Block except that it uses ESCape key sequences to position the cursor at a specified position on the VT100 screen:

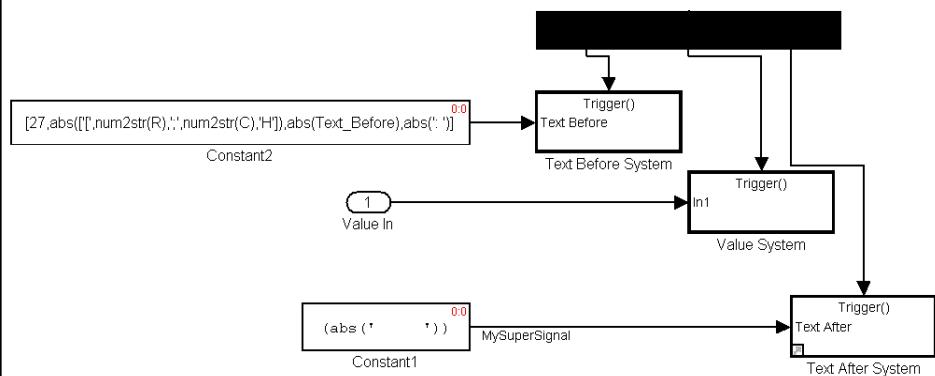




19

## MPC555x ASCII Position Debug Block

- If you look under the mask, you will see that the text string contains the sequence <ESC[ {row},{Column}H.
- Note that 27 is the ASCII code for the ESCAPE key.



20

## Debug Blocks

- Several other blocks are defined for placing text at specific locations on the screen and making the text underlined, reverse, or blinking.
- Note that the **MPC555x Debug Clear Screen** block only clears the screen and should be used as the first block called to clear the screen and remove unwanted old text.
- The input to the clear screen block is a trigger.





## Lecture 14A Problem1

21

- Add debug blocks to model Lecture14A\_Model1 to obtain the display shown on the following slide.
- Values should be updated every 250 ms.  
No unwanted text should appear on the screen.



## Lecture 14A Problem1

Demo\_\_\_\_\_

22

```
MPC5554 - HyperTerminal
File Edit View Call Transfer Help
Connected 1:56:04 | VT100 | 57600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo |
Introduction to Model-Based Design
Value Before 1/16 Gain: 7488
Value after 1/16 gain.: 468
Electrical_Engineering
BIT 0: 1
BIT 1: 1
BIT 2: 1
Rose-Hulman Institute of Technology
```





Demo\_\_\_\_\_

23

## Lecture 14A Problem2

- We would like to create a new subsystem block that displays text with the following characteristics.
- The input to the block is a numerical value of type double.
- The block should be a masked subsystem that specifies:
  - The row and column where the block is placed on the screen.
  - The text displayed on the screen before the numerical value.
  - The period in milliseconds at which the numerical value is refreshed.
  - Two threshold values:
    - When the numerical value is below Threshold1, the text is displayed in black.
    - When the numerical value is between Threshold1 and Threshold2, the text is underlined.
    - When the numerical value is above Threshold2, the text blinks.



The MathWorks



Questions?



The MathWorks





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

# Introduction to Model-Based Systems Design

## Lecture 15: Analog Output (PWM)



The MathWorks



freescale  
semiconductor



## PWM

2

- A high current variable DC supply is hard and expensive to build.
- Many systems have a fixed voltage DC supply available.
- We can make a “variable” DC supply by “chopping” a constant DC supply on and off at high frequency.
- The average value of the voltage is variable.



The MathWorks



freescale  
semiconductor

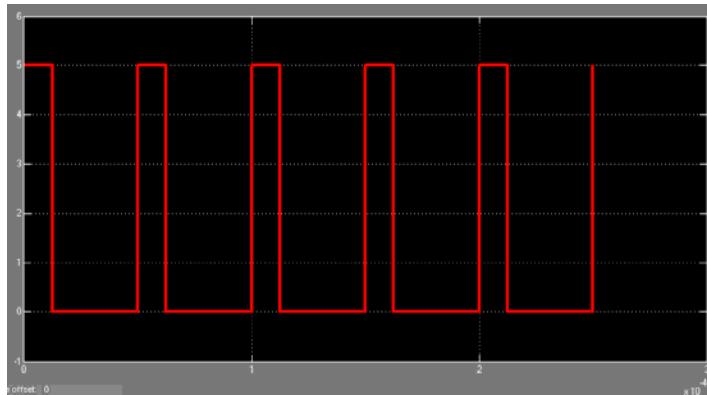
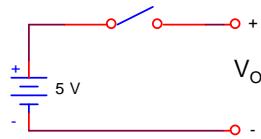




Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

## PWM

3



**MotoTron**

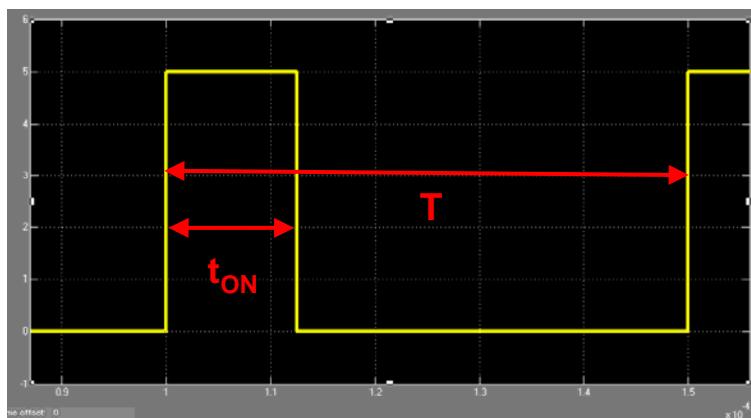
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## PWM

4



$$V_{avg} = 5V \left( \frac{t_{on}}{T} \right) = 5V \left( \frac{12.5 \mu s}{50 \mu s} \right) = 1.25V$$

**MotoTron**

The MathWorks

**freescale**  
semiconductor

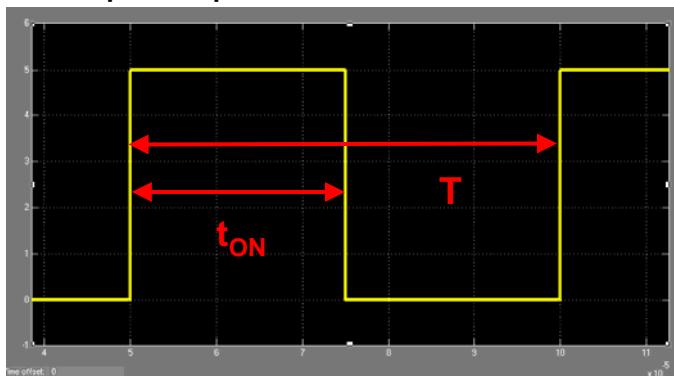
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



5

## PWM

- In a PWM waveform, we vary the on time and keep the period constant.



$$V_{avg} = 5V \left( \frac{t_{on}}{T} \right) = 5V \left( \frac{25\mu s}{50\mu s} \right) = 2.5V$$

**MotoTron**

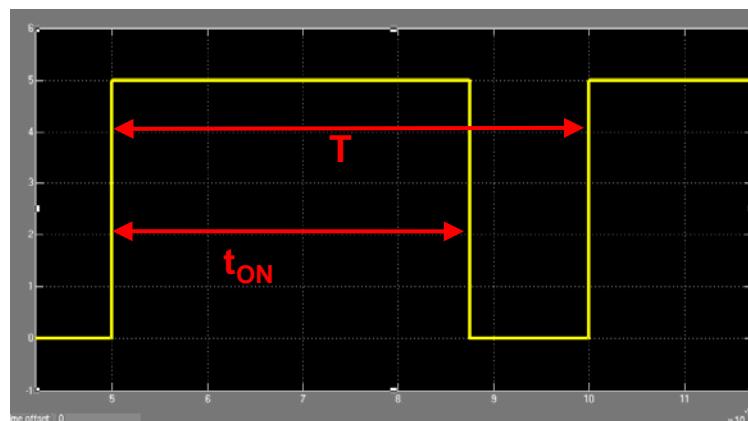
The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

6

## PWM



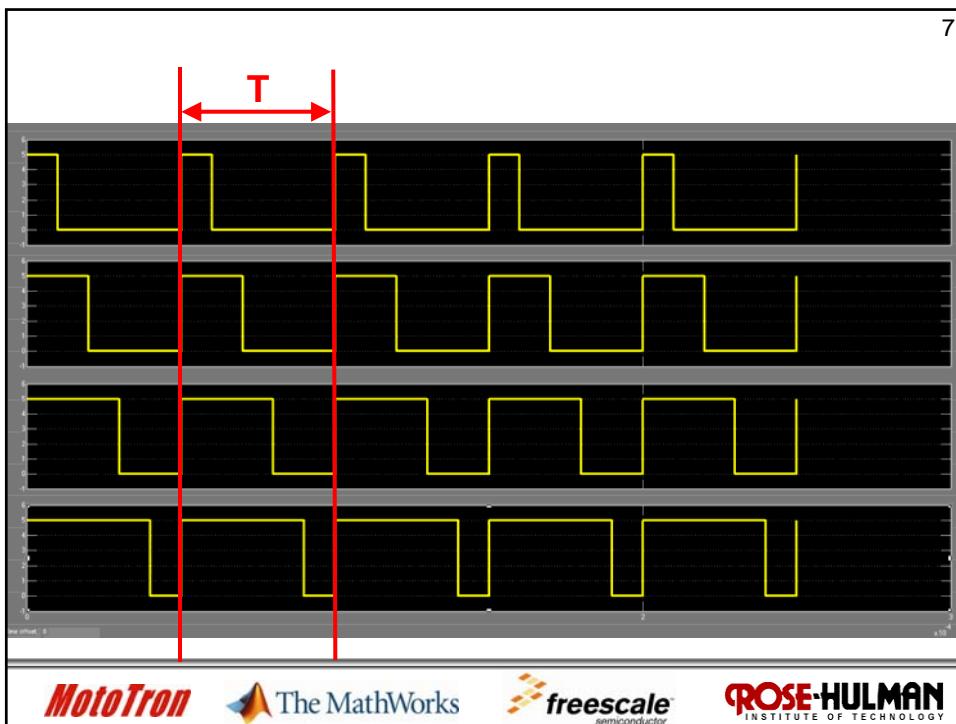
$$V_{avg} = 5V \left( \frac{t_{on}}{T} \right) = 5V \left( \frac{37.5\mu s}{50\mu s} \right) = 3.75V$$

**MotoTron**

The MathWorks

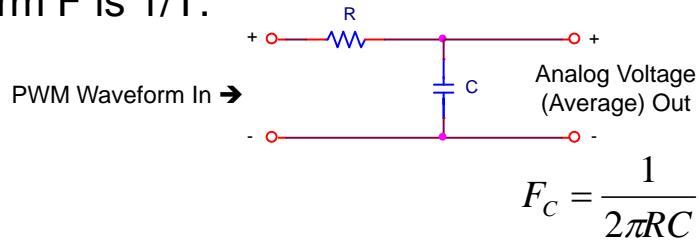
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## PWM

- From a circuit point of view, we can extract the average value using a low pass filter.
- Choose the cutoff frequency ( $F_C$ ) of the filter to be much less than the frequency of the PWM waveform.
- Note that the frequency of the PWM waveform  $F$  is  $1/T$ .





9

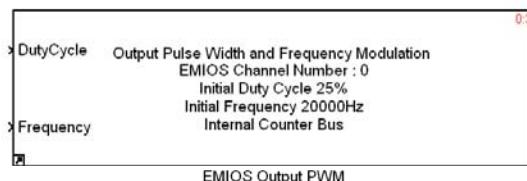
## PWM

- Note that objects with a large inertia cannot respond on a microsecond time scale.
- These items respond to the average of the waveform rather than the instantaneous value.
- Examples are solenoids and motors.
- We can speed control a motor using pulse-width modulation.



10

## MPC555x eMIOS PWM Output



- Both inputs use a **UINT32** data type.
- The duty cycle has a range of 0 to 100% and a resolution of 1%.
- The frequency input is 1 bit per Hz. (I was able to set a PWM frequency from 10 Hz to 1 MHz. See datasheet for more information.)

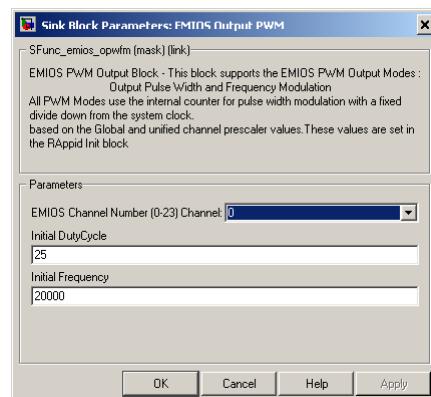




11

## MPC555x eMIOS PWM Output

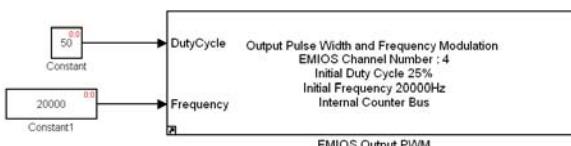
- Open the block to specify the initial duty cycle, the initial PWM frequency, and the PWM channel:
- In file MPC5554DEMO\_man\_G.pdf, search for the text EMIOS to find the pin number for your channel.



12

## Mobile Studio Desktop

- Create the model shown below.
- Build your model and download it to your MPC555x board.
- We will use Mobile Studio Desktop to observe the output waveforms for this lecture.





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

## Mobile Studio Desktop

13

- Mobile Studio Desktop is a combination hardware/software solution that replaces traditional bench top in an educational lab with a single board that plugs into your laptop's USB port.
- Instruments included are an oscilloscope, spectrum analyzer, function generator, and logic analyzer.
- The product was developed as a joint venture between Rose-Hulman Institute of Technology and Rensselaer Polytechnic Institute



## Mobile Studio Desktop

14

- The hardware board is shown below:

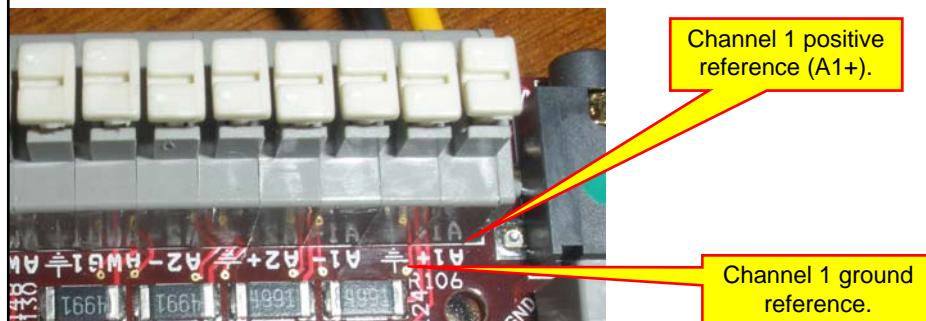




## Mobile Studio Desktop

15

- We will be using Channel 1 of the oscilloscope to display the PWM waveform.
- Hook up two clip leads to the A1+ and Ground terminals of the board as shown:



## Mobil Studio Desktop

16

- Plug in the USB cable into the board and your computer.
- Connect the A1+ terminal to the PWM output pin on your MPC555x board
- Connect the ground terminal on the Mobile Desktop Studio board to a ground pin on your MPC555x board.
- Run the Mobil Studio Desktop application from your PC.
- The application is in the Windows Start Menu at Rensselaer/Mobile Studio Desktop/Mobile Studio Desktop.





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

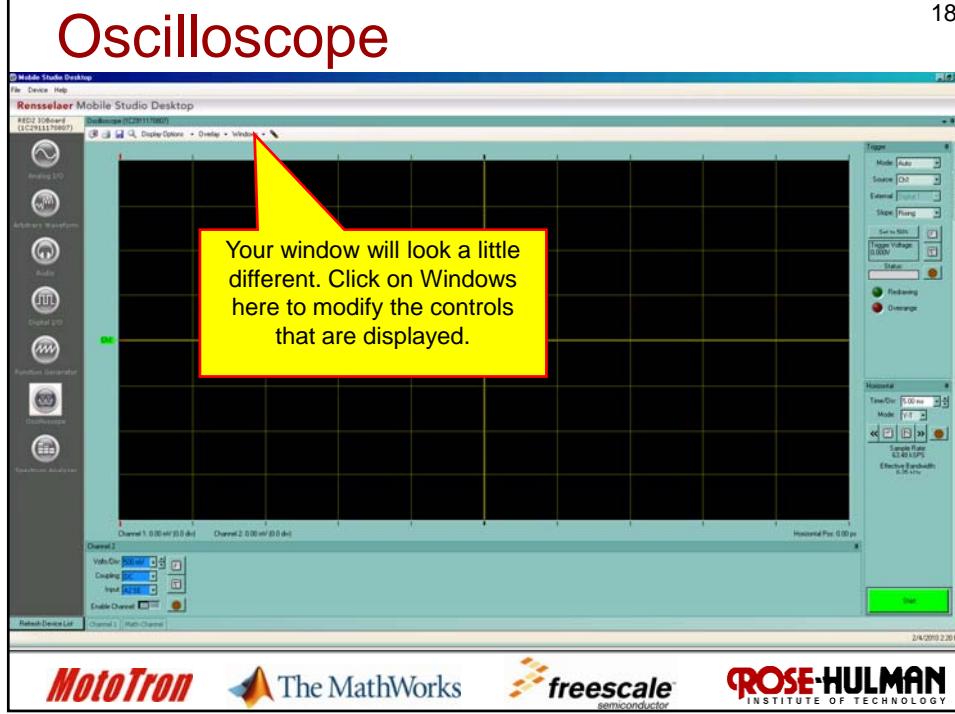
## Mobile Studio Desktop

17

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

## Oscilloscope

18

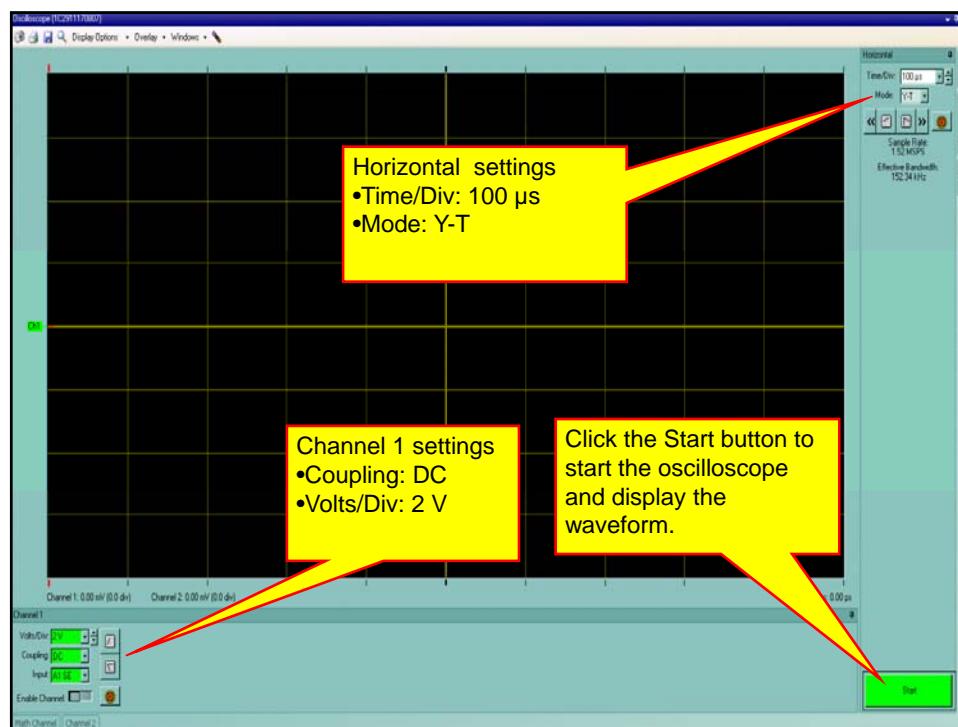
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



19

## Oscilloscope

- Select Windows and then choose to display Channel 1 and the Horizontal Settings

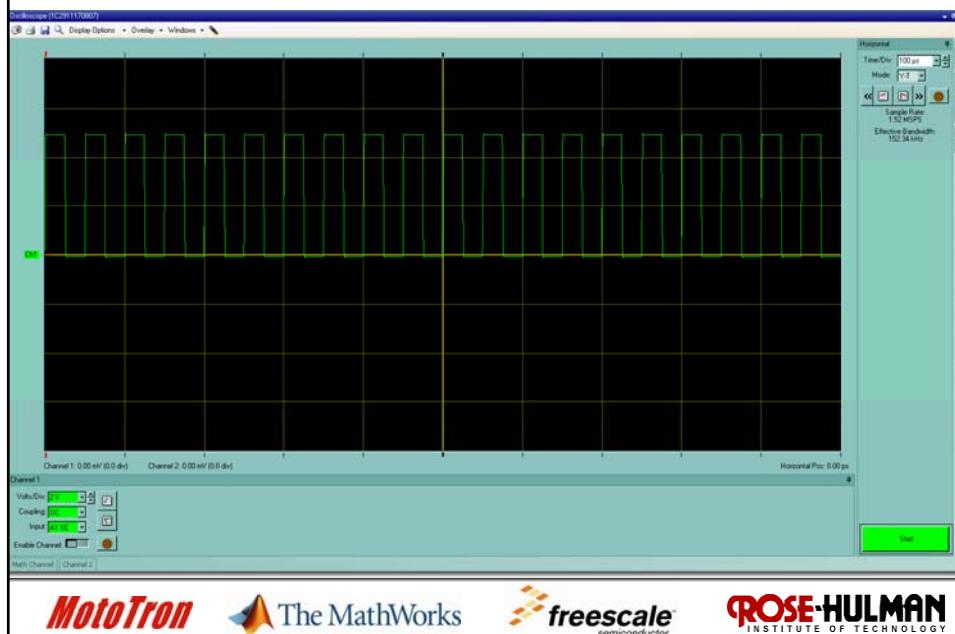
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



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

## PWM Waveform

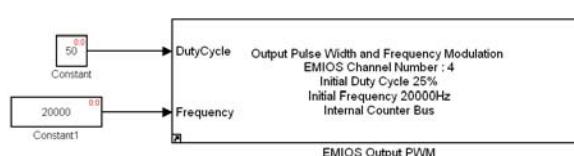
21



## Lecture 15 Demo 1

22

- MPC555x eMIOS PWM Output
- Verify that the frequency is 20 kHz and that the duty cycle is 50%.
- Display waveforms of the PWM output with the Mobile Studio Desktop Oscilloscope



Demo \_\_\_\_\_



The MathWorks

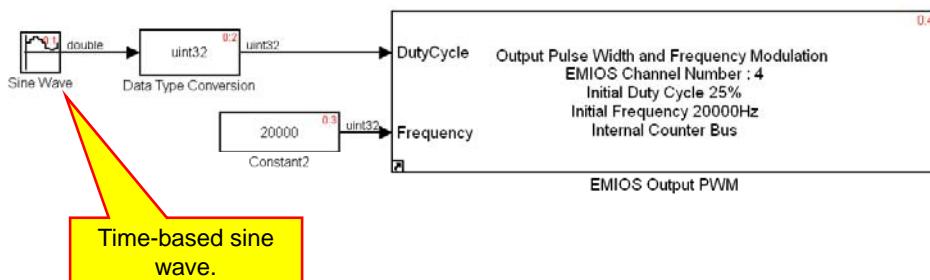




23

## MPC5554 eMIOS PWM Output

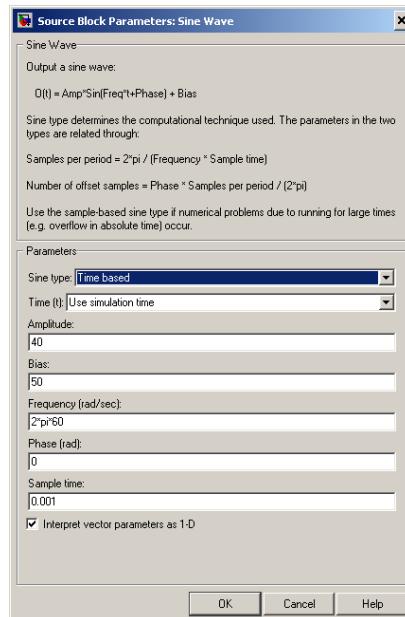
- Next, we would like to create a 60 Hz analog sine wave using the PWM output.
- Create the model shown:

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

## MPC5554 eMIOS PWM Output

24

- The settings of the sine wave are shown below. Note that the pulse width never goes to 0 or 100 %.
- Observe the output PWM waveform and show that the frequency is constant but the pulse-width changes.





25

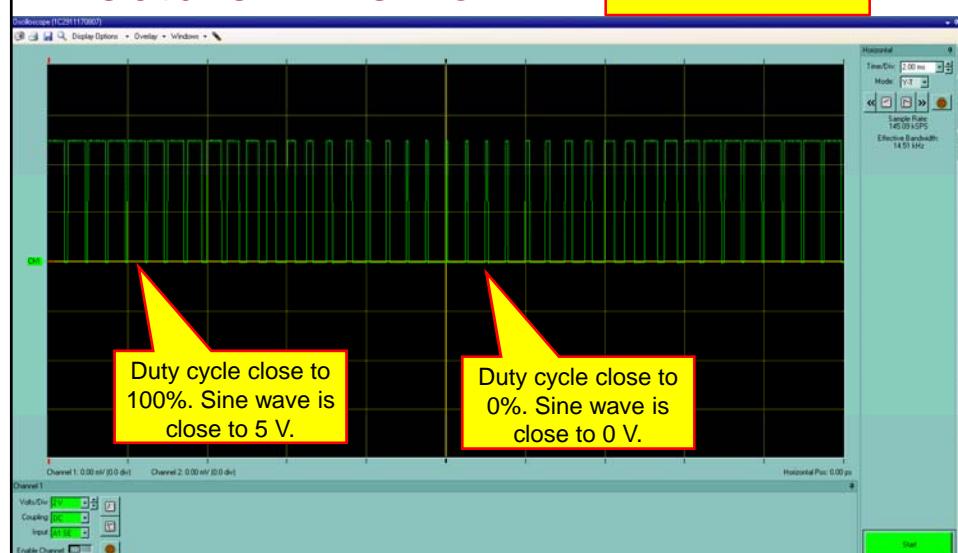
## Lecture 15 Demo 2

- Demo the PWM output of the sine wave using a 2 kHz PWM waveform.
- The duty cycle should change with the sine wave input.
- We are using a 2 kHz waveform so that we can easily see that the pulse-width change with the sine wave.
- When we filter the sine wave later, we will need to increase the PWM frequency to generate a smooth analog waveform.

26

## Lecture 2 Demo 2

Demo \_\_\_\_\_

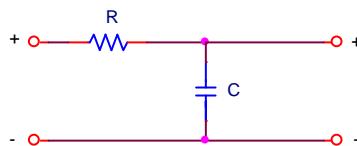




27

## Lecture 15 Demo 3

- MPC5554 eMIOS PWM Output
- Add the filter shown below and observe a 60 Hz Sine Wave at the output of the filter.
- Choose the cutoff frequency of the filter to be 600 Hz or higher. (A decade above 60 Hz)
- You may need to increase the PWM frequency to reduce the ripple on the filter output. (20 kHz or higher.)
- Display the Sine wave on Channel 1 of the scope, Display the PWM waveform on Channel 2 of the scope.



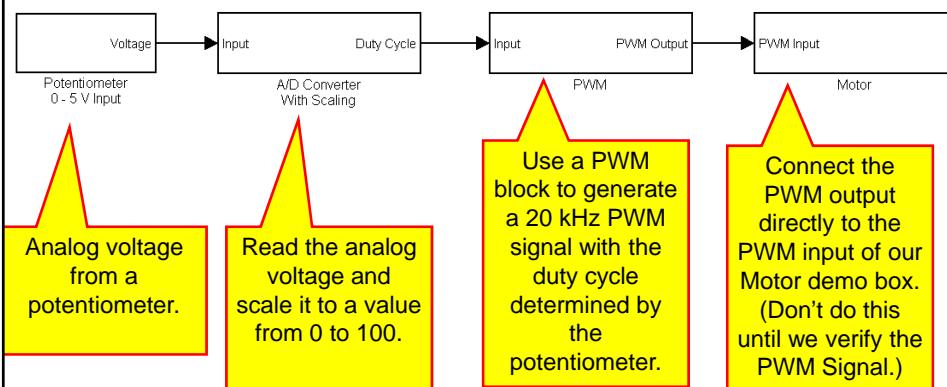
Demo\_\_\_\_\_

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

## Motor Control Demo

28

Create the following model.

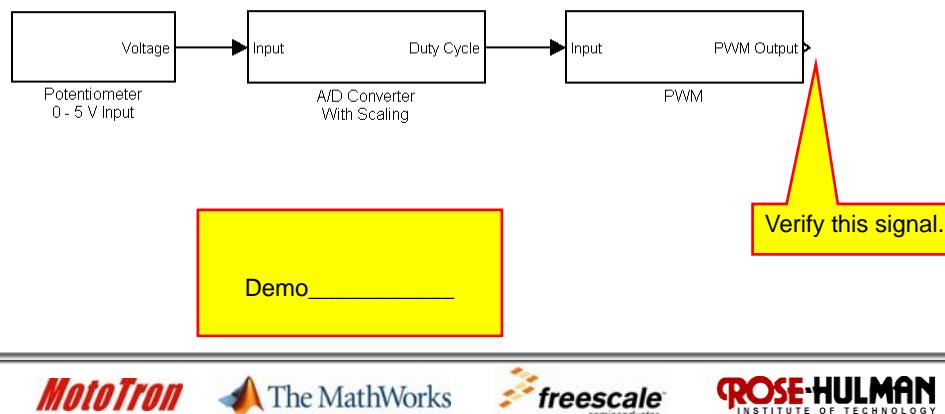
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



29

## Lecture 15 Demo 4

- 1) Before connecting the motor, verify that you can use the POT to control the output duty cycle. (Display in the scope)


**MotoTron**

The MathWorks

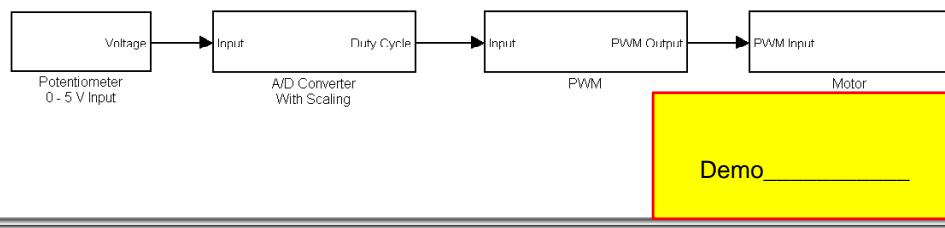
freescale

 ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY

## Motor Control Demo 5

30

- 2) Once you have verified the PWM signal on the scope, connect the PWM output of the MPC555x to the PWM input of the motor controller demo.
- 3) Verify the motor speed is controlled by the potentiometer.


**MotoTron**

The MathWorks

freescale

 ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



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

## Questions?



The MathWorks





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



## Introduction to Model-Based Systems Design

### Lecture 15A: Hardware In The loop Simulations



## HIL

2

- Up to this point we have:
  - Learned several levels of simulations: Model-in-the-Loop (MIL), Software-in-the-Loop (SIL) and Real-Time.
  - Learned several software packages: MATLAB, Simulink, RAppID, FreeMASTER,
  - Used many platforms: Windows, xPC
  - Used several different hardware targets: PC, xPC, MPC5554.





## HIL

3

- It is now time to put it all together and perform Hardware-in-the-loop (HIL) simulations.
- We will start with the full model developed in Lecture 10A and split the model so that:
  - The controller runs in the MPC5554 target.
  - The plant runs on the Speedgoat xPC target.
- The two targets will be connected with a wiring harness, the same harness that will be used in the final product.
- Both models will run in real time.



## HIL

4

- 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 Harness - It is connected to the plant using the same interface that will be used in the final implementation.
- 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.





## HIL

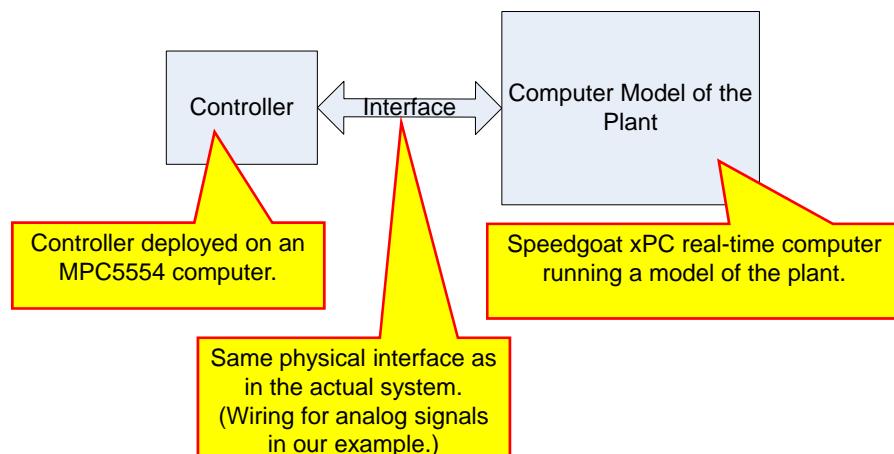
5

- We will start with Lecture10A\_Model3.mdl and split it into two models, the plant and the controller.
- The plant will:
  - Run on the xPC Target.
  - The xPC Target: Speedgoat I/O Driver blocks will be used to create a shell to interface between the model and the physical world.
  - The inputs and outputs will be analog voltages.
- The controller will
  - Run on an MPC5554 target.
  - Use RAppID to interface between the model and physical world.
  - The inputs and outputs will be analog voltages.

## HIL

6

- We will be using the test platform below:



The MathWorks





7

## HIL

- We will split the development of the HIL system into several steps:
  1. We will deploy the controller on to the MPC5554 target.
  2. We will deploy the plant on to the xPC target.
  3. We will test the plant with analog inputs and verify its operation.
  4. We will connect the controller and plant together using a wiring harness.
  5. We will test the entire system.



8

## HIL

- If the controller on the MPC5554 target works successfully in the HIL simulation:
  - We will take the final step of testing the controller by connecting it to the physical plant.
  - If it works, we will drink (depending on your state of mind):
    - Champagne
    - An energy drink
    - Mountain Dew (non-diet for the full compliment of sugar)
    - Or all of the above in no specific order.





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

9

## HIL Simulations

### Part 1: Implementing the Controller on the MPC5554 Target



## Motor Controller Deployment

10

- From lectures 11 through 15, we now know how to use the hardware resources of the MPC555x 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 MIL, SIL, and real-time portions of the class.
- First, we will create a shell that accesses the hardware resources of our target (MPC555x).





11

## Motor Controller Deployment

- The controllers we developed for our motor-generator system all used the following input:
  - Speed Set point – A continuous (analog) signal between 0 and 1. We will use one of the potentiometers on the MPC555x board for this reference. We read this as a 5 V signal and then scale it to 0 to 1 in the MPC555x.



12

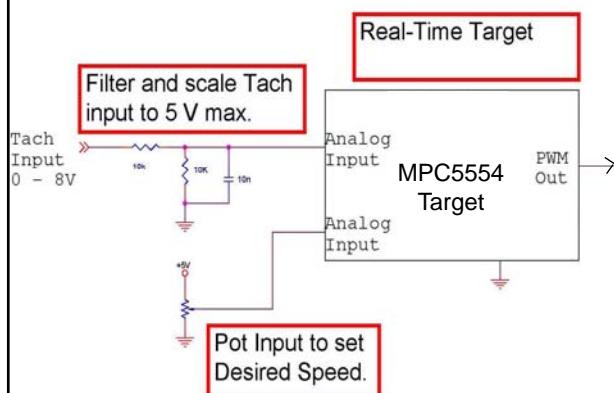
## Motor Controller Deployment

- The controllers we developed for our motor-generator system all used the following input:
  - Motor rpm – An analog signal measured from the motor rpm sensor (shaft encoder). The encoder outputs a signal that is 2.5 V per 1000 rpm. We scale this signal to 0 to 4 V with resistors and then read the signal with the MPC555x. Internal to the MPC555x, the signal is scaled to a number between 0 and 1.





## Controller Realization on the MPC5554 Target



## Motor Controller Deployment

14

- The controllers we developed for our motor-generator system all used the following output:
  - Motor torque request. An analog signal between 0 and 1. The realization will be a PWM signal between 0 and 100%
  - The actual input to the plant is a PWM input that directly controls the gate of a MOSFET chopper circuit.
  - The PWM output of the controller directly modulates the waveform to the motor.





## Hardware Shell

15

- We will now create a top-level “shell” for our controller that:
  - Initializes the MPC555x.
  - Reads and scales the analog inputs.
  - Passes the information to a subsystem that contains our control method.
  - Outputs a PWM signal for the requested torque.



The MathWorks



## Hardware Shell

16

- 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.)



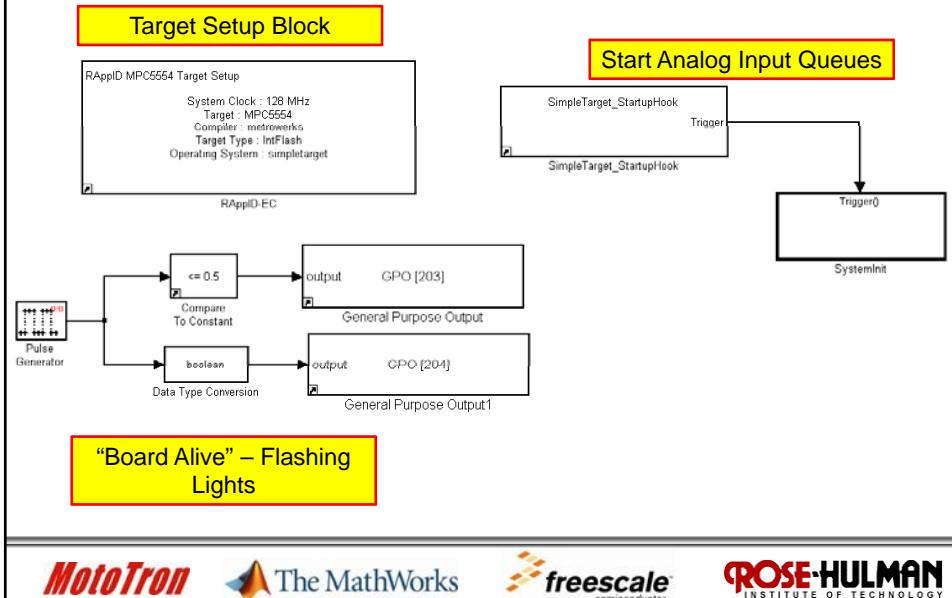
The MathWorks





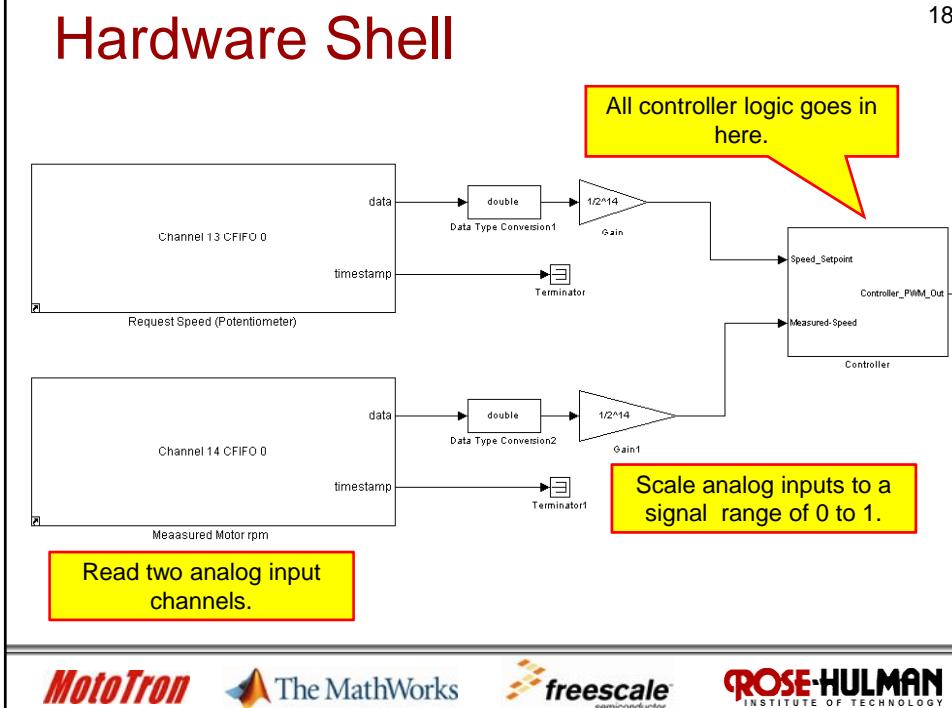
## Hardware Shell

17

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

## Hardware Shell

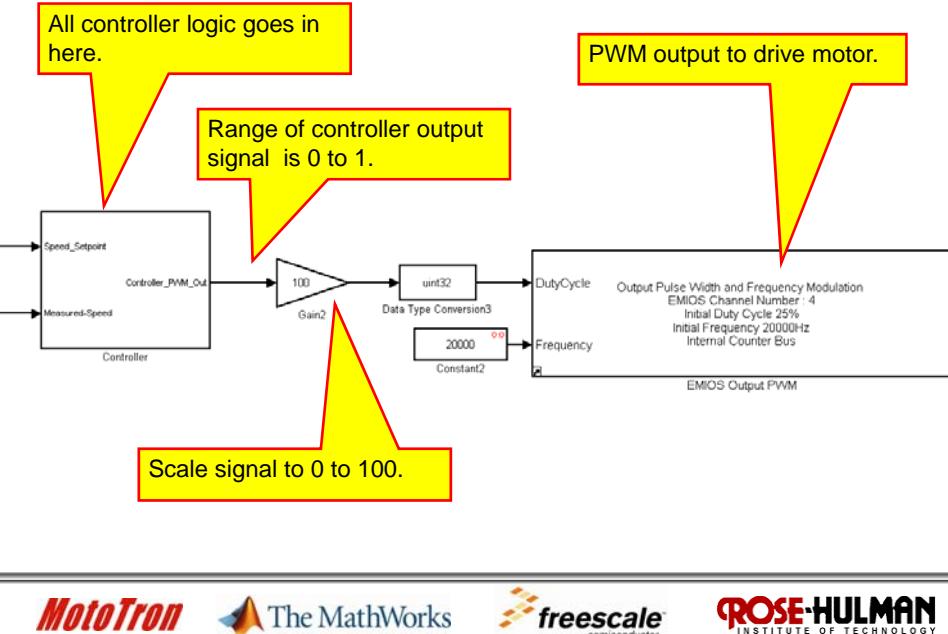
18

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



## Hardware Shell

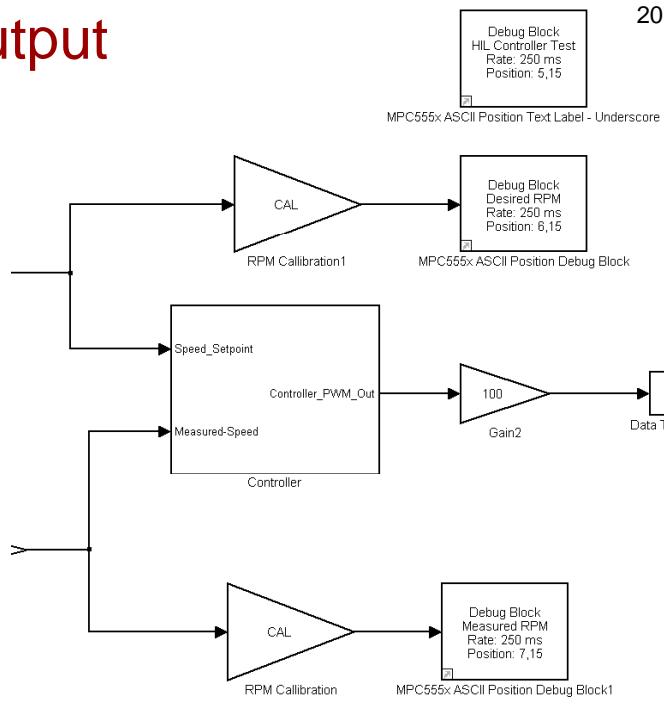
19

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

## ASCII Output

20

- As a last enhancement, we will display the desired and measured rpm using RHIT Debug blocks we used earlier.

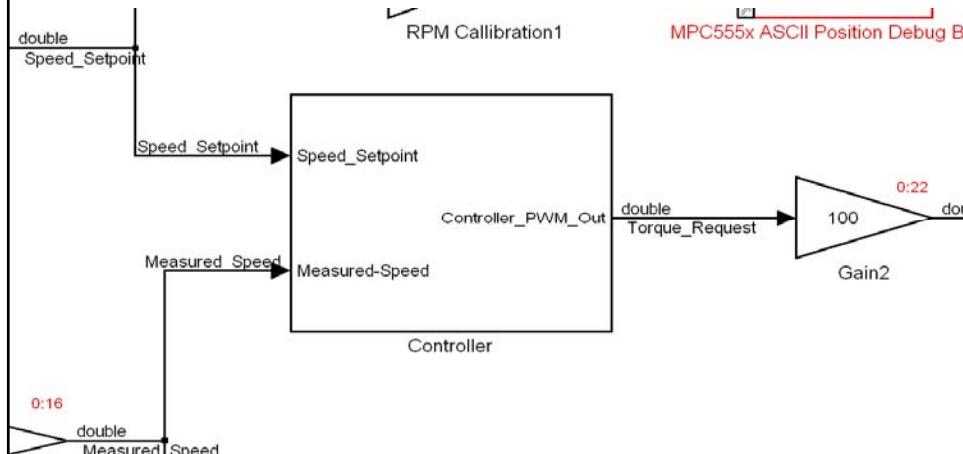




## FreeMASTER

21

- It is also suggested that you use FreeMASTER to monitor signals within the controller including the Speed\_Setpoint, Measured\_Speed, and the controller Torque\_Request output.



## Debugging Tools

22

- Note that you might not be able to use both the FreeMASTER tool and the RHIT debug blocks at the same time as they both use the eSCI port.
- You can attempt to use both. However, if they are incompatible, you should use the one you feel most comfortable with and the one that gives you the most information and help in debugging the system.



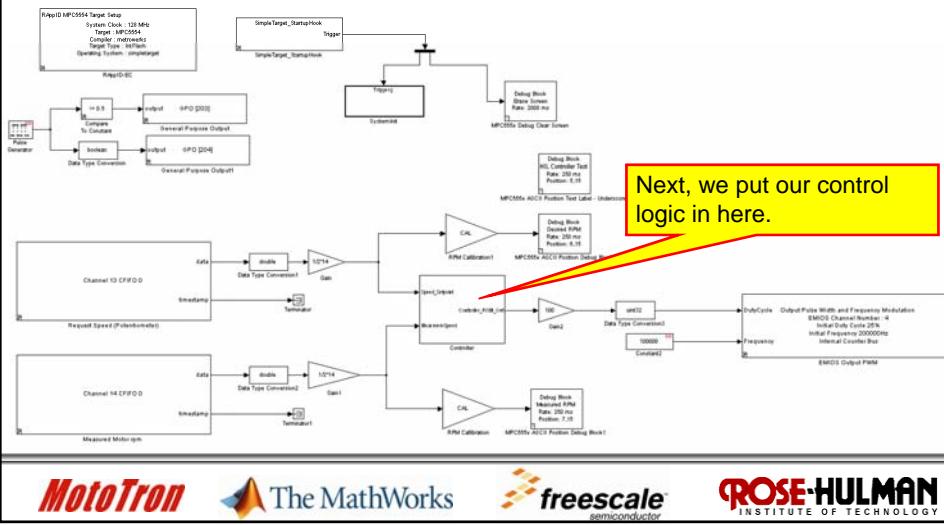
The MathWorks

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



23

## Complete Hardware Shell


**MotoTron**
**The MathWorks**
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Controller

24

- We now will use the controller from Lecture10A\_Model3.
- Note that this model was an SIL test and had two versions of the controller, an s-function and a Simulink model. We will be using the Simulink model, as it will be compiled and downloaded to the MPC555x target.
- The idea is that we proved the controller in Lecture 10A in many simulation methods and this is the most mature of our controller models.
- In the HIL simulation we are about to run, except for any misunderstanding of the hardware target performance or the wiring interface between the controller and plant, the controller performance should be close to what we observed during our previous simulations.

**MotoTron**
**The MathWorks**
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



25

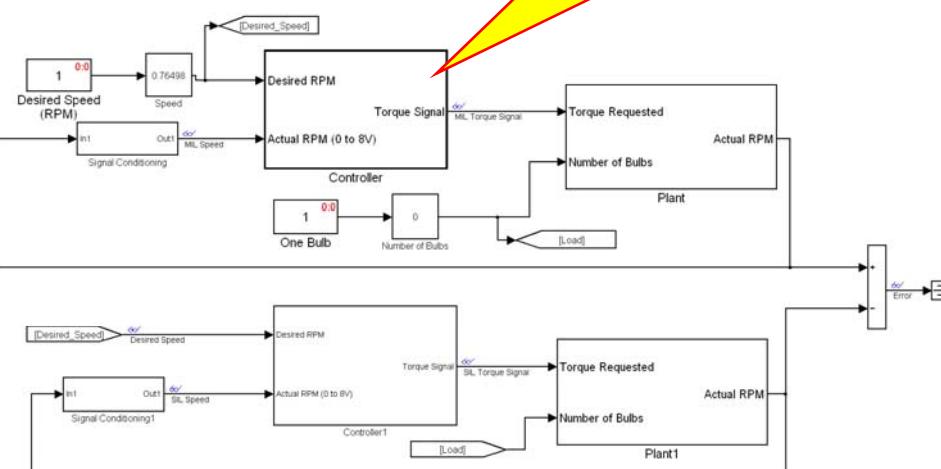
## Controller.

- Lecture10A\_Model3 was the most mature controller we developed.
- The last tests we did were our SIL simulations.
- The model is shown on the next few slides.

26

## SIL Model

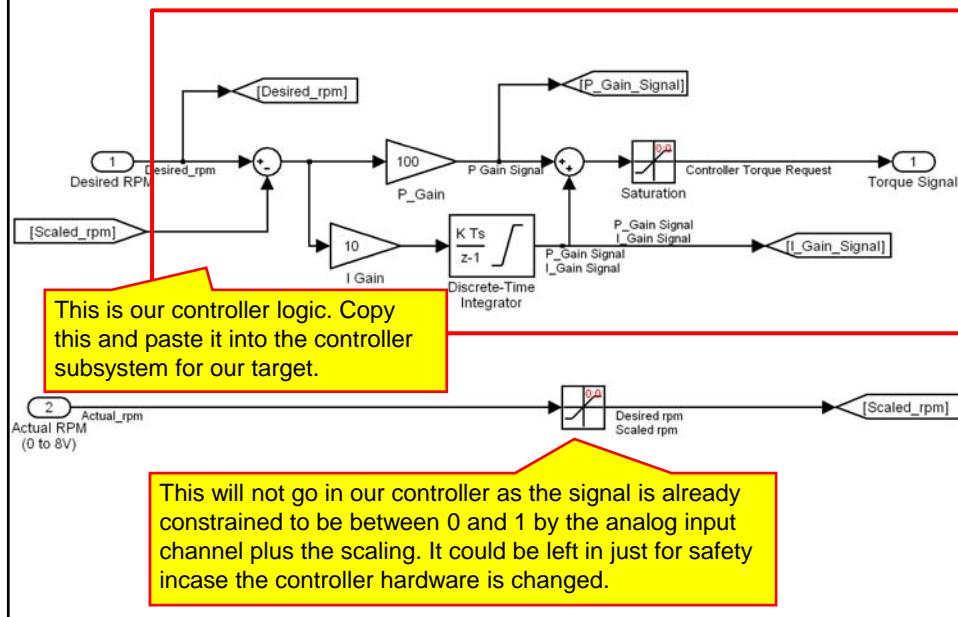
Most of the stuff in here goes in our controller in the target. (Look inside on next slide.)





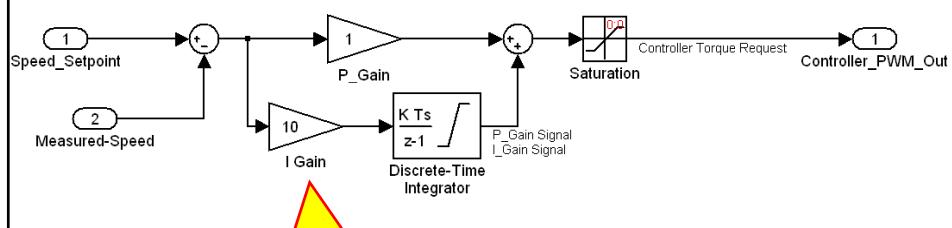
## SIL Model - Controller

27



## MPC555x Target - Controller

28





## Controller

29

- Build the model and download it to the MPC5554 target.
- We will not be able to test it yet, but you should verify that the two LEDs flash and toggle back and forth.
- We are now ready to work on the plant.



The MathWorks



## HIL Simulations

30

### Part 2: Implementing the Plant on the Speedgoat xPC Target



The MathWorks





## Plant Model

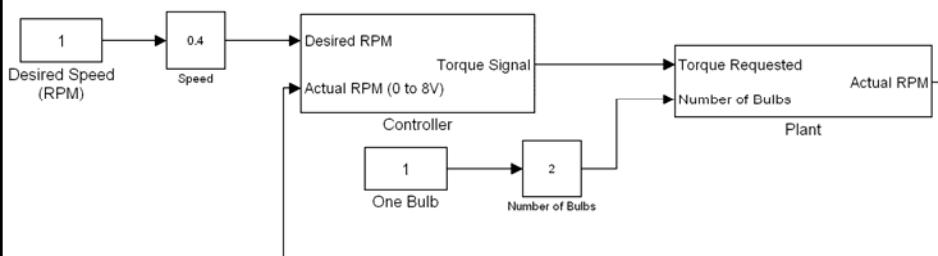
31

- In Lecture 10, we simulated the entire model (plant and controller) using xPC Target.
- Here we want to run just the plant on xPC target.
- The same plant model was used for Lecture 10 and Lecture 10A.
- Thus, we will use model Lecture 10 as a starting point.
- Most of the settings used in lecture 10 will apply here as well.



## Lecture 10 Model 3

32



- Resave this model as Lecture15A\_Plant.mdl

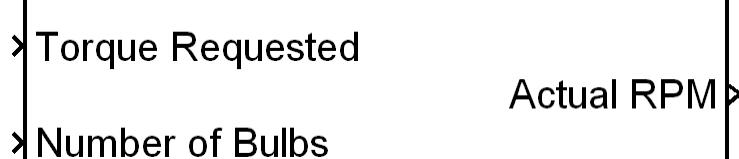




## Plant Model

33

- At the top level, delete the controller and control inputs.



Plant



## Plant Model

34

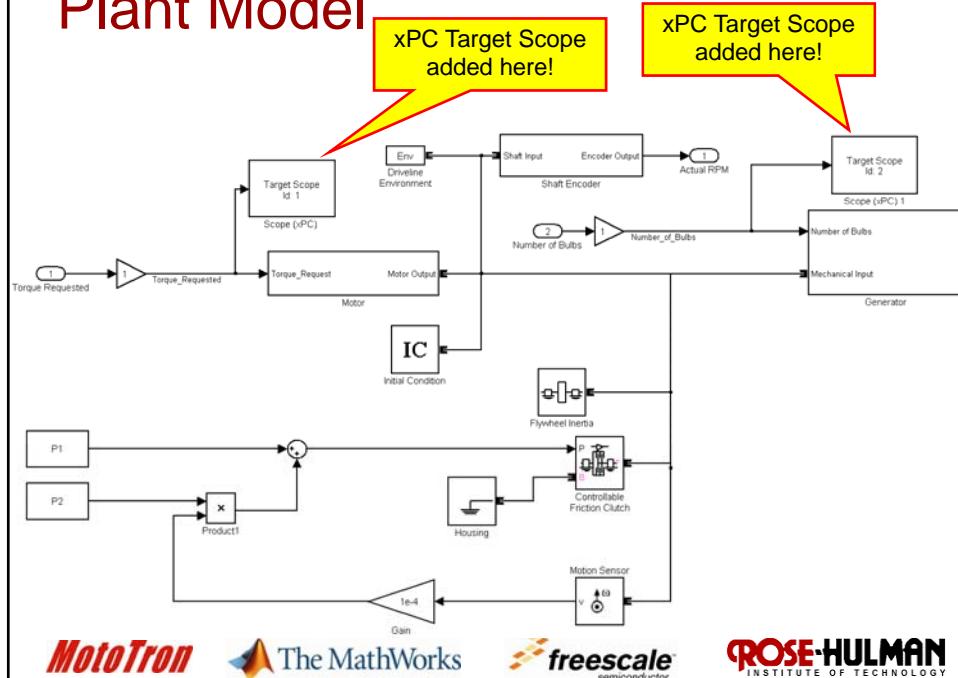
- Inside the plant is a single xPC Target Scope displaying the motor speed in rpm.
- Add xPC Target scopes to display:
  - The Torque Request input to the plant.
  - The Number of Bulbs input to the plant.
  - The encoder output in volts.





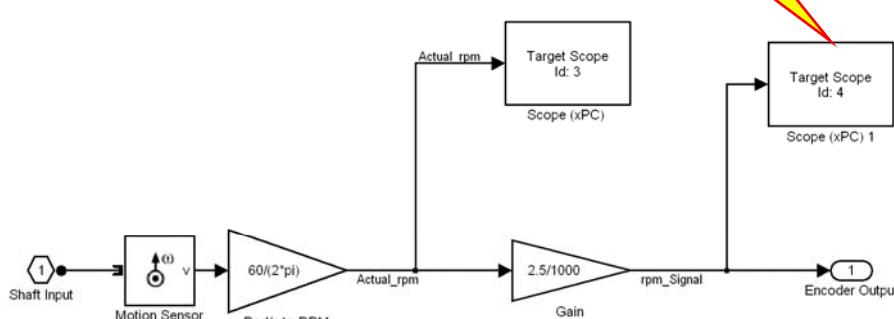
## Plant Model

35

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

## Encoder Model

36

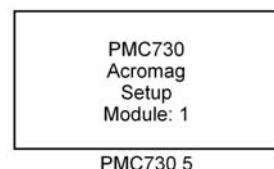
**MotoTron****The MathWorks****freescale**  
semiconductor**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



37

## Speedgoat target.

- We are going to use the Speedgoat Target libraries and I/O blocks to provide physical inputs and outputs for our plant model.
- We must first configure the hardware on the Speedgoat PCM730 board.
- We did this in Lecture 3A, but we will review it here.
- Place a Simulink block named **PMC730 5** located in the **xPC Target: Speedgoat I/O Driver Library / IO101** library in your model.



PMC730 5



38

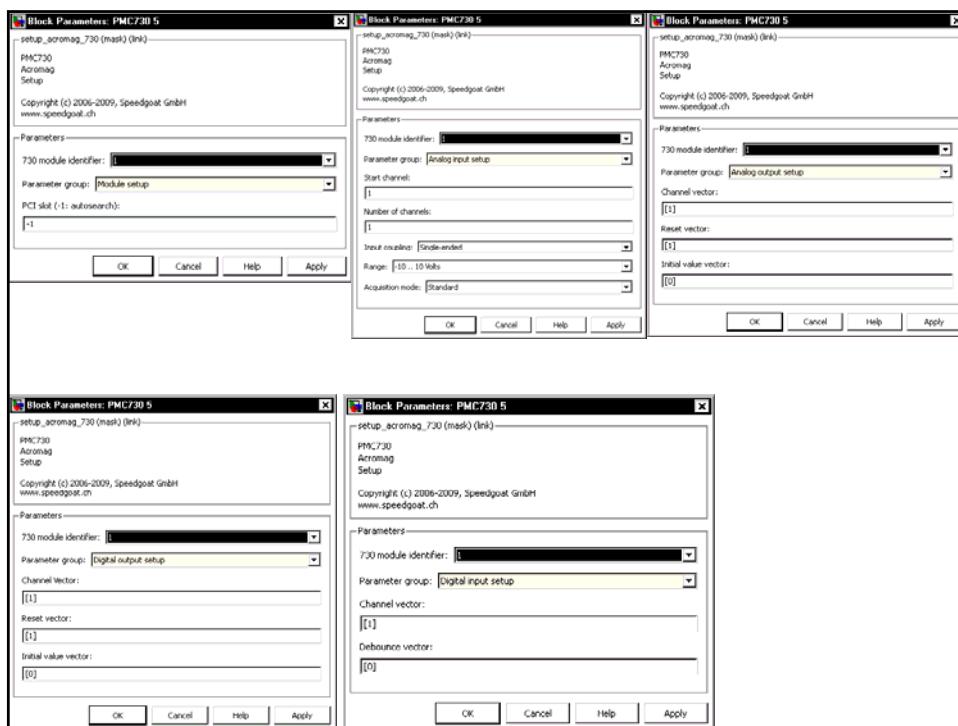
## PMC730 Acromag Setup

- This single block is used to setup all functions of the PMC730 card.
- Only 1 copy of this block is needed to set up the analog I/O and digital I/O facilities of the card.
- Double-click on the block to open it.
- Note that the block will appear differently depending on the parameter group selected.
- The screen captures on the next page show the same block with different Parameter groups selected.





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



## PMC730 5 Acromag Setup Bblock

40

- We notice that the module identifier in all of the screen captures on the previous slide was set to 1.
- If a Speedgoat target has more than one PMC730 target, we can use this card to address each PMC730 individually.
- Our targets have a single PMC730 installed, so the module identifier will be specified as 1.
- We will now set the parameters on this block to have a single analog input and a single analog output.

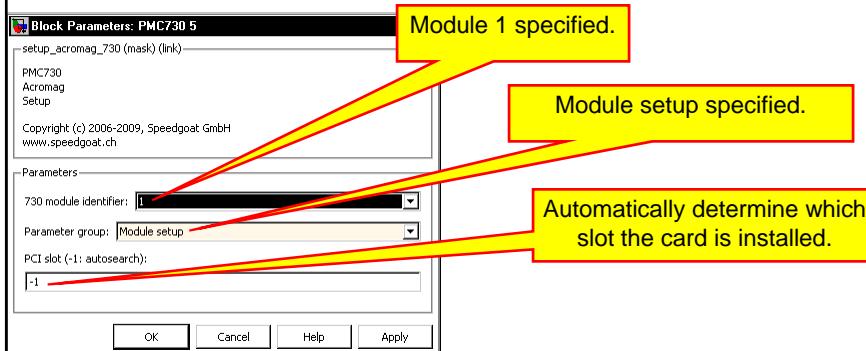




41

## PMC730 5 Acromag setup block

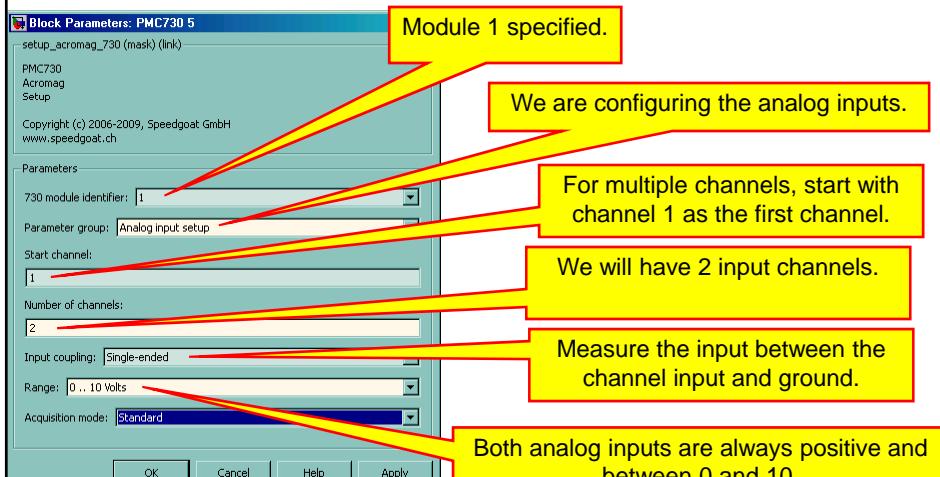
- We first must specify the Module setup.
- Since we only have a single PMC730 module, we can specify that xPC automatically determine in which slot the PMC730 is installed. If we had more than one PMC730 installed, we would specify the slot manually:



42

## PMC730 5 Acromag setup block

- We need to specify two channels for analog input:

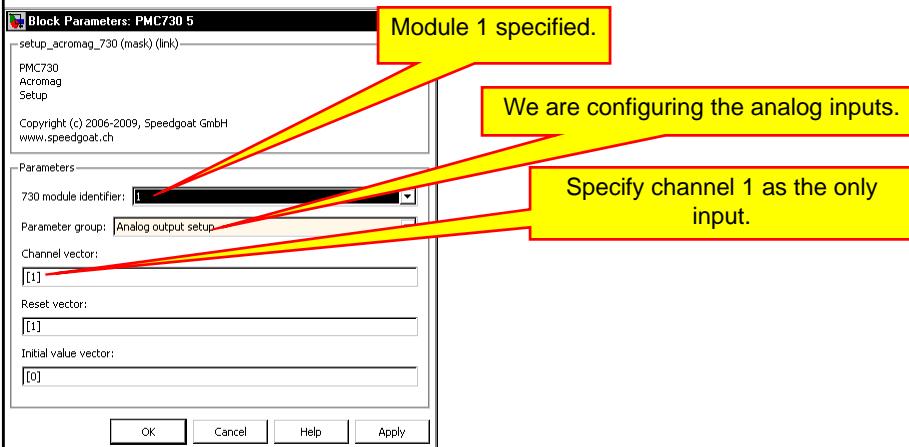




43

## PMC730 5 Acromag setup block

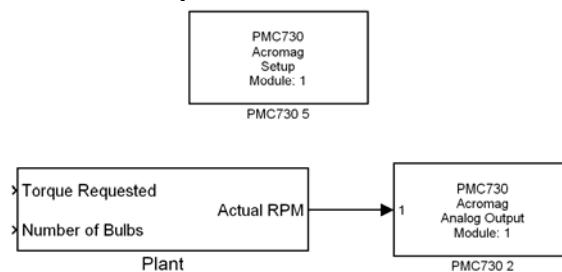
- We need to specify one channel for analog output:



44

## Analog Output

- Next, we will add an analog output block to the plant output.
- Place a block called **PMC730 2** located in the **xPC Target: Speedgoat I/O Driver Library / IO101** library in your model.
- Connect it to the plant as shown:

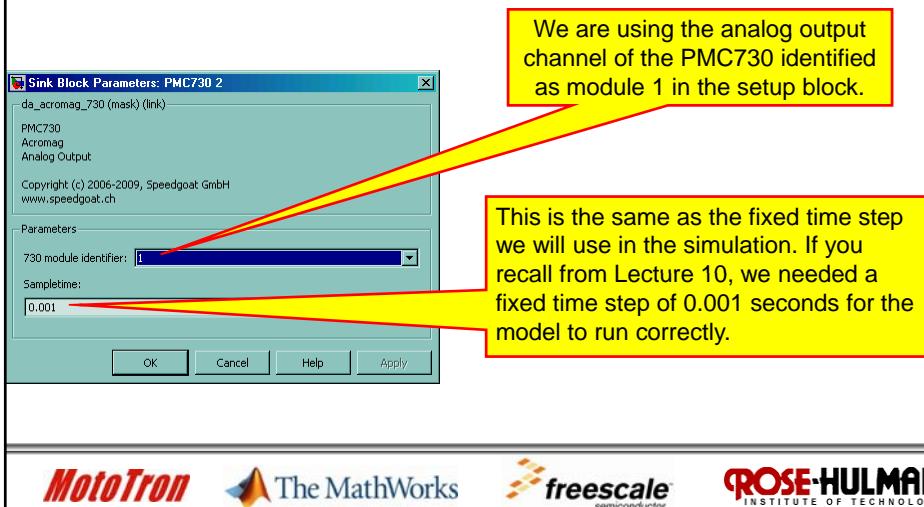




## Analog Output

45

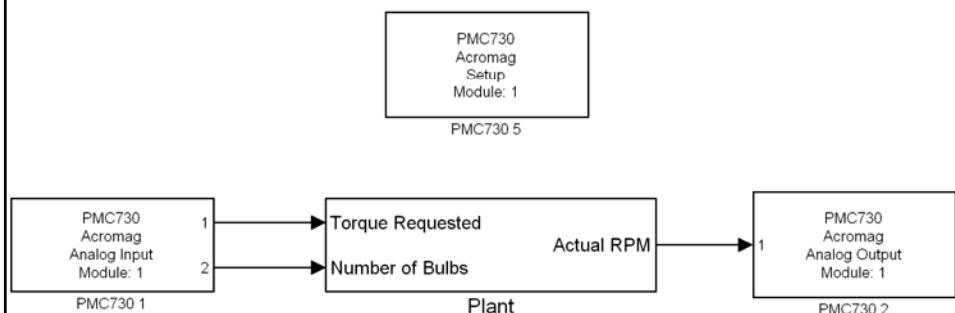
- The settings for the analog output block are:

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

## Analog Input

46

- We will use an analog input channels to measure the torque request and number of bulbs.
- Place a block called **PMC730 1** located in the **xPC Target: Speedgoat I/O Driver Library / IO101** library in your model.
- After placing the block, if you type Ctrl-d, two inputs will be shown on the block. This is because in the Acromag block, we specified two analog input channels.

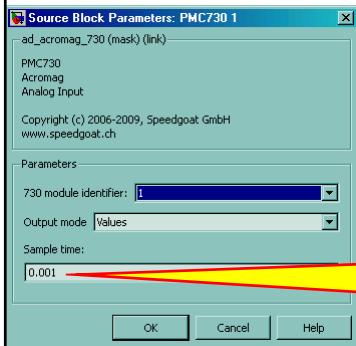




47

## Analog Input

- The properties of the analog input block are:



This is the same as the fixed time step we will use in the simulation. If you recall from Lecture 10, we needed a fixed time step of 0.001 seconds for the model to run correctly.

**MotoTron**

The MathWorks

freescale<sup>®</sup>

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY

48

## Plant Model

- We are now done with the hardware setup for the plant.
- Since we copied this model from the xPC model developed in Lecture 10, the simulation parameters should be set up correctly.
- However, just to make sure the settings are correct, we will show the required settings in the next two slides.

**MotoTron**

The MathWorks

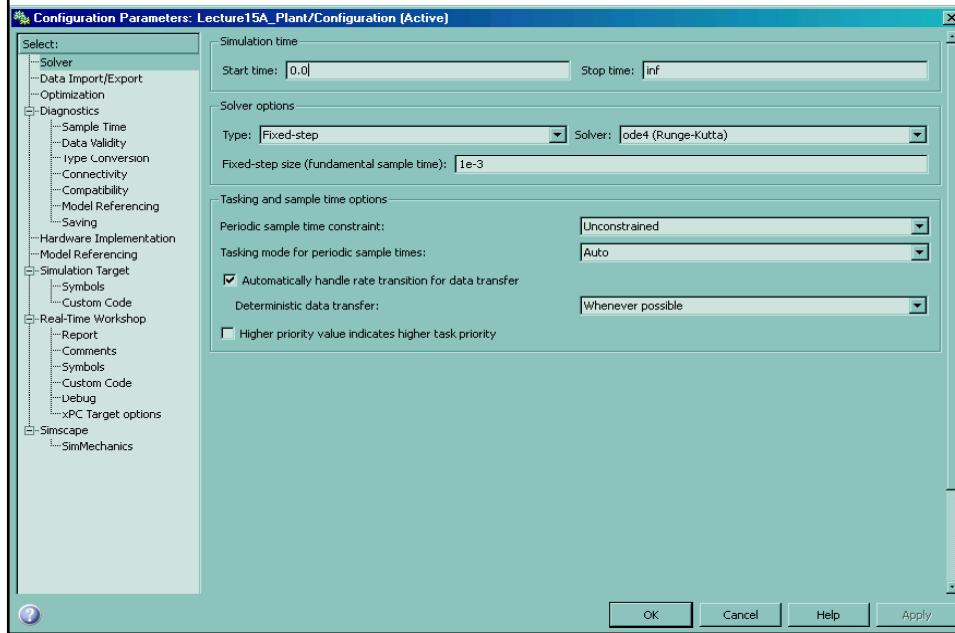
freescale<sup>®</sup>

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



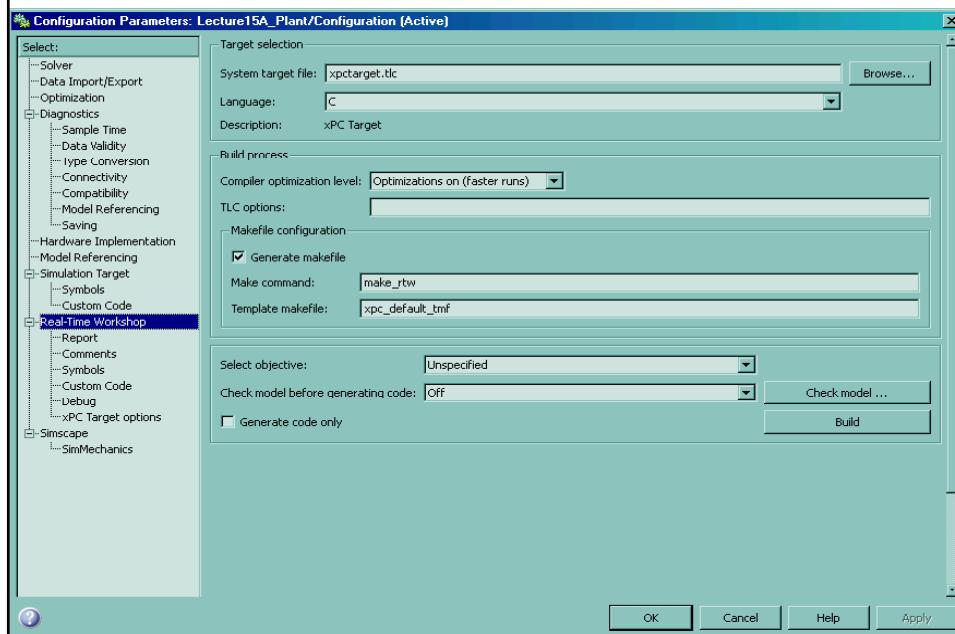
## Simulation Parameters

49



## Simulation Parameters

50





## Plant Deployment

51

- Build and download the plant model to the Speedgoat xPC target.
- Connect to the target.
- Remember to use the xPC Explorer (xpceexplr) to specify the target IP address. (Review lectures 3A and 10 if needed.)



## Model Wiring

52

- The last thing we need to do before running the model is to identify the pin numbers for the inputs and output.
- Pin connections for the PMC730 are defined in the users manual located on the website for MBSD1.
- We are using Analog Output 1.
- We need to also find the ground reference for this output.
- The data sheet is shown on the next slide:





53

## Analog Output Pin Connection

Table 2.2: PMC730 Field I/O Pin Connections

Pin Description	Pin	Pin Description	Pin
Counter Output	1	COMMON	
Dig CH0/ ADC Trigger In	2	Dig CH8/ ADC Trig Out	36
Dig CH1/ DAC Trigger In	3	Dig CH9/ DAC Trig Out	37
Dig CH2/ Counter Input	4	Digital CH10/ Counter Output	38
Dig CH3/ Counter Trig In	5	Digital CH11	39
Dig CH4/ Counter Ext Clk	6	Digital CH12	40
Dig CH5/ Counter Gate off	7	Digital CH13	41
Digital CH6	8	Digital CH14	42
Digital CH7	9	Digital CH15	43
COMMON	10	Analog Out CH4	44
COMMON	11	Analog Out CH5	45
Analog Out CH0	12	COMMON	46
Analog Out CH1	13	COMMON	47
Analog Out CH2	14	COMMON	48
Analog Out CH3	15	COMMON	49
COMMON	16	Analog Out CH6	50
COMMON	17	Analog Out CH7	51
COMMON	18	SENSE	52
Analog In S15/D15+	19	Analog In S31/D15-	53
Analog In S14/D14+	20	Analog In S30/D14-	54
Analog In S13/D13+	21	Analog In S29/D13-	55
Analog In S12/D12+	22	Analog In S28/D12-	56
Analog In S11/D11+	23	Analog In S27/D11-	57
Analog In S10/D10+	24	Analog In S26/D10-	58
Analog In S9/D9+	25	Analog In S25/D9-	59
Analog In S8/D8+	26	Analog In S24/D8-	60
Analog In S7/D7+	27	Analog In S23/D7-	61
Analog In S6/D6+	28	Analog In S22/D6-	62
Analog In S5/D5+	29	Analog In S21/D5-	63
Analog In S4/D4+	30	Analog In S20/D4-	64
Analog In S3/D3+	31	Analog In S19/D3-	65
Analog In S2/D2+	32	Analog In S18/D2-	66
Analog In S1/D1+	33	Analog In S17/D1-	67

Pin connection information is contained in table 2.2.

Analog output channels listed here.

## Analog Output Pin Connection

We specified the first analog output channel. Thus, our output is pin 12.

COMMON	10	Analog Out CH4	44
COMMON	11	Analog Out CH5	45
Analog Out CH0	12	COMMON	46
Analog Out CH1	13	COMMON	47
Analog Out CH2	14	COMMON	48
Analog Out CH3	15	COMMON	49
COMMON	16	Analog Out CH6	50
COMMON	17	Analog Out CH7	51
COMMON	18	SENSE	52

We can use any of these common connections as our ground reference.

54



## Analog Input Pin Connection

55

We can use any of the common connections specified in the previous slide as our ground reference.

Analog In S9/D9+	25	Analog In S25/D9-	59
Analog In S8/D8+	26	Analog In S24/D8-	60
Analog In S7/D7+	27	Analog In S23/D7-	61
Analog In S6/D6+	28	Analog In S22/D6-	62
Analog In S5/D5+	29	Analog In S21/D5-	63
Analog In S4/D4+	30	Analog In S20/D4-	64
Analog In S3/D3+	31	Analog In S19/D3-	65
Analog In S2/D2+	32	Analog In S18/D2-	66
Analog In S1/D1+	33	Analog In S17/D1-	67
Analog In S0/D0+	34	Analog In S16/D0-	68

We specified the two analog input channels. Thus, our inputs are pins 34 and 33. Analog In 0 is the first input and is the torque request. Analog in 1 is the second input as is the number of bulbs.

## HIL Simulations

56

### Part 3: Testing the Stand-Alone Plant



The MathWorks





## Plant Test

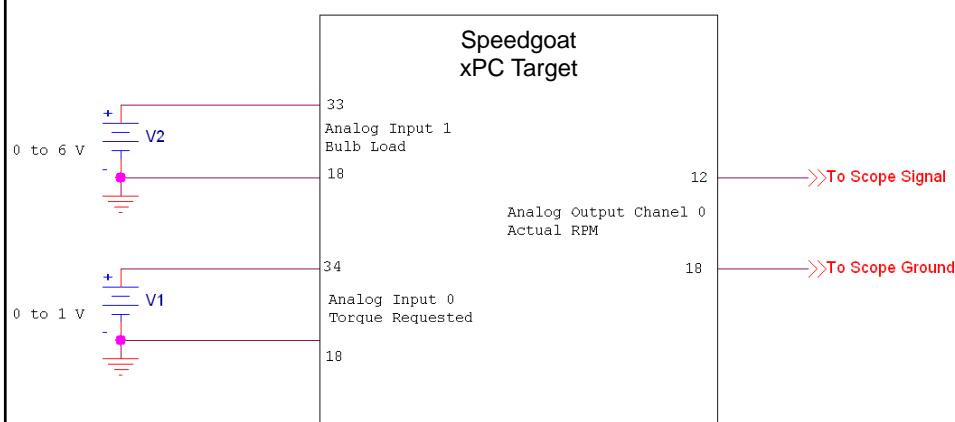
57

- We are now ready to test our plant.
- As a first step, we will use two DC voltage sources to test the inputs of the plant.
- We will connect a 0 to 6 V DC voltage source to the Bulb Load input (Analog input 1 pin 33).
- We will connect a 0 to 1 V DC voltage source to the Torque Requested input signal (Analog input 0 pins 34).
- We will observe the rpm signal (channel Analog out 0 pin 12) with a scope.
- Note that ground is pin 18 (and others if needed).



## Plant Test

58

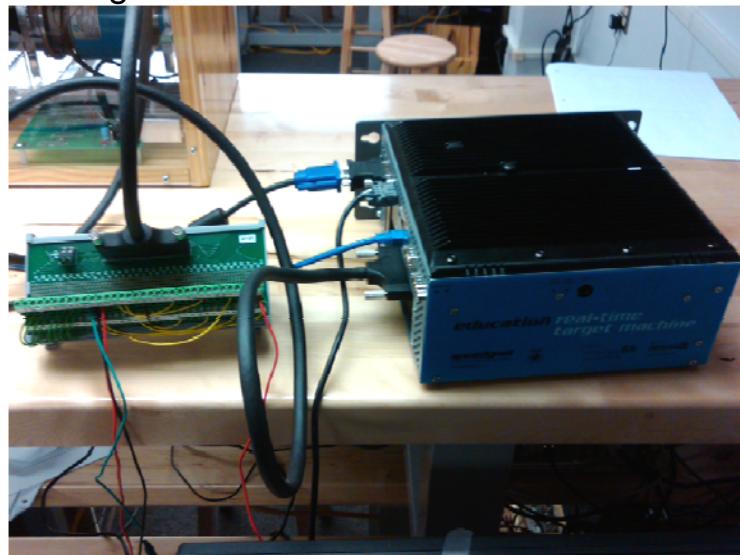




## Plant Test

59

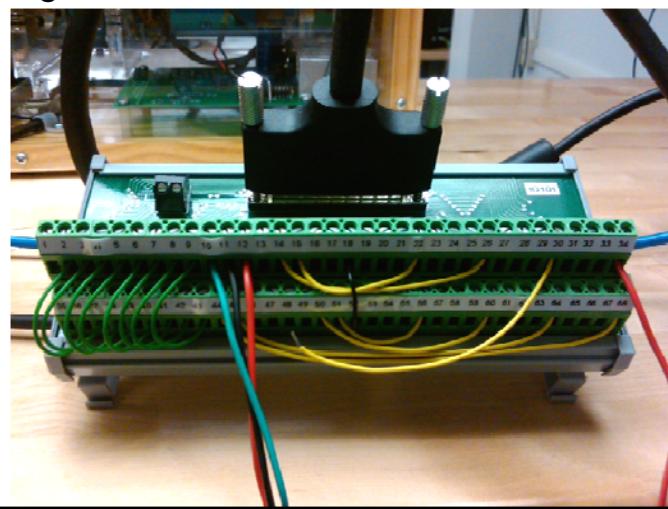
- The Speedgoat Target is connected to the outside world through a cable and a breakout connector:



## Plant Test

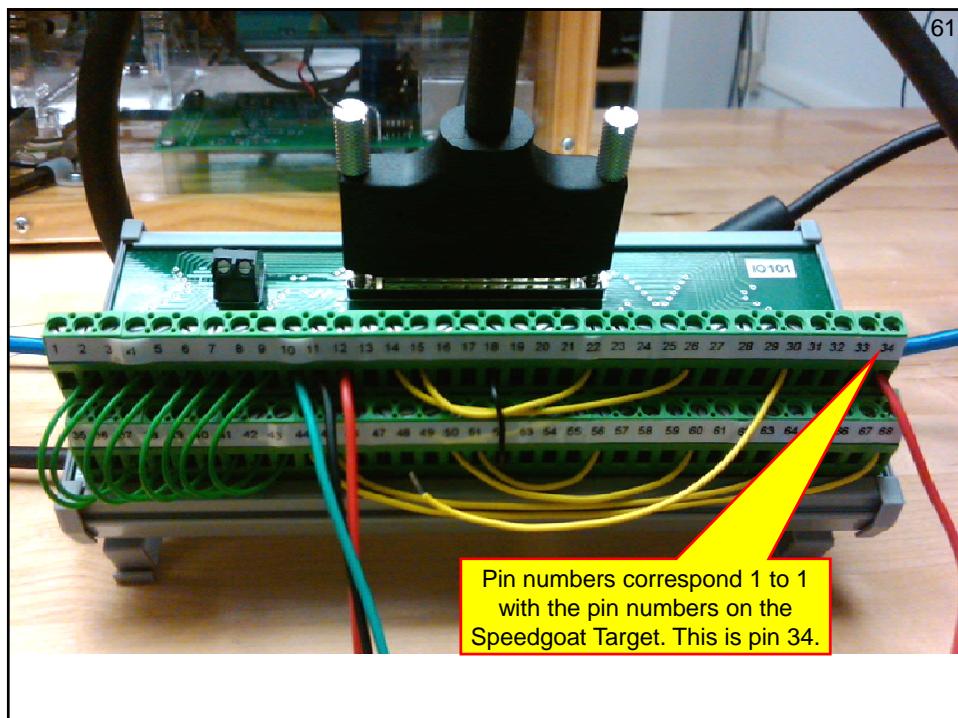
60

- The pin numbers on the breakout connector match 1 to 1 with the pin numbers on the Speedgoat target:
- This connector allows easy access to the inputs through screw terminals.





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

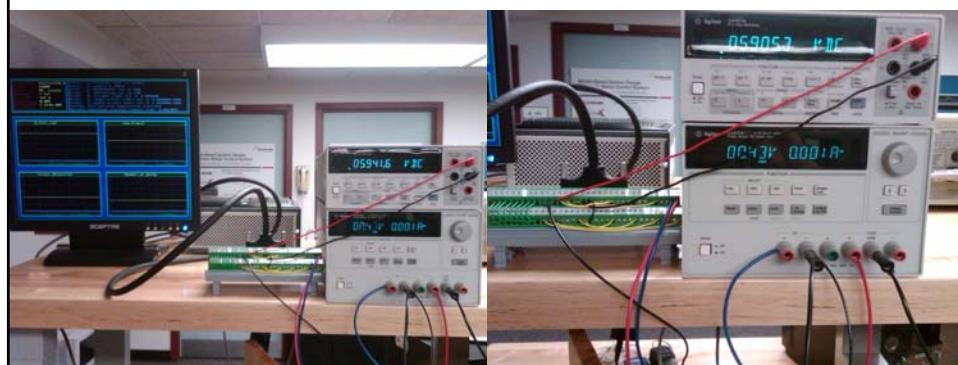


61

## Plant Test

62

- For the first test, we will just connect the two DC power supplies directly to the breakout box with jumper wires and alligator clips.





63

## Plant Test

- Connect to the Speedgoat xPC target
- Run the model.



The MathWorks



64

## Bulb Load

- As you change the DC input for the bulb load, you should see the indicators change as integer steps even though the DC source is an analog voltage and changes continuously.
- This is because we specified that these indicators change in integer steps and are limited to 0 to 6 volts.
- Remember that we used a floor block in our Simulink model for the bulb input.



The MathWorks



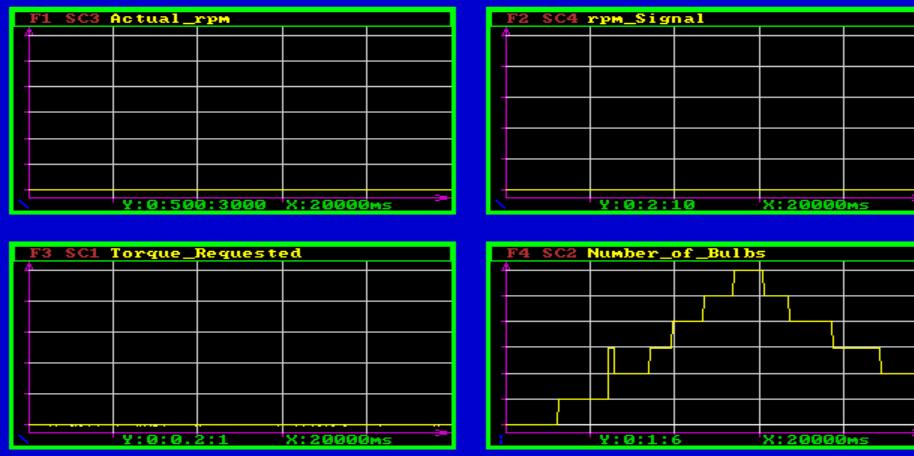


## Bulb Load

65

```

Loaded App: Lecture15a_P
Memory: 244MB
Mode: RT, single
Logging: t tet
StopTime: Inf d
SampleTime: 0.001
AverageTET: 7.523e-005
Execution: 96.60 s
Scope: 2, created, type is target
Scope: 2, signal 3 added
Scope: 2, NumSamples set to 500
Scope: 2, decimation set to 200
Scope: 2, trigger level set to 0.000000
Scope: 2, lower y-axis limit set to 0.000000e+000
Scope: 2, upper y-axis limit set to 6.000000e+000
System: initializing application finished
System: execution started (sample time: 0.001000)
  
```



## Torque Request

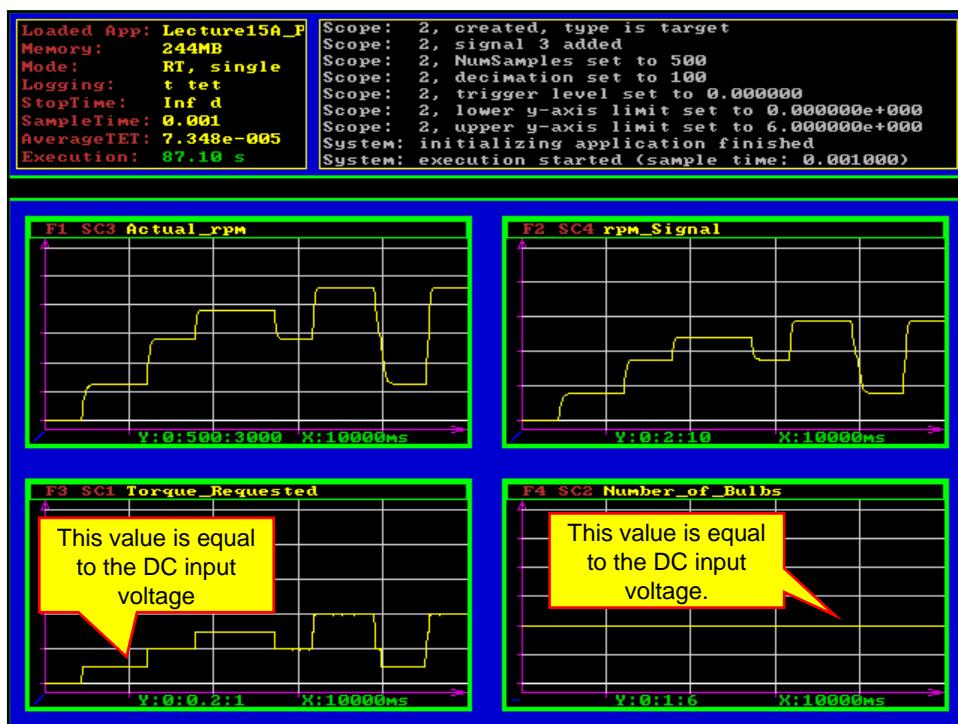
66

- As you change the torque request from 0 to 1, the indicators should follow and agree with the input voltage displayed on the DC source.
- The stepped nature of the waveform shown in the next slide is because the DC source I am using is set to change its voltage in 0.1 V steps.(It has nothing to do with how we set up the LabVIEW indicators.)



The MathWorks

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



## Plant Test

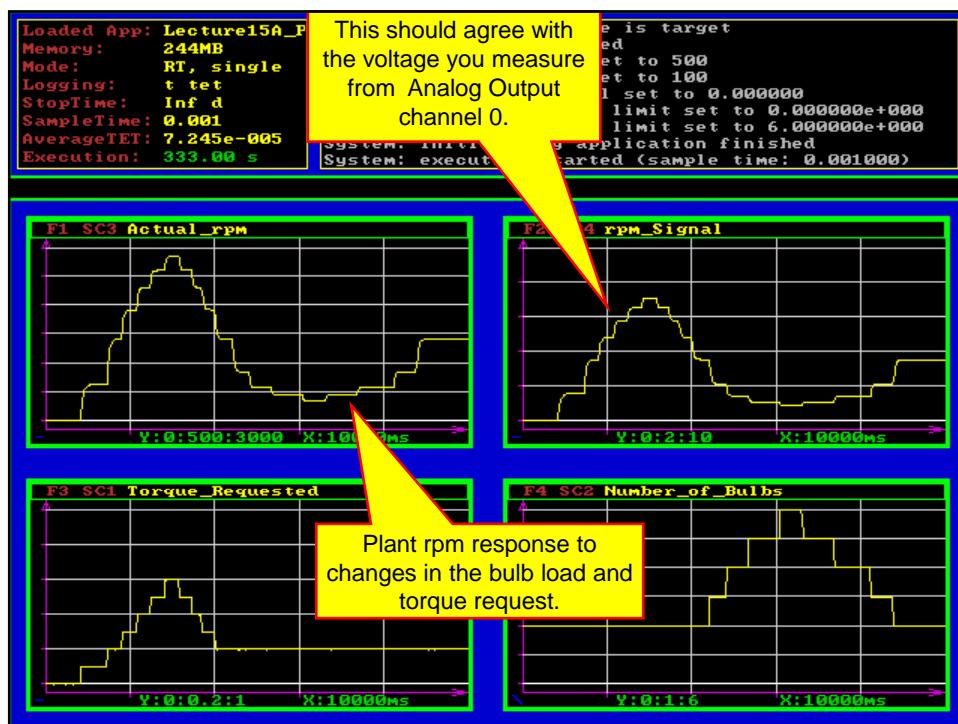
68

- As we change the input torque request, we see the motor RPM increase and decrease in response to the changing torque request and bulb load.
- Verify that the RPM signal corresponds to the analog output voltage of channel Analog Output 0.

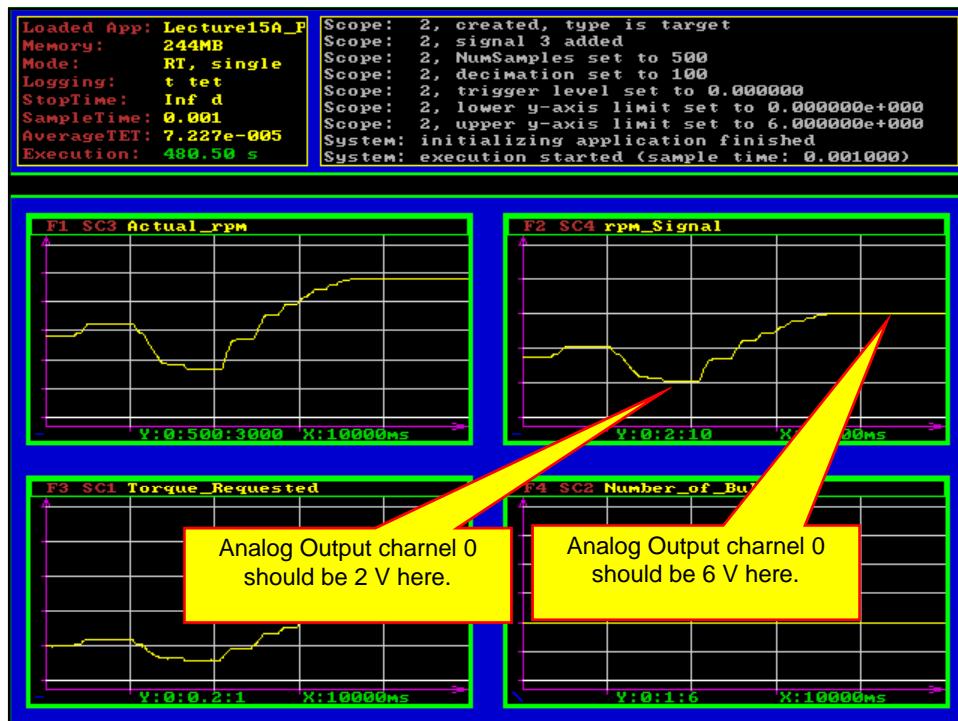




Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



Plant rpm response to changes in the bulb load and torque request.





71

## Lecture 15A – Demo 1

### Plant Test

- From previous simulations, you have a good idea of how the plant should respond to changes in the torque request and bulb load.
- Test your plant for various inputs and verify that it responds correctly.
- If your plant passes all of the above tests, we can now connect the plant to the controller and test our system using a complete HIL setup.

Demo\_\_\_\_\_



The MathWorks



freescale<sup>®</sup>  
semiconductor



72

## HIL Simulations

### Part 4: Interfacing the Controller to the Plant



The MathWorks



freescale<sup>®</sup>  
semiconductor





73

## HIL Simulation

- We are now ready to connect our controller to the plant, but we require one more modification.
- The torque output of our controller is a 20 kHz, 0 to 5 V PWM signal, and this is the signal expected by our motor controller. (The duty cycle will go between 0 to 100% corresponding to 0 to max torque request.)
- The Plant expects an analog voltage from 0 to 1 volt corresponding to 0 to max torque request.
- (In future model improvements, we might design our plant to use a PWM input rather than an analog voltage input. – However, this would require that the plant use a timer, which is not a facility available on our Speedgoat hardware.)



74

## HIL Simulation

- The difference between the two signals requires that we create a PWM to analog voltage converter circuit.
- We will use an inexpensive low-pass filter with a voltage divider.
- We will choose the pole of the filter to be much less than 20 kHz so that it filters out the 20 kHz PWM frequency, but high enough so that the delay of the filter is small compared to the dynamic response of the plant.
- Keep in mind that this filter will not be part of the system when we control the physical plant.

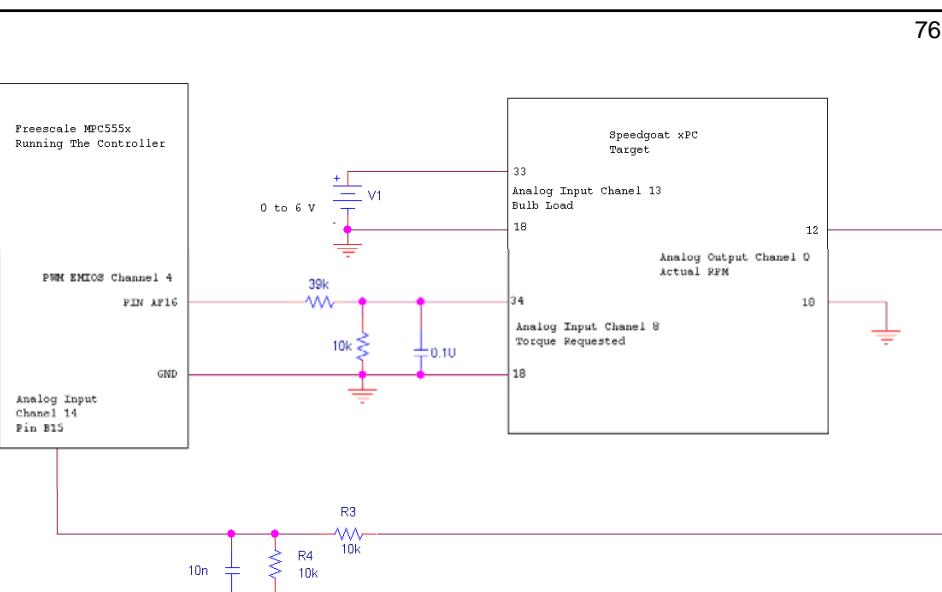


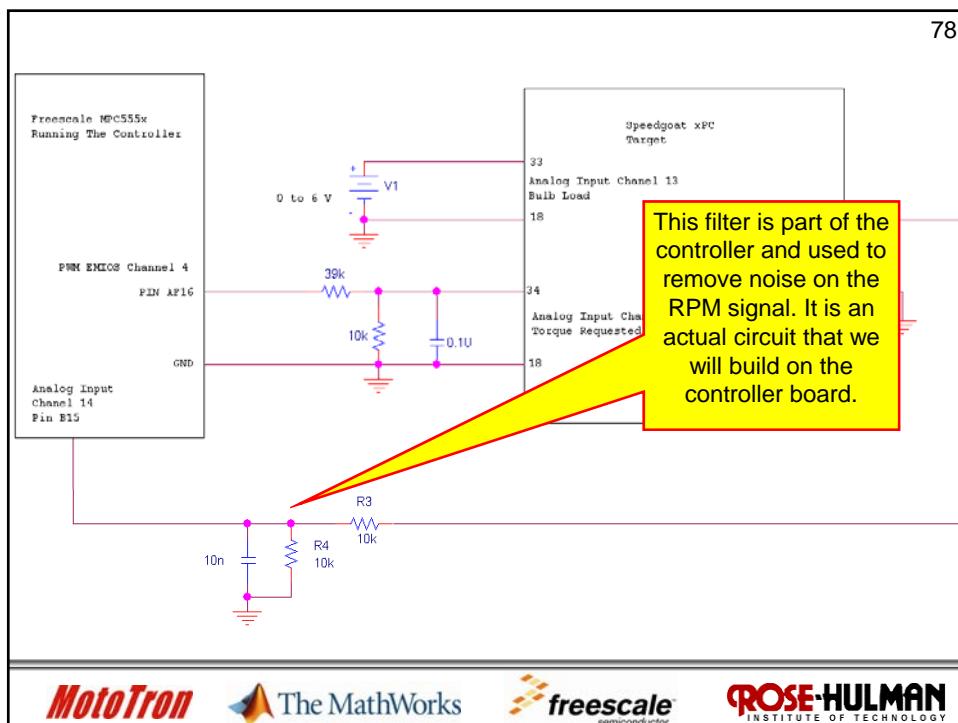
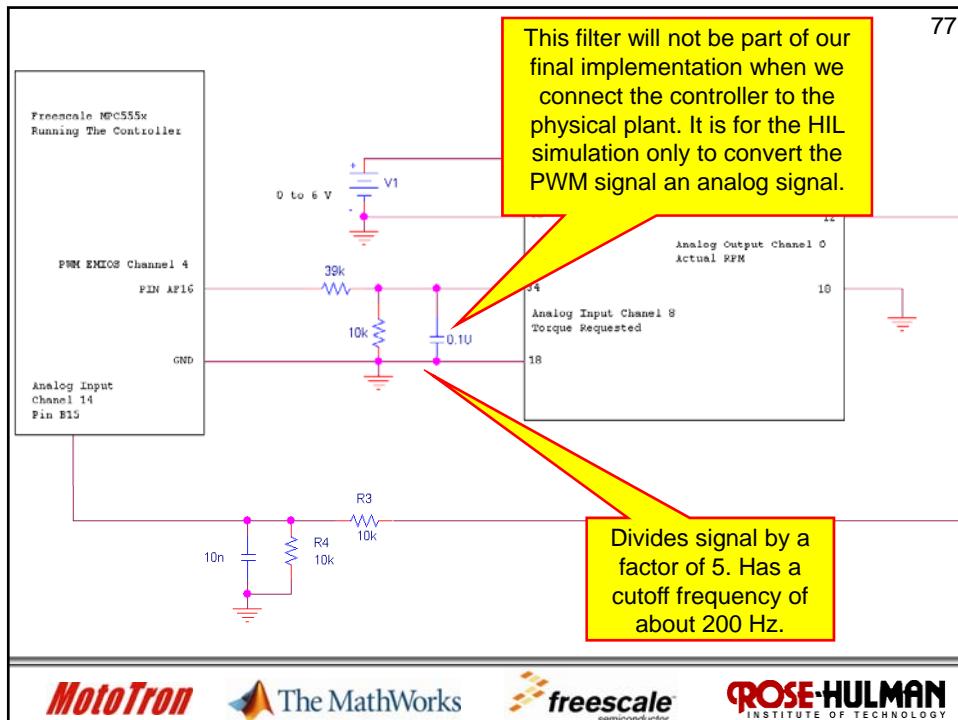


## HIL Simulation

75

- Also note that this filter adds delay to the HIL simulation that is not in the physical system nor was it in our MIL and SIL simulations.
- It is possible that we may see an instability in the HIL simulation due to this added delay that is not present in the physical realization nor our previous simulations.
- We will connect the MPC555x controller target to the Speedgoat Plant target with the connections shown on the next slide.







79

## Low-Pass Filter

- Note that the filter we added to convert the PWM waveform to an analog signal has a cutoff frequency of 200 Hz.
- This frequency was chosen because it is a factor of 100 below the PWM frequency of 20 kHz.
- Since we have a first order filter that attenuates the signal at 20 dB per decade above the cutoff frequency, the 20 kHz signal will be reduced by a factor of 100 (40 dB) by the filter.
- This is good, but not great.
- For a 5 V, 20 kHz PWM waveform input the filter, the filter will have 50 mV of ripple at 20 kHz at the output.
- This ripple will cause a ripple on the torque output of the motor.



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Low-Pass Filter

80

- The cutoff frequency of 200 Hz is close to the bandwidth of the motor-generator system.
- This means that the delay added by the filter will be significant and may make the system unstable.
- We will keep this in mind as we do our testing.



The MathWorks

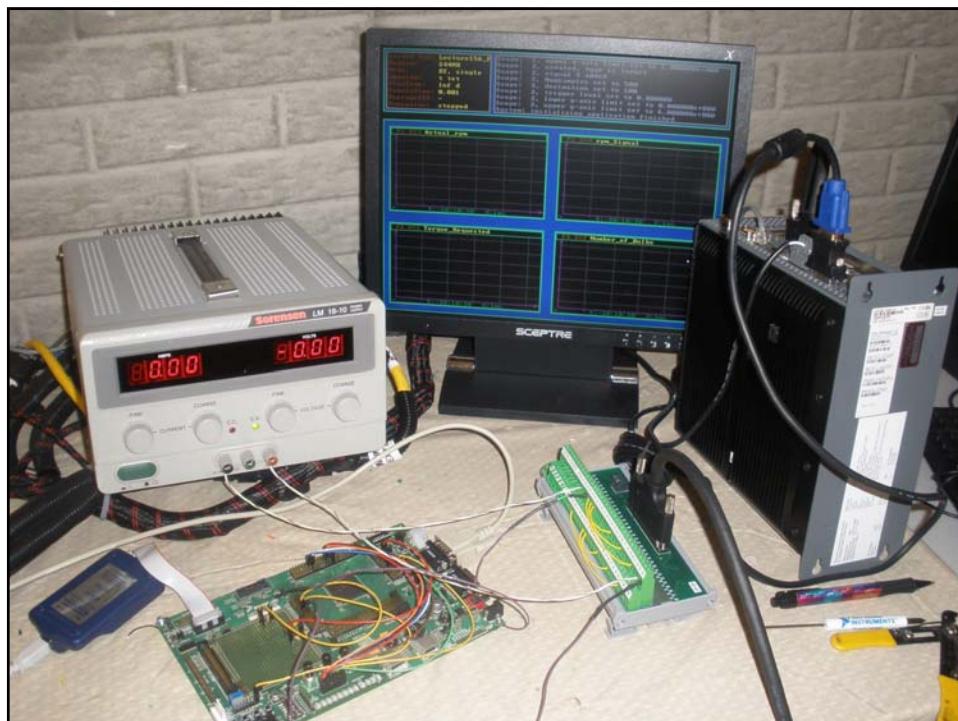
freescale<sup>®</sup>  
semiconductor



81

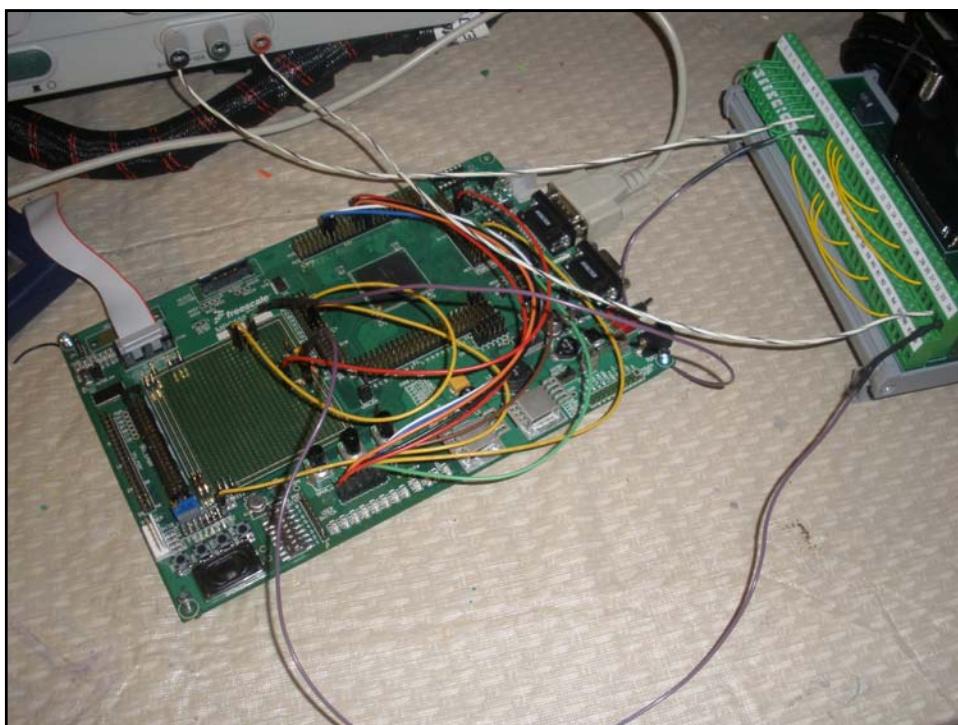
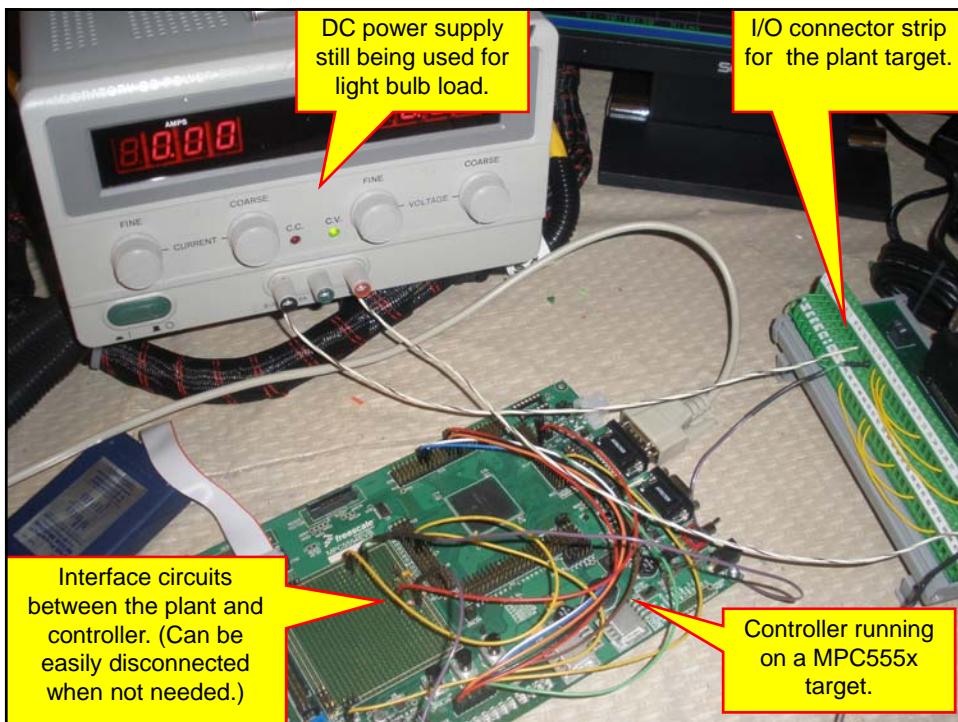
## HIL Setup

- My setup is shown on the next two slides.
- Note that the circuits to interface between the controller and plant been hardwired on to the prototyping area of the MPC555xEWB to keep the implementation clean.
- You may want to use a prototyping board to construct the circuits needed for this lab so that you can change the circuits and filter constants easily.



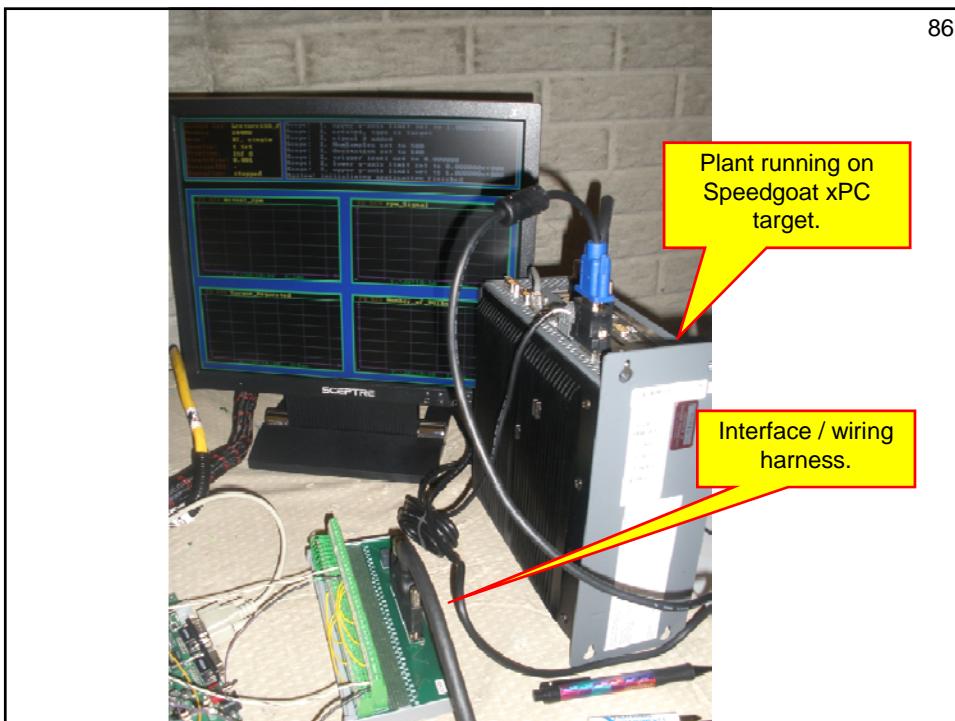
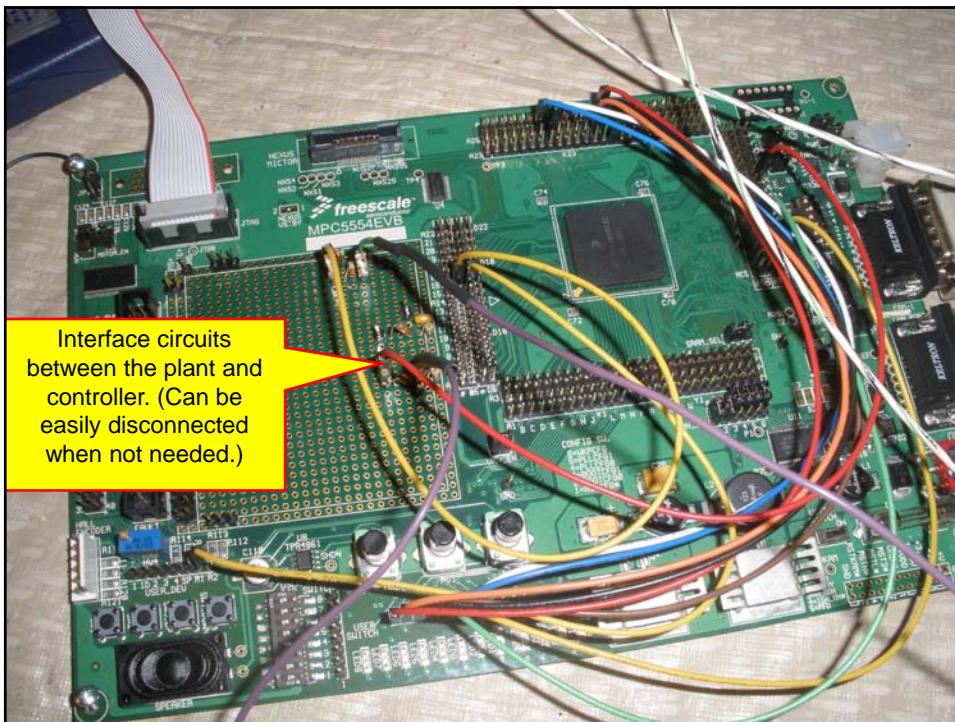


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





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





87

## HIL Simulations

### Part 5: HIL Simulation



The MathWorks



## HIL Simulations

88

- We are now ready to run our HIL simulation.
- Use the Simulink Plant model or the xPC Explorer to run your plant model on the Speedgoat Target.
- Download and run your controller model on the MPC555x.
- Monitor important signals on the MPC555x using FreeMASTER or the RHIT ASCII debug blocks.



The MathWorks





89

## HIL Simulations

- The next slide shows my plant performance for the following controller settings:
  - Proportional Gain = 1
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 20 kHz
- The number of bulbs is set to 2.

**MotoTron**

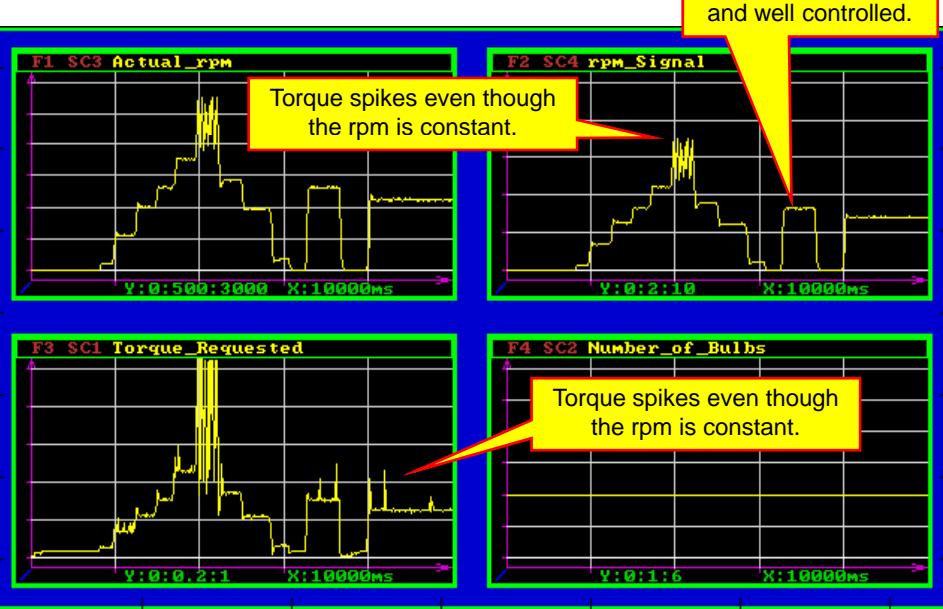
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

- Step response in rpm speed request. Bulb load held constant at 2 bulbs.

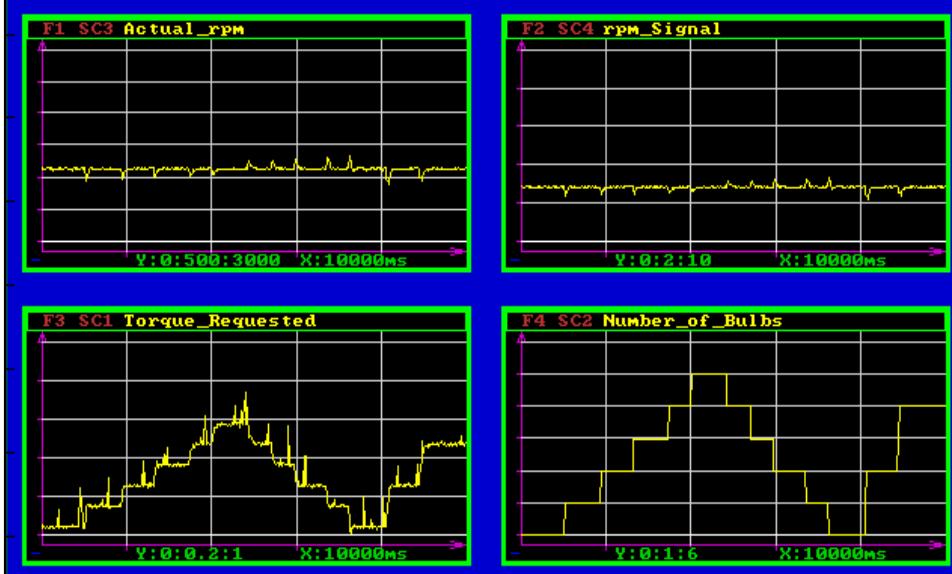
90





91

- Constant speed request. Bulb load step response.



92

## Lecture 15A – Demo 2

- Demo of HIL simulation.
  - Proportional Gain = 1
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 20 kHz
- Speed Step Response
- Bulb Step Response

Demo \_\_\_\_\_





## HIL Simulations

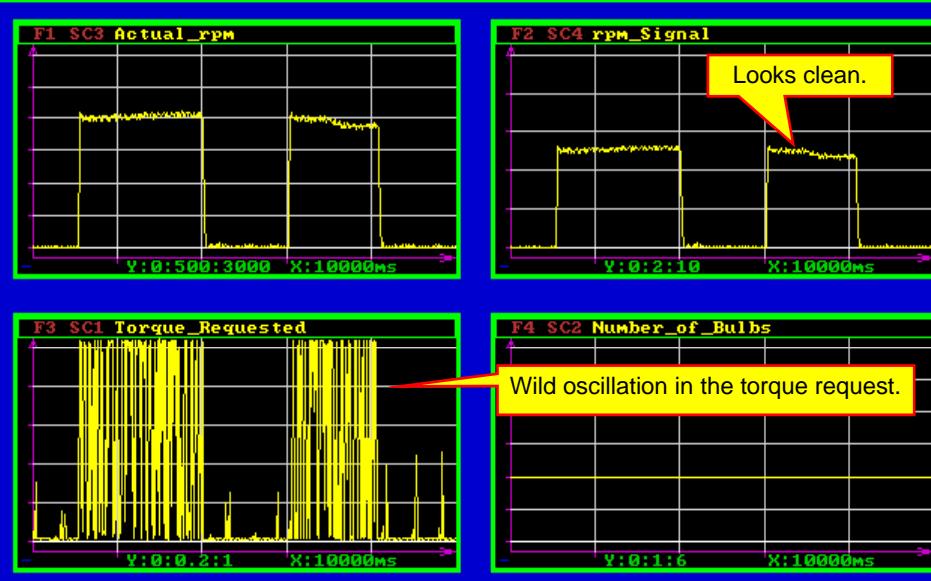
93

- We notice that there is a lot of ringing in the system response.
- From previous simulations, we found that a proportional gain of 100 and an integral gain of 10 worked well.
- Change these gains, rebuild the controller, download the controller model to the MPC555x and then redo the previous tests.



- Step response in rpm speed request. Bulb load held constant at 2 bulbs.

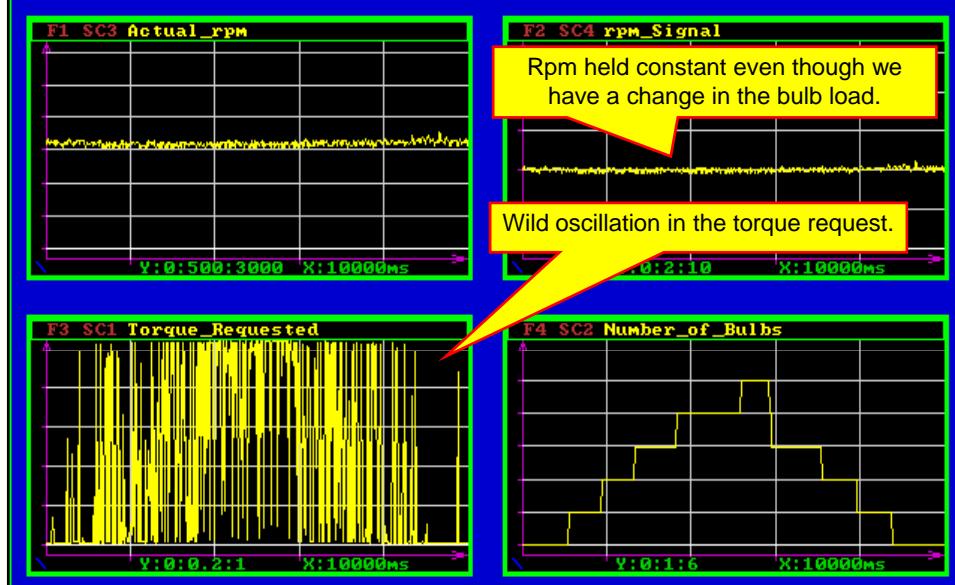
94





95

- Constant speed request. Bulb load step response.



96

## Lecture 15A – Demo 3

- Demo of HIL simulation.
  - Proportional Gain = 100
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 20 kHz
- Speed Step Response
- Bulb Step Response

Demo \_\_\_\_\_





97

## HIL Simulations

- The step response (RPM signal output) now looks clean, but we see large oscillations in the torque request signal.
- This could be for several reasons:
  - The filter we added for the PWM signal (which is not part of the physical system) added extra delay.
  - The interface / wiring harness added noise and delay that was not modeled earlier.
  - The PWM nature of the torque request signal is adding noise to the system. (Because we only attenuate the 20 kHz signal by a factor of 100.)
  - The gain is just too high, and caused the system to oscillate. (Review earlier simulations and see if the selected gains caused the system to be unstable.)

98

## HIL Simulations

- Note that we are not really sure that the wild oscillations on the torque request signals are actually there.
- The signal is a PWM signal that is a square wave that oscillates between 0 & 5 volts. We are trying to use the average of that signal by using a low-pass filter.
- The filter output contains 50 mV of 20 kHz ripple.
- This ripple may be what is causing the oscillations in the torque request.



The MathWorks



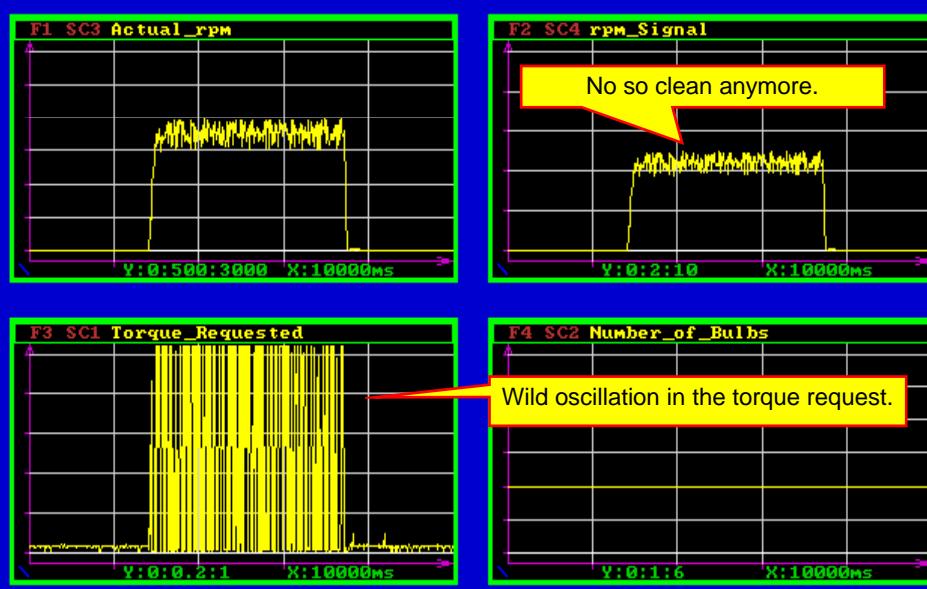


99

## HIL Simulations

- We will try different gains to improve the performance of the system while still having a stable system.
- Unfortunately, we cannot change the controller gains in real-time, so we will have to modify the gains, rebuild the model, and then download it to the MPC555x every time we make a change.
- Next, try a proportional gain of 10 and an integral gain of 10.

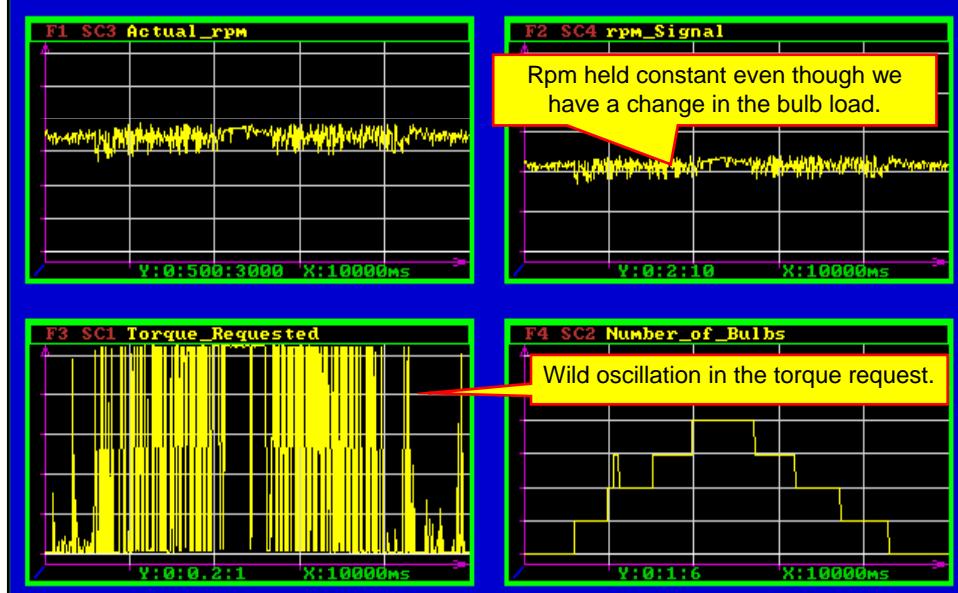
- Step response in rpm speed request. Bulb load held constant at 2 bulbs. <sup>100</sup>





101

- Constant speed request. Bulb load step response.



102

## Lecture 15A – Demo 4

- Demo of HIL simulation.
  - Proportional Gain = 10
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 20 kHz
- Speed Step Response
- Bulb Step Response

Demo \_\_\_\_\_





## HIL Simulations

103

- The step response of the rpm signal looks clean, but we still see large spikes on the torque request signal.
- If we compare the response of the HIL simulations to our previous simulations for the specified gains and the simulations give different results, then we have learned something:
  - Our interface may have introduced something that was not previously modeled. (We need to solve this.)
  - The controller running on the actual target with a discrete time step is causing the problem. (We need to solve this.)
- The HIL simulations have shown us something new and we need to make adjustments or resolve the issues before we deploy our controller on the physical plant.

## HIL Simulations

104

- It is also possible that the plant running on a discrete target is causing the problem.
- This not an issue in the real system because the plant will be replaced by the actual physical system.
- We do need to test this possibility and make changes if it is the cause. (By say, decreasing the time step of the plant – if possible.)
- Since we ran the entire system in real-time on an xPC target with the same time step as used here and we did not see the oscillations, the plant time step is probably not the issue.





105

## HIL Simulations

- It is also possible that the filter we added for the PWM signal is causing a problem.
- This filter is not part of the physical system or controller, and we should investigate to see if this is causing the problem.
  - Noise:
    - Measure the output of the filter and make sure that 20 kHz ripple is not present on the signal.
  - Delay:
    - Increase the PWM frequency (to say 200 kHz).
    - Increase the pole of the filter (to say 2 kHz).



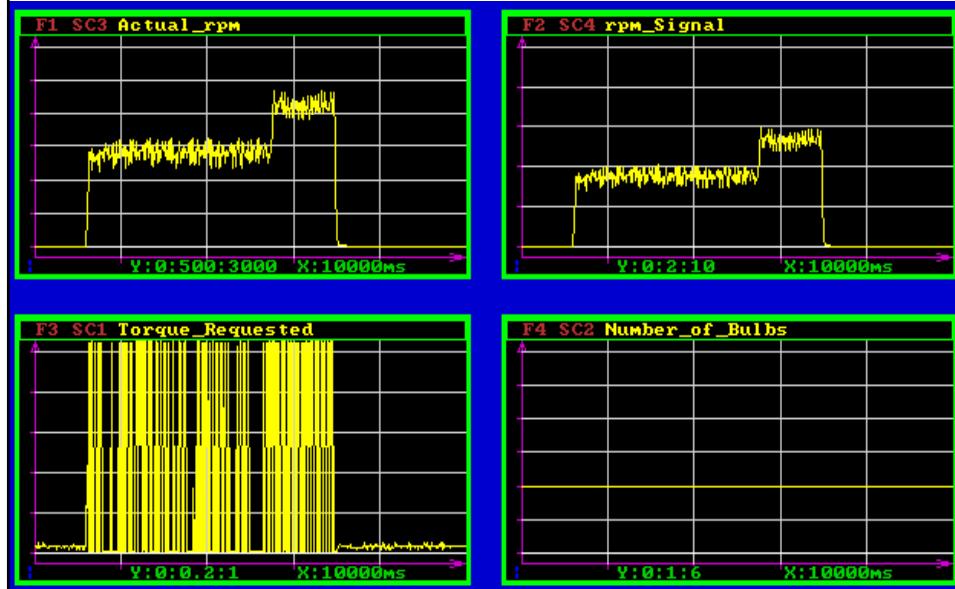
106

## HIL Simulations

- The next slides show the system response for proportional and integral gains of 10 (same as the previous simulation).
- The PWM frequency was increased to 200 kHz and the filter cutoff frequency was changed to 2 kHz (a much faster filter).
  - This filter will have the same 50 mV of ripple as the last filter since the PWM frequency is 100 times higher than the filter's cutoff frequency.
  - Since the filter's cutoff frequency has been increased to 2 kHz, (a factor of 10 higher) we have decreased the amount of delay added by the filter by a factor of 10.
- This experiment will test if the filter delay was the cause of the problem.



- Step response in rpm speed request. Bulb load held constant at 2 bulbs. <sup>107</sup>



## Lecture 15A – Demo 5 <sup>108</sup>

- Demo of HIL simulation.
  - Proportional Gain = 10
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 200 kHz
  - Filter Cutoff frequency 2 kHz
- Speed Step Response
- Bulb Step Response

Demo\_\_\_\_\_



The MathWorks



ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



## HIL Simulations

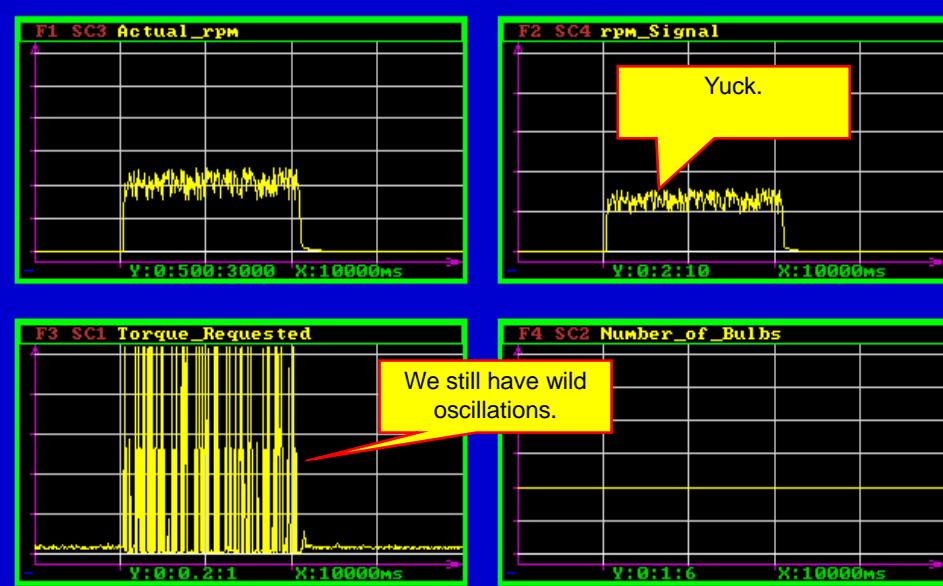
109

- The wild oscillations are still present in the system.
- Next, we will keep the PWM frequency at 200 kHz and then change the filter cutoff frequency back to 200 Hz.
- This will increase the filter delay, but reduce the 200 kHz ripple at the filter by a factor of 10. (The 200 kHz PWM frequency will now be attenuated by a factor of 1000.)



- Step response in rpm speed request. Bulb load held constant at 2 bulbs.

110





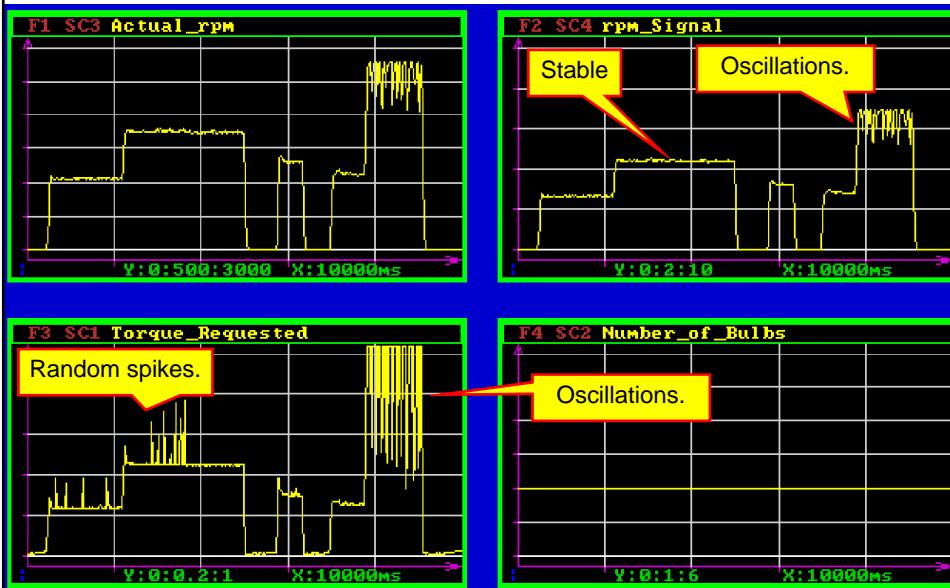
## HIL Simulations

111

- Reducing the delay and reducing the ripple do not seem to fix the problem.
- We could test the hypothesis further that the oscillation is due to the filter and PWM ripple further by eliminating the filter completely and using a plant target that can measure the pulse-width directly.
- Our hardware does not allow this test.
- Next, we will see if reducing the gain fixes the problem. We will keep the filter cutoff frequency at 200 Hz and the PWM frequency at 200 kHz.

- Step response in rpm speed request. Bulb load held constant at 2 bulbs.

112





113

## Lecture 15A – Demo 6

- Demo of HIL simulation.
  - Proportional Gain = 1
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 200 kHz
  - Filter Cutoff frequency 200 Hz
- Speed Step Response
- Bulb Step Response

Demo \_\_\_\_\_



114

## Deja Vu

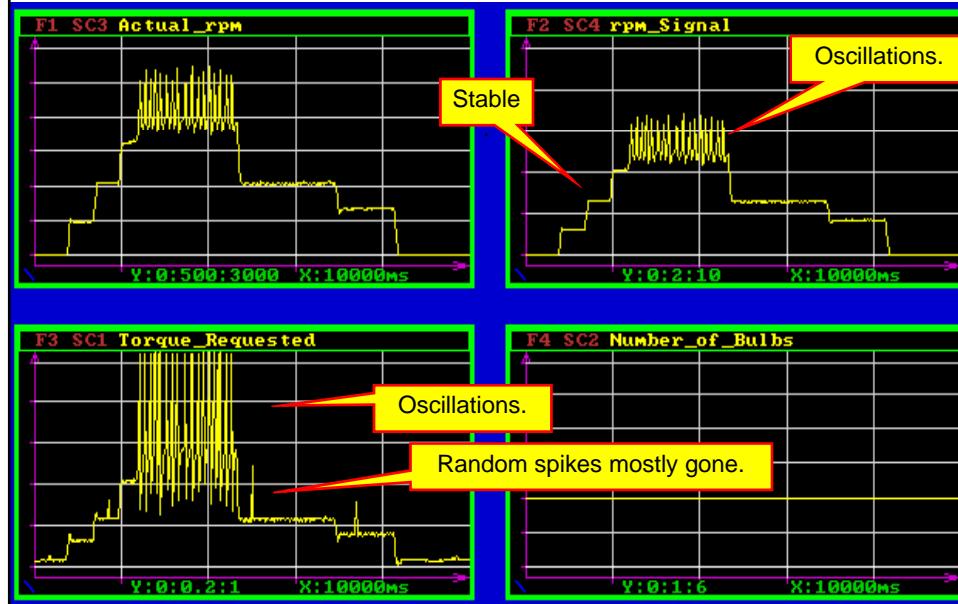
- I know that I remember this system working better before.
- As a last step, we will change the filter back to having a cutoff frequency of 2 kHz. We will keep everything else the same.
- (PWM at 200 kHz, proportional gain = 1, integral gain = 10).
- This will reduce the delay of the added filter.





- Step response in rpm speed request. Bulb load held constant at 2 bulbs.

115



- Constant speed request. Bulb load step response.

116





117

## Lecture 15A – Demo 7

- Demo of HIL simulation.
  - Proportional Gain = 1
  - Integral Gain = 10
  - Fixed Step size = 1 ms
  - PWM frequency is set to 200 kHz
  - Filter Cutoff frequency 2 kHz
- Speed Step Response
- Bulb Step Response

Demo \_\_\_\_\_



118

## HIL Simulations - Conclusions

- The HIL simulations give us some new results.
- First, it appears that we need a proportional gain of 1 to make the system stable. This is much lower than the gain we determined in earlier systems.
- We are not sure if the added filter is the cause of the reduced gain and bad system behavior.
- We should really eliminate the added filter as it is not part of the physical plant and may be causing issues in the simulation.
- The poor performance of the system in the HIL simulations suggest that we need for work in improving the system's performance before we deploy the controller on the physical plant. Torque spikes can damage equipment.
- (In this course, we do not have time to investigate the cause of the observed problems.)





## Controller Notes

119

- Remember that we only increased the PWM frequency to 200 kHz to try to eliminate the effects of the filter and PWM ripple.
- When we deploy the controller on the physical plant in the next lecture, we need to use a frequency of **20 kHz** as the plant hardware is designed for a PWM frequency of **20 kHz**.



The MathWorks





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

# Introduction to Model-Based Systems Design

## Lecture 16: Motor Controller Deployment



The MathWorks



freescale  
semiconductor



## System Deployment

2

- We have placed our control algorithm and controller hardware through several levels of testing:
  - Model-in-the-Loop (MIL) where we tested the controller as a Simulink model connected to a Simulink model of the plant running in the Windows environment.
  - Real-Time simulations where we ran both the controller and plant in real time on a single xPC target.
  - Software-in-the-Loop (SIL) where we compiled the controller into the same code that would run on the hardware target and ran the model with a fixed time step.
  - Hardware-in-the-Loop (HIL) simulations where we deployed the controller onto the target that will be used in the final implementation and tested that controller by connecting it to another real-time-computer running the plant model.



The MathWorks



freescale  
semiconductor





3

## System Deployment

- By going through all of these levels of simulation, we gain confidence that when we hook the controller to the physical plant:
  - The system will behave as expected.
  - The system will operate in a safe manner.
  - We will not break anything mechanically.
  - We will not let the smoke out of any electrical components.
- The only reason the system will fail is that our models of the plant are not accurate:
  - This is always possible and we should always expect that there are things we can do to improve the plant model.
  - One thing we did not do in this course is make exhaustive measurements of the plant components, and then update the plant model to reflect the measurements.

4

## System Deployment

- With the HIL simulations, we deployed the control method on the target hardware that will be used in the final realization.
- We identified all of the inputs and outputs for the plant and controller.
- We hooked the controller inputs and outputs to a real-time computer running a model of the physical plant, not the physical plant itself.
- The last step is to take that same controller and connect it to the physical plant.



The MathWorks

freescale<sup>®</sup>  
semiconductorROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



5

## Connections

- The controller has the following connections:
  - A PWM output for the motor torque. This is connected directly to the PWM input of the plant and the motor is PWM controlled. We do not need the low-pass filter that causes us problems in the HIL simulation.
  - An analog input to measure the motor rpm. This is connected to the rpm output of the plant. Note that a 2:1 voltage divider is included in the plant, but the filter capacitor is not included. You will have to add the capacitor.
  - You will need to run one ground connection between the plant and the controller
- A wiring diagram is shown on the next slide.

**MotoTron**

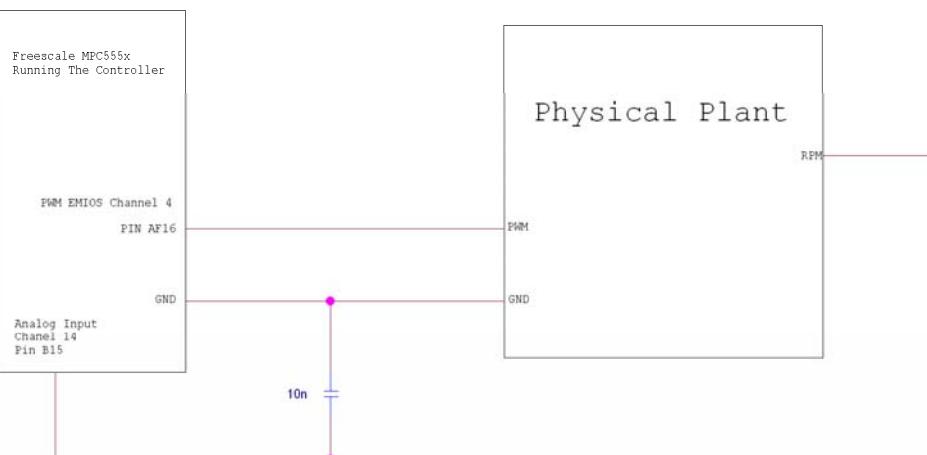
The MathWorks

freescale<sup>®</sup>  
semiconductor

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY

6

## Wiring Diagram



**MotoTron**

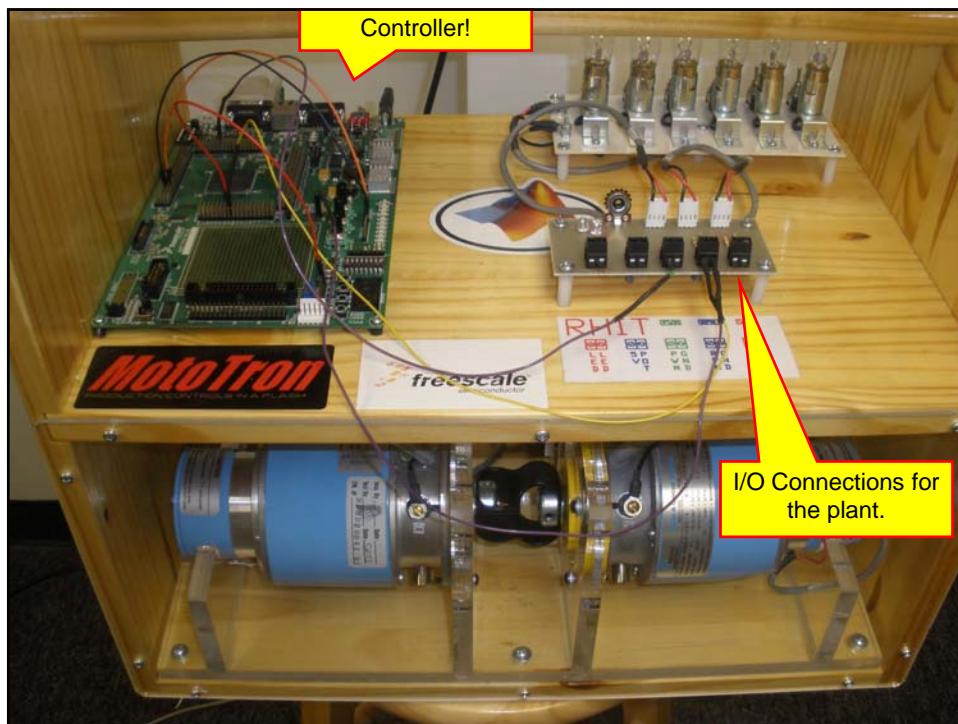
The MathWorks

freescale<sup>®</sup>  
semiconductor

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY

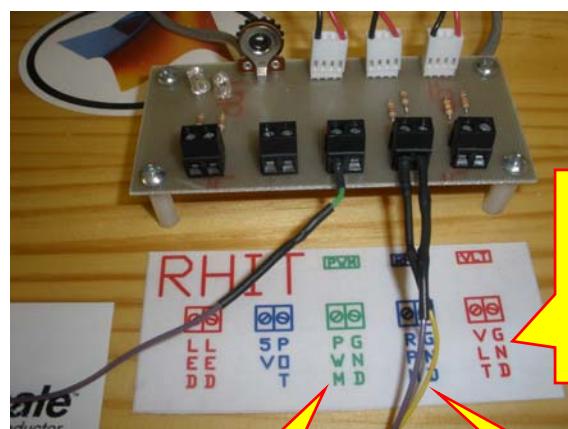


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



## Plant I/O Connections

8



PWM input signal.

rpm output signal.





## Lecture 16 Demo 1

Demo \_\_\_\_\_

9

- Once you have made all of the connections:
  - Plug in your plant. (Wear safety glasses!)
  - Verify that the motor speed follows the desired speed specified by the POT on the MPC555x demo board.
  - Display the desired speed and actual speed using either the FreeMASTER tool or the RHIT Debug Blocks.

Model-Based System Design Laboratory  
 HIL System Demonstration

Desired rpm: 1445  
 Measured rpm: 1444  
 PWM Duty Cycle (%): 48

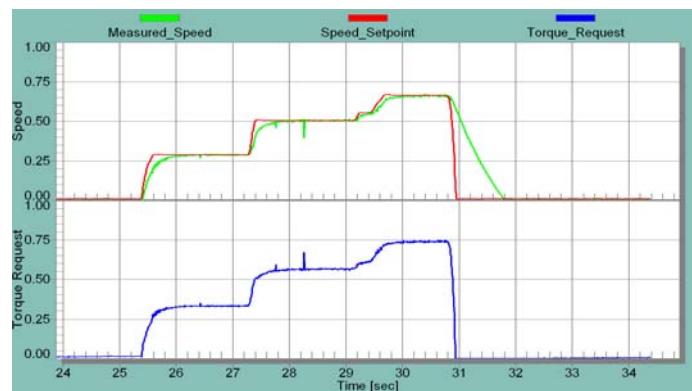


## Lecture 16 Demo 2

Demo \_\_\_\_\_

10

- Show the step response of your system using the FreeMASTER tool. The screen capture below show the step response with no load:



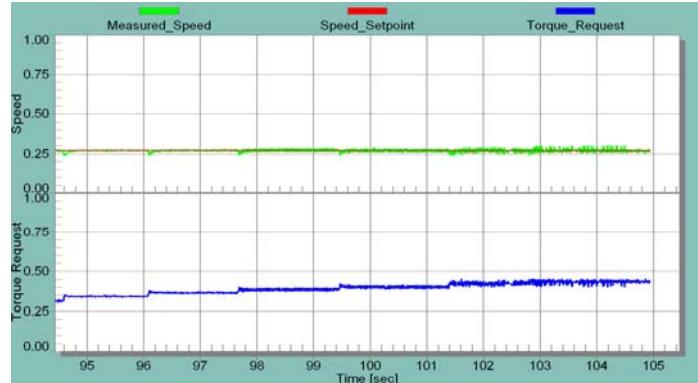


## Lecture 16 Demo 3

Demo \_\_\_\_\_

11

- For constant speed request, show the system response as the bulb load is increased from no bulbs to 6 bulbs:



**MotoTron**

The MathWorks

freescale™  
semiconductor

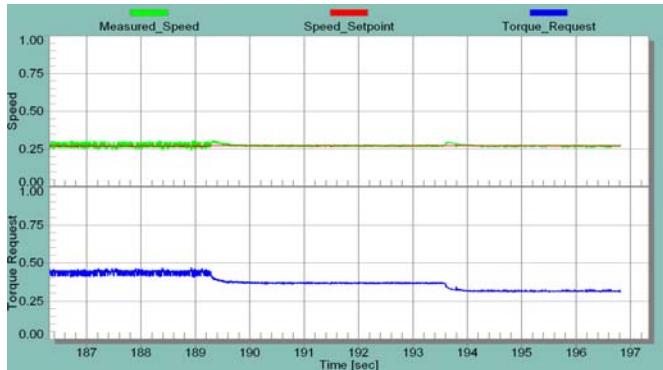
ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY

## Lecture 16 Demo 4

Demo \_\_\_\_\_

12

- For constant speed request, show the system response as the bulb load is decreased from 6 bulbs to 2 bulbs, and then from 2 bulbs to no load: (I only have 4 fingers. If you have more fingers, you can go from 6 bulbs to no bulbs in one step.)



**MotoTron**

The MathWorks

freescale™  
semiconductor

ROSE-HULMAN  
INSTITUTE OF TECHNOLOGY



13

## System Verification

- Now that we have a working system, we need to test the system and compare the results to our simulations.
- If measured and simulated results do not agree, we will need to determine the discrepancy and update our plant models if necessary.
- We will start with steady state testing.



14

## System Tests

- We will be measuring the rpm on the next few tests.
- We can use a volt meter and measure the encoder output to determine the system rpm.
- Remember that the nameplate spec for the encoder is 2.5 V per 1000 rpm.





15

## System Tests

- Although these are steady state tests, you will learn a great deal about the system transient response as you turn on an off light bulbs.
- You will be able to see and hear overshoot in the system rpm as you turn on and off light bulbs.



16

## Bulb Load Test, rpm = 1200 rpm

- Set Integral Gain = 0. With No load, set the rpm to 1200. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the rpm for various gains and loads.)

P_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1	1200						
10	1200						
100	1200						





17

## Bulb Load Test, rpm = 1800 rpm

- Set Integral Gain = 0. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the rpm for various gains and loads.)

P_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1	1800						
10	1800						
100	1800						



18

## Bulb Load Test, rpm = 2400 rpm

- Set Integral Gain = 0. With No load, set the rpm to 2400. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the rpm for various gains and loads.)

P_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1	2400						
10	2400						
100	2400						





19

## Integral Gain Tests

- For the next tests, we will keep the rpm constant at the 1800 rpm.
- We will then specify a fixed proportional gain, and observe the system response for varying values of integral gain.
- Note that these are steady state responses.



20

## Integral Gain Test, rpm = 1800

- Set Proportional Gain = 1. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the steady state rpm for various integral gains and loads.)

I_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1							
10							
100							





21

## Integral Gain Test, rpm = 1800

- Set Proportional Gain = 10. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the steady state rpm for various integral gains and loads.)

I_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1							
10							
100							



22

## Integral Gain Test, rpm = 1800

- Set Proportional Gain = 100. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the steady state rpm for various integral gains and loads.)

I_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1							
10							
100							





23

## Fixed Step Size Test

- For the next tests, we will keep the rpm constant at the 1800 rpm.
- Choose values for the integral and proportional gains that give the system, the best response.
- We will keep these values constant in the next set of tests and see the effect of changing the fixed step size.



The MathWorks

freescale<sup>®</sup>  
semiconductor

## Controller Step Size Test

24

- Set Proportional Gain = \_\_\_\_\_. Set Integral Gain = \_\_\_\_\_. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the steady state rpm for various values of the system step size.)

Step Size (sec)	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
0.1							
0.01							
0.001							
0.00001							



The MathWorks

freescale<sup>®</sup>  
semiconductor



25

## SIL Tests

- We would like to compare the measured results to the simulated results.
- Repeat all of the above tests using the model from Lecture 10A Model 3.



26

## (SIL) Bulb Load Test, rpm = 1200 rpm

- Set Integral Gain = 0. With No load, set the rpm to 1200. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the rpm for various gains and loads.)

P_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1	1200						
10	1200						
100	1200						





27

## (SIL) Bulb Load Test, rpm = 1800 rpm

- Set Integral Gain = 0. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the rpm for various gains and loads.)

P_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1	1800						
10	1800						
100	1800						



28

## (SIL) Bulb Load Test, rpm = 2400 rpm

- Set Integral Gain = 0. With No load, set the rpm to 2400. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the rpm for various gains and loads.)

P_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1	2400						
10	2400						
100	2400						





29

## (SIL)Integral Gain Test, rpm = 1800

- Set Proportional Gain = 1. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the steady state rpm for various integral gains and loads.)

I_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1							
10							
100							



30

## (SIL) Integral Gain Test, rpm = 1800

- Set Proportional Gain = 10. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs. (Measure the steady state rpm for various integral gains and loads.)

I_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1							
10							
100							





31

## (SIL) Integral Gain Test, rpm = 1800

- Set Proportional Gain = 100. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.  
 (Measure the steady state rpm for various integral gains and loads.)

I_Gain	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
1							
10							
100							



32

## (SIL) Controller Step Size Test

- Set Proportional Gain = \_\_\_\_\_. Set Integral Gain = \_\_\_\_\_.  
 With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.  
 (Measure the steady state rpm for various values of the system step size. Use the same values of gain as we did for the physical system in slide 25.)

Spec Size (sec)	No Bulbs	1 Bulb	2 Bulbs	3 Bulbs	4 Bulbs	5 Bulbs	6 Bulbs
0.1							
0.01							
0.001							
0.00001							





33

## Comparison of SIL and Measured Results

- Use the tables below to compare the measured results to the simulated results.
- No new measurements are needed.
- Just place both measured and simulated results in the same table.
- Turn in the tables below.



## Lecture 16 Comparison 1200 rpm

34

- Compare SIL results to measured (act) results.
- Set Integral Gain = 0. With No load, set the rpm to 1200. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

P_Gain	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
1	1200	1200												
10	1200	1200												
100	1200	1200												





35

## Lecture 16 Comparison 1800 rpm

- Compare SIL results to measured (act) results.
- Set Integral Gain = 0. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

P_Gain	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
1	1800	1800												
10	1800	1800												
100	1800	1800												



36

## Lecture 16 Comparison 2400 rpm

- Compare SIL results to measured (act) results.
- Set Integral Gain = 0. With No load, set the rpm to 2400. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

P_Gain	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
1	2400	2400												
10	2400	2400												
100	2400	2400												





37

## Lecture 16 Comparison 1800 rpm

- Compare SIL results to measured (act) results.
- Set Proportional Gain = 1. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

I_Gain	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
1	1800	1800												
10	1800	1800												
100	1800	1800												



38

## Lecture 16 Comparison 1800 rpm

- Compare SIL results to measured (act) results.
- Set Proportional Gain = 10. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

I_Gain	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
1	1800	1800												
10	1800	1800												
100	1800	1800												





39

## Lecture 16 Comparison 1800 rpm

- Compare SIL results to measured (act) results.
- Set Proportional Gain = 100. With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

I_Gain	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
1	1800	1800												
10	1800	1800												
100	1800	1800												



The MathWorks



40

## Lecture 16 Comparison Step Size Test

- Compare SIL results to measured (act) results.
- Set Proportional Gain = \_\_\_\_ (optimum value). Set integral Gain = \_\_\_\_ (optimum value). With No load, set the rpm to 1800. Observe how the rpm changes as the load changes from 0 bulbs to 6 bulbs.

Spec Size (sec)	No Bulbs		1 Bulb		2 Bulbs		3 Bulbs		4 Bulbs		5 Bulbs		6 Bulbs	
	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act	SIL	Act
0.1														
0.01														
0.001														
0.00001														



The MathWorks





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

## Questions?



The MathWorks





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



## Introduction to Model-Based Systems Design

### Lecture 17: Improved Bulb Model Intro to Design of Experiments



The MathWorks



freescale<sup>®</sup>  
semiconductor



## Model Problems - Motor

2

- When we first designed our plant, we understood that the models we created were very idealized:
  - The motor model uses a fictitious torque curve.
  - The motor is 100% efficient.
  - There is no current limit on the motor. (For us, this would be a limit on the DC power supply.)



The MathWorks



freescale<sup>®</sup>  
semiconductor





3

## Model Problems - Generator

- The generator has the same non-idealities as the motor, plus a few more that we need to consider:
- For our load (generator combined with the light bulbs), we assumed that the load torque was a linear function of the rpm.
- In reality, a bulb looks like a temperature dependent resistance. The temperature depends on the bulb voltage, which in turn depends on the generator speed.



4

## Generator Model

- The experience we have gained by using our motor-generator system has shown us the following about the generator:
  - The output voltage is a function of the rpm.
  - The output voltage drops as we draw more current from the generator.
- We would like to create a model that includes both of these effects.
- (After we do the bulb model.)





## Bulb Model

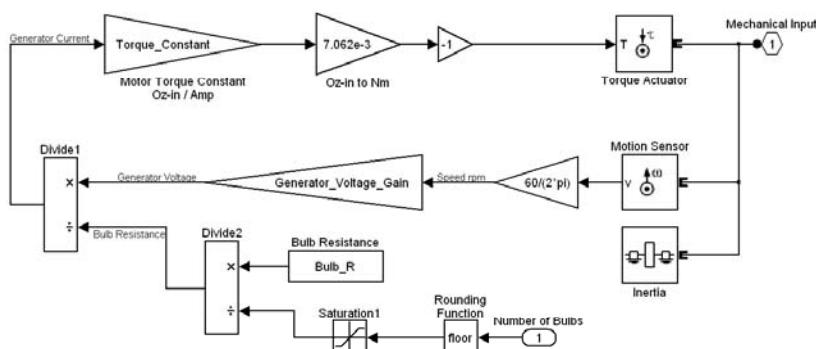
5

- The bulb looks like a temperature dependent resistor (with a large positive temperature coefficient).
  - As the temperature of the resistor goes up, its resistance goes up.(Or so we think.)
  - The power drawn by the bulb is the voltage squared divided by the bulb resistance.
  - The resistance is a function of the bulb power, since more bulb power heats up the bulb.
- We will create a model that includes all of these effects.

## Old Generator Load Model

6

- Just as a reminder, the old generator and bulb load is shown below:



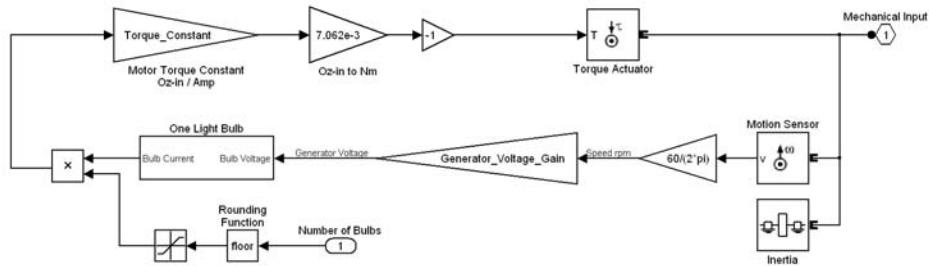
- In this model, the bulb resistance is constant and we calculate the generator torque by dividing the generator voltage by the bulb resistance.



7

## New Generator Load Model

- We will use the following model for the generator that includes all of the bulb effects:



- The generator will produce an output voltage based on the generator speed and load current.
- The bulb will draw a current that is dependent on the voltage across it (thus including temperature effects).
- The generator current is the number of bulbs times the current through one bulb.

8

## New Plant Model

- We can measure the light bulb I-V characteristic if we have a variable voltage power supply.
- We will use the Model-Based Calibration toolbox to:
  - Design a set of experiments.
  - Measure the bulb and generator characteristics with the specified set of experiments.
  - Create a model for the measured data and export it to Simulink.
- We could also use the measured data to:
  - Create a look up table.
  - Implement a Simulink function.

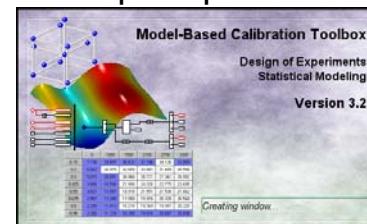




## Blub Model

9

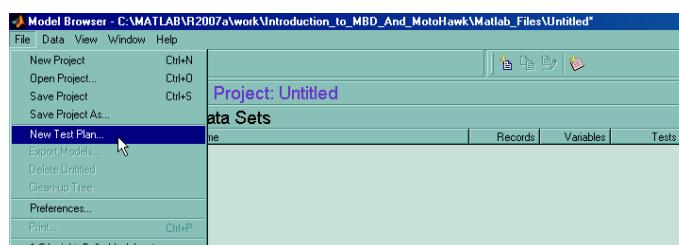
- We will first develop the model of the bulb.
- The input to the model is the bulb voltage.
- The output of the mode is the bulb current.
- We will use the Model-Based Calibration toolbox to set up a set of experiments for us to measure the bulb characteristic.
- To start the Model-Based Calibration, type **mbcmodel** at the Matlab command prompt:



## Model-Based Calibration Toolbox

10

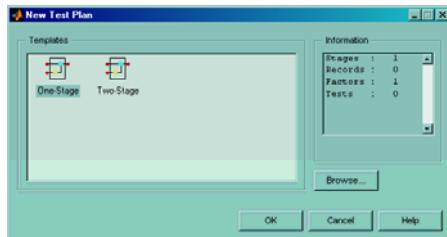
- All of the files we create will be saved in the current working directory.
- Select **File** and then **New Test Plan** from the Model Browser menus:



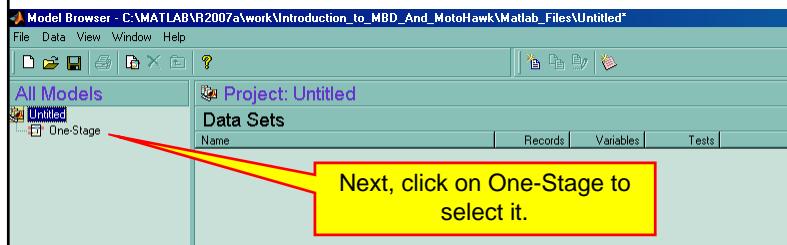


11

## Model-Based Calibration Toolbox



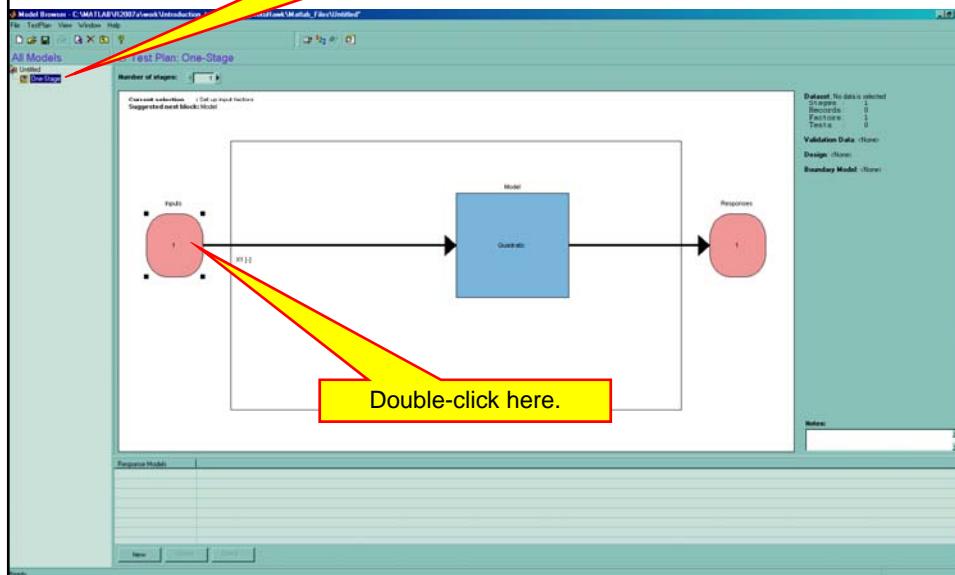
- Select a **One-Stage** model and click the **OK** button.



12

## Model-Based Calibration

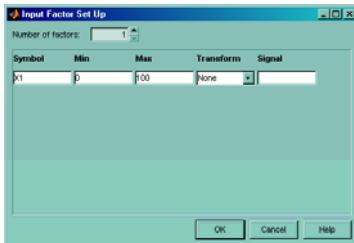
One-Stage selected.





13

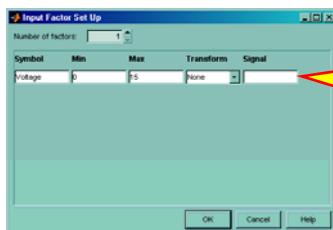
## Model-Based Calibration



- This box allows us to specify the inputs to our model and the ranges for those inputs.
- We only have a single input for our model.
- Fill in the dialog box as shown next:

14

## Model-Based Calibration



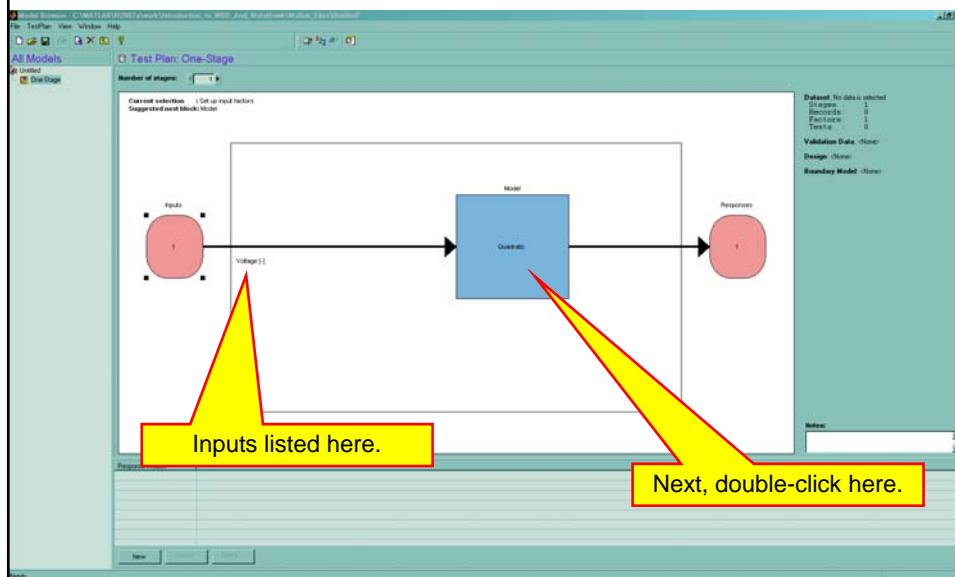
Symbol: Voltage  
 Min: 0  
 Max: 15  
 Transform: None

- We specified a range of 0 to 15 volts.
- Note that the Signal field is a 3-character abbreviation that you can use for the signal.
- Click the **OK** button when done. Note that the signal will be displayed on the block diagram.



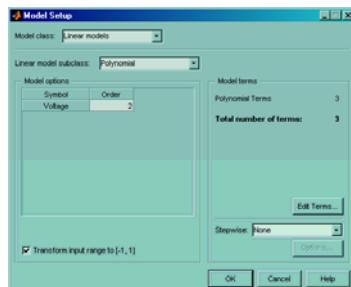
## Model-Based Calibration

15



## Model-Based Calibration

16



- This dialog box allows us to set up the type of curve we think the will best fit our model.
- A typical resistor has a straight line current-voltage (I-V) characteristic (a first order polynomial – a straight line).





## Model-Based Calibration

17

- We know that the resistance of the bulb will change with temperature.
- Thus, the bulb I-V characteristic will deviate from a straight line. (How much – we don't know – we've never done this before.)
- We will guess that it will deviate slightly from a straight line so a 2<sup>nd</sup> or 3<sup>rd</sup> order polynomial should be a good fit.
- We can change the model later if the model we pick does not fit the data very well.
- We will choose a 3<sup>rd</sup> order polynomial model:

**MotoTron**

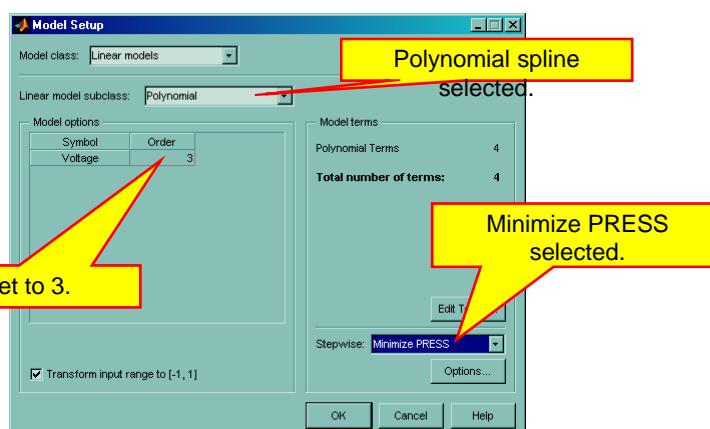
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Model-Based Calibration

18



- Click the **OK** button when done.

**MotoTron**

**The MathWorks**

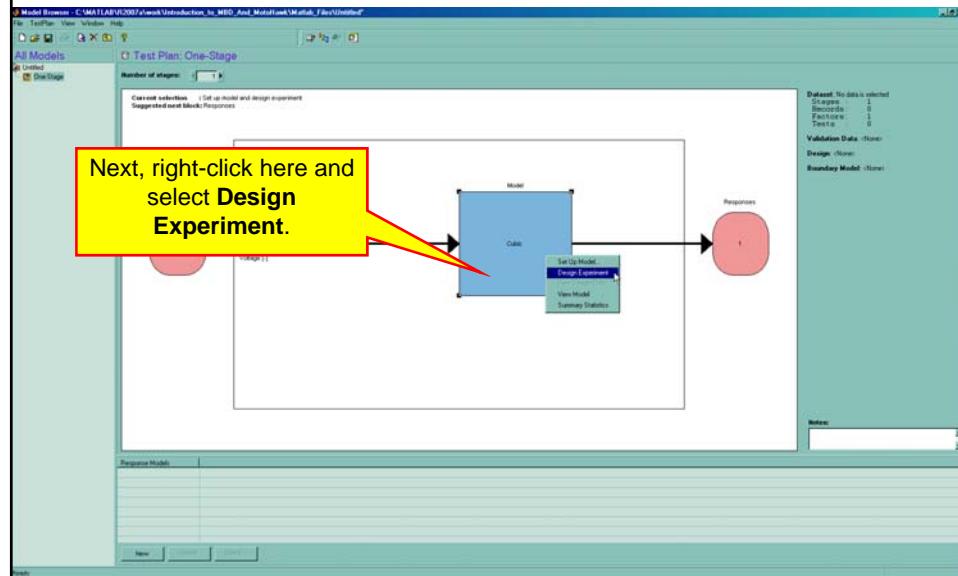
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



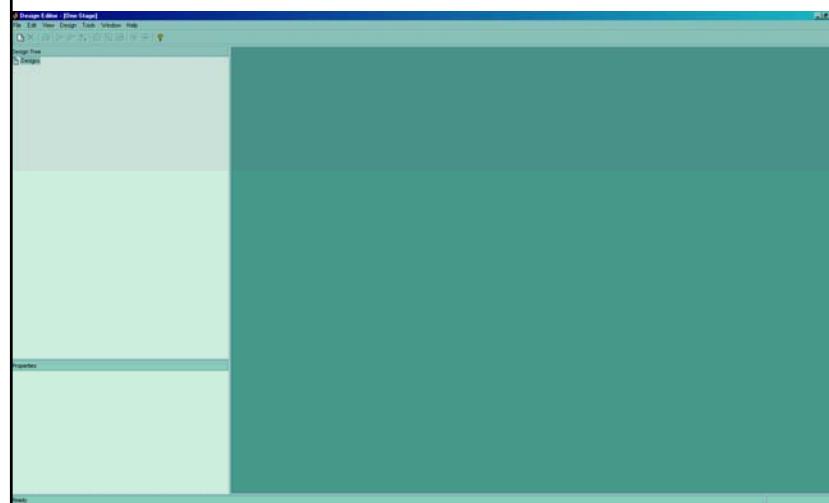
## Model-Based Calibration

19



## Model-Based Calibration

20

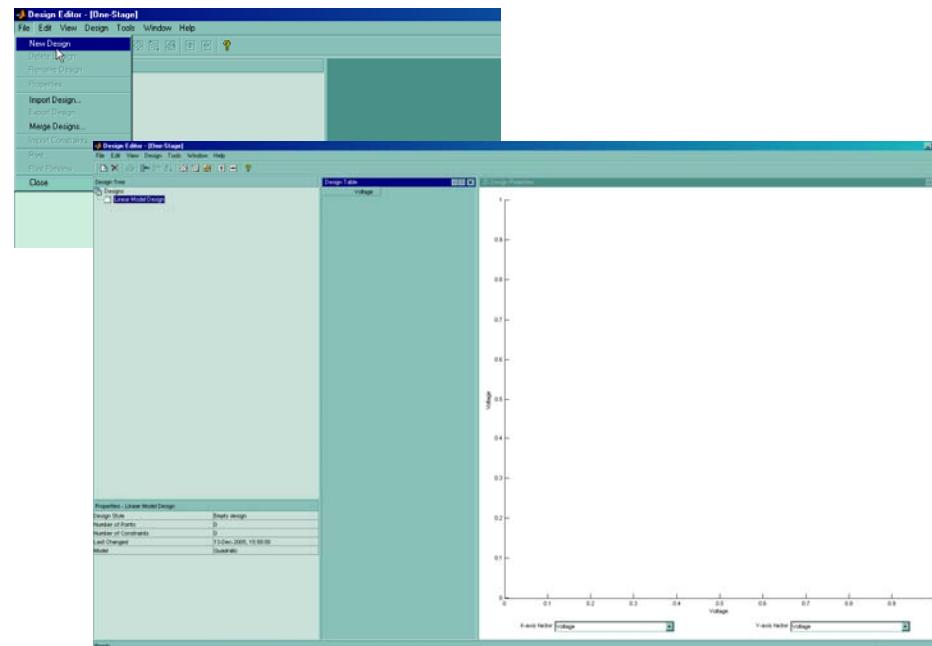


- Next, select **File** and then **New Design** from the Design Editor menus.



## Model-Based Calibration

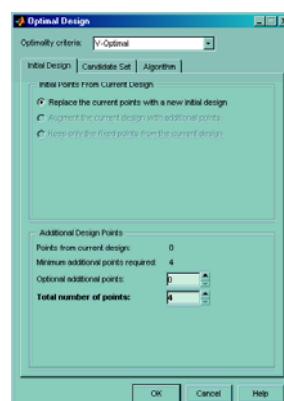
21



## Model-Based Calibration

22

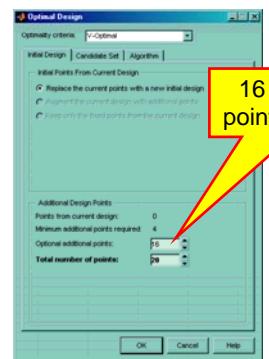
- Select **Design** and then **Optimal** from the **Design Editor** menus:
- For a 3<sup>rd</sup> order polynomial fit, we need a minimum of 4 points.
- Since we are not really sure a 3<sup>rd</sup> order polynomial will work, we will specify that we want a 16 additional points.
- More points will take more time to measure (and thus be more expensive for your company), but it will also help reduce the effects of measurement error on our model.





## Model-Based Calibration

23



16 additional points selected.

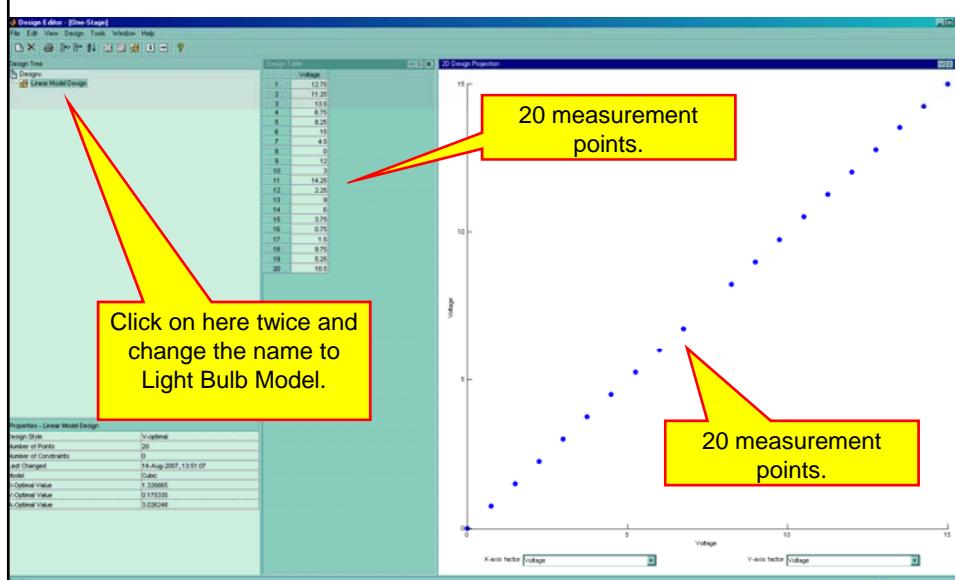
After clicking the **OK** button, the tool will calculate the voltage points for the model.



When complete, the selected points will be displayed. (Shown on next slide.)

## Model-Based Calibration

24





## Model-Based Calibration

25

	Voltage
1	12.75
2	11.25
3	13.5
4	6.75
5	8.25
6	15
7	4.5
8	0
9	12
10	3

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

## Model-Based Calibration

26

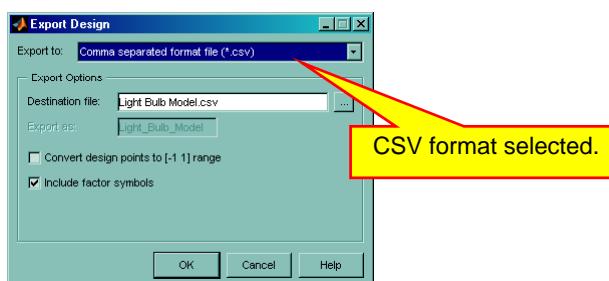
- The points for our experiment have been created.
- We will export them to an Excel spreadsheet so that we can easily collect data.
- Select **File** and then **Export Design** from the Design Editor menus:



27

## Model-Based Calibration

- Select a comma separated values format:



- After clicking the **OK** button, you can open the file with Excel.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

28

## Model-Based Calibration

A	B	C	D	E	F	G
1 Voltage						
2 12.75						
3 11.25						
4 13.5						
5 6.75						
6 8.25						
7 15						
8 4.5						
9 0						
10 12						
11 3						
12 14.25						
13 2.25						
14 9						
15 6						
16 3.75						
17 0.75						
18 1.5						
19 9.75						
20 5.25						
21 10.5						
22						

- We are now ready to run our experiment.
- For the specified voltages, we will measure the bulb current.
- Note that we do not have to use the exact same voltages, but we will try to come close.



## Measured Data

29

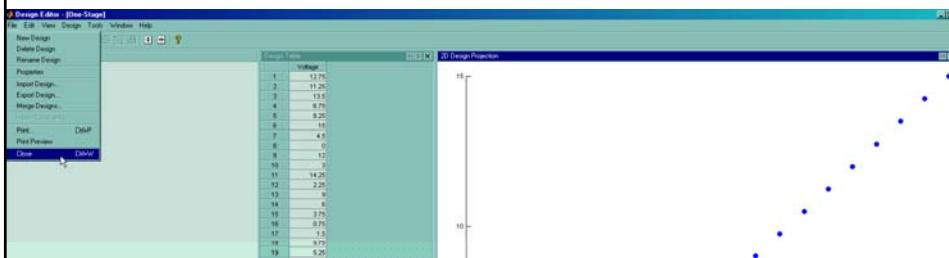
	A	B	C
1	Name	Voltage	Current
2	Units	Volts	Amps
3		14.99	1.01
4		14.27	0.98
5		13.5	0.96
6		12.74	0.93
7		12	0.91
8		11.25	0.87
9		10.49	0.84
10		9.74	0.8
11		8.93	0.78
12		8.28	0.74
13		6.76	0.67
14		5.96	0.63
15		4.52	0.55
16		3.76	0.5
17		2.98	0.45
18		2.31	0.41
19		1.48	0.34
20		0.75	0.27
21		0.28	0.18
22		0	0

- The data can be saved in a standard Excel format file.

## Model-Based Calibration

30

- Close the Design Editor by selecting **File** and then **Close** from the menus:



- Switch to the Model Browser window.



The MathWorks

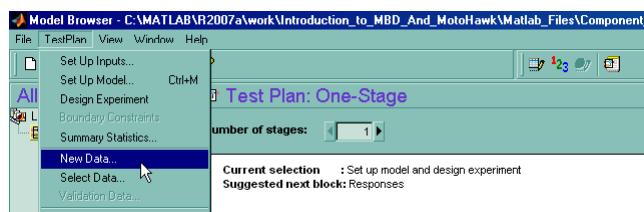




## Model-Based Calibration

31

- Before we continue, select **File** and then **Save Project** from the Model Browser Window to save our design.
- Save the file as Light\_Bulb\_Model.mat.
- Select **TestPlan** and then **New Data** from the menus:



32

The figure consists of three vertically stacked windows from the Data Selection Wizard and Data Import Wizard.

- Data Selection Wizard:** Shows the 'Data and Design' step. A red callout points to the 'Load new dataset' button with the text: "Click here to open a data file."
- Data Import Wizard:** Shows the 'File Type' step with 'Text' selected. A red callout points to the 'Next >' button with the text: "Click the Next button."
- Data Import Wizard:** Shows the 'Imported 20 records and 3 variables from' step. A red callout points to the 'Finish' button with the text: "Click the Finish button."

A yellow callout at the bottom left points to the table in the third window with the text: "Columns in our measured data."

Variable Name	Units	Min	Max	Mean	Std. Dev.
Voltage	Volts	0	15	7.25	5.02
Current	Amper	0	1.07	0.641	0.293



33

Select this option to match measured data to our model created earlier.

Click the Next button.

Voltage signal in our model matched to voltage signal in the measured data.

Click the Next button.

**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

34

Responses are the model outputs.

(1) Select Current.

(2) Click the Add button.

Current added as an output.

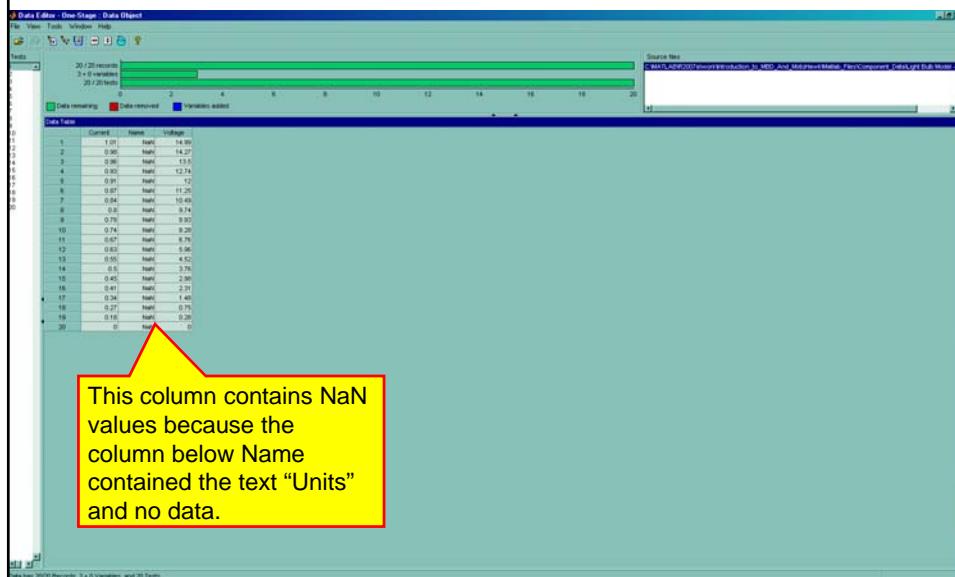
Click the Finish button.



## Data Editor

35

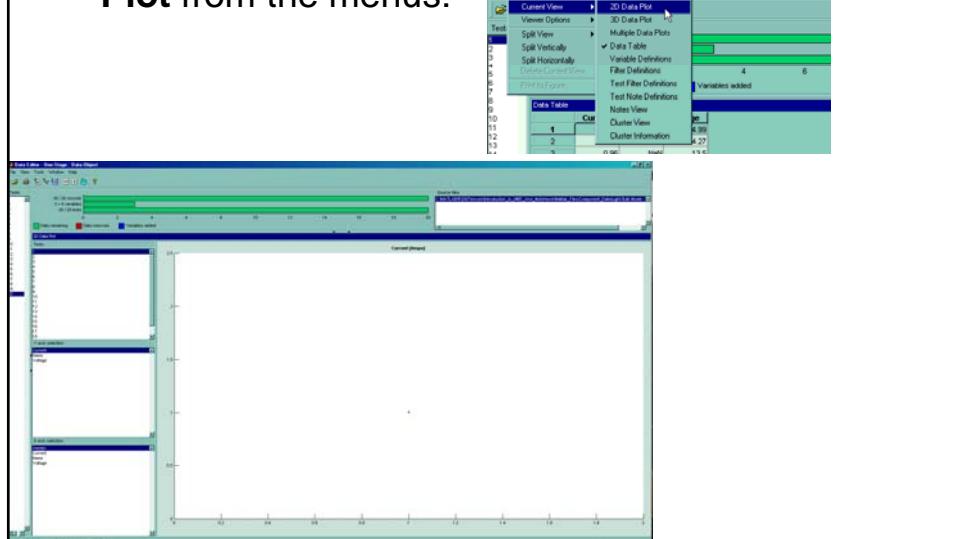
- The Data Editor opens and displays the data.



## Data Editor

36

- Select View, Current View, and then 2D Data Plot from the menus:

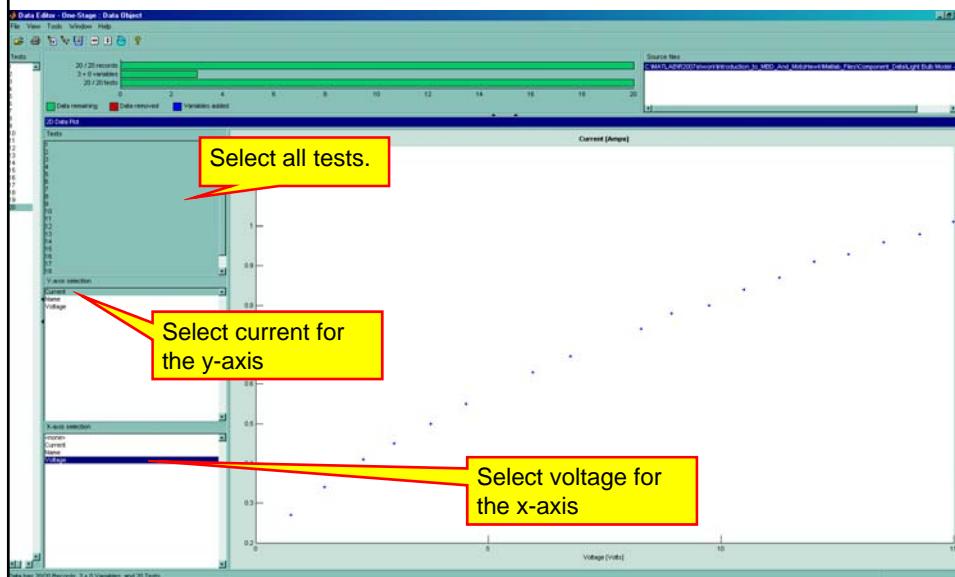




## Data Editor

37

- To view the data, make the selections below



## Data Editor

38

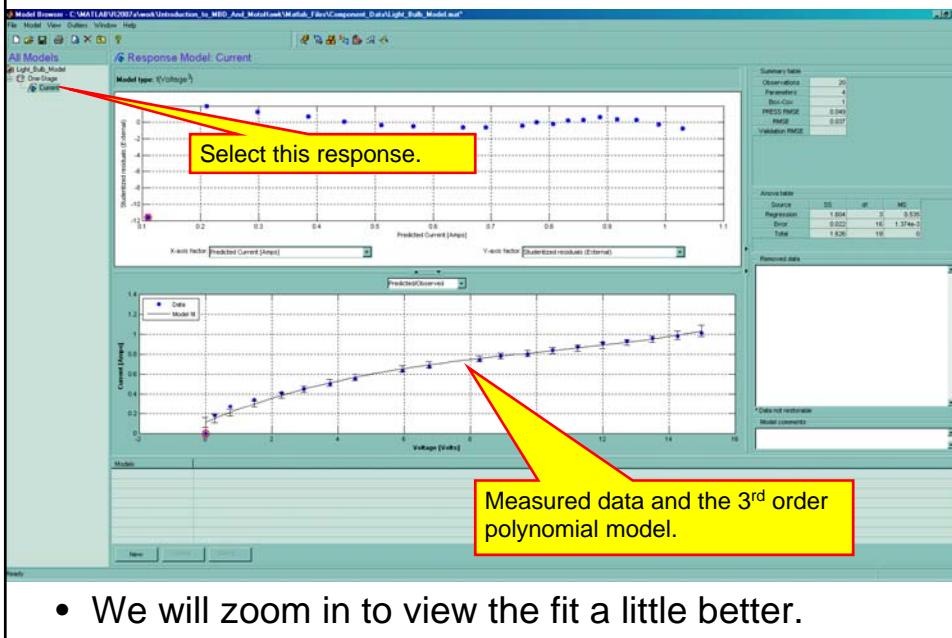
- We see that the current-voltage characteristic is indeed not a straight line.
- You can experiment with other views if you want to view the data differently. (For a one-dimensional plot, there is not that much else to look at.)
- Select **File** and then **Close** to close the Data Editor.
- When you return to the Model Browser, you will notice that there is now a response model for the Current.





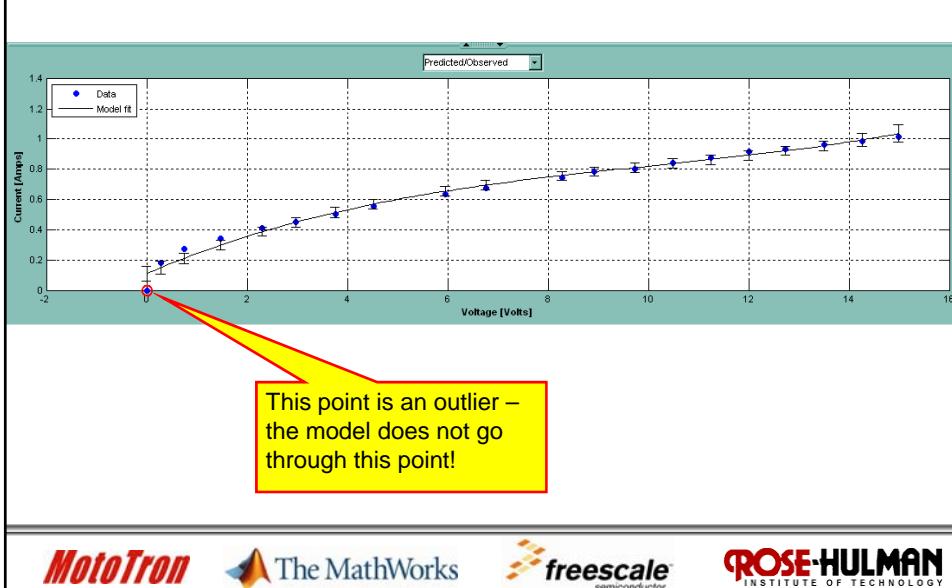
## Model Browser

39



## Model Browser

40

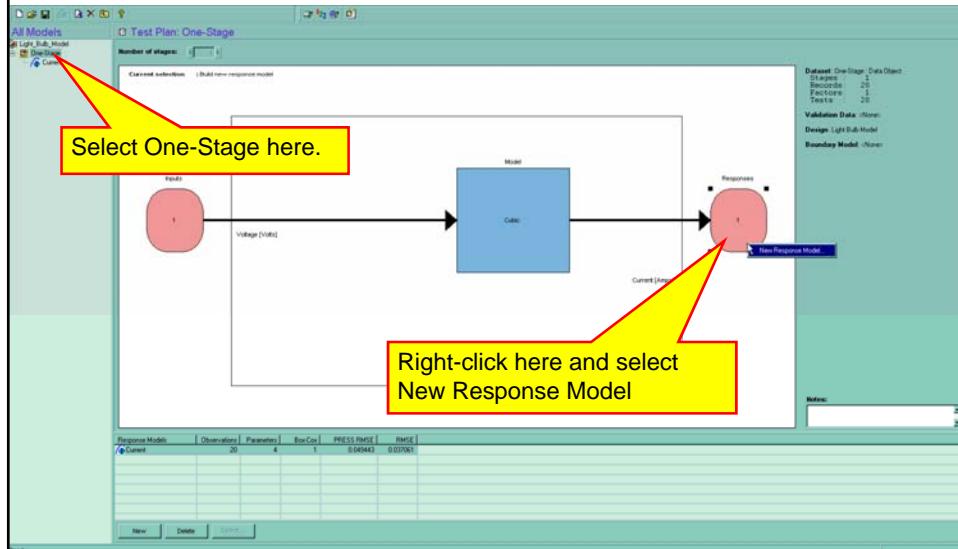




41

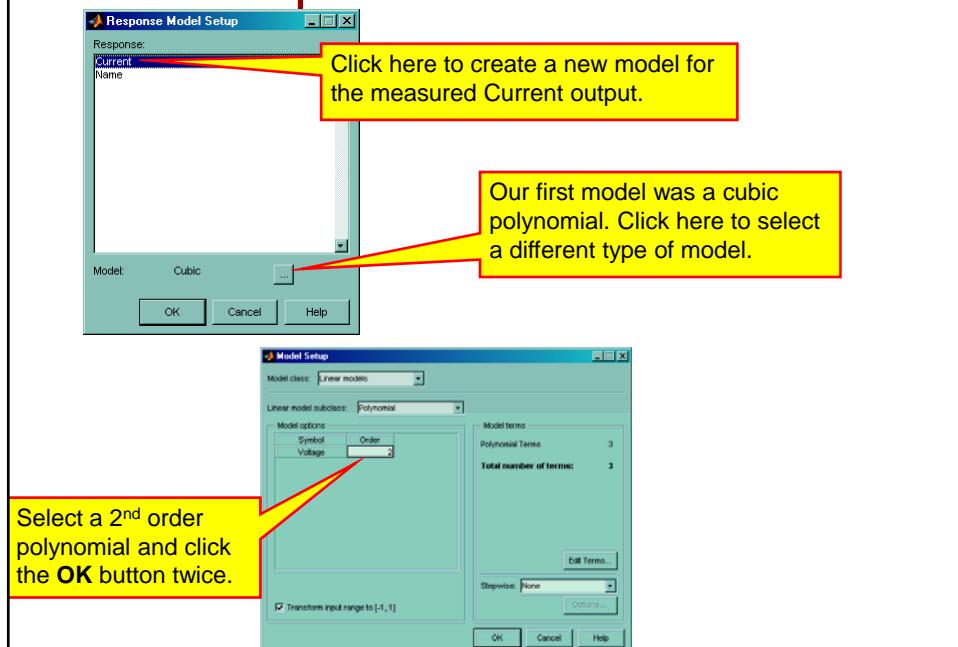
## Model Browser

- Suppose that we want to try a different fit to our data.



42

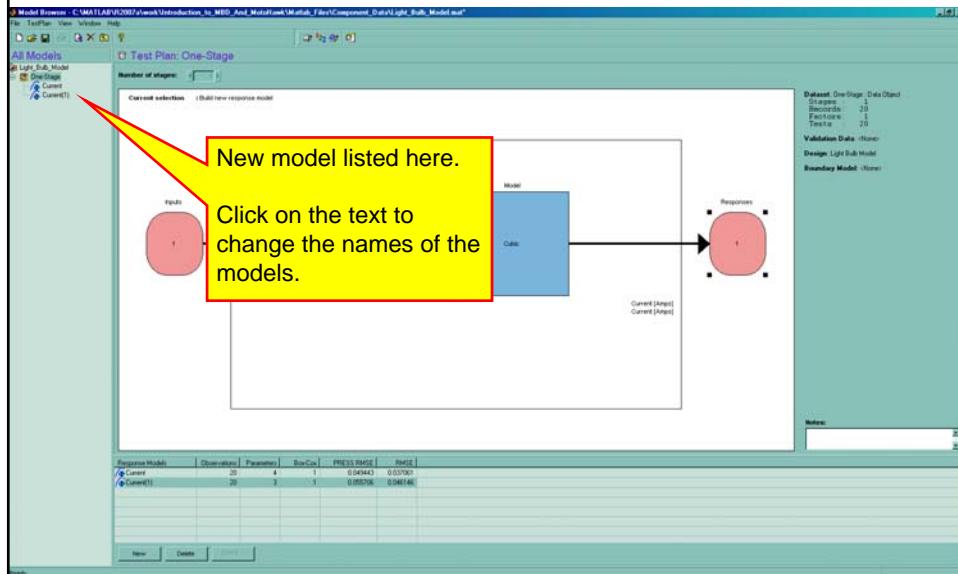
## New Response Model





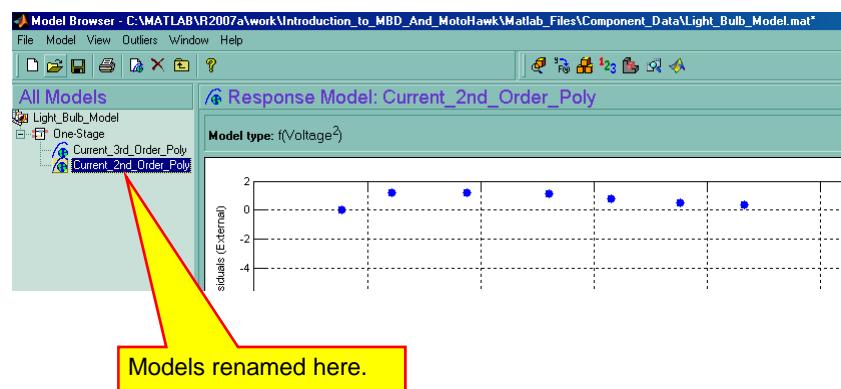
43

## New Response Model



44

## New Response Model



- You can display the plots for each fit by clicking on the name of the model.

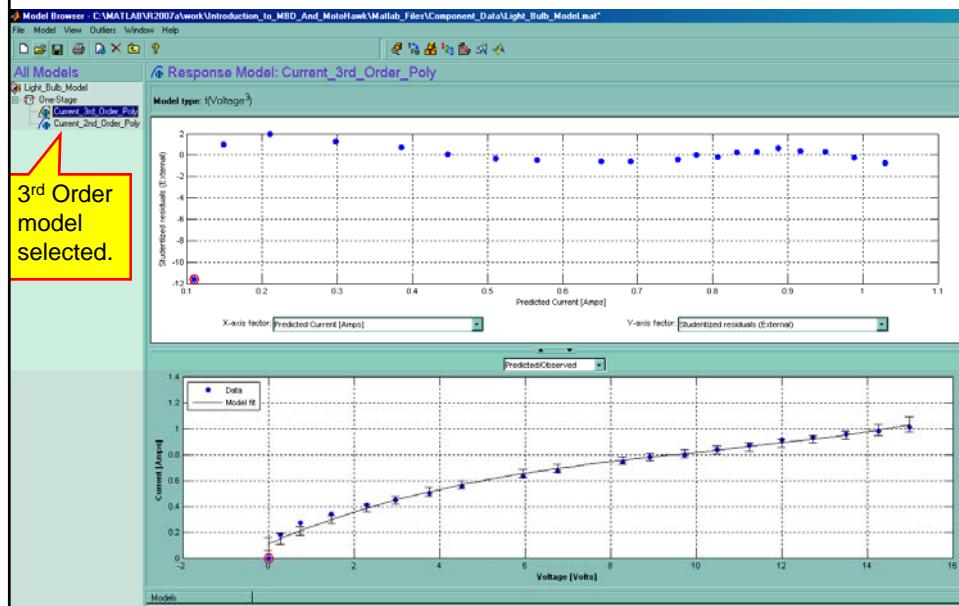




Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>

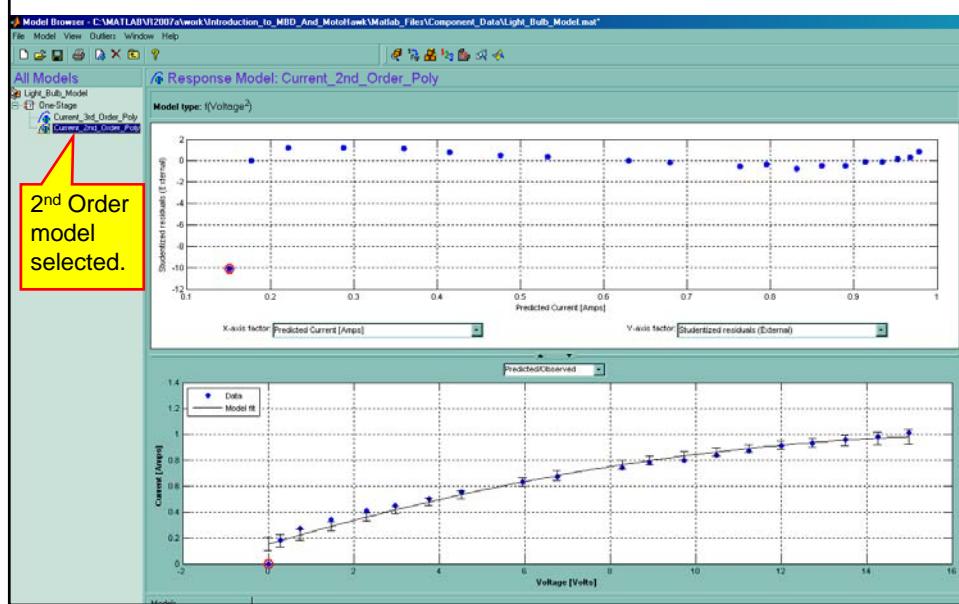
45

## 3<sup>rd</sup> Order Response Model



46

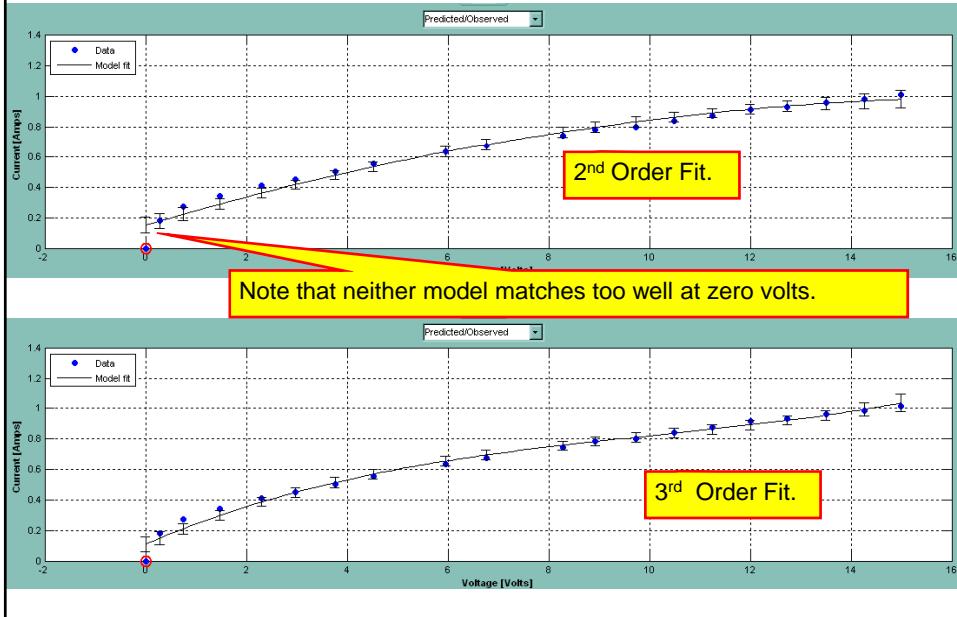
## 2<sup>nd</sup> Order Response Model





47

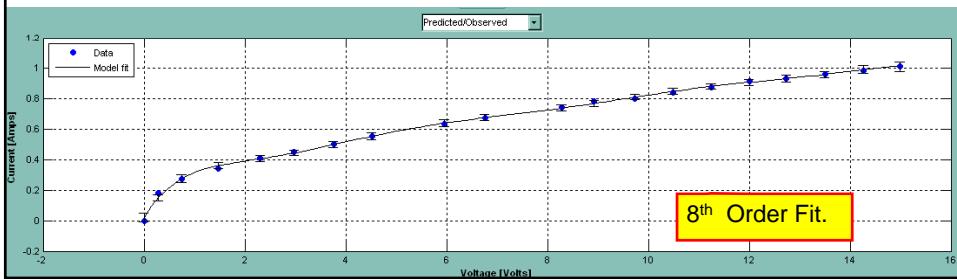
## The Two Models are Slightly Different



48

## Models

- You can experiment with more models.
- An 8<sup>th</sup> order polynomial comes pretty close to modeling the bulb at low voltages.
- An 8<sup>th</sup> order poly is much more expensive to calculate than a 3<sup>rd</sup> order polynomial

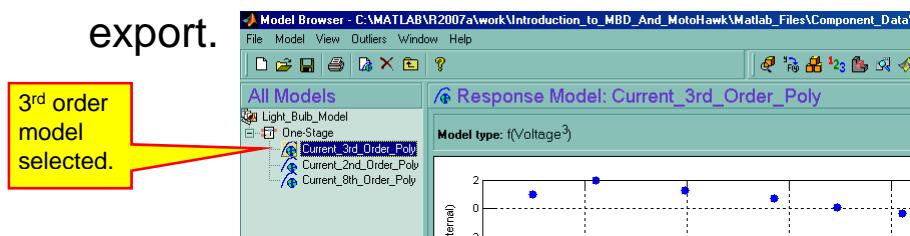




49

## Model Export

- Once we are happy with the model, we can export it to Simulink.
- We have several different models, so we need to select the model we want to export.



- Select File and then **Export Models** from the Model Browser menus.

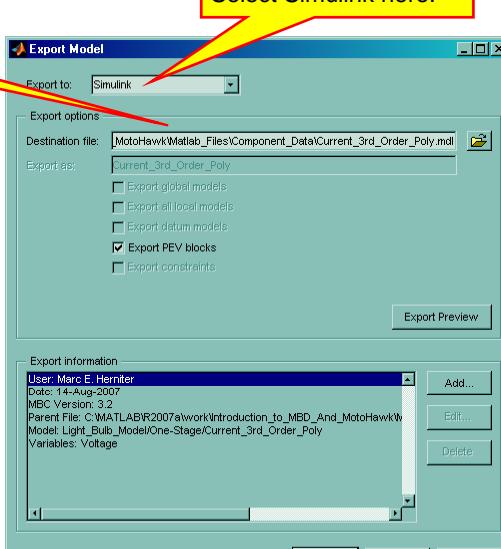
50

## Model Exporter

- Click the **OK** button.
- Simulink will start and display the model.
- Resize the model to see the ports.

Note the file name.

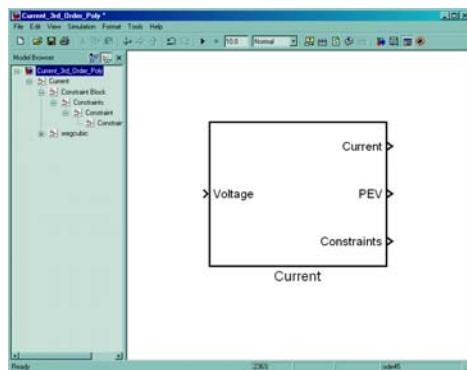
Select Simulink here.





## Export Model

51

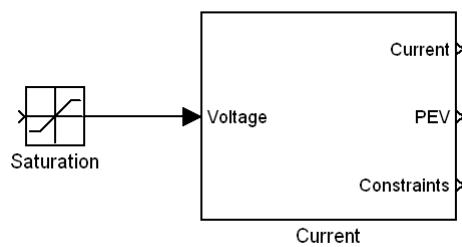


- We can now copy this block and use it in our motor-generator model.

## Export Model

52

- You should probably add a saturation block to limit the input voltage range of the model.
- The accuracy of the model outside the range of measured data will be questionable.



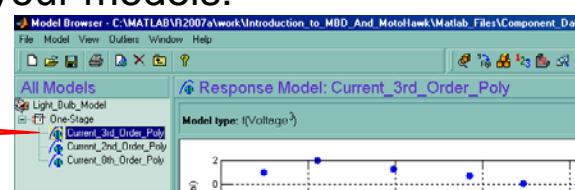


## Model Evaluation

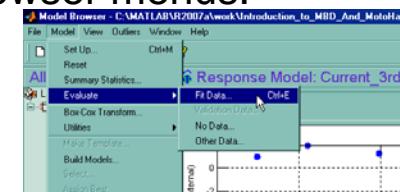
53

- Next, we will look at a tool for evaluating our model.
- Select one of your models:

Model selected.



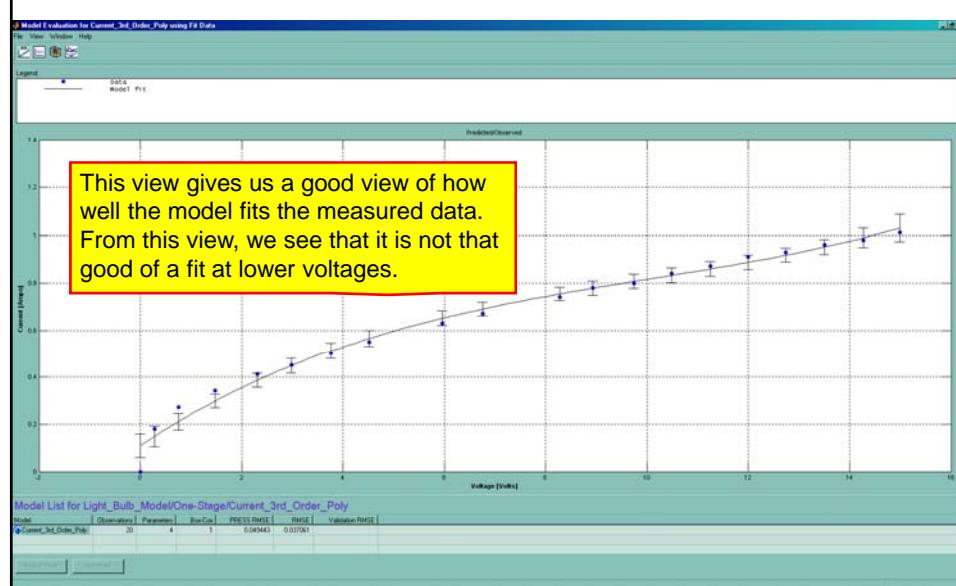
- Select **Model**, **Evaluate**, and then **Fit Data** from the Model Browser menus:



## Model Evaluation

54

This view gives us a good view of how well the model fits the measured data. From this view, we see that it is not that good of a fit at lower voltages.

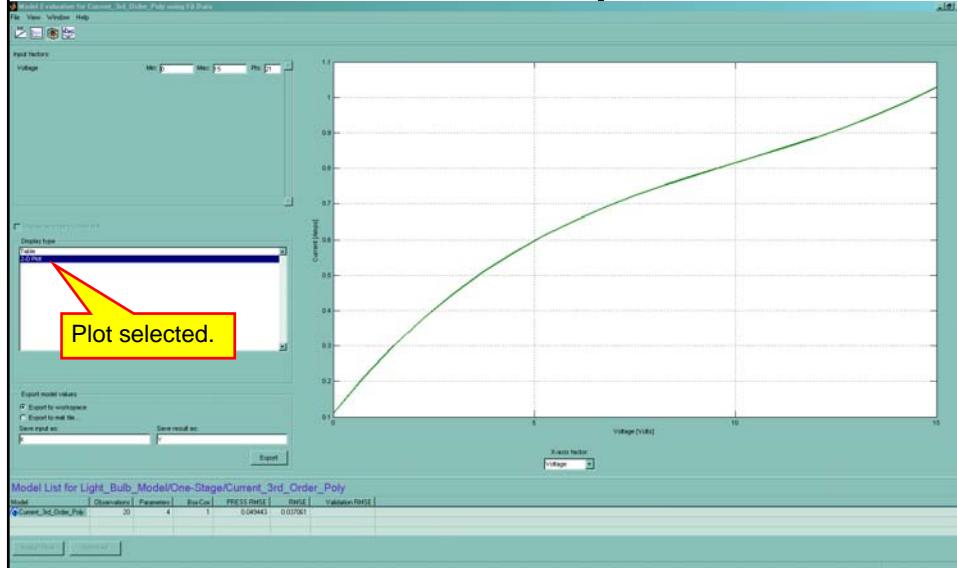




## Model Evaluation

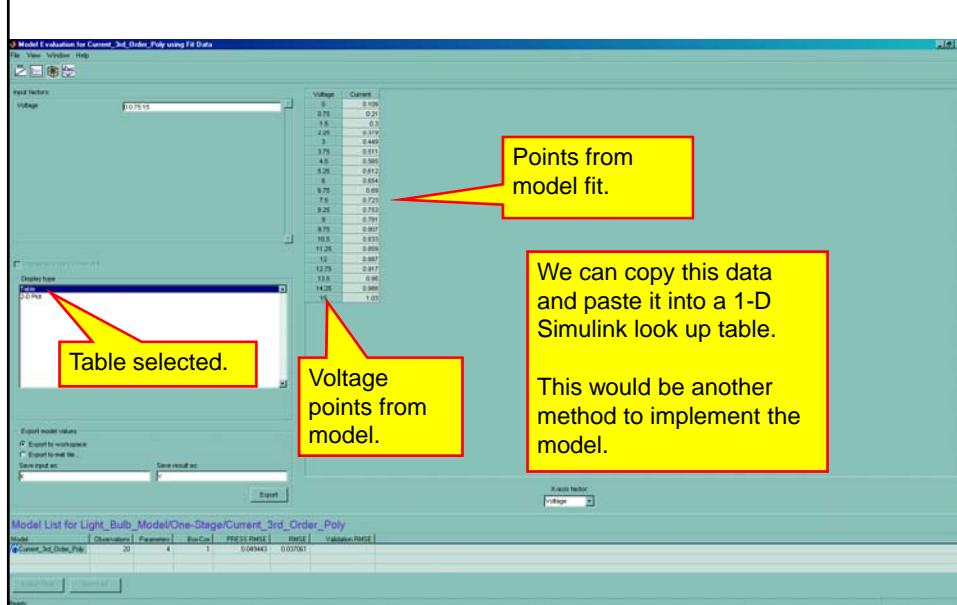
55

- Select View and then Response Surface.



## Model Evaluation

56

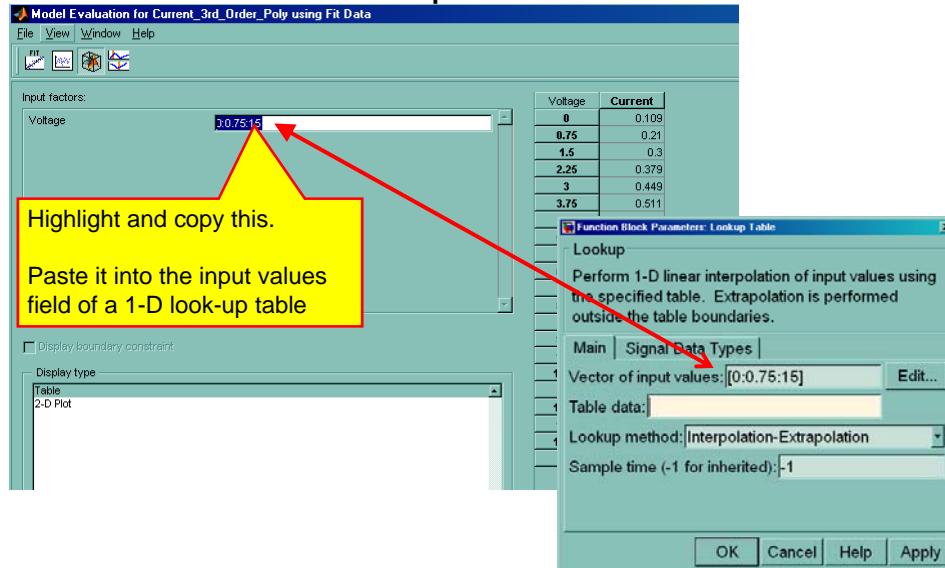




## Simulink Look-Up Table

57

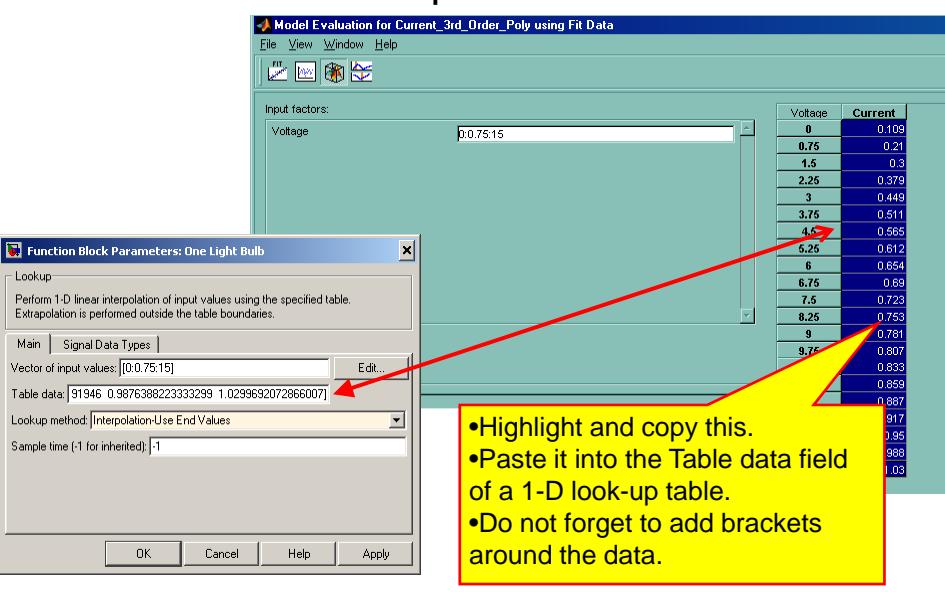
- To create a look up table in Simulink:



## Simulink Look-Up Table

58

- To create a look up table in Simulink:

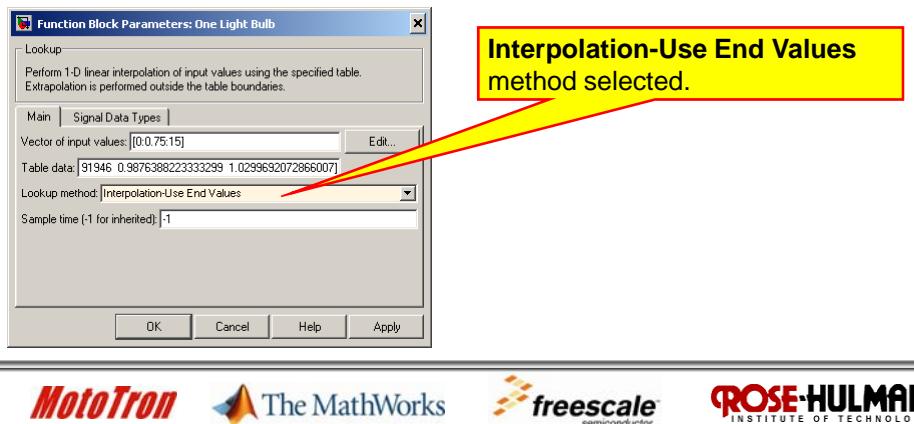




## Simulink Look-Up Table

59

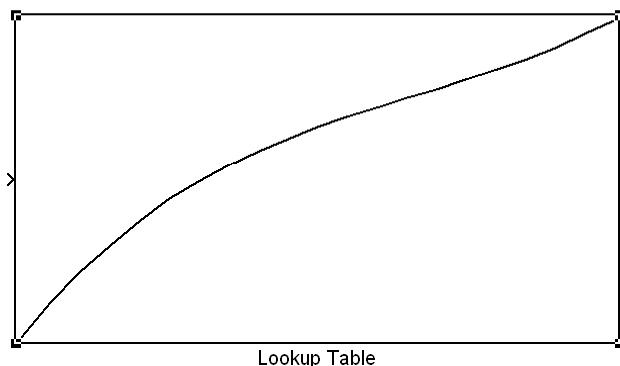
- Select a **Lookup method** of **Interpolation-Use End Values** so that we only use the table for the values that we actually measured.

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

## Simulink Look-up Table

60

- When you click the **OK** button on the Look-Up table dialog box, the curve should be shown on the block:





## New Bulb Model

61

- You can use either the look-up table model or one of the models that we exported from the MCBModel tool.
- One note is that with the models generated by MCBModel, none of the ones we created had zero current when the bulb voltage was zero.
- This is unphysical and will cause the generator to apply a negative torque when the generator is not spinning.



## New Bulb Model

62

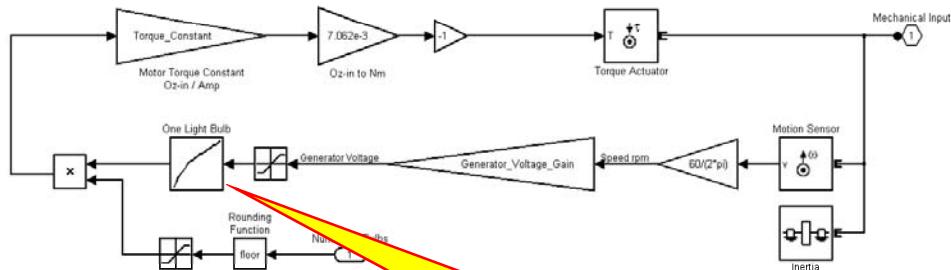
- This negative torque will tend to cause the motor-generator system to spin backwards when the motor-generator system should be at rest and no motor torque is applied.
- We probably will not notice anything because this torque might be small compared to the rotational friction we added.
- To avoid this problem, we will use the look-up table for our model.





## New Generator Model

63



Lookup table added here. Input is voltage, the output is current.

**MotoTron**

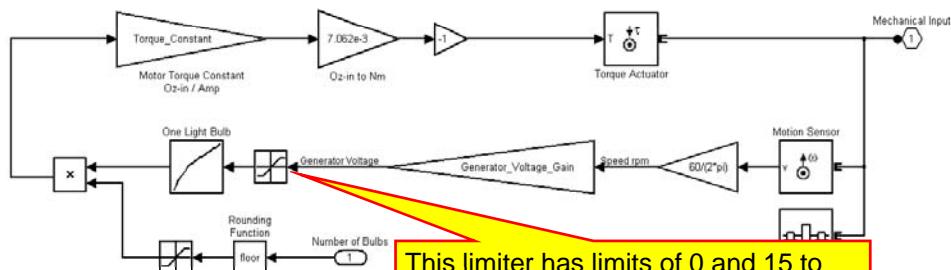
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## New Generator Model

64



This limiter has limits of 0 and 15 to limit the input voltage to the table. This limit is redundant since we selected the Interpolation-Use End Values method for the lookup table.

**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/>

## Lecture 17

### Generator Model With Bulb

65

- Demo your working model with the New Bulb Load

Demo\_\_\_\_\_



The MathWorks



freescale<sup>®</sup>  
semiconductor



## Lecture 17

66

- Start Lecture 17 MBC Project



The MathWorks



freescale<sup>®</sup>  
semiconductor





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



## Introduction to Model-Based Systems Design

### Lecture 18: Verification and Validation MathWorks SystemTest



## Verification and Validation

2

- As we develop our model, we will make periodic changes.
- Every time we make a change, we need to verify that our system passes a rigorous set of tests.
- Ideally, we would have a predefined set of tests, and run all of the tests each time we make a change.
- We would then analyze the results of the tests and determine if our system meets the required specifications.





3

## Verification and Validation

- Ideally, we would set up a battery of tests at several levels in the Model-Based-System Design process:
  - Simulations (SIL)
  - Real-Time Simulations (xPC)
  - Hardware in the Loop Simulations
- We want to create a set of tests that “thoroughly” test the system, and then run all of the tests automatically.



4

## Verification and Validation

- Proving that a set of tests completely covers a design is a challenge and will not be discussed here.
- Instead, we will demonstrate a tool that:
  - Allows us to create a set of tests
  - Run the tests automatically
  - Perform data analysis on the results
  - Evaluate the results against performance specifications.





5

## Verification and Validation

- We will use a MathWorks tool called SystemTest and use it to verify the step response of our system.
- We will start with model Lecture17\_Model3.
- Save the model as Lecture18\_Model1.
- We will need to make a few changes to the model.
- Changes to the top level are shown next:

**MotoTron**

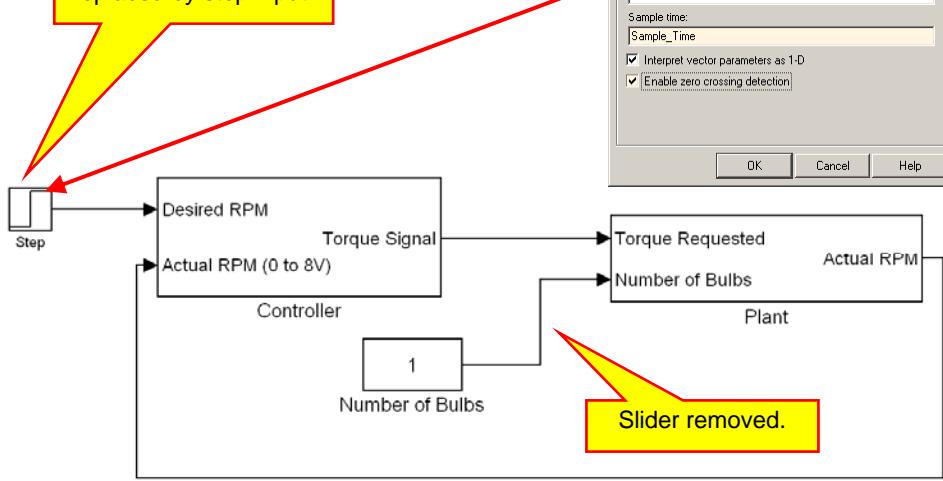
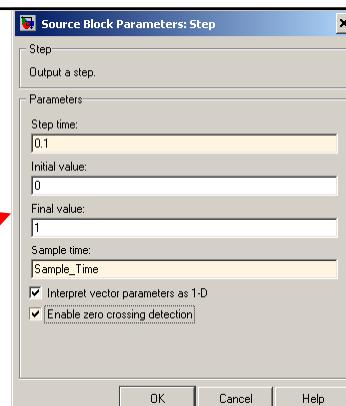
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

### Motor-Generator Model

Slider and constant replaced by step input.





7

## Model Changes

- We are going to need to collect some data from our model when it runs.
- We will do this using the **To Workspace** part located in the **Simulink / Sinks** library.
- This part writes data from the Simulink simulation to a MATLAB variable in the workspace.
- Place one in the controller subsystem and connect as shown:

**MotoTron**

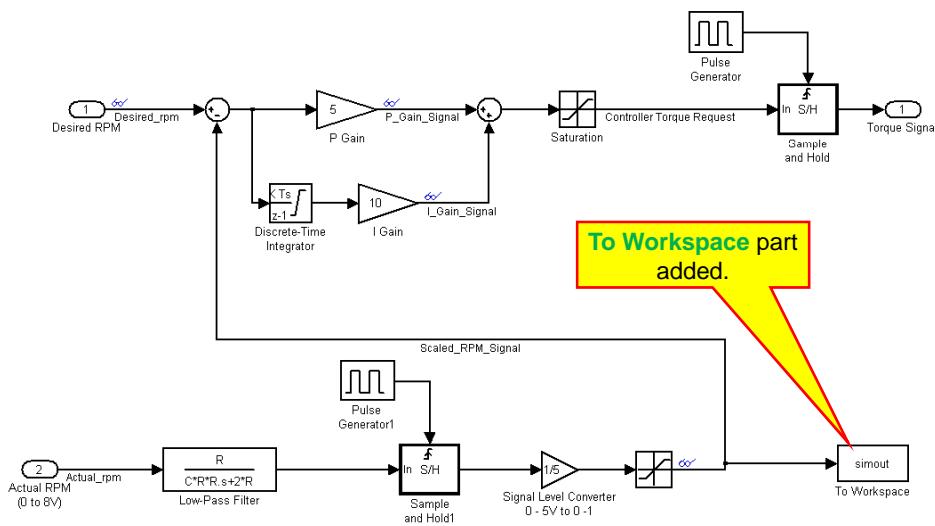
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

8

## Controller Subsystem



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

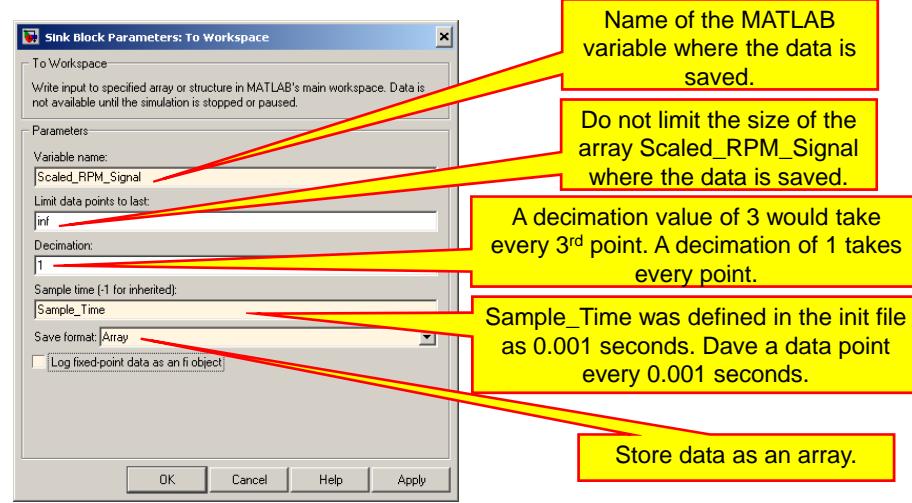
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



## Controller Subsystem

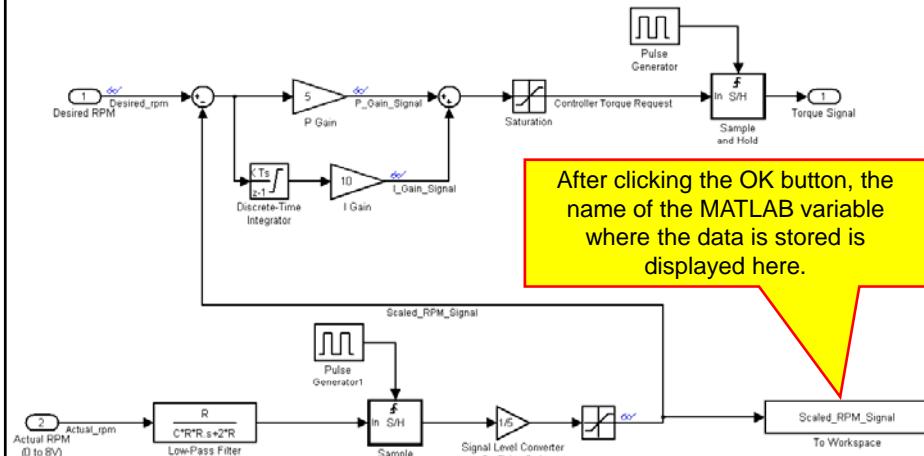
9

- Change the parameters of the To Workspace block as shown:



## Controller Subsystem

10

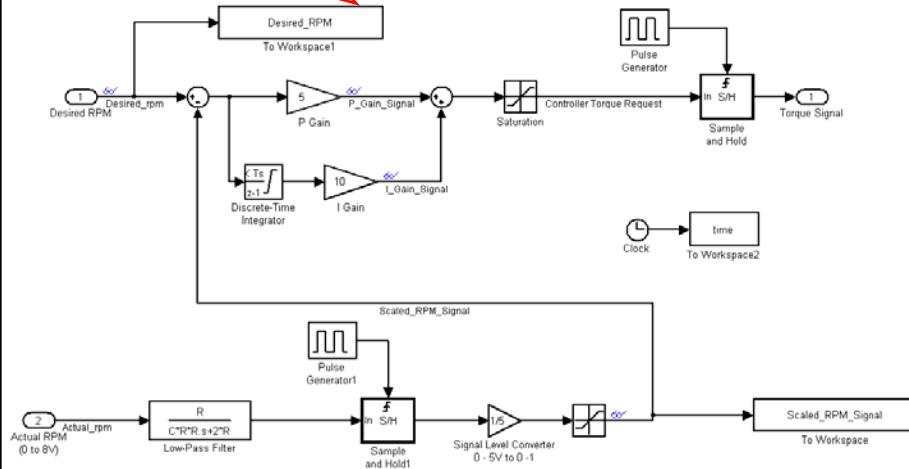




## Controller Subsystem

11

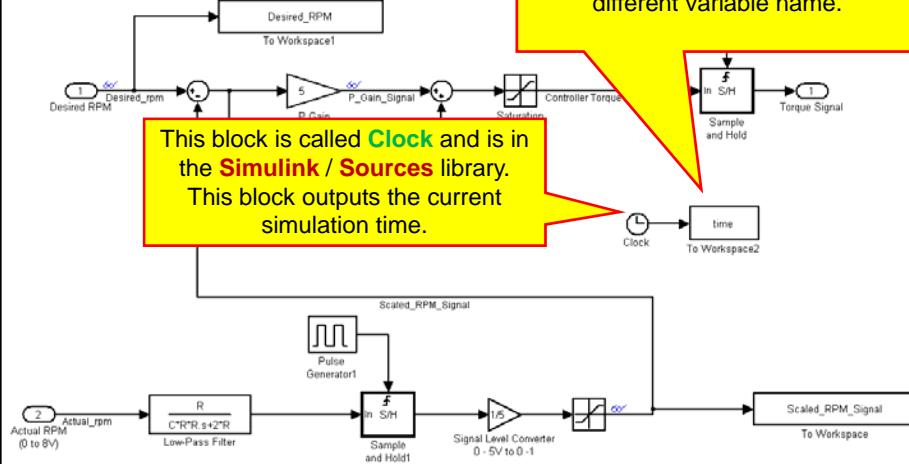
Add another **To Workspace** here  
with the same settings except a  
different variable name.



## Controller Subsystem

12

Add another **To Workspace** here  
with the same settings except a  
different variable name.

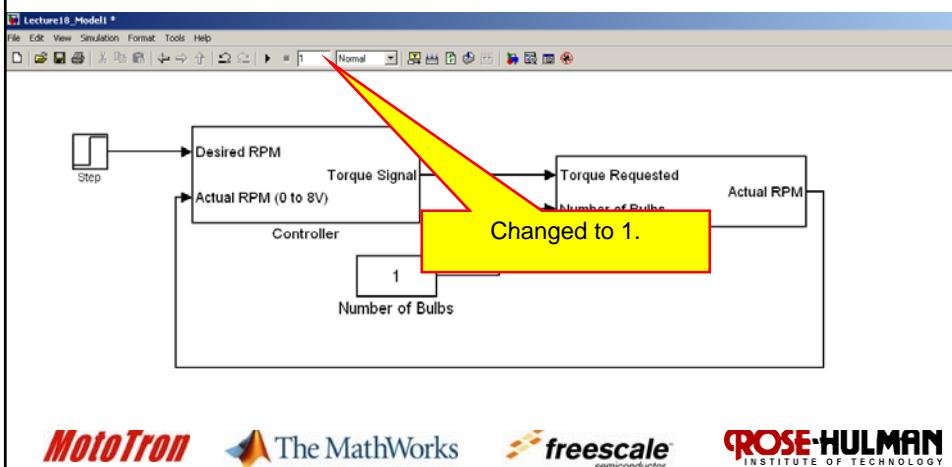




## Model Changes

13

- Next, change the simulation ending time to 1 second:

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

## Init File

14

- We do not need to make any changes to the Init\_File for our model.
- The SystemTest tool will run our Simulink model a large number of times.
- The contents of the Init\_File never change during the test and we never need to change the values of the variables defined in the Init\_File during the series of tests.

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



15

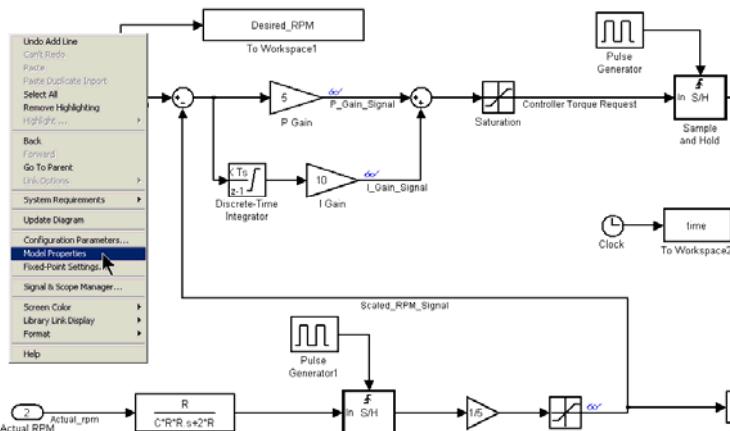
## Init File

- We only need to run the init file once, then we can run our model through the series of tests.
- Instead of using a Callback to have Simulink run the init file, we will use the Pre Test function in SystemTest to initialize the needed parameters once.
- We will change the CallBack function in Simulink so that it does not ruin our Init\_File.

16

## Init\_File

- Right-click somewhere on the motor-generator Simulink model and select **Model Properties** from the menu:

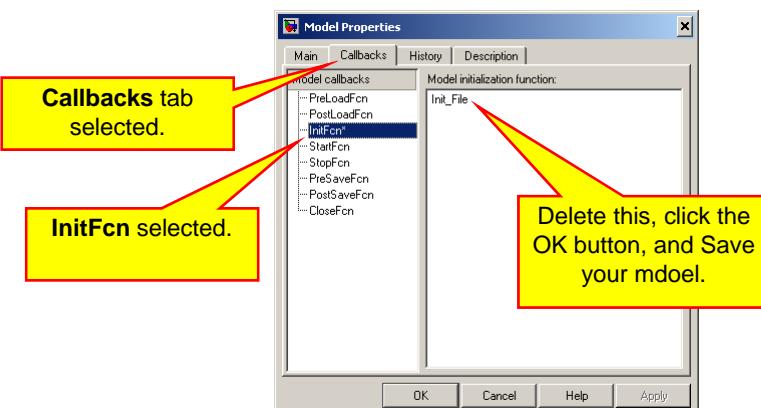




## Init\_File

17

- Select the Callbacks tab and select the InitFcn selection



**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Post Processing File

18

- The test we are going to run is to observe the step response of our motor-generator system and calculate the rise time.
- We will then determine if the system passes the step response specification.
- We need to create a MATLAB script that calculates the rise time from the measured data:

**MotoTron**

The MathWorks

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



19

## Rise Time MATLAB Script

```
%Find the rise time.

%Calculate the Rise Time

% Define the rise time as the amount of time it takes the
% output to go from 10% of the final value to 90%
% of the final value.

Ten_Pct_Value = 0.1*Speed_Step;
Ninety_Pct_Value = 0.9*Speed_Step;
Ten_Pct_Time = 0;
Ninety_Pct_Time = 0;
```



The MathWorks



20

## Rise Time MATLAB Script

```
%Find the time of the 10% Point.

for i = 1: (length(time)-1)
    if (Scaled_RPM_Signal (i) <= Ten_Pct_Value) &&...
        (Scaled_RPM_Signal (i+1) >= Ten_Pct_Value)
        Ten_Pct_Time=time(i);
    end

%Find the time of the 90% Point.

    if (Scaled_RPM_Signal (i) <= Ninety_Pct_Value) &&...
        (Scaled_RPM_Signal (i+1) >= Ninety_Pct_Value)
        Ninety_Pct_Time=time(i);
    end
end
```



The MathWorks





## Rise Time MATLAB Script

21

```
% Calculate the rise time as the difference between the
% 10% and 90% points. If a rise time is not found, set the
% Rise_Time to infinity.
```

```
if (Ten_Pct_Time > 0) && (Ninety_Pct_Time > 0)
    Rise_Time = Ninety_Pct_Time-Ten_Pct_Time;
else
    Rise_Time = inf;
end
```



## SystemTest

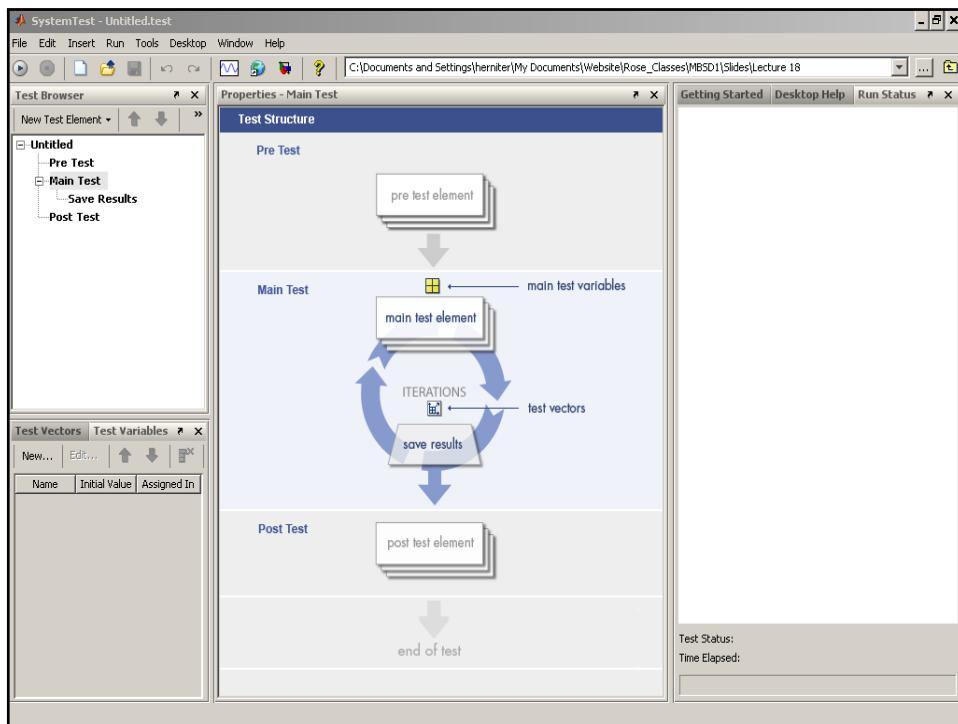
22

- We will now use MathWorks System test to run the model at various speeds and measure the step response for different loads.
- Type systemtest at the MATLAB command prompt to run the tool.





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



## SystemTest

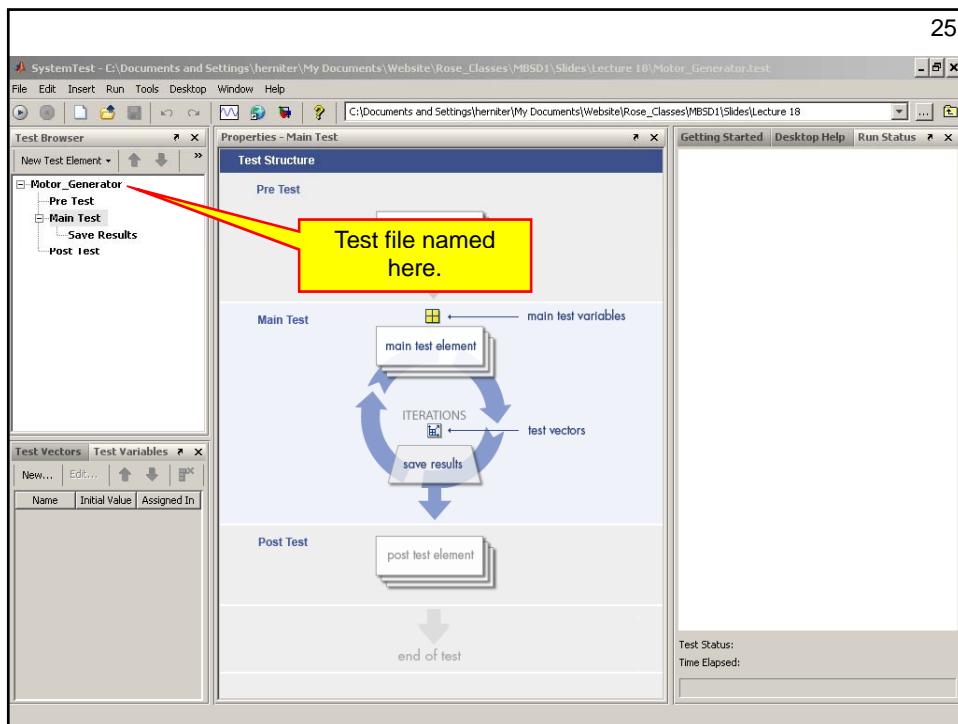
24

- Select **File** and then **Save As** from the System Test menus
- Change the directory to your MATLAB working directory
- Save the file as “Motor\_Generator.test”
- The tree will change as shown:





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



## Pre Test

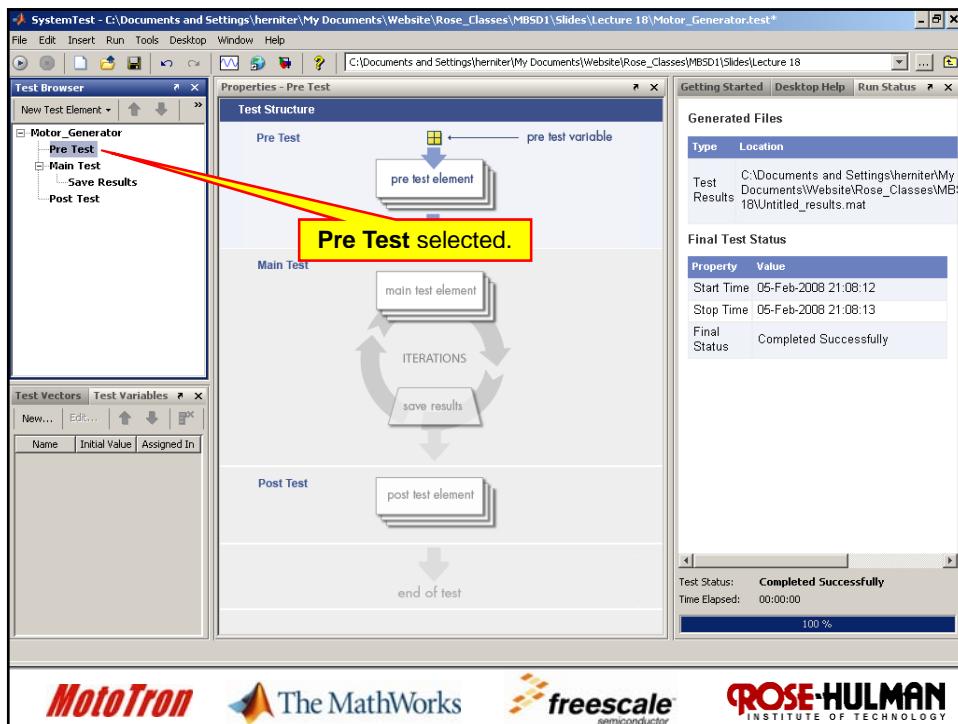
26

- The pretest section allows us to initialize variables that we could use in our test.
- This is done once even though multiple tests will be run.
- We would like to use the Pre Test section to initialize the workspace constants that we use in our model.
- Select the **Pre Test** leaf of the tree:





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



**MotoTron**

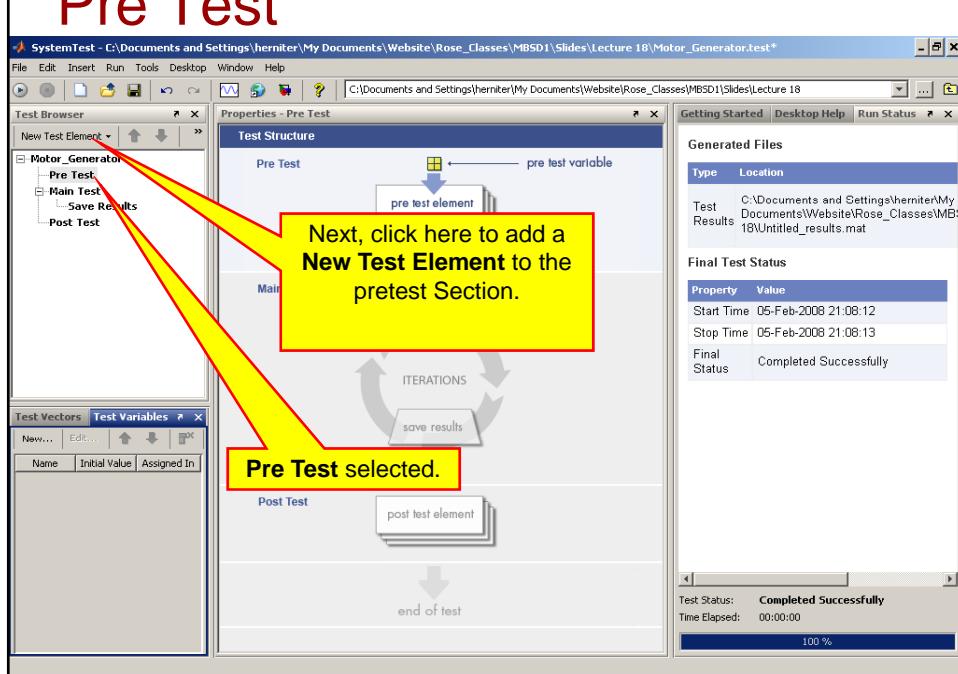
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

## Pre Test

28





## Pre Test

29

Select MATLAB to create a MATLAB script that will execute once before all of the tests are run.

## Pre Test

30

```

1 %%% MATLAB Element
2 % Executes MATLAB code during Pre Test, Main Test, and Post Test
3 %
4 % MATLAB variables declared here can be accessed in all three sections
5 % a test by creating SystemTest test variables as follows:
6 % 1) Select the Insert menu option "Test Variable..." and choose the
7 %    variable you want to use.
8 
```

New MATLAB script.  
Delete the comments shown here, copy the contents of file Init\_File.m, and paste them in this window.



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

The screenshot shows the SystemTest software interface with a MATLAB script titled "Motor\_Generator.test". The script contains the following code:

```

4
5
6 %Low Pass Filter Constants
7 R=10000; %Ohms
8 C=0.1e-6; %Farads
9
10
11 %Motor and Generator Contants
12 Motor_Inertia = 4.378e-6; %kg*m^2
13 Torque_Constant = 9.1; %Oz-In per amp
14 Motor_Rated_Current = 4; % Amps
15 Generator_Inertia = 4.378e-6; %kg*m^2
16 Generator_Voltage_Gain = 24/3500; %Volts per
17
18 %Load Constants
19 Bulb_R = 12; %Standard 12 V automotive bulb (C)
20
21 %Encoder Constant
22 Encoder_Gain = 2.5/1000; %Volts per rpm.
23
24

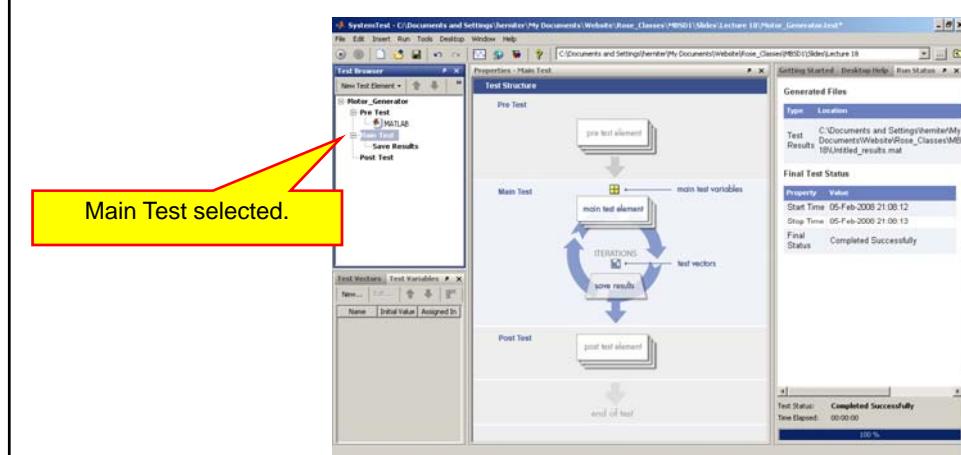
```

The "Properties - MATLAB" panel shows the generated files and final test status. The "Final Test Status" table indicates a successful completion.

## Main Test

32

- The remainder of our elements will go in the Main Test Section.
- Select the section as shown:





33

## Test Vectors

- Test vectors are MATLAB row vectors whose values are used as inputs to our tests.
- If we have a test vector with the values [1, 3, 5, 8, 10], a test will be run for the value 1, the value 5, the value 8, and the value 10.
- We can assign the values of the test vectors to parts of our Simulink model.



34

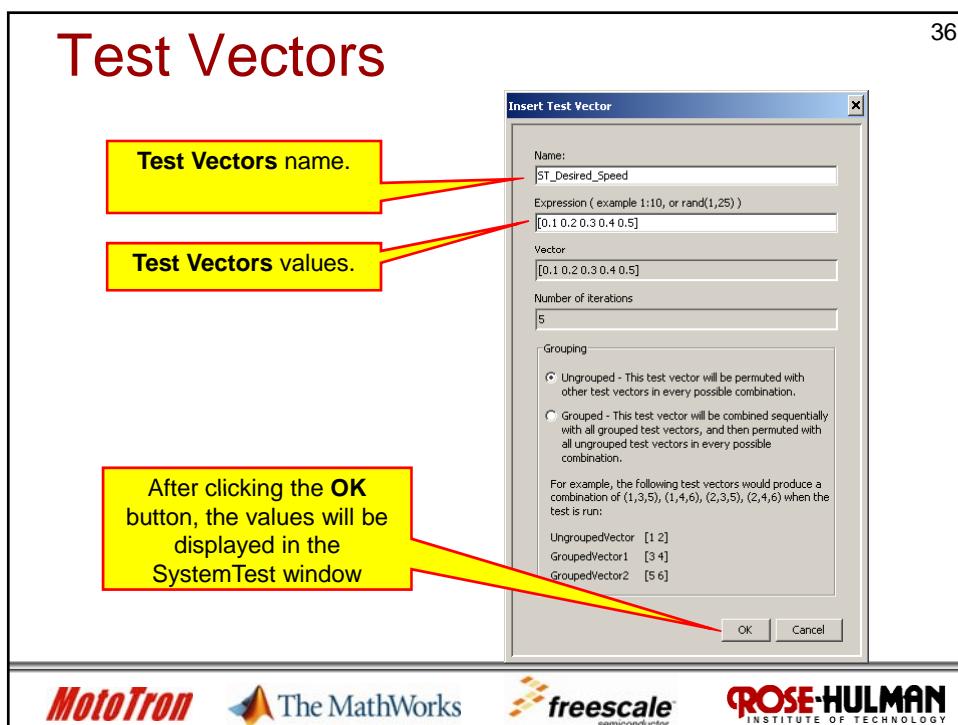
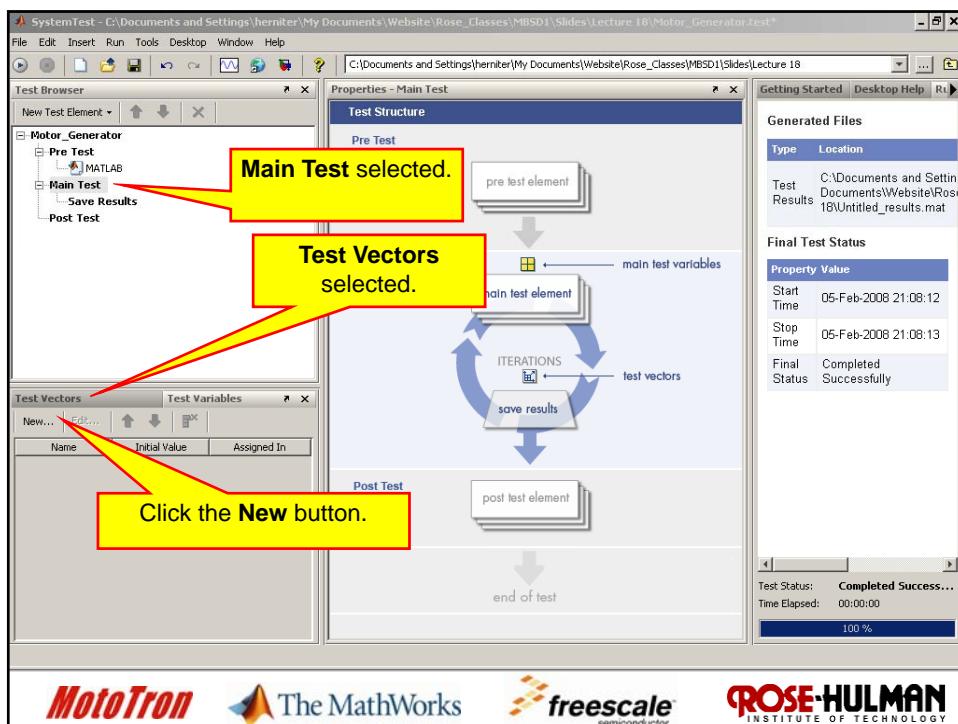
## Test Vectors

- For each magnitude of step, we would like to measure the rise time for different bulb loads.
- We will set the bulb load test vector to [0, 1, 2, 3, 4, 5, 6].
- We will call this test vector ST\_Bulb\_Load.
- We will have a total of 35 tests.
- We now need to define these test vectors.



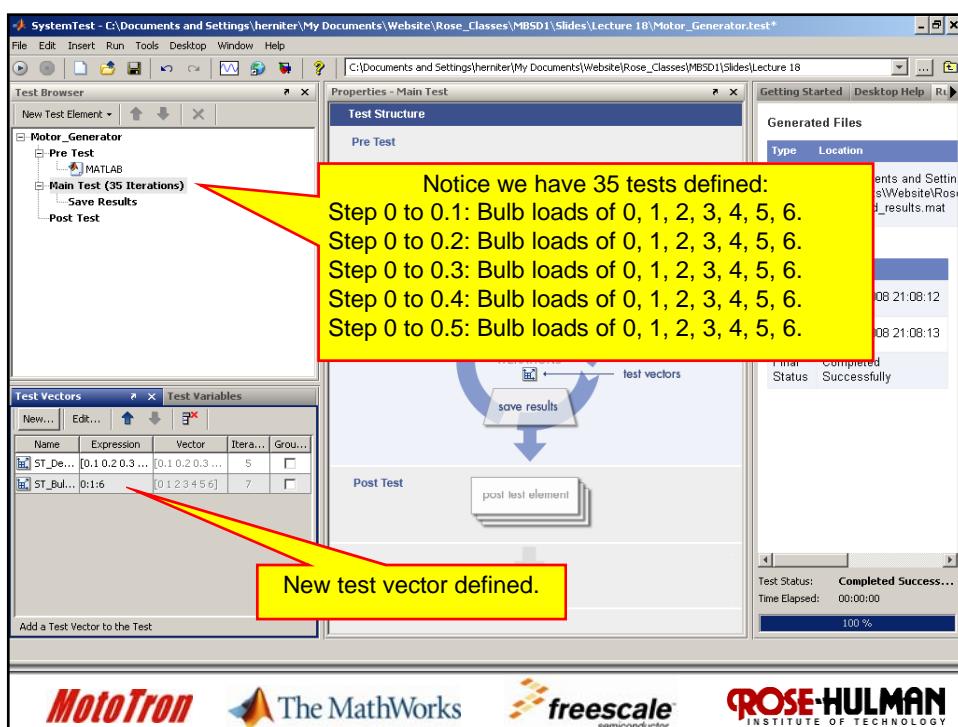
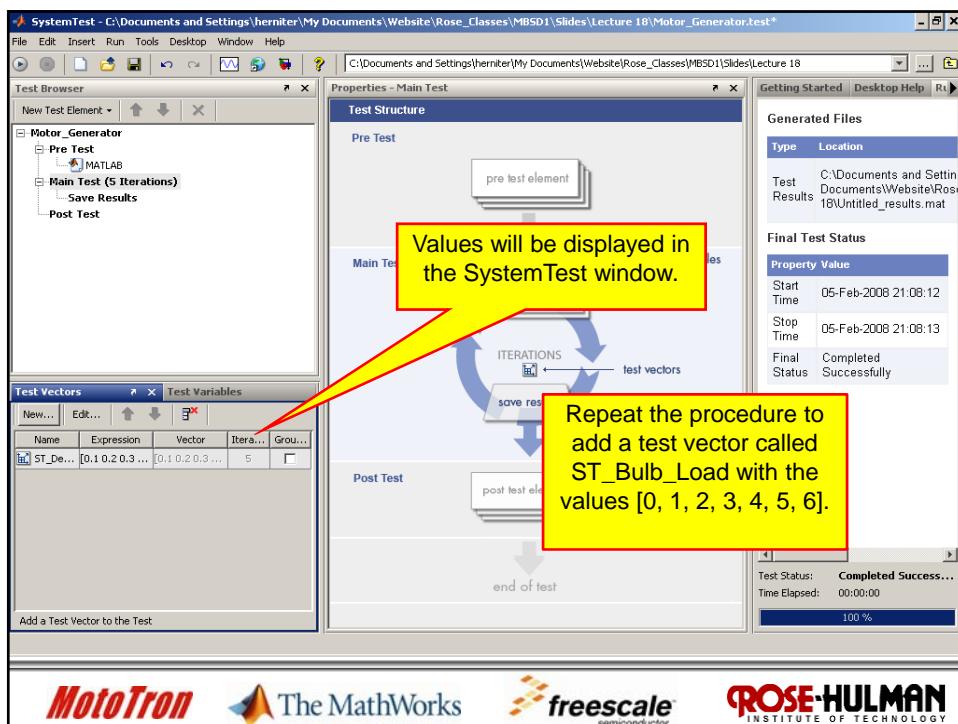


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





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





## Main Test

39

- Next we will define the Main Test Section.
- In this section, for each iteration in the test vectors, we want to:
  - Run our Simulink model
  - Calculate the Rise Time
  - Plot the desired and measured rpm response.
  - Determine if the response meets a performance criteria.



## Init\_File

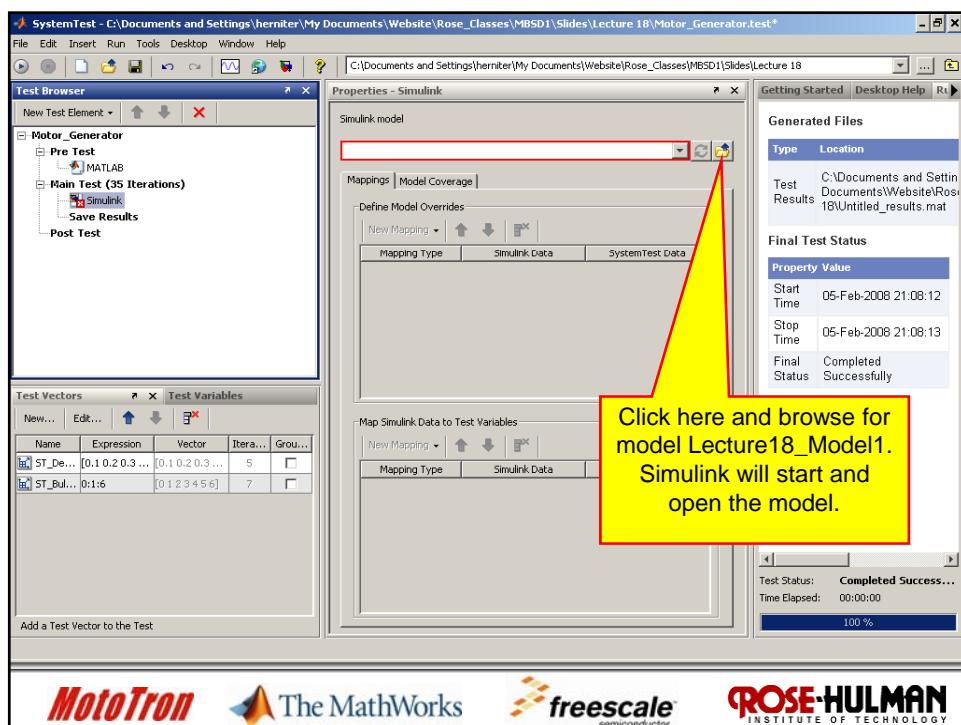
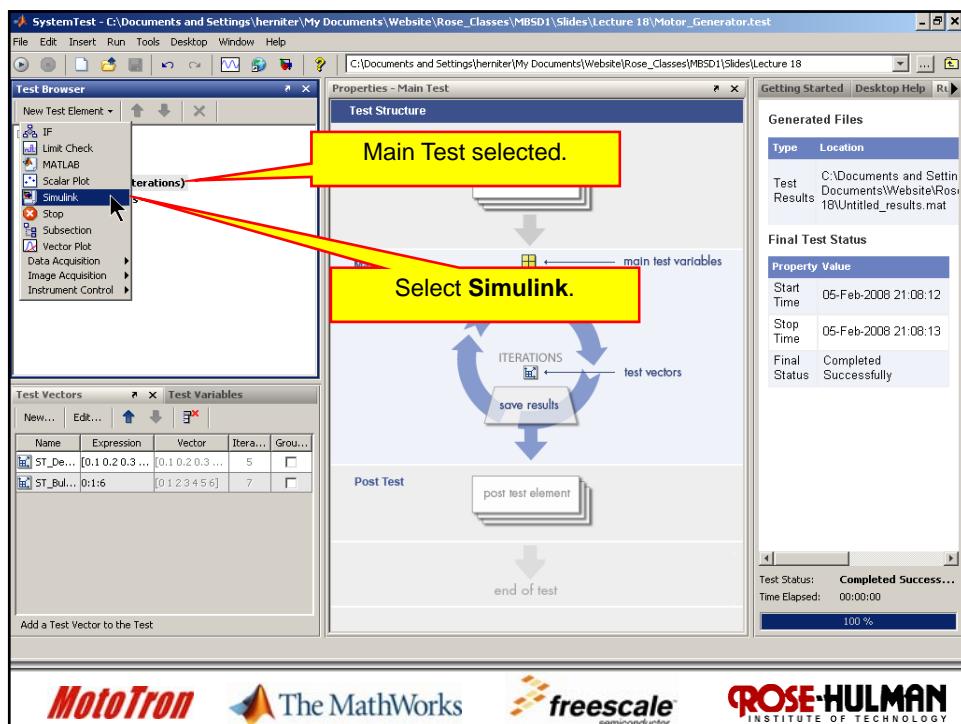
40

- Before adding your Simulink Model to the test plan, manually run the Init\_File to define workspace parameters for the model.
- The Pre\_Test function will not run until we start the test, so some model parameters are not known and will generate an error message.



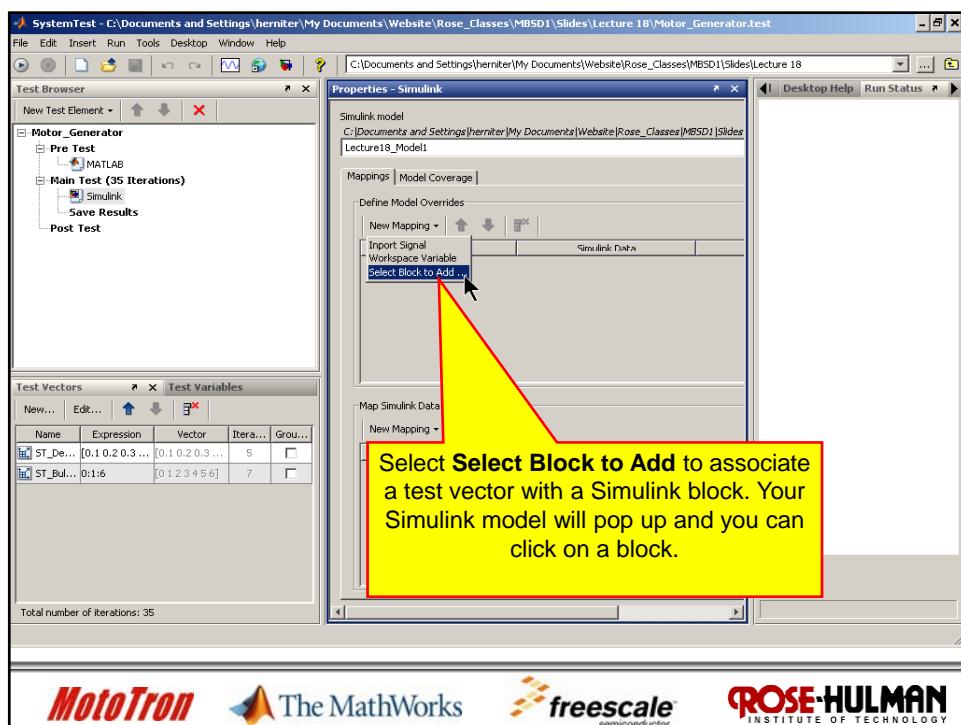
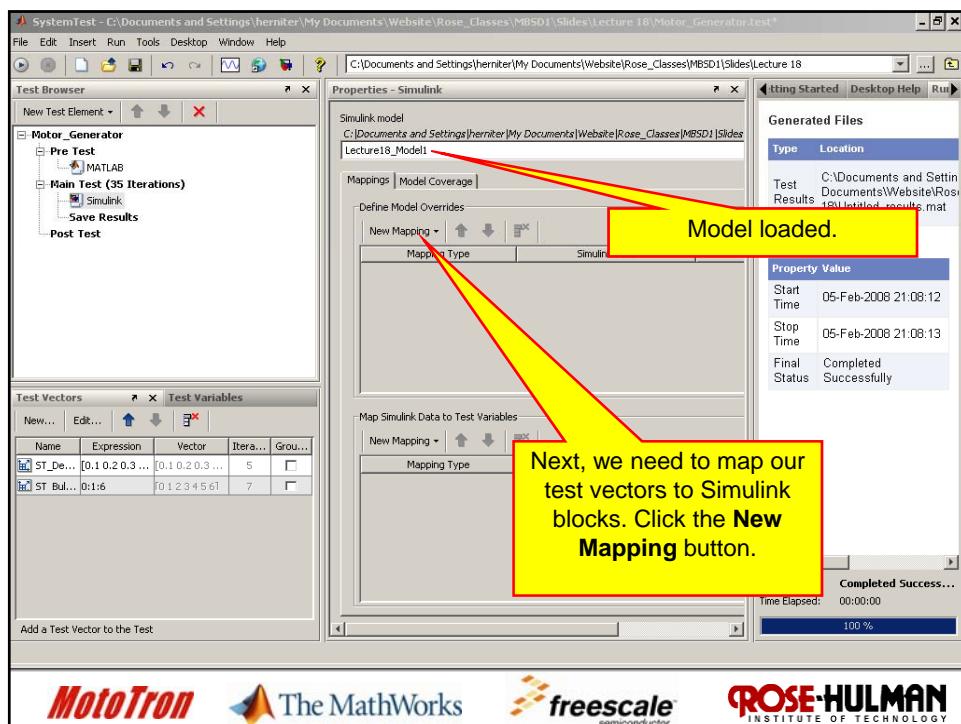


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





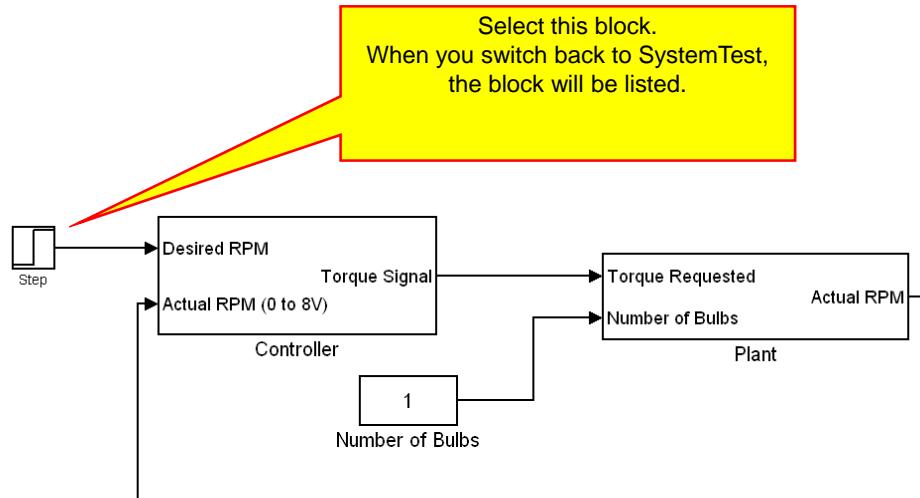
Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





45

## Simulink Model

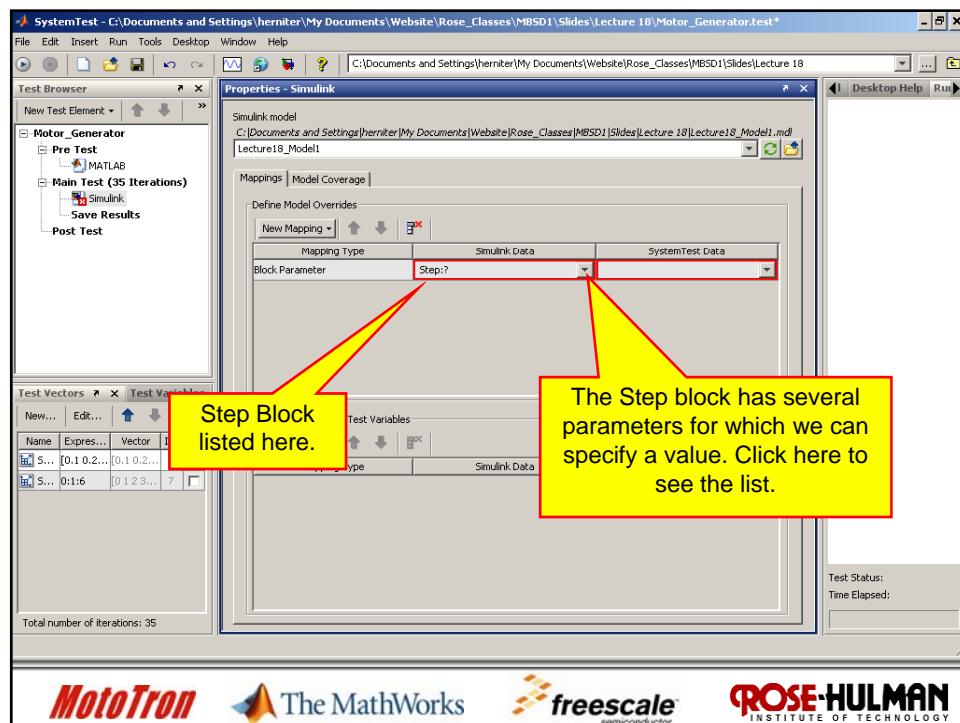


**MotoTron**

**The MathWorks**

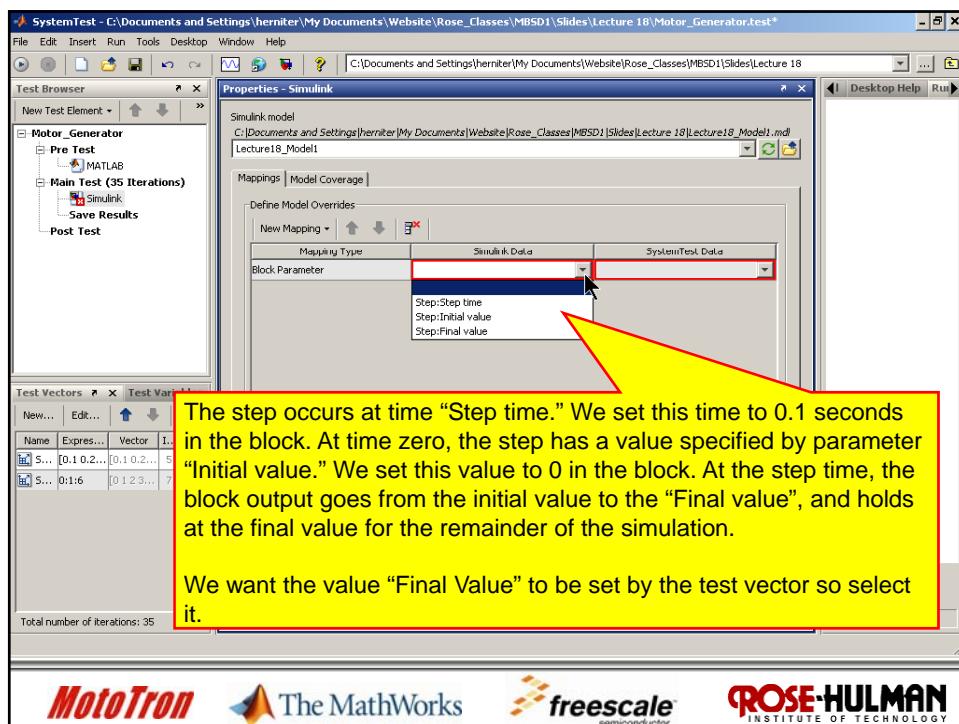
**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY





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

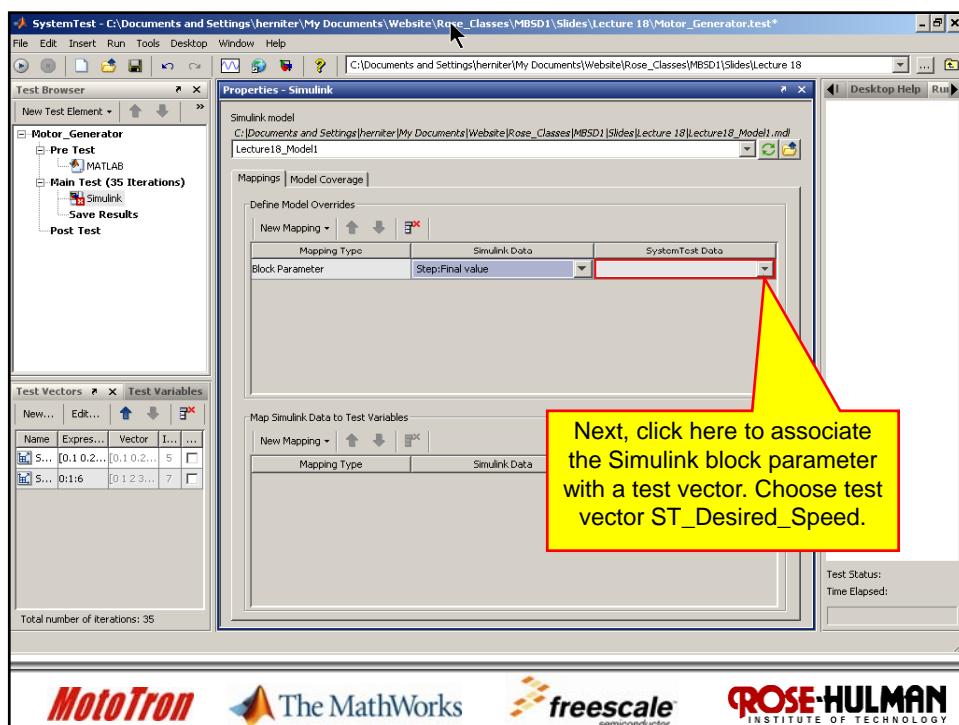


**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



**MotoTron**

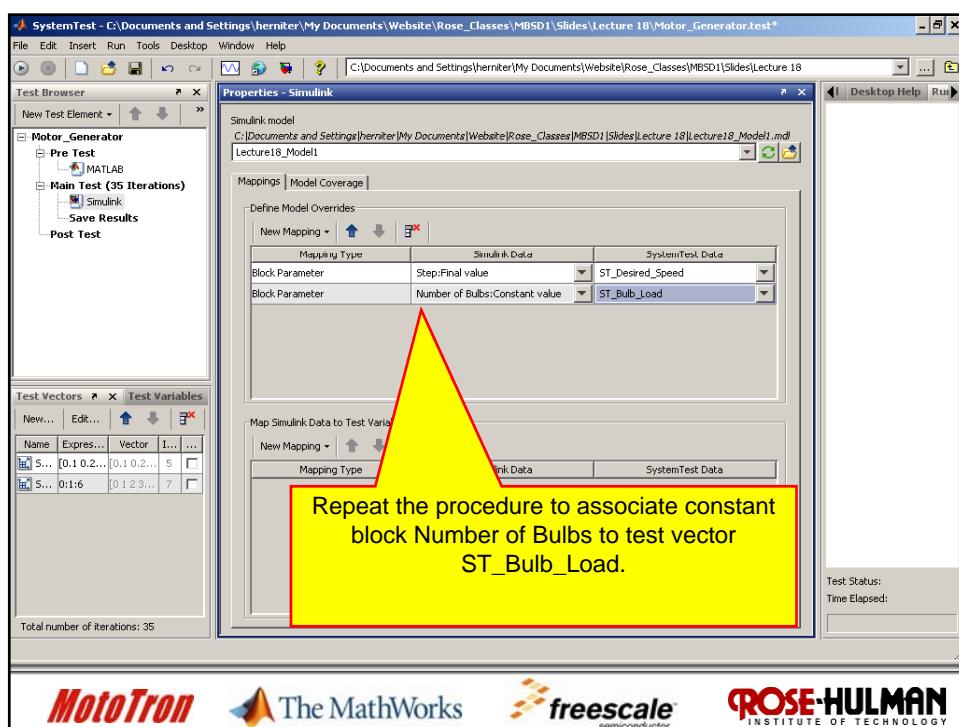
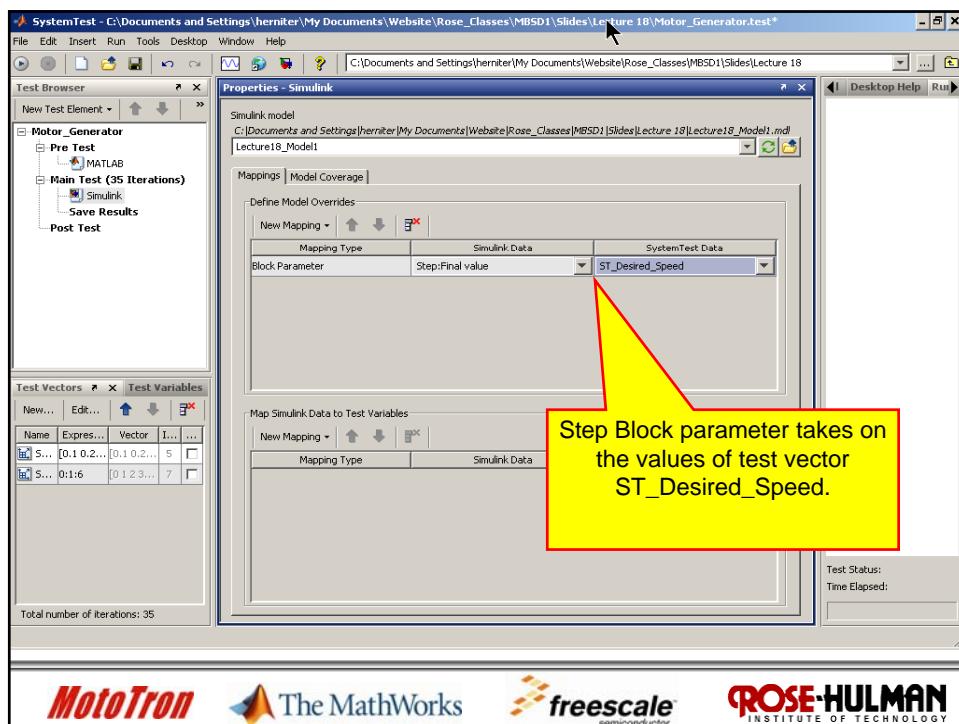
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



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

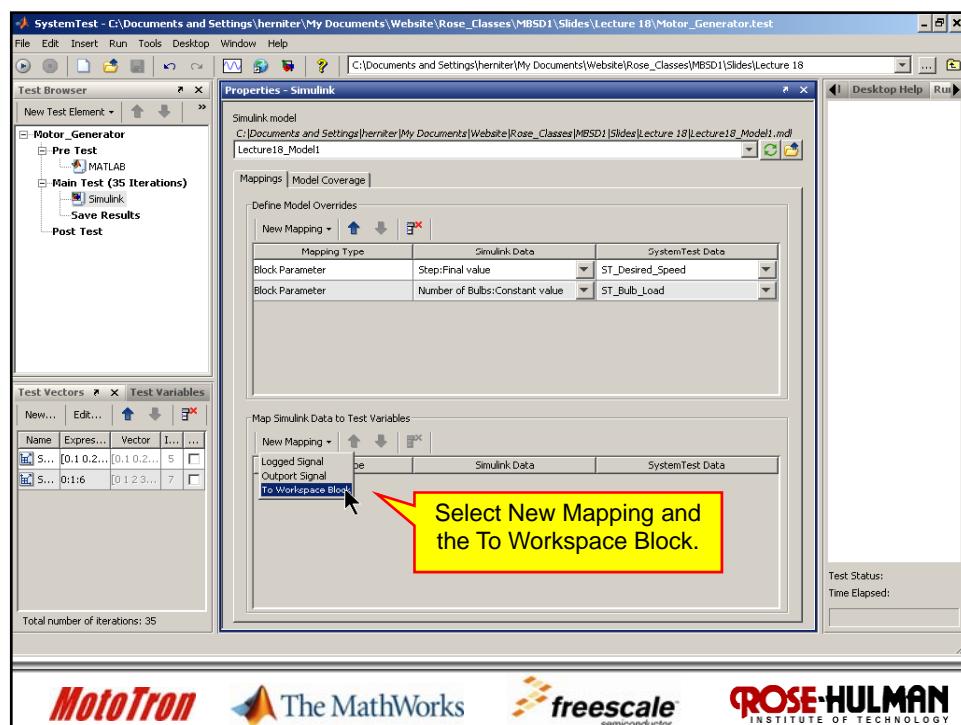




51

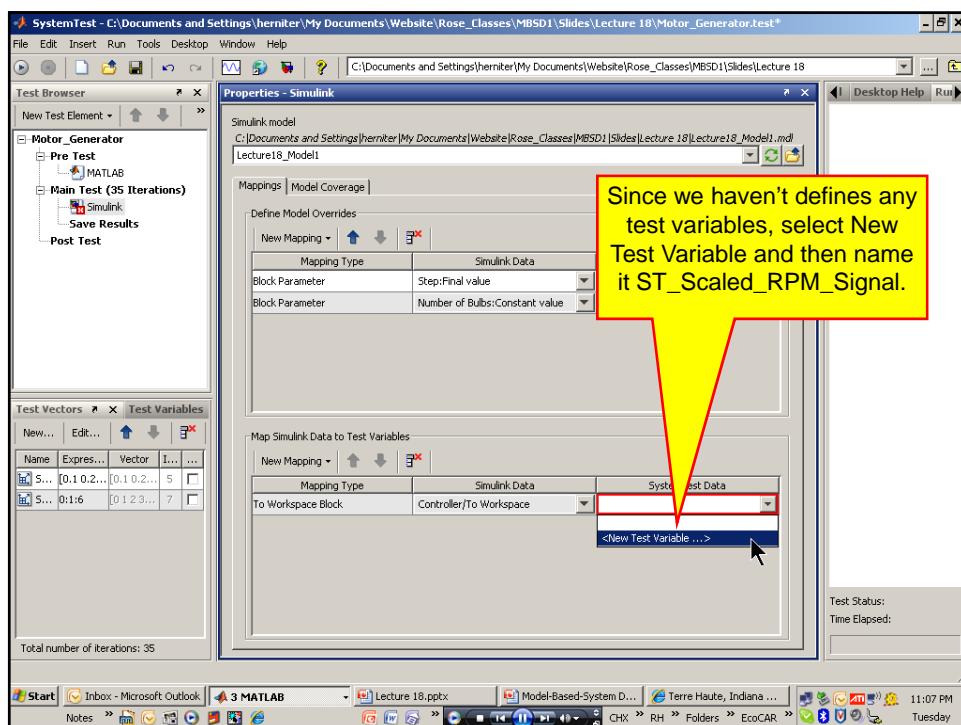
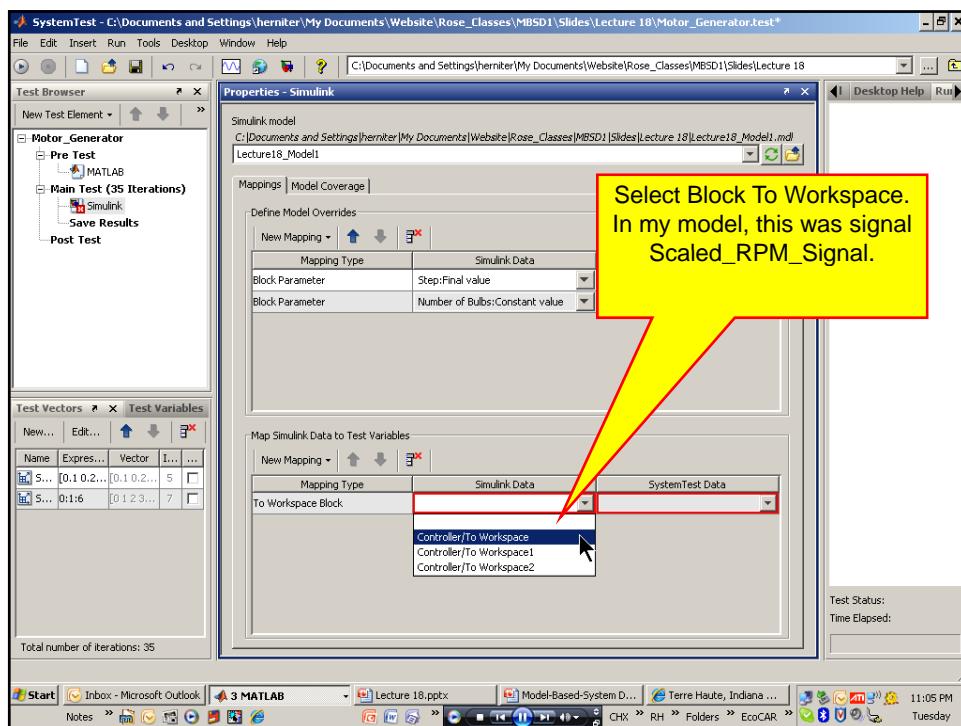
## Simulink Output

- For our reports, we would like to include a plot of the desired and measured step response.
- To do this, we need to get the data from the MATLAB workspace to System Test.
- In our Simulink model, we added three To Workspace blocks so that we would save the desired data in the workspace.
- We now need to take that data and save it as a test variable.



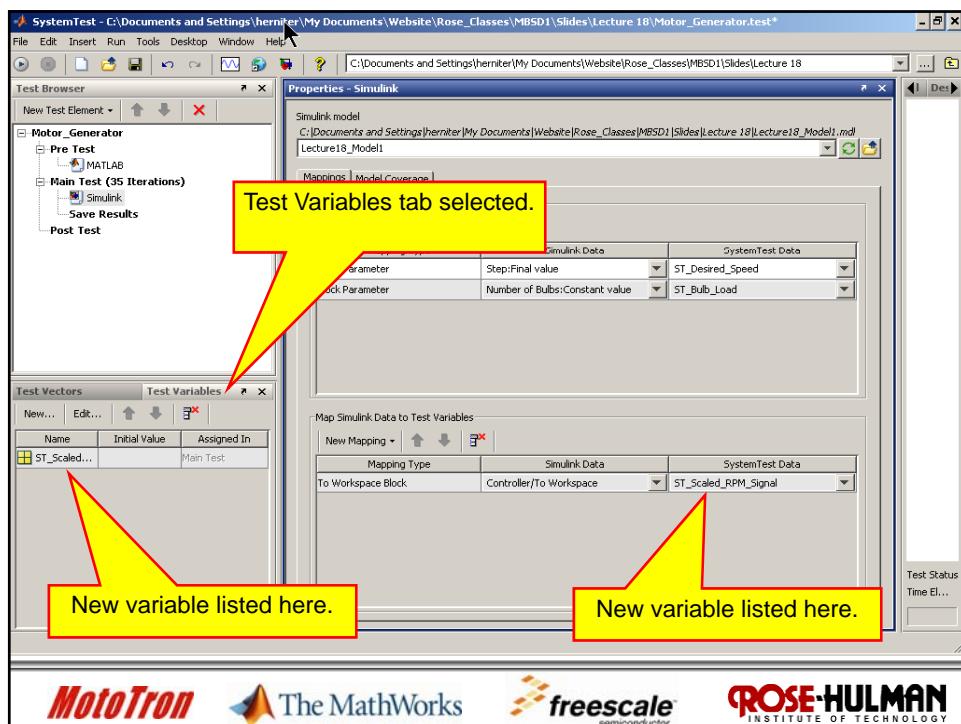


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





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



## Simulink Output

56

- Repeat the procedure for the other Two Workspace Blocks:
  - To Workspace → ST\_Desired\_RPM
  - To Workspace2 → ST\_time

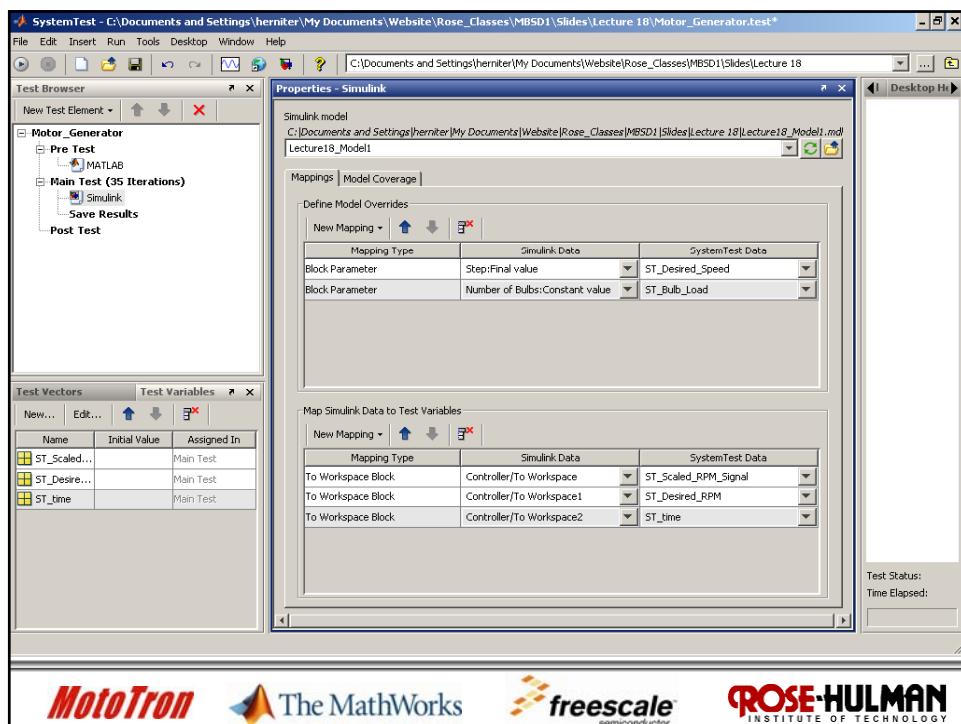


The MathWorks



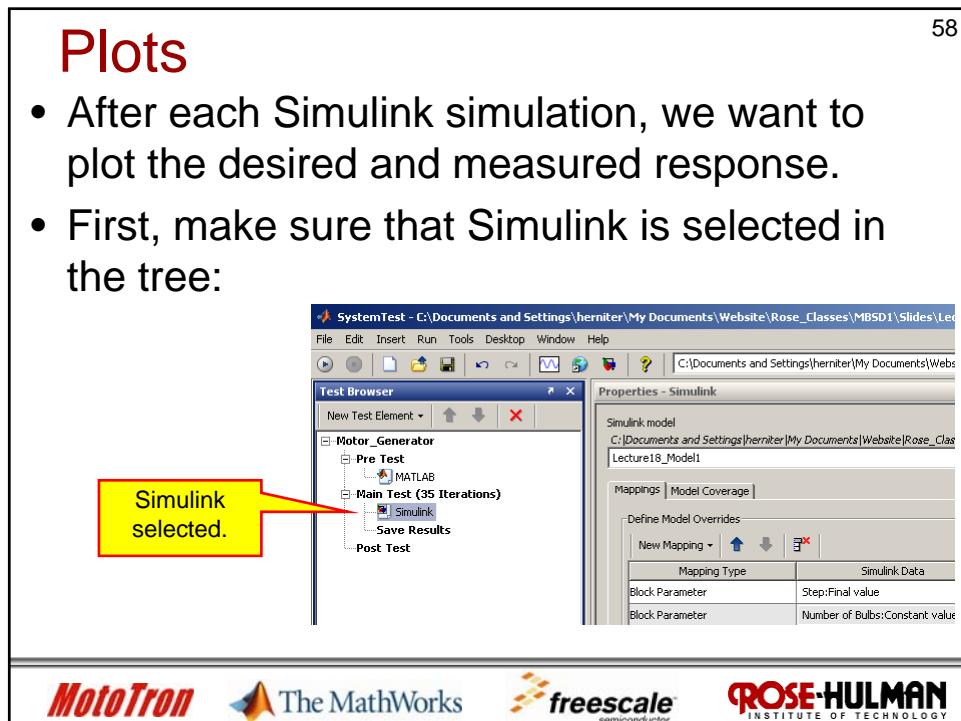


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>



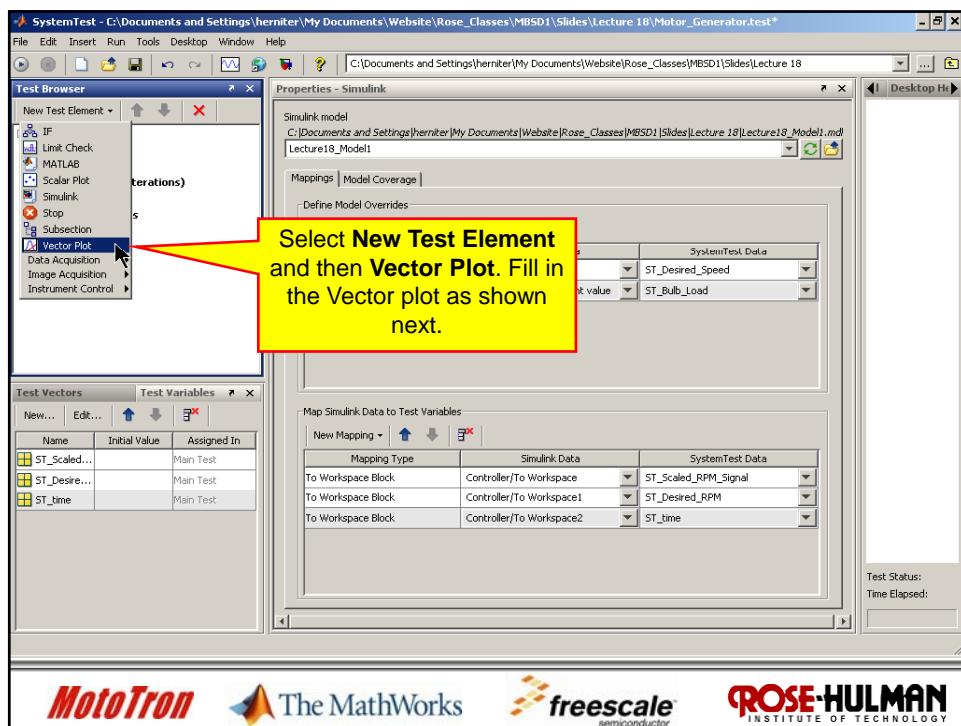
## Plots

- After each Simulink simulation, we want to plot the desired and measured response.
- First, make sure that Simulink is selected in the tree:





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

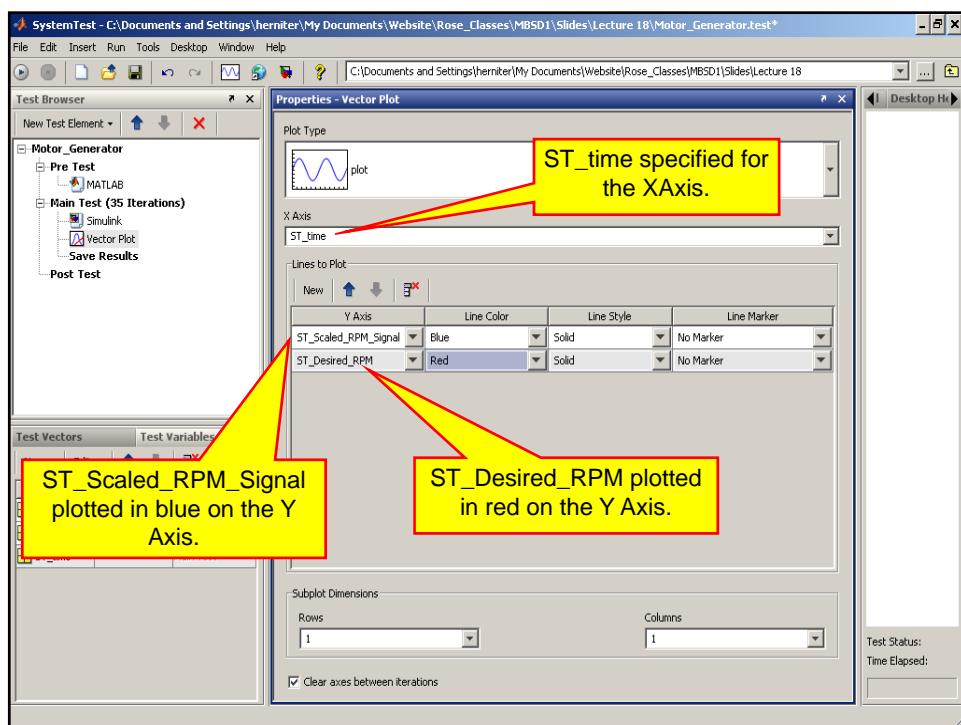


**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

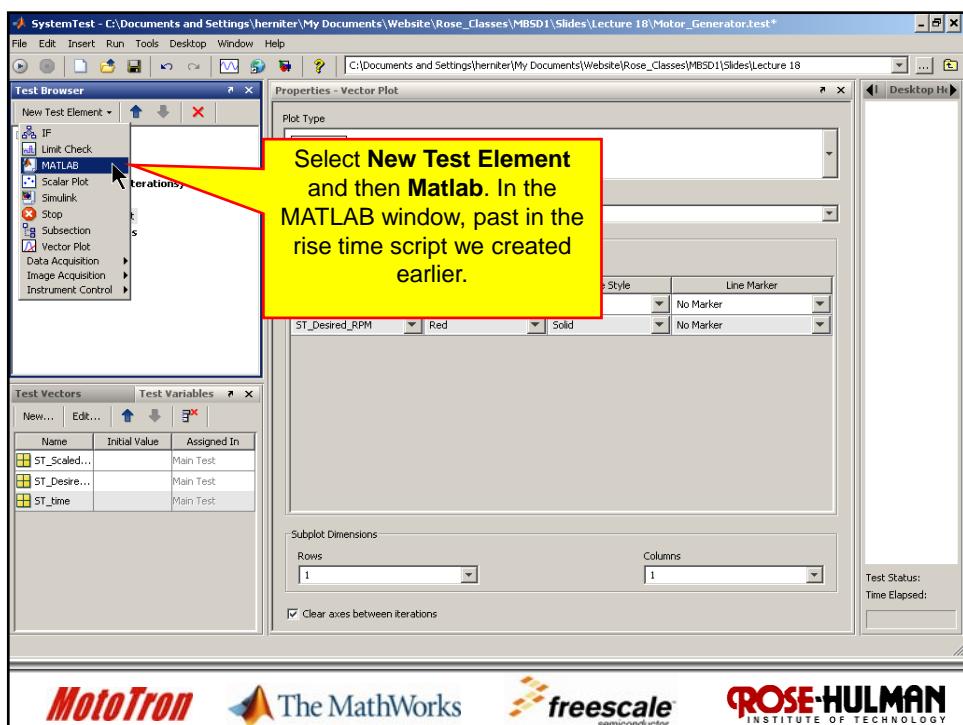
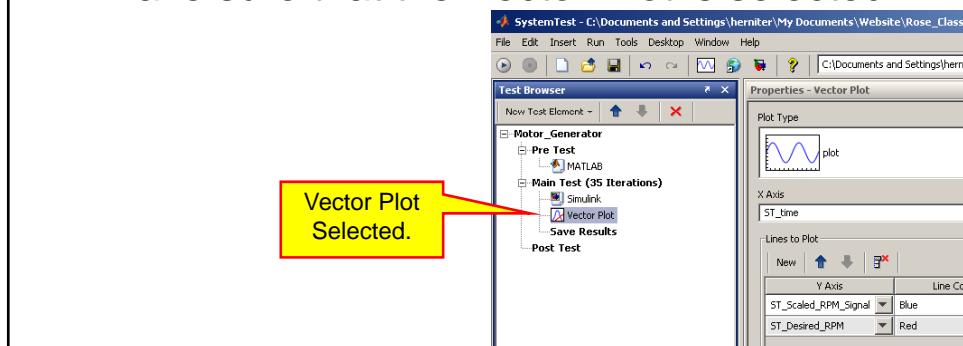




## Rise Time

61

- Next, we have to evaluate the data in the workspace and calculate the Rise Time.
- To do this we need to add a MATLAB script to the test plan.
- Make sure that the Vector Plot is selected.





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

SystemTest - C:\Documents and Settings\herniter\My Documents\Website\Rose\_Classes\MBSD1\Slides\Lecture 18\Motor\_Generator.test\*

File Edit Insert Run Tools Desktop Window Help

Test Browser

New Test Element ▾

- Motor\_Generator
  - Pre Test
  - Main Test (35 Iterations)
    - Simulink
    - Vector Plot
    - MATLAB
  - Save Results
  - Post Test

Properties - MATLAB

MATLAB Script

```

22
23 %Find the time of the 90% Point.
24
25 if (Scaled_RPM_Signal (i) <= Ninety_Pct_Value) &&...
26     (Scaled_RPM_Signal (i+1) >= Ninety_Pct_Value)
27     Ninety_Pct_Time=time(i);
28 end
29 end
30
31 % Calculate the rise time as the difference between the
32 % 10% and 90% points. If a rise time is not found, set the
33 % Rise_Time to infinity.
34
35 if (Ten_Pct_Time > 0) && (Ninety_Pct_Time > 0)
36     Rise_Time = Ninety_Pct_Time-Ten_Pct_Time;
37 else
38     Rise_Time = inf;
39 end
40
41
42
43

```

Script to calculate the Rise Time.

Test Status:  
Time Elapsed:

SystemTest - C:\Documents and Settings\herniter\My Documents\Website\Rose\_Classes\MBSD1\Slides\Lecture 18\Motor\_Generator.test\*

File Edit Insert Run Tools Desktop Window Help

Test Browser

New Test Element ▾

- Motor\_Generator
  - Pre Test
  - Main Test (35 Iterations)
    - Simulink
    - Vector Plot
    - MATLAB
    - Limit Check
  - Save Results
  - Post Test

Properties

MATLAB Script

```

1 %Find the rise time.
2
3 %Calculate the Rise Time
4
5
6 % Define the rise time as the amount of time
7 % output got go from 10% of the final value to
8 % of the final value.
9
10 Ten_Pct_Value = 0.1*ST_Desired_Speed;
11 Ninety_Pct_Value = 0.9*ST_Desired_Speed;
12 Ten_Pct_Time = 0;
13 Ninety_Pct_Time = 0;
14
15
16 %Find the time of the 10% Point.
17 for i = 1:(length(time)-1)
18     if (Scaled_RPM_Signal (i) <= Ten_Pct_Value) &&...
19         (Scaled_RPM_Signal (i+1) >= Ten_Pct_Value)
20         Ten_Pct_Time=time(i);
21     end

```

Change these two lines because the magnitude of our step input is specified by the test vector ST\_Desired\_Speed.

Plots - Motor\_Generator: Vector Plot: 06-Feb-2008 09:27:02

Getting Started Desktop Help

ed

Feb-2008 16:59 Feb-2008 16:24

Test Status: Failed Time Elapsed: 00:01:24

100 %

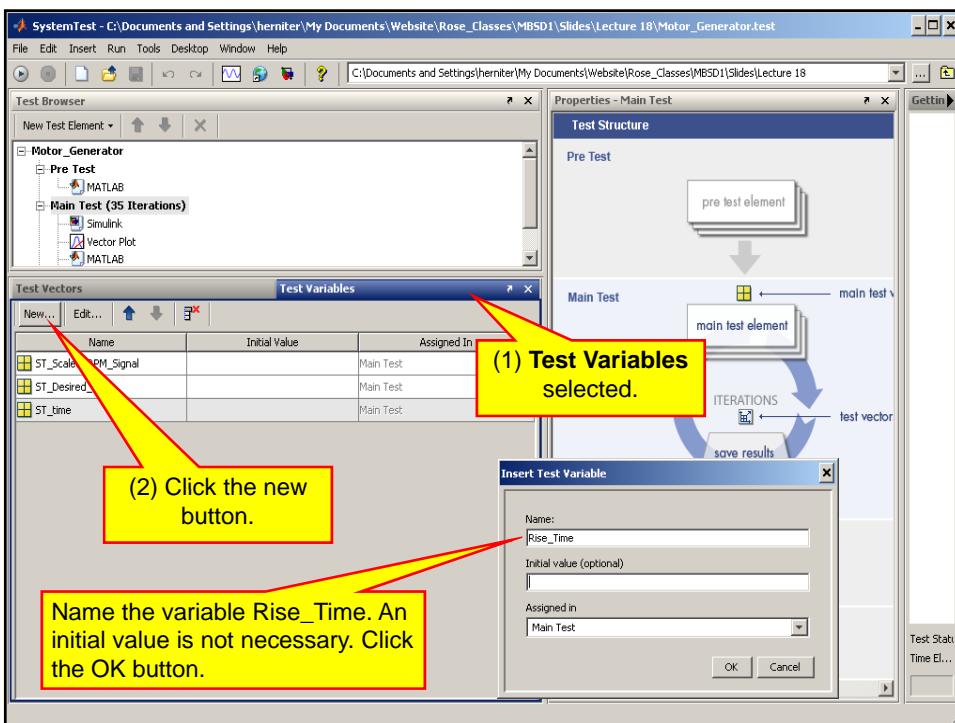
MotoTron The MathWorks freescale ROSE-HULMAN INSTITUTE OF TECHNOLOGY



65

## Rise Time

- We are going to use the value of Rise\_Time calculated in the script to evaluate against a constraint in System\_Test.
- We will need to define Rise\_Time as a Test Variable.

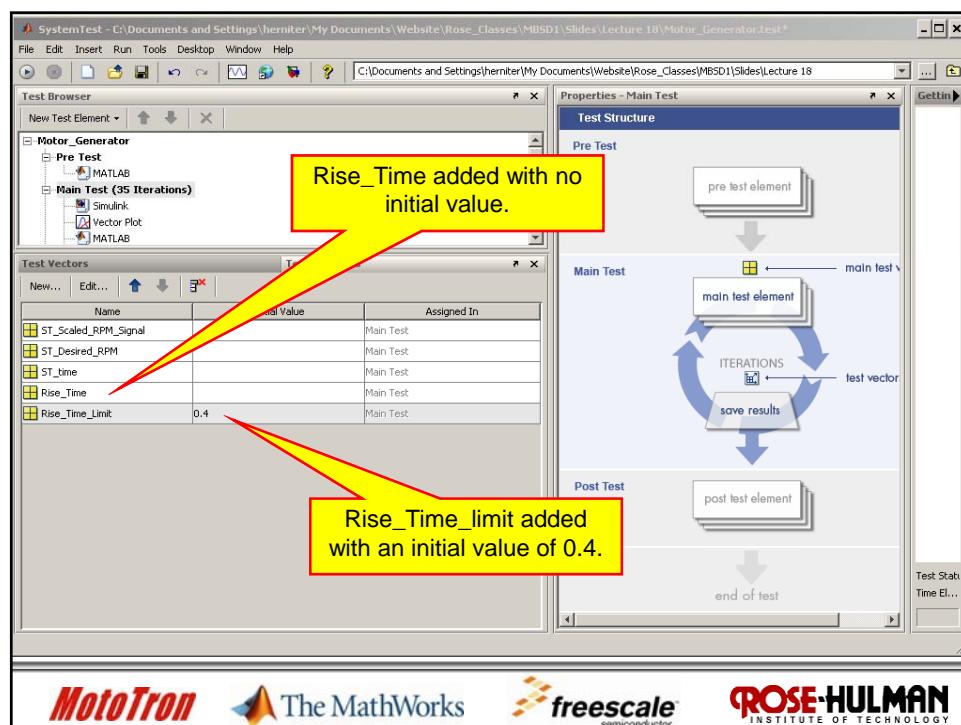




67

## Test Variables

- Repeat the process to add a variable called Rise\_Time\_limit.
- Give this variable an Initial Value of 0.4.
- This variable will be used as a constraint where we will check to see that the measured rise time is less than or equal to this limit.





69

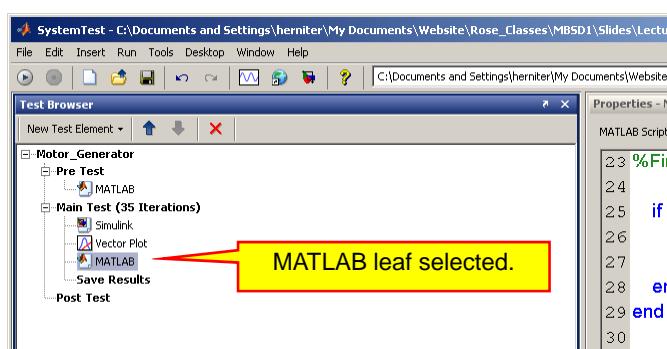
## Constraints

- Next we need to add our constraints to verify the performance of our system.
- We can add several constraints and then measure what proportion of our tests pass the constraints.
- For this example, we will only have a single constraint.

70

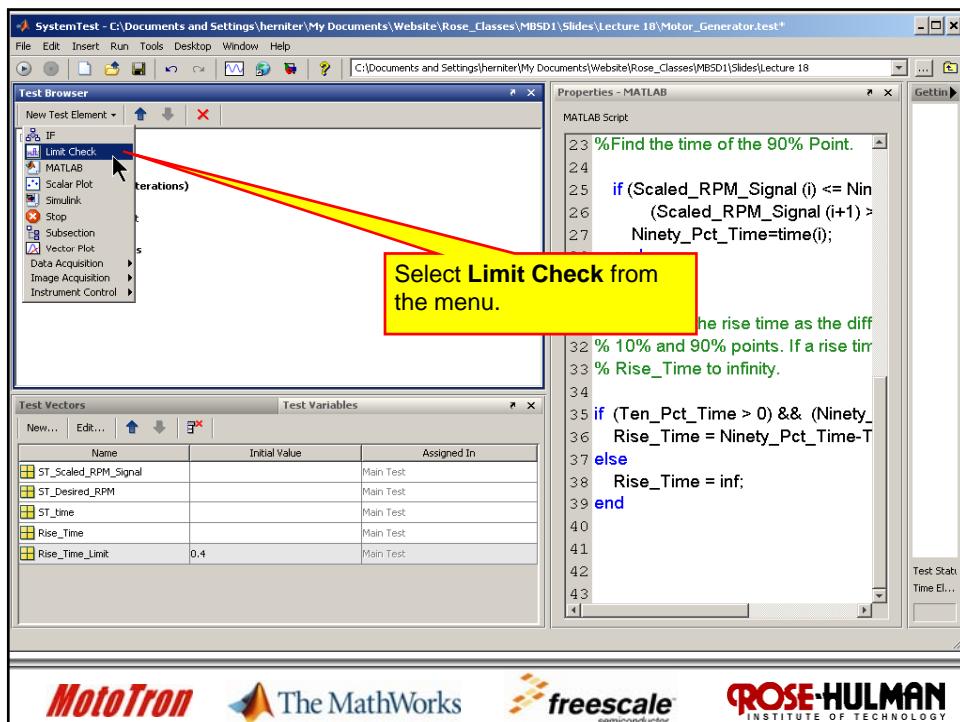
## Constraints

- We would like to check the constraint after we run the MATLAB script that calculates the rise time.
- Select MATLAB from the tree:





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

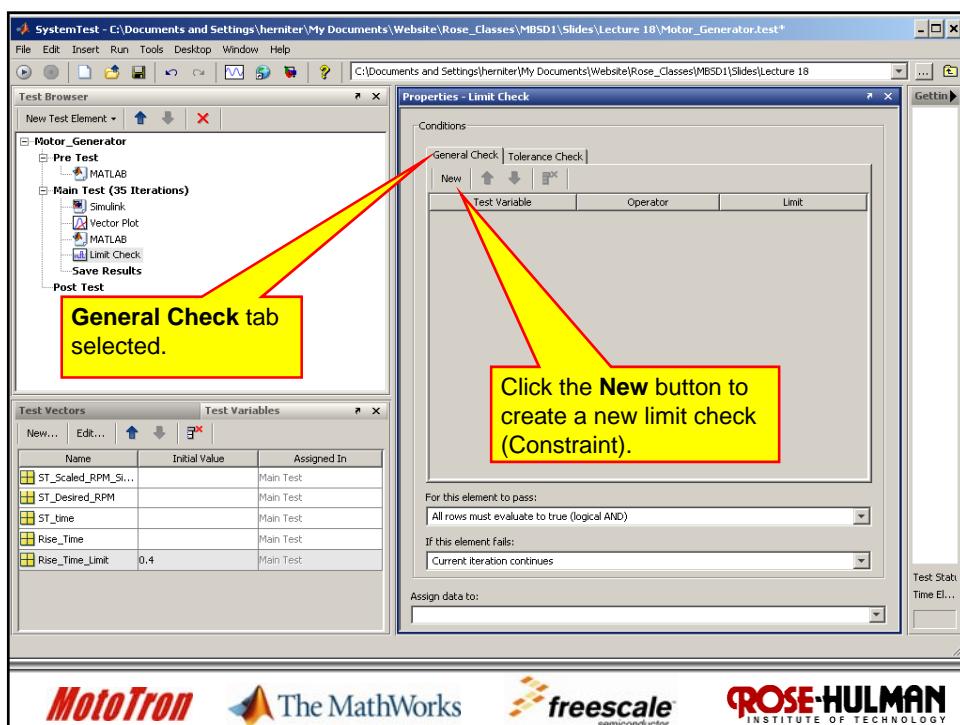


**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



**MotoTron**

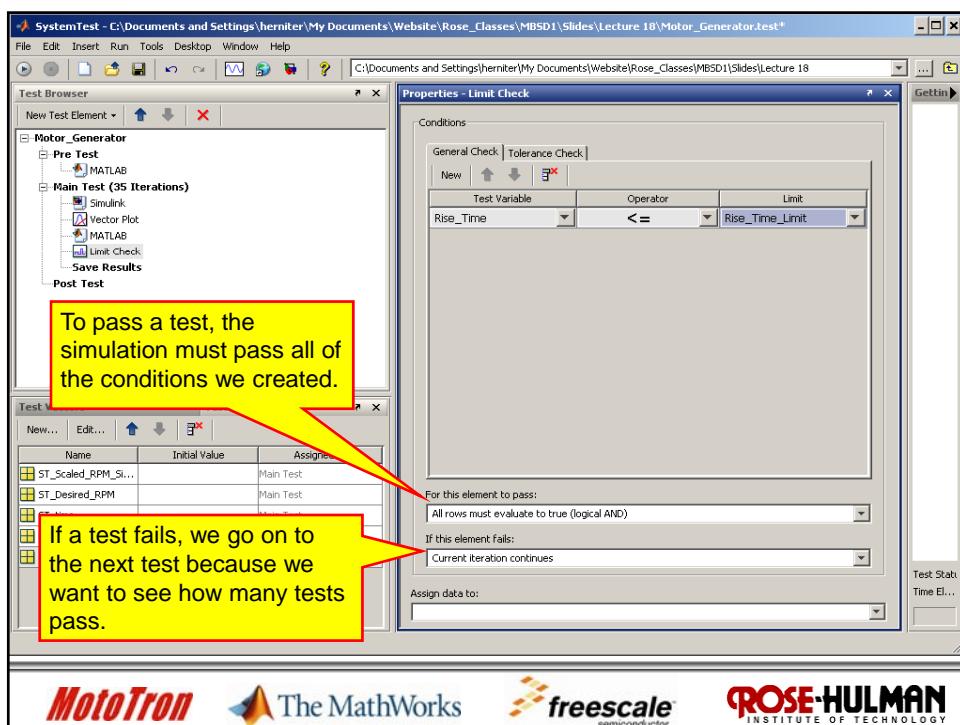
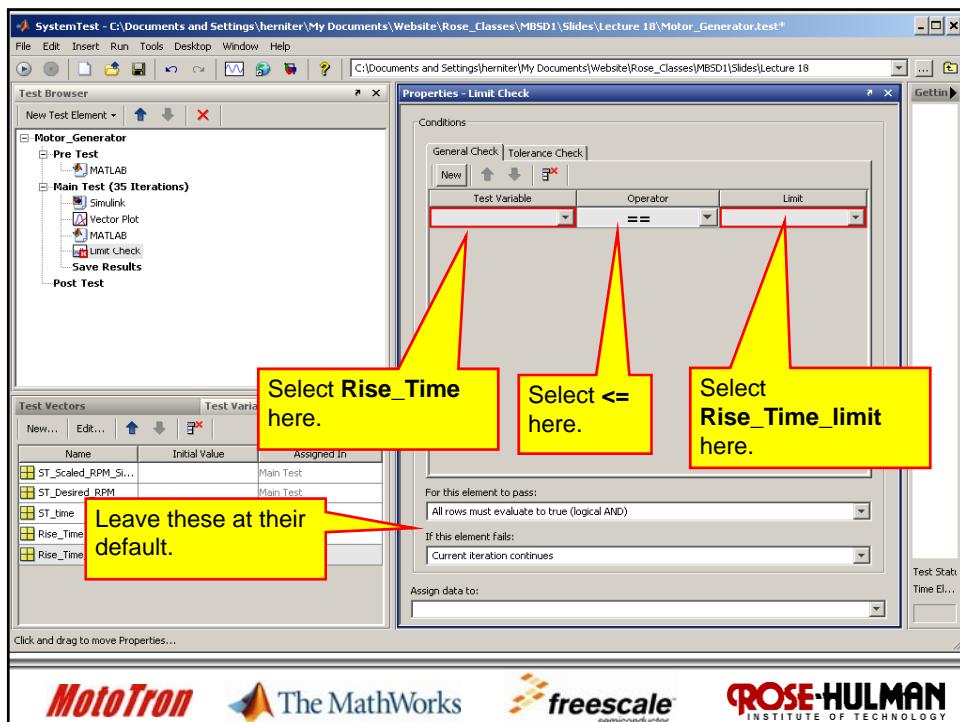
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



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



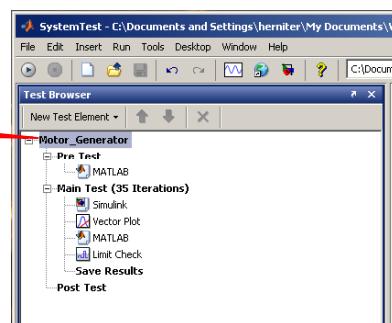


75

## After Test Runs

- We are now finished setting up our test.
- We would like to create a report that we can view after all 35 tests run.
- Select the Motor\_Generator leaf.

Motor\_Generator leaf selected.



**MotoTron**

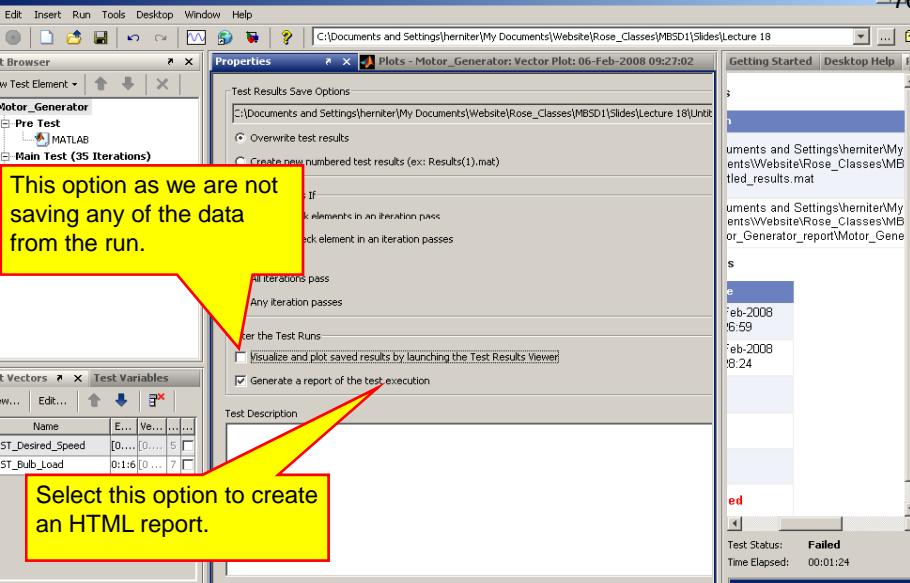
**The MathWorks**

**freescale**  
semiconductor

**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY

76

SystemTest - C:\Documents and Settings\herniter\My Documents\Website\Rose\_Classes\MBSD1\Slides\Lecture 18\Motor\_Generator.test\*



**MotoTron**

**The MathWorks**

**freescale**  
semiconductor

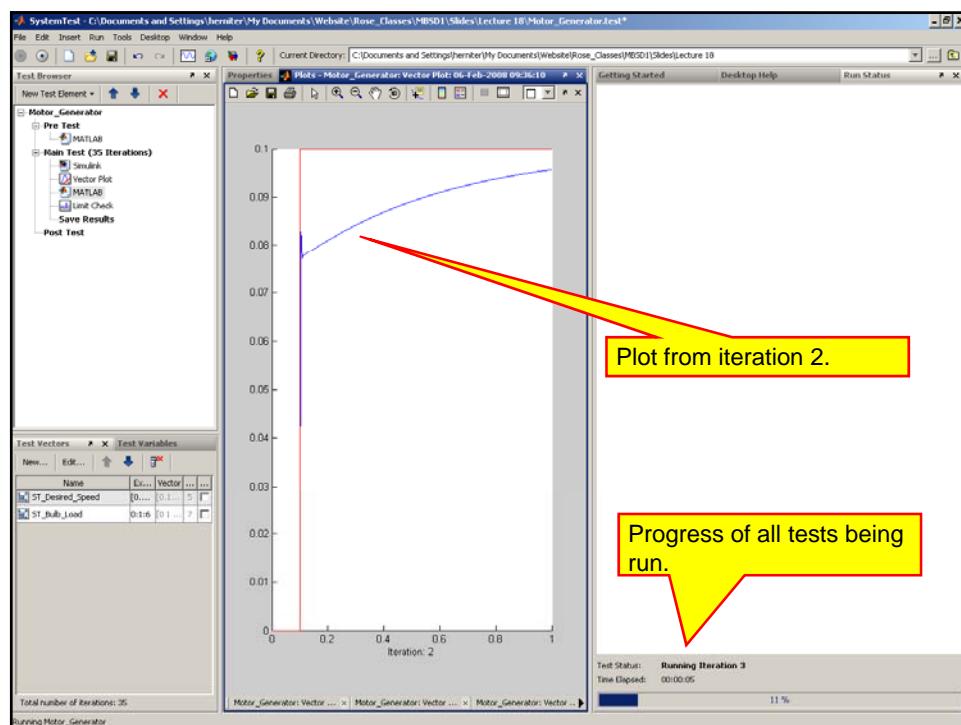
**ROSE-HULMAN**  
INSTITUTE OF TECHNOLOGY



77

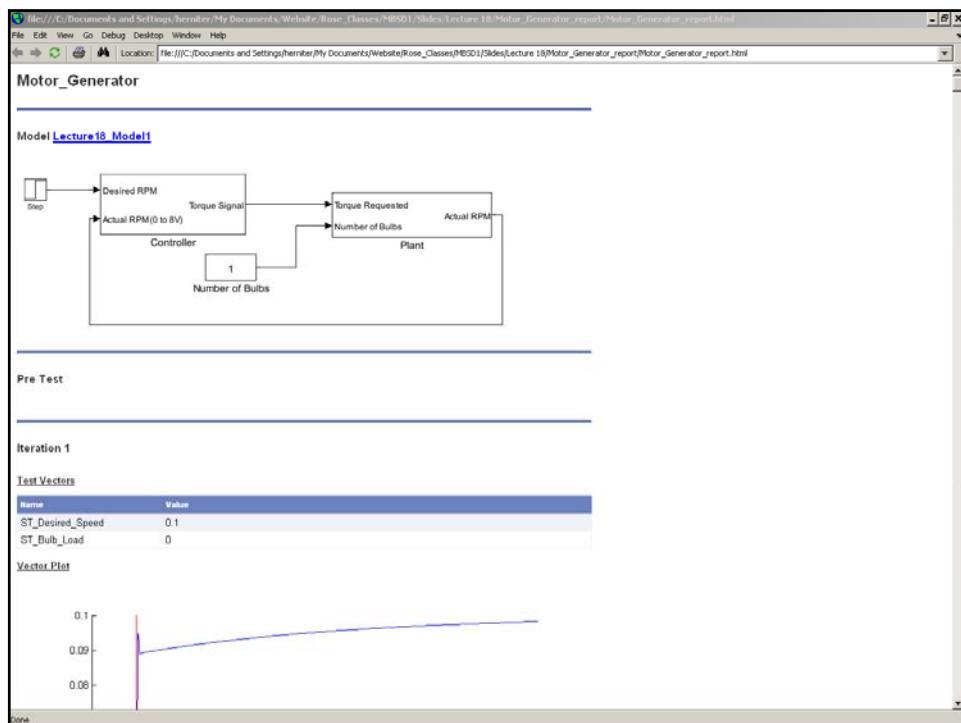
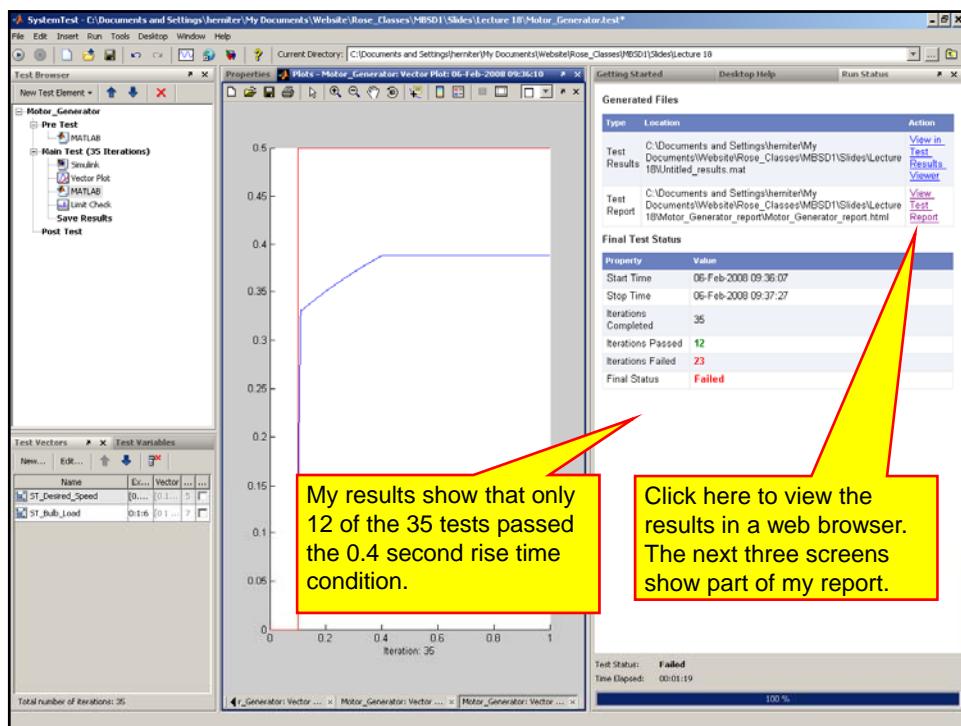
## Running the Test

- Select Run from the SystemTest menus or click the Play  button.
- A plot should display after each test showing the desired and measured step response.



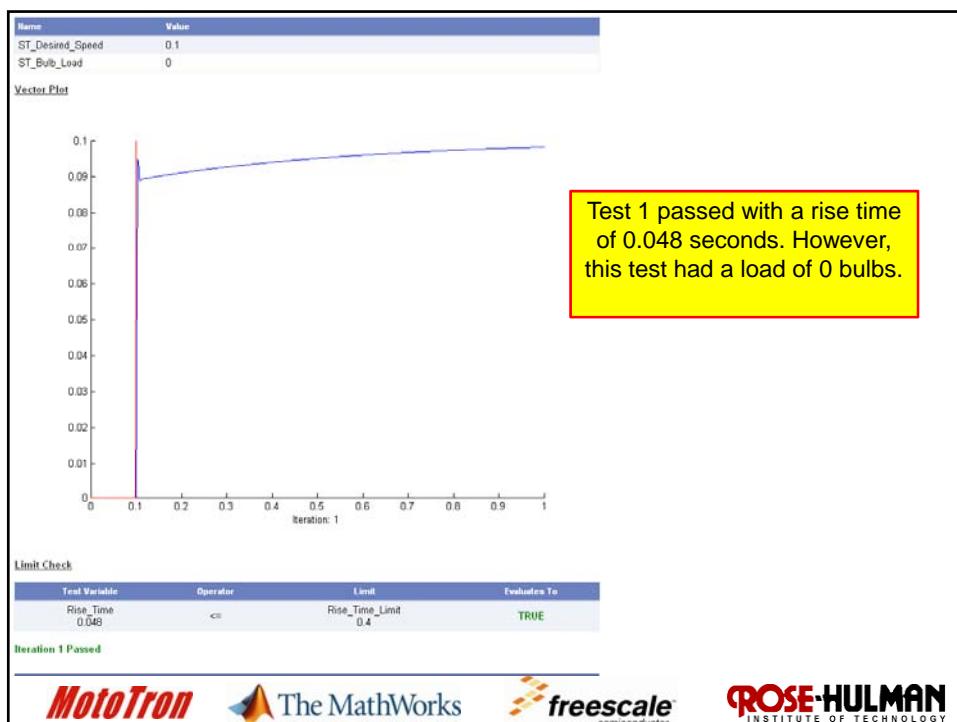


Except where otherwise noted, this work is licensed under  
<http://creativecommons.org/licenses/by/3.0/>





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

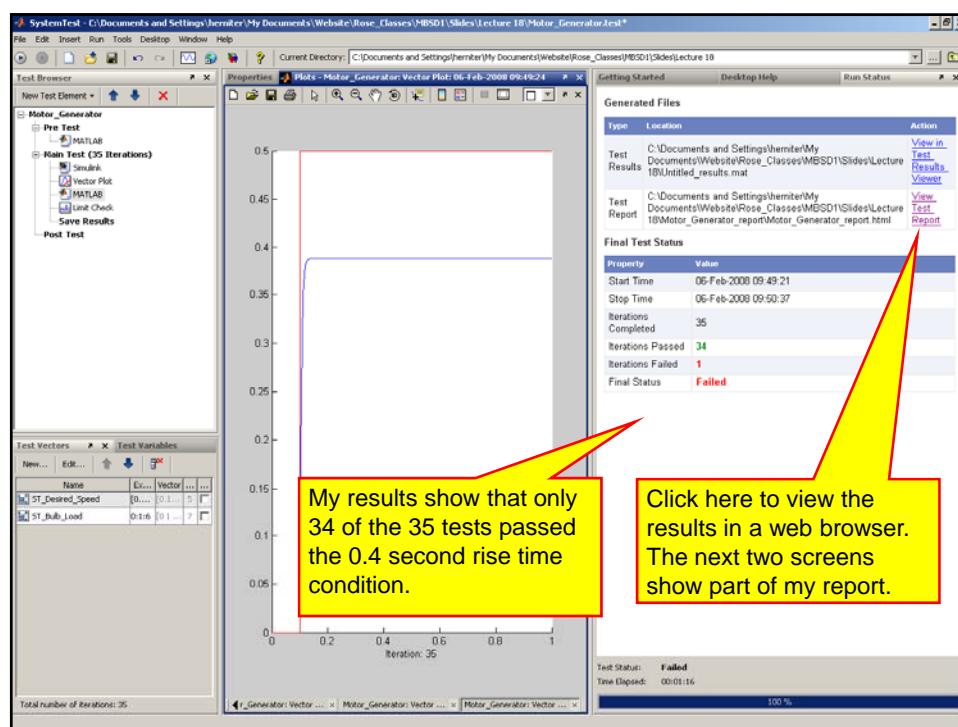




83

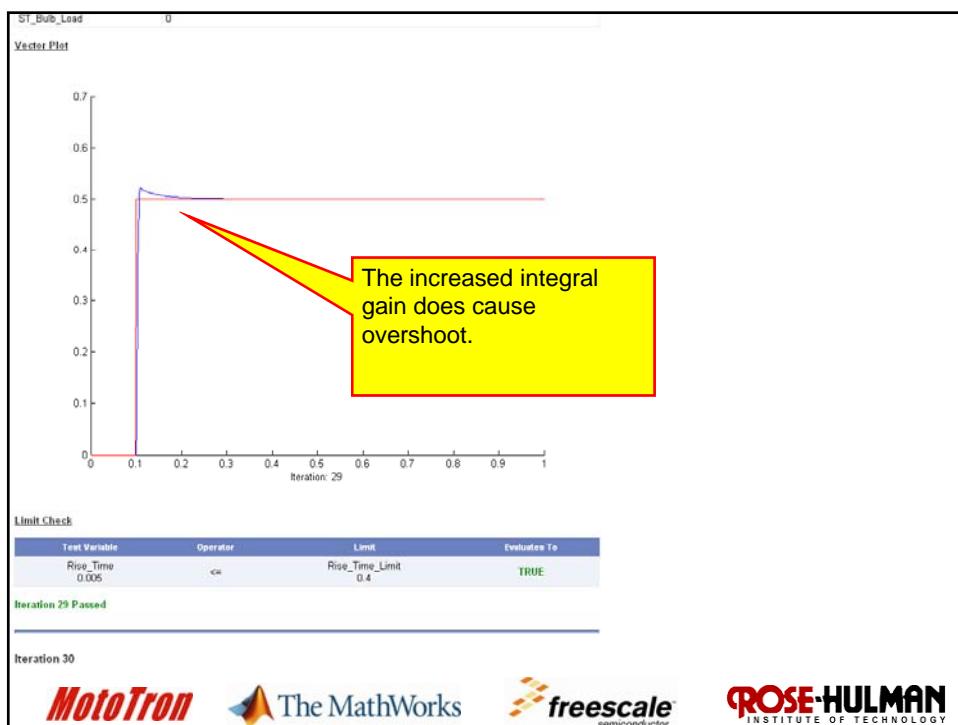
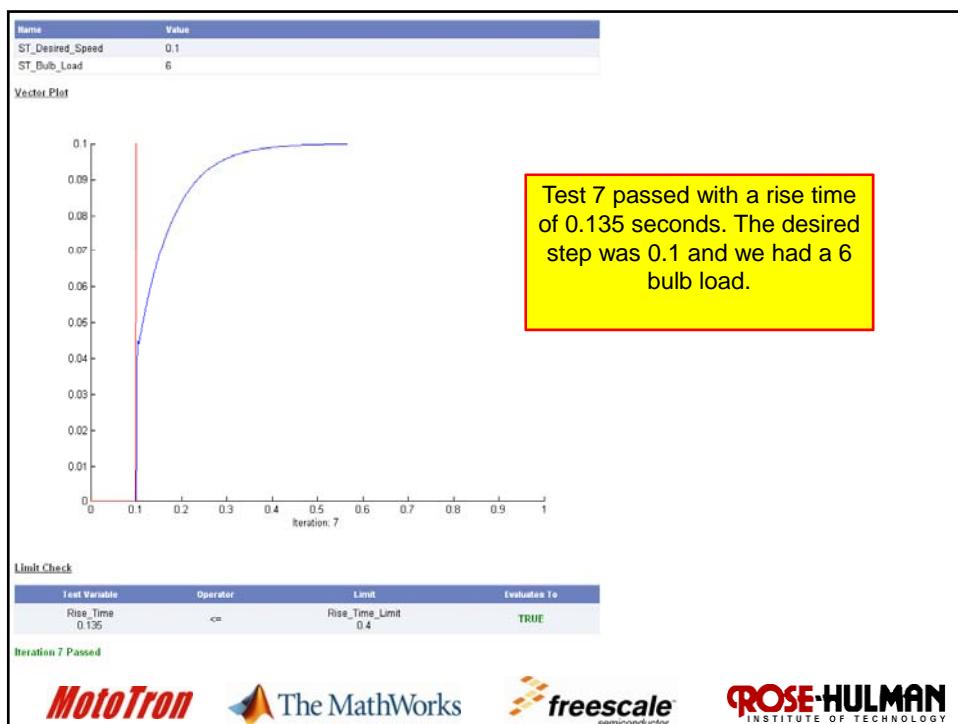
## New Test

- The controller model in the previous test had a proportional gain of 5 and an integral gain of 10.
- Lets try increasing the integral gain to reduce the rise time.
- Run the test again and measure the results.





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





87

## Lecture 18 Demo 1

- We notice that we now have overshoot.
- Add a second test that verifies that the overshoot is less than 5%.
- You need to check to see how many tests pass both the rise time specification and the overshoot specification.

Demo\_\_\_\_\_



88

## Lecture 18 Exercise 1

- Verify summary of results showing that 34 of 35 runs passed the specification.

Demo\_\_\_\_\_





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

## Lecture 18 Project 3

89

- Start Project 3.



The MathWorks

