

**Wireless Testbench™**

User's Guide



**MATLAB®**

R2023a



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



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

*Wireless Testbench™ User's Guide*

© COPYRIGHT 2022–2023 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](http://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](http://www.mathworks.com/patents) for more information.

## 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)

<b>1</b>	<b>Radio Management</b>	
	<b>Install Support Package for NI USRP Radios</b> .....	<b>1-2</b>
	<b>Connect and Set Up NI USRP Radios</b> .....	<b>1-3</b>
	Start Radio Setup Wizard .....	<b>1-3</b>
	<b>Reference Information for NI USRP Radio Setup Wizard</b> .....	<b>1-4</b>
	SD Card Image Creation .....	<b>1-4</b>
	Radio Device Connection .....	<b>1-4</b>
	Network Configuration .....	<b>1-4</b>
	Validate Radio Connection .....	<b>1-5</b>
	Update Radio Filesystem .....	<b>1-5</b>
	Data Transfer Optimization .....	<b>1-6</b>
	<b>Baseband Sample Rate in NI USRP Radios</b> .....	<b>1-8</b>
	Supported Master Clock Rates .....	<b>1-8</b>
	Set Baseband Sample Rate .....	<b>1-8</b>
Farrow Rate Converter .....	<b>1-9</b>	
Resampling Transmit Waveform .....	<b>1-9</b>	

<b>2</b>	<b>Transmit and Capture</b>	
	<b>Capture Wideband Spectrum by Combining Data from Multiple Antennas</b> .....	<b>2-2</b>
	<b>Transmit App-Generated Wireless Waveform Using Radio Transmitters</b> .....	<b>2-6</b>
	<b>Wideband Spectrum Analysis</b> .....	<b>2-9</b>
	<b>Calibrate Radio Gain For Signal Capture</b> .....	<b>2-12</b>
	<b>Save Captured Signal with Metadata to Baseband File</b> .....	<b>2-21</b>

<b>OFDM Wi-Fi Scanner Using SDR Preamble Detection . . . . .</b>	<b>3-2</b>
<b>Triggered WLAN Waveform Capture Using Preamble Detection . . . . .</b>	<b>3-16</b>

# Radio Management


---

## Install Support Package for NI USRP Radios

The Wireless Testbench™ Support Package for NI USRP Radios enables you to connect and set up your NI USRP radio for use in Wireless Testbench.

You can download and install the support package from File Exchange. Alternatively, you can get the support package from the Add-On Explorer by following these steps:

- 1** On the MATLAB® **Home** tab, in the **Environment** section, click **Add-Ons > Get Hardware Support Packages**.
- 2** In the Add-On Explorer window, browse or search for the Wireless Testbench Support Package for NI USRP Radios.
- 3** Select the support package and then click **Install**.

When the support package installation is complete, you can proceed directly to setting up the radio for use in Wireless Testbench by clicking the setup button  in the Add-On Manager window. For more information, see “Connect and Set Up NI USRP Radios” on page 1-3.

### See Also

#### More About

- “Connect and Set Up NI USRP Radios” on page 1-3
- “Supported Radio Devices”

## Connect and Set Up NI USRP Radios

Before you can start interacting with RF signals using Wireless Testbench features and an NI USRP radio, you must set up your radio using the Radio Setup wizard.

The Radio Setup wizard enables you to perform these tasks.

- Set up your radio for use in Wireless Testbench.
- Save a radio setup configuration under a name. You use this name later when configuring your radio for use with a specific Wireless Testbench reference application.
- List all saved radio setup configurations.
- Update, delete or re-validate saved radio setup configurations.

### Start Radio Setup Wizard

After installing the Wireless Testbench Support Package for NI USRP Radios, you can go directly to the Radio Setup wizard. Alternatively, you can start the Radio Setup wizard by following these steps.

- 1 On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons > Manage Add-Ons**.
- 2 In the Add-On Manager window, find the Wireless Testbench Support Package for NI USRP Radios, then click the setup button .

The Radio Setup wizard leads you through several steps. If you have to close the Radio Setup wizard at any time, you can start the wizard again from the Add-On Manager window. If you need more information about how to perform the wizard steps, see “Reference Information for NI USRP Radio Setup Wizard” on page 1-4.

Once your radio setup is complete, you can start exploring Wireless Testbench examples.

---

**Note** Saved radio setup configurations are persistent across MATLAB sessions. You only need to run the Radio Setup wizard again if you want to set up a new radio or you want to manage saved radio setup configurations.

---

### See Also

#### More About

- “Install Support Package for NI USRP Radios” on page 1-2
- “Reference Information for NI USRP Radio Setup Wizard” on page 1-4
- “Supported Radio Devices”

## Reference Information for NI USRP Radio Setup Wizard

The Radio Setup wizard of the Wireless Testbench Support Package for NI USRP Radios leads you through steps to connect and set up your radio for use in Wireless Testbench. Use this section as a reference to get more information about how to perform these steps. For more information about how to start the wizard, see “Start Radio Setup Wizard” on page 1-3.

### SD Card Image Creation

For NI USRP radios with an SD card, the host computer must have at least one microSD card reader and one writable microSD card. The radio takes a microSD card, and if the host has a standard SD card reader, you can use an adapter. If the host computer does not have an integrated card reader, use an external USB SD card reader.

- If your host computer has a Linux<sup>®</sup> operating system (OS), your SD card must have a FAT32 partition with a minimum of 14.8 GB of free space.
- If your host computer has a Windows<sup>®</sup> OS, your SD card must have a formatted partition and a minimum of 14.8 GB of free space.

---

**Note** If you previously used your SD card with a USRP radio on a Windows OS, your SD card can show up as a disk containing multiple partitions in the **Drive** dropdown.

---

### Radio Device Connection

The Ethernet connection is often referred to as a network connection. You can use an integrated network interface card (NIC) with a Gigabit Ethernet cable. This connection is necessary for transmitting data, such as the field programmable gate array (FPGA) image or firmware image, from the computer to the radio. It is also necessary for sending and receiving signals to and from the radio. For NI USRP radios, the host computer must contain at least one dedicated 1 or 10 Gigabit NIC for connecting to the radio. If you have a single NIC that is required for internet access, consider using a USB3/Ethernet adapter dongle. However, using an adapter can reduce data transfer speed between the computer and the radio. You can use the small form-factor pluggable (SFP) network interface with a Gigabit Ethernet cable.

### Network Configuration

The network configuration of the host IP address is persistent on Windows. However, on Linux systems, unless you make the changes in your network connection persistent, a system reboot resets the network connection changes and loses the host-to-radio connection.

To retain the host-to-radio connection during a computer reboot on Linux systems, make the network connection changes to the host computer persistent by editing the `/etc/network/interfaces` file.

- 1 Edit `/etc/network/interfaces` to define settings for `eth1`.
- 2 Use an IP address on the same subnet as your radio (that is, with three initial octets that match those of your radio) and a unique value for the fourth octet. For example:

```
auto eth1
iface eth1 inet static
    address 192.168.30.1
    netmask 255.255.255.0
```



## Validate Radio Connection

### Host and Radio Configuration

During the first three validation tests, the Radio Setup wizard validates the host and the radio configuration. You can progress to the next wizard step by passing these tests.

- If the host IP address configuration test fails, check that the host IP address that you selected in the **Select Link Configuration** wizard step matches the host OS network configuration.
- If the host-radio connection test fails:
  - Check that the physical connection between the radio and host is through the correct ports.
  - Check that the radio is on or try powering the radio off then on again.
  - Go back to the **Set Up SD Card** step in the wizard and set up the SD card again with a compatible image.
- If the radio SD card image version test fails (applies only to NI USRP radios with an SD card reader):
  - Disable any OS firewall for the host network interface that is connected to the radio.
  - Go back to the **Set Up SD Card** step in the wizard and set up the SD card again with a compatible image.
- If the version test fails for the image of the radio filesystem (applies only to NI USRP X410 radios):
  - Disable any OS firewall for the host network interface that is connected to the radio.
  - Update the radio filesystem manually. For instructions, see “Update Radio Filesystem” on page 1-5.

### OS Optimization and Data Transfer

During the last two validation tests, the Radio Setup wizard validates the OS optimization setting and the data transfer. Any validation failure that occurs during these steps does not stop you from progressing to the next wizard step. However, to optimize the transfer speed between the host computer and the radio and to ensure that your radio can transmit and receive signals, consider fixing all validation failures.

- If the host OS optimization test fails, update your OS settings. For instructions, see “Data Transfer Optimization” on page 1-6.
- If the radio transmit and receive test fails:
  - Fix any OS optimization test failure that occurs in the previous validation test.
  - Disable any OS firewall for the host network interface that is connected to the radio.
  - Check that you have attached an antenna or a loopback cable to the correct port.

## Update Radio Filesystem

If the version validation of the NI USRP X410 radio filesystem image fails, you must update the filesystem manually.

- 1 Follow the steps in the wizard to download, extract, and validate the correct filesystem image file.

- 2 Update your radio filesystem by using a Windows PowerShell or Linux terminal. By default, the radio IP address is 192.168.10.2. If in the network configuration step of the wizard you configured your radio with a different IP address, specify that IP address instead of 192.168.10.2 in these commands.

- a Copy the extracted .mender filesystem image file onto the radio. If necessary, provide the full path to the image file.

```
$ scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no <mender_filesystem_image> root@192.168.10.2:/
```

- b Log in to the radio.

```
$ ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@192.168.10.2
```

- c Install the new filesystem image. Then restart your radio. These steps take several minutes.

```
# mender install /tmp/wt-uhd-image.mender  
# reboot
```

- d Log in to the radio.

```
$ ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@192.168.10.2
```

- e Make the filesystem changes permanent on the radio.

```
# mender commit
```

For more information on updating the radio filesystem, see [Installing the Mender Artifact on the hardware vendor website](#).

- 3 Return to the Radio Setup wizard and follow the instructions to revalidate your radio setup.

## Data Transfer Optimization

To optimize the transfer speed between the host computer and the radio, you must update the *FastSendDatagramThreshold* registry key on Windows or the *wmem* and *rmem* network buffer sizes on Linux systems. To automatically update your OS with this setting, click **Press here to set** in the **Validate** step of the Radio Setup wizard. If you encounter issues during this update, follow these steps to manually update your OS.

### Windows

- 1 Locate the *FastSendDatagramThreshold.reg* file at *UHD\_INSTALL/win64* path by running these commands.

```
>> uhd_install_location = getUHDInstallLocation  
>> cd(fullfile(uhd_install_location, 'win64'))
```

- 2 Double-click and run the *FastSendDatagramThreshold.reg* file.
- 3 Restart your computer.

### Linux

Run these commands in a terminal.

```
`echo "net.core.rmem_max=2500000" | sudo tee -a /etc/sysctl.conf`  
`echo "net.core.wmem_max=2500000" | sudo tee -a /etc/sysctl.conf`  
`sudo sysctl -p`
```

## **See Also**

### **More About**

- “Connect and Set Up NI USRP Radios” on page 1-3

## Baseband Sample Rate in NI USRP Radios

### In this section...

“Supported Master Clock Rates” on page 1-8

“Set Baseband Sample Rate” on page 1-8

“Farrow Rate Converter” on page 1-9

“Resampling Transmit Waveform” on page 1-9

Each Wireless Testbench reference application object enables you to set the baseband sample rate of the radio by using the `SampleRate` object property. The object automatically selects the master clock rate on the radio hardware based on the specified sample rate. If necessary, to achieve the specified sample rate, the radio uses a Farrow rate converter.

### Supported Master Clock Rates

The analog-to-digital converter (ADC) and digital-to-analog converter (DAC) in USRP radios run at the full master clock rate, which is hardware-dependent. This table shows the master clocks rates available on supported NI USRP radios.

Radio Device	Master Clock Rate
USRP N310	<ul style="list-style-type: none"> <li>• 122.88 MHz</li> <li>• 125.00 MHz</li> <li>• 153.60 MHz</li> </ul>
USRP N320	<ul style="list-style-type: none"> <li>• 200.00 MHz</li> <li>• 245.76 MHz</li> <li>• 250.00 MHz</li> </ul>
USRP N321	<ul style="list-style-type: none"> <li>• 200.00 MHz</li> <li>• 245.76 MHz</li> <li>• 250.00 MHz</li> </ul>
USRP X310	<ul style="list-style-type: none"> <li>• 184.32 MHz</li> <li>• 200.00 MHz</li> </ul>
USRP X410	<ul style="list-style-type: none"> <li>• 245.76 MHz</li> <li>• 250.00 MHz</li> </ul>

### Set Baseband Sample Rate

The object automatically selects the master clock rate on the radio hardware based on the `SampleRate` property value. If necessary, to achieve the specified sample rate, the radio uses a Farrow rate converter along with integer interpolation for transmit signals or integer decimation for capture signals.

To bypass the Farrow filter, set the `SampleRate` property to a master clock rate value or to a value such that  $MCR/\text{SampleRate}$  is a supported decimation or interpolation factor, where  $MCR$  is the master clock rate that the object selects.

Radio	Supported Decimation or Interpolation Factor
USRP N310	1
USRP N320	2
USRP N321	3
USRP X410	Even integer in the range from 4 to 256
	Multiple of 4 in the range from 256 to 512
	Multiple of 8 in the range from 512 to 1016
USRP X310	Integer in the range from 1 to 128
	Even integer in the range from 128 to 256
	Multiple of 4 in the range from 256 to 512
	Multiple of 8 in the range from 512 to 1016

## Farrow Rate Converter

If  $MCR/SampleRate$  is not a supported decimation or interpolation factor, the signal passes through the Farrow rate converter. Because of hardware limitations, the Farrow rate converter can only convert sample rates that are less than  $MCR/2$ .

## Resampling Transmit Waveform

If the `basebandTransceiver`, `basebandTransmitter`, or `preambleDetector` object uses a Farrow rate converter to achieve the sample rate specified by the `SampleRate` property, the object resamples the transmit waveform that you specify when calling the `transmit` object function. The resulting waveform has an increased number of data samples.

## See Also

### Objects

`basebandReceiver` | `basebandTransceiver` | `basebandTransmitter` | `preambleDetector`

## More About

- “Connect and Set Up NI USRP Radios” on page 1-3
- “Supported Radio Devices”



# Transmit and Capture

---

## Capture Wideband Spectrum by Combining Data from Multiple Antennas

This example shows how to configure a software-defined radio (SDR) as a baseband receiver to capture a wideband spectrum by combining received data from multiple antennas using a multiband combiner. The example also plots the combined wideband spectrogram of the captured data.

### Introduction

In this example, you use the `basebandReceiver` object to capture a wide frequency band by combining data from two radio antennas with different center frequencies. Because each antenna captures data at a different center frequency, you can capture a wider frequency band than using a single antenna. To view the whole frequency band, you then use the `comm.MultibandCombiner` System object™.

### 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” on page 1-3.

```
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 =   ;
```

### 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 largest combined bandwidth, set the `SampleRate` property to the maximum supported value on the SDR. Choose a single antenna bandwidth less than or equal to the maximum instantaneous bandwidth of your radio and less than the chosen sample rate to reduce aliasing effects.

To obtain the maximum sample rate and instantaneous bandwidth available for your radio, call the `hMaxSampleRate` and `hMaxBandwidth` helper functions. Alternatively, you can set custom values.

```
maxSampleRate = hMaxSampleRate(radio);  
bbrx.SampleRate =  ;  
maxBandwidth = hMaxBandwidth(radio);  
singleAntennaBandwidth =  ;
```



Set the `Antennas` object property to a value that corresponds to two antennas that support different center frequencies on the SDR.

To update the dropdown menu with antennas that support different center frequencies, call the `hIndependentFrequencyCaptureAntennas` helper function. Then select the antennas to use with this example.

```
[firstAntennaSelection, secondAntennaSelection] = hIndependentFrequencyCaptureAntennas(radio);
bbrx.Antennas = [ RF0:RX2 , RF1:RX2 ];
```

Because the multiband combining operation (after capture) requires double-precision or single-precision input data, configure the baseband receiver to return the captured data in single precision.

```
bbrx.CaptureDataType = "single";
```

Set the `RadioGain` object property according to the local signal strength.

```
bbrx.RadioGain = 30;
```

Set the center frequency for each antenna by shifting the center frequency of the combined band by half of the single antenna bandwidth.

```
centerFrequency = 2.55e9;
bbrx.CenterFrequency = [centerFrequency - singleAntennaBandwidth/2, ...
    centerFrequency + singleAntennaBandwidth/2];
```

### Capture IQ Data

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

```
captureLength = milliseconds(100);
data = capture(bbrx,captureLength);
```

### Filter and Combine Captured Data

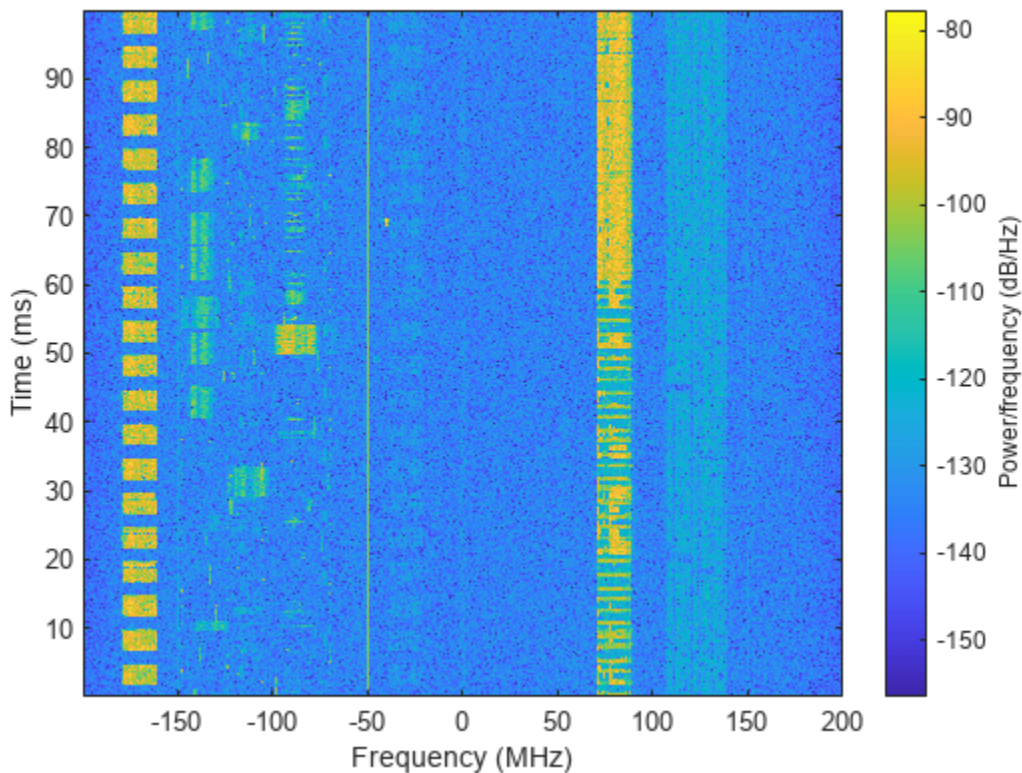
To reduce aliasing effects, filter the captured data to the selected bandwidth. Create a `comm.MultibandCombiner` System object and combine the captured data from each antenna.

```
filteredData = lowpass(data,singleAntennaBandwidth/2,bbrx.SampleRate);
mbc = comm.MultibandCombiner( ...
    InputSampleRate=bbrx.SampleRate, ...
    FrequencyOffsets=(bbrx.CenterFrequency-centerFrequency), ...
    OutputSampleRateSource="Auto");
combinedData = mbc(filteredData);
mbcInfo = info(mbc);
resampledData = resample(combinedData,2*singleAntennaBandwidth,mbcInfo.OutputSampleRate);
```

### Plot Spectrogram

Plot the spectrogram of the resampled data. The example uses 4096 FFT points and a 50% overlapping Hann window of length equal to the length of the resampled data divided by 4096. Alternatively, you can experiment with custom values.

```
window = hann(floor(length(resampledData)));  
nOverlap = floor(length(window)/2);  
nFFT = 4096;  
spectrogram(resampledData,window,nOverlap,nFFT,2*singleAntennaBandwidth,"centered");
```



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

- “Capture from Frequency Band”
- “Calibrate Radio Gain For Signal Capture” on page 2-12

- “Connect and Set Up NI USRP Radios” on page 1-3
- “Supported Radio Devices”

# Transmit App-Generated Wireless Waveform Using Radio Transmitters

Use the NI™ USRP™ N310, USRP N320, USRP N321, USRP X310, and USRP X410 radio transmitters, available in the **Wireless Waveform Generator** app, to transmit an app-generated waveform over the air (requires Wireless Testbench™). These radio transmitters enable you to transmit up to 2 GB of contiguous data over the air at full radio device rate.

## Introduction

The Wireless Waveform Generator app is an interactive tool for creating, impairing, visualizing, and transmitting waveforms. Using a radio transmitter available in the app, you can transmit your generated waveform repeatedly over the air. You can also export the waveform generation and transmission parameters to a runnable MATLAB script. Configure these radio transmitters to transmit an OFDM waveform. The same process applies for all waveform types that you can generate with the app.

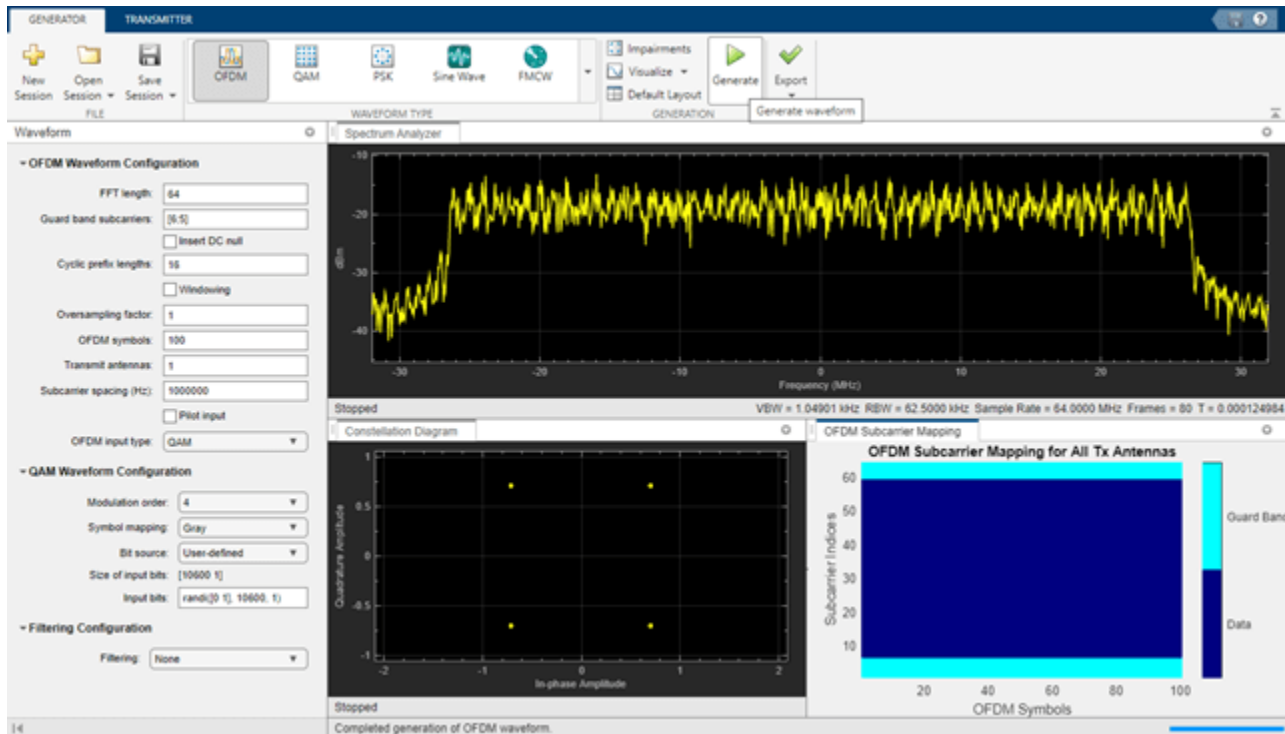
## Set Up for Radio Transmission

To use the radio transmitters in the app, you must install the Wireless Testbench Support Package for NI USRP Radios add-on, and set up your radio outside the app. For more information, see “Connect and Set Up NI USRP Radios” on page 1-3.

## Generate Waveform for Transmission

Open the **Wireless Waveform Generator** app by clicking the app icon on the **Apps** tab, under **Signal Processing and Communications**. Alternatively, enter `wirelessWaveformGenerator` at the MATLAB command prompt.

In the **Waveform Type** section, select an OFDM waveform by clicking **OFDM**. In the **Waveform** pane of the app, specify the parameters of **OFDM Waveform Configuration**, **QAM Waveform Configuration**, and **Filtering Configuration** for the selected waveform. Then, generate the configuration by clicking **Generate** in the app toolstrip.



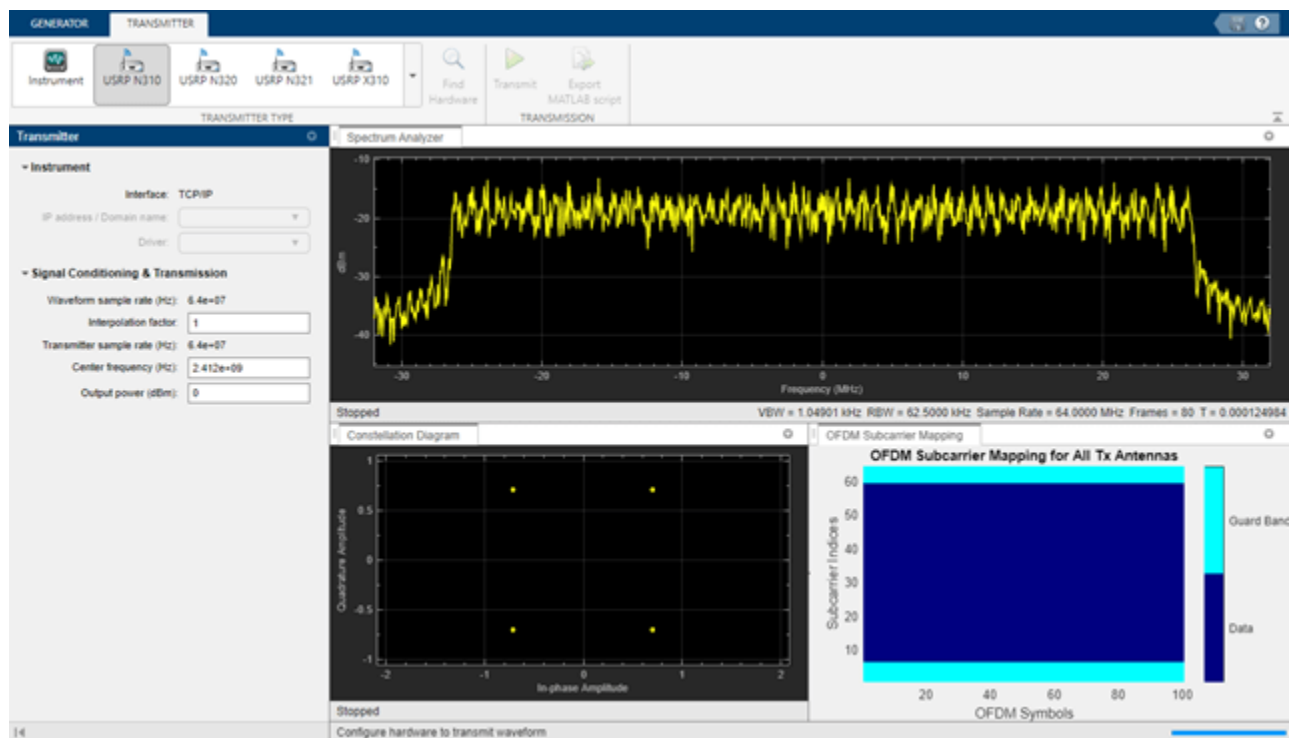
## Configure Radio Transmitter

Select the **Transmitter** tab from the app toolstrip. In the transmitter gallery, select a radio transmitter.

In the **Waveform** pane of the app, select the name of a radio setup configuration that you saved using the Radio Setup wizard. For more information, see “Connect and Set Up NI USRP Radios” on page 1-3.

Set the **Center frequency**, **Gain**, and **Antennas** configuration parameters. The app automatically sets the waveform sample rate based on the waveform that you generated earlier. The radio transmitter uses onboard data buffering to ensure contiguous data transmission at up to the full hardware sample rate. If necessary, to achieve the specified sample rate, the radio uses a Farrow rate converter. Use this list as a reference when setting the sample rate:

- **USRP N310:** 120,945 Hz to 76.8 MHz, or one of: 122.88 MHz, 125 MHz, or 153.6 MHz
- **USRP N320:** 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz or 250 MHz
- **USRP N321:** 196,851 Hz to 125 MHz, or one of: 200 MHz, 245.76 MHz, or 250 MHz
- **USRP X310:** 181,418 Hz to 100 MHz, or one of: 184.32 MHz or 200 MHz
- **USRP X410:** 241,890 Hz to 125 MHz, or one of: 245.76 MHz or 250 MHz



### Transmit Waveform

To transmit the waveform continuously, click **Transmit**. To end the continuous transmission, click **Stop transmission**. To export the waveform generation and transmission parameters to a runnable MATLAB script, click **Export MATLAB script**.

### See Also

#### Functions

radioConfigurations

### More About

- “Supported Radio Devices”
- “Calibrate Radio Gain For Signal Capture” on page 2-12

## Wideband Spectrum Analysis

This example shows how to capture a wideband signal from the air using a software-define-radio (SDR), then analyzes the captured data using the Signal Analyzer app.

### 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” on page 1-3.

```
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 =   ;
```

### 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. To speed up the execution time of the example in subsequent runs, reuse your new workspace object.

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

Set the baseband receiver object properties to capture the signal of interest.

```
maxSampleRate = hMaxSampleRate(radio);
bbrx.SampleRate =  ;
bbrx.RadioGain =  ;
```

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 =  ;
```

### Capture Signal

To capture a signal, call the `capture` function on the baseband receiver object. Specify the length of the capture and the center frequency.

```
bbrx.CenterFrequency =  ;
captureLength = milliseconds( );
[data,timestamp] = capture(bbrx,captureLength);
```

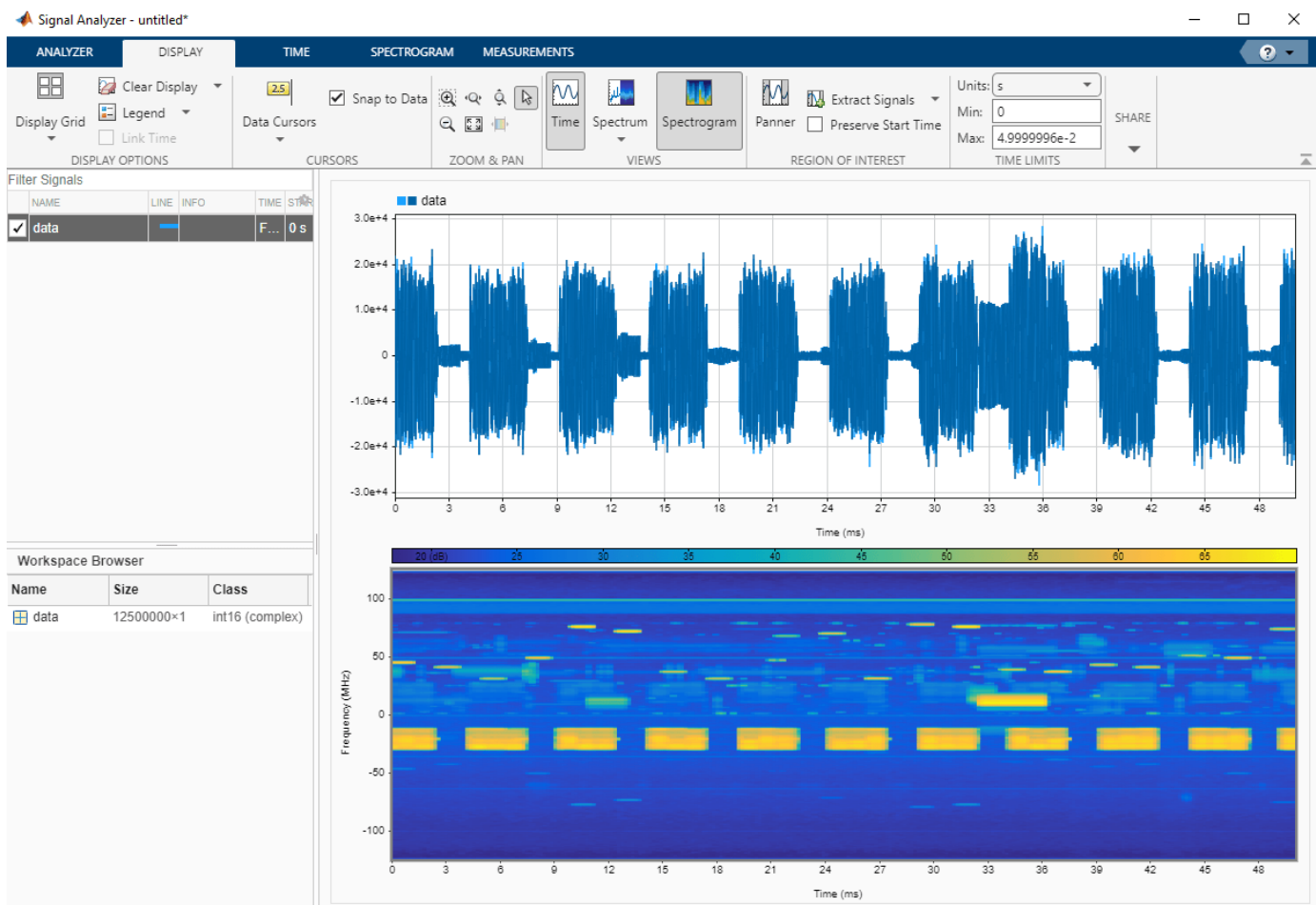
## Analyze Captured Signal

To analyze the captured signal in the time, frequency, or time-frequency domains, open the **Signal Analyzer** app, specifying the captured data as an input argument. The app enables you to visualize and compare multiple signals, plot the signal spectrum, measure signal statistics, or manipulate the data, such as, smoothing, filtering, resampling, or region extraction.

Open the **Signal Analyzer** app with the captured data.

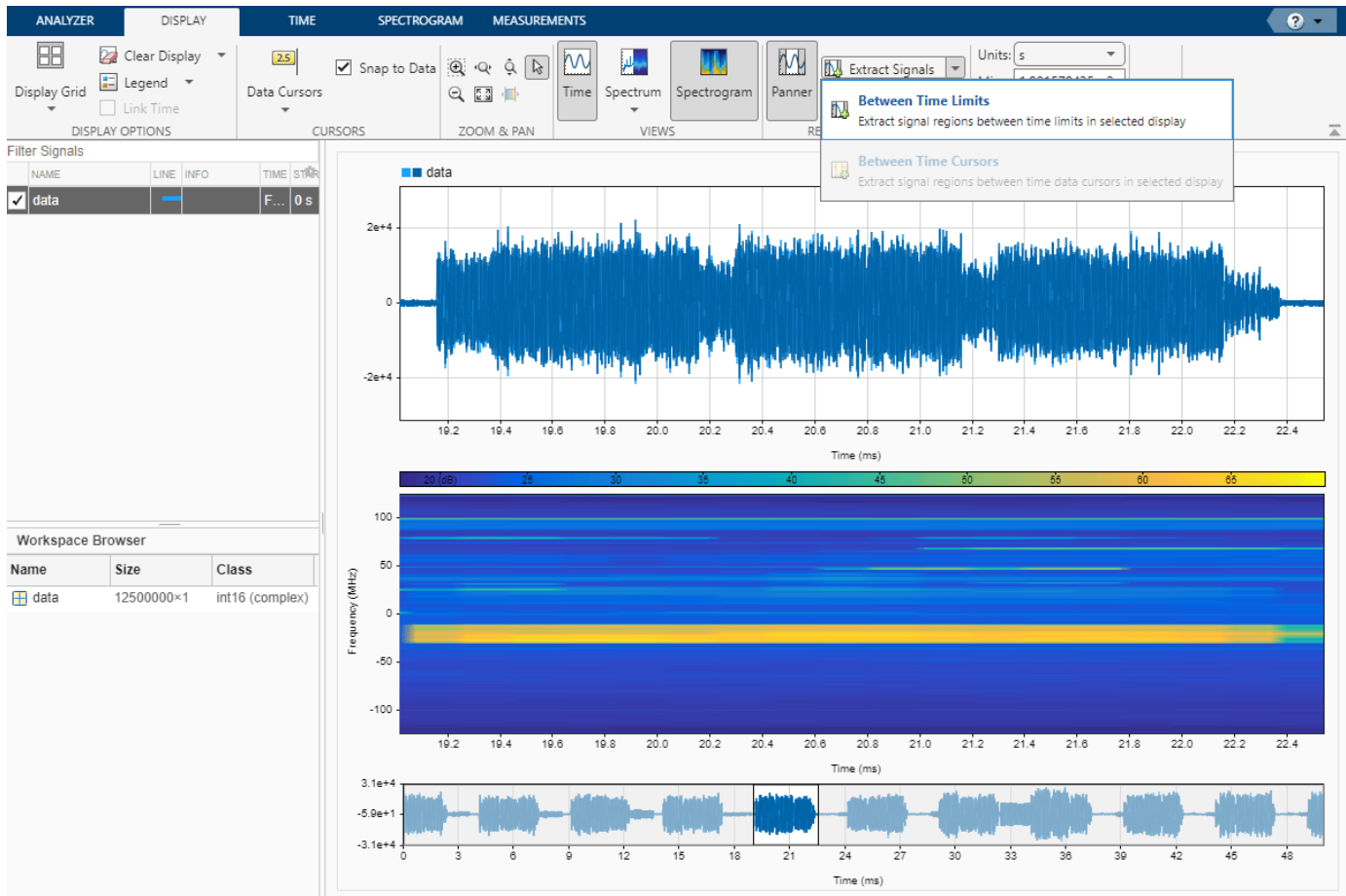
```
signalAnalyzer(data, 'SampleRate', bbrx.SampleRate);
```

This figure shows a time and spectrogram plot of 50 ms duration of captured data at 250 MHz sampling rate at a center frequency of 2.4GHz. You can clearly see bursty WLAN data in the plot.



You can further explore the captured data using the tools available in the app. For example, you can zoom into a region and extract that particular section of signal, or compare multiple captures by including additional signals from the Workspace. This figure shows the Panner tool that you can use to select a region of interest for extraction.





## See Also

### Functions

radioConfigurations | capture

### Objects

basebandReceiver

## More About

- “Capture from Frequency Band”
- “Capture Wideband Spectrum by Combining Data from Multiple Antennas” on page 2-2
- “Calibrate Radio Gain For Signal Capture” on page 2-12
- “Supported Radio Devices”
- “Using Signal Analyzer App”

## Calibrate Radio Gain For Signal Capture

This example shows how to configure the front-end radio gain of your software-defined radio (SDR) hardware for your local environment. The example configures the SDR as a baseband transceiver to transmit a test waveform and to capture data from the air. The example then calculates and analyzes the average received signal power of the captured signals using different gain values.

### Introduction

Setting the front-end radio gain correctly plays an important role when you use your radio to capture radio frequency (RF) data from the air. Too much gain can lead to saturation that results in signal distortion. Not enough gain hinders signal recovery. This example shows how to configure the radio gain of your radio device for your local environment. The example uses the `basebandTransceiver` object to show the calibration process with a known transmit signal. However, you can apply the same process for over-the-air signals by using the `basebandReceiver` object.

The main steps of the gain calibration process are:

- 1 Configure a `basebandTransceiver` object to transmit a test waveform and then to capture data at the correct sampling rate and center frequency. Alternatively, you can configure a `basebandReceiver` to capture data from the air.
- 2 Set the radio gain to the minimum gain value.
- 3 Capture the data, plot the time-domain waveform, then calculate the average signal power and peak amplitude.
- 4 In the time-domain waveform, inspect how close the peak signal values come to the full-scale amplitude.
- 5 Adjust the radio gain until the average signal power and amplitude headroom meet your requirements.

### 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” on page 1-3.

```
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 =   ;
```

### 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

```

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 ;

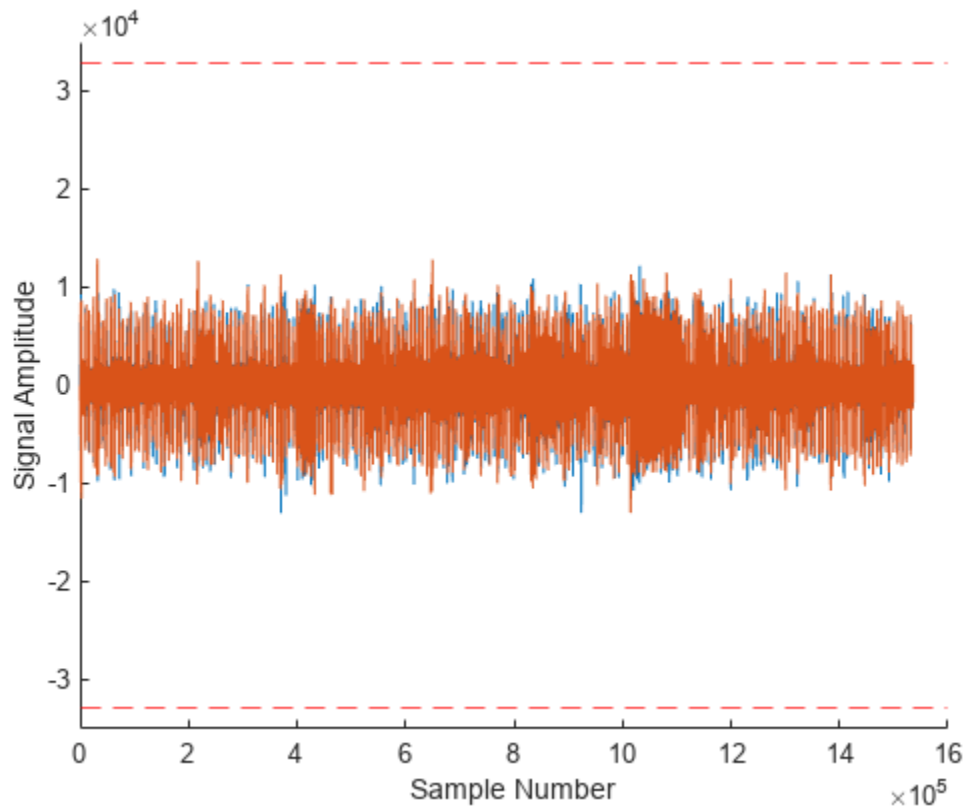
```

Create a test waveform by using the attached `50ms_lte_data.bb` file. The baseband file contains 50 ms of LTE data captured from the air at 30.72 MHz sampling rate.

```

bbfr = comm.BasebandFileReader('50ms_lte_data.bb');
waveformInfo = info(bbfr);
bbfr.SamplesPerFrame = waveformInfo.NumSamplesInData;
transmitData = bbfr();
plotData(transmitData);

```



### Calibrate Radio Gain

Set the transmit and capture sample rate and center frequencies. Set the capture duration to a value that enables the capture of enough data to show signal power variations. If you are working with a

bursty signal, set the capture duration long enough to capture a burst. For more details on how to deal with bursty signals, see the Calibrate Radio Gain For Bursty Data section of this example.

```
bbtrx.SampleRate = bbfr.SampleRate;  
bbtrx.CaptureCenterFrequency = 2.45e9;  
bbtrx.TransmitCenterFrequency = bbtrx.CaptureCenterFrequency;
```

Set the transmit radio gain and transmit the test waveform.

```
bbtrx.TransmitRadioGain = 30;  
transmit(bbtrx, transmitData, 'continuous');
```

Because the test waveform in this example does not experience significant channel effects and the transmission is continuous, a capture duration of 200 ms is sufficient. When working with signals from-the-air, set the capture duration to a value that enables the capture of enough data to show signal power variations. If the signal is bursty, aim at capturing a complete burst.

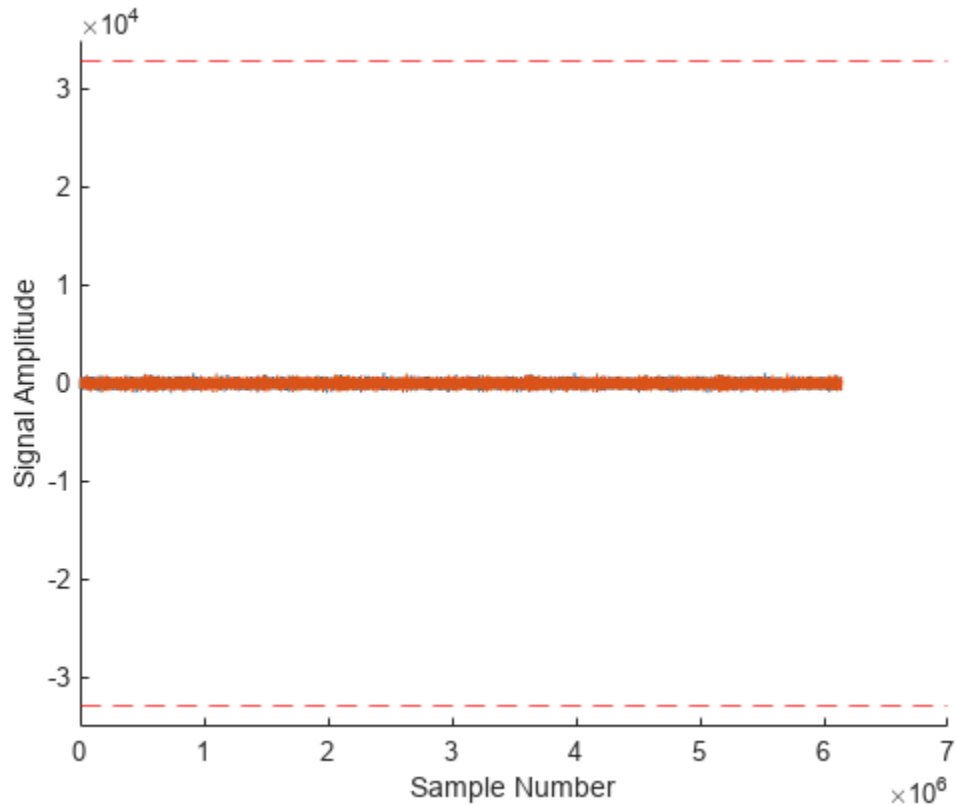
```
captureDuration = milliseconds(200);
```

Set the radio gain to a value that provides a balance between sufficient received signal power for your application and enough headroom to avoid saturation from signal amplitude changes. To avoid damaging the receiver, set the capture radio gain to an initial value of 0 dB, then capture data.

```
bbtrx.CaptureRadioGain = 0;  
data = capture(bbtrx, captureDuration);
```

Plot the captured signal and calculate the average signal power relative to the full scale of the 16-bit integer range [-32768, 32767]. In the plotted figure, the dotted red line represents the full-scale levels.

```
plotData(data);
```

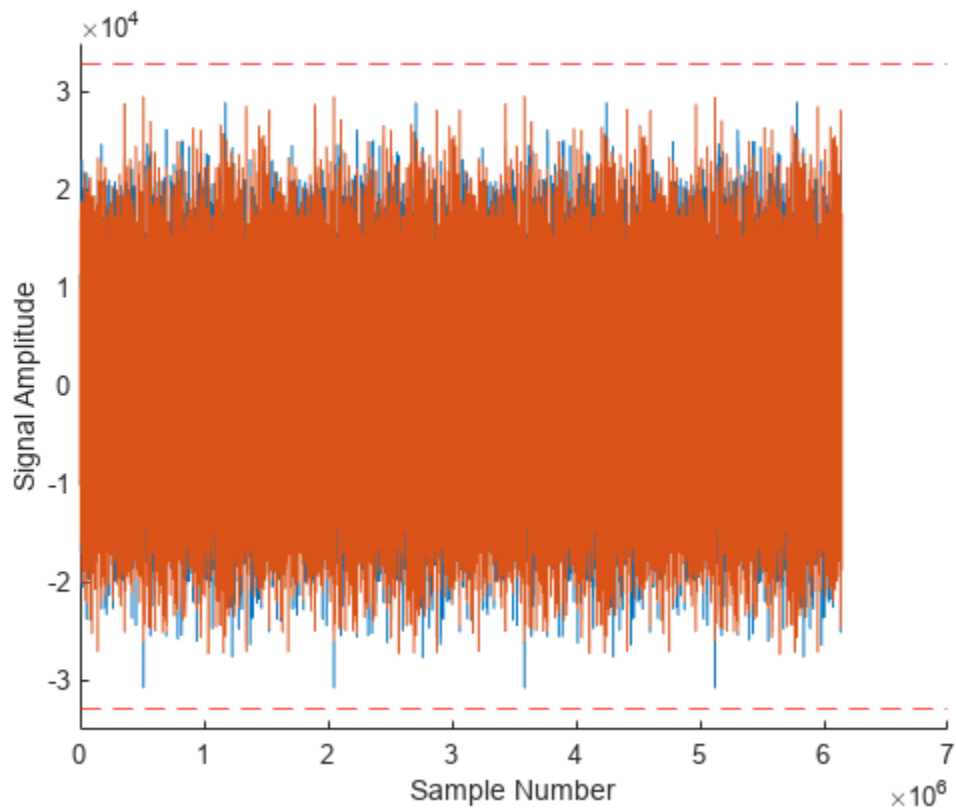


```
[pow,peak] = calcPowerAndPeak(data);
disp("Average signal power is " + pow + " dBFS, peak amplitude is " + peak + ".");
```

Average signal power is -45.7666 dBFS, peak amplitude is 1098.

Adjust the gain to a value that meets your requirement for signal power and headroom. Increase the gain to 30 dB, plot the captured data and calculate the average signal power. The captured data now occupies more of the dynamic range of the receiver than before while there is still some headroom for signal amplitude changes.

```
bbtrx.CaptureRadioGain = ;
data = capture(bbtrx,captureDuration);
plotData(data);
```



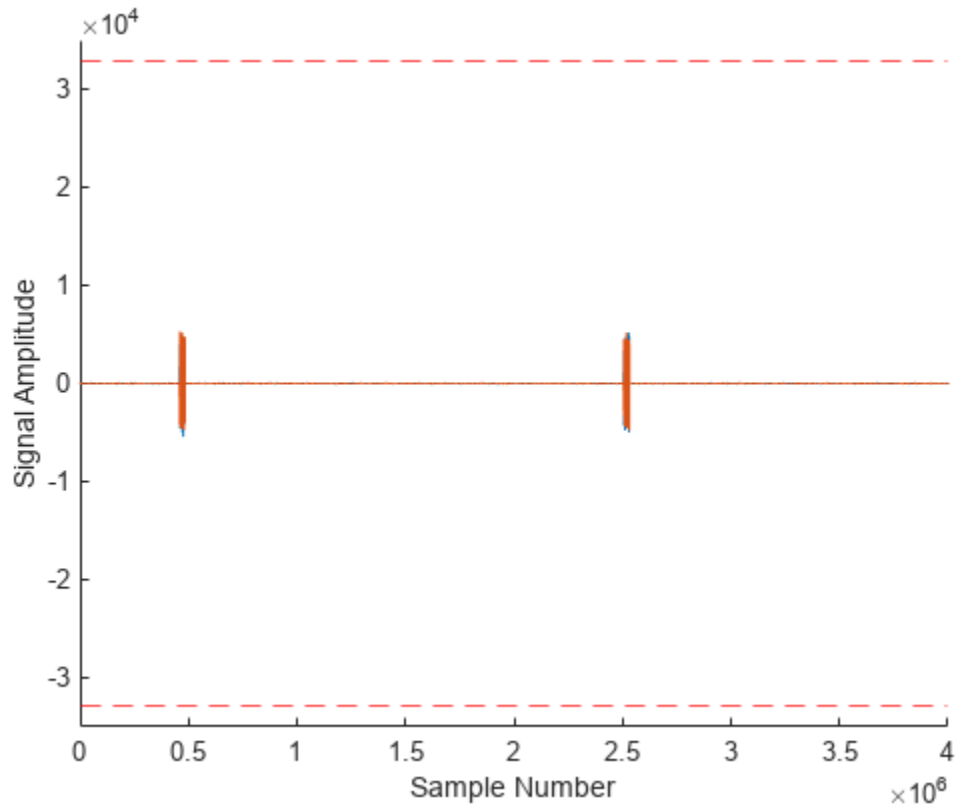
```
[pow,peak] = calcPowerAndPeak(data);  
disp("Average signal power is " + pow + " dBFS, peak amplitude is " + peak + ".");
```

Average signal power is -16.0269 dBFS, peak amplitude is 30776.

### Calibrate Radio Gain For Bursty Data

When you calibrate against a bursty signal, the average power in the signal capture does not represent the power of the signal of interest. To find the power of the signal of interest, you must isolate the signal of interest before calculating the power. Load some previously captured bursty WLAN signal and plot.

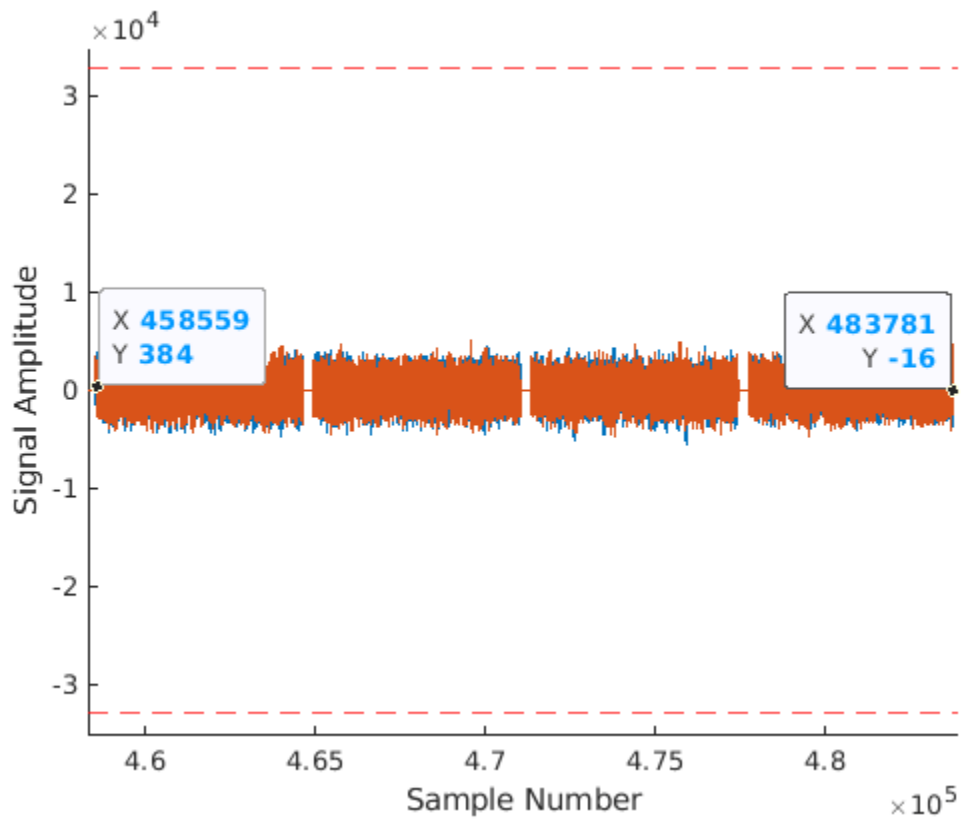
```
load("200ms_ch60_wlan_data.mat");  
plotData(data);
```



```
[pow,peak] = calcPowerAndPeak(data);
disp("Average signal power is " + pow + " dBFS, peak amplitude is " + peak + ".");
```

Average signal power is -43.3097 dBFS, peak amplitude is 5428.

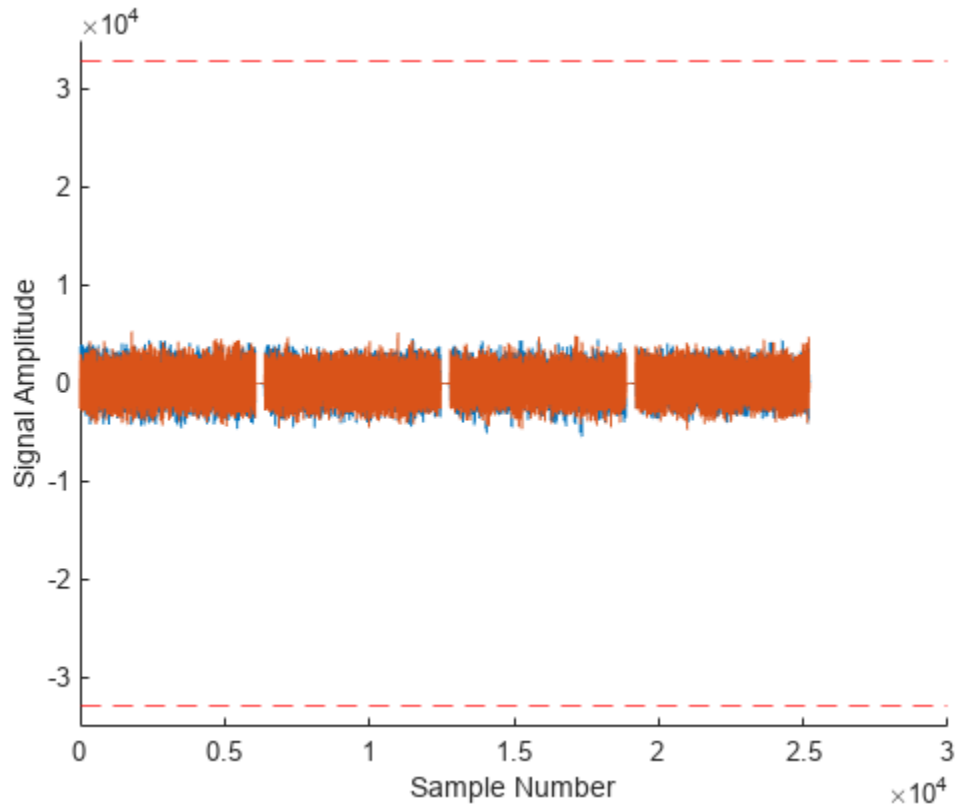
To calculate the power in the signal of interest, use the Zoom In and Data Tips tools of the plot to obtain the sample number for the start and end of the signal of interest. The figure shows the WLAN signal from the previous figure with the results of using the Zoom In and Data Tips tools that highlight the sample number for the start and end of the signal of interest.



Specify the sample numbers for the start and the end of the signal of interest using the values that are highlighted in the zoomed-in figure. Then calculate the average power and peak amplitude of the signal of interest.

```
signalStartSample =  ;  
signalEndSample =  ;  
plotData(data(signalStartSample:signalEndSample));
```





```
[pow,peak] = calcPowerAndPeak(data(signalStartSample:signalEndSample));
disp("Average signal power in the signal of interest is " + pow + " dBFS, peak amplitude is " + peak);
```

Average signal power in the signal of interest is -24.3584 dBFS, peak amplitude is 5428.

Because the average power of the signal of interest is -24 dBFS, you can apply extra gain. In this case, you can increase the radio gain by 10-15 dB.

### Further Exploration

This example uses the `basebandTransceiver` object to show the calibration process with a known transmit signal. However, you can apply the same process to calibrate your radio for any signal from the air by using the `basebandReceiver` object. Once you calibrate the radio gain, you can use the gain value when you configure the radio gain for any Wireless Testbench™ application object. For an overview of available application objects, see “Wireless Testbench Applications on SDR”.

### Local Functions

```
function plotData(data)
figure();
hold on;
ylim([-35000 35000]);
plot(real(double(data)));
plot(imag(double(data)));
yline(intmin("int16"),"r--");
yline(intmax("int16"),"r--");
ylabel("Signal Amplitude");
```

```
xlabel("Sample Number");  
end  
  
function [pow,peak] = calcPowerAndPeak(data)  
pow = pow2db(bandpower(double(data)./double(intmax("int16"))));  
peak = max(max(abs(real(data))),max(abs(imag(data))));  
end
```

### See Also

#### Functions

radioConfigurations | capture

#### Objects

basebandReceiver

### More About

- "Supported Radio Devices"

## Save Captured Signal with Metadata to Baseband File

This example shows how to configure a software-defined radio (SDR) as a baseband receiver to capture a signal from the air. The example then shows how to save to a baseband file with metadata that includes a custom label.

### 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” on page 1-3.

```
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 =   ;
```

### 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
```

Set the baseband receiver object properties. To use the maximum sample rate available for your radio, call the `hMaxSampleRate` helper function. Alternatively, you can set a custom sample rate.

```
maxSampleRate = hMaxSampleRate(radio);
bbrx.SampleRate =  ;
bbrx.CenterFrequency =  ;
bbrx.RadioGain =  ;
```

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 =  ;
```

### Capture Signal and Create Metadata with Custom Label

To capture a signal, call the `capture` function on the baseband receiver object. Specify the length of the capture. Create a metadata structure that includes a custom text label.

```
[data,timestamp] = capture(bbrx,milliseconds());
metadata.Timestamp = char(timestamp);
```

```
metadata.Antennas = char(bbrx.Antennas);  
metadata.RadioConfiguration = char(radio);  
metadata.CustomLabel = 'Data from capture example';
```

### Save Signal to Baseband File

Call the Baseband File Writer System object™ to write the captured signal to a baseband file with the metadata. To read the content of the baseband file, you can use the Baseband File Reader System object.

```
bbw = comm.BasebandFileWriter;  
bbw.FileName = 'captured_data.bb';  
bbw.SampleRate = bbrx.SampleRate;  
bbw.CenterFrequency = bbrx.CenterFrequency;  
bbw.Metadata = metadata;  
bbw(data);
```

### See Also

#### Functions

radioConfigurations | capture

#### Objects

basebandReceiver

### More About

- “Capture from Frequency Band”
- “Capture Wideband Spectrum by Combining Data from Multiple Antennas” on page 2-2
- “Calibrate Radio Gain For Signal Capture” on page 2-12
- “Supported Radio Devices”

# Spectrum Monitoring

---

## OFDM Wi-Fi Scanner Using SDR Preamble Detection

This example shows how to retrieve information about Wi-Fi® networks using a software-defined radio (SDR) and preamble detection. The example scans over the 5 GHz channels and uses an SDR preamble detector to detect and capture orthogonal frequency-division multiplexing (OFDM) packets from the air. The example then decodes the OFDM packets to determine which packets are access point (AP) beacons. The AP beacon information includes the service set identifier (SSID), media access control (MAC) address (also known as the basic SSID, or BSSID), AP channel bandwidth, and 802.11 standard used by the AP.

### Introduction

This example scans through a set of Wi-Fi channels to detect AP beacons that are transmitted on 20 MHz subchannels. The scanning procedure uses a preamble detector on an NI™ USRP™ radio.

The scanning procedure comprises of these steps.

- Configure the `preambleDetector` object with a preamble that is generated from the legacy long training field (L-LTF).
- Set the frequency band and channels for the preamble detector to scan.
- Scan each specified channel and with each successful detection of an OFDM packet, capture a waveform for a set duration.
- Process the waveform in MATLAB® by searching for beacon frames in the captured waveform and extracting relevant information from each successfully decoded beacon frame.
- Display key information about the detected APs.

### 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” on page 1-3.

```
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 =   ;
```

### 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. In subsequent runs, to speed up the execution time of the example, reuse your new workspace object.

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

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 = ;
```

To increase the capture sample rate to 40 MHz, specify an oversampling factor of 2.

```
osf = ;
pd.SampleRate = 20e6*osf;
pd.CaptureDataType = "double";
pd.ThresholdMethod = "adaptive";
```

### Configure Preamble For Radio

The 802.11 standard requires that all Wi-Fi APs must transmit OFDM beacons using non-high throughput (non-HT) packets over a 20 MHz bandwidth. Therefore, generate a 20 MHz L-LTF waveform and use one long training symbol from the generated waveform as the preamble to detect WLAN OFDM packets.

```
cbw = "CBW20";
cfg = wlanNonHTConfig(ChannelBandwidth=cbw);
lltf = wlanLLTF(cfg,OversamplingFactor=osf);
```

Extract the first long training symbol from the L-LTF waveform.

```
cyclicPrefixLength = 1.6e-6*pd.SampleRate;
trainingSymbolLength = 3.2e-6*pd.SampleRate;
preamble = lltf(cyclicPrefixLength+1:cyclicPrefixLength+trainingSymbolLength);
```

Because the preamble detector requires the preamble to be between -1 and 1, normalize and set the preamble.

```
preamble = preamble/sqrt(sum(abs(preamble).^2));
pd.Preamble = preamble;
```

To capture the entire first non-HT packet, you must set the trigger offset to a negative value. Since you created a matched filter based on the long training symbol in the L-LTF waveform, the offset is at least one legacy short training field (L-STF), one L-LTF cyclic prefix, and one long training symbol.

```
lstfLength = 8e-6*pd.SampleRate;
pd.TriggerOffset = -(lstfLength + cyclicPrefixLength + trainingSymbolLength + 5);
```

### Tune Preamble Detector

Configure the adaptive threshold gain, the adaptive threshold offset, and the radio gain values of the preamble detector for the local environment. Configuring these values requires manual tuning by exploring the trigger points provided by the `plotThreshold` function. For more information on tuning these values, see “Triggered Capture Using Preamble Detection”.

For tuning the preamble detector, specify a channel in the 5 GHz band with a known OFDM packet.

In the 5 GHz band, the valid channel numbers are 1-200. However, the valid 20 MHz control channels for APs that use the 5 GHz band are 32, 36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 149, 153, 157, 161, 165, 169, 173, 177.

```
band = 5;
channel = 100;
pd.CenterFrequency = wlanChannelFrequency(channel,band);
```

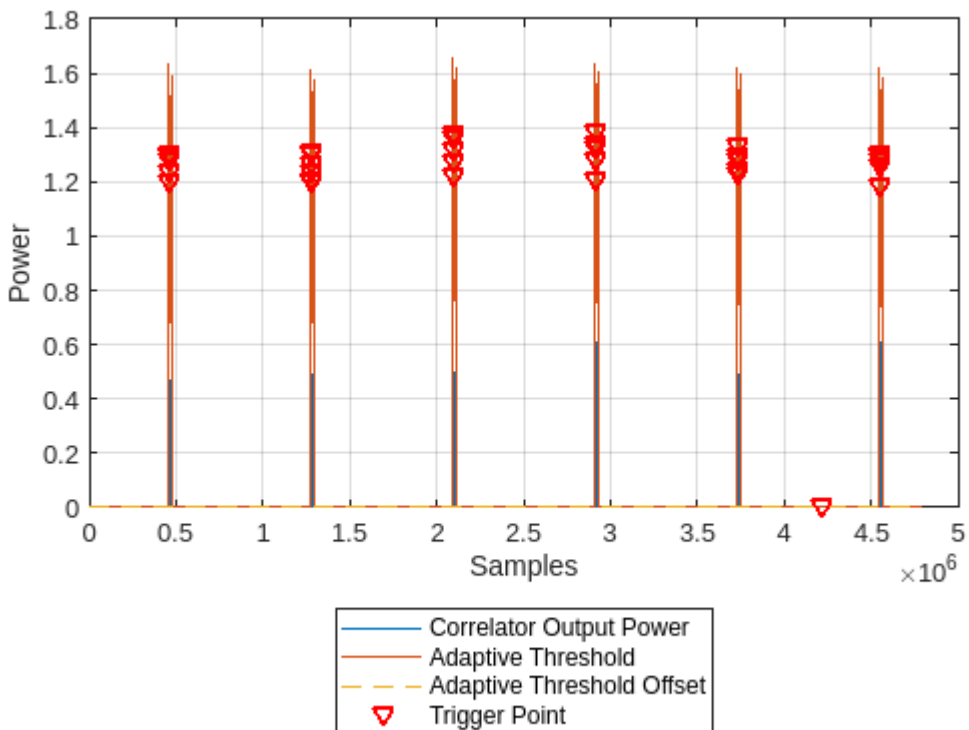
Adjust these values for tuning the preamble detector.

```
pd.AdaptiveThresholdGain = 0.3;
pd.AdaptiveThresholdOffset = 0.0005;
pd.RadioGain = 60;
```

Plot the filter output power, adaptive threshold, and trigger points of the reconfigured preamble detector. The generated figure contains two trigger points for each OFDM packet. Each trigger point corresponds to a long training symbol.

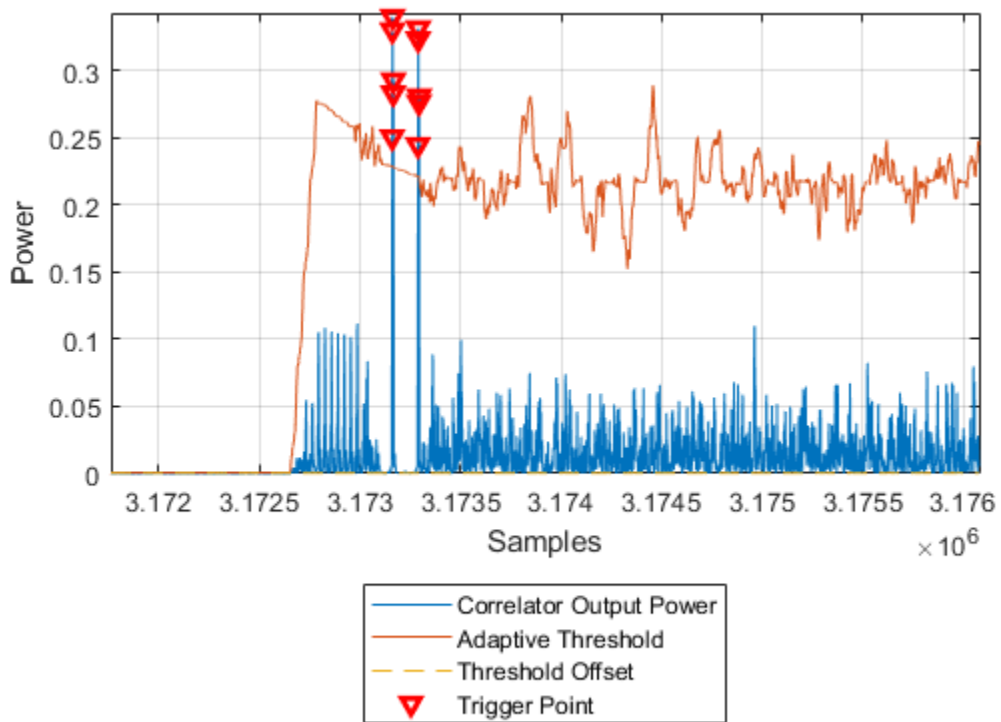
When you generate a plotThreshold figure, if you do not have at least two trigger points for each OFDM packet, readjust the adaptive threshold gain, the adaptive threshold offset, and the radio gain until there are at least two trigger points per OFDM packet.

```
captureDuration = milliseconds(120);
plotThreshold(pd,captureDuration);
```



Inspect the trigger points by zooming in along the x-axis of the plot. For example, this figure shows a zoomed-in view of an OFDM packet with the trigger points on the correlation peaks.





## Scan Wi-Fi Channels

### Specify Scanning Region

Specify the channels in the 5 GHz band for the SDR to scan.

band = 5;

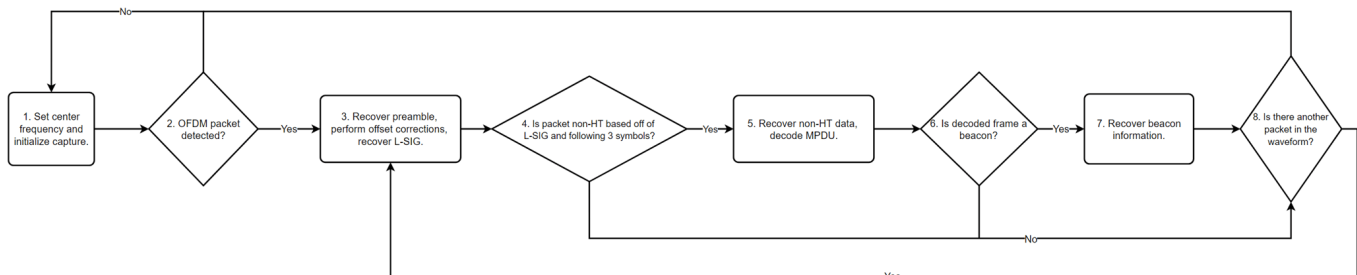
channels = [32 36 40 44 48 52 56 60];

Generate the center frequencies associated with the selected channels and band values.

centerFrequencies = wlanChannelFrequency(channels, band);

### Receiver Design

This diagram shows an overview of the receiver for scanning the selected channels and frequency band.



These steps provide further information on the diagram.

- 1 Set the center frequency of the preamble detector, then initialize the detection and capture of a waveform for a set duration.
- 2 Check if the preamble detector detects an OFDM packet.
- 3 Determine and apply frequency and timing corrections on the waveform, then attempt to recover the legacy signal (L-SIG) field bits.
- 4 Check that the packet format is non-HT.
- 5 From the recovered L-SIG, extract the modulation and coding scheme (MCS) and the length of the PLCP service data unit (PSDU). Then recover the non-HT data and subsequently decode the MAC protocol data unit (MPDU).
- 6 Using the recovered MAC frame configuration, check if the non-HT packet is a beacon.
- 7 Recover the SSID, BSSID, vendor of the AP, SNR, primary 20 MHz channel, current channel center frequency index, supported channel width, frequency band, and wireless standard used by the AP.
- 8 Check if the waveform contains another packet that you can decode.

### Initialize Variables

When you call the `capture` function to detect and capture a signal, you must specify the length of the capture and the signal detection timeout. Since beacons transmit every 100 ms, set `captureLength` to `milliseconds(100)` and `timeout` to `milliseconds(100)`.

```
captureLength = milliseconds(100);  
timeout = milliseconds(100);
```


Create a structure (APs) for storing this information for each successfully decoded beacon.

- SSID
- BSSID
- Vendor of AP
- Signal-to-noise ratio (SNR)
- Primary 20 MHz channel
- Current channel center frequency
- Channel width
- Frequency band
- Operating mode supported by the AP
- MAC frame configuration
- Waveform in which the beacon exists
- Index value at which the non-HT beacon packet begins in the captured waveform

```
APs = struct(...  
    "SSID", [], "BSSID", [], "Vendor", [], "SNR_dB", [], "Beacon_Channel", [], ...  
    "Operating_Channel", [], "Channel_Width_MHz", [], "Band", [], "Mode", [], ...  
    "MAC_Config", wlanMACFrameConfig, "Waveform", [], "Offset", []);
```

To determine the hardware manufacturer of the AP, select the `retrieveVendorInfo` box. Selecting the `retrieveVendorInfo` box downloads the organizationally unique identifier (OUI) CSV file from the IEEE® Registration Authority website for vendor AP identification.

```

retrieveVendorInfo = ;
counter = 1;
ind = wlanFieldIndices(cfg);

% Begin scanning and decoding for specified channels.
for i = 1:length(centerFrequencies)

    pd.CenterFrequency = centerFrequencies(i);

    fprintf("Scanning channel %d on band %.1f.\n",channels(i),band);
    [capturedData, ~, ~, status] = capture(pd, captureLength, timeout);

    if ~status
        % If no non-HT packet is decoded, go to next channel.
        fprintf("No non-HT packet detected on channel %d in band %.1f.\n",channels(i),band);
        continue;
    else
        fprintf("<strong>Non-HT packet detected on channel %d in band %.1f.</strong>\n",channels
    end
    % Resample the captured data to 20 MHz for beacon processing.
    capturedData = resample(capturedData,1,osf);
    searchOffset = 0;
    while searchOffset<length(capturedData)

        % recoverPreamble detects a packet and performs analysis of the non-HT preamble.
        [preambleStatus,res] = recoverPreamble(capturedData,cbw,searchOffset);

        if matches(preambleStatus,"No packet detected")
            break;
        end

        % Retrieve synchronized data and scale it with LSTF power as done
        % in the recoverPreamble function.
        syncData = capturedData(res.PacketOffset+1:end)./sqrt(res.LSTFPower);
        syncData = frequencyOffset(syncData,pd.SampleRate/osf,-res.CFOEstimate);

        % Need only 4 OFDM symbols (LSIG + 3 more symbols) following LLTF
        % for format detection
        fmtDetect = syncData(ind.LSIG(1):(ind.LSIG(2)+4e-6*pd.SampleRate/osf*3));

        [LSIGBits, failcheck] = wlanLSIGRecover(fmtDetect(1:4e-6*pd.SampleRate/osf*1), ...
            res.ChanEstNonHT,res.NoiseEstNonHT,cbw);

        if ~failcheck
            format = wlanFormatDetect(fmtDetect,res.ChanEstNonHT,res.NoiseEstNonHT,cbw);
            if matches(format,"Non-HT")

                % Extract MCS from first 3 bits of L-SIG.
                rate = double(bit2int(LSIGBits(1:3),3));
                if rate <= 1
                    cfg.MCS = rate + 6;
                else
                    cfg.MCS = mod(rate,6);
                end
            end
        end
    end
end

```

```

end

% Determine PSDU length from L-SIG.
cfg.PSDULength = double(bit2int(LSIGBits(6:17),12,0));
ind.NonHTData = wlanFieldIndices(cfg,"NonHT-Data");

if double(ind.NonHTData(2)-ind.NonHTData(1))> ...
    length(syncData(ind.NonHTData(1):end))
    % Exit while loop as full packet not captured.
    break;
end

nonHTData = syncData(ind.NonHTData(1):ind.NonHTData(2));
bitsData = wlanNonHTDataRecover(nonHTData,res.ChanEstNonHT, ...
    res.NoiseEstNonHT,cfg);
[cfgMAC,~,decodeStatus] = wlanMPDUDecode(bitsData,cfg, ...
    SuppressWarnings=true);

% Extract information about channel from the beacon.
if ~decodeStatus && matches(cfgMAC.FrameType,"Beacon")
    fprintf("Beacon detected on channel %d in band %.1f.\n",channels(i),band);

    % Populate the table with information about the beacon.
    if isempty(cfgMAC.ManagementConfig.SSID)
        APs(counter).SSID = "Hidden";
    else
        APs(counter).SSID = string(cfgMAC.ManagementConfig.SSID);
    end

    APs(counter).BSSID = string(cfgMAC.Address3);
    if retrieveVendorInfo
        APs(counter).Vendor = determineVendor(cfgMAC.Address3);
    else
        APs(counter).Vendor = "Skipped"; %#ok<UNRCH>
    end
    [APs(counter).Mode, APs(counter).Channel_Width_MHz, operatingChannel] = ...
        determineMode(cfgMAC.ManagementConfig.InformationElements);

    if isempty(operatingChannel)
        % Default to scanning channel if operating channel
        % cannot be determined.
        operatingChannel = channels(i);
    end

    APs(counter).Beacon_Channel = channels(i);
    APs(counter).Operating_Channel = operatingChannel;
    APs(counter).SNR_dB = res.LLTFSSNR;
    APs(counter).MAC_Config = cfgMAC;
    APs(counter).Offset = res.PacketOffset;
    APs(counter).Waveform = capturedData;
    counter = counter + 1;
end

% Shift packet search offset for next iteration of while loop.
searchOffset = res.PacketOffset + double(ind.NonHTData(2));
else
    % Packet is NOT non-HT; shift packet search offset by 10 OFDM symbols (minimum
    % packet length of non-HT) for next iteration of while loop.
    searchOffset = res.PacketOffset + 4e-6*pd.SampleRate/osf*10;
end

```

```
        end
    else
        % L-SIG recovery failed; shift packet search offset by 10 OFDM symbols (minimum
        % packet length of non-HT) for next iteration of while loop.
        searchOffset = res.PacketOffset + 4e-6*pd.SampleRate/osf*10;
    end
end
end
```

Scanning channel 32 on band 5.0.

No non-HT packet detected on channel 32 in band 5.0.

Scanning channel 36 on band 5.0.

No non-HT packet detected on channel 36 in band 5.0.

Scanning channel 40 on band 5.0.

Non-HT packet detected on channel 40 in band 5.0.

Scanning channel 44 on band 5.0.

No non-HT packet detected on channel 44 in band 5.0.

Scanning channel 48 on band 5.0.

Non-HT packet detected on channel 48 in band 5.0.

Beacon detected on channel 48 in band 5.0.

Beacon detected on channel 48 in band 5.0.

Beacon detected on channel 48 in band 5.0.

Beacon detected on channel 48 in band 5.0.

Beacon detected on channel 48 in band 5.0.

Scanning channel 52 on band 5.0.

No non-HT packet detected on channel 52 in band 5.0.

Scanning channel 56 on band 5.0.

No non-HT packet detected on channel 56 in band 5.0.

Scanning channel 60 on band 5.0.

Non-HT packet detected on channel 60 in band 5.0.

Beacon detected on channel 60 in band 5.0.

Beacon detected on channel 60 in band 5.0.

Beacon detected on channel 60 in band 5.0.

Scanning channel 64 on band 5.0.

No non-HT packet detected on channel 64 in band 5.0.

Scanning channel 100 on band 5.0.

Non-HT packet detected on channel 100 in band 5.0.

Beacon detected on channel 100 in band 5.0.

Beacon detected on channel 100 in band 5.0.

Beacon detected on channel 100 in band 5.0.

Beacon detected on channel 100 in band 5.0.  
Beacon detected on channel 100 in band 5.0.

Scanning channel 104 on band 5.0.

No non-HT packet detected on channel 104 in band 5.0.

Scanning channel 108 on band 5.0.

No non-HT packet detected on channel 108 in band 5.0.

Scanning channel 112 on band 5.0.

Non-HT packet detected on channel 112 in band 5.0.

Scanning channel 116 on band 5.0.

No non-HT packet detected on channel 116 in band 5.0.

Scanning channel 120 on band 5.0.

Non-HT packet detected on channel 120 in band 5.0.

Beacon detected on channel 120 in band 5.0.  
Beacon detected on channel 120 in band 5.0.  
Beacon detected on channel 120 in band 5.0.  
Beacon detected on channel 120 in band 5.0.

Scanning channel 124 on band 5.0.

No non-HT packet detected on channel 124 in band 5.0.

Scanning channel 128 on band 5.0.

No non-HT packet detected on channel 128 in band 5.0.

Scanning channel 132 on band 5.0.

No non-HT packet detected on channel 132 in band 5.0.

Scanning channel 136 on band 5.0.

No non-HT packet detected on channel 136 in band 5.0.

Scanning channel 140 on band 5.0.

No non-HT packet detected on channel 140 in band 5.0.

Scanning channel 144 on band 5.0.

No non-HT packet detected on channel 144 in band 5.0.

Scanning channel 149 on band 5.0.

No non-HT packet detected on channel 149 in band 5.0.

Scanning channel 153 on band 5.0.

No non-HT packet detected on channel 153 in band 5.0.

Scanning channel 157 on band 5.0.

No non-HT packet detected on channel 157 in band 5.0.

Scanning channel 161 on band 5.0.

No non-HT packet detected on channel 161 in band 5.0.

Scanning channel 165 on band 5.0.

No non-HT packet detected on channel 165 in band 5.0.

Scanning channel 169 on band 5.0.

No non-HT packet detected on channel 169 in band 5.0.

Scanning channel 173 on band 5.0.

No non-HT packet detected on channel 173 in band 5.0.

Scanning channel 177 on band 5.0.

No non-HT packet detected on channel 177 in band 5.0.

Convert the APs structure to a table and display the information specified in step 7 on page 3-6 by using the local function `generateBeaconTable`.

```
detectedBeaconsInfo = generateBeaconTable(APs,band)
```

```
detectedBeaconsInfo=17x9 table
```

SSID	BSSID	Vendor	SNR (dB)	Primary
"ClassForKids Secure"	"9A1898BEB142"	"Unknown"	13.532	
"ClassForKids Guest"	"9E1898BEB142"	"Unknown"	14.314	
"ClassForKids Music"	"921898BEB142"	"Unknown"	13.211	
"Test SSID"	"961898BEB142"	"Unknown"	13.671	
"Hidden"	"A61898BEB142"	"Unknown"	13.902	
"w-inside"	"B0B867F3D9B0"	"Hewlett Packard Enterprise"	12.655	
"w-mobile"	"B0B867F3D9B1"	"Hewlett Packard Enterprise"	13.278	
"w-guest"	"B0B867F3D9B2"	"Hewlett Packard Enterprise"	13.225	
"wlan1234_5"	"04D4C451C584"	"ASUSTek COMPUTER INC."	37.074	
"wlan1234_5"	"04D4C451C584"	"ASUSTek COMPUTER INC."	33.754	
"wlan1234_5"	"04D4C451C584"	"ASUSTek COMPUTER INC."	34.933	
"wlan1234_5"	"04D4C451C584"	"ASUSTek COMPUTER INC."	36.619	
"wlan1234_5"	"04D4C451C584"	"ASUSTek COMPUTER INC."	36.484	
"w-inside"	"B0B867F6B2D0"	"Hewlett Packard Enterprise"	29.415	
"w-mobile"	"B0B867F6B2D1"	"Hewlett Packard Enterprise"	29.295	
"w-guest"	"B0B867F6B2D2"	"Hewlett Packard Enterprise"	28.261	
:				

### Further Exploration

- The `detectedBeaconsInfo` table shows only key information about the APs. To get further information about the beacons, such as data rates supported by the AP, explore the MAC frame configuration in the APs structure.
- If you have access to a configurable AP, change the channel width of your AP and rerun the example to confirm the channel width.

### Local Functions

These functions assist in processing the incoming beacons.

```

function vendor = determineVendor(mac)
% DETERMINEVENDOR returns the vendor name of the AP by extracting the
% organizationally unique identifier (OUI) from the specified MAC address.

persistent ouis

vendor = strings(0);
try
    if isempty(ouis)
        if ~exist("oui.csv","file")
            disp("Downloading oui.csv from IEEE Registration Authority...")
            % Increase websave timeout if necessary
            options = weboptions("Timeout",5);
            websave("oui.csv","http://standards-oui.ieee.org/oui/oui.csv",options);
        end
        ouis = readtable("oui.csv",VariableNamingRule="preserve");
    end

    % Extract OUI from MAC Address.
    oui = mac(1:6);

    % Extract vendors name based on OUI.
    vendor = string(cell2mat(ouis.("Organization Name")(matches(ouis.Assignment,oui))));

catch ME
    % Rethrow caught error as warning.
    warning(ME.message+"\nTo skip the determineVendor function call, set retrieveVendorInfo to false");
end

if isempty(vendor)
    vendor = "Unknown";
end

end

function [mode,bw,operatingChannel] = determineMode(informationElements)
% DETERMINEMODE determines the 802.11 standard that the AP uses.
% The function checks for the presence of HT, VHT, and HE capability
% elements and determines the 802.11 standard that the AP uses. The element
% IDs are defined in IEEE Std 802.11-2020 and IEEE Std 802.11ax-2021.

elementIDs = cell2mat(informationElements(:,1));
IDs = elementIDs(:,1);

if any(IDs==255)
    if any(elementIDs(IDs==255,2)==35)
        % HE Packet Format
        mode = "802.11ax";
    else
        mode = "Unknown";
    end
    vhtElement = informationElements{IDs==192,2};
    htElement = informationElements{IDs==61,2};
    [bw,operatingChannel] = determineChannelWidth(htElement,vhtElement);
elseif any(IDs==191)
    % VHT Packet Format
    mode = "802.11ac";
end

```



```

    vhtElement = informationElements{IDs==192,2};
    htElement = informationElements{IDs==61,2};
    [bw,operatingChannel] = determineChannelWidth(htElement,vhtElement);
elseif any(IDs==45)
    % HT Packet Format
    mode = "802.11n";
    htElement = informationElements{IDs==61,2};
    [bw,operatingChannel] = determineChannelWidth(htElement);
else
    % Non-HT Packet Format
    % Exclude b as only DSSS is supported
    mode = "802.11a/g/j/p";
    bw = "Unknown";
    operatingChannel = [];
end
end

function [bw,operatingChannel] = determineChannelWidth(htElement,varargin)
% DETERMINECHANNELWIDTH returns the bandwidth of the channel from the
% beacons HT/VHT operation information elements as defined in IEEE Std 802.11-2020
% Section 9.4.2.56 and Section 9.4.2.158.

msbFirst = false;

% IEEE Std 802.11-2020 Figure 9-382 and Table 9-190 define each bit in
% htOperationInfoBits
% Convert to bits to get STA channel width value in 3rd bit.
htOperationInfoBits = int2bit(htElement(2),5*8,msbFirst);
operatingChannel = 0;

if nargin == 2
    % IEEE Std 802.11-2020 Figure 9-163 and Table 9-274 define each octet
    % in vhtElement
    vhtElement = varargin{1};

    % VHT Operation Channel Width Field
    CW = vhtElement(1);
    % Channel Center Frequency Segment 0
    CCFS0 = vhtElement(2);
    % Channel Center Frequency Segment 1
    CCFS1 = vhtElement(3);

    % IEEE Std 802.11-2020 Table 11-23 defines the logic below
    if htOperationInfoBits(3) == 0
        bw = "20";
        operatingChannel = CCFS0;
    elseif CW == 0
        % HT Operation Channel Width Field is 1
        bw = "40";
        operatingChannel = CCFS0;
    elseif CCFS1 == 0
        % HT Operation Channel Width Field is 1 and
        % VHT Operation Channel Width Field is 1
        bw = "80";
        operatingChannel = CCFS0;
    elseif abs(CCFS1 - CCFS0) == 8
        % HT Operation Channel Width Field is 1 and

```

```

        % VHT Operation Channel Width Field is 1 and
        % CCFS1 is greater than 0
        bw = "160";
        operatingChannel = CCFS1;
    else
        % HT Operation Channel Width Field is 1 and
        % VHT Operation Channel Width Field is 1 and
        % CCFS1 is greater than 0 and
        % |CCFS1 - CCFS0| is greater than 16
        bw = "80+80";
    end
end

if operatingChannel == 0
    if htOperationInfoBits(3) == 1
        bw = "40";
        secondaryChannelOffset = bit2int(htOperationInfoBits(1:2),2,false);
        if secondaryChannelOffset == 1
            % Secondary Channel is above the primary channel.
            operatingChannel = htElement(1) + 2;
        elseif secondaryChannelOffset == 3
            % Secondary Channel is below the primary channel.
            operatingChannel = htElement(1) - 2;
        else
            warning("Could not determine operating channel.")
        end
    else
        bw = "20";
        operatingChannel = htElement(1);
    end
end

end

function tbl = generateBeaconTable(APs,band)
% GENERATEBEACONTABLE converts the access point structure to a table and
% cleans up the variable names.

tbl = struct2table(APs,"AsArray",true);
tbl.Band = repmat(band,length(tbl.SSID),1);
tbl = renamevars(tbl,["SNR_dB","Beacon_Channel","Operating_Channel","Channel_Width_MHz"], ...
    ["SNR (dB)","Primary 20 MHz Channel","Current Channel Center Frequency Index", ...
    "Channel Width (MHz)"]);
tbl = tbl(:,1:9);

end

```

## See Also

### Functions

radioConfigurations

### Objects

preambleDetector

## **More About**

- “Triggered WLAN Waveform Capture Using Preamble Detection” on page 3-16
- “Connect and Set Up NI USRP Radios” on page 1-3
- “Supported Radio Devices”

## Triggered WLAN Waveform Capture Using Preamble Detection

This example shows how to use a software-defined-radio (SDR) to capture a WLAN waveform from the air by detecting the legacy long training field (L-LTF).

### 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” on page 1-3.

```
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 =   ;
```

### Configure WLAN Channel Information

Select a frequency band and channel to search for a WLAN waveform.

These are the valid WLAN frequency bands.

- 2.4 GHz
- 5 GHz

These are the valid WLAN channel numbers.

- 1-14 for the 2.4 GHz band
- 1-200 for the 5 GHz band. However, the valid 20 MHz control channels for access points using 5 GHz are 32, 36, 40, 44, 48, 52, 56, 60, 64, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 149, 153, 157, 161, 165, 169, 173, and 177.

```
band =  ;
```

```
channel =  ;
```

### Configure WLAN Preamble Detector

Load the WLAN preamble sequence and center frequency information. The preamble consists of one long training symbol from a 20 MHz L-LTF waveform, sampled at 40 MHz. Normalize the preamble sequence to be between -1 and 1.

```
load("TriggeredWLANData.mat");
preamble = preamble/sqrt(sum(abs(preamble).^2));
```

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. In subsequent runs, to speed up the execution time of the example, reuse your new workspace object.

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

Set the RF properties of the preamble detector.

```
pd.SampleRate = 40e6;
pd.CenterFrequency = getWLANCenterFrequency(WLANFrequenciesMap, band, channel);
```

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);
pd.Antennas = ;
```

Configure the filter coefficients for preamble detection.

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

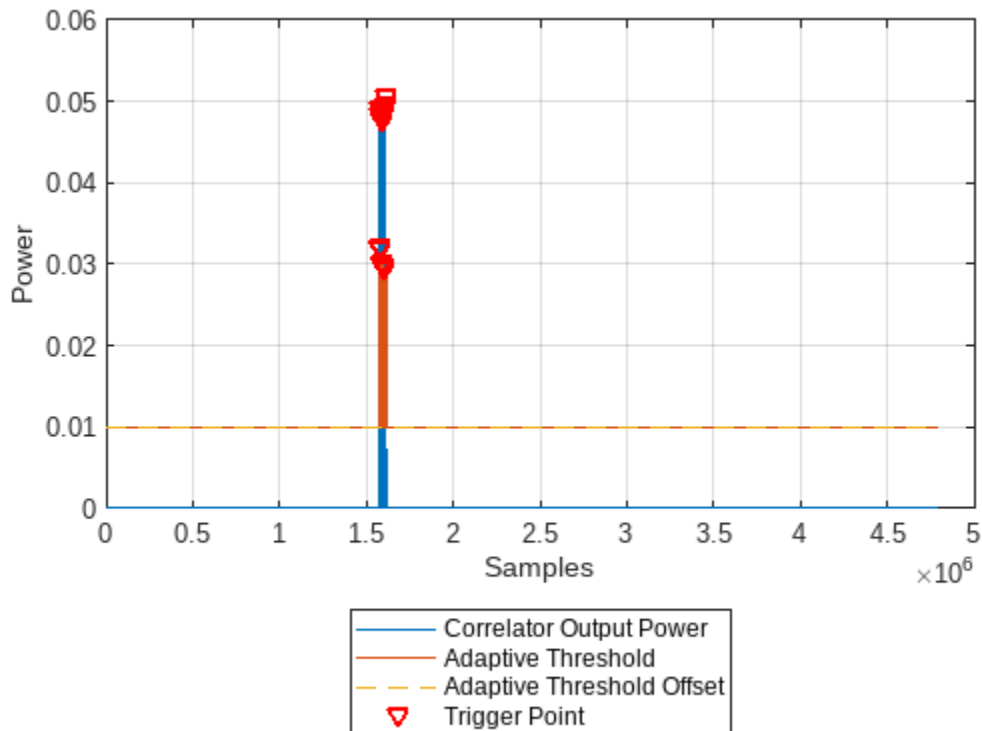
Set the threshold method to adaptive. To include the preamble sequence in the captured waveform, set the trigger offset to a negative value.

```
pd.ThresholdMethod = "adaptive";
pd.TriggerOffset = ;
```

### Configure Adaptive Threshold for Triggering

Set the adaptive threshold gain, adaptive threshold offset, and radio gain values of the preamble detector for the local environment. Use the `plotThreshold` function to analyze the behavior of the detector by plotting 120 ms of data. The function plots the correlator output, adaptive threshold, and detection points. The correlator output contains two peaks for each OFDM packet. Each peak corresponds to a long training symbol. Adjust the adaptive threshold gain, adaptive threshold offset, and radio gain values such that the trigger points occur only on the correlator output peaks. For more information on tuning these values, see "Triggered Capture Using Preamble Detection".

```
pd.AdaptiveThresholdGain = ;
pd.AdaptiveThresholdOffset = ;
pd.RadioGain = ;
plotThreshold(pd, milliseconds(120));
```



### Capture WLAN Signal

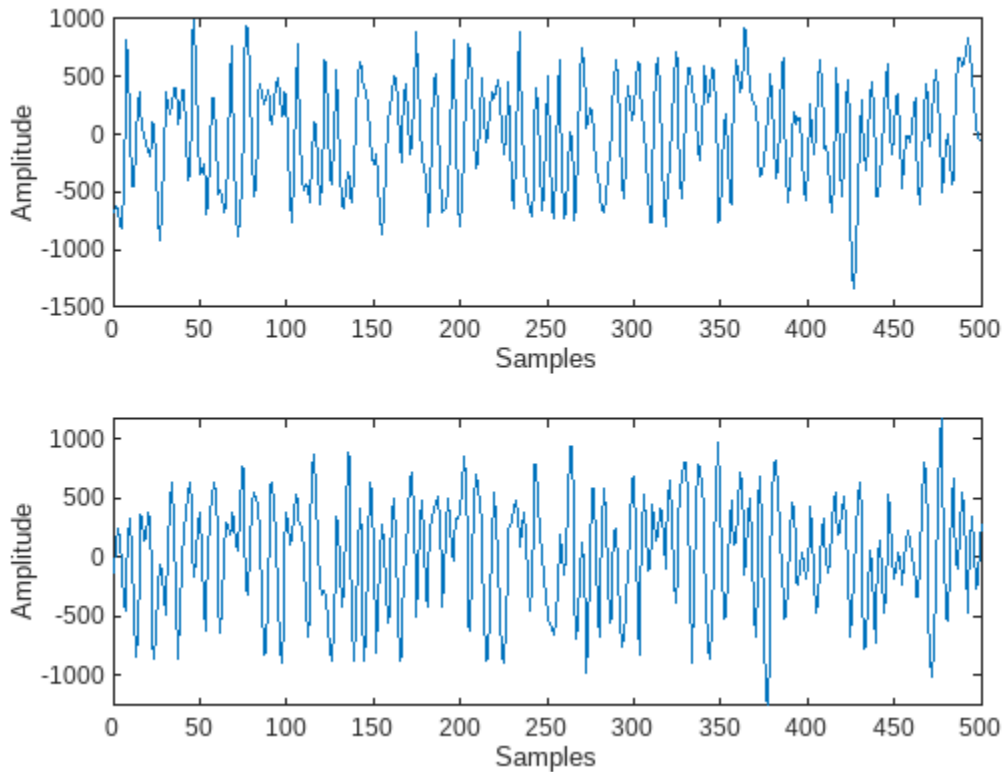
Use the capture function to capture data with the configured preamble detector. Because WLAN beacons are transmitted every 100 ms, capture 100 ms of data with a 100 ms timeout.

```
recordLen = milliseconds(100);
timeout = milliseconds(100);
[data, timestamp, ~, status] = capture(pd, recordLen, timeout);
```

If detection is successful, plot the first 500 samples.

```
if ~status
    disp("Detection failed.")
else
    disp(" WLAN signal detected at " + string(timestamp) + ".");
    figure();
    subplot(2,1,1); plot(real(double(data(1:500)))); ...
        xlabel("Samples"); ylabel("Amplitude");
    subplot(2,1,2); plot(imag(double(data(1:500)))); ...
        xlabel("Samples"); ylabel("Amplitude");
end
```

WLAN signal detected at 06-Jan-2023 09:15:27.



### Local Functions

```
function frequency = getWLANCenterFrequency(WLANFrequenciesMap,band,channel)
% Look up center frequency according to band and channel
channelSelectKey = band + "GHz:" + channel;
frequency = WLANFrequenciesMap(channelSelectKey);
end
```

### See Also

#### Functions

radioConfigurations

#### Objects

preambleDetector

### More About

- “Triggered Capture Using Preamble Detection”
- “OFDM Wi-Fi Scanner Using SDR Preamble Detection” on page 3-2
- “Supported Radio Devices”

