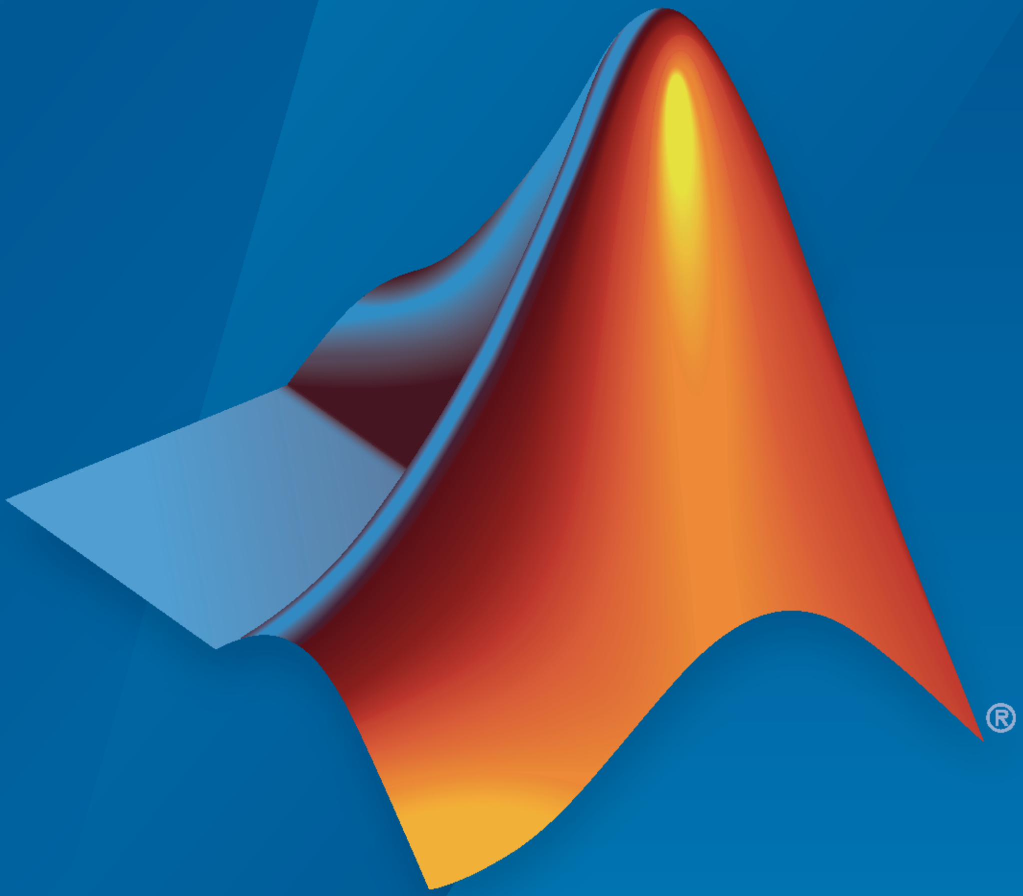
Wireless Testbench™

Getting Started Guide



MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Wireless Testbench™ Getting Started Guide

Revision History

March 2022	Online only	New for Version 1.0 (Release 2022a)
September 2022	Online only	Revised for Version 1.1 (Release 2022b)
March 2023	Online only	Revised for Version 1.2 (Release 2023a)

Contents

Product Overview

Wireless Testbench Product Description

Test wideband wireless systems and perform spectrum monitoring

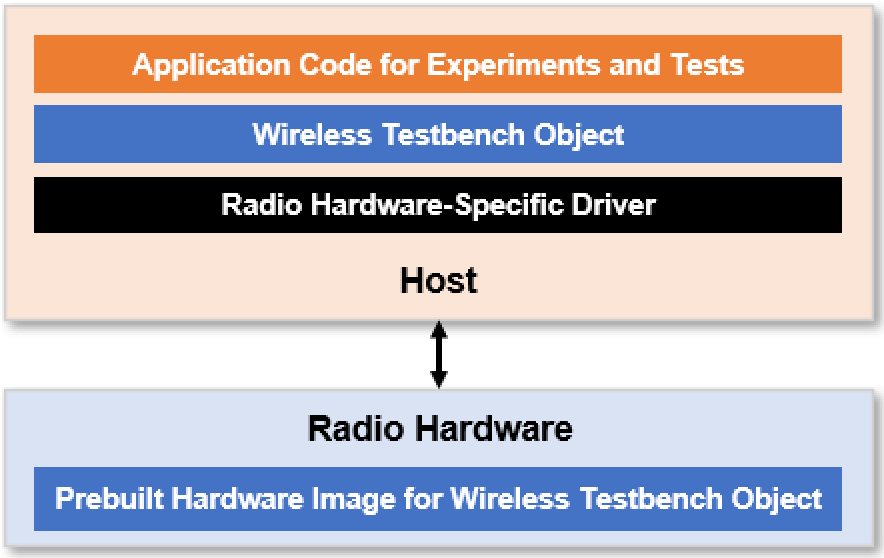
Wireless Testbench enables you to test wideband wireless systems using MATLAB® with a software-defined radio (SDR). You can use your SDR to perform spectrum monitoring with high-speed data capture.

The product provides capabilities such as intelligent signal capture and hardware-based resampling that leverage the FPGA hardware on the SDR. You can specify waveform-specific characteristics to trigger signal capture and analyze the data of interest. You can also transmit and capture custom signals at arbitrary sample rates or standards-based signals (5G and WLAN) at their native sample rates.

About Wireless Testbench Applications Using SDR

Wireless Testbench Applications on SDR

Wireless Testbench provides prebuilt hardware images that enable you to explore wireless applications for software-defined-radio (SDR) hardware by using MATLAB objects. This diagram is a high-level overview of how Wireless Testbench objects integrate SDR capabilities of supported radios.


Application Code for Experiments and Tests
Wireless Testbench Object
Radio Hardware-Specific Driver
Host
Radio Hardware
Prebuilt Hardware Image for Wireless Testbench Object

Using a Wireless Testbench object, you can configure the prebuilt hardware image to transmit, capture, or detect wireless signals and write your application code for experimenting and testing.

Wireless Testbench Object	Description
`basebandReceiver`	Configure SDR as baseband receiver
`basebandTransceiver`	Configure SDR as baseband transceiver
`basebandTransmitter`	Configure SDR as baseband transmitter
`preambleDetector`	Configure SDR as preamble detector

See Also

More About

- "Supported Radio Devices" on page 2-3

Supported Radio Devices

The Wireless Testbench software provides radio support through dedicated hardware support packages. To use a supported radio with a Wireless Testbench feature, you must install the associated support package from the Add-On Explorer.

Supported NI USRP Radios

Wireless Testbench Support Package for NI™ USRP™ Radios provides support for these radios.

USRP Networked Series

- USRP N310
- USRP N320
- USRP N321

USRP X Series

- USRP X310 with UBX 160 daughterboard
  - The USRP X310 radio support also enables you to use the equivalent USRP NI–2944R and USRP NI–2954R radios with Wireless Testbench. For more information on how to convert these radios into an equivalent X310 radio, see Running UHD and GNU Radio on NI USRP-RIO on the hardware vendor website.
- USRP X410

See Also

More About

- “Install Support Package for NI USRP Radios”
- “Connect and Set Up NI USRP Radios”

Tutorials

Capture from Frequency Band

This example shows how to configure a software-defined radio (SDR) as a baseband receiver to capture data from a specified frequency band. The example also plots the frequency spectrum of the captured data.

Set Up Radio

Call the `radioConfigurations` function. The function returns all available radio setup configurations that you saved using the Radio Setup wizard. For more information, see "Connect and Set Up NI USRP Radios".

```
savedRadioConfigurations = radioConfigurations;
```

To update the dropdown menu with your saved radio setup configuration names, click **Update**. Then select the radio to use with this example.

```
savedRadioConfigurationNames = [string({savedRadioConfigurations.Name})];
radio = myRadio ▾ Update ;
```

Specify Frequency Band

Specify the start and the end of the frequency band. By default, this example captures the 87.5-108 MHz frequency band, typically allocated to FM radio.

```
frequencyBand.Start = 87500000 ;
frequencyBand.End = 108000000 ;
frequencyBand.Width = frequencyBand.End-frequencyBand.Start;
frequencyBand.MidPoint = frequencyBand.Start + frequencyBand.Width/2;
```

Configure Baseband Receiver

Create a baseband receiver object with the specified radio. Because the object requires exclusive access to radio hardware resources, before running this example for the first time, clear any other object associated with the specified radio. In subsequent runs, to speed up the execution time of the example, reuse your new workspace object.

```
if ~exist("bbrx","var")
    bbrx = basebandReceiver(radio);
end
```

To capture the full width of the frequency band:

- Set the `SampleRate` property to a value that is greater than or equal to the width of the frequency band.
- Set the `CenterFrequency` property to the value that corresponds to the middle of the frequency band.
- Set the `RadioGain` property according to the local signal strength.

```
bbrx.SampleRate = frequencyBand.Width ;
bbrx.CenterFrequency = frequencyBand.MidPoint;
bbrx.RadioGain = 30 ;
```

To update the dropdown menu with the antennas available for your radio, call the `hCaptureAntennas` helper function. Then select the antenna to use with this example.

```
antennaSelection = hCaptureAntennas(radio);
bbrx.Antennas = RF0:TX/RX;
```

Capture IQ Data

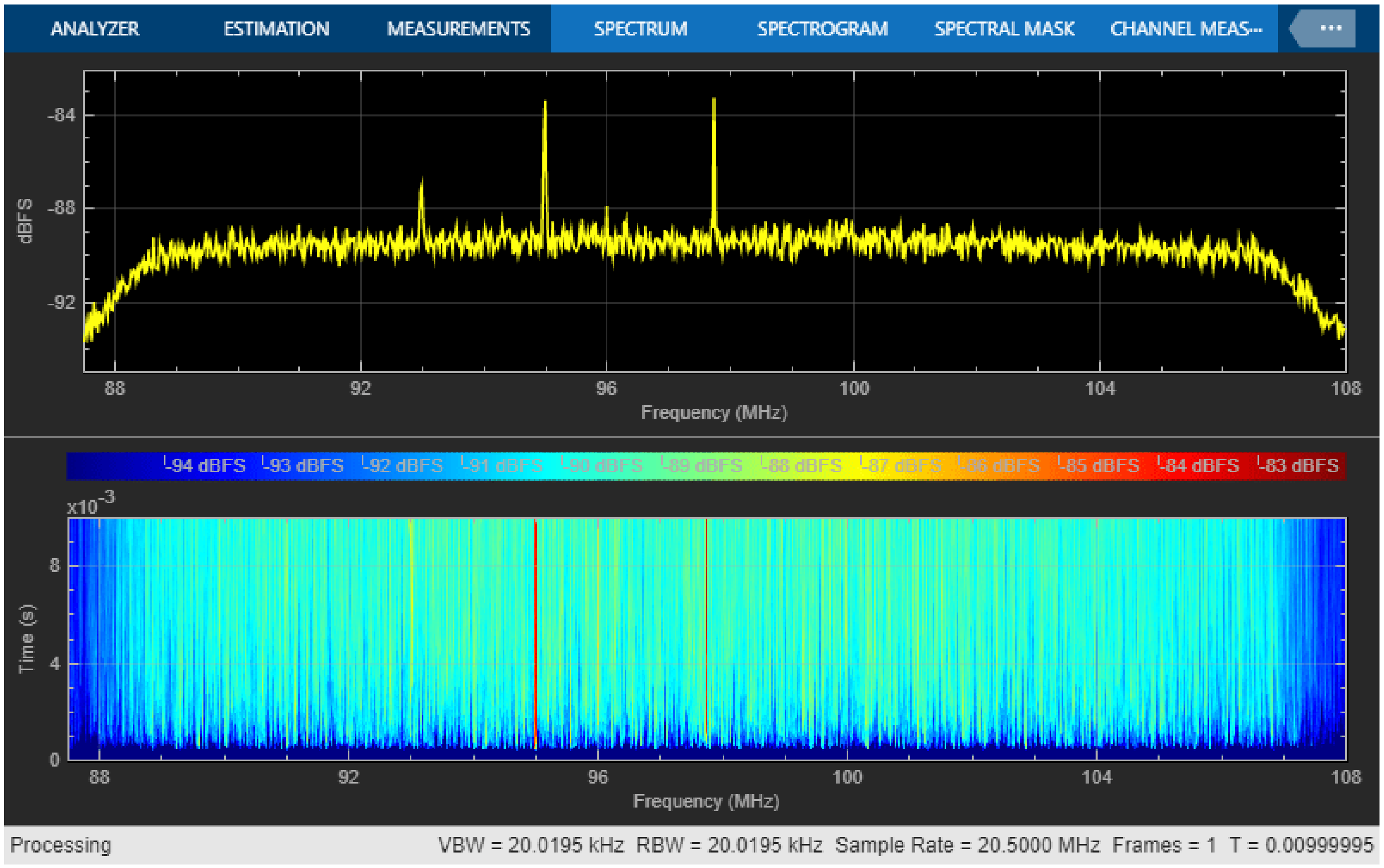
To capture IQ data from the specified frequency band, call the `capture` function on the baseband receiver object. Specify the length of the capture.

```
captureLength = milliseconds(10);
data = capture(bbrx,captureLength);
```

Plot Spectrum of Captured Data

Create a `spectrumAnalyzer` object. To speed up the execution time of this example upon subsequent runs, reuse the spectrum analyzer object. Set the sample rate of the spectrum analyzer object to the sample rate of the baseband receiver object. Plot the spectrum and spectrogram of the captured data.

```
if ~exist("spectrumScope","var")
    spectrumScope = spectrumAnalyzer;
end
spectrumScope.SampleRate = bbrx.SampleRate;
spectrumScope.ChannelNames = bbrx.Antennas;
spectrumScope.FrequencyOffset = bbrx.CenterFrequency;
spectrumScope.ViewType = "Spectrum and spectrogram";
spectrumScope.TimeSpanSource = "Property";
spectrumScope.TimeSpan = seconds(captureLength);
spectrumScope.SpectrumUnits = "dBFS";
spectrumScope.FullScaleSource = "Property";
spectrumScope.FullScale = double(intmax('int16'));
spectrumScope(data);
spectrumScope.show;
release(spectrumScope);
```



Processing VBW = 20.0195 kHz RBW = 20.0195 kHz Sample Rate = 20.5000 MHz Frames = 1 T = 0.00999995

To call the capture function again and to update the spectrum analyzer by rerunning the current section, click **Capture and plot frequency spectrum**.

Capture and plot frequency spectrum

See Also

Functions
`radioConfigurations`

Objects
`basebandReceiver`

More About

- “Calibrate Radio Gain For Signal Capture”
- “Capture Wideband Spectrum by Combining Data from Multiple Antennas”
- “Supported Radio Devices” on page 2-3

Transmit Waveform

This example shows how to configure a software-defined radio (SDR) as a baseband transmitter to transmit a custom generated wireless waveform.

Set Up Radio

Call the `radioConfigurations` function. The function returns all available radio setup configurations that you saved using the Radio Setup wizard. For more information, see “Connect and Set Up NI USRP Radios”.

```
savedRadioConfigurations = radioConfigurations;
```

To update the dropdown menu with your saved radio setup configuration names, click **Update**. Then select the radio to use with this example.

```
savedRadioConfigurationNames = [string({savedRadioConfigurations.Name})];
radio = myRadio Update;
```

Specify Wireless Waveform

Use the attached `QAM-4-GeneratedWaveform.mat` file to specify the transmit waveform. The `waveStruct` structure in this file contains a QAM-4 waveform that is generated by using the Wireless Waveform Generator app.

```
load("QAM-4-GeneratedWaveform.mat")
```

Configure Baseband Transmitter

Create a baseband transmitter object with the specified radio. Because the object requires exclusive access to radio hardware resources, before running this example for the first time, clear any other object associated with the specified radio. In subsequent runs, to speed up the execution time of the example, reuse your new workspace object.

```
if ~exist("bbtx","var")
    bbtx = basebandTransmitter(radio);
end
```

Configure the baseband transmitter object according to the parameters of the wireless waveform.

- Set the `SampleRate` property to the sample rate of the generated waveform.
- Set the `CenterFrequency` property to a value in the frequency spectrum indicating the position of the waveform transmission.

```
bbtx.SampleRate = waveStruct.Fs;
bbtx.CenterFrequency = 2.4e9;
```

Set the `RadioGain` property according to the local wireless channel. Specify the transmit antennas.

```
bbtx.RadioGain = 30;
```

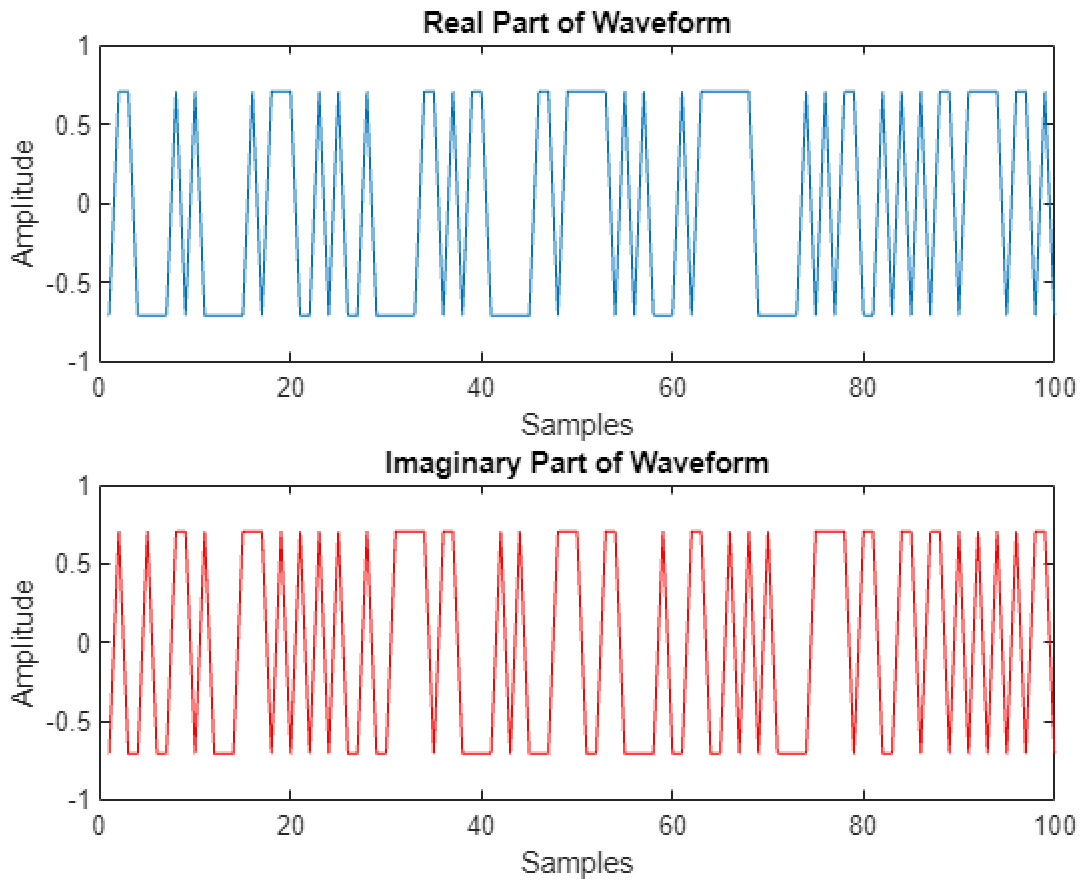
To update the dropdown menu with the antennas available for your radio, call the `hTransmitAntennas` helper function. Then select the antenna to use with this example.

```
antennaSelection = hTransmitAntennas(radio);
bbtx.Antennas = RF0:TX/RX ;
```

Plot Wireless Waveform

Plot the first hundred samples of the wireless waveform.

```
waveform = waveStruct.waveform ;
figure();
subplot(2,1,1); plot(real(double(waveform(1:100))));
title("Real Part of Waveform")
xlabel("Samples"); ylabel("Amplitude");
subplot(2,1,2); plot(imag(double(waveform(1:100))),color='r');
title("Imaginary Part of Waveform")
xlabel("Samples"); ylabel("Amplitude");
```



Transmit Wireless Waveform

Call the `transmit` function on the baseband transmitter object. Specify the type of transmission.

```
transmit(bbtx,waveform,continuous);
```

End Transmission

To end a continuous transmission, call the `stopTransmission` function on the baseband transmitter object.

```
stopTransmission(bbtx);
```

See Also

Functions
`radioConfigurations`

Objects
`basebandTransmitter`

More About

- “Capture from Frequency Band” on page 3-2
- “Supported Radio Devices” on page 2-3

Loopback Transmit and Capture

This example shows how to configure a software-defined radio (SDR) as a baseband transceiver to transmit and capture a custom wireless waveform over the air.

Set Up Radio

Call the `radioConfigurations` function. The function returns all available radio setup configurations that you saved using the Radio Setup wizard. For more information, see "Connect and Set Up NI USRP Radios".

```
savedRadioConfigurations = radioConfigurations;
```

To update the dropdown menu with your saved radio setup configuration names, click **Update**. Then select the radio to use with this example.

```
savedRadioConfigurationNames = [string({savedRadioConfigurations.Name})];
radio = myRadio ▾ Update ;
```

Specify Wireless Waveform

Use the attached `TestTone.mat` file to specify the transmit waveform. The `waveStruct` structure contains a complex sine tone that is generated by using the Wireless Waveform Generator app.

```
load("TestTone.mat")
```

Configure Baseband Transceiver

Create a baseband transceiver object with the specified radio. Because the object requires exclusive access to radio hardware resources, before running this example for the first time, clear any other object associated with the specified radio. In subsequent runs, to speed up the execution time of the example, reuse your new workspace object.

```
if ~exist("bbtrx","var")
    bbtrx = basebandTransceiver(radio);
end
```

Configure the baseband transceiver object using the parameters of the wireless waveform.

- Set the `SampleRate` property to the sample rate of the generated waveform.
- Set the `CenterFrequency` property to a value in the frequency spectrum indicating the position of the waveform transmission.

```
bbtrx.SampleRate = waveStruct.Fs ;
bbtrx.TransmitCenterFrequency = 2.4e9 ;
bbtrx.CaptureCenterFrequency = bbtrx.TransmitCenterFrequency;
```

Set the `TransmitRadioGain` and `CaptureRadioGain` properties according to the local wireless channel.

```
bbtrx.TransmitRadioGain = 10 ;
bbtrx.CaptureRadioGain = 10 ;
```

To update the dropdown menus with the antennas available for your radio, call the `hTransmitAntennas` and `hCaptureAntennas` helper functions. Then select the antennas to use with this example.

```
transmitAntennaSelection = hTransmitAntennas(radio);
captureAntennaSelection = hCaptureAntennas(radio);
bbtrx.TransmitAntennas = RF0:TX/RX ;
bbtrx.CaptureAntennas = RF0:RX2 ;
```

Transmit Wireless Waveform

Call the `transmit` function on the baseband transceiver object. Specify a continuous transmission.

```
transmit(bbtrx, waveStruct.waveform ,"continuous");
```

Capture IQ Data

To capture the transmitted waveform, call the `capture` function on the baseband receiver object. Specify the length of the capture.

```
pause(1)
captureLength = milliseconds( 10 );
data = capture(bbtrx,captureLength);
```

End Transmission

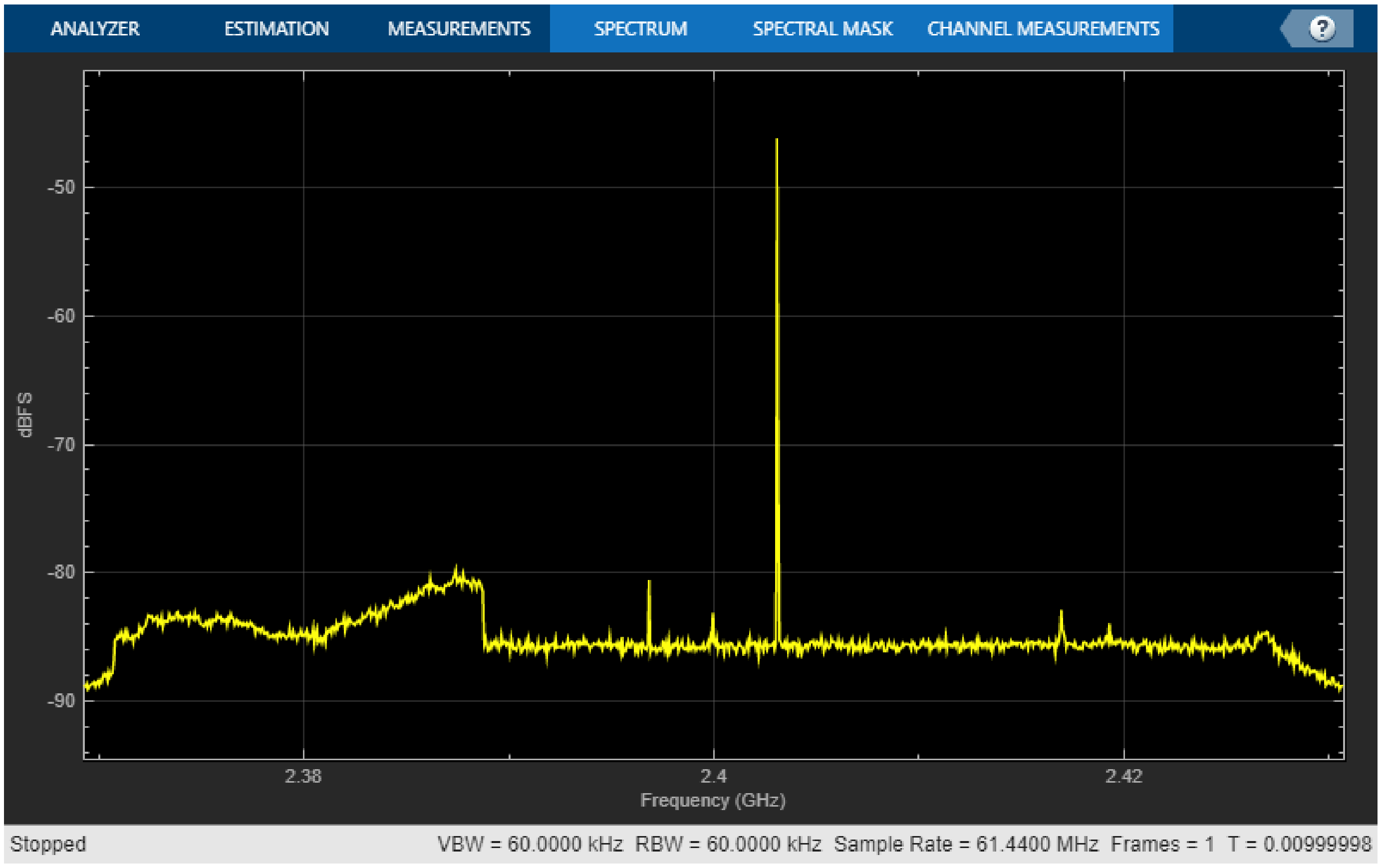
To end the continuous transmission, call the `stopTransmission` function on the baseband transceiver object.

```
stopTransmission(bbtrx);
```

Plot Spectrum of Captured Waveform

Create a `spectrumAnalyzer` object. To speed up the execution time of this example upon subsequent runs, reuse the spectrum analyzer object. Set the sample rate of the spectrum analyzer object to the sample rate of the baseband transceiver object. Plot the spectrum and spectrogram of the captured data.

```
if ~exist("spectrumScope","var")
    spectrumScope = spectrumAnalyzer;
end
spectrumScope.SampleRate = bbtrx.SampleRate;
spectrumScope.ChannelNames = bbtrx.CaptureAntennas;
spectrumScope.FrequencyOffset = bbtrx.CaptureCenterFrequency;
spectrumScope.SpectrumUnits = "dBFS";
spectrumScope.FullScaleSource = "Property";
spectrumScope.FullScale = double(intmax('int16'));
spectrumScope(data);
spectrumScope.show;
release(spectrumScope);
```



Stopped VBW = 60.0000 kHz RBW = 60.0000 kHz Sample Rate = 61.4400 MHz Frames = 1 T = 0.00999998

To transmit and capture the waveform again and to update the spectrum analyzer by rerunning the current section, click **Capture and plot frequency spectrum**.

Transmit, Capture, and plot frequency spectrum

See Also

Functions
`radioConfigurations`

Objects
`basebandTransceiver`

More About

- “Transmit Waveform” on page 3-5
- “Calibrate Radio Gain For Signal Capture”
- “Supported Radio Devices” on page 2-3

Triggered Capture Using Preamble Detection

This example shows how to use a software-defined-radio (SDR) to capture data from the air using preamble detection. The example also shows how to use the transmit capabilities of the same radio to loop back a test waveform.

Introduction

The example demonstrates these steps.

1. Generate a waveform containing a preamble.
2. Configure the preamble detector to detect the preamble sequence.
3. Use the `plotThreshold` function to calibrate a fixed or adaptive threshold and capture data.
4. Explore trigger offset.

Set Up Radio

Call the `radioConfigurations` function. The function returns all available radio setup configurations that you saved using the Radio Setup wizard. For more information, see “Connect and Set Up NI USRP Radios”.

```
savedRadioConfigurations = radioConfigurations;
```

To update the dropdown menu with your saved radio setup configuration names, click **Update**. Then select the radio to use with this example.

```
savedRadioConfigurationNames = [string({savedRadioConfigurations.Name})];
radio = MyRadio  Update;
```

Generate Transmission Waveform

Create a transmission waveform containing a Zadoff-Chu preamble sequence. To enable straightforward demonstration of the preamble detection workflow, concatenate zeros before and after the preamble.

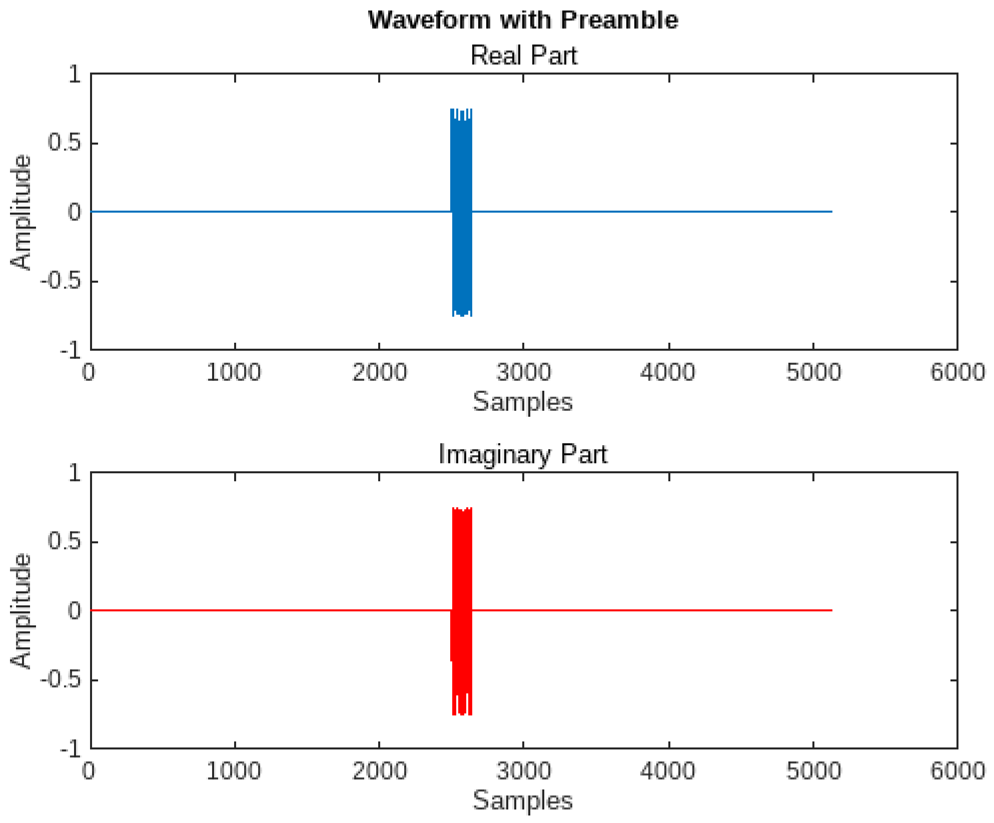
Generate a preamble sequence of length 137 by using 38th root of the Zadoff-Chu sequence and normalize. Concatenate with zeros.

```
zcseq = zadoffChuSeq(38,137);
preamble = zcseq/norm(zcseq,2);
prePadLen = 2501;
postPadLen = 2500;
headSignal = complex(zeros(prePadLen,1),zeros(prePadLen,1));
rearSignal = complex(zeros(postPadLen,1),zeros(postPadLen,1));
inputSignal = [headSignal; zcseq*0.75; rearSignal];
```

Plot transmission waveform.

```
figure();
subplot(2,1,1); plot(real(inputSignal));
subtitle("Real Part");
xlabel("Samples");
ylabel("Amplitude");
title("Waveform with Preamble");
```

```
subplot(2,1,2);
plot(imag(inputSignal),Color='r');
subtitle("Imaginary Part");
xlabel("Samples");
ylabel("Amplitude");
```



Configure Preamble Detector

Create a preamble detector object with the specified radio. Because the object requires exclusive access to radio hardware resources, before running this example for the first time, clear any other object associated with the specified radio. To speed up the execution time of the example in subsequent runs, reuse your new workspace object.

```
if ~exist("pd","var")
    pd = preambleDetector(radio);
end
```

Set the RF properties of the preamble detector. Set the `RadioGain` property according to the local wireless channel.

```
pd.SampleRate = 30720000;
pd.CenterFrequency = 2450000000;
pd.RadioGain = 30; % Increase if signal levels are low.
```

To update the dropdown menu with the antennas available for capture on your radio, call the `hCaptureAntennas` helper function. Then select the antenna to use with this example.

```
captureAntennaSelection = hCaptureAntennas(radio);
pd.Antennas = RF0:RX2;
```

Configure the preamble sequence for preamble detection.

```
pd.Preamble = preamble;
```

Set the capture data type to the data type of the generated transmission waveform.

```
pd.CaptureDataType = "double";
```

Configure Transmission Variables

Set the transmit gain and transmit antenna values. Set the transmit gain variable according to the local wireless channel.

```
txGain = 30; % Increase if signal levels are low.
```

To update the dropdown menu with the antennas available for transmit on your radio, call the `hTransmitAntennas` helper function. Then select the antenna to use with this example.

```
transmitAntennaSelection = hTransmitAntennas(radio);
txAntenna = RF0:TX/RX;
```

Detect Preamble Using Fixed Threshold and Capture Data

Data capture is triggered when the correlator output is greater than the fixed threshold. By setting the fixed threshold to 0, you can analyze the behavior of the preamble detector and understand how to set the fixed threshold value for successful detection.

Set the preamble detector to use fixed threshold. To set the threshold method, stop any ongoing transmission.

```
stopTransmission(pd);
pd.ThresholdMethod = "fixed";
```

Set the fixed threshold initially to 0.

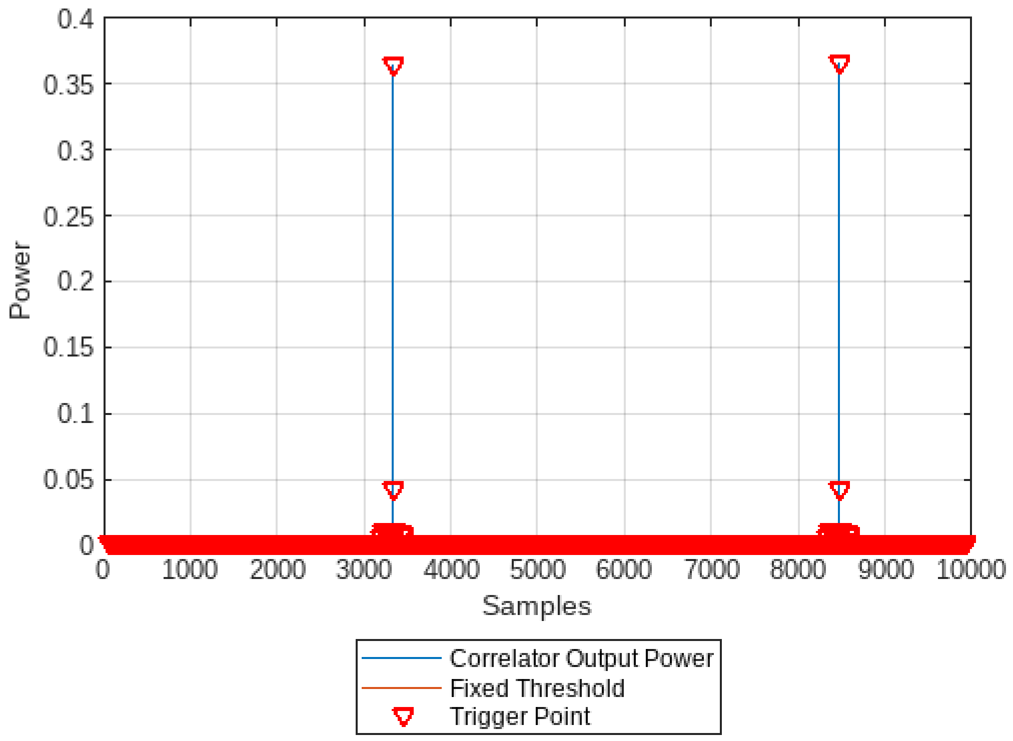
```
pd.FixedThreshold = 0;
```

Transmit the test waveform.

```
transmit(pd,inputSignal,"continuous",TransmitGain=txGain, ...
    TransmitCenterFrequency=pd.CenterFrequency,TransmitAntennas=txAntenna);
```

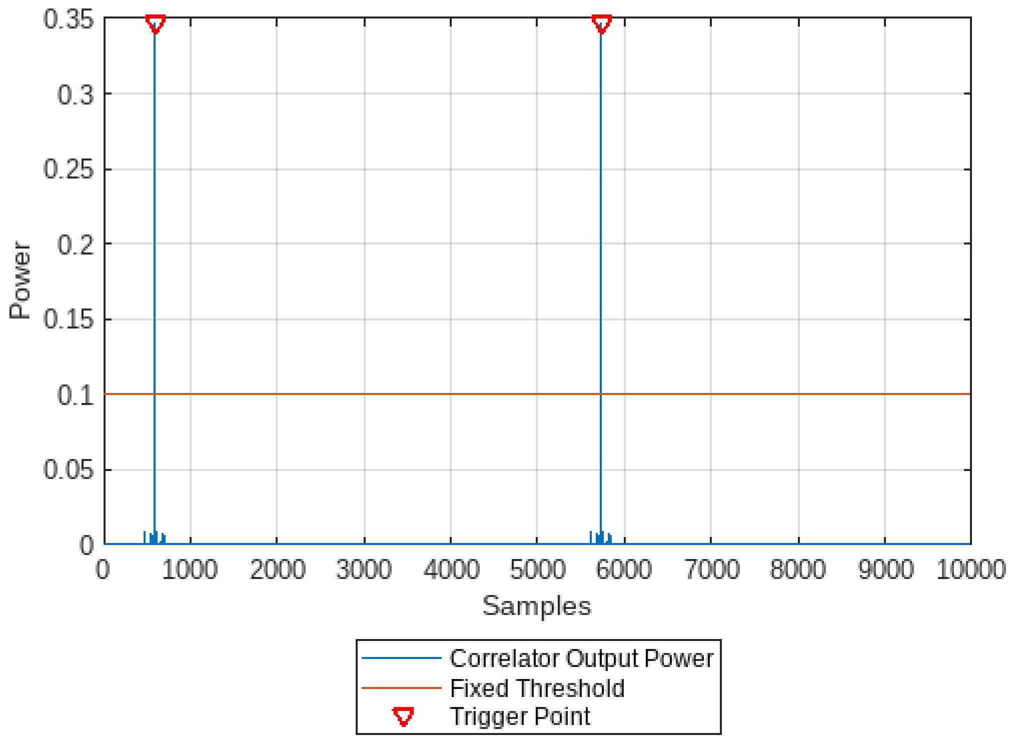
Use the `plotThreshold` function to analyze the behavior of the detector by plotting 10,000 samples. Because the fixed threshold value is 0, all samples from the correlator output are possible trigger points. Check the correlator output values at the peak trigger points. Because the sampling phase determines the quality of the correlator peak, run the `plotThreshold` function multiple times to see how the trigger points change.

```
plotThreshold(pd,10e3);
```



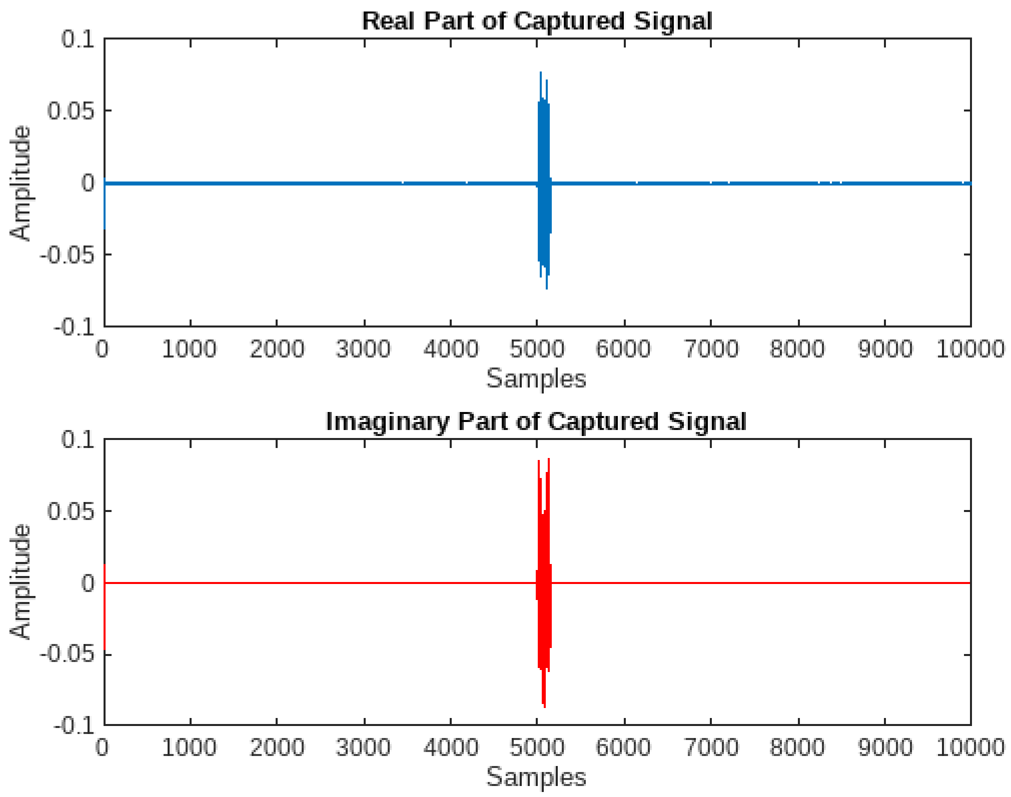
Choose a threshold value that is below any of the trigger point values. Plot the threshold information again and adjust the fixed threshold until the trigger points appear only on the correlator output peak. Run the `plotThreshold` function multiple times to see any sampling phase effects.

```
pd.FixedThreshold = 0.1;
plotThreshold(pd,10e3);
```



Once the threshold is set, capture data.

```
[data, ~, ~, status] = capture(pd,10e3,seconds(1));
plotCapturedData(data,status);
```



Detect Preamble Using Adaptive Threshold and Capture Data

As an alternative to the fixed threshold, data capture can be triggered when the correlator output is greater than the adaptive threshold, which dynamically varies with the input signal power. By setting the adaptive threshold gain and offset to 0, you can analyze the behavior of the preamble detector and understand how to configure the adaptive threshold for successful detection.

Set the preamble detector to use adaptive threshold. To set the threshold method, stop any ongoing transmission.

```
stopTransmission(pd);
pd.ThresholdMethod = 'adaptive';
```

Set the adaptive threshold gain and offset initially to 0.

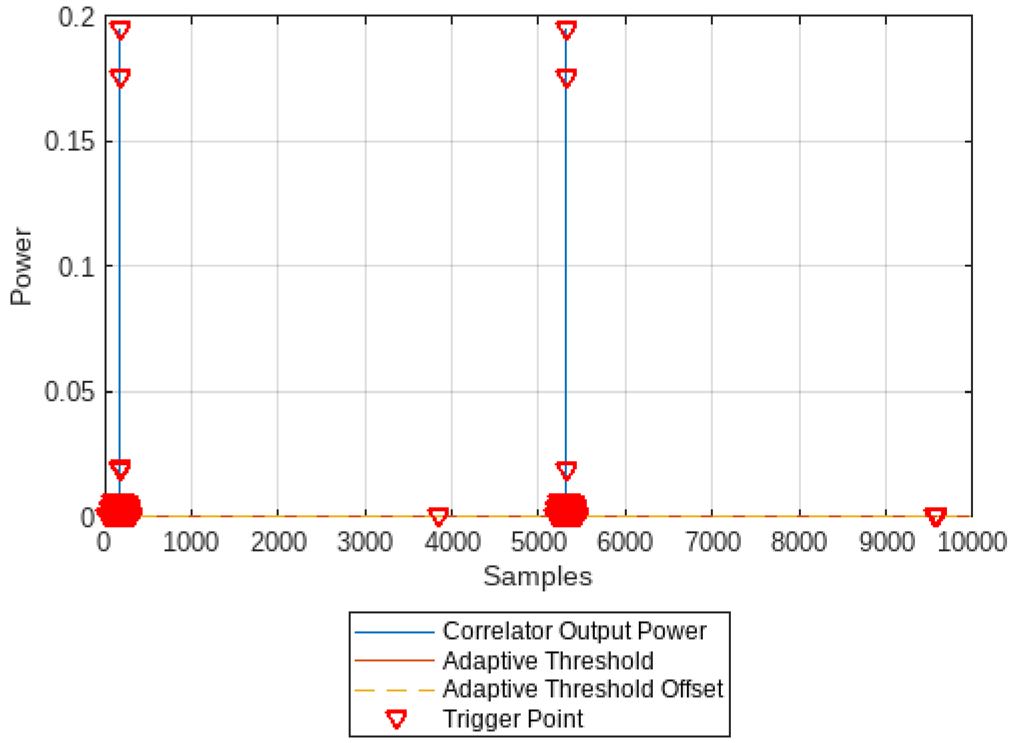
```
pd.AdaptiveThresholdGain = 0;
pd.AdaptiveThresholdOffset = 0;
```

Transmit the test waveform.

```
transmit(pd,inputSignal,"continuous",TransmitGain=txGain, ...
    TransmitCenterFrequency=pd.CenterFrequency,TransmitAntennas=txAntenna);
```

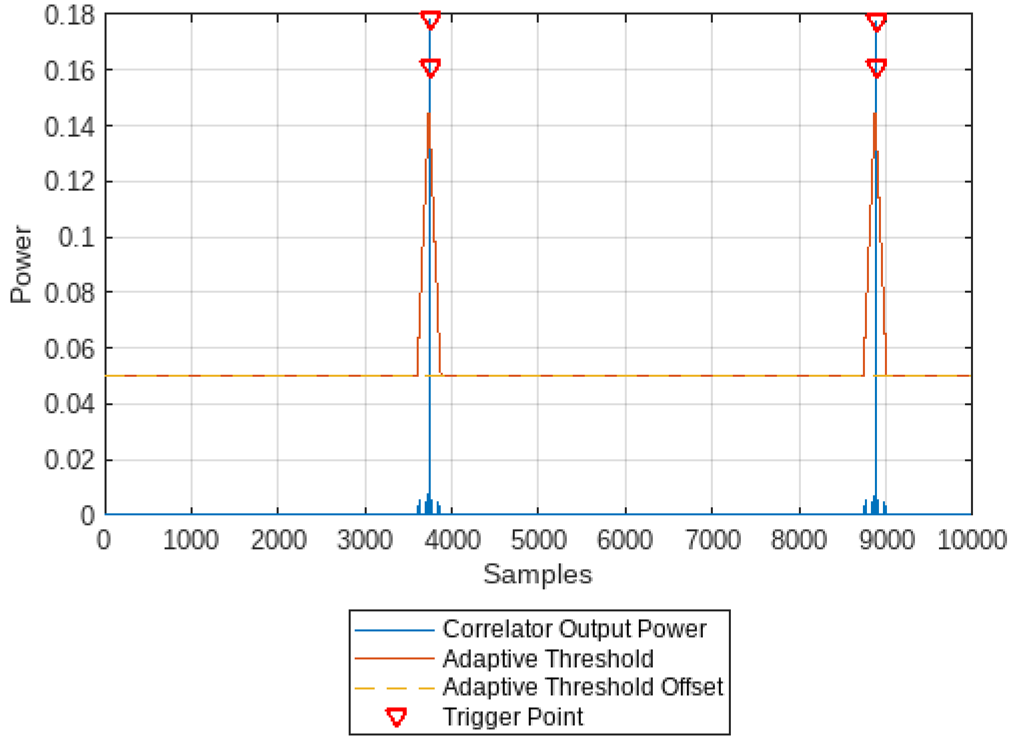
Use the `plotThreshold` function to analyze the behavior of the detector by plotting 10,000 samples. Check the correlator output values at the peak trigger points and run the `plotThreshold` function multiple times if necessary.

```
plotThreshold(pd,10e3);
```



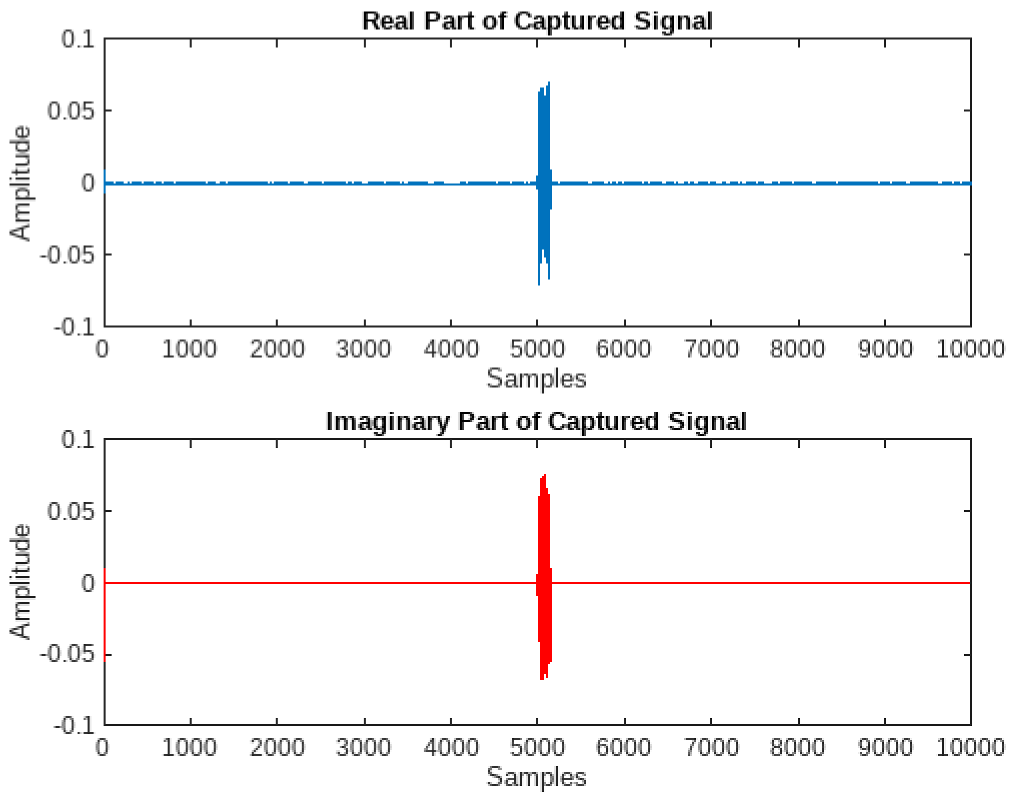
To remove all the trigger points from the bottom of the plot, set the adaptive threshold offset to a value that is above the noise floor. Adjust the adaptive threshold gain and plot the threshold information repeatedly until the correlator output is greater than the adaptive threshold.

```
pd.AdaptiveThresholdOffset = 0.05;
pd.AdaptiveThresholdGain = 0.25;
plotThreshold(pd,10e3);
```



Once the threshold is configured, capture data.

```
[data, ~, ~, status] = capture(pd,10e3,seconds(1));
plotCapturedData(data,status);
```



Set Trigger Offset to Include Preamble in Captured Data

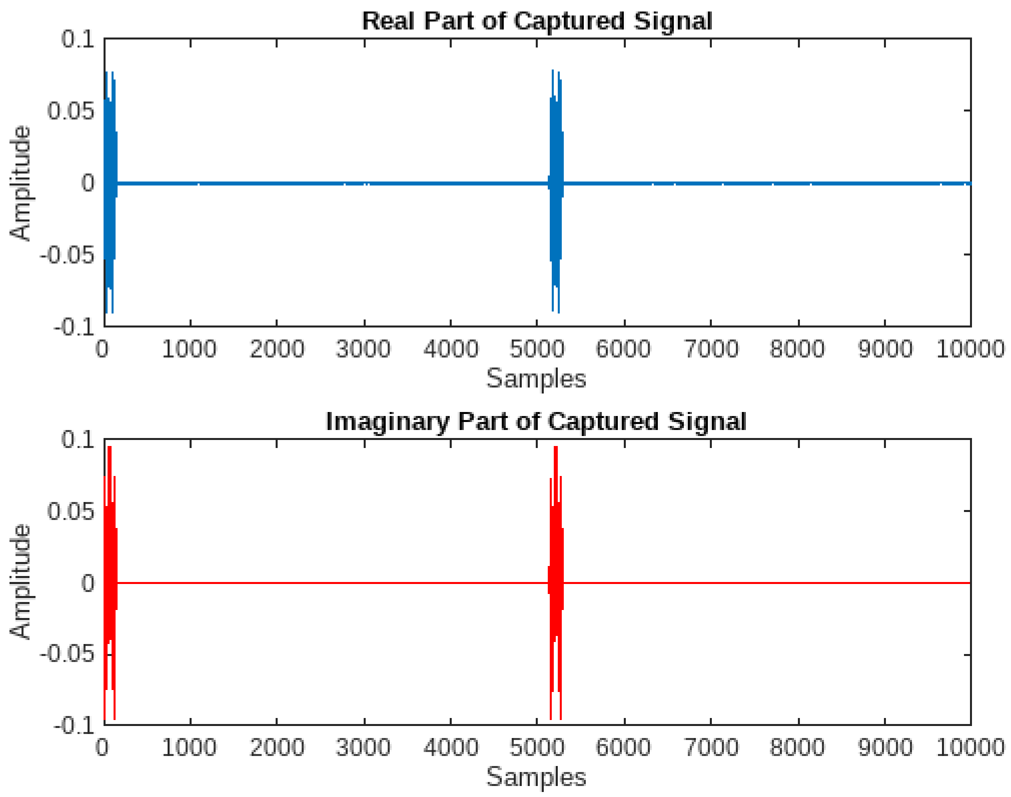
To capture the preamble sequence, set the trigger offset to the length of the preamble. To set the trigger offset, stop any ongoing transmission.

```
stopTransmission(pd);
```

```
pd.TriggerOffset = -137;
```

Transmit the test waveform and capture data.

```
transmit(pd, inputSignal,"continuous",TransmitGain=txGain,...
    TransmitCenterFrequency=pd.CenterFrequency,TransmitAntennas=txAntenna);
% Detect and capture 10,000 samples, with a 1 second timeout
[data, ~, ~, status] = capture(pd,10e3,seconds(1));
plotCapturedData(data,status);
```



End Transmission

To end the continuous transmission, call the `stopTransmission` function on the preamble detector object.

```
stopTransmission(pd);
```

Local Functions

```
function plotCapturedData(data,status)
if status     % If detection is successful, plot data
    figure();
    subplot(2,1,1); plot(real(double(data)));
    title("Real Part of Captured Signal")
    xlabel("Samples"); ylabel("Amplitude");
    subplot(2,1,2); plot(imag(double(data)),color='r');
    title("Imaginary Part of Captured Signal")
    xlabel("Samples"); ylabel("Amplitude");
else
    disp("Detection failed.")
end
end
```

See Also

Functions
`radioConfigurations`

Objects
`preambleDetector`

More About

- “Capture from Frequency Band” on page 3-2
- “Supported Radio Devices” on page 2-3