# CHAPTER 5
# Getting Started with SIMULINK

# Chapter 5: Getting Started with SIMULINK

## 5.1. Introduction

This chapter introduces SIMULINK, a graphical extension of MATLAB specifically designed for simulating dynamic systems. SIMULINK is widely used for modeling, simulation, and analysis of complex systems across various engineering fields. Throughout this chapter, you will learn how to navigate SIMULINK libraries, create simple models, and understand the fundamentals of simulation. Mastering SIMULINK is essential for those who aim to simulate and analyze dynamic systems in an integrated environment.

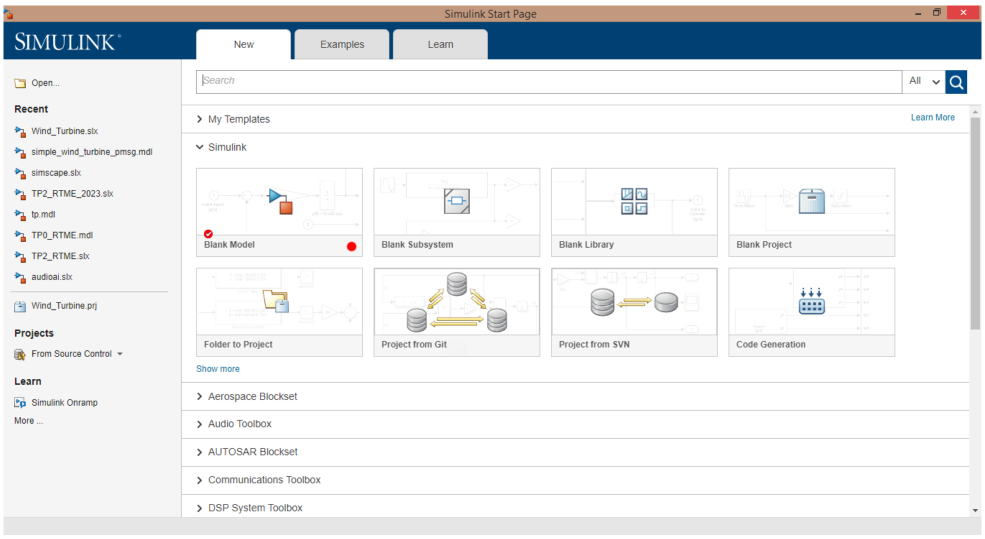

**Figure 5.1:** Simulink start page

After completing this chapter, you should be able to:

- Familiarize yourself with the SIMULINK interface and understand its key components.
- Use SIMULINK libraries to design and simulate simple dynamic models.
- Understand the importance of subsystems, masks, and callbacks in creating modular and efficient simulations.
- Utilize SIMULINK tools to effectively model and analyze real-world systems.

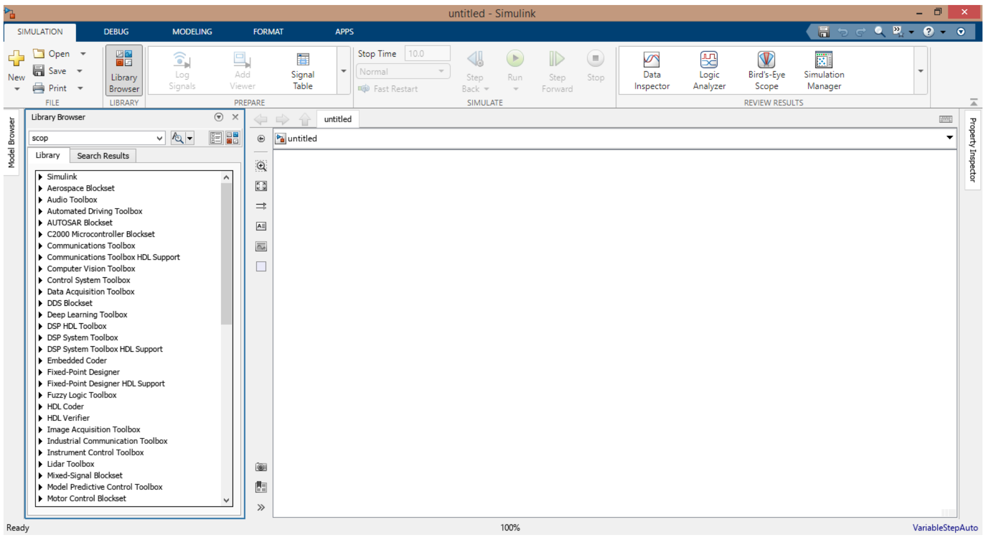

**Figure 5.2:** SIMULINK Libraries

## 5.2. Quick Start Guide

SIMULINK is a graphical simulation environment that enables users to model, simulate, and analyze dynamic systems. It is widely used in engineering, control systems, and research to simulate systems such as electrical circuits, mechanical systems, controllers, and more.

### 5.2.1. Basic Concepts

- **Blocks**: The fundamental elements in SIMULINK. Each block represents a specific operation or component (e.g., a mathematical function, a gain, a filter, etc.).
- **Signal Lines**: These lines connect blocks and allow signals to flow between them.
- **Block Library**: SIMULINK provides a vast library of **predefined blocks** that can be used to construct models.

### 5.2.2. Creating a Simple Model

**Step 1: Launching SIMULINK**

1. **Open MATLAB**: Ensure that MATLAB is running.
2. **Launch SIMULINK**:
   - Click on the **SIMULINK** icon in the MATLAB toolbar.
   - Or type **simulink** in the MATLAB command window and press **Enter**.

**Step 2: Creating a New Model**

1. **Create a Model**:
   - In the SIMULINK window, click **Blank Model** to create a new model.
   - A new **SIMULINK canvas** will appear where you can assemble your model.

**Step 3: Adding Blocks**

1. **Open the Block Libraries**:
   - Click on the **Library Browser** icon to open the SIMULINK block library.
   - The **Block Library** will display various categories such as **Sources, Sinks, Math Operations**, etc.
2. **Add Blocks to Your Model**:
   - **Source Block**: From **Sources**, drag a **Sine Wave** block onto the model canvas (generates a sinusoidal wave).
   - **Sink Block**: From **Sinks**, drag a **Scope** block (used to visualize the signal over time).
   - **Math Operations Block**: From **Math Operations**, drag a **Gain** block to amplify the signal.
3. **Connect the Blocks**:
   - Click on the output of the **Sine Wave** block and drag it to the input of the **Gain** block.
   - Then, connect the output of the **Gain** block to the input of the **Scope** block.
   - Your simple model is now ready.

### Step 4: Configuring the Blocks

1. **Configure the Gain Block**:
   - Double-click on the **Gain** block to open its settings window.
   - Enter a gain value (e.g., **2**) and click **OK**.
2. **Configure the Sine Wave Block**:
   - Double-click on the **Sine Wave** block to adjust parameters such as **amplitude, frequency, and phase** if necessary.

### Step 5: Running the Simulation

1. **Start the Simulation**:
   - Click the **Run** button (green triangle) in the model toolbar.
   - The simulation will run in **real-time**, and results will appear in the **Scope window**.
2. **Visualizing the Results**:
   - Double-click the **Scope** block to view the amplified sinusoidal waveform.
   - You can zoom in, adjust the display, and analyze the signal characteristics.

### Step 6: Saving the Model

1. **Save the Model**:
   - Click **File > Save As** in the model window.
   - Choose a location, name the model (e.g., **my_simple_model**), and click **Save**.

## 5.3. SIMULINK Libraries and Their Blocks

SIMULINK provides a graphical representation of systems using interconnected functional blocks. It is an invaluable tool for modeling linear, nonlinear, continuous, discrete, or hybrid systems.

SIMULINK libraries contain predefined blocks that can be used to build models. These blocks are categorized into different functional groups, such as Sources, Math Operations, Continuous Systems, etc.

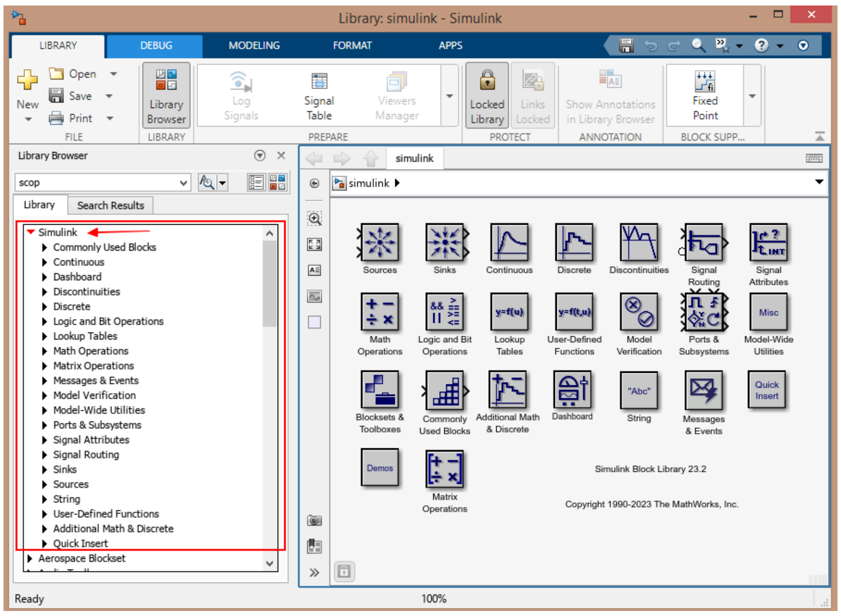

**Figure 5.3: SIMULINK Libraries and Their Blocks**

### 5.3.1. Sources Library

The **Sources** library contains blocks used to generate input signals for SIMULINK models.

- **Constant**: Produces a constant value.
- **Sine Wave**: Generates a sinusoidal waveform.
- **Step**: Generates a step signal transitioning from zero to a given value at a specific time.
- **Ramp**: Generates a ramp signal that increases or decreases linearly.
- **Random Number**: Produces random numbers (uniform or Gaussian distribution).
- **From Workspace**: Imports data from the MATLAB workspace.
- **Clock**: Provides a time-based output corresponding to the current simulation time.

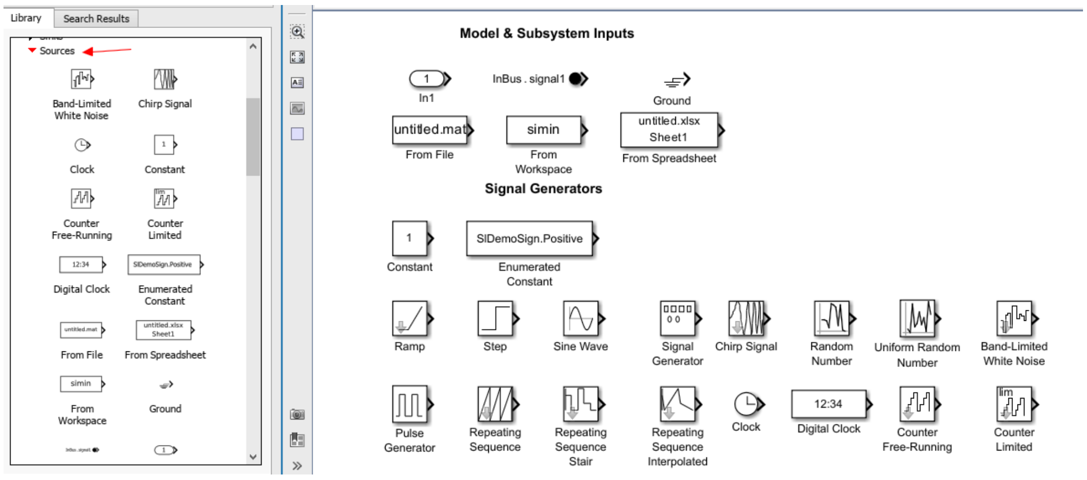

**Figure 5.4: Sources Library**

### 5.3.2. Sinks Library

The Sinks library contains blocks used for visualizing, recording, or processing signals from the simulation.

- **Scope**: Displays real-time waveforms during simulation.
- **To Workspace**: Saves simulation data to the MATLAB workspace for post-processing.
- **Display**: Shows real-time signal values during simulation.
- **To File**: Saves simulation results to a .mat file for later analysis.
- **XY Graph**: Plots **X-Y** relationships, useful for visualizing system characteristics.

### Typical Uses

- **Visualization**: Use blocks like **Scope** to monitor signals in real-time.
- **Recording**: Use **To Workspace** or **To File** to store simulation results.
- **Instantaneous Display**: Use **Display** to check the current value of a variable.
- **Graphical Analysis**: Use **XY Graph** to compare relationships between different signals.

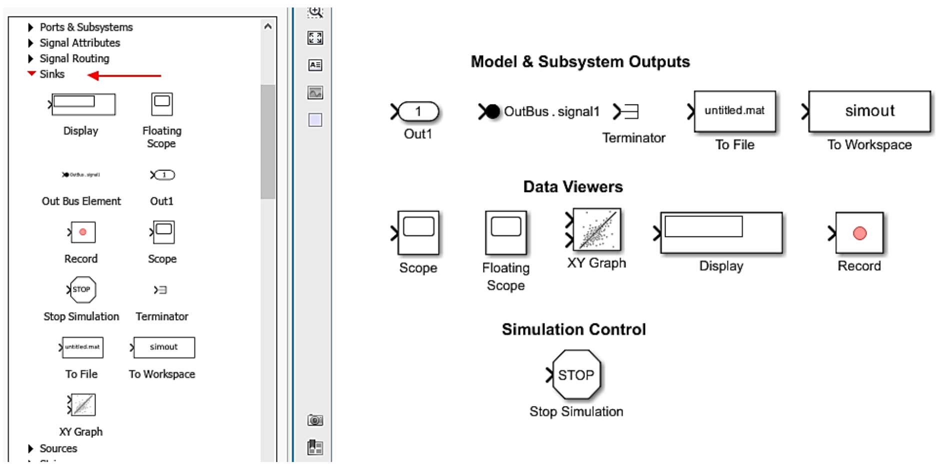

Figure 5.5: Sinks Library

## 5.3.3. Continuous Library

The Continuous library consists of blocks used to model and simulate continuous-time systems, which evolve smoothly over time without discrete steps. These blocks are particularly useful for modeling systems governed by differential equations (e.g., electrical circuits, mechanical systems, thermal systems).

### Key Continuous Blocks

1. **Integrator**: Computes the integral of an input signal over time.
2. **Transfer Fcn**: Represents a **linear system** using a **transfer function** (Laplace domain).
3. **State-Space**: Models systems in **state-space form** using matrices.
4. **Derivative**: Computes the **time derivative** of an input signal.
5. **PID Controller**: Implements a **Proportional-Integral-Derivative (PID) controller** for system control.

### Example: First-Order System Response

A first-order system can be modeled using:

- **State-Space** or **Transfer Fcn** for system representation.
- **Scope** to visualize voltage and current over time.
- **To Workspace** to save and analyze results.
- **Display** to monitor real-time power output.

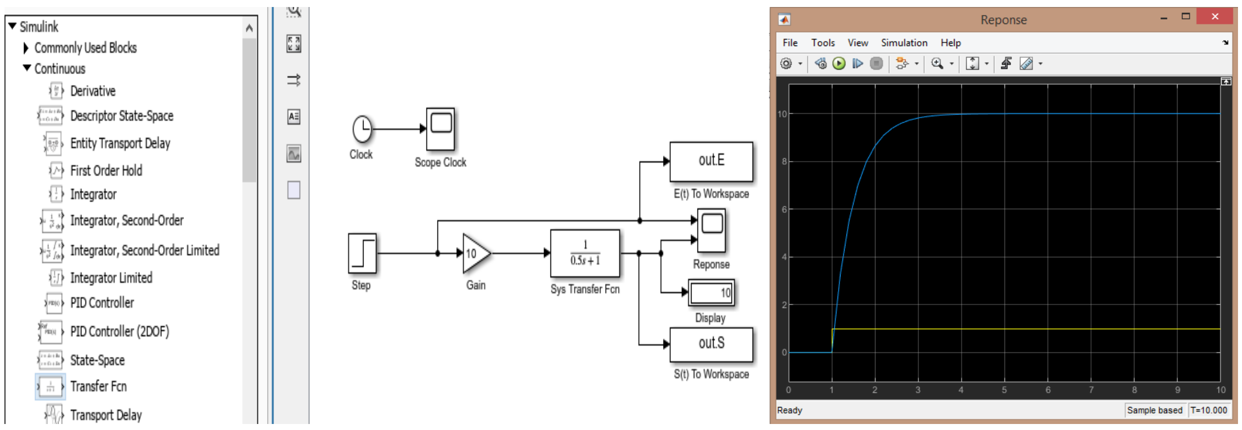

Figure 5.6: First-Order System Response

### 5.3.4. Math Operations

In SIMULINK, the Math Operations category contains blocks used to perform various mathematical operations on signals within a simulation. These blocks allow for manipulation, transformation, and combination of signals through algebraic, logical, or statistical operations. They are essential for constructing complex models that require mathematical computations.

- Enable mathematical and logical operations on signals within a SIMULINK model.
- Use these operations to transform signals, perform calculations, and implement mathematical algorithms.

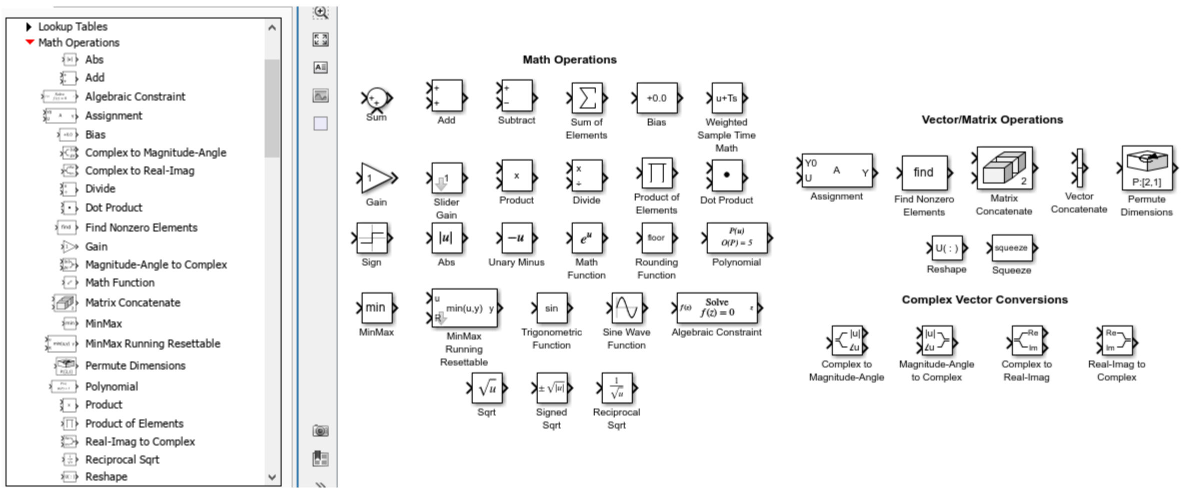

**Figure 5.7: Math Operations Blocks**

#### 5.3.4.1. Key Math Operations Blocks

1. **Add (Summation)**
   - Performs **addition or subtraction** of signals. The block can be configured to add or subtract multiple inputs.
   - Used to **arithmetically combine** signals.
2. **Gain**
   - Multiplies an input signal by a **gain factor**, commonly used for amplifying or attenuating a signal.
   - The gain can be **scalar or matrix-based**, enabling more complex linear transformations.
3. **Product**
   - Multiplies two or more signals. Can also perform **division** if configured accordingly.
   - Often used to model **multiplicative relationships**, such as **physical laws** (e.g., power = voltage × current).
4. **Math Function**
   - Applies basic **mathematical functions** such as **square root, exponential, logarithm, etc.**
   - Flexible for performing **various calculations** within models.

5. **Abs (Absolute Value)**
    - Computes the **absolute value** of a signal, useful when dealing with magnitudes.
6. **Sum of Elements**
    - Computes the sum of all elements in a **vector or matrix**, often used in **statistical computations** or data aggregation.
7. **Dot Product**
    - Computes the **dot product** of two vectors, crucial for **linear algebra operations** or **control systems**.
8. **Trigonometric Function**
    - Applies **trigonometric operations** such as **sin, cos, tan**, and their inverses.
    - Commonly used to model **periodic or oscillatory systems**.
9. **Logical Operator**
    - Performs **logical operations** like **AND, OR, NOT**, etc., on **Boolean signals**.
    - Essential for implementing **conditional logic and control mechanisms**.
10. **MinMax (Minimum/Maximum)**

- Determines the **minimum or maximum** between two or more input signals.
- Useful in **selection or comparison systems**.

### 5.3.4.2. Typical Applications

- **Mathematical Modeling**: These blocks help create **mathematical representations** of physical, economic, or other systems.
- **Signal Transformation**: Apply **multiplication, addition, or trigonometric functions** to transform signals as needed in a model.
- **Control Algorithms**: Use **mathematical and logical operators** to implement **control strategies**, such as **PID controllers** or **conditional logic**.

### 5.3.4.3. Application Example

In a **motor speed control system**:

- The **Gain** block can be used to adjust the **amplification** of a command signal.
- The **Add** block can combine **feedback signals** to determine the **control error**.
- The **Product** block could calculate **motor power** based on voltage and current.

The **Math Operations** blocks are fundamental in **SIMULINK** for performing computations and signal manipulations, making it possible to create complex and precise models.

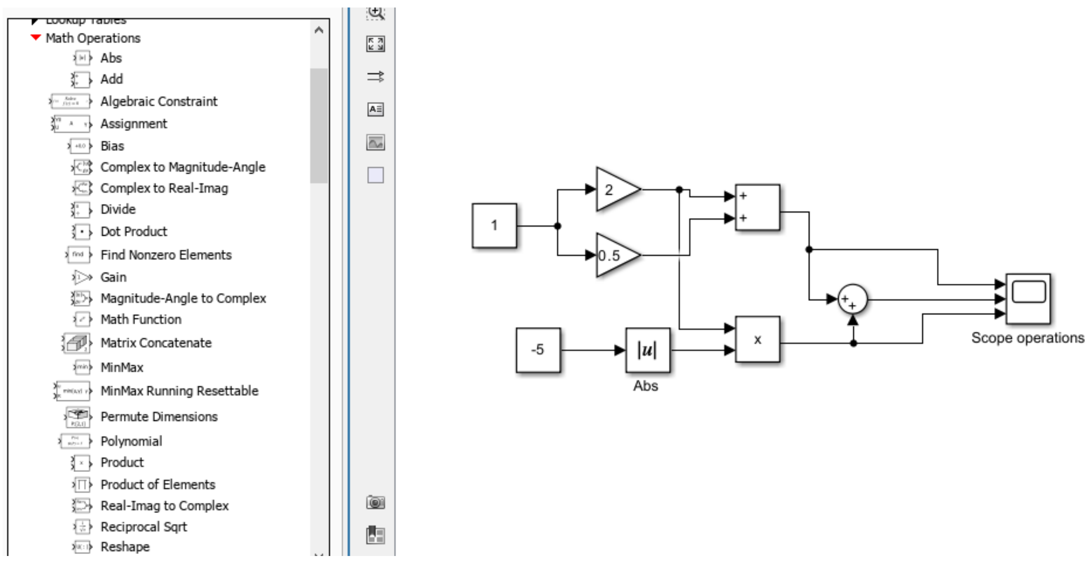

**Figure 5.8: Math Operations Example**

### 5.3.5. Commonly Used Blocks

In SIMULINK, certain blocks are widely used due to their fundamental role in modeling, simulation, and analysis of dynamic systems. These blocks, although part of different categories, are frequently employed in most SIMULINK models.

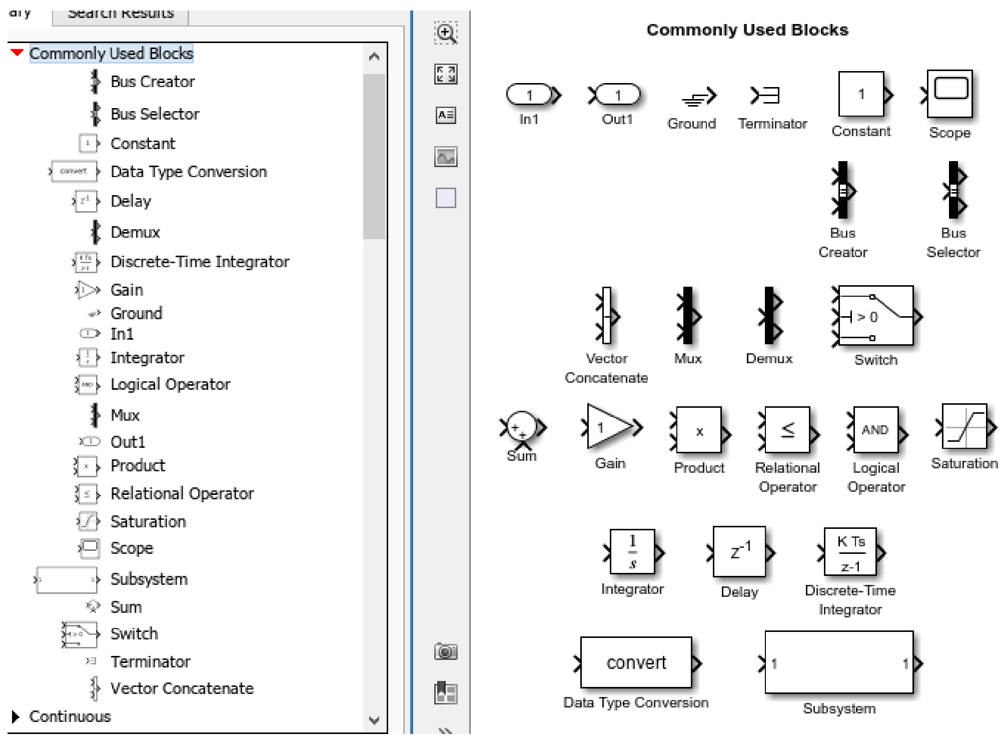

**Figure 5.9: Commonly Used Blocks**

#### 5.3.5.1. Frequently Used Blocks

1. **Integrator (Continuous)**
   - **Purpose**: Computes the **integral of a signal** over time, crucial for **modeling dynamic systems**.
   - **Use Case**: Often used to model **state variables**, such as **position derived from velocity** or **charge accumulation in a capacitor**.

2. **Gain (Math Operations)**
   - **Purpose**: Multiplies the input signal by a **constant factor (gain)**.
   - **Use Case**: Useful for **scaling signals** or applying **linear transformations**, e.g., converting **voltage to current**.
3. **Sum (Math Operations)**
   - **Purpose**: Adds or subtracts multiple input signals.
   - **Use Case**: Commonly used in **feedback control systems**, such as **PID controllers** or **voltage summation in circuits**.
4. **Scope (Sinks)**
   - **Purpose**: Displays signals **in real time** as graphical waveforms.
   - **Use Case**: Essential for **visualizing system behavior** and **debugging models**.
5. **Constant (Sources)**
   - **Purpose**: Generates a **constant signal**.
   - **Use Case**: Used to **set reference values**, such as **control setpoints** or **gain parameters**.
6. **Mux/Demux (Signal Routing)**
   - **Purpose**:
     - **Mux** combines multiple signals into **one vector**.
     - **Demux** splits a **vector into multiple signals**.
   - **Use Case**: Organizes **multiple signals** in **complex systems**, such as **state feedback systems**.
7. **Transfer Function (Continuous)**
   - **Purpose**: Models a **linear system** using a **transfer function** in the **Laplace domain**.
   - **Use Case**: Used for **dynamic system modeling**, including **filters, amplifiers, and control systems**.
8. **To Workspace (Sinks)**
   - **Purpose**: Saves simulation signals to the **MATLAB workspace** for **post-processing**.
   - **Use Case**: Frequently used for **data collection** and **performance analysis**.
9. **Switch (Signal Routing)**
   - **Purpose**: Selects **one input signal** among multiple options based on a condition.
   - **Use Case**: Used for **decision-making logic**, such as **switching between operation modes**.
10. **Logical Operator (Math Operations)**

- **Purpose**: Performs **Boolean operations** (**AND, OR, NOT**) on logical signals.
- **Use Case**: Essential for **control systems** requiring **logic-based decisions**.

### 5.3.5.2. Application Example

In a first-order system model, these blocks can be used as follows:

- Integrator: Computes position from velocity.
- Gain: Adjusts control signals based on error feedback.
- Sum: Combines error and reference signals.
- Scope: Visualizes real-time motor speed response.

These commonly used blocks form the foundation of many SIMULINK models, enabling efficient construction and analysis of complex systems.

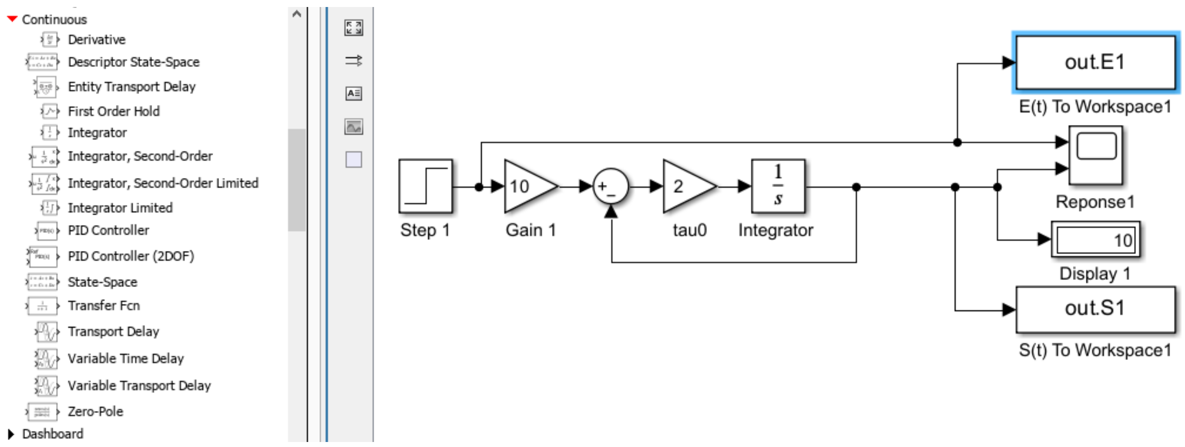

Figure 5.10: First-Order System with Integrator

### 5.3.6. Signal Routing

The Signal Routing category includes blocks designed to direct, combine, split, or manipulate signals within a SIMULINK model. These blocks are essential for organizing and managing data flow, especially in complex systems.

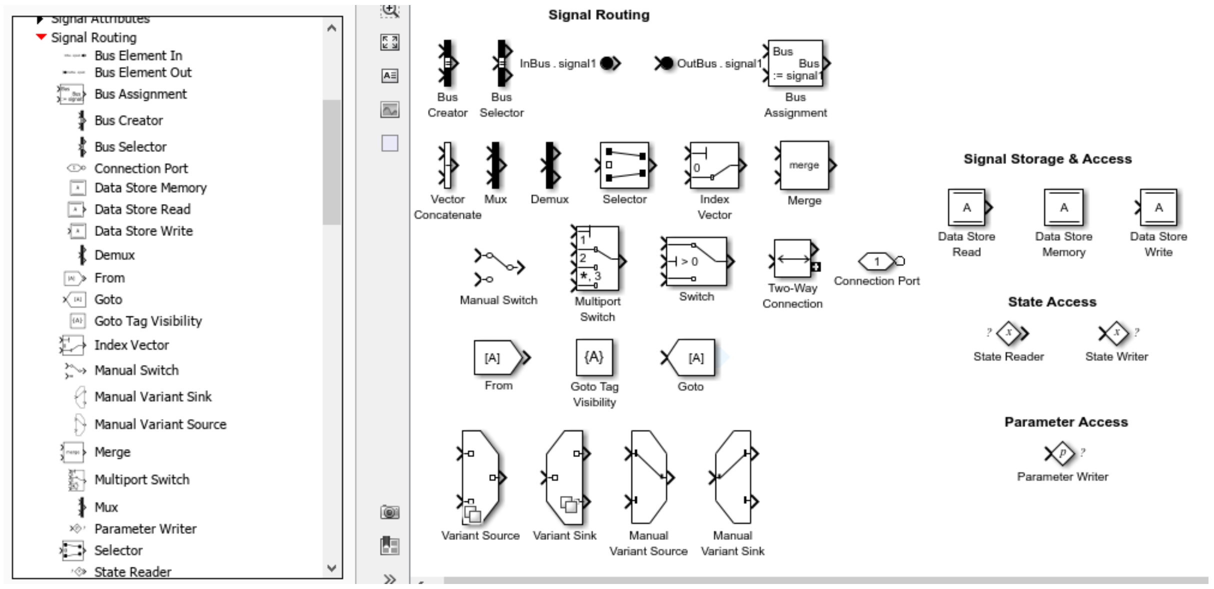

Figure 5.11: Signal Routing

### 5.3.6.1. Key Signal Routing Blocks

1. **Mux (Multiplexer)**
   - **Purpose**: Combines multiple input signals into **one vector output**.
   - **Use Case**: Used to **group signals** before processing, e.g., **multivariable system inputs**.
2. **Demux (Demultiplexer)**
   - **Purpose**: Splits a **vector signal** into multiple individual signals.
   - **Use Case**: Extracts specific **components from a processed signal**.
3. **Bus Creator / Bus Selector**
   - **Bus Creator**: Groups **multiple signals** into **one bus signal**.
   - **Bus Selector**: Extracts specific **signals from a bus**.
   - **Use Case**: Useful for organizing signals efficiently in **large-scale models**.
4. **Switch**
   - **Purpose**: Selects **one input signal** based on a **logical condition**.
   - **Use Case**: Switches between **operating modes** in control systems.
5. **Selector**
   - **Purpose**: Extracts **specific elements** from a **vector or matrix**.
   - **Use Case**: Useful for **targeted analysis or specialized computations**.

These Signal Routing blocks help streamline signal management, ensuring efficient and modular system design in SIMULINK.

## 5.3.7. Logic and Bit Operations

The Logic and Bit Operations category includes blocks that perform logical and bitwise operations on signals. These operations are often used for decision-making, flow control, and digital signal manipulation.

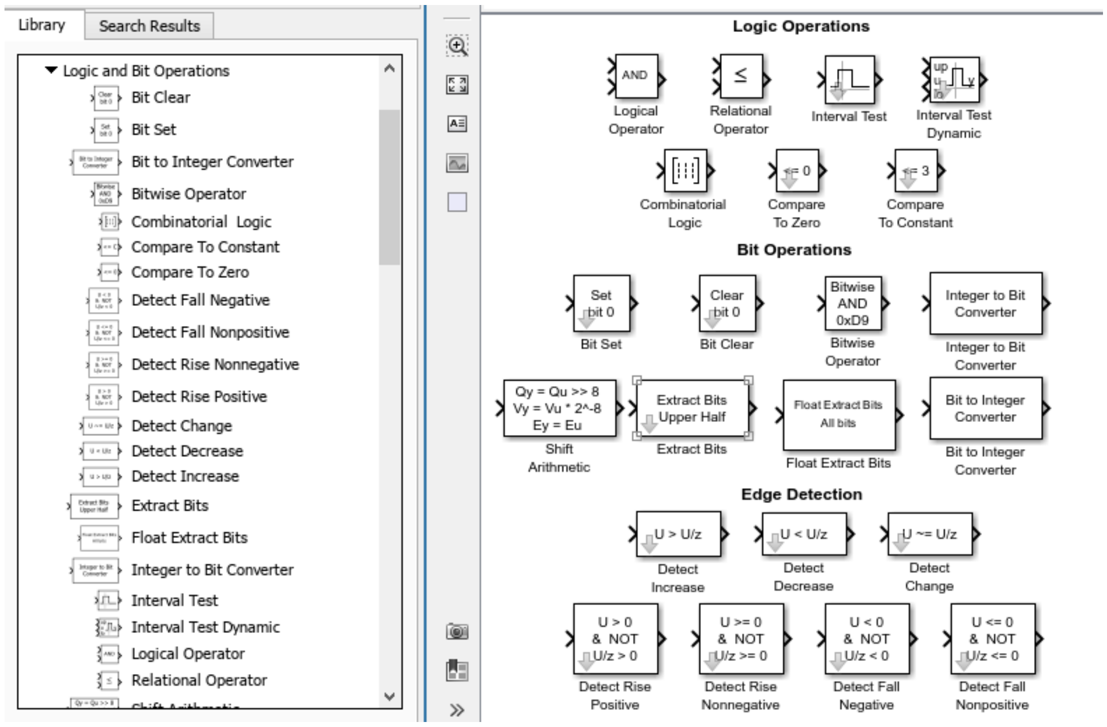

**Figure 5.12: Logic and Bit Operations**

#### 5.3.7.1. Key Blocks

1. **Logical Operator**
   - **Purpose**: Performs **logical operations** such as **AND, OR, NOT, XOR, etc.** on Boolean signals.
   - **Use Case**: Used in **control systems, safety mechanisms, and decision-making processes.**
2. **Relational Operator**
   - **Purpose**: Compares two signals using **relational operators** such as **>, <, ==, etc.**
   - **Use Case**: Essential for evaluating conditions, such as **threshold detection or boundary limits**, in control models.
3. **Bitwise Operator**
   - **Purpose**: Executes **bitwise operations** (AND, OR, NOT, XOR, Shift) on **integer signals.**
   - **Use Case**: Useful for **data manipulation at the binary level**, often used in **digital processing and embedded systems**.
4. **Compare To Zero / Compare To Constant**
   - **Purpose**: Compares a signal to **zero** or a **specified constant.**
   - **Use Case**: Used to detect **signal transitions or threshold crossings** in a system.

### 5.3.8. User-Defined Functions

The User-Defined Functions category allows users to create custom blocks to execute specific functions that are not available in standard blocks. These blocks provide maximum flexibility for integrating custom algorithms or logic into a SIMULINK model.

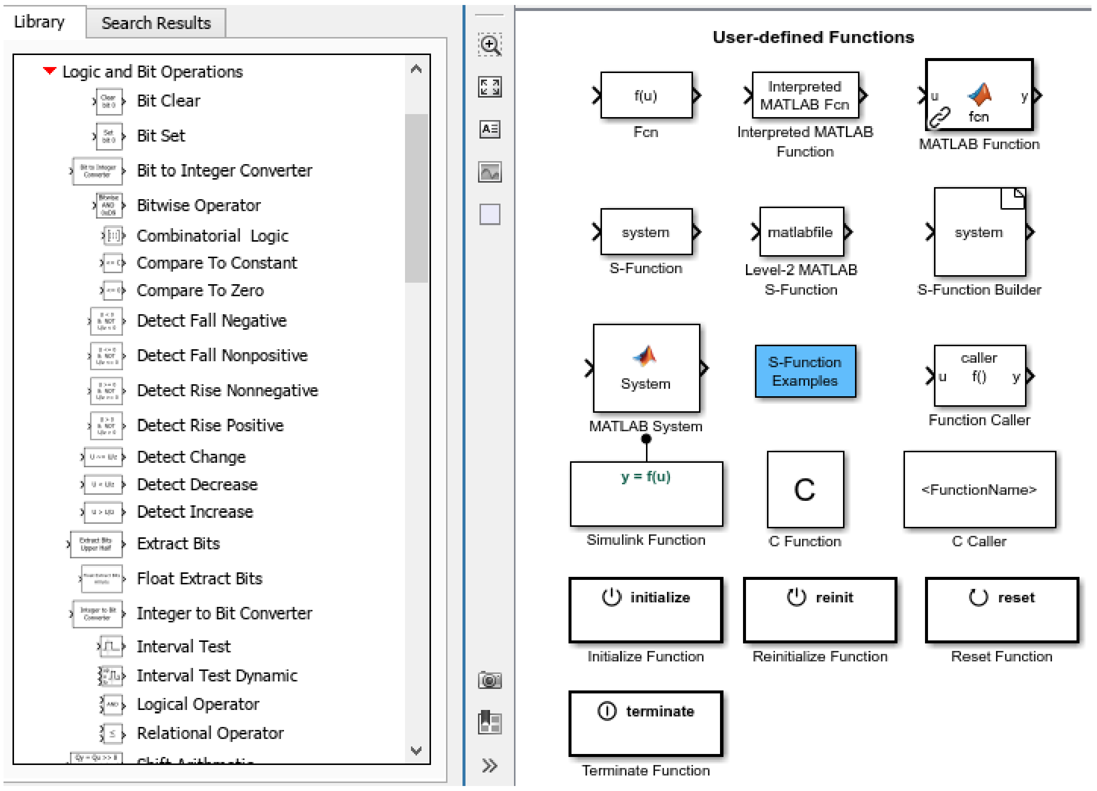

Figure 5.13: User-Defined Functions

#### 5.3.8.1. Key Blocks

1. **MATLAB Function**
    - **Purpose**: Enables users to write **MATLAB code directly** inside a **SIMULINK block** to perform complex calculations or logic.
    - **Use Case**: Used for **custom functions, specialized algorithms, or flexible data manipulations** within a simulation.
2. **S-Function**
    - **Purpose**: Creates **custom blocks** using **MATLAB, C, C++, or Fortran code** to execute specific functions in SIMULINK.
    - **Use Case**: Essential for integrating **complex algorithms or external models** into a **SIMULINK model**, providing maximum customization.
3. **Interpreted MATLAB Function**
    - **Purpose**: Executes a **MATLAB function dynamically** during simulation.
    - **Use Case**: Used for calculations that require **real-time execution of MATLAB code** without generating compiled code.

### 5.3.9. Ports & Subsystems

The Ports & Subsystems category contains blocks that help organize SIMULINK models into subsystems, creating hierarchical structures that simplify complex models and improve reusability.

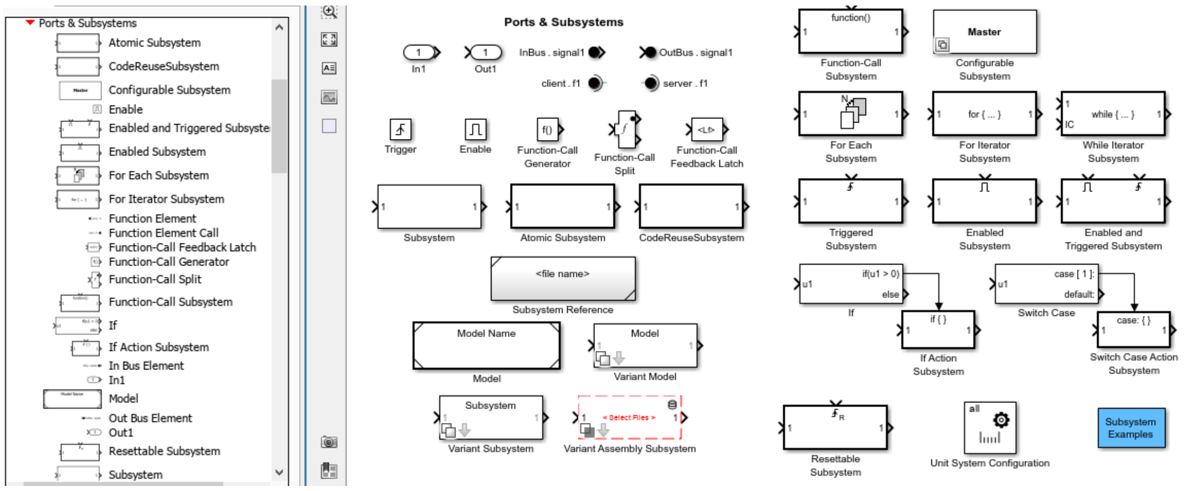

**Figure 5.14: User-Defined Functions**

#### 5.3.9.1. Key Blocks

1. **Subsystem**
    - **Purpose**: Groups multiple blocks into a **subsystem**, which can be treated as a **single block** in the main model.
    - **Use Case**: Used to **structure complex models into simpler modules**, facilitating **design, testing, and reusability**.
2. **Inport / Outport**
    - **Purpose**: Defines **input (Inport) and output (Outport) ports** for signals in a **subsystem** or **SIMULINK model**.

- **Use Case**: Essential for **connecting subsystems** and organizing **data flow** within a model.

3. **Enable / Trigger / Action Subsystem**
    - **Enable Subsystem**: Activates **when the enable signal is true**.
    - **Trigger Subsystem**: Activates **in response to a trigger signal**.
    - **Action Subsystem**: Executes actions **conditionally based on control signals**.
    - **Use Case**: Used to control **execution flow** in a model **based on dynamic conditions**.
4. **For Each Subsystem**
    - **Purpose**: Repeats an operation **on individual elements** of a **vector or matrix signal**.
    - **Use Case**: Useful for **processing signals in parallel**, such as **batch processing of data subsets**.
5. **Bus Creator / Bus Selector (Ports & Subsystems)**
    - **Purpose**: Organizes signals into a **bus** or **extracts signals from a bus**.
    - **Use Case**: Helps **manage complex signal structures efficiently**, simplifying the **connectivity of subsystems**.

## 5.4. Masks and Subsystems

This section introduces subsystem management in SIMULINK, including creating masks for subsystems and using callbacks to automate actions. These tools are crucial for structuring complex models, improving reusability, and customizing the user interface.

### 5.4.1. What is a Subsystem?

A subsystem is a group of SIMULINK blocks encapsulated into a single block, simplifying the main model. Subsystems help organize complex models into simpler, reusable modules.

#### 5.4.1.1. Creating a Subsystem

1. **Select blocks to encapsulate**: Choose multiple blocks in the **SIMULINK model**.
2. **Create a Subsystem**:
    - **Right-click** on the selected blocks and choose **Create Subsystem**.
    - The selected blocks are grouped into a **single block named "Subsystem"**.
3. **Inport and Outport**:
    - The **input and output connections** of the selected blocks are automatically replaced by **Inport and Outport blocks** inside the subsystem.

#### 5.4.1.2. Types of Subsystems

- **Standard Subsystem**: The most common type, used for **organizing model components**.
- **Enabled Subsystem**: Activates **only when the enable signal is true**.
- **Triggered Subsystem**: Executes **in response to a trigger signal**.

- **Function-Call Subsystem**: Executes when **called by a function signal**.

### 5.4.2. Masking Subsystems

Masking allows users to create a custom interface for a subsystem, hiding its internal complexity and simplifying its usage.

#### 5.4.2.1. Creating a Mask for a Subsystem

1. **Access the Mask Editor**:
   - **Right-click** on the subsystem and select **Create Mask**.
2. **Configure the Mask**:
   - In the **Mask Editor**, define mask parameters, add **user controls** (text fields, checkboxes), and **customize the appearance**.
   - **Parameters & Dialog**: Define parameters that the user can modify externally.
   - **Icon & Ports**: Customize the **subsystem's icon and visible ports**.
3. **Using Mask Parameters**:
   - The defined **parameters** can be used inside the subsystem to **modify block behavior** based on user inputs.

#### 5.4.2.2. Advantages of Masking

- **Encapsulation**: Hides the **internal complexity** of a subsystem.
- **Reusability**: Makes the subsystem **easier to reuse** in other models.
- **Simplification**: Simplifies interaction **with complex subsystems** for users.

### 5.4.3. Using Callbacks

Callbacks are MATLAB scripts associated with specific SIMULINK events (e.g., opening a block, modifying a parameter). They help automate model actions.

#### 5.4.3.1. Main Types of Callbacks

1. **Model Callbacks**
   - **PreLoadFcn**: Executes **before loading the model**.
   - **InitFcn**: Runs **at the beginning of the simulation** or **when updating the model**.
   - **StartFcn**: Executes **at the start of each simulation**.
   - **StopFcn**: Executes **at the end of each simulation**.
2. **Block Callbacks**
   - **OpenFcn**: Executes when **the block is opened**.
   - **CloseFcn**: Executes when **the block is closed**.
   - **MaskInitialization**: Executes **when initializing a block mask**.

#### 5.4.3.2. Configuring a Callback

1. **Access Model/Block Properties**
   - For a **model**: Go to **Model Properties** via **File > Model Properties**.

- For a **block**: **Right-click** on the block and select **Properties**.

2. **Add Callback Code**
    - In the **callback section** (e.g., **InitFcn, OpenFcn**), enter the **MATLAB code** to execute.

**Example Applications**

- **Parameter Initialization**: Use callbacks to **initialize model parameters** before simulation.
- **Task Automation**: Automate **data loading, logging, or report generation**.

## 5.5. Study of Simulation Examples

These examples cover different simulation applications and demonstrate how to use SIMULINK to model, simulate, and analyze various dynamic systems.

### 5.5.1. Example 1: First-Order System

**Description**

A first-order system is one of the simplest dynamic systems. It is commonly used to model phenomena such as the response of an RC circuit or the thermal behavior of an object.

#### 5.5.1.1. Modeling

1. **Create the model**:
    - In **SIMULINK**, add an **Integrator** block to represent the integration of the **derivative of the signal**.
    - Add a **Gain** block to define the **time constant** of the system.
    - Use a **Sum** block to subtract the **system output from the input**.
    - Connect the blocks to form a **closed-loop system**.
2. **Configure the parameters**:
    - Set the **time constant** in the **Gain block** based on the system being modeled.
    - Add a **Step block** to provide a **step input** to the system.
3. **Simulate the model**:
    - Run the **simulation** and observe the **system's response** to the step input.
    - Use a **Scope block** to visualize the **output signal**.
4. **Analyze the results**:
    - Examine the **system response**, particularly the **response time** and **final value**, to verify consistency with **first-order system theory**.

### 5.5.2. Example 2: PID Control System

A PID (Proportional-Integral-Derivative) controller is used to regulate the output of a system based on a reference setpoint. This type of controller is widely used in industrial applications for controlling temperature, speed, position, etc.

#### 5.5.2.1. Modeling

1. **Create the model**:
   - Add a **PID Controller** block from the **SIMULINK library**.
   - Connect a **Plant block** (which could be the previous first-order system) in a **closed-loop configuration** with the **PID controller**.
   - Use a **Step** or **Sine Wave** block to provide an **input signal** to the system.
2. **Configure the parameters**:
   - Set the **proportional, integral, and derivative gains** of the **PID controller**.
   - Modify the **Plant parameters** to represent the **dynamics of the controlled system**.
3. **Simulate the model**:
   - Run the **simulation** and observe how the **PID controller** regulates the system to **track the reference setpoint**.
   - Use a **Scope block** to visualize both the **setpoint** and the **system response**.
4. **Analyze the results**:
   - Evaluate the **controller's performance** in terms of:
     - **Overshoot**,
     - **Settling time**,
     - **Steady-state error**.
   - Adjust the **PID gains** to optimize the **system response**.

### 5.5.3. Example 3: RLC Electrical Circuit

An RLC circuit consists of a resistor (R), inductor (L), and capacitor (C). It is used to model oscillatory behavior, such as resonance in electrical systems.

#### 5.5.3.1. Modeling

1. **Create the model**:
   - In **SIMULINK**, use the **Resistor, Inductor, and Capacitor** blocks from **Simscape > Foundation Library > Electrical**.
   - Connect these components **in series or parallel** to form the **RLC circuit**.
   - Add a **Voltage Source block** to provide **excitation to the circuit**.
2. **Configure the parameters**:
   - Set the **R, L, and C values** according to the circuit to be simulated.
   - Define the **voltage source parameters** (amplitude, frequency) to excite the circuit.
3. **Simulate the model**:
   - Run the **simulation** to observe the **RLC circuit response** to the excitation.
   - Use a **Scope block** to visualize the **voltages and currents** in the circuit.
4. **Analyze the results**:
   - Examine the **resonance frequency**, **transient response**, and **steady-state response** of the circuit.

- Compare the **simulation results** with **theoretical calculations** to **validate the model**.

## 5.6. Conclusion

This chapter introduced you to SIMULINK, a powerful tool for simulating dynamic systems in the MATLAB environment. By understanding the basic components and libraries of SIMULINK, you are now able to build and simulate simple models, which is essential for analyzing complex systems in various engineering fields. The knowledge gained here will provide a foundation for more advanced simulations and system analyses, especially when you explore specialized tools such as the Power System Blockset in the next chapter.