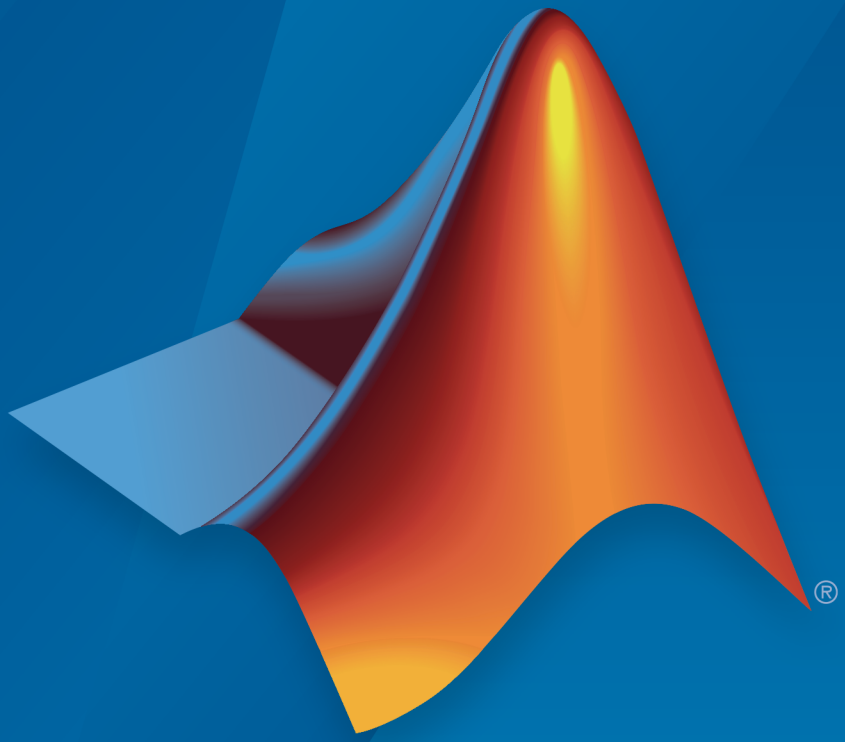
DSP System Toolbox™

Getting Started Guide



MATLAB® & SIMULINK®

R2017a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

DSP System Toolbox™ Getting Started Guide

Revision History

April 2011	First printing	Revised for Version 8.0 (R2011a)
September 2011	Online only	Revised for Version 8.1 (R2011b)
March 2012	Online only	Revised for Version 8.2 (R2012a)
September 2012	Online only	Revised for Version 8.3 (R2012b)
March 2013	Online only	Revised for Version 8.4 (R2013a)
September 2013	Online only	Revised for Version 8.5 (R2013b)
March 2014	Online only	Revised for Version 8.6 (R2014a)
October 2014	Online only	Revised for Version 8.7 (R2014b)
March 2015	Online only	Revised for Version 9.0 (R2015a)
September 2015	Online only	Revised for Version 9.1 (R2015b)
March 2016	Online only	Revised for Version 9.2 (R2016a)
September 2016	Online only	Revised for Version 9.3 (R2016b)
March 2017	Online only	Revised for Version 9.4 (R2017a)

Contents

1 Introduction

2 Design a Filter with fdesign and Filter Builder

3 Design Filters in Simulink

1

Introduction

DSP System Toolbox Product Description

Design and simulate streaming signal processing systems

DSP System Toolbox™ provides algorithms, apps, and scopes for designing, simulating, and analyzing signal processing systems in MATLAB® and Simulink®. You can model real-time DSP systems for communications, radar, audio, medical devices, IoT, and other applications.

With DSP System Toolbox you can design and analyze FIR, IIR, multirate, multistage, and adaptive filters. You can stream signals from variables, data files, and network devices for system development and verification. The Time Scope, Spectrum Analyzer, and Logic Analyzer let you dynamically visualize and measure streaming signals. For desktop prototyping and deployment to embedded processors, including ARM® Cortex® architectures, the toolbox supports C/C++ code generation. It also supports bit-accurate fixed-point modeling and HDL code generation from filters, FFT, IFFT, and other algorithms.

Algorithms are available as MATLAB functions, System objects, and Simulink blocks.

Key Features

- Signal processing and linear algebra blocks for Simulink
- Streaming signal processing in MATLAB
- Single-rate, multirate, FIR, IIR, and adaptive filter design
- Time Scope, Spectrum Analyzer, and Logic Analyzer for visualizing and measuring streaming signals
- Fixed-point modeling and simulation of signal processing algorithms
- Support for C and C++ code generation
- Support for HDL code generation

Configure Simulink Environment for Signal Processing Models

In this section...

Installation

Before you begin working, you need to install the product on your computer.

Installing the DSP System Toolbox Software

The DSP System Toolbox software follows the same installation procedure as the MATLAB toolboxes.

Installing Online Documentation

Installing the documentation is part of the installation process:

- Installation from a DVD — Start the MathWorks® installer. When prompted, select the **Product** check boxes for the products you want to install. The documentation is installed along with the products.
- Installation from a Web download — If you update the DSP System Toolbox software using a Web download and you want to view the documentation with the MATLAB Help browser, you must install the documentation on your hard drive.

  Download the files from the Web. Then, start the installer, and select the **Product** check boxes for the products you want to install. The documentation is installed along with the products.

Required Products

The DSP System Toolbox product is part of a family of MathWorks products. You need to install several products to use the toolbox. For more information about the required products, see the MathWorks Web site, at `http://www.mathworks.com/products/dsp-system/requirements.html`.

Related Products

MathWorks provides several products that are relevant to the kinds of tasks you can perform with DSP System Toolbox software.

For more information about any of these products, see either

- The online documentation for that product if it is installed on your system
- The MathWorks Web site, at `http://www.mathworks.com/products/dsp-system/related.html`.

ARM Cortex-M and ARM Cortex-A Optimization

The DSP System Toolbox supports optimized C code generation for popular algorithms like FIR filtering and FFT on ARM Cortex-M and ARM Cortex-A processors.

You can generate C code that can be linked with the CMSIS library or calls the Ne10 library functions and compiled to provide optimized executables to run on ARM Cortex-M or ARM Cortex-A processors.

To use the DSP System Toolbox support packages for ARM Cortex-M and ARM Cortex-A processors, you must have the following products in addition to the DSP System Toolbox: Simulink, Simulink Coder™ , Embedded Coder®, and MATLAB Coder.

To obtain more information and download the DSP System Toolbox support packages for the ARM Cortex processors, see http://www.mathworks.com/hardware-support/index.html.

Configure the Simulink Environment for Signal Processing Models

In this section...

About DSP Simulink Model Templates

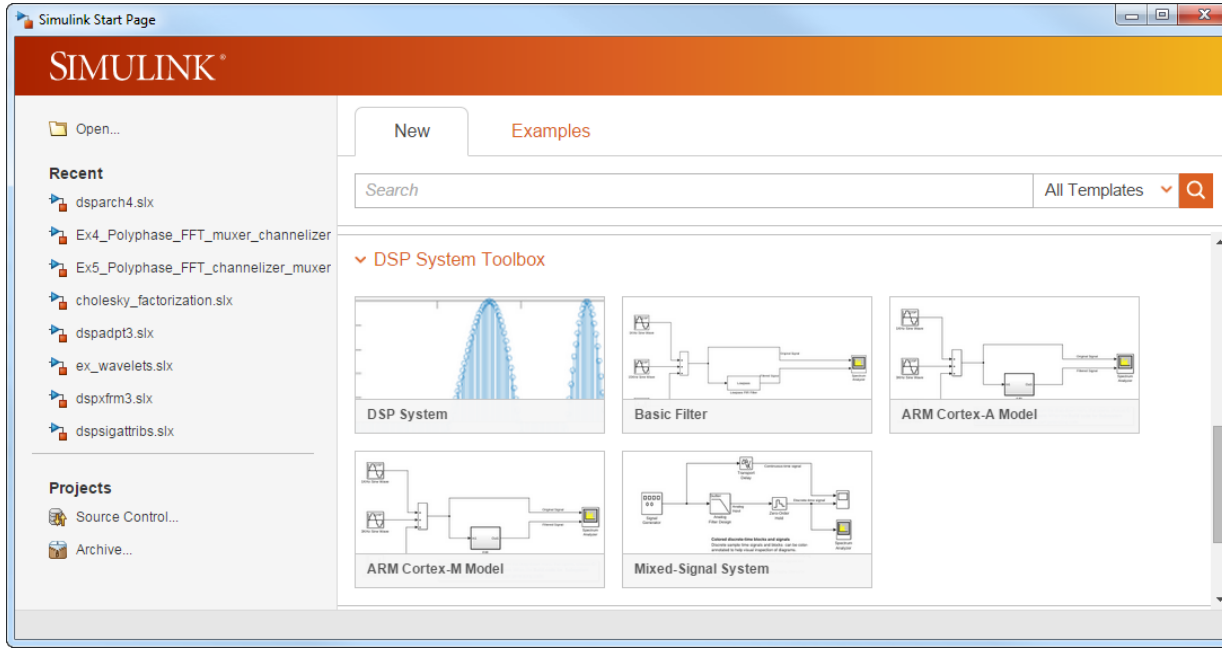
The DSP Simulink model templates let you automatically configure the Simulink environment with the recommended settings for digital signal processing modeling. DSP Simulink model templates enable reuse of settings, including configuration parameters. You can create models from templates that use best practices and take advantage of previous solutions to common problems. Instead of the default canvas of a new model, select a template model to help you get started.

For more information on Simulink model templates, see "Create a Model" (Simulink).

Create Model Using the DSP System Toolbox Simulink Model Template

To create a new blank model and open the library browser:

1. On the MATLAB **Home** tab, click **Simulink**.
2. Click on **DSP System** to create an empty model with settings suitable for use with DSP System Toolbox. The new model opens. To access the library browser, click the **Library Browser** button on the model toolbar.



The new model using the template settings and contents appears in the Simulink Editor. The model is only in memory until you save it.

DSP Simulink Model Templates

When you create a model by choosing one of the DSP Simulink model templates, the model is configured to use the settings recommended for DSP System Toolbox. Some of these settings are:

Configuration Parameter	Setting
SingleTaskRateTransMsg	`error`
multiTaskRateTransMsg	`error`
Solver	`fixedstepdiscrete`
EnableMultiTasking	`Off`
StartTime	`0.0`

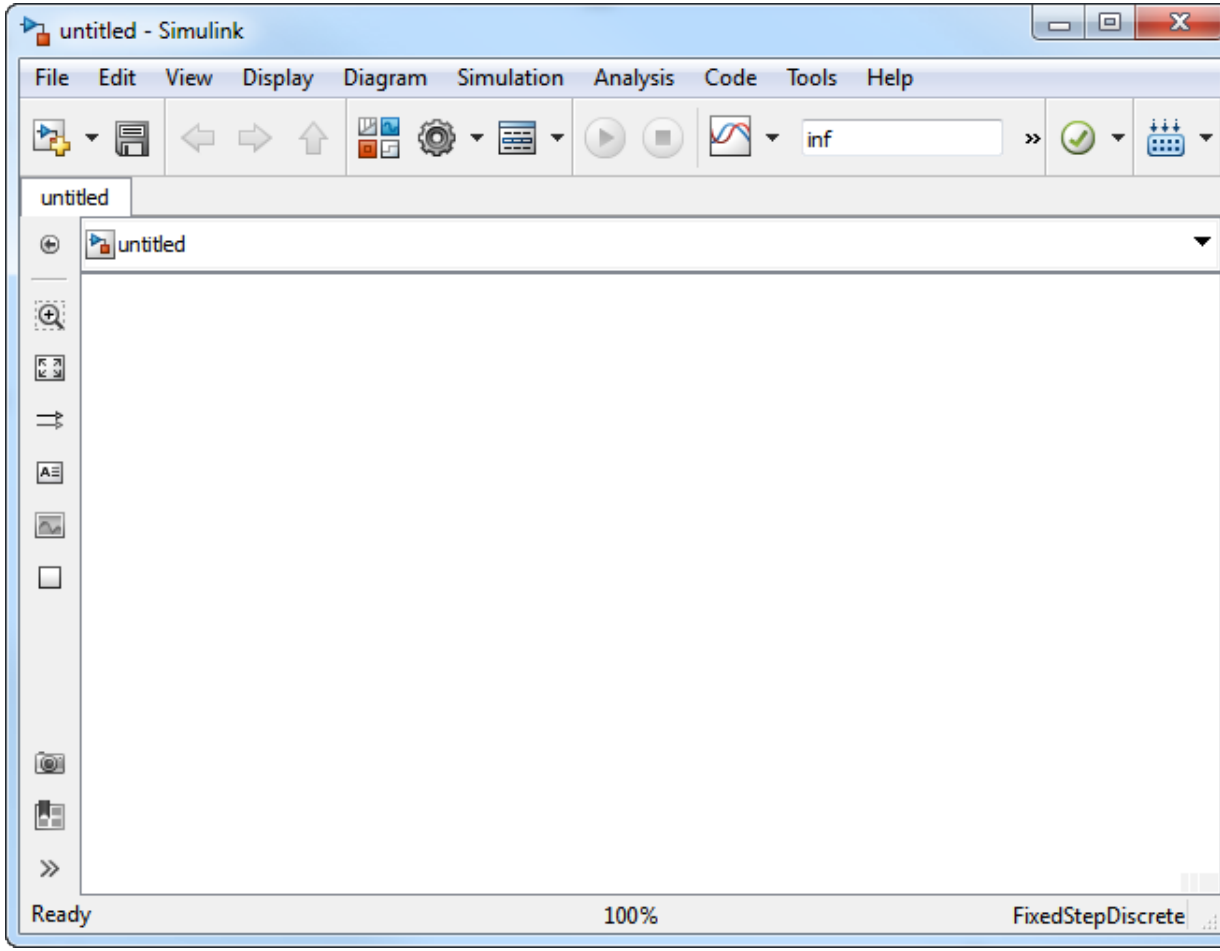
Configuration Parameter	Setting
StopTime	`inf`
FixedStep	`auto`
SaveTime	`off`
SaveOutput	`off`
AlgebraicLoopMsg	`error`
SignalLogging	`off`
FrameProcessingCompatibilityMsg	`error`

The DSP Simulink model templates are:

- “DSP System Template” on page 1-8
- “Basic Filter Template” on page 1-9
- “ARM Cortex-A Model Template” on page 1-10
- “ARM Cortex-M Model Template” on page 1-11
- “Mixed-Signal System Template” on page 1-12

DSP System Template

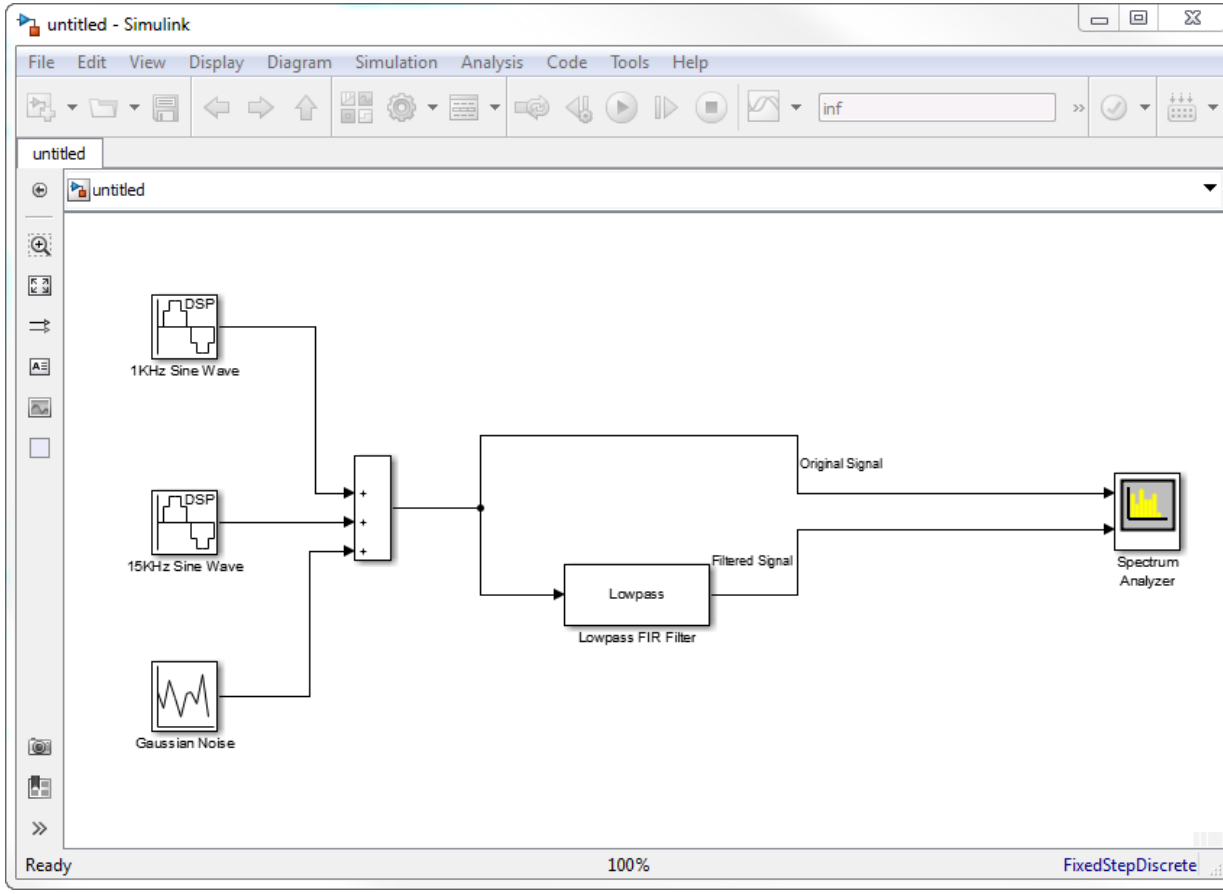
Click on **DSP System** to create a blank model configured with settings recommended for DSP System Toolbox.



Basic Filter Template

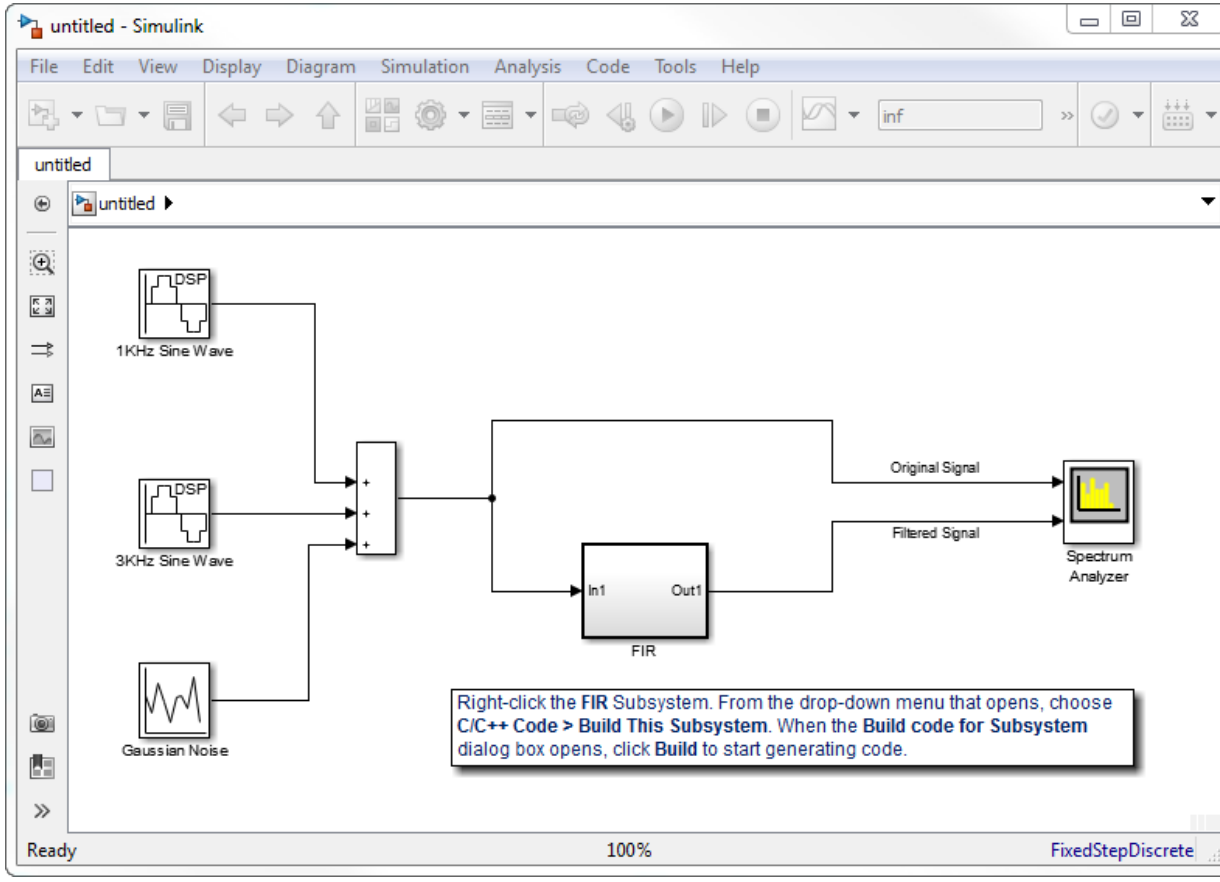
Click on **Basic Filter** to create a basic filtering model configured with settings recommended for DSP System Toolbox.

This model implements a low pass filter and enables you to compare the filtered signal with the original signal. The model acts as a starting point for modeling filtering algorithms in Simulink using DSP System Toolbox.



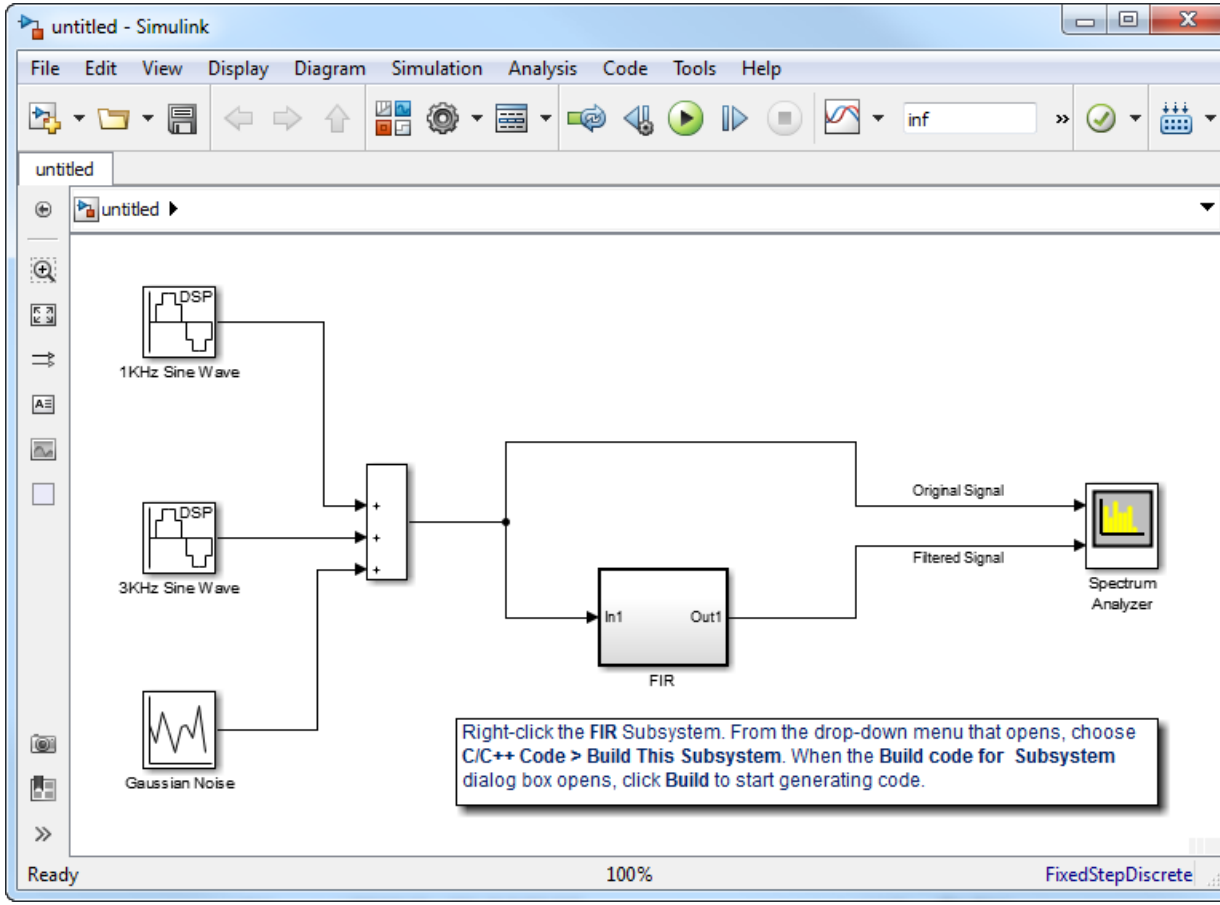
ARM Cortex-A Model Template

Click the **ARM Cortex-A Model** to create a basic model for code replacement with the Ne10 library, which is optimized for ARM Cortex-A Processors. This model is configured with settings recommended for DSP System Toolbox and ARM Cortex-A processors.



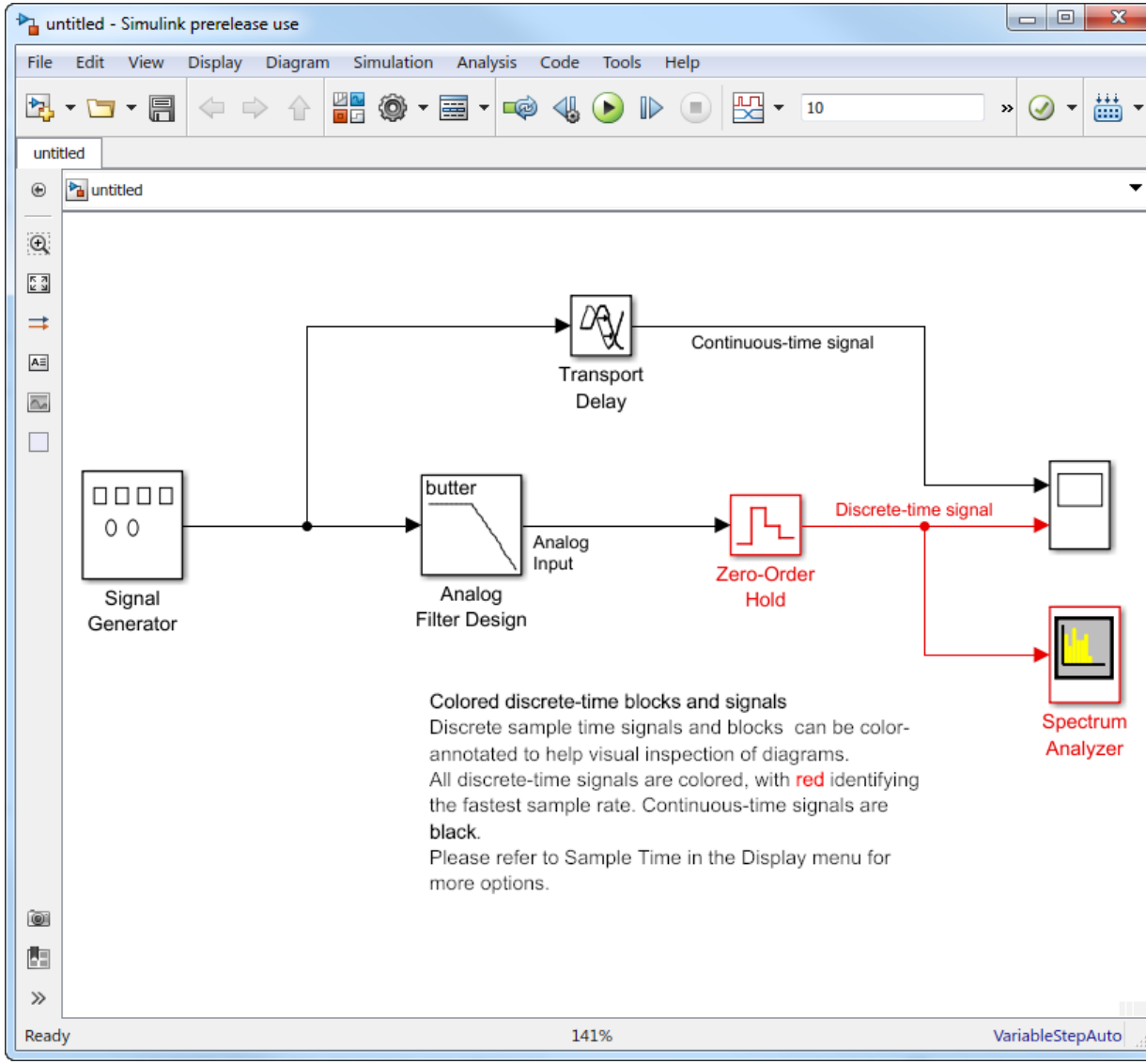
ARM Cortex-M Model Template

Click the **ARM Cortex-M Model** to create a basic model for code replacement with the CMSIS library, which is optimized for ARM Cortex-M processors. This model is configured with settings recommended for DSP System Toolbox and ARM Cortex-M processors.



Mixed-Signal System Template

Click the **Mixed-Signal System** template to create a basic A/D converter model configured with settings recommended for DSP System Toolbox and mixed-signal systems. This model performs A/D conversion by implementing an analog antialiasing filter followed by a zero-order hold circuit. The model acts as a starting point for modeling mixed-signal systems in Simulink using DSP System Toolbox. All discrete-time signals are colored in red to indicate the fastest sample rate. Continuous-time signals are colored in black. For additional sample time options, from the **Display** menu, select **Sample Time**.



See Also

- "Filter Frames of a Noisy Sine Wave Signal in Simulink"

Design a Filter with fdesign and Filter Builder

Filter Design Process Overview

Note: You must have the Signal Processing Toolbox installed to use `fdesign` and `filterBuilder`. Advanced capabilities are available if your installation additionally includes the DSP System Toolbox license. You can verify the presence of both toolboxes by typing `ver` at the command prompt.

Filter design through user-defined specifications is the core of the `fdesign` approach. This specification-centric approach places less emphasis on the choice of specific filter algorithms, and more emphasis on performance during the design a good working filter. For example, you can take a given set of design parameters for the filter, such as a stopband frequency, a passband frequency, and a stopband attenuation, and— using these parameters— design a specification object for the filter. You can then implement the filter using this specification object. Using this approach, it is also possible to compare different algorithms as applied to a set of specifications.

There are two distinct objects involved in filter design:

- Specification Object — Captures the required design parameters of a filter
- Implementation Object — Describes the designed filter; includes the array of coefficients and the filter structure

The distinction between these two objects is at the core of the filter design methodology. The basic attributes of each of these objects are outlined in the following table.

Specification Object	Implementation Object
High-level specification	Filter coefficients
Algorithmic properties	Filter structure

You can run the code in the following examples from the Help browser (select the code, right-click the selection, and choose **Evaluate Selection** from the context menu), or you can enter the code on the MATLAB command line. Before you begin this example, start MATLAB and verify that you have installed the Signal Processing Toolbox software. If you wish to access the full functionality of `fdesign` and `filterBuilder`, you should additionally obtain the DSP System Toolbox software. You can verify the presence of these products by typing `ver` at the command prompt.

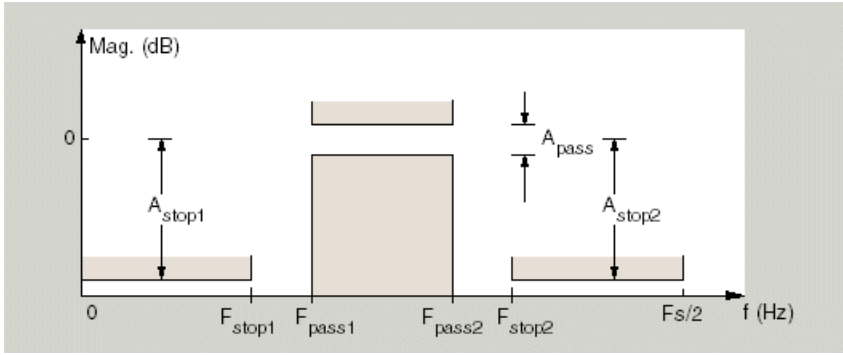
Design a Filter Using fdesign

Use the following two steps to design a simple filter.

1. Create a filter specification object.
2. Design your filter.

Design a Filter in Two Steps

Assume that you want to design a bandpass filter. Typically a bandpass filter is defined as shown in the following figure.



In this example, a sampling frequency of `Fs` = 48 kHz is used. This bandpass filter has the following specifications, specified here using MATLAB code:

```
A_stop1 = 60;  % Attenuation in the first stopband = 60 dB
F_stop1 = 8400;  % Edge of the stopband = 8400 Hz
F_pass1 = 10800; % Edge of the passband = 10800 Hz
F_pass2 = 15600; % Closing edge of the passband = 15600 Hz
F_stop2 = 18000; % Edge of the second stopband = 18000 Hz
A_stop2 = 60;  % Attenuation in the second stopband = 60 dB
A_pass = 1;  % Amount of ripple allowed in the passband = 1 dB
```

In the following two steps, these specifications are passed to the `fdesign.bandpass` method as parameters.

Step 1

To create a filter specification object, evaluate the following code at the MATLAB prompt:

```
d = fdesign.bandpass
```

Now, pass the filter specifications that correspond to the default `Specification` — `fst1`,`fp1`,`fp2`,`fst2`,`ast1`,`ap`,`ast2`. This example adds `fs` as the final input argument to specify the sampling frequency of 48 kHz.

```
>> BandPassSpecObj = ...
   fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2', ...
  F_stop1, F_pass1, F_pass2, F_stop2, A_stop1, A_pass, ...
  A_stop2, 48000)
```

Note: The order of the filter is not specified, allowing a degree of freedom for the algorithm design in order to achieve the specification. The design will be a minimum order design.

The specification parameters, such as `Fstop1`, are all given default values when none are provided. You can change the values of the specification parameters after the filter specification object has been created. For example, if there are two values that need to be changed, `Fpass2` and `Fstop2`, use the `set` command, which takes the object first, and then the parameter value pairs. Evaluate the following code at the MATLAB prompt:

```
>> set(BandPassSpecObj, 'Fpass2', 15800, 'Fstop2', 18400)
```

`BandPassSpecObj` is the new filter specification object which contains all the required design parameters, including the filter type.

You may also change parameter values in filter specification objects by accessing them as if they were elements in a `struct` array.

```
>> BandPassSpecObj.Fpass2=15800;
```

Step 2

Design the filter by using the `design` command. You can access the design methods available for you specification object by calling the `designmethods` function. For example, in this case, you can execute the command

```
>> designmethods(BandPassSpecObj)
```

```
Design Methods for class
fdesign.bandpass (Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2):


butter
cheby1
cheby2
ellip
equiripple
kaiserwin
```

After choosing a design method use, you can evaluate the following at the MATLAB prompt (this example assumes you've chosen '`equiripple`'):

```
>> BandPassFilt = design(BandPassSpecObj, 'equiripple')

BandPassFilt =

     FilterStructure: 'Direct-Form FIR'
          Arithmetic: 'double'
           Numerator: [1x44 double]
    PersistentMemory: false
```

If you have the DSP System Toolbox installed, you can also design your filter with a filter System object™. To create a filter System object with the same specification object `BandPassSpecObj`, you can execute the commands

```
>> designmethods(BandPassSpecObj,...
'SystemObject',true)


Design Methods that support System objects for class
fdesign.bandpass (Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2):


butter
cheby1
cheby2
ellip
equiripple
kaiserwin
```

```
>> BandPassFiltSysObj = design(BandPassSpecObj,...
'equiripple','SystemObject',true)


  System: dsp.FIRFilter

  Properties:
              Structure: 'Direct form'
        NumeratorSource: 'Property'
              Numerator: [1x44 double]
      InitialConditions: 0
    FrameBasedProcessing: true

  Show fixed-point properties
```

Available design methods and design options for filter System objects are not necessarily the same as those for filter objects.

Note: If you do not specify a design method, a default method will be used. For example, you can execute the command

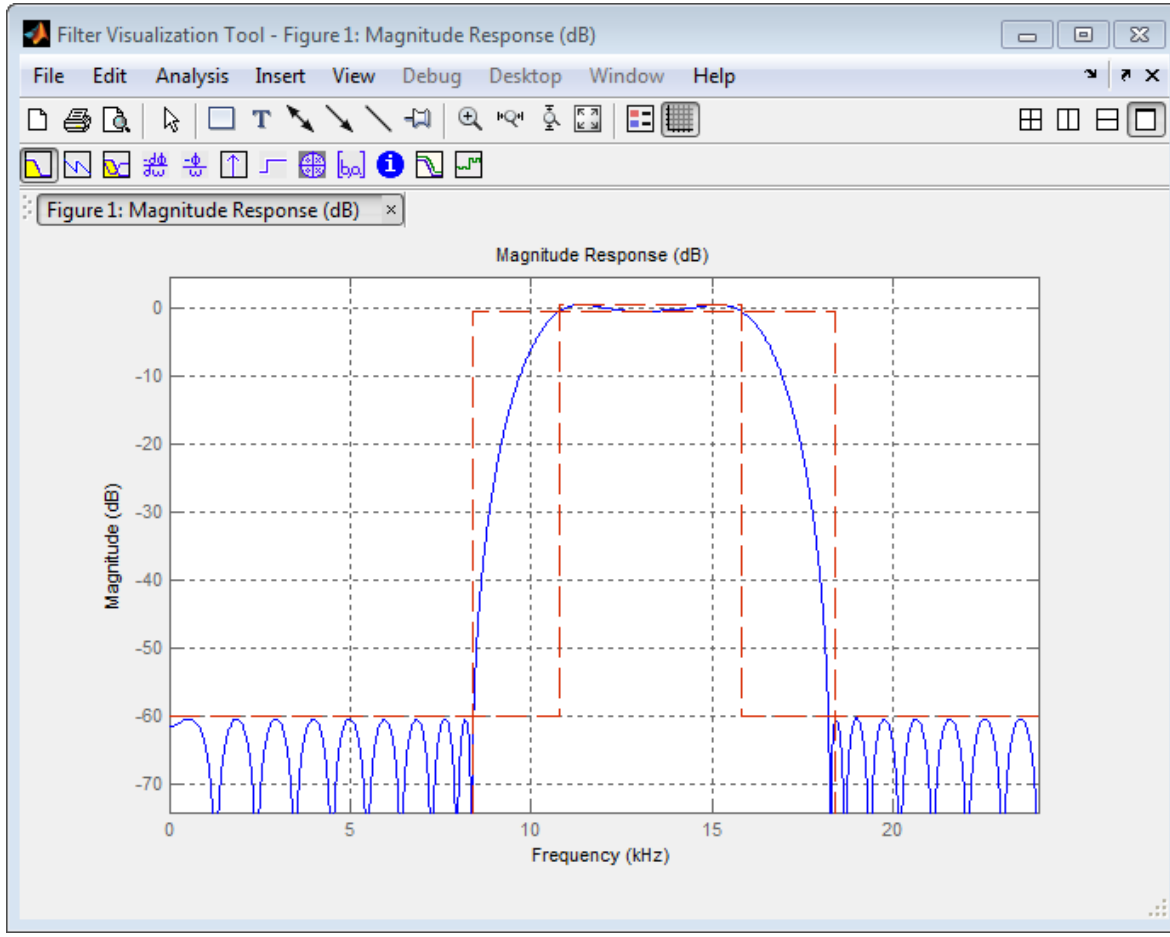
```
>> BandPassFilt = design(BandPassSpecObj)

BandPassFilt =

     FilterStructure: 'Direct-Form FIR'
          Arithmetic: 'double'
           Numerator: [1x44 double]
    PersistentMemory: false
```

and a design method will be selected automatically.

To check your work, you can plot the filter magnitude response using the Filter Visualization tool. Verify that all the design parameters are met:

```
>> fvtool(BandPassFilt) %plot the filter magnitude response
```

If you have the DSP System Toolbox installed, the Filter Visualization tool produces the following figure with the dashed red lines indicating the transition bands and unity gain (0 in dB) over the passband.


Filter Visualization Tool - Figure 1: Magnitude Response (dB)
File Edit Analysis Insert View Debug Desktop Window Help
Figure 1: Magnitude Response (dB)
Magnitude Response (dB)
Magnitude (dB)
0
-10
-20
-30
-40
-50
-60
-70
0
5
10
15
20
Frequency (kHz)

Design a Filter Using Filter Builder

Filter Builder presents the option of designing a filter using a GUI dialog box as opposed to the command line instructions. You can use Filter Builder to design the same bandpass filter designed in the previous section, “Design a Filter Using fdesign” on page 2-3

Design a Simple Filter in Filter Builder

To design the filter using the Filter Builder GUI:

1. Type the following at the MATLAB prompt:

   ```
   filterBuilder
   ```

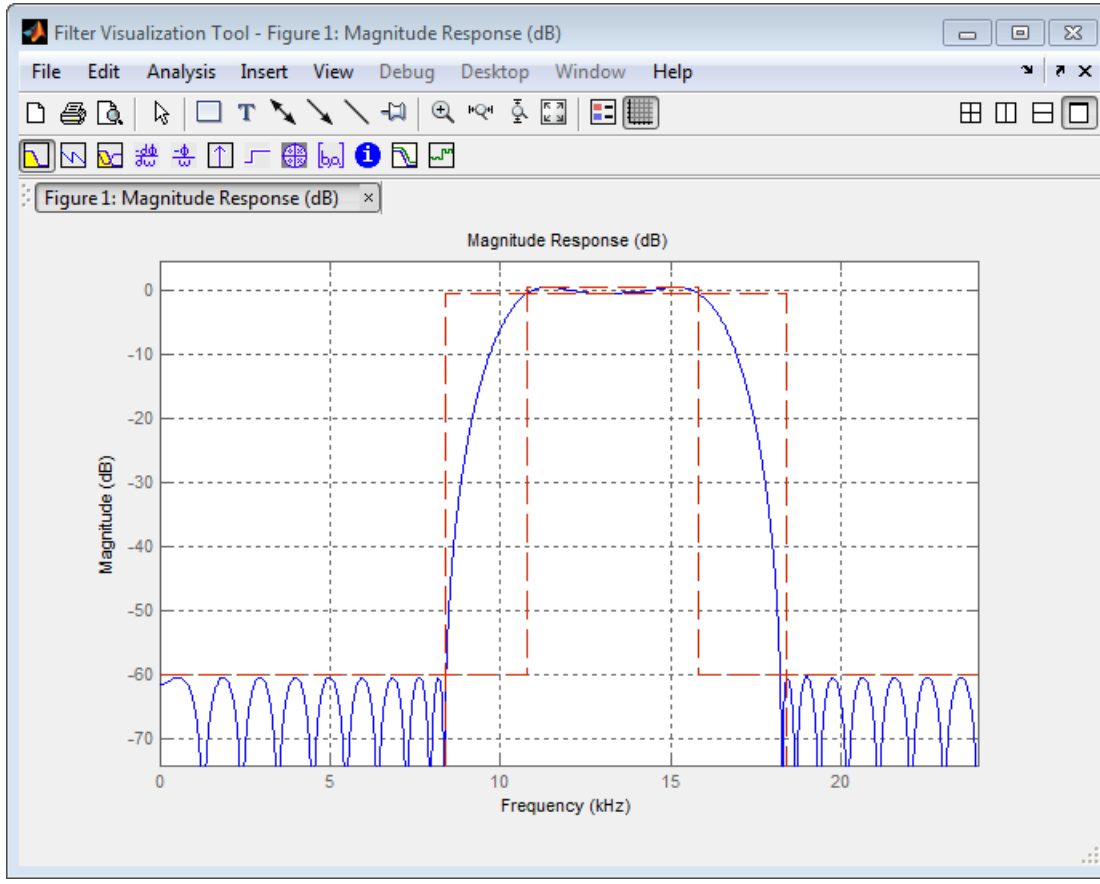
2. Select `Bandpass` filter response from the list in the dialog box, and hit the **OK** button.

3. Enter the correct frequencies for **Fpass2** and **Fstop2**, then click **OK**. Here the specification uses normalized frequency, so that the passband and stopband edges are expressed as a fraction of the Nyquist frequency (in this case, 48/2 kHz). The following message appears at the MATLAB prompt:

   ```
   The variable 'Hbp' has been exported to the command window.
   ```
   If you display the Workspace tab, you see the object `Hbp` has been placed on your workspace.

4. To check your work, plot the filter magnitude response using the Filter Visualization tool. Verify that all the design parameters are met:

   ```
   fvtool(Hbp) %plot the filter magnitude response
   ```



Note that the dashed red lines on the preceding figure will only appear if you are using the DSP System Toolbox software.

3

Design Filters in Simulink

Design and Implement a Filter

In this section...

Design a Digital Filter in Simulink

You can design lowpass, highpass, bandpass, and bandstop filters using either the Digital Filter Design block or the Filter Realization Wizard. These blocks are capable of calculating filter coefficients for various filter structures. In this section, you use the Digital Filter Design block to convert white noise to low frequency noise so you can simulate its effect on your system.

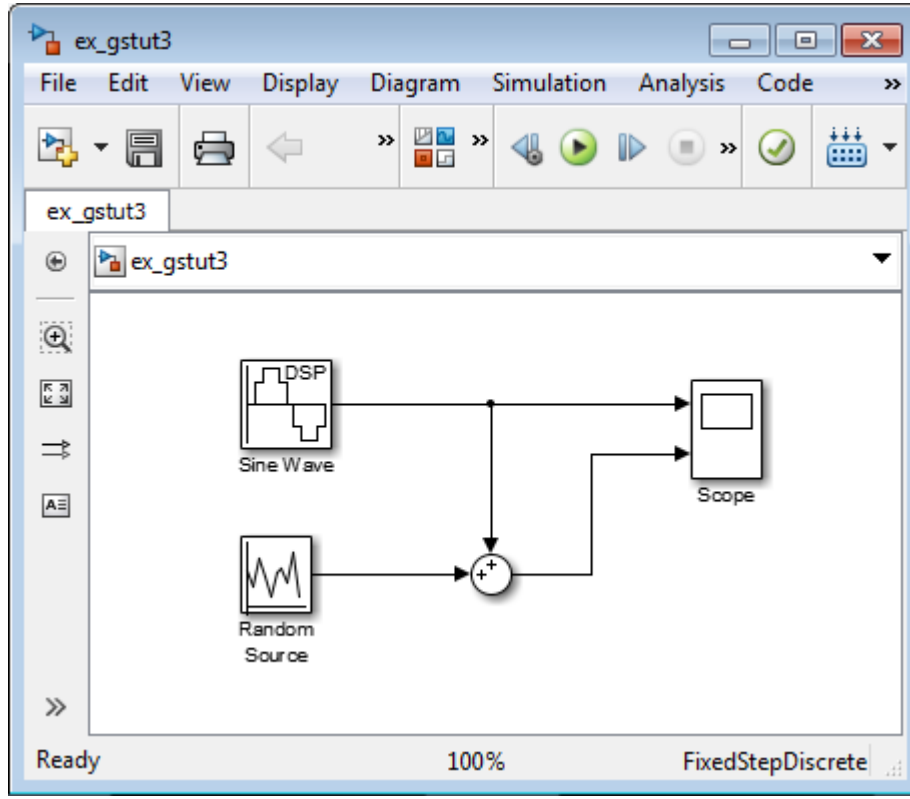
As a practical application, suppose a pilot is speaking into a microphone within the cockpit of an airplane. The noise of the wind passing over the fuselage is also reaching the microphone. A sensor is measuring the noise of the wind on the outside of the plane. You want to estimate the wind noise inside the cockpit and subtract it from the input to the microphone so that only the pilot's voice is transmitted. In this chapter, you first learn how to model the low frequency noise that is reaching the microphone. Later, you learn how to remove this noise so that only the pilot's voice is heard.

In this topic, you use a Digital Filter Design block to create low frequency noise, which models the wind noise inside the cockpit:

1 Open the model by typing

```
ex_gstut3
```

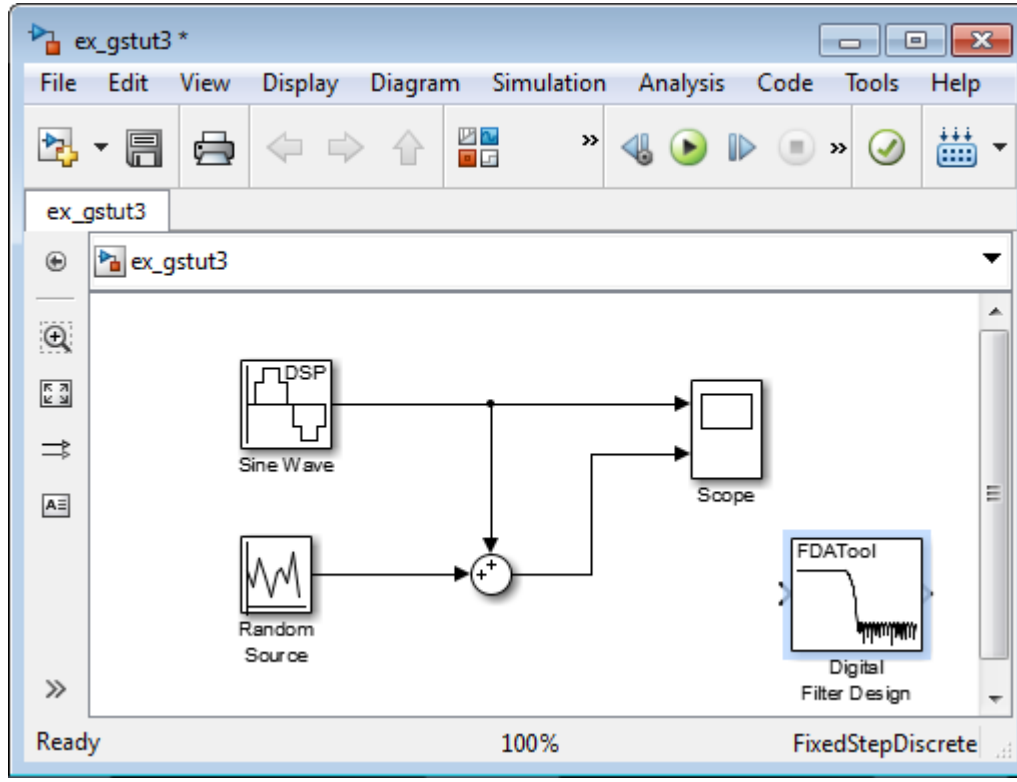
at the MATLAB command prompt. This model contains a Scope block that displays the original sine wave and the sine wave with white noise added.



2 Open the DSP System Toolbox library by typing `dsplib` at the MATLAB command prompt.

3 Convert white noise to low frequency noise by introducing a Digital Filter Design block into your model. In the airplane scenario, the air passing over the fuselage creates white noise that is measured by a sensor. The Random Source block models this noise. The fuselage of the airplane converts this white noise to low frequency noise, a type of colored noise, which is heard inside the cockpit. This noise contains only certain frequencies and is more difficult to eliminate. In this example, you model the low frequency noise using a Digital Filter Design block. This block uses the functionality of the Filter Design and Analysis Tool (FDATool) to design a filter.

   Double-click the Filtering library, and then double-click the Filter Implementations sublibrary. Click-and-drag the Digital Filter Design block into your model.

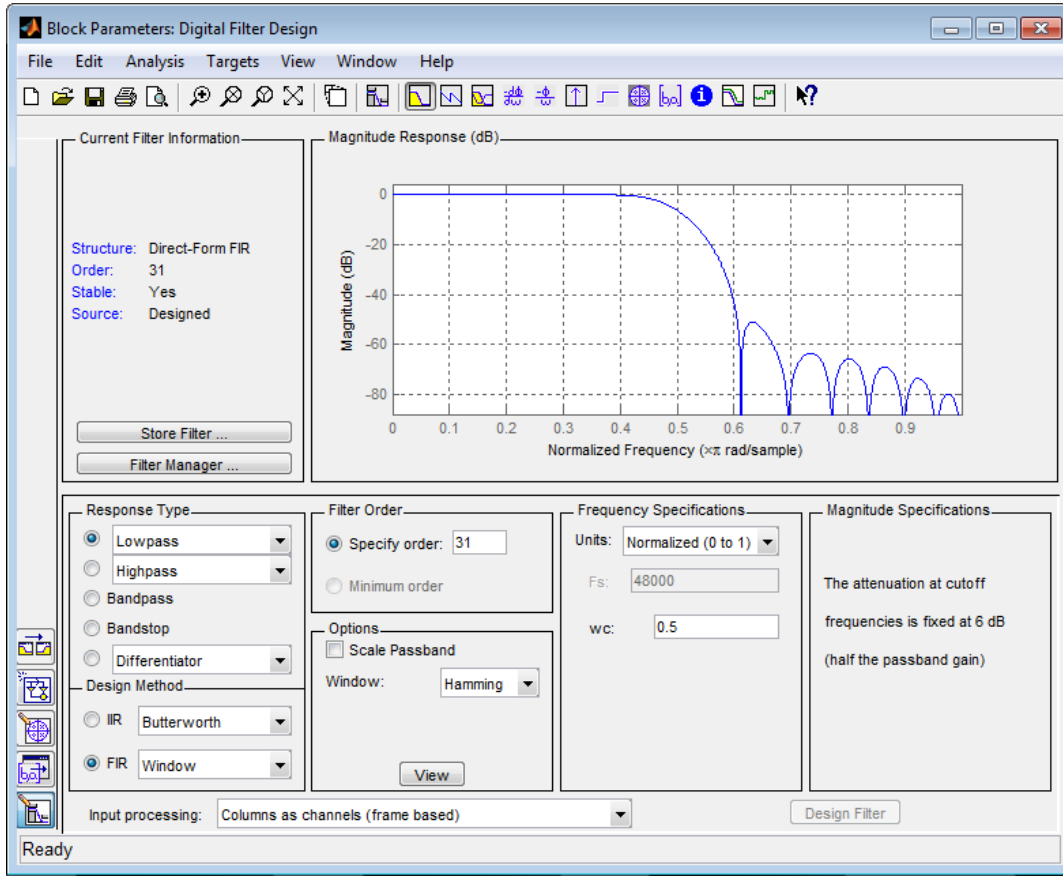


4 Set the Digital Filter Design block parameters to design a lowpass filter and create low frequency noise. Open the block parameters dialog box by double-clicking the block. Set the parameters as follows:

- **Response Type** = `Lowpass`
- **Design Method** = **FIR** and, from the list, choose `Window`
- **Filter Order** = **Specify order** and enter `31`
- **Scale Passband** — Cleared
- **Window** = `Hamming`
- **Units** = `Normalized (0 to 1)`
- **wc** = `0.5`

Based on these parameters, the Digital Filter Design block designs a lowpass FIR filter with 32 coefficients and a cutoff frequency of 0.5. The block multiplies the time-domain response of your filter by a 32 sample Hamming window.

5 Click **Design Filter** at the bottom center of the dialog box to view the magnitude response of your filter in the **Magnitude Response** pane. The Digital Filter Design dialog box should now look similar to the following figure.



You have now designed a digital lowpass filter using the Digital Filter Design block.

You can experiment with the Digital Filter Design block in order to design a filter of your own. For more information on the block functionality, see the Digital Filter Design block

reference page. For more information on the Filter Design and Analysis Tool, see “Filter Designer” (Signal Processing Toolbox).

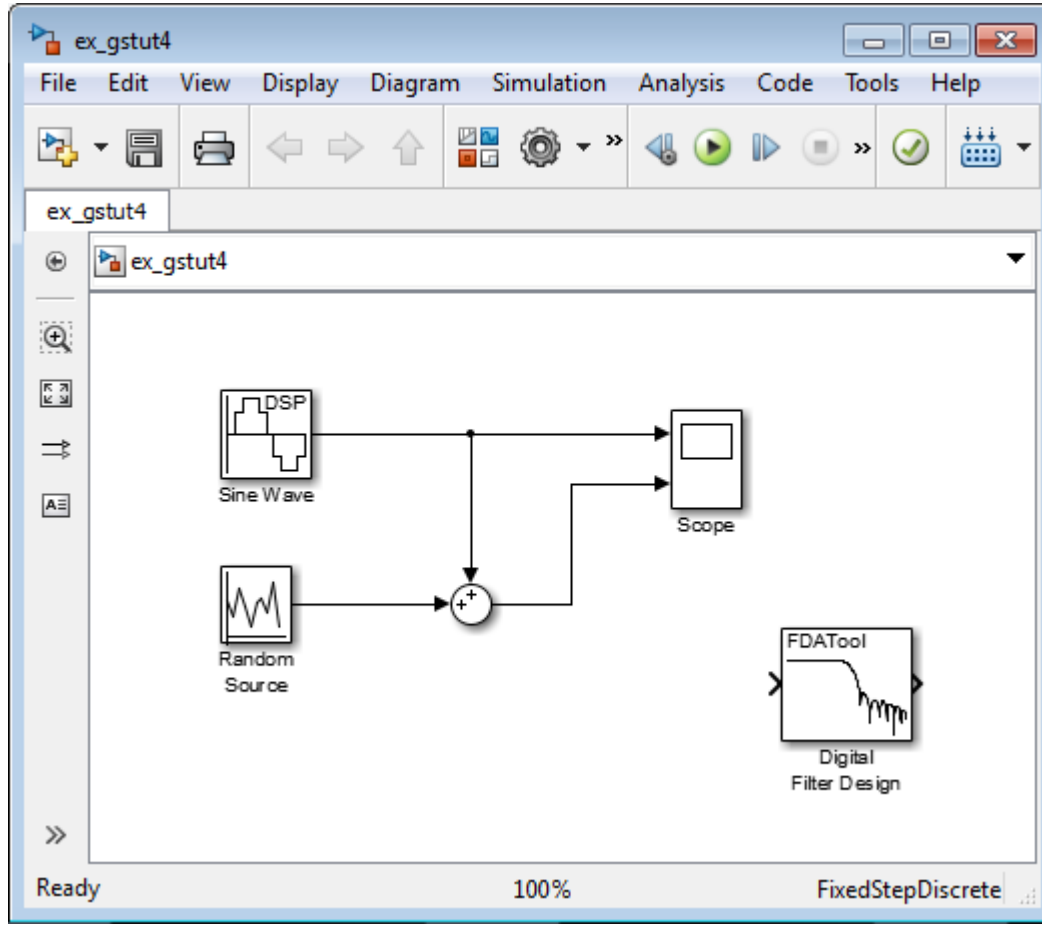
Add a Digital Filter to Your Model

In this topic, you add the lowpass filter you designed in “Design a Digital Filter in Simulink” on page 3-2 to your block diagram. Use this filter, which converts white noise to colored noise, to simulate the low frequency wind noise inside the cockpit:

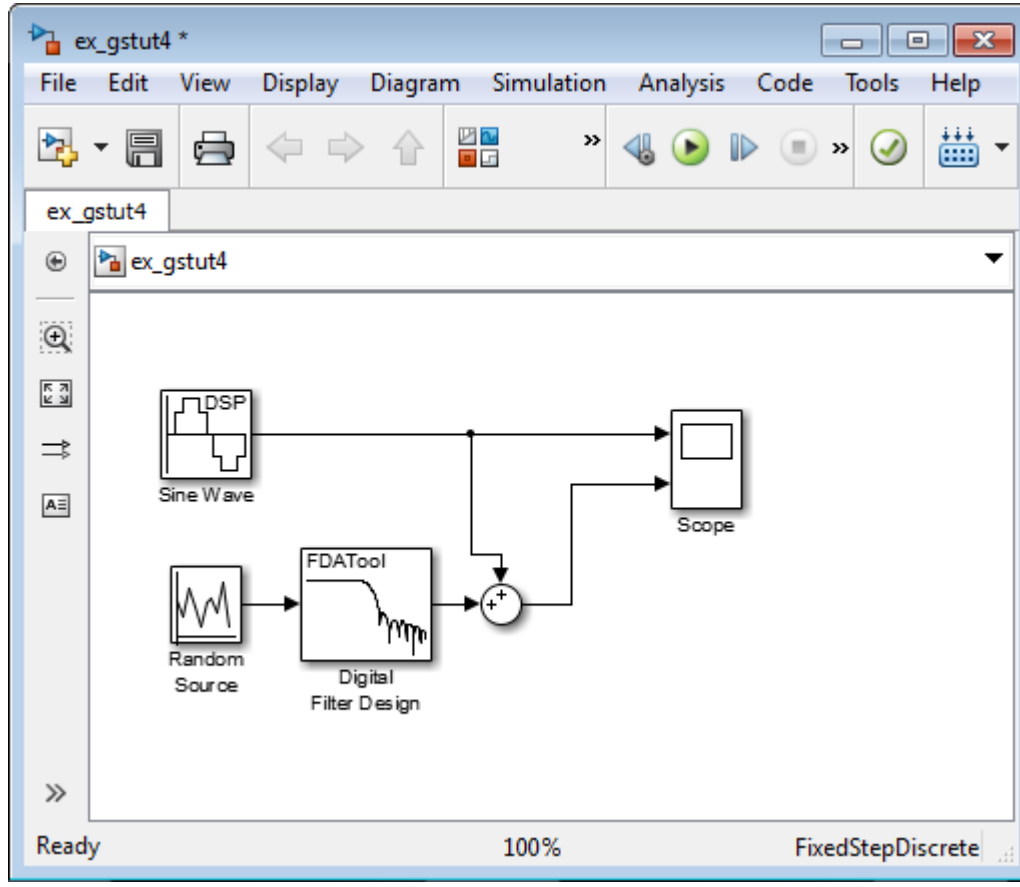
1 If the model you created in “Design a Digital Filter in Simulink” on page 3-2 is not open on your desktop, you can open an equivalent model by typing

```
ex_gstut4
```

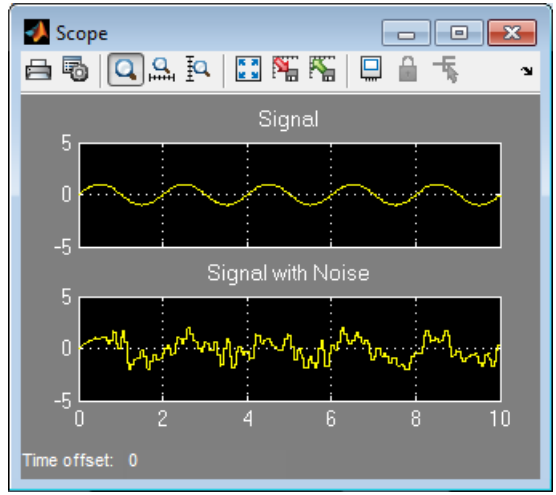
at the MATLAB command prompt.



2 Incorporate the Digital Filter Design block into your block diagram by placing it between the Random Source block and the Sum block.



3 Run your model and view the results in the Scope window. This window shows the original input signal and the signal with low frequency noise added to it.



You have now built a digital filter and used it to model the presence of colored noise in your signal. This is analogous to modeling the low frequency noise reaching the microphone in the cockpit of the aircraft. Now that you have added noise to your system, you can experiment with methods to eliminate it.

Adaptive Filters

In this section...

Design an Adaptive Filter in Simulink

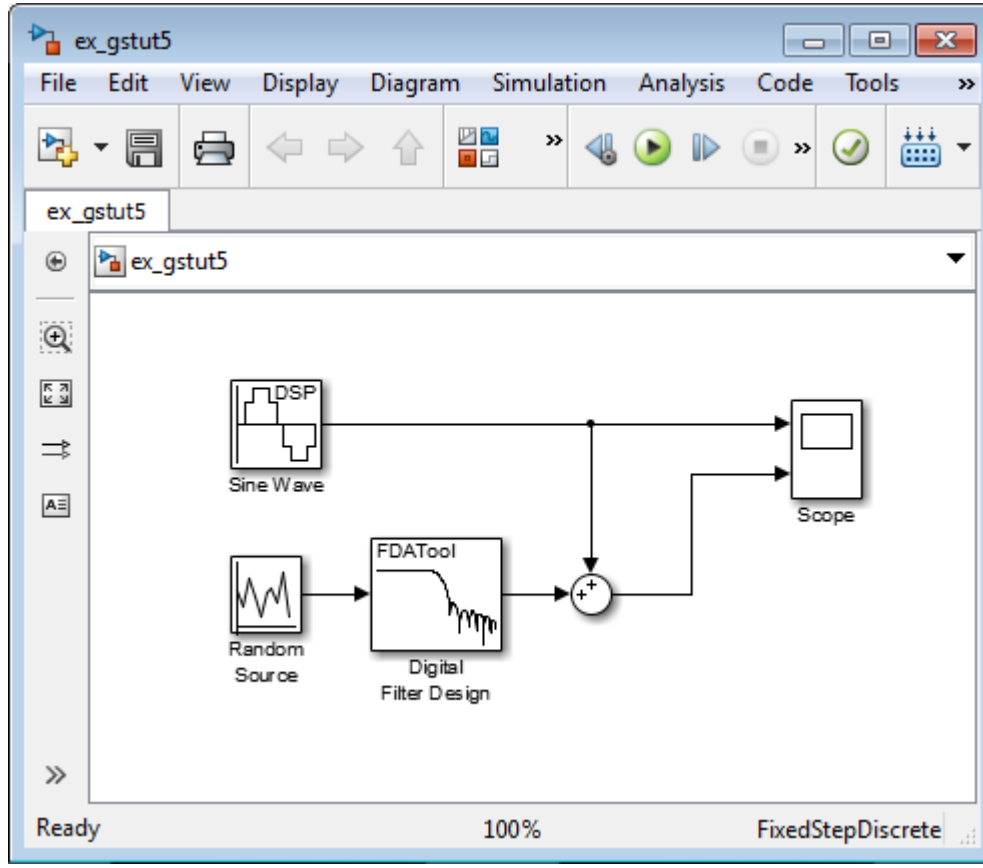
Adaptive filters track the dynamic nature of a system and allow you to eliminate time-varying signals. The DSP System Toolbox libraries contain blocks that implement least-mean-square (LMS), Block LMS, Fast Block LMS, and recursive least squares (RLS) adaptive filter algorithms. These filters minimize the difference between the output signal and the desired signal by altering their filter coefficients. Over time, the adaptive filter's output signal more closely approximates the signal you want to reproduce.

In this topic, you design an LMS adaptive filter to remove the low frequency noise in your signal:

1 If the model you created in “Add a Digital Filter to Your Model” on page 3-6 is not open on your desktop, you can open an equivalent model by typing

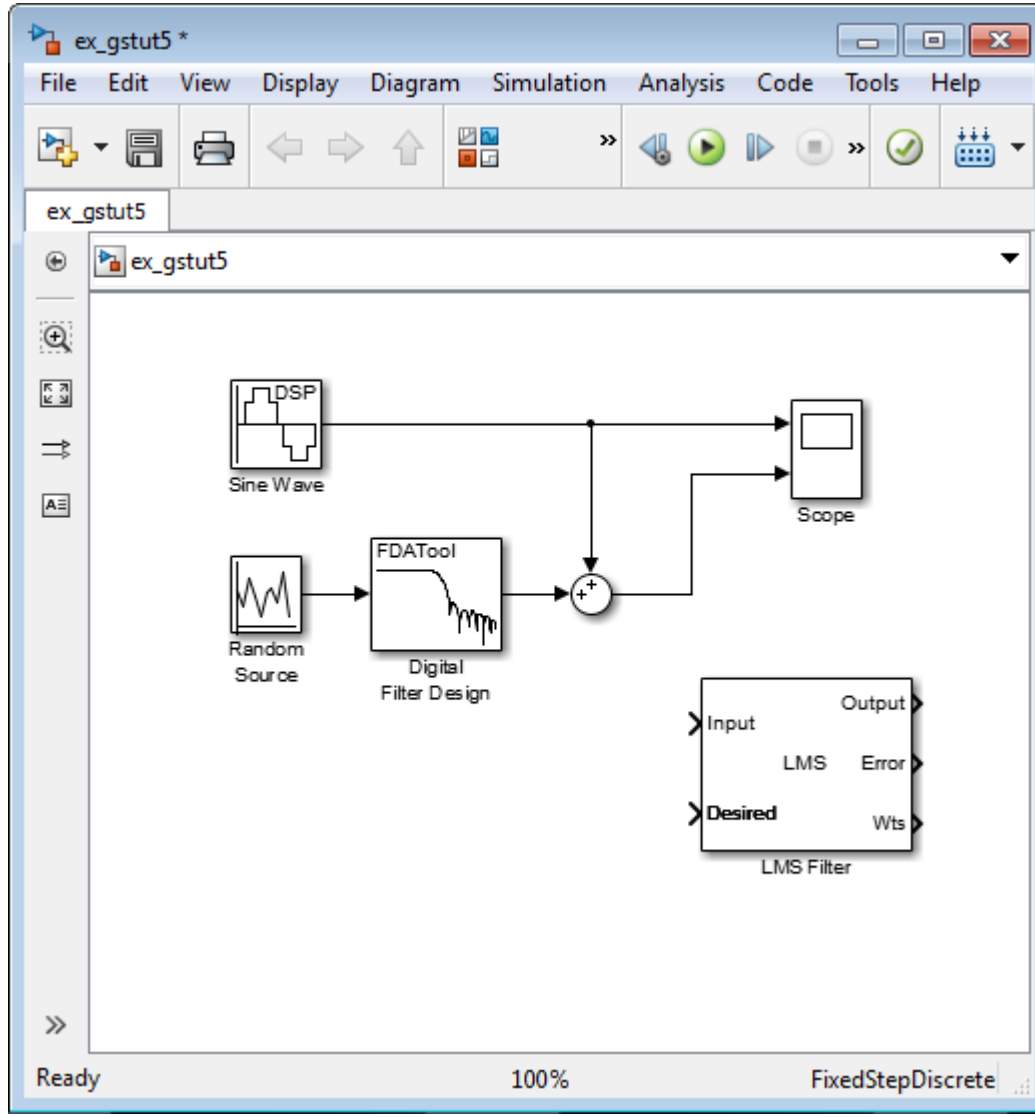
```
ex_gstut5
```

at the MATLAB command prompt.



2 Open the DSP System Toolbox library by typing `dsplib` at the MATLAB command prompt.

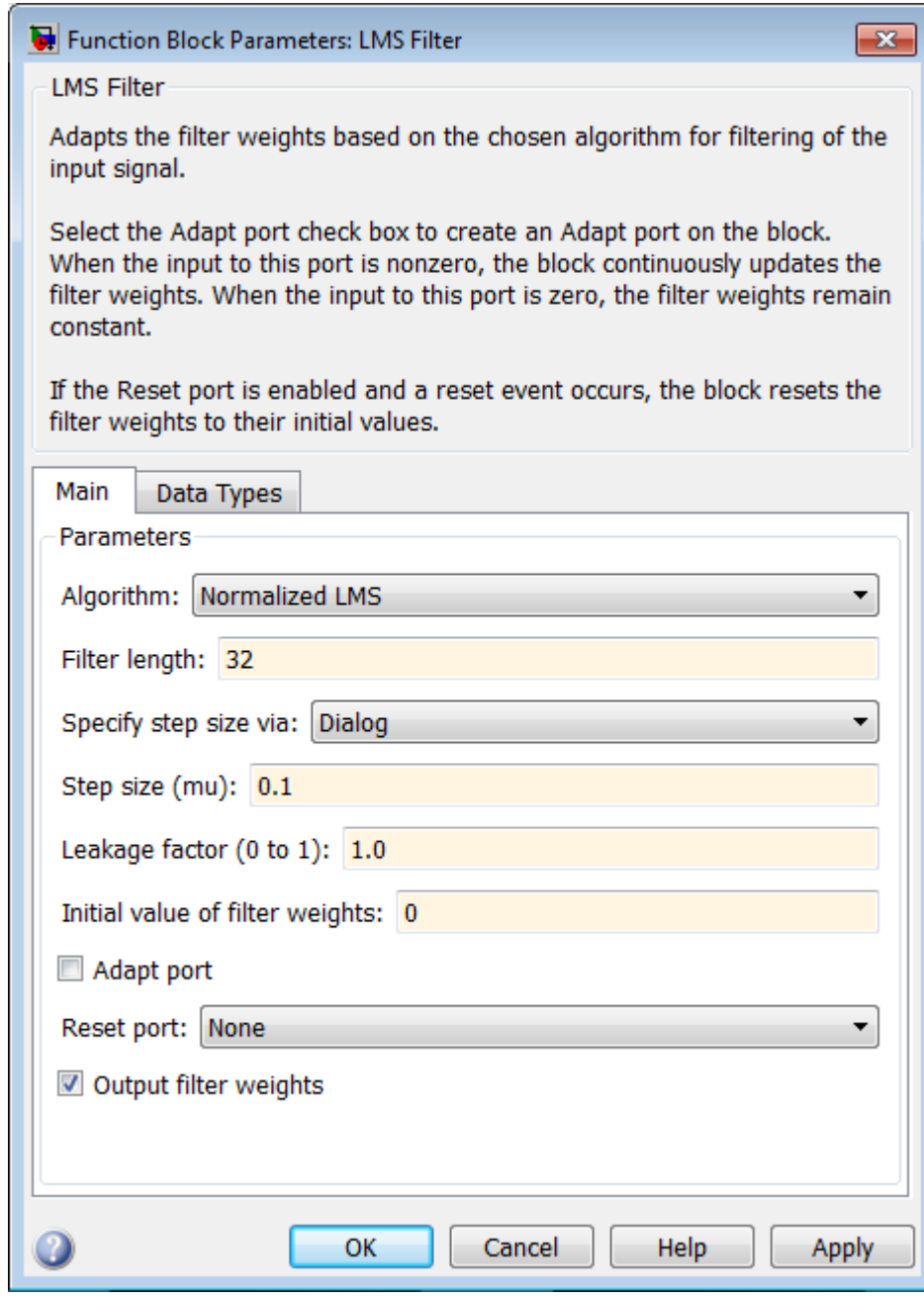
3 Remove the low frequency noise from your signal by adding an LMS Filter block to your system. In the airplane scenario, this is equivalent to subtracting the wind noise inside the cockpit from the input to the microphone. Double-click the Filtering sublibrary, and then double-click the Adaptive Filters library. Add the LMS Filter block into your model.



4 Set the LMS Filter block parameters to model the output of the Digital Filter Design block. Open its dialog box by double-clicking the block. Set the block parameters as follows:

- **Algorithm** = `Normalized LMS`
- **Filter length** = `32`
- **Specify step size via** = `Dialog`
- **Step size (mu)** = `0.1`
- **Leakage factor (0 to 1)** = `1.0`
- **Initial value of filter weights** = `0`
- Clear the **Adapt port** check box.
- **Reset port** = `None`
- Select the **Output filter weights** check box.

The LMS Filter dialog box should now look like the following figure:


Function Block Parameters: LMS Filter
LMS Filter
Adapts the filter weights based on the chosen algorithm for filtering of the input signal.
Select the Adapt port check box to create an Adapt port on the block. When the input to this port is nonzero, the block continuously updates the filter weights. When the input to this port is zero, the filter weights remain constant.
If the Reset port is enabled and a reset event occurs, the block resets the filter weights to their initial values.
Main
Data Types
Parameters
Algorithm: Normalized LMS
Filter length: 32
Specify step size via: Dialog
Step size (mu): 0.1
Leakage factor (0 to 1): 1.0
Initial value of filter weights: 0
Adapt port
Reset port: None
Output filter weights
OK
Cancel
Help
Apply

5 Click **Apply**.

Based on these parameters, the LMS Filter block computes the filter weights using the normalized LMS equations. The filter order you specified is the same as the filter order of the Digital Filter Design block. The **Step size (mu)** parameter defines the granularity of the filter update steps. Because you set the **Leakage factor (0 to 1)** parameter to `1.0`, the current filter coefficient values depend on the filter's initial conditions and all of the previous input values. The initial value of the filter weights (coefficients) is zero. Since you selected the **Output filter weights** check box, the Wts port appears on the block. The block outputs the filter weights from this port.

Now that you have set the block parameters of the LMS Filter block, you can incorporate this block into your block diagram.

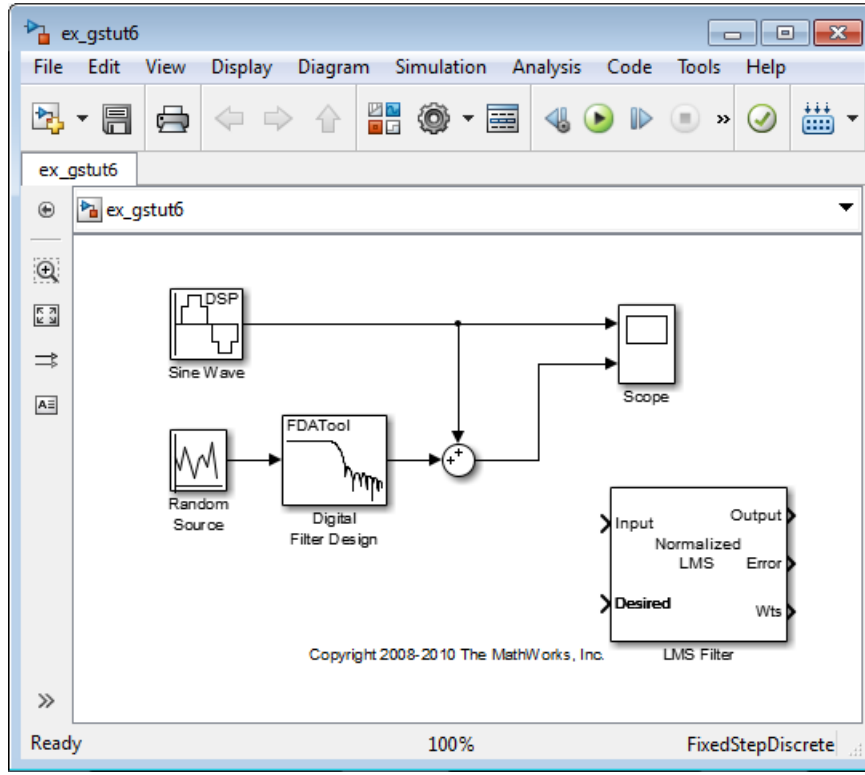
Add an Adaptive Filter to Your Model

In this topic, you recover your original sinusoidal signal by incorporating the adaptive filter you designed in “Design an Adaptive Filter in Simulink” on page 3-10 into your system. In the aircraft scenario, the adaptive filter models the low frequency noise heard inside the cockpit. As a result, you can remove the noise so that the pilot's voice is the only input to the microphone:

1 If the model you created in “Design an Adaptive Filter in Simulink” on page 3-10 is not open on your desktop, you can open an equivalent model by typing

```
ex_gstut6
```

at the MATLAB command prompt.



2 Add a Sum block to your model to subtract the output of the adaptive filter from the sinusoidal signal with low frequency noise. From the Simulink Math Operations library, drag a Sum block into your model. Open the Sum dialog box by double-clicking this block. Change the **List of signs** parameter to `|+-` and then click **OK**.

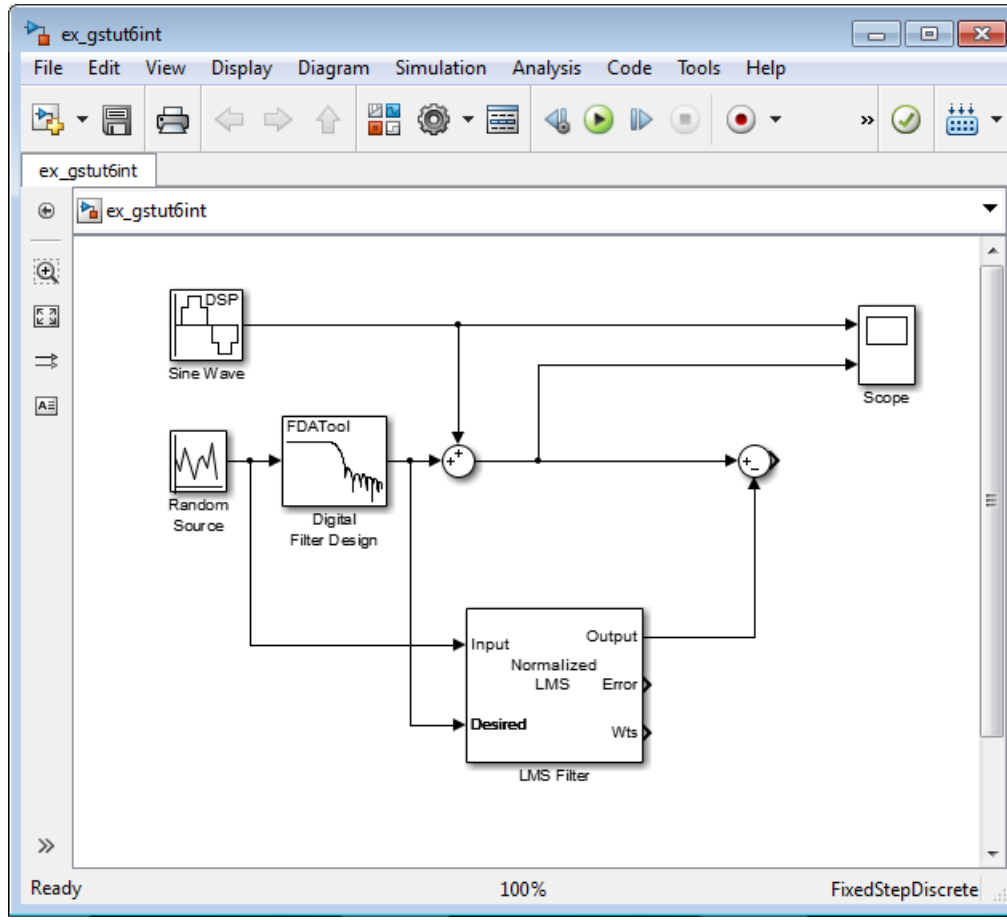
3 Incorporate the LMS Filter block into your system.

   a Connect the output of the Random Source block to the Input port of the LMS Filter block. In the aircraft scenario, the random noise is the white noise measured by the sensor on the outside of the airplane. The LMS Filter block models the effect of the airplane's fuselage on the noise.

   b Connect the output of the Digital Filter Design block to the Desired port on the LMS Filter block. This is the signal you want the LMS block to reproduce.

   c Connect the output of the LMS Filter block to the negative port of the Sum block you added in step 2.

d Connect the output of the first Sum block to the positive port of the second Sum block. Your model should now look similar to the following figure.



The positive input to the second Sum block is the sum of the input signal and the low frequency noise, $s(n) + y$. The negative input to the second Sum block is the LMS Filter block's best estimation of the low frequency noise, y'. When you subtract the two signals, you are left with an approximation of the input signal.

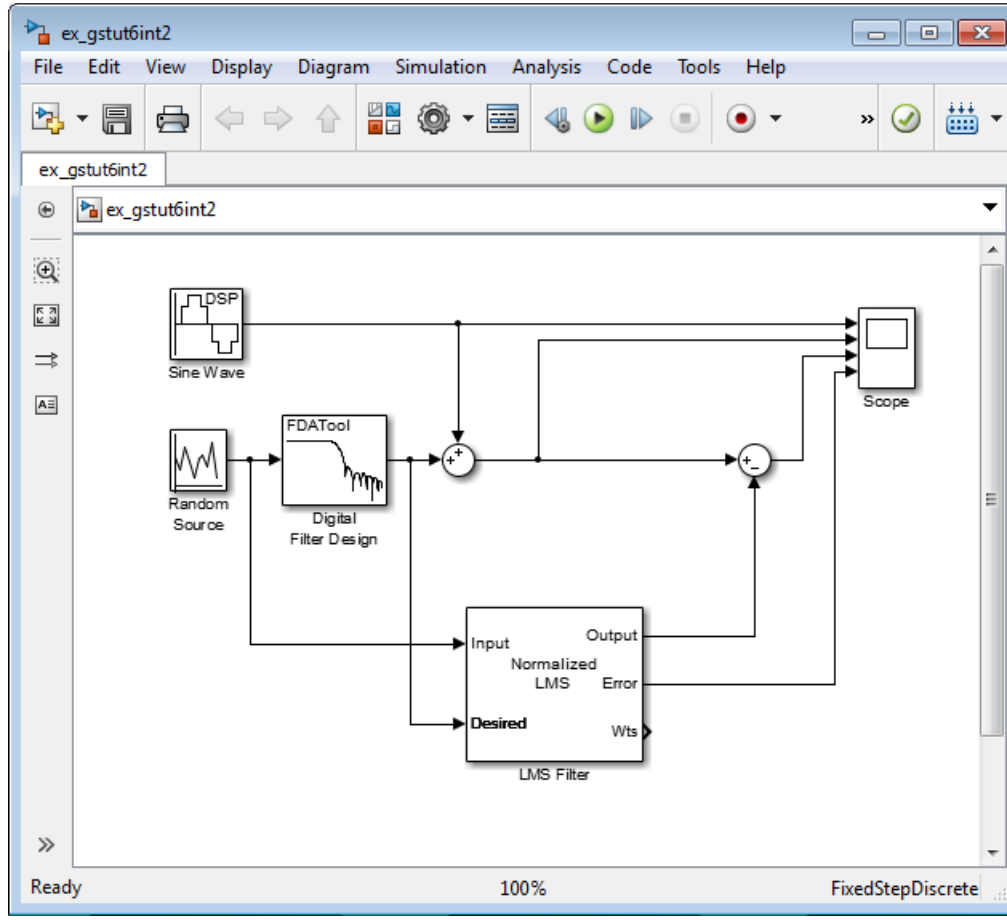
$$s(n)_{approx} = s(n) + y - y'$$

In this equation:

- $s(n)$ is the input signal
- $s(n)_{approx}$ is the approximation of the input signal
- y is the noise created by the Random Source block and the Digital Filter Design block
- y' is the LMS Filter block's approximation of the noise

Because the LMS Filter block can only approximate the noise, there is still a difference between the input signal and the approximation of the input signal. In subsequent steps, you set up the Scope block so you can compare the original sinusoidal signal with its approximation.

4. Add two additional inputs and axes to the Scope block. Open the Scope dialog box by double-clicking the Scope block. Click the **Parameters** button. For the **Number of axes** parameter, enter `4`. Close the dialog box by clicking **OK**.
5. Label the new Scope axes. In the Scope window, right-click on the third axes and select **Axes properties**. The Scope properties: axis 3 dialog box opens. In the **Title** box, enter `Approximation of Input Signal`. Close the dialog box by clicking **OK**. Repeat this procedure for the fourth axes and label it `Error`.
6. Connect the output of the second Sum block to the third port of the Scope block.
7. Connect the output of the Error port on the LMS Filter block to the fourth port of the Scope block. Your model should now look similar to the following figure.



In this example, the output of the Error port is the difference between the LMS filter's desired signal and its output signal. Because the error is never zero, the filter continues to modify the filter coefficients in order to better approximate the low frequency noise. The better the approximation, the more low frequency noise that can be removed from the sinusoidal signal. In the next topic, “View the Coefficients of Your Adaptive Filter” on page 3-20, you learn how to view the coefficients of your adaptive filter as they change with time.

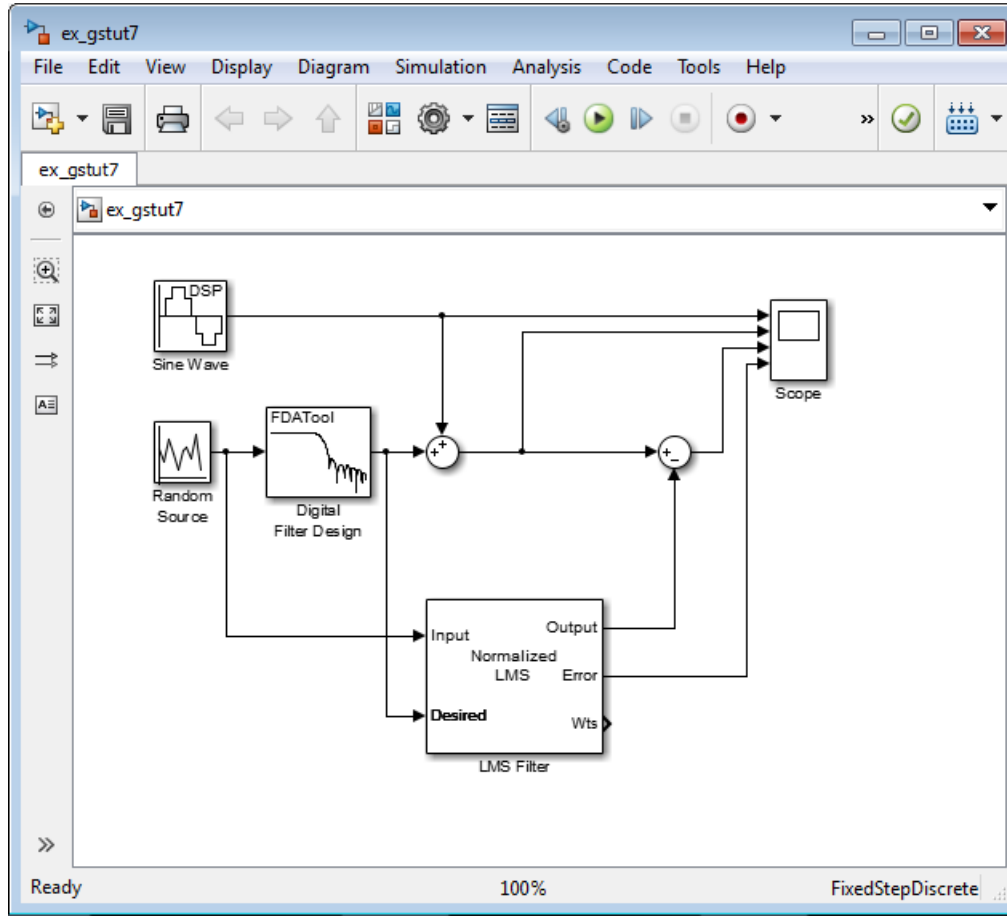
View the Coefficients of Your Adaptive Filter

The coefficients of an adaptive filter change with time in accordance with a chosen algorithm. Once the algorithm optimizes the filter's performance, these filter coefficients reach their steady-state values. You can view the variation of your coefficients, while the simulation is running, to see them settle to their steady-state values. Then, you can determine whether you can implement these values in your actual system:

1 If the model you created in “Add an Adaptive Filter to Your Model” on page 3-15 is not open on your desktop, you can open an equivalent model by typing

```
ex_gstut7
```

at the MATLAB command prompt. Note that the Wts port of the adaptive filter, which outputs the filter weights, still needs to be connected.



2 Open the DSP System Toolbox library by typing `dsplib` at the MATLAB command prompt.

3 View the filter coefficients using a Vector Scope block from the Sinks library.

4 Open the Vector Scope dialog box by double-clicking the block. Set the block parameters as follows:

   a Click the **Scope Properties** tab.

      - **Input domain** = `Time`

- **Time display span (number of frames)** = `1`

b Click the **Display Properties** tab.

- Select the following check boxes:
  - **Show grid**
  - **Frame number**
  - **Compact display**
  - **Open scope at start of simulation**

c Click the **Axis Properties** tab.

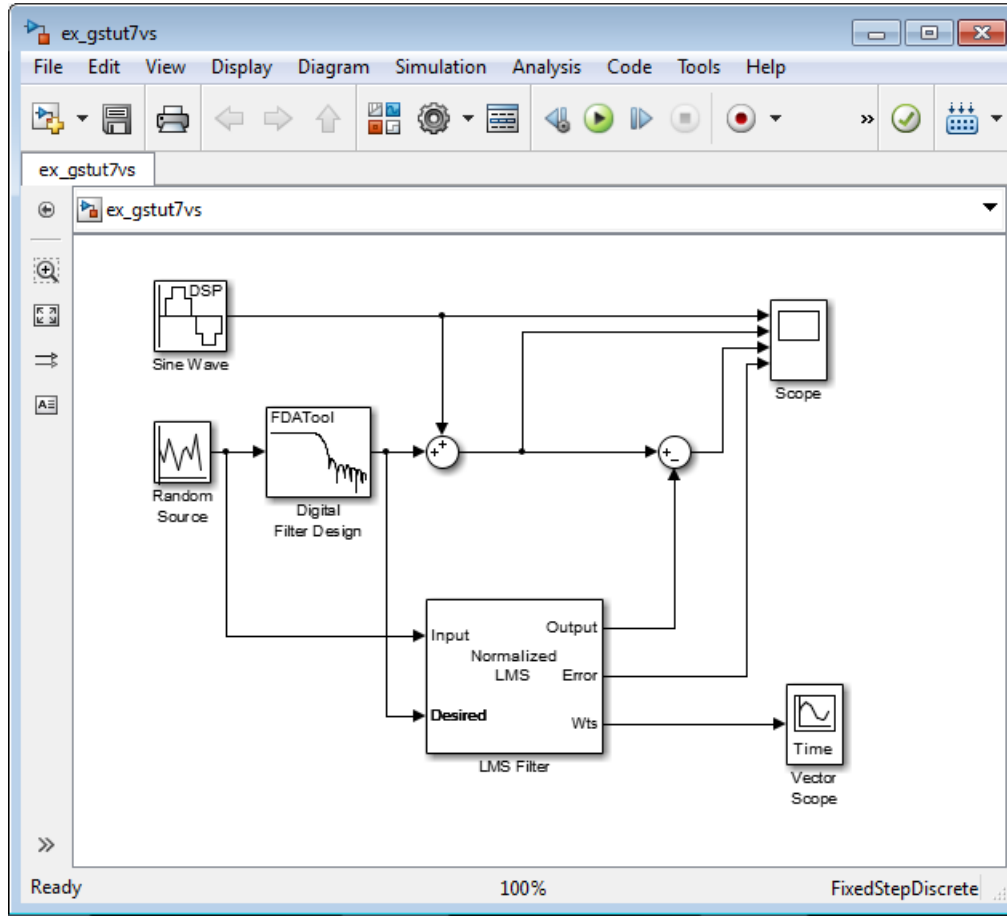
- **Minimum Y-limit** = `-0.2`
- **Maximum Y-limit** = `0.6`
- **Y-axis label** = `Filter Weights`

d Click the **Line Properties** tab.

- **Line visibilities** = `on`
- **Line style** = **:**
- **Line markers** = **.**
- **Line colors** = `[0 0 1]`

e Click **OK**.

5 Connect the Wts port of the LMS Filter block to the Vector Scope block.



6 Set the configuration parameters:

a Open the Configuration Parameters dialog box by selecting **Model Configuration Parameters** from the **Simulation** menu, and navigate to the **Solver** pane.

b Enter `inf` for the **Stop time** parameter.

c Choose `Fixed-step` from the **Type** list.

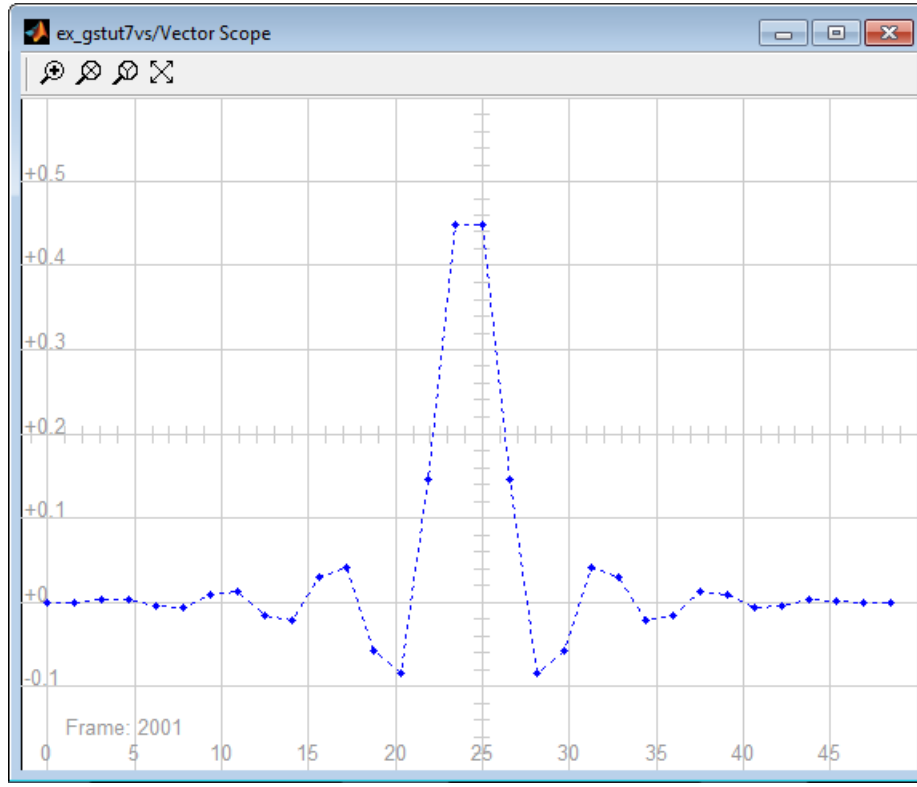
d Choose `Discrete (no continuous states)` from the **Solver** list.

We recommend these configuration parameters for models that contain DSP System Toolbox blocks. Because these blocks calculate values directly rather than solving differential equations, you must configure the Simulink Solver to behave like a scheduler. The Solver, while in scheduler mode, uses a block's sample time to determine when the code behind each block is executed. For example, the sample time of the Sine Wave and Random Source blocks in this model is `0.05`. The Solver executes the code behind these blocks, and every other block with this sample time, once every 0.05 second.

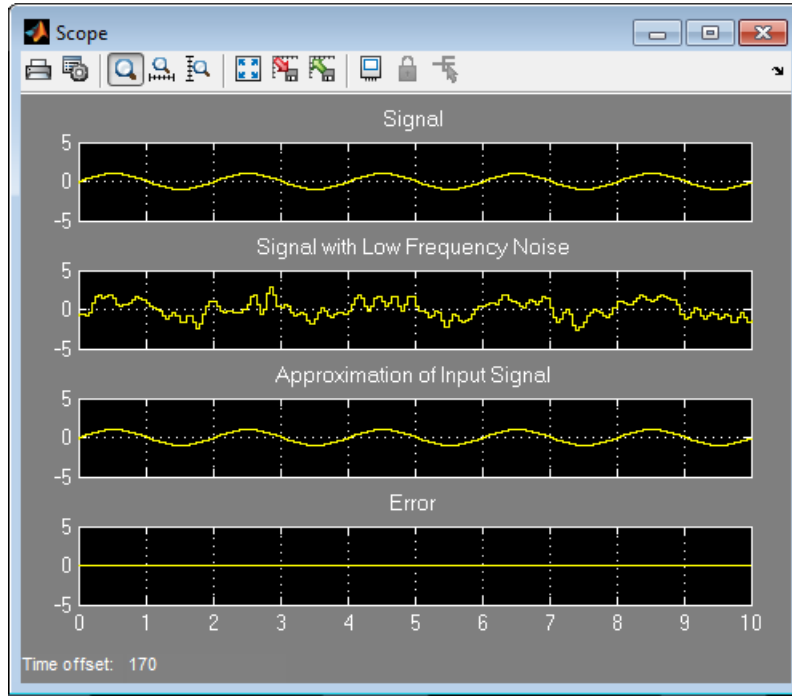
Note: When working with models that contain DSP System Toolbox blocks, use source blocks that enable you to specify their sample time. If your source block does not have a **Sample time** parameter, you must add a Zero-Order Hold block in your model and use it to specify the sample time. For more information, see "Continuous-Time Source Blocks" in the *DSP System Toolbox User's Guide*. The exception to this rule is the Constant block, which can have a constant sample time. When it does, Simulink executes this block and records the constant value once at the start of the simulation and any time you tune a parameter. This allows for faster simulations and more compact generated code.

---

7. Close the dialog box by clicking **OK**.
8. Open the Scope window by double-clicking the Scope block.
9. Run your model and view the behavior of your filter coefficients in the Vector Scope window, which opens automatically when your simulation starts. Over time, you see the filter coefficients change and approach their steady-state values, shown below.



You can simultaneously view the behavior of the system in the Scope window. Over time, you see the error decrease and the approximation of the input signal more closely match the original sinusoidal input signal.



You have now created a model capable of adaptive noise cancellation. So far, you have learned how to design a lowpass filter using the Digital Filter Design block. You also learned how to create an adaptive filter using the LMS Filter block. The DSP System Toolbox product has other blocks capable of designing and implementing digital and adaptive filters. For more information on the filtering capabilities of this product, see "Filter Design" and "Filter Analysis".

Because all blocks in this model have the same sample time, this model is single rate and Simulink ran it in `SingleTasking` solver mode. If the blocks in your model have different sample times, your model is multirate and Simulink might run it in `MultiTasking` solver mode. For more information on solver modes, see "Recommended Settings for Discrete-Time Simulations" in the *DSP System Toolbox User's Guide*.

To learn how to generate code from your model using the Simulink Coder product, see the "C Code Generation from Simulink" section.