

Wireless Testbench™

User's Guide



MATLAB®

R2022a



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™ User's Guide

© COPYRIGHT 2022 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History


March 2022	Online only	New for Version 1.0 (Release 2022a)
------------	-------------	-------------------------------------

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.

- 1** On the MATLAB® **Home** tab, in the **Environment** section, click **Add-Ons > Get Hardware Support Packages**.
- 2** In 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 several 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, 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. The SD card must have at least 14.8 GB of storage and must be formatted as a single 14.8 GB (or greater) partition. If the host computer does not have an integrated card reader, use an external USB SD card reader. If you have already set up an SD card with the Linux system file image, select the **I have SD Card Setup and Connected to Hardware** option on the Radio Setup wizard and click **Next** to proceed with the hardware setup.

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 network interface card (NIC) for connecting to 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® (it is in effect even after you reboot the machine). 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 after 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
```

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 network buffer sizes of `wmem` and `rmem` on Linux systems.

To update the *FastSendDatagramThreshold* register key on Windows or the network buffer sizes of *wmem* and *rmem* on Linux systems, click **Press here to set** in the **Validate** screen on the Radio Setup wizard.

Make Buffer Configuration Changes Persistent

The buffer configuration on the host computer is persistent on Windows (it is in effect even after you reboot the machine). However, on Linux systems, unless you make buffer configuration changes persistent, a system reboot resets the network buffer sizes and loses the host-to-radio connection.

On Linux systems, the maximum network buffer sizes are capped by the sysctl values *net.core.rmem_max* and *net.core.wmem_max*. To make the network buffer sizes persistent, run these commands on the 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`
```

If you encounter any issues on Windows when setting the *FastSendDatagramThreshold* register key from the Radio Setup wizard, follow these steps to make the changes persistent.

- You can locate the *FastSendDatagramThreshold.reg* file at *UHD_INSTALL/win64* path by running these commands.

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

- Double click and run the *FastSendDatagramThreshold.reg* file.
- Rebooting the machine after you change the register key is recommended.

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 1-6
“Set Baseband Sample Rate” on page 1-6
“Farrow Rate Converter” on page 1-7

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"> • 122.88 MHz • 125.00 MHz • 153.60 MHz
USRP N320	<ul style="list-style-type: none"> • 200.00 MHz • 245.76 MHz • 250.00 MHz
USRP N321	<ul style="list-style-type: none"> • 200.00 MHz • 245.76 MHz • 250.00 MHz

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 $MCR/SampleRate$ 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
	3

Radio	Supported Decimation or Interpolation Factor
USRP N321	Even integer in the range from 4 to 256
	Integer multiple of 4 in the range from 256 to 512
	Integer multiple of 8 in the range from 512 to 1020

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 $MCR/2$.

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 from Frequency Band with Multiple Antennas

This example shows how to use a software-defined radio (SDR) to capture data from a specified frequency band using multiple antennas. The example then 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” on page 1-3.

```
radios = radioConfigurations;
```

Specify the name of a saved radio setup configuration to use with this example.

```
radioName = radios(1).Name;
```

Specify Frequency Band

Specify the start and the end of the frequency band. By default, this example captures the 470–694 MHz frequency band, typically allocated to TV broadcasting channels 21–48.

```
frequencyBand.Start = 470000000;
frequencyBand.End = 694000000;
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. To speed up the execution time of this example upon subsequent runs, reuse the workspace object from the first run of the example.

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

To capture the full width of the frequency band with two antennas:

- Set the `SampleRate` property to a value that is greater than or equal to half the width of the frequency band.
- Set the `CenterFrequency` property to the values that correspond to the middle of each half of the frequency band.
- Set the `Antennas` property to values that correspond to two independently tunable antennas on the SDR.

```
bbrx.SampleRate = 122.88e6;
bbrx.CenterFrequency = [frequencyBand.MidPoint - frequencyBand.Width/4, ...
    frequencyBand.MidPoint + frequencyBand.Width/4];
bbrx.Antennas = [RF0:RX2, RF1:RX2];
```

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

```
bbrx.RadioGain = 60;
bbrx.CaptureDataType = "double";
```

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

Resample then Shift Captured Data

Resample then shift the captured data channels by modulating with a carrier waveform.

```
resampledData = resample(data, length(bbrx.Antennas), 1);

for antenna = 1 : length(bbrx.Antennas)
    carrierGen = dsp.SineWave(...
        Frequency=bbrx.CenterFrequency(antenna) - frequencyBand.MidPoint, ...
        SampleRate=bbrx.SampleRate*length(bbrx.Antennas), ...
        SamplesPerFrame=length(resampledData), ...
        ComplexOutput=true);
    resampledData(:, antenna) = resampledData(:, antenna).*carrierGen();
end
```

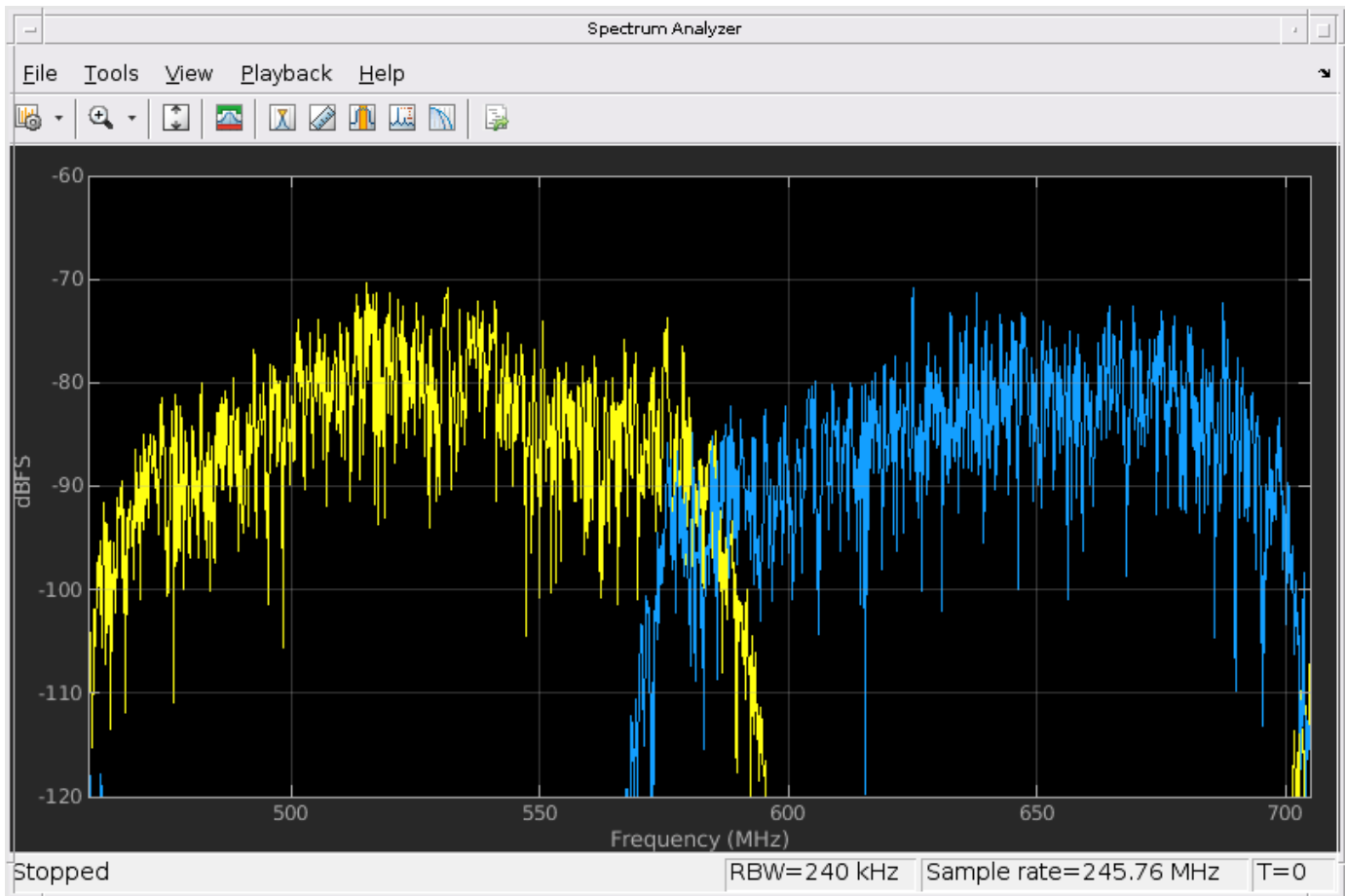
Plot Spectrum of Captured Data

Create a `dsp.SpectrumAnalyzer` object. Set the sample rate of the spectrum analyzer object to the resampled rate of the captured data. Plot the spectrum of the resampled data.

```
PlotPowerLimits = [-120, -60];
```

Capture and plot frequency spectrum

```
spectrumScope = dsp.SpectrumAnalyzer;
spectrumScope.SampleRate = bbrx.SampleRate*length(bbrx.Antennas);
spectrumScope.FrequencyOffset = frequencyBand.MidPoint;
spectrumScope.YLimits = PlotPowerLimits;
spectrumScope.SpectrumUnits = "dBFS";
spectrumScope.FullScaleSource = "Property";
spectrumScope.FullScale = 1;
spectrumScope(resampledData);
release(spectrumScope);
```



See Also

Functions

radioConfigurations

Objects

basebandReceiver

More About

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

Spectrum Monitoring

OFDM WiFi Scanner Using SDR Preamble Detection

This example shows how to retrieve information about WiFi networks using a software-defined radio (SDR) and preamble detection. The example scans over the 2.4 GHz and 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 WiFi 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.

```
radios = radioConfigurations;
```

Specify the name of a saved radio setup configuration to use with this example.

```
radioName = radios(1).Name;
```

Configure Preamble Detector

Create a preamble detector object with the specified radio. To speed up the execution time of this example upon subsequent runs, reuse the workspace object from the first run of the example.

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

Set the receive antenna to the radio frequency (RF) port that you use to capture the waveforms with OFDM packets.

```
rxAntenna =  ;
```

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

```
osf = 2;
pd.SampleRate = 20e6*osf;
pd.CaptureDataType = "double";
pd.Antennas = rxAntenna;
pd.ThresholdMethod = "adaptive";
```

Configure Preamble For Radio

The 802.11 standard requires that all WiFi 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 frequency band and channel with a known OFDM packet.

These are the valid 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 APs 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, 177.

```
band = 5;
channel = 52;
```

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

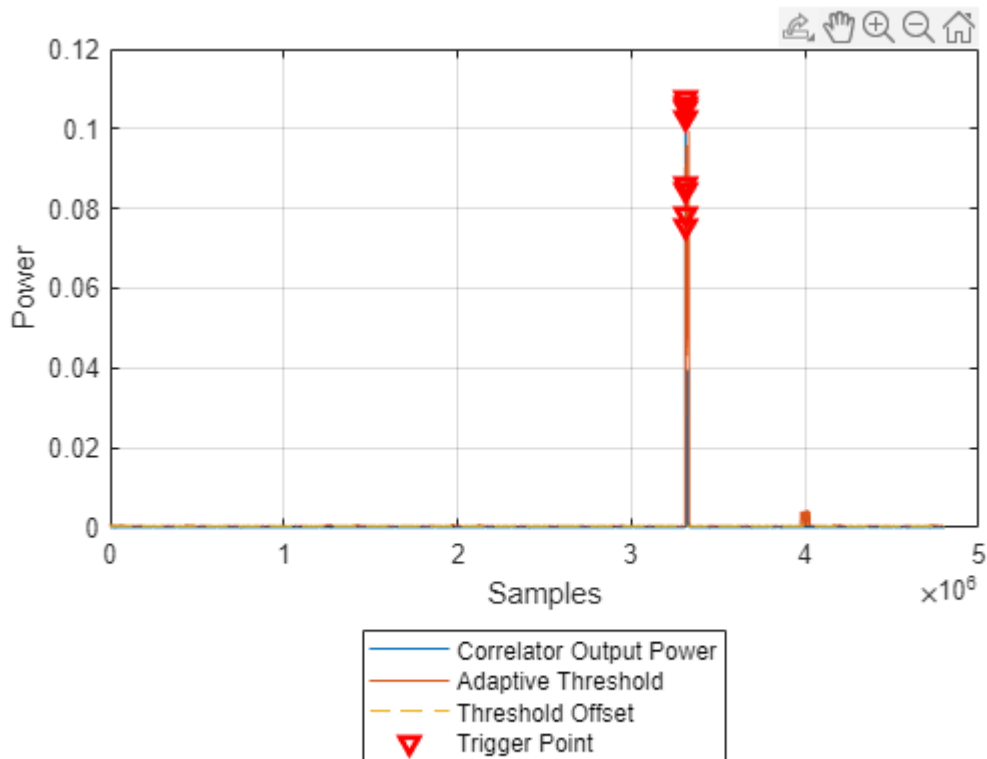
Adjust these values for tuning the preamble detector.

```
pd.AdaptiveThresholdGain = 0.25;
pd.AdaptiveThresholdOffset = 0.0004;
pd.RadioGain = 50;
```

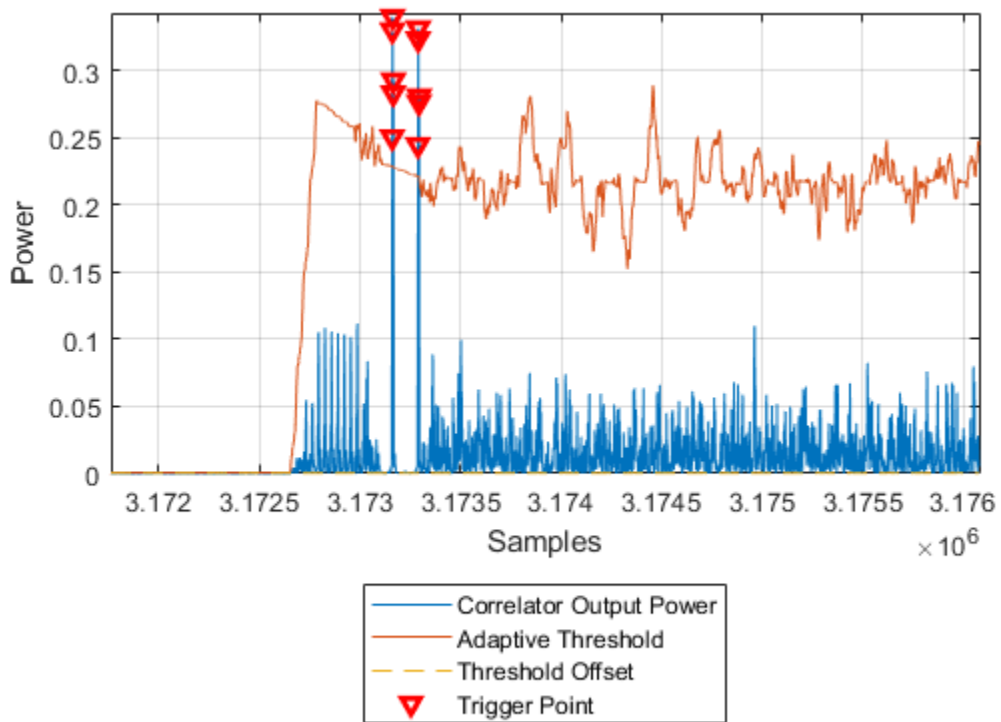
Plot the filter output power, adaptive threshold, and trigger points of the reconfigured preamble detector. The generated figure contains two trigger points for each OFDM packet. Each trigger point corresponds to a long training symbol.

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 WiFi Channels

Specify Scanning Region

Specify the frequency band and the channels for the SDR to scan.

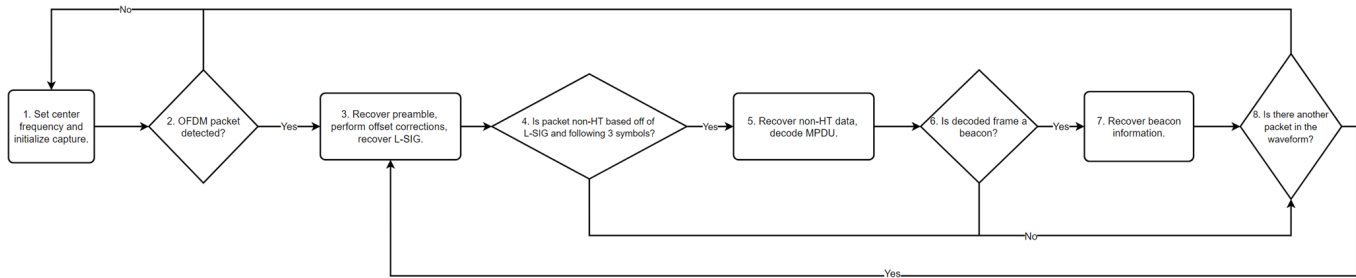
```
band =  ;
channels =  ;
```

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, BSSI, 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(200)`.

```
captureLength = milliseconds(100);
timeout = milliseconds(200);
```

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("Non-HT packet detected on channel %d in band %.1f.\n", channels(i), band)
    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

        % 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.LLTFSNR;
        end
    end
end

```



```

        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
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
end

```

Scanning channel 52 on band 5.0.

Non-HT packet detected on channel 52 in band 5.0.

Beacon 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 157 on band 5.0.

Non-HT packet detected on channel 157 in band 5.0.

Beacon detected on channel 157 in band 5.0.

Beacon detected on channel 157 in band 5.0.

Beacon detected on channel 157 in band 5.0.

Beacon detected on channel 157 in band 5.0.

Convert the APs structure to a table and display the information specified in step 7 on page 3-0 by using the local function `generateBeaconTable`.

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

```
detectedBeaconsInfo=5x9 table
```

SSID	BSSID	Vendor	SNR (dB)
"WLAN_5G"	"04D4C451C584"	"ASUSTek COMPUTER INC."	31.896
"w-inside"	"E82689688B70"	"Aruba, a Hewlett Packard Enterprise Company"	18.527
"w-mobile"	"E82689688B71"	"Aruba, a Hewlett Packard Enterprise Company"	19.53
"w-guest"	"E82689688B72"	"Aruba, a Hewlett Packard Enterprise Company"	18.308
"w-iot"	"E82689688B73"	"Aruba, a Hewlett Packard Enterprise Company"	19.118

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...")
            websave("oui.csv","http://standards-oui.ieee.org/oui/oui.csv");
        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 f
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
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";
    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 operation information elements as defined in IEEE Std 802.11-2020
% Table 11-23.

msbFirst = false;

% Convert to bits to get STA channel width value in 3rd bit.
htOperationInfoBits = int2bit(htElement(2),5*8,msbFirst);

if nargin == 2
    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);

    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
elseif nargin == 1
    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 3-14

- “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

Use the `radioConfigurations` function to identify a radio to use with this example. 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.

```
radios = radioConfigurations;
```

Specify the name of a saved radio setup configuration to use with this example.

```
radioName = radios(1).Name;
```

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 = 5;
channel = 52;
```

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. To speed up the execution time of this example upon subsequent runs, reuse the workspace object from the first run of the example.

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

Set the RF properties of the preamble detector.

```
pd.SampleRate = 40e6;  
pd.CenterFrequency = getWLANCenterFrequency(WLANFrequenciesMap,band,channel);  
pd.Antennas = RF0:RX2;
```

Configure the filter coefficients for preamble detection.

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

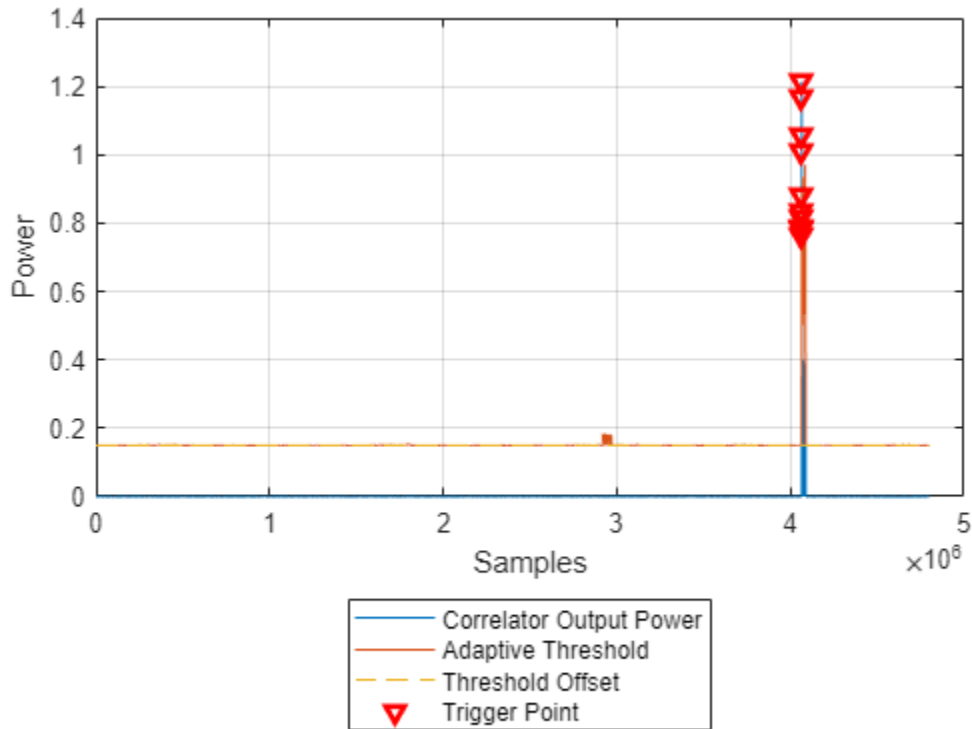
Set the threshold method to adaptive. To include the preamble sequence in the captured waveform, set the trigger offset to a negative value.

```
pd.ThresholdMethod = "adaptive";  
pd.TriggerOffset = -200;
```

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 = 0.2;  
pd.AdaptiveThresholdOffset = 0.15;  
pd.RadioGain = 60;  
plotThreshold(pd,milliseconds(120));
```



Capture WLAN Signal

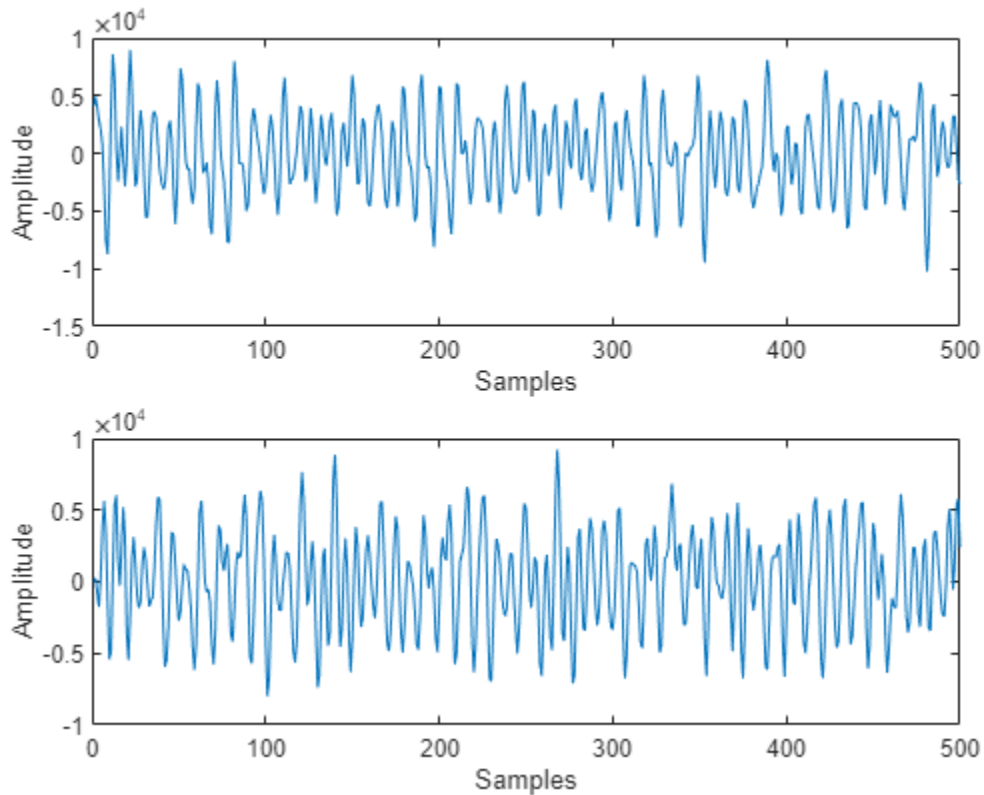
Use the capture function to capture data with the configured preamble detector. Because WLAN beacons are transmitted every 100 ms, capture 100 ms of data with a 200 ms timeout.

```
recordLen = milliseconds(100);
timeout = milliseconds(200);
[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 19-Jan-2022 11:50:19.



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 WiFi Scanner Using SDR Preamble Detection” on page 3-2
- “Supported Radio Devices”

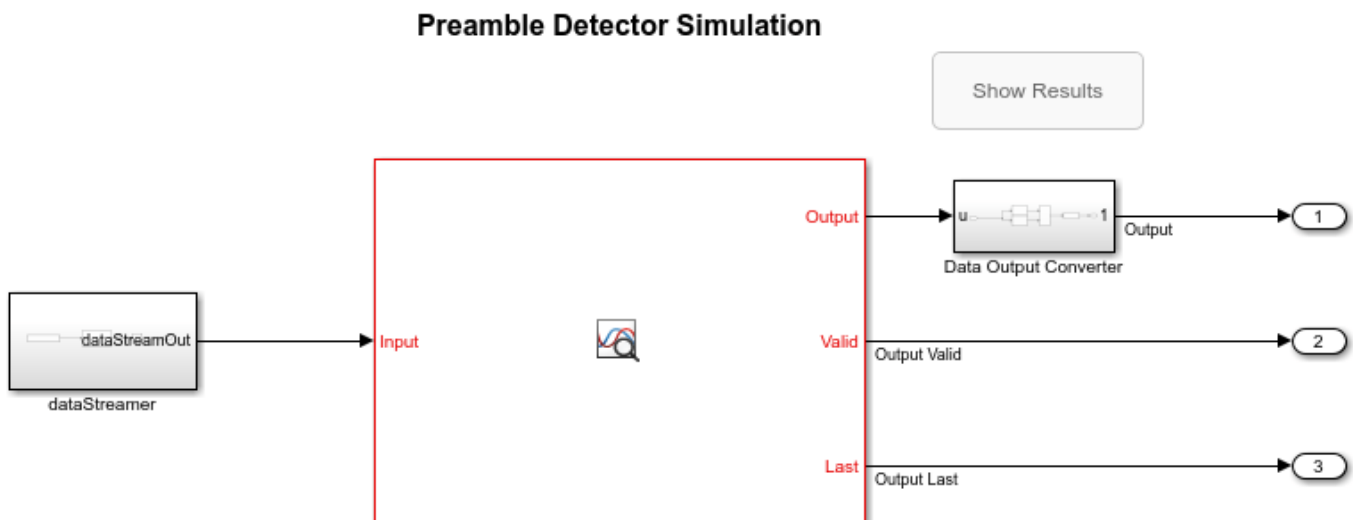
Simulate Triggered Capture Using Preamble Detection

This example shows how to simulate triggered data capture without a connected radio using a Simulink® model. The model simulates a radio that is configured as a preamble detector. The preamble detection parameters of the model are the same as the corresponding properties of the `preambleDetector`. Use this model to explore the behavior of custom preambles and test waveforms before using the `preambleDetector` object to detect and capture live signals.

Introduction

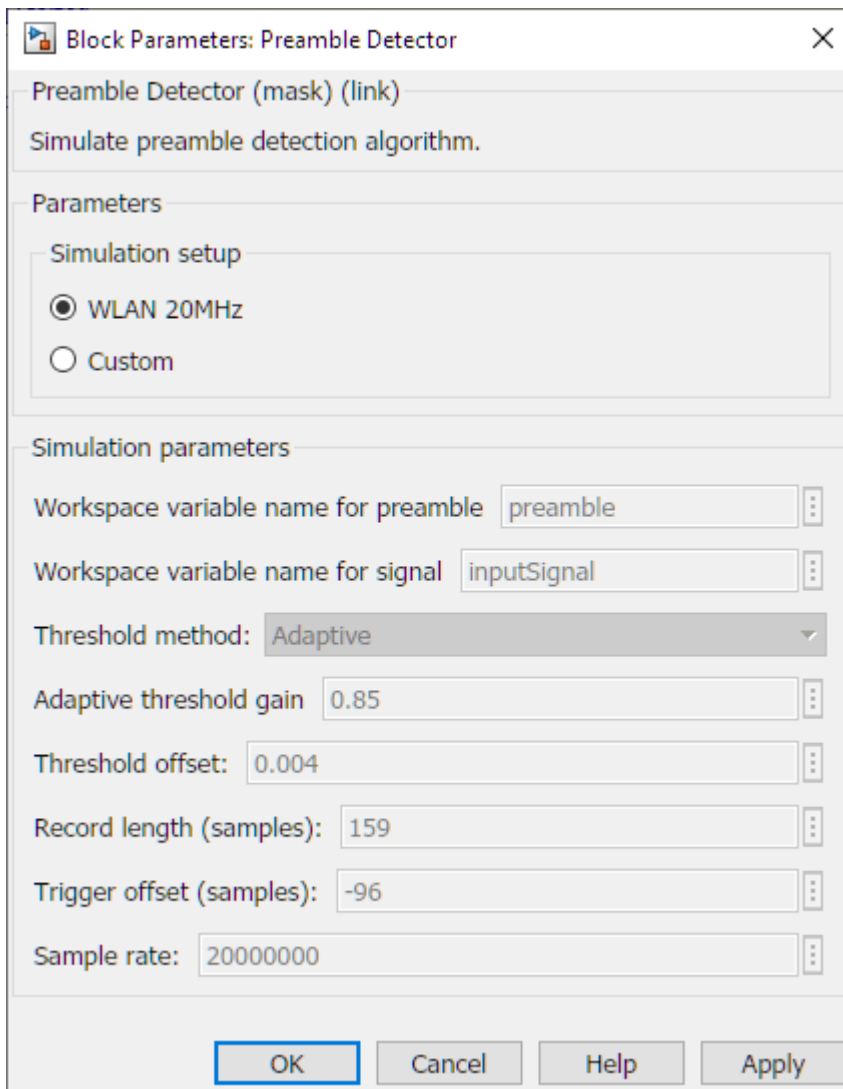
This model simulates a radio that is configured as a preamble detector.

Open the model.

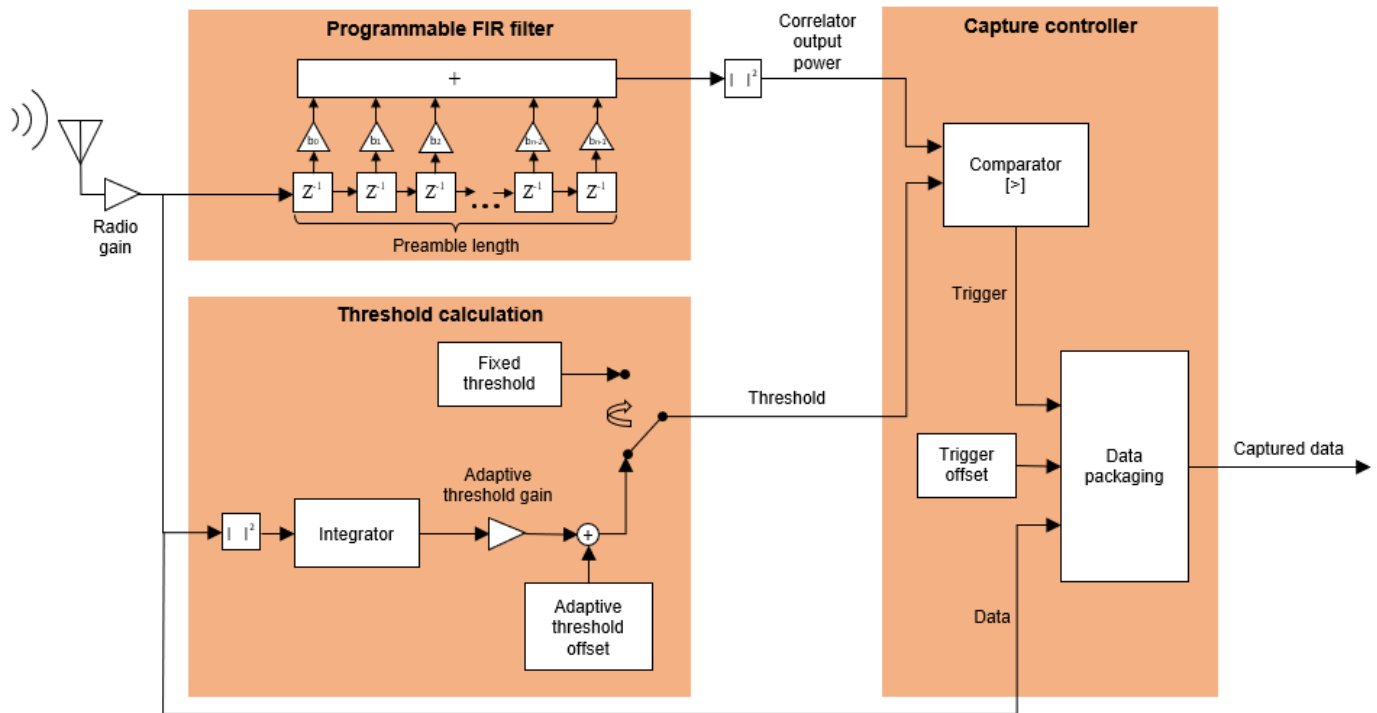


Copyright 2021-2022 The MathWorks, Inc.

The mask parameters of the Preamble Detector subsystem enable you to configure the preamble detector and the input test waveform inside the `dataStreamer` subsystem. The WLAN simulation setup configures the model to simulate the detection of a WLAN test waveform. In the custom simulation setup, you can configure your own test waveform and preamble detector parameters. For more information on how to set the mask parameters, see the corresponding properties of the `preambleDetector` object.



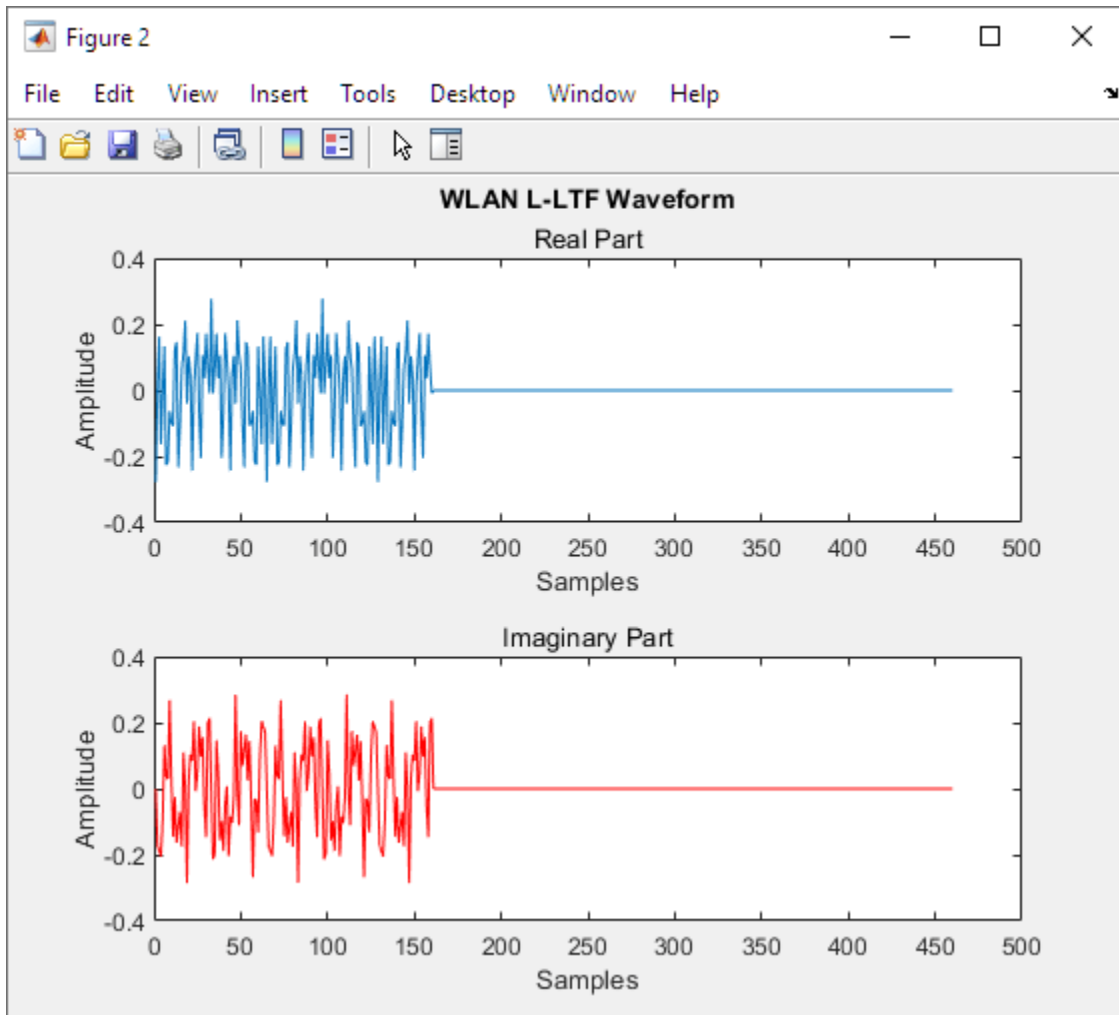
This diagram shows how the Preamble Detector subsystem models preamble detection.



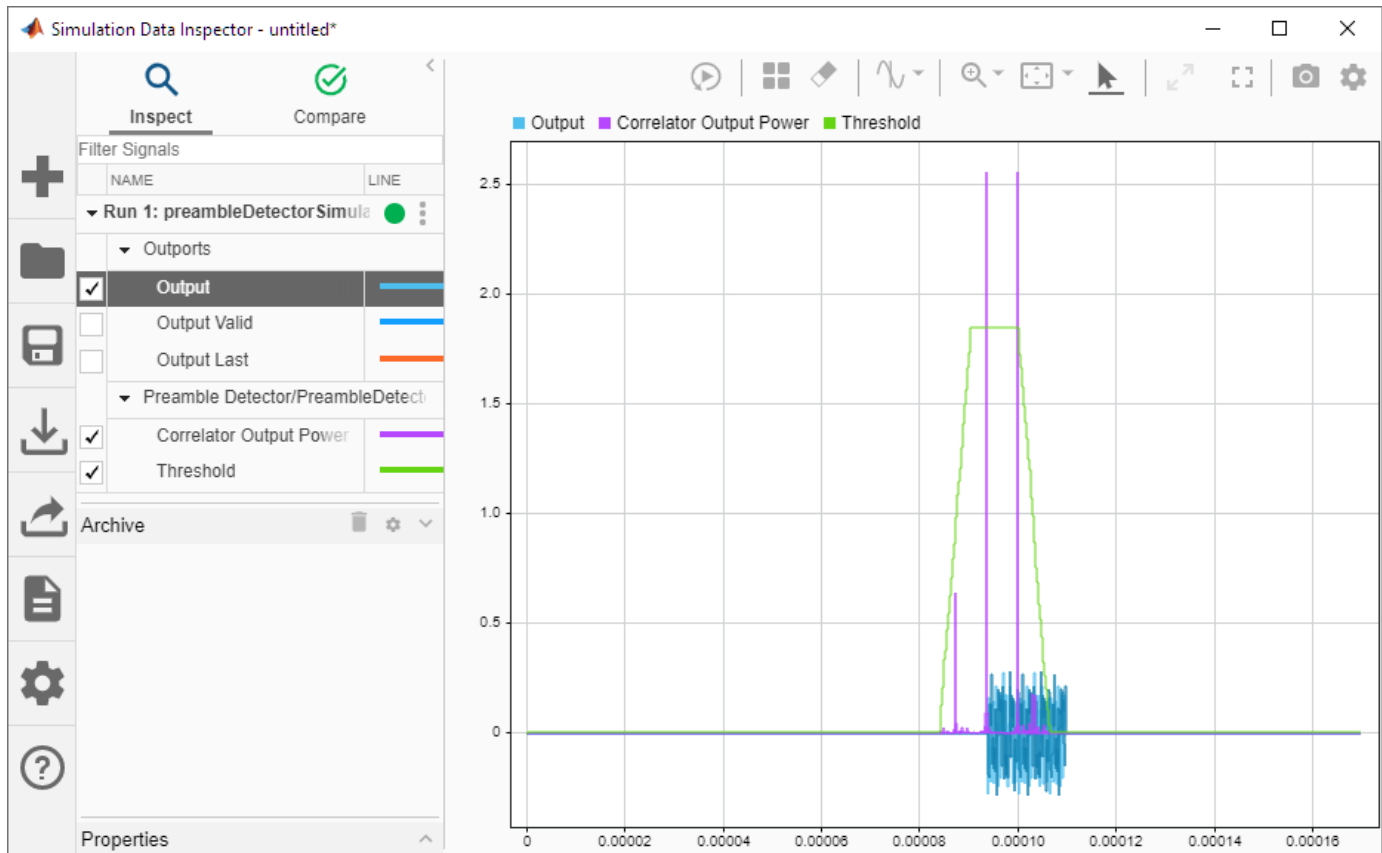
- The Programmable FIR filter correlates the input signal with the preamble sequence.
- The Threshold calculation component provides the threshold for the trigger point generation.
- The Comparator component generates the trigger point, which is the first data sample for which the correlator output power is greater than the threshold.
- The Data packaging component starts the data capture at the trigger offset relative to the trigger point.

Run WLAN Simulation Setup

Under **Simulation setup**, select **WLAN 20MHz**. This selection models the detection and capture of a WLAN waveform that is configured with a 160 sample legacy long training field (L-LTF). The sample rate is 20 MHz. The preamble consists of the first long training symbol from the L-LTF, which is 64 samples.



Simulate the model for 0.17×10^{-3} seconds, then click **Show Results** to open the Simulink Data Inspector. To see the result of the detection, select the **Output**, **Correlator Output Power**, and **Threshold** signals. The trigger point for detection is the first data sample for which the **Correlator Output Power** value is greater than the **Threshold** value. The **Output** signal from the Preamble Detector subsystem contains samples from the test waveform. The Trigger Offset parameter specifies the first data sample in the **Output** signal relative to the trigger point. The **Output Valid** signal defines which samples in the **Output** signal are valid. The **Output Last** signal is an AXI-S signal that is not relevant in the detection analysis of this example.



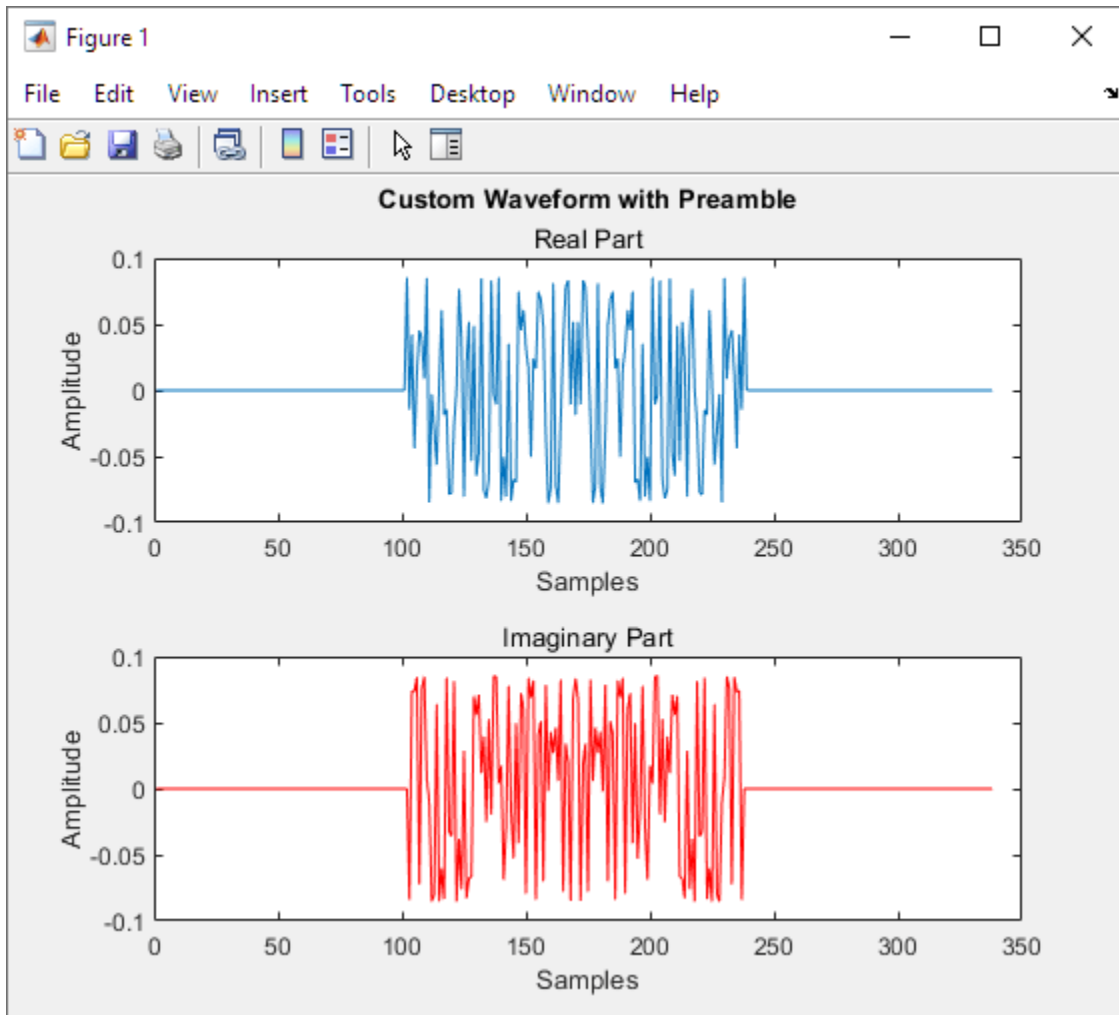
Run Custom Simulation Setup

Under **Simulation Setup**, select **Custom**. This selection enables you to configure the model to simulate the detection and capture of a custom waveform. To configure the model:

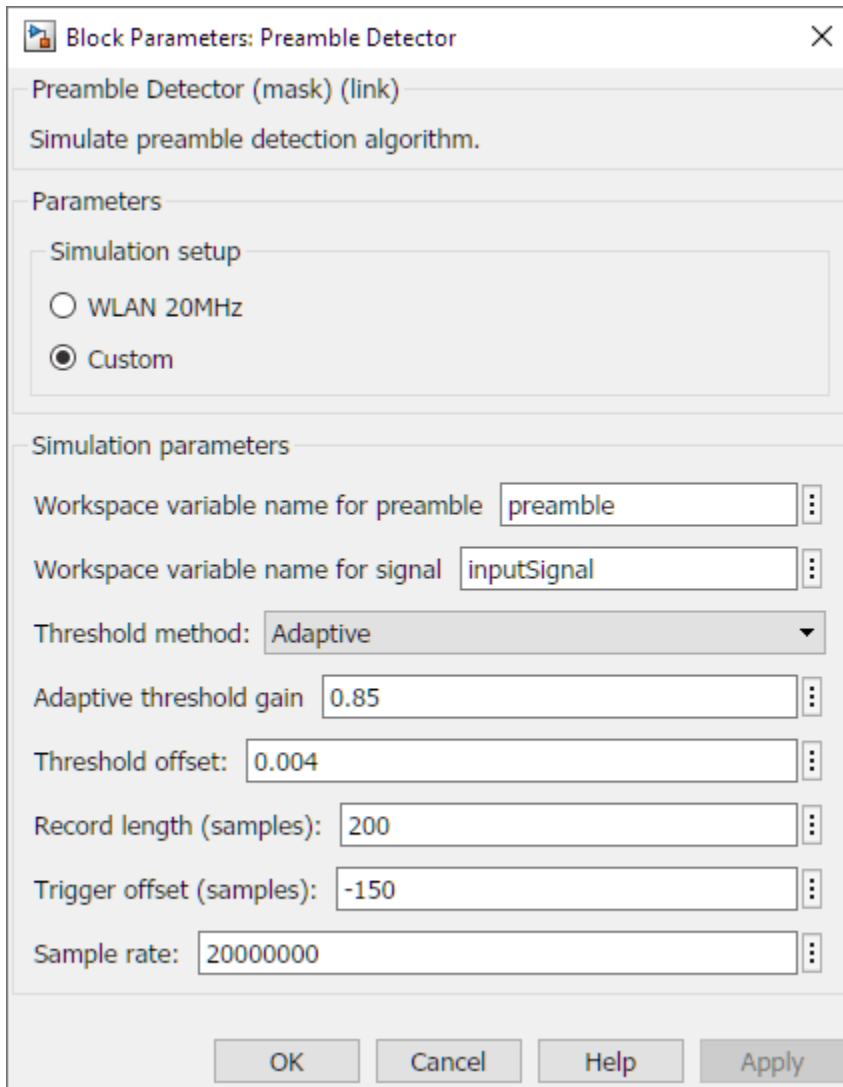
- Create the `preamble` variable in the workspace as a numeric vector with values in the range $[-1, 1)$.
- Create the `inputSignal` variable in the workspace as a numeric vector of type `double` with values in the range $[-1, 1)$.
- Set the Preamble Detector subsystem parameters.
- Set the simulation stop time.

To illustrate this workflow, run the `hGenerateZadoffChuPreambleWaveform` script to generate the `preamble` and `inputSignal` variables.

```
>> hGenerateZadoffChuPreambleWaveform
```



Configure the Preamble Detector subsystem using the values in this figure.



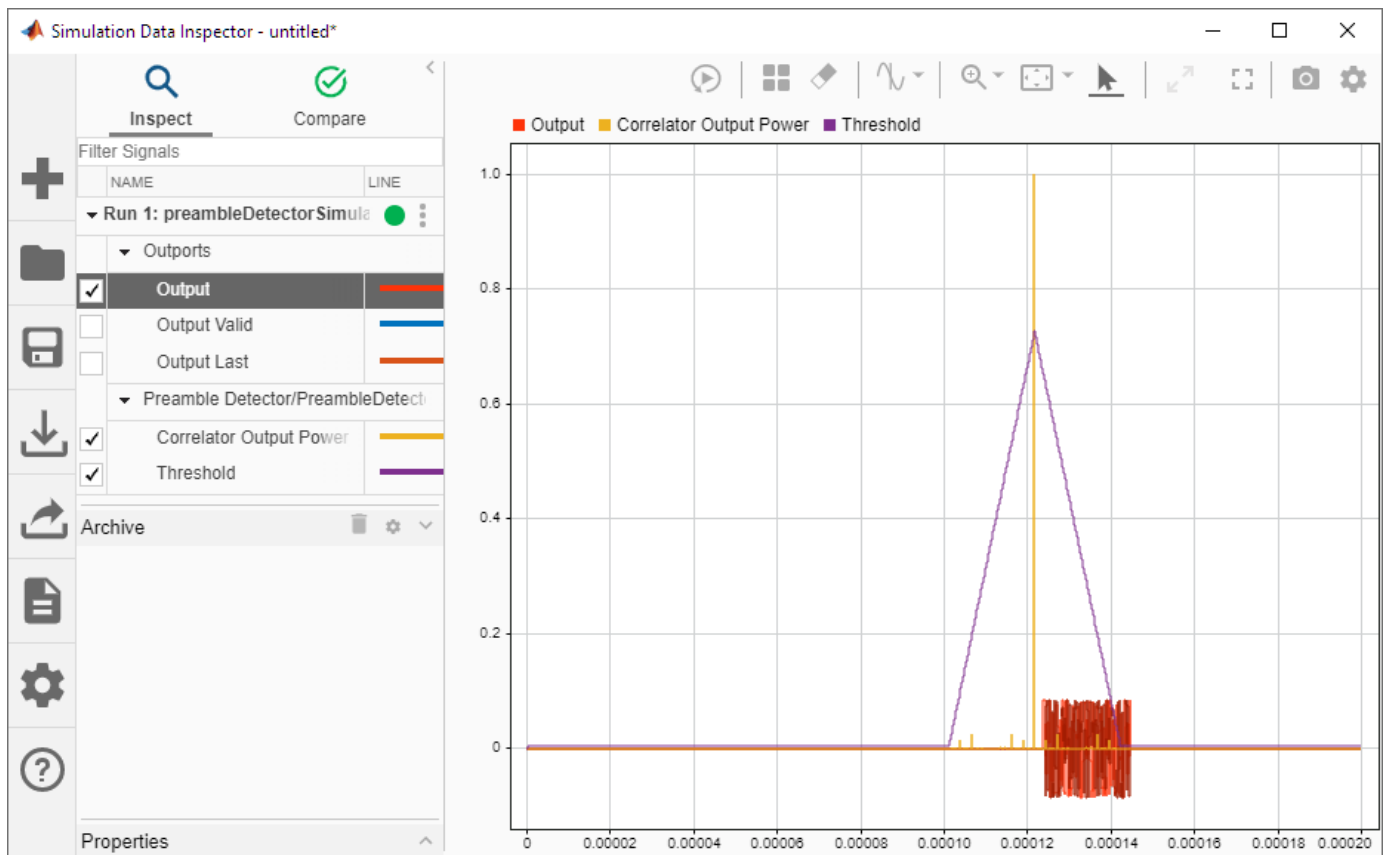
To enable the model to fully process the input waveform, use this formula to calculate the minimum simulation stop time.

$$StopTime = (1580 + oversamplingRatio * (dataLength + preambleLength) + 280) / SampleRate$$

- *preambleLength* is the length of the preamble variable.
- *dataLength* is the length of the inputSignal variable.
- *oversamplingRatio* is $\text{ceil}(preambleLength/48)$.
- *SampleRate* is the sample rate parameter defined on the Preamble Detector subsystem.

For this example, the minimum stop time is $(1580 + 3*(338 + 137) + 280)/20e6 = 1.6425e-04$ seconds. To enable some overhead for viewing the output signals, set the stop time to $2e-4$ seconds.

Run the simulation and view the output.



See Also

Objects

preambleDetector

More About

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

