

RF Blockset™

User's Guide



MATLAB® & SIMULINK®

R2023a



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.
1 Apple Hill Drive
Natick, MA 01760-2098

RF Blockset™ User's Guide

© COPYRIGHT 2010–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

September 2010	Online only	New for Version 3.0 (Release 2010b)
April 2011	Online only	Revised for Version 3.0.2 (Release 2011a)
September 2011	Online only	Revised for Version 3.1 (Release 2011b)
March 2012	Online only	Revised for Version 3.2 (Release 2012a)
September 2012	Online only	Revised for Version 3.3 (Release 2012b)
March 2013	Online only	Revised for Version 4.0 (Release 2013a)
September 2013	Online only	Revised for Version 4.1 (Release 2013b)
March 2014	Online only	Revised for Version 4.2 (Release 2014a)
October 2014	Online only	Revised for Version 4.3 (Release 2014b)
March 2015	Online only	Revised for Version 4.4 (Release 2015a)
September 2015	Online only	Revised for Version 4.5 (Release 2015b)
March 2016	Online only	Revised for Version 5.0 (Release 2016a)
September 2016	Online only	Revised for Version 5.1 (Release 2016b)
March 2017	Online only	Revised for Version 6.0 (Release 2017a)
September 2017	Online only	Revised for Version 6.1 (Release 2017b)
March 2018	Online only	Revised for Version 7.0 (Release 2018a)
September 2018	Online only	Revised for Version 7.1 (Release 2018b)
March 2019	Online only	Revised for Version 7.2 (Release 2019a)
September 2019	Online only	Revised for Version 7.3 (Release 2019b)
March 2020	Online only	Revised for Version 7.4 (Release 2020a)
September 2020	Online only	Revised for Version 8.0 (Release 2020b)
March 2021	Online only	Revised for Version 8.1 (Release 2021a)
September 2021	Online only	Revised for Version 8.2 (Release 2021b)
March 2022	Online only	Revised for Version 8.3 (Release 2022a)
September 2022	Online only	Revised for Version 8.4 (Release 2022b)
March 2023	Online only	Revised for Version 8.5 (Release 2023a)

Circuit Envelope

		Sensitivity
1	Model System Noise Figure	1-2
	Design Receiver with ADC	1-8

		Intermodulation Distortion
2	Model Direct Conversion Receiver	2-2
	Noise in RF Systems	2-7
	White and Colored Noise	2-7
	Thermal Noise	2-7
	Phase Noise	2-8
	Noise Figure	2-8

		Testbenches
3	Use RF Measurement Testbench for RF-to-IQ Converter	3-2
	Device Under Test	3-3
	RF Measurement Unit	3-3
	RF Measurement Unit Parameters	3-5
	Using RF Measurement Testbench for IQ-to-RF Converter	3-10
	Device Under Test	3-11
	RF Measurement Unit	3-11
	RF Measurement Unit Parameters	3-13

RF Blockset Models

Analog Devices Transceiver Models

4

AD9361 Models	4-2
AD9361_TX Analog Devices Transmitter	4-3
AD9361_RX Analog Devices Receiver	4-4
AD9361 Testbenches	4-6
AD9361_TX Analog Devices Transmitter Testbench	4-7
AD9361_RX Analog Devices Receiver Testbench	4-7
AD9361_QPSK Analog Devices Testbench	4-8
AD9371 Models	4-9
AD9371_TX Analog Devices Transmitter	4-10
AD9371_RX Analog Devices Receiver	4-11
AD9371_ORX Analog Devices Observer Receiver	4-12
AD9371_SNF Analog Devices Sniffer Receiver	4-13
AD9371 Testbenches	4-15
AD9371_TX Analog Devices Transmitter Testbench	4-16
AD9371_RX Analog Devices Receiver Testbench	4-17
AD9371_ORX Analog Devices Observer Receiver Testbench	4-18
AD9371_SNF Analog Devices Sniffer Receiver Testbench	4-18
AD9371_TX_ORX Analog Devices Transmitter-Observer Testbench	4-19

Equivalent Baseband

Model an RF System

5

Model RF Components	5-2
Add RF Blocks to a Model	5-2
Connect Model Blocks	5-3
Specify or Import Component Data	5-5
Specify Operating Conditions	5-13
Model Nonlinearity	5-14
Amplifier and Mixer Nonlinearity Specifications	5-14
Add Nonlinearity to Your System	5-14

Model Noise	5-16
Amplifier and Mixer Noise Specifications	5-16
Add Noise to Your System	5-17
Plot Noise	5-20

Plot Model Data

6

Create Plots	6-2
Available Data for Plotting	6-2
Validate Individual Blocks and Subsystems	6-2
Types of Plots	6-3
Plot Formats	6-3
How to Create a Plot	6-10
Example — Plot Component Data on a Z Smith Chart	6-16
Update Plots	6-20
Modify Plots	6-21
Create and Modify Subsystem Plots	6-23

RF Blockset Equivalent Baseband Algorithms

A

Simulate an RF Model	A-2
Determine Modeling Frequencies	A-3
Map Network Parameters to Modeling Frequencies	A-4
Model Noise in an RF System	A-5
Output-Referred Noise in RF Models	A-5
Calculate Noise Figure at Modeling Frequencies	A-6
Calculate System Noise Figure	A-7
Calculate Output Noise Power	A-8
Create Complex Baseband-Equivalent Model	A-9
Baseband-Equivalent Modeling	A-9
Simulation Efficiency of a Baseband-Equivalent Model	A-12
Example — Select Parameter Values for a Baseband-Equivalent Model	A-12
Convert to and from Simulink Signals	A-22
Signal Conversion Specifications	A-22
Interpret Simulink Signals as Incident Power Waves	A-22
Interpret Simulink Signals as Source Voltages	A-24
Specify Input Signal Conversions	A-24

B

2-Port Mixer Blocks	B-2
Model a Mixer Chain	B-4
Quadrature Mixers	B-6
Use RF Blockset Equivalent Baseband Software to Model Quadrature Mixers	B-6
Model Upconversion I/Q Mixers	B-6
Model Downconversion I/Q Mixers	B-7
Simulate I/Q Mixers	B-7

Examples

7

Vary Phase Of Signal During Simulation	7-2
Vary Attenuation of Signal During Simulation	7-4
Explicitly Simulate Resistor Thermal Noise	7-5
Attenuate Signal Power	7-6
Demodulate Two-Tone RF Signal Using IQ Demodulator	7-7
Modulate Two-Tone DC Signal Using IQ Modulator	7-12
Spot Noise Data in Amplifiers and Effects on Measured Noise Figure ..	7-16
Measure Transducer Gain of Device Under Test	7-20
Measure Noise Figure of Device Under Test	7-22
Measure IIP2 of Device Under Test	7-25
Measure IIP3 of Device Under Test	7-27
Measure OIP2 of Device Under Test	7-30
Measure OIP3 of Device Under Test	7-32
Single Pole Triple Throw Switch	7-35
Frequency Response of Lowpass Chebyshev Filter	7-38
Model LO Phase Noise	7-42
Carrier to Interference Performance of Weaver Receiver	7-48

Modulate Two-Tone DC Signal Using IQ Modulator	7-55
Measurement of Gain and Noise Figure Spectrum	7-59
Idealized Baseband Amplifier with Nonlinearity and Noise	7-72
Use Ladder Filter Block to Filter Gaussian Noise	7-74
Measure S-Parameter Data of Chebyshev Filter	7-77
Measure S-Parameter of Nonlinear System	7-81
Simulation of RF Systems with Antenna Blocks	7-87
Power Amplifier Characterization	7-92
Modulate Quadrature Baseband Signals Using IQ Modulators	7-104
Intermodulation Analysis of Mathematical Amplifier	7-107
Create Virtual Connections Using Connection Label Block	7-109
Model Wilkinson Power Divider	7-110
Modulate Input Signal Onto Square Carrier Wave	7-115
Time-Domain Filtering of RF Complex Baseband Signals in Simulink	7-122
Model RF Complex Baseband S-Parameters in Simulink	7-125

RF Blockset Examples

8

Getting Started with RF Modeling	8-2
Passband Signal Representation in Circuit Envelope	8-6
Power Ports and Signal Power Measurement in RF Blockset	8-11
Communications System with Embedded RF Receiver	8-13
Automatic Sample-Time Interpolation at Input Port	8-17
Analysis of Frequency Response of RF System	8-32
Compare Time and Frequency Domain Simulation Options for S-parameters	8-40
Transmission Lines, Delay-Based and Lumped Models	8-47
Validating IP2/IP3 Using Complex Signals	8-57

Two-Tone Envelope Analysis Using Real Signals	8-61
Measuring Image Rejection Ratio in Receivers	8-66
Executable Specification of a Direct Conversion Receiver	8-70
Frequency Response of RF Transmit/Receive Duplex Filter	8-75
Digital Predistortion to Compensate for Power Amplifier Nonlinearities	8-78
Radar System Modeling	8-87
RF Receiver Modeling for LTE Reception	8-92
Create Custom RF Blockset Models	8-99
Multiple Realizations of Cascaded Filters	8-104
Cascaded RF Systems	8-106
Power in Simulink Sources and Signals	8-108
Effect of Nonlinear Amplifier on 16-QAM Modulation	8-120
Executable Specification for System Design	8-123
Radar Tracking System	8-132
User-Defined Nonlinear Amplifier Model	8-138
Modeling and Simulation of MIMO RF Receiver Including Beamforming	8-143
Modeling RF mmWave Transmitter with Hybrid Beamforming	8-149
Wireless Digital Video Broadcasting with RF Beamforming	8-156
Top-Down Design of an RF Receiver	8-166
Architectural Design of a Low IF Receiver System	8-178
RF Noise Modeling	8-184
Impact of Thermal Noise on Communication System Performance ...	8-187
100 Watt TR Module for S-Band Applications	8-191
Massive MIMO Hybrid Beamforming with RF Impairments	8-196
Speed Up PA and DPD simulation	8-210
Model RF Systems with Antenna Arrays Using RF Blockset Antenna Block	8-219

PA and DPD Modeling for Dynamic EVM Measurement	8-232
RF Impairments for 5G NR Downlink Waveforms	8-262
Design and Simulate Monopulse Tracking System	8-273
Enable Model Protection and Accelerator Modes in RF Blockset Models	8-282
Design RF Direct-Conversion Receiver	8-284
Protect Circuit Envelope Model	8-288
Implement Automatic Gain Control for RF Receiver	8-295

Cross-Product Workflow Topics

9

RF System Design for Radar and Wireless Communications	9-2
Design Considerations	9-2
Design Workflows	9-2
RF Transceiver Design	9-4
Design Considerations	9-4
Design Workflows	9-4
RF Noise and Nonlinearity Simulations	9-6
PA Characterizations and Spot Noise Measurements	9-6
Idealized Baseband Simulations	9-6
Simulation Workflows	9-6

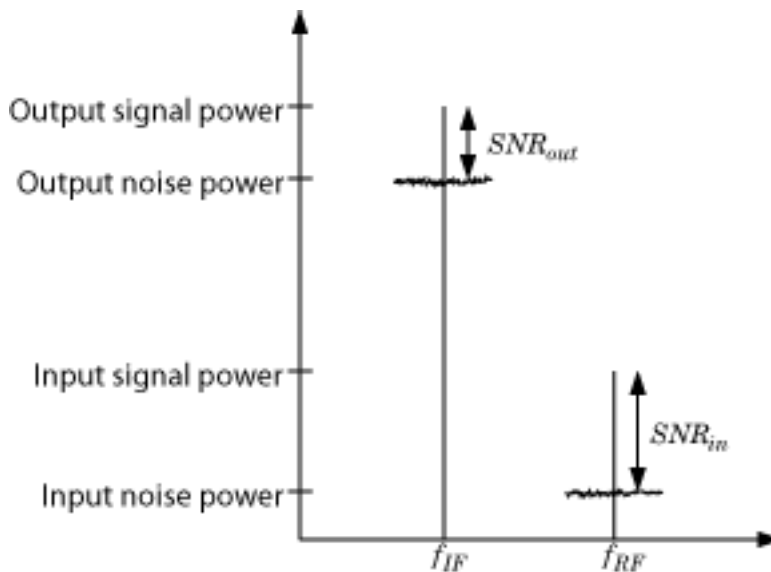
Circuit Envelope

Sensitivity

- “Model System Noise Figure” on page 1-2
- “Design Receiver with ADC” on page 1-8

Model System Noise Figure

This example shows how to model system noise figure in RF Blockset™. RF receivers amplify signals and shift them to lower frequencies. The receiver itself introduces noise that degrades the received signal. The signal-to-noise ratio (SNR) at the receiver output ultimately determines the usability of the receiver.



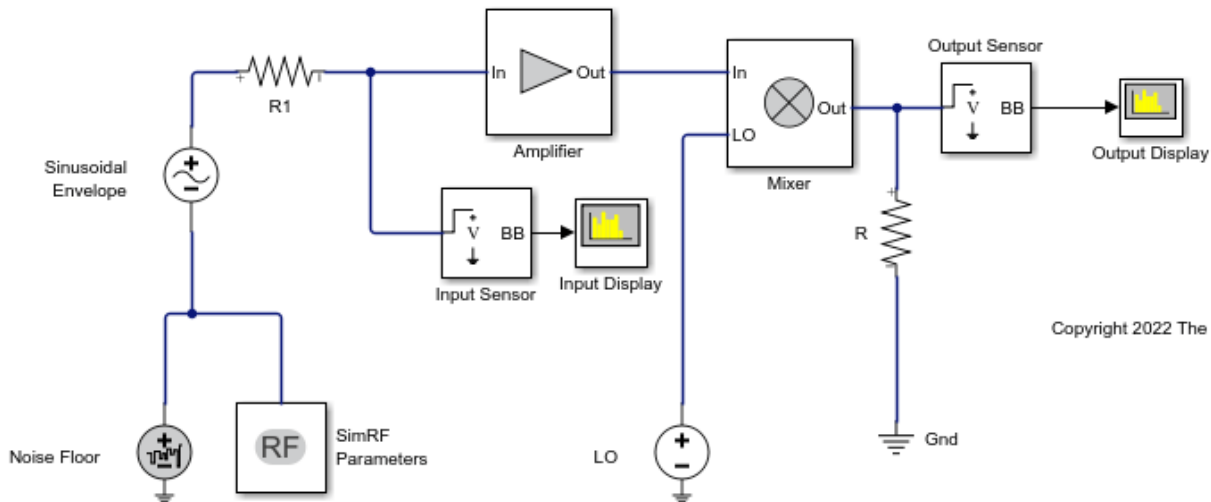
The preceding figure illustrates the effect of the receiver on the signal. The receiver amplifies a low-power RF signal at the carrier f_{RF} with a high SNR and downconverts the signal to f_{IF} . The noise figure (NF) of the system determines the difference between the SNR at the output and the SNR at the input:

$$\text{SNR}(\text{out}) = \text{SNR}(\text{in}) - \text{NF}(\text{sys})$$

Where the difference is calculated in decibels. Excessive noise figure in the system causes the noise to overwhelm the signal, making the signal unrecoverable.

Create Low-IF Receiver Model

The model `ex_simrf_snr_model` simulates a simplified IF receiver architecture. A Sinusoid block and a Noise block model a two-tone input centered at f_{RF} and low-level thermal noise. The RF system amplifies the signal and mixes it with the local oscillator f_{LO} down to an intermediate frequency f_{IF} . A voltage sensor recovers the signal at the IF.



Setup RF Blockset Environment

To maximize performance, the Fundamental tones and Harmonic order parameters specify the simulation frequencies explicitly in the Configuration block:

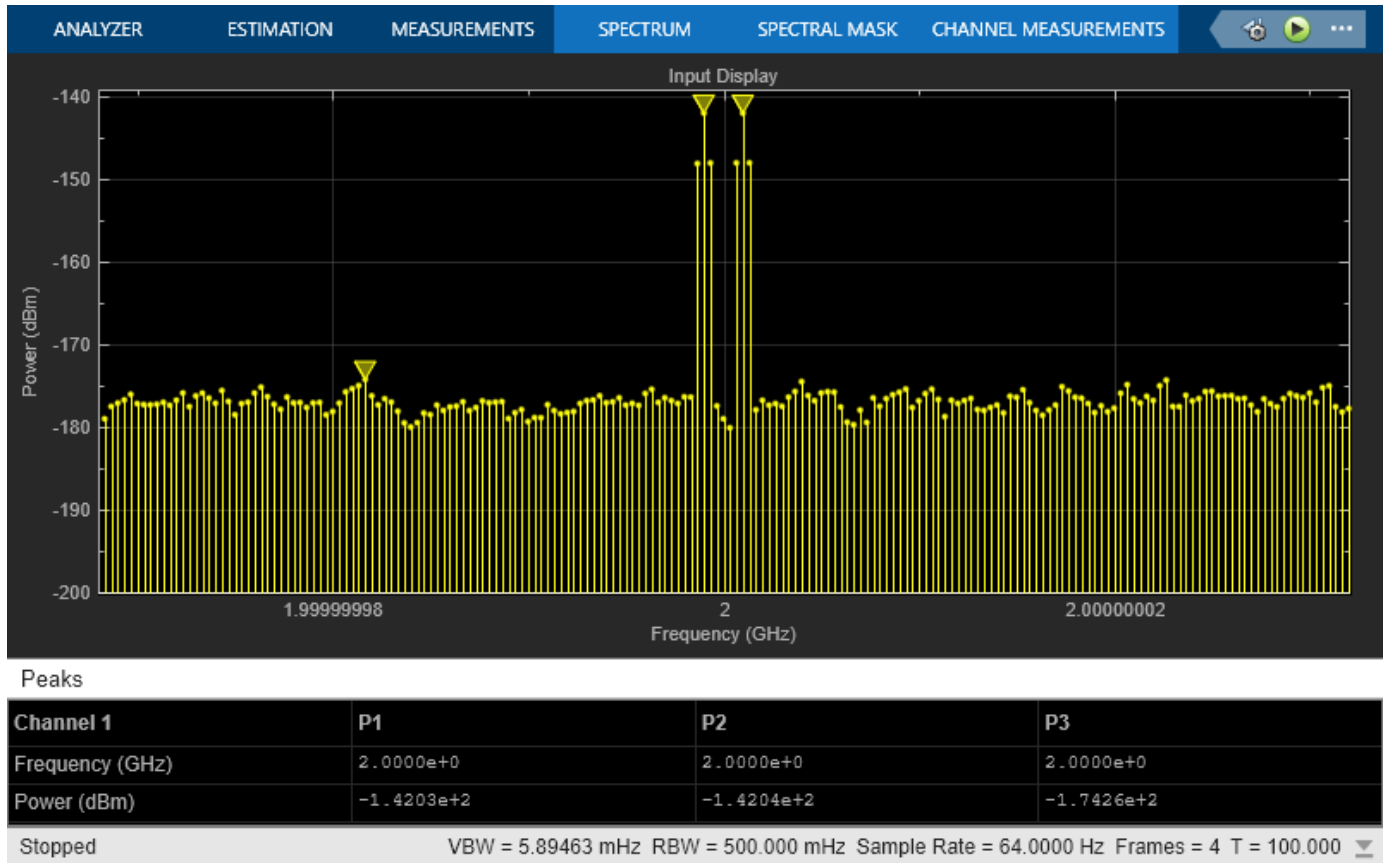
- `fLO`, the frequency of the LO in the first mixing stage, equals 1.9999 GHz. and appears in the list of fundamental tones as `carriers.LO`.
- `fRF`, the carrier of the desired signal, equals 2 GHz and appears in the list of fundamental tones as `carriers.RF`.
- `fIF`, the intermediate frequency, equals `fRF - fLO`. The frequency is a linear combination of the first-order (fundamental) harmonics of `fLO` and `fRF`. Setting **Harmonic order** to 1 is sufficient to ensure this frequency appears in the simulation frequencies. This minimal value for the harmonic order ensures a minimum of simulation frequencies.

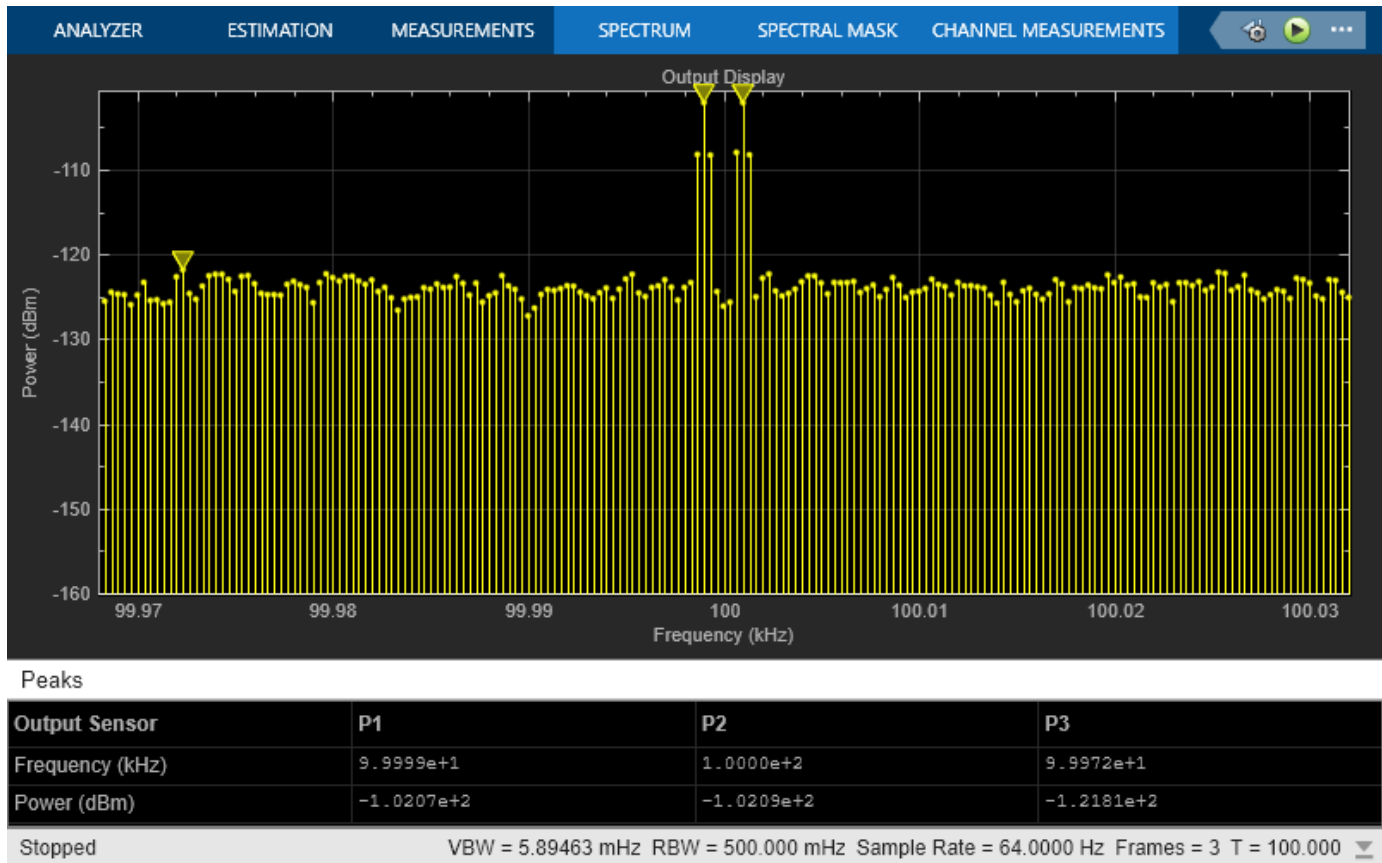
Solver conditions and noise settings are also specified for the Configuration block:

- The **Solver type** is set to `auto`. For more information on choosing solvers, see Configuration block.
- The **Sample time** parameter is set to $1/(\text{mod_freq} * 64)$. This setting ensures a simulation bandwidth 64 times greater than the envelope signals in the system.
- The **Simulate noise** box is checked, so the environment includes noise parameters during simulation.

The model uses subsystems with a MATLAB Coder™ implementation of a fast Fourier transform (FFT) to generate two plots. The FFT uses 64 bins, so for a sampling frequency of 64 Hz, the bandwidth of each bin is 1 Hz. Subsequently, the power levels shown in the figures also represent the power spectral density (PSD) of the signals in dBm/Hz.

View Simulation Output





The Input Display plot shows the power spectrum of the signal and noise at the input of the receiver. The measured power of each tone is consistent with the expected power level of a $0.1\text{ }\mu\text{V}$ two-tone envelope:

$$\begin{aligned}
 P_{in} &= 10\log_{10}\left(\frac{V^2}{2R}\right) + 30 \\
 &= 10\log_{10}\left(\frac{\left(\frac{1}{2} \cdot \frac{10^{-7}}{2}\right)^2}{2.50}\right) + 30 = -142\text{dBm}
 \end{aligned}$$

A factor of $1/2$ is due to voltage division across source and load resistors, and another factor of $1/2$ is due to envelope scaling. For more information on scaling envelope signals for power calculation, see “Two-Tone Envelope Analysis Using Real Signals” on page 8-61. The measured noise floor at -177 dBm/Hz is reduced by 3 dB from the specified -174 dBm/Hz noise floor. The difference is due to power transfer from the source to the input of the amplifier. The amplifier also models a thermal noise floor, so although this decrease is unrealistic, it does not affect accuracy at the output stage.

The Output Display plot shows the power spectrum of the signal and noise at the output of the receiver. The measured PSD of -102 dBm/Hz for each tone is consistent with the 40-dB combined gain of the amplifier and mixer. The noise PSD in the figure is shown to be approximately 50 dB higher at the output, due to the gain and noise figure of the system.

Simulate Thermal Noise Floor

Thermal noise power can be modeled according to the equation,

$$P_{noise} = 4k_B T R_S \Delta f$$

where:

k_B is Boltzmann's constant, equal to 1.38065×10^{-23} J/K.

T is the noise temperature, specified as 293.15 K in this example.

R_S is the noise source impedance, specified as 50 ohm in this example to agree with the resistance value of the Resistor block labeled R1.

Δf is the noise bandwidth.

To model the noise floor on the RF signal at the resistor, the model includes a Noise block:

The ***Noise Power Spectral Density (Watts/Hz)** parameter is calculated as $\frac{P_{noise}}{\Delta f} = 4k_B T R_S$. The ***Carrier frequencies** parameter, set to `carriers.RF`, places noise on the RF carrier only.

Compute System Noise Figure

To model RF noise from component noise figures:

- Select Simulate noise in the RF Blockset Parameters block dialog box, if it is not already selected.
- Specify a value for the Noise figure (dB) parameter in Amplifier and Mixer blocks.

The noise figures are not strictly additive. The amplifier contributes more noise to the system than the mixer because it appears first in the cascade. To calculate the total noise figure of the RF system with n stages, use the Friis equation:

$$F_{sys} = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \dots + \frac{F_n - 1}{G_1 G_2 \dots G_{n-1}}$$

where F_i and G_i are the noise factor and gain of the i th stage, and $NFi = 10 \log_{10}(Fi)$.

In this example, the noise figure of the amplifier is 10 dB, and the noise figure of the mixer is 15 dB, so the noise figure of the system is:

$$10 \log_{10} \left(10^{\frac{10}{10}} + \frac{10^{\frac{15}{10}} - 1}{10000} \right) = 10 \text{dB}$$

The Friis equation shows that although the mixer has a higher noise figure, the amplifier contributes more noise to the system.

See Also

Noise | Amplifier | Mixer

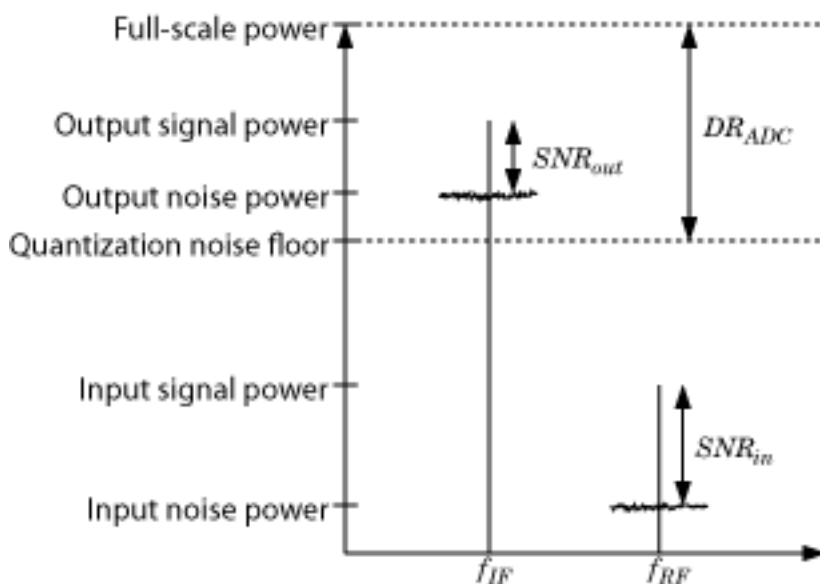
More About

- “Noise in RF Systems” on page 2-7

Design Receiver with ADC

Most RF receivers in modern communications or radar systems feed signals to an analog-to-digital converter (ADC). Due to their finite resolution, ADCs introduce quantization error into the system. The resolution of the ADC is determined by the number of bits and the full-scale (FS) range of the ADC.

The preceding figure illustrates an RF signal that falls within the dynamic range (DR) of an ADC. The input signal and noise at the carrier f_{RF} has high signal-to-noise ratio (SNR). The received signal at f_{IF} has reduced SNR due to system noise figure. However, if the quantization error is near or above the receiver noise, system performance degrades.



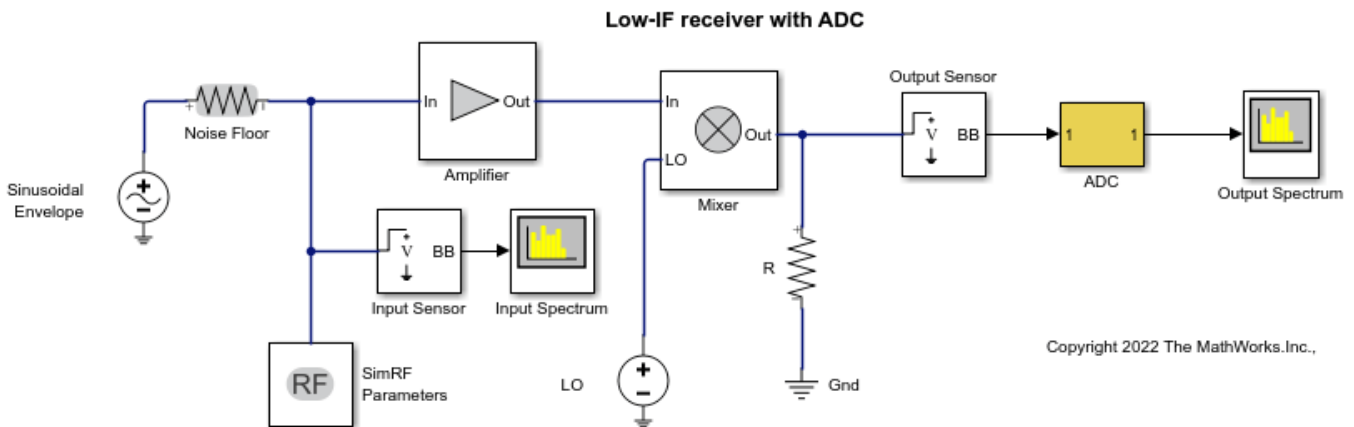
To ensure that the ADC contributes no more than 0.1 dB of noise to the signal at f_{IF} , the quantization noise floor must be 16 dB lower than the receiver noise. This condition can be met by:

- Reducing the full-scale (FS) range or increasing the resolution of the ADC, which lowers the quantization noise floor.
- Increasing the gain of the RF receiver, which raises the receiver noise floor.

Overcome Quantization Error of an ADC

The model `ex_simrf_adc_rx` simulates a low IF receiver with an ADC. This model is based on low IF receiver design from the “Model System Noise Figure” on page 1-2. At the output of the RF system, the ADC subsystem models an ADC with an FS range of $\sqrt{100e-3}$ V and a resolution of 16 bits.

```
open_system("ex_simrf_adc_rx")
```



The power of a voltage signal at the full-scale range of the ADC in dBm is

$$P_{vs} = 10 \cdot \log_{10}(\sqrt{100^{-3}}) + 30 = 0$$

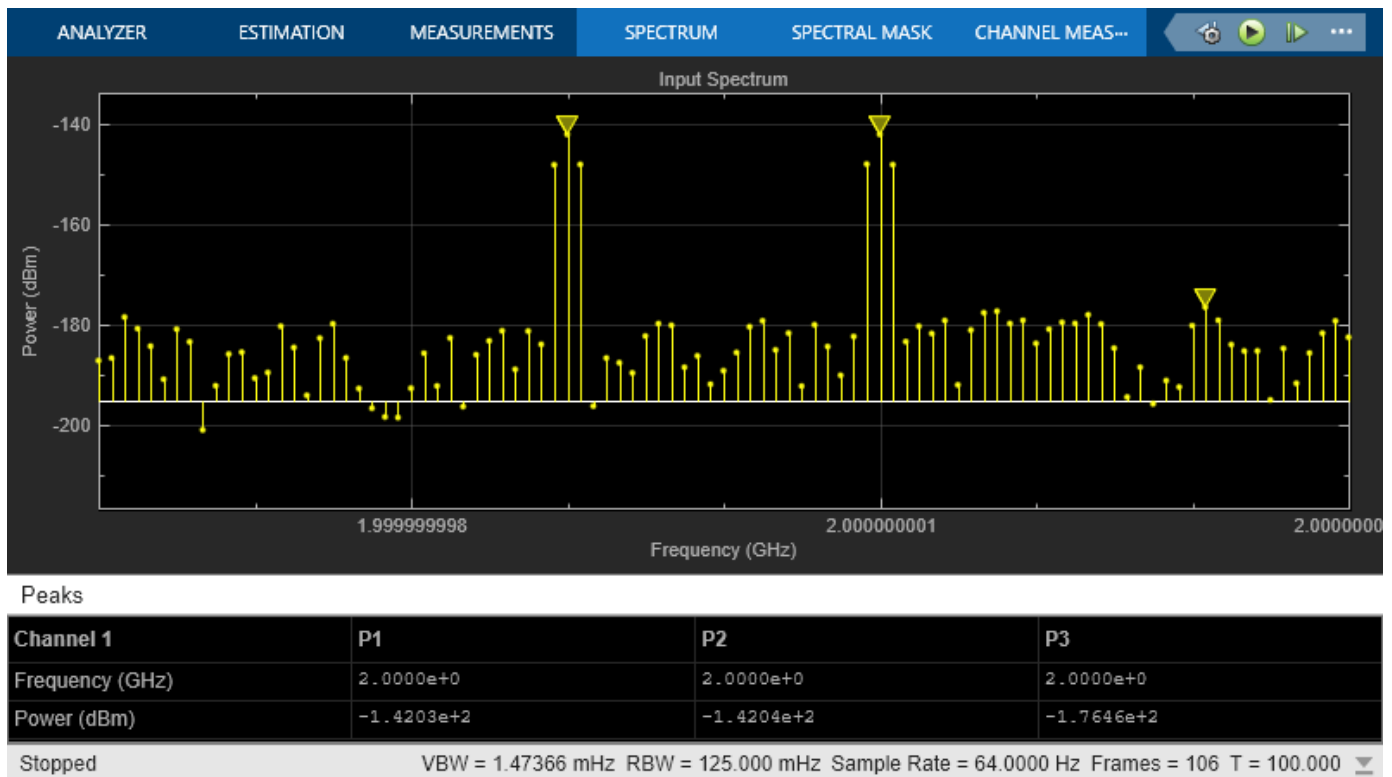
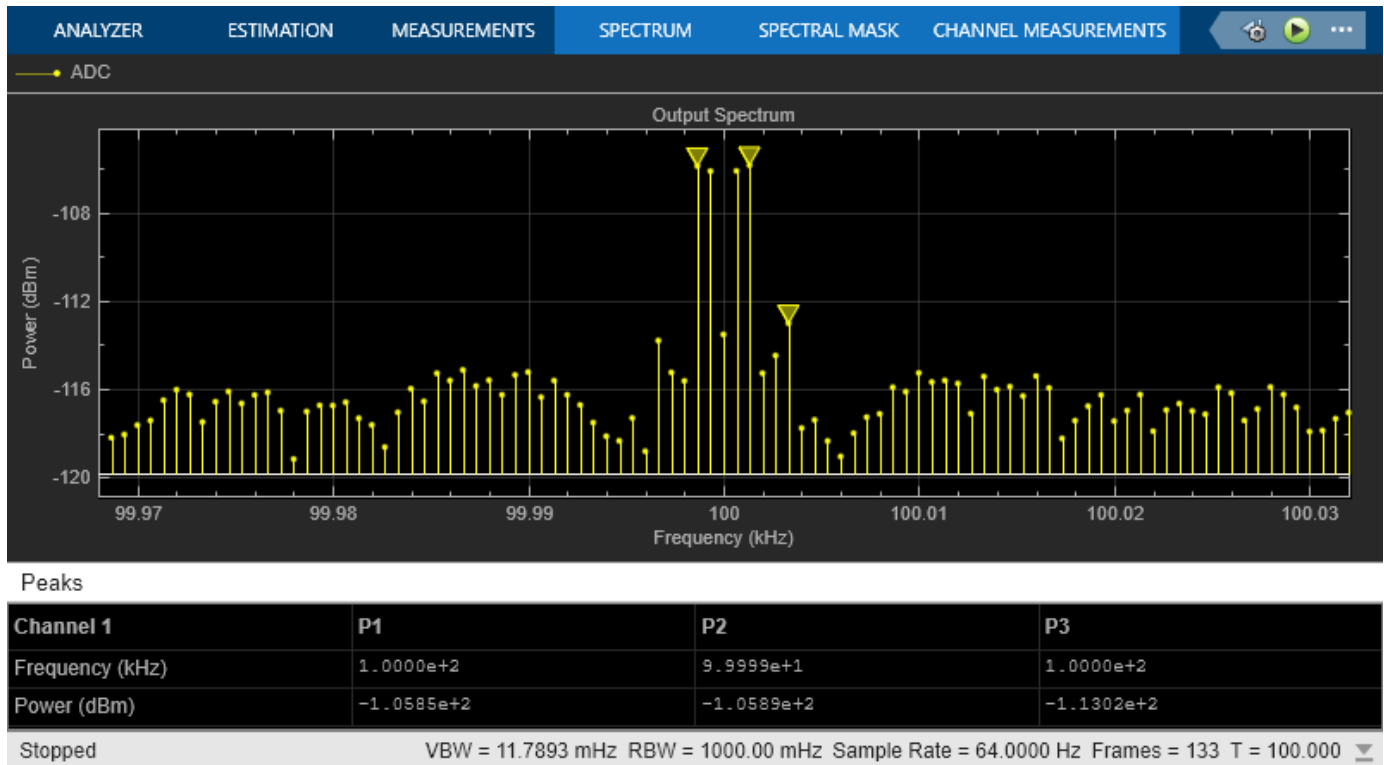
View Simulation Output

The power levels shown in the figures represent the power spectral density (PSD) of the signals in dBm/Hz.

The Spectrum Analyzer Block shows the power spectrum of the two-tone signal and noise at the input of the receiver-ADC system. The measured power of each tone of -142 dBm is consistent with the expected power level of a 0.1 uV signal. The power level of the noise is consistent with a -174 dBm/Hz noise floor. The output spectrum plot shows power spectrum of the output signal.

```
sim("ex_simrf_adc_rx")
```

1 Sensitivity



The quantization error exceeds the receiver noise.

Measure Quantization Noise Floor

To calculate the quantization noise floor (QNF_ADC) of the ADC, subtract the dynamic range from the full-scale power, which is 0 dBm. QNF_ADC is calculated using this equation:

$$\text{QNF_ADC} = 6.02 \cdot N_{\text{bits}} + 10 \cdot \log_{10}(\text{delta}f) + 1.76 = 116.1 \text{ dBm/Hz}$$

where

- N_{bits} is the resolution. The ADC in this example uses 16 bits.
- $\text{delta}f$ is the bandwidth of the FFT, which is 64 in this example. Oversampling in an ADC yields lower quantization noise.
- The value 1.76 is a correction factor for a pure sinusoidal input.

Therefore, the quantization noise floor is -116 dBm/Hz, in agreement with the measured output levels.

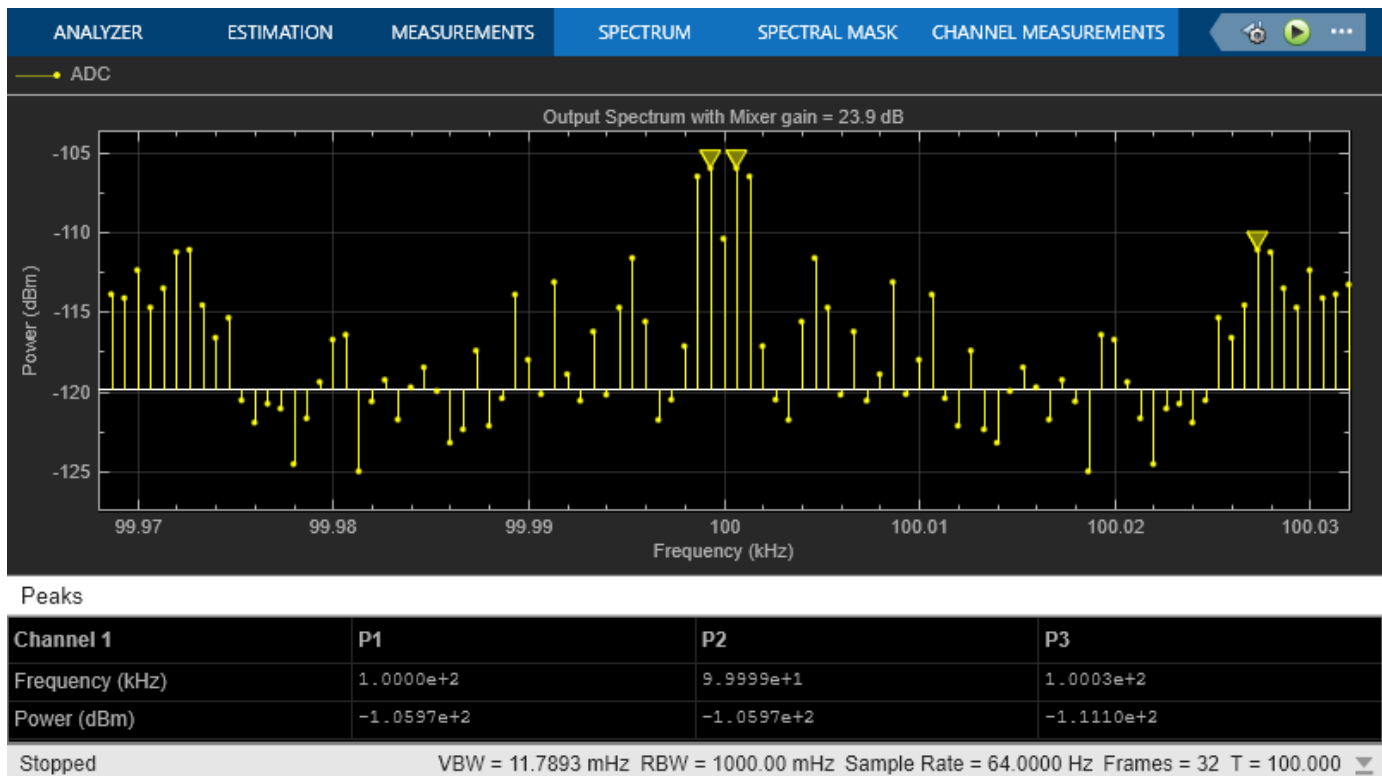
Improving Receiver-ADC Performance

Increasing the gain in the mixer raises the receiver noise without increasing the noise figure. Calculate the mixer gain required to achieve a 16-dB margin between the quantization noise floor and the receiver noise:

$$G_{\text{mixer}} = (\text{QNF_ADC} + 16) - (-174 + G_{\text{sys}} + \text{NF}_{\text{sys}}) = (-116.1 + 16) - (-174 + 40 + 10) = 23.9 \text{ dB}$$

To simulate a receiver that clears the quantization noise floor: set the Available power gain parameter of the mixer to 23.9 and click **Run**.

```
sim("ex_simrf_adc_rx_mixer")
```



The figure shows that the receiver noise is 16 dB above the quantization noise floor.

See Also

Mixer | Amplifier

More About

- “RF Noise Modeling” on page 8-184
- “Noise in RF Systems” on page 2-7

Intermodulation Distortion

- “Model Direct Conversion Receiver” on page 2-2
- “Noise in RF Systems” on page 2-7

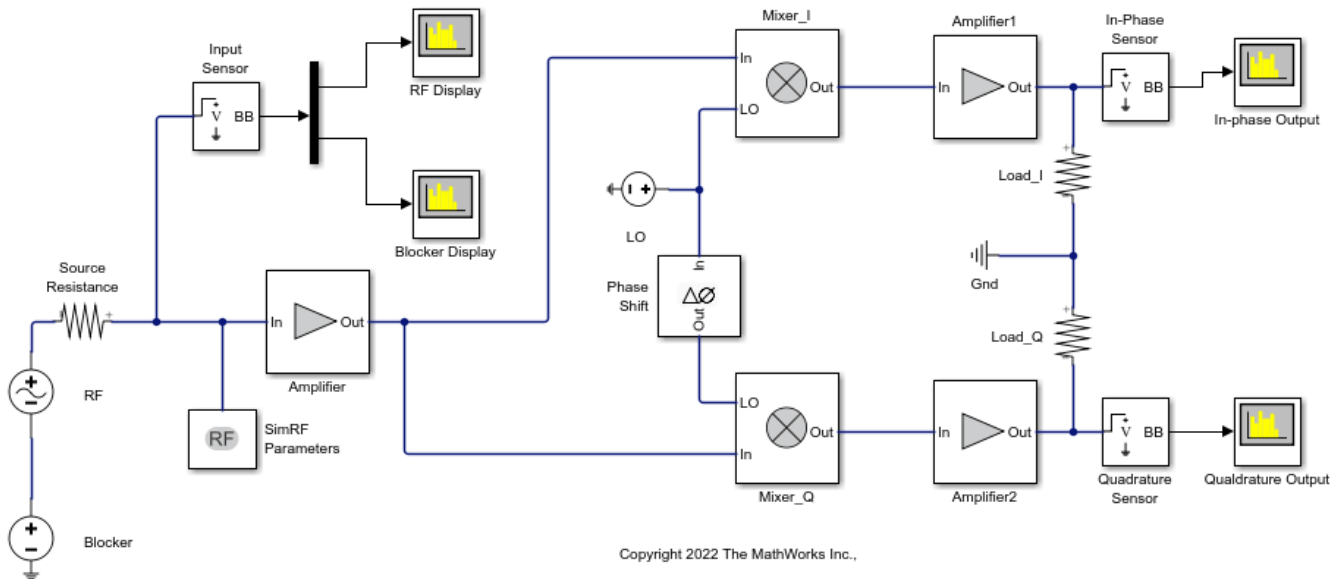
Model Direct Conversion Receiver

This example shows you how to model a direct conversion receiver. Direct conversion receivers are sensitive to second-order intermodulation products because they transfer the RF signal directly to baseband. The RF system consists of a low-noise amplification (LNA) stage, a direct-conversion stage, and a final amplification stage.

System Specifications

Open the model to inspect the direct conversion receiver.

```
open_system("ex_simrf_dc_model")
%
```



The model runs according to the following environment settings:

In the Configuration block, the **Fundamental tones** parameter specifies the carriers in the RF Blockset™ environment:

- $f_{RF} = f_{LO}$, the carrier of the RF and the local oscillator.
- f_{BL} , the blocker carrier

The RF Blockset environment always simulates the 0 Hz carrier, regardless of whether the RF Blockset parameters block specifies it.

In the Solver Configuration dialog box, the Use local solver box is selected. This setting causes the RF Blockset environment to simulate with a local solver with the following settings:

- **Solver type** is Trapezoidal rule .
- **Sample time** is `sample_time`, defined as $1.25e-4$ in the model initialization function.

Since the model uses a local solver, the global solver settings do not affect the simulation within the RF Blockset environment. For more information on global and local solvers, see “Choosing Simulink and Simscape Solvers”

To maximize performance, set **Fundamental tones** and **Harmonic order** parameters to specify the simulation frequencies explicitly in the Configuration block:

- $f_{RF} = f_{LO}$, the carrier of the RF and the local oscillator, appears as a fundamental tone.
- f_{BL} , the blocker carrier, appears as a fundamental tone.

A carrier of 0 Hz, representing the passband signal, is included in the set of first-order harmonics of both fundamental tones. Therefore, setting **Harmonic order** to 1 is sufficient to ensure that this frequency appears in the simulation frequencies. This minimal value for the harmonic order ensures a minimum of simulation frequencies.

Solver conditions and noise settings are also specified for the Configuration block: The **Solver type** is set to `auto`. For more information on choosing solvers, see Configuration or see “Choosing Simulink and Simscape Solvers”.

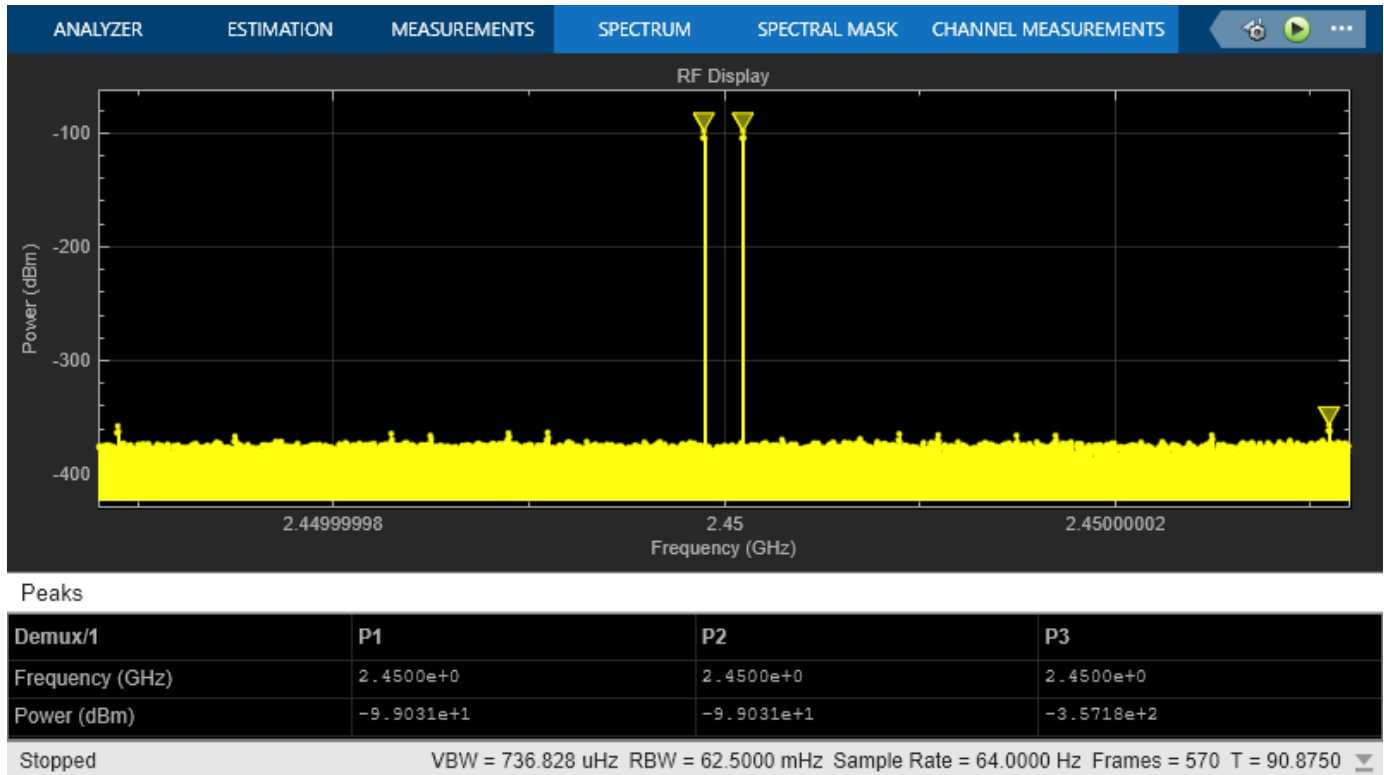
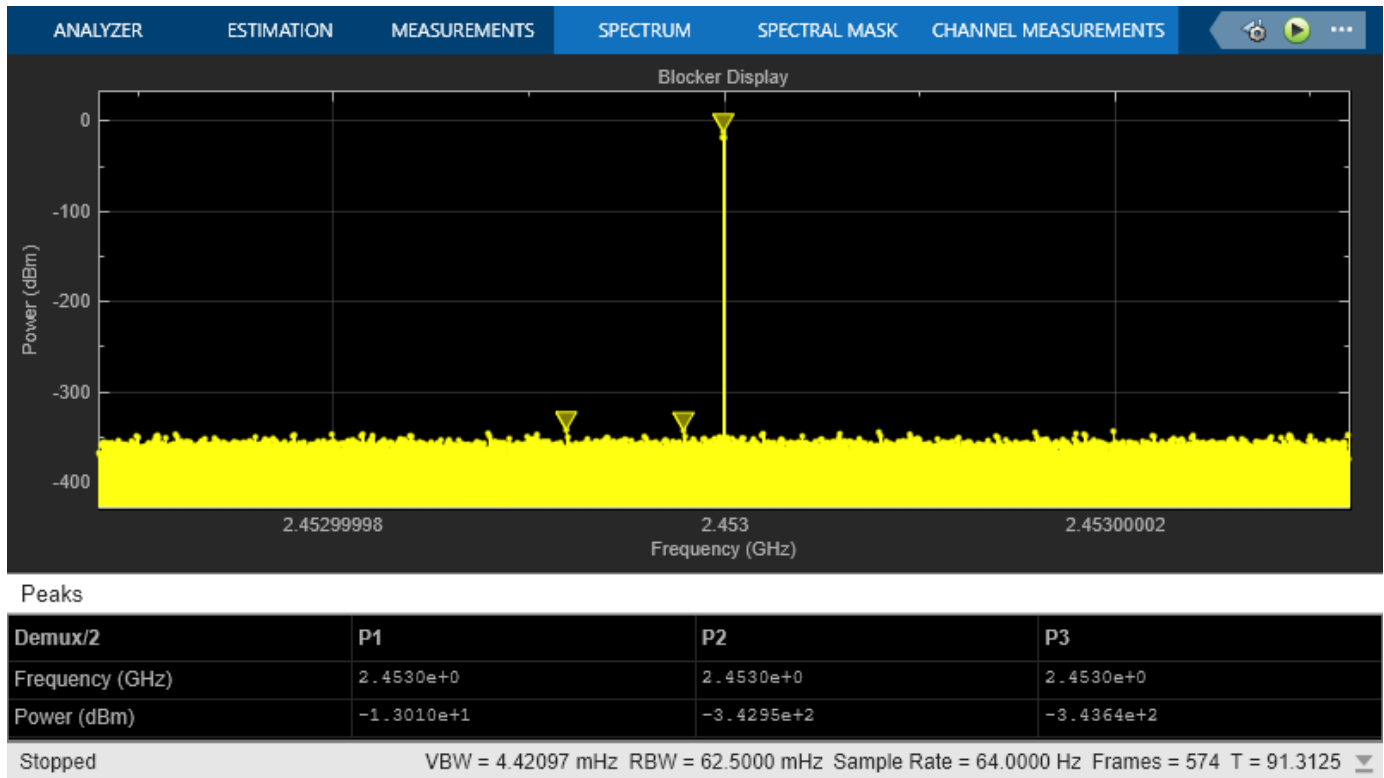
The **Sample time parameter** is set to `sample_time`, which is equal to $1/(mod_freq*64)$. This setting ensures a simulation bandwidth 64 times greater than the envelope signals in the system.

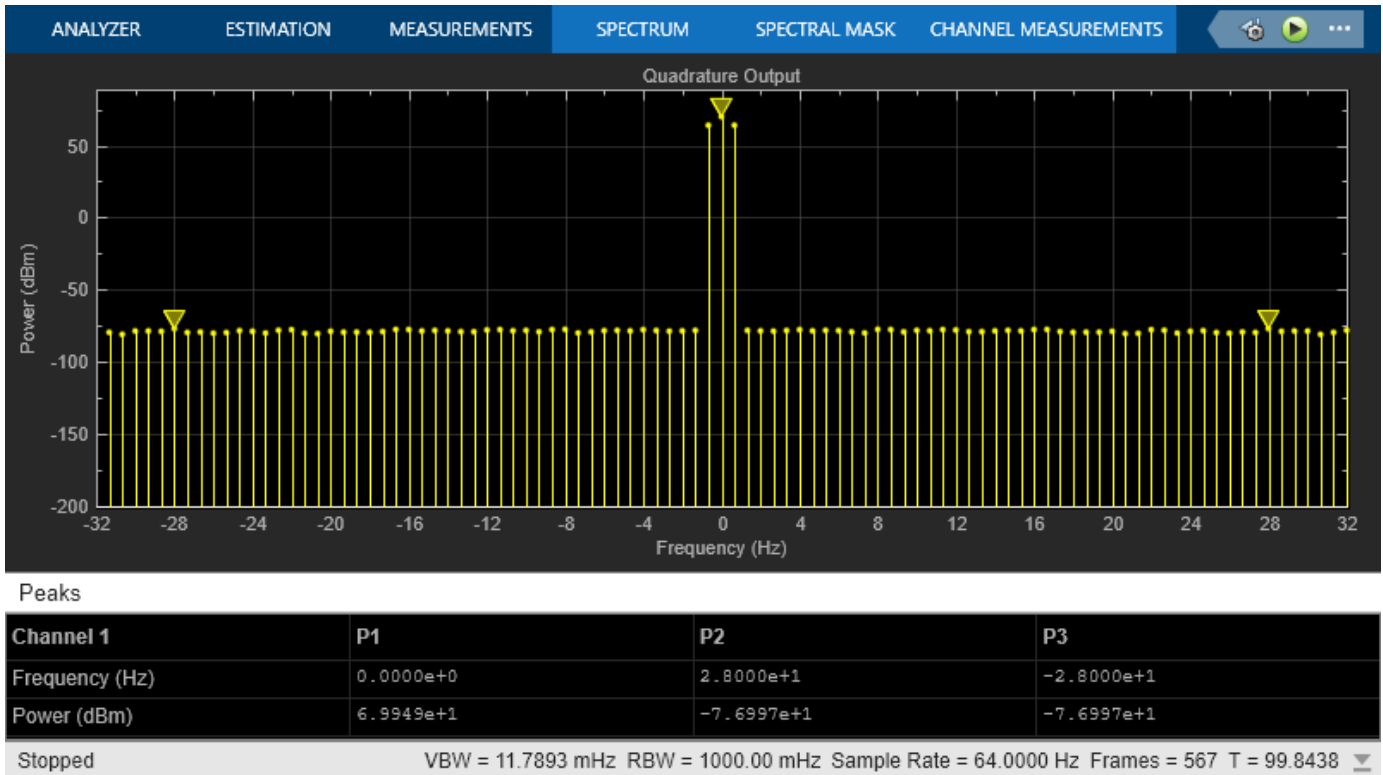
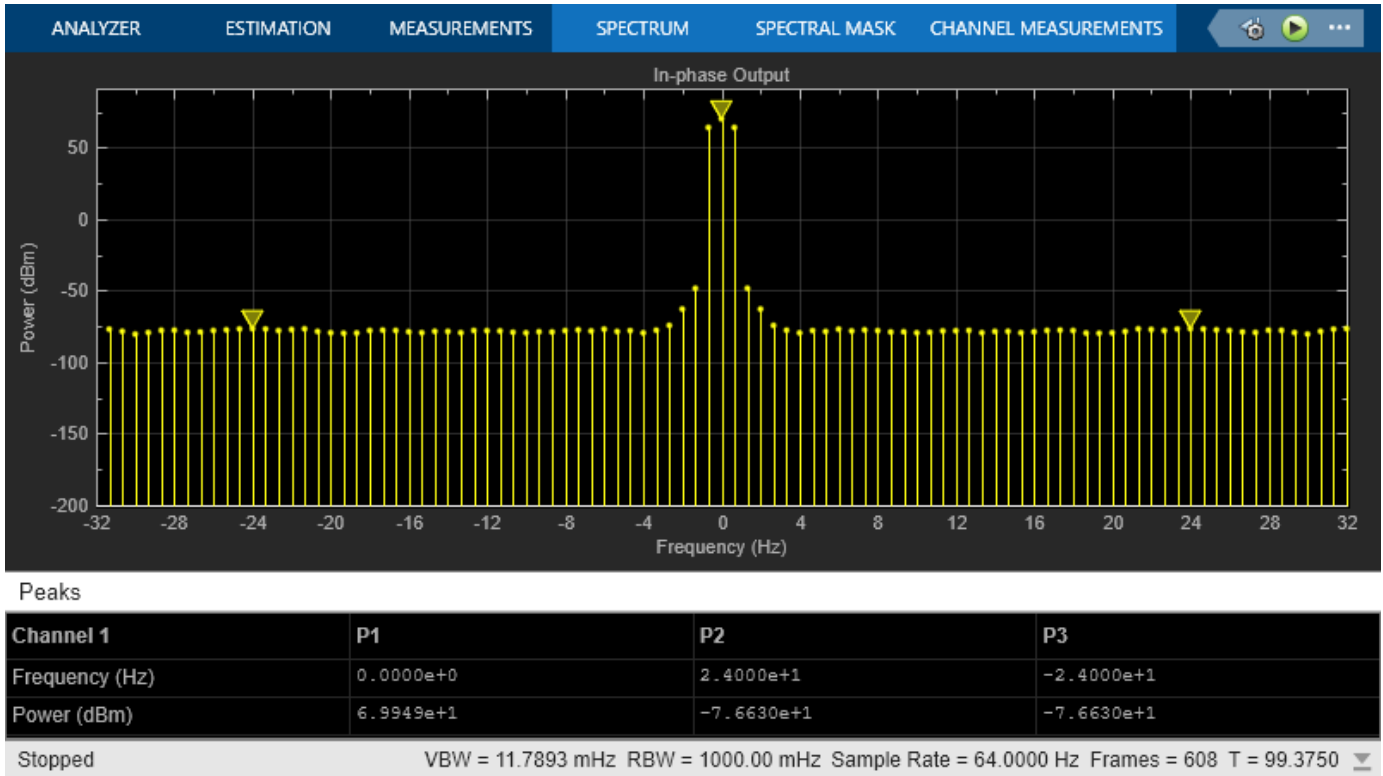
The **Simulate noise** check box is checked to include noise parameters during simulation.

View Simulation Output

Four Spectrum Analyzer blocks from Circuit Envelope Utilities sublibrary are used to visualize the outputs.

```
sim("ex_simrf_dc_model")  
%
```





- The RF Display plot shows the power level of the RF signal. The power level of the RF is about 100 dBm.
- The Blocker Display plot shows the power spectrum centered at the carrier f_{BL} . The power level of the blocker is about 90 dB higher than the signal power of the RF.
- The In-Phase Output plot shows the power spectrum of the in-phase signal at baseband. In the figure, DC power is a direct result of the blocker and the IP2 in the mixers.
- The Quadrature Output plot shows the power spectrum of the quadrature signal at baseband.

Modeling IMD in System-Level Components

The IP2 and IP3 parameters specify the second- and third-order intercept points of Amplifier and Mixer blocks:

- The amplifiers have infinite IP2 and IP3, so the amplifiers are linear.
- IP2 of the mixer is 15 dB

Amplifier and Mixer components have specified gains and noise figures:

- The gain and noise figure in the LNA stage are 25 dB and 6 dB, respectively.
- The gain and noise figure in the mixing stage are 10 dB and 10 dB. The Input impedance parameters of the two mixers are both 100 ohms, which sum in parallel to a resistance of 50 ohms to match the output impedance of the LNA.
- The gain and noise figure in the final amplification stage are 20 dB and 15 dB, respectively.

To calculate RF system noise figure, use the Friis equation:

$$F_{\text{sys}} = F_1 + F_2 - 1/G_1 + F_3 - 1/G_1G_2 + \dots + F_n - 1/G_1G_2\dots G_{n-1}$$

Where F_n and G_n are the noise figure and gain of the n th stage.

Examining DC Impairments

In addition to intermodulation distortion from IP2, direct-conversion receivers are subject to additional DC impairments. For example, coupling between mixer input and local oscillator (LO) ports causes self-mixing of the LO. For more information, see “Executable Specification of a Direct Conversion Receiver” on page 8-70.

See Also

Related Examples

- “Top-Down Design of an RF Receiver” on page 8-166

Noise in RF Systems

In this section...

“White and Colored Noise” on page 2-7

“Thermal Noise” on page 2-7

“Phase Noise” on page 2-8

“Noise Figure” on page 2-8

Noise in an RF system is generated internally by active components in the system or introduced externally like channel interference or antenna.

White and Colored Noise

White noise: Noise with a flat frequency spectrum is called **white noise**. White noise has equal power across all frequencies of the system band width.

Colored noise: Noise with power that varies according to frequencies in an RF system bandwidth is called **colored noise**.

To simulate white or colored noise in RF Blockset, use the Noise block.

Thermal Noise

Thermal noise is the most common noise introduced in an RF system. This noise is generated internally by active components in the system or externally due to channel interference or antenna. Thermal noise is also known as Johnson or Nyquist noise. The equation for thermal noise is:

$$P_N = k_B T B$$

- k_B is Boltzmann's constant, equal to 1.38065×10^{-23} J/K.
- T is the noise temperature, specified as 293.15 K in this example.
- R_s is the noise source impedance, specified as 50Ω in this example to agree with the resistance value of the Resistor block labeled R1.
- B is the bandwidth.

At room temperature, the thermal noise generated by system with a band width of 1 Hz is -174 dBm.

To generate thermal noise in a RF Blockset system use Resistor and Configuration block. You can also generate thermal noise using the S-Parameters block if the S-parameter is passive.

Thermal noise floor in RF Blockset is defined by the equation:

$$P_{noise} = 4k_B T R_s \Delta f$$

where:

- k_B is Boltzmann's constant, equal to 1.38065×10^{-23} J/K.
- T is the noise temperature, specified as 293.15 K in this example.
- R_s is the noise source impedance, specified as 50Ω in this example to agree with the resistance value of the Resistor block labeled R1.

- Δf is the noise bandwidth.

Phase Noise

Phase noise is a short-term fluctuation in the phase of an oscillator signal. This noise introduces uncertainty in the detection of digitally modulated signals. Phase noise is defined as the ratio of power in one-phase modulation sideband to the total signal power per unit bandwidth. It is expressed in decibels relative to the carrier power per hertz of bandwidth (dBc/Hz). To know how to model LO phase noise in an oscillator see, “Model LO Phase Noise” on page 7-42.

Noise Figure

Noise figure value determines the degradation of signal to noise ratio of a signal as it goes through the network. Noise figure is defined by the equation:

$$N_f = \frac{\frac{\text{Signal}}{\text{Noise}} \text{ at the input}}{\frac{\text{Signal}}{\text{Noise}} \text{ at the output}}$$

Excessive noise figure in the system causes the noise to overwhelm the signal, making the signal unrecoverable. Noise figure is a function of frequency but it is independent of the bandwidth of the system. Noise figure is expressed in dB.

In RF Blockset, you can specify noise figure for an RF system using the Amplifier or Mixer blocks.

See Also

More About

- “Model System Noise Figure” on page 1-2

Testbenches

- “Use RF Measurement Testbench for RF-to-IQ Converter” on page 3-2
- “Using RF Measurement Testbench for IQ-to-RF Converter” on page 3-10

Use RF Measurement Testbench for RF-to-IQ Converter

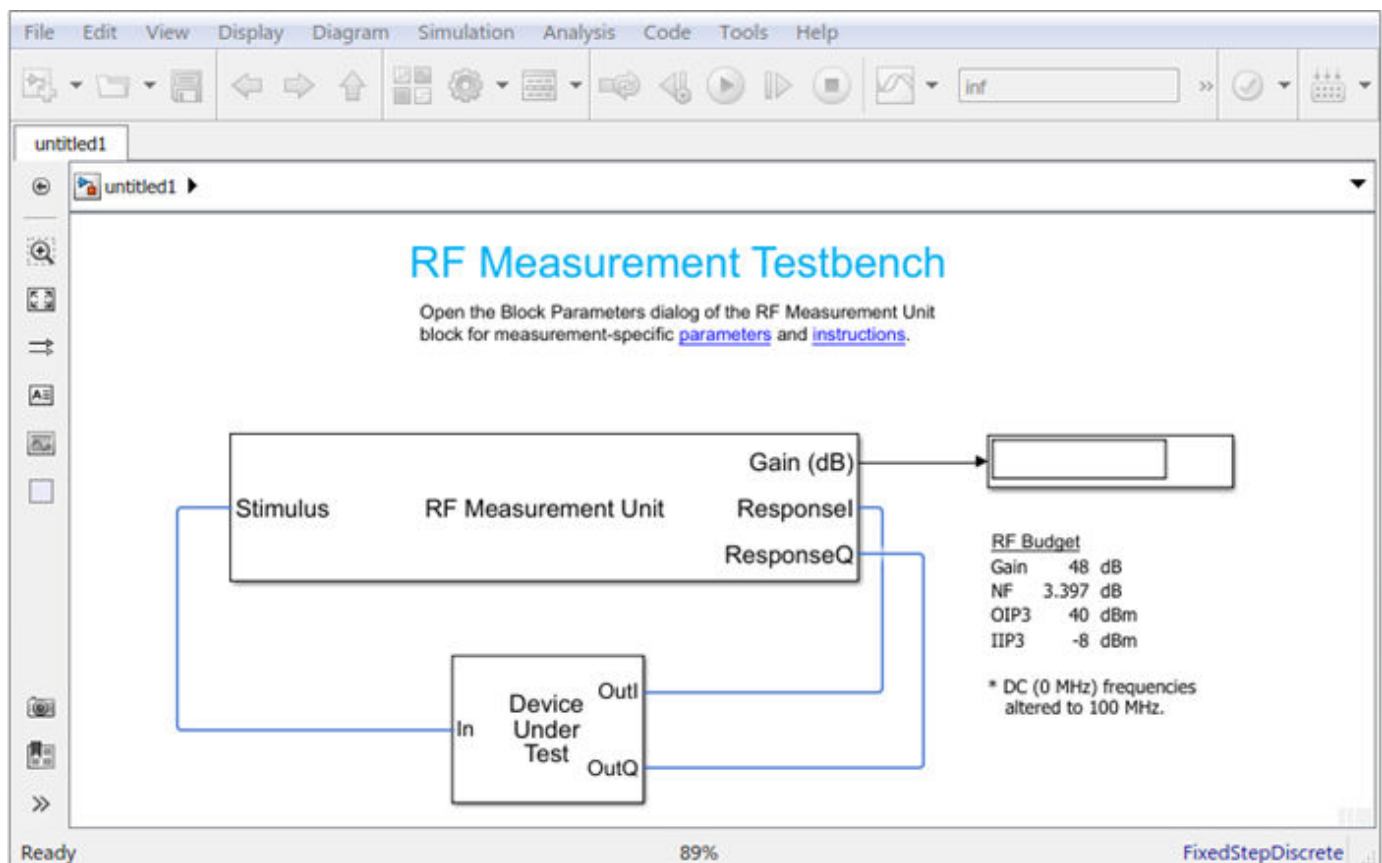
In this section...

“Device Under Test” on page 3-3

“RF Measurement Unit” on page 3-3

“RF Measurement Unit Parameters” on page 3-5

Use the RF Measurement Testbench to measure various quantities of an RF-to-IQ converter. Measurable quantities include cumulative gain, noise figure, and nonlinearity (IP3) values. To open the testbench and measure the quantities, use the **RF Budget Analyzer** app to create an RF-to-IQ converter and then click Export > Measurement testbench.

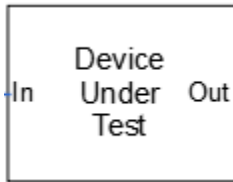


The testbench has two subsystems:

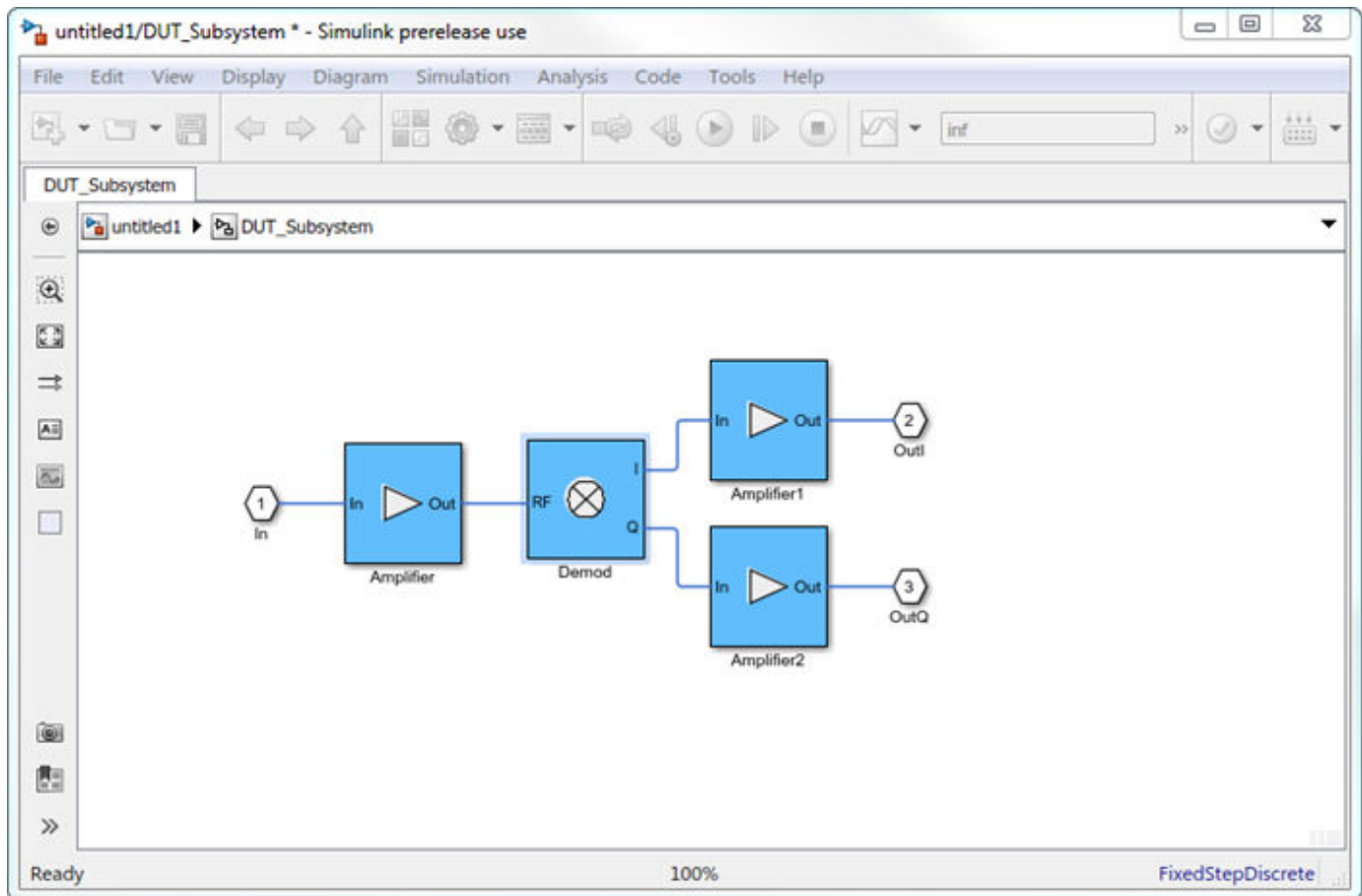
- RF Measurement Unit
- Device Under Test

The testbench display shows the measured output values of the gain, NF (noise figure), IP3 (third-order intercept), and other quantities etc.

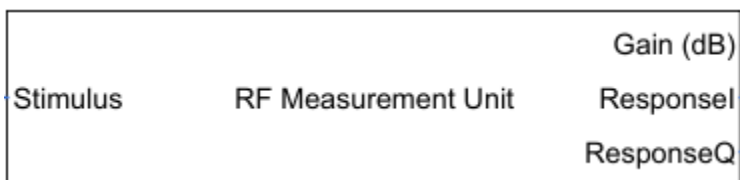
Device Under Test



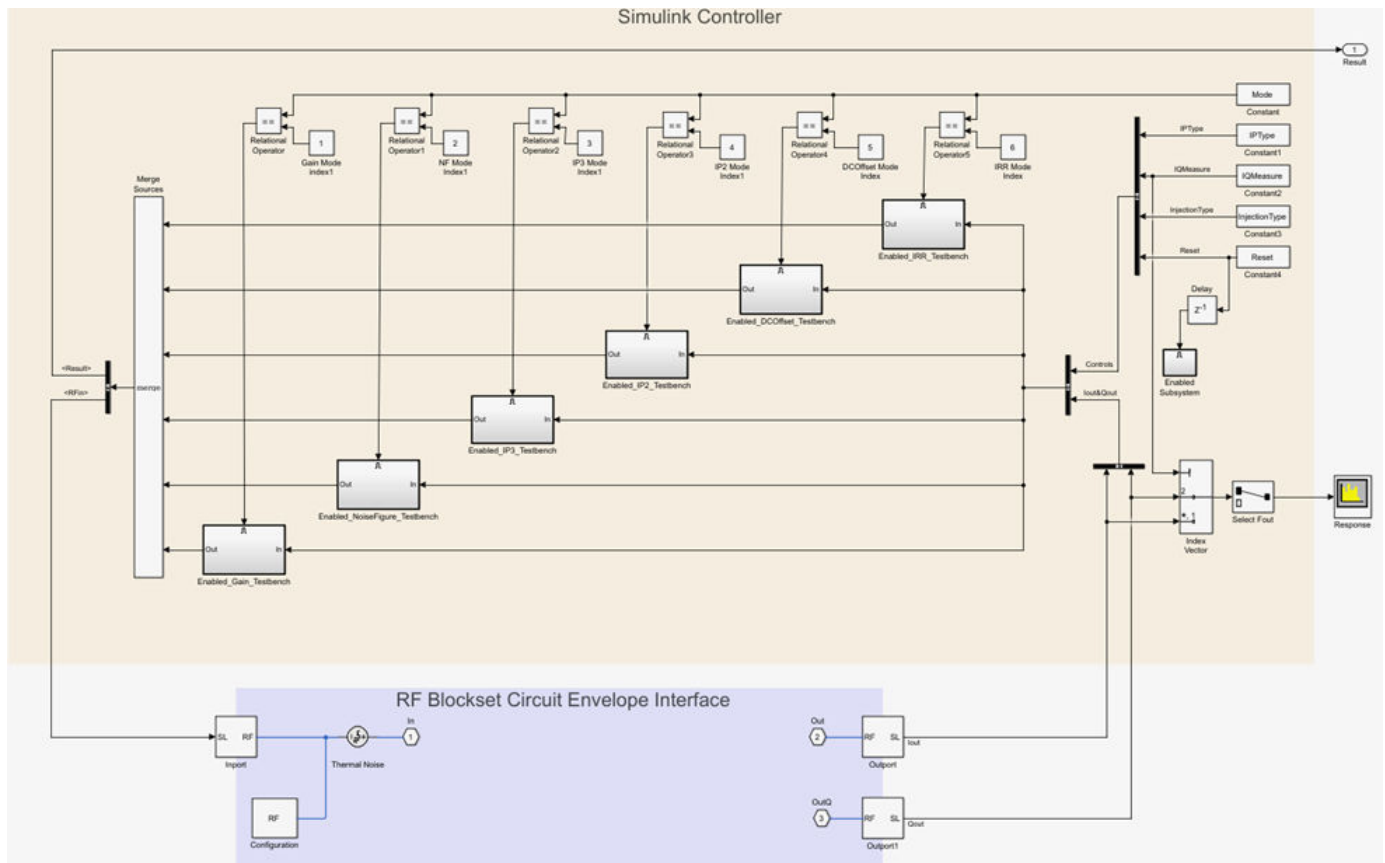
The Device Under Test subsystem contains the RF system exported from the app.



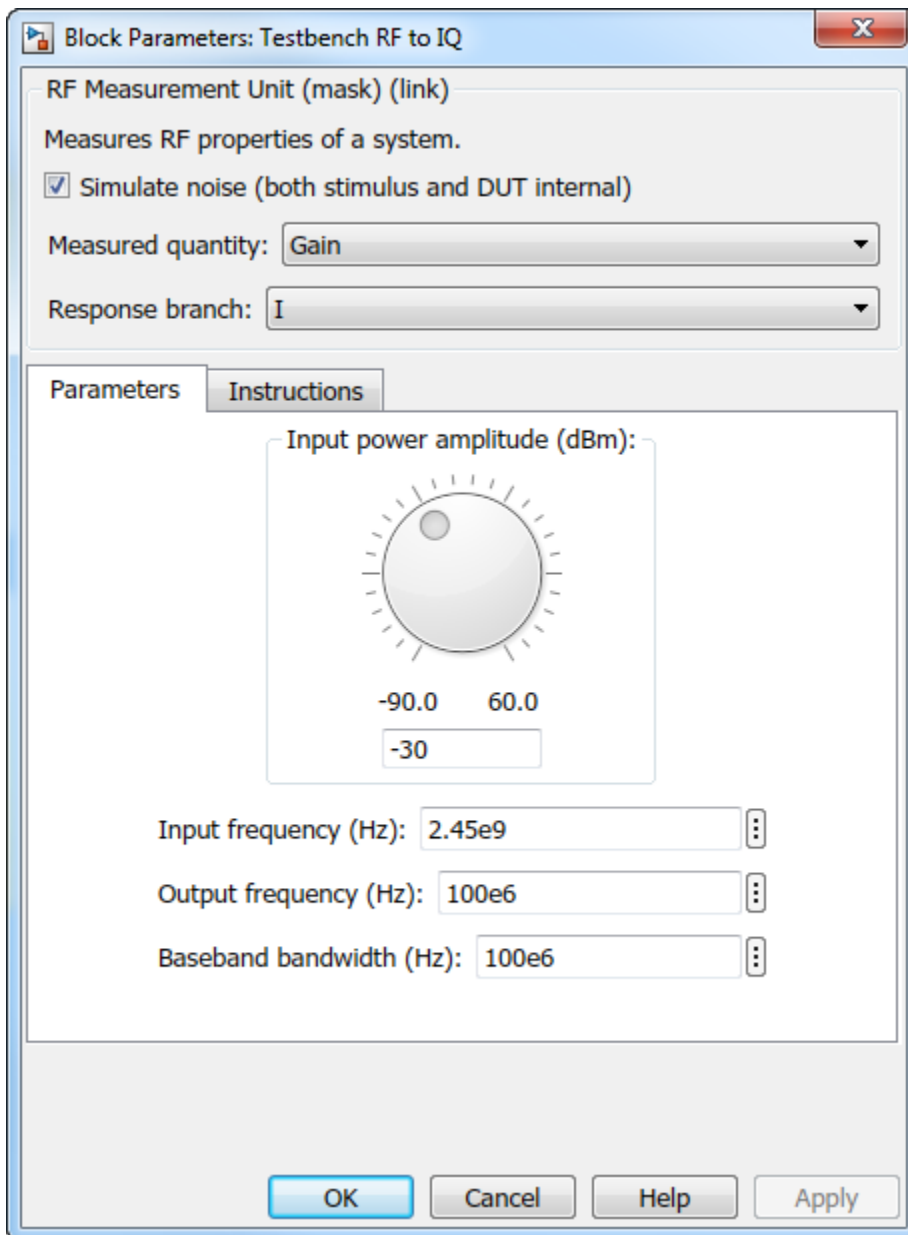
RF Measurement Unit



The RF Measurement Unit subsystem consists of a Simulink Controller and RF Blockset Circuit Envelope interface. The RF Blockset interface is used as input and output from the DUT.



RF Measurement Unit Parameters



- **Simulate noise (both stimulus and DUT)** – Select this check box to enable noise modeling in the stimulus signal entering the DUT and inside the DUT.
- **Measured quantity** – Choose the quantity you want to measure:
 - **Gain** - Measure the transducer gain of the converter, assuming a load of 50 ohm. If you choose only I or only Q from **Response branch**, you see only half the value of the measured gain.
 - **NF** - Measure the noise figure value at the output of the converter.
 - **IP3** - Measure the output or input third-order intercept (IP3).
 - **IP2** - Measure the output or input second-order intercept (IP2).

- **DC Offset** - Measure the DC level interference centered on the desired signal due to LO leakage mixing with input signal.
- **Image Rejection Ratio** - Measure the image rejection ratio required to cancel the effect of images at the RF input signal.

By default, the testbench measures the Gain. The contents in the **Instructions** tab changes according to the **Measured quantity** value.

- **IP Type** — Choose the type of intercept points (IP) to measure: **Output referred** or **Input referred**.

By default, the testbench measures **Output referred**. This option is available when you set the **Measured quantity** to IP2 or IP3.

- **Injection Type** - Choose the local oscillator (LO) injection for image rejection ratio: **Low-side** or **High-side**.

By default, the testbench measures the **Low-side**. This option is available when you set the **Measured quantity** to **Image Rejection Ratio**.

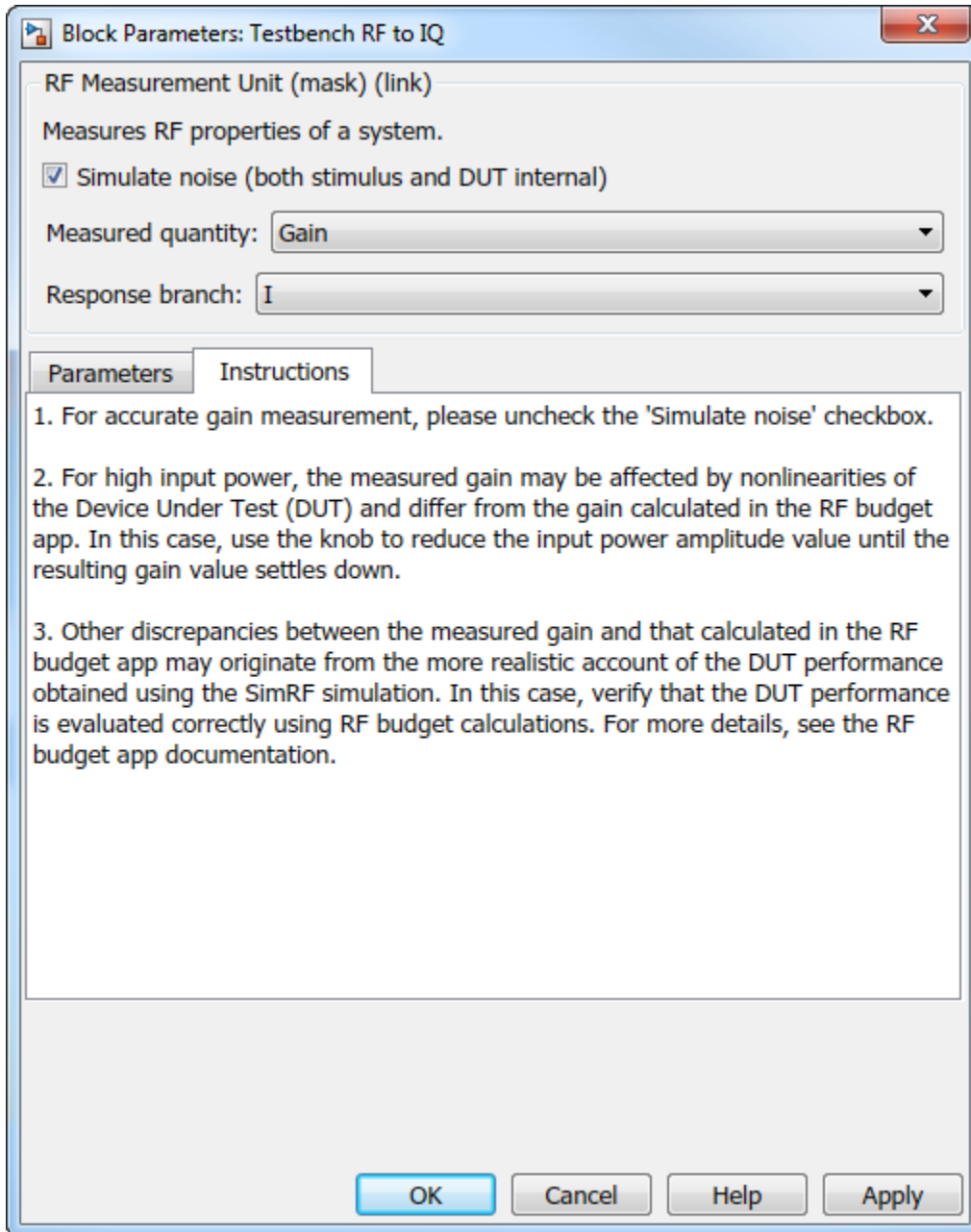
- **Response branch** — Choose the output branch you want to measure from:
 - **I only (Q=0)** - Output signal measured at the in-phase branch.
 - **Q only (I=0)** - Output signal measured at the quadrature branch.

This option is not available when you set the **Measured quantity** to **Image Rejection Ratio**.

Parameters Tab

- **Input power amplitude (dBm)** - Available input power to the DUT. You can change the input power by manually specifying a value or by turning the knob. When measuring DC Offset, this input field is **Input RMS voltage (dBmV)**, because the Offset is measured in voltage units. The specified voltage represents the voltage falling on the input ports of the DUT.
- **Input frequency (Hz)** - Carrier frequency fed at the RF input of the DUT.
- **Output frequency (Hz)** - Output frequency to measure the I and Q outputs of the DUT. By default, this frequency is one bandwidth above DC, to allow meaningful measurement.
- **Baseband bandwidth (Hz)** - Bandwidth of the input signal.
- **Ratio of test tone frequency to baseband bandwidth** - Position of the test tones used for IP3 measurements. By default, the value is 1/8.

Instructions Tab



Instructions for Gain Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate gain measurement. Select the check box to account for the noise.
- Change the **Input power amplitude (dBm)** or turn the knob to reduce the input power amplitude. For high input power, nonlinearities in the DUT can affect the gain measurements.

Instructions for NF Measurement

- The testbench measures the spot NF calculated. This calculation assumes a frequency-independent system within a given bandwidth. To simulate a frequency-independent system and calculate the correct NF value, reduce the baseband bandwidth until this condition is fulfilled. In common RF systems, the bandwidth is reduced below 1 kHz for NF testing.
- Change **Input power amplitude (dBm)** or turn the knob to reduce or increase the input power amplitude. For high input power, nonlinearities in the DUT can affect the NF measurements. For low input power, the signal is too close or below the noise floor of the system. As a result, the NF fails to converge.

Instructions for IP3 and IP2 Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate IP3 and IP2 measurement.
- Change **Input power amplitude (dBm)** or turn the knob to reduce the input power amplitude. For high input power, higher-order nonlinearities in the DUT can affect the OIP3 and IIP3 measurements.

Instructions for DC Offset Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate DC offset measurement.
- Correct calculation of the DC offset assumes a frequency-independent system in the frequencies surrounding the test tones. Reduce the frequency separation between the test tones or reduce the baseband bandwidth until this condition is fulfilled. In common RF systems, the bandwidth is reduced below 1 KHz for DC offset testing.
- . Change **Input RMS voltage amplitude (dBmV)** or turn the knob to reduce the input RMS voltage amplitude. For high input RMS voltage, higher-order nonlinearities in the DUT can affect the DC offset measurements

Instructions for Image Rejection Ratio

- Clear **Simulate noise (both stimulus and DUT)** for accurate OIP3 and IIP3 measurement.
- Correct calculation of the image rejection ratio (IRR) assumes a frequency-independent system in the frequencies surrounding the test tones. Reduce the frequency separation between the test tones or reduce the baseband bandwidth until this condition is fulfilled. In common RF systems, the bandwidth is reduced below 1 KHz for IRR testing.
- . Change **Input power amplitude (dBm)** or turn the knob to reduce the input power amplitude. For high input power, higher-order nonlinearities in the DUT can affect the image rejection ratio measurement.

For all measurements using the testbench, you cannot correct result discrepancies using the **RF Budget Analyzer** app. The RF Blockset testbench provides true RF circuit simulation that incorporates RF phenomena including saturation and interaction between multiple tones and harmonics in nonlinear devices. These RF phenomena are not yet incorporated in **RF Budget Analyzer**, leading to some differences in the values between the testbench and the app.

See Also

RF Budget Analyzer | Amplifier | Mixer

More About

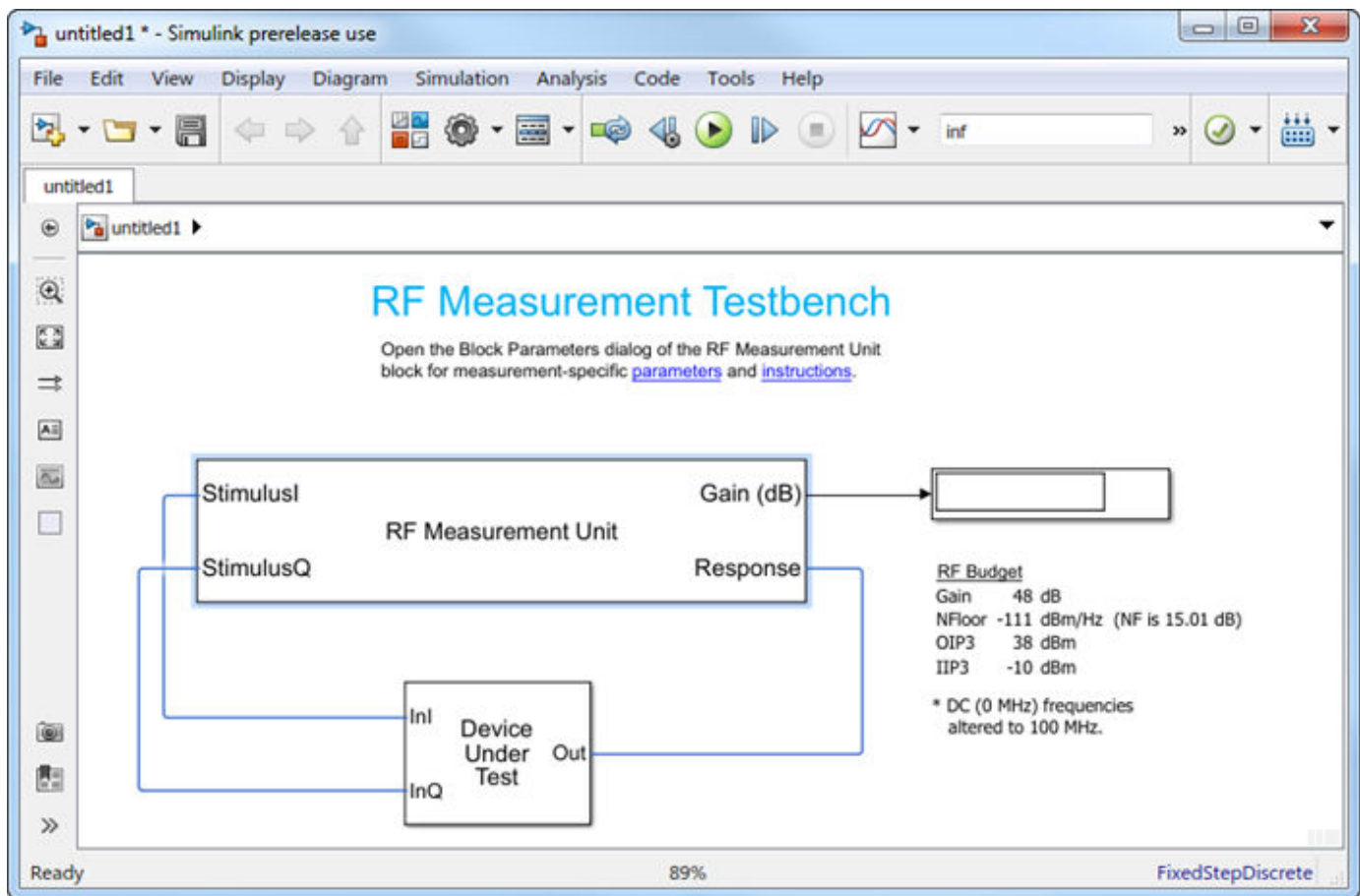
- “Using RF Measurement Testbench for IQ-to-RF Converter” on page 3-10
- “Using RF Measurement Testbench”

Using RF Measurement Testbench for IQ-to-RF Converter

In this section...

- “Device Under Test” on page 3-11
- “RF Measurement Unit” on page 3-11
- “RF Measurement Unit Parameters” on page 3-13

Use the RF Measurement Testbench to measure various quantities of an IQ-to-RF converter system. Measurable quantities include cumulative gain, noise figure, and nonlinearity (IP3) values. To open the testbench and measure the quantities, use the **RF Budget Analyzer** app to create an RF system and then click **Export > Measurement testbench**.

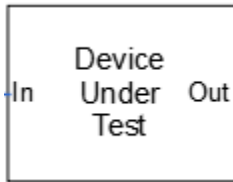


The testbench has two subsystems:

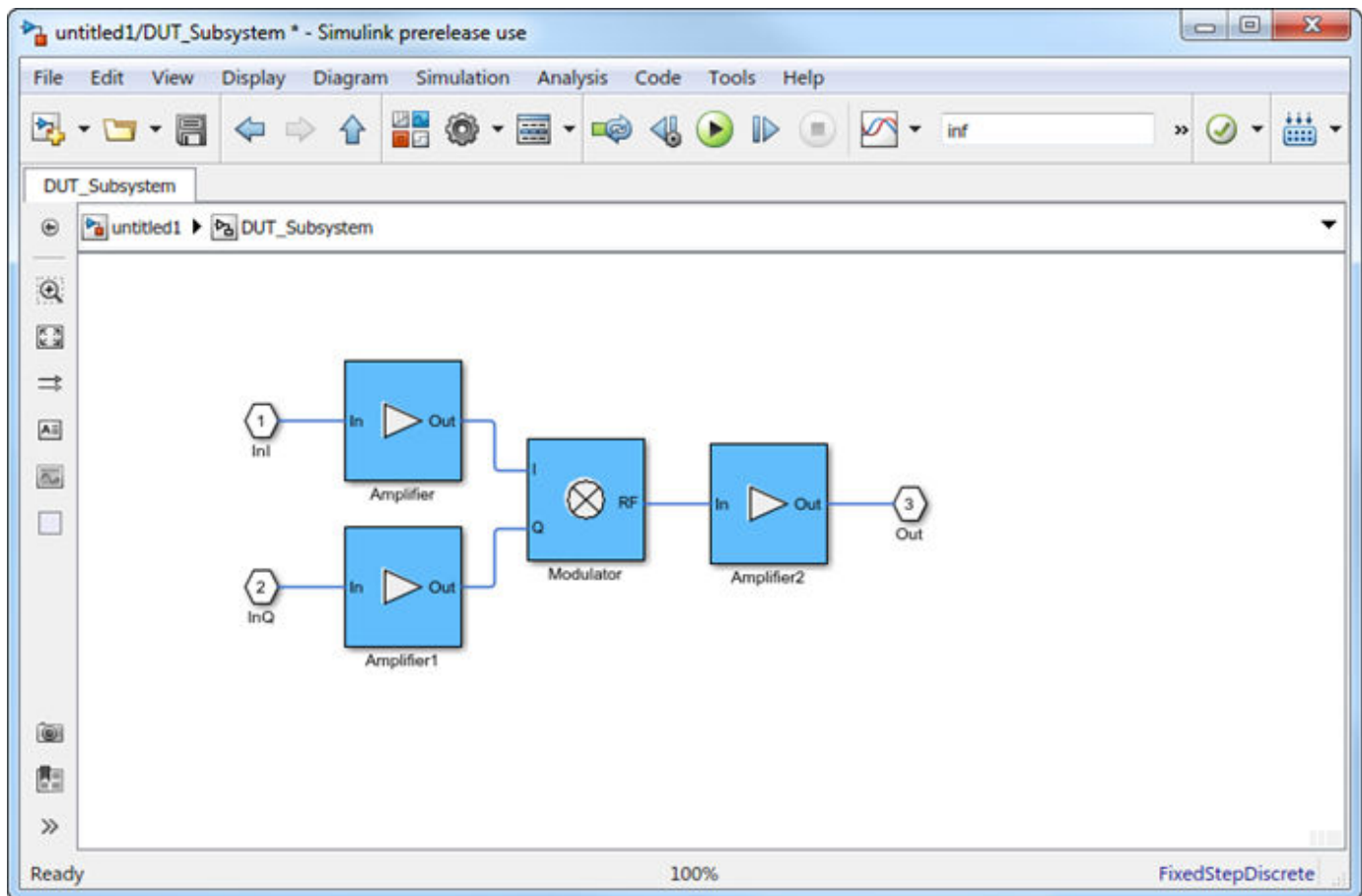
- RF Measurement Unit
- Device Under Test

The testbench display shows the measured output values of the gain, NF (noise figure), IP3 (third-order intercept), and other quantities.

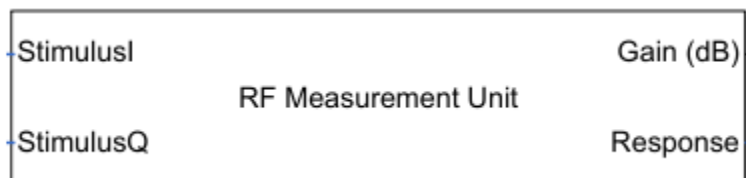
Device Under Test



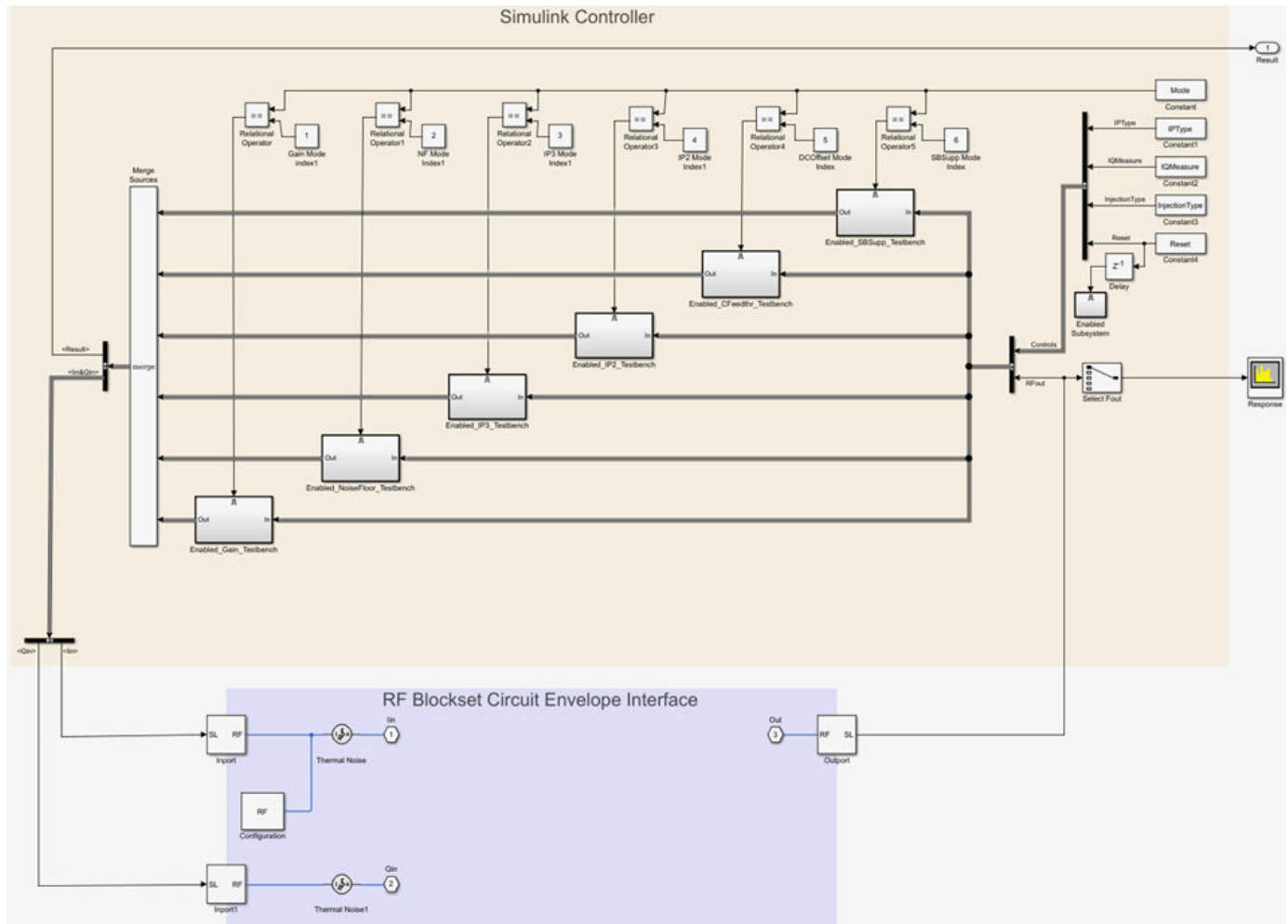
The Device Under Test subsystem contains the RF system exported from the app.



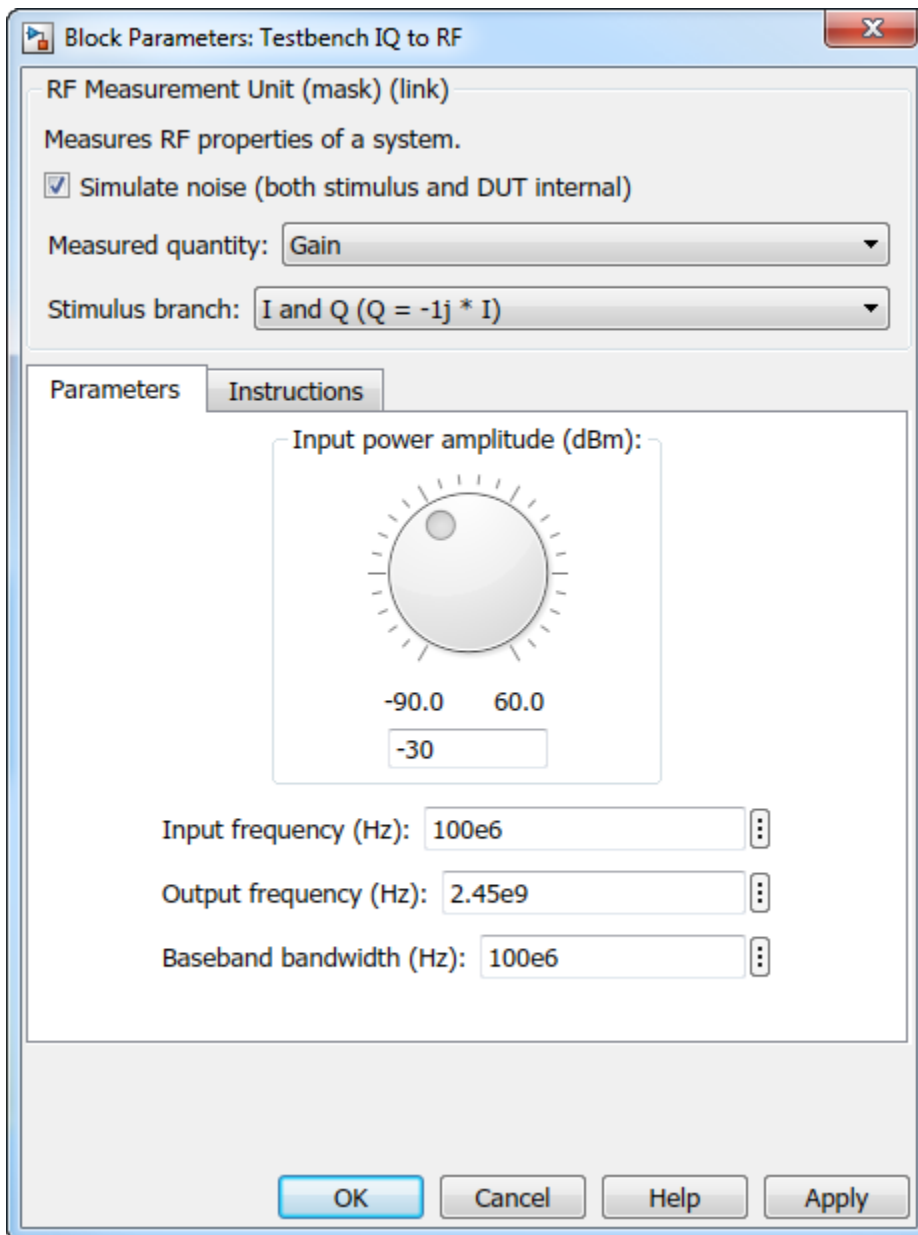
RF Measurement Unit



The RF Measurement Unit subsystem consists of a Simulink Controller and RF Blockset Circuit Envelope interface. The RF Blockset interface is used as input and output from the DUT.



RF Measurement Unit Parameters



- **Simulate noise (both stimulus and DUT)** – Select this check box to enable noise modeling in the stimulus signal entering the DUT and inside the DUT.
- **Measured quantity** – Choose the quantity you want to measure:
 - **Gain** - Measure the transducer gain of the converter. If you choose only I or only Q from **Stimulus branch**, you only see half the value of the measured gain.
 - **Noise Floor** - Measure the noise floor value of the converter.
 - **IP3** - Measure the output or input third-order intercept (IP3).
 - **IP2** - Measure the output or input second-order intercept (IP2).

- **Carrier Feedthrough** - Measure the leakage of carrier tone into the RF spectrum due to imbalances in the in-phase and quadrature phase inputs.
- **Sideband Suppression** - Measure the sideband suppression required for the ideal cancellation of image signals around the RF output signal.

By default, the testbench measures **Gain**. The contents in the **Instructions** tab changes according to the **Measured quantity** value.

- **IP Type** - Choose the type of intercept points (IP) to measure: **Output referred** or **Input referred**,

By default, the testbench measures **Output referred**. This option is available when you set the **Measured quantity** to IP2 or IP3.

- **Injection Type** - Choose the local oscillator (LO) injection for sideband suppression: **Low-side** or **High-side**,

By default, the testbench measures **Low-side**. This option is available when you set the **Measured quantity** to **Sideband Suppression**.

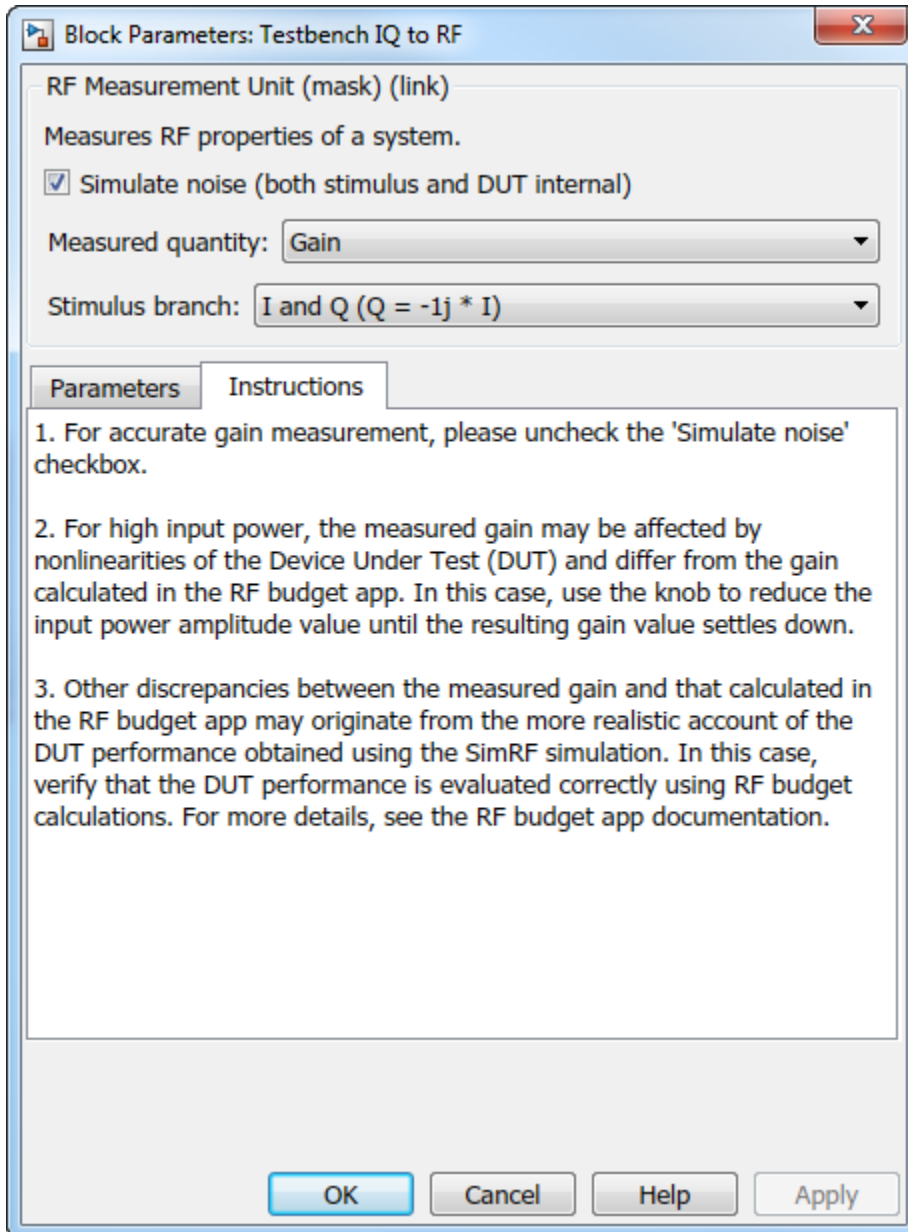
- **Stimulus branch** — Choose the branch you want to use as input stimulus for the measurement:
 - **I and Q ($Q=-j*I$)** — Signal measured is based on a combination of input signals. Quadrature input is same as the in-phase input but is -90 degrees out of phase.
 - **I and Q ($Q=j*I$)** — Signal measured is based on a combination of input signals. Quadrature input is same as the in-phase input but is 90 degrees out of phase.
 - **I only ($Q=0$)** — Signal measured is only the output of the in-phase input signal. Gain measured using this input is only one quarter of the total output signal gain.
 - **Q only ($I=0$)** — Signal verified is only the output of the quadrature input signal. Gain measured using this input is only one quarter of the total output signal gain.

Parameters Tab

- **Input power amplitude (dBm)** — Available input power to the DUT. You can change the input power by manually specifying or by turning the knob. When measuring **Carrier Feedthrough**, this input field is **Input RMS voltage (dBmV)**, because the feedthrough is measured in voltage units. The specified voltage represents the voltage falling on the input ports of the DUT.
- **Input frequency (Hz)** — Carrier frequency fed into the I and Q inputs of the DUT. By default, this frequency is by default one bandwidth above DC.
- **Output frequency (Hz)** — Output frequency to measure the DUT.
- **Baseband bandwidth (Hz)** — Bandwidth of the input signal.
- **Ratio of test tone frequency to baseband bandwidth** — Position of the test tones used for IP3 measurements. By default, the value is 1/8.

This option is only available when you set **Measured quantity** to IP2, IP3, and **Carrier Feedthrough**.

Instructions Tab



Instructions for Gain Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate gain measurement. Select the check box for account for noise.
- Change the **Input power amplitude (dBm)** or turn the knob to reduce the input power amplitude. For high input power, nonlinearities in the DUT can affect the gain measurements.

Instructions for Noise Floor Measurement

- The testbench measures the spot noise floor calculated. This calculation assumes a frequency-independent system within a given bandwidth. To simulate a frequency-independent system and

calculate the correct noise floor value, reduce the baseband bandwidth until this condition is fulfilled. In common RF systems, the bandwidth is reduced below 1 kHz for noise floor testing.

- Change **Input power amplitude (dBm)** or turn the knob to reduce or increase the input power amplitude. For high input power, nonlinearities in the DUT can affect the noise floor measurements.

Instructions for IP3 and IP2 Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate IP3 and IP2 measurement.
- Change **Input power amplitude (dBm)** or turn the knob to reduce the input power amplitude. For high input power, higher-order nonlinearities in the DUT can affect the IP3 and IP2 measurements.

Instructions for Carrier Feedthrough Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate IP3 and IP2 measurement.
- Change **Input RMS voltage amplitude (dBmV)** or turn the knob to reduce the input RMS voltage amplitude. For high input RMS voltage, higher-order nonlinearities in the DUT can affect the carrier feedthrough measurements
- Correct calculation of the carrier feedthrough assumes a frequency-independent system in the frequencies surrounding the test tones. Reduce the frequency separation between the test tones or reduce the baseband bandwidth until this condition is fulfilled. In common RF systems, the bandwidth is reduced below 1 KHz for carrier feedthrough testing.

Instructions for Sideband Suppression Measurement

- Clear **Simulate noise (both stimulus and DUT)** for accurate IP3 and IP2 measurement.
- Change **Input power amplitude (dBm)** or turn the knob to reduce the input power amplitude. For high input power, higher-order nonlinearities in the DUT can affect the sideband suppression measurement.

For all measurements using the testbench, you cannot correct result discrepancies using the **RF Budget Analyzer** app. The RF Blockset testbench provides true RF circuit simulation that incorporates RF phenomena including saturation and interaction between multiple tones and harmonics in nonlinear devices. These RF phenomena are not incorporated in **RF Budget Analyzer**, leading to some differences in the values between the testbench and the app.

See Also

RF Budget Analyzer

RF Blockset Models

Analog Devices Transceiver Models

- “AD9361 Models” on page 4-2
- “AD9361 Testbenches” on page 4-6
- “AD9371 Models” on page 4-9
- “AD9371 Testbenches” on page 4-15

AD9361 Models

In this section...

“AD9361_TX Analog Devices Transmitter” on page 4-3

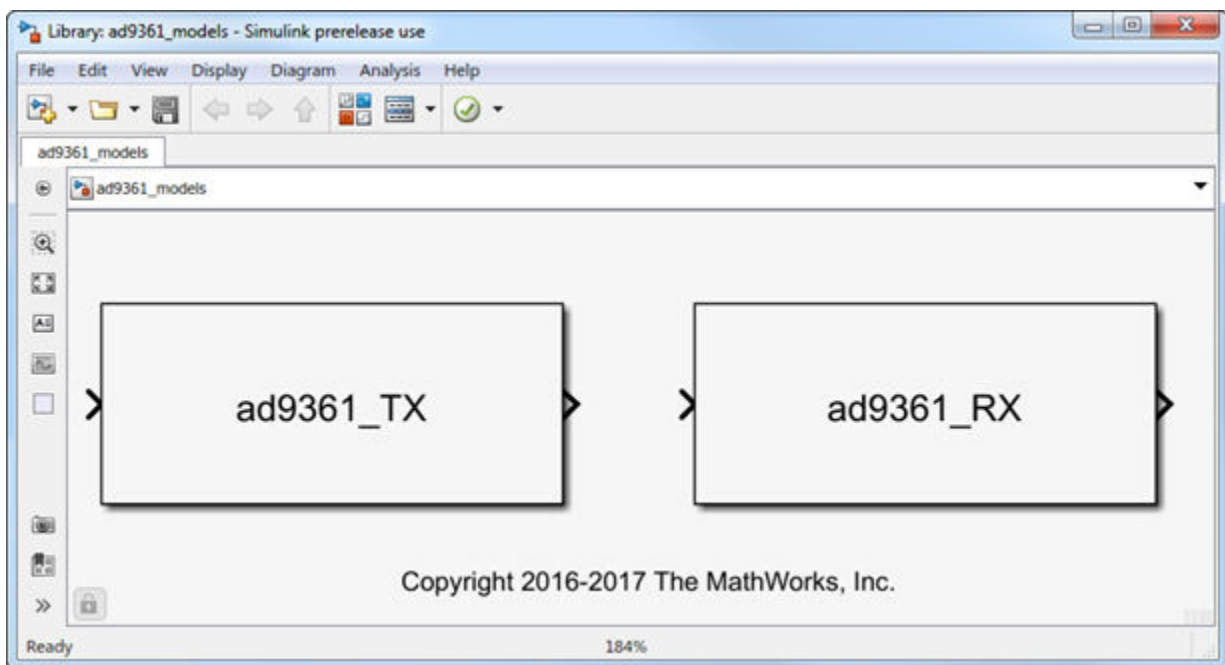
“AD9361_RX Analog Devices Receiver” on page 4-4

You can use the AD9361 models to simulate Analog Devices® AD9361 RF transmitter or receiver designs. These models also help to see the impact of RF imperfections on your transmitted or received signal.

Install Analog Devices RF Transceivers using, `simrfSupportPackages`. You can open models using the Simulink® Library Browser and opening RF Blockset Models for Analog Devices RF Transceivers, or by typing the following in the MATLAB® command prompt:

```
open ad9361_models
```

Choose the model you want from the library:



Note You require these additional licenses to run this model:

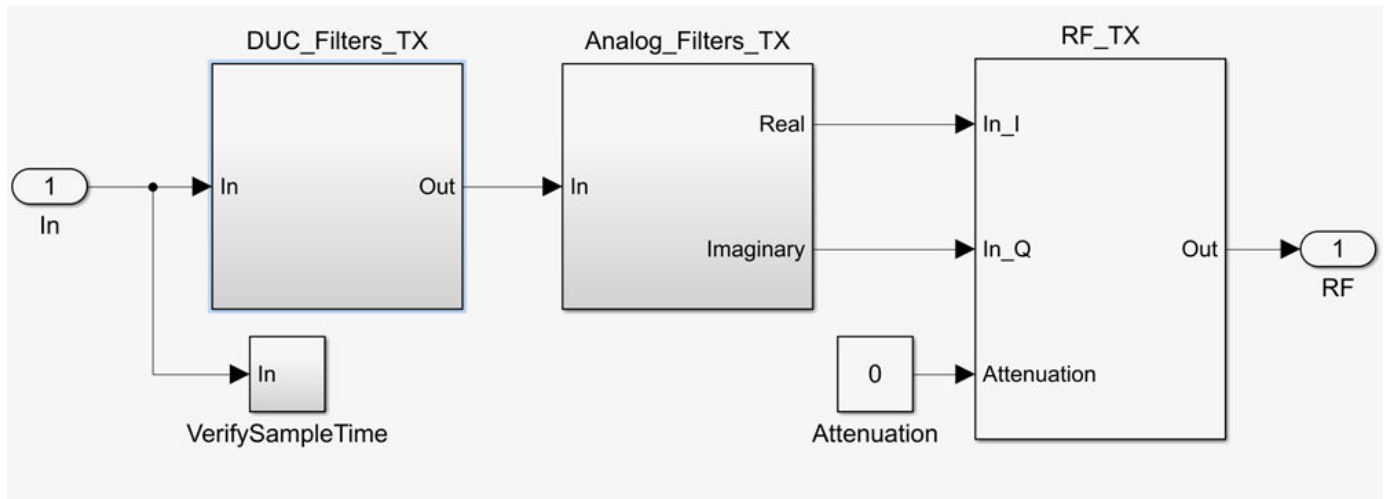
- Communications Toolbox™
- Stateflow®
- Fixed-Point Designer™

Complete documentation on how to use the models is available with the software download.

```
open(ad93xx_getdoc)
```

AD9361_TX Analog Devices Transmitter

Model Description



The transmitter model consists of three stages:

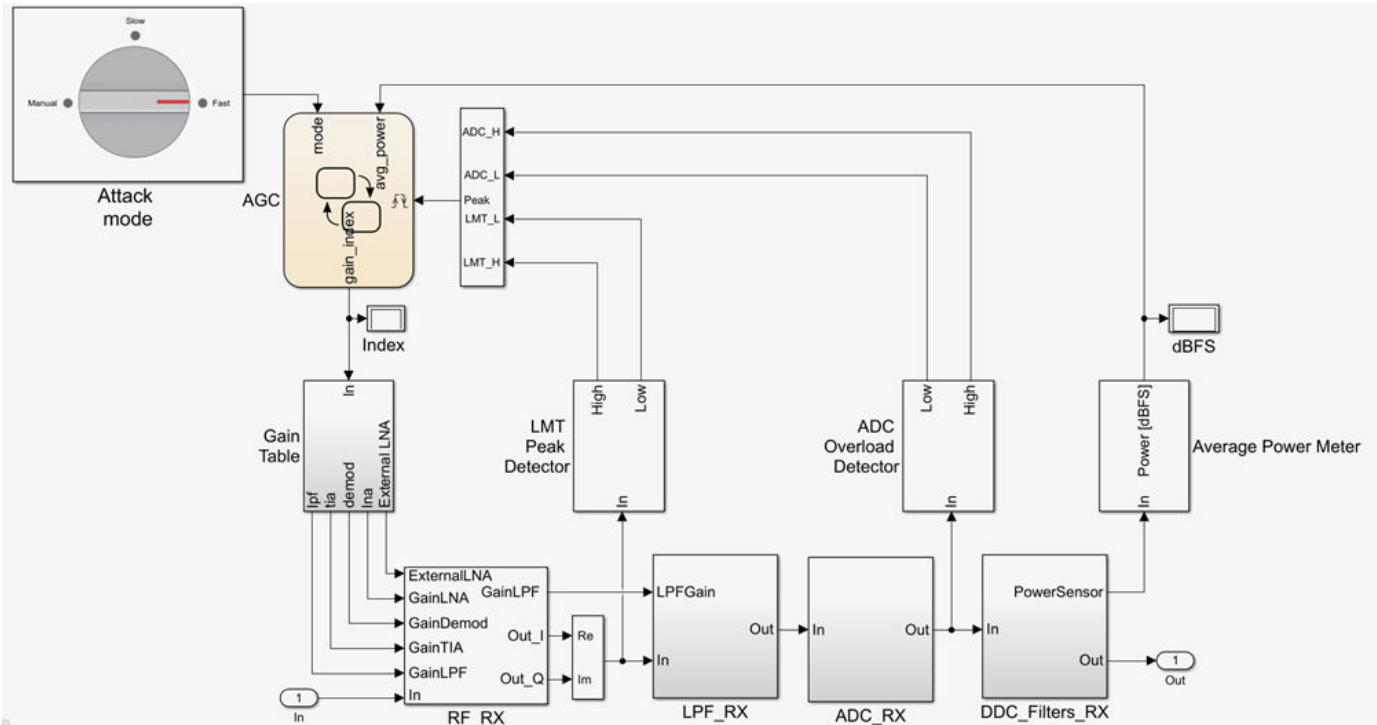
- Digital up-conversion filters (DUC_Filters_TX)
- Analog filters (Analog_Filters_TX)
- RF front end (RF_TX)

You can use the transmitter model to simulate the following behaviors:

- Tunable attenuation
- Oscillator phase noise
- Carrier-dependent noise floor
- Attenuation and carrier-dependent nonlinearities like output-referred third-order intercept (OIP3)
- Attenuation dependent gain imbalance
- Attenuation dependent local oscillator (LO) carrier leak

AD9361_RX Analog Devices Receiver

Model Description



The receiver model consists of:

- RF receiver (RF_RX)
- Analog filters (Analog_Filters_RX)
- Analog to Digital Converter (ADC_RX)
- Digital down-conversion filters (DDC)
- Receiver signal strength indicators
- Automatic gain control (AGC) state machine
- Gain table

You can use the receiver model to simulate the following behaviors:

- Tunable low-noise amplifier, mixer, and trans-impedance amplifier (LMT) gains
- Carrier-dependent noise floor
- Gain and carrier-frequency dependent nonlinearities like output-referred third-order intercept (OIP3)
- Gain and carrier-frequency dependent nonlinearities like output-referred second-order intercept (OIP2)
- Gain dependent gain imbalance
- Gain dependent local oscillator (LO) carrier leak

See Also

simrfSupportPackages

More About

- “AD9361 Testbenches” on page 4-6
- “AD9371 Models” on page 4-9

AD9361 Testbenches

In this section...

“AD9361_TX Analog Devices Transmitter Testbench” on page 4-7

“AD9361_RX Analog Devices Receiver Testbench” on page 4-7

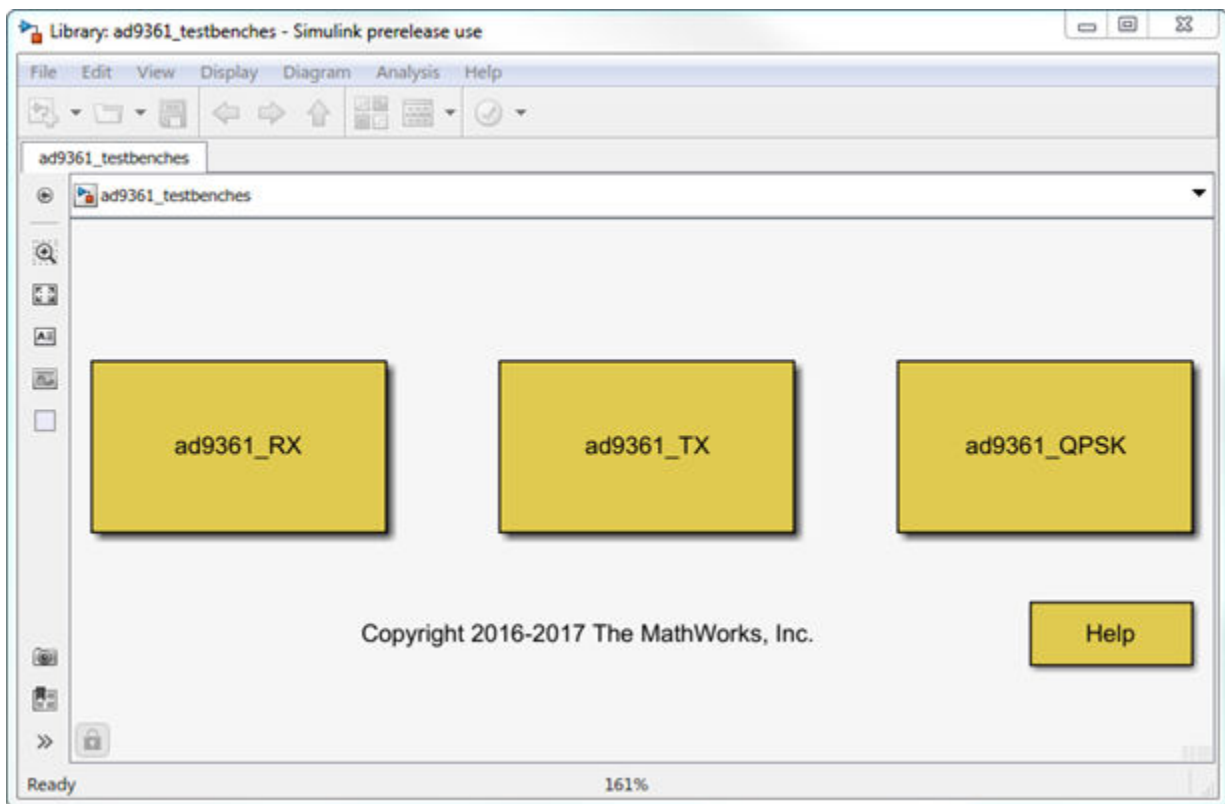
“AD9361_QPSK Analog Devices Testbench” on page 4-8

You can use the AD9361 testbench models to analyze the functioning and values of Analog Devices AD9361 RF transmitter, receiver, or end-to-end designs.

Install Analog Devices RF Transceivers using `simrfSupportPackages`. You can open the testbench models using the Simulink library browser and opening RF Blockset Models for Analog Devices RF Transceivers, or by typing the following in the MATLAB command prompt:

```
open ad9361_testbenches
```

Click to open the AD9361 testbench model from the library:



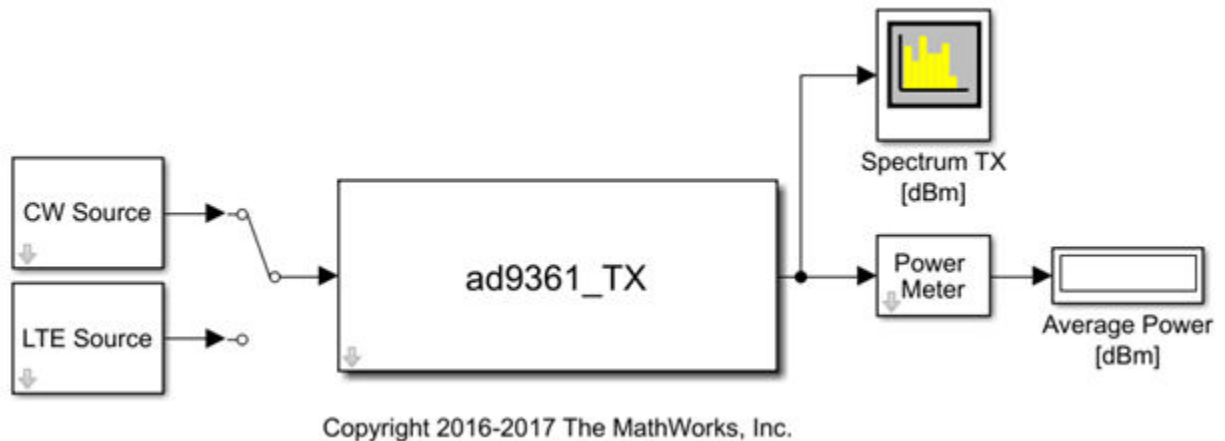
Note You require these additional licenses to run this model:

- Communications Toolbox
- Stateflow
- Fixed-Point Designer

Complete documentation on how to use the models is available with the software download.

`open(ad93xx_getdoc)`

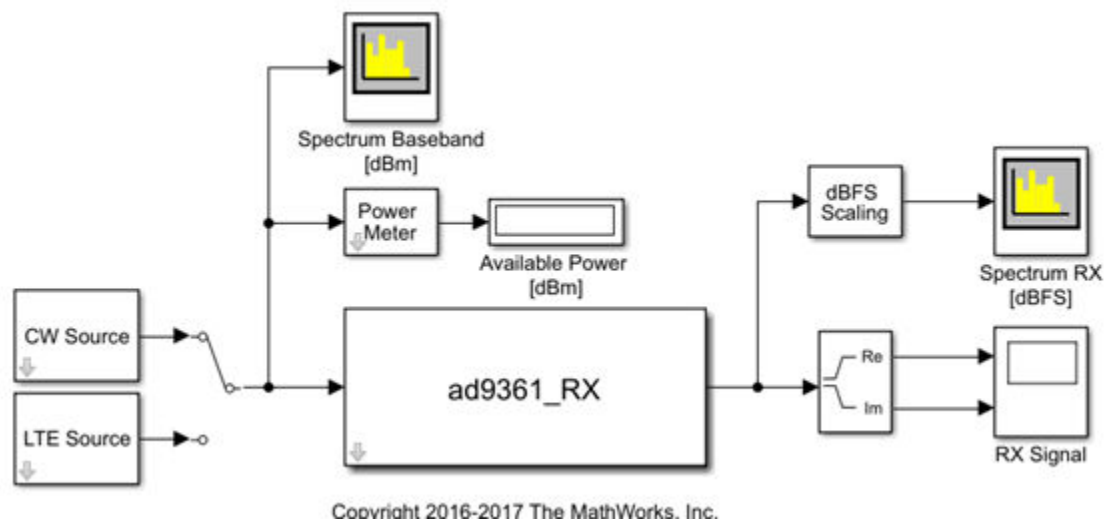
AD9361_TX Analog Devices Transmitter Testbench



The transmitter testbench consists of:

- CW and LTE Signal sources
- AD9361 transmitter which is the device under test
- Spectrum analyzer and power meter for signal visualization

AD9361_RX Analog Devices Receiver Testbench

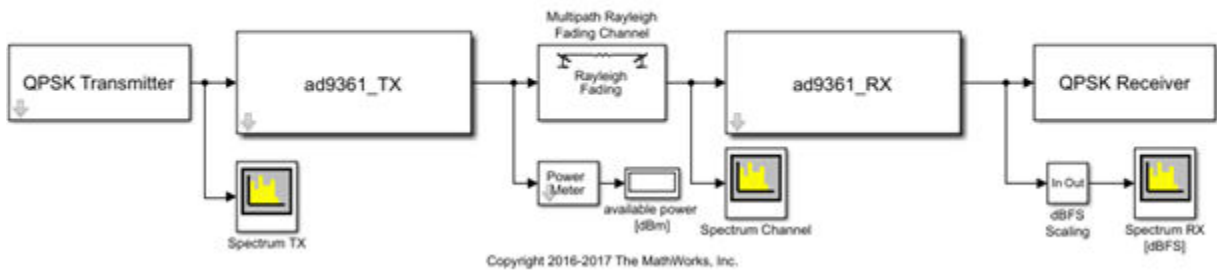


The receiver testbench consists of:

- CW and LTE Signal sources

- AD9361 receiver which is the device under test
- Spectrum analyzer and power meter for signal visualization

AD9361_QPSK Analog Devices Testbench



The QPSK testbench consists of:

- QPSK signal generator
- AD9361 transmitter operating at 2 GHz, with default LTE 5-MHz filter configuration
- Multi-path Rayleigh fading channel
- Multi-path Rayleigh fading channel
- AD9361 receiver operating at 2 GHz, with default LTE 5-MHz filter configuration
- QPSK baseband signal demodulator decoder.

See Also

`simrfSupportPackages`

More About

- “AD9371 Testbenches” on page 4-15
- “AD9361 Models” on page 4-2

AD9371 Models

In this section...

“AD9371_TX Analog Devices Transmitter” on page 4-10

“AD9371_RX Analog Devices Receiver” on page 4-11

“AD9371_ORX Analog Devices Observer Receiver” on page 4-12

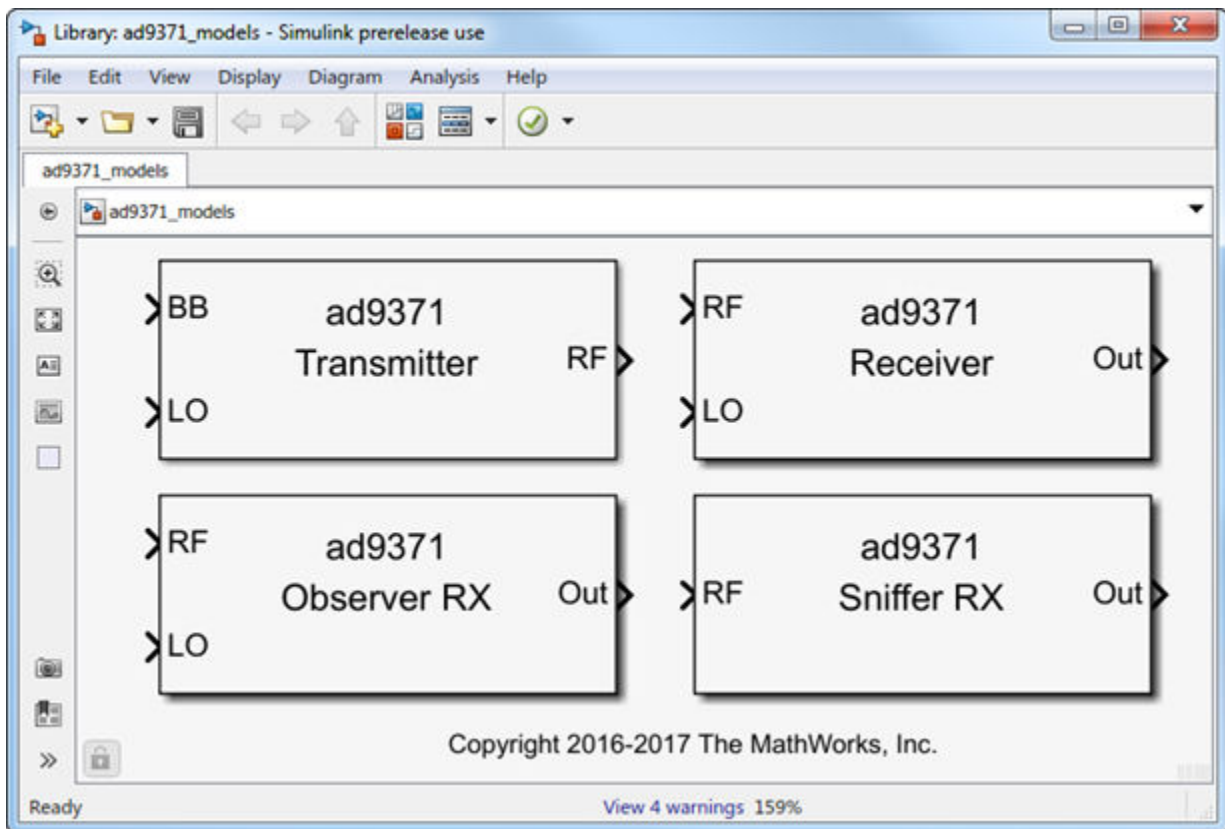
“AD9371_SNF Analog Devices Sniffer Receiver” on page 4-13

You can use the AD9371 models to simulate Analog Devices AD9371 RF transmitter, receiver, observer, and sniffer designs. These models also helps to see the impact of RF imperfections on your transmitted or received signal.

Install Analog Devices RF Transceivers using `simrfSupportPackages`. You can open models using the Simulink Library Browser and opening RF Blockset Models for Analog Devices RF Transceivers, or by typing the following in the MATLAB command prompt:

```
open ad9371_models
```

Choose the model you want from the library:



Note You require these additional licenses to run this model:

- Communications Toolbox

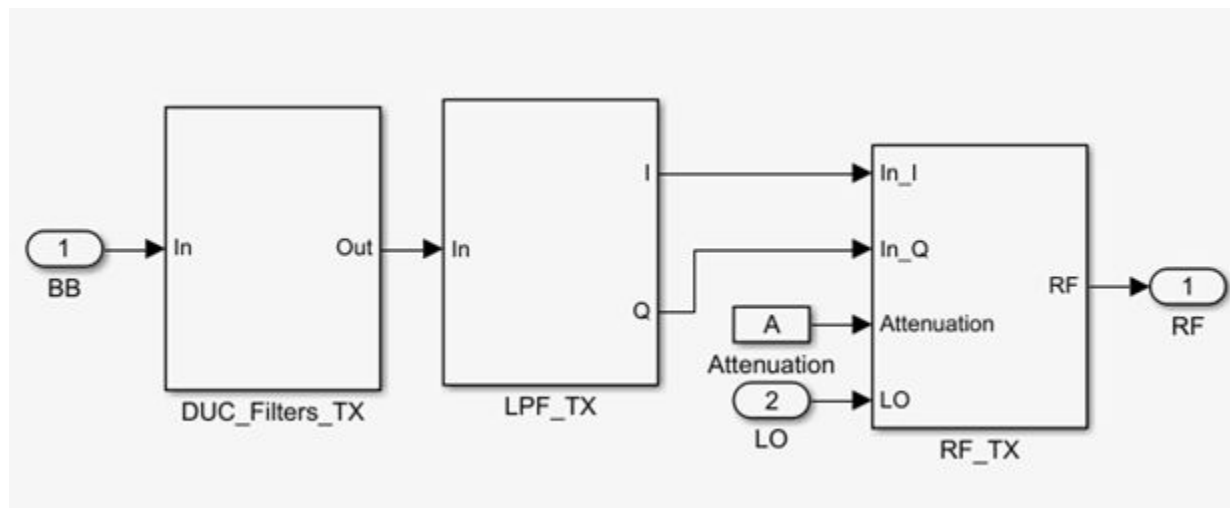
- Stateflow
- Fixed-Point Designer

Complete documentation on how to use the models is available with the software download.

`open(ad93xx_getdoc)`

AD9371_TX Analog Devices Transmitter

Model Description



The transmitter model consists of three stages:

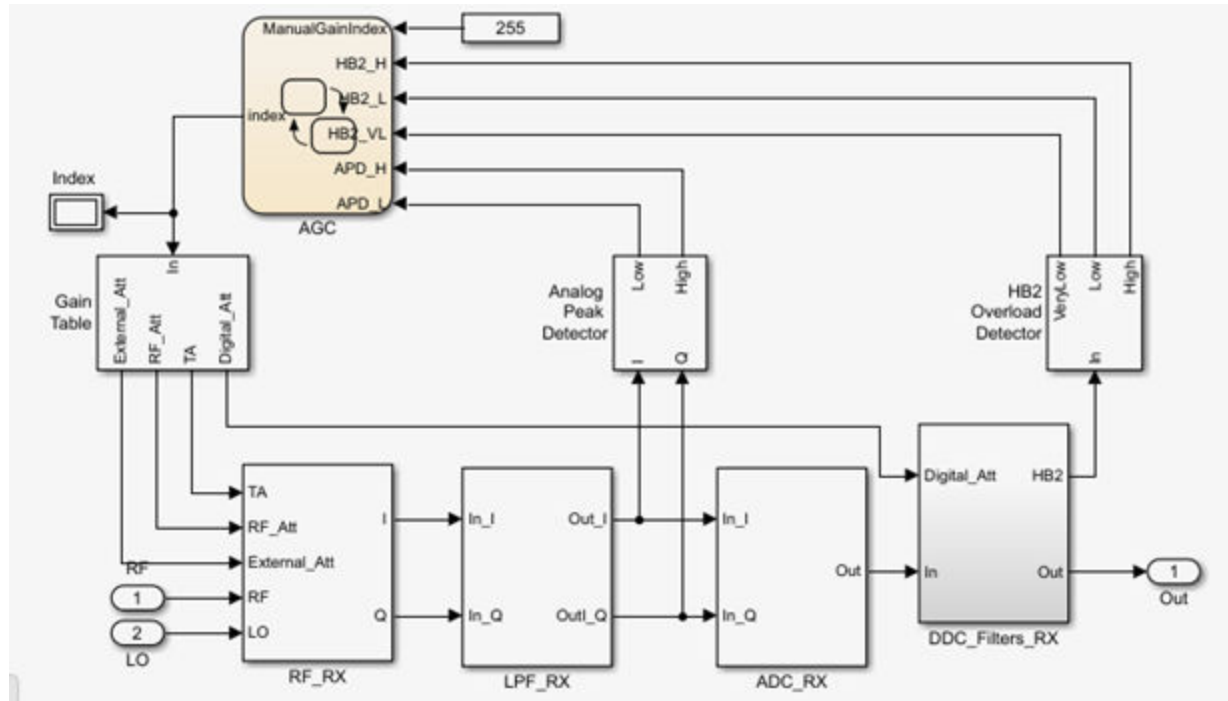
- Digital up-conversion filters (DUC_Filters_TX)
- Analog filters (Analog_Filters_TX)
- RF front end (RF_TX)

You can use the transmitter model to simulate the following behaviors:

- Tunable attenuation
- Attenuation and carrier-frequency dependent noise floor
- Attenuation and carrier-frequency dependent output-referred third-order intercept (OIP3) and output-referred second-order intercept (OIP2)
- Attenuation and carrier-frequency dependent gain imbalance (to model finite image rejection)
- Attenuation and carrier-frequency dependent local oscillator (LO) carrier leakage

AD9371_RX Analog Devices Receiver

Model Description



The receiver model consists of three stages:

- RF receiver (RF_RX)
- Analog filters (LPF_RX)
- Analog-to-Digital converter (ADC_RX)
- Digital down-conversion filters (DDC_Filters_RX)
- Receiver signal strength indicator (RSSI): two power meters to detect the strength of the received signal at different section of the chain (Analog Peak Detector, and HB2 Overload Detector)
- Automatic gain control state machine (AGC)
- Programmable gain table (Gain Table)

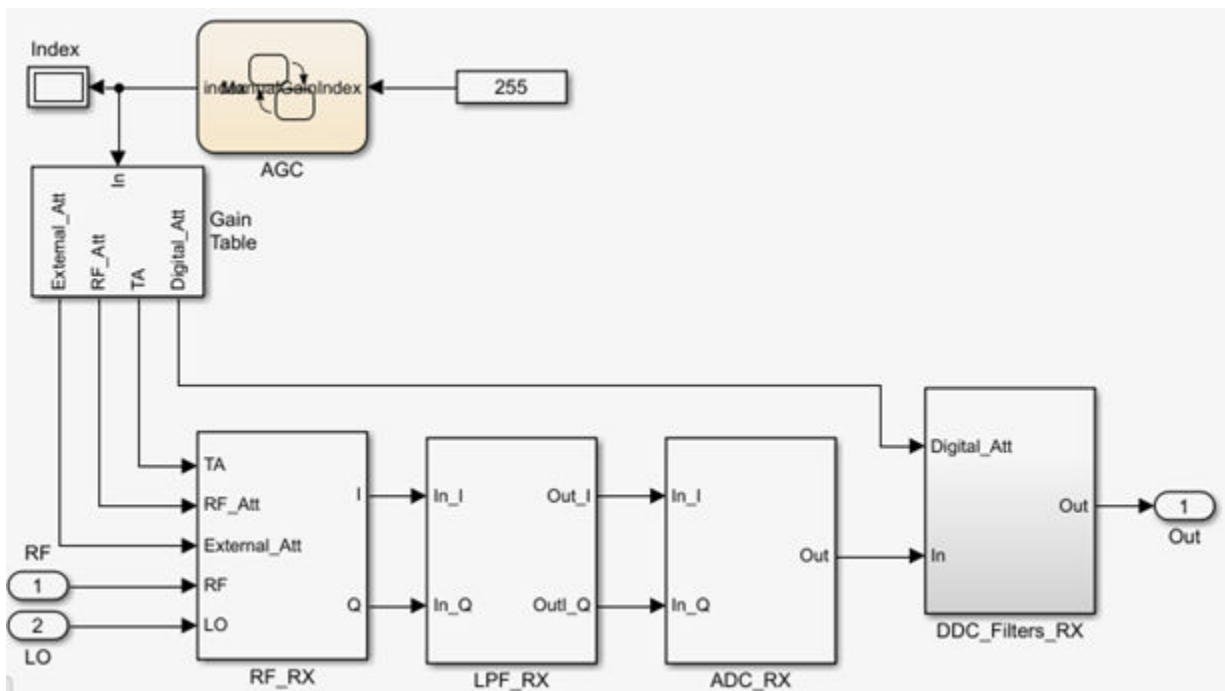
You can use the receiver model simulate the following behaviors:

- Tunable internal attenuation
- Attenuation and carrier-frequency dependent noise figure
- Attenuation and carrier-frequency dependent output-referred third-order intercept (OIP3) and output-referred second-order intercept (OIP2)
- Attenuation and carrier-frequency dependent gain imbalance (to model finite image rejection)
- Attenuation and carrier-frequency dependent local oscillator (LO) carrier leakage
- Analog filters provide continuous time signal
- ADC models a high-sampling rate third order delta-sigma modulator.

- Digital down conversion digital filters converts the highly sampled signal at the output of the ADC to a lower baseband rate.
- Received signal strength indicator measures power at two stages, at the RF receiver output and after the first half-band filter
- AGC changes the index of the gain table according to the flags of threshold crossing reported by the RSSI.
- Default gain table is read from the MATLAB file, DefaultGainTables. You can customize this file.

AD9371_ORX Analog Devices Observer Receiver

Model Description



The observer receiver model consists of three stages:

- RF receiver (RF_RX)
- Analog filters (LPF_RX)
- Analog to Digital converter (ADC_RX)
- Digital down-conversion filters (DDC_Filters_RX)
- Automatic gain control state machine (AGC) operating in manual control mode
- Programmable gain table (Gain Table)

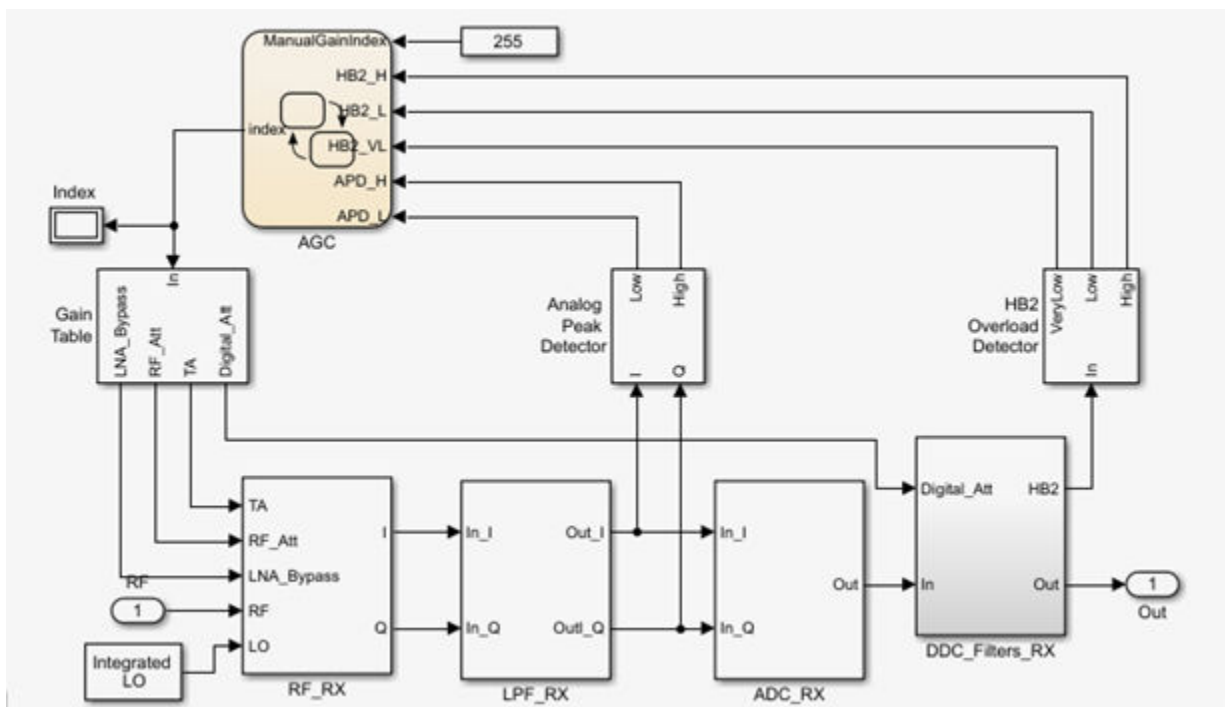
You can use the observer receiver model to simulate the following behaviors:

- Tunable internal attenuation
- Attenuation and carrier-frequency dependent noise figure

- Attenuation and carrier-frequency dependent output-referred third-order intercept (OIP3) and output-referred second-order intercept (OIP2)
- Attenuation and carrier-frequency dependent gain imbalance (to model finite image rejection)
- Attenuation and carrier-frequency dependent local oscillator (LO) carrier leakage
- Analog filters provide continuous time signal.
- ADC models a high-sampling rate third-order delta-sigma modulator.
- Digital down conversion digital filters converts the highly sampled signal at the output of the ADC to a lower baseband rate.
- Received signal strength indicator measures power at two stages, at the RF receiver output and after the first half-band filter
- Default gain table is read from the MATLAB file, DefaultGainTables. You can customize this file.

AD9371_SNF Analog Devices Sniffer Receiver

Model Description



The sniffer receiver model consists of three stages:

- RF receiver (RF_RX)
- Analog filters (LPF_RX)
- Analog to Digital converter (ADC_RX)
- Digital down-conversion filters (DDC_Filters_RX)
- Receiver signal strength indicator (RSSI): two power meters to detect the strength of the received signal at different section of the chain (Analog Peak Detector, and HB2 Overload Detector)

- Automatic gain control state machine (AGC) operating in manual control mode
- Programmable gain table (Gain Table)

You can use the AD9371 sniffer receiver model to simulate the following behaviors:

- Tunable internal attenuation
- Carrier-frequency dependent noise figure
- Attenuation and carrier-frequency dependent output-referred third-order intercept (OIP3) and output-referred second-order intercept (OIP2)
- Attenuation and carrier-frequency dependent gain imbalance (to model finite image rejection)
- Attenuation and carrier-frequency dependent local oscillator (LO) carrier leakage
- Analog filters provide continuous time signal.
- ADC models a high-sampling rate third-order delta-sigma modulator.
- Digital down conversion digital filters converts the highly sampled signal at the output of the ADC to a lower baseband rate.
- Received signal strength indicator measures power at two stages, at the RF receiver output and after the first half-band filter
- AGC changes the index of the gain table according to the flags of threshold crossing reported by the RSSI.
- Default gain table is read from the MATLAB file, `DefaultGainTables`. You can customize this file.

See Also

`simrfSupportPackages`

More About

- “AD9371 Testbenches” on page 4-15
- “AD9361 Models” on page 4-2

AD9371 Testbenches

In this section...

“AD9371_TX Analog Devices Transmitter Testbench” on page 4-16

“AD9371_RX Analog Devices Receiver Testbench” on page 4-17

“AD9371_ORX Analog Devices Observer Receiver Testbench” on page 4-18

“AD9371_SNF Analog Devices Sniffer Receiver Testbench” on page 4-18

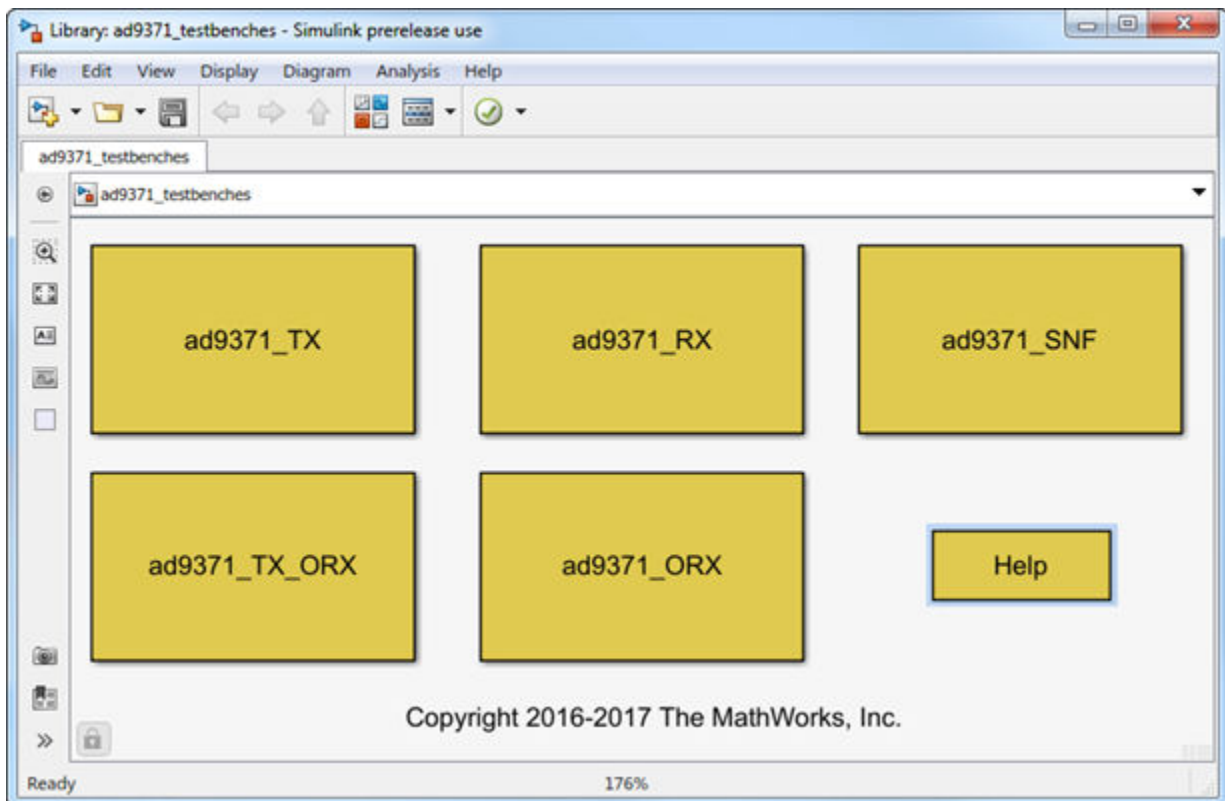
“AD9371_TX_ORX Analog Devices Transmitter-Observer Testbench” on page 4-19

You can use the AD9371 testbench models to analyze the functioning and values of Analog Devices AD9371 RF transmitter, receiver, observer, sniffer, or end-to-end designs.

Install Analog Devices RF Transceivers using `simrfSupportPackages`. You can open the testbench models using the Simulink library browser and opening RF Blockset Models for Analog Devices RF Transceivers, or by typing the following in the MATLAB command prompt:

open `ad9371_testbenches`

Click to open the AD9371 testbench model from the library:



Note You require these additional licenses to run this model:

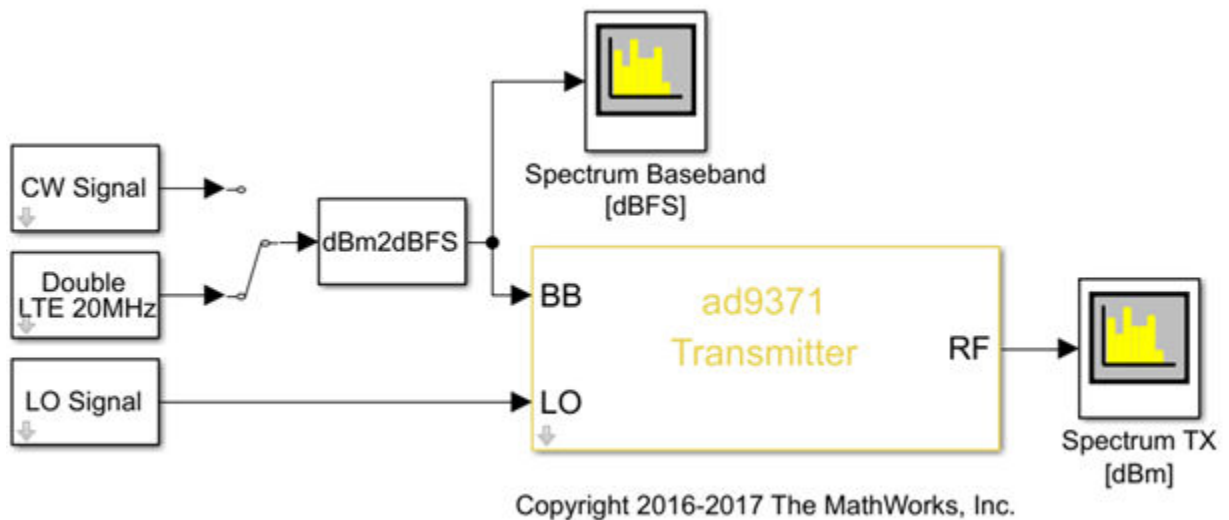
- Communications Toolbox

- Stateflow
- Fixed-Point Designer

Complete documentation on how to use the models is available with the software download.

`open(ad93xx_getdoc)`

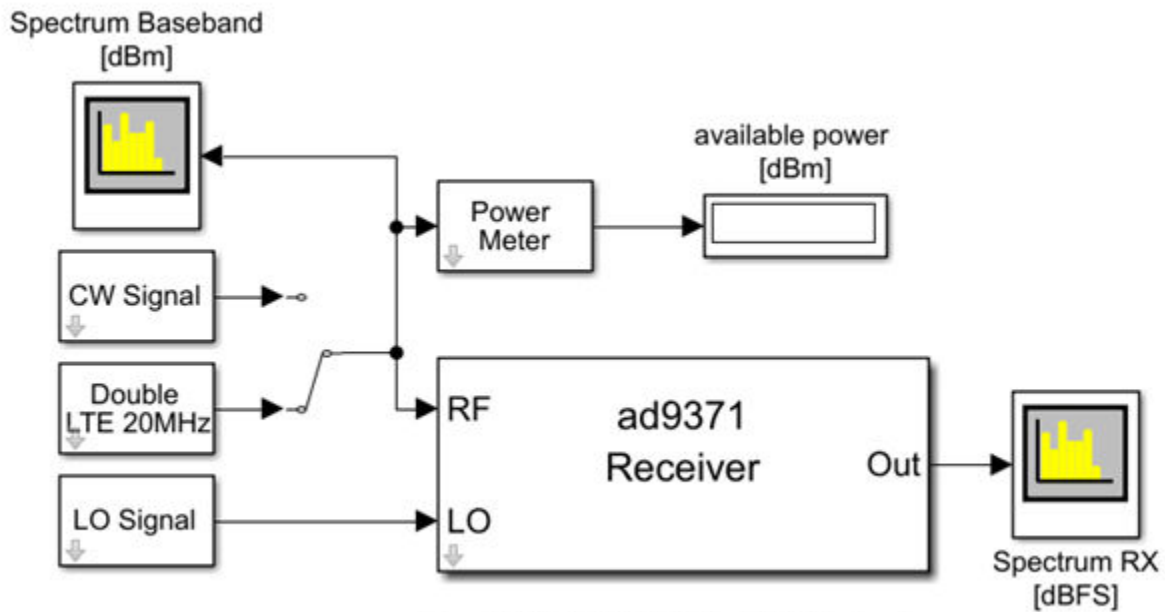
AD9371_TX Analog Devices Transmitter Testbench



The transmitter testbench consists of:

- CW and LTE Signal sources
- External local oscillator signal source
- AD9371 transmitter which is the device under test
- Spectrum analyzer and power meter for signal visualization

AD9371_RX Analog Devices Receiver Testbench

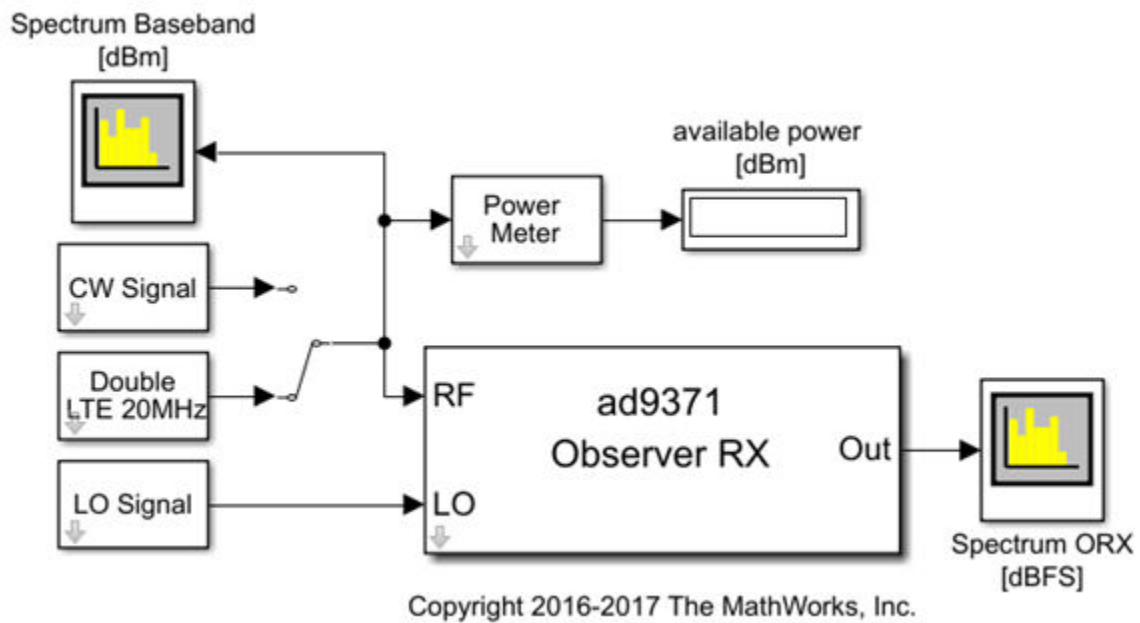


Copyright 2016-2017 The MathWorks, Inc.

The receiver testbench consists of:

- CW and LTE 20 MHz Signal sources
- External local oscillator signal source
- AD9371 receiver which is the device under test
- Spectrum analyzer and power meter for signal visualization

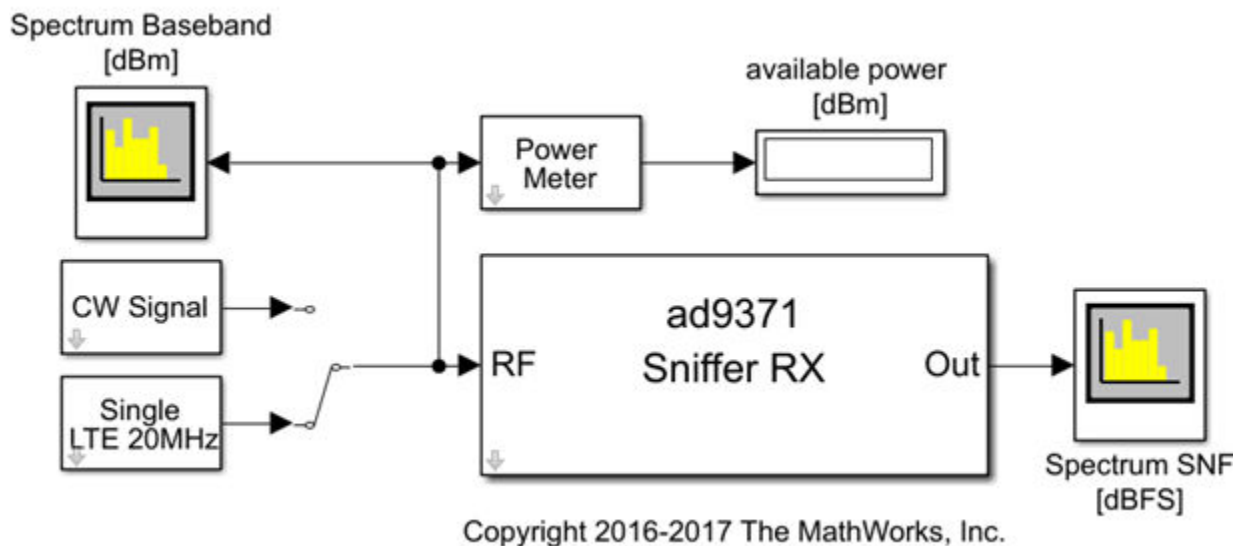
AD9371_ORX Analog Devices Observer Receiver Testbench



The observer receiver testbench consists of:

- CW and LTE 20 MHz Signal sources
- External local oscillator signal source
- AD9371 receiver which is the device under test
- Spectrum analyzer and power meter for signal visualization

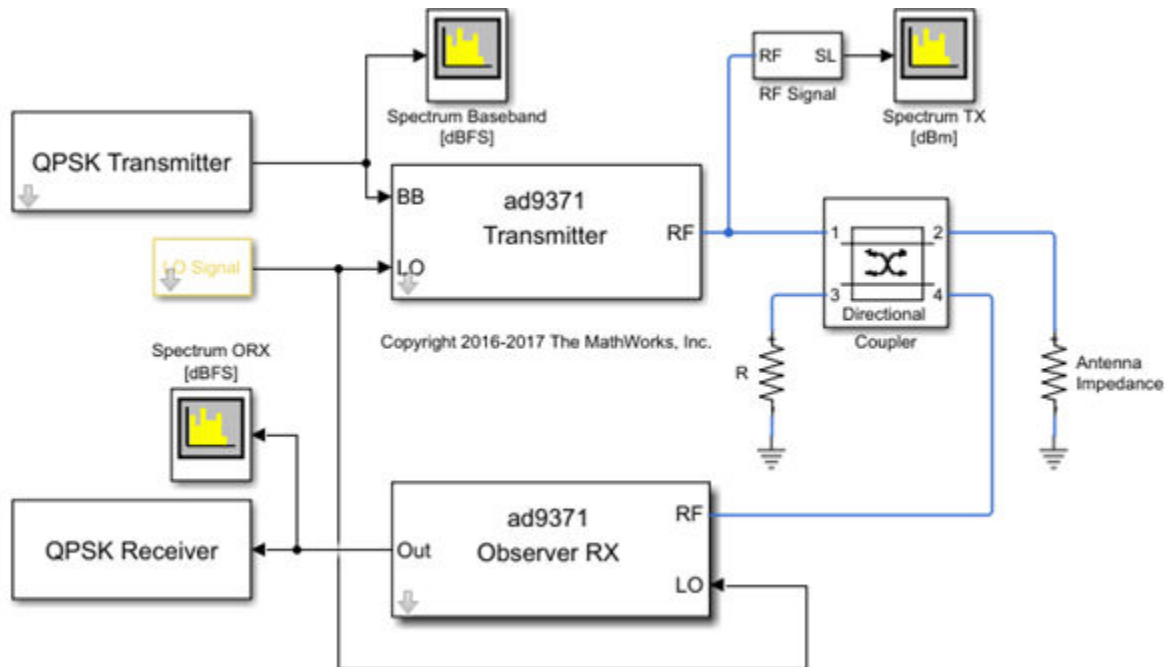
AD9371_SNF Analog Devices Sniffer Receiver Testbench



The sniffer receiver testbench consists of:

- CW and LTE 20 MHz Signal sources
- AD9371 sniffer receiver which is the device under test
- Spectrum analyzer and power meter for signal visualization

AD9371_TX_ORX Analog Devices Transmitter-Observer Testbench



The transmitter-observer testbench consists of:

- CW and LTE 20 MHz Signal sources
- External local oscillator signal source driving both transmitter and observer
- AD9371 transmitter and observer connected via directional coupler
- Spectrum analyzer and power meter for signal visualization

See Also

`simrfSupportPackages`

More About

- “AD9361 Testbenches” on page 4-6
- “AD9371 Models” on page 4-9

Equivalent Baseband

Model an RF System

- “Model RF Components” on page 5-2
- “Specify or Import Component Data” on page 5-5
- “Specify Operating Conditions” on page 5-13
- “Model Nonlinearity” on page 5-14
- “Model Noise” on page 5-16

Model RF Components

In this section...
“Add RF Blocks to a Model” on page 5-2
“Connect Model Blocks” on page 5-3

Add RF Blocks to a Model

You can include blocks from the RF Blockset Equivalent Baseband Physical and Mathematical libraries in a Simulink model. For more information on the libraries and the available RF blocks, see “RF Blockset Equivalent Baseband Libraries”.

This section contains the following topics:

- “Input Signal Requirements” on page 5-2
- “How to Add RF Blocks to a Model” on page 5-2

Input Signal Requirements

Most RF Blockset Equivalent Baseband blocks only support complex single-channel signals. The signals can be either sample-based or frame-based. The following blocks have this requirement:

- All Physical blocks
- Mathematical Amplifier and Mixer blocks

You can model the effect of these components on a multichannel signal as follows:

- 1 Use a Simulink Demux block to split the multichannel signal into single-channel signals.
- 2 Create duplicate RF models, with one model for each channel, and pass each single-channel signal into a separate model.
- 3 Use a Simulink Mux block to multiplex the signals at the output of the RF models.

How to Add RF Blocks to a Model

To add RF blocks to a Simulink model:

- 1 Type `rflib` at the MATLAB prompt to open the RF Blockset Equivalent Baseband library.
- 2 Navigate to the desired library or sublibrary.
- 3 Drag instances of RF Blockset Equivalent Baseband blocks into the model window using the mouse.

Note You can also access RF Blockset Equivalent Baseband blocks and other Simulink blocks from the Simulink Library Browser window. Open this window by typing `sLibraryBrowser` at the MATLAB prompt. Add blocks to the model by dragging them from this window and dropping them into the model window.

Connect Model Blocks

You follow the same procedure for connecting RF Blockset Equivalent Baseband blocks as for connecting Simulink blocks: you click a port and drag the mouse to draw a line to another port on a different block.

You can only connect blocks that use the same type of signal. Physical library blocks use different types of signals than Mathematical library blocks, and are represented graphically by a different port style. Therefore, you can freely connect pairs of Mathematical modeling blocks. You can also freely connect pairs of Physical modeling blocks. However, you cannot directly connect Physical blocks to Mathematical blocks. Instead, you must use the Input Port and Output Port blocks to bridge them.

For more information on the Physical and Mathematical libraries, including how to open the libraries and a description of the available blocks, see “Overview of RF Blockset Equivalent Baseband Libraries”.

This section contains the following topics:

- “Connect Mathematical Blocks” on page 5-3
- “Connect Physical Blocks” on page 5-3
- “Bridge Physical and Mathematical Blocks” on page 5-4

Connect Mathematical Blocks

The Mathematical library blocks use the same input and output ports as standard Simulink blocks. These ports show the direction of the signal at the port, as shown in the following diagram.



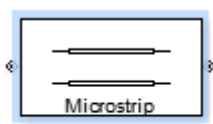
Mixers

Similar to standard Simulink blocks, you draw lines between the ports of the Mathematical modeling blocks, called *signal lines*, to represent signals that are inputs to and outputs from the mathematical functions represented by the blocks. Therefore, you can connect Simulink, DSP System Toolbox™, and RF Blockset Equivalent Baseband mathematical blocks by drawing signal lines between their ports.

You can connect a port to multiple ports by branching the signal line, or you can leave a port unconnected.

Connect Physical Blocks

The Physical library blocks have specialized *connector ports*. These ports only represent physical connections; they do not imply signal direction.



Microstrip
Transmission Line

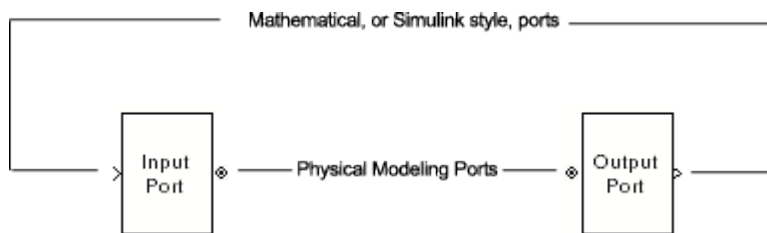
The lines you draw between the physical modeling blocks, called *connection lines*, represent physical connections among the block components. Connection lines appear as solid black when connected and as dashed red lines when either end is unconnected.

You can draw connection lines only between the connector ports of physical modeling blocks. You cannot branch these connection lines. You cannot leave connector ports unconnected.

Bridge Physical and Mathematical Blocks

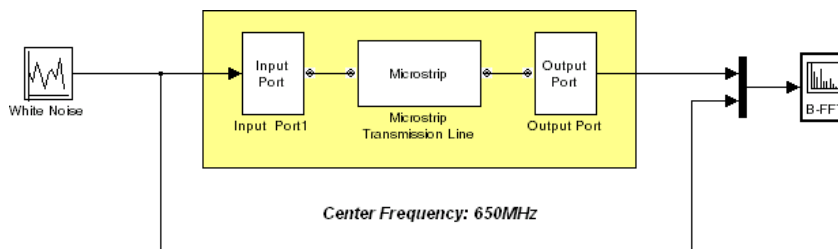
The blockset provides the Input Port and Output Port blocks to connect the physical and mathematical parts of the model. These blocks convert mathematical signals to and from the physical modeling environment.

The Input Port and Output Port blocks have one of each kind of connector port: a standard Simulink style input port and a physical modeling port. These ports are shown in the following figure:



The Input Port and Output Port blocks must bound a physical subsystem to connect it to the mathematical part of a model.

For example, a simple RF model of a microstrip transmission line might resemble the following figure.



The Microstrip Transmission Line block uses an Input Port block to get its white noise input from a Random Source block, and an Output Port block to pass its output to a Spectrum Scope block. The Random Source and Spectrum Scope blocks are from DSP System Toolbox library.

For information on how RF Blockset Equivalent Baseband software converts mathematical signals to and from the physical modeling environment, see “Convert to and from Simulink Signals” on page A-22.

See Also

Input Port | Output Port | Microstrip Transmission Line

More About

- “Simulate High Frequency Components”

Specify or Import Component Data

This example shows how to specify or import component data in an equivalent baseband system.

Specify Parameter Values

You can set block parameter values either

- Setting the parameter values directly in the block
- Using the `set_param` and `get_param` functions to set and get parameter values of the blocks, respectively.

Supported File Types for Importing Data

RF blockset™ lets you import the following types of data files:

- Industry-standard file formats — Touchstone S2P, Y2P, Z2P, and H2P formats specify the network parameters and noise information for measured and simulated data. For more information, see *Touchstone specifications*.
- Agilent® P2D file format — Specifies amplifier and mixer large-signal, power-dependent network parameters, noise data, and intermodulation tables for several operating conditions, such as temperature and bias values. The P2D file format lets you import system-level verification models of amplifiers and mixers.
- Agilent S2D file format — Specifies amplifier and mixer network parameters with gain compression, power-dependent S21 parameters, noise data, and intermodulation tables for several operating conditions. The S2D file format lets you import system-level verification models of amplifiers and mixers.
- MathWorks™ amplifier (AMP) file format — Specifies amplifier network parameters, power data, noise data, and third-order intercept point. For more information, see “AMP File Data Sections”.
- MATLAB® circuits — RF Toolbox™ circuit objects in the MATLAB workspace specify network parameters, noise data, and third-order intercept point information of circuits with different topologies. For more information, see “RF Circuit Objects”.

Import Data Files into RF Blocks

RF blockset lets you import industry-standard data files, Agilent P2D and S2D files, and MathWorks AMP files into specific blocks to simulate the behavior of measured components in the Simulink modeling environment.

This section contains the following topics:

- Blocks Used to Import Data
- How to Import Data Files

Blocks Used to Import Data

Three blocks in the Physical library accept data from a file. The following table lists the blocks and any corresponding data format that each supports.

Block	Description	Supported Format(s)
General Amplifier	Generic amplifier	Touchstone, AMP, P2D, S2D
General Mixer	Generic mixer	Touchstone, AMP, P2D, S2D
General Passive Network	Generic passive component	Touchstone

How to Import Data Files

To import a data file:

- Choose the block that best represents your component from the list of blocks that accept file data shown in **Blocks Used to Import Data** section.
- Open the Physical library, and navigate to the sublibrary that contains the block.
- Click and drag the block into your Simulink model.
- In the block dialog box, enter the name of your data file for the **Data file** parameter. The file name must include the extension. If the file is not in your MATLAB path, specify the full path to the file or use the **Browse** button to find the file.

Example: Import Touchstone Data File into RF Model

In this example, you model the frequency response of a passive component using data from a Touchstone file, `defaultbandpass.s2p`.

You use a model from one of the RF Blockset Equivalent Baseband examples to perform the following tasks:

- Import Data into General Passive Network Block
- Validate Passive Component

Import Data into General Passive Network Block

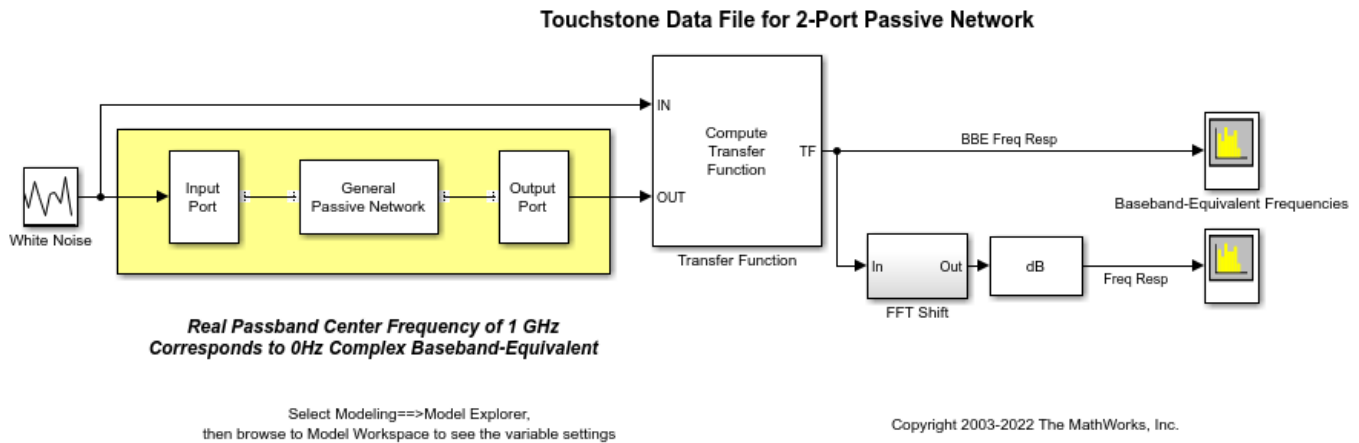
In this part of the example, you inspect the `defaultbandpass.s2p` file and import data into the RF model using the General Passive Network block.

Type the following at the MATLAB prompt to open the `defaultbandpass.s2p` file:

```
edit defaultbandpass.s2p
```

Open `sparam_filter.slx`, Touchstone Data File for 2-port Bandpass Filter.

```
open_system("sparam_filter.slx")
```



Double-click the General Passive Network block to display its parameters.

The **Data source** parameter is set to `Data file`, to specify the data file to import. The Data file parameter is set to `defaultbandpass.s2p`. The block uses this data with the other block parameters during simulation.

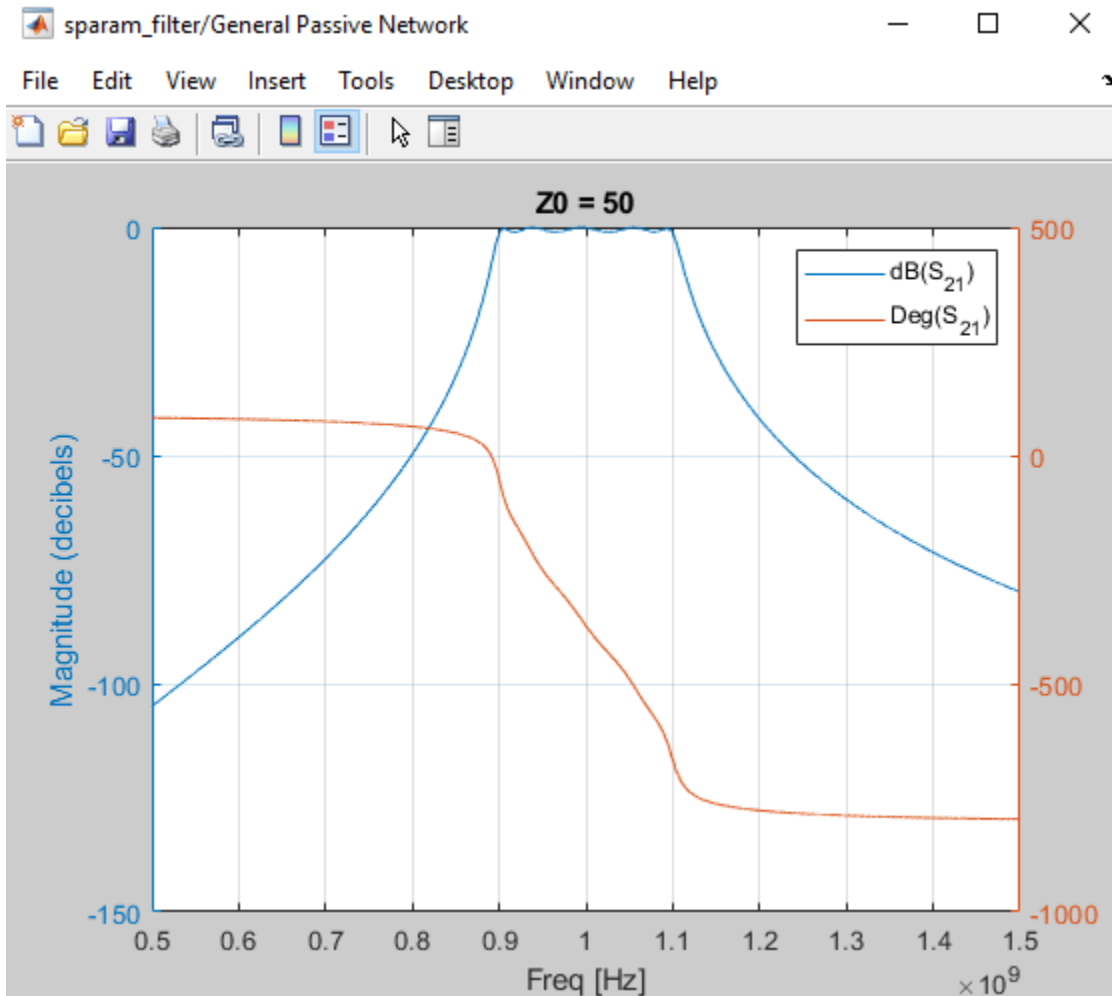
Note that When the imported file contains data that is measured at frequencies other than the modeling frequencies, use the **Interpolation method** parameter to specify how the block determines the data values at the modeling frequencies. For more information, see “Determine Modeling Frequencies” on page A-3 and “Map Network Parameters to Modeling Frequencies” on page A-4.

Validate Passive Component

In this part of the example, you plot the network parameters of the General Passive Network block to validate the data you imported in Import Data into General Passive Network Block.

- Open the General Passive Network block dialog box, and select the **Visualization** tab.
- Set the **Source of frequency data** parameter to `User-specified`.
- Set the **Frequency data (Hz)** parameter to `[0.5e9:0.1e6:1.5e9]`.
- Click **Plot**.

These actions create a plot of the magnitude and phase of S21 as a function of frequency.



Import Circuits from MATLAB Workspace

You can only connect Physical library blocks in cascade. However, the blockset works with RF Toolbox software to let you include additional circuit topologies in an RF model. To model circuit topologies that contain other types of connections, you must define a circuit in the MATLAB workspace and import it into an RF model.

To import a circuit from the MATLAB workspace:

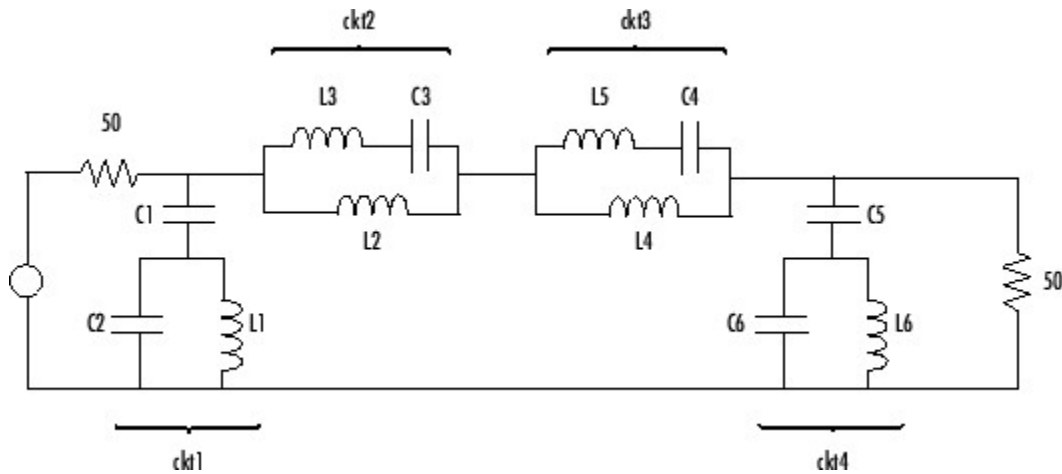
- Define the circuit object in the MATLAB workspace using the RF Toolbox functions. For more information, see "RF Circuit Objects".
- Add a General Circuit Element block to your RF model from the Black Box Elements sublibrary of the Physical library. For information on how to open this library, see "Equivalent Baseband Library".
- Enter the circuit object name in the RFACT object parameter in the General Circuit Element block dialog box.

This procedure is illustrated by example in the following section.

Example: Import Bandstop Filter into RF Model

In this example, you simulate the frequency response of a filter that you model using circuit objects from the MATLAB workspace.

The filter in this example is the 50-ohm bandstop filter shown in the following figure.



You represent the filter using four circuit objects that correspond to the four parts of the filter, ckt1, ckt2, ckt3, and ckt4 in the diagram. You use an input signal with random, complex input values that have a Gaussian distribution to stimulate the filter. The scope block displays the output signal.

This example illustrates how to perform the following tasks:

- Create Circuit Objects in MATLAB Workspace
- Build Model
- Specify and Import Component Data
- Run Simulation and Plot Results

Create Circuit Objects in MATLAB Workspace

In this part of the example, you define MATLAB variables to represent the physical properties of the filter shown in the previous figure, Bandstop Filter Diagram, and use functions from RF Toolbox software to create RF circuit objects that model the filter components.

Type the following at the MATLAB prompt to define the filter's capacitance and inductance values in the MATLAB workspace:

```
C1 = 1.734e-12;
C2 = 4.394e-12;
C3 = 7.079e-12;
C4 = 7.532e-12;
C5 = 1.734e-12;
C6 = 4.394e-12;
L1 = 25.70e-9;
L2 = 3.760e-9;
L3 = 17.97e-9;
L4 = 3.775e-9;
L5 = 17.63e-9;
L6 = 25.70e-9;
```

Type the following at the MATLAB prompt to create RF circuit objects that model the components labeled ckt1, ckt2, ckt3, and ckt4 in the circuit diagram:

```

ckt1 = ...
    rfckt.series('Ckts',{rfckt.shuntrlc('C',C1),...
    rfckt.shuntrlc('L',L1,'C',C2)});
ckt2 = ...
    rfckt.parallel('Ckts',{rfckt.seriesrlc('L',L2),...
    rfckt.seriesrlc('L',L3,'C',C3)});
ckt3 = ...
    rfckt.parallel('Ckts',{rfckt.seriesrlc('L',L4),...
    rfckt.seriesrlc('L',L5,'C',C4)});
ckt4 = ...
    rfckt.series('Ckts',{rfckt.shuntrlc('C',C5),...
    rfckt.shuntrlc('L',L6,'C',C6)});

```

For more information about the RF Toolbox objects used in this example, see the `rfckt.series`, `rfckt.parallel`, `rfckt.seriesrlc`, and `rfckt.shuntrlc` object reference pages in the RF Toolbox documentation.

Build Model

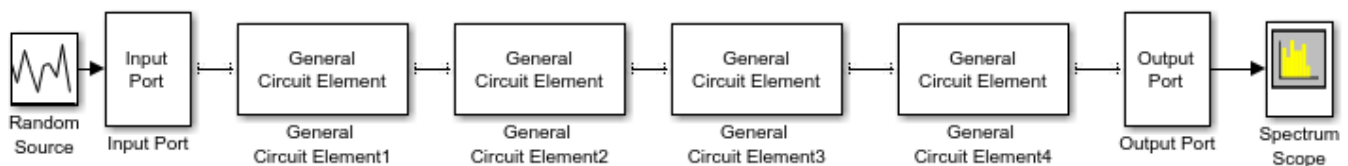
In this portion of the example, you create a Simulink model. For more information about adding and connecting components, see “Model RF Components” on page 5-2.

Create a new model. Add to the model the blocks shown in the following table. The Library column of the table specifies the hierarchical path to each block.

Block	Library	Quantity
Random Source	DSP System Toolbox > Sources	1
Input Port	RF Blockset > Equivalent Baseband > Input/Output Ports	1
General Circuit Element	RF Blockset > Equivalent Baseband > Black Box Elements	4
Output Port	RF Blockset > Equivalent Baseband > Input/Output Ports	1
Spectrum Analyzer	DSP System Toolbox > Sinks	1

Connect the blocks as shown in the following figure. Change the names of your General Circuit Element blocks to match those in the figure by double-clicking the text below the block and typing a new name. Alternatively, open the model attached in this example.

```
open_system('importing_circuits.slx')
```



Copyright 2022 The MathWorks, Inc.

Specify and Import Component Data

In this portion of the example, you specify block parameters. To open the parameter dialog box for each block, double-click the block.

In the Random Source block, set:

- **Source type** — Gaussian
- **Sample time** — 1/100e6
- **Samples per frame** — 256
- **Complexity** — Complex

Selecting these settings creates an input signal with random, complex input values that have a Gaussian distribution.

In the Input Port block, set:

- **Treat input Simulink signal as** — Source voltage
- **Finite impulse response filter length** — 256
- **Center frequency (Hz)** — 400e6
- **Sample time** — 1/100e6
- **Input processing** — Columns as channels (frame based)

and clear the **Add noise** check box. Selecting these settings defines the physical characteristics and modeling bandwidth of the filter.

Set the parameters of the General Circuit Element blocks as follows:

- In the General Circuit Element1 block, set the **RFCKT object** parameter to ckt1
- In the General Circuit Element2 block, set the **RFCKT object** parameter to ckt2
- In the General Circuit Element3 block, set the **RFCKT object** parameter to ckt3
- In the General Circuit Element4 block, set the **RFCKT object** parameter to ckt4

Selecting these settings imports the circuit objects that model the filter components into the model.

In the Output Port block, set the **Load impedance (ohms)** parameter to 50.

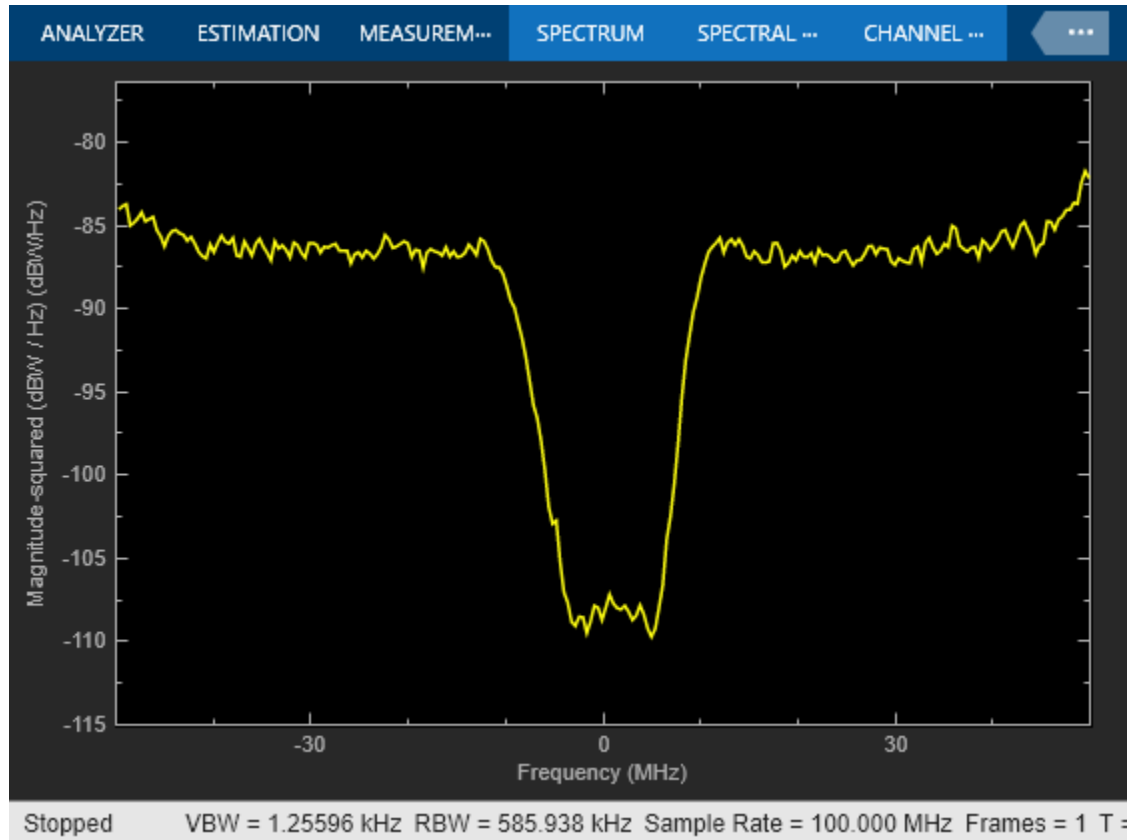
Set the Spectrum Analyzer block parameters as follows:

- In the View tab, under Spectrum Settings > Main options, set **Type** to **Power density**, **Method** to Welch, and **Window Length** to 1024 and **NFFT** to 256
- In the **View** tab, under Spectrum Settings > Trace options, set the **Averaging method** to Running and **Averages** to 100. This parameter establishes the number of spectra that the scope averages to produce the displayed signal. You use a value of 100 because the input signal is random and you want to display the average filter response over a large number of input values.
- In the **View** tab, under Configuration Properties, set the **Y-limit (Minimum)** parameter to -115 and the **Y-limit** parameter (Maximum) to -76.4. These values set the range of x- and y-values on the display such that the entire signal is visible when you run the simulation.
- Set the **Y label** parameter to Magnitude-squared.

Run Simulation and Plot the Results

Click Run in the model window to start the simulation.

```
sim("importing_circuits.slx")
pause(5)
```



The Spectrum Scope block displays the frequency response at the shifted (baseband-equivalent) frequencies, not at the selected passband frequencies. You can relabel the x-axis of the Spectrum Scope window to display the passband signal by entering the **Center frequency** parameter value of $400\text{e}6$ (from the Input Port block) for the **Frequency display offset (Hz)** parameter in the **Axis Properties** tab of the Spectrum Scope block. For more information on complex-baseband modeling, see “Create Complex Baseband-Equivalent Model” on page A-9.

References

[1] Geffe, P.R., “Novel designs for elliptic bandstop filters,” *RF Design*, February 1999.

See Also

Input Port | Output Port | Configuration

More About

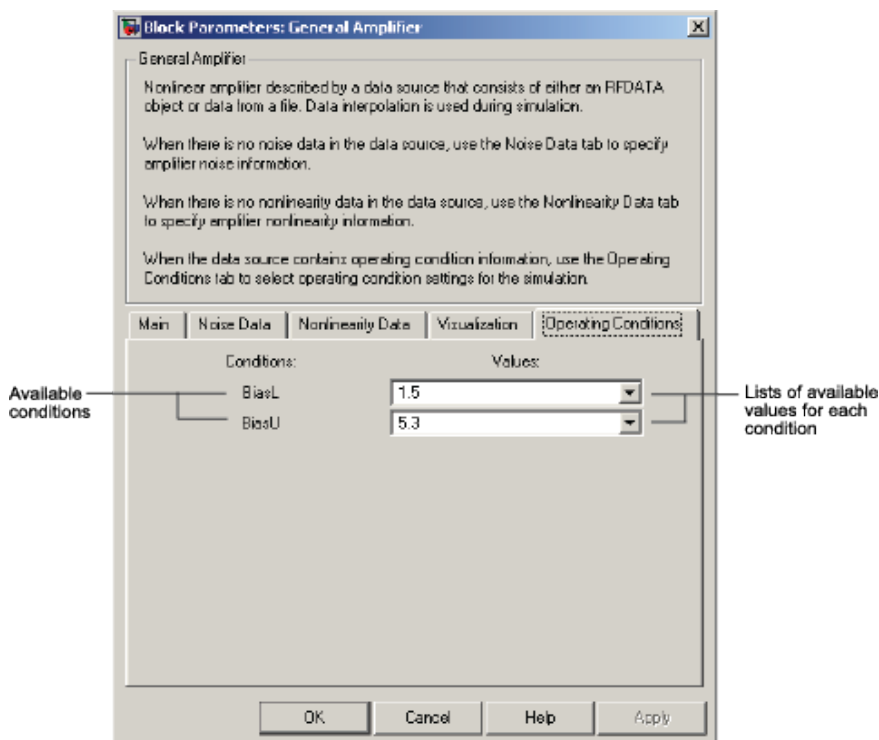
- “Create Complex Baseband-Equivalent Model” on page A-9

Specify Operating Conditions

Agilent® P2D and S2D files contain simulation results at one or more operating conditions. Operating conditions define the independent parameter settings that are used when creating the file data. The specified conditions differ from file to file.

When you import component data from a .p2d or .s2d file into a General Amplifier or General Mixer block, the block contains parameter values for several operating conditions. The available conditions depend on the data in the file. By default, the blockset defines the object behavior using the property values that correspond to the operating conditions that appear first in the file. To use other property values, you must select a different operating condition in the block dialog box.

If the block contains data at multiple operating conditions, the **Operating Conditions** tab contains two columns. The **Conditions** column shows the available conditions, and the **Values** column contains a drop-down list of the available values for the corresponding condition.



Block Dialog Box Showing Operating Conditions

To specify the operating condition values for a simulation:

- 1 Double-click the block to open the block dialog box.
- 2 Select the **Operating Conditions** tab.
- 3 In the **Conditions** column, find the condition to specify. Select the corresponding pull-down list in the **Values** column, and choose the desired operating condition value.

Repeat the preceding step as needed to specify the desired operating condition values.

Model Nonlinearity

In this section...

“Amplifier and Mixer Nonlinearity Specifications” on page 5-14

“Add Nonlinearity to Your System” on page 5-14

Amplifier and Mixer Nonlinearity Specifications

You define nonlinearity for the physical amplifier and mixer blocks at one or more frequency points through one of the following specifications:

- Power data, consisting of output power as a function of input power, imported into the block.
- Third-order intercept data, with or without power parameters, in the block dialog box. The available power parameters are gain compression power (defined as the ratio of output power to input power at small input power) and output saturation power.

The following table summarizes the nonlinearity specification options for each type of physical amplifier and mixer block.

Block	Nonlinearity Specification
General Amplifier	You can choose either of the following specifications: Power data (using a P2D, S2D, or AMP data file) or Third-order intercept data or one or more power parameters, in the block dialog box.
S-Parameters Amplifier	Third-order intercept data or one or more power parameters, in the block dialog box.
Y-Parameters Amplifier	
Z-Parameters Amplifier	
General Mixer	You can choose either of the following specifications: Power data (using a P2D, S2D, or AMP data file) or Third-order intercept data or one or more power parameters, in the block dialog box.
S-Parameters Mixer	Third-order intercept data or one or more power parameters, in the block dialog box.
Y-Parameters Mixer	
Z-Parameters Mixer	

Add Nonlinearity to Your System

To simulate the nonlinearity of an amplifier or mixer, you must specify or import nonlinearity data at one or more frequency points into the block.

The method you use to add nonlinearity data to a block depends on whether you specify the data manually or import the data into a block.

The following table provides instructions for adding nonlinearity data.

Nonlinearity Specification	Instructions
IP3	<p>In the Nonlinearity Data tab of the block dialog box:</p> <ul style="list-style-type: none"> • Set the IP3 type parameter to IIP3 or OIP3. • Enter input third-order intercept values at one or more frequency points in the IP3 (dBm) parameter. • Enter corresponding frequency values in the Frequency (Hz) parameter.
Power parameters	<p>Enter the gain compression power in the 1 dB gain compression power (dBm) parameter or the saturation power in the Output saturation power (dBm) parameter.</p> <p>If you choose a scalar value for the Frequency (Hz) parameter, then you must also use scalar values for the power parameters.</p> <p>If you choose a vector value for the Frequency (Hz) parameter, then you can use either scalar or vector values for the power parameters.</p>
Power data (from a file)	<p>Import file data that includes power information into the Data file or RFCKT object parameter of the General Amplifier or General Mixer block.</p>

Note If you import file data with no power information into a General Amplifier or General Mixer block, the **Nonlinearity Data** tab lets you add nonlinearity data manually in the block dialog box.

For information on how the blockset simulates nonlinearity data of an amplifier or mixer, see the block reference page.

See Also

More About

- “User-Defined Nonlinear Amplifier Model” on page 8-138

Model Noise

In this section...

“Amplifier and Mixer Noise Specifications” on page 5-16

“Add Noise to Your System” on page 5-17

“Plot Noise” on page 5-20

Amplifier and Mixer Noise Specifications

You only need to specify noise information for the physical amplifier and mixer blocks that generate noise other than resistor noise. For the other blocks, the blockset calculates the noise automatically based on the resistor values.

You define noise for the physical amplifier and mixer blocks through one of the following specifications:

- Spot noise data in the data source.
- Spot noise data in the block dialog box.
- Spot noise data (`rfddata.noise`) object in the block dialog box.
- Frequency-independent noise figure, noise factor, or noise temperature value in the block dialog box.
- Frequency-dependent noise figure data (`rfddata.nf`) object in the block dialog box.

The following table summarizes the noise specification options for each type of physical amplifier and mixer block.

Block	Noise Specification
General Amplifier	Spot noise data (using a Touchstone, P2D, S2D, or AMP data file) OR Spot noise data, noise figure value, noise factor value, noise temperature value, <code>rfddata.noise</code> , or <code>rfddata.nf</code> object in the block dialog box
S-Parameters Amplifier Y-Parameters Amplifier Z-Parameters Amplifier	Spot noise data, noise figure value, noise factor value, noise temperature value, <code>rfddata.noise</code> , or <code>rfddata.nf</code> object in the block dialog box
General Mixer	Spot noise data (using a Touchstone, P2D, S2D, or AMP data file) OR Spot noise data, noise figure value, noise factor value, noise temperature value, <code>rfddata.noise</code> , or <code>rfddata.nf</code> object in the block dialog box

Block	Noise Specification
S-Parameters Mixer	Spot noise data, noise figure value, noise factor value, noise temperature value, <code>rfddata.noise</code> , or <code>rfddata.nf</code> object in the block dialog box
Y-Parameters Mixer	
Z-Parameters Mixer	

Add Noise to Your System

To simulate the noise of a physical subsystem, you perform the following tasks:

- “Specify or Import Noise Data” on page 5-17
- “Add Noise to the Simulation” on page 5-18

Specify or Import Noise Data

The method you use to add noise data to a block depends on whether you are specifying noise data manually or importing spot-noise data.

The following table provides instructions for adding noise data.

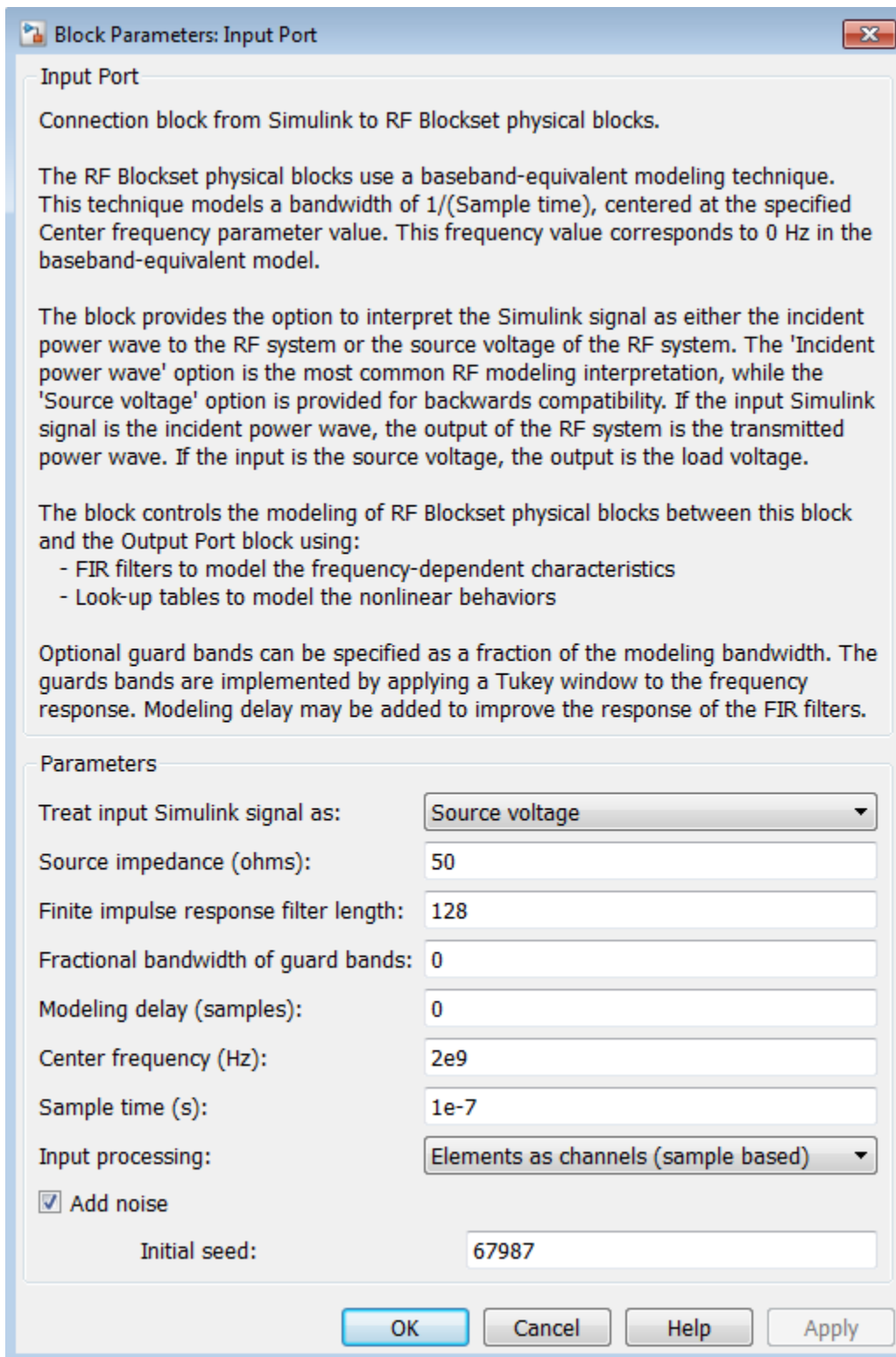
Noise Specification	Instructions
Frequency-independent noise figure	In the Noise Data tab of the block dialog box, set the Noise type parameter to Noise figure , and enter the noise figure value in the Noise figure (dB) parameter.
Frequency-dependent noise figure	In the Noise Data tab of the block dialog box, set the Noise type parameter to Noise figure , and enter the name of the <code>rfddata.nf</code> object in the Noise figure (dB) parameter.
Noise factor	In the Noise Data tab of the block dialog box, set the Noise type parameter to Noise factor , and enter the noise factor value in the Noise factor parameter.
Noise temperature	In the Noise Data tab of the block dialog box, set the Noise type parameter to Noise temperature , and enter the noise temperature value in the Noise temperature (K) parameter.
Spot noise data (in a block dialog box)	In the Noise Data tab of the block dialog box, set the Noise type parameter to Spot noise data . Enter the spot noise information in the Minimum noise figure (dB) , Optimal reflection coefficient , and Equivalent normalized noise resistance parameters.
Spot noise data (from a data object)	In the Noise Data tab of the block dialog box, set the Noise type parameter to Noise figure and enter the name of the <code>rfddata.noise</code> object in the Noise figure (dB) parameter.

Noise Specification	Instructions
Spot noise data (from a file)	Import file data that includes noise information into the Data file or RFCKT object parameter of the General Amplifier or General Mixer block.

Note If you import file data with no noise information into a General Amplifier or General Mixer block, the **Noise Data** tab lets you add noise data manually in the block dialog box.

Add Noise to the Simulation

To include noise in the simulation, you must select the **Add noise** check box on the Input Port block dialog box. This check box is selected by default.



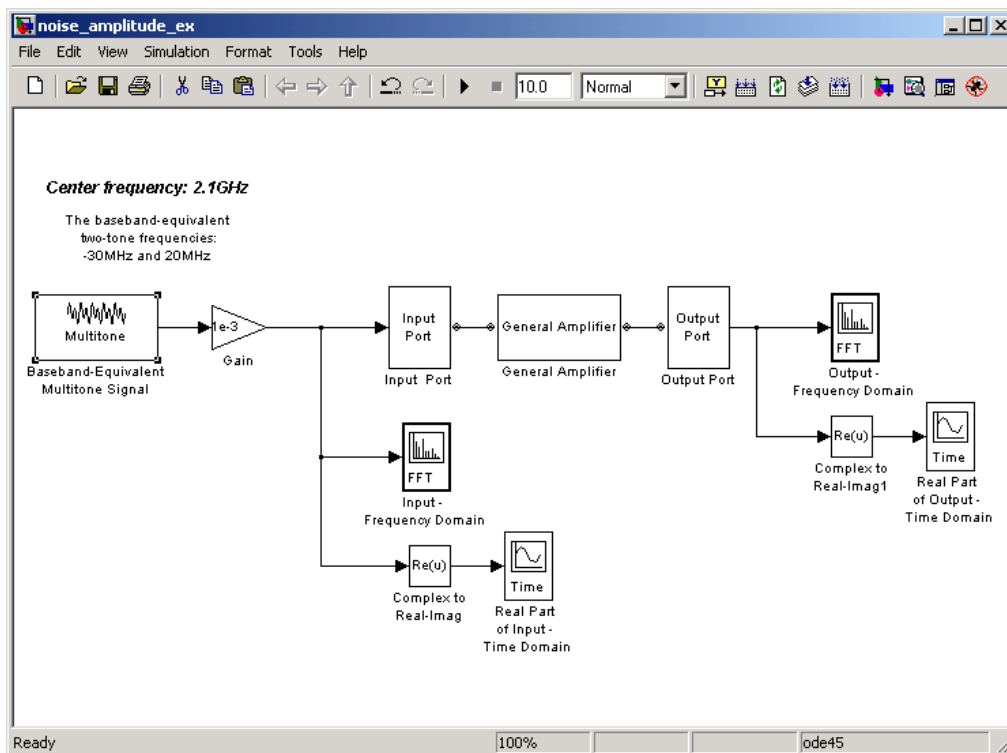
For information on how the blockset simulates noise, see “Model Noise in an RF System” on page A-5.

Plot Noise

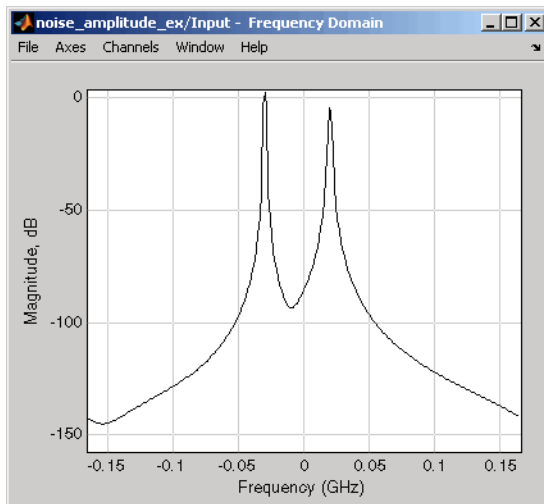
RF Blockset Equivalent Baseband software models communications systems. The noise in these systems has a very small amplitude, typically from $1e-6$ to $1e-12$ Watts. In contrast, the default signal power of a Communications Toolbox modulator block is 1 Watt at a nominal 1 ohm. Therefore, the signal-to-noise ratio in an RF system simulation is large, making it difficult to view the noise that the RF system adds to your signal.

To display the noise on a plot, you might need to attenuate the signal amplitude to a value within a couple orders of magnitude of the noise.

For example, suppose you have the following model that contains a multitone test signal source.

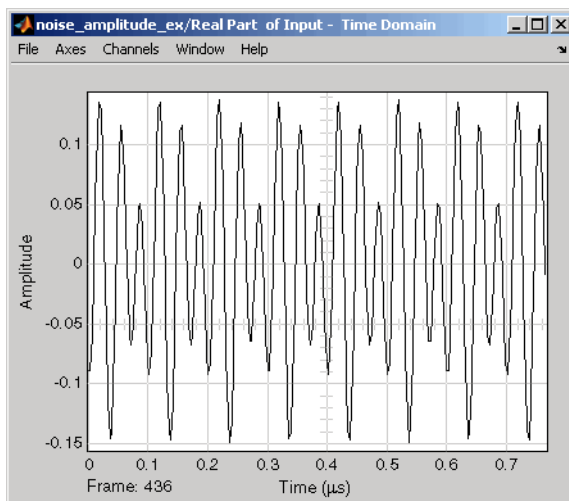


When you simulate this model, Simulink brings up several windows showing the input and output for the physical subsystem. The Input - Frequency Domain window shown in the following figure displays the input signal in the frequency domain.



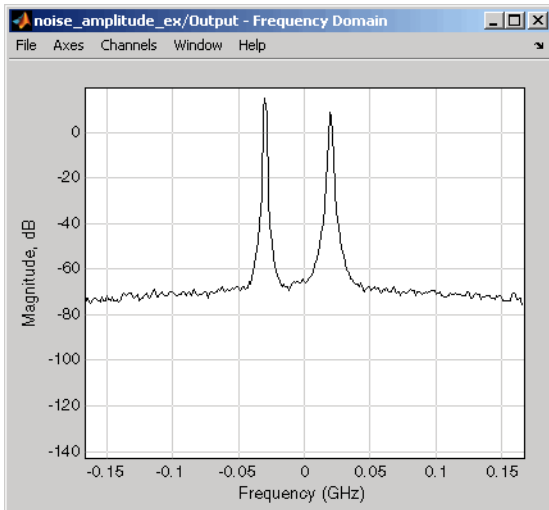
Input Signal Spectrum

The Real Part of Input - Time Domain window displays the real part of the complex-valued input signal in the time domain.



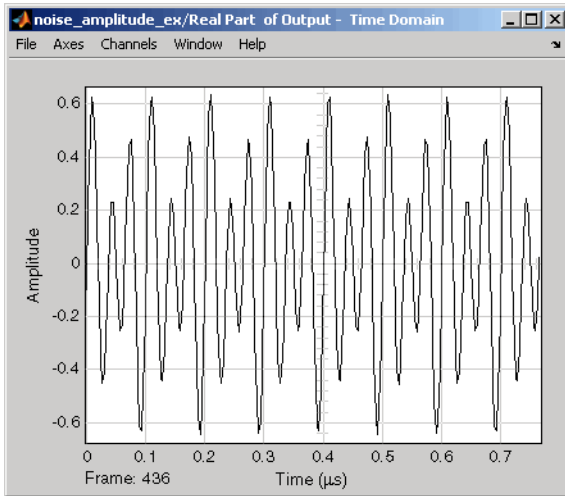
Real Part of Input Signal

In the model, the physical subsystem adds noise to the input signal. The Output - Frequency Domain window shows the noisy output signal in the frequency domain.



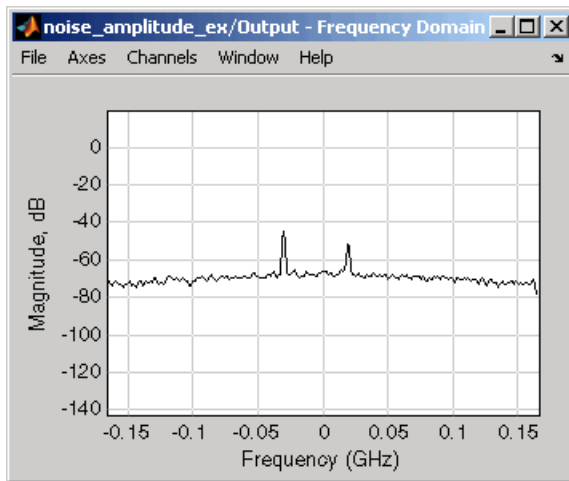
Output Signal Spectrum

The amplitude of the signal is large compared to the amplitude of the noise, so the noise is not visible in the Real Part of Output - Time Domain window that shows the real part of the time-domain output signal. Therefore, you must attenuate the amplitude of the input signal to display the noise of the time-domain output signal.



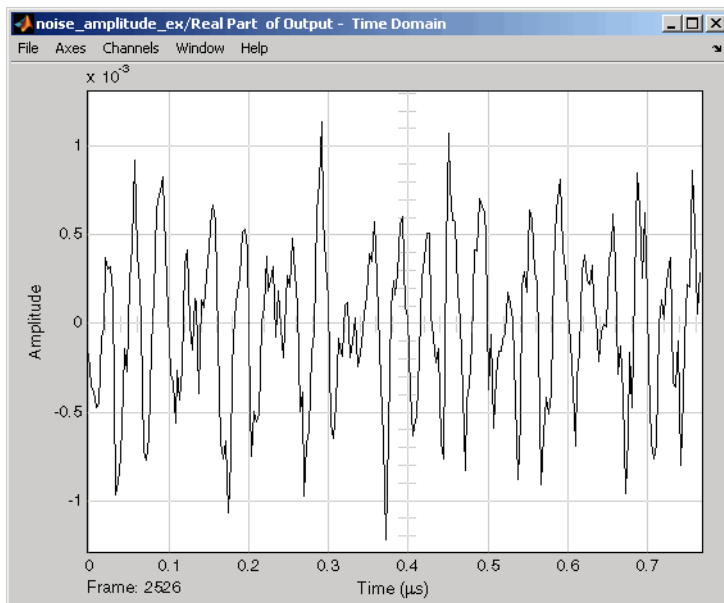
Real Part of Output Signal

Attenuate the amplitude of the input signal by setting the **Gain** parameter to $1e-3$. This is equivalent to attenuating the input signal by 60 dB. When you run the model again, the two signal peaks are not as pronounced in the Output - Frequency Domain window.



Output Signal Spectrum for Attenuated Input

You can now view the noise that the RF system adds to your signal in the Real Part of Output - Time Domain window.



Real Part of Output Signal Showing Noise

See Also

[Input Port](#) | [Output Port](#) | [General Amplifier](#)

More About

- "Model Nonlinearity" on page 5-14
- "Noise in RF Systems" on page 2-7

Plot Model Data

- “Create Plots” on page 6-2
- “Update Plots” on page 6-20
- “Modify Plots” on page 6-21
- “Create and Modify Subsystem Plots” on page 6-23

Create Plots

In this section...

“Available Data for Plotting” on page 6-2

“Validate Individual Blocks and Subsystems” on page 6-2

“Types of Plots” on page 6-3

“Plot Formats” on page 6-3

“How to Create a Plot” on page 6-10

“Example — Plot Component Data on a Z Smith Chart” on page 6-16

Available Data for Plotting

RF Blockset Equivalent Baseband software lets you validate the behavior of individual RF components and physical subsystems in your model by plotting the following data:

- Large- and small-signal S-parameters
- Noise figure, noise factor and noise temperature
- Output third-order intercept point
- Power data
- Phase noise
- Voltage standing-wave ratio
- Transfer function
- Group delay
- Reflection coefficients

Note When you plot information about a physical block, the blockset plots the actual frequency response of the block, as specified in the block dialog box. The blockset does not plot the frequency response of the complex-baseband model that it uses to simulate the block, in which the frequency response is centered at zero.

Validate Individual Blocks and Subsystems

You can plot model data for an individual physical block or for a physical subsystem. A *subsystem* is a collection of one or more physical blocks bracketed by an Input Port block and an Output Port block. To understand the behavior of specific subsystems, plot the data of the corresponding Output Port block after you run a simulation.

To validate the behavior of individual RF components in the model, plot the data of the corresponding physical blocks. You can plot data for individual blocks from each of these components either before or after you run a simulation.

You create a plot by selecting options in the block dialog box, as shown in “Create and Modify Subsystem Plots” on page 6-23. To learn about the available plots, see “Types of Plots” on page 6-3. For more information about creating plots, see “How to Create a Plot” on page 6-10.

Types of Plots

RF Blockset Equivalent Baseband software provides a variety of plots for analyzing the behavior of RF components and subsystems. The following table summarizes the available plots and charts and describes each one.

Plot Type	Plot Contents
X-Y Plane (Rectangular) Plot	Parameters as a function of frequency, input power, or operating condition, such as <ul style="list-style-type: none"> • S-parameters • Noise figure (NF), Noise factor (NFactor), and Noise Temperature (NTemp) • Voltage standing-wave ratio (VSWR) • Output third-order intercept point (OIP3) • Input and output reflection coefficients (GammaIn and GammaOut)
Link Budget Plot (3-D)	Parameters as a function of frequency for each component in a physical subsystem <i>where</i> The curve for a given component represents the cumulative contribution of each RF component up to and including the parameter value of that component. For more information, see “Link Budget” on page 6-8.
Polar Plane Plot	Magnitude and phase of parameters as a function of frequency or operating condition, such as <ul style="list-style-type: none"> • S-parameters • Input and output reflection coefficients (GammaIn and GammaOut)
Smith® Chart	Real and imaginary parts of S-parameters as a function of frequency or operating condition, used for analyzing the reflections caused by impedance mismatch.
Composite Plot	Multiple plots and charts in one figure.

To learn how to create these plots, see “How to Create a Plot” on page 6-10.

Plot Formats

When you create a plot from a block dialog box, you must specify the format of the data for both the x- and y-axes.

Source of frequency data: Same as the s-parameters frequency

Frequency data (Hz): [1.9e9:1.0e8:2.2e9]

Reference impedance (ohms): 50

Plot type: XY plane

Y parameter1: S11 Y format1: Magnitude (decibels)

Y parameter2: Y format2:

X parameter: Freq X format: Hz

Y scale: Linear X scale: Linear

Plot

Plot format of first dependent variable

Plot format of second dependent variable

Plot format of independent variable

These plot options define how RF Blockset Equivalent Baseband software displays the data on the plot.

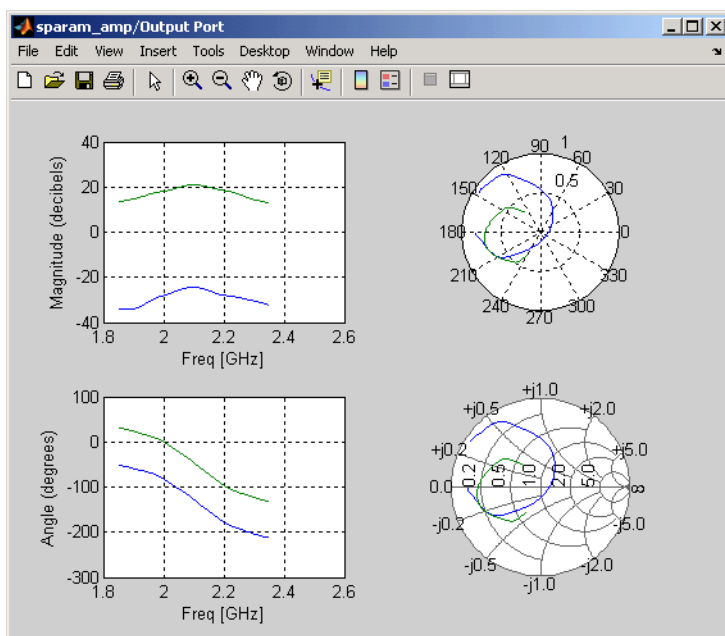
The available formats vary with the data you select to plot. The data you can plot depends on the plot type you select. The plot formats determine whether the blockset converts the data to a new set of units, or performs a calculation on the data. For example, setting the format to Real tells the blockset to compute and plot the real part of the parameter.

The following topics describe the available parameters and formats for each plot type:

- “Composite Data” on page 6-4
- “X-Y Plane” on page 6-6
- “Link Budget” on page 6-8
- “Polar Plane Plots and Smith Charts” on page 6-9

Composite Data

The composite data plot automatically generates four separate plots in one figure window, showing the frequency dependence of several parameters. The following figure shows an example of such a plot.



Example — Composite Data Plot

Note For composite data plots, you do not need to specify the parameters or the formats—they are set automatically.

The combination of plots differs based on the type of block and the specified block data. The following table describes the contents of the composite data plot for each specification. The Plot Contents column lists the types of plots as they appear on the composite plot, counterclockwise and starting in the upper-left corner. The blockset plots all data as a function of frequency.

Block	Specified Data	Plot Contents
General Amplifier or General Mixer	Network parameters OR Network parameters and noise	<ul style="list-style-type: none"> • X-Y plot, magnitude of S_{12} and S_{21} in decibels • X-Y plot, phase of S_{12} and S_{21} in degrees • Z Smith Chart, real and imaginary parts of S_{11} and S_{22} • Polar plot, magnitude and phase of S_{11} and S_{22}
	Network parameters and power OR Network parameters, noise, and power	<ul style="list-style-type: none"> • X-Y plot, magnitude of S_{12} and S_{21} in decibels • X-Y plot, output power (P_{out}) in dBm (decibels referenced to one milliwatt) • Z Smith Chart, real and imaginary parts of S_{11} and S_{22} • Polar plot, magnitude and phase of S_{11} and S_{22}

Block	Specified Data	Plot Contents
Other Physical block	<p>Network parameters</p> <p>OR</p> <p>Network parameters and noise (S-, Y-, and Z-Parameters Amplifiers and Mixers only)</p> <hr/> <p>Note Only the General Amplifier and General Mixer blocks accept power data.</p>	<ul style="list-style-type: none"> • X-Y plot, magnitude of S_{12} and S_{21} in decibels • X-Y plot, phase of S_{12} and S_{21} in degrees • Z Smith Chart, real and imaginary parts of S_{11} and S_{22} • Polar plot, magnitude and phase of S_{11} and S_{22}

X-Y Plane

You can plot any parameters that are relevant to your block on an X-Y plane plot. For this type of plot, you specify data for both the x- and y-axes. If you specify two Y parameters, and you specify different formats for the two Y parameters, the blockset plots the second Y parameter on the right y-axis.

The following table summarizes the available Y parameters and formats. The parameters and formats are the same for both the left and right y-axes.

Note LS11, LS12, LS21, and LS22 are large-signal S-parameters. You can plot these parameters as a function of input power or as a function of frequency.

Y Parameter	Y Format
S11, S12, S21, S22	Magnitude (decibels) Magnitude (linear)
LS11, LS12, LS21, LS22 (General Amplifier and General Mixer blocks with multiple operating conditions only)	Angle (degrees) Angle (radians) Real Imaginary
NF	Magnitude (decibels)
NFactor	None This format tells the blockset to plot the noise factor as it is specified to or calculated by the block.
NTemp	Kelvin
OIP3	dBm dBW W mW
VSWRIn, VSWRout	Magnitude (decibels) None This format tells the blockset to plot the voltage standing-wave ratio as it is specified to or calculated by the block.

Y Parameter	Y Format
Pout (General Amplifier and General Mixer blocks with power data only)	dBm dBW W mW
Phase (General Amplifier and General Mixer blocks with power data only)	Angle (degrees) Angle (radians)
AM/AM (General Amplifier and General Mixer blocks with power data only)	Magnitude (decibels) None This format tells the blockset to plot the AM/AM conversion as it is specified to or calculated by the block.
AM/PM (General Amplifier and General Mixer blocks with power data only)	Angle (degrees) Angle (radians)
PhaseNoise (Mixer blocks only)	dBc/Hz
FMIN (Amplifier and Mixer blocks with spot noise data only)	Magnitude (decibels) None This format tells the blockset to plot the minimum noise figure as it is specified to or calculated by the block.
GammaIn, GammaOut (Output Port block only)	Magnitude (decibels) Magnitude (linear) Angle (degrees) Angle (radians) Real Imaginary
GAMMAOPT (Amplifier and Mixer blocks with spot noise data only)	Magnitude (decibels) Magnitude (linear) Angle (degrees) Angle (radians) Real Imaginary
RN (Amplifier and Mixer blocks with spot noise data only)	None This format tells the blockset to plot the noise resistance as it is specified to or calculated by the block.

The available X parameters depend on the Y parameters you select. The following table summarizes the available X parameters for each of the Y parameters in the preceding table.

Y Parameter	X Parameter
Pout, Phase, LS11, LS12, LS21, LS22	Pin Freq
S11, S12, S21, S22, NF, OIP3, VSWRIn, VSWRout, GAMMAIn, GAMMAOut, FMIN, GAMMAOPT, RN	Freq
AM/AM, AM/PM	AM

The following table shows the X formats that are available for the X parameters listed in the preceding table.

X Parameter	X Format
Pin	dBm dBW W mW
Freq	THz GHz MHz KHz Hz Auto (xformat is chosen to provide the best scaling for the given xparameter values.)
AM	Magnitude (dB) Magnitude (linear)

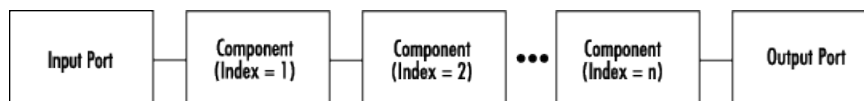
When you import block data from a .p2d or .s2d file, you can also plot Y parameters as a function of any operating condition from the file that has numeric values, such as bias. You can specify an operating condition as the X parameter only when validating individual blocks, and the format is always None. This format tells the blockset to plot the operating condition values as they are specified in the file.

Link Budget

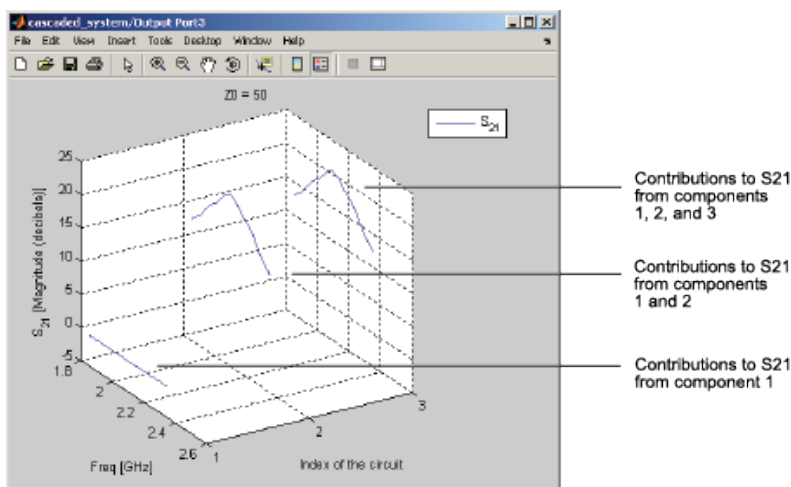
You use the Link budget plot to understand the individual contribution of each block to a plotted Y parameter value in a physical subsystem with multiple components between the Input Port and the Output Port blocks.

The link budget plot is a three-dimensional plot that shows one or more curves of parameter values as a function of frequency, ordered by the subsystem circuit index.

The following figure shows how the circuit index is assigned to a component in a physical subsystem based on its sequential position in the subsystem.



A curve on the link budget plot for each circuit index represents the contributions to the parameter value of the RF components up to that index. The following figure shows an example of a link budget plot.



Example — Link Budget Plot

The following table summarizes the Y parameters and formats that are available for a link budget plot.

Y Parameter	Y Format
S11, S12, S21, S22	Magnitude (decibels) Magnitude (linear) Angle (degrees) Real Imaginary
OIP3	dBm dBW W mW
NF	Magnitude (decibels) Magnitude (linear)
NFactor	None This format tells the blockset to plot the noise factor as it is specified to the block.
NTemp	Kelvin

If you specify two Y parameters, the blockset puts both parameters in a single plot. The Y parameters must have the same formats.

For a link budget plot, the X parameter is always Freq. The format of the X parameter specifies the units of the x-axis.

Polar Plane Plots and Smith Charts

You can use RF Blockset Equivalent Baseband software to generate Polar plots and Smith Charts. When you select these plot types, you do not need to specify the format of any Y parameters—the formats are set automatically. If you specify two Y parameters, the blockset puts both parameters in a single plot.

The following table describes the Polar plot and Smith Chart options. It also lists the available Y parameters.

Plot Type	Y Parameter
Polar plane	S11, S12, S21, S22 LS11, LS12, LS21, LS22 (General Amplifier and General Mixer blocks with data from a P2D file only) GammaIn, GammaOut (Output Port block only)
Z Smith chart	S11, S22 LS11, LS22 (General Amplifier and General Mixer blocks with data from a P2D file only) GammaIn, GammaOut (Output Port block only)
Y Smith chart	S11, S22 LS11, LS22 (General Amplifier and General Mixer blocks with data from a P2D file only) GammaIn, GammaOut (Output Port block only)
ZY Smith chart	S11, S22 LS11, LS22 (General Amplifier and General Mixer blocks with data from a P2D file only) GammaIn, GammaOut (Output Port block only)

By default, the X parameter is Freq. The format of the X parameter specifies the units of the x-axis. When you import block data from a .p2d or .s2d file, you can also plot Y parameters as a function of any operating condition from the file that has numeric values, such as bias. You can specify an operating condition as the X parameter only when validating individual blocks, and the format is always None.

How to Create a Plot


- 1 Double-click the block to open the block dialog box, and select the **Visualization** tab. The following figure shows the contents of the tab.

Source of frequency data:	Extracted from data source	
Frequency data (Hz):	[1e9:1e8:2.9e9]	
Source of input power data:	Extracted from data source	
Input power data (dBm):	[0:19]	
Reference impedance (ohms):	50	
Plot type:	X:Y plane	
Y parameter1:	S11	Y format1: Magnitude (decibels)
Y parameter2:		Y format2:
X parameter:	Freq	X format: Hz
Y scale:	Linear	X scale: Linear
Plot		

2 Select the **Source of frequency data**.

Source of frequency data:	Extracted from data source		Select the source of frequencies at which to plot block data
Frequency data (Hz):	[1e9:1e8:2.9e9]		
Source of input power data:	Extracted from data source		
Input power data (dBm):	[0:19]		
Reference impedance (ohms):	50		
Plot type:	X:Y plane		
Y parameter1:	S11	Y format1: Magnitude (decibels)	
Y parameter2:		Y format2:	
X parameter:	Freq	X format: Hz	
Y scale:	Linear	X scale: Linear	
Plot			

This value is the source of the frequency values at which to plot block data. The following table summarizes the available types of sources for the various types of blocks.

Source of frequency data	Description	Blocks
User-specified	<p>Vector of frequencies that you enter.</p> <p>When you select User-specified in the Source of frequency data list, the Frequency range (Hz) field is displayed. Enter a vector specifying the range of frequencies you want to plot.</p> <p>For example, to plot block data from 0.3 MHz to 5 GHz by 0.1 MHz, enter <code>[0.3e6:0.1e6:5e9]</code>.</p> <hr/> <p>Note When you select PhaseNoise in the Parameter list and User-specified in the Source of frequency data list, the Frequency range (Hz) field is disabled. You use the Phase noise frequency offset (Hz) block parameter to specify the frequency values at which to plot block data.</p>	All physical blocks
Derived from Input Port parameters (Available after running a simulation or clicking the Update Diagram button )	Modeling frequencies derived from the Input Port block parameters. For information on how the blockset computes the modeling frequencies, see “Determine Modeling Frequencies” on page A-3.	All physical blocks
Same as the S-parameters	Frequency values specified in the Frequency block parameter	S-Parameters Passive Network, S-Parameters Amplifier, S-Parameters Mixer
Same as the Y-parameters	Frequency values specified in the Frequency block parameter.	Y-Parameters Passive Network, Y-Parameters Amplifier, Y-Parameters Mixer
Same as the Z-parameters	Frequency values specified in the Frequency block parameter.	Z-Parameters Passive Network, Z-Parameters Amplifier, Z-Parameters Mixer
Extracted from data source	Frequency values imported into the Data file or RFDATA object block parameter.	General Passive Network, General Amplifier, and General Mixer

3 Enter the **Reference impedance**.

Source of frequency data: Extracted from data source
 Frequency data (Hz): [1e9,1e8,2.9e9]
 Source of input power data: Extracted from data source
 Input power data (dBm): [0.19]
 Reference impedance (ohms): 50
 Plot type: X-Y plane
 Y parameter1: S11 Y format1: Magnitude (decibels)
 Y parameter2: Y format2:
 X parameter: Freq X format: Hz
 Y scale: Linear X scale: Linear
 Plot

Enter the reference impedance

This value is the reference impedance to use when plotting small-signal parameters.

4 Select the **Plot type**.

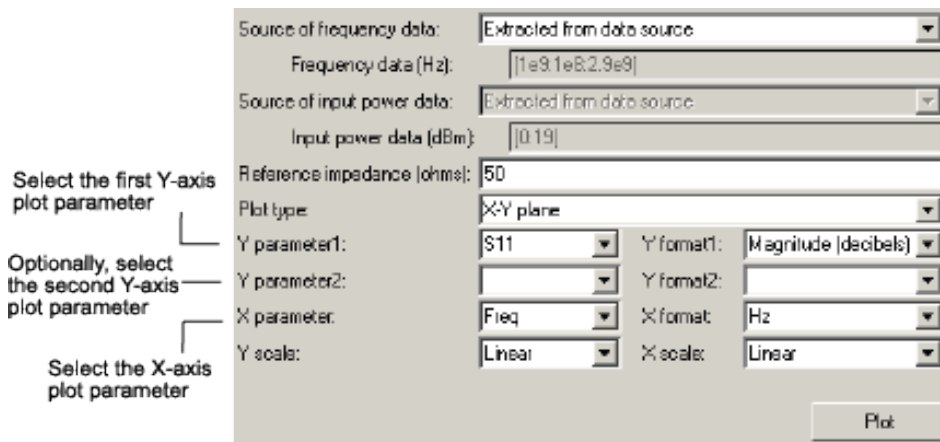
Source of frequency data: Extracted from data source
 Frequency data (Hz): [1e9,1e8,2.9e9]
 Source of input power data: Extracted from data source
 Input power data (dBm): [0.19]
 Reference impedance (ohms): 50
 Plot type: X-Y plane
 Y parameter1: S11 Y format1: Magnitude (decibels)
 Y parameter2: Y format2:
 X parameter: Freq X format: Hz
 Y scale: Linear X scale: Linear
 Plot

Select the plot type

This value is the type of plot. For a description of the options, see “Types of Plots” on page 6-3.

5 Select the following parameters:

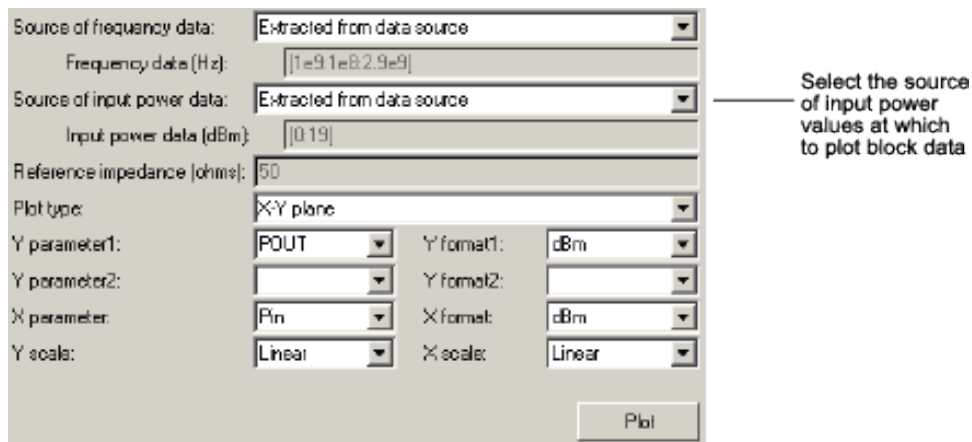
- **Y Parameter1** — The first parameter for the Y-axis.
- **Y Parameter2** — The second parameter for the Y-axis (optional).
- **X Parameter** — The parameter for the X-axis.



These parameters specify the data to be plotted. The available choices vary with the type of plot. For a description of the options for a particular plot type, see the topic on that plot type in “Plot Formats” on page 6-3.

- 6 If you select a large-signal parameter for one or more y-axis parameters, select the **Source of power data**.

Note Large-signal parameters are available only for General Amplifier or General Mixer blocks that contain power data.



This value is the source of the input power values at which to plot block data. The following table summarizes the available types of sources for the General Amplifier and General Mixer blocks.

Source of frequency data	Description
Extracted from data source	Input power values imported into the Data file or RFDATA object block parameter.

Source of frequency data	Description
User-specified	<p>Vector of power values that you enter.</p> <p>When you select User-specified in the Source of power data list, the Input power data (dBm) field is displayed. Enter a vector specifying the range of power values you want to plot.</p> <p>For example, to plot block data from 1 dBm to 10 dBm by 2 dBm, enter [1:2:10].</p>

7 Select the following formats:

- **Y Format1** — The format for the first Y parameter.
- **Y Format2** — The format for the second Y parameter (optional).
- **X Format** — The format for the X parameter.

The screenshot shows a configuration dialog box for plotting. The fields are as follows:

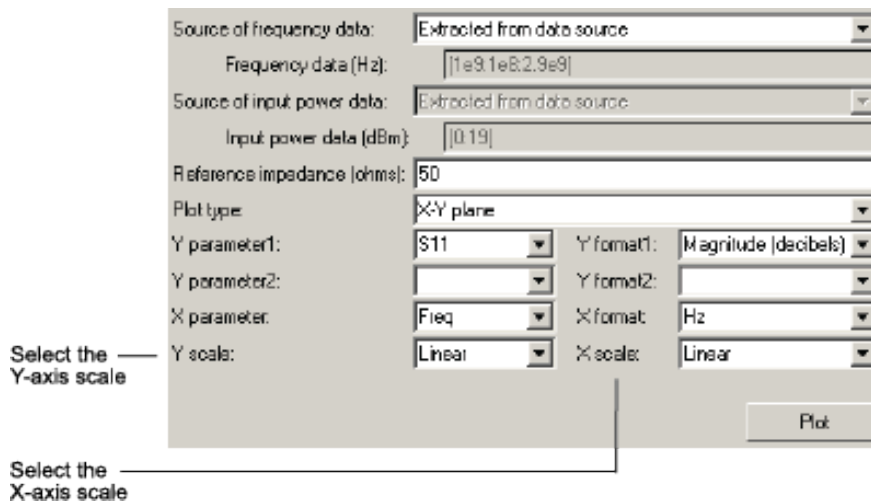
- Source of frequency data: Extracted from data source
- Frequency data (Hz): [1e9:1e8:2.9e9]
- Source of input power data: Extracted from data source
- Input power data (dBm): [0:19]
- Reference impedance (ohms): 50
- Plot type: X-Y plane
- Y parameter1: S11
- Y parameter2: (empty)
- X parameter: Freq
- Y scale: Linear
- Y format1: Magnitude (decibels)
- Y format2: (empty)
- X format: Hz
- X scale: Linear

Annotations on the right side of the dialog box:

- A line points to the Y format1 dropdown: "Select the format for Y parameter1"
- A line points to the Y format2 dropdown: "Optionally, select the format for Y parameter2"
- A line points to the X format dropdown: "Select the format for X parameter"

These are the X and Y formats for plotting the selected parameter. The available choices vary based on the selected parameter. For a description of the options for a particular plot type, see the topic on that plot type in "Plot Formats" on page 6-3.

8 Select the **X Scale** and **Y Scale**.



These are the scales on which to plot the data. The available choices are Linear and Log.

- 9 Click **Plot**.

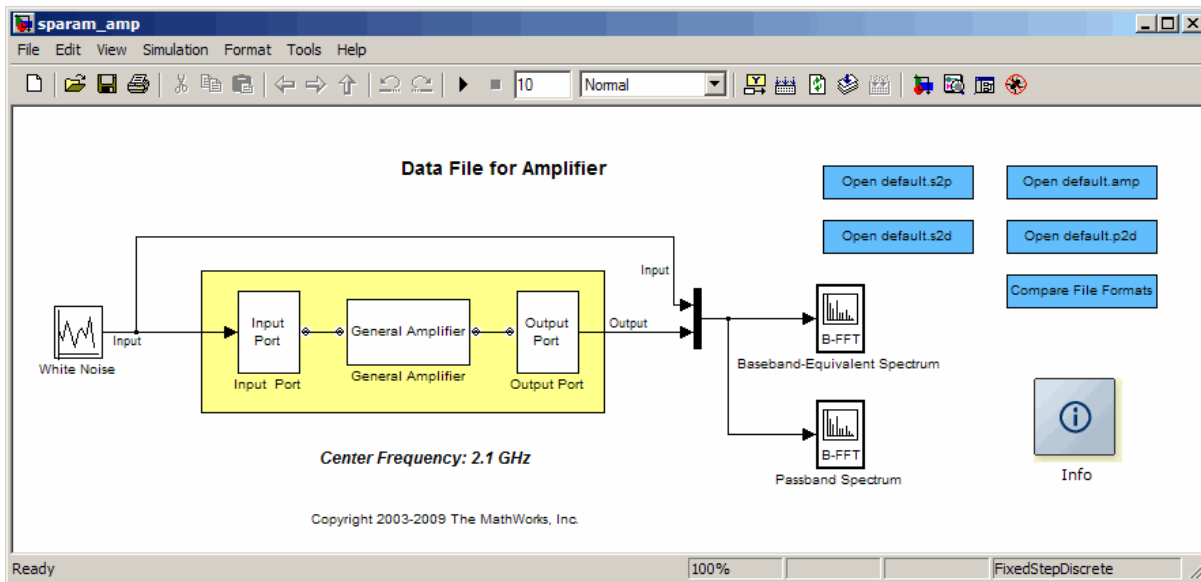
Note By default, the blockset does not add a legend to some plots. To display the plot legend, type `legend show` at the MATLAB prompt.

Example — Plot Component Data on a Z Smith Chart

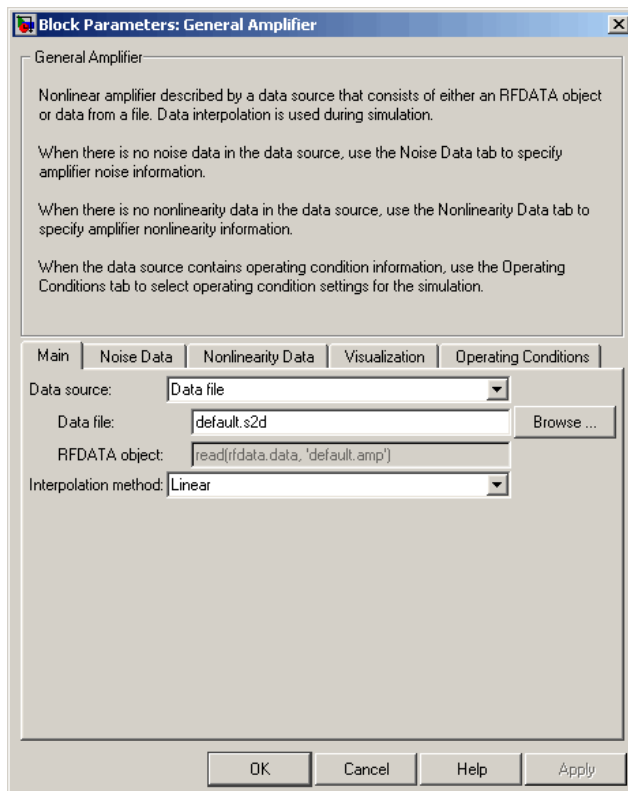
In this example, you simulate the frequency response of an amplifier using data from the `default.s2d` S2D file.

Using a RF Blockset Equivalent Baseband example model, you import the data file into a General Amplifier block and validate the amplifier by plotting the S-parameters of the block on a Z Smith Chart.

- 1 Type `sparam_amp` at the MATLAB prompt to open the “AMP Data File for Amplifier” example.



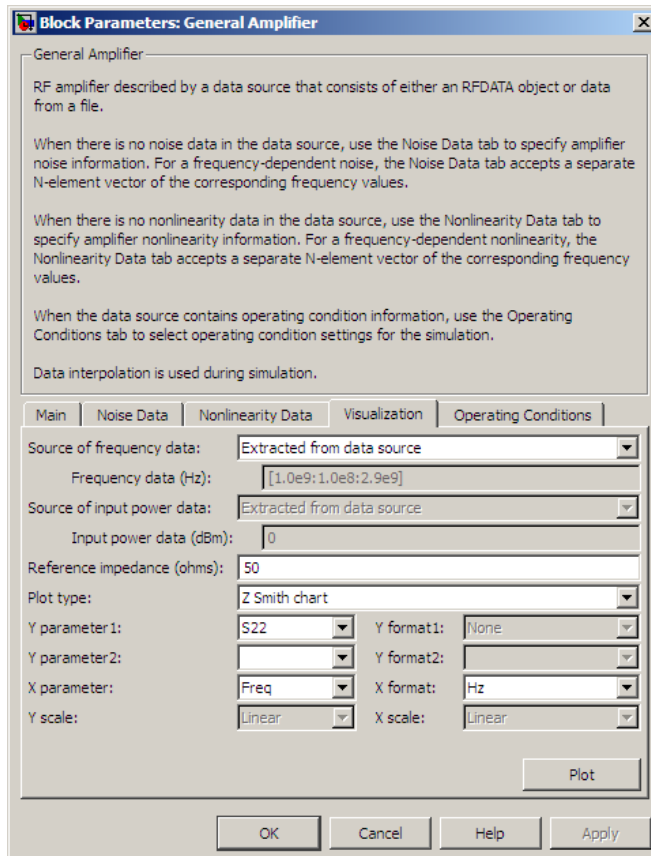
- 2 Double-click the General Amplifier block to display its parameters.



As shown in the preceding figure, the **Data source** parameter is set to **Data file** and the **Data file** parameter is set to **default.s2d**. These values tell the blockset to import data from the file **default.s2d**. The block uses this data, along with the other block parameters, in simulation.

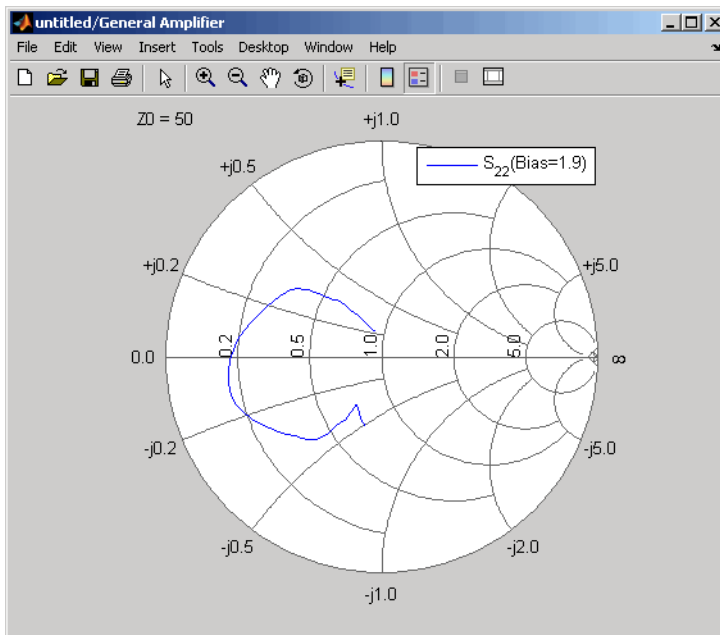
- 3 Select the **Visualization** tab and set the General Amplifier block parameters as follows:

- In the **Plot type** list, select Z Smith chart.
- In the **Y Parameter1** list, select S22.



- 4 Click **Plot**.

This action creates a Z Smith Chart of the S_{22} parameters using the frequency data from the default.s2d file.



General Amplifier Frequency Response

Note To display data tips for a plotted line, select **Tools > Data Cursor**. Click the data cursor on the plotted line to see the frequency and the parameter value at that point. See the MATLAB documentation for more information.

See Also

More About

- “Modify Plots” on page 6-21
- “Update Plots” on page 6-20
- “Create and Modify Subsystem Plots” on page 6-23

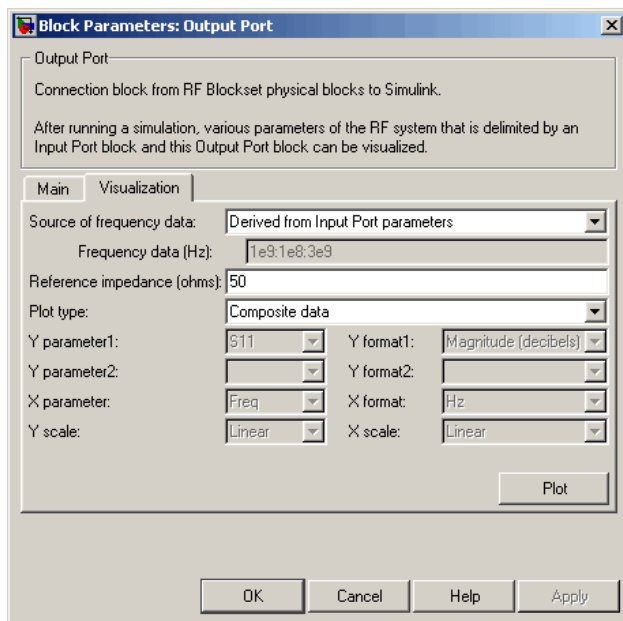
Update Plots

When you run a simulation, the blockset continues to display any open plots, but does not update the plots to reflect new simulation results. You must update the subsystem plots after the simulation to display the behavior of the revised subsystem.

When you make changes to the parameters of blocks that represent individual RF components, you need to update any open plots, because the blockset does not automatically redraw the plots.

To update an existing plot:

- 1 Double-click the block to open the block dialog box, and select the **Visualization** tab.



Example Block Dialog Box Showing Visualization Tab

- 2 Click **Plot**.

See Also

More About

- “Create Plots” on page 6-2
- “Modify Plots” on page 6-21
- “Create and Modify Subsystem Plots” on page 6-23

Modify Plots

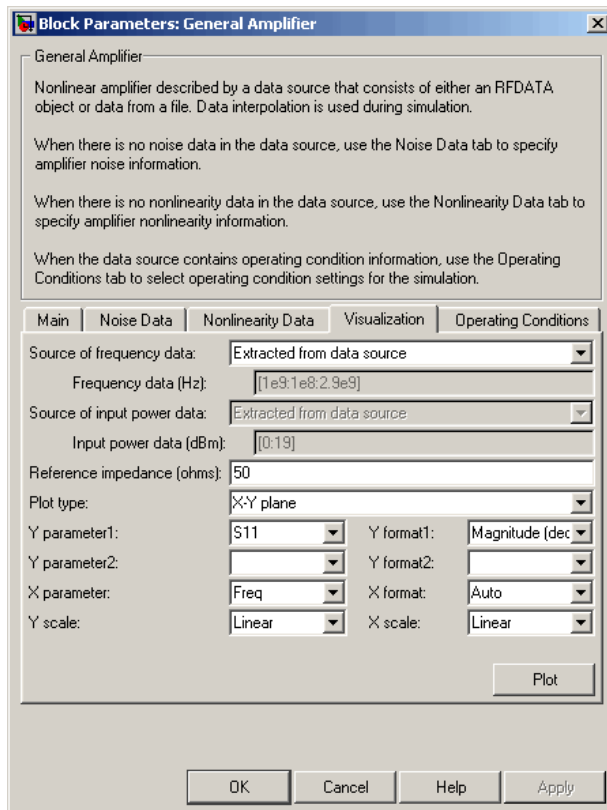
You can modify an existing plot by changing the plot options. The outcome depends on the parameter you change.

The following table summarizes the results of changing the plot options.

Block Parameter	Plot Change
Source of frequency data OR Frequency data (Hz)	Redraws plot using the new frequency data.
Source of power data OR Input power data (dBm)	Redraws plot using the new power data.
Plot type	Draws plot in a new figure using the new plot type. Note If the current plot options are valid for the new plot type, they retain their values. Otherwise, they revert to their default values.
Y Parameter1 OR Y Parameter2	If the new parameter has the same independent variable and format as the one on the plot, the blockset adds the new parameter to the existing plot. Otherwise, it redraws the plot for the new parameter and independent variable.
Y Format1 OR Y Format2	Redraws plot using the new format.
X Parameter	Redraws plot using the new independent variable.
X Format	Redraws plot using the new format.
X Scale	Redraws plot using the new scale.
Y Scale	Redraws plot using the new scale.

To modify a plot:

- 1 Double-click the block to open the block dialog box, and select the **Visualization** tab.



Example Block Dialog Box Showing Plot Parameters

- 2 Change the plot options.
- 3 Click **Plot**.

See Also

More About

- “Create and Modify Subsystem Plots” on page 6-23
- “Create Plots” on page 6-2
- “Update Plots” on page 6-20

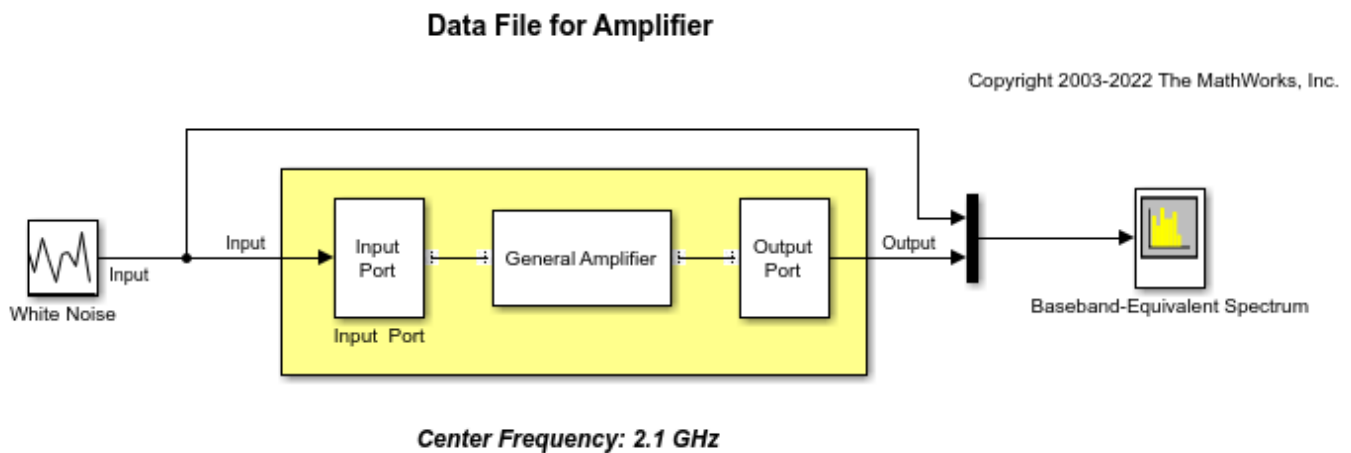
Create and Modify Subsystem Plots

This example shows how to create and modify plots in an equivalent baseband subsystem.

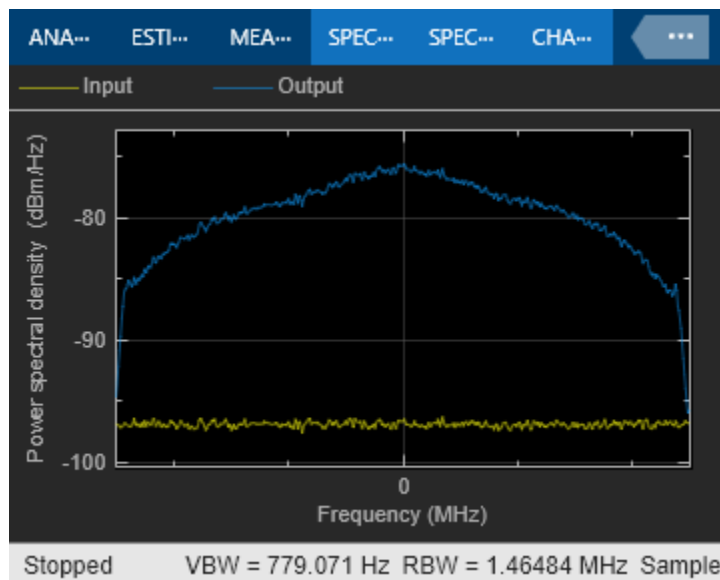
Plot Network Parameters of Subsystem

In this part of the example, you open and run a RF Blockset™ equivalent baseband example that uses data file to specify an amplifier in a physical subsystem. Then, you plot the network parameters of the physical subsystem, which consists of the General Amplifier, the Input Port, and the Output Port blocks.

Open the AMP Data File for Amplifier model.



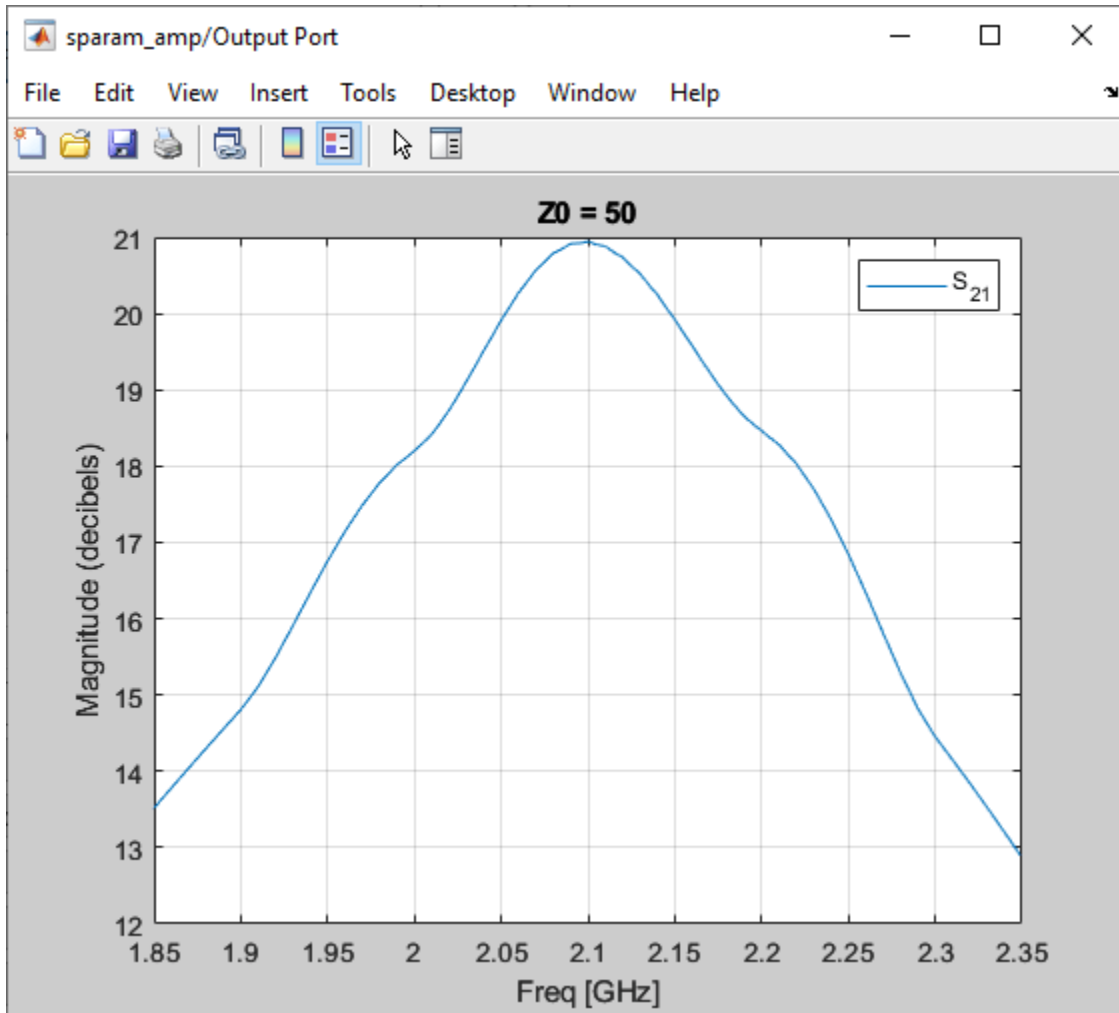
In the model window, click **Run** to run the simulation.



Double-click the Output Port block to open the block mask. Select the **Visualization** tab and set the Output Port block parameters as follows:

- **Source of frequency data** — Derived from Input Port parameters
- **Plot type** — X-Y plane
- **Y Parameter1** — S21

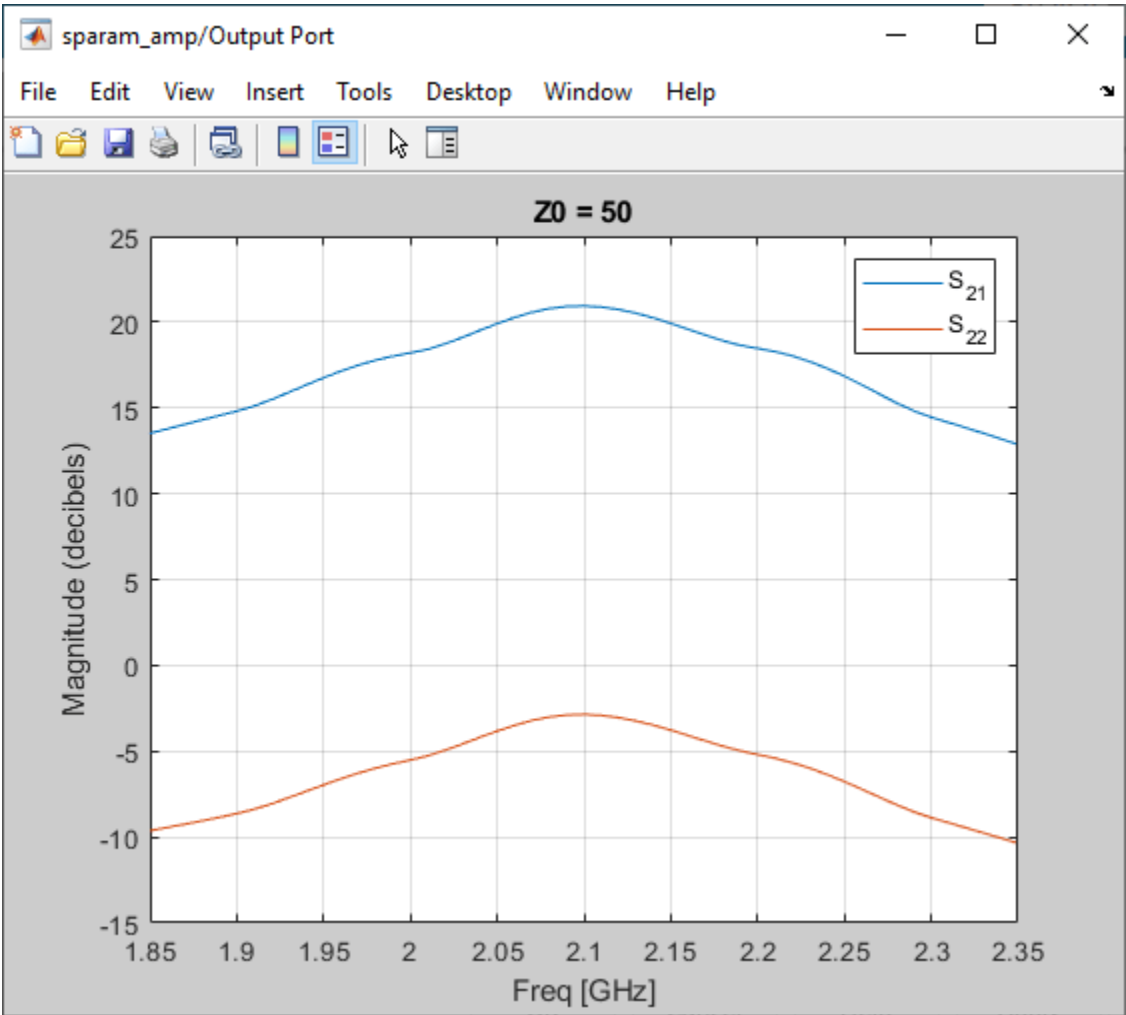
Click **Plot** to plot the magnitude of S21 (in decibels) as a function of frequency on an X-Y plot.



Add Data to Existing Plot

In this part of the example, you add data to the plot you created in Plot Network Parameters of Subsystem section.

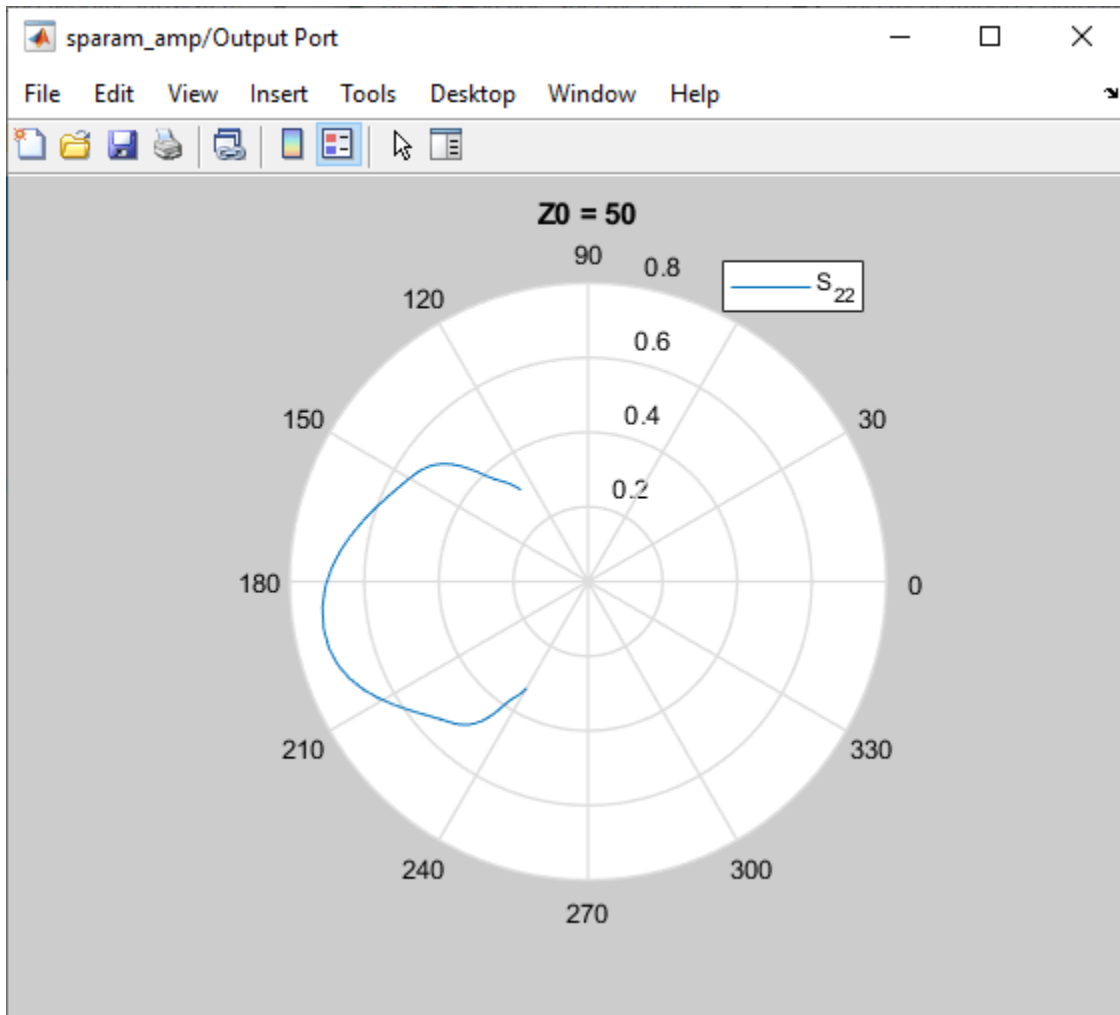
Double-click the Output Port block to open the block mask. Select the **Visualization** tab and set the **Y Parameter1** to S22. Click **Plot** to add S22 to the S21 plot.



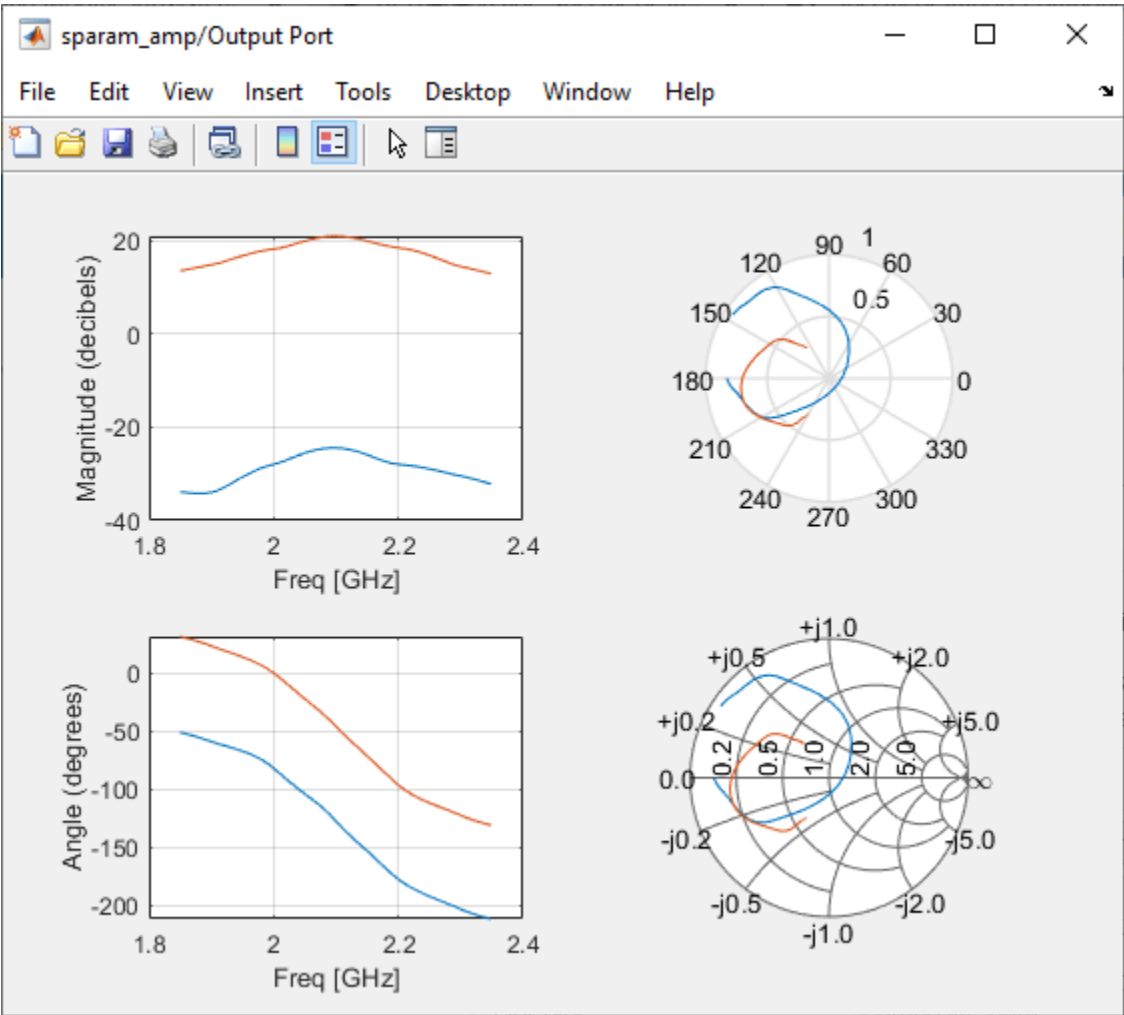
Change Data on an Existing Plot

In this part of the example, you change the data on the plot you created in the previous steps of the example by modifying the **Plot type**.

Double-click the Output Port block to open the block mask. Select the **Visualization** tab and set **Plot type** to **Polar plane**. Click **Plot** to create a Polar plane plot of S22 as a function of frequency.



In the Output Port block, change the **Plot type** to **Composite data** to generate four plots in one figure. The parameters for the plots are defined by the block, so the **Y Parameter1**, **Y Parameter2**, and **X Parameter** parameters are not enabled.



Composite plot displays magnitude, phase, polar, and Z-Smith chart.

See Also

More About

- "Modify Plots" on page 6-21
- "Create Plots" on page 6-2
- "Update Plots" on page 6-20

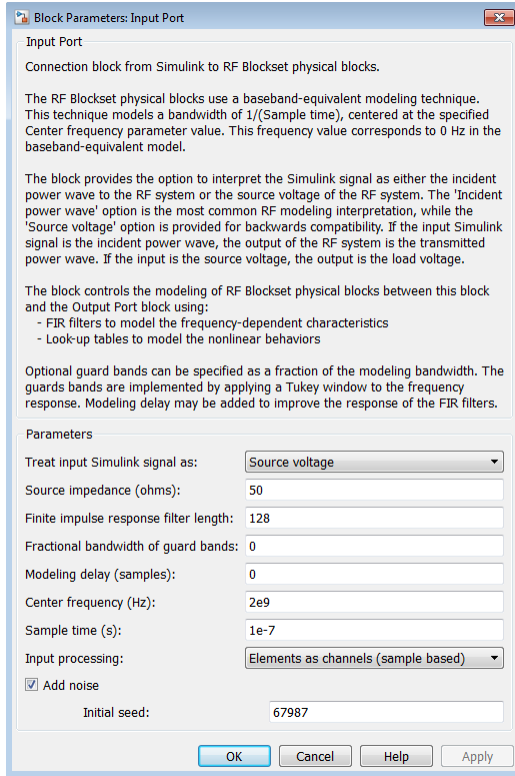
RF Blockset Equivalent Baseband Algorithms

Simulate an RF Model

When you simulate a model that contains physical blocks, RF Blockset Equivalent Baseband software determines the modeling frequencies of the physical system using the Input Port block parameters. The *modeling frequencies* are the frequencies at which the blockset takes information from the blocks to construct the baseband-equivalent model. Then, the software determines the block parameter values at those frequencies and uses the information to create a baseband-equivalent model for Simulink time-domain simulation.

Determine Modeling Frequencies

When you simulate an RF model, the Output Port block uses Input Port block parameters to determine the modeling frequencies f for the physical system that is bracketed between the Input Port block and the Output Port block. f is an N -element vector, where N is the finite impulse response filter length. The modeling frequencies are a function of the center frequency f_c and the sample time t_s . The following figure shows the Input Port block parameters that determine the modeling frequencies.



f_n is the n th element of the vector of modeling frequencies, f , and is given by

$$f_n = f_{\min} + \frac{n-1}{t_s N} \quad n = 1, \dots, N$$

where

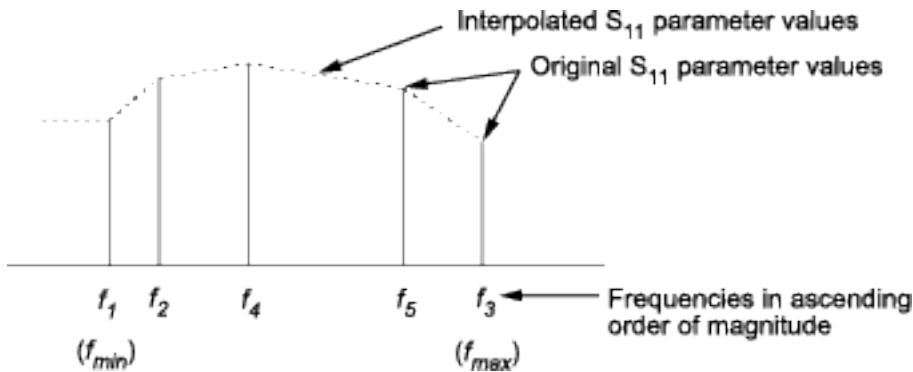
$$f_{\min} = f_c - \frac{1}{2t_s}$$

Map Network Parameters to Modeling Frequencies

In a physical system, each block provides network parameters at different frequencies. These frequencies are not necessarily the modeling frequencies for the physical system in which the block resides. To create a baseband-equivalent model, RF Blockset Equivalent Baseband software must calculate the values of the S-parameters at the modeling frequencies.

Individual physical blocks calculate the S-parameters at the modeling frequencies determined by the Input Port block parameters. Each block interpolates its S-parameters to determine the S-parameters at the modeling frequencies. If the block contains network Y- or Z-parameters, it first converts them to S-parameters.

Specifically, the block orders the S-parameters in the ascending order of their frequencies, f_n . Then, it interpolates the S-parameters using the MATLAB `interp1` function. For example, the curve in the following diagram illustrates the result of interpolating the S_{11} parameters at original frequencies f_1 through f_5 .



The **Interpolation method** field in the individual block dialog boxes enables you to specify the interpolation method as Cubic, Linear (default), or Spline. For more information about these methods, see the `interp1` reference page in the MATLAB documentation.

As shown in the previous diagram, each block uses the parameter values at f_{min} for all modeling frequencies smaller than f_{min} . The block uses the parameter values at f_{max} for all modeling frequencies greater than f_{max} . In both cases, the results may not be accurate, so you need to specify network parameter values over a range of frequencies that is wide enough to account for the block behavior.

Model Noise in an RF System

In this section...

- “Output-Referred Noise in RF Models” on page A-5
- “Calculate Noise Figure at Modeling Frequencies” on page A-6
- “Calculate System Noise Figure” on page A-7
- “Calculate Output Noise Power” on page A-8

The RF Blockset Equivalent Baseband Physical library blocks can model noise. The Input Port block parameters specify whether to include noise in a simulation. When you include noise information in your model, the blockset simulates the noise of the physical system by combining the noise contributions from each individual block. This section explains how the blockset simulates noise from user-specified information. For information on how to add noise to an RF model, see “Model Noise” on page 5-16.

Output-Referred Noise in RF Models

In general, you can specify output-referred noise in one of three ways:

- **Noise temperature** — Specifies the noise in kelvin.
- **Noise factor** — Specifies the noise by the following equation:

$$\text{Noise factor} = 1 + \frac{\text{Noise temperature}}{290}$$

- **Noise figure** — Specifies the noise in decibels relative to the standard reference noise temperature of 290 K. In terms of noise factor

$$\text{Noise figure} = 10\log(\text{Noise factor})$$

These three specifications are equivalent, because you can compute each one from any of the others.

The blockset lets you simulate the noise associated with any physical block in your RF model.

The blockset automatically determines the noise properties of passive blocks from their network parameters. The blockset gets these network parameters either explicitly from the block dialog box or specified data files, or implicitly by calculating them from the specified block parameters.

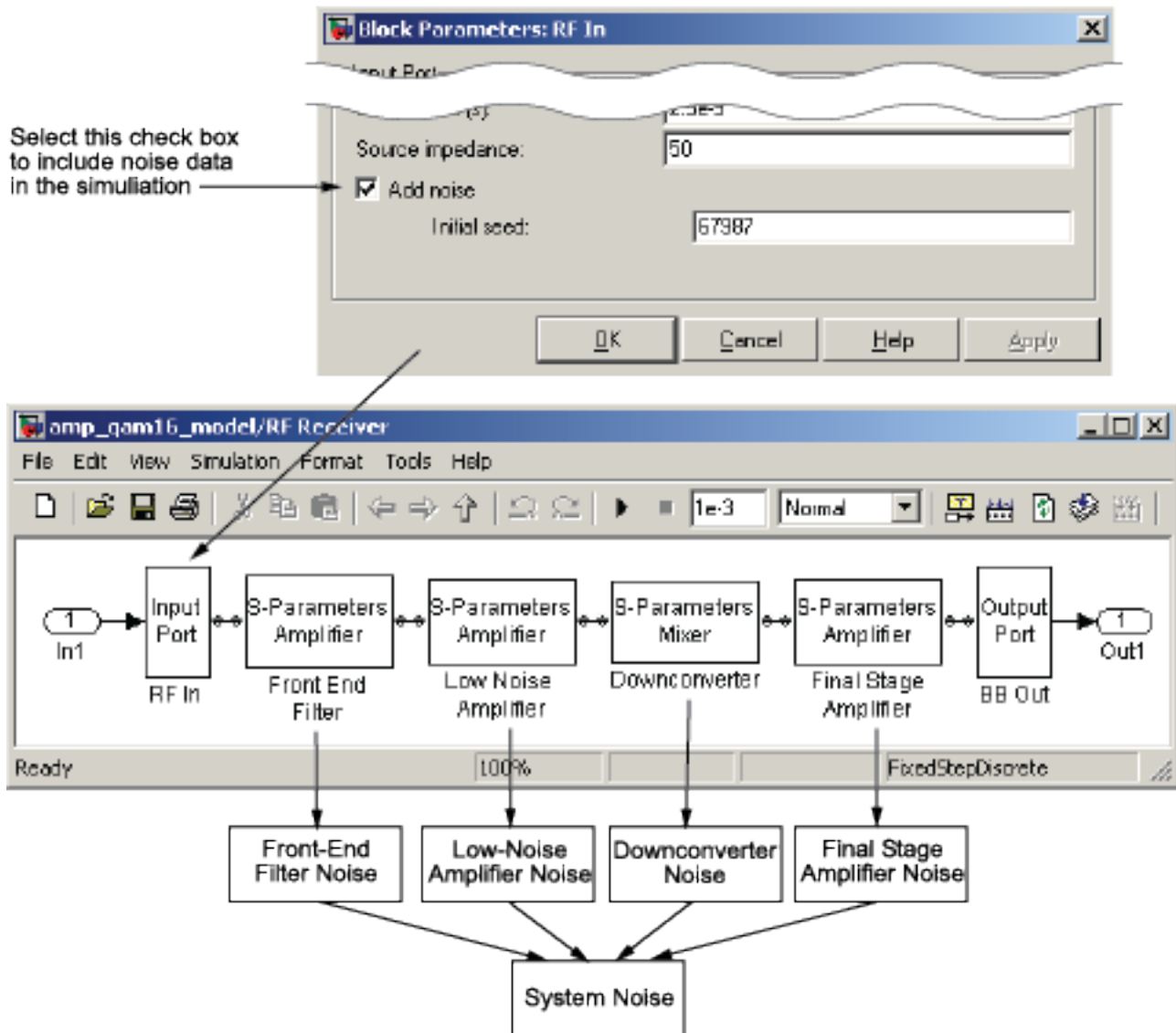
For active devices such as amplifiers and mixers, the noise properties cannot be inferred from network parameters. Therefore, for the amplifier and mixer blocks, you must specify the noise information explicitly, either in the dialog block or the associated data file.

For physical amplifier and mixer blocks, you can specify active block noise in one of the following ways:

- Spot noise data
- Frequency-independent noise figure, noise factor, or noise temperature values
- Frequency-dependent noise figure data (`rfddata.nf`) or spot noise data (`rfddata.noise`) object

These noise specification options are described in “Amplifier and Mixer Noise Specifications” on page 5-16.

When you run the simulation, the blockset first computes the noise figure values for each individual block at the modeling frequencies. Then, it computes the noise figure of the physical system from the individual noise figure values and uses the system noise figure information to calculate the output noise power. This process is shown in the following figure.



Calculate Noise Figure at Modeling Frequencies

To include noise information in a simulation, the blockset must compute the noise figure values of each individual block at the modeling frequencies.

If you specify the frequency-independent noise figure value directly, or if the blockset computes the noise figure value from the block resistance, the blockset uses this value for the noise figure value at each of the modeling frequencies.

If you specify the noise factor or noise temperature value, the blockset computes the noise figure value from the specified value using the equations in the preceding section and uses the computed value for the noise figure value at each of the modeling frequencies.

If you specify frequency-dependent noise figure values using an `rfdata.nf` object, the blockset interpolates the values using the **Interpolation method** specified in the block dialog box to get the noise figure value at each of the modeling frequencies.

If you specify spot noise data, the blockset computes frequency-dependent noise figure information from this data. It takes the minimum noise figure, NF_{min} , equivalent noise resistance, R_n , and optimal source admittance, Y_{opt} , values in the file and interpolates to find the values at the modeling frequencies. Then, the blockset uses the following equation to calculate the noise correlation matrix, C_A :

$$C_A = 2kT \begin{bmatrix} R_n & \frac{NF_{min} - 1}{2} - R_n Y_{opt}^* \\ \frac{NF_{min} - 1}{2} - R_n Y_{opt} & R_n |Y_{opt}|^2 \end{bmatrix}$$

where k is Boltzmann's constant, and T is the noise temperature in Kelvin.

The blockset then calculates the noise factor, F , from the noise correlation matrix as follows:

$$F = 1 + \frac{z^+ C_A z}{2kT \operatorname{Re}\{Z_S\}}$$

$$z = \begin{bmatrix} 1 \\ Z_S^* \end{bmatrix}$$

In the two preceding equations, Z_S is the nominal impedance, which is 50 ohms, and z^+ is the Hermitian conjugation of z .

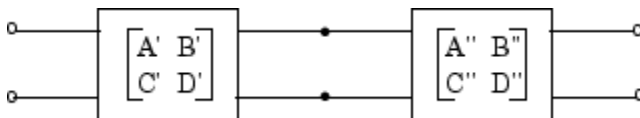
The blockset obtains the noise figure, NF , from the noise factor:

$$NF = 10 \log(F)$$

Calculate System Noise Figure

The blockset uses a recursive process to calculate system noise figure. The noise correlation matrices for the first two elements of the cascade are combined into a single matrix, and the process is repeated.

The following figure shows a cascaded network consisting of two 2-port networks, each represented by its ABCD-parameters.



First, the blockset calculates noise correlation matrices $C_{A'}$ and $C_{A''}$ for the two networks. Then, the blockset combines $C_{A'}$ and $C_{A''}$ into a single correlation matrix C_A using the equation

$$C_A = C_A + \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} C_{A''} \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix}$$

The ABCD-parameter matrices in the cascade combine according to matrix multiplication:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix} \begin{bmatrix} A'' & B'' \\ C'' & D'' \end{bmatrix}$$

If there is another element in the cascade, the same calculations will be performed using these ABCD-parameters as well as the ABCD-parameters corresponding to the following element. The recursion will terminate with a noise correlation matrix pertaining to the entire system. The blockset then calculates the system noise figure from this matrix.

For more information about these calculation techniques, see the following article:

Hillbrand, H. and P.H. Russer, "An Efficient Method for Computer Aided Noise Analysis of Linear Amplifier Networks," *IEEE Transactions on Circuits and Systems*, Vol. CAS-23, Number 4, pp. 235-238, 1976.

Calculate Output Noise Power

The blockset uses noise power to determine the amplitude of the noise that it adds to the physical system using a Gaussian distributed pseudorandom number generator. It uses both the noise temperature and the modeling bandwidth to calculate the noise power:

$$\text{Noise power} = kTB$$

where k is Boltzmann's constant, T is the noise temperature in Kelvin, and B is the bandwidth in hertz.

The blockset computes noise temperature from the specified or calculated noise figure values for the system, and it computes the modeling bandwidth from the model's sample time and center frequency.

Create Complex Baseband-Equivalent Model

In this section...

“Baseband-Equivalent Modeling” on page A-9

“Simulation Efficiency of a Baseband-Equivalent Model” on page A-12

“Example — Select Parameter Values for a Baseband-Equivalent Model” on page A-12

Baseband-Equivalent Modeling

RF Blockset Equivalent Baseband software simulates the physical system in the time domain using a complex baseband-equivalent model that it creates from the passband frequency-domain parameters of the physical blocks. This type of modeling is also known as lowpass equivalent (LPE), complex envelope, or envelope modeling.

To create a complex baseband-equivalent model in the time domain based on the network parameters of the physical system, the blockset performs a mathematical transformation that consists of the following three steps:

- 1 “Calculate the Passband Transfer Function” on page A-9
- 2 “Calculate the Baseband-Equivalent Transfer Function” on page A-11
- 3 “Calculate the Baseband-Equivalent Impulse Response” on page A-11

Calculate the Passband Transfer Function

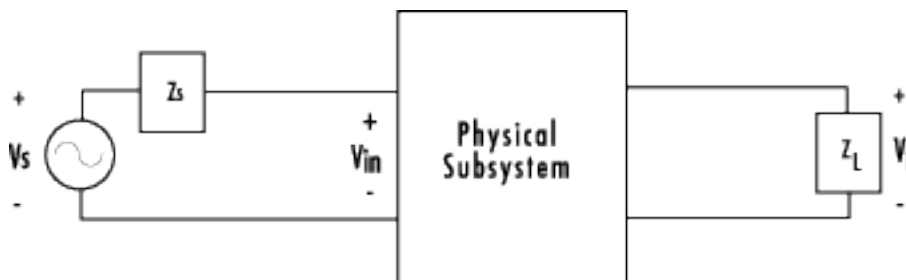
The blockset calculates the passband transfer function from the physical block parameters at the modeling frequencies by calculating the transfer function of the physical subsystem and then applying the Tukey window to obtain the passband transfer function.

Note To learn how the blockset uses the specified network parameters to compute the network parameters at the modeling frequencies, see “Map Network Parameters to Modeling Frequencies” on page A-4.

The transfer function of the physical subsystem is defined as:

$$H(f) = \frac{V_L(f)}{V_S(f)}$$

where V_S and V_L are the source and load voltages shown in the following figure, and f represents the modeling frequencies.



More specifically,

$$H(f) = \frac{S_{21}(1 + \Gamma_l)(1 - \Gamma_s)}{2(1 - S_{22}\Gamma_l)(1 - \Gamma_{in}\Gamma_s)}$$

where

$$\Gamma_l = \frac{Z_l - Z_o}{Z_l + Z_o}$$

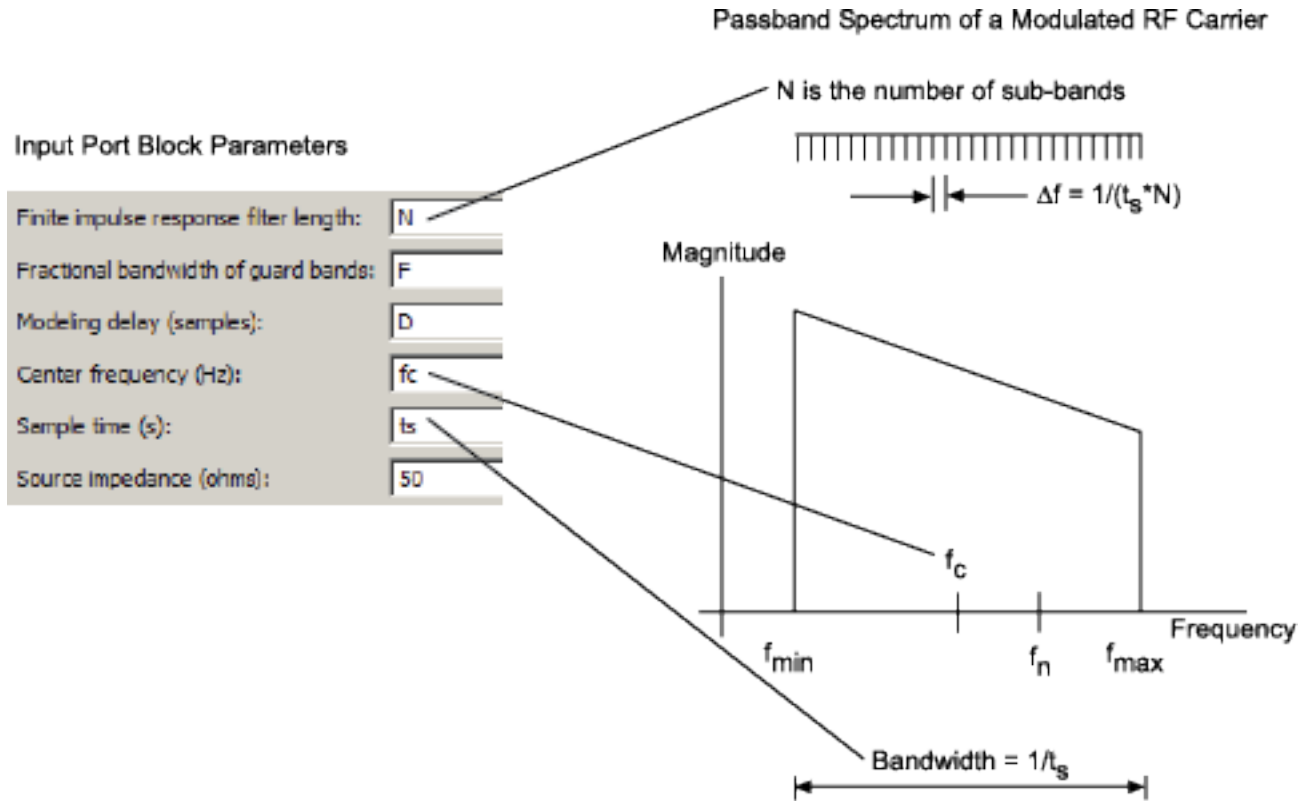
$$\Gamma_s = \frac{Z_s - Z_o}{Z_s + Z_o}$$

$$\Gamma_{in} = S_{11} + \left(S_{12}S_{21} \frac{\Gamma_l}{1 - S_{22}\Gamma_l} \right)$$

and

- Z_S is the source impedance.
- Z_L is the load impedance.
- S_{ij} are the S-parameters of a two-port network.

The blockset derives the transfer function of the physical subsystem from the Input Port block parameters as shown in the following figure.



The blockset then applies the Tukey window to obtain the passband transfer function:

$$H_{passband}(f) = H(f) \cdot tukeywin(N, F)$$

where `tukeywin` is the Signal Processing Toolbox™ `tukeywin` function.

Calculate the Baseband-Equivalent Transfer Function

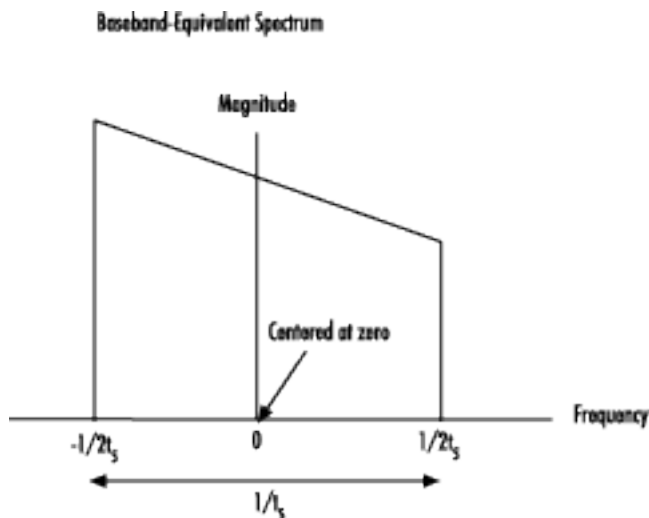
The blockset calculates the baseband transfer function, $H_{baseband}(f)$, by translating the passband transfer function to its equivalent baseband transfer function:

$$H_{baseband}(f) = H_{passband}(f + f_c)$$

where f_c is the specified center frequency.

The resulting baseband-equivalent spectrum is centered at zero, so the blockset can simulate the system using a much larger time step than Simulink can use for the same system. For information on why this translation allows for a larger time step, see “Simulation Efficiency of a Baseband-Equivalent Model” on page A-12.

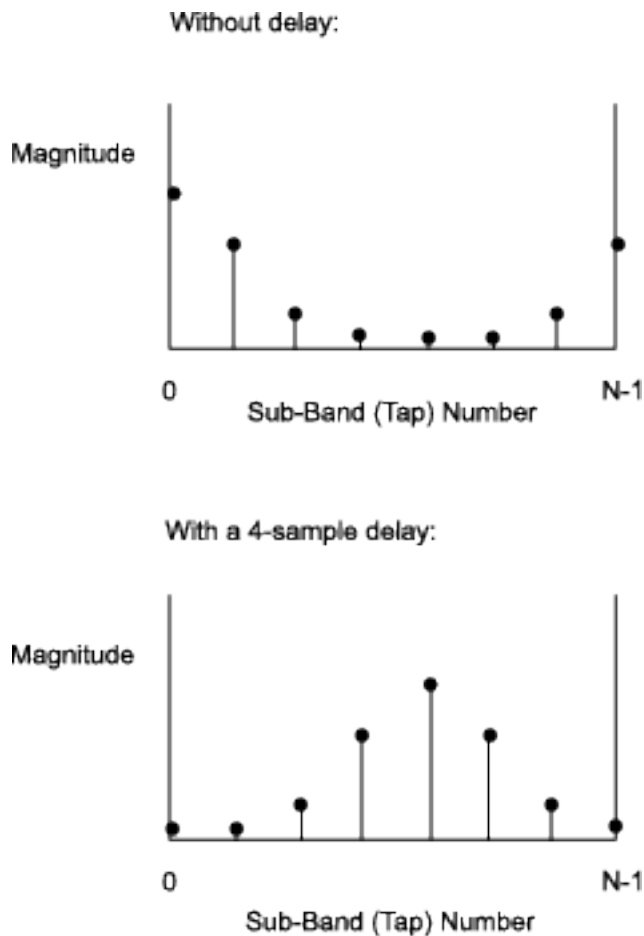
The baseband transfer function is shown in the following figure.



Calculate the Baseband-Equivalent Impulse Response

The blockset calculates the baseband-equivalent impulse response by performing the following steps:

- 1 Calculate the inverse FFT of the baseband transfer function. For faster simulation, the block calculates the IFFT using the next power of 2 greater than the specified finite impulse response filter length. Then, it truncates the impulse response to a length equal to the filter length specified. When the finite impulse response is truncated to the length specified by the user, the effect of the truncation is similar to windowing with a rectangular window.
- 2 Apply the delay specified by the **Modeling delay (samples)** parameter in the Input Port block dialog box. Selecting an appropriate value for this delay ensures that the baseband-equivalent model has a causal response by moving the time window such that the model energy is concentrated at the center of the window, as shown in the following figure:



Simulation Efficiency of a Baseband-Equivalent Model

The baseband-equivalent modeling technique improves simulation speed by allowing the simulator to take larger time steps. To simulate a system in the time domain, Simulink would require a step size of:

$$t_{step} = \frac{1}{2f_{max}}$$

Using the baseband-equivalent model of the same system, whose spectrum has been shifted down by f_c , allows for a much larger time step of:

$$t_{step} = \frac{1}{2(f_{max} - f_c)} = \frac{1}{f_{max} - f_{min}}$$

Example — Select Parameter Values for a Baseband-Equivalent Model

- “Baseband-Equivalent Modeling Example Overview” on page A-13
- “Create the Model” on page A-13
- “Specify Model Parameters” on page A-15

- “Run the Simulation and Analyze the Results” on page A-19
- “Reducing Acausal Response in the Baseband-Equivalent Model” on page A-19
- “Introduce Delay into the Baseband-Equivalent Model” on page A-20

Baseband-Equivalent Modeling Example Overview

In this example, you model an RF transmission line stimulated by a pulse and plot the baseband-equivalent model that the blockset uses to simulate the transmission line in the time domain. You compare the effects of using different parameter values for the baseband-equivalent model. This example helps you understand how to use these parameters to best apply the baseband-equivalent modeling paradigm of performing time-domain simulation using a limited band of frequency data.

Create the Model

In this part of the example, you perform the following tasks:

- “Select Blocks to Represent System Components” on page A-13
- “Build the Model” on page A-13
- “Specify Model Variables” on page A-14

Select Blocks to Represent System Components

In this part of the example, you select the blocks to represent:

- The input signal
- The RF transmission line
- The baseband-equivalent model display

The following table lists the blocks that represent the system components and a description of the role of each block.

Block	Description
Discrete Impulse	Generates a frame-based pulse input signal.
Real-Imag to Complex	Converts the real pulse signal into a complex pulse signal.
Input Port	Establishes parameters that are common to all blocks in the RF transmission line subsystem, including the source impedance of the subsystem that is used to convert Simulink signals to the physical modeling environment.
RLCG Transmission Line	Models the signal attenuation caused by an RF transmission line.
Output Port	Establishes parameters that are common to all blocks in the RF transmission line subsystem. These parameters include the load impedance of the subsystem, which is used to convert RF signals to Simulink signals.
Complex to Magnitude-Angle	Converts the complex signal from the Output Port block into magnitude-angle format.

Build the Model

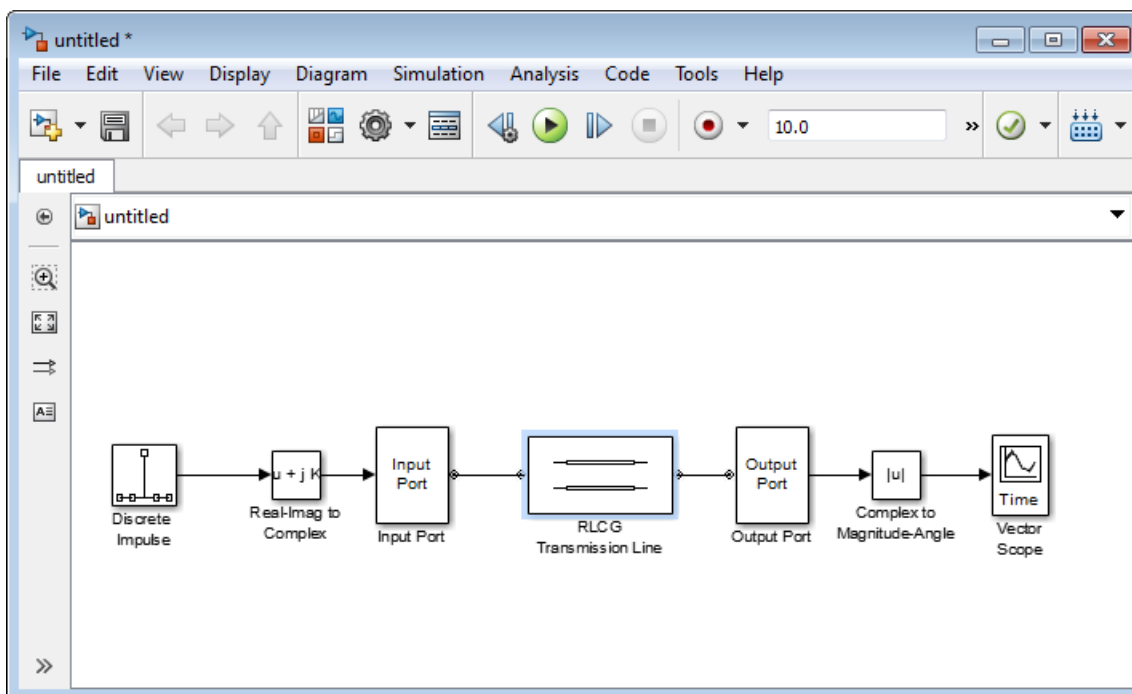
In this part of the example, you create a Simulink model, add blocks to the model, and connect the blocks.

A Create Complex Baseband-Equivalent Model

- 1 Create a model.
- 2 Add to the model the blocks shown in the following table. The Library Path column of the table specifies the hierarchical path to each block.

Block	Library Path	Quantity
Discrete Impulse	DSP System Toolbox > Sources	1
Real-Imag to Complex	Simulink > Math Operations	1
Input Port	RF Blockset > Equivalent Baseband > Input/Output Ports	1
RLCG Transmission Line	RF Blockset > Equivalent Baseband > Transmission Lines	1
Output Port	RF Blockset > Equivalent Baseband > Input/Output Ports	1
Complex to Magnitude-Angle	Simulink > Math Operations	1

- 3 Connect the blocks as shown in the following figure.



Now you are ready to specify model variables.

Specify Model Variables

Type the following at the MATLAB prompt to set up workspace variables for the model:

```
t_s = 5e-10; % Sample time
f_c = 3e9;   % Center frequency
taps = 64;  % Filter length
```

Now you are ready to specify the block parameters.

Specify Model Parameters

In this part of the example, you specify the following parameters to represent the behavior of the system components:

- “Input Signal Parameters” on page A-15
- “Transmission Line Subsystem Parameters” on page A-16
- “Signal Display Parameters” on page A-18

Input Signal Parameters

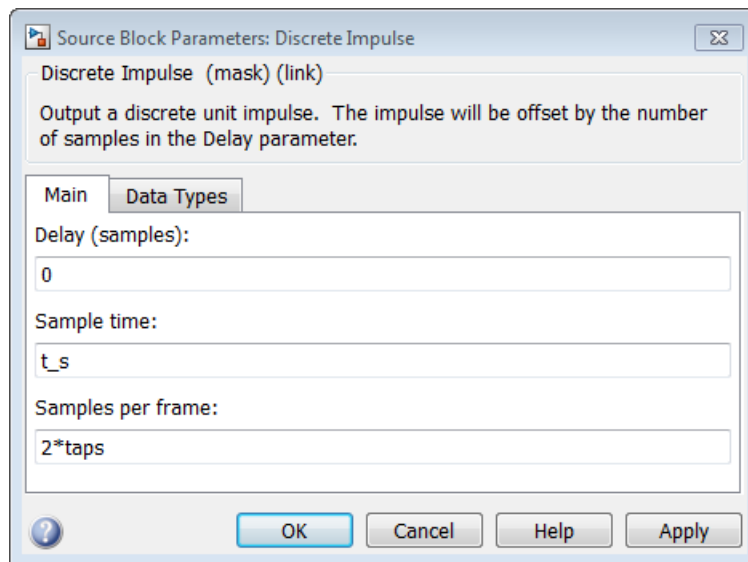
You generate the frame-based complex pulse source signal using two blocks:

- The Discrete Impulse block generates a real pulse signal.
- The Real-Imag to Complex block converts the real signal to a complex signal.

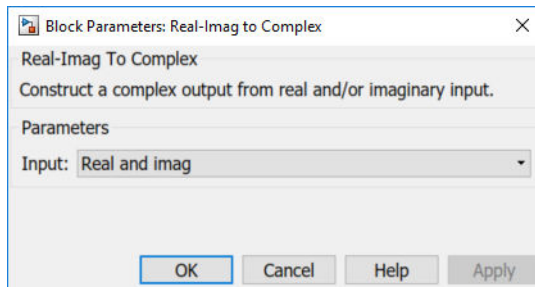
Note All signals in the RF model must be complex to match the signals in the physical subsystem, so you create a complex input signal.

1 In the Discrete Impulse block parameters dialog box:

- Set **Sample time** to t_s .
- Set **Samples per frame** to $2 \cdot \text{taps}$.



2 Set the Real-Imag to Complex block **Input** parameter to `Real`. Changing this parameter changes the number of block inputs from two to one, making the block fully connected.



Transmission Line Subsystem Parameters

In this part of the example, you configure the blocks that model the RF filter subsystem—the Input Port, Transmission Line, and Output Port blocks.

- 1 In the Input Port block parameters dialog box:

- Set **Treat input Simulink signal as** to Incident power wave.

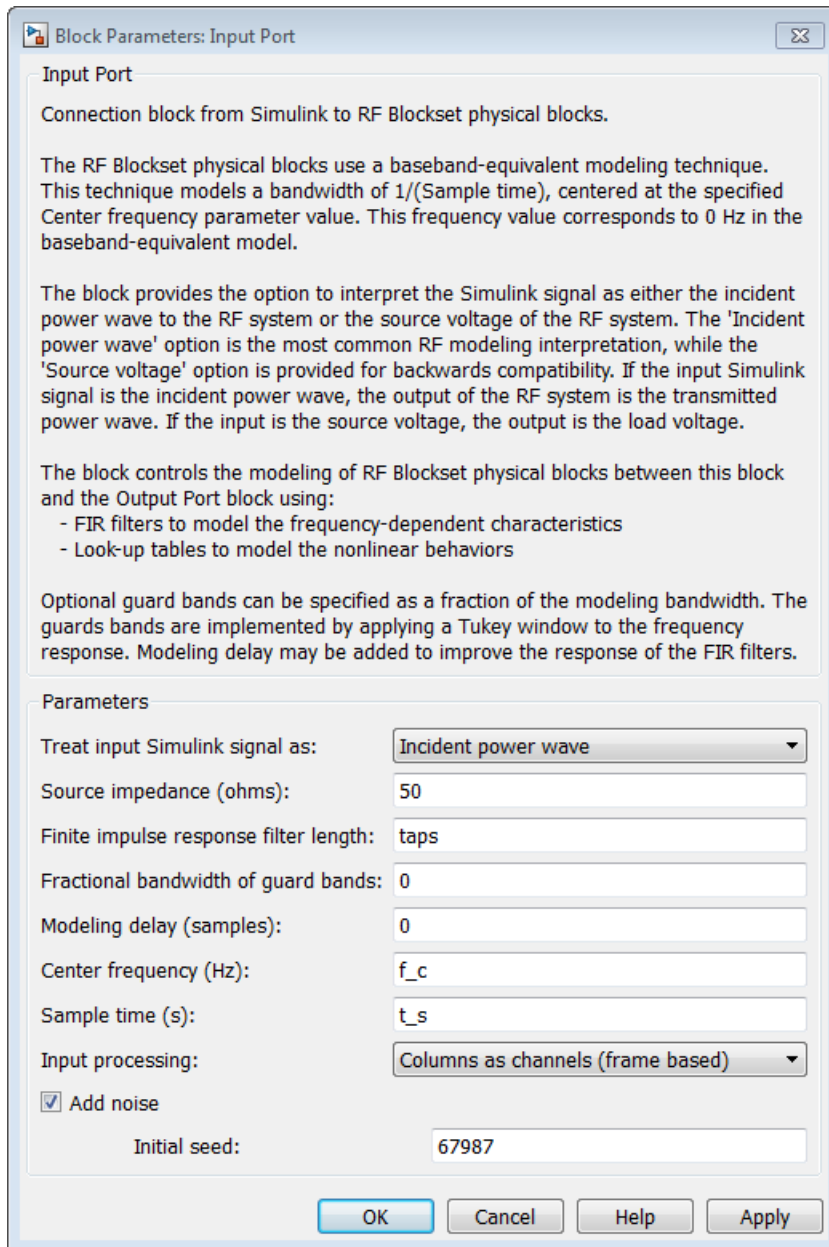
This option tells the blockset to interpret the input signal as the incident power wave to the RF subsystem, and not the source voltage of the RF subsystem.

Note If you use the default value for this parameter, the software interprets the input Simulink signal as the source voltage. As a result, the source and the load that model the Input Port and Output Port blocks, respectively, introduce 6 dB of loss into the physical system at all frequencies. For more information on why this loss occurs, see the note in “Convert to and from Simulink Signals” on page A-22.

- Set **Finite impulse response filter length** to taps.
- Set **Center frequency** to f_c .
- Set **Sample time (s)** to t_s .

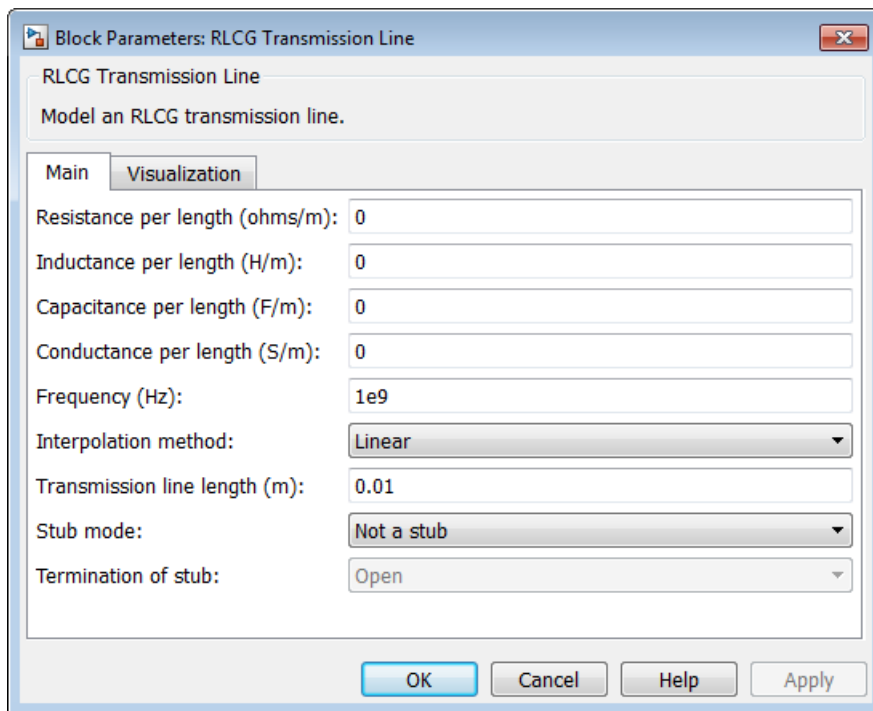
This sample time is equivalent to a modeling bandwidth of $1/t_s$ seconds.

- Set **Input Processing** to Columns as channels (frame based).

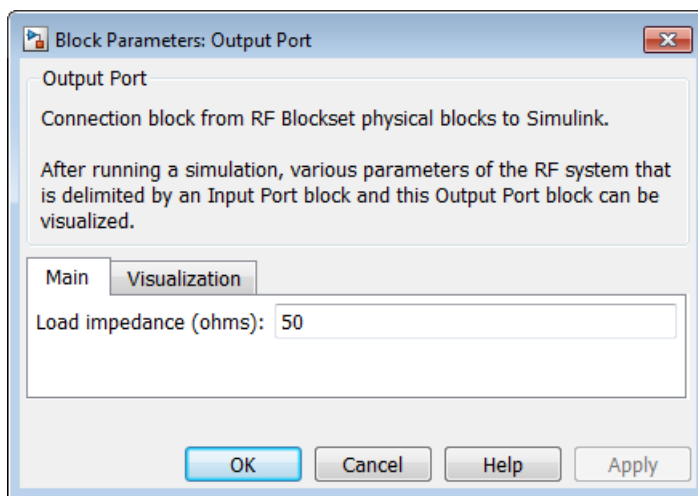


2 In the RLCG Transmission Line block parameters dialog box:

- Set **Inductance per length (H/m)** to 50.
- Set **Capacitance per length (F/m)** to .02.
- Set **Frequency (Hz)** to f_c .
- Set **Transmission line length (m)** to $0.5 * t_s$.



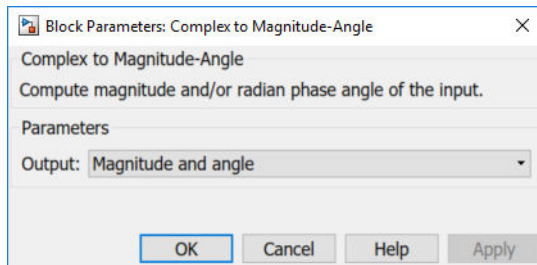
- 3 Accept the default parameters for the Output Port block to use a load impedance of 50 ohms.



Signal Display Parameters

In this part of the example, you specify the parameters that set up the baseband-equivalent model display. You use the Complex to Magnitude-Angle block to convert the RF subsystem output to magnitude format.

- 1 Set the Complex to Magnitude-Angle block **Output** parameter to **Magnitude**. Changing this parameter changes the number of block outputs from two to one, making the block fully connected.

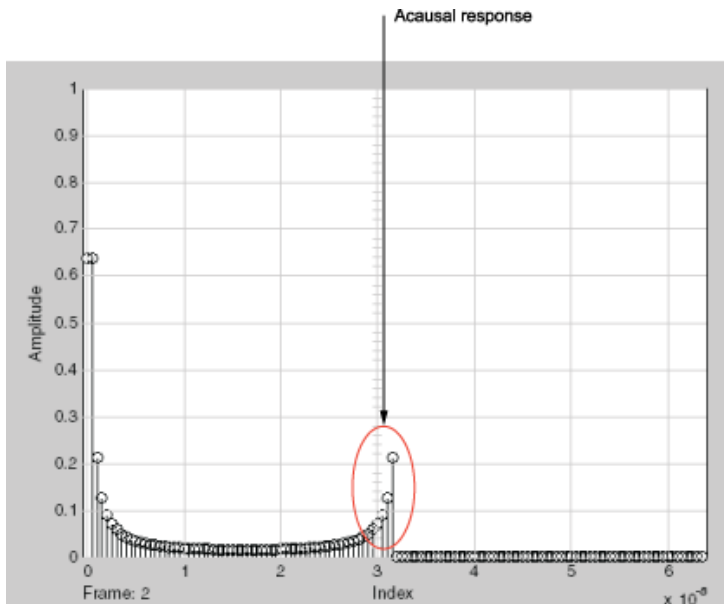


Run the Simulation and Analyze the Results

Before you run the simulation, set the stop time. Click **Simulation** in the **PREPARE**, click **Model Settings** in **Configuration and Simulation**. Enter $2 * t_s * (\text{taps} - 1)$ for the **Stop time** parameter.

To run the simulation, click **Run** in the model window.

This window appears automatically when you start the simulation. The following plot shows the baseband-equivalent model, which contains a significant amount of acausal energy because of the limited bandwidth of the model.



Baseband-Equivalent Model

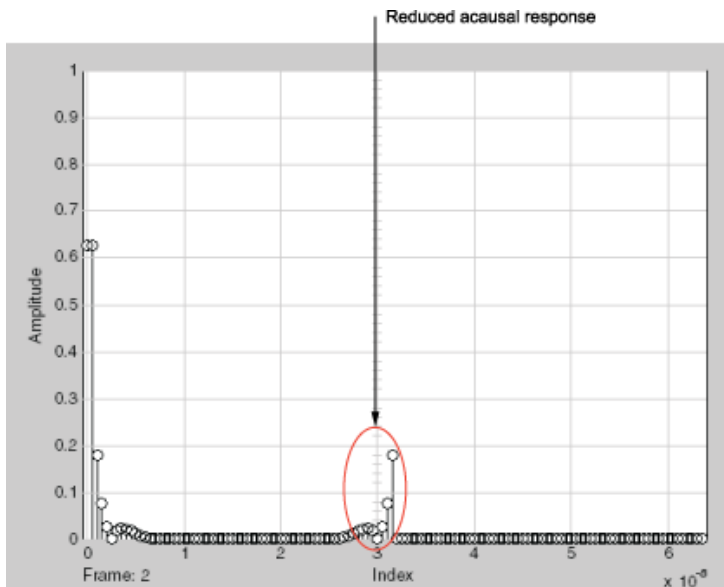
The next part of the example shows you how to reduce this acausal response.

Reducing Acausal Response in the Baseband-Equivalent Model

In this part of the example, you adjust the **Fractional bandwidth of guard bands** parameter. This parameter controls the shaping of the filter that the blockset applies to create the baseband-equivalent model.

- 1 Set the Input Port **Fractional bandwidth of guard bands** parameter to 0.2.
- 2 Rerun the simulation.

You can see that the acausal response is lower than it was for the previous simulation, but there is still some energy wrapping around the end of the model because it is periodic.



Baseband-Equivalent Model with Filter Shaping

Note You can further reduce the acausal response in the baseband-equivalent model by increasing the value of the **Fractional bandwidth of guard bands** parameter above 0.2, but if you use a high value, you compromise the fidelity of the gain of the transmission line.

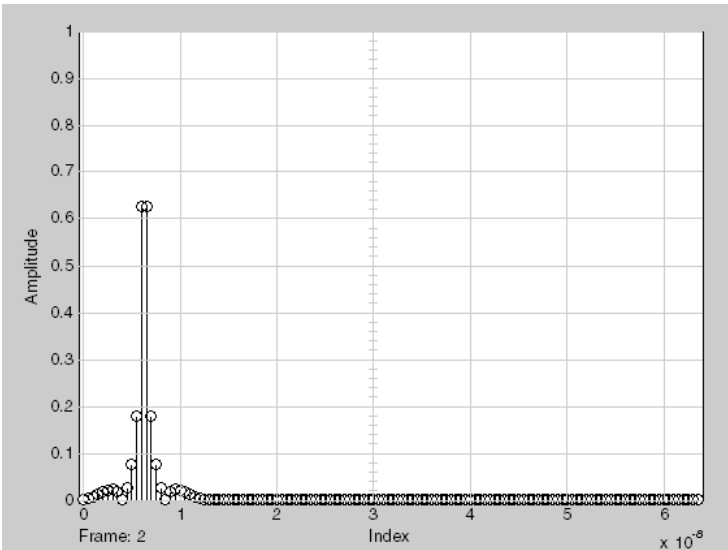
The next section shows you how to shift the response to avoid this wrapping.

Introduce Delay into the Baseband-Equivalent Model

In this part of the example, you adjust the **Modeling delay (samples)** parameter. This parameter controls the delay the blockset applies to create the baseband-equivalent model.

- 1 Set the Input Port **Modeling delay (samples)** parameter to 12.
- 2 Rerun the simulation.

The response of the baseband-equivalent model is concentrated in a small time window. This model provides the most accurate time-domain simulation of the specified band of frequency data.



Baseband-Equivalent Model with Filter Shaping and Delay

See Also

RLCG Transmission Line | Input Port | Output Port

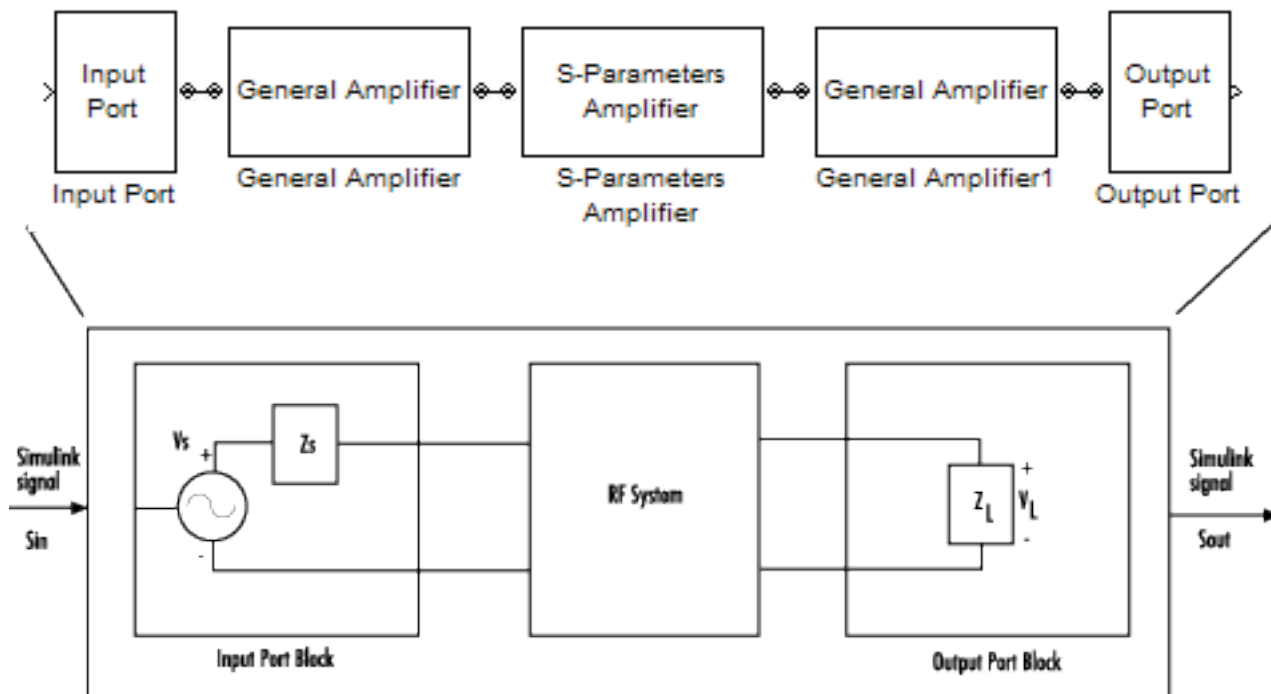
More About

- “Model Nonlinearity” on page 5-14
- “Convert to and from Simulink Signals” on page A-22

Convert to and from Simulink Signals

Signal Conversion Specifications

When you simulate an RF model, the blockset must convert the mathematical Simulink signals to and from the physical modeling environment. The following figure shows the signals involved in the conversion.



Where:

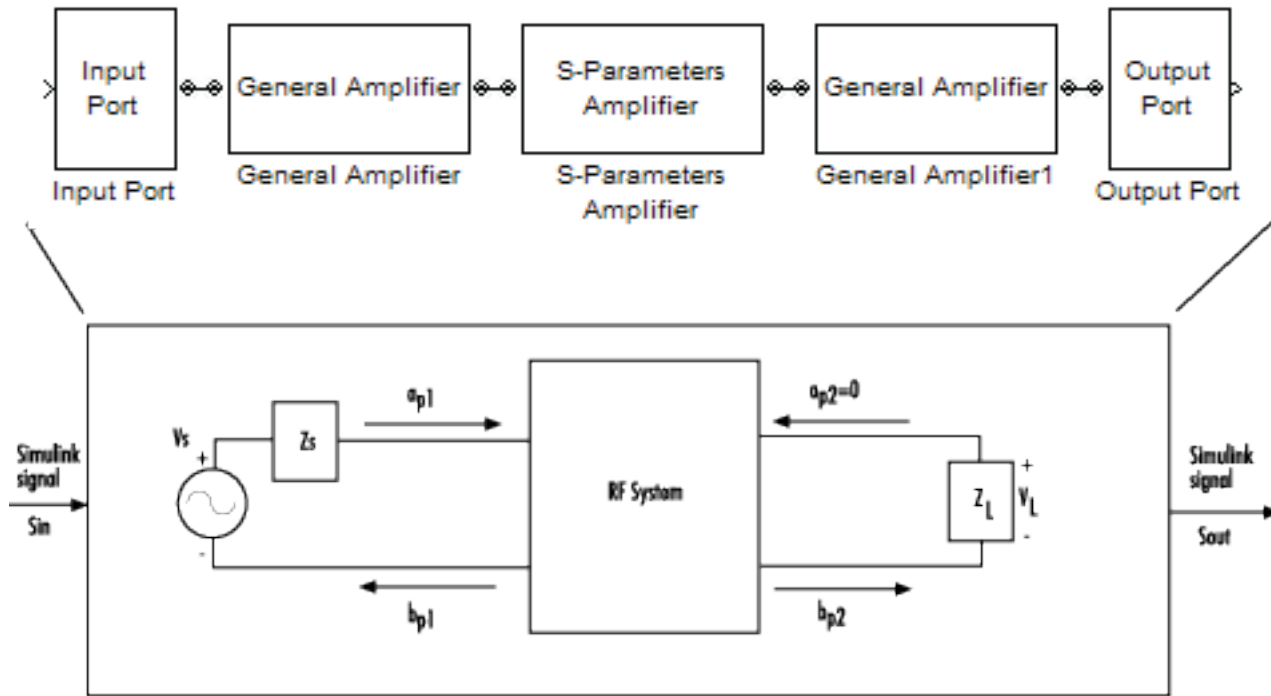
- Z_s is the **Source impedance (ohms)** parameter of the Input Port block.
- Z_L is the **Load impedance (ohms)** parameter of the Output Port block.

There are two options for interpreting the Simulink signal that enters the Input Port block:

- S_{in} is the incident power wave. For more information about this option, see “Interpret Simulink Signals as Incident Power Waves” on page A-22.
- S_{in} is the source voltage. For more information about this option, see “Interpret Simulink Signals as Source Voltages” on page A-24.

Interpret Simulink Signals as Incident Power Waves

The blockset provides the option to interpret the input Simulink signal, S_{in} , as the incident power wave, a_{p1} , at the first port of the RF system. The following figure shows the model for this interpretation.



In the figure, b_{p2} is the transmitted power wave at the second port of the RF system. This is the signal at the output of the Output Port block, S_{out} .

For a 2-port RF system, the incident and transmitted power waves are defined as:

$$a_{p1} = \frac{V_S}{2\sqrt{R_S}}$$

$$b_{p2} = \frac{\sqrt{R_L}}{Z_L} V_L$$

where:

- Z_S , the **Source impedance (ohms)** parameter of the Input Port block, is defined as:

$$Z_S = R_S + jX_S$$

- Z_L , the **Load impedance (ohms)** parameter of the Output Port block, is defined as:

$$Z_L = R_L + jX_L$$

Solving the power wave equations for S_{in} and S_{out} gives the following relationships:

$$S_{in} = \frac{V_S}{2\sqrt{R_S}}$$

$$S_{out} = \frac{\sqrt{R_L}}{Z_L} V_L$$

The implications of this interpretation are:

- $|S_{in}|^2$ is equal to the power available from the source, P_{avs} .
- $|S_{out}|^2$ is equal to the power delivered to the load, P_{out} .

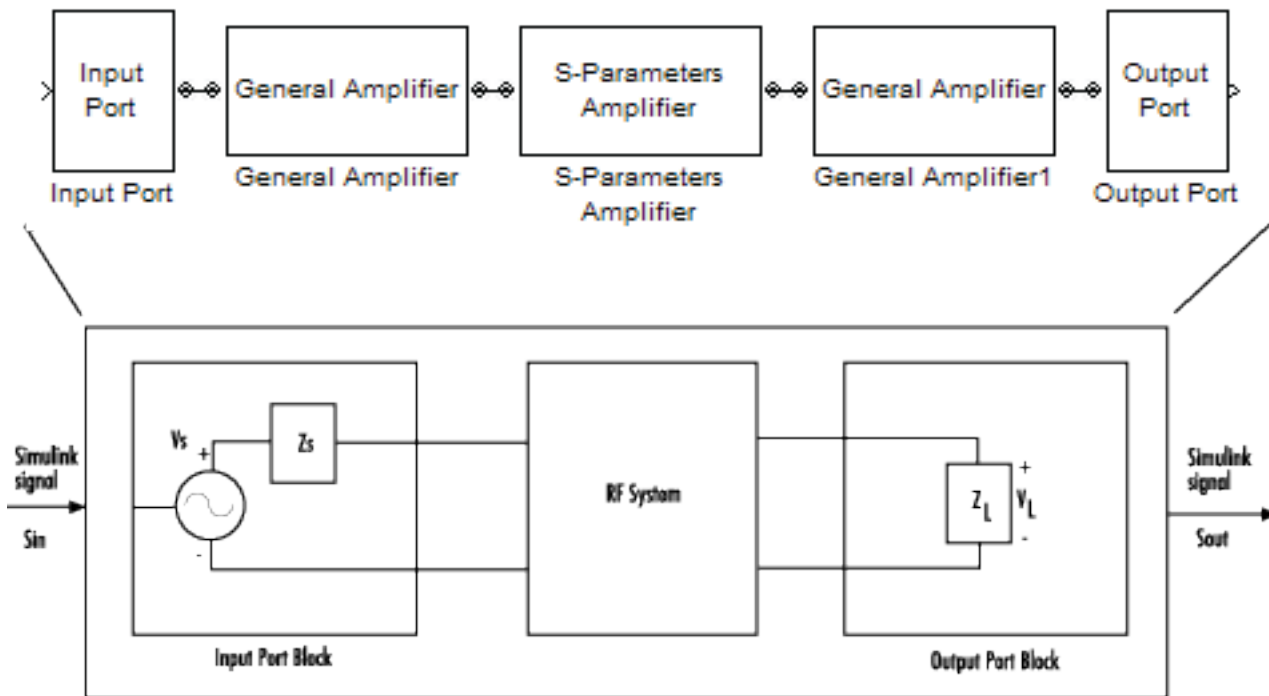
For a linear RF system, $P_{out} = G_t P_{avs}$ where G_t is the transducer power gain. As a result, the Simulink signals at the input and output of the RF system have the following relationship:

$$|S_{out}|^2 = G_t |S_{in}|^2$$

Note You can plot G_t from the Output Port block's **Visualization** tab.

Interpret Simulink Signals as Source Voltages

The blockset provides the option to interpret the input Simulink signal, S_{in} , as the source voltage, V_S , of the RF system. The following figure shows the model for this interpretation.



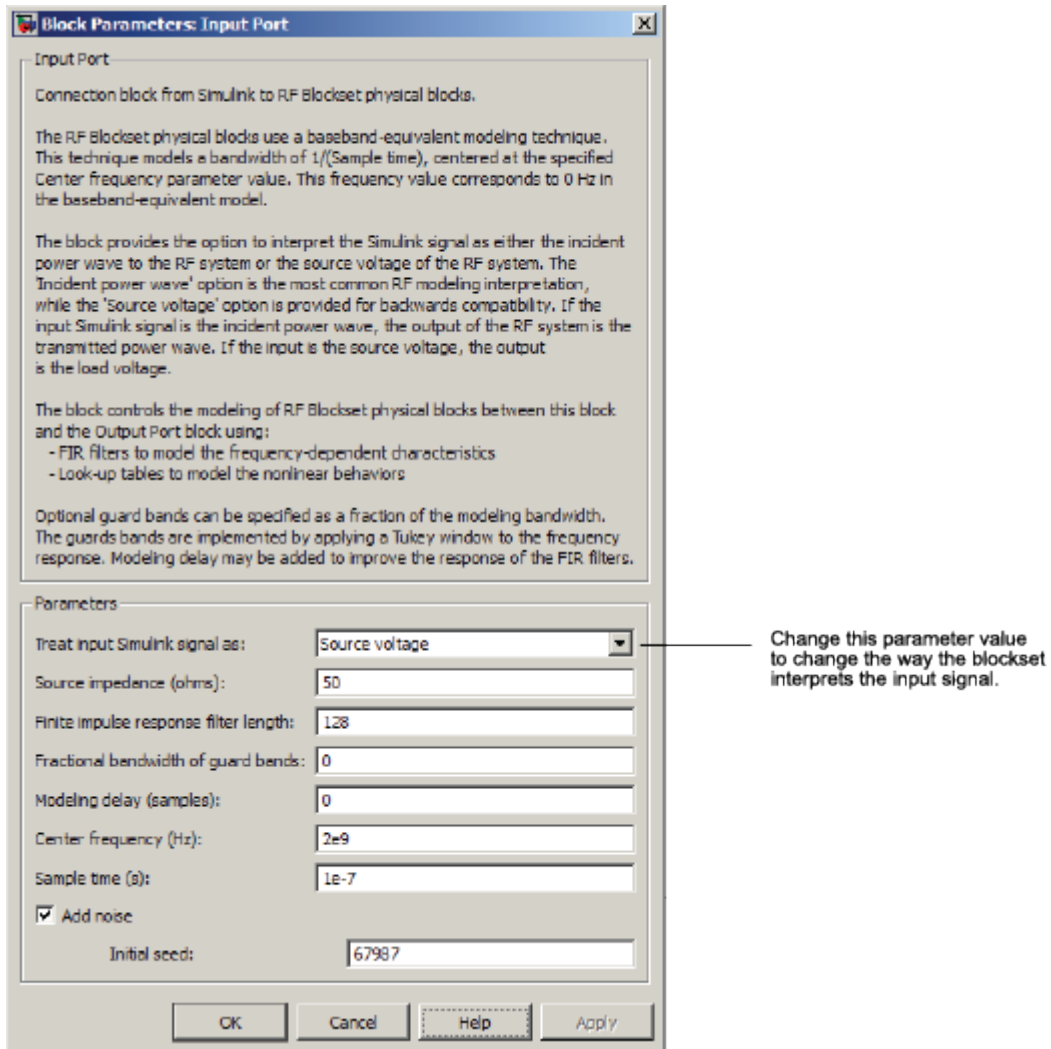
With this interpretation, the signal at the output of the Output Port block is the load voltage, V_L .

The blockset interpretation of the input Simulink signal as the source voltage, V_S , produces different results than the interpretation where the input Simulink signal is the incident power wave. When the source and load impedances are the same and real, the former interpretation produces 6 dB of loss compared to the latter.

Specify Input Signal Conversions

To specify the way in which the blockset interprets the input Simulink signal, you change the value of the **Treat input Simulink signal as** parameter in the Input Port dialog box. The available parameter values are:

- Incident power wave — Interpret the input signal as the incident power wave.
- Source voltage — Interpret the input signal as the source voltage.



See Also

General Amplifier | Input Port | Output Port | S-Parameters Amplifier

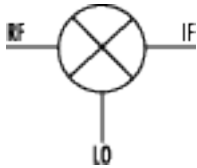
More About

- “Model RF Components” on page 5-2

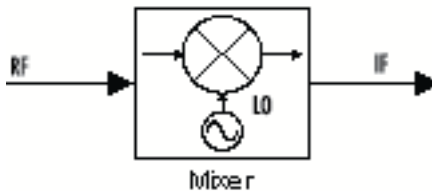
Model Mixers

2-Port Mixer Blocks

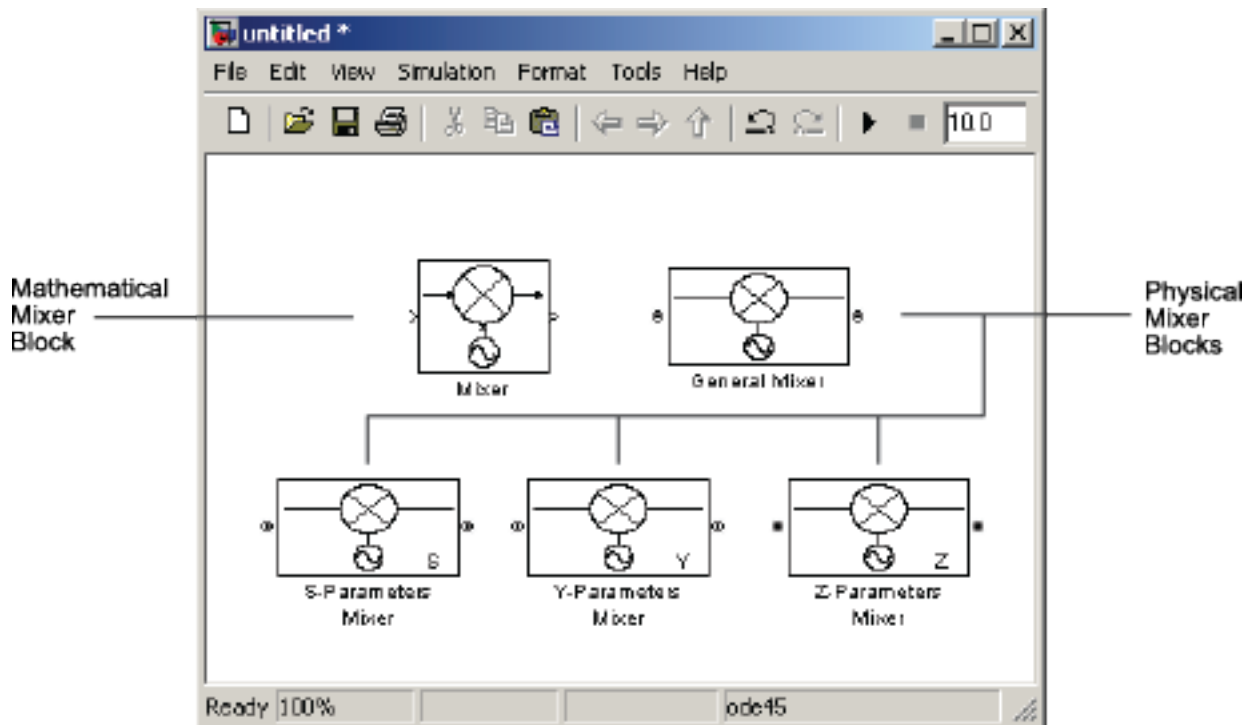
Typically, the block diagram of a mixer has three ports, as shown in the following representation of a downconversion mixer:



The mathematical and physical mixer blocks model both a mixer and a local oscillator, so they have only two ports.



The following figure shows the icons of the mixer blocks. The icons of all the mixer blocks show the internal local oscillator.



For the physical blocks, you can use the **LO frequency (Hz)** parameter to specify the local oscillator's frequency.

For more information, see the individual block reference pages.

Model a Mixer Chain

RF Blockset Equivalent Baseband software uses a baseband equivalent model to simulate RF components in the time domain. The blockset only models a band of frequencies around the carrier frequency of each component; the frequency band is determined by the following parameters of the corresponding Input Port block:

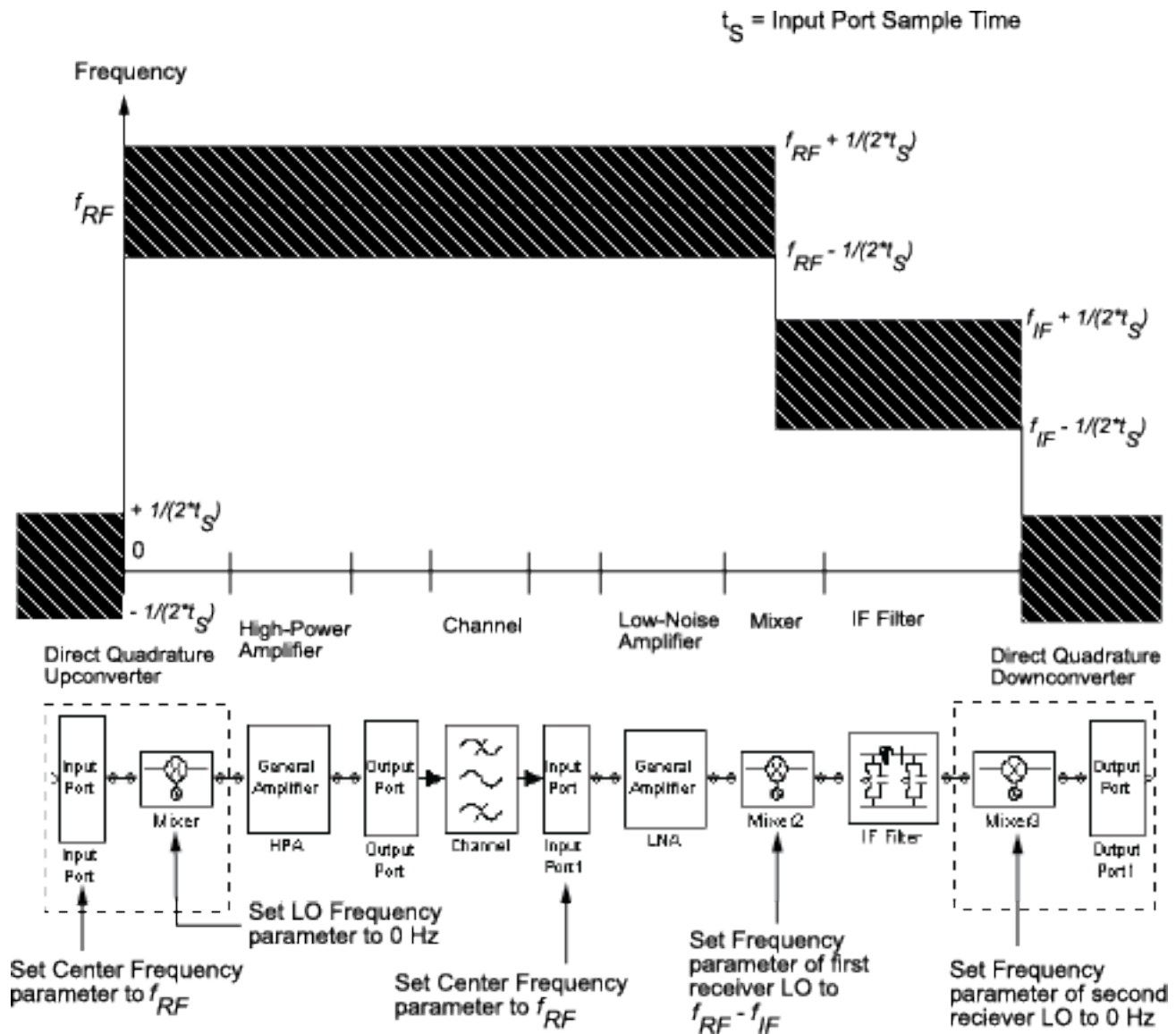
- Reciprocal of the **Sample time (s)**
- **Center frequency (Hz)**

When a mixer is present in a physical subsystem, it shifts the carrier frequency of the signal. This shift affects the frequencies that are used to create baseband equivalent model.

To illustrate how the mixer works, consider a typical RF mixer chain that consists of the following components:

- Direct Quadrature Upconverter
- High-Power Amplifier
- Channel
- Low-Noise Amplifier
- Downconverting Mixer
- IF Filter
- Direct Quadrature Downconverter

The following diagram shows these components and the band of frequencies that are simulated for each component. The signals at the input and output of the cascade are baseband complex. For the cascade, the diagram shows the real passband frequencies that are used to create the baseband-equivalent model, which is centered at zero. For a detailed explanation of how to use the blockset to model quadrature mixers, see “Quadrature Mixers” on page B-6.



See Also

General Mixer

More About

- "Model RF Mixer"

Quadrature Mixers

In this section...

“Use RF Blockset Equivalent Baseband Software to Model Quadrature Mixers” on page B-6

“Model Upconversion I/Q Mixers” on page B-6

“Model Downconversion I/Q Mixers” on page B-7

“Simulate I/Q Mixers” on page B-7

Use RF Blockset Equivalent Baseband Software to Model Quadrature Mixers

RF Blockset software lets you model upconversion and downconversion quadrature mixers using Physical blocks. These mixers convert a complex baseband signal up to and down from the desired carrier frequency by mixing the real and imaginary parts of the signal with a cosine and sine of the same frequency.

Model Upconversion I/Q Mixers

You use the Input Port block to model the upconversion of in-phase/quadrature baseband signals to modulated signals at a finite real carrier frequency. The real component of the block input represents the in-phase signal. The imaginary component of the block input represents the quadrature signal.

To model a perfect quadrature upconversion mixer, use the Input Port block with the **Center frequency (Hz)** parameter set to the carrier frequency.

To model an imperfect quadrature upconversion mixer, use the Input Port block with the **Center frequency (Hz)** parameter set to the carrier frequency. Follow this block immediately by a mixer block with the **LO frequency (Hz)** parameter set to θ . Specify imperfections as follows:

- S-parameters — Use S-parameters to specify imperfections such as frequency response. For a mixer, S_{21} describes the conversion gain, as explained in the Network Parameters section of the reference page for each mixer block. Use purely real and purely imaginary S_{21} parameters to represent multiplying the input signal by a pure cosine and a pure sine, respectively. Use a complex S_{21} parameter to represent multiplying the input signal by a combination of sine and cosine.
- Thermal noise — Use thermal noise to specify temperature-dependent random noise.
- Phase noise — Use the phase noise to specify noise to add to the angle component of the input signal.
- Nonlinearity — Use nonlinearity (specified as output power and phase as a function of input power and frequency in an AMP file or as third-order intercept point) to specify nonlinear mixer behavior as a function of input power.

Note If you specify a nonzero value for the local oscillator frequency of the mixer and set the **Type** parameter to `Upconverter`, the blockset converts the signal to a frequency above the center frequency. The final IF value is the sum of the Input Port center frequency and the mixer local oscillator frequency.

Model Downconversion I/Q Mixers

You use the Output Port block to model the downconversion of in-phase/quadrature modulated carrier signals to baseband signals. The real component of the block output represents the in-phase signal. The imaginary component of the block output represents the quadrature signal.

The finite real carrier frequency is set automatically as the sum of the center frequency of the Input Port block and the LO frequencies in any mixer blocks in the cascade.

Note In the cascade, upconversion mixers increase the carrier frequency and downconversion mixers decrease the carrier frequency.

The Output Port block models a perfect quadrature downconversion mixer. To model an imperfect quadrature downconversion mixer, precede the Output Port block immediately by a mixer block with the **LO frequency (Hz)** parameter set to 0. Specify imperfections as follows:

- **S-parameters** — Use S-parameters to specify imperfections such as frequency response. For a mixer, S_{21} describes the conversion gain, as explained in the Network Parameters section of the reference page for each mixer block. Use purely real and purely imaginary S_{21} parameters to represent multiplying the input signal by a pure cosine and a pure sine, respectively. Use a complex S_{21} parameter to represent multiplying the input signal by a combination of sine and cosine.
- **Thermal noise** — Use thermal noise to specify temperature-dependent random noise.
- **Phase noise** — Use the phase noise to specify noise to add to the angle component of the input signal.
- **Nonlinearity** — Use nonlinearity (specified as output power and phase as a function of input power and frequency in an AMP file or as third-order intercept point) to specify nonlinear mixer behavior as a function of input power.

Note The mixer output frequency must be positive. This means that if you choose a downconverting mixer, the input carrier frequency f_{in} must be greater than the local oscillator frequency f_{lo} . Otherwise, an error appears.

Simulate I/Q Mixers

When you model an I/Q mixer in the blockset, the center frequency you specify in the Input Port block dialog is only used to build a complex-baseband equivalent model of the cascade that represents the mixer. The blockset simulates this model using a fixed time step equal to the sample time that you specify in the Input Port block dialog box.

To examine the model in the Simulink window:

- 1 Click **Modeling > Compile Diagram > Update Model** to update the model diagram.
- 2 Right-click the Output Port block and select **Mask > Look Under Mask**.

For more information on baseband-equivalent modeling, see “Model RF Components” on page 5-2.

See Also

General Mixer

More About

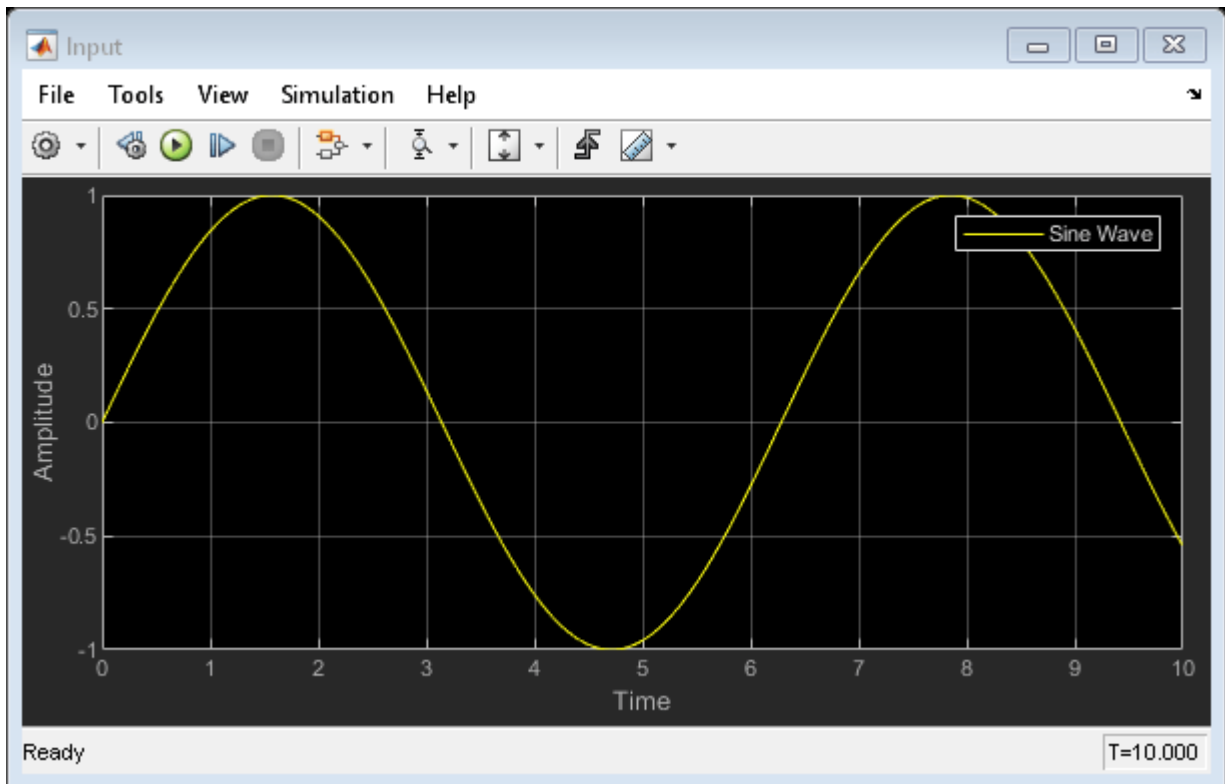
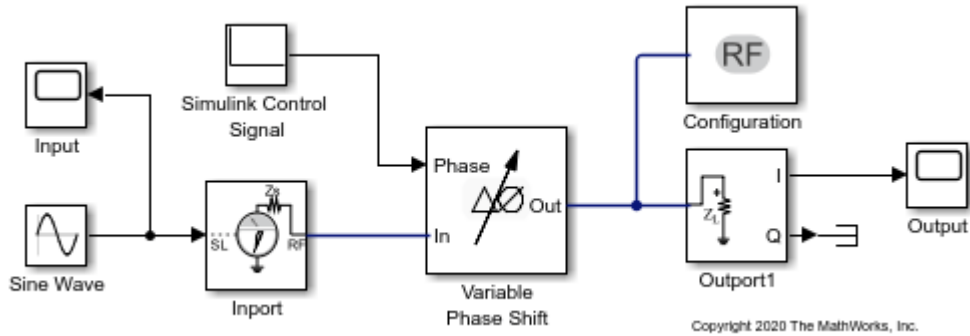
- “Model a Mixer Chain” on page B-4
- “Model RF Components” on page 5-2
- “Model RF Mixer”

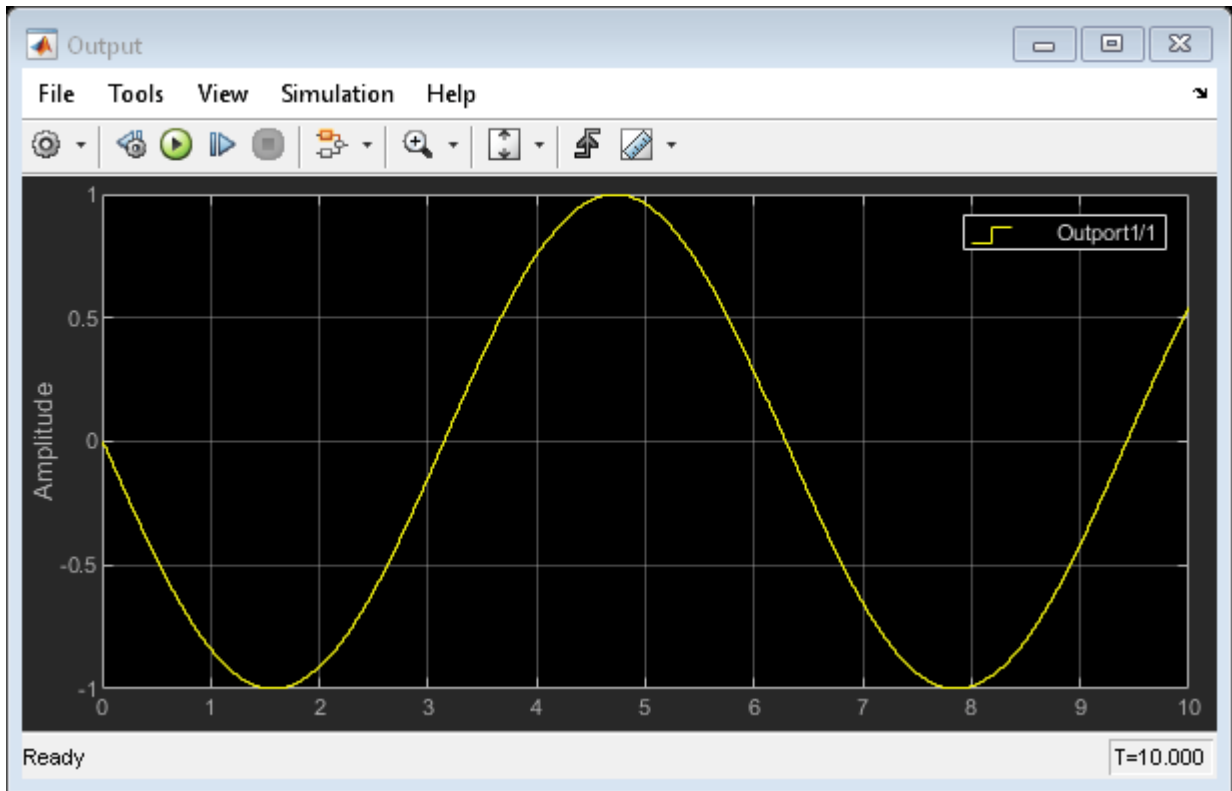
Examples

- “Vary Phase Of Signal During Simulation” on page 7-2
- “Vary Attenuation of Signal During Simulation” on page 7-4
- “Explicitly Simulate Resistor Thermal Noise” on page 7-5
- “Attenuate Signal Power” on page 7-6
- “Demodulate Two-Tone RF Signal Using IQ Demodulator” on page 7-7
- “Modulate Two-Tone DC Signal Using IQ Modulator” on page 7-12
- “Spot Noise Data in Amplifiers and Effects on Measured Noise Figure” on page 7-16
- “Measure Transducer Gain of Device Under Test” on page 7-20
- “Measure Noise Figure of Device Under Test” on page 7-22
- “Measure IIP2 of Device Under Test” on page 7-25
- “Measure IIP3 of Device Under Test” on page 7-27
- “Measure OIP2 of Device Under Test” on page 7-30
- “Measure OIP3 of Device Under Test” on page 7-32
- “Single Pole Triple Throw Switch” on page 7-35
- “Frequency Response of Lowpass Chebyshev Filter” on page 7-38
- “Model LO Phase Noise” on page 7-42
- “Carrier to Interference Performance of Weaver Receiver” on page 7-48
- “Modulate Two-Tone DC Signal Using IQ Modulator” on page 7-55
- “Measurement of Gain and Noise Figure Spectrum” on page 7-59
- “Idealized Baseband Amplifier with Nonlinearity and Noise” on page 7-72
- “Use Ladder Filter Block to Filter Gaussian Noise” on page 7-74
- “Measure S-Parameter Data of Chebyshev Filter” on page 7-77
- “Measure S-Parameter of Nonlinear System” on page 7-81
- “Simulation of RF Systems with Antenna Blocks” on page 7-87
- “Power Amplifier Characterization” on page 7-92
- “Modulate Quadrature Baseband Signals Using IQ Modulators” on page 7-104
- “Intermodulation Analysis of Mathematical Amplifier” on page 7-107
- “Create Virtual Connections Using Connection Label Block” on page 7-109
- “Model Wilkinson Power Divider” on page 7-110
- “Modulate Input Signal Onto Square Carrier Wave” on page 7-115
- “Time-Domain Filtering of RF Complex Baseband Signals in Simulink” on page 7-122
- “Model RF Complex Baseband S-Parameters in Simulink” on page 7-125

Vary Phase Of Signal During Simulation

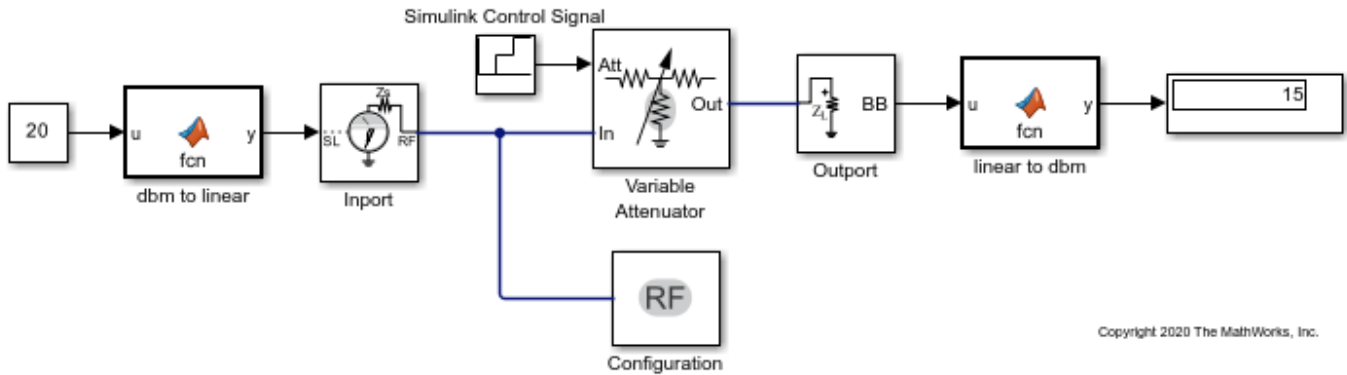
Use the Variable Phase Shift block to shift the phase of a sine wave to 180 degrees. Use Repeating Sequence Stair block as a Simulink control signal to control the phase of the signal. To see the variation in phase to 180 degrees, first open and run the model. During simulation, change the value of the Simulink control signal to 90 degrees and see a change in phase in the Output Scope.





Vary Attenuation of Signal During Simulation

Use the Variable Attenuator block to attenuate a 20 dB constant signal. Use the Repeating Sequence Stair block as a Simulink control signal to vary the attenuation of the signal. In this model, the signal attenuation varies between 5 and 10 dB during simulation. To see the variation in attenuation, open and run the model.

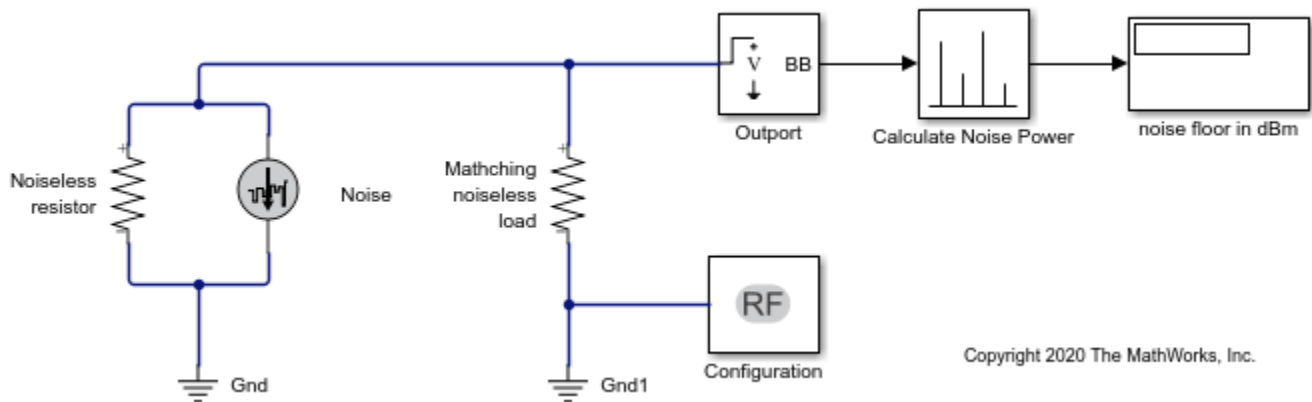


Explicitly Simulate Resistor Thermal Noise

Use the Noise block to calculate the classic thermal noise floor, kT , for a matched resistor circuit. Model configuration is as follows:

- Time step of the model is $1e-6$ and frequency is 2 GHz.
- The Resistor noise source is modelled explicitly to make it noiseless. The resistance is 50 ohms. In the Resistor blocks, Simulate Noise is not selected.
- The Noise current source is parallel to the Resistor block models the noise. In the Noise block, the Source type is set to Ideal current to make it a current source. The Noise spectral density is defined as $4kT/R$ in A^2/Hz . Where k is Boltzmann constant and T is temperature in kelvin.
- The masked block, Calculate Noise Power, calculates the noise floor as a standard deviation of the output signal.

```
open_system('model_simrf_noise_source1')
```



To run the model, select **Simulation > Run**. With the bandwidth included using the Configuration block, noise power is in the range of -173.98 to 174.1 dBm

See Also

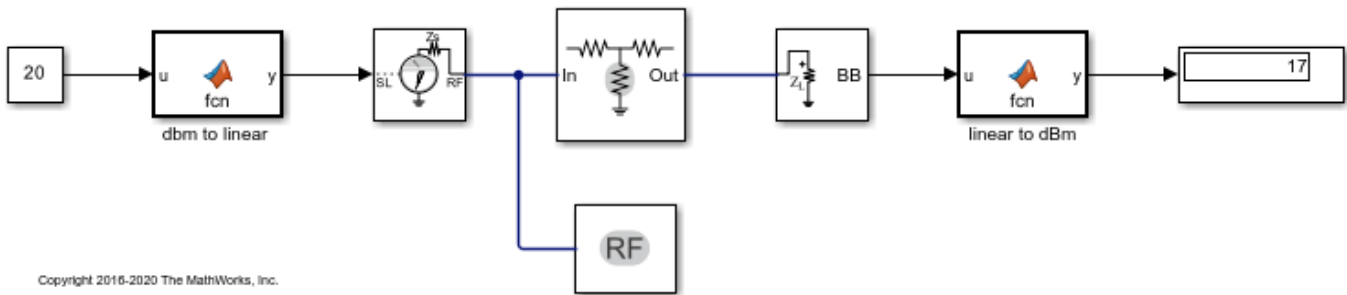
Noise | S-Parameters | Configuration

Related Examples

- “Spot Noise Data in Amplifiers and Effects on Measured Noise Figure” on page 7-16

Attenuate Signal Power

Use the Attenuator block to attenuate a constant signal of 20 dB by 3 dB.



Demodulate Two-Tone RF Signal Using IQ Demodulator

Use the IQ Demodulator block to demodulate a two-tone RF signal to DC level. Observe the impairments in the demodulated output signal such as images due to gain imbalance, intermodulation distortion, and output third-order intercept (OIP3).

The two tones are at 10 MHz and 15 MHz. The power of each tone is -30 dBm. The carrier frequency is 2 GHz.

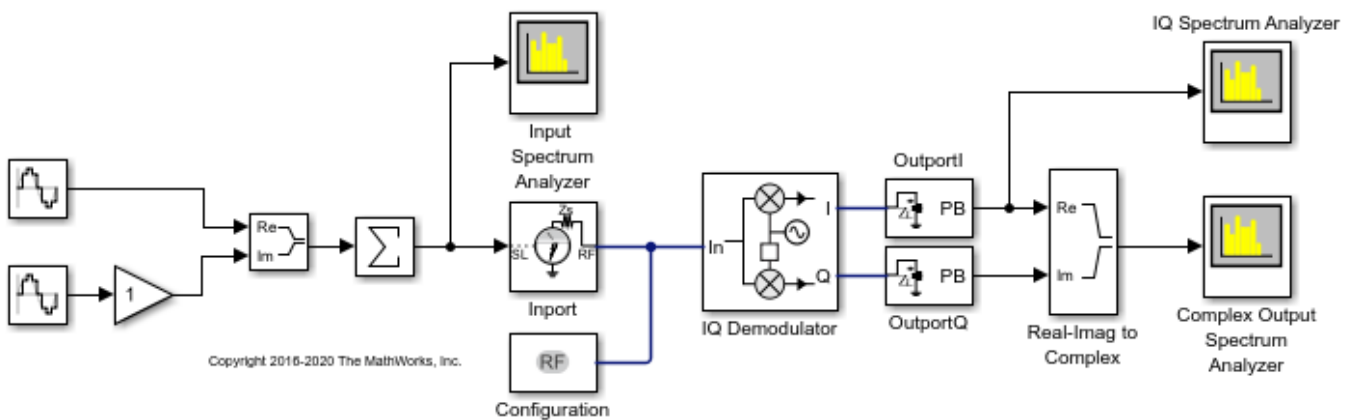
IQ Demodulator

The IQ Demodulator parameters are:

- Available power gain: 10 dB
- Local oscillator frequency: 2 GHz
- I/Q gain mismatch: 0.1 dB
- LO to RF Isolation: 90 dB
- Noise Figure: 6 dB/Hz

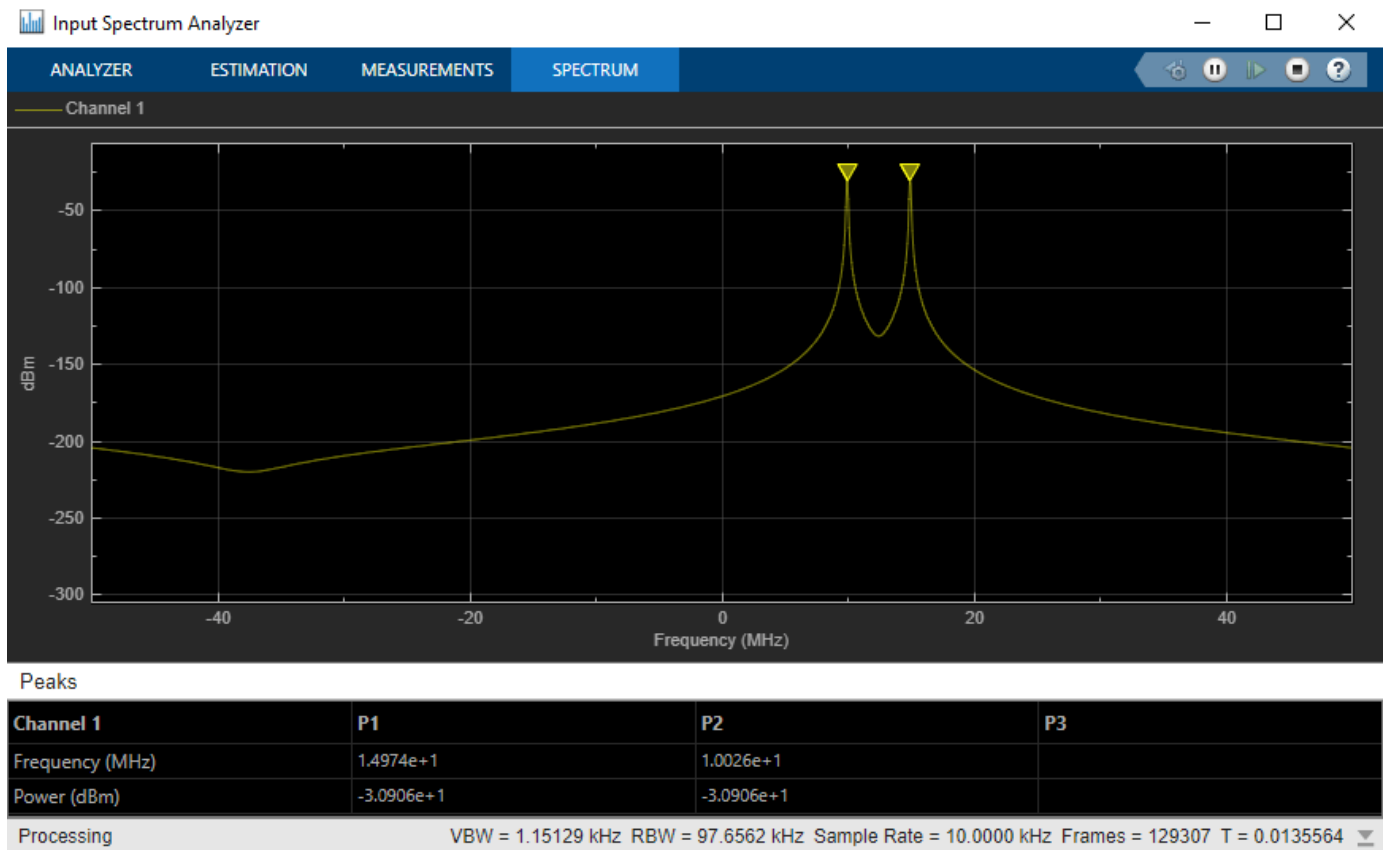
Open the model.

```
open('model_IQdemod')
```



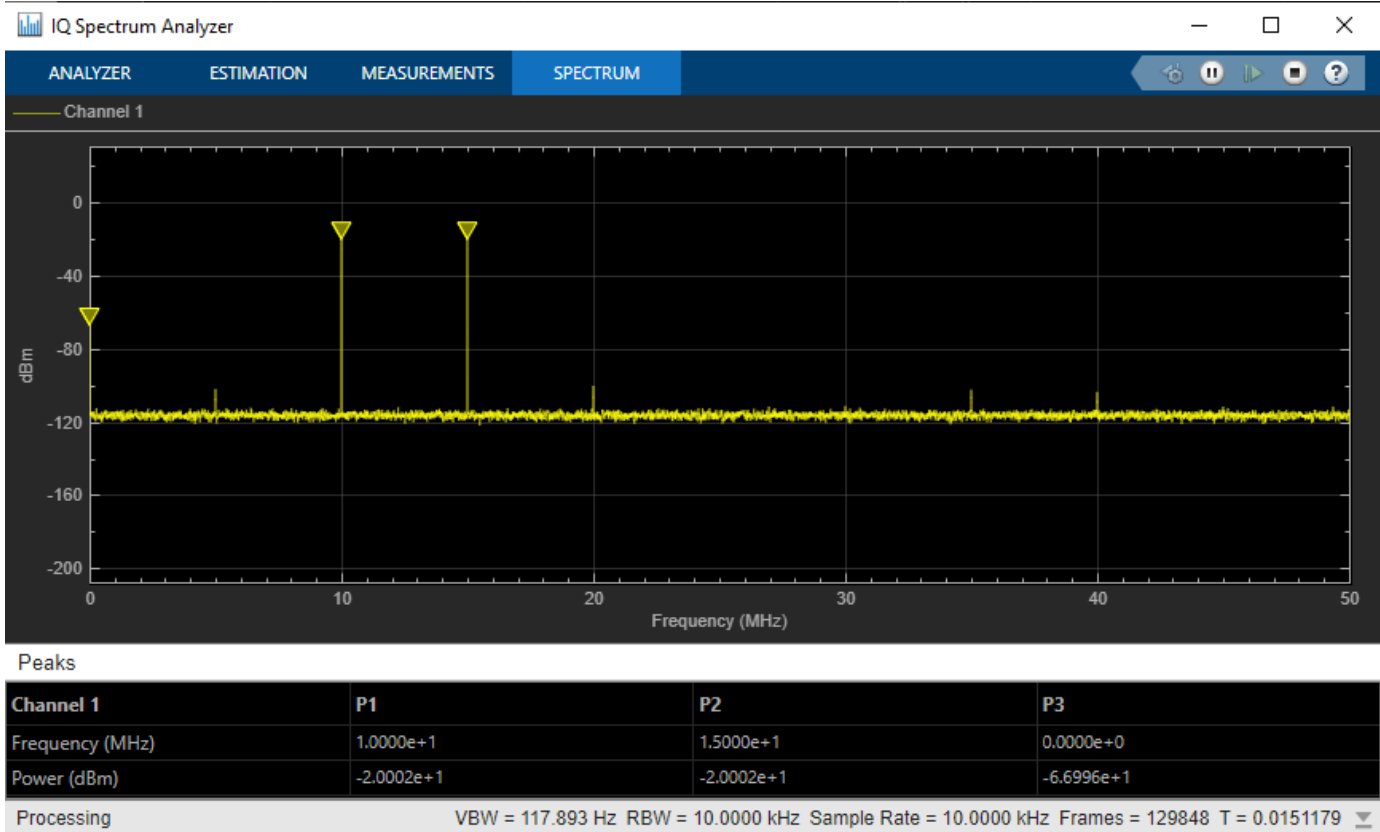
Run the model and observe the spectrum analyzers.

Input Spectrum Analyzer



In the input spectrum analyzer, you see the input RF signal with the two tones at 10 MHz and 15 MHz. The power level of each tone is -30 dBm. The carrier frequency is 2 GHz.

I/Q Spectrum Analyzer



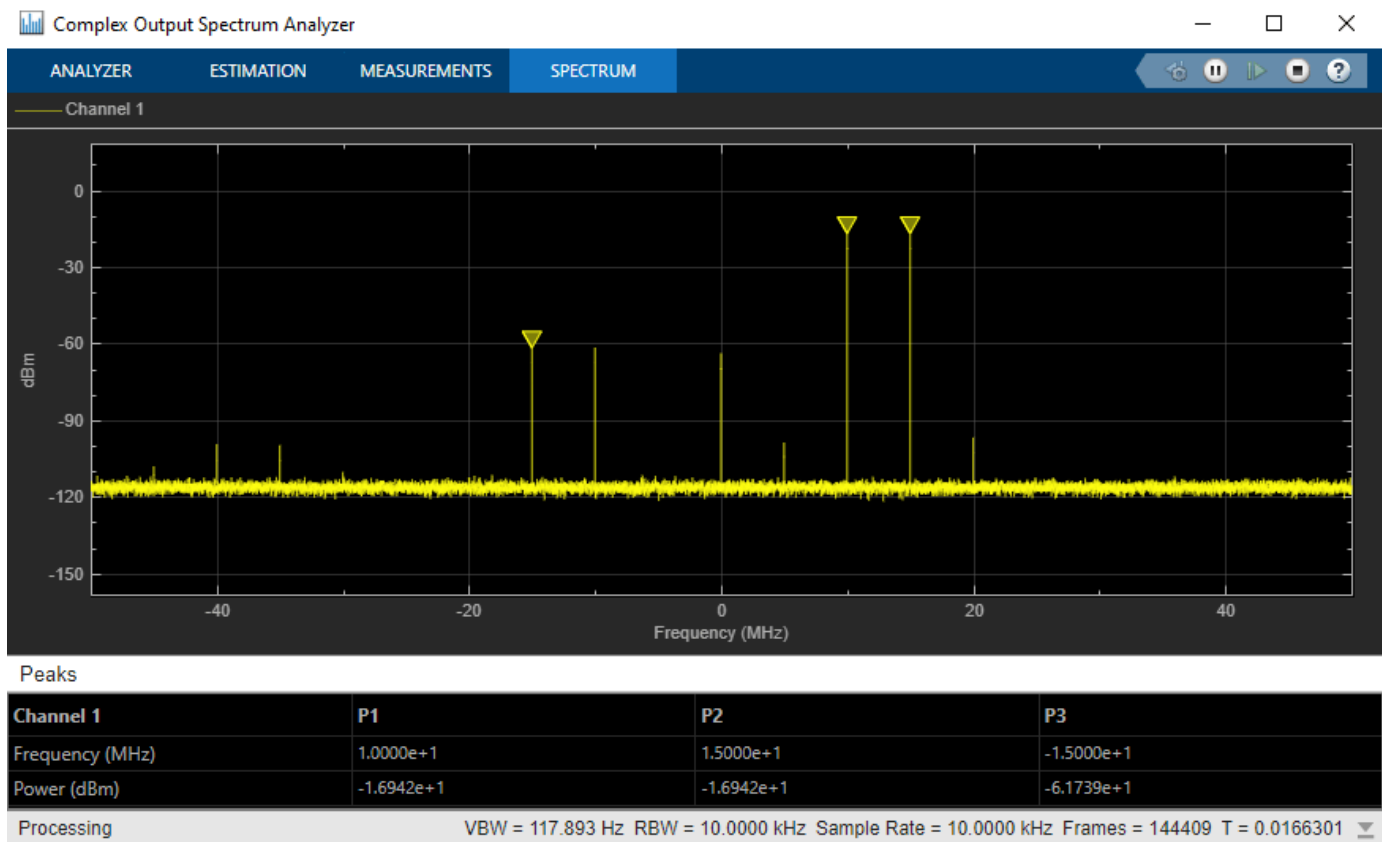
In the I/Q spectrum analyzer, you see the inphase part of the demodulated signal including the DC signal level and the two tones. The following formula gives you the DC power level of the signal:

$$DC_{level\ I/Q} = Power_{LO} - dBm_{Isolation} + Gain$$

$$DC_{level\ I/Q} = 20 * \log(1/\sqrt{50}) + 30 - 90 + 10 = -67dBm$$

The output power level of the two tones are -20 dBm. You also see the OIP3 value (measured by the spectrum analyzer) at approximately 20 dBm.

Complex Output Spectrum Analyzer



In the complex output spectrum analyzer, you see the whole demodulated signal including the imaginary parts. The output power level of the two tones (10 MHz and 15 MHz) is -17 dBm.

Image Rejection Ratio

The images of the two tones are at -10 MHz and -15 MHz. The output power level of the images are -61.78 dBm. Image rejection ratio is given by the formula:

$$IMRR = \frac{[(gain\ imbalance)^2 + 1 - 2 * (gain\ imbalance)]}{[(gain\ imbalance)^2 + 1 + 2 * (gain\ imbalance)]}$$

$$Gain\ Imbalance = 10^{(0.1/20)} = 1.0116$$

$$IMRR_{dB} = 10 * \log_{10}(3.3253e - 05) = -44.78dB$$

$$Image\ level = -17dBm - 44.78dB = -61.78dBm$$

See Also

[IQ Demodulator](#) | [Configuration](#) | [Inport](#) | [Outport](#)

Related Topics

“Modulate Two-Tone DC Signal Using IQ Modulator” on page 7-55

Modulate Two-Tone DC Signal Using IQ Modulator

Use the IQ Modulator block to Modulate a two-tone signal to RF level. Observe the impairments in the modulated output signal such as images due to gain imbalance, intermodulation distortion, and output third-order intercept (OIP3).

The two-tones are at 10 MHz and 15 MHz. The power of each tone is -30 dBm. The carrier frequency is 0 GHz.

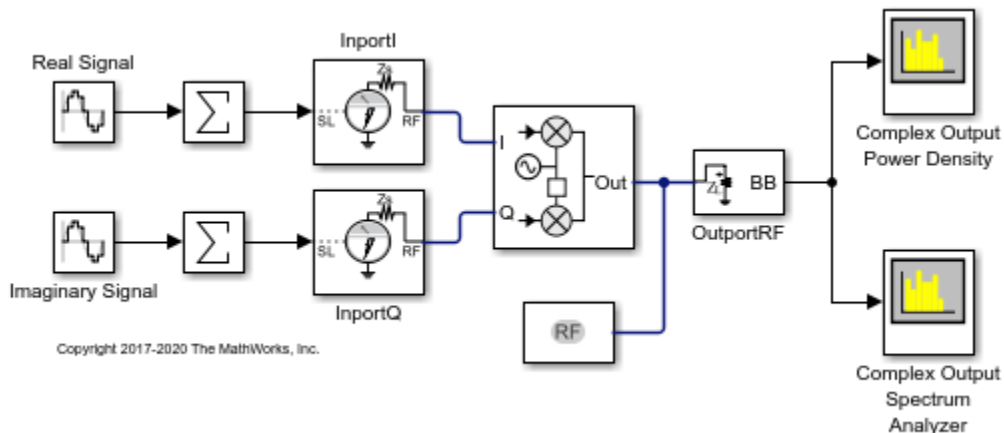
IQ Modulator

The IQ modulator parameters are :

- Available power gain: 10 dB
- Local oscillator frequency: 2 GHz
- I/Q gain mismatch: 0.1 dB
- LO to RF Isolation: 90 dB
- Noise Floor: -160 dBm/Hz
- IP3: 10 dBm

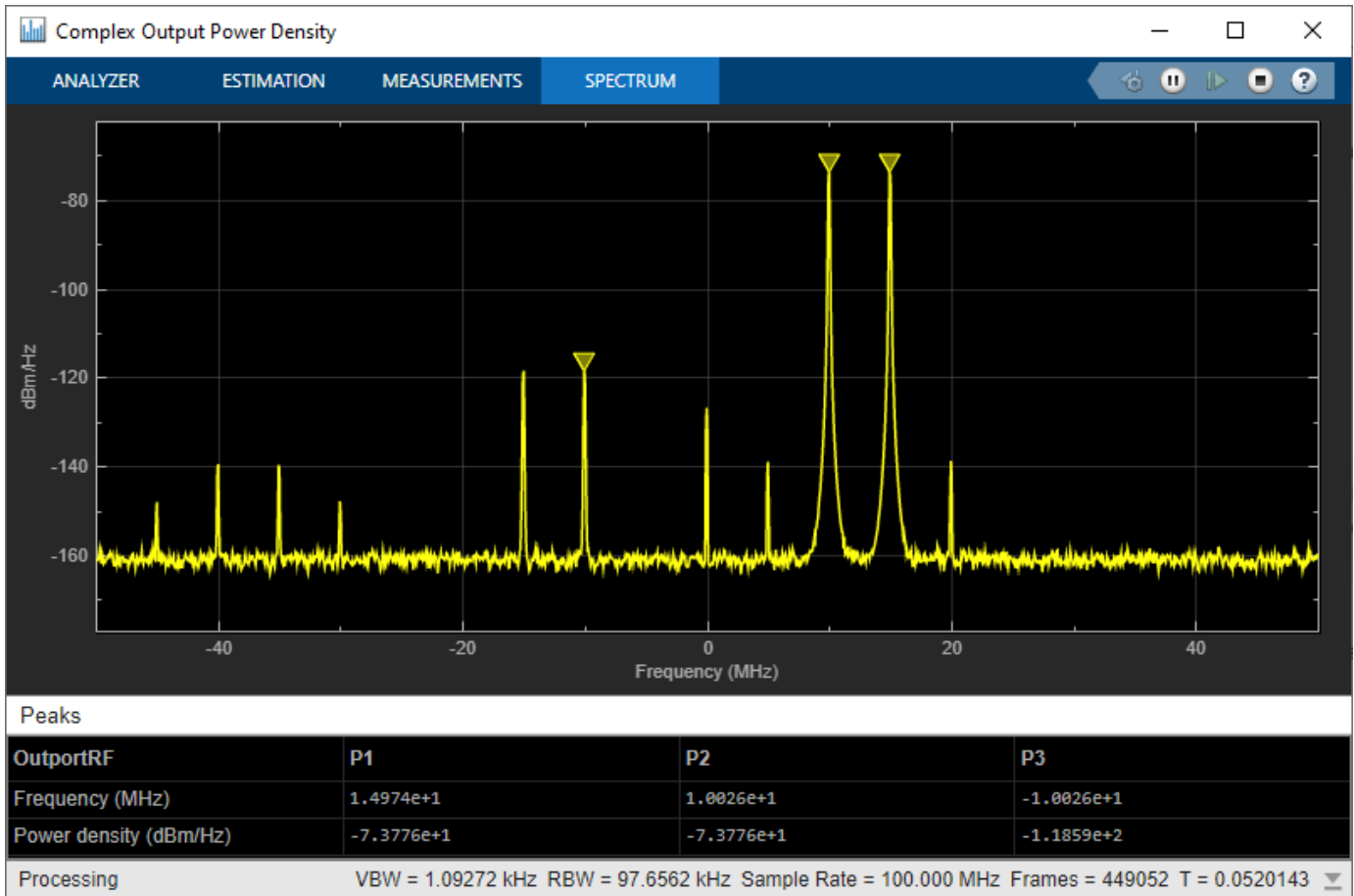
Open the model.

```
open('model_IQmod')
```



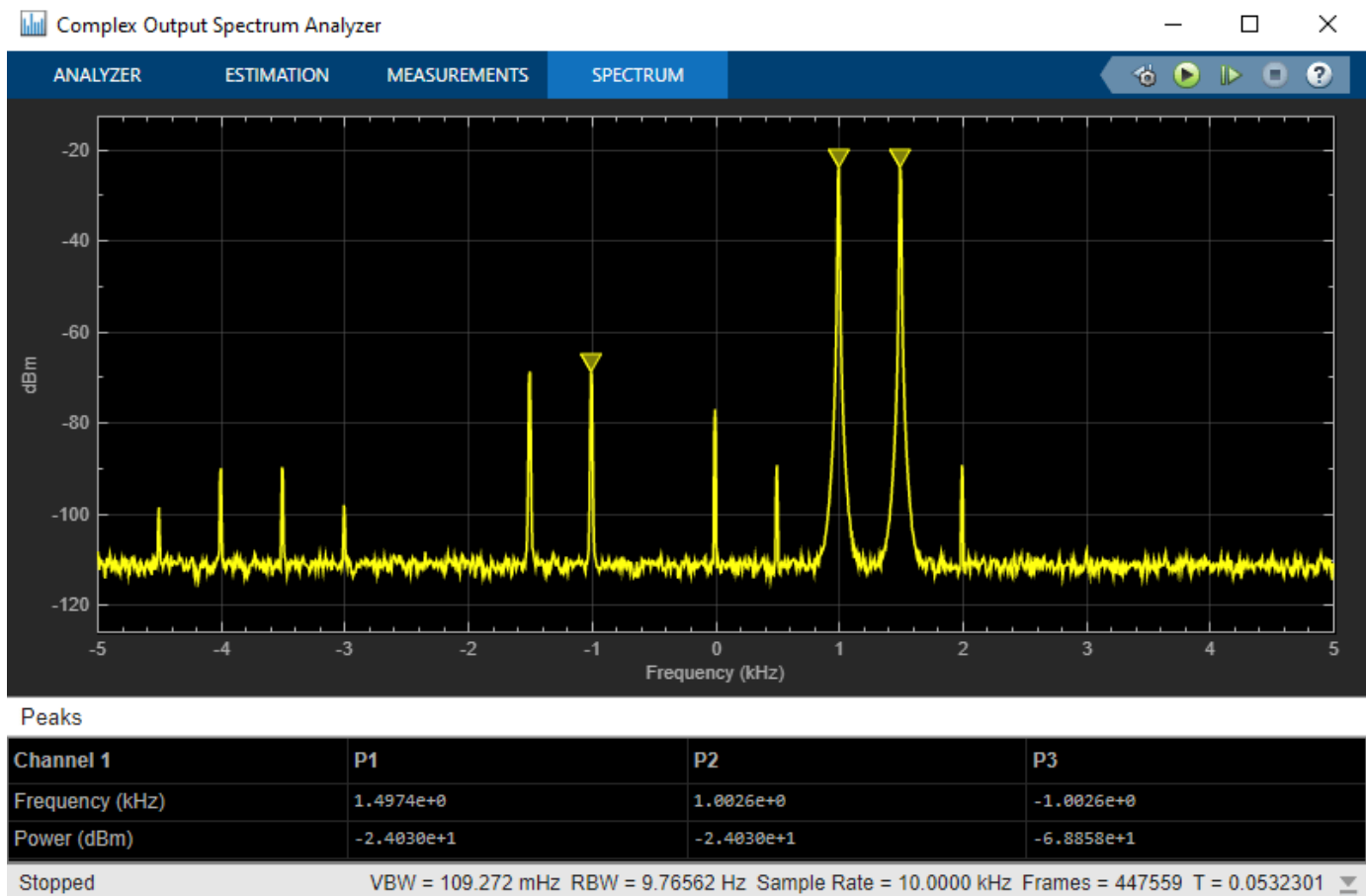
Run the model and observe the spectrum analyzers.

Complex Output Power Density



In the complex output power density spectrum analyzer, you see the noise floor of the signal at -160 dBm/Hz.

Complex Output Spectrum Analyzer



In the complex output spectrum analyzer, you see the whole modulated signal including the imaginary parts. The output power level of the two tones (10 MHz and 15 MHz) is -20 dBm.

$$\text{Output power level} = \text{Input power level} + \text{gain} = -30\text{dBm} + 10\text{dB} = -20\text{dBm}$$

The output third-order intercept (OIP3) is at 10 dBm. The spectrum analyzer measures this value.

Image Rejection Ratio

The images of the two-tones are at -10 MHz and -15 MHz. The output power level of the two images are -67.8 dBm. Image rejection ratio is calculated using:

$$\text{IMRR} = \frac{[(\text{gain imbalance})^2 + 1 - 2 * (\text{gain imbalance})]}{[(\text{gain imbalance})^2 + 1 + 2 * (\text{gain imbalance})]}$$

where,

$$\text{Gain Imbalance} = 10^{(0.1/20)} = 1.0116$$

$$\text{IMRR}_{\text{dB}} = 10 * \log_{10}(3.3253e - 05) = -44.78\text{dB}$$

The image power level is calculated using this:

$$\text{Imagelevel} = -23\text{dBm} - 44.78\text{dB} = -67.78\text{dBm}$$

See Also

[IQ Modulator](#)

Related Topics

[“Demodulate Two-Tone RF Signal Using IQ Demodulator”](#) on page 7-7

Spot Noise Data in Amplifiers and Effects on Measured Noise Figure

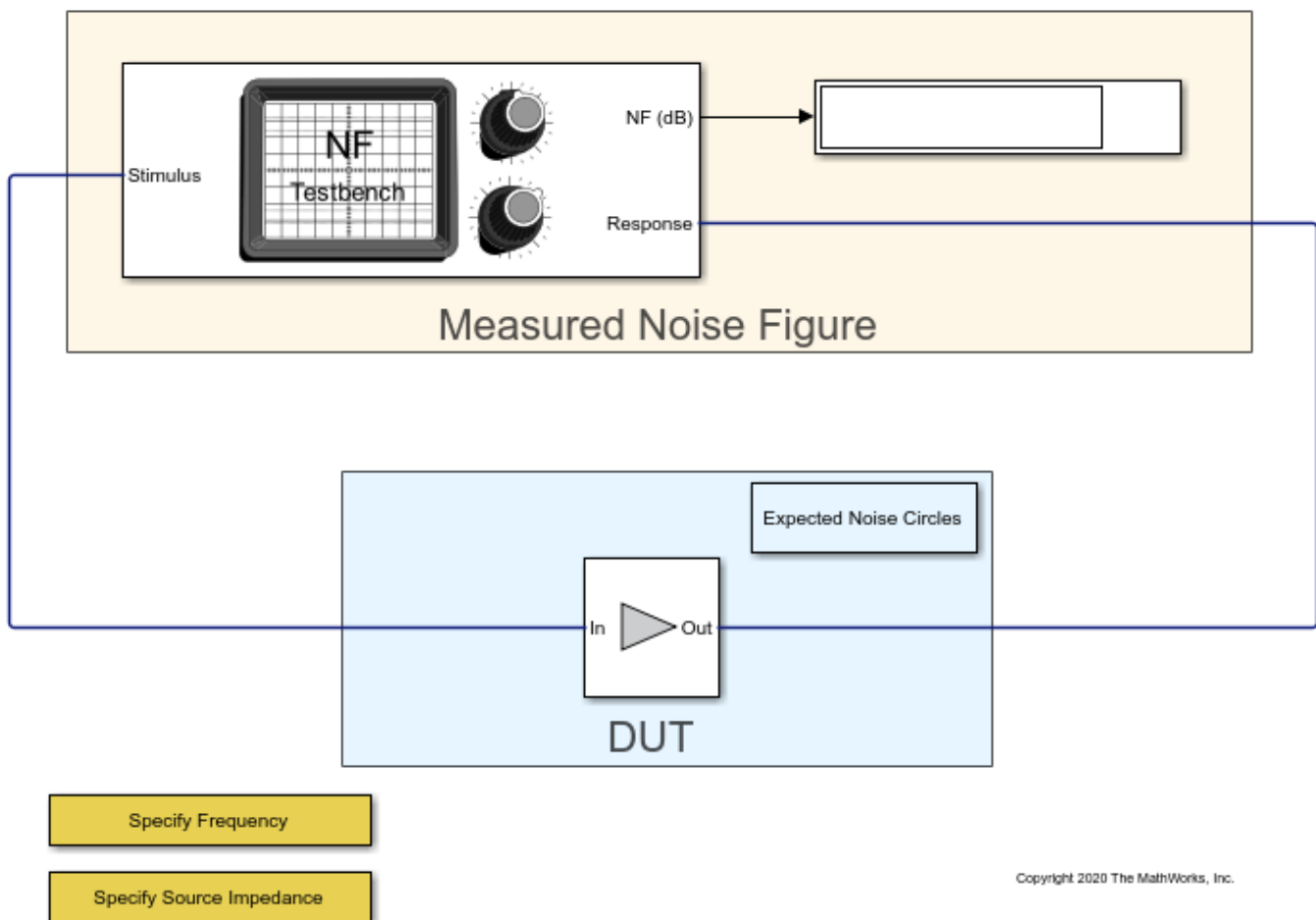
This example shows a test bench model to describe the noise introduced by a 2-port device.

The spot noise data parameters, F_{min} , Γ_{opt} , and R_n , fully describe the noise introduced by a 2-port device. These parameters along with the source impedance, Z_s uniquely determine the measured noise figure of the device. You can use noise circles plotted on a Smith chart to show interaction between Z_s and the noise figure.

Measure Noise Figure in RF Blockset

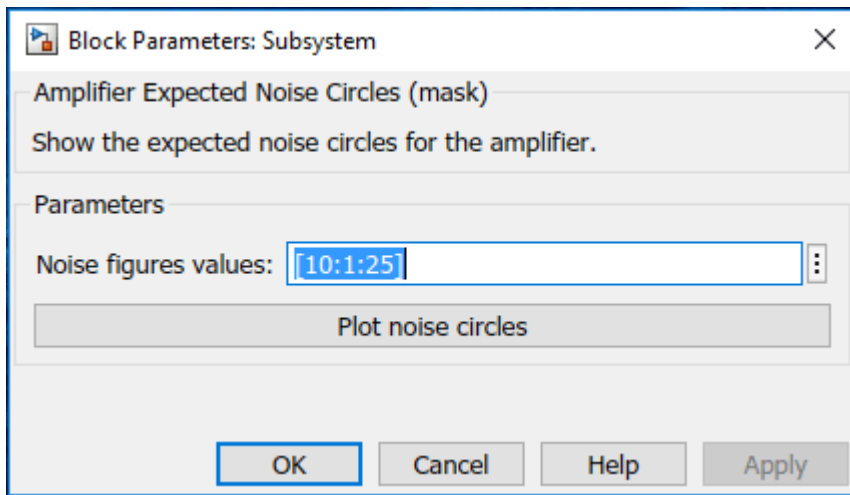
The model `Noise_figure_ex` simulates a simple noise figure measurement. In this model, the device under test comprises of a single amplifier. To open the model,

```
open_system('Noise_figure_ex.slx')
```

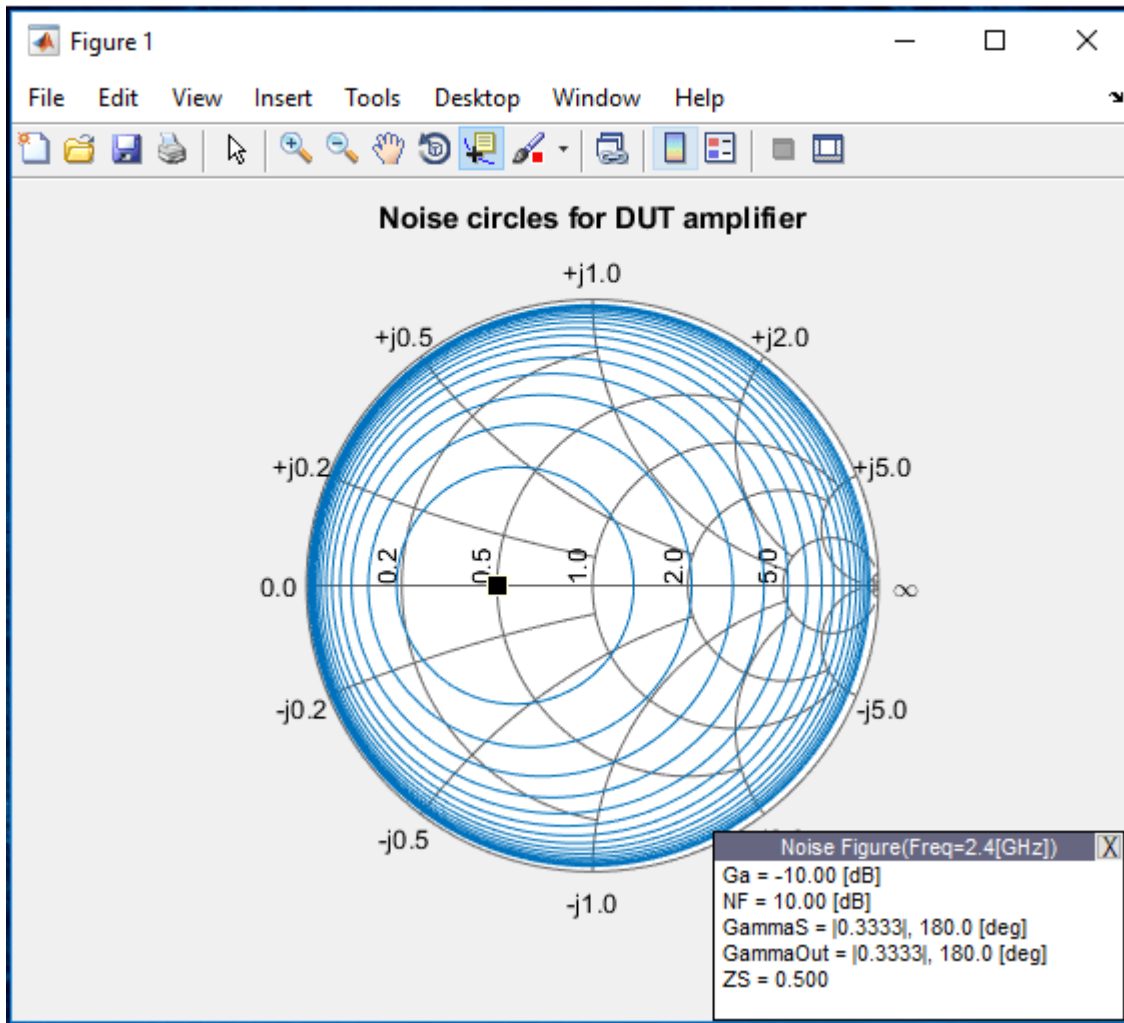


Click on the **Open Script** button before running the model. To run the model, select **Simulation > Run**.

You see that the displayed value settles down at 10.0 dB of the measured noise figure. In this testbench, the amplifier parameters represent a simple attenuator of 10 dB matched to 25 Ohms at both input and output. Due to thermal equilibrium, the expected noise figure of the attenuator is 10 dB when matched, corresponding to the measured value. To gain a broader view of the expected noise figure values for the amplifier when the source impedance deviates from the matched value of 25 Ohm, double-click the Expected Noise Circles subsystem placed in the vicinity of the amplifier:



Click **Plot noise circles** to bring up a figure showing a Smith chart with circles corresponding to the noise figure values specified above the button:



The values shown in the Smith chart represent the expected noise figures obtained theoretically from the parameters specified in the amplifier [1]. The Smith chart is interactive and you can place the data cursor on any circle to view the corresponding noise figure, source impedance (normalized to the reference impedance of the amplifier, Z_0), and other RF properties. The initial data cursor position corresponds to a point on a noise circle that is closest to the source impedance specified. To control the complex value of this source impedance, double-click the Specify Source Impedance subsystem, and specify the desired value in the edit box.

To validate that the simulated measured noise figure corresponds to theoretical values, specify a reference impedance from the Smith chart, using the Specify Source Impedance subsystem. Run the model again.

Measuring Other RF Systems

You can replace the amplifier in the model by any other RF Blockset system and measure its noise figure. In case the system is frequency depended, you can change the frequency for the

measurements by double-clicking the **Specify Frequency** subsystem and specifying the desired frequency. This frequency is also used for the amplifier noise circles plot.

The plotted expected noise circles apply to the **Amplifier** block alone. The plotted circles capture correctly all types of data inputs specified in the amplifier, including s2p based network and noise data. Note that the **Available Gain** shown in the data window of the Smith chart is based on the original data and ignores inaccuracies introduced by the amplifier modeling method. The plotting fails if a block named **Amplifier** does not exist in the model.

Using Other RF Blockset Blocks

Three additional ways to implement the attenuator specified in the amplifier are:

- 1 Use an **RF Blockset Attenuator** block with attenuation of 10dB, with input and output impedances set to 25 Ohms.
- 2 Implement the attenuator using three resistors arranged in a T or π topology.
- 3 Use an **S-parameter** block with the same Scattering matrix used in the amplifier. Select **Simulate noise** in the block. Simulating noise in a passive S-parameter block accounts for the resistive noise introduced by the S-parameters.

See Also

Noise | Noise Figure Testbench | S-Parameters | Attenuator

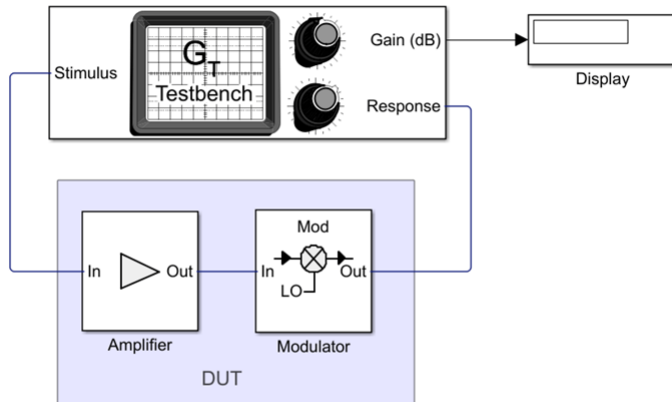
More About

- “RF Noise Modeling” on page 8-184

Measure Transducer Gain of Device Under Test

Use the Transducer Gain Testbench block to verify the gain of a device under test (DUT).

Connect the blocks as shown in the model.



Copyright 2021 The MathWorks, Inc.

Set the parameters for DUT and the testbench.

Amplifier block:

- **Available power gain** — 10 dB

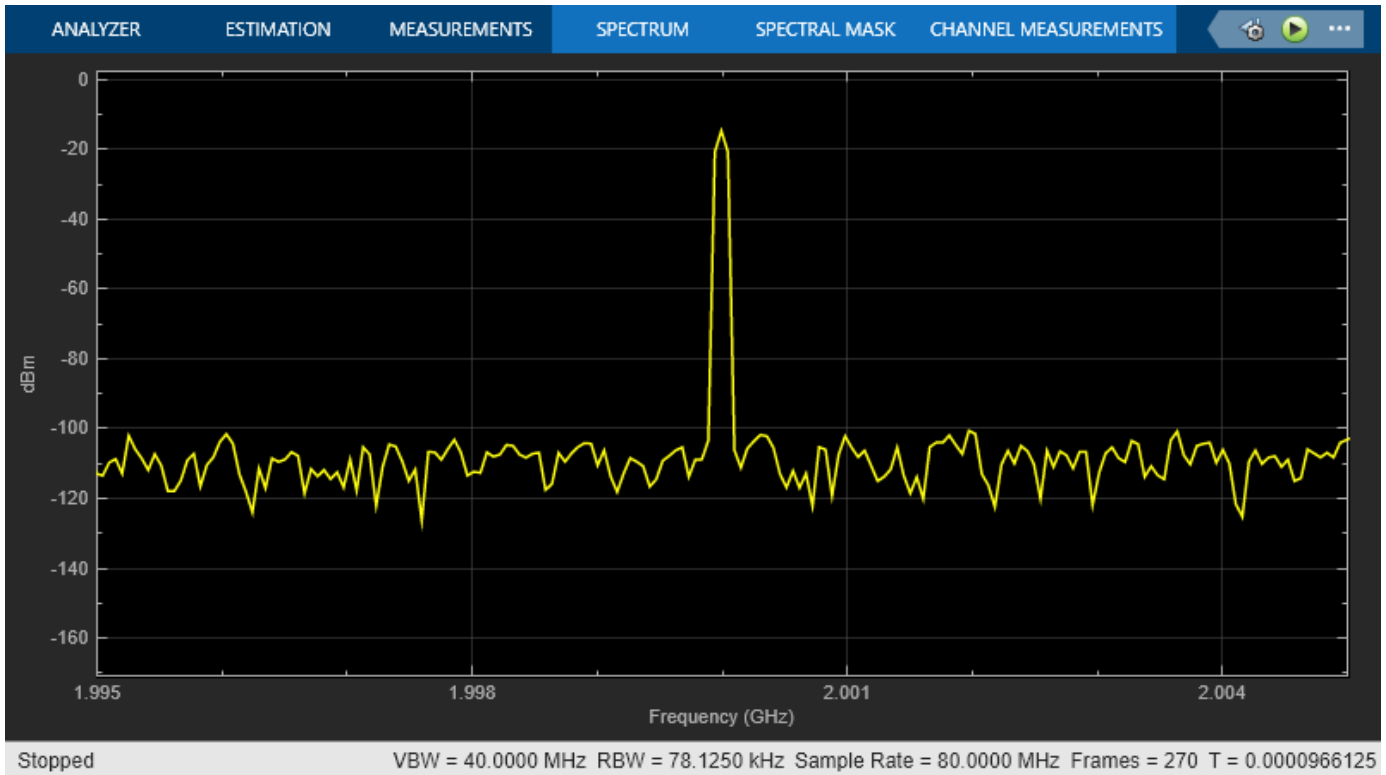
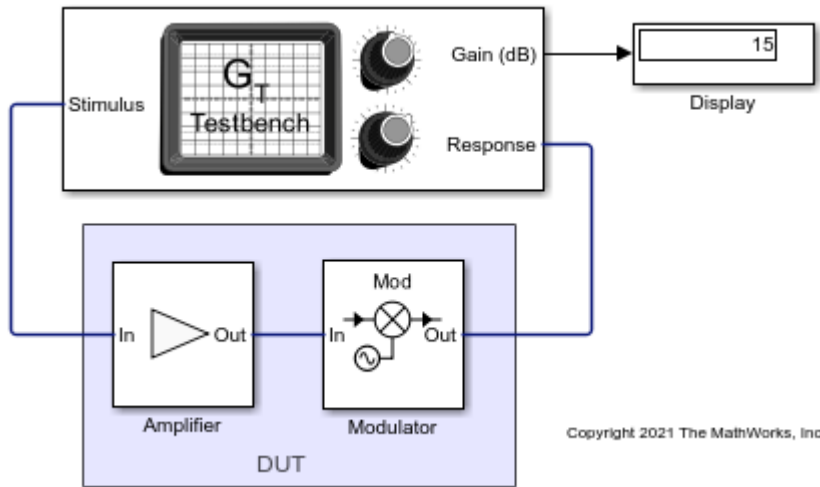
Mixer block:

- **Available power gain** — 5 dB
- **Local oscillator frequency** — 1.9 GHz

Transducer Gain Testbench:

- **Input frequency (Hz)** — 0.1e9
- **Output frequency (Hz)** — 2.0e9

Run the model. You will see that the display shows a transducer gain value of 15 dB. This can be calculated using the equation, Transducer gain of the DUT = Available power gain of Amplifier + Available power gain of Mixer = 10 dB + 5 dB = 15 dB.



See Also

Transducer Gain Testbench | Noise Figure Testbench

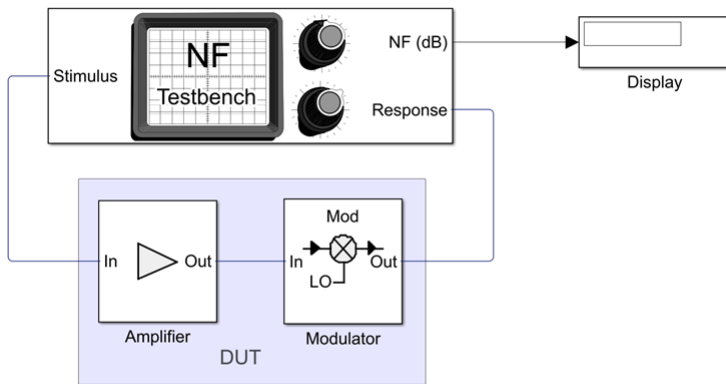
Related Examples

- “Measure Noise Figure of Device Under Test” on page 7-22

Measure Noise Figure of Device Under Test

Use the Noise Figure Testbench block to measure the noise figure of a device under test (DUT).

Connect the blocks as shown in the model.



Copyright 2021 The MathWorks, Inc.

Set the parameters for DUT and the testbench.

Amplifier block:

- **Available power gain** — 10 dB
- **Noise type** — Noise figure
- **Noise figure (dB)** — 4 dB

Mixer block:

- **Available power gain** — 5 dB
- **Local oscillator frequency** — 2.0 GHz
- **Add Image Reject filter** — on
- **Noise figure (dB)** — 8 dB
- **Filter type** — Highpass
- **Implementation** — Constant per carrier
- **Passband edge frequency** — 2.05 GHz

Noise Figure Testbench block:

- **Input frequency (Hz)** — 2.1e9
- **Output frequency (Hz)** — 0.1e9

Run the model. You will see that the display shows a OIP3 value of 4.817 dB. This value can be verified analytically using the noise figure (NF) equation provided in [1].

$$NF_{DUT} = 10 \cdot \log_{10}(F_{tot}) = 10 \cdot \log_{10}(F_1 + (F_2 - 1) / A_1)$$

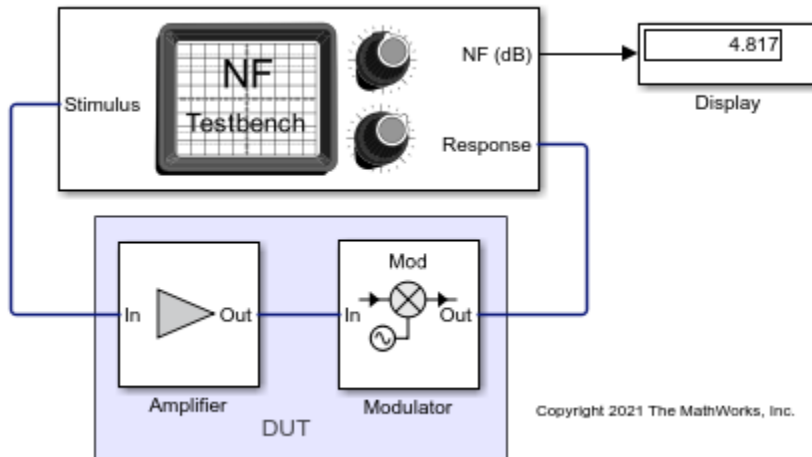
$$= 10 \cdot \log_{10} \left((10^{(4/10)}) + (10^{(8/10)} - 1) / 10^{(10/10)} \right) = 4.8328 \text{ dB}$$

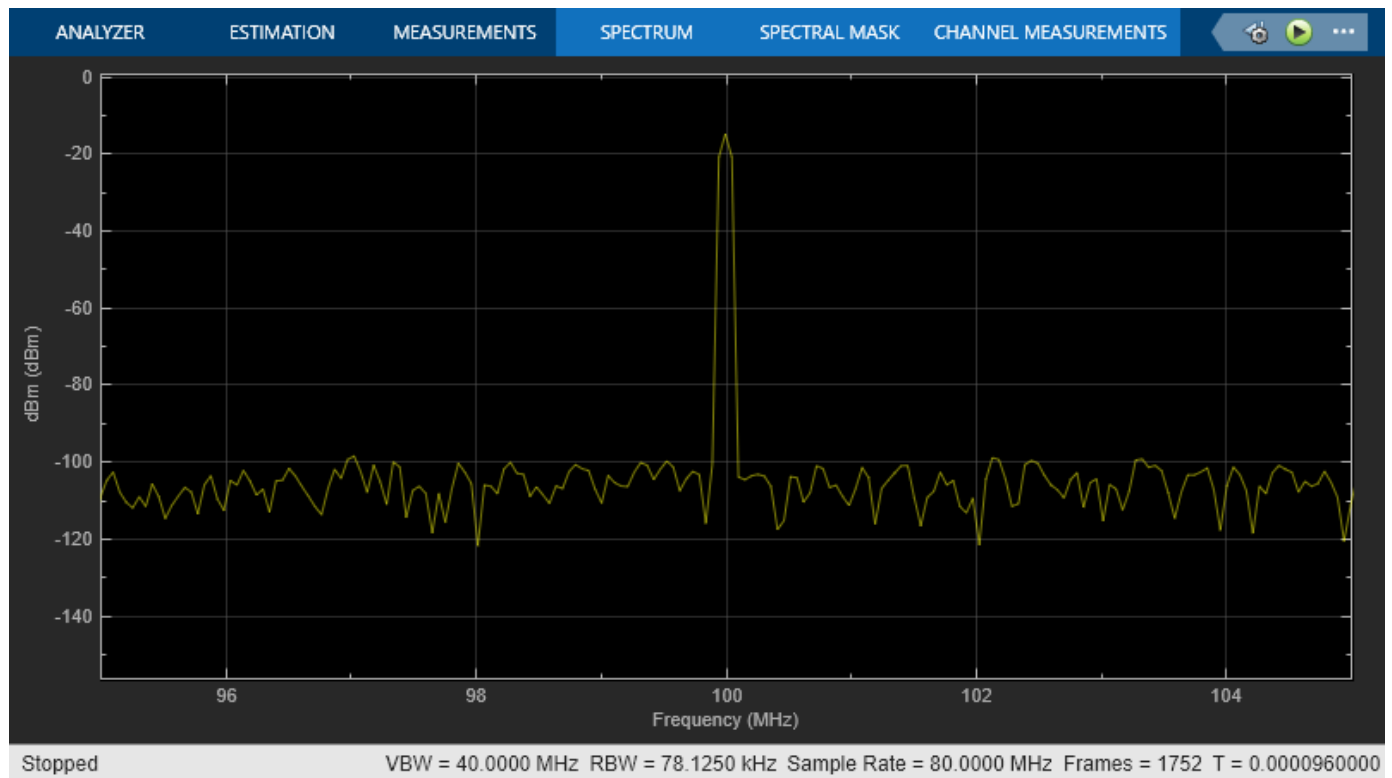
where,

Noise factor of amplifier, $F_1 = 10^{(4/10)}$

Noise factor of mixer, $F_2 = 10^{(8/10)}$

Gain of amplifier, $A_1 = 10^{(10/10)}$





References

- [1] Razavi, B.. "RF Microelectronics (2nd Edition) (Prentice Hall Communications Engineering and Emerging Technologies Series)." (2011).

See Also

Transducer Gain Testbench | Noise Figure Testbench

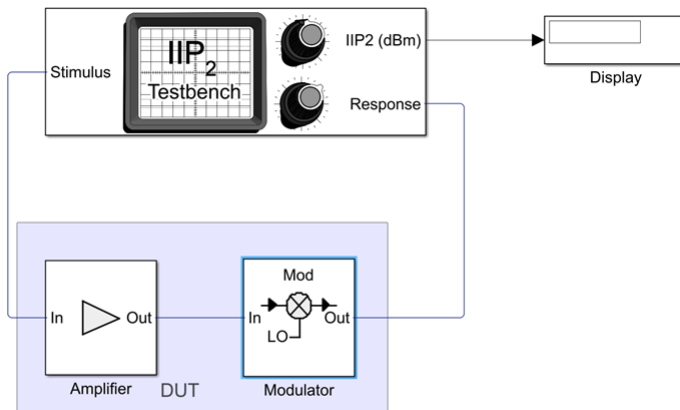
Related Examples

- "Measure Transducer Gain of Device Under Test" on page 7-20

Measure IIP2 of Device Under Test

Use the IIP2 Testbench block to verify the input second-order intercept (IIP2) of a device under test (DUT).

Connect the blocks as shown in the model.



Copyright 2021 The MathWorks, Inc.

Set the parameters for DUT and the testbench.

Amplifier block:

- **Available power gain** — 10 dB
- **Intercept points convention** — Input

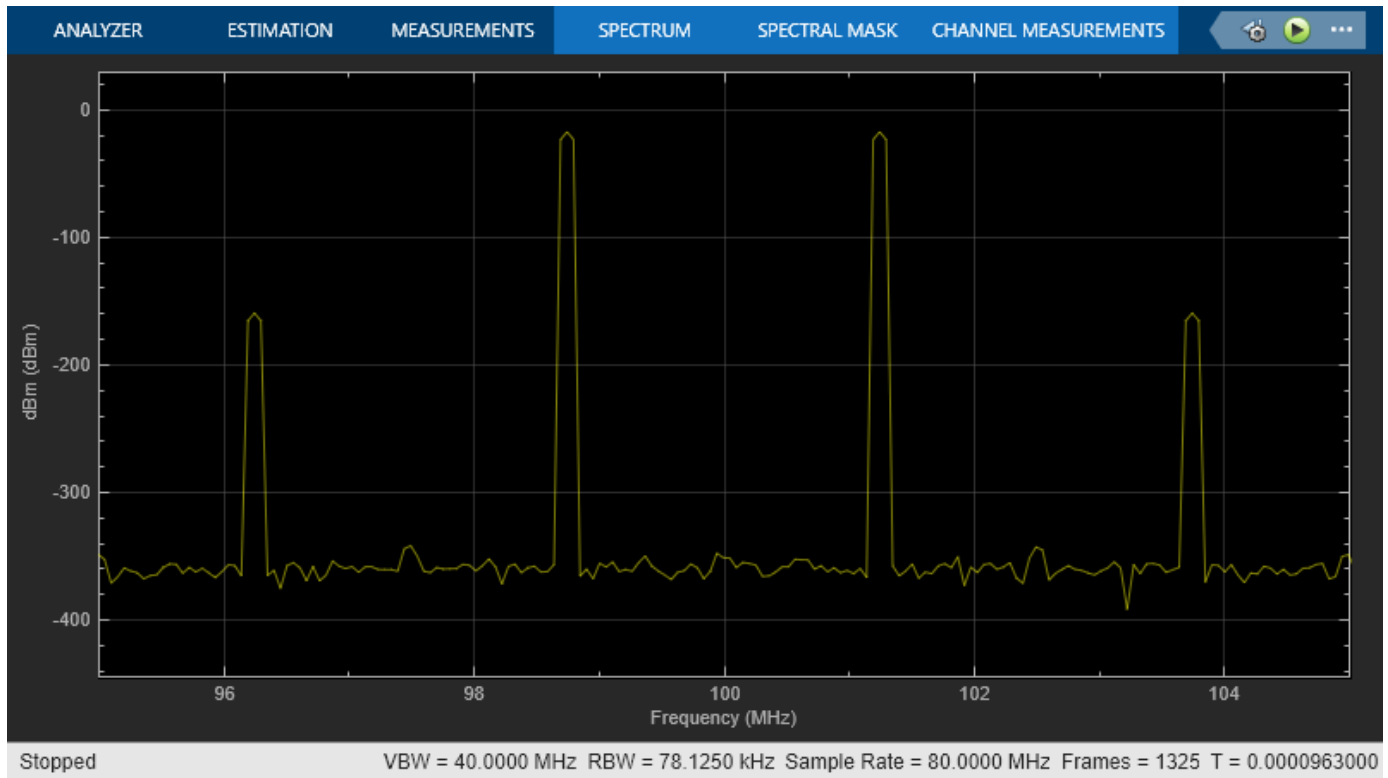
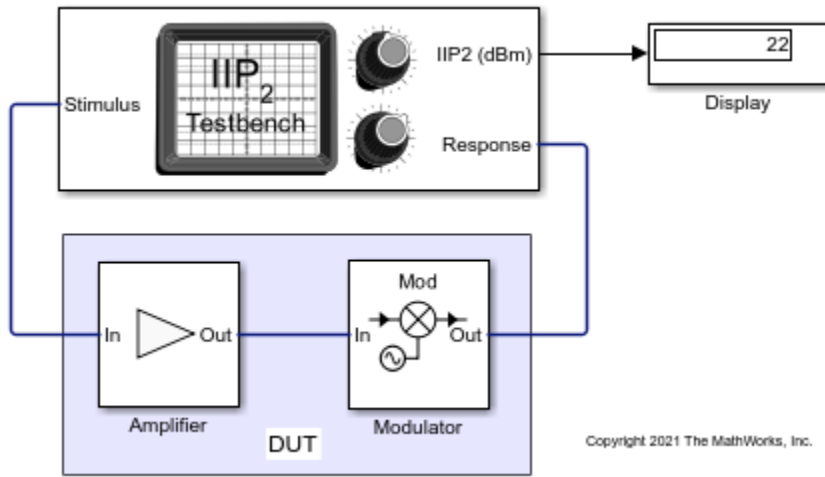
Mixer block:

- **Available power gain** — 5 dB
- **Local oscillator frequency** — 2.0 GHz
- **Add Image Reject filter** — on
- **Intercept points convention** — Input
- **IP2** — 32 dBm
- **Filter type** — Highpass
- **Implementation** — Constant per carrier
- **Passband edge frequency** — 2.05 GHz

IIP2 Testbench block:

- **Input frequency (Hz)** — 2.1e9
- **Output frequency (Hz)** — 0.1e9
- **Simulate noise (both stimulus and DUT internal)** — off

Run the model. Since the mixer is preceded by a 10 dB amplifier you will see that the display shows an IIP2 value of 22 dBm. This is calculated using the equation, $IIP2_{DUT} = IIP2 \text{ of mixer} - \text{available power gain of amplifier} = 22 \text{ dBm}$.



See Also

OIP2 Testbench | OIP3 Testbench | IIP3 Testbench | IIP2 Testbench

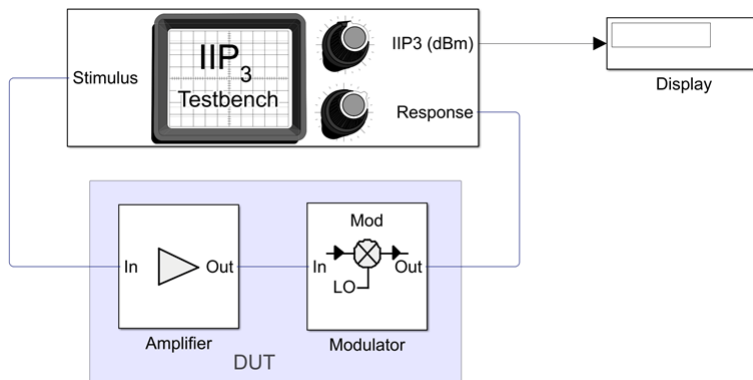
Related Examples

- “Measure IIP3 of Device Under Test” on page 7-27

Measure IIP3 of Device Under Test

Use the IIP3 Testbench block to measure the input third-order intercept (IIP3) of device under test (DUT).

Connect the blocks as shown in the model.



Copyright 2021 The MathWorks, Inc.

Set the parameters for DUT and the testbench.

Amplifier block:

- **Available power gain** — 10 dB
- **Intercept points convention** — Input
- **IP3** — 32 dBm

Mixer block:

- **Available power gain** — 5 dB
- **Local oscillator frequency** — 2.0 GHz
- **Add Image Reject filter** — on
- **Intercept points convention** — Input
- **IP3** — 35 dBm
- **Filter type** — Highpass
- **Implementation** — Constant per carrier
- **Passband edge frequency** — 2.0 GHz

IIP3 Testbench block:

- **Input frequency (Hz)** — 2.1e9
- **Output frequency (Hz)** — 0.1e9
- **Simulate noise (both stimulus and DUT internal)** — off

Run the model. You will see that the display shows an IIP3 value of 24.19 dBm. This value can be verified analytically using the equation provided in [1].

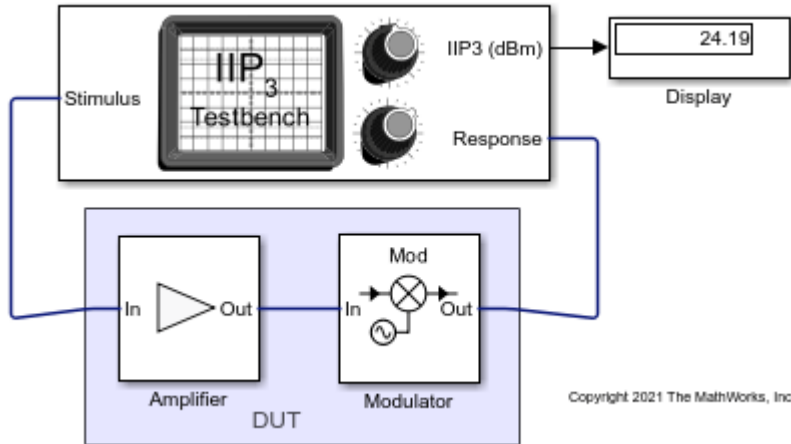
IIP3 of the DUT = $-10 \cdot \log_{10}(1/10^{((32)/10)} + 10^{(10/10)}/10^{((35)/10)}) = 24.2099 \text{ dBm}$

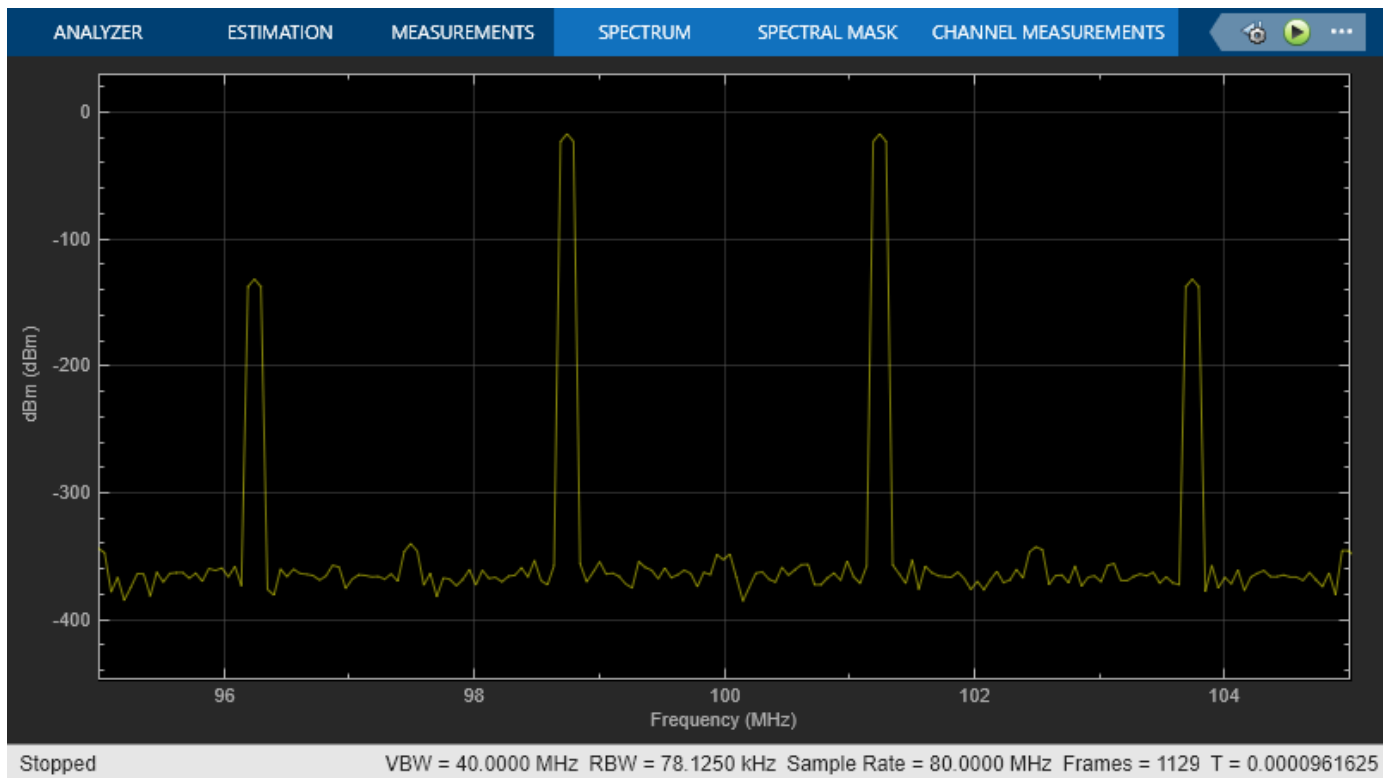
where,

IIP3 of the amplifier in linear scale = $10^{((32)/10)}$

IIP3 of the mixer in linear scale = $10^{((35)/10)}$

Available power gain of the amplifier = $10^{(10/10)}$





References

[1] Razavi, Behzad. "Basic Concepts " in *RF Microelectronics*, 2nd edition, Prentice Hall, 2012.

See Also

OIP2 Testbench | OIP3 Testbench | IIP3 Testbench | IIP2 Testbench

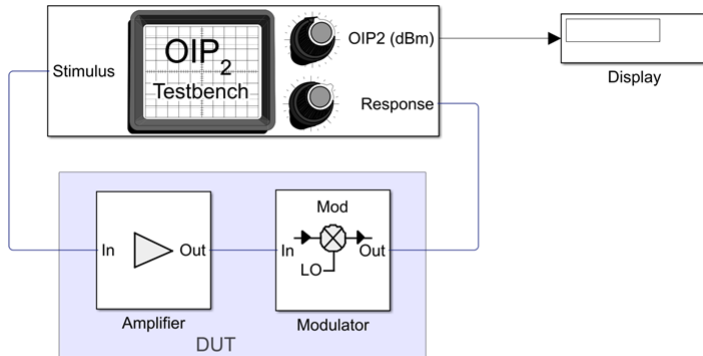
Related Examples

- "Measure OIP3 of Device Under Test" on page 7-32

Measure OIP2 of Device Under Test

Use the OIP2 Testbench block to verify the output second-order intercept (OIP2) of a device under test (DUT).

Connect the blocks as shown in the model.



Copyright 2021 The MathWorks, Inc

Set the parameters for DUT and the testbench.

Amplifier block:

- **Available power gain** — 10 dB
- **Intercept points convention** — Input

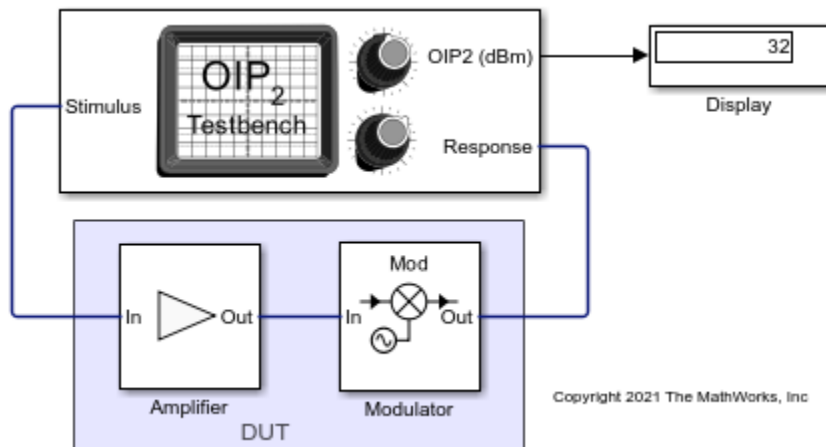
Mixer block:

- **Available power gain** — 5 dB
- **Local oscillator frequency** — 2.0 GHz
- **Add Image Reject filter** — on
- **Intercept points convention** — Output
- **IP2** — 32 dBm
- **Filter type** — Highpass
- **Implementation** — Constant per carrier
- **Passband edge frequency** — 2.05 GHz

OIP2 Testbench block:

- **Input frequency (Hz)** — 2.1e9
- **Output frequency (Hz)** — 0.1e9
- **Simulate noise (both stimulus and DUT internal)** — off

Run the model. You will see that the display shows an OIP2 value of 32 dBm since the OIP2 was specified only in the Mixer block.



See Also

OIP2 Testbench | OIP3 Testbench | IIP3 Testbench | IIP2 Testbench

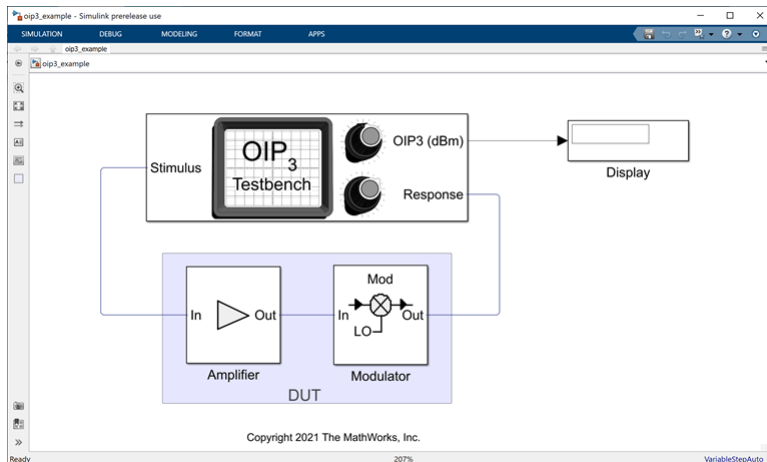
Related Examples

- “Measure OIP3 of Device Under Test” on page 7-32

Measure OIP3 of Device Under Test

Use the OIP3 Testbench block to measure the output third order intercept (OIP3) of a device under test (DUT).

Connect the blocks as shown in the model.



Set the parameters for DUT and the testbench.

Amplifier block:

- **Available power gain** — 10 dB
- **Intercept points convention** - Output
- **IP3** - 32 dBm

Mixer block:

- **Available power gain** — 5 dB
- **Local oscillator frequency** — 2.0 GHz
- **Add Image Reject filter** — on
- **Intercept points convention** — Output
- **IP3** — 35 dBm
- **Filter type** — Highpass
- **Implementation** — Constant per carrier
- **Passband edge frequency** — 2.05 GHz

OIP3 Testbench block:

- **Input frequency (Hz)** — 2.1e9
- **Output frequency (Hz)** — 0.1e9
- **Simulate noise (both stimulus and DUT internal)** — off

Run the model. You will see that the Display block shows an OIP3 value of 32.87 dBm. This value can be verified analytically using the IIP3 equation provided in [1].

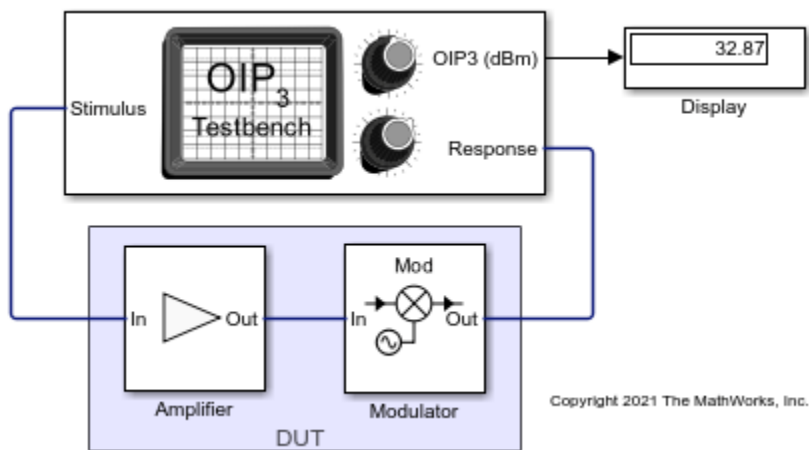
OIP3 of the DUT = IIP3 + Gain = $-10 \cdot \log_{10}(1/10^{((22)/10)} + 10^{(10/10)}/10^{((30)/10)}) + 15 = 32.8756 \text{ dBm}$

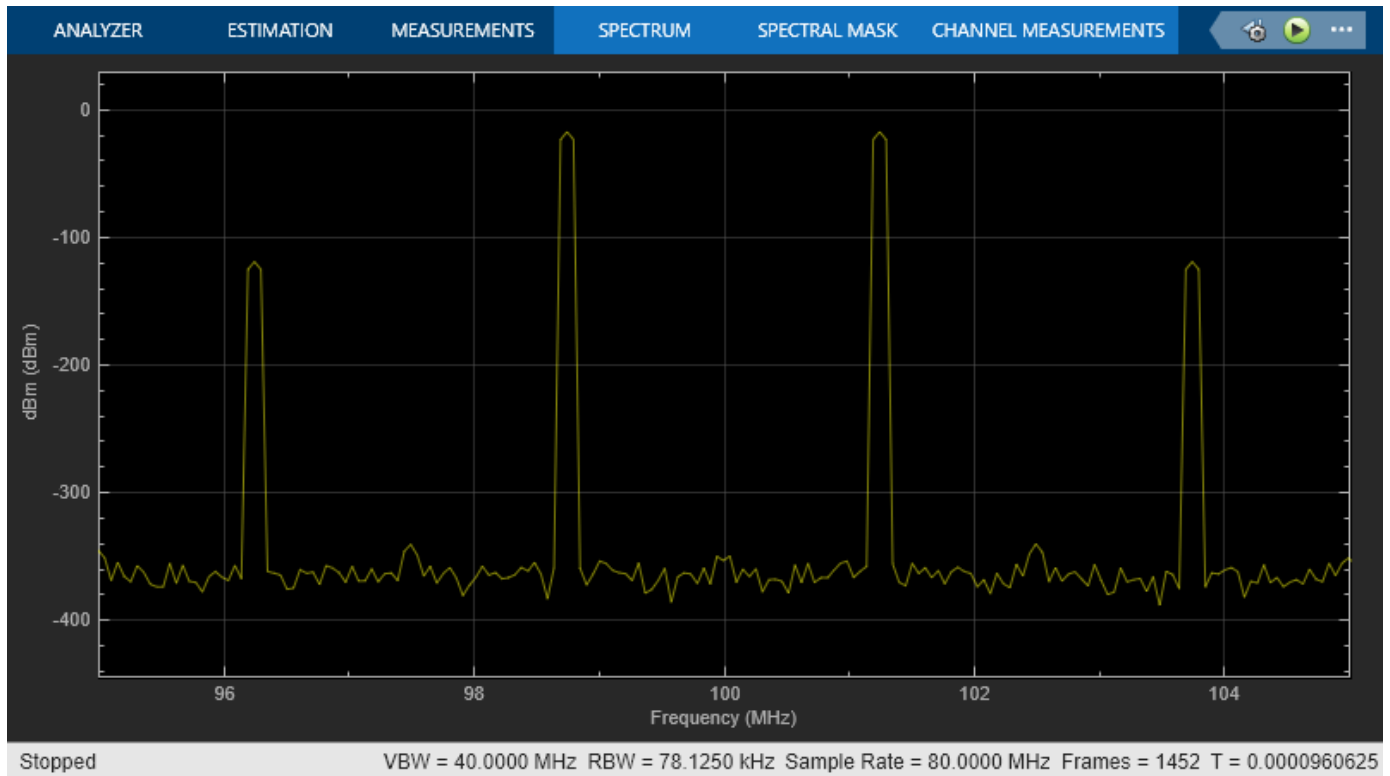
where,

IIP3 of the amplifier in linear scale = $10^{((32-10)/10)}$, as IIP3 = OIP3 - Gain ; IIP3 = 32 dBm - 10 dB = 22 dBm

IIP3 of the mixer in linear scale = $10^{((35-5)/10)}$, as IIP3 = OIP3 - Gain ; IIP3 = 35 dBm - 5 dB = 30 dBm

Available power gain of the amplifier = $10^{(10/10)}$





References

[1] Razavi, Behzad. "Basic Concepts " in *RF Microelectronics*, 2nd edition, Prentice Hall, 2012.

See Also

OIP2 Testbench | OIP3 Testbench | IIP3 Testbench | IIP2 Testbench

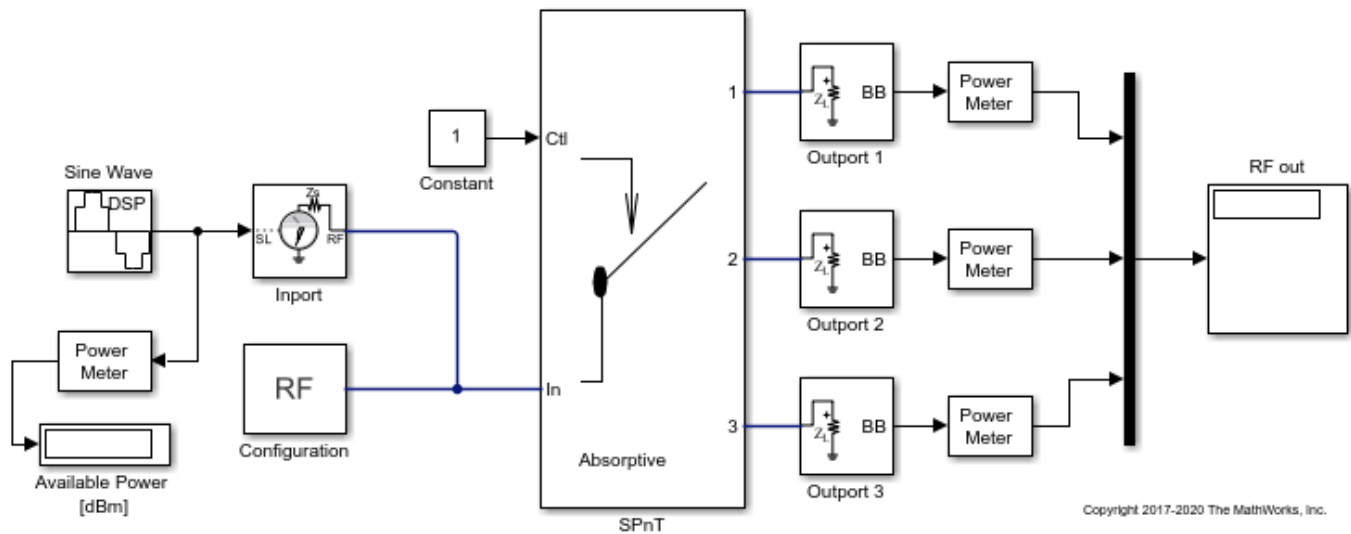
Related Examples

- "Measure IIP3 of Device Under Test" on page 7-27

Single Pole Triple Throw Switch

Use the SPnT block to create a single pole triple throw switch to switch a signal between three outputs.

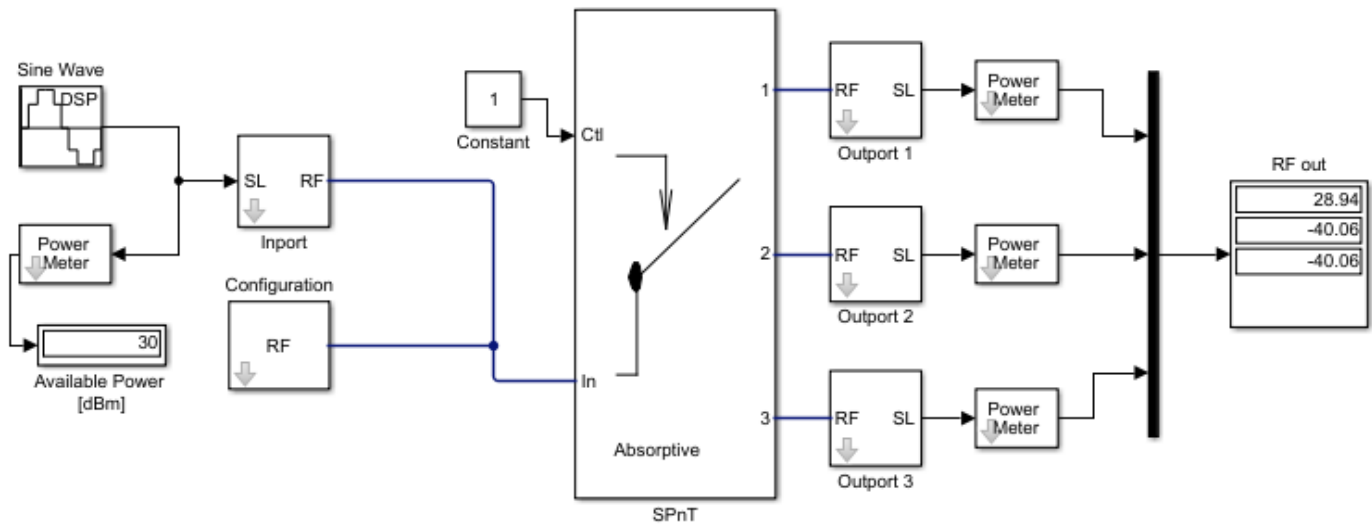
Open the model.



The model consists of:

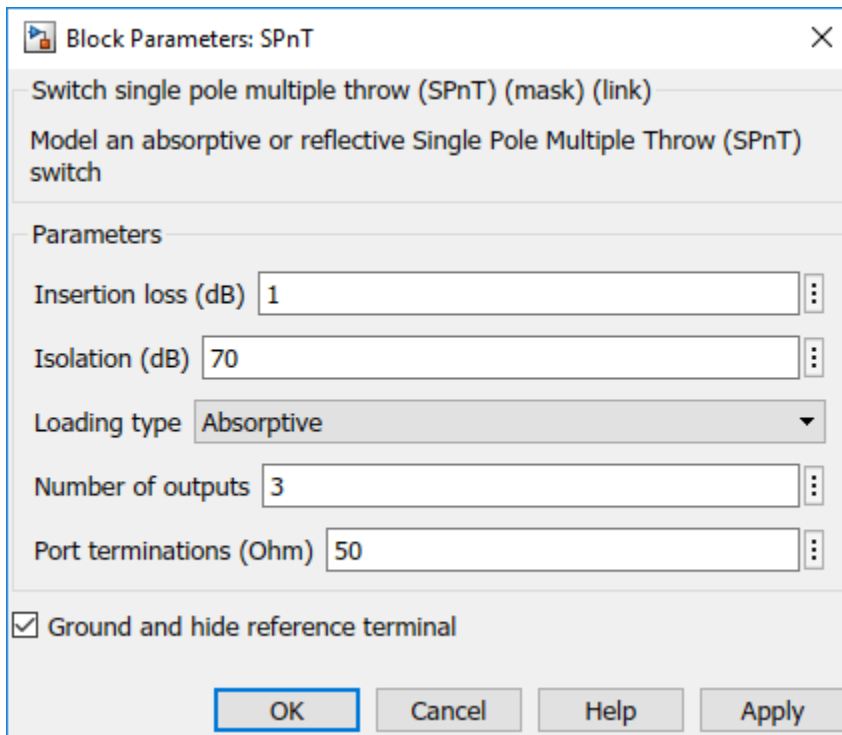
- Sine Wave block to generate a sine wave of amplitude 1.
- Power Meter block to determine the power of the sine wave. This is the input power. The input power is 30 dB.
- Inport and Configuration blocks connected to the "In" port of the SPnT block.
- A Constant block connected to the "Ctl" port of the SPnT block. This signal is used to control the switch outputs.
- SPnT switch block with 3 output ports.
- Output 1, Output 2, Output 3, and Power meter blocks to calculate the power of each output signal.
- Display block to display the three outputs.

Run the model.



The Display block shows that the signal power is available through the first port of the switch as the "Ctl" port is set to 1.

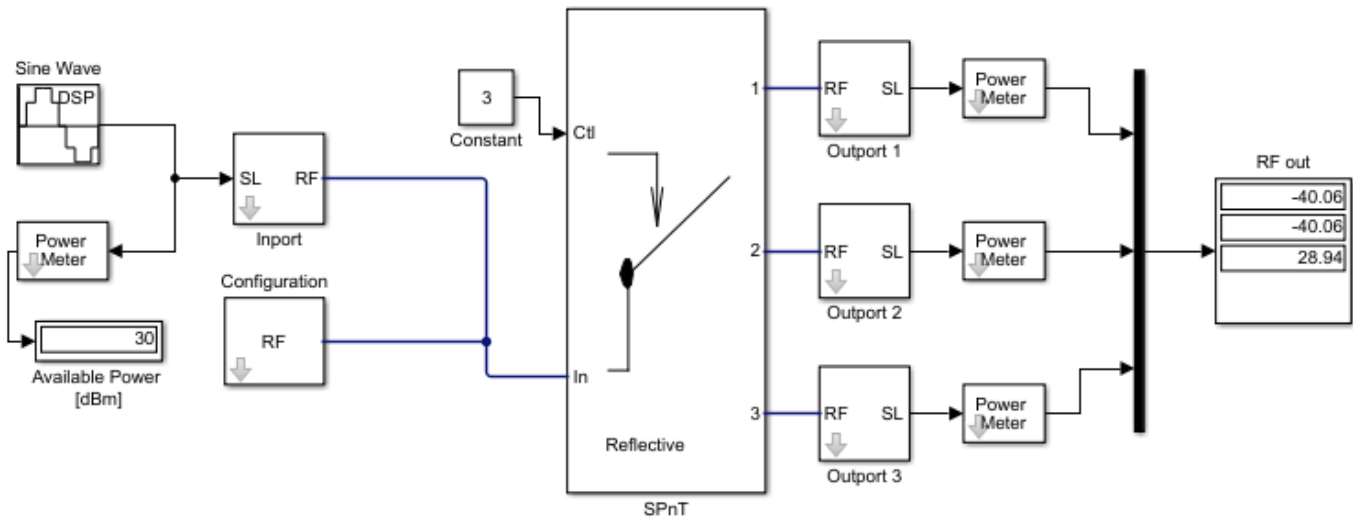
Open the SPnT block to see set values. Currently the switch is set to **Absorptive** using the **Load Type** parameter.



Change the **Load Type** value to **Reflective**.

Change the value of the Constant block to 3.

Run the model again.



The Display block shows that the signal power is available through the third port of the switch as the "Ctl" port is set to 3 .

See Also

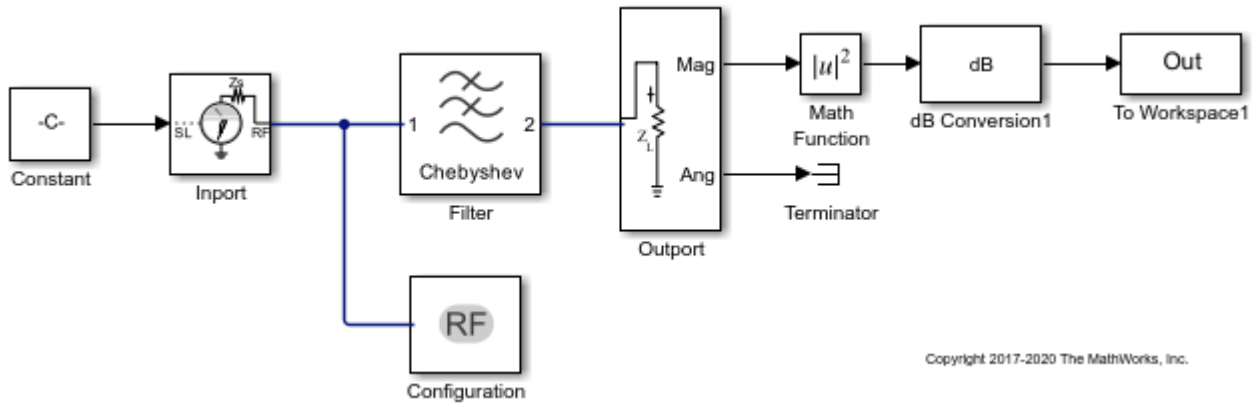
Inport | Output | SPnT

Frequency Response of Lowpass Chebyshev Filter

Use the Filter block to study the frequency response of a lowpass Chebyshev filter.

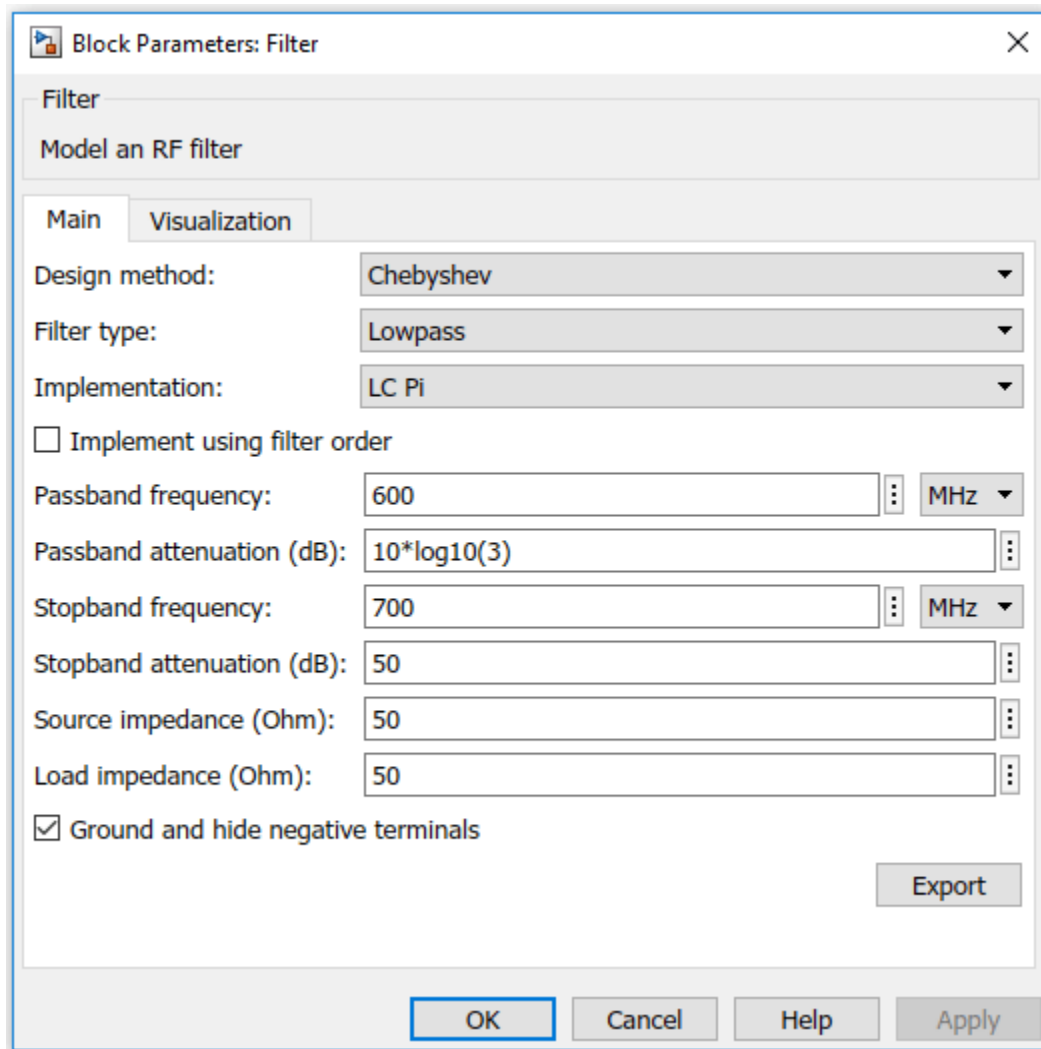
From the MATLAB command prompt, open the model.

```
open_system('ex_simrf_filter_lowpass_cheby_resp')
```



Copyright 2017-2020 The MathWorks, Inc.

The Constant block sets the amplitude of the 201 carrier signals to ones (1, 201). The Inport block generates the 201 carrier frequencies for the mask value of logspace (7, 9, 201). Generate an 11th order lowpass LC Pi Chebyshev filter by setting appropriate block parameters in the Filter block.



The output signal from the Filter block is fed into the Outport block. The Outport block is configured to give both magnitude and angle of the signal. The angle output is terminated using the Terminator block. The magnitude output is squared and converted to dB using Math Function and dB Conversion blocks.

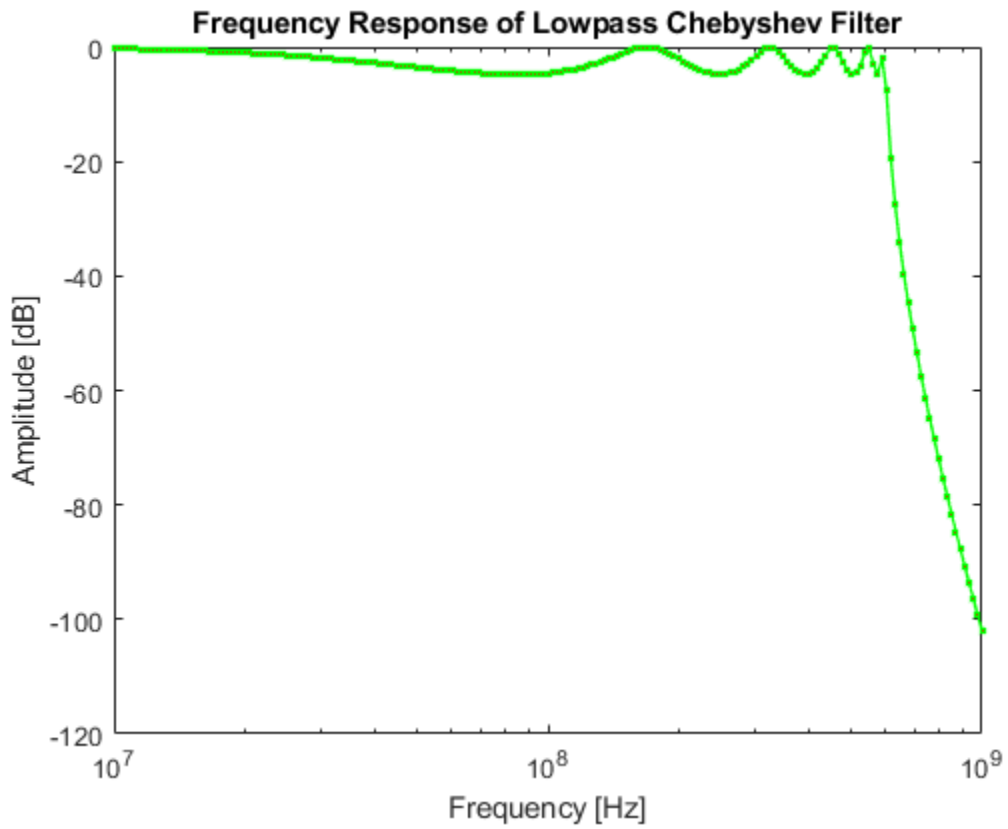
To run the model, select Simulation > Run. You can also use the following command:

```
sim('ex_simrf_filter_lowpass_cheby_resp')
```

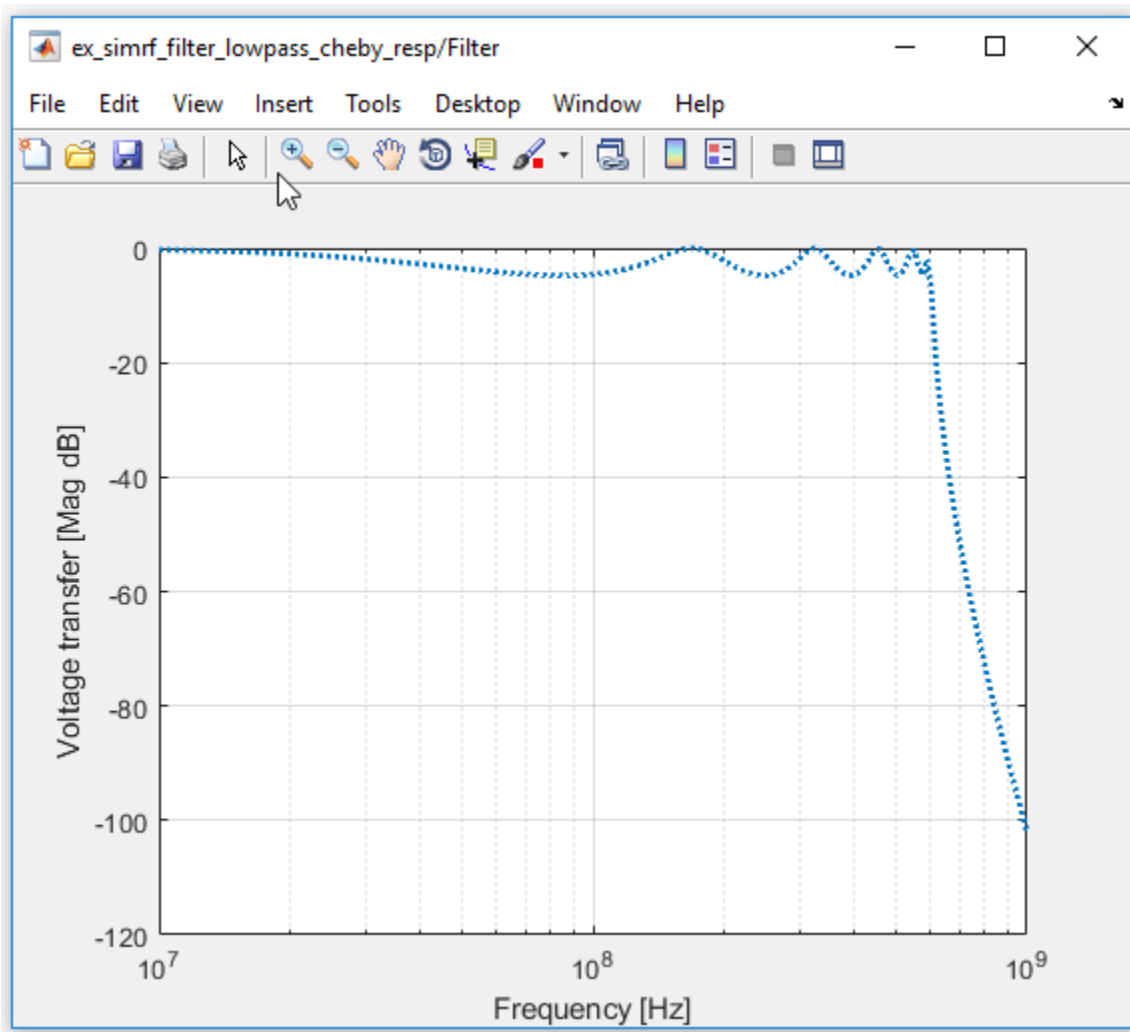
The model creates an Out array in the MATLAB workspace. Since the simulation stop time is set to 0, the frequency response corresponds to the steady state solution.

To plot the frequency response, use the following commands in the MATLAB command window.

```
figure
freq = logspace(7,9,201);
h = semilogx(freq, Out, '-gs', 'LineWidth',1, 'MarkerSize',3, 'MarkerFaceColor','r');
xlabel('Frequency [Hz]');
ylabel('Amplitude [dB]');
title('Frequency Response of Lowpass Chebyshev Filter');
```



You can also use the Plot button in the Visualization tab of the Filter block parameters. Set the Frequency points to `logspace(7, 9, 201)` and X-axis scale to Logarithmic to achieve a similar plot.

**See Also**

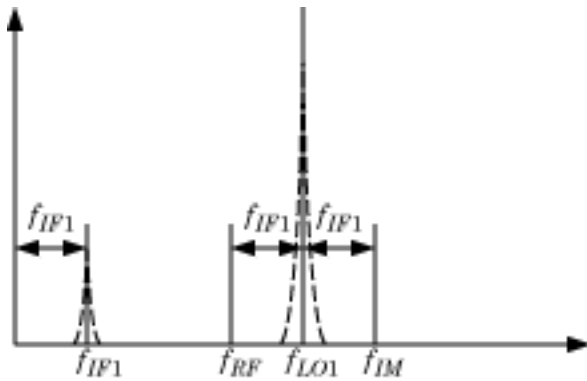
Filter | Inport | Outport | Configuration

Related Topics

"Model RF Filter Using Circuit Envelope"

Model LO Phase Noise

This example shows how to model and visualize LO phase noise. A mixer transfers local oscillator (LO) phase noise directly to its output.

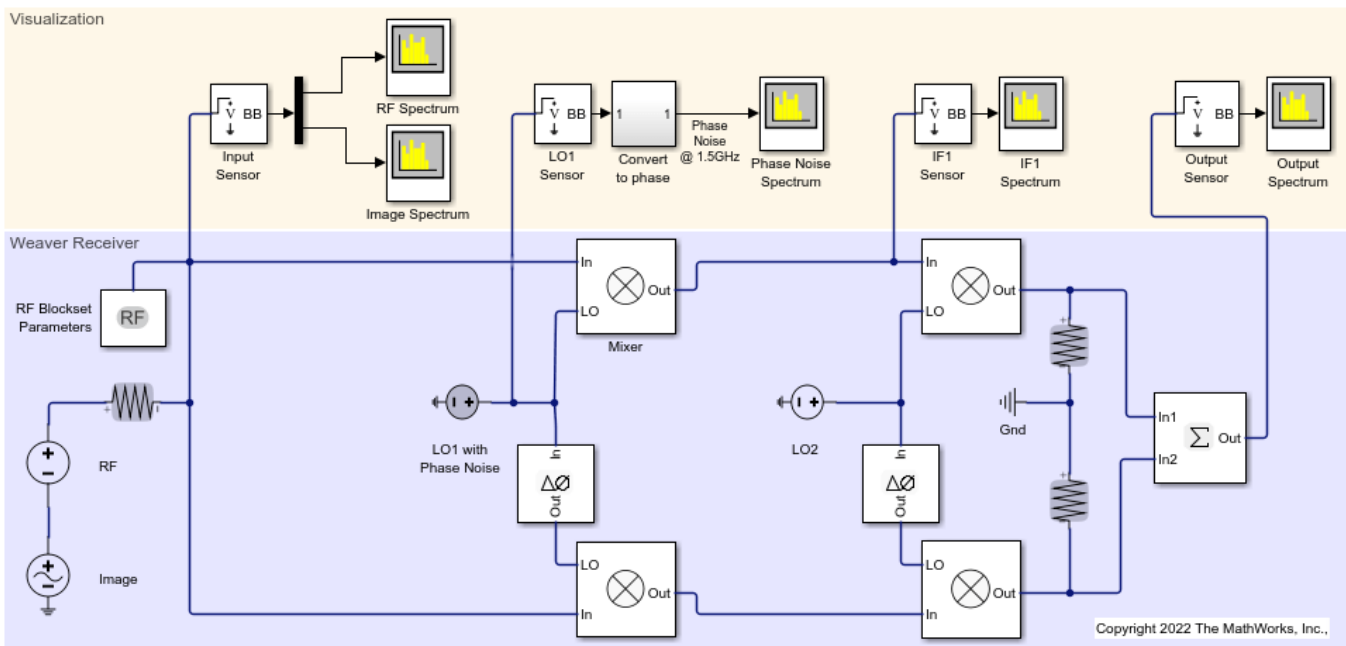


The preceding figure shows the transfer of phase noise from f_{LO1} to f_{IF1} .

Create Model with Phase Noise

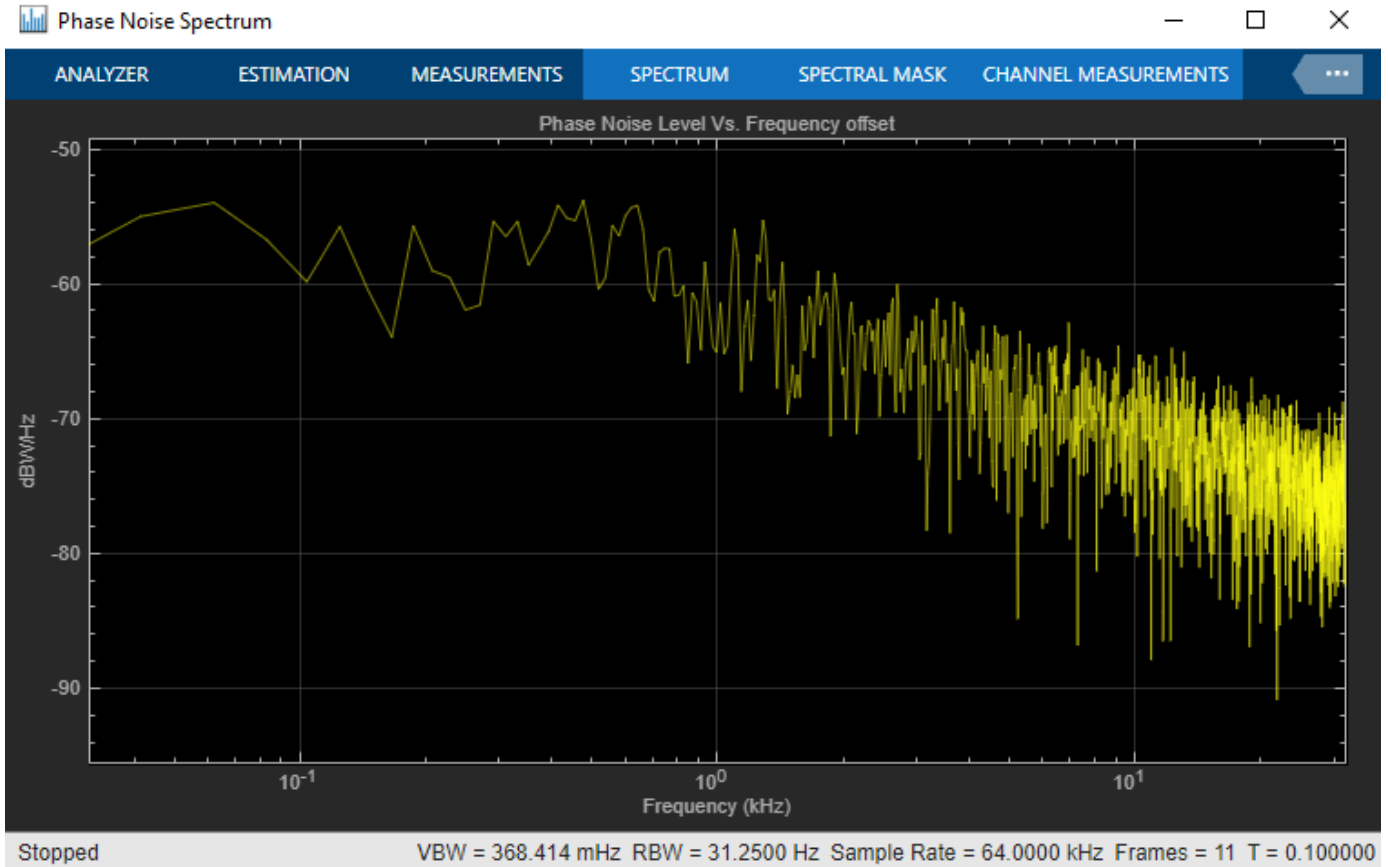
The model `ex_simrf_phase_noise` introduces phase noise into the model from the example, “Carrier to Interference Performance of Weaver Receiver” on page 7-48. The first mixing stage downconverts the RF and image to f_{IF} . To open the model:

```
open_system('ex_simrf_phase_noise')
```



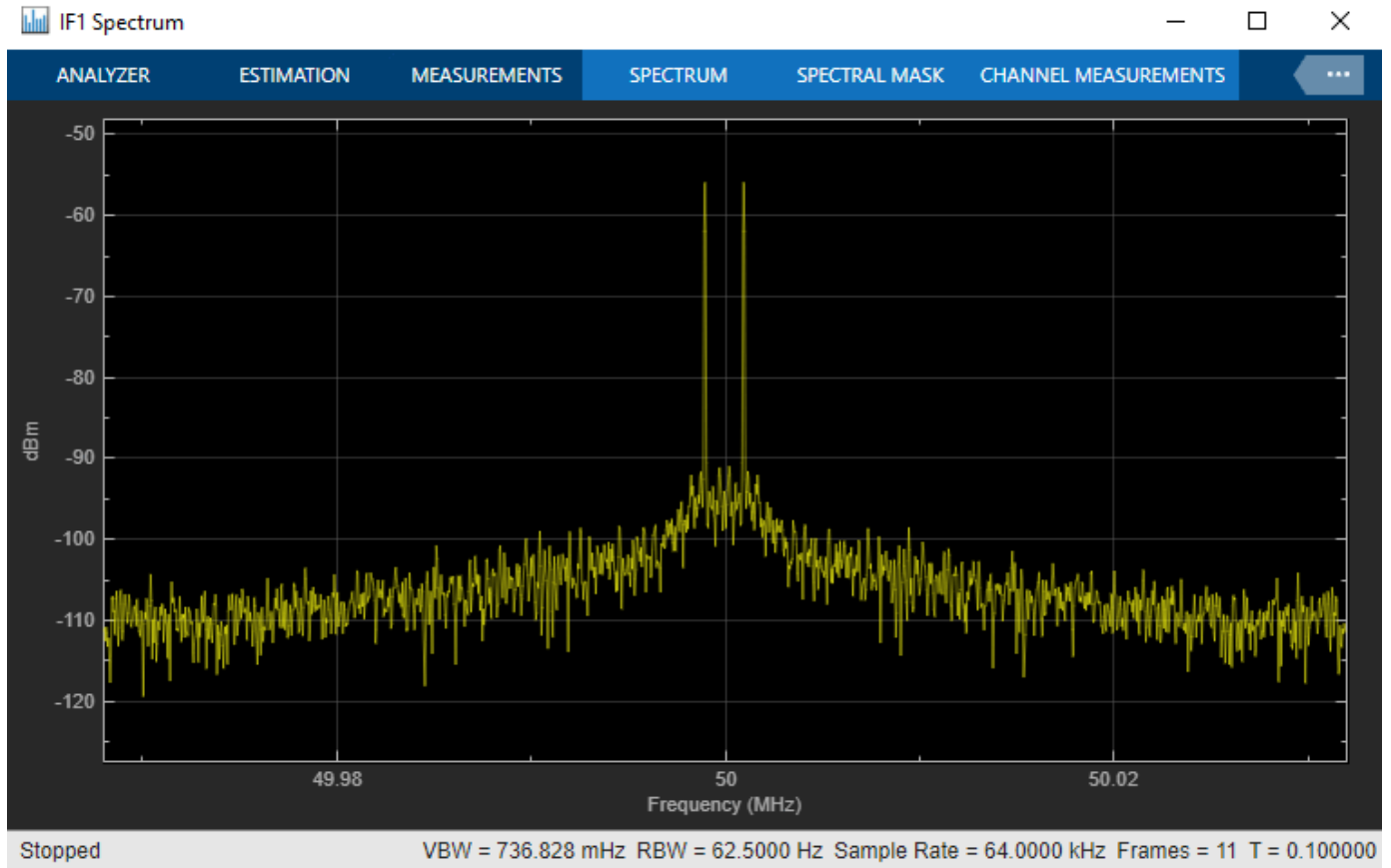
View Simulation Output

The model uses Spectrum Analyzer to generate 5 plots.



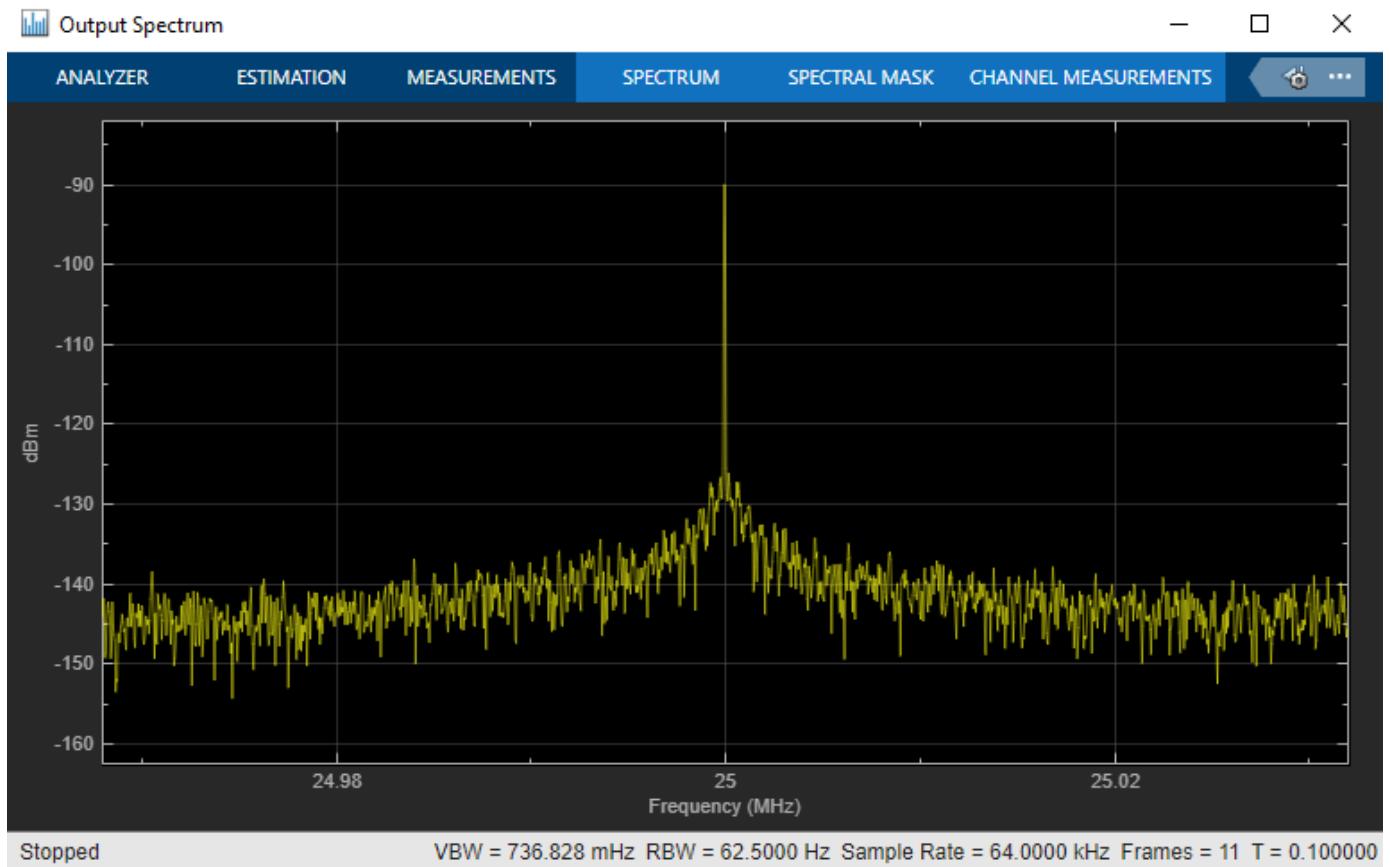
The Phase Noise spectrum scope shows a single-sided power density spectrum measuring the phase noise level at the LO1 source versus frequency offset shown in logarithmic scale.

The IF1 spectrum scope shows a power spectrum centered at the first intermediate frequency, measured between the first and second stages.



The scope shows that the LO phase noise has been transferred to the image. The RF signal on the carrier f_{IF1} is not visible in the figure because its power level is below the phase noise power of the downconverted image signal.

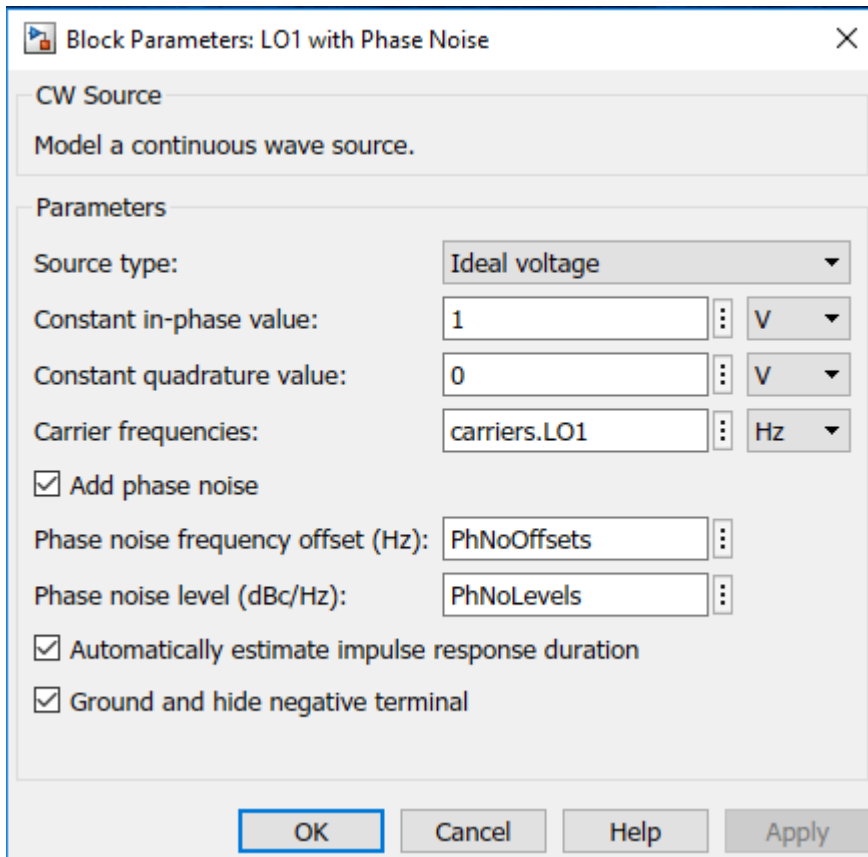
The Output spectrum scope shows the downconverted RF with the images removed.



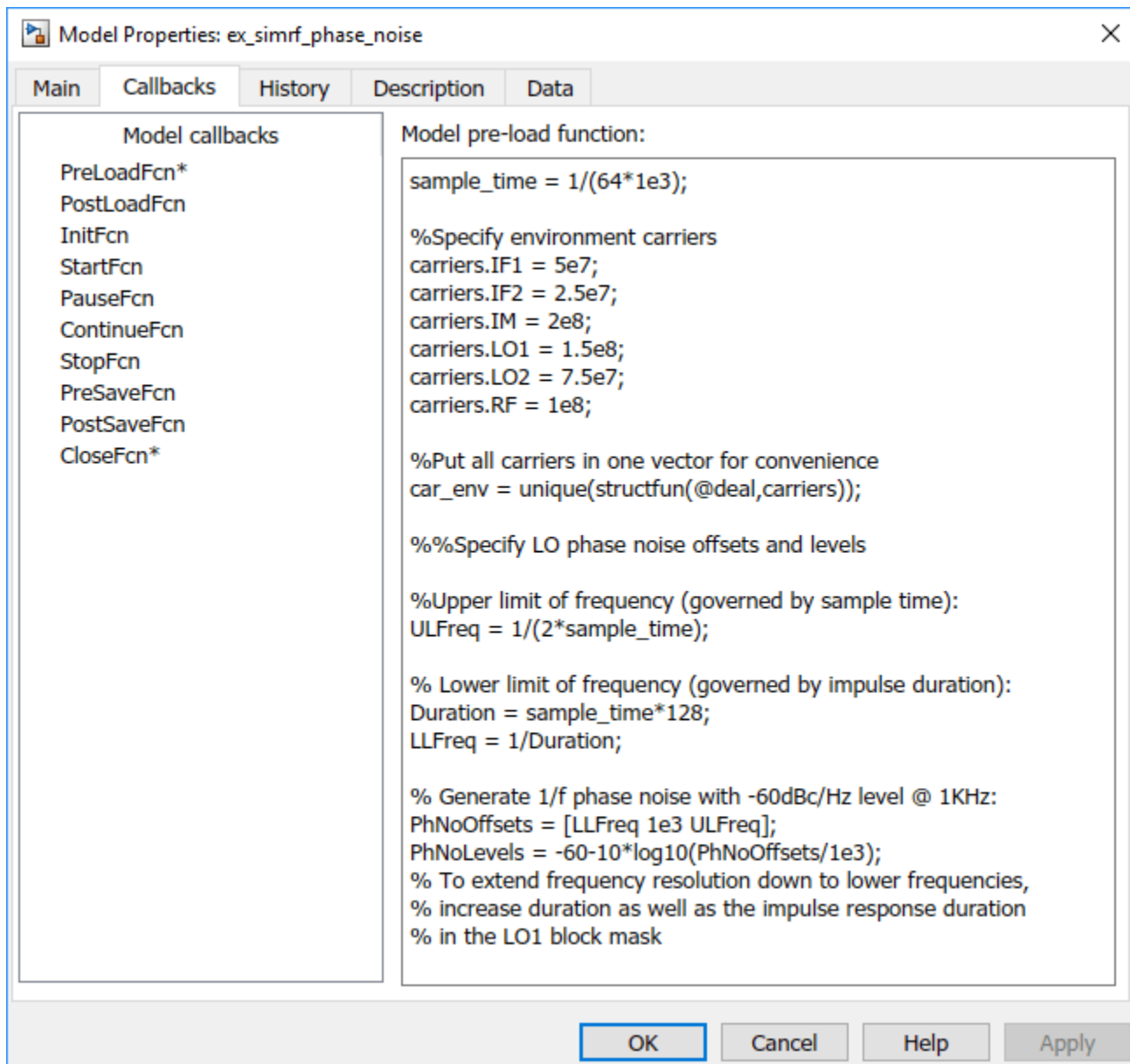
The LO phase noise has been transferred to the receiver output.

Shaping LO Noise Skirt

As seen in the phase noise scope, the added phase noise is pink ($1/f$) and is specified within the CW source LO1. Specifically, the **Add phase noise** checkbox is checked in the block parameters dialog:



The phase noise frequency offset, `PhNoOffsets`, and phase noise level, `PhNoLevels`, variables are defined in the models `PreLoadFcn` callback, accessible through **SETUP > Model Settings > Model Properties** :



The upper limit of the offset frequency is governed by the sample time and is limited to the envelope bandwidth of the simulation. The lower limit of the offset frequency is governed by the duration of the impulse response generating the phase noise frequency profile. Increasing the duration length in time steps from 128 to 256 will double the frequency resolution and allow simulation of the 1/f profile down to 250Hz (as opposed to the current lower limit of 500 Hz).

See Also

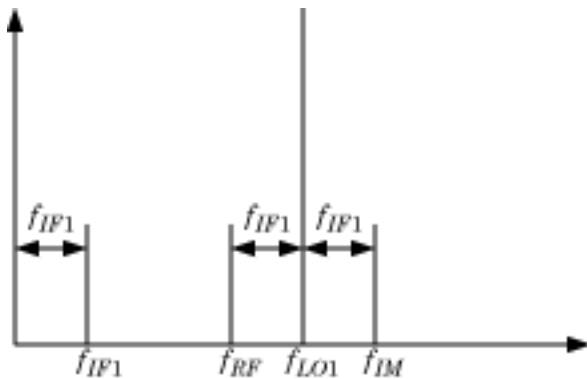
Noise Figure Testbench | Mixer

Related Topics

“RF Noise Modeling” on page 8-184

Carrier to Interference Performance of Weaver Receiver

A classic superheterodyne architecture filters images prior to frequency conversion. In contrast, image-reject receivers remove the images at the output without filtering but are sensitive to phase offsets.

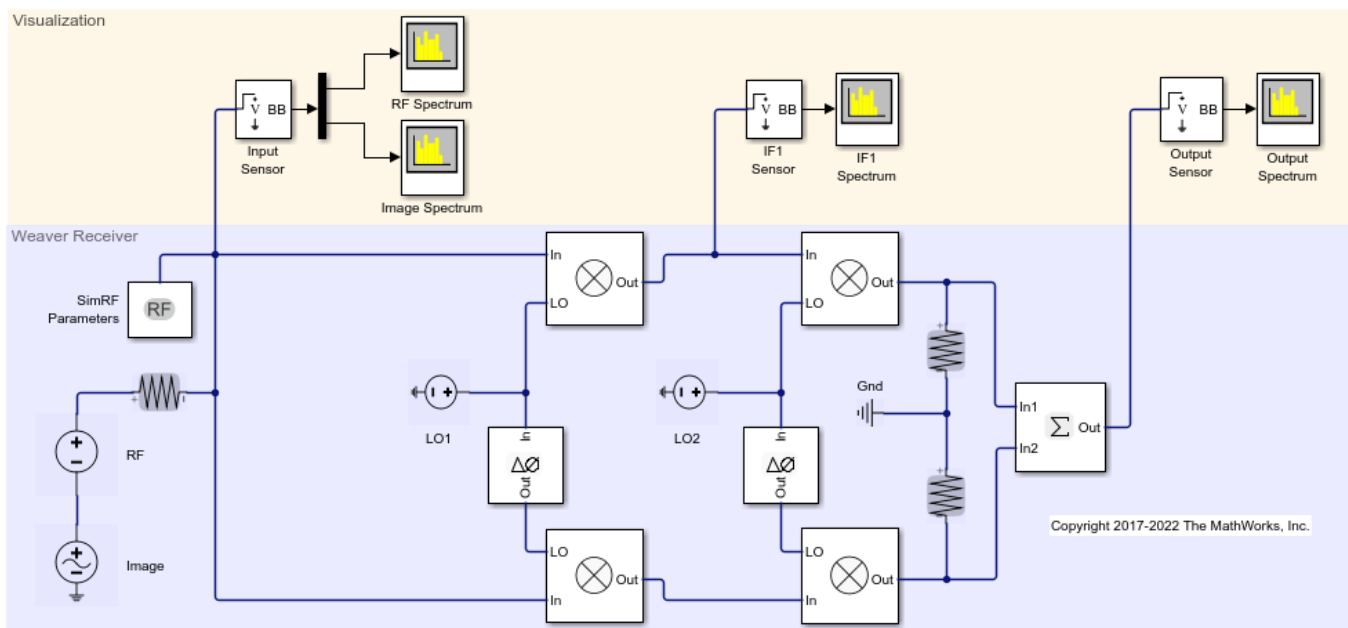


The preceding figure illustrates two input signals at the carriers f_{RF} and f_{IM} that both differ from the LO frequency, f_{LO1} , by an amount f_{IF1} . Mixing translates both input signals down to f_{IF1} . Perfect image rejection in the final stage of the receiver removes the image signal from the output entirely.

Create Model with RF Interference

The model `ex_simrf_ir` performs image rejection with a Weaver architecture. The receiver downconverts the signals f_{RF} and f_{IM} to f_{IF1} and f_{IF2} in two sequential stages.

```
open_system('ex_simrf_ir')
```



Set Up RF Blockset Environment

To maximize performance, the **Fundamental tones** and **Harmonic order** parameters are explicitly set in the **Configuration** block. To create the minimal set of simulation frequencies, the following carrier frequencies are set or created within the model.

- f_{RF} , the RF carrier, equals 100 MHz.
- f_{IM} , the image of the RF carrier relative to f_{LO1} , equals 200 MHz.
- f_{LO1} , the frequency of the LO in the first mixing stage, equals 150 MHz.
- f_{IF1} , the intermediate frequency of the signal after the first mixing stage equals:
 $f_{LO1} - f_{RF} = f_{IM} - f_{LO1} = 50$ MHz.
- f_{LO2} , the frequency of the LO second mixing stage equals 75 MHz.
- f_{IF2} , the intermediate frequency of the signal after the second mixing stage equals: $f_{LO2} - f_{IF1} = 25$ MHz.

In this system, every carrier is a multiple of f_{IF2} . The largest carrier, f_{IM} , is the 8th harmonic of f_{IF2} , so setting **Fundamental tones** to f_{IF2} and the **Harmonic order** to 8 ensures that every carrier is in the set of simulation frequencies.

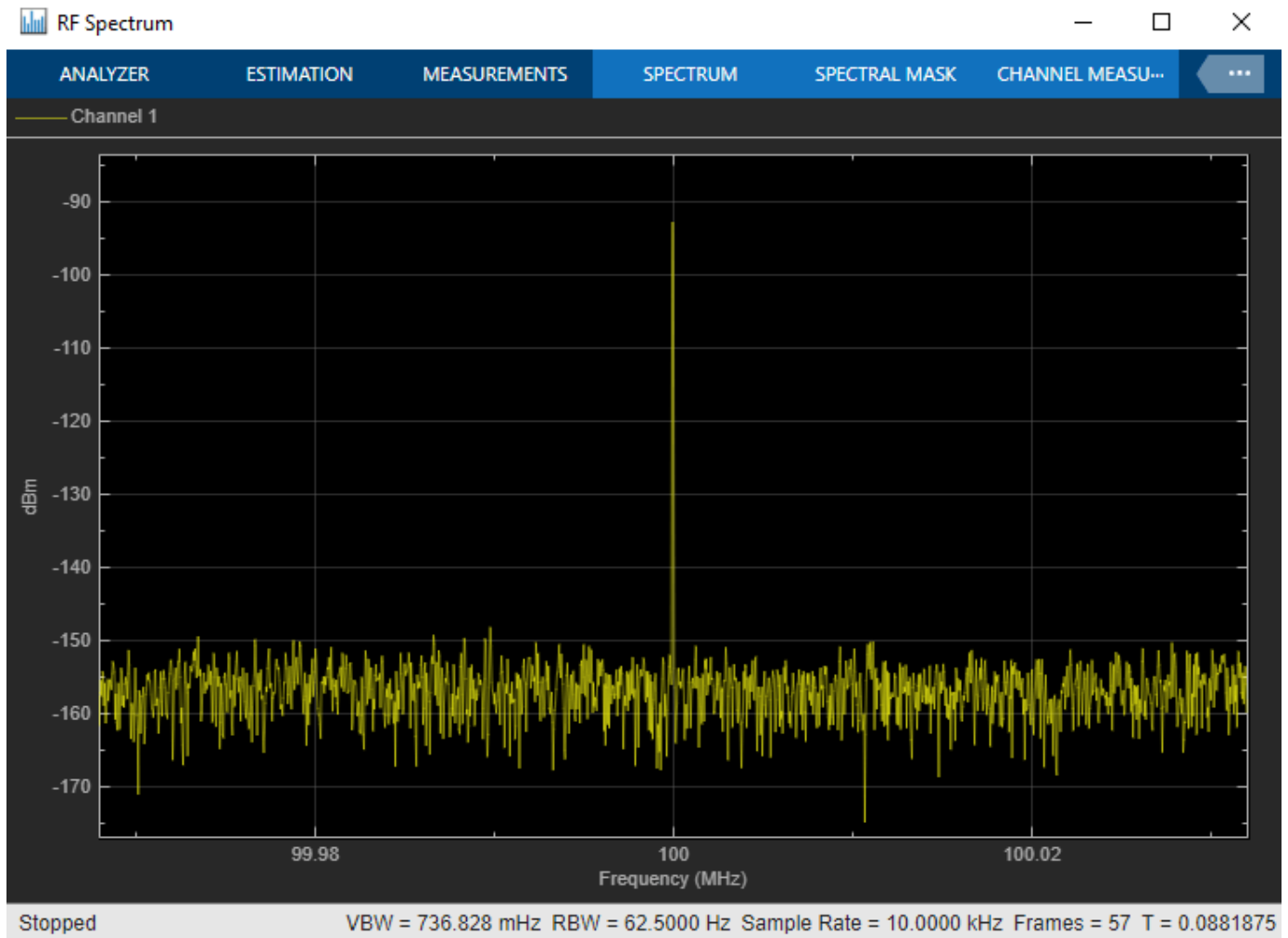
Specify the solver conditions and noise settings in the **Configuration** block:

- The **Solver type** is set to auto. For more information on choosing solvers, see the reference page for the **Configuration** block or see **Choosing Simulink and Simscape Solvers**.
- The **Step size** parameter is set to $1/(\text{mod_freq} * 64)$. This setting ensures a simulation bandwidth 64 times greater than the envelope signals in the system.
- The **Simulate noise** box is checked, so the environment includes noise in the simulation.

View Simulation Output

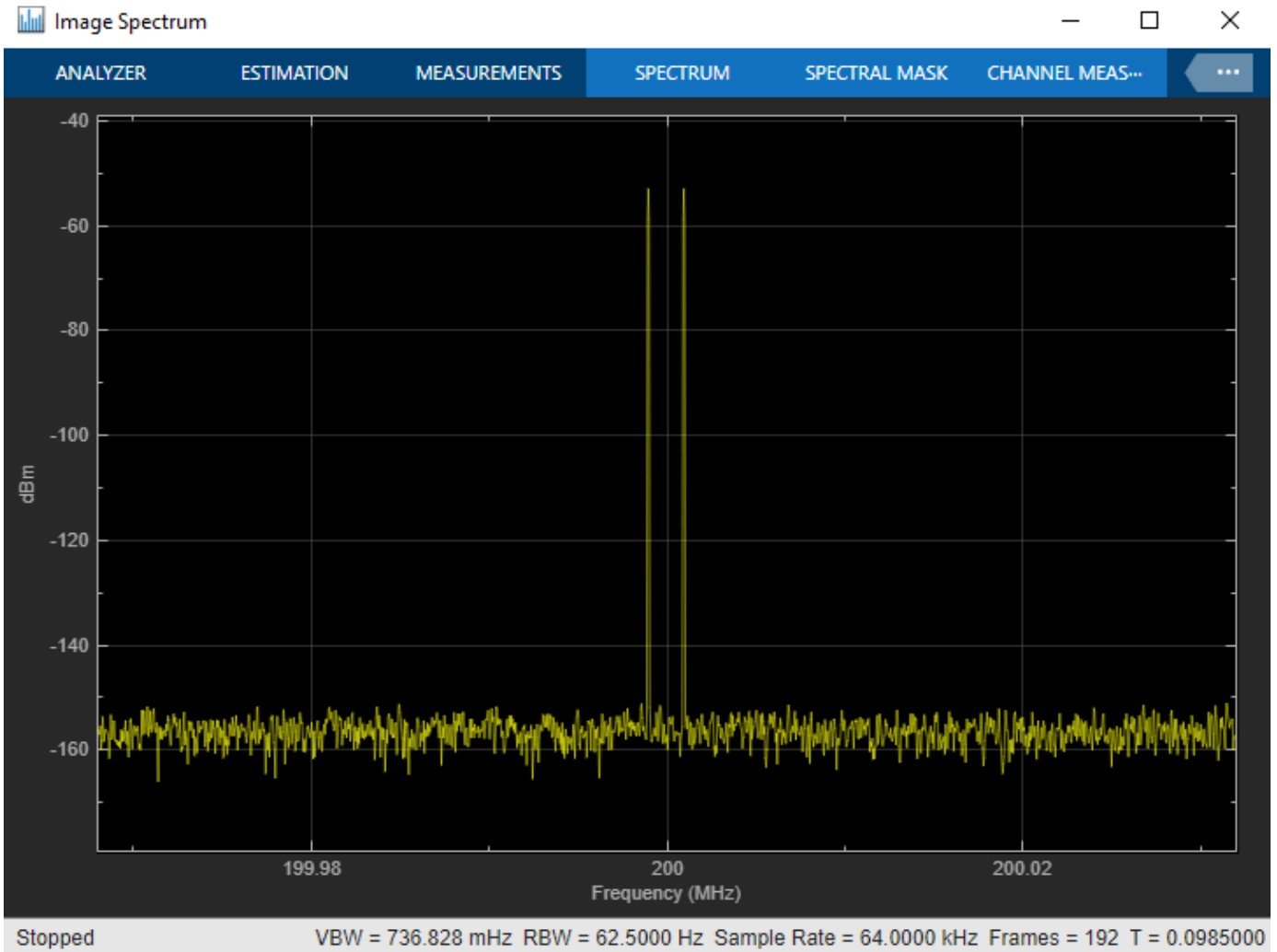
The model uses **Spectrum Analyzer** to generate four plots.

The RF spectrum scope shows the power spectrum of the carrier signal f_{RF} , specified as **carriers.RF** in the **Carrier frequencies** parameter of the **Input Sensor Output** block.



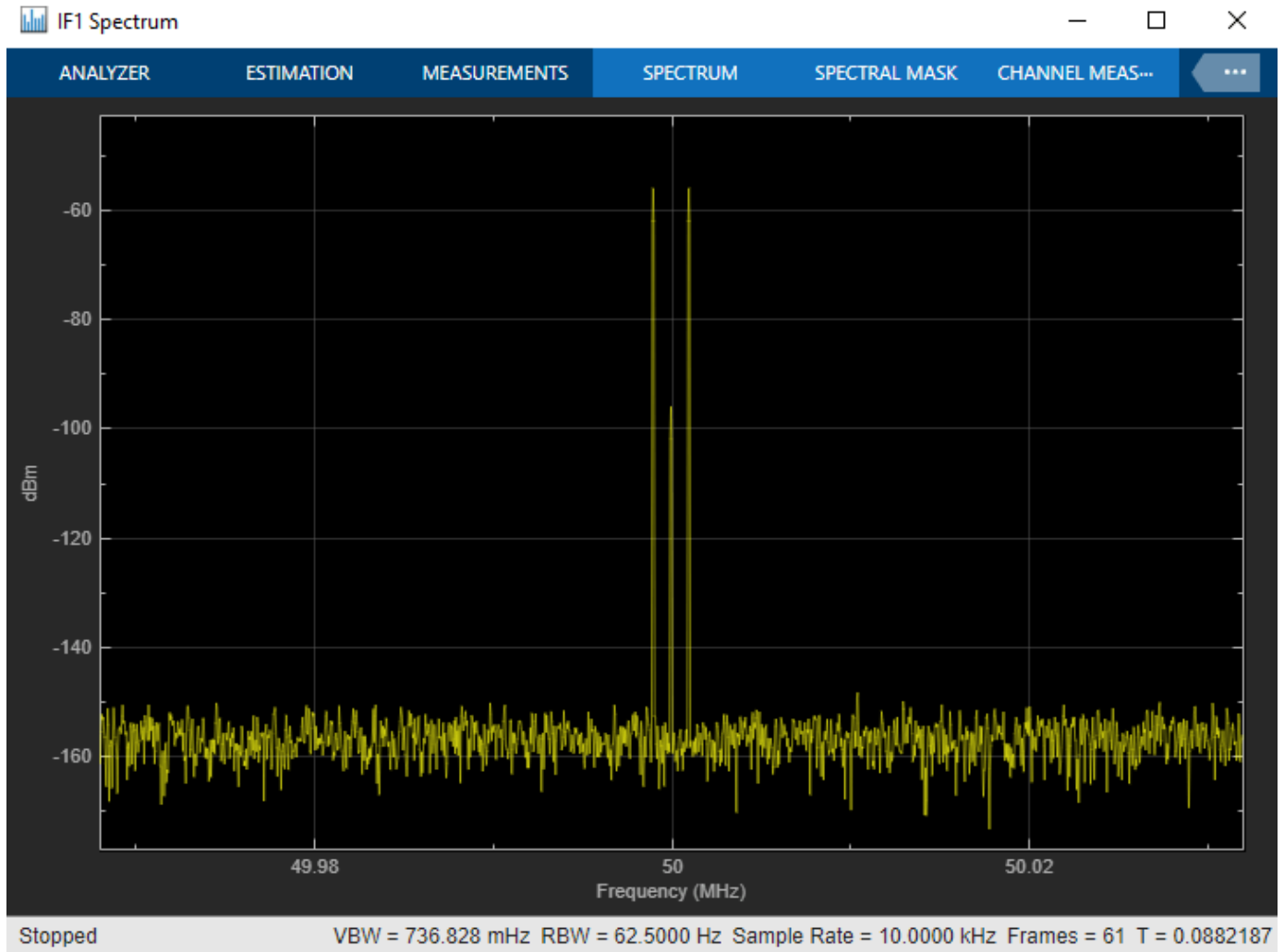
The modulation of the RF carrier is a constant envelope generated by a Continuous Wave block which generates a single peak centered at the carrier.

The Image Spectrum scope shows the power spectrum of the image. The signal is recovered from the carrier f_{IM} , specified as `carriers.IM` in the **Carrier frequencies** parameter of the Input Sensor Output block.



The Image Sinusoidal Source block generates a two-tone signal centered at f_{IM} .

The IF1 spectrum scope plot shows a power spectrum centered at the first intermediate frequency, measured between the first and second stages. The sensor outputs the modulation from the carrier f_{IF1} , specified as carriers .IF1 in the **Carrier frequencies** parameter.



The Output spectrum scope shows the complete effects of the RF system. The sensor outputs the modulation from the carrier f_{IF2} , specified as carriers .IF2 in the **Carrier frequencies** parameter.



Model RF and Blocker Sources

To model more robust input signals, you can use a RF Blockset `Inport` block to specify a circuit envelope signal generated using blocks from other Simulink™ libraries. For example, see the featured example [Impact of an RF Receiver on Communication System Performance](#). This example uses Communications System Toolbox™ to model a QPSK-modulated waveform of random bits with RF Blockset `Inport` that brings the signal into the RF Blockset environment.

Simulating LO Phase Offset

The phase shifters have specified Phase shift parameters of 90. Deviation from this value results in a phase offset and causes imperfect image rejection. The featured example [Measuring Image](#)

Rejection Ratio in Receivers analyzes the IRR of the Weaver and Hartley architectures several times, calculating the image rejection ratio (IRR) for several different phase offsets.

See Also

Related Examples

- “Top-Down Design of an RF Receiver” on page 8-166
- “Architectural Design of a Low IF Receiver System” on page 8-178

Modulate Two-Tone DC Signal Using IQ Modulator

Use the IQ Modulator block to Modulate a two-tone signal to RF level. Observe the impairments in the modulated output signal such as images due to gain imbalance, intermodulation distortion, and output third-order intercept (OIP3).

The two-tones are at 10 MHz and 15 MHz. The power of each tone is -30 dBm. The carrier frequency is 0 GHz.

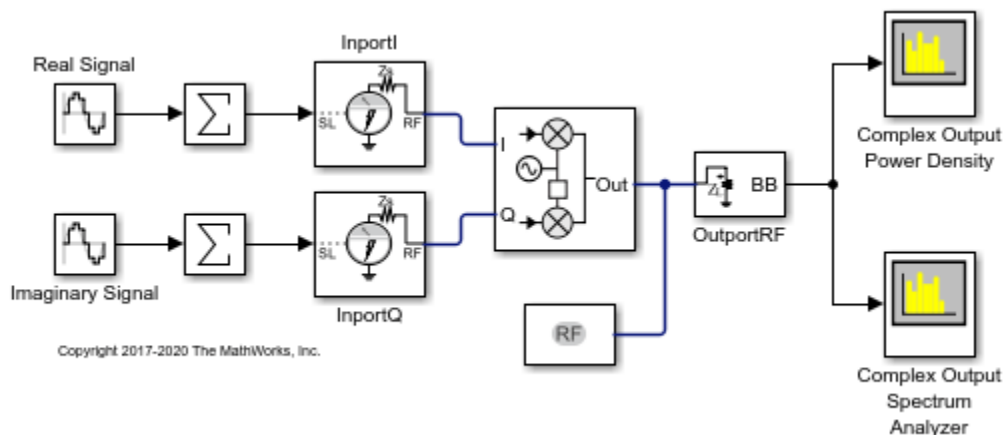
IQ Modulator

The IQ modulator parameters are :

- Available power gain: 10 dB
- Local oscillator frequency: 2 GHz
- I/Q gain mismatch: 0.1 dB
- LO to RF Isolation: 90 dB
- Noise Floor: -160 dBm/Hz
- IP3: 10 dBm

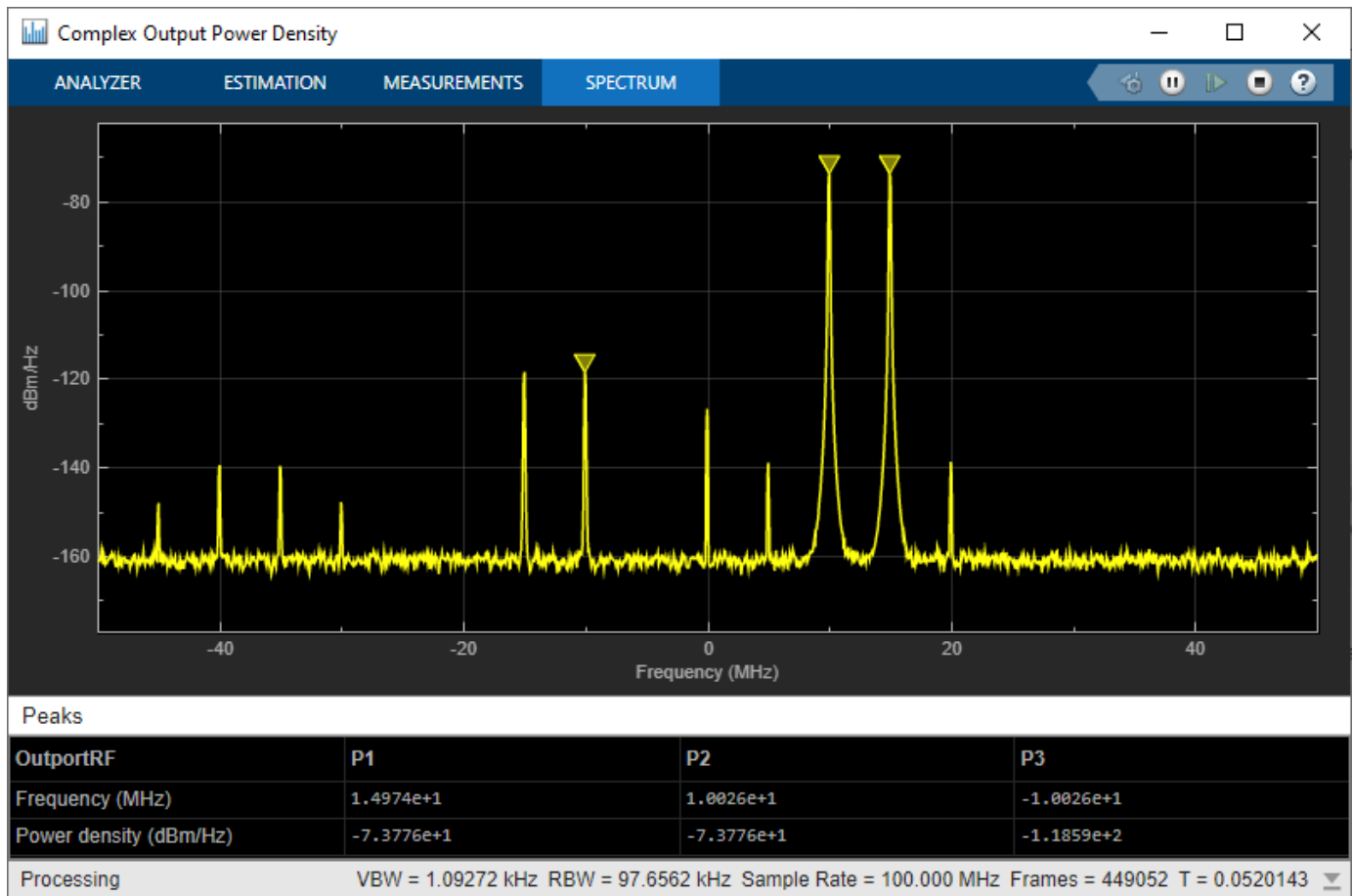
Open the model.

```
open('model_IQmod')
```



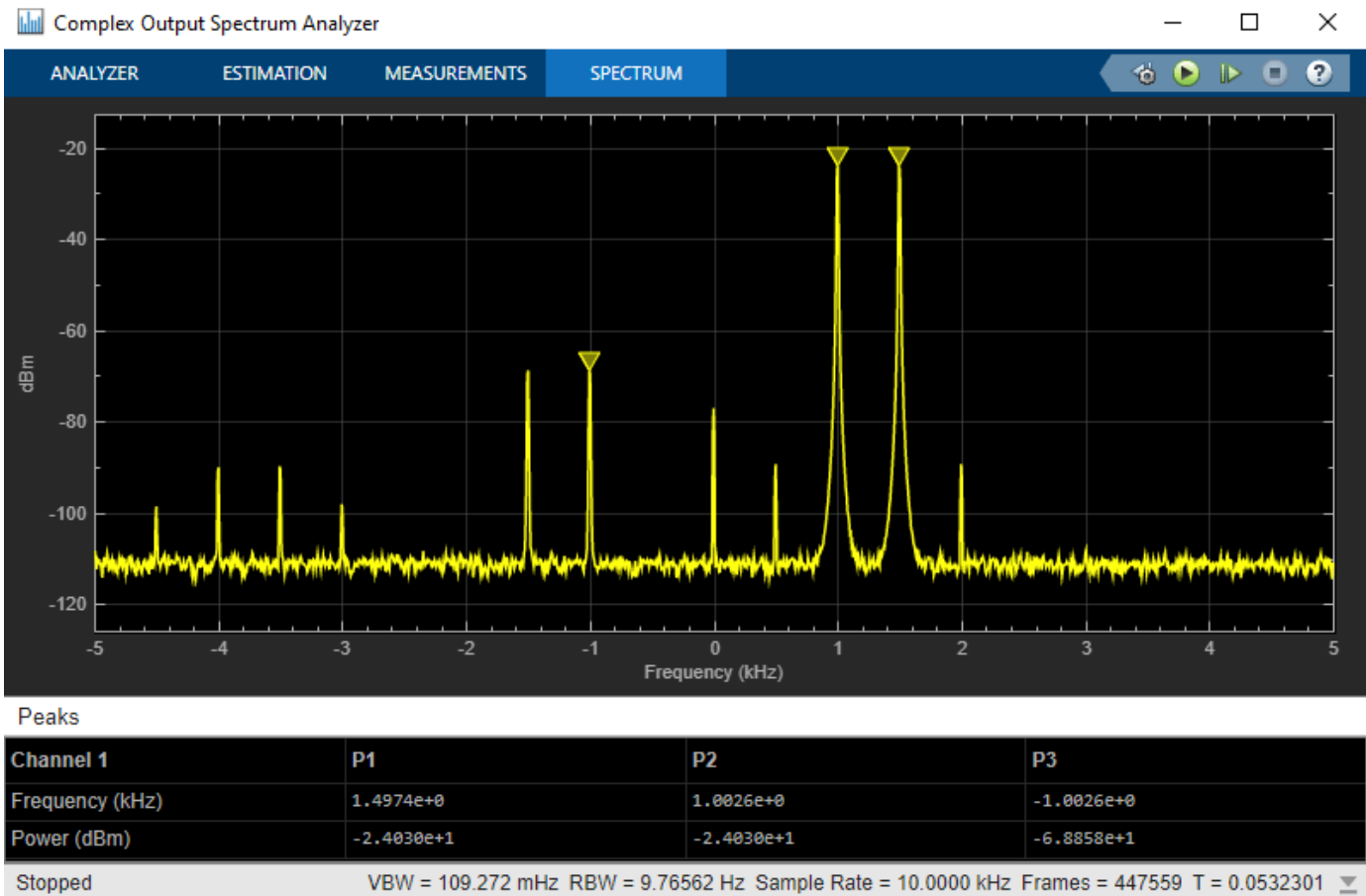
Run the model and observe the spectrum analyzers.

Complex Output Power Density



In the complex output power density spectrum analyzer, you see the noise floor of the signal at -160 dBm/Hz.

Complex Output Spectrum Analyzer



In the complex output spectrum analyzer, you see the whole modulated signal including the imaginary parts. The output power level of the two tones (10 MHz and 15 MHz) is -20 dBm.

$$\text{Output power level} = \text{Input power level} + \text{gain} = -30\text{dBm} + 10\text{dB} = -20\text{dBm}$$

The output third-order intercept (OIP3) is at 10 dBm. The spectrum analyzer measures this value.

Image Rejection Ratio

The images of the two-tones are at -10 MHz and -15 MHz. The output power level of the two images are -67.8 dBm. Image rejection ratio is calculated using:

$$\text{IMRR} = \frac{[(\text{gain imbalance})^2 + 1 - 2 * (\text{gain imbalance})]}{[(\text{gain imbalance})^2 + 1 + 2 * (\text{gain imbalance})]}$$

where,

$$\text{Gain Imbalance} = 10^{(0.1/20)} = 1.0116$$

$$\text{IMRR}_{\text{dB}} = 10 * \log_{10}(3.3253e - 05) = -44.78\text{dB}$$

The image power level is calculated using this:

$$\text{Imagelevel} = -23\text{dBm} - 44.78\text{dB} = -67.78\text{dBm}$$

See Also

IQ Modulator

Related Topics

“Demodulate Two-Tone RF Signal Using IQ Demodulator” on page 7-7

Measurement of Gain and Noise Figure Spectrum

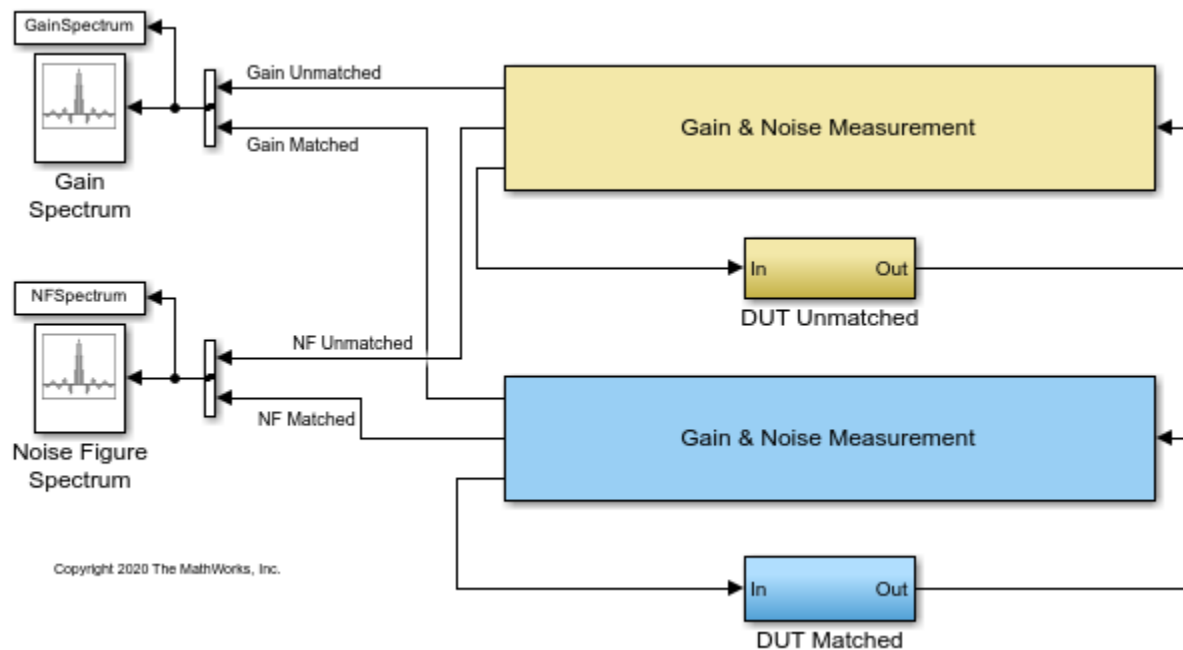
This example shows how to use RF Blockset to measure the Gain and Noise Figure of an RF system over a given spectral range.

The example requires DSP System Toolbox™.

Introduction

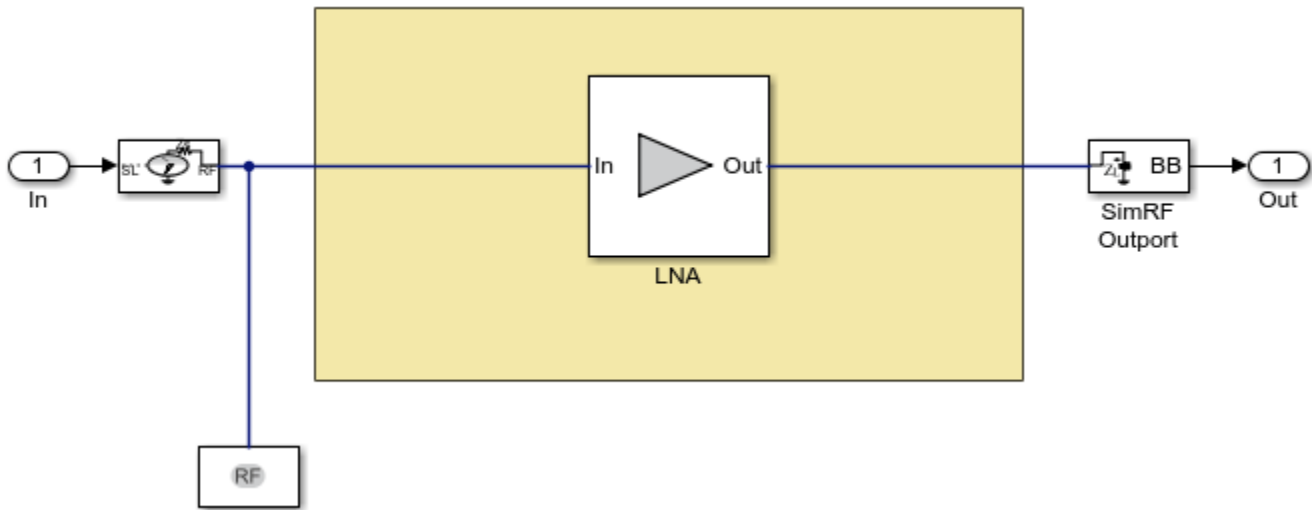
In this example, a method for measuring the frequency-dependent gain and noise figure of an RF system is described. These spectral properties are measured for two RF systems; A single Low Noise Amplifier and the same amplifier when matched. The model used for the measurement is shown below:

```
model = 'GainNoiseMeasurementExample';
open_system(model);
```



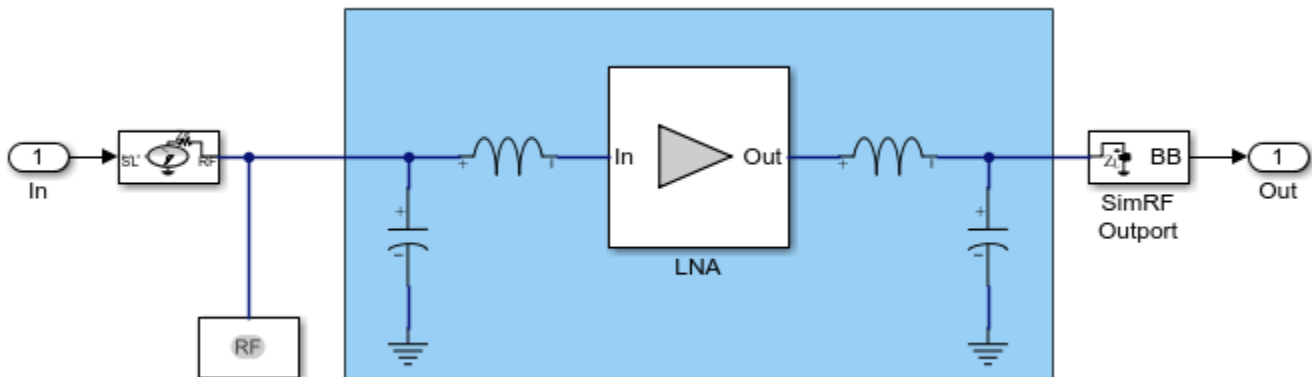
The model has two measurement units, each connected to a different subsystem containing the DUT. The upper measurement unit is connected to an unmatched LNA in the DUT subsystem with yellow background:

```
open_system([model '/DUT Unmatched']);
```



The lower measurement unit is connected to a matched LNA in the DUT subsystem with blue background:

```
open_system([model '/DUT Matched']);
```



Each measurement unit outputs two vector signals representing the spectrums of the Gain and Noise Figure of the corresponding DUT and those are inputted into two Array Plot (DSP System Toolbox) blocks that plot the above properties versus frequency, comparing the unmatched and matched DUT systems. In the following sections, the matching network design process is described, the simulation results are given and compared with these expected from LNA and matching network properties. Finally, the procedure used within the measurement units to obtain the spectral Gain and Noise results is explained.

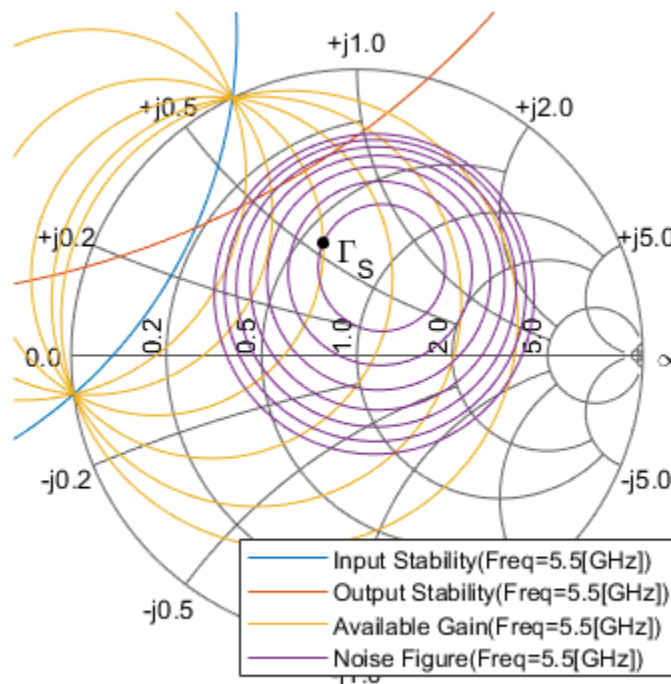
Design of the matching network

The matching network used in the matched DUT subsystem comprises a single stage L-C network that is designed following the same procedure as the one described in the RF Toolbox example “Designing Matching Networks for Low Noise Amplifiers”. Since the LNA used here is different, the design is described below

Initially, an `rfckt.amplifier` object is created to represent an Heterojunction Bipolar Transistor based low noise amplifier that is specified in the file, 'RF_HBT_LNA.S2P'. Then, the `circle` method of the `rfckt.amplifier` object is used to place the constant available gain and the constant noise figure circles on a Smith chart, and select an appropriate source reflection coefficient, Γ_S , that provides a suitable compromise between gain and noise. The Γ_S value chosen yields an available gain of $G_a=21\text{dB}$, and a noise figure of $NF=0.9\text{dB}$ at the center frequency $f_c=5.5\text{GHz}$:

```
unmatched_amp = read(rfckt.amplifier, 'RF_HBT_LNA.S2P');
fc = 5.5e9; % Center frequency (Hz)
circle(unmatched_amp,fc,'Stab','In','Stab','Out','Ga',15:2:25, ...
       'NF',0.9:0.1:1.5);

% Choose GammaS and show it on smith chart:
hold on
GammaS = 0.411*exp(1j*106.7*pi/180);
plot(GammaS,'k.','MarkerSize',16)
text(real(GammaS)+0.05,imag(GammaS)-0.05,'\Gamma_{S}','FontSize', 12, ...
     'FontUnits','normalized')
hLegend = legend('Location','SouthEast');
hLegend.String = hLegend.String(1:end-1);
hold off
```



```
Available Gain(Freq=5.5[GHz]) X
Ga = 21.00 [dB]:
NF = 0.90 [dB]:
GammaS = |0.4095|, 106.7 [deg]:
GammaOut = |0.5951|, -134.9 [deg]:
ZS = 0.593 + j0.559:
```

For the chosen GammaS, the following properties can be obtained:

```
% Normalized source impedance:
Zs = gamma2z(GammaS,1);

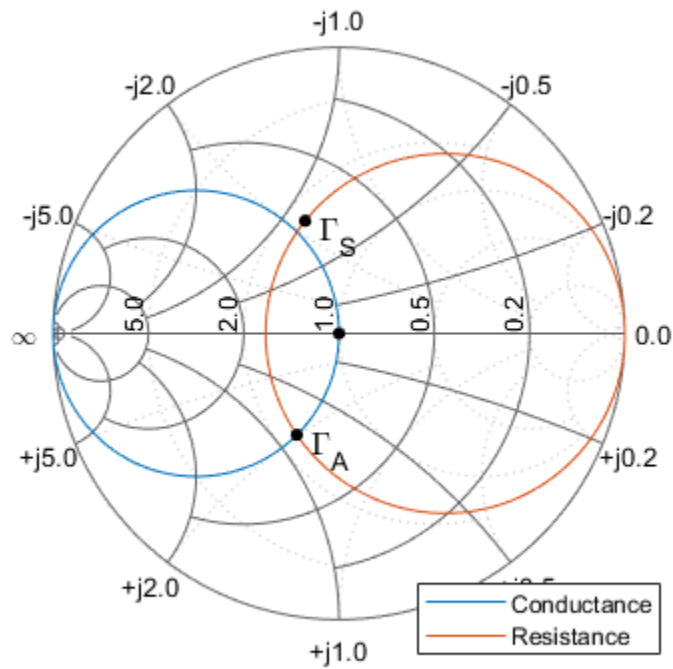
% Matching |GammaL| that is equal to the complex conjugate of
% |GammaOut| shown on the data tip:
GammaL = 0.595*exp(1j*135.0*pi/180);

% Normalized load impedance:
Zl = gamma2z(GammaL,1);
```

The input matching network consists of one shunt capacitor, C_{in} , and one series inductor, L_{in} . The Smith chart is used to find the component values. To do this, the constant conductance circle that crosses the center of the Smith chart and the constant resistance circle that crosses GammaS are plotted and the intersection points (Point Γ_A) is found:

```
[~, hsm] = circle(unmatched_amp,fc,'G',1,'R',real(Zs));
hsm.Type = 'YZ';

% Choose GammaA and show points of interest on smith chart:
hold on
plot(GammaS,'k.','MarkerSize',16)
text(real(GammaS)+0.05,imag(GammaS)-0.05,'\Gamma_{S}','FontSize',12,...
      'FontUnits','normalized')
plot(0,0,'k.','MarkerSize',16)
GammaA = 0.384*exp(1j*(-112.6)*pi/180);
plot(GammaA,'k.','MarkerSize',16)
text(real(GammaA)+0.05,imag(GammaA)-0.05,'\Gamma_{A}','FontSize',12,...
      'FontUnits','normalized')
hLegend = legend('Location','SouthEast');
hLegend.String = hLegend.String(1:end-3);
hold off
```



```

Conductance
Gamma = |0.3830|, -112.5 [deg]:
Z = 0.592 - j0.491:
Y = 1.000 + j0.829:

```

Using the chosen Γ_A , the input matching network components, C_{in} and L_{in} , are obtained:

```
% Obtain admittance  $Y_a$  corresponding to  $\Gamma_A$ :
```

```
 $Z_a = \text{gamma2z}(\Gamma_A, 1);$ 
```

```
 $Y_a = 1/Z_a;$ 
```

```
% Using  $Y_a$ , find  $C_{in}$  and  $L_{in}$ :
```

```
 $C_{in} = \text{imag}(Y_a)/50/2/\pi/f_c$ 
```

```
 $L_{in} = (\text{imag}(Z_s) - \text{imag}(Z_a))*50/2/\pi/f_c$ 
```

```
 $C_{in} =$ 
```

```
4.8145e-13
```

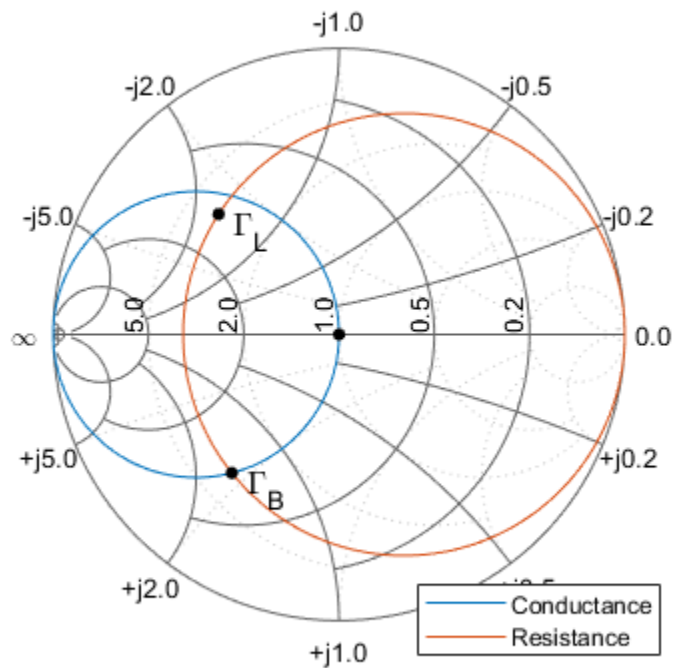
```
 $L_{in} =$ 
```

```
1.5218e-09
```

In a similar manner, the output matching network components are obtained using the intersection points (Point Γ_B) between a constant conductance circle that crosses the center of the Smith chart and the constant resistance circle that crosses Γ_L :

```
[hLine, hsm] = circle(unmatched_amp,fc,'G',1,'R',real(Zl));
hsm.Type = 'YZ';

% Choose GammaB and show points of interest on smith chart:
hold on
plot(GammaL,'k.','MarkerSize',16)
text(real(GammaL)+0.05,imag(GammaL)-0.05,'\Gamma_{L}','FontSize',12,...
     'FontUnits','normalized')
plot(0,0,'k.','MarkerSize',16)
GammaB = 0.612*exp(1j*(-127.8)*pi/180);
plot(GammaB,'k.','MarkerSize',16)
text(real(GammaB)+0.05,imag(GammaB)-0.05,'\Gamma_{B}','FontSize',12,...
     'FontUnits','normalized')
hLegend = legend('Location','SouthEast');
hLegend.String = hLegend.String(1:end-3);
hold off
```



```
Conductance
Gamma = [0.6133], -127.8 [deg]:
Z = 0.293 - j0.455:
Y = 1.000 + j1.553:
```

Using the chosen Γ_B , the input matching network components, C_{out} and L_{out} , are obtained:

```
% Obtain admittance Yb corresponding to GammaB:
```

```
Zb = gamma2z(GammaB, 1);
```

```
Yb = 1/Zb;
```

```
% Using Yb, find Cout and Lout:
```

```
Cout = imag(Yb)/50/2/pi/fc
```

```
Cout =
```

```
8.9651e-13
```

```
Lout = (imag(Zl) - imag(Zb))*50/2/pi/fc
```

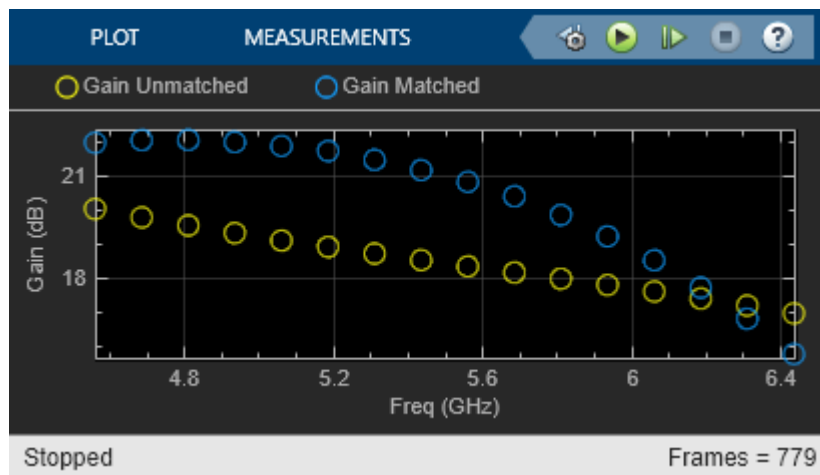
```
Lout =
```

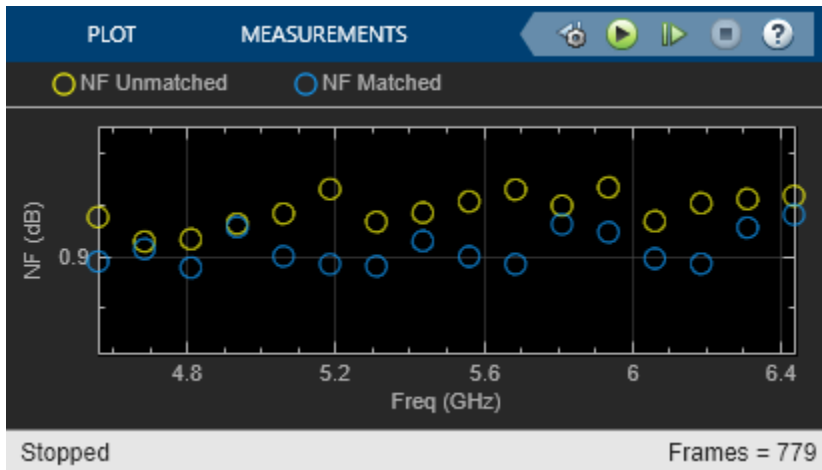
```
1.2131e-09
```

Simulation results for gain and noise figure spectrum measurement model

The above input and output network component values are used in the simulation of the matched DUT in the gain and noise figure spectrum measurement model described earlier. The spectral results displayed in the Array Plot blocks are given below:

```
open_system([model '/Gain Spectrum']);
open_system([model '/Noise Figure Spectrum']);
sim(model, 1e-4);
```





Next, the simulation results are compared with those expected analytically. To facilitate the comparison, the unmatched and matched amplifier networks are analyzed using RF Toolbox. In addition, as finer details are required, the simulation is run for a longer time. The results of the longer simulation are given in the file 'GainNoiseResults.mat'.

```
% Analyze unmatched amplifier
BW_analysis = 2e9; % Bandwidth of the analysis (Hz)
f_analysis = (-BW_analysis/2:1e6:BW_analysis/2)+fc;
analyze(unmatched_amp, f_analysis);

% Create and analyze an RF network for the matched amplifier
input_match = rfckt.cascade('Ckts', ...
    {rfckt.shuntrlc('C',Cin),rfckt.seriesrlc('L',Lin)});
output_match = rfckt.cascade('Ckts', ...
    {rfckt.seriesrlc('L',Lout),rfckt.shuntrlc('C',Cout)});
matched_amp = rfckt.cascade('ckts', ...
    {input_match,unmatched_amp,output_match});
analyze(matched_amp,f_analysis);

% Load results of a longer simulation
load 'GainNoiseResults.mat' f GainSpectrum NFSpectrum;

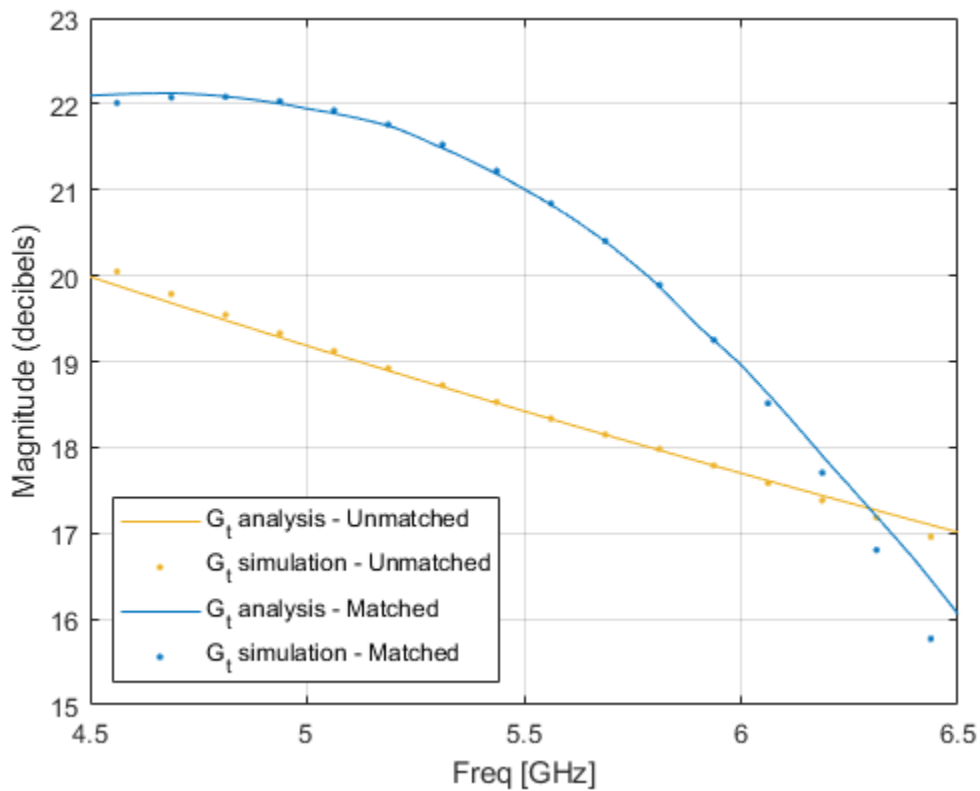
% Plot expected and simulated Transducer Gain
StdBlue = [0 0.45 0.74];
StdYellow = [0.93,0.69,0.13];
hLineUM = plot(unmatched_amp, 'Gt', 'dB');
hLineUM.Color = StdYellow;
hold on
plot(f, GainSpectrum(:,1), '.', 'Color', StdYellow);
hLineM = plot(matched_amp, 'Gt', 'dB');
hLineM.Color = StdBlue;
plot(f, GainSpectrum(:,2), '.', 'Color', StdBlue);
legend({'G_t analysis - Unmatched', ...
    'G_t simulation - Unmatched', ...
    'G_t analysis - Matched', ...
    'G_t simulation - Matched'}, 'Location','SouthWest');

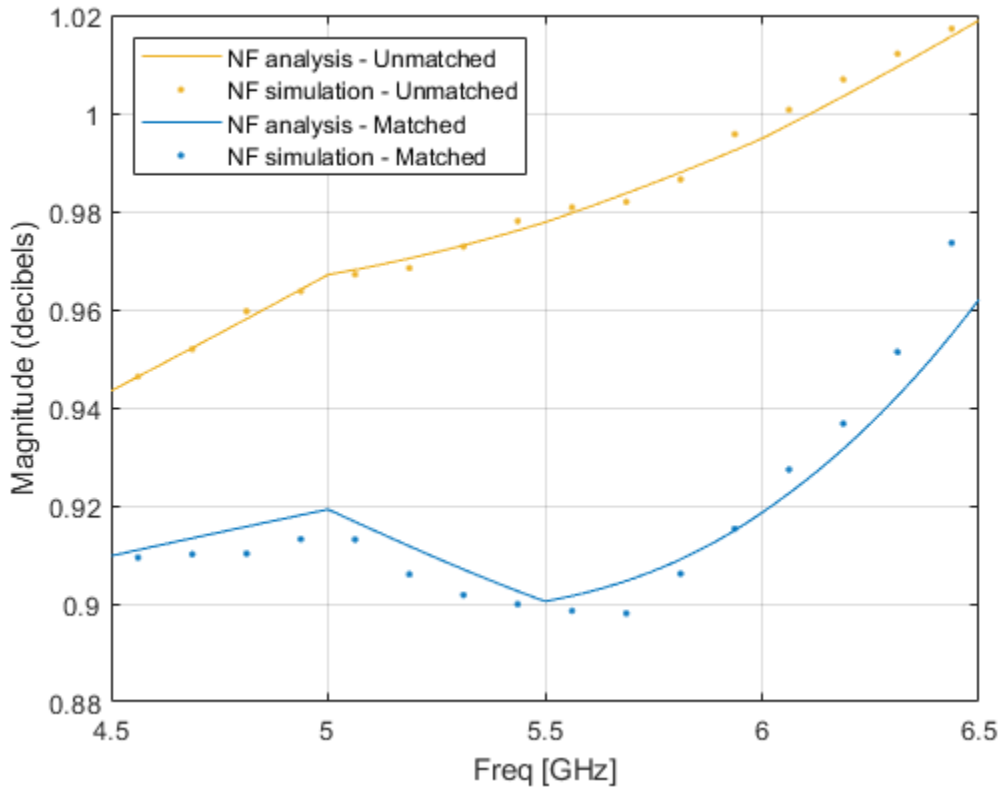
% Plot expected and simulated Noise Figure
hFig = figure;
hLineUM = plot(unmatched_amp, 'NF', 'dB');
```

```

hLineUM.Color = StdYellow;
legend('Location','NorthWest')
hold on
plot(f, NFSpectrum(:,1), '.', 'Color', StdYellow);
hLineM = plot(matched_amp, 'NF', 'dB');
hLineM.Color = StdBlue;
plot(f, NFSpectrum(:,2), '.', 'Color', StdBlue);
legend({'NF analysis - Unmatched', ...
       'NF simulation - Unmatched', ...
       'NF analysis - Matched', ...
       'NF simulation - Matched'}, 'Location','NorthWest');

```

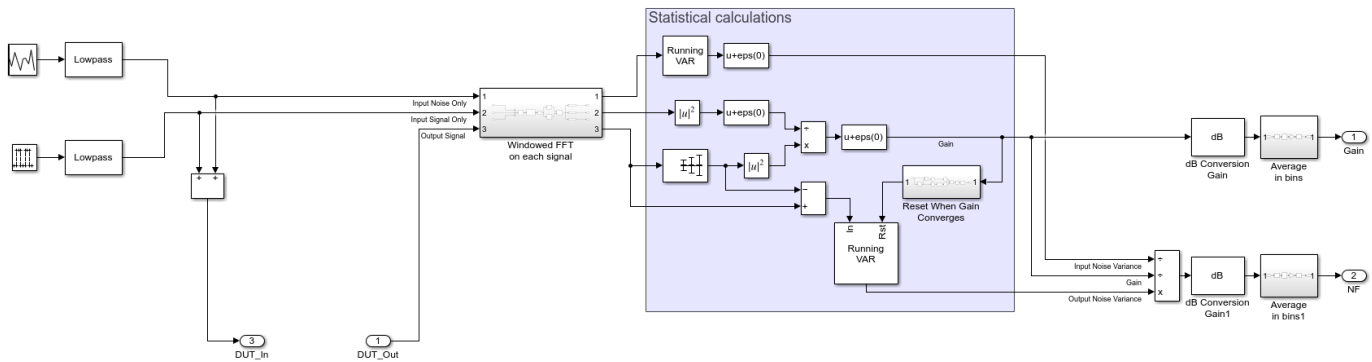




Operation of the measurement unit

The measurement unit produces an input signal, DUT_in, that is composed of zero-mean white noise and zero-variance impulse response signal. The latter is used to determine the frequency response of the DUT gain and together with the white noise determine the DUT noise figure. The measurement unit collects the DUT output signal, performs a windowed FFT on it and then facilitates statistical calculations to obtain the gain and the noise figure of the DUT.

```
open_system([model '/Noise and Gain Measurement'], 'force');
```



The statistical calculations are done in the area marked in blue. The calculations use three inputs in the frequency domain; Input Noise Only, Input Signal Only, and Output Signal. The Input Signal Only is compared with the mean of the Output Signal to determine the DUT's gain, G , at each frequency

bin. The variance of the Output Signal, with mean signal removed, yields the output noise of the DUT system, N_o . Together with the input noise fed to the DUT, N_i , calculated by taking the variance of the Input Noise Only, the Noise Figure, NF , can be calculated using the following formula:

$$NF = \frac{SNR_{in}}{SNR_{out}} = \frac{N_o}{N_i G}$$

Where, SNR_{in} and SNR_{out} in the above equation are the Signal-to-Noise ratios at the input and output of the DUT. Finally, after conversion to decibels, the spectral results are divided into bins and averaged within them to facilitate faster convergence. Also, to improve the noise calculation convergence, the output noise variance is reset once the gain has reached convergence.

The properties affecting the operation of the measurement unit are specified in the block's mask parameter dialog box as shown below:

Block Parameters: Noise and Gain Measurement

Subsystem (mask)

Measure Gain and Noise Figure Spectral content. Spectrum obtained is between $[-BW/2, BW/2] \times SPR$, where $BW = 1/ts$, and SPR is the spectrum coverage ratio.

Parameters

Sample Time (s)
1/BW

FFT size
N

Beta of Kaiser window
0.5

Spectrum coverage ratio
SpecCovRatio

Number of bins
Nbins

Ratio of mean signal to RMS noise
1000

Gain tolerance
0.0005

OK Cancel Help Apply

These parameters are described below:

- Sample time - Sample time of the signal created by the measurement unit. The sample time also governs the total simulation bandwidth captured by the measurement unit.
- FFT size - Number of FFT bins used to obtain the frequency domain representation of the signals within the measurement unit.
- Beta of Kaiser window - The β parameter of the Kaiser window used in all FFT calculations within the measurement unit. Increasing β widens the mainlobe and decreases the amplitude of the sidelobes of the frequency response of the window.
- Spectrum coverage ratio - Value between 0 and 1, representing the part of total simulation bandwidth processed by the measurement unit.

- Number of bins - Number of output frequency bins in the Gain and NF signals created by the measurement unit. The FFT bins within the covered spectrum are re-distributed into those output bins. Multiple FFT bins falling into the same output bin are averaged.
- Ratio of mean signal to RMS noise - The ratio of the mean signal amplitude to the RMS noise in the DUT_in signal created by the measurement unit. A large value improves the convergence of the DUT gain calculation, but reduces the accuracy of noise calculation due to numerical inaccuracies.
- Gain tolerance - The threshold of gain variation relative to its average. When the threshold is hit, the gain is considered as converged, triggering a reset for the output noise calculation.

```
close(hFig);
bdclose(model);
clear model hLegend hsm hLine hLegend StdBlue StdYellow hLineUM hLineM hFig;
clear GammaS Zs GammaL Zl GammaA Za Ya GammaB Zb Yb;
clear unmatched_amp BW_analysis f_analysis input_match output_match matched_amp;
```

See Also

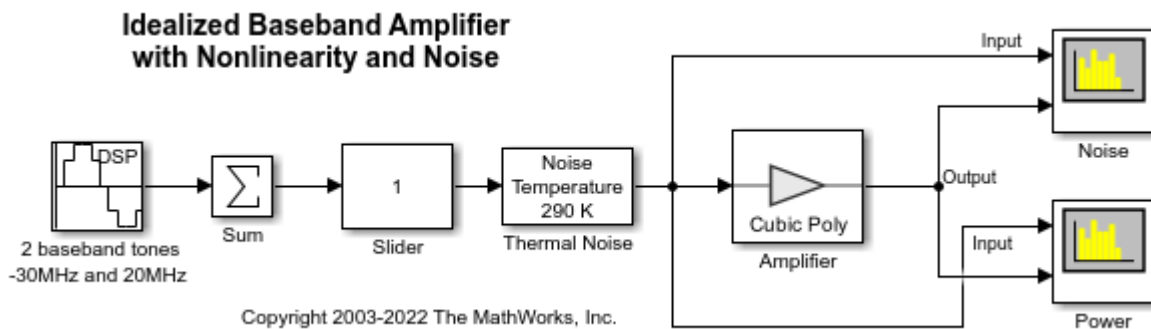
Amplifier | Configuration | Inport | Outport

Related Examples

- “RF Noise Modeling” on page 8-184
- “Model LO Phase Noise” on page 7-42
- “Spot Noise Data in Amplifiers and Effects on Measured Noise Figure” on page 7-16
- “Explicitly Simulate Resistor Thermal Noise” on page 7-5

Idealized Baseband Amplifier with Nonlinearity and Noise

The example shows how to use the idealized baseband library Amplifier block to amplify a signal with nonlinearity and noise. The Amplifier uses the Cubic Polynomial model with a Linear power gain of 10 dB, an Input IP3 nonlinearity of 30 dBm, and a Noise figure of 3 dB.



System Architecture

The DSP Sine Wave block inputs two complex baseband tones with a power level of -20 dBm and -25 dBm at frequencies of -30 MHz and 20 MHz. In this block you can also:

- Increase the samples per frame to increase the simulation speed.
- Use output complexity and phase offset to control the I-Q relationship of each baseband signal
- Control the bandwidth of the scopes using the inverse of the sample time parameter.

The Amplifier block only accepts a vector input. The Sum block combines the two baseband signals into a vector length equal to the samples per frame in the DSP Sine Wave block.

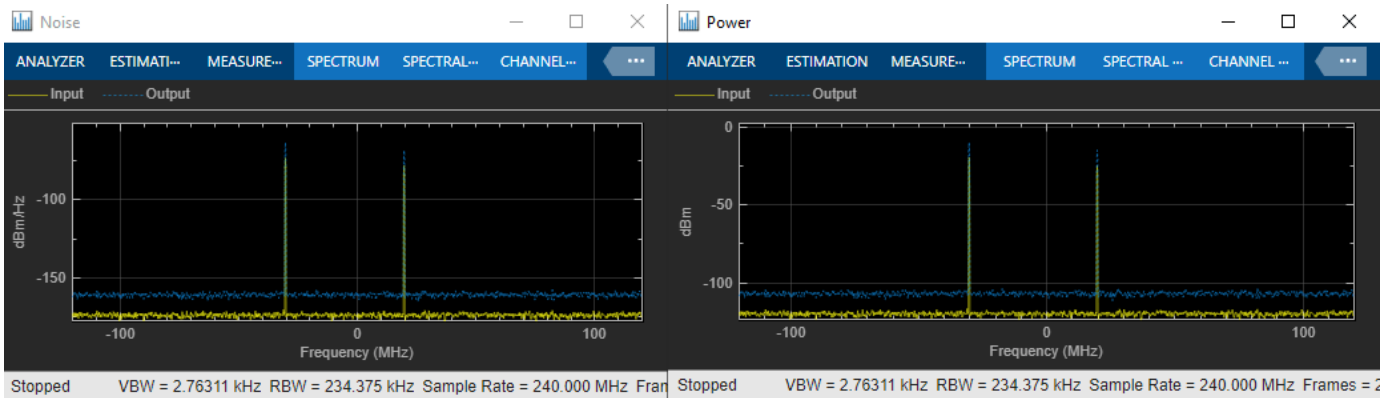
The Thermal Noise block creates a thermal noise floor input of -174 dBm/Hz.

Simulation Analysis

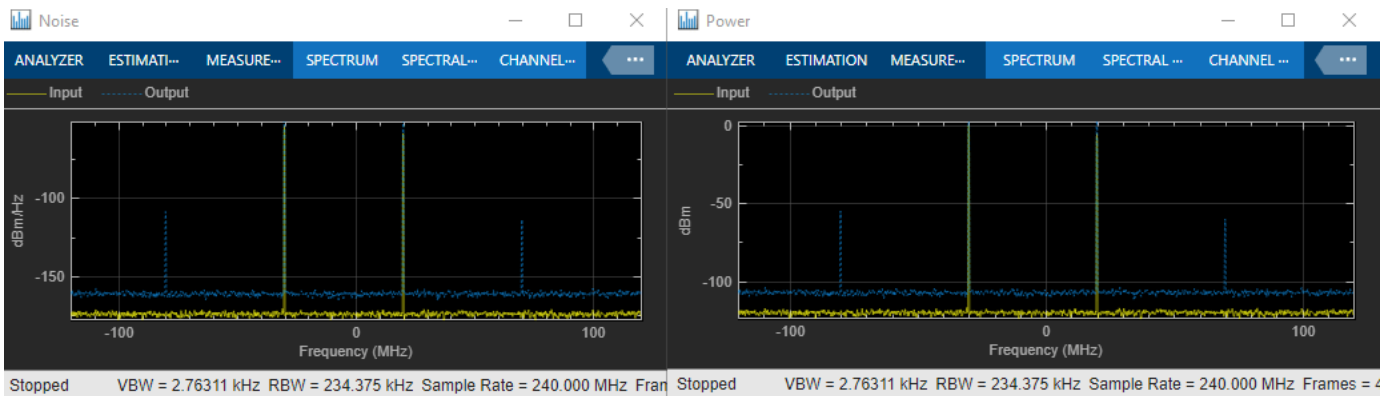
The Amplifier block with Linear power gain of 10 dB outputs tone with magnitude -10 dBm and -15 dBm as seen in the Power plot. The Amplifier also increases the thermal noise floor to -161 dBm/Hz. You can calculate the output thermal noise using:

$$\text{InputNoiseFloor} + \text{linearpowergain} + \text{NoiseFigure} = -174\text{dBm/Hz} + 10\text{dBm} + 3\text{dBm} = -161\text{dBm/Hz}$$

The following plots illustrate the differences in the input and output noise floors. The spurs appear at 70 MHz ($2 \times 20 \text{ MHz} + 30 \text{ MHz}$) and -80 MHz ($2 \times (-30 \text{ MHz}) - 20 \text{ MHz}$). This shows the third order intercept nature of the spurs.



Increasing the Slider value from 1 to 10, shows nonlinear effects in the plots. These are the Noise and Power plots when the gain of the Slider is 10.



See Also

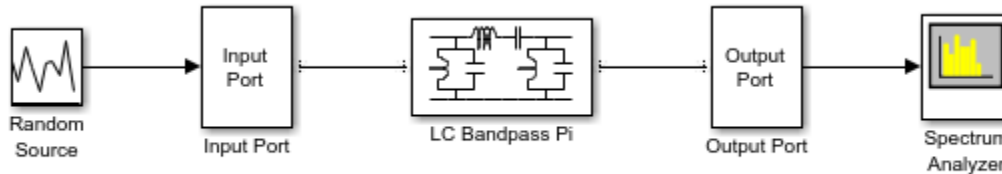
Amplifier

Related Examples

- “Modulate Quadrature Baseband Signals Using IQ Modulators” on page 7-104
- “Nonlinearities and Noise in Idealized Baseband Amplifier Block”

Use Ladder Filter Block to Filter Gaussian Noise

This example provides complex random noise in Gaussian form as input to an LC Bandpass Pi block. A DSP System Toolbox™ fallback for Spectrum Scope block plots the filtered output.



Copyright 2018-2020 The MathWorks, Inc.

The DSP System Toolbox fallback for Random Source block produces frame-based output at 512 samples per frame. Its **Sample time** parameter is set to $1.0e-9$. This sample time must match the sample time for the physical part of the model, which you provide in the Input Port block diagram.

The Input Port block specifies **Finite impulse response filter length** as 256, **Center frequency** as $700.0e6$ Hz, **Sample time** as $1.0e-9$, and **Source impedance** as 50 ohms.

Block Parameters: Input Port
✕

Input Port

Connection block from Simulink to RF Blockset physical blocks.

The RF Blockset physical blocks use a baseband-equivalent modeling technique. This technique models a bandwidth of $1/(\text{Sample time})$, centered at the specified Center frequency parameter value. This frequency value corresponds to 0 Hz in the baseband-equivalent model.

The block provides the option to interpret the Simulink signal as either the incident power wave to the RF system or the source voltage of the RF system. The 'Incident power wave' option is the most common RF modeling interpretation, while the 'Source voltage' option is provided for backwards compatibility. If the input Simulink signal is the incident power wave, the output of the RF system is the transmitted power wave. If the input is the source voltage, the output is the load voltage.

The block controls the modeling of RF Blockset physical blocks between this block and the Output Port block using:

- FIR filters to model the frequency-dependent characteristics
- Look-up tables to model the nonlinear behaviors

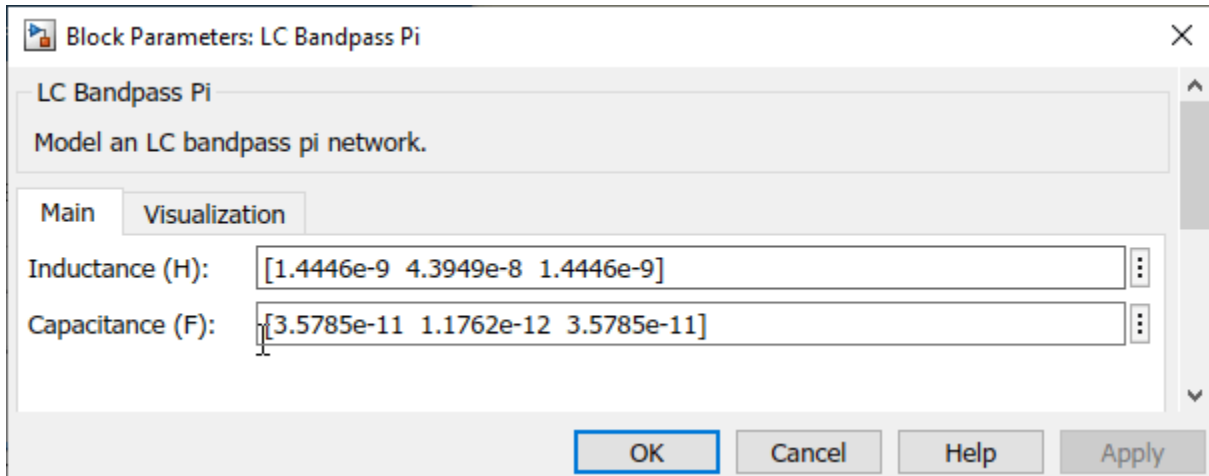
Optional guard bands can be specified as a fraction of the modeling bandwidth. The guards bands are implemented by applying a Tukey window to the frequency response. Modeling delay may be added to improve the response of the FIR filters.

Parameters

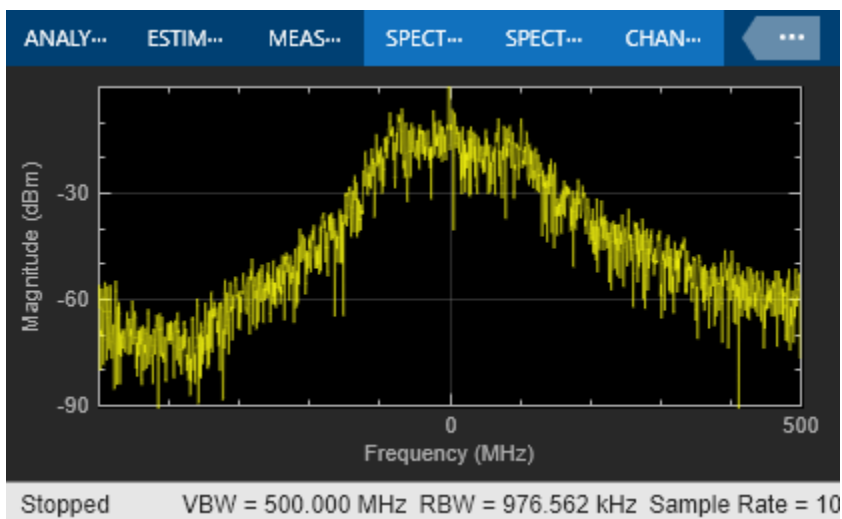
Treat input Simulink signal as:	Source voltage	▼
Source impedance (ohms):	50	⋮
Finite impulse response filter length:	256	⋮
Fractional bandwidth of guard bands:	0	⋮
Modeling delay (samples):	0	⋮
Center frequency (Hz):	700.0e6	⋮
Sample time (s):	1.0e-9	⋮
Input processing:	Elements as channels (sample based)	▼
<input checked="" type="checkbox"/> Add noise		
Initial seed:	67987	⋮

OK
Cancel
Help
Apply

The LC Bandpass Pi block provides the inductances for three inductors, in order from source to load, [1.4446e-9, 4.3949e-8, 1.4446e-9] . Similarly, it provides the capacitances for three capacitors [3.5785e-11, 1.1762e-12, 3.5785e-11] .



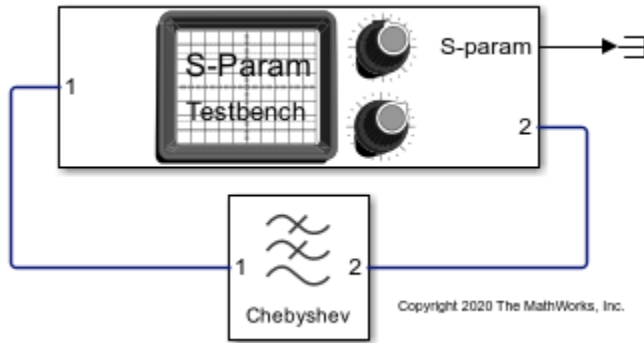
The following plot shows a sample of the baseband-equivalent RF signal generated by this LC Bandpass Pi block. Zero (0) on the frequency axis corresponds to the center frequency specified in the Input Port block. The bandwidth of the frequency spectrum is 1/sample time. You specify the Sample time parameter in the Input Port block.



The Axis Properties of the Spectrum Scope block have been adjusted to show the frequencies above and below the carrier. The **Minimum Y-limit** parameter is -90 , and **Maximum Y-limit** is 0 .

Measure S-Parameter Data of Chebyshev Filter

Use the S-Parameter Testbench block to measure S-parameter data of a ninth-order Chebyshev Tee LC low-pass filter.



Set the parameters of the Filter block as shown in this image.

Block Parameters: Filter
✕

Filter

Model an RF filter

Main

Visualization

Design method: Chebyshev

Filter type: Lowpass

Implementation: LC Tee

Implement using filter order

Filter order: 9

Passband frequency: 1 GHz

Passband attenuation (dB): 10*log10(2)

Source impedance (Ohm): 50

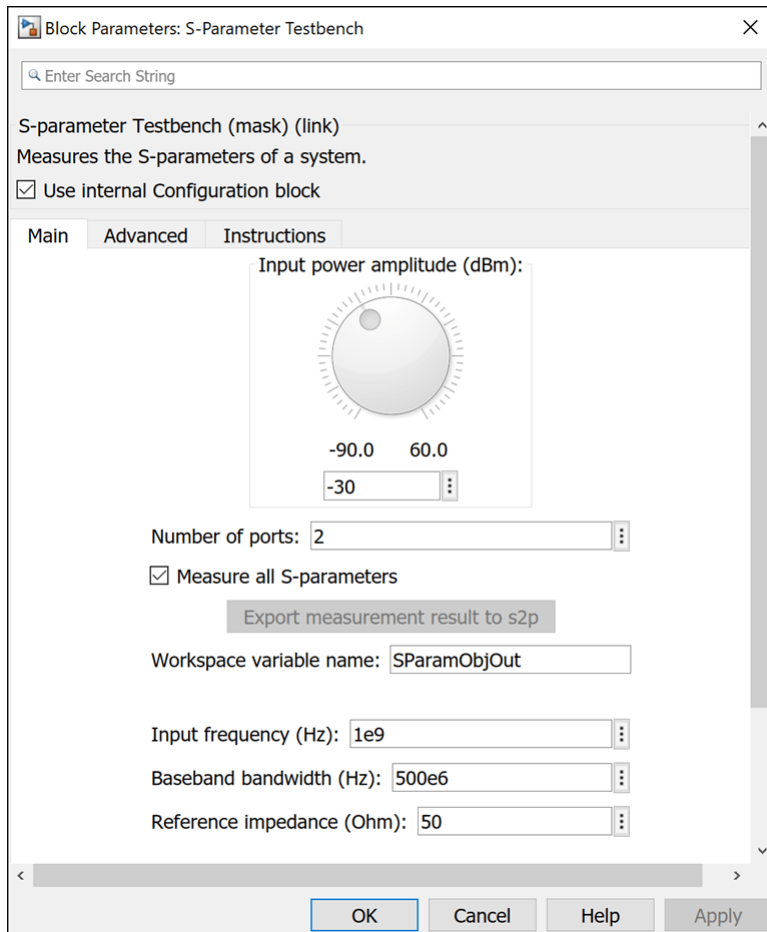
Load impedance (Ohm): 50

Ground and hide negative terminals

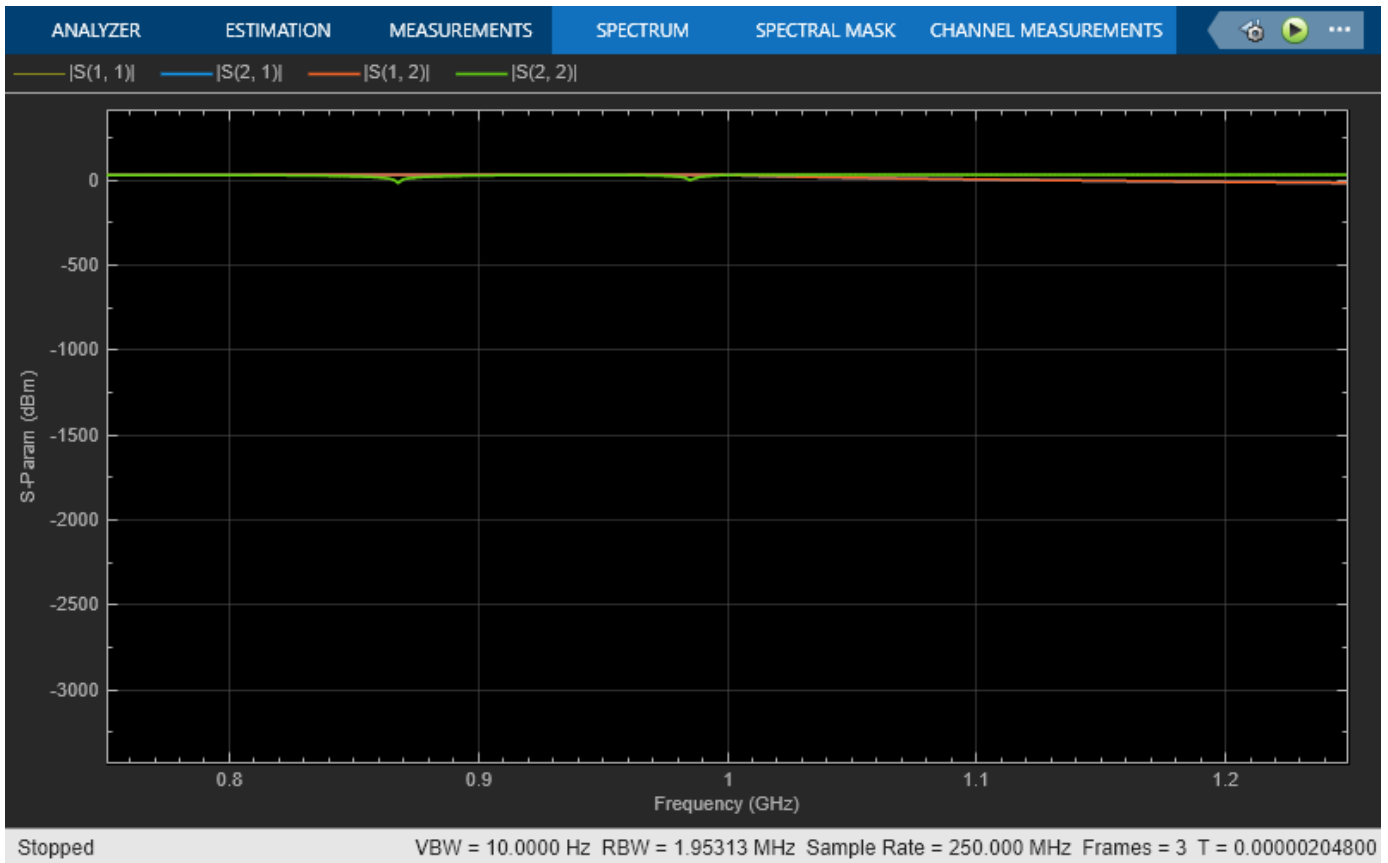
Export

OK
Cancel
Help
Apply

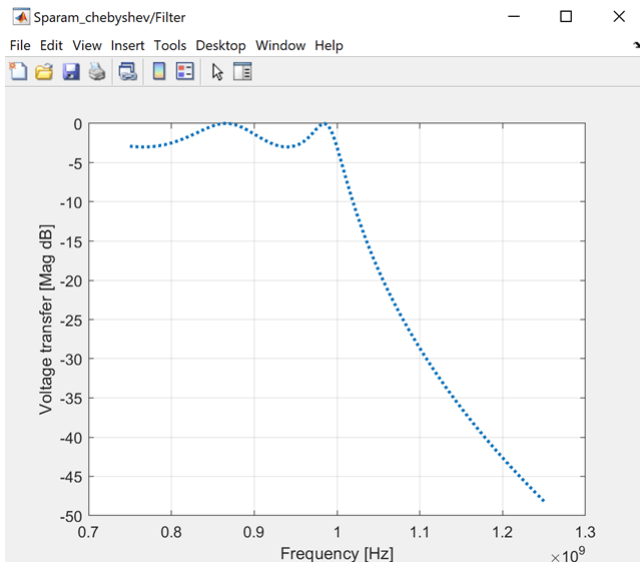
Set the parameters of the S-Parameter Testbench block to measure the S-parameters of the Chebyshev filter.



Run the simulation to yield the measured S-parameter data for the Chebyshev filter.



Note that due to the symmetry and reciprocity of the filter, the plots for S11 and S22 and S12 and S21 overlap each other. The results can be compared with the plot obtained from the **Visualization** tab of the Filter block.



In addition to viewing the results in the Spectrum Analyzer, you can export the result as an RF Toolbox™ S-parameter object or as a Touchstone® file for further processing.

See Also

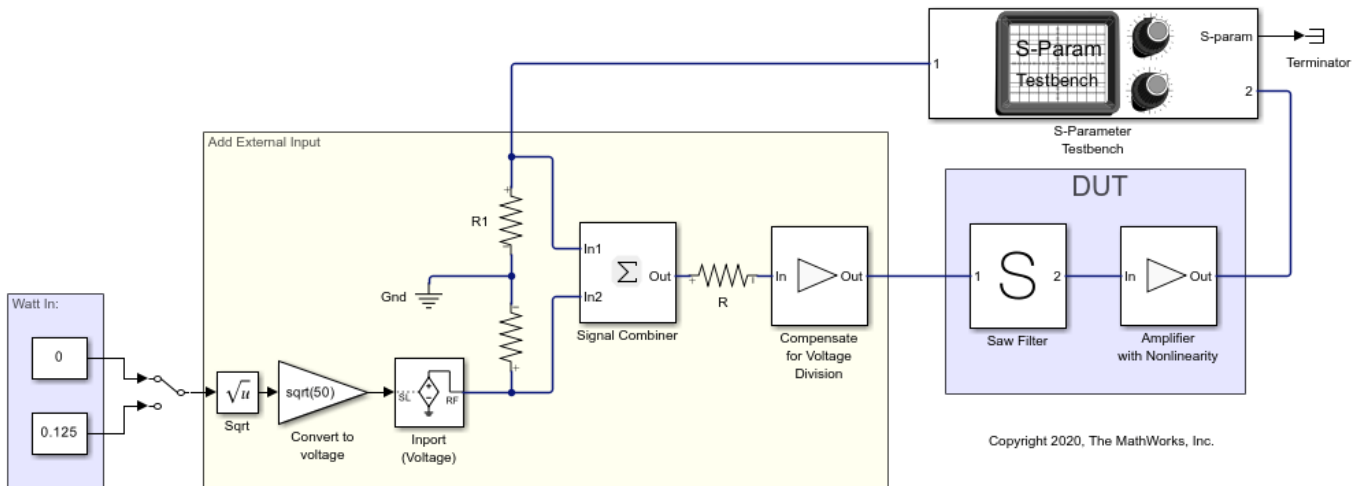
S-Parameter Testbench

Related Examples

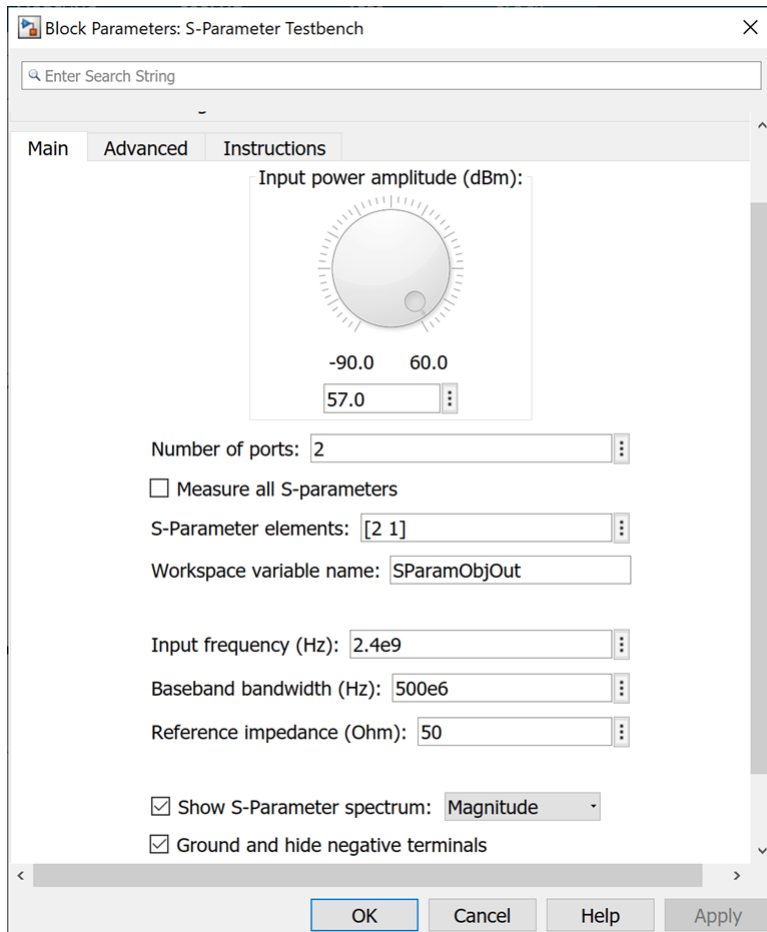
- “Measure S-Parameter of Nonlinear System” on page 7-81

Measure S-Parameter of Nonlinear System

Use the S-Parameter Testbench block to measure the S-parameters of a nonlinear system. The system shown here includes a device under test (DUT) with a linear filter followed by a nonlinear amplifier. The system allows an external steady state input that is switched off initially. When the DUT is nonlinear, the stimulus signal created in the S-Parameter Testbench block must be small enough to operate in the linear region.



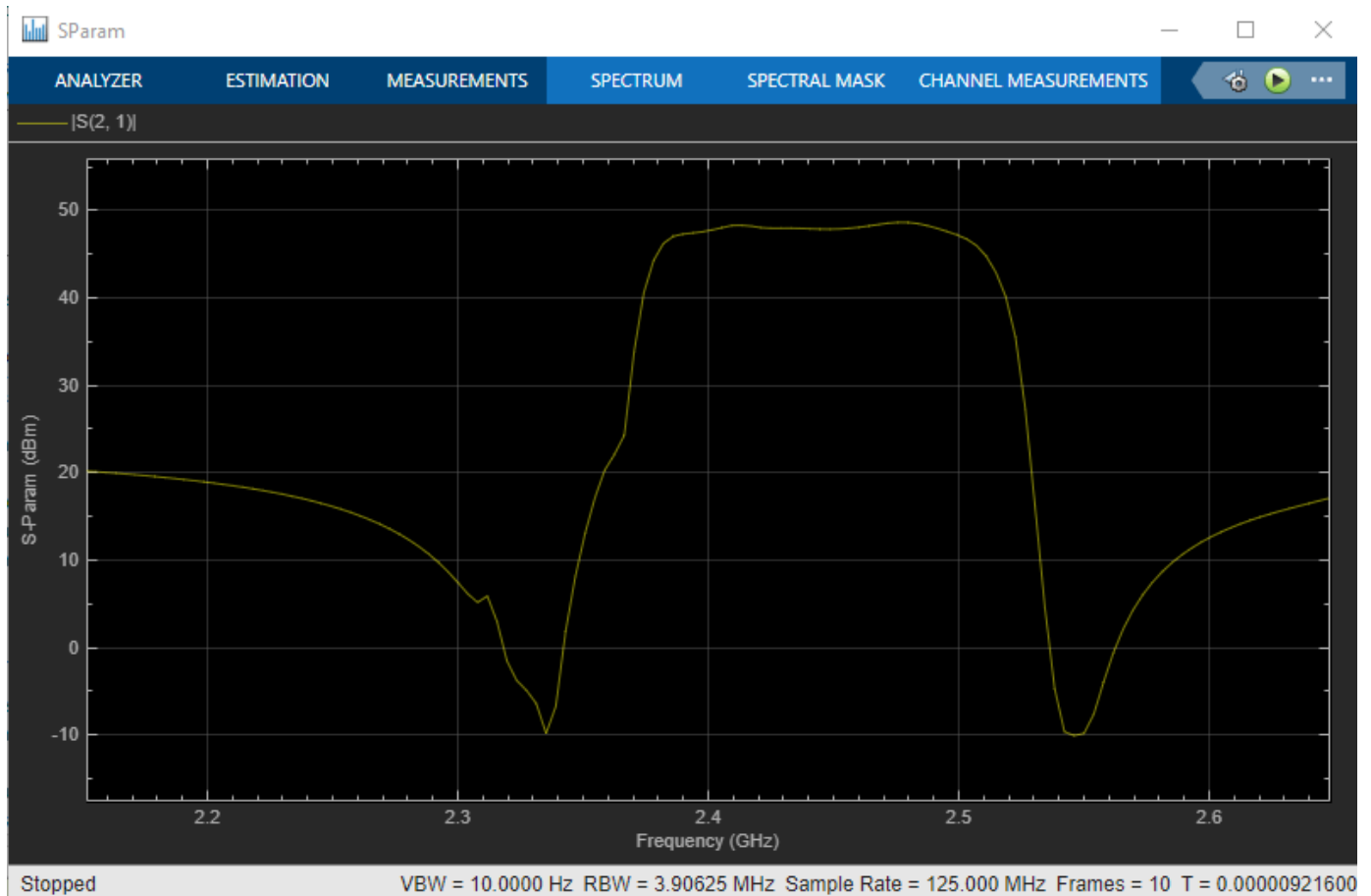
Set the S-Parameter Testbench block parameters as shown.



Select **Run** to display the output of the S-Parameter Testbench block, S21 magnitude.

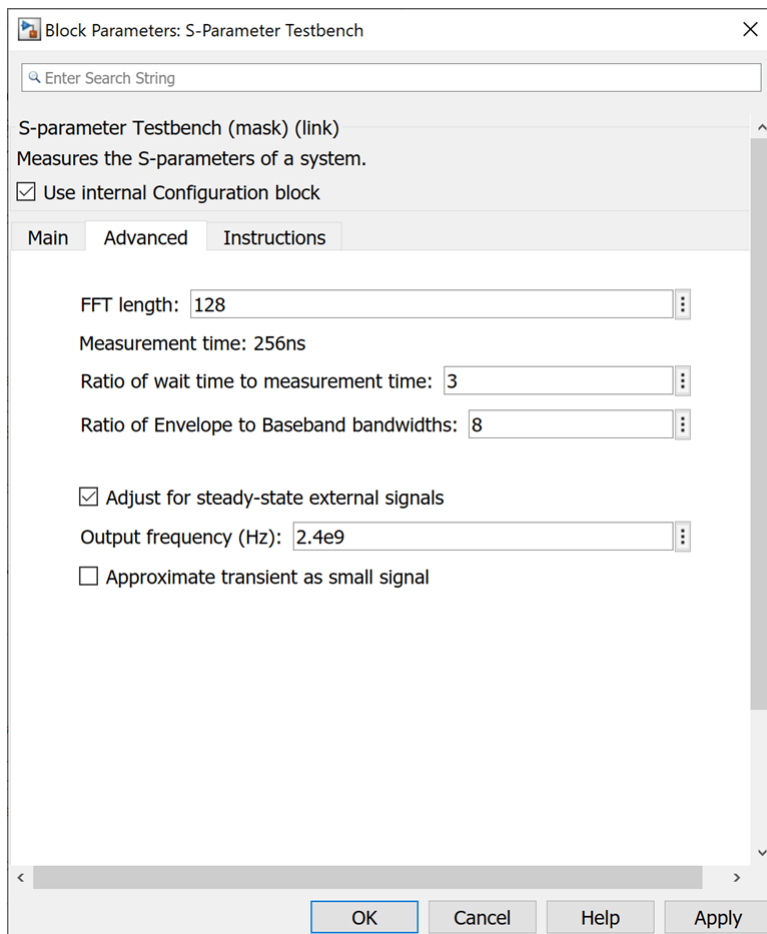


Open the mask parameters dialog box of the S-Parameter Testbench block and set the **Input power amplitude (dBm)** parameter to 15 dBm. You can also vary the **Input power amplitude (dBm)** parameter while the simulation is running. Rerun the simulation and observe the change in the magnitude of the S21 of the system.

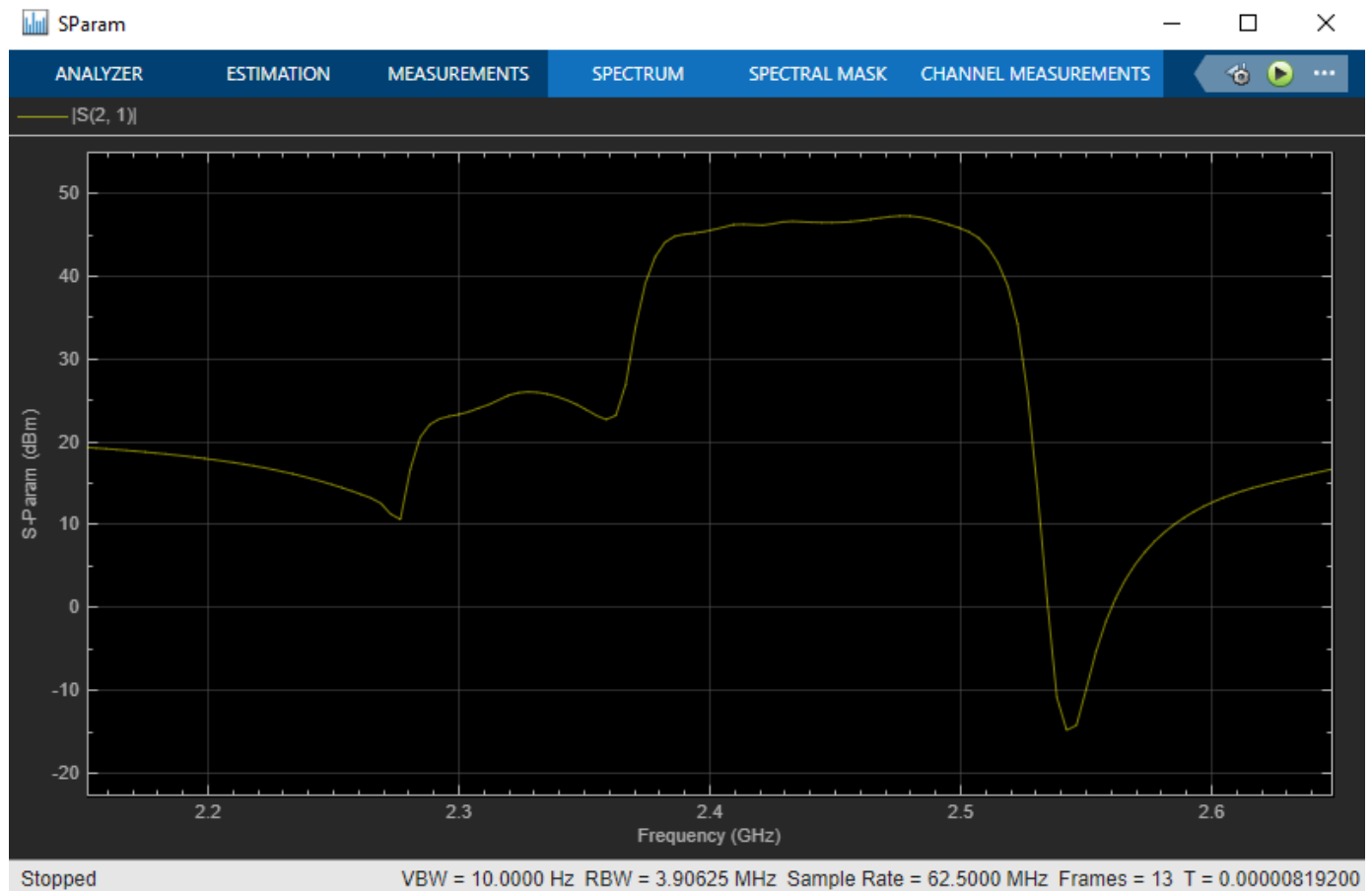


Note the difference between the results. When the input power is significant enough to excite the nonlinearity of the system, the dips in the spectrum of the filter fill up due to spectral regrowth. The simulation results are correct in both cases, but the initial S-parameter measurement is invalid due to nonlinearity.

While it is recommended that the stimulus signal be kept small when measuring S-parameter data, it is also possible to measure the S-parameter data with a small stimulus signal around a large signal operating point. To do this, set the switch to an input of 0.125 watts. In addition, in the **Advanced** tab in the block parameters dialog box of the S-Parameter Testbench block, select the **Adjust for steady-state external signals** parameter. This allows the testbench to first measure the output of the system with an external signal and then subtract the external signal from the output to ensure that only the small signal stimulus is accounted for in the calculation of the S-parameters.



Rerun the simulation and observe the S21 magnitude.



When you compare the result with a zero-watt external signal and a 0.125-watt signal, the effect of a large signal operation point is evident.

See Also

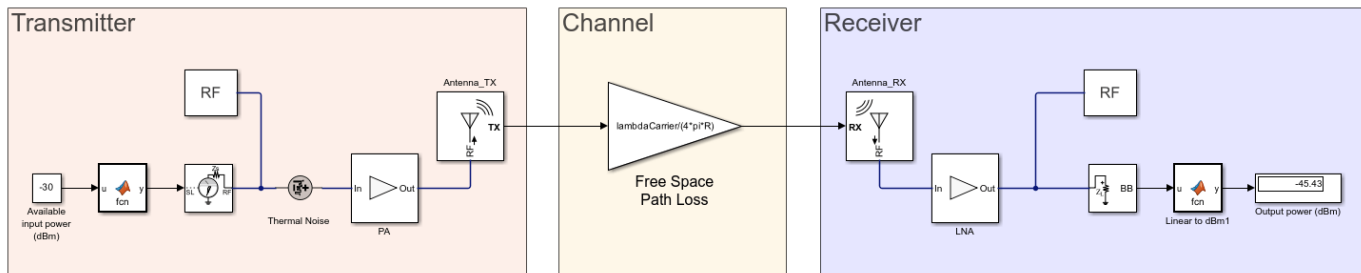
S-Parameter Testbench

Related Examples

- “Measure S-Parameter Data of Chebyshev Filter” on page 7-77

Simulation of RF Systems with Antenna Blocks

Use the Antenna block to incorporate the effect of an antenna into an RF simulation. In this model, a single tone is fed to the transmitter and the power of the received signal at the output of the receiver is calculated.



Copyright 2020 The MathWorks, Inc.

Set the **Antenna_TX** and **Antenna_RX** blocks to be isotropic radiators with the following parameters:

Block Parameters: Antenna
✕

Antenna (mask) (link)

Model antenna accounting for incident power wave (RX) and radiated power wave (TX).

Parameters

Main

Source of antenna model: Isotropic radiator

Antenna gain: Gt dBi

Impedance (Ohm): Zin_t

Input incident wave Output radiated wave

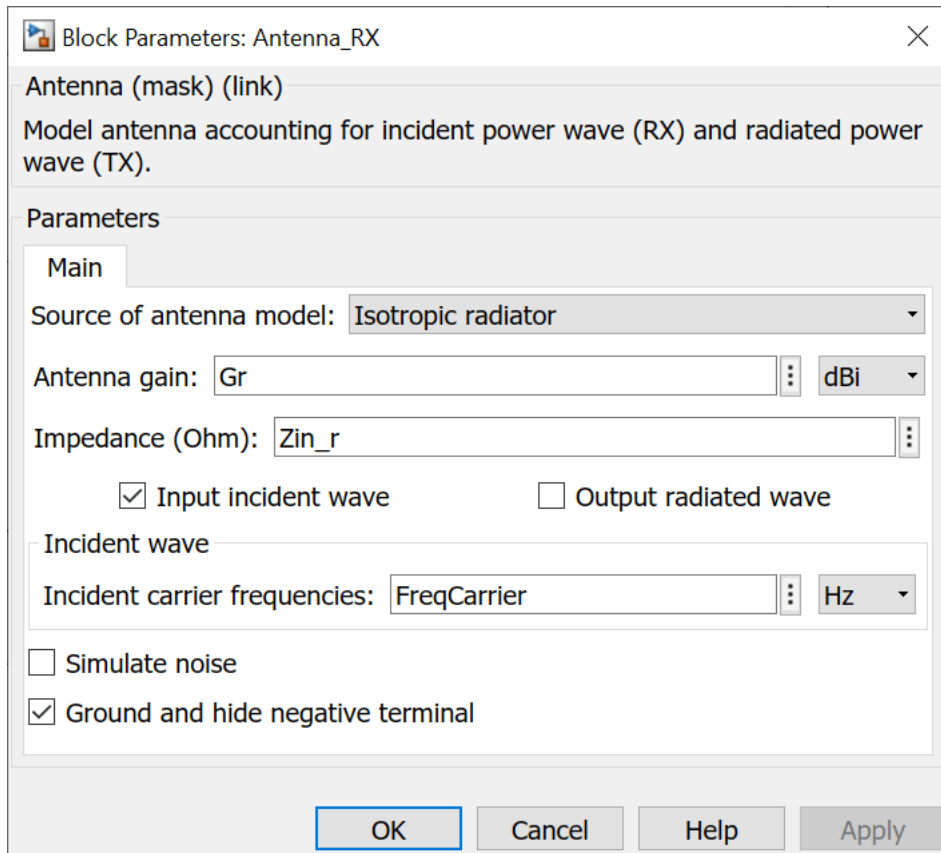
Radiated wave

Radiated carrier frequencies: FreqCarrier Hz

Simulate noise

Ground and hide negative terminal

OK
Cancel
Help
Apply

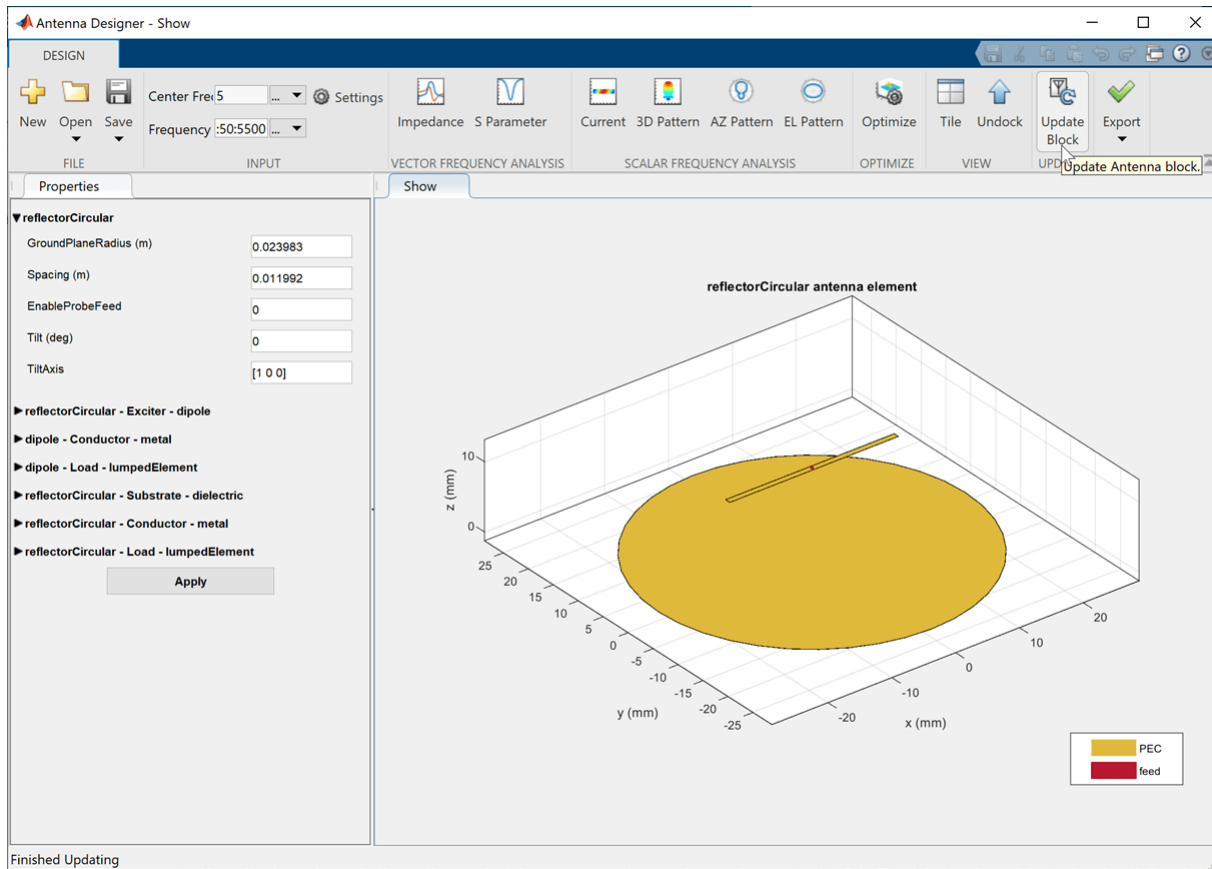


The following values are set upon loading the model:

- $R = 100$ [m]
- $\text{FreqCarrier} = 5.0$ [GHz]
- $G_t = G_r = 7.9988$ [dBi]
- $Z_{in_t} = Z_{in_r} = 56.2947 - 4.2629i$ [Ohm]

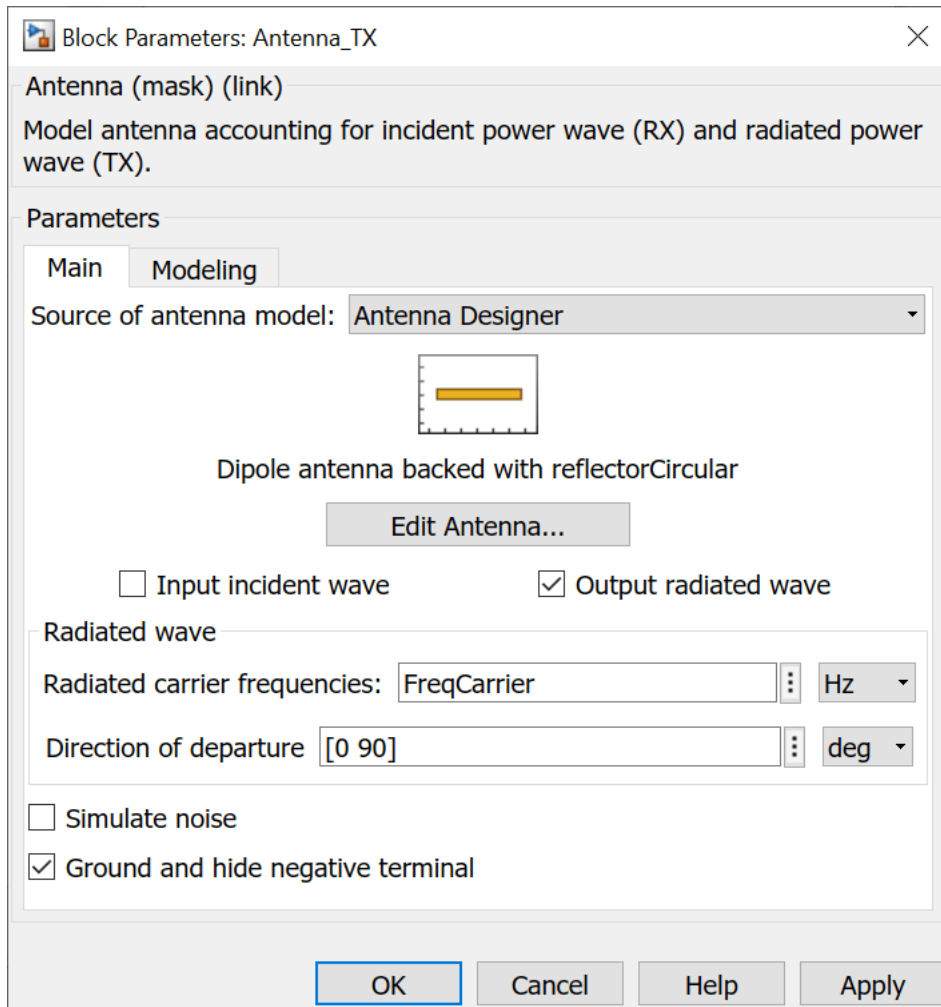
where the antenna gains and impedances were calculated beforehand from dipoles backed by circular reflectors.

With Antenna Toolbox™, it is possible to design the antenna using the **Antenna Designer** app invoked directly from the block. To do so, change the choice of **Source of the antenna model** to **Antenna Designer** and press the **Create antenna** button. Within the Antenna Designer app, create a new antenna, choose **Dipole** from the **Antenna Gallery**, **Circular** from the **Backing Structure Gallery** in the app toolstrip and select **Accept**. Note that the design frequency was prepopulated with the RF system frequency of 5 GHz.



Select the **Impedance** button in the app toolbar to analyze the structure and select the **Update Block** button to update the block with the chosen antenna. Note that the Antenna block requires that the designed antenna be analyzed for at least one frequency in the **Antenna Designer** app before updating and using it in the block.

In the **Antenna_TX** block mask parameter dialog box, change the default **Direction of departure** to 0 degrees in azimuth and 90 degrees in elevation:



Repeat the above steps to design **Antenna_RX**. However, the receiving antenna needs to be rotated to face the transmitting antenna. To do so, in the **Antenna Properties** panel of the **Antenna Designer** app, set **Tilt** to 180 degrees. Again, select the **Impedance** button and then press the **Update Block** button to update the block. In the **Antenna_RX** block mask parameter dialog box, change the **Direction of arrival** to 180 degrees in azimuth and -90 degrees in elevation. -90-degree elevation is chosen since the radiated signal that was transmitted in the positive z direction in the coordinate system of the transmitter, is now arriving from the negative z direction in the coordinate system of the receiver. Azimuth is set to 180 degrees to align vector fields of the transmitter and receiver antennas.

Run the model again, and note that the output power remained almost exactly the same. This is since the original gain and impedance values used for the isotropically radiating antenna in the beginning were calculated from the same antennas and spatial settings. However, it is now possible to change the antenna properties and observe the effect on the output power in the model. For example: Select the 'Edit Antenna' button in the **Antenna_TX** block mask parameter dialog box to reopen the **Antenna Designer** app. In the **Antenna Properties** panel of the **Antenna Designer** app, change **Tilt** to 30 degrees and the **TiltAxis** to [0 1 0]. Select the **Impedance** button and then press the

Update Block button to update the block. Rerun the model to observe reduction of 2.5 dB in the output received power due to the mismatch in antenna orientation.

See Also

Antenna

Power Amplifier Characterization

This example shows how to characterize a power amplifier (PA) using measured input and output signals of an NXP Airfast PA. Optionally, you can use a hardware test setup including an NI PXI chassis with a vector signal transceiver (VST) to measure the signals at run time.

You can use the characterization results to simulate the PA using the `comm.MemorylessNonlinearity` (Communications Toolbox) System object™ or Memoryless Nonlinearity (Communications Toolbox) block. For a PA model with memory, you can use Power Amplifier block. You can use these models to design digital predistortion (DPD) using `comm.DPD` (Communications Toolbox) and `comm.DPDCoefficientEstimator` (Communications Toolbox) System objects or DPD (Communications Toolbox) and DPD Coefficient Estimator (Communications Toolbox) blocks. For more information, see “Digital Predistortion to Compensate for Power Amplifier Nonlinearities” (Communications Toolbox).

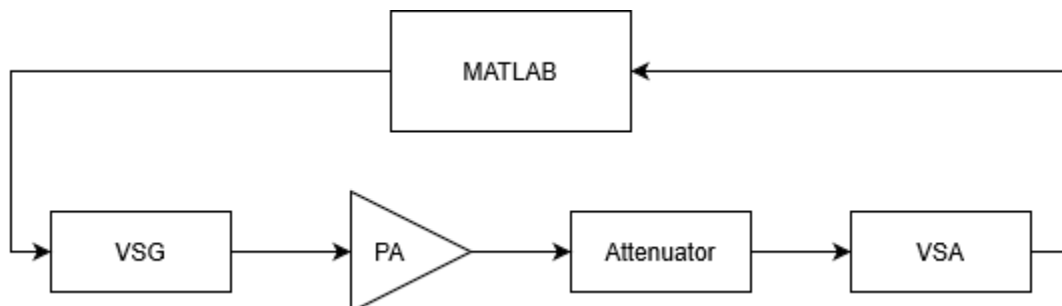
Optional Hardware and Software

This example can run on an NI PXI chassis with a VST to measure PA input and output signals during run time. The VST is a high-bandwidth RF instrument that combines a Vector Signal Generator (VSG) with a Vector Signal Analyzer (VSA). The following NI PXI chassis configuration was used to capture the saved signal:

- NI PXIe-5840 Vector Signal Transceiver (VST)
- NI PXIe-4139 Source Measure Unit (SMU)
- NI PXIe-4145 SMU
- NI RFmx SpecAn software
- NI-RFSG software
- NI-RFSG Playback Library software

As the device under test (DUT), this example uses an NXP Airfast LDMOS Doherty PA with operating frequency 3.6-3.8 GHz and 29 dB gain. This PA requires 29V, 5V, 3 V, 1.6V and 1.4V DC bias, which are provided using PXIe-4139 and PXIe-4145 SMUs.

Install MATLAB® on the NI PXI controller to run this example with the hardware setup, which is illustrated in the following figure. MATLAB, running on the PXI controller, generates test waveform and downloads the waveform to the VSG. The VSG transmits this test waveform to the PA and the VSA receives the impaired waveform at the PA output. MATLAB collects the PA output from the VSA and performs PA characterization.



Set `dataSource` variable to "Hardware" to run a test signal through the PA using the hardware setup described above. The test signal can be either a 5G-like OFDM waveform or two tones, as described in the following section. Set `dataSource` variable to "From file" to use prerecorded data.

```
dataSource =  ;
```

Generate Test Signals

To generate a test signal, specify the type of test signal as "OFDM" or "Tones". Specifying the `testSignal` as "OFDM" uses a 5G-like OFDM waveform with 64-QAM modulated signals for each subcarrier. "Tones" uses two tones at 1.8 MHz and 2.6 MHz, to test the intermodulation caused by the PA.

The example will use an oversampling factor of 7 to run the grid search up to an expected seventh-order nonlinearity, and normalize the waveform amplitude.

```
testSignal =  ;
switch testSignal
case "OFDM"
    bw =  ;
    [txWaveform,sampleRate,numFrames] = helperPACCharGenerateOFDM(bw);
case "Tones"
    bw = 3e6;
    [txWaveform,sampleRate,numFrames] = helperPACCharGenerateTones();
end
txWaveform = txWaveform/max(abs(txWaveform)); % Normalize the waveform
```

Hardware Test

If the `dataSource` variable is set to "From file", load the prerecorded data. If the `dataSource` variable is set to "Hardware", run the test signal through the PA using the VST. Create a `helperVSTDriver` object to communicate with the VST device. Set the resource name to the resource name assigned to the VST device. This example uses 'VST_01'. For NI devices, you can find the resource name using the NI Measurement & Automation Explorer (MAX) application.

```
if strcmp(dataSource, "Hardware")
    VST = helperVSTDriver('VST_01');
```

Set the expected gain values of the DUT and the attenuator. Since PA output is connected to a 30 dB attenuator, set VSA external attenuation to 30. Set the expected gain of the DUT to 29 dB and gain accuracy to 1 dB. Set the acquisition time to a value that will result in about 40k samples. Set the target input power to 8 dBm. You can increase this value to drive the PA more into the non-linear region.

```
VST.DUTExpectedGain      = 29;      % dB
VST.ExternalAttenuation  = 30;      % dB
VST.AcquisitionTime     = 0.9e-3*(53.76e6/sampleRate); % seconds

VST.DUTTargetInputPower  = 8  ; % dBm
VST.CenterFrequency     = 3.7e9    % Hz
```

Download the test waveform to the VSG. Measure PA output.

```
writeWaveform(VST,txWaveform,sampleRate,testSignal)
results = runPAMeasurements(VST);
```

```

release(VST)
else
% Load the prerecorded results from VST
switch testSignal
case "OFDM"
dataFileName = sprintf("helperPACharSavedData%dMHz",bw/1e6);
case "Tones"
dataFileName = "helperPACharSavedDataTones";
end
load(dataFileName,"results","sampleRate","overSamplingRate","testSignal","numFrames")
end

```

Map results into local variables.

```

referencePower = results.ReferencePower;
measuredAMToAM = results.MeasuredAMToAM;
paInput = results.InputWaveform;
paOutput = results.OutputWaveform;
linearGaindB = results.LinearGain;

```

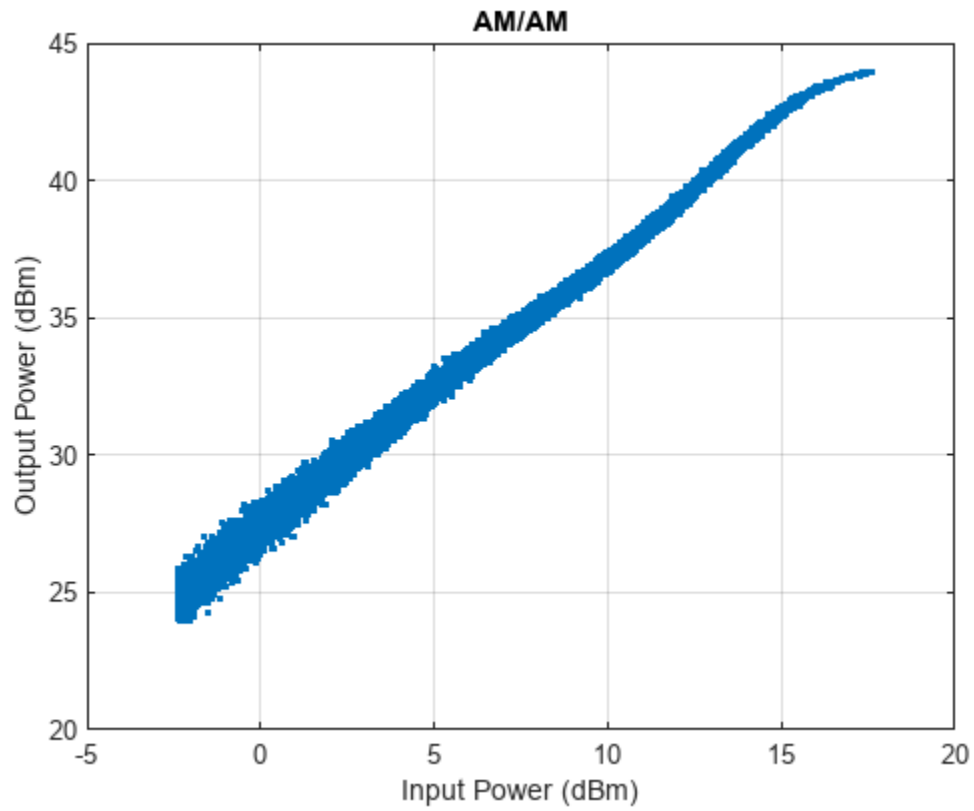
Plot the spectrum of the test signal using the spectrumAnalyzer (DSP System Toolbox) function.

```
saInput = helperPACharPlotInput(paInput, sampleRate, testSignal, bw);
```



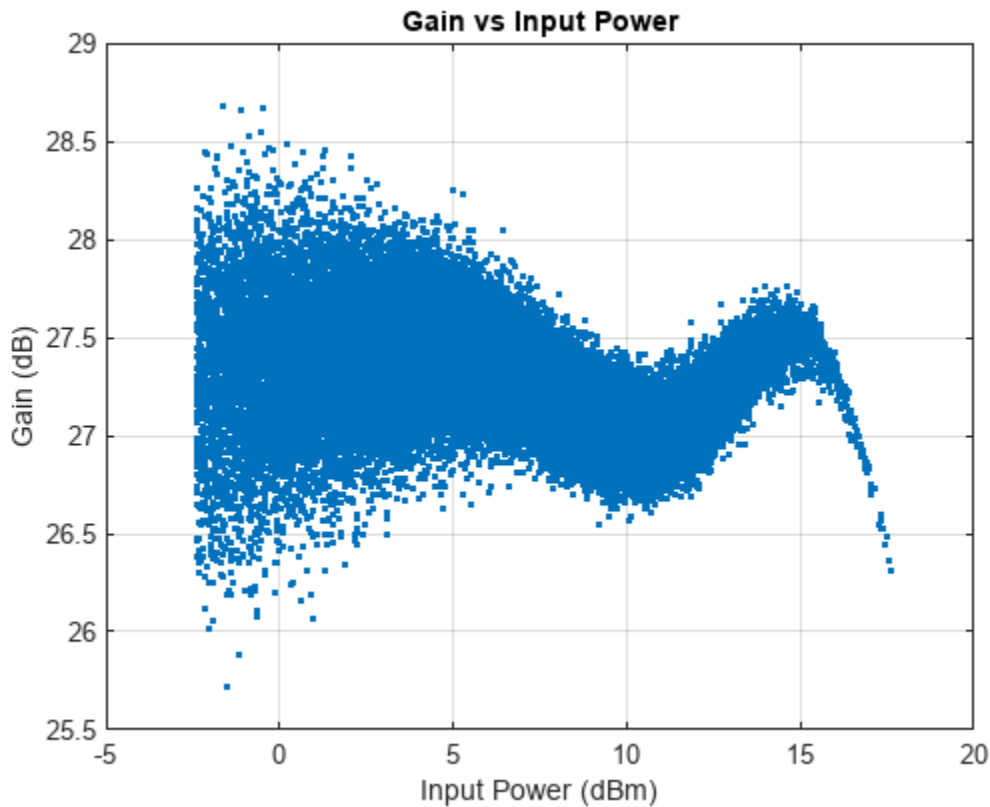
Plot the AM/AM characteristics of the PA.

```
helperPACharPlotSpecAnAMAM(referencePower, measuredAMToAM)
```



For a better view, focus on gain vs input power instead of output power vs input power and plot again.

```
helperPACCharPlotSpecAnGain(referencePower, measuredAMToAM)
```



The PA is mostly linear of the input power range -1 to 17 dBm, with only about 1dB variation over that range. The width of the gain curve is due to the memory effects of the PA.

PA Characterization

Use the measured PA input and output data to model the PA. Then, you can use this model to simulate a system that contains this PA and fine tune the parameters. This example considers three models: memoryless nonlinearity, memory polynomial and memory polynomial with cross terms.

Memoryless Nonlinearity Model

Memoryless nonlinear impairments distort the input signal amplitude and phase. The amplitude distortion is amplitude-to-amplitude modulation (AM/AM) and the phase distortion is amplitude-to-phase modulation (AM/PM). The `comm.MemorylessNonlinearity` (Communications Toolbox) System object and Memoryless Nonlinearity (Communications Toolbox) block implements several such distortions. Use the PA input and output data to create a lookup table to use with this object or block.

To characterize the AM/AM transfer function, calculate the average output power for a range of input power values. Measurements are in volts over an overall 100 ohm impedance, split between the transmitter and receiver. Convert the measured baseband samples to power values in dBm. The +30 dB term is for dBW to dBm conversion and the -20 dB term is for the 100 ohm impedance.

```
paInputdBm = mag2db(abs(paInput)) + 30 - 20;
paOutputdBm = mag2db(abs(paOutput)) + 30 - 20;
```

Partition the input power values into bins. The `edges` variable contains the bin edges, and the `idx` variable contains the index of the bin values for each input power value.

```
[N,edges,idx] = histcounts(paInputdBm, 'BinWidth', 0.5);
```

For each bin, calculate the midpoint of the bin, average output power and average phase shift. Do not include any input power value that is less than 20 dB below the maximum input power. Store the results in a three-column matrix where the first column is the input power in dBm, second column is the output power in dBm and last column is the phase shift.

```
minInPowerdBm = max(paInputdBm) - 20;
minIdx = find(edges < minInPowerdBm, 1, 'last');
tableLen = length(edges)-minIdx-1;
inOutTable = zeros(tableLen,2);
for p = minIdx+1:length(edges)-1
    inOutTable(p-minIdx,1) = mean(paInputdBm(idx == p)); % Average input power for current bin
    inOutTable(p-minIdx,2) = mean(paOutputdBm(idx == p)); % Average output power for current bin
    inOutTable(p-minIdx,3) = mean(angle(paOutput(idx == p)./paInput(idx == p))); % Average phase shift
end
```

Use the table in the `comm.MemorylessNonlinearity` System object to model the PA. Compare the estimated output with the actual output.

```
pa = comm.MemorylessNonlinearity('Method','Lookup table','Table',inOutTable,'ReferenceImpedance',100);
```

```
pa =
comm.MemorylessNonlinearity with properties:
```

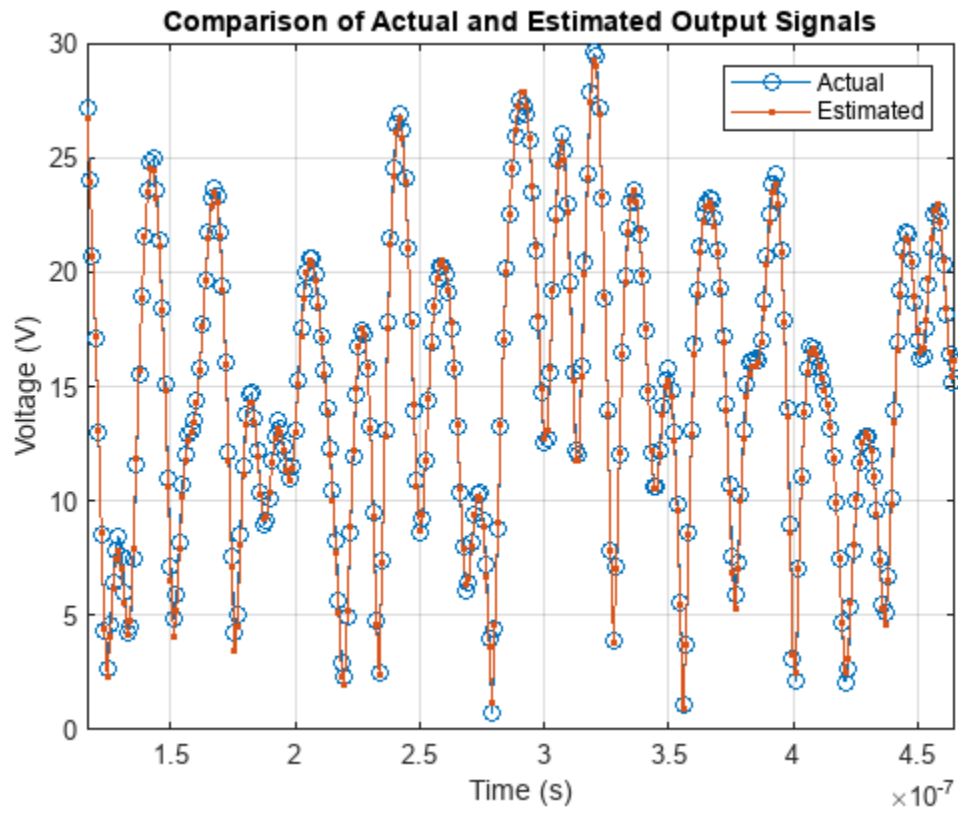
```
Method: 'Lookup table'
Table: [40x3 double]
ReferenceImpedance: 100
```

```
paOutputFitMemless = pa(paInput);
err = abs(paOutput - paOutputFitMemless)./abs(paOutput);
rmsErrorMemless = rms(err)*100;
disp(['Percent RMS error in time domain is ' num2str(rmsErrorMemless) '%'])
```

```
Percent RMS error in time domain is 12.1884%
```

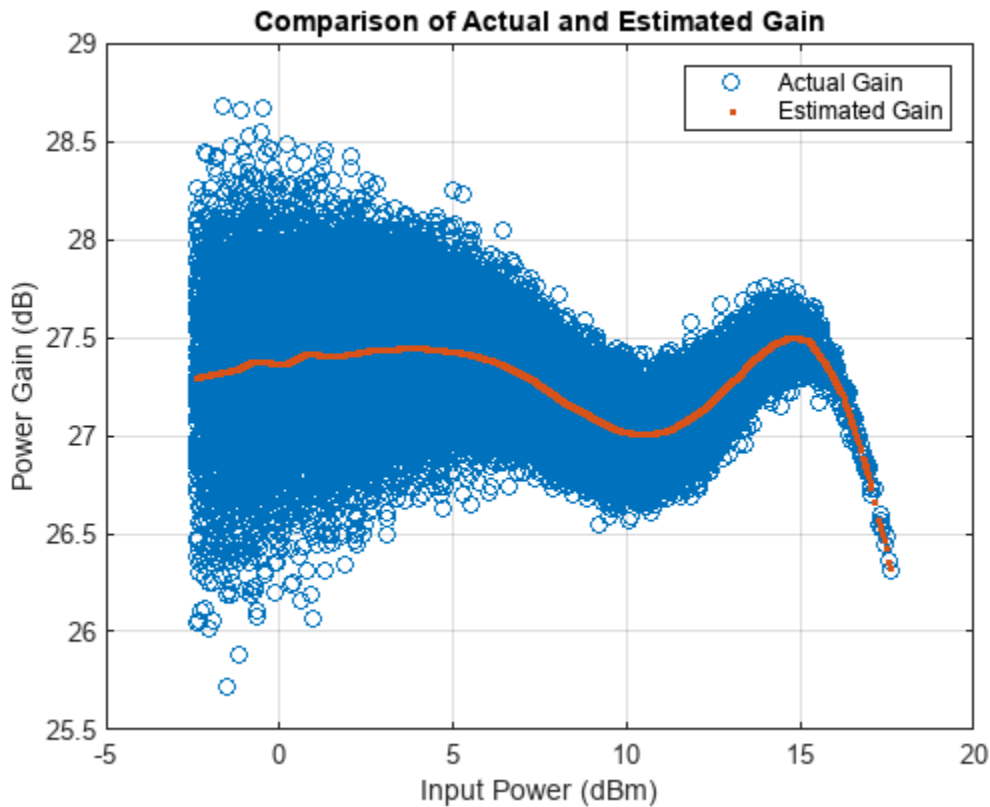
To visualize both the measured output signal and the fitted output signal, plot the actual and fitted time-domain output voltages.

```
helperPACCharPlotTime(paOutput, paOutputFitMemless, sampleRate)
```



Plot the magnitude of the gain.

```
helperPCharPlotGain(paInput, paOutput, paOutputFitMemless)
```



Memory Polynomial Model

The memory polynomial model includes the memory effects of the PA in addition to the nonlinear gain. Use the multipurpose helper function `helperPACCharMemPolyModel` to determine the complex coefficients of a memory polynomial model for the amplifier characteristics. Set the model type to 'Memory Polynomial'.

```
modType =  ;
```

Perform a grid search as shown in Appendix Grid Search for Memory Length and Polynomial Order on page 7-102. Based on this grid search results, the best fit is obtained when memory length and polynomial degree values are as follows:

```
memLen = 5;
degLen = 5;
```

Perform the fit and RMS error calculation for these values. Only half of the data is used to compute the fitting coefficients, as the whole data set will be used to compute the relative error. The helper function `helperPACCharMemPolyModel` calculates the coefficients of the model.

```
numDataPts = length(paInput);
halfDataPts = round(numDataPts/2);
```

The helper function `helperPACCharMemPolyModel` is editable for custom modifications, and to return the desired matrix. The PA model has some zero valued coefficients, which results in a rank deficient matrix.

```
fitCoefMatMem = helperPACCharMemPolyModel('coefficientFinder', ...
    paInput(1:halfDataPts), paOutput(1:halfDataPts), memLen, degLen, modType);
```

```
Warning: Rank deficient, rank = 24, tol = 1.870608e-01.
```

```
disp(abs(fitCoefMatMem))
```

```
    23.1549    8.8540   17.8385   13.3026    3.2168
         0    11.7686   26.4685   23.1937    5.5476
    20.9745   16.8531   25.7336   22.1925    5.0688
    32.6199    8.4042    9.4903   10.6984    2.5613
    15.3875    2.3630    2.0867    2.9339    0.7370
```

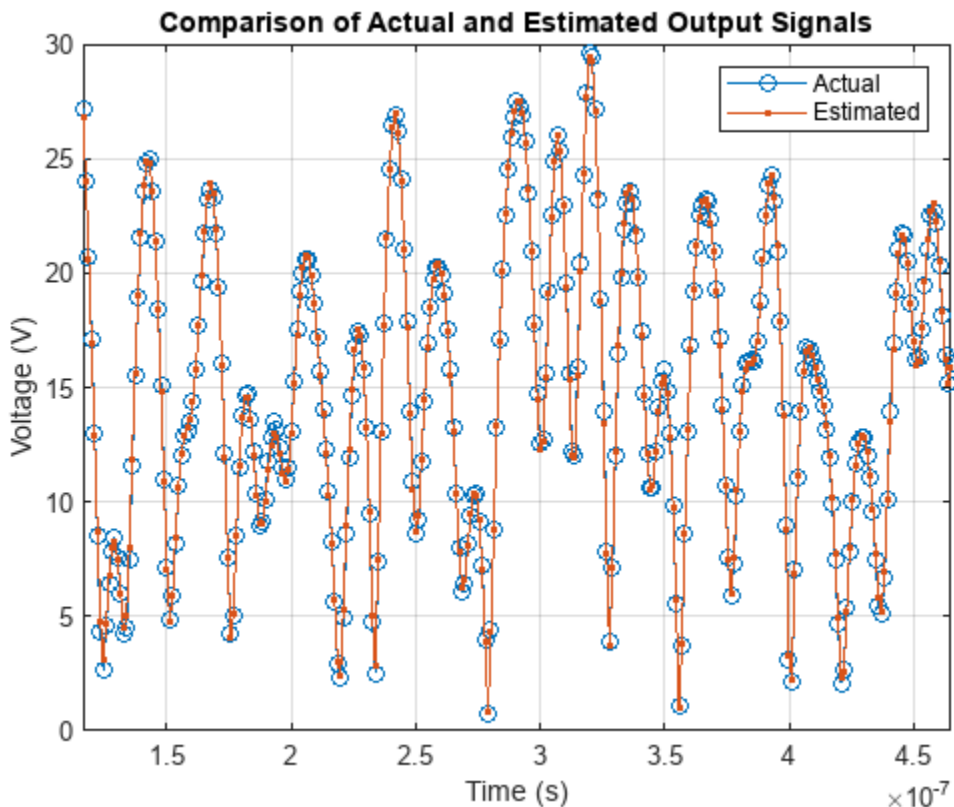
To validate the fitting, use the helper function to compute percent RMS error with respect to the measured signal.

```
rmsErrorTimeMem = helperPACCharMemPolyModel('errorMeasure', ...
    paInput, paOutput, fitCoefMatMem, modType);
disp(['Percent RMS error in time domain is ' num2str(rmsErrorTimeMem) '%'])
```

```
Percent RMS error in time domain is 6.1056%
```

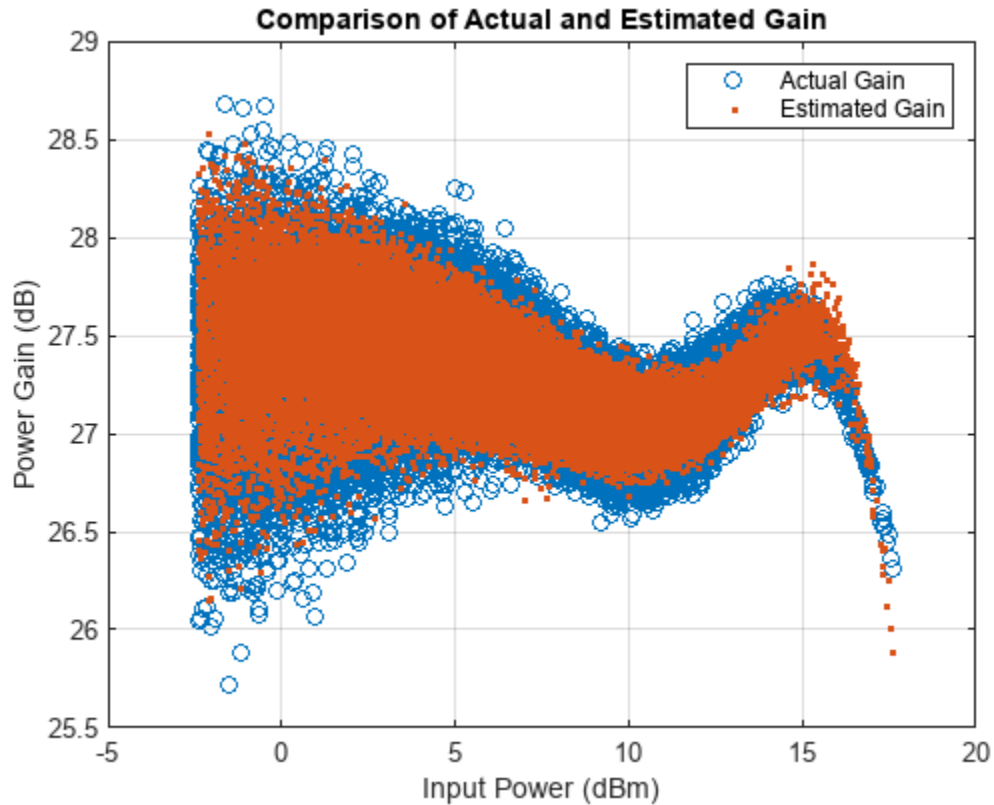
To visualize both the measured output signal and the fitted output signal, plot the actual and fitted time-domain output voltages.

```
paOutputFitMem = helperPACCharMemPolyModel('signalGenerator', ...
    paInput, fitCoefMatMem, modType);
helperPACCharPlotTime(paOutput, paOutputFitMem, sampleRate)
```



Plot the magnitude of the gain.

```
helperPACharPlotGain(paInput, paOutput, paOutputFitMem)
```

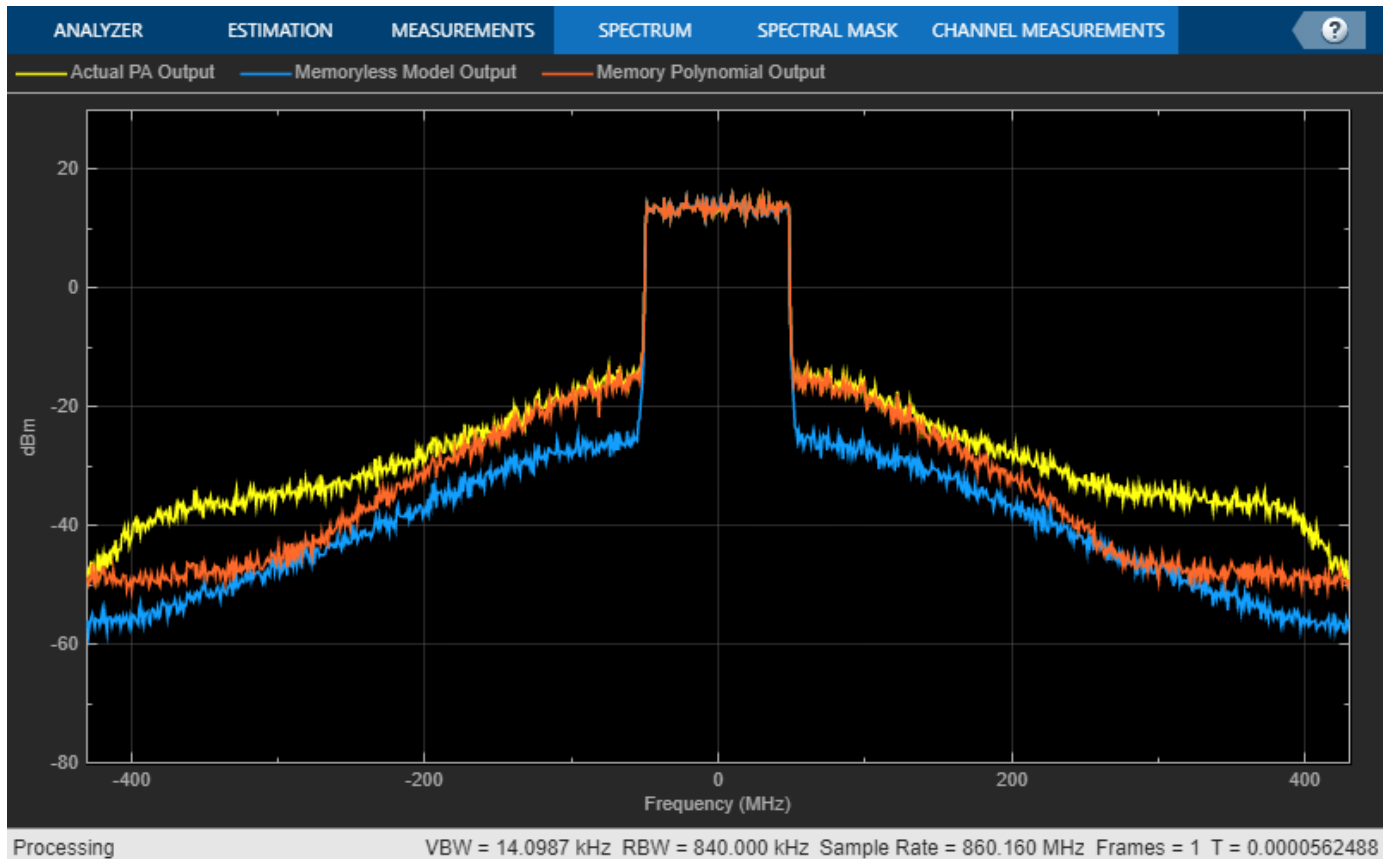


Discussions

The percent RMS estimation error in time domain for the memoryless nonlinearity model, which is between 9% and 13%, is about 3 to 4 times more than the error for the memory polynomial model is, which is between 2% and 6%, for the OFDM signals with different bandwidths.

Check the estimation error in frequency domain by plotting the spectrum of the actual PA output together with the spectrum of the estimated PA output for all three models. The memoryless nonlinearity table lookup model is not able to simulate the spectral growth seen in the measured PA output. For this PA, memory polynomial model provides a good approximation of the PA characteristics.

```
sa = helperPACharPlotSpectrum(...
    [paOutput paOutputFitMemLess paOutputFitMem],...
    {'Actual PA Output', 'Memoryless Model Output', ...
    'Memory Polynomial Output'},...
    sampleRate, testSignal);
```



The helper function `helperPACCharMemPolyModel` can also use the memory polynomial with cross terms model, which includes the leading and lagging memory cross terms in addition to the memory effects of the PA and the nonlinear gain. Set the model type to 'Cross-Term Memory' to explore this model.

For further exploration, try different memory length and polynomial degree combinations. Modify the oversampling factor and explore its effect on the PA model performance. Modify the helper function `helperPACCharMemPolyModel` to try different PA models.

Using PA Model for DPD Testing

Save the coefficient matrix of the PA model to be used in the Power Amplifier block for simulation at the system-level in the “Digital Predistortion to Compensate for Power Amplifier Nonlinearities” (Communications Toolbox).

```
frameSize = floor(length(paInput)/numFrames);
paIn.signals.values = double(reshape(paInput(1:frameSize*numFrames,1),numFrames,frameSize));
paIn.signals.dimensions = frameSize;
paIn.time = [];
save('PAcoefficientsAndInput.mat','modType','fitCoefMatMem','memLen','degLen','paIn','linearGain')
```

Appendix: Grid Search for Memory Length and Polynomial Order

Uncomment following lines to perform the grid search when the cost function is the percent RMS error in time. First choose the model type.

```
modType =  ;  
% rmsErrorTime = helperPACharGridSearchTime(paInput,paOutput,modType,overSamplingRate)
```

Repeat the search when the cost function is the percent RMS error in frequency.

```
% rmsErrorFreq = helperPACharGridSearchFrequency(paInput,paOutput,modType,overSamplingRate)
```

See Also

Related Examples

- “Digital Predistortion to Compensate for Power Amplifier Nonlinearities” on page 8-78
- “Power in Simulink Sources and Signals” on page 8-108
- “Power Ports and Signal Power Measurement in RF Blockset” on page 8-11

Modulate Quadrature Baseband Signals Using IQ Modulators

This example shows how to modulate quadrature baseband signals using two different RF Blockset™ blocks. You can use either an idealized baseband Mixer block or a circuit envelope IQ Modulator block in your model to modulate quadrature baseband signals to the RF level. Observe the impairments in the modulated output signal due to gain imbalance, third-order intercepts (OIP3) and system noise in the complex output power density and output power spectrum analyzers.

The input in-phase and quadrature baseband signals are each composed of two tones at 10 MHz and 15 MHz and the power of each baseband signal is -30 dBm. When using a circuit envelope IQ Modulator block to modulate the signal, you must use Inport and Outport blocks to set the input and output carrier frequencies. The explicit noise sources in the model set the noise floor in each signal branch.

Set Parameters

For idealized baseband Mixer block

- **Mixer type** — IQ Modulator
- **Mixer sideband** — Upper
- **Conversion gain** — 10 dB
- **IIP3 (dBm)** — 10 dBm
- Select **Include mixer noise**
- ***I/Q gain imbalance (dB)** — 0.1 dB
- **Noise figure (dB)** — 3.9752 dB

The **Noise figure (dB)** is calculated using this equation

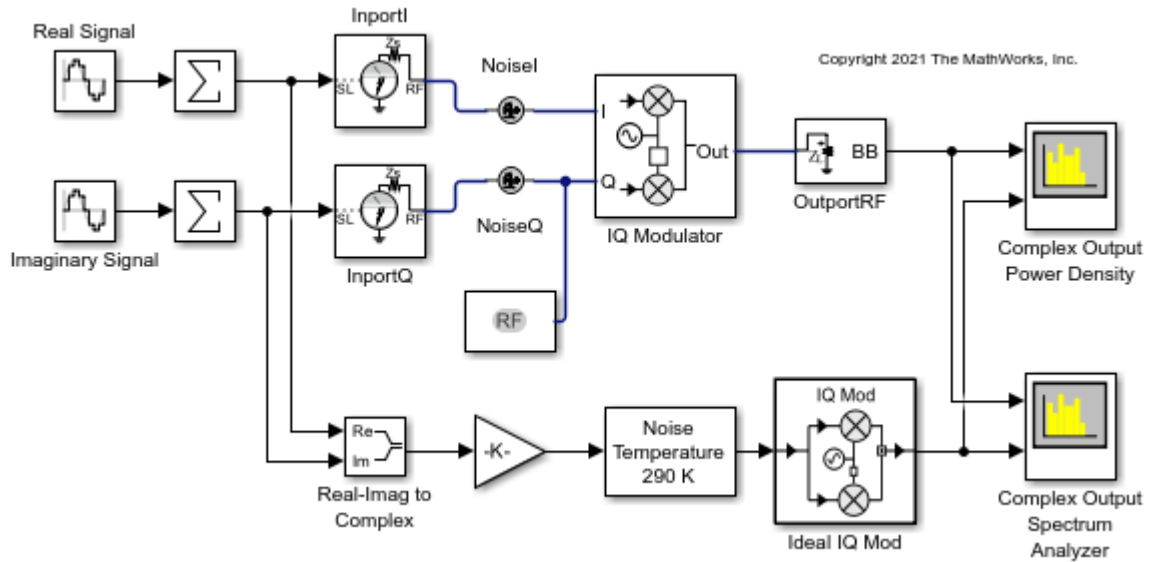
$$(Room\ Temperature\ Noise\ Floor, Boltmanns\ \times\ 290) - dB_to_dBm - CE_IQModulator_Noise_Floor - IDBB_Mixer_Conversion_Gain$$

$$= 10 \times \log_{10}(1.3806452e-23 \times 290) - 30 - 160 - 10 = 3.9752\ dB$$

For circuit envelope IQ Modulator block

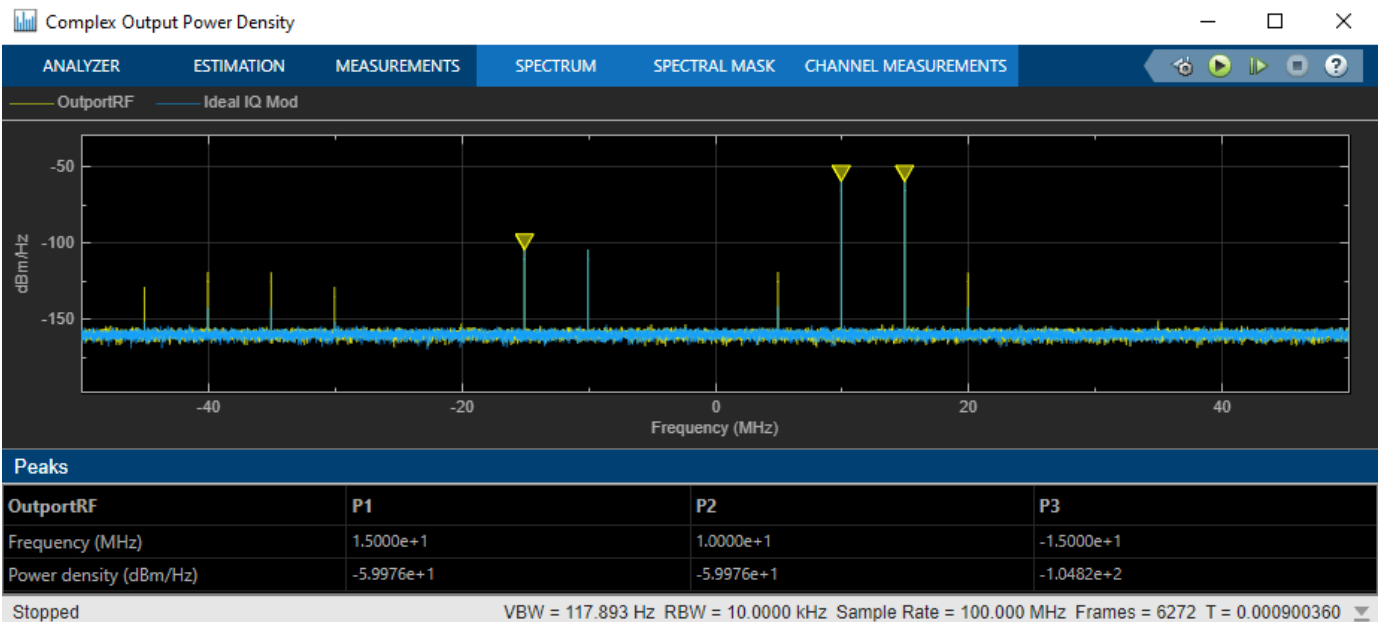
- **Available power gain** — 10 dB
- **Local oscillator frequency** — 2 GHz
- **IP3** — 10 dBm
- **I/Q gain mismatch** — 0.1 dB
- **LO to RF Isolation** — inf dB
- **Noise Floor** — -160 dBm/Hz
- Select **Add Image Reject filter**

Since the input signals are baseband signals, the inport blocks **Carrier frequencies** are set to 0 and the input signal power is interpreted correctly. For the Ideal Baseband Mixer branch, a Gain block with **Gain** equal to $1/(\sqrt{2})$ is required. If the input signals are complex baseband, then the gain is set to $1/2$.



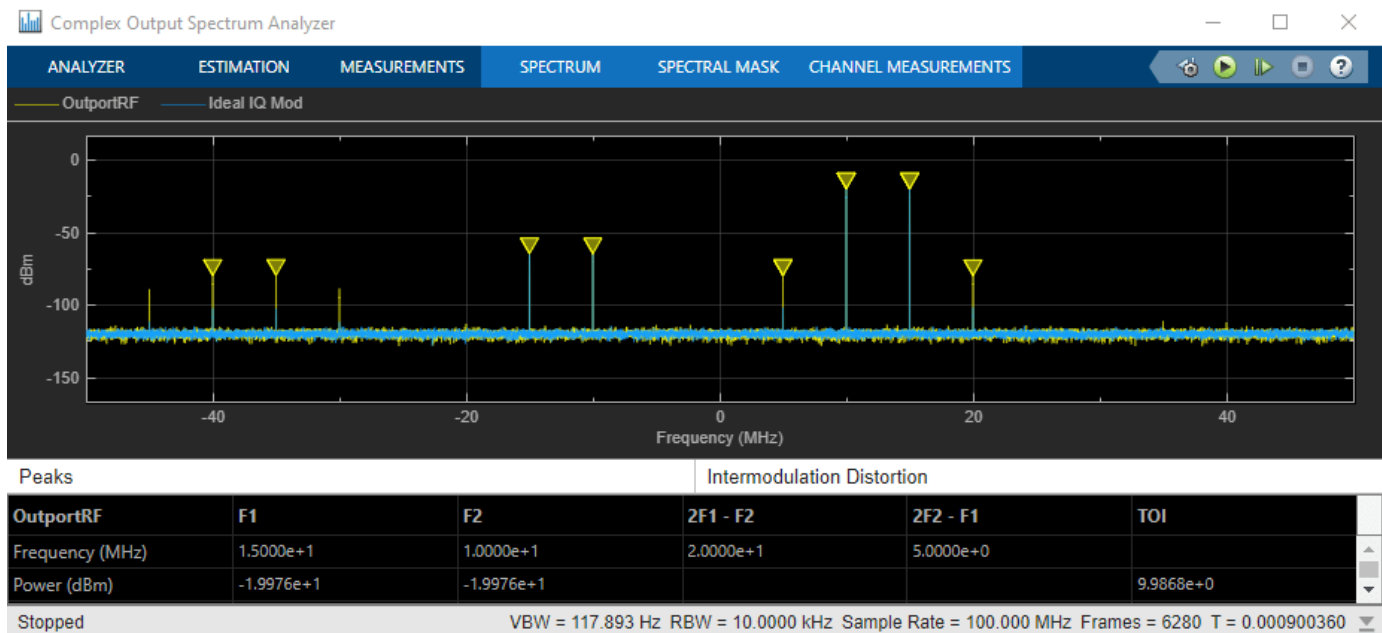
Run the model and observe the output of the Spectrum Analyzer blocks.

Analyze Complex Output Power Density Plot



In the complex output power density spectrum analyzer plot, the noise floor of the signal is at - 160 dBm/Hz for the two output signals OutputRF and Ideal IQ Mod.

Analyze Complex Output Spectrum Analyzer Plot



In the complex output spectrum analyzer plot, the modulated signal along with other signal tones produced by the impairments are shown. The output power levels at 10 MHz and 15 MHz are -20 dBm. The level of the other output tones are correct for circuit envelope, since it is a multi-tone simulator. Any differences between the modulated signals, 10 MHz and 15 MHz, result from different model implementations for the I/Q gain and phase mismatch. You can calculate the output power level using this formula:

$$\text{Output_Power_Level(dBm)} = \text{Input_Power_Level(dBm)} + \text{Gain(dBm)} = -30 + 10 = -20$$

See Also

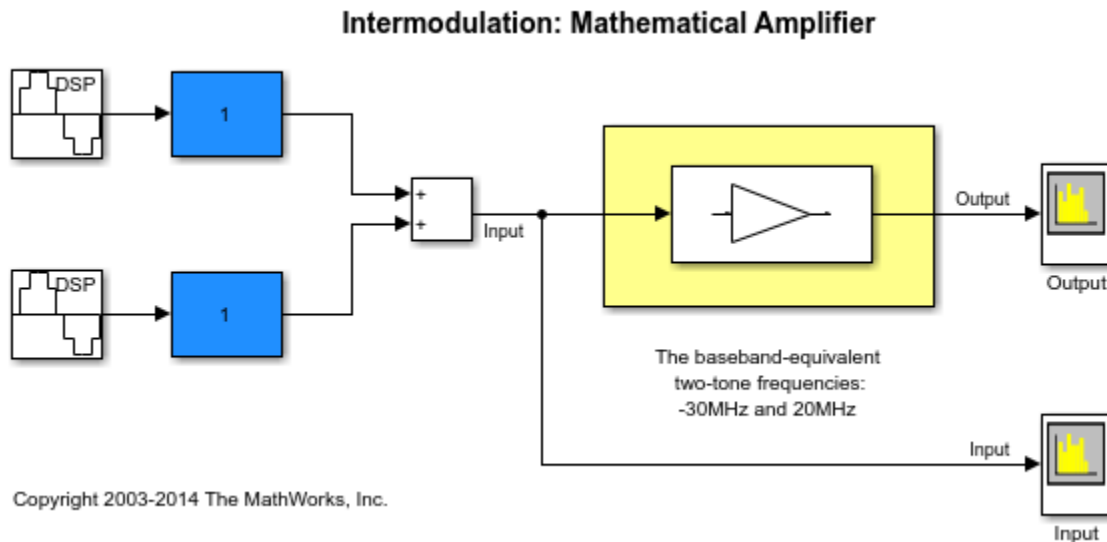
Mixer

Related Examples

- “Idealized Baseband Amplifier with Nonlinearity and Noise” on page 7-72
- “Nonlinearities and Noise in Idealized Baseband Mixer Block”

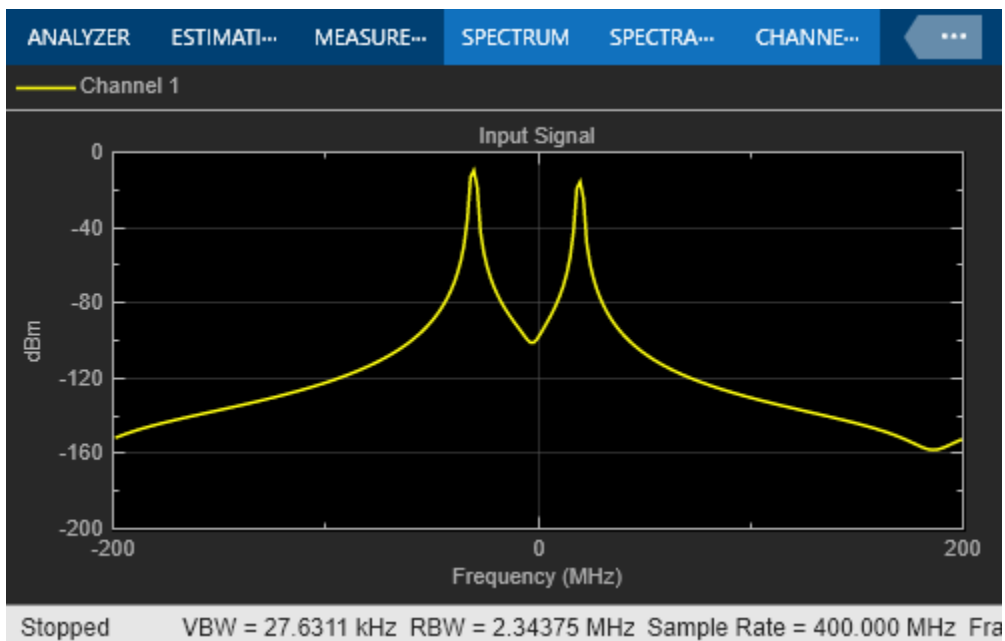
Intermodulation Analysis of Mathematical Amplifier

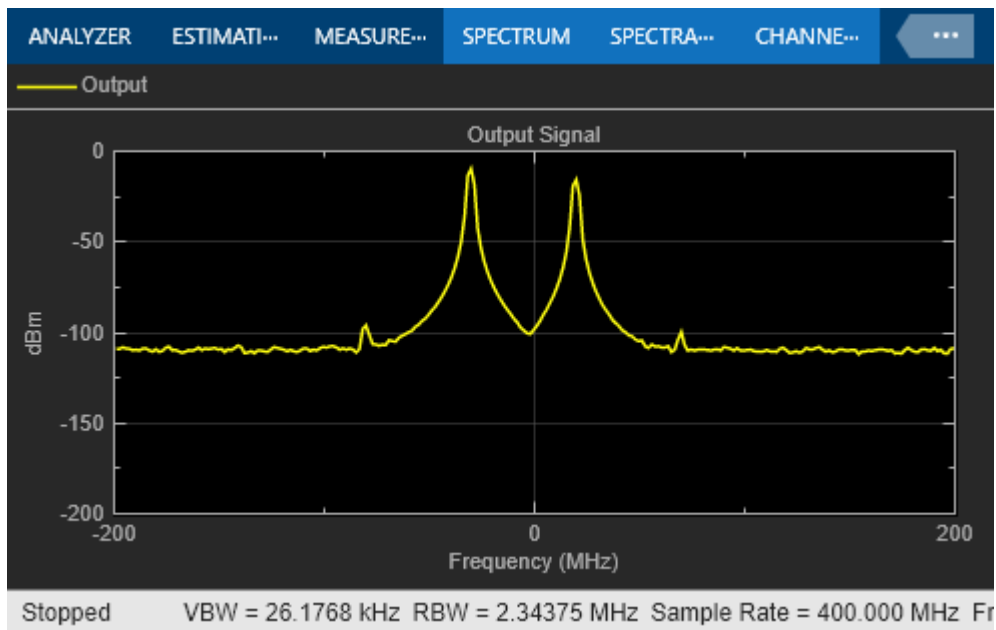
This example uses a baseband-equivalent multitone signal as input to the Amplifier block. A Simulink® Slider Gain block enables you to vary the gain from 1 to 10.



The Input scope block displays the spectrum of the two-tone signal with gain set to the default 1.

The Output scope block shows the output spectrum of the two-tone input signal when the input signal passes through the Amplifier block, with the **Method** parameter set to *Hyperbolic tangent*. The example uses the default Amplifier block **IIP3 (dBm)** value of 30. It uses no AM/PM conversion. The example specifies thermal noise as Noise figure, for which it uses the default 3.01 dB.





To view the intermodulation performance at the output, double click the Slider Gain blocks and change the gain while the model is running.

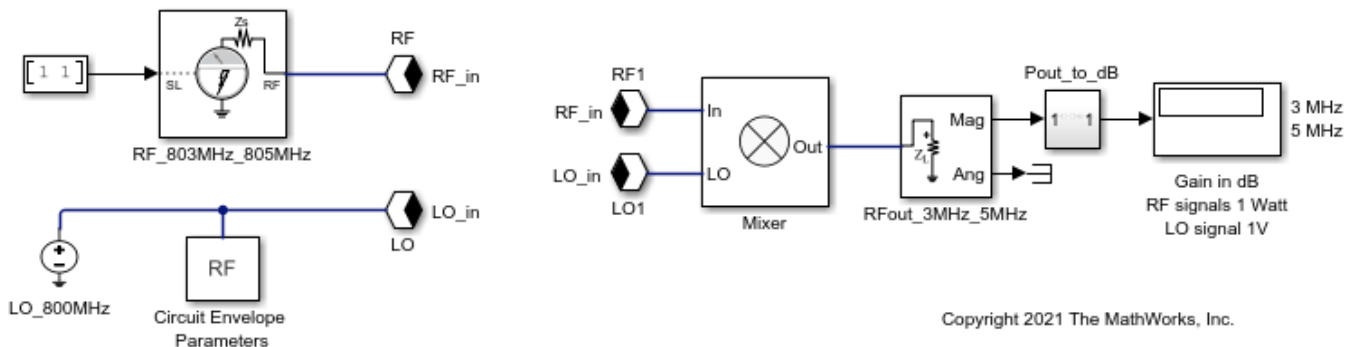
See Also

“Idealized Baseband Amplifier with Nonlinearity and Noise” on page 7-72

Create Virtual Connections Using Connection Label Block

This example shows how to use a Connection Label block to create a virtual connection between two conserving ports.

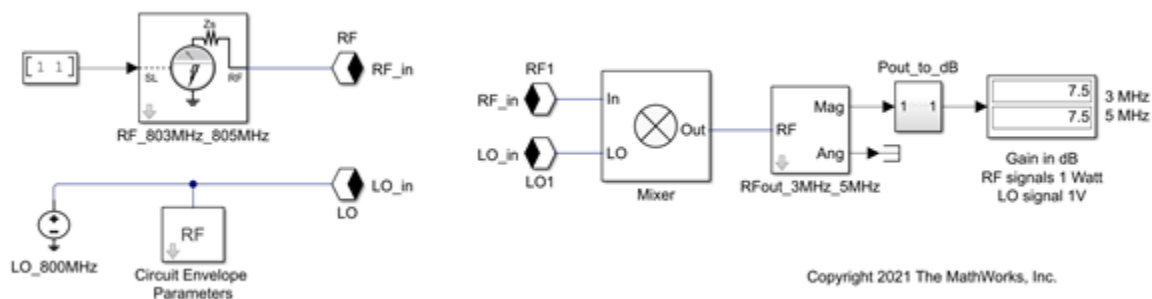
System Architecture



The Connection Label blocks in this example connect RF and local oscillator (LO) signals to the 7.5 dB Circuit Envelope Mixer block. To enumerate constant inputs and carrier frequencies in input frequencies, supply one Watt 803 MHz and 805 MHz RF signals through the Inport block by setting the **Carrier frequencies** parameter to [803 805] MHz. Use a Continuous Source block to generate a local oscillator (LO) signal. Translate the Outport block output from sqrt(power) to (power)dB.

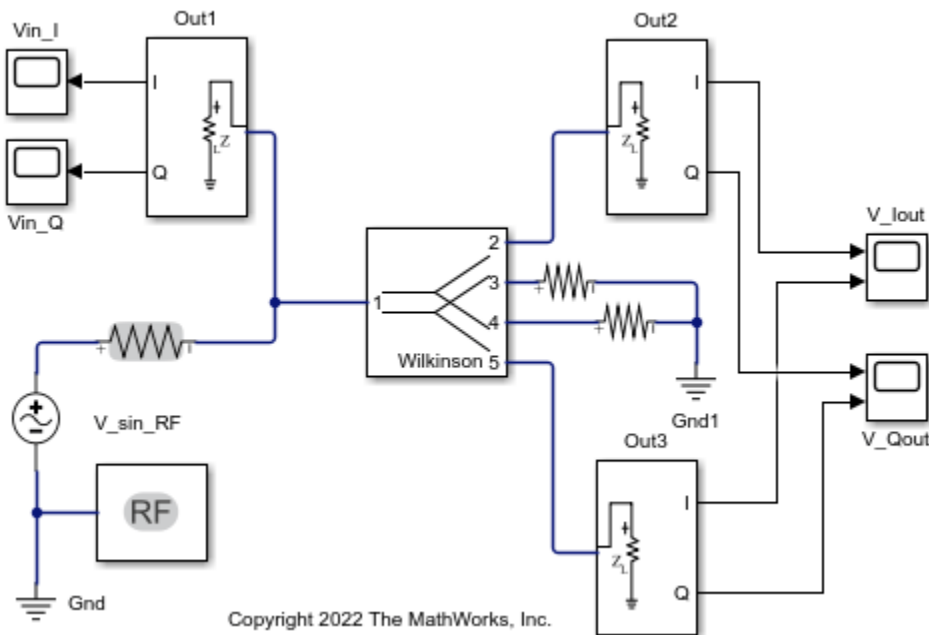
System Analysis

Available gain of RF signals at 3 MHz and 5 MHz is 7.5 dB. To increase the speed of this simulation, set the Configuration block **Fundamental tones** and **Harmonic order** parameters to [800 1] MHz and [1 5], respectively.

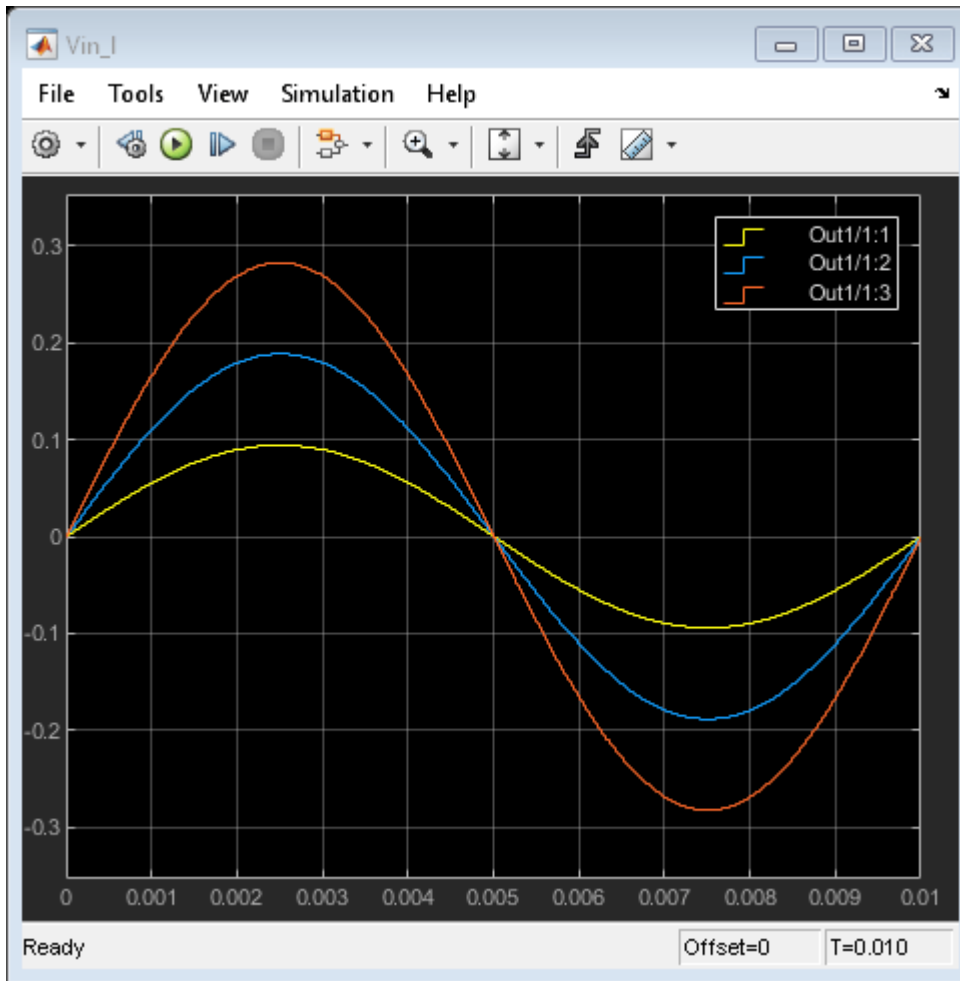


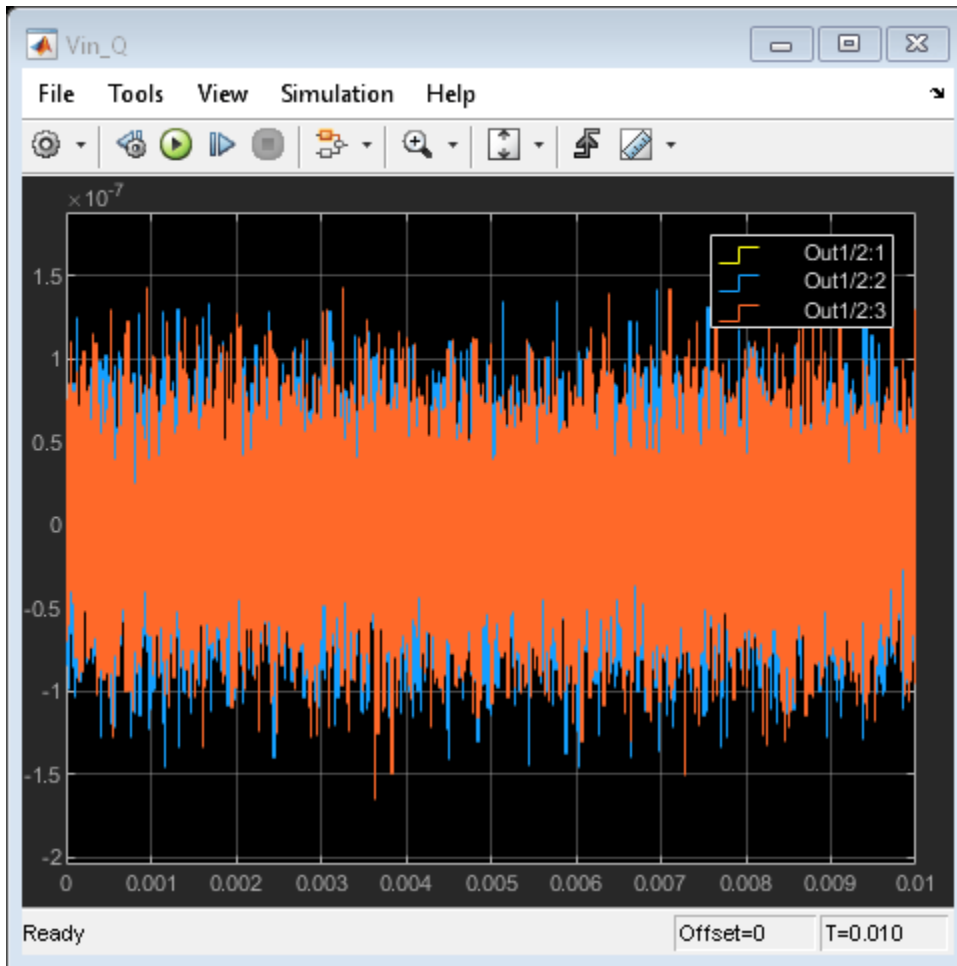
Model Wilkinson Power Divider

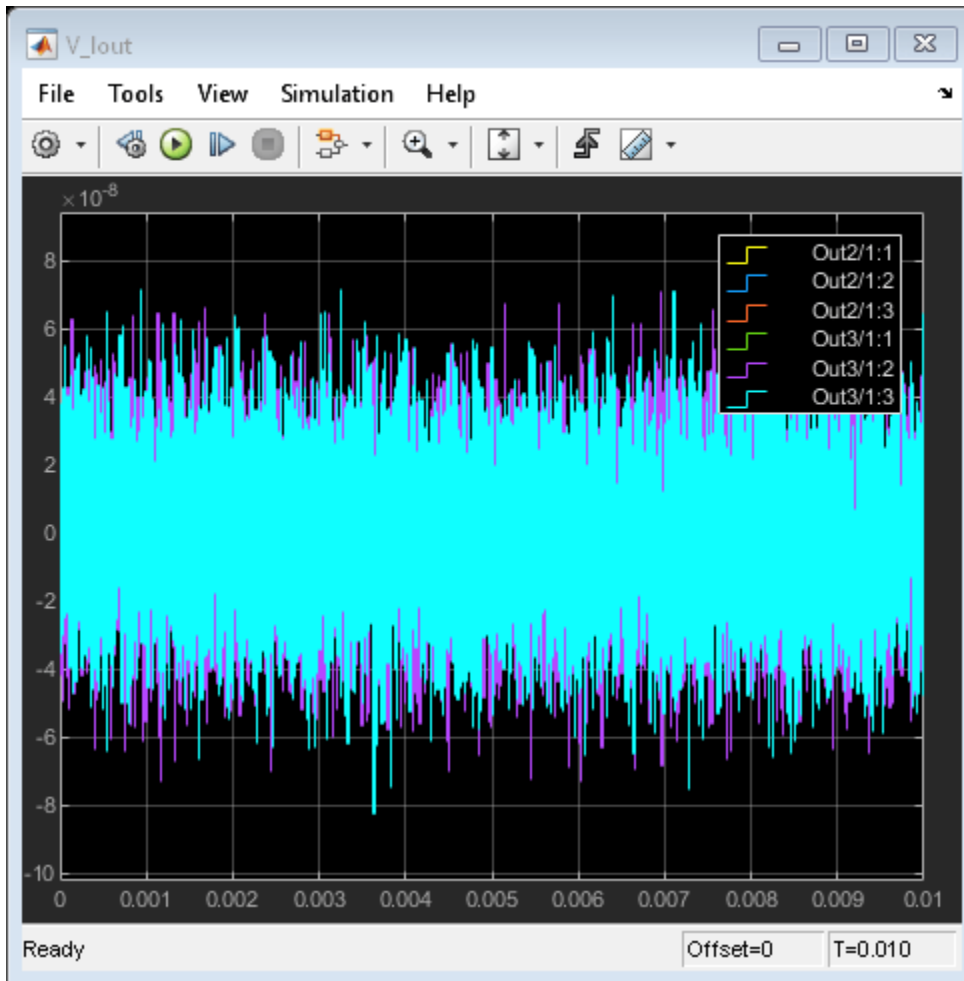
Use the Divider block to design a 1:4 Wilkinson power divider. You can use the **Number of divider outputs** parameter in the block to design a Wilkinson power divider with up to 65 ports. You can also design a Wilkinson power divider that works as a combiner by reversing the inputs and outputs.

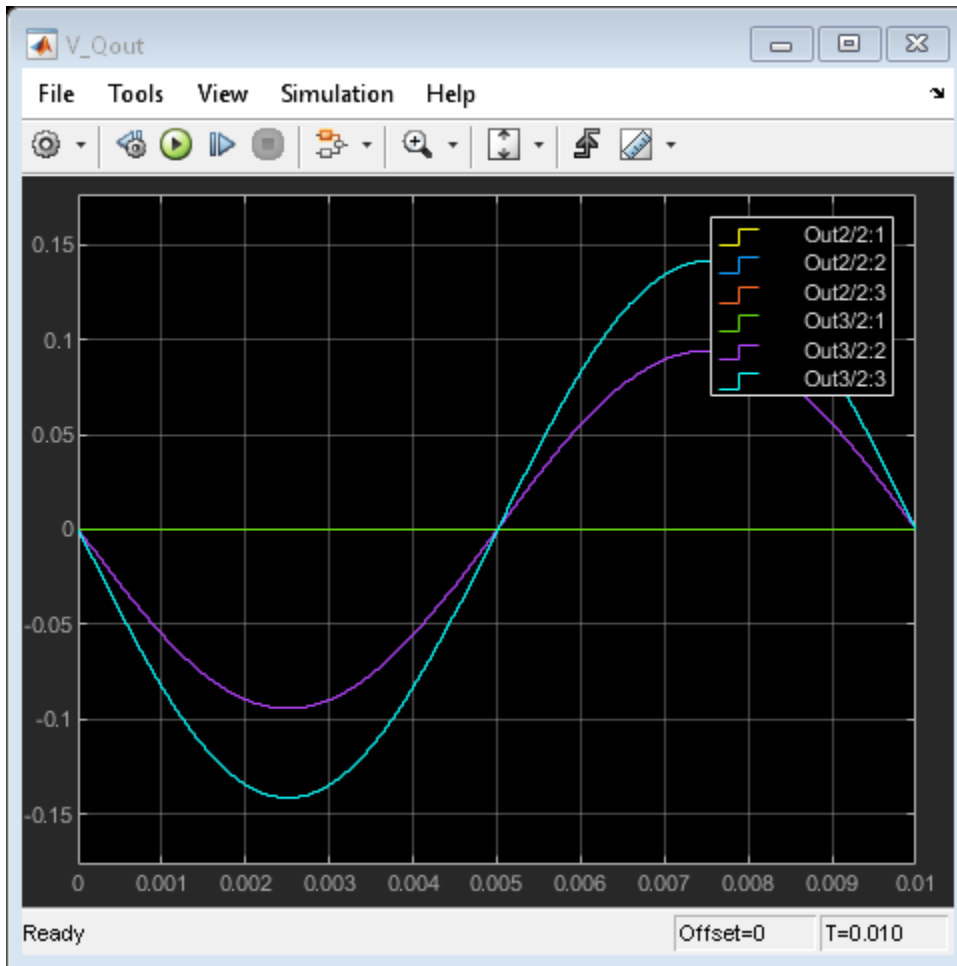


- 1 Open the model and observe that the three input carrier frequencies are set to [0 3 5] MHz in the Sinusoid block.
- 2 Run the model.
- 3 In the setup in this model, the output waveform is out of phase with the input. The output magnitude is $V_{out} = V_{in}/\sqrt{N}$, where N is the number of outputs. The other waveforms V_{in_Q} are noise added and transmitted by the system. Add or delete load resistors or Outputport blocks, rerun the simulation, and check the magnitude of the outputs.









See Also

Coupler | Circulator

Related Examples

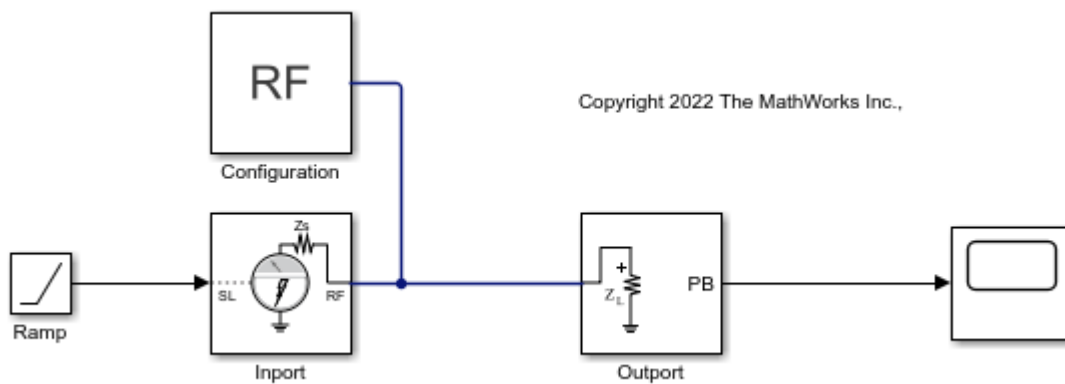
- "100 Watt TR Module for S-Band Applications" on page 8-191

Modulate Input Signal Onto Square Carrier Wave

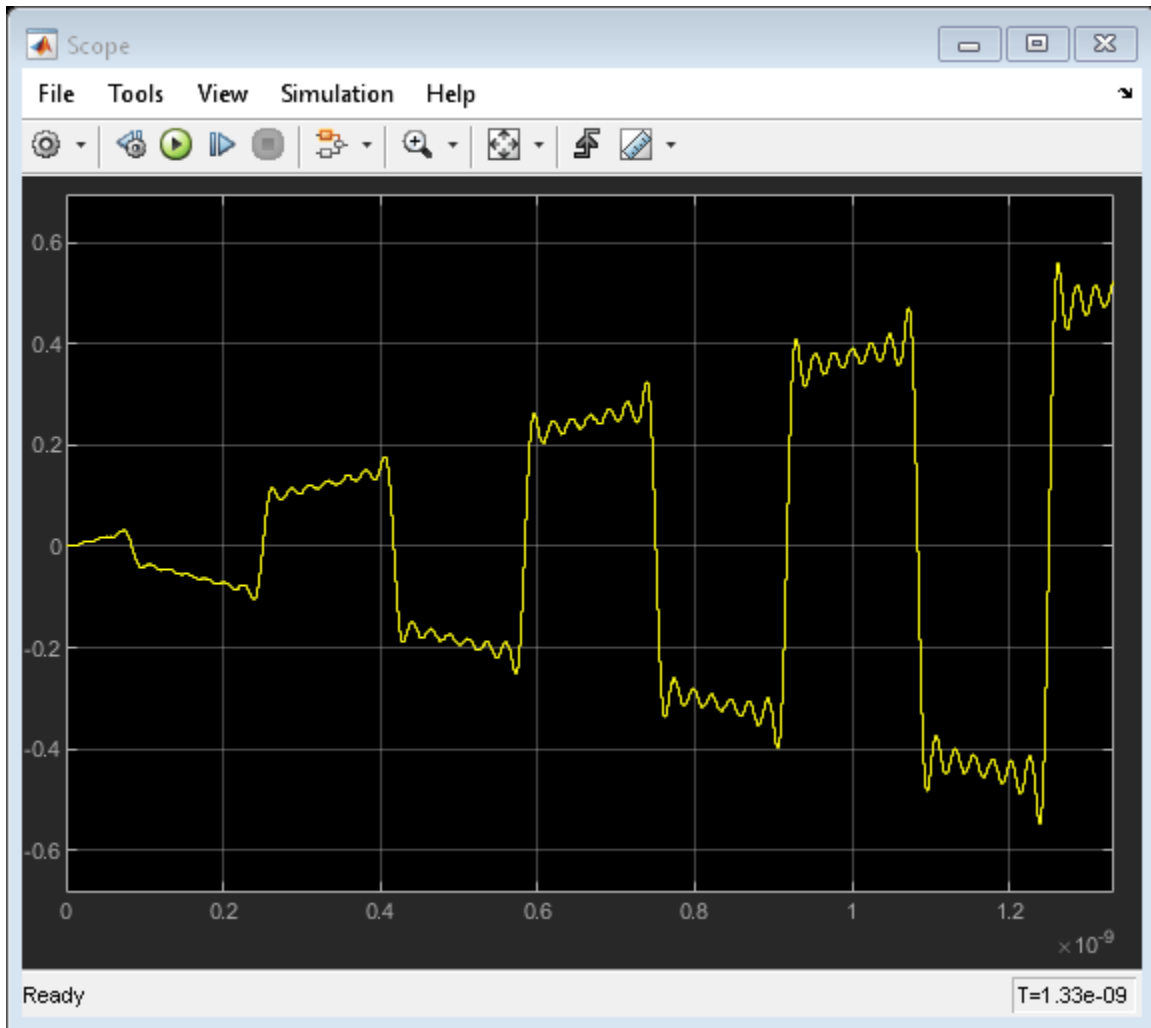
Use the Inport block to modulate an input signal onto a square carrier wave in the circuit envelope simulation environment.

Open the model and double-click the Inport block to set the parameters as follows.

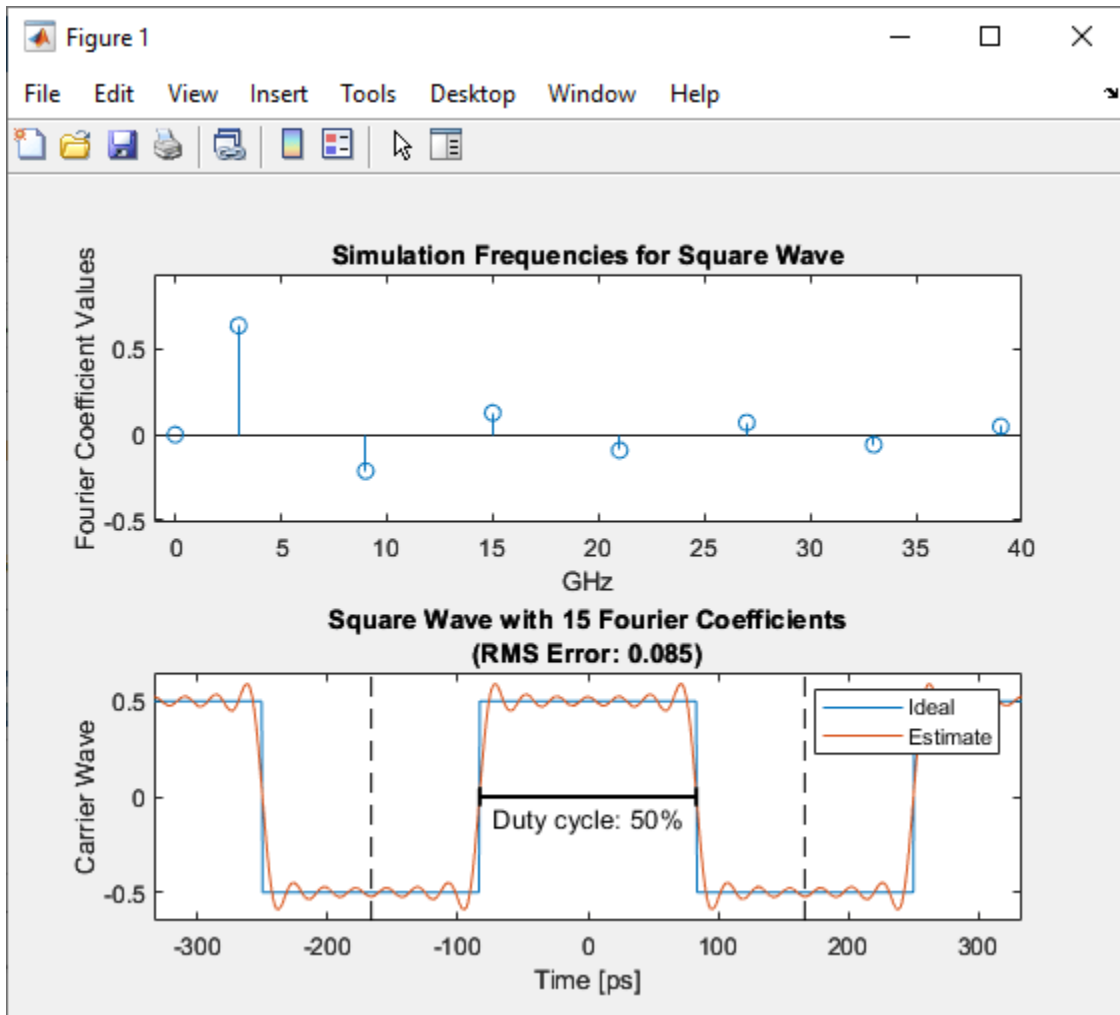
- **Source type** — power
- **Source impedance (Ohm)** — 50
- **Carrier frequencies** — 3e9 Hz
- **Use square wave** — on
- **Number of Fourier Coefficients** — 15
- **DC Bias** — - 0.5
- **Duty Cycle (%)** — 50
- **Ground and hide negative terminals** — on



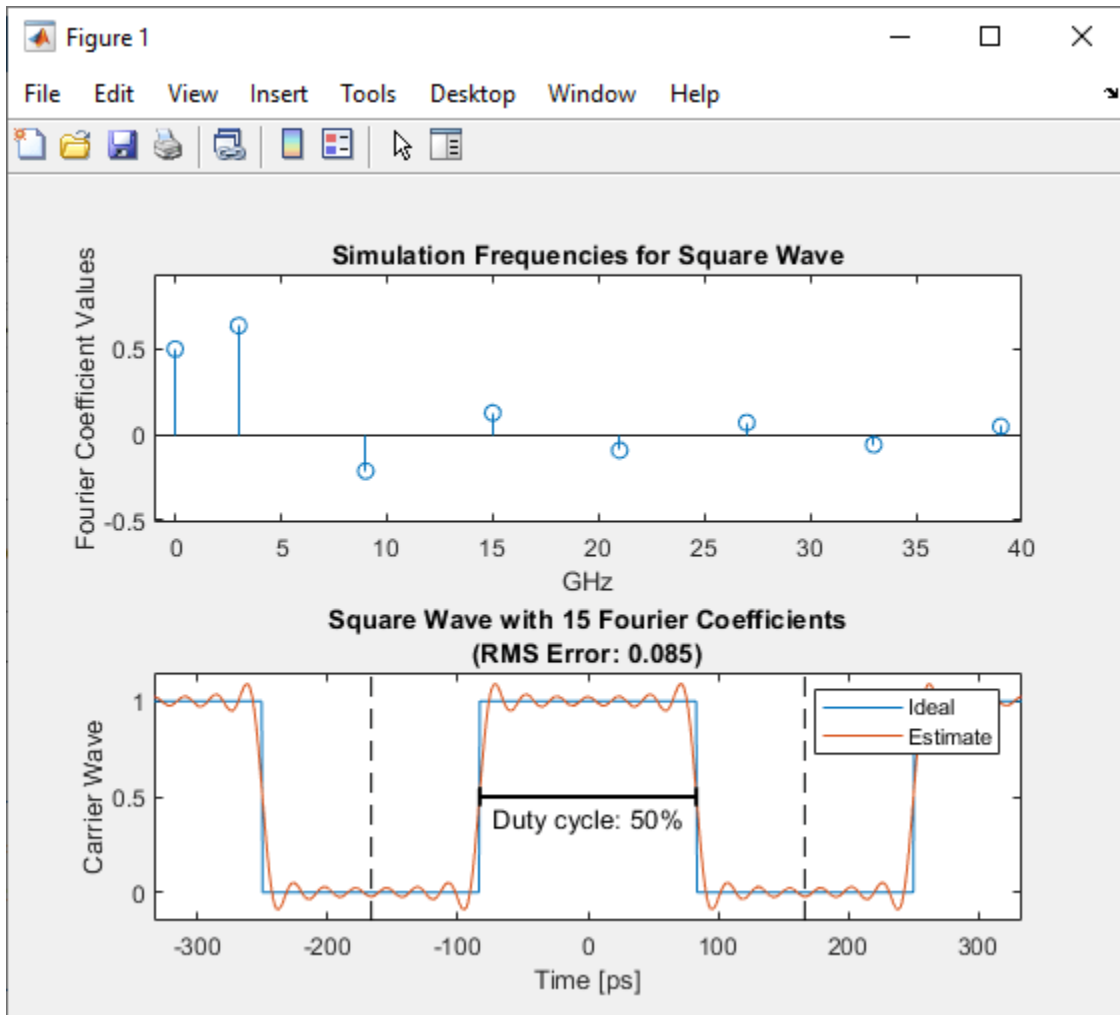
Run the model and view the result in the scope.



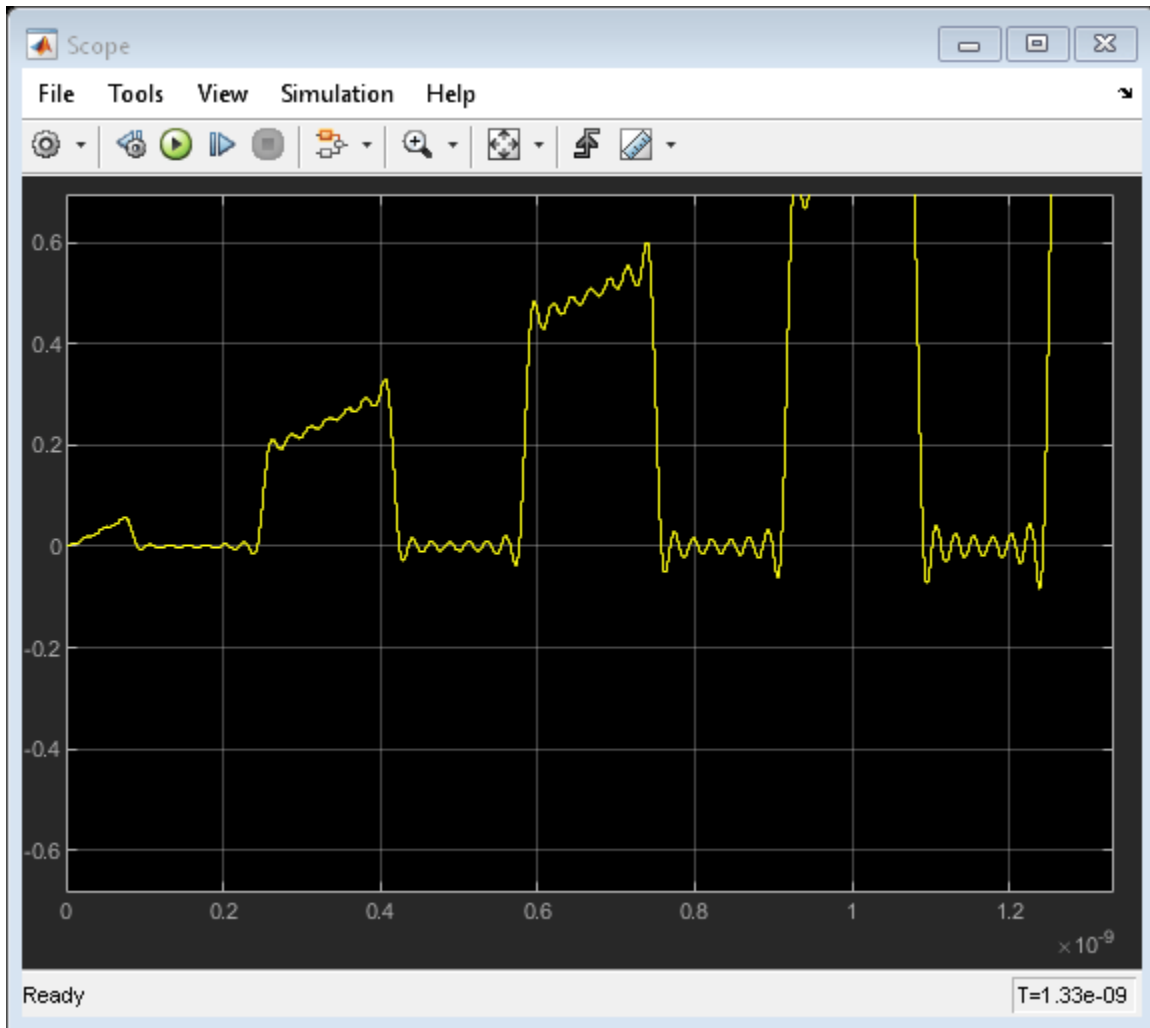
Double-click the Inport block and select the **View** button to plot the simulation frequencies for the square carrier wave and the approximate square wave with RSM error of the approximation.



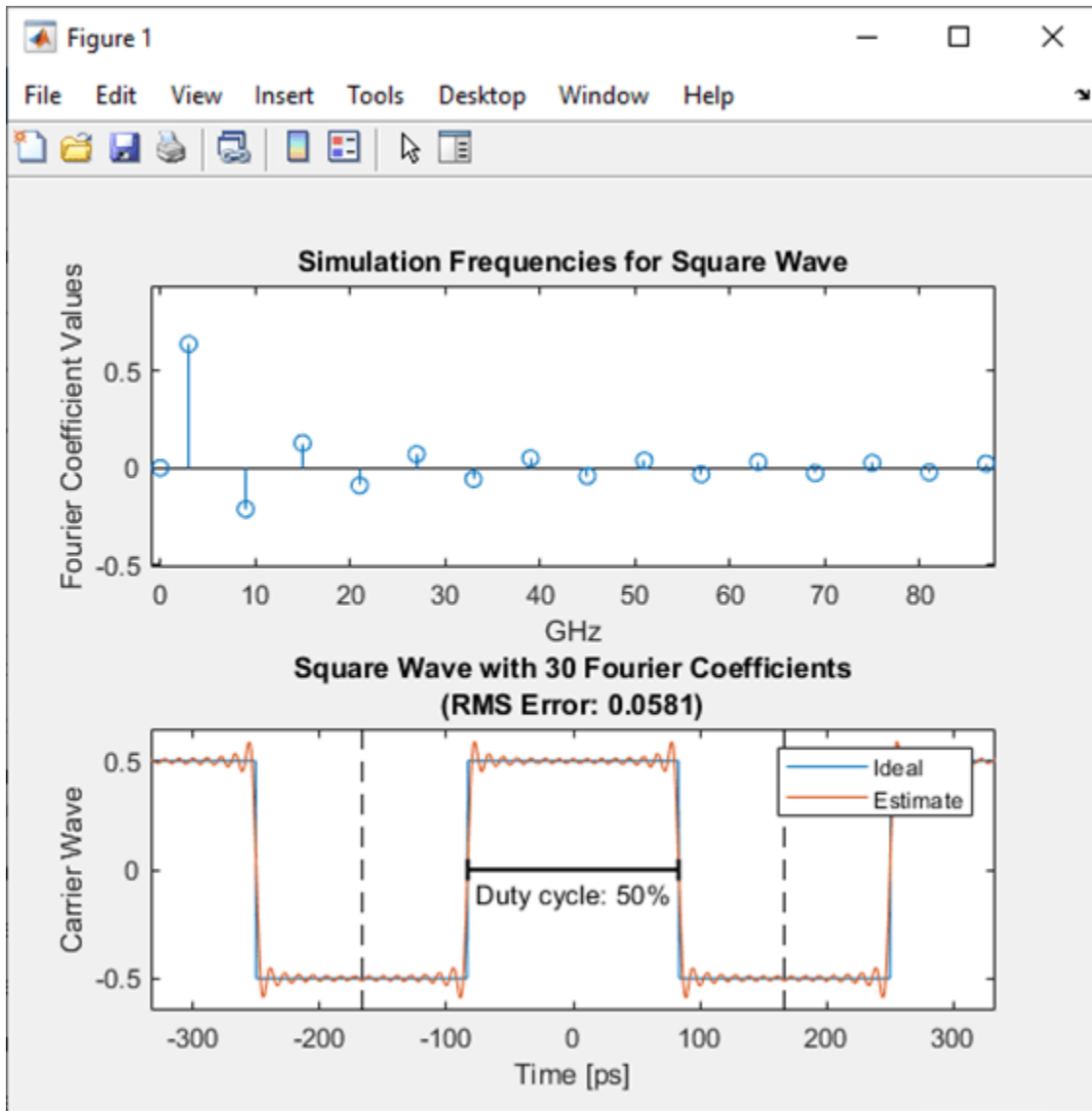
Change the **DC Bias** value to 0, press **Apply**, and then press **View** again. The DC coefficient has increased and the Carrier wave function is between 0 and 1.



Run the model. The scope now shows that the modulated square wave is alternating between 0 and the ramp envelope.

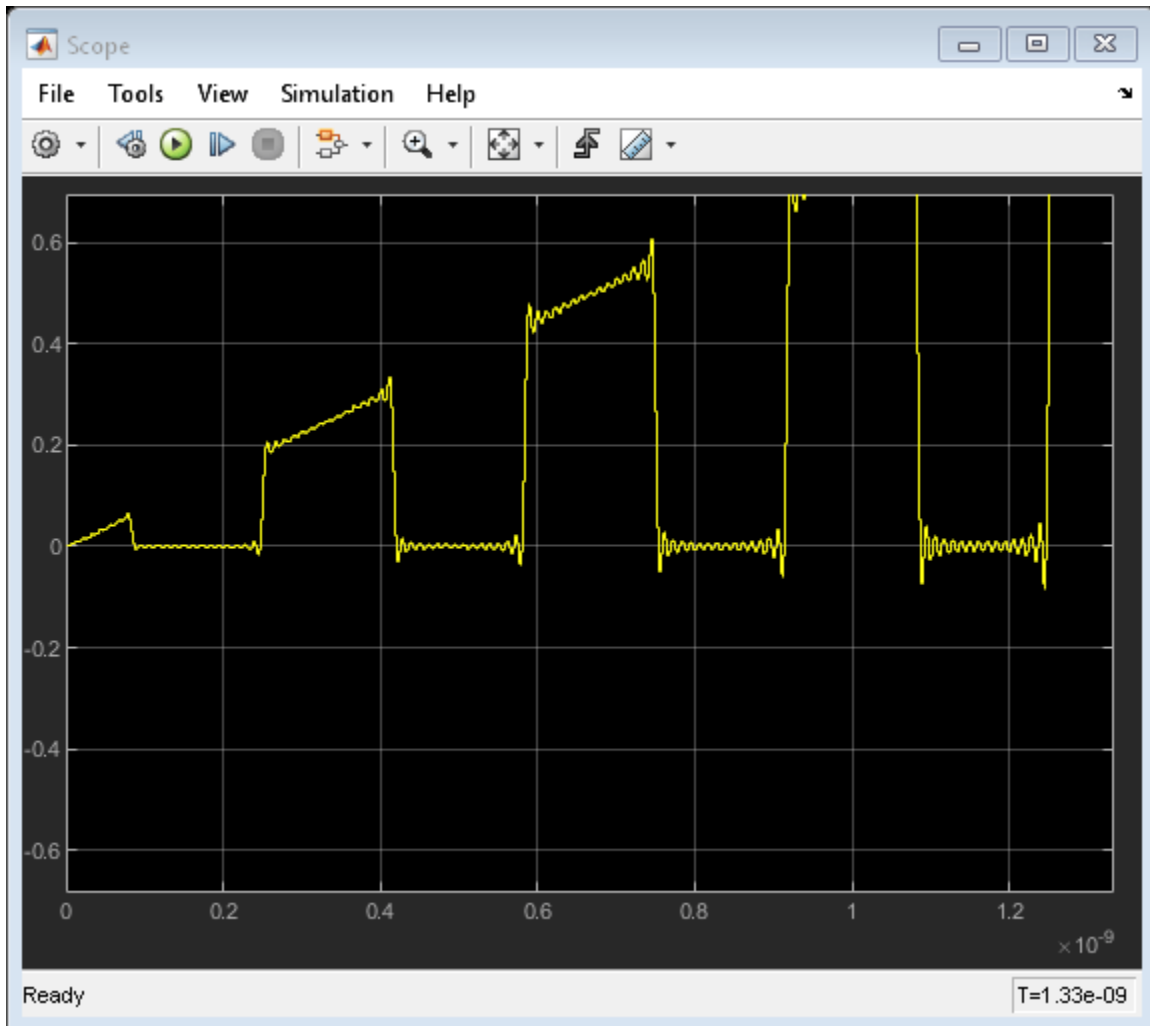


In the Inport block, change the **Number of Fourier Coefficients** from 15 to 30 . Press **Apply** and select the **View** button to see the effect on the expected square wave.



The calculated RMS Error is down from 0.085 to 0.0581.

Double-click the Output block to open its mask and change the value of the **Carrier frequencies parameter** from $3e9*[0:15-1]$ to $3e9*[0:30-1]$. Rerun the model to view the change to the output signal in the scope. The ripples are now denser and smaller (except for the expected Gibbs effect at the discontinuities) with more Fourier coefficients to approximate the square wave.



You can also try:

- Changing the **Number of Fourier coefficients**, **DC Bias**, and **Duty Cycle** parameters to different values to view the effect on the visualization plot and output.
- Changing the Ramp block to another function to represent a different envelope.
- Changing the RF system (for example, adding a nonlinear amplifier) and output complex baseband signals. The input square wave is created without RMS normalization, so the peak values correspond to the Fourier coefficients, while the output complex baseband are RMS normalized, and the values at the output are expected to be $1/\sqrt{2}$ smaller than the Fourier coefficients seen in the visualization plot.

See Also

Output

Related Examples

- “Attenuate Signal Power” on page 7-6

Time-Domain Filtering of RF Complex Baseband Signals in Simulink

This example shows you how to use the Idealized Baseband Filter block to filter of RF complex baseband signals in Simulink®. The Idealized Baseband Filter block performs time-domain or frequency-domain simulation for Butterworth, Chebyshev, or inverse Chebyshev filters. The example compares time-domain simulation of RF complex baseband signals between Idealized Baseband and Circuit Envelope Filter blocks in Simulink® and the circuit envelope domain, respectively. The Idealized Baseband Filter block allows you to design single-carrier perfectly matched RF systems, whereas the Circuit Envelope (CE) Filter block allows you to design multi-carrier RF systems with impedance mismatches.

System Architecture

This example uses three filter blocks:

- Idealized Baseband Filter block simulated using `Interpreted` execution. This option performs block signal processing using the MATLAB® interpreter, and shortens startup time, but the speed is slower than Code generation.
- Idealized Baseband Filter block simulated using `Code` generation. This option performs block signal processing using generated C code. The first time you run a simulation, Simulink generates C code for the block. The C code is reused for subsequent simulations, as long as the model code path does not change. This option requires additional startup time, but the speed of the subsequent simulations is faster than interpreted execution.
- Circuit Envelope Filter block implemented using `Transfer` function. This option allows you to model an analog filter using two-port S-parameters.

Define Simulation Parameters

Define sample time, samples per frame, and carrier frequency as the simulation parameters.

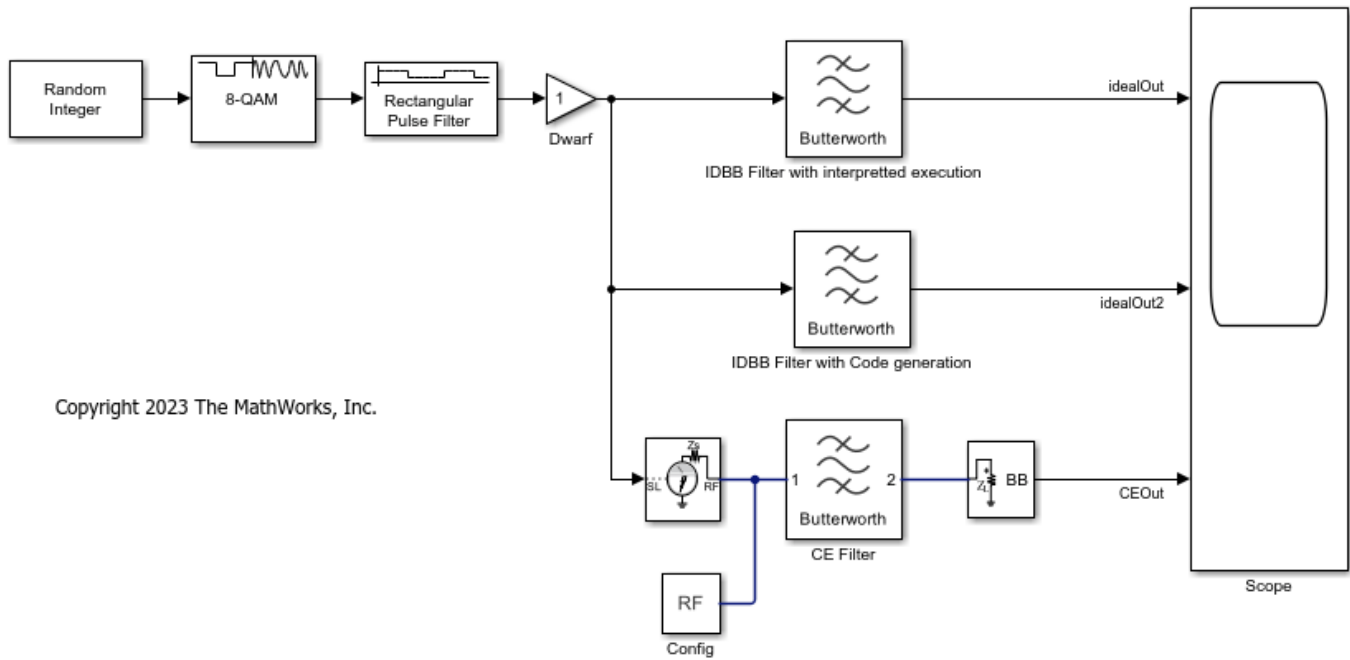
```
sampleTime = 8e-6;  
samplesPerFrame = 128;  
carrierFrequency = 1e9;
```

Set **Stop Time** to `sampleTime*1e3`.

Open Model

The **Filter type** parameter in the three filter blocks is set to `Butterworth`. Set the **Carrier frequency (Hz)** parameter to `1e9` Hz, and **Solver type** to `NDF2`. For the Circuit Envelope Filter block, specify the carrier frequency, solver type, and frame size in the Configuration block, and for the Idealized Baseband blocks, specify the carrier frequency and solver type in the block masks. The frame size for the Idealized Baseband Filter blocks is automatically detected by the software.

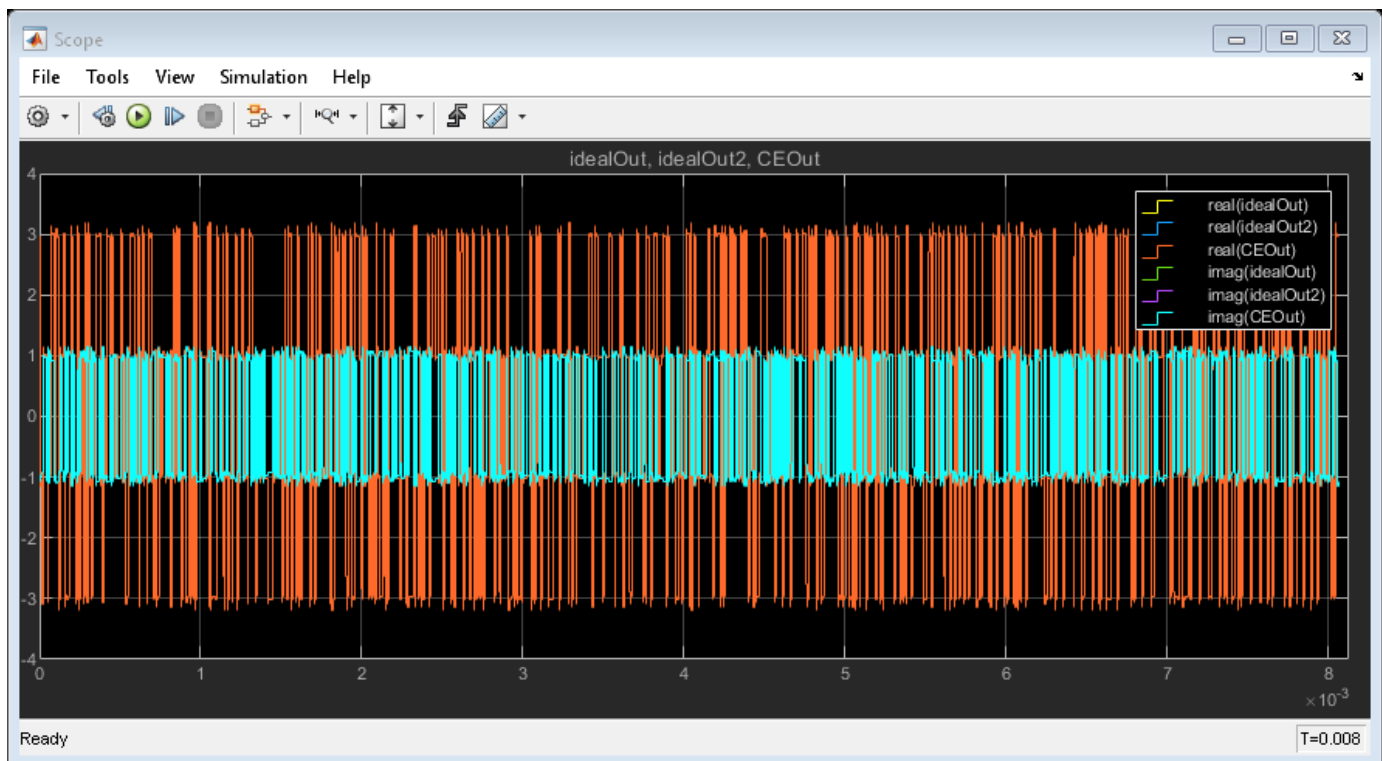
```
open_system('idealfilter.slx')
```



Simulate Model

Simulate the model using the `sim('idbbfilter.slx')` command.

```
sim('idealfilter.slx');
```



Compare the real and imaginary output waveforms and observe that the output waveforms are identical. Idealized Baseband and Circuit Envelope Filter blocks are simulated using NDF2, a fixed-step time-domain solver. This solver balances narrowband and wideband accuracy and is suitable for situations where the frequency content of the signals in the system is unknown relative to the Nyquist rate.

See Also

Filter | Configuration | Filter

Related Examples

- “Model RF Complex Baseband S-Parameters in Simulink” on page 7-125

Model RF Complex Baseband S-Parameters in Simulink

This example shows you how to use the Idealized Baseband S-parameters block to model the RF complex baseband S-parameters in Simulink®. The example compares different solvers that you can use to model the RF complex baseband S-parameters in Simulink and in the circuit envelope domain. The Idealized Baseband S-parameters block allows you to design single-carrier perfectly matched RF systems, whereas the Circuit Envelope (CE) S-parameter block allows you to design multi-carrier RF systems with impedance mismatches.

System Architecture

This example uses of two systems. The first system compares fixed-step time-domain solvers, and the second system compares continuous time-domain solvers and frequency-domain solvers.

Fixed-step solvers include:

- NDF2 — Balance narrowband and wideband accuracy. This solver is suitable for situations where the frequency content of the signals in the system is unknown relative to the Nyquist rate.
- Trapezoidal — Perform narrowband simulations. Frequency warping and the lack of damping effects make this method unsuitable for most wideband simulations.
- Backward Euler — Simulate the largest class of systems and signals. Damping effects make this solver suitable for wideband simulation, but overall accuracy for this solver is low.

Continuous solvers include:

- ode15s - Solve stiff differential equations and DAEs using a variable order method
- ode23s - Solve stiff differential equations using a low-order method
- ode23t (trap) - Solve moderately stiff ODEs and DAEs using the trapezoidal rule
- ode23tb (trap+BE) - Solve stiff differential equations using the trapezoidal rule and backward differentiation formula

Frequency-domain (digital filter) modeling uses a 1-D digital filter.

The systems also model and compare the S-parameters in the interpreted execution and code generation simulation modes.

Define Simulation Parameters

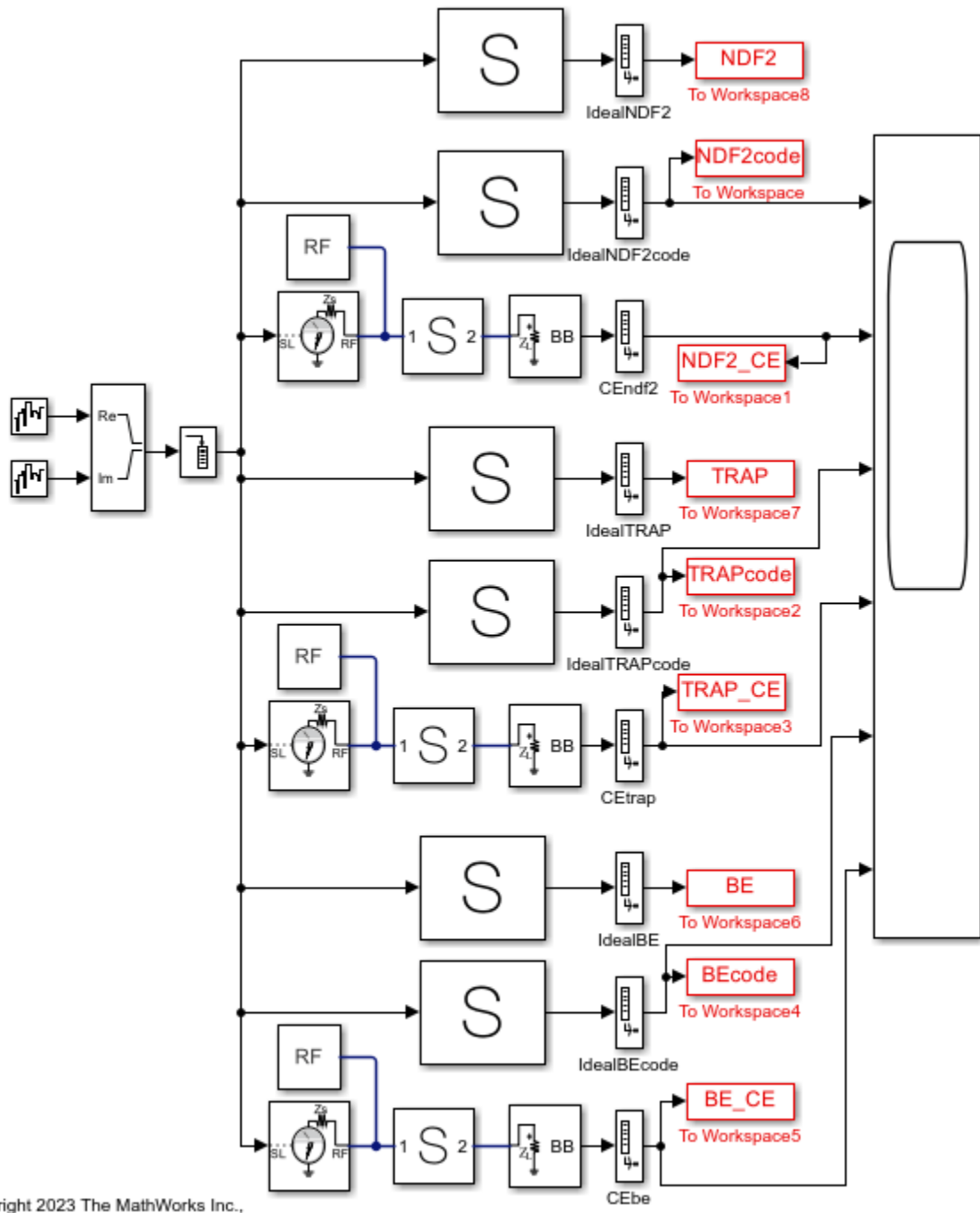
Define sample time, samples per frame, and carrier frequency as the simulation parameters.

```
sampleTime = 8e-6;
samplesPerFrame = 512;
carrierFrequency = 2.45e9;
```

Open and Run Fixed-Step Solver System

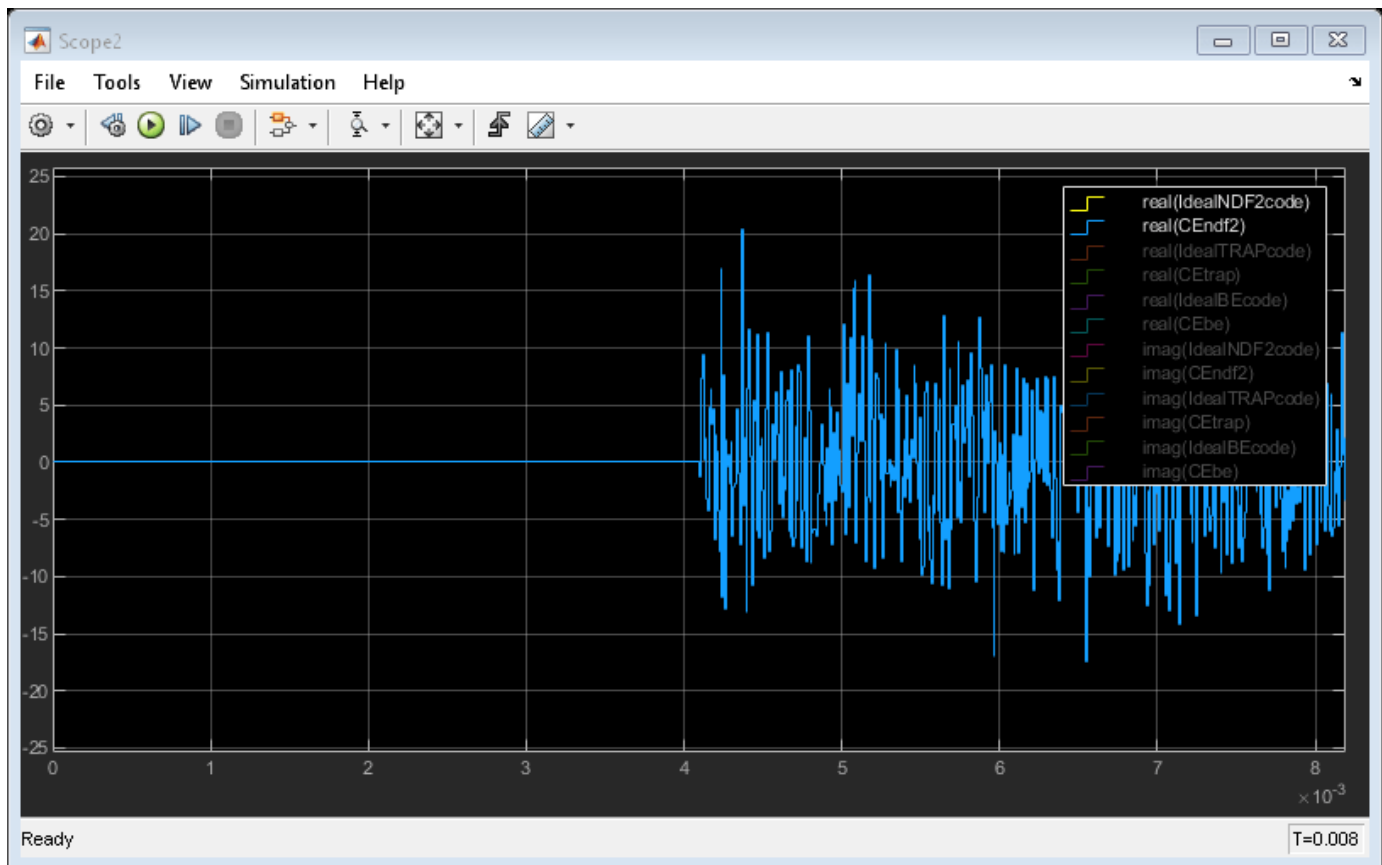
Open the fixed-step solver system.

```
open_system("idealsparams_fs.slx")
```



Run the fixed-step solver system.

```
sim("idealsparams_fs.slx");
```

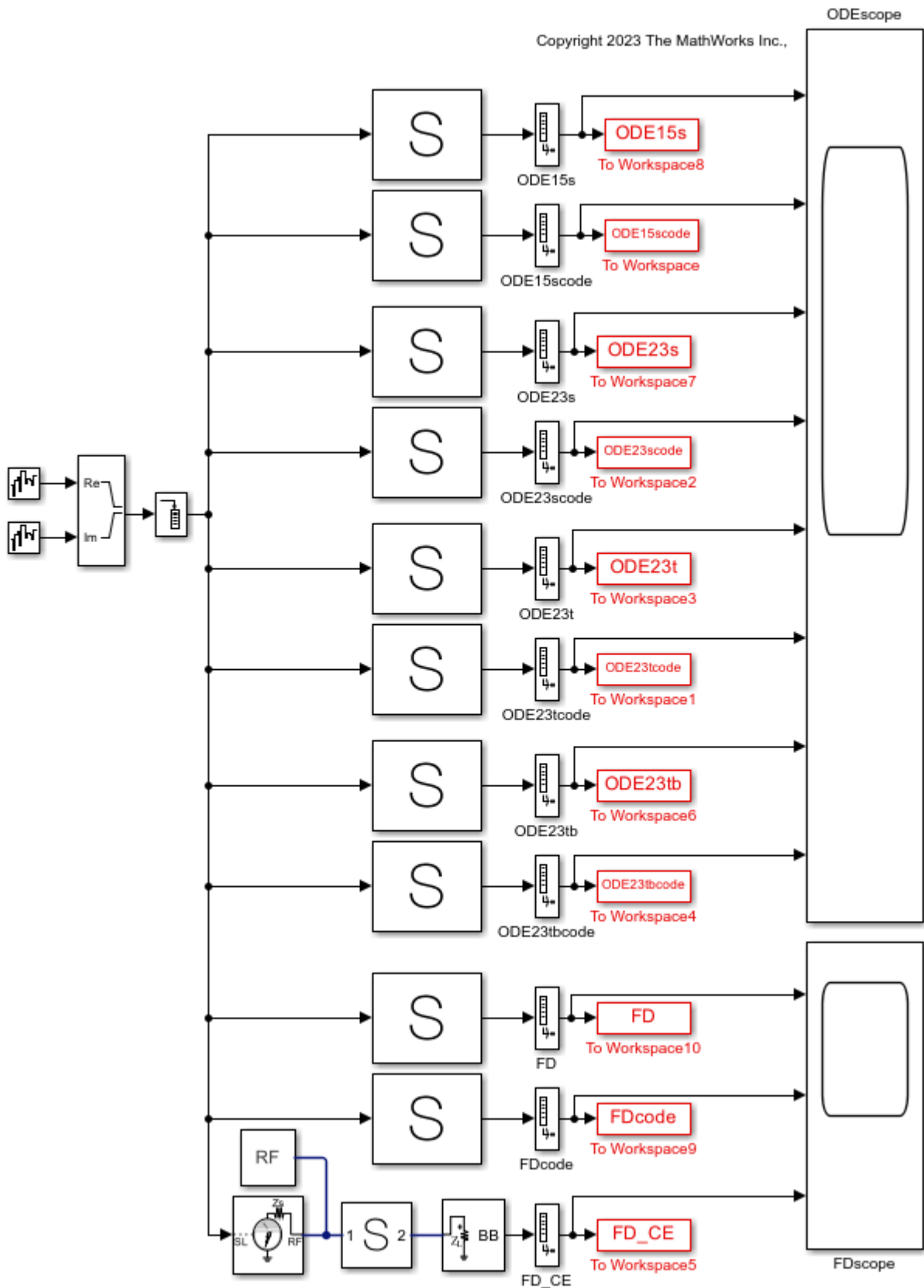


All solutions for a particular fixed-step time-domain solver are the same after the differences in the initialization time disappear. Compare Ideal and CE S-parameter characteristics using the mask visualization tabs.

Open and Run Continuous Time Domain Solvers and Frequency Domain Solvers System

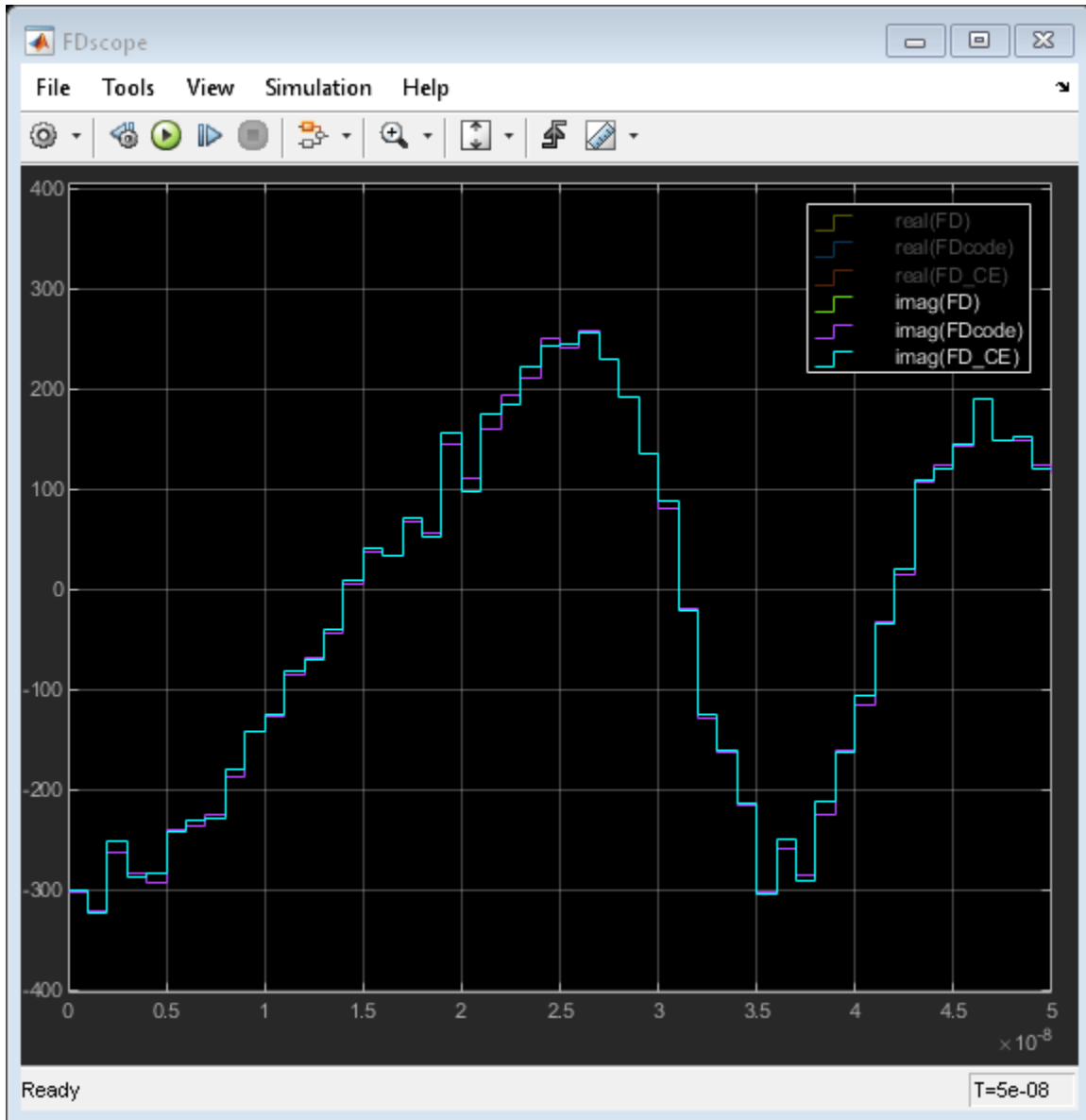
Open the continuous time-domain and frequency-domain modeling system.

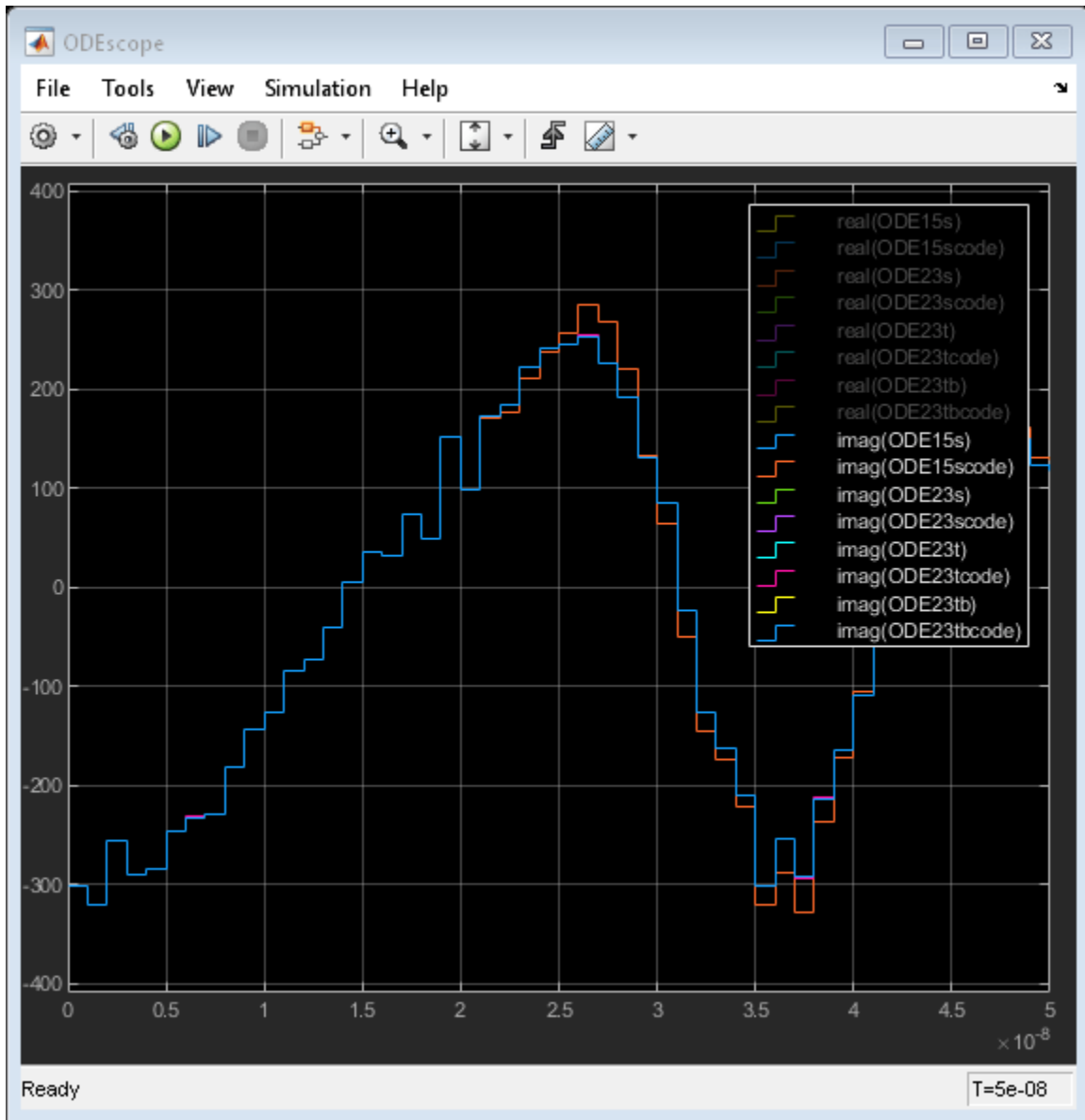
```
open_system("idealsparams_ode.slx")
```



Run the modeling system.

```
sim("idealsparams_ode.slx");
```





All solutions for a particular ODE solver are the same. But the solutions differ for the Ideal and CE frequency-domain solvers as they use different digital filter windowing schemes.

See Also

S-parameters | Configuration | S-Parameters

Related Examples

- “Time-Domain Filtering of RF Complex Baseband Signals in Simulink” on page 7-122

RF Blockset Examples

Getting Started with RF Modeling

Learn how to use the **RF Budget Analyzer** app to build a simple RF receiver and then create an RF Blockset™ Circuit Envelope multi-carrier model to perform simulation.

Build a Cascade (row vector) of RF Elements.

You can build and analyze an RF cascade by adding elements characterized by their data sheet specifications.

You can use the **RF Budget Analyzer** app and drag and drop new elements, or you can script the chain elements using MATLAB® commands. If you are not familiar with the syntax, you can start with app and generate a MATLAB script.

Add elements to your chain in the following order:

- Filter specified by an S-parameters Touchstone file
- Low Noise Amplifier (LNA)
- Direct conversion demodulator
- Baseband amplifier

```
elements(1) = nport('sawfilterpassive.s2p');
elements(2) = amplifier(
    'Name','LNA',
    'Gain',18,
    'NF',3,
    'OIP3',10);
elements(3) = modulator(
    'Name','Demod',
    'Gain',10,
    'NF',6.4,
    'OIP3',36,
    'LO',2.45e9,
    'ConverterType','Down');
elements(4) = amplifier(
    'Gain',20,
    'NF',11.3,
    'OIP3',42);
```

Inspect RF Budget Using RF Budget Analyzer App

Construct an `rfbudget` object. The MATLAB command window dynamically displays the budget analysis results.

```
b = rfbudget(
    'Elements',elements,
    'InputFrequency',2.45e9,
    'AvailableInputPower',-70,
    'SignalBandwidth',8e6)
```

b =

rfbudget with properties:

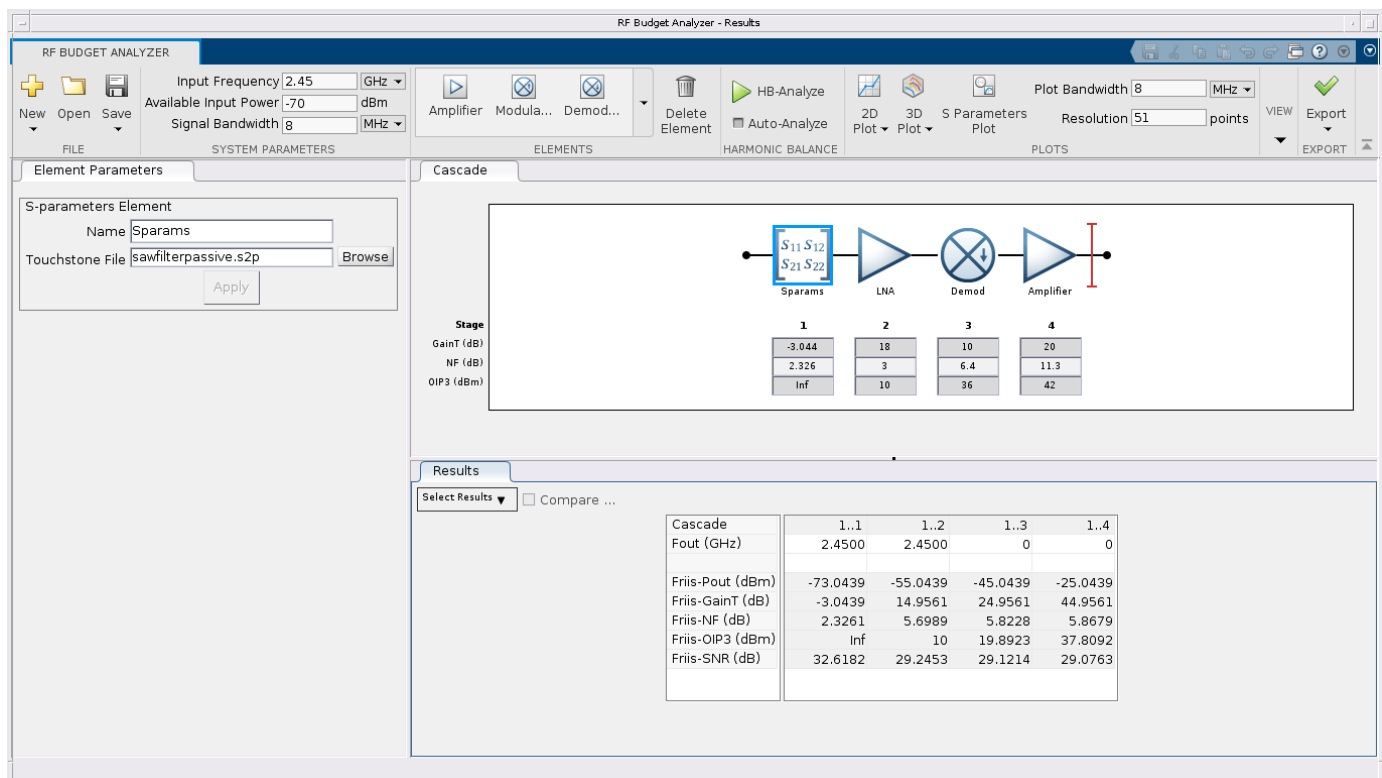

```

Elements: [1x4 rf.internal.rfbudget.Element]
InputFrequency: 2.45 GHz
AvailableInputPower: -70 dBm
SignalBandwidth: 8 MHz
Solver: Friis
AutoUpdate: true

Analysis Results
OutputFrequency: (GHz) [ 2.45 2.45 0 0]
OutputPower: (dBm) [-73.04 -55.04 -45.04 -25.04]
TransducerGain: (dB) [-3.044 14.96 24.96 44.96]
NF: (dB) [ 2.326 5.699 5.823 5.868]
IIP2: (dBm) []
OIP2: (dBm) []
IIP3: (dBm) [ Inf -5.674 -5.782 -7.865]
OIP3: (dBm) [ Inf 10 19.89 37.81]
SNR: (dB) [ 32.62 29.25 29.12 29.08]

```

Or you can visualize the `rfbudget` object in the app using the MATLAB command `show(b)`.



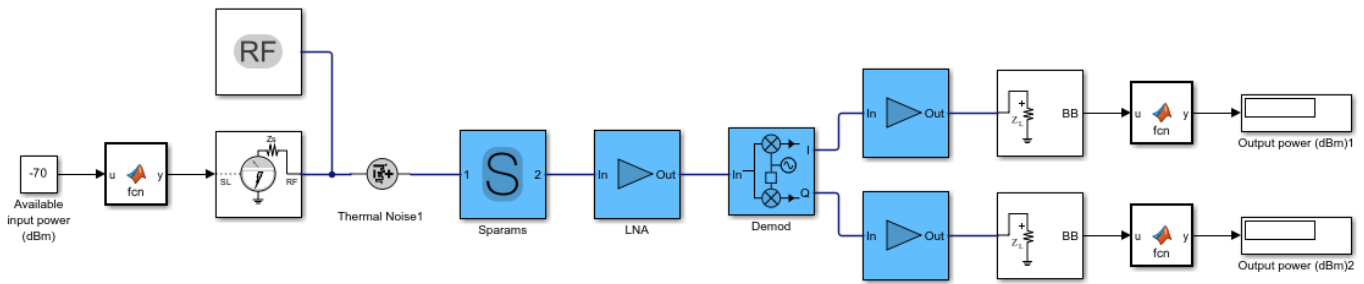
Generate RF Blockset Model

Use the **Export** button in the **RF Budget Analyzer** app to create an RF Blockset model or:

```

exportRFBlockset(b)
save_system(gcs, 'model_1')

```



You can use this model for multi-carrier circuit envelope simulation. The Input Port / Output Port ports and Configuration block are set up correctly and you can copy the model for use in any other Simulink® testbench.

- The input port specifies a complex powerwave signal centered at 2.45 GHz.
- The output ports terminate the cascade and extract the envelope centered at DC (0 Hz). The I and Q signals are real baseband signals.
- The configuration block runs the simulation for a total of eight simulation frequencies in order to capture the non linearity introduced by the demodulator and amplifiers.
- The simulation stop time in this case is set equal to 0. This means that the simulation does only a static analysis of the model (harmonic balance).

Observe and understand the model blocks:

- The S-parameter block describing the filter uses rational fitting in order to simulate frequency data in the time domain. Notice that at 2.45 GHz it introduces a phase rotation of approximately -58 degrees.
- Both amplifiers specify IP3, but you can also specify IP2.
- The demodulator includes ideal channel selection filters. Additional impairments can be added such as LO leakage and I/Q imbalance.

Simulate the model to compare the output power values with the **RF Budget Analyzer** app values. Notice that due to the phase rotation introduced by the S-parameter block, the complex input signal is partly downconverted on the I and on the Q branch, and thus the output power on the two branches is different. For this reason, the gain and other specs of direct conversion receivers are measured at an arbitrary low frequency.

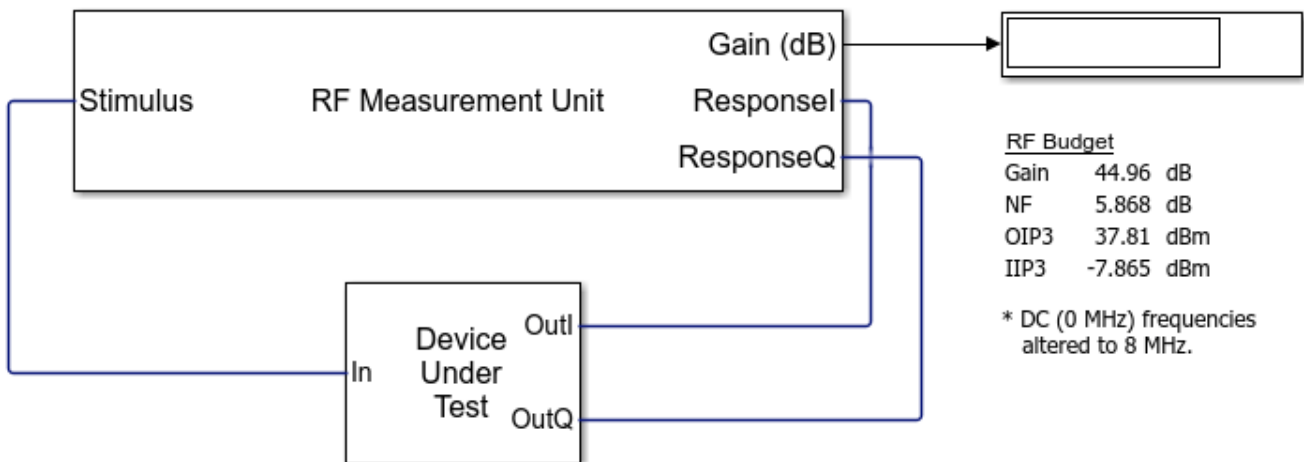
Generate Measurement Testbench

Use the **Export** button in the **RF Budget Analyzer** app to create a measurement testbench or:

```
exportTestbench(b)
save_system(gcs, 'model_2')
```

RF Measurement Testbench

Open the Block Parameters dialog of the RF Measurement Unit block for measurement-specific [parameters](#) and [instructions](#).



To measure the gain, noise figure, and OIP3 use the RF Measurement Unit dialog box to choose the value you want to verify.

Observe and understand the testbench block:

- You can measure the output on the I or Q branches.
- Measurements are done at an arbitrary low frequency
- Measurements are done in the time domain over an arbitrary signal bandwidth

Run the following simulation:

- Measure the gain (disable the noise for accurate measurements).
- Measure the NF. Reduce the baseband bandwidth to 8e3 for narrowband measurements. In this way, the noise figure measurement is not affected by the filter selectivity.
- Measure the OIP3. Keep the smaller baseband bandwidth and disable noise for accurate measurements.

On comparing, you will see that the values of gain, noise figure, and IP3 match the values in the **RF Budget Analyzer** app reported in the testbench.

See Also

RF Budget Analyzer| “Using RF Blockset for the First Time”| “Power Ports and Signal Power Measurement in RF Blockset” on page 8-11| “Create Custom RF Blockset Models” on page 8-99

Passband Signal Representation in Circuit Envelope

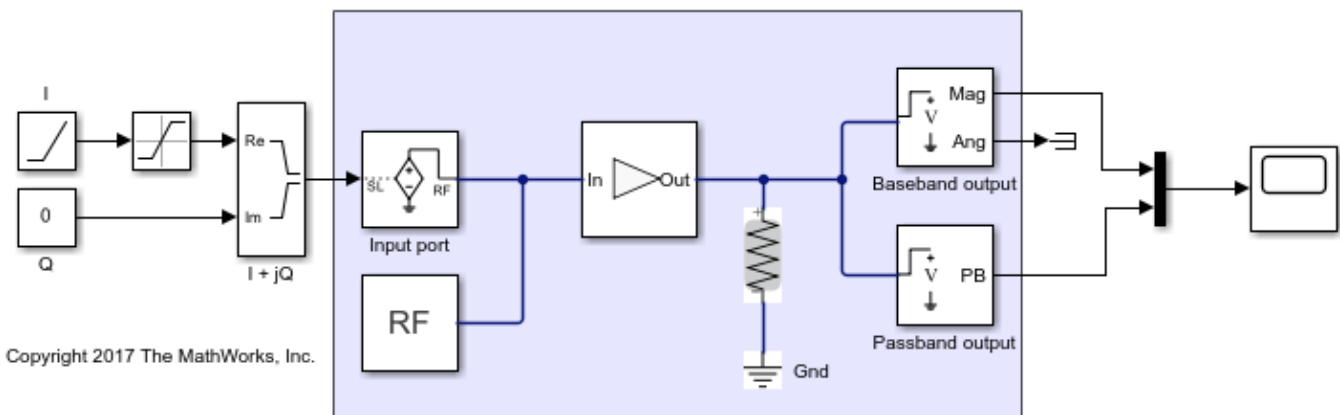
This model shows the relationship between two signal representations in RF Blockset™ Circuit Envelope: complex baseband (envelope) signal and passband (time domain) signal. The step size of a RF Blockset solver is usually much larger than the period of the carrier, so upsampling is necessary to construct a reasonable passband signal.

System Architecture

The system consists of:

- Simulink blocks that generate a complex $I+jQ$ input baseband signal.
- A RF Blockset Inport block that specifies the carrier frequency of the signal as $f = 3\text{GHz}$.
- A simple RF Blockset system that consists of an Amplifier with **0dB** gain and matching **50 Ohm** load (that is, its input and output signals are identical). It has two outputs: baseband (where the complex envelope signal $I+jQ$ is represented as magnitude and angle) and passband, where the actual time domain signal is reconstructed.
- A Scope block that displays baseband magnitude (that is, the envelope of the signal) versus the passband (actual) signal.

```
model = 'simrfv2_passband';
open_system(model)
```



Definition of the Passband Signal

RF Blockset interprets the complex signal $I(t) + jQ(t)$, as a modulation (envelope) of the sinusoidal carrier signal with a frequency f . By default, RF Blockset assumes that the carrier signal is normalized (that is, its average power is equal to 1), so the passband signal is

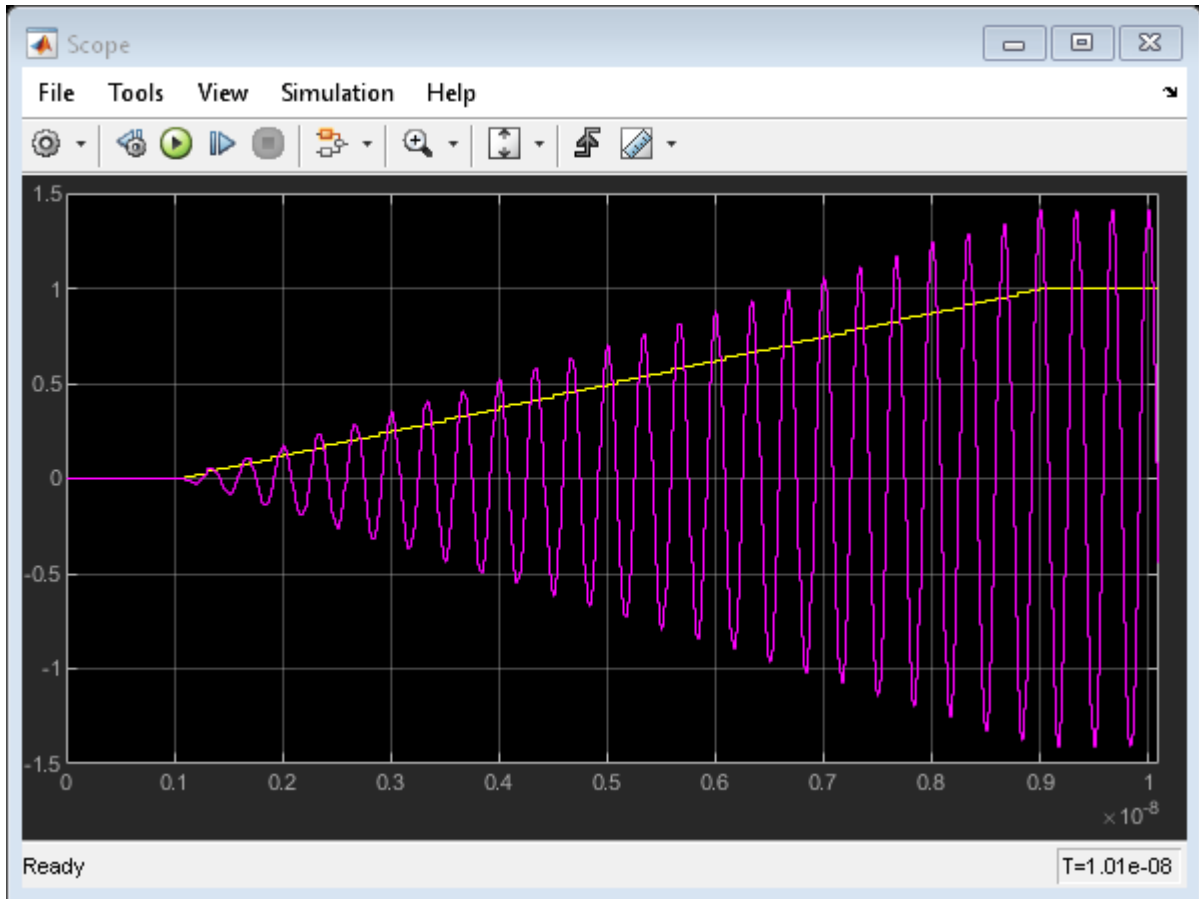
$$s(t) = I(t)\sqrt{2}\cos 2\pi ft - Q(t)\sqrt{2}\sin 2\pi ft$$

With this definition, the average power of the signal is

$$\overline{s^2(t)} = I^2 + Q^2.$$

In this example, I is a ramp that goes from 0 to 1 and $Q = 0$.

```
scope = [model '/Scope'];
set_param(scope, 'YMax', '1.5');
set_param(scope, 'YMin', '-1.5');
open_system(scope)
sim(model);
```



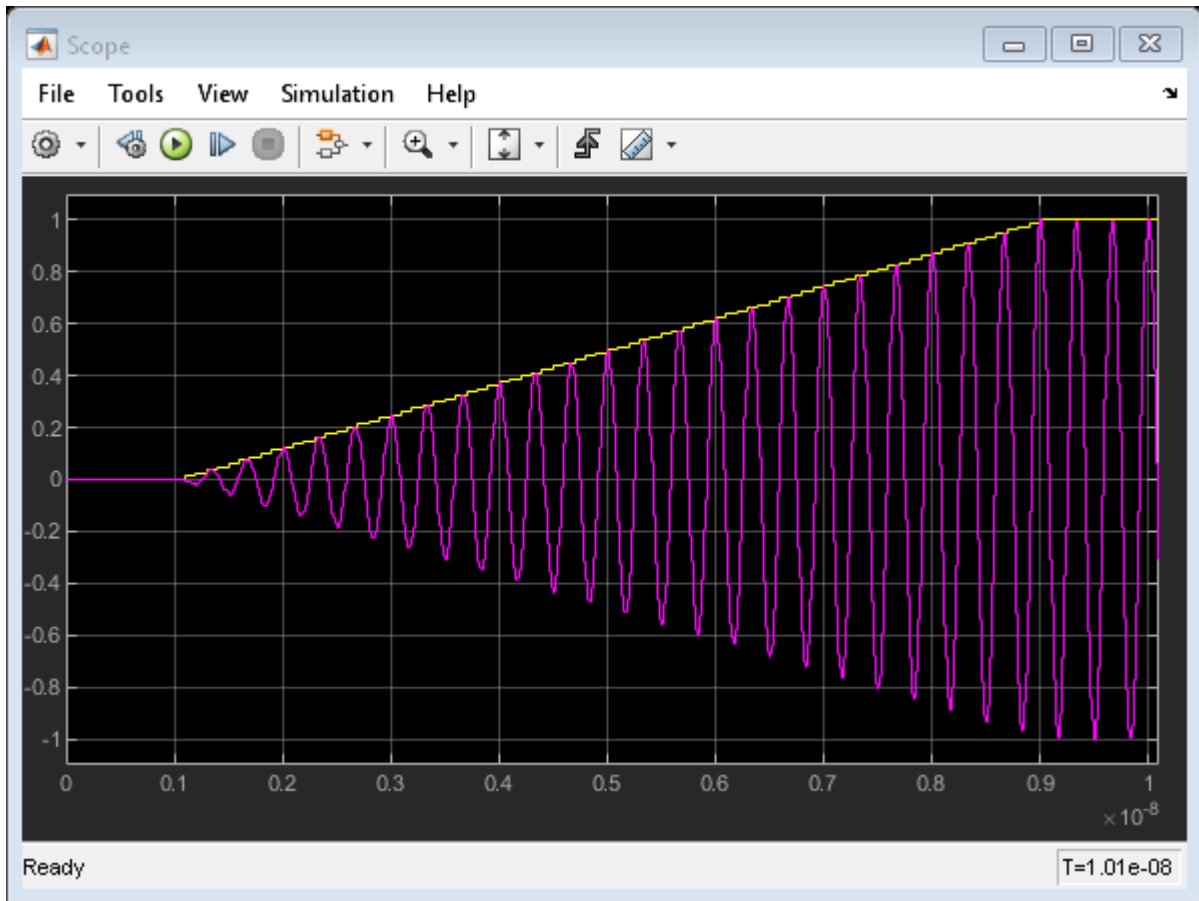
When the **Normalize Carrier Power** Option on the **Configuration** block is not selected, RF Blockset assumes that $I(t) + jQ(t)$ represent the peak values of the carrier, that is

$$s(t) = I(t) \cos 2\pi ft - Q(t) \sin 2\pi ft$$

and the average power of the signal is therefore

$$\overline{s^2(t)} = (I^2 + Q^2)/2$$

```
params = [model '/Configuration'];
set_param(params, 'NormalizeCarrierPower', 'off');
set_param(scope, 'YMax', '1.1');
set_param(scope, 'YMin', '-1.1');
sim(model);
```



Effects of the Normalize Carrier Power Option

It is very important to understand that when you change the **Normalize Carrier Power** option, RF Blockset changes the interpretation of the complex input/output $I(t) + jQ(t)$ baseband signal. Consider the simple case when the input baseband voltage is constant, $I = 1$ and $Q = 0$. The amplifier has a gain of **0dB**, which means that the output signal is the same as the input.

When the **Normalize** option is checked, the output baseband voltage is equal to $I_{out} = 1$, the output passband voltage is $\sqrt{2} \cos 2\pi ft$, and the average power at the **R = 50 Ohm** load is $1^2/50 = 0.02$.

When the **Normalize** option is unchecked, the output baseband signal does not change, $I_{out} = 1$, while the output passband signal is now $\cos 2\pi ft$, which means that the average power is $1^2/50/2 = 0.01$.

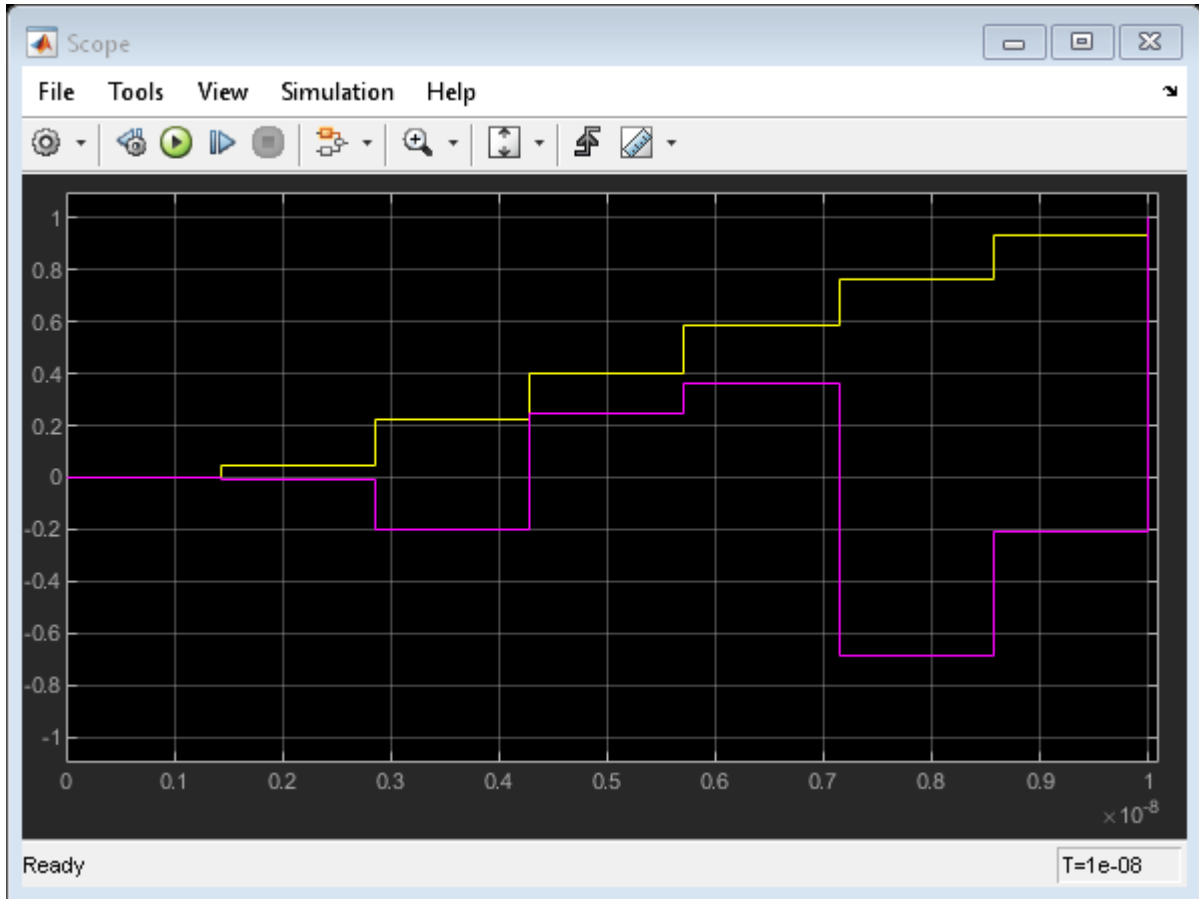
In other words, for linear models the **Normalize** option does not affect the baseband output, but affects the actual passband signal and the average power formula.

Note that the zero carrier frequency is special: the passband and baseband representations for $f = 0$ are always the same: $s(t) = I(t)$

Simulation Step Size Versus Passband Output Step Size

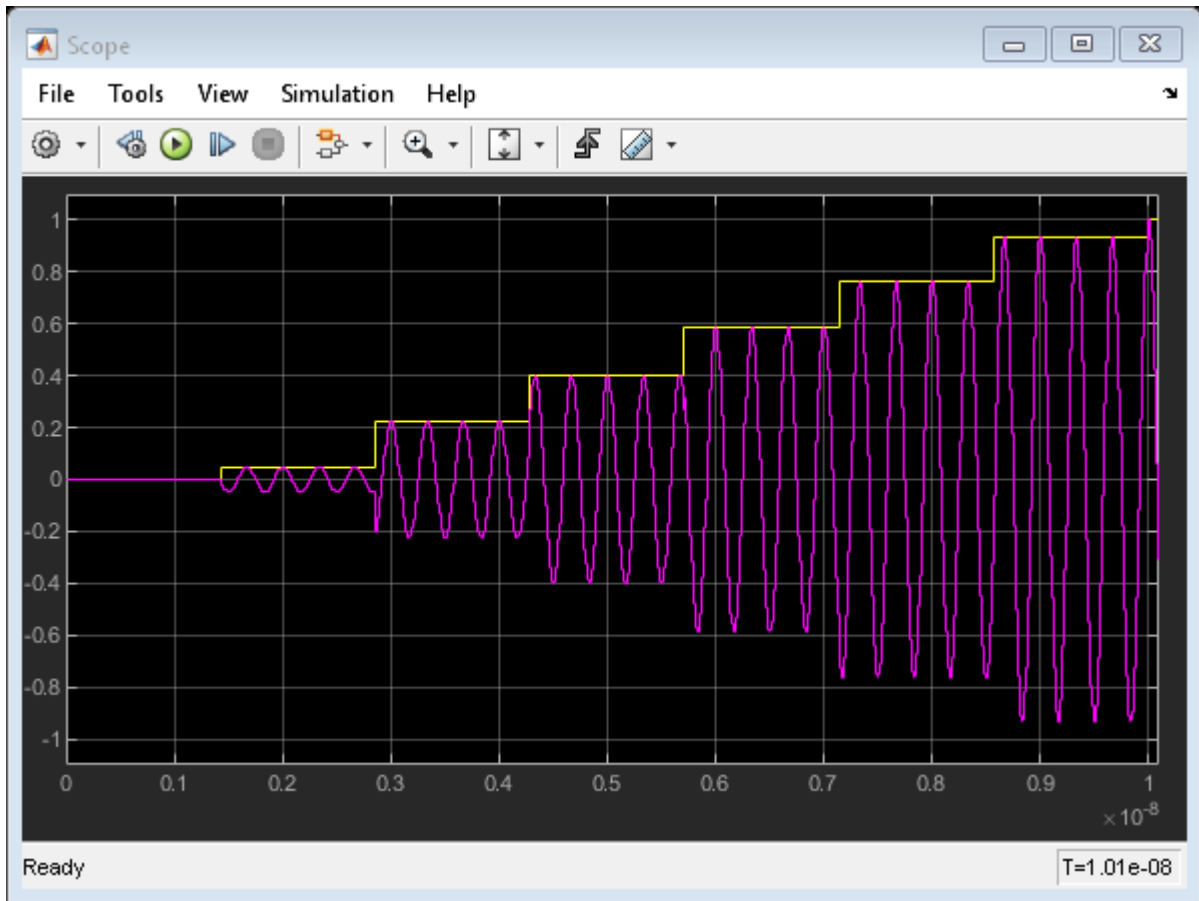
In general, the RF Blockset simulation step is much larger than the period of the carrier, which allows faster simulation compared to regular methods. For such time steps the passband output is severely undersampled and exhibits aliasing effects. Set the **Step Size** value of the **Configuration** block to the large value **$1e-8/7$**

```
set_param(params, 'StepSize', '1e-8/7')
sim(model);
```



To obtain a realistic passband signal, resample the signal in the output. Change the **Step Size** parameter of the **Passband output** block from **-1** (which means that the step size is inherited from RF Blockset simulation) to **$1e-11$** .

```
output = [model '/Passband output'];
set_param(output, 'StepSize', '1e-11');
sim(model);
```



Notes:

- Generating passband output at a higher rate (compared to RF Blockset simulation) requires resampling the signal's envelope. Current implementation uses a zero-hold resampling method that introduces "stepping" artifacts. Better interpolation techniques require delaying the output by several time steps.
- The **'auto'** time step option is available on the RF Blockset Outport block (the time step is selected to resolve the highest output carrier frequency).
- Passband output might slow RF Blockset simulation because of the higher output sampling rate.

```
bdclose(model)
```

See Also

Configuration | Amplifier

Related Topics

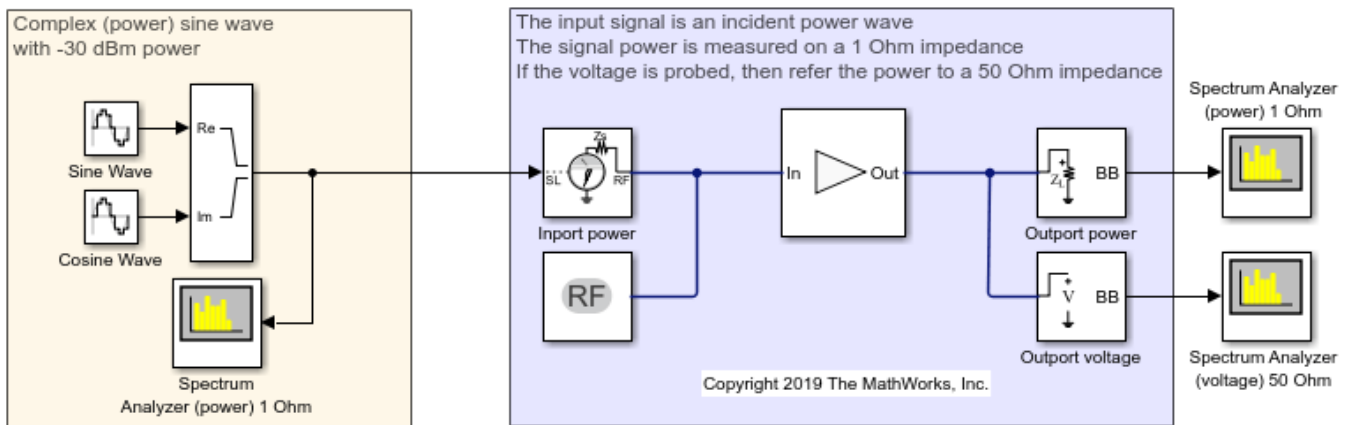
"Getting Started with RF Modeling" on page 8-2

Power Ports and Signal Power Measurement in RF Blockset

This example shows how to use power ports and measure the signal power using the spectrum analyzer.

Power Model

```
model = 'simrfV2_powerportdefinition_1.slx';
open(model)
```



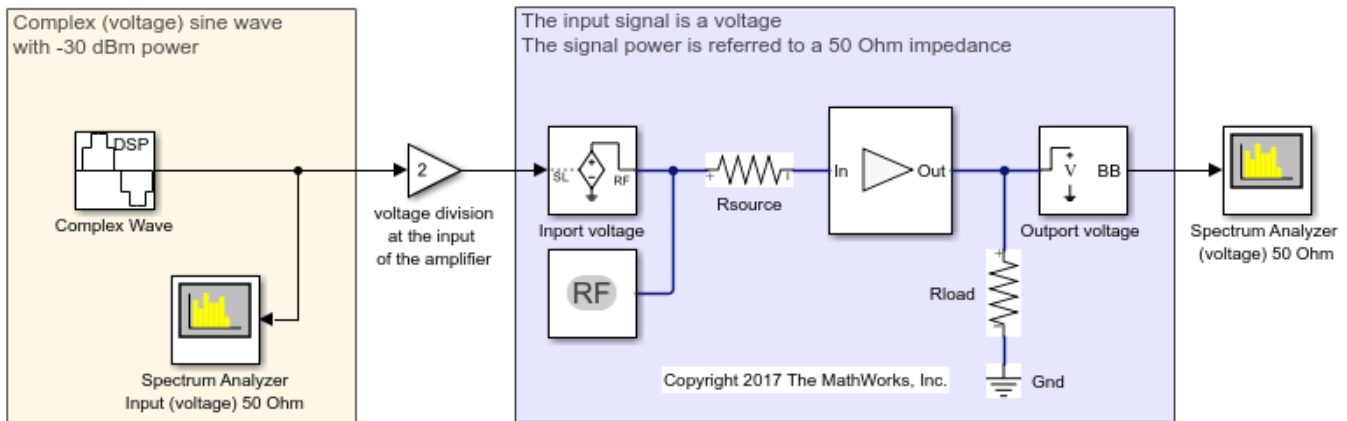
In this model, the Simulink signal is interpreted as a $|V_{rms}^2 \text{ signal}|$ referred to 1 Ohm. This is the implicit convention used in the DSP System Toolbox.

When measuring the signal power with the spectrum analyzer, refer the measurement to 1 Ohm. Use the power option for the ports at the input and output of the system. Notice that the ports automatically add source and load impedances, and scale the signal to refer it to the specified impedance. In this way, the available power of the input signal is referred to 50 Ohm. The RF Blockset amplifier is thus processing a signal with the same power as the Simulink signal.

When probing internal nodes, use the voltage option in the "sensing" port to avoid mismatch in the circuit. Also, adjust the reference impedance in the spectrum analyzer to 50 Ohm.

Voltage Model

```
model = 'simrfV2_powerportdefinition_2.slx';
open(model)
```



In this model, the Simulink signal is interpreted as a Voltage signal. When measuring the signal power with the spectrum analyzer, refer the measurement to 50 Ohm. Notice that the two sources and the "Real-Imag to Complex" of the Power Model are replaced using a single Sine Wave block with property `Output complexity` set to "Complex".

In RF Blockset, use the voltage option for the ports at the input and output of the system. Add source and load impedances to avoid mismatch in the amplifier. Also add a factor of 2 to make sure that the voltage at the input of the RF Blockset amplifier has the same value as the Simulink signal. The factor of 2 takes into account the voltage division between the source and input impedances.

Simulate both the models and observe.

See Also

Amplifier | Inport | Spectrum Analyzer

Related Topics

"Getting Started with RF Modeling" on page 8-2 | "Passband Signal Representation in Circuit Envelope" on page 8-6

Communications System with Embedded RF Receiver

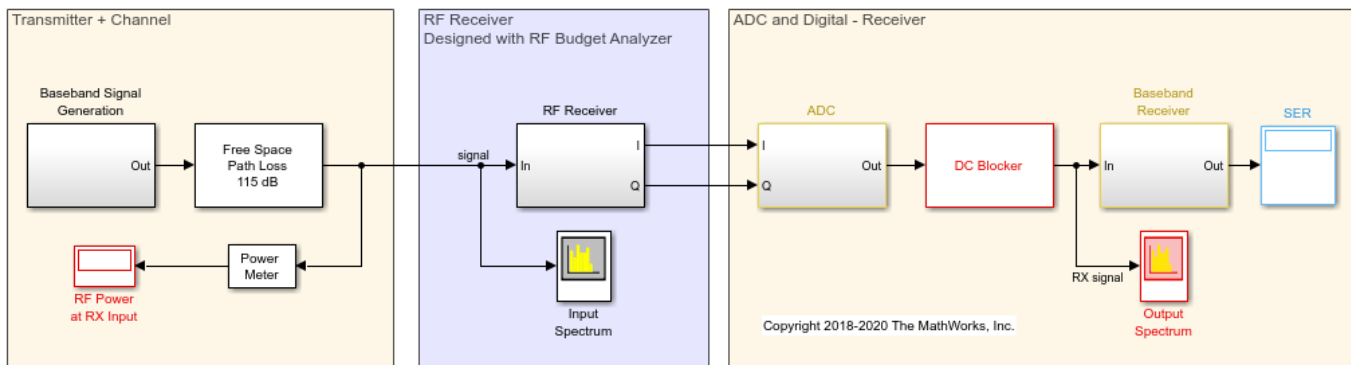
This example shows how to integrate an RF receiver together with baseband signal processing algorithms to model an end-to-end communications system.

The example requires Communications Toolbox™.

Part 1: Baseband Communications Link with Integrated RF Receiver Model

The following model includes a baseband signal generator, a simple channel, an RF receiver initially designed using the RF budget analyzer as described in “Getting Started with RF Modeling” on page 8-2, an analog-to-digital conversion, a demodulation scheme, and a computation block for the symbol error rate.

```
model = 'simrfv2_comms_rf_example';
open_system(model);
```



For this model, blocks from Communications Toolbox and DSP System Toolbox™ are used to perform baseband signal processing. The nonstandard compliant baseband signal has a rectangular QAM constellation with raised cosine filtering and the baseband receiver does not include carrier/clock synchronization. Parameters for the baseband signal generation are defined in the Model Properties -> Model callbacks **PreLoadFcn**, which sets these parameters in the MATLAB workspace when the model is loaded:

- $BW = 8 \text{ MHz}$;
- $T_{\text{step}} = 125 \text{ ns}$; % 1/BW
- $\text{FrameLength} = 128$;
- $M = 4$; % Constellation size 2^M
- $T_{\text{symbol}} = 64 \text{ us}$; % $M \cdot \text{FrameLength} \cdot T_{\text{step}}$

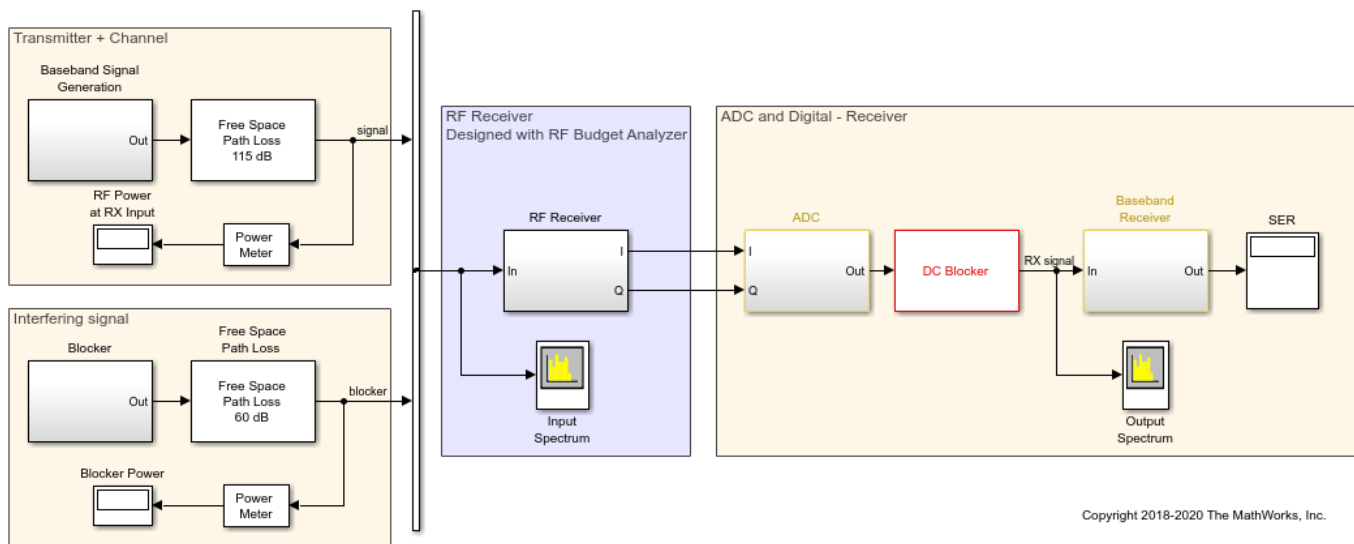
Sample time for the baseband signal and Step size of the RF Blockset receiver Configuration block have the same value. This guarantees that the RF simulation bandwidth is consistent with the sampling rate of the input signal. The RF Blockset Receiver has input and output ports that convert the Simulink signals into RF domain quantities and scale their power to a 50 Ohm reference impedance. The input port centers the baseband signal at a specified center frequency of 2.45 GHz, and the RF IQ Demodulator downconverts the input signal to baseband with a single quadrature stage.

```
bdclose(model);
```

Part 2: Include an Out-of-Band Interfering Blocker Signal

The model `simrfv2_comms_rf_interferer` shows how to add a high power out-of-band interferer centered around 2.5 GHz. This blocker affects the RF receiver by driving it into the nonlinear region. Use the following steps to complete this task.

```
model = 'simrfv2_comms_rf_interferer';
open_system(model);
```



Add an 8-PSK Modulator Baseband block source to include a blocking signal with a higher power level than the transmitter signal. Using the Vector Concatenate block, combine the baseband and blocker signals. The input signal to the RF receiver is now composed of two complex baseband signals. It is important that the two baseband sources use the same sample rate to insure equal simulation bandwidths for each signal (same envelope bandwidth). If the two signals don't have the same sample time, they need to be resampled before combining. This is the recommended best practice for simulating blocker signals when they are "far away" from the desired signal in the frequency spectrum and cannot be included in the same envelope for a particular carrier. To display the spectral positioning of the two input signals in the Spectrum Analyzer block, the `Offset` option has two frequencies specified for the two baseband signals.

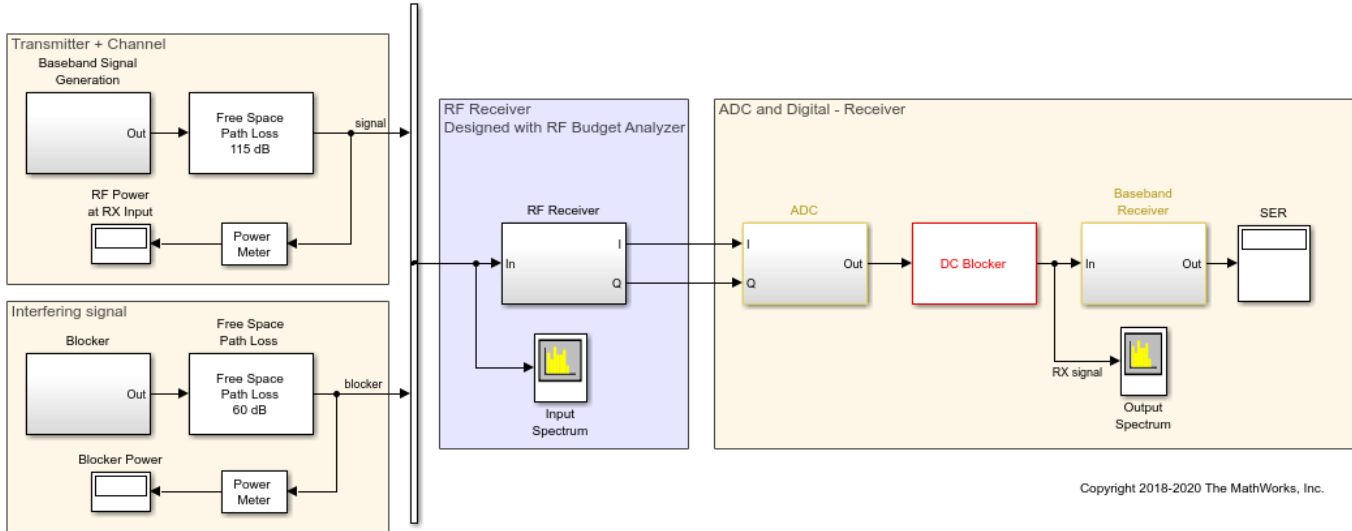
The input port of the RF receiver has been modified to include the two carrier (`Carrier frequencies`) signals (2.45 GHz and 2.5 GHz). Initially we leave the configuration block to automatically select the fundamental tones and the harmonic order.

```
bdclose(model);
```

Part 3: Add Imperfections to the RF Receiver

The model `simrfv2_comms_rf_impairments` shows how to add impairments to the RF receiver that were initially not estimated in the link budget of the RF Budget Analyzer.

```
model = 'simrfv2_comms_rf_impairments';
open_system(model);
```



Under the mask of the RF receiver, modify the RF demodulator to add imperfections that will be driven by the blocking signal. In the mask of the IQ demodulator change these parameters:

- I/Q gain mismatch = 0.5 dB
- I/Q phase mismatch = 1 degree
- L0 to RF isolation = 85 dB
- IIP2 = 45 dB
- Phase noise frequency offset = [1e5 5e5 2e6] Hz
- Phase noise level = [-95 -120 -140] dBc/Hz

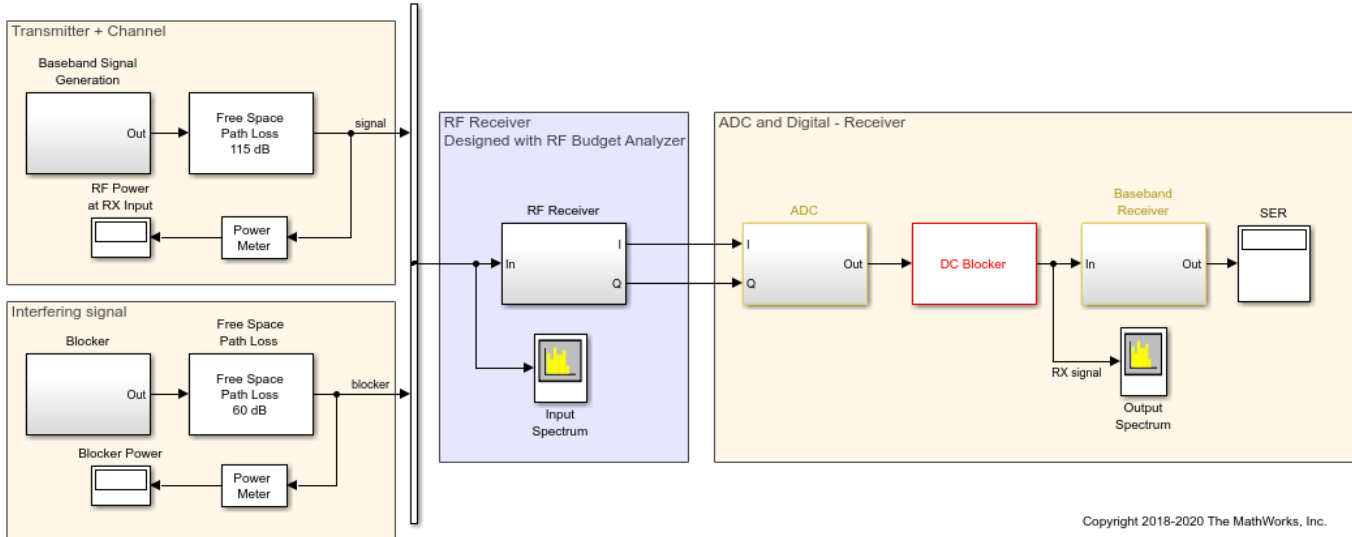
Each of these imperfections separately increases the bit error rate. These imperfections cause finite image rejection and a DC offset that is removed in the baseband domain. As observed, the DC offset correction requires time to integrate the signal power and remove the DC component. To further modify the structure of the I/Q demodulator system, you can click on the "Edit System" button. With this operation you disable the link to the library, inline the value of the parameters, and have the ability to manually modify the block parameters as well as the block architecture.

```
bdclose(model);
```

Part 4: How to Decrease Simulation Time

The model `simrfv2_comms_rf_speed` shows how to decrease the simulation time of the previous model described in this example. Follow these steps to speed up the simulation of the model.

```
model = 'simrfv2_comms_rf_speed';
open_system(model);
```



In Simulink, select Accelerator mode to speed up the simulation by leveraging automatic C code generation.

In the RF Blockset section, to speed up simulation reduce the Harmonic order of the Circuit Envelope configuration block. Uncheck Automatically select fundamental tones and harmonic order and set the Harmonic order equal to 3. The Total simulation frequencies is reduced from 61 to 25, equivalent to an approximate 2.5 times speed up. After reducing the Harmonic order, verify that simulation results do not change.

To further increase simulation speed, use Frequency domain modeling instead of Time domain modeling for the S-parameters SAW filter block. You need to verify that when changing the “Compare Time and Frequency Domain Simulation Options for S-parameters” on page 8-40 approach the simulated transfer function is still correct and that the model uses a sufficiently long Impulse response duration.

With the above modifications, the simulation is approximately five times faster without significantly affecting the simulation results.

```
bdclose(model);
clear model;
```

Related Topics

“Getting Started with RF Modeling” on page 8-2

Automatic Sample-Time Interpolation at Input Port

This example shows how to manage models consisting of both digital communication and RF systems that process signals at different sampling rates. To perform a model simulation where the Nyquist sampling rate of the digital communication signal is less than the inverse of the RF section time step an interpolation filter will be employed. The use of this interpolation filter diminishes the introduction of artificial signal artifacts at the boundaries of the communication and RF systems resulting from the sampling rate differences.

Part 1: Single signal entering the RF system

The following model includes a Zigbee (802.15) baseband signal feeding a direct conversion RF receiver. The ZigBee baseband transmitter is built using blocks from Communications Toolbox™ and DSP System Toolbox™ while the RF receiver is constructed using blocks from the RF Blockset™ Circuit Envelope library.

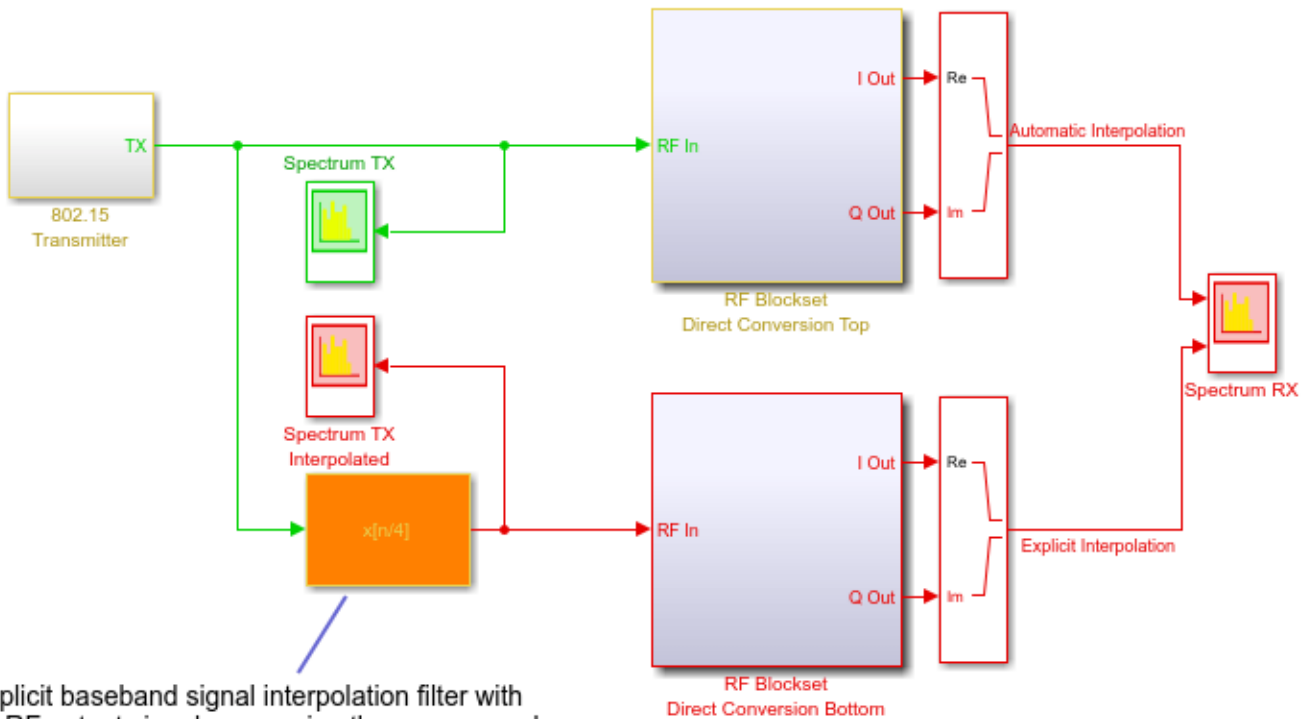
For the RF Blockset Circuit Envelope solver it is recommended to use a simulation time step that is 4 to 8 times smaller than the reciprocal of the input baseband signal sample time. This provides a simulation bandwidth that is sufficient for the RF solver to capture artifacts at the edge of the bandwidth accurately and the physical effects that require additional bandwidth such as spectral regrowth. In general, using an interpolation factor of 4 to 8 increases the simulation bandwidth beyond the Nyquist rate of the baseband signal generated in the transmitter.

In this model, the two different signal sample rates are:

- green for the communications baseband signal
- red for the RF circuit envelope signal

```
model = 'simrfv2_sampletime_example';  
open_system(model)  
sim(model)
```

```
% Hide all scopes (see PostLoadFcn Model Callback for more details):  
SpTxScopeConf.Visible = false;  
SpTXiScopeConf.Visible = false;  
SpRxScopeConf.Visible = false;
```

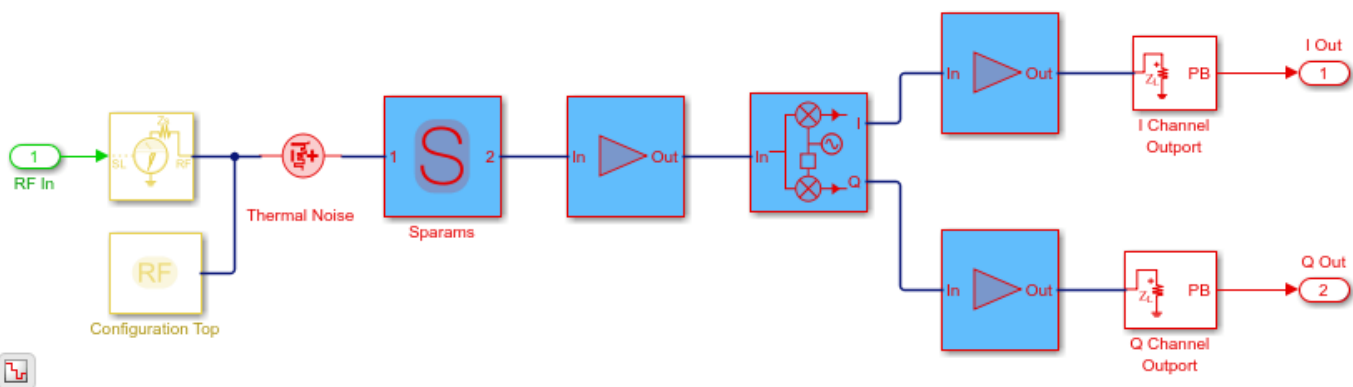


Explicit baseband signal interpolation filter with an RF output signal possessing the same sample rate as the automated filter in the top receiver.

Copyright 2018-2020 The MathWorks, Inc.

The Top and Bottom RF receiver systems in the model are identical and consist of Pre-LNA filter, followed by an LNA, quadrature demodulator, and another amplification stage. All RF components include typical impairments such as noise, nonlinearity and finite isolation.

```
open_system([model '/RF Blockset Direct Conversion Top'])
```



As specified in the Configuration block Mask Parameters dialog box, the simulation is performed with the input interpolation filter enabled for the top receiver,

Block Parameters: Configuration Top

Configuration

Define RF Blockset Circuit Envelope system simulation settings.

Main **Advanced**

Spectrum

Automatically select fundamental tones and harmonic order

Fundamental tones: GHz

Harmonic order:

Total simulation frequencies: 4

Step size: s

Envelope bandwidth: 16 MHz

Noise

Simulate noise

Use default random number generator

Noise seed:

Temperature: K

Input/Output Signals

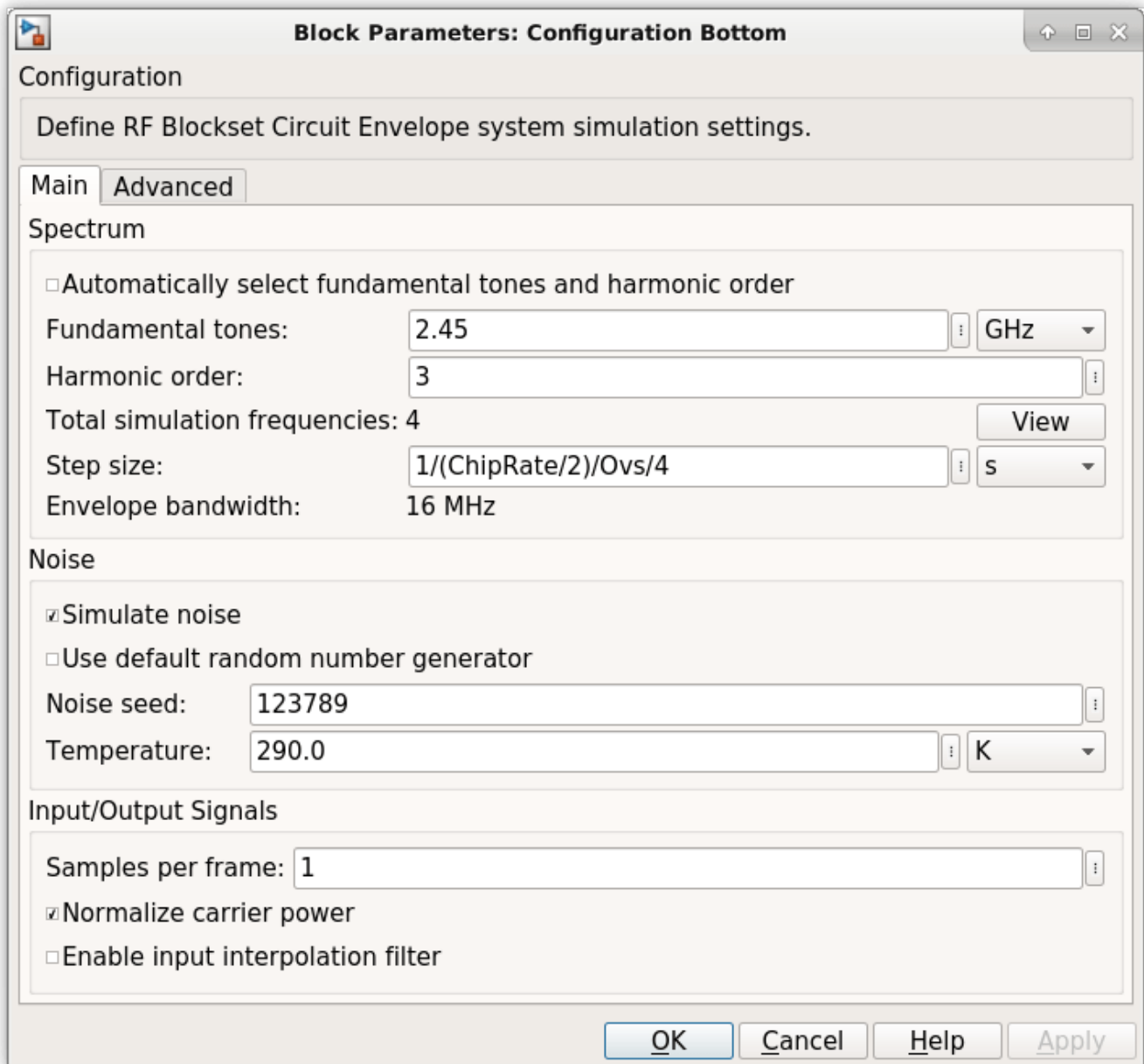
Samples per frame:

Normalize carrier power

Enable input interpolation filter

Filter delay (in samples): 320

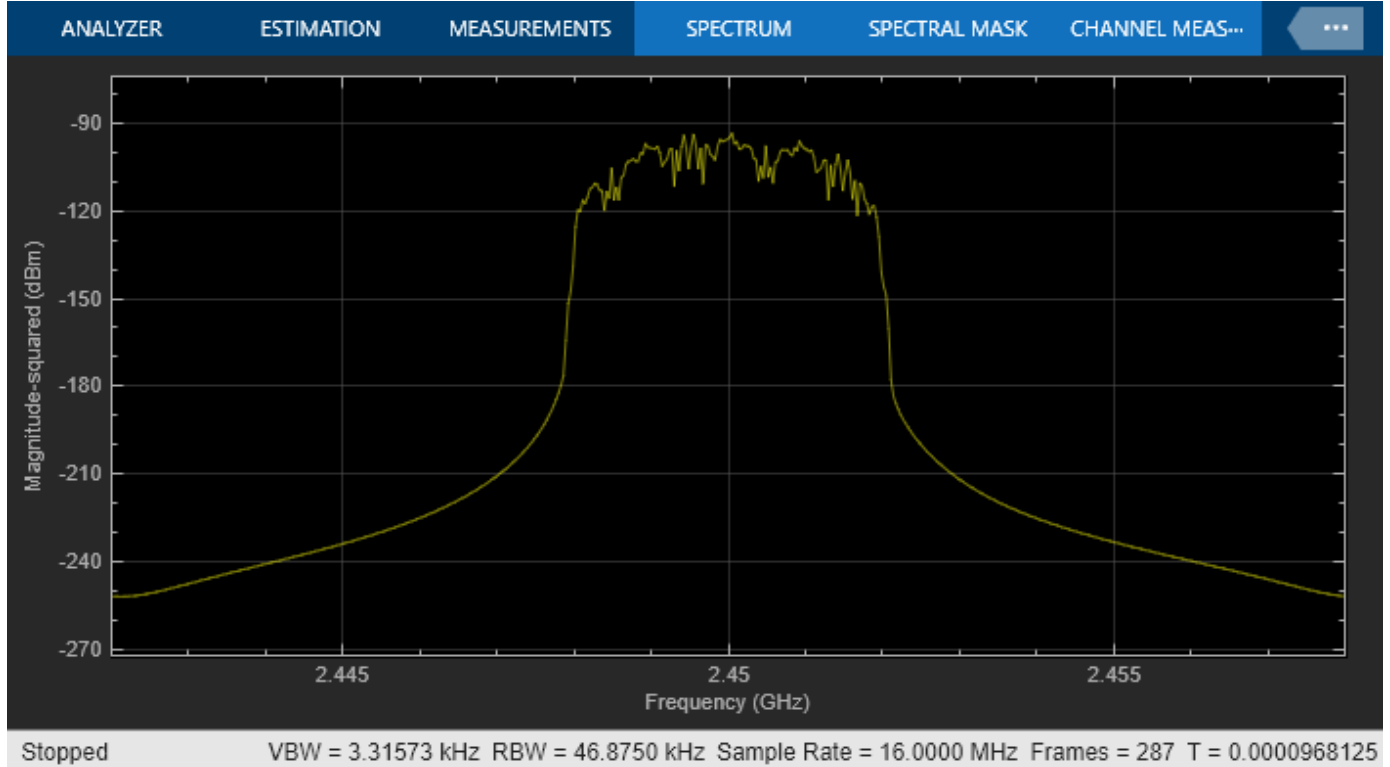
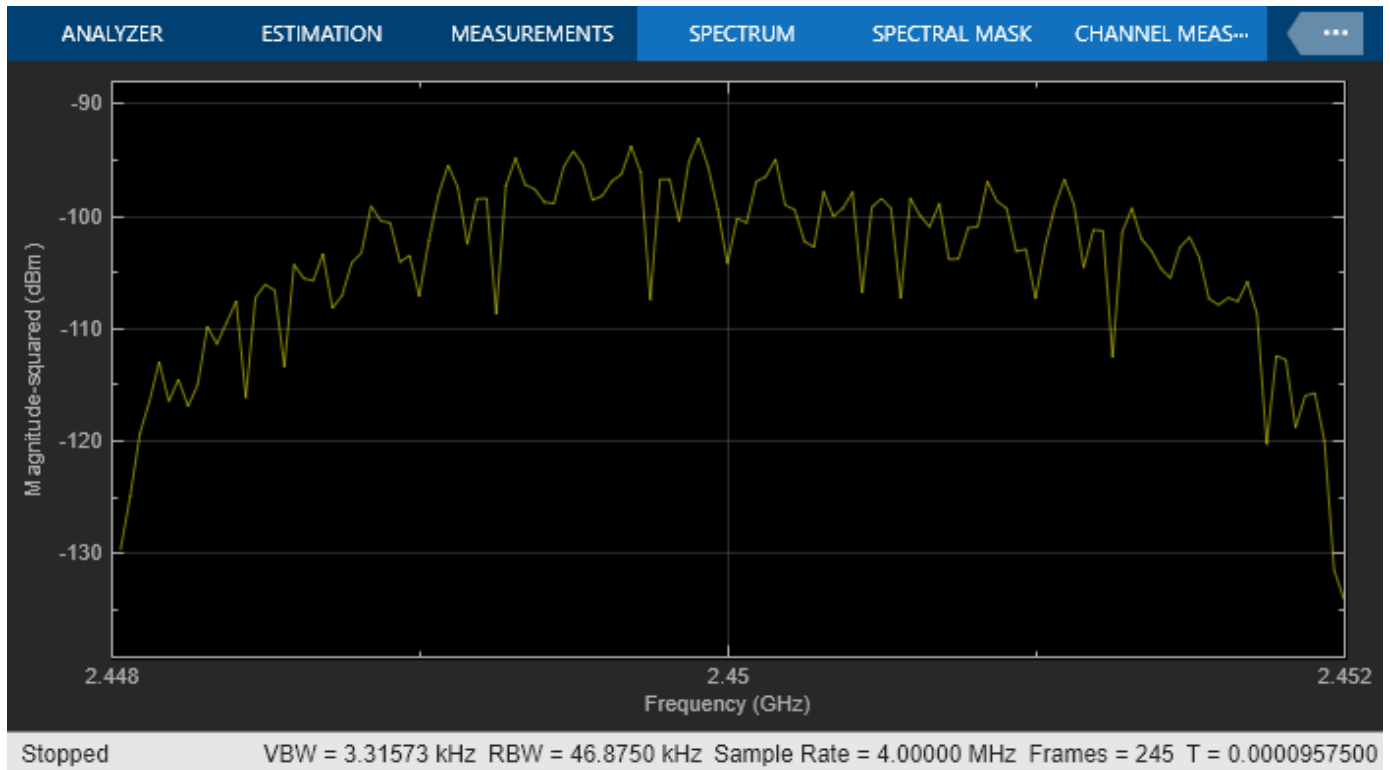
and disabled for the bottom receiver.



The top RF receiver is fed with a baseband signal possessing a sample rate 4 times slower than the reciprocal of the RF simulation step size set in its Configuration block. The RF Inport block automatically interpolates the input signal at the required RF rate.

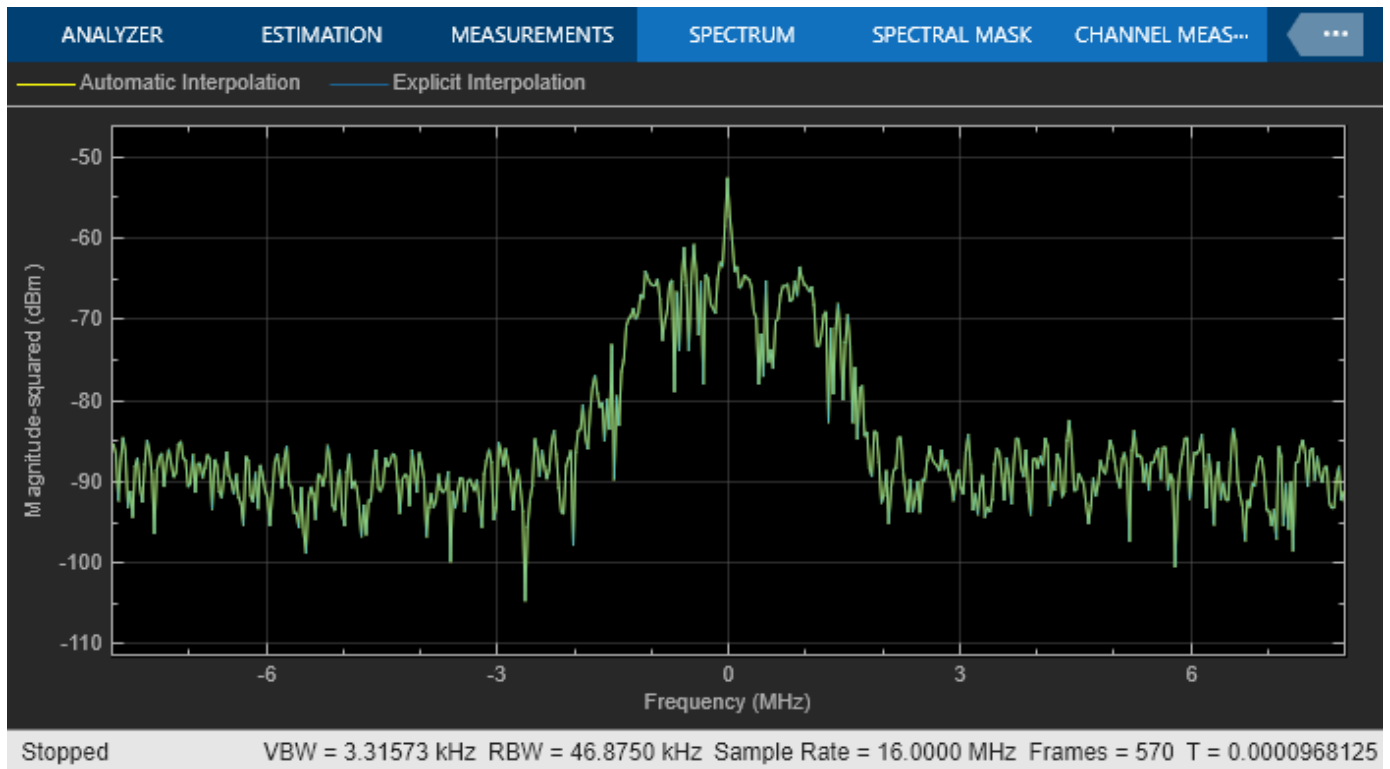
The bottom RF receiver is fed with a baseband signal sample rate equaled to the reciprocal of the step size specified in its RF Configuration block. The bottom RF receiver uses an explicit interpolation filter highlighted in orange to up-sample the communication baseband signal.

```
% Show these two scope results:
SpTxScopeConf.Visible = true;
SpTXiScopeConf.Visible = true;
```



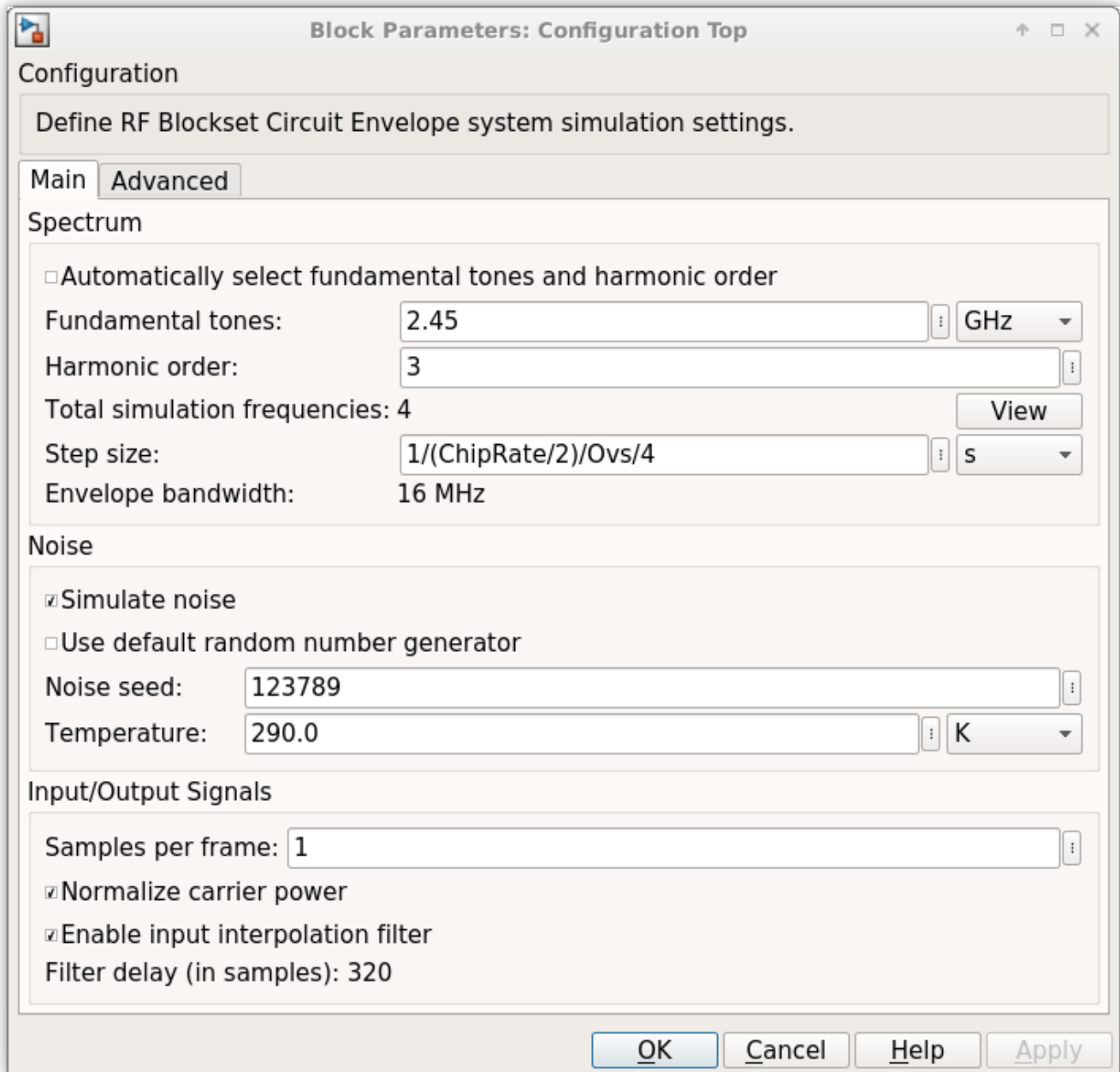
The outputs of both receivers are the same, since both input signals are resampled by interpolation filters to reduce sample rate transition aliasing effects. In the top receiver, the sample rate transition is automatically managed by the circuit envelope Inport block. In the bottom receiver, the sample rate transition is explicitly managed by the addition of the interpolation filter.

```
% Show this scope result:
SpRxScopeConf.Visible = true;
```



Using an interpolation filter improves the spectral results of the simulation, but comes at a price: it introduces a delay. Since an FIR filter is used for the interpolation, the delay corresponds to half the number of filter coefficients. In this case, the filter has 640 taps and introduces a delay of 320 time steps at the faster RF sample rate or 80 time steps at the slower baseband communication sample rate. In case of multiple baseband communication signal inputs, it may be necessary to compensate for the delay by aligning all signals entering the RF system.

When an input interpolation filter is enabled in the Configuration block Mask Parameters dialog box, the RF signal delay introduced will be displayed next to the enabling switch.



By default, RF Blockset automatically inserts an interpolation filter and resamples the input signal. You might decide to disable the default option and explicitly insert an interpolation filter if you have:

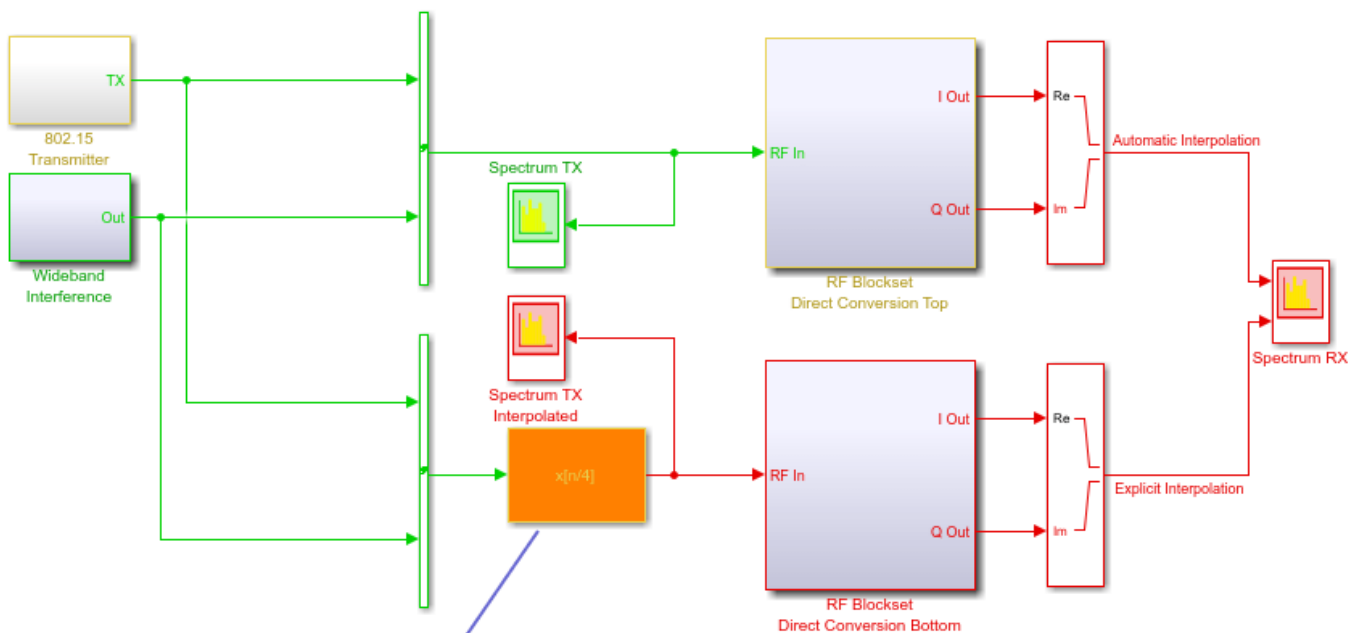
- specific requirements regarding the specifications of the interpolation filter;
- multiple input signals requiring different input ports (case described below);
- Simulink control signals (e.g. applied to VGA, variable phase shifter or switch blocks) that are intrinsically slower than the RF signal and do not necessitate resampling.

Part 2: Multiple signals entering the RF system

The automatic interpolation option discussed above can only support a single RF Inport block. When using multiple Inport blocks, the user is required to manually insert interpolation filters before these blocks. The interpolation filters are then adjusted to have all entering communication signals resampled at the rate specified in the RF Configuration block.

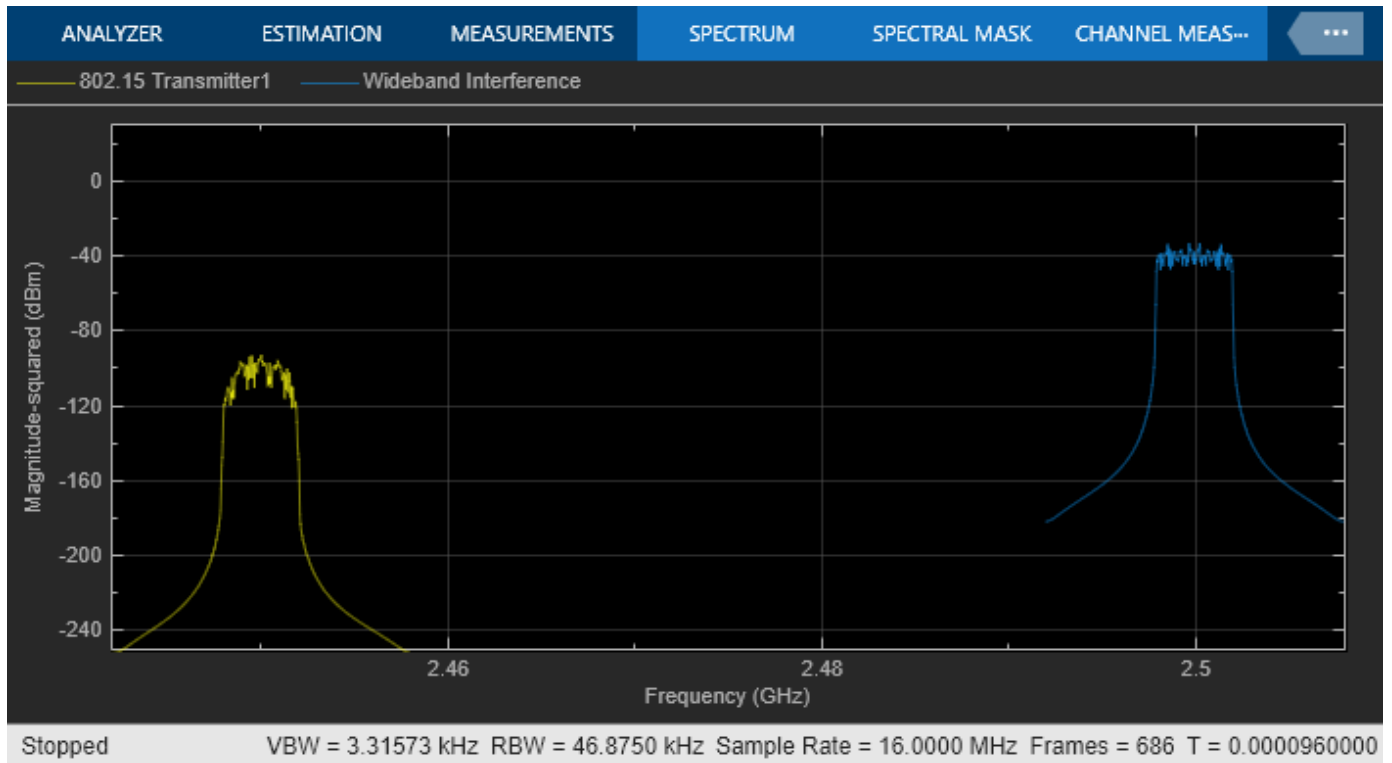
While the RF Blockset Inport block can accept a vector of multiple signals each specified at a different carrier frequency, these signals must have the same sample rates. The following model describes two RF systems with multiple inputs centered on different carriers and correctly resampled. The model is like the one in Part 1 of this example, but also includes a wideband interfering signal that is generated using blocks from Communications Toolbox and DSP System Toolbox. The two input signals have the same sample rate and the RF Blockset Configuration block has a Step size that samples the RF signal 4 times faster than the baseband communications signal.

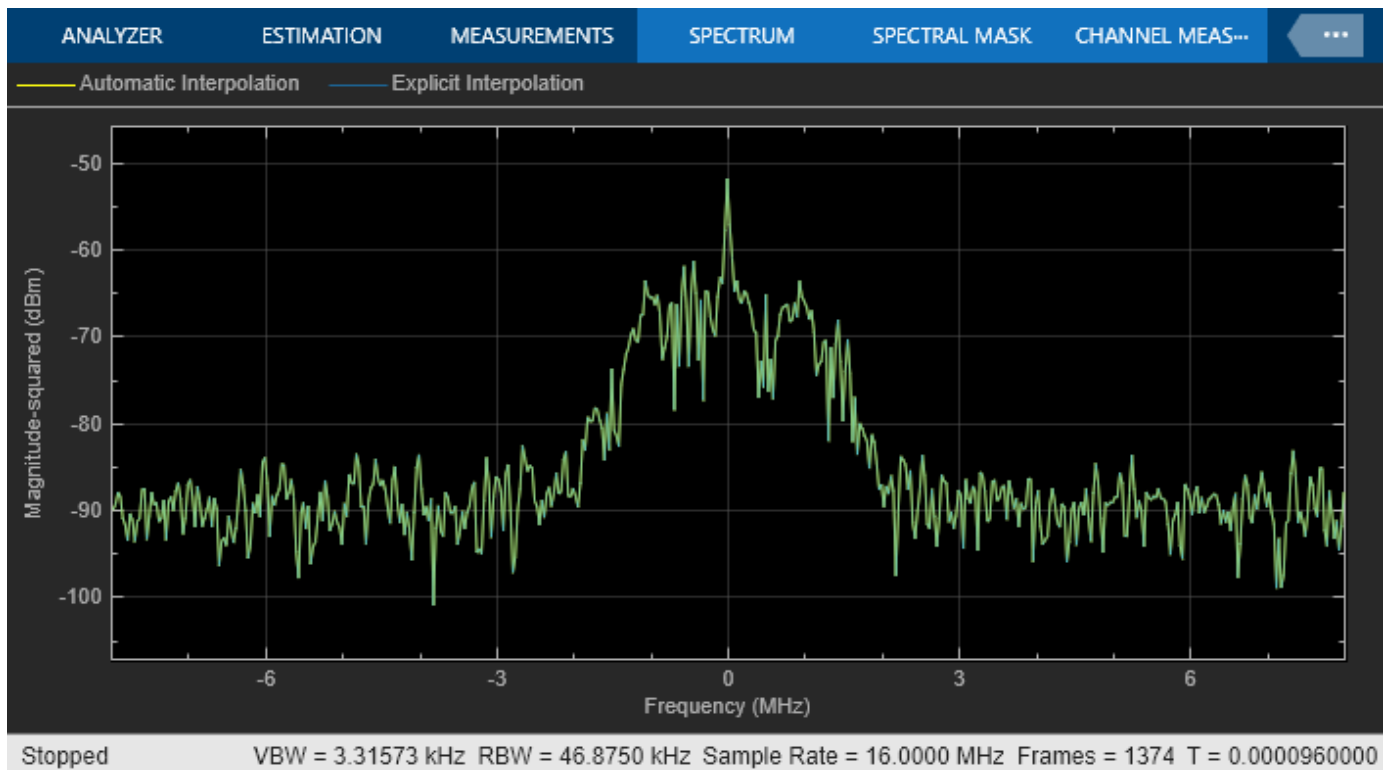
```
bdclose(model);
model = 'simrfv2_sampletime_example_interf1';
open_system(model);
sim(model);
```



Explicit baseband signal interpolation filter with an RF output signal possessing the same sample rate as the automated filter in the top receiver.

Copyright 2018-2020 The MathWorks, Inc.





The model is like the one described in Part 1 of this example. The interpolation filter is necessary to avoid aliasing effects due to rate transition.

A more interesting scenario occurs in the following model when the desired and interferer signals have different sample rates. In this model, the desired signal is explicitly interpolated by the filter (highlighted in orange) and then combined with the wideband interferer as a vector.

To avoid the aliasing effect, the slower rate of the desired input signal is interpolated and filtered before combining with the faster rate interfering signal.

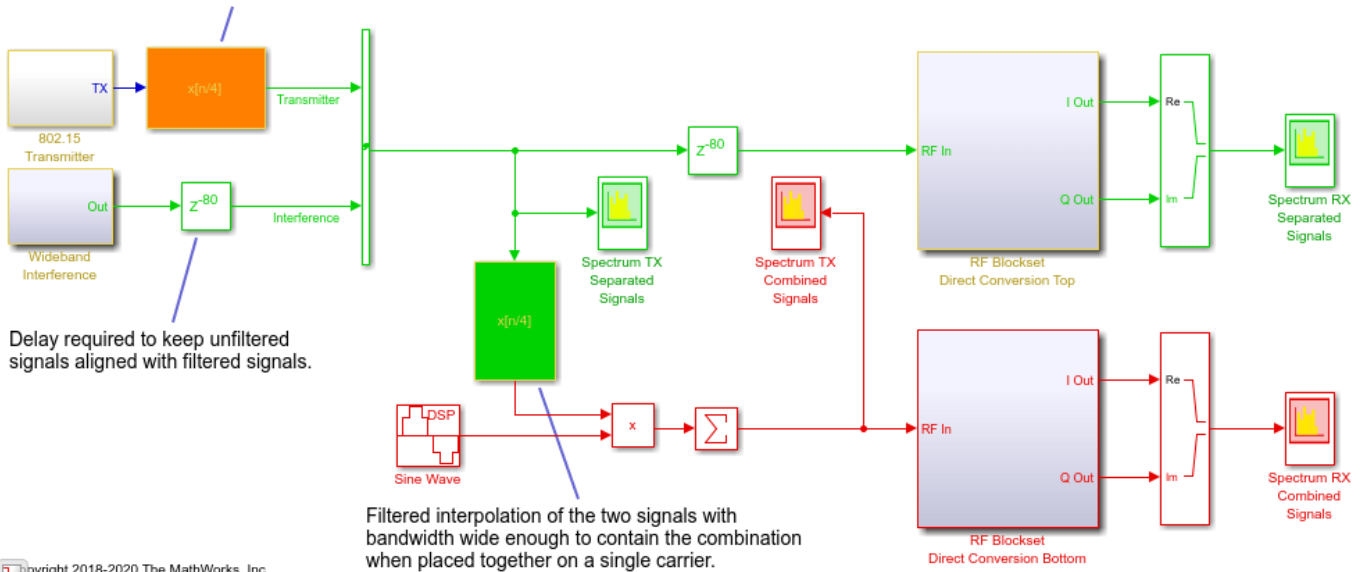
```

bdclose(model);
model = 'simrfv2_sampletime_example_interf2';
open_system(model);
sim(model);

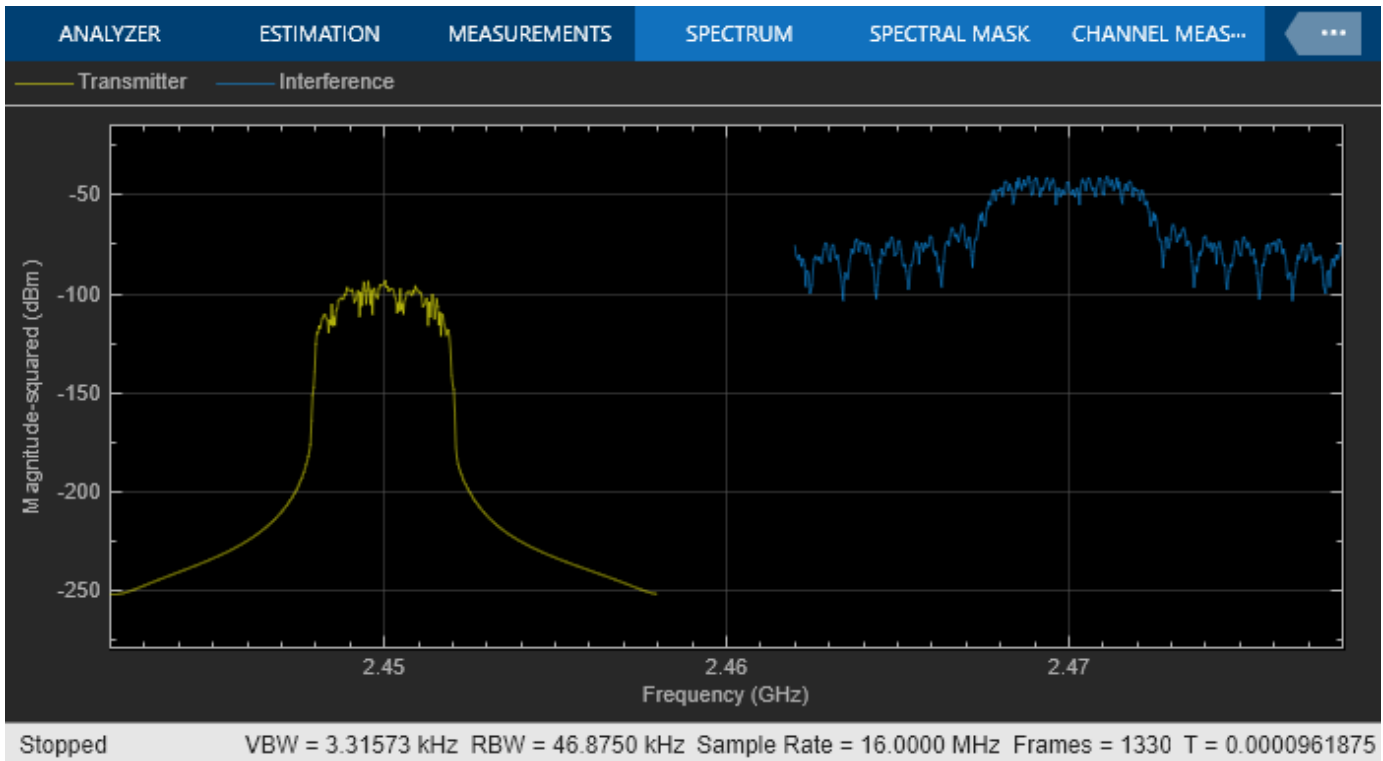
% Hide all scope results (see PostLoadFcn Model Callback for more details):
SpTXComScopeConf.Visible = false;
SpRxSepScopeConf.Visible = false;
SpRxComScopeConf.Visible = false;

```


Filtered interpolation of the transmitter signal producing an output signal with the same sample rate as the Interferer to avoid aliasing when combined.

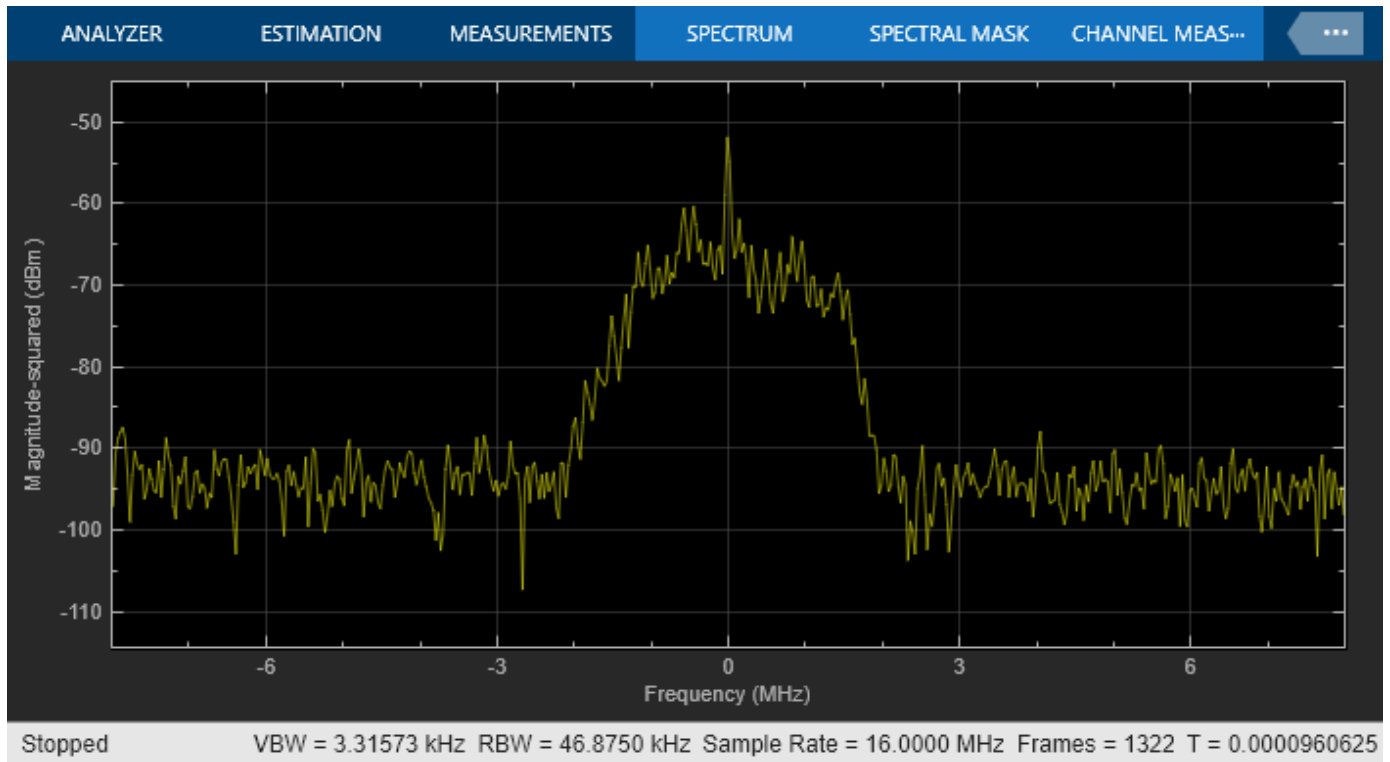


Copyright 2018-2020 The MathWorks, Inc.



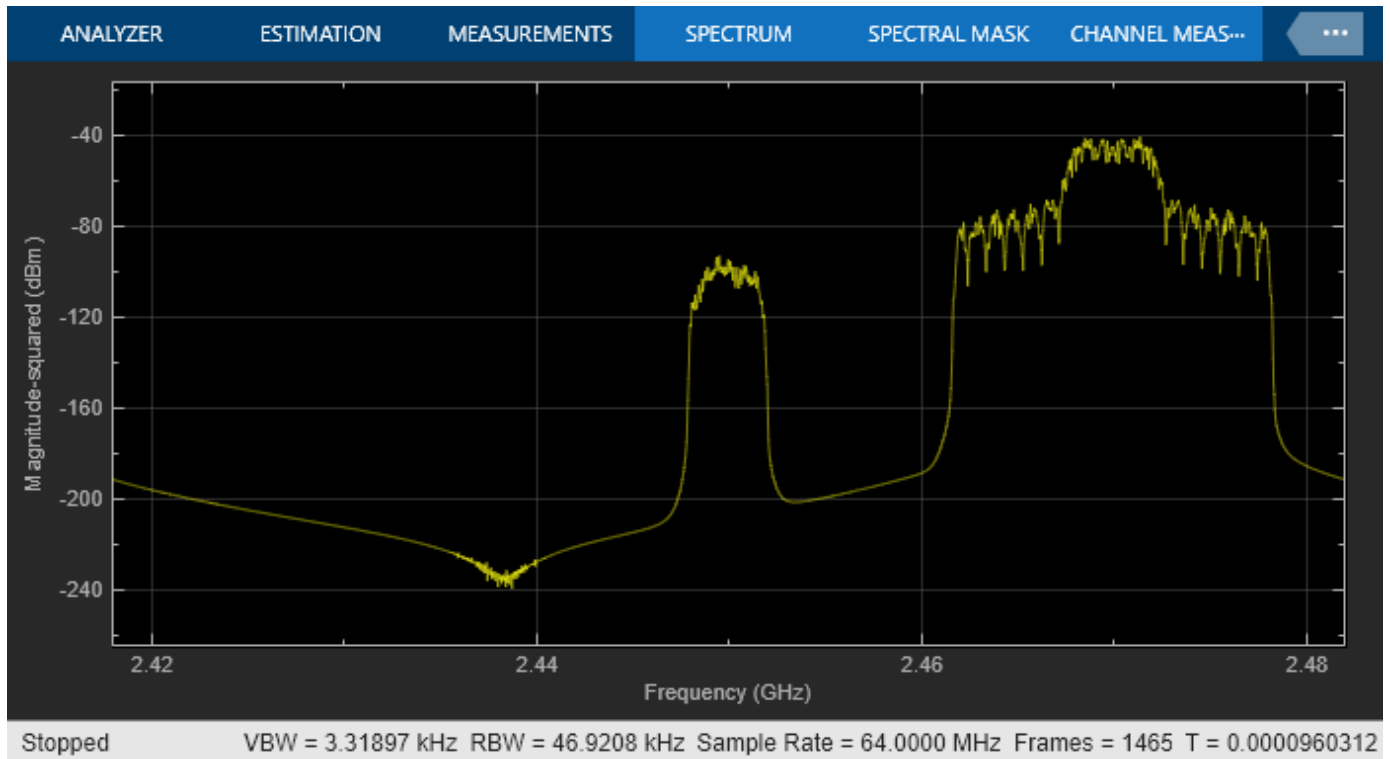
In the top RF receiver, the two signals entering the RF system are centered on different carriers. Note that the sample rate of the signal entering the top RF system is the same as defined in the RF Configuration block. In this case, enabling the automatic input interpolation filter in the RF Configuration block does not introduce any interpolation.

```
SpRxSepScopeConf.Visible = true;
```



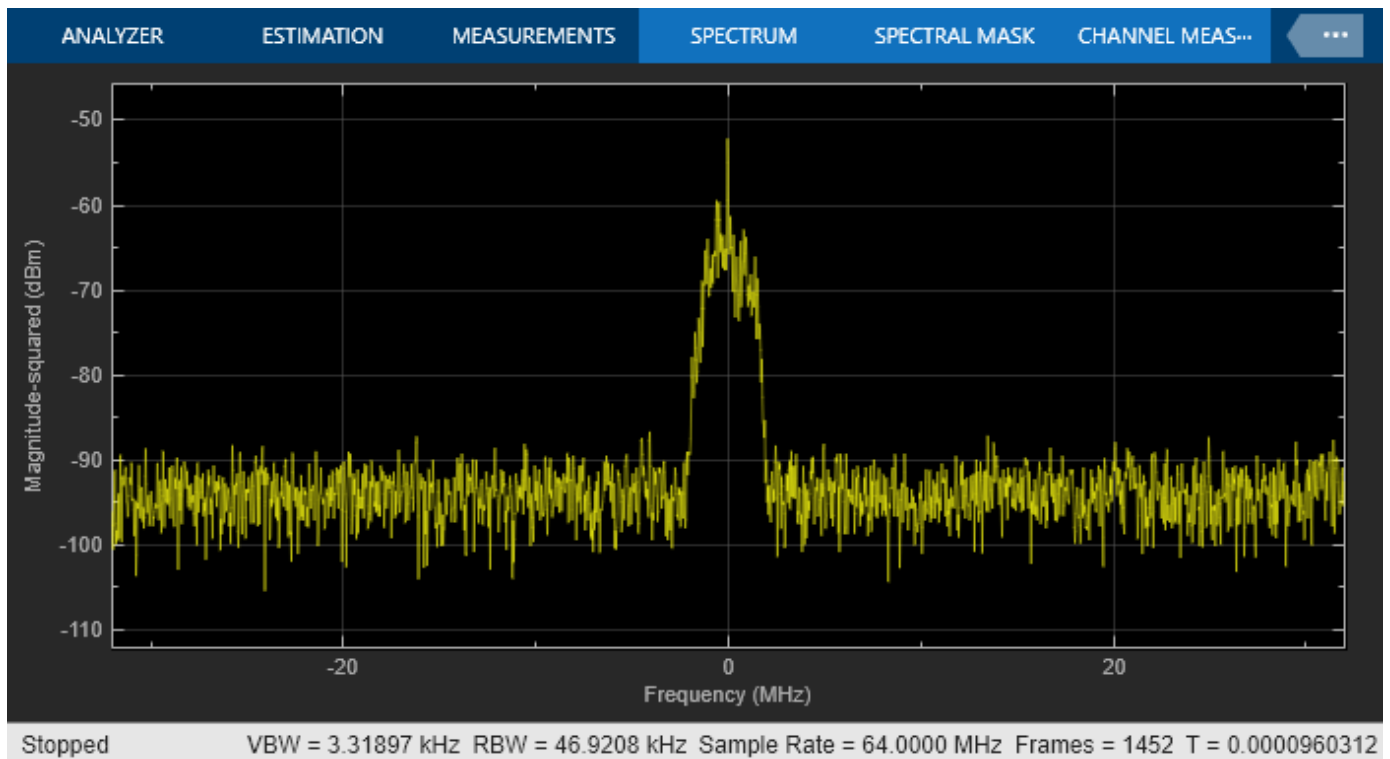
The last scenario discussed occurs when the two signals entering the RF system are placed on carriers that are relatively close to each other. Since the number of mixing harmonics required for simulation can be large in strongly nonlinear systems, it is recommended to combine the two signals onto one carrier when they are close by.

```
SpTXComScopeConf.Visible = true;
```



In the bottom receiver, the RF system is fed with the desired signals combined onto a single carrier signal. The combined signal is achieved by multiplying the interfering signal with a complex exponent to shift its operation frequency by 20MHz relative to the frequency of the desired signal. Note that the bandwidth needed to capture both signals when combined on a single carrier is larger than the bandwidth of each individual carrier signal. This is the reason for introducing the interpolation filter highlighted in green before combining the signals.

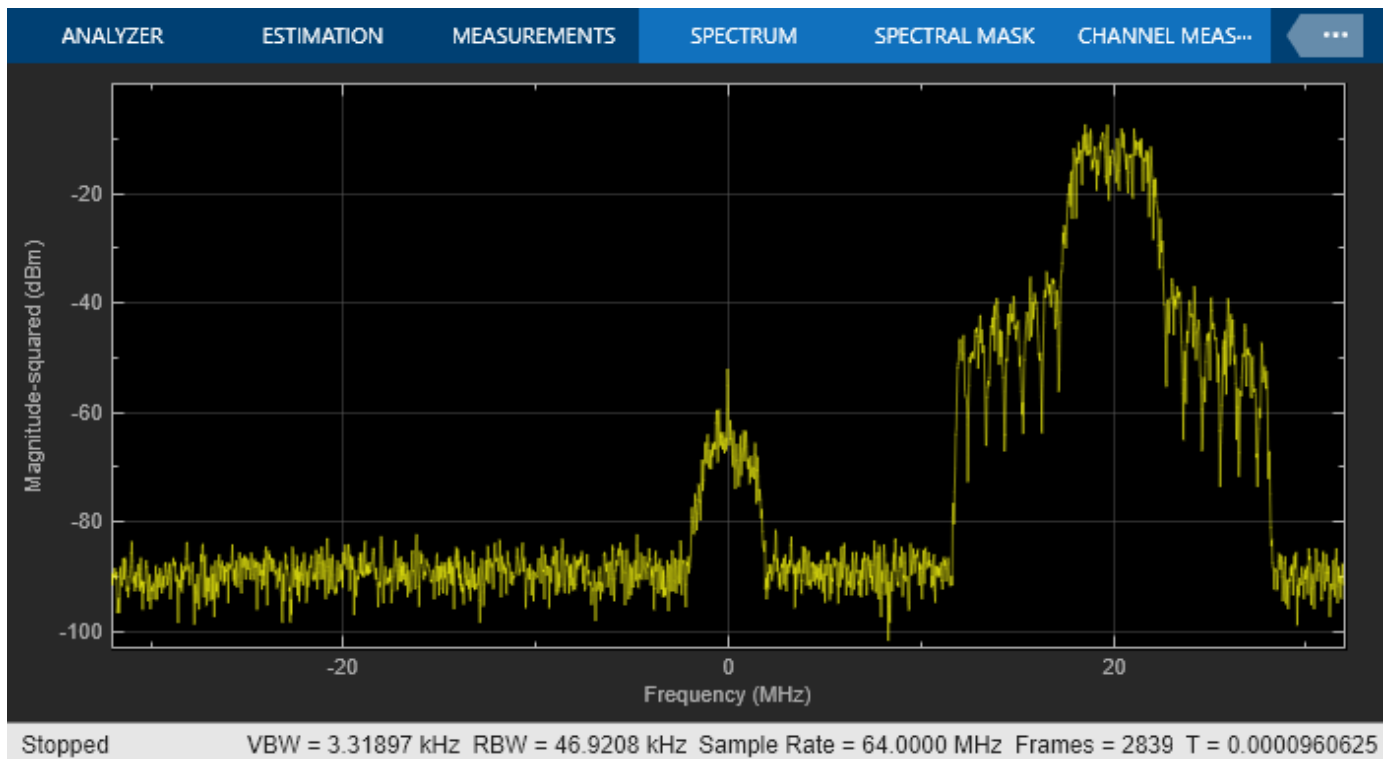
```
SpRxComScopeConf.Visible = true;
```



The results of the two RF systems (top and bottom) in the above model show excellent correspondence. The interferer signal is missing from the spectrum in the top RF system since the output port behaves as an ideal filter and only selects the real passband signal centered at DC. The interferer signal is missing from the spectrum in the bottom RF system since the IQ Demodulator includes a channel select filter. To see the effects of the interferer signal, turn off the filter by unchecking the 'Add Channel Select filter' checkbox in the IQ Demodulator block Mask Parameter dialog. The resulting spectrum is

```
set_param([model '/RF Blockset Direct Conversion Bottom/IQ Demodulator'], ...
    'AddCSFilters', 'off');
sim(model);

% Do not show other scopes and rescale Y axis:
SpTxSepScopeConf.Visible = false;
SpTxComScopeConf.Visible = false;
SpRxSepScopeConf.Visible = false;
SpRxComScopeConf.YLimits = [-103 0];
```



```
bdclose(model);  
clear model;
```

See Also

Amplifier | Configuration

Related Topics

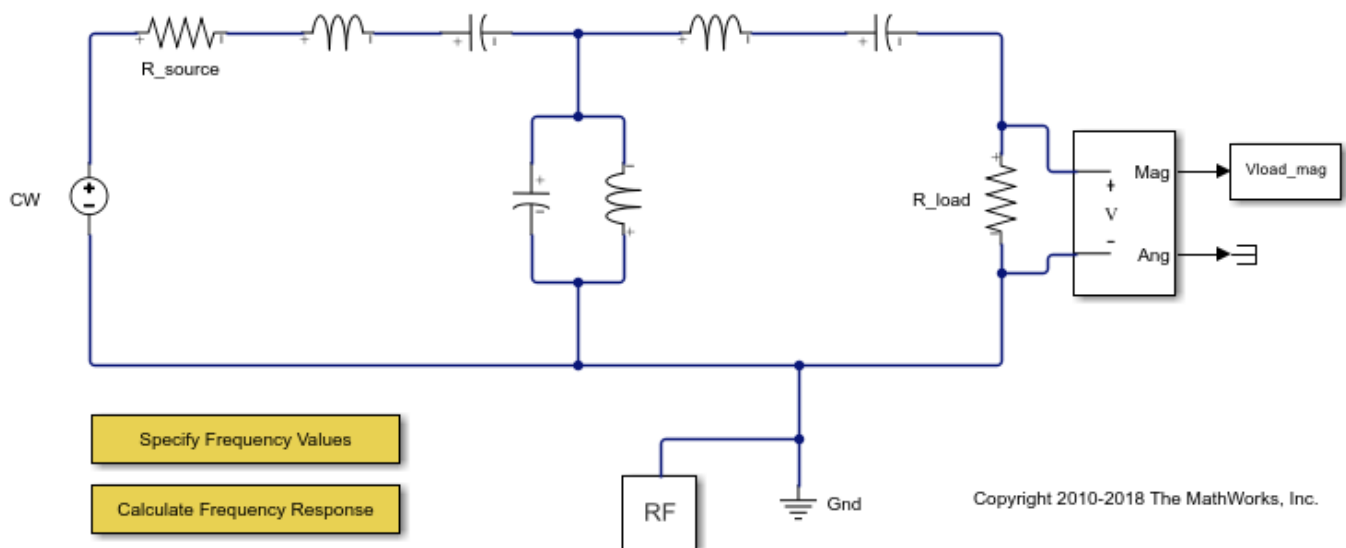
“Power Ports and Signal Power Measurement in RF Blockset” on page 8-11

Analysis of Frequency Response of RF System

This example uses a few techniques to calculate the steady-state frequency response for a filter-based RF system built from RF Blockset™ Circuit Envelope library blocks. The first technique performs static analysis (harmonic balance) on a circuit comprising of inductors and capacitors. The second technique does time domain simulation using a similar circuit built with the Filter library block. The third technique facilitates small-signal analysis to obtain the frequency response of a filtering system that exhibits nonlinearity at a given operation point. This example helps you validate a circuit envelope model using a static analysis in the frequency domain, a time domain simulation, and small signal analysis in cases where the system exhibits non-linearity.

Frequency Domain Analysis

```
model = 'simrfv2_ac_analysis';
open_system(model);
```

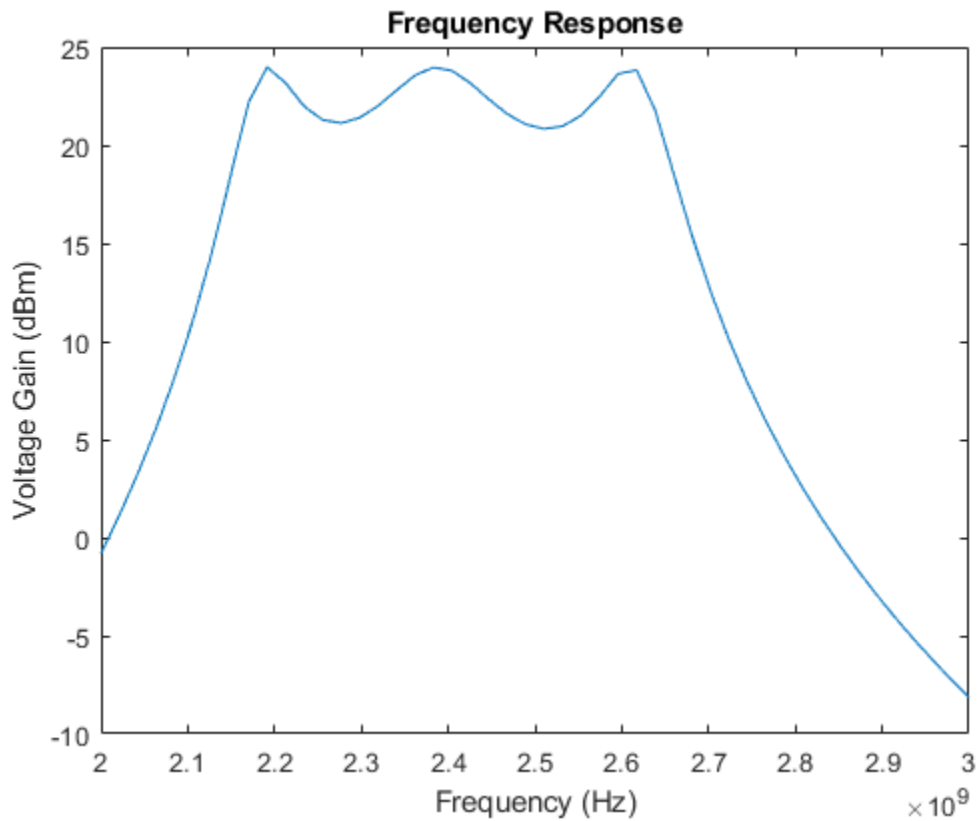


The system consists of:

- A Continuous Wave source and a series resistor to model a voltage source with internal source impedance.
- Inductor and Capacitor blocks configured to model a third-order Chebyshev filter with a center frequency of 2.4 GHz.
- An Output block configured as a voltage sensor to measure the voltage across a load resistor.
- A Configuration block, which sets up the circuit envelope simulation environment. As the system is linear, the harmonic balance analysis is done with a single simulation frequency and corresponds to an AC analysis.

- 1 Type `open_system('simrfv2_ac_analysis')` at the Command Window prompt.
- 2 Double-click the block labeled 'Specify Frequency Values' to provide a vector of frequencies.
- 3 Double-click the block labeled 'Calculate Frequency Response' to execute a script, `simrfv2_ac_analysis_callback`, that analyzes the model at the specified frequencies and plots the response.

```
simrfV2_ac_analysis_callback([model '/Subsystem'], 'OpenFcn');
```



To configure a model with circuit envelope library blocks for harmonic balance:

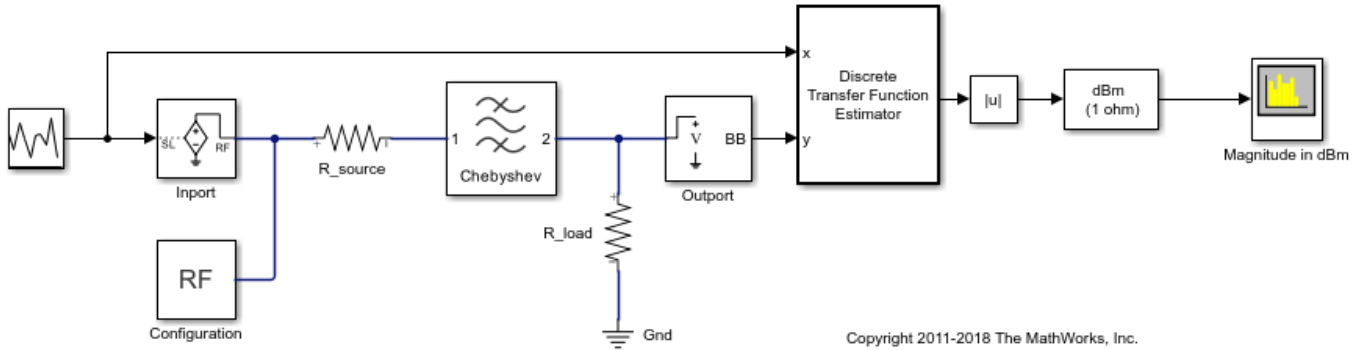
- In the Model Configuration parameters dialog box, set the **Stop time** parameter to zero.
- Use a Continuous Wave block to drive the system.
- Set the **Carrier frequencies** parameter in the Continuous Wave, Outport blocks, and the **Fundamental Tones** parameter in the Configuration block to the same vector of frequencies.

Close the open model

```
bdclose(model)
```

Time Domain Simulation

```
model = 'simrfV2_ac_analysis_tf';
open_system(model)
```



The system consists of:

- A Random source generator that outputs a continuous random signal.
- A Chebyshev filter constructed using the Filter library block and designed with a center frequency of 2.4 GHz and a bandwidth of 480 MHz.
- Discrete Transfer Function Estimator block to view the frequency domain output of a time domain simulation.
- Spectrum Analyzer to view the output.

View the filter designed parameters used in the Filter block mask.

Filter

Model an RF filter

Main Visualization

Design method: Chebyshev

Filter type: Bandpass

Implementation: LC Tee

Implement using filter order

Filter order: 3

Passband frequencies: [2.16 2.64] GHz

Passband attenuation (dB): $10 \cdot \log_{10}(2)$

Source impedance (Ohm): 50

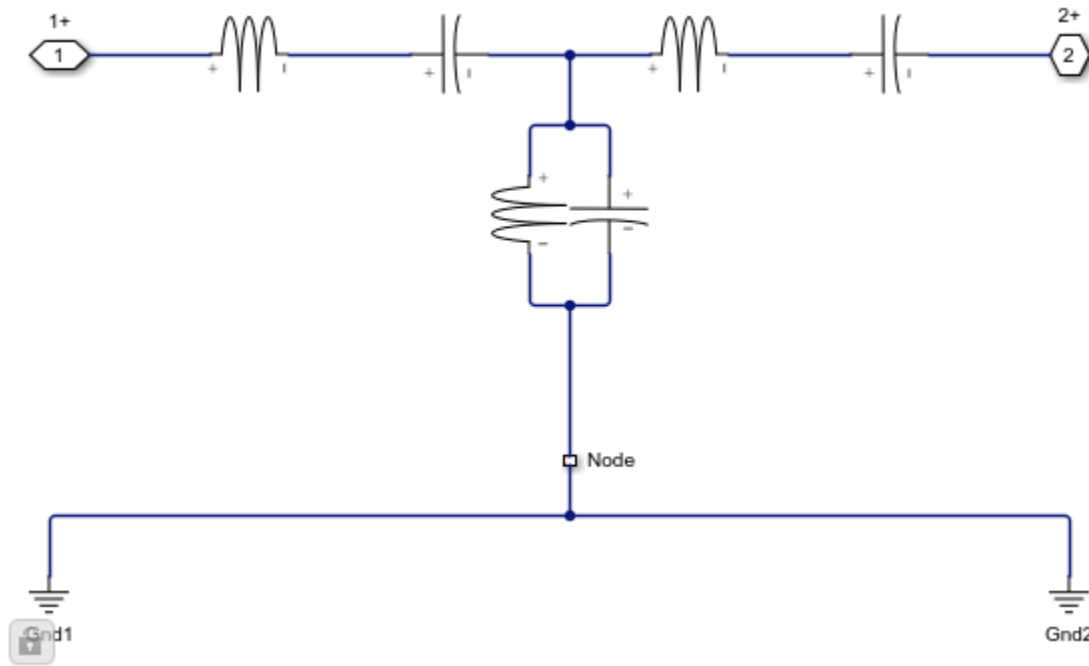
Load impedance (Ohm): 50

Ground and hide negative terminals

Export

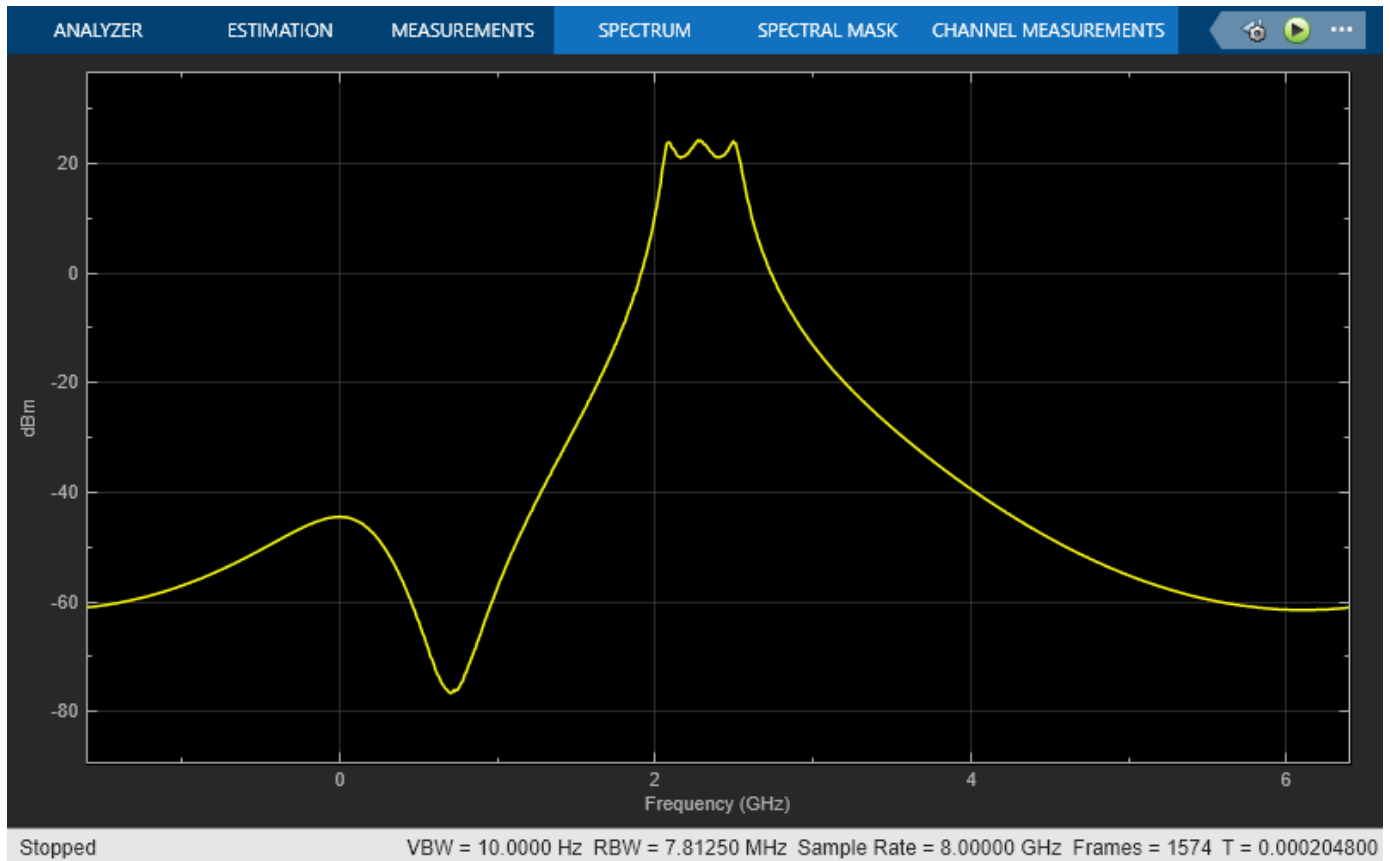
View the implemented filter under the Filter block mask.

```
open_system([model '/Filter'], 'force')
```

Simulate the transfer system model.

```
sim(model)
pause(5)
```

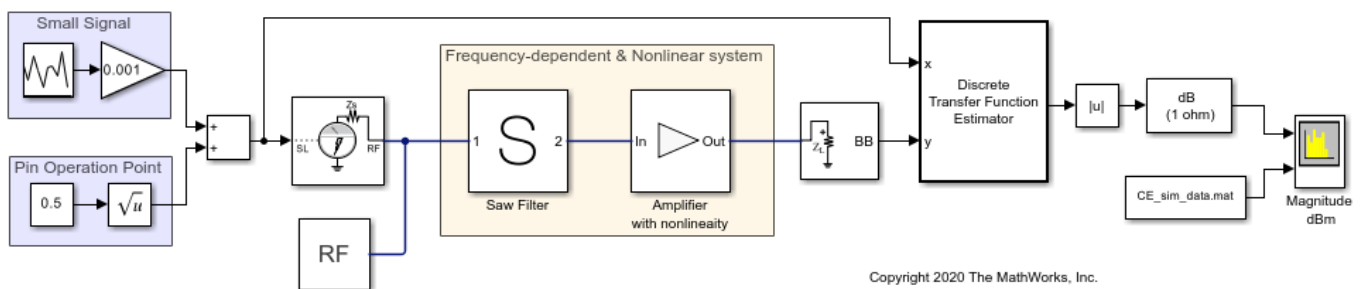


Compare the outputs of the first and second model.

```
bdclose(model)
```

Small Signal Analysis

```
model = 'simrfv2_ac_analysis_ss';
open_system(model)
```

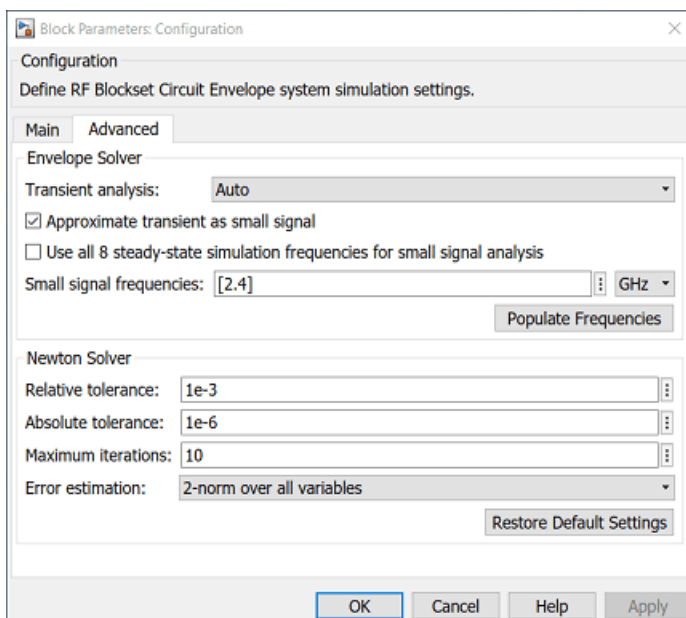


The system consists of:

- A Random source generator that outputs a continuous random signal that is subsequently attenuated to ensure small signal input.
- A constant source added to the random source to determine the non-linear operation point. Both signals are centered at 2.4 GHz.

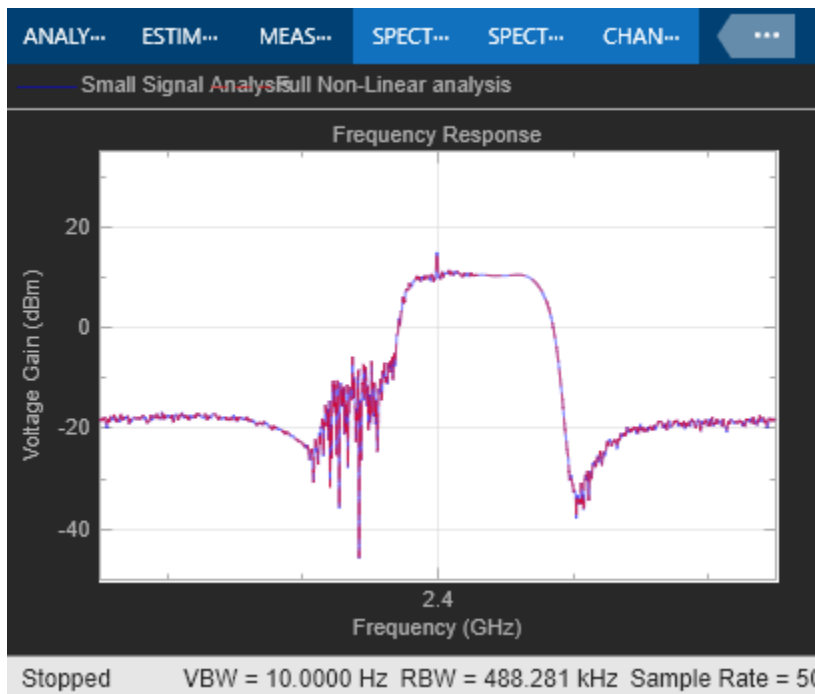
- An RF System comprising two elements; A saw filter constructed using the S-parameter library block with a center frequency of 2.45 GHz and a bandwidth of 112 MHz and an amplifier with 20dB of available power gain and non-linearity described by a 3rd-order intercept point of 30dBm.
- Discrete Transfer Function Estimator block to view the frequency domain output of a time domain simulation measured over the 2.4 GHz carrier.
- Spectrum Analyzer to view the output and compare it to saved output data.

Since the transient signal is small while the operation point is determined based on carrier-constant large signals, it is possible to use the transient small-signal approximation. In this approximation, non-linear interaction between transient signals is ignored, however the non-linear interaction between carrier-constant signals and its effect on the small signals is captured accurately. The small signal analysis is enabled in the advanced tab of the Configuration block mask.

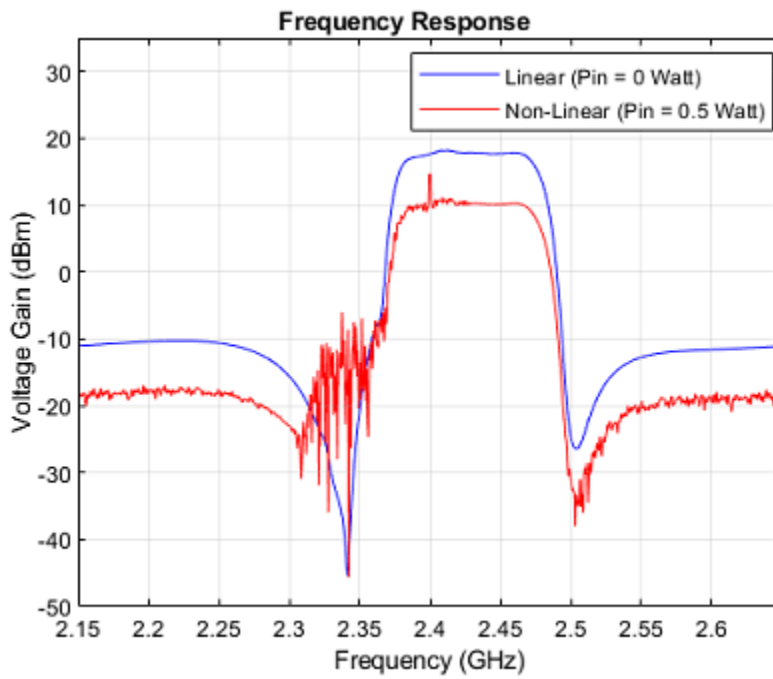


Using small signal analysis, a subset of the full set of carriers used for steady state solution can be chosen for transient simulation. In this example, only the 2.4 GHz is of interest for transient analysis. Reducing the number of simulated carriers, accelerates the simulation. In this case, the small signal simulation is more than 15 times faster than a full non-linear circuit-envelope based simulation. Comparing the small-signal simulation results with those of a full circuit envelope simulation loaded from a file, it is evident that the results are practically identical.

```
sim(model)
```



Decreasing the operation point power in the constant block from 0.5 Watt down to zero, the system becomes effectively linear. A comparison between the curves illustrates the effect of the non-linearity on the transfer function. These effects include a decrease in overall amplitude due to compression and a widening of the filter profile at the lower-frequency side. The widening can be explained as the result of the cubic term in the amplifier polynomial response folding the original RF frequency of 2.4 GHz back onto itself, but with a frequency response that is flipped around its central frequency since 2.4 GHz is reached by reflection from -2.4 GHz. Since the Saw filter is centered at 2.45GHz, the flipped frequency response is centered at 2.35GHz. Summing the linear and cube terms effects yields a widened profile.



```
bdclose(model)
```

References

- 1 Ludwig, Reinhold and Pavel Bretchko, *RF Circuit Design: Theory and Applications*. Prentice-Hall, 2000.
- 2 Mass A. Stephen, *Nonlinear Microwave and RF Circuits*. Artech House, 2003.

See Also

“Compare Time and Frequency Domain Simulation Options for S-parameters” on page 8-40

Compare Time and Frequency Domain Simulation Options for S-parameters

This example shows how to use two different options for modeling S-parameters with the RF Blockset™ Circuit Envelope library. The Time-domain (rationalfit) technique creates an analytical rational model that approximates the whole range of the data. This is a preferable technique when a good fit could be achieved with a small number of poles. When the data has a lot of details or high level of noise, this model becomes large and slow to simulate.

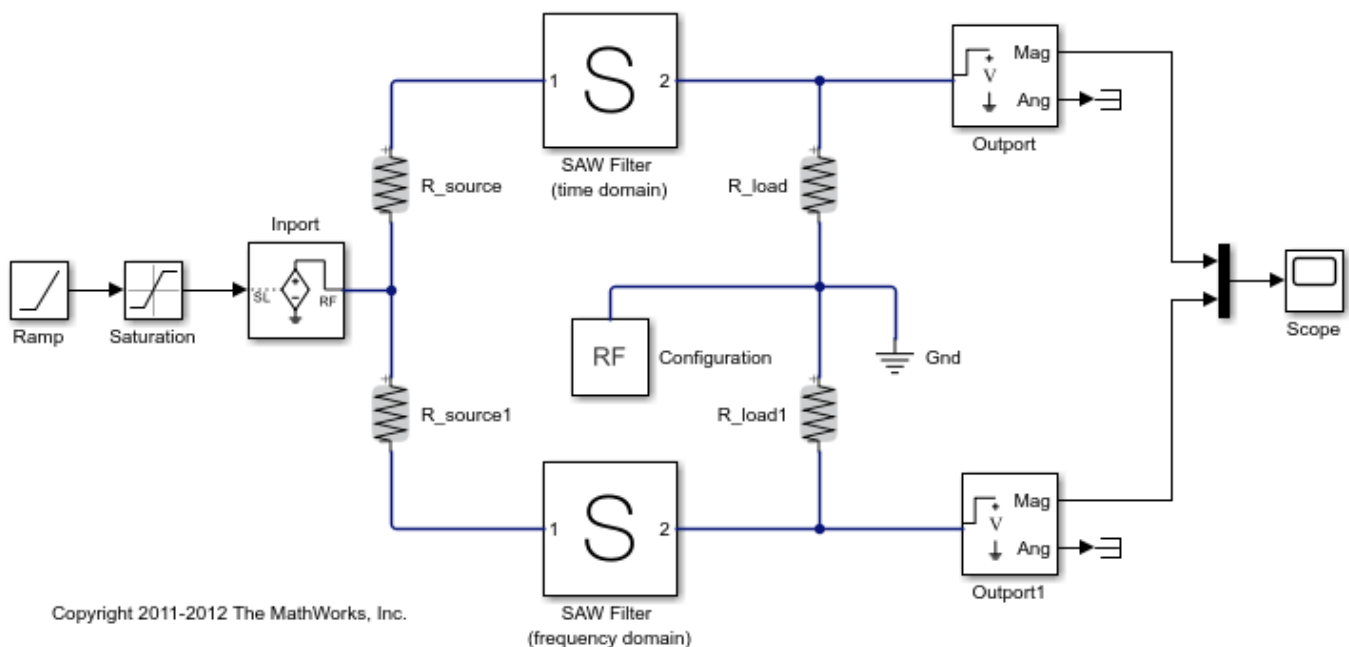
The frequency-domain technique is based on convolution, where the baseband impulse response depends on the simulation time step and the carrier frequency.

System Architecture

The system consists of:

- An input envelope signal modeled with Simulink blocks. The input signal is a ramp that goes from 0 to 1 in TF_RAMP_TIME; the initial value of TF_RAMP_TIME is set to 1e-6 s. The carrier frequency of the signal is TF_FREQ; the initial value of TF_FREQ is set to 2.4e9 Hz.
- Two SAW filters, modeled by two S-parameter blocks using the same data file, sawfilter.s2p. The block labeled SAW Filter (time domain) has its **Modeling options** parameter in the Modeling tab set to Time domain (rationalfit). The block labeled SAW Filter (frequency domain) has its **Modeling options** parameter in the Modeling tab set to Frequency domain and the **Automatically estimate impulse response duration** is checked.
- A Scope block that displays the outputs of the two S-parameter blocks.

```
model = 'simrfv2_sparam_t_vs_f';
open_system(model);
```

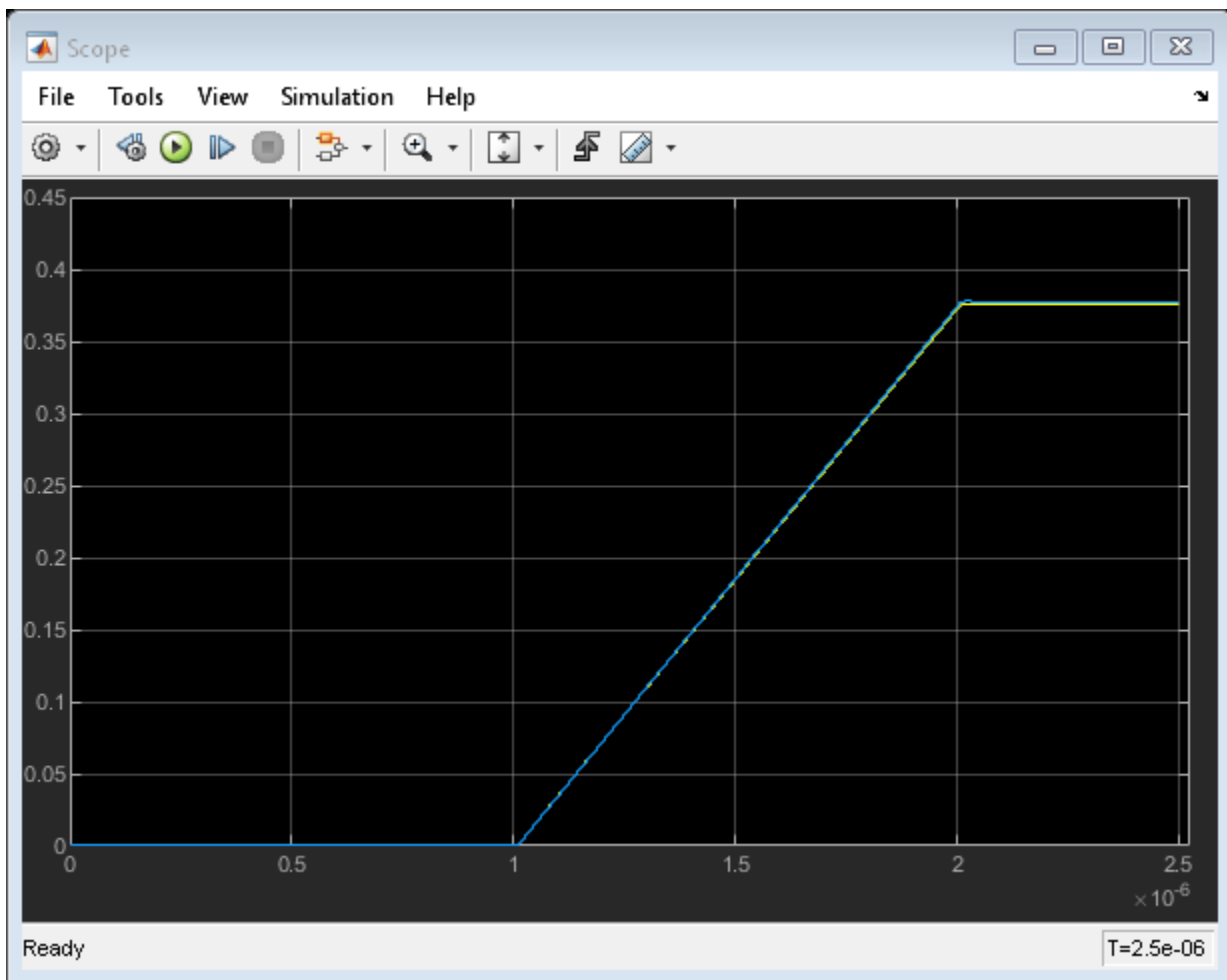


Run Simulation with the Default Settings

- 1 Type `open_system('simrfV2_sparam_t_vs_f')` at the Command Window prompt.
- 2 Select **Simulation > Run**.

The outputs from both methods are very close to each other. The frequency-domain model (purple curve) captures the transfer function (steady-state value) a bit better.

```
scope = [model '/Scope'];
open_system(scope);
set_param(scope, 'YMax', '0.45');
set_param(scope, 'YMin', '0');
set_param(scope, 'TimeRange', num2str(1.01*TF_END_TIME));
sim(model);
```



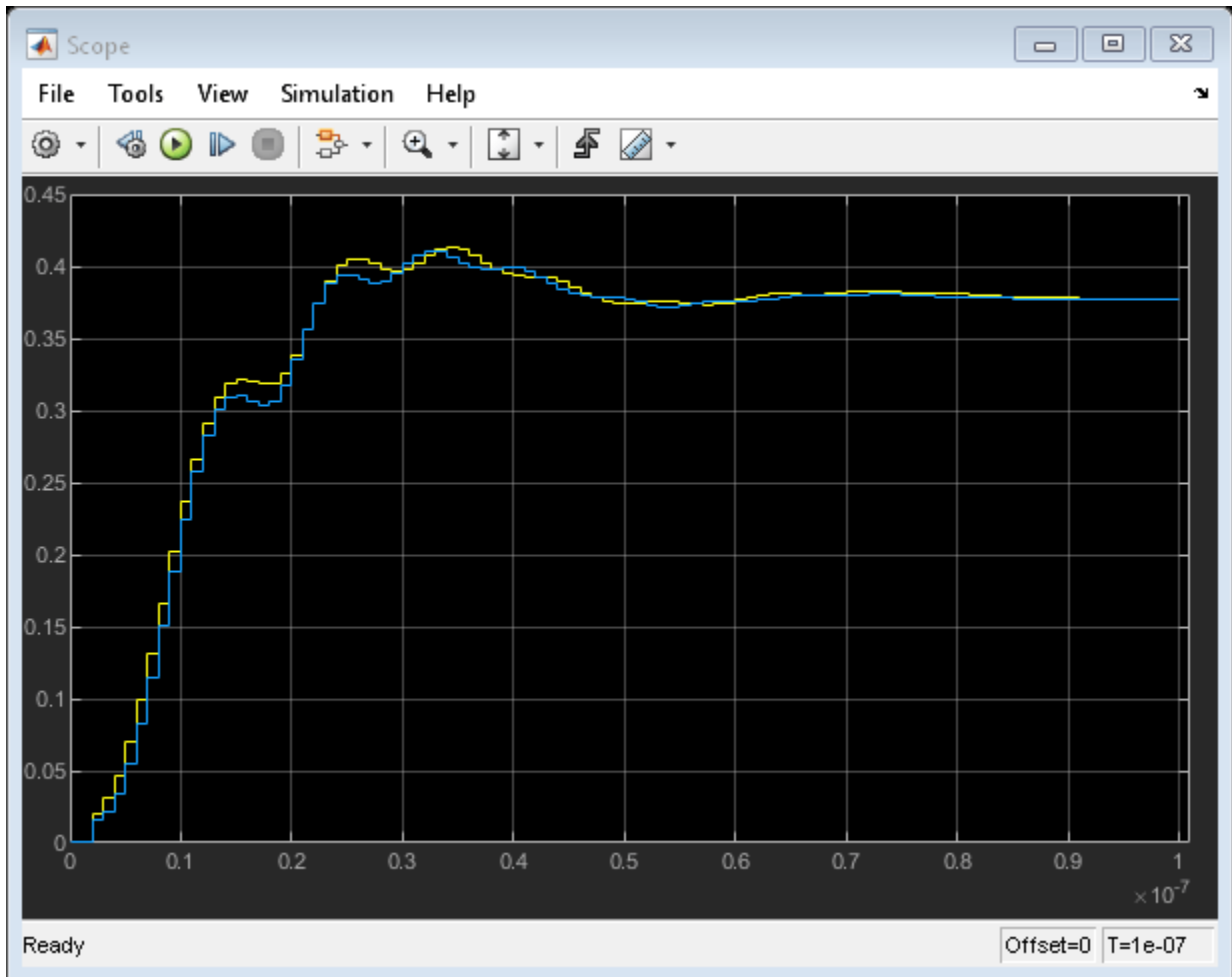
Run the Simulation with the Very Steep Ramp

In the previous simulation, the rise time of the envelope $TF_RAMP_TIME = 1e-6$ was many orders of magnitude greater than the period of the carrier signal $T = 1/TF_FREQ = 4.1667e-10$. In other words, the envelope was much slower than the carrier. As the ramp time approaches the period of the carrier, the corresponding time effects are better captured by the time-domain model (yellow curve).

To continue the example:

- 1 Type `TF_RAMP_TIME = 1e-9; TF_END_TIME = 1e-7;` at the Command Window prompt.
- 2 Select **Simulation > Run**.

```
TF_RAMP_TIME = 1e-9;
TF_END_TIME = 1e-7;
set_param(scope, 'TimeRange', num2str(1.01*TF_END_TIME));
sim(model);
open_system(scope);
```



The result of the frequency-domain simulation can be improved by decreasing the time step of the simulation and manually setting the impulse duration time.

To continue the example:

- 1 Type `TF_STEP = 5e-10;` at the Command Window prompt.
- 2 Uncheck **Automatically estimate impulse response duration** in the modeling pane of Saw filter (frequency domain) block and specify the Impulse Response Duration as `1e-7`.
- 3 Select **Simulation > Run**.

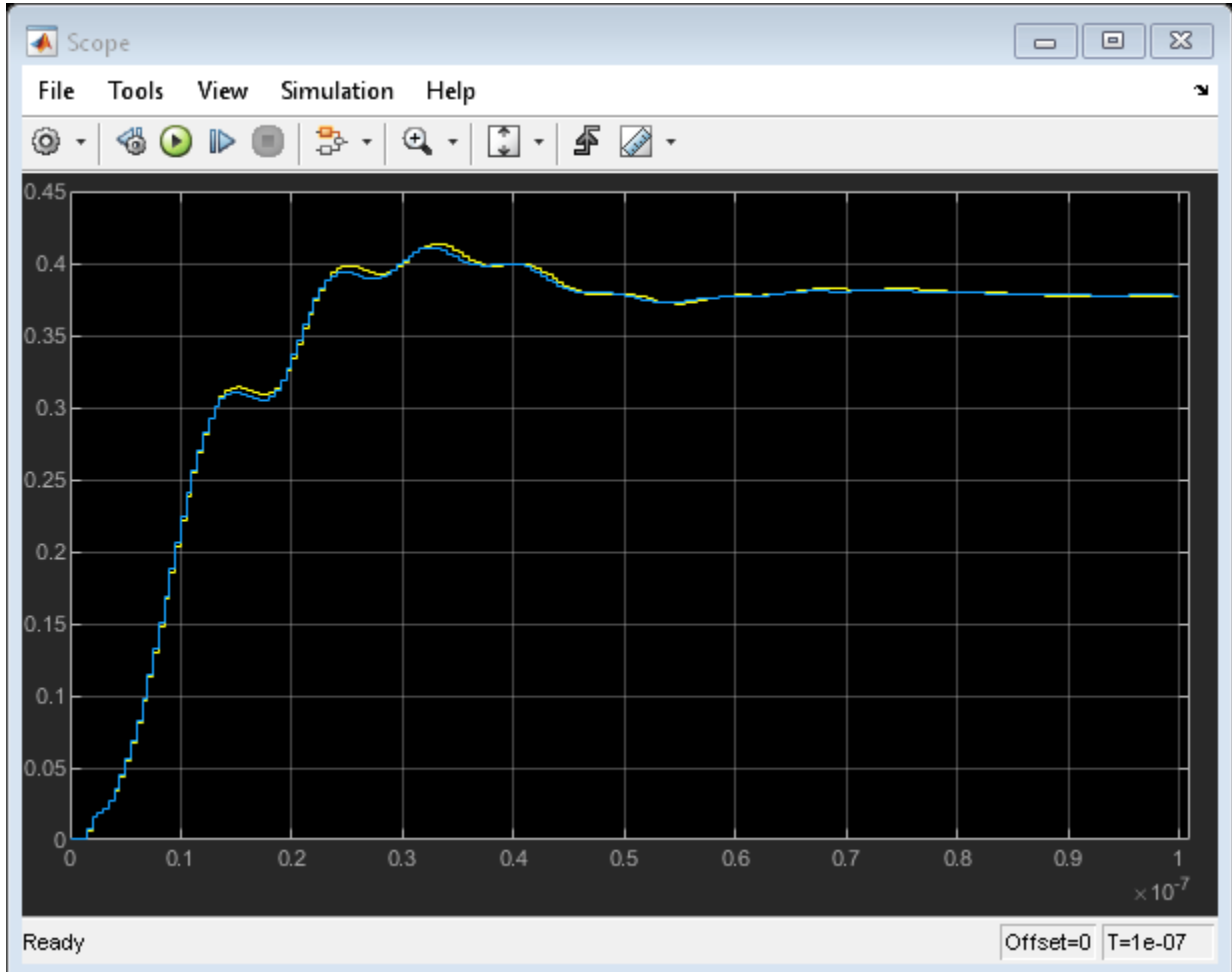
```
TF_STEP = 5e-10;
sparam_freq = [model '/SAW Filter (frequency domain)'];
```



```

set_param(sparam_freq, 'AutoImpulseLength', 'off');
set_param(sparam_freq, 'ImpulseLength', '1e-7');
sim(model);
open_system(scope);

```



Run Simulation with Different Frequency

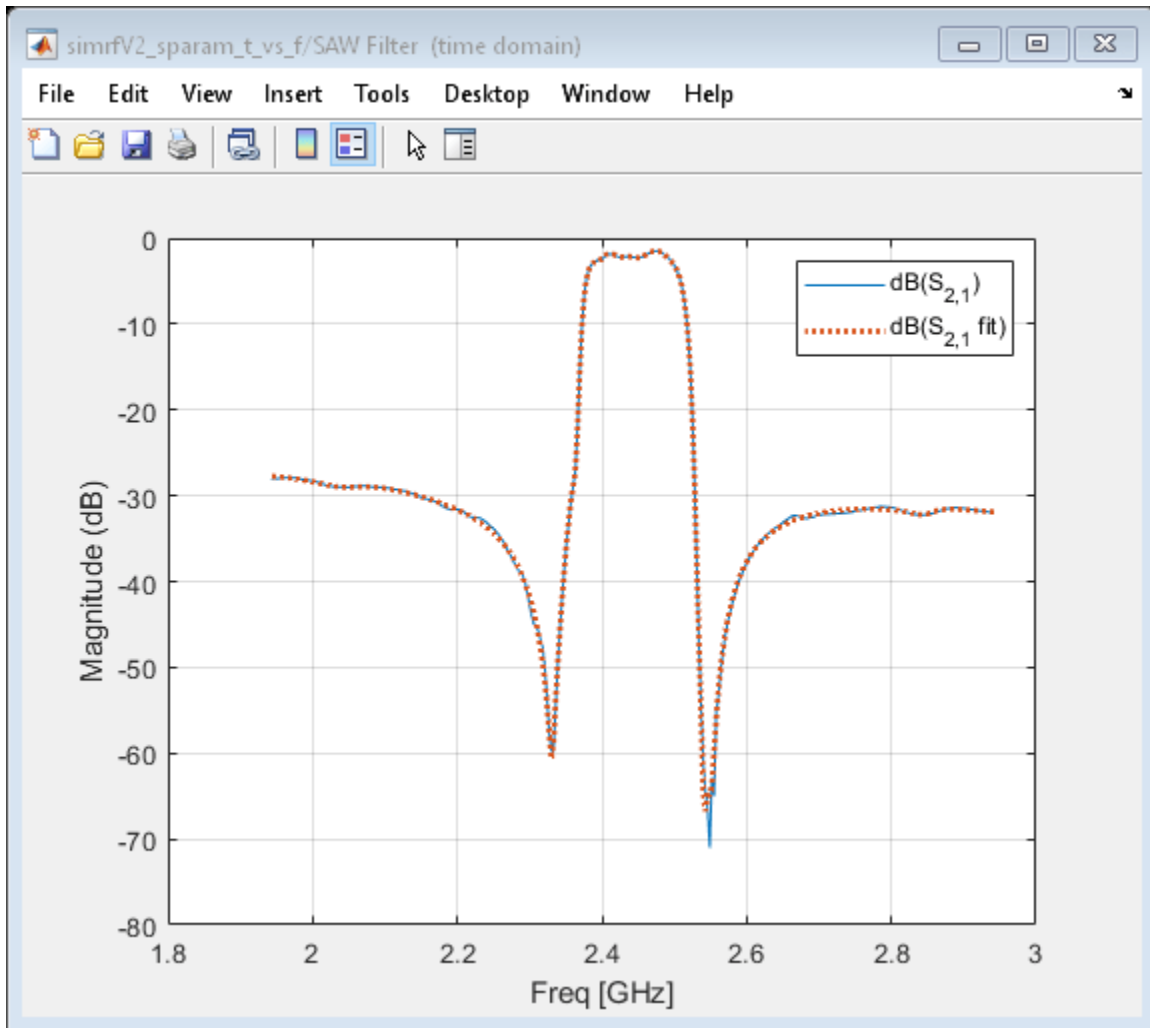
Rational-function approximation is not exact. To see the approximation error, double-click the "SAW Filter (time domain)" block. Information about the approximation appears under "Rational fitting results" in the bottom of the dialog 'Modeling' pane.

```
open_system([model sprintf('/SAW Filter (time domain)')]);
```

For more details, select 'Visualization' panel, and click the 'Plot' button.

The rationalfit algorithm (dotted curve) does a very good job for the most of the frequencies. However, sometimes it does not capture the sharp changes of S-parameter data.

```
simrfV2_click_dialog_button('Block Parameters: SAW Filter (time domain)', 'PlotButton');
```



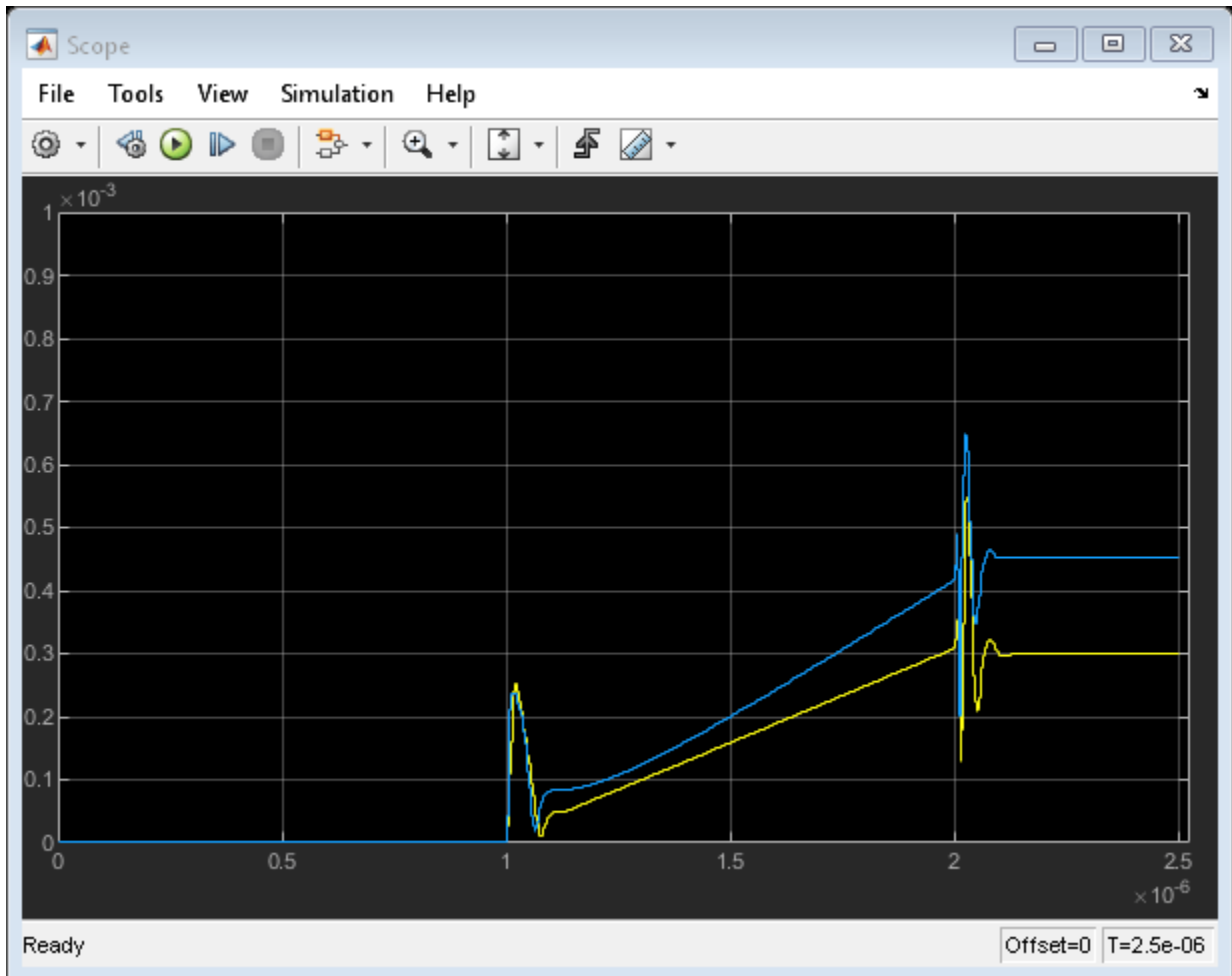
Conversely, the frequency-domain method exactly reproduces the steady-state behavior at all carrier frequencies (by definition). Running the simulation for $TF_FREQ = 2.54e9$ produces drastically different results between the two S-parameter methods.

To continue the example:

- 1 Type `TF_FREQ = 2.54e9; TF_RAMP_TIME = 1e-6; TF_STEP = 3e-9; TF_END_TIME = 2.5e-6;` at the Command Window prompt.
- 2 Select **Simulation > Run**.

In this case, the frequency-domain model provides a better approximation of the original data.

```
TF_STEP = 3e-9;
TF_RAMP_TIME = 1e-6;
TF_FREQ = 2.54e9;
TF_END_TIME = 2.5e-6;
set_param(scope, 'YMax', '1e-3');
set_param(scope, 'TimeRange', num2str(1.01*TF_END_TIME));
sim(model);
open_system(scope);
```



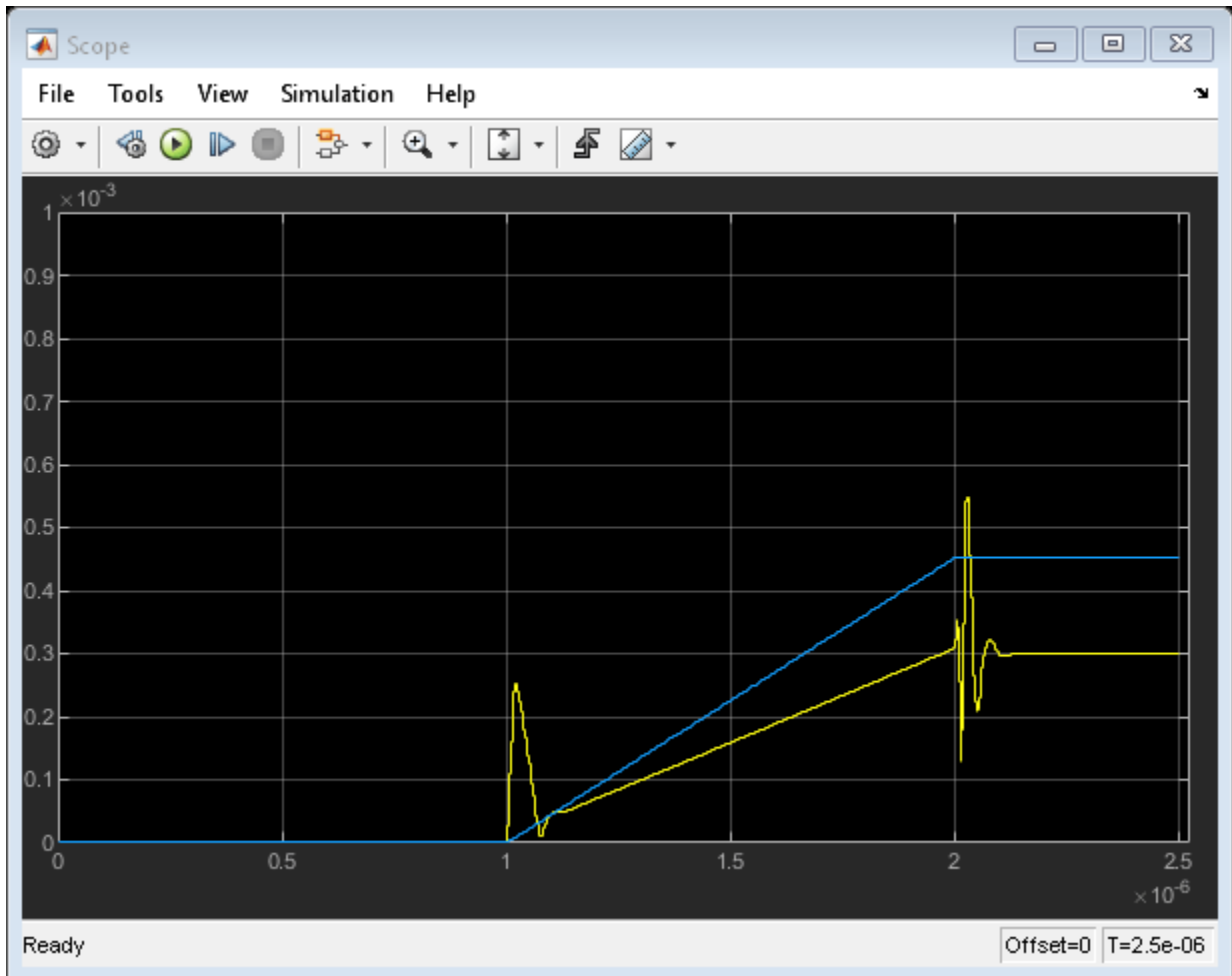
Run Simulation with Impulse Duration Set to Zero.

There is a special case that could be very helpful in practice. When the "Impulse Response Duration" of the s-parameters block is set to zero, the history of the input is no longer taken into consideration. Still, the model captures the transfer function (steady-state value) correctly. This is a fast and reliable way to model the ideal devices when the transient effects could be ignored.

To continue the example:

- 1 Specify the Impulse Response Duration of Saw filter (frequency domain) block as 0.
- 2 Select **Simulation > Run**.

```
set_param(sparam_freq, 'ImpulseLength', '0');
sim(model);
open_system(scope);
```



Conclusion

In most practical RF systems, time- and frequency-domain techniques give similar answers. The time-domain method better captures the time-domain effects of the fast-changing envelopes, but relies on a rationalfit approximation of the original data. The frequency-domain method is sensitive to the simulation time step; this option is recommended when the time-domain model does not provide a good fit.

```
close(gcf);
bdclose(model);
clear model scope;
```

See Also

S-Parameters | Configuration | Inport | Output

Related Topics

“Analysis of Frequency Response of RF System” on page 8-32 | “Transmission Lines, Delay-Based and Lumped Models” on page 8-47

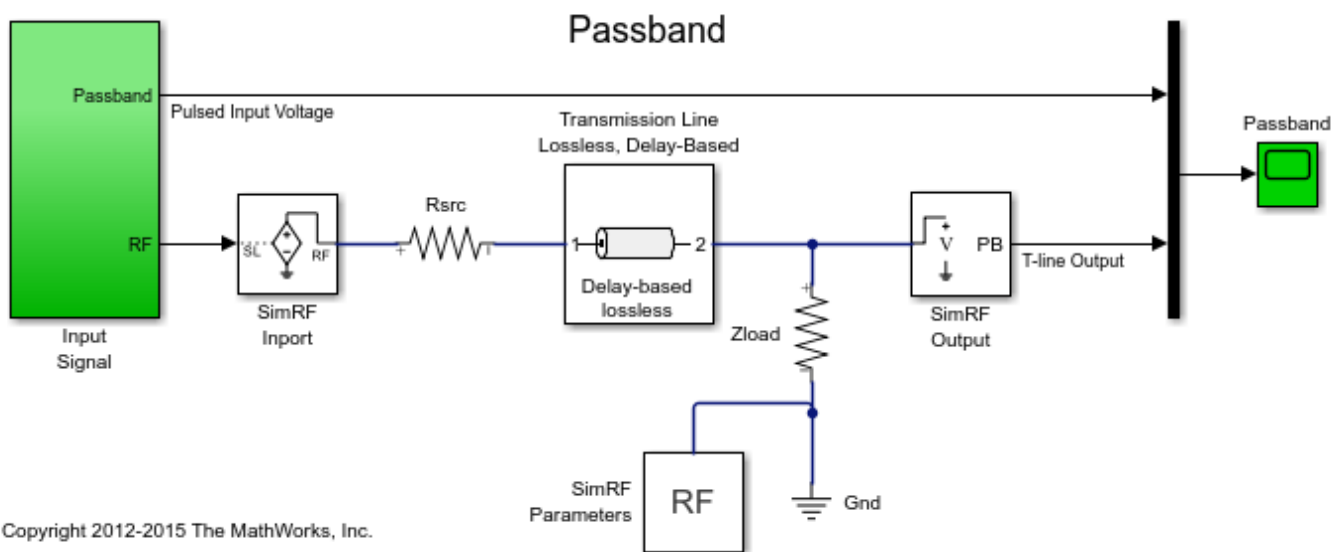
Transmission Lines, Delay-Based and Lumped Models

This example shows how to simulate delay-based and lumped-element Transmission Line using blocks in the RF Blockset™ Circuit Envelope library. The example is sequenced to examine circuit envelope and passband differences, delay-based lossy transmission line sectioning, and lumped element implementation of delay.

System Architecture for Lossless Delay-Based Transmission Line

In this section, two RF Blockset™ models, `simrf_xline_pb` and `simrf_xline_ce`, illustrate lossless delay-based transmission line effects and the computational benefit of circuit envelope techniques.

```
model_pb = 'simrf_xline_pb';
model_ce = 'simrf_xline_ce';
load_system(model_ce)
open_system(model_pb)
```

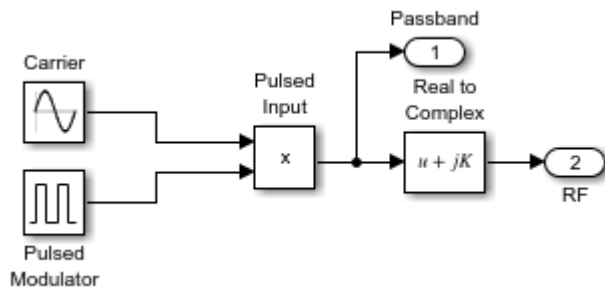


The model, `simrf_xline_pb`, represents a passband signal as:

$$I(t) \cos 2\pi f_c t - Q(t) \sin 2\pi f_c t$$

The input is a pulse-modulated sinusoidal passband signal. For this particular case, $I(t)$ equals zero, and $Q(t)$ is the pulse modulation. The carrier frequencies are set to zero in the RF Blockset Inport and Output blocks.

```
open_system([model_pb '/Input Signal']);
```

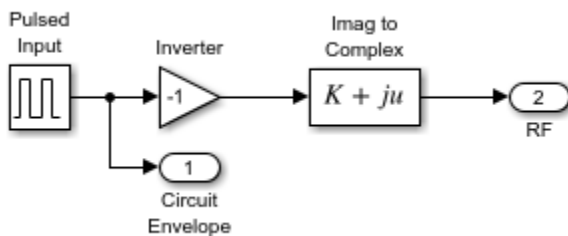


The circuit envelope model, `simrf_xline_ce`, represents an envelope signal as:

$$I(t) + jQ(t)$$

Again, $I(t)$ equals zero, and $Q(t)$ is the pulse modulation, but the carrier signal is not specified as part of the input signal. To model the carrier, the Carrier Frequencies parameter is set to f_c in the RF Blockset Inport and Outport blocks.

```
open_system([model_ce '/Input Signal']);
```



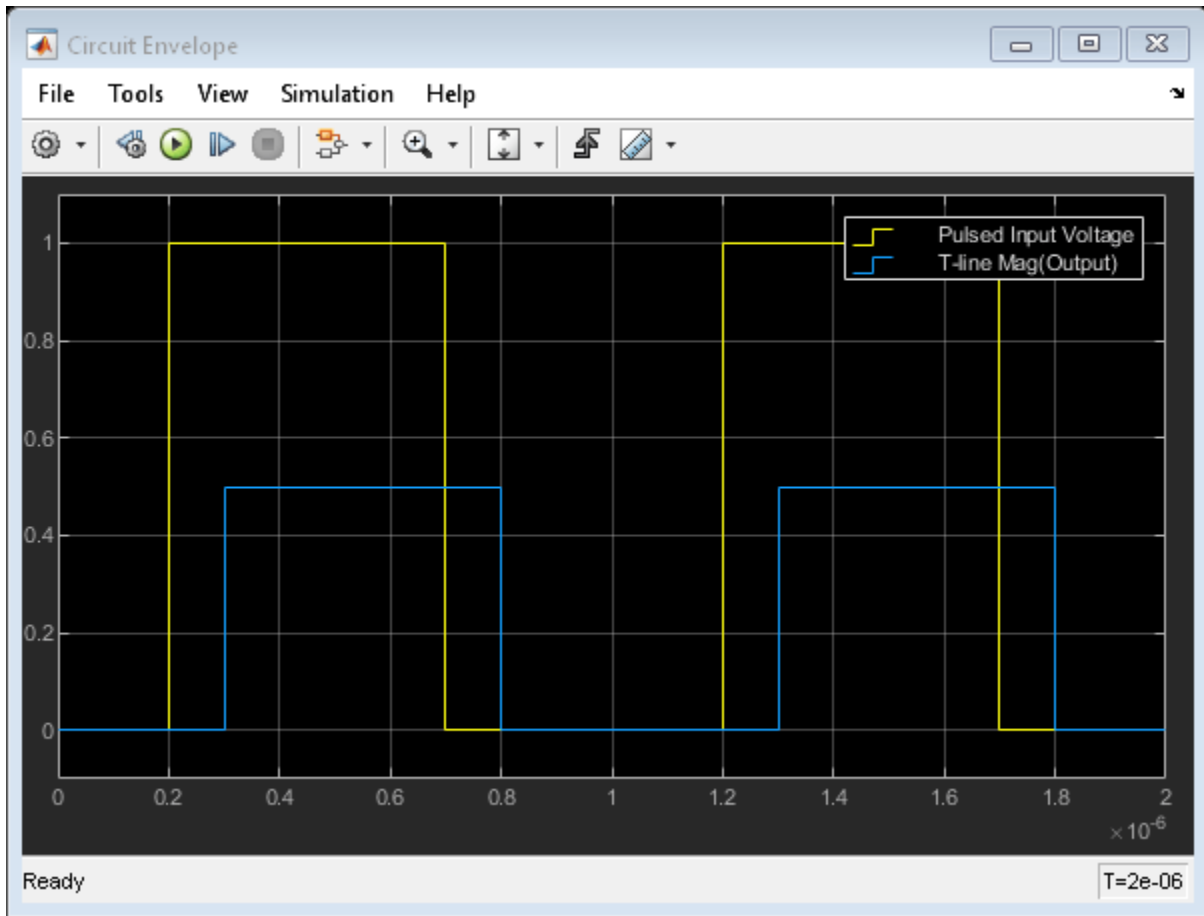
Removal of the explicit sinusoidal carrier in the circuit-envelope model allows the simulation to reduce time-steps relative to the passband model.

Running the Lossless Delay-Based Transmission Line

- 1 Type `open_system('simrf_xline_pb')` or `open_system('simrf_xline_ce')` at the Command Window prompt.
- 2 Select **Simulation > Run**.

After simulating, the transmission delay is observable in a plot of input and output signals.

```
open_system([model_ce '/Circuit Envelope']);
sim(model_ce);
```

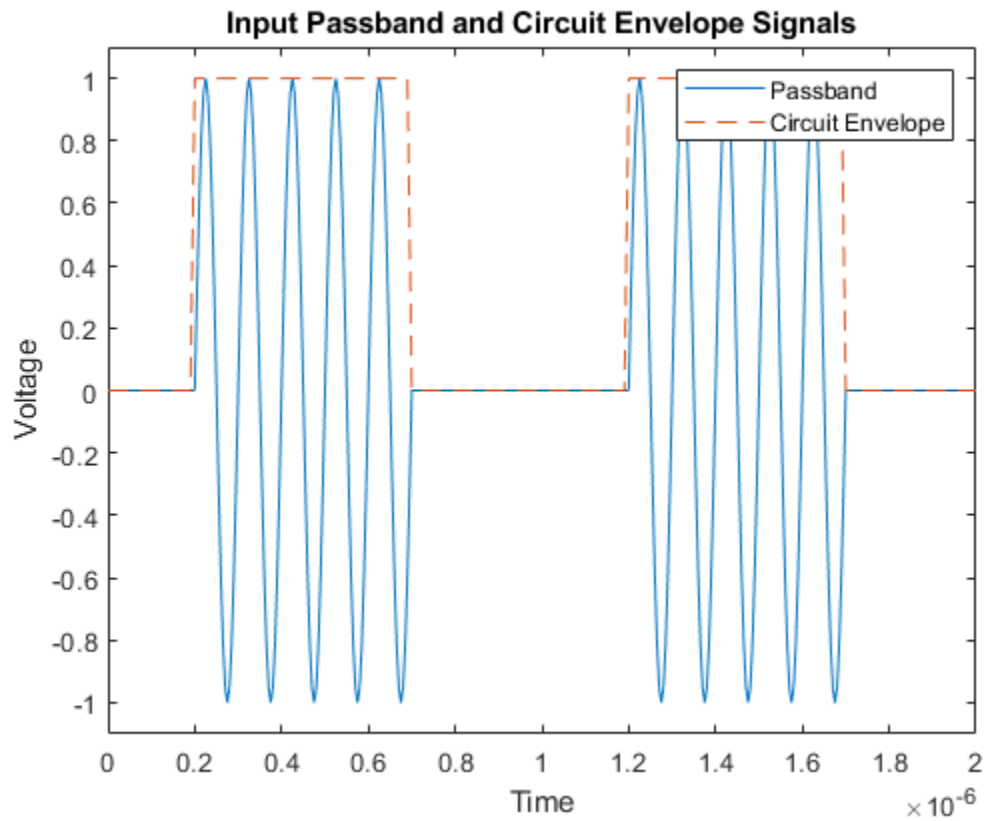


The carriers in modulated waveforms appear in passband signals, but only the modulation envelopes appear in circuit-envelope signals. Passband signals can be reconstructed from circuit envelope signals as:

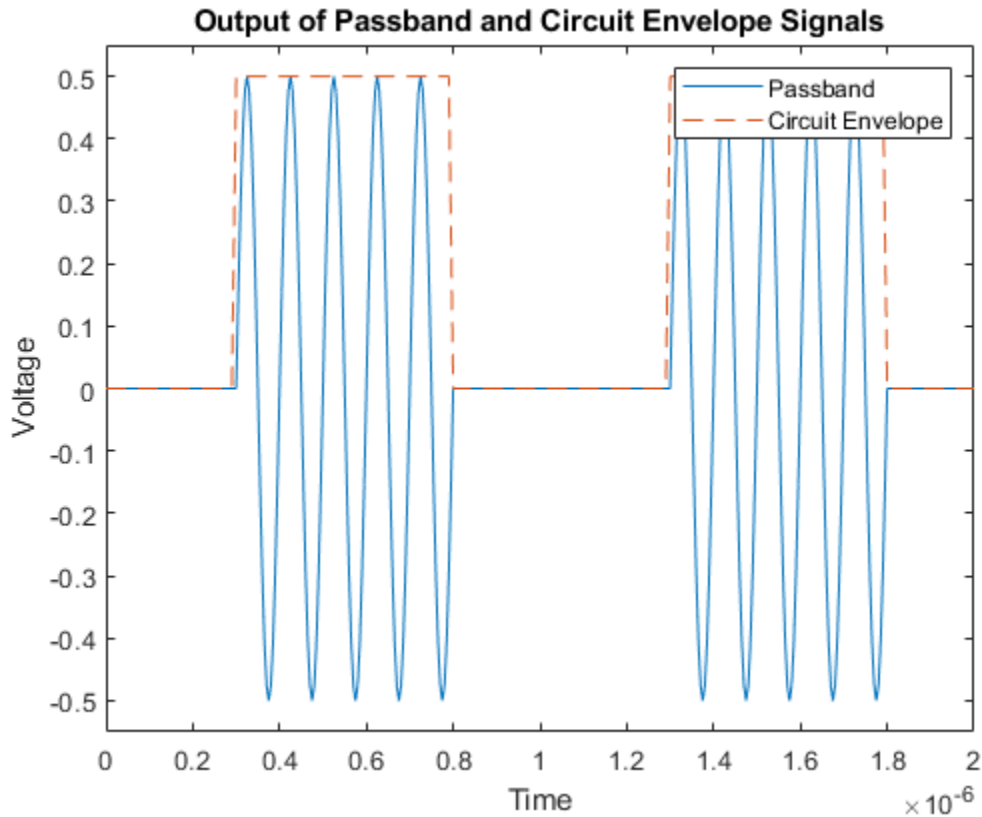
$$\text{Re}((I(t) + jQ(t))e^{j2\pi f_c t})$$

However, reconstruction of the passband signal this way requires additional time steps for the carrier.

```
sim(model_pb);
hline = plot(SPB_Data(:,1),SPB_Data(:,2),SCE_Data(:,1),SCE_Data(:,2),'--');
legend('Passband', 'Circuit Envelope')
title('Input Passband and Circuit Envelope Signals')
xlabel('Time')
ylabel('Voltage')
ylim([-1.1 1.1])
```



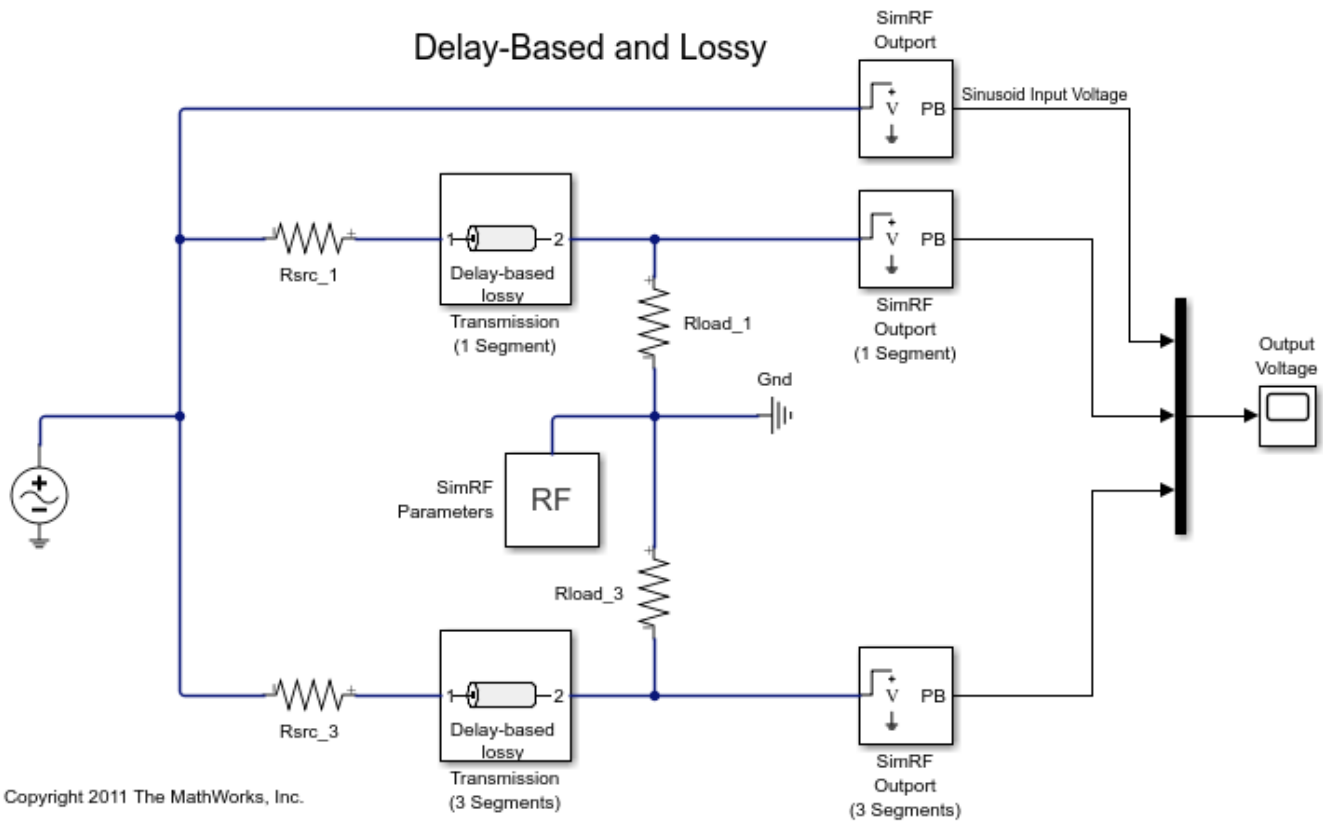
```
haxis = get(hline(1), 'Parent');  
plot(haxis, SPB_Data(:,1), SPB_Data(:,3), SCE_Data(:,1), SCE_Data(:,3), '--')  
legend('Passband', 'Circuit Envelope')  
title('Output of Passband and Circuit Envelope Signals')  
xlabel('Time')  
ylabel('Voltage')  
ylim([-0.55 0.55])
```

Partitioning Delay-Based Lossy Transmission Lines

A conventional method for modeling distributed lossy transmission lines employs N two-port segments in cascade. Each segment consists of an ideal lossless delay line and resistance, where the segment delay equals the total line delay divided by N and the segment resistance equals the total line resistance divided by N . As the number of segments increases, the lumped model will more accurately represent the distributed system. This methodology requires a compromise between simulation time and model accuracy for increasing N . In RF Blockset, the **Number of segments**, the **Resistance per unit length** and the **Line length** are specified as dialog box parameters in the transmission line block.

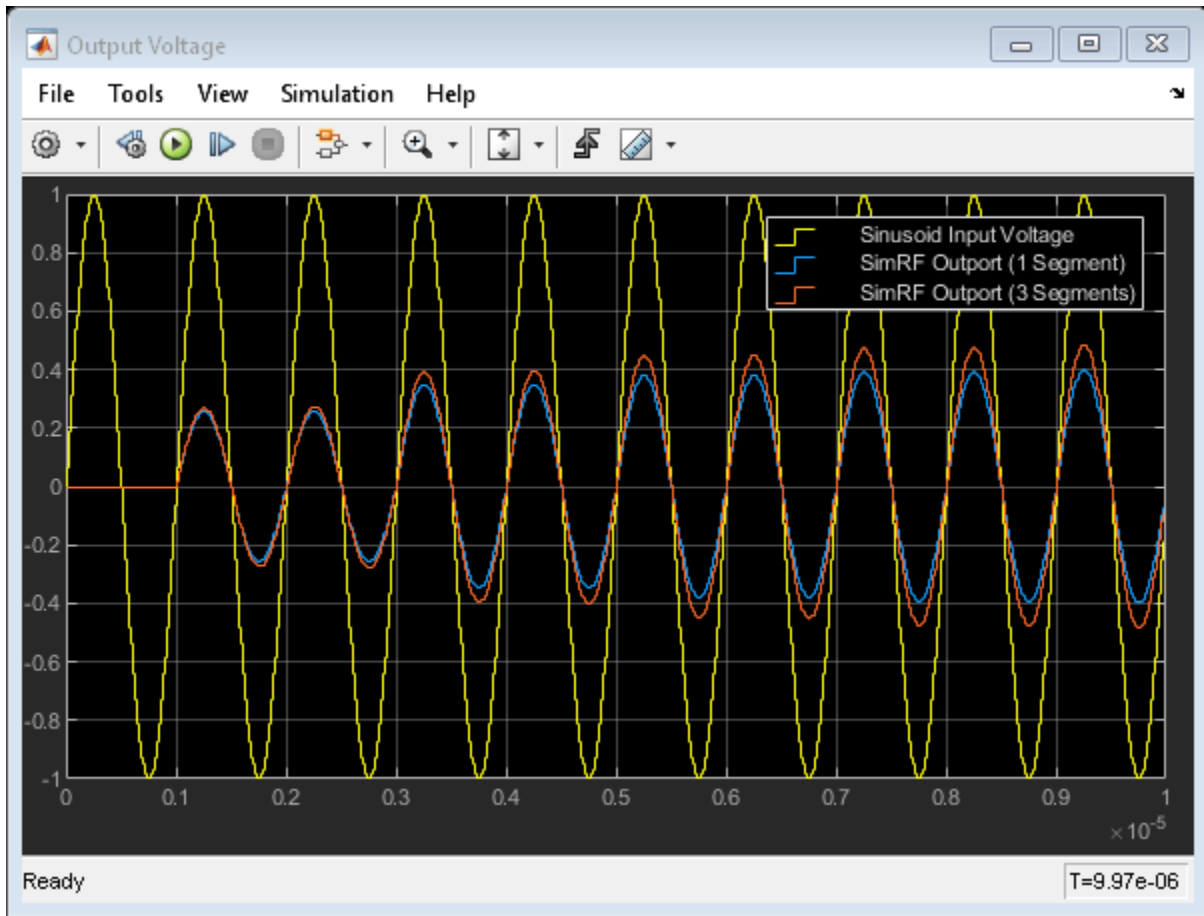
```
model_seg = 'simrf_xline_seg';
open_system(model_seg)
```



System Architecture for Lossy Delay-Based Transmission Line

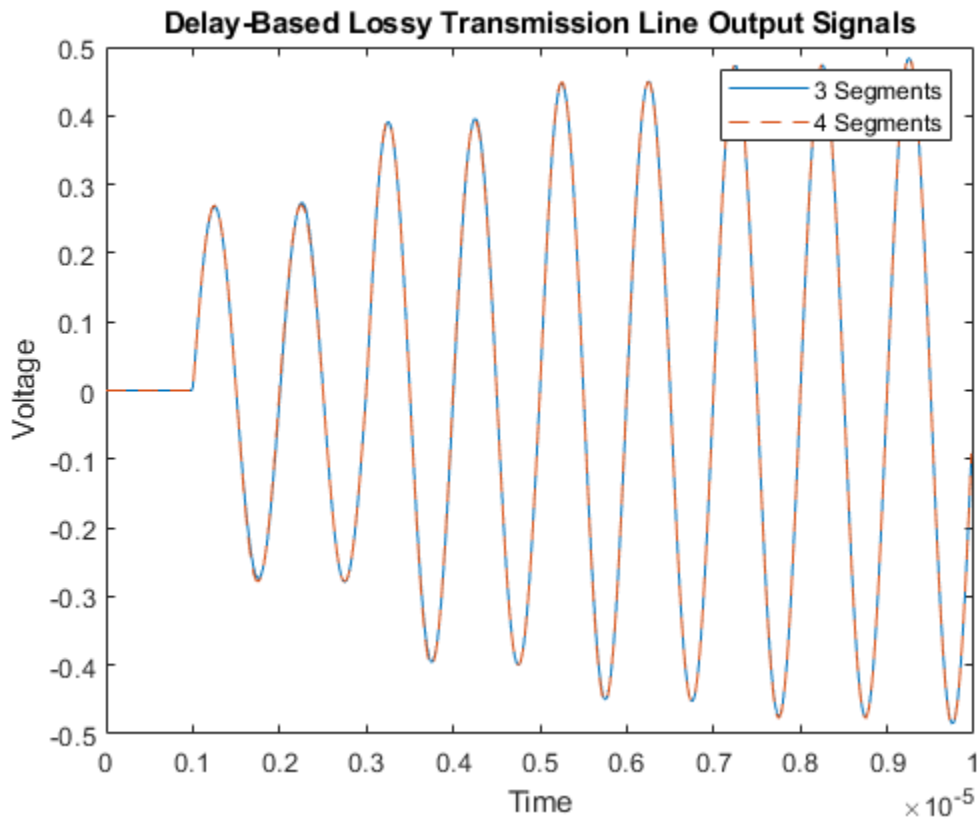
The lossy delay-based transmission line model, `simrf_xline_seg`, consists of two parallel arms excited by a RF Blockset sinusoidal source. The top arm employs a single segment transmission line, while the bottom arm uses a line consisting of 3 segments. The source and load resistances are not equal to the characteristic impedance of the transmission line. These differences affect the shape of the output response. For example, the output response will be overdamped when the source and load resistances are less than the characteristic impedance.

```
open_system([model_seg '/Output Voltage']);
sim(model_seg);
```



Increasing the number of line segments in the bottom arm from three to four and comparing responses show that three segments suffice for this configuration.

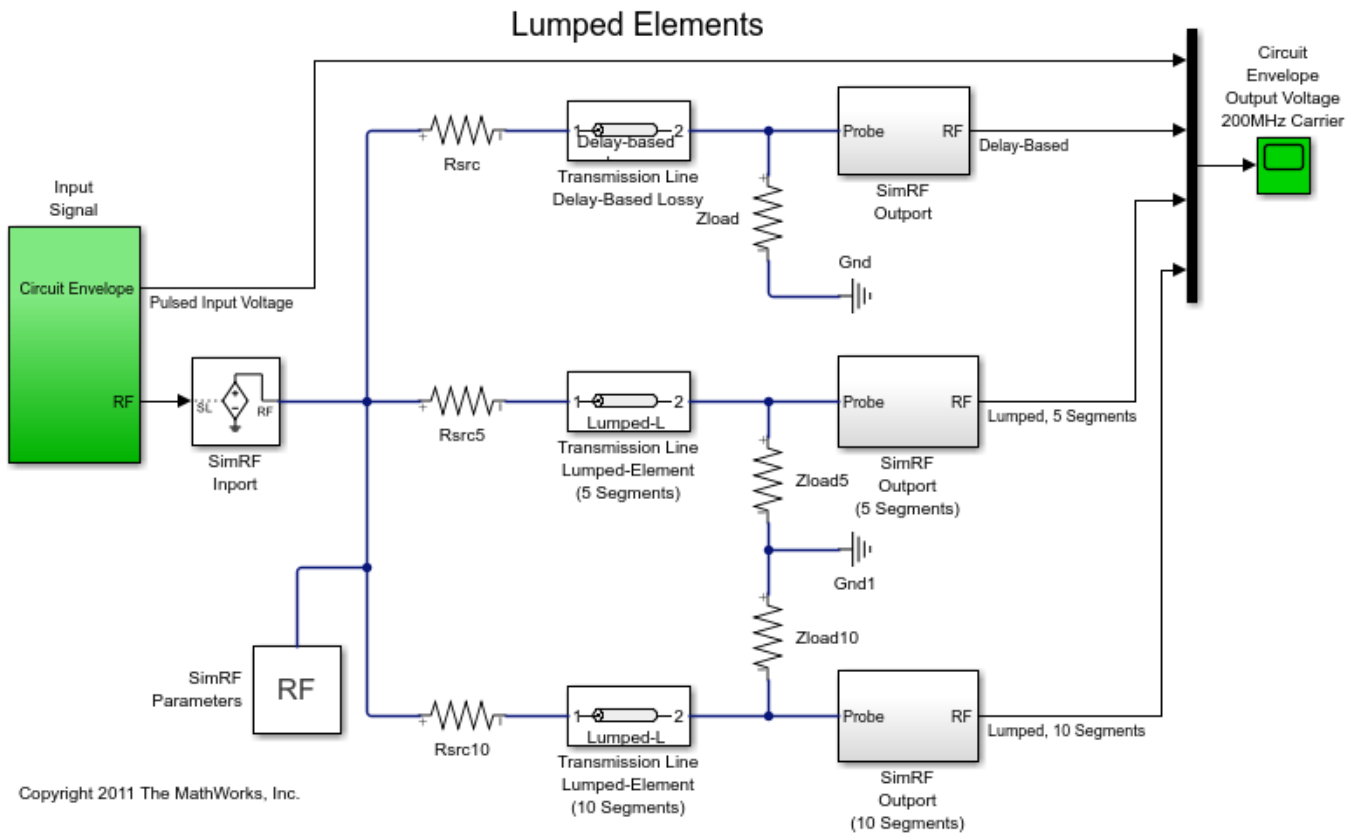
```
close_system([model_seg '/Output Voltage']);
ST_Data3 = ST_Data;
set_param([model_seg '/Transmission (3 Segments)'],'NumSegments','4')
sim(model_seg);
plot(haxis, ST_Data3(:,1), ST_Data3(:,4), ST_Data(:,1), ST_Data(:,4), '--')
legend('3 Segments', '4 Segments')
title('Delay-Based Lossy Transmission Line Output Signals')
xlabel('Time')
ylabel('Voltage')
```



System Architecture for Lumped Element Transmission Line

Differences between the lumped element and delay-based transmission lines are now examined. Consider the model `simrf_xline_ll`, where the dialog box parameter `Model_type` is `Delay-based` and `lossy` for the top arm and `Lumped` parameter `L-section` for the other two arms. The Inductance per unit length and Capacitance per unit length parameters values for the L-section lines are similar to a $50\ \Omega$ coaxial cable. Basic first order approximations for these lines are $Z_0 = \sqrt{L/C}$ and $T_D = \sqrt{L * C} * Length$.

```
model_ll = 'simrf_xline_ll';
open_system(model_ll)
```

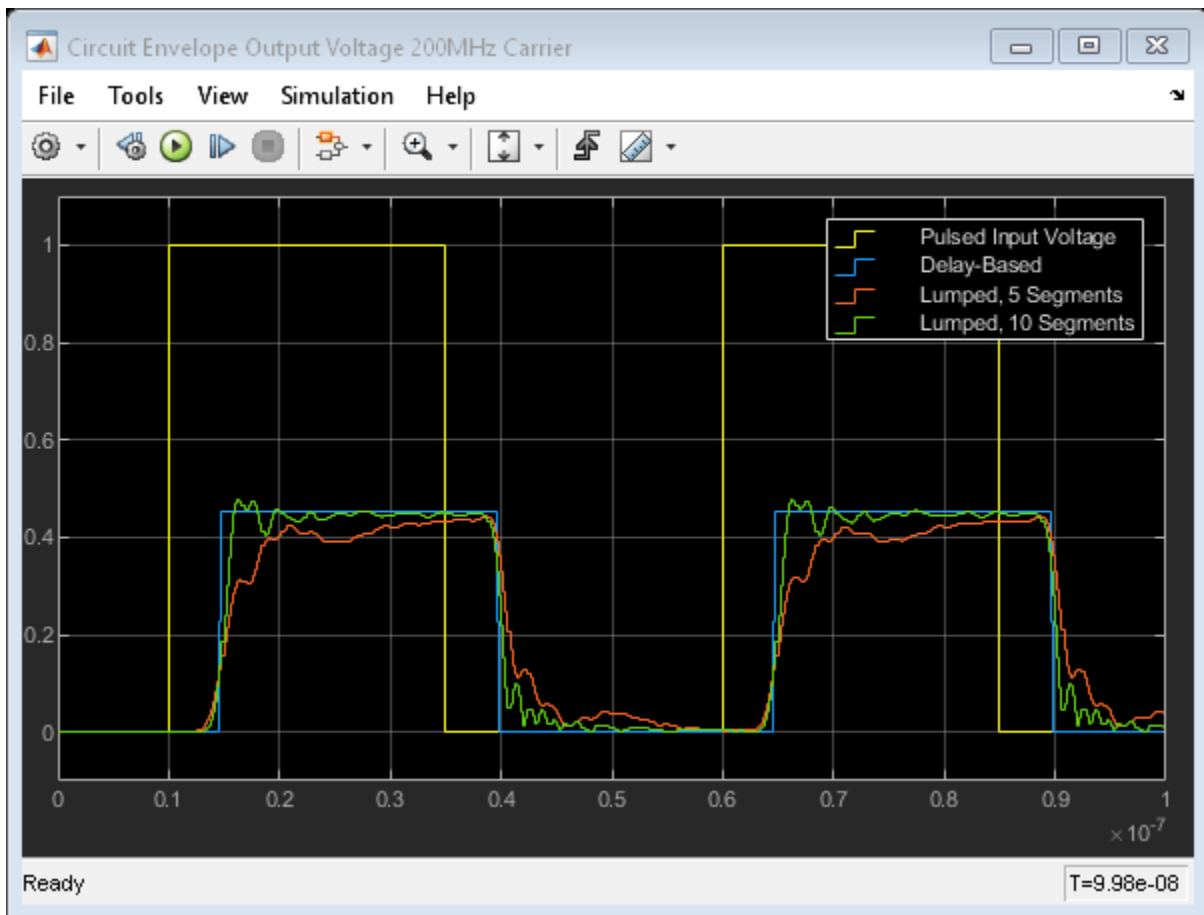


Running the Lumped Element Transmission Line

- 1 Type `open_system('simrf_xline_ll')` at the Command Window prompt.
- 2 Select **Simulation > Run**.

The following graph shows how the number of lumped element segments affects the output. Speed and accuracy must be balanced when using the lumped-element transmission line block.

```
open_system([model_ll '/Circuit Envelope Output Voltage 200MHz Carrier']);
sim(model_ll);
```



Cleaning Up

Close the model and remove workspace variables.

```
close(get(haxis, 'Parent'))
clear haxis hline;
bdclose({model_pb model_ce model_seg model_ll});
clear SCE_Data SPB_Data ST_Data ST_Data3 SLL_Data;
clear model_pb model_ce model_seg model_ll;
```

References

Sussman-Fort and Hantgan, *SPICE Implementation of Lossy Transmission Line and Schottky Diode Models*. IEEE Transactions on Microwave Theory and Techniques, Vol. 36, No. 1, January 1988

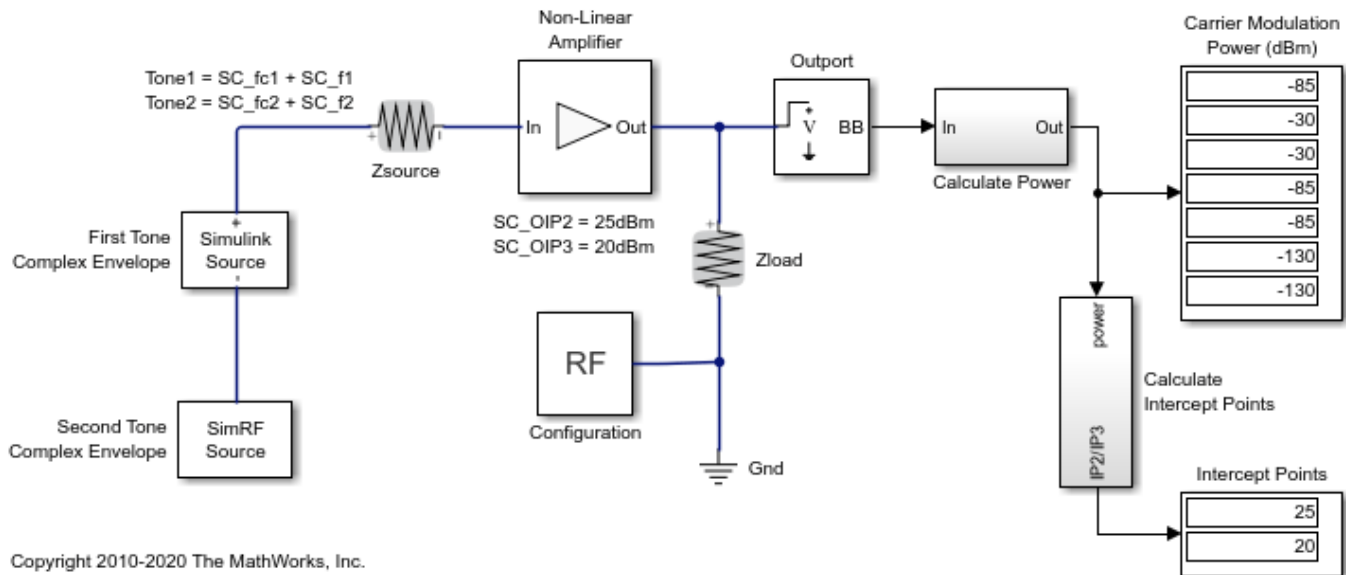
True Kenneth M, *Data Transmission Lines and Their Characteristics*. Application Note 806, April 1992

See Also

“Compare Time and Frequency Domain Simulation Options for S-parameters” on page 8-40

Validating IP2/IP3 Using Complex Signals

This example shows how to use the RF Blockset™ Circuit Envelope library to run a two-tone experiment that measures the second- and third-order intercept points of an amplifier. The model computes the intercept points of the amplifier from the modulated signal power measured on each carrier, verifying the behavior of the RF Blockset system. These values are confirmed using the RF Budget Analyzer app and a measurement testbench.



System Architecture

The system consists of:

- Two complex voltage sources connected in series. The first voltage source is modeled with Simulink® blocks, and the second with blocks from the RF Blockset circuit envelope library. In the Simulink Source subsystem, two series Sine Wave blocks model in-phase and quadrature components of the first tone. An Inport block assigns the Simulink signal to the carrier $fc1$. In the RF Blockset Source subsystem, two series Sinusoid blocks model in-phase and quadrature voltage signals that modulate the carrier $fc2$.
- A resistor modeling the voltage source impedance.
- An amplifier with input impedance, output impedance specified in the Main tab; output IP2, and output IP3 specified in the Nonlinearity tab.
- An Output block that probes the output voltage of the amplifier across a shunt Resistor block. The ordering of the output signals is determined by the ordering of the carriers specified in the Output block dialog.
- A subsystem to compute running rms power levels at the input, IP2 and IP3 frequencies.
- A subsystem to compute IP2 and IP3 intercept points [1].

The example model defines variables for block parameters using a callback function. To access model callbacks, select **MODELING > Model Settings > Model Properties** and click the Callbacks tab in the Model Properties window.

Running the Example

- 1 Type `open_system('simrfV2_carriers')` at the Command Window prompt.
- 2 Select **Simulation > Run**.

Output power and amplifier output intercept points are displayed on the right side of the model. The Calculate Power subsystem computes the power in dBm of each intermodulation product using a running root-mean-square (RMS) average.

Modeling Nonlinear RF Blockset Components

To model nonlinearities in the RF Blockset circuit envelope environment:

- Place an Amplifier or Mixer block in your model.
- Specify parameters that generate nonlinearities, such as **IP2** and **IP3**, taking care to specify the convention or specify a polynomial directly in the Nonlinearity tab of the block dialog.
- Specify any additional carrier frequencies for simulation in the Configuration block. In this example, the Configuration block specifies a total of twenty five frequencies : `fc1` and `fc2`, as the **Fundamental Tones** of the input signals; a **Harmonic Order** of 3 for each tone resulting in a complete set of second, third, fourth-order intermodulation products(second and third-order harmonic products included), and a partial set of fifth and sixth-order intermodulation products.

In order to calculate the power level of each envelope, the measured voltage signals are scaled with the inverse of the square root of the characteristic impedance. An additional scaling of $1/\sqrt{2}$ in the Calculate Power subsystem normalizes the complex-valued output signal.

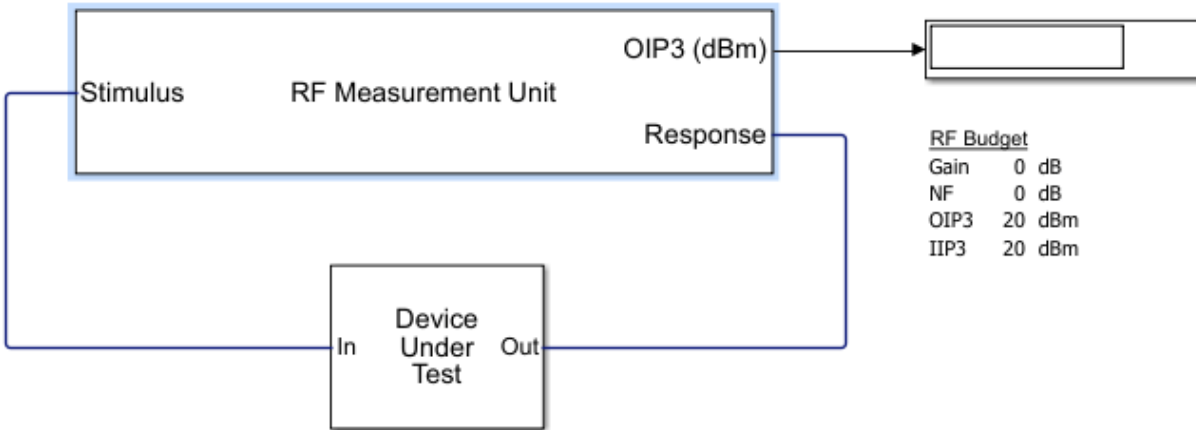
Validate Using Measurement Testbench

The same measurements can be performed using the RF Budget Analyzer app to automatically generate the model and testbench.

- Open the RF Budget Analyzer app and specify an amplifier.
- Define its IP3 (IP2 cannot be specified at this time).
- Generate the measurement testbench.

RF Measurement Testbench

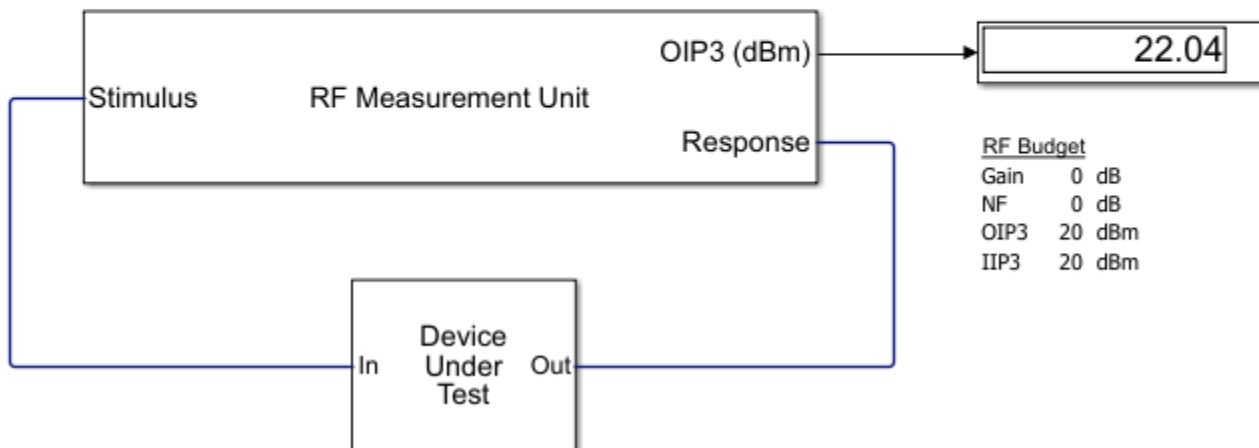
Open the Block Parameters dialog of the RF Measurement Unit block for measurement-specific [parameters](#) and [instructions](#).



- Open Device Under Test to reveal the amplifier. Specify the IP2 value.
- Disable the noise to make an accurate measurement for IP2 and IP3. (Use the RF Measurement Unit dialog box).
- Run the simulation and measure IP3.

RF Measurement Testbench

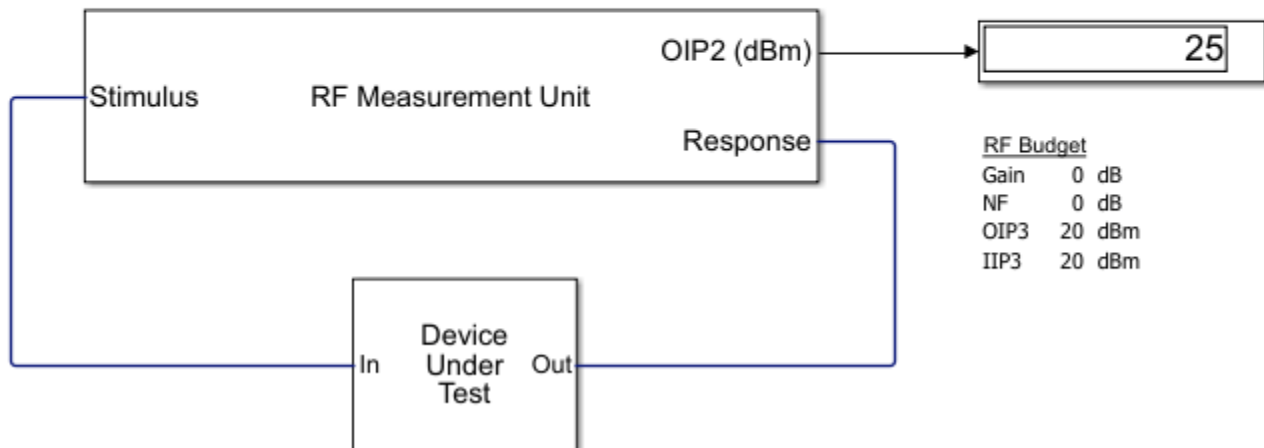
Open the Block Parameters dialog of the RF Measurement Unit block for measurement-specific [parameters](#) and [instructions](#).



- Change the quantity to be verified to IP2 and rerun the simulation.

RF Measurement Testbench

Open the Block Parameters dialog of the RF Measurement Unit block for measurement-specific [parameters](#) and [instructions](#).



If you look under the mask of the testbench, you will find the logic to measure IP2 and IP3. This methodology is very similar to what is described in the initial model.

Reference

- 1 Kundert, Ken. "Accurate and Rapid Measurement of IP2 and IP3." *The Designers Guide Community*, Version 1b, May 22, 2002.

See Also

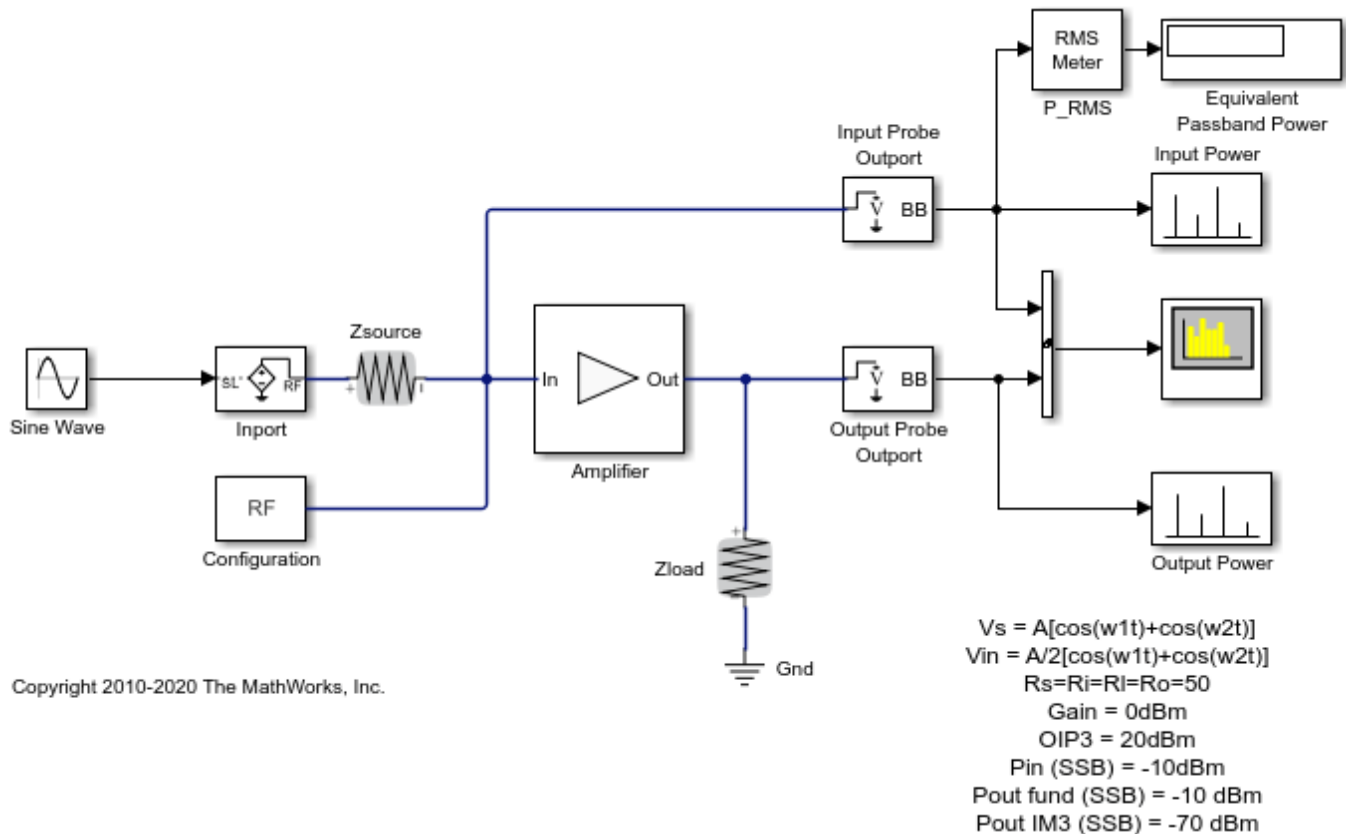
IIP3 Testbench | IIP2 Testbench

Related Examples

- "Two-Tone Envelope Analysis Using Real Signals" on page 8-61

Two-Tone Envelope Analysis Using Real Signals

This example shows how to use the RF Blockset™ Circuit Envelope library to test intermodulation distortion of an amplifier using two-carrier envelope analysis.



System Architecture

The system consists of:

- A Simulink® sinusoidal input with frequency $(f_1 - f_2)/2$, and an Inport that assigns the input modulation to the carrier $(f_1 + f_2)/2$. This formulation is equivalent to the sum of two sinusoids with frequency f_1 and f_2 , according to the sum-product formula for cosines:

$$\cos 2\pi f_1 t + \cos 2\pi f_2 t = 2 \cos \left(2\pi \frac{f_1 + f_2}{2} t \right) \cos \left(2\pi \frac{f_1 - f_2}{2} t \right)$$

- An amplifier with specified 0 dB linear gain and -20 dB OIP3. For an input signal u , the amplifier computes the output v according to the polynomial

$$v(t) = 2u(t) - \frac{4}{15}u^3(t)$$

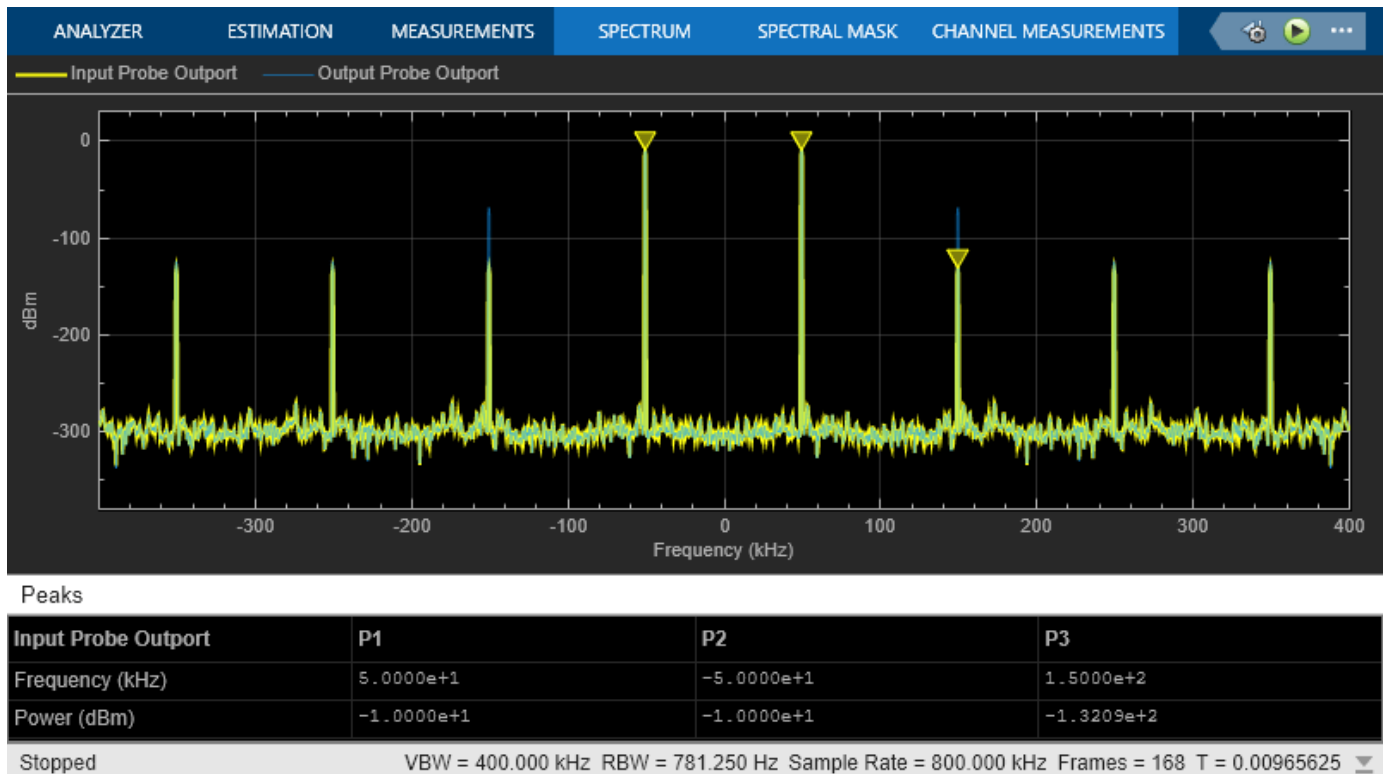
- Two Output blocks that probe the input and output voltages (across shunt resistors) at the carrier frequency.

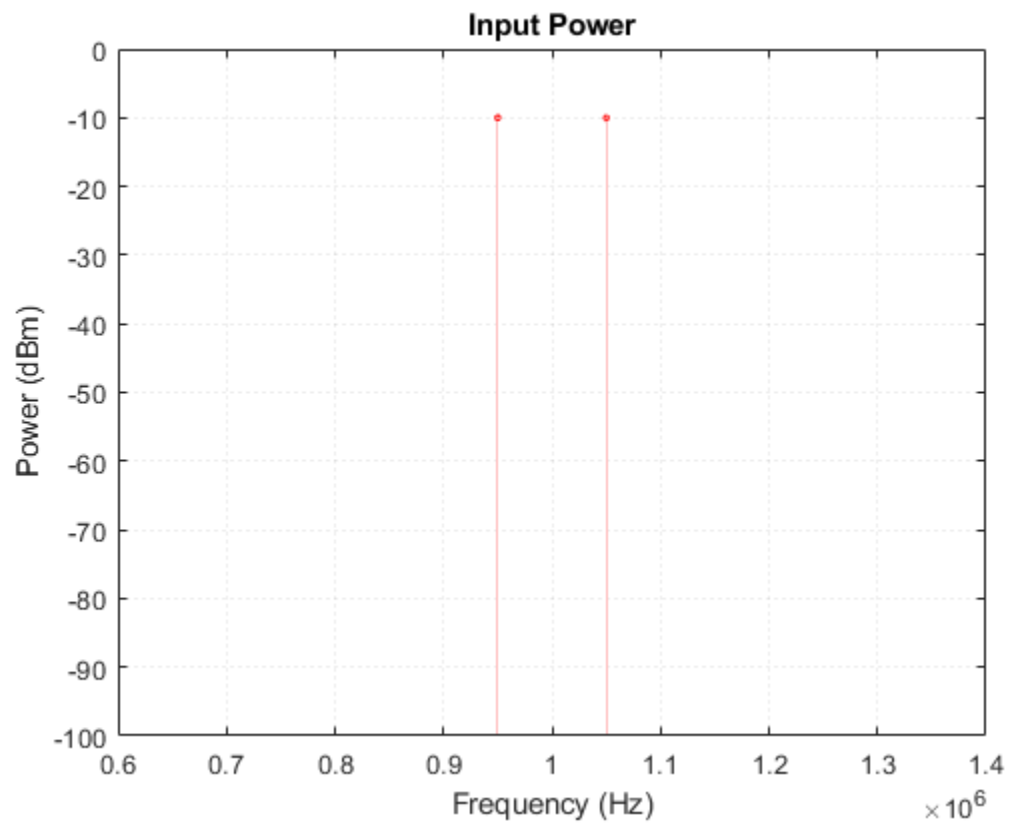
- A spectrum analyzer

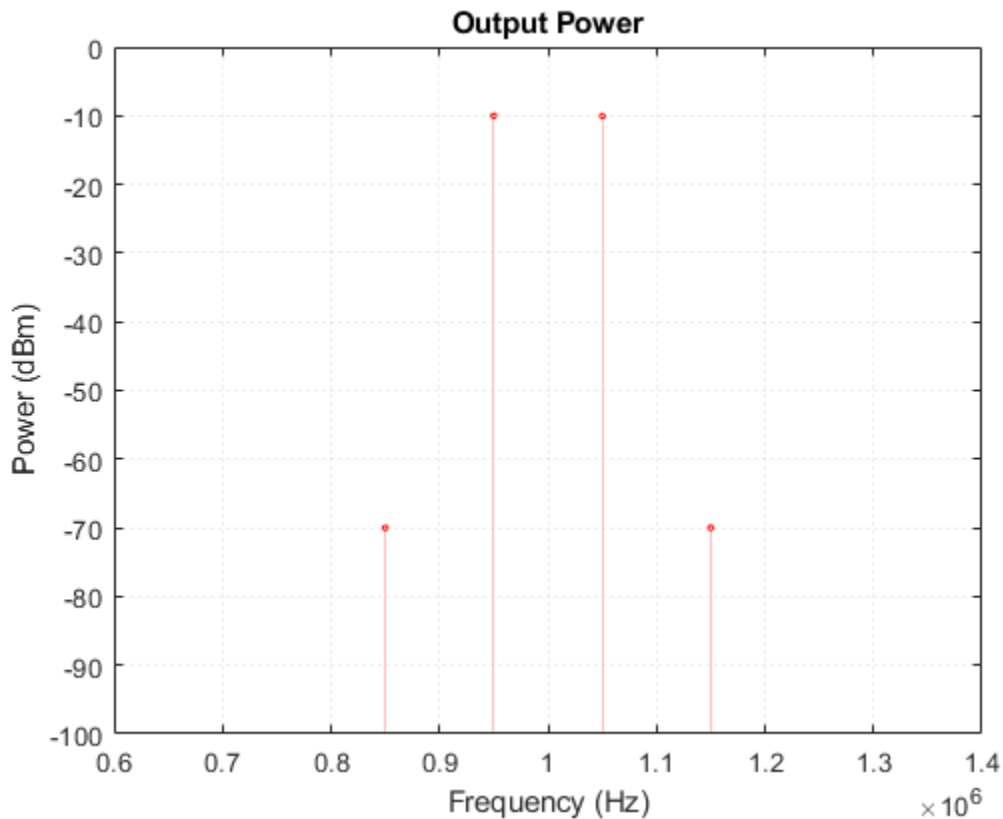
The example model defines variables for block parameters using a callback function. To access model callbacks, select **MODELING > Model Settings > Model Properties** and click the Callbacks tab in the Model Properties window.

Running the Example

- 1 Type `open_system('simrfV2_power_imd')` at the Command Window prompt.
- 2 Select **Simulation > Run**.







The resulting output power spectrum, labeled 'Output Power', shows third-order intermodulation distortion. The Display block shows the power of the bandpass waveform, which is half of power of the envelope waveform.

The spectrum analyzer is set to probe for the Output Power. On comparing values, you will see that the spectrum analyzer matches the output power spectrum.

The input drives the nonlinear amplifier into compression, so the linear component of the output is attenuated. To simulate the amplifier in a linear region:

- Specify In_f for the IP3 parameter of the Amplifier block located in the Nonlinearity tab of the dialog, or
- Reduce the power of the input signal by decreasing the value of the Amplitude parameter of the Sine Wave block.

Performing Two-Tone Analysis Using Circuit Envelope Simulation

This example takes advantage of the properties of real signals--namely, the sum-product equivalence of sinusoids. To perform the same experiment on a different RF system:

- 1 Choose f_1 and f_2 , the frequencies of the test tones.
- 2 Use a Simulink Sine source and an Inport block to model modulation with frequency $(f_1 - f_2)/2$ on a carrier with frequency $(f_1 + f_2)/2$. Alternatively, you can use a Sinusoid block from the RF Blockset Circuit Envelope library to specify both a modulation and a carrier simultaneously.
- 3 Specify $(f_1 + f_2)/2$ as one of your carrier frequencies in the Configuration block dialog.

- 4 Use an Outport block to probe the distorted signal.

References

Cripps, Steve C. *RF Power Amplifiers for Wireless Communications*. Artech House, Inc., 2006.

See Also

Amplifier | Configuration | Inport

Related Topics

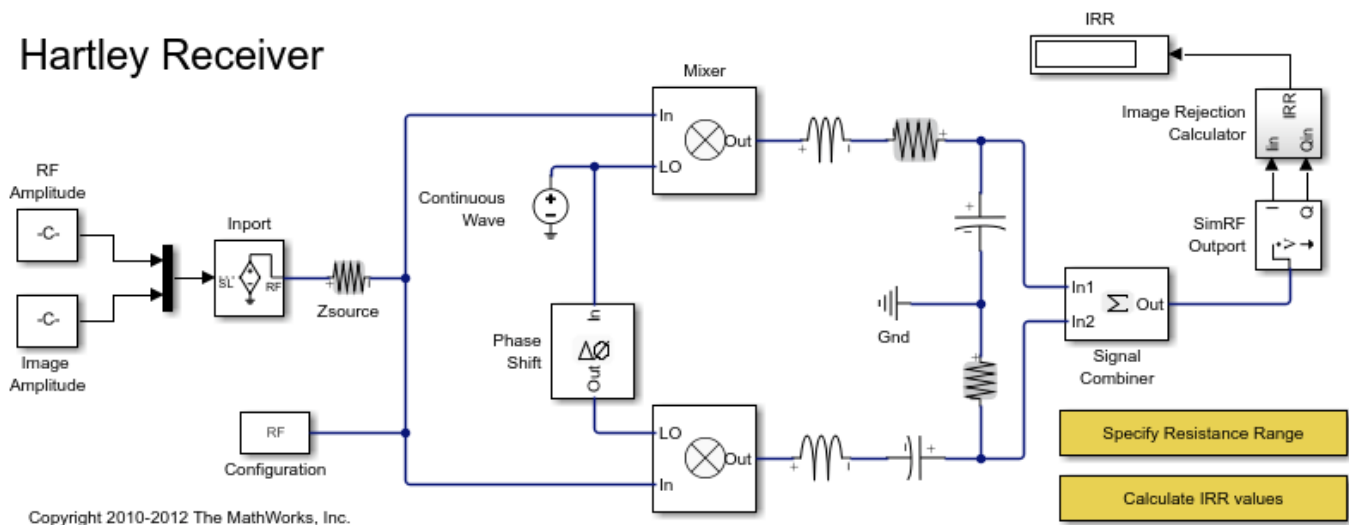
“Validating IP2/IP3 Using Complex Signals” on page 8-57

Measuring Image Rejection Ratio in Receivers

This example shows how to use the RF Blockset™ Circuit Envelope library to calculate the image rejection ratio (IRR) for high-side-injection in Weaver and Hartley receivers. The Weaver receiver shows the effect of phase offset on IRR, and the Hartley receiver shows a similar effect for resistor variation.

```
model1 = 'simrfv2_hartley';
open_system(model1);
```

Hartley Receiver



System Architecture

The RF system consists of:

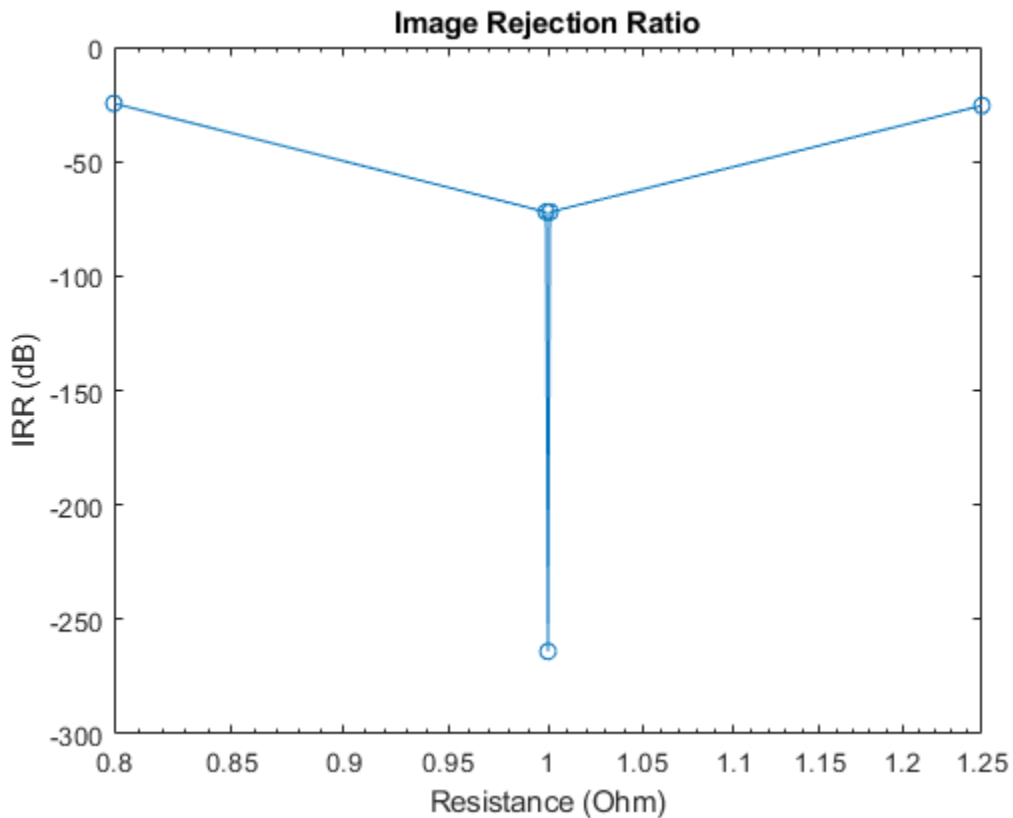
- An Inport block that assigns multiplexed outputs of the RF and Image, by using Simulink® Constant blocks, to the carriers f_{c_RF} and f_{c_IM} respectively. Real and imaginary values of the Constant blocks are matched to in-phase and quadrature carrier components.
- First stage mixers that mix the input signal with a local oscillator modeled by a Continuous Wave block with frequency f_{c_LO} . The LO frequency is the average of the RF and image frequencies, so both signals are mixed down to the same frequency, f_{c_IF} . The LO phase is shifted 90 degrees in one mixer relative to the other.
- The second stage of the Hartley uses a frequency independent RC-CR network to produce an additional 90 degree phase shift between the two signal paths, while the Weaver employs two additional mixers for channel selection.
- A Signal Combiner block that sums the voltage signals at its two inputs to yield the RF signal. If the Signal Combiner block is used to perform subtraction, the image can be obtained instead of the RF signal at its output. For low-side-injection, the Signal Combiner block needs to perform subtraction.
- The values of the in-phase and quadrature components of the RF and image signals are chosen to reduce the number of IRR calculations and facilitate reuse of the Image Rejection Calculator.

Simulating the Hartley Receiver

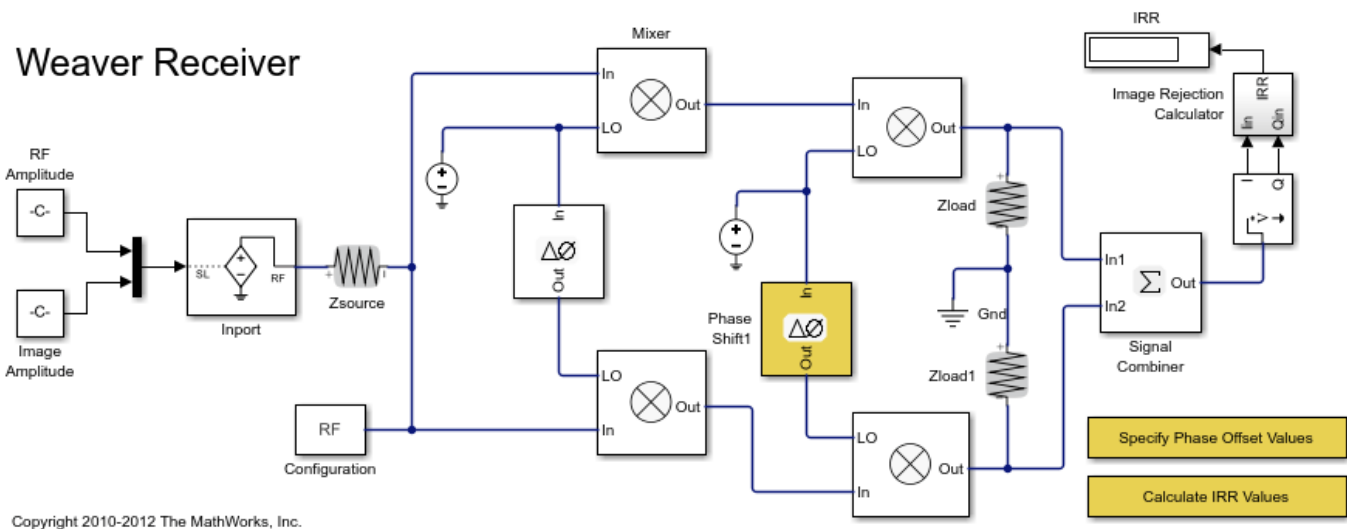
- 1 Type `open_system('simrfV2_hartley')` at the Command Window prompt.
- 2 Double-click 'Specify Resistance Range' and specify a set of resistance values for the highlighted resistor.
- 3 Double-click 'Calculate IRR values' to execute a script, `simrfV2_hartley_callback`, that simulates the model once for each specified resistance value and generates a plot.

The sensitivity of the architecture to the component variation is shown by simulating the system multiple times, varying the resistance of the highlighted Resistor block at each iteration. When the highlighted resistor has a resistance of 1 Ohm, the images sum to zero in the Signal Combiner block and the IRR is minus infinity.

```
evalc('simrfV2_hartley_callback');
```



```
bdclose(model1);
model2 = 'simrfV2_weaver';
open_system(model2);
```

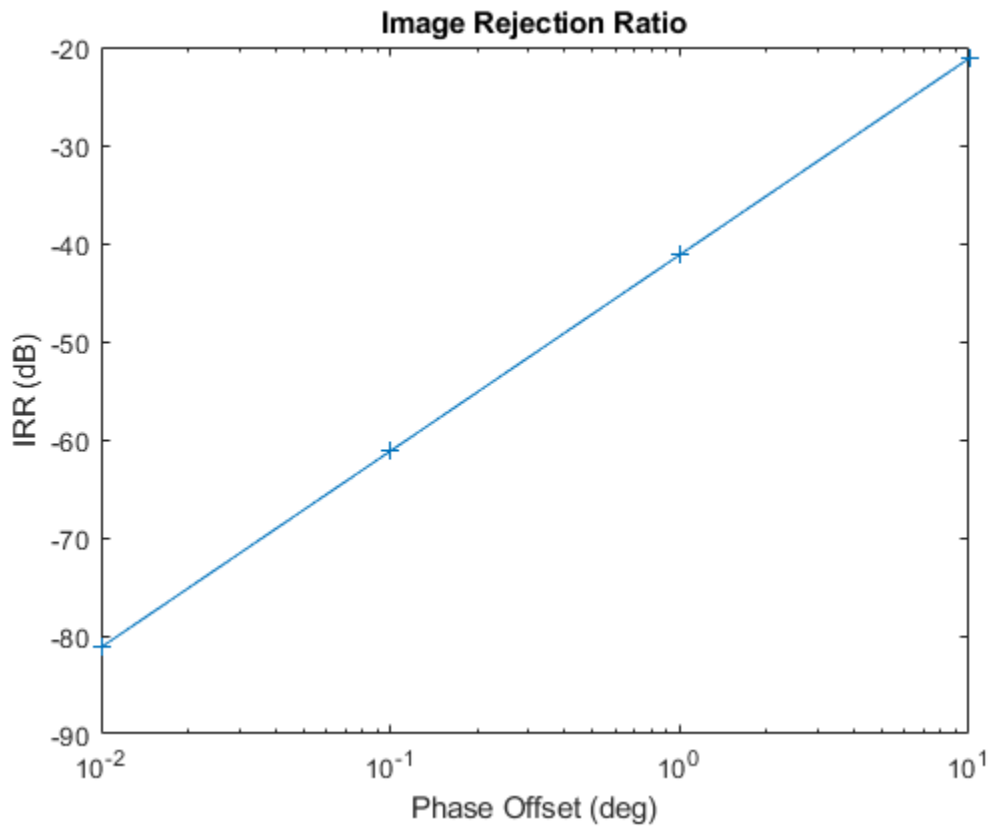


Simulating the Weaver Receiver

- 1 Type `open_system('simrfV2_weaver')` at the Command Window prompt.
- 2 Double-click 'Specify Phase Offset Values' and specify a set of phase offset values.
- 3 Double-click 'Calculate IRR values' to execute a script, `simrfV2_weaver_callback`, that simulates the model once for each specified offset and generates a plot.

The sensitivity of the architecture to LO phase offset is shown by simulating the system multiple times, varying the phase offset of the highlighted Phase Shift block at each iteration. When the phase offset of the highlighted Phase Shift block is zero, the images sum to zero in the Signal Combiner block and the IRR is minus infinity.

```
evalc('simrfV2_weaver_callback');
```



```
bdclose(model2)
```

See Also

Mixer

Related Topics

“Measuring Image Rejection Ratio in Receivers” on page 8-66

Executable Specification of a Direct Conversion Receiver

This example shows how to use the RF Blockset™ Circuit Envelope library to simulate the sensitivity performance of a direct conversion architecture with the following RF impairments:

- Component noise
- LO-RF isolation
- Interference from blocker signals
- Local oscillator phase offset
- ADC dynamic range
- Component mismatch

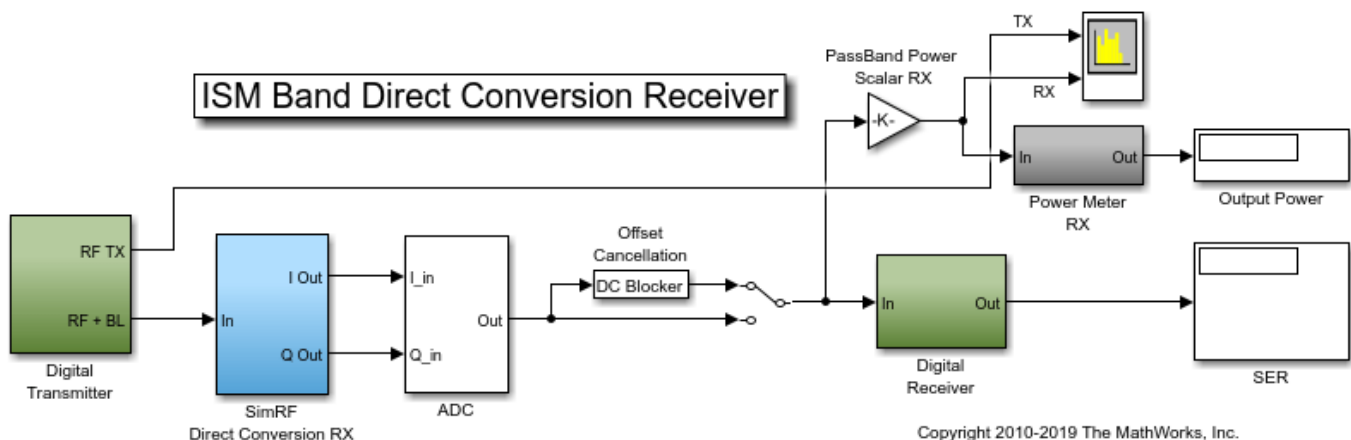
The RF portion of the model includes the explicit specification of gain, noise figure, IP2 and IP3, input/output impedance, and LO phase offset. The transmitter side of the RF interface includes modulation scheme, signal power, and blocker power. The carrier frequencies for the transmitted waveforms are specified in the Inport block. The baseband side includes the number of symbols and full scale range of the ADC.

System Architecture:

This system model illustrates the design and simulation of a Direct Conversion ISM Band Receiver. The model is comprised of blocks from RF Blockset, Communications Toolbox™, DSP System Toolbox™, and Simulink® libraries. The primary subsystems in the model include a digital transmitter, an RF receiver, an ADC, a DC offset correction, and a digital receiver. Blocks and plotted signals are color coded:

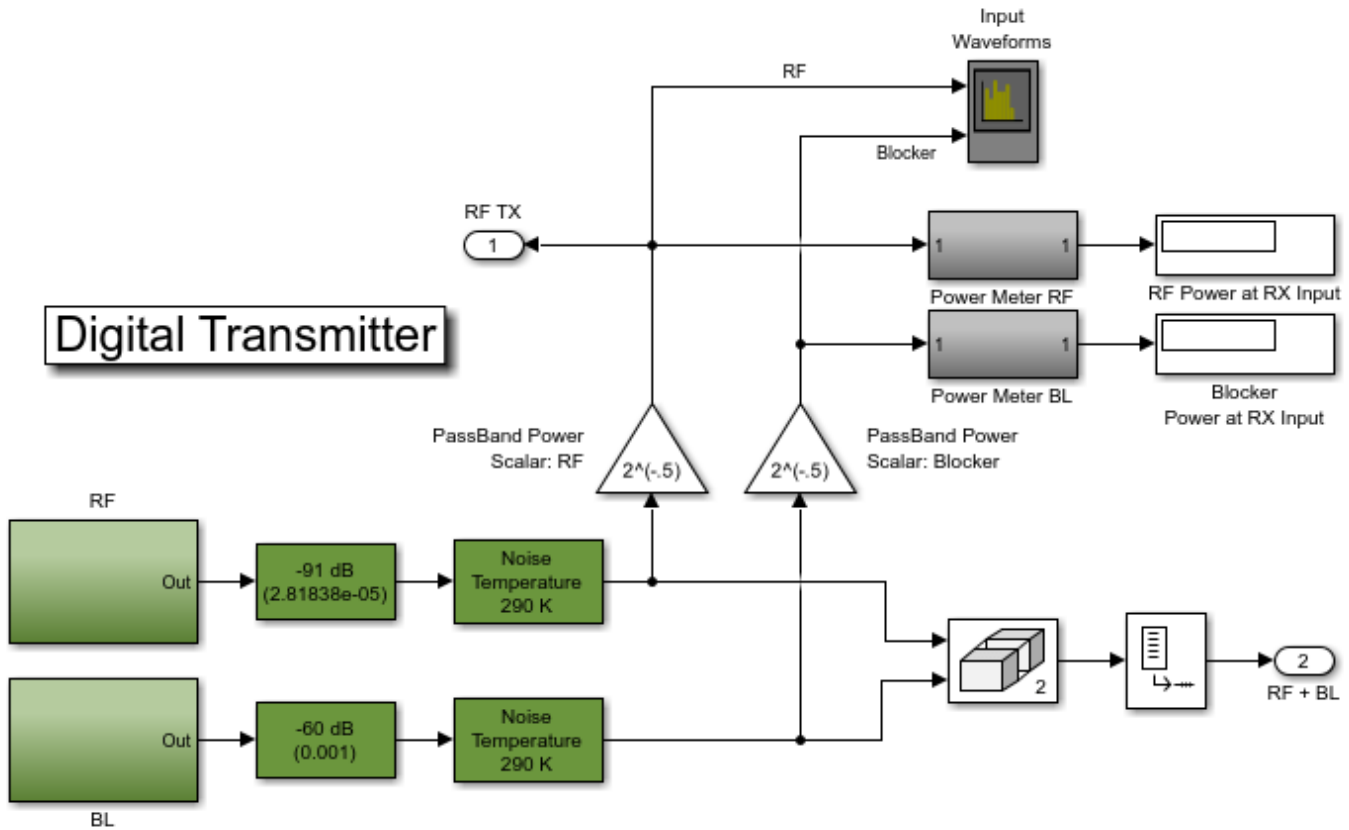
- RF Blockset: Light Blue
- Communications Toolbox: Green
- DSP System Toolbox: Grey
- Simulink: White

```
model = 'simrfv2_direct_conv';
open_system(model)
```

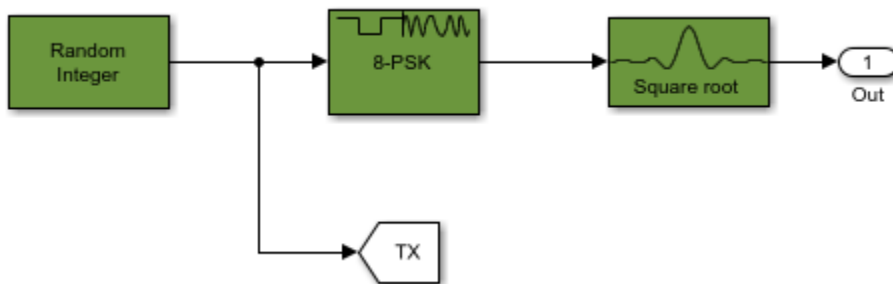


The digital transmitter consists of two 8-PSK modulated waveforms, a target waveform and an interfering waveform. The waveforms are scaled by $1/\sqrt{2}$ for passband-waveform power measurement and spectrum visualization.

```
open_system([model '/Digital Transmitter'])
```



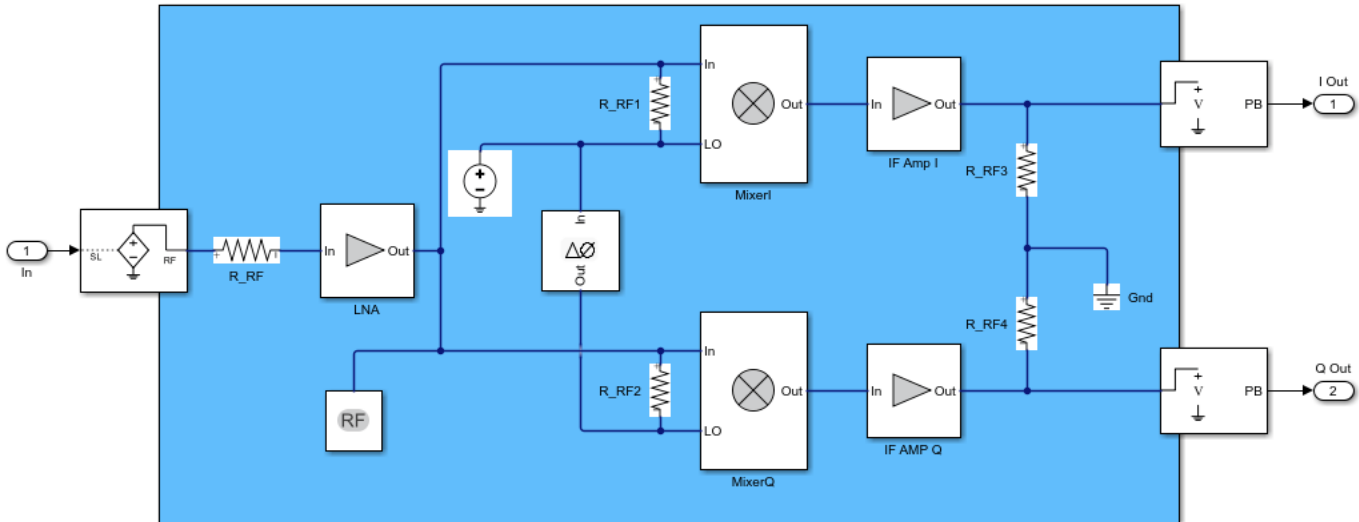
```
open_system([model '/Digital Transmitter/RF'])
```



The direct-conversion RF receiver has a frequency conversion stage and two gain stages. Resistors model input and output impedances of the RF system as well as the isolation between the LO and RF ports of the mixers. Each of the blocks captures RF impairments relevant to this design. Each of the nonlinear blocks is specified by noise figure. The LNA non-linearity is specified by IP3, and the nonlinearity in the IF amplifiers are specified by both IP2 and IP3. The mixer nonlinearity is specified by IP2. A single LO and a phase shift block provide the cosine and sine terms to the I and Q branches,

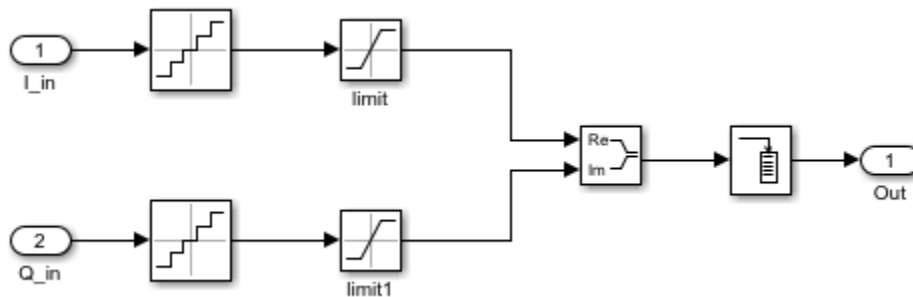
respectively. To model a thermal noise floor in the RF Blockset environment, the Temperature parameter in the Configuration block specifies a noise temperature of 290.0 K.

```
open_system([model '/SimRF Direct Conversion RX'])
```



The ADC uses an N-bit quantizer followed by a saturation block to model the full-scale range. Hence, the ADC properly models the system quantization noise floor.

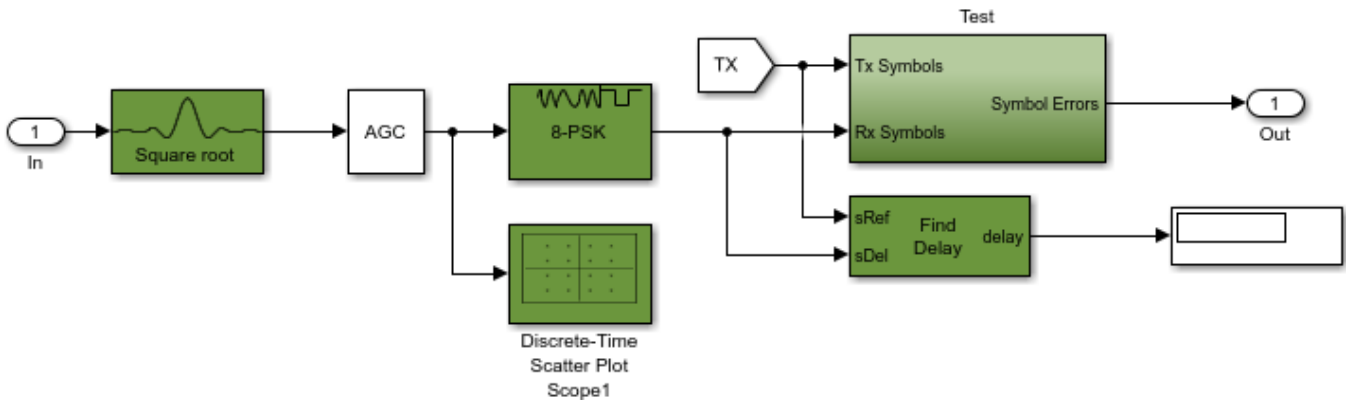
```
open_system([model '/ADC'], 'Force')
```



The DC offset cancellation block employs an IIR algorithm for estimating offset. This DC offset correction is enable or disable during simulation using the manual switch that follows the offset block.

The digital receiver applies a matched filter to the received waveform followed by an AGC function, and demodulates the waveform for symbol-error-rate calculation.

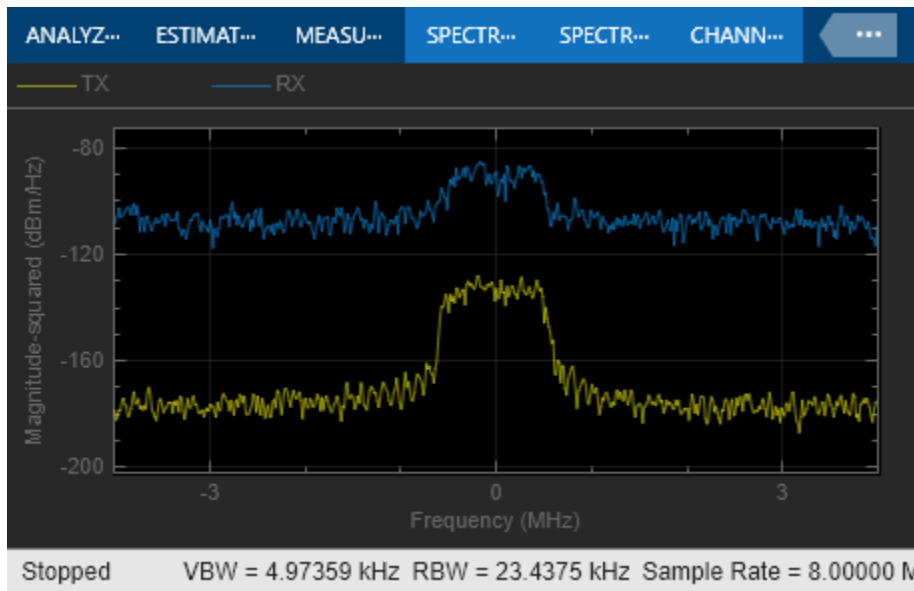
```
open_system([model '/Digital Receiver'])
```

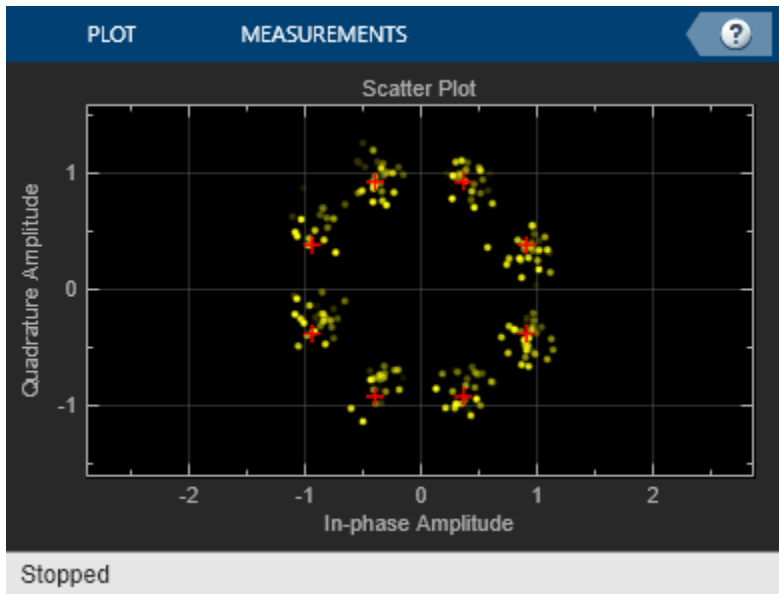


Running the Example

Running the example simulates a design that meets an uncoded 0.5% SER specification. Modifications to the signal power levels and component specifications in the receiver and ADC have a direct impact on the receiver performance. The manual switch in the design enables the user switch the offset correction subsystem in and out to visualize the DC offset effect associated with RF-LO isolation.

```
sim(model, 'stoptime', '1e-3');
```



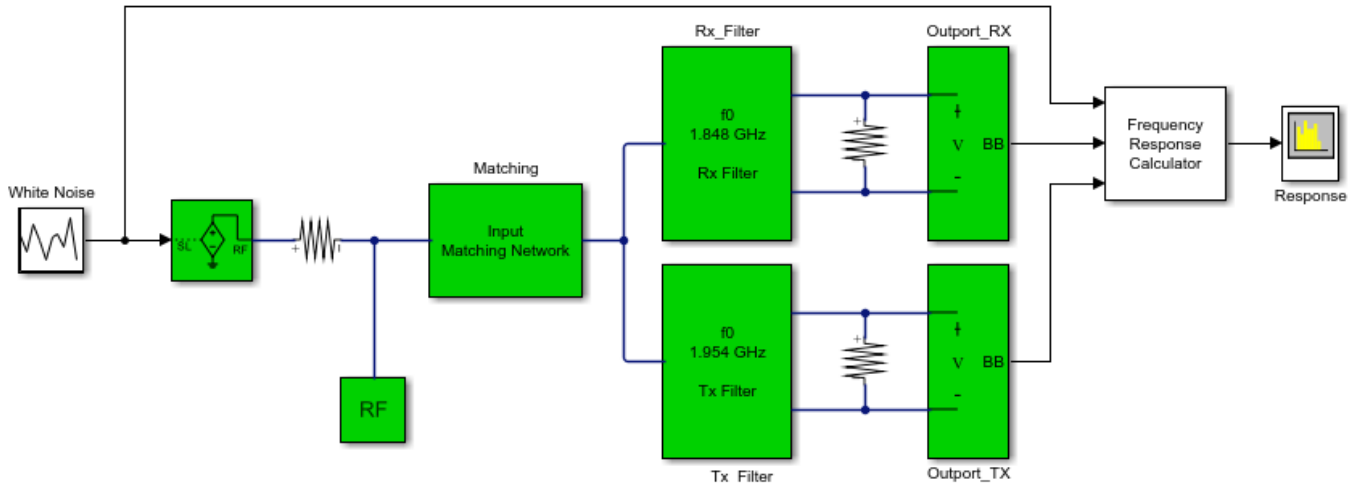


```

bdclose(model)
clear model
    
```


Frequency Response of RF Transmit/Receive Duplex Filter

This example shows how to use blocks from the RF Blockset™ Circuit Envelope library to simulate a transmit/receive duplex filter and calculate frequency response curves from a broadband white-noise input. Blocks from the DSP System Toolbox™ libraries generate the input signal, process the output signal, and display the results.



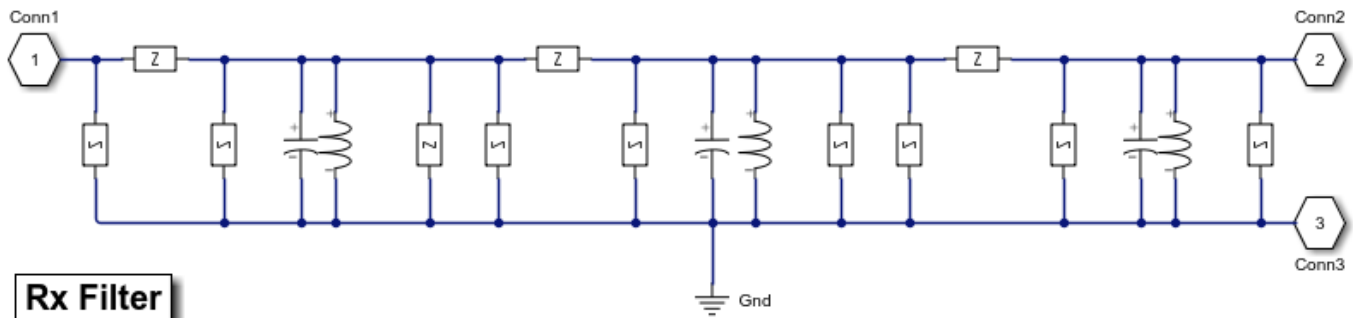
Copyright 2010-2020 The MathWorks, Inc.

System Architecture

The system consists of:

- A DSP System Toolbox Random Source block, which generates broadband white noise.
- Inport and Resistor blocks from the RF Blockset circuit envelope library, which model a controlled voltage source with internal impedance.
- An input matching network, which maximizes the power of the signal received by the filters.
- Duplexed transmit and receive filters, composed of circuit envelope blocks from the RF Blockset Elements sublibrary. The center frequencies of the filters are 1.954 GHz and 1.848 GHz, and the bandwidths are 27 MHz and 18 MHz, respectively. The receive filter is shown in the figure below.
- Two Outputs acting as voltage sensors, measuring the voltage across two load resistors.
- The Frequency Response Calculator subsystem, which computes the voltage transfer functions.
- A DSP System Toolbox Spectrum Analyzer, which displays the frequency response curves.

Block parameters in this example are specified by variables stored in the model workspace. To alter the model workspace, select **Modeling > Model Explorer** and click on the Model Workspace node for this model in the Model Hierarchy pane.

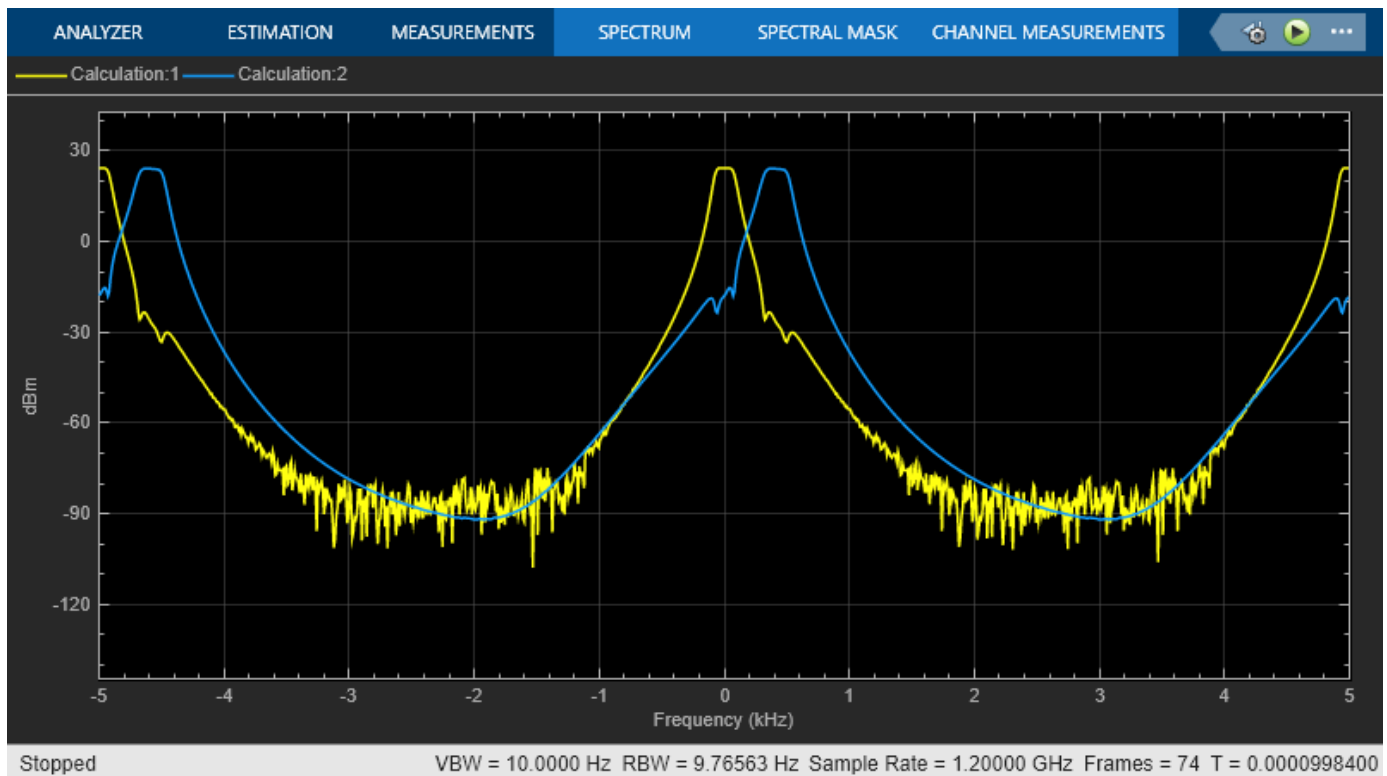


Rx Filter

Running the Example

- 1 Type `open_system('simrfv2_duplexer')` at the Command Window prompt.
- 2 Select **Simulation > Run**.

The Spectrum Analyzer displays the envelope voltage transfer functions of the two filters. Duplexing is evident in the frequency response of the two filters. The frequency values along the x -axis are relative to the RF carrier. To compute the absolute frequencies, add the value of the **Carrier frequencies** parameter of the Inport block to the frequencies shown.



Calculating a Transfer Function Using a Broadband White Noise Source

The Frequency Response Calculator subsystem processes the Simulink® signals converted from the RF Blockset environment at the Output blocks. Within this subsystem, two Compute Transfer Function subsystems correlate the outputs to the input signal.

The Compute Transfer Function subsystems operate on discrete signals. To configure the RF Blockset circuit envelope environment for discrete-time simulation:

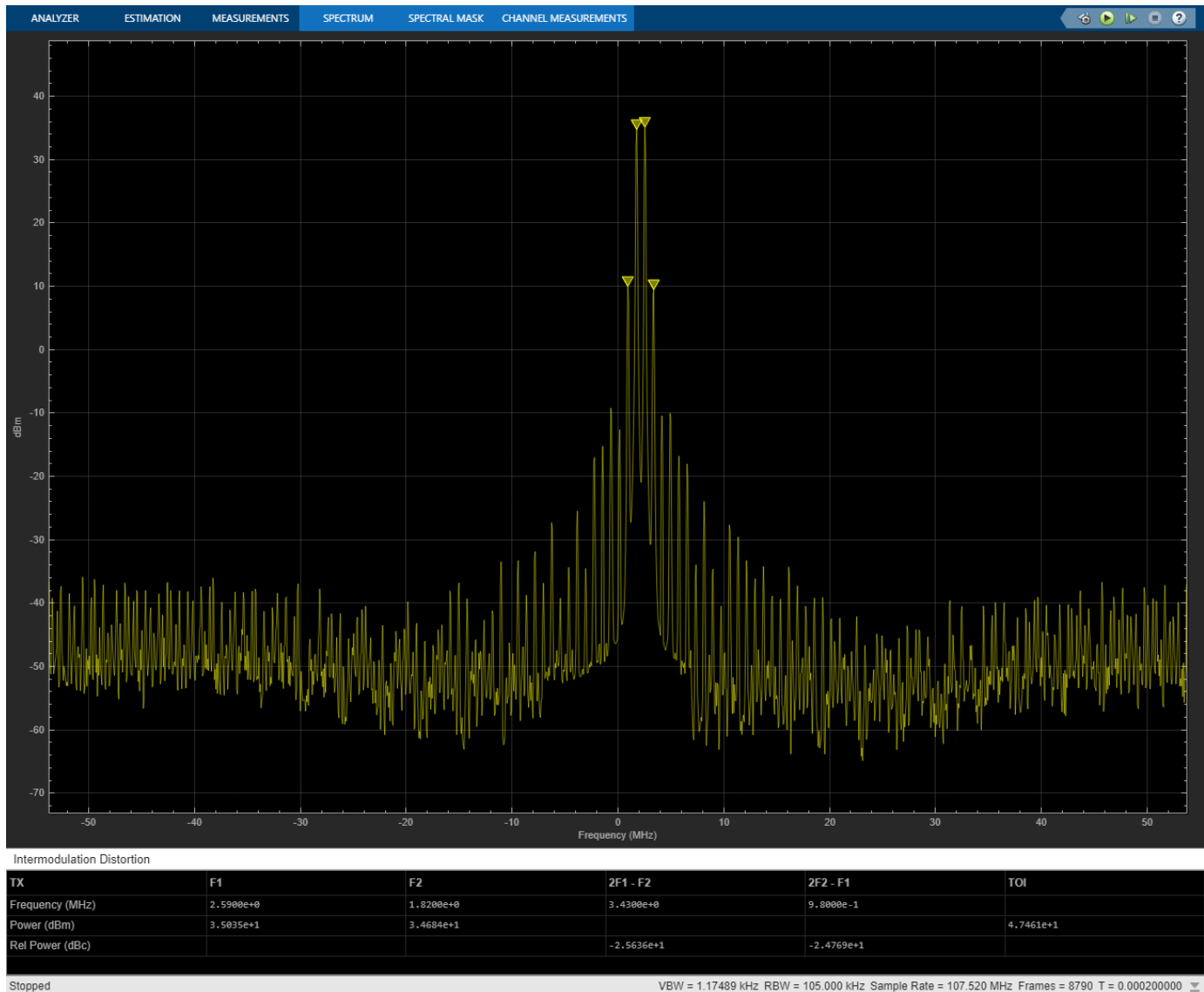
- In the Configuration block dialog, select **Auto** from the **Solver** drop-down menu, and specify the **Step size** parameter. Within the RF Blockset circuit envelope environment, the local solver overrides the solver options specified in the Configuration Parameters dialog box.
- Use a continuous source, or use a discrete source with a sample time equal to the value specified in the Configuration block. In this example, the **Sample mode** parameter of the Random Source block is **Continuous**.

See Also

Inductor | Capacitor | Z (Impedance)

Related Topics

“Architectural Design of a Low IF Receiver System” on page 8-178



The manual switch is toggled to enable the DPD algorithm. When toggled, the TOI (third-order intercept point) is improved significantly. Inspect the distortion measurement in the Spectrum Analyzer to validate these results and see how the power of the harmonics is reduced thanks to the DPD linearization.

Before the two-tone signal enters the DPD block or the power amplifier, it goes through an FIR interpolator, the same FIR interpolator used during PA characterization. This is necessary because the power amplifier model was obtained for the sample rate after interpolation, not the original sample rate of the two-tone signal, and oversampling the signal is required for modeling high order nonlinearities introduced by the power amplifier.

The desired amplitude gain of the DPD Coefficient Estimator is set based on the expected gain of the power amplifier (obtained during PA characterization), because in addition to linearization, the overall goal is to make the combined gain from the DPD input to the power amplifier output as close to the expected gain as possible. To estimate the DPD coefficients correctly, the input signals to the DPD Coefficient Estimator block, PA In and PA Out, must be aligned in the time domain. This is

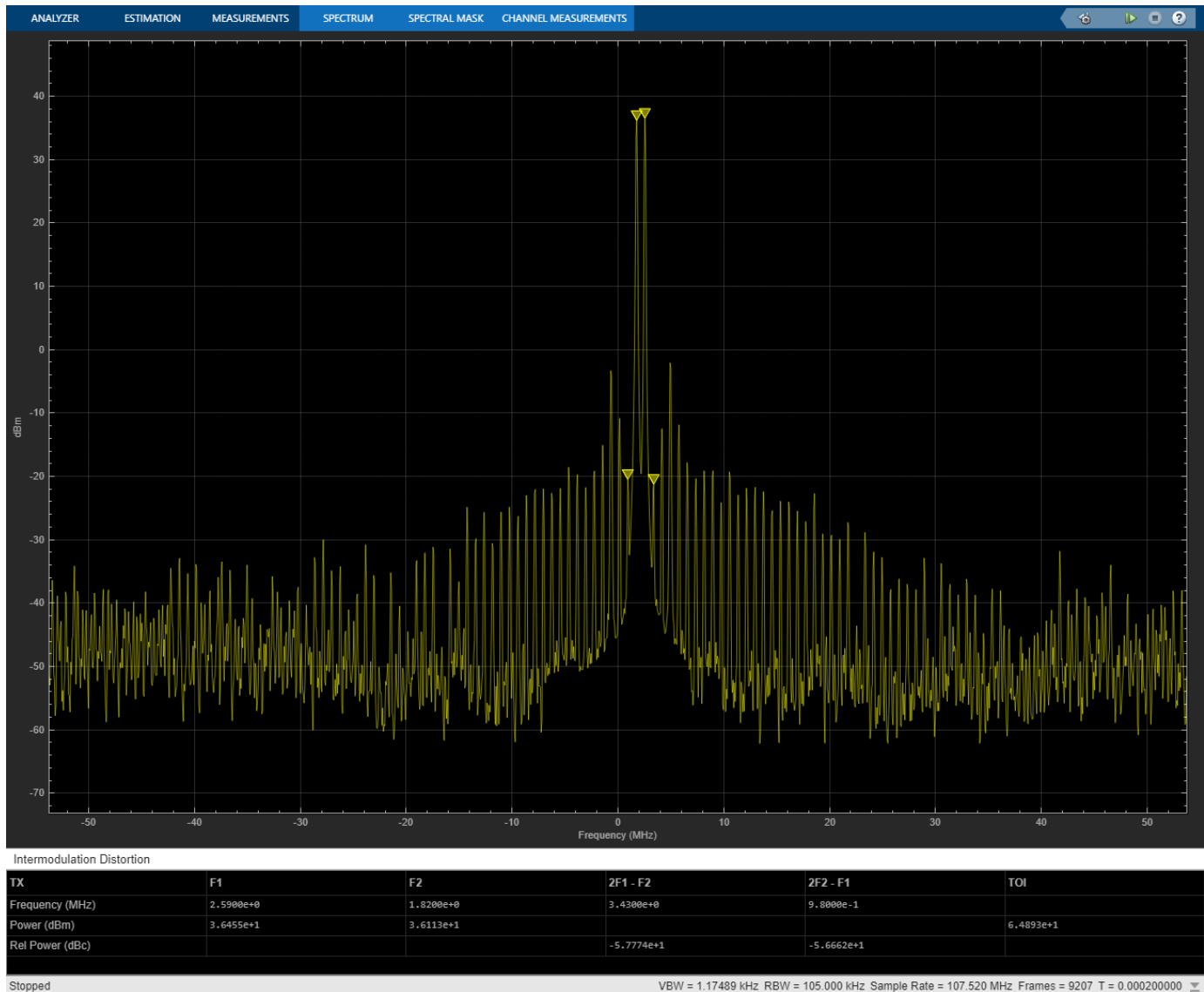
verified by the Find Delay block which shows that the delay introduced by the RF system is 0. Moreover, PA In and PA Out must be accurate baseband representations of the power amplifier input signal and output signal, i.e. no extra gain or phase shift. Otherwise, the DPD Coefficient Estimator block would not observe the power amplifier correctly and would not produce the right DPD coefficients. This is done by ensuring that both the upconversion and downconversion steps have a gain of 1 and the loss and phase shift due to the coupler are properly compensated for before the feedback signal reaches PA Out.

The purpose of the scale factor in front of the FIR interpolator is to help utilize the linearized power amplifier effectively. Even with DPD enabled, two undesirable scenarios may occur. The two-tone signal may be very small with respect to the input range of the linearized system, hence under-utilizing the amplification capability of the linearized system. Or the two-tone signal may be so large that the power amplifier model operates outside the range observed during PA characterization and therefore the power amplifier model may not be an accurate model of the physical device. We use the following heuristic approach to set the scale factor.

Assuming that the DPD block perfectly linearizes the power amplifier to achieve the expected amplitude gain, then the maximum input amplitude allowed by the DPD block should be the maximum power amplifier output amplitude observed during PA characterization divided by the expected amplitude gain. The scale factor before the DPD block should then be the maximum input amplitude allowed by the DPD block divided by the maximum amplitude of the interpolated signal observed during PA characterization.

The system model has a block that calculates the maximum normalized PA input amplitude. If it is equal to 1, it means that the baseband signal entering the RF system has a maximum amplitude equal to the maximum PA input amplitude observed during PA characterization. Therefore, if the maximum normalized PA input amplitude is smaller than 1, the scale factor set by the heuristic approach above may be increased. If the maximum normalized PA input amplitude is greater than 1, the scale factor should be reduced.

```
set_param([model '/Manual Switch'], 'action', '1')
sim(model)
```



By changing the degree and the memory depth defined in the DPD Coefficient Estimator block, you can find the most suitable tradeoff between performance and implementation cost.

```
close_system(model,0)
close all; clear
```

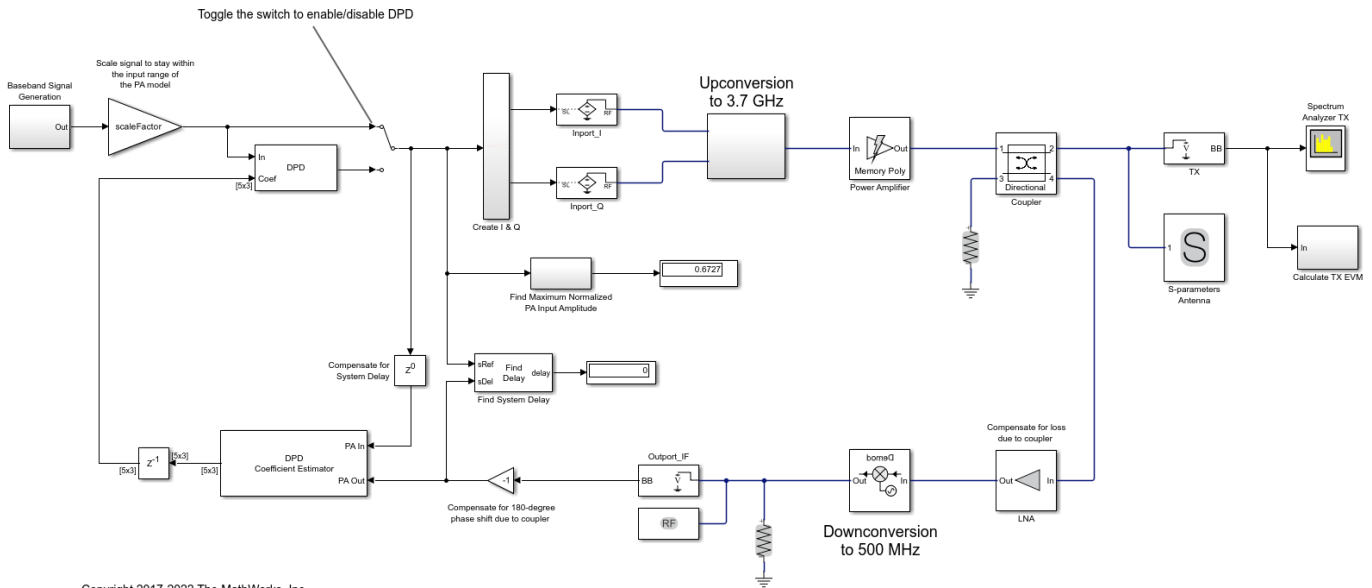
DPD with a 5G-like OFDM Waveform

Open the Simulink RF Blockset model: System-level model PA + DPD with a 5G-like OFDM waveform.

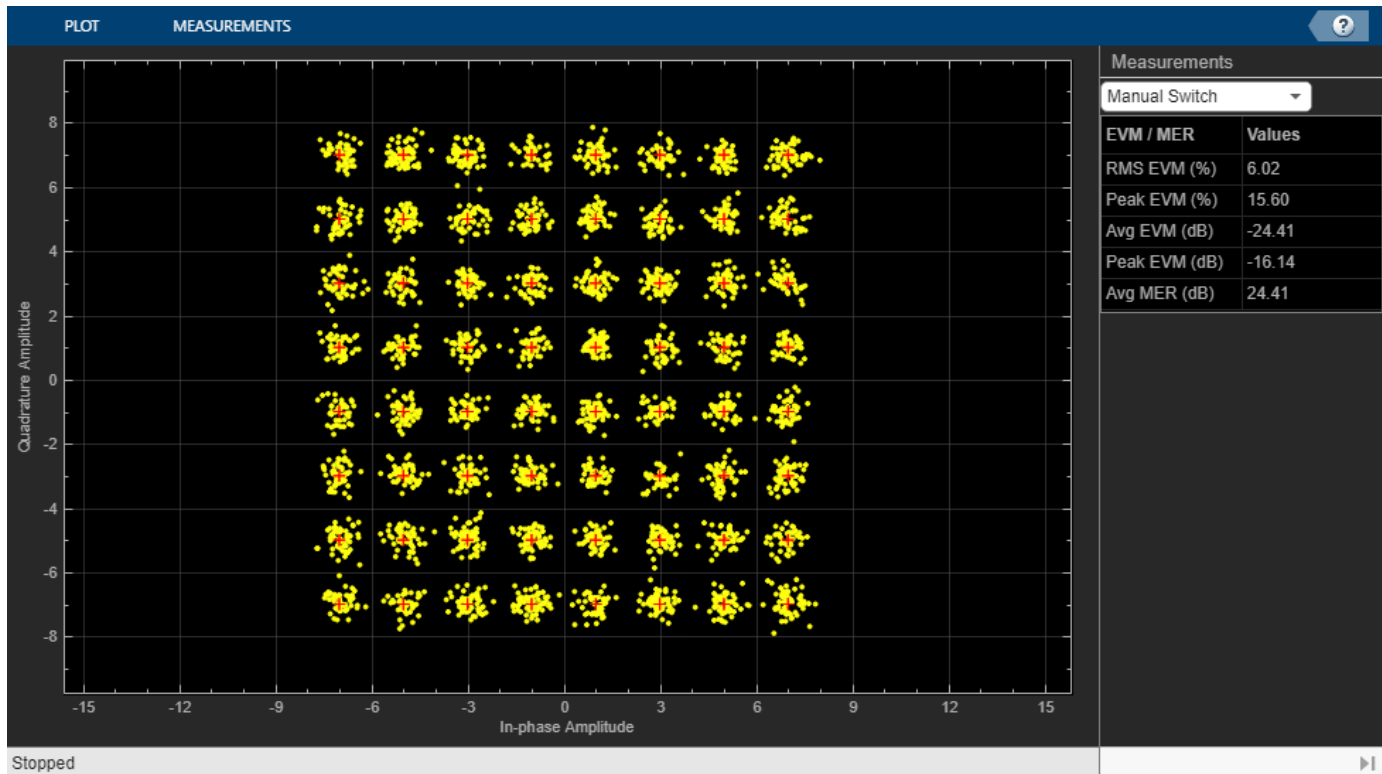
The structure of this Simulink model is the same as that of the previous Simulink model. The signal being amplified is now a 5G-like OFDM waveform, rather than a two-tone signal. Oversampling is done at the OFDM modulator within the baseband signal generation block. The spectrum analyzer measures ACPR instead of TOI and we add a subsystem to measure the EVM and MER of the amplified OFDM waveform.

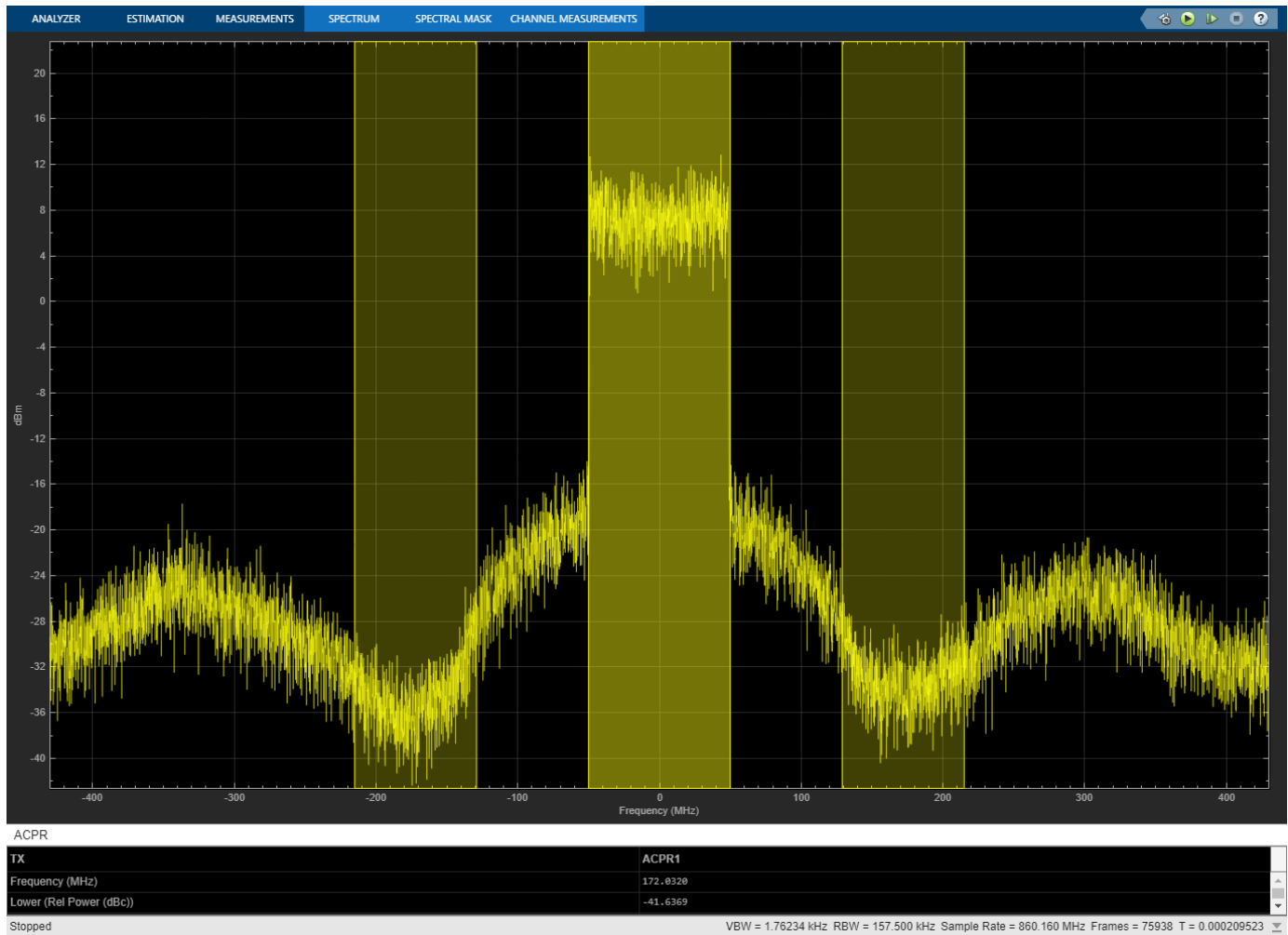
Without DPD linearization, the system achieves an average Modulation Error Ratio of 24.4 dB, as seen from the constellation plot measurement.

```
model = 'simrfV2_powamp_dpd_comms';
open_system(model)
sim(model)
```



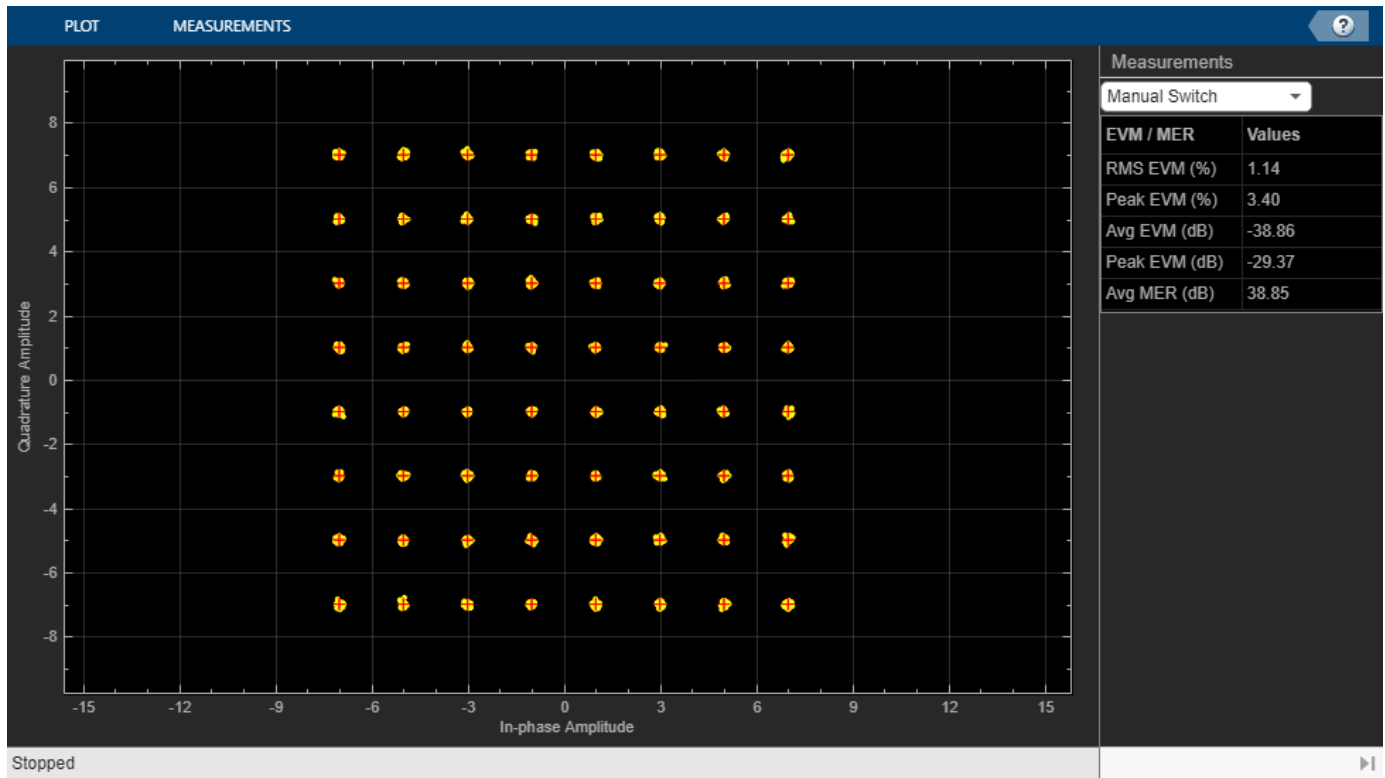
Copyright 2017-2022 The MathWorks, Inc.

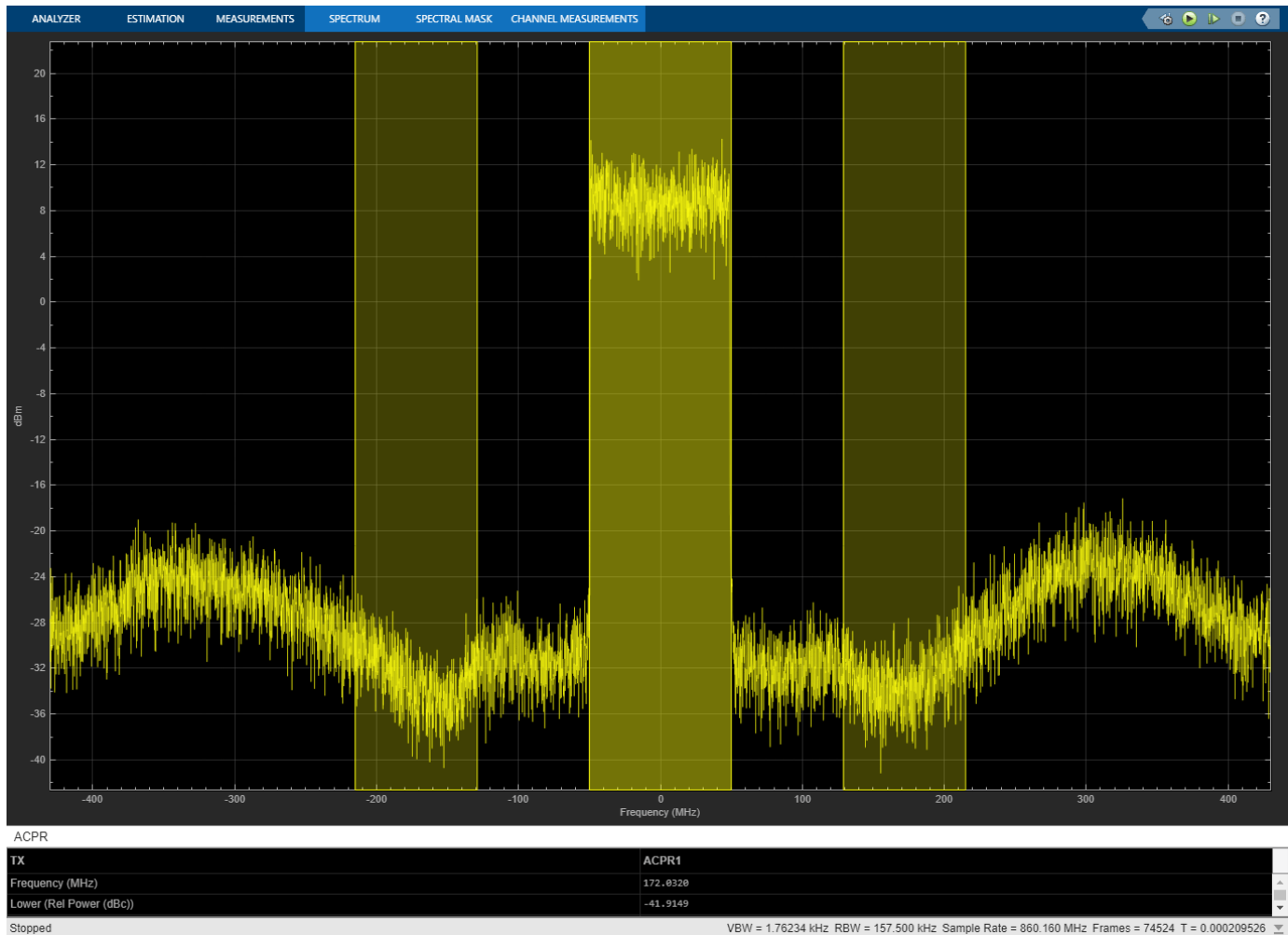




The manual switch is toggled to enable the DPD algorithm. When toggled, the average MER is improved significantly.

```
set_param([model '/Manual Switch'], 'action', '1')
sim(model)
```





```
close_system(model,0)
close all; clear
```

Selected Bibliography

- 1 Morgan, Dennis R., Zhengxiang Ma, Jaehyeong Kim, Michael G. Zierdt, and John Pastalan. "A Generalized Memory Polynomial Model for Digital Predistortion of Power Amplifiers." *IEEE Transactions on Signal Processing*. Vol. 54, No. 10, October 2006, pp. 3852-3860.
- 2 Gan, Li, and Emad Abd-Elrady. "Digital Predistortion of Memory Polynomial Systems Using Direct and Indirect Learning Architectures." In *Proceedings of the Eleventh IASTED International Conference on Signal and Image Processing (SIP)* (F. Cruz-Roldán and N. B. Smith, eds.), No. 654-802. Calgary, AB: ACTA Press, 2009.

See Also

Power Amplifier

More About

- "Power Amplifier Characterization" on page 7-92

- “Two-Tone Envelope Analysis Using Real Signals” on page 8-61
- “Power in Simulink Sources and Signals” on page 8-108
- “Power Ports and Signal Power Measurement in RF Blockset” on page 8-11

Radar System Modeling

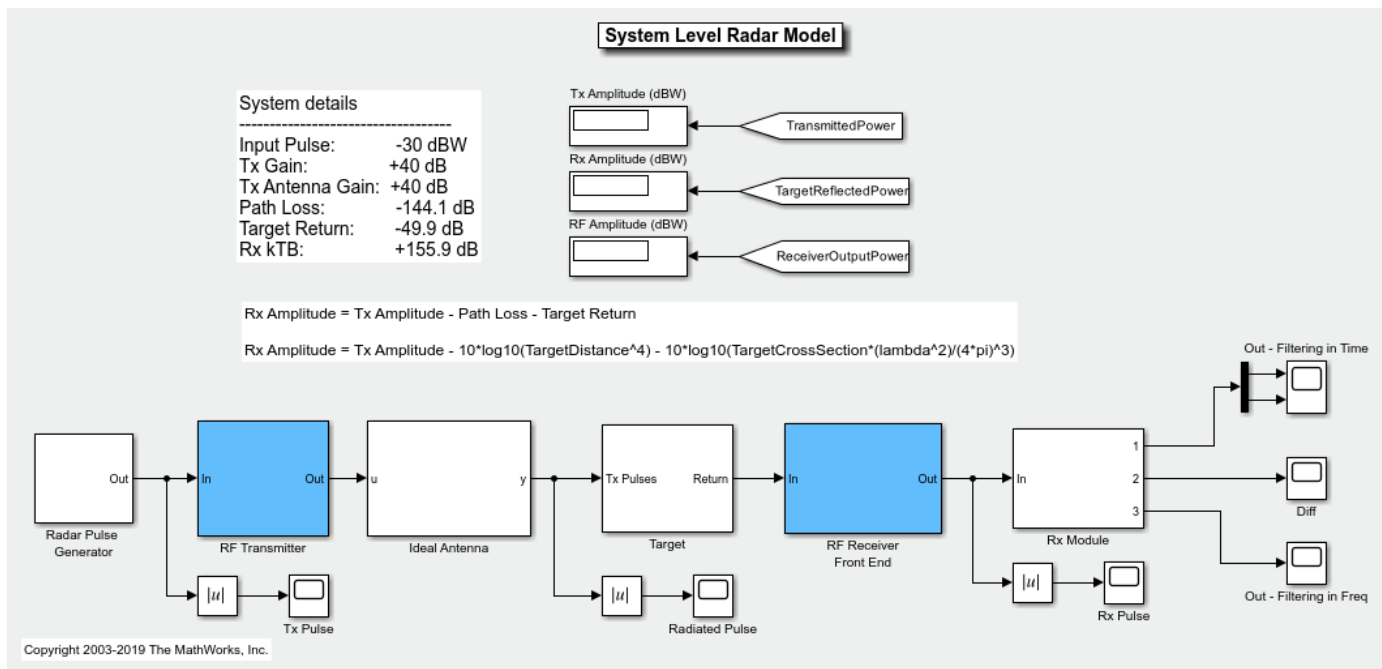
This example shows how to set up a radar system simulation consisting of a transmitter, a channel with a target, and a receiver. For the Aerospace Defense industry, this is an important multi-discipline problem. RF Blockset™ is used for modeling the RF transmitter and receiver sections.

System Architecture

The system consists of:

- A radar pulse generator, which outputs a chirp with a power of 1 mW at a 2% duty cycle (On time = 2 ms, period = 100 ms).
- An RF transmitter section consisting of a filter and an Amplifier implemented using RF Blockset Circuit Envelope library blocks. Since the filter is a linear device and the amplifier is a non-linear device, they are split into two separate independent subsystems. This separation allows the use of different simulation frequency sets in each subsystem. This separation also permits a trade-off between faster simulation speed and the loss of inter-stage loading effects available in a cascaded chain.
- An ideal antenna element with specified boresight gain operating at 2.1 GHz.
- A moving target implementation that reflects the entire incident signal from its cross-sectional surface. The target surface is perpendicular to the incident radar pulse direction of travel.
- An RF Receiver built using the RF Blockset Circuit Envelope library. The direct conversion structure is implemented in the receiver together with LNA and matching networks. The LNA is describe in a touchstone file and the local oscillator includes a phase noise model. Similar to the RF transmitter section, the receiver is split into independent linear and non-linear subsystems. The matching networks, LNA and filter are in the linear section, while the Mixer and final stage amplifiers are in the non-linear section.

The Receive Module in this example serves two purposes. First, the module contains a matched filter detector for target detection. Second, the module serves as a testbench where a theoretical filter implementation is realized via Simulink blocks. The output of each of these filters is compared and their differences plotted.

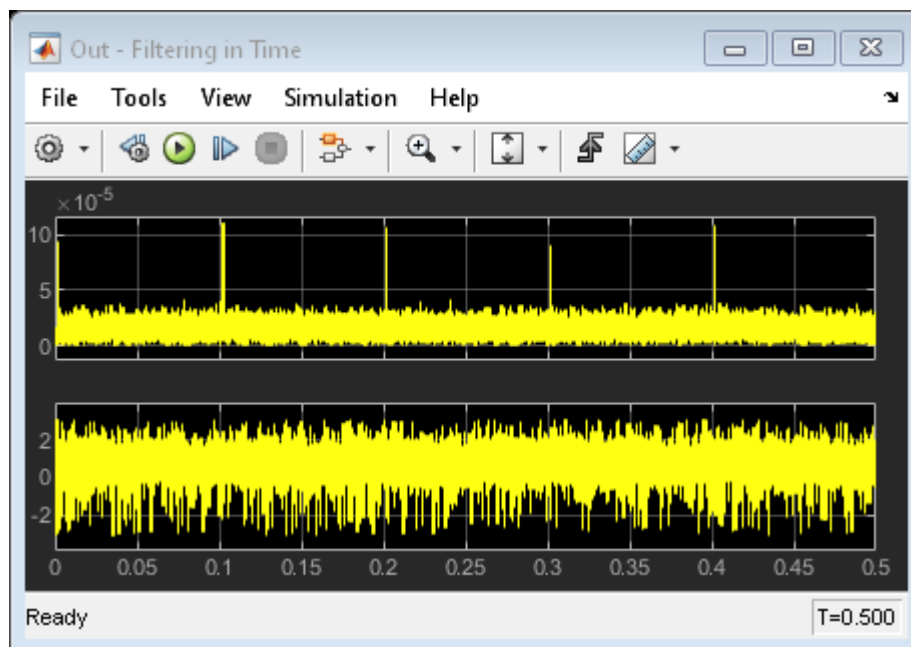


Running Example Using Default Settings

Set the target cross section, target speed, and relative distance to the target by double-clicking the Target icon. At sufficiently large distances or if the target cross section is too small, the return signal cannot be detected because of the noise.

To start the example simulation:

- 1 Select **Simulation > Run**



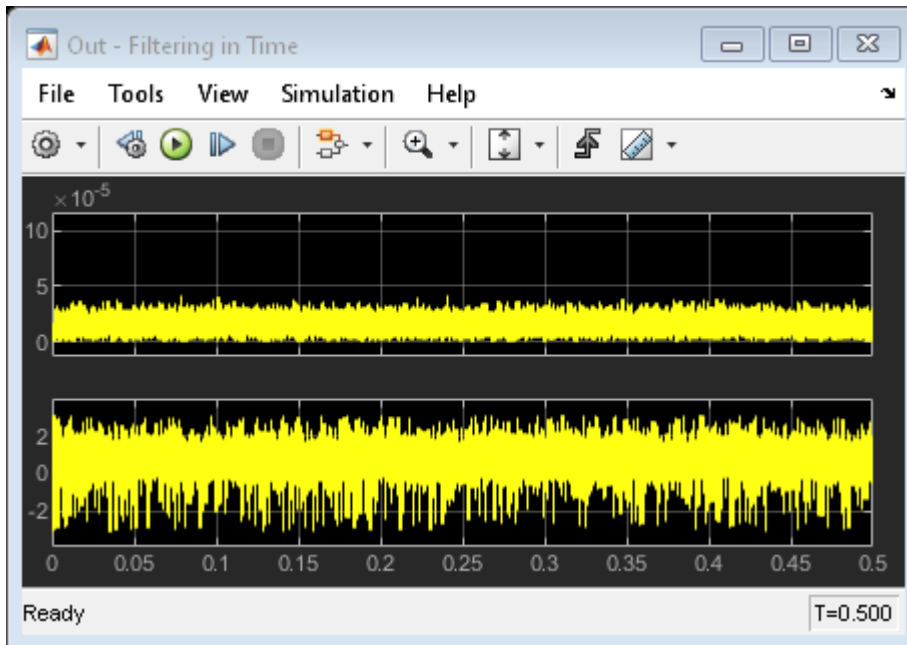
The scope output shows the results for a 0.5 second simulation, while received pulses indicate the presence of the target.

Effect of Antenna Gain/Direction

Open the 'Ideal Antenna' block and change the transmit gain to 10 dB. The target will no longer receive the signal from the main beam of the transmit antenna.

To run the example under this scenario:

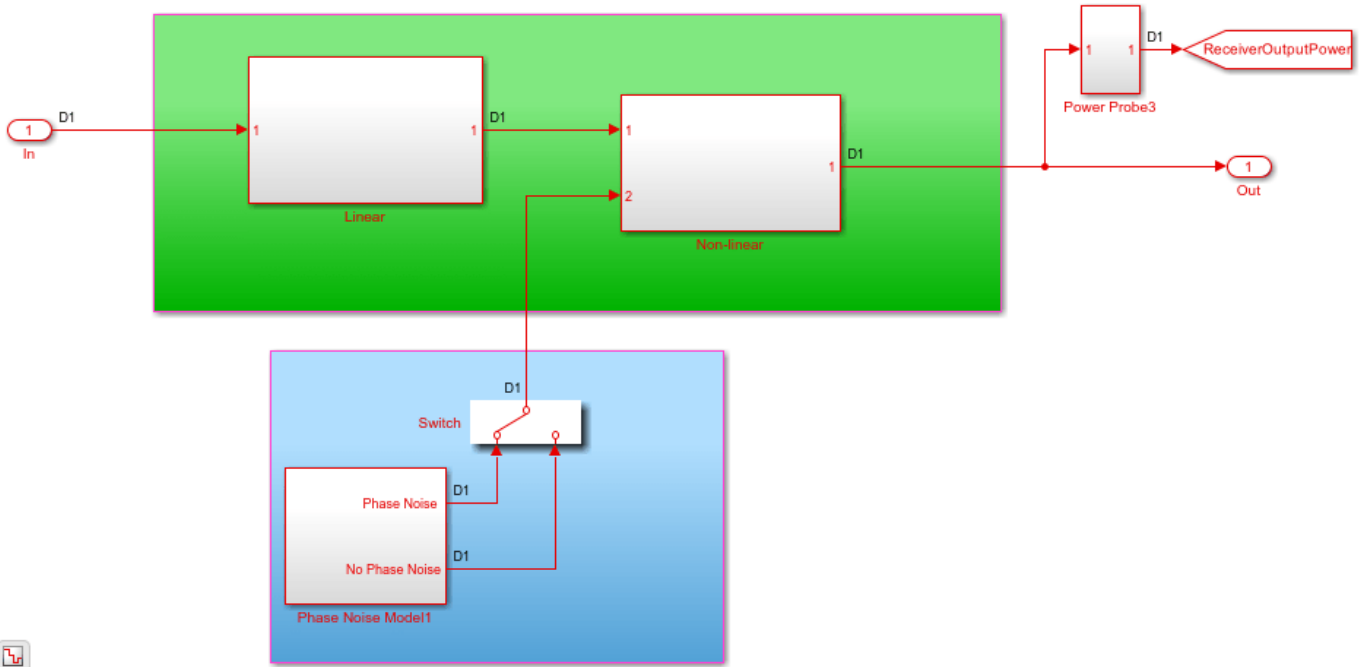
- 1 Select **Simulation > Run**



The effect of the change in antenna gain is observed in the scope. Notice that the pulses are now buried in the noise, rendering the object electromagnetically invisible.

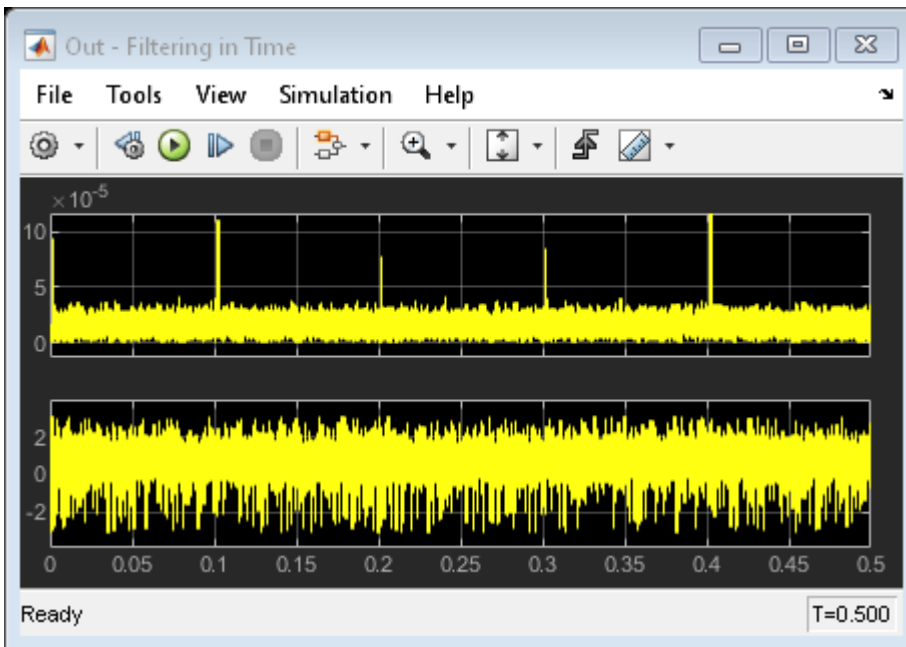
Phase Noise Enabled On Receiver LO

Open the Receiver Front-End subsystem, and use the manual switch to include the phase noise model for the Local Oscillator.



Effect of Phase Noise

- 1 Double click on the 'Ideal Antenna' block and change the transmit gain back to 40 dB.
- 2 Select **Simulation > Run**



The effect of the phase noise from the Local oscillator is observed in the varying strength of the detected pulses. This varying pulse strength can have an impact on the probability of detection and will result in the target being detected only at certain times.

See Also

Related Examples

- “Modeling RF Front End in Radar System Simulation” (Phased Array System Toolbox)
- “Radar Tracking System” on page 8-132

Only one frame is generated. This frame will then be repeated a number of times to perform the EVM measurements.

```
% Configuration TS 36.101 25 REs (5 MHz), 64-QAM, full allocation
rmc = lteRMCDL('R.6');
rmc.OCNGPDSCHEnable = 'On';

% Create eNodeB transmission with fixed PDSCH data
rng(2) % Fixed random seed (arbitrary)
data = randi([0 1],sum(rmc.PDSCH.TrBlkSizes),1);

% Generate 1 frame, to be repeated to simulate a total of N frames
[tx,~,info] = lteRMCDLTool(rmc,data); % 1 frame

% Calculate the sampling period and the length of the frame
SamplePeriod = 1/info.SamplingRate;
FrameLength = length(tx);
```

Initialize Simulation Components

This section initializes some of the simulation components:

- Number of frames: this is the number of times the generated frame is repeated
- Preallocate result vectors

```
% Number of simulation frames N >= 1
N = 3;

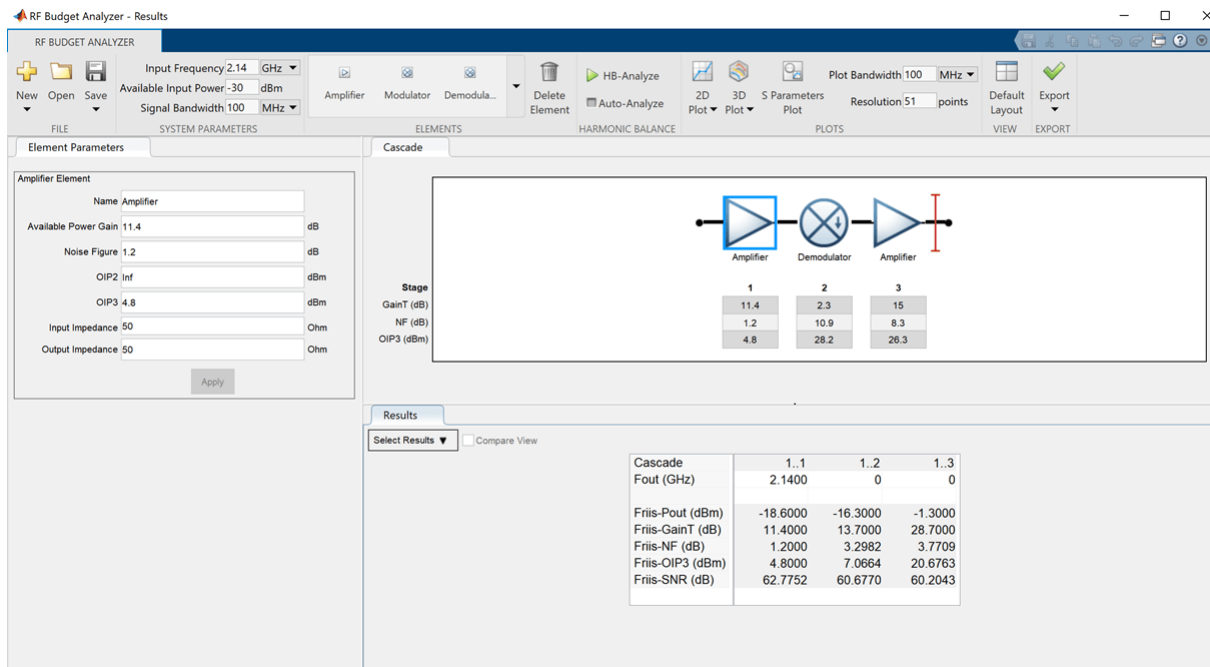
% Preallocate vectors for results for N-1 frames
% Note: EVM is not measured in the first frame to avoid transient effects
evmpeak = zeros(N,1);
evmrms = zeros(N,1);
```

Design RF Receiver

The initial design of the RF receiver is done using the **RF Budget Analyzer** app. The receiver consists of a LNA, a direct conversion demodulator, and a final amplifier. All stages include noise and non-linearity.

```
load rfb.mat
```

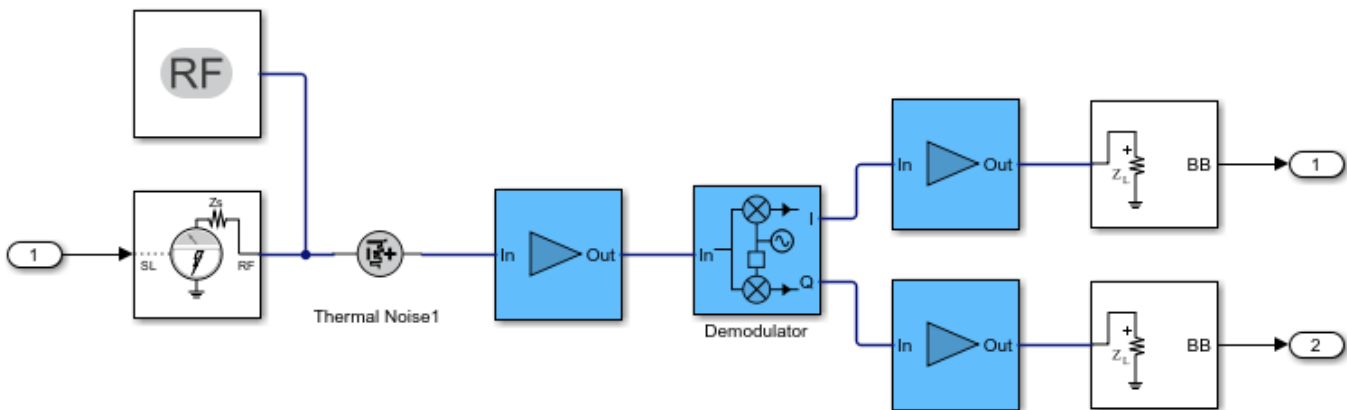
Type `show(rfb)` to display the initial design of the RF receiver in the **RF Budget Analyzer** app.



Create RF model for simulation

From the RF budget object you can automatically create a model that can be used for circuit envelope simulation.

```
rfx = rfsystem(rfb);
rfx.SampleTime = SamplePeriod;
open_system(rfx)
```



Extend the model of the RF receiver

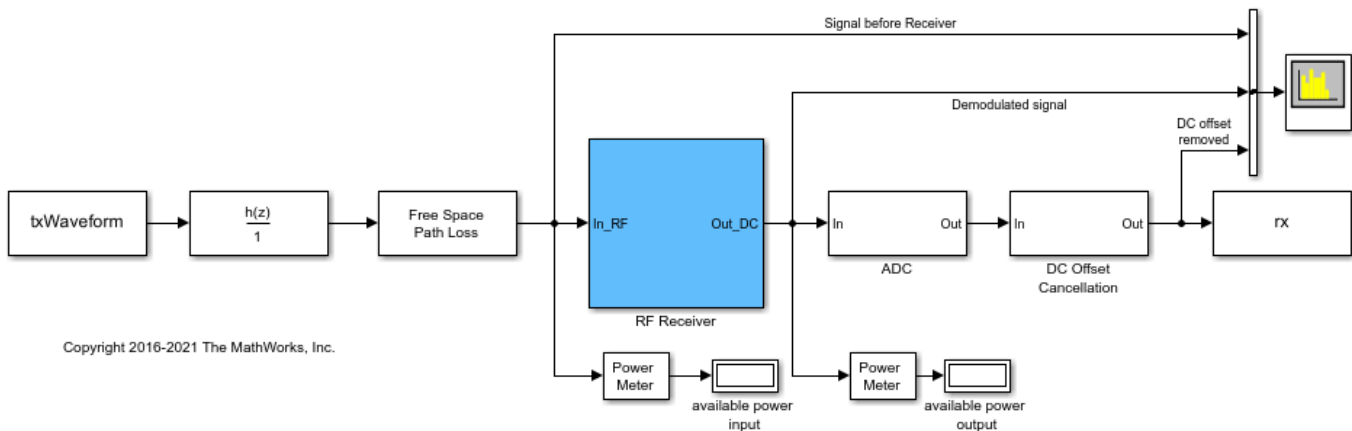
You can modify the model created in the previous section to include additional RF impairments and components. You can modify the created RF Blockset model as long as you do not change the input / output ports. This section loads a modified Simulink model that performs the following functions:

- Channel model: includes free space path loss

- RF receiver: includes direct conversion demodulator
- ADC and DC offset cancellation

You can open and inspect the modified model.

```
model = 'simrfv2_lte_receiver';
open_system(model)
```



Simulate Frames

This section simulates the specified number of frames. The RF system object simulates the Circuit Envelope model in Accelerator mode to decrease run time. After processing the first frame with the Simulink model, its state is preserved and automatically passed to the successive frame.

The output of the Simulink model is stored in the variable `rx`, which is available in the workspace. Any delays introduced to this signal are removed after performing synchronization. The EVM is measured on the resulting waveform.

```
load rfs.mat
EVMalg.EnablePlotting = 'Off';
cec.PilotAverage = 'TestEVM';

for n = 1:N
    [I,Q] = rfs(tx);
    rx = complex(I,Q);

    % Synchronize to received waveform
    if n == 1
        Offset = lteDLFrameOffset(rmc,squeeze(rx),'TestEVM');
    end

    % Compute and display EVM measurements
    evmmeas = simrfv2_lte_receiver_evm_cal(rmc,cec,squeeze(rx),EVMalg);
    evmpeak(n) = evmmeas.Peak;
    evmrms(n) = evmmeas.RMS;
end
```

```
Low edge EVM, subframe 0: 2.884%
High edge EVM, subframe 0: 2.885%
Low edge EVM, subframe 1: 2.845%
```

High edge EVM, subframe 1: 2.838%
Low edge EVM, subframe 2: 2.800%
High edge EVM, subframe 2: 2.804%
Low edge EVM, subframe 3: 2.764%
High edge EVM, subframe 3: 2.764%
Low edge EVM, subframe 4: 2.773%
High edge EVM, subframe 4: 2.765%
Low edge EVM, subframe 6: 2.845%
High edge EVM, subframe 6: 2.838%
Low edge EVM, subframe 7: 2.832%
High edge EVM, subframe 7: 2.837%
Low edge EVM, subframe 8: 2.778%
High edge EVM, subframe 8: 2.772%
Low edge EVM, subframe 9: 2.883%
High edge EVM, subframe 9: 2.868%
Averaged low edge EVM, frame 0: 2.822%
Averaged high edge EVM, frame 0: 2.818%
Averaged EVM frame 0: 2.822%
Averaged overall EVM: 2.822%
Low edge EVM, subframe 0: 3.026%
High edge EVM, subframe 0: 3.018%
Low edge EVM, subframe 1: 2.944%
High edge EVM, subframe 1: 2.935%
Low edge EVM, subframe 2: 2.772%
High edge EVM, subframe 2: 2.760%
Low edge EVM, subframe 3: 2.795%
High edge EVM, subframe 3: 2.791%
Low edge EVM, subframe 4: 2.907%
High edge EVM, subframe 4: 2.902%
Low edge EVM, subframe 6: 2.833%
High edge EVM, subframe 6: 2.815%
Low edge EVM, subframe 7: 2.792%
High edge EVM, subframe 7: 2.794%
Low edge EVM, subframe 8: 2.786%
High edge EVM, subframe 8: 2.787%
Low edge EVM, subframe 9: 2.806%
High edge EVM, subframe 9: 2.807%
Averaged low edge EVM, frame 0: 2.849%
Averaged high edge EVM, frame 0: 2.844%
Averaged EVM frame 0: 2.849%
Averaged overall EVM: 2.849%
Low edge EVM, subframe 0: 2.953%
High edge EVM, subframe 0: 2.961%
Low edge EVM, subframe 1: 2.899%
High edge EVM, subframe 1: 2.898%
Low edge EVM, subframe 2: 2.776%
High edge EVM, subframe 2: 2.780%
Low edge EVM, subframe 3: 2.840%
High edge EVM, subframe 3: 2.858%
Low edge EVM, subframe 4: 2.853%
High edge EVM, subframe 4: 2.859%
Low edge EVM, subframe 6: 2.896%
High edge EVM, subframe 6: 2.878%
Low edge EVM, subframe 7: 2.804%
High edge EVM, subframe 7: 2.801%
Low edge EVM, subframe 8: 2.777%
High edge EVM, subframe 8: 2.778%
Low edge EVM, subframe 9: 2.904%

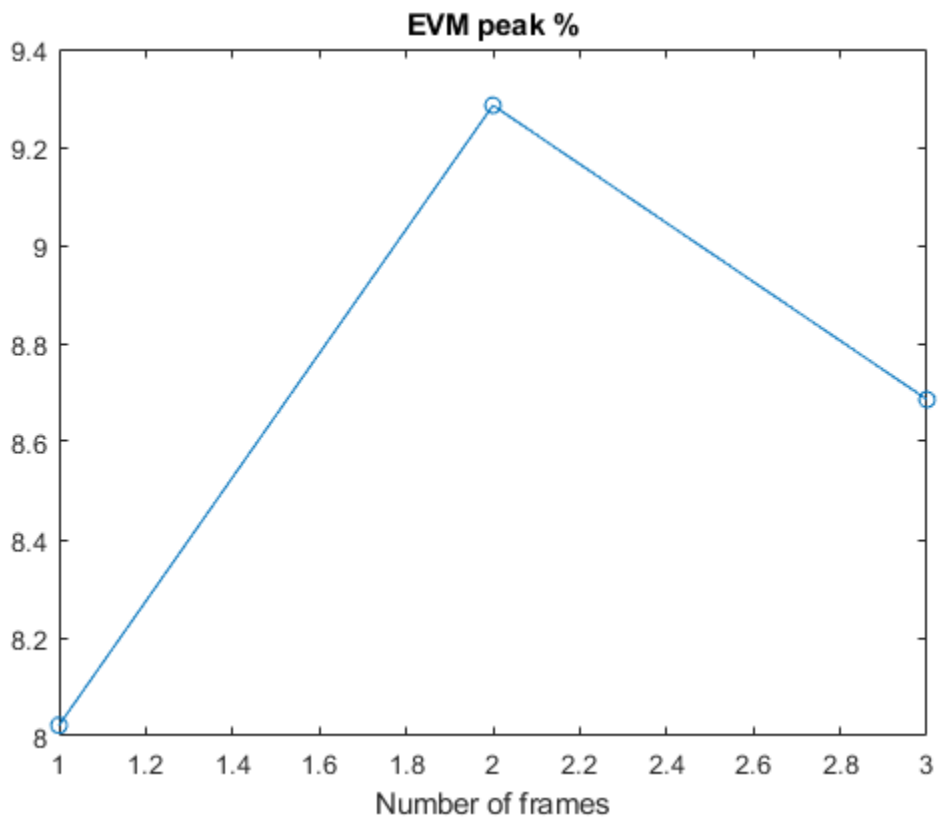
High edge EVM, subframe 9: 2.896%
Averaged low edge EVM, frame 0: 2.855%
Averaged high edge EVM, frame 0: 2.856%
Averaged EVM frame 0: 2.856%
Averaged overall EVM: 2.856%

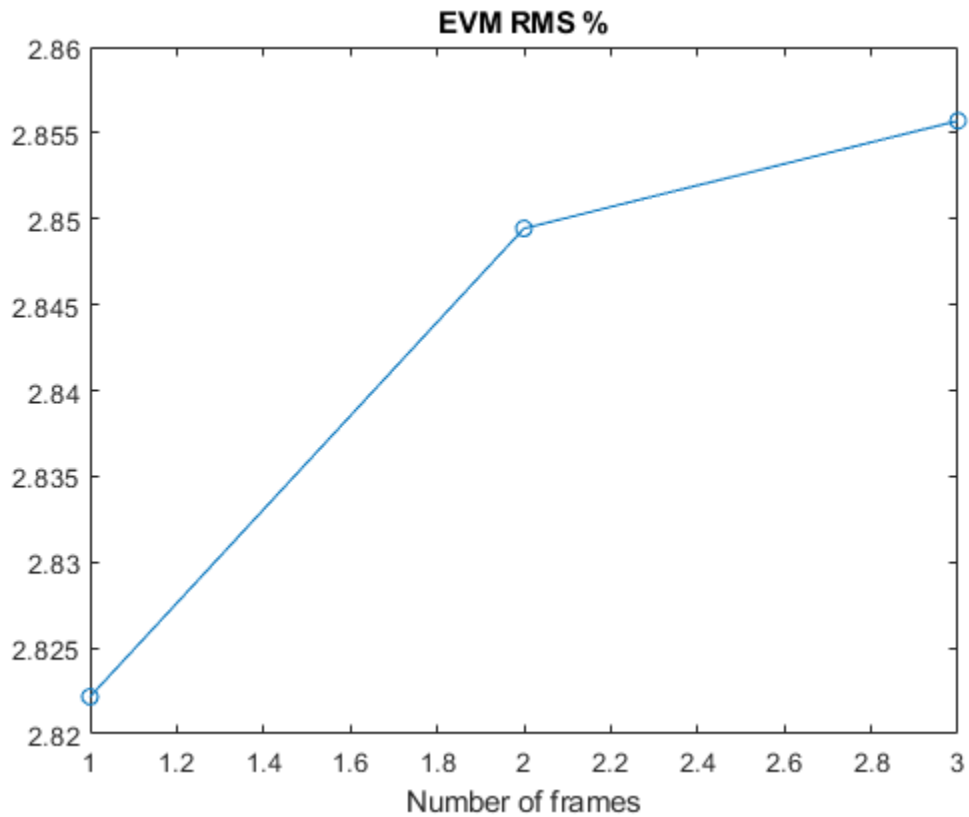
Visualize Measured EVM

This section plots the measured peak and RMS EVM for each simulated frame.

```
figure
plot((1:N),100*evmpeak,'o-')
title('EVM peak %')
xlabel('Number of frames')
```

```
figure
plot((1:N),100*evmrms,'o-')
title('EVM RMS %')
xlabel('Number of frames')
```





Cleaning Up

Close the Simulink model and remove the generated files.

```
release(rfs)
close_system(rfs,0)
```

Appendix

This example uses the following helper function:

- `simrfv2_lte_receiver_evm_cal.m`

Selected Bibliography

- 1 3GPP TS 36.101 "User Equipment (UE) radio transmission and reception"

See Also

Amplifier | VGA | Configuration | Inport | Outport

Related Examples

- "Carrier to Interference Performance of Weaver Receiver" on page 7-48

Create Custom RF Blockset Models

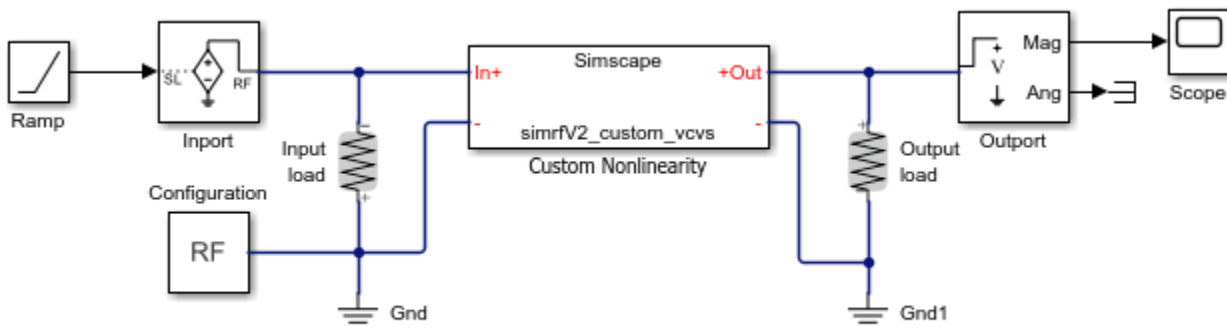
This example shows how to write your own RF Blockset™ Circuit Envelope model in Simscape® language for complex baseband simulation. An RF Circuit Envelope complex baseband signal resides on a carrier with specified frequency. This baseband signal will modulate with other signals when the system is nonlinear. The example nonlinearity is implemented with a Simscape Component block and includes a Simscape ssc-file to describe the nonlinear voltage polynomial.

System Architecture

The system consists of:

- An input voltage signal, linearly increasing in time and generated with a Simulink Ramp block.
- An RF Blockset Inport block to specify the Carrier frequencies (**Input_Freq**) of the input voltage signal. This setup allows observation of the system nonlinear behavior for different input settings.
- A custom nonlinear voltage amplifier (polynomial voltage controlled voltage source), modeled with a Simscape Component block. The device equations are written in passband (time) domain and assume instantaneous voltage $V(t)$ and current $I(t)$ values. These equations are interpreted by RF Blockset envelope solver in both passband and baseband domains (zero and nonzero carrier frequencies).
- An Output block to specify output Carrier frequencies (**Output_Freqs**). The output carrier frequencies are higher order harmonics (integer multiples) of the Inport frequency resulting from the amplifier nonlinearity.
- A Scope to display the magnitudes of the output voltages at **Output_Freqs** frequencies as specified in the Output block.
- Load resistors and ground nodes needed to make the circuit electrically sound. By construction, the resistor values do not affect the output voltage.
- A Configuration block to control the system carrier frequencies required for accurate simulation and other simulation properties.

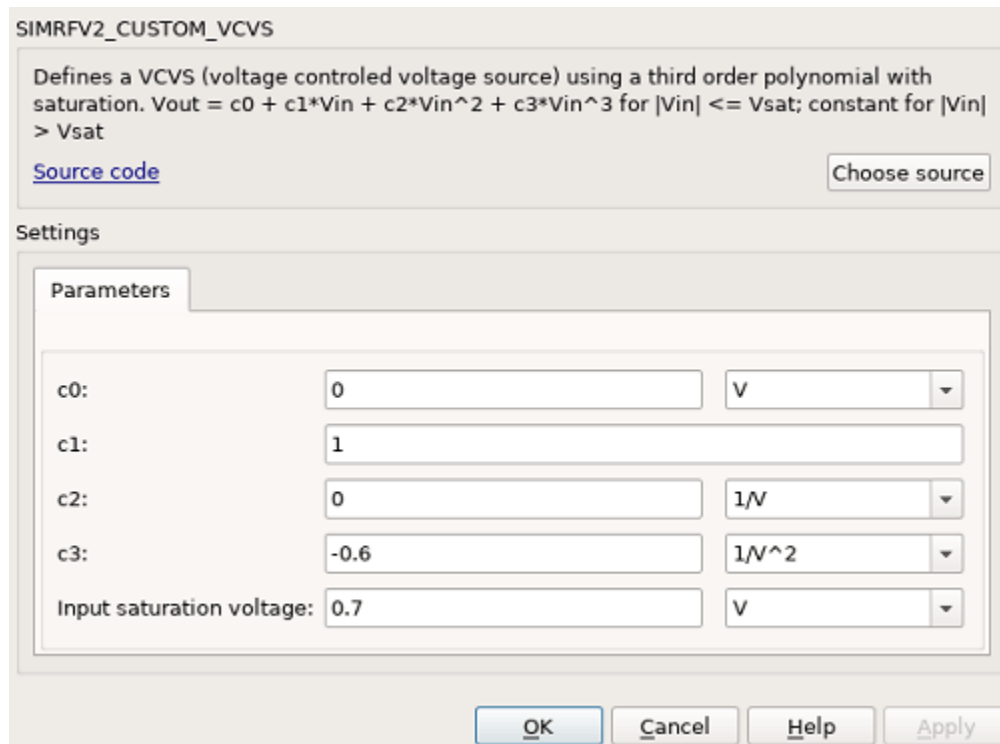
```
model = 'simrfv2_custom_polynomial';
open_system(model);
```



Copyright 2013-2018 The MathWorks, Inc.

Examine Model

Double-click the "Custom Nonlinearity" block or type `open_system([model '/Custom Nonlinearity'])` in the command window to open the Custom Nonlinearity block mask.



The file `simrfv2_custom_vcvss.ssc` describes the custom device. View the source code by clicking the block mask "Source code" link or typing `edit simrfv2_custom_vcvss` at the command prompt.

Copy the file `simrfv2_custom_vcvss.ssc` to a directory where you have write permission to rename and modify the file. Click the block mask "Choose source" button to replace the current device implementation with yours. Use the mask `Help` button for additional information.

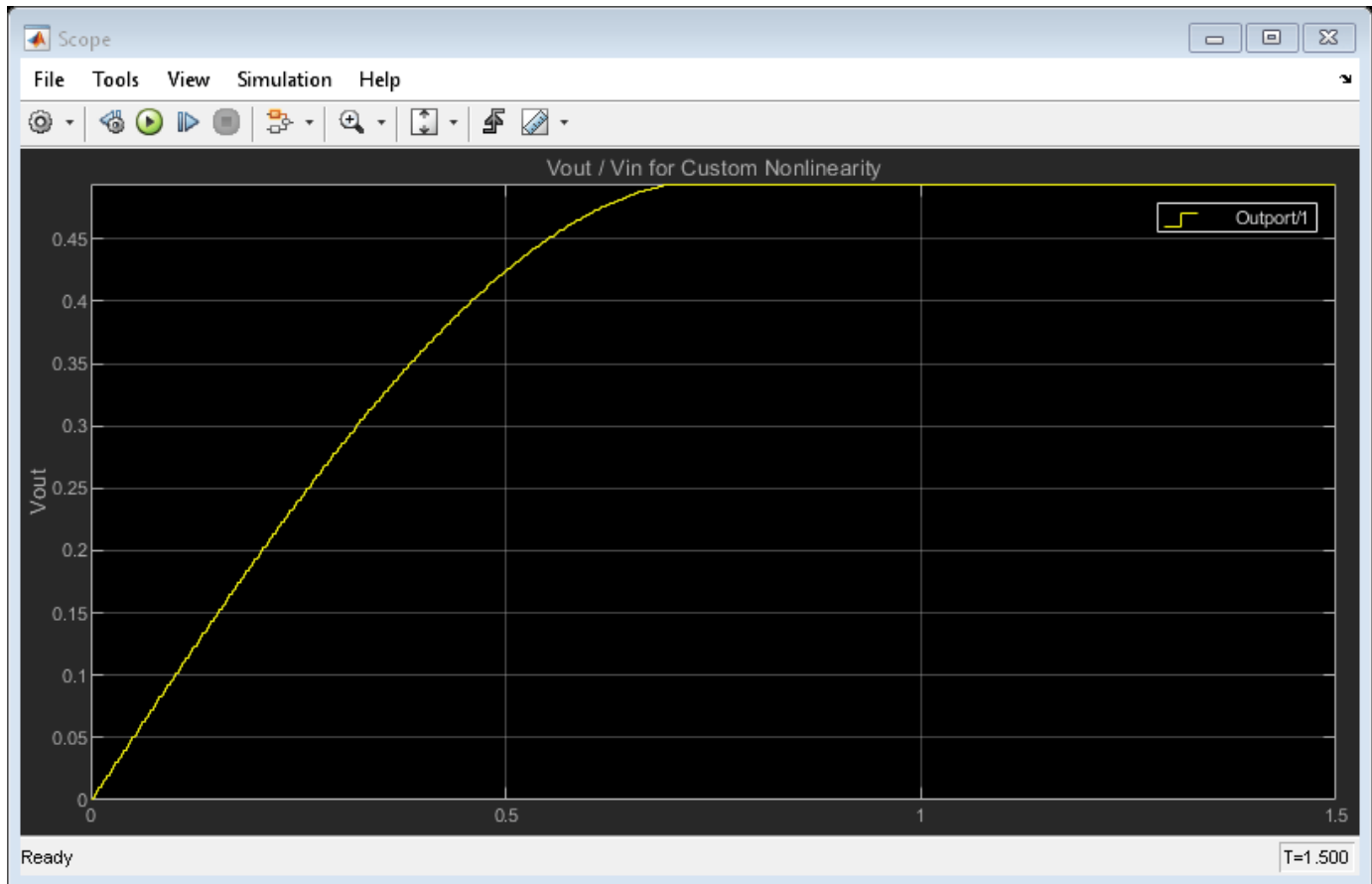
The above method uses the Simscape Component block from the Simscape Utilities library to avoid the library build process. For more information, see "Custom Components".

Run Model Using Default Settings

For this example, default input and output frequencies are set to 0 and the result is a passband simulation. The input voltage magnitude is linearly increasing in time, $V_{in}(t) = t$, and the custom nonlinearity relationship $V_{out}(V_{in})$ is shown in the scope.

The model is simulated after entering the following into the command window

```
sim(model);
```



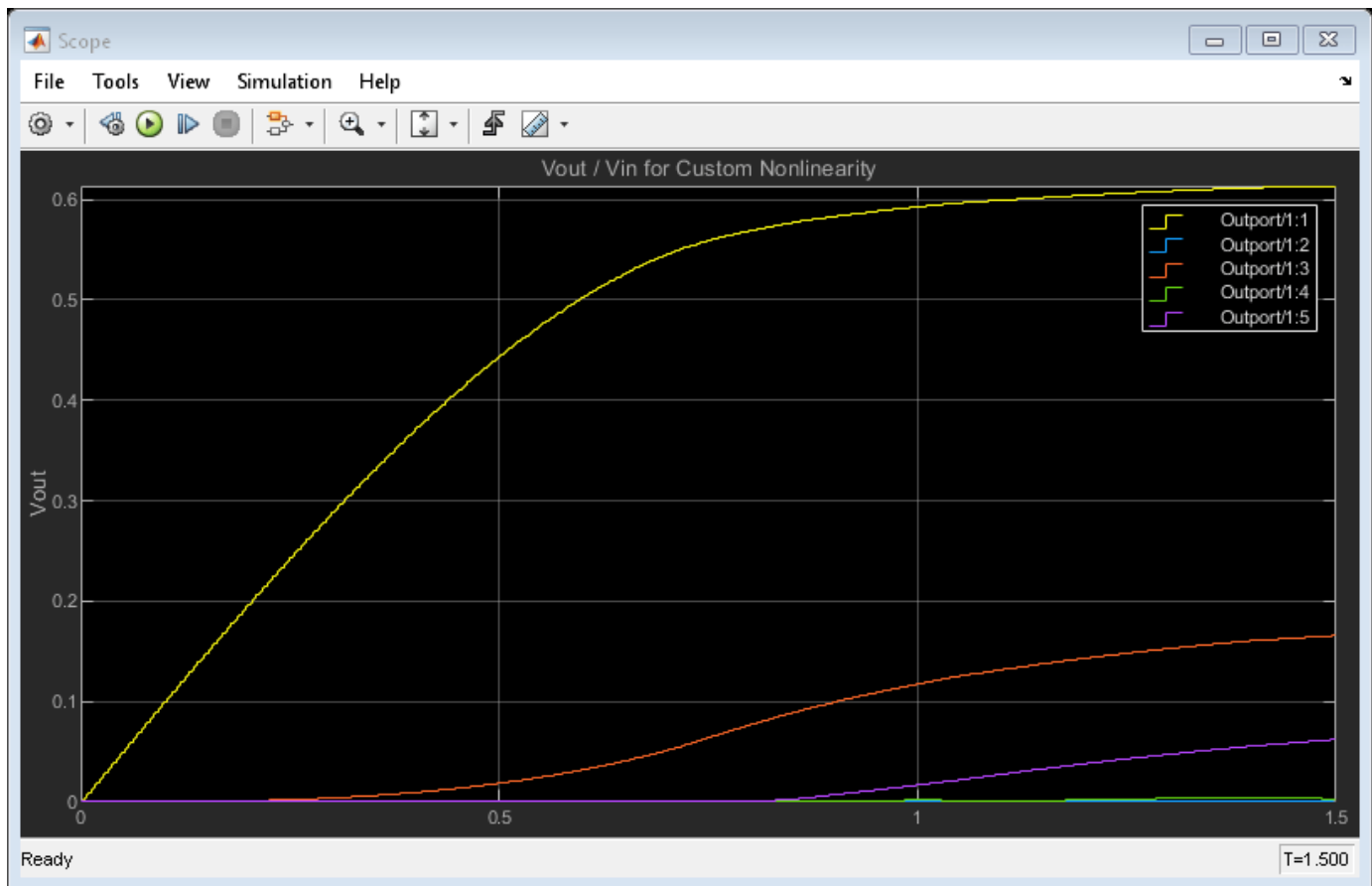
Observe the response produced by the cubic polynomial specified in the 'Custom Nonlinearity' model. The saturated output voltage occurs at time 0.7 seconds and corresponds to the input voltage of 0.7 V.

Run Model with Non-zero Input Carrier

Set the input carrier frequency to 1 GHz and the output frequencies to the first five harmonics of the input. For non-zero carrier input frequency, RF Blockset interprets the input as a complex baseband signal. This complex baseband signal only has a in-phase part specified.

Type the following at the Command Prompt:

```
Input_Freq = 1e9;
Output_Freqs = (1:5)*Input_Freq;
sim(model);
```



Since the coefficients c_0 and c_2 are zero, the output has only odd harmonics (1 GHz, 3 GHz and 5 GHz) until the output voltage reaches saturation. Other harmonics are introduced for large values of input voltage because of saturation effects.

The relationship between the output curves, polynomial coefficients and IP2/IP3/P1db coefficients is well-studied in the literature [1,2].

Conclusion

An RF Blockset model can be written as a time-domain electrical model in the Simscape language. The model equation can include many types of characteristics, such as derivatives and history (not shown in this example). As with any other model description language, the modeler is responsible for the validity of the model:

- The equations are consistent.
- The equations cannot be degenerate, unstable, or discontinuous. Avoid negative resistances, large nonlinearities and sharp transitions.
- The model does not produce convergence errors during the simulation.

Bibliography

- 1 Kundert, Ken. "Accurate and Rapid Measurement of IP2 and IP3." *The Designers Guide Community*, Version 1b, May 22, 2002.

2 Chen, Jesse. "Modeling RF systems." *The Designers Guide Community*, Version 1, 6 March 2005.

```
bdclose(model)
```

See Also

Configuration | Inport | Outport

Related Examples

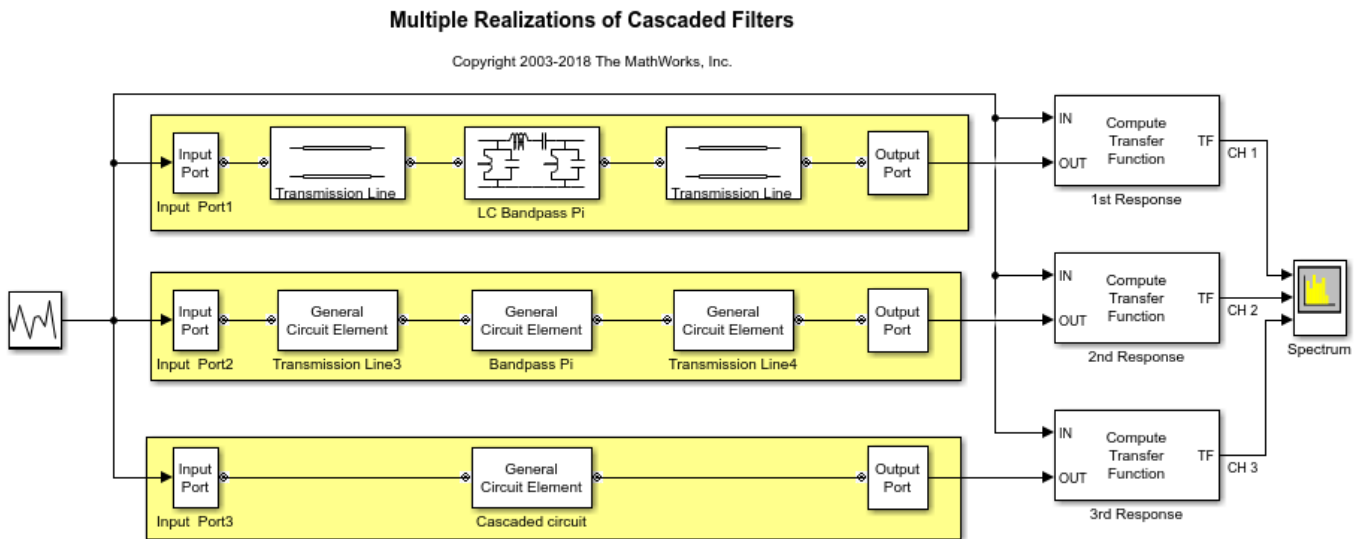
- "Getting Started with RF Modeling" on page 8-2

Multiple Realizations of Cascaded Filters

This model shows three different ways to use RF Blockset™ Equivalent Baseband library blocks and RF Toolbox™ objects to implement filters.

Example Model

Example model of multiple realizations of cascaded filters:



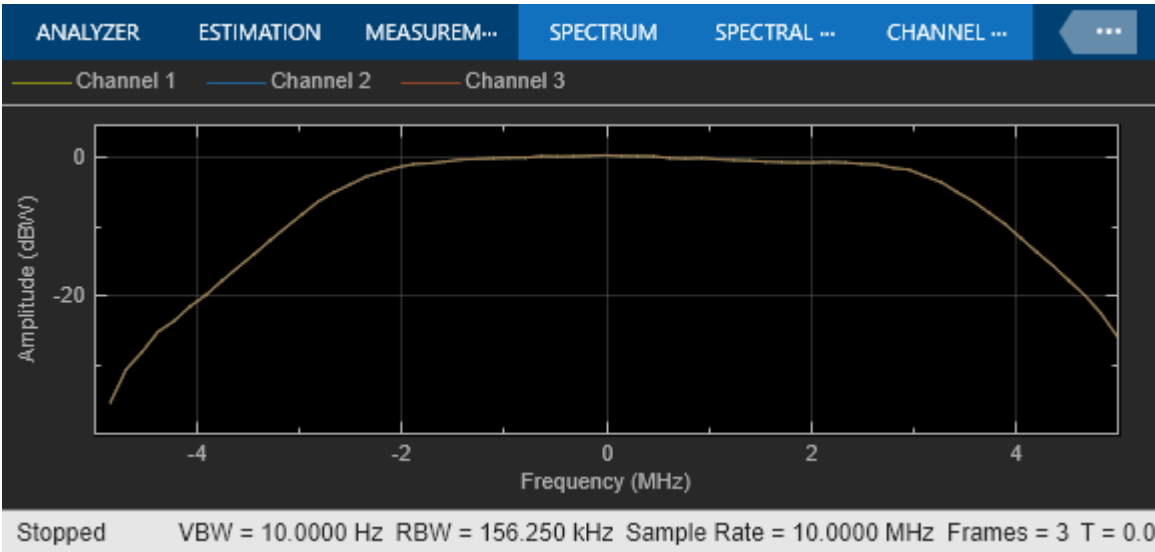
The first method creates a filter using transmission line blocks from the Transmission Lines sublibrary and a LC Bandpass Filter block from the Ladder Filters sublibrary.

The second method creates a filter using three General Circuit Element blocks and three RF circuit objects.

The third method creates a filter using a General Circuit Element block and an RF circuit object, which represents a cascade of three RF circuit objects.

Simulation Results

The spectrum scope shows the frequency responses for the three filter implementations from 70 MHz (-5.0 MHz) to 80 MHz (5.0 MHz).



See Also

LC Bandpass Pi | Transmission Line (Equivalent Baseband)

Related Topic

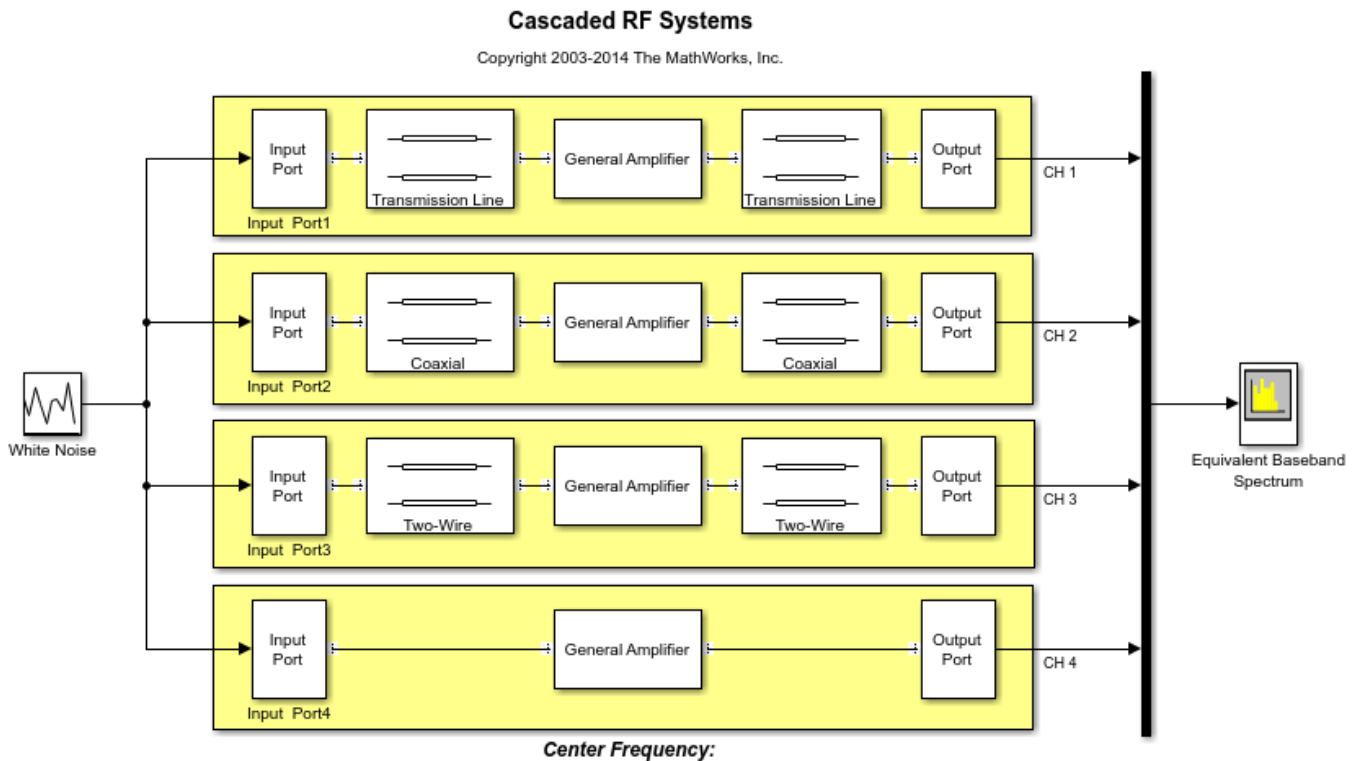
“Cascaded RF Systems” on page 8-106

Cascaded RF Systems

This model shows how to use blocks from the RF Blockset™ Equivalent Baseband library to build cascaded RF systems.

Example Model

Fig. 1 Example model of cascaded RF systems:

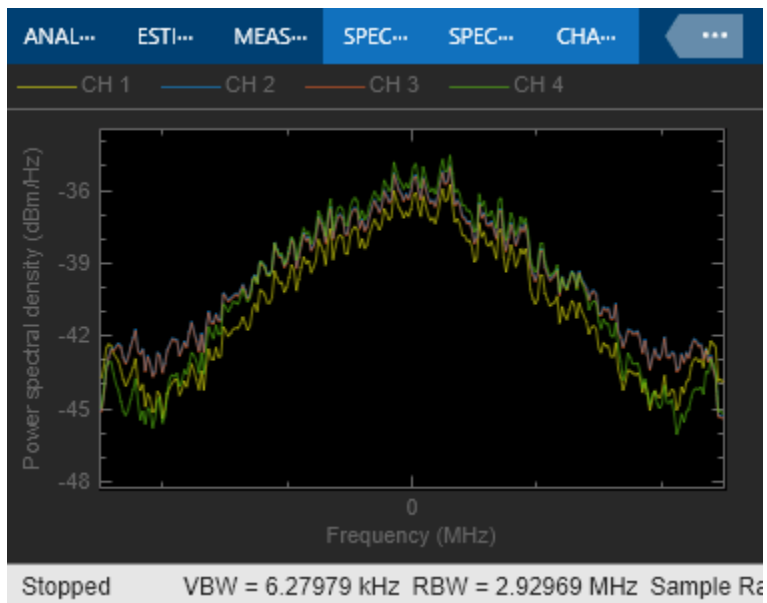


Simulation Results

In the first three systems, an amplifier with S-parameters taken from the default.s2p data file is cascaded with different types of transmission lines. The fourth system represents the amplifier itself.

The following figure shows the equivalent baseband frequency response of four RF systems ranging from 1.85 GHz (-250 MHz) to 2.35 GHz (250 MHz).

Fig. 2 Frequency response:



To see the frequency response centered at another RF frequency, change the Center frequency parameter in the Input Port block.

See Also

Equivalent Baseband Transmission Line | LC Bandpass Pi

More About

- “Multiple Realizations of Cascaded Filters” on page 8-104

Power in Simulink Sources and Signals

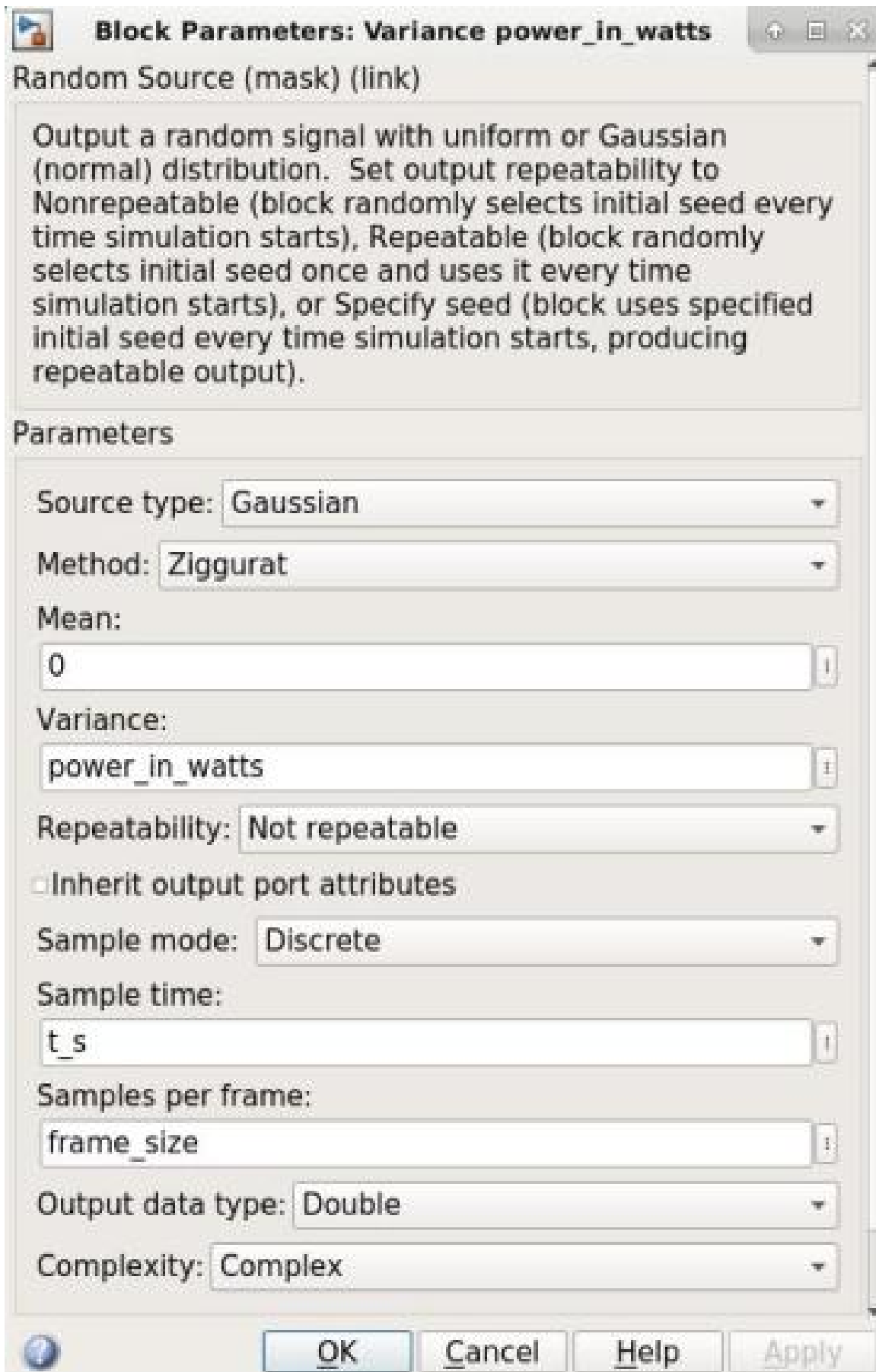
This example shows how to use the Input Port and Output Port blocks of the RF Blockset™ Equivalent Baseband library to convert between dimensionless Simulink® signals and equivalent-baseband signals.

In general, signals in Simulink are dimensionless, so their amplitudes do not correspond to a particular voltage or power. However, in an RF system, power is a quantity of interest. When you use blocks from the RF Blockset Equivalent Baseband library in a Simulink model, you must specify how the software interprets the Simulink signals that exist outside the boundaries of the Input Port and Output Port blocks. RF Blockset Equivalent Baseband software provides two options to interpret the Simulink signal: power wave or voltage. The amplitude of a source in Simulink determines the signal power level and affects the signal power and power spectrum.

All the models used in this example interpret the Simulink signal as a power wave with dimensions of $W^{0.5}$. This means for an RF system, the source signal generated by regular Simulink blocks is treated as the incident power wave to the RF system, and the RF output signal is the transmitted power wave of the RF system. If you choose to interpret the Simulink signal as a voltage, you need to modify the models by considering the impedance effects when you calculate the powers. For more details, see “Convert to and from Simulink Signals” on page A-22.

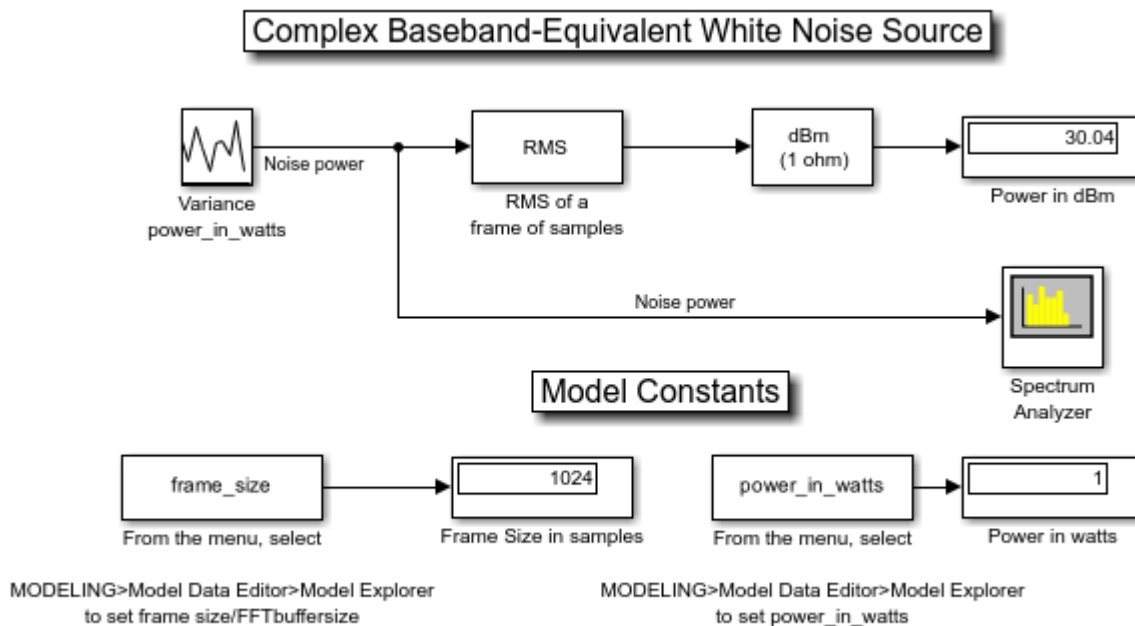
White Noise Source

This part of the example shows how to create a complex baseband-equivalent White Noise Source. This type of source is useful, for example, as a stimulus for visualizing the frequency response of an RF system. Use a Random Source block from the DSP System Toolbox™ Sources sublibrary to create this source. In the Random Source block dialog box, set the **Complexity** parameter to **Complex** and in the **Variance** parameter enter the desired noise power in watts using the expression `power_in_watts`.

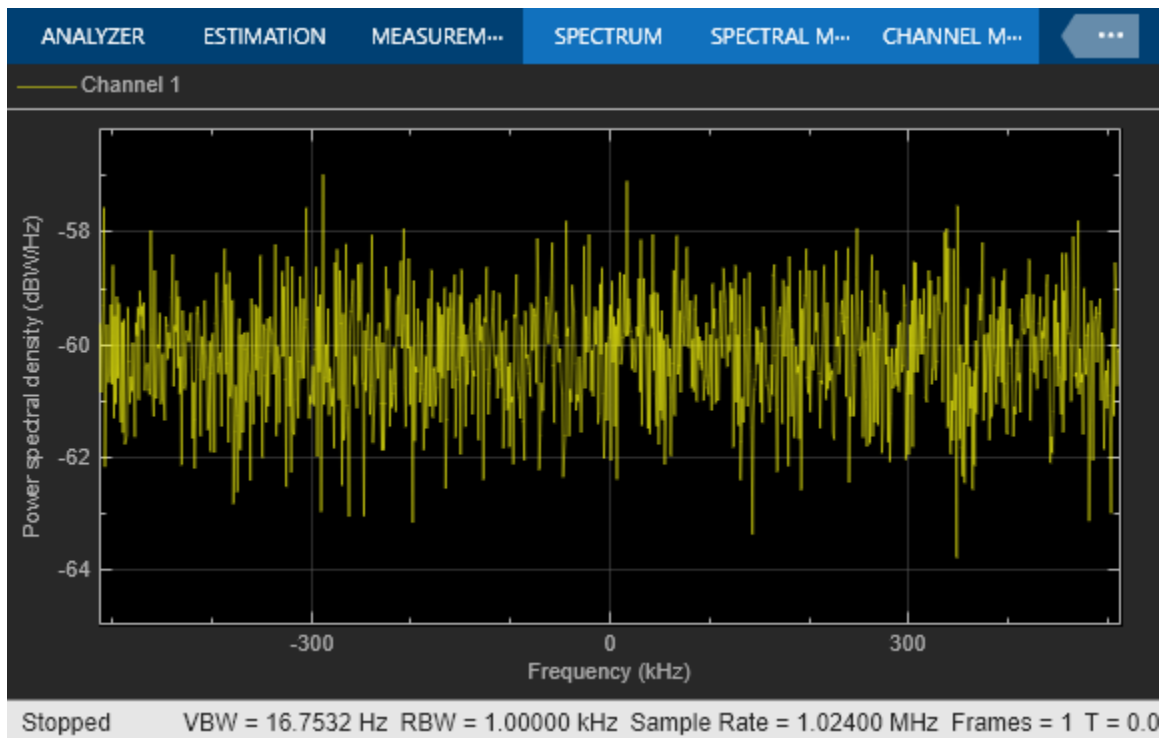


To calculate a signal power in dBm, use an RMS block (from the DSP System Toolbox Statistics sublibrary), followed by a dB Conversion block (from the DSP System Toolbox Math Functions/Math Operations sublibrary). In the dB Conversion block dialog box, set **Convert to** parameter to dBm , **Input Signal** parameter to Amplitude , and **Load resistance (ohms)** parameter to 1 .

To display the power spectrum of a signal, use the Spectrum Analyzer block (from the RF Blockset Circuit Envelope Utilities sublibrary). To show the Spectrum Settings, in the Spectrum Analyzer menu, select **View > Spectrum Settings** or use the far left button in the toolbar. **Two-sided spectrum** is checked by default in the trace options. This is the desired frequency range because a complex baseband-equivalent representation translates the carrier frequency to zero hertz. The real-world frequencies above and below the carrier (i.e. higher and lower sidebands) are represented as positive and negative frequencies, respectively. A few of the **Spectrum Settings** options (**Type, Window, Units, Averaging method, Averages**) have been changed from their default.



Copyright 2003-2020 The MathWorks, Inc.



In addition, note that the selected Spectrum Scope Window Type can affect how the power is distributed among the channels closest to the actual frequency. For example, if a pure sine wave falls between two channels, you may need to sum the power in one or two channels either side of the actual frequency to determine the exact total power.

Complex Sine Source

The next model, of a Complex Sine Wave, shows how to use power to set the amplitude of a complex sine wave source block for an RF system. Complex sine wave sources are often used in baseband-equivalent Simulink models. These sources have the following time-domain output:

$$\text{signal}(t) = \text{amplitude} * (\cos(2*\pi*f*t+\text{phi})+j*\sin(2*\pi*f*t+\text{phi}))$$

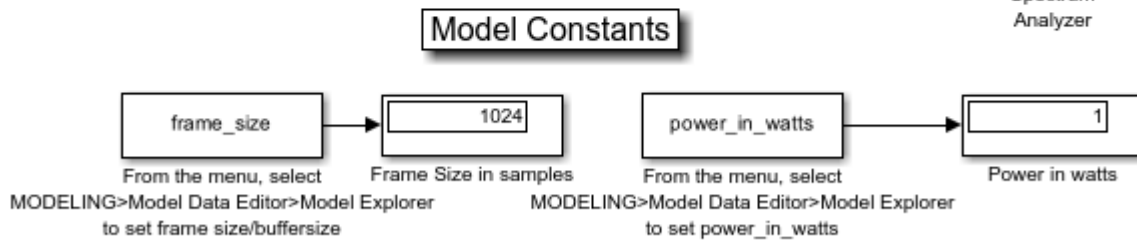
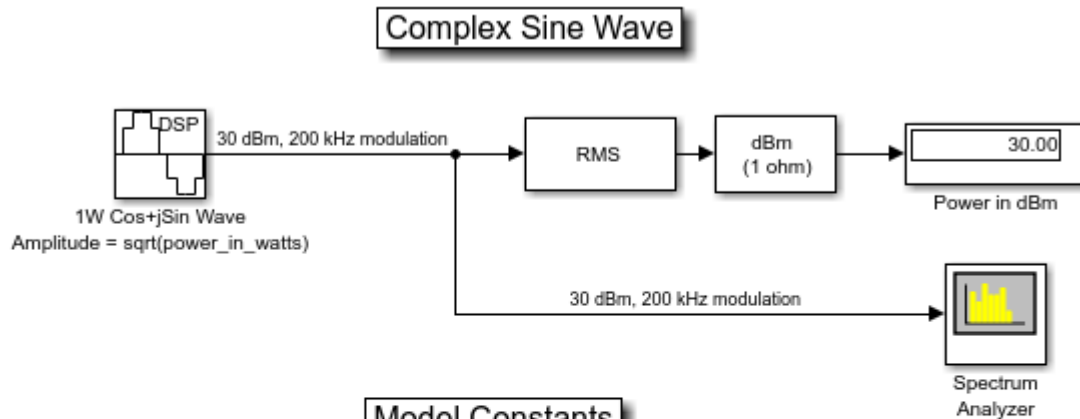
The mean square power of the output, `signal`, is `amplitude^2`.

By contrast, the time-domain output of a real sine wave source is:

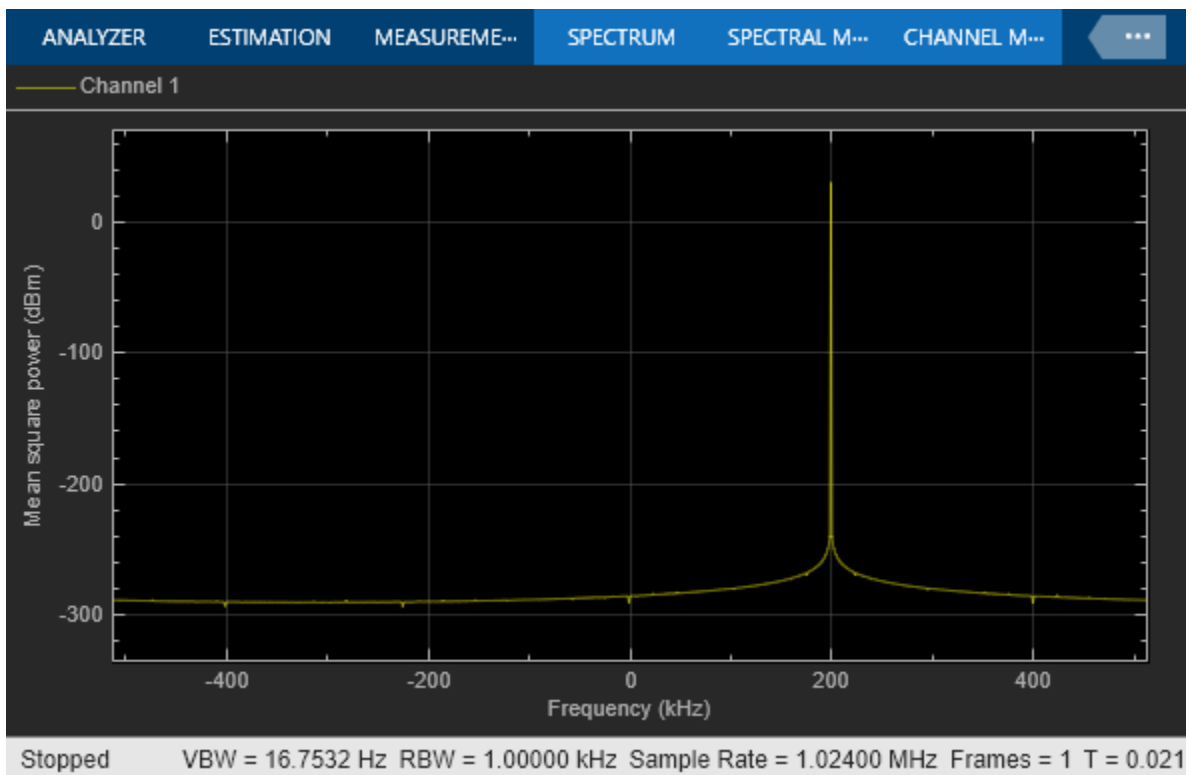
$$\text{signal2}(t) = \text{amplitude} * \sin(2*\pi*f*t+\text{phi})$$

where the mean square power of `signal2` is `amplitude^2/2`, half that of a complex sine wave with the same amplitude.

Use the Sine Wave block to create a complex sine source. In the block dialog box, set the **Output complexity** parameter to **Complex** and the **Amplitude** parameter to `sqrt(power_in_watts)`. By default, the Spectrum Analyzer displays power spectral density normalized to the unit sampling frequency in units of dBm/Hertz. For this section, the Spectrum Scope displays the tone as a positive frequency (upper side band).

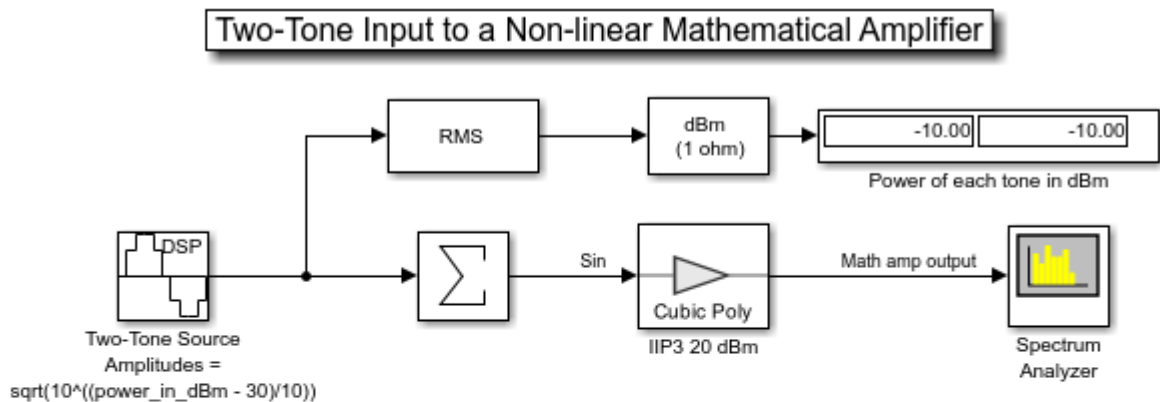


Copyright 2003-2020 The MathWorks, Inc.



Two-Tone Input to Idealized Baseband Nonlinear Amplifier

The third model, of a Two-Tone Input to an Idealized Baseband Nonlinear Amplifier, shows how the Amplifier block in the RF Blockset Idealized Baseband library affects the signal. In the Amplifier block dialog box set the **IIP3 (dBm)** parameter to 20 dBm. In the Sine Wave block dialog box, set **Amplitude** parameter to $\sqrt{10^{((\text{power_in_dBm} - 30)/10)}}$. Setting $\text{power_in_dBm} = -10$ in the model workspace results in -10 dBm per tone. Note that we must use a Matrix Sum block with the **Sum along** parameter set to "Rows" after the source block to sum the two-channel output of the source. Without the Row Sum, a two-channel signal would be created, all blocks downstream would have two independent channels, and no mixing would occur.



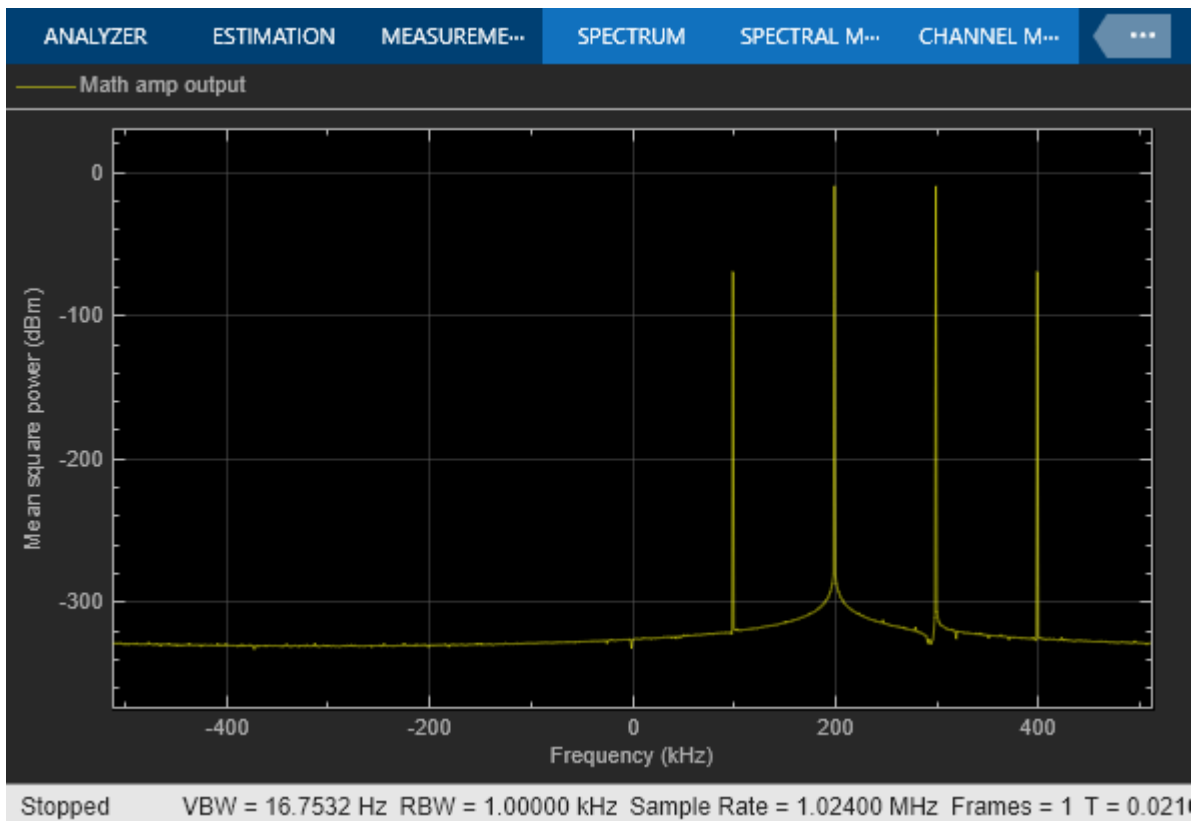
Model Constants



MODELING>Model Data Editor>Model Explorer
to set frame size/buffer size

MODELING>Model Data Editor>Model Explorer
to set the tone power in dBm

Copyright 2003-2020 The MathWorks, Inc.



The Spectrum Scope displays the power level in each intermodulation tone. The power level of each is:

$$-10\text{dBm} - 2*(20\text{dBm} - -10\text{dBm}) = -70\text{dBm}.$$

Two-Tone Input to Equivalent Baseband Nonlinear Amplifier

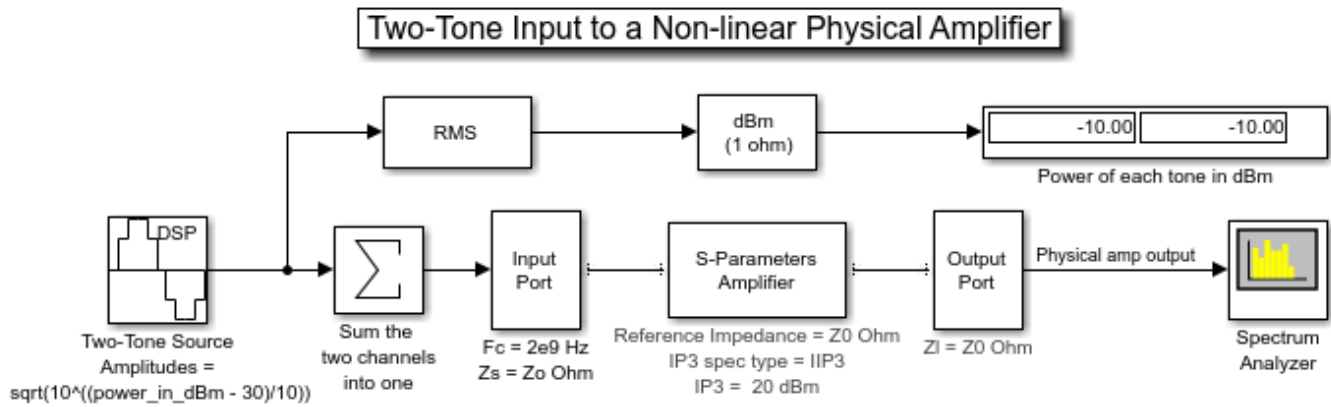
Like the third model, the fourth model, of a Two-Tone Input to an Equivalent Baseband Nonlinear Amplifier, shows how an Amplifier block affects the signal. However, this time we use the S-Parameters Amplifier block from the Equivalent Baseband library of RF Blockset. Unlike the Idealized Baseband blocks, the Equivalent Baseband blocks allow you to set the center frequency and impedances. Thus, if you want to model an RF system at real RF frequencies, the loading and reflection effects, we recommend these physical blocks.

In this model, we set several parameters to Z_0 :

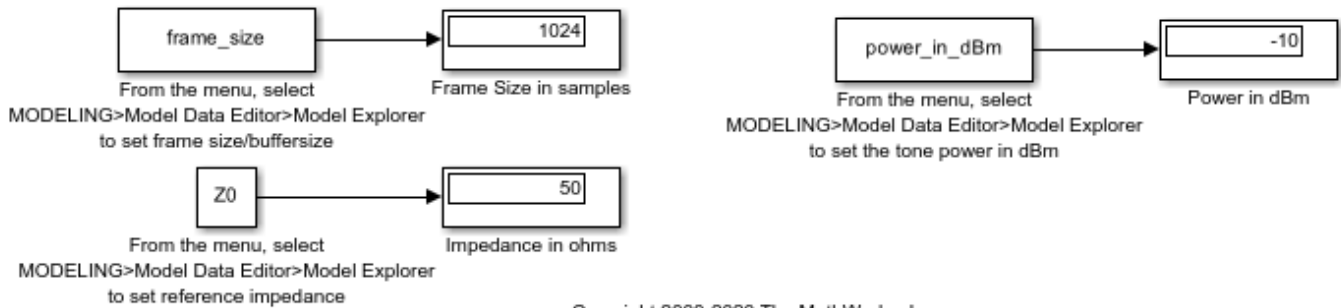
- **Source impedance** parameter of the Input Port block dialog box
- **Reference impedance** parameter of the S-Parameters Amplifier block dialog box
- **Load impedance** parameter of the Output Port block dialog box

In the Input Port block, set the **Center frequency (Hz)** parameter to $2e9$ (2 GHz). The baseband frequencies of the two-tone complex Simulink signal are 200 kHz and 300 kHz. Thus, in the RF system (the Equivalent Baseband blocks connected between the Input Port and Output Port blocks), the real RF two-tone frequencies are 2.0002 GHz and 2.0003 GHz. By default, the Spectrum Scope displays in baseband. To display the desired tones at 2.0002 GHz and 2.0003 GHz (-10 dBm each) and the intermodulation tones at 2.0001 GHz and 2.0004 GHz (-70 dBm each), set the **Frequency display**

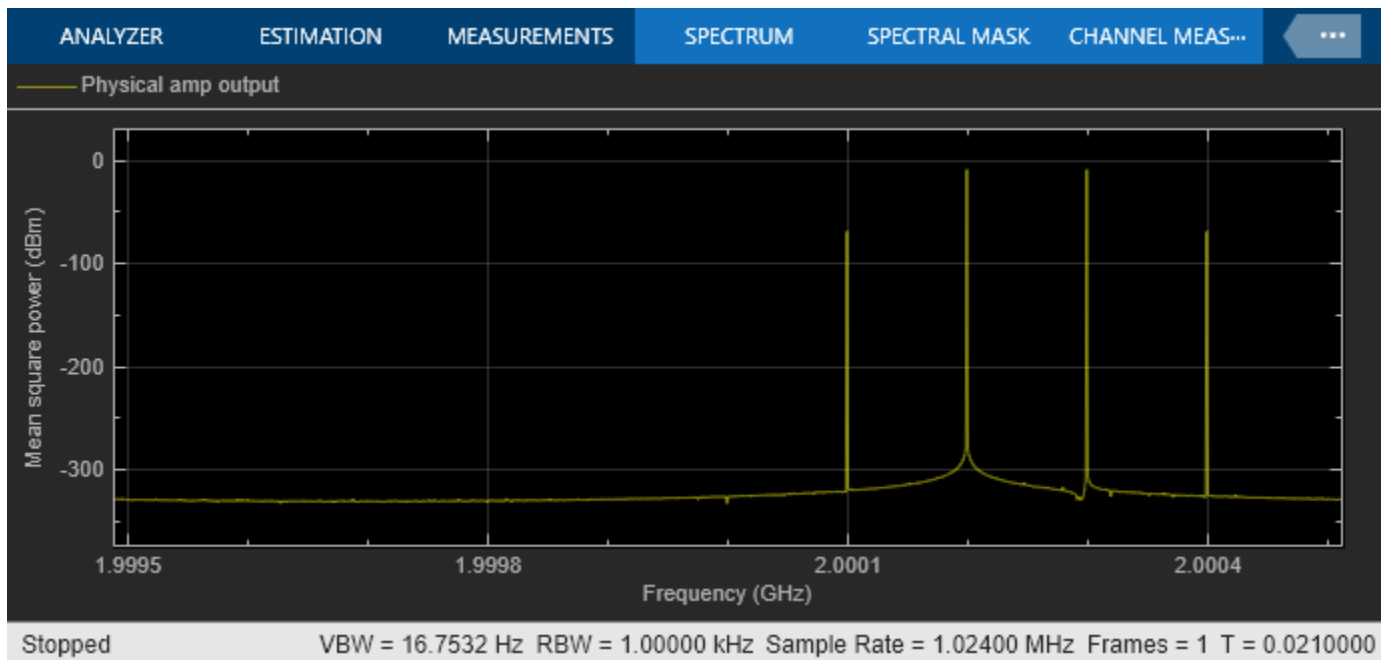
offset: parameter in the "Axis Properties" tab to the value of the Center frequency, in this case it is $2e9$ (2 GHz).



Model Constants



Copyright 2003-2020 The MathWorks, Inc.

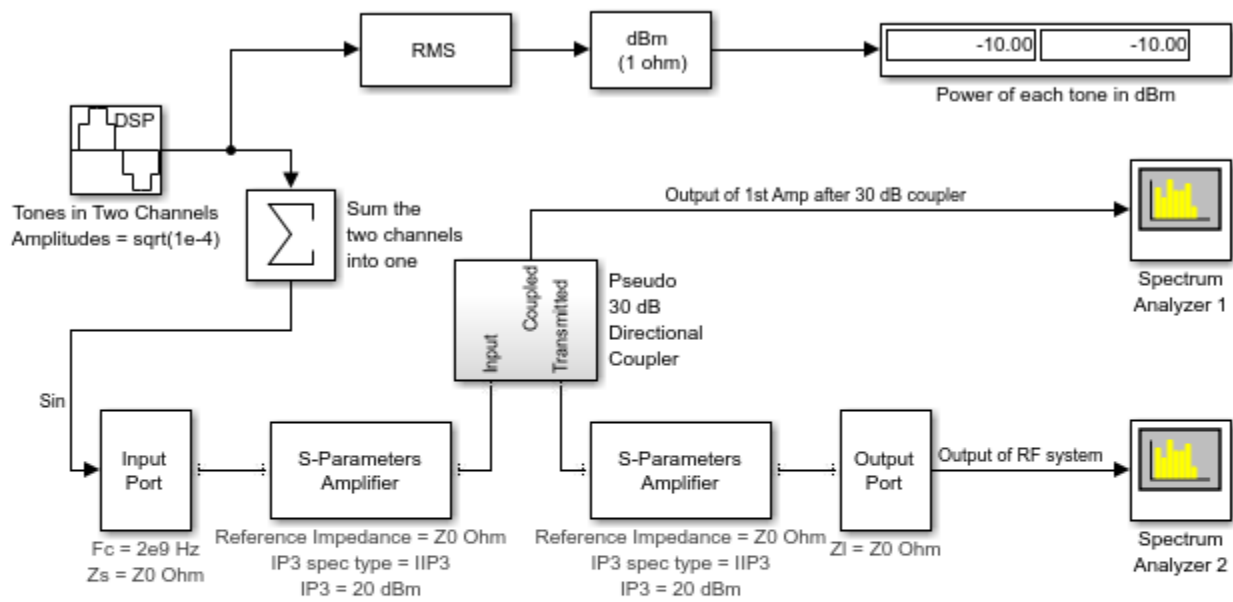


Displaying Power Spectrum Within Cascade of RF Blockset Equivalent Baseband Blocks

The Output Port block lets you create a link budget plot for multi-block cascades. This feature allows you to visualize the characteristics of the cascade non-intrusively. Therefore, it is not usually necessary to tap a cascade of RF Blockset Equivalent Baseband blocks. However, it is sometimes useful to do so, for example, to see the modulated spectrum at an intermediate point. The final model of a Tap Cascade of Equivalent Baseband Blocks in RF Blockset, achieves this with a subsystem that approximately models a real-world directional coupler. As with its real-world counterpart, the tapping is intrusive in that it presents a load impedance to the downstream part of the cascade and it drives the upstream part with a source impedance.

Double click on the subsystem "Pseudo 30 dB Directional Coupler" to open it and see how the model works. The Output Port and Input Port blocks correspond to the input and output impedance of the mainline of a real-world directional coupler, respectively. However, the phase behavior of a real-world directional coupler is not modeled here.

Tap Cascade of Physical Blocks in RF Blockset

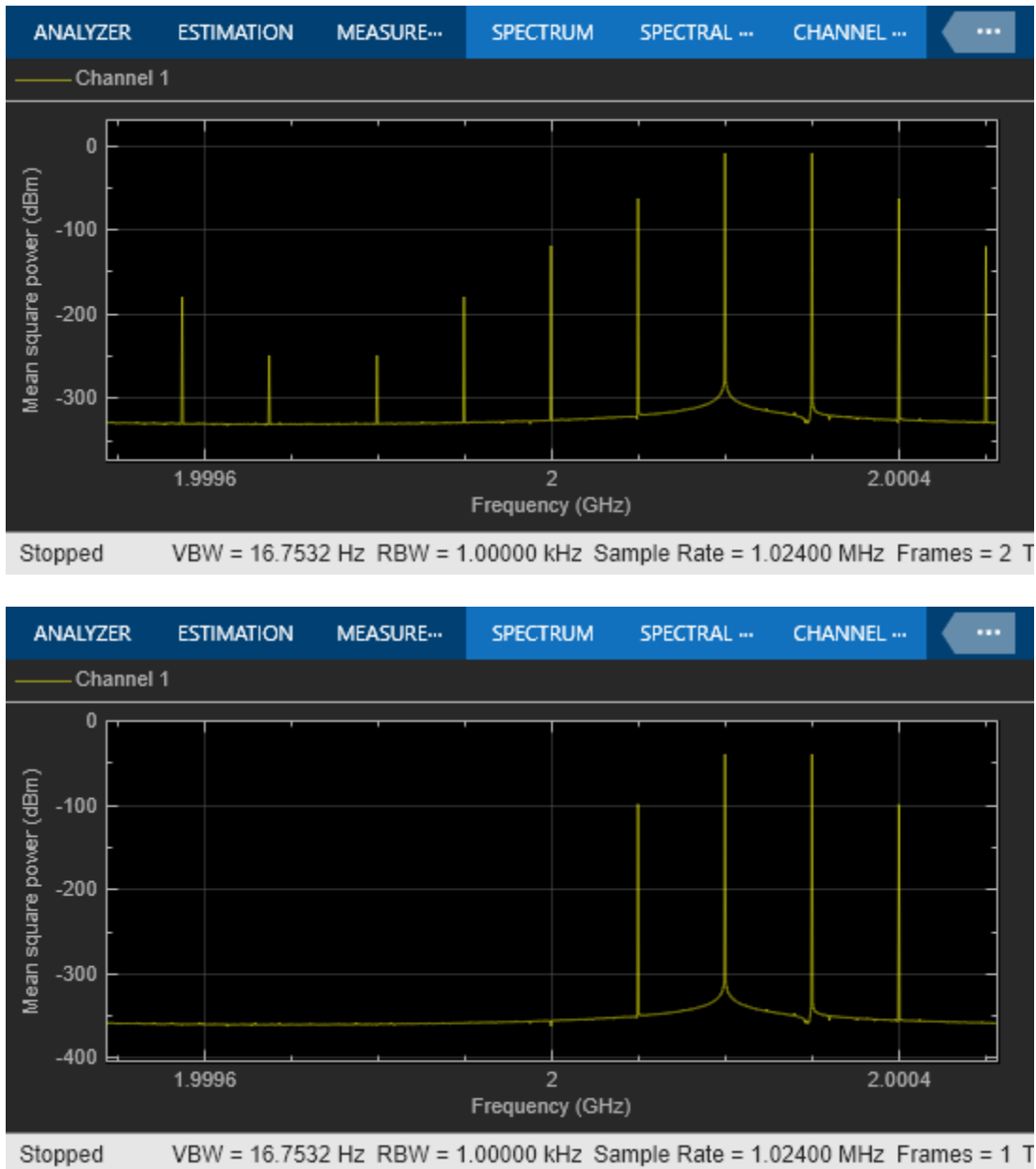


Model Constants



MODELING>Model Data Editor>Model Explorer
to set frame size/buffer size

MODELING>Model Data Editor>Model Explorer
to set reference impedance



The first Spectrum Scope shows the intermodulation tones after one amplifier. Note that the power is 30 dB down because of the characteristics of the "Pseudo 30 dB Directional Coupler" subsystem. You could calibrate this out with a gain block, or even modify the subsystem to model a 0 dB-loss "active" directional coupler. The second Spectrum Scope shows an increased level of intermodulation tones after a cascade of two non-linear amplifiers. The second Spectrum Scope shows an increased level of intermodulation tones after a cascade of two non-linear amplifiers.

See Also

Input Port | Output Port

Related Examples

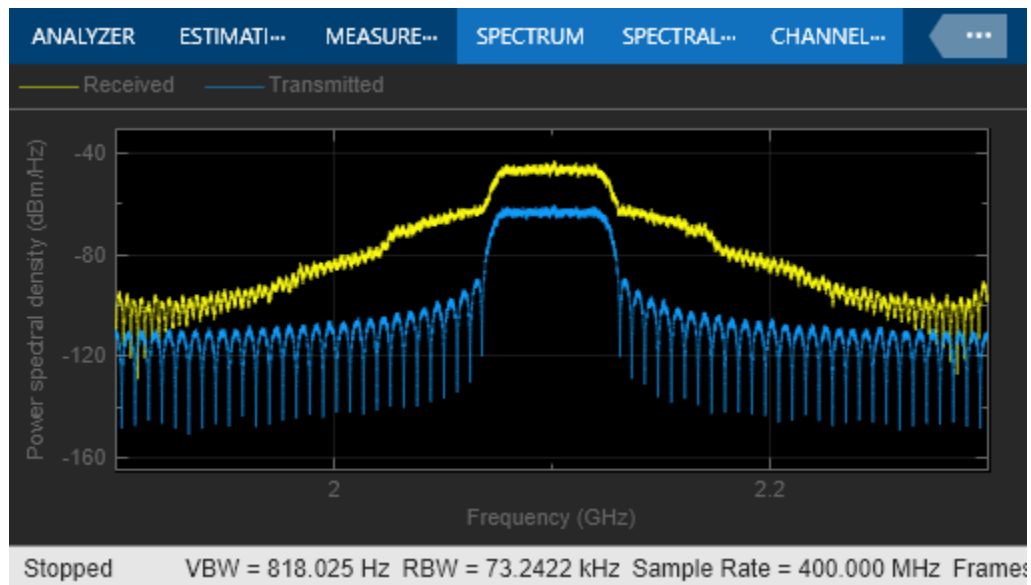
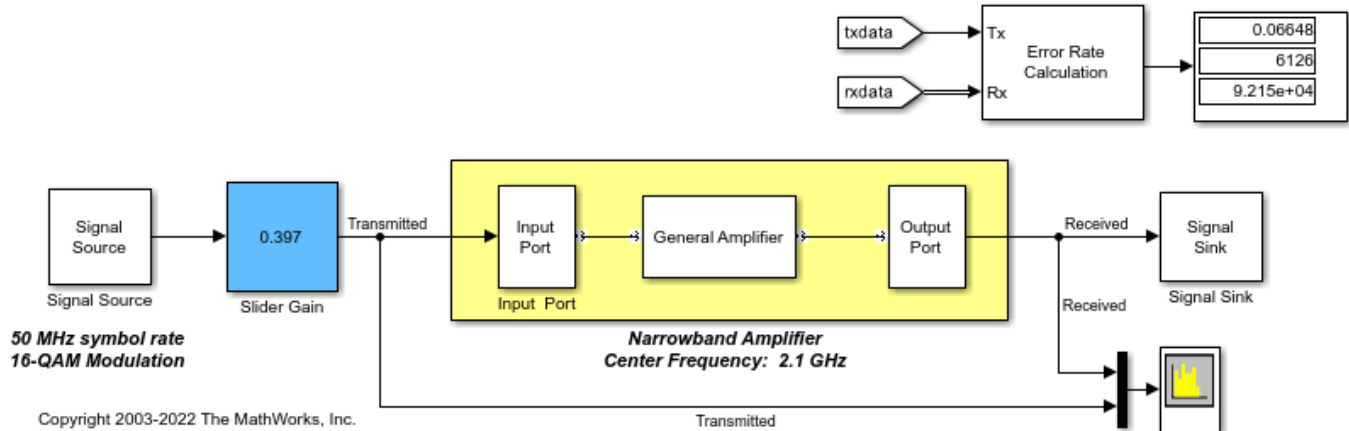
- “Power Ports and Signal Power Measurement in RF Blockset” on page 8-11

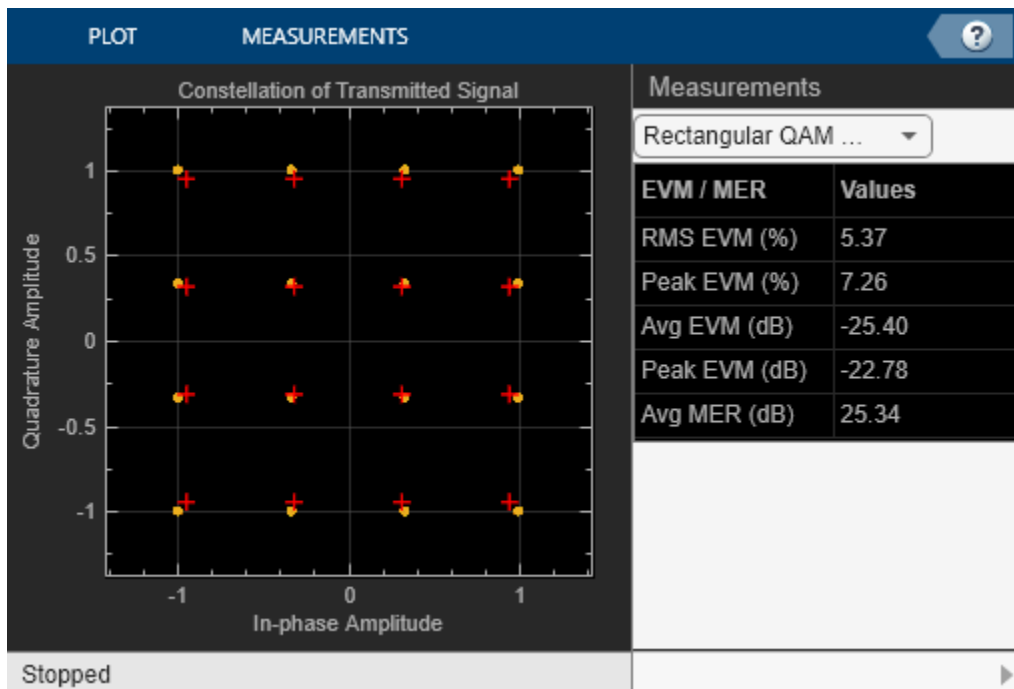
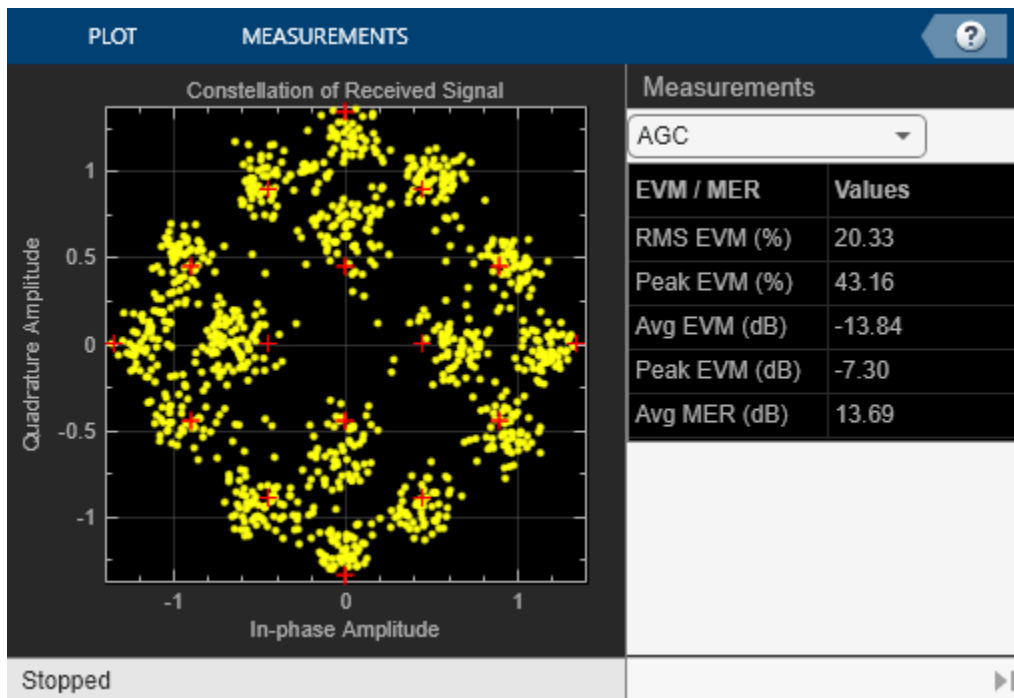
Effect of Nonlinear Amplifier on 16-QAM Modulation

This model shows the nonlinear effect of an RF Blockset™ Equivalent Baseband amplifier on a 16-QAM modulated signal.

- 1 Open the model and select **Run** to simulate the amplifier nonlinear effect on a 16-QAM modulated signal.
- 2 Compare the spectrums of transmitted and received signals, and observe the spectrum regrowth at the received signal. This regrowth is due to the nonlinearity of the amplifier.
- 3 Compare the constellations of transmitted and received signals, and observe the signal distortion at the received signal. This distortion is due to the nonlinearity of the amplifier.

16-QAM Modulation with Nonlinear Amplifier





To view the nonlinear effect, double-click the Slider Gain block and change the value of the gain while the model is running.

See Also

General Amplifier | Input Port | Output Port

Related Examples

- “User-Defined Nonlinear Amplifier Model” on page 8-138
- “Radar Tracking System” on page 8-132

Executable Specification for System Design

This example shows how to use the Model-Based Design methodology to overcome the challenge of exchanging specifications, design information, and verification models between multiple design teams working on a single project. The example uses a simple project: an executable specification that encapsulates information from all teams. The example includes information on how to use Signal Processing Toolbox™, DSP System Toolbox™, Communications Toolbox™, RF Toolbox™, and RF Blockset™ in a multi-domain design.

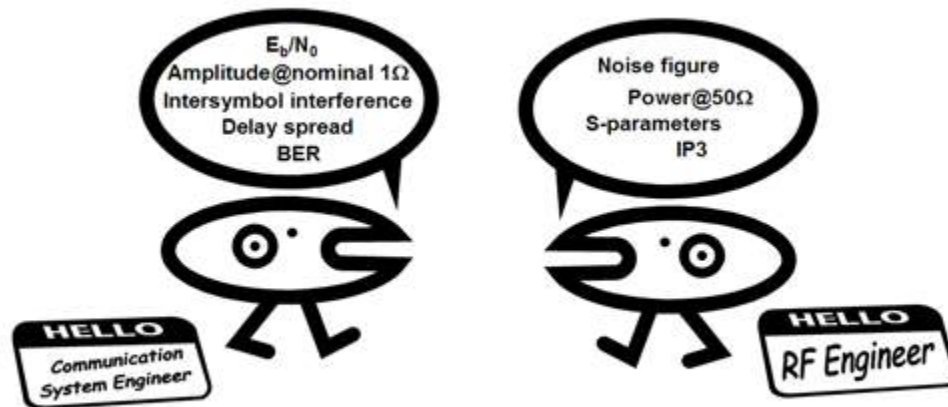


Figure 1: Bridging the Jargon Gap between RF and System Engineers

Model-Based Design

Model-Based Design uses a system-level model at the center of the development process. Before partitioning the system-level model among various design teams, the initial system model, developed by the system engineer, is validated against requirements and standards. With a validated error-free executable specification, design and implementation go smoothly. As the design progresses, verification can include co-simulation and testing with hardware-in-the-loop.

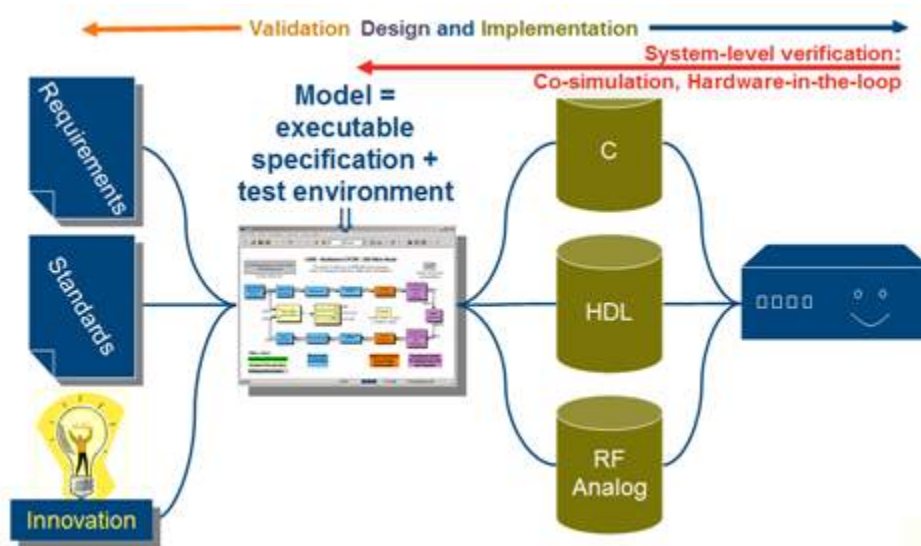


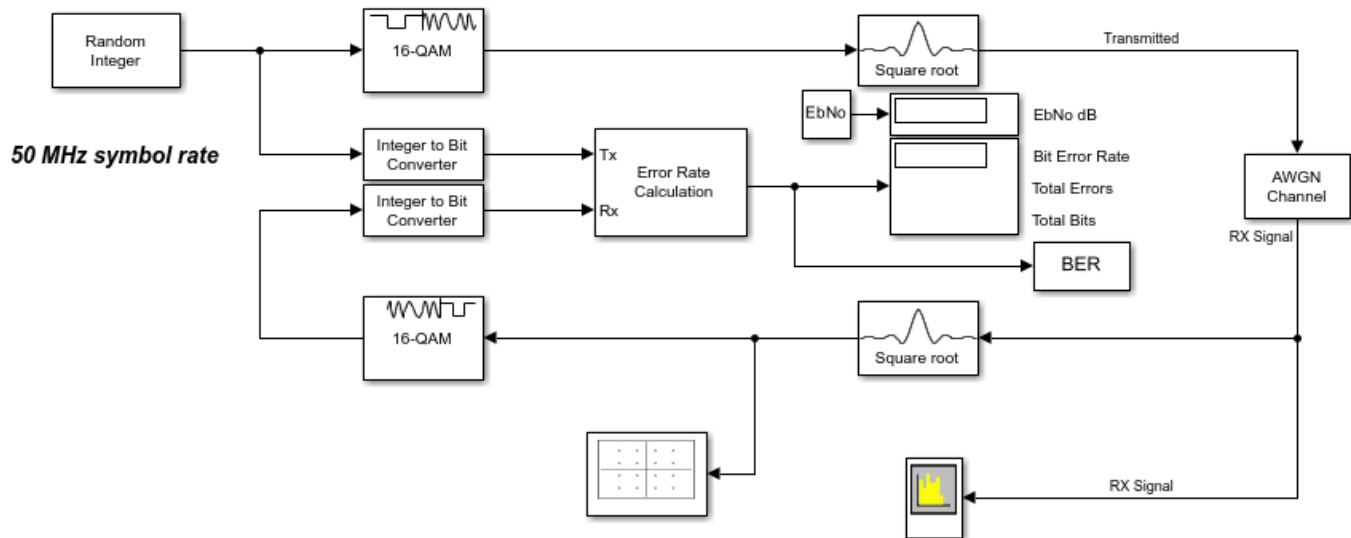
Figure 2: Model-Based Design -- A system-level model is at the center of the development process

Rather than talking about all the elements in the development flow, this example focuses on how Model-Based Design aids your engineering teams. The idea is to enable the System Engineer to initially create an executable specification in the form of a Simulink model that can be distributed to design teams. A team, such as the RF team, will devise a subsystem, extract a verification model and import it into the RF Toolbox. The RF team then returns the solution to the System Engineer, who reevaluates the overall performance of the system with the impairments from the RF subsystem. The design teams can go back and forth, iterating to find an optimal solution as the design proceeds. Perhaps the RF section can use a more efficient or less costly device if the signal processing algorithms are altered. Or, perhaps a small increase in fixed-point wordlength can free up some of the implementation loss budgeted, and enable a lower cost RF component to be used. The opportunities for cross-domain optimization are enhanced by this Model-Based design methodology.

Baseline Model: Communications Toolbox™ with No RF Modeling

```
open('rfb_receiver_0.slx')
```

16-QAM Modulation with Root-Raised Cosine Filtering



Copyright 2003-2012 The MathWorks, Inc.

The model `rfb_receiver_0.slx` shows the kind of Communication System Toolbox model that inspired the creation of the RF Blockset Equivalent Baseband library. Note that this is a simple model for illustrative purposes. Communications Toolbox includes more complex models of WCDMA, 802.11, DVB-S2, etc. However, the concepts presented can be applied to more complex models as well.

The simple wireless communication system consists of a message source, QAM modulator, root raised cosine filter and an AWGN channel. The model is an executable specification, and is used to validate the specification against requirements and acceptance criteria, "At a BER of $1e-3$, the E_b/N_0 must be no greater than 1dB above the theoretical bound for 16QAM."

To validate the spec, you can use a previously saved BERTool session file `rfb_receiver_0.ber`. To find this file, type the following command at the MATLAB prompt

which `rfb_receiver_0.ber`

Open the BERTool using the MATLAB command `bertool`. From the File==>Open Session... dialog box, navigate to the saved session `rfb_receiver_0.ber`. Now click on the Monte Carlo tab, and then click on the Run button. A figure like the one below is generated:

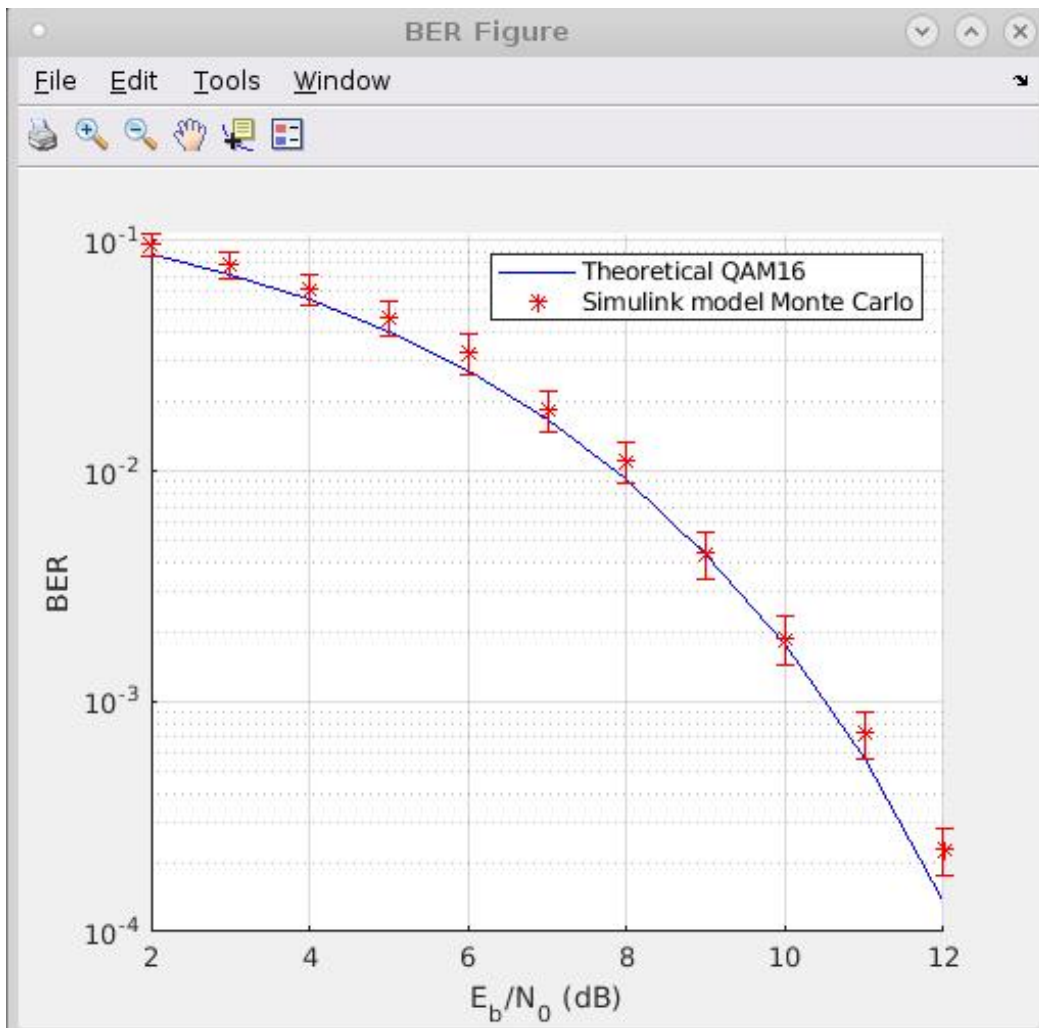


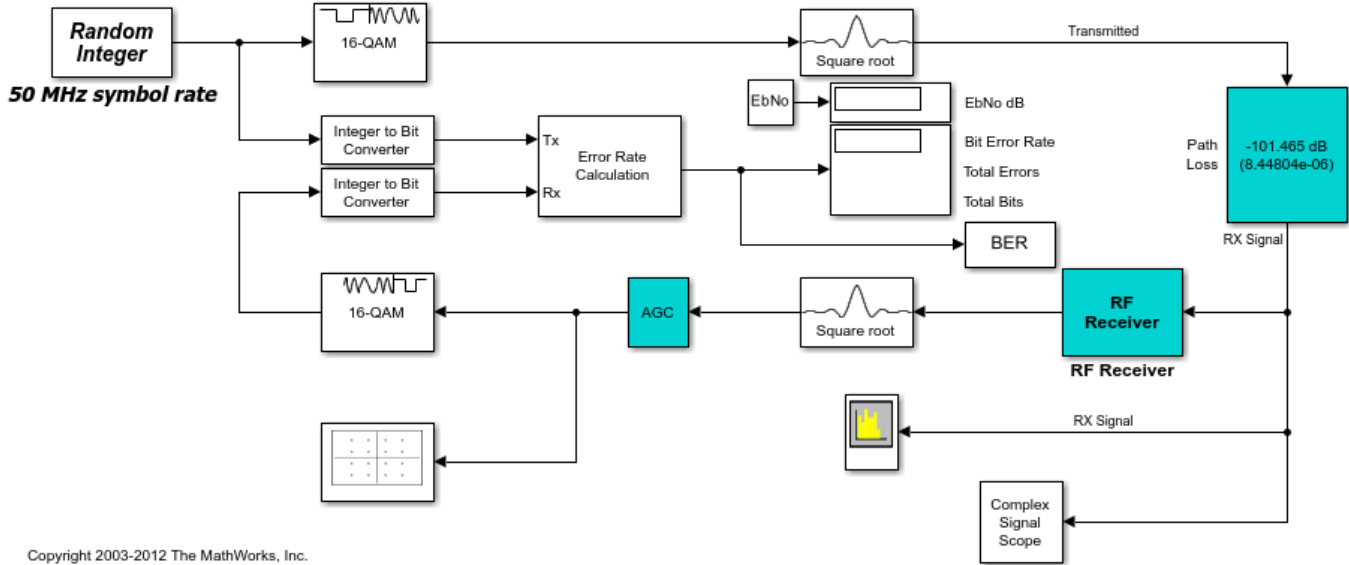
Figure 3: BER versus E_b/N_0 plot without RF impairments

The E_b/N_0 for a given BER value is a little higher than the theoretical bound because of implementation losses. (In the present case, the main loss is due to the finite length of the root raised cosine filters.) But the degradation is within acceptance criteria.

Adding RF Specifications to the Baseline Model

```
open('rfb_receiver_1.slx')
```

16-QAM Modulation with Root-Raised Cosine Filtering



Copyright 2003-2012 The MathWorks, Inc.

Let's elaborate the baseline model and see how it changes with additional refinement using RF Blockset components. The first step is to replace the AWGN block with a path loss block (shown in the preceding figure in cyan); this will lower the signal level close to the end of range value. The path loss (in dB) required to bring the unit power (1W) down to a given Eb/No (also in dB) at the receiver input is:

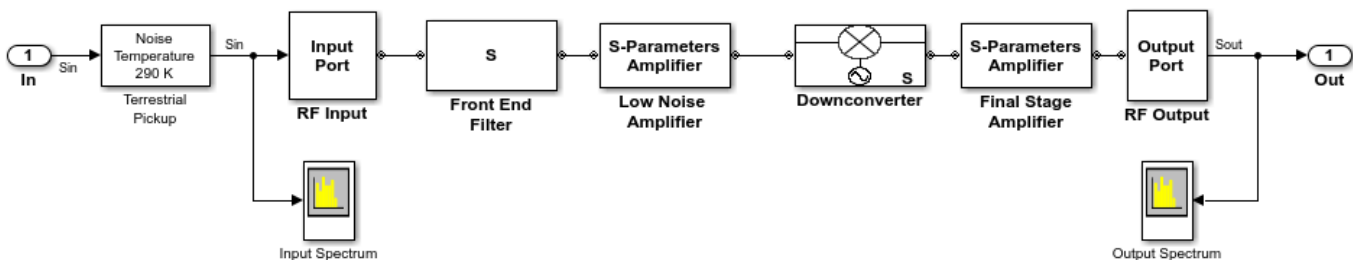
$$\text{path_loss} = 10 \cdot \log_{10}(k \cdot T_{\text{ref}} \cdot B \cdot M) + \text{EbNo} + \text{NF}$$

where k is Boltzmann's constant (~1.38e-23 J/K), T_ref is the IEEE® standard noise reference temperature (290K), B is the noise bandwidth (~50 MHz in this case), and NF is the receiver noise figure in dB.

Next, the cyan-colored RF receiver subsystem and AGC Blocks are included. The AGC Block is a consequence of using realistic signal levels required by the demodulator.

The RF Receiver Subsystem Examined

```
open('rfb_receiver_1.slx')
open_system('rfb_receiver_1/RF Receiver')
```



Now examine the RF Receiver subsystem, which is a cascaded model of a super heterodyne receiver. The receiver uses blocks from the RF Blockset Equivalent Baseband library. The Simulink signal enters the RF domain through a gateway "Input Port" block. Notice that the connectors after the gateway are different. The standard Simulink arrows have been replaced with RF connection lines. This is to remind us that RF signals are bidirectional. The receiver is a cascade of components each represented as a 2-port network: a filter, a LNA, a mixer, and an IF strip. The Output Port, in this case, is not only the gateway back to Simulink but also represents an ideal quadrature down conversion mixer. Here is a framework or architecture for a receiver that is not yet designed. An executable specification for the RF engineer has been created. Each stage of the RF subsystem includes a budget for the overall gain, noise and nonlinearities, as shown in the following figure.

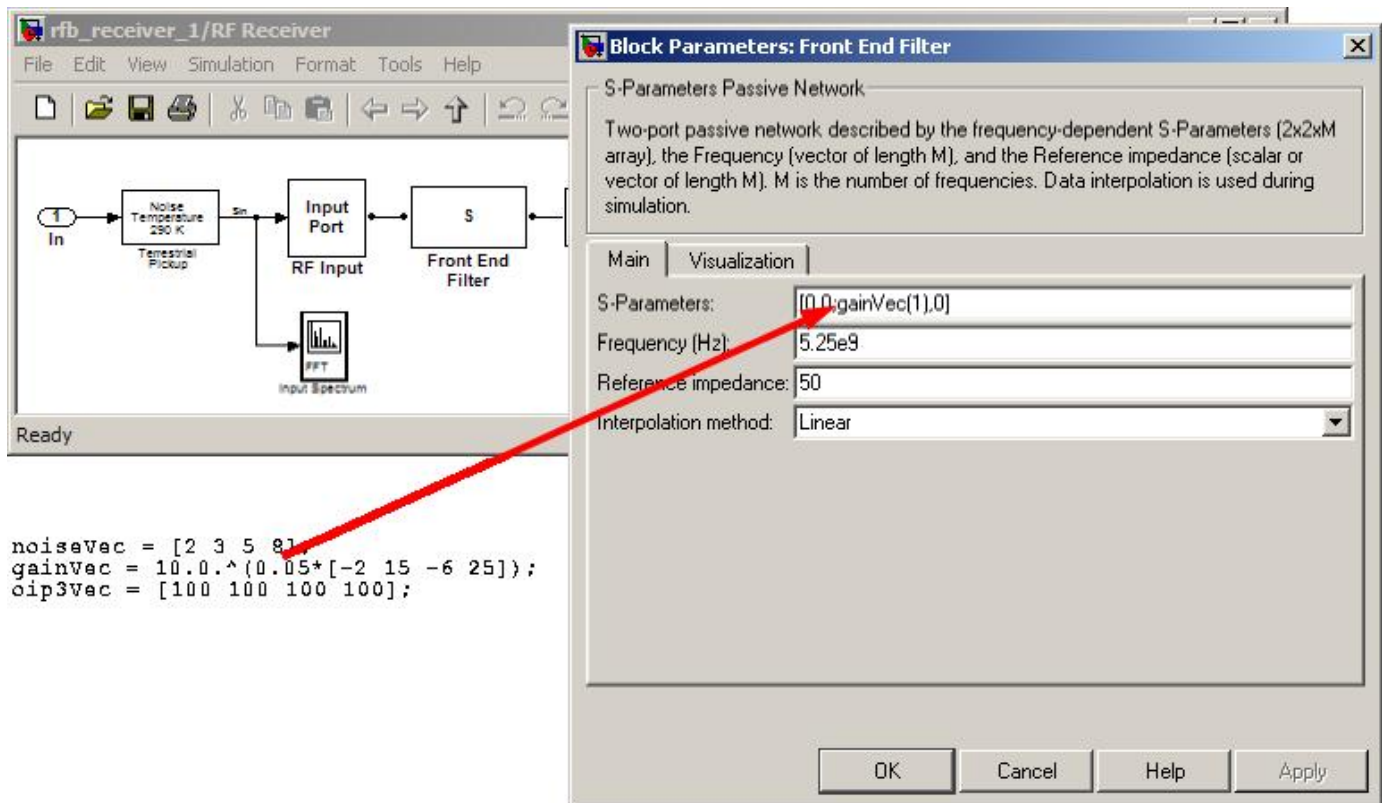


Figure 4: Specification of Amplifier Block Parameters

As an example of budgeting, consider the front end filter in the above figure. The S-parameters are specified at a single frequency point using the first element of the gainVec array that was entered into the base workspace using the **PostLoadFcn*** under the Callbacks tab in the Model Properties panel. Each element of the array refers to a stage, so the index 1 refers to the first stage. Values for OIP3, on the Nonlinearity data tab, and for Noise Figure, on the Noise data tab, are similarly specified.

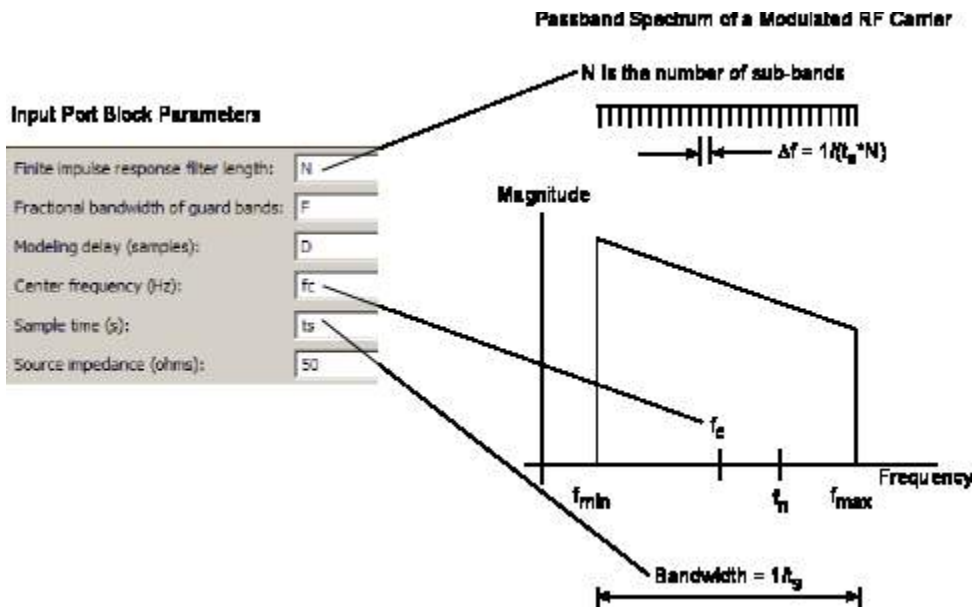


Figure 5: Specification of Complex Baseband-Equivalent Simulation Parameters

Now open the Input Port block. This port contains parameters that apply to the overall RF subsystem. A narrowband modeling approach is used to capture the in-band effects that impact downstream signal processing blocks. The range of frequencies is specified through the **Center frequency** parameter; the **Sample time** parameter (which is $1/\text{Bandwidth}$), and the **Finite impulse response filter length** parameter (which is the length of the impulse response filters that are used in modeling RF components). A longer length time-domain filter will give finer frequency-domain resolution within the specified bandwidth. To model mismatch at the input of the first component, source impedance is also specified here. Notice the "Add noise" checkbox. To include noise in the simulation, you must select this "Add noise" checkbox.

Block Parameters: RF Input

Input Port
Connection block from Simulink to RF Blockset physical blocks.

The RF Blockset physical blocks use a baseband-equivalent modeling technique. This technique models a bandwidth of $1/(\text{Sample time})$, centered at the specified Center frequency parameter value. This frequency value corresponds to 0 Hz in the baseband-equivalent model.

The block provides the option to interpret the Simulink signal as either the incident power wave to the RF system or the 'Incident power wave' option is the 'Source voltage' option is provided. If the Simulink signal is the incident power transmitted power wave. If the input voltage.

The block controls the modeling of and the Output Port block using:
 * FIR filters to model the frequency response
 * Look-up tables to model the noise response.

Optional guard bands can be specified. The guard bands are implemented using a notch filter response. Modeling delay may be specified.

Parameters:

Treat input Simulink signal as: Incident power wave

Source impedance (ohms): 50

Finite impulse response filter length: 512

Fractional bandwidth of guard bands: 0

Modeling delay (samples): 0

Center frequency (Hz): 5.25e9

Sample time (s): 1/(8*50e6)

Add noise

Initial seed: 67987

OK Cancel Help Apply

RF Receiver

File Edit View Simulation Format Tools Help

Normal

In → Noise Temperature (250 K) → Input Port → Front End Filter → S-Parameters Amplifier (Low Noise Amplifier) → Downconverter → S-Parameters Amplifier (Final Stage Amplifier) → Output Port → Out

Input Spectrum, Output Spectrum

Ready 72% FixedStepDiscrete

$C_A = 2kT \begin{bmatrix} R_n & \frac{NF_{\min} - 1}{2} - R_n Y_{opt}^* \\ \frac{NF_{\min} - 1}{2} - R_n Y_{opt} & R_n |Y_{opt}|^2 \end{bmatrix}$

Noise Correlation Matrix is computed for each block

Figure 6: Noise Modeling with the RF Blockset Equivalent Baseband library

The AWGN block models overall noise as a signal-to-noise ratio. By contrast, blocks from the RF Blockset Equivalent Baseband library model noise by adding the noise contribution of each block individually. For each block, the noise is modeled using an appropriate formulation determined by the set of noise parameters supplied for that block. Once the noise for each block is calculated, the overall system noise model is developed. This overall model includes the position of each block in the cascade (i.e., includes the gain of the subsequent stages).

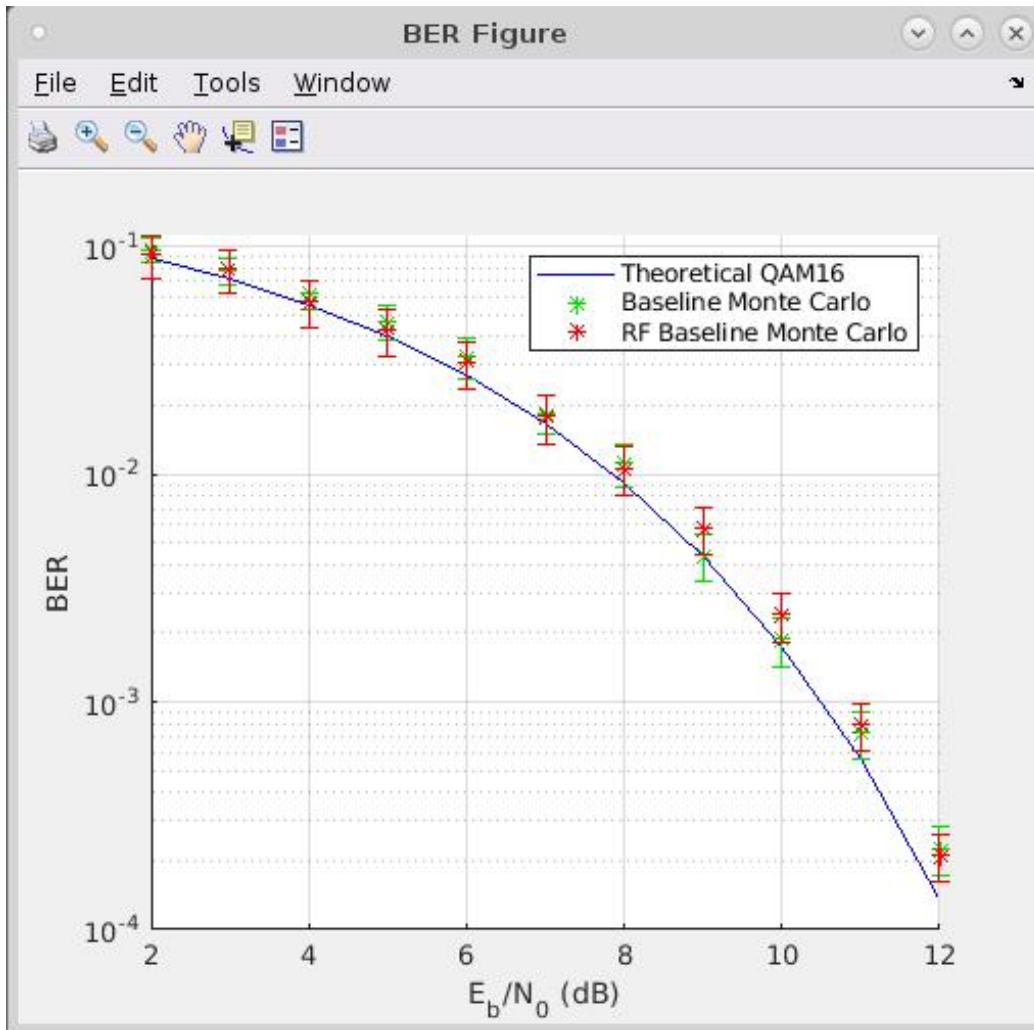


Figure 7: BER versus E_b/N_0 plot with RF impairments

Plots of BER versus E_b/N_0 comparing the theoretical, Baseline and Baseline with RF impairments models are given in Figure 7. This is a simple illustration of the convenience afforded by the Model-Based Design methodology. At this point in the process, an executable specification has been developed. This specification will be used by teams to design their subsystems. In the case of the RF subsystem, the abstract RF blocks will be replaced by discrete components. As each RF block is realized, its effect on the system's design criteria can be assessed.

```

bdclose('rfb_receiver_0');
bdclose('rfb_receiver_1');

```

See Also

S-Parameters Amplifier

Related Topics

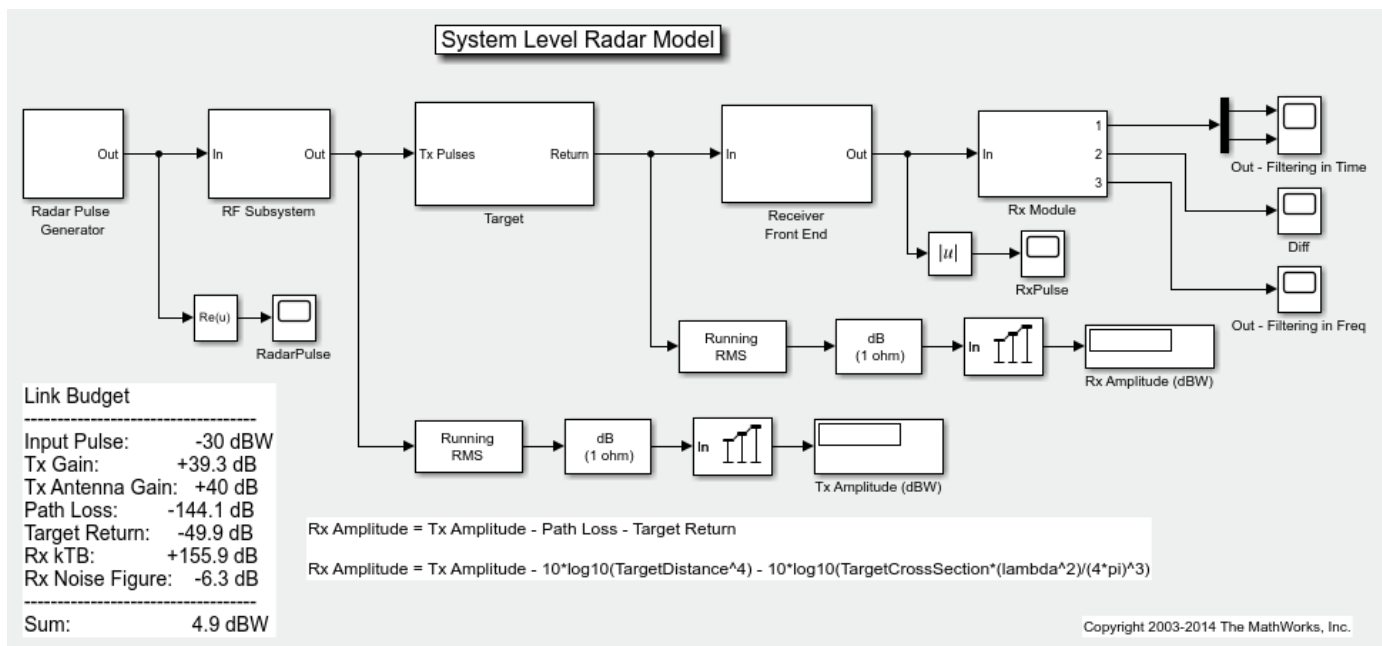
"Effect of Nonlinear Amplifier on 16-QAM Modulation" on page 8-120

Radar Tracking System

This model shows how to simulate a key multi-discipline design problem from the Aerospace Defense industry sector.

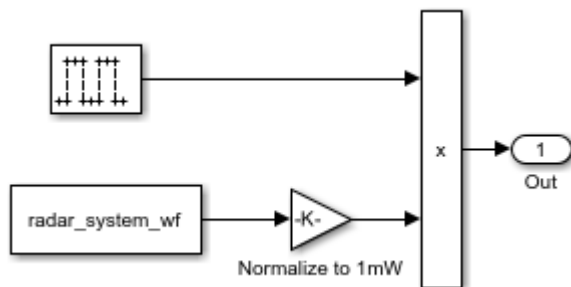
Structure of the Example

This example contains subsystems that model the essential features of a radar system. The model is typical of a radar system that is used for target position and velocity detection. The example includes a radar pulse generator, an RF Transmitter subsystem, a Simulink representation of a moving target, an RF Receiver and a Receive Module (Rx Module).



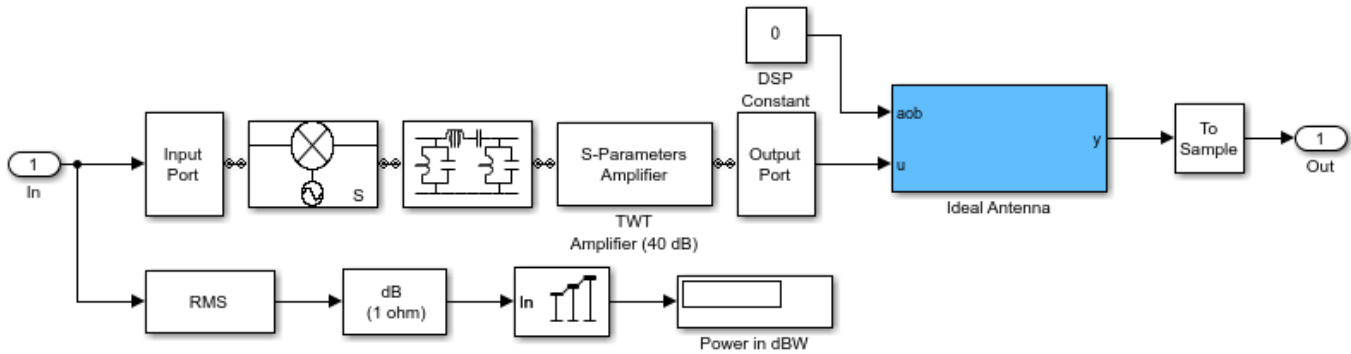
Radar Pulse Generator

The radar Pulse Generator creates a swept frequency signal (chirp signal) that has a 10 percent duty cycle. The subsystem is implemented by using Simulink® blocks and a signal from the MATLAB workspace that represents a chirp signal.



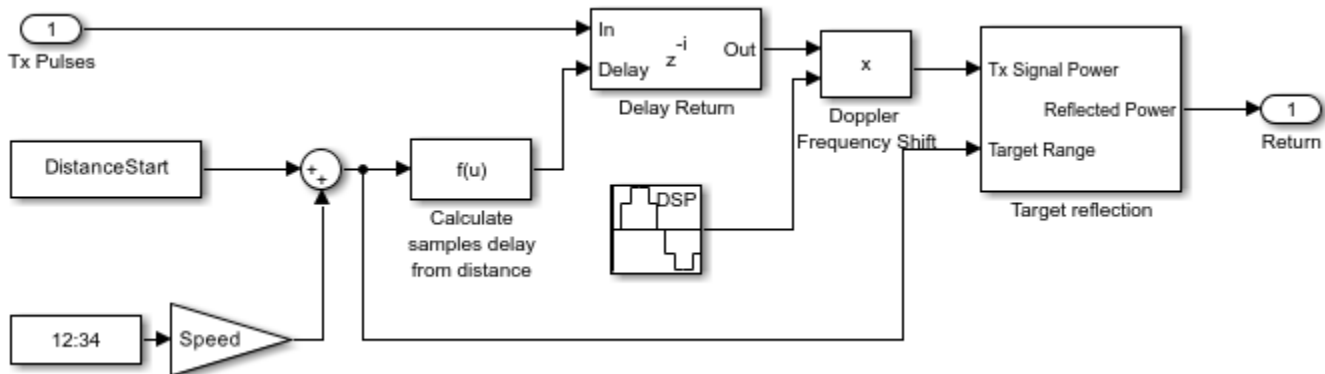
RF Transmitter Subsystem

This Subsystem is implemented with both core Simulink blocks as well as blocks from the RF Blockset Equivalent Baseband library. The RF Blockset subsystem represents a traveling wave tube amplifier. An ideal antenna is implemented via a Simulink gain block. Within the subsystem, there are DSP System Toolbox blocks used for calculating the power level of the baseband signal.



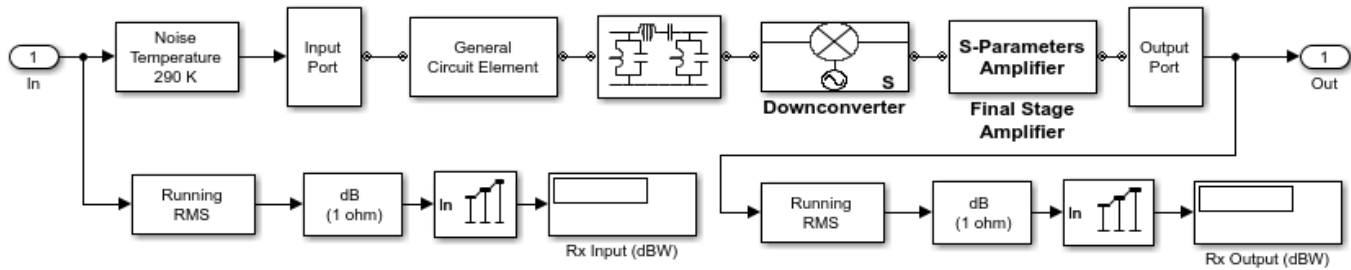
Target

The Target is implemented using theoretical implementations of a moving target that fully reflects all of the incident signal off of its cross-sectional surface, which is perpendicular to the direction of travel of the incident radar pulses.



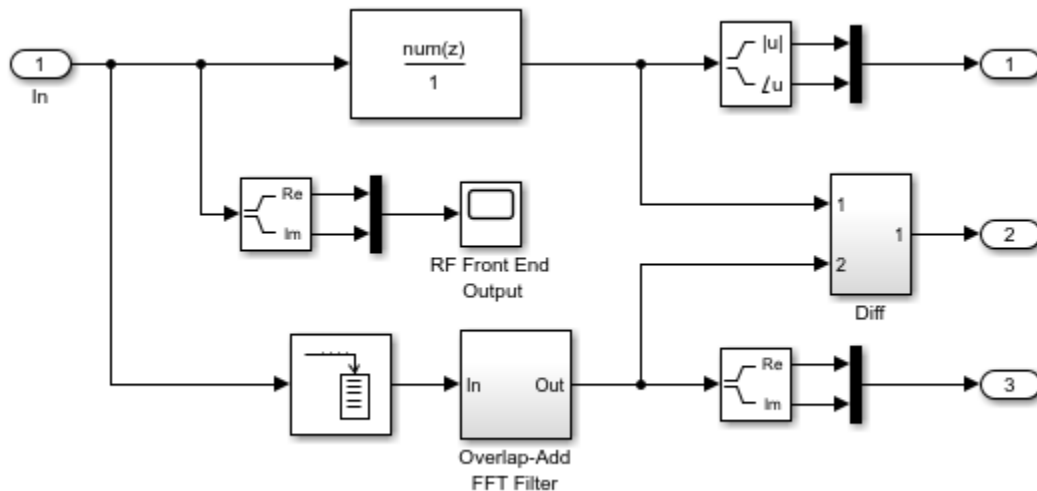
RF Receiver

The RF Receiver is implemented using the RF Blockset Equivalent Baseband library. The RF receiver is a super heterodyne receiver. The LNA is a matched amplifier. For broadband impedance matching, see the example of RF Toolbox: “Design Broadband Matching Networks for Amplifier”. The LNA is represented by a Touchstone® data file with noise data. The name of the datafile is samplebjt.s2p. Following the amplifier are behavioral models for a bandpass filter, mixer and a high gain, high noise amplifier.



Rx Module

The Rx Module in this example serves two purposes. First, the module contains a matched filter detector for target detection. Also, this module serves as a testbench where a theoretical filter implementation is realized via Simulink blocks, the output of each of these filters is compared, and the difference is plotted.



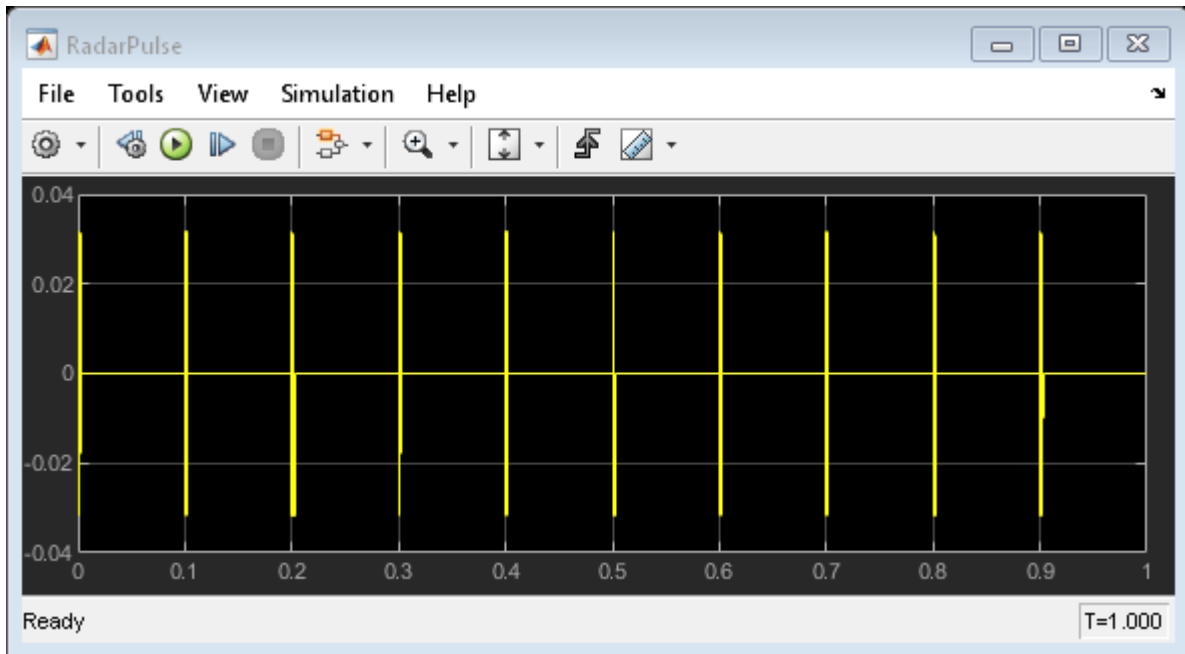
Exploring the Example

You can set the target cross section, target speed, and relative distance to the target by double-clicking the Target icon and specifying the corresponding parameters. At sufficiently large distances, the return signal cannot be detected within the noise. Similarly, the return signal cannot be detected in the noise if the target cross section is too small.

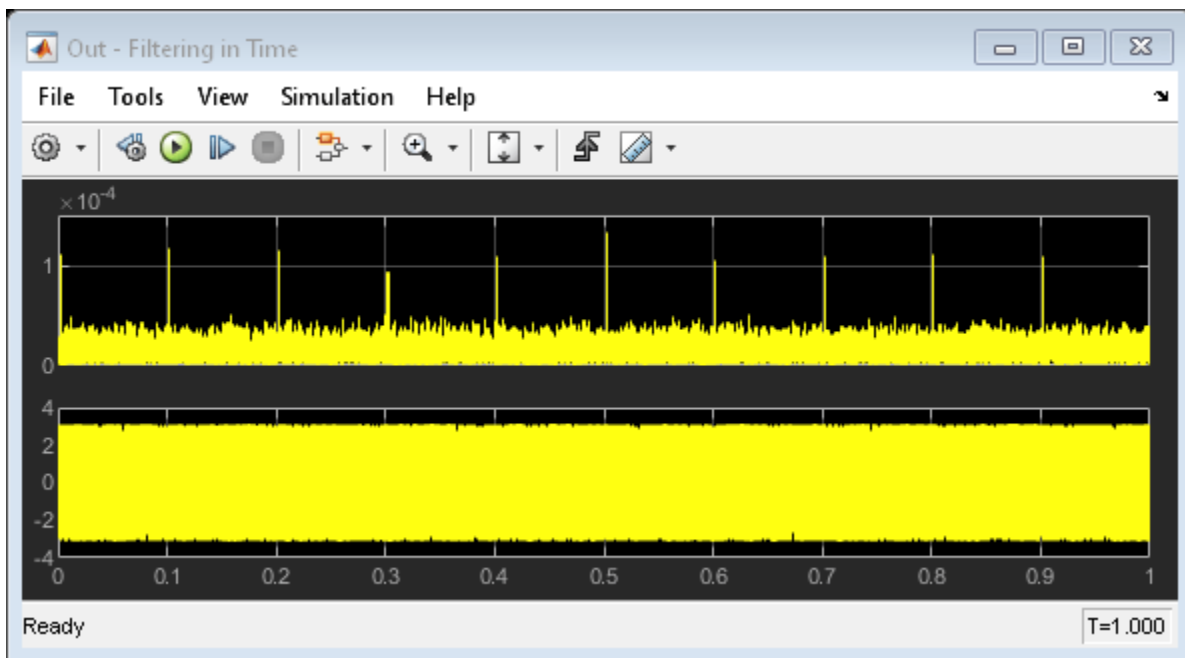
Results and Displays

After simulation, open four time-domain signal graphs of interest for examination.

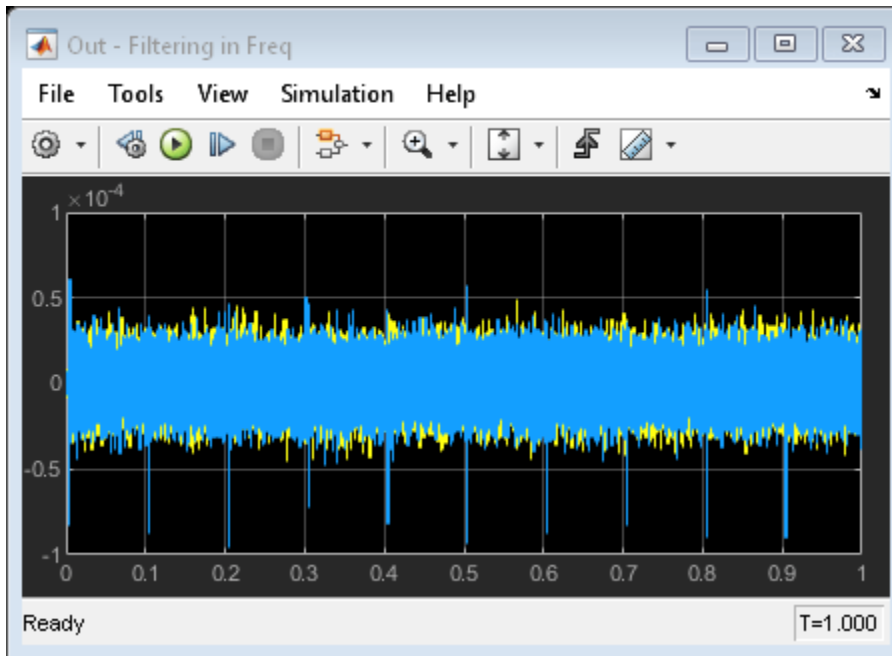
The first graph, "RadarPulse", displays the time-domain representation of a chirp signal with a 10% duty cycle.



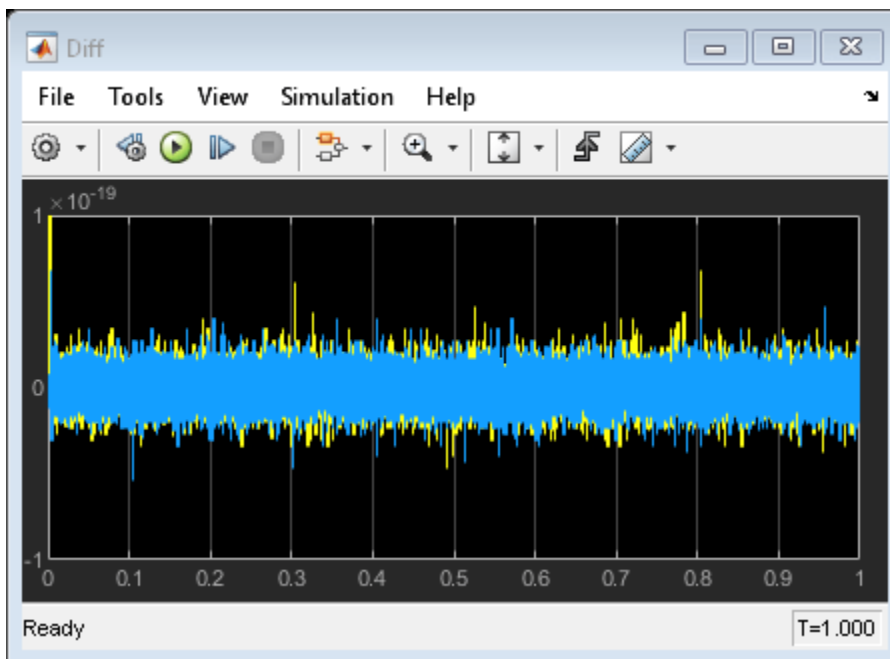
The second graph, "Out - Filtering in Time", displays the magnitude and phase of the filtered return signal with noise.



The third graph, "Out - Filtering in Frequency", displays the real and imaginary response of the filtered return signal with noise through an ideal filter implementation.



The fourth graph, "Diff", displays the difference between the results obtained in calculating the results for graphs two and three.



The following blocks display numerical results:

The Tx Amplitude (dBW) block displays the power transmitted in dBW.

The Rx Amplitude (dBW) block displays the target return power in dBW.

See Also

“Radar System Modeling” on page 8-87 | “Executable Specification for System Design” on page 8-123
| S-Parameters Amplifier | General Mixer

User-Defined Nonlinear Amplifier Model

This example shows how to:

- Generate user-defined custom models by creating an RF Toolbox™ object in the MATLAB® workspace and importing it into an Equivalent Baseband amplifier block.
- Create a nonlinear amplifier with an adjustable Pin-Pout curve.
- Simulate intermodulation products in a two-tone test model.

In the example, the Vin-Vout relationship for the amplifier is a simple polynomial. The example uses this voltage relationship to generate a Pin-Pout curve that is incorporated into the amplifier model. Although the example uses a specific polynomial function, the same approach can be used to create arbitrary functions of power out and phase versus power in and frequency. Indeed, it can be used to set any of the writable properties of a component, such as S-parameters or noise properties.

Define the Voltage-Out Versus Voltage-In Relationship

Use MATLAB commands to create a vector of polynomial coefficients that define the desired Vin-Vout relationship. A MATLAB convention is to store nth order polynomial coefficients in a row vector of length n+1 with the nth power in the first element and the zeroth power (constant) in the last (n+1 th) element. The power series here is for illustration, and can easily be altered. You may recognize the values as the first twelve terms of the polynomial series expansion of the hyperbolic tangent function.

```
TanhSeries = [-1382/155925 0 62/2835 0 -17/315 0 2/15 0 -1/3 0 1 0];
```

Next choose a range for the independent variable, Vin, and define a vector of input voltage values over this range. The low end (1 mV here) should be sufficiently low that the linear term dominates Vout. The high end should be chosen such that Vout just reaches its local maximum value. The resulting Vout will be extrapolated for all Vin greater than 1.23. Next compute the vector of output voltage values, Vout, using the power series.

```
Vin = linspace(0.001,1.23,100);    % volts
Vout = polyval(TanhSeries, Vin);    % volts
```

Create a Nonlinear Amplifier and Generate the Pin-Pout Curve

Create an RF Toolbox amplifier object with the default property values. Then, generate the Pin-Pout data (in watts) for the amplifier by dividing the square of the input and output voltage vectors by the reference impedance of the amplifier. Use the Pin-Pout data to specify the nonlinearity of the amplifier object. For this example, the Pin-Pout curve is defined for one frequency point (2.1 GHz) and used (by extrapolation) at all frequency points. See the RF Toolbox documentation for information on how to create a component with separately defined curves for any number of frequency points.

```
amp = rfckt.amplifier;
amp.NonlinearData.Freq = 2.1e9;    % Hz
Zref = 50;                          % ohm
amp.NonlinearData.Pin = {(Vin.^2)./Zref}; % watts
amp.NonlinearData.Pout = {(Vout.^2)./Zref}; % watts
```

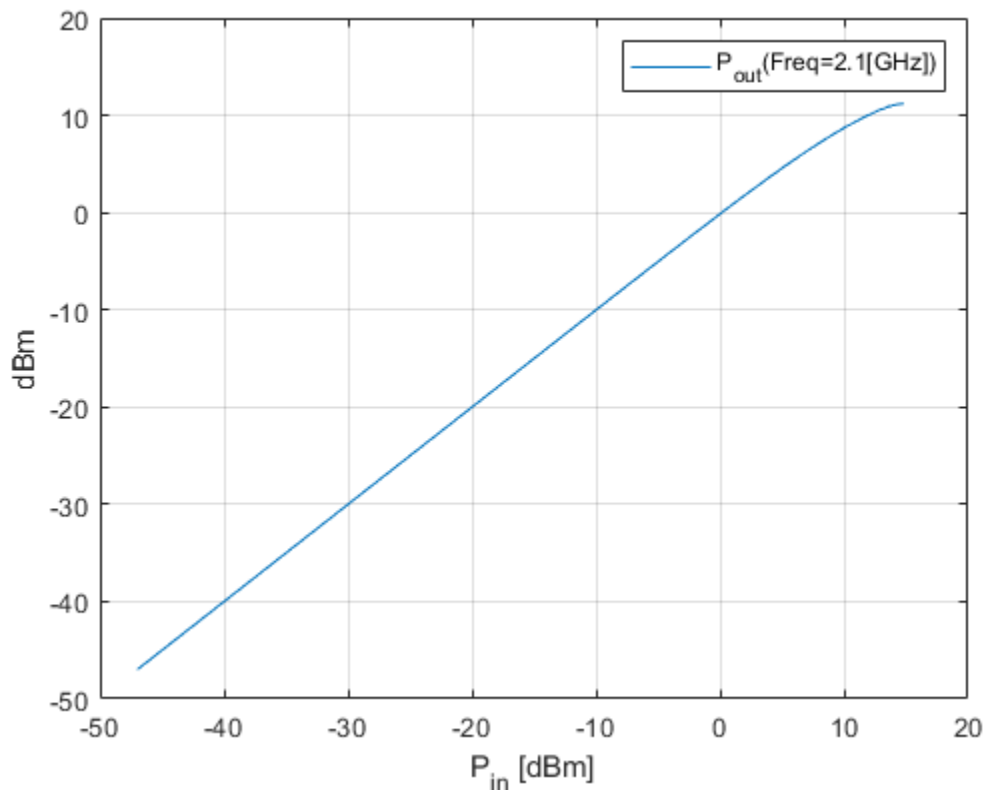
In this example, we define the phase change to be zero for all Pin and all frequencies, but RF Toolbox lets you set it to be a function of Pin and frequency.

```
amp.NonlinearData.Phase = {zeros(size(Vin))};
```


Make the Small-Signal Gain Consistent with the Power Gain Slope at Low Power

Define the S21 parameter of the amplifier at the frequency point for which you specified the power data. The S21 network parameter must be consistent with the gain slope at the low power end of the Pin-Pout curve at that frequency point. If these values are inconsistent, RF Blockset software will attempt to reconcile the data and issue a warning that it has done so. To ensure consistency, define the S21 parameter to be the linear term of the power series that defines the Vin-Vout relationship. The linear term is the next-to-last element in the vector. Plot the Pin-Pout curve.

```
amp.NetworkData.Freq = amp.NonlinearData.Freq;
amp.NetworkData.Data = [0 0; TanhSeries(end-1) 0];
fig = figure;
plot(amp, 'Pout');
```



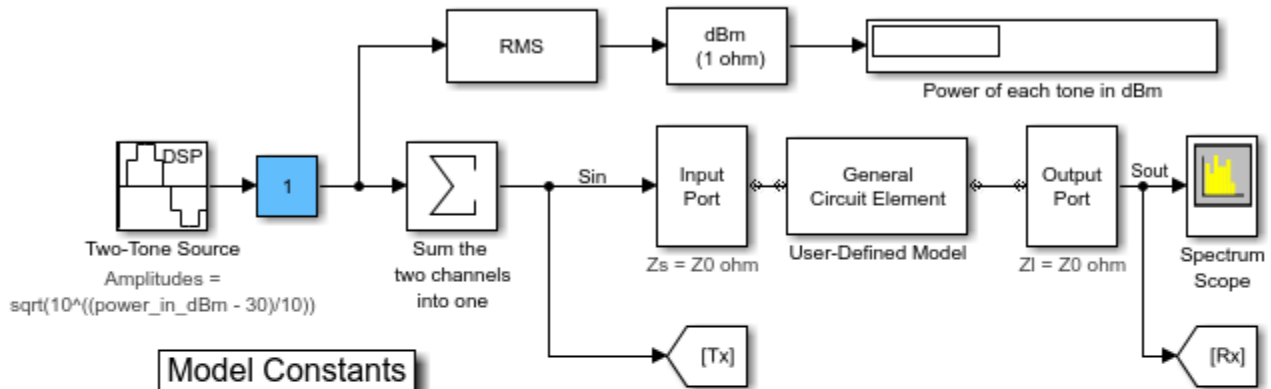
Run the Test Harness with an Input Power of 7 dBm Per Tone

The following figure shows the test harness for your new amplifier. The input signal consists of the sum of two tones, one 10kHz below the center frequency, and one 10kHz above it. The spectrum scope shows the various intermodulation products at higher and lower frequencies than the two test tones. The EVM subsystem calculates error vector magnitude. Open and Run "rfb_user_defined_amp.mdl" model. To view and edit the preset model workspace values, click the **Modeling** Tab in the Toolstrip and select **Model Explorer**.

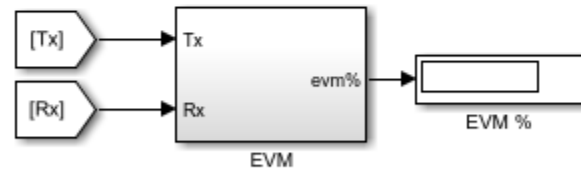
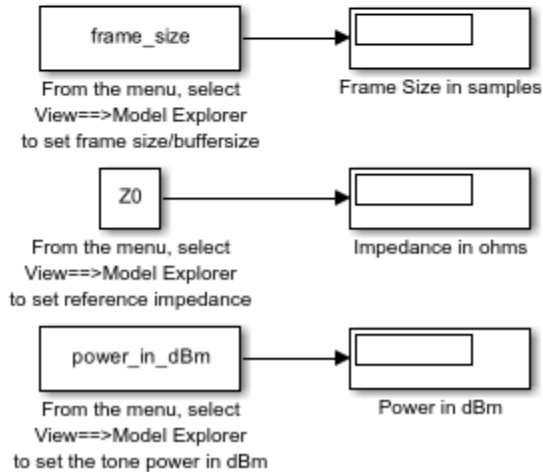
```
%
open('rfb_user_defined_amp.mdl.slx');
```

Two-Tone Input to User-Defined Model: Nonlinear Amplifier

Run this model from the demo script "rfb_user_defined_amp.m"

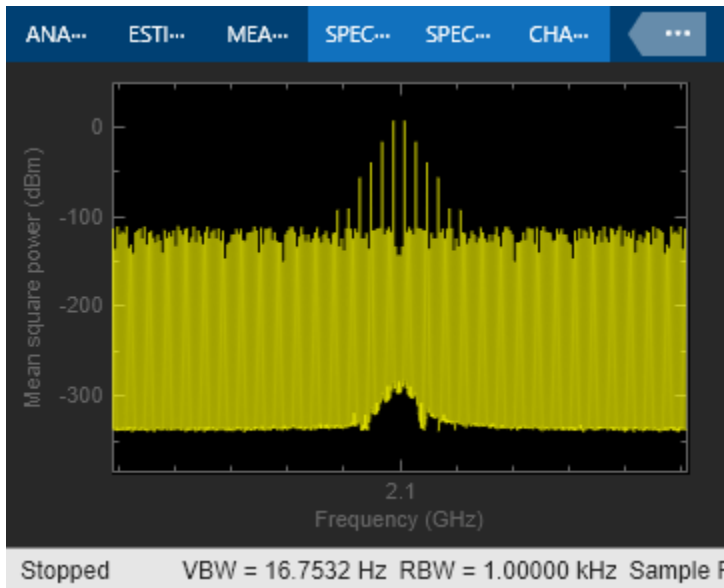


Model Constants



Copyright 2003-2014 The MathWorks, Inc.

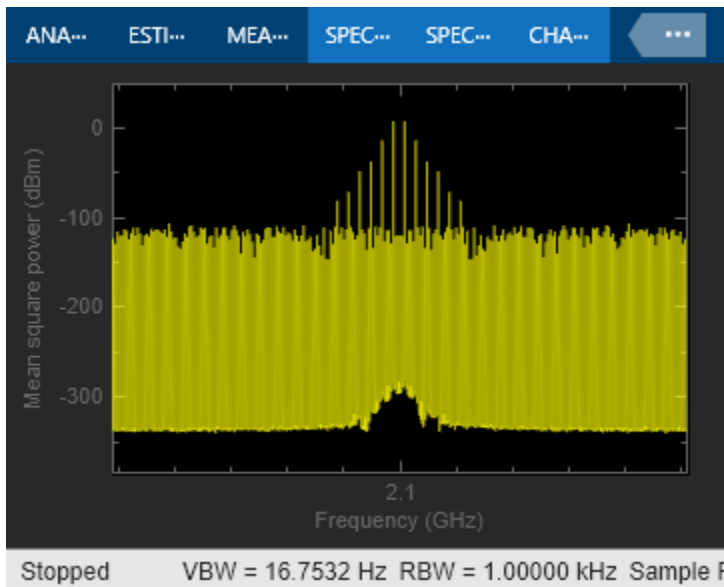
```
sim('rfb_user_defined_amp mdl');
```



Rerun the Simulation Using an Input Power Level of 8 dBm Per Tone

To increase the source power from 7 dBm to 8 dBm from a MATLAB script, first get the handle to the current Simulink® model workspace. Then set the appropriate workspace variable (power_in_dBm in this example) to the value 8. Rerun the simulation. Notice that 11th order intermods (outermost tones on the spectrum scope) increase by about 11 dB.

```
hws=get_param(bdroot, 'modelworkspace');
hws.assignin('power_in_dBm', 8);
sim('rfb_user_defined_amp mdl');
```

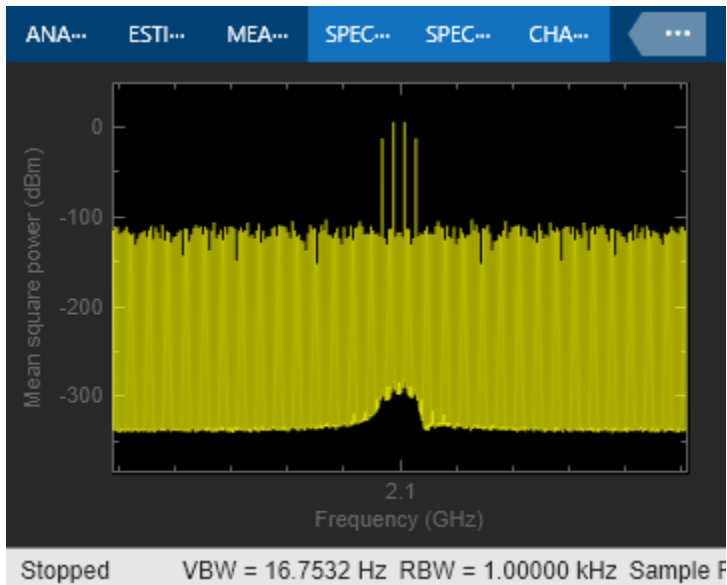


Truncate the Hyperbolic Tangent Series to Only the First Four Terms

Reset the input power to 7 dBm. Then, specify the power series for the hyperbolic tangent using only the first four terms of the series. Recalculate the output voltage values and the amplifier Pin-Pout

data. Rerun the simulation. The spectrum scope shows that only the third intermodulation products are produced. You can also set the stop time to inf, rerun the simulation, and experiment to see the effect of the slider gain (Blue box on the Simulink model).

```
hws.assignin('power_in_dBm', 7);
TanhSeries = [-1/3 0 1 0];
Vin = linspace(0.001,1,100); % volts
Vout = polyval(TanhSeries, Vin); % volts
amp.NonlinearData.Pin = {(Vin.^2)./Zref}; % watts
amp.NonlinearData.Pout = {(Vout.^2)./Zref}; % watts
sim('rfb_user_defined_amp mdl');
```



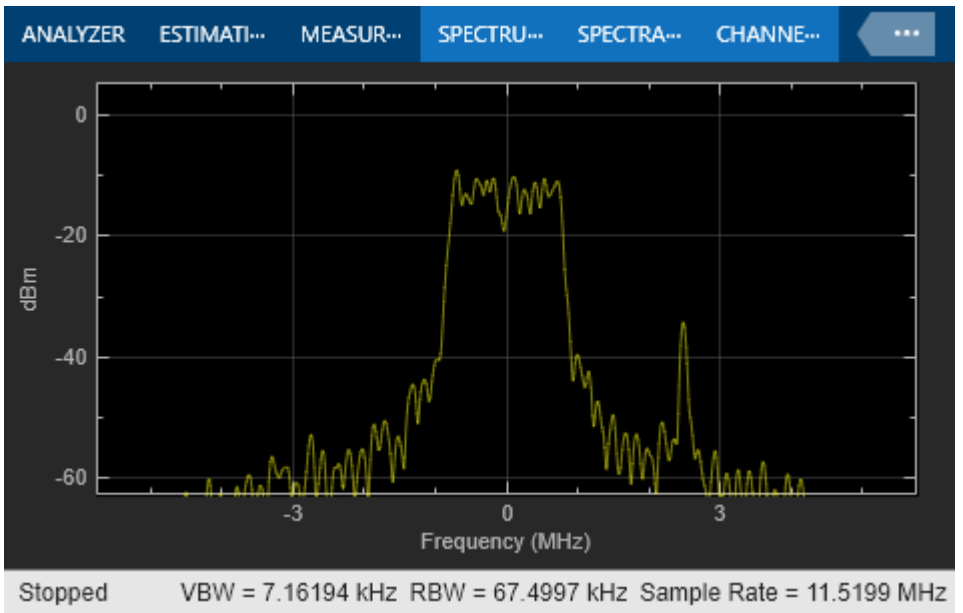
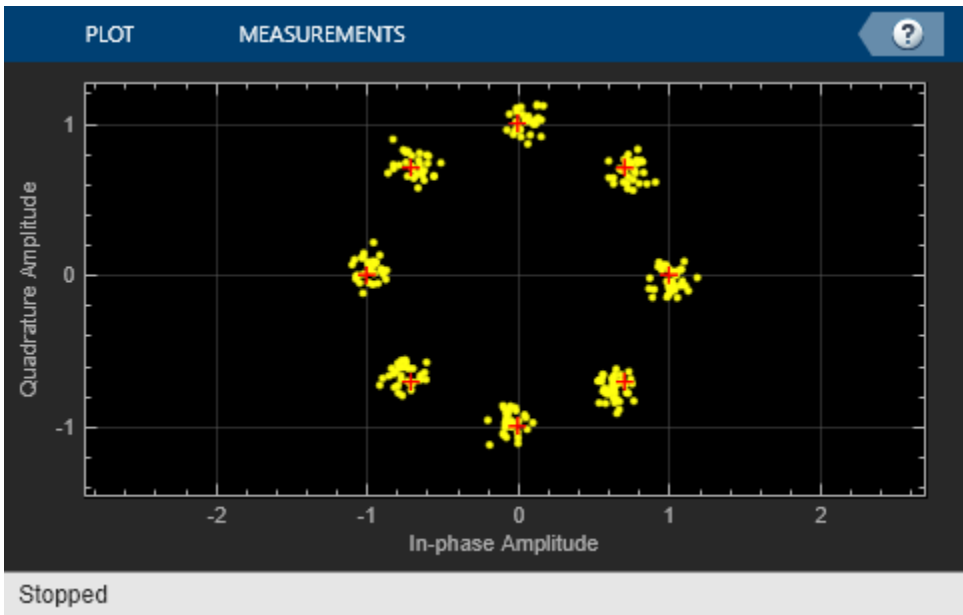
```
bdclose('rfb_user_defined_amp mdl');
close(fig);
```

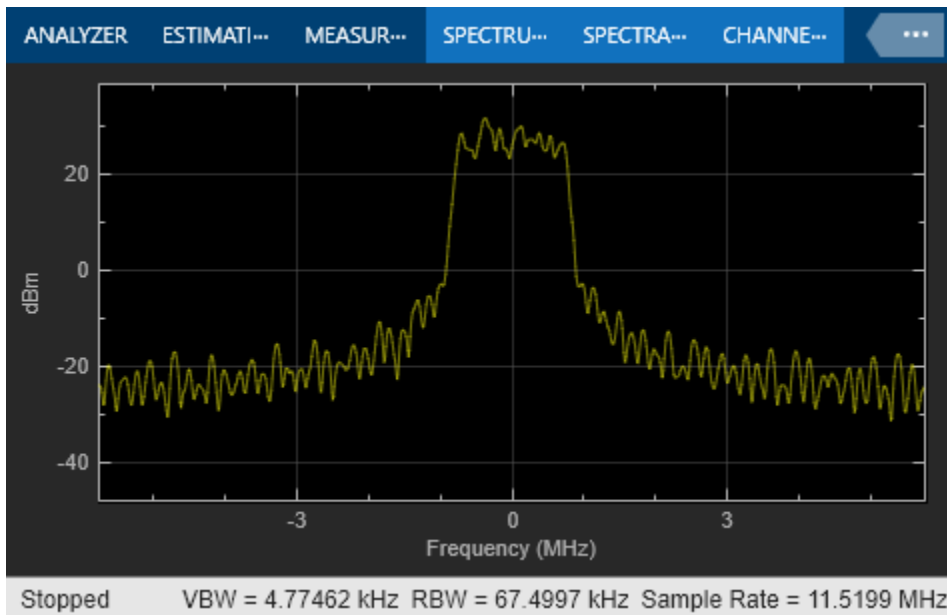
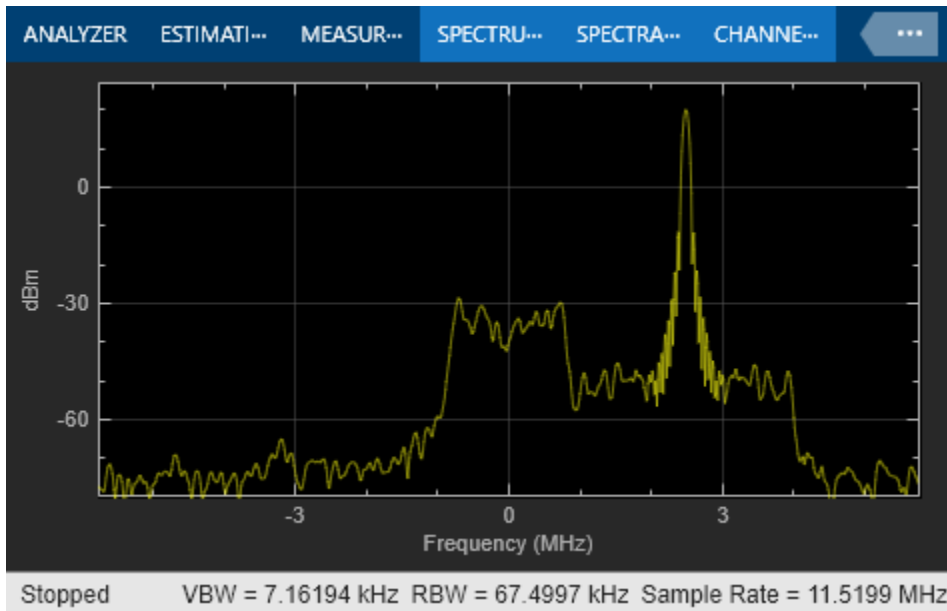
See Also

Output Port | Input Port | Configuration | `rfckt.amplifier`

Related Topics

“Effect of Nonlinear Amplifier on 16-QAM Modulation” on page 8-120



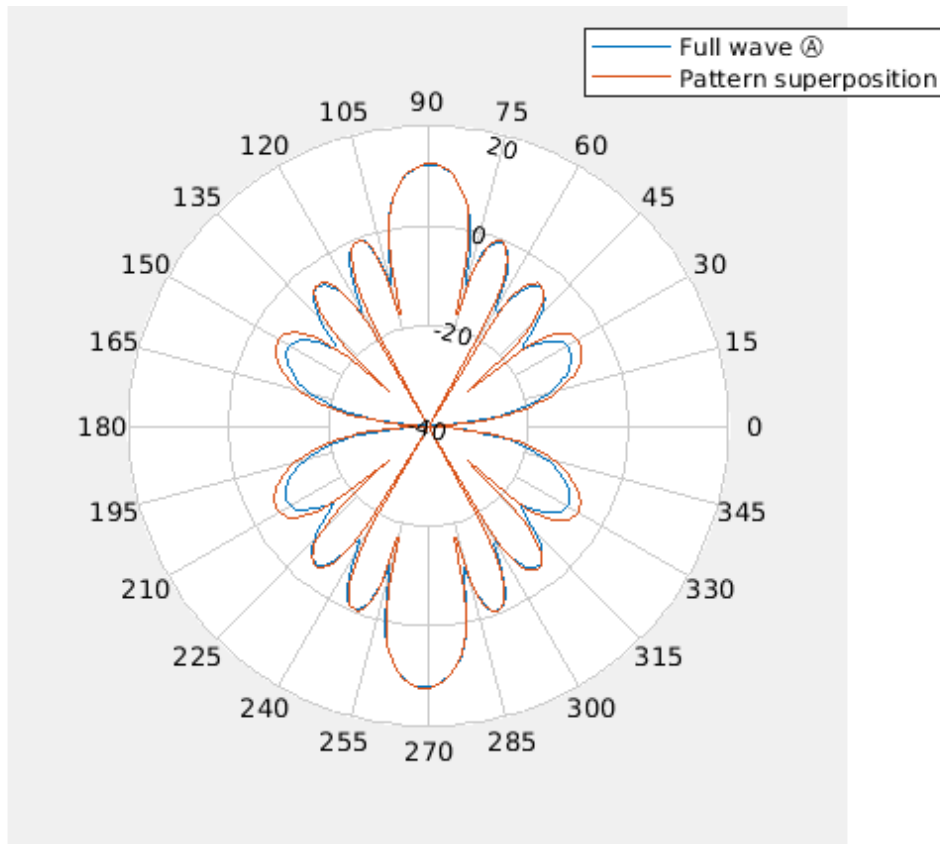


Design the Receiver Antenna Array

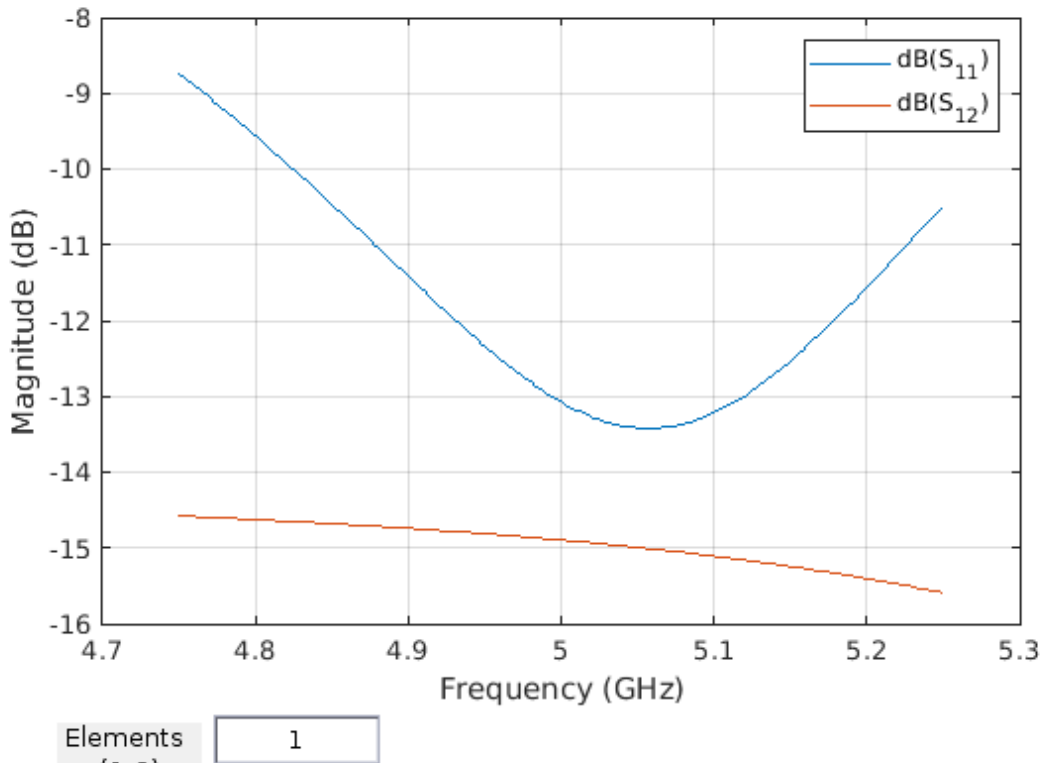
The receiver antenna array is designed using Antenna Toolbox™. The Antenna Toolbox helps you design an antenna at the desired operating frequency and verify that the pattern superposition of the isolated element is an acceptable approximation for the array simulation.

Script to design and verify the antenna array

As you can see, the antenna array consists of 8 dipole antennas resonating at 5 GHz. The comparison of the far-field radiation pattern of the array computed with full-wave analysis and pattern superposition of the isolated element shows modest differences:



However, the S-parameters show a non-negligible leakage between adjacent antennas.



RF Receiver

The receiver model includes:

- Model of receiver antenna array. The receiver antenna array is composed using 8 dipole antennas operating at 5 GHz. The array radiation pattern is modeled with Phased Array System Toolbox™ "Narrowband Rx array". The array is simulated using pattern superposition of the isolated element stored in the variable `P_antenna`, computed using Antenna Toolbox and the script. You can visualize the radiation pattern by clicking the **Analyze** button in the sensor array tab.
- Model of RF receiver. The RF receiver is composed with eight non-linear superheterodyne receivers and filters described with S-parameters. Each chain is designed with the RF Toolbox™ **RF Budget Analyzer** app as described in: RF Receiver Design example.
- Antenna array impedance is described with the eight-port S-parameters computed using Antenna Toolbox. The S-parameters capture the loading of the antenna array on the RF receiver as well as the coupling between the antenna elements. A lumped inductance for each receiver is used to retune the respective antenna.
- Eight 12-bit ADCs capturing the finite dynamic range of the data converters by modeling saturation and quantization.

DOA & Beamforming

The baseband receiver algorithm consists of four main elements in a closed feedback loop.

- Root MUSIC algorithm to determine the Direction of Arrival assuming that two signals are present. The two estimated DOA angles are passed to a state machine that determines which angle produces the higher Modulation Error Ratio (MER). This state machine includes some time delay in between state transitions to avoid decision jitter.
- MVDR Beamforming algorithm for the receiver to focus on the desired signal and suppress interference and noise from other directions. It uses the angle chosen by the Control Logic to maximize the MER.
- Signal Conditioning and estimation of the Modulation Error Ratio. The MER is used to determine which angle to select for the beamforming algorithm.

Related Topics

“Modeling RF mmWave Transmitter with Hybrid Beamforming” on page 8-149

Modeling RF mmWave Transmitter with Hybrid Beamforming

This example illustrates a methodology for system-level modeling and simulation of a 66 GHz QPSK RF transmit and receive systems with a 32-element hybrid beamforming antenna. The system includes RF imperfections, transmit array radiation effects, a narrowband receive array and a baseband receiver with corrections for system impairments and message decoding. The antenna beamforming direction is defined using azimuth and elevation angles and it is estimated in the RF receive antenna using a Root Music DOA algorithm.

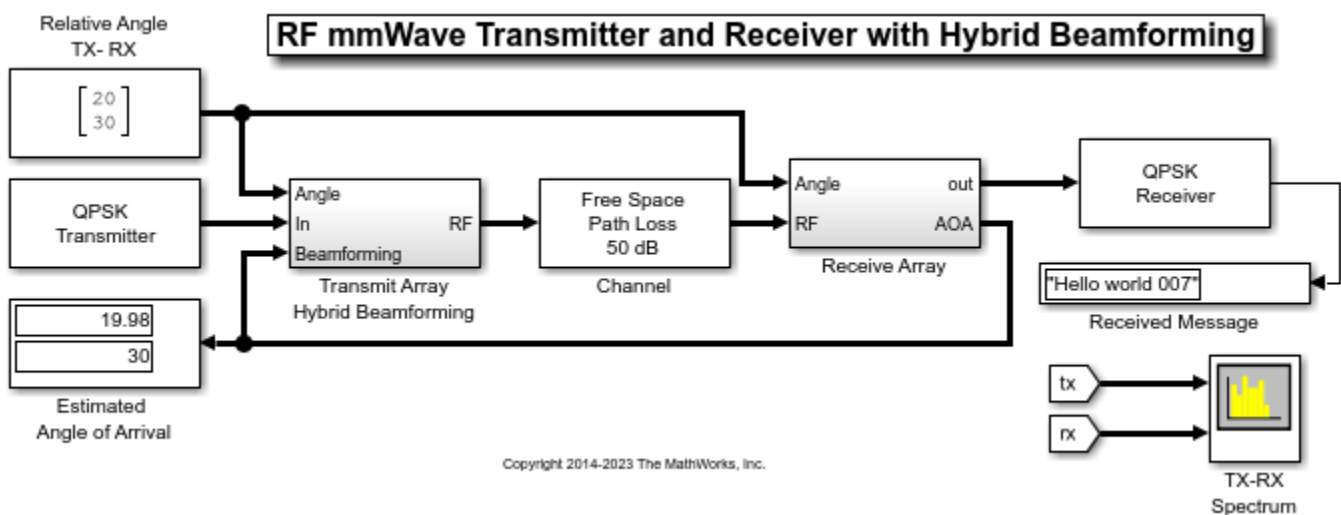
In the following sections you will see more details about the system design.

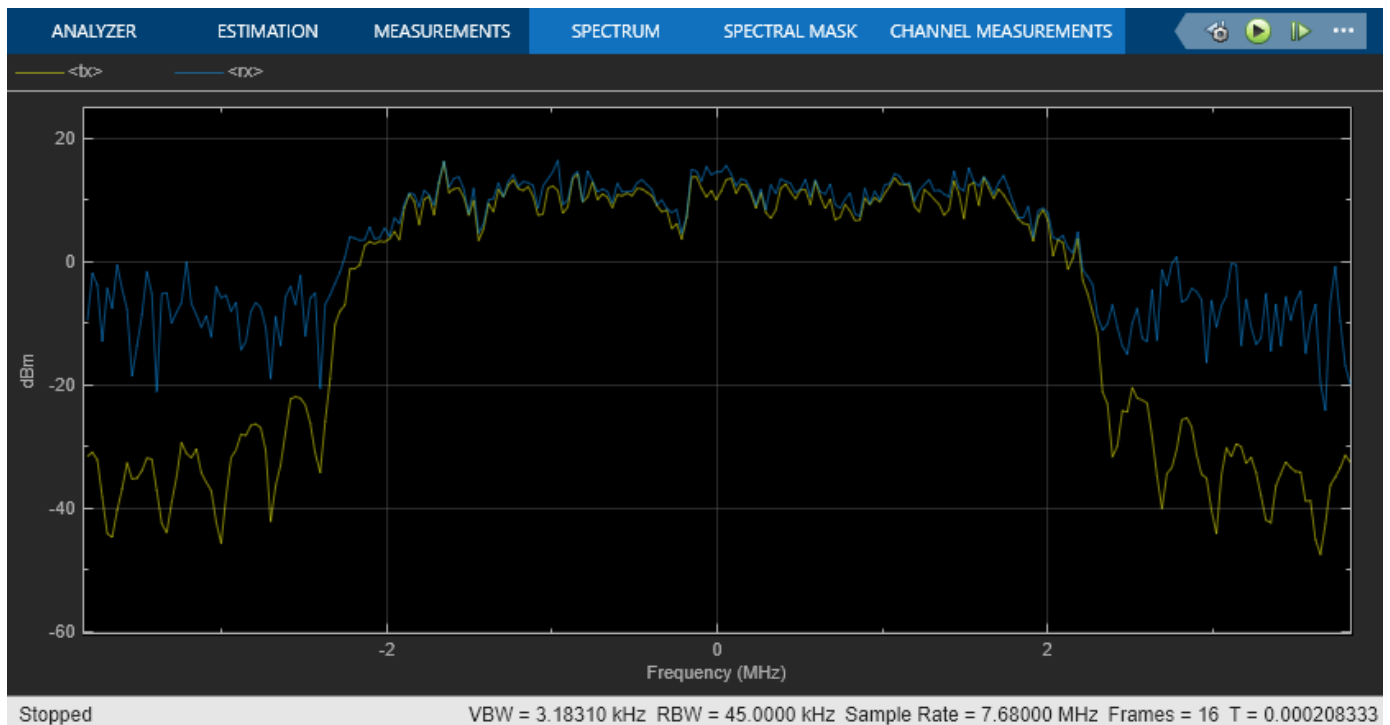
Model Description

The top-level of this example consists of five sub-system blocks, a block to control the relative angle between transmitter and receiver, and 2 displays:

- A QPSK baseband transmitter encodes the message "Hello World ###".
- An RF transmitter with IQ modulation, mixing, amplification and hybrid beamforming with control circuitry. The RF transmitter model includes RF imperfections such as noise, non-linear effects and antenna element coupling.
- An ideal channel attenuating the transmitted signal with a free space path loss model.
- An RF receiver with two narrowband receive array antennas, receiver gain and SNR, 12-bit ADC with finite dynamic range, and two root MUSIC algorithms for angle of arrival estimation along azimuth and elevation.
- A QPSK receiver, including carrier and frame synchronization, demodulation and data decoding.
- A block where the user sets the relative angle between the transmitter and the receiver.
- A spectrum analyzer scope comparing normalized transmitted and received signals and a display for the received message.

```
model = 'simrfv2_qpsk';
open_system(model)
sim(model)
```

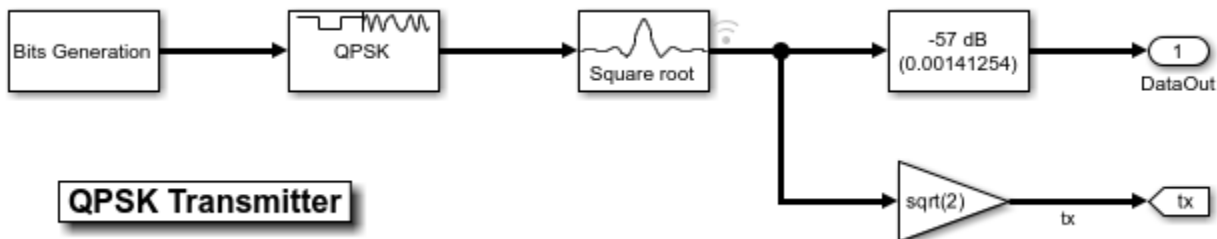




QPSK Transmitter

The QPSK transmitter includes a Bit Generation subsystem, a QPSK Modulator block, a Raised Cosine Transmit Filter block for pulse shaping, and a Gain block. The Bit Generation subsystem generates frames. Each frame contains 26 header bits followed by a payload of 174 bits, 105 bits for the message 'Hello world ###' and 69 random bits. The payload is scrambled to guarantee a balanced distribution of zeros and ones for the timing recovery operation in the receiver model.

```
open_system([model '/QPSK TX'], 'force')
```

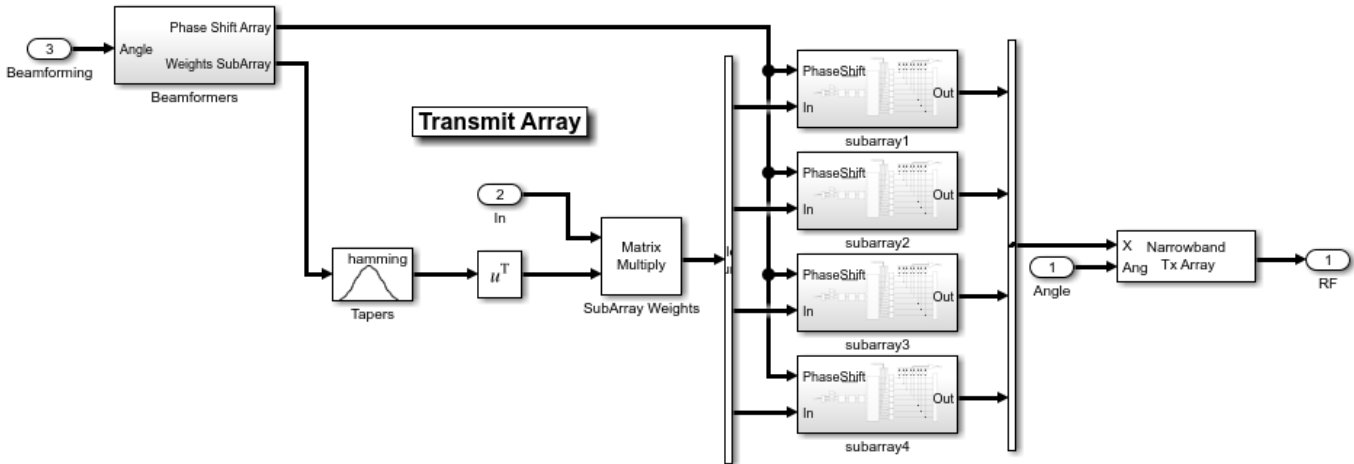


RF Transmitter

The RF transmitter is composed of three sections: array beamformers, a hybrid beamforming antenna and a Narrowband Transmit Array block. The 32-element hybrid beamforming antenna is divided in 4 sub-arrays. Each subarray consists of 8 RF transmitters operating at 66 GHz. The antennas are microstrip patches. These antenna elements and the subarrays have been designed and verified with a MATLAB script that uses Antenna Toolbox™.

The far field antenna array gain is computed with the Phased Array System Toolbox™ Narrowband Transmit Array block. The computed radiation pattern is the superposition of the fields generated by the isolated microstrip patches.

```
open_system([model '/Transmit Array Hybrid Beamforming'])
```



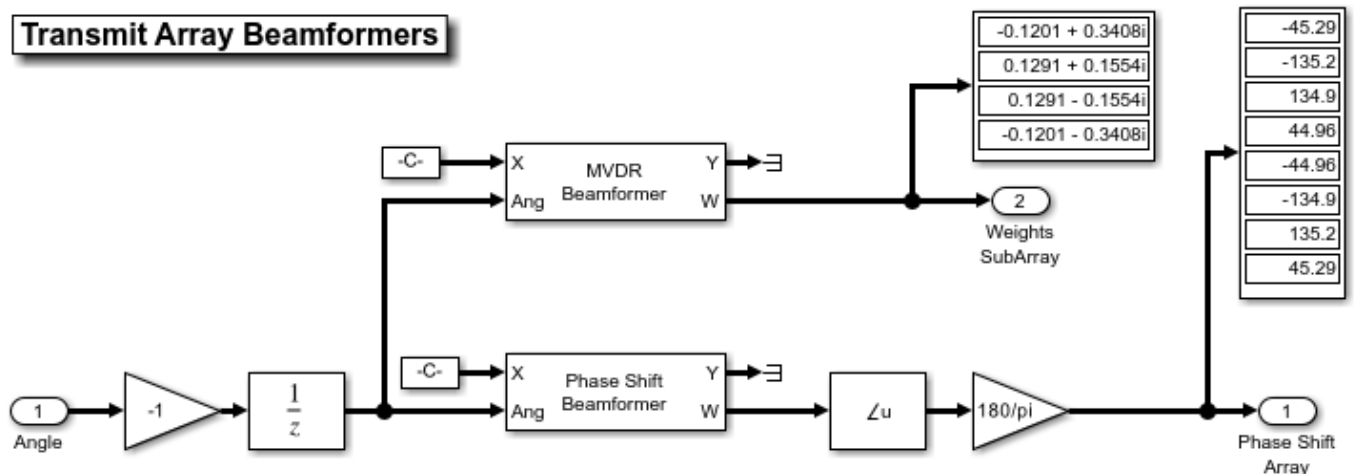
Transmit Array Beamformers

The transmit array is steered towards the direction estimated by the receiver. For demonstration purposes, two different beamforming algorithms are used to compute the weights applied to the four subarrays and to the elements of each subarray.

The subarrays weights are computed with an MVDR beamformer. A complex multiplication in the MVDR beamformer combines the transmitted signal and subarrays weights, steering the transmitted signal along the azimuth direction. Tapering is used to reduce the effects of grating lobes.

The phase shifts applied to the eight subarray elements are computed with a phase shifter beamforming algorithm. The four subarrays apply the same phase shifts that steer the transmitter along the elevation direction.

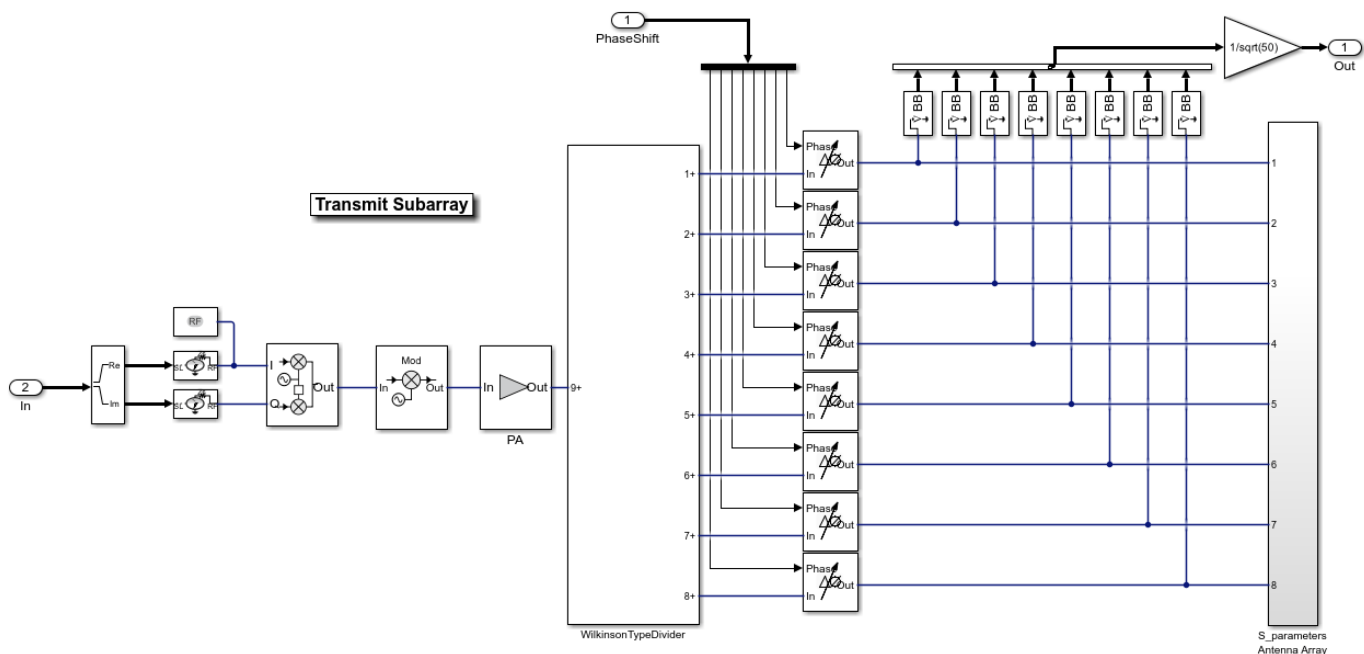
```
open_system([model '/Transmit Array Hybrid Beamforming/Beamformers'])
```



Transmit Subarrays

The four transmit subarrays are identical. Each subarray performs upconversion to 66 GHz using a quadrature modulator and a 5 GHz local oscillator followed by a superhet modulator that consists of a 61 GHz local oscillator, an image filter and a channel select filter. Impairments such as noise, I/Q imbalance, LO leakage and non-linearities are included in the appropriate subarray components. A non-linear power amplifier increases the transmitter gain, and a Wilkinson type 1-to-8 power divider followed by variable phase shifters connects the PA to 8 antennas. The eight variable phase shifters are used to steer the beam. The loading of the antenna subarray and the coupling in between the antenna elements is modeled by its S-parameters.

```
open_system([model '/Transmit Array Hybrid Beamforming/subarray1'])
```



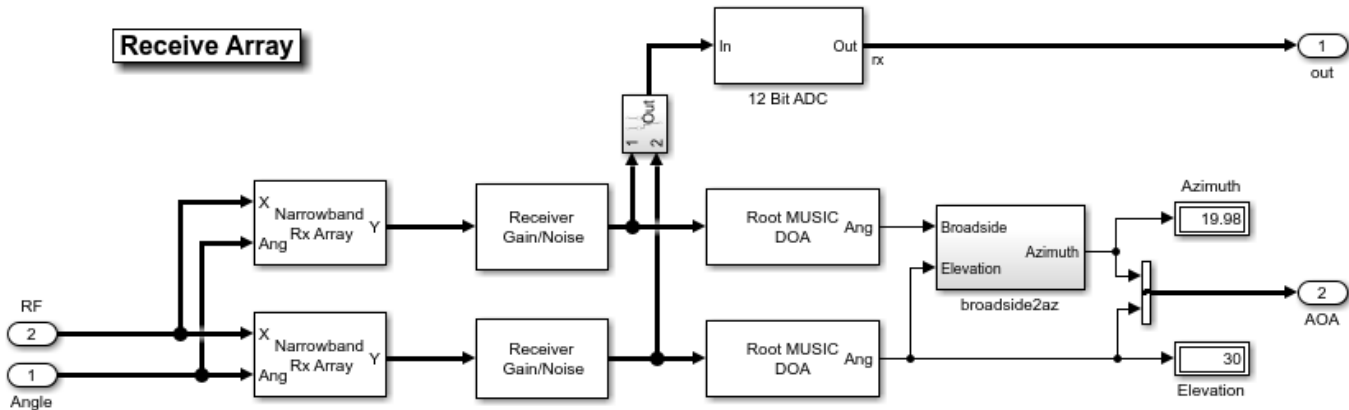
Receive Array

The receiver is modeled at a higher abstraction level compared to the transmitter. The receiver uses two orthogonal linear arrays, each with four isotropic antenna elements. The arrays are used to provide spatial diversity for the identification of the angle of arrival. The receiver does not implement any beamforming algorithm.

The receiver finite gain and SNR is modeled for each of the received signals followed by a 12-bit ADC with finite dynamic range including saturation and quantization effects.

Two root MUSIC algorithms are used to estimate the direction of arrival using the linear array signals. Each algorithm operates across one dimension, thus together can estimate the transmitter position in terms of azimuth and elevation angles.

```
open_system([model '/Receive Array'])
```

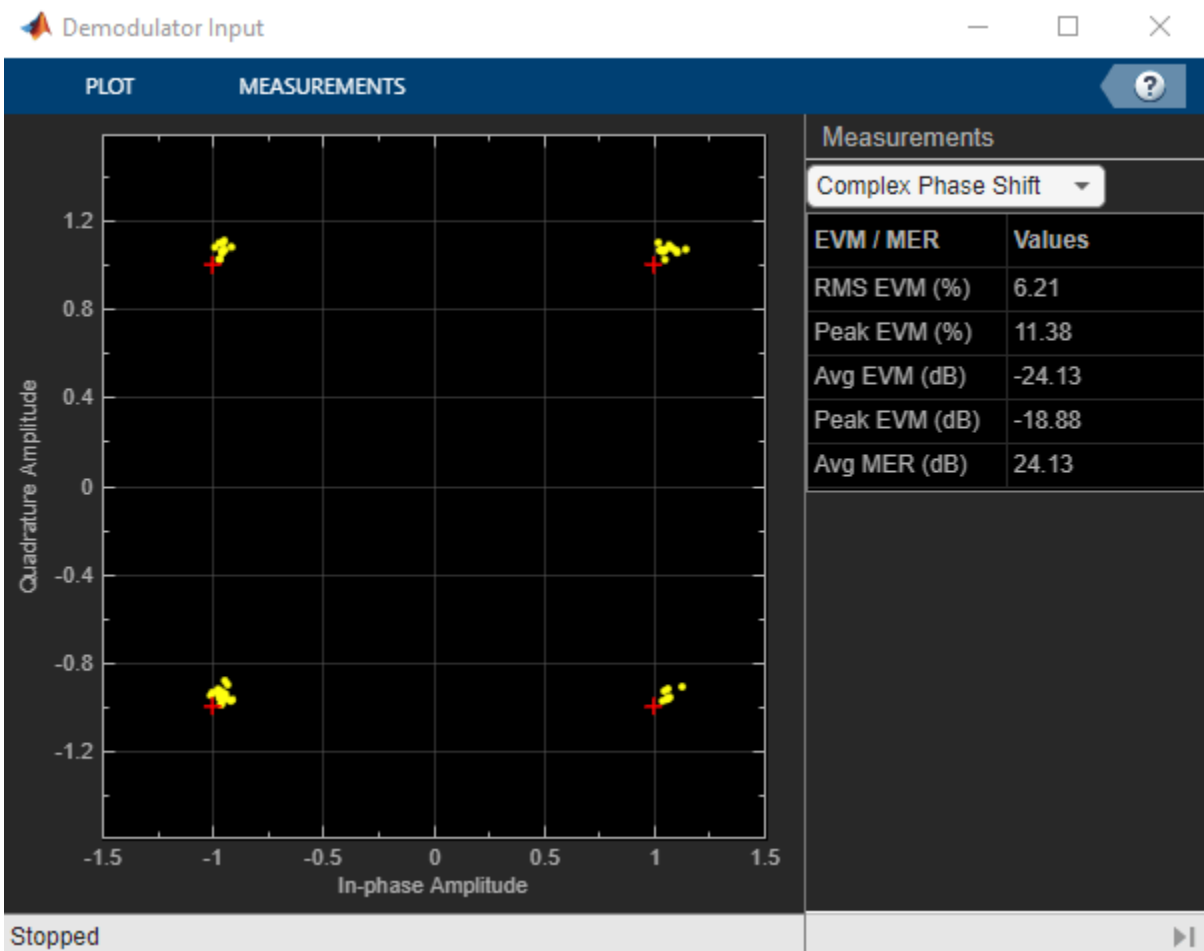
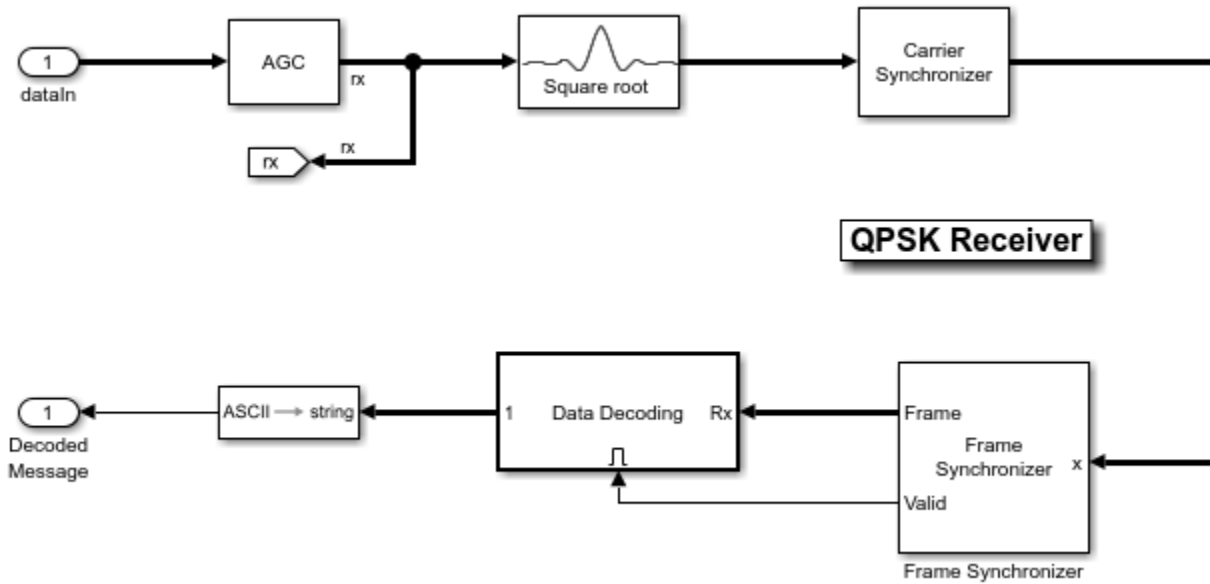


QPSK Receiver

The QPSK receiver from the Communications Toolbox™ example “QPSK Transmitter and Receiver” (Communications Toolbox) is used in this example with modification. These modifications remove blocks from this receiver when the signal impairment is absent.

- The AGC controls and stabilizes the received signal amplitude which affects the accuracy of the carrier symbol synchronizer.
- The Raised Cosine Receive Filter provides matched filtering for the transmitted waveform.
- The Carrier Synchronizer Block performs fine frequency compensation.
- The Frame Synchronizer block uses the known frame header (QPSK-modulated Barker code) to correlate against the received QPSK symbols to find the location of the frame header. The block uses this frame location information to align the frame boundaries. The second output of the block is a boolean scalar indicating if the first output is a valid frame with the desired header and if so, enables the Data Decoding subsystem to run.
- The Data Decoding enabled subsystem performs phase ambiguity resolution, demodulation and text message decoding.

```
open_system([model '/QPSK Receiver'])
```




```
bdclose(model)
clear model;
```

See Also

[IQ Demodulator](#) | [Mixer](#) | [Power Amplifier](#)

Related Examples

- “Modeling and Simulation of MIMO RF Receiver Including Beamforming” on page 8-143
- “Wireless Digital Video Broadcasting with RF Beamforming” on page 8-156

Wireless Digital Video Broadcasting with RF Beamforming

This example shows how to model a digital video broadcasting system which includes a 16 antenna phased array receiver operating at 28 GHz. The baseband transmitter, receiver and channel are realized with Communications Toolbox™. The RF receiver is implemented with the RF Blockset™ Circuit Envelope library, and the receive phased array antennas are constructed using Phased Array System Toolbox™. The 4 x 4 planar phased array feeds a 16 channel receive module that includes phase shifters to enable RF beamforming.

System Architecture

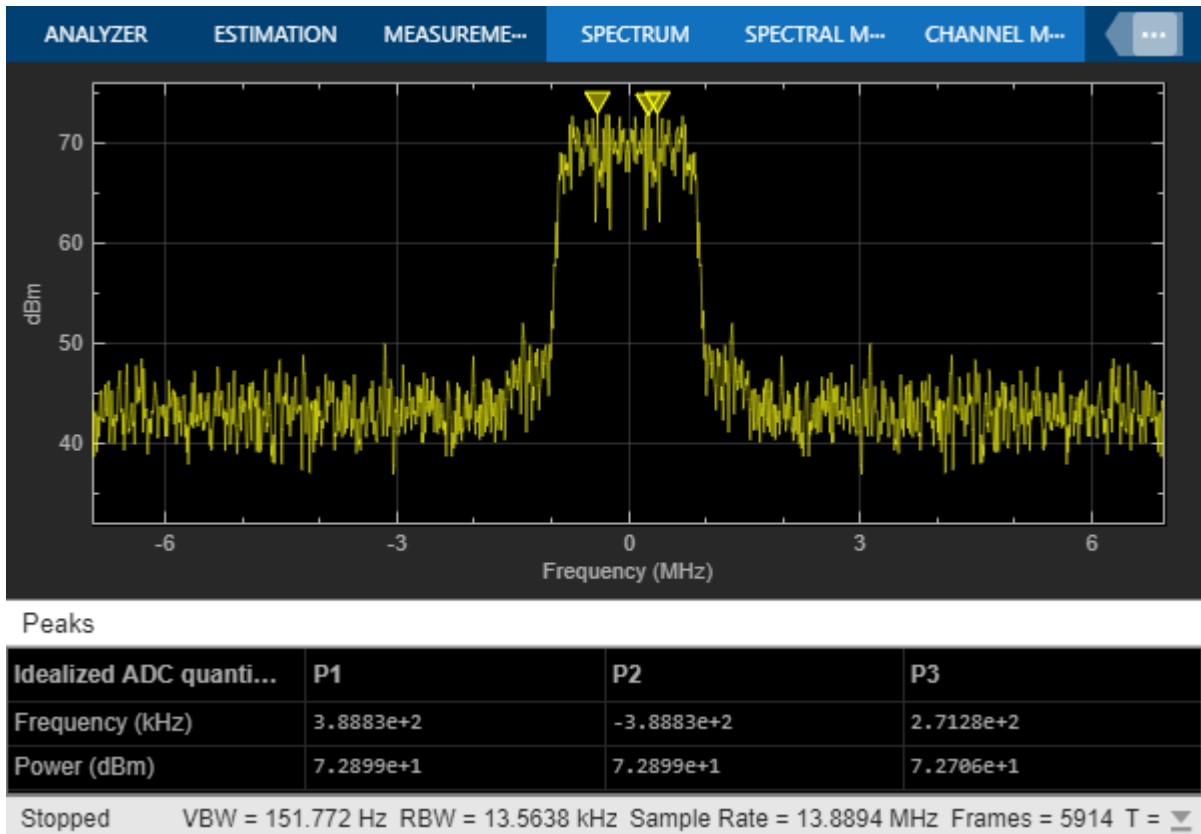
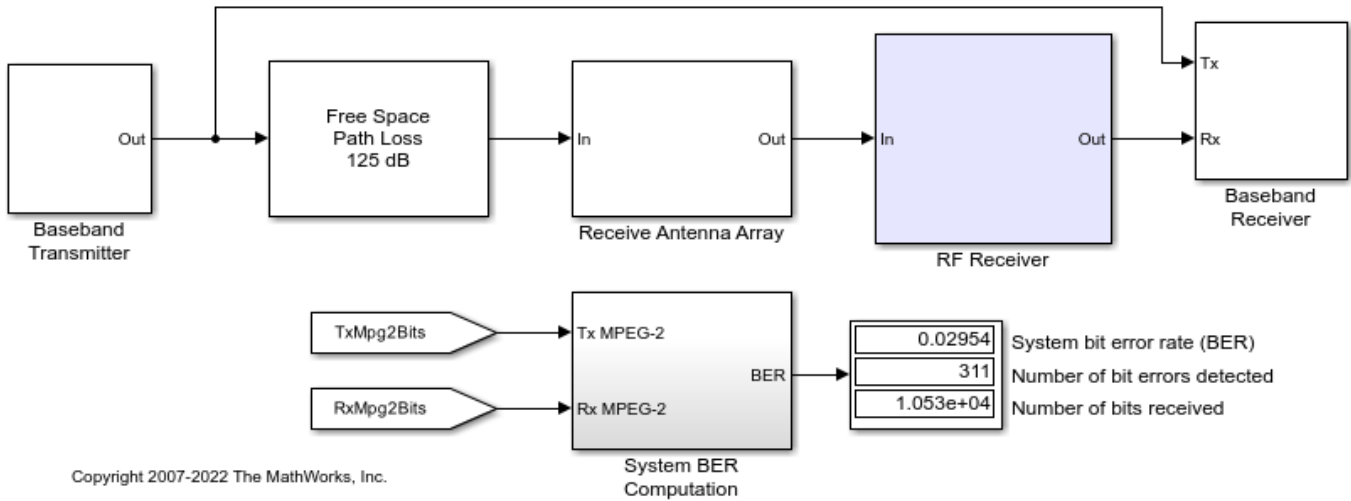
The system consists of:

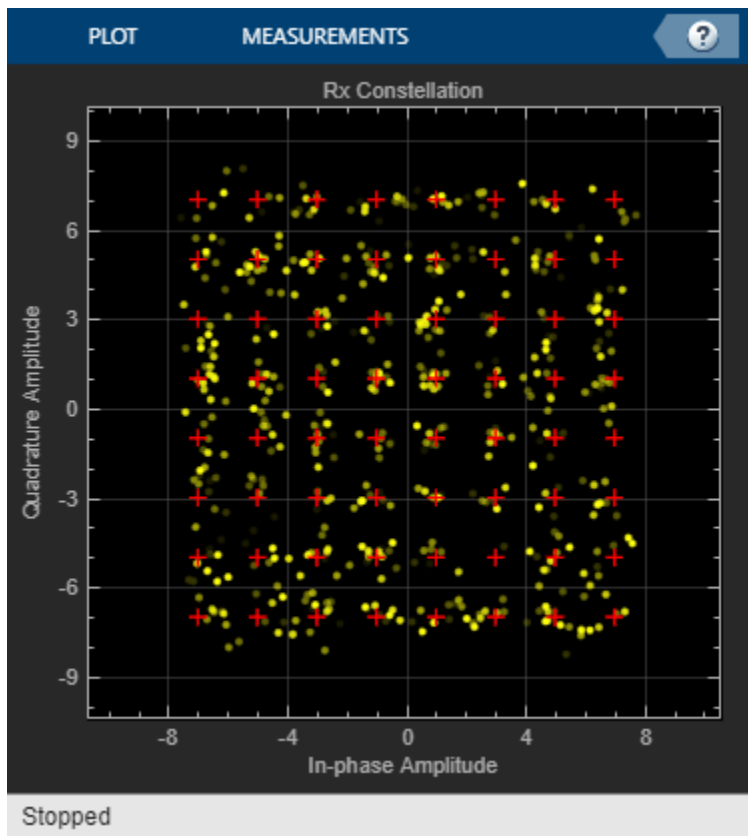
- A Baseband Transmitter subsystem that is responsible for generating a 64-QAM signal occupying 2 MHz of bandwidth that adheres to the DVB-C standard.
- Channel effects in the form of path loss.
- A 16 element phased array receiver arranged in a 4 X 4 rectangular grid. This includes design parameters for operating frequency, element radiation pattern, and receive direction.
- An RF receiver module consisting of 16 paths combined with a network of 2:1 power combiners and then downconverted to baseband. Each path includes LNAs and variable phase shifters for RF beamforming. The network of 2:1 power combiners is constructed twice to emulate a typical design process. The initial design employs ideal Wilkinson power dividers as behavioral combiners, while the second implementation uses actual hardware modeled with S-Parameters blocks and measured data supplied via a Touchstone™ file (wireless.s3p).
- A Baseband Receiver subsystem that is responsible for extracting the transmitted signal. The receiver includes simple models for correcting phase offsets and gain control effects.

Diagnostics are available at various stages in the system using the received constellation, the bit error rate calculation, and the received spectrum.

```
model_ideal = 'simrfv2_wirelessdvh_beamform_ideal';  
open_system(model_ideal)  
sim(model_ideal)
```

Wireless Digital Video Broadcasting with RF Beamforming and Ideal Combiner





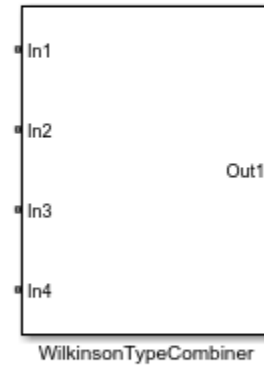
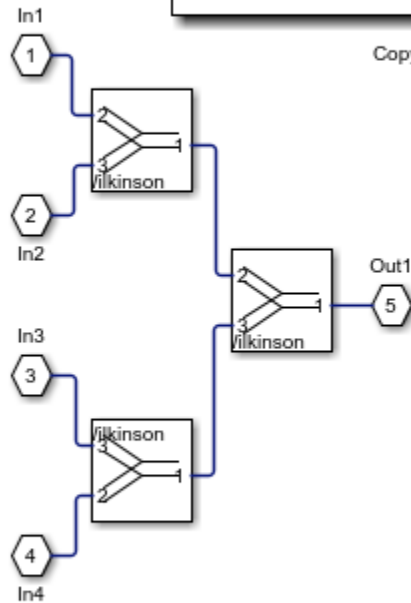
Designing with Ideal Components

An initial design may use ideal components to speed up the overall design process. For example, ideal Wilkinson dividers from the RF Blockset Junctions library can be used to build a combiner system in the Receive Antenna Array. This combiner system can be consolidated into a 17-port S-parameters block to improve simulation performance. A simplified example of consolidation is shown where the 4:1 combiner system on the left is replaced with the 5-port S-parameters block on the right. The S-parameters block entries are calculated in the WilkinsonTypeCombiner block mask initialization commands.

```
model_combiner = 'simrfV2_wirelessdvb_beamform_prototype_combiner';
open_system(model_combiner)
```

4-Input and Single Output Implementation Examples for Wilkinson Ideal Combiner Networks

Copyright 2020 The MathWorks, Inc.



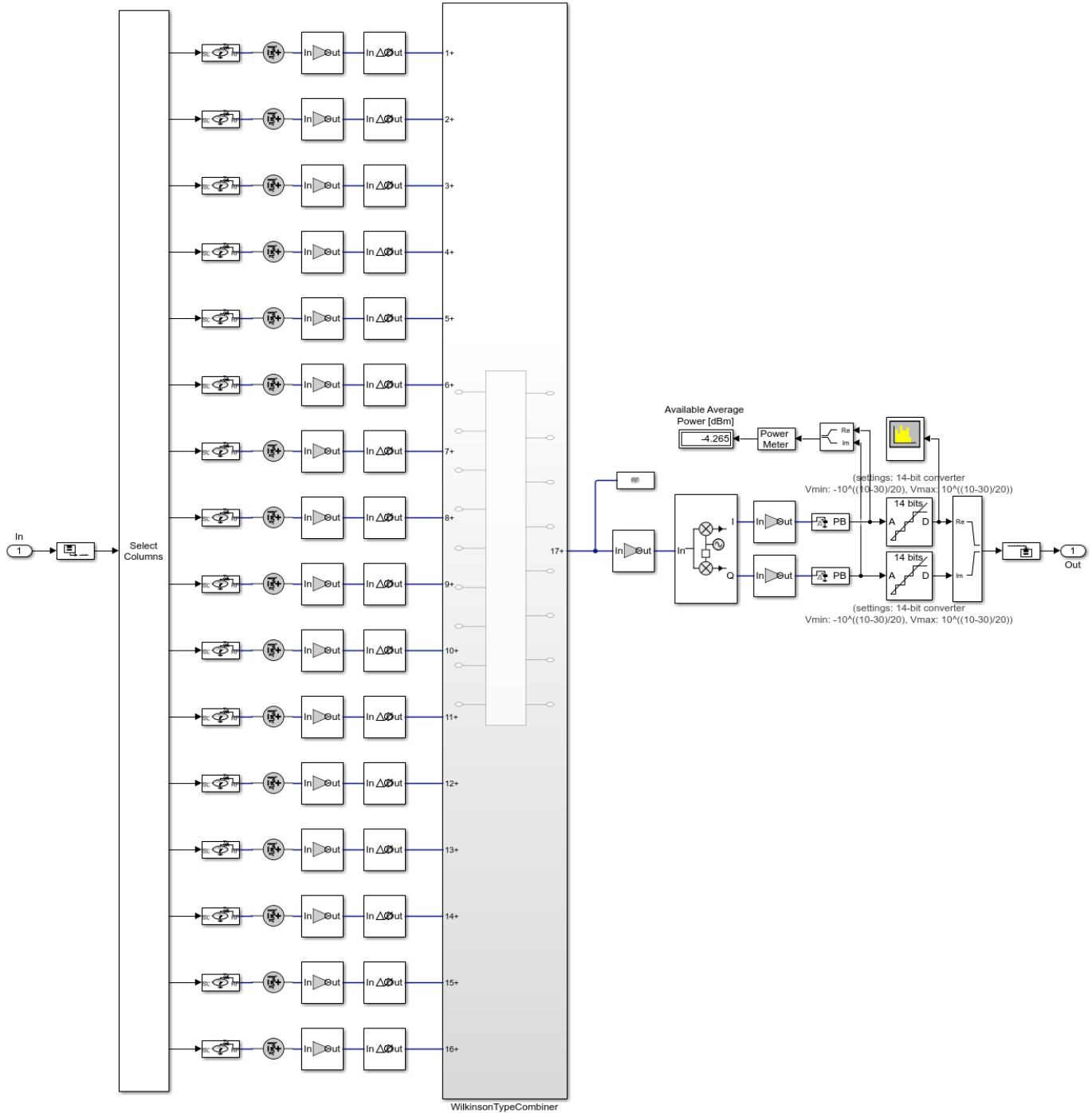
**Network of
Ideal Wilkinson Combiners**

**5-Port
S-parameter Block**

Each input channel branch of the RF Receiver employs a separate amplifier to introduce thermal noise and non-linearities. The quadrature demodulator following the combiner performs direct downconversion and includes its non-linearities, LO leakage, I/Q mismatch, and noise impairments.

```
open_system([model_ideal '/RF Receiver'], 'force')
```

RF RECEIVE MODULE FOR A 16 CHANNEL PHASED ARRAY with Ideal 16:1 Combiner



```

bdclose(model_combiner)
bdclose(model_ideal)
clear model_ideal
    
```

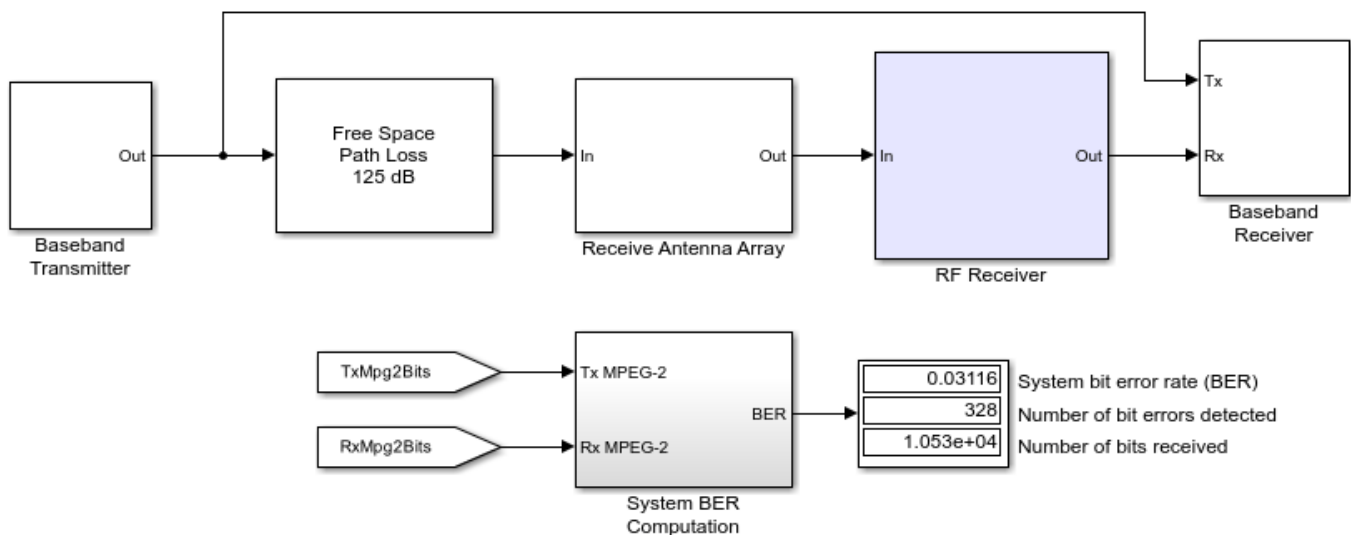
Designing with Real Components

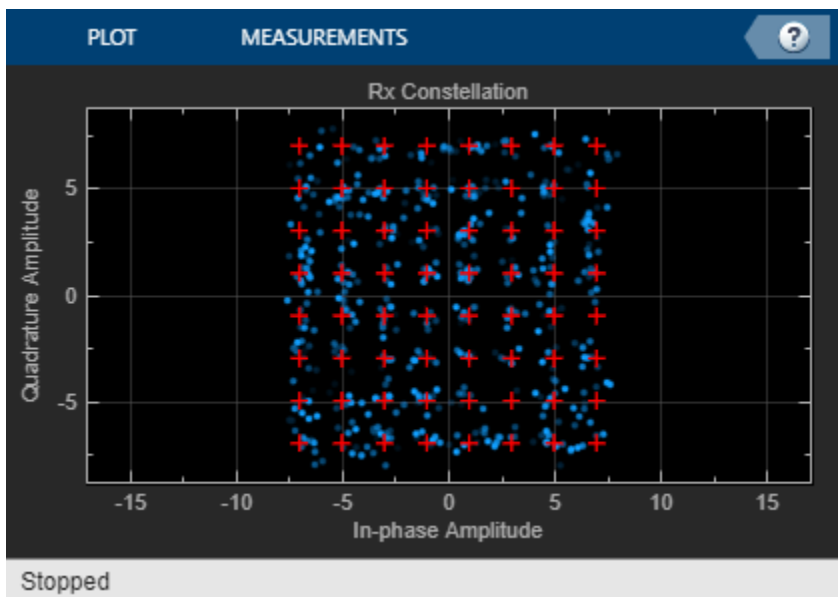
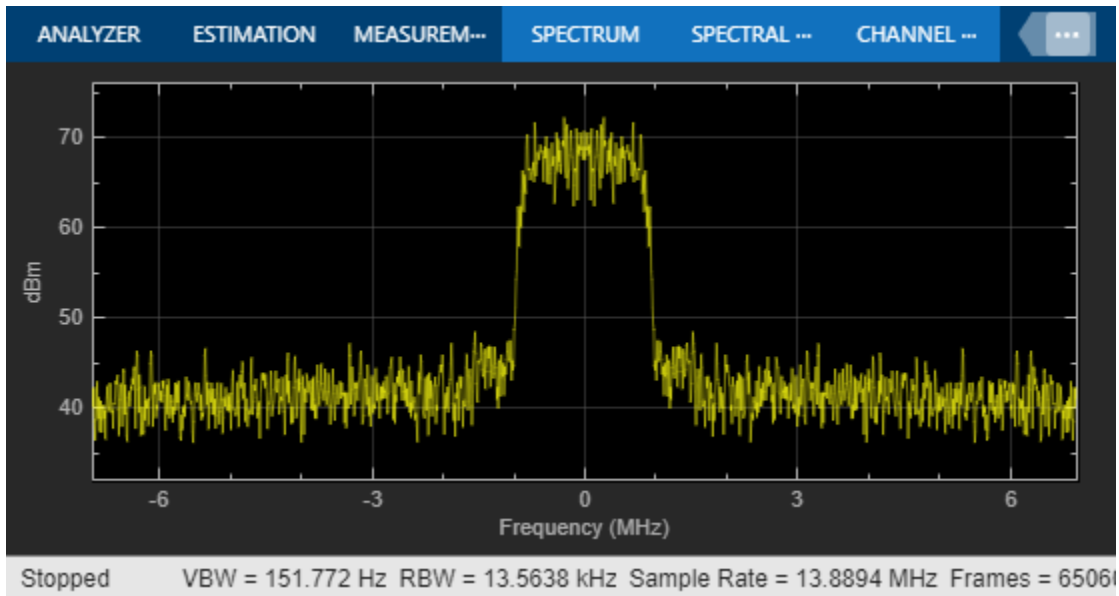
Use S-Parameters blocks to model a real combiner system. There are several options available to characterize the behavior of each individual block in the combiner system; one approach utilizes a data file directly while another approach provides a rational model of the data. For the latter approach, utilize the `rational` function in RF Toolbox™, save the resulting parameters in the base workspace and use them in the S-Parameters blocks. For this example, the measured data is described in the Touchstone file `wireless.s3p` and used directly. To improve simulation performance the combiner system is replaced with a 17-port S-Parameters block. The S-Parameters block entries are calculated using the function `simrfV2_wirelessdvb_beamforming_findcombinerspars` in the Initialization commands of the RF Receiver subsystem mask.

```
model = 'simrfV2_wirelessdvb_beamforming';
open_system(model)
sim(model)
```

Wireless Digital Video Broadcasting with RF Beamforming

Copyright 2007-2022
The MathWorks, Inc.



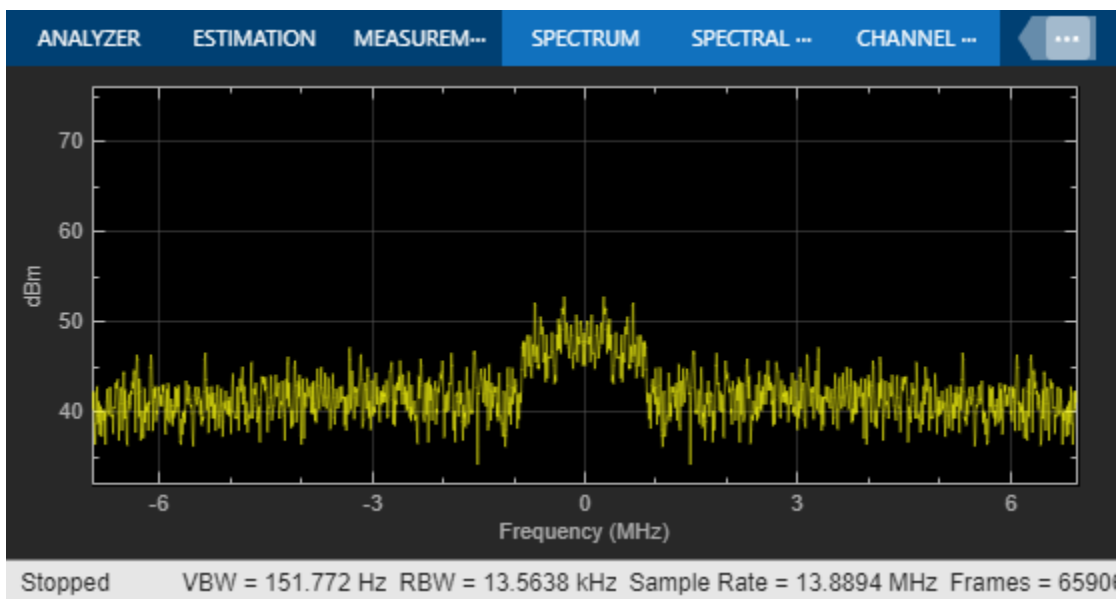
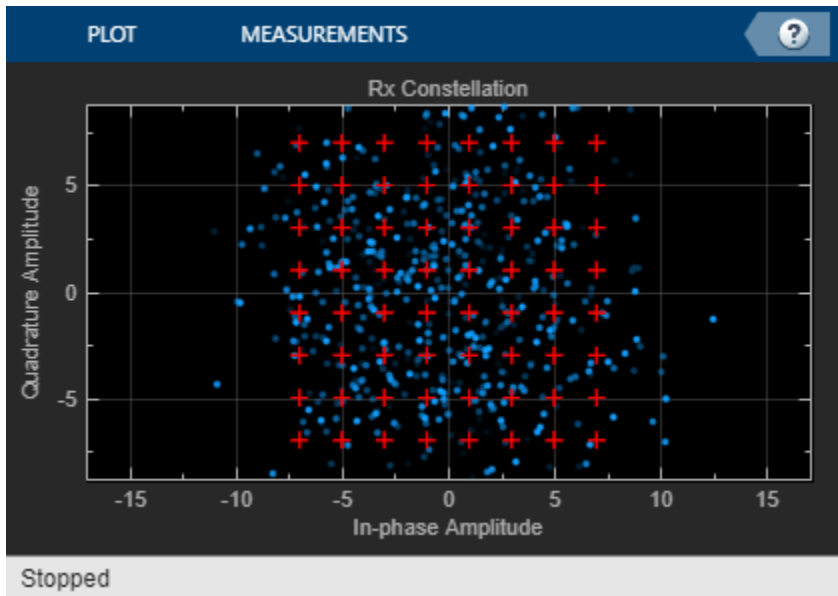


The transmit side planar array is chosen to have 16 elements and transmits along the main beam (azimuth = 0 deg. and elevation = 0 deg.) at a frequency of 28 GHz. An isotropic radiation pattern is chosen for each element. Note that the power dividers introduce a phase shift at 28 GHz. This is estimated and corrected in the Baseband receiver subsystem.

Modify the Receive Direction and Simulate

Modify the receive direction by changing the **Receive Direction** mask dialog parameter of the 16-element Receive Antenna Array. The angle chosen decreases the signal strength due to the proximity of a null in the array radiation pattern.

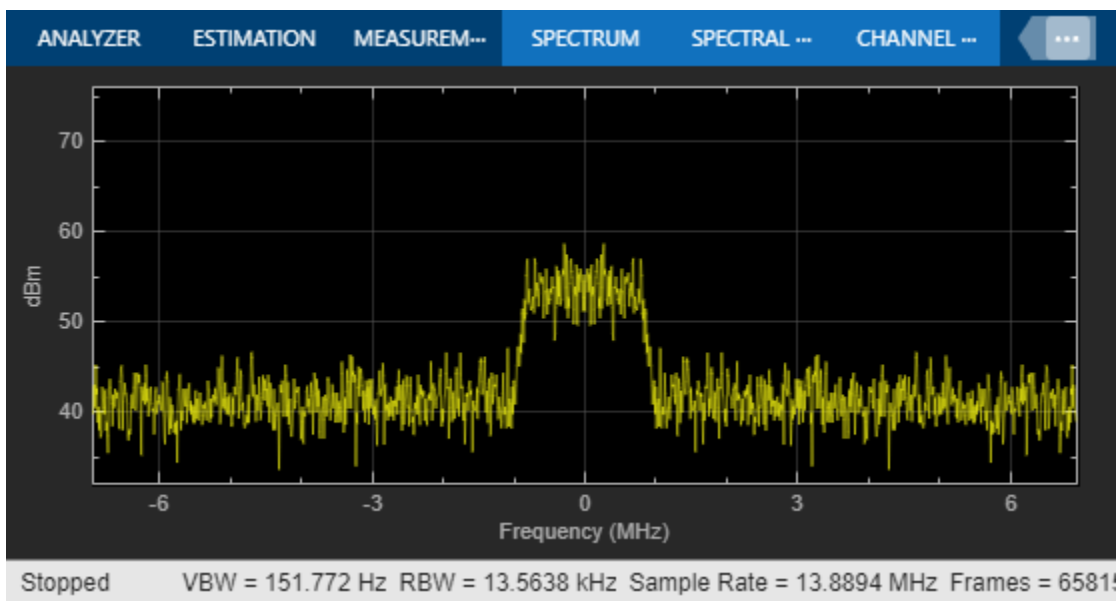
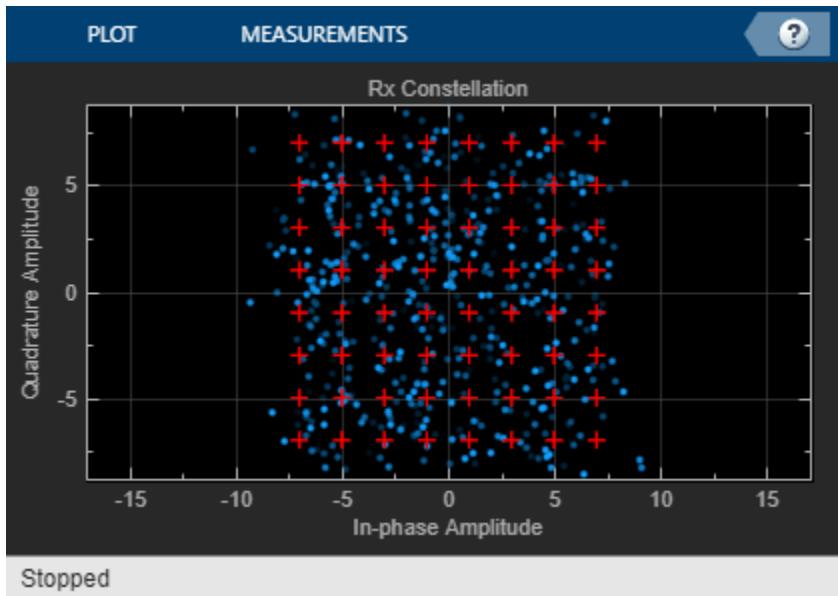
```
open_system(model)
set_param([model '/Receive Antenna Array'],'RecDir','[20;25]')
sim(model)
```

Improve RF Reception with Beamforming

Modify the **Beamforming direction** parameter for the 4 X 4 phased array on the receive side. This mask parameter will automatically adjust the phase shift of each channel in the RF Receiver subsystem. Run the simulation to observe an increase in the received signal level.

```
open_system(model)
set_param([model '/RF Receiver'], 'BeamDir', '[20;25]')
sim(model)
```



```

bdclose(model)
clear model
    
```

References

S. Emami, R. F. Wiser, E. Ali, M. G. Forbes, M. Q. Gordon, X. Guan, S. Lo, P. T. McElwee, J. Parker, J. R. Tani, J. M. Gilbert,, and C. H. Doan, "A 60 GHz CMOS Phased-Array Transceiver Pair for Multi-Gb/s Wireless Communications," in IEEE Int. Solid-State Circuits Conf. Tech. Dig., Feb. 2011, pp. 164-165

See Also

More About

- "Modeling RF mmWave Transmitter with Hybrid Beamforming" on page 8-149
- "Modeling and Simulation of MIMO RF Receiver Including Beamforming" on page 8-143

Top-Down Design of an RF Receiver

This example designs an RF receiver for a ZigBee®-like application using a top-down methodology. It verifies the BER of an impairment-free design, then analyzes BER performance after the addition of impairment models. The example uses the RF Budget Analyzer App to rank the elements contributing to the noise and nonlinearity budget.

Design specifications:

- Data rate = 250 kbps
- OQPSK modulation with half sine pulse shaping, as specified in IEEE® 802.15.4 for the physical layer of ZigBee
- Direct sequence spread spectrum with chip rate = 2 Mchips/s
- Sensitivity specification = -100 dBm
- Bit Error Rate (BER) specification = $1e-4$
- Analog to digital converter (ADC) with 10 bits and 0 dBm saturation power

To create fully standard-compliant ZigBee waveforms, you can use the *Communications Toolbox Library for ZigBee and UWB Add-on*.

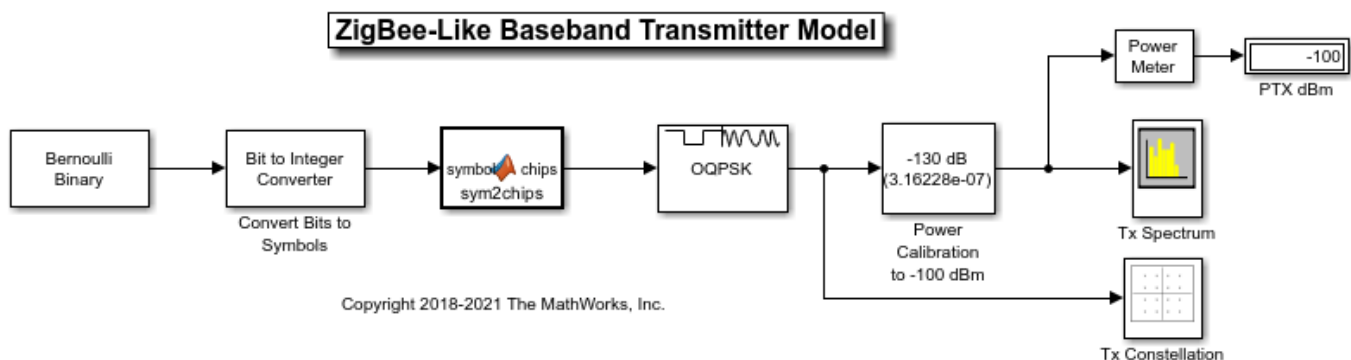
This example guides you through the following steps:

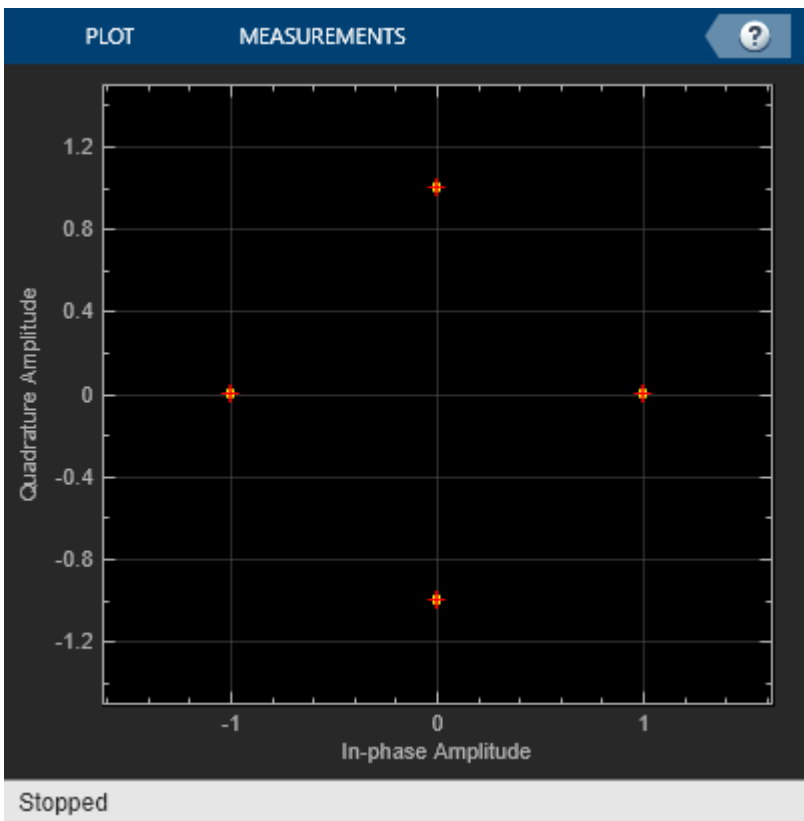
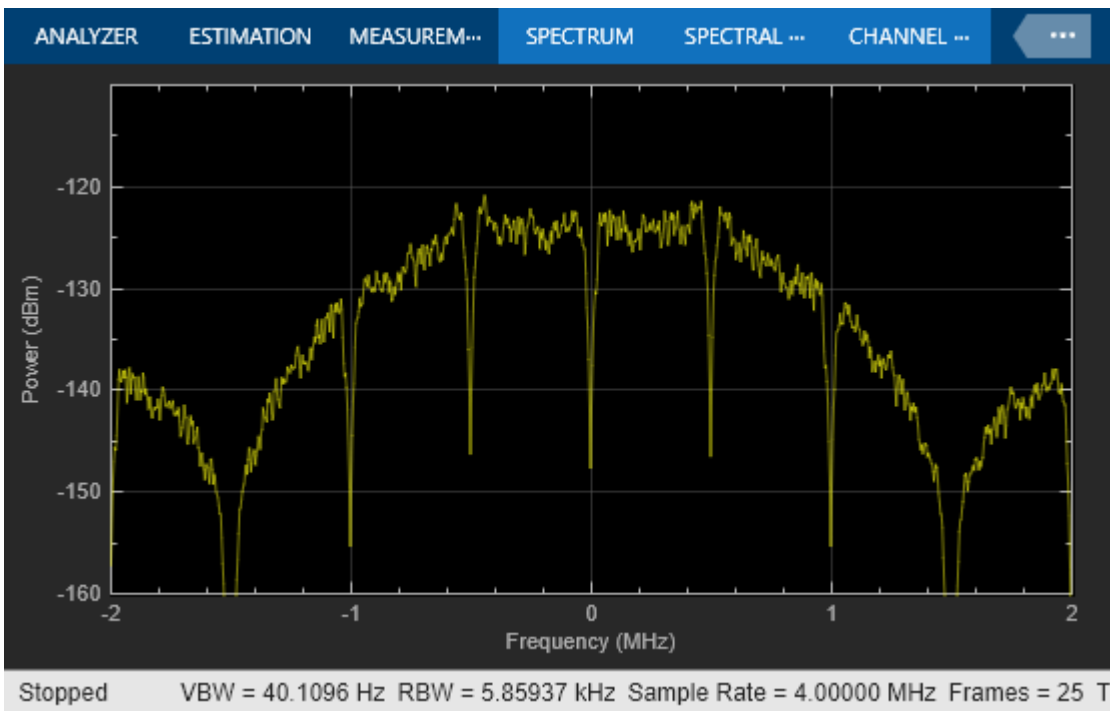
- Develop the baseband transmitter model for waveform generation
- Determine SNR specification to achieve the $1e-4$ BER from a link-level idealized baseband model
- Derive RF subsystem specifications from equivalent-baseband model of RF receiver and ADC
- Derive direct conversion specifications from circuit envelope model of RF receiver
- Perform multi-carrier simulation including interfering signals and derive the specifications of the DC offset compensation algorithm

Design and Verify Baseband Transmitter

To evaluate the performance of the RF receiver design, it is necessary and sufficient to use a signal spectrally representative of an 802.15.4 waveform.

The baseband transmitter model creates and illustrates a spectrally representative ZigBee waveform in the spectral and constellation domains. This model and all the subsequent models use callbacks to create MATLAB workspace variables that parameterize the systems.





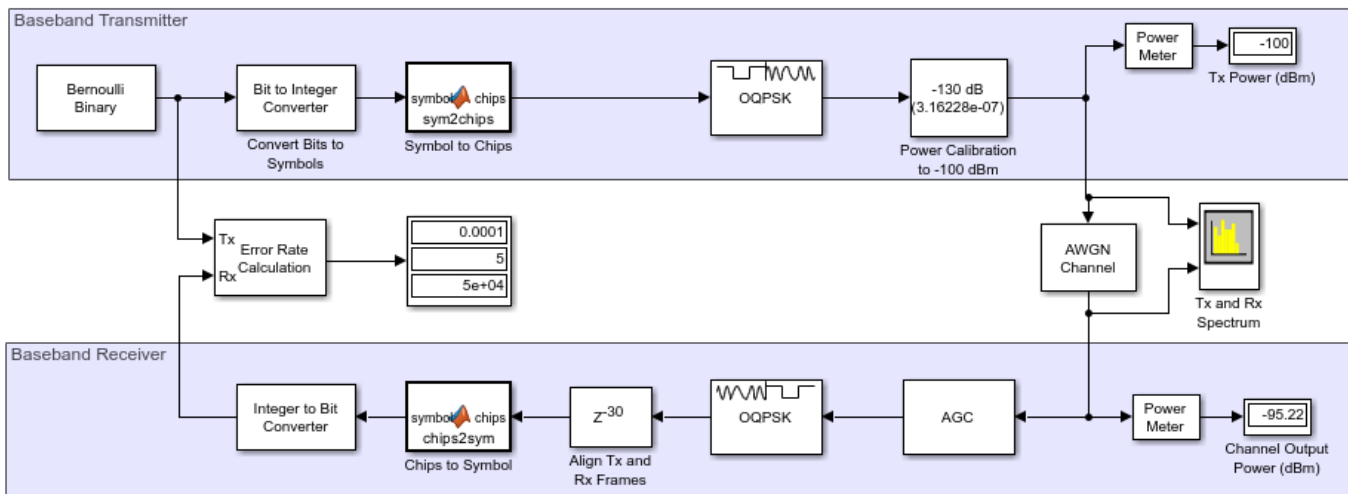
Determine Receiver SNR Requirement

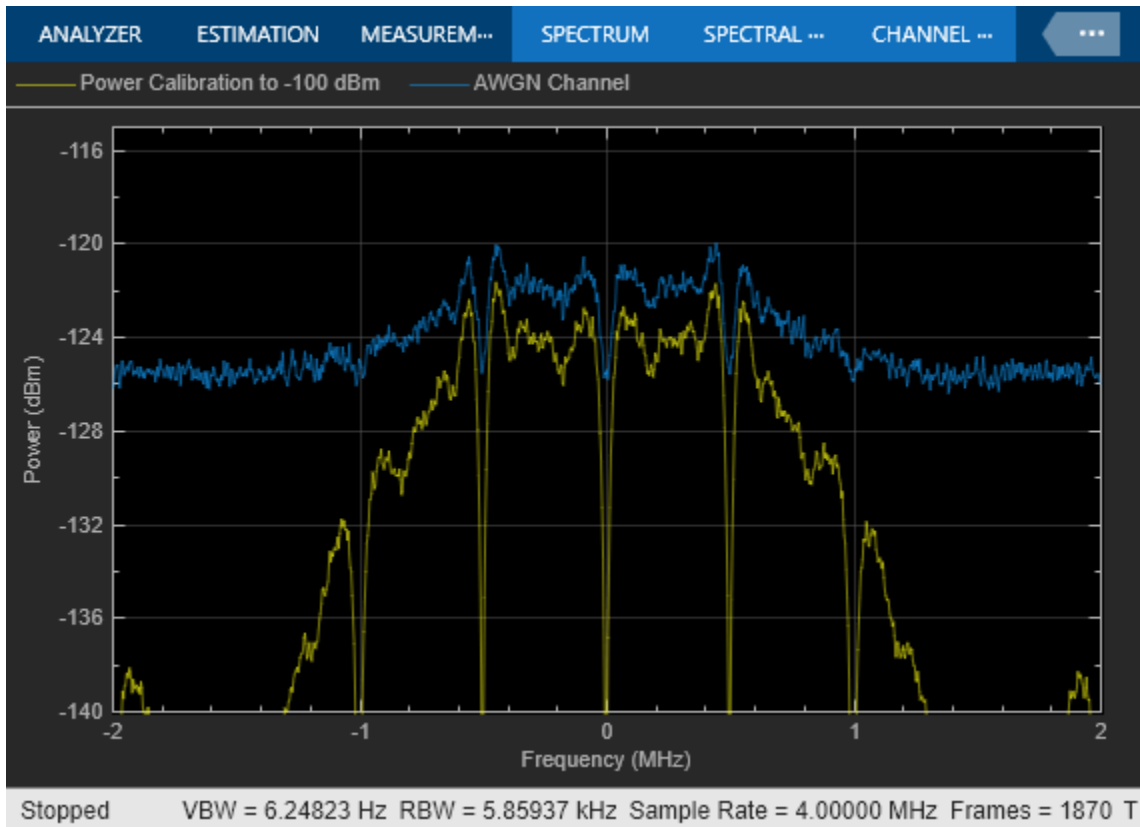
To design the receiver, first determine the SNR needed to achieve the specified BER less than $1e-4$, calculated in the simulation bandwidth of 4 MHz. Run the link-level model to simulate the receiver processing required to achieve the target BER.

Computing the BER accurately requires alignment of the transmit and receive signals. The simulation must compensate for a two-sample delay of the received signal compared to the transmitted signal. Also, to ensure correct chip-to-symbol-to-bit mapping, the simulation must align the chips to frame boundaries at the input to the Chips to Symbol block on a frame boundary. Accounting for the receive signal delay and the frame boundary alignment requires addition of a **Delay** block set to a $32-2=30$ delay on the receiver branch before recovering the received symbols.

ZigBee-Like System for SNR Determination

Copyright 2018-2020 The MathWorks, Inc.





The model achieves a $1e-4$ BER at an SNR of -2.7 dB, which can be verified by collecting 100 bit errors.

In the link-level model, the AWGN block accounts for the overall channel and RF receiver SNR budget.

Add ADC and Determine Receiver Total Gain and Noise Figure (NF)

This section uses traditional heuristic derivations to determine the high-level specifications of the RF receiver and ADC.

- $B = 4$ MHz = simulation bandwidth = simulation sampling frequency
- $kT = 174$ dBm/Hz = thermal noise floor power density
- Sensitivity = -100 dBm = receiver sensitivity
- SNR = -2.7 dB
- Noise power in simulation bandwidth = P_n = sensitivity-SNR = -100 dBm - (-2.7 dB) = -97.3 dBm
- $P_n = kT + 10 \cdot \log_{10}(B) + NF = -97.3$ dBm

Solving for the receiver noise figure (NF):

$$NF = -97.3 \text{ dBm} + 174 \text{ dBm/Hz} - 10 \cdot \log_{10}(4e6 \text{ Hz}) = 10.7 \text{ dB}$$

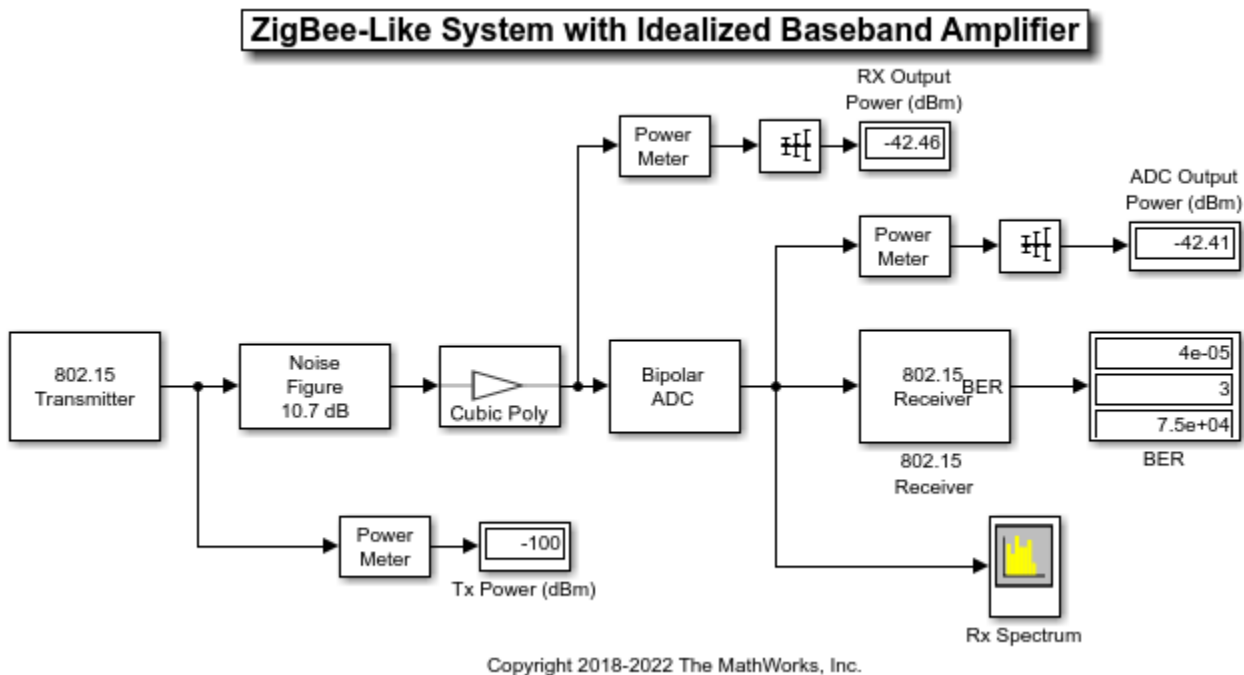
Derive the receiver gain using the ADC specifications and dynamic range.

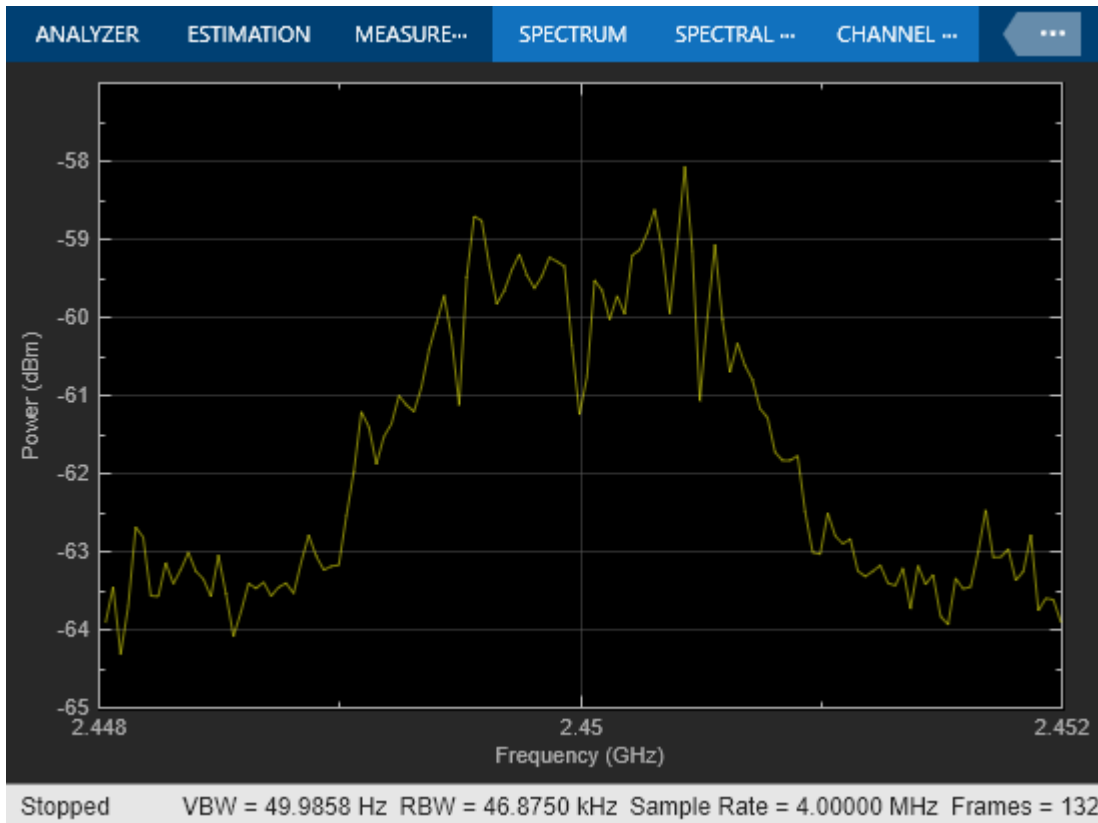
- ADC Number of bits = Nbits = 10

- ADC Saturation power = $P_{sat} = 0$ dBm (50 Ohm normalization)
- ADC Sampling frequency = $F_{adc} = 2.6$ MHz
- ADC Dynamic Range = $6 * N_{bits} + 1.8 = 61.8$ dB
- Noise power in ADC bandwidth = $P_{Nadc} = P_n + 10\log_{10}(F_{adc}/F_s) = -99.2$ dBm
- Assuming a 0.1 dB contribution to SNR, Quantization noise = $P_{Nadc} - 16$ dB
- Receiver Gain = $(P_{sat} - \text{Dynamic Range}) - P_{Nadc} + 16$ dB = $(0$ dBm - 61.8 dB) - (-99.2 dBm) + 16 dB = 53.4 dB

Simulating an idealized baseband model of the RF Receiver, verify the preliminary RF receiver specifications (NF = 10.7 dB and receiver gain = 53.4 dB). This can be done by collecting 100 errors.

The spectrum analyzer shows that the received spectrum with the ADC is roughly identical in shape to the spectrum of the previous section, without the ADC.





Refine Architectural Description of RF Receiver

In this section the RF receiver, and its noise figure and gain budget specifications, are modelled by using four discrete subcomponents with these characteristics:

- SAW Filter: Noise Figure = 2.3 dB, Gain = -3 dB
- LNA: Noise Figure = 6 dB, Gain = 22 dB
- Passive Mixer: Noise Figure = 10 dB, Gain = -5 dB
- VGA: Noise Figure = 14 dB, Gain = 40 dB

The SAW filter performance is derived from a Touchstone file that specifies S-parameters characteristics. You can verify the gain by visualizing the S21 parameter in the X-Y plane at the operating frequency of 2.45 GHz. You can verify the noise figure by visualizing the NF parameter in the X-Y plane at the operating frequency of 2.45 GHz. Typically, an LNA with low noise and high gain follows the SAW filter, which greatly reduces the impact of the noise figure of the components after the LNA. Also, the passive mixer is specified with a high IP2. Similar to the SAW filter, you can verify the mixer gain by visualizing the S21 parameter in the X-Y plane over a user-specified frequency range of [2e9 3e9].

An equivalent baseband model simulates the refined RF receiver.

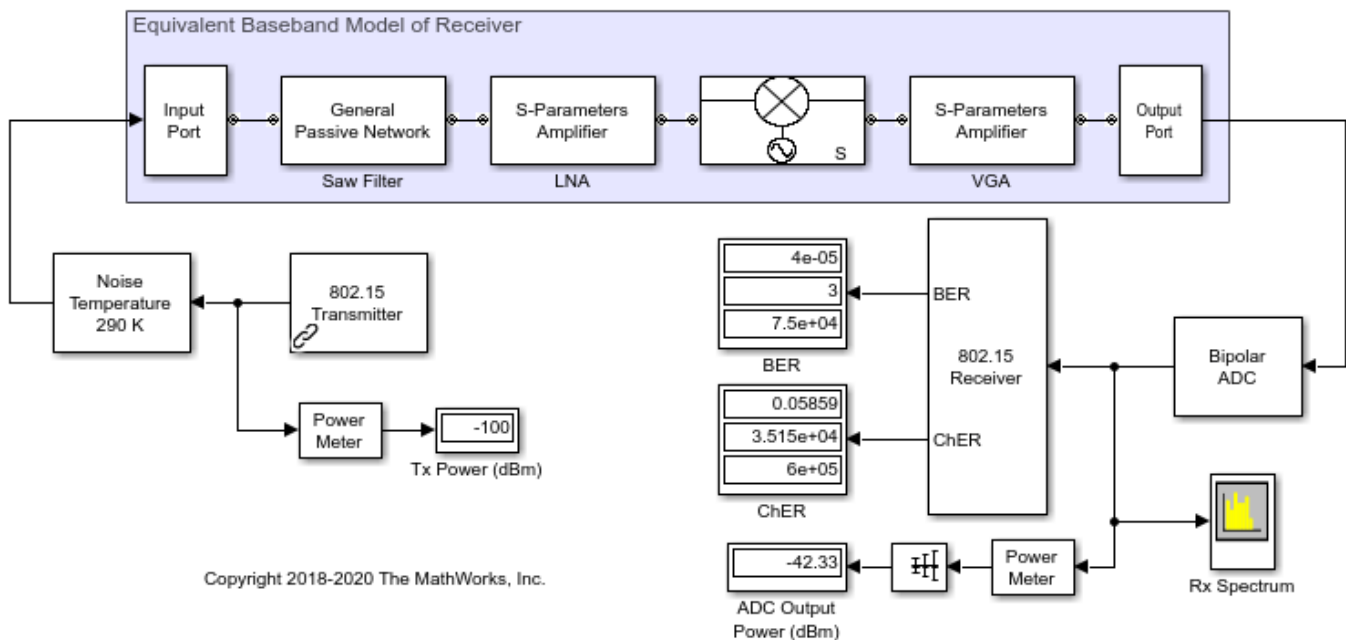
Run the simulation and verify the RF receiver link budget by using the output port visualization pane. The total noise figure and gain across the four stages has been divided according to the following budget:

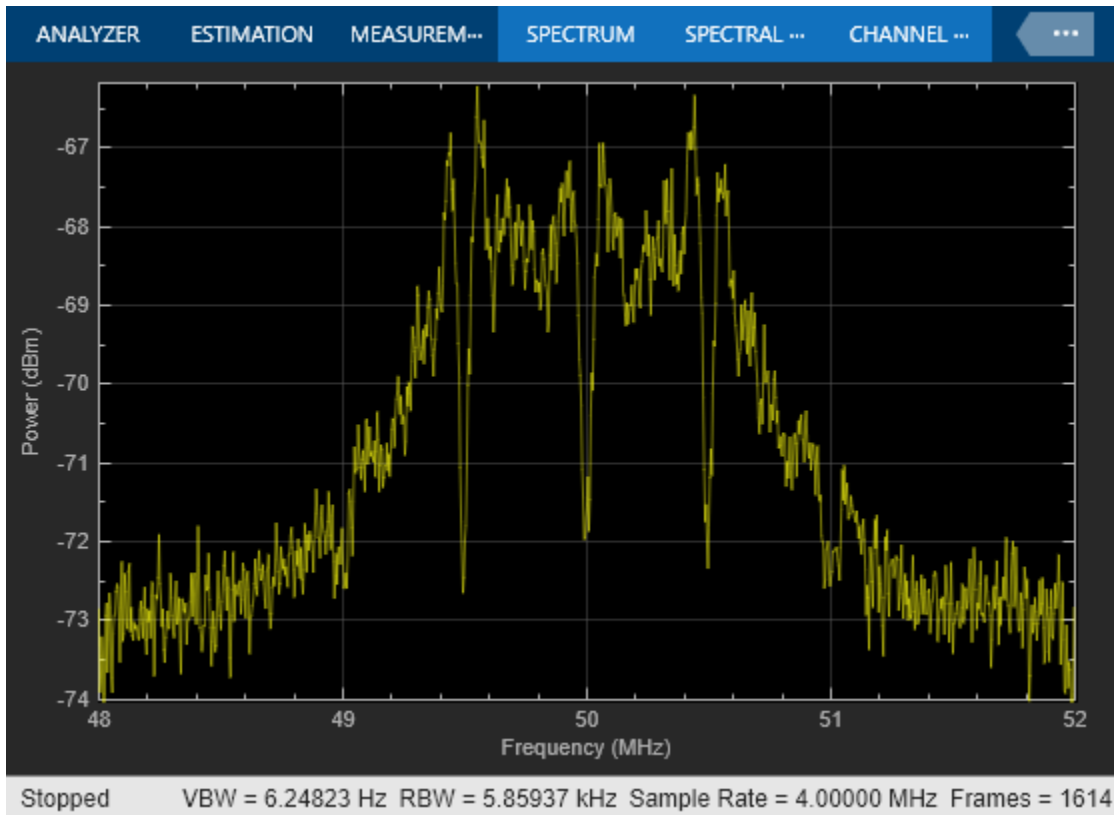
- Component NF (dB) = [2.3, 6, 10, 14]
- Component noise factor F (linear) = $10^{(NF/10)}$ = [1.78 3.98 10.0 25.1]
- Power gain (dB) = [-3, 22, -5, 40] = 54 dB > 53.4 dB
- Voltage gain VG (linear) = $10^{(Power\ gain/20)}$ = [0.71 12.59 0.56 100.0]
- System noise factor Fsys (linear) = $1 + [F(1) - 1] + \frac{[F(2) - 1]}{VG(1)} + \frac{[F(3) - 1]}{VG(1) \times VG(2)} + \frac{[F(4) - 1]}{VG(1) \times VG(2) \times VG(3)} = 11.8$
- System noise figure NFsys (dB) = $10 \cdot \log_{10}(F_{sys}) = 10.7$ dB

The actual noise figure of the chain, taking into account impedance mismatches, can be verified at the output port of the Equivalent Baseband Model of Receiver, and it is equal to 9.42dB.

With this model you can verify that a BER < 1e-4 corresponds to a Chip Error Rate (ChER) around 7%. By computing ChER, you can run the subsequent models for less time and still collect accurate BER statistics.

ZigBee-Like System with Equivalent Baseband Receiver





Use Circuit Envelope to Simulate Additional RF Impairments

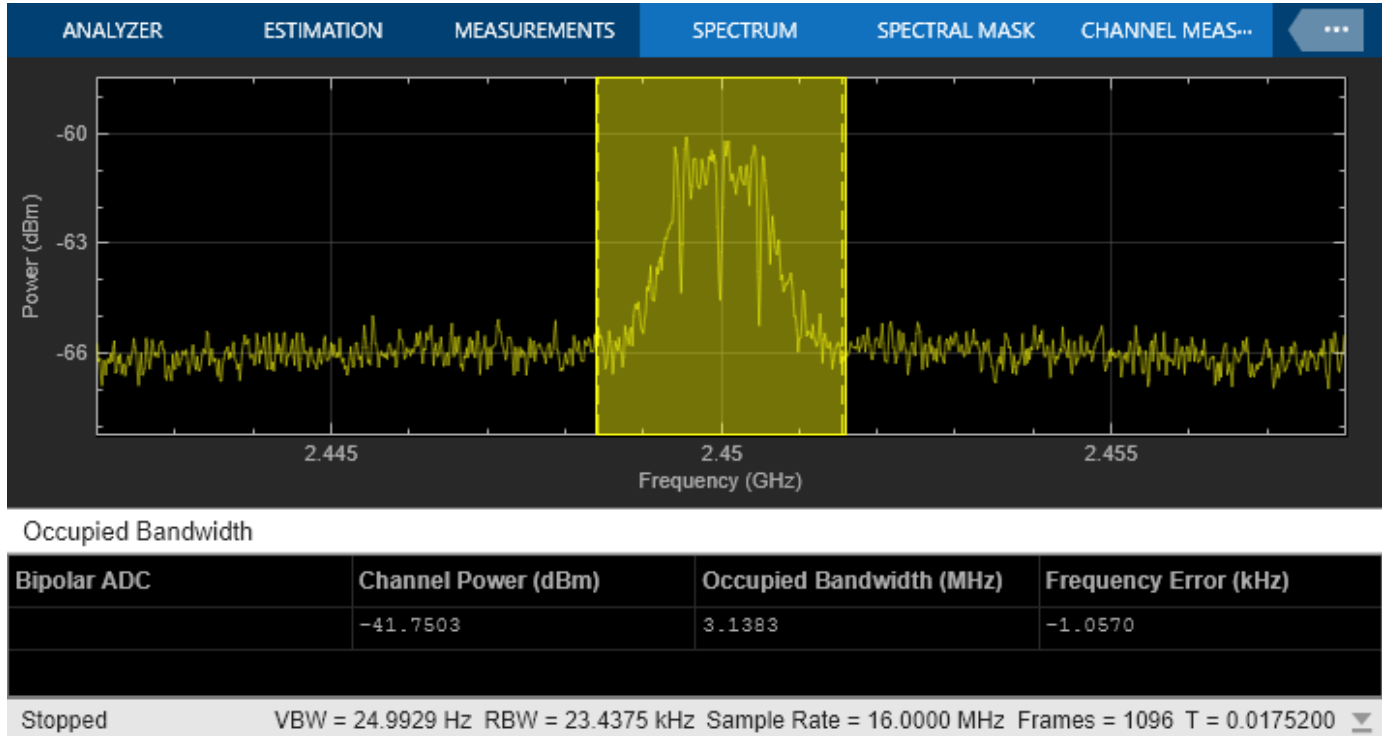
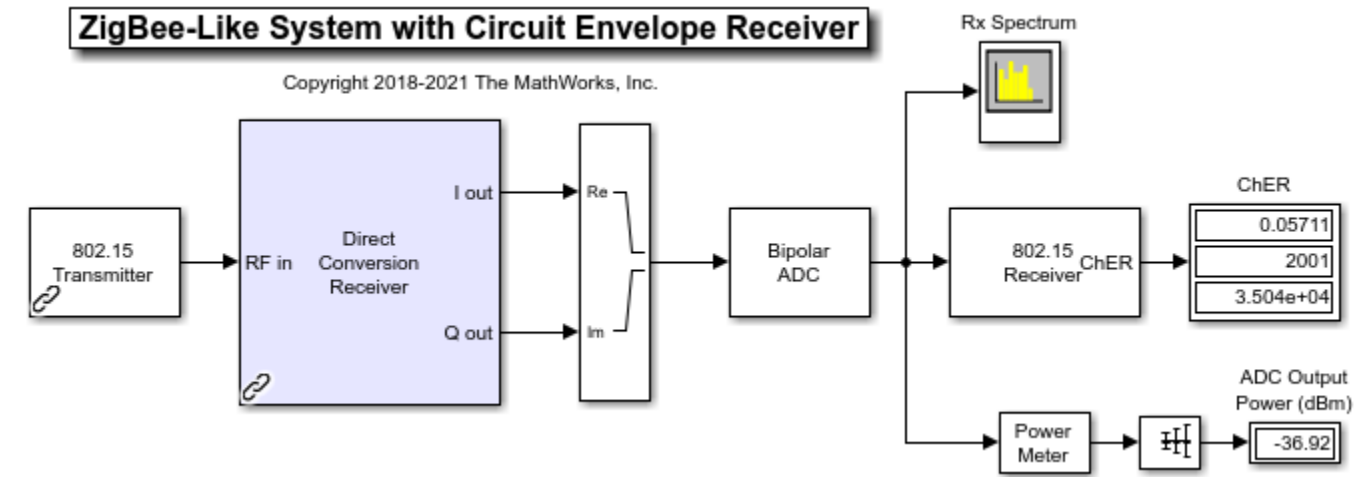
The equivalent baseband modeling technique used in the previous section cannot model a true direct conversion receiver. That model used a mixer with an input frequency of 2.45 GHz and an LO frequency of 2.4 GHz, which led to a spectrum analyzer center frequency of 50 MHz. This modeling limitation motivates a change to the circuit envelope method.

Using the circuit envelope modeling approach, continue refining the RF receiver architecture by adding more realistic impairments.

The circuit envelope model of the RF Receiver differs from the equivalent baseband model as it:

- Replaces the equivalent baseband mixer with a quadrature modulator, consisting of parameterizable I and Q mixers and phase shifter block, and an LO with impairments
- Uses broadband impedances (50 ohm) to explicitly model the power transfer between blocks

Comparing spectra, power measurements, and ChER to the equivalent baseband model, there are no significant performance differences. However, with the circuit envelope model, you can include even order nonlinearity effects, I/Q imbalance, and specifications of colored noise distributions for each of the components.



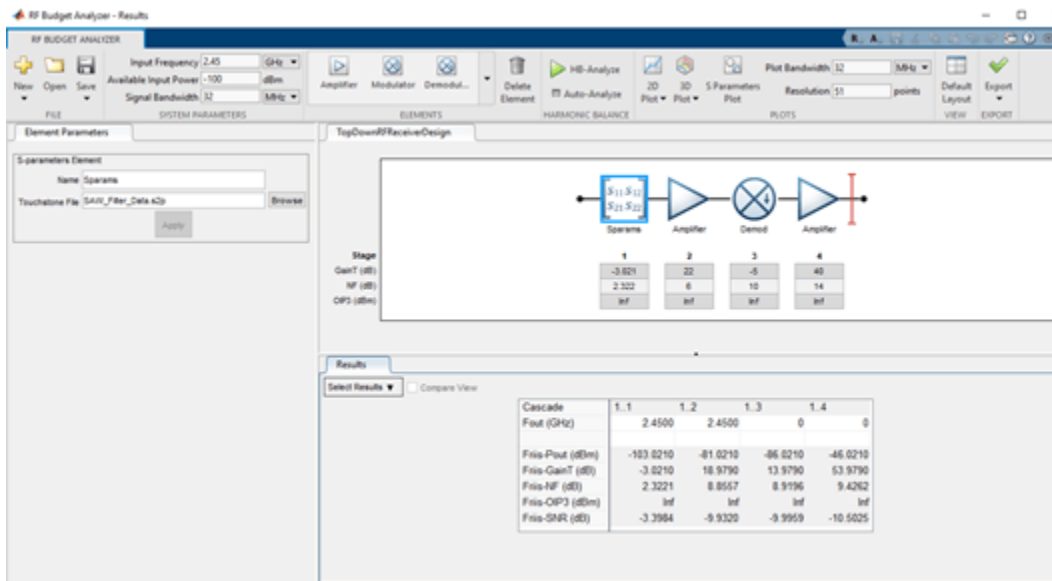
You can manually build the circuit envelope model of the RF Receiver by using blocks from the Circuit Envelope library, or it can be automatically generated using the **RF Budget Analyzer** app.

The **RF Budget Analyzer** app

- Uses Friis equations to determine the noise, gain, and nonlinearity budget of an RF chain, also taking into account impedance mismatches
- Allows you to explore the receiver design space and determine how to break down the specifications across the elements of the chain
- Helps you determine which element has the largest contribution to the noise and nonlinearity budget

- Can generate an RF receiver model with which you can perform multi-carrier simulation and further modify.

Type `rfBudgetAnalyzer('TopDownRFReceiverDesign.mat')` command at the command line to visualize the RF receiver in the **RF Budget Analyzer** app.



Add Wideband Interference, LO Leakage, and DC Offset Cancellation

This section modifies the circuit envelope model to create this circuit envelope with interferer model. The circuit envelope with interferer model includes a wideband interfering signal and these impairments:

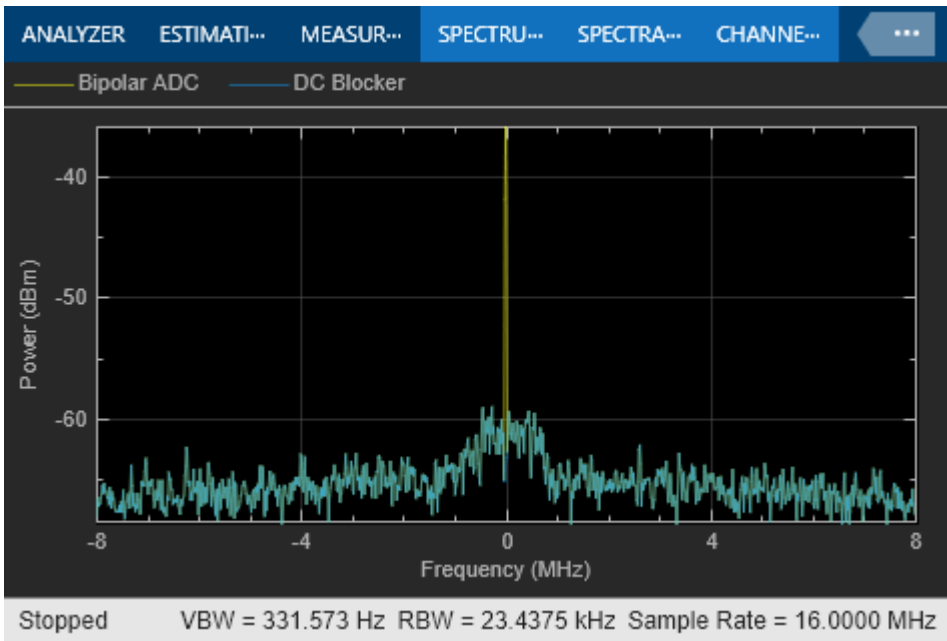
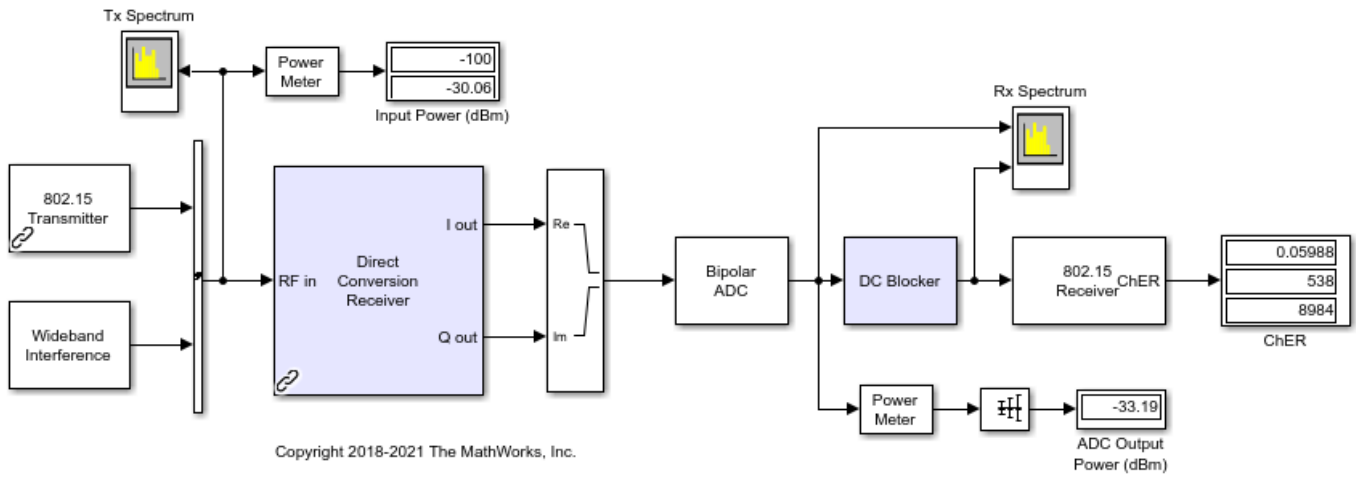
- LO-RF isolation of 90 dB in the quadrature demodulator
- OIP2 equal to 55 dBm in the quadrature demodulator
- WCDMA-like out-of-band blocker of -30 dBm at 2500 MHz

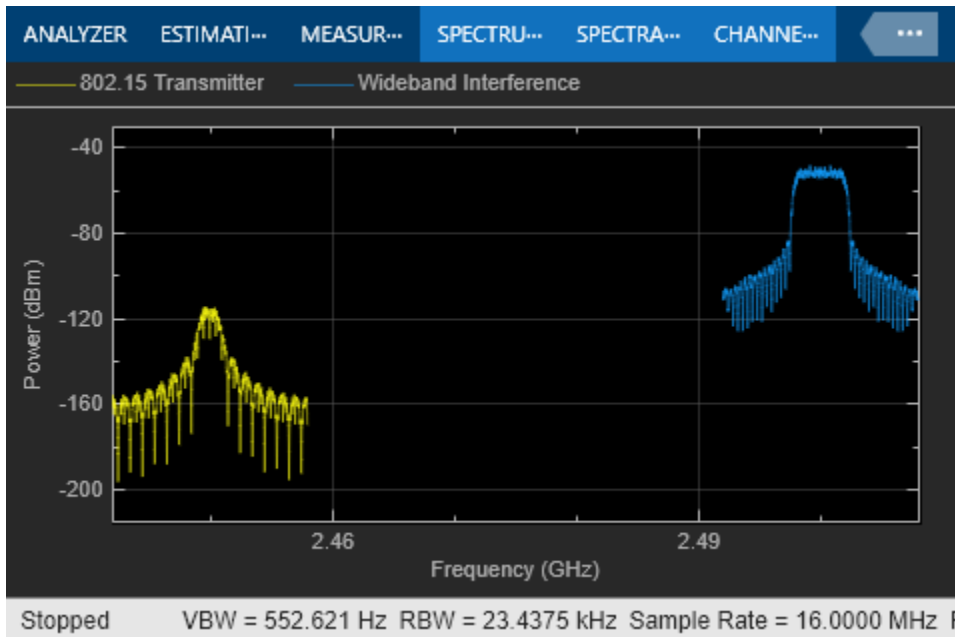
This simulation models a non-standard-compliant interfering signal that has power and spectral distribution characteristics realistic for a WCDMA signal. The simulation of the wideband interfering signal requires a larger simulation bandwidth of 16MHz. Therefore the 1 MHz OQPSK signal is oversampled by 16, and the Circuit Envelope simulation bandwidth is also increased to 16 MHz.

The design requires a DC offset compensation algorithm to achieve the desired ChER due to the DC offset that results from the LO leakage and the nonlinearity in the demodulator caused by the high out-of-band interfering signal power. In this case you include a very selective filter, that introduces a long latency with corresponding computation delay increases in the ChER measurement block.

The spectrum centered at 0 Hz shows the DC offset compensation reducing the DC offset. As you run the model, note that the DC offset is eventually completely removed.

ZigBee-Like System and Interferer with Circuit Envelope Receiver





Conclusion

Following a top-down design methodology, RF receiver components specifications were derived. Impairment, interferer, and RF receiver subcomponent models were iteratively refined to increase fidelity and validated at each stage to confirm overall system performance goals were achieved.

See Also

VGA | Mixer | S-Parameters Amplifier | General Passive Network

More About

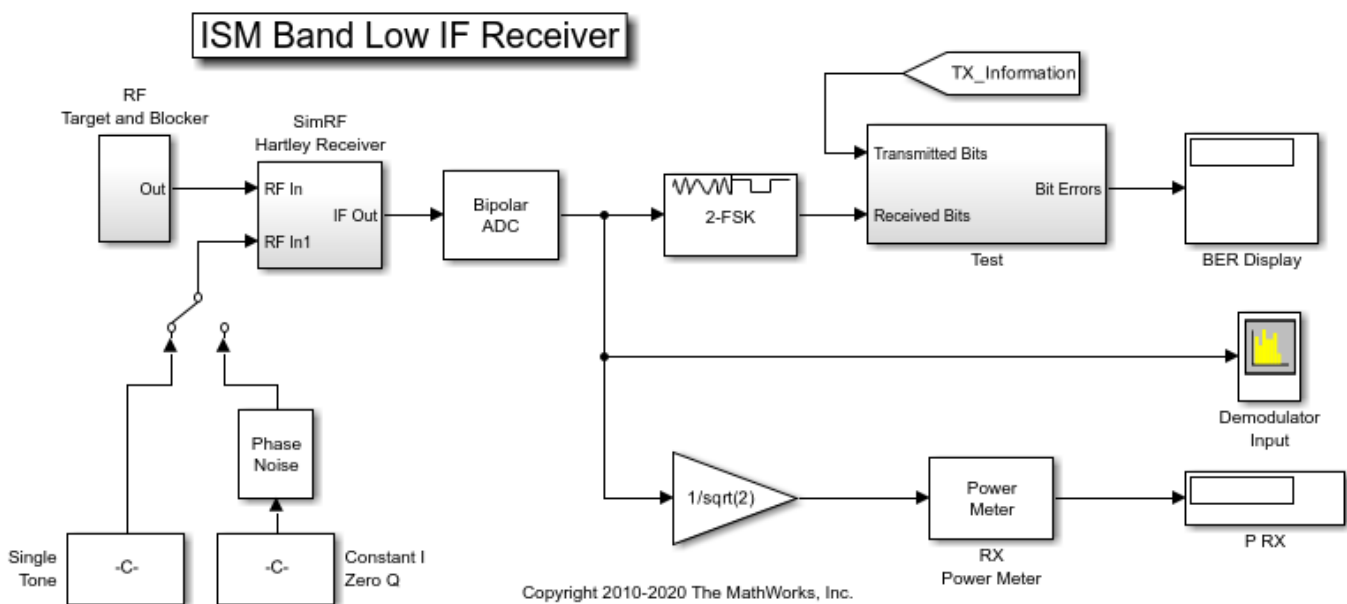
- “Architectural Design of a Low IF Receiver System” on page 8-178
- “Carrier to Interference Performance of Weaver Receiver” on page 7-48
- “Frequency Response of RF Transmit/Receive Duplex Filter” on page 8-75

Architectural Design of a Low IF Receiver System

This example shows how to use the RF Blockset™ Circuit Envelope library to simulate the performance of a Low IF architecture with the following RF impairments:

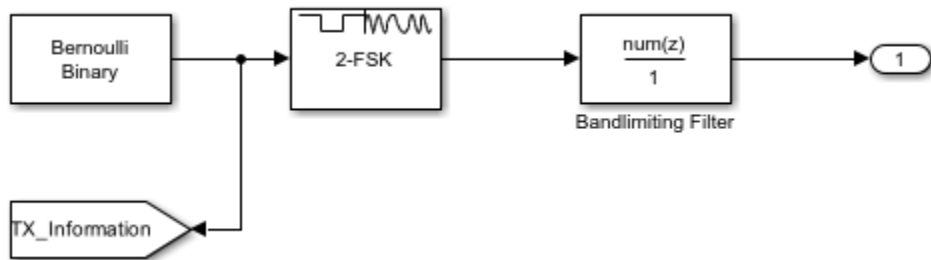
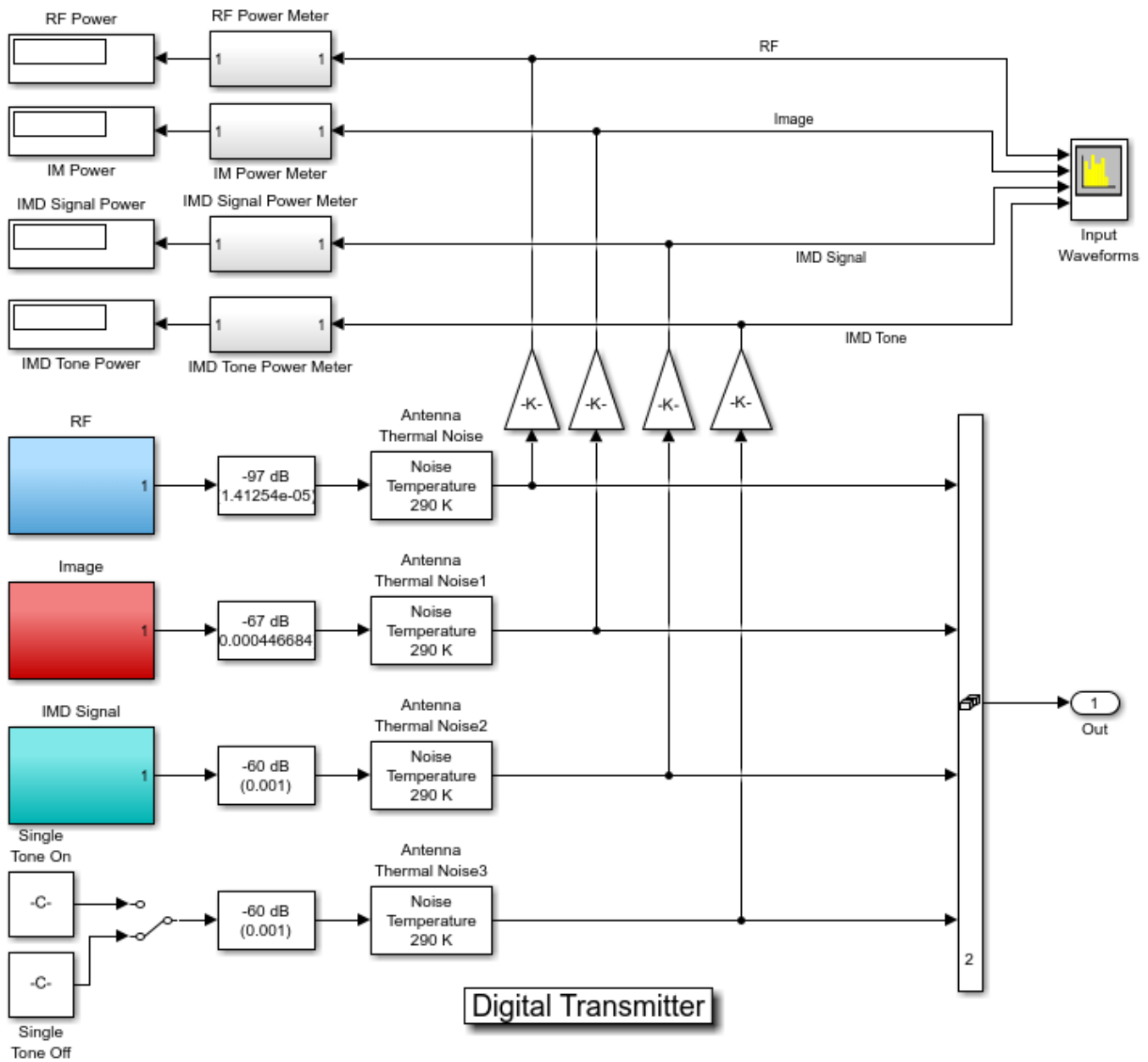
- Component noise
- Interference from blocker signals
- LO phase noise
- Analog-to-digital converter (ADC) dynamic range
- Component mismatch

Design variables in the RF portion of the model include explicit specification of gain, noise figure, IP3, input/output impedance, LO phase offset, and LO phase noise. Carrier frequencies for waveforms entering RF Blockset subsystems are specified in the Inport blocks. Design variables for the transmitter side of the RF interface include carrier frequency, modulation scheme, signal power, and blocker power level. Baseband design variables are number of bits and full scale range of the ADC.

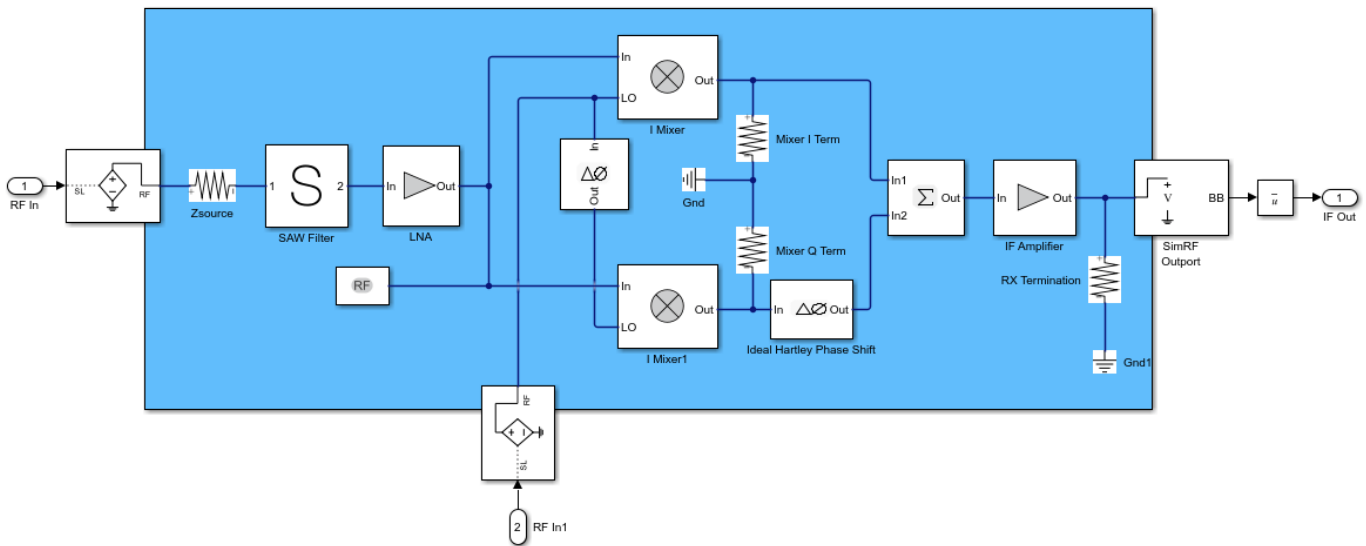


System Architecture

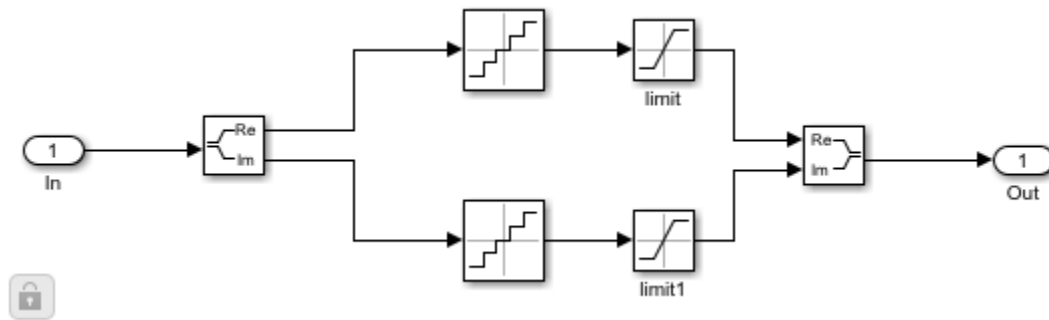
This model illustrates the design and simulation of an ISM Band Receiver. Primary subsystems include a digital transmitter, an RF receiver, an ADC, a phase noise block for noisy LO modeling, and a digital receiver. The remaining blocks are used for analysis.



The digital transmitter consists of three FSK modulated waveforms and a high power tone. The three FSK waveform generators use a bandlimiting filter that suppresses the FSK sidebands below the expected thermal noise level. The target waveform at 2450 MHz has a 1 ohm referenced passband power level of approximately -70 dBm. Similarly defined image and intermodulation distortion (IMD) blocker waveforms have passband powers of approximately -40 dBm and -33 dBm, respectively. The IMD tone that couples with the IMD blocker to generate in-band IM3 products has a passband power of -33 dBm. Since the baseband processing defines the complex envelope waveforms, computing passband power requires the insertion of $1/\sqrt{2}$ gain as shown in the design. An IF of 2 MHz can be inferred by inspecting the demodulator input signal spectrum, where a 2 MHz offset is specified for the display.



The Low IF receiver is comprised of a receive band SAW filter, a frequency conversion stage, an image rejection stage, and two gain stages. Resistors are used to model input and output impedances. Each nonlinear block has a noise figure specification. Power nonlinearities in the low noise amplifier (LNA), IF amplifier and mixers are specified by IP3. Image rejection is accomplished with a Hartley design, and single LO and phase shift blocks provide cosine and sine terms to mix with the I and Q branches, respectively. The summation block recombines the signals on the I branch and the phase-shifted Q branch. Image rejection quality can be controlled directly by setting a non-ideal phase offset in the Phase Shift block. To capture the RF, Image, IMD Signal and IMD Tone waveforms/spectra, choose the **Fundamental tones** to be 2450 MHz, 1 MHz and the **Harmonic Order** as 1 for the first tone and 8 for the second tone within the Configuration block. To model a thermal noise floor in the RF Blockset environment, the **Temperature** within the System Parameters section in the Configuration block is set to a noise temperature of 290.0 K.



The ADC is modeled using an a 12-bit quantizer. The quantizer takes into account the full-scale and dynamic ranges of the ADC, properly modeling its quantization noise floor.

A digital receiver demodulates the waveform for bit error rate calculation. This noncoherent FSK receiver assumes perfect timing synchronization, such that each FSK pulse is integrated over one and only one symbol.

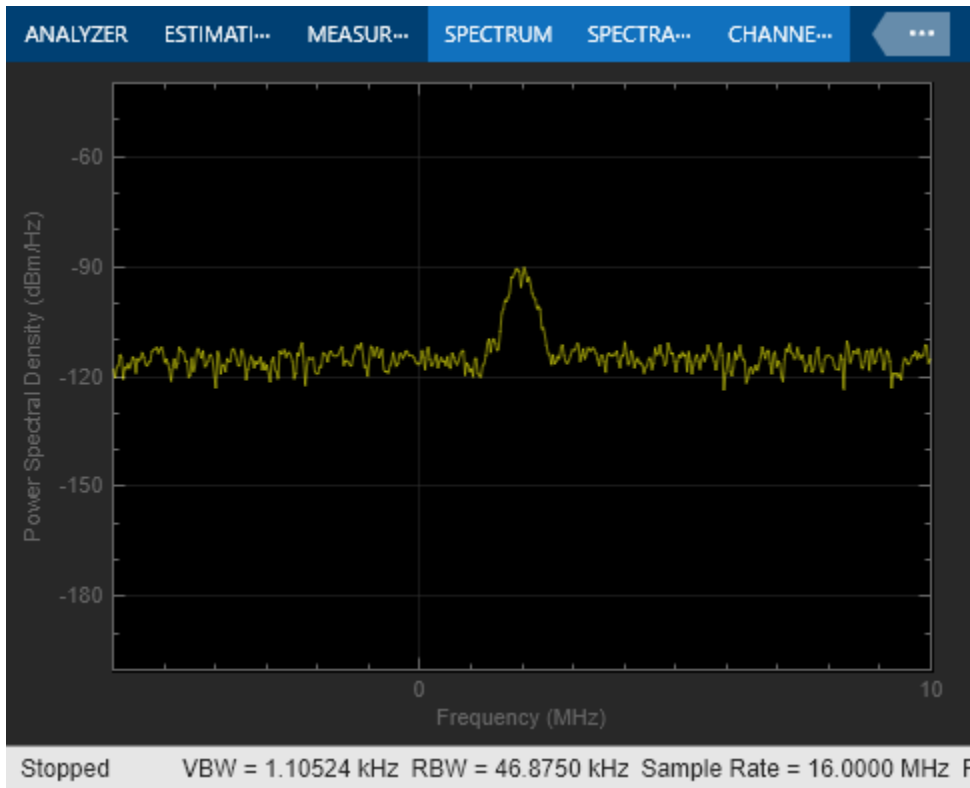
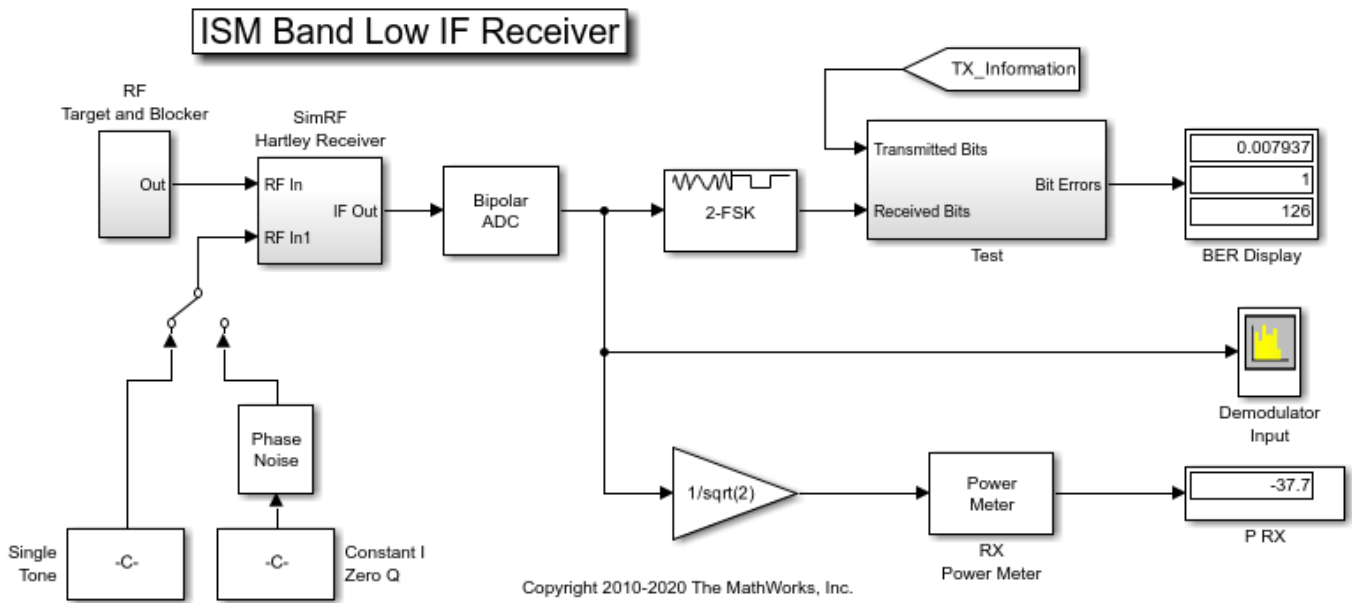
Running the Example

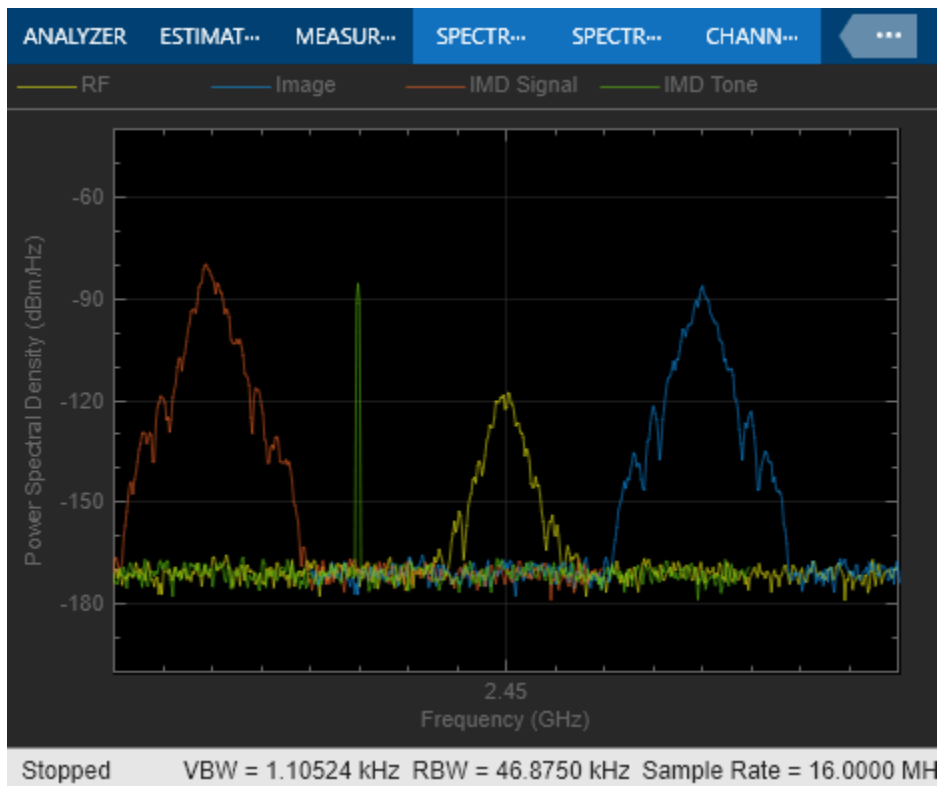
Running the example simulates a design that meets an uncoded BER spec of less than 1%. Modifications to the signals and component specifications in the receiver and ADC have a direct impact on the receiver performance. Manual switches enable you to:

- 1 Select a power level for the IMD blocker tone of -33 dBm or -45 dBm
- 2 Select an ideal or noisy LO.

Other possible changes to the design include:

- Image rejection ratio (IRR) of the Hartley design. The IRR of the present design ($d\Phi=0.01$ degrees) is -40 dB. For more information on calculating IRR, see the example “Measuring Image Rejection Ratio in Receivers” on page 8-66 Measuring Image Rejection Ratio in Receivers>.
- Modulation schemes
- Baseband filtering options
- Signal power levels
- Signal carrier frequencies
- Noise figures
- Non-linear gain parameters
- Interstage matching
- ADC bit length and full scale range





See Also

Amplifier | Mixer | Output

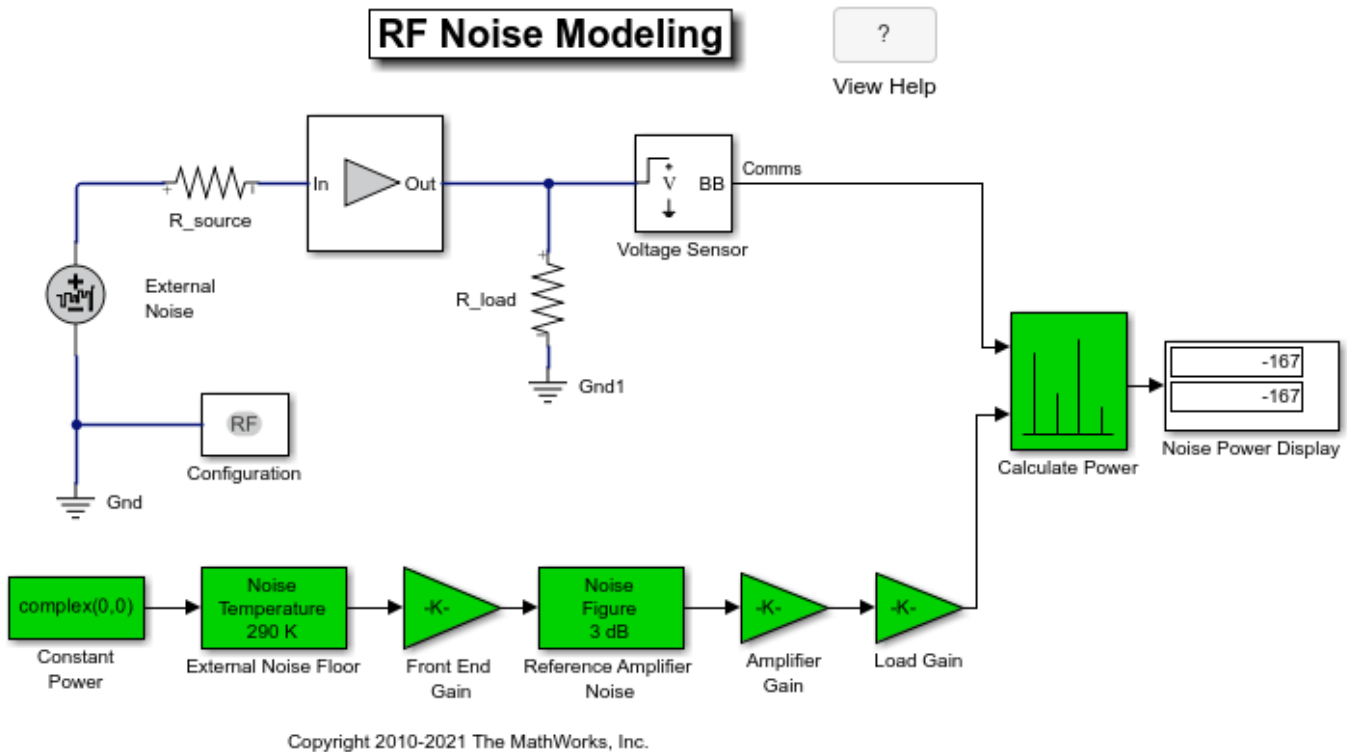
More About

- “Top-Down Design of an RF Receiver” on page 8-166
- “Carrier to Interference Performance of Weaver Receiver” on page 7-48
- “Frequency Response of RF Transmit/Receive Duplex Filter” on page 8-75

RF Noise Modeling

This example shows how to use the RF Blockset™ Circuit Envelope library to simulate noise and calculate noise power. Results are compared against theoretical calculations and a Communications Toolbox™ reference model.

System Architecture



The model defines variables for block parameters using the `PreLoadFcn` callback function. To access model callbacks, select **MODELING > Model Settings > Model Properties** and select the Callbacks tab in the Model Properties window.

The RF system, shown in white, consists of these blocks.

- Configuration block - This block sets global simulation parameters for the RF Blockset system. Selecting **Simulate Noise** adds noise to the simulation.
- External Noise block - This block adds a power spectral density of $4kT_sR$ at the input. In this equation, k is the Boltzmann constant, T_s is the temperature of the source, and R is the noise reference impedance. The calculated noise level of -174 dBm/Hz is used in this example. The External Noise block provides an explicit signal source.
- Amplifier block - This block specifies the power gain and noise figure.
- Voltage Sensor block - This is an Outport block with the **Source type** parameter set to **Ideal voltage**.

- Resistor blocks - These blocks specify the source and load resistance.

The Communications Toolbox reference system, shown in green, consists of these blocks.

- Constant Power block - This block provides a constant input signal source.
- Gain blocks - These blocks model front end gain, amplifier gain and loading effects.
- Receiver Thermal Noise blocks - These blocks model the external noise floor and the reference amplifier noise, respectively.

The Calculate Power block computes RMS noise power for the actual load resistance, R_{load} .

Run Example

- 1 Use the **Open Model** button to open and run the model.

The Noise Power Display block verifies that the RF Blockset and Communications Toolbox noise models are equivalent.

Compute RF System Noise

To enable noise in the RF Blockset circuit envelope environment:

- In the Configuration block dialog, select **Simulate noise**.
- Specify a **Temperature**. RF Blockset uses this value to calculate the equivalent noise temperature inside the amplifier.
- Specify the **Noise figure (dB)** parameter of any amplifiers or mixers in the system.

In the example, for a specified LNA gain of 4 dB and noise figure of 3 dB, the output noise is calculated using the following equations:

$$G_1 = 2.5119 \text{ (4 dB)}$$

$$F_1 = 1.9953 \text{ (3 dB)}$$

The next equation converts the noise factor to an equivalent noise temperature. T is the **Temperature** parameter of the RF Blockset Configuration block.

$$T_e = (F_1 - 1) * T = 288.63$$

The final equation calculates the output noise power. T_s is the temperature of the SimRF™ External Noise block and the Communications Toolbox External Noise Floor block.

$$N_{out,sys} = 10 \log_{10} (k(T_s + T_e)G_1) + 30 = -166.97 \text{ dBm/Hz}$$

The available noise power is the power that can be supplied by a resistive source when it is feeding a noiseless resistive load equal to the source resistance. The green External Noise Floor block generates an available power referenced to 50 ohms.

The Front End Gain block models the voltage divider due to the source resistance and the input impedance of the amplifier.

The green Reference Amplifier Noise and Amplifier Gain blocks model the noise added by the amplifier and the amplifier gain, respectively.

The output of the green Amplifier Gain block is equal to the voltage across the RF Blockset R_load block.

See Also

Noise | Noise Figure Testbench

More About

- “Noise in RF Systems” on page 2-7
- “Model LO Phase Noise” on page 7-42

Impact of Thermal Noise on Communication System Performance

This example shows how to use the RF Blockset™ Circuit Envelope library to model thermal noise in a super-heterodyne RF receiver and measure its effects on a communications system noise figure (NF) and bit error rate (BER). A Communications Toolbox™ reference model with parameters computed using Friis equations and a RF Blockset Noise Testbench are used to verify the results.

RF Receiver System Architecture

The Modulator and Channel subsystems consist of Communications Toolbox blocks that model:

- A QPSK-modulated waveform of random bits
- A raised cosine pulse-shaping filter for spectral limiting
- free-space path loss

The RF receiver subsystem, shown in light purple, consists of RF Blockset blocks:

- An Inport block converts the complex input waveform to available power in the RF system with reference impedance equal to the **Source impedance** and assigns the input modulation waveform to a 2.1 GHz RF carrier.
- A noise source to set the RF system noise floor for all simulation carrier frequencies. The block performs this action when **White** is selected for the mask Noise distribution option. To set the Noise power spectral density level, a value of $4 \cdot K \cdot T \cdot 50$ is used (K is Boltzmann's constant, T is set to a room temperature of 290 kelvin, and 50 ohms is the system reference impedance).
- Cascaded RF amplifier and RF demodulator blocks with specified noise figure and gain. These blocks only enable noise impairments. The Demodulator block's image reject filter is enabled using a mask checkbox and defines with other mask parameters a bandpass filter whose edges are 2.0 and 2.2 GHz. This filter prevents the down-conversion of thermal noise centered around 2.6 GHz or folding of other carrier frequencies with noise into the intermediate frequency (IF) defined as the absolute difference of the RF and LO frequencies. If the image rejection filter is removed, the noise contribution on the IF increases above the estimation provided by Friis equations and the BER will deteriorate.
- An Output block, with the parameter **Sensor type** is set to Power, **Carrier frequencies** set to the IF frequency, and **Output** parameter is set to Complex baseband. These block settings enable the RF system to supply a complex baseband communication signal to the ensuing Communication Toolbox system blocks.
- A Configuration block to set model conditions for simulation. Since the model's RF Blockset section has only included noise impairments, an accurate simulation can be achieved by setting the Configuration block **Fundamental tones** to the Inport Carrier (RF), 5e8 Hz and Demodulator Local oscillator (LO), 1.6e9 Hz frequencies and the **Harmonic order** 1. Use the Configuration blocks **View** button to explore simulation carrier frequencies.
- All blocks in the RF receiver are matched to 50 ohms. To understand the effects of impedance mismatch on noise simulation see, "RF Noise Modeling" on page 8-184.

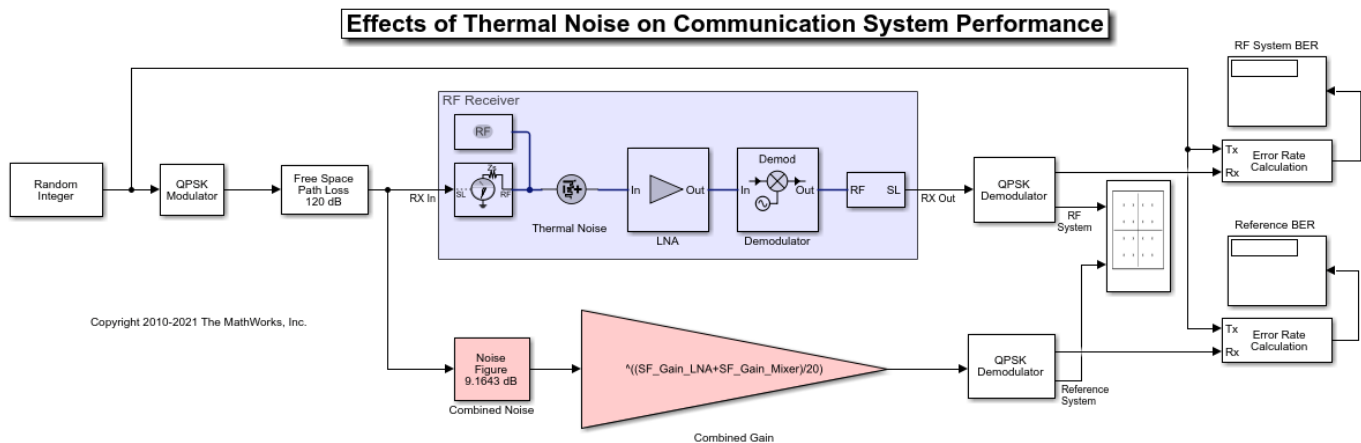
The reference system, shown in red, consists of:

- A Communications Toolbox Receiver Thermal Noise block that includes both the thermal noise floor along with the amplifier and demodulator block noise. The Friis Equation is used to correctly

combined noise contributed by the amplifier and demodulator blocks. You can find the calculation in the model's pre-load callback function.

- A Simulink Gain block that models the combined gain of the RF receiver.
- Baseband filters and demodulators process the received signal.

Circuit Envelope Simulation of RF Receiver



Select **Simulation > Run** .

Error Rate Calculation blocks compute the BER for the system and reference. To observe the BER as it approaches steady state, increase the total simulation time. For this example, the steady-state bit error rate is approximately $1e-4$.

Computing RF Receiver Noise Figure and Gain

To model noise and gain in the RF Blockset circuit envelope environment:

- In the Configuration block dialog, select **Simulate noise**.
- Specify the **Noise figure (dB)** parameter of RF Amplifier and RF Mixer blocks in your system. The following specifications for the RF receiver in this example produce a combined noise figure of 9.16 dB (as per the Friis Equation): LNA gain of 20 dB, LNA noise figure of 9 dB, Demodulator gain of -5 dB and RF Demodulator noise figure of 15 dB.

$$G_1 = 100 \text{ (20 dB)}$$

$$G_2 = .316 \text{ (-5 dB)}$$

$$F_1 = 7.94 \text{ (9 dB)}$$

$$F_2 = 31.62 \text{ (15 dB)}$$

$$F_{sys} = F_1 + \frac{F_2 - 1}{G_1} = 8.25$$

$$NF_{sys} = 10 \log_{10} F_{sys} = 9.16 \text{ dB}$$

$$G_{sys} = G_1 \text{ dB} + G_2 \text{ dB} = 15 \text{ dB}$$

RF Blockset Noise Figure Testbench

The RF Blockset Noise Figure Testbench simplifies the measurement of system noise figure. To setup a noise figure test system, insert an RF Noise Figure Testbench in a new model. Copy the settings found in the Model Properties Callbacks PreLoadFcn to the new models Model Properties Callback InitFcn.

For the system composed of RF Blockset blocks in the above model, copy the LNA and Demodulator blocks with previously set parameters to the new model. The Testbench includes a Noise source that sets the noise floor.

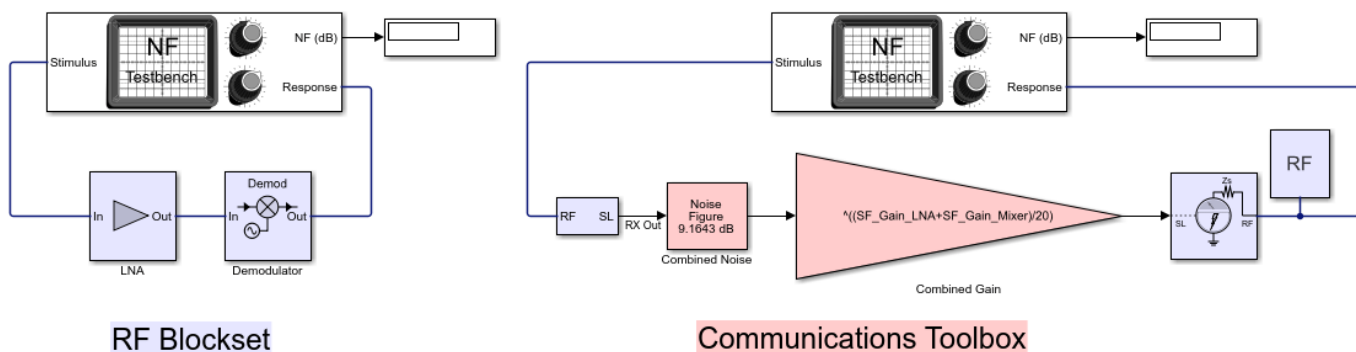
- Connect the Stimulus terminal of the testbench to the In terminal of the LNA and the Out terminal of the Demodulator to testbench Response terminal. A Display block can be connected to the testbench NF terminal to display the measured Noise figure.
- Set the Testbench mask parameters. The RF **Input frequency (Hz)** is 2.1 GHz and the IF **Output frequency (Hz)** is .5 GHz as in the previous example. A 10e6 Hz **Baseband bandwidth (Hz)** was chosen for this example. The mask instructions provide additional information for configuring the testbench.

For the Communications Friis system in the above model, copy the Combined Noise and Gain blocks with previously set parameters to the new model. The Combined Noise block's **Add 290K antenna noise** checkbox needs to be deselected since the Testbench includes a Noise source that sets the noise floor.

- Three RF Blockset blocks are included: an Outputport, an Inputport and a Configuration since the testbench expects RF Blockset blocks at its connection points. The type setting for the Inputport and Outputport blocks is Power. Since the Communication branch is agnostic to carrier frequencies, these blocks Carrier frequencies and Fundamental tones need to be the same and are set to 2.1 GHz. The **Output** parameter of the Outputport is Complex Baseband. For accuracy, the configuration block **Step size** needs an Envelope bandwidth (**Step size** of 1/80e6 s) at least 8 times larger than the 10 MHz Baseband bandwidth of the testbench.

Run Noise Figure Testbench

Noise Figure Testbenches



Select **Simulation > Run** .

Exploring Example

You can include additional RF model impairments using RF block mask selections: Impedance mismatch, nonlinearities or LO isolation.

See Also

Noise | Amplifier | Demodulator

Related Examples

- “RF Noise Modeling” on page 8-184
- “Explicitly Simulate Resistor Thermal Noise” on page 7-5

100 Watt TR Module for S-Band Applications

This example shows how to use the RF Blockset™ Circuit Envelope library to simulate a 100 Watt transmit and receive (TR) module. Three TR switches provide isolation between the transmitter and receiver and share a common path and components between both chains. You simulate the output power of transmit and receive chains and derive the achieved isolation.

System Architecture

The system consists of a transmit and receive module, and contains three TR single-pole double-throw switches arranged in a network that maximizes isolation and allows having a common path with programmable attenuation for both transmitter and receiver. The tunable TR switch control allows you to toggle the three TR switches simultaneously while running the simulation. When you toggle the controller, you enable either the transmit or the receive path. A slider allows you to change the tunable attenuation on the common path between 0 dB and 10 dB. The attenuation can be changed while running the simulation.

A continuous wave (CW) source centered at 2.1 GHz with nominal input power of 0 dBm is applied to both transmitter and receiver. Three circulators are used to separate the input and the output power of both transmitter and receiver paths. Thermal noise with white distribution is also added to the input signal to define the noise floor.

All amplifiers in the transmit, receive, and common path also generate noise specified by their noise figure.

TR Switches

You toggle the single-pole double-throw switches simultaneously. The switches are of absorptive type to reduce reflections at their ports. They are characterized by a very small insertion loss (0.01 dB) and very high isolation (100 dB). This small insertion loss and high isolation of the switches directly reflects on the high isolation between the transmit and receive path, and has the benefit of reducing the impact of impedance mismatches and reflections along the chain.

When the TR Switch control is ON, the TX path is enabled and the first output (Out1) of the switches is connected to the input. When the TR Switch control is OFF, the RX path is enabled and the second output (Out2) of the switches is connected to the input.

Transmitter

When transmit mode is enabled, the signal is first processed by the common path, and then amplified by the Driver Amplifier and Power Amplifier blocks. The total gain of the transmit chain, for nominal 0 dB attenuation, is approximately 52 dB. The combined gain of the Driver Amplifier and Power Amplifier blocks (transmit gain) is 40 dB.

When the input power of the transmitter is 0 dBm, the output power is approximately 50 dBm (100 Watts) and the received power is below -50 dBm. This result is due to input reflections of the transmitted signal on the TR switches.

When the input power of the transmitter is 0 W, the output power of the receiver is -73 dBm, equal to the receiver input power (0 dBm), plus the receive gain (27 dB), minus the switch isolation (100 dB).

Receiver

When receive mode is enabled, the signal is first amplified by the LNA and Stage Amplifier blocks, then it is processed by the common path. The total gain of the receive chain, for nominal 0 dB attenuation, is approximately 39 dB. The combined gain the LNA and Stage Amplifier is 27 dB (receive gain).

When the input power of the receiver is 0 dBm, the output power is approximately 40 dBm, and the transmitted power is below -20 dBm, equal to the receiver input power (0 dBm), plus the common path (12 dB) and receive gain (40 dB), minus the switch isolation (100 dB), plus the transmit gain (40 dB).

When the input power of the receiver is 0 W, the output power of the transmitter is -60 dBm, equal to the transmit input power (0 dBm), plus the transmit gain (40 dB), minus the switch isolation (100 dB).

Common Path Components

The common path is active during transmission and reception and consists of a Common Path Gain, Phase Shifter, and Attenuator blocks. The total gain of the common chain, for nominal 0 dB attenuation, is approximately 12 dB. The attenuator block is tunable while the simulation is running, and it generates noise proportional to the insertion loss.

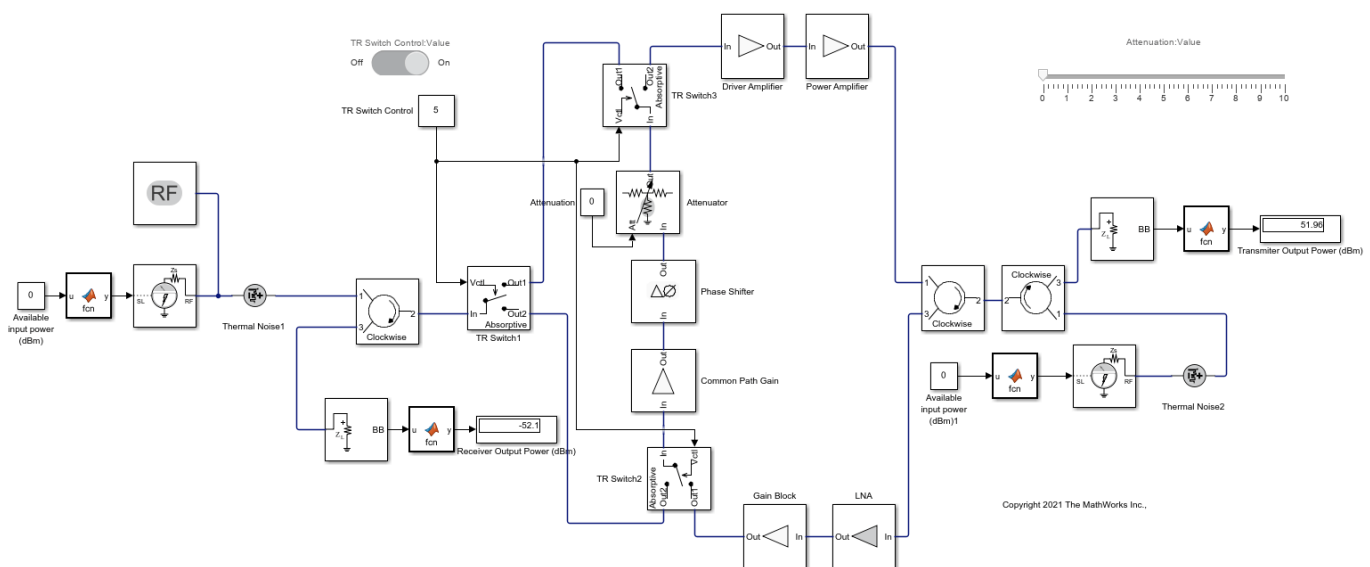
Select **Run** from the **Simulation** tab to simulate the TR module.

```
open_system("TRModule.slx")
sim("TRModule.slx")
```

ans =

```
Simulink.SimulationOutput:
    tout: [80001x1 double]

SimulationMetadata: [1x1 Simulink.SimulationMetadata]
ErrorMessage: [0x0 char]
```



Noise Figure Measurement

Measure the noise figure of the receiver using the Noise Figure Testbench. This testbench provides Stimulus and Response ports that must be connected to the input and output of the receive chain. In this model, the TR module is configured in receive mode, and the transmit input port is terminated on a 50 ohm impedance.

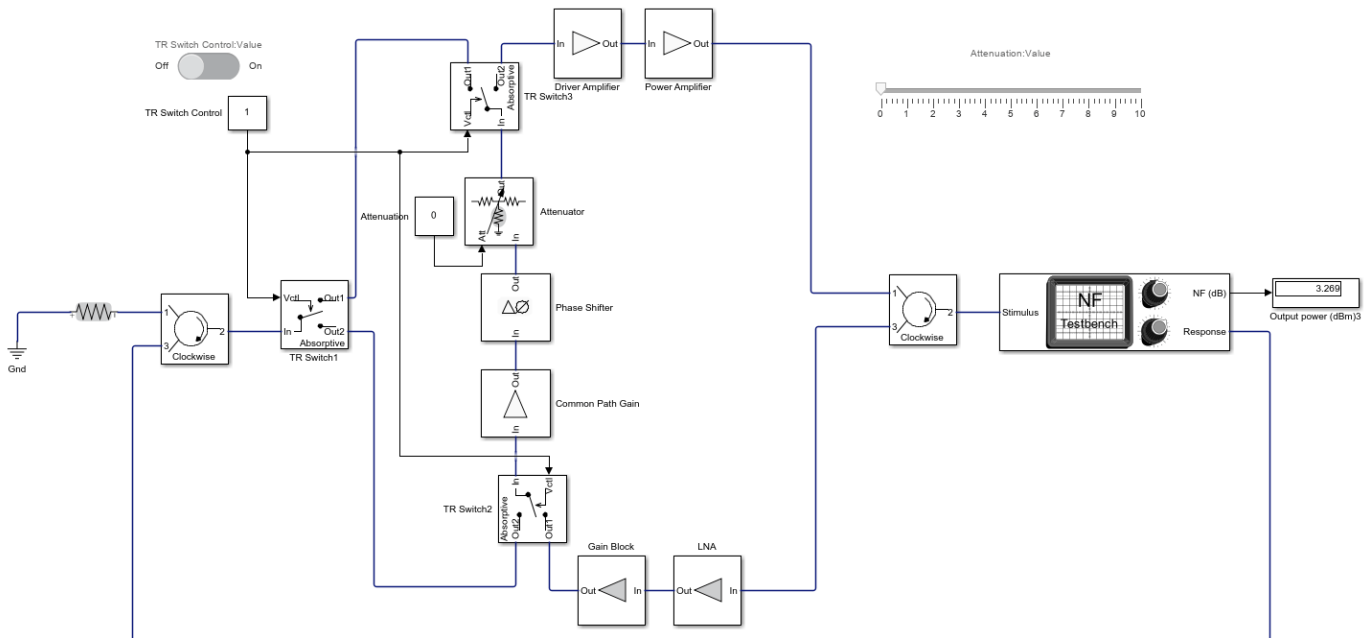
Select **Run** from the **Simulation** tab to simulate the TR module with the Noise Figure Testbench to measure the noise figure of the TR Module.

```
open_system("TRModule_NF.slx")
sim("TRModule_NF.slx")
```

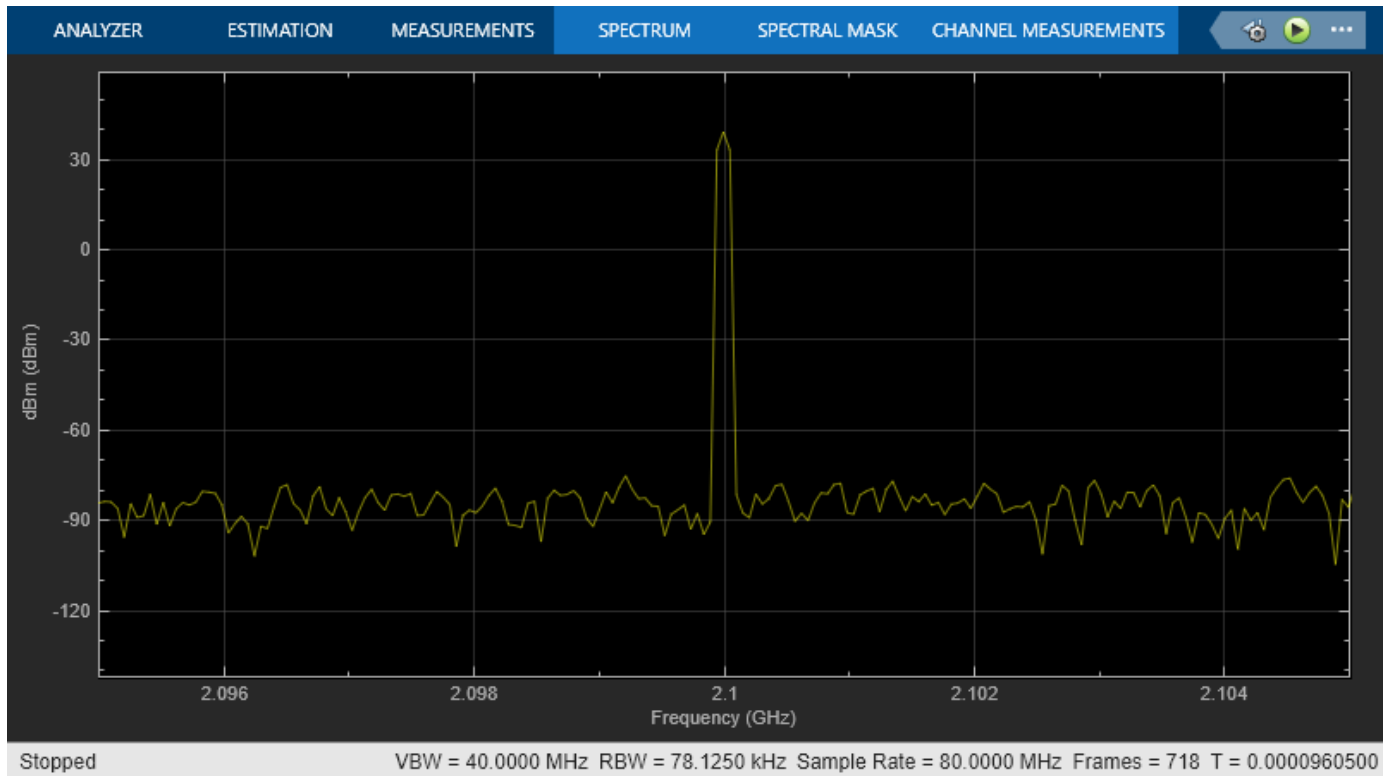
ans =

```
Simulink.SimulationOutput:
    tout: [8001x1 double]

SimulationMetadata: [1x1 Simulink.SimulationMetadata]
ErrorMessage: [0x0 char]
```



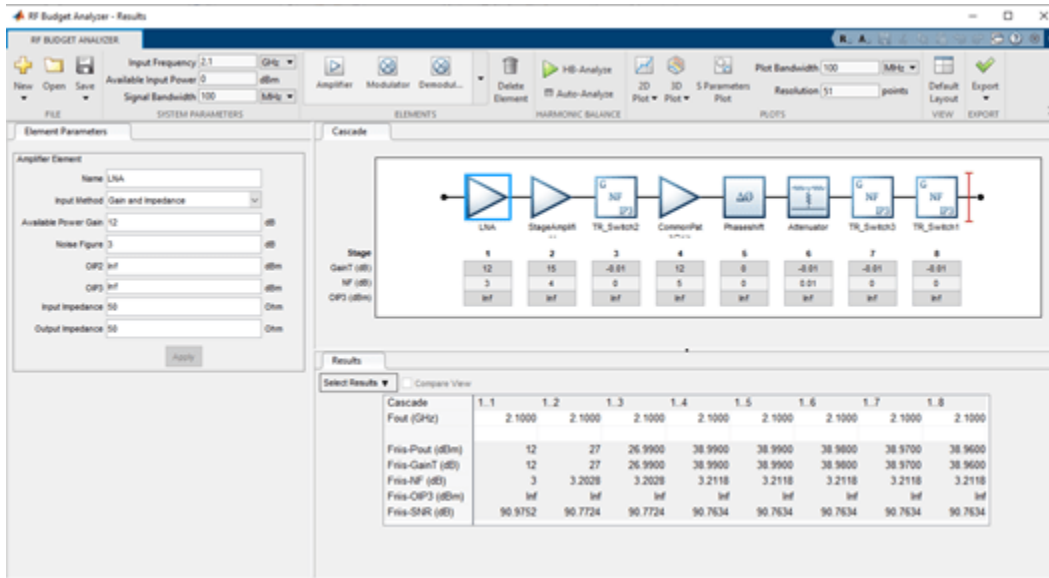
Copyright 2021 The MathWorks Inc.



You can experiment and increase the attenuation; however, high gain and low noise of the amplifiers at the beginning of the chain, the total noise figure remains almost unchanged. If you change the attenuation slider while running the simulation, you need to clear the noise history to reset the measurement in the noise figure testbench. The measured Noise Figure value is approximately 3.2 dB, which can be confirmed with a receiver budget analysis in the **RF Budget Analysis** app as shown below.

```
load('RX_Budget.mat')
```

Type `show(rfb)` to display the system in the **RF Budget Analyzer** app.



See Also

Circulator | SPST | SPDT

Related Examples

- “Transmission Lines, Delay-Based and Lumped Models” on page 8-47

Massive MIMO Hybrid Beamforming with RF Impairments

This example shows how hybrid beamforming is employed at the transmit end of a massive MIMO communications system, using techniques for both multi-user and single-user systems. The example employs full channel sounding for determining the channel state information at the transmitter. It partitions the required precoding into digital baseband and analog RF components, using different techniques for multi-user and single-user systems. Simplified all-digital receivers recover the multiple transmitted data streams to highlight the common figures of merit for a communications system, namely, EVM, and BER.

The example employs a scattering-based spatial channel model which accounts for the transmit/receive spatial locations and antenna patterns. A simpler static-flat MIMO channel is also offered for link validation purposes.

The example uses RF Toolbox™, RF Blockset (TM), Communications Toolbox™, Phased Array System Toolbox™, and Parallel Computing Toolbox™ to simulate massive MIMO hybrid beamforming communication systems. .

Introduction

The ever-growing demand for high data rate and more user capacity increases the need to use the available spectrum more efficiently. Multi-user MIMO (MU-MIMO) improves the spectrum efficiency by allowing a base station (BS) transmitter to communicate simultaneously with multiple mobile stations (MS) receivers using the same time-frequency resources. Massive MIMO allows the number of BS antenna elements to be on the order of tens or hundreds, thereby also increasing the number of data streams in a cell to a large value.

The next generation, 5G, wireless systems use millimeter wave (mmWave) bands to take advantage of their wider bandwidth. The 5G systems also deploy large scale antenna arrays to mitigate severe propagation loss in the mmWave band.

Compared to current wireless systems, the wavelength in the mmWave band is much smaller. Although this allows an array to contain more elements within the same physical dimension, it becomes much more expensive to provide one transmit-receive (TR) module, or an RF chain, for each antenna element. Hybrid transceivers are a practical solution as they use a combination of analog beamformers in the RF and digital beamformers in the baseband domains, with fewer RF chains than the number of transmit elements [1].

This example uses a multi-user MIMO-OFDM system to highlight the partitioning of the required precoding into its digital baseband and RF analog components at the transmitter end. Building on the system highlighted in the “MIMO-OFDM Precoding with Phased Arrays” (Phased Array System Toolbox) example, this example shows the formulation of the transmit-end precoding matrices and their application to a MIMO-OFDM system.

```
tic
s = rng(67); % Set RNG state for repeatability
```

System Parameters

Define system parameters for the example. Modify these parameters to explore their impact on the system.

```
% Multi-user system with single/multiple streams per user
prm.numUsers = 3; % Number of users
```

```

prm.numSTSVec = [2 1 1]; % Number of independent data streams per user
prm.numSTS = sum(prm.numSTSVec); % Must be a power of 2
prm.numTx = prm.numSTS*8; % Number of BS transmit antennas (power of 2)
prm.numRx = prm.numSTSVec*4; % Number of receive antennas, per user (any >= numSTSVec)

% Each user has the same modulation
prm.bitsPerSubCarrier = 4; % 2: QPSK, 4: 16QAM, 6: 64QAM, 8: 256QAM
prm.numDataSymbols = 10; % Number of OFDM data symbols

% MS positions: assumes BS at origin
% Angles specified as [azimuth;elevation] degrees
% az in range [-180 180], el in range [-90 90], e.g. [45;0]
maxRange = 1000; % all MSs within 1000 meters of BS
prm.mobileRanges = randi([1 maxRange],1,prm.numUsers);
prm.mobileAngles = [rand(1,prm.numUsers)*360-180; ...
rand(1,prm.numUsers)*180-90];

prm.fc = 28e9; % 28 GHz system
prm.chanSRate = 100e6; % Channel sampling rate, 100 Msps
prm.ChanType = 'MIMO'; % Channel options: 'Scattering', 'MIMO'
prm.NFig = 8; % Noise figure (increase to worsen, 5-10 dB)
prm.nRays = 500; % Number of rays for Frf, Fbb partitioning

```

Define OFDM modulation parameters used for the system.

```

prm.FFTLength = 256;
prm.CyclicPrefixLength = 64;
prm.numCarriers = 234;
prm.NullCarrierIndices = [1:7 129 256-5:256]'; % Guards and DC
prm.PilotCarrierIndices = [26 54 90 118 140 168 204 232]';
nonDataIdx = [prm.NullCarrierIndices; prm.PilotCarrierIndices];
prm.CarriersLocations = setdiff((1:prm.FFTLength)', sort(nonDataIdx));

numSTS = prm.numSTS;
numTx = prm.numTx;
numRx = prm.numRx;
numSTSVec = prm.numSTSVec;
codeRate = 1/3; % same code rate per user
numTails = 6; % number of termination tail bits
prm.numFrmBits = numSTSVec.*(prm.numDataSymbols*prm.numCarriers* ...
prm.bitsPerSubCarrier*codeRate)-numTails;
prm.modMode = 2^prm.bitsPerSubCarrier; % Modulation order
% Account for channel filter delay
numPadSym = 3; % number of symbols to zeropad
prm.numPadZeros = numPadSym*(prm.FFTLength+prm.CyclicPrefixLength);

```

Define transmit and receive arrays and positional parameters for the system.

```

prm.cLight = physconst('LightSpeed');
prm.lambda = prm.cLight/prm.fc;

% Get transmit and receive array information
[isTxURA,expFactorTx,isRxURA,expFactorRx] = helperArrayInfo(prm,true);

% Transmit antenna array definition
% Array locations and angles
prm.posTx = [0;0;0]; % BS/Transmit array position, [x;y;z], meters
if isTxURA

```

```

% Uniform Rectangular array
txarray = phased.PartitionedArray(...
    'Array',phased.URA([expFactorTx numSTS],0.5*prm.lambda),...
    'SubarraySelection',ones(numSTS,numTx),'SubarraySteering','Custom');
else
% Uniform Linear array
txarray = phased.ULA(numTx, 'ElementSpacing',0.5*prm.lambda, ...
    'Element',phased.IsotropicAntennaElement('BackBaffled',false));
end
prm.posTxElem = getElementPosition(txarray)/prm.lambda;

spLoss = zeros(prm.numUsers,1);
prm.posRx = zeros(3,prm.numUsers);
for uIdx = 1:prm.numUsers

% Receive arrays
if isRxURA(uIdx)
% Uniform Rectangular array
rxarray = phased.PartitionedArray(...
    'Array',phased.URA([expFactorRx(uIdx) numSTSVec(uIdx)], ...
    0.5*prm.lambda),'SubarraySelection',ones(numSTSVec(uIdx), ...
    numRx(uIdx)), 'SubarraySteering','Custom');
    prm.posRxElem = getElementPosition(rxarray)/prm.lambda;
else
if numRx(uIdx)>1
% Uniform Linear array
rxarray = phased.ULA(numRx(uIdx), ...
    'ElementSpacing',0.5*prm.lambda, ...
    'Element',phased.IsotropicAntennaElement);
    prm.posRxElem = getElementPosition(rxarray)/prm.lambda;
else
rxarray = phased.IsotropicAntennaElement;
prm.posRxElem = [0; 0; 0]; % LCS
end
end

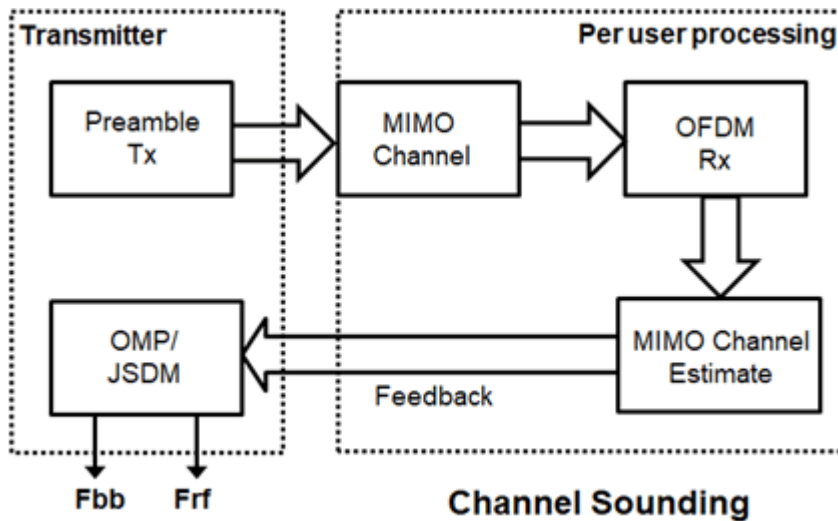
% Mobile positions
[xRx,yRx,zRx] = sph2cart(deg2rad(prm.mobileAngles(1,uIdx)), ...
    deg2rad(prm.mobileAngles(2,uIdx)), ...
    prm.mobileRanges(uIdx));
prm.posRx(:,uIdx) = [xRx;yRx;zRx];
[toRxRange,toRxAng] = rangeangle(prm.posTx,prm.posRx(:,uIdx));
spLoss(uIdx) = fspl(toRxRange,prm.lambda);
end

```

Channel State Information

For a spatially multiplexed system, availability of channel information at the transmitter allows for precoding to be applied to maximize the signal energy in the direction and channel of interest. Under the assumption of a slowly varying channel, this is facilitated by sounding the channel first. The BS sounds the channel by using a reference transmission, that the MS receiver uses to estimate the channel. The MS transmits the channel estimate information back to the BS for calculation of the precoding needed for the subsequent data transmission.

The following schematic shows the processing for the channel sounding modeled.



For the chosen MIMO system, a preamble signal is sent over all transmitting antenna elements, and processed at the receiver accounting for the channel. The receiver antenna elements perform pre-amplification, OFDM demodulation, and frequency domain channel estimation for all links.

```

% Generate the preamble signal
prm.numSTS = numTx; % set to numTx to sound out all channels
preambleSig = helperGenPreamble(prm);

% Transmit preamble over channel
prm.numSTS = numSTS; % keep same array config for channel
[rxPreSig,chanDelay] = helperApplyMUChannel(preambleSig,prm,spLoss);

% Channel state information feedback
hDp = cell(prm.numUsers,1);
prm.numSTS = numTx; % set to numTx to estimate all links
for uIdx = 1:prm.numUsers

    % Front-end amplifier gain and thermal noise
    rxPreAmp = phased.ReceiverPreamp( ...
        'Gain',spLoss(uIdx), ... % account for path loss
        'NoiseFigure',prm.NFig,'ReferenceTemperature',290, ...
        'SampleRate',prm.chanSRate);
    rxPreSigAmp = rxPreAmp(rxPreSig{uIdx});
    % scale power for used sub-carriers
    rxPreSigAmp = rxPreSigAmp * (sqrt(prm.FFTLength - ...
        length(prm.NullCarrierIndices))/prm.FFTLength);

    % OFDM demodulation
    rxOFDM = ofdmmod(rxPreSigAmp(chanDelay(uIdx)+1: ...
        end-(prm.numPadZeros- chanDelay(uIdx)),:),prm.FFTLength, ...
        prm.CyclicPrefixLength,prm.CyclicPrefixLength, ...
        prm.NullCarrierIndices,prm.PilotCarrierIndices);

    % Channel estimation from preamble
    % numCarr, numTx, numRx
    hDp{uIdx} = helperMIMOChannelEstimate(rxOFDM(:,1:numTx,:),prm);
end
  
```

For a multi-user system, the channel estimate is fed back from each MS, and used by the BS to determine the precoding weights. The example assumes perfect feedback with no quantization or implementation delays.

Hybrid Beamforming

The example uses the orthogonal matching pursuit (OMP) algorithm [3] for a single-user system and the joint spatial division multiplexing (JSDM) technique [2, 4] for a multi-user system, to determine the digital baseband F_{bb} and RF analog F_{rf} precoding weights for the selected system configuration.

For a single-user system, the OMP partitioning algorithm is sensitive to the array response vectors A_t . Ideally, these response vectors account for all the scatterers seen by the channel, but these are unknown for an actual system and channel realization, so a random set of rays within a 3-dimensional space to cover as many scatterers as possible is used. The `prm.nRays` parameter specifies the number of rays.

For a multi-user system, JSDM groups users with similar transmit channel covariance together and suppresses the inter-group interference by an analog precoder based on the block diagonalization method [5]. Here each user is assigned to be in its own group, thereby leading to no reduction in the sounding or feedback overhead.

```
% Calculate the hybrid weights on the transmit side
if prm.numUsers==1
    % Single-user OMP
    % Spread rays in [az;el]=[-180:180;-90:90] 3D space, equal spacing
    % txang = [-180:360/prm.nRays:180; -90:180/prm.nRays:90];
    txang = [rand(1,prm.nRays)*360-180;rand(1,prm.nRays)*180-90]; % random
    At = steervec(prm.posTxElem,txang);
    AtExp = complex(zeros(prm.numCarriers,size(At,1),size(At,2)));
    for carrIdx = 1:prm.numCarriers
        AtExp(carrIdx,,:) = At; % same for all sub-carriers
    end

    % Orthogonal matching pursuit hybrid weights
    [Fbb,Frf] = omphyweights(hDp{1},numSTS,numSTS,AtExp);

    v = Fbb; % set the baseband precoder (Fbb)
    % Frf is same across subcarriers for flat channels
    mFrf = permute(mean(Frf,1),[2 3 1]);
else
    % Multi-user Joint Spatial Division Multiplexing
    [Fbb,mFrf] = helperJSDMTransmitWeights(hDp,prm);

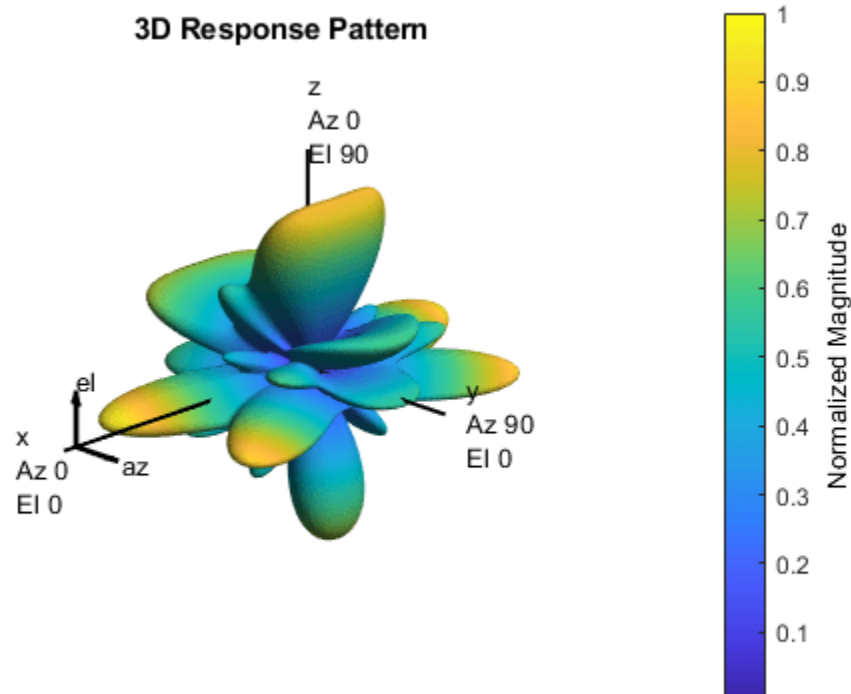
    % Multi-user baseband precoding
    % Pack the per user CSI into a matrix (block diagonal)
    steeringMatrix = zeros(prm.numCarriers,sum(numSTSVec),sum(numSTSVec));
    for uIdx = 1:prm.numUsers
        stsIdx = sum(numSTSVec(1:uIdx-1))+(1:numSTSVec(uIdx));
        steeringMatrix(:,stsIdx,stsIdx) = Fbb{uIdx}; % Nst-by-Nsts-by-Nsts
    end
    v = permute(steeringMatrix,[1 3 2]);
end

% Transmit array pattern plots
if isTxURA
```

```

% URA element response for the first subcarrier
pattern(txarray,prm.fc,-180:180,-90:90,'Type','efield', ...
        'ElementWeights',mFrf.*squeeze(v(1,:,:)), ...
        'PropagationSpeed',prm.cLight);
else % ULA
% Array response for first subcarrier
wts = mFrf.*squeeze(v(1,:,:));
figure
pattern(txarray,prm.fc,-180:180,-90:90,'Type','efield', ...
        'Weights',wts(:,1),'PropagationSpeed',prm.cLight);
end
prm.numSTS = numSTS; % revert back for data transmission

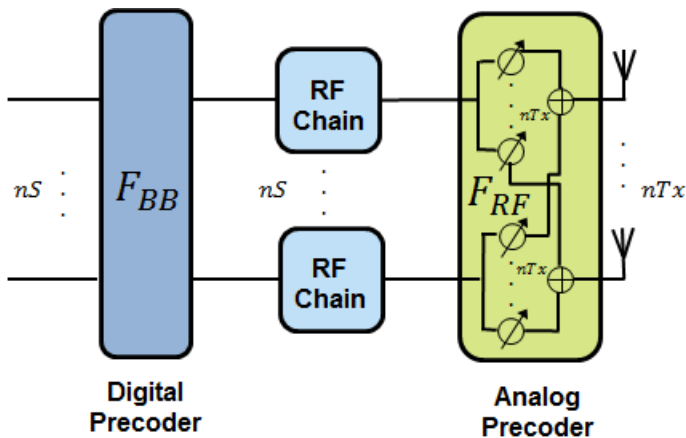
```



For the wideband OFDM system modeled, the analog weights, $mFrf$, are the averaged weights over the multiple subcarriers. The array response pattern shows distinct data streams represented by the stronger lobes. These lobes indicate the spread or separability achieved by beamforming. The “Introduction to Hybrid Beamforming” (Phased Array System Toolbox) example compares the patterns realized by the optimal, fully digital approach, with those realized from the selected hybrid approach, for a single-user system.

Data Transmission

The example models an architecture where each data stream maps to an individual RF chain and each antenna element is connected to each RF chain. This is shown in the following diagram.



Next, we configure the system's data transmitter. This processing includes channel coding, bit mapping to complex symbols, splitting of the individual data stream to multiple transmit streams, baseband precoding of the transmit streams, OFDM modulation with pilot mapping and RF analog beamforming for all the transmit antennas employed.

```
% Convolutional encoder
encoder = comm.ConvolutionalEncoder( ...
    'TrellisStructure',poly2trellis(7,[133 171 165]), ...
    'TerminationMethod','Terminated');

txDataBits = cell(prm.numUsers, 1);
gridData = complex(zeros(prm.numCarriers,prm.numDataSymbols,numSTS));
for uIdx = 1:prm.numUsers
    % Generate mapped symbols from bits per user
    txDataBits{uIdx} = randi([0,1],prm.numFrmBits(uIdx),1);
    encodedBits = encoder(txDataBits{uIdx});

    % Bits to QAM symbol mapping
    mappedSym = qammod(encodedBits,prm.modMode,'InputType','bit', ...
        'UnitAveragePower',true);

    % Map to layers: per user, per symbol, per data stream
    stsIdx = sum(numSTSVec(1:(uIdx-1)))+(1:numSTSVec(uIdx));
    gridData(:,:,stsIdx) = reshape(mappedSym,prm.numCarriers, ...
        prm.numDataSymbols,numSTSVec(uIdx));
end

% Apply precoding weights to the subcarriers, assuming perfect feedback
preData = complex(zeros(prm.numCarriers,prm.numDataSymbols,numSTS));
for symIdx = 1:prm.numDataSymbols
    for carrIdx = 1:prm.numCarriers
        Q = squeeze(v(carrIdx,:,:));
        normQ = Q * sqrt(numTx)/norm(Q,'fro');
        preData(carrIdx,symIdx,:) = squeeze(gridData(carrIdx,symIdx,:)).' ...
            * normQ;
    end
end

% Multi-antenna pilots
pilots = helperGenPilots(prm.numDataSymbols,numSTS);
```



```

% OFDM modulation of the data
txOFDM = ofdmmod(preData,prm.FFTLength,prm.CyclicPrefixLength,...
                prm.NullCarrierIndices,prm.PilotCarrierIndices,pilots);
% scale power for used sub-carriers
txOFDM = txOFDM * (prm.FFTLength/ ...
                sqrt((prm.FFTLength-length(prm.NullCarrierIndices))));

% Generate preamble with the feedback weights and prepend to data
preambleSigD = helperGenPreamble(prm,v);
txSigSTS = [preambleSigD;txOFDM];
[m,n] = size(txSigSTS);
k = prm.numTx;

```

Add RF Impairments from the Tx RFFE

Add RF impairments from the Tx RFFE and instantiate the Parallel Pool of workers and load RF system object

```

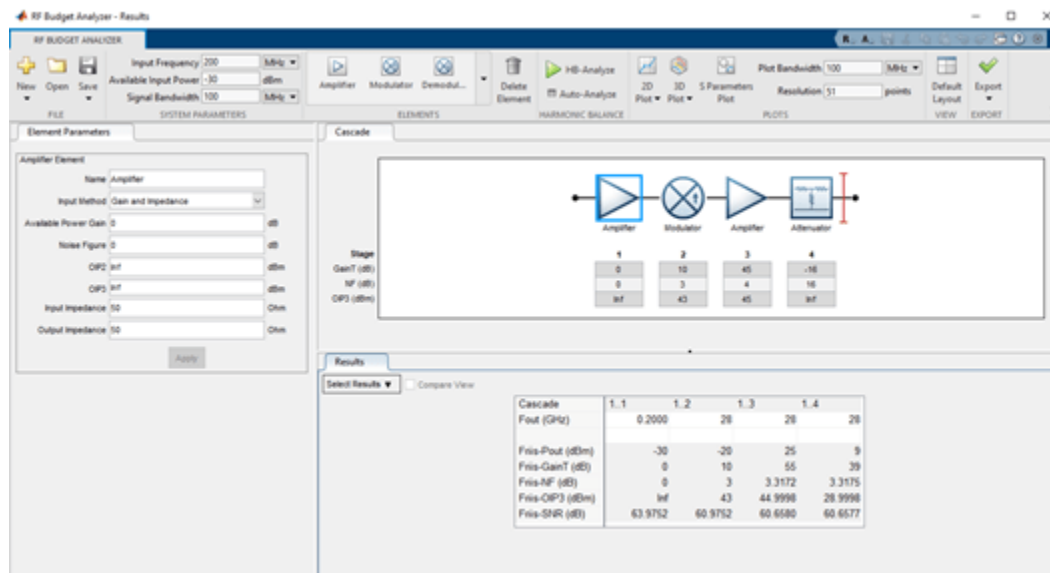
poolobj = gcp;
if isempty(poolobj)
    poolsize = 0;
else
    poolsize = poolobj.NumWorkers;
end

```

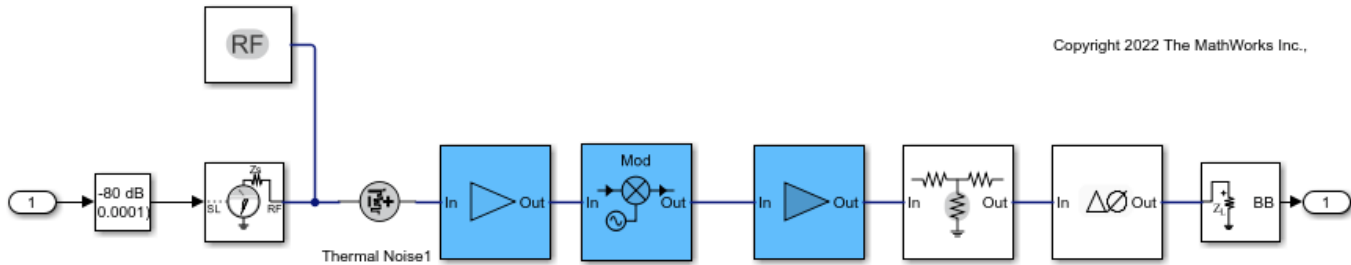
```
load rfb_mu_tx
```

Starting parallel pool (parpool) using the 'Processes' profile ...
Connected to parallel pool with 20 workers.

Type `rfBudgetAnalyzer(rfb)` command at the MATLAB® command line to visualize the RF system in the **RF Budget Analyzer** app.



```
load rf_sys_mu_tx
open_system('rfs_mu_tx_chan');
```



Add behavior of each RF path of each RF chain

```
tx_out(1:m,1:prm.numTx,1:n) = complex(zeros(m,prm.numTx,n));
```

```
% Use 4 workers to handle each data stream for each user
```

```
parfor i = 1 : n
    for j = 1 : k
        tx_out(:,j,i) = step(rfs_mu_tx,txSigSTS(:,i));
        release(rfs_mu_tx);
    end
end
```

Apply the Analog Beamformers to the Output signal from the RF front end

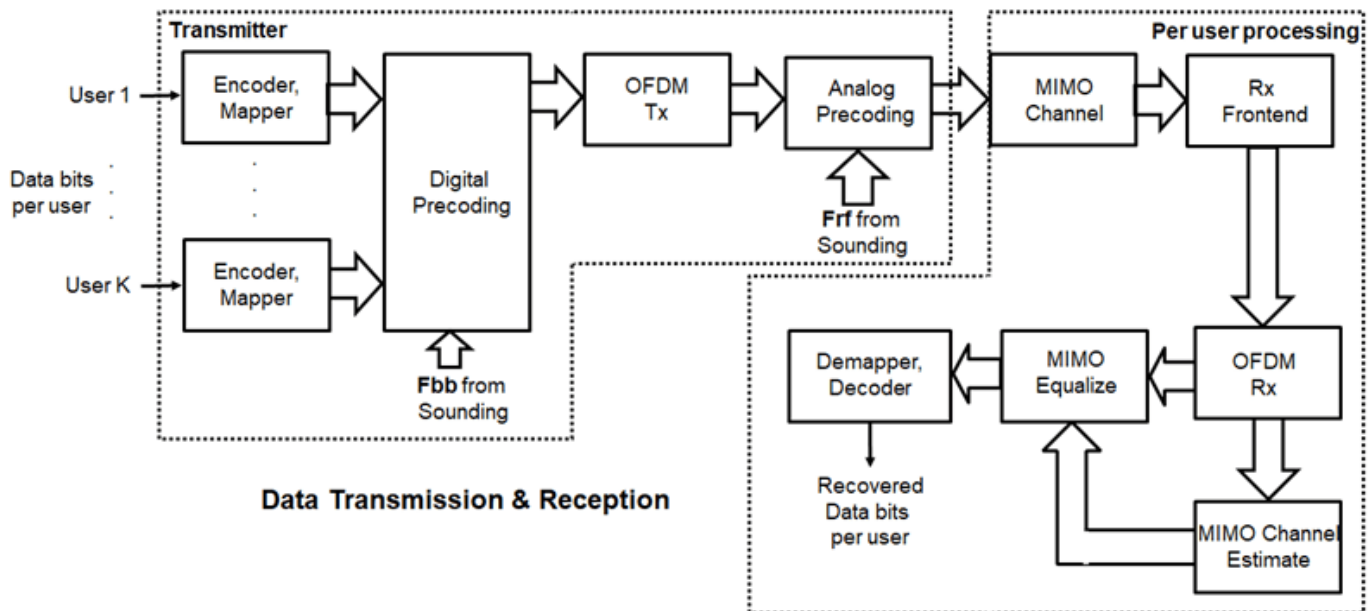
```
delete(poolobj);
txsig_int(1:m,1:prm.numTx,1:n) = complex(zeros(m,prm.numTx,n));
```

```
for i = 1 : n
    for j = 1 : prm.numTx
        txsig_int(:,j,i) = tx_out(:,j,i)*mFrf(i,j);
    end
end
txsig = sum(txsig_int,3);
```

Parallel pool using the 'Processes' profile is shutting down.

For the selected, fully connected RF architecture, each antenna element uses `prm.numSTS` phase shifters, as given by the individual columns of the `mFrf` matrix.

The processing for the data transmission and reception modeled is shown below.



Signal Propagation

The example offers an option for spatial MIMO channel and a simpler static-flat MIMO channel for validation purposes.

The scattering model uses a single-bounce ray tracing approximation with a parametrized number of scatterers. For this example, the number of scatterers is set to 100. The 'Scattering' option models the scatterers placed randomly within a sphere around the receiver, similar to the one-ring model [6].

The channel models allow path-loss modeling and both line-of-sight (LOS) and non-LOS propagation conditions. The example assumes non-LOS propagation and isotropic antenna element patterns with linear or rectangular geometry.

`% Apply a spatially defined channel to the transmit signal`

```
[rxSig,chanDelay] = helperApplyMUChannel(txsig,prm,spLoss,preambleSig);
```

The same channel is used for both sounding and data transmission. The data transmission has a longer duration and is controlled by the number of data symbols parameter, `prm.numDataSymbols`. The channel evolution between the sounding and transmission stages is modeled by prepending the preamble signal to the data signal. The preamble primes the channel to a valid state for the data transmission, and is ignored from the channel output.

For a multi-user system, independent channels per user are modeled.

Receive Amplification and Signal Recovery

The receiver modeled per user compensates for the path loss by amplification and adds thermal noise. Like the transmitter, the receiver used in a MIMO-OFDM system contains many stages including OFDM demodulation, MIMO equalization, QAM demapping, and channel decoding.

```
hfig = figure('Name','Equalized symbol constellation per stream');
scFact = ((prm.FFTLength-length(prm.NullCarrierIndices))...
```

```

        /prm.FFTLength^2)/numTx;
nVar = noisepow(prm.chanSRate,prm.NFig,290)/scFact;
decoder = comm.ViterbiDecoder('InputFormat','Unquantized', ...
    'TrellisStructure',poly2trellis(7, [133 171 165]), ...
    'TerminationMethod','Terminated','OutputDataType','double');

for uIdx = 1:prm.numUsers
    stsU = numSTSVec(uIdx);
    stsIdx = sum(numSTSVec(1:(uIdx-1)))+(1:stsU);

    % Front-end amplifier gain and thermal noise
    rxPreAmp = phased.ReceiverPreamp( ...
        'Gain',spLoss(uIdx), ... % account for path loss
        'NoiseFigure',prm.NFig,'ReferenceTemperature',290, ...
        'SampleRate',prm.chanSRate);
    rxSigAmp = rxPreAmp(rxSig{uIdx});

    % Scale power for occupied sub-carriers
    rxSigAmp = rxSigAmp*(sqrt(prm.FFTLength-length(prm.NullCarrierIndices)) ...
        /prm.FFTLength);

    % OFDM demodulation
    rxOFDM = ofdmmod(rxSigAmp(chanDelay(uIdx)+1: ...
        end-(prm.numPadZeros-chanDelay(uIdx)),:),prm.FFTLength, ...
        prm.CyclicPrefixLength,prm.CyclicPrefixLength, ...
        prm.NullCarrierIndices,prm.PilotCarrierIndices);

    % Channel estimation from the mapped preamble
    hD = helperMIMOChannelEstimate(rxOFDM(:,1:numSTS,:),prm);

    % MIMO equalization
    % Index into streams for the user of interest
    [rxEq,CSI] = helperMIMOEqualize(rxOFDM(:,numSTS+1:end,:),hD(:,stsIdx,:));

    % Soft demodulation
    rxSyms = rxEq(:)/sqrt(numTx);
    rxLLRBits = qamdemod(rxSyms,prm.modMode,'UnitAveragePower',true, ...
        'OutputType','approxllr','NoiseVariance',nVar);

    % Apply CSI prior to decoding
    rxLLRtmp = reshape(rxLLRBits,prm.bitsPerSubCarrier,[], ...
        prm.numDataSymbols,stsU);
    csitmp = reshape(CSI,1,[],1,numSTSVec(uIdx));
    rxScaledLLR = rxLLRtmp.*csitmp;

    % Soft-input channel decoding
    rxDecoded = decoder(rxScaledLLR(:));

    % Decoded received bits
    rxBits = rxDecoded(1:prm.numFrmBits(uIdx));

    % Plot equalized symbols for all streams per user
    scaler = ceil(max(abs([real(rxSyms(:)); imag(rxSyms(:))]))));
    for i = 1:stsU
        subplot(prm.numUsers, max(numSTSVec), (uIdx-1)*max(numSTSVec)+i);
        plot(reshape(rxEq(:,i))/sqrt(numTx), [], 1), '.');
        axis square
        xlim(gca,[-scaler scaler]);
    end
end

```

```

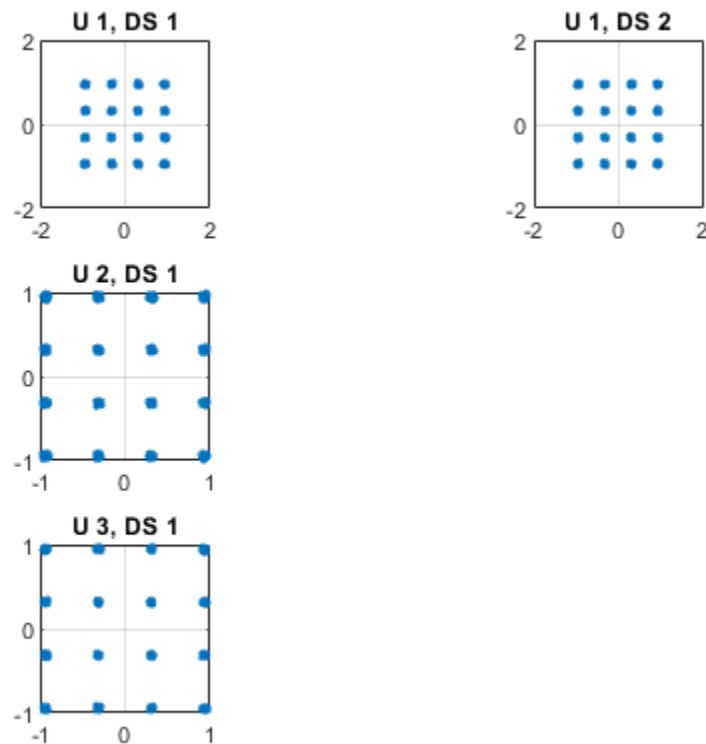
ylim(gca,[-scaler scaler]);
title(['U ' num2str(uIdx) ', DS ' num2str(i)]);
grid on;
end

% Compute and display the EVM
evm = comm.EVM('Normalization','Average constellation power', ...
    'ReferenceSignalSource','Estimated from reference constellation', ...
    'ReferenceConstellation', ...
    qammod((0:prm.modMode-1)',prm.modMode,'UnitAveragePower',1));
rmsEVM = evm(rxSyms);
disp(['User ' num2str(uIdx)]);
disp([' RMS EVM (%) = ' num2str(rmsEVM)]);

% Compute and display bit error rate
ber = comm.ErrorRate;
measures = ber(txDataBits{uIdx},rxBits);
fprintf(' BER = %.5f; No. of Bits = %d; No. of errors = %d\n', ...
    measures(1),measures(3),measures(2));
end
toc

User 1
RMS EVM (%) = 2.7924
BER = 0.00000; No. of Bits = 6234; No. of errors = 0
User 2
RMS EVM (%) = 2.3394
BER = 0.00000; No. of Bits = 3114; No. of errors = 0
User 3
RMS EVM (%) = 1.8391
BER = 0.00000; No. of Bits = 3114; No. of errors = 0
Elapsed time is 508.345275 seconds.

```



For the MIMO system modeled, the displayed receive constellation of the equalized symbols offers a qualitative assessment of the reception. The actual bit error rate offers the quantitative figure by comparing the actual transmitted bits with the received decoded bits per user.

```
rng(s);           % restore RNG state
```

Conclusion and Further Exploration

The example highlights the use of hybrid beamforming for multi-user MIMO-OFDM systems. It allows you to explore different system configurations for a variety of channel models by changing a few system-wide parameters.

The set of configurable parameters includes the number of users, number of data streams per user, number of transmit/receive antenna elements, array locations, and channel models. Adjusting these parameters you can study the parameters' individual or combined effects on the overall system. As examples, vary:

- the number of users, `prm.numUsers`, and their corresponding data streams, `prm.numSTSVec`, to switch between multi-user and single-user systems, or
- the channel type, `prm.ChanType`, or
- the number of rays, `prm.nRays`, used for a single-user system.

Explore the following helper functions used by the example:

- `helperApplyMUChannel.m`

- `helperArrayInfo.m`
- `helperGenPreamble.m`
- `helperGenPilots.m`
- `helperJSDMTransmitWeights.m`
- `helperMIMOChannelEstimate.m`
- `helperMIMOEqualize.m`

References

- [1] Molisch, A. F., et al. "Hybrid Beamforming for Massive MIMO: A Survey." *IEEE® Communications Magazine*, Vol. 55, No. 9, September 2017, pp. 134-141.
- [2] Li Z., S. Han, and A. F. Molisch. "Hybrid Beamforming Design for Millimeter-Wave Multi-User Massive MIMO Downlink." *IEEE ICC 2016, Signal Processing for Communications Symposium*.
- [3] El Ayach, Oma, et al. "Spatially Sparse Precoding in Millimeter Wave MIMO Systems." *IEEE Transactions on Wireless Communications*, Vol. 13, No. 3, March 2014, pp. 1499-1513.
- [4] Adhikary A., J. Nam, J-Y Ahn, and G. Caire. "Joint Spatial Division and Multiplexing - The Large-Scale Array Regime." *IEEE Transactions on Information Theory*, Vol. 59, No. 10, October 2013, pp. 6441-6463.
- [5] Shui, D. S., G. J. Foschini, M. J. Gans and J. M. Kahn. "Fading Correlation and its Effect on the Capacity of Multielement Antenna Systems." *IEEE Transactions on Communications*, Vol. 48, No. 3, March 2000, pp. 502-513.

See Also

Related Examples

- "RF Receiver Modeling for LTE Reception" on page 8-92
- "RF Impairments for 5G NR Downlink Waveforms" on page 8-262
- "PA and DPD Modeling for Dynamic EVM Measurement" on page 8-232

Speed Up PA and DPD simulation

This example shows how to simulate a power amplifier (PA) within a wireless communications system and how to apply digital predistortion (DPD) to improve the transmitter linearity. The simulation speed of the model is increased by using the Idealized Baseband library and hence increasing the abstraction level of the model.

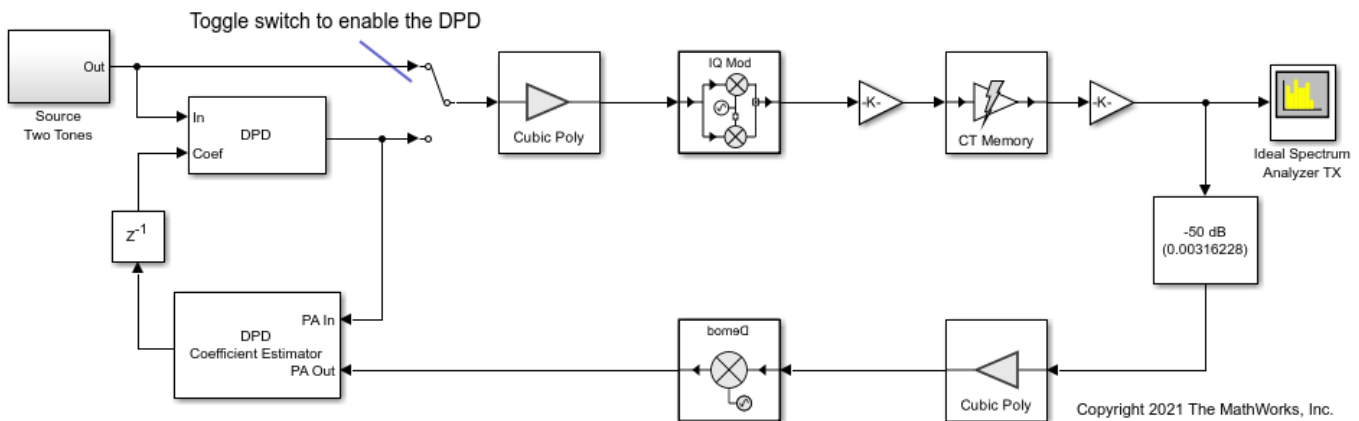
Idealized Baseband Library Model for Two Tones Input Signal

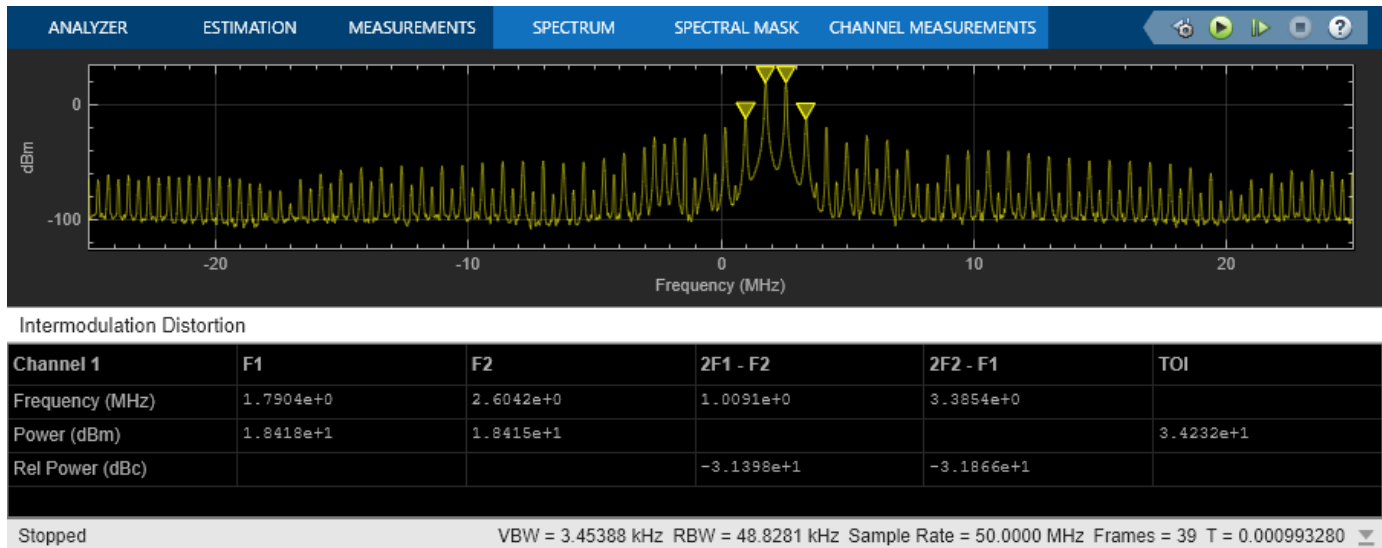
The first model of the RF system uses blocks from the RF Blockset™ Idealized Baseband library. With this library an equivalent baseband modeling approach is adopted. Complex modulated signals represent the transmitted information, assumed to be centered around an implicit carrier frequency.

The blocks from the Idealized Baseband library do not model out-of-band behavior or spurious harmonics generated by nonlinear effects or interfering signals. Additionally, blocks from the Idealized Baseband library do not model impedance mismatches and assume that all blocks are perfectly matched to 50 ohms. As a result, RF models built with the Idealized Baseband library simulate quickly.

The first testbench used for PA and DPD simulation generates two tones to test the PA with or without DPD. In this case, test the model with a narrowband excitation. You can toggle the manual switch in the simulation and verify that the overall IP3 performance of the system is improved.

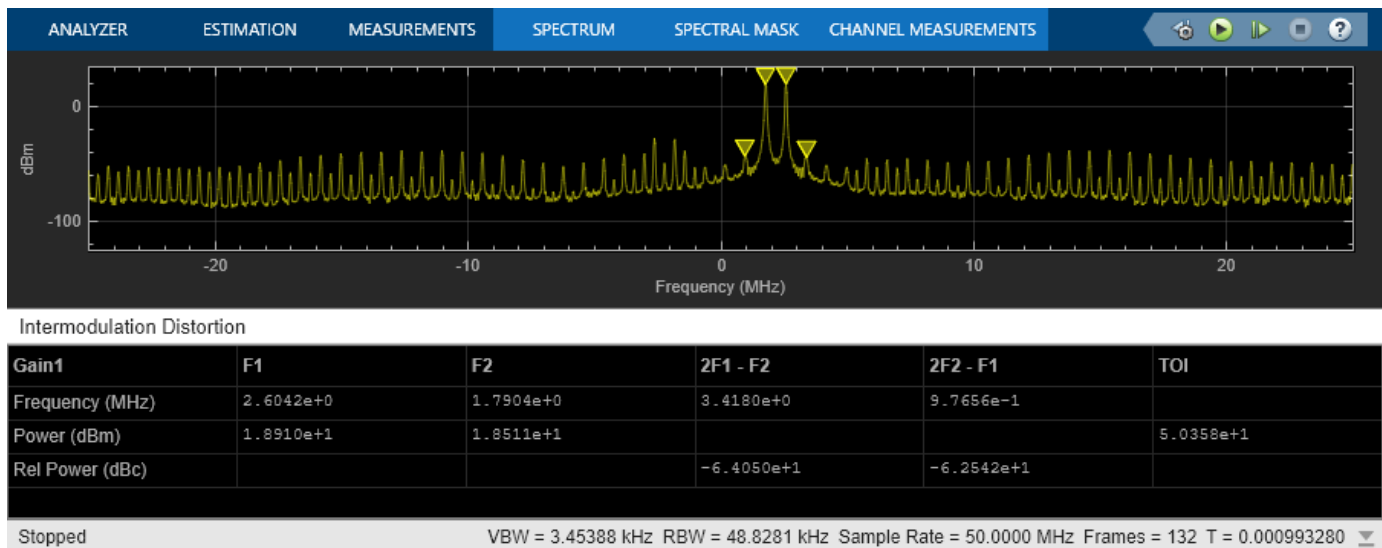
Elapsed time is 3.689039 seconds.





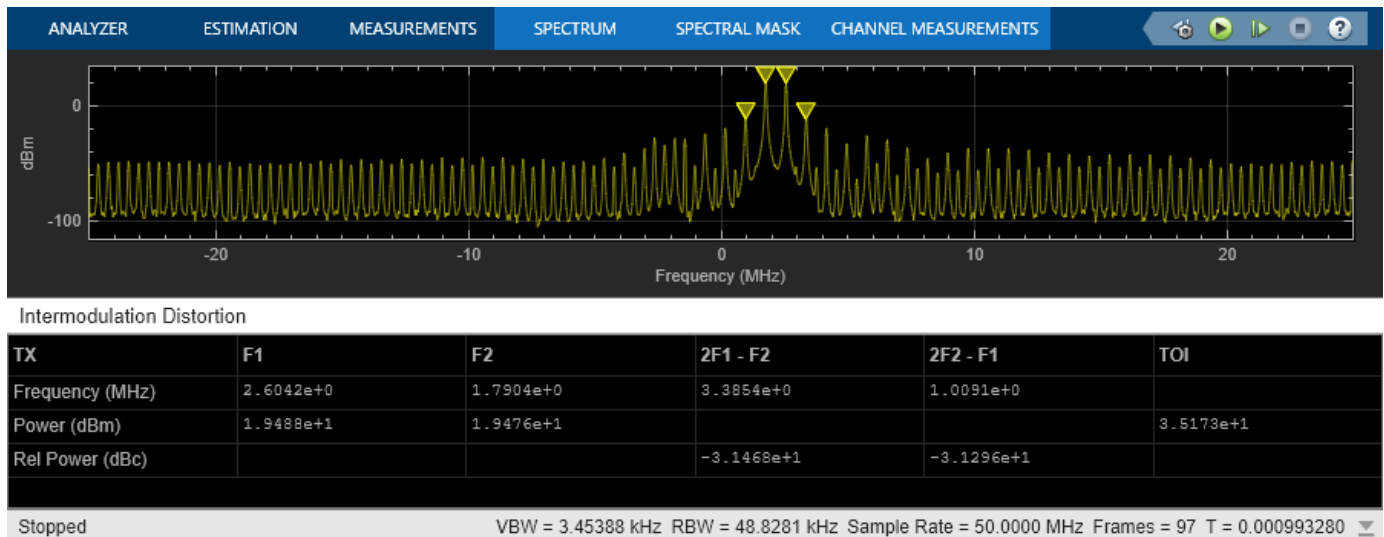
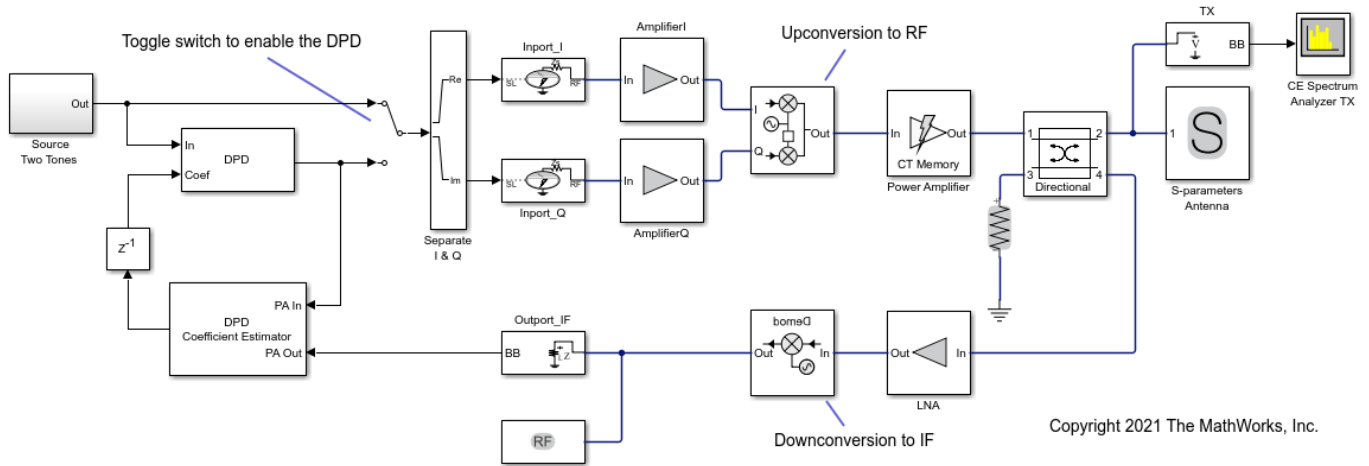
Enable the DPD and verify the improvement on linearity.

Elapsed time is 3.093843 seconds.



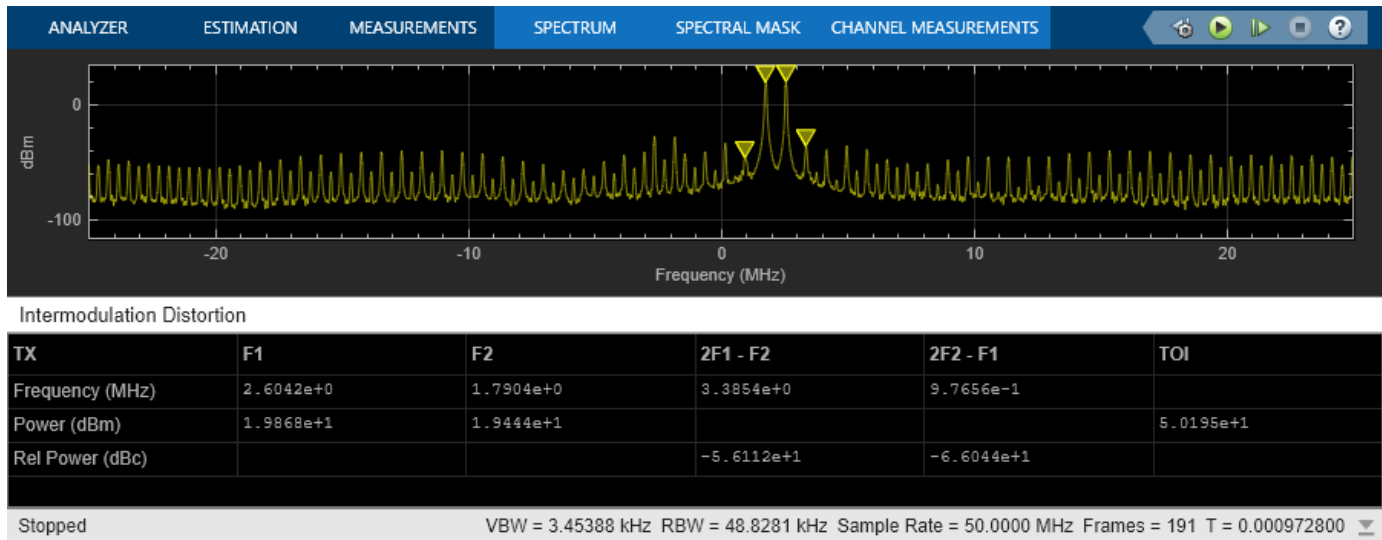
To validate the fidelity of the model, compare it with an equivalent implementation built using blocks from the Circuit Envelope library. In this case, multicarrier effects, second order nonlinearity, and impedance mismatches are taken into account.

Elapsed time is 108.899604 seconds.



Enable the DPD and verify the improvement on linearity.

Elapsed time is 107.505567 seconds.



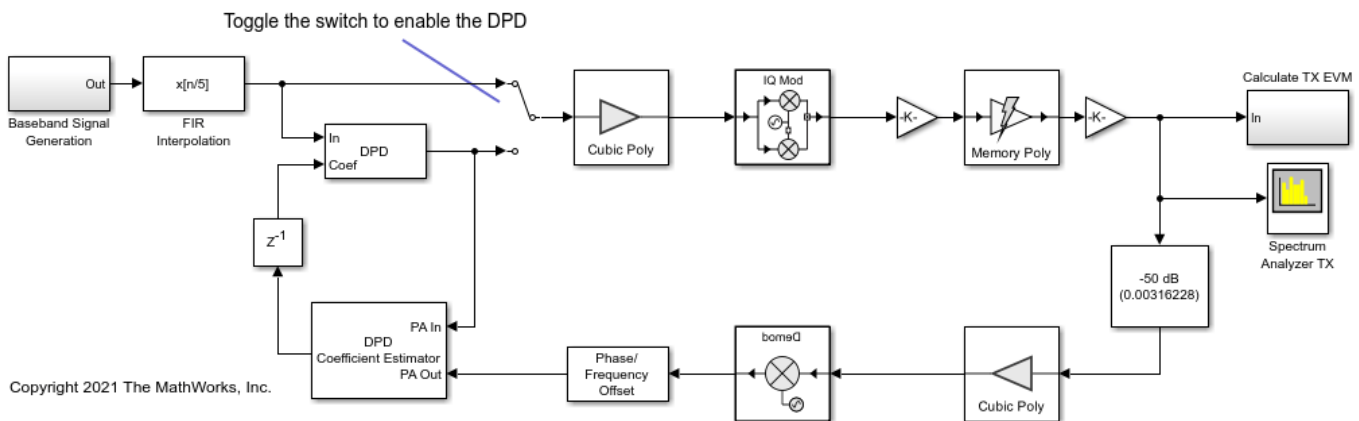
Results obtained with the two models are very similar, but the simulation speed of the idealized baseband model is much faster.

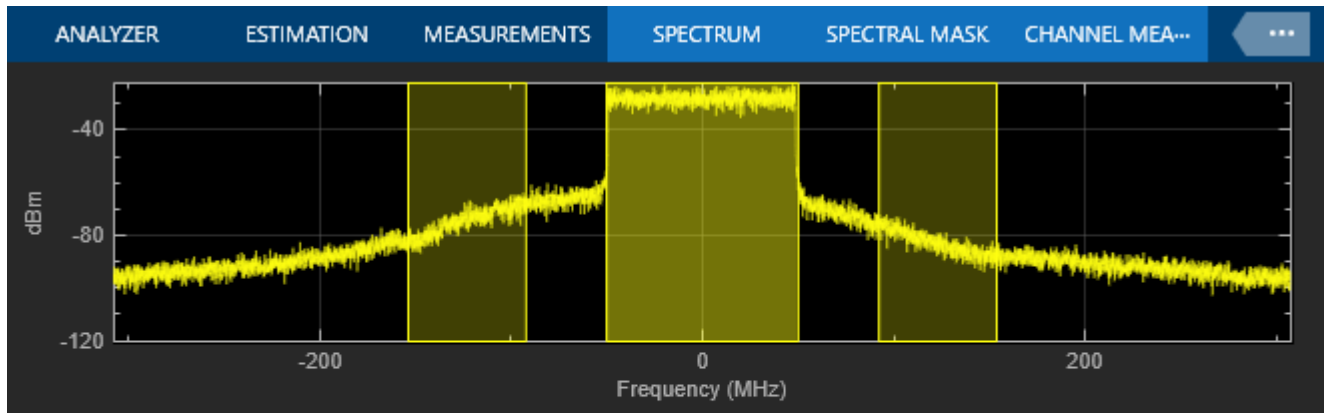
Idealized Baseband Library Model for OFDM Input Signal

Test the Idealized Baseband library model using a modulated OFDM input signal. In this case, the baseband signal has a bandwidth of 100 MHz. The simulation bandwidth of the model has been increased by a factor of five to improve the capture of in-band effects (spectral regrowth) resulting from nonlinearities. EVM is computed using the constellation diagram and ACLR using the spectrum analyzer scope.

In this second testbench, test the RF system and its linearization with a wideband signal. Note that the PA coefficients now represent a different device compared to the one used in the first testbench and that the PA has been characterized for wideband operation.

Elapsed time is 2.774933 seconds.

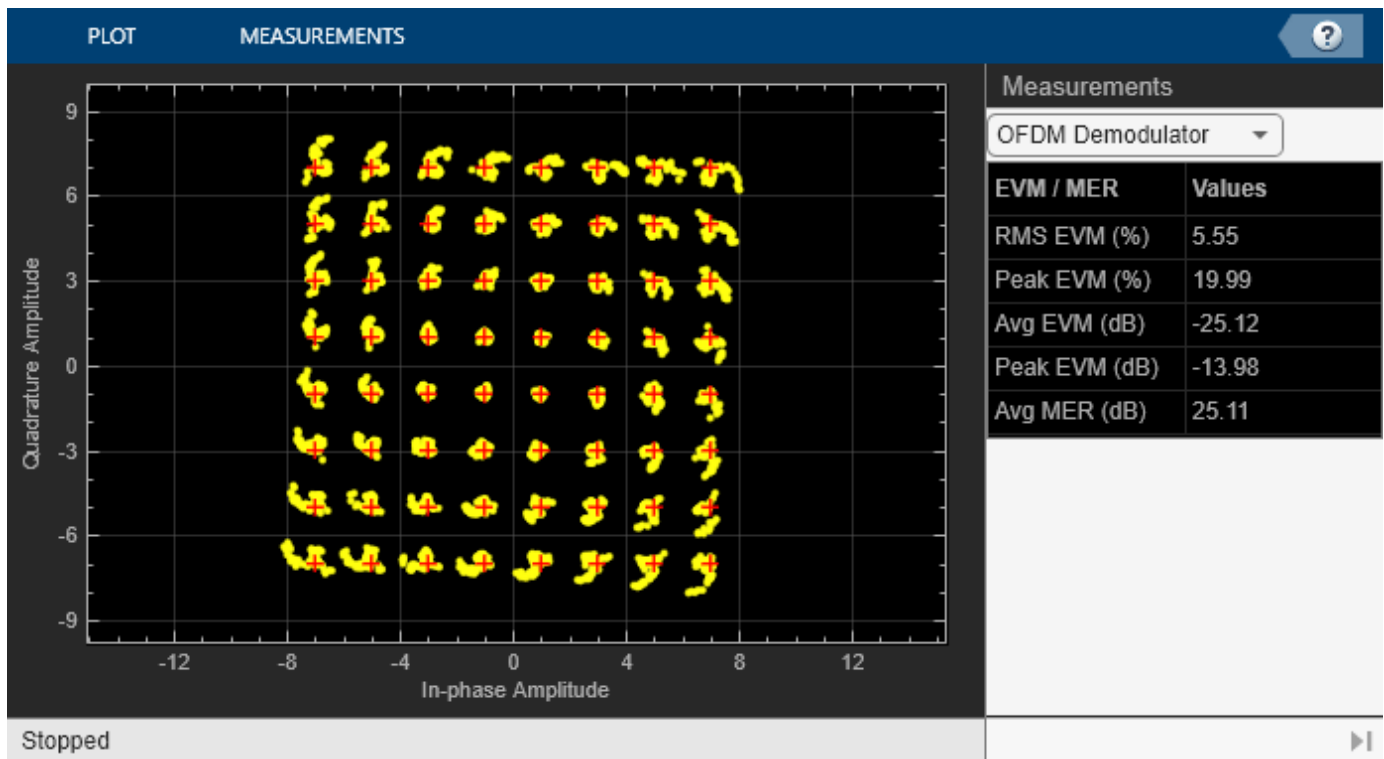




ACPR

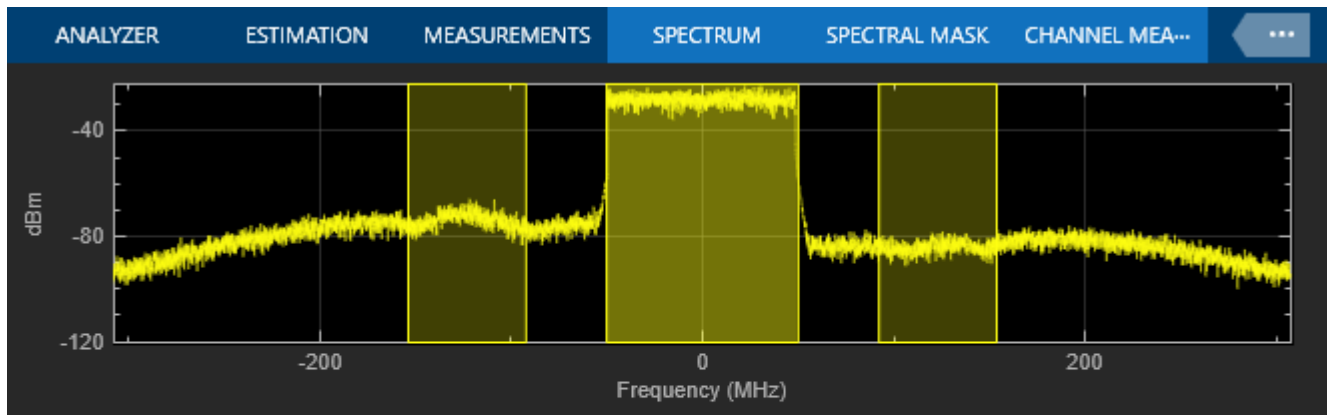
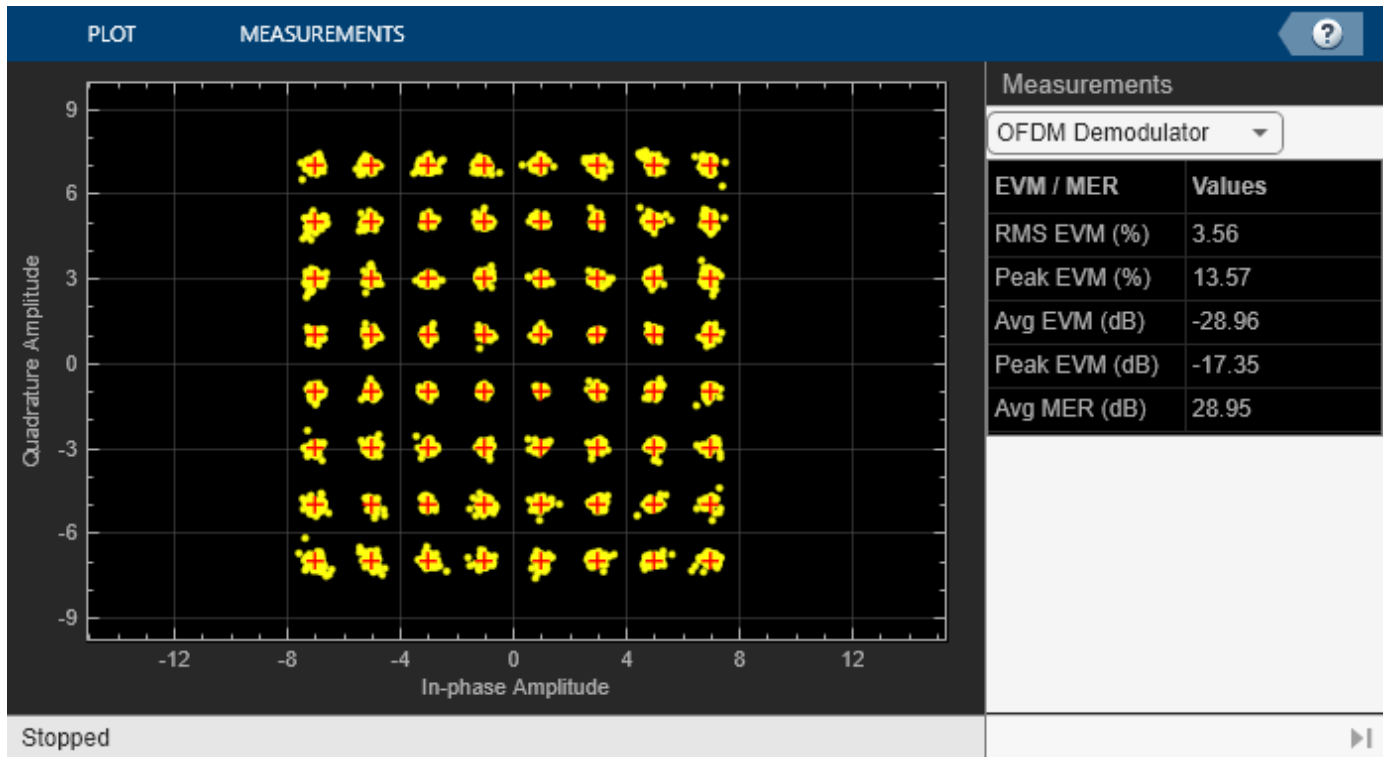
Gain1	ACPR1
Frequency (MHz)	
Lower (Rel Power (dBc))	

Stopped VBW = 7.95775 kHz RBW = 112.500 kHz Sample Rate = 614.400 MHz Frames = 134 T = 0.00020425



Enable the DPD and verify the improvement on linearity.

Elapsed time is 2.115482 seconds.



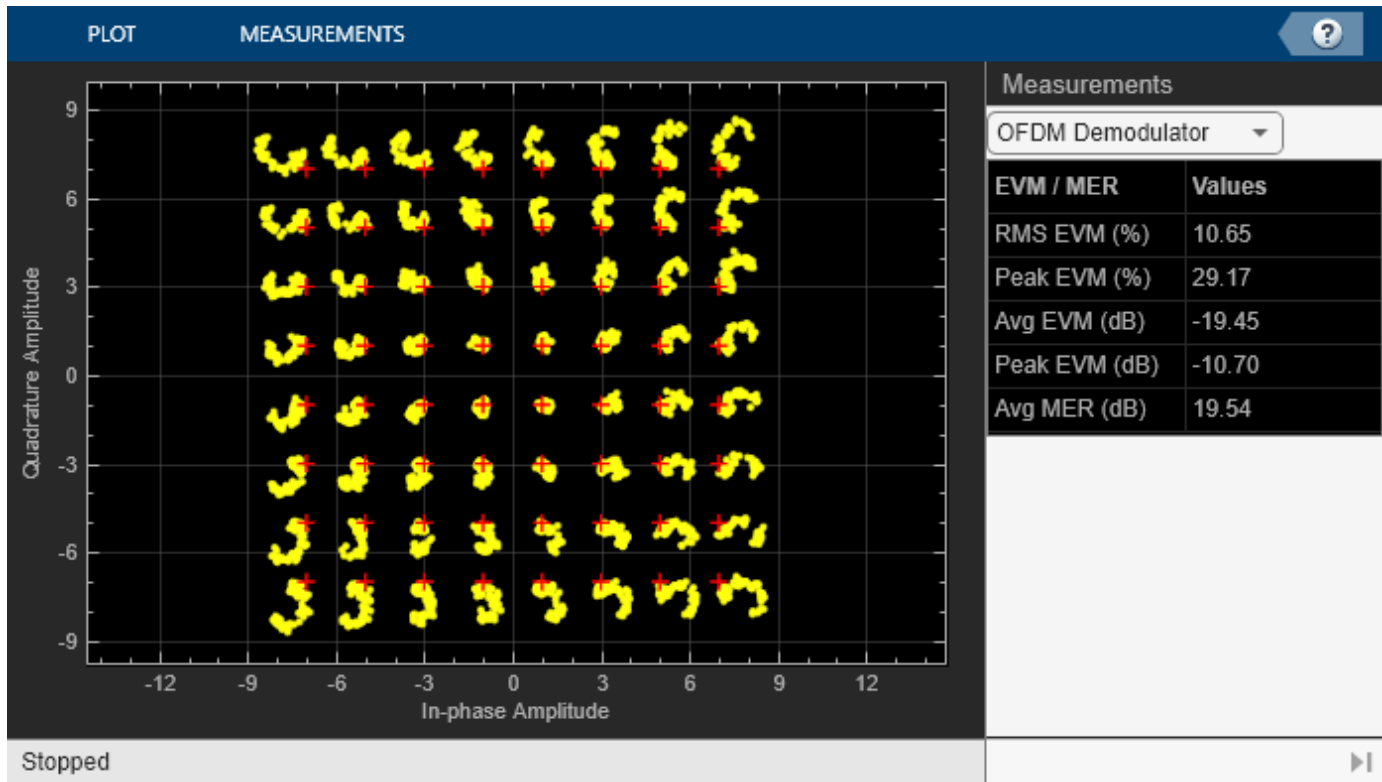
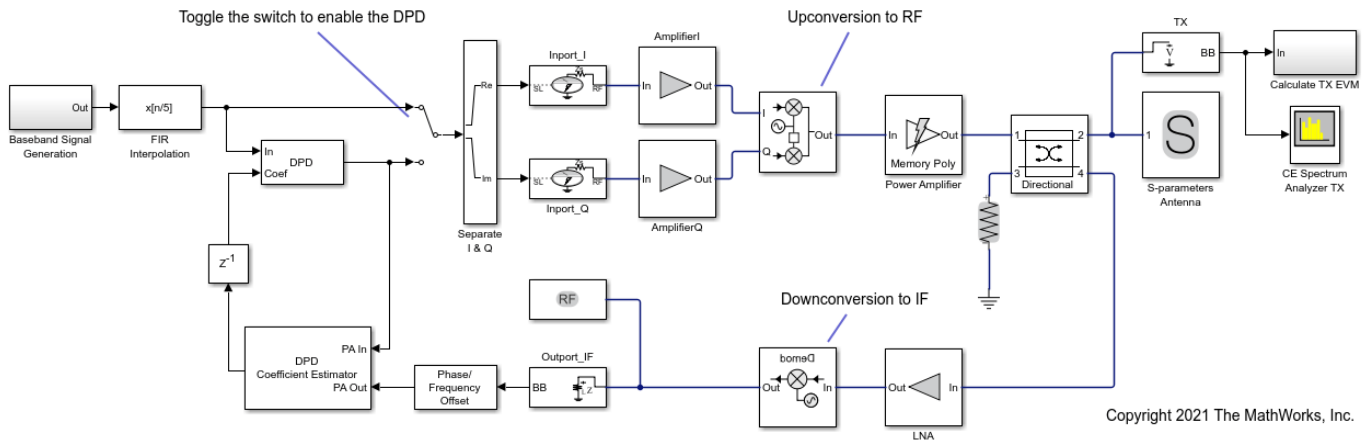
ACPR

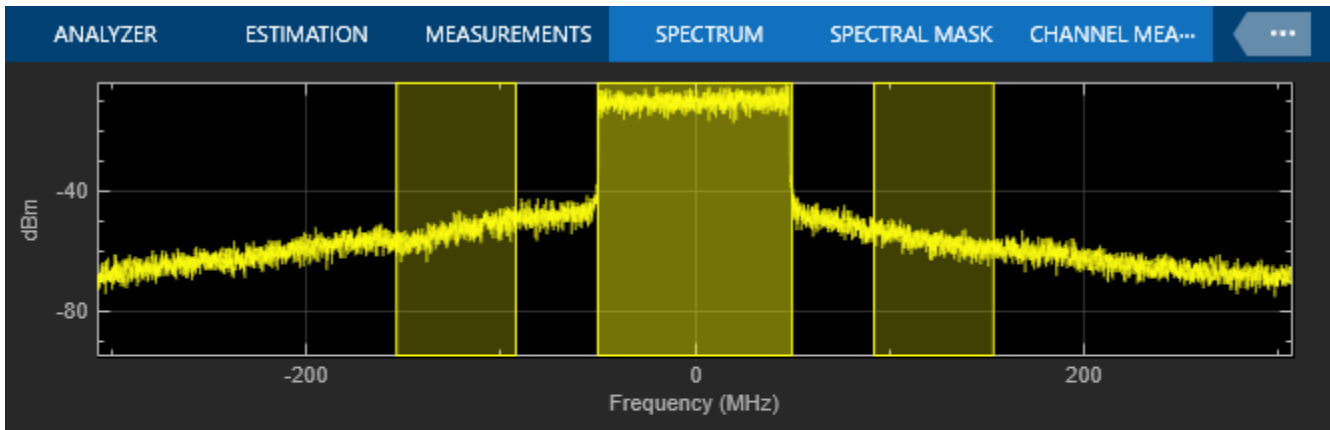
Gain1	ACPR1
Frequency (MHz)	
Lower (Rel Power (dBc))	

Stopped VBW = 7.95775 kHz RBW = 112.500 kHz Sample Rate = 614.400 MHz Frames = 272 T = 0.00021406

Compare the results obtained with an equivalent model built using blocks from the Circuit Envelope Library. In this case the effects of the antenna loading on the coupler and PA are taken into account. These effects lead to additional differences when compared to the model built using blocks from the Idealized Baseband Library.

Elapsed time is 349.242430 seconds.



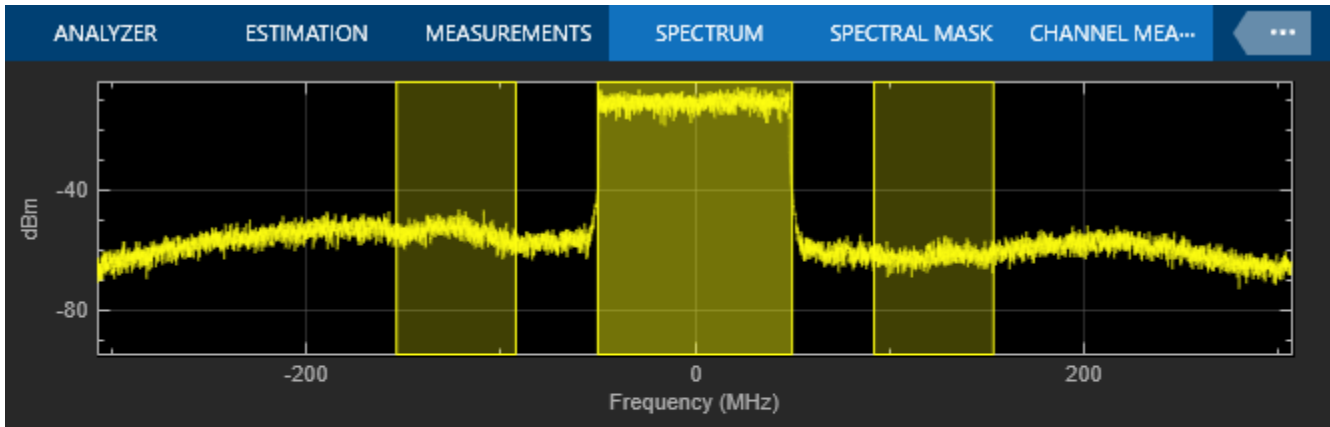


ACPR

TX	ACPR1
Frequency (MHz)	
Lower (Rel Power (dBc))	
Stopped VBW = 7.95775 kHz RBW = 112.500 kHz Sample Rate = 614.400 MHz Frames = 227 T = 0.00020157	

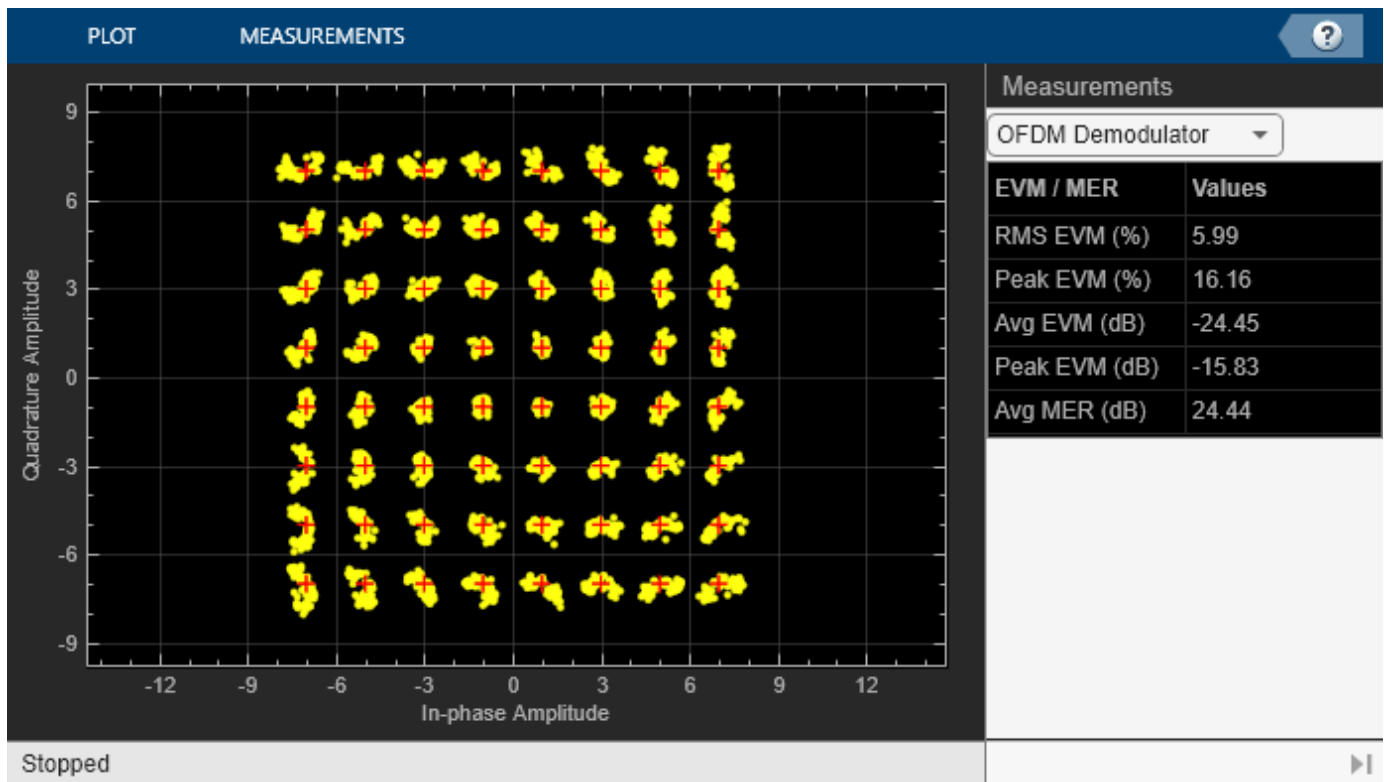
Enable the DPD and verify the improvement on linearity.

Elapsed time is 370.167872 seconds.



ACPR

TX	ACPR1
Frequency (MHz)	
Lower (Rel Power (dBc))	
Stopped VBW = 7.95775 kHz RBW = 112.500 kHz Sample Rate = 614.400 MHz Frames = 479 T = 0.00021317	



See Also

Power Amplifier

Related Examples

- “Power Amplifier Characterization” on page 7-92

Model RF Systems with Antenna Arrays Using RF Blockset Antenna Block

This example shows how to model MIMO receiving and transmitting RF systems including antenna arrays. The design starts from the budget analysis of a single RF chain, and it is then extended to multiple antennas. The RF Blockset Antenna block performs full-wave analysis of the antenna array, enabling high-fidelity modeling of the effects and imperfections coupled with the simulation of the RF system.

In the following sections, you design a MIMO receiver starting from its RF budget analysis. Then, you design a transmitter and connect the two. As a final step, the models are used to transmit and receive a wideband 100 MHz OFDM signal, including beamsteering and clock recovery.

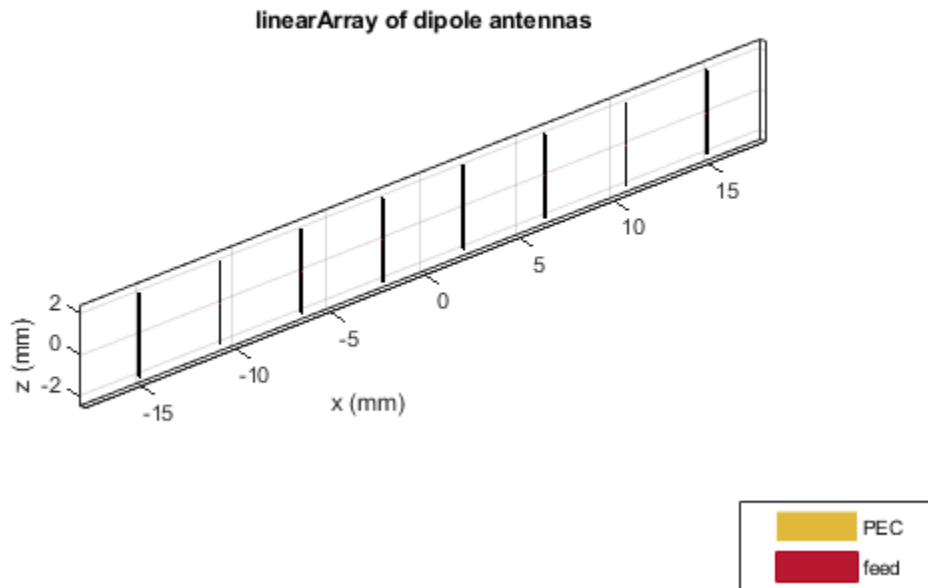
MIMO Receiver System

Design a MIMO receiver (RX) system starting with the budget analysis of a single antenna RF chain. In this example, the input signal is centered around 35GHz, and it is generated by a transmitter (TX) with effective isotropic radiated power (EIRP) equal to 20 dBm, located at a distance of 100 wavelengths away from the receiver.

```
TX_EIRP = 20;  
CF = 35e9;  
lambda = physconst('lightspeed')/CF; % Wavelength (1/m)  
d = 100*lambda; % Distance between TX and RX antennas (m)
```

The RX array is made of eight dipole antennas located at a distance of half a wavelength from each other.

```
arrayRXObj = design(linearArray, CF, dipole);  
arrayRXObj.NumElements = 8;  
arrayRXObj.show;
```



Assume that the TX antenna is similar to the RX antenna and located on the same elevation plane and such that the direction of arrival is normal to the RX array axis.

```
DOA = 0;
Az_RX = 90-DOA; % Azimuth of direction of arrival
El_RX = 0; % Elevation of direction of arrival
```

First calculate the array gain using full-wave analysis, and then approximate the single antenna gain to be equal to entire array gain divided by number of elements in the array, in this case 8.

```
GAntRX = pattern(arrayRXObj,CF,Az_RX,El_RX, 'Type','gain');
GSingleAntRX = GAntRX - 10*log10(8);
```

The next element in the receiver chain is a low noise amplifier. Calculate the input impedance of the amplifier using its S-parameters interpolated at the center frequency. Note that the Touchstone file also includes the noise data.

```
s_amp = sparameters('amplifier.s2p');
Zin = gamma2z(gammaIn(rfinterp1(s_amp,CF),50));
```

Next, compute the impedance of the first antenna element of the array using the impedance of the amplifier determined at the previous step as load to the remaining antenna elements.

```
sp = sparameters(arrayRXObj,CF);
gammaInAnt1 = snp2smp(sp.Parameters,50,1,Zin);
ZAnt_RX = gamma2z(gammaInAnt1);
```

Calculate the free space path loss between the TX and RX.

```
PL = 20*log10(4*pi*d/lambda);
```

If the TX and RX are not perfectly aligned on the same array normal ($\text{DOA} \approx 0$), the 8 received signals have different phases. To coherently receive the transmitted signal, phase shifts are needed for aligning the array beam with the direction of arrival of the received signal. The phase shift beamformer object from Phased Array System Toolbox is used to compute the necessary phase shifts.

```
Beam_Az_RX = DOA; % Beamforming angle (deg)
beamformer = phased.PhaseShiftBeamformer(...
    'SensorArray',phased.ULA('NumElements',8,...
    'ElementSpacing', lambda/2), ...
    'OperatingFrequency',CF,...
    'Direction',[Beam_Az_RX; 0],...
    'WeightsOutputPort',true);
[~, phase_shifts_RX] = beamformer(ones(1,8));
phase_shifts_RX = angle(phase_shifts_RX)'/pi*180;
```

Define the third-order output intercept point in dBm of the first amplifier stage in the RX chain.

```
oIP3_RX = 25.5;
```

Include an additional amplifier stage to each chain in the RX system.

```
G_RX = zeros(1,8);
```

Build a cascade (row vector) of RF receiver elements:

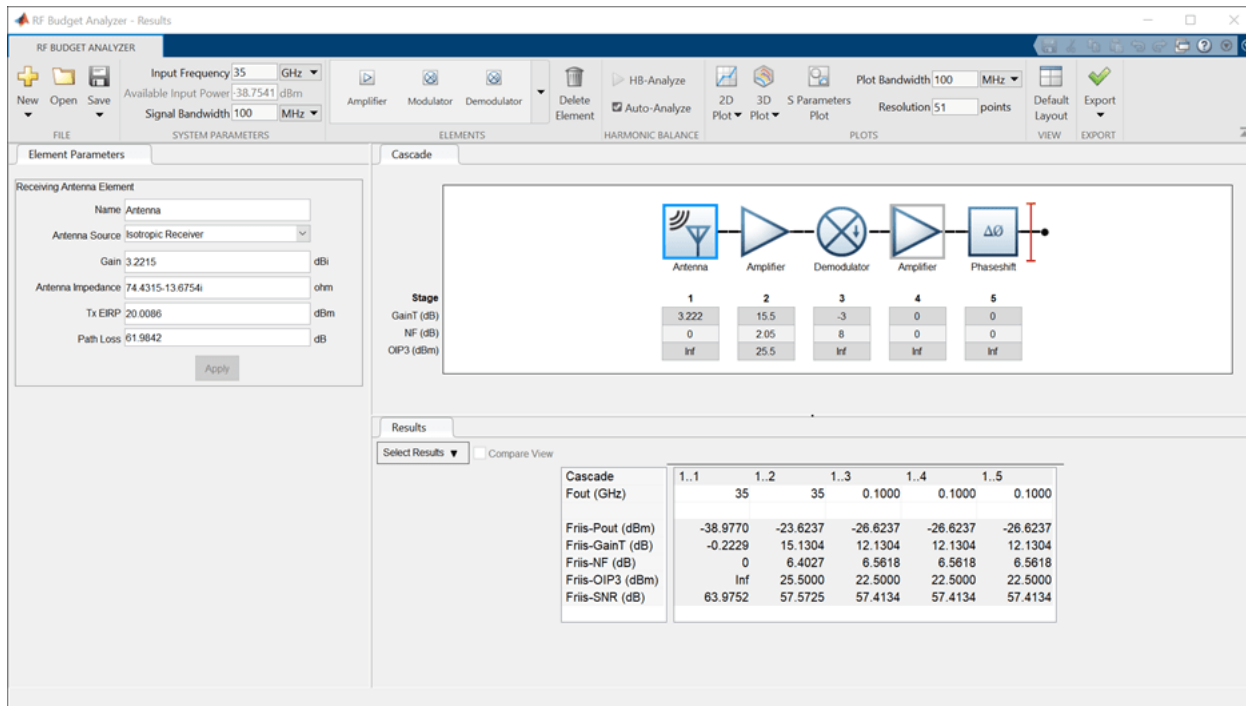
- Antenna defined by gain and impedance, also including TX EIRP and path loss
- Low noise amplifier defined by S-parameters (including noise data) and OIP3
- IF Demodulator stage defined by gain and noise figure
- Additional amplifier stage
- Phase shifters for beamforming

```
% Antenna
elementsRX(1) = rfantenna( ...
    'Type','Receiver', ...
    'Gain',GSingleAntrX, ...
    'Z',ZAnt_RX, ...
    'PathLoss',PL, ...
    'TxEIRP',TX_EIRP);
% Front-end amplifier
elementsRX(2) = amplifier( ...
    'FileName','amplifier.s2p', ...
    'OIP3',oIP3_RX);
% Demodulator
elementsRX(3) = modulator( ...
    'Name','Demodulator', ...
    'Gain',-3, ...
    'NF',8, ...
    'LO',CF-100e6, ...
    'ConverterType','Down');
% Additional amplifier
elementsRX(4) = amplifier( ...
    'Gain',G_RX(1));
% Phase shifter:
elementsRX(5) = phaseshift( ...
    'PhaseShift',phase_shifts_RX(1));
```

Construct an `rfbudget` object from the above elements at the command line.

```
b = rfbudget( ...
    'Elements',elementsRX, ...
    'InputFrequency',CF, ...
    'SignalBandwidth',100e6, ...
    'Solver','Friis');
```

Type `show(b)` command at the command line to visualize the chain in the **RF Budget Analyzer** app.



Note that the Available input power, shown in the **System parameters** section of the app toolstrip, is obtained by adding the transmitter EIRP, minus the path-loss plus the gain of the antenna.

$$P_{av} = TX_EIRP - PL + G_{SingleAnRX}$$

$$P_{av} = -38.7648$$

Create RF Blockset Model for Receiving System

You can export the above cascade as an RF Blockset™ model and copy it to create an eight-chain RF system. When simulating the MIMO RX system, the coupling between antenna elements are captured by replacing the single antenna element used in the RF budget with a full antenna array. This is done by using an Antenna block with the antenna array object `arrayRXObj`.

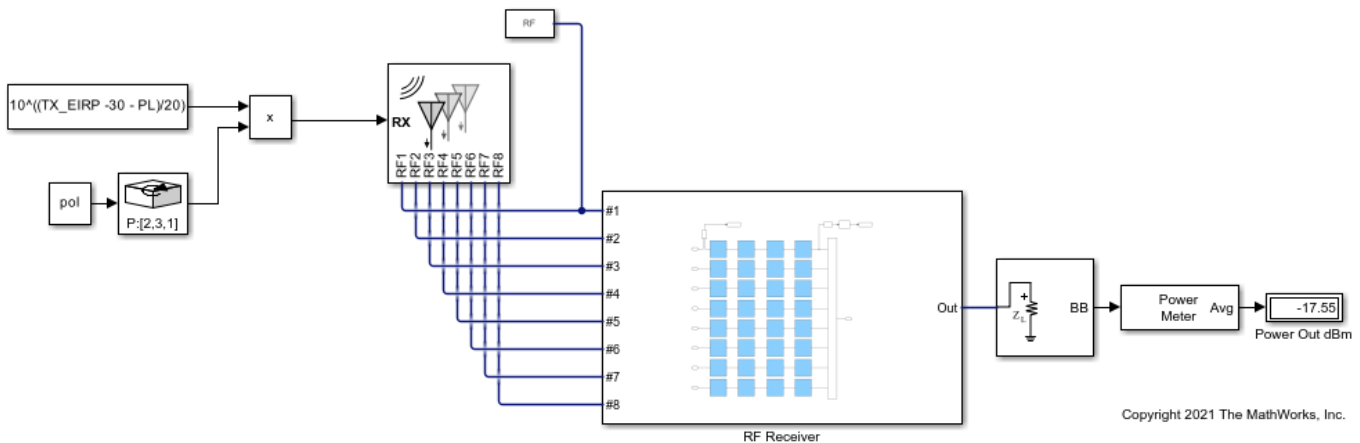
The input to the Antenna block is the received signal described as a normalized power wave split onto the two $[\theta, \phi]$ polarization components. The received power wave, RX , is normalized such that the total power is $\|RX\|^2 = EIRP_{TX} - PL$. The antenna elements in `arrayRXObj`, are z-directed dipoles.

Such an array creates a field that is polarized along the $-\theta$ direction. Assuming that the TX antenna array and the RX antenna array are of the same type, it can assume that the received signal is cast along the θ polarization component.

```
pol = [-1;0];
```

The resulting RX MIMO model includes an antenna block connected to a subsystem RF Receiver representing the RX system, including the eight chains:

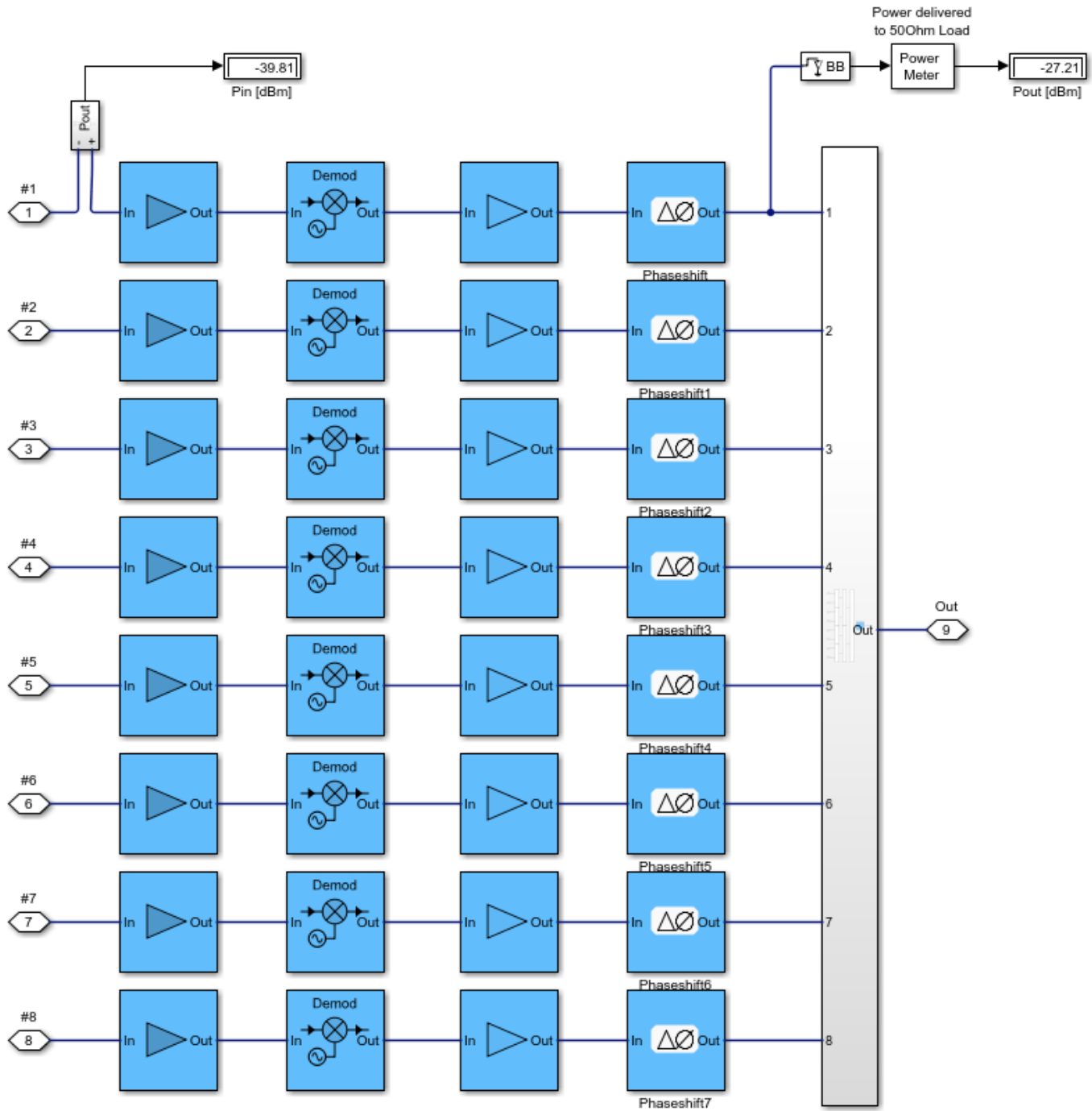
```
model = 'simrfV2_RX_array';
open_system(model)
sim(model)
```



Note that the input signal is a three-dimensional array: the first dimension is used to frame data, the second dimension is used for multi-carrier signals, and the third dimension is used to provide the two polarization components.

Looking under the mask of the RF Receiver subsystem shows the structure of the multichain RF system. Each chain ends with a phase shifter, such that when the signals are combined, the array beam is aimed at the given direction of arrival. The signals are combined using inverted Wilkinson power dividers.

```
open_system([model '/RF Receiver'], 'force');
```



The power delivered at the input (Pin) and the output (Pout) of the first chain are measured in the model and correspond approximately to the expected values. Pout is close to the value anticipated by the analysis computed with the RF Budget Analyzer app as shown above. Pin is close to the delivered power calculated using the antenna impedance matching efficiency, η_z , as follows:

$$\eta_z = 10 \cdot \log_{10} \left(1 - \frac{\text{abs}((Z_{in} - Z_{Ant_RX}'))}{(Z_{in} + Z_{Ant_RX})} \right)^2;$$

$$Pin_{RX} = P_{av} + \eta_z$$

```
Pin_RX =
    -39.2521
```

The difference between the simulation results and the expected values computed with the budget analysis are due the approximation of the gain of the antenna element in the single RX chain as the gain of the antenna array divided by 8. This approximation ignores differences between the power received by different antenna elements in a finite array.

Close the RX model and proceed to model the TX.

```
bdclose(model)
```

MIMO Transmitter System

Design a MIMO transmitter (TX) system starting with the budget analysis of a single antenna RF chain. For MIMO transmitter system, assume an input power of -7.41 dBm and the same center frequency as the receiver.

```
TX_Pin = -7.41; % Transmitter input power (dBm)
DOD = 180; % Direction of departure (deg)
```

Design the TX antenna to be identical to the RX antenna. The array orientation is such that the direction of departure is normal to the array axis, and flipped by 180 degrees compared to the RX antenna. While this rotation does not play an important role for the current array due to symmetry along the z-axis, it might be important for other types of antennas.

```
arrayTXObj = design(linearArray, CF, dipole);
arrayTXObj.NumElements = 8;
arrayTXObj.TiltAxis = [0 0 1];
arrayTXObj.Tilt = 180;
```

The TX array is located on the same elevation plane as the RX antenna, and the direction of departure is along the array normal.

```
Az_TX = 90-DOD; % Azimuth direction of departure
El_TX = 0; % Elevation of direction of departure
```

Calculate the TX antenna array gain using full-wave analysis.

```
GAntTX = pattern(arrayTXObj,CF,Az_TX,El_TX, 'Type','gain');
```

The last stage of the TX before the antenna is a power amplifier with input and output impedance equal to 50 Ohm. Calculate the antenna impedance of the first chain of the transmitter.

```
Zout = 50;
sp = sparameters(arrayTXObj,CF);
gammaInAnt1 = snp2smp(sp.Parameters,50,1,Zout);
ZAnt_TX = gamma2z(gammaInAnt1);
```

If the TX and RX are not perfectly aligned on the same array normal ($DOD \approx 0$), the 8 transmitted signals have different phases. To make sure that the transmitter steer the beam towards the receiver, phase shifts are used. Use the phase shift beamformer object from Phased Array System Toolbox to calculate the phase shifts needed for aligning the array beam with direction of arrival of the received signal.

```

Beam_Az_TX = DOD; % Beamforming angle (deg)
beamformer = phased.PhaseShiftBeamformer(...
    'SensorArray',phased.ULA('NumElements',8,...
    'ElementSpacing', lambda/2), ...
    'OperatingFrequency',CF,...
    'Direction',[Beam_Az_TX; 0],...
    'WeightsOutputPort',true);
[~, phase_shifts_TX] = beamformer(ones(1,8));
phase_shifts_TX = angle(phase_shifts_TX)'/pi*180;

```

Define the gain and third order non-linearity of the power amplifier. Add a fixed gain to each element in the TX antenna array, and define the third-order output intercept point in dBm.

```

G_TX = 18.6*ones(1,8); % dB
oIP3_TX = 30; % 3rd order output intercept point (dBm)

```

Build a cascade (row vector) of RF transmitter elements:

- Phase shifters for beamforming
- IF Modulator stage defined by gain and noise figure
- Power amplifier defined by gain and OIP3
- Antenna defined by gain and impedance

```

% Phase shifter
elementsTX(1) = phaseshift( ...
    'PhaseShift',phase_shifts_TX(1));
% Modulator
elementsTX(2) = modulator( ...
    'Name','Modulator', ...
    'Gain',-3, ...
    'NF',8, ...
    'LO',CF-100e6, ...
    'ConverterType','Up');
% Power amplifier
elementsTX(3) = amplifier( ...
    'Gain',G_TX(1), ...
    'OIP3',oIP3_TX);
% Antenna
elementsTX(4) = rfantenna( ...
    'Type','Transmitter', ...
    'Gain',GAntTX, ...
    'Z',ZAnt_TX);

```

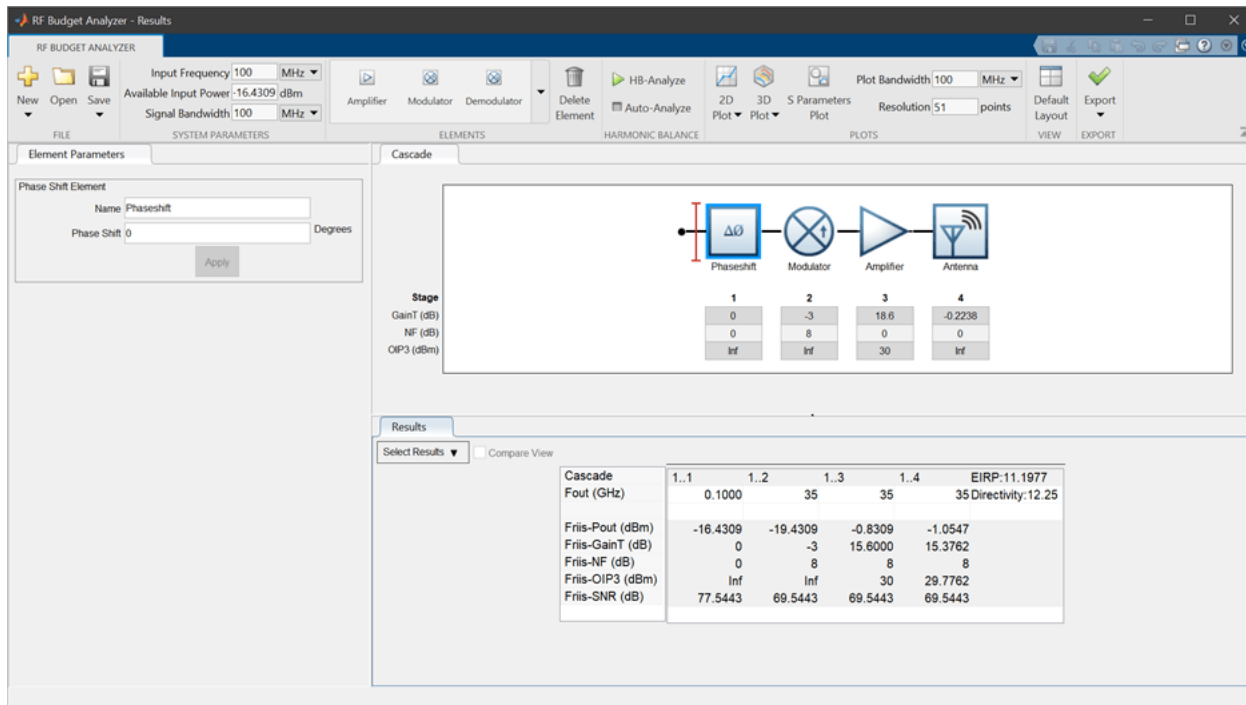
Construct the TX rfbudget object:

```

b = rfbudget( ...
    'Elements',elementsTX, ...
    'InputFrequency',100e6, ...
    'AvailableInputPower', TX_Pin - 10*log10(8), ...
    'SignalBandwidth',100e6, ...
    'Solver','Friis');

```

Type `show(b)` command at the command line to visualize the TX chain in the **RF Budget Analyzer** app.



Note that the available input power is the input to the transmitter divided by 8, due to the 8-way splitter in front of the eight chains. Also, the antenna element in the budget is approximated as having the gain of the array. This assumption allows adding the EIRP values from each chain to obtain the total EIRP of the system:

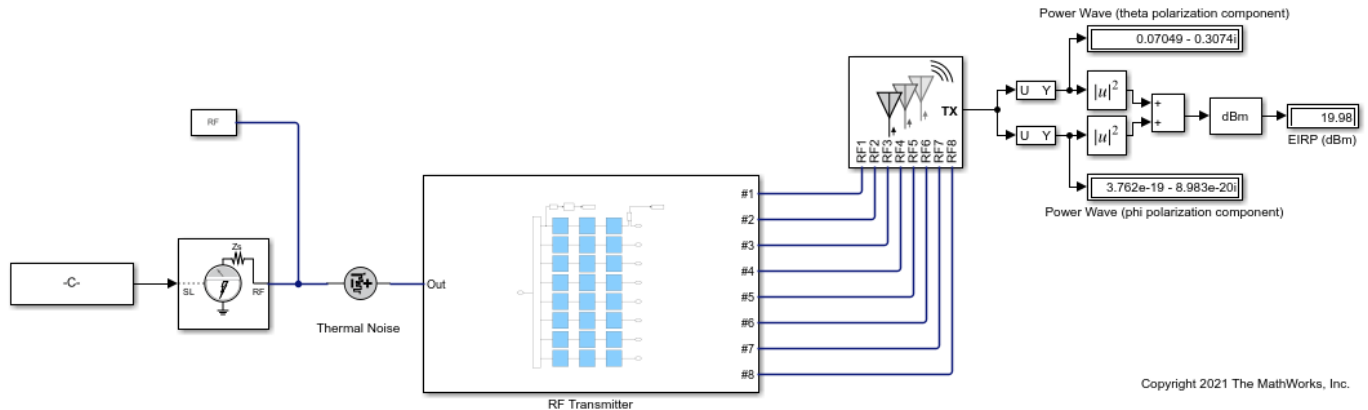
$$TX_EIRP = b.EIRP + 10 \cdot \log_{10}(8)$$

$$TX_EIRP = 20.2147$$

Create RF Blockset Model for Transmitting System

Similarly to the receiving system, the above TX cascade can be exported as an RF Blockset model and copied to create an eight-chain RF system, with the 8 individual antennas replaced by a single antenna array. The output of the Antenna block is the transmitted signal, TX, which is described as a power wave split onto the two $[\theta, \phi]$ polarization components and is normalized such that the total transmitted power is equal to $\|TX\|^2 = EIRP_{TX}$. You can now confirm the previous assumption that most of the transmitted (and received) power is aligned with the θ polarization component.

```
model = 'simrfV2_TX_array';
open_system(model)
sim(model)
```



The total normalized transmitted power is equal to the EIRP value of 20 dBm, as anticipated by the budget analysis.

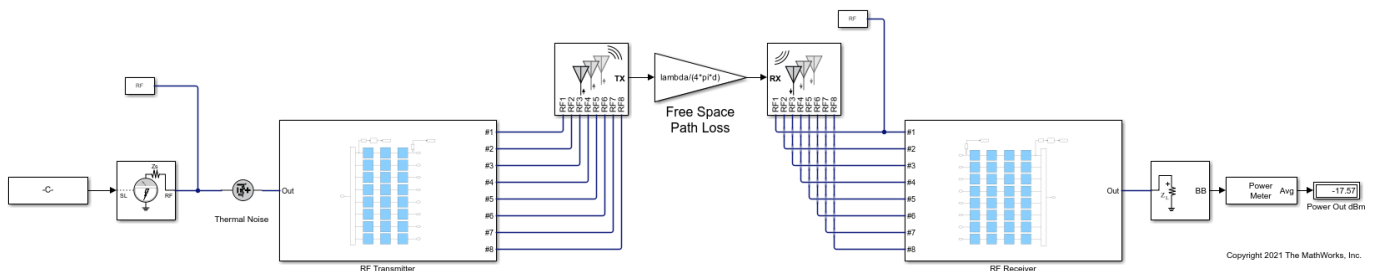
Close the TX model and proceed to combine together the TX and RX.

```
bdclose(model)
```

Combine TX and RX systems in Single Model

To account for the entire communication link behavior, you can combine the two systems above into a single model. The output of the transmitting antenna is connected to the input of the receiving antenna through a gain block representing the ideal path loss between the antennas. A more complex channel model, for example including fading effects, can be used.

```
model = 'simrfv2_TXRX_arrays';
open_system(model)
sim(model)
```



The far-field interaction between the TX and RX is captured using the signal propagating between the two arrays, and the effect of changes both in the RF systems (such as beam-steering phase shift changes, or impedance matching) and in the antennas (such as change of orientation, elements, or the entire antenna array altogether) is fully accounted for.

As an example, change the TX array while keeping the RX array as above. Specifically, rotate the transmitting antenna so that the array axis is set along the z-axis and the dipoles are oriented parallel to the x-axis. With this rotation, the TX power only radiates in the ϕ polarization, orthogonal to the polarization component of the RX antenna. This can be validated by re-designing the TX antenna array with the following commands and simulating the TX+RX model.

```

arrayTXObj = design(linearArray, CF, dipole);
arrayTXObj.NumElements = 8;
% Rotate antenna array so that array axis is set along z-axis:
arrayTXObj.TiltAxis = [0 1 0];
arrayTXObj.Tilt = 90;
% Antenna should be presolved before reusing in block
sp = sparameters(arrayTXObj,CF);
    
```

While the EIRP of the transmitter remains at a level of 20 dBm, rerunning the simulation of the full communication link shows a received power of -188.3 dBm due to the strong polarization mismatch.

Close the combined TX and RX model and proceed to perform a time domain simulation of the system.

```
bdclose(model)
```

Time Domain Simulation of Combined TX and RX System

All the above models perform static analysis (Harmonic Balance) on the RF systems. However, these models can easily be extended to simulate the time domain performance of the system. Previously, the antenna performance was calculated at a single frequency point. To capture the time domain behavior of the antennas, recalculate the antenna S-parameters over a band that encompasses the simulation band around the central frequency.

```

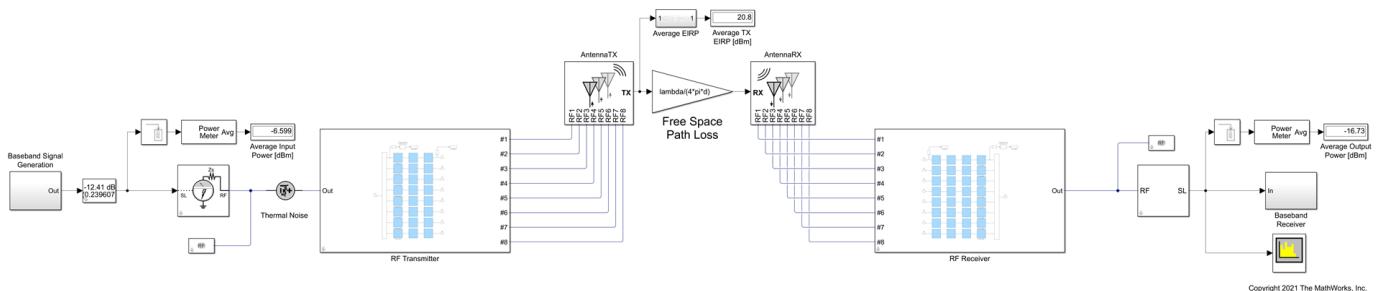
spRX = sparameters(arrayRXObj, linspace(CF-100e6, CF+100e6, 31));
spTX = sparameters(arrayTXObj, linspace(CF-100e6, CF+100e6, 31));
    
```

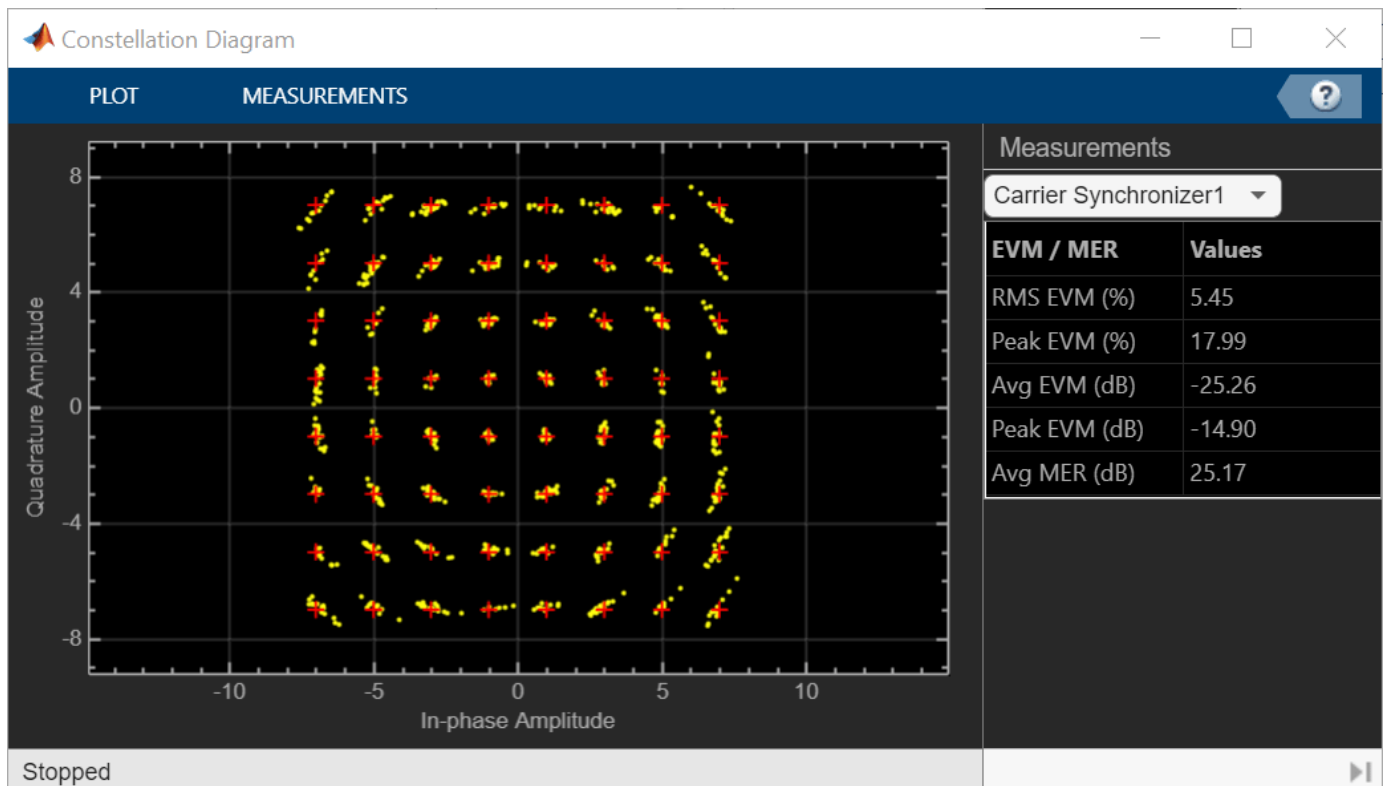
Note that the new antenna calculation results are kept within the antenna objects and used by the Antenna Blocks to estimate their temporal behavior within the simulation band.

The time domain simulation is carried out in a new model that has the same structure as the previous model. However, the signal being transmitted is now an OFDM waveform, rather than a single tone signal. In addition, the received signal coming out of the RF Receiver is now measured using a spectrum analyzer and goes into a Baseband receiver subsystem that performs baseband demodulation and computes the EVM and MER of the received OFDM waveform.

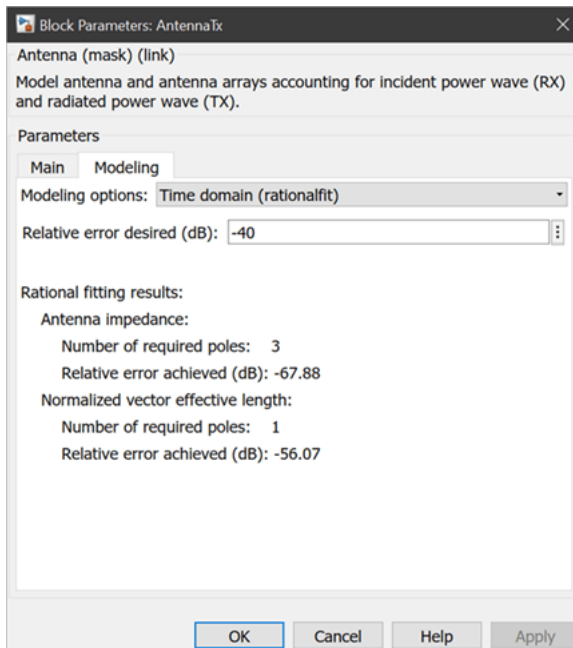
```

model = 'simrfv2_TXRX_OFDM2';
open_system(model)
sim(model)
    
```





Part of the measured EVM can be attributed to distortion due to the frequency dependent antenna impedance and pattern. The Antenna block allows control over the modeling of these quantities via parameters in the Modeling pane of the Mask Parameters dialog box of the block:



The choice of 'Time domain (rationalfit)' in the Modeling options creates an analytical rational model that approximates the antenna parameters within the entire frequency range of the simulation. The modeling choices for the Antenna block are similar to the modeling choices in other RF Blockset blocks such as the S-Parameter block. However, for the antenna block there are two separate quantities that require modeling: Antenna impedance and Normalized vector effective length. The modeling pane indicates the number of poles used and relative error achieved for each of those quantities.

```
bdclose(model)  
clear model
```

See Also

Antenna

PA and DPD Modeling for Dynamic EVM Measurement

This example shows how to use input and output modulated waveforms to extract a generalized memory polynomial model of a power amplifier (PA). You use a PA model to measure the PA dynamic error vector magnitude (EVM) using a standard-compliant 5G NR test model waveform, as defined in TS 38.141-1. You also measure the EVM in different operating conditions using digital predistortion (DPD).

The waveforms have been measured using Rohde & Schwarz instruments R&S@SMW200A and R&S@FSW. For more information, see *Linearization of RF Amplifiers: Connecting simulation and measurements on physical devices*.

This example guides you step-by-step through all the operations required to extract a PA model using measured data, verify the quality of the fitting, and simulate such model using Circuit Envelope blocks with and without DPD. Overview of this example as follows:

- Import the PA characterization data. This example uses two different input/output complex modulated waveforms measured in different conditions to extract and verify the quality of the PA model.
- Visualize the characterization data in different domains to make it suitable for fitting a memory polynomial PA model. The data might need manipulation, such as adjusting the input/output timing alignment. In this case, filter the data to reduce the characterization bandwidth.
- Identify the PA model coefficients matrix using the characterization waveform with the largest dynamic range and then verify the quality of the fitting using the second waveform. At this stage, you can experiment by changing the harmonic order and memory depth of the model.
- Define a simple RF Blockset circuit envelope testbench to verify that the extracted PA model is correctly configured for time domain simulation. This step is necessary to make sure that the simulation set up uses the desired port definition and time step.
- Define a 5G Toolbox testbench to generate a 5G standard compliant waveform and measure the EVM and then integrate the RF Blockset circuit envelope model of the PA in this testbench using the `rfsystem` workflow.
- Run the 5G EVM testbench with and without DPD algorithm in the loop and visualize results.

Read PA Input/Output Characterization Data

This data file collects the PA input and output waveforms measured using Rohde&Schwarz vector signal generator R&S@SMW200A in conjunction with the vector signal analyzer R&S@FSW. The file includes four different complex waveforms, measured at 2.6 GHz, with a sample time of 1.6276 ns.

The four waveforms provide two different sets of input/output characterization data. The first set (reference) is measured using a standard compliant 5G waveform. The second set is measured using a predistorted waveform generated by the R&S set up. This excitation waveform is created using the iterative direct DPD approach, and it provides a measure of the PA linearized performance that can be achieved in the field. In this example, we use one of these two sets of waveforms to extract the PA model, and the second to verify the quality of the fitting.

- `iq_in` — original PA input data (reference)
- `iq_out` — original PA output data (no DPD active)
- `iq_in_dpd` — predistorted PA input signal (iterative direct DPD)

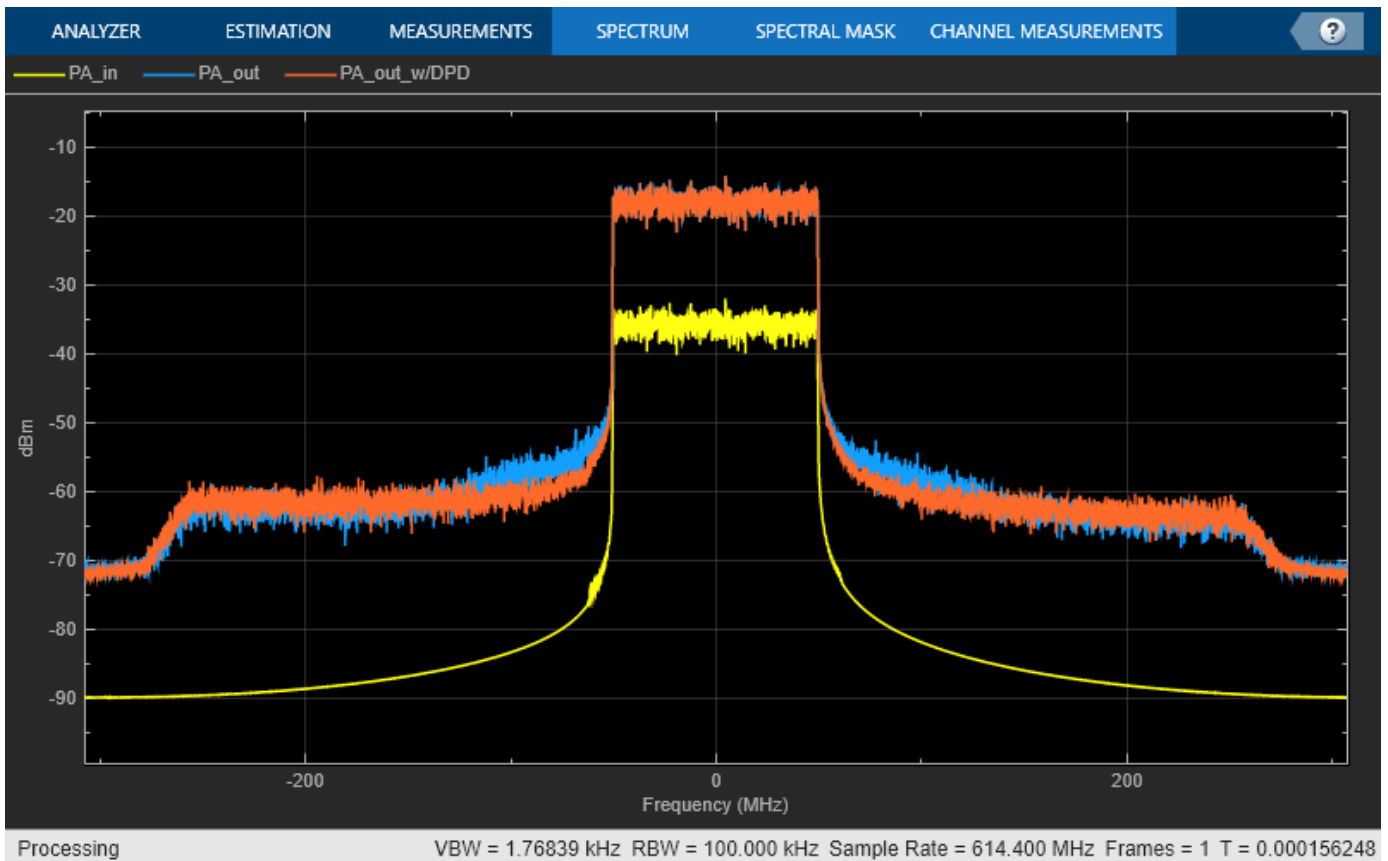
- `iq_out_dpd` — PA output signal using pre-distorted input signal

```
load('K18m_PA_Data.mat');
```

Plot Spectrum of PA Characterization Data

Plot the spectrum of the data used to characterize the PA. You can see the effects of the filtering introduced by the measurement setup at the edges of the signal.

```
SpectAnalyzer = spectrumAnalyzer;
SpectAnalyzer.SampleRate = fs;
SpectAnalyzer.ShowLegend = true;
SpectAnalyzer.SpectralAverages = 32;
SpectAnalyzer.ReferenceLoad = 50;
SpectAnalyzer.RBWSource = "Property";
SpectAnalyzer.RBW = 1e5;
SpectAnalyzer.OverlapPercent = 50;
SpectAnalyzer([iq_in, iq_out, iq_out_dpd]);
SpectAnalyzer.ChannelNames = {'PA_in', 'PA_out', 'PA_out_w/DPD'};
```



Apply Resampling Filter to Remove Filter Effects

Apply a digital multirate FIR filter to resample the data and remove the effects of filtering introduced by the measurement setup. This step is recommended to improve the quality of the fitting of the PA model.

Set the filter parameters starting with the number of filter taps, interpolation and decimation factors, and overall length of the filter delay in number of samples.

```
filtLength = 48*2;
I = 6;
D = 8;
N = ceil(filtLength*((I/D)+1));
```

Compute the oversampling factor and sample time of the resampled waveforms.

```
ovs = I/D;
Tstep = 1/fs/ovs;
```

Create a filter object with the given parameters.

```
FIR_bw = designMultirateFIR(I,D,filtLength,'SystemObject',true);
```

Apply the filter to all four characterization waveforms.

```
inDataPA1 = FIR_bw(iq_in);
inDataPA2 = FIR_bw(iq_in_dpd);
outDataPA1 = FIR_bw(iq_out);
outDataPA2 = FIR_bw(iq_out_dpd);
```

Remove the filter transient at the beginning of the waveforms and store the data in two matrices: one for the input waveforms, and one for the output waveforms. The first column in both matrices represents the original data, and the second column represents the data measured using the iterative direct DPD approach.

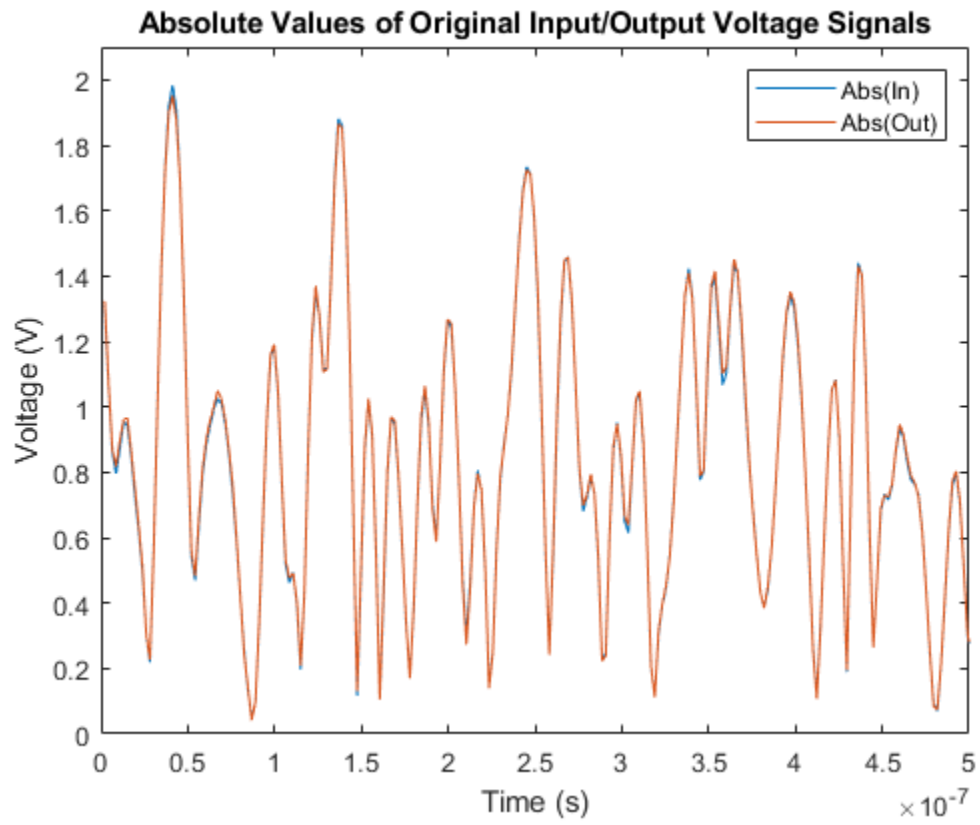
```
in(:,1) = inDataPA1(N:end);
in(:,2) = inDataPA2(N:end);
out(:,1) = outDataPA1(N:end);
out(:,2) = outDataPA2(N:end);
numDataPts = length(in(:,1));
```

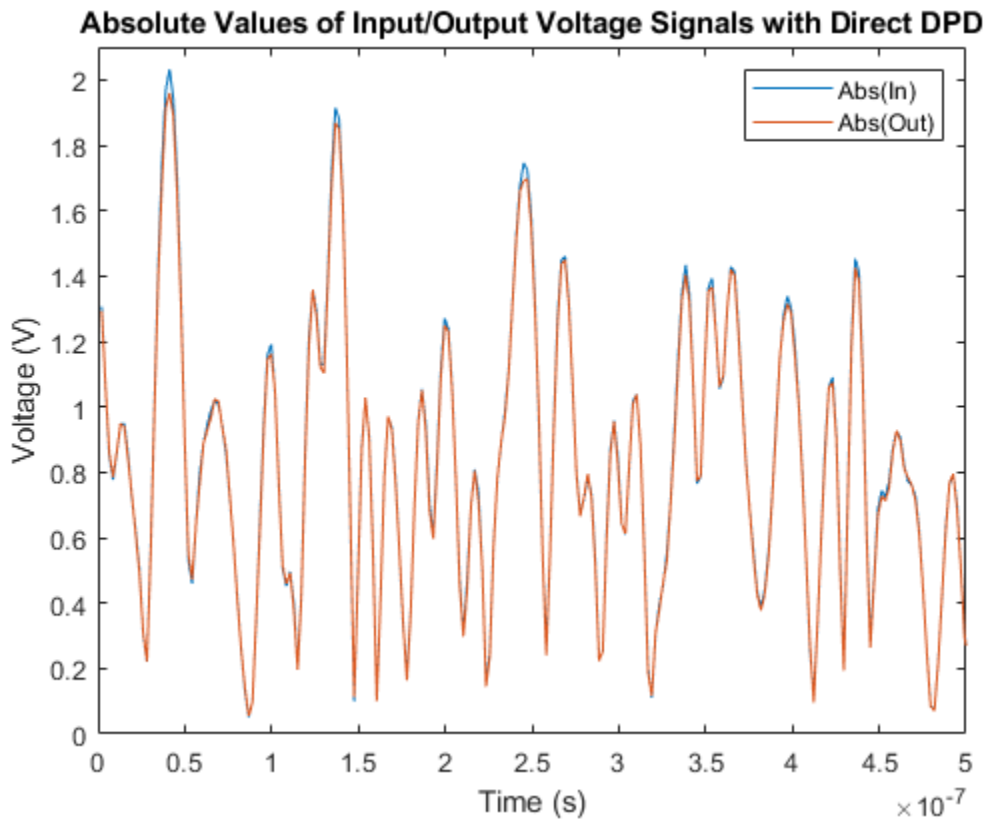
Plot PA Waveforms in Time and Frequency Domain

Visualize the input and output characterization data with and without iterative direct DPD.

In the figures below, you plot the first 500 ns of data. The data is well aligned in time, and the effects of nonlinearity are barely visible on the peaks of the output signals. For direct comparison you multiply the data times the PA gain.

```
g = 10^(g0/20);
for i = 1:2
    figure;
    plot((1:numDataPts)*Tstep, abs(in(:,i))*g, ...
         (1:numDataPts)*Tstep, abs(out(:,i)))
    legend('Abs(In)', 'Abs(Out)', 'Location', 'northeast')
    xlabel('Time (s)')
    xlim([0 0.5e-6]); ylim([0 2.1]);
    ylabel('Voltage (V)')
    if i==1
        title('Absolute Values of Original Input/Output Voltage Signals');
    else
        title('Absolute Values of Input/Output Voltage Signals with Direct DPD');
    end
end
```

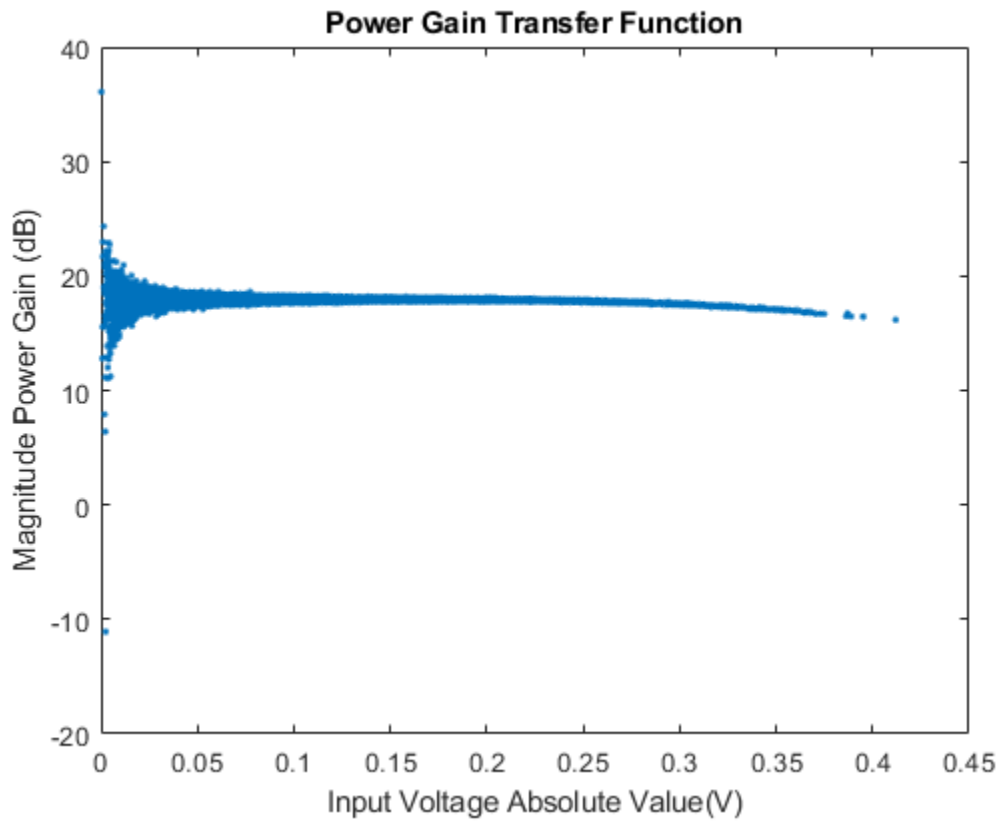


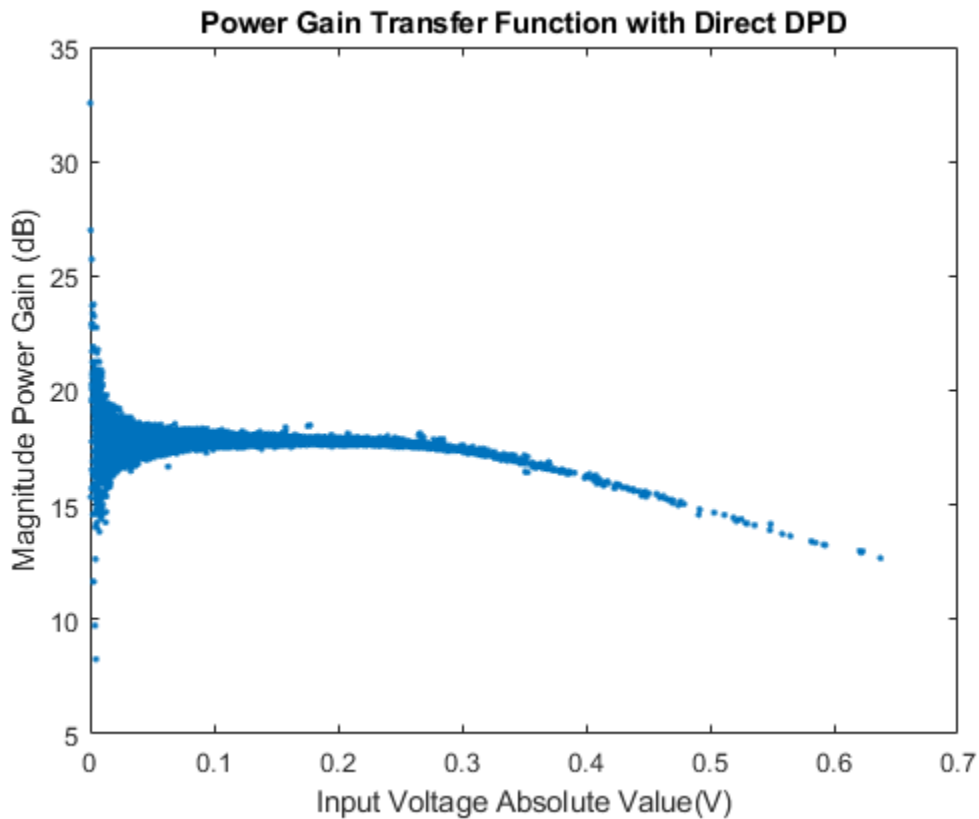
Also, plot the power gain transfer function. This plot is helpful to visualize memory effects as well as the PA nonlinearity. The power transfer function of a perfectly linear device without any memory effect is a straight line, so any deviation from it is due to nonidealities. From the comparison of the two plots, you can verify that the characterization data measured with the iterative direct DPD approach has a larger dynamic range.

```

for i = 1:2
    figure;
    TransferPA = abs(out(:,i)./in(:,i));
    plot(abs(in(:,i)),20*log10(TransferPA),'.');
    xlabel('Input Voltage Absolute Value(V)')
    ylabel('Magnitude Power Gain (dB)')
    if i==1
        title('Power Gain Transfer Function')
    else
        title('Power Gain Transfer Function with Direct DPD')
    end
end
end

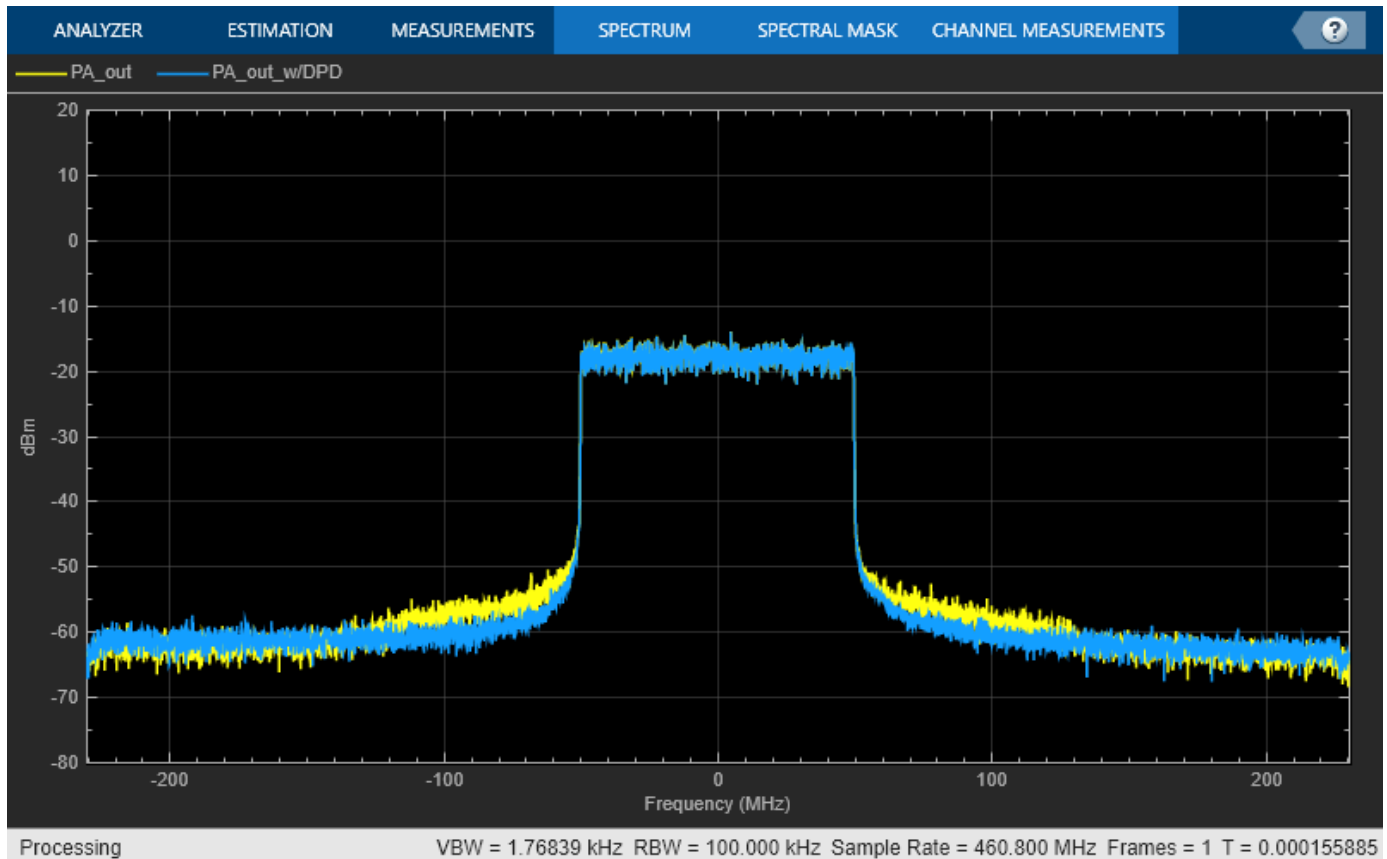
```





Plot the power spectrum of the output signal and verify that the resampling filter delivers the desired results. The characterization data can be used to identify a memory polynomial model. With the given measurement bandwidth, you can capture the spectral regrowth up to the fifth order of nonlinearity.

```
SpectAnalyzer = spectrumAnalyzer;
SpectAnalyzer.SampleRate = fs*ovs;
SpectAnalyzer.ShowLegend = true;
SpectAnalyzer.SpectralAverages = 32;
SpectAnalyzer.ReferenceLoad = 50;
SpectAnalyzer.RBWSource= "Property";
SpectAnalyzer.RBW = 1e5;
SpectAnalyzer.OverlapPercent = 50;
SpectAnalyzer(out(:,1:2));
SpectAnalyzer.ChannelNames = {'PA_out', 'PA_out_w/DPD'};
```



Determine PA Model Coefficient Matrix from Measured Input/Output Signals

Identify a generalized memory polynomial model, using a memory length of five and a degree of nonlinearity of three

```
memLen = 5;
degLen = 3;
modType = 'memPoly';
```

To compute the model coefficient matrix, use a predistorted waveform as it has a larger dynamic range. The coefficient matrix is used to verify the quality of the fitted model.

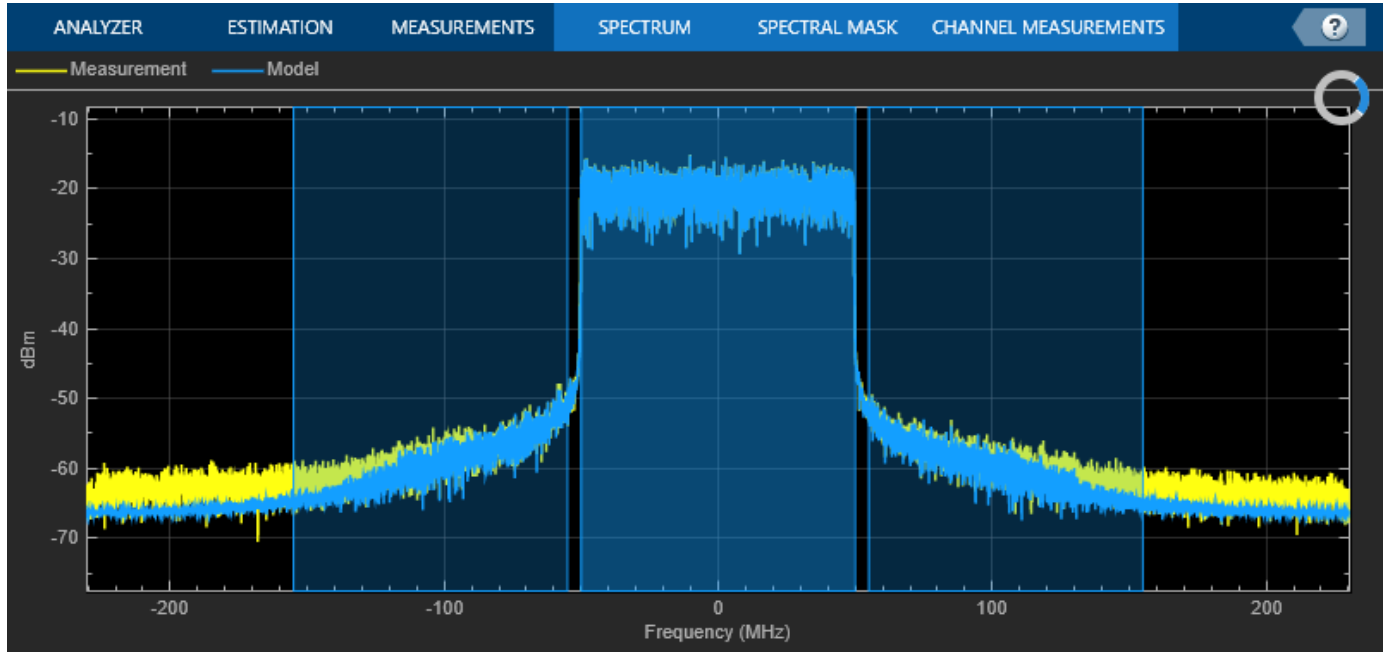
```
fitCoefMat = helperPACCharMemPolyModel('coefficientFinder', in(:,2), out(:,2), ...
    memLen,degLen,modType);
% Alternatively, you can use a subset of the data range to fit the model
% fitCoefMat = helperPACCharMemPolyModel('coefficientFinder', in(14e4:15e4,2), out(14e4:15e4,2),
%     memLen,degLen,modType);
```

Verify the quality of the PA model by computing the RMS error of the fitting, as well as plotting the fitted and measured waveforms in the time domain, the power transfer function, the spectrum, and the ACLR spectral measurement.

First, verify the quality of the fitted model by comparing the measurement and the prediction using the original data.

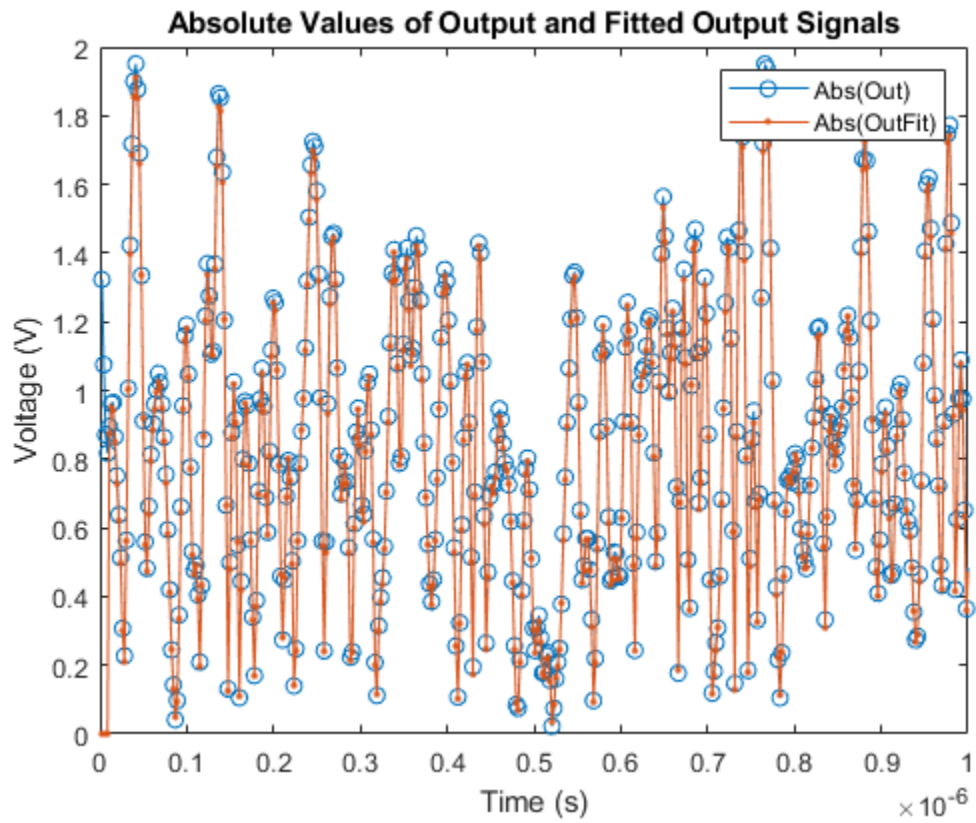
```
helperPAVerifyMemPolyModel(in(:,1), out(:,1), fitCoefMat, modType, fs*ovs);
```

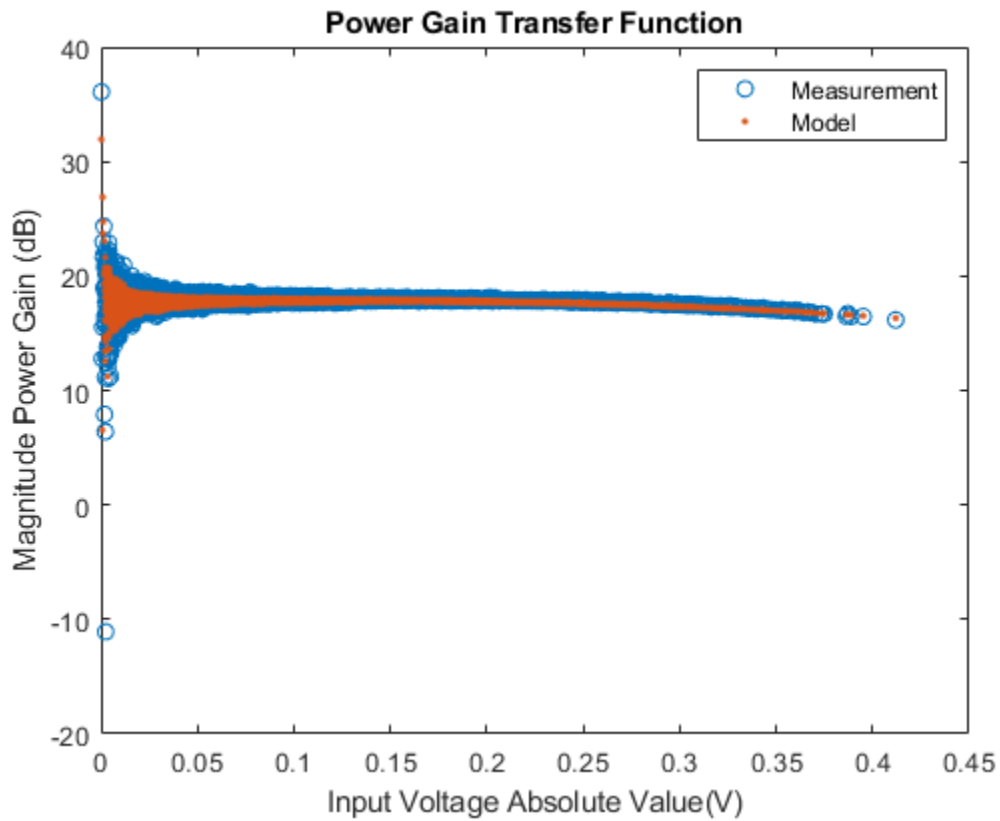
Signal standard deviation = 7.8385%
 ACPR data = -36.9763 -37.6772
 ACPR fit = -36.9763 -37.6772



ACPR

Model	ACPR1
Frequency (MHz)	105.0000
Lower (Rel Power (dBc))	-37.4842
Processing VBW = 884.194 Hz RBW = 50.0000 kHz Sample Rate = 460.800 MHz Frames = 1 T = 0.000155885	





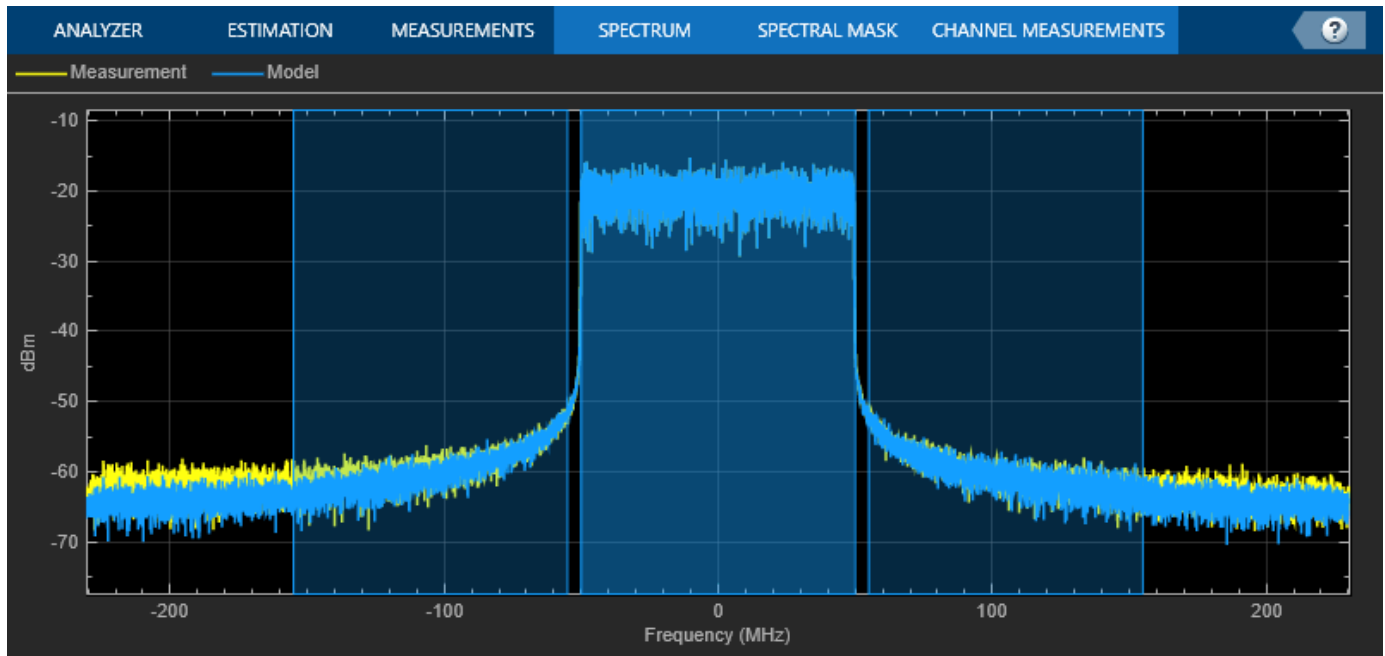
Then, verify the quality of the model using the measured data with the iterative direct DPD approach. This data has also been used to identify the PA model.

```
helperPAVerifyMemPolyModel(in(:,2), out(:,2), fitCoefMat, modType, fs*ovs);
```

```
Signal standard deviation = 3.3434%
```

```
ACPR data = -38.0929 -38.5962
```

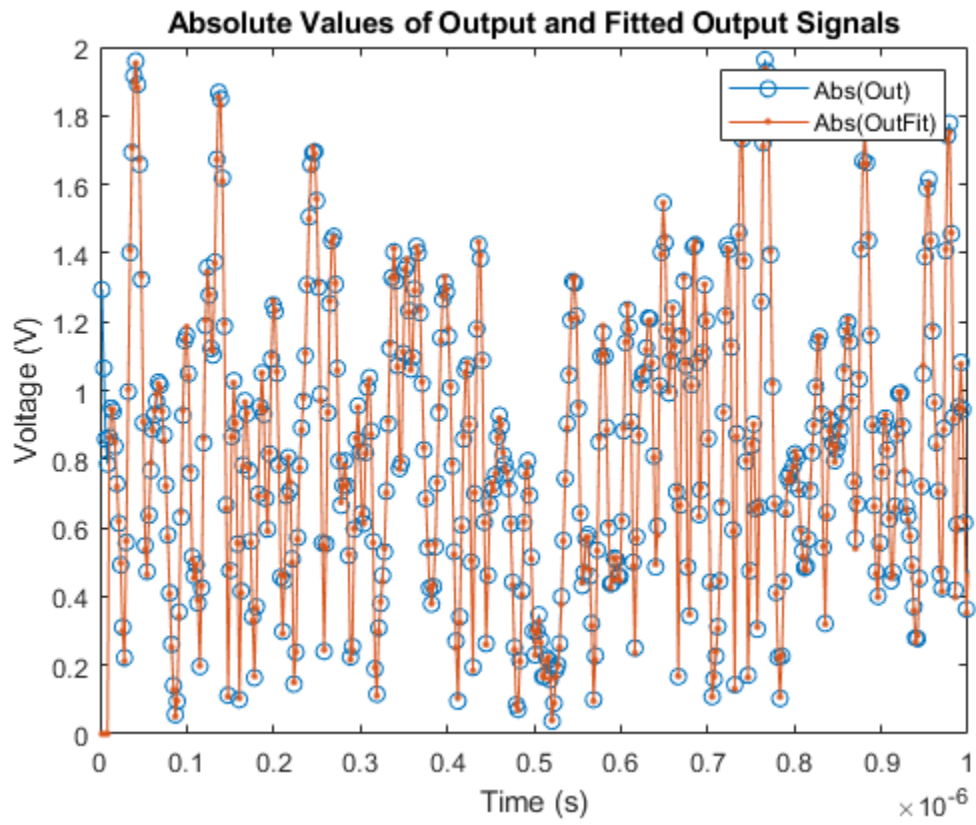
```
ACPR fit = -38.5555 -38.8375
```

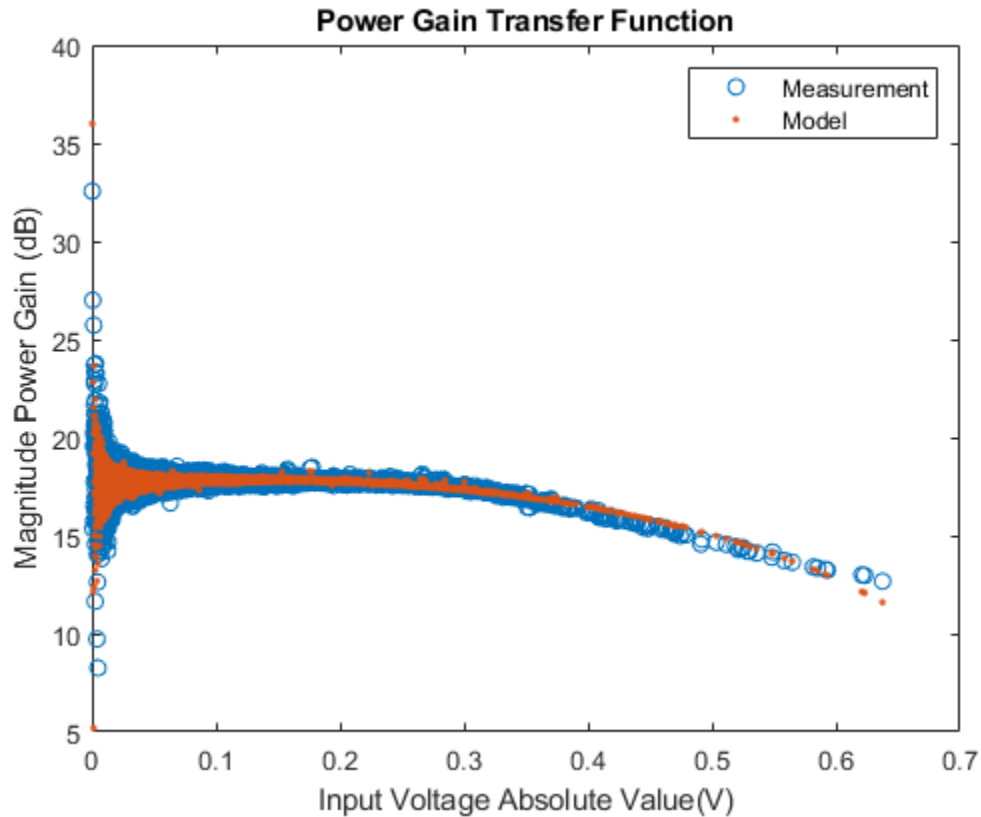



ACPR

Model	ACPR1
Frequency (MHz)	105.0000
Lower (Rel Power (dBc))	-38.5555

Processing VBW = 884.194 Hz RBW = 50.0000 kHz Sample Rate = 460.800 MHz Frames = 1 T = 0.000155885





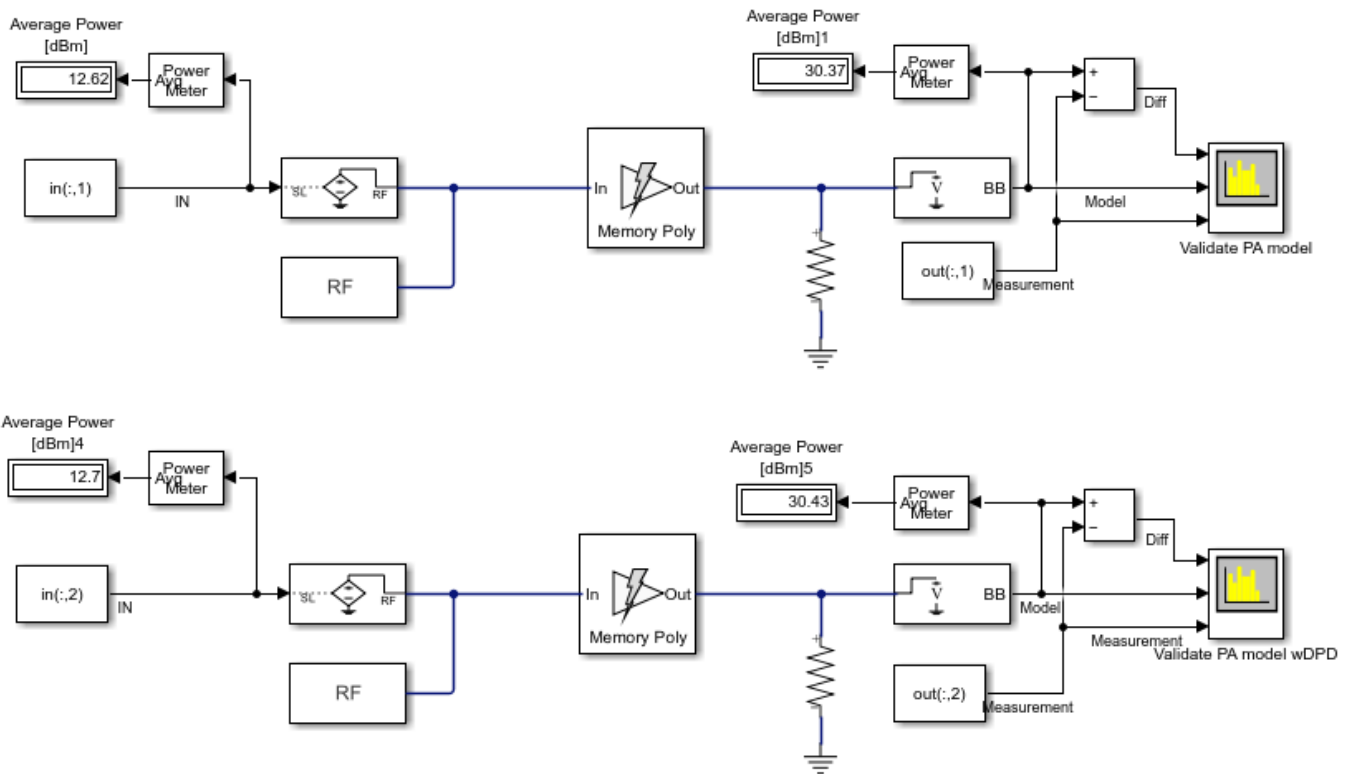
Finally, save the PA model and relevant parameters for later use.

```
close all;
save('PA_model', 'Tstep','degLen','memLen','modType','fitCoefMat','g0');
```

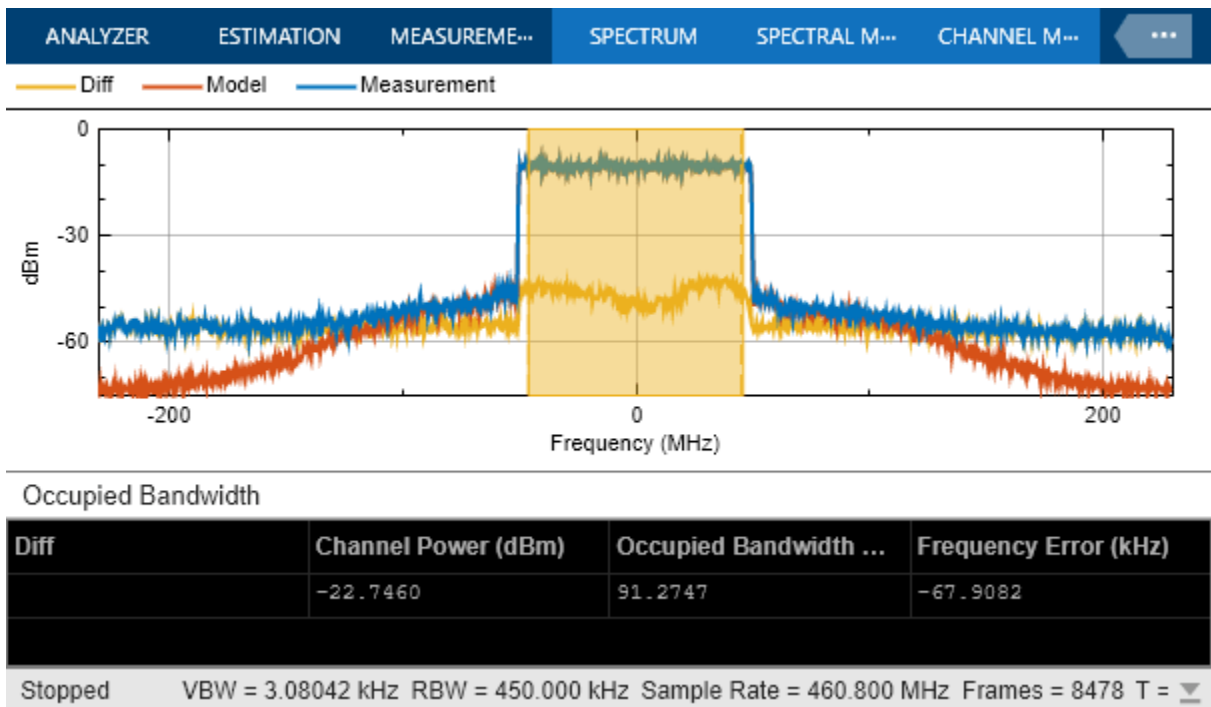
Use PA Model for Simulink Circuit Envelope Simulation

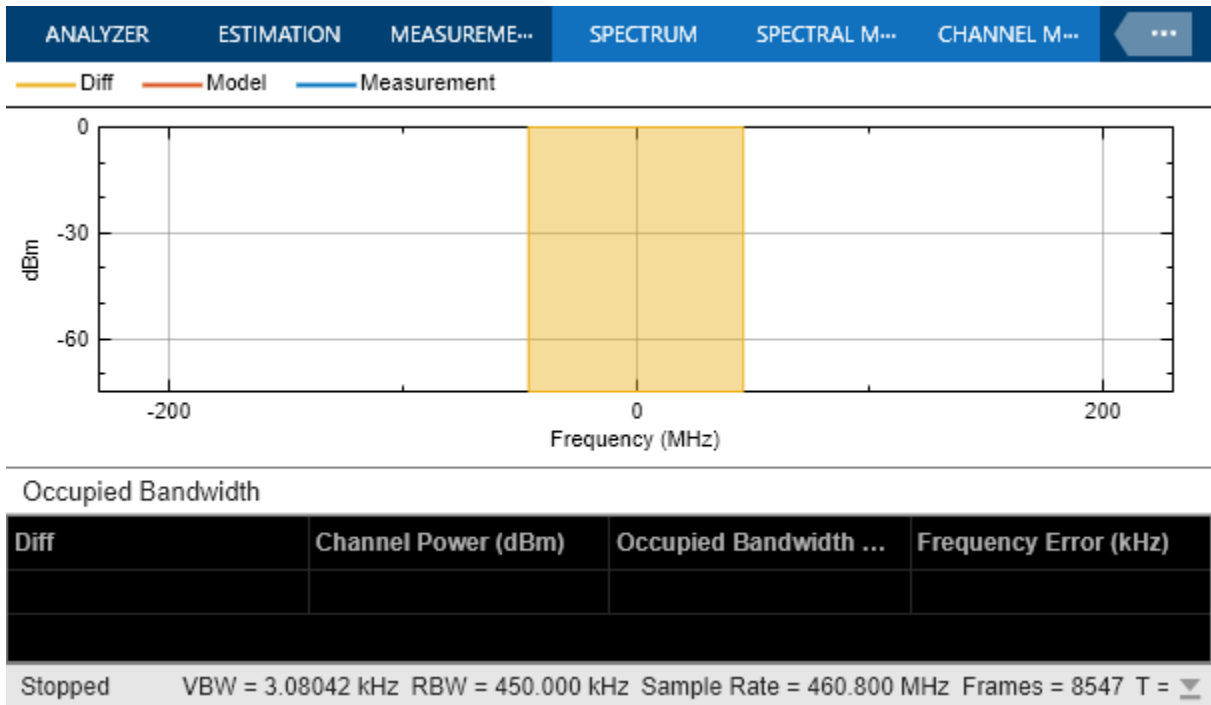
This simple Simulink testbench shows how to use the PA model for circuit envelope simulation and compare the predicted results with the characterization data. The top branch uses the PA original characterization data, while the bottom branch uses the PA data including the iterative direct DPD linearization. The spectrum analyzer directly compares the measured and predicted waveforms, and additionally shows the spectrum of the difference between the two.

```
model = 'verificationPA';
open_system(model)
sim(model,numDataPts*Tstep/3);
```



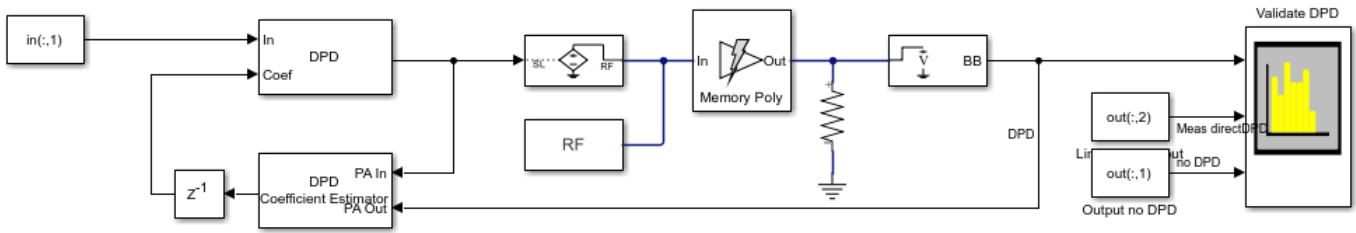
Copyright 2017-2021 The MathWorks, Inc.



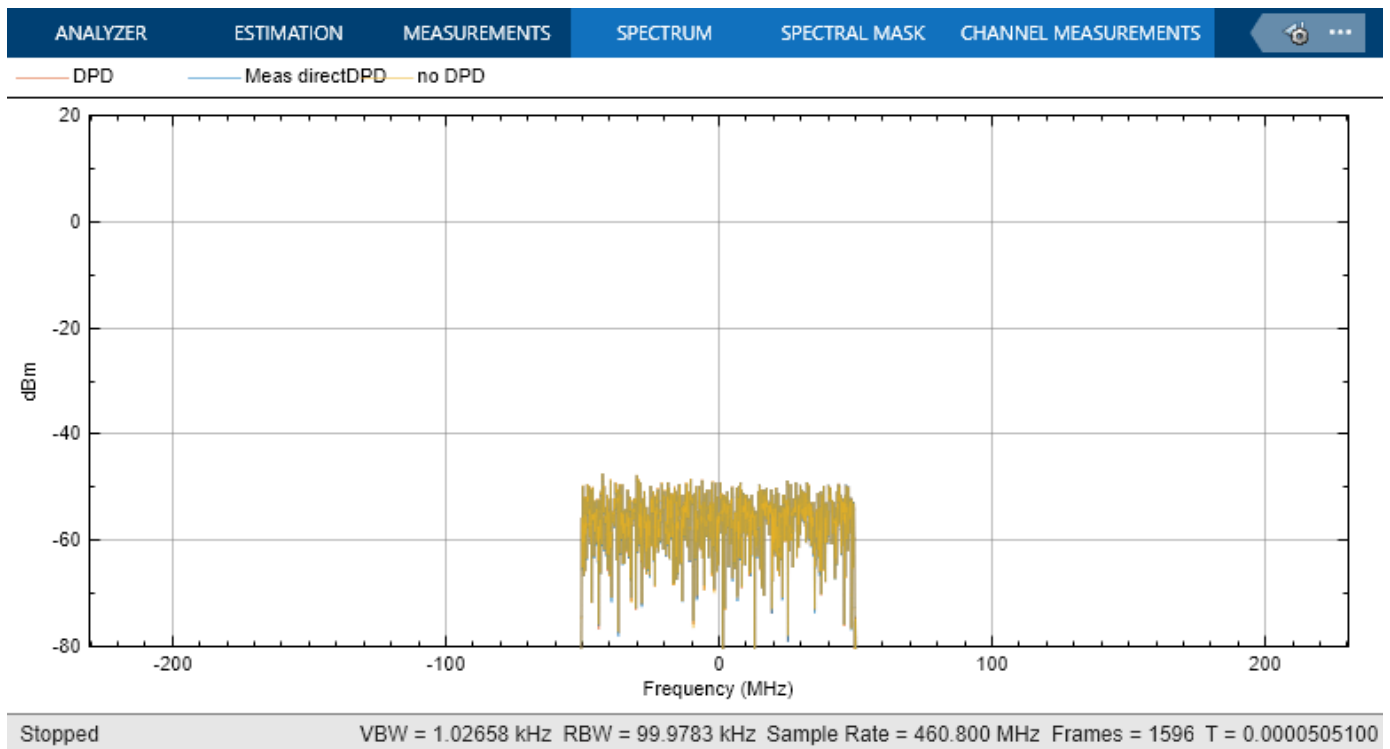


This second Simulink testbench shows how to linearize the PA model, including an adaptive digital predistortion algorithm. The output signal is compared with the measured waveforms including iterative direct DPD and the original data.

```
model = 'DPD';
open_system(model)
sim(model,numDataPts*Tstep/3);
```



Copyright 2017-2021 The MathWorks, Inc.



Setup 5G NR PDSCH Waveforms for EVM Measurement

Measure the PA 3GPP dynamic EVM with and without DPD. The EVM is measured as defined in TS 38.104, Annex B(FR1) / Annex C(FR2).

The `GenerateNRTMWaveform` script generates a standard-compliant 5G NR test model 3.1 (NR-TM3.1) waveform for frequency range 1 (FR1), as defined in TS 38.141-1. To generate a different NR test model, open the **5G Waveform Generator (5G Toolbox)** app, choose your preferred configuration and click on **Export MATLAB script**.

`GenerateNRTMWaveform`

```
% Set the simulation parameters.
targetRNTIs = [];
displayEVM = true;
plotEVM = true;
evm3GPP = true;
```



Increase the amplitude of the input signal to excite the PA nonlinearity. At this stage, make sure that the input signal is within the characterization range of the PA model.

```
txWaveform_5G = waveform*9;
disp(['Maximum input voltage = ' num2str(max(abs(txWaveform_5G))])])
```

```
Maximum input voltage = 0.51318
```

As the 5G signal has a sample rate that is different from the one used for the PA characterization, you need to interpolate the signal like before. Here you create an interpolation filter to increase the simulation bandwidth.

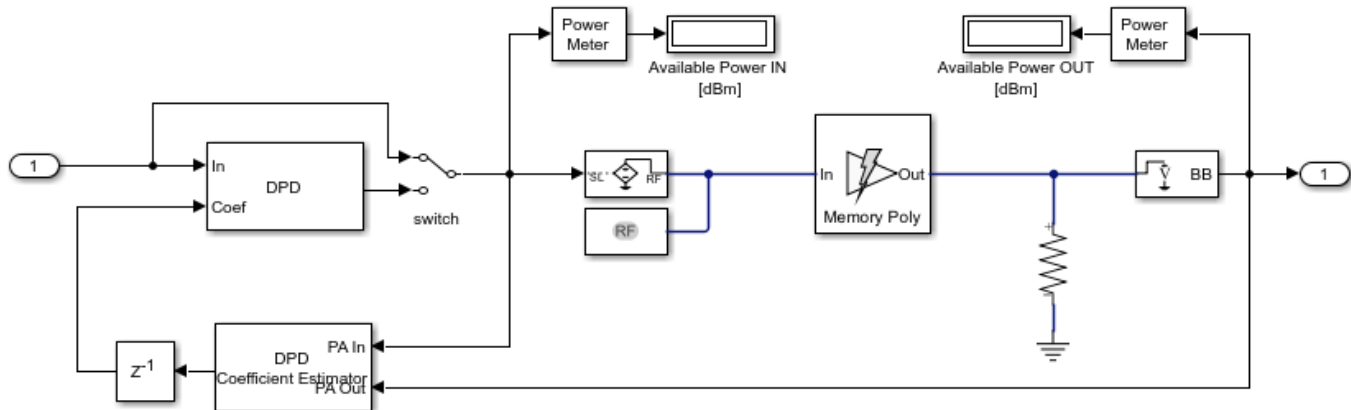
```
[I,D] = rat((1/info.ResourceGrids.Info.SampleRate)/Tstep);
filtLength = 120;
FIR_tx = designMultirateFIR(I,D,filtLength,180,'SystemObject',true);
N1 = ceil(filtLength*(I/D))+1;
```

Apply the filter to the 5G baseband waveform.

```
txWaveform_RF = FIR_tx(txWaveform_5G);
```

Load the RF system object and inspect the associated Simulink model with PA and DPD. For more information see, `rfsystem`.

```
load RF_system;
open_system(rf_dpd)
```



Copyright 2021 The MathWorks, Inc.

Simulate PA Model with 5G Waveforms

First simulate the PA model without DPD. Make sure that the manual switch is set in the up position. This simulation takes a few minutes as it is actually testing one entire frame (10 ms) of 5G data.

```
set_param('rf_dpd_model/switch','sw','1')
rxWaveform_RF = rf_dpd(txWaveform_RF);
```

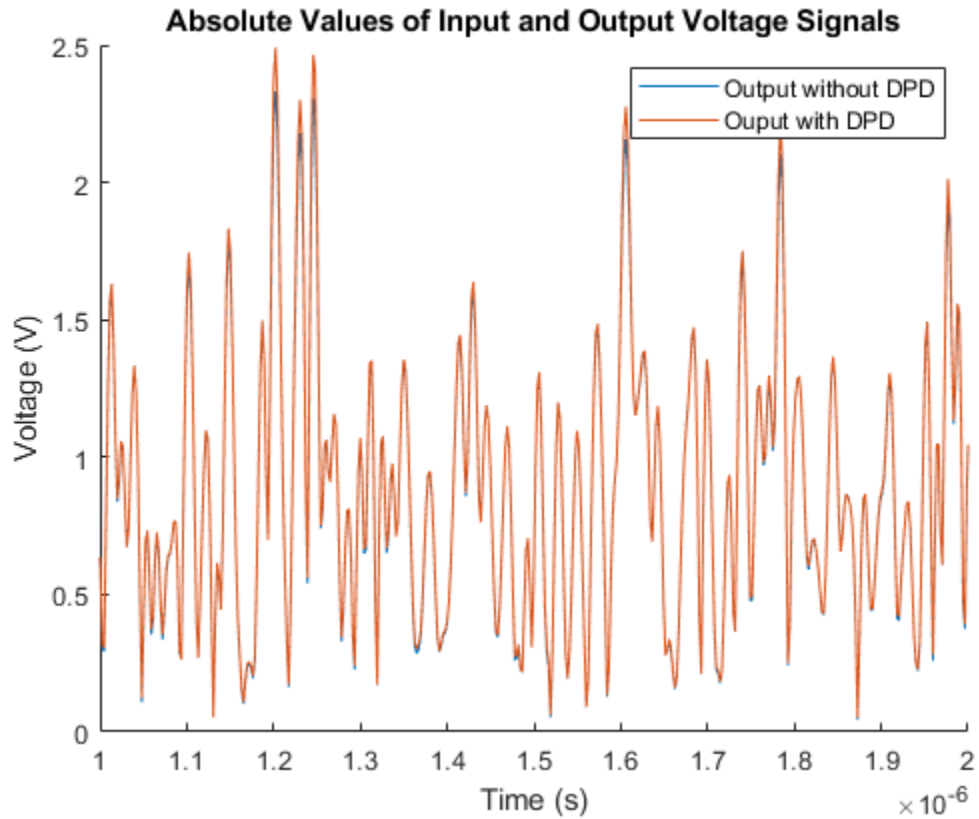
As a second step, simulate the PA including DPD by toggling the manual switch towards the down position.

```
set_param('rf_dpd_model/switch','sw','0')
rxWaveform_DPD = rf_dpd(txWaveform_RF);
```

Compare the results by visualizing the output waveforms with and without DPD.

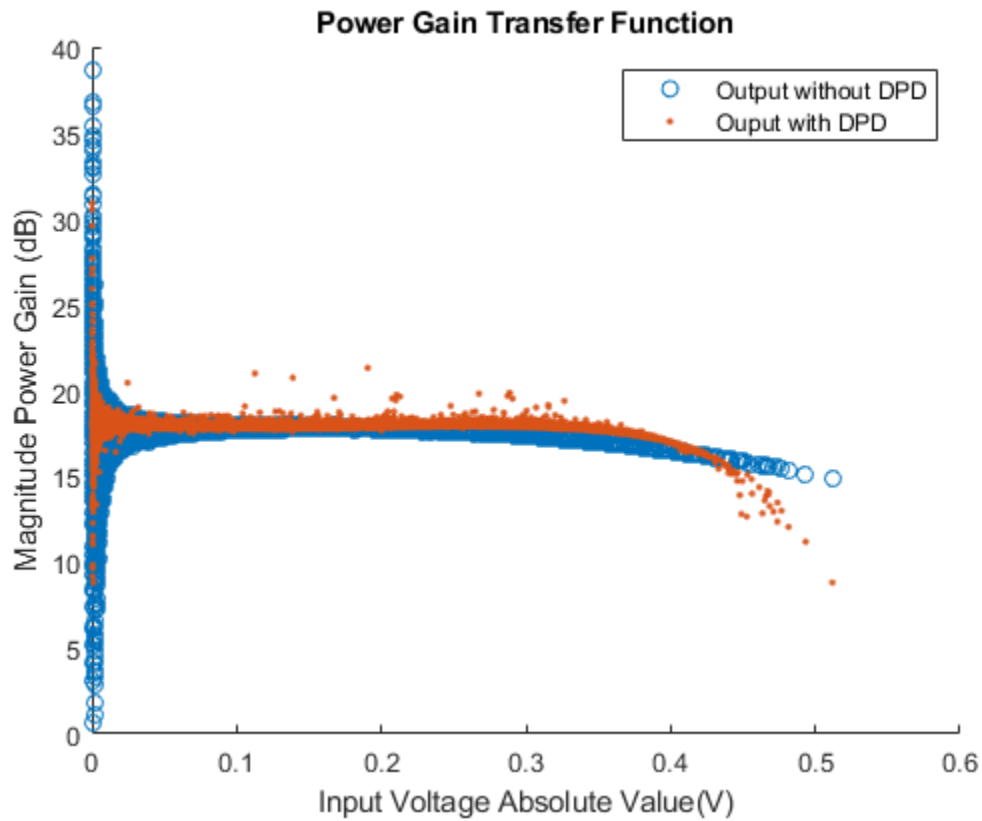
First, visualize results in the time domain.

```
figure; hold on
plot((1:length(rxWaveform_RF))*Tstep, abs(rxWaveform_RF));
plot((1:length(rxWaveform_DPD))*Tstep, abs(rxWaveform_DPD));
legend('Output without DPD','Output with DPD','Location','northeast')
xlabel('Time (s)')
xlim([1e-6 2e-6])
ylabel('Voltage (V)')
title('Absolute Values of Input and Output Voltage Signals');
```

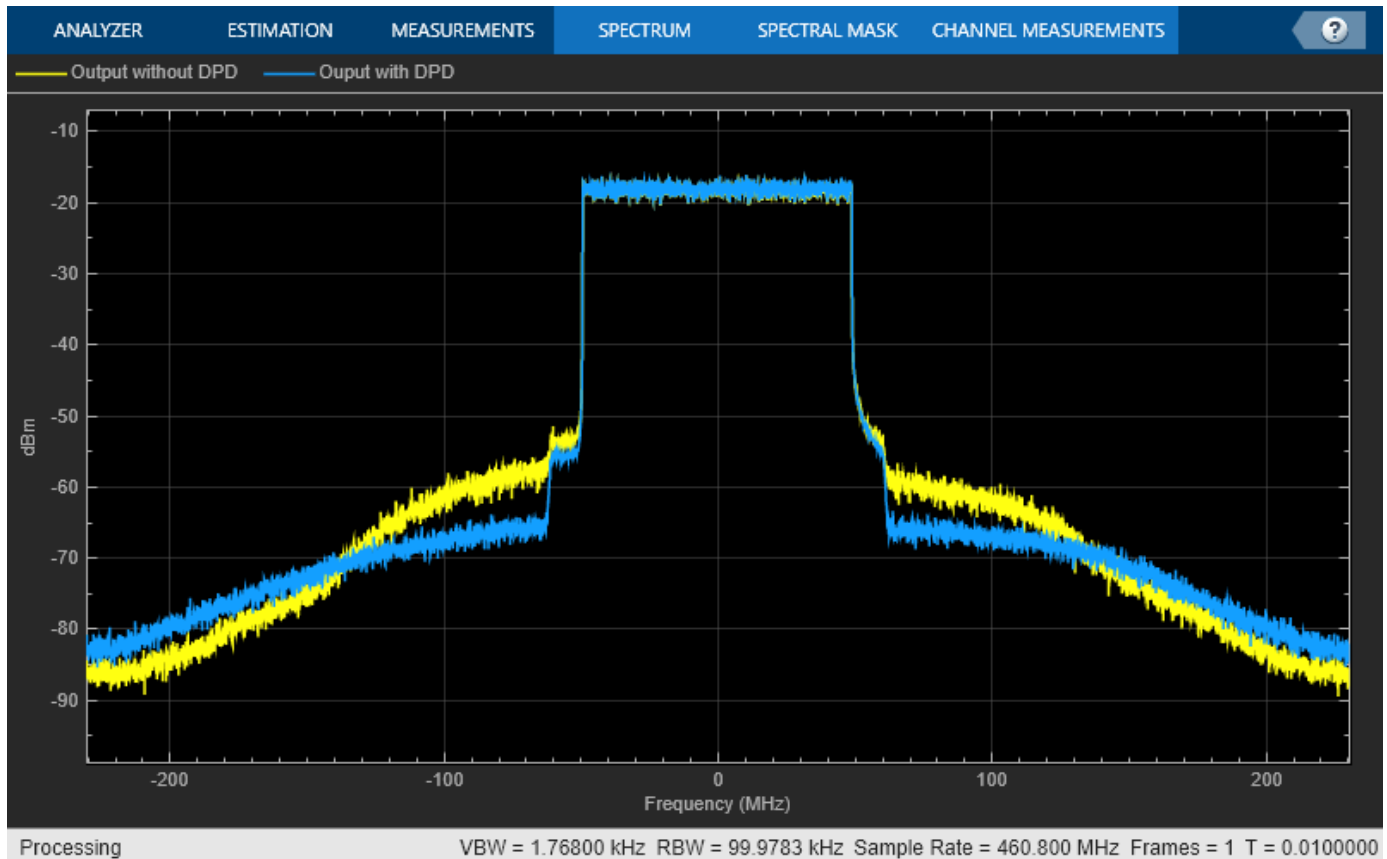
Second, visualize the power gain transfer function.

```
figure; hold on;
TransferPA = abs(rxWaveform_RF./txWaveform_RF);
plot(abs(txWaveform_RF), 20*log10(TransferPA), 'o')
TransferPA_dpd = abs(rxWaveform_DPD./txWaveform_RF);
plot(abs(txWaveform_RF), 20*log10(TransferPA_dpd), '.')
ylim([0 40])
xlabel('Input Voltage Absolute Value(V)')
ylabel('Magnitude Power Gain (dB)')
title('Power Gain Transfer Function')
legend({'Output without DPD', 'Output with DPD'})
```



Finally, plot the spectrum of the output signals.

```
SpectAnalyzer = spectrumAnalyzer;
SpectAnalyzer.SampleRate = 1/Tstep;
SpectAnalyzer.ShowLegend = true;
SpectAnalyzer.ChannelNames = {'Output without DPD', 'Ouput with DPD'};
SpectAnalyzer.RBWSource = "Property";
SpectAnalyzer.RBW = 1e5;
SpectAnalyzer.SpectralAverages = 32;
SpectAnalyzer.ReferenceLoad = 50;
SpectAnalyzer([rxWaveform_RF rxWaveform_DPD]);
```



Before computing the EVM, you need to decimate the output waveforms. Create a decimation filter to resample the signals with the original sample time.

```
FIR_rx = designMultirateFIR(D,I,filLength,180,'SystemObject',true);
N2 = ceil(filLength*(D/I))+1;
```

Apply the decimation filter to the output waveforms and remove the transient introduced by the filter.

```
M = floor(length(rxWaveform_RF)/I);
rxWaveform_5G = FIR_rx(rxWaveform_RF(1:M*I));
rxWaveform_5G = rxWaveform_5G(N2:end);
rxWaveform_5G_DPD = FIR_rx(rxWaveform_DPD(1:M*I));
rxWaveform_5G_DPD = rxWaveform_5G_DPD(N2:end);
```

Perform 3GPP Dynamic EVM Measurements

The helper function `hNRPDSCEVM`, performs these steps to decode and analyze the waveform:

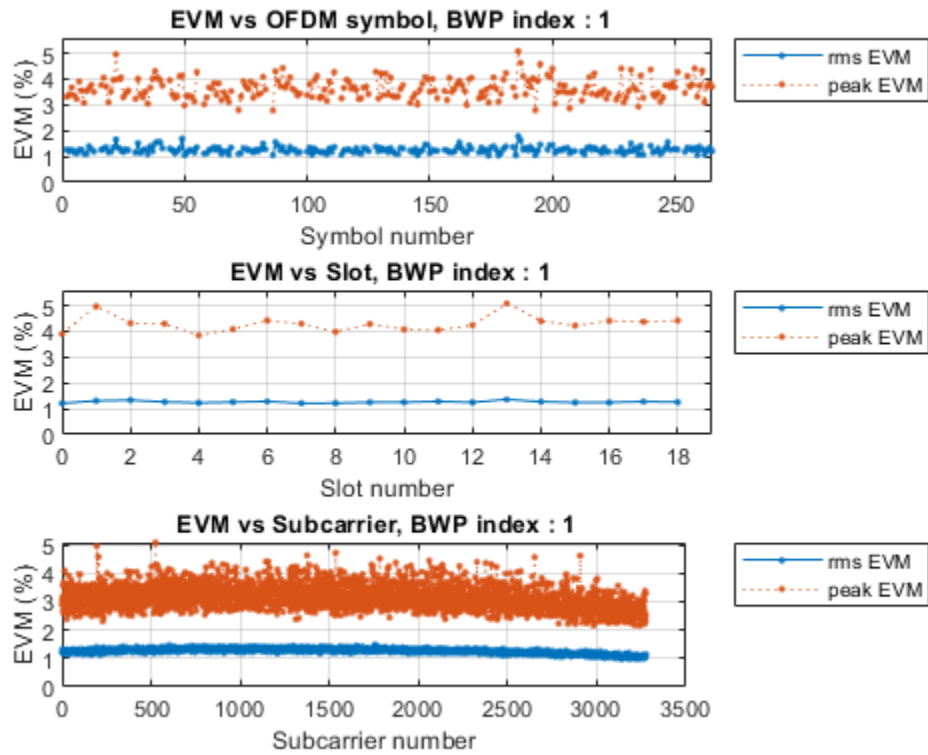
- Synchronization using the DM-RS over one frame for FDD (two frames for TDD)
- OFDM demodulation of the received waveform
- Channel estimation
- Equalization
- PDSCH EVM computation (enable the switch `evm3GPP`, to process according to the EVM measurement requirements specified in TS 38.104, Annex B(FR1) / Annex C(FR2))

The example measures and outputs various EVM related statistics (that is per symbol, per slot, and per frame peak EVM and RMS EVM). The example displays EVM for each slot and frame on the command window. It also displays the overall EVM averaged over the entire input waveform. The example produces a number of plots: EVM versus per OFDM symbol, slot, subcarrier, and overall EVM. Each plot displays the peak versus the RMS EVM.

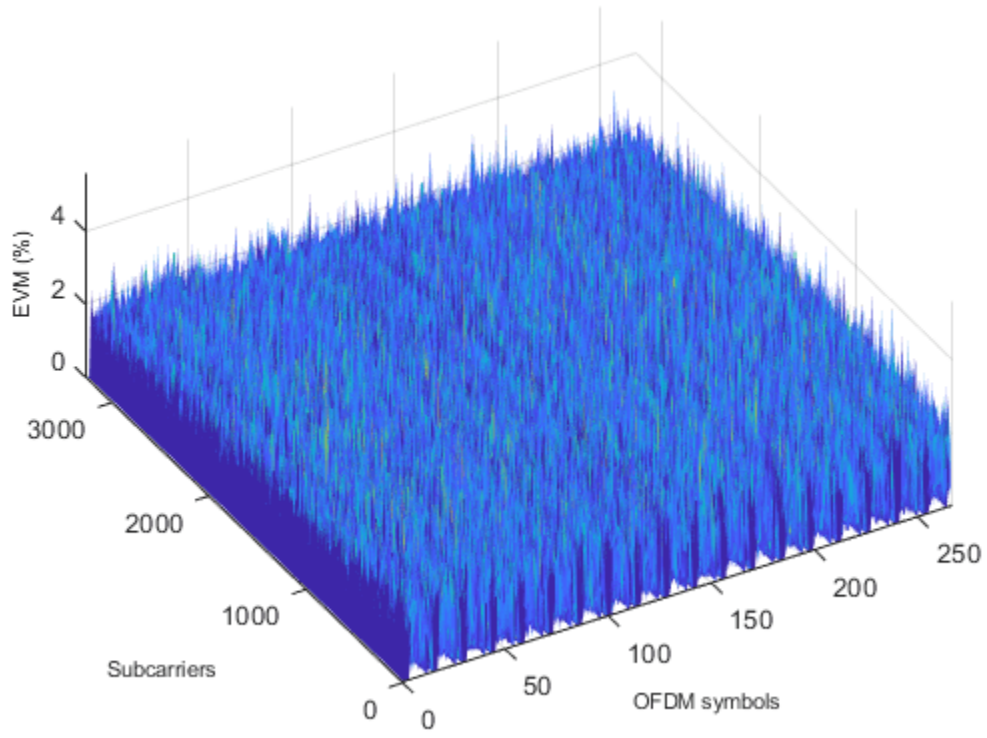
Compute and display EVM measurements for the waveform without DPD.

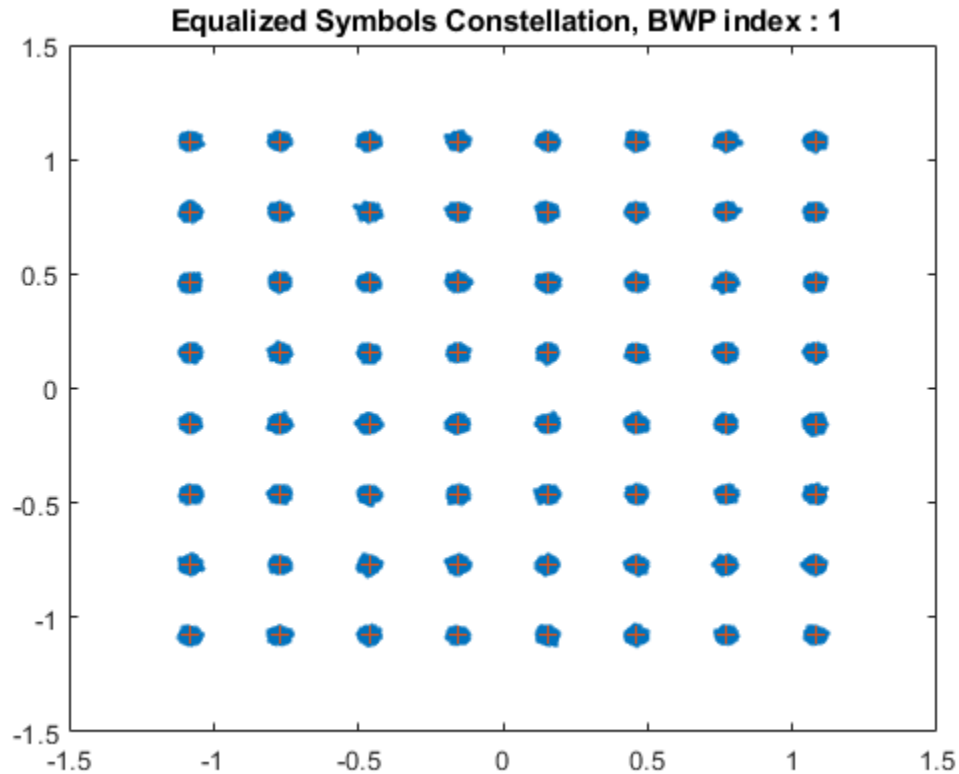
```
cfg = struct();
cfg.Evm3GPP = evm3GPP;
cfg.TargetRNTIs = targetRNTIs;
cfg.PlotEVM = plotEVM;
cfg.DisplayEVM = displayEVM;
cfg.Label = cfgDLTM.Label;
[evmGrid,eqSym,refSym] = hNRPDSCHEVM(cfgDLTM,rxWaveform_5G, cfg);
```

```
EVM stats for BWP idx : 1
Low edge RMS EVM, Peak EVM, slot 0: 1.207 3.898%
High edge RMS EVM, Peak EVM, slot 0: 1.207 3.898%
Low edge RMS EVM, Peak EVM, slot 1: 1.306 4.950%
High edge RMS EVM, Peak EVM, slot 1: 1.306 4.960%
Low edge RMS EVM, Peak EVM, slot 2: 1.333 4.309%
High edge RMS EVM, Peak EVM, slot 2: 1.333 4.302%
Low edge RMS EVM, Peak EVM, slot 3: 1.261 4.269%
High edge RMS EVM, Peak EVM, slot 3: 1.261 4.291%
Low edge RMS EVM, Peak EVM, slot 4: 1.228 3.836%
High edge RMS EVM, Peak EVM, slot 4: 1.228 3.824%
Low edge RMS EVM, Peak EVM, slot 5: 1.256 4.080%
High edge RMS EVM, Peak EVM, slot 5: 1.256 4.019%
Low edge RMS EVM, Peak EVM, slot 6: 1.280 4.414%
High edge RMS EVM, Peak EVM, slot 6: 1.279 4.413%
Low edge RMS EVM, Peak EVM, slot 7: 1.211 4.268%
High edge RMS EVM, Peak EVM, slot 7: 1.210 4.287%
Low edge RMS EVM, Peak EVM, slot 8: 1.219 3.971%
High edge RMS EVM, Peak EVM, slot 8: 1.219 3.960%
Low edge RMS EVM, Peak EVM, slot 9: 1.247 4.275%
High edge RMS EVM, Peak EVM, slot 9: 1.247 4.282%
Low edge RMS EVM, Peak EVM, slot 10: 1.246 4.072%
High edge RMS EVM, Peak EVM, slot 10: 1.246 4.027%
Low edge RMS EVM, Peak EVM, slot 11: 1.280 4.044%
High edge RMS EVM, Peak EVM, slot 11: 1.280 4.036%
Low edge RMS EVM, Peak EVM, slot 12: 1.245 4.222%
High edge RMS EVM, Peak EVM, slot 12: 1.245 4.229%
Low edge RMS EVM, Peak EVM, slot 13: 1.359 5.075%
High edge RMS EVM, Peak EVM, slot 13: 1.359 5.068%
Low edge RMS EVM, Peak EVM, slot 14: 1.272 4.397%
High edge RMS EVM, Peak EVM, slot 14: 1.272 4.384%
Low edge RMS EVM, Peak EVM, slot 15: 1.238 4.216%
High edge RMS EVM, Peak EVM, slot 15: 1.237 4.219%
Low edge RMS EVM, Peak EVM, slot 16: 1.244 4.400%
High edge RMS EVM, Peak EVM, slot 16: 1.244 4.382%
Low edge RMS EVM, Peak EVM, slot 17: 1.274 4.360%
High edge RMS EVM, Peak EVM, slot 17: 1.274 4.364%
Low edge RMS EVM, Peak EVM, slot 18: 1.253 4.406%
High edge RMS EVM, Peak EVM, slot 18: 1.253 4.411%
Averaged overall RMS EVM: 1.262%
Overall Peak EVM = 5.0746%
```



EVM Resource Grid, BWP index : 1



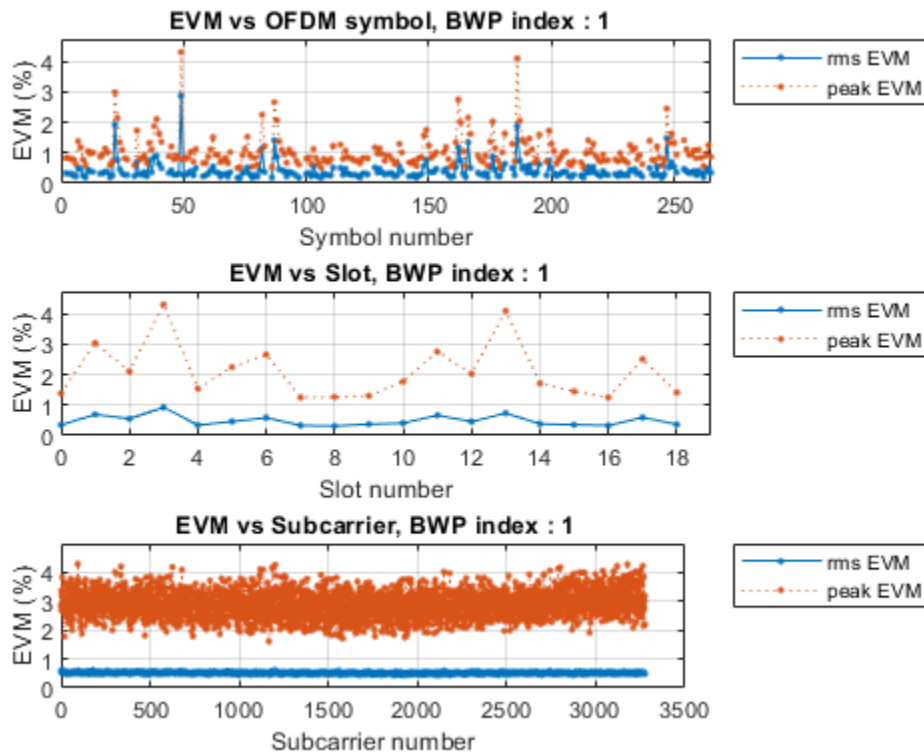


Compute and display EVM measurements for the waveform with DPD.

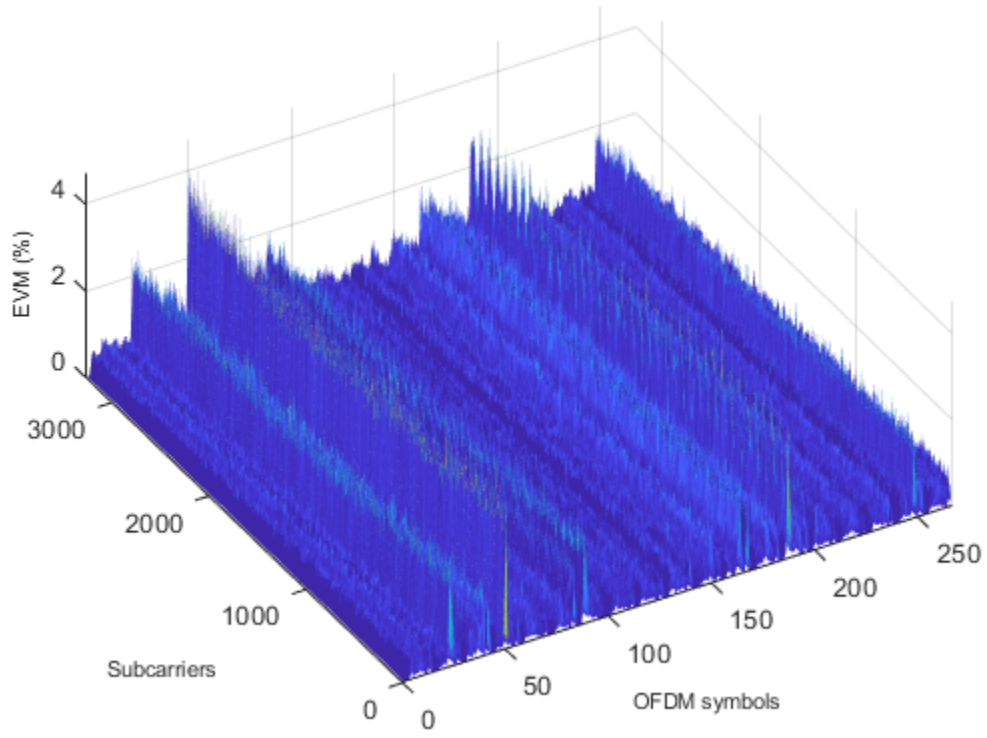
```
cfg = struct();
cfg.Evm3GPP = evm3GPP;
cfg.TargetRNTIs = targetRNTIs;
cfg.PlotEVM = plotEVM;
cfg.DisplayEVM = displayEVM;
cfg.Label = cfgDLTM.Label;
[evmGrid_DPD,eqSym_DPD,refSym_DPD] = hNRPDSCEVM(cfgDLTM,rxWaveform_5G_DPD, cfg);
```

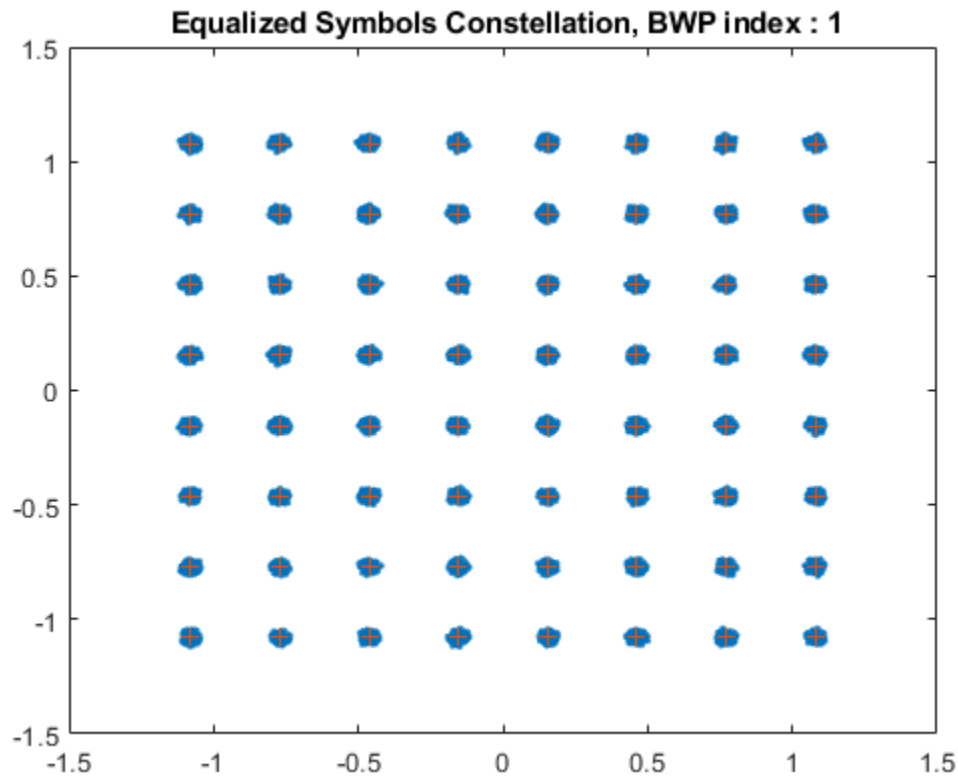
```
EVM stats for BWP idx : 1
Low edge RMS EVM, Peak EVM, slot 0: 0.349 1.383%
High edge RMS EVM, Peak EVM, slot 0: 0.349 1.365%
Low edge RMS EVM, Peak EVM, slot 1: 0.690 2.984%
High edge RMS EVM, Peak EVM, slot 1: 0.690 3.037%
Low edge RMS EVM, Peak EVM, slot 2: 0.551 2.102%
High edge RMS EVM, Peak EVM, slot 2: 0.551 2.091%
Low edge RMS EVM, Peak EVM, slot 3: 0.925 4.299%
High edge RMS EVM, Peak EVM, slot 3: 0.924 4.300%
Low edge RMS EVM, Peak EVM, slot 4: 0.339 1.510%
High edge RMS EVM, Peak EVM, slot 4: 0.337 1.536%
Low edge RMS EVM, Peak EVM, slot 5: 0.460 2.244%
High edge RMS EVM, Peak EVM, slot 5: 0.460 2.244%
Low edge RMS EVM, Peak EVM, slot 6: 0.583 2.659%
High edge RMS EVM, Peak EVM, slot 6: 0.583 2.646%
Low edge RMS EVM, Peak EVM, slot 7: 0.333 1.226%
High edge RMS EVM, Peak EVM, slot 7: 0.333 1.252%
Low edge RMS EVM, Peak EVM, slot 8: 0.312 1.261%
```

High edge RMS EVM, Peak EVM, slot 8: 0.310 1.182%
 Low edge RMS EVM, Peak EVM, slot 9: 0.371 1.286%
 High edge RMS EVM, Peak EVM, slot 9: 0.371 1.303%
 Low edge RMS EVM, Peak EVM, slot 10: 0.405 1.752%
 High edge RMS EVM, Peak EVM, slot 10: 0.405 1.768%
 Low edge RMS EVM, Peak EVM, slot 11: 0.665 2.747%
 High edge RMS EVM, Peak EVM, slot 11: 0.664 2.749%
 Low edge RMS EVM, Peak EVM, slot 12: 0.456 2.024%
 High edge RMS EVM, Peak EVM, slot 12: 0.455 2.000%
 Low edge RMS EVM, Peak EVM, slot 13: 0.733 4.093%
 High edge RMS EVM, Peak EVM, slot 13: 0.734 4.034%
 Low edge RMS EVM, Peak EVM, slot 14: 0.380 1.718%
 High edge RMS EVM, Peak EVM, slot 14: 0.379 1.718%
 Low edge RMS EVM, Peak EVM, slot 15: 0.354 1.405%
 High edge RMS EVM, Peak EVM, slot 15: 0.354 1.447%
 Low edge RMS EVM, Peak EVM, slot 16: 0.329 1.247%
 High edge RMS EVM, Peak EVM, slot 16: 0.328 1.249%
 Low edge RMS EVM, Peak EVM, slot 17: 0.591 2.447%
 High edge RMS EVM, Peak EVM, slot 17: 0.591 2.505%
 Low edge RMS EVM, Peak EVM, slot 18: 0.364 1.412%
 High edge RMS EVM, Peak EVM, slot 18: 0.363 1.369%
 Averaged overall RMS EVM: 0.512%
 Overall Peak EVM = 4.2987%



EVM Resource Grid, BWP index : 1





Helper Functions

Explore the following helper functions used by this example:

- helperPACCharMemPolyModel.m
- helperPAVerifyMemPolyModel.m
- hListTargetPDSCHs.m
- hSlotResource.m
- hChannelEstimateEVM3GPP.m
- hEVM.m
- hEVMPlots.m
- hRawEVM.m
- hNRPDSCEVM.m

References

- [1] Morgan, Dennis R., Zhengxiang Ma, Jaehyeong Kim, Michael G. Zierdt, and John Pastalan. "A Generalized Memory Polynomial Model for Digital Predistortion of Power Amplifiers." *IEEE Transactions on Signal Processing* 54, no. 10 (October 2006): 3852-60. <https://doi.org/10.1109/TSP.2006.879264>.
- [2] Gan, Li, and Emad Abd-Elrady. "Digital Predistortion of Memory Polynomial Systems Using Direct and Indirect Learning Architectures." In *Proceedings of the Eleventh IASTED International*

Conference on Signal and Image Processing (SIP) ed. F. Cruz-Roldán and N. B. Smith, No. 654-802. Calgary, AB: ACTA Press, 2009.

- [3] Lörner Markus, Florian Ramian, and Giorgia Zucchelli. "Linearization of RF amplifiers." Application note. Version 1e.09.2021. https://www.rohde-schwarz.com/us/applications/linearization-of-rf-amplifiers-application-note_56280-1110210.html.
- [4] 3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

See Also

Power Amplifier

Related Examples

- "RF Receiver Modeling for LTE Reception" on page 8-92
- "Massive MIMO Hybrid Beamforming with RF Impairments" on page 8-196
- "RF Impairments for 5G NR Downlink Waveforms" on page 8-262

RF Impairments for 5G NR Downlink Waveforms

This example shows how to generate a 5G new radio (NR) downlink waveform with full band allocation. This example also shows how to model RF impairments and compute and visualize the error vector magnitude (EVM) of the generated waveform. The generated waveform contains the physical downlink shared channel (PDSCH) channel and its associated DM-RS signal, and the physical downlink control channel (PDCCH).

Main Parameters

Use this suggested settings as a guide for setting up the parameters of the 5G Waveform as it provides information for subcarrier spacing, number of resource blocks as a function of bandwidth and subcarrier spacing as well as parameters for FR1 and FR2 carriers.

Suggested settings with transform precoding off for frequency range (FR1) (*)

BW(MHz) 5 10 15 20 25 30 40 50 60 80 90 100

NRB @15kHz 25 52 79 106 133 160 216 270

NRB @30kHz 11 24 38 51 65 78 106 133 162 217 245 273

NRB @60kHz 11 18 24 31 38 51 65 79 107 121 135

Suggested settings with transform precoding off for FR2 (*)

BW(MHz) 50 100 200 400

NRB @60kHz 66 128 264

NRB @120kHz 32 66 128 264

(*) Based on reference measurement channels from TS 38.101-1 and TS 38-101-2 Annex A2.3

Waveform and Carrier Configuration

Use the `nrDLCarrierConfig` object to configure the parameters needed for 5G NR downlink carrier waveform generation. This section sets parameters such as the subcarrier spacing (SCS), the cell ID, and the length of the generated waveform in subframes.

```

waveconfig = nrDLCarrierConfig;    % Create a downlink carrier configuration object
waveconfig.Label = 'DL carrier 1'; % Label for this downlink waveform configuration
waveconfig.NCellID = 0;           % Cell identity
waveconfig.ChannelBandwidth = 100; % Channel bandwidth (MHz)
waveconfig.FrequencyRange = 'FR2'; % 'FR1' or 'FR2'
waveconfig.NumSubframes = 1;     % Number of 1ms subframes in generated waveform
                                   % (1,2,4,8 slots per 1ms subframe, depending on SCS)

```

Define the SCS carrier using the maximum sizes for a 40 MHz NR channel. See TS 38.101-1 for more information on defined bandwidths and guard band requirements.

```

scscarrier = {nrSCSCarrierConfig};
scscarrier{1}.SubcarrierSpacing = 60;
scscarrier{1}.NSizeGrid = 128;
scscarrier{1}.NStartGrid = 0;

```

Bandwidth Parts

A BWP is formed by a set of contiguous resources sharing a numerology on a given carrier. This example can support the use of multiple BWPs using `nrWavegenBWPConfig` objects. For each BWP you can specify the subcarrier spacing (SCS), the cyclic prefix (CP) length, and the bandwidth. The `SubcarrierSpacing` parameter maps the BWP to the SCS carrier defined earlier. The `NStartBWP` parameter controls the location of the BWP in the carrier. Different BWPs can overlap each other.

```
% Set BWP configurations
bwp = {nrWavegenBWPConfig};
bwp{1}.BandwidthPartID = 1; % BWP ID
bwp{1}.Label = 'BWP 1 @ 60 kHz'; % Label for this BWP
bwp{1}.SubcarrierSpacing = scscarrier{1}.SubcarrierSpacing; % BWP subcarrier spacing
bwp{1}.CyclicPrefix = 'Normal'; % BWP cyclic prefix for 60 kHz
bwp{1}.NSizeBWP = scscarrier{1}.NSizeGrid; % Size of BWP in PRBs
bwp{1}.NStartBWP = 0; % Position of BWP, relative to point
```

CORESET and Search Space Configuration

Specify the CORESET and the PDCCH search space configuration. The CORESET and search spaces specify the possible locations (in time and frequency) of the control channel transmissions for a given numerology.

```
% Define the CORESET and search configuration
coreset = {nrCORESETConfig};
searchspace = {nrSearchSpaceConfig};
```

PDCCH Instances Configuration

Specify the set of PDCCH transmission instances in the waveform by using a cell array.

```
pdcch = {nrWavegenPDCCHConfig};
```

PDSCH Instances Configuration

This section specifies the set of PDSCH instances in the waveform. You can set the following parameters for each instance:

- Enable or disable this PDSCH sequence.
- Specify the BWP carrying the PDSCH. The PDSCH uses the SCS specified for this BWP.
- Power scaling in dB
- Enable or disable the DL-SCH transport channel coding.
- Transport block data source. You can use one of the following standard PN sequences: 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. You can specify the seed for the generator using a cell array in the form {'PN9', seed}. If no seed is specified, the generator is initialized with all ones.
- Target code rate used to calculate the transport block sizes.
- Overhead parameter
- Symbol modulation
- Number of layers
- Redundancy version (RV) sequence

```
pdsch = {nrWavegenPDSCHConfig}; % Create a PDSCH configuration object
pdsch{1}.Enable = 1; % Enable PDSCH sequence
```

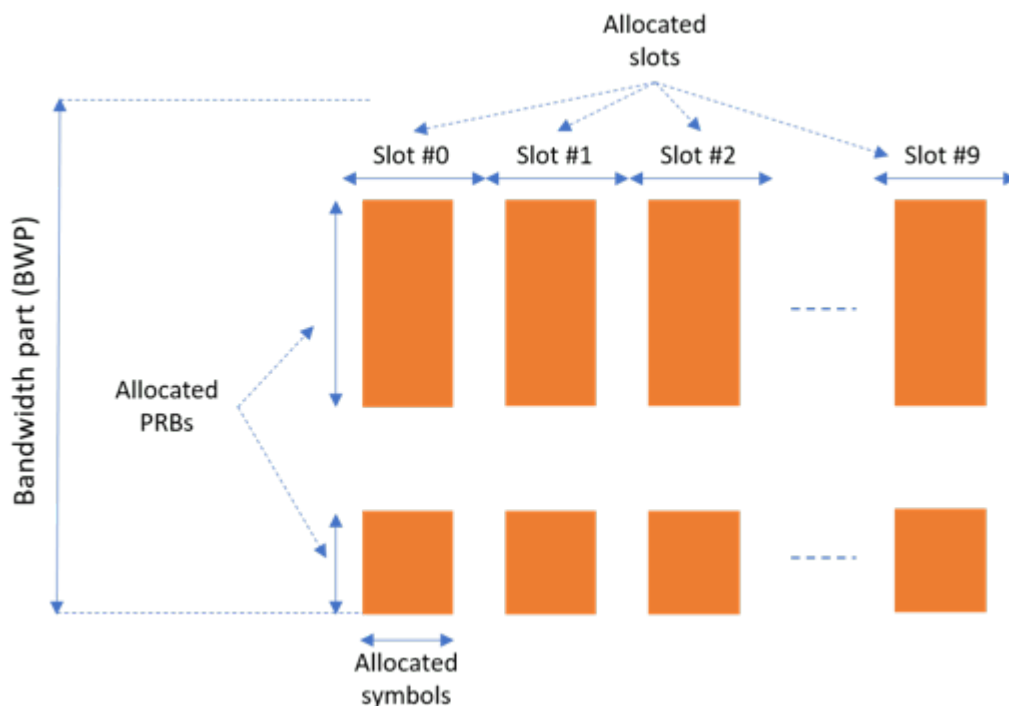
```

pdsch{1}.Label = 'PDSCH 1'; % Label for this PDSCH sequence
pdsch{1}.BandwidthPartID = 1; % Bandwidth part of PDSCH transmission
pdsch{1}.Power = 0; % Power scaling in dB
pdsch{1}.Coding = 1; % Enable the DL-SCH transport channel coding
pdsch{1}.DataSource = 'PN9'; % Channel data source
pdsch{1}.TargetCodeRate = 0.4785; % Code rate used to calculate transport block sizes
pdsch{1}.XOverhead = 0; % Rate matching overhead
pdsch{1}.Modulation = '64QAM'; % 'QPSK', '16QAM', '64QAM', '256QAM'
pdsch{1}.NumLayers = 1; % Number of PDSCH layers
pdsch{1}.RVSequence = 0; % RV sequence to be applied cyclicly across the PDSCH

```

Allocation

The following diagram represents some of the parameters used in the PDSCH allocation.



You can set the following parameters to control the PDSCH allocation. Note that these parameters are relative to the BWP.

- Symbols in a slot allocated to each PDSCH instance.
- Slots in a frame used for the sequence of PDSCH.
- Period of the allocation in slots. If this is empty it indicates no repetition.
- PRB allocation, which is relative to the BWP.
- RNTI. This value is used to link the PDSCH to an instance of the PDCCH.
- NID for scrambling the PDSCH bits.

```

pdsch{1}.SymbolAllocation = [0 14]; % First symbol and length
pdsch{1}.SlotAllocation = 0; % Allocated slot indices for PDSCH sequence
pdsch{1}.Period = 1; % Allocation period in slots
pdsch{1}.PRBSet = 0:scscarrier{1}.NSizeGrid-1; % PRB allocation

```

```
pdsch{1}.RNTI = 0; % RNTI
pdsch{1}.NID = 1; % Scrambling for data part
pdsch{1}.ReservedCORESET = 1; % Rate matching pattern, defined by CORESET IDs
```

PDSCH DM-RS Configuration

Set the DM-RS parameters.

```
% Antenna port and DM-RS configuration (TS 38.211 section 7.4.1.1)
pdsch{1}.MappingType = 'A'; % PDSCH mapping type ('A'(slot-wise),'B'(non slot-wise))
pdsch{1}.DMRSPower = 0; % Additional power boosting in dB
pdsch{1}.DMRS.DMRSConfigurationType = 2; % DM-RS configuration type (1,2)
pdsch{1}.DMRS.NumCDMGroupsWithoutData = 1; % Number of DM-RS CDM groups without data. The value (1,2)
pdsch{1}.DMRS.DMRSPortSet = []; % DM-RS antenna ports used ([] gives port numbers 0:N)
pdsch{1}.DMRS.DMRSTypeAPosition = 2; % Mapping type A only. First DM-RS symbol position (2)
pdsch{1}.DMRS.DMRSLength = 1; % Number of front-loaded DM-RS symbols (1(single symbol),2)
pdsch{1}.DMRS.DMRSAdditionalPosition = 0; % Additional DM-RS symbol positions (max range 0...3)
pdsch{1}.DMRS.NIDNSCID = 1; % Scrambling identity (0...65535)
pdsch{1}.DMRS.NSCID = 0; % Scrambling initialization (0,1)
```

Waveform Generation

Assign all the channel and signal parameters into the main carrier configuration object, `nrDLCarrierConfig`, then generate and plot the waveform.

```
waveconfig.SCSCarriers = scscarrier;
waveconfig.BandwidthParts = bwp;
waveconfig.CORESET = coreset;
waveconfig.SearchSpaces = searchspace;
waveconfig.PDCCH = pdcch;
waveconfig.PDSCH = pdsch;
waveconfig.SSBurst.Enable = 0; % Disable SS Burst
```

```
% Generate complex baseband waveform
[waveform,info] = nrWaveformGenerator(waveconfig);
```

The `nrWaveformGenerator` function returns the time domain waveform and a structure array, `info`, which contains the following information:

- The resource grid corresponding to this BWP
- The resource grid of the overall bandwidth containing the channels and signals in this BWP
- An `info` structure with information corresponding to the BWP. The contents of this `info` structure for the selected BWP are shown below.
- Information regarding the control and data channels

```
disp('Information associated to BWP 1:')
disp(info.ResourceGrids.Info)
```

```
Information associated to BWP 1:
      Nfft: 2048
      SampleRate: 122880000
CyclicPrefixLengths: [208 144 144 144 144 144 144 144 144 144 144 ... ]
      SymbolLengths: [2256 2192 2192 2192 2192 2192 2192 2192 2192 2192 ... ]
      Windowing: 0
      SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... ]
      SymbolsPerSlot: 14
      SlotsPerSubframe: 4
```

```
SlotsPerFrame: 40
k0: 0
```

Perform Filtering of Waveform to Improve ACLR

The waveform generated may have out-of-band spectral emissions owing to the implicit rectangular pulse shaping in the OFDM modulation (each OFDM subcarrier has a sinc shape in the frequency domain). In order to achieve a better adjacent channel leakage ratio (ACLR) performance, you can apply filtering to the waveform. Design a filter with a transition band that starts at the edge of the occupied transmission bandwidth and stops at the edge of the overall channel bandwidth. This filter involves no rate change: it just shapes the spectrum within the original bandwidth of the waveform. The filter is first designed, then applied to the waveform.

```
% Design filter
fir = dsp.LowpassFilter();
fir.SampleRate = info.ResourceGrids.Info.SampleRate;
BW = scscarrier{1}.SubcarrierSpacing * scscarrier{1}.NSizeGrid * 12 * 1e3;
fir.PassbandFrequency = BW/2;
fir.StopbandFrequency = (BW/2)*1.03;
fir.StopbandAttenuation = 80; % dB

% Apply filter
waveform = [waveform; zeros(200,size(waveform,2))];
txWaveform = step(fir,waveform);
infoFIR = cost(fir);

% Account for filter delay
filDelay = floor(infoFIR.NumCoefficients/2);
txWaveform = [txWaveform(filDelay+1:end,:); zeros(filDelay,size(txWaveform,2))];

% Generate timeseries 5G Signal
OSR = 4;
waveformUp = resample(txWaveform,OSR,1);
```

Spectral Response of Generated Waveform

Plot the spectral response of generated waveform.

```
Title = sprintf('5G NR %s waveform - SCS = %dkHz - BW = %dMHz', ...
    waveconfig.FrequencyRange,...
    scscarrier{1}.SubcarrierSpacing,waveconfig.ChannelBandwidth);
sp = spectrumAnalyzer('SampleRate', fir.SampleRate*OSR,...
    'ViewType','Spectrum','SpectralAverages',100,...
    'ShowLegend',true,'ChannelNames',{'original' 'filtered'},...
    'YLimits',[-100 0], 'Title',Title,'Method','welch','AveragingMethod','exponential');
sp(waveformUp);
```




Create RF Transmitter with Impairments

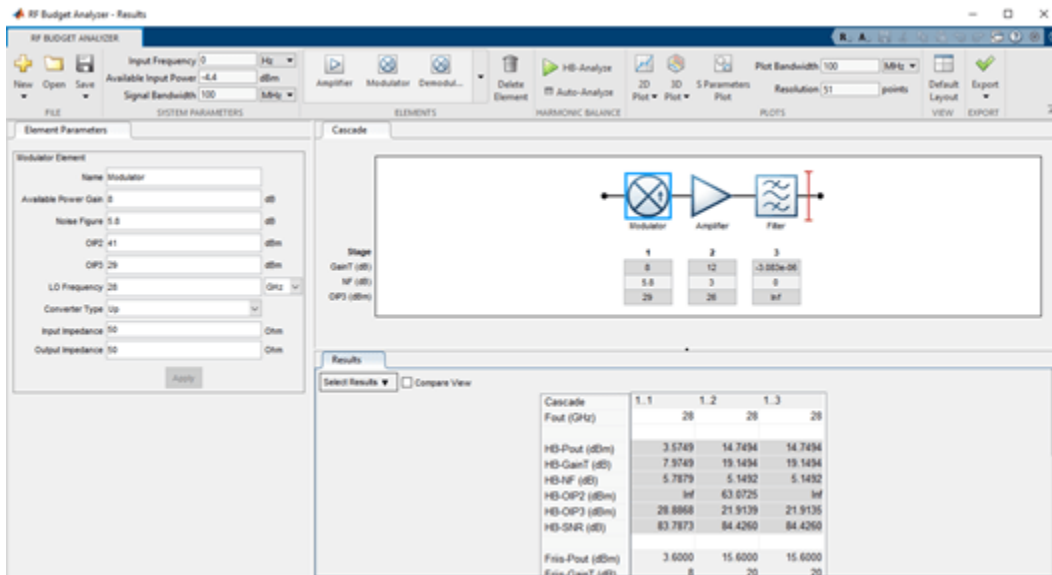
Build a cascade of RF elements with impairments.

```
elements(1) = modulator(Gain=8,NF=5.8,0IP2=41,0IP3=29,L0=28e9);
elements(2) = amplifier(Gain=12,NF=3,0IP3=26);
elements(3) = rffilter('FilterType','Chebyshev','ResponseType','Bandpass','Implementation','Transmit',
    'PassbandFrequency',[27.92 28.08]*1e9, ...
    'Zin',50, ...
    'Zout',50, ...
    'Name','Filter');
```

Construct an `rfbudget` object to perform RF budget analysis.

```
b = rfbudget(Elements=elements,InputFrequency=0,AvailableInputPower=-4.4,SignalBandwidth=100e6,SampleRate=491.52e6);
```

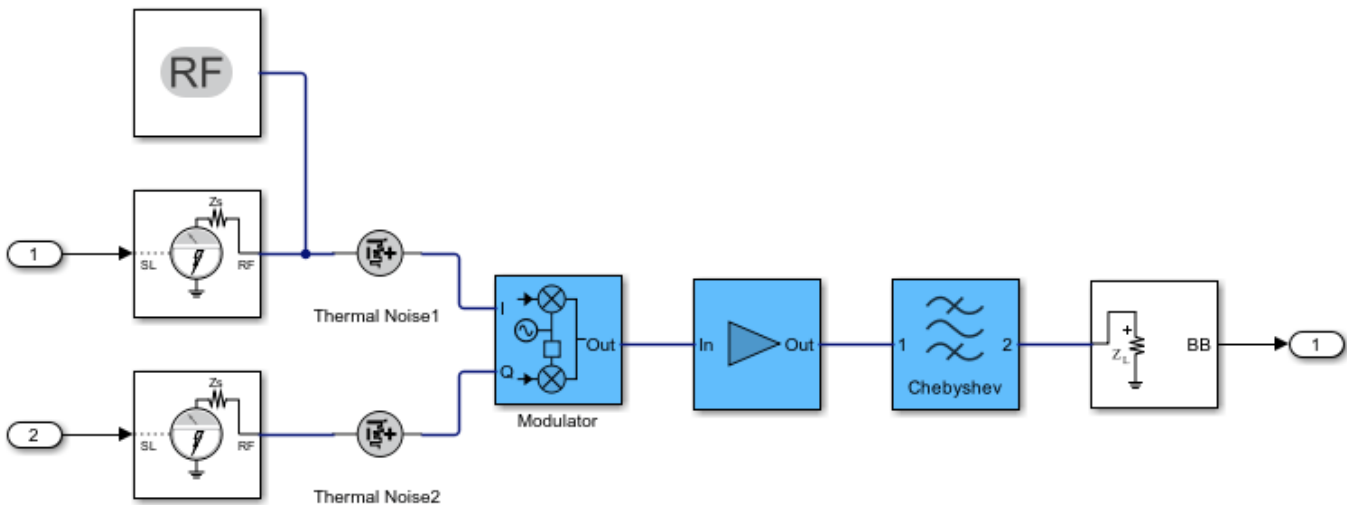
Type `show(b)` at the command line to visualize the transmitter in the **RF Budget Analyzer** app.



Create RF Model for Simulation

Create an RF model from the RF budget object for circuit envelope simulation.

```
rfs_mmW_FR2 = rfsystem(b, 'ModelName', 'mmW_FR2_Tx');
rfs_mmW_FR2.SampleTime = 1/fir.SampleRate/OSR;
open_system(rfs_mmW_FR2);
```



Add the Tx front end impairments to the NR waveform.

```
rxWaveformUp = rfs_mmW_FR2(real(waveformUp), imag(waveformUp));
release(rfs_mmW_FR2);
```

Display the 5G NR waveform.

```
Title = sprintf('5G NR %s waveform - SCS=%dkHz - BW = %dMHz', ...
    waveconfig.FrequencyRange, ...
```

```

scscarrier{1}.SubcarrierSpacing,waveconfig.ChannelBandwidth);
sp = spectrumAnalyzer('SampleRate', fir.SampleRate*OSR,...
    'ViewType','Spectrum','SpectralAverages',100,...
    'ShowLegend',true,'ChannelNames',{ 'original' 'filtered'},...
    'YLimits',[-100 0], 'Title',Title,'Method','welch','AveragingMethod','exponential');
sp(rxWaveformUp);

```



Downconversion

Downconvert the receiver waveform using the `dsp.FIRDecimator` object.

```

nr_fir_dec = dsp.FIRDecimator('DecimationFactor',OSR);
rxWaveform = nr_fir_dec(rxWaveformUp);
release(nr_fir_dec);

```

Synchronization

Estimate the timing offset.

```

carrier = nrCarrierConfig;
carrier.NCellID = waveconfig.NCellID;
carrier.NSizeGrid = waveconfig.SCSCarriers{1}.NSizeGrid;
carrier.SubcarrierSpacing = waveconfig.SCSCarriers{1}.SubcarrierSpacing;
carrier.CyclicPrefix = waveconfig.BandwidthParts{1}.CyclicPrefix;
[offset,mag] = nrTimingEstimate(carrier,rxWaveform,info.ResourceGrids.ResourceGridBWP);
waveformSync = rxWaveform(1+offset:end,:);

```

Demodulation

Demodulate the multi-antenna time domain waveform to return the received resource element array.

```
rxGrid = nrOFDMDemodulate(carrier, waveformSync);
NrSlot = floor(size(rxGrid,2)/info.ResourceGrids.Info.SymbolsPerSlot);
```

Channel Estimation and Equalization for Each Slot

Get the allocated slots and OFDM symbols per slot

```
allocatedSlots = zeros(1,NrSlot);
for i=1:length(allocatedSlots)
    allocatedSlots(i) = info.WaveformResources.PDSCH.Resources(i).NSlot;
end
L = carrier.SymbolsPerSlot; % OFDM symbols per slot

% Perform channel estimation to detect the distorted transmitted signal.
for NSlot = 1:length(allocatedSlots)

    % Grid for current slot
    SlotID = allocatedSlots(NSlot);
    rxSlot = rxGrid(:,(1:L)+(SlotID*L),:);
    refSlot = info.ResourceGrids.ResourceGridBWP(:,(1:L)+(SlotID*L),:);

    % Perform channel estimation
    estChannelGrid = nrChannelEstimate(carrier,rxSlot,refSlot);

    % Get PDSCH resource elements from the received grid
    pdschIndices = info.WaveformResources.PDSCH.Resources(NSlot).ChannelIndices;
    [pdschRx,pdschHest] = nrExtractResources(pdschIndices,rxSlot,estChannelGrid);

    % Equalize the signal to remove co-channel interference
    noiseEst = 0; % Set noise to 0 for zero-forcing equalization
    [pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

    % Perform layer demapping, symbol demodulation, and descrambling
    modulation = waveconfig.PDSCH{1}.Modulation;
    RNTI = waveconfig.PDSCH{1}.RNTI;
    [dlschLLRs,rxSymbols] = nrPDSCHDecode(pdschEq,modulation,...
        carrier.NCellID,RNTI,noiseEst);
    rxSymbols = rxSymbols{1};
```

Compute EVM

Compute the EVM on the resulting waveform to determine the impact of the impairments. Set modulation order

```
M = modOrder(modulation);
% Demodulate
refBits = qamdemod(rxSymbols,M,'UnitAveragePower',true);
% Remodulate to produce reference symbols
refSym = qammod(refBits,M,'UnitAveragePower',true);
evm = comm.EVM;
rmsEVM = evm(rxSymbols,refSym);
fprintf('slot %d: EVM = %.3f%% (%.1f dB)\n', NSlot-1, rmsEVM, 20*log10(rmsEVM/100));

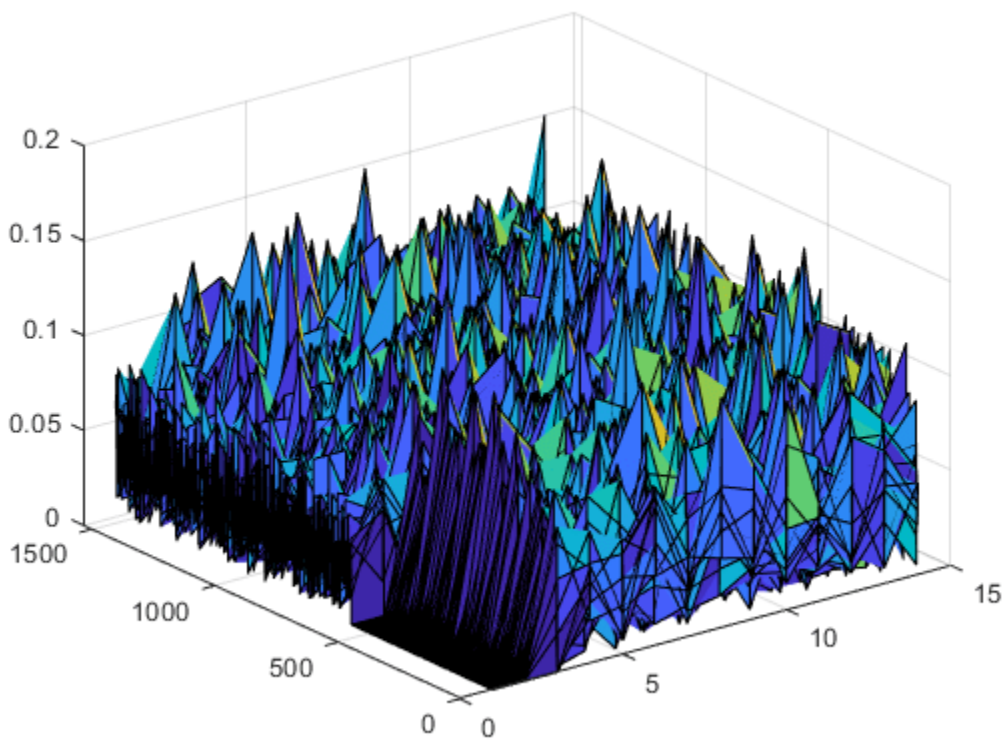
slot 1: EVM = 5.121% (-25.8 dB)
```

```
slot 2: EVM = 5.114% (-25.8 dB)
slot 3: EVM = 5.277% (-25.6 dB)
end
```

Visualize EVM in 3-D Plot

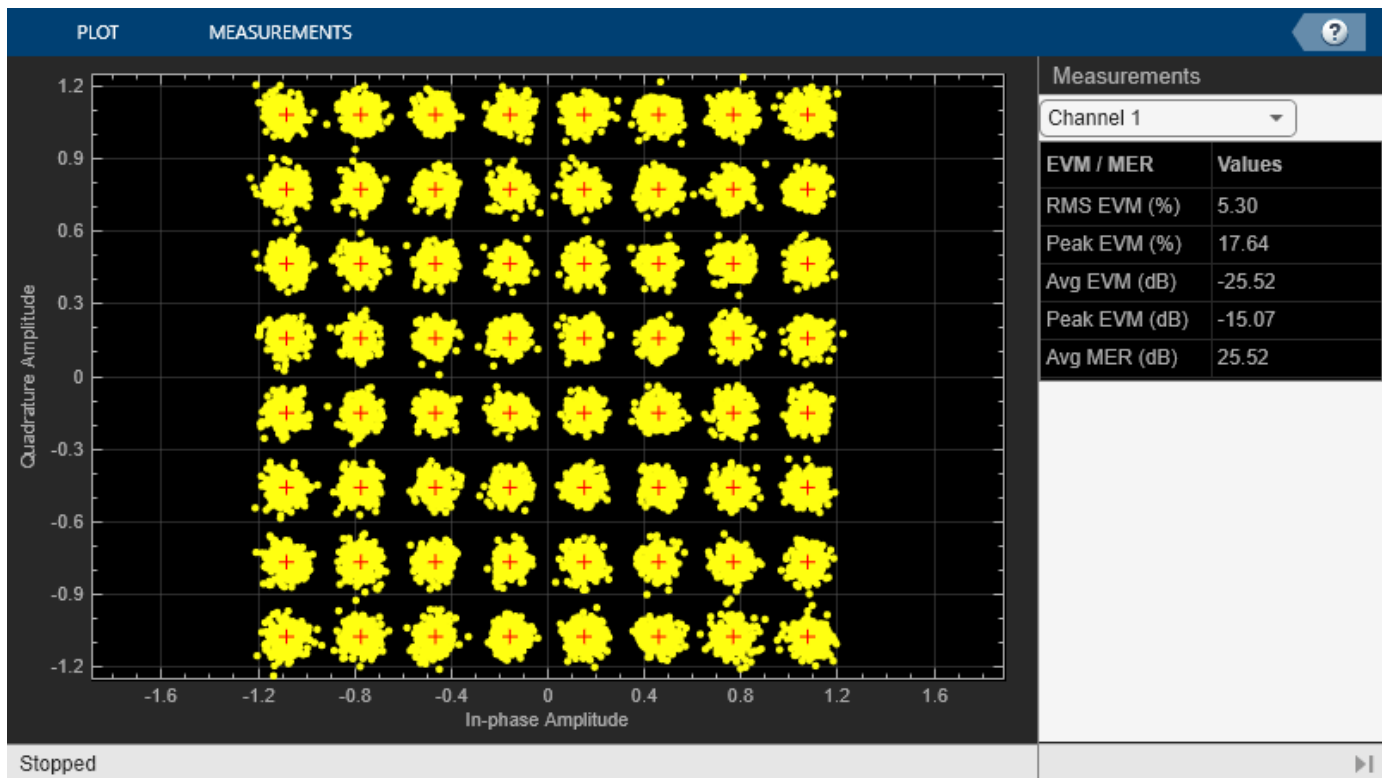
Visualize EVM of the last slot in a 3-D plot along with the 2-D constellation plot.

```
x = zeros(size(rxSlot));
x(pdschIndices) = abs(rxSymbols-refSym);
surf(abs(x))
```



Visualize the constellation diagram.

```
refC = qammod(0:2^log2(M)-1,2^log2(M))/6*.925;
constDiagram = comm.ConstellationDiagram('ReferenceConstellation',refC,...
    'XLimits',[-1 1], 'YLimits',[-1.25 1.25],...
    'EnableMeasurements',1, 'Position',[1000,250, 800,450]);
constDiagram(rxSymbols);
release(constDiagram);
```



Local function

```
function M = modOrder(Modulation)
% This function maps the modulation scheme to a QAM modulation order
switch Modulation
case 'QPSK'
    M=4;
case '16QAM'
    M=16;
case '64QAM'
    M=64;
otherwise
    M=256;
end
end
```

slot 0: EVM = 5.181% (-25.7 dB)

See Also

rfsystem

Related Examples

- “RF Receiver Modeling for LTE Reception” on page 8-92
- “Massive MIMO Hybrid Beamforming with RF Impairments” on page 8-196
- “PA and DPD Modeling for Dynamic EVM Measurement” on page 8-232
- “Circuit Envelope Simulation at MATLAB Command Line”

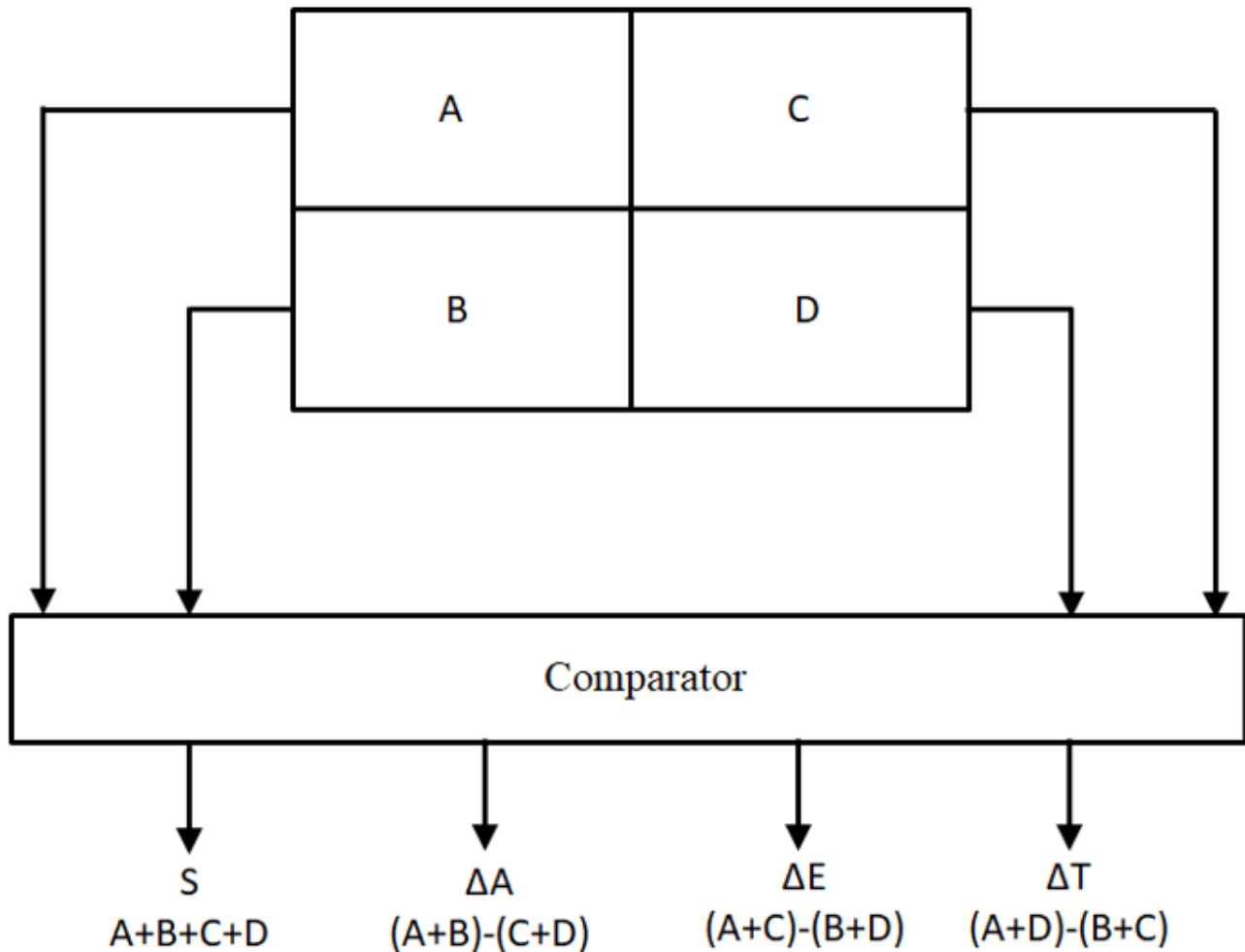
Design and Simulate Monopulse Tracking System

This example shows how to model and simulate a monopulse tracking system with a transmitter antenna, receiver antenna, and comparator[1]. The monopulse principle is used in many RADAR applications to estimate the position of the target in space by calculating the error in the azimuth and elevation direction. Monopulse tracking can be achieved using various antenna and comparator configurations. The antenna can be a 3-D antenna such as a horn or a planar antenna such as a simple patch antenna.

The comparator can be designed using four rat-race couplers that provide sum, azimuth difference, and elevation difference. Different configurations can be used based on the requirements, but the integration of the antenna and coupler needs to be perfect to achieve the desired results. Integrating the antenna and comparator directly on the hardware results in amplitude and phase mismatches, which can cause the outputs to vary. Hence, it is important to simulate the integrated model to get accurate results, anticipate the mismatches in phase, and to devise a corrective action before the hardware is fabricated.

Monopulse System Theory

The monopulse tracking radar can be used to track single or multiple targets by estimating the position of the target in space. This is a diagrammatic representation of the monopulse system.



Different types of antennas are used to implement the monopulse technique and the diagram shows that the power received from the four quadrants of antenna is given to the passive comparator network to obtain sum (S), azimuth difference (AD), elevation difference (ED), and a cross-quadrant difference (CQD), all calculated using the following equations.

- $(A + B + C + D) = S$
- $(A + B) - (C + D) = AD$
- $(A + D) - (B + C) = ED$
- $(A + C) - (B + D) = CQD$

A, B, C, and D are the four quadrants of the 2-by-2 rectangular array antenna. Three receiver channels are required to process these outputs and the target range and position can be estimated using a single pulse. Hence, this technique is named monopulse. The difference signals AD and ED are used for calculating the angular error of the target with respect to the boresight. At the boresight, the sum and difference pattern have a maximum and minimum value, respectively, and this null depth defines the angular resolution and target tracking accuracy. The null depth can be increased by improving the design of the comparator [2]. The combined simulation of the antenna

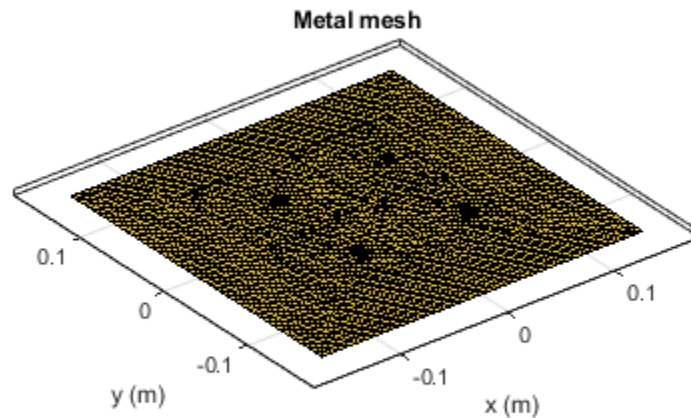
and the comparator is important as it gives the sum and difference patterns that are essential in calculating the monopulse error signal.

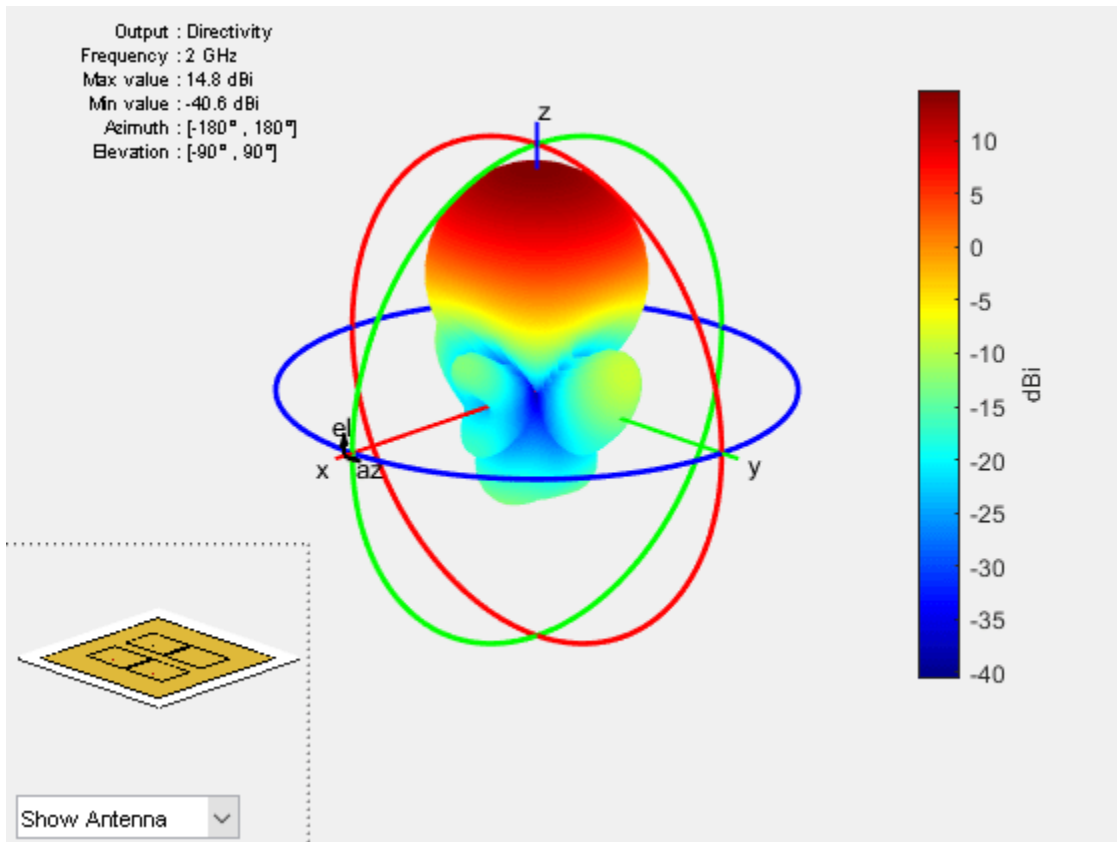
Subsystem Simulation Model

The monopulse system model is built for a transmitter and receiver antenna with four elements arranged in a 2-by-2 rectangular array configuration. The antenna pattern for the transmitter and receiver is designed at the frequency of 2 GHz. The maximum value is obtained for [azimuth elevation] of [0 90] degrees and the gain is 14.8 dBi.

```
ant = patchMicrostrip;  
ant=design(ant,2e9);  
array = pcbStack(design(rectangularArray,2e9,ant));  
figure, mesh(array, 'MaxEdgeLength',8e-3);  
figure, pattern(array,2e9);
```

```
NumTriangles: 4434  
NumTetrahedra: 0  
NumBasis:  
MaxEdgeLength: 0.008  
MeshMode: manual
```

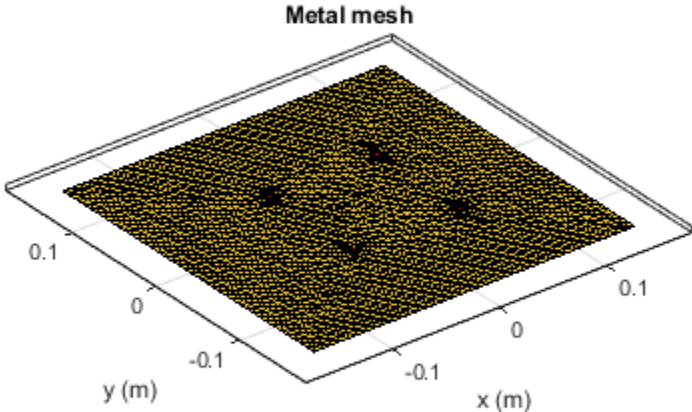


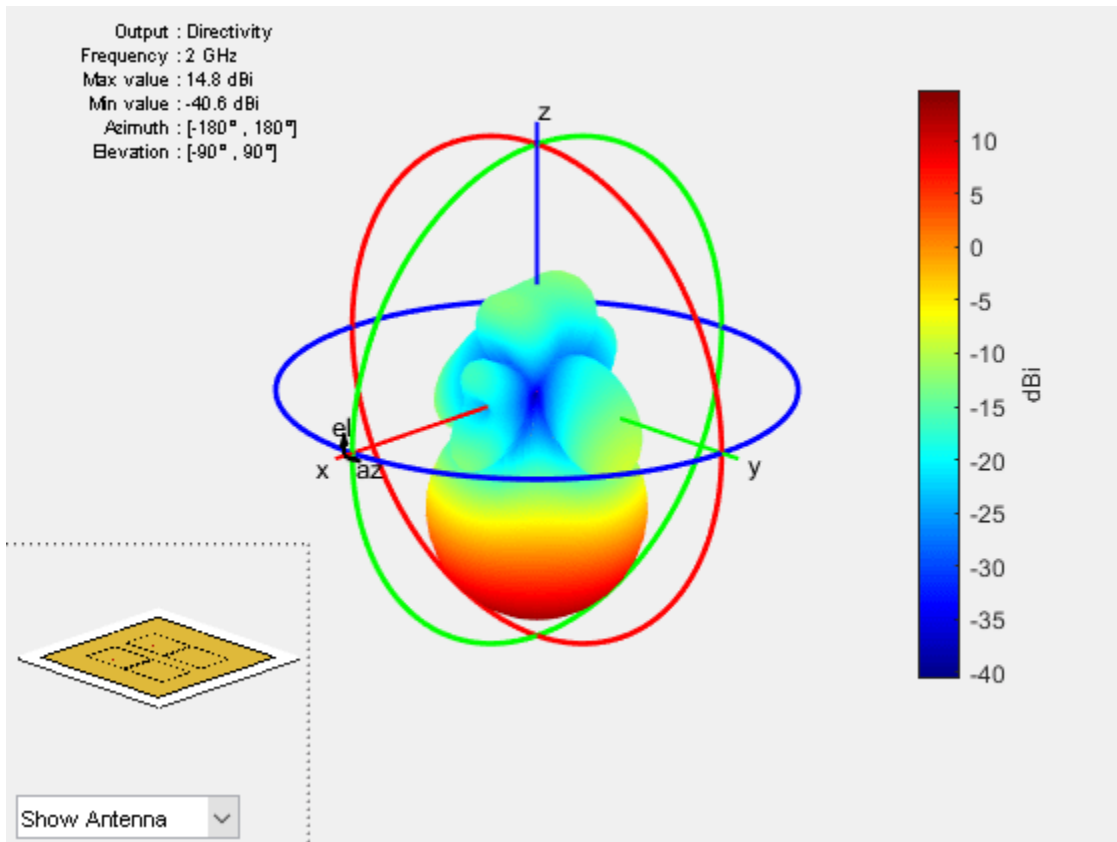


Use the 2-by-2 rectangular array antenna at the receiver, but tilt it by 180 degrees so that the two antennas face each other. The calculated pattern has a maximum value of [0 -90] degrees.

```
ant = patchMicrostrip;
ant = design(ant,2e9);
array1 = pcbStack(design(rectangularArray,2e9,ant));
array1.Tilt = 180;
figure, mesh(array1, 'MaxEdgeLength', 8e-3);
figure, pattern(array1,2e9);
```

NumTriangles: 4434
NumTetrahedra: 0
NumBasis:
MaxEdgeLength: 0.008
MeshMode: manual





The antennas are solved using the method of moments full-wave simulator. The antenna object is given as an input to the transmitter and receiver antenna block in the system model.

System Simulation Model

The simulation model contains a transmitter and a receiver. A 10 dBm input is divided into four equal parts using the Wilkinson splitter blocks and provided to the array block. The antenna array block in RF Blockset is used to specify the transmitter and receiver antenna. An antenna array object and the direction of departure can be specified in the transmitter block, and the array object and direction of arrival can be specified in the receive array block. The receiver antenna is connected to the comparator circuit, built using rat-race couplers, and the outputs obtained are sum, azimuth difference, and elevation difference. The Wilkinson and the rat-race blocks are modelled using the ideal S-parameters.

The power received in these channels with different angles of arrival on the receive antenna simulates the monopulse tracking system. The antennas in the catalog can be used in the transmit or receive antenna block. The S-parameters of different antennas and comparators can be imported into the model to verify whether these devices are suitable for a monopulse application.

The model can be further refined using S-parameters, calculated using a behavioral model or a method-of-moments solution. These S-parameters can be imported into the model in RF Blockset to simulate the monopulse system. The RF Blockset uses the harmonic balance and circuit envelope techniques and takes nonlinear effects into the simulation.

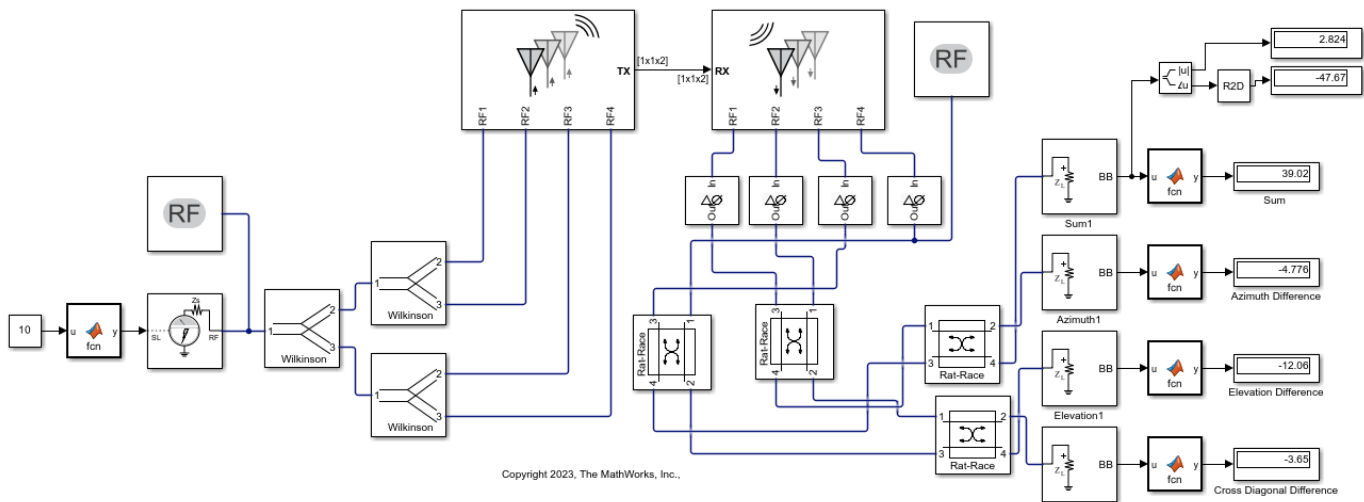
The sum port provides the value of 39.02 dBm, which matches the expected result obtained by adding input power, gain of the transmitter, and receiver antennas. The difference between the sum port and

azimuth difference provides a null depth value that is 20 dB lower than the sum signal. The null depth value needs to be improved for better performance of the monopulse system. The observed lower null depth value is caused by a small phase difference between the antenna ports. To compensate for this phase shift, the phaseShift blocks are connected to the output of the antenna ports. The null depth improves and is 44 dB lower than the sum signal.

Simulation Results

Open and simulate the monopulse tracking system model.

```
open_system("monopulse.slx")
sim("monopulse.slx");
```

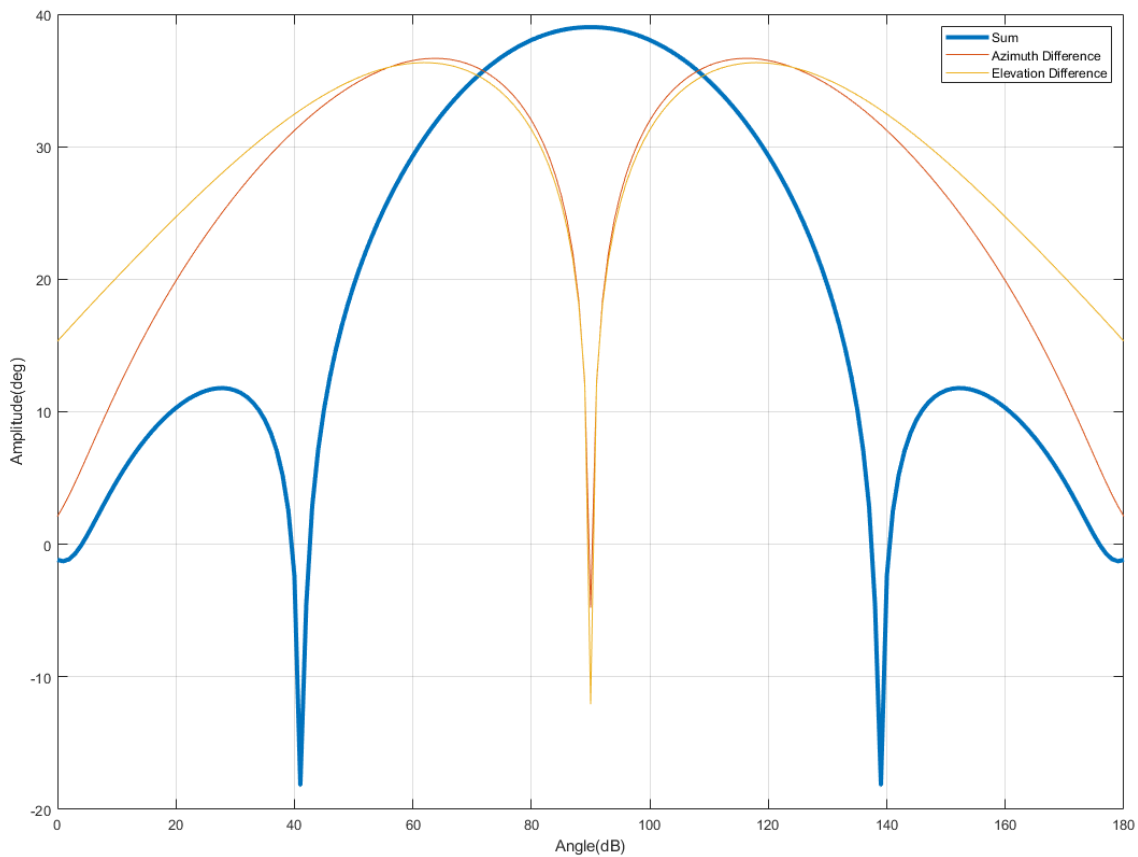


The working of the monopulse tracking system can be shown by varying the direction of the target and then measuring the results on the sum port, azimuth, and elevation difference ports. To visualize the system's behavior based on the target location, the simulations were run on this model by varying the angle of arrival on the receiver array block. The angle of arrival specifies the direction from which the signal is received and mimics the behavior of the target's location. The angle of arrival is initially specified along the boresight direction of the receiver antenna at $[0 -90]$, and then decreased in the steps of 10 degrees to measure the power received in the sum and azimuth difference ports.

As the angle of arrival is decreased, the power received on the azimuth difference port increases, and for -70 degrees, this value is greater than the sum port, which signifies that the target is not in the boresight direction. By looking at the relative ratios between the three received powers, the angular position of the receiver can be measured in real time. A similar approach can be used to measure the error observed in the elevation difference port. When the signal arrives from a target at $[45 -70]$, both the azimuth and elevation differences show the change in the values, and both values are close to 29 dB, whereas the sum is around 32 dB.

The model is simulated for all the angles of arrival and the power received in the sum and difference ports is plotted. The figure shows that the elevation difference has a greater null depth compared to the azimuth difference. The elevation difference is formed by subtracting two difference signals from two rat-race couplers, whereas the azimuth difference is formed by subtracting two sum signals from two rat-race couplers. Hence, the null depth for the elevation difference is more than the azimuth difference.

```
openfig("result.fig");
```



Design Rat-Race Coupler and Wilkinson Power Dividers Using RF PCB Toolbox

Instead of using the ideal models from RF Blockset, more realistic PCB designs can be built and simulated using the EM analysis to reflect a more practical result. The rat-race coupler and the Wilkinson power dividers can be designed and simulated using RF PCB Toolbox™ objects. The S-parameters can be saved in s3p and s4p files, respectively.

Run these commands at the command line to retrace the coupler for the comparator.

```
c = couplerRatrace;
c = design(c,2e9);
spar_ratrace = sparameters(c,linspace(1.5e9,2.5e9,51));
```

Run these commands at the command line to retrace the Wilkinson splitter for the comparator

```
d = wilkinsonSplitter;
d = design(d,2e9);
spar_wilkinson = sparameters(d,linspace(1.5e9,2.5e9,51));
```

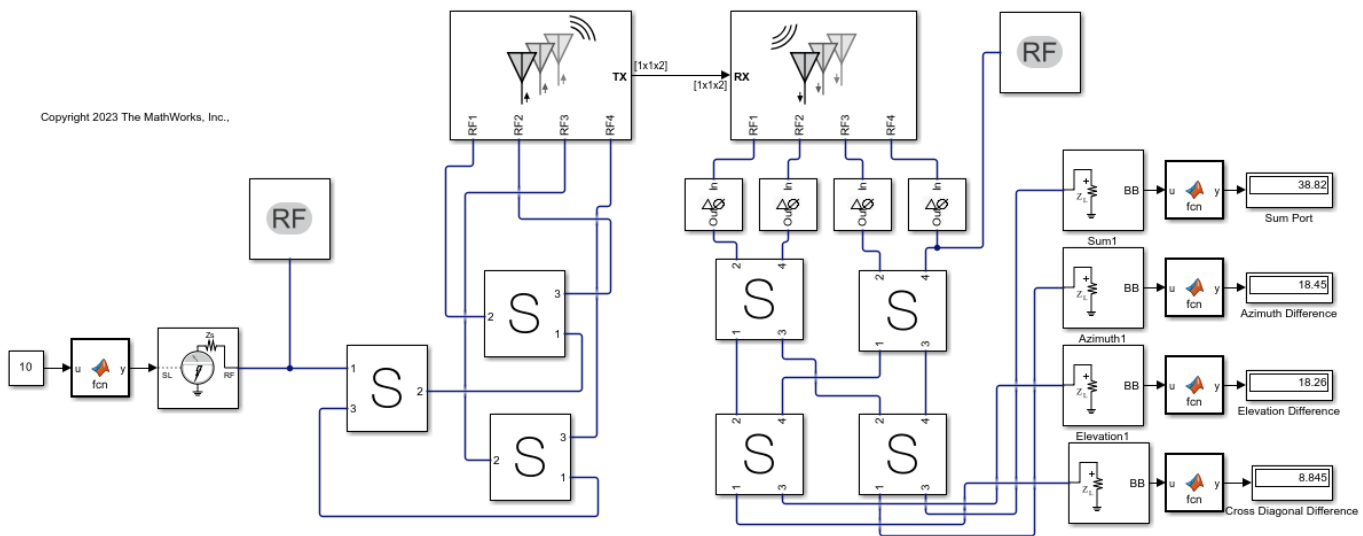
Run these commands at the command line to create the S3P and S4P files.

```
rfwrite(spar_wilkinson,'spar_wilkinson.s3p');
rfwrite(spar_ratrace,'spar_ratrace.s4p');
```

The Wilkinson and the rat-race coupler blocks are replaced with the S-parameters blocks and the S3P and S4P files calculated above are used.

Open and simulate monopulse system designed with the rat-race coupler and Wilkinson splitter.

```
open_system("monopulse_sparameters.slx")
sim("monopulse_sparameters.slx");
```



The result shows that the null depth is particularly good when the behavioral S-parameters are used, but when the method-of-moments solution is used on the coupler object, the null depth degrades and is 20 dB lower than the sum signal, which is what we might observe when the design is fabricated. Efficient designs of rat-race couplers that provide good isolation between the ports and exact phase difference between the output ports can improve the null depth.

The result shows that the practical designs need to be improved and the amplitude and phase on each port need to be balanced perfectly to achieve the desired result in the monopulse system. This can be prototyped in the model by adding the phase shift block to the rat-race couplers and adjusting the phase of the output signals such that the elevation and azimuth difference can be improved.

See Also

"Rat-Race Coupler - Visualize and Analyze" (RF PCB Toolbox)

References

- [1] Joshi, Sourabh. Kulkarni, Shashank. " Model-Based Design and Simulation of Monopulse Tracking System." *IEEE MAPCON* Dec 2022.
- [2] Kian Sen Ang, Y. C. Leong and Chee How Lee, "A wide-band monopulse comparator with complete nulling in all delta channels throughout sum channel bandwidth," in *IEEE Transactions on Microwave Theory and Techniques*, vol. 51, no. 2, pp. 371-373, Feb. 2003.

Enable Model Protection and Accelerator Modes in RF Blockset Models

This topic discusses the workflow for enabling model protection and accelerator modes in RF Blockset™ models designed using the Circuit Envelope environment. To demonstrate this workflow, an automatic gain control (AGC) algorithm for RF baseband processing is designed. You can also enable model protection and accelerator modes in Equivalent Baseband and Idealized Baseband environments.

Overview

You can conceal the contents of your model by creating a protected model in the SLXP file format. Doing so is useful when you want to share a model with a third party without revealing intellectual property. Protecting a model conceals the implementation details of the original model by compiling it into a referenced model. Additionally, if you opt to protect your model with a password, the software uses AES-256 encryption. For more information, see “Protect Models to Conceal Contents” (Simulink Coder).

You can also enable rapid accelerator and model reference accelerator modes in RF Blockset models to speed up your simulations. These accelerator modes are ideal when you need to perform iterative, long-running, computationally expensive simulations such as Monte Carlo simulation or system optimizations on your RF system. Rapid accelerator mode speeds up simulation by generating an executable for your model. The exact speedup varies depending on the model. For more information, see “How Acceleration Modes Work” and “Choose Simulation Modes for Model Hierarchies”. The accelerator modes are useful when you tune parameters outside of the RF system, for example, system optimizations on models containing RF systems, specifically when tuning parameters outside of the RF network and keeping parameters of blocks within the RF system fixed.

To use a protected model containing Circuit Envelope blocks, you first need to ensure that the necessary RF Blockset libraries and dependencies are loaded in MATLAB®. Note that you only need to do this once per MATLAB session, but you must do this before the first time you use a protected model containing Circuit Envelope blocks in that MATLAB session. Execute the following command to preload the Circuit Envelope Utilities library and to configure the RF Blockset environment.

```
open_system simrfV2util1
```

Workflow

The workflow to enable model protection and accelerator modes in the Circuit Envelope environment is as follows:

- 1 RF Model Creation — Create your RF system using Circuit Envelope blocks. Start by designing your RF network in the **RF Budget Analyzer** app, then export the network to the Circuit Envelope environment. Simulate and test your model behavior. For more information, see “Design RF Direct-Conversion Receiver” on page 8-284.
- 2 RF Model IP Protection — Conceal your model using an SLXP model file. The model protection workflow enables you to view and simulate the model with password protection. You can also generate a test harness for your generated SLXP model file. For more information, see “Protect Circuit Envelope Model” on page 8-288.
- 3 Baseband Processing and Algorithm Design — Use the SLXP file from step 2 in a model to process baseband signals. In this step, you design the baseband signal processing and

communication algorithm around the RF network that you designed in step 1. Leverage accelerated simulations for rapid design iteration. For more information, see “Implement Automatic Gain Control for RF Receiver” on page 8-295.

You can also perform baseband algorithm tuning and batch simulations by enabling rapid accelerator mode in the top-level model. Doing so can speed up top-down simulation run time across the model reference hierarchy, enabling you to more quickly perform parameter sweeping, tuning, optimization, and batch simulations.

Licensing Considerations

- To create an RF Blockset protected model, you need a Simulink Coder™ license in addition to an RF Blockset license. If you want to generate a read-only web view to include with the protected model, you additionally need a Simulink Report Generator™ license.
- To use an RF Blockset protected model, you need Simulink® and RF Blockset.

Limitations

Rapid Accelerator mode does not support:

- Antenna block
- RF Measurement Testbench
- Circuit Envelope testbench library
- Frequency domain simulation
- Phase noise simulation

Model reference accelerator mode does not support:

- Antenna block
- RF Measurement Testbench
- Circuit Envelope testbench library

RF Blockset does not support Simulink Compiler™.

See Also

Related Examples

- “Comparing Performance”
- “Design Your Model for Effective Acceleration”
- “Model Reference Requirements and Limitations”
- “Reference Protected Models from Third Parties”

Design RF Direct-Conversion Receiver

This example shows how to design an RF direct-conversion receiver (DCR) that supports variable gain and attenuation control. The design workflow entails the following steps:

- 1 Start with a preliminary design in the RF Budget Analyzer app.
- 2 Export the RF budget to RF Blockset™.
- 3 Make further modifications to the circuit envelope model.
- 4 Simulate and test the model behavior.

Design Receiver in RF Budget Analyzer App

You can model the DCR with an RF chain that has the following stages:

- Attenuator — Prevents saturation and overload of the RF front-end by strong incoming signals that can desensitize or damage the receiver
- Surface acoustic wave (SAW) filter — Performs RF band selection
- Low-noise amplifier (LNA) — Amplifies the received RF signal without significantly degrading the signal-to-noise ratio (SNR)
- Demodulator — Downconverts the RF signal to baseband
- Baseband amplifier — Regulates the output signal power, for example, to keep it within the dynamic range of a downstream analog-to-digital converter (ADC)

Typically, a low-pass filter is appended to the end of the RF chain for the DCR to extract the baseband signal within the band of interest, that is, to perform channel selection. For the RF budget analysis here, assume ideal channel-selection filtering at the output of the demodulator.

Design this RF chain either interactively with the **RF Budget Analyzer** app or programmatically with `circuit` and `rfbudget` objects.

```
RFDCRBudget = rfbudget([ ...
    attenuator(Name='Attenuator',Attenuation=0.2), ...
    nport(Name='SAWFilter',FileName='SAW_Filter_Data.s2p'), ...
    amplifier(Name='LNA',Gain=22,NF=7), ...
    modulator(Name='Demodulator',ConverterType='Down',LO=2.45e9,Gain=-7,NF=10), ...
    amplifier(Name='BasebandAmplifier',Gain=40,NF=14)], ...
    InputFrequency=2.45e9, ...
    AvailableInputPower=-100, ...
    SignalBandwidth=2e6)
```

```
RFDCRBudget =
```

```
rfbudget with properties:
```

```

    Elements: [1x5 rf.internal.rfbudget.Element]
    InputFrequency: 2.45 GHz
    AvailableInputPower: -100 dBm
    SignalBandwidth: 2 MHz
    Solver: Friis
    AutoUpdate: true
```

Analysis Results

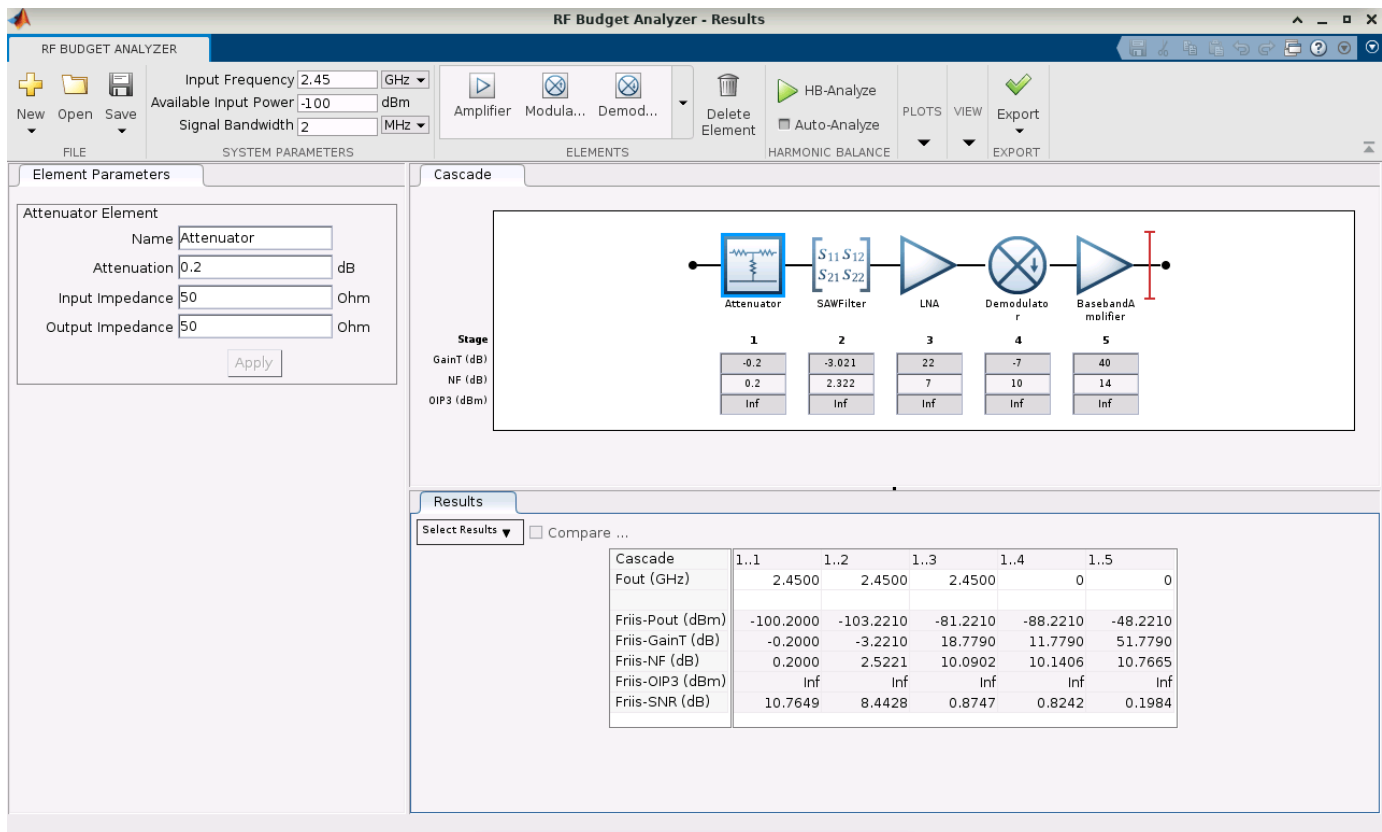
```

OutputFrequency: (GHz) [ 2.45 2.45 2.45 0 0]
OutputPower: (dBm) [ -100.2 -103.2 -81.22 -88.22 -48.22]
TransducerGain: (dB) [ -0.2 -3.221 18.78 11.78 51.78]
NF: (dB) [ 0.2 2.522 10.09 10.14 10.77]
IIP2: (dBm) [ ]
OIP2: (dBm) [ ]
IIP3: (dBm) [ Inf Inf Inf Inf Inf]
OIP3: (dBm) [ Inf Inf Inf Inf Inf]
SNR: (dB) [ 10.76 8.443 0.8747 0.8242 0.1984]

```

Execute the following command to visualize the design of the RF chain and its budget analysis in the **RF Budget Analyzer** app.

```
rfBudgetAnalyzer(RFDCRBudget)
```

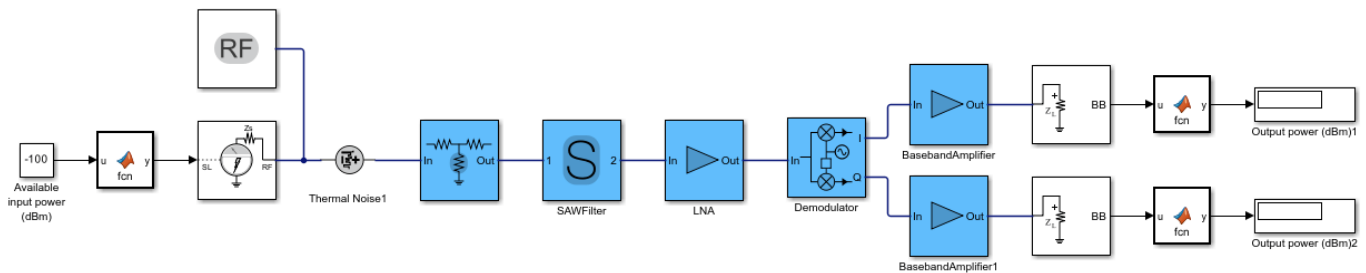


Export RF Budget to RF Blockset to Automatically Generate Circuit Envelope Model

The circuit envelope model supports multicarrier simulation of the DCR while taking into account nonlinear effects and impairments, mixer phase noise and LO-RF isolation, and the impact of and interactions between the signal of interest and other in-band and out-of-band interfering signals.

Create a circuit envelope model from the `rfbudget` object or interactively from within the **RF Budget Analyzer** app.

```
exportRFBlockset(RFDCRBudget)
```



As seen in the automatically generated model, this example makes the following assumptions:

- In the Configuration block, the simulation step size is automatically set to 1/8 times the reciprocal of the input signal envelope bandwidth. In this example, this input signal bandwidth is 2 MHz and is centered around the input carrier frequency of 2.45 GHz. This corresponds to a simulation step size of 62.5 ns and a simulation bandwidth of 16 MHz.
- Like that of the budget analysis, the demodulator is configured to perform ideal low-pass channel-selection filtering within the simulation bandwidth around the output carrier of interest, which is at DC (0 Hz). Although it is out of scope for this example, you can modify the channel-selection filter options in the IQ Demodulator block to model realistic and realizable channel-selection designs and implementations.
- The demodulator is implemented with a quadrature architecture, which is a standard modeling paradigm when the signal of interest is QPSK-modulated and contains both in-phase (I) and quadrature (Q) components. It is standard modeling practice to combine the two extracted, real-valued baseband signals, representing the two components as a single complex-valued baseband signal ($I+jQ$) and feeding this signal into the baseband QPSK demodulator.

The S-Parameters block describing the SAW filter uses rational fitting in order to simulate frequency data in the time domain. Note that at 2.45 GHz the filter introduces a phase rotation of approximately -59 degrees. While it is not shown here, if you simulate this model, you can see that the complex input signal is partly downconverted on the I and Q branches, and thus the instantaneous output powers on the two branches can be different. However, the time-averaged output powers on the I and Q branches are expected to be the same. In addition, the average of the I and Q instantaneous output powers, that is $(I(t)+Q(t))/2$, is expected to match the output power reported in the budget analysis.

Configure Model to Support Automatic Gain Control

Make the following changes to the model to prepare it for use within an automatic gain control (AGC) loop:

- Replace the Attenuator block with a Variable Attenuator block. Control the variable attenuation using one of the model's root-level input ports.
- Replace the Amplifier blocks with VGA blocks. Control the variable gain using one of the model's root-level input ports.
- Assume that the IP2 and IP3 values for each VGA block are gain independent. Accordingly, specify them as constant values, and configure the VGA block to use the input-referred convention for the intercept points: IIP2 and IIP3. If you want to specify gain-dependent IP2 and IP3 values, then you can configure the VGA block to use the output-referred convention for the intercept points and then use lookup table blocks to specify the OIP2 vs. Gain and OIP3 vs. Gain relations. Note that this is out of scope for this example.
- Configure the RF Output block to output complex baseband signals for I and Q each, and take the real part of each signal for the purposes of data type conversion. Note that the imaginary part of

each signal is zero. Typically, a real passband representation is the more appropriate choice, given that the output signal is at DC (0 Hz). However, this model uses frame-based input signals to accelerate simulation time, and the RF Blockset Circuit Envelope library does not support frame-based processing when the Output block's **Output** is set to Real Passband. Although this example does not explicitly show it, you can confirm that each I and Q signal has zero imaginary part.

- Set **Samples per frame** to 256 in the Configuration block to help accelerate simulation time.
- Configure the signal attributes of the model's root-level input, specifically the one corresponding to the input RF signal In RF, to be of complex-valued signal type. This input RF signal is a complex equivalent baseband signal, and it corresponds to the envelope of the RF carrier, which is at 2.45 GHz.

Specify amplifier nonlinearities for the VGA blocks.

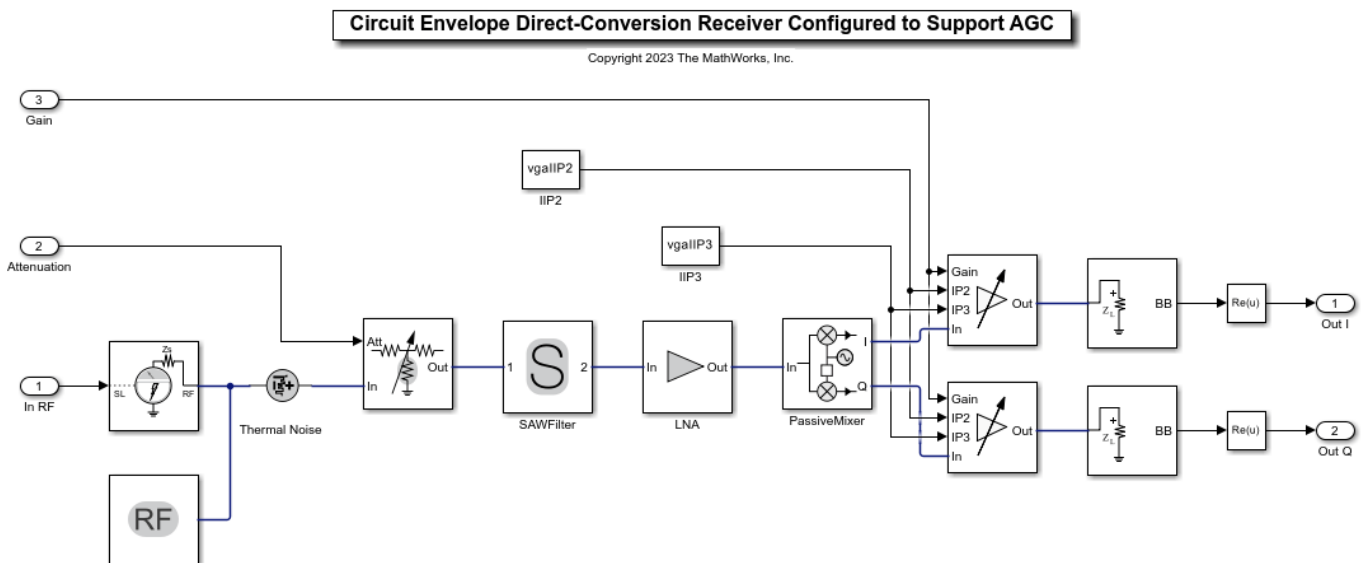
```
vgaIIP2 = 50;
vgaIIP3 = 30;
```

Specify mixer impairments for the IQ Demodulator block.

```
mixerParameters.L0ToRFIsolation = 105;
mixerParameters.PhaseNoiseFrequencyOffsets = 1e6*[0.2 1 2 3 7];
mixerParameters.PhaseNoiseLevels = [-70 -80 -95 -110 -120];
mixerParameters.IIP2 = 55;
mixerParameters.CSFilterPassBandEdgeFrequency = 50e6;
```

View the final model.

open_system RFDRCR



Protect Circuit Envelope Model

This example shows how to use Simulink Coder™ model protection with an RF Blockset™ circuit envelope model so that it can be shared as an SLXP file with a third party without revealing implementation details and intellectual property.

Model Protection Overview

You can configure the protected model to allow end users to perform only a specific subset of tasks, such as viewing and simulation. In addition, you can optionally password protect each option with AES-256 encryption. For example, you can enable third party end users to simulate the protected model without a password, and you can protect the ability to open a read-only web view of the model with a password to conceal the implementation details. Meanwhile, when you share the same protected model with your internal team collaborators, you can also provide them the password to allow them to see and inspect the implementation details too. Furthermore, you can create a harness model that references the protected model. End users can use this harness model along with the interface report to understand how to use the protected model and how to interface it with their existing systems. For more information, see “Protect Models to Conceal Contents” (Simulink Coder).

Create Protected Model for Simulation and Password-Protected Viewing

Protect the model of an RF direct-conversion receiver (DCR) configured for use within an automatic gain control (AGC) loop. Note that all parameters of blocks from the Circuit Envelope library are *nontunable block parameters*, whose values cannot change during simulation. Consequently, none of the parameters of blocks *within the RF network* can be designated as tunable parameters of the protected model. However, the VGA block and Variable Attenuator block accept Simulink® signal controls, and these are root-level input signals to the model. Furthermore, the IIP2 and IIP3 values of the VGA block are controlled by Constant block sources in the model, whose parameters are tunable. For more information, see “Tune and Experiment with Block Parameter Values”.

Specify amplifier nonlinearities for the VGA blocks. These are tunable parameters for the model.

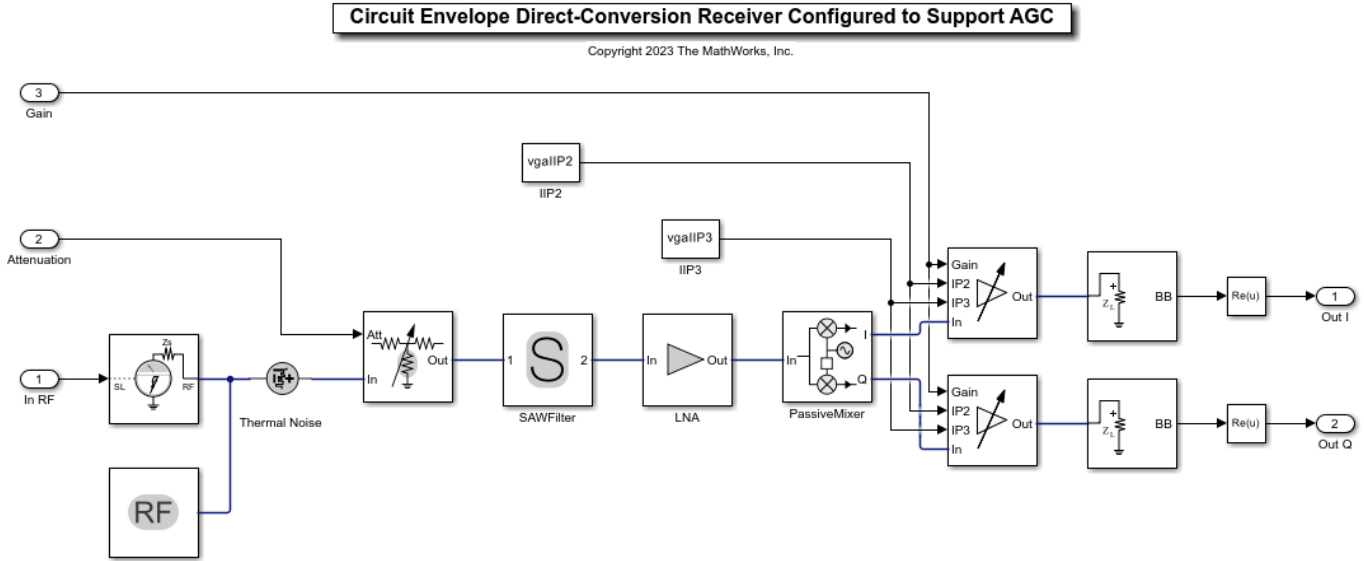
```
vgaIIP2 = 50;
vgaIIP3 = 30;
```

Specify mixer impairments for the IQ Demodulator block. These are nontunable parameters for the model.

```
mixerParameters.L0ToRFIsolation = 105;
mixerParameters.PhaseNoiseFrequencyOffsets = 1e6*[0.2 1 2 3 7];
mixerParameters.PhaseNoiseLevels = [-70 -80 -95 -110 -120];
mixerParameters.IIP2 = 55;
mixerParameters.CSFilterPassBandEdgeFrequency = 50e6;
```

Open the model of the DCR.

```
model = 'RFDCR';
open_system(model)
```



You can protect the model interactively or programmatically; both approaches are illustrated here. In this example, protect the model such that end users can simulate it without a password and open a read-only web view with the password. In addition, automatically generate a test harness model that provides an isolated environment to test the protected model. Note that Simulink Coder is required to create the protected model, while Simulink Report Generator™ is required to generate the web view. Alternatively, if you do not have Simulink Report Generator, then you can still create a protected model enabled only for simulation.

In either case, temporarily suppress the warning regarding non-passivity of noise correlation data derived from the rational fit of the S-parameters. This consideration is specific to this model and S2P file.

```
warningState(1) = warning('off', ...
    'simrf:simrfV2errors:DerivedNoiseNotPassive');
restoreWarningState(1) = onCleanup(@( )warning(warningState(1)));
```

To interactively create a protected model, follow the steps in “Explore Protected Model Capabilities”. If you have Simulink Report Generator, the option to create a read-only view appears as shown. Otherwise, the option does not appear.

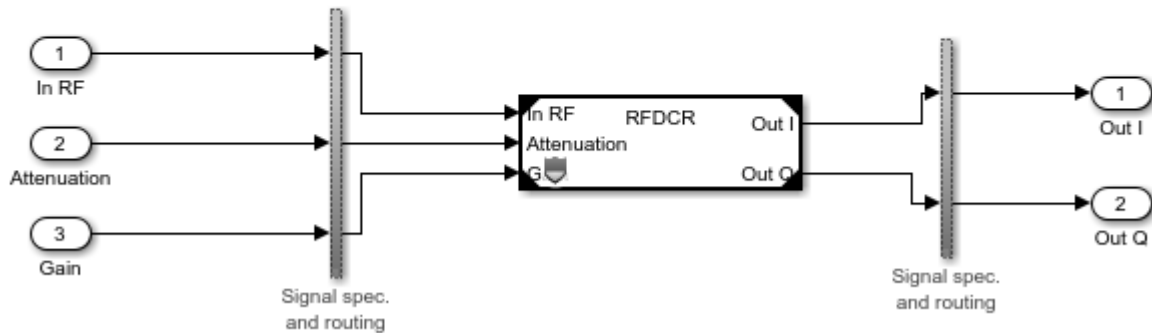
To programmatically create a protected model, use the `Simulink.ModelReference.protect` (Simulink Coder) function. Prior to calling this function, optionally specify passwords for the various functionalities of the protected model that you intend to enable. In this example, set the password as "RF Blockset" for the read-only view, and suppress the warning regarding the omission of password protection for simulation of the protected model. In addition, if you have Simulink Report Generator, set `generateWebView` to `true` in the following code.

```
warningState(2) = warning('off', ...
    'Simulink:protectedModel:EncryptOnNoPasswordForCategory');
restoreWarningState(2) = onCleanup(@()warning(warningState(2)));

generateWebView = false; % Requires Simulink Report Generator
Simulink.ModelReference.ProtectedModel.setPasswordForView( ...
    model, 'RF Blockset')
[~] = Simulink.ModelReference.protect(model, ...
    Harness=true,WebView=generateWebView,Encrypt=true,Report=true, ...
    TunableParameters={'vgaIIP2','vgaIIP3'});
```

```
Creating protected model for 'RFDCR'.
### Starting serial model reference simulation build.
### Generating code for Physical Networks associated with solver block 'RFDCR/Configuration/Solve
done.
### Successfully updated the model reference simulation target for: RFDCR
Finished creating protected model 'C:\TEMP\Bdoc23a_2213998_3568\ib570499\4\tp1199c11b\simrf-ex178
Creating harness model for protected model 'RFDCR.slxp'.
Finished creating harness model 'C:\TEMP\Bdoc23a_2213998_3568\ib570499\4\tp1199c11b\simrf-ex1781
```


Use this harness model to simulate the protected model in an isolated environment. This harness model must have access to supporting files, such as a MAT-file with base workspace definitions or a data dictionary. For information about the protected model environment, functionality, and interface, [open the protected model report](#).



Revert the temporary warning suppressions to restore the warning states.

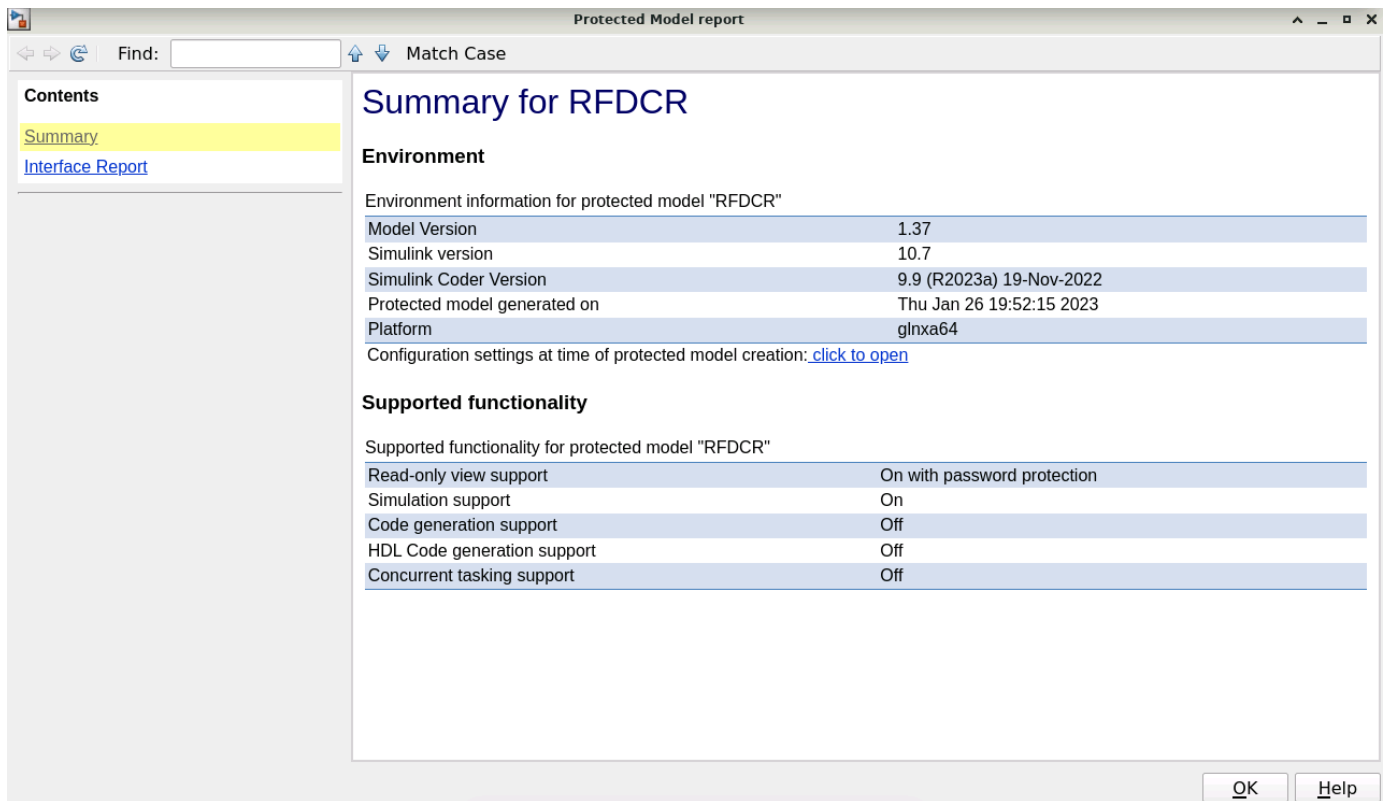
```
clear restoreWarningState
```

Examine Protected Model Interface Report

View the protected model interface report, which you can use as a reference to communicate with the end users of your protected model. To open the report interactively, you can right-click on the newly created SLXP file in the Current Folder browser and select **Open Report**. To open the report programmatically, execute the following MATLAB command.

```
Simulink.ProtectedModel.open('RFDCR', 'report')
```

Select the Summary subpage to view details about the environment, platform, and version information, as well as the model functionality supported by the protected model.



Select the Interface Report subpage to view details such as the input and output specifications and tunable parameters associated with the model.

The screenshot shows a window titled "Protected Model report" with a search bar and "Match Case" option. The left sidebar contains a "Contents" panel with "Summary" and "Interface Report" (highlighted). The main content area is titled "Interface Report for RFDCCR" and includes a "Table of Contents" with links to "Inports", "Outputs", "Interface Parameters", and "Data Stores".

Inports

Block Name	Data Type	Dimension
<Root>/In RF	creal_T *	[256]
<Root>/Attenuation	real_T *	1
<Root>/Gain	real_T *	1

Outputs

Block Name	Data Type	Dimension
<Root>/Out I	real_T *	[256 1]
<Root>/Out Q	real_T *	[256 1]

Interface Parameters

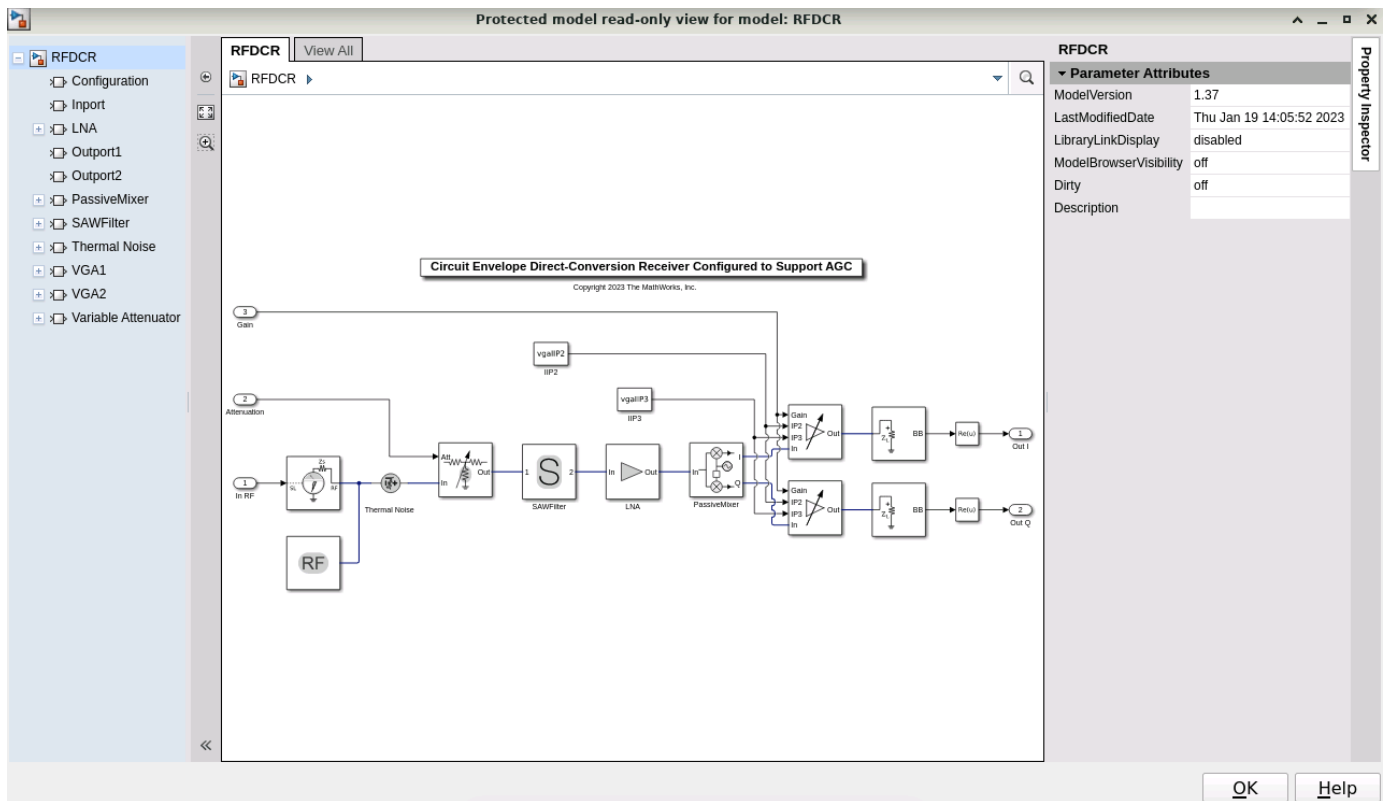
Parameter Source	Data Type	Dimension
vgallP2	real_T	1
vgallP3	real_T	1

Buttons for "OK" and "Help" are visible at the bottom right of the window.

Examine Protected Model Read-Only Web View

If available, open the read-only web view of the protected model. If applicable, enter the password in the respective dialog prompt. In this example, the password for the read-only view is "RF Blockset". Note that Simulink Report Generator is not required to open the web view, but it is required to generate it. To open the read-only view interactively, you can right-click on the newly created SLXP file in the Current Folder browser and select **Open Webview**. To open the read-only view programmatically, execute the following MATLAB command.

```
Simulink.ProtectedModel.open('RFDCCR', 'webview')
```



Test Protected Model Using Harness

Execute the following MATLAB command to simulate the test harness model. Note that no password is required for simulation in this example.

```
sim RFDCR_harness
```

Implement Automatic Gain Control for RF Receiver

This example shows how to use a protected RF Blockset™ circuit envelope model as a referenced model within a system that implements baseband signal processing, communications algorithms, and adaptive architectures around an RF network. To illustrate these concepts, this example designs and implements an automatic gain control (AGC) algorithm for an RF direct-conversion receiver (DCR) in a context similar to that of the ZigBee®-like application explored in “Top-Down Design of an RF Receiver” on page 8-166.

In addition, this example demonstrates how to use model references, modeling hierarchies, and variants for modular development, protected model usage, accelerated simulation, and overall rapid design iteration. For more information about these concepts, see “Model Reference Basics”.

ZigBee-Like System Containing RF DCR and Subject to Strong Interference

The top-level system consists of the following components:

- Baseband transmitter used to generate a ZigBee-like waveform spectrally representative of signals that conform to the IEEE® 802.15.4 standard
- Wideband interferer used to generate a WCDMA-like, in-band but out-of-channel blocker spectrally representative of interfering signals noncompliant with the standard
- Circuit envelope model of an RF DCR
- AGC algorithm implemented via a MATLAB Function block to supply the variable gain and variable attenuation control signals to the RF DCR
- Peak detector used by the AGC feedback loop to regulate the fast response, which decreases gain and increases attenuation when the peak power exceeds a prespecified threshold
- Average power meter used by the AGC feedback loop to regulate the slow response, which adjusts gain and attenuation to steadily bring the average power and signal-to-noise ratio (SNR) to the target levels
- Analog-to-digital converter (ADC) used to convert the baseband analog signal output by the RF receiver into baseband digital signals consumed by downstream communications and digital signal processing
- Baseband receiver that receives and processes the 802.15.4 waveform and computes the chip error rate (ChER) system-level performance metric.

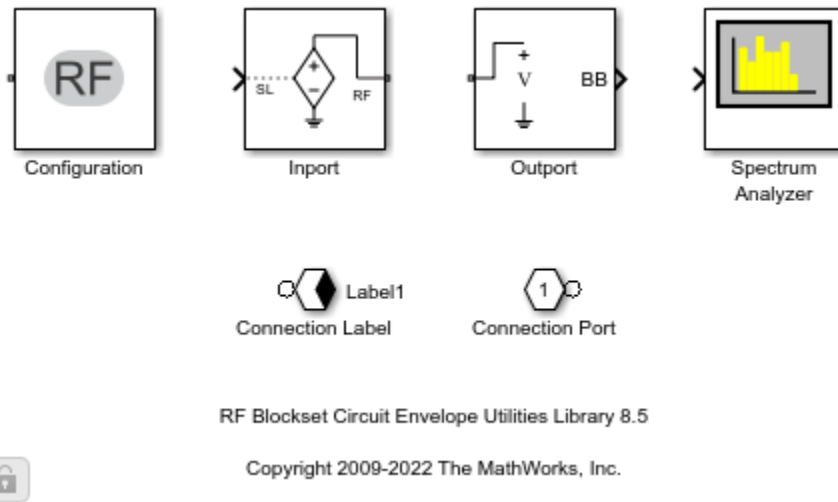
To begin, specify the baseband and RF parameters that are required for simulation. Note that `vgaIIP2` and `vgaIIP3` are tunable parameters of the protected model for the RF DCR. The remaining parameters are associated with various baseband components in the top-level system.

```
% Baseband parameters
bitRate = 250e3;
spf = 4; % samples per frame
bps = 4; % bits per ZigBee symbol
sps = 16; % samples per OQPSK symbol
cps = 32; % chips per ZigBee symbol
chipRate = bitRate*cps/bps;

% RF parameters
oversampling = 4; % no out-of-band signal
vgaIIP2 = 50; % IIP2 of RF receiver's VGA
vgaIIP3 = 30; % IIP3 of RF receiver's VGA
```

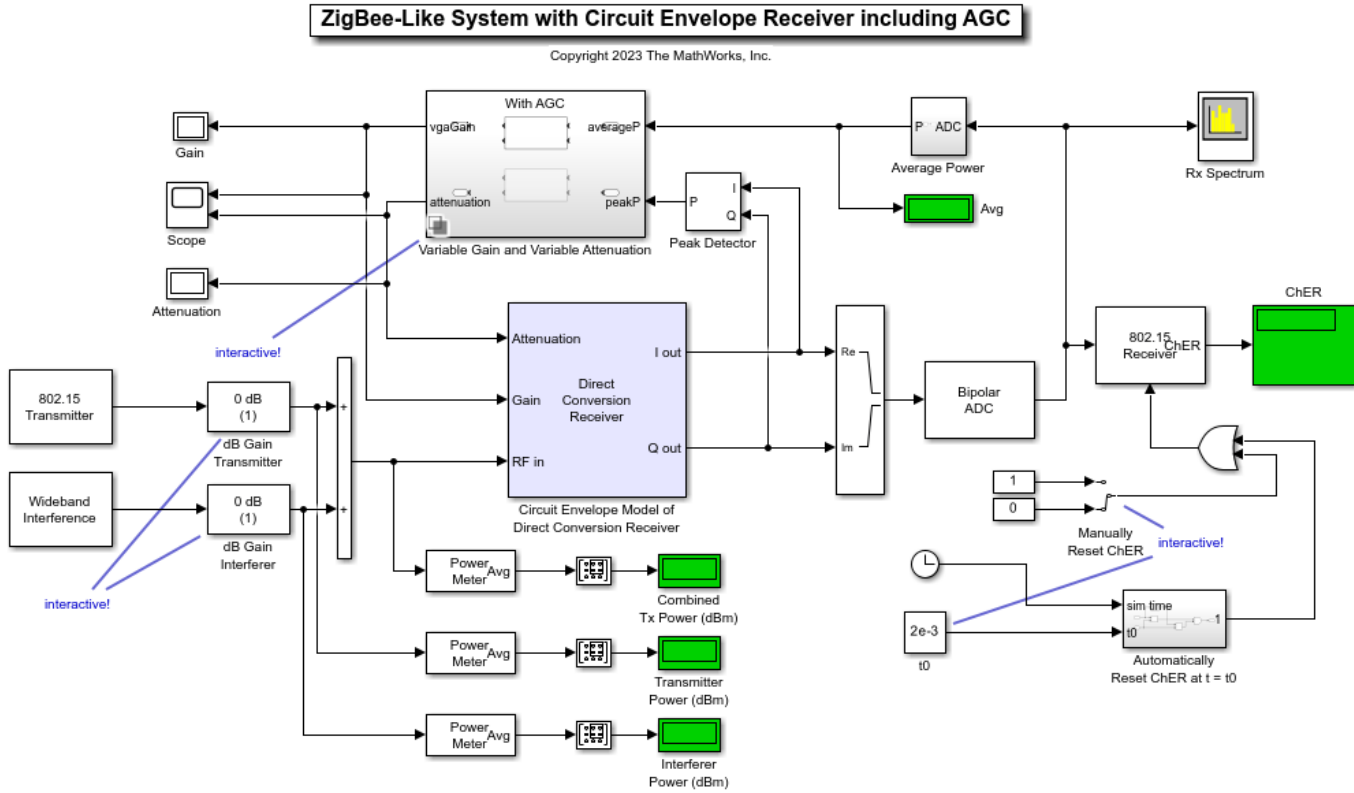
To use a protected model containing Circuit Envelope blocks, you first need to ensure that the necessary RF Blockset libraries and dependencies are loaded in MATLAB®. Note that you only need to do this once per MATLAB session, but you must do this before the first time you use a protected model containing Circuit Envelope blocks in that MATLAB session. Execute the following command to preload the Circuit Envelope Utilities library and to configure the RF Blockset environment.

```
open_system('simrfV2util')
```



Open the top-level model. Note that the model contains interactive controls, which you can use to quickly modify it.

```
model = 'RFDCRWithAGC';  
open_system(model)
```



Reference Protected Model of RF DCR

Open the subsystem containing the referenced model of the RF DCR. Although the SLXP file is platform-specific, the model's `PostLoadFcn` callback is set up to automatically choose the correct version of the SLXP file for your specific platform.

```

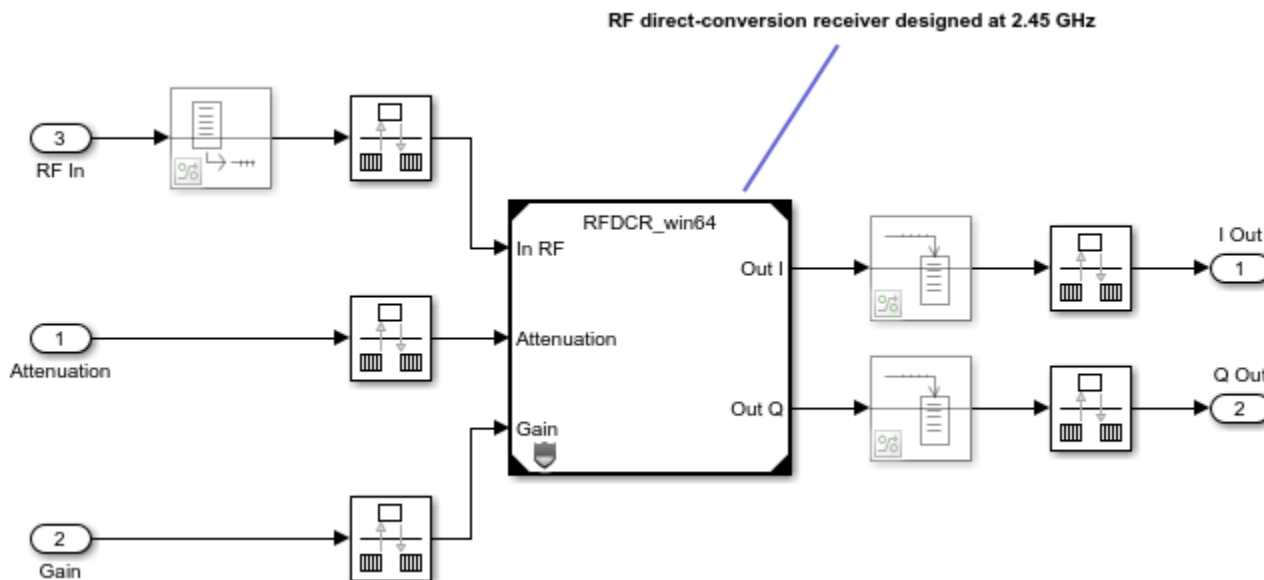
subsystemDCR = ...
    [model, '/Circuit Envelope Model of Direct Conversion Receiver'];
blockModelReference = [subsystemDCR, '/Model'];

if ismac || isunix || ispc
    open_system( ...
        [model, '/Circuit Envelope Model of Direct Conversion Receiver'])
else
    disp("Protected model in this example not supported for " + computer)
end

```

Note that the Model block references a protected model, as indicated by the shield badge on the bottom-left corner of the block. Also note that the protected model has predefined sample rates as well as fixed input and output signal dimensions. To review the protected model's interface specifications, you can right-click on the shield badge and select **Display Report** or see "Protect Circuit Envelope Model" on page 8-288.

This protected model is designed to work with framed signals of size 256 and period 8e-6 seconds at its input and output ports. Consequently, to implement baseband system designs with different frame sizes and frame rates, you might need to use `Unbuffer`, `Buffer`, and `Rate Transition` blocks to correctly interface with the referenced model for the RF system. While a full treatment of this is out of scope for this example, the model already includes these blocks, although some of them are commented out.



Use Variants to Switch Between AGC and Fixed Mode Implementations

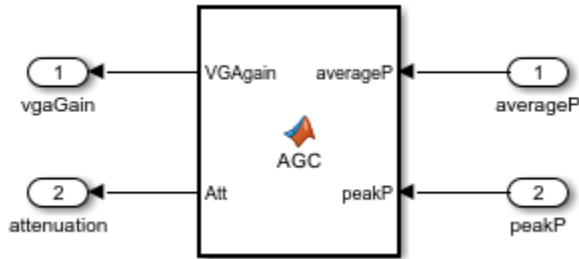
The top-level model also makes use of Simulink® variants. In general, you can use variant subsystems and Simulink variant capabilities to create flexible models with built-in variabilities to efficiently manage and compare various designs. In this example model, you can quickly enable or disable the AGC algorithm as per the following steps. First, right-click on the variant badge on the bottom-left corner of the Variant Subsystem block. Then, set **Label Mode Active Choice** as desired.

```
subsystemAGC = [model, '/Variable Gain and Variable Attenuation'];
open_system(subsystemAGC)
```



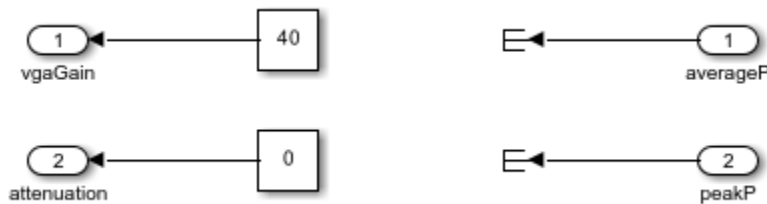
The AGC dynamically adjusts the gain and attenuation of the DCR to maximize the SNR. It amplifies the output power from the RF network to stay within the dynamic range of the ADC, and it attenuates the incoming power to the RF network to prevent saturation and overload of the RF front-end. In contrast, without the AGC the gain and attenuation are fixed and require manual retuning to maximize the SNR. In this example, the AGC maintains the variable gain between -10 dB to 40 dB. It also maintains the variable attenuation between 0 dB to 60 dB. You can enable the AGC by setting the active variant in the variant subsystem.


```
set_param(subsystemAGC, 'LabelModeActiveChoice', 'With AGC')
activeVariant = get_param(subsystemAGC, 'CompiledActiveChoiceBlock');
open_system(activeVariant)
```



When the AGC is disabled, the fixed value set for the variable gain is 40 dB, and the fixed value set for the variable attenuation is 0 dB. You can disable the AGC by setting the active variant in the variant subsystem.

```
set_param(subsystemAGC, 'LabelModeActiveChoice', 'Without AGC')
activeVariant = get_param(subsystemAGC, 'CompiledActiveChoiceBlock');
open_system(activeVariant)
```



Store Block Paths to Tune Signal and Interferer Levels and View Scopes

Store the paths to the blocks used to adjust the power levels of the transmitter and interferer. Each of these blocks applies a dB gain to its input, as specified by its **dB** parameter. At 0 dB gain, the nominal power levels for both the transmitter and the interferer are -100 dBm. You can interactively tune the dB gains from within the Simulink Editor, or you can programmatically set them.

```
blockGainTransmitter = [model, '/dB Gain Transmitter'];
blockGainInterferer = [model, '/dB Gain Interferer'];
```

Store the block paths of the scopes in the model so that you can open and examine the scopes later in this example.

```
blockScope = [model, '/Scope'];
blockRxSpectrum = [model, '/Rx Spectrum'];
```

Simulate Model with AGC for Sensitivity Level Input and Weak Interferer

As per the design of the DCR in “Design RF Direct-Conversion Receiver” on page 8-284, the minimum signal power required to operate the receiver is around -100 dBm. In the top-level system, the baseband transmitter starts at -100 dBm, which is the sensitivity level of the DCR. The interferer is also set to -100 dBm and is thus a weak interferer. As a result, the ChER is essentially determined by the noise floor. Given the low input power levels, the AGC maximizes the variable gain to 40 dB and minimizes the variable attenuation to 0 dB.

```

set_param(subsystemAGC, 'LabelModeActiveChoice', 'With AGC')
set_param(blockGainTransmitter, 'dB', '0')
set_param(blockGainInterferer, 'dB', '0')

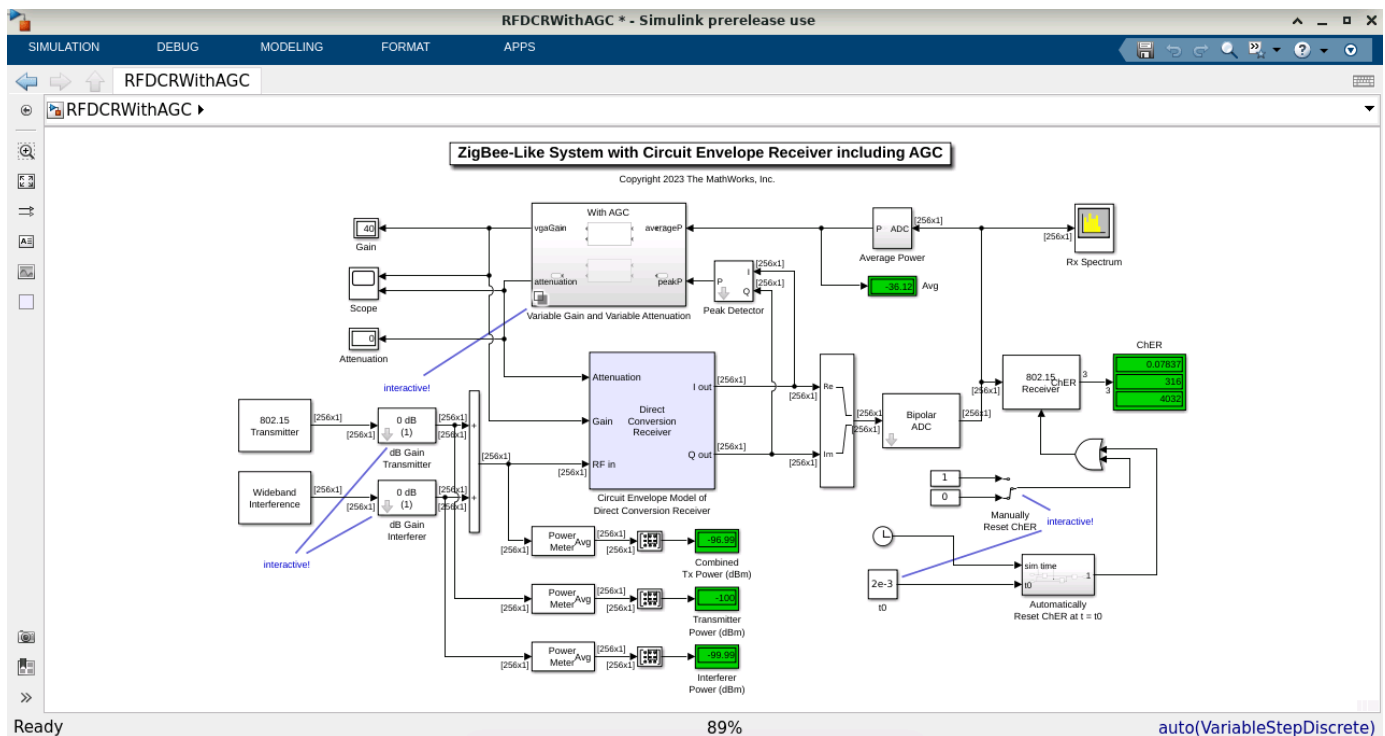
```

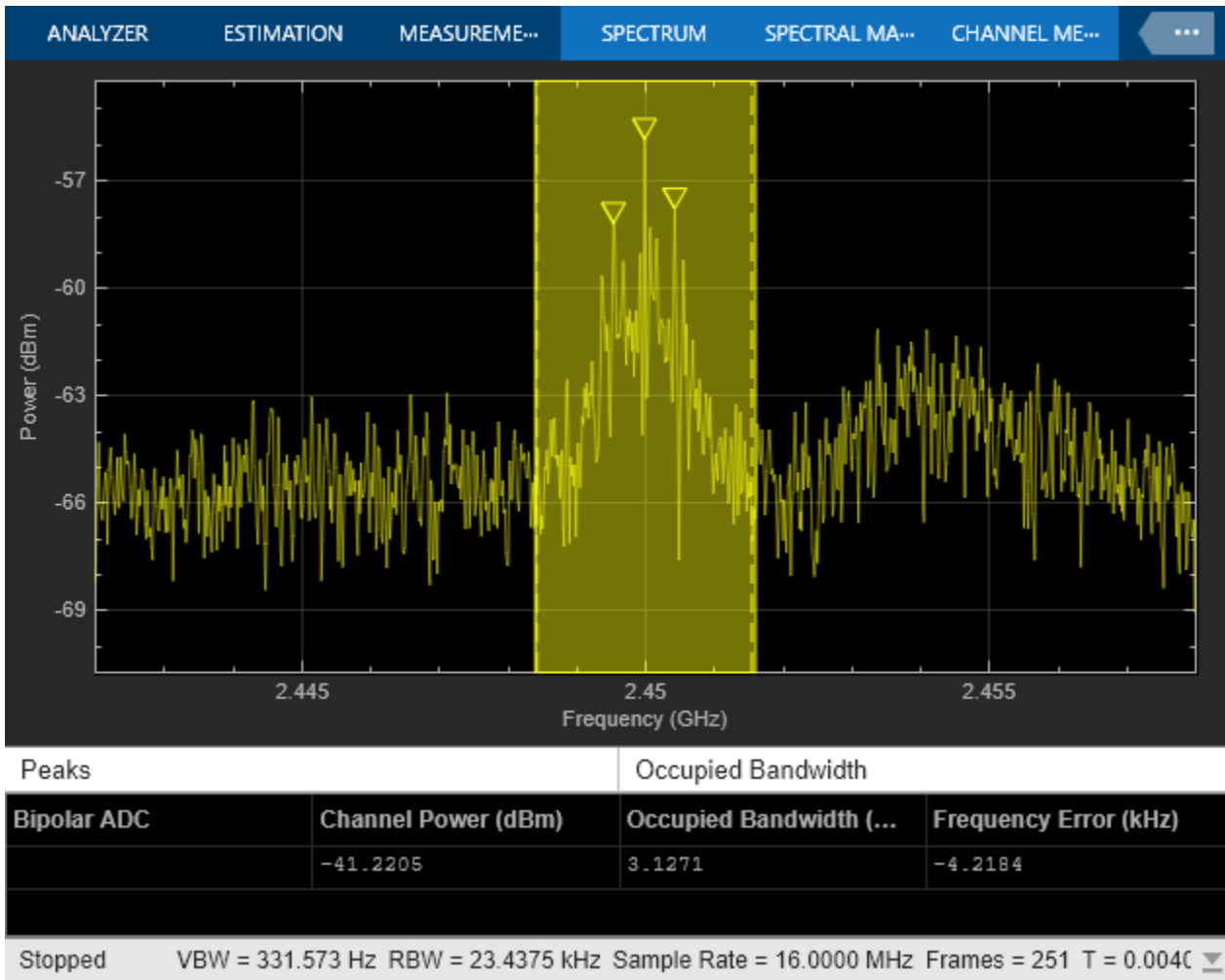
Simulate the model and examine the performance metrics, the spectrum of the received signal after the ADC, and the evolution of the variable gain and attenuation as regulated by the AGC. For your convenience, the model automatically resets the ChER metrics after 2 ms and recomputes it from 2–4 ms. This provides enough time for the AGC-controlled variable gain to stabilize, and so the final reported ChER is not affected by the startup dynamics of the model. The ChER is close to 8%.

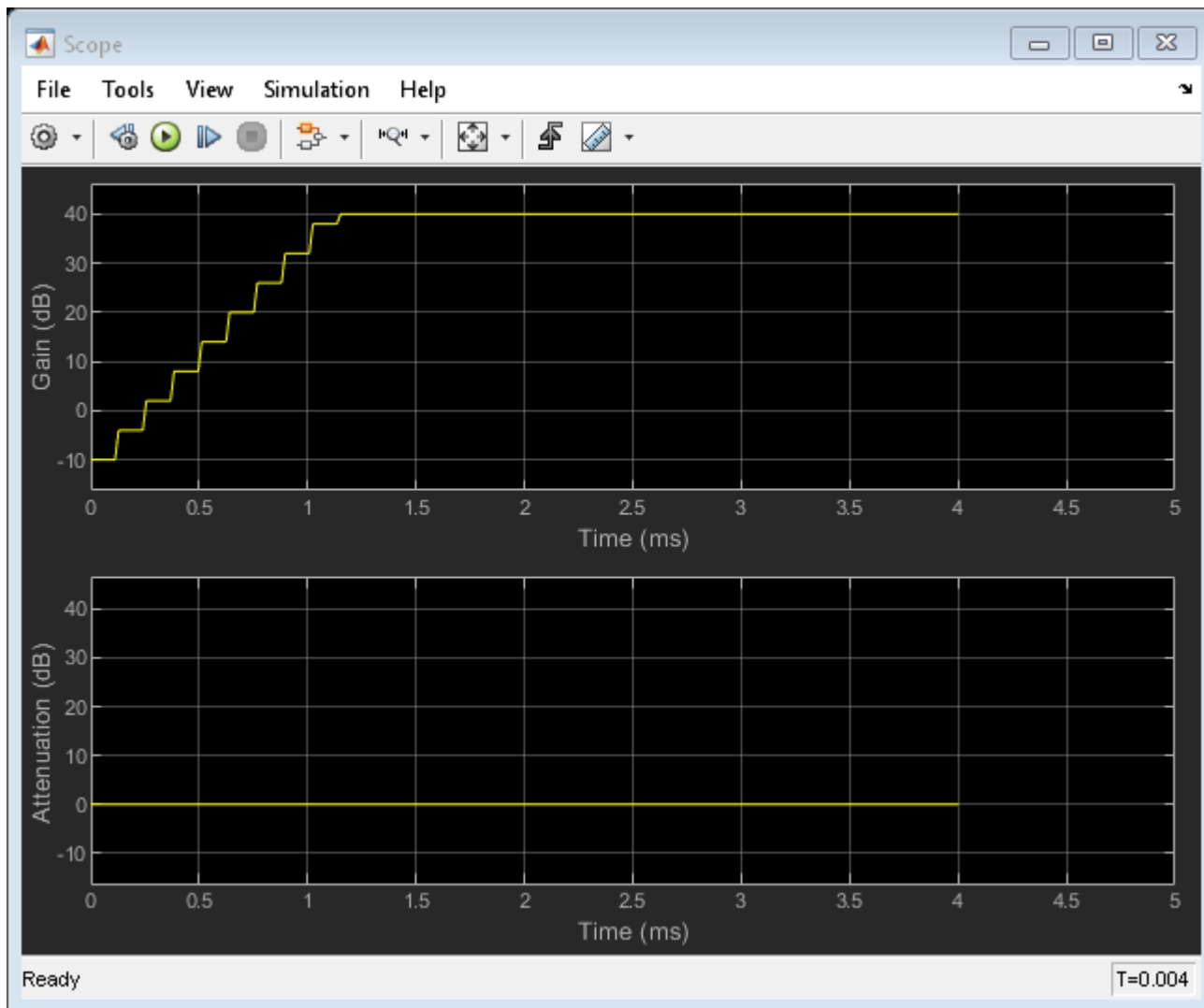
```

if ismac || isunix || ispc
    [-] = sim(model, StopTime='4E-3');
    open_system(blockRxSpectrum)
    open_system(blockScope)
else
    disp("Simulation of this example model not supported for " + computer)
end

```







Without the AGC, the variable gain stays at 40 dB, and the variable attenuation stays at 0 dB. This matches the respective levels to which the variable gain and attenuation settle in the presence of the AGC, so you can expect similar system performance with vs. without the AGC. Although this example does not explicitly show it, you can rerun this simulation with the AGC disabled to confirm the same.

Increase Input Power Above Noise Floor and Simulate Model with AGC

Increase the input power by 10 dB, which is well above the noise floor. Meanwhile, keep the interferer power the same at -100 dBm. Accordingly, the ChER is close to zero. The AGC still maximizes the variable gain to 40 dB and minimizes the variable attenuation to 0 dB. The overall input power is still far from the DCR's saturation level. The ChER is around 0.02%.

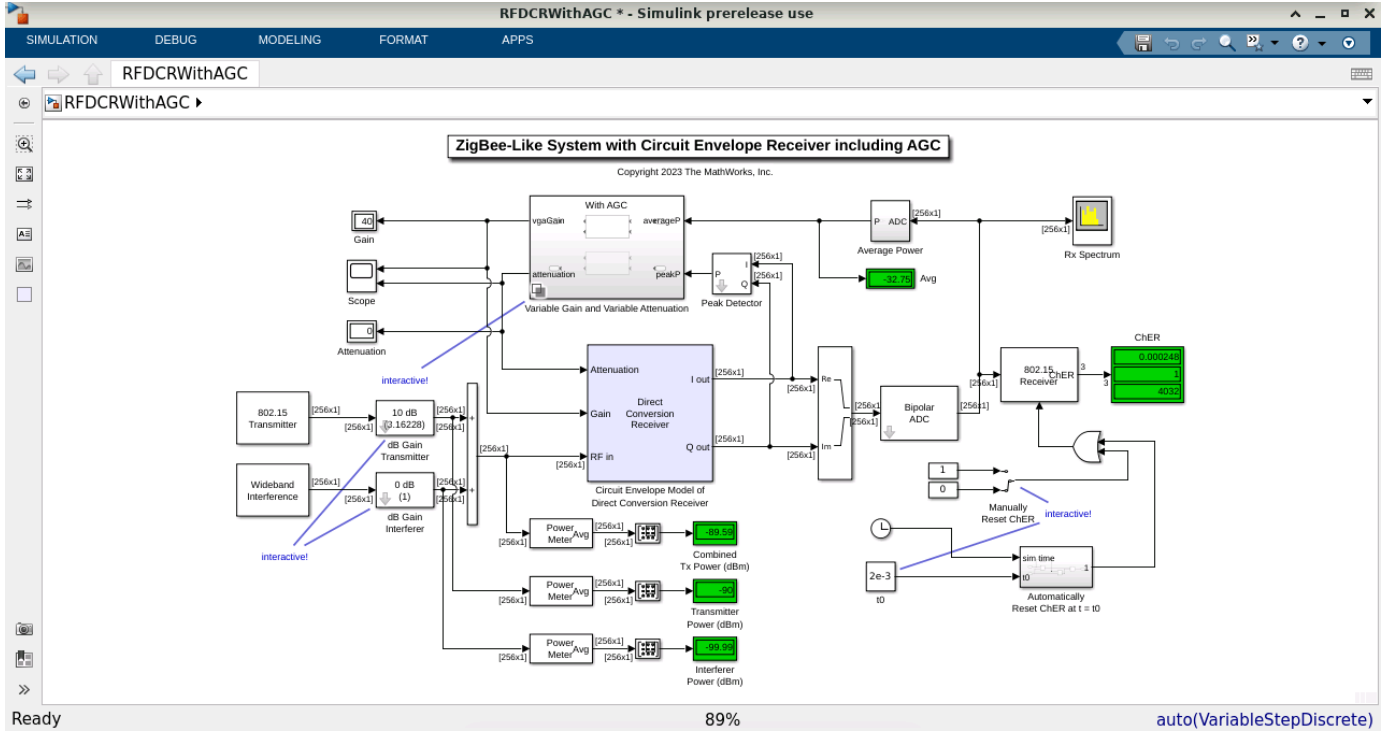
```
set_param(subsystemAGC, 'LabelModeActiveChoice', 'With AGC')
set_param(blockGainTransmitter, 'dB', '10')
set_param(blockGainInterferer, 'dB', '0')
```

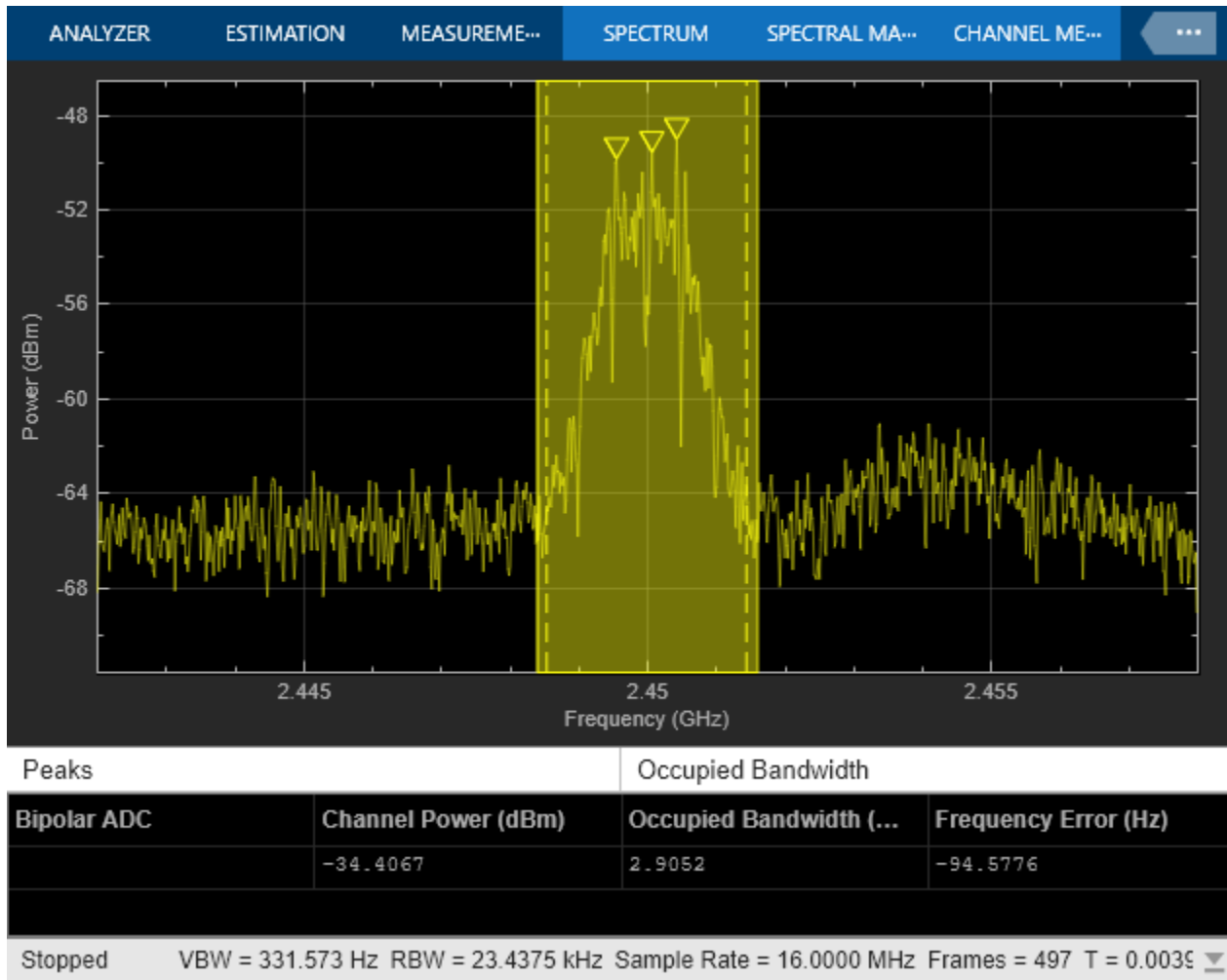
```
if ismac || isunix || ispc
    [~, ~] = sim(model, StopTime='4E-3');
    open_system(blockRxSpectrum)
```

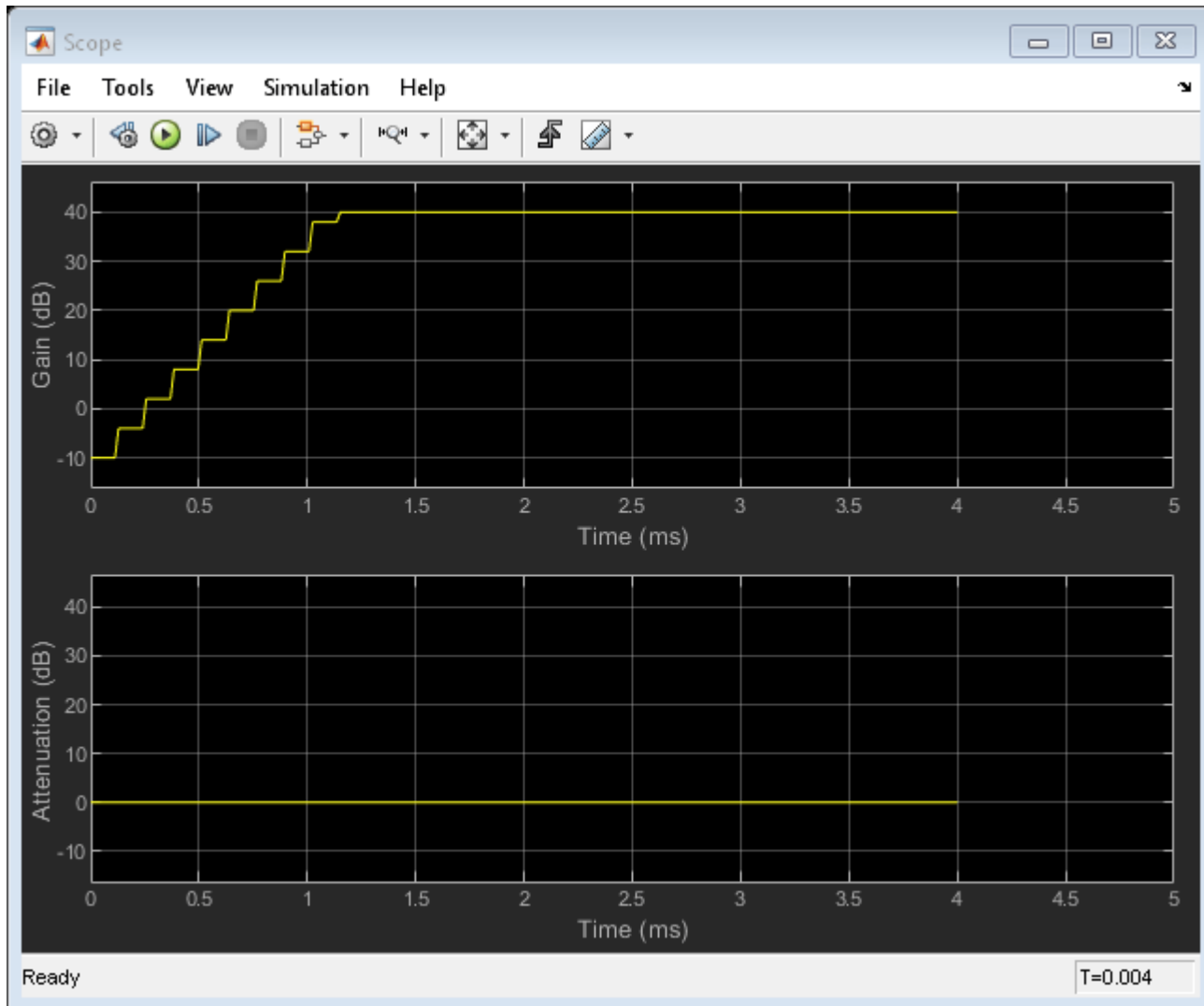
```

open_system(blockScope)
else
disp("Simulation of this example model not supported for " + computer)
end

```







Simulate Model with AGC in Presence of Strong Interferer

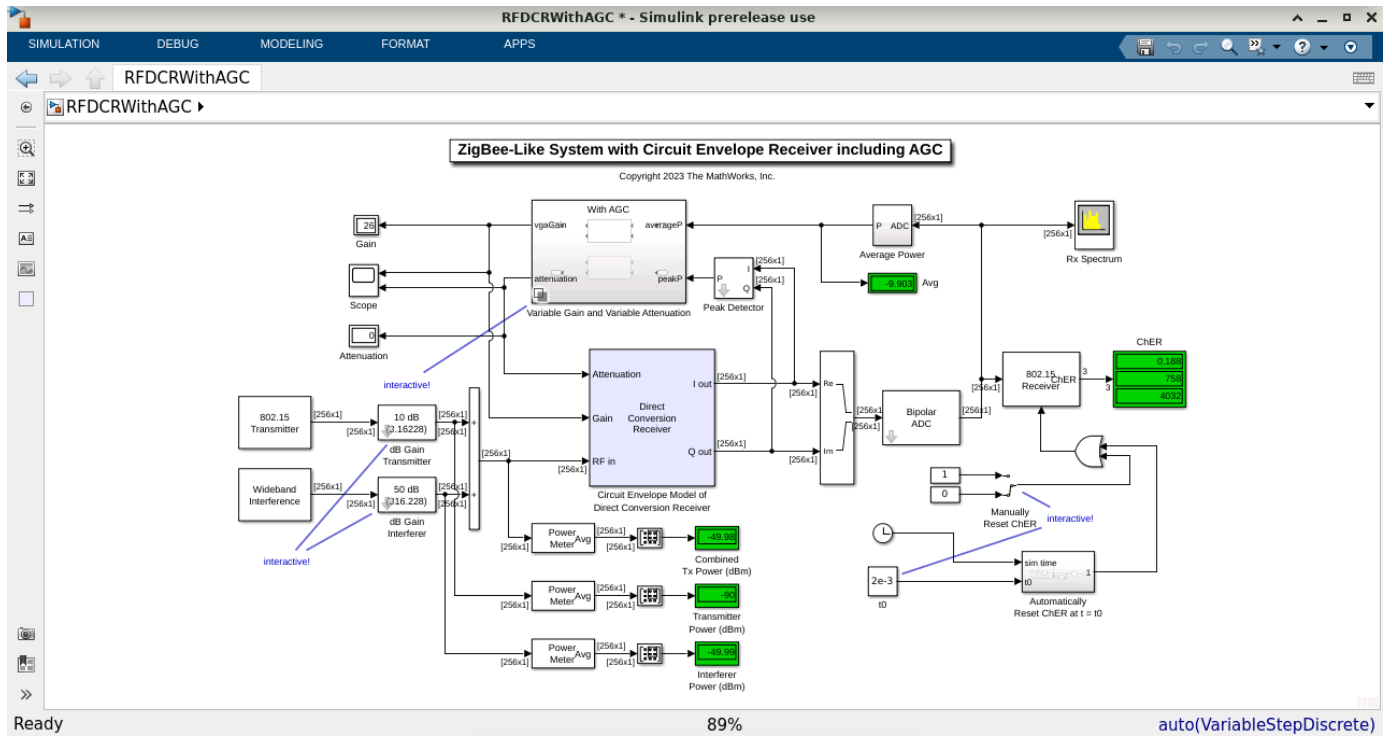
Increase the interferer power by 50 dB, thus creating substantial in-band blocking, capable of driving the DCR into saturation. In response, the AGC reduces the variable gain to 26 dB, lower than the maximum of 40 dB. The ChER worsens to 19%, but the degradation is mitigated due to the AGC.

```

set_param(subsystemAGC, 'LabelModeActiveChoice', 'With AGC')
set_param(blockGainTransmitter, 'dB', '10')
set_param(blockGainInterferer, 'dB', '50')

if ismac || isunix || ispc
    [~, ~] = sim(model, StopTime='4E-3');
    open_system(blockRxSpectrum)
    open_system(blockScope)
else
    disp("Simulation of this example model not supported for " + computer)
end

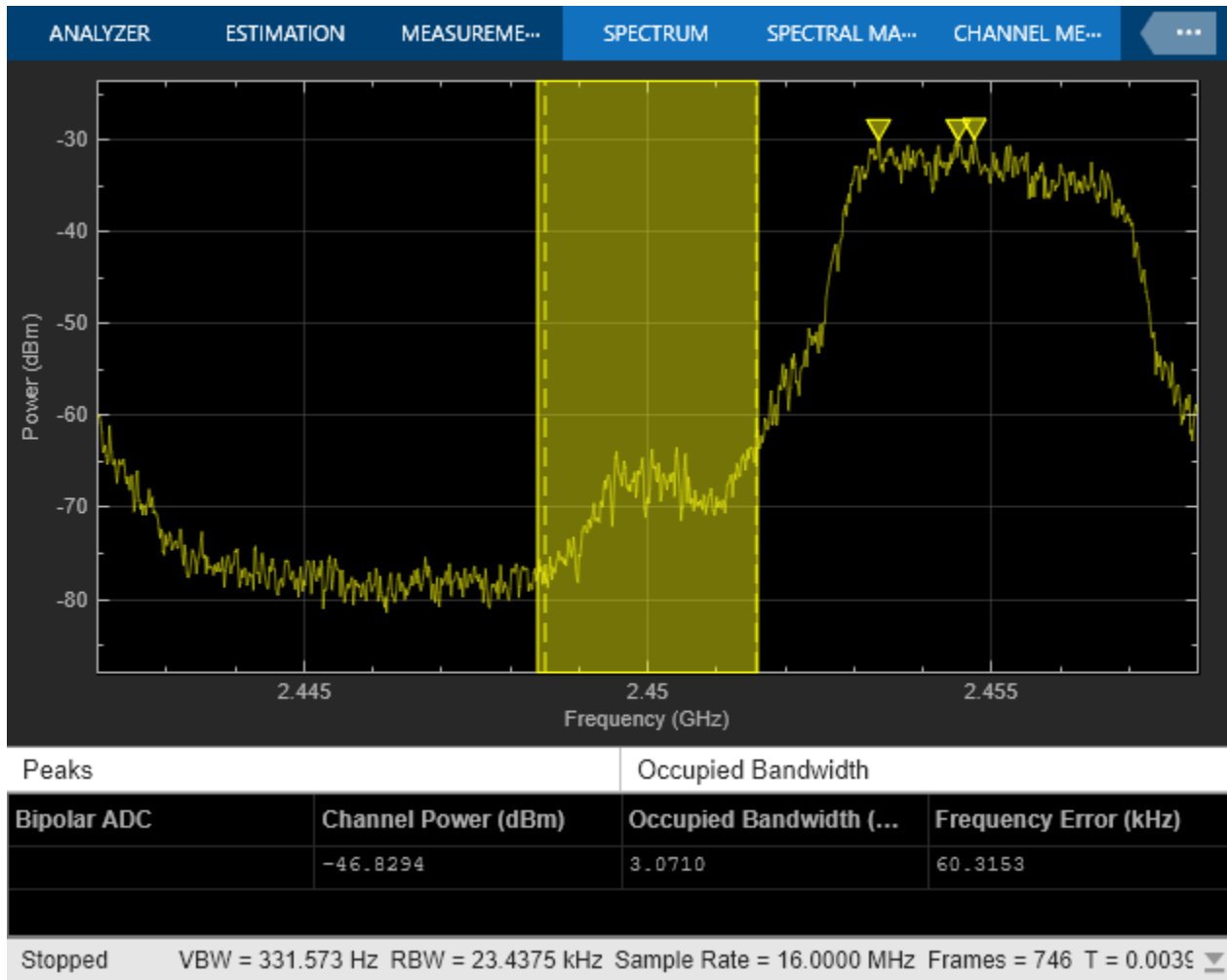
```

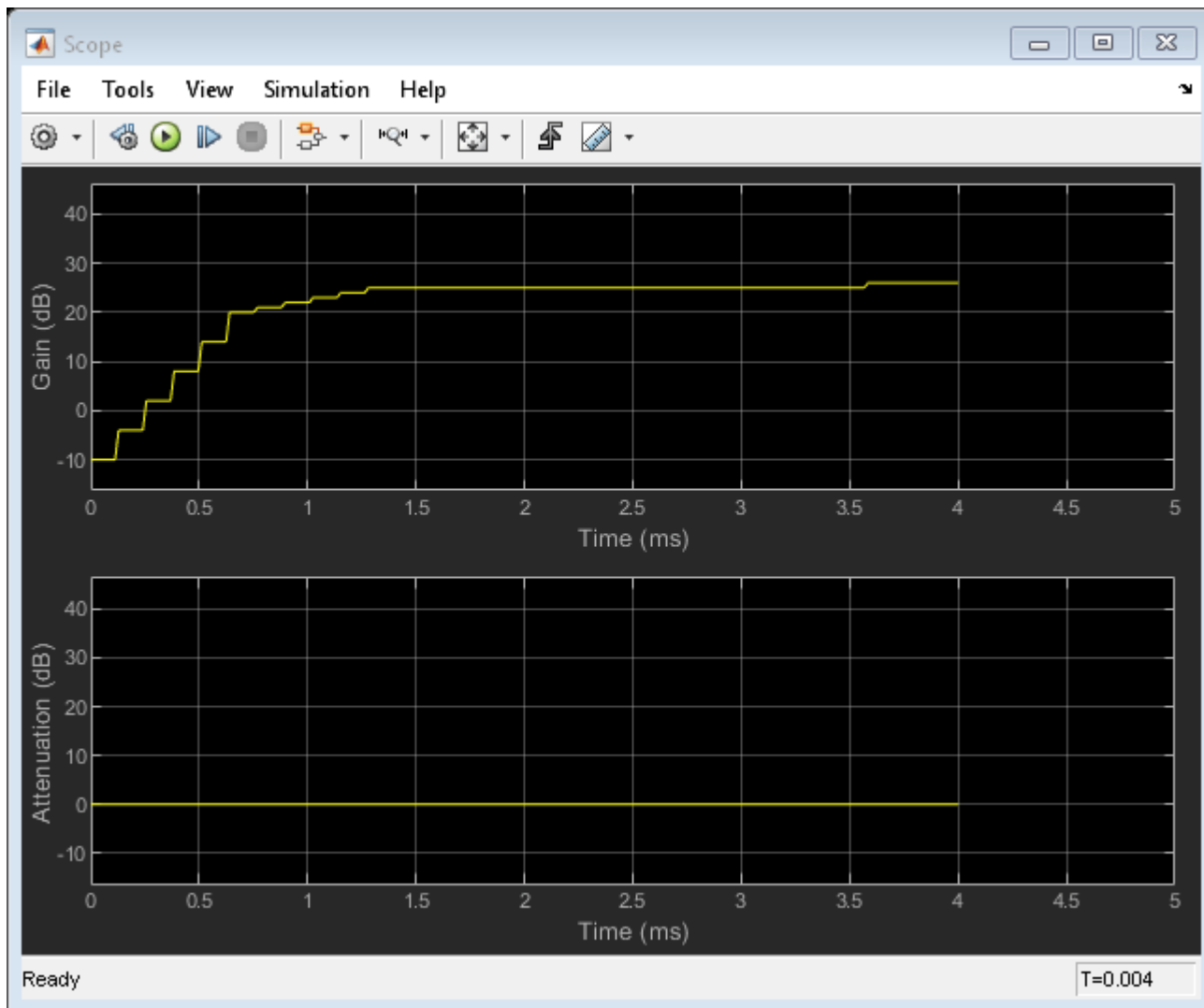


Ready

89%

auto(VariableStepDiscrete)



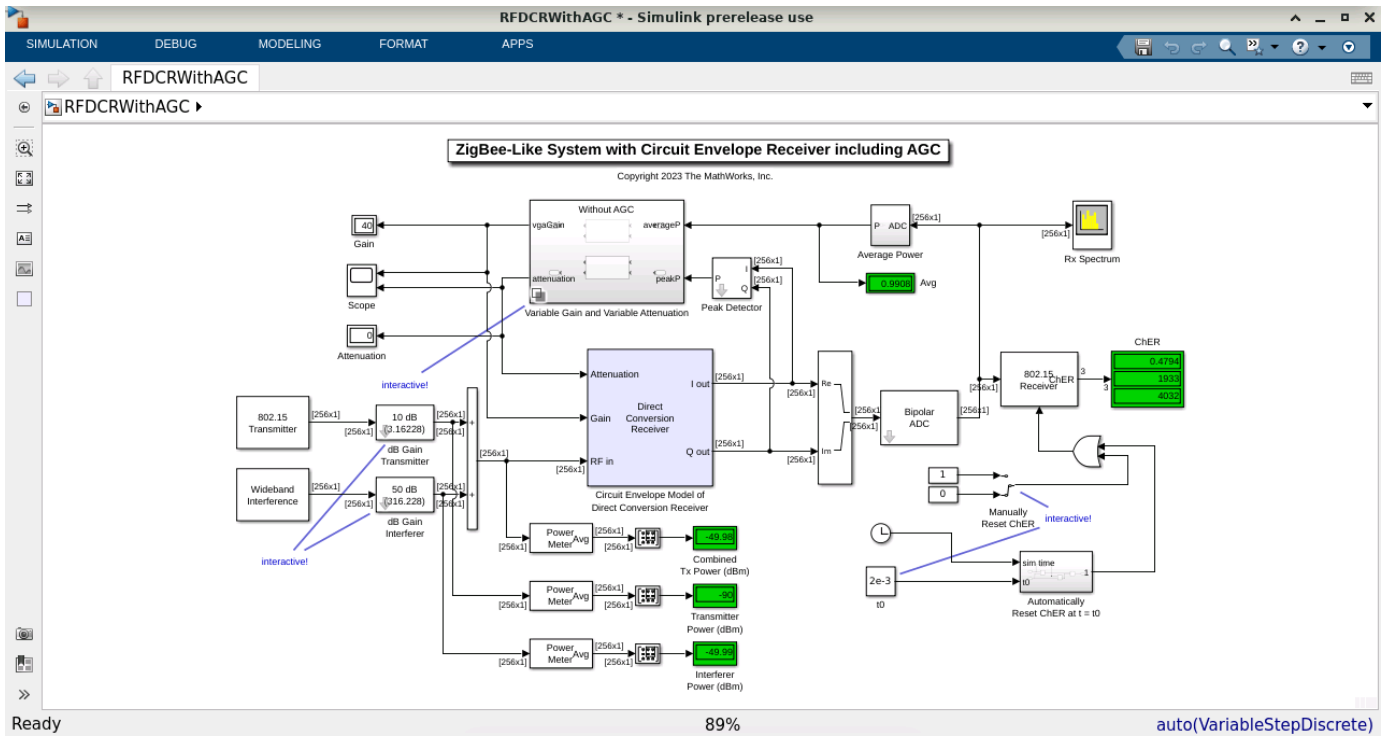


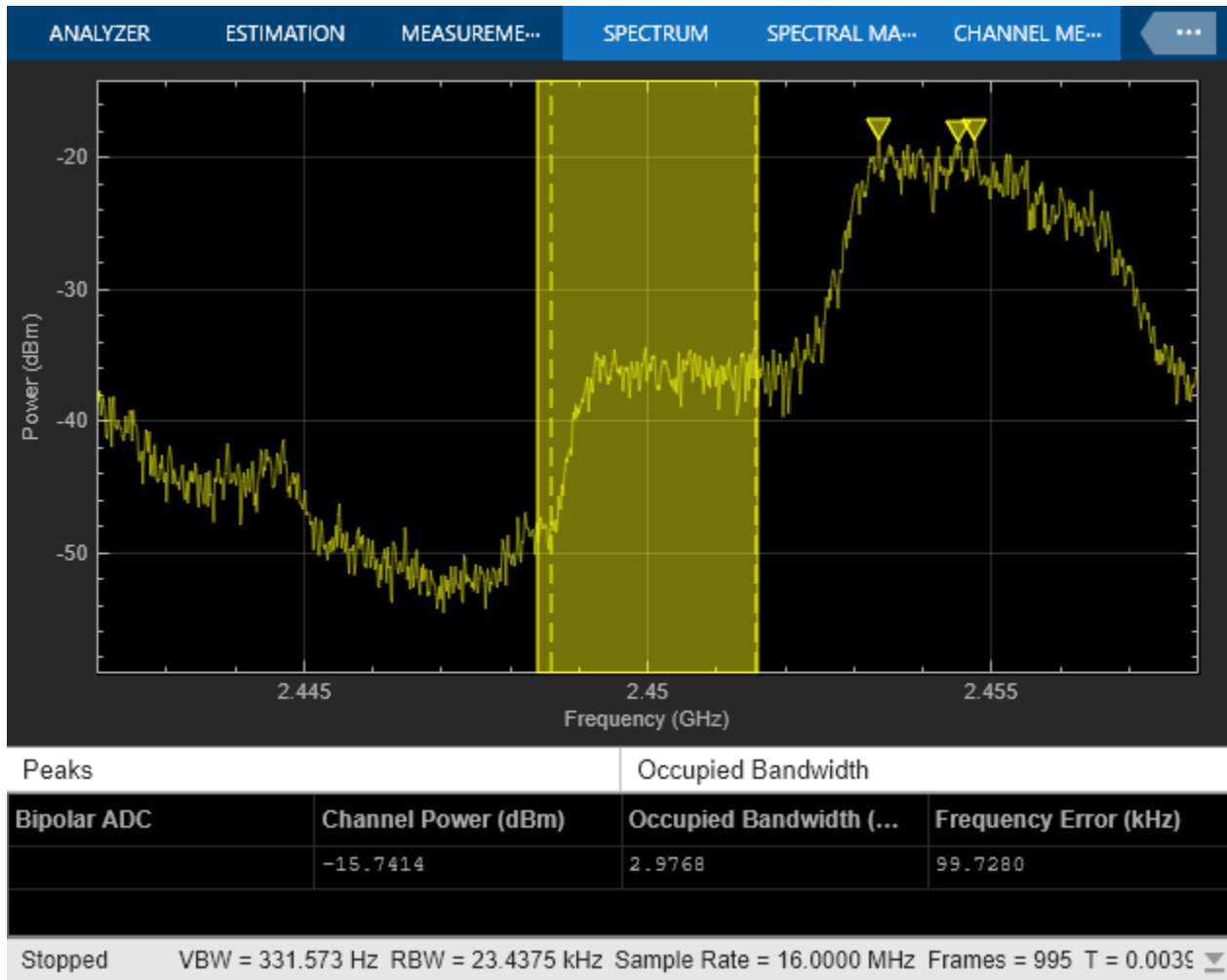
Disable AGC and Compare System Performance

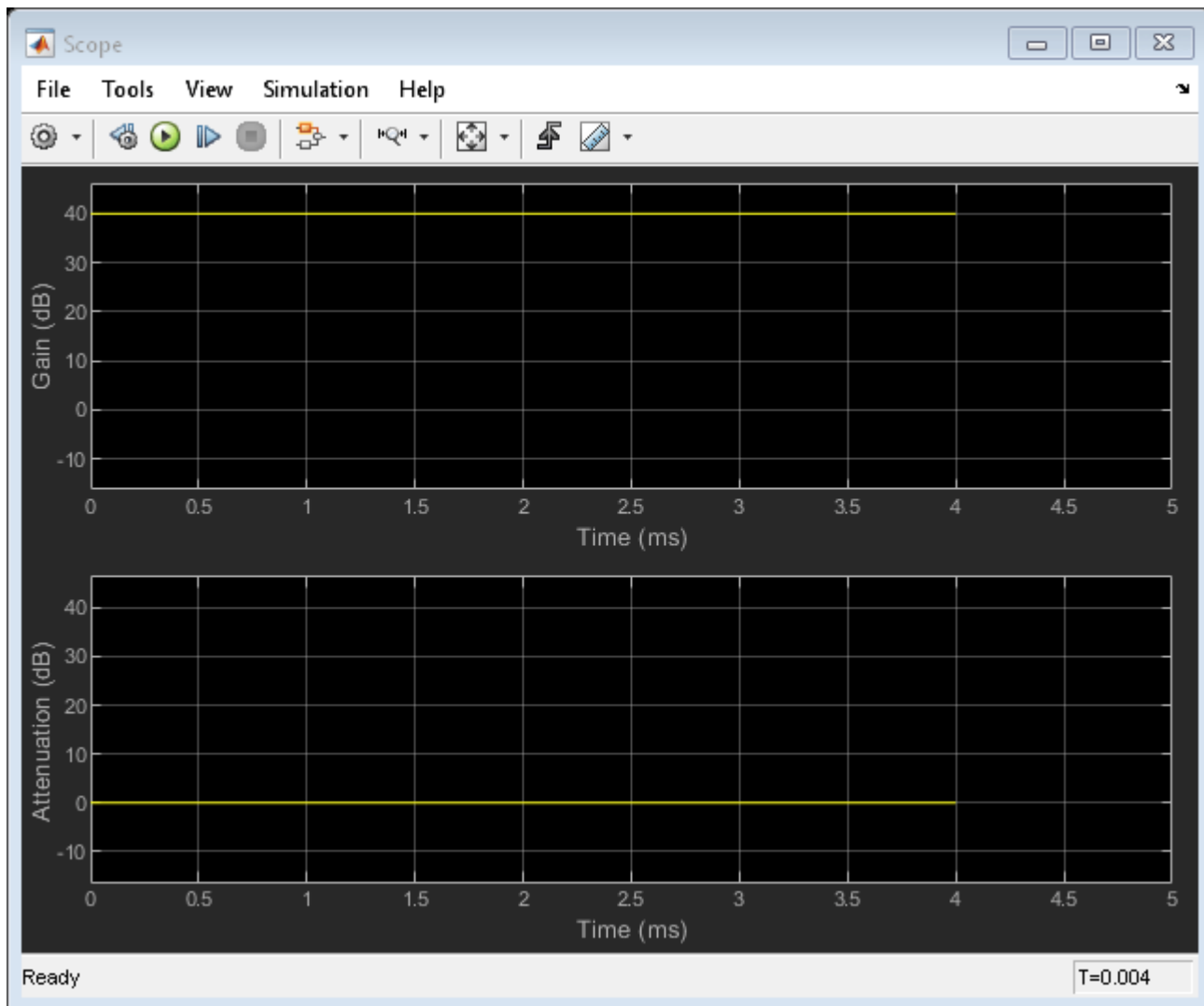
Disable the AGC and rerun the simulation. In absence of the AGC, the strong interferer drives the DCR into saturation, as the system does not have a mechanism to adapt to and avoid this situation. The variable gain remains at 40 dB, and the variable attenuation remains at 0 dB. As a result, the ChER suffers from severe degradation and hovers near 48%.

```
set_param(subsystemAGC, 'LabelModeActiveChoice', 'Without AGC')
set_param(blockGainTransmitter, 'dB', '10')
set_param(blockGainInterferer, 'dB', '50')

if ismac || isunix || ispc
    [~, ~] = sim(model, StopTime='4E-3');
    open_system(blockRxSpectrum)
    open_system(blockScope)
else
    disp("Simulation of this example model not supported for " + computer)
end
```







Cross-Product Workflow Topics

- “RF System Design for Radar and Wireless Communications” on page 9-2
- “RF Transceiver Design” on page 9-4
- “RF Noise and Nonlinearity Simulations” on page 9-6

RF System Design for Radar and Wireless Communications

RF system design in radar and wireless communications involves the integration of various RF components, such as antennas, filters, amplifiers, modulators, and demodulators. Your RF system design must consider the tradeoffs in these components to mitigate noise and intermodulation distortion effects. This topic discusses RF system design considerations and provides cross-product workflows for radar, long-term evolution (LTE), 5G, and wireless local area network (WLAN) applications.

Design Considerations

Radar

Radar applications require RF transmitters and RF receivers. RF transmitters primarily consist of an amplifier and a filter. Since the filter is a linear device and the amplifier is a nonlinear device, you can split the RF transmitter into two subsystems. This separation allows the use of different simulation frequency sets in each subsystem and permits a tradeoff between faster simulation speed and the loss of inter-stage loading effects available in a cascaded chain.

For RF receiver design you can use the direct conversion structure with LNA and matching networks. The low noise amplifier (LNA) can be described in a touchstone file and the local oscillator must include a phase noise model. As with RF transmitters, you can split RF receivers into linear and nonlinear subsystems. The linear subsystem can contain matching networks, the LNA, and the filter, while the nonlinear subsystem can contain the Mixer block and final stage amplifiers.

LTE and 5G

LTE and 5G applications require characterizing the impact of LTE interference in the RF reception of a new radio (NR) waveform. To characterize the impact of LTE interferences, you can use RF Blockset to design an RF receiver and downconvert baseband LTE and NR waveforms. Using downconverted waveforms, you can calculate the error vector magnitude (EVM), adjacent channel leakage ratio (ACLR), occupied bandwidth, channel power, and complementary cumulative distributive functions (CCDF) with LTE Toolbox and 5G Toolbox. You can use the `rfsystem` object as a device under test (DUT) to implement a measurement testbench for your RF receiver for LTE reception.

You can also characterize the impact of RF impairments such as IQ imbalance, phase noise, and PA nonlinearities on the performance of an NR RF transmitter.

WLAN

WLAN applications require characterizing the impact of RF impairments, such as in-phase and quadrature (IQ) imbalance, phase noise, and power amplifier (PA) nonlinearities in the transmission of an 802.11ax waveform. To characterize the impact of RF impairments on an 802.11ax network, you can use WLAN Toolbox™ to generate and oversample a baseband 802.11ax waveform. This waveform can be imported as an RF signal into the RF transmitter block to upconvert. Using this upconverted waveform, you can calculate the spectral mask, occupied bandwidth, channel power, CCDF, and peak-to-average power ratio (PAPR).

Design Workflows

You can design RF systems for radar and wireless communications using these cross-product workflows:

- “Radar System Modeling” on page 8-87 — This workflow shows how to set up a radar system simulation consisting of a transmitter, a channel with a target, and a receiver. RF Blockset is used for modeling the RF transmitter and receiver sections.
- “Modeling RF Front End in Radar System Simulation” (Phased Array System Toolbox) — This workflow shows how to incorporate RF front-end behavior into an existing radar system design. In a radar system, the RF front end often plays an important role in defining the system performance. For example, since the RF front end is the first section in the receiver chain, the design of its low noise amplifier is critical to achieving the desired SNR.
- “Modeling and Testing an NR RF Receiver with LTE Interference” (5G Toolbox) — This workflow shows how to characterize the impact of RF impairments in the RF reception of an NR waveform coexisting with LTE interference. The baseband waveforms are generated using 5G Toolbox and LTE Toolbox, and the RF receiver is modeled using RF Blockset.
- “Modeling and Testing an 802.11ax RF Transmitter” (WLAN Toolbox) — This workflow shows how to characterize the impact of RF impairments in an 802.11ax transmitter. The example generates a baseband IEEE® 802.11ax™ waveform by using WLAN Toolbox and models the RF transmitter by using RF Blockset.
- “RF Receiver Modeling for LTE Reception” on page 8-92 — This workflow shows how to model and test an LTE RF receiver using LTE Toolbox and RF Blockset.
- “Radar Tracking System” on page 8-132 — This workflow shows how to simulate a key multi-discipline design problem from the Aerospace Defense industry sector.

See Also

Related Examples

- “RF Transceiver Design” on page 9-4
- “RF Noise and Nonlinearity Simulations” on page 9-6

RF Transceiver Design

An RF transceiver module consists of two submodules: an RF transmitter and RF receiver. You can design an RF transceiver architecture and integrate it into your system design using RF Toolbox™ and RF Blockset. This topic discusses RF transceiver design considerations and provides workflows.

Design Considerations

You can begin the transmitter or receiver design process with a budget specification for how much gain, noise figure (NF), and nonlinearity (IP3) the entire system must satisfy. To assure the feasibility of an architecture modeled as a simple cascade of RF elements, you can calculate both the per-stage and cascade values for gain, noise figure, and IP3 (third-intercept point) using the **RF Budget Analyzer** app. For example, you can design a superheterodyne transceiver architecture in the **RF Budget Analyzer** app and export this architecture to RF Blockset for circuit envelope analysis. You can also export this cascaded architecture to RF Blockset measurement testbench as a device under test (DUT) to verify the results obtained in the **RF Budget Analyzer** app.

You can design an RF receiver using the top-down methodology. For example, using a top-down approach, you can design an RF receiver in Zigbee®-like applications to verify BER impairment models. Designing RF transceivers for mm-wave systems often requires adding hybrid-beamforming antennas to your RF transceiver modules. Using RF Blockset, Phased Array System Toolbox™, and Communications Toolbox you can include these hybrid-beamforming antennas in your mm-wave transmitter and receiver modules and analyze RF imperfections and transmit radiation effects.

You can calculate RF impairments such as component noise, interference from blocker signals, LO phase noise, the dynamic range of the analog-to-digital convertor, and component mismatch in a low IF architecture using RF Blockset Circuit Envelope simulations. Using Communication Toolbox and RF Blockset, you can also integrate an RF receiver with baseband signal processing algorithms to model end-to-end communication systems.

You can also use RF Blockset Analog Devices support software models “AD9361 Models” on page 4-2 and “AD9371 Models” on page 4-9 to simulate and verify agile RF transceiver designs. MathWorks® and Analog Devices co-developed the models and validated the values using lab measurements.

Design Workflows

You can design an RF transceiver using these cross-product workflows:

- “Superheterodyne Receiver Using RF Budget Analyzer App” — This workflow shows how to build a superheterodyne receiver and analyze the RF budget of the receiver for gain, noise figure, and IP3 using the **RF Budget Analyzer** app.
- “Top-Down Design of an RF Receiver” on page 8-166 — This workflow designs an RF receiver for a ZigBee-like application using a top-down methodology. It verifies the BER of an impairment-free design, then analyzes BER performance after the addition of impairment models. The example uses the **RF Budget Analyzer** app to rank the elements contributing to the noise and nonlinearity budget.
- “Modeling RF mmWave Transmitter with Hybrid Beamforming” on page 8-149 — This workflow illustrates a methodology for system-level modeling and simulation of a 66 GHz QPSK RF transmit and receive system with a 32-element hybrid beamforming antenna. The system includes RF imperfections, transmit array radiation effects, a narrowband receive array, and a baseband receiver with corrections for system impairments and message decoding. The antenna

beamforming direction is defined using azimuth and elevation angles and it is estimated in the RF receive antenna using the root-MUSIC DOA algorithm.

- “Architectural Design of a Low IF Receiver System” on page 8-178 — This workflow shows how to use the RF Blockset Circuit Envelope library to simulate the performance of a low-IF architecture with RF impairments.
- “Communications System with Embedded RF Receiver” on page 8-13 — This workflow shows how to integrate an RF receiver with baseband signal processing algorithms to model an end-to-end communications system.

See Also

Related Examples

- “RF Noise and Nonlinearity Simulations” on page 9-6
- “RF System Design for Radar and Wireless Communications” on page 9-2

RF Noise and Nonlinearity Simulations

Excess noise and nonlinearities in amplifiers and mixers can degrade RF system performance. You can design a robust RF system by simulating noise and nonlinearities in amplifiers and mixers to determine the optimal noise and nonlinearities in your RF system. This topic discusses power amplifier (PA) characterization and spot noise measurements using Circuit Envelope library blocks as well as how you can use Idealized Baseband library blocks to simulate noise and nonlinearities in your RF systems.

PA Characterizations and Spot Noise Measurements

PA in RF transmitters is not immune to noise and other nonlinearities. To understand the effects of these impairments you must characterize your power amplifiers. Characterization of power amplifiers involves simulating and measuring AM/AM and plotting gain against the input power data. Characterization reveals the linearity of your PA for the input signal given to the system. In addition to nonlinear gain, you can also simulate the memory effect of the PA using the memory polynomial model. The memory polynomial model yields complex coefficients of PA. With these coefficients, you can perform fit and calculate root mean squared (RMS) errors. With this fitted data, you can visualize both fitted and measured output signals. You can also use the memoryless nonlinearity model to study the effects of amplitude and phase distortion. In a transmitter, to offset the effects of nonlinearities in the power amplifier you can perform a digital predistortion.

Spot noise parameters such as the noise factor, optimum reflection coefficient, and resistor noise help you to describe the noise introduced by a 2-port device. These parameters along with the source impedance Z_s uniquely determine the measured noise figure of the device. You can use noise circles plotted on a Smith chart generated by a Noise Figure Testbench block to show an interaction between Z_s and the noise figure.

Idealized Baseband Simulations

The RF Blockset™ Idealized Baseband library extends your Simulink® environment with a library of blocks that model single-carrier complex-baseband systems with discrete-time signals. You can directly couple blocks from this library with DSP System Toolbox™ blocks or other Simulink blocks to estimate the impact of RF phenomena on overall system performance.

Idealized Baseband Amplifier and Mixer blocks can be used to simulate nonlinearities and noise in your RF system design. The Amplifier block provides four nonlinearity models: cubic polynomial, AM/AM-AM/PM, Saleh, and modified Rapp. Both the Amplifier and Mixer blocks provide three options to represent noise: noise temperature, noise factor, and noise figure. You can also visualize the power characteristics and noise characteristics of your systems using these blocks.

Simulation Workflows

You can simulate RF nonlinearities and noise in your system using these cross-product workflows:

- “Power Amplifier Characterization” on page 7-92 — This workflow shows how to characterize a power amplifier (PA) using measured input and output signals of an NXP Airfast PA. Optionally, you can use a hardware test setup including an NI PXI chassis with a vector signal transceiver (VST) to measure the signals at runtime.
- “Digital Predistortion to Compensate for Power Amplifier Nonlinearities” on page 8-78 — This workflow shows how to use digital predistortion (DPD) in a transmitter to offset the effects of

nonlinearities in a power amplifier. by sending two tones and 5G-like OFDM waveform with a bandwidth of 100 MHz to a RF transmitter.

- “Spot Noise Data in Amplifiers and Effects on Measured Noise Figure” on page 7-16 — This workflow shows a testbench model to describe the noise introduced by a 2-port device. The spot noise data parameters, minimum noise figure, reflection coefficient, and resistor noise fully describe the noise introduced by a 2-port device. These parameters along with the source impedance Z_s uniquely determine the measured noise figure of the device. You can use noise circles plotted on a Smith chart to show an interaction between Z_s and the noise figure.
- “Idealized Baseband Amplifier with Nonlinearity and Noise” on page 7-72 — This workflow shows how to use the Amplifier block from the Idealized Baseband library to amplify a signal with nonlinearity and noise. The Amplifier block uses the cubic polynomial model with a linear power gain of 10 dB, an input IP3 nonlinearity of 30 dBm, and a noise figure of 3 dB.
- “Modulate Quadrature Baseband Signals Using IQ Modulators” on page 7-104 — This workflow shows how to modulate quadrature baseband signals using two different RF Blockset blocks. You can use a Mixer block from the Idealized Baseband library or a circuit envelope IQ Modulator block in your model to modulate quadrature baseband signals to the RF level. Observe the impairments in the modulated output signal due to gain imbalance, third-order intercepts (OIP3), and system noise in the complex output power density and output power spectrum analyzers.
- “RF Budget Harmonic Balance Analysis of Low-IF Receiver, IP2 and NF” — This workflow shows how to use the harmonic balance solver of the `rfbudget` object to analyze a low intermediate frequency (low-IF) receiver RF budget for the second-order intercept point (IP2), and to compute a more accurate noise figure (NF) that correctly accounts for system nonlinearity and noise folding.

See Also

Related Examples

- “RF Transceiver Design” on page 9-4
- “RF System Design for Radar and Wireless Communications” on page 9-2

