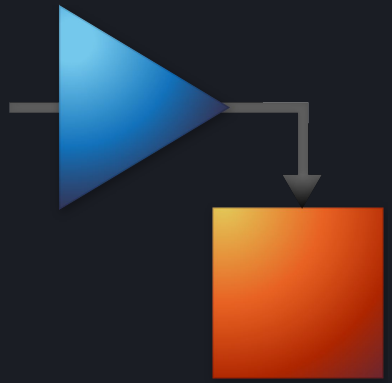
Real World Application

Session Content

- Subsystems
- Masking Fundamentals
- Merge Block, Lookup table
- Data Inspector
- Applications



Real World Application

Subsystems

why Subsystem ?

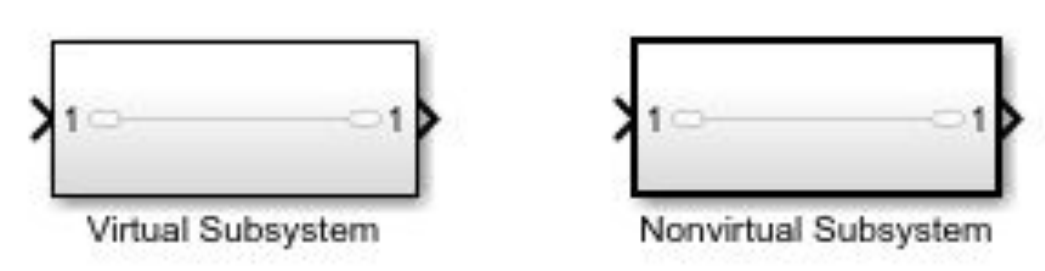
- Organizing a model that grows in size and complexity can be achieved by grouping blocks into subsystems.
- Subsystems consist of a collection of blocks encapsulated within a single Subsystem block.
- Utilizing subsystems creates a hierarchical block diagram structure, with the Subsystem block representing one layer and the blocks within the subsystem residing on another layer.
  Grouping functionally related blocks together within a subsystem promotes coherence and clarity in the model design.
- Subsystems help streamline the model's appearance by reducing the number of blocks visible in the main model window.
- Each subsystem establishes an interface with defined inputs and outputs, facilitating modular design and integration within the larger system.

Real World Application

Subsystems

A subsystem can be virtual and non-virtual:

- A virtual subsystem provides graphical hierarchy in a model.
- A non-virtual subsystem provides graphical hierarchy and executes as a unit within a model.



Real World Application

Subsystems

What is a subsystem:

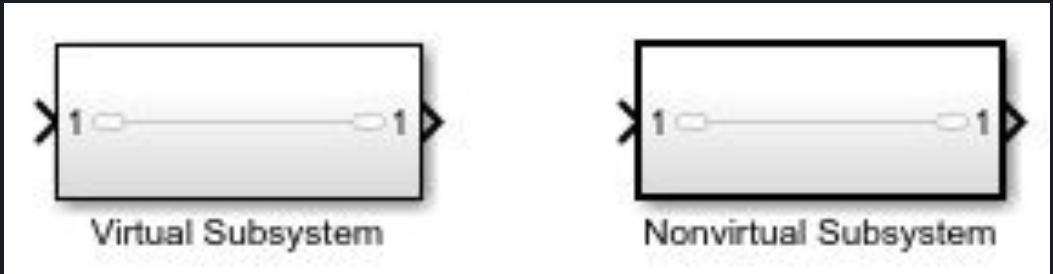
- Subsystems, encapsulated within a Subsystem block, consist of a group of blocks within a model or system.
- Subsystem blocks can represent either virtual or nonvirtual subsystems, serving different purposes within the model.
- Virtual subsystems contribute to the visual organization of a block diagram without directly influencing simulation outcomes.
- Unlike nonvirtual subsystems, virtual subsystems do not actively participate in the simulation process and are not executed conditionally or atomically.
- Additionally, virtual subsystems lack checksums, distinguishing them from nonvirtual ones.
- On the other hand, nonvirtual subsystems actively contribute to the simulation and can impact model behavior when added or removed.
- Nonvirtual subsystems function as atomic units within the parent model, executing as a single block during simulation.

Real World Application

Subsystems

How you know the type of subsystem:

- Check the border of the block. A block that represents a virtual subsystem has a thin border. A block that represents a nonvirtual subsystem has a thick border.

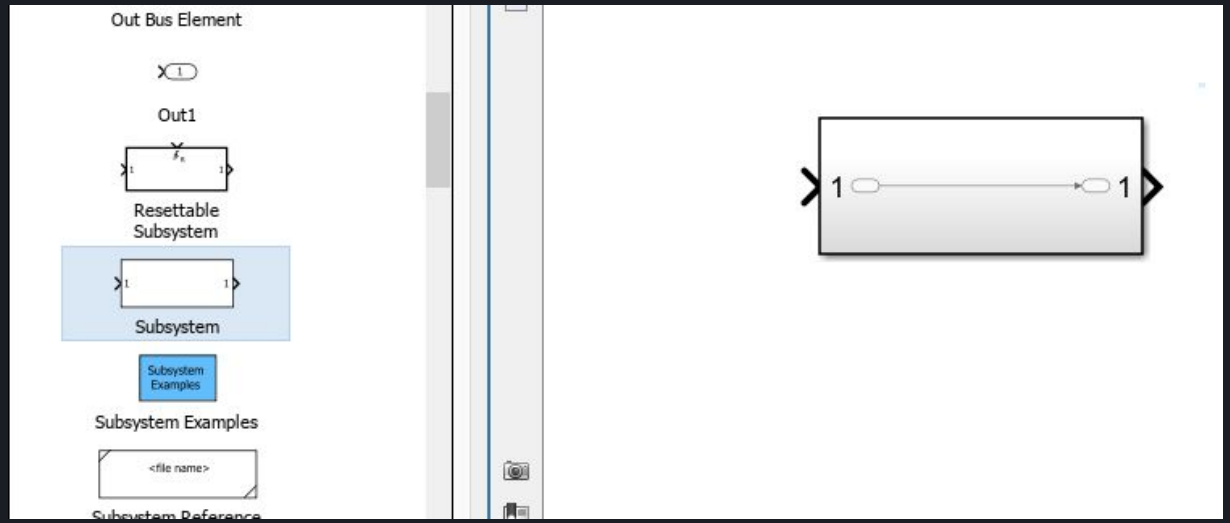


Real World Application

Subsystems

How to create subsystem:

- You can create a subsystem by adding a Subsystem block, then adding contents to the subsystem. Insert a Subsystem block into your model.

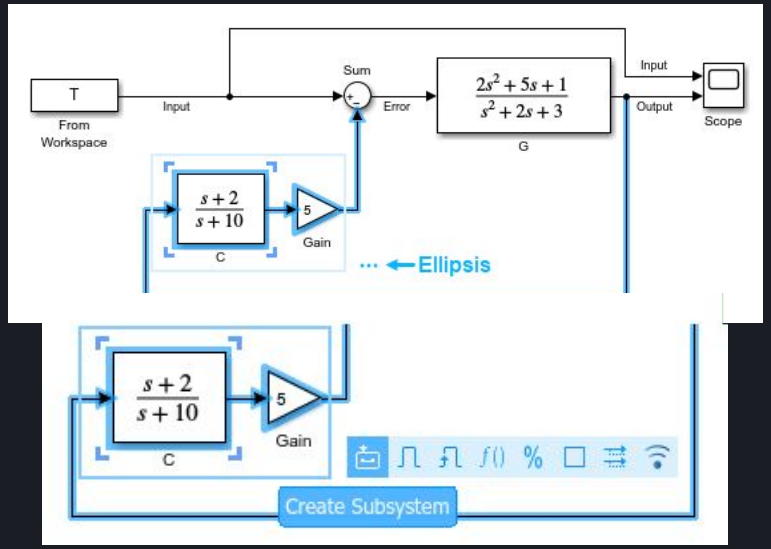


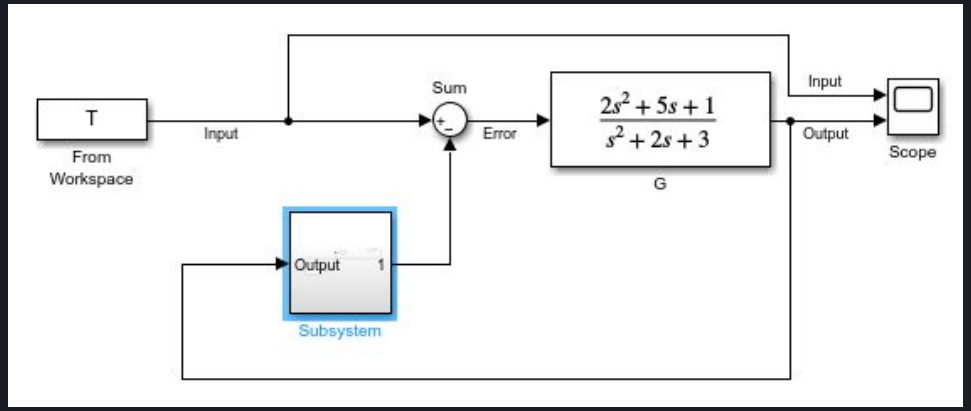
Real World Application

Subsystems

How to create subsystem:

- You can create a subsystem by converting part of an existing model into a subsystem.



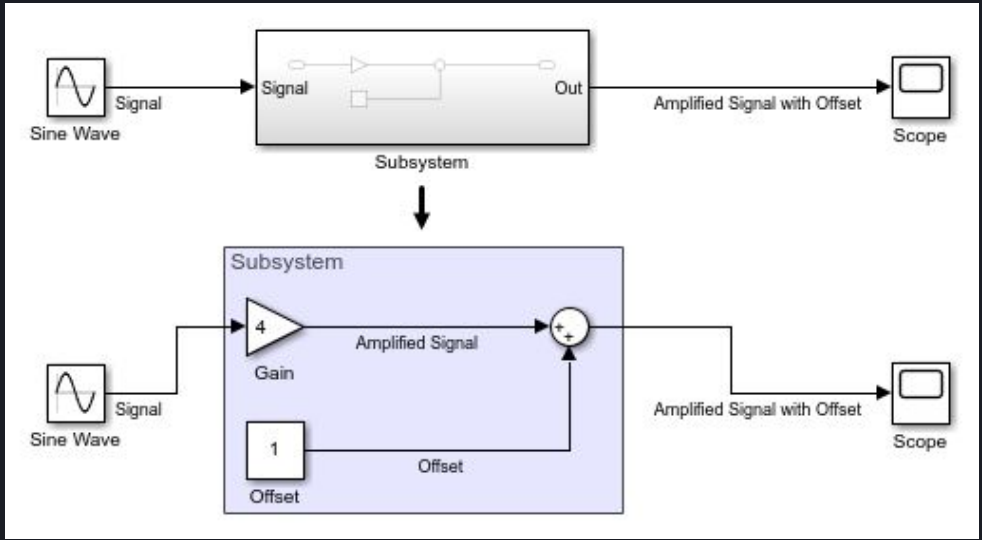


Real World Application

Subsystems

Replace Subsystem with its contents:

- Select the Subsystem block. Then, in the Simulink Toolstrip, on the **Subsystem Block** tab, click **Expand**.

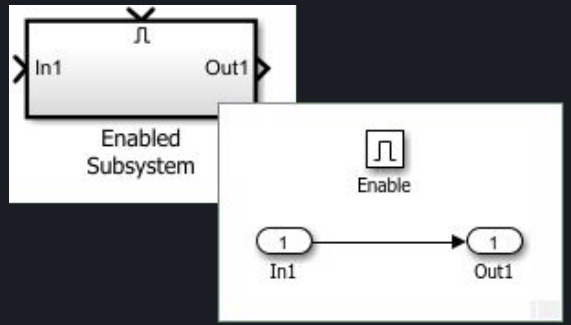


Real World Application

Subsystems

Enable Subsystem:

- Subsystems whose execution is enabled by external input.
- An Enabled Subsystem block refers to a Subsystem block that operates in response to an external signal with a positive value.
- Execution of an enabled subsystem occurs when the signal received at the Enable port transitions from negative to positive.
- To configure an Enabled Subsystem block, an Enable block is positioned within a Subsystem block.
- The Enable port of the Enabled Subsystem block regulates the execution of the enabled subsystem, determining when it becomes active.

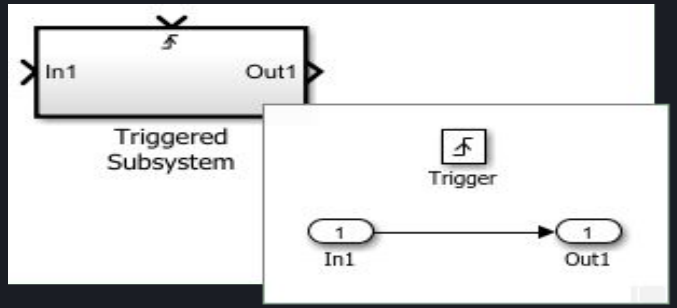


Real World Application

Subsystems

Triggered Subsystem:

- The Triggered Subsystem block serves as a template Subsystem block designed to initiate execution whenever the control signal reaches a specified trigger value.
- Triggered Subsystem blocks are commonly employed to represent tasks that activate upon detecting a trigger value, such as:
  - A task triggered by a specific event or condition.
  - An interruption triggered by input/output (I/O) hardware.
  - A processor request triggered by an exception or error handling.
- By utilizing Triggered Subsystem blocks, you can effectively model dynamic processes that respond to specific trigger events or conditions within a system.

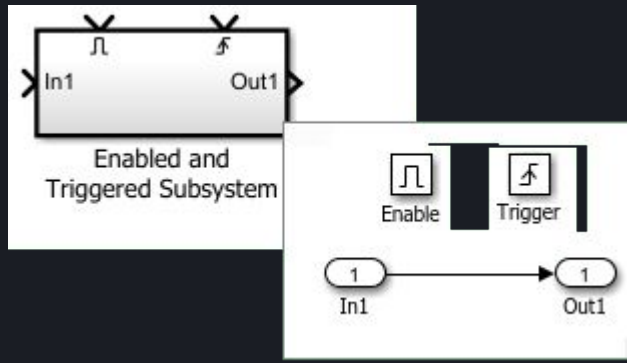


Real World Application

Subsystems

Enable and Triggered Subsystem:

- The Enabled and Triggered Subsystem block is a predefined Subsystem block intended as a foundation for constructing a subsystem that activates under two specific conditions:
  - When the enable control signal possesses a positive value.
  - When the trigger control signal reaches a trigger value.
- Enabled and Triggered Subsystem blocks are utilized to represent scenarios where functionality is initiated based on the presence of both enable and trigger signals.
- These blocks are particularly useful for modeling:
  - Optional functionalities that are only activated when both conditions are met.
  - Alternative functionalities that are triggered by specific combinations of enable and trigger signals.



Real World Application

Subsystems

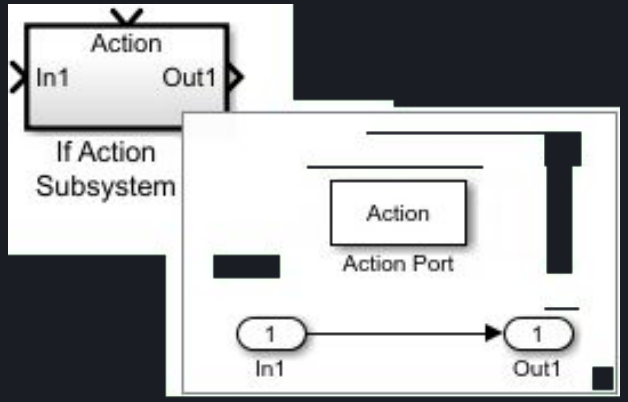
If action Subsystem:

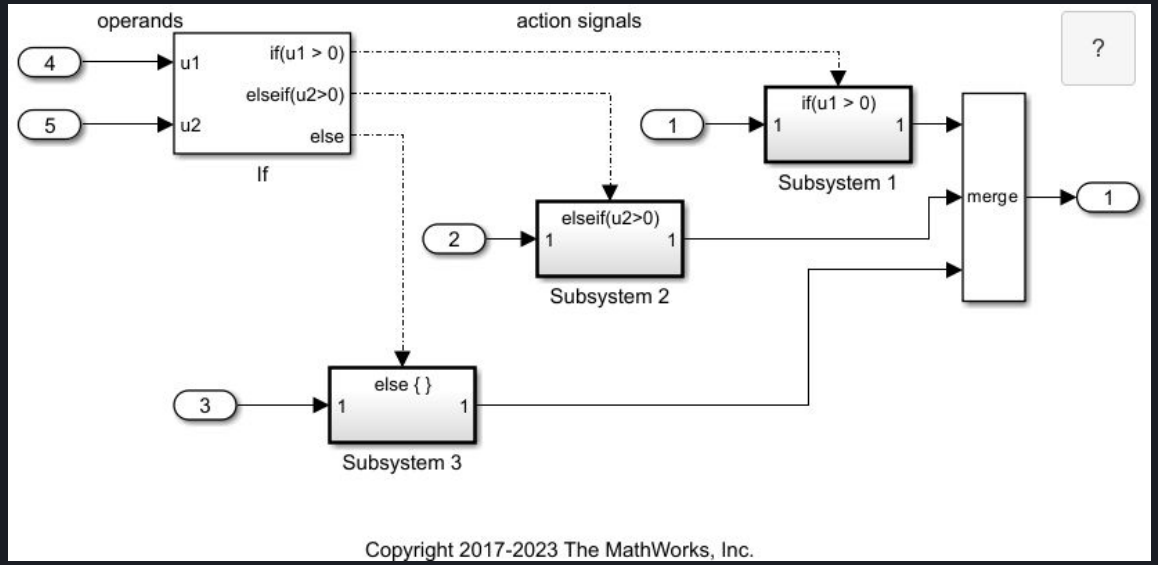
- The If Action Subsystem block serves as a preconfigured Subsystem block for creating a subsystem that operates under the control of an If block.
- The If block assesses a logical expression and, based on the evaluation outcome, generates an action signal that dictates the execution of the subsystem.
- Within the subsystem, an Action Port block is placed to regulate its execution according to the action signal produced by the If block.
- The If block facilitates the selection of an If Action Subsystem block from a group of subsystems, each representing a potential action outcome.
- Regardless of the chosen subsystem, a Merge block can be employed to consolidate the output signals into a single output.
- It's imperative that all blocks within an If Action Subsystem execute at the same rate as the controlling If block.
- To adhere to this requirement, the sample time parameter for each block should be configured to either inherit the sample time (-1) or match the sample time of the If block.

Real World Application

Subsystems

If action Subsystem:





Real World Application

Subsystems

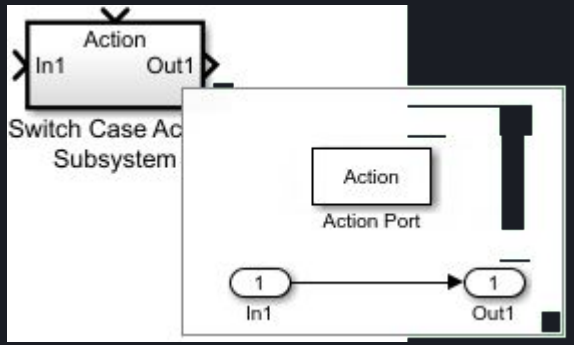
switch case action Subsystem:

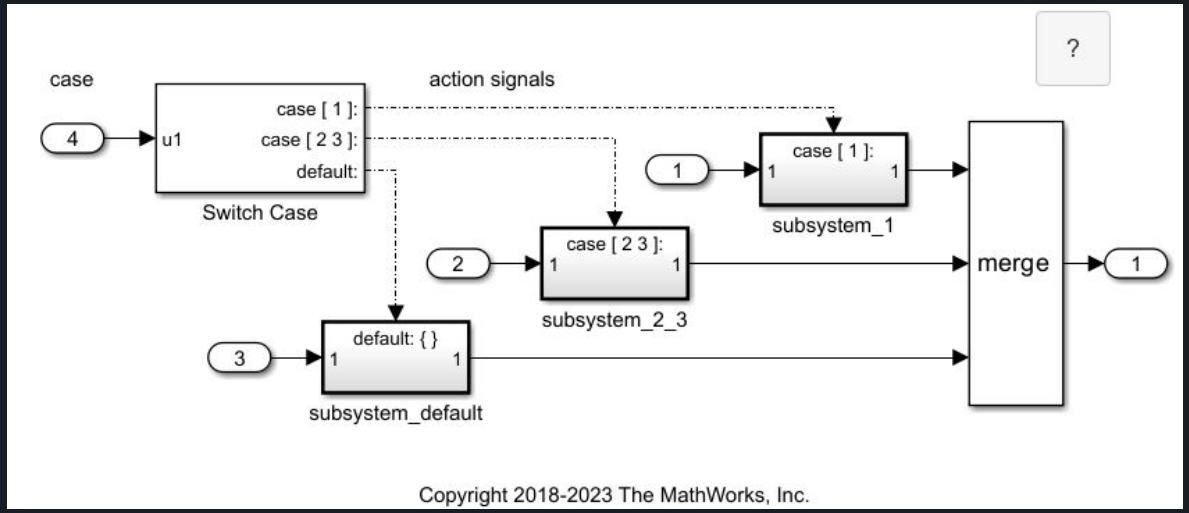
- The Switch Case Action Subsystem block is a predefined Subsystem block used to create a subsystem whose operation is governed by a Switch Case block.
- A Switch Case block accepts an input port that determines which case to select based on defined conditions specified in the Case conditions parameter.
- Depending on the input value and the chosen case, an action signal is transmitted to initiate the execution of a Switch Case Action Subsystem block.
- Within the subsystem, an Action Port block is positioned to regulate the execution according to the received action signal.
- Any priority assigned to a Switch Case Action Subsystem block is disregarded by Simulink. Instead, prioritize the Switch Case block responsible for triggering the subsystem's execution.
- To ensure consistency, all blocks within a Switch Case Action Subsystem must operate at the same rate as the controlling Switch Case block.
- This requirement can be met by configuring the sample time parameter of each block to either inherit the sample time (-1) or match the sample time of the Switch Case block.
-

Real World Application

Subsystems

switch case action Subsystem:





Real World Application

Subsystems

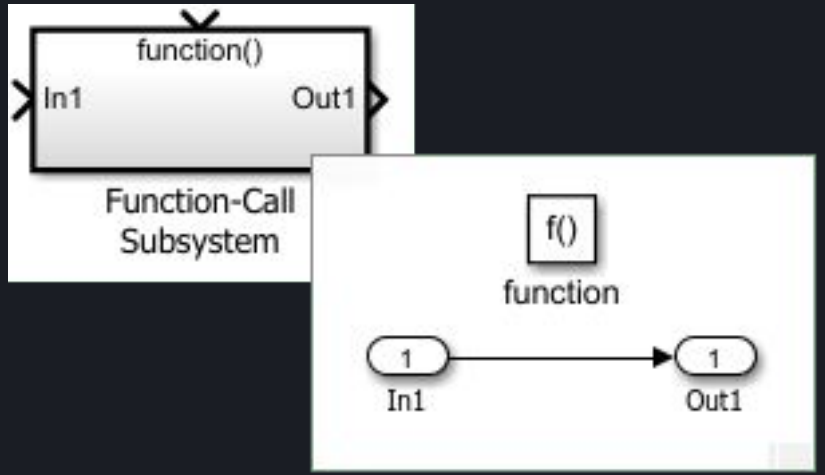
function call Subsystem:

- The Function-Call Subsystem block is a predefined Subsystem block designed to create a subsystem that triggers execution upon receiving a function-call event through its input port.
- Function-call events can be generated by various sources such as a Stateflow® chart, Function-Call Generator block, S-Function block, or Hit Crossing block.
- Function-Call Subsystem blocks are utilized for:
  - Orchestrating the sequence of execution for different components within the model.
  - Regulating the timing and frequency of execution for model components.
- Any priority assigned to a Function-Call Subsystem block is disregarded by Simulink®. However, priority can be assigned to a block connected to the function-call port of the subsystem.
- The function-call port of the subsystem can receive function-call events from diverse sources including a Stateflow chart, MATLAB Function block, Function-Call Generator block, S-Function block, or Hit Crossing block.

Real World Application

Subsystems

function call Subsystem:



Real World Application

Masking Fundamentals

Create block masks:

- Simulink allows you to create block masks, which are custom user interfaces for blocks.
- When you mask a block, you create a custom parameter dialog box specifically for that block. This dialog box includes its own block description, parameter prompts, and help texts.
- Masking a block helps encapsulate its block diagram, making it easier to manage and understand.
- You can use block masks to create independent custom blocks that can be reused as unique blocks within Simulink, similar to those predefined in the software.

Real World Application

Masking Fundamentals

What is the mask?

- A mask serves as a tailored interface for a block, concealing its internal content and presenting it as a unified entity with its own icon and parameter dialog box. This feature encapsulates the block's logic, offers managed access to its data, and streamlines the visual presentation of a model.

- When a block is masked, a specific mask definition is generated and stored alongside the block. This mask modification affects solely the block's interface, leaving its fundamental characteristics intact. By defining corresponding mask parameters within the mask, you can grant access to one or more underlying block parameters.

Real World Application

Masking Fundamentals

Mask Use cases:

- Display a meaningful icon on a block
- Provide a customized dialog box for the block
- Provide a dialog box that enables you to access only select parameters of the underlying blocks
- Provide users customized description that is specific to the masked block
- Initialize parameters using MATLAB® code

Real World Application

Masking Fundamentals

Example::

- Try to model equation of line

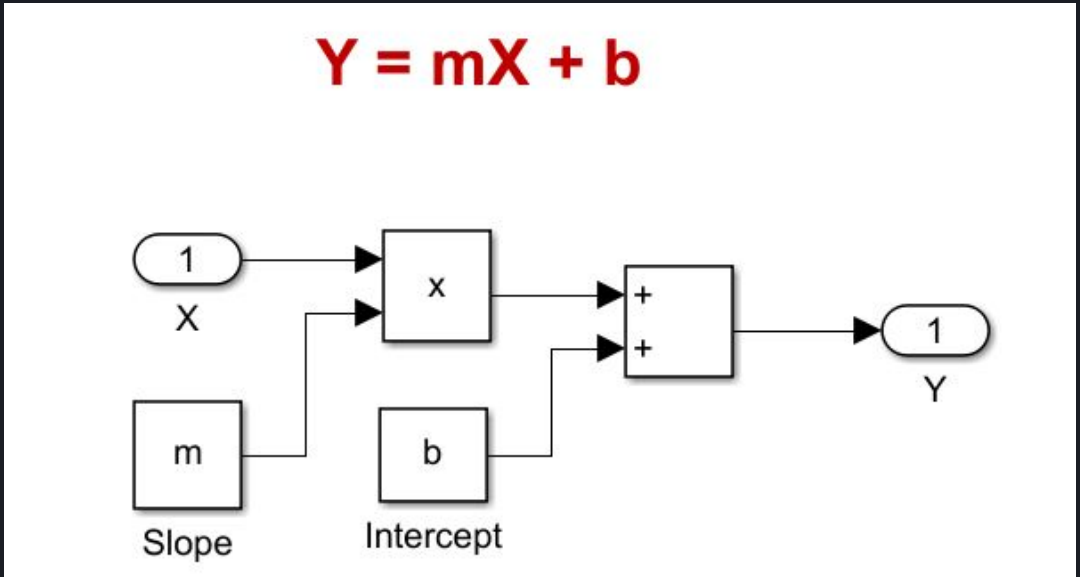
$$Y = mX + b$$

Real World Application

Masking Fundamentals

Example:

- Try to model equation of line

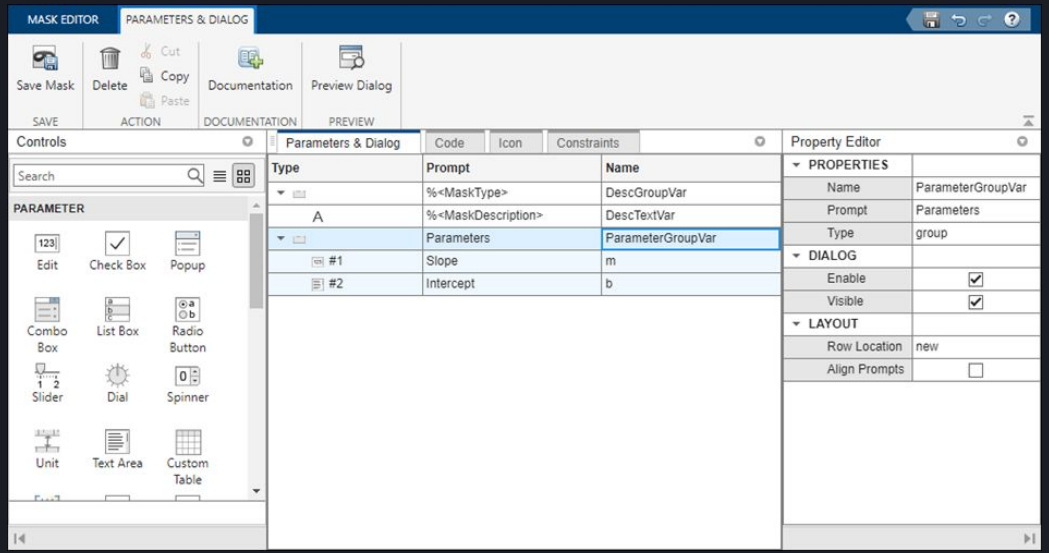


Real World Application

Masking Fundamentals

Example:

- Try to model equation of line

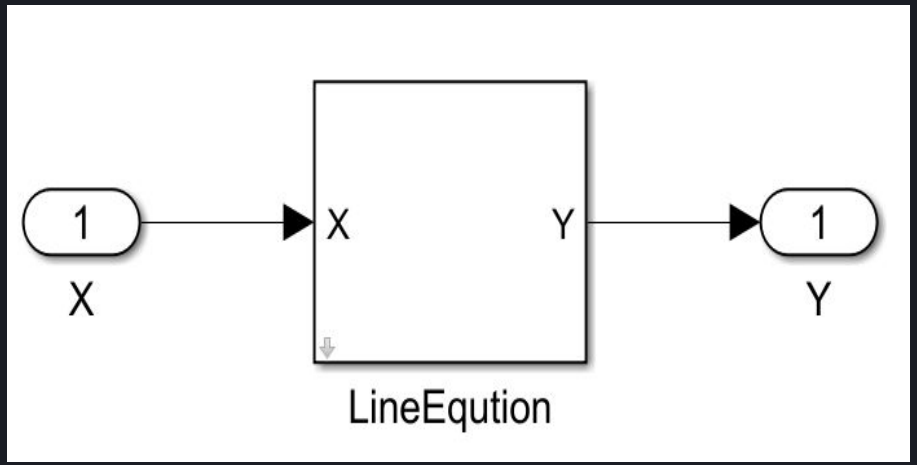


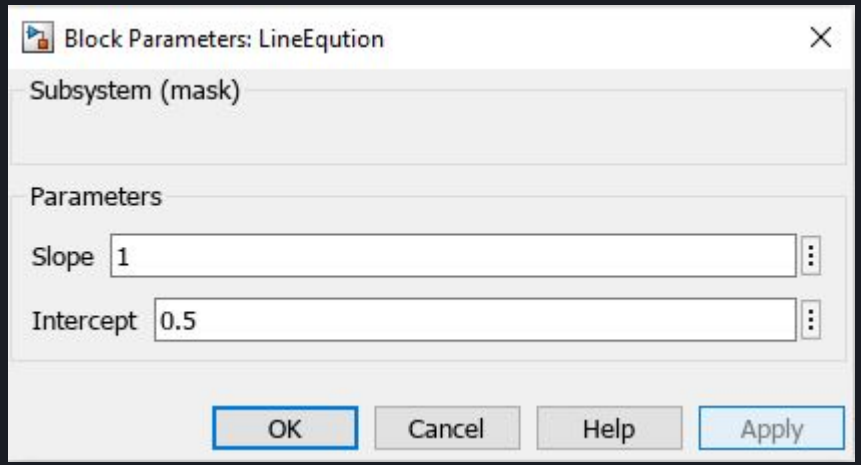
Real World Application

Masking Fundamentals

Example::

- Try to model equation of line



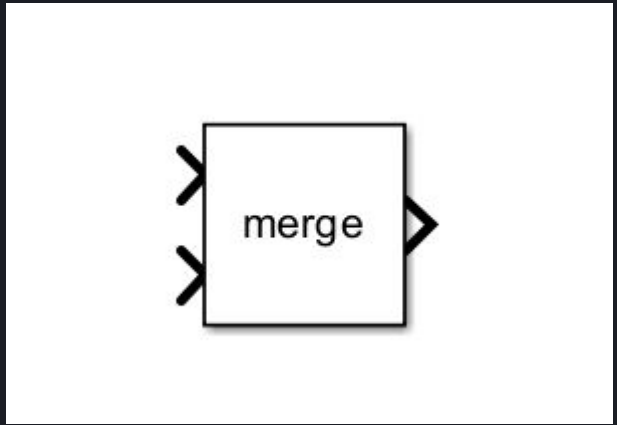


Real World Application

Masking Fundamentals

Merge Block

- In Simulink, the Merge block is used to combine multiple input signals into a single output signal. This block is particularly useful when you need to merge signals from different parts of your model into a unified stream of data. The Merge block is commonly used in scenarios where signals need to be aggregated before being processed further downstream in the model.

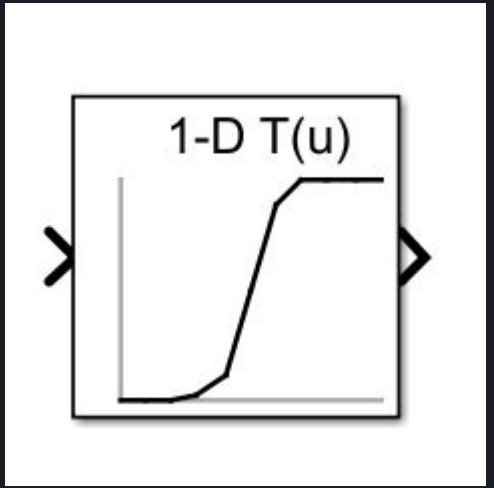


Real World Application

Masking Fundamentals

1D Lookup Table

- A 1D Lookup Table, also known as a 1-dimensional lookup table, is a block commonly used in Simulink for mapping input values to corresponding output values. It is essentially a mathematical function or table that associates a set of input values with specific output values.



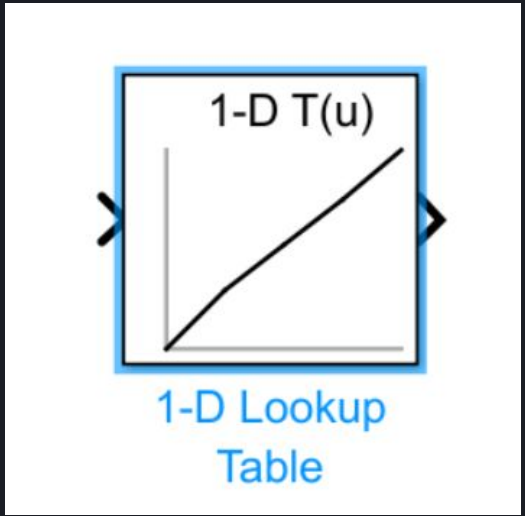
Real World Application

Masking Fundamentals

1D Lookup Table Example

Suppose you have a system where the output voltage of a sensor depends on the temperature. You have collected data showing the relationship between temperature and voltage:

- Temperature (°C): [0, 20, 40, 60, 80]
- Voltage (V): [1.0, 2.5, 3.7, 4.9, 6.2]

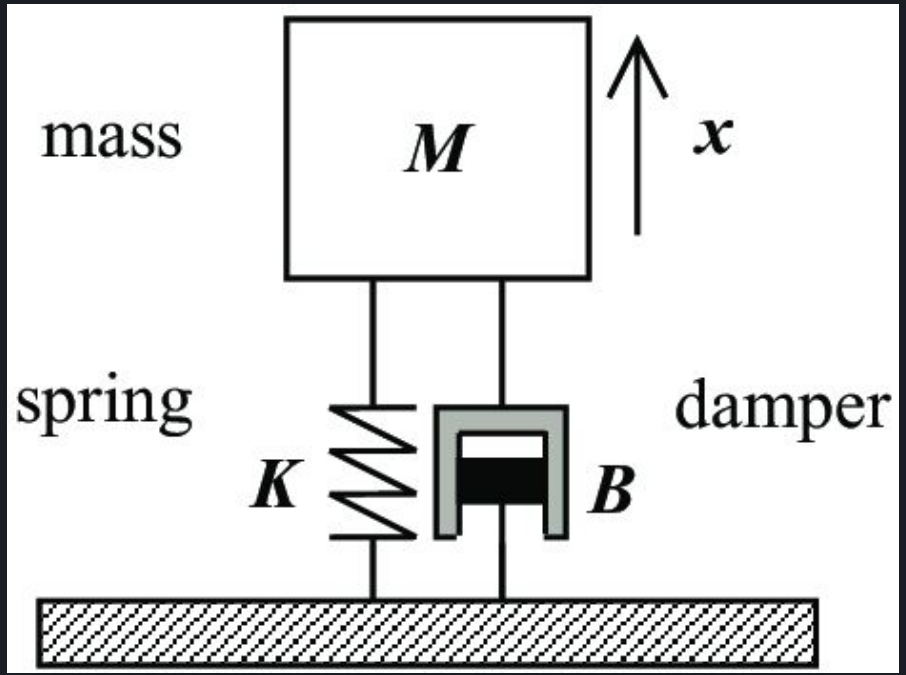


Real World Application

Applications

Mass Spring Damper System

$$m\ddot{x} + b\dot{x} + Kx = F \qquad x_0 = 1$$
$$Ma + bv + Kx = 0 \qquad b = 10$$
$$ma = -bv - Kx \qquad k = 400$$



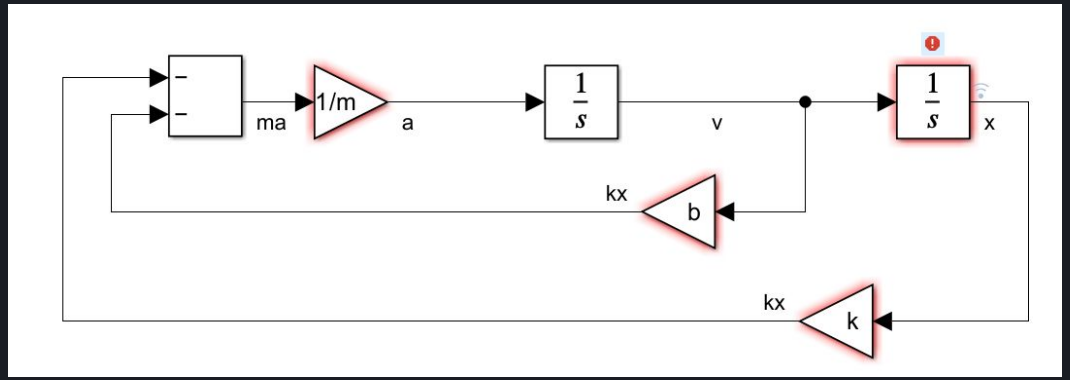
Real World Application

Applications

Mass Spring Damper System

MassSpringDamperSystemInit.m

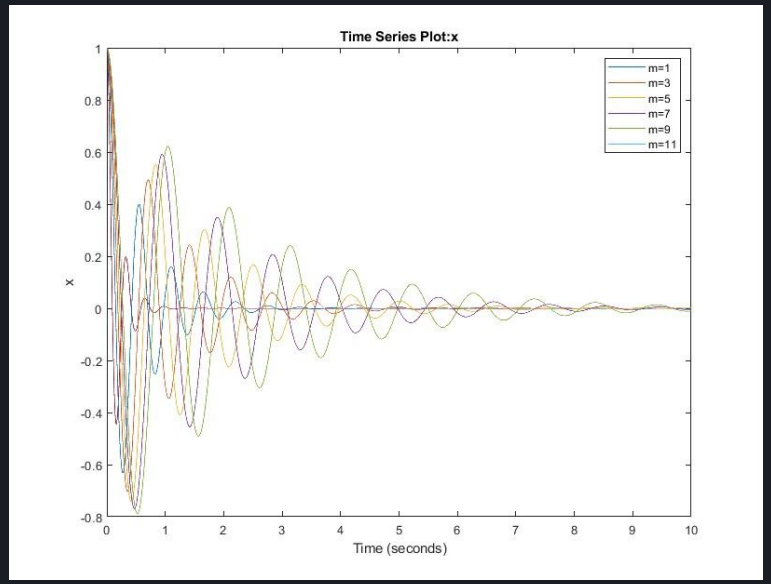
```
1  % define simulink paramters
2  b = 10;
3  k = 400;
4  x0 = 1;
5  m = 3;
6
```

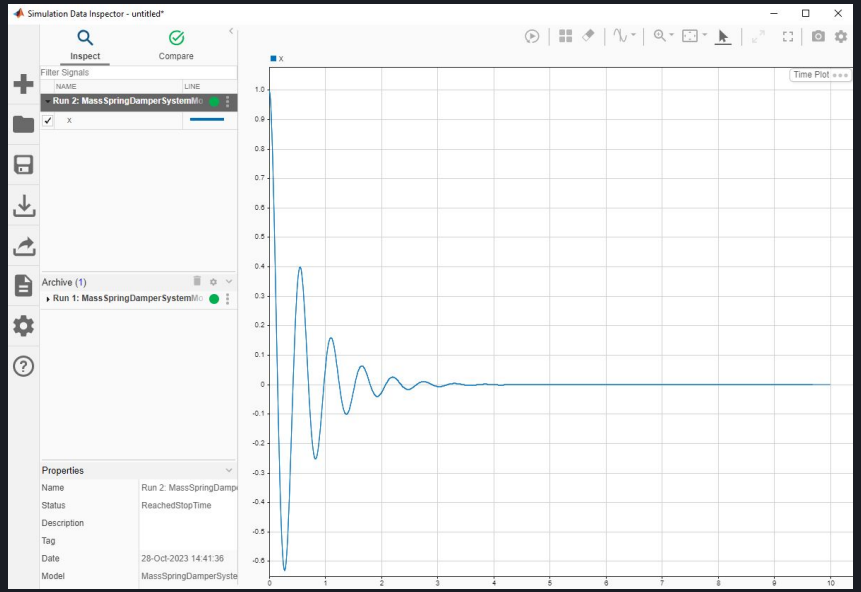


Real World Application

Applications

Mass Spring Damper System





Real World Application

Applications

Mass Spring Damper System

This code is used to perform simulations in Simulink for different mass values and plot the results. Here's a breakdown of what the code does:

1. It defines several Simulink parameters, including the parameters for a spring-mass-damper system. These parameters are `b` (damping coefficient), `k` (spring constant), `x0` (initial displacement), and `m` (mass).
2. It identifies the Simulink model to be used for simulation and stores it in the variable `mdl` using the `gcs` function (which stands for "get current system").
3. It creates an array `massValues` with values from 1 to 11 with a step of 2, representing different mass values to be used in the simulations.

MassSpringDamperSystemInit.m

```
% define simulink paramters
b = 10;
k = 400;
x0 = 1;
m = 3;

% sim model multiple time

mdl = gcs
massValues = 1:2:11;
for i = 1:numel(massValues)
    m = massVlaues(i);
    results = sim(mdl);
    plot(results.logsout.get("x").Values)
    hold on
    legendLables{i} ="m=" + num2str(m);
    disp("Simulation " + num2str(i) + "Compelete");
end
legend(legendLables);

```

Real World Application

Applications

4. The code enters a loop to perform simulations for each mass value in the `massValues` array. In each iteration of the loop:
   - It updates the mass parameter `m` to the current mass value from the `massValues` array.
   - It performs a simulation of the Simulink model using the `sim` function, and the results are stored in the `results` variable.
   - It plots the simulation results by extracting the values of the 'x' signal using `results.logsout.get("x").Values`.
   - It adds the simulation's legend label to differentiate the plotted results.
   - A message is displayed to indicate the completion of each simulation.
5. Finally, it creates a legend for the plot, indicating the different mass values used in the simulations.

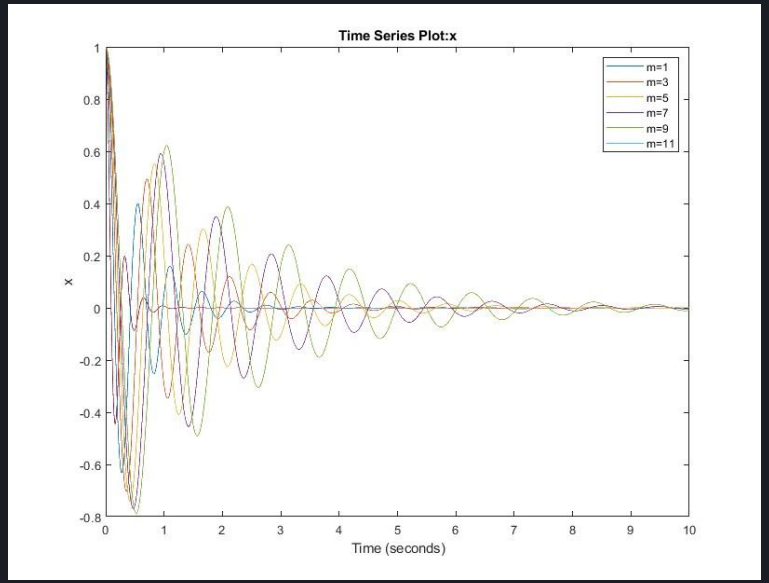
MassSpringDamperSystemInit.m

```
% define simulink paramters
b = 10;
k = 400;
x0 = 1;
m = 3;

% sim model multiple time

mdl = gcs
massValues = 1:2:11;
for i = 1:numel(massValues)
    m = massVlaues(i);
    results = sim(mdl);
    plot(results.logsout.get("x").Values)
    hold on
    legendLables{i} ="m=" + num2str(m);
    disp("Simulation " + num2str(i) + "Compelete");
end
legend(legendLables);

```

Real World Application

Applications

This code allows you to analyze the behavior of a spring-mass-damper system for various mass values and visualize the results with legends differentiating the simulations for each mass value.

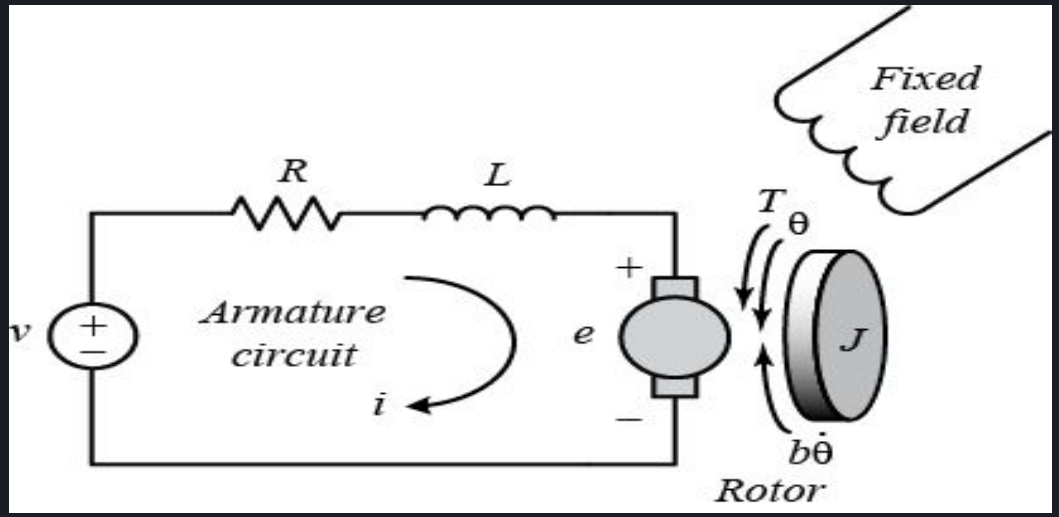


Real World Application

Applications

Separately excited dc motor (SEDC Motor)

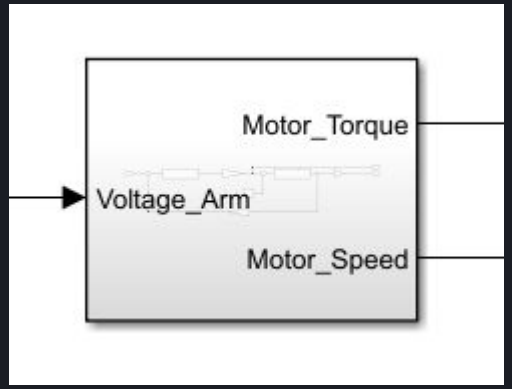
Modeling a separately excited DC motor in Simulink involves representing the electrical and mechanical components of the motor system

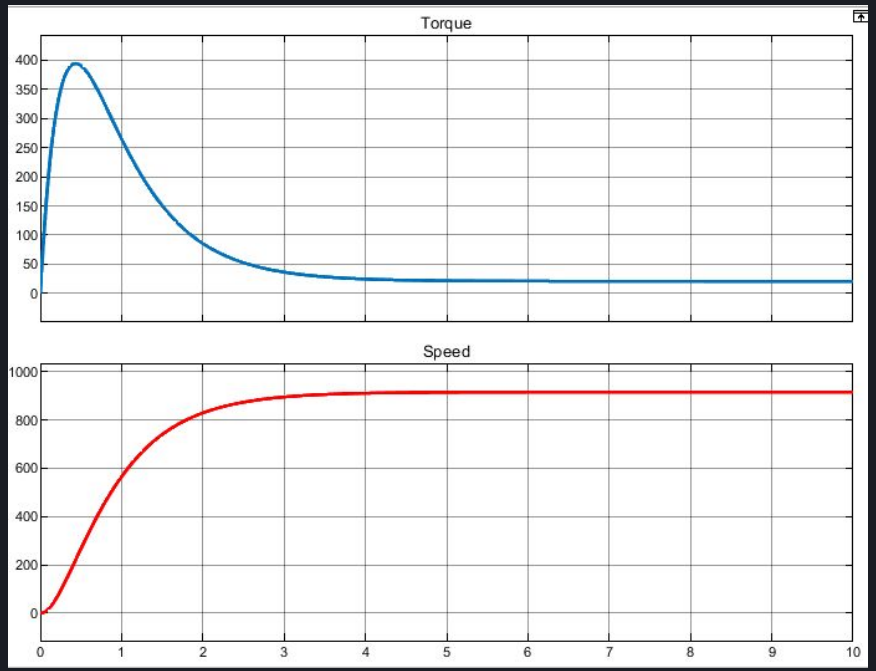


Real World Application

Applications

Separately excited dc motor (SEDC Motor)

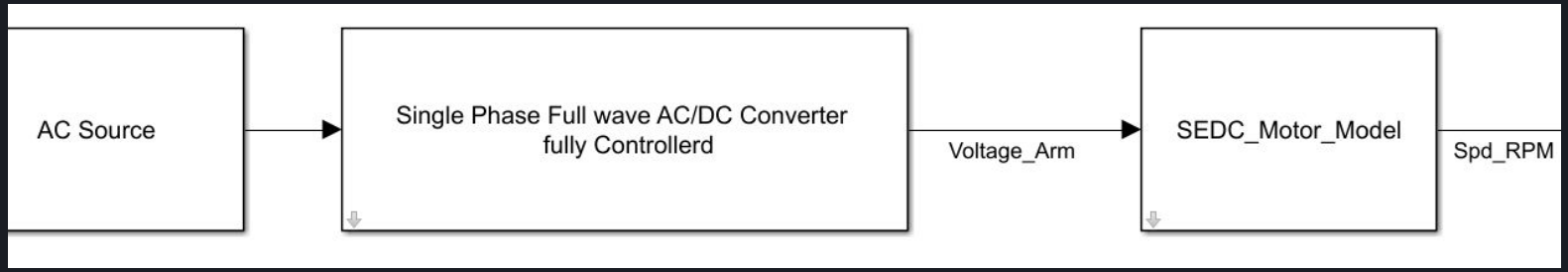




Real World Application

Applications

Try to do that




Thank You!