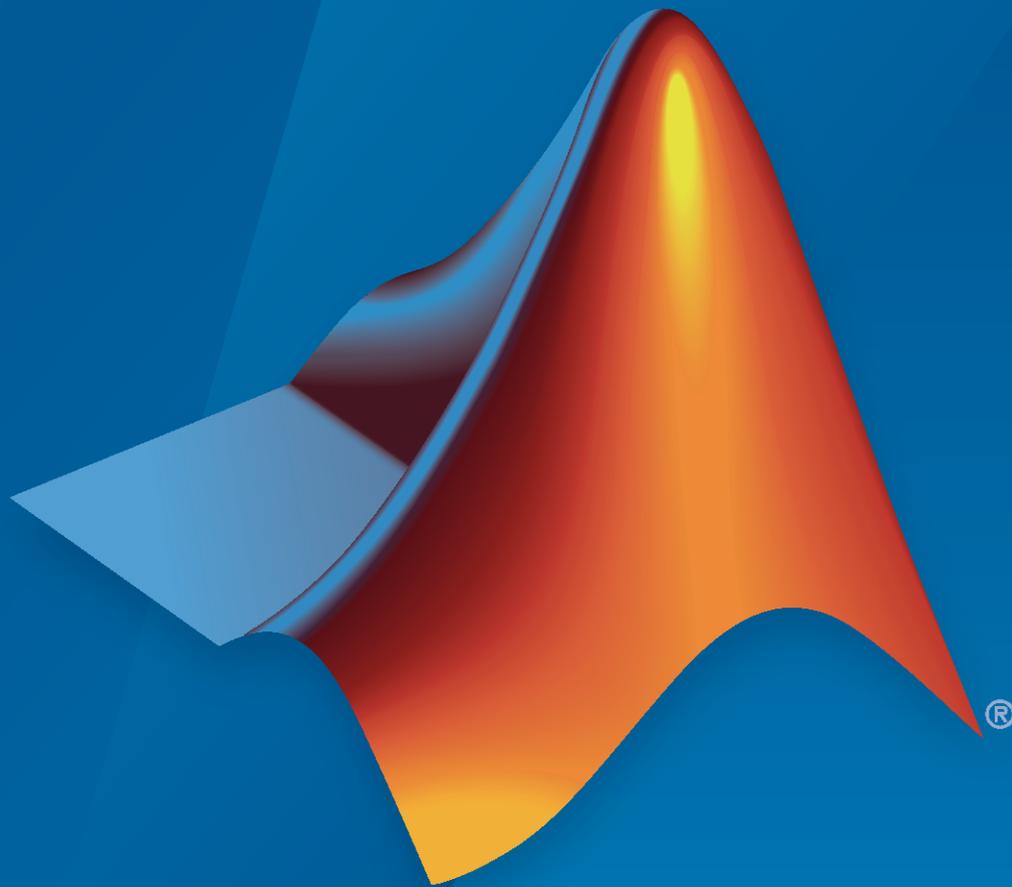


Satellite Communications Toolbox

User's Guide



MATLAB[®]

R2022b



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

Satellite Communications Toolbox User's Guide

© COPYRIGHT 2021–2022 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 2021	Online only	New for Version 1.0 (Release 2021a)
September 2021	Online only	Revised for Version 1.1 (Release 2021b)
March 2022	Online only	Revised for Version 1.2 (Release 2022a)
September 2022	Online only	Revised for Version 1.3 (Release 2022b)

	Satellite Scenario Generation
1	
	Multi-Hop Satellite Communications Link Between Two Ground Stations 1-2
	Satellite Constellation Access to Ground Station 1-14
	Comparison of Orbit Propagators 1-25
	Modeling Satellite Constellations Using Ephemeris Data 1-33
	Estimate GNSS Receiver Position with Simulated Satellite Constellations 1-43
	Calculate Latency and Doppler in a Satellite Scenario 1-49
	Interference from Satellite Constellation on Communications Link ... 1-59
	Multi-Hop Path Selection Through Large Satellite Constellation 1-91
	Modeling Custom Satellite Attitude and Gimbal Steering 1-99
	Signal Transmission
2	
	GPS Waveform Generation 2-2
	CCSDS Optical High Photon Efficiency Telemetry Waveform Generation 2-17
	DVB-S2X Super-Frame Generation for Formats 0 and 1 2-25
	DVB-S2X Super-Frame Generation for Formats 2 and 3 2-33
	RF Propagation and Channel Models
3	
	Simulate and Visualize Land Mobile-Satellite Channel 3-2

Model NR NTN Channel	3-8
Antenna Size Analysis Using ITU-R P.618 Propagation Model	3-16

End-to-End Simulation

4

End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections	4-2
End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections	4-13
End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections	4-23
End-to-End DVB-S2 Simulation with RF Impairments and Corrections	4-37
End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames	4-54
End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames	4-69
End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode	4-87
End-to-End CCSDS SCPPM Simulation Using Deep Space Poisson Channel	4-102
DVB-S2 Bent Pipe Simulation with RF Impairments and Corrections	4-110
GPS Receiver Acquisition and Tracking Using C/A-Code	4-118
GPS Data Decode	4-139
NR NTN PDSCH Throughput	4-147
End-to-End GPS Legacy Navigation Receiver Using C/A-Code	4-169

Over-the-Air Testing

5

Capture Satellite Data Using AWS Ground Station	5-2
VITA 49 File Reader	5-17

What is C Code Generation from MATLAB?	6-2
Using MATLAB Coder	6-2
C/C++ Compiler Setup	6-2
Functions and System Objects That Support Code Generation	6-3
DVB-S2 HDL Transmitter	6-4
DVB-S2 HDL PL Header Recovery	6-16
DVB-S2 HDL Receiver	6-36
GPS HDL Acquisition and Tracking Using C/A Code	6-46
DVB-S.2 System Simulation Using a GPU-Based LDPC Decoder System Object	6-57

Satellite Scenario Generation

Multi-Hop Satellite Communications Link Between Two Ground Stations

This example demonstrates how to set up a multi-hop satellite communications link between two ground stations. The first ground station is located in India (Ground Station 1), and the second ground station is located in Australia (Ground Station 2). The link is routed via two satellites (Satellite 1 and Satellite 2). Each satellite acts as a regenerative repeater. A regenerative repeater receives an incoming signal, and then demodulates, remodulates, amplifies, and retransmits the received signal. The times over the course of a day during which Ground Station 1 can send data to Ground Station 2 are determined.

Create Satellite Scenario

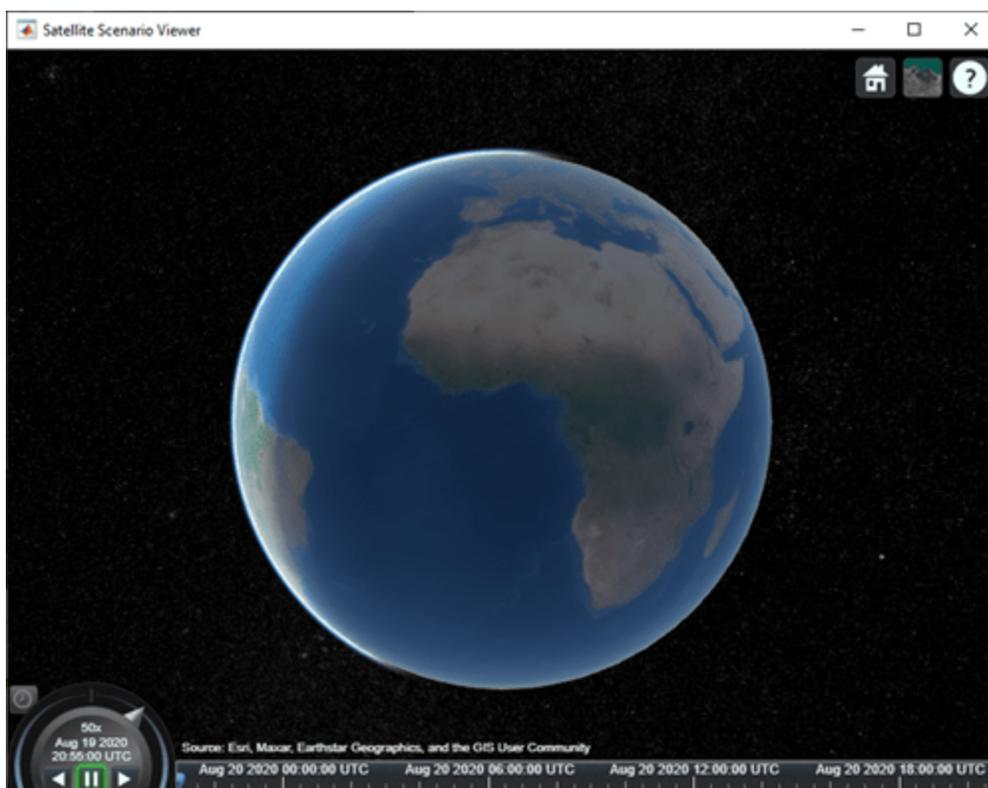
Use `satelliteScenario` to create a satellite scenario. Use `datetime` to define the start time and stop time of the scenario. Set the sample time to 60 seconds.

```
startTime = datetime(2020,8,19,20,55,0);           % 19 August 2020 8:55 PM UTC
stopTime = startTime + days(1);                   % 20 August 2020 8:55 PM UTC
sampleTime = 60;                                  % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Launch Satellite Scenario Viewer

Use `satelliteScenarioViewer` to launch a Satellite Scenario Viewer.

```
satelliteScenarioViewer(sc);
```



Add the Satellites

Use `satellite` to add Satellite 1 and Satellite 2 to the scenario by specifying their Keplerian orbital elements corresponding to the scenario start time.

```

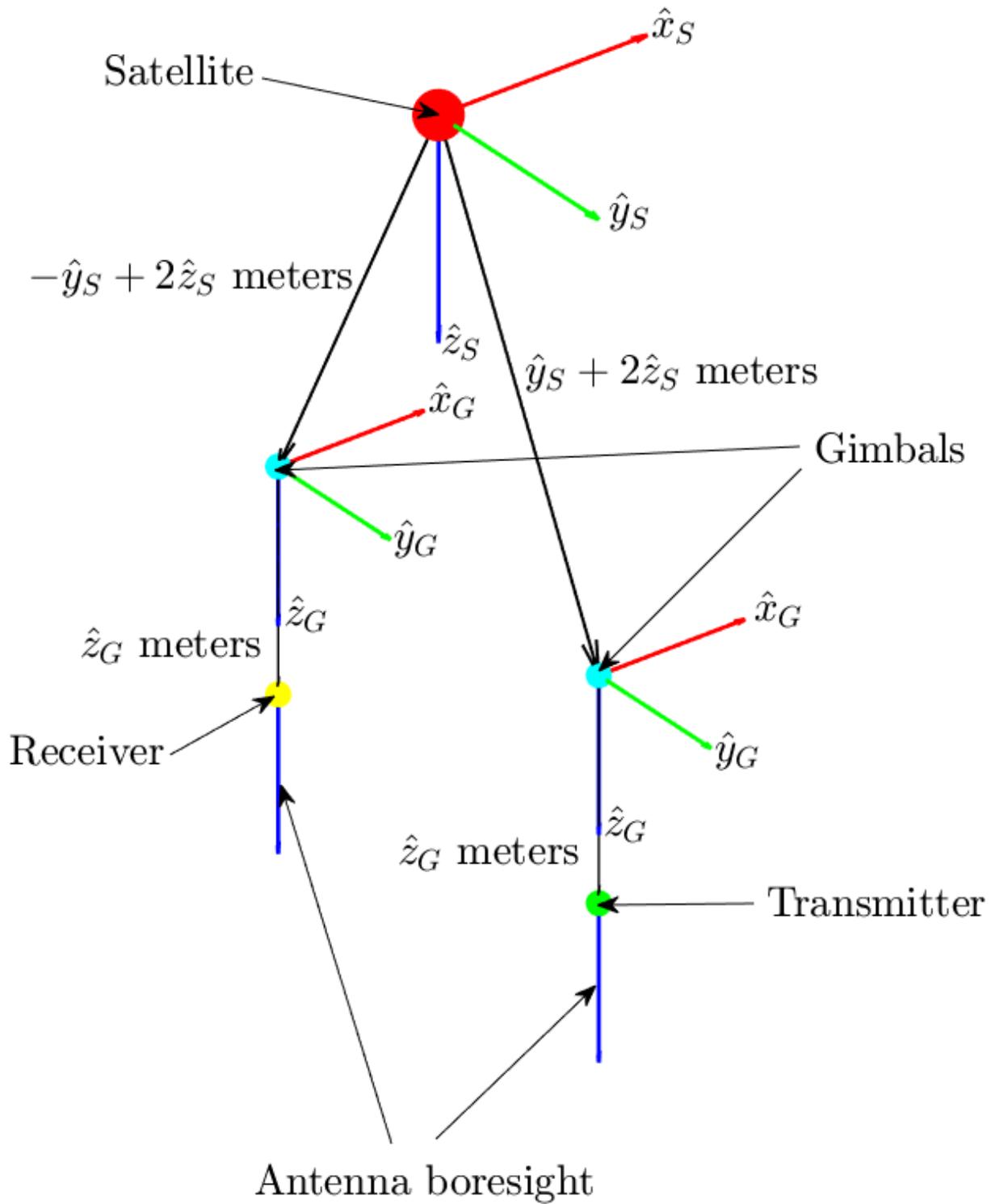
semiMajorAxis = 10000000;           % meters
eccentricity = 0;
inclination = 0;                     % degrees
rightAscensionOfAscendingNode = 0;  % degrees
argumentOfPeriapsis = 0;           % degrees
trueAnomaly = 0;                    % degrees
sat1 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
    "Name", "Satellite 1", ...
    "OrbitPropagator", "two-body-keplerian");

semiMajorAxis = 10000000;           % meters
eccentricity = 0;
inclination = 30;                    % degrees
rightAscensionOfAscendingNode = 120; % degrees
argumentOfPeriapsis = 0;           % degrees
trueAnomaly = 300;                  % degrees
sat2 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
    "Name", "Satellite 2", ...
    "OrbitPropagator", "two-body-keplerian");

```

Add Gimbals to the Satellites

Use `gimbal` to add gimbals to the satellites. Each satellite consists of two gimbals on opposite sides of the satellite. One gimbal holds the receiver antenna and the other gimbal holds the transmitter antenna. The mounting location is specified in cartesian coordinates in the body frame of the satellite, which is defined by $(\hat{x}_S, \hat{y}_S, \hat{z}_S)$, where \hat{x}_S , \hat{y}_S and \hat{z}_S are the roll, pitch and yaw axes respectively, of the satellite. The mounting location of the gimbal that holds the receiver is $-\hat{y}_S + 2\hat{z}_S$ meters and that of the gimbal that holds the transmitter is $\hat{y}_S + 2\hat{z}_S$ meters, as illustrated in the diagram below.



```

gimbalSat1Tx = gimbal(sat1, ...
    "MountingLocation",[0;1;2]); % meters
gimbalSat2Tx = gimbal(sat2, ...
    "MountingLocation",[0;1;2]); % meters
gimbalSat1Rx = gimbal(sat1, ...
    "MountingLocation",[0;-1;2]); % meters
gimbalSat2Rx = gimbal(sat2, ...
    "MountingLocation",[0;-1;2]); % meters

```

Add Receivers and Transmitters to the Gimbals

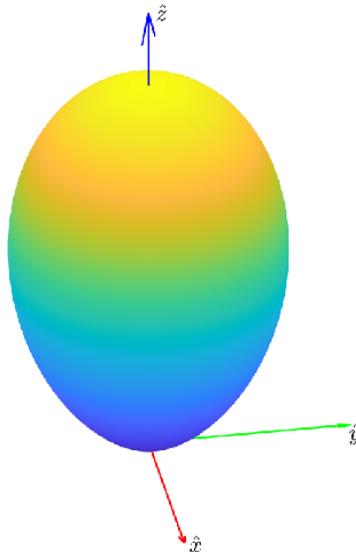
Each satellite consists of a receiver and a transmitter, constituting a regenerative repeater. Use `receiver` to add a receiver to the gimbals `gimbalSat1Rx` and `gimbalSat2Rx`. The mounting location of the receiving antenna with respect to the gimbal is \hat{z}_G meters, as illustrated in the diagram above. The receiver gain to noise temperature ratio is 3dB/K and the required E_b/N_0 is 4 dB.

```

sat1Rx = receiver(gimbalSat1Rx, ...
    "MountingLocation",[0;0;1], ... % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",4); % decibels
sat2Rx = receiver(gimbalSat2Rx, ...
    "MountingLocation",[0;0;1], ... % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",4); % decibels

```

Use `gaussianAntenna` to set the dish diameter of the receiver antennas on the satellites to 0.5 m. A Gaussian antenna has a radiation pattern that peaks at its boresight and decays radial-symmetrically based on a Gaussian distribution while moving away from boresight, as shown in the diagram below. The peak gain is a function of the dish diameter and aperture efficiency.



```

gaussianAntenna(sat1Rx, ...
    "DishDiameter",0.5);    % meters
gaussianAntenna(sat2Rx, ...
    "DishDiameter",0.5);    % meters

```

Use `transmitter` to add a transmitter to the gimbals `gimbalSat1Tx` and `gimbalSat2Tx`. The mounting location of the transmitting antenna with respect to the gimbal is \hat{z}_G meters, where $(\hat{x}_G, \hat{y}_G, \hat{z}_G)$ define the body frame of the gimbal. The boresight of the antenna is aligned with \hat{z}_G . Both satellites transmit with a power of 15 dBW. The transmitter onboard Satellite 1 is used in the crosslink for sending data to Satellite 2 at a frequency of 30 GHz. The transmitter onboard Satellite 2 is used in the downlink to Ground Station 2 at a frequency of 27 GHz.

```

sat1Tx = transmitter(gimbalSat1Tx, ...
    "MountingLocation",[0;0;1], ...    % meters
    "Frequency",30e9, ...              % hertz
    "Power",15);                       % decibel watts
sat2Tx = transmitter(gimbalSat2Tx, ...
    "MountingLocation",[0;0;1], ...    % meters
    "Frequency",27e9, ...              % hertz
    "Power",15);                       % decibel watts

```

Like the receiver, the transmitter also uses a Gaussian antenna. Set the dish diameter of the transmitter antennas of the satellites to 0.5 m.

```

gaussianAntenna(sat1Tx, ...
    "DishDiameter",0.5);    % meters
gaussianAntenna(sat2Tx, ...
    "DishDiameter",0.5);    % meters

```

Add the Ground Stations

Use `groundStation` to add the ground stations at India (Ground Station 1) and Australia (Ground Station 2).

```

latitude = 12.9436963;           % degrees
longitude = 77.6906568;         % degrees
gs1 = groundStation(sc, ...
    latitude, ...
    longitude, ...
    "Name","Ground Station 1");

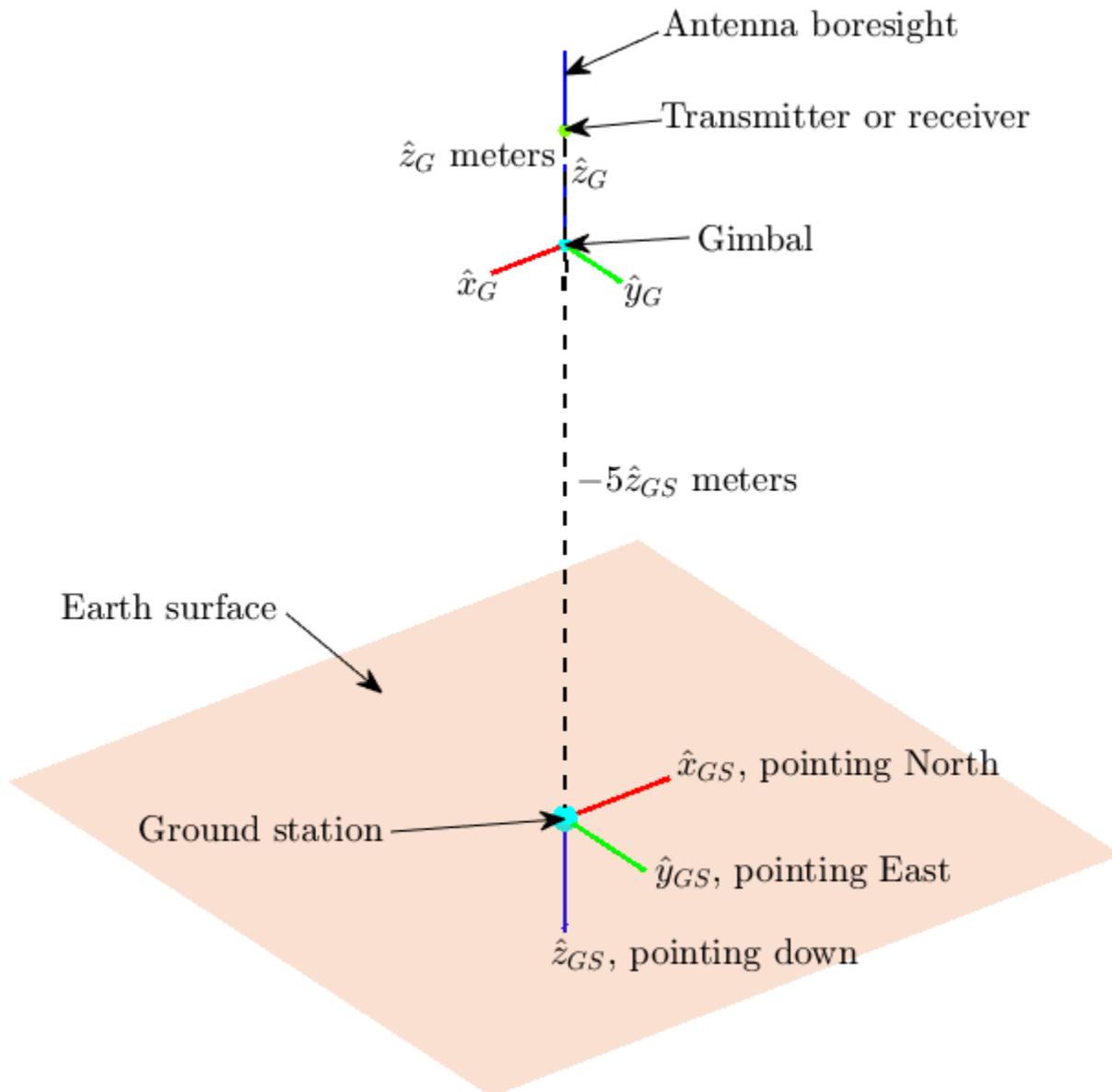
latitude = -33.7974039;         % degrees
longitude = 151.1768208;       % degrees
gs2 = groundStation(sc, ...
    latitude, ...
    longitude, ...
    "Name","Ground Station 2");

```

Add Gimbal to Each Ground Station

Use `gimbal` to add a gimbal to Ground Station 1 and Ground Station 2. The gimbal at Ground Station 1 holds a transmitter, and the gimbal at Ground Station 2 holds a receiver. The gimbals are located 5 meters above their respective ground stations, as illustrated in the diagram below. Consequently, their mounting locations are $-5\hat{z}_{GS}$ meters, where $(\hat{x}_{GS}, \hat{y}_{GS}, \hat{z}_{GS})$ define the body axis of the ground stations. \hat{x}_{GS} , \hat{y}_{GS} and \hat{z}_{GS} always point North, East and down respectively. Therefore, the \hat{z}_{GS} component of the gimbals is -5 meters so that they are placed above the ground station and not

below. Furthermore, by default, the mounting angles of the gimbal are such that their body axes ($\hat{x}_G, \hat{y}_G, \hat{z}_G$) are aligned with the parent (in this case, the ground station) body axes ($\hat{x}_{GS}, \hat{y}_{GS}, \hat{z}_{GS}$). As a result, when the gimbals are not steered, their \hat{z}_G axis points straight down, and so does the antenna attached to it using default mounting angles as well. Therefore, you must set the mounting pitch angle to 180 degrees, so that \hat{z}_G points straight up when the gimbal is not steered.



```
gimbalGs1 = gimbal(gs1, ...
    "MountingAngles", [0;180;0], ... % degrees
    "MountingLocation", [0;0;-5]); % meters
gimbalGs2 = gimbal(gs2, ...
```

```
"MountingAngles",[0;180;0], ... % degrees
"MountingLocation",[0;0;-5]); % meters
```

Add Transmitters and Receivers to Ground Station Gimbals

Use `transmitter` to add a transmitter to the gimbal at Ground Station 1. The uplink transmitter sends data to Satellite 1 at a frequency of 30 GHz and a power of 30 dBW. The transmitter antenna is mounted at \hat{z}_G meters with respect to the gimbal.

```
gs1Tx = transmitter(gimbalGs1, ...
    "Name","Ground Station 1 Transmitter", ...
    "MountingLocation",[0;0;1], ... % meters
    "Frequency",30e9, ... % hertz
    "Power",30); % decibel watts
```

Use `gaussianAntenna` to set the dish diameter of the transmitter antenna to 2 m.

```
gaussianAntenna(gs1Tx, ...
    "DishDiameter",2); % meters
```

Use `receiver` to add a receiver to the gimbal at Ground Station 2 to receive downlink data from Satellite 2. The receiver gain to noise temperature ratio is 3 dB/K and the required Eb/No is 1 dB. The mounting location of the receiver antenna is \hat{z}_G meters with respect to the gimbal.

```
gs2Rx = receiver(gimbalGs2, ...
    "Name","Ground Station 2 Receiver", ...
    "MountingLocation",[0;0;1], ... % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",1); % decibels
```

Use `gaussianAntenna` to set the dish diameter of the receiver antenna to 2 m.

```
gaussianAntenna(gs2Rx, ...
    "DishDiameter",2); % meters
```

Set Tracking Targets for Gimbals

For the best link quality, the antennas must continuously point at their respective targets. The gimbals can be steered independent of their parents (satellite or ground station), and configured to track other satellites and ground stations. Use `pointAt` to set the tracking target for the gimbals so that:

- The transmitter antenna at Ground Station 1 points at Satellite 1
- The receiver antenna aboard Satellite 1 points at Ground Station 1
- The transmitter antenna aboard Satellite 1 points at Satellite 2
- The receiver antenna aboard Satellite 2 points at Satellite 1
- The transmitter antenna aboard Satellite 2 points at Ground Station 2
- The receiver antenna at Ground Station 2 points at Satellite 2

```
pointAt(gimbalGs1,sat1);
pointAt(gimbalSat1Rx,gs1);
pointAt(gimbalSat1Tx,sat2);
pointAt(gimbalSat2Rx,sat1);
pointAt(gimbalSat2Tx,gs2);
pointAt(gimbalGs2,sat2);
```

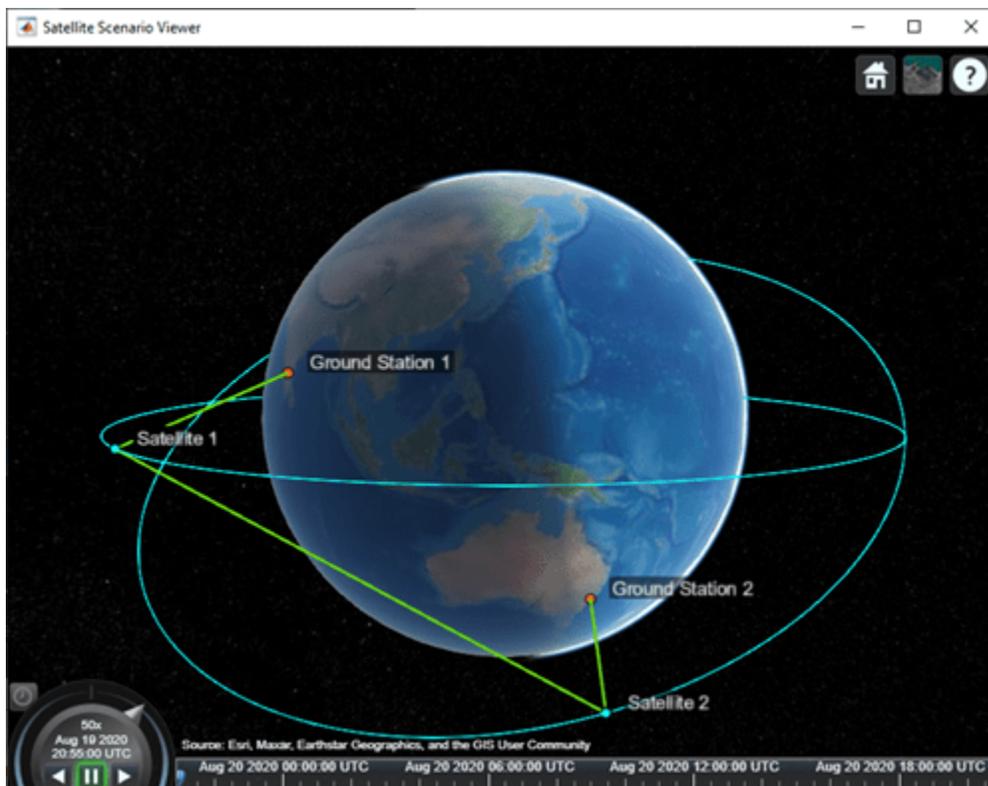
When a target for a gimbal is set, its \hat{z}_G axis will track the target. Since the antenna is on \hat{z}_G and its boresight is aligned with \hat{z}_G , the antenna will also track the desired target.

Add Link Analysis and Visualize Scenario

Use `link` to add link analysis to the transmitter at Ground Station 1. The link is of regenerative repeater-type that originates at `gs1Tx` and ends at `gs2Rx`, and is routed via `sat1Rx`, `sat1Tx`, `sat2Rx` and `sat2Tx`.

```
lnk = link(gs1Tx,sat1Rx,sat1Tx,sat2Rx,sat2Tx,gs2Rx);
```

The Satellite Scenario Viewer automatically updates to display the entire scenario. Use the viewer as a visual confirmation that the scenario has been set up correctly. The green lines represent the link and confirm that the link is closed.



Determine Times When Link is Closed and Visualize Link Closures

Use the `linkIntervals` method to determine the times when the link is closed. The `linkIntervals` method outputs a table of the start and stop times of link closures that represent the intervals during which Ground Station 1 can send data to Ground Station 2. Source and Target are the first and last nodes in the link. If one of Source or Target is on a satellite, `StartOrbit` and `EndOrbit` provide the orbit count of the source or target satellite that they are attached directly or via gimbals, starting from the scenario start time. If both Source and Target are attached to a satellite, `StartOrbit` and `EndOrbit` provide the orbit count of the satellite to which Source is attached. Since both Source and Target are attached to ground stations, `StartOrbit` and `EndOrbit` are NaN.

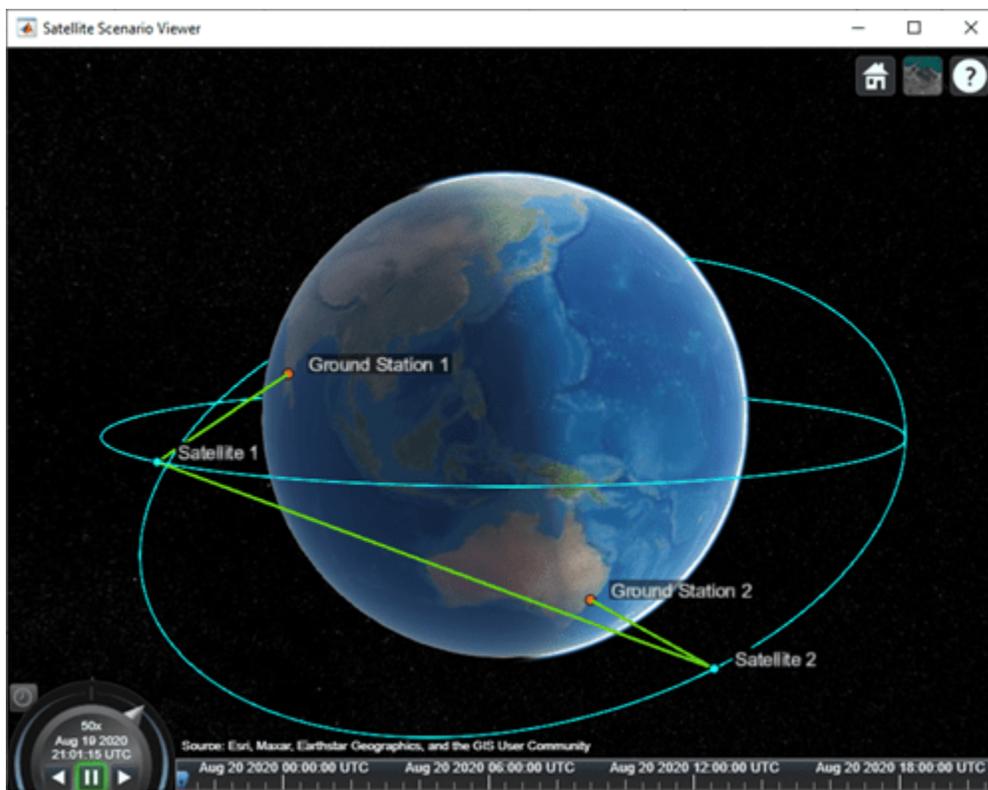
```
linkIntervals(lnk)
```

ans=6x8 table

Source	Target	IntervalNumber	Start
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	1	19-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	2	19-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	3	20-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	4	20-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	5	20-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	6	20-Aug-2020

Use `play` to visualize the scenario simulation from its start time to stop time. The green lines disappear whenever the link cannot be closed.

`play(sc);`

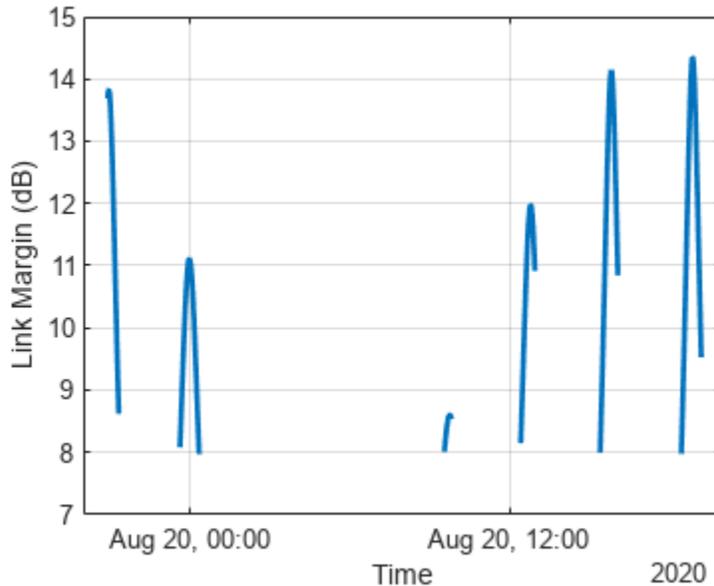


Plot Link Margin at Ground Station 2

The link margin at a receiver is the difference between the energy per bit to noise power spectral density ratio (E_b/N_0) at the receiver and its `RequiredEbNo`. For successful link closure, the link margin must be positive at all receiver nodes. Higher the link margin, better the link quality. To calculate the link margin at final node, that is, Ground Station 2 Receiver, use `ebno` to get the E_b/N_0 history at the Ground Station 2 Receiver, and subtract its `RequiredEbNo` from this quantity to get the link margin. Also, use `plot` to plot the calculate link margin.

```
[e, time] = ebno(lnk);
margin = e - gs2Rx.RequiredEbNo;
plot(time,margin,"LineWidth",2);
```

```
xlabel("Time");
ylabel("Link Margin (dB)");
grid on;
```



The gaps in the plot imply that the link was broken before reaching the final node in the link, or the line of sight between final node and the node before it, that is, Satellite 2, was broken. At all other times, the link margin is positive. This implies that Satellite 2 Transmitter power and Ground Station 2 Receiver sensitivity are always sufficient. It also implies that the margin is positive at all other hops of the link.

Modify Required Eb/No and Observe Effect on Link Intervals

Increase the `RequiredEbNo` of the receiver at Ground Station 2 from 1 dB to 10 dB and recompute the link intervals. Increasing `RequiredEbNo` essentially reduces the sensitivity of Ground Station 2 Receiver. This negatively impacts the resultant link closure times. The number of closed link intervals drops from six to five, and the duration of the closed link intervals is shorter.

```
gs2Rx.RequiredEbNo = 10; % decibels
linkIntervals(lnk)
```

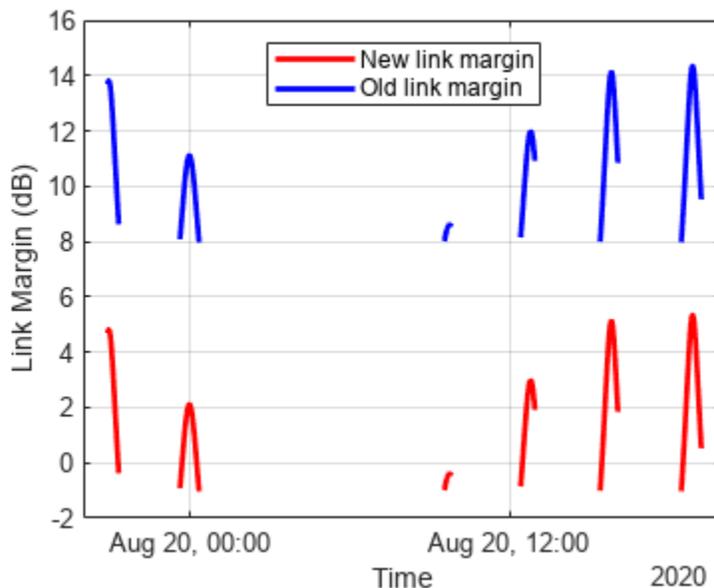
ans=5×8 table

Source	Target	IntervalNumber	Start
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	1	19-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	2	19-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	3	20-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	4	20-Aug-2020
"Ground Station 1 Transmitter"	"Ground Station 2 Receiver"	5	20-Aug-2020

Additionally, the increase in `RequiredEbNo` negatively impacts the link margin. To observe this, recompute and plot the new link margin, and compare it with the previous plot. The link margin has reduced in general, implying that the link quality has gone down as a result of reducing the

sensitivity of the receiver by increasing `RequiredEbNo`. At certain instances, the link margin is negative, signifying that there are times when the link does get broken at Ground Station 2 Receiver, even if it has line of sight to Satellite 2. This implies that the link closure is sometimes limited by the link margin, as opposed to just the line of sight between adjacent nodes.

```
[e, newTime] = ebno(lnk);
newMargin = e - gs2Rx.RequiredEbNo;
plot(newTime,newMargin,"r",time,margin,"b","LineWidth",2);
xlabel("Time");
ylabel("Link Margin (dB)");
legend("New link margin","Old link margin","Location","north");
grid on;
```



Next Steps

This example demonstrated how to set up a multi-hop regenerative repeater-type link and how to determine the times when the link is closed. The link closure times are influenced by the link margin at each receiver in the link. The link margin is the difference between energy per bit to noise power spectral density ratio (E_b/N_o) at the receiver and the required E_b/N_o . The E_b/N_o at a receiver is a function of:

- Orbit and pointing mode of satellites holding the transmitters and receivers
- Position of ground stations holding the transmitters and receivers
- Position, orientation, and pointing mode of the gimbals holding the transmitters and receivers
- Position and orientation of the transmitters and receivers with respect to their parents
- Specifications of the transmitters - power, frequency, bit rate, and system loss
- Specifications of the receivers - gain to noise temperature ratio, required E_b/N_o , and system loss
- Specifications of the transmitter and receiver antennas, such as dish diameter and aperture efficiency for a Gaussian antenna

Modify the above parameters and observe their impact on the link to perform different types of what-if analyses.

See Also

Objects

satelliteScenario | satellite | access | groundStation | satelliteScenarioViewer | conicalSensor | transmitter | receiver

Functions

show | play | hide

Related Examples

- “Satellite Constellation Access to Ground Station” on page 1-14
- “Comparison of Orbit Propagators” on page 1-25
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Satellite Constellation Access to Ground Station

This example demonstrates how to set up access analysis between a ground station and conical sensors onboard a constellation of satellites. A ground station and a conical sensor belonging to a satellite are said to have access to one another if the ground station is inside the conical sensor's field of view and the conical sensor's elevation angle with respect to the ground station is greater than or equal to the latter's minimum elevation angle. The scenario involves a constellation of 40 low-Earth orbit satellites and a geographical site. Each satellite has a camera with a field of view of 90 degrees. The entire constellation of satellites is tasked with photographing the geographical site, which is located at 42.3001 degrees North and 71.3504 degrees West. The photographs are required to be taken between 12 May 2020 1:00 PM UTC and 12 May 2020 7:00 PM UTC when the site is adequately illuminated by the sun. In order to capture good quality pictures with minimal atmospheric distortion, the satellite's elevation angle with respect to the site should be at least 30 degrees (please note that 30 degrees was arbitrarily chosen for illustrative purposes). During the 6 hour interval, it is required to determine the times during which each satellite can photograph the site. It is also required to determine the percentage of time during this interval when at least one satellite's camera can see the site. This percentage quantity is termed the system-wide access percentage.

Create a Satellite Scenario

Create a satellite scenario using `satelliteScenario`. Use `datetime` to set the start time to 12-May-2020 1:00:00 PM UTC, and the stop time to 12-May-2020 7:00:00 PM UTC. Set the simulation sample time to 30 seconds.

```
startTime = datetime(2020,5,12,13,0,0);
stopTime = startTime + hours(6);
sampleTime = 30; % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc =
  satelliteScenario with properties:
    StartTime: 12-May-2020 13:00:00
    StopTime: 12-May-2020 19:00:00
    SampleTime: 30
    AutoSimulate: 1
    Satellites: [1x0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1x0 matlabshared.satellitescenario.GroundStation]
    Viewers: [0x0 matlabshared.satellitescenario.Viewer]
    AutoShow: 1
```

Add Satellites to the Satellite Scenario

Use `satellite` to add satellites to the scenario from the TLE file `leoSatelliteConstellation.tle`. The TLE file defines the mean orbital parameters of 40 generic satellites in nearly circular low-Earth orbits at an altitude and inclination of approximately 500 km and 55 degrees respectively.

```
tleFile = "leoSatelliteConstellation.tle";
sat = satellite(sc,tleFile)
```

```
sat =
  1x40 Satellite array with properties:
```

```

Name
ID
ConicalSensors
Gimbals
Transmitters
Receivers
Accesses
GroundTrack
Orbit
OrbitPropagator
MarkerColor
MarkerSize
ShowLabel
LabelFontColor
LabelFontSize

```

Add Cameras to the Satellites

Use `conicalSensor` to add a conical sensor to each satellite. These conical sensors represent the cameras. Specify their `MaxViewAngle` to be 90 degrees, which defines the field of view.

```

names = sat.Name + " Camera";
cam = conicalSensor(sat, "Name", names, "MaxViewAngle", 90)

```

```

cam =
  1x40 ConicalSensor array with properties:

```

```

Name
ID
MountingLocation
MountingAngles
MaxViewAngle
Accesses
FieldOfView

```

Define the Geographical Site to be Photographed in the Satellite Scenario

Use `groundStation` to add a ground station, which represents the geographical site to be photographed. Specify its `MinElevationAngle` to be 30 degrees. If latitude and longitude are not specified, they default to 42.3001 degrees North and 71.3504 degrees West.

```

name = "Geographical Site";
minElevationAngle = 30; % degrees
geoSite = groundStation(sc, ...
  "Name", name, ...
  "MinElevationAngle", minElevationAngle)

```

```

geoSite =
  GroundStation with properties:

```

```

          Name: Geographical Site
          ID: 81
    Latitude: 42.3 degrees
  Longitude: -71.35 degrees
    Altitude: 0 meters

```

```
MinElevationAngle: 30 degrees
ConicalSensors: [1x0 matlabshared.satellitescenario.ConicalSensor]
    Gimbals: [1x0 matlabshared.satellitescenario.Gimbal]
    Transmitters: [1x0 satcom.satellitescenario.Transmitter]
    Receivers: [1x0 satcom.satellitescenario.Receiver]
    Accesses: [1x0 matlabshared.satellitescenario.Access]
    MarkerColor: [1 0.4118 0.1608]
    MarkerSize: 6
    ShowLabel: true
LabelFontColor: [1 1 1]
LabelFontSize: 15
```

Add Access Analysis Between the Cameras and the Geographical Site

Use `access` to add access analysis between each camera and the geographical site. The access analyses will be used to determine when each camera can photograph the site.

```
ac = access(cam,geoSite);
```

```
% Properties of access analysis objects
ac(1)
```

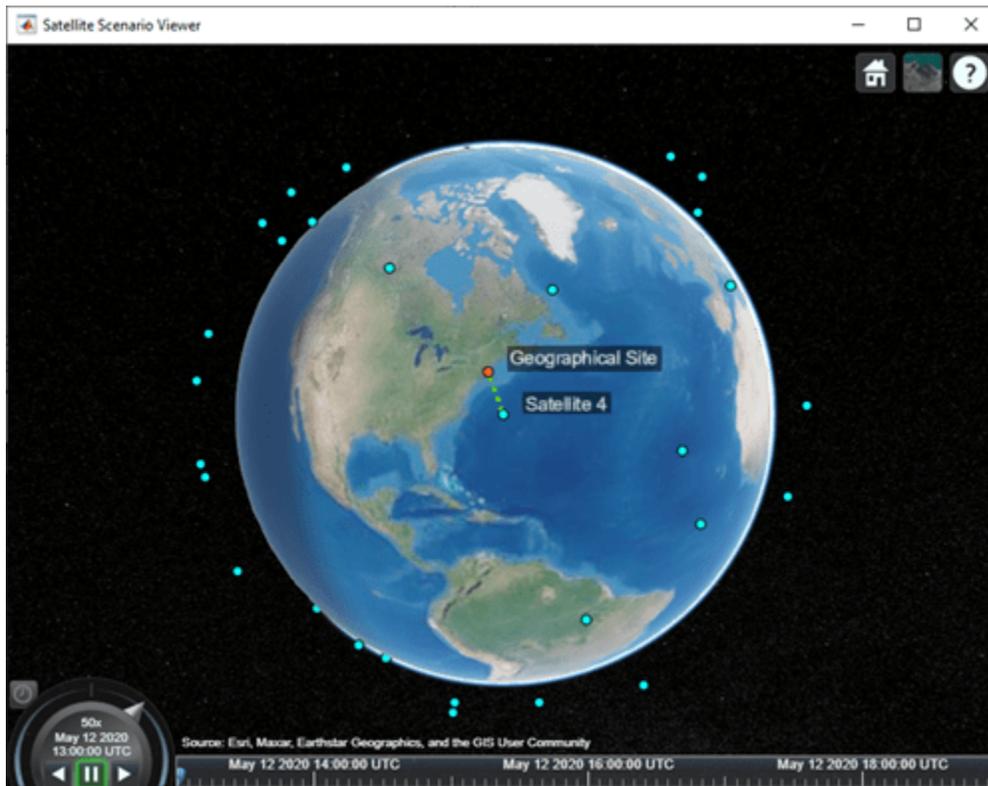
```
ans =
    Access with properties:
        Sequence: [41 81]
        LineWidth: 3
        LineColor: [0.3922 0.8314 0.0745]
```

Visualize the Scenario

Use `satelliteScenarioViewer` to launch a satellite scenario viewer and visualize the scenario. Hide the orbits and labels of satellites and ground stations by setting the `ShowDetails` name-value pair to `false`. Show labels for the geographical site and Satellite 4, and center the satellite in view.

```
v = satelliteScenarioViewer(sc,"ShowDetails",false);
sat(4).ShowLabel = true;
geoSite.ShowLabel = true;
show(sat(4));
```

When the `ShowDetails` property is set to `false`, only satellites and ground stations will be shown. Labels, orbits, fields of view, and ground tracks will be hidden. Mouse over satellites and ground stations to show their labels. Click on a satellite or ground station to reveal its label, orbit, and any other hidden graphics. Click on the satellite or ground station again to dismiss them.



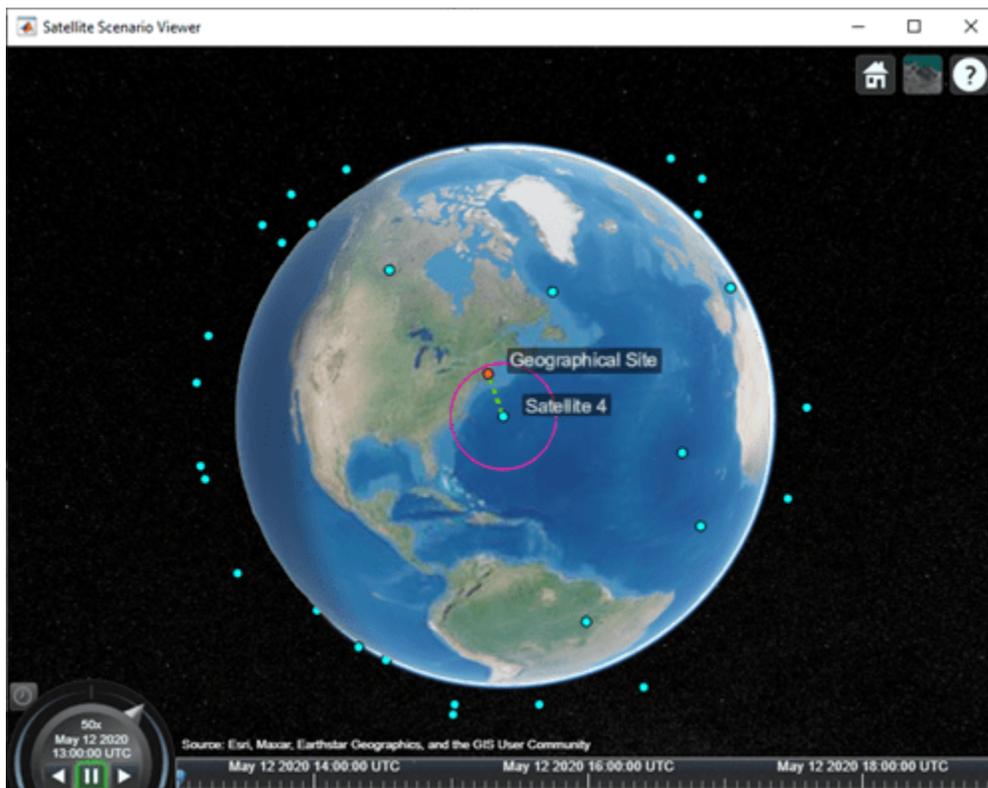
The viewer may be used as a visual confirmation that the scenario has been set up correctly. The violet line indicates that the camera on Satellite 4 and the geographical site have access to one another. This means that the geographical site is inside the camera's field of view and the camera's elevation angle with respect to the site is greater than or equal to 30 degrees. For the purposes of this scenario, this means that the camera can successfully photograph the site.

Visualize the Field Of View of the Camera

Use `fieldOfView` to visualize the field of view of each camera on Satellite 4.

```
fov = fieldOfView(cam([cam.Name] == "Satellite 4 Camera"))
```

```
fov =  
  FieldOfView with properties:  
    LineWidth: 1  
    LineColor: [1 0.0745 0.6510]  
    VisibilityMode: 'inherit'
```

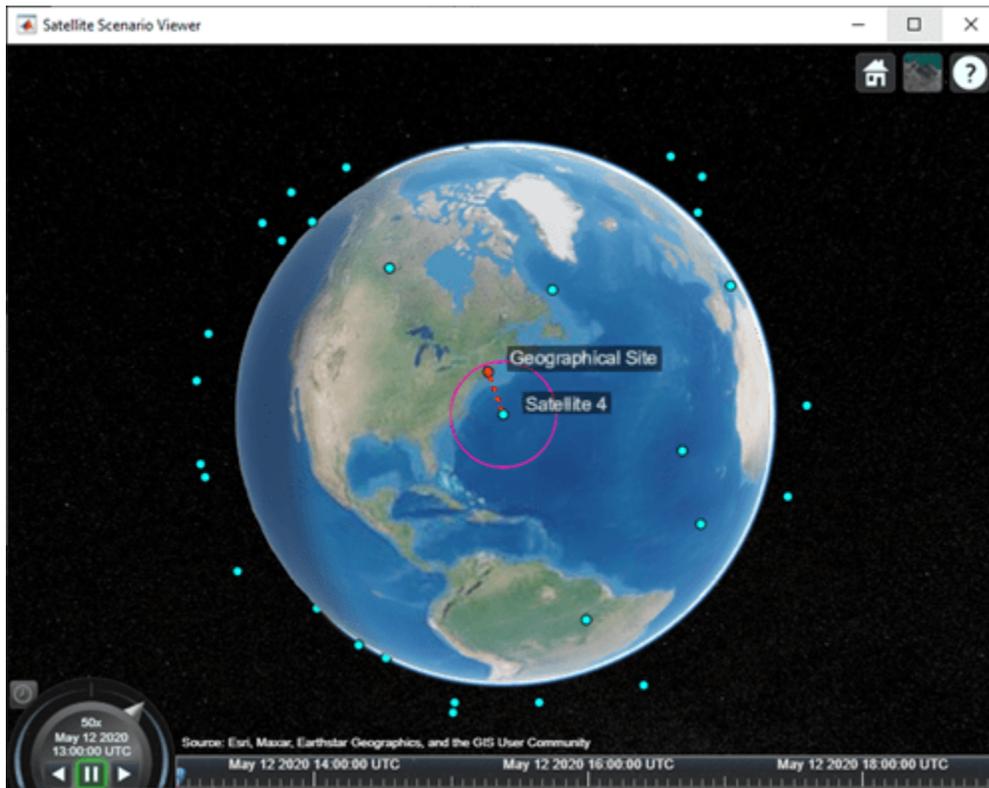


The presence of the geographical site inside the contour is a visual confirmation that it is inside the field of view of the camera onboard Satellite 4.

Customize the Visualizations

Change the color of access visualizations to red.

```
ac.LineColor = 'red';
```



Determine the Times when the Cameras can Photograph the Geographical Site

Use `accessIntervals` to determine the times when there is access between each camera and the geographical site. These are the times when the camera can photograph the site.

`accessIntervals(ac)`

`ans=30x8 table`

Source	Target	IntervalNumber	StartTime
"Satellite 1 Camera"	"Geographical Site"	1	12-May-2020 13:36:00
"Satellite 1 Camera"	"Geographical Site"	2	12-May-2020 15:23:00
"Satellite 2 Camera"	"Geographical Site"	1	12-May-2020 14:30:30
"Satellite 3 Camera"	"Geographical Site"	1	12-May-2020 13:28:30
"Satellite 4 Camera"	"Geographical Site"	1	12-May-2020 13:00:00
"Satellite 4 Camera"	"Geographical Site"	2	12-May-2020 14:46:00
"Satellite 5 Camera"	"Geographical Site"	1	12-May-2020 16:28:30
"Satellite 6 Camera"	"Geographical Site"	1	12-May-2020 17:05:30
"Satellite 7 Camera"	"Geographical Site"	1	12-May-2020 16:20:00
"Satellite 8 Camera"	"Geographical Site"	1	12-May-2020 15:18:00
"Satellite 8 Camera"	"Geographical Site"	2	12-May-2020 17:03:30
"Satellite 9 Camera"	"Geographical Site"	1	12-May-2020 17:55:30
"Satellite 10 Camera"	"Geographical Site"	1	12-May-2020 18:44:30
"Satellite 11 Camera"	"Geographical Site"	1	12-May-2020 18:39:30
"Satellite 12 Camera"	"Geographical Site"	1	12-May-2020 17:58:00
"Satellite 29 Camera"	"Geographical Site"	1	12-May-2020 13:09:30
:			

The above table consists of the start and end times of each interval during which a given camera can photograph the site. The duration of each interval is reported in seconds. StartOrbit and EndOrbit are the orbit counts of the satellite that the camera is attached to when the access begins and ends. The count starts from the scenario start time.

Use `play` to visualize the simulation of the scenario from its start time to stop time. It can be seen that the green lines appear whenever the camera can photograph the geographical site.

```
play(sc);
```

Calculate System-Wide Access Percentage

In addition to determining the times when each camera can photograph the geographical site, it is also required to determine the system-wide access percentage, which is the percentage of time from the scenario start time to stop time when at least one satellite can photograph the site. This is computed as follows:

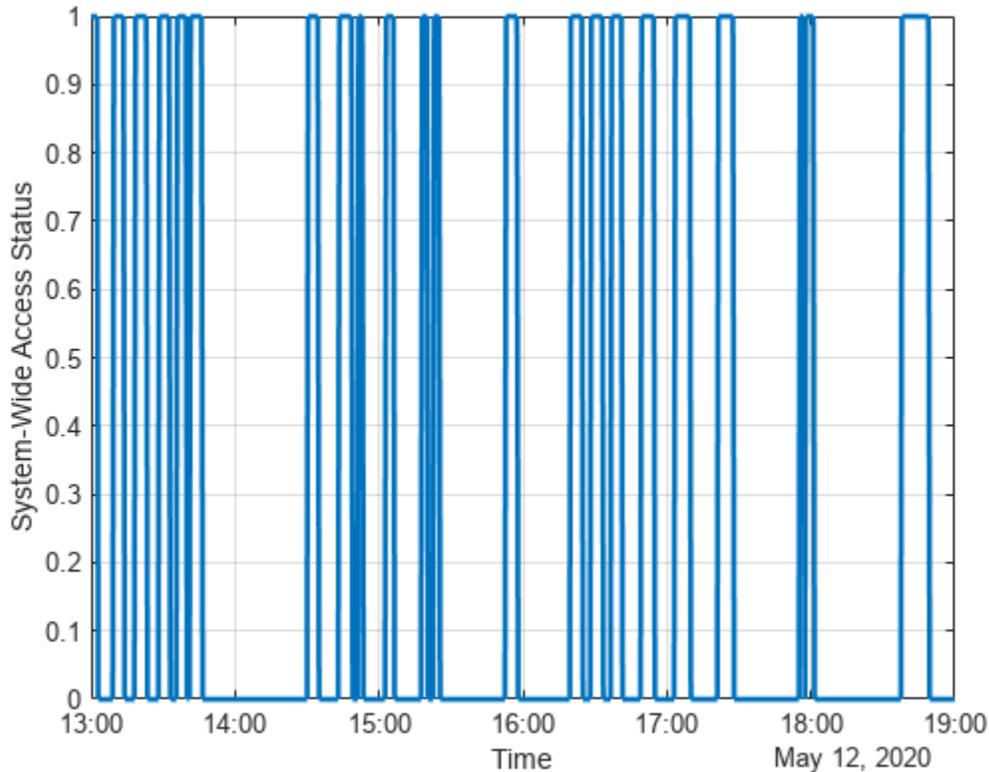
- For each camera, calculate the access status history to the site using `accessStatus`. For a given camera, this is a row vector of logicals, where each element in the vector represents the access status corresponding to a given time sample. A value of `True` indicates that the camera can photograph the site at that specific time sample.
- Perform a logical `OR` on all these row vectors corresponding to access of each camera to the site. This will result in a single row vector of logicals, in which a given element is true if at least one camera can photograph the site at the corresponding time sample for a duration of one scenario sample time of 30 seconds.
- Count the number of elements in the vector whose value is `True`. Multiply this quantity by the sample time of 30 seconds to determine the total time in seconds when at least one camera can photograph the site.
- Divide this quantity by the scenario duration of 6 hours and multiply by 100 to get the system-wide access percentage.

```
for idx = 1:numel(ac)
    [s,time] = accessStatus(ac(idx));

    if idx == 1
        % Initialize system-wide access status vector in the first iteration
        systemWideAccessStatus = s;
    else
        % Update system-wide access status vector by performing a logical OR
        % with access status for the current camera-site access
        % analysis
        systemWideAccessStatus = or(systemWideAccessStatus,s);
    end
end
```

Use `plot` to plot the system-wide access status with respect to time.

```
plot(time,systemWideAccessStatus,"LineWidth",2);
grid on;
xlabel("Time");
ylabel("System-Wide Access Status");
```



Whenever system-wide access status is 1 (True), at least one camera can photograph the site.

Use `nnz` to determine the number of elements in `systemWideAccessStatus` whose value is True.

```
n = nnz(systemWideAccessStatus)
```

```
n = 203
```

Determine the total time when at least one camera can photograph the site. This is accomplished by multiplying the number of True elements by the scenario's sample time.

```
systemWideAccessDuration = n*sc.SampleTime % seconds
```

```
systemWideAccessDuration = 6090
```

Use `seconds` to calculate the total scenario duration.

```
scenarioDuration = seconds(sc.StopTime - sc.StartTime)
```

```
scenarioDuration = 21600
```

Calculate the system-wide access percentage.

```
systemWideAccessPercentage = (systemWideAccessDuration/scenarioDuration)*100
```

```
systemWideAccessPercentage = 28.1944
```

Improve the System-Wide Access Percentage by Making the Cameras Track the Geographical Site

The default attitude configuration of the satellites is such that their yaw axes point straight down towards nadir (the point on Earth directly below the satellite). Since the cameras are aligned with the yaw axis by default, they point straight down as well. As a result, the geographical site goes outside the field of view of the cameras before their elevation angle dips below 30 degrees. Therefore, the cumulative access percentage is limited by the cameras' field of view.

If instead the cameras always point at the geographical site, the latter is always inside the cameras' field of view as long as the Earth is not blocking the line of sight. Consequently, the system-wide access percentage will now be limited by the `MinElevationAngle` of the geographical site, as opposed to the cameras' field of view. In the former case, the access intervals began and ended when the site entered and left the camera's field of view. It entered the field of view some time after the camera's elevation angle went above 30 degrees, and left the field of view before its elevation angle dipped below 30 degrees. However, if the cameras constantly point at the site, the access intervals will begin when the elevation angle rises above 30 degrees and end when it dips below 30 degrees, thereby increasing the duration of the intervals. Therefore, the system-wide access percentage will increase as well.

Since the cameras are rigidly attached to the satellites, each satellite is required to be continuously reoriented along its orbit so that its yaw axis tracks the geographical site. As the cameras are aligned with the yaw axis, they too will point at the site. Use `pointAt` to make each satellite's yaw axis track the geographical site.

```
pointAt(sat,geoSite);
```

Re-calculate the system-wide access percentage.

```
% Calculate system-wide access status
for idx = 1:numel(ac)
    [s,time] = accessStatus(ac(idx));

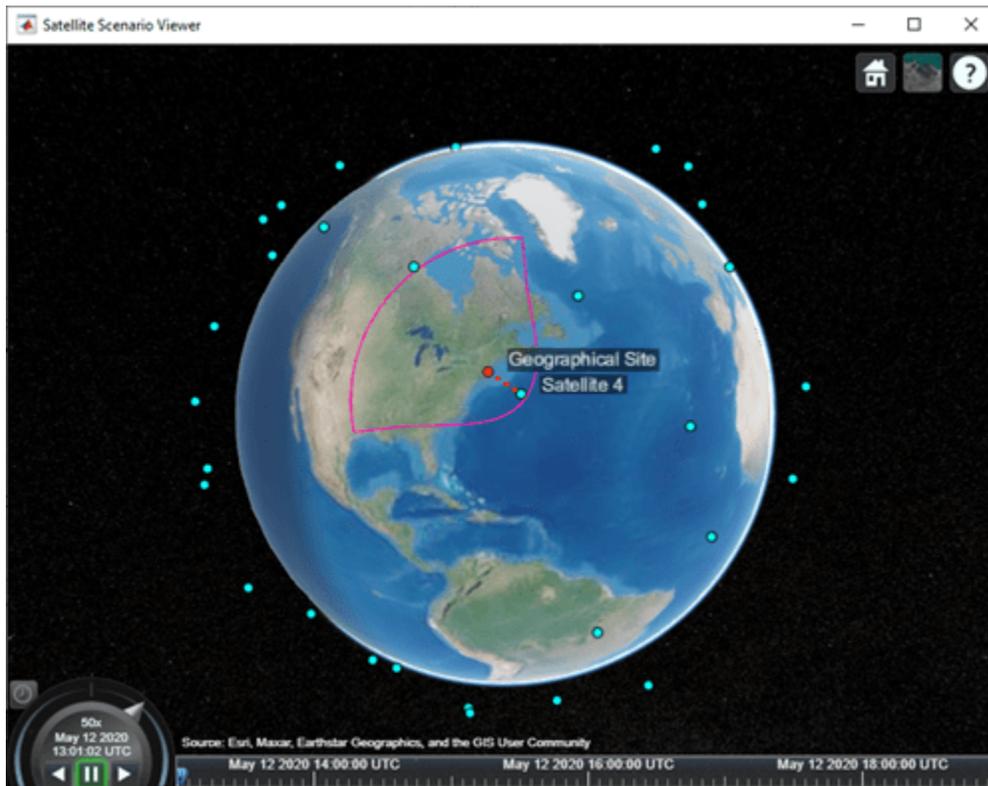
    if idx == 1
        % Initialize system-wide access status vector in the first iteration
        systemWideAccessStatus = s;
    else
        % Update system-wide access status vector by performing a logical OR
        % with access status for the current camera-site combination
        systemWideAccessStatus = or(systemWideAccessStatus,s);
    end
end

% Calculate system-wide access percentage
n = nnz(systemWideAccessStatus);
systemWideAccessDuration = n*sc.SampleTime;
systemWideAccessPercentageWithTracking = (systemWideAccessDuration/scenarioDuration)*100

systemWideAccessPercentageWithTracking = 38.3333
```

The system-wide access percentage has improved by about 36%. This is the result of the cameras continuously pointing at the geographical site. This can be visualized by using `play` again.

```
play(sc)
```



The field of view contour is no longer circular because the camera is not pointing straight down anymore as it is tracking the geographical site.

Exploring the Example

This example demonstrated how to set up access analysis between a ground station and conical sensors onboard a constellation of satellites. The conical sensors represented cameras onboard the satellites, and the ground station represented the geographical site to be photographed. The times at which cameras onboard the satellites can photograph the geographical site were determined using access analysis. Additionally, system-wide access percentage was computed to determine the percentage of time during a 6 hour period when at least one satellite can photograph the site. It was seen that these results depended on the direction at which the cameras were pointing.

These results are also a function of:

- Orbit of the satellites
- `MinElevationAngle` of the geographical site
- Mounting position and location of the cameras with respect to the satellites
- Field of view (`MaxViewAngle`) of the cameras if they are not continuously pointing at the geographical site

Modify the above parameters to your requirements and observe their influence on the access intervals and system-wide access percentage. The orbit of the satellites can be changed by explicitly specifying their Keplerian orbital elements using `satellite`. Additionally, the cameras can be mounted on `gimbals`, which can be rotated independent of the satellite. This way, the satellites can

point straight down (the default behavior), while the gimbals can be configured so that the cameras independently track the geographical site.

See Also

Objects

satelliteScenario | satellite | access | groundStation | satelliteScenarioViewer | conicalSensor | transmitter | receiver

Functions

show | play | hide

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Comparison of Orbit Propagators” on page 1-25
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Comparison of Orbit Propagators

This example compares the orbits predicted by the Two-Body-Keplerian, Simplified General Perturbations-4 (SGP4) and Simplified Deep-Space Perturbations-4 (SDP4) orbit propagators. An orbit propagator is a solver that calculates the position and velocity of an object whose motion is predominantly influenced by gravity from celestial bodies. The Two-Body-Keplerian orbit propagator is based on the relative two-body model that assumes a spherical gravity field for the Earth and neglects third body effects and other environmental perturbations, and hence, is the least accurate. The SGP4 orbit propagator accounts for secular and periodic orbital perturbations caused by Earth's geometry and atmospheric drag, and is applicable to near-Earth satellites whose orbital period is less than 225 minutes. The SDP4 orbit propagator builds upon SGP4 by accounting for solar and lunar gravity, and is applicable to satellites whose orbital period is greater than or equal to 225 minutes. The default orbit propagator for `satelliteScenario` is SGP4 for satellites whose orbital period is less than 225 minutes, and SDP4 otherwise.

Create a Satellite Scenario

Create a satellite scenario by using the `satelliteScenario` function. Set the start time to 11-May-2020 12:35:38 PM UTC, and the stop time to 13-May-2020 12:35:38 PM UTC, by using the `datetime` function. Set the sample time to 60 seconds.

```
startTime = datetime(2020,5,11,12,35,38);
stopTime = startTime + days(2);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc =
  satelliteScenario with properties:
      StartTime: 11-May-2020 12:35:38
      StopTime: 13-May-2020 12:35:38
      SampleTime: 60
      AutoSimulate: 1
      Satellites: [1x0 matlabshared.satellitescenario.Satellite]
      GroundStations: [1x0 matlabshared.satellitescenario.GroundStation]
      Viewers: [0x0 matlabshared.satellitescenario.Viewer]
      AutoShow: 1
```

Add Satellites to the Satellite Scenario

Add three satellites to the satellite scenario from the two-line element (TLE) file `eccentricOrbitSatellite.tle` by using the `satellite` function. TLE is a data format used for encoding the orbital elements of an Earth-orbiting object defined at a specific time. Assign a Two-Body-Keplerian orbit propagator to the first satellite, SGP4 to the second satellite, and SDP4 to the third satellite.

```
tleFile = "eccentricOrbitSatellite.tle";
satTwoBodyKeplerian = satellite(sc,tleFile, ...
    "Name","satTwoBodyKeplerian", ...
    "OrbitPropagator","two-body-keplerian")

satTwoBodyKeplerian =
  Satellite with properties:
```

```
        Name: satTwoBodyKeplerian
        ID: 1
    ConicalSensors: [1x0 matlabshared.satellitescenario.ConicalSensor]
        Gimbals: [1x0 matlabshared.satellitescenario.Gimbal]
    Transmitters: [1x0 satcom.satellitescenario.Transmitter]
        Receivers: [1x0 satcom.satellitescenario.Receiver]
        Accesses: [1x0 matlabshared.satellitescenario.Access]
    GroundTrack: [1x1 matlabshared.satellitescenario.GroundTrack]
        Orbit: [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator: two-body-keplerian
    MarkerColor: [0.059 1 1]
    MarkerSize: 6
    ShowLabel: true
    LabelFontColor: [1 1 1]
    LabelFontSize: 15
```

```
satSGP4 = satellite(sc,tleFile, ...
    "Name","satSGP4", ...
    "OrbitPropagator","sgp4")
```

```
satSGP4 =
    Satellite with properties:
```

```
        Name: satSGP4
        ID: 2
    ConicalSensors: [1x0 matlabshared.satellitescenario.ConicalSensor]
        Gimbals: [1x0 matlabshared.satellitescenario.Gimbal]
    Transmitters: [1x0 satcom.satellitescenario.Transmitter]
        Receivers: [1x0 satcom.satellitescenario.Receiver]
        Accesses: [1x0 matlabshared.satellitescenario.Access]
    GroundTrack: [1x1 matlabshared.satellitescenario.GroundTrack]
        Orbit: [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator: sgp4
    MarkerColor: [0.059 1 1]
    MarkerSize: 6
    ShowLabel: true
    LabelFontColor: [1 1 1]
    LabelFontSize: 15
```

```
satSDP4 = satellite(sc,tleFile, ...
    "Name","satSDP4", ...
    "OrbitPropagator","sdp4")
```

```
satSDP4 =
    Satellite with properties:
```

```
        Name: satSDP4
        ID: 3
    ConicalSensors: [1x0 matlabshared.satellitescenario.ConicalSensor]
        Gimbals: [1x0 matlabshared.satellitescenario.Gimbal]
    Transmitters: [1x0 satcom.satellitescenario.Transmitter]
        Receivers: [1x0 satcom.satellitescenario.Receiver]
        Accesses: [1x0 matlabshared.satellitescenario.Access]
    GroundTrack: [1x1 matlabshared.satellitescenario.GroundTrack]
        Orbit: [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator: sdp4
    MarkerColor: [0.059 1 1]
```

```

    MarkerSize: 6
    ShowLabel: true
    LabelFontColor: [1 1 1]
    LabelFontSize: 15

```

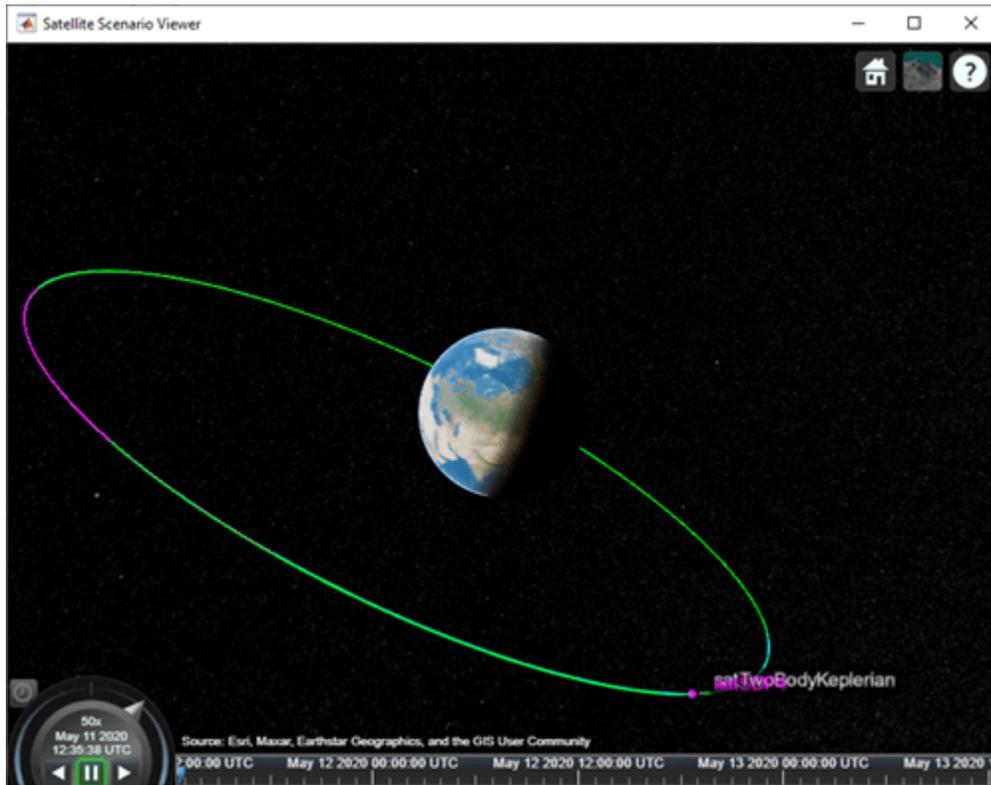
Visualize the Satellites and their Orbits

Launch a satellite scenario viewer and visualize the satellite scenario by using the `satelliteScenarioViewer` function. Set the visualizations of `satTwoBodyKeplerian` to red, `satSGP4` to green, and `satSDP4` to magenta.

```

v = satelliteScenarioViewer(sc);
satSGP4.MarkerColor = [0 1 0];
satSGP4.Orbit.LineColor = [0 1 0];
satSGP4.LabelFontColor = [0 1 0];
satSDP4.MarkerColor = [1 0 1];
satSDP4.Orbit.LineColor = [1 0 1];
satSDP4.LabelFontColor = [1 0 1];

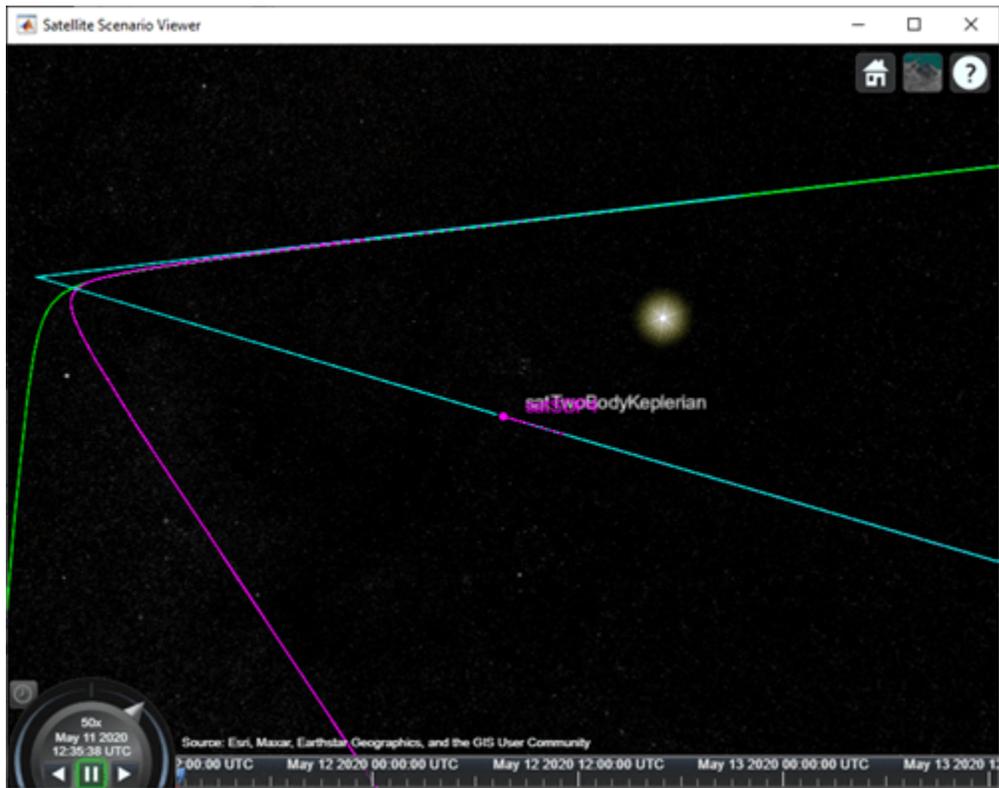
```



Focus the camera on `satTwoBodyKeplerian` by using the `camtarget` function.

```
camtarget(v, satTwoBodyKeplerian);
```

Left-click anywhere inside the satellite scenario viewer window and move the mouse while holding the click to pan the camera. Adjust the zoom level using the scroll wheel to bring all three satellites into view.



Visualize a Dynamic Animation of the Satellite Movement

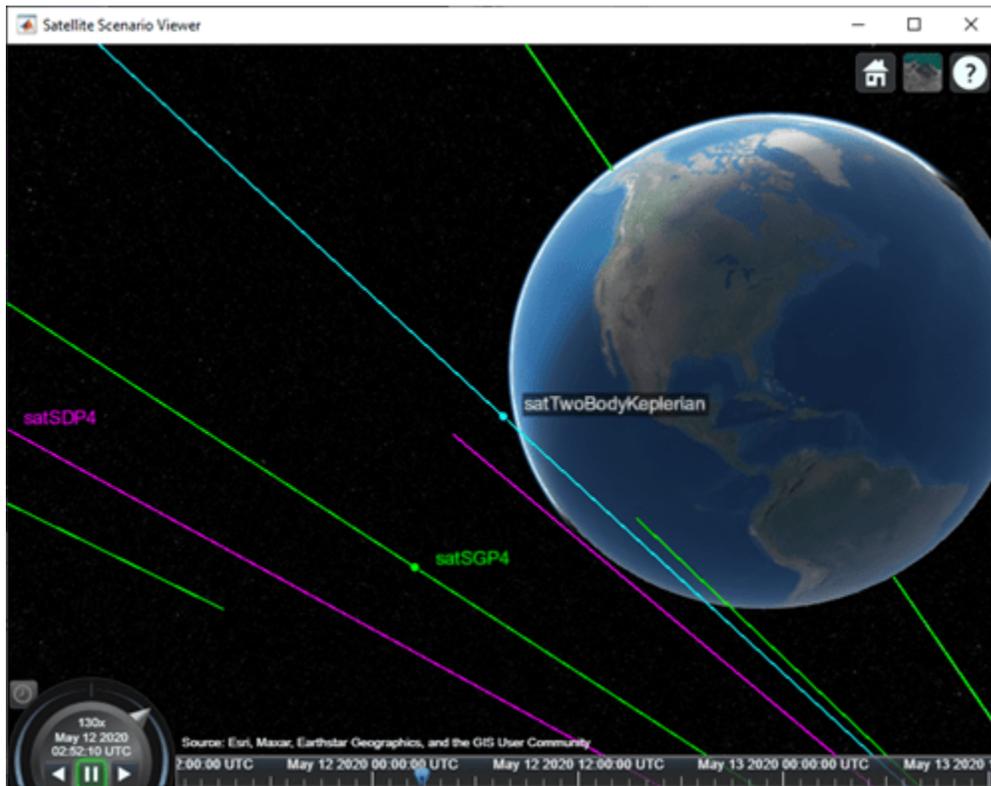
Visualize the movement of the satellites by using the `play` function on the satellite scenario. The `play` function simulates the satellite scenario from the specified `StartTime` to `StopTime` using a step size specified by `SampleTime`, and plays the results on the satellite scenario viewer.

```
play(sc)
```

Use the playback controls located at the bottom of the satellite scenario viewer window to control the playback speed and direction. Focus the camera again on `satTwoBodyKeplerian` by using the `camtarget` function, and bring all three satellites into view by adjusting the zoom level.

```
camtarget(v, satTwoBodyKeplerian);
```

The positions of the three satellites diverge over time.



Obtain the Position and Velocity History of the Satellites

Return the position and velocity history of the satellites in the Geocentric Celestial Reference Frame (GCRF) by using the `states` function.

```
[positionTwoBodyKeplerian,velocityTwoBodyKeplerian,time] = states(satTwoBodyKeplerian);
[positionSGP4,velocitySGP4] = states(satSGP4);
[positionSDP4,velocitySDP4] = states(satSDP4);
```

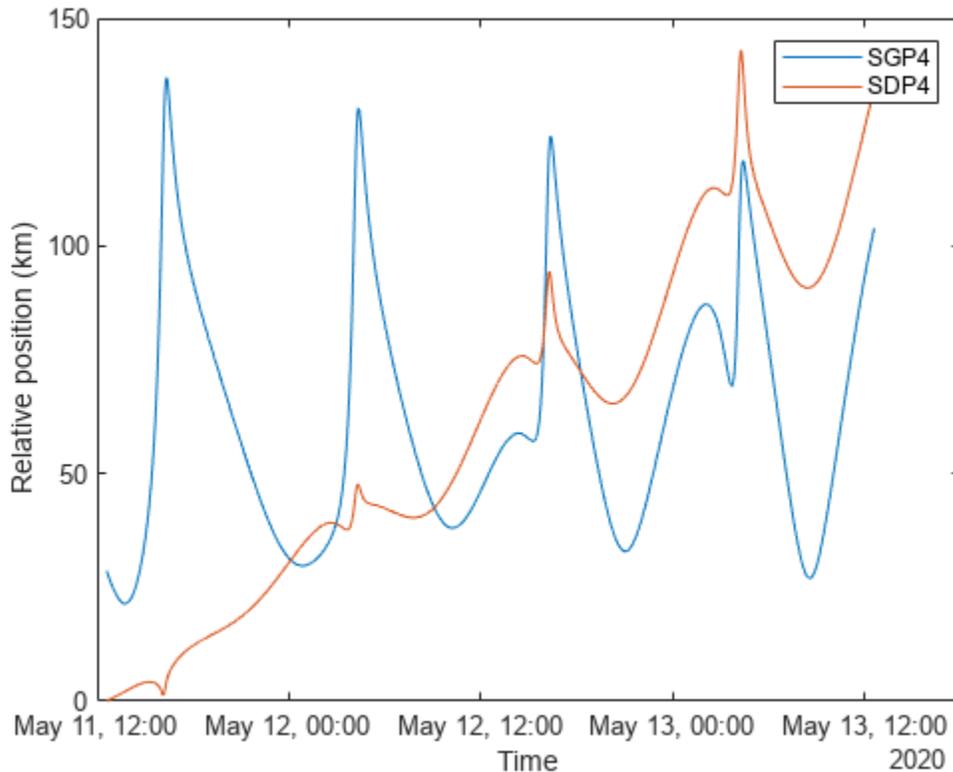
Plot Magnitude of Relative Position with Respect to Two-Body-Keplerian Prediction

Calculate the magnitude of the relative position of `satSGP4` and `satSDP4` with respect to `satTwoBodyKeplerian` by using the `vecnorm` function.

```
sgp4RelativePosition = vecnorm(positionSGP4 - positionTwoBodyKeplerian,2,1);
sdp4RelativePosition = vecnorm(positionSDP4 - positionTwoBodyKeplerian,2,1);
```

Plot the magnitude of the relative positions in kilometers of `satSGP4` and `satSDP4` with respect to that of `satTwoBodyKeplerian` by using the `plot` function.

```
sgp4RelativePositionKm = sgp4RelativePosition/1000;
sdp4RelativePositionKm = sdp4RelativePosition/1000;
plot(time,sgp4RelativePositionKm,time,sdp4RelativePositionKm)
xlabel("Time")
ylabel("Relative position (km)")
legend("SGP4","SDP4")
```



The initial relative position of `satSGP4` is non-zero and that of `satSDP4` is zero because the initial positions of `satTwoBodyKeplerian` and `satSDP4` are calculated from the TLE file using the SDP4 orbit propagator, while the initial position of `satSGP4` is calculated using the SGP4 orbit propagator. Over time, the position of `satSDP4` deviates from that of `satTwoBodyKeplerian` because the subsequent positions of the former are calculated using the SDP4 orbit propagator, while those of the latter are calculated using the Two-Body-Keplerian orbit propagator. The SDP4 orbit propagator provides higher precision because unlike the Two-Body-Keplerian orbit propagator, it accounts for oblateness of the Earth, atmospheric drag, and gravity from the sun and the moon.

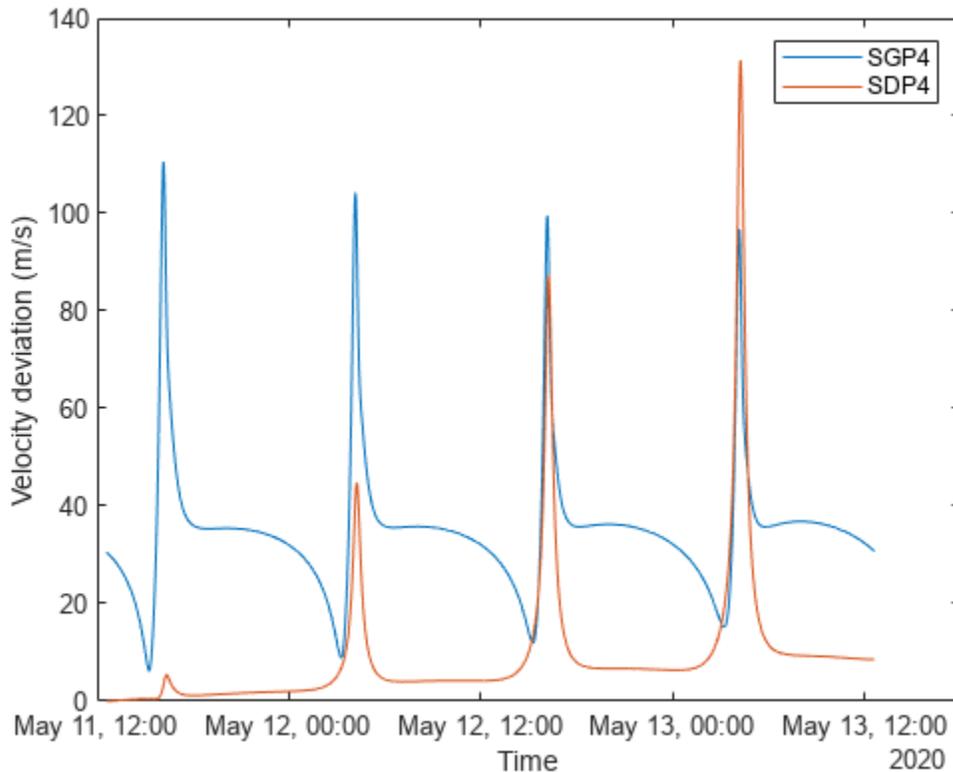
Plot Magnitude of Relative Velocity with Respect to Two-Body-Keplerian Prediction

Calculate the magnitude of the relative velocity of `satSGP4` and `satSDP4` with respect to `satTwoBodyKeplerian` by using the `vecnorm` function.

```
sgp4RelativeVelocity = vecnorm(velocitySGP4 - velocityTwoBodyKeplerian,2,1);
sdp4RelativeVelocity = vecnorm(velocitySDP4 - velocityTwoBodyKeplerian,2,1);
```

Plot the magnitude of the relative velocities in meters per second of `satSGP4` and `satSDP4` with respect to `satTwoBodyKeplerian` by using the `plot` function.

```
plot(time,sgp4RelativeVelocity,time,sdp4RelativeVelocity)
xlabel("Time")
ylabel("Velocity deviation (m/s)")
legend("SGP4","SDP4")
```



The initial relative velocity of `satSDP4` is zero because just like the initial position, the initial velocity of `satTwoBodyKeplerian` and `satSDP4` are also calculated from the TLE file using the SDP4 orbit propagator. Over time, the velocity of `satSDP4` deviates from that of `satTwoBodyKeplerian` because at all other times, the velocity of `satTwoBodyKeplerian` is calculated using the Two-Body-Keplerian orbit propagator, which has lower precision when compared to that of the SDP4 orbit propagator that is used for calculating the velocity of `satSDP4`. The spikes correspond to the perigee (the closest point in the orbit from the center of mass of the Earth), where the magnitudes of the velocity errors are pronounced.

Conclusion

The deviations in the plots are the result of varying levels of accuracy of the three orbit propagators. The Two-Body-Keplerian orbit propagator is the least accurate as it assumes that the gravity field of the Earth is spherical, and also neglects all other sources of orbital perturbations. The SGP4 orbit propagator is more accurate as it accounts for the oblateness of the Earth and atmospheric drag. The SDP4 orbit propagator is the most accurate among the three because it also accounts for solar and lunar gravity, which is more pronounced in this example because the orbital period is greater than 225 minutes, thereby taking the satellite farther away from the Earth.

See Also

Objects

`satelliteScenario` | `satellite` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor` | `transmitter` | `receiver`

Functions

show | play | hide

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to Ground Station” on page 1-14
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Modeling Satellite Constellations Using Ephemeris Data

This example demonstrates how to add time-stamped ephemeris data for a constellation of 24 satellites (similar to ESA Galileo GNSS constellation) to a satellite scenario for access analysis. The example uses data generated by the Aerospace Blockset **Orbit Propagator** block. For more information, see the Aerospace Blockset example *Constellation Modeling with the Orbit Propagator Block*.

The **satelliteScenario** object supports loading previously generated, time-stamped satellite ephemeris data into a scenario from a **timeseries** or **timetable** object. An ephemeris is a table containing position (and optionally velocity) state information of a satellite during a given period of time. Ephemeris data used to add satellites to the scenario object is interpolated via the **makima** interpolation method to align with the scenario time steps. This allows you to incorporate data generated by a Simulink model into either a new or existing satelliteScenario.

Define Mission Parameters and Constellation Initial Conditions

Specify a start date and duration for the mission. This example uses MATLAB structures to organize mission data. These structures make accessing data later in the example more intuitive. They also help declutter the global base workspace.

```
mission.StartDate = datetime(2020, 11, 30, 22, 23, 24);
mission.Duration = hours(24);
```

The constellation in this example is a Walker-Delta constellation modeled similar to Galileo, the European GNSS (global navigation satellite system) constellation. The constellation consists of 24 satellites in medium Earth orbit (MEO). The satellites' Keplerian orbital elements at the mission start date epoch are:

```
mission.ConstellationDefinition = table( ...
    29599.8e3 * ones(24,1), ... % Semi-major axis (m)
    0.0005 * ones(24,1), ... % Eccentricity
    56 * ones(24,1), ... % Inclination (deg)
    350 * ones(24,1), ... % Right ascension of the ascending node (deg)
    sort(repmat([0 120 240], 1,8))', ... % Argument of periapsis (deg)
    [0:45:315, 15:45:330, 30:45:345]', ... % True anomaly (deg)
    'VariableNames', ["a (m)", "e", "i (deg)", ...
    "Ω (deg)", "ω (deg)", "ν (deg)"]);
mission.ConstellationDefinition
```

```
ans=24x6 table
    a (m)         e         i (deg)    Ω (deg)    ω (deg)    ν (deg)
    _____ _____ _____ _____ _____ _____
    2.96e+07    0.0005     56         350         0           0
    2.96e+07    0.0005     56         350         0           45
    2.96e+07    0.0005     56         350         0           90
    2.96e+07    0.0005     56         350         0          135
    2.96e+07    0.0005     56         350         0          180
    2.96e+07    0.0005     56         350         0          225
    2.96e+07    0.0005     56         350         0          270
    2.96e+07    0.0005     56         350         0          315
    2.96e+07    0.0005     56         350        120           15
    2.96e+07    0.0005     56         350        120           60
    2.96e+07    0.0005     56         350        120          105
```

2.96e+07	0.0005	56	350	120	150
2.96e+07	0.0005	56	350	120	195
2.96e+07	0.0005	56	350	120	240
2.96e+07	0.0005	56	350	120	285
2.96e+07	0.0005	56	350	120	330
:					

Load Ephemeris Timeseries Data

The timeseries objects contain position and velocity data for all 24 satellites in the constellation. The data is referenced in the International Terrestrial Reference frame (ITRF), which is an Earth-centered Earth-fixed (ECEF) coordinate system. The data was generated using the Aerospace Blockset **Orbit Propagator** block. For more information, see the Aerospace Blockset example *Constellation Modeling with the Orbit Propagator Block*.

```
mission.Ephemeris = load("SatelliteScenarioEphemerisData.mat", "TimeseriesPosITRF", "TimeseriesVelITRF");
mission.Ephemeris.TimeseriesPosITRF
```

```
timeseries

Common Properties:
    Name: ''
    Time: [57x1 double]
    TimeInfo: [1x1 tsdata.timemetadata]
    Data: [24x3x57 double]
    DataInfo: [1x1 tsdata.datametadata]
```

More properties, Methods

```
mission.Ephemeris.TimeseriesVelITRF
```

```
timeseries

Common Properties:
    Name: ''
    Time: [57x1 double]
    TimeInfo: [1x1 tsdata.timemetadata]
    Data: [24x3x57 double]
    DataInfo: [1x1 tsdata.datametadata]
```

More properties, Methods

Load the Satellite Ephemerides into a satelliteScenario Object

Create a satellite scenario object for the analysis.

```
scenario = satelliteScenario(mission.StartDate, mission.StartDate + hours(24), 60);
```

Use the **satellite** method to add all 24 satellites to the satellite scenario from the ECEF position and velocity timeseries objects. This example uses position and velocity information; however satellites can also be added from position data only and velocity states are then estimated. Available coordinate frames for Name-Value pair `CoordinateFrame` are "ECEF", "Inertial", and "Geographic". If the timeseries object contains a value for `ts.TimeInfo.StartDate`, the method uses that value as the epoch for the timeseries object. If no `StartDate` is defined, the method uses the scenario start date by default.

```
sat = satellite(scenario, mission.Ephemeris.TimeseriesPosITRF, mission.Ephemeris.TimeseriesVelITRF,
    CoordinateFrame="ecef", Name="GALILEO " + (1:24))
```

```
sat =
    1x24 Satellite array with properties:
```

```
    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

```
disp(scenario)
```

```
satelliteScenario with properties:
```

```
    StartTime: 30-Nov-2020 22:23:24
    StopTime: 01-Dec-2020 22:23:24
    SampleTime: 60
    AutoSimulate: 1
    Satellites: [1x24 matlabshared.satellitescenario.Satellite]
    GroundStations: [1x0 matlabshared.satellitescenario.GroundStation]
    Viewers: [0x0 matlabshared.satellitescenario.Viewer]
    AutoShow: 1
```

Alternatively, satellites can also be added as ephemerides to the satellite scenario as a MATLAB **timetable**, **table**, or **tscollection**. For example, a **timetable** containing the first 3 satellites of the position **timeseries** object in the previous section, formatted for use with **satelliteScenario** objects is shown below.

- Satellites are represented by variables (column headers).
- Each row contains a position vector associated with the row's Time property.

```
timetable(...
    datetime(getabstime(mission.Ephemeris.TimeseriesPosITRF), Locale="en_US"), ...
    squeeze(mission.Ephemeris.TimeseriesPosITRF.Data(1,:,:))', ...
    squeeze(mission.Ephemeris.TimeseriesPosITRF.Data(2,:,:))', ...
    squeeze(mission.Ephemeris.TimeseriesPosITRF.Data(3,:,:))',...
    VariableNames=["Satellite_1", "Satellite_2", "Satellite_3"])
```

```
ans=57x3 timetable
```

Time	Satellite_1	Satellite_2	Satellite_3
30-Nov-2020 22:23:24	1.8249e+07	-2.2904e+07	-4.2009e+06
30-Nov-2020 22:23:38	1.8252e+07	-2.2909e+07	-4.1563e+06
30-Nov-2020 22:24:53	1.8268e+07	-2.2937e+07	-3.933e+06

30-Nov-2020	22:31:05	1.8326e+07	-2.3055e+07	-2.8121e+06	2.3185e+07	-1.028e+07
30-Nov-2020	22:48:39	1.8326e+07	-2.3223e+07	3.9182e+05	2.2005e+07	-8.9966e+06
30-Nov-2020	23:08:30	1.8076e+07	-2.3078e+07	3.9992e+06	2.0643e+07	-7.2057e+06
30-Nov-2020	23:28:27	1.7624e+07	-2.2538e+07	7.5358e+06	1.9321e+07	-5.0678e+06
30-Nov-2020	23:50:59	1.6968e+07	-2.1428e+07	1.1328e+07	1.7977e+07	-2.3021e+06
01-Dec-2020	00:14:27	1.6244e+07	-1.9712e+07	1.4937e+07	1.6838e+07	8.7771e+05
01-Dec-2020	00:38:42	1.5585e+07	-1.7375e+07	1.8189e+07	1.6017e+07	4.355e+05
01-Dec-2020	01:04:35	1.5124e+07	-1.4345e+07	2.1006e+07	1.5585e+07	8.1065e+05
01-Dec-2020	01:31:17	1.5035e+07	-1.079e+07	2.3096e+07	1.562e+07	1.1816e+05
01-Dec-2020	01:58:58	1.5443e+07	-6.8501e+06	2.4303e+07	1.6102e+07	1.5274e+05
01-Dec-2020	02:27:08	1.6406e+07	-2.8152e+06	2.4478e+07	1.6925e+07	1.8197e+05
01-Dec-2020	02:55:18	1.7869e+07	1.001e+06	2.3582e+07	1.7894e+07	2.0376e+05
01-Dec-2020	03:23:29	1.9711e+07	4.381e+06	2.1653e+07	1.8787e+07	2.1739e+05
	:					

Set Graphical Properties on the Satellites

Set satellite in the same orbital plane to have the same orbit color.

```
set(sat(1:8), MarkerColor="#FF6929");
set(sat(9:16), MarkerColor="#139FFF");
set(sat(17:24), MarkerColor="#64D413");
orbit = [sat(:).Orbit];
set(orbit(1:8), LineColor="#FF6929");
set(orbit(9:16), LineColor="#139FFF");
set(orbit(17:24), LineColor="#64D413");
```

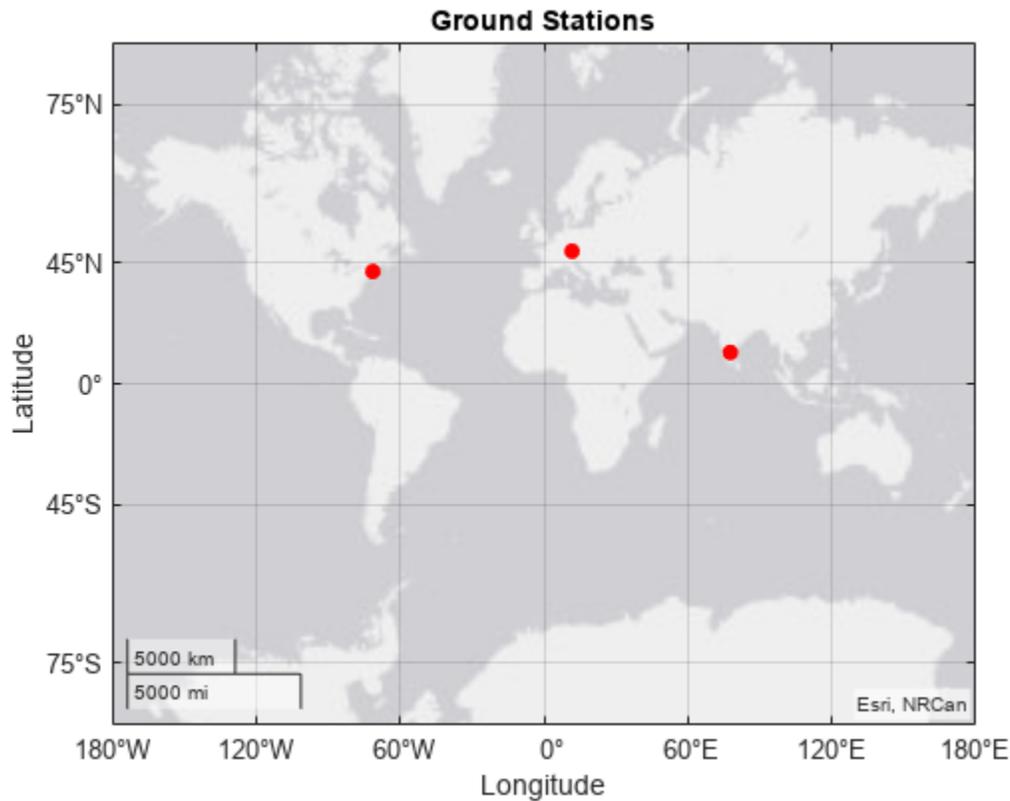
Add Ground Stations to Scenario

To provide accurate positioning data, a location on Earth must have access to at least 4 satellites in the constellation at any given time. In this example, use three locations to compare total constellation access over the 1 day analysis window to different regions of Earth:

- Natick, Massachusetts, USA (42.30048°, -71.34908°)
- München, Germany (48.23206°, 11.68445°)
- Bangalore, India (12.94448°, 77.69256°)

```
gsUS = groundStation(scenario, 42.30048, -71.34908, ...
    MinElevationAngle=10, Name="Natick");
gsUS.MarkerColor = "red";
gsDE = groundStation(scenario, 48.23206, 11.68445, ...
    MinElevationAngle=10, Name="Munchen");
gsDE.MarkerColor = "red";
gsIN = groundStation(scenario, 12.94448, 77.69256, ...
    MinElevationAngle=10, Name="Bangalore");
gsIN.MarkerColor = "red";

figure
geoscatter([gsUS.Latitude gsDE.Latitude gsIN.Latitude], ...
    [gsUS.Longitude gsDE.Longitude gsIN.Longitude], "red", "filled")
geolimits([-75 75], [-180 180])
title("Ground Stations")
```



Compute Ground Station to Satellite Access (Line-of-Sight Visibility)

Calculate line-of-sight access between the ground stations and each individual satellite using the `access` method.

```
accessUS = access(gsUS, sat);
accessDE = access(gsDE, sat);
accessIN = access(gsIN, sat);
```

Set access colors to match orbital plane colors assigned earlier in the example.

```
set(accessUS, LineWidth="1");
set(accessUS(1:8), LineColor="#FF6929");
set(accessUS(9:16), LineColor="#139FFF");
set(accessUS(17:24), LineColor="#64D413");
```

```
set(accessDE, LineWidth="1");
set(accessDE(1:8), LineColor="#FF6929");
set(accessDE(9:16), LineColor="#139FFF");
set(accessDE(17:24), LineColor="#64D413");
```

```
set(accessIN, LineWidth="1");
set(accessIN(1:8), LineColor="#FF6929");
set(accessIN(9:16), LineColor="#139FFF");
set(accessIN(17:24), LineColor="#64D413");
```

View the full access table between each ground station and all satellites in the constellation as tables. Sort the access intervals by interval start time. Satellites added from ephemeris data do not display values for StartOrbit and EndOrbit.

```
intervalsUS = accessIntervals(accessUS);
intervalsUS = sortrows(intervalsUS, "StartTime", "ascend")
```

intervalsUS=40x8 table

Source	Target	IntervalNumber	StartTime	EndTime
"Natick"	"GALILEO 1"	1	30-Nov-2020 22:23:24	01-Dec-2020 04:04:24
"Natick"	"GALILEO 2"	1	30-Nov-2020 22:23:24	01-Dec-2020 01:24:24
"Natick"	"GALILEO 3"	1	30-Nov-2020 22:23:24	30-Nov-2020 22:57:24
"Natick"	"GALILEO 12"	1	30-Nov-2020 22:23:24	01-Dec-2020 00:00:24
"Natick"	"GALILEO 13"	1	30-Nov-2020 22:23:24	30-Nov-2020 23:05:24
"Natick"	"GALILEO 18"	1	30-Nov-2020 22:23:24	01-Dec-2020 04:00:24
"Natick"	"GALILEO 19"	1	30-Nov-2020 22:23:24	01-Dec-2020 01:42:24
"Natick"	"GALILEO 20"	1	30-Nov-2020 22:23:24	30-Nov-2020 22:46:24
"Natick"	"GALILEO 11"	1	30-Nov-2020 22:25:24	01-Dec-2020 00:18:24
"Natick"	"GALILEO 17"	1	30-Nov-2020 22:50:24	01-Dec-2020 05:50:24
"Natick"	"GALILEO 8"	1	30-Nov-2020 23:20:24	01-Dec-2020 07:09:24
"Natick"	"GALILEO 7"	1	01-Dec-2020 01:26:24	01-Dec-2020 10:00:24
"Natick"	"GALILEO 24"	1	01-Dec-2020 01:40:24	01-Dec-2020 07:12:24
"Natick"	"GALILEO 14"	1	01-Dec-2020 03:56:24	01-Dec-2020 07:15:24
"Natick"	"GALILEO 6"	1	01-Dec-2020 04:05:24	01-Dec-2020 12:14:24
"Natick"	"GALILEO 23"	1	01-Dec-2020 04:10:24	01-Dec-2020 08:03:24
:				

```
intervalsDE = accessIntervals(accessDE);
intervalsDE = sortrows(intervalsDE, "StartTime", "ascend")
```

intervalsDE=40x8 table

Source	Target	IntervalNumber	StartTime	EndTime
"Munchen"	"GALILEO 2"	1	30-Nov-2020 22:23:24	01-Dec-2020 04:34:24
"Munchen"	"GALILEO 3"	1	30-Nov-2020 22:23:24	01-Dec-2020 01:58:24
"Munchen"	"GALILEO 4"	1	30-Nov-2020 22:23:24	30-Nov-2020 23:05:24
"Munchen"	"GALILEO 10"	1	30-Nov-2020 22:23:24	30-Nov-2020 23:58:24
"Munchen"	"GALILEO 19"	1	30-Nov-2020 22:23:24	01-Dec-2020 01:36:24
"Munchen"	"GALILEO 20"	1	30-Nov-2020 22:23:24	01-Dec-2020 00:15:24
"Munchen"	"GALILEO 21"	1	30-Nov-2020 22:23:24	30-Nov-2020 22:28:24
"Munchen"	"GALILEO 9"	1	30-Nov-2020 22:34:24	01-Dec-2020 02:22:24
"Munchen"	"GALILEO 18"	1	30-Nov-2020 22:41:24	01-Dec-2020 02:31:24
"Munchen"	"GALILEO 1"	1	30-Nov-2020 23:05:24	01-Dec-2020 06:42:24
"Munchen"	"GALILEO 16"	1	30-Nov-2020 23:29:24	01-Dec-2020 04:47:24
"Munchen"	"GALILEO 15"	1	01-Dec-2020 00:50:24	01-Dec-2020 07:27:24
"Munchen"	"GALILEO 17"	1	01-Dec-2020 01:05:24	01-Dec-2020 03:00:24
"Munchen"	"GALILEO 8"	1	01-Dec-2020 01:57:24	01-Dec-2020 08:25:24
"Munchen"	"GALILEO 14"	1	01-Dec-2020 02:36:24	01-Dec-2020 10:19:24
"Munchen"	"GALILEO 7"	1	01-Dec-2020 04:35:24	01-Dec-2020 09:43:24
:				

```
intervalsIN = accessIntervals(accessIN);
intervalsIN = sortrows(intervalsIN, "StartTime", "ascend")
```

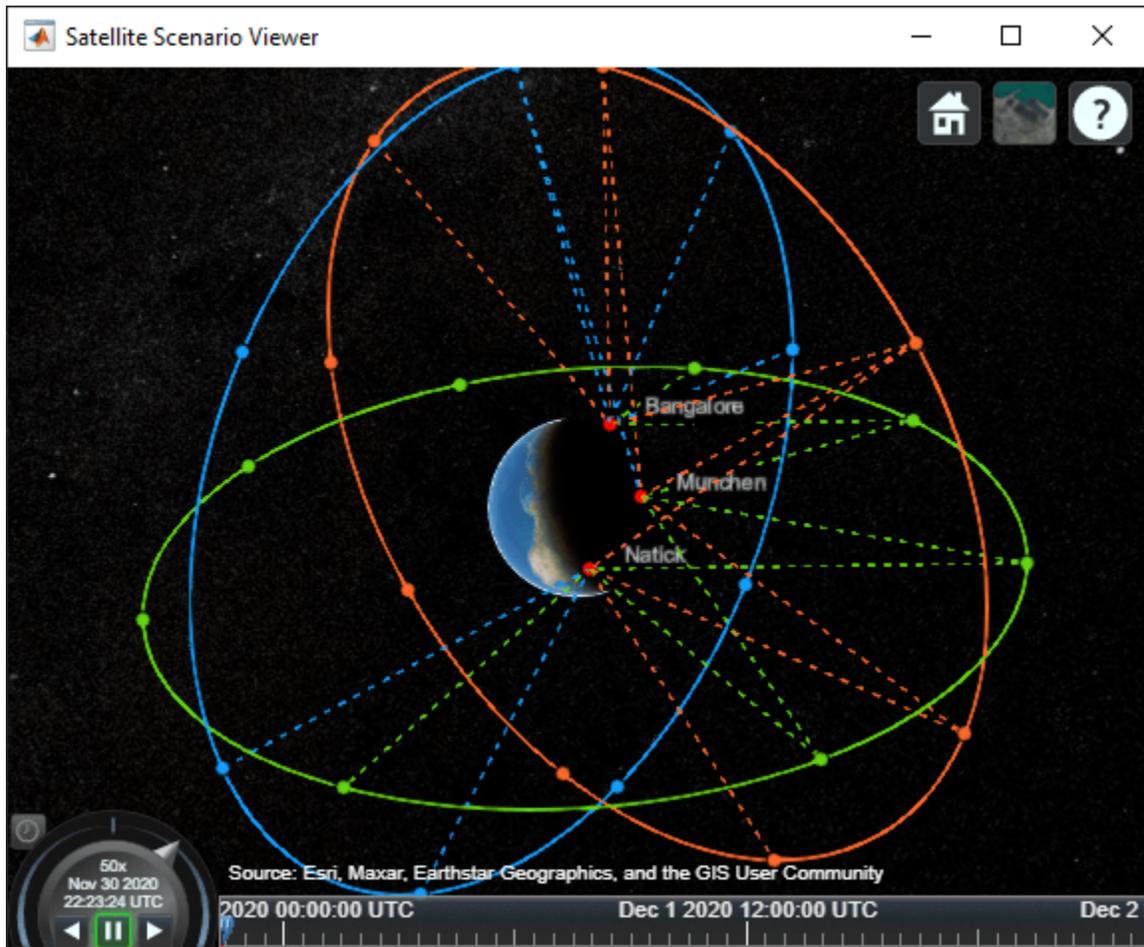
intervalsIN=31x8 table

Source	Target	IntervalNumber	StartTime	EndTime
"Bangalore"	"GALILEO 3"	1	30-Nov-2020 22:23:24	01-Dec-2020 05:12:24
"Bangalore"	"GALILEO 4"	1	30-Nov-2020 22:23:24	01-Dec-2020 02:59:24
"Bangalore"	"GALILEO 5"	1	30-Nov-2020 22:23:24	01-Dec-2020 00:22:24
"Bangalore"	"GALILEO 9"	1	30-Nov-2020 22:23:24	01-Dec-2020 03:37:24
"Bangalore"	"GALILEO 10"	1	30-Nov-2020 22:23:24	01-Dec-2020 00:09:24
"Bangalore"	"GALILEO 16"	1	30-Nov-2020 22:23:24	01-Dec-2020 08:44:24
"Bangalore"	"GALILEO 21"	1	30-Nov-2020 22:23:24	30-Nov-2020 23:25:24
"Bangalore"	"GALILEO 22"	1	30-Nov-2020 22:23:24	30-Nov-2020 22:58:24
"Bangalore"	"GALILEO 15"	1	01-Dec-2020 00:17:24	01-Dec-2020 11:16:24
"Bangalore"	"GALILEO 2"	1	01-Dec-2020 00:25:24	01-Dec-2020 07:10:24
"Bangalore"	"GALILEO 22"	2	01-Dec-2020 00:48:24	01-Dec-2020 05:50:24
"Bangalore"	"GALILEO 21"	2	01-Dec-2020 01:32:24	01-Dec-2020 08:29:24
"Bangalore"	"GALILEO 1"	1	01-Dec-2020 03:06:24	01-Dec-2020 07:17:24
"Bangalore"	"GALILEO 20"	1	01-Dec-2020 03:36:24	01-Dec-2020 12:38:24
"Bangalore"	"GALILEO 14"	1	01-Dec-2020 05:48:24	01-Dec-2020 13:29:24
"Bangalore"	"GALILEO 19"	1	01-Dec-2020 05:53:24	01-Dec-2020 17:06:24
:				

View the Satellite Scenario

Open a 3-D viewer window of the scenario. The viewer window contains all 24 satellites and the three ground stations defined earlier in this example. A line is drawn between each ground station and satellite during their corresponding access intervals. Hide the details of the satellites and ground stations by setting the `ShowDetails` name-value pair to `false`. Show satellite orbits and labels for the ground station locations.

```
viewer3D = satelliteScenarioViewer(scenario, ShowDetails=false);
show(sat.Orbit);
gsUS.ShowLabel = true;
gsUS.LabelFontSize = 11;
gsDE.ShowLabel = true;
gsDE.LabelFontSize = 11;
gsIN.ShowLabel = true;
gsIN.LabelFontSize = 11;
```



Compare Access Between Ground Stations

Calculate access status between each satellite and ground station using the `accessStatus` method. Each row of the output array corresponds with a satellite in the constellation. Each column corresponds with time steps in the scenario. A value of `True` indicates that the satellite can access the aircraft at that specific time sample. The second output of `accessStatus` contains the time steps of the scenario. Plot cumulative access for each ground station over the one day analysis window.

```
[statusUS, timeSteps] = accessStatus(accessUS);
statusDE = accessStatus(accessDE);
statusIN = accessStatus(accessIN);
```

```
% Sum cumulative access at each timestep
```

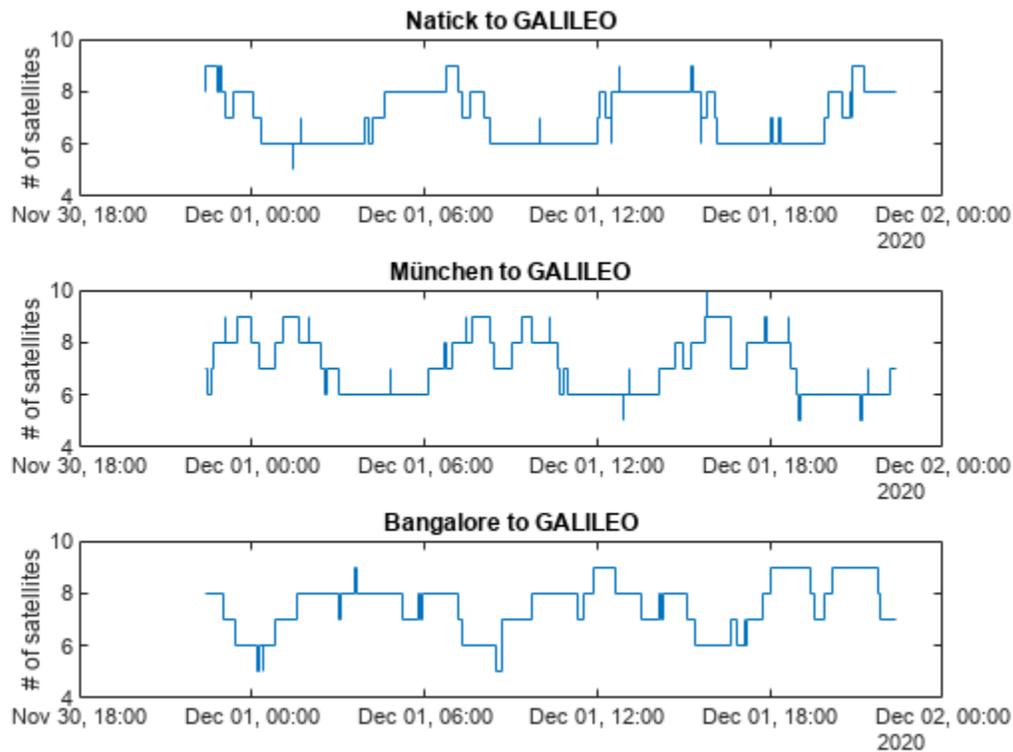
```
statusUS = sum(statusUS, 1);
statusDE = sum(statusDE, 1);
statusIN = sum(statusIN, 1);
```

```
subplot(3,1,1);
stairs(timeSteps, statusUS);
title("Natick to GALILEO")
ylabel("# of satellites")
subplot(3,1,2);
stairs(timeSteps, statusDE);
```

```

title("München to GALILEO")
ylabel("# of satellites")
subplot(3,1,3);
stairs(timeSteps, statusIN);
title("Bangalore to GALILEO")
ylabel("# of satellites")

```



Collect access interval metrics for each ground station in a table for comparison.

```

statusTable = [table(height(intervalsUS), height(intervalsDE), height(intervalsIN)); ...
  table(sum(intervalsUS.Duration)/3600, sum(intervalsDE.Duration)/3600, sum(intervalsIN.Duration)/3600); ...
  table(mean(intervalsUS.Duration/60), mean(intervalsDE.Duration/60), mean(intervalsIN.Duration)/60); ...
  table(min(statusUS), min(statusDE), min(statusIN)); ...
  table(max(statusUS), max(statusDE), max(statusIN))];
statusTable.Properties.VariableNames = ["Natick", "München", "Bangalore"];
statusTable.Properties.RowNames = ["Total # of intervals", "Total interval time (hrs)", ...
  "Mean interval length (min)", "Mean # of satellites in view", ...
  "Min # of satellites in view", "Max # of satellites in view"];
statusTable

```

statusTable=6×3 table

	Natick	München	Bangalore
Total # of intervals	40	40	31
Total interval time (hrs)	167.88	169.95	180.42
Mean interval length (min)	251.82	254.93	349.19

Mean # of satellites in view	7.018	7.1041	7.5337
Min # of satellites in view	5	5	5
Max # of satellites in view	9	10	9

Walker-Delta constellations like Galileo are evenly distributed across longitudes. Natick and München are located at similar latitudes, and therefore have very similar access characteristics with respect to the constellation. Bangalore is at a latitude closer to the equator. Despite having a lower number of individual access intervals, it has the highest average number of satellites in view, the highest overall interval time, and the longest average interval duration (by about 95 minutes). All locations always have at least 4 satellites in view, as is required for GNSS trilateration.

References

[1] Wertz, James R, David F. Everett, and Jeffery J. Puschell. *Space Mission Engineering: The New Smad*. Hawthorne, CA: Microcosm Press, 2011. Print.

[2] The European Space Agency: Galileo Facts and Figures. https://www.esa.int/Applications/Navigation/Galileo/Facts_and_figures

See Also

Objects

satelliteScenario | satellite | access | groundStation | satelliteScenarioViewer | conicalSensor | transmitter | receiver

Functions

show | play | hide

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to Ground Station” on page 1-14
- “Comparison of Orbit Propagators” on page 1-25
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Estimate GNSS Receiver Position with Simulated Satellite Constellations

Track the position of a ground vehicle using a simulated Global Navigation Satellite System (GNSS) receiver. The satellites are simulated using the `satelliteScenario` object, the satellite signal processing of the receiver are simulated using the `lookangles` (Navigation Toolbox) and `pseudoranges` (Navigation Toolbox) functions, and the receiver position is estimated with the `receiverposition` (Navigation Toolbox) function.

Overview

This example focuses on the *space segment*, or satellite constellations, and the GNSS sensor equipment for a satellite system. To obtain an estimate of the GNSS receiver position, the navigation processor requires the satellite positions from the space segment and the pseudoranges from the ranging processor in the receiver.

Specify Simulation Parameters

Load the MAT-file that contains the ground-truth position and velocity of a ground vehicle travelling toward the Natick, MA campus of The MathWorks, Inc.

Specify the start, stop, and sample time of the simulation. Also, specify the mask angle, or minimum elevation angle, of the GNSS receiver. Finally, specify the RINEX navigation message file for the initial satellite orbital parameters.

```
% Load ground truth trajectory.
load("routeNatickMA.mat", "lat", "lon", "pos", "vel", "lla0");
recPos = pos;
recVel = vel;

% Specify simulation times.
startTime = datetime(2021,6,24,8,0,0, "TimeZone", "America/New_York");
simulationSteps = size(pos,1);
dt = 1;
stopTime = startTime + seconds((simulationSteps-1)*dt);

% Specify mask angle.
maskAngle = 5; % degrees

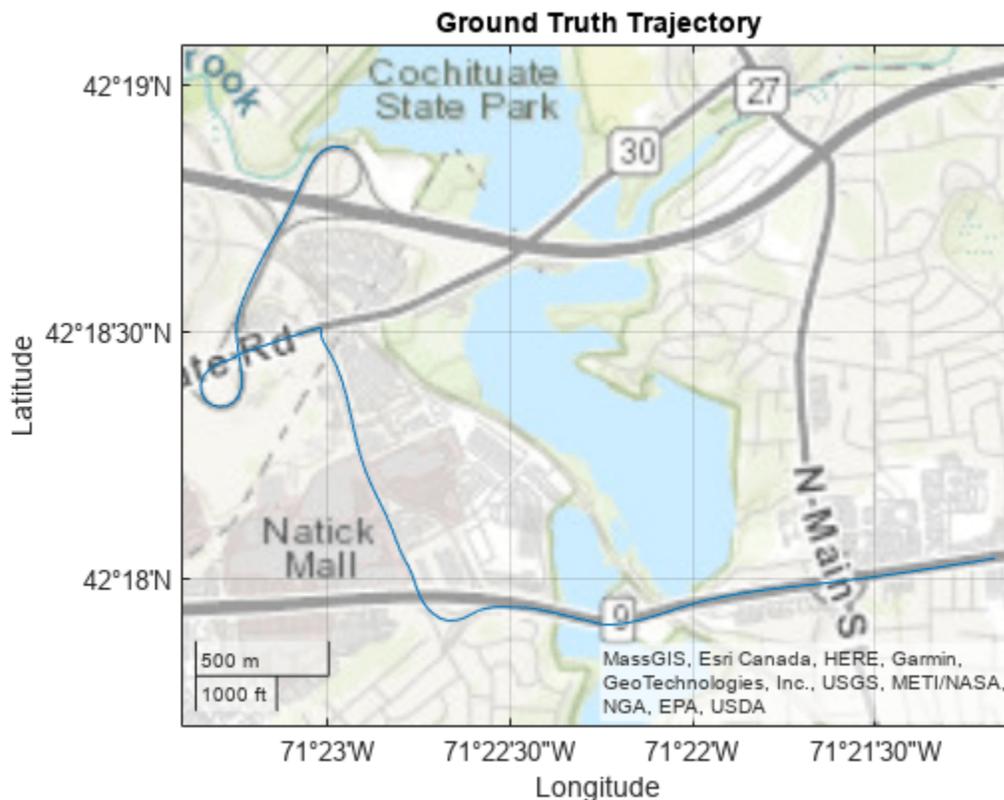
% Convert receiver position from north-east-down (NED) to geodetic
% coordinates.
receiverLLA = ned2lla(recPos,lla0,"ellipsoid");

% Specify the RINEX file.
rinexFile = "GODS00USA_R_20211750000_01D_GN.rnx";

% Set RNG seed to allow for repeatable results.
rng("default");
```

Visualize the `geoplot` for the ground truth trajectory.

```
figure
geoplot(lat,lon)
geobasemap("topographic")
title("Ground Truth Trajectory")
```



Simulate Satellite Positions Over Time

The `satelliteScenario` object enables you to specify initial orbital parameters and visualize them using the `satelliteScenarioViewer` object. This example uses the `satelliteScenario` and a RINEX file with initial orbital parameters to simulate the GPS constellations over time. Alternatively, you could use the `gnssConstellation` (Navigation Toolbox) object which simulates satellite positions using nominal orbital parameters or a RINEX file, and only the current simulation time is needed to calculate the satellite positions.

```
% Create scenario.
sc = satelliteScenario(startTime, stopTime, dt);

% Initialize satellites.
navmsg = rinexread(rinexFile);
satellite(sc,navmsg);
satID = sc.Satellites.Name;

% Preallocate results.
numSats = numel(sc.Satellites);
allSatPos = zeros(numSats,3,simulationSteps);
allSatVel = zeros(numSats,3,simulationSteps);

% Save satellite states over entire simulation.
for i = 1:numel(sc.Satellites)
    [oneSatPos, oneSatVel] = states(sc.Satellites(i),"CoordinateFrame","ecef");
    allSatPos(i,:,:)= permute(oneSatPos,[3 1 2]);
end
```

```

    allSatVel(i, :, :) = permute(oneSatVel, [3 1 2]);
end

```

Calculate Pseudoranges

Use the satellite positions to calculate the pseudoranges and satellite visibilities throughout the simulation. The mask angle is used to determine the satellites that are visible to the receiver. The pseudoranges are the distances between the satellites and the GNSS receiver. The term *pseudorange* is used because this distance value is calculated by multiplying the time difference between the current receiver clock time and the timestamped satellite signal by the speed of light.

```

% Preallocate results.
allP = zeros(numSats, simulationSteps);
allPDot = zeros(numSats, simulationSteps);
allIsSatVisible = false(numSats, simulationSteps);

% Use the skyplot to visualize satellites in view.
sp = skyplot([], [], MaskElevation=maskAngle);

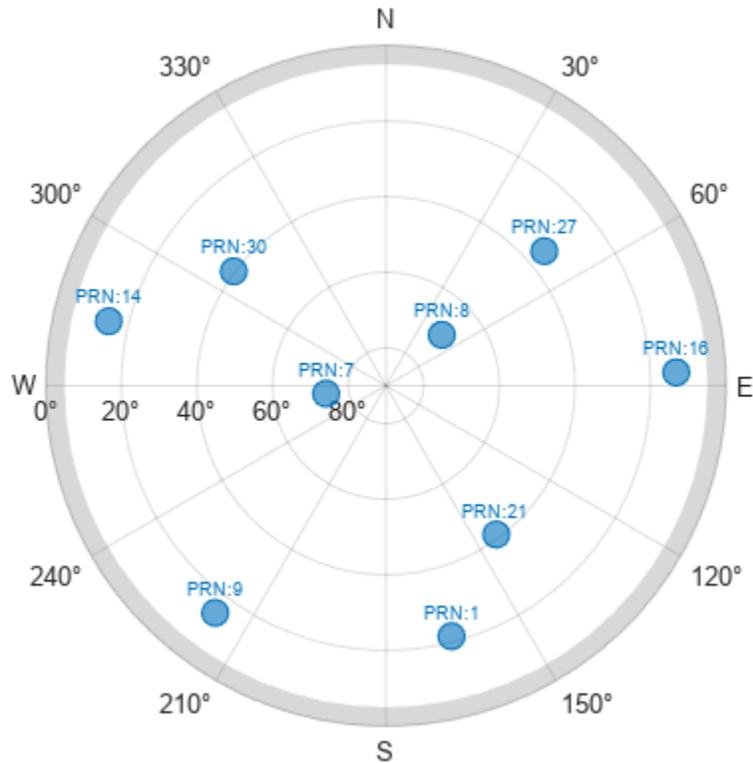
for idx = 1:simulationSteps
    satPos = allSatPos(:, :, idx);
    satVel = allSatVel(:, :, idx);

    % Calculate satellite visibilities from receiver position.
    [satAz, satEl, allIsSatVisible(:, idx)] = lookangles(receiverLLA(idx, :), satPos, maskAngle);

    % Calculate pseudoranges and pseudorange rates using satellite and
    % receiver positions and velocities.
    [allP(:, idx), allPDot(:, idx)] = pseudoranges(receiverLLA(idx, :), satPos, recVel(idx, :), satVel);

    set(sp, "AzimuthData", satAz(allIsSatVisible(:, idx)), ...
        "ElevationData", satEl(allIsSatVisible(:, idx)), ...
        "LabelData", satID(allIsSatVisible(:, idx)))
    drawnow limitrate
end

```



Estimate Receiver Position from Pseudoranges and Satellite Positions

Finally, use the satellite positions and pseudoranges to estimate the receiver position with the `receiverposition` function.

```
% Preallocate results.
lla = zeros(simulationSteps,3);
gnssVel = zeros(simulationSteps,3);
hdop = zeros(simulationSteps,1);
vdop = zeros(simulationSteps,1);

for idx = 1:simulationSteps
    p = allP(:,idx);
    pdot = allPDot(:,idx);
    isSatVisible = allIsSatVisible(:,idx);
    satPos = allSatPos(:, :, idx);
    satVel = allSatVel(:, :, idx);

    % Estimate receiver position and velocity using pseudoranges,
    % pseudorange rates, and satellite positions and velocities.
    [lla(idx,:), gnssVel(idx,:), hdop(idx,:), vdop(idx,:)] = receiverposition(p(isSatVisible), ...
        satPos(isSatVisible,:), pdot(isSatVisible), satVel(isSatVisible,:));
end
```

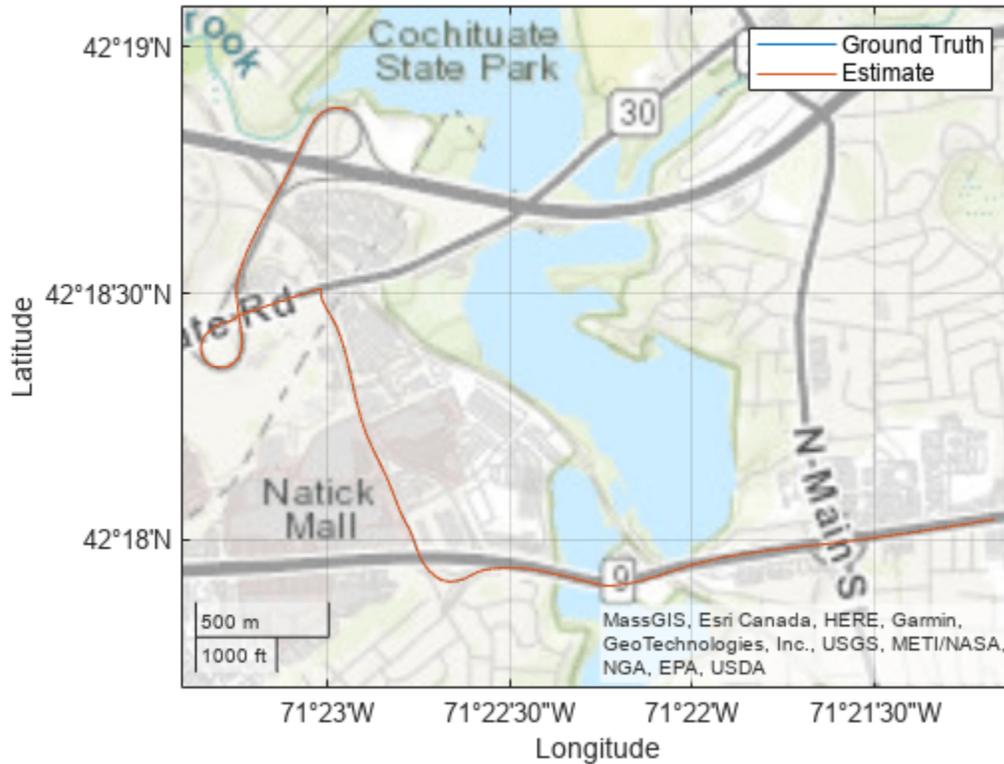
Visualize Results

Plot the ground truth position and the estimated receiver position on a `geoplot`.

```

figure
geoplot(lat,lon,lla(:,1),lla(:,2))
geobasemap("topographic")
legend("Ground Truth","Estimate")

```

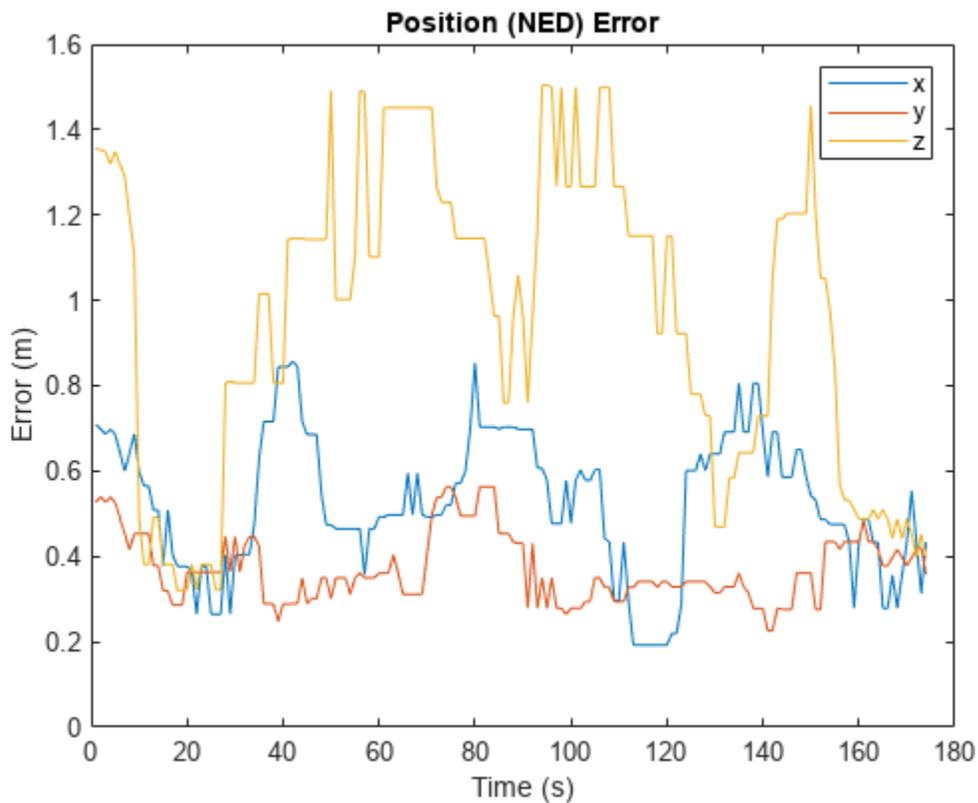


Plot the absolute error in the position estimate. The error is smoothed by a moving median to make the plot more readable. The error in the x- and y-axis is smaller because there are satellites on either side of the receiver. The error in the z-axis is larger because there are only satellites above the receiver, not below it. The error changes over time as the receiver moves and some satellites come in and out of view.

```

estPos = lla2ned(lla,lla0,"ellipsoid");
winSize = floor(size(estPos,1)/10);
figure
plot(smoothdata(abs(estPos-pos),"movmedian",winSize))
legend("x","y","z")
xlabel("Time (s)")
ylabel("Error (m)")
title("Position (NED) Error")

```



See Also

Objects

[satelliteScenario](#) | [satellite](#) | [access](#) | [groundStation](#) | [satelliteScenarioViewer](#) | [conicalSensor](#) | [transmitter](#) | [receiver](#)

Functions

[show](#) | [play](#) | [hide](#)

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to Ground Station” on page 1-14
- “Comparison of Orbit Propagators” on page 1-25
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Calculate Latency and Doppler in a Satellite Scenario

This example shows how to model a GPS satellite constellation from a SEM almanac, analyze access between the satellites and a ground station, and calculate the latency and the doppler frequency between the satellites and the ground station.

Create a Satellite Scenario

Create a satellite scenario with a start time of 11-January-2020 2:50:00 PM UTC and a stop time 3 days later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,1,11,14,50,0);
stopTime = startTime + days(3);
sampleTime = 60; % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a Satellite to the Scenario

Add a satellite to the scenario from the `gpsAlmanac.txt` SEM almanac file.

```
sat = satellite(sc,"gpsAlmanac.txt");
```

The default orbit propagator when creating satellites using SEM almanac is `gps`. This can be verified by observing the `OrbitPropagator` property. Additionally, the output of the `orbitalElements` function contains GPS satellite-specific information.

```
orbitProp = sat(1).OrbitPropagator

orbitProp =
'gps'

elements = orbitalElements(sat(1))

elements = struct with fields:
    PRN: 1
    GPSWeekNumber: 2087
    GPSTimeOfApplicability: 589824
    SatelliteHealth: 0
    SemiMajorAxis: 2.655951847442722e+07
    Eccentricity: 0.009273052215576
    Inclination: 56.066459655761712
    GeographicLongitudeOfOrbitalPlane: -40.477838516235359
    RateOfRightAscension: -4.616595106199388e-07
    ArgumentOfPerigee: 43.383057117462180
    MeanAnomaly: 1.726436662673951e+02
    Period: 4.307658596414280e+04
```

Add a Ground Station

Add the Madrid Deep Space Communications Complex as the ground station of interest, and specify its latitude and longitude.

```
name = "Madrid Deep Space Communications Complex";
lat = 40.43139; % degrees
lon = -4.24806; % degrees
gs = groundStation(sc,Name=name,Latitude=lat,Longitude=lon);
```

Add an Access Analysis

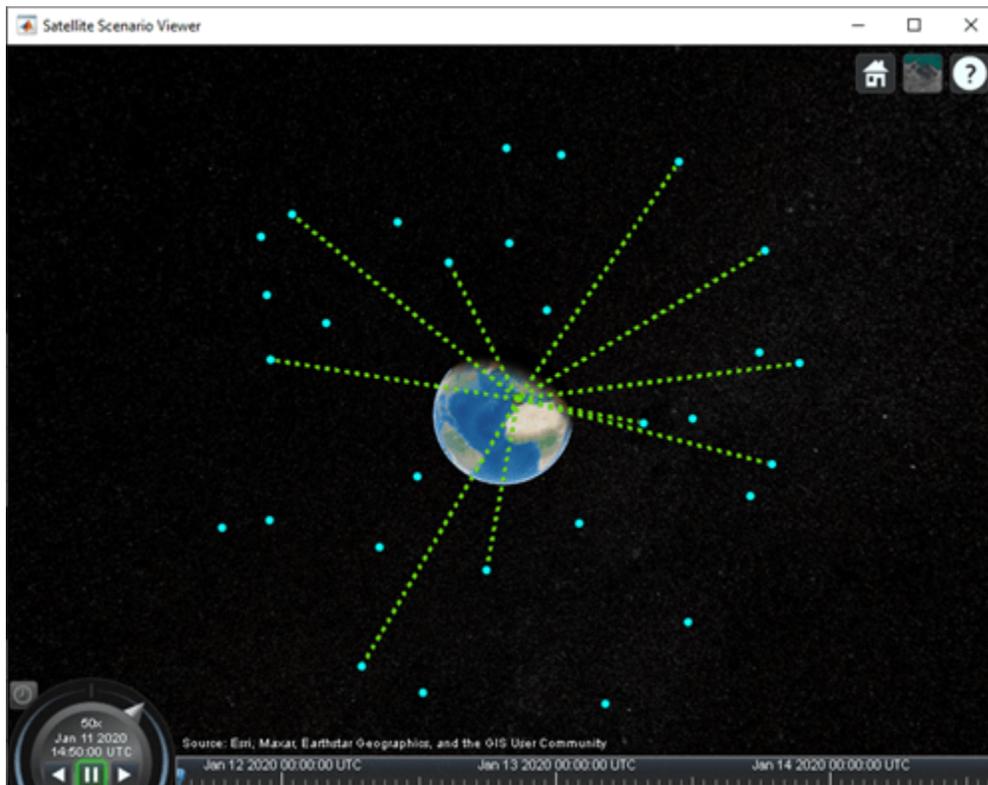
Add access analysis between the GPS satellites and Madrid ground station, which determines when the ground station is visible to the satellites. This determines when latency and doppler should be calculated.

```
ac = access(sat,gs);  
[acStatus,time] = accessStatus(ac);
```

Visualize Scenario

Visualize the scenario by launching a satellite scenario viewer. Set `ShowDetails` of the viewer to false to ensure the visualization is not cluttered. Set the camera position to bring all satellites that have access to the ground station into view.

```
v = satelliteScenarioViewer(sc,ShowDetails=false);  
campos(v, ...  
    29, ... % Latitude, degrees  
    -19, ... % Longitude, degrees  
    7.3e7); % Altitude, degrees
```



Calculate Latency and Velocity

Calculate the latency between the GPS satellites and the Madrid ground station. Also, calculate the velocity along the line between each satellite and the ground station. A positive value indicates that the satellite is moving towards the ground station, and a negative value indicates the satellite is moving away from the ground station.

```
% Calculate azimuth, elevation, and range from each satellite to the ground  
% station.
```

```

[az,el,r] = aer(sat,gs);

% Calculate latency.
c = physconst("Lightspeed");
latency = r/c;

% Calculate the velocity of each satellite in its respective
% North/East/Down (NED) frame. Physically, this is the ECEF velocity,
% represented in NED frame of the satellite. Therefore, the relative
% velocity with respect to the ground station is also the same.
[~,satV] = states(sat,CoordinateFrame="geographic");

% Calculate the direction of the ground station with respect to each
% satellite in its respective NED frame. satV is a 3-dimensional array,
% wherein the first dimension represents the cartesian component, second
% dimension represents the time sample, and third dimension represents the
% satellite. Therefore, compute the direction also as a 3-dimensional
% array, wherein the first dimension has a size of 1, the second dimension
% represents the time sample, and the third dimension corresponds to the
% satellite. To do this, az and el must also be re-formatted such that the
% first dimension, which represents the satellite becomes the third
% dimension. You can use permute to re-format the arrays as described.
dir = cat(1,permute(cosd(el).*cosd(az),[3 2 1]), ...
    permute(cosd(el).*sind(az),[3 2 1]), ...
    permute(-sind(el),[3 2 1]));

% Calculate the velocity along the line between the ground station and each
% satellite. This velocity determines the doppler frequency corresponding
% to each satellite. Re-format this velocity such that the first dimension
% corresponds to the satellite.
dopV = permute(dot(satV,dir),[3 2 1]);

% Set the latency and doppler velocity to NaN whenever access status is
% false because these quantities are irrelevant when there is no access.
latency(~acStatus) = NaN;
dopV(~acStatus) = NaN;

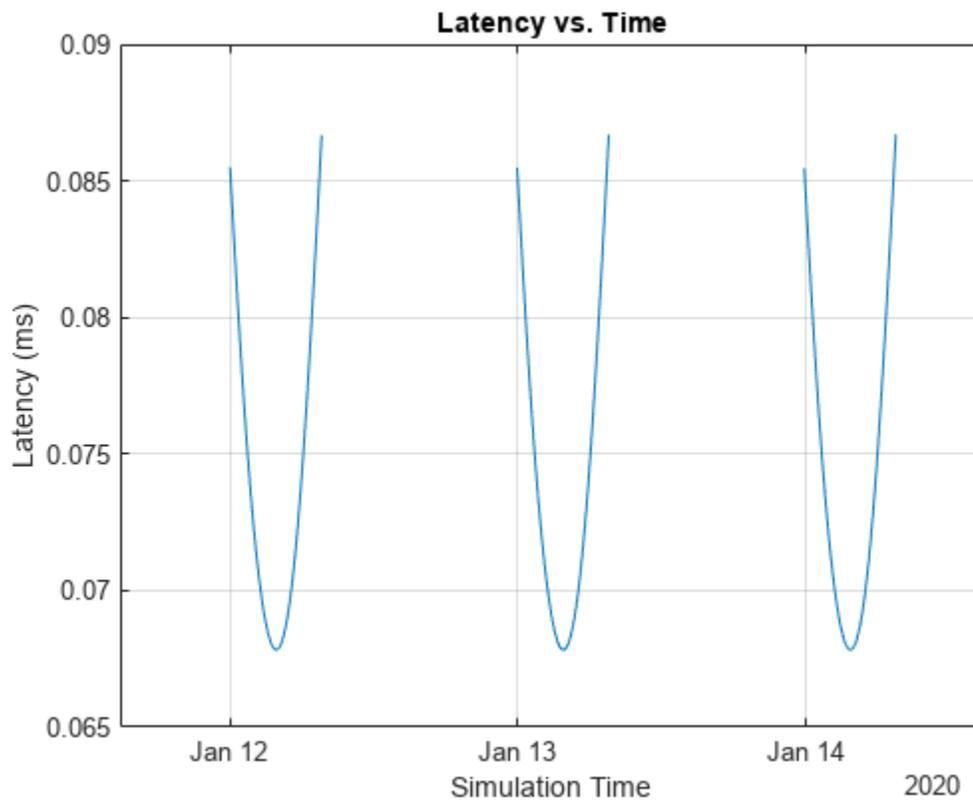
```

Plot the latency corresponding to the first satellite. You may choose to plot the latency corresponding to all satellites. However, this example plots it only for the first satellite to serve as a demonstration and to reduce plot clutter.

```

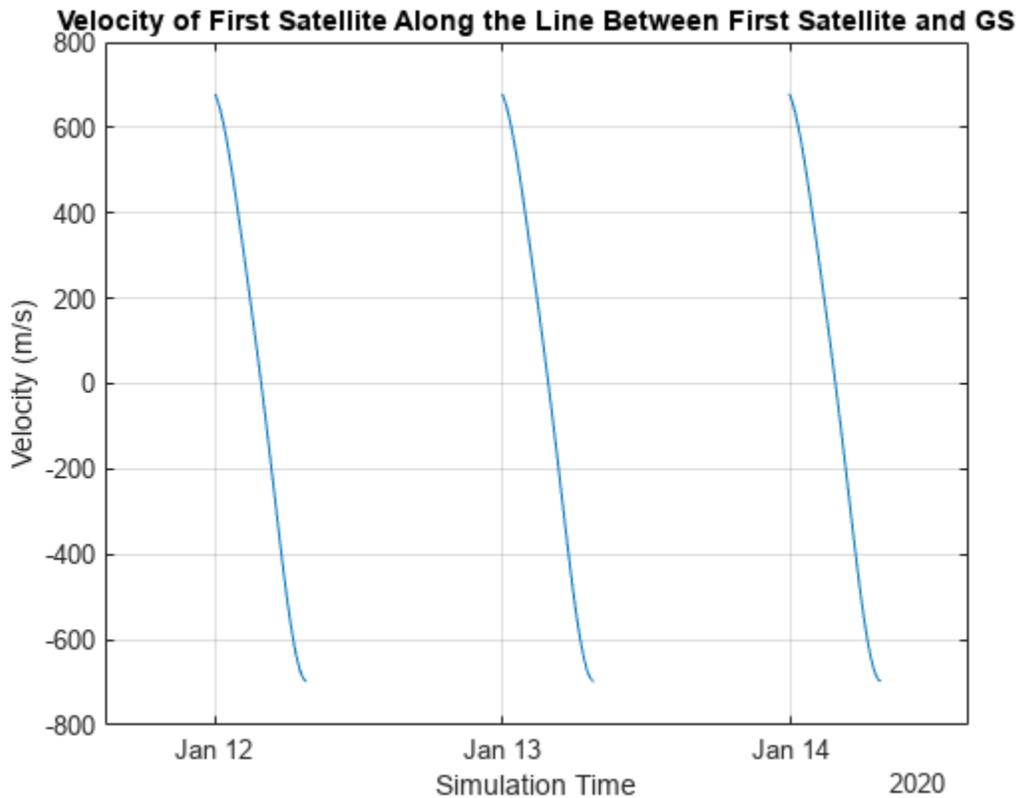
plot(time,latency(1,:))
xlim([sc.StartTime sc.StopTime])
title("Latency vs. Time")
xlabel("Simulation Time")
ylabel("Latency (ms)")
grid on

```



Plot the velocity of the first satellite along the line between itself and the ground station.

```
plot(time,dopV(1,:))
xlim([sc.StartTime sc.StopTime])
title("Velocity of First Satellite Along the Line Between First Satellite and GS")
xlabel("Simulation Time")
ylabel("Velocity (m/s)")
grid on
```



Calculate Doppler Frequency from Velocity

The doppler frequency is calculated from the following equation:

$$f_o = \left(\frac{c \pm v_r}{c \pm v_s} \right) f_e,$$

where c is the speed of light in m/s,

v_r is the speed, in m/s, of the receiver relative to the medium, added to c if the receiver is moving towards the source, subtracted if the receiver is moving away from the source,

v_s is the speed, in m/s, of the source relative to the medium, added to c if the source is moving away from the receiver, subtracted if the source is moving towards the receiver,

f_e is the emitted frequency, in Hz, and

f_o is the observed frequency, in Hz.

In this example, we consider the observed frequency from the standpoint of a receiving ground station. In that case, v_r is 0. We also consider a C band frequency of 5 GHz.

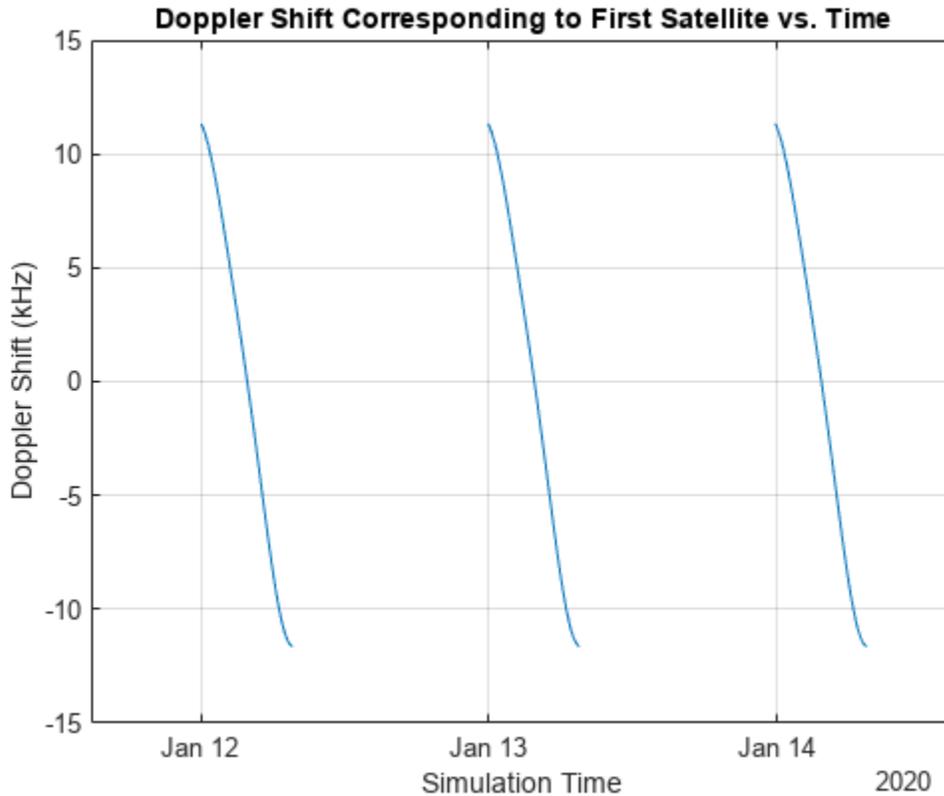
```
fe = 5e9; % emitted frequency in Hz
fShift = (((c ./ (c-dopV)) * fe) - fe); % doppler shift in Hz
```

Plot the doppler shift corresponding to the first satellite.

```

plot(time,fShift(1,+)/1e3) % plot kHz
xlim([sc.StartTime sc.StopTime])
title("Doppler Shift Corresponding to First Satellite vs. Time")
xlabel("Simulation Time")
ylabel("Doppler Shift (kHz)")
grid on

```



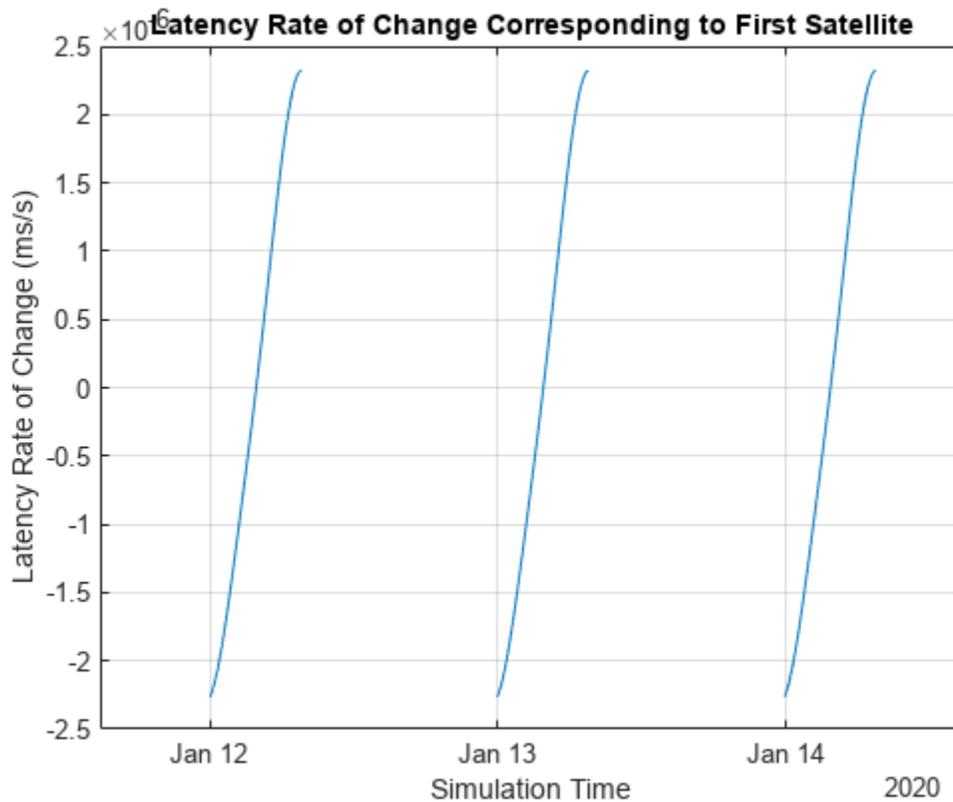
Calculate Rates of Change for Latency and Doppler

Satellite communications links need to be designed to track varying latencies and Doppler frequencies. Thus, it is important to calculate these quantities. Plot these quantities corresponding to the first satellite.

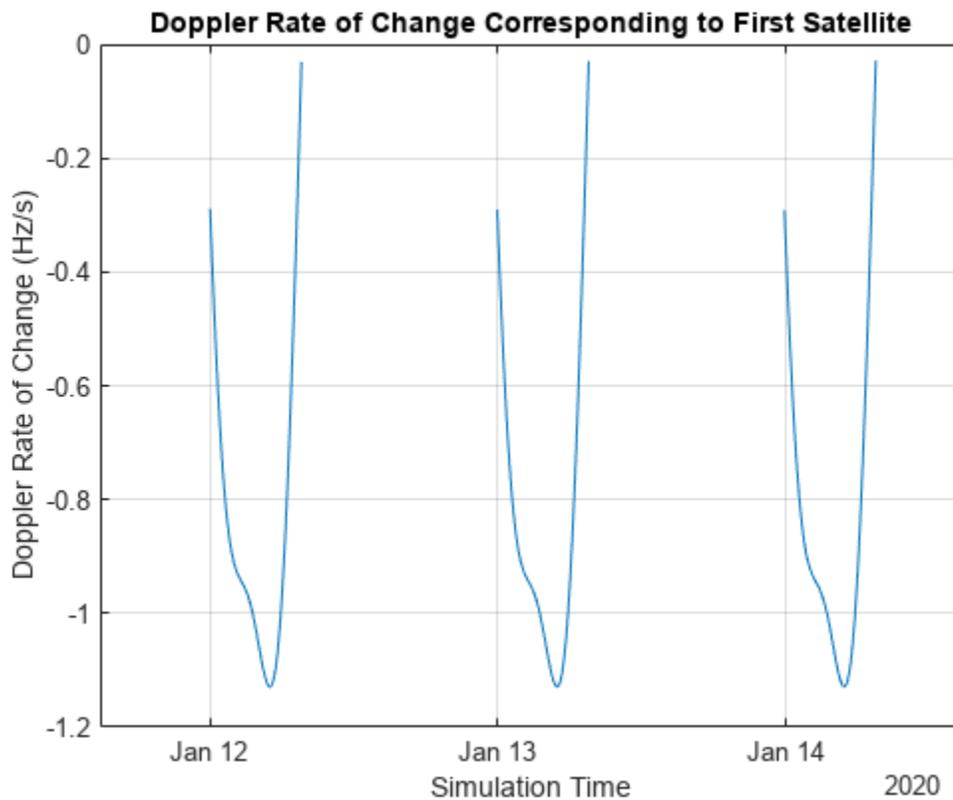
```

latencyRate = diff(latency,1,2)/sampleTime;
plot(time(1:end-1),latencyRate(1,:)) % seconds/second
xlim([sc.StartTime time(end-1)])
title("Latency Rate of Change Corresponding to First Satellite")
xlabel("Simulation Time")
ylabel("Latency Rate of Change (ms/s)")
grid on

```

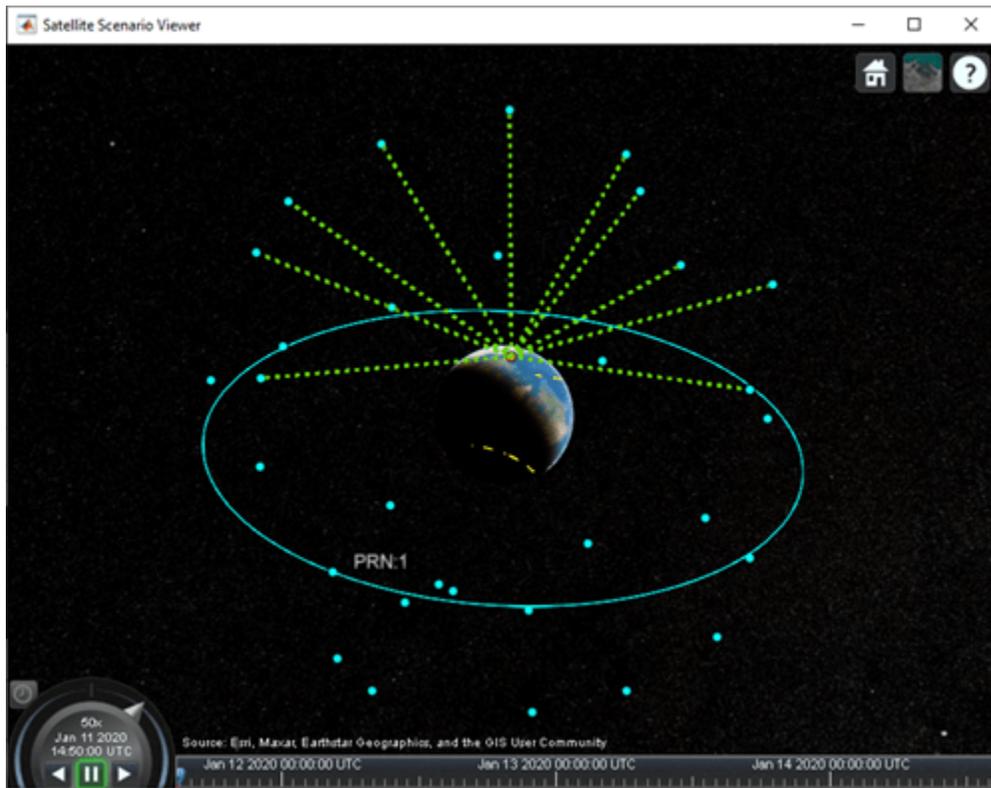


```
dopplerRate = (diff(fShift,1,2)/sampleTime);
plot(time(1:end-1),dopplerRate(1,:))
xlim([sc.StartTime time(end-1)])
title("Doppler Rate of Change Corresponding to First Satellite")
xlabel("Simulation Time")
ylabel("Doppler Rate of Change (Hz/s)")
grid on
```



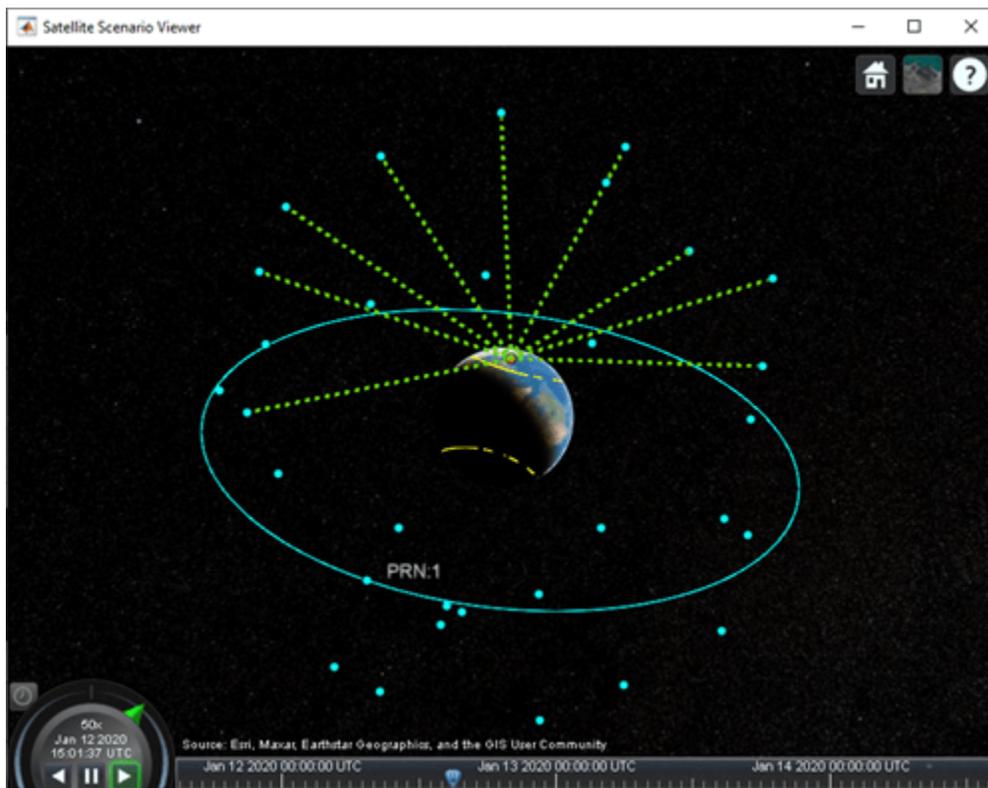
Show the name and orbit of the first satellite and plot its future ground track, or lead time, over 12 hours. Dashed yellow shows the future ground track, and solid yellow shows the past ground track.

```
sat(1).ShowLabel = true;
show(sat(1).Orbit);
groundTrack(sat(1), ...
    LeadTime=12*3600); % seconds
campos(v,84, -176,7.3e7);
```



Play the scenario with the satellites and ground station.

`play(sc)`



See Also

Objects

[satelliteScenario](#) | [satellite](#) | [groundTrack](#) | [groundStation](#) | [access](#)

Functions

[orbitalElements](#) | [aer](#) | [play](#)

Related Examples

- “Satellite Constellation Access to Ground Station” on page 1-14
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Interference from Satellite Constellation on Communications Link

This example shows how to analyze interference on a downlink from a constellation of satellites in a medium-Earth orbit (MEO) to a ground station located in the Pacific Ocean. The interfering constellation consists of 40 satellites in a low-Earth orbit (LEO). This example determines the times at which the downlink is closed, the carrier to noise plus interference ratio, and the link margin.

This example requires Satellite Communications Toolbox. If you also have Antenna Toolbox™, you can use this example to learn how to import antennas from Antenna Toolbox into satellite scenario. If you also have Phased Array System Toolbox™, you can use this example to learn how to import antennas from Phased Array System Toolbox into satellite scenario and use beamforming to improve the carrier-to-noise-and-interference ratio (CNIR).

Create Satellite Scenario

Create a satellite scenario. Define the start time and stop time of the scenario. Set the sample time to 60 seconds.

```
startTime = datetime(2021,3,17,22,52,0);           % 17 March 2021 10:52 PM UTC
stopTime = startTime + minutes(10);               % 17 March 2021 11:02 PM UTC
sampleTime = 60;                                  % In s
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add Medium-Earth Orbit Satellite

Add a satellite in an MEO by specifying its Keplerian orbital elements. This satellite is the satellite from which data is downlinked.

```
semiMajorAxis = 12000000;                          % In m
eccentricity = 0;
inclination = 8;                                    % In degrees
raan = 0;                                           % Right ascension of ascending node, in degrees
argOfPeriapsis = 0;                                 % In degrees
trueAnomaly = 343.9391;                             % In degrees
meoSat = satellite(sc, semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    raan, ...
    argOfPeriapsis, ...
    trueAnomaly, ...
    Name = "MEO Satellite", ...
    OrbitPropagator = "two-body-keplerian");
```

Add Interfering Satellite Constellation

Add the interfering satellite constellation from a two-line-element (TLE) file. These satellites are placed in LEO.

```
interferingSat = satellite(sc,"leoSatelliteConstellation.tle");
```

Add Transmitter to MEO Satellite

Add a transmitter to the MEO satellite. This transmitter is used for the downlink. Define the antenna specifications and set the operating carrier frequency to 3 GHz.

```
txMEOFreq = 3e9; % In Hz
txMEOSat = transmitter(meoSat, ...
    Frequency = txMEOFreq, ... % In Hz
    Power = 11); % In dBW
gaussianAntenna(txMEOSat, ...
    DishDiameter = 1); % In m
```

Add Transmitter to LEO Satellites

Add a transmitter to each satellite in the LEO constellation, and then define the antenna specifications. These transmitters are the ones that interfere with the downlink from the MEO satellite. Set the operating carrier frequency of the interfering satellites to 2.99 GHz. The example assigns each interfering satellite a random power in the range from 10 to 20 dBW.

```
interferenceFreq = 2.99e9; % In Hz
rng("default");
txInterferingSat = transmitter(interferingSat, ...
    Frequency = interferenceFreq, ... % In Hz
    Power = 10+10*rand(1,numel(interferingSat))); % In dBW
gaussianAntenna(txInterferingSat, ...
    DishDiameter = 0.2); % In m
```

Add Ground Station

Add a ground station to the satellite scenario by specifying its latitude and longitude.

```
gs = groundStation(sc, ...
    0, ... % Latitude in degrees
    180, ... % Longitude in degrees
    Name = "Ground station");
```

Specify Ground Station Antenna Type

For this example, you can choose from one of the following antennas:

- Gaussian Antenna
- Parabolic Reflector from Antenna Toolbox
- Uniform Rectangular Array from Phased Array System Toolbox

% Select the desired ground station antenna.

```
groundStationAntennaType =  ;
```

Add Receiver to Ground Station

Add a receiver to the ground station. If you selected Gaussian Antenna or Parabolic Reflector, attach the receiver to a gimbal, which is in turn attached to the ground station. Configure the gimbal to track the MEO satellite, so that the antenna also tracks the MEO satellite. If you selected Uniform Rectangular Array, attach the receiver directly to the ground station. Specify the mounting location and mounting angles of the gimbal and receiver, and the antenna specifications appropriately.

```
switch groundStationAntennaType
case {"Gaussian Antenna", "Parabolic Reflector"}
    % When Gaussian Antenna or Parabolic Reflector is selected, attach
    % a gimbal to the ground station.
    gim = gimbal(gs, ...
        MountingLocation = [0;0;-5], ... % In m
```

```

        MountingAngles = [0;180;0]);    % In degrees

% Set the gimbal to track the MEO satellite.
pointAt(gim,meoSat);

if groundStationAntennaType == "Gaussian Antenna"
    % When Gaussian Antenna is selected

    % Create the receiver object and add it to the gimbal
    rxGs = receiver(gim, ...
        MountingLocation = [0;0;1]); % In m

    % Provide the Gaussian Antenna specifications
    gaussianAntenna(rxGs, ...
        DishDiameter = 0.8);    % In m
else
    % When Parabolic Reflector is selected

    % Size the antenna based on the frequency
    % Requires Antenna Toolbox(TM)
    ant = design(reflectorParabolic,txMEOFreq);

    % Create the receiver object and add it to the gimbal
    rxGs = receiver(gim, ...
        Antenna = ant, ...
        MountingLocation = [0;0;1]); % In m
end
case "Uniform Rectangular Array"
    % When Uniform Rectangular Array is selected

    % Determine the wavelength of the downlink signal
    c = physconst('LightSpeed');
    lambda = c/txMEOFreq;

    % Define array size
    nrow = 8;
    ncol = 8;

    % Define element spacing
    drow = lambda/2;
    dcol = lambda/2;

    % Create a back-baffled 6-by-6 antenna array
    % Requires Phased Array System Toolbox(TM)
    ant = phased.URA(Size = [nrow ncol], ...
        ElementSpacing = [drow dcol]);
    ant.Element.BackBaffled = true;

    % Create the receiver object and add it to the ground station
    rxGs = receiver(gs, ...
        Antenna = ant, ...
        MountingAngles = [0;90;0]); % In degrees
end

```

Create Access Analysis Between Interfering Satellite Constellation and Ground Station

Add an access analysis between each satellite in the interfering constellation and the ground station. This analysis enables the visualization of interference in the satellite scenario viewer that will be

launched later. Any time a satellite in the constellation is visible to the ground station, there is some level of interference from that visible satellite.

```
ac = access(interferingSat,gs);  
ac.LineColor = [1 1 0];           % Yellow
```

Set Tracking Targets for Satellites

Set the satellites to track the ground station. This ensures that the transmitter antennas on board each satellite track the ground station. Setting the interfering satellite transmitters to track the ground station results in the worst-case interference on the downlink.

```
pointAt([meoSat interferingSat],gs);
```

Calculate Weights of Uniform Rectangular Array

If you selected Uniform Rectangular Array as the ground station antenna, compute the weights that are required to point the main lobe towards the MEO satellite, and the nulls towards the interfering satellites, thereby cancelling the interference. Assign the computed weights using `pointAt`.

```
if groundStationAntennaType == "Uniform Rectangular Array"  
    % Find the LEO satellites that are in the line of sight of the ground  
    % station. These satellites are the potential interferers.  
    currentInterferingSat = interferingSat(accessStatus(ac,sc.StartTime) == true);  
  
    % Calculate the direction of the MEO satellite with respect to the  
    % array. This is the lookout direction.  
    [azd,eld] = aer(rxGs,meoSat,sc.StartTime,CoordinateFrame='body');  
  
    % Calculate the directions of the potentially interfering satellites  
    % with respect to the array. These are the null directions.  
    [azn,eln] = aer(rxGs,currentInterferingSat,sc.StartTime,CoordinateFrame='body');  
  
    % Calculate the steering vectors for the lookout direction.  
    % Requires Phased Array System Toolbox.  
    wd = steervec(getElementPosition(ant)/lambda,[wrapTo180(azd);-eld]);  
  
    % Calculate the steering vector for null directions.  
    % Requires Phased Array System Toolbox.  
    wn = steervec(getElementPosition(ant)/lambda,[wrapTo180(azn)';-eln']);  
  
    % Compute the response of the desired steering at null directions.  
    rn = (wn'*wn)\(wn'*wd);  
  
    % Sidelobe canceler - remove the response at null directions.  
    w = wd-wn*rn;  
  
    % Assign the weights to the phased array.  
    pointAt(rxGs,Weights=w);  
end
```

Create Desired Downlink

Create a downlink from the transmitter on board the MEO satellite to the receiver on board the ground station. This link is the downlink which encounters interference from the LEO constellation.

```
downlink = link(txMEOSat,rxGs);
```

Create Interfering Links

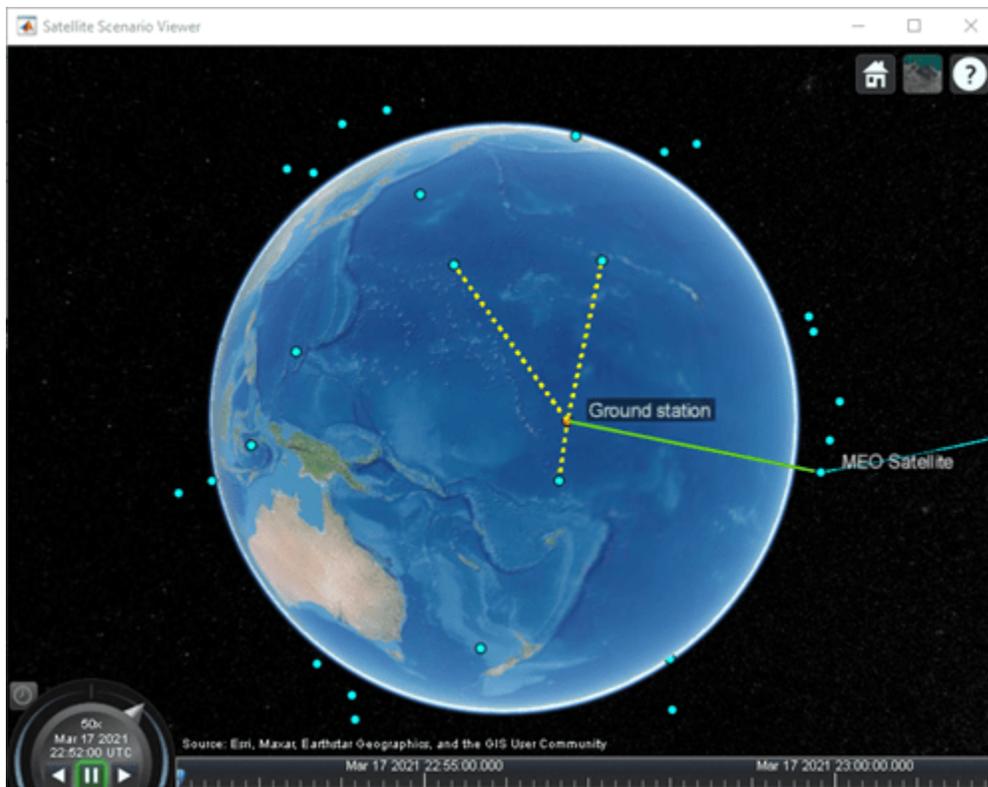
Create a link between the transmitter on board each satellite in the LEO constellation and the receiver on board the ground station. These links are the interferer links with the desired downlink.

```
lnkInterference = link(txInterferingSat,rxGs);
```

Launch Satellite Scenario Viewer

Launch the Satellite Scenario Viewer with `ShowDetails` set to `false`. When the `ShowDetails` property is set to `false`, only the satellites, the ground station, accesses, and links will be shown. The labels and orbits will be hidden. Mouse over the satellites and the ground stations to show their labels. Click on the MEO satellite so that its orbit projected up to the scenario `StopTime` and its label are visible without mousing over. Click on the ground station so that its label is visible without mousing over. The presence of the green line between the transmitter on board the MEO satellite and the receiver on board the ground station signifies that the downlink can be closed successfully assuming no interference from the satellite constellation exists. The presence of yellow lines between a given satellite in the constellation and the ground station signifies that they have access to one another, and as a result, interference from that satellite exists.

```
v = satelliteScenarioViewer(sc,ShowDetails=false);
```



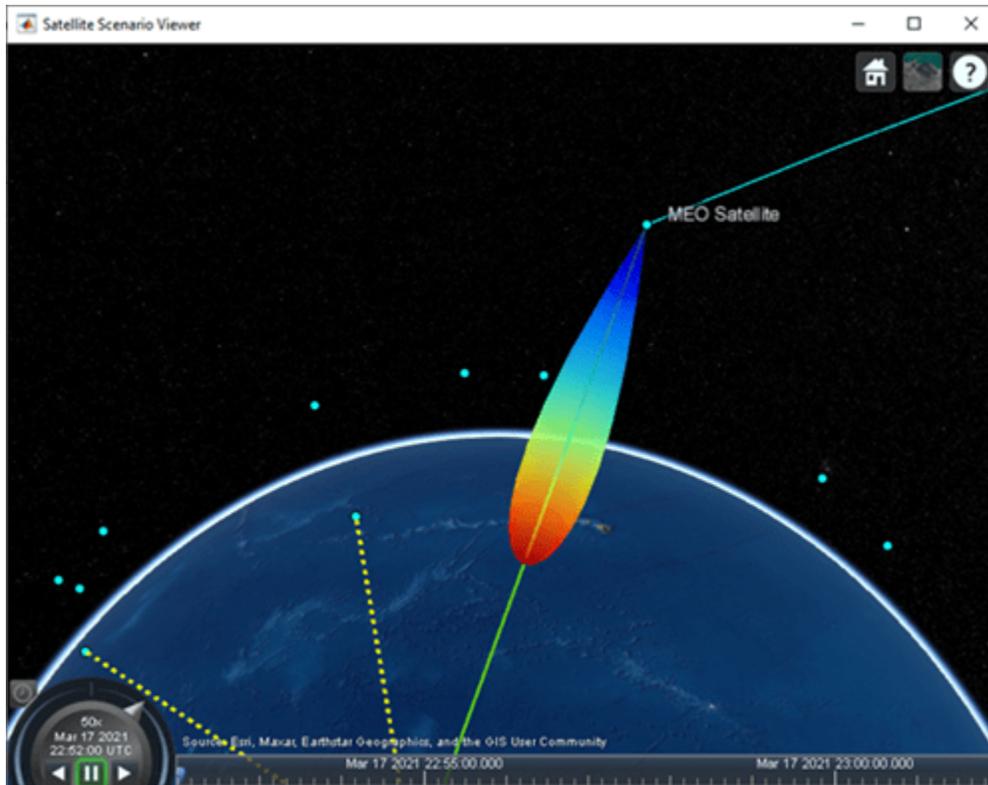
Visualize Radiation Pattern of Antennas Involved in Downlink

Visualize the radiation pattern of the transmitter antenna on board the MEO satellite and the receiver on board the ground station.

```
pattern(txMEOSat, ...  
    Size = 1000000); % In m
```

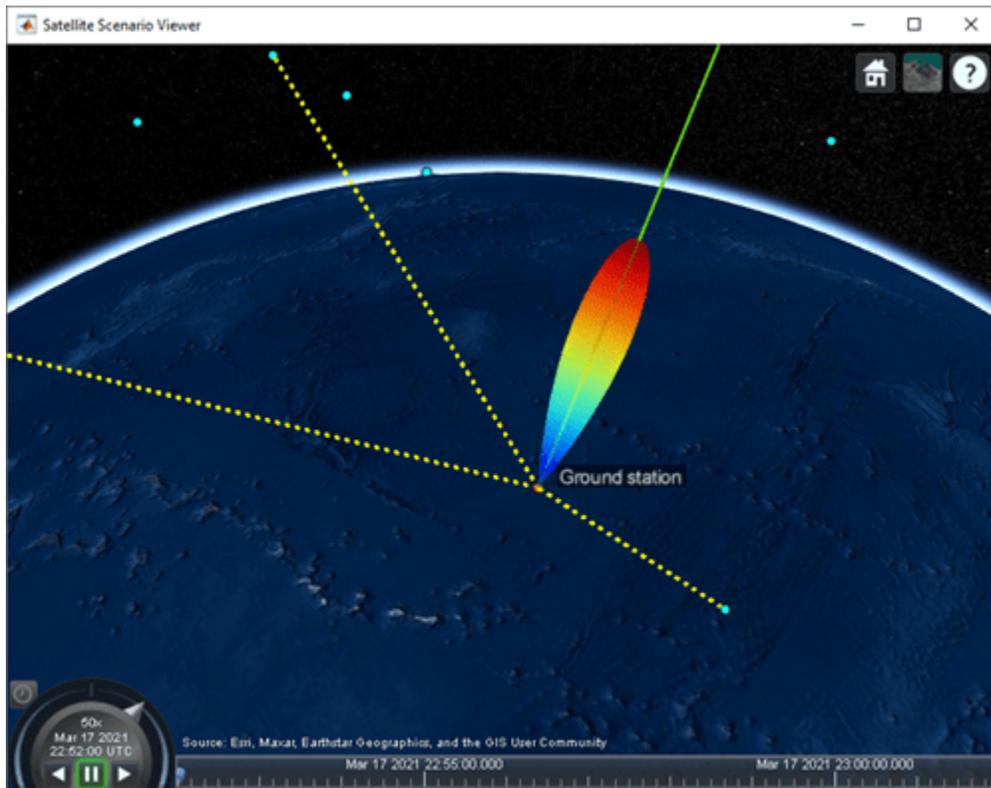
```
pattern(rxGs,txME0Freq, ...  
    Size = 1000000); % In m
```

Radiation Pattern of MEO Satellite Antenna

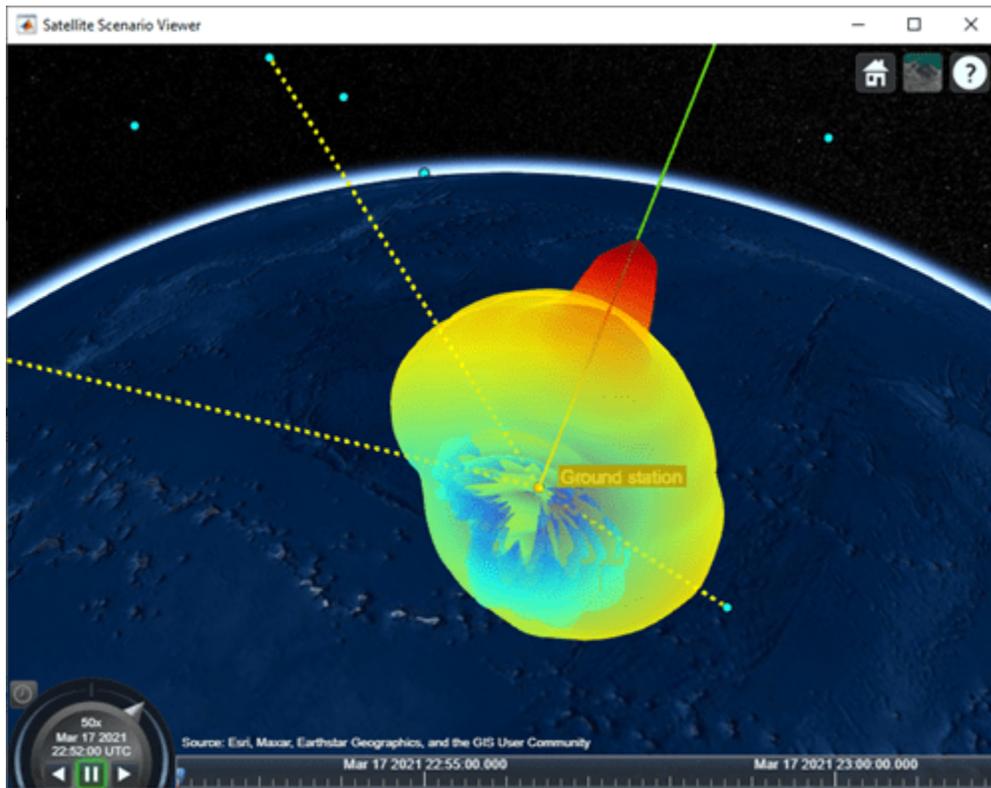


Set Camera to View Ground Station Antenna Radiation Pattern

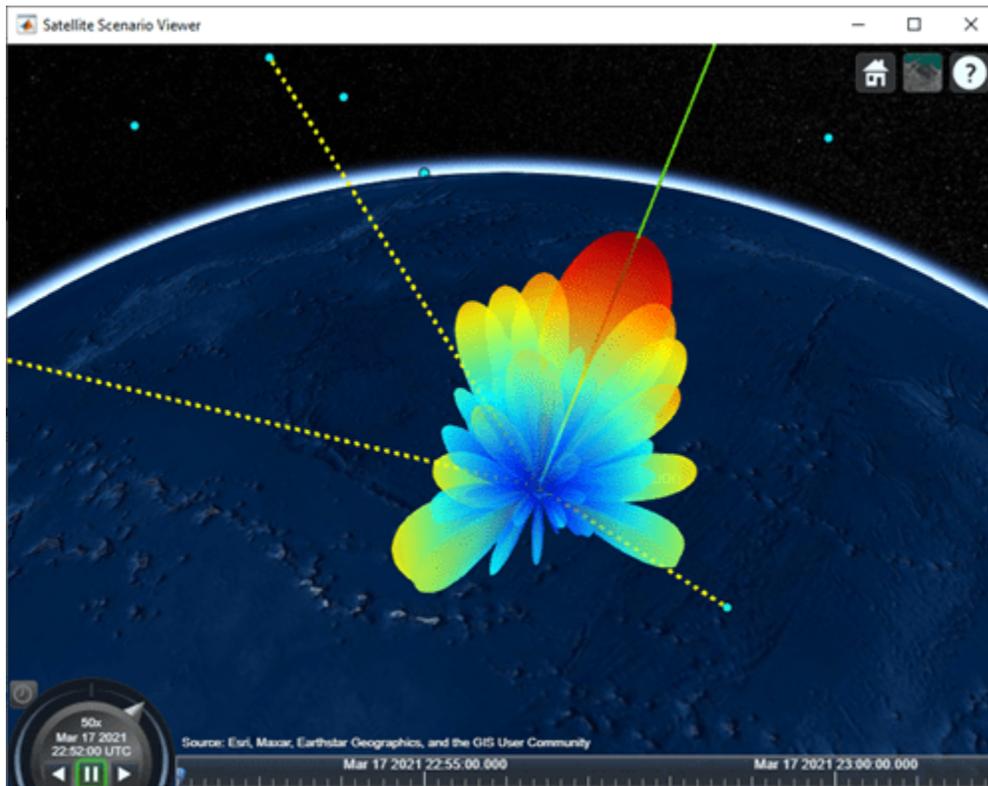
```
% Set camera position and orientation to view the ground station antenna  
% radiation pattern.  
campos(v,-8,172,2500000);  
camheading(v,40);  
campitch(v,-60);
```



Gaussian Antenna



Parabolic Reflector



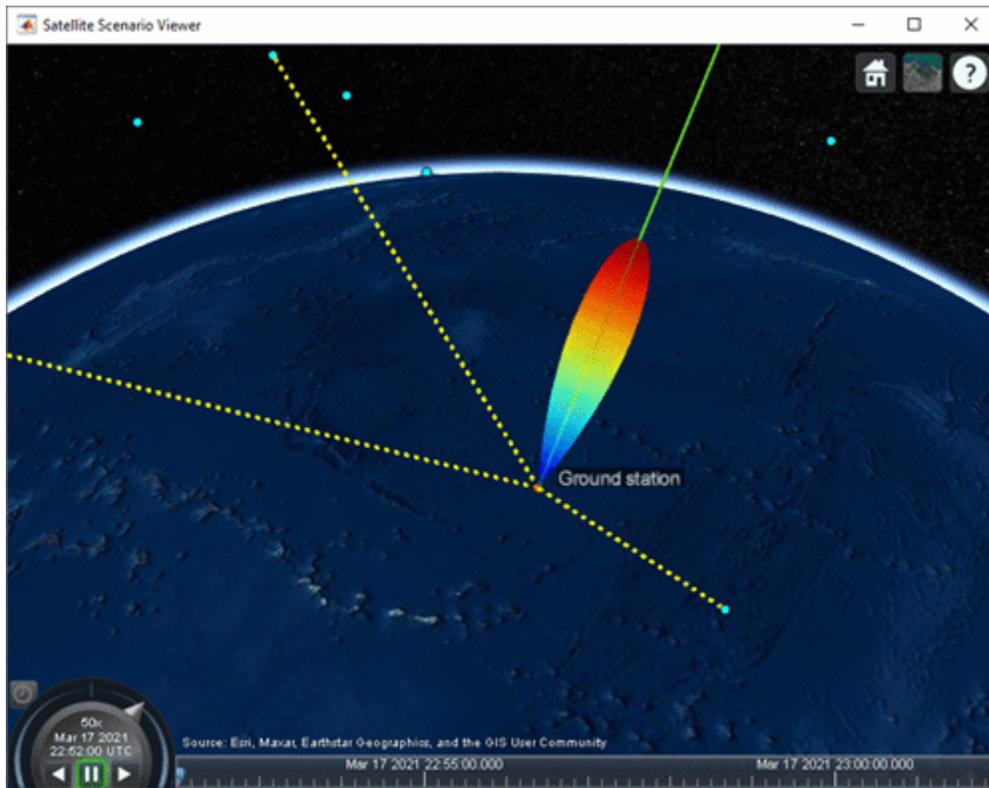
Uniform Rectangular Array

Simulate Scenario and Visualize

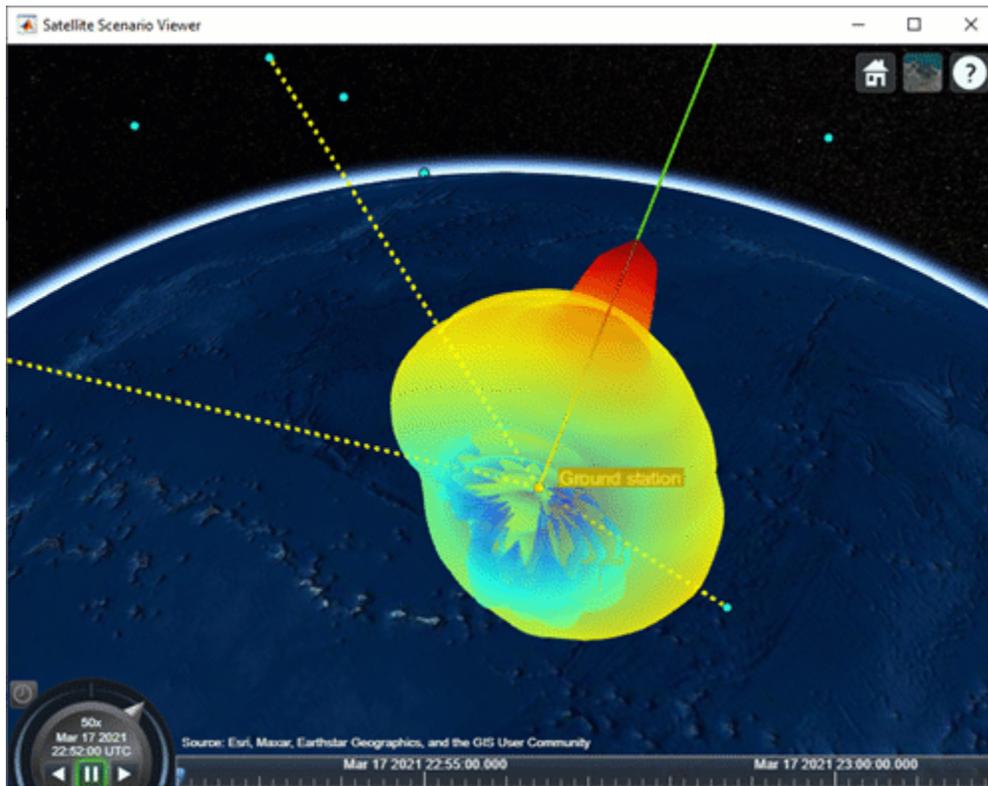
With Gaussian Antenna or Parabolic Reflector

If you selected Gaussian Antenna or Parabolic Reflector (requires Antenna Toolbox), use `play` to visualize the scenario from `StartTime` to `StopTime`. This will automatically simulate the scenario before playing back the visualization. Note how the antenna pointing changes as the gimbal tracks the MEO satellite.

```
if groundStationAntennaType == "Gaussian Antenna" || groundStationAntennaType == "Parabolic Reflector"
    play(sc);
    campos(v, -8, 172, 2500000);
    camheading(v, 40);
    campitch(v, -60);
end
```



With Gaussian Antenna



With Parabolic Reflector

With Uniform Rectangular Array

If you selected Uniform Rectangular Array (requires Phased Array System Toolbox), you must manually step through the simulation so that you can recompute the weights at each time step based on the new position of the MEO satellite and the interfering LEO satellites. To manually step through the simulation, first set `AutoSimulate` to false. Following this, you can call `advance` to move the simulation by one time step. The first call to `advance` will compute the simulation states at `StartTime`. Subsequent calls will advance the time step by one `SampleTime` and compute the states accordingly.

```
if groundStationAntennaType == "Uniform Rectangular Array"
    % Set AutoSimulate to false.
    sc.AutoSimulate = false;

    % Manually step through the simulation.
    while advance(sc)
        % Determine the access status history for each LEO satellite
        % corresponding to the current SimulationTime.
        acStatusHistory = accessStatus(ac);
        acStatus = acStatusHistory(:,end);

        % Determine the LEO satellites that are visible to the ground
        % station. These are the satellites that will potentially
        % interfere with the ground station at the current simulation
        % time.
        currentInterferingSat = interferingSat(acStatus == true);
```

```
% Determine the direction of the MEO satellite in the body frame of
% the Uniform Rectangular Array. This is the lookout direction of
% the array.
[azdHistory,eldHistory] = aer(rxGs,meoSat,CoordinateFrame='body');
azd = azdHistory(:,end);
eld = eldHistory(:,end);

% Determine the direction of these interfering satellites in
% the body frame of the Uniform Rectangular Array. These are
% the directions in which the array must point a null.
[aznHistory,elnHistory] = aer(rxGs,currentInterferingSat,CoordinateFrame='body');
azn = aznHistory(:,end);
eln = elnHistory(:,end);

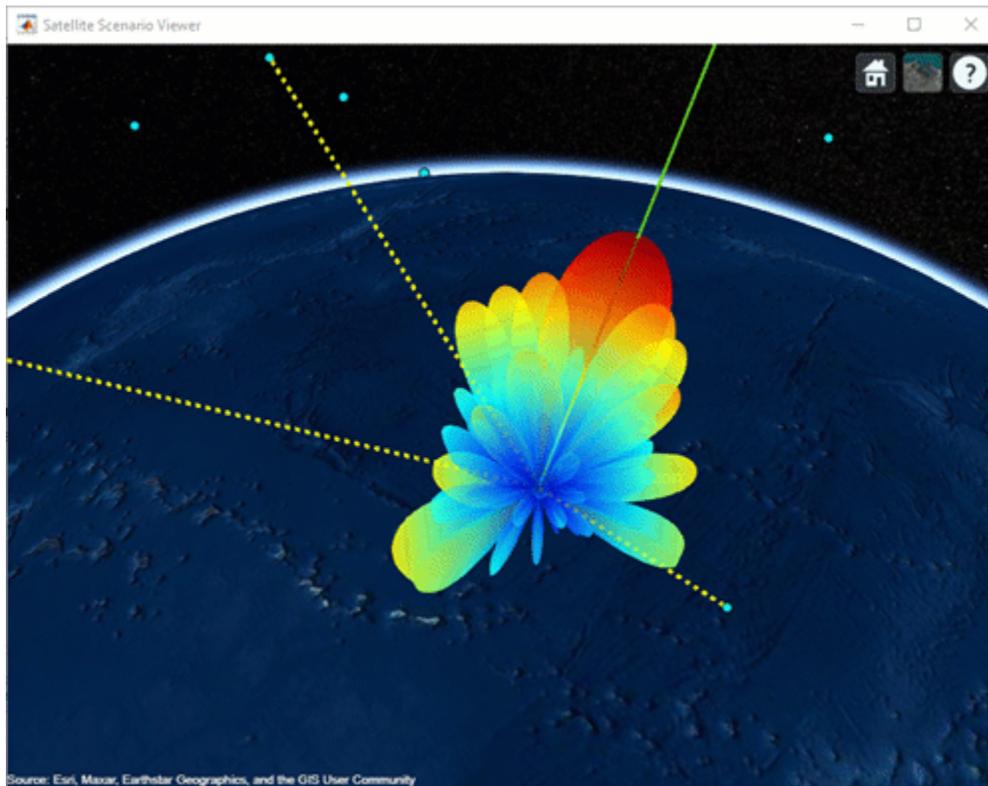
% Calculate the steering vectors for lookout direction.
% Requires Phased Array System Toolbox.
wd = steervec(getElementPosition(ant)/lambda,[wrapTo180(azd);-eld]);

% Calculate the steering vector for null directions.
% Requires Phased Array System Toolbox.
wn = steervec(getElementPosition(ant)/lambda,[wrapTo180(azn)';-eln']);

% Compute the response of desired steering at null direction.
rn = (wn'*wn)\(wn'*wd);

% Sidelobe canceler - remove the response at null direction.
w = wd-wn*rn;

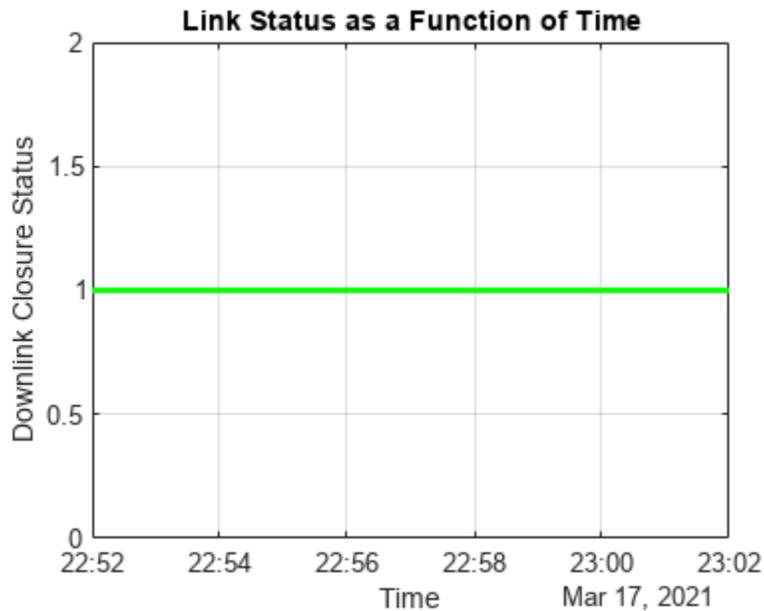
% Assign the weights to the phased array.
pointAt(rxGs,Weights=w);
end
end
```



Plot Downlink Closure Status Neglecting Interference

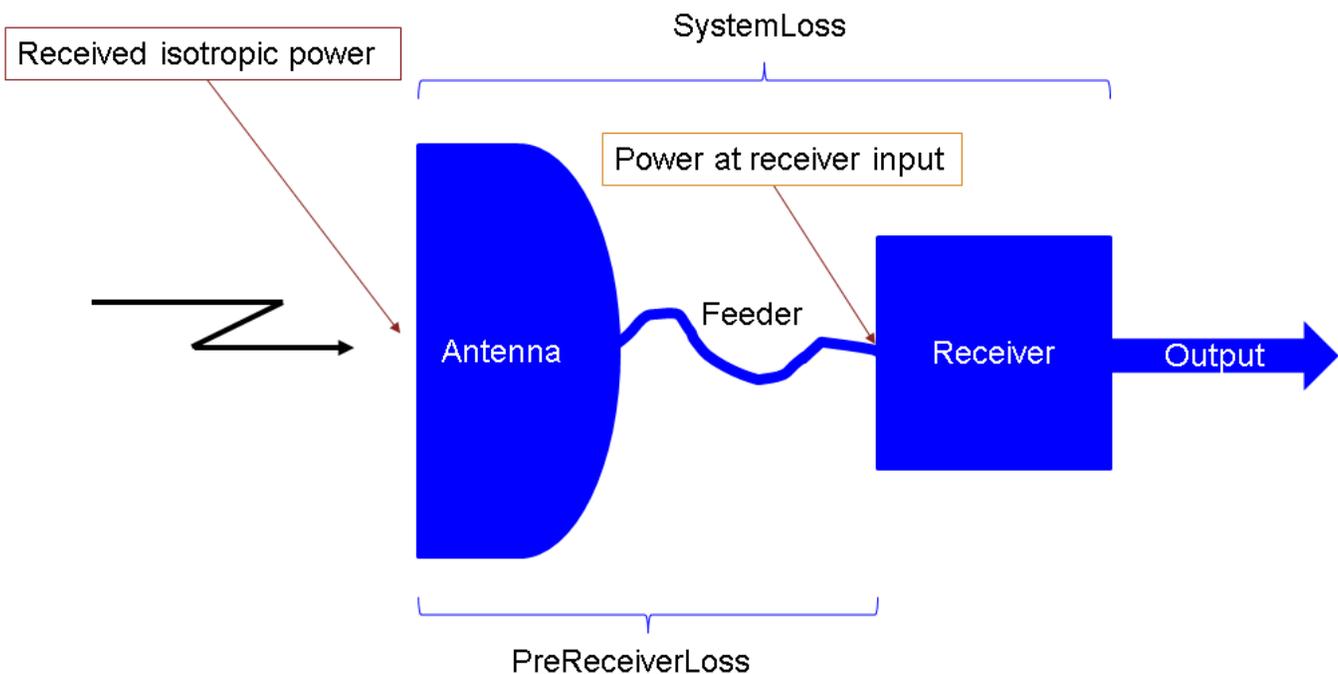
Determine the closure status of the desired downlink from the MEO satellite. The `linkStatus` function neglects interference from other transmitters. Any time the downlink is closed, the status is true. Otherwise, the status is false. The status is indicated by 1 and 0, respectively in the plot.

```
[downlinkStatus,t] = linkStatus(downlink);  
plot(t,downlinkStatus,"-g",LineWidth=2);  
xlabel("Time");  
ylabel("Downlink Closure Status");  
title("Link Status as a Function of Time");  
grid on;
```



Calculate Downlink Closure Status with Interference

Calculate the downlink closure status with interference by first calculating the MEO downlink and interference signal power levels at the ground station receiver input using sigstrength. The locations of received power measurements and losses are illustrated in the ground station receiver diagram below.



`% Calculate the power at receiver input corresponding to the downlink from the MEO satellite.`

```
[~,downlinkPowerRxInput] = sigstrength(downlink); % In dBW
```

```
% Calculate the interference power at receiver input corresponding to each  
% LEO satellite.
```

```
[~,interferencePowerRxInput] = sigstrength(lnkInterference); % In dBW
```

Calculate total interfering signal power at the receiver input. Get this quantity by summing the individual power levels from the interfering LEO satellites in Watts.

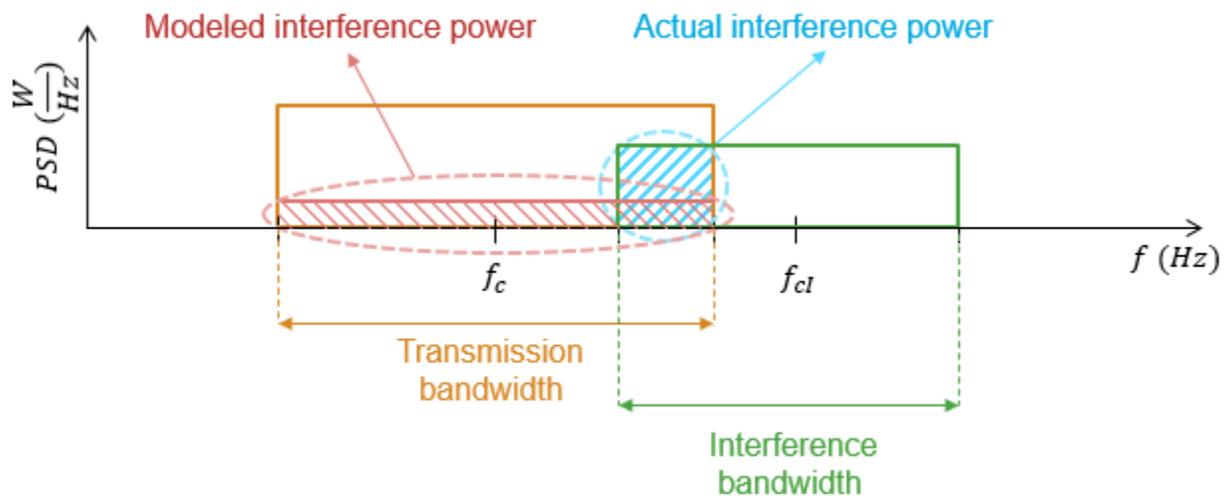
```
interferencePowerRxInputW = 10.^(interferencePowerRxInput/10); % W  
interferencePowerRxInputSumW = sum(interferencePowerRxInputW); % W
```

Calculate the amount of total interfering signal power that contributes to interference in the signal bandwidth by following these steps.

1) Calculate the overlapping portion of the signal bandwidth with the bandwidth of the interferers. This example considers the transmission power of interfering satellites and the MEO satellite as constant across the whole bandwidth of respective MEO satellite and interfering satellites.

2) Calculate the amount of interference power that acts as interference to signal bandwidth.

This diagram shows the power spectral density (PSD) plot, which shows the actual interference power and modeled interference power when the transmission bandwidth and interfering bandwidth overlap. The actual interference power is the area occupied by the interference power density in the overlapped bandwidth region. This actual interference power is then spread across the entire transmission bandwidth and assumed to be noise-like.



This example assumes that the transmission (or signal) bandwidth of the MEO satellite is 30 MHz and that the bandwidth of the interfering signal is 20 MHz.

```
txBandwidth = 30e6; % In Hz  
interferenceBandWidth = 20e6; % In Hz
```

```
% Get the overlap portion of the interfering bandwidth and the bandwidth of  
% interest. The assumption is to have same interference power across  
% the whole bandwidth.
```

```
overlapFactor = getOverlapFactor(txMEOfreq,txBandwidth, ...
```

```

interferenceFreq,interferenceBandwidth);

% Get the interference power that contributes as interference to the signal
% of interest from the total interference power
interferencePowerRxInputActual = interferencePowerRxInputSumW*overlapFactor; % In W

Interference is modeled by treating the contribution of interfering signal power in the overlapped
bandwidth as noise. Accordingly, add this quantity to the thermal noise at the ground station receiver
input. Note that the interference and noise power levels must be added in Watts.

% Calculate the thermal noise at the ground station receiver input.
T = HelperGetNoiseTemperature(txMEOFreq,rxGs); % In K
kb = physconst("Boltzmann");
thermalNoise = kb*T*txBandwidth; % In W

% Calculate the noise plus interference power at the receiver input.
noisePlusInterferencePowerW = thermalNoise + interferencePowerRxInputActual; % In W
noisePlusInterferencePower = 10*log10(noisePlusInterferencePowerW); % In dBW

```

Calculate the carrier to noise plus interference power spectral density ratio at the demodulator input as follows:

$$C/(N_0 + I_0) = P_{RxInput} - (N + I) + 10\log_{10}TxBandwidth - LOSS_{Rx},$$

where:

- $C/(N_0 + I_0)$ is the carrier to noise plus interference power density ratio at the demodulator input (in dB).
- $P_{RxInput}$ is the received downlink power from the MEO satellite measured at the ground station receiver input (in dBW).
- $(N + I)$ is the sum of receiver system thermal noise and the contribution of interfering signal power in the overlapped bandwidth measured at the receiver input (in dBW).
- $TxBandwidth$ is the downlink transmission bandwidth from MEO satellite (in Hz).
- $LOSS_{Rx}$ is the loss that occurs between the receiver input and the demodulator input (in dB).

```

% Calculate loss that occurs between the receiver input and the demodulator
% input.
rxGsLoss = rxGs.SystemLoss - rxGs.PreReceiverLoss;

```

```

% Calculate C/(N0+I0) at the demodulator input.
CNoPlusInterference = downlinkPowerRxInput - ...
    noisePlusInterferencePower + 10*log10(txBandwidth) - rxGsLoss;

```

Calculate the energy per bit to noise plus interference power spectral density ratio at the demodulator input as follows:

$$E_b/(N_0 + I_0) = C/(N_0 + I_0) - 10\log_{10}BITRATE - 60,$$

where:

- $E_b/(N_0 + I_0)$ is the energy per bit to noise plus interference power spectral density ratio at the demodulator input (in dB).
- $BITRATE$ is the bit rate of the downlink from the MEO satellite (in Mbps).

```
bitRate = txMEOSat.BitRate;
ebNoPlusInterference = CNoPlusInterference - 10*log10(bitRate) - 60;
```

Calculate the link margin as follows:

$$MARGIN = E_b/(N_0 + I_0) - (E_b/N_0)_{Required}$$

where:

- $MARGIN$ is the link margin (in dB).
- $(E_b/N_0)_{Required}$ is the minimum received energy per bit to noise power spectral density ratio at the demodulator input that is required to close the link (in dB).

```
marginWithInterference = ebNoPlusInterference - rxGs.RequiredEbNo;
```

Calculate the downlink closure status with interference. The status is true whenever the link margin is greater than or equal to 0 dB.

```
downlinkStatusWithInterference = marginWithInterference >= 0;
```

Calculate Energy per Bit to Noise Power Spectral Density Ratio

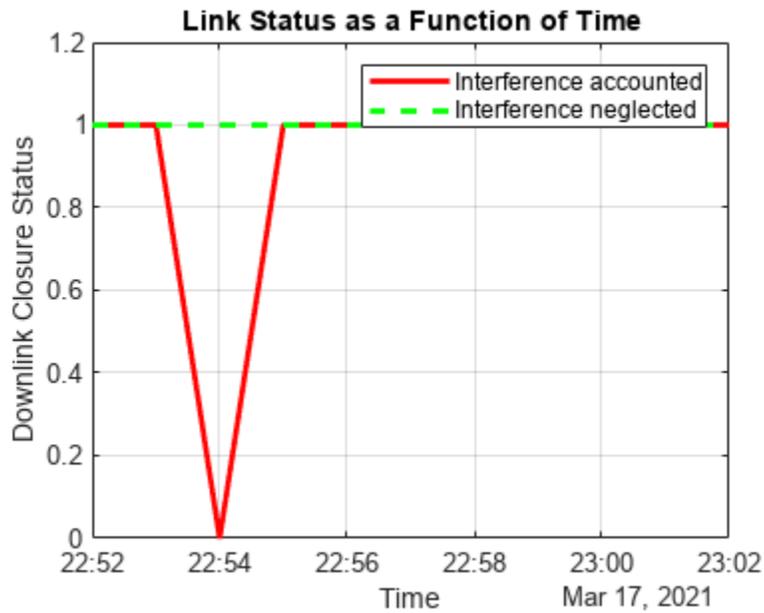
Calculate the energy per bit to noise power spectral density ratio (E_b/N_0) of the downlink and interfering links at the demodulator input for analysis later.

```
ebnoDownlink = ebno(downlink);           % In dB
ebnoInterference = ebno(lnkInterference); % In dB
```

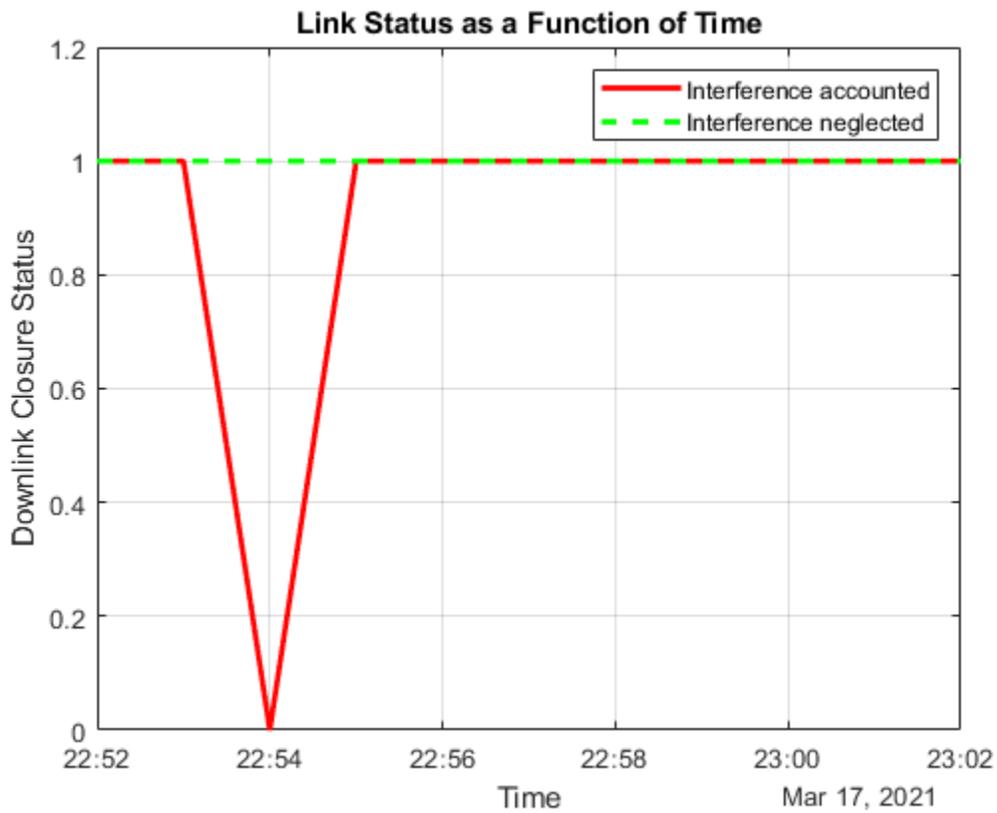
Plot Downlink Closure Status with Interference

Plot the new downlink closure status that accounts for interference. Compare the new link status with the previous case when interference was neglected.

```
plot(t,downlinkStatusWithInterference,"-r",t,downlinkStatus,"--g",LineWidth=2);
legend("Interference accounted","Interference neglected");
xlabel("Time");
ylabel("Downlink Closure Status");
title("Link Status as a Function of Time");
ylim([0 1.2]);
grid on
```



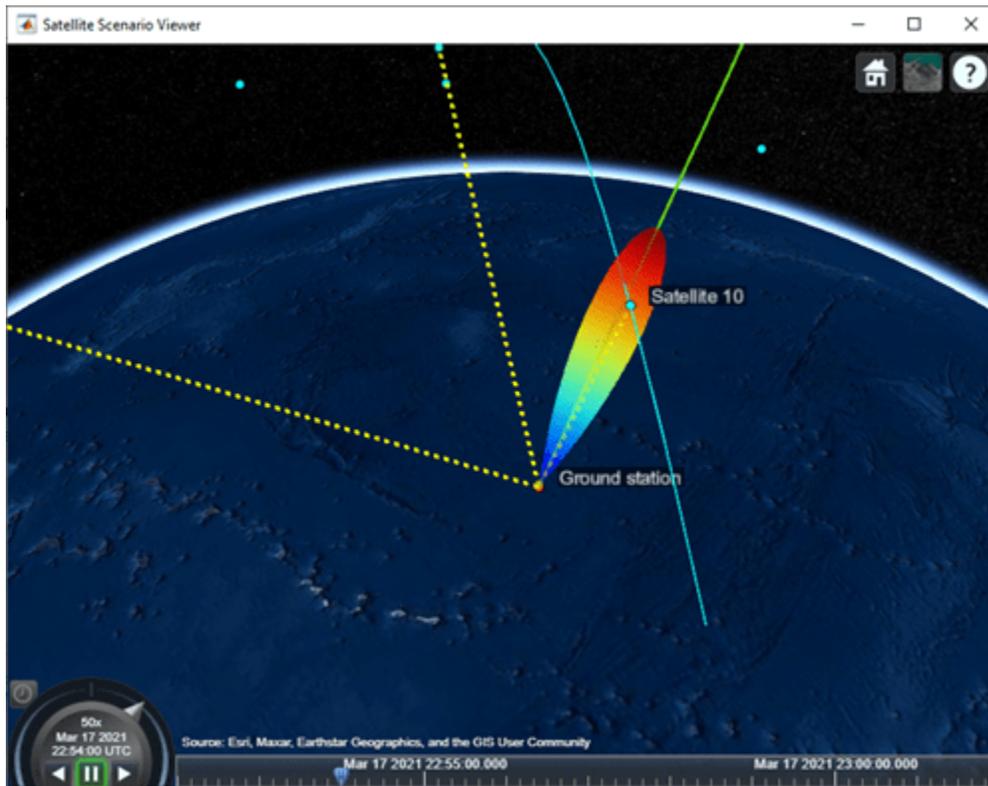
When Gaussian Antenna or Parabolic Reflector are Chosen



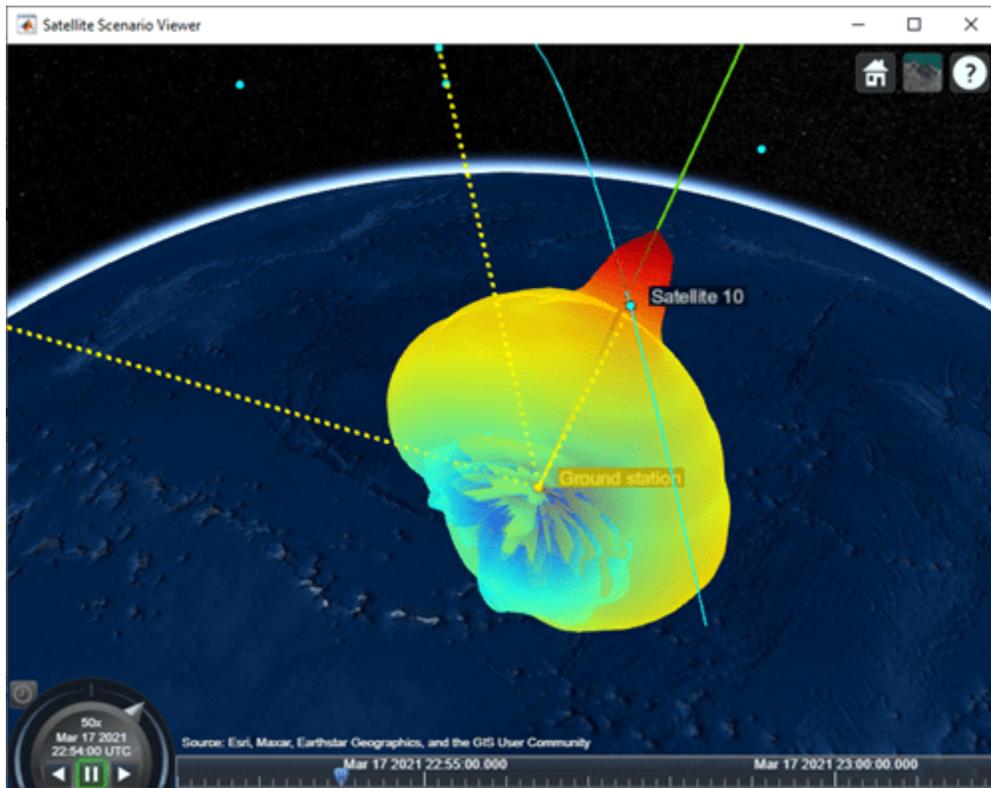
The plot shows that at 10:54 PM, the downlink cannot be closed because of excessive interference. This occurs because Satellite 10 of the LEO constellation flies overhead, and its transmission is

picked up by its main lobe. This can also be visually confirmed by setting the current time of the viewer to 10:54 PM and clicking on the satellite near the main lobe of the antenna. Note that you require Antenna Toolbox to select Parabolic Reflector.

```
if groundStationAntennaType == "Gaussian Antenna" || groundStationAntennaType == "Parabolic Reflector"  
    v.CurrentTime = datetime(2021,3,17,22,54,0);  
end
```

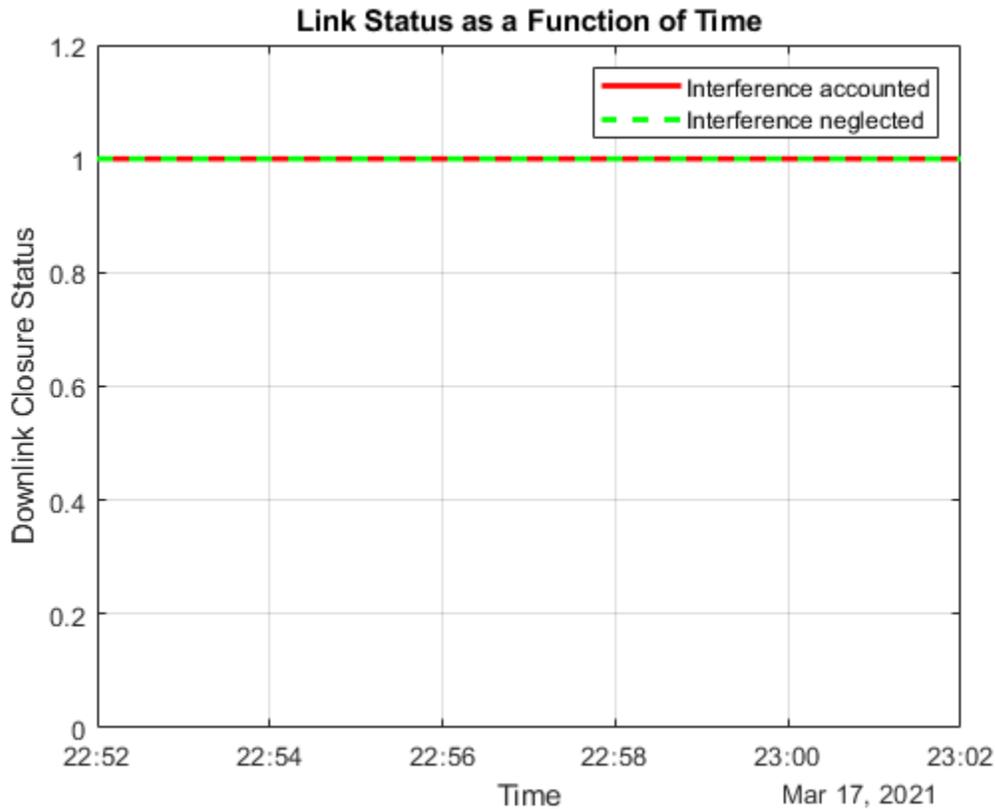


Gaussian Antenna



Parabolic Reflector

When Uniform Rectangular Array with Interference Cancellation Is Chosen



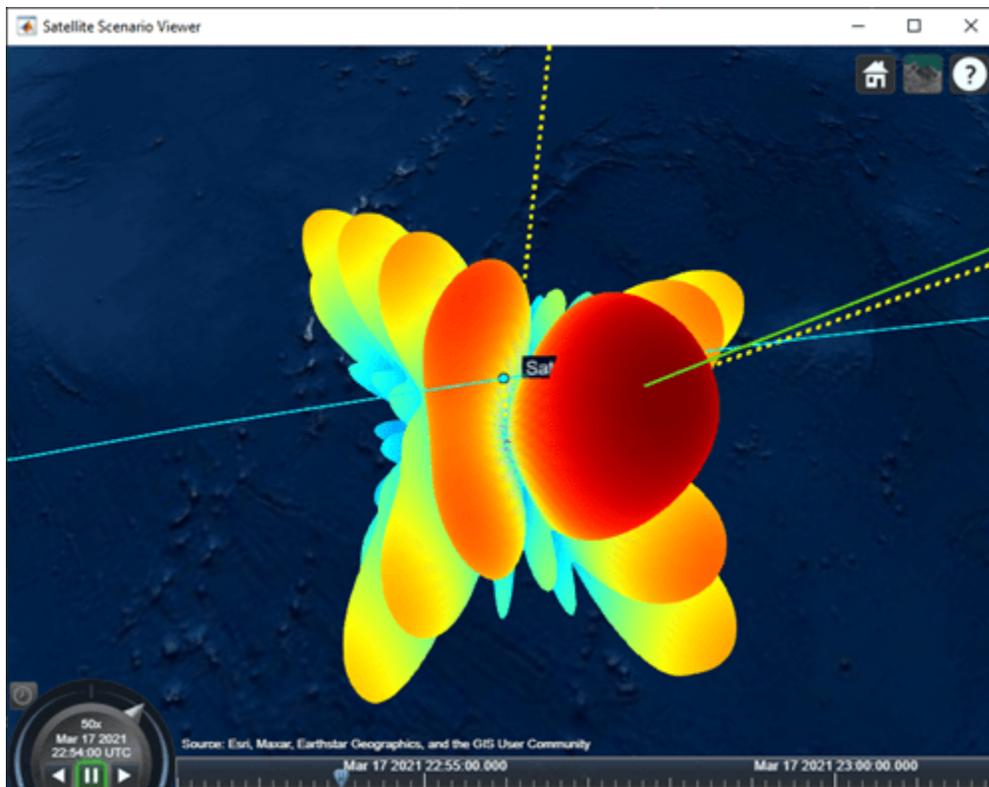
If you selected Uniform Rectangular Array (requires Phased Array System Toolbox), the plot shows that the downlink can be closed for the duration of the scenario because the array is pointing nulls towards the direction of the interfering LEO satellites. This can also be visually confirmed by setting the current time of the viewer to 10:54 PM and 10:55 PM. To be able to manually set the viewer `CurrentTime`, you must change `AutoSimulate` to true. Note that this will clear the simulation data. Also, you will be required to re-compute the weights for these times and assign them to the array using `pointAt`. The satellite that is overflying the ground station is `Satellite 10`. Click on it to see its name and orbit. Drag the mouse while holding down on the left mouse button or scroll button to bring the camera to the desired position and orientation. Rotate the scroll wheel to control camera zoom. Additionally, make the radiation pattern opaque to clearly visualize the position of `Satellite 10` with respect to the lobes. You can see that at both times, `Satellite 10` is in between lobes. This is because the array is pointing a null towards the satellite, thereby cancelling interference from it.

```
if groundStationAntennaType == "Uniform Rectangular Array"
    % Set AutoSimulate to true.
    sc.AutoSimulate = true;

    % Set viewer CurrentTime to 10:54 PM.
    time = datetime(2021,3,17,22,54,0);
    v.CurrentTime = time;

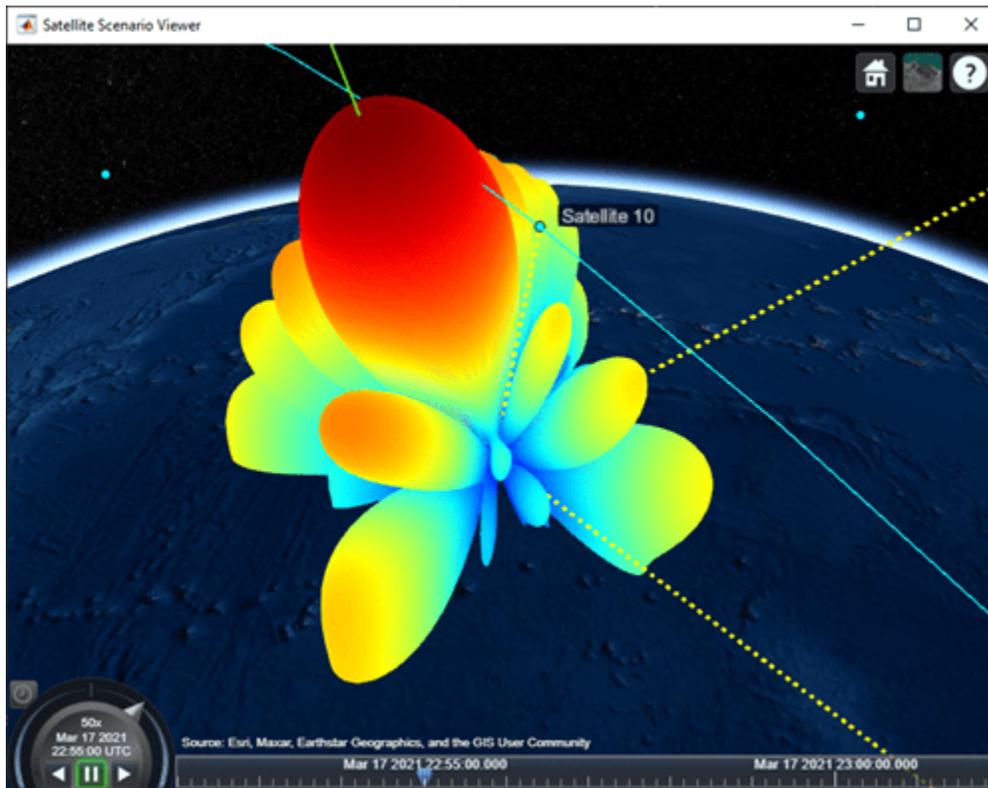
    % Calculate the weights and assign them to the array.
    currentInterferingSat = interferingSat(accessStatus(ac,time) == true);
    [azd,eld] = aer(rxGs,meoSat,time,CoordinateFrame='body');
```

```
[azn,eln] = aer(rxGs,currentInterferingSat,time,CoordinateFrame='body');  
  
% Requires Phased Array System Toolbox.  
wd = steervec(getElementPosition(ant)/lambda,[wrapTo180(azd);-eld]);  
wn = steervec(getElementPosition(ant)/lambda,[wrapTo180(azn)';-eln']);  
  
rn = (wn'*wn)\(wn'*wd);  
w = wd-wn*rn;  
pointAt(rxGs,Weights=w);  
  
% Make the radiation pattern opaque.  
pattern(rxGs,txMEOFreq, ...  
        Size = 1000000, ...  
        Transparency = 1);  
end
```



At 10:54 PM UTC

You can run the above code with time set to 10:55 PM and observe the nulls pointing towards the new positions of the interfering satellites.



At 10:55 PM UTC

Calculate and Plot Carrier to Noise Ratio and Carrier to Noise Plus Interference Ratio

Calculate the carrier to noise ratio (CNR) at the demodulator input as follows:

$$C/N_0 = E_b/N_0 + 10\log_{10}BITRATE + 60,$$

$$C/N = C/N_0 - 10\log_{10}TxBandwidth,$$

where:

- C/N is the carrier to noise ratio at the demodulator input (in dB).
- E_b/N_0 is the carrier to noise power spectral density ratio at the demodulator input (in dB).

```
% Calculate the carrier to noise power spectral density ratio.
CNoDownlink = ebnoDownlink + 10*log10(bitRate) + 60;
```

```
% Calculate the carrier to noise ratio.
cByN = CNoDownlink - 10*log10(txBandwidth);
```

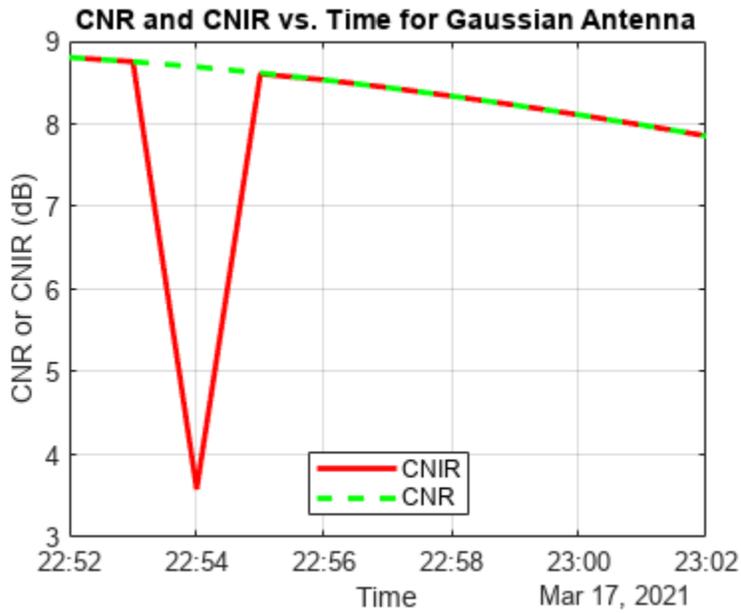
Calculate the carrier to noise plus interference ratio (CNIR) at the demodulator input as follows:

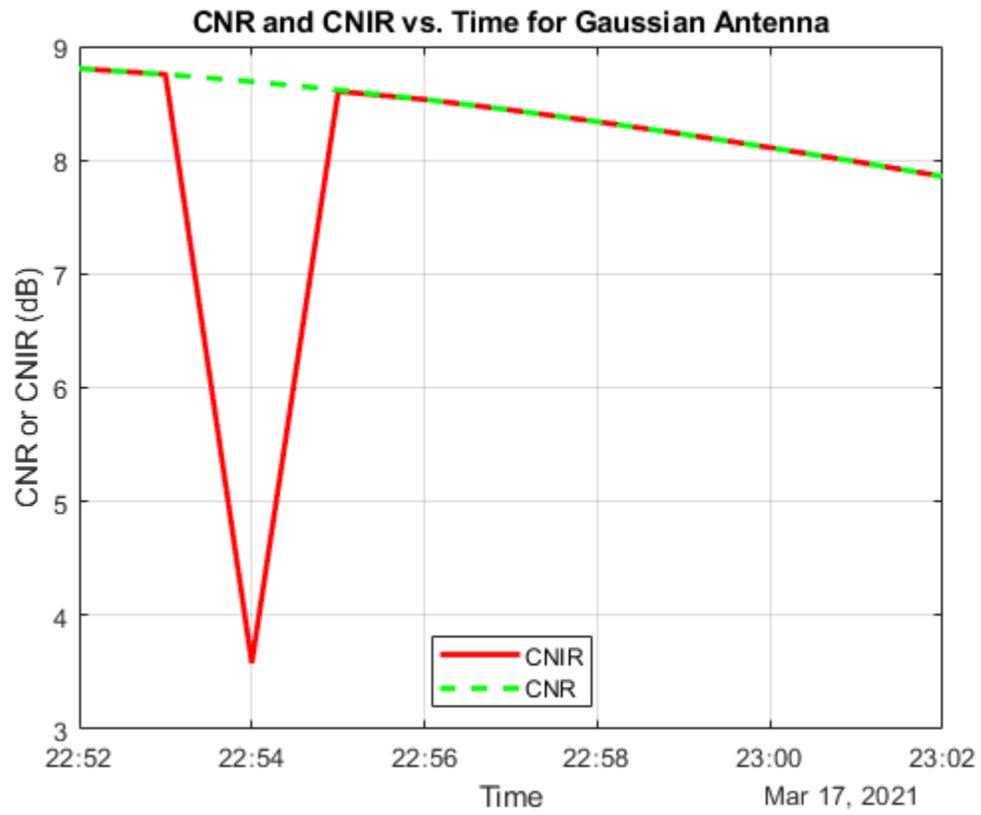
$$C/(N + I) = C/(N_0 + I_0) - 10\log_{10}TxBandwidth$$

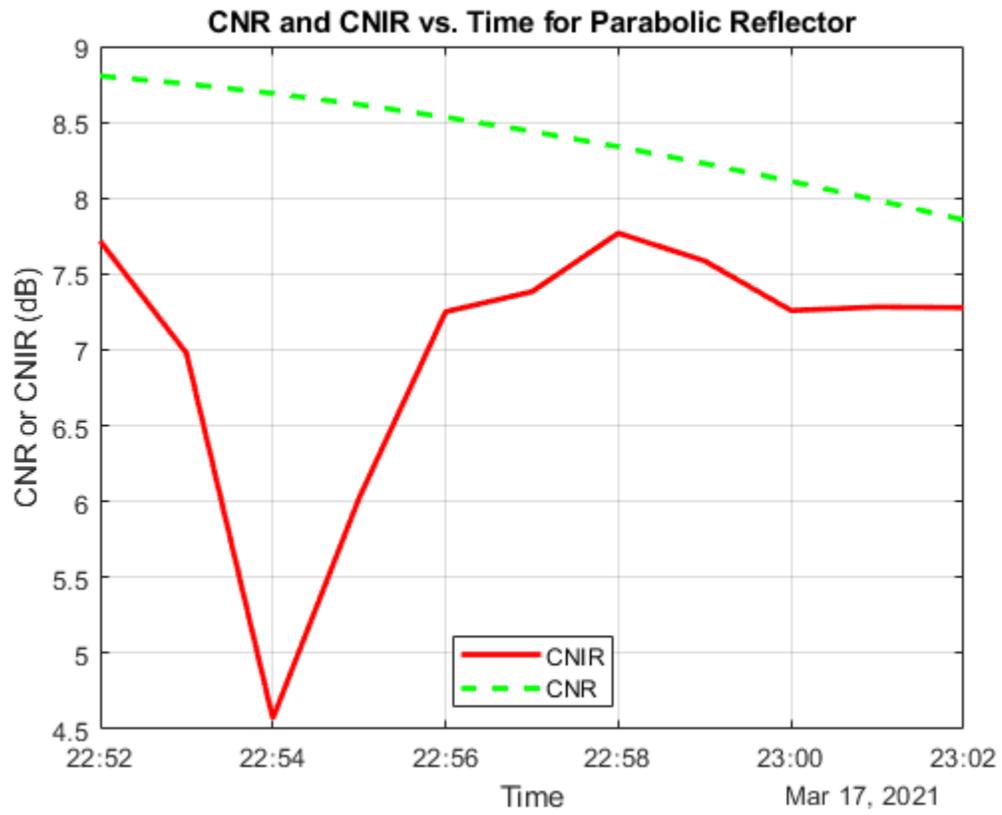
```
cByNPlusI = CNoPlusInterference - 10*log10(txBandwidth);
```

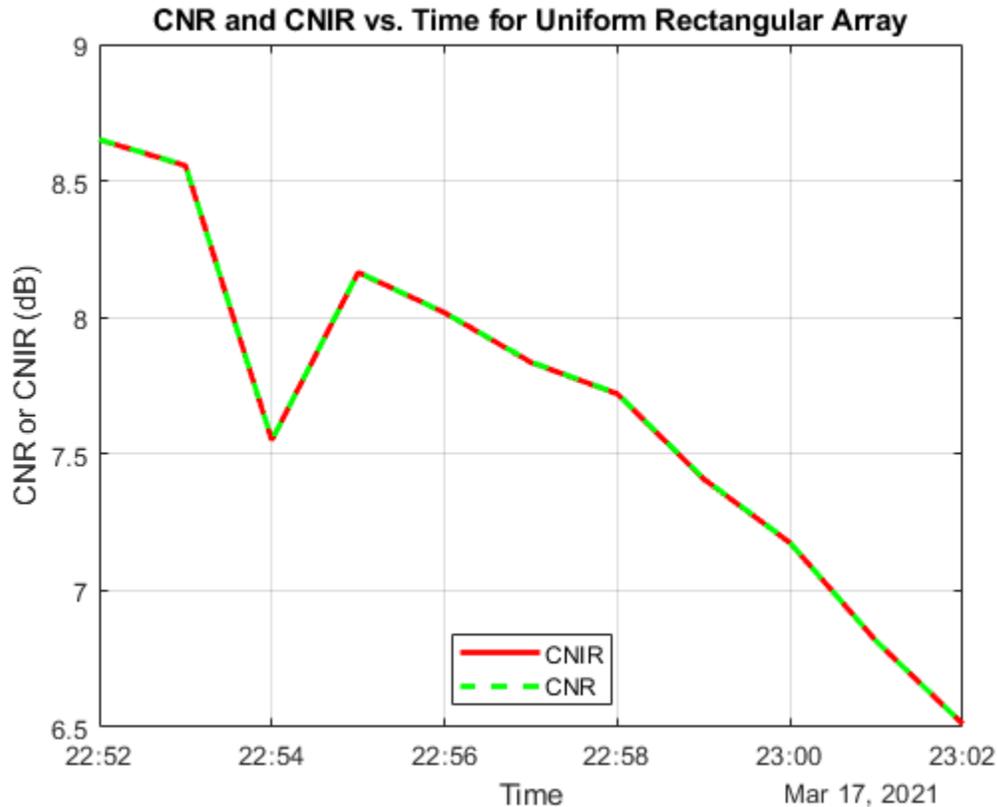
Plot CNR and CNIR

```
plot(t,cByNPlusI,"-r",t,cByN,"-g",LineWidth=2);  
legend("CNIR", "CNR",Location="south");  
xlabel("Time");  
ylabel("CNR or CNIR (dB)");  
title("CNR and CNIR vs. Time for " + groundStationAntennaType);  
grid on
```









When Uniform Rectangular Array (requires Phased Array System Toolbox) with MEO satellite tracking and interference cancellation is chosen, both CNR and CNIR overlap because there is no interference from the LEO satellites. This is because the array is pointing nulls towards the LEO satellites. This can be confirmed by noting that the maximum power at ground station receiver input from the interfering satellites is about -167.3 dBW, which is very low. For all other antennas used in this example, the maximum power at ground station receiver input is much higher (-125.9 dBW for Gaussian Antenna, and -127.3 dBW for Parabolic Reflector).

```
maxInterferencePowerRxInput = max(interferencePowerRxInput,[],'all');
disp("The maximum power at ground station receiver input from the interfering LEO satellites over the time period is:");
```

The maximum power at ground station receiver input from the interfering LEO satellites over the time period is: -167.3 dBW

Compare Link Margins with and without Interference

Calculate the link margin without interference as follows:

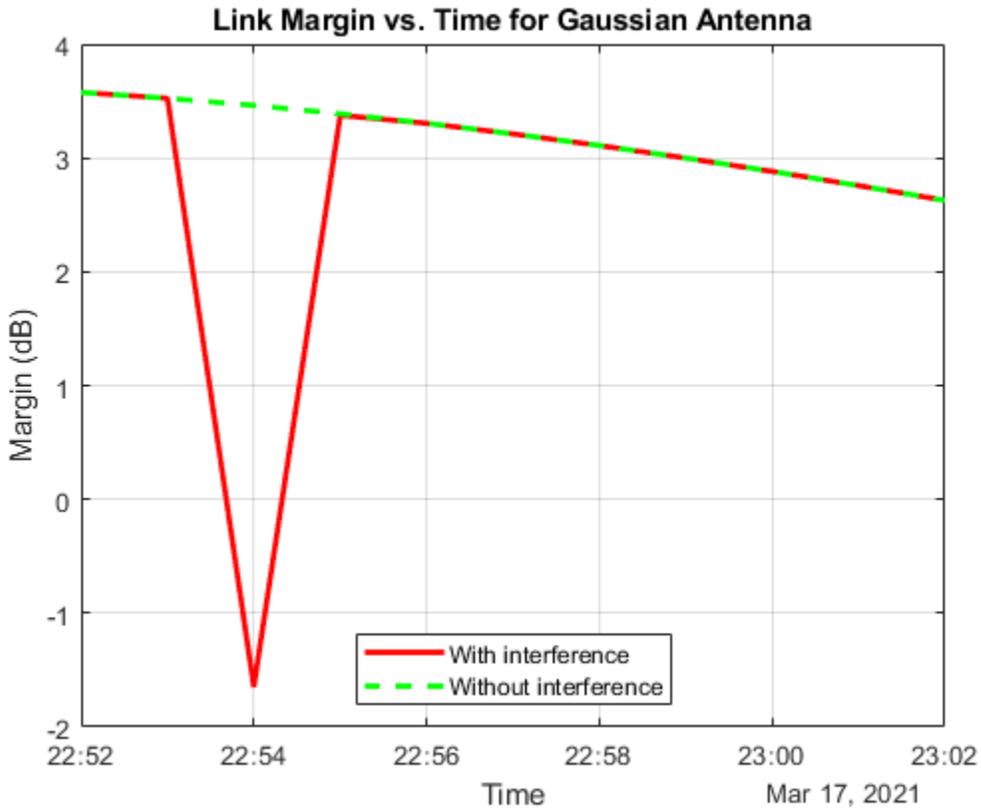
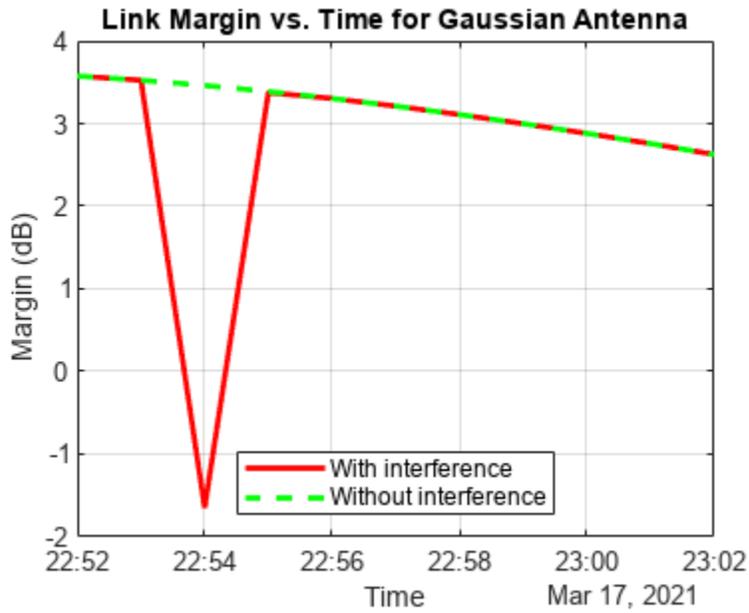
$$MARGIN = E_b/N_0 - (E_b/N_0)_{Required}$$

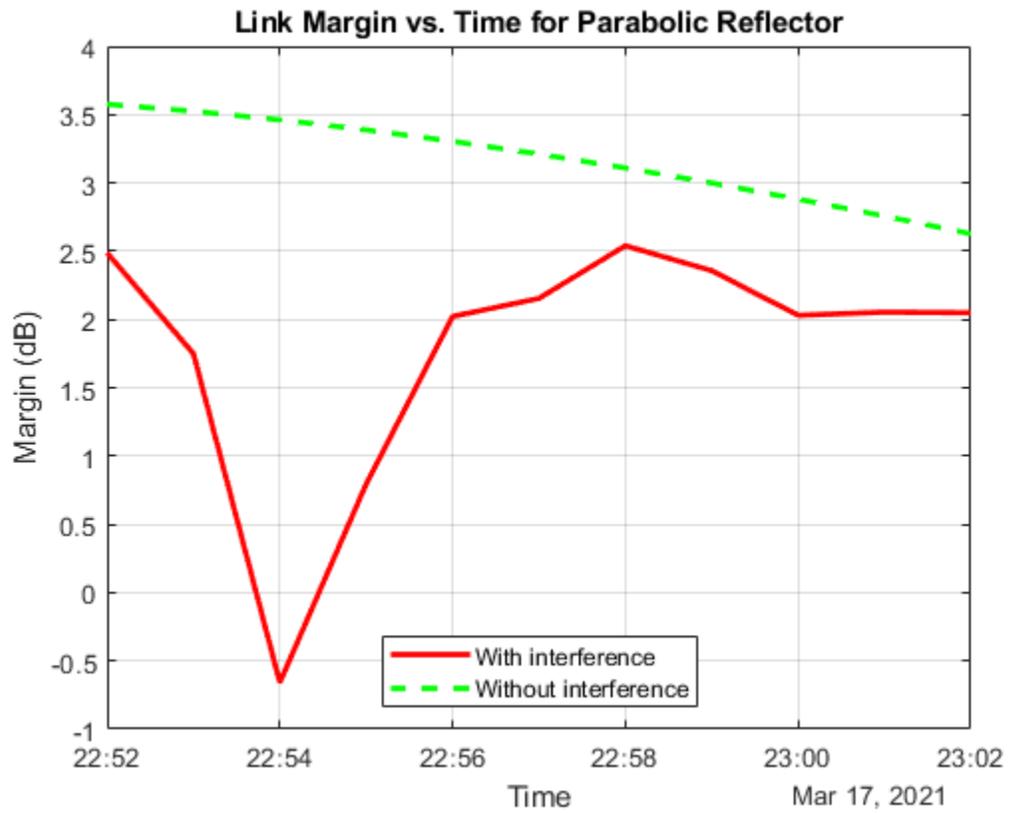
```
marginWithoutInterference = ebnoDownLink - rxGs.RequiredEbNo;
```

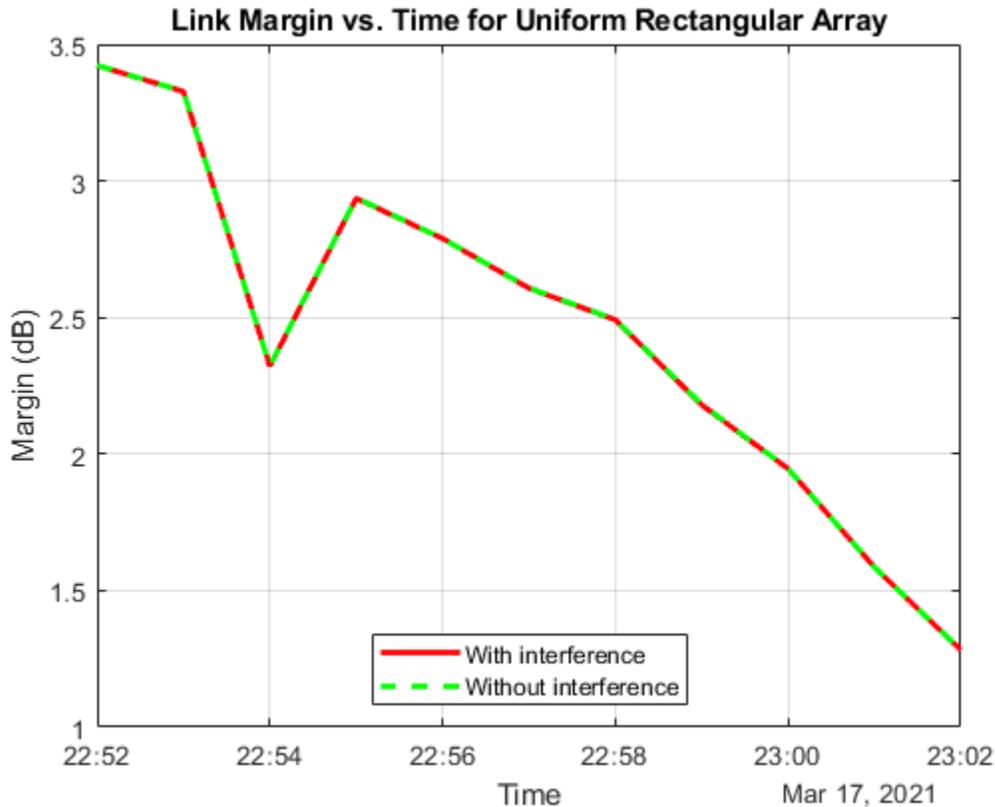
Plot the link margins with and without interference.

```
plot(t,marginWithInterference,"-r",t,marginWithoutInterference,"--g",LineWidth=2);
legend("With interference","Without interference",Location="south");
xlabel("Time");
ylabel("Margin (dB)");
```

```
title("Link Margin vs. Time for " + groundStationAntennaType);  
grid on
```







Any time the link margin is greater than or equal to 0 dB, the downlink is closed. With Gaussian Antenna, Parabolic Reflector (requires Antenna Toolbox), and Uniform Rectangular Array (requires Phased Array System Toolbox) without interference cancellation, there exist times when the link margin dips below 0 dB because of interference. At these times, the downlink is broken.

Further Exploration

This example demonstrates how to analyze interference on a satellite communication link. The link closure times are a function of these parameters:

- The orbit of the satellites
- The position of the ground station
- The specifications of the transmitters and the receiver
- The specifications of the transmitter and receiver antennas
- Weights if using a Uniform Rectangular Array
- The signal and interference bandwidth

Modify these parameters to observe their influence on the level of interference on the link. You can also choose the different antennas from Antenna toolbox and Phased Array System Toolbox for transmitters and receivers and observe the link performance. When using phased arrays and if you are only interested in making the main lobe track a single target and not deal with pointing nulls, you can use `pointAt` to automatically track other satellites, ground stations, and geographic locations without having to manually simulate by setting `AutoSimulate` to false. The limitation of calling `play`

when using dynamically steered phased arrays is that you cannot visualize the variation of their radiation pattern over the course of the simulation.

Helper Functions

The example uses the helper function `HelperGetNoiseTemperature` to obtain the noise temperature of the receiver antenna.

The example also uses this local function to compute the amount of overlap between the transmission bandwidth and the interfering bandwidth.

```
function overlapFactor = getOverlapFactor(txFreq,txBW,interferenceFreq,interferenceBW)
% getOverlapFactor provides the amount of interference bandwidth overlapped
% with transmission bandwidth

    txFreq_Limits = [txFreq-(txBW/2) txFreq+(txBW/2)];
    interferenceFreq_Limits = [interferenceFreq-(interferenceBW/2) ...
        interferenceFreq+(interferenceBW/2)];
    if (interferenceFreq_Limits(2) < txFreq_Limits(1)) || ...
        (interferenceFreq_Limits(1) > txFreq_Limits(2))
        % If no overlap exists between transmission bandwidth and
        % interfering bandwidth, then overlap factor is 0
        overlapFactor = 0;
    elseif (interferenceFreq_Limits(2) <= txFreq_Limits(2)) && ...
        (interferenceFreq_Limits(1) >= txFreq_Limits(1))
        % If interfering bandwidth lies completely within transmission
        % bandwidth, then overlap factor is 1
        overlapFactor = 1;
    elseif (interferenceFreq_Limits(2) > txFreq_Limits(2)) && ...
        (interferenceFreq_Limits(1) < txFreq_Limits(1))
        % If transmission bandwidth lies completely within interfering
        % bandwidth, then overlap factor is the ratio of transmission
        % bandwidth with that of interference bandwidth
        overlapFactor = txBW/interferenceBW;
    elseif (interferenceFreq_Limits(2) <= txFreq_Limits(2)) && ...
        (interferenceFreq_Limits(1) <= txFreq_Limits(1))
        % If the start edge of transmission bandwidth lies within
        % interfering bandwidth, then overlap factor is the ratio of
        % difference from last edge of interfering bandwidth and first edge
        % of signal bandwidth, with that of interference bandwidth
        overlapFactor = (interferenceFreq_Limits(2)-txFreq_Limits(1))/interferenceBW;
    else
        % If the last edge of transmission bandwidth lies within
        % interfering bandwidth, then overlap factor is the ratio of difference
        % from last edge of signal bandwidth and first edge of interfering
        % bandwidth, with that of interference bandwidth
        overlapFactor = (-interferenceFreq_Limits(1)+txFreq_Limits(2))/interferenceBW;
    end
end
```

See Also

Objects

`satelliteScenario` | `Satellite` | `Access` | `GroundStation` | `satelliteScenarioViewer` | `ConicalSensor` | `Transmitter` | `Receiver` | `Gimbal`

Functions

show | play | hide

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to Ground Station” on page 1-14
- “Comparison of Orbit Propagators” on page 1-25
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Multi-Hop Path Selection Through Large Satellite Constellation

This example shows how to determine the path through a large constellation consisting of 1,000 low-Earth orbit (LEO) satellites to gain access between two ground stations. Following this, it demonstrates how to calculate the intervals during the next 3 hour period when this path can be used.

Create Satellite Scenario

Assume that the path through the large constellation to establish access between two ground stations must be determined as of 10 December 2021, 6:27:57 PM UTC. We must then determine the times over the next 3 hours when this path can be used. Accordingly, create a satellite scenario with the appropriate `StartTime` and `StopTime`. Set `SampleTime` to 60 seconds. Since the path must be determined only for the first time step, set `AutoSimulate` of the scenario to false to prevent it from automatically advancing through the time steps until `StopTime`. When `AutoSimulate` is false, `SimulationStatus` becomes available.

```
startTime = datetime(2021,12,10,18,27,57); % 10 December 2021, 6:27:57 PM UTC
stopTime = startTime + hours(3);          % 10 December 2021, 9:27:57 PM UTC
sampleTime = 60;                          % Seconds
sc = satelliteScenario(startTime,stopTime,sampleTime,"AutoSimulate",false)
```

```
sc =
  satelliteScenario with properties:
      StartTime: 10-Dec-2021 18:27:57
      StopTime: 10-Dec-2021 21:27:57
      SampleTime: 60
      SimulationTime: 10-Dec-2021 18:27:57
      SimulationStatus: NotStarted
      AutoSimulate: 0
      Satellites: [1x0 matlabshared.satellitescenario.Satellite]
      GroundStations: [1x0 matlabshared.satellitescenario.GroundStation]
      Viewers: [0x0 matlabshared.satellitescenario.Viewer]
      AutoShow: 1
```

Add Large Constellation of Satellites

Add the large satellite constellation from the Two-Line-Element (TLE) file `largeConstellation.tle`. The constellation consists of 1,000 LEO satellites.

```
sat = satellite(sc,"largeConstellation.tle");
numSatellites = numel(sat)
```

```
numSatellites = 1000
```

Add Ground Stations

Add the ground stations. A multi-hop path must be established through the satellite constellation for access between the ground stations.

```
gsSource = groundStation(sc,42.3001,-71.3504, ... % Latitude and longitude in degrees
    "Name","Source Ground Station");
gsTarget = groundStation(sc,17.4351,78.3824, ... % Latitude and longitude in degrees
    "Name","Target Ground Station");
```

Determine Elevation Angles of Satellites with Respect to Ground Stations

Determine the elevation angle of each satellite with respect to source and target ground stations corresponding to `StartTime`. Accordingly, use `advance` to simulate the scenario for the first time step, namely, `StartTime`. Following this, use `aer` to retrieve the elevation angle of each satellite with respect to the ground stations. Assume that for the initial routing, the elevation angle of the first satellite in the path with respect to "Source Ground Station" and the last satellite in the path with respect to "Target Ground Station" must be at least 30 degrees. Accordingly, determine the elevation angles that are greater than or equal to this value.

```
% Calculate the scenario state corresponding to StartTime.
advance(sc);

% Retrieve the elevation angle of each satellite with respect to the ground
% stations.
[~,elSourceToSat] = aer(gsSource,sat);
[~,elTargetToSat] = aer(gsTarget,sat);

% Determine the elevation angles that are greater than or equal to 30
% degrees.
elSourceToSatGreaterThanOrEqual30 = (elSourceToSat >= 30)';
elTargetToSatGreaterThanOrEqual30 = (elTargetToSat >= 30)';
```

Determine Best Satellite for Initial Access to Constellation

The best satellite to be used for the initial access to the large constellation is assumed to be the one that satisfies the following simultaneously:

- Has the closest range to "Target Ground Station".
- Has an elevation angle of at least 30 degrees with respect to "Source Ground Station".

```
% Find the indices of the elements of elSourceToSatGreaterThanOrEqual30
% whose value is true.
trueID = find(elSourceToSatGreaterThanOrEqual30 == true);

% These indices are essentially the indices of satellites in sat whose
% elevation angle with respect to "Source Ground Station" is at least 30
% degrees. Determine the range of these satellites to "Target Ground
% Station".
[~,~,r] = aer(sat(trueID), gsTarget);

% Determine the index of the element in r bearing the minimum value.
[~,minRangeID] = min(r);

% Determine the element in trueID at the index minRangeID.
id = trueID(minRangeID);

% This is the index of the best satellite for initial access to the
% constellation. This will be the first hop in the path. Initialize a
% variable 'node' that stores the first two nodes of the routing - namely,
% "Source Ground Station" and the best satellite for initial constellation
% access.
nodes = {gsSource sat(id)};
```

Determine Remaining Nodes in Path to Target Ground Station

The remaining nodes in the path are determined using a similar logic as what was used for determining the first satellite. If the elevation angle of the satellite in the current node is already at

least 30 degrees with respect to "Target Ground Station", the next node is "Target Ground Station", thereby completing the path. Otherwise, the next node in the constellation is chosen using a logic that is similar to what was used for determining the first satellite in the path. The next node is the one that simultaneously satisfies the following:

- Has the closest range to "Target Ground Station".
- Elevation angle with respect to the satellite in the current node is greater than or equal to -15 degrees.

The elevation value of -15 degrees is chosen because the horizon with respect to each satellite in the constellation is about -21.9813 degrees. This value can be derived by assuming a spherical Earth geometry and the fact that these satellites are in near-circular orbits at an altitude of roughly 500 kilometers. Note that the spherical Earth assumption is used only for computing the elevation angle of the horizon below. The satellite scenario simulation itself assumes a WGS84 ellipsoid model for the Earth.

```
earthRadius = 6378137; % meters
altitude = 500000; % meters
horizonElevationAngle = asind(earthRadius/(earthRadius + altitude)) - 90 % degrees

horizonElevationAngle = -21.9813
```

Any satellite whose elevation angle with respect to another satellite is greater than -21.9813 degrees is guaranteed to be visible to the latter. However, choosing -15 degrees provides an adequate margin.

The subsequent nodes are continually added to the path until reaching a satellite whose elevation angle with respect to "Target Ground Station" is at least 30 degrees. After this, the final node is "Target Ground Station" itself, and the routing is complete.

```
% Minimum elevation angle of satellite nodes with respect to the prior
% node.
minSatElevation = -15; % degrees

% Flag to specify if the complete multi-hop path has been found.
pathFound = false;

% Determine nodes of the path in a loop. Exit the loop once the complete
% multi-hop path has been found.
while ~pathFound
    % Index of the satellite in sat corresponding to current node is
    % updated to the value calculated as index for the next node in the
    % prior loop iteration. Essentially, the satellite in the next node in
    % prior iteration becomes the satellite in the current node in this
    % iteration.
    idCurrent = id;

    % This is the index of the element in elTargetToSatGreaterThanOrEqual30
    % tells if the elevation angle of this satellite is at least 30 degrees
    % with respect to "Target Ground Station". If this element is true, the
    % routing is complete, and the next node is the target ground station.
    if elTargetToSatGreaterThanOrEqual30(idCurrent)
        nodes = {nodes{:} gsTarget}; %#ok<CCAT>
        pathFound = true;
        continue
    end

    % If the element is false, the path is not complete yet. The next node
```

```
% in the path must be determined from the constellation. Determine
% which satellites have elevation angle that is greater than or equal
% to -15 degrees with respect to the current node. To do this, first
% determine the elevation angle of each satellite with respect to the
% current node.
[~,els] = aer(sat(idCurrent),sat);

% Overwrite the elevation angle of the satellite with respect to itself
% to be -90 degrees to ensure it does not get re-selected as the next
% node.
els(idCurrent) = -90;

% Determine the elevation angles that are greater than or equal to -15
% degrees.
s = els >= minSatElevation;

% Find the indices of the elements in s whose value is true.
trueID = find(s == true);

% These indices are essentially the indices of satellites in sat whose
% elevation angle with respect to the current node is greater than or
% equal to -15 degrees. Determine the range of these satellites to
% "Target Ground Station".
[~,~,r] = aer(sat(trueID), gsTarget);

% Determine the index of the element in r bearing the minimum value.
[~,minRangeID] = min(r);

% Determine the element in trueID at the index minRangeID.
id = trueID(minRangeID);

% This is the index of the best satellite among those in sat to be used
% for the next node in the path. Append this satellite to the 'nodes'
% variable.
nodes = {nodes{:} sat(id)}; %#ok<CCAT>
end
```

Determine Intervals When Calculated Path Can Be Used

We must now determine the intervals over the next 3 hours during which calculated path can be used. To accomplish this, manually stepping through each time step of the scenario is not necessary. Instead, we can make the scenario auto-simulate from `StartTime` to `StopTime`. Therefore, set `AutoSimulate` of the scenario to `true`.

```
sc.AutoSimulate = true;
```

Add an access analysis with the calculated nodes in the path. Set `LineColor` of the access visualization to red.

```
ac = access(nodes{:});
ac.LineColor = "red";
```

Determine the access intervals using the `accessIntervals` function. Since `AutoSimulate` has been set to `true`, the scenario will automatically simulate from `StartTime` to `StopTime` at the specified `SampleTime` before calculating the access intervals. These intervals are the times when the calculated multi-hop path can be used.

```
intvls = accessIntervals(ac)
```

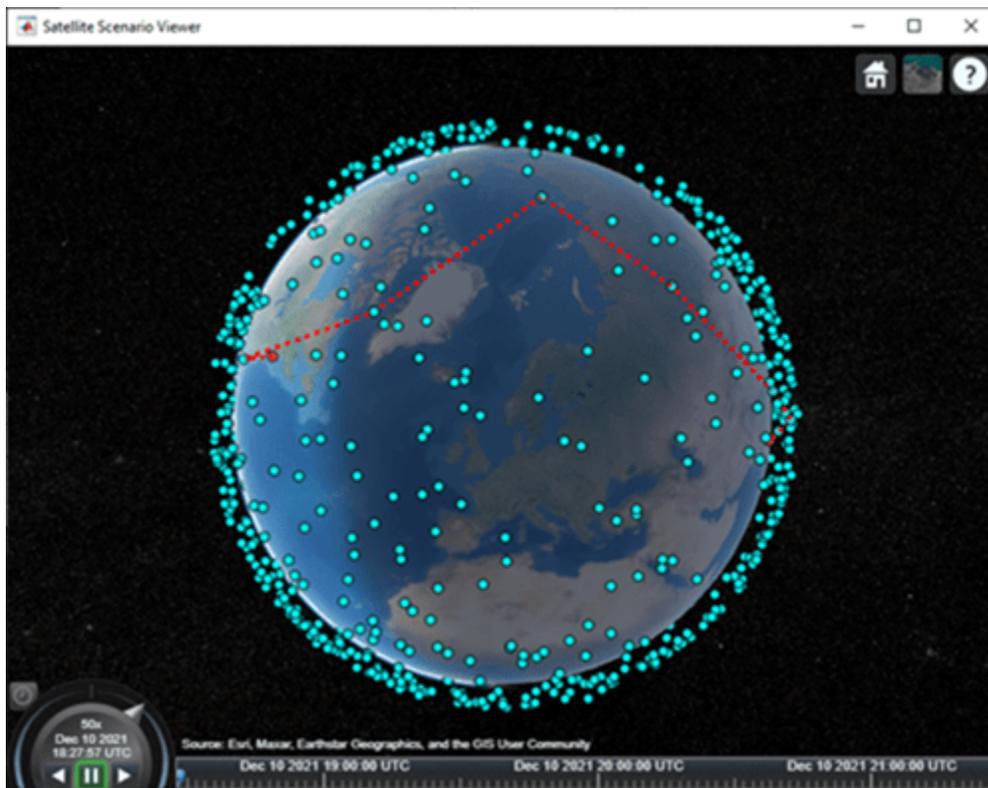
```
intvls=2x8 table
```

Source	Target	IntervalNumber	StartTime
"Source Ground Station"	"Target Ground Station"	1	10-Dec-2021 18:27:57
"Source Ground Station"	"Target Ground Station"	2	10-Dec-2021 20:01:57

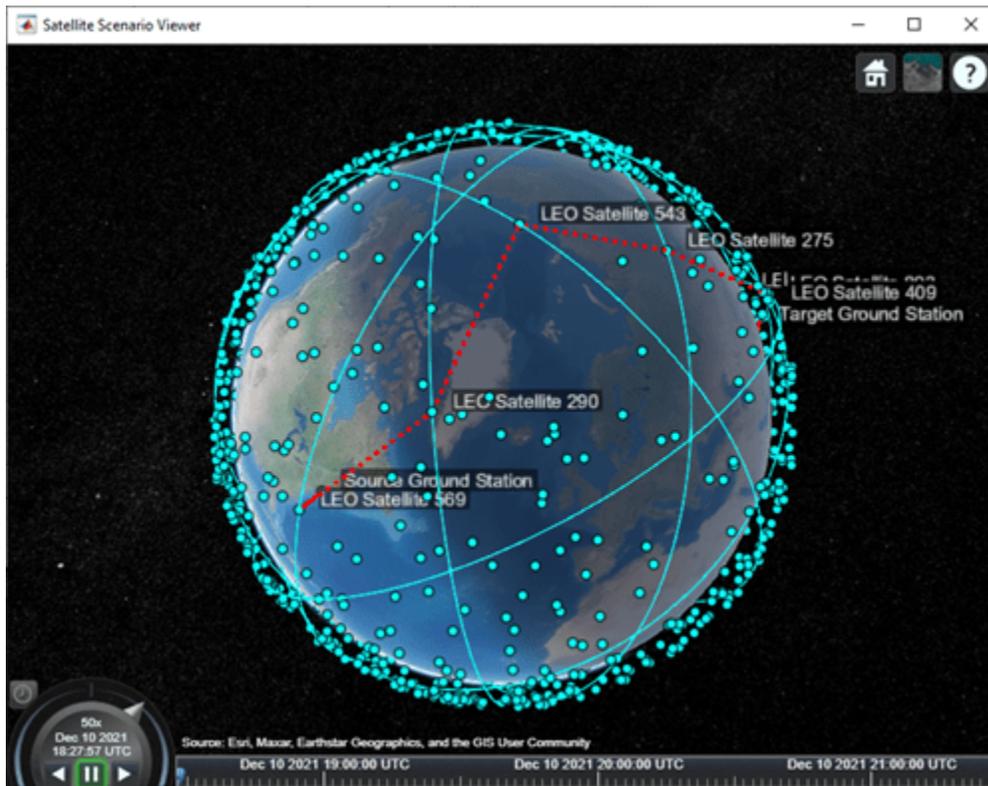
Visualize Path

Launch the satellite scenario viewer with `ShowDetails` set to `false`. When the `ShowDetails` property is set to `false`, only satellites and ground stations will be shown. Labels and orbits will be hidden. Mouse over satellites and ground stations to show their labels. The green lines represent the multi-hop path. Also, set `MarkerSize` of the satellites to 6 to further declutter the visualization. Set the camera position to 60 degrees latitude and 5 degrees longitude to bring the multi-hop path into view.

```
v = satelliteScenarioViewer(sc,"ShowDetails",false);
sat.MarkerSize = 6; % Pixels
campos(v,60,5); % Latitude and longitude in degrees
```

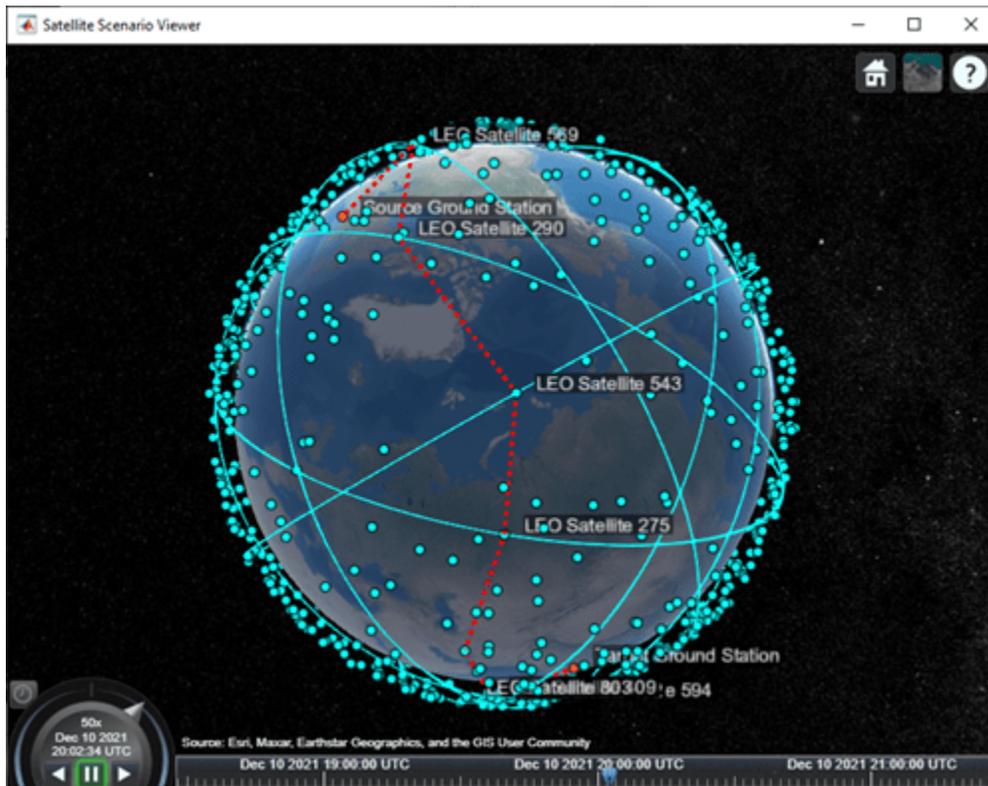


Click on the ground stations to display their labels without requiring to mouse over. Click on the satellites that are part of the multi-hop path to display their orbits and labels without requiring to mouse over.



Play the scenario.

```
play(sc);
```



Next Steps

This example demonstrated how to determine the multi-hop path through a large satellite constellation to gain access between two ground stations. Each subsequent node in the path was selected such that its distance to "Target Ground Station" was the minimum among those satellites whose elevation angle with respect to the previous node was at least -15 degrees. Note that this does not necessarily result in the shortest overall distance along the multi-hop path. One way to find the shortest path is to find all possible paths using a graph search algorithm and then choose the shortest path among them. To find the distance between the different satellites and ground stations, use the third output of the `aer` function.

Additionally, this example computed the path only once for the scenario `StartTime`, which was then used for the duration of the scenario. As a result, the ground stations had access only for 5 minutes out of the 3 hours spanning the scenario duration. To increase the access duration between the ground stations, you can modify the above code to re-compute the path at each scenario time step using `advance` after setting `AutoSimulate` to `false`. Calling `advance` advances `SimulationTime` by one `SampleTime`. Once you compute all the paths, set `AutoSimulate` back to `true`, create access objects corresponding to each unique path, and re-compute the access intervals by calling `accessIntervals` on all the access objects.

See Also

Objects

`satelliteScenario` | `Satellite` | `Access` | `GroundStation` | `satelliteScenarioViewer`

Functions

`play` | `aer` | `advance` | `accessIntervals`

Related Examples

- “Satellite Constellation Access to Ground Station” on page 1-14
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Modeling Custom Satellite Attitude and Gimbal Steering

This example shows how to point a satellite or gimbal in a satellite scenario using logged orientation data from a `timetable` or `timeseries`. It uses data generated by the Aerospace Blockset™ **Spacecraft Dynamics** block. For more information about the data and how to generate it, see the Aerospace Blockset example *Analyzing Spacecraft Attitude Profiles with Satellite Scenario*.

The `satelliteScenario` object lets you load previously generated, time-stamped ephemeris and attitude data into a scenario as `timeseries` or `timetable` objects. Data is interpolated in the `scenario` object to align with the scenario time steps, allowing you to incorporate data generated in a Simulink model into either a new or existing `satelliteScenario` object. For this example, the satellite orbit and attitude states are precomputed by the **Spacecraft Dynamics** block. Load this data to the workspace and use it to model a satellite to a `satelliteScenario` object for access analysis.

Define Mission Parameters and Satellite Initial Conditions

Specify a start date and duration for the mission. This example uses MATLAB® structures to organize mission data. These structures make accessing data later in the example more intuitive. They also help declutter the global base workspace.

```
mission.StartDate = datetime(2021,1,1,12,0,0);
mission.Duration = hours(1.5);
```

Specify initial orbital elements for the satellite.

```
mission.Satellite.SemiMajorAxis = 7.2e6; % meters
mission.Satellite.Eccentricity = .05;
mission.Satellite.Inclination = 70; % deg
mission.Satellite.ArgOfPeriapsis = 0; % deg
mission.Satellite.RAAN = 215; % deg
mission.Satellite.TrueAnomaly = 200; % deg
```

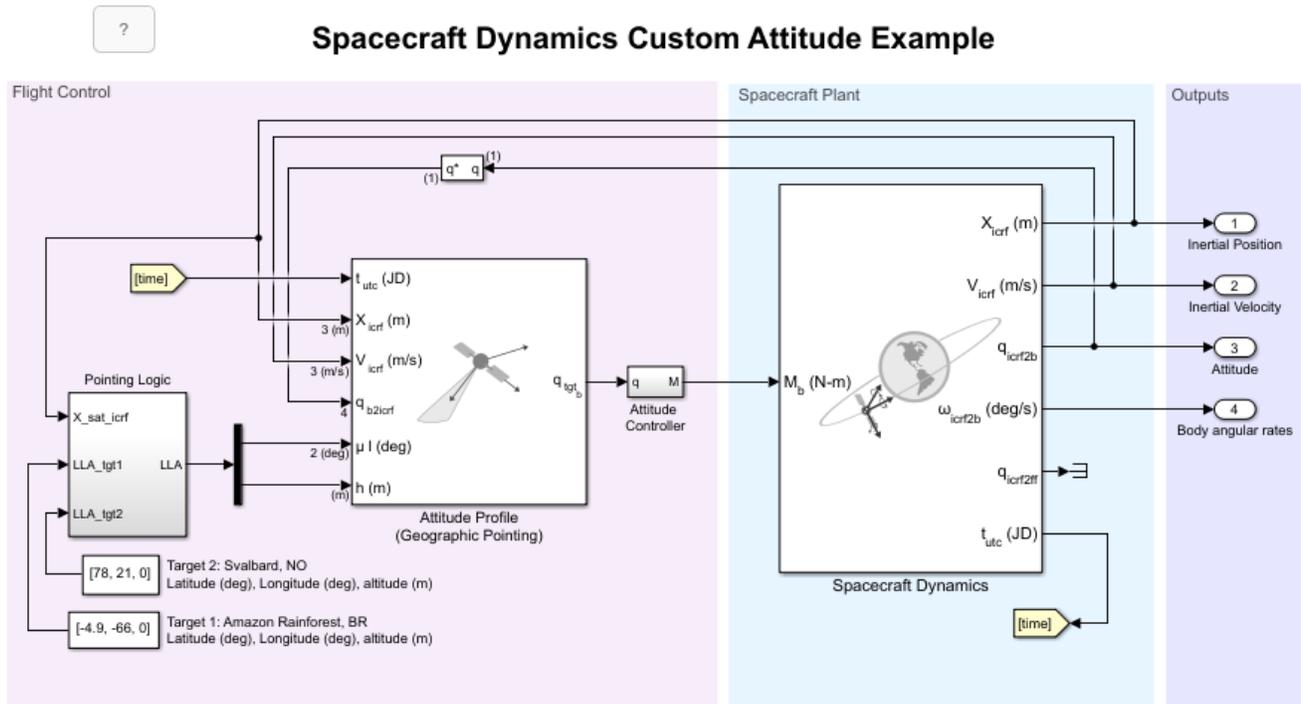
Specify an initial attitude state for the satellite.

```
mission.Satellite.q0 = [1, 0, 0, 0];
mission.Satellite.pqr = [0, 0, 0]; % deg/s
```

Load Ephemeris and Attitude Profile

The Simulink® model used to generate data for this example is configured to perform an Earth Observation mission during which a satellite performs a flyover of a region of the Amazon Rainforest to capture images of, and track deforestation trends in, the area.

The satellite points at the nadir when not actively imaging or downlinking to the ground station in Svalbard, NO. The Aerospace Blockset **Attitude Profile** block calculates commanded attitude. The satellite uses spherical harmonic gravity model EGM2008 for orbit propagation. Gravity gradient torque contributions are also included in attitude dynamics. For more information about this model, see the Aerospace Blockset example *Analyzing Spacecraft Attitude Profiles with Satellite Scenario*.



The timetable objects contain position and attitude data for the satellite throughout the mission. The data is referenced in the inertial (ICRF/GCRF) reference frame. Attitude values are expressed as quaternions, although Euler angles are also supported.

```
mission.Data = load("SatelliteScenarioCustomAttitudeData.mat", "PositionTimeTableGCRF", "AttitudeTimeTableGCRF2Body");
display(mission.Data.PositionTimeTableGCRF)
```

1148x1 timetable

Time	Data		
0 sec	5.2953e+06	4.784e+06	-2.422e+06
5 sec	5.2783e+06	4.7859e+06	-2.4532e+06
10 sec	5.2611e+06	4.7877e+06	-2.4842e+06
13.263 sec	5.2499e+06	4.7888e+06	-2.5045e+06
15 sec	5.2439e+06	4.7894e+06	-2.5152e+06
:	:	:	:
5380 sec	6.3454e+06	3.4735e+06	2.1279e+06
5385 sec	6.3479e+06	3.4893e+06	2.0962e+06
5390 sec	6.3503e+06	3.5051e+06	2.0645e+06
5395 sec	6.3526e+06	3.5207e+06	2.0327e+06
5400 sec	6.3547e+06	3.5363e+06	2.0009e+06

Display all 1148 rows.

```
display(mission.Data.AttitudeTimeTableGCRF2Body)
```

1148x1 timetable

Time	Data			
0 sec	0.1509	0.48681	0.30311	-0.80522
5 sec	0.15061	0.48761	0.3033	-0.80472
10 sec	0.15003	0.48914	0.30368	-0.80375
13.263 sec	0.14977	0.48986	0.30387	-0.80329
15 sec	0.14967	0.49013	0.30395	-0.80311
:	:	:	:	:
5380 sec	-0.043839	-0.72806	-0.33468	0.59666
5385 sec	-0.04461	-0.72663	-0.3346	0.59838
5390 sec	-0.04538	-0.72521	-0.33451	0.60009
5395 sec	-0.046149	-0.72378	-0.33443	0.60181
5400 sec	-0.046919	-0.72235	-0.33434	0.60351

Display all 1148 rows.

Create the Satellite Scenario

Create a satellite scenario object to use for analysis. Specify a timestep of 1 minute.

```
scenario = satelliteScenario(mission.StartDate, ...
    mission.StartDate + mission.Duration, 60);
```

Add the two targets as ground stations in Brazil and Svalbard.

```
gsNO = groundStation(scenario, 78, 21, Name="Svalbard, NO")
```

gsNO =

GroundStation with properties:

```

    Name: Svalbard, NO
    ID: 1
    Latitude: 78 degrees
    Longitude: 21 degrees
    Altitude: 0 meters
    MinElevationAngle: 0 degrees
    ConicalSensors: [1x0 matlabshared.satellitescenario.ConicalSensor]
    Gimbals: [1x0 matlabshared.satellitescenario.Gimbal]
    Transmitters: [1x0 satcom.satellitescenario.Transmitter]
    Receivers: [1x0 satcom.satellitescenario.Receiver]
    Accesses: [1x0 matlabshared.satellitescenario.Access]
    MarkerColor: [1 0.4118 0.1608]
    MarkerSize: 6
    ShowLabel: true
    LabelFontColor: [1 1 1]
    LabelFontSize: 15
```

```
gsAmazon = groundStation(scenario, -4.9, -66, Name="Amazon Rainforest")
```

gsAmazon =

GroundStation with properties:

```

    Name: Amazon Rainforest
```

```
        ID: 2
        Latitude: -4.9 degrees
        Longitude: -66 degrees
        Altitude: 0 meters
MinElevationAngle: 0 degrees
ConicalSensors: [1x0 matlabshared.satellitescenario.ConicalSensor]
        Gimbals: [1x0 matlabshared.satellitescenario.Gimbal]
        Transmitters: [1x0 satcom.satellitescenario.Transmitter]
        Receivers: [1x0 satcom.satellitescenario.Receiver]
        Accesses: [1x0 matlabshared.satellitescenario.Access]
        MarkerColor: [1 0.4118 0.1608]
        MarkerSize: 6
        ShowLabel: true
LabelFontColor: [1 1 1]
LabelFontSize: 15
```

Add the Satellite From the Loaded Trajectory

Add the observation satellite to the scenario.

```
sat = satellite(scenario, mission.Data.PositionTimeTableGCRF, ...
    "CoordinateFrame", "inertial", "Name", "ObservationSat");
```

Add a conical sensor to the satellite, with a 35 deg half angle to represent the onboard camera. Enable field of view visualization in the scenario viewer. To assist in visualization, the sensor is mounted 10m from the satellite in the +z direction.

```
snsr = conicalSensor(sat, MaxViewAngle=70, MountingLocation=[0 0 10]);
fieldOfView(snsr);
```

Add access between the conical sensor and the two ground stations.

```
acNO = access(snsr, gsNO)
```

```
acNO =
    Access with properties:
        Sequence: [4 1]
        LineWidth: 3
        LineColor: [0.3922 0.8314 0.0745]
```

```
acAmazon = access(snsr, gsAmazon)
```

```
acAmazon =
    Access with properties:
        Sequence: [4 2]
        LineWidth: 3
        LineColor: [0.3922 0.8314 0.0745]
```

Point the Satellite With the Loaded Attitude Profile

Use the `pointAt` method to associate the logged attitude `timetable` with the satellite. Parameter `ExtrapolationMethod` controls the pointing behavior outside of the `timetable` range.

```
pointAt(sat, mission.Data.AttitudeTimeTableGCRF2Body, ...  
        "CoordinateFrame", "inertial", "Format", "quaternion", "ExtrapolationMethod", "nadir");
```

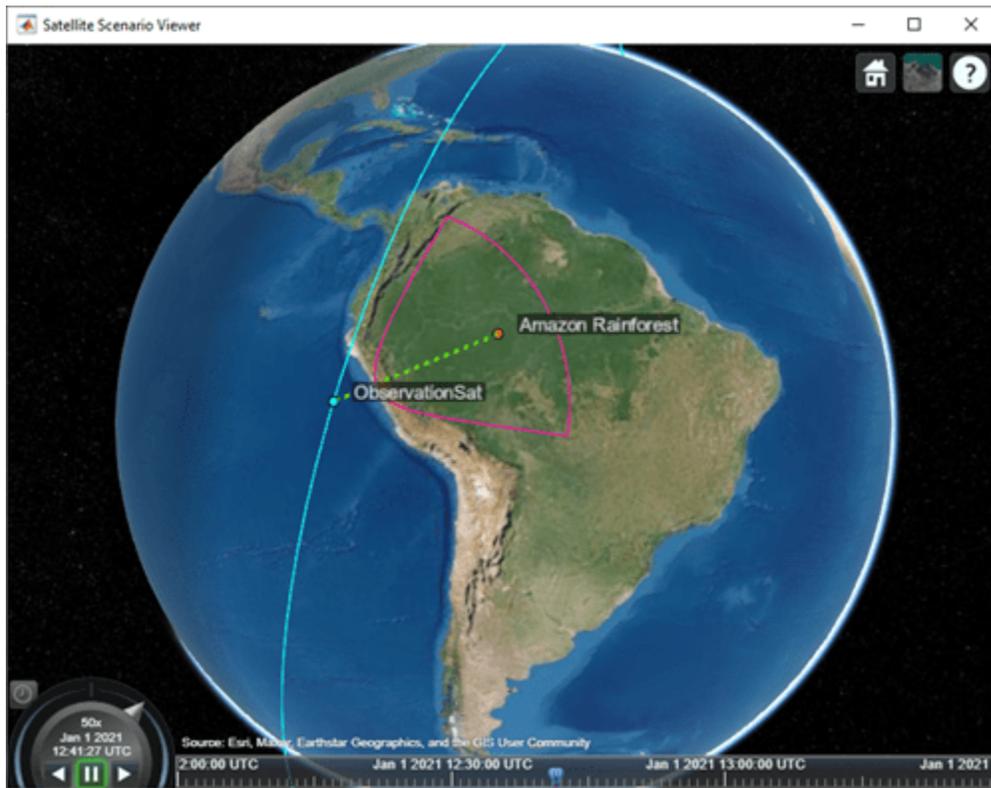
Visualize the Scenario

Open the **Satellite Scenario Viewer** to view and interact with the scenario.

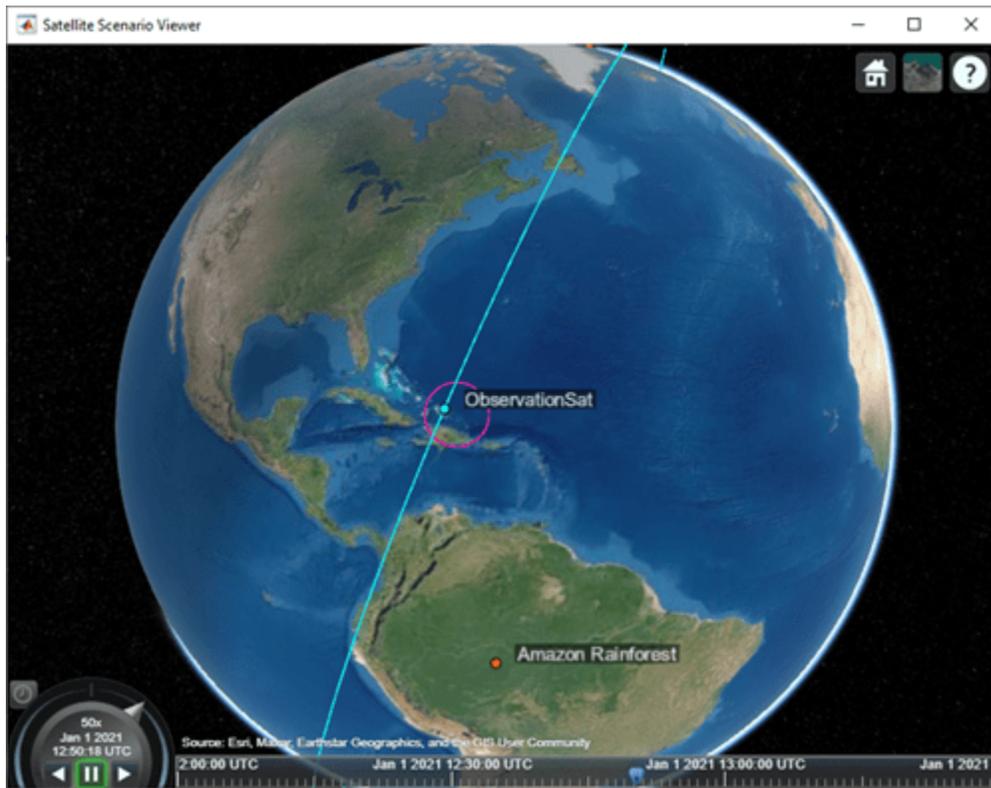
```
viewer1 = satelliteScenarioViewer(scenario);
```



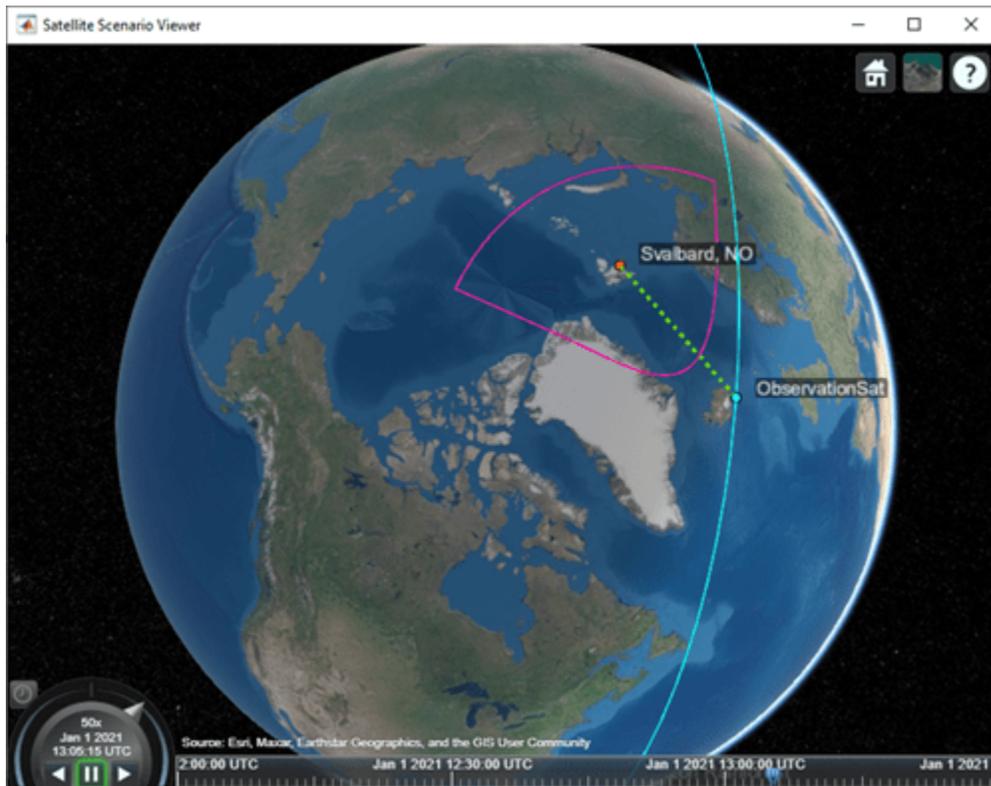
The satellite points at the nadir to begin the scenario. As it nears Target 1 in the Amazon Rainforest, it slews to point and track this target.



After the imaging segment is complete, the satellite returns to pointing at the nadir.



As the satellite comes into range of the arctic ground station, it slews to point at this target.



Custom Gimbal Steering

This example shows how to import custom attitude data for a simple Earth Observation satellite mission, where the onboard camera is fixed to the satellite body. Another common approach is to fix the sensor on a gimbal and orient the sensor by maneuvering the gimbal, rather than the spacecraft body itself. Modify the above scenario to mount the sensor on a gimbal and steer the gimbal to perform uniform sweeps of the area directly below the satellite.

Reset the satellite to always point at the nadir, overwriting the previously provided custom attitude profile.

```
delete(viewer1);
pointAt(sat, "nadir");
```

Delete the existing sensor object to remove it from the satellite and attach a new sensor with the same properties to a gimbal.

```
delete(snsr);
gim = gimbal(sat);
snsr = conicalSensor(gim, MaxViewAngle=70, MountingLocation=[0 0 10]);
fieldOfView(snsr);
```

Define azimuth and elevation angles for gimbal steering to model a sweeping pattern over time below the satellite.

```
gimbalSweep.Time = seconds(1:50:5000)';
```

```
gimbalSweep.Az = [...
    45*ones(1,7),...]
```

```

45:-5:-45,...
-45*ones(1,13),...
-45:5:45,...
45*ones(1,13),...
45:-5:-45,...
-45*ones(1,13)];
gimbalSweep.Az(end-2:end) = [];
gimbalSweep.Az = gimbalSweep.Az + 90;

gimbalSweep.El = [...
0:-5:-30,...
-30*ones(1,19),...
-30:5:30,...
30*ones(1,19),...
30:-5:-30,...
-30*ones(1,19),...
-30:5:30];
gimbalSweep.El(end-2:end) = [];

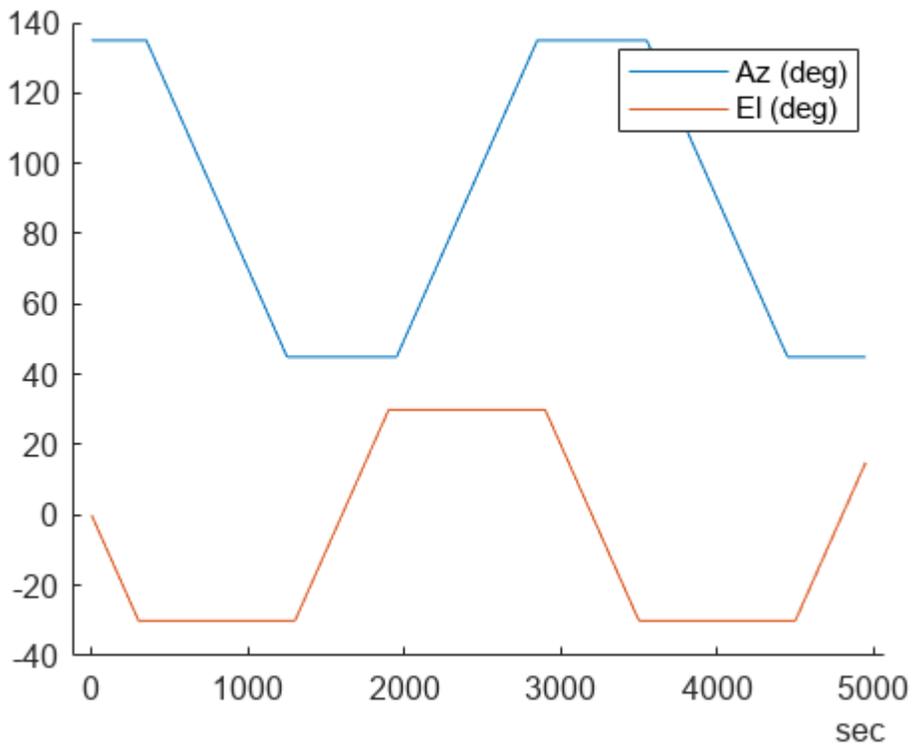
```

Plot the commanded azimuth and elevation values over time.

```

figure(1)
hold on;
plot(gimbalSweep.Time', gimbalSweep.Az);
plot(gimbalSweep.Time', gimbalSweep.El);
hold off;
legend(["Az (deg)", "El (deg)"]);

```



Store the azimuth and elevation angles in a timetable.

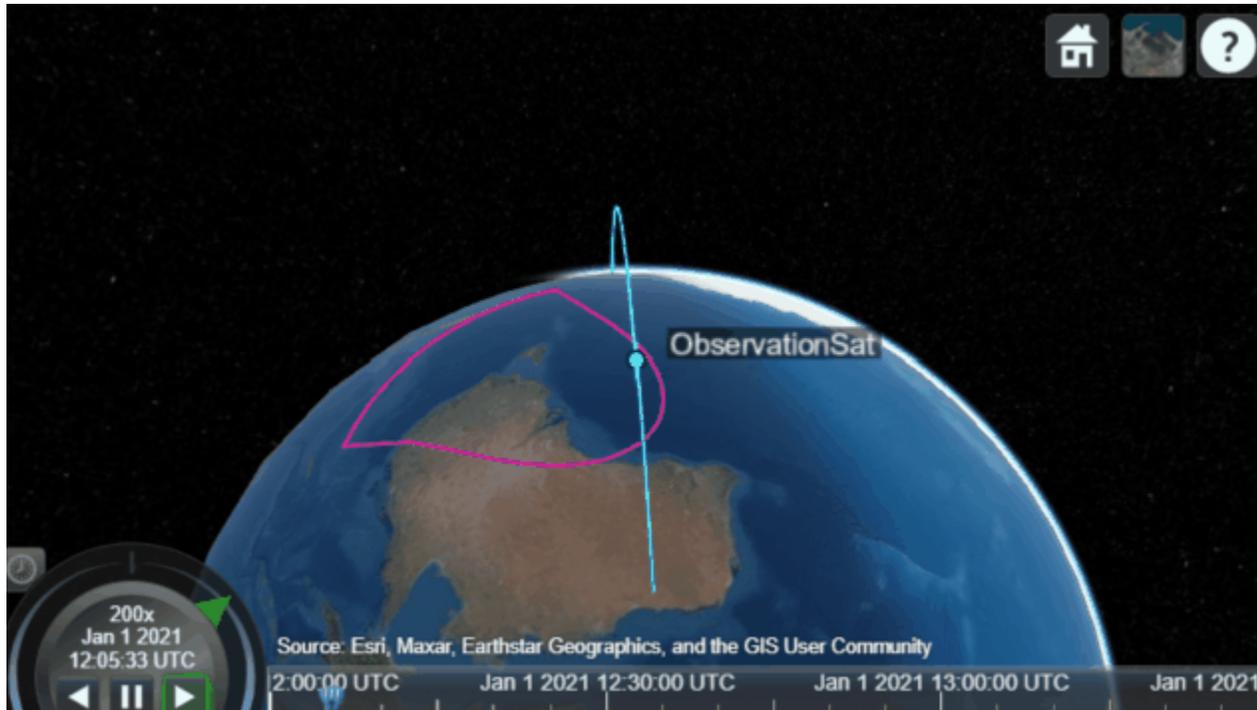
```
gimbalSweep.TT = timetable(gimbalSweep.Time, [gimbalSweep.Az', gimbalSweep.El']);
```

Steer the gimbal with the `timetable`. The gimbal returns to its default orientation for timesteps that are outside of the provided data.

```
pointAt(gim, gimbalSweep.TT);
```

View the updated scenario in the **Satellite Scenario Viewer**.

```
viewer2 = satelliteScenarioViewer(scenario);
```



See Also

Objects

[satelliteScenario](#) | [satelliteScenarioViewer](#) | [Satellite](#) | [GroundStation](#) | [ConicalSensor](#) | [Gimbal](#) | [Access](#) | [FieldOfView](#)

Functions

[pointAt](#)

Related Examples

- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-33
- “Satellite Constellation Access to Ground Station” on page 1-14

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Signal Transmission

GPS Waveform Generation

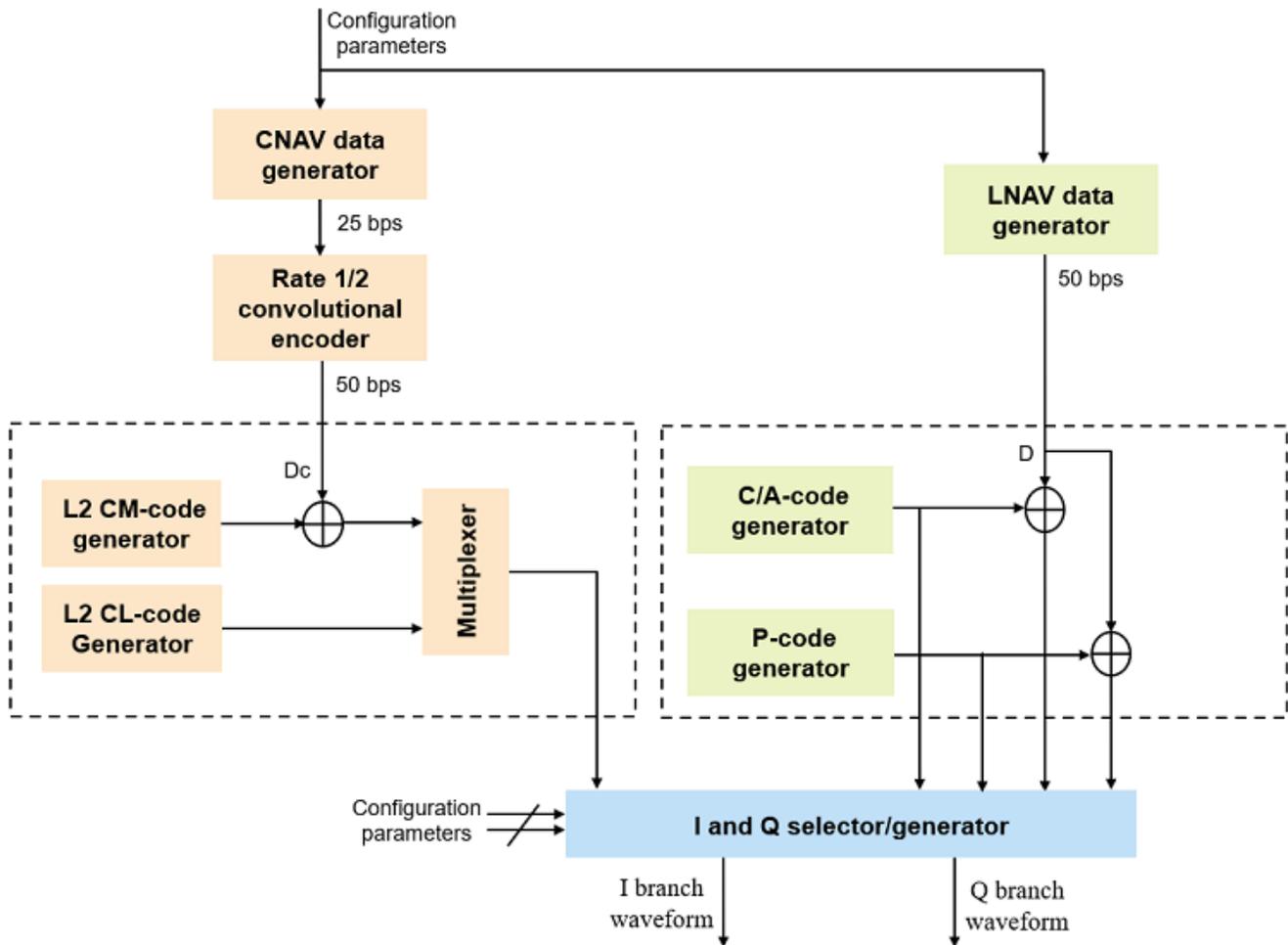
This example shows how to generate Global Positioning System (GPS) legacy navigation (LNAV) data, civil navigation (CNAV) data, and a complex baseband waveform. The spreading of the data is performed with coarse acquisition code (C/A-code), precision code (P-code), or civil moderate / civil long code (L2 CM-/L2 CL-code). This example shows GPS waveform generation according to the IS-GPS-200L standard [1] on page 2-16. To design a navigation system based on GPS, you must test the receiver with a received signal. Because you cannot control transmitter and channel parameters, a signal that is received from a satellite is not useful for testing a receiver. To test the receiver, you must use a waveform that is generated under a controlled set of parameters.

Introduction

Generate a GPS signal by using these three steps.

- 1** Generate GPS data bits by using the configuration parameters that are described in later sections. The data bits are generated at a rate of 50 bits per second (bps).
- 2** Spread these low rate data bits by using high rate spreading codes. The GPS standard [1] on page 2-16 specifies three kinds of spreading codes: C/A-code, P-code, and L2 CM-/L2 CL-code. In addition to these three codes, this standard also specifies Y-code to use instead of P-code when anti-spoofing mode of an operation is active. P-code and Y-code together are called P(Y)-code. The configuration parameters determine the spreading code that is used to generate the waveform.
- 3** Generate the GPS complex baseband waveform from the bits that are spread by the spreading codes by selecting codes on the in-phase branch and quadrature-phase branch according to the set of configuration parameters.

This figure shows the block diagram of the GPS waveform generator from the configuration parameters.



GPS Signal Structure

The standard [1] on page 2-16 describes the transmission of GPS signals on two frequencies: L1 (1575.42 MHz) and L2 (1227.60 MHz). A signal of base frequency 10.23 MHz generates both of these signals. The signal frequency of L1 is $154 \times 10.23 \text{ MHz} = 1575.42 \text{ MHz}$, and the signal frequency of L2 is $120 \times 10.23 \text{ MHz} = 1227.60 \text{ MHz}$. The code-division-multiple-access technique enables you to differentiate between the satellites even though all GPS satellites transmit on the same frequency.

With the evolution of GPS, the signal structure is expanded to improve the navigation performance. For instance, LNAV data is transmitted on the L1-band from the inception of GPS. From GPS block IIR-M onwards, CNAV data is transmitted on the L2-band in addition to the LNAV data that exists on both L1 and L2 bands. This table shows these signal configurations and GPS evolution. "Dc" represents the CNAV data bits and "D" represents the LNAV data bits. Bit operator, \oplus , represents the XOR operation.

SV Blocks	L1		L2	
	In-phase	Quadrature-phase	In-phase	Quadrature-phase
Block II/IIA/IIR	$P(Y) \oplus D$	$C/A \oplus D$	$P(Y) \oplus D$ Or $P(Y)$ Or $C/A \oplus D$	None
Block IIR-M/IIF and GPS III/IIIF	$P(Y) \oplus D$	$C/A \oplus D$	$P(Y) \oplus D$ Or $P(Y)$	$L2\ C/M \oplus D_c$ with $L2\ C/L$ Or $C/A \oplus D$ Or C/A

Using this table to choose the content that needs to be transmitted in the in-phase and quadrature-phase branches.

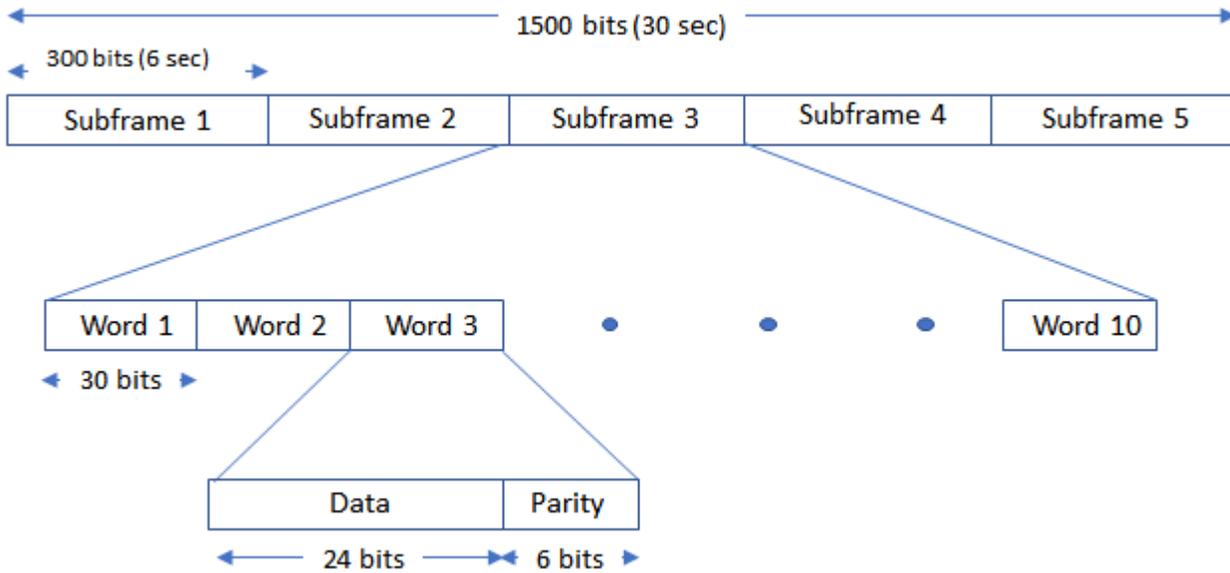
IBranchContent =

IBranchContent =
"P(Y) + D"

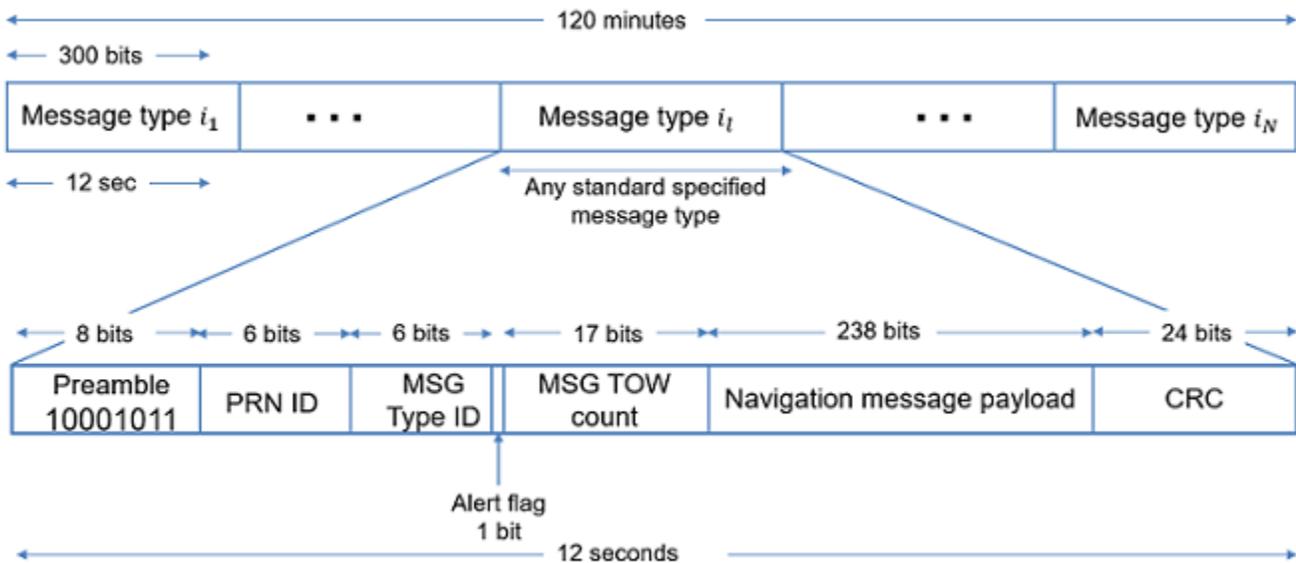
QBranchContent =

QBranchContent =
"C/A + D"

Many of the transmitted properties in the LNAV and CNAV data are same, but the frame structure is different. The LNAV data is transmitted in 1500 bit-length frames, with each frame consisting of five subframes of 300 bits in each subframe. Because the data rate is 50 bps, transmitting each subframe takes 6 seconds and transmitting each frame takes 30 seconds. Each subframe consists of 10 words with 30 bits (24 data bits and 6 parity bits) in each word. The GPS data contains information regarding the clock and the position of the satellites. This figure shows the frame structure of the LNAV data.



The CNAV data is transmitted continuously in the form of *message types*. Each message type consists of 300 bits that are transmitted at 25 bps. These bits are passed through a rate-half convolutional-encoder to obtain 600 bits from each message type at 50 bps. Transmitting each message type takes 12 seconds. The standard [1] on page 2-16 defines 14 message types in this order: 10, 11, 30, 31, 32, 33, 34, 35, 36, 37, 12, 13, 14, and 15. For a detailed description of each message type and the data transmitted, see IS-GPS-200L Appendix III [1] on page 2-16. The order in which each message type is transmitted is completely arbitrary but is sequenced to provide optimal user experience. In this example, you can choose the order in which these message types are transmitted. This figure shows the CNAV message structure.



The message type shown in this figure has these fields:

- PRN ID: Pseudo-random noise (PRN) index

- MSG: Message
- TOW: Time of week
- CRC: Cyclic redundancy check

Set the `ShowVisualizations` property to enable the spectrum and correlation plot visualizations. Set the `WriteWaveformToFile` property to write the complex baseband waveform to a file if needed. This example enables visualizations and disables writing the waveform to a file.

```
ShowVisualizations = ;
```

```
WriteWaveformToFile = .
```

Specify the satellite PRN index as an integer in the range [1,63].

```
PRNID = 1;
```

Because generating the GPS waveform for the entire navigation data can take a lot of time and memory, this example demonstrates generating a waveform for only one bit of the navigation data. You can control generating the waveform for a specified number of data bits by using the property `NumNavDataBits`.

```
% Set this value to 1 to generate the waveform from the first bit of the  
% navigation data  
NavDataBitStartIndex = 1321;
```

```
% Set this value to control the number of navigation data bits in the  
% generated waveform  
NumNavDataBits = 1;
```

GPS Data Initialization

Initialize the data configuration object to generate the CNAV data. You can create a configuration object from the `HelperGPSNavigationConfig` object. Update the configuration object properties to customize the waveform as required.

```
cnavConfig = HelperGPSNavigationConfig("DataType","CNAV","PRNID",PRNID)
```

```
cnavConfig =
```

```
HelperGPSNavigationConfig with properties:
```

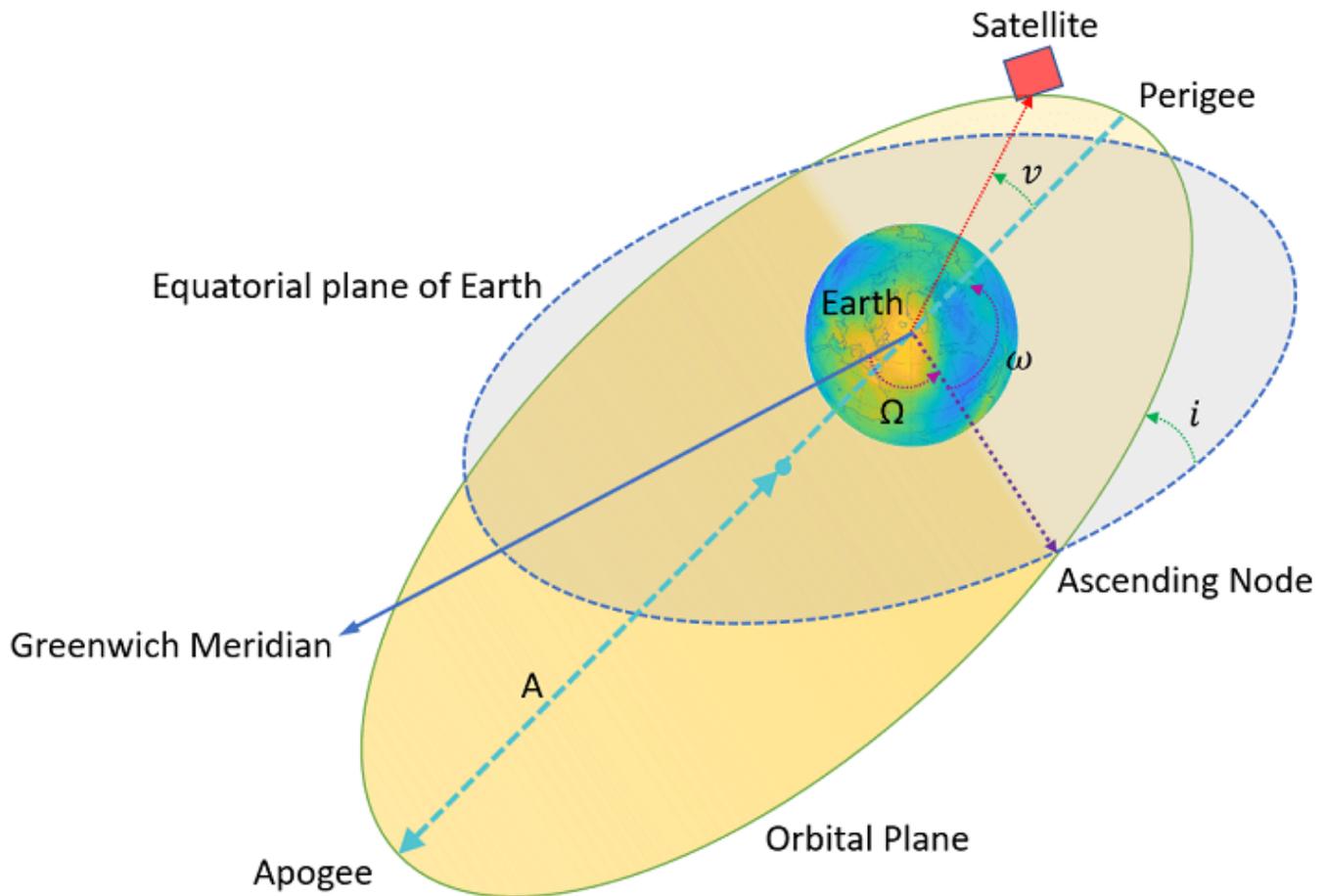
```
        DataType: "CNAV"  
        PRNID: 1  
        MessageTypes: [4x15 double]  
        Preamble: 139  
        HOWTOW: 1  
        L2CPhasing: 0  
        CEIDDataSet: [1x1 HelperGPSCEIConfig]  
        AgeOfDataOffset: 0  
        AlmanacFileName: "gpsAlmanac.txt"  
        Ionosphere: [1x1 struct]  
        EarthOrientation: [1x1 struct]  
        UTC: [1x1 struct]  
        DifferentialCorrection: [1x1 struct]  
        TimeOffset: [1x1 struct]  
        TextInMessageType36: 'This content is part of Satellite Communications Toolbox'  
        TextInMessageType15: 'This content is part of Satellite Communications Toolbox'
```

The property, `CEIDataSet`, is a configuration object of type `HelperGPSCEIConfig`, which contains clock, ephemeris, and integrity (CEI) data parameters.

In theory, clocks of all satellites must be synchronized, which implies that all GPS satellite clocks must show the same time at a given moment in time. In practice, deterministic satellite clock error characteristics of bias, drift, and aging as well as satellite implementation characteristics of group delay bias and mean differential group delay exist. These errors deviate the satellite clocks from the GPS system time.

GPS satellites revolve around the Earth in an elliptical orbit, with Earth at one of the focal points of the ellipse. A set of orbital parameters that accurately defines a satellite position in this elliptical orbit is called *ephemeris*. Each GPS satellite transmits its own ephemeris data on subframes 2 and 3 for LNAV data and on message types 10 and 11 for CNAV data. This figure shows five orbital parameters for a satellite in the Earth-centered Earth-fixed (ECEF) coordinate system. This figure does not show an actual GPS satellite, and the figure is for illustration only.

- Length of semimajor axis, A : Distance from the center of the elliptical orbit of the satellite to the apogee or perigee
- Inclination angle, i : Angle between the equatorial plane of Earth and the plane of the orbit of the satellite
- Longitude of ascending node, Ω : Angle between the Greenwich meridian and the direction of the ascending node
- Argument of perigee, ω : Angle between the direction of the ascending node and the direction of the perigee
- True anomaly, v : Angle between the direction of the perigee and the direction of the current position of the satellite



Per Kepler's second law of planetary motion, the angular velocity (rate of change of true anomaly) is different at different locations in the orbit. You can define the mean anomaly, whose rate of change is constant over the entire orbit of the satellite. In GPS ephemeris parameters, rather than specifying true anomaly, mean anomaly is specified (from which true anomaly can be found). IS-GPS-200L Table 20-IV [1] on page 2-16 specifies the algorithms that relate mean anomaly and true anomaly.

The eccentricity of the ellipse also defines the orbit. Eccentricity gives a measure of deviation of the elliptical orbit from the circular shape.

This command displays the CEI data set properties that are related to the CNAV data.

```
cnvConfig.CEIDataSet
```

```
ans =
```

```
  HelperGPSCEIConfig with properties:
```

```

        SignalHealth: [3x1 double]
        WeekNumber: 2149
        GroupDelayDifferential: 0
    SVClockCorrectionCoefficients: [3x1 double]
        ReferenceTimeOfClock: 0
        SemiMajorAxisLength: 26560
        ChangeRateInSemiMajorAxis: 0
        MeanMotionDifference: 0
```

```

RateOfMeanMotionDifference: 0
    Eccentricity: 0.0200
    MeanAnomaly: 0
ReferenceTimeOfEphemeris: 0
    HarmonicCorrectionTerms: [6x1 double]
    IntegrityStatusFlag: 0
    ArgumentOfPerigee: -0.5200
    RateOfRightAscension: 0
LongitudeOfAscendingNode: -0.8400
    Inclination: 0.3000
    InclinationRate: 0
    URAEDID: 0
    InterSignalCorrection: [4x1 double]
ReferenceTimeCEIPropagation: 0
ReferenceWeekNumberCEIPropagation: 101
    URANEDID: [3x1 double]
AlertFlag: 0

```

A similar set of properties exist for the LNAV data. Create a configuration object to store the LNAV data.

```
lnavConfig = HelperGPSNavigationConfig("DataType", "LNAV", "PRNID", PRNID)
```

```
lnavConfig =
```

```
    HelperGPSNavigationConfig with properties:
```

```

        DataType: "LNAV"
        PRNID: 1
        FrameIndices: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ... ]
        Preamble: 139
        TLMMessage: 0
        HOWTOW: 1
        AntiSpoofFlag: 0
        CodesOnL2: "P-code"
        L2PDDataFlag: 0
        CEIDataSet: [1x1 HelperGPSCEIConfig]
        AgeOfDataOffset: 0
        NMCTAvailabilityIndicator: 0
        NMCTERD: [30x1 double]
        AlmanacFileName: "gpsAlmanac.txt"
        Ionosphere: [1x1 struct]
        UTC: [1x1 struct]
        TextMessage: 'This content is part of Satellite Communications Toolbox'

```

Similar to the CNAV data, the LNAV data has CEI data properties that are different from that of CNAV. This command displays the LNAV CEI data properties.

```
lnavConfig.CEIDataSet
```

```
ans =
```

```
    HelperGPSCEIConfig with properties:
```

```

        SVHealth: 0
        IssueOfDataClock: 0
        URAID: 0
        WeekNumber: 2149

```

```

        GroupDelayDifferential: 0
    SVClockCorrectionCoefficients: [3x1 double]
        ReferenceTimeOfClock: 0
        SemiMajorAxisLength: 26560
        MeanMotionDifference: 0
            FitIntervalFlag: 0
                Eccentricity: 0.0200
                MeanAnomaly: 0
        ReferenceTimeOfEphemeris: 0
        HarmonicCorrectionTerms: [6x1 double]
        IssueOfDataEphemeris: 0
        IntegrityStatusFlag: 0
            ArgumentOfPerigee: -0.5200
            RateOfRightAscension: 0
        LongitudeOfAscendingNode: -0.8400
            Inclination: 0.3000
            InclinationRate: 0
            AlertFlag: 0
    
```

GPS Signal Generation

To generate a GPS signal at the baseband, follow these steps.

- 1 Generate the navigation data bits at 50 bits per second.
- 2 Based on the configuration, generate C/A-code, P-code, L2 CM-/L2 CL-code, or a combination.
- 3 Spread the CNAV or LNAV data bit with the appropriate ranging codes.
- 4 Collect data for the in-phase branch and quadrature-phase branch by rate-matching the codes in each branch.
- 5 Map the bits on both of the branches as bit 0 to +1 and bit 1 to -1.
- 6 (Optional) Write this baseband waveform to a file (depending on the WriteWaveformToFile property value).

Based on the configuration, generate the CNAV data.

```
cnavData = HelperGPSNAVDDataEncode(cnavConfig);
```

Pass the CNAV data through the convolutional encoder.

```

% Initialize the trellis for convolutional encoder
trellis = poly2trellis(7,{'1+x+x^2+x^3+x^6', '1+x^2+x^3+x^5+x^6'});
cenc = comm.ConvolutionalEncoder('TrellisStructure',trellis, ...
    "TerminationMethod", "Continuous");
encodedCNAVData = cenc(cnavData);
    
```

Based on the configuration, generate the LNAV data.

```
lnavData = HelperGPSNAVDDataEncode(lnavConfig);
```

Specify all of the required properties for waveform generation.

```

CLCodeResetIdx = 75; % CL-code spans over 75 data bits before resetting
numBBSamplesPerDataBit = 204600;
CLCodeIdx = mod(NavDataBitStartIndex-1,CLCodeResetIdx);
IQContent = [IBranchContent,QBranchContent];
pgen = gpsPCCode("PRNID",PRNID,"InitialTime", ...
    
```

```

    lnavConfig.CEIDataSet.ReferenceTimeOfEphemeris, ...
    "OutputCodeLength", numBBSamplesPerDataBit);
% Pre-initialize the baseband waveform for speed
gpsBBWaveform = zeros(numBBSamplesPerDataBit*NumNavDataBits,1);

```

Create a file into which the waveform is written.

```

if WriteWaveformToFile == 1
    bbWriter = comm.BasebandFileWriter('Waveform.bb',10.23e6,0);
end

```

Independently process each navigation data bit in a loop.

```

for iDataBit = 1:NumNavDataBits
    dataBitIdx = iDataBit+NavDataBitStartIndex-1;
    bbSamplesIndices = ((iDataBit-1)*numBBSamplesPerDataBit+1): ...
        (iDataBit*numBBSamplesPerDataBit);
    gpsBBWaveform(bbSamplesIndices) = HelperGPSBasebandWaveform(IQContent,pgen,PRNID, ...
        CLCodeIdx,lnavData(dataBitIdx),encodedCNAVData(dataBitIdx));
    CLCodeIdx = mod(CLCodeIdx+1,CLCodeResetIdx);
    if WriteWaveformToFile == 1
        bbWriter(gpsBBWaveform(bbSamplesIndices));
    end
end

```

Close the file if it is opened.

```

if WriteWaveformToFile == 1
    release(bbWriter);
end

```

Signal Visualization

Plot autocorrelation of the C/A-code and visualize the spectrum of the GPS signals.

```

if ShowVisualizations

```

Autocorrelation of the ranging code sequence is near-zero except at zero delay, and crosscorrelation of two different sequences is near-zero. Because the C/A-code is periodic with a period of 1023 bits, autocorrelation has a peak for a delay of every 1023 bits. Calculate and plot the autocorrelation of GPS spreading code.

```

% Because P-code is 10 times faster than C/A-code or L2 CM-/L2 CL-code,
% initialise down sample factor to 10
downsampleFactor = 10;
IBranchData = real(gpsBBWaveform);
QBranchData = imag(gpsBBWaveform(1:downsampleFactor:end));
lags = (-1023:1023).';
plot(lags,xcorr(real(QBranchData(1:1023)),1023))
grid on
xlabel('Number of Samples Delayed')
ylabel('Autocorrelation Value')
title('Autocorrelation of GPS Spreading Code')

```

Compare the power spectral density of in-phase branch and quadrature-branch signals. This spectrum plot shows that the P-code is wider when P-code is used.

```

repeatFactor = 40;
% Repeat the generated BPSK signal of C/A-code to see the adjacent bands spectrum

```

```

QBranchUpsampled = repmat(QBranchData(:).',repeatFactor,1);
QBranchUpsampled = QBranchUpsampled(:);
% Repeat the generated BPSK signal of in-phase component to see the
% adjacent bands spectrum. Repeat the in-phase branch samples ten times less
% as every sample in quadrature-branch corresponds to 10 samples in in-phase branch
IBranchUpsampled = repmat(IBranchData(:).',repeatFactor/10,1);
IBranchUpsampled = real(IBranchUpsampled(:));
iqScope = dsp.SpectrumAnalyzer('SampleRate',1.023e6*repeatFactor, ...
    'PlotAsTwoSidedSpectrum',true, ...
    'SpectrumType','Power density', ...
    'AveragingMethod','Exponential', ...
    'SpectrumUnits','dBW', ...
    'YLimits',[-130, -50],'Title', ...
    'Comparison of Power Spectral Density of GPS baseband I and Q Signals', ...
    'ShowLegend',true,'ChannelNames', ...
    {'Q-branch spectrum with content: ' char(QBranchContent)}, ...
    {'I-branch spectrum with content: ' char(IBranchContent)});

iqScope([QBranchUpsampled,IBranchUpsampled]);

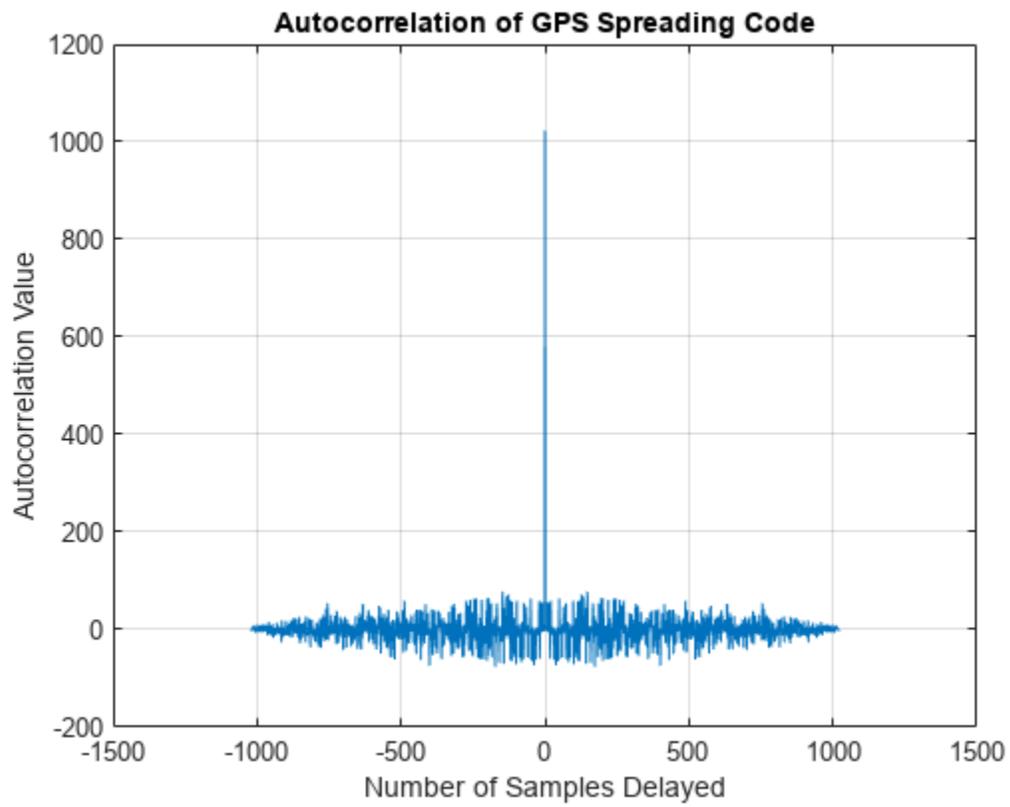
```

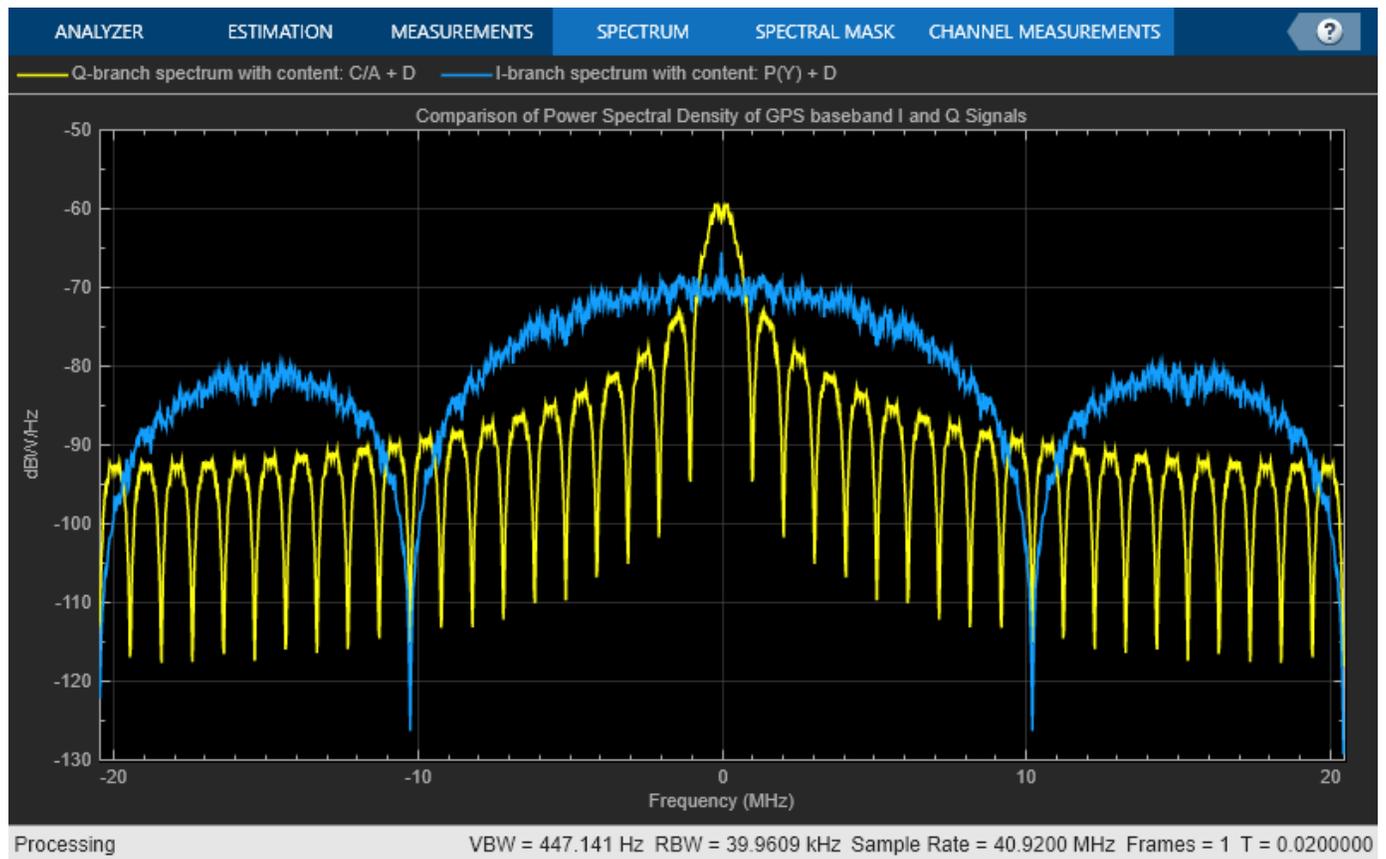
Plot the signal power spectral density at the baseband. To observe the adjacent band spectrum for the GPS signal, repeat the signal at the baseband.

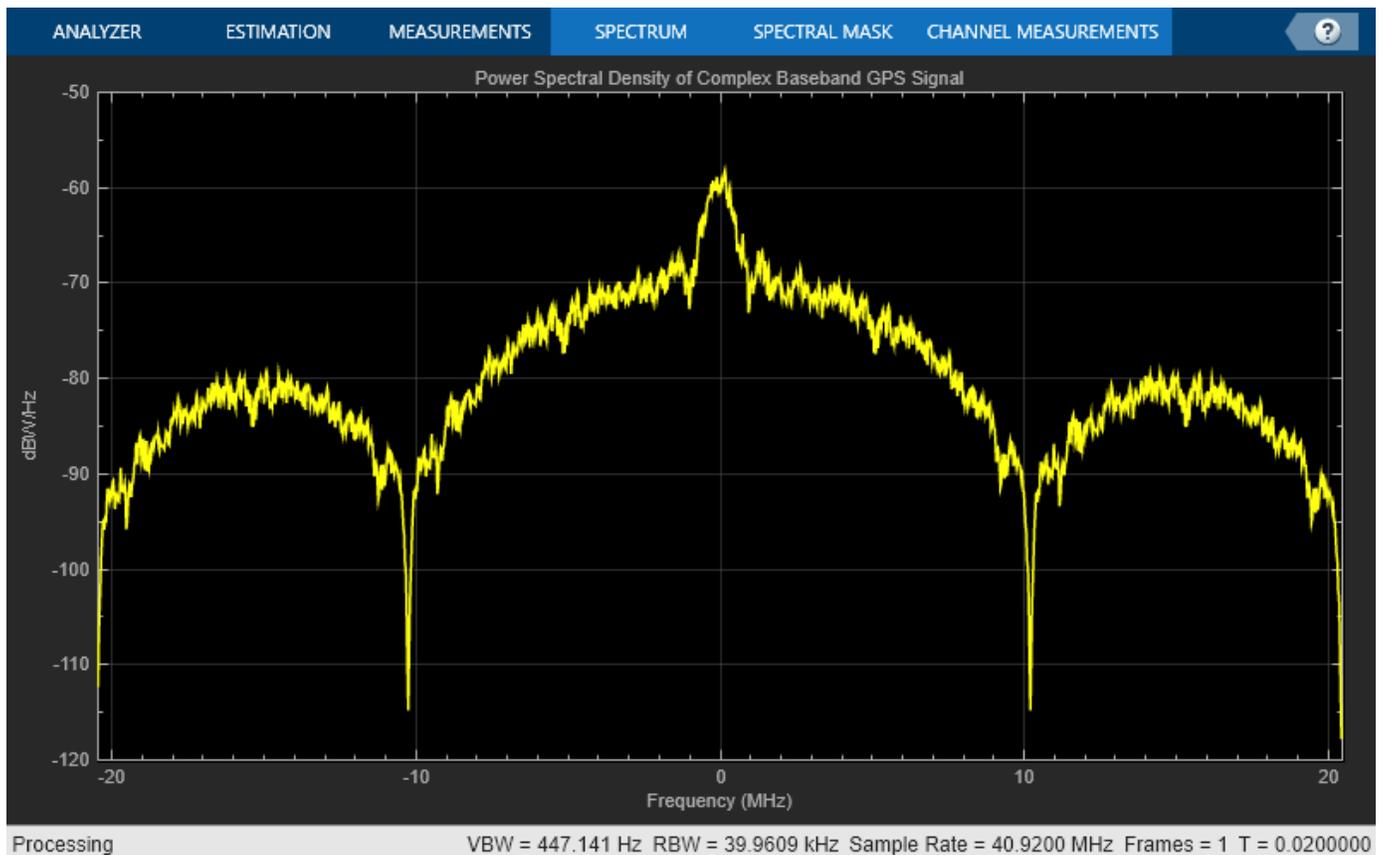
```

repeatFactor = 4;
% Repeat the generated BPSK signal to see the adjacent bands spectrum
update = repmat(gpsBBWaveform(:).',repeatFactor,1);
update = update(:);
bbscope = dsp.SpectrumAnalyzer('SampleRate',10*1.023e6*repeatFactor, ...
    'PlotAsTwoSidedSpectrum',true, ...
    'SpectrumType','Power density', ...
    'AveragingMethod','Exponential', ...
    'SpectrumUnits','dBW', ...
    'YLimits',[-120,-50], ...
    'Title','Power Spectral Density of Complex Baseband GPS Signal');
bbscope(update);
end

```







Further Exploration

This example uses a configuration object to generate GPS data bits and the navigation signal in the baseband. You can replace the property values in this configuration object and observe how the GPS data is generated. You can also change the ephemeris parameters with an existing real data set and pass those parameters into the CEI data set. Additionally, you can specify your own almanac file. If you are using your own almanac file, the week number in the almanac file and the week number in the configuration object must match.

Further, this example shows how to generate a GPS baseband waveform, which can be extended to generate an intermediate frequency (IF) waveform from the baseband waveform by multiplying a cosine signal on the in-phase branch and a sine signal on the quadrature-branch.

Additionally, this example shows how to generate a GPS waveform from one satellite, which can be combined along with multiple satellite PRN codes to get an integrated signal.

Appendix

This example uses these data and helper files:

- `gpsAlmanac.txt` — Almanac data file downloaded from Navcen website
- `HelperGPSAlmanac2Struct.m` — Convert text file of almanac to structure
- `HelperGPSBasebandWaveform.m` — Create GPS baseband waveform from data bits
- `HelperGPSCEIConfig.m` — Create configuration object for GPS navigation data

- `HelperGPSL2CRangingCode.m` — Generate L2 CM-/L2 CL-code
- `HelperGPSNAVDDataEncode.m` — Encode navigation data into bits from data that is in configuration object
- `HelperGPSNavigationConfig.m` — Create configuration object for GPS navigation data

Bibliography

[1] IS-GPS-200, Rev: L. *NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*. May 14, 2020; Code Ident: 66RP1.

See Also

`gnssCACode` | `gpsPCode`

Related Examples

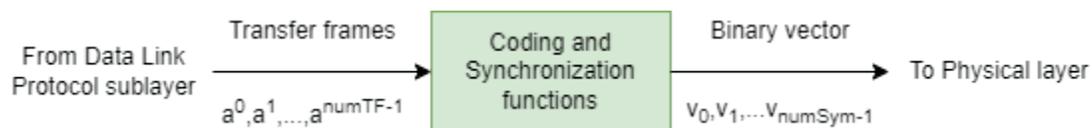
- “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118
- “GPS Data Decode” on page 4-139

CCSDS Optical High Photon Efficiency Telemetry Waveform Generation

This example shows how to generate a high photon efficiency (HPE) waveform for the free space optical communications systems defined in Consultative Committee for Space Data Systems (CCSDS) 142.0-B-1 section 3 [1] on page 2-23. HPE optical communications are required for applications in which power efficiency is the dominant consideration in link design. The example describes the functions of the channel coding and synchronization sublayer of the data link layer for transferring the transfer frames over an optical photon-starved space link. In these photon-starved links, the photon-efficiency of the link is the primary concern. This example shows how to generate a binary vector to be provided to the physical layer, given a set of CCSDS transfer frames produced by the data link protocol sublayer. The binary vector indicates the positions of the pulsed slots. The 1s and 0s of the resultant binary vector indicate pulsed and non-pulsed slots, respectively, in the optical transmission.

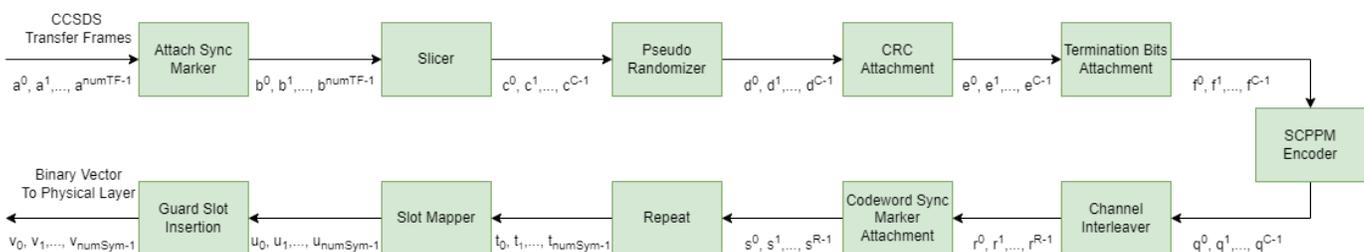
Introduction

This figure shows the internal organization of the coding and synchronization sublayer of telemetry (TM) signaling at the transmitting end. The coding and synchronization sublayer accepts CCSDS transfer frames of constant code rate and data length from the data link protocol sublayer, performs the necessary functions, and delivers a binary vector to the physical layer to indicate the slots containing light pulses. The input is a sequence of fixed-length transfer frames generated from the CCSDS TM Space Data Link Protocol defined in CCSDS 132.0-B-2 [3] on page 2-23, or the CCSDS Advanced Orbiting Systems (AOS) Protocol defined in CCSDS 732.0-B-3 [4] on page 2-23, or the CCSDS Universal Space Link Protocol (USLP) defined in CCSDS 732.1-B-1 [5] on page 2-24.



HPE Telemetry Signal Generation

This figure shows the functional blocks of the architecture of HPE TM signaling at the transmitting end performed by the coding and synchronization sublayer.

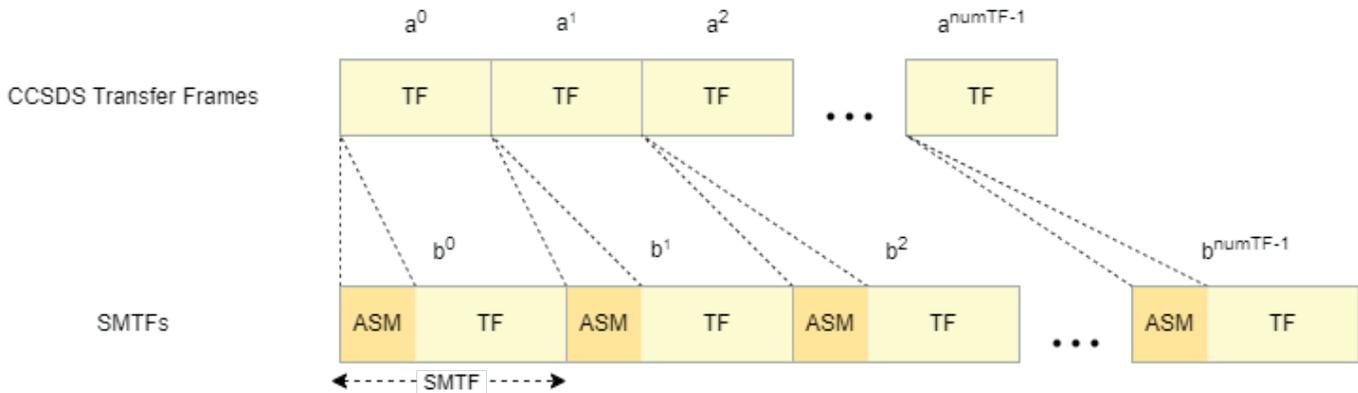


The input to the coding and synchronization sublayer is a sequence of CCSDS TM, AOS, or USLP transfer frames. Consider CCSDS transfer frames $a^0, a^1, \dots, a^{\text{numTF}-1}$, where numTF is the number of CCSDS transfer frames, and each transfer frame a^i consists of 1024 bytes of data. Generate a set of 15 randomized transfer frames.

```
% Generate transfer frames with random data
rng("default")
```

```
numTF = 15;
numBytesInTF = 1024;
dataTF = randi([0 1],numBytesInTF*8,numTF); % a_0,a_1,...,a_numTF-1
```

Prepend a 32-bit attached synchronization marker (ASM) in hexadecimal notation, 1ACFFC1D, to each transfer frame. A data unit that consists of an ASM and a transfer frame is a synchronization-marked transfer frame (SMTF). This figure shows the construction of SMTFs.



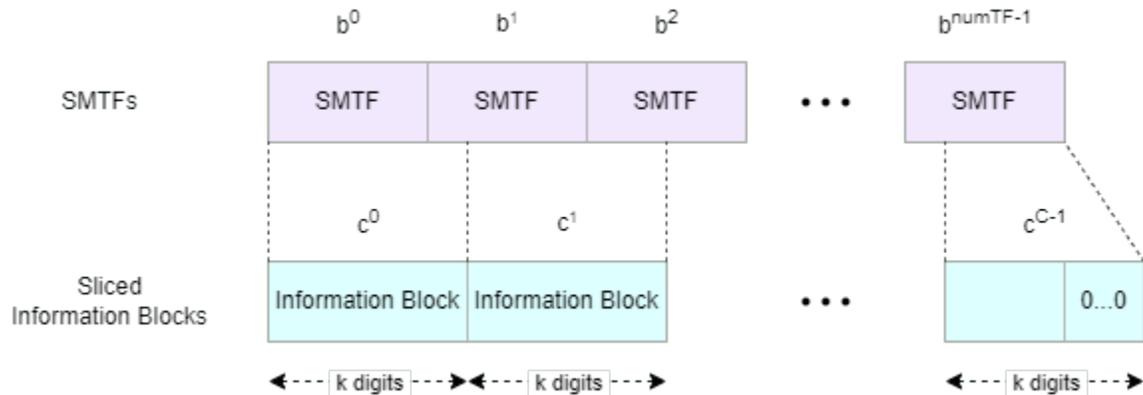
Map CCSDS transfer frames to the corresponding SMTFs.

```
% Attach ASM (0x1ACFFC1D) to the transfer frames
ASM = [0 0 0 1 1 0 1 0 1 1 0 0 1 1 1, ...
       1 1 1 1 1 1 0 0 0 0 0 1 1 0 1]';
SMTF = [repmat(ASM,1,size(dataTF,2)); dataTF]; % b_0,b_1,...,b_numTF-1
```

Slice the sequence of SMTFs into information blocks of length k , where k is determined by the code rate r , which is a managed parameter with values from the set $\{1/3, 1/2, 2/3\}$. This table gives information on each supported code rate and its corresponding information block size. If the size of the last information block is less than k , then the slicer fills that block with the minimum number of 0s (P) required to increase the block size to a multiple of k . The resultant number of information blocks after slicing is numInfoBlocks (C), where $\text{numInfoBlocks} = \frac{\text{numTF} + P}{k}$. This list shows the block size after slicing, in bits, for each of the supported code rates:

- Code rate 1/3 — 5006 bits
- Code rate 1/2 — 7526 bits
- Code rate 2/3 — 10046 bits

This figure shows the slicing and zero fill of SMTFs.



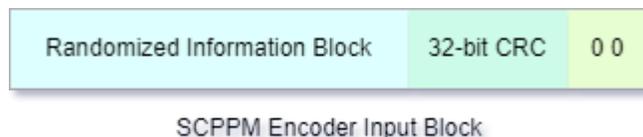
Select the block size (k) value. The code rate (r) is determined based on the block size (k) value.

```
k = 7526;
P = k - rem(length(SMTF(:)),k);
slicerIn = [SMTF(:); zeros(P,1)];
slicerOut = reshape(slicerIn,k,[]); % c_0,c_1,...,c_{C-1}
numInfoBlocks = size(slicerOut,2);
```

Each k -bit information block from the slicer is pseudo-randomized by performing the digit-wise modulo-2 addition with a pseudo-random sequence produced by a linear-feedback shift register, as specified in CCSDS 142.0-B-1 section 3.5 [1] on page 2-23. The shift register is initialized to the all 1s state at the beginning of each sliced information block.

```
% Initialize pseudonoise sequence generator System object
pnSequence = comm.PNSequence("SamplesPerFrame",k, ...
    Polynomial=[1 1 0 1 0 1 0 0 1], ...
    InitialConditions=ones(8,1));
% Perform pseudo randomization on SMTFs
psIn = repmat(pnSequence(),1,numInfoBlocks);
psOut = xor(psIn,slicerOut); % d_0,d_1,...,d_{C-1}
```

Append 32 cyclic redundancy check (CRC) bits to the end of each pseudo-randomized information block, as described in CCSDS 142.0-B-1 section 3.6 [1] on page 2-23. The CRC is used for block error detection and to terminate the serially concatenated pulse position modulation (SCPPM) decoder. Append two 0s to each information block with attached CRC to terminate the outer convolutional encoder in the SCPPM encoder. The CRC and termination bits add a combined 34 bits to the randomized information block. This figure shows the structure of a pseudo-randomized information block with CRC and termination bits.



Pass the resultant SCPPM encoder input block through the SCPPM encoder. The SCPPM forward error correction scheme is necessary for efficient operation of the deep space link. The encoder maps $k + 34$ bits into $S = 15120/m$ pulse position modulation (PPM) symbols (where m is the modulation order, in the range $[2, 8]$).

```

% Perform SCPPM encoding
crc32Generator = comm.CRCGenerator(...
    Polynomial="x^32+x^29+x^18+x^14+x^3+1", ...
    InitialConditions=1);
m = ;
encSym = zeros(15120/m,numInfoBlocks);
for frmIdx=1:numInfoBlocks
    crcData = crc32Generator(psOut(:,frmIdx));           % Generate CRC
    scppmIn = [crcData; 0; 0];                          % f_0,f_1,...,f_C-1
    [encSym(:,frmIdx),info] = ccstdsSCPPMEncode(scppmIn,m); % q_0,q_1,...,q_C-1
end

```

Channel interleave the sequence of PPM symbols with a convolutional interleaver, as described in CCSDS 142.0-B-1 section 3.9 [1] on page 2-23. N and B are managed parameters and are so chosen that $B \times N$ is a multiple of S , which in turn is a multiple of N . The interleaver has N rows, where an i th row contains a shift register of length $i \times B$, consisting of $i \times B$ PPM symbols. After passing the last symbol, the interleaver is operated for another $B \times N \times (N - 1)$ steps before the last symbol appears at the output. Thus, the output contains $B \times N \times (N - 1)$ more symbols than the input. Since $B \times N \times (N - 1)$ is a multiple of S , there are no leftover PPM symbols in the last block. The resultant sequence r has $R = C + \frac{(B \times N \times (N - 1))}{S}$ blocks, each containing S symbols.

```

% Channel interleaving
N = 36;
B = 420;
channelInterleaver = comm.ConvolutionalInterleaver("NumRegisters",N, ...
    "RegisterLengthStep",B);

% Add B*N*(N-1) zeros for the last symbol to appear at the output
chInterleaveOut = channelInterleaver([encSym(:); zeros(B*N*(N-1),1)]); % r_0,r_1,...,r_R-1

```

Prepend a code synchronization marker (CSM) of W PPM symbols based on m to each interleaved codeword.

```

% CSM Specification
if m == 2
    CSM = [0 3 1 2 1 3 2 0 0 3 2 1 0 2 1 3 1 0 3 2 3 2 1 0];
elseif m == 3
    CSM = [0 3 1 2 5 4 7 6 6 7 4 5 2 1 3 0];
else
    CSM = [0 2 7 14 1 2 15 5 8 4 10 2 14 3 14 11];
end

% Attach CSM to each S symbol block
csmIn = reshape(chInterleaveOut,15120/m,[]);
csmOut = [repmat(CSM',1,size(csmIn,2)); csmIn]; % s_0,s_1,...,s_R-1

```

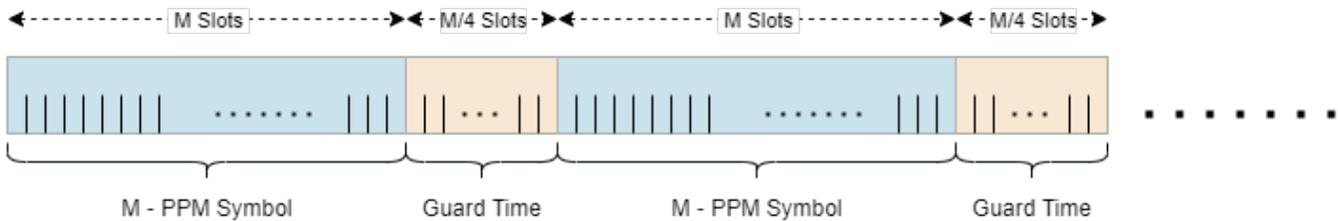
Repeat each PPM symbol q_d times, where q_d is a managed parameter with values from the set $\{1, 2, 3, 4, 8, 16, 32\}$. Use the repetition factor to increase the level of soft information for the decoder and to aid super-symbol synchronization. The output of the repeater has numSym symbols, where $\text{numSym} = q_d \times R \times (S + W)$.

```

% Repeat each PPM symbol
qd = ;
repOut = repelem(csmOut(:),qd,1); % t_0,t_1,...,t_T-1

```

Map each PPM symbol to a binary vector of length M (where $M = 2^m$) by placing $(M - 1)$ 0s and a single 1 into one of the possible positions, based on the code symbol value. M -ary PPM provides excellent power efficiency for the photon-starved operating regime of deep space optical links. After each set of M slots, insert $\frac{M}{4}$ guard slots. Guard slots are used for compatibility with existing laser technology, and to aid in synchronization. Each transmitted PPM symbol occupies $\frac{5 \times M}{4}$ PPM slots, which are transmitted over the physical channel. This figure shows slot mapping and guard slot insertion.



After slot mapping and guard slot insertion, the result is a slot sequence $v_0, v_1, \dots, v_{\text{numSym} - 1}$, where each v_j is a vector of length $\frac{5 \times M}{4}$.

```
% Slot mapping and guard slot insertion
M = 2^m;
txOut = zeros(length(repOut)*5*M/4,1);
mapIdx = (0:length(repOut)-1)'*5*M/4 + repOut + 1;
txOut(mapIdx) = 1; % v_0,v_1,...,v_T-1
```

This table shows the managed parameters for the HPE telemetry signaling.

Managed Parameter	Allowed Values
TM/AOS/USLP transfer frame length (octets)	Max 65536
Modulation order, M	4, 8, 16, 32, 64, 128, 256
Code rate, r	1/3, 1/2, 2/3
Number of rows in channel interleaver, N	B*N must be a multiple of S , which in turn must be a multiple of N , where $S = 15120/\log_2(M)$
Shift register length increment in channel interleaver, B	
Repeat factor, q	1, 2, 3, 4, 8, 16, 32

Signal Visualization

Modulate the binary vector received from the coding and synchronization sublayer using On-Off keying (OOK).

```
Rb = 1e9; % Bit rate
Tb = 1/Rb; % Bit duration
```

```

sps = 10; % Samples per symbol
fsamp = Rb*sps; % Sampling frequency
unrz = ones(1,sps); % Signal for unipolar Non-return-to-zero (NRZ)
sigLen = 1000; % Number of bits to visualize
binData = txOut(1:sigLen); % Generate data of length sigLen
modSignal = upfirdn(binData,unrz,sps); % Modulate data using 00K

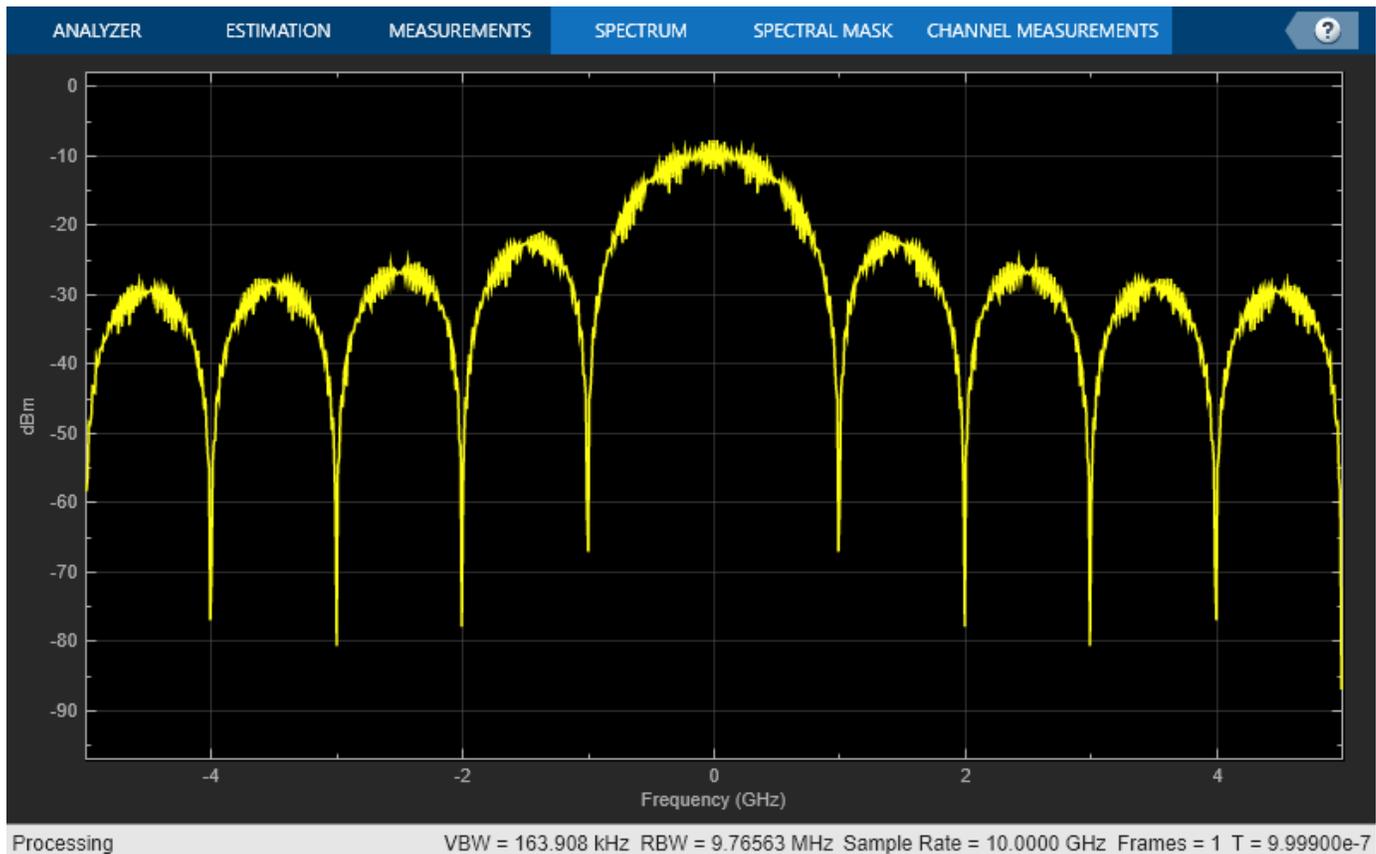
```

Create a `spectrumAnalyzer` object to display the frequency spectrum of the generated CCSDS optical HPE TM waveform.

```

txAnalyzer = spectrumAnalyzer(SampleRate=fsamp,AveragingMethod="exponential");
txAnalyzer(modSignal)

```



Pass the modulated signal through a deep space Poisson channel.

```

ns = 3; % Average signal photons per pu
nb = 0.0001; % Average noise photons per slo
receivedSignal = HelperDeepSpacePoissonChannel(modSignal,ns,nb);

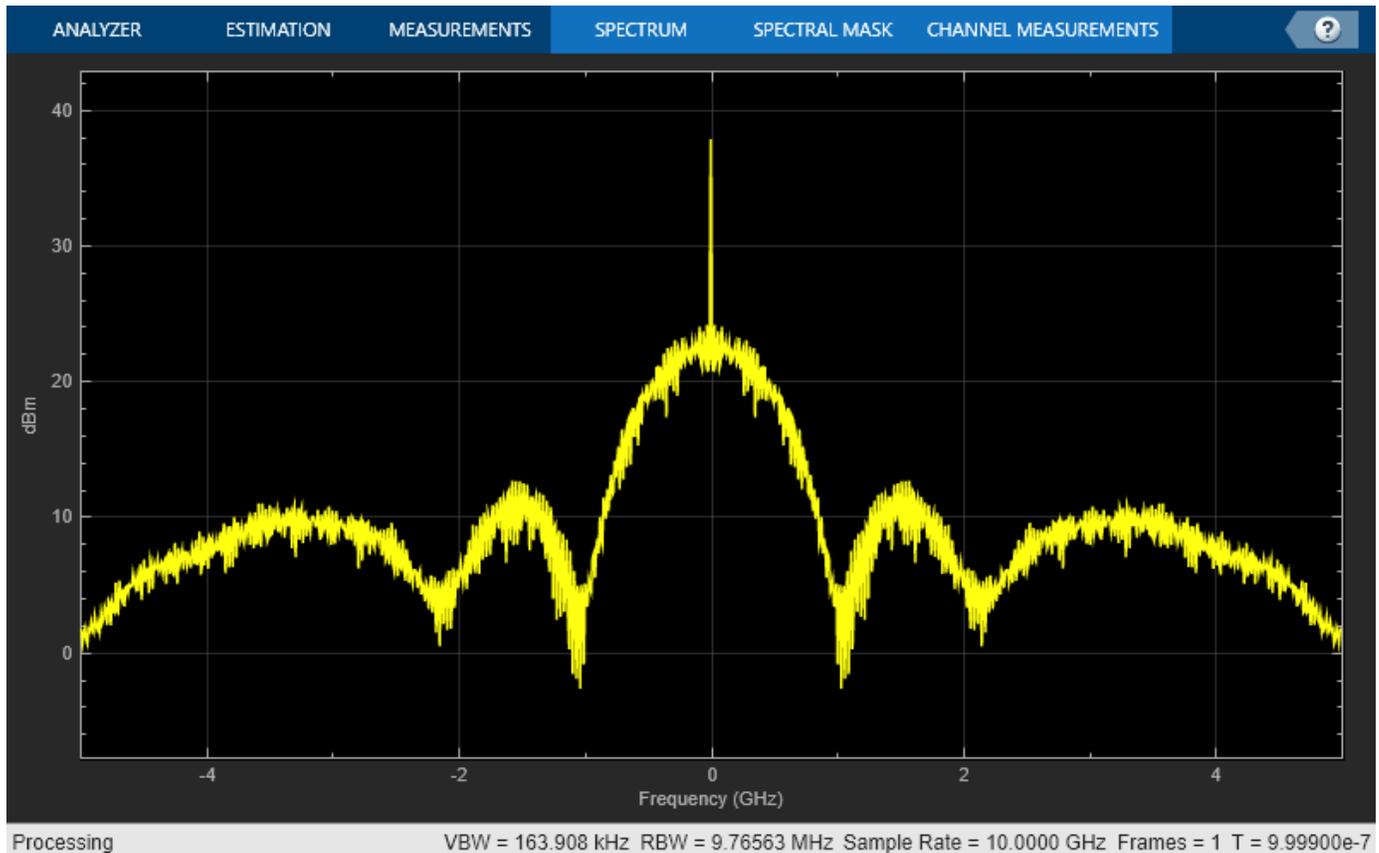
```

Create a `spectrumAnalyzer` object to display the frequency spectrum of the received CCSDS optical HPE TM waveform.

```

rxAnalyzer = spectrumAnalyzer(SampleRate=fsamp,AveragingMethod="exponential");
rxAnalyzer(receivedSignal)

```



Appendix

This example uses the helper file:

HelperDeepSpacePoissonChannel — Add Poisson distributed noise to input signal

References

[1] The Consultative Committee for Space Data Systems. *Optical Communications Coding and Synchronization, Recommended Standard, Issue 1*. CCSDS 142.0-B-1. Washington, D.C.: CCSDS, August 2019. <https://public.ccsds.org/Pubs/142x0b1.pdf>.

[2] The Consultative Committee for Space Data Systems. *Optical Communications Physical Layer, Recommended Standard, Issue 1*. CCSDS 141.0-B-1. Washington, D.C.: CCSDS, August 2019. <https://public.ccsds.org/Pubs/141x0b1.pdf>.

[3] The Consultative Committee for Space Data Systems. *TM Space Data Link Protocol, Recommended Standard, Issue 2*. CCSDS 132.0-B-2. Washington, D.C.: CCSDS, September 2015. <https://public.ccsds.org/Pubs/132x0b3.pdf>.

[4] The Consultative Committee for Space Data Systems. *AOS Space Data Link Protocol. Recommended Standard, Issue 3*. CCSDS 732.0-B-3. Washington, D.C.: CCSDS, September 2015. <https://public.ccsds.org/Pubs/732x0b3e1s.pdf>.

[5] The Consultative Committee for Space Data Systems. *Unified Space Data Link Protocol. Recommended Standard, Issue 1*. CCSDS 732.1-B-1. Washington, D.C.: CCSDS, October 2018. <https://public.ccsds.org/Pubs/732x1b1s.pdf>.

See Also

ccsdsSCPPMEncode

Related Examples

- “End-to-End CCSDS SCPPM Simulation Using Deep Space Poisson Channel” on page 4-102

DVB-S2X Super-Frame Generation for Formats 0 and 1

This example shows how to generate a Super-Frame (SF) complex baseband waveform. The SF waveform consists of several fields that are inserted in the SF structure, such as start of super-frame (SOSF), super-frame format indicator (SFFI), DVB-S2/S2X physical layer frames (PLFRAMEs), and SF-aligned pilots.

Introduction

The super-framing structure provides better support for synchronization algorithms using several SF-specific fields such as SOSF, SFFI, and SF-aligned pilots, to enhance the receiver performance. The structure for formats 0 and 1 is of a fixed-length of 612,540 symbols and hosts DVB-S2X and DVB-S2 PLFRAMEs, respectively.

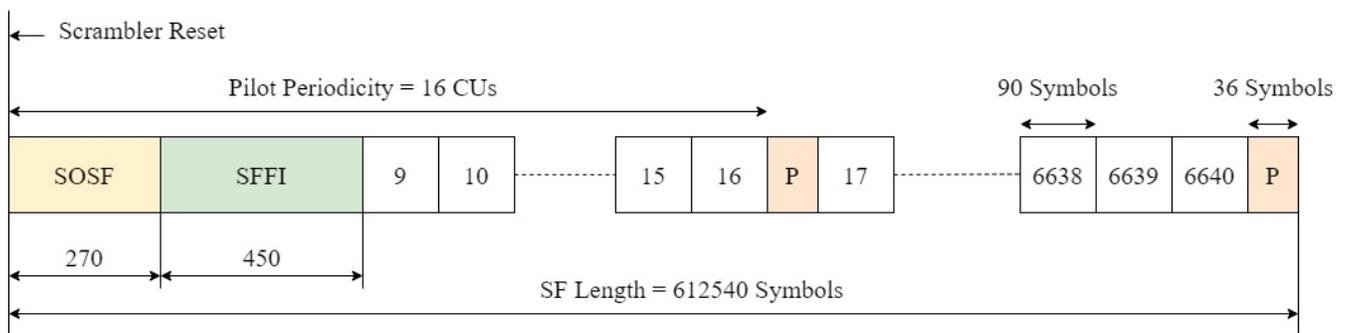
- The first field in the SF is the SOSF, which is of length 270 symbols.
- The second field is the SFFI, which is of length 450 symbols.
- The remaining portion of the SF is allocated to the DVB-S2/S2X PLFRAMEs and the SF-aligned pilots (if signalled).

The SF symbols are scrambled using a SF-wide scrambling technique, where two scramblers are used. The reference scrambler is used to scramble the SOSF and the SF-aligned pilots. The payload scrambler is used to scramble the SFFI field. The DVB-S2X PLFRAMEs and DVB-S2 PLFRAMEs are scrambled using the scrambler defined in ETSI EN 302 307-1 [2] on page 2-31, clause 5.5.4.

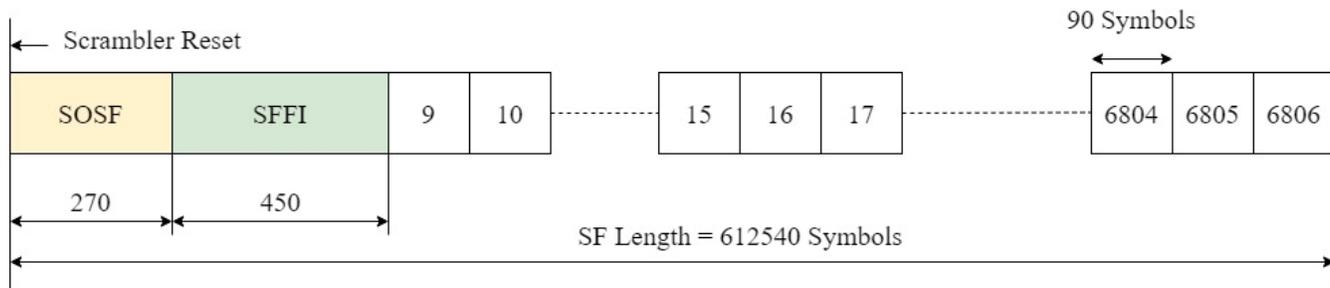
The format-specific content section of the SF is divided into resource slots called Capacity Units (CU). Each CU is of length 90 symbols and contains the PLFRAMEs symbols.

These figures show the SF structure:

- If SF-aligned pilots are inserted



- If SF-aligned pilots are not inserted



Because PLFRAMES are divided into slots of 90 symbols each, they can be assigned the necessary number of CUs to be inserted into the SF. Very low signal-to-noise ratio (VL-SNR) frames are modified such that the VL-SNR pilot field lengths are of length 90 symbols to avoid any overlap with the SF-aligned pilots. When the pilot fields are used, the total number of CUs available in a single SF is 6640. In the absence of pilots, the number of CU available is 6806.

This example demonstrates how to parameterize and generate a SF with its respective fields. You can generate multiple DVB-S2/S2X PLFRAMES that are inserted into the SF.

SOSF, SFFI, and SF-aligned Pilots

The SOSF field is of length 270 symbols, and is a sequence derived from a Walsh-Hadamard matrix of size 256, padded with a sequence of length 14 symbols. Any one of the 256 sequences can be used as the SOSF sequence for the SF. The SOSF sequence is mapped onto the constellation by multiplying the selected sequence by $(1 + j)/\sqrt{2}$.

The SFFI field is of length 450 symbols. The format can be either 0 or 1. These operations generate the SFFI field:

- 1 Converting the numeric format value to a 4-bit representation
- 2 Performing simplex coding as described in Annexure E.2.3 [1] on page 2-31, to achieve the required code rate of 4/15
- 3 Bit-wise repetition of the encoded bits by a factor for 30 to obtain X_{sffi}
- 4 Constellation mapping using the equation $(-2 * X_{sffi} + 1)(1 + j)/\sqrt{2}$

The pilot field (P) of length 36 symbols and is a derived from a Walsh-Hadamard matrix of size 32, padded with a sequence of 4 symbols. Any one of the 32 sequences can be used as the pilot sequence for the SF. The pilot sequence is mapped onto the constellation by multiplying the selected sequence by $(1 + j)/\sqrt{2}$. The pilot fields are placed with a periodicity of 16 CUs, however, they do not occupy a full CU. When SF-aligned pilots are inserted, a complete SF has 415 pilot fields.

Download DVB-S2X LDPC Parity Matrices Data Set

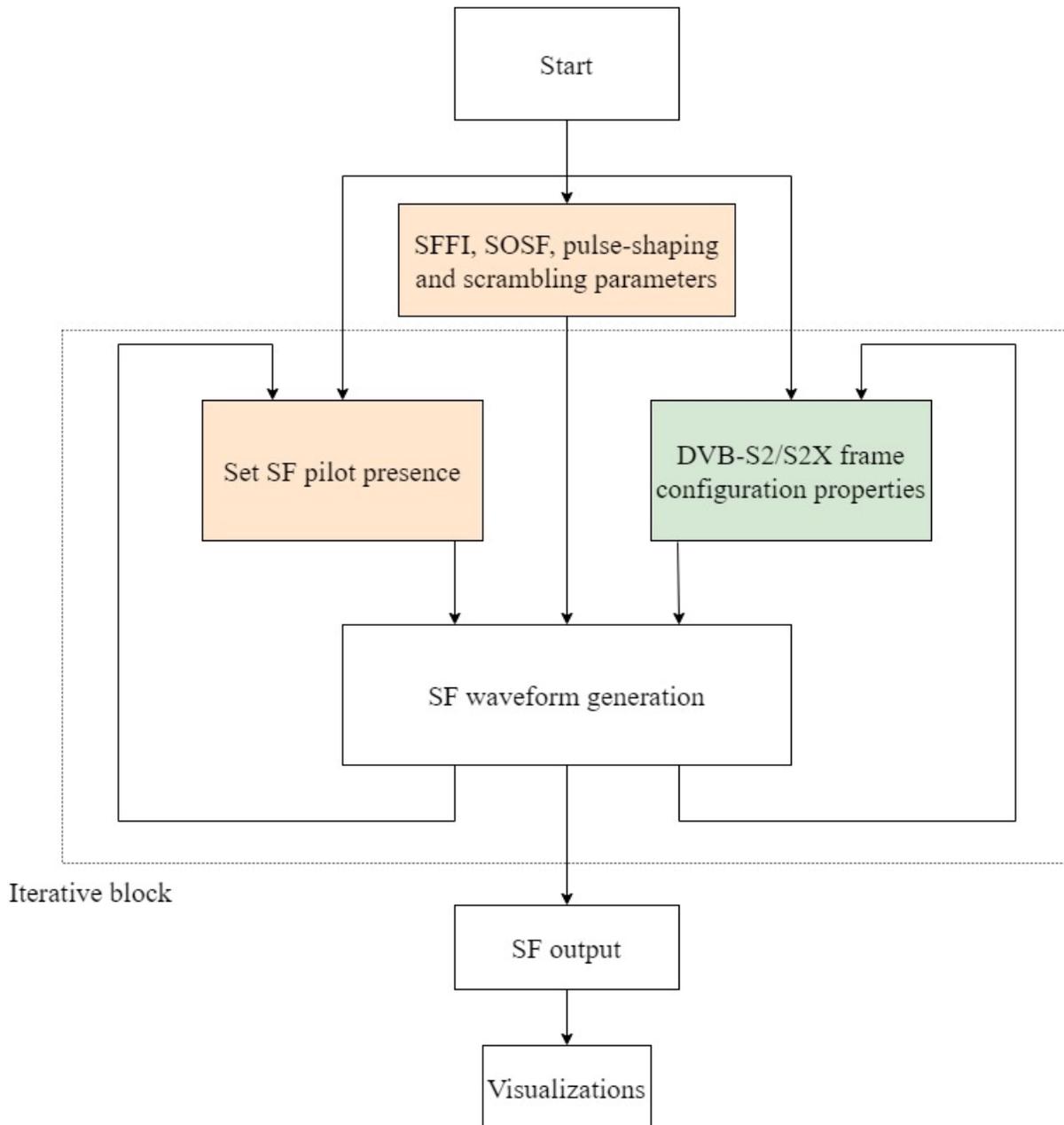
This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
    end
end
```

```
    unzip('s2xLDPCParityMatrices.zip');  
end  
addpath('s2xLDPCParityMatrices');  
end
```

SF Generation Workflow

This figure summarizes the workflow to generate the SF waveform and its output.



SOSF, SFFI, Pulse-Shaping, and Scrambling Properties of SF

Generate the SF using the `HelperSuperFrameGenerator0And1.m` object and set its parameters.

- **SF Format:** Set to 0 or 1.
- **SOSFIndex:** Select the appropriate Walsh-Hadamard sequence from the range [0, 255] to use as the SOSF.
- **Pulse-Shaping Parameters:** These include `SamplesPerSymbol`, `RolloffFactor`, and `FilterSpanInSymbols`.
- **ScramblingCodeNumbers:** Select the N th gold code for generating the scrambling sequence for the reference and the payload scramblers. The property accepts a row vector of length two, [N_{ref} $N_{payload}$]. The value of N is in the range [0, 148,574].

```
sfWaveGen = HelperSuperFrameGenerator0And1;
sfWaveGen.Format = 0;
sfWaveGen.SOSFIndex = 0;
sfWaveGen.ScramblingCodeNumbers = [14 26];
sfWaveGen.RolloffFactor = 0.05;
sfWaveGen.FilterSpanInSymbols = 10;
sfWaveGen.SamplesPerSymbol = 4;
```

SF-Aligned Pilots Properties

These properties signal pilot existence in the SF and the pilot sequence to be used.

- **PilotIndex** — Select an appropriate Walsh-Hadamard sequence from the range [0, 31] to use as the pilot sequence.
- **HasPilotsInFirstSuperFrame** — Flag to determine whether SF pilots are inserted in the first SF.
- **HasPilotsInNextSuperFrame** — Flag to determine whether the next SF has pilots.

```
sfWaveGen.PilotIndex = 0;
sfWaveGen.HasPilotsInFirstSuperFrame = false;
sfWaveGen.HasPilotsInNextSuperFrame = false;
```

DVB-S2/S2X Frames Properties

These properties generate the PLFRAMES that constitute a SF.

- **StreamFormat** — Set as either "TS" or "GS", corresponding to transport stream and generic stream, respectively.
- **PLSDecimalCode** — Physical layer signalling, specified as a decimal value.
- **DFL** — Data field length.
- **PLScramblingIndex** — Value in the range [0, 7] to specify the scrambling sequence used for the DVB-S2/S2X frames only.

```
sfWaveGen.StreamFormat = GS;
sfWaveGen.PLSDecimalCode = 132;
```

```
sfWaveGen.DFL = 18448;
sfWaveGen.PLScramblingIndex = 0; ;
```

If StreamFormat is "GS", then you must set the user packet length (UPL). UPL can either be 0 or less than the data field length (DFL) value.

```
if strcmp(sfWaveGen.StreamFormat, "GS")
    sfWaveGen.UPL = 0;
end
```

If the PLSDecimalCode value specifies a VL-SNR FRAME, you must also set the canonical MODCOD property. Note: VL-SNR frames must not be combined with regular frames.

```
if sfWaveGen.PLSDecimalCode == 129 || sfWaveGen.PLSDecimalCode == 131
    sfWaveGen.CanonicalMODCODName = "BPSK 1/5";
end
disp(sfWaveGen)
```

HelperSuperFrameGenerator0And1 with properties:

```
Format: 0
SOSFIndex: 0
PilotIndex: 0
ScramblingCodeNumbers: [14 26]
HasPilotsInFirstSuperFrame: false
HasPilotsInNextSuperFrame: false
RolloffFactor: 0.0500
FilterSpanInSymbols: 10
SamplesPerSymbol: 4
```

Show all properties

Set the number of DVB-S2/S2X frames to be generated and inserted into the SF—if all DVB-S2/S2X frames cannot be inserted in one SF, the output contains multiple SFs.

```
numPLFramesPerPLSDecimalCode = 19;
```

Obtain characteristic information of the SF and the individual DVB-S2/S2X frames. These output fields of the info object function as follows.

- 1 **FECFrame:** Forward error correction (FEC) frame format, specified as "normal", "medium", or "short".
- 2 **ModulationScheme:** Modulation scheme used to map DVB-S2/S2X frames to the constellation.
- 3 **LDPCCodeIdentifier:** Output code rate of the low density parity code (LDPC) encoder.
- 4 **NumPLFRAMESPerSuperFrame:** For a specified PLSDecimalValue, the value of this field is equal to the number for PLFRAMES with the same PLSDecimalCode that can be inserted in one SF.

```
superFrameInfo = sfWaveGen.info;
```

```
FECFrame: "normal"
ModulationScheme: "QPSK"
LDPCCodeIdentifier: "13/45"
MinNumPLFRAMESPerSuperFrame: 19
```

For the given PLSDecimalCode, if 19 PLFRAMES are generated, the last PLFRAME in Super-Frame is

Initialize the random number generator with a seed. Vary the seed to obtain different input data. The value used here, 73, is arbitrary.

```
seed = 73;  
rng(seed);
```

Create an iterative loop to generate the SF output.

```
superFrameOut = [];  
for numPLFrames = 1:numPLFramesPerPLSDecimalCode  
    % Generate data using the generateInputData method  
    data = sfWaveGen.generateInputData;  
    sfOutputFiltered = sfWaveGen(data);  
    superFrameOut = [superFrameOut; sfOutputFiltered]; %#ok<AGROW>  
end
```

```
Super-Frame Number 1 is being generated.  
Super-Frame Number 1 is complete.  
Super-Frame Number 2 is being generated.
```

Display the number of CUs that are still unused in the latest SF.

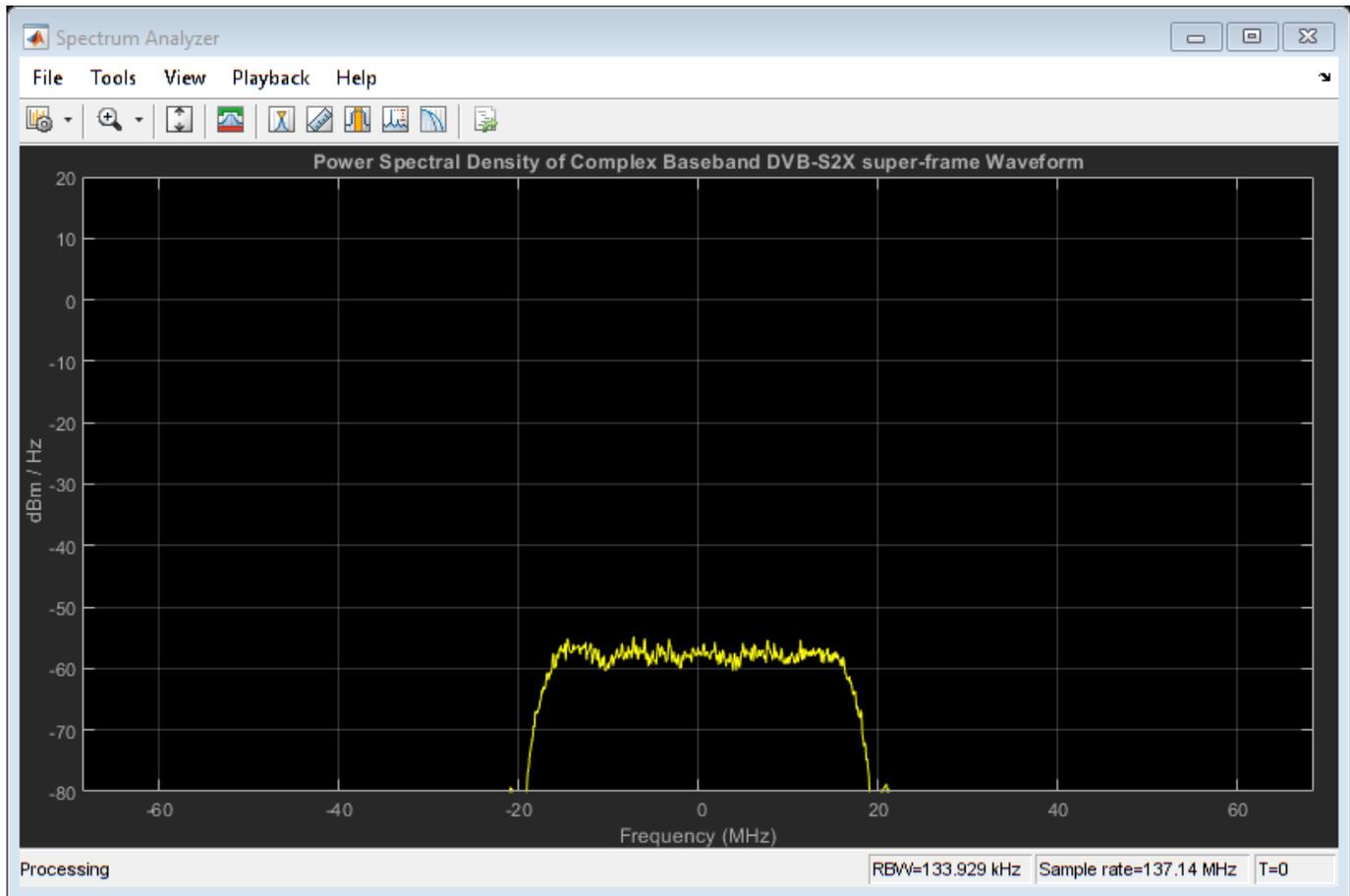
```
getRemainingCUValue(sfWaveGen);
```

```
The current Super-Frame has 6737 Capacity Units available.
```

Visualization

Plot the signal power spectral density at the baseband. Set the channel bandwidth to get an appropriate plot.

```
chanBW = 36e6; % Channel Bandwidth  
Rsymb = chanBW/(1 + sfWaveGen.RolloffFactor);  
Fsamp = Rsymb*sfWaveGen.SamplesPerSymbol;  
bbscope = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...  
    'PlotAsTwoSidedSpectrum',true, ...  
    'SpectrumType','Power density', ...  
    'AveragingMethod','Exponential', ...  
    'Title','Power Spectral Density of Complex Baseband DVB-S2X super-frame Waveform');  
bbscope(superFrameOut);
```



Further Exploration

The example shows how to generate the SF output for formats 0 and 1. Explore these options to generate unique waveforms.

- Vary the number of PLFRAMEs. You can also try to change the `PLSDecimalCode` value of each PLFRAME and then generate the SF.
- Adjust the `ScramblingCodeNumbers` to get SFs scrambled with different scrambling sequences. You can perform cross-correlation on the SOSF and SFFI sequences in the SF output.
- Select different SOSF and pilot sequences by varying their respective property values.
- Vary the pilot presence in neighboring SFs by changing the `HasPilotsInNextSuperFrame` property.

Appendix

The example uses the helper function:

- `HelperSuperFrameGenerator0And1.m`: Generate DVB-S2X Super-Frame waveform samples for formats 0 and 1.

Bibliography

- 1 ETSI Standard EN 302 307-2 V1.3.1(2021-07). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 extensions (DVB-S2X)*.
- 2 ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 3 ETSI Standard TR 102 376-2 V1.3.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: S2 extensions (DVB-S2X)*.
- 4 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

See Also

Objects

dvbs2xWaveformGenerator | dvbs2WaveformGenerator

Related Examples

- “DVB-S2X Super-Frame Generation for Formats 2 and 3” on page 2-33
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames” on page 4-69
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames” on page 4-54
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode” on page 4-87

DVB-S2X Super-Frame Generation for Formats 2 and 3

This example shows how to generate a Super-Frame (SF) complex baseband waveform for formats 2 and 3. The SF waveform consists of several fields that are inserted in the SF structure, such as start of super-frame (SOSF), super-frame format indicator (SFFI), complex forward error correction frames (XFECFRAMES), SF-aligned pilots and modulated pilot symbols (P2), and the physical layer header (PLH).

Introduction

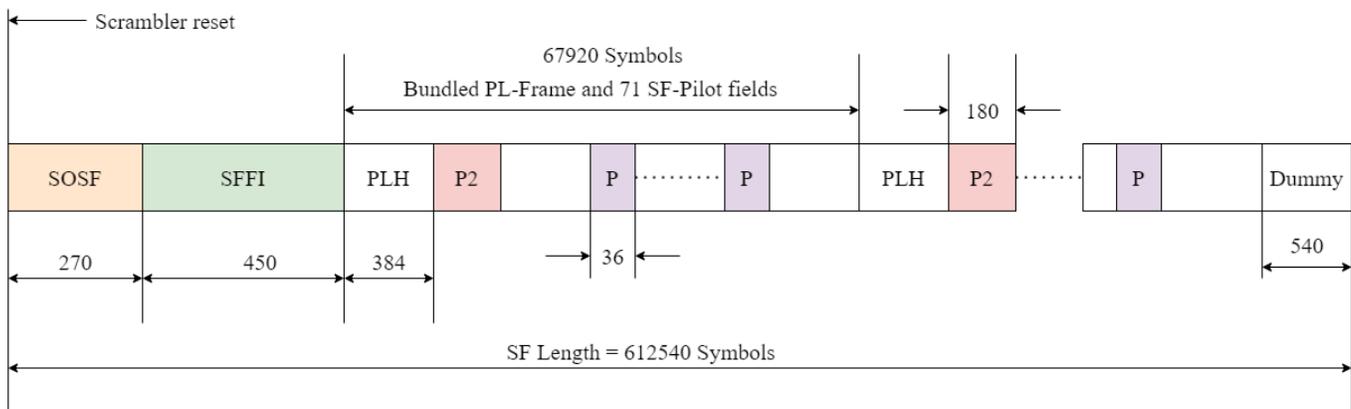
The super-framing structure provides better support for synchronization algorithms using several SF-specific fields, such as SOSF, SFFI, and SF-aligned pilots, to enhance the receiver performance. The structure for formats 2 and 3 is of a fixed-length of 612,540 symbols and hosts bundled physical layer frames of length 64,800 and 16,200 symbols, respectively.

- The first field in the SF is the SOSF, which is of length 270 symbols.
- The second field is the SFFI, which is of length 450 symbols.
- The remaining portion of the SF is allocated to the bundled PLFRAMES, PLHs, P2 fields, and SF-aligned pilots.

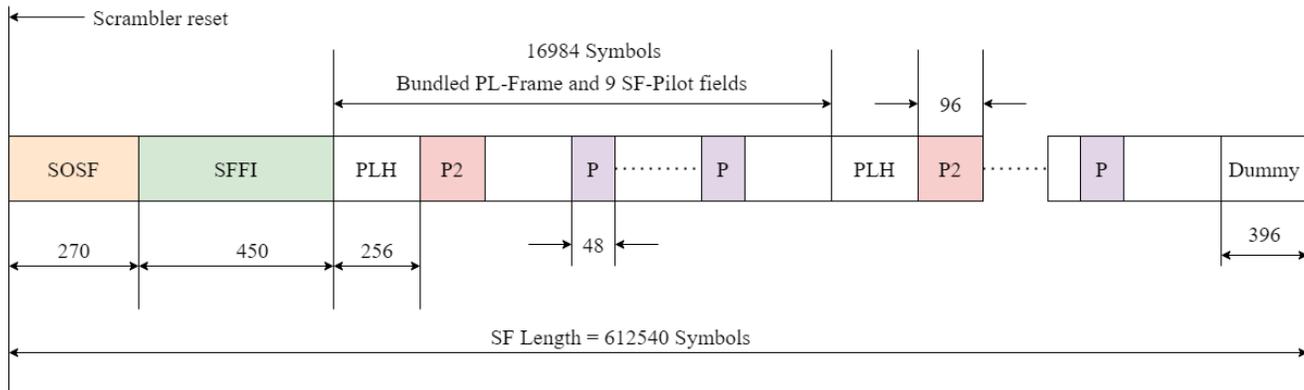
The SF symbols are scrambled using a SF-wide scrambling technique, where two scramblers are used. The reference scrambler is used to scramble the SOSF and the SF-aligned pilots. The payload scrambler is used to scramble the SFFI field, PLH fields, P2 fields, dummy symbols, and bundled PLFRAMES.

These figures show the SF structure:

- Format 2



- Format 3



SOSF, SFFI, SF-aligned Pilots, PLH, and P2

The SOSF field is of length 270 symbols, and is a sequence derived from a Walsh-Hadamard matrix of size 256, padded with a sequence of length 14 symbols. Any one of the 256 sequences can be used as the SOSF sequence for the SF. The SOSF sequence is mapped onto the constellation by multiplying the selected sequence by $(1 + j)/\sqrt{2}$.

The SFFI field is of length 450 symbols. The format can be either 2 or 3. These operations generate the SFFI field:

- 1 Converting the numeric format value to a 4-bit representation
- 2 Performing simplex coding as described in Annexure E.2.3 [1] on page 2-38, to achieve the required code rate of 4/15
- 3 Bit-wise repetition of the encoded bits by a factor for 30 to obtain X_{sffi}
- 4 Constellation mapping using the equation $(-2 * X_{sffi} + 1)(1 + j)/\sqrt{2}$

For format 2, the pilot field (P) length is 36 symbols and is derived from a Walsh-Hadamard matrix of size 32, padded with a sequence of 4 symbols. For format 3, P is of length 48 symbols and is a sequence derived from a Walsh-Hadamard matrix of size 32 padded with a sequence of length 16 symbols. Any one of the 32 sequences can be used as the pilot sequence for the SF. The pilot sequence is mapped onto the constellation by multiplying the selected sequence with $(1 + j)/\sqrt{2}$.

The PLH has the physical layer signalling (PLS) information of the corresponding bundled PL-frame. The PLH is generated by repeating the 64-bit PLS information, as defined in 5.5.2.4 [2] on page 2-38. To generate the PLH, use a repetition factor of 6 for format 2 and a repetition factor of 4 for format 3. The bits are modulated using $\pi/2$ -BPSK.

Modulate P2 using the same modulation scheme used by the XFECFRAMEs in the bundle. The generation of P2 for format 2 is described in Annexure E.3.4.3 [1] on page 2-38, and Annexure E.3.5.3 [1] on page 2-38 for format 3.

Download DVB-S2X LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```

if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
    addpath('s2xLDPCParityMatrices');
end

```

SOSF, SFFI, Pulse-Shaping and Scrambling Properties of SF

Generate the SF using the `HelperSuperFrameGenerator2And3` object and set its parameters.

- **SF Format** – Set to 2 or 3.
- **SOSFIndex** – Select the appropriate Walsh-Hadamard sequence from the range [0, 255] to use as the SOSF.
- **Pulse-Shaping Parameters** – These include `SamplesPerSymbol`, `RolloffFactor`, and `FilterSpanInSymbols`.
- **ScramblingCodeNumbers** – Select the N th gold code for generating the scrambling sequence for the reference and the payload scramblers. This property accepts a row vector of length two, $[N_{ref} N_{payload}]$. The value of N is in the range [0, 148,574].

```

sfWaveGen = HelperSuperFrameGenerator2And3;
sfWaveGen.Format = 2;
sfWaveGen.SOSFIndex = 0;
sfWaveGen.ScramblingCodeNumbers = [14 26];
sfWaveGen.RolloffFactor = 0.05;
sfWaveGen.FilterSpanInSymbols = 10;
sfWaveGen.SamplesPerSymbol = 4;

```

SF-aligned Pilots property

The following property determines the pilot sequence to be used. Select an appropriate Walsh-Hadamard sequence from the range [0, 31] to use as the pilot sequence.

```

sfWaveGen.PilotIndex = 0;

```

XFECFRAMEs properties

The following properties are used to generate the XFECFRAMEs that constitute a bundle:

- **StreamFormat** – Set as either "TS" or "GS", corresponding to transport stream and generic stream respectively.
- **PLSDecimalCode** – Physical layer signalling, specified as a decimal value.
- **DFL** – Data field length.

```

sfWaveGen.StreamFormat = TS;
sfWaveGen.PLSDecimalCodeValue = 33;
sfWaveGen.DFL = 2992;

```

If `StreamFormat` is "GS", you must set the user packet length (UPL). UPL can either be 0 or less than the data field length (DFL) value.

```

if strcmp(sfWaveGen.StreamFormat, "GS")
    sfWaveGen.UPL = 0;
end
disp(sfWaveGen)

```

HelperSuperFrameGenerator2And3 with properties:

```

        Format: 2
        SOSFIndex: 0
        PilotIndex: 0
    ScramblingCodeNumbers: [14 26]
        RolloffFactor: 0.0500
        FilterSpanInSymbols: 10
        SamplesPerSymbol: 4

```

Show all properties

Set the number of bundles to be generated and inserted into the SF.

```
numBundlesToGenerate = 9;
```

Obtain characteristic information regarding the construction of the SF and the individual XFECFRAMEs. The fields are as follows.

- 1 **FECFrame** – Forward error correction (FEC) frame format, specified as "normal", "medium", or "short".
- 2 **ModulationScheme** – Modulation scheme used to map DVB-S2/S2X frames to the constellation.
- 3 **LDPCCodeIdentifier** – Output code rate of the low density parity code (LDPC) encoder.

```
superFrameInfo = sfWaveGen.info
```

```

superFrameInfo = struct with fields:
        FECFrame: "short"
        ModulationScheme: "QPSK"
        LDPCCodeIdentifier: "1/4"

```

Initialize random number generator with seed. Vary the seed to obtain different input data. The value used here, 73, is arbitrary.

```
seed = 73;
rng(seed);
```

Create an iterative loop to generate the SF output.

```

superFrameOut = [];
for numBundles = 1:numBundlesToGenerate
    % Generate data using the generateInputData method
    data = sfWaveGen.generateInputData;
    sfOutputFiltered = sfWaveGen(data);
    superFrameOut = [superFrameOut; sfOutputFiltered]; %#ok<AGROW>
end

```

```

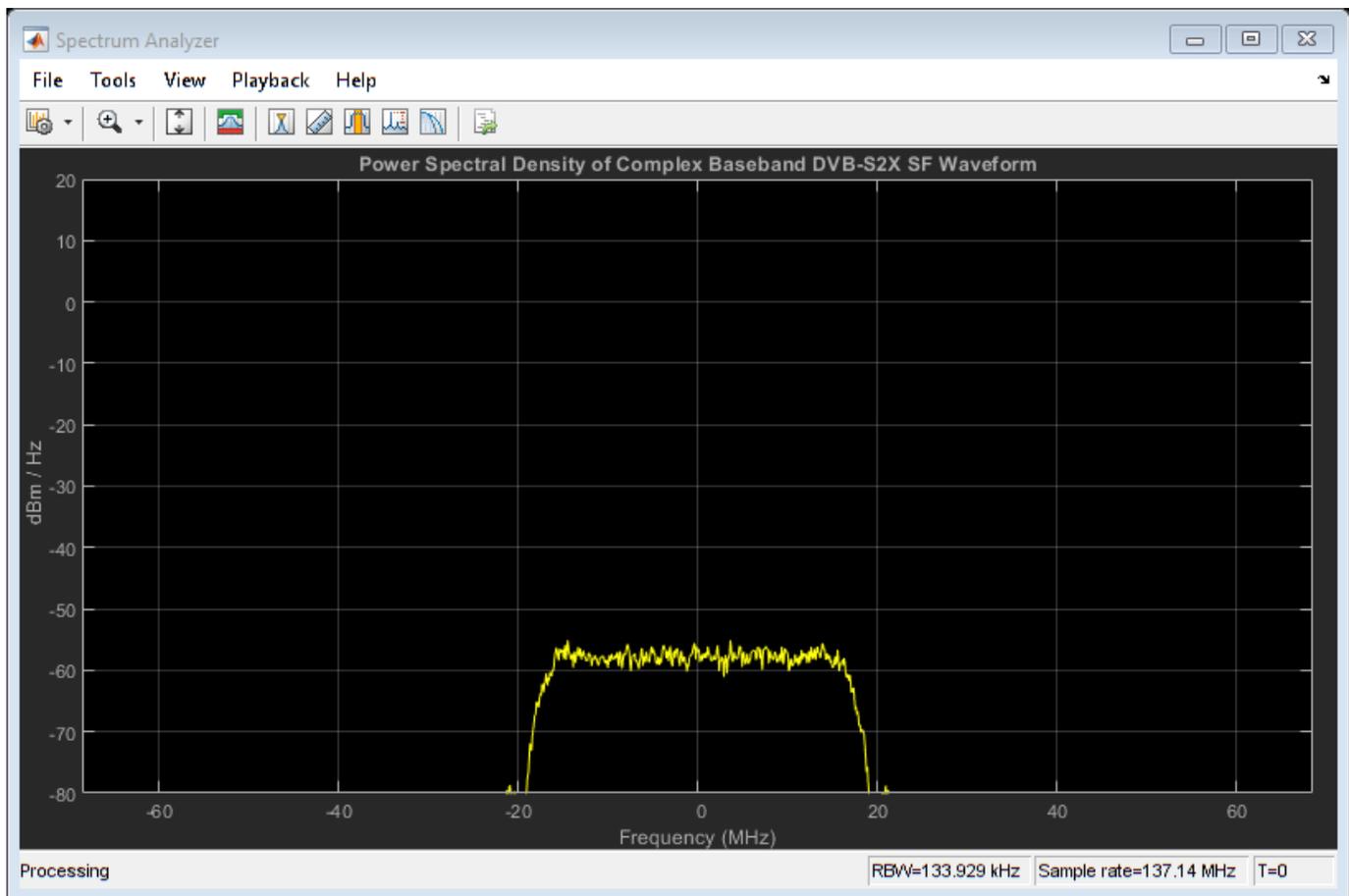
Super-Frame Number 1 is being generated.
Super-Frame Number 1 is complete.

```

Visualization

Plot the signal power spectral density at the baseband. Set the channel bandwidth to get an appropriate plot.

```
chanBW = 36e6; % Channel Bandwidth
Rsymb = chanBW/(1 + sfWaveGen.RolloffFactor);
Fsamp = Rsymb*sfWaveGen.SamplesPerSymbol;
bbscope = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'PlotAsTwoSidedSpectrum',true, ...
    'SpectrumType','Power density', ...
    'AveragingMethod','Exponential', ...
    'Title', 'Power Spectral Density of Complex Baseband DVB-S2X SF Waveform');
bbscope(superFrameOut);
```



Further Exploration

The example shows how to generate the SF output for formats 2 and 3. Explore these options to generate unique waveforms.

- Vary the number of bundles. You can also try to change the `PLSDecimalCodeValue` of each bundle, and then generate the SF.
- Adjust the `ScramblingCodeNumbers` to get SFs scrambled with different scrambling sequences. You can perform cross-correlation on the scrambling sequences in the SF output.

- Select different SOSF and pilot sequences by varying their respective property values.

Appendix

The example uses the helper function:

- `HelperSuperFrameGenerator2And3.m`: Generate DVB-S2X Super-frame waveform samples for formats 2 and 3.

Bibliography

- 1 ETSI Standard EN 302 307-2 V1.3.1(2021-07). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 extensions (DVB-S2X)*.
- 2 ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 3 ETSI Standard TR 102 376-2 V1.3.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: S2 extensions (DVB-S2X)*.
- 4 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

See Also

Objects

`dvbs2xWaveformGenerator` | `dvbs2WaveformGenerator`

Related Examples

- “DVB-S2X Super-Frame Generation for Formats 0 and 1” on page 2-25
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames” on page 4-69
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames” on page 4-54
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode” on page 4-87

RF Propagation and Channel Models

Simulate and Visualize Land Mobile-Satellite Channel

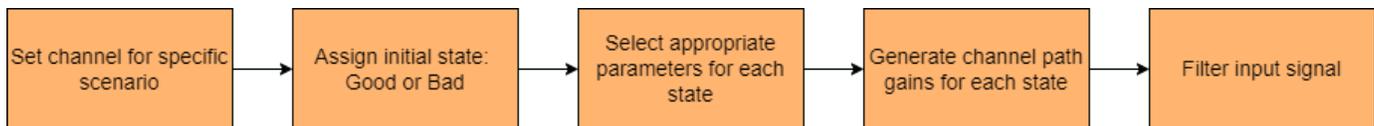
This example shows how to model a two-state land mobile-satellite (LMS) channel model by generating a state series, its respective space series, and the channel coefficients. In a scenario involving a satellite terminal and a mobile terminal, a signal being transmitted through the channel does not always have an ideal line-of-sight path. In some cases, the signal experiences phenomena such as Doppler shift, shadowing, and multipath fading. Appropriately modeling the effects of such phenomena is essential to properly design end-to-end communication links that are able to handle and compensate the effects of the channel.

Introduction

An LMS channel model is used to simulate the channel envelope that is observed in a satellite-to-ground channel. Given the moving nature of the terminals, the channel envelope experiences variations due to movement of the transmitting and receiving terminals, blockage due to buildings and foliage, shadowing, and multipath.

This example models ITU-R P.681-11 LMS channel and Lutz LMS channel by using a two-state semi-Markov chain, where the channel alternates between a good and bad state. A *good state* is characterized by either line-of-sight conditions or partial shadowing conditions, whereas a *bad state* is characterized by either severe shadowing conditions or complete blockage. Both the channel models are used for a single geostationary satellite. An ITU-R P.681-11 LMS channel uses Loo distribution in both good and bad state, whereas Lutz LMS channel uses Rician distribution in good state and Rayleigh with log-normal distribution in bad state. The data sets provided for ITU-R P.681-11 channel in [1 on page 3-7] are applicable for frequency range of 1.5 GHz to 20 GHz, and the data set provided for Lutz LMS channel in [2 on page 3-7] is applicable for frequency of 1.54 GHz (L-band).

The following block diagram shows the procedure to model the LMS channel:



Channel Setup

Set up a channel between a satellite terminal and a mobile terminal on the ground using either `p681LMSChannel` or `lutzLMSChannel` System object.

Set these parameters to model a specific scenario using ITU-R P.681-11 LMS channel:

- Environment
- Carrier frequency
- Elevation angle
- Speed of the ground terminal
- Azimuth orientation of the ground terminal

Set these parameters to model a specific scenario using Lutz LMS channel:

- Rician K-factor

- Lognormal fading parameters
- State duration distribution
- Mean state duration
- Maximum Doppler shift

In addition to a specific scenario modeling, the LMS channel requires defining these parameters:

- Sampling rate of input signal
- Initial state of the channel
- Fading technique

By default, the example selects an ITU-R P.681-11 LMS channel and configures the channel to an urban scenario with 3.8 GHz carrier frequency having a mobile terminal moving at a speed of 2 m/s.

```
% Create an ITU-R P.681-11 channel or Lutz LMS channel
chan = p681LMSChannel ;
% Set channel properties
if isa(chan, "p681LMSChannel")
    % For ITU-R P.681 LMS channel

    % Environment type
    chan.Environment = "Urban";
    % Carrier frequency (in Hz)
    chan.CarrierFrequency = 3.8e9;
    % Elevation angle with respect to ground plane (in degrees)
    chan.ElevationAngle = 45;
    % Speed of movement of ground terminal (in m/s)
    chan.MobileSpeed = 2;
    % Direction of movement of ground terminal (in degrees)
    chan.AzimuthOrientation = 0;
else
    % For Lutz LMS channel

    % Rician K-factor (in dB)
    chan.KFactor = 5.5;
    % Lognormal fading parameters (in dB)
    chan.LogNormalFading = [-13.6 3.8];
    % State duration distribution

    chan.StateDurationDistribution = Exponential ;
    % Mean state duration (in seconds)
    chan.MeanStateDuration = [21 24.5];
    % Maximum Doppler shift (in Hz)
    chan.MaximumDopplerShift = 2.8538;
end
% Sampling rate (in Hz)
chan.SampleRate = 400;
```

Assign a suitable initial state for the channel.

```
chan.InitialState = Good ;
```

Set the fading technique used to realize the Doppler spectrum. The fading technique is either "Filtered Gaussian noise" or "Sum of sinusoids". When FadingTechnique property is set to "Sum of sinusoids", you can also set the number of sinusoids through NumSinusoids property.

```
chan.FadingTechnique = "Filtered Gaussian noise";
```

Initialize random number generator with seed. Vary the seed to obtain different channel realizations. The default value 73 is an arbitrary value.

```
seed = 73;  
chan.RandomStream = "mt19937ar with seed";  
chan.Seed = seed;
```

Display the properties of the channel.

```
disp(chan)
```

```
p681LMSChannel with properties:  
  
    SampleRate: 400  
    InitialState: "Good"  
    CarrierFrequency: 3.8000e+09  
    ElevationAngle: 45  
    MobileSpeed: 2  
    AzimuthOrientation: 0  
    Environment: "Urban"  
    ChannelFiltering: true
```

Use get to show all properties

Channel Model

Generate the channel for a duration of 100 seconds. Use random samples as input waveform.

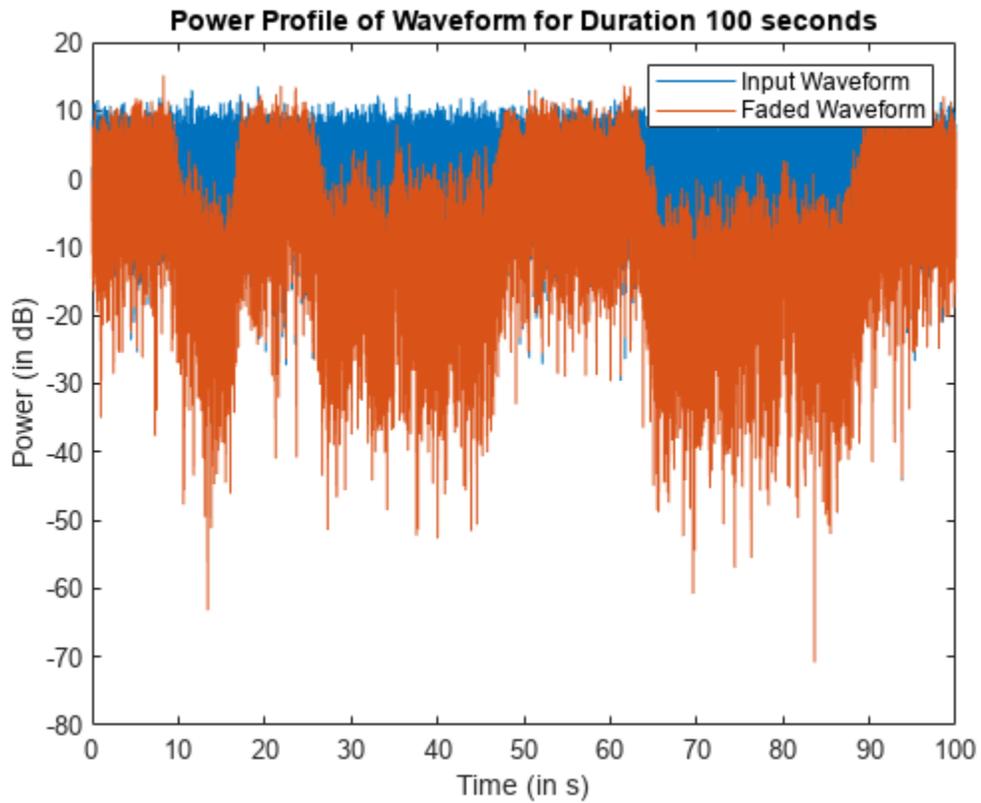
```
% Set random number generator with seed  
rng(seed);  
% Channel duration (in seconds)  
chanDur = 100;  
% Random input waveform  
numSamples = floor(chan.SampleRate*chanDur)+1;  
in = complex(randn(numSamples,1),randn(numSamples,1));  
% Pass the input signal through channel  
[fadedWave,channelCoefficients,sampleTimes,stateSeries] = step(chan,in);
```

Channel Visualization

Visualize the power profile, the space series, and the state series generated as part of channel modeling.

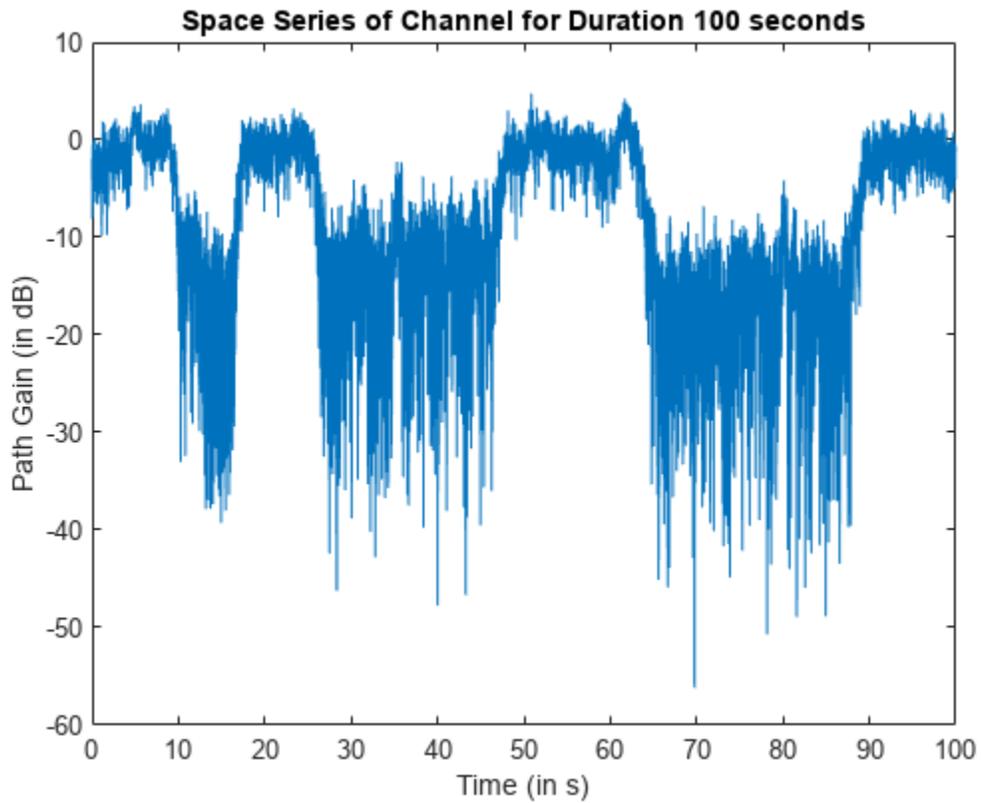
Plot the power profile of input waveform and the faded waveform.

```
figure(1)  
plot(sampleTimes,20*log10(abs(in)),sampleTimes,20*log10(abs(fadedWave)))  
title(['Power Profile of Waveform for Duration ' num2str(chanDur) ' seconds'])  
legend('Input Waveform', 'Faded Waveform')  
xlabel('Time (in s)')  
ylabel('Power (in dB)')
```



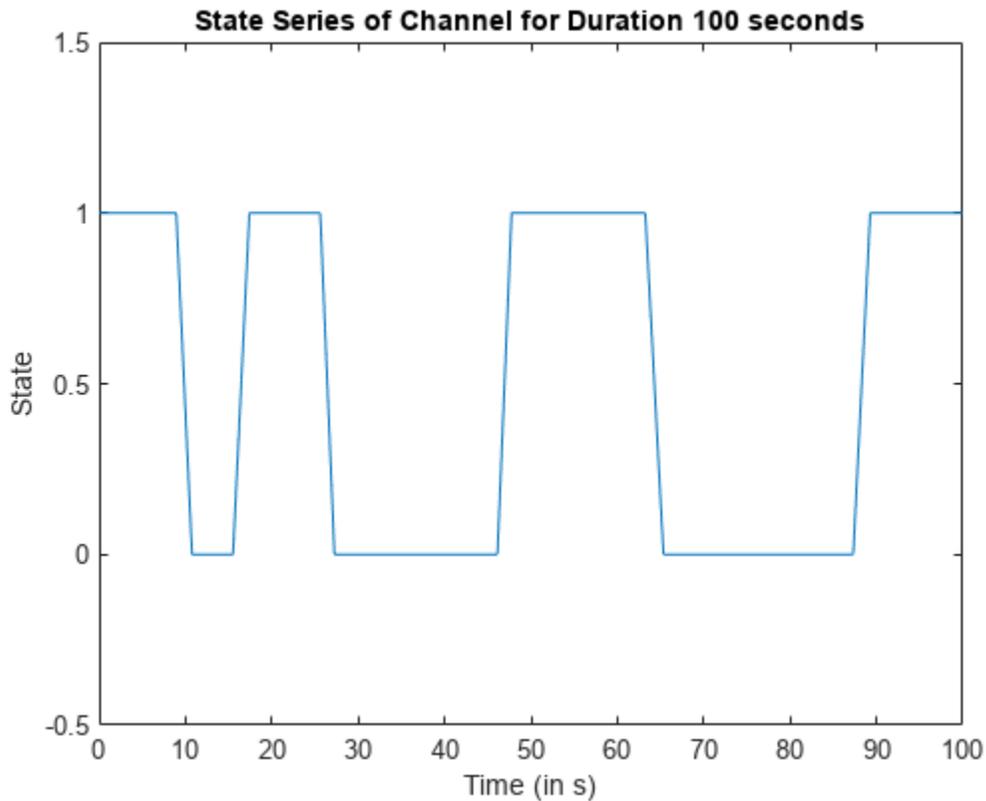
Plot the space series to show how the instantaneous power of the channel envelope varies with time.

```
figure(2)
plot(sampleTimes,20*log10(abs(channelCoefficients)))
title(['Space Series of Channel for Duration ' num2str(chanDur) ' seconds'])
xlabel('Time (in s)')
ylabel('Path Gain (in dB)')
```



Plot the state series to show how the channel state varies with time.

```
figure(3)
plot(sampleTimes,stateSeries)
title(['State Series of Channel for Duration ' num2str(chanDur) ' seconds'])
axis([0 sampleTimes(end) -0.5 1.5])
xlabel('Time (in s)')
ylabel('State')
```



Further Exploration

This example uses either `p681LMSChannel` or `lutzLMSChannel` System object to generate the two-state LMS channel for the defined channel properties. You can modify the properties of the System object to observe the variations with respect to time in the power profile, channel coefficients, and state series. To model the ITU-R P.681 channel for different frequency bands, you can set the parameters related to any of the data tables available in ITU-R P.681-11 Recommendation Section 3.1 Annexure 2 [1 on page 3-7]. You can also set the ITU-R P.681 LMS channel to custom environment with any other data set available. To model the Lutz LMS channel for different scenarios, you can use the data table present in [2 on page 3-7].

References

[1] ITU-R Recommendation P.681-11 (08/2019). "Propagation data required for the design systems in the land mobile-satellite service." P Series; Radio wave propagation.

[2] E. Lutz, D. Cygan, M. Dippold, F. Dolainsky, and W. Papke, "The land mobile-satellite communication channel-recording, statistics, and channel model", *IEEE Trans. Veh. Technol.*, vol 40, no. 2, pp. 375-386, 1991.

Model NR NTN Channel

This example shows how to model two types of New Radio (NR) non-terrestrial network (NTN) channels: a flat fading narrowband channel and a frequency selective fading tapped delay line (TDL) channel. These channels are defined in 3GPP TR 38.811 Section 6.7.1. and Section 6.9.2, respectively [1 on page 3-15]. These channels are used to model different NTN deployment scenarios, covering both geo-synchronous orbit (GSO) satellites and non geo-synchronous orbit (NGSO) satellites.

Introduction

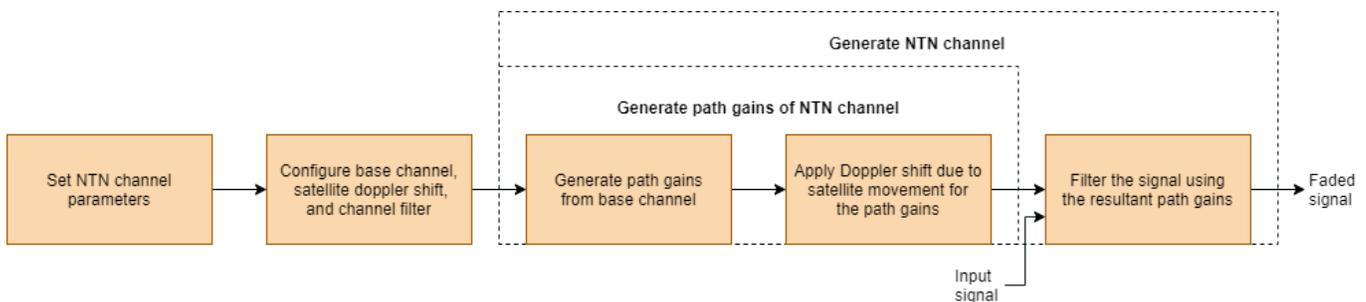
In this example, you generate the NTN channel model by generating path gains from a base channel model and then apply the Doppler shift due to satellite movement. The base channel model for the NTN flat fading narrowband channel is implemented per ITU-R P.681-11, which defines the propagation data for a land mobile-satellite (LMS) channel [4 on page 3-15]. The base channel model for the NTN frequency selective TDL channel is implemented per 3GPP TR 38.901, which defines the terrestrial TDL channel [2 on page 3-15].

The Doppler shift due to satellite movement depends on the satellite speed, satellite orbit, elevation angle, and carrier frequency. The Doppler shift due to satellite motion, $f_{d, \text{sat}}$, as defined in 3GPP TR 38.811 Section 6.7.1 and Section 6.9.2 [1 on page 3-15], is:

$$f_{d, \text{sat}} = \left(\frac{\nu_{\text{sat}}}{c} \right) * \left(\frac{R}{R+h} \cos(\alpha_{\text{model}}) \right) * f_c$$

- ν_{sat} is the satellite speed.
- c is the speed of light.
- R is the earth radius.
- h is the satellite altitude.
- α_{model} is the satellite elevation angle.
- f_c is the carrier frequency.

To model an NTN narrowband or TDL channel, you need to set all the required channel parameters in addition to the parameters used for the base channel model. After defining all the channel parameters, construct a suitable base channel for the flat fading narrowband channel or frequency selective TDL channel. Generate the path gains from the base channel and apply a Doppler shift due to satellite movement to get the path gains of the NTN channel. Then, filter the input signal with the resultant path gains. The figure shows the workflow for generating the NTN channel.



Set NTN Channel Common Parameters

Set the common parameters required to model an NTN narrowband flat fading channel and NTN TDL channel. This example models a low Earth orbit (LEO) satellite moving at a speed of 7.5622 km/s and an altitude of 600 km that operates in the S-band. Assume a mobile or UE speed of 3 km/hr. These default parameters are from 3GPP TR 38.821 Table 6.1.2-4 [3 on page 3-15].

```
commonParams = struct;
commonParams.CarrierFrequency = 2e9;           % In Hz
commonParams.ElevationAngle = 50;             % In degrees
commonParams.SatelliteAltitude = 600000;     % In m
commonParams.SatelliteSpeed = 7562.2;        % In m/s
commonParams.MobileSpeed = 3*1000/3600;       % In m/s
commonParams.SampleRate = 7680000;           % In Hz
% Set the random stream and seed, for reproducibility

commonParams.RandomStream = mt19937ar with seed ;
commonParams.Seed = 73;
% Set the number of sinusoids used in generation of Doppler spread
commonParams.NumSinusoids = 48;
```

NTN Narrowband Channel

This example supports all the land-mobile satellite (LMS) scenarios available for the NTN flat fading narrowband channel, as defined in 3GPP TR 38.811 Section 6.7.1 [1 on page 3-15]. The LMS scenarios defined for S-band are:

- Urban
- Suburban
- RuralWooded
- Residential

The LMS scenarios defined for Ka-band are:

- Suburban
- RuralWooded

Follow these steps to model the NTN flat fading narrowband channel, as specified in 3GPP TR 38.811 Section 6.7.1 [1 on page 3-15].

- 1 Set the channel parameters specific to an NTN flat fading narrowband channel.
- 2 Generate the NTN flat fading narrowband channel.
- 3 Visualize the spectrum of the faded or filtered signal.

The NTN flat fading channel is used for narrowband single-input-single-output (SISO) simulations.

Set NTN Narrowband Channel Parameters

Set the NTN flat fading narrowband channel parameters using the common parameters in a new structure. This example configures an urban LMS scenario for the NTN narrowband channel.

```
% Initialize the NTN flat fading narrowband channel parameters in a
% structure
ntnNarrowbandParams = commonParams;
ntnNarrowbandParams.NTNChannelType = "Narrowband";
```

```

ntnNarrowbandParams.Environment = Urban;
ntnNarrowbandParams.AzimuthOrientation = 0;
ntnNarrowbandParams.FadingTechnique = Sum of sinusoids;

% Set the below parameters when Environment is set to Custom
ntnNarrowbandParams.StateDistribution = [3.0639 2.9108; 1.6980 1.2602];
ntnNarrowbandParams.MinStateDuration = [10 6];
ntnNarrowbandParams.DirectPathDistribution = [-1.8225 -15.4844; 1.1317 3.3245];
ntnNarrowbandParams.MultipathPowerCoefficients = [-0.0481 0.9434; -14.7450 -1.7555];
ntnNarrowbandParams.StandardDeviationCoefficients = [-0.4643 -0.0798; 0.3334 2.8101];
ntnNarrowbandParams.DirectPathCorrelationDistance = [1.7910 1.7910];
ntnNarrowbandParams.TransitionLengthCoefficients = [0.0744; 2.1423];
ntnNarrowbandParams.StateProbabilityRange = [0.05 0.1; 0.95 0.9];

```

Get these building blocks of NTN flat fading narrowband channel by using the `ntnNarrowbandParams` structure and `HelperSetupNTNChannel` function.

- Base channel ITU-R P.681-11 LMS System object™ with `ChannelFiltering` set to false
- Doppler shift due to satellite movement
- Channel filtering System object (`comm.ChannelFilter`) to get the filtered or faded waveform

```
ntnNarrowbandChan = HelperSetupNTNChannel(ntnNarrowbandParams)
```

```

ntnNarrowbandChan = struct with fields:
    ChannelName: "NTN narrowband with Urban environment"
    BaseChannel: [1x1 p681LMSChannel]
    ChannelFilter: [1x1 comm.ChannelFilter]
    SatelliteDopplerShift: 2.9637e+04
    MobileDopplerSpread: 5.5594

```

Get information about the P681-11 LMS base channel model and check that the channel filter delay is 0 due to the flat fading nature of the channel.

```
p681ChannelInfo = info(ntnNarrowbandChan.BaseChannel)
```

```

p681ChannelInfo = struct with fields:
    PathDelays: 0
    ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
    NumSamplesProcessed: 0

```

Generate NTN Narrowband Channel

Generate the path gains of NTN flat fading narrowband channel using the base channel ITU-R P.681-11 System object and Doppler shift due to the satellite movement. Then apply channel filtering to a random input signal using the resultant path gains.

```

% Generate a random input
rng(commonParams.Seed);
in = complex(randn(commonParams.SampleRate,1), ...
    randn(commonParams.SampleRate,1));

```

Generate the faded waveform for the NTN flat fading narrowband channel.

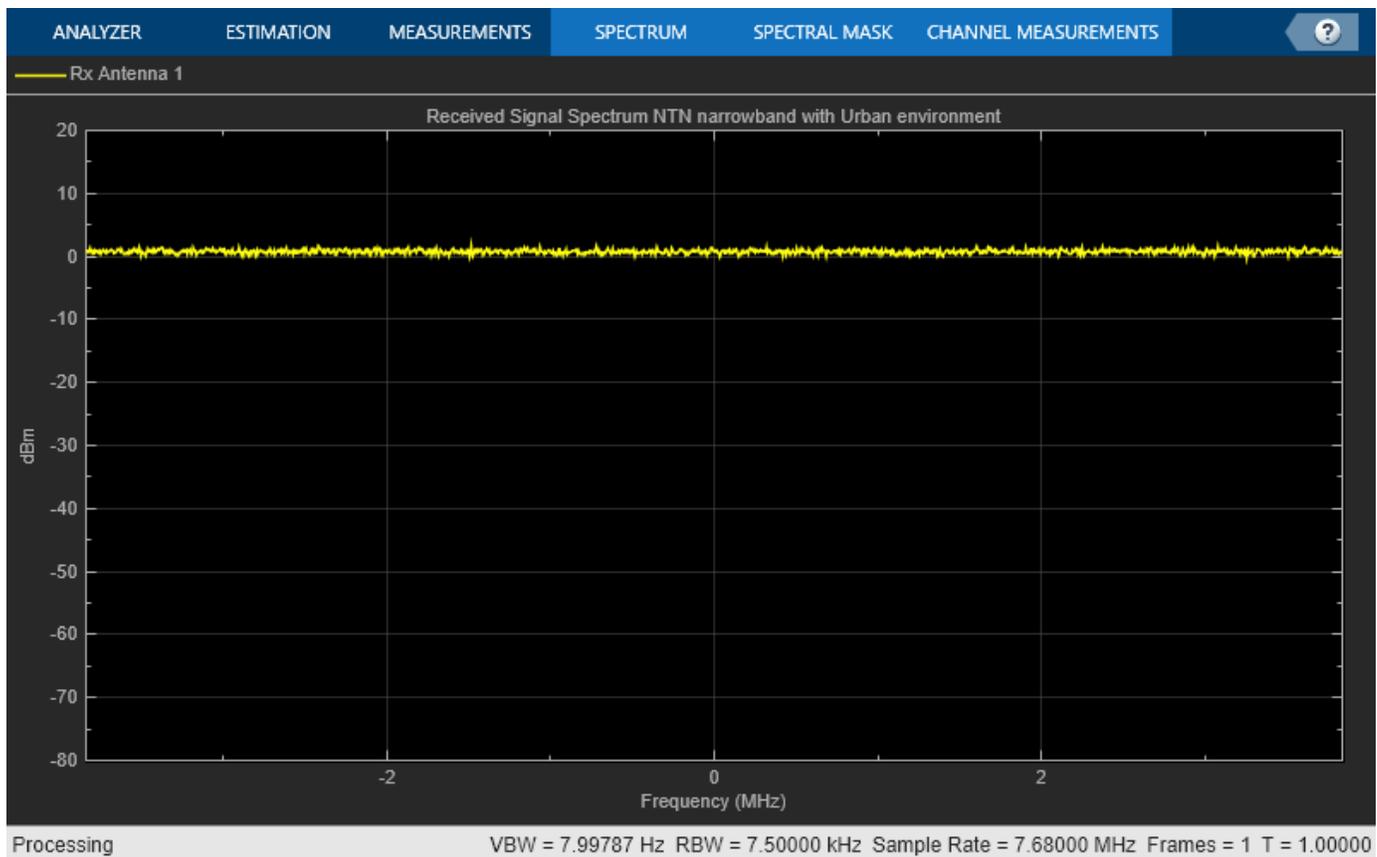
```
[narrowbandOut,narrowbandPathGains,narrowbandSampleTimes] = ...
    HelperGenerateNTNChannel(ntnNarrowbandChan,in);
```

Note that the state is maintained in the `BaseChannel` and `ChannelFilter` fields of the `ntnNarrowbandChan` structure. To realize the same outputs without reconstructing both the base channel and channel filter System objects, set the `RandomStream` property of the base channel to "mt19937ar with seed" and call the `reset` method of both the System objects.

Visualize NTN Narrowband Channel Received Spectrum

Plot the received spectrum of the faded signal from the NTN flat fading narrowband channel.

```
ntnNarrowbandAnalyzer = dsp.SpectrumAnalyzer( ...
    'SampleRate',ntnNarrowbandParams.SampleRate, ...
    'AveragingMethod','Exponential','ForgettingFactor',0.99);
ntnNarrowbandAnalyzer.Title = "Received Signal Spectrum " ...
    + ntnNarrowbandChan.ChannelName;
ntnNarrowbandAnalyzer.ShowLegend = true;
ntnNarrowbandAnalyzer.ChannelNames{1} = 'Rx Antenna 1';
ntnNarrowbandAnalyzer(narrowbandOut)
```



NTN TDL Channel

This example supports all the four channel profiles of the NTN TDL channel, defined in 3GPP TR 38.811 Section 6.9.2 [1 on page 3-15]. These four channel profiles are:

- NTN-TDL-A
- NTN-TDL-B
- NTN-TDL-C
- NTN-TDL-D

The channel profiles NTN-TDL-A and NTN-TDL-B are defined for non-line-of-sight (NLOS) conditions. The channel profiles NTN-TDL-C and NTN-TDL-D are defined for line-of-sight (LOS) conditions. All four channel profiles are defined at an elevation angle of 50 degrees. The main differences between the NTN TDL channel [1 on page 3-15] and terrestrial TDL channel [2 on page 3-15] are:

- NTN TDL channel accounts for Doppler shift due to satellite motion in addition to the Doppler shift due to mobile or user equipment (UE) movement.
- NTN TDL channel delay profiles are different from the terrestrial TDL channel profiles, due to large propagation delays and different scattering environment.

Follow these steps to model the NTN frequency selective fading TDL channel, as specified in 3GPP TR 38.811 Section 6.9.2 [1 on page 3-15].

- 1 Set the channel parameters specific to the NTN frequency selective fading TDL channel.
- 2 Generate the NTN frequency selective fading TDL channel.
- 3 Visualize the spectrum of the faded or filtered signal.

Set NTN TDL Channel Parameters

Set the NTN frequency selective fading TDL channel parameters using the common parameters in a new structure.

```
% Initialize the NTN TDL channel parameters in a structure
ntnTDLParams = commonParams;
ntnTDLParams.NTNChannelType = "TDL";

ntnTDLParams.DelayProfile = NTN-TDL-A;
ntnTDLParams.DelaySpread = 30e-9; % In s

ntnTDLParams.TransmissionDirection = Downlink;
ntnTDLParams.MIMOCorrelation = Low;
ntnTDLParams.Polarization = Co-Polar;
% Modify the below parameters, when DelayProfile is set to Custom
ntnTDLParams.PathDelays = 0; % In s
ntnTDLParams.AveragePathGains = 0; % In dB

ntnTDLParams.FadingDistribution = Rayleigh;

% Set the antenna configuration
% Modify the below parameters, when MIMOCorrelation is set to a value other
% than Custom
ntnTDLParams.NumTransmitAntennas = 1;
ntnTDLParams.NumReceiveAntennas = 2;
% Modify the below parameters, when MIMOCorrelation is set to Custom and
% Polarization is set to Co-Polar or Cross-Polar
ntnTDLParams.TransmitCorrelationMatrix = 1;
ntnTDLParams.ReceiveCorrelationMatrix = [1 0; 0 1];
% Modify the below parameters, when MIMOCorrelation is set to Custom and
```

```

% Polarization is set to Cross-Polar
ntnTDLParams.TransmitPolarizationAngles = [45 -45];           % In degrees
ntnTDLParams.ReceivePolarizationAngles = [90 0];             % In degrees
ntnTDLParams.XPR = 10;                                       % In dB
% Modify the below parameters, when both MIMOCorrelation and Polarization
% are set to Custom
ntnTDLParams.SpatialCorrelationMatrix = [1 0; 0 1];

```

Get these building blocks of NTN frequency selective fading TDL channel by using the `ntnTDLParams` structure and `HelperSetupNTNChannel` function.

- Base channel terrestrial TDL System object with `DelayProfile` set to "Custom" and `ChannelFiltering` set to false
- Doppler shift due to satellite movement
- Channel filtering System object (`comm.ChannelFilter`) to get the filtered or faded waveform

```

ntnTDLChan = HelperSetupNTNChannel(ntnTDLParams)

ntnTDLChan = struct with fields:
    ChannelName: "NTN TDL with NTN-TDL-A delay profile"
    BaseChannel: [1x1 nrTDLChannel]
    ChannelFilter: [1x1 comm.ChannelFilter]
    SatelliteDopplerShift: 2.9637e+04
    MobileDopplerSpread: 5.5594

```

Check that the channel is configured for the defined NTN channel delay profile and delay spread by calling the object function `info` to observe the path delays, average path gains, and K factor first tap value.

```

tdlChanInfo = info(ntnTDLChan.BaseChannel)

tdlChanInfo = struct with fields:
    ChannelFilterDelay: 7
    MaximumChannelDelay: 8
    PathDelays: [0 3.2433e-08 8.5248e-08]
    AveragePathGains: [0 -4.6750 -6.4820]
    KFactorFirstTap: -Inf
    NumTransmitAntennas: 1
    NumReceiveAntennas: 2
    SpatialCorrelationMatrix: [2x2 double]

```

Generate NTN TDL Channel

Generate the path gains of the NTN frequency selective fading TDL channel using the base channel terrestrial TDL System object and the Doppler shift due to the satellite movement. Then apply channel filtering to a random input signal using the resultant path gains.

```

% Generate a random input
rng(commonParams.Seed);
in = complex(randn(commonParams.SampleRate,tdlChanInfo.NumTransmitAntennas), ...
    randn(commonParams.SampleRate,tdlChanInfo.NumTransmitAntennas));
% Generate the faded waveform for NTN TDL channel
[tdlOut,tdlPathGains,tdlSampleTimes] = HelperGenerateNTNChannel(ntnTDLChan,in);

```

Note that the state is maintained in the `BaseChannel` and `ChannelFilter` fields of the `ntnTDLChan` structure. To realize the same outputs without reconstructing both the base channel

and channel filter System objects, set the RandomStream property of the base channel to "mt19937ar with seed" and call the reset method of both the System objects.

Visualize NTN TDL Channel Received Spectrum

Plot the received spectrum of the faded signal from the NTN frequency selective fading TDL channel.

```
ntnTDLAnalyzer = dsp.SpectrumAnalyzer('SampleRate',ntnTDLParams.SampleRate, ...
    'AveragingMethod','Exponential','ForgettingFactor',0.99);
ntnTDLAnalyzer.Title = "Received Signal Spectrum " ...
    + ntnTDLChan.ChannelName;
ntnTDLAnalyzer.ShowLegend = true;
for nRx = 1:size(tdlOut,2)
    ntnTDLAnalyzer.ChannelNames{nRx} = ['Rx Antenna ' num2str(nRx)];
end
ntnTDLAnalyzer(tdlOut)
```



Further Exploration

You can use this example to further explore these options:

- To configure and analyze the NTN narrowband or TDL channels for other satellite orbits, change the SatelliteAltitude and SatelliteSpeed fields of the commonParams structure.
- To analyze the NTN flat fading narrowband channel for another environment, change the Environment field of the ntnNarrowbandParams structure.

- To analyze the NTN TDL channel for LOS delay profiles, change the `DelayProfile` field of the `ntnTDLParams` structure to `NTN-TDL-C` or `NTN-TDL-D`.
- To configure the NTN TDL channel for multiple antennas, change the `MIMOCorrelation` and `Polarization` fields of the `ntnTDLParams` structure. You might need to set other fields depending on these values.
- Try using the two NTN channel models used in the example in a link simulation and compute the link metric. For more information, see the “NR NTN PDSCH Throughput” on page 4-147 example.

Appendix

The example uses these helper functions:

- `HelperGenerateNTNChannel` — Generate NTN channel
- `HelperSetupNTNChannel` — Set up NTN channel

References

[1] 3GPP TR 38.811. "Study on new radio (NR) to support non-terrestrial networks." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[2] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[3] 3GPP TR 38.821. "Solutions for NR to support non-terrestrial networks (NTN)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[4] ITU-R Recommendation P.681-11 (08/2019). "Propagation data required for the design systems in the land mobile-satellite service." P Series; Radio wave propagation.

See Also

Related Examples

- “NR NTN PDSCH Throughput” on page 4-147

Antenna Size Analysis Using ITU-R P.618 Propagation Model

Introduction

This example shows how to select a parabolic antenna diameter for a particular ground location using the ITU P.618 [1] propagation model function in the Satellite Communications Toolbox. The appropriate antenna size may be quite sensitive to local climate conditions, such that locations with rainy climates will require larger antennas than those with dry climates. The ITU-R P.618 recommendation provides a comprehensive model for attenuation, noise temperature, and depolarization of radio signals through the atmosphere for space-to-ground links in every area of the earth.

This example uses the Satellite Communications Toolbox P.618 propagation model function. The example particularly applies to low earth orbit (LEO) satellites as they traverse a range of elevation angles on each pass.

Obtain Map Data Required by P.618

Download and unpack ITU data maps if the zipped map file is not already present on the MATLAB® path. If the zipped map file already exists on the MATLAB path, just unpack it.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Specify System Parameters

Specify the range of ground antenna diameters and elevation angles to be analyzed. An outage percentage of 1% is chosen, which corresponds to an availability of 99%.

Singapore is chosen for the ground station site as it has a high rain rate when compared to the rest of the world.

Other parameters specified are satellite equivalent isotropic radiated power (EIRP), satellite altitude, satellite transmit frequency, ground station antenna efficiency, ground station radome loss, pointing loss, receiver temperature, implementation loss, required Es/No, and either single or dual polarization.

```
mgnParams.pctOutage = 1;           % Outage percentage;
                                   % Availability = (100 - p) percent
mgnParams.gndAntLat = 1.29;       % North latitude, deg
mgnParams.gndAntLon = 103.5;      % East longitude, deg
mgnParams.satEIRP = 27.0;         % Satellite EIRP, dBW
```

```

mgnParams.satAlt = 500;           % Satellite altitude, km
mgnParams.satTransmitFreq = 8.2; % Satellite transmit frequency, GHz
mgnParams.antEff = 0.631;        % Ground station antenna efficiency
mgnParams.radomeLoss = 0.74;     % Radome loss, dB
mgnParams.ptgLoss = 0.2;        % Pointing loss, dB
mgnParams.rcvrTemp = 100;       % Receiver noise temperature, K
mgnParams.symRate = 100e6;      % Symbol rate, sym/sec
mgnParams.implLoss = 2.5;       % Implementation loss, dB
mgnParams.EsNoRqd = 17.8;      % Required Es/No for BER=1e-5 in
                                % AWGN-only channel, dB
                                % 12.6 for QPSK, 20.6 for D8PSK,
                                % 17.8 dB for 8PSK

gndAntDiam = [3 5 7 9];         % Ground antenna diameters, m
gndAntEl = [5 10 15 20];      % Elevation angles, deg

satAntPol = ; % Antenna polarization

% Initialize outputs
[marginSinglePol,marginDualPol] = deal(zeros(size(gndAntDiam)));
linkMarginVsElAng = zeros(length(gndAntDiam),length(gndAntEl));
lgndEntries = repmat("",numel(gndAntEl)*2,1); % *2 for clear sky and rain

```

Run the Analysis

This code section calls `HelperLinkMarginVsEl`, which calls the MATLAB function `p618PropagationLosses` for each set of elevation angles and antenna diameters. These results are then plotted to clearly illustrate the link margins available for various configurations and geometries.

Note that both *clear sky* and *with rain* results are produced. This plot helps to clearly distinguish the performance under the best and worst case rain conditions for the given location and availability percentage.

```

for clrSky = ["y" "n"]
    if clrSky == "y"
        clrSkyMkr = "--o";
        lgndSkyStr = "clear sky";
        lgndIdx = 0;
    else % "n"
        clrSkyMkr = "-o";
        lgndSkyStr = "w/rain";
        lgndIdx = 1;
    end
    numAntEl = length(gndAntEl);
    for idxEl = 1:numAntEl
        for idxDiam = 1:length(gndAntDiam)
            [marginSinglePol(idxDiam),marginDualPol(idxDiam)] = ...
                HelperLinkMarginVsEl(mgnParams,gndAntDiam(idxDiam), ...
                    gndAntEl(idxEl),clrSky);
        end
        if satAntPol == "Dual"
            margin = marginDualPol;
        elseif satAntPol == "Single"
            margin = marginSinglePol;
        end
        plot(gndAntDiam,margin,clrSkyMkr);
        xlabel("Antenna Diameter, m"); ylabel("Link Margin, dB");
        title("Link Margin vs. Antenna Diameter, " + satAntPol + ...
            " Polarization");
    end
end

```

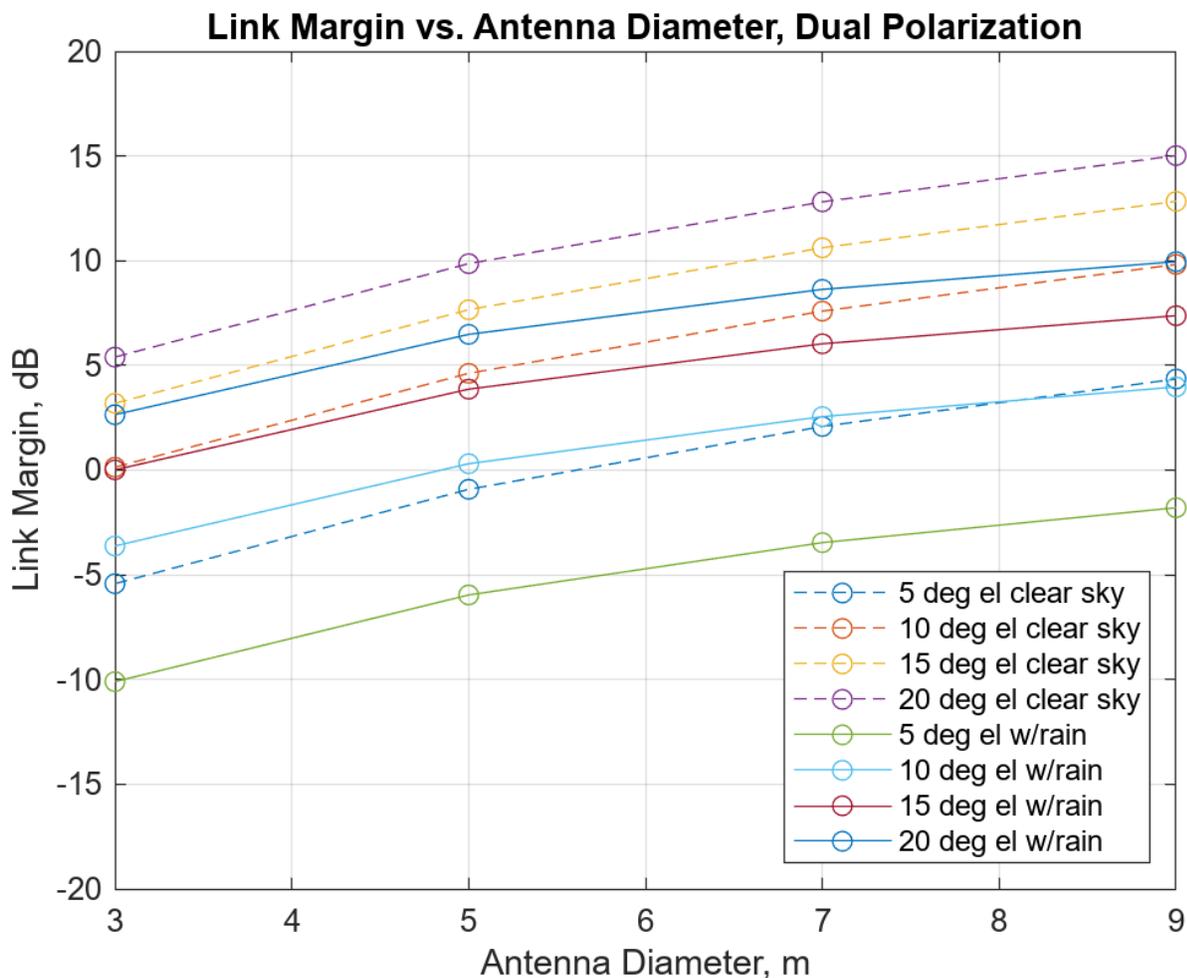
```

hold on;
lgndEntries(numAntEl*lgndIdx+idxEl) = ...
    string(gndAntEl(idxEl)) + " deg el " + lgndSkyStr;

% Accumulate columns into a matrix
linkMarginVsElAng(:,idxEl) = margin;
end

end
legend(lgndEntries(1:end), ...
    "Location","Southeast");
grid on;
ylim([-20 20]);

```



For low elevation angles (5 and 10 degrees), it is difficult to close the link with an antenna size of less than 6 meters when rain falls at an annual 1 percent exceedance rate. For clear sky situations (no rain present), a 6 meter antenna would have almost zero margin at 5 degrees elevation. For additional margin, therefore, a diameter larger than 6 meters must be chosen.

Generate Tabular Output

A tabular output of the matrix of link margins from the *with rain* analysis is computed and displayed. Each row corresponds to the analysis of one antenna diameter. Each column corresponds to the link margins at a given elevation angle for the various antenna diameters.

A possible application for such a table would be finding a polynomial fit for the margins at each elevation angle which are indicated by the columns in the table. Then the antenna diameter at each elevation angle for a particular margin (for example, 3 dB) value could be found. This assists in choosing a minimum elevation angle for a particular choice of antenna diameter.

```
AntennaDiameter = gndAntDiam';
Margin5Deg = linkMarginVsElAng(:,1);
Margin10Deg = linkMarginVsElAng(:,2);
Margin15Deg = linkMarginVsElAng(:,3);
Margin20Deg = linkMarginVsElAng(:,4);
T = table(AntennaDiameter,Margin5Deg,Margin10Deg,Margin15Deg,Margin20Deg)
```

T=4×5 table

AntennaDiameter	Margin5Deg	Margin10Deg	Margin15Deg	Margin20Deg
3	-10.101	-3.6207	0.012847	2.6489
5	-5.9608	0.30741	3.8668	6.4839
7	-3.4614	2.5567	6.0346	8.6315
9	-1.7944	3.9742	7.3766	9.9556

Further Exploration

This example analyzes the location of Singapore and a satellite in a 500 km altitude circular orbit. Adjust the parameters in the function call to the helper function `HelperLinkMarginVsEl` and explore different ground station locations with different environmental parameters. These parameters are computed automatically by the MATLAB `p618PropagationLosses` function, or you can adjust any of the P.618 parameters manually. You can also adjust the altitude of the satellite, the carrier frequency, satellite EIRP, antenna diameters, system losses, receiver temperature, and required E_s/N_0 as desired.

The example showcases the dual polarization case. You can change the `satAntPol` variable to `single`, rerun the analysis, and compare the results to the dual polarized case.

Another important parameter is the annual exceedance percentage (p), which is the probability that a particular rain rate and atmospheric noise temperature will be exceeded. Annual exceedance percentage relates to availability (a) as

$$a = (100 - p) \%$$

For LEO systems, the availability value has a less concrete meaning than for the geostationary orbit (GSO) case, where the geometry between the ground station and satellite is fixed. But as a qualitative measure, setting the availability to 99% is a typical value to use when assessing ground sites with moderate rain rates.

The example could be modified to address the GSO case by using the Satellite Communications Toolbox function `aer` in conjunction with a satellite scenario to compute the slant range between a ground station and a GSO satellite at a particular longitude. In this case the `HelperLinkMarginVsEl` function would have to be modified accordingly.

Appendix

This example uses these helper functions:

- `HelperLinkMarginVsEl`: Compute link margin over a range of elevation values
- `HelperComputeSlantRangeForCircularLEO`: Compute slant range for a circular LEO orbit

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

See Also

Functions

`p618PropagationLosses`

Objects

`p618Config`

Related Examples

- “Earth-Space Propagation Losses”

End-to-End Simulation

End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) and number of communications link transmission units (CLTUs) lost in a Consultative Committee for Space Data Systems (CCSDS) telecommand (TC) link. The example adds radio frequency (RF) front-end impairments and additive white gaussian noise (AWGN) to the link.

Introduction

CCSDS TC generally is used for sending commands from a ground station to a spacecraft. CCSDS TC receivers are subjected to large frequency errors due to the frequency uncertainties in spacecraft receivers and the doppler frequency shift. To compensate large frequency offsets, the ground stations perform a carrier sweep in frequency or use an FFT-based acquisition at the spacecraft during satellite acquisition. This example shows how to add a 200 KHz frequency offset to the signal and use an FFT-based acquisition for the correction.

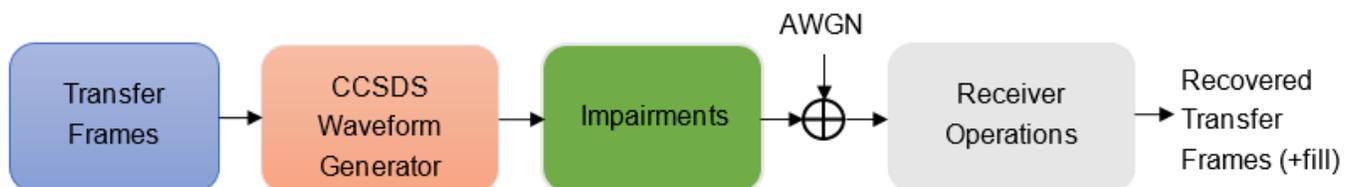
For each signal to noise ratio (SNR) point, CCSDS TC waveforms that are generated with a CLTU and acquisition sequence are distorted by RF impairments and passed through an AWGN channel. The example shows how to model these RF impairments:

- Carrier frequency and phase offset
- Subcarrier frequency and phase offset
- Timing phase offset

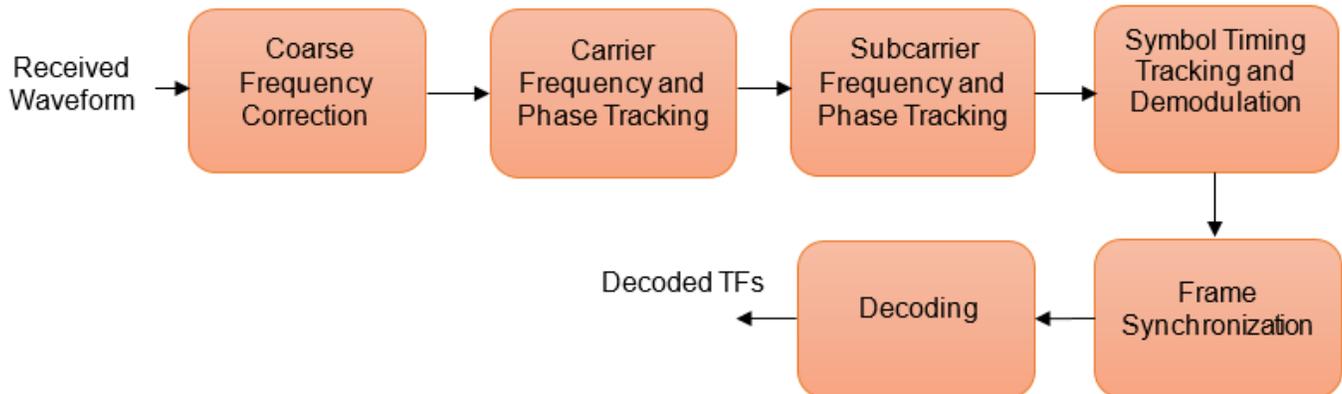
The CCSDS TC receiver compensates for the impairments, and the transfer frames (TFs) in the CLTUs are recovered. This example supports BPSK, PCM/PM/biphase-L, and PCM/PSK/PM modulation schemes. Subcarrier impairments are applicable only with the PCM/PSK/PM modulation scheme. These modulation schemes [8] on page 4-12 are used to generate the CCSDS TC waveform, in the form of baseband in-phase quadrature (IQ) samples.

- PCM/PSK/PM: The line coded signal as per the pulse code modulation (PCM) format is phase shift keying (PSK) modulated on a sine wave subcarrier and then phase modulated (PM) on a residual carrier.
- PCM/PM/biphase-L: Biphase-L (Manchester) encoded data is phase modulated on a residual carrier.
- BPSK: Suppressed carrier modulation by using non-return-to-zero (NRZ) data on the carrier.

This figure shows the processing steps involved in the recovery of transfer frames.



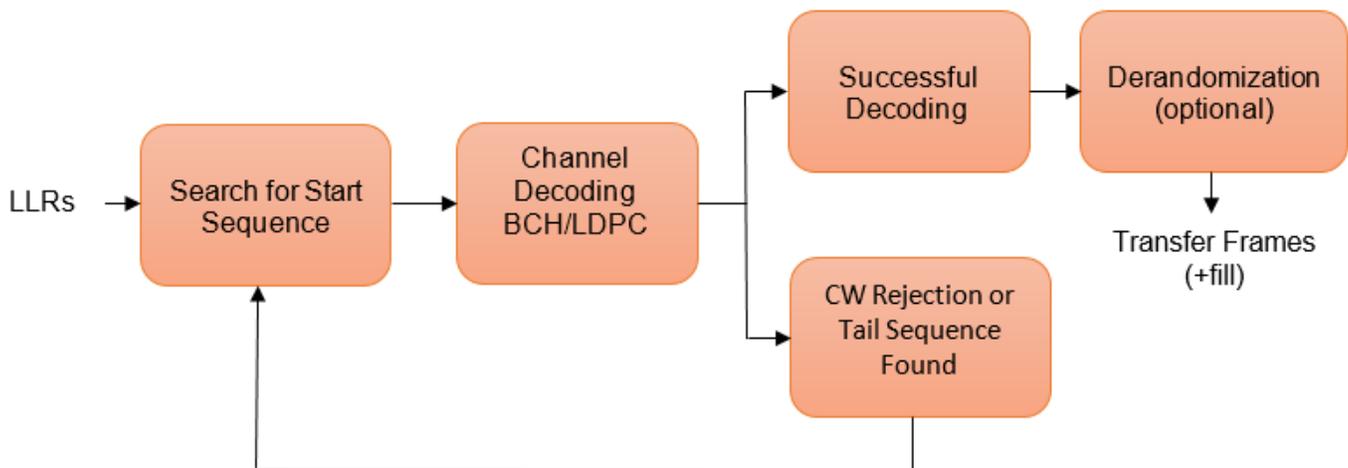
This figure shows the receiver operations, which include RF impairments compensation, demodulation, frame synchronization, and the decoding of transfer frames.



To recover the TFs from the received waveform, follow these steps.

- 1 Coarse frequency correction: Use the FFT-based algorithm to estimate the frequency offset.
- 2 Carrier frequency and phase tracking: Use the second order phase locked loop (PLL) [1] on page 4-12 for carrier tracking.
- 3 Subcarrier frequency and phase tracking: Use the second order Costas loop [1] on page 4-12 for subcarrier tracking.
- 4 Symbol timing tracking and demodulation: Use the second-order data transition tracking loop (DTTL) [3] on page 4-12 module for timing synchronization and symbol demodulation [1] on page 4-12.
- 5 Frame synchronization and decoding: Use a hard symbol based algorithm for Bose Chaudhuri Hocquenghem (BCH) code and soft symbol based algorithm for low density parity check (LDPC) code.

This figure shows the processing steps that are involved in frame synchronization and the decoding of TFs [4] on page 4-12.



- 1 Search for start sequence: When the channel coding is BCH, the incoming bit stream is searched bit by bit for the start sequence pattern. When the channel coding is LDPC, the incoming soft

symbols stream is searched with a soft correlator for the start sequence pattern. For BCH, the permissible number of errors in the start sequence is 0 or 1 (depending on the decoding mode). In error detecting mode, the permissible number of errors in the start sequence is 0. In error correcting mode, the permissible number of errors in the start sequence is 1.

- 2 Decoding: When a start sequence is detected, the decoding operation begins. The codewords (CWs) are decoded and optionally derandomized.
- 3 CW rejection or tail sequence detection: If the decoder has any decoding failure or any uncorrected errors in the decoded output, data from this failed CW is not transferred to the data link sublayer operations. The CW is rejected, and the search for the start sequence restarts. If a tail sequence is present, search for the tail sequence to detect the end of the CLTU. For BCH decoding, the CW rejection method is employed. For LDPC, use the tail sequence correlation or the CW rejection. When no tail sequence is used, the search for the start sequence must resume at the beginning of the uncorrected CW. When a tail sequence is used, the search can resume at the end of the uncorrected CW.

Simulation Configuration

Configure the number of samples per symbol and symbol rate.

```
% Samples per symbol
% Due to the low symbol rate and 200 KHz frequency offset, a large value of
% 200 samples per symbol must be used as a default value with
% PCM/PSK/PM modulation. For BPSK and PCM/PM/biphase-L modulation, a
% default value of 20 samples per symbol is used (due to medium and high
% symbol rates).
sps = 20;
% Symbol rate
% The symbol rates specified in TC for each modulation are:
% - For PCM/PSK/PM modulation, the coded symbol rates are 4000, 2000, 1000,
%   500, 250, 125, 62.5, 31.25, 15.625, or 7.8125 symbols/s (as specified in
%   CCSDS TC recommendation [6]).
% - For PCM/PM/biphase-L modulation, the coded symbol rates are 8000, 16000,
%   32000, 64000, 128000, or 256000 symbols/s.
% - For BPSK modulation, the coded symbol rates are 1000, 2000, 4000, 8000,
%   16000, 32000, 64000, 128000, 256000, 512000, 1024000, or 2048000
%   symbols/s.
symbolRate = 2048000;
```

The DTTL for the symbol synchronization performs better with an even number of samples per symbol. For an odd number of samples per symbol, the timing error estimate is nonzero at the perfect tracking of timing offset. The nonzero timing error drags the DTTL away from the perfect tracking condition.

Configure and display the CCSDS TC transmission parameters.

```
cfg = ccsdsTCConfig;
cfg.ChannelCoding = "BCH";
cfg.Modulation = "BPSK";
cfg.ModulationIndex = 1.2; % Applicable with PCM/PSK/PM and PCM/PM/biphase-L. Supported range in
if strcmpi(cfg.Modulation,"PCM/PSK/PM")
    cfg.SymbolRate = symbolRate;
end
cfg.SamplesPerSymbol = sps

cfg =
    ccsdsTCConfig with properties:
```

```
DataFormat: "CLTU"
ChannelCoding: "BCH"
HasRandomizer: 1
Modulation: "BPSK"
```

Configure the receiver parameters.

```
normLoopBWCarrier = 0.005; % Normalized loop bandwidth for carrier synchronizer
normLoopBWSubcarrier = 0.00005; % Normalized loop bandwidth for subcarrier synchronizer
normLoopBWSymbol = 0.005; % Normalized loop bandwidth for symbol synchronizer
```

To reduce noise contribution in the loop, decrease the loop bandwidth. The pull-in range of the frequency offset is also reduced because of decrement in the loop bandwidth. When you use a small loop bandwidth in the synchronization modules, the acquisition takes a longer time to converge. To improve the performance at low SNRs, reduce the loop bandwidth and use a higher value for the acquisition sequence length. If the loops do not track the offsets, consider increasing the loop bandwidth to increase the pull-in range.

Simulation Parameters

This example executes two burst transmissions for a number of energy per symbol to noise power spectral density ratio (E_s/N_0) points. E_s/N_0 can be a vector or a scalar. For statistically valid BER results, run the simulation for at least 1000 number of transmissions.

```
numBurst = 2; % Number of burst transmissions
EsNodB = [8 8.5]; % Es/No in dB
SNRIn = EsNodB - 10*log10(sps); % SNR in dB from Es/No
```

Processing Chain

The distorted CCSDS TC waveform with acquisition sequence and a single CLTU is processed at a time. To synchronize the received data and recover the TFs, these processing steps occur.

- 1 Generate the bits in the TC TF.
- 2 Generate the TC waveform for the acquisition sequence with alternating ones and zeros.
- 3 Generate the CCSDS TC waveform for the TFs with random bits.
- 4 Apply pulse shaping using a square root raised cosine filter (applicable only with BPSK modulation).
- 5 Apply the subcarrier frequency and phase offset (applicable only with PCM/PSK/PM modulation).
- 6 Apply the carrier frequency and phase offset.
- 7 Apply the timing phase offset.
- 8 Pass the transmitted signal through an AWGN channel.
- 9 Correct the coarse frequency and phase offset.
- 10 Filter the received signal.
- 11 Correct the carrier frequency and phase offset.
- 12 Correct the subcarrier frequency and phase offset (applicable only with PCM/PSK/PM modulation).
- 13 Correct the timing offset and symbol demodulation.
- 14 Detect the start of CLTU and decode the TFs.

```

% Initialization of variables to store BER and number of CLTUs lost
bitsErr = zeros(length(SNRIn),1);
cltuErr = zeros(length(SNRIn),1);

% Square root raised cosine (SRRC) transmit and receive filter objects for BPSK
if strcmpi(cfg.Modulation,"BPSK")
    % SRRC transmit filter object
    txfilter = comm.RaisedCosineTransmitFilter;
    txfilter.RolloffFactor = 0.35; % Filter rolloff
    txfilter.FilterSpanInSymbols = 6; % Filter span
    txfilter.OutputSamplesPerSymbol = sps;
    % SRRC receive filter object
    rxfilter = comm.RaisedCosineReceiveFilter;
    rxfilter.RolloffFactor = 0.35; % Filter rolloff
    rxfilter.FilterSpanInSymbols = 6; % Filter span
    rxfilter.DecimationFactor = 1;
    rxfilter.InputSamplesPerSymbol = sps;
end

% Sample rate
if strcmpi(cfg.Modulation,"PCM/PM/biphase-L")
    % In CCSDS TC recommendation [6] section 2.2.7, coded symbol rates are
    % defined prior to biphase-L encoding.
    fs = 2*sps*symbolRate; % Biphase-L encoding has 2 symbols for each bit
else
    fs = sps*symbolRate;
end

for iSNR = 1:length(SNRIn)

    % Set the random number generator to default
    rng default

    % SNR value in the loop
    SNRdB = SNRIn(iSNR);

    % Initialization of error computing parameters
    totNumErrs = 0;
    numErr = 0;
    totNumBits = 0;
    cltuLost = 0;

    for iBurst = 1:numBurst

        % Acquisition sequence with 800 octets
        acqSeqLength = 6400;
        acqBits = repmat([0;1], 0.5*acqSeqLength, 1); % Alternating ones and zeros with zero as

        % CCSDS TC Waveform for acquisition sequence
        % Maximum subcarrier frequency offset specified in CCSDS TC is
        %  $\pm(2 \times 10^{-4}) \times f_{sc}$ , where  $f_{sc}$  is the subcarrier frequency
        subFreqOffset = 3.2; % Subcarrier frequency offset in Hz
        subPhaseOffset = 4; % Subcarrier phase offset in degrees
        % Frequency offset in Hz
        if strcmpi(cfg.Modulation,'PCM/PSK/PM')
            % Signal modulation along with subcarrier frequency and phase offset
            acqSymb = HelperCCSDSTCSubCarrierModulation(acqBits,cfg,subFreqOffset,subPhaseOffset);
        else

```

```

    % Signal modulation as per the specified scheme in CCSDS telecommand
    % Subcarrier impairments are not applicable with BPSK and PCM/PM/biphase-L
    cfg.DataFormat = 'acquisition sequence';
    acqSymb = ccsdsTCWaveform(acqBits,cfg);
    cfg.DataFormat = 'CLTU';
end

% CCSDS TC waveform for CLTU
transferFramesLength = 640; % Number of octets in the transfer frame
inBits = randi([0 1],transferFramesLength,1); % Bits in the TC transfer frame
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    % Encoded bits after TC synchronization and channel coding sublayer operations
    [~,encBits] = ccsdsTCWaveform(inBits,cfg);
    % Signal modulation along with subcarrier frequency and phase offset
    waveSymb = HelperCCSDSTCSubCarrierModulation(encBits,cfg,subFreqOffset,subPhaseOffset);
else
    waveSymb = ccsdsTCWaveform(inBits,cfg);
end

% CCSDS TC waveform with acquisition sequence and CLTU
waveform = [acqSymb;waveSymb];

% Transmit filtering for BPSK
if strcmpi(cfg.Modulation,'BPSK')
    % Pulse shaping using SRRC filter
    data = [waveform;zeros(txfilter.FilterSpanInSymbols,1)];
    txSig = txfilter(data);
else
    txSig = waveform;
end

% Add carrier frequency and phase offset
freqOffset = 200000; % Frequency offset in Hz
phaseOffset = 20; % Phase offset in degrees
if fs <= (2*(freqOffset+cfg.SubcarrierFrequency)) && strcmpi(cfg.Modulation,'PCM/PSK/PM')
    error('Sample rate must be greater than twice the sum of frequency offset and subcarrier frequency');
elseif fs <= (2*freqOffset)
    error('Sample rate must be greater than twice the frequency offset');
end
pfo = comm.PhaseFrequencyOffset('FrequencyOffset',freqOffset, ...
    'PhaseOffset',phaseOffset,'SampleRate',fs);
txSigOffset = pfo(txSig);

% Timing offset as an integer number of samples
timingErr = 5; % Timing error must be <= 0.4*sps
delayedSig = [zeros(timingErr,1);txSigOffset];

% Pass the signal through an AWGN channel
rxSig = awgn(complex(delayedSig),SNRdB,'measured',iBurst);

% Coarse carrier frequency synchronization
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    % Coarse carrier frequency synchronization for PCM/PSK/PM
    coarseSync = HelperCCSDSTCCoarseFrequencyCompensator('FrequencyResolution',100,...
        'SampleRate',fs);
else
    % Coarse carrier frequency synchronization for BPSK and PCM/PSK/biphase-L
    coarseSync = comm.CoarseFrequencyCompensator( ...

```

```

        'Modulation','BPSK','FrequencyResolution',100, ...
        'SampleRate',fs);
end

% Compensation for coarse frequency offset
[rxCoarse,estCoarseFreqOffset] = coarseSync(rxSig);

% Receive filtering
if strcmpi(cfg.Modulation,'BPSK')
    % SRRRC receive filtering for BPSK
    rxFiltDelayed = rxfilter(rxCoarse);
    rxFilt = rxFiltDelayed(rxfilter.FilterSpanInSymbols*sps+1:end);
else
    % Low-pass filtering for PCM/PSK/PM and PCM/PSK/biphase-L
    % Filtering is done with a lowpass filter to reduce the effect of
    % noise to the carrier phase tracking loop
    b = fir1(40,0.3); % Coefficients for 40th-order lowpass filter with cutoff frequency
    rxFiltDelayed = filter(b,1,[rxCoarse;zeros(0.5*(length(b)-1),1)]);
    % Removal of filter delay
    rxFilt = rxFiltDelayed(0.5*(length(b)-1)+1:end);
end

% Fine frequency and phase correction
if strcmpi(cfg.Modulation,'BPSK')
    fineSync = comm.CarrierSynchronizer('SamplesPerSymbol',sps, ...
        'Modulation','BPSK','NormalizedLoopBandwidth',normLoopBWCarrier);
else
    fineSync = HelperCCSDSTCCarrierSynchronizer('SamplesPerSymbol', ...
        cfg.SamplesPerSymbol,'NormalizedLoopBandwidth',normLoopBWCarrier);
end
[rxFine,phErr] = fineSync(rxFilt);

% Subcarrier frequency and phase correction
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    subSync = HelperCCSDSTCSubCarrierSynchronizer('SamplesPerSymbol',sps, ...
        'NormalizedLoopBandwidth',normLoopBWSubcarrier);
    [rxSub,subCarPhErr] = subSync(real(rxFine));
else
    rxSub = real(rxFine);
end

% Timing synchronization and symbol demodulation
timeSync = HelperCCSDSTCSymbolSynchronizer('SamplesPerSymbol',sps, ...
    'NormalizedLoopBandwidth',normLoopBWSymbol);
[rxSym,timingErr] = timeSync(rxSub);

% Search for start sequence and bit recovery
bits = HelperCCSDSTCCLTUBitRecover(rxSym,cfg,'Error Correcting',0.8);
bits = bits(~cellfun('isempty',bits)); % Removal of empty cell array contents

% Length of transfer frames with fill bits
if strcmpi(cfg.ChannelCoding,'BCH')
    messageLength = 56;
else
    messageLength = 0.5*cfg.LDPCCodewordLength;
end
frameLength = messageLength*ceil(length(inBits)/messageLength);

```

```
if (isempty(bits)) || (length(bits{1})~= frameLength) ||(length(bits)>1)
    cltuLost = cltuLost + 1;
else
    numErr = sum(abs(double(bits{1}(1:length(inBits)))-inBits));
    totNumErrs = totNumErrs + numErr;
    totNumBits = totNumBits + length(inBits);
end
end
bitsErr(iSNR) = totNumErrs/totNumBits;
cltuErr(iSNR) = cltuLost;

% Display of bit error rate and number of CLTUs lost
fprintf(['\nBER with ', num2str(SNRdB+10*log10(sps)) ],' dB Es/No : %1.2e\n'],bitsErr(iSNR))
fprintf(['\nNumber of CLTUs lost with ', num2str(SNRdB+10*log10(sps)) ],' dB Es/No : %d\n']
end

BER with 8 dB Es/No : 0.00e+00

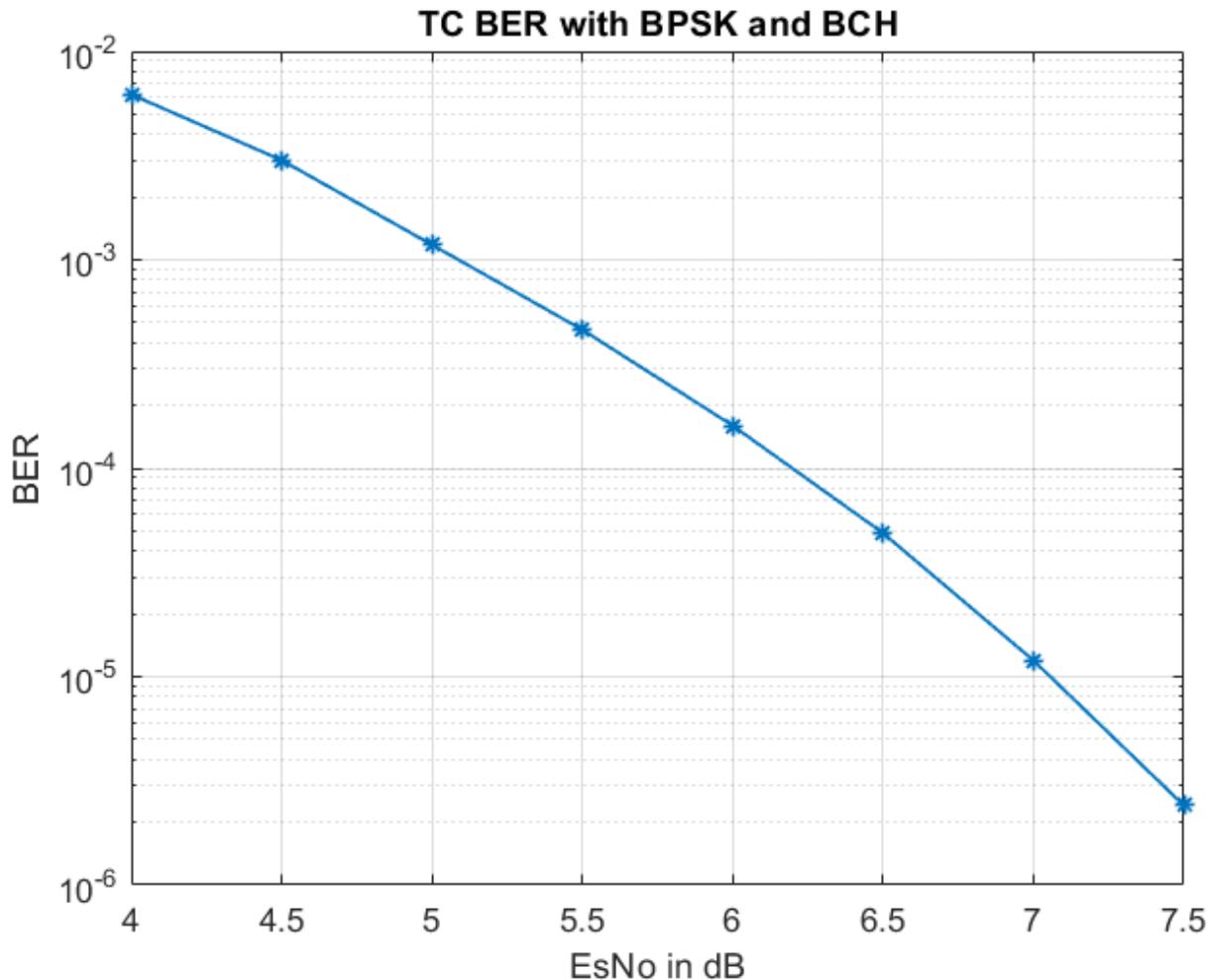
Number of CLTUs lost with 8 dB Es/No : 0

BER with 8.5 dB Es/No : 0.00e+00

Number of CLTUs lost with 8.5 dB Es/No : 0
```

BER Results

When each Es/No point is completed, the BER results for the simulation are plotted. `bitsErr` is an array with the measured BER for all simulated Es/No points. The figure shows the simulation result that are obtained with 10,000 number of transmissions and Es/No points in the range [4 7.5].



Further Exploration

Normalized Loop Bandwidth and Acquisition Sequence Length

This example uses a large value for the acquisition sequence length (800 octets) to improve the performance of synchronizers at low SNR values. This table shows the normalized loop bandwidth values and the samples per symbol used in the simulation with each modulation scheme, for an acquisition sequence of 800 octets.

```
T = table({'BPSK';'PCM/PSK/PM';'PCM/PM/biphase-L'},[0.005; 0.0002; 0.0003], ...
        {'Not Applicable';0.00005;'Not Applicable'},[0.005; 0.0005; 0.0005], ...
        [20; 200; 20],[2048000; 4000; 256000],'VariableNames',{'Modulation','Carrier Synchronizer',
        'Subcarrier Synchronizer','Symbol Synchronizer','Samples per symbol','Symbol rate'})
```

T=3x6 table

Modulation	Carrier Synchronizer	Subcarrier Synchronizer	Symbol Synchronizer
{'BPSK' }	0.005	{'Not Applicable'}	0.005
{'PCM/PSK/PM' }	0.0002	{[5.0000e-05]}	0.0005

'PCM/PM/biphase-L'}

0.0003

'Not Applicable'}

0.0005

You can use this example to further explore these synchronization modules.

- Carrier synchronization: To improve the accuracy of the phase estimate, you can reduce the noise contribution to the tracking loops by decreasing the normalized bandwidth. Reducing the loop bandwidth reduces the pull-in range, and the acquisition takes a longer time to converge.
- Subcarrier synchronization: You can plot the estimated subcarrier offset to identify a more accurate loop bandwidth. To help improve the accuracy of the subcarrier frequency estimate, you can increase the sample rate and SNR.
- Symbol synchronization: The DTTL for the symbol synchronization performs well at higher number of samples per symbol. As you increase the samples per symbol, the resolution increases, and the DTTL performance improves. Too many samples per symbol can reduce the SNR and affect the performance. If the SNR is less than -15 dB (due to a large number of samples per symbol), the performance of the tracking loops is affected.

For any PLL-based loop, to operate at very low SNR, the loop bandwidth must be very low. This low loop bandwidth reduces the pull-in range. For the CLTUs with LDPC channel coding, if the number of CLTUs lost is high, you can reduce the threshold value for the detection of the start sequence in the helper function `HelperCCSDSTCCLTUBitRecover`. You can also try improving the BER results by selecting only the CLTUs with a very high normalized correlation metric with the start sequence. To maximize the frame detection and minimize the false alarm, 0.8 is used as a detection threshold in this example. To reduce the false alarm, you can increase the detection threshold value. If you increase the detection threshold value, the frame detection rate reduces.

Subcarrier Frequency Offset with PCM/PSK/PM

The maximum subcarrier frequency offset specified in the CCSDS TC recommendation [6] on page 4-12 is $\pm(2 \times 10^{-4})f_{sc}$, where f_{sc} is the frequency of telecommand subcarrier. You must consider a frequency offset maximum of 3.2 Hz or 1.6 Hz with a 16 KHz or 8 KHz sine wave subcarrier, respectively. You can plot the estimated subcarrier frequency offset to analyze the performance of the subcarrier tracking. When the synchronizer converges, the mean value of the estimate is approximately equal to the input subcarrier frequency offset value of 3.2 Hz.

```
if strcmpi(cfg.Modulation, 'PCM/PSK/PM')
    estSubCarFreqOffset = diff(subCarPhErr)*fs/(2*pi);
    rmean = cumsum(estSubCarFreqOffset)./(1:length(estSubCarFreqOffset));
    plot(rmean)
    xlabel('Symbols')
    ylabel('Estimated Subcarrier Frequency Offset (Hz)')
    title('PCM/PSK/PM: Subcarrier Frequency Offset')
    grid on
end
```

Appendix

The example uses these helper functions:

- `HelperCCSDSTCCoarseFrequencyCompensator`: Perform coarse carrier frequency synchronization
- `HelperCCSDSTCCarrierSynchronizer`: Perform fine carrier synchronization
- `HelperCCSDSTCSubCarrierSynchronizer`: Perform subcarrier synchronization

- HelperCCSDSTCSymbolSynchronizer: Perform symbol timing synchronization and demodulation
- HelperCCSDSTCSubCarrierModulation: Perform subcarrier modulation with frequency and phase offset
- HelperCCSDSTCCLTUBitRecover: Search for start sequence and bit recovery

Bibliography

- 1 J. Vilà-Valls, M. Navarro, P. Closas and M. Bertinelli, "Synchronization challenges in deep space communications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 1, pp. 16-27, Jan. 2019.
- 2 M. Baldi et al., "State-of-the-art space mission telecommand receivers," *IEEE Aerospace and Electronic Systems Magazine*, vol. 32, no. 6, pp. 4-15, June 2017.
- 3 S. Million and S. Hinedi, "Effects of symbol transition density on the performance of the data transition tracking loop at low signal-to-noise ratios," *Proceedings IEEE International Conference on Communications ICC '95*, Seattle, WA, USA, 1995, pp. 1036-1040 vol.2.
- 4 TC Synchronization and Channel Coding. *Recommendation for Space Data System Standards*, CCSDS 231.0-B-3. Blue Book. Issue 3. Washington, D.C.:CCSDS, September 2017.
- 5 TC Synchronization and Channel Coding. *Summary of Concept and Rationale*. CCSDS 230.1-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, November 2012.
- 6 Radio Frequency and Modulation Systems - Part 1. *Earth Stations and Spacecraft*. CCSDS 401.0-B-29. Blue Book. Issue 29. Washington, D.C.: CCSDS, March 2019.
- 7 Michael Rice, *Digital Communications - A Discrete-Time Approach*. New York: Prentice Hall, 2008.
- 8 Nguyen, T.M., W.L. Martin, and Hen-Geul Yeh. "Required Bandwidth, Unwanted Emission, and Data Power Efficiency for Residual and Suppressed Carrier Systems-a Comparative Study." *IEEE transactions on electromagnetic compatibility* 37, no. 1 (February 1995): 34-50. <https://doi.org/10.1109/15.350238>.

See Also

Objects

ccsdsTCConfig | ccsdsTMWaveformGenerator

Functions

ccsdsTCWaveform

Related Examples

- "End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections" on page 4-13
- "End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections" on page 4-23

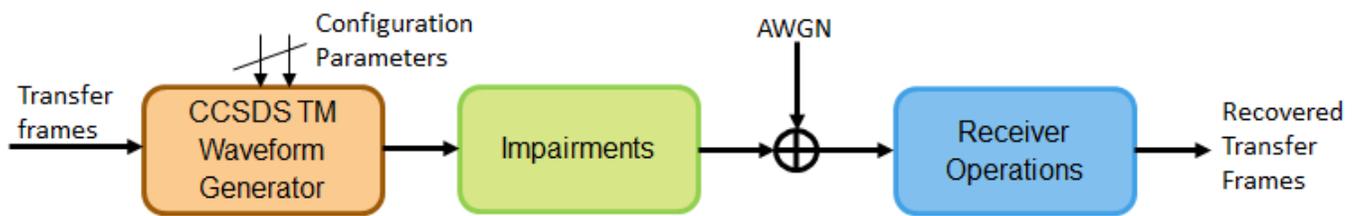
End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) of the end-to-end chain of the Consultative Committee for Space Data Systems (CCSDS) telemetry (TM) system. The simulation chain follows the coding and modulation schemes that are specified by these two standards:

- TM Synchronization and Channel Coding - CCSDS 131.0-B-3 for channel coding schemes [1] on page 4-21
- Radio Frequency and Modulation Systems - part 1 Earth Stations and Spacecraft - CCSDS 401.0-B-30 for modulation schemes [2] on page 4-21

Introduction

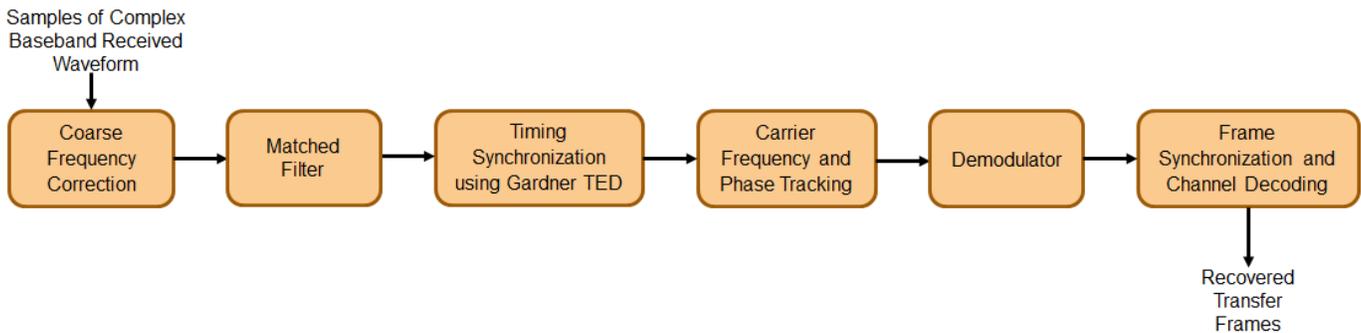
Data from various instruments is generated on board the satellite. This data collectively is called *TM data*. CCSDS specifies the coding and modulation schemes for the transmission of TM data from the satellite to an Earth station. This example shows the end-to-end simulation of the satellite to the Earth station communication link. The example shows how to generate a complex baseband CCSDS TM waveform from the randomly generated transfer frames (TFs), introduce radio frequency (RF) impairments to the baseband signal, and add additive white Gaussian noise (AWGN) to the impaired signal. Then, the example shows the synchronization, demodulation, and decoding of this impaired noisy signal to get the final bits in the form of TFs. The example also shows how to measure the BER with respect to the signal-to-noise ratio (SNR) for one configuration of the CCSDS TM signal. This figure shows the end-to-end simulation chain.



This example models these RF impairments:

- Carrier frequency offset (CFO)
- Carrier phase offset (CPO)
- Symbol timing offset (STO)

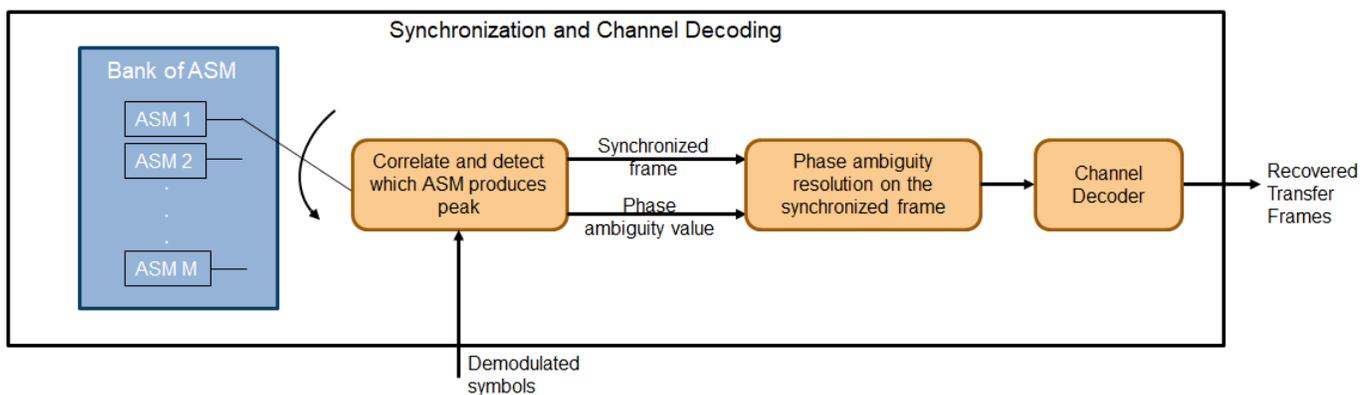
This figure shows the receiver side operations.



The frame synchronization and channel decoding processing step performs these three tasks.

- 1 Perform phase ambiguity resolution
- 2 Correctly synchronize the frame to the starting of the attached sync marker (ASM)
- 3 Perform channel decoding of the synchronized frame to get the recovered TF

This figure shows these three tasks. To start, form a bank of ASM sequences. Each sequence corresponds to the original ASM value in which phase ambiguity is introduced. Correlate each of these sequences with the demodulated symbols. Choose the phase ambiguity value that has the highest correlation peak. Perform the frame synchronization with this correlation process. The process to perform correlation is illustrated in section 9.3.7 in [3] on page 4-21. This example adopts the simplified Massey algorithm for frame synchronization. Resolve the phase ambiguity on the complete set of demodulated symbols after the frame synchronization process is complete. Finally, perform channel decoding on these symbols to obtain the recovered TFs.



Simulation Parameters

This example uses a quadrature phase shift keying (QPSK) modulation scheme for signal generation and reception and a rate-1/2 convolutional coding scheme for channel coding. The end-to-end chain this example shows can also be used for the channel coding schemes that are specified in [1] on page 4-21: Reed-Solomon (RS) codes and concatenated codes. For convolutional and concatenated codes, this example supports rates of 1/2 and 2/3 along with pulse code modulation (PCM)-format of non-return to zero-line (NRZ-L). The supported modulation schemes in this example are binary phase shift keying (BPSK) and QPSK.

```

seeConstellation = true;           % Flag to toggle the visualization of constellation
channelCoding = "convolutional"; % Channel coding scheme
  
```

```

transferFrameLength = 1115;      % In bytes corresponding to 223*5
modScheme = "QPSK";             % Modulation scheme
alpha = 0.35;                   % Root raised cosine filter roll-off factor
sps = 8;                         % Samples per symbol

```

Set the frequencies that are used for signal generation and the RF impairment values.

```

fSym = 2e6; % Symbol rate or Baud rate in Hz
cfo = 2e5; % In Hz

```

Initialize the energy per bit to noise power ratio (Eb/N0), which is used to calculate the SNR using system parameters.

```

EbN0 = 10; % To see a proper BER result, run the simulation for 3.2:0.2:5

```

Initialize the parameters to terminate the simulation. The parameters are set to small values in this example to get quick results. Increase these parameters value to get a smoother BER curve.

```

maxNumErrors = 1e2; % Simulation stops after maxNumErrors bit errors
maxNumBits = 1e5;   % Simulation stops after processing maxNumBits
                  % Set maxNumBits = 1e8 for a smoother BER curve
maxFramesLost = 1e2; % Simulation stops after maxFramesLost frames are lost

```

System Parameters

Initialize all of the objects that are required for the proper functioning of the end-to-end chain.

Create a CCSDS TM waveform generator with these parameters by using the `ccsdsTMWaveformGenerator` System object™. Display the properties of the object.

```

tmWaveGen = ccsdsTMWaveformGenerator("ChannelCoding",channelCoding, ...
    "NumBytesInTransferFrame",transferFrameLength, ...
    "Modulation",modScheme, ...
    "RolloffFactor",alpha, ...
    "SamplesPerSymbol",sps);
disp(tmWaveGen)

```

```

ccsdsTMWaveformGenerator with properties:

```

```

    WaveformSource: "synchronization and channel coding"
    NumBytesInTransferFrame: 1115
        HasRandomizer: true
            HasASM: true
    PCMFormat: "NRZ-L"

```

```

Channel coding
    ChannelCoding: "convolutional"
    ConvolutionalCodeRate: "1/2"

```

```

Digital modulation and filter
    Modulation: "QPSK"
    PulseShapingFilter: "root raised cosine"
        RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
        SamplesPerSymbol: 8

```

```

Use get to show all properties

```

Calculate the SNR from the Eb/N0 and initialize the parameters related to the calculation of BER.

```

rate = tmWaveGen.info.ActualCodeRate;
M = tmWaveGen.info.NumBitsPerSymbol;
numBitsInTF = tmWaveGen.NumInputBits;
snr = EbN0 + 10*log10(rate) + ...
      10*log10(M) - 10*log10(sps);           % As signal power is scaled to one while introducing noise
                                             % SNR value should be reduced by a factor of SPS

numSNR = length(snr);
ber = zeros(numSNR,1);                       % Initialize the BER parameter
bercalc = comm.ErrorRate;

```

Create a receive filter object by using the `comm.RaisedCosineReceiveFilter` System object.

```

b = rcosdesign(alpha,tmWaveGen.FilterSpanInSymbols,sps);
% |H(f)| = 1 for |f| < fN(1-alpha) - Annex 1 in Section 2.4.17A in [2]
Gain = sum(b);
rxFilterDecimationFactor = sps/2;
rxfilter = comm.RaisedCosineReceiveFilter( ...
    "DecimationFactor",rxFilterDecimationFactor, ...
    "InputSamplesPerSymbol",sps, ...
    "RolloffFactor",alpha, ...
    "Gain",Gain);

```

Model frequency and phase offsets by using the `comm.PhaseFrequencyOffset` System object. Compensate for frequency and phase offset at the receiver in two steps.

- 1 Compensate for the coarse frequency offset by using the `comm.CoarseFrequencyCompensator` System object.
- 2 Compensate for the fine frequency offset and the phase offset by using the `comm.CarrierSynchronizer` System object.

```

phaseOffset = pi/8;
fpyoffsetobj = comm.PhaseFrequencyOffset( ...
    "FrequencyOffset",cfo, ...
    "PhaseOffset",phaseOffset, ...
    "SampleRate",sps*fSym);
coarseFreqSync = comm.CoarseFrequencyCompensator( ...
    "Modulation",modScheme, ...
    "FrequencyResolution",100, ...
    "SampleRate",sps*fSym);
fineFreqSync = comm.CarrierSynchronizer("DampingFactor",1/sqrt(2), ...
    "NormalizedLoopBandwidth",0.0007, ...
    "SamplesPerSymbol",1, ...
    "Modulation",modScheme);

```

Create a variable fractional delay object by using the `dsp.VariableFractionalDelay` System object, which introduces the fractional delay in the transmitted waveform. Create a symbol synchronization object by using the `comm.SymbolSynchronizer` System object, which performs symbol timing synchronization.

```

varDelay = dsp.VariableFractionalDelay("InterpolationMethod","Farrow");
fixedDelayVal = 10.2;
Kp = 1/(pi*(1-((alpha^2)/4)))*cos(pi*alpha/2);
symsyncobj = comm.SymbolSynchronizer( ...
    "DampingFactor",1/sqrt(2), ...
    "DetectorGain",Kp, ...
    "TimingErrorDetector","Gardner (non-data-aided)", ...
    "Modulation","PAM/PSK/QAM", ...

```

```
"NormalizedLoopBandwidth",0.0001, ...
"SamplesPerSymbol",sps/rxFILTERDecimationFactor);
```

Demodulate and decode the received signal by using the `HelperCCSDSTMDemodulator` and `HelperCCSDSTMDecoder` helper files, respectively. Display the properties of the resulting objects.

```
demodobj = HelperCCSDSTMDemodulator("Modulation",modScheme,"ChannelCoding",channelCoding)
```

```
demodobj =
  HelperCCSDSTMDemodulator with properties:
```

```
    Modulation: "QPSK"
    PCMFormat: "NRZ-L"
    ChannelCoding: "convolutional"
```

```
decoderobj = HelperCCSDSTMDecoder("ChannelCoding",channelCoding, ...
    "NumBytesInTransferFrame",transferFrameLength, ...
    "Modulation",modScheme)
```

```
decoderobj =
  HelperCCSDSTMDecoder with properties:
```

```
    ChannelCoding: "convolutional"
    HasRandomizer: true
    HasASM: true
    DisableFrameSynchronization: 0
    DisablePhaseAmbiguityResolution: 0
    NumBytesInTransferFrame: 1115
    ConvolutionalCodeRate: "1/2"
    ViterbiTraceBackDepth: 60
    ViterbiTrellis: [1x1 struct]
    ViterbiWordLength: 8
    Modulation: "QPSK"
    PCMFormat: "NRZ-L"
```

Initialize the constellation diagram object by using the `comm.ConstellationDiagram` System object to visualize how the constellation evolves as the synchronizers converge.

```
constellationobj = comm.ConstellationDiagram; % Default view is for QPSK
if strcmp(modScheme,'BPSK')
    constellationobj.ReferenceConstellation = [1, -1]
end
```

Processing Chain

To simulate the end-to-end chain and measure the BER of the CCSDS TM system, follow these steps.

- 1 Generate random bits to form a TF.
- 2 Generate the TM waveform by passing the TF through the `ccsdSTMWaveformGenerator` System object.
- 3 Introduce RF impairments, such as CFO and symbol delay.
- 4 Add AWGN to the RF-impaired signal. This noisy signal is considered the received signal.
- 5 Pass the received signal through coarse frequency correction, which performs the initial coarse carrier frequency synchronization. Coarse frequency estimation is done using the "FFT-based" algorithm.

- 6 Use a matched filter (root raised cosine filter) with the same configuration that is applied at the transmitter end. Because the symbol timing synchronization module works at a sampling rate that is higher than the symbol rate, the complex baseband samples are not down sampled to the symbol rate after filtering. It is down sampled such that at least 2 samples per symbol exist.
- 7 Perform symbol timing synchronization by using the Gardner timing error detector (TED) to remove the timing offset that is present in the signal.
- 8 Perform carrier frequency and phase tracking by using the `comm.CarrierSynchronizer` System object, which has a type 2 phase locked loop (PLL). This System object can track a stationary carrier frequency offset. The System object also introduces phase ambiguity, which is then removed by the frame synchronization module.
- 9 Visualize the constellation after symbol timing and carrier frequency synchronization is complete. Observe how the constellation evolves over multiple iterations.
- 10 Demodulate the received signal and verify that the signal is at the symbol rate (that is, the samples per symbol is 1).
- 11 Perform frame synchronization and channel decoding to resolve the phase ambiguity, synchronize the frame to the start of the ASM, and then decode the synchronized frame to recover the TF.

```

numBitsForBER = 8; % For detecting which frame is synchronized
numMessagesInBlock = 2^numBitsForBER;
for isnr = 1:numSNR
    rng default; % Reset to get repeatable results
    reset(bercalc);
    berininfo = bercalc(int8(1), int8(1)); % Initialize berininfo before BER is calculated
    tfidx = 1;
    numFramesLost = 0;
    prevdectfidx = 0;
    inputBuffer = zeros(numBitsInTF, 256, "int8");
    while((berininfo(2) < maxNumErrors) && ...
        (berininfo(3) < maxNumBits) && ...
        (numFramesLost < maxFramesLost))
        seed = randi([0 2^32-1],1,1); % Generate seed for repeatable simulation

        % Transmitter side processing
        bits = int8(randi([0 1],numBitsInTF-numBitsForBER,1));
        % The first 8 bits correspond to the TF index modulo 256. When
        % synchronization modules are included, there can be a few frames
        % where synchronization is lost temporarily and then locks again.
        % In such cases, to calculate the BER, these 8 bits aid in
        % identifying which TF is decoded. If an error in these 8 bits
        % exists, then this error is detected by looking at the difference
        % between consecutive decoded bits. If an error is detected, then
        % that frame is considered lost. Even though the data link layer is
        % out of scope of this example, the data link layer has a similar
        % mechanism. In this example, only for calculating the BER, this
        % mechanism is adopted. The mechanism that is adopted in this
        % example is not as specified in the data link layer of the CCSDS
        % standard. And this mechanism is not specified in the physical
        % layer of the CCSDS standard.
        msg = [de2bi(mod(tfidx-1,numMessagesInBlock),numBitsForBER,"left-msb").';bits];
        inputBuffer(:,mod(tfidx-1,numMessagesInBlock)+1) = msg;
        tx = tmWaveGen(msg);

        % Introduce RF impairments
        cfoIntroduced = fqyoffsetobj(tx); % Introduce CFO
        delayed = varDelay(cfoIntroduced,fixedDelayVal); % Introduce timing offset

```

```

rx = awgn(delayed, snr(isnr), 'measured', seed); % Add AWGN

% Receiver-side processing
coarseSynced = coarseFreqSync(rx); % Apply coarse frequency synchronization
filtered = rxfilter(coarseSynced); % Filter received samples through RRC filter
TimeSynced = symsyncobj(filtered); % Apply symbol timing synchronization
fineSynced = fineFreqSync(TimeSynced); % Track frequency and phase

% Visualize constellation
if seeConstellation
    % Plot constellation of first 1000 symbols in a TF so
    % that variable size of fineSynced does not impede the
    % requirement of constant input size for the
    % comm.ConstellationDiagram System object.
    constellationobj(fineSynced(1:1000));
end

demodData = demodobj(fineSynced); % Demodulate
decoded = decoderobj(demodData); % Perform phase ambiguity resolution,
    % frame synchronization, and channel decoding

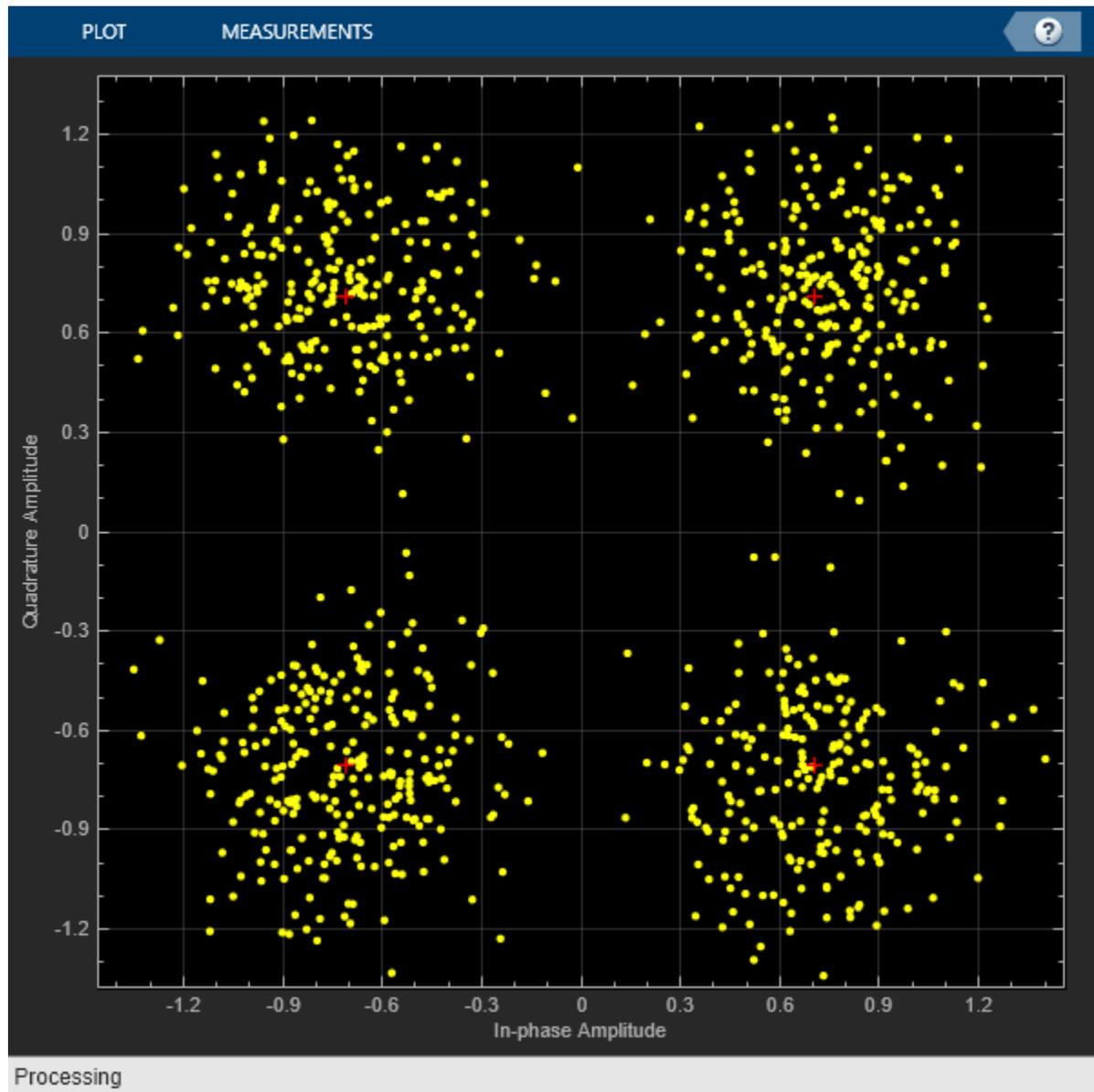
% Calculate BER and adjust all buffers accordingly
dectfidx = bi2de(double(decoded(1:8).'), ...
    "left-msb")+1; % See the value of first 8 bits
if tfidx > 30 % Consider to calculate BER only after 30 TFs are processed
    % As the value of first 8 bits is increased by one in each
    % iteration, if the difference between the current decoded
    % decimal value of first 8 bits is not equal to the previously
    % decoded one, then it indicates a frame loss.
    if dectfidx - prevdectfidx ~= 1
        numFramesLost = numFramesLost + 1;
        disp(['Frame lost at tfidx: ' num2str(tfidx) ...
            '. Total frames lost: ' num2str(numFramesLost)]);
    else
        berinfo = bercalc(inputBuffer(:, dectfidx), decoded);
        if nnz(inputBuffer(:, dectfidx) - decoded)
            disp(['Errors occurred at tfidx: ' num2str(tfidx) ...
                '. Num errors: ' num2str(nnz(inputBuffer(:, dectfidx) - decoded))])
        end
    end
end
prevdectfidx = dectfidx;

% Update tfidx
tfidx = tfidx + 1;
end
fprintf("\n");
currentBer = berinfo(1);
ber(isnr) = currentBer;
disp(['Eb/N0: ' num2str(EbN0(isnr)) '. BER: ' num2str(currentBer) ...
    '. Num frames lost: ' num2str(numFramesLost)]);

% Reset objects
reset(tmWaveGen);
reset(fqyoffsetobj);
reset(varDelay);
reset(coarseFreqSync);
reset(rxfilter);

```

```
reset(symsyncobj);  
reset(fineFreqSync);  
reset(demodobj);  
reset(decoderobj);  
end
```



E_b/N_0 : 10. BER: 0. Num frames lost: 0

Further Exploration

This example demonstrates BER simulation for convolutional codes with QPSK modulation in the presence of several RF impairments. To observe the end-to-end simulation chain for different scenarios, change the properties related to the channel coding and modulation schemes. The modulation schemes that are supported by the receiver in this example are BPSK and QPSK. The

channel coding schemes that are supported by the receiver in this example are none (that is, no channel coding), RS, convolutional, and concatenated codes.

Run a full BER simulation by setting the Eb/N0 value to 3.2:0.2:5 and observe the BER by setting `maxNumBits` to 1e8. Uncomment this code to plot the BER results.

```
% semilogy(EbN0,ber);  
% grid on;  
% xlabel('E_b/N_0 (dB)');  
% ylabel('BER');  
% title('BER plot');
```

Always reserve the initial few TFs for the symbol timing and carrier frequency synchronizers to lock. This example discards the first 30 TFs. This number can vary based on the SNR at which the receiver is operating and the parameters of the synchronization loops, such as loop bandwidth and damping factor. If you operate the receiver at low SNR and observe large errors in the initial values of `tfidx`, then the synchronizers are not yet locked. For the given simulation parameters, discard the initial TFs as appropriate. The second output arguments of `comm.CoarseFrequencyCompensator` and `comm.CarrierSynchronizer` System objects contain the information related to the estimated CFO, which can be used to assess whether the synchronization loops are locked or not.

Appendix

The example uses the following helper files:

- `HelperCCSDSTMDemodulator.m` - Performs the demodulation of signals that are specified in CCSDS TM [2] on page 4-21
- `HelperCCSDSTMDecoder.m` - Performs, phase ambiguity resolution, frame synchronization and channel decoding of the codes specified in [1] on page 4-21

References

[1] TM Synchronization and Channel Coding. Recommendation for Space Data System Standards, CCSDS 131.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, September 2017.

[2] Radio Frequency and Modulation Systems--Part 1: Earth Stations and Spacecraft. Recommendation for Space Data System Standards, CCSDS 401.0-B-30. Blue Book. Issue 30. Washington, D.C.: CCSDS, February 2020.

[3] TM Synchronization and Channel Coding - Summary of Concept and Rationale. Report Concerning Space Data System Standards, CCSDS 130.1-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, June 2020.

See Also

Objects

`ccsdsTMWaveformGenerator` | `ccsdsTCConfig`

Related Examples

- “End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections” on page 4-2
- “End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections” on page 4-23

End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) of the end-to-end chain of the Consultative Committee for Space Data Systems (CCSDS) flexible advanced coding and modulation (FACM) scheme for high rate telemetry (TM) applications system [1] on page 4-36.

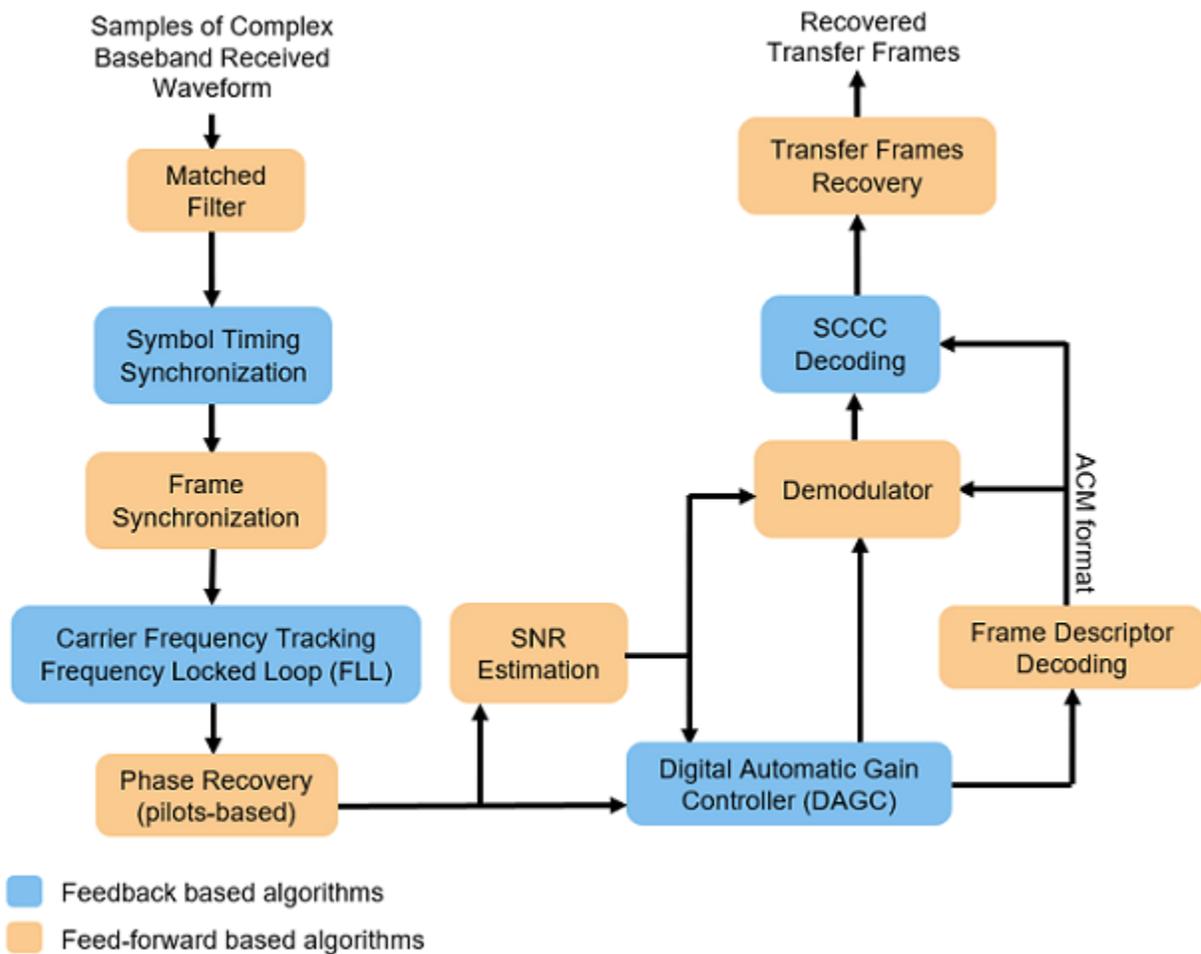
Introduction

Data from various instruments is generated in the satellite. This data collectively is called *TM data*. Earth exploration satellite service (EESS) missions carry payloads that produce substantial TM data rates, starting from a few hundreds of Mbps. To achieve high spectral efficiency for such missions, coding and modulation schemes must be adjusted based on the link budget. The CCSDS FACM scheme for high rate TM applications [1] on page 4-36 standard supports a high data rate by adopting the serial concatenated convolutional codes (SCCC) and modulation schemes from the family of phase shift keying (PSK) and amplitude phase shift keying (APSK). This example shows how to generate a complex baseband CCSDS FACM waveform from the randomly generated transfer frames (TFs), introduce radio frequency (RF) impairments to the baseband signal, and add additive white Gaussian noise (AWGN) to the impaired signal. Then, the example shows the symbol timing recovery, carrier frequency synchronization, demodulation, and decoding of this impaired noisy signal to get the final bits in the form of TFs. This example also shows how to measure the BER with respect to the signal-to-noise ratio (SNR) for one configuration of the CCSDS FACM signal.

This example models these RF impairments on the baseband signal:

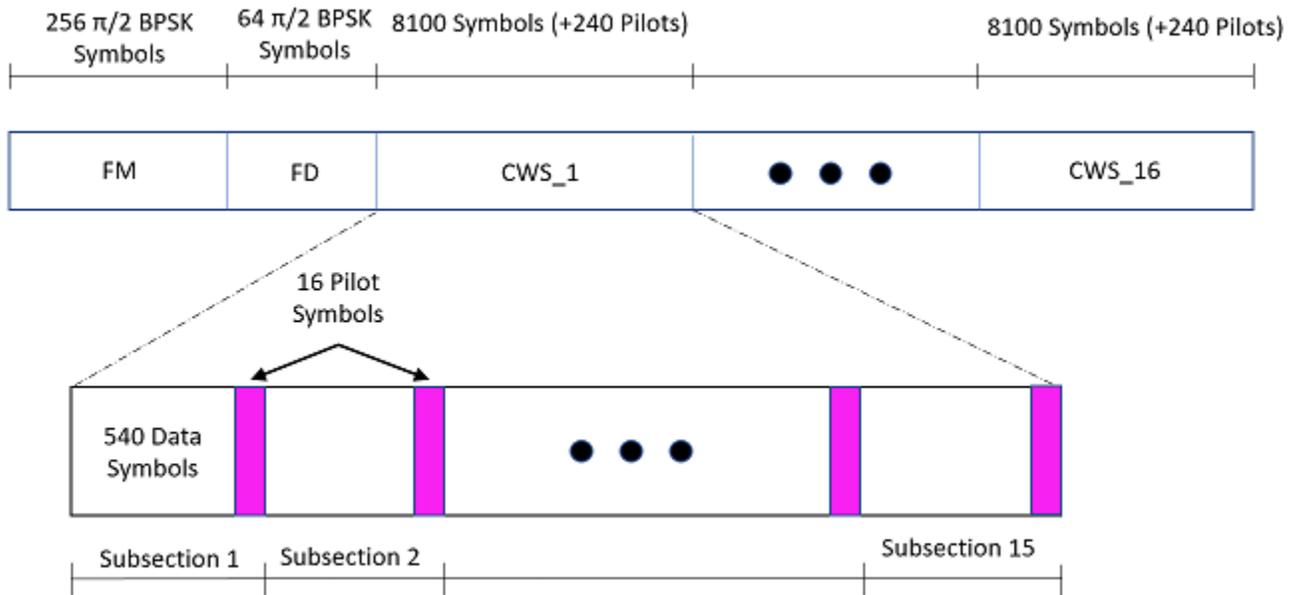
- Carrier frequency offset (CFO)
- Doppler rate
- Sampling rate offset (SRO)
- Phase noise

This figure shows the receiver process, which recovers the symbol timing, carrier frequency, and carrier phase in the presence of the RF impairments.



In the receiver chain, frequency locked loop (FLL) uses the frame marker (FM) that is available in the physical layer (PL) header. The phase recovery module uses the pilot fields to recover the phase. Because the phase recovery algorithm is pilots-based, you must enable the pilot fields for the example to work. The phase recovery module can tolerate some amount of residual CFO that is left after the FLL operation.

This figure shows the PL frame structure of the FACM waveform. In one PL frame, 16 code word sections (CWS) exist and each codeword section is formed from each SCCC encoder output. Use the `ccsdstmWaveformGenerator` System object to generate the FACM waveform.



Configuration and Simulation Parameters

This example shows various visualizations such as constellation plots and the spectrum of the signal. You can optionally disable these visualizations. For this example, enable them.

```
showVisualizations = ;
```

Set the configuration parameters that control the waveform properties.

```
sps = 2; % Samples per symbol (SPS)
rolloff = 0.35; % Filter roll-off factor
cfgCCSDSFACM.NumBytesInTransferFrame = 223;
cfgCCSDSFACM.SamplesPerSymbol = sps;
cfgCCSDSFACM.RolloffFactor = rolloff;
cfgCCSDSFACM.FilterSpanInSymbols = 10;
cfgCCSDSFACM.ScramblingCodeNumber = 1;
cfgCCSDSFACM.ACMFormat = 1;
cfgCCSDSFACM.HasPilots = true; % HasPilots must be set to true
% for this example to work
```

Specify the simulation parameters in `simParams` structure. Specify the SNR in dB as energy per symbol to noise power ratio (E_s/N_0) in the `EsNodB` field.

```
simParams.EsNodB = 1;
```

Specify the CFO and SRO. Model CFO using the `comm.PhaseFrequencyOffset` System object. Model the sampling rate offset is using the `comm.SampleRateOffset` System object.

```
simParams.CFO = 1e6; % In Hz
simParams.SRO = 20; % In PPM
```

Specify the attenuation factor for the signal. In the case of no attenuation, specify this value as 1.

```
simParams.AttenuationFactor = 0.1; % Must be less than or equal to 1
```

Specify the number of PL frames to be generated by specifying the number of frames for initial synchronization that are not accounted in the BER calculation and the number of frames that are accounted in the BER calculations. In this example, set `simParams.NumFramesForBER` to 10 to complete the simulation early. To see a proper BER value, set this value to 200.

```
% Initialize the number of frames used for synchronization
simParams.InitalSyncFrames = 15;
% Initialize the number of frames that are used for calculation of BER
simParams.NumFramesForBER = 10;
simParams.NumPLFrames = simParams.InitalSyncFrames + simParams.NumFramesForBER;
```

Specify the symbol rate and samples per symbol (SPS).

```
simParams.SymbolRate = 100e6; % 100 MBaud
simParams.SPS = sps;
```

Calculate Latency and Doppler in a Satellite Scenario example shows how the Doppler shift changes with time based on the satellite orbit. The Doppler shift follows a sinusoidal curve with the peak Doppler shift occurring when the satellite starts to become visible to a receiver on the ground, or when the satellite is receding. Such a cyclic change in Doppler shift is modeled as a sinusoidal Doppler profile. The current example simulates Doppler frequency that changes as given in this equation from [2] on page 4-36.

$$f(t) = f_D \cos\left(\frac{f_R}{f_D}t\right)$$

f_D is the peak Doppler shift, and f_R is the Doppler rate. Specify these properties of the Doppler profile.

```
simParams.PeakDoppler = 1e6; % In Hz
simParams.DopplerRate = 50e3; % In Hz/Sec
```

If needed, disable RF impairments for debugging.

```
simParams.DisableRFImpairments = false; % Disable RF impairments
```

Generation of CCSDS FACM Waveform Distorted with RF Impairments

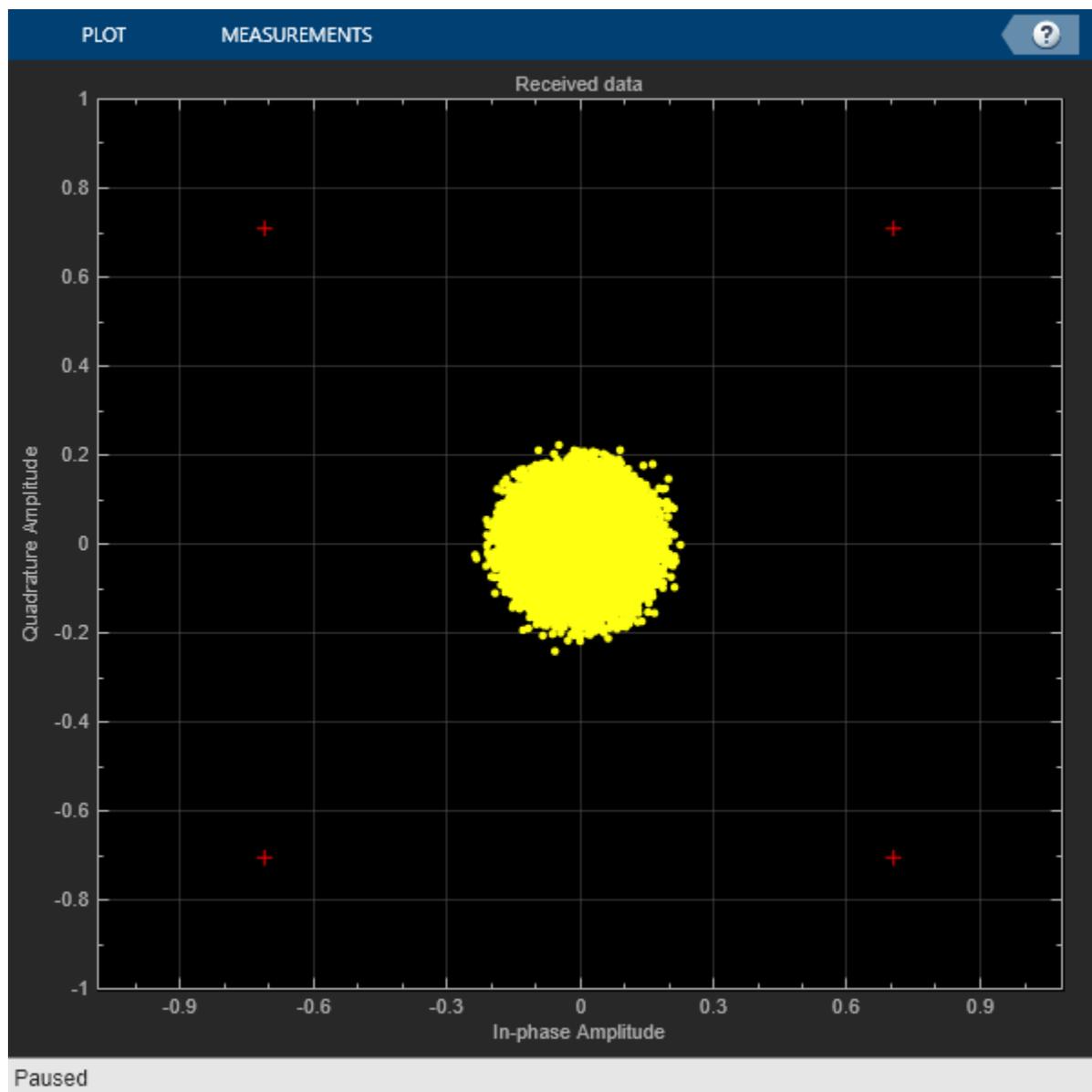
To create a CCSDS FACM waveform, use the `HelperCCSDSFACMRxInputGenerate` helper function, with the `simParams` and `cfgCCSDSFACM` structures as inputs. This function uses the `ccsdstmWaveformGenerator` System object to generate the transmitted waveform. This function returns the data signal, transmitted and received waveforms, and receiver processing structure. The received waveform is impaired with carrier frequency, Doppler shift, timing phase offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields and pilot indices. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

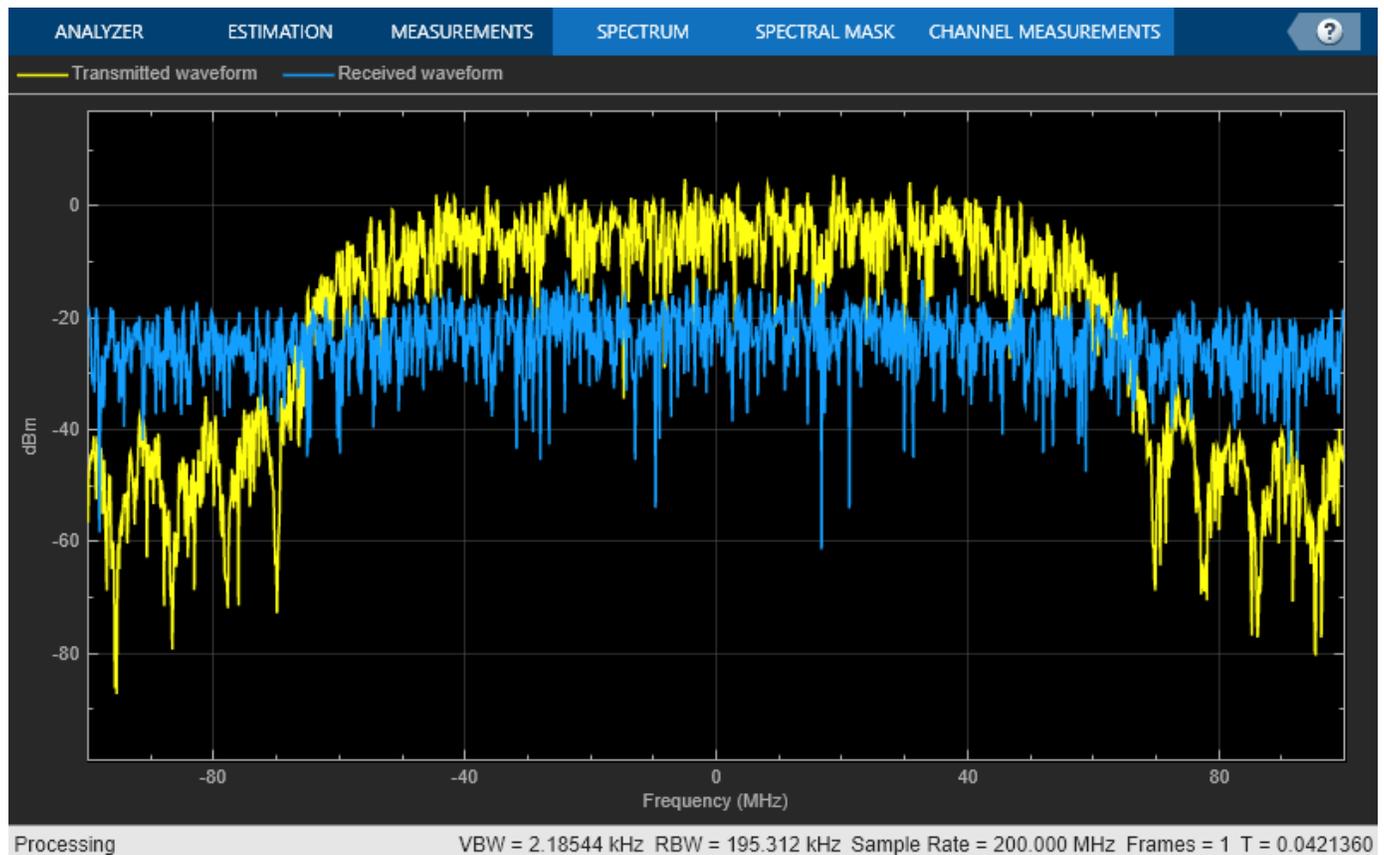
```
[bits,txOut,rxIn,phyParams,rxParams] = ...
    HelperCCSDSFACMRxInputGenerate(cfgCCSDSFACM,simParams);

if showVisualizations == true

    % Plot the received signal constellation
    rxConst = comm.ConstellationDiagram('Title','Received data', ...
        'XLimits',[-1 1],'YLimits',[-1 1], ...
        'ShowReferenceConstellation',true, ...
```

```
    'SamplesPerSymbol',simParams.SPS);  
  
% Calculate the reference constellation for the specified ACM format.  
refConstellation = HelperCCSDSFACMReferenceConstellation(cfgCCSDSFACM.ACMFormat);  
rxConst.ReferenceConstellation = refConstellation;  
rxConst(rxIn(1:rxParams.plFrameSize*sps))  
  
% Plot the transmitted and received signal spectrum  
Fsamp = simParams.SymbolRate*simParams.SPS;  
scope = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...  
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...  
    'ShowLegend',true);  
scope([txOut,rxIn(1:length(txOut))]);  
end
```





Configuration of the Receiver

Create a square root raised cosine (SRRC) receive filter:

```
rxFilterDecimationFactor = sps/2;
rrcfilt = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor',double(rolloff), ...
    'FilterSpanInSymbols',double(cfgCCSDSFACM.FilterSpanInSymbols), ...
    'InputSamplesPerSymbol',sps, ...
    'DecimationFactor',rxFilterDecimationFactor);
b = coeffs(rrcfilt);

% |H(f)| = 1 for |f| < fN(1-alpha), as per Section 6 in the standard [1]
rrcfilt.Gain = sum(b.Numerator);
```

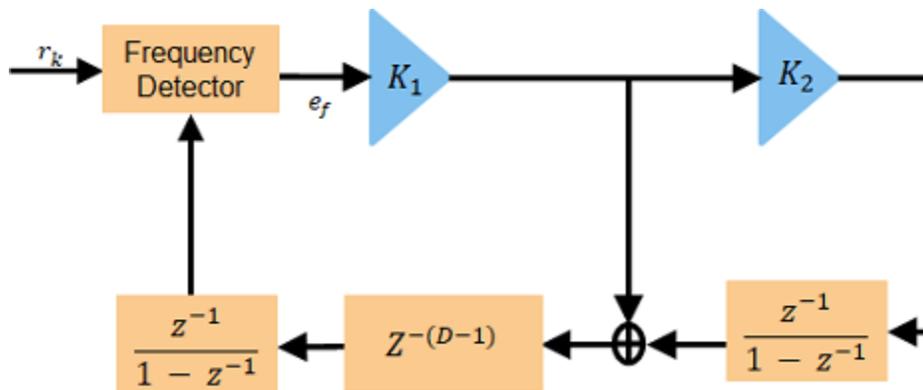
Create a symbol timing synchronization System object, `comm.SymbolSynchronizer`.

```
% Initialize the detector gain. See Eq 8.47 in Digital communications by
% Michael Rice [3].
Kp = 1/(pi*(1-((rolloff^2)/4))*sin(pi*rolloff/2));
symsyncobj = comm.SymbolSynchronizer( ...
    'DampingFactor',1/sqrt(2), ...
    'DetectorGain',Kp, ...
    'TimingErrorDetector','Gardner (non-data-aided)', ...
    'Modulation','PAM/PSK/QAM', ...
    'NormalizedLoopBandwidth',0.0001, ...
    'SamplesPerSymbol',sps/rxFilterDecimationFactor);
```

Visualize the constellation plot after timing and frequency recovery by creating a `comm.ConstellationDiagram` System object.

```
if showVisualizations == true
    constelDiag = comm.ConstellationDiagram();
    constelDiag.ReferenceConstellation = refConstellation;
end
```

Initialize the FLL for carrier frequency synchronization. This figure shows the FLL implementation, as described in section 5.7 in [2] on page 4-36. In this case, the frequency detector is implemented with a fast Fourier transform (FFT) based approach, where the peak in frequency domain indicates the residual carrier frequency. Because this approach is limited by the resolution of the FFT, interpolate around the peak in the frequency domain to detect the residual frequency. Because this approach uses the FM that is available in the PL header, complete the frame synchronization before passing the signal through the FLL. This figure shows the type-2 FLL that handles large Doppler rates.



Initialize the FLL. When you set `K2` to 0, this FLL becomes a type-1 FLL.

```
fll = HelperCCSDSFACMFL('SampleRate',simParams.SymbolRate, ...
    'K1',0.17,'K2',0);
```

Initialize local variables for the receiver chain to work.

```
plFrameSize = rxParams.plFrameSize;
stIdx = 0; % PL frame starting index

% Use one PL frame for each iteration.
endIdx = stIdx + plFrameSize*sps;
rxParams.ffBuffer = complex(zeros(plFrameSize,1));
```

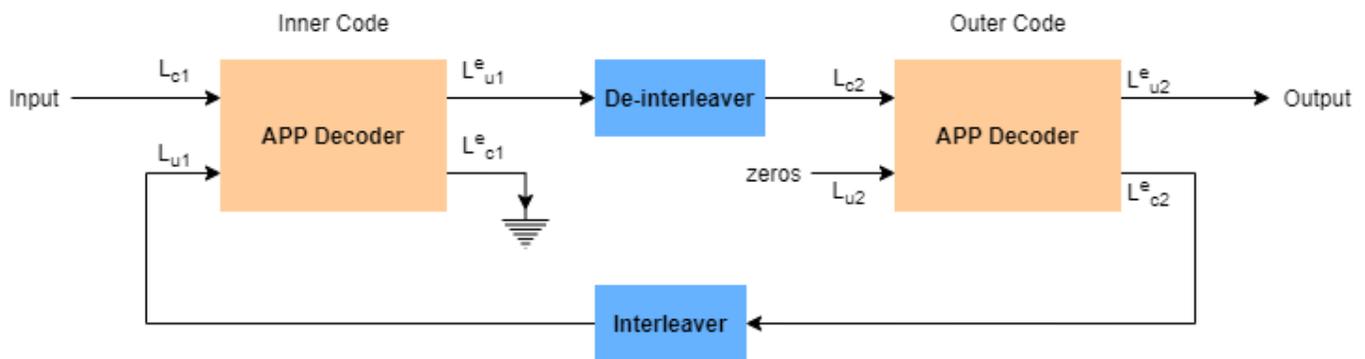
Synchronization and Data Recovery

To recover the data from the received signal, follow these steps.

- 1 Pass the received baseband samples through the SRRRC receive filter.
- 2 Perform symbol timing synchronization. use the `comm.SymbolSynchronizer` System object for symbol timing synchronization. The SRO is compensated while performing symbol timing synchronization.
- 3 Apply frame synchronization to detect the frame boundaries.
- 4 Pass the frame synchronized symbols through the FLL. In the FLL, along with the CFO, the Doppler rate and Doppler shift are also tracked.

- 5 Estimate and compensate the remaining frequency offset in the FM using the reference FM.
- 6 Recover the residual phase from the phase recovery module. The phase recovery module can tolerate some residual CFO.
- 7 Estimate the SNR in the signal by using the phase compensated FM.
- 8 Pass the phase compensated signal through the digital automatic gain controller (DAGC).
- 9 Demodulate the synchronized symbols to get soft bits.
- 10 Perform SCCC decoding on the soft bits to obtain the decoded bits. SCCC decoding can be done using a-posteriori probability (APP) decoder System object, `comm.APPDecoder`.
- 11 Recover the transfer frames from the decoded bits.

This figure shows the SCCC decoder block diagram.



```

% Initialize the number of symbols in a code block. Assuming pilots are
% present in the signal, 15*16 pilots and 8100 data symbols exist in one
% code block.
n = 8100 + 15*16;

% Perform frame synchronization for the first frame outside the loop.

% In the last frame, consider all of the remaining samples in the received
% waveform.
isLastFrame = endIdx > length(rxIn);
endIdx(isLastFrame) = length(rxIn);
rxData = rxIn(stIdx+1:endIdx);
stIdx = endIdx; % Update start index after extracting required data

filteredRx = rrcfilt(rxData); % RRC filter
syncsym = symsyncobj(filteredRx); % Symbol timing synchronization

% Frame synchronization
syncidx = HelperCCSDSFACMFrameSync(syncsym, rxParams.RefFM);
fineCFOsync = comm.PhaseFrequencyOffset('SampleRate', simParams.SymbolRate);

leftOutSym = syncsym(syncidx(1):end);
extraBits = [];
numIterations = 10;
frameIndex = 1;
berinfo = struct('NumBitsInError',0, ...
    'TotalNumBits',0, ...
    'BitErrorRate',0);
snrAveragingFactor = 6; % Average over 6 frames to get accurate estimate of SNR

```

```

SNREst = zeros(snrAveragingFactor,1);
idxTemp = 0;
G = 1;
numFramesLost = 0;
fqyoffset = zeros(1000,1);
while stIdx < length(rxIn)
    isFrameLost = false;

    % Use one PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*sps;

    % In the last frame, consider all of the remaining samples in the received
    % waveform.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);
    stIdx = endIdx; % Update start index after extracting required data
    if ~isLastFrame
        % Filter the received data
        filteredRx = rrcfilt(rxData); % RRC filter

        % Synchronize the symbol timing
        syncsym = symsyncobj(filteredRx);

        % Synchronize the data to the frame boundaries
        syncidx = HelperCCSDSFACMFrameSync(syncsym,rxParams.RefFM);

        % Consider one complete PL frame beginning with a header. Append
        % zeros if data is not sufficient. This situation typically happens
        % when samples are lost while doing timing synchronization or when
        % synchronization is lost.
        tempSym = [leftOutSym;syncsym(1:syncidx(1)-1)];
        leftOutSym = syncsym(syncidx(1):end);
        if length(tempSym)<n*16+320 % 16 code blocks and 320 header symbols
            fllIn = [tempSym;zeros(n*16+320-length(tempSym),1)];
        else % length(tempSym)>=n*16 + 320
            fllIn = tempSym(1:n*16+320);
        end

        % Track the frequency offset
        [fllOut,fqyoffset(frameIndex)] = fll(fllIn);

        % Estimate and compensate the CFO from the FM.
        cfoEst = HelperCCSDSFACMFrequencyEstimate(fllOut(1:256), ...
            rxParams.RefFM,simParams.SymbolRate);
        fineCFOSync.FrequencyOffset = -cfoEst;
        fqysyncedFM = fineCFOSync(fllOut(1:256));

        % Estimate and compensate for the phase offset in each frame
        % independently. Remove the pilots in the process.
        [noPilotsSym,frameDescriptor] = ...
            HelperCCSDSFACMPhaseRecovery(fllOut,rxParams.PilotSeq,rxParams.RefFM);
        agcIn = [frameDescriptor;noPilotsSym];

        % Estimate the SNR. See CCSDS 130.11-G-1 Section 5.5 [2].
        SNREst(idxTemp+1) = HelperCCSDSFACMSNREstimate(fqysyncedFM(1:256), ...
            rxParams.RefFM);
    end
end

```

```

% Average the estimated SNR over multiple frames
finalSNREst = mean(SNREst);
idxTemp = mod(idxTemp+1,6);

% Pass the signal through DAGC
if frameIndex >= snrAveragingFactor
    [agcRecovered,G] = HelperDigitalAutomaticGainControl(agcIn,finalSNREst,G);
else
    agcRecovered = agcIn;
end

if showVisualizations == true
    % Plot the constellation.
    release(constelDiag)
    constelDiag(agcRecovered(:))
end

% Recover the ACM format pilots availability indicator.
[ACMFormat, hasPilots,decFail] = HelperCCSDSFACMFDRecover(agcRecovered(1:64));

if decFail
    isFrameLost = true;
end

if (ACMFormat ~= cfgCCSDSFACM.ACMFormat) || (hasPilots ~= cfgCCSDSFACM.HasPilots)
    isFrameLost = true;
end

% Continue further processing only if the frame is not lost.
if ~isFrameLost
    % De-randomize the PL-frame.
    derandomized = agcRecovered(65:end).*conj(rxParams.PLRandomSymbols(:));

    % Demodulate the signal
    nVar = 1/finalSNREst;
    demodulated = reshape(HelperCCSDSFACMDemodulate(derandomized,ACMFormat,nVar), ...
        [],16);

    fullFrameDecoded = zeros(16*phyParams.K,1);

    for iCodeWord = 1:16
        decoded = HelperSCCCDecode(demodulated(:,iCodeWord),ACMFormat,numIterations);
        fullFrameDecoded((iCodeWord-1)*phyParams.K+1:iCodeWord*phyParams.K) = ...
            decoded;
    end

    % Extract TFs.
    [initBits,decodedBuffer,extraBits] = ...
        HelperCCSDSFACMTFSynchronize([extraBits;fullFrameDecoded], ...
            phyParams.ASM, ...
            phyParams.NumInputBits);
    PRNSeq = satcom.internal.ccsds.tmrndseq(phyParams.NumInputBits);
    if ~isempty(decodedBuffer)
        finalBits = xor(decodedBuffer(33:end,:)>0,PRNSeq);
    else
        isFrameLost = true;
        extraBits = [];
    end
end

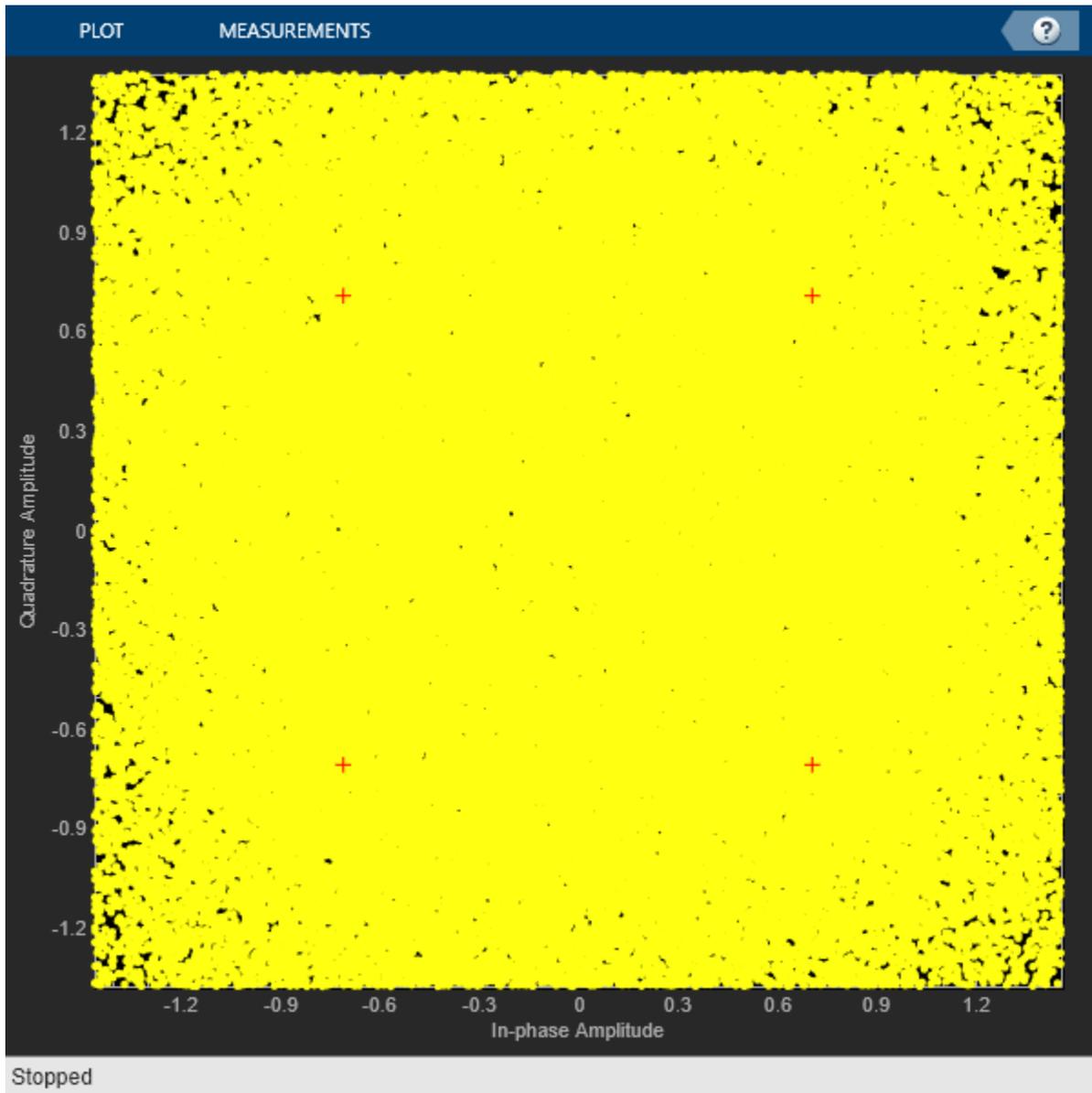
```

```
end

if isFrameLost
    numFramesLost = numFramesLost + 1;
end

% Find BER
if frameIndex > simParams.InitialSyncFrames && ~isFrameLost
    berinfo = HelperBitErrorRate(bits, finalBits, berinfo);
    disp(['frameIndex = ' num2str(frameIndex) '. BER = ' ...
        num2str(berinfo.BitErrorRate)])
end
frameIndex = frameIndex + 1;
end
end

frameIndex = 16. BER = 0
frameIndex = 17. BER = 0
frameIndex = 18. BER = 0
frameIndex = 19. BER = 0
frameIndex = 20. BER = 0
frameIndex = 21. BER = 0
frameIndex = 22. BER = 0
frameIndex = 23. BER = 0
frameIndex = 24. BER = 0
frameIndex = 25. BER = 0
frameIndex = 26. BER = 0
frameIndex = 27. BER = 0
frameIndex = 28. BER = 0
frameIndex = 29. BER = 0
```



```
frameIndex = 30. BER = 0
```

```
disp(['ACM format = ' num2str(cfgCCSDSFACM.ACMFormat) ' . Es/No(dB) = ' ...
      num2str(simParams.EsNodB) ' . BER = ' num2str(berinfo.BitErrorRate)])
```

```
ACM format = 1. Es/No(dB) = 1. BER = 0
```

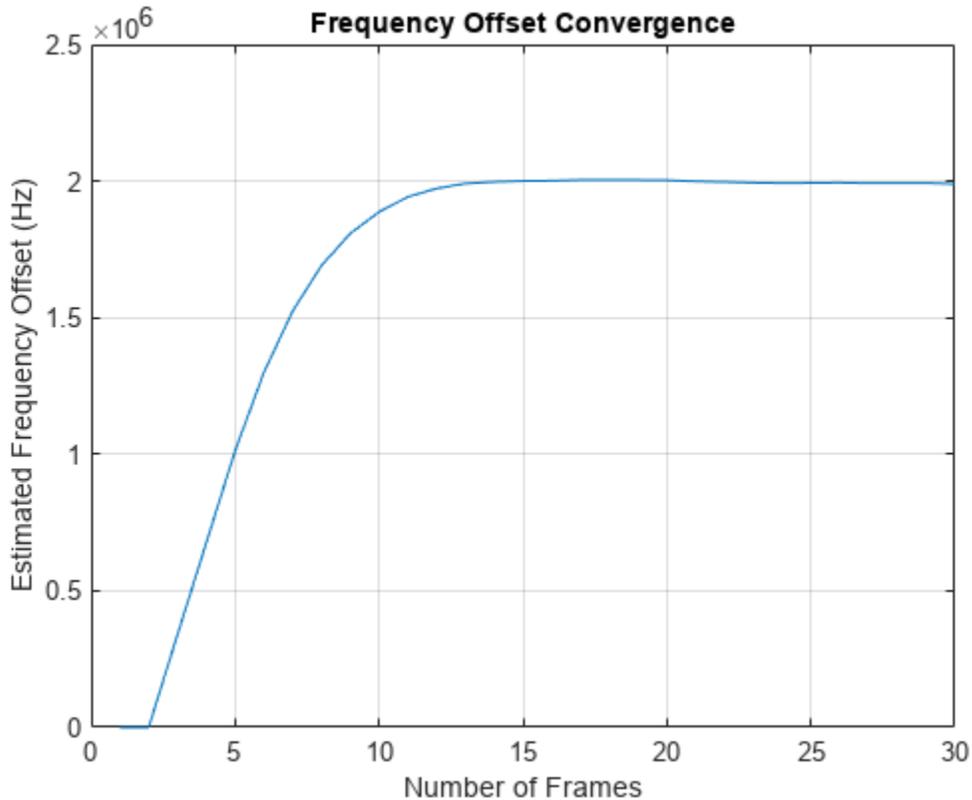
This plot shows the frequency convergence of the estimated frequency offset. This plot shows the number of frames required for the FLL to converge. The plot shows that the frequency offset converges even at a very low SNR (0 dB Es/No). This observation shows that the FLL can operate effectively at low SNR values.

```
if showVisualizations == true
    plot(fqyoffset(1:frameIndex-1));
    grid on
```

```

ylabel('Estimated Frequency Offset (Hz)')
xlabel('Number of Frames')
title('Frequency Offset Convergence')
end

```



Further Exploration

This example shows the calculation of BER for one ACM format at one SNR point. Run BER simulations for multiple SNR points and multiple ACM formats.

This example uses Es/No as the SNR metric. To convert this to energy per bit to noise power ratio (Eb/No), use this code.

```

% ccsdsWaveGen = ccsdsTMWaveformGenerator('WaveformSource','flexible advanced coding and modulatio
%   'ACMFormat',cfgCCSDSFACM.ACMFormat);
% codeRate = ccsdsWaveGen.info.ActualCodeRate;
% modOrder = ccsdsWaveGen.info.NumBitsPerSymbol;
% EbNodB = simParams.EsNodB - 10*log10(codeRate) - 10*log10(modOrder);

```

Appendix

The example uses these helper files:

- HelperBitErrorRate.m - Calculate bit error rate
- HelperCCSDSFACMDemodulate.m - Demodulate the received FACM signal
- HelperCCSDSFACMFDRecover.m - Recover frame descriptor

- HelperCCSDSFACMFLL.m - FLL for carrier frequency recovery
- HelperCCSDSFACMFrequencyEstimate.m - Estimate frequency offset in FM
- HelperCCSDSFACMFrameMarker.m - Generate reference FM
- HelperCCSDSFACMFrameSync.m - Estimate frame beginning
- HelperCCSDSFACMPhaseRecovery.m - Recover phase in the signal
- HelperCCSDSFACMReferenceConstellation.m - Generate reference constellation for a given ACM format
- HelperCCSDSFACMRxInputGenerate.m - Generate transmitter waveform, model RF impairments, and add AWGN
- HelperCCSDSFACMSNREstimate.m - Estimate SNR in received signal
- HelperCCSDSFACMTFSynchronize.m - Transfer frame synchronization
- HelperDigitalAutomaticGainControl.m - Digital automatic gain control
- HelperDopplerShift.m - Model sinusoidal Doppler profile
- HelperSCCCDecode.m - Iterative decoder for SCCC

References

[1] CCSDS 131.2-B-1. Blue Book. Issue 1. "Flexible Advanced Coding and Modulation Scheme for High Rate Telemetry Applications." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, March 2012.

[2] CCSDS 130.11-G-1. Green Book. Issue 1. "SCCC—Summary of Definition and Performance." *Informational Report Concerning Space Data System Standards*. Washington, D.C.: CCSDS, April 2019.

[3] Rice, Michael. *Digital Communications: A Discrete-Time Approach*. Pearson/Prentice Hall, 2008.

See Also

Objects

ccsdSTMWaveformGenerator

Related Examples

- "Calculate Latency and Doppler in a Satellite Scenario" on page 1-49
- "End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections" on page 4-13
- "End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections" on page 4-2

End-to-End DVB-S2 Simulation with RF Impairments and Corrections

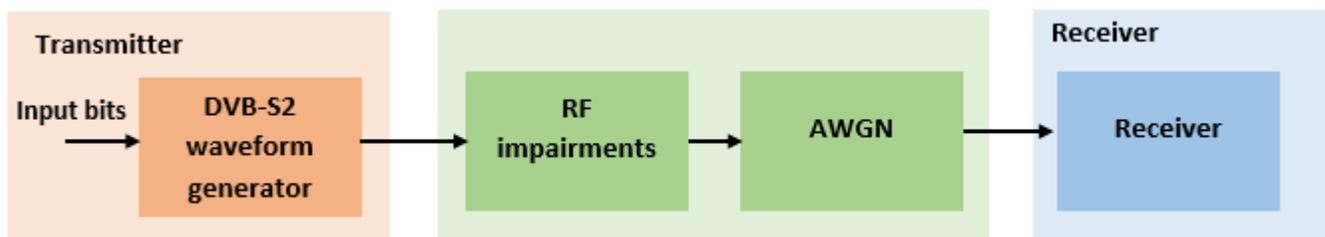
This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) link that has constant coding and modulation. The example describes the symbol timing and carrier synchronization strategies in detail, emphasizing how to estimate the RF front-end impairments under heavy noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

DVB-S2 receivers are subjected to large carrier frequency errors in the order of 20% of the input symbol rate and substantial phase noise. The use of powerful forward error correction (FEC) mechanisms, such as Bose–Chaudhuri–Hocquenghem (BCH) and low density parity check (LDPC) codes, caused the DVB-S2 system to work at very low energy per symbol to noise power spectral density ratio (E_s/N_o) values, close to the Shannon limit.

ETSI EN 302 307-1 Section 6 Table 13 [1] on page 4-52 summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. The operating E_s/N_o range for different transmission modes can be considered as +2 or -2 dB from the E_s/N_o point where QEF performance is observed. Because the operating E_s/N_o range is low, carrier and symbol timing synchronization strategies are challenging design problems.

This diagram summarizes the example workflow.



Main Processing Loop

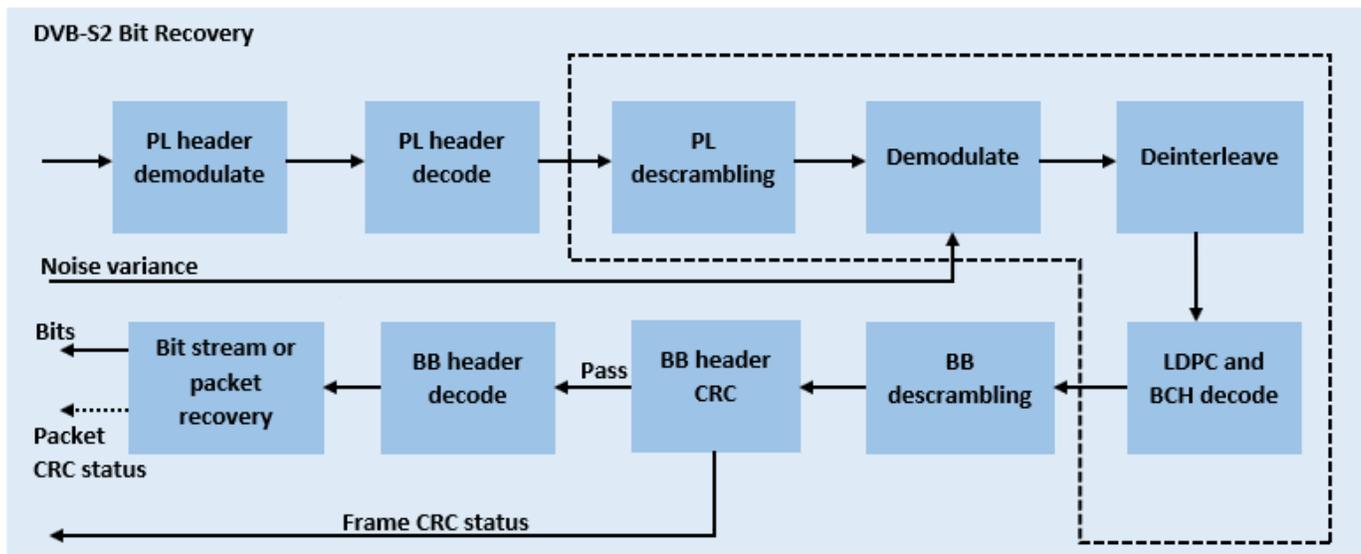
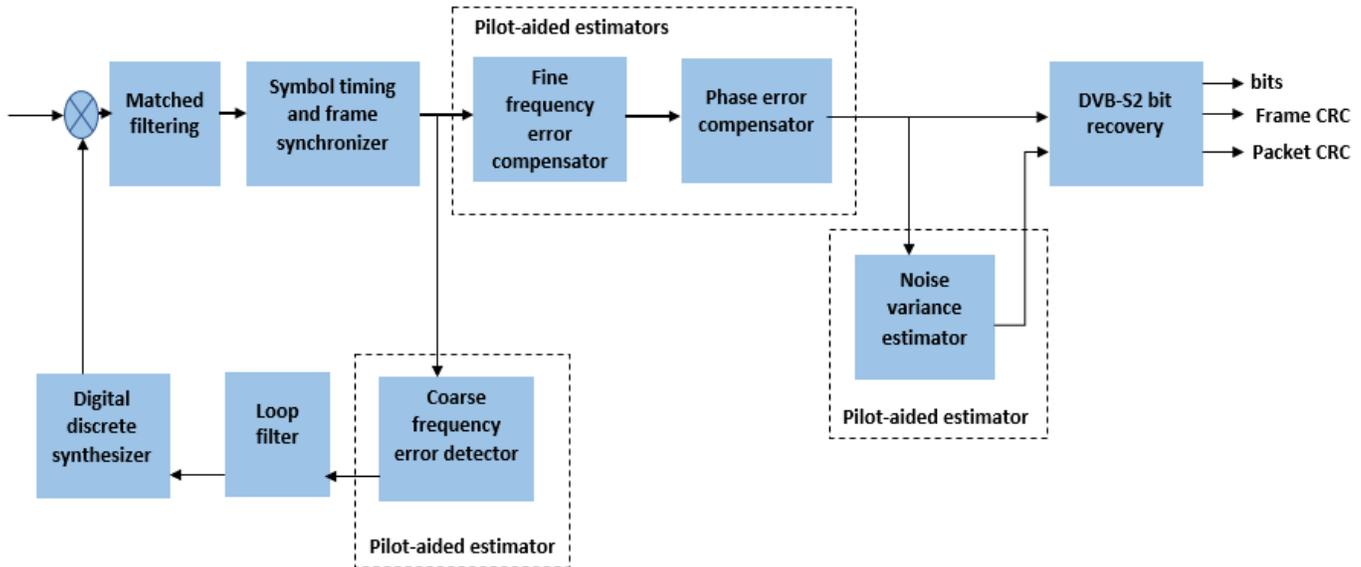
The example processes 25 physical layer (PL) frames of data with the E_s/N_o set to 20 dB, and then computes the BER and PER. Carrier frequency offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

At the receiver, after matched filtering, timing and carrier recovery operations are run to recover the transmitted data. To extract PL frames, the distorted waveform is processed through various timing and carrier recovery strategies to extract PL frames. The carrier recovery algorithms are pilot-aided. To decode the data frames, the physical layer transmission parameters such as modulation scheme, code rate, and FEC frame type, are recovered from the PL header. To regenerate the input bit stream, the baseband (BB) header is decoded.

Because the DVB-S2 standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet

cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured.

These block diagrams show the synchronization and input bit recovery workflows.



Download DVB-S2 LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2 LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
```

```

end
addpath('s2xLDPCParityMatrices');
end

```

DVB-S2 Configuration in Pilot-Aided Mode

Specify the `cfgDVBS2` structure to define DVB-S2 transmission configuration parameters. The `ScalingMethod` property applies when `MODCOD` is in the range [18, 28] (that is, when the modulation scheme is APSK only). `UPL` property is applicable when you set the `StreamFormat` to "GS".

```

cfgDVBS2.StreamFormat = "TS";
cfgDVBS2.FECFrame = "normal";
cfgDVBS2.MODCOD = 18; % 16APSK 2/3
cfgDVBS2.DFL = 42960;
cfgDVBS2.ScalingMethod = "Unit average power";
cfgDVBS2.RolloffFactor = 0.35;
cfgDVBS2.HasPilots = true;
cfgDVBS2.SamplesPerSymbol = 2

cfgDVBS2 = struct with fields:
    StreamFormat: "TS"
    FECFrame: "normal"
    MODCOD: 18
    DFL: 42960
    ScalingMethod: "Unit average power"
    RolloffFactor: 0.3500
    HasPilots: 1
    SamplesPerSymbol: 2

```

Simulation Parameters

The DVB-S2 standard supports flexible channel bandwidths. Use a typical channel bandwidth such as 36 MHz. The channel bandwidth can be varied. The coarse frequency synchronization algorithm implemented in this example can track carrier frequency offsets up to 20% of the input symbol rate. The symbol rate is calculated as $B/(1+R)$, where B is the channel bandwidth, and R is the transmit filter roll-off factor. The algorithms implemented in this example can correct the sampling clock offset up to 10 ppm.

```

simParams.sps = cfgDVBS2.SamplesPerSymbol; % Samples per symbol
simParams.numFrames = 25; % Number of frames to be processed
simParams.chanBW = 36e6; % Channel bandwidth in Hertz
simParams.cfo = 3e6; % Carrier frequency offset in Hertz
simParams.sco = 5; % Sampling clock offset in parts
% per million

simParams.phNoiseLevel = ; % Phase noise level provided as
% 'Low', 'Medium', or 'High'
simParams.EsNodB = 20; % Energy per symbol to noise ratio
% in decibels

```

This table defines the phase noise mask (dBc/Hz) used to generate the phase noise applied to the transmitted signal.

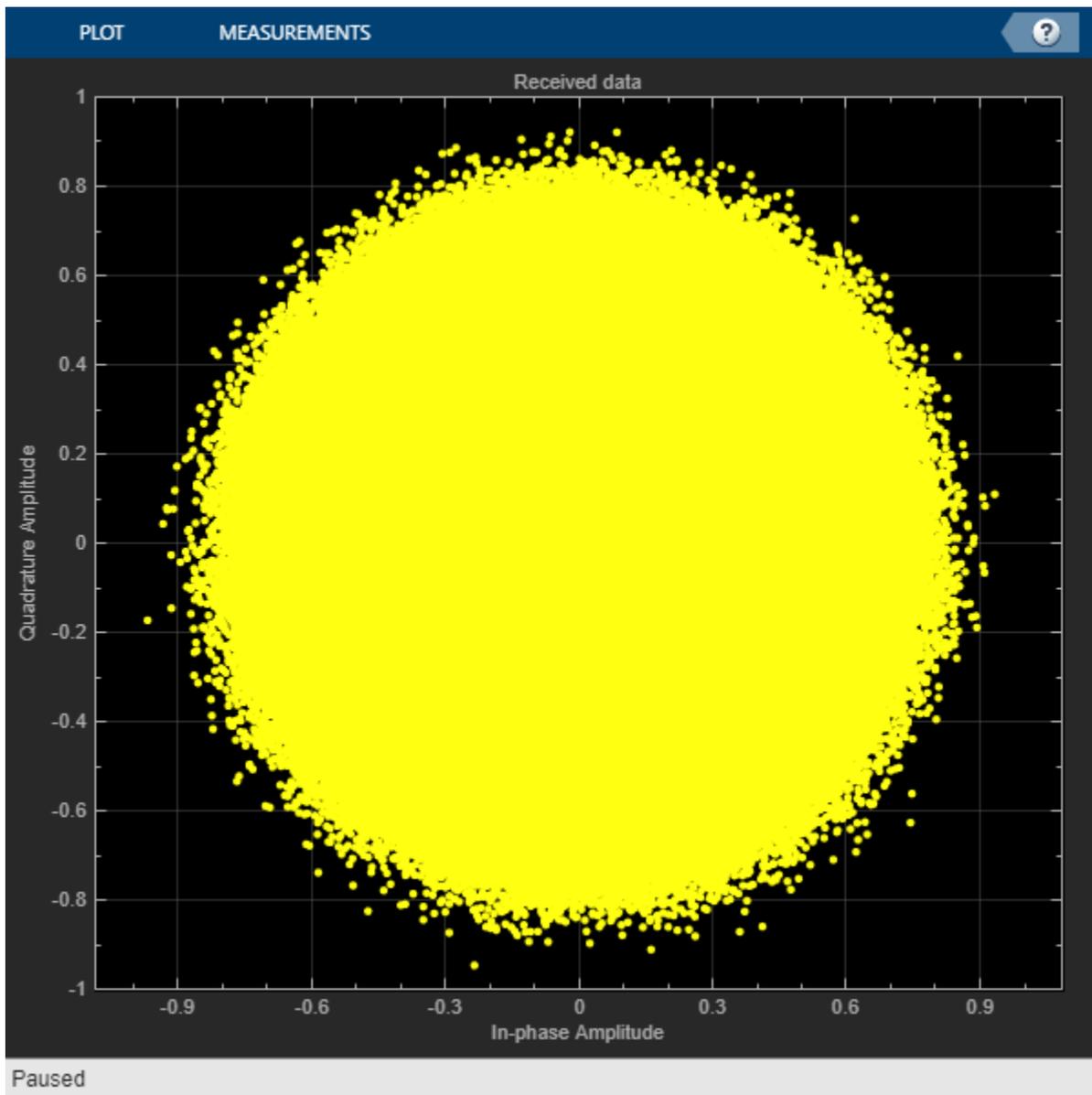
Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz
Low	-73	-83	-93	-112	-128
Medium	-59	-77	-88	-94	-104
High	-25	-50	-73	-85	-103

Generate DVB-S2 Waveform Distorted with RF Impairments

To create a DVB-S2 waveform, use the `HelperDVBS2RxInputGenerate` helper function with the `simParams` and `cfgDVBS2` structures as inputs. The function returns the data signal, transmitted and received waveforms, and a receiver processing structure. The received waveform is impaired with carrier frequency, timing phase offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

```
[data,txOut,rxIn,rxParams] = HelperDVBS2RxInputGenerate(cfgDVBS2,simParams);
```

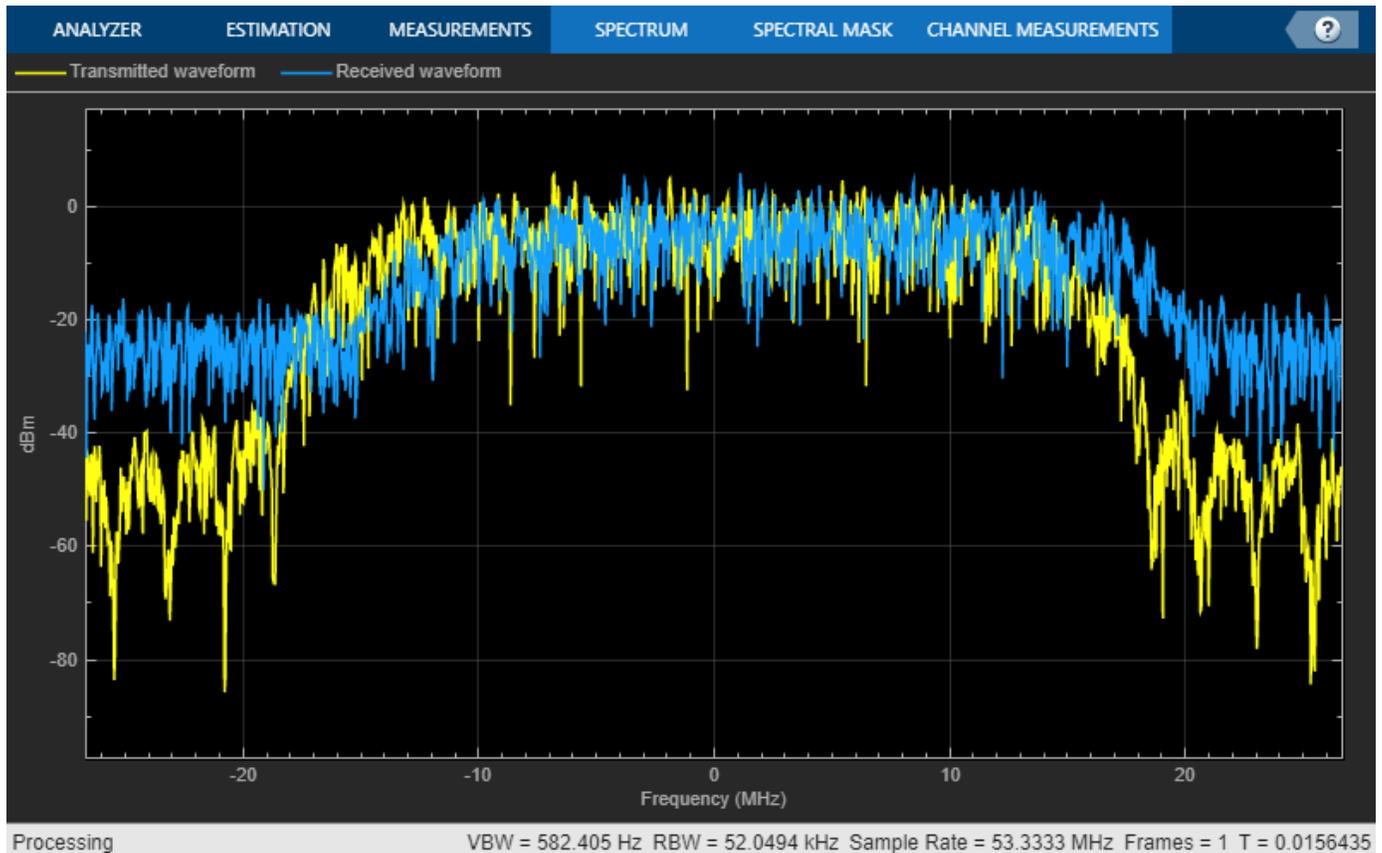
```
% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:length(txOut)))
```



```

% Transmitted and received signal spectrum visualization
Rsymb = simParams.chanBW/(1 + cfgDVBS2.RolloffFactor);
Fsamp = Rsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'ShowLegend',true);
specAn([txOut, rxIn(1:length(txOut))]);

```



Configure Receiver Parameters

At the receiver, symbol timing synchronization is performed on the received data and is then followed by frame synchronization. The receiver algorithms include coarse and fine frequency impairment correction algorithms. The carrier frequency estimation algorithm can track carrier frequency offsets up to 20% of the input symbol rate. The coarse frequency estimation, implemented as a frequency locked loop (FLL), reduces the frequency offset to a level that the fine frequency estimator can track. The preferred loop bandwidth for symbol timing and coarse frequency compensation depends on the E_s/N_o setting.

A block of 36 pilots is repeated every 1476 symbols. The coarse frequency error estimation uses 34 of the 36 pilot symbols. The ratio of used pilots per block (34) and pilot periodicity (1476) is 0.023. Using the 0.023 value as a scaling factor for the coarse frequency synchronizer loop bandwidth is preferred.

When you decrease the E_s/N_o , you can reduce the loop bandwidth to filter out more noise during acquisition. The number of frames required for the symbol synchronizer and coarse FLL to converge depends on the loop bandwidth setting.

The frame synchronization uses the PL header. Because the carrier synchronization is data-aided, the frame synchronization must detect the start of frame accurately. E_s/N_o plays a crucial role in determining the accuracy of the frame synchronization. When QPSK modulated frames are being recovered at E_s/N_o values below 3 dB, the frame synchronization must be performed on multiple frames for accurate detection.

The fine frequency estimation can track carrier frequency offsets up to 4% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency errors less than 0.02% of the input symbol rate. Fine phase compensation is only required for APSK modulation schemes in the presence of significant phase noise.

These settings are assigned in the rxParams structure for synchronization processing. For details on how to set these parameters for low E_s/N_0 values, see the Further Exploration on page 4-50 section.

```

rxParams.carrSyncLoopBW = 1e-2*0.023;           % Coarse frequency estimator loop bandwidth
                                                % normalized by symbol rate
rxParams.symbSyncLoopBW = 8e-3;                % Symbol timing synchronizer loop bandwidth
                                                % normalized by symbol rate
rxParams.symbSyncLock = 6;                     % Number of frames required for symbol
                                                % timing error convergence
rxParams.frameSyncLock = 1;                    % Number of frames required for frame
                                                % synchronization
rxParams.coarseFreqLock = 3;                   % Number of frames required for coarse
                                                % frequency acquisition
rxParams.fineFreqLock = 6;                     % Number of frames required for fine
                                                % frequency estimation
rxParams.hasFinePhaseCompensation = false;     % Flag to indicate whether fine phase
                                                % compensation is used
rxParams.finePhaseSyncLoopBW = 3.5e-4;        % Fine phase compensation loop bandwidth
                                                % normalized by symbol rate

% Total frames taken for symbol timing and coarse frequency lock to happen
rxParams.initialTimeFreqSync = rxParams.symbSyncLock + rxParams.frameSyncLock + ...
    rxParams.coarseFreqLock;
% Total frames used for overall synchronization
rxParams.totalSyncFrames = rxParams.initialTimeFreqSync + rxParams.fineFreqLock;

% Create time frequency synchronization System object by using
% HelperDVBS2TimeFreqSynchronizer helper object
timeFreqSync = HelperDVBS2TimeFreqSynchronizer( ...
    'CarrSyncLoopBW',rxParams.carrSyncLoopBW, ...
    'SymbSyncLoopBW',rxParams.symbSyncLoopBW, ...
    'SamplesPerSymbol',simParams.sps, ...
    'DataFrameSize',rxParams.xFecFrameSize, ...
    'SymbSyncTransitFrames',rxParams.symbSyncLock, ...
    'FrameSyncAveragingFrames',rxParams.frameSyncLock);

% Create fine phase compensation System object by using
% HelperDVBS2FinePhaseCompensator helper object. Fine phase
% compensation is only required for 16 and 32 APSK modulated frames
if cfgDVBS2.MODCOD >= 18 && rxParams.hasFinePhaseCompensation
    finePhaseSync = HelperDVBS2FinePhaseCompensator( ...
        'DataFrameSize',rxParams.xFecFrameSize, ...
        'NormalizedLoopBandwidth',rxParams.finePhaseSyncLoopBW);
end

normFlag = cfgDVBS2.MODCOD >= 18 && strcmpi(cfgDVBS2.ScalingMethod,'Outer radius as 1');

% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

```

```

% Initialize data indexing variables
stIdx = 0;
dataSize = rxParams.inputFrameSize;
plFrameSize = rxParams.plFrameSize;
dataStInd = rxParams.totalSyncFrames + 1;
isLastFrame = false;
symSyncOutLen = zeros(rxParams.initialTimeFreqSync,1);

```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data and recover the input bit stream, the distorted DVB-S2 waveform samples are processed one frame at a time by following these steps.

- 1 Apply matched filtering, outputting at the rate of two samples per symbol.
- 2 Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate. The Gardner TED is not data-aided, so it is performed before carrier synchronization.
- 3 Apply frame synchronization to detect the start of frame and to identify the pilot positions.
- 4 Estimate and apply coarse frequency offset correction.
- 5 Estimate and apply fine frequency offset correction.
- 6 Estimate and compensate for residual carrier frequency and phase noise.
- 7 Decode the PL header and compute the transmission parameters.
- 8 Demodulate and decode the PL frames.
- 9 Perform CRC check on the BB header, if the check passes, recover the header parameters.
- 10 Regenerate the input stream of data or packets from BB frames.

```

while stIdx < length(rxIn)

    % Use one DVB-S2 PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are considered.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % After coarse frequency offset loop is converged, the FLL works with a
    % reduced loop bandwidth.
    if rxParams.frameCount < rxParams.initialTimeFreqSync
        coarseFreqLock = false;
    else
        coarseFreqLock = true;
    end

    % Retrieve the last frame samples.
    if isLastFrame
        resSymb = plFrameSize - length(rxParams.cfBuffer);
        resSampCnt = resSymb*rxParams.sps - length(rxData);
        if resSampCnt >= 0 % Inadequate number of samples to fill last frame
            syncIn = [rxData;zeros(resSampCnt, 1)];
        else % Excess samples are available to fill last frame
            syncIn = rxData(1:resSymb*rxParams.sps);
        end
    end
end

```

```

else
    syncIn = rxData;
end

% Apply matched filtering, symbol timing synchronization, frame
% synchronization, and coarse frequency offset compensation.
[coarseFreqSyncOut, syncIndex, phEst] = timeFreqSync(syncIn, coarseFreqLock);
if rxParams.frameCount <= rxParams.initialTimeFreqSync
    symSyncOutLen(rxParams.frameCount) = length(coarseFreqSyncOut);
    if any(abs(diff(symSyncOutLen(1:rxParams.frameCount))) > 5)
        error(['Symbol timing synchronization failed. The loop will not ' ...
            'converge. No frame will be recovered. Update the symbSyncLoopBW ' ...
            'parameter according to the EsNo setting for proper loop convergence.']);
    end
end

rxParams.syncIndex = syncIndex;

% The PL frame start index lies somewhere in the middle of the chunk being processed.
% From fine frequency estimation onwards, the processing happens as a PL frame.
% A buffer is used to store symbols required to fill one PL frame.
if isLastFrame
    resCnt = resSymb - length(coarseFreqSyncOut);
    if resCnt <= 0
        fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut(1:resSymb)];
    else
        fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut; zeros(resCnt, 1)];
    end
elseif rxParams.frameCount > 1
    fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut(1:rxParams.plFrameSize-length(rxParams
end

% Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
% helper function.
% Add 1 to the conditional check because the buffer used to get one PL
% frame introduces a delay of one to the loop count.
if (rxParams.frameCount > rxParams.initialTimeFreqSync + 1) && ...
    (rxParams.frameCount <= rxParams.totalSyncFrames + 1)
    rxParams.fineFreqCorrVal = HelperDVBS2FineFreqEst( ...
        fineFreqIn(rxParams.pilotInd), rxParams.numPilots, ...
        rxParams.refPilots, rxParams.fineFreqCorrVal);
end
if rxParams.frameCount >= rxParams.totalSyncFrames + 1
    fineFreqLock = true;
else
    fineFreqLock = false;
end

if fineFreqLock
    % Normalize the frequency estimate by the input symbol rate
    % freqEst = angle(R)/(pi*(N+1)), where N (18) is the number of elements
    % used to compute the mean of auto correlation (R) in
    % HelperDVBS2FineFreqEst.
    freqEst = angle(rxParams.fineFreqCorrVal)/(pi*(19));

    % Generate the symbol indices using frameCount and plFrameSize.
    % Subtract 2 from the rxParams.frameCount because the buffer used to get one
    % PL frame introduces a delay of one to the count.

```

```

ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
phErr = exp(-1j*2*pi*freqEst*ind);
fineFreqOut = fineFreqIn.*phErr(:);

% Estimate the phase error estimation by using the HelperDVBS2PhaseEst
% helper function.
[phEstRes,rxParams.prevPhaseEst] = HelperDVBS2PhaseEst( ...
    fineFreqOut(rxParams.pilotInd),rxParams.refPilots,rxParams.prevPhaseEst);

% Compensate for the residual frequency and phase offset by using
% the
% HelperDVBS2PhaseCompensate helper function.
% Use two frames for initial phase error estimation. Starting with the
% second frame, use the phase error estimates from the previous frame and
% the current frame in compensation.
% Add 3 to the frame count comparison to account for delays: One
% frame due to rxParams.cfBuffer delay and two frames used for phase
% error estimate.
if rxParams.frameCount >= rxParams.totalSyncFrames + 3
    coarsePhaseCompOut = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
        rxParams.pilotEst,rxParams.pilotInd,phEstRes(2));
    % MODCOD >= 18 corresponds to APSK modulation schemes
    if cfgDVBS2.MODCOD >= 18 && rxParams.hasFinePhaseCompensation
        phaseCompOut = finePhaseSync(coarsePhaseCompOut);
    else
        phaseCompOut = coarsePhaseCompOut;
    end
end

rxParams.ffBuffer = fineFreqOut;
rxParams.pilotEst = phEstRes;

% The phase compensation on the data portion is performed by
% interpolating the phase estimates computed on consecutive pilot
% blocks. The second phase estimate is not available for the data
% portion after the last pilot block in the last frame. Therefore,
% the slope of phase estimates computed on all pilot blocks in the
% last frame is extrapolated and used to compensate for the phase
% error on the final data portion.
if islastFrame
    pilotBlkLen = 36; % Symbols
    pilotBlkFreq = 1476; % Symbols
    avgSlope = mean(diff(phEstRes(2:end)));
    chunkLen = rxParams.plFrameSize - rxParams.pilotInd(end) + ...
        rxParams.pilotInd(pilotBlkLen);
    estEndPh = phEstRes(end) + avgSlope*chunkLen/pilotBlkFreq;
    coarsePhaseCompOut1 = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
        rxParams.pilotEst,rxParams.pilotInd,estEndPh);
    % MODCOD >= 18 corresponds to APSK modulation schemes
    if cfgDVBS2.MODCOD >= 18 && rxParams.hasFinePhaseCompensation
        phaseCompOut1 = finePhaseSync(coarsePhaseCompOut1);
    else
        phaseCompOut1 = coarsePhaseCompOut1;
    end
end
end

% Recover the input bit stream.

```

```

if rxParams.frameCount >= rxParams.totalSyncFrames + 3
    isValid = true;
    if isLastFrame
        syncOut = [phaseCompOut; phaseCompOut1];
    else
        syncOut = phaseCompOut;
    end
else
    isValid = false;
    syncOut = [];
end

% Update the buffers and counters.
rxParams.cfBuffer = coarseFreqSyncOut(rxParams.syncIndex:end);
rxParams.syncIndex = syncIndex;
rxParams.frameCount = rxParams.frameCount + 1;

if isValid % Data valid signal

    % Decode the PL header by using the HelperDVBS2PLHeaderRecover helper
    % function. Start of frame (SOF) is 26 symbols, which are discarded
    % before header decoding. They are only required for frame
    % synchronization.
    rxPLSCode = syncOut(27:90);
    [M,R,fecFrame,pilotStat] = HelperDVBS2PLHeaderRecover(rxPLSCode);
    xFECFrameLen = fecFrame/log2(M);
    % Validate the decoded PL header.
    if M ~= rxParams.modOrder || R ~= rxParams.codeRate || ...
        fecFrame ~= rxParams.cwLen || ~pilotStat
        fprintf('%s\n','PL header decoding failed')
        dataStInd = dataStInd + 1;
    else % Demodulation and decoding
        for frameCnt = 1:length(syncOut)/plFrameSize
            rxFrame = syncOut((frameCnt-1)*plFrameSize+1:frameCnt*plFrameSize);
            % Estimate noise variance by using
            % HelperDVBS2NoiseVarEstimate helper function.
            nVar = HelperDVBS2NoiseVarEstimate(rxFrame,rxParams.pilotInd,...
                rxParams.refPilots,normFlag);
            % The data begins at symbol 91 (after the header symbols).
            rxDataFrame = rxFrame(91:end);
            % Recover the BB frame.
            rxBBFrame = satcom.internal.dvbs.s2BBFrameRecover(rxDataFrame,M,R, ...
                fecFrame,pilotStat,nVar,false);
            % Recover the input bit stream by using
            % HelperDVBS2StreamRecover helper function.
            if strcmpi(cfgDVBS2.StreamFormat,'GS') && ~rxParams.UPL
                [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame);
                if ~isFrameLost && length(decBits) ~= dataSize
                    isFrameLost = true;
                end
            else
                [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame);
                if ~isFrameLost && length(decBits) ~= dataSize
                    isFrameLost = true;
                    pktCRC = zeros(0,1,'logical');
                end
            end
            % Compute the packet error rate for TS or GS packetized
            % mode.

```

```

        pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
        pktsRec = pktsRec + numel(pktCRC);
    end
    if ~isFrameLost
        ts = sprintf('%s', 'BB header decoding passed. ');
    else
        ts = sprintf('%s', 'BB header decoding failed. ');
    end
    % Compute the number of frames lost. CRC failure of baseband header
    % is considered a frame loss.
    numFramesLost = isFrameLost + numFramesLost;
    fprintf('%s(Number of frames lost = %ld)\n', ts, numFramesLost)
    % Compute the bits in error.
    bitInd = (dataStInd-1)*dataSize+1:dataStInd*dataSize;
    if isLastFrame && ~isFrameLost
        bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
    else
        if ~isFrameLost
            bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
        end
    end
    dataStInd = dataStInd + 1;
end
end
end
end
stIdx = endIdx;
end

```

```

BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)

```

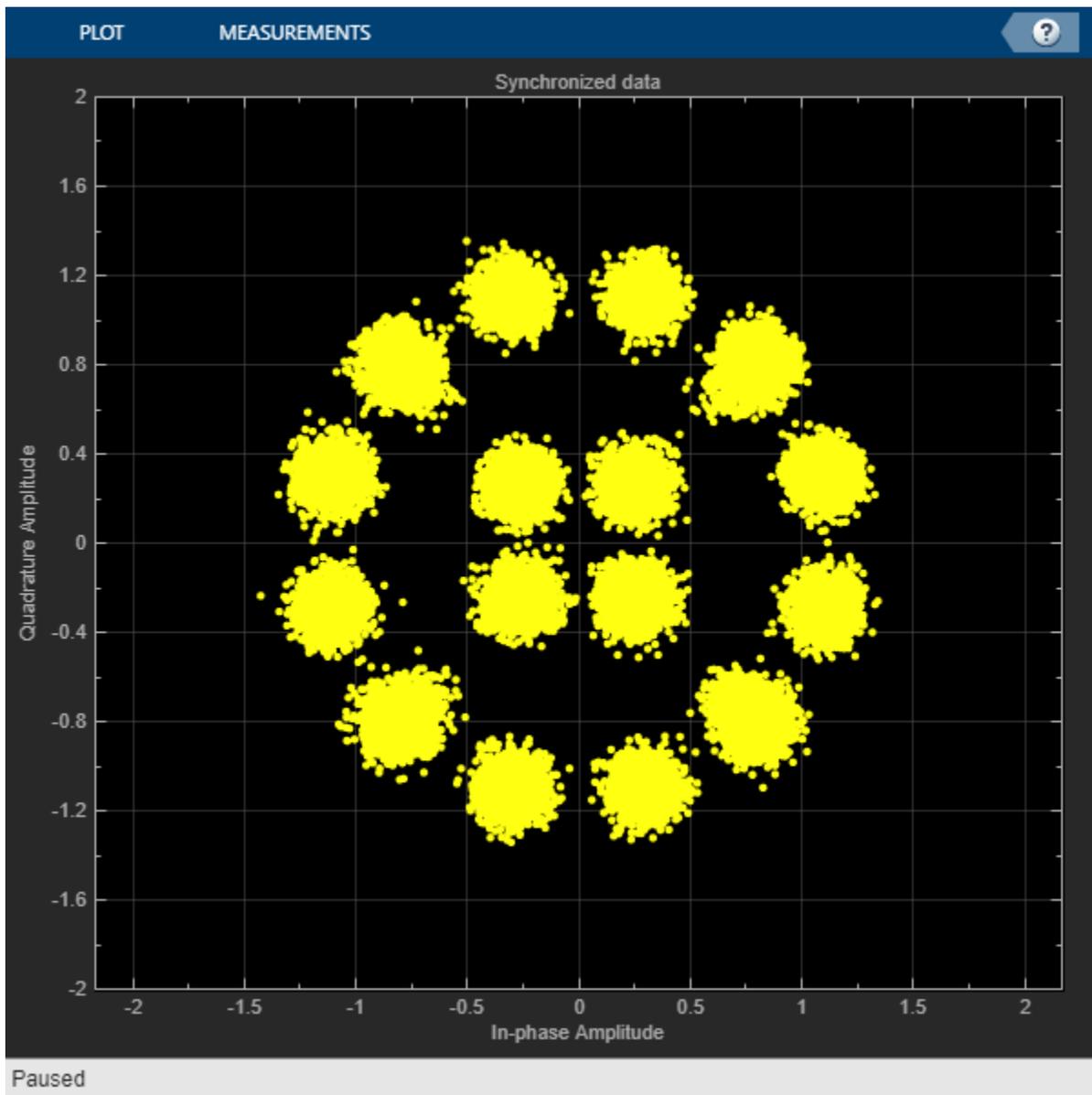
Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```

% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title', 'Synchronized data', ...
    'XLimits', [-2 2], 'YLimits', [-2 2], ...
    'ShowReferenceConstellation', false);
syncConst(syncOut)

```



```

pause(0.5)
% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames-rxParams.totalSyncFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        ber = bitsErr/((dataStInd-rxParams.totalSyncFrames)*dataSize);
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
    else

```

```

if strcmpi(cfgDVBS2.StreamFormat, 'TS')
    pktLen = 1504;
else
    pktLen = cfgDVBS2.UPL;      % UP length including sync byte
end
ber = bitsErr/(pktsRec*pkLen);
per = pktsErr/pktsRec;
fprintf('PER: %1.2e\n',per)
fprintf('BER: %1.2e\n',ber)
end
end

```

PER: 0.00e+00

BER: 0.00e+00

Further Exploration

The operating E_s/N_o range of the DVB-S2 standard being very low requires the normalized loop bandwidth of the symbol synchronizer and coarse FLL to be very small for accurate estimation. These parameters are set via `rxParams.symbSyncLoopBW` and `rxParams.carrSyncLoopBW`.

Configure Symbol Timing Synchronization Parameters

Try running the simulation using the symbol timing synchronizer configured with a normalized loop bandwidth of $1e-4$. With loop bandwidths at this level, this table shows the typical number of frames required for convergence of the symbol timing loop for specific modulation schemes and 'normal' FEC frames.

Modulation scheme	Number of frames
QPSK	10 - 12
8 PSK	15 - 17
16 APSK	20 - 22
32 APSK	25 - 27

For 'short' FEC frames, the number of frames used for symbol timing synchronization is thrice the number required for 'normal' FEC frames. If the symbol timing loop doesn't converge, try decreasing the `rxParams.symbSyncLoopBW` and increasing the `rxParams.symbSyncLock`. To achieve convergence of the timing loop, the ratio `rxParams.symbSyncLoopBW/simParams.sps` must be greater than $1e-5$.

Configure Frame Synchronization Parameters

Choose a `rxParams.symbSyncLock` value from the table provided in Configure Symbol Timing Synchronization Parameters on page 4-50 section. Set `rxParams.frameSyncLock` as a value in the range of [5, 15] frames based on the E_s/N_o setting. If the output is not as expected, increase the number of frames required for frame synchronization.

Configure Carrier Synchronization Parameters

Try running the simulation using the coarse FLL configured with a normalized loop bandwidth of $1e-4*0.023$ for PSK signals and $4e-4*0.023$ for APSK signals.

When you set the FECFrame property to 'normal', set the rxParams.coarseFreqLock property to 20. When you set the FECFrame property to 'short', set the rxParams.coarseFreqLock property to 80. Set simParams.EsNodB to the lowest E_s/N_o for the chosen modulation scheme from ETSI EN 302 307-1 Section 6 [1] on page 4-52. For the HelperDVBS2TimeFreqSynchronizer system object, set its properties as discussed in the previous sections based on the chosen configuration.

```
% timeFreqSync = HelperDVBS2TimeFreqSynchronizer( ...
%   'CarrSyncLoopBW',rxParams.carrSyncLoopBW, ...
%   'SymbSyncLoopBW',rxParams.symbSyncLoopBW, ...
%   'SamplesPerSymbol',simParams.sps, ...
%   'DataFrameSize',rxParams.xFecFrameSize, ...
%   'SymbSyncTransitFrames',rxParams.symbSyncLock, ...
%   'FrameSyncAveragingFrames',rxParams.frameSyncLock)
```

Replace the code in the symbol timing and coarse frequency synchronization section with these lines of code. Run the simulation for different carrier frequency offset (CFO) values. After coarse frequency compensation, view the plot and the residual CFO value (resCoarseCFO) over each frame to observe the performance of the coarse frequency estimation algorithm. Ideally, the coarse frequency compensation reduces the error to 2% of the symbol rate. If the coarse frequency compensation is not reduced to less than 3% of the symbol rate, try decreasing the loop bandwidth and increasing the rxParams.coarseFreqLock. Because the frequency error is estimated using the pilot symbols, verify that the frame synchronizer is properly locked to the beginning of frame.

```
% [out,index,phEst] = timeFreqSync(rxData,false);
%% Frequency offset estimate normalized by symbol rate
% freqOffEst = diff(phEst(1:simParams.sps:end))/(2*pi);
% plot(freqOffEst)
% actFreqOff = simParams.cfo/(simParams.chanBW/(1 + cfgDVBS2.RolloffFactor));
% resCoarseCFO = abs(actFreqOff-freqOffEst(end));
```

When the residual carrier frequency offset value (resCoarseCFO) is reduced to approximately 0.02 or 0.03, set the rxParams.frameCount to the rxParams.coarseFreqLock value.

For 'normal' FEC frames, set the rxParams.fineFreqLock value to 10. For 'short' FEC frames, set the rxParams.fineFreqLock value to 40. Replace the code in the fine frequency error estimation section with this code.

```
% rxParams.fineFreqCorrVal = HelperDVBS2FineFreqEst(fineFreqIn(rxParams.pilotInd), ...
%   rxParams.numPilots,rxParams.refPilots,rxParams.fineFreqCorrVal);
% fineFreqEst = angle(rxParams.fineFreqCorrVal)/(pi*(19));
% resFineCFO = abs(actFreqOff-freqOffEst(end)-fineFreqEst);
```

Repeat the simulation process and observe the residual CFO value (resFineCFO) over each frame. If the fine frequency estimator does not reduce the residual carrier frequency error to approximately 0.01% of the symbol rate, try increasing the rxParams.fineFreqLock property value.

When the residual CFO value (resFineCFO) is reduced to approximately 0.0001 or 0.0002, set the rxParams.frameCount+1 to the rxParams.fineFreqLock value.

Fine phase compensation PLL is used for only 16 APSK and 32 APSK modulation schemes with substantial phase noise.

After refining the synchronization parameters set in the rxParams structure, perform the BER simulation for the updated configuration.

Appendix

The example uses these helper functions:

- `HelperDVBS2RxInputGenerate.m`: Generate DVB-S2 waveform samples distorted with RF impairments and structure of parameters for receiver processing
- `HelperDVBS2PhaseNoise.m`: Generate phase noise samples for different DVB-S2 phase noise masks and apply it to the input signal
- `HelperDVBS2TimeFreqSynchronizer.m`: Perform matched filtering, symbol timing synchronization, frame synchronization, and coarse frequency estimation and correction
- `HelperDVBS2FrameSync.m`: Perform frame synchronization and detect the start of frame
- `HelperDVBS2FineFreqEst.m`: Estimate fine frequency offset
- `HelperDVBS2PhaseEst.m`: Estimate carrier phase offset
- `HelperDVBS2PhaseCompensate.m`: Perform carrier phase compensation
- `HelperDVBS2FinePhaseCompensator.m`: Perform fine carrier phase error tracking and compensation for APSK modulation schemes
- `HelperDVBS2PLHeaderRecover.m`: Demodulate and decode PL header to recover transmission parameters
- `HelperDVBS2NoiseVarEstimate.m`: Estimate noise variance of received data
- `HelperDVBS2StreamRecover.m`: Perform CRC check of BB header and recover input stream from BB frame based on header parameters

Bibliography

- 1 ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 2 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 3 Mengali, Umberto, and Aldo N.D'Andrea. *Synchronization Techniques for Digital Receivers*. New York: Plenum Press, 1997.
- 4 E. Casini, R. De Gaudenzi, and Alberto Ginesi. "DVB-S2 modem algorithms design and performance over typical satellite channels." *International Journal of Satellite Communications and Networking* 22, no. 3 (2004): 281-318.
- 5 Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

See Also

Objects

`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator`

Related Examples

- "End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames" on page 4-54
- "End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames" on page 4-69

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode” on page 4-87

End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames

This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) link that has constant coding and modulation for regular frames. The example describes the symbol timing and carrier synchronization strategies in detail emphasizing on how to estimate the RF front-end impairments under heavy noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

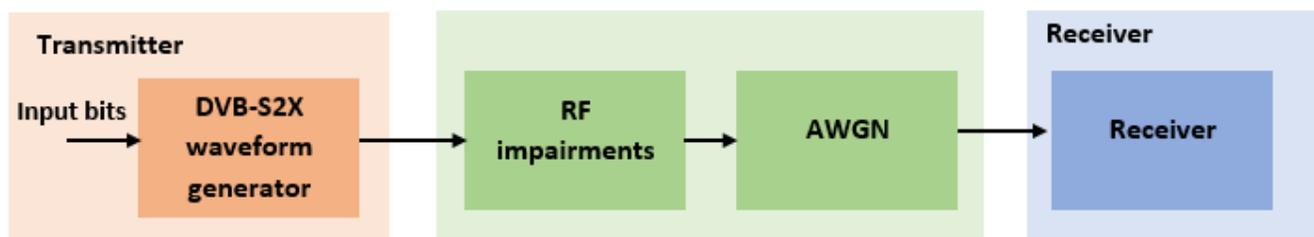
The DVB-S2X standard, an extension of the DVB-S2 specification, enhances the support provided for core DVB-S2 applications and improves overall efficiency over satellite links. The DVB-S2X standard supports these additional features:

- More granularity of modulation and code rates
- Smaller filter roll-off options for better utilization of bandwidth
- Constellations optimized for linear and nonlinear channels
- More scrambling options for critical co-channel interference scenarios

DVB-S2X caters to a variety of different target applications, and the receivers are subjected to different types and levels of RF impairments based on the application used. This example designs the synchronization aspects of a DVB-S2X receiver used for core DVB-S2 applications. The example supports the newer code rates, higher modulation schemes such as 64, 128 and 256 APSK, and smaller filter roll-off options.

ETSI EN 302 307-2 Section 6 Table 20a and Table 20c [1] on page 4-68 summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. The operating E_s/N_o range for different transmission modes can be considered as +3 or -2 dB from the E_s/N_o point where QEF performance is observed. Because the operating E_s/N_o range is low, carrier and symbol timing synchronization strategies are challenging design problems.

This diagram summarizes the example workflow.



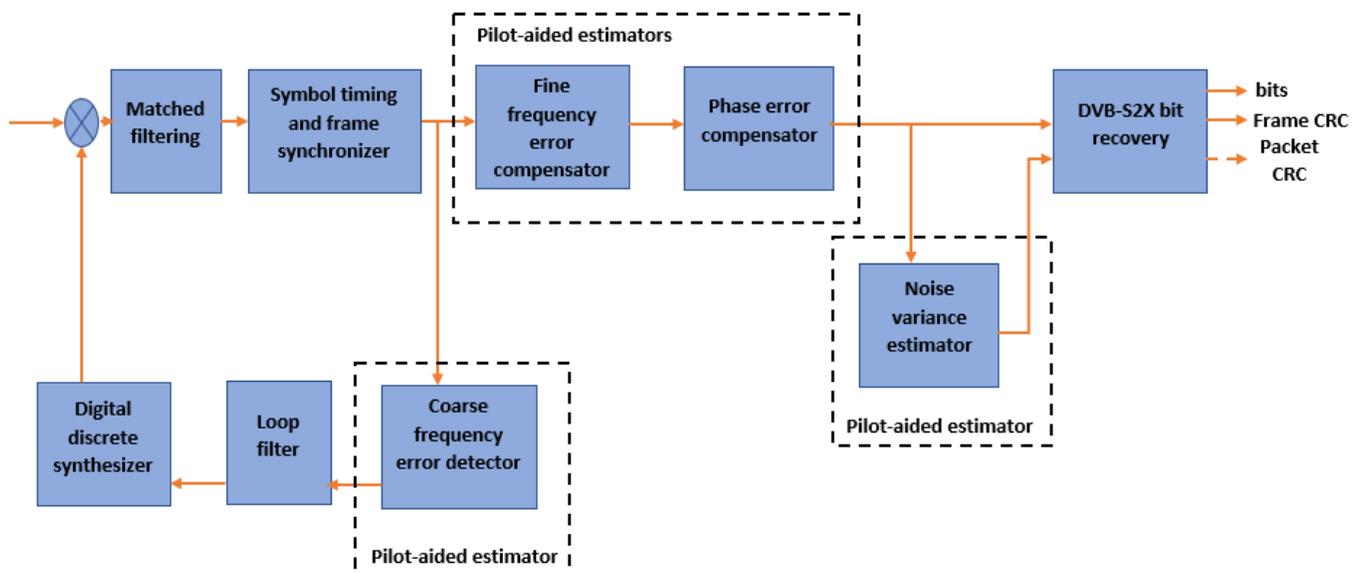
Main Processing Loop

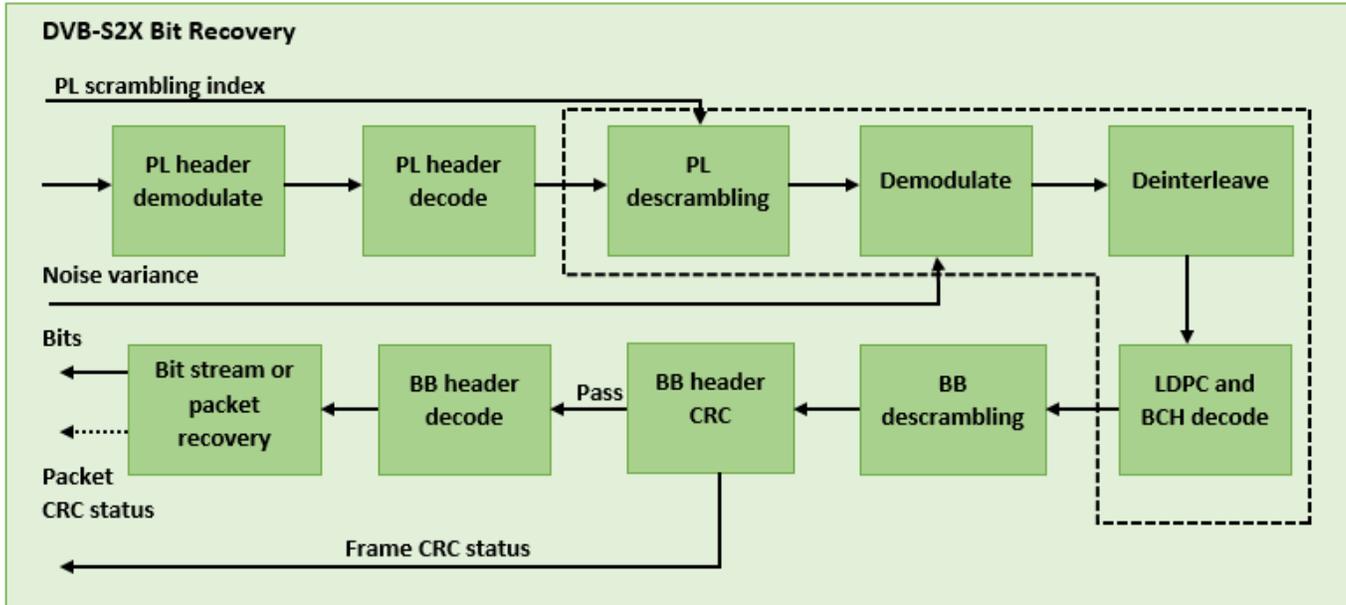
The example processes 30 physical layer (PL) frames of data with the E_s/N_o set to 25 dB, and then computes the BER and PER. Carrier frequency offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

At the receiver, after matched filtering, timing and carrier recovery operations are run to recover the transmitted data. To extract PL frames, the distorted waveform is processed through various timing and carrier recovery strategies. The carrier recovery algorithms are pilot-aided. To decode the data frames, the physical layer transmission parameters, such as modulation scheme, code rate, and FEC frame type, are recovered from the PL header. To regenerate the input bit stream, the baseband (BB) header is decoded.

Because the DVB-S2X standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured.

These block diagrams show the synchronization and input bit recovery workflows.





Download DVB-S2X LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```

if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
    addpath('s2xLDPCParityMatrices');
end
  
```

DVB-S2X Configuration in Pilot-Aided Mode

Specify the `cfgDVBS2X` structure to define DVB-S2X transmission configuration parameters. `PLSDecimalCode` 129 and 131 are not supported because they are used for generating VL-SNR frames. Only the regular frames are supported.

```

cfgDVBS2X.StreamFormat = "TS";
cfgDVBS2X.PLSDecimalCode = 191; % 64APSK 7/9 with pilots
cfgDVBS2X.DFL = 50128;
cfgDVBS2X.ScalingMethod = "Unit average power";
cfgDVBS2X.RolloffFactor = 0.35;
cfgDVBS2X.SamplesPerSymbol = 2

cfgDVBS2X = struct with fields:
    StreamFormat: "TS"
    PLSDecimalCode: 191
    DFL: 50128
    ScalingMethod: "Unit average power"
    RolloffFactor: 0.3500
    SamplesPerSymbol: 2
  
```

Simulation Parameters

The DVB-S2X standard supports flexible channel bandwidths. Use a typical channel bandwidth such as 36 MHz. The channel bandwidth can be varied. The coarse frequency synchronization algorithm implemented in this example can track carrier frequency offsets up to 11% of the input symbol rate. The symbol rate is calculated as $B/(1+R)$, where B is the channel bandwidth, and R is the transmit filter roll-off factor. The algorithms implemented in this example can correct the sampling clock offset up to 10 ppm.

```

simParams.sps = cfgDVBS2X.SamplesPerSymbol;           % Samples per symbol
simParams.numFrames = 30;                             % Number of frames to be processed
simParams.chanBW = 36e6;                              % Channel bandwidth in Hertz
simParams.cfo = 2e6;                                  % Carrier frequency offset in Hertz
simParams.sco = 2;                                    % Sampling clock offset in parts
                                                    % per million

simParams.phNoiseLevel = ; % Phase noise level provided as
                                                    % 'Low', 'Medium', or 'High'
simParams.EsNodB = 25;                               % Energy per symbol to noise ratio
                                                    % in decibels

```

This table defines the phase noise mask (dBc/Hz) used to generate the phase noise applied to the transmitted signal. These noise masks are taken from ETSI TR 102 376-1 Section 4.3.2.1.3 Figure 12 [2] on page 4-68.

Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz
Low	-73	-85	-93	-97	-110
Medium	-50	-60	-83	-105	-115
High	-25	-50	-73	-93	-103

Generate DVB-S2X Waveform Distorted with RF Impairments

To create a DVB-S2X waveform, use the `HelperDVBS2XRxInputGenerate` helper function with the `simParams` and `cfgDVBS2X` structures as inputs. The function returns the data signal, transmitted and received waveforms, physical layer configuration parameters as a structure, and a receiver processing structure. The received waveform is impaired with carrier frequency, timing phase offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

```
[data,txOut,rxIn,phyConfig,rxParams] = HelperDVBS2XRxInputGenerate(cfgDVBS2X,simParams);
disp(phyConfig)
```

```

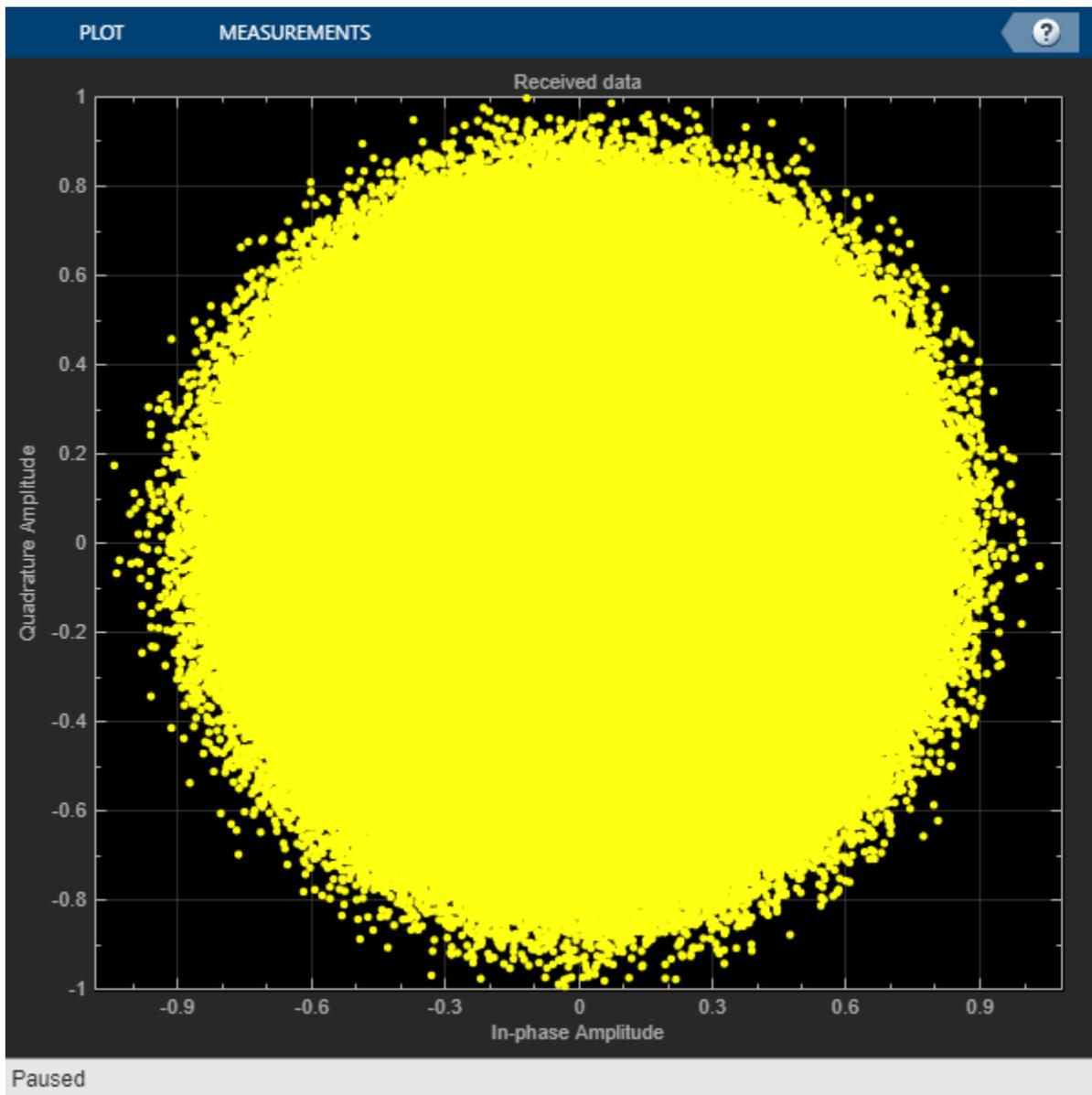
    FECFrame: "normal"
  ModulationScheme: "64APSK"
  LDPCCodeIdentifier: "7/9"

```

```

% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:length(txOut)))

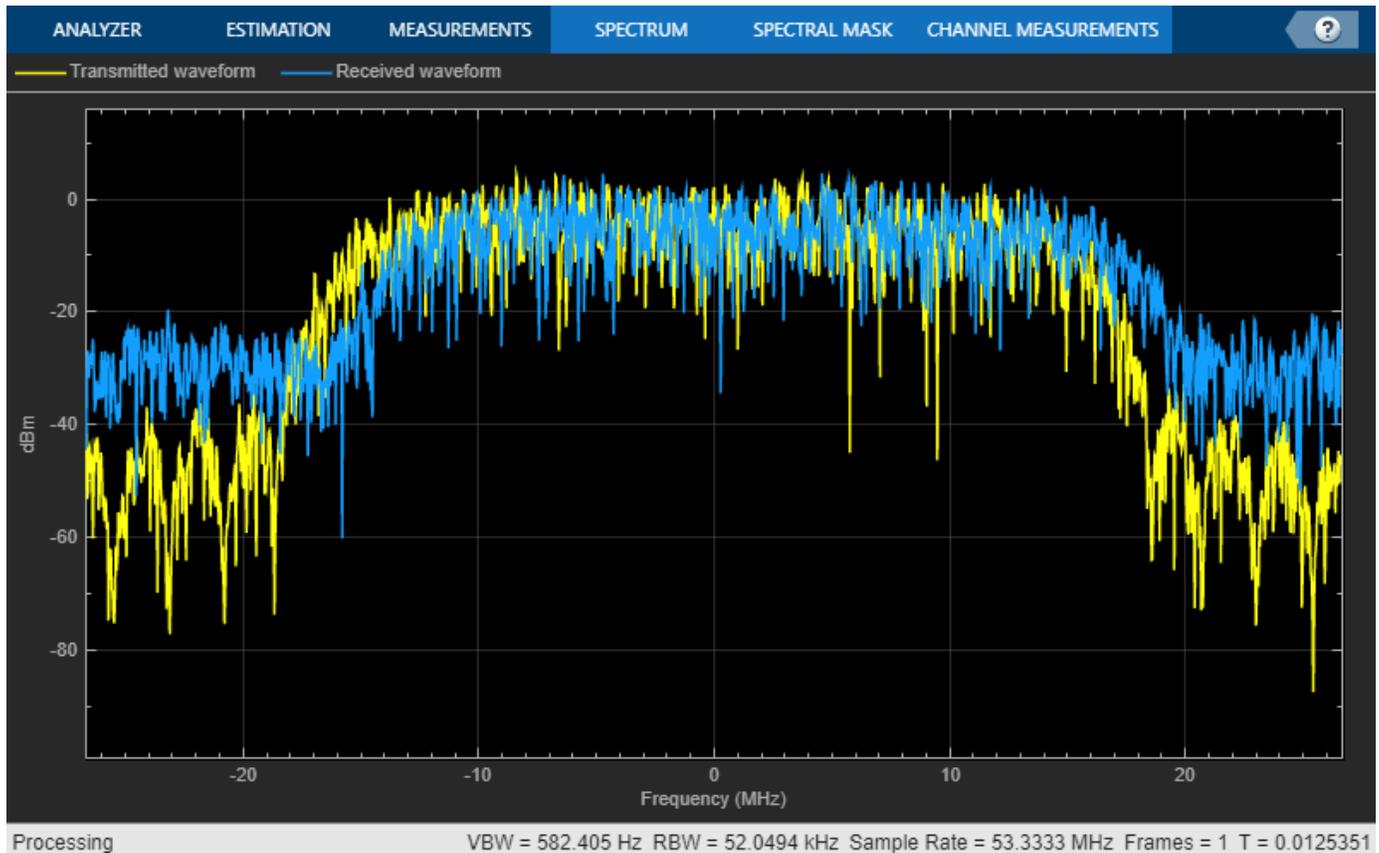
```



```

% Transmitted and received signal spectrum visualization
RsymB = simParams.chanBW/(1 + cfgDVBS2X.RolloffFactor);
Fsamp = Rsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'ShowLegend',true);
specAn([txOut,rxIn(1:length(txOut))]);

```



Configure Receiver Parameters

At the receiver, symbol timing synchronization is performed on the received data and is then followed by frame synchronization. The receiver algorithms include coarse and fine frequency impairment correction algorithms. The carrier frequency estimation algorithm can track carrier frequency offsets up to 20% of the input symbol rate. The coarse frequency estimation, implemented as a frequency locked loop (FLL), reduces the frequency offset to a level that the fine frequency estimator can track. The preferred loop bandwidth for symbol timing and coarse frequency compensation depends on the E_s/N_o setting.

A block of 36 pilots is repeated every 1476 symbols. The coarse frequency error estimation uses 34 of the 36 pilot symbols. The ratio of used pilots per block (34) and pilot periodicity(1476) is 0.023. Using the 0.023 value as a scaling factor for the coarse frequency synchronizer loop bandwidth is preferred.

When you decrease the E_s/N_o , reduce the loop bandwidth to filter out more noise during acquisition. The number of frames required for the symbol synchronizer and coarse FLL to converge depends on the loop bandwidth setting.

The frame synchronization uses the PL header. Because the carrier synchronization is data-aided, the frame synchronization must detect the start of frame accurately. E_s/N_o plays a crucial role in determining the accuracy of the frame synchronization. When QPSK modulated frames are being recovered at E_s/N_o values below 3 dB, the frame synchronization must be performed on multiple frames for accurate detection.

The fine frequency estimation can track carrier frequency offsets up to 4% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency error less than 0.02% of the input symbol rate.

These settings are assigned in the `rxParams` structure for synchronization processing. For details on how to set these parameters for low E_s/N_o values, see the Further Exploration on page 4-67 section.

```
rxParams.carrSyncLoopBW = 1e-2*0.023;           % Coarse frequency estimator loop bandwidth
                                                % normalized by symbol rate
rxParams.symbSyncLoopBW = 8e-3;                 % Symbol timing synchronizer loop bandwidth
                                                % normalized by symbol rate
rxParams.symbSyncLock   = 8;                     % Number of frames required for symbol timing
                                                % error convergence
rxParams.frameSyncLock = 1;                     % Number of frames required for frame
                                                % synchronization
rxParams.coarseFreqLock = 5;                    % Number of frames required for coarse
                                                % frequency acquisition
rxParams.fineFreqLock   = 4;                    % Number of frames required for fine
                                                % frequency estimation

% Total frames taken for symbol timing and coarse frequency lock to happen
rxParams.initialTimeFreqSync = rxParams.symbSyncLock + rxParams.frameSyncLock + ...
    rxParams.coarseFreqLock;
% Total frames used for overall synchronization
rxParams.totalSyncFrames = rxParams.initialTimeFreqSync + rxParams.fineFreqLock;

% Create time frequency synchronization System object by using
% HelperDVBS2TimeFreqSynchronizer helper object
timeFreqSync = HelperDVBS2TimeFreqSynchronizer( ...
    'CarrSyncLoopBW',rxParams.carrSyncLoopBW, ...
    'SymbSyncLoopBW',rxParams.symbSyncLoopBW, ...
    'SamplesPerSymbol',simParams.sps, ...
    'DataFrameSize',rxParams.xFecFrameSize, ...
    'SymbSyncTransitFrames',rxParams.symbSyncLock, ...
    'FrameSyncAveragingFrames',rxParams.frameSyncLock);

% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

% Initialize data indexing variables
stIdx = 0;
dataSize = rxParams.inputFrameSize;
plFrameSize = rxParams.plFrameSize;
dataStInd = rxParams.totalSyncFrames + 1;
isLastFrame = false;
symSyncOutLen = zeros(rxParams.initialTimeFreqSync,1);
```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data and recover the input bit stream, the distorted DVB-S2X waveform samples are processed one frame at a time by following these steps.

- 1 Apply matched filtering, outputting at the rate of two samples per symbol.
- 2 Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate. The Gardner TED is not data-aided, so it is performed before carrier synchronization.

- 3 Apply frame synchronization to detect the start of frame and to identify the pilot positions.
- 4 Estimate and apply coarse frequency offset correction.
- 5 Estimate and apply fine frequency offset correction.
- 6 Estimate and compensate for residual carrier frequency and phase noise.
- 7 Decode the PL header and compute the transmission parameters.
- 8 Demodulate and decode the PL frames.
- 9 Perform CRC check on the BB header, if the check passes, recover the header parameters.
- 10 Regenerate the input stream of data or packets from BB frames.

```

while stIdx < length(rxIn)

    % Use one DVB-S2X PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are considered.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % After coarse frequency offset loop is converged, the FLL works with a
    % reduced loop bandwidth.
    if rxParams.frameCount < rxParams.initialTimeFreqSync
        coarseFreqLock = false;
    else
        coarseFreqLock = true;
    end

    % Retrieve the last frame samples.
    if isLastFrame
        resSymb = plFrameSize - length(rxParams.cfBuffer);
        resSampCnt = resSymb*rxParams.sps - length(rxData);
        if resSampCnt >= 0 % Inadequate number of samples to fill last frame
            syncIn = [rxData; zeros(resSampCnt, 1)];
        else % Excess samples are available to fill last frame
            syncIn = rxData(1:resSymb*rxParams.sps);
        end
    else
        syncIn = rxData;
    end

    % Apply matched filtering, symbol timing synchronization, frame
    % synchronization, and coarse frequency offset compensation.
    [coarseFreqSyncOut, syncIndex, phEst] = timeFreqSync(syncIn, coarseFreqLock);
    if rxParams.frameCount <= rxParams.initialTimeFreqSync
        symSyncOutLen(rxParams.frameCount) = length(coarseFreqSyncOut);
        if any(abs(diff(symSyncOutLen(1:rxParams.frameCount)))) > 5
            error(['Symbol timing synchronization failed. The loop will not ' ...
                'converge. No frame will be recovered. Update the symbSyncLoopBW ' ...
                'parameter according to the EsNo setting for proper loop convergence.']);
        end
    end

    rxParams.syncIndex = syncIndex;

```

```

% The PL frame start index lies somewhere in the middle of the chunk being processed.
% From fine frequency estimation onwards, the processing happens as a PL frame.
% A buffer is used to store symbols required to fill one PL frame.
if isLastFrame
    resCnt = resSymb - length(coarseFreqSyncOut);
    if resCnt <= 0
        fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut(1:resSymb)];
    else
        fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut; zeros(resCnt, 1)];
    end
elseif rxParams.frameCount > 1
    fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut(1:rxParams.plFrameSize-length(rxParams
end

% Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
% helper function.
% Add 1 to the conditional check because the buffer used to get one PL
% frame introduces a delay of one to the loop count.
if (rxParams.frameCount > rxParams.initialTimeFreqSync + 1) && ...
    (rxParams.frameCount <= rxParams.totalSyncFrames + 1)
    rxParams.fineFreqCorrVal = HelperDVBS2FineFreqEst( ...
        fineFreqIn(rxParams.pilotInd), rxParams.numPilotBlks, ...
        rxParams.refPilots, rxParams.fineFreqCorrVal);
end
if rxParams.frameCount >= rxParams.totalSyncFrames + 1
    fineFreqLock = true;
else
    fineFreqLock = false;
end

if fineFreqLock
    % Normalize the frequency estimate by the input symbol rate
    % freqEst = angle(R)/(pi*(N+1)) where N (18) is the number of elements
    % used to compute the mean of auto correlation (R) in
    % HelperDVBS2FineFreqEst.
    freqEst = angle(rxParams.fineFreqCorrVal)/(pi*(19));

    % Generate the symbol indices using frameCount and plFrameSize.
    % Subtract 2 from the rxParams.frameCount because the buffer used to get one
    % PL frame introduces a delay of one to the count.
    ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
    phErr = exp(-1j*2*pi*freqEst*ind);
    fineFreqOut = fineFreqIn.*phErr(:);

    % Estimate the phase error estimation by using the HelperDVBS2PhaseEst
    % helper function.
    [phEstRes, rxParams.prevPhaseEst] = HelperDVBS2PhaseEst( ...
        fineFreqOut(rxParams.pilotInd), rxParams.refPilots, rxParams.prevPhaseEst);

    % Compensate for the residual frequency and phase offset by using
    % the
    % HelperDVBS2PhaseCompensate helper function.
    % Use two frames for initial phase error estimation. Starting with the
    % second frame, use the phase error estimates from the previous frame and
    % the current frame in compensation.
    % Add 3 to the frame count comparison to account for delays: One
    % frame due to rxParams.cfBuffer delay and two frames used for phase

```

```

% error estimate.
if rxParams.frameCount >= rxParams.totalSyncFrames + 3
    phaseCompOut = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
        rxParams.pilotEst,rxParams.pilotInd,phEstRes(2));
end

rxParams.ffBuffer = fineFreqOut;
rxParams.pilotEst = phEstRes;

% The phase compensation on the data portion is performed by
% interpolating the phase estimates computed on consecutive pilot
% blocks. The second phase estimate is not available for the data
% portion after the last pilot block in the last frame. Therefore,
% the slope of phase estimates computed on all pilot blocks in the
% last frame is extrapolated and used to compensate for the phase
% error on the final data portion.
if isLastFrame
    pilotBlkLen = 36; % Symbols
    pilotBlkFreq = 1476; % Symbols
    avgSlope = mean(diff(phEstRes(2:end)));
    chunkLen = rxParams.plFrameSize - rxParams.pilotInd(end) + ...
        rxParams.pilotInd(pilotBlkLen);
    estEndPh = phEstRes(end) + avgSlope*chunkLen/pilotBlkFreq;
    phaseCompOut1 = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
        rxParams.pilotEst,rxParams.pilotInd,estEndPh);
end
end

% Recover the input bit stream.
if rxParams.frameCount >= rxParams.totalSyncFrames + 3
    isValid = true;
    if isLastFrame
        syncOut = [phaseCompOut;phaseCompOut1];
    else
        syncOut = phaseCompOut;
    end
else
    isValid = false;
    syncOut = [];
end

% Update the buffers and counters.
rxParams.cfBuffer = coarseFreqSyncOut(rxParams.syncIndex:end);
rxParams.syncIndex = syncIndex;
rxParams.frameCount = rxParams.frameCount + 1;

if isValid % Data valid signal

    % Decode the PL header by using the HelperDVBS2XPLHeaderRecover helper
    % function. Start of frame (SOF) is 26 symbols which are discarded
    % before header decoding. They are only required for frame
    % synchronization.
    rxPLSCode = syncOut(1:90); % First 90 symbols of frame is PL header
    [plsDecCode,phyParams] = HelperDVBS2XPLHeaderRecover(rxPLSCode,rxParams.s2xStatus);
    % Validate the decoded PL header.
    if plsDecCode ~= cfgDVBS2X.PLSDecimalCode
        fprintf('%s\n','PL header decoding failed')
        dataStInd = dataStInd + 1;
    end
end

```

```

else % Demodulation and decoding
    for frameCnt = 1:length(syncOut)/rxParams.plFrameSize
        rxFrame = syncOut((frameCnt-1)*plFrameSize+1:frameCnt*plFrameSize);
        % Estimate noise variance by using
        % HelperDVBS2NoiseVarEstimate helper function.
        nVar = HelperDVBS2NoiseVarEstimate(rxFrame,rxParams.pilotInd,...
            rxParams.refPilots,rxParams.normFlag);
        % The data begins at symbol 91 (after the header symbols).
        rxDataFrame = rxFrame(91:end);
        % Recover the BB frame by using HelperDVBS2XBBFrameRecover
        % helper function.
        rxBBFrame = HelperDVBS2XBBFrameRecover(rxDataFrame,phyParams,...
            rxParams.plScramblingIndex,rxParams.numPilotBlks,nVar,false);
        % Recover the input bit stream by using
        % HelperDVBS2StreamRecover helper function.
        if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
            [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame);
            if ~isFrameLost && length(decBits) ~= dataSize
                isFrameLost = true;
            end
        else
            [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame);
            if ~isFrameLost && length(decBits) ~= dataSize
                isFrameLost = true;
                pktCRC = zeros(0,1,'logical');
            end
            % Compute the packet error rate for TS or GS packetized
            % mode.
            pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
            pktsRec = pktsRec + numel(pktCRC);
        end
        if ~isFrameLost
            ts = sprintf('%s','BB header decoding passed.');
```

```

        else
            ts = sprintf('%s','BB header decoding failed.');
```

```

        end
        % Compute the number of frames lost. CRC failure of
        % baseband header is considered a frame loss.
        numFramesLost = isFrameLost + numFramesLost;
        fprintf('%s(Number of frames lost = %ld)\n',ts,numFramesLost)
        % Compute the bits in error.
        bitInd = (dataStInd-1)*dataSize+1:dataStInd*dataSize;
        if isLastFrame && ~isFrameLost
            bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
        else
            if ~isFrameLost
                bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
            end
        end
        dataStInd = dataStInd + 1;
    end
end
end
stIdx = endIdx;
end

BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)

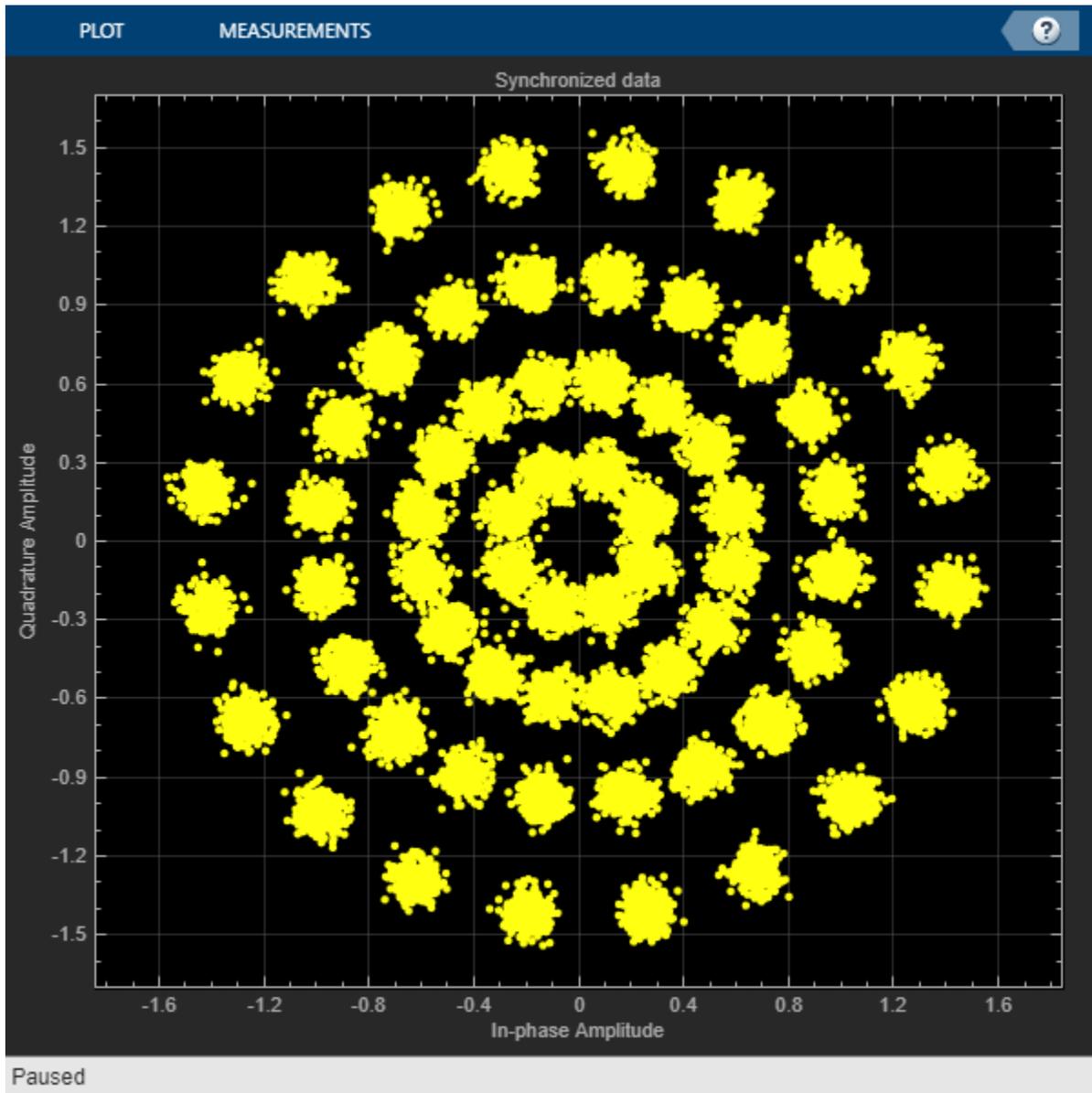
```

```
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
```

Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```
% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title','Synchronized data', ...
    'XLimits',[-1.7 1.7],'YLimits',[-1.7 1.7], ...
    'ShowReferenceConstellation',false);
syncConst(syncOut)
```



```

% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames-rxParams.totalSyncFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        ber = bitsErr/((dataStInd-rxParams.totalSyncFrames)*dataSize);
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
    else

```

```

if strcmpi(cfgDVBS2X.StreamFormat, 'TS')
    pktLen = 1504;
else
    pktLen = cfgDVBS2X.UPL;      % UP length including sync byte
end
ber = bitsErr/(pktsRec*pktLen);
per = pktsErr/pktsRec;
fprintf('PER: %1.2e\n',per)
fprintf('BER: %1.2e\n',ber)
end
end

```

PER: 0.00e+00

BER: 0.00e+00

Further Exploration

For BER simulations in AWGN assuming perfect synchronization, use the `HelperDVBS2XBitRecover` helper function to evaluate the receiver performance. See the examples provided in the M-help section of the `HelperDVBS2XBitRecover` helper function. For details on how to configure the synchronization parameters of the `rxParams` for other `cfgDVBS2X` and `simParams` settings, see the 'Further Exploration section' of "End-to-End DVB-S2 Simulation with RF Impairments and Corrections" on page 4-37 on how to configure the synchronization parameters of `rxParams` for other `cfgDVBS2X` and `simParams` settings. For higher modulation schemes like 64 APSK and above, this table shows the typical number of frames required for convergence of the symbol timing loop.

Modulation scheme	Number of frames
64 APSK	30 - 35
128 APSK	35 - 40
256 APSK	38 - 43

Appendix

The example uses these helper functions:

- `HelperDVBS2XRxInputGenerate.m`: Generate DVB-S2X waveform samples distorted with RF impairments and structure of parameters for receiver processing
- `HelperDVBS2PhaseNoise.m`: Generate phase noise samples for different DVB-S2X phase noise masks and apply it to the input signal
- `HelperDVBS2TimeFreqSynchronizer.m`: Perform matched filtering, symbol timing synchronization, frame synchronization, and coarse frequency estimation and correction
- `HelperDVBS2FrameSync.m`: Perform frame synchronization and detect the start of frame
- `HelperDVBS2FineFreqEst.m`: Estimate fine frequency offset
- `HelperDVBS2PhaseEst.m`: Estimate carrier phase offset
- `HelperDVBS2PhaseCompensate.m`: Perform carrier phase compensation
- `HelperDVBS2XPLHeaderRecover.m`: Demodulate and decode the PL header to recover transmission parameters

- HelperDVBS2NoiseVarEstimate.m: Estimate noise variance of received data
- HelperDVBS2XBBFrameRecover.m: Perform PL de-scrambling, demodulation, decoding and recover BB frame from PL frame
- HelperDVBS2XDemapper.m: Perform soft demodulation for all DVB-S2X based modulation schemes
- HelperDVBS2XLDPCDecode.m: Perform LDPC decoding for all DVB-S2X based LDPC frame formats and code rates
- HelperDVBS2XBCHDecode.m: Perform BCH decoding for all DVB-S2X based frame formats and code rates
- HelperDVBS2StreamRecover.m: Perform CRC check of BB header and recover input stream from BB frame based on header parameters
- HelperDVBS2XBitRecover.m: Perform PL header demodulation and decoding, PL de-scrambling, demodulation, decoding and recover BB frame. Perform CRC check of BB header and recover the input stream from BB frame.

Bibliography

- 1 ETSI Standard EN 302 307-2 V1.1.1(2015-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 extensions (DVB-S2X)*.
- 2 ETSI Standard TR 102 376-2 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: S2 extensions (DVB-S2X)*.
- 3 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 4 Mengali, Umberto, and Aldo N.D'Andrea. *Synchronization Techniques for Digital Receivers*. New York: Plenum Press,1997.
- 5 E. Casini, R. De Gaudenzi, and Alberto Ginesi. "DVB-S2 modem algorithms design and performance over typical satellite channels." *International Journal of Satellite Communications and Networking* 22, no. 3 (2004): 281-318.
- 6 Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

See Also

Objects

dvbs2xWaveformGenerator | dvbs2WaveformGenerator

Related Examples

- "End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames" on page 4-69
- "End-to-End DVB-S2 Simulation with RF Impairments and Corrections" on page 4-37
- "End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode" on page 4-87

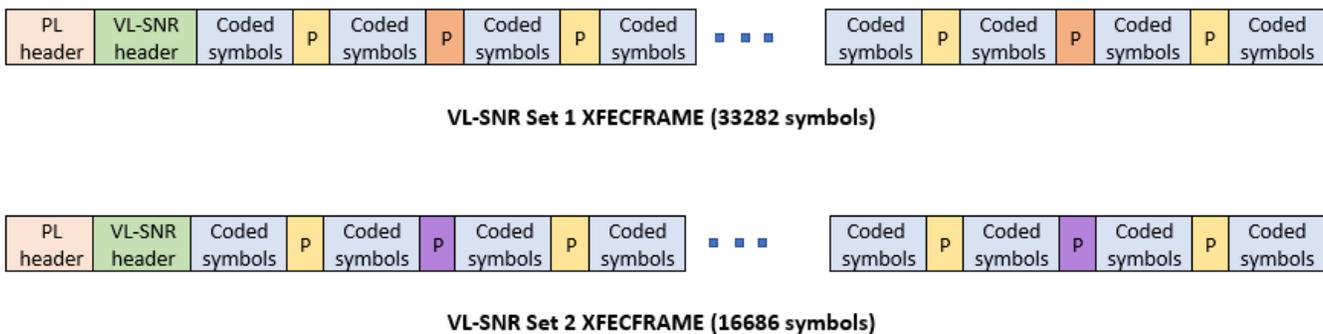
End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames

This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) link that has constant coding and modulation for very low signal to noise ratio (VL-SNR) frames. The example describes the symbol timing, frame and carrier synchronization strategies in detail emphasizing on how to estimate the RF front-end impairments under severe noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

An increasing number of DVB-S2X terminals are being used on trains, buses, boats, and airplanes. Many applications, such as sensor networks, remote infrastructure monitoring, and emergency services, require ubiquitous coverage, and low data rates. Support for these applications was therefore included in the design of DVB-S2X, introducing VL-SNR configurations to increase the operating range to cover current and emerging applications that can benefit from operation at very low SNR. The DVB-S2X standard added nine additional modulation and coding schemes (MODCODs) in the QPSK and BPSK range. These MODCODs enable the satellite networks to deal with heavy atmospheric fading and to enable use of smaller antennas for applications in motion (land, sea, and air).

These figures show the two formats used for VL-SNR frames.

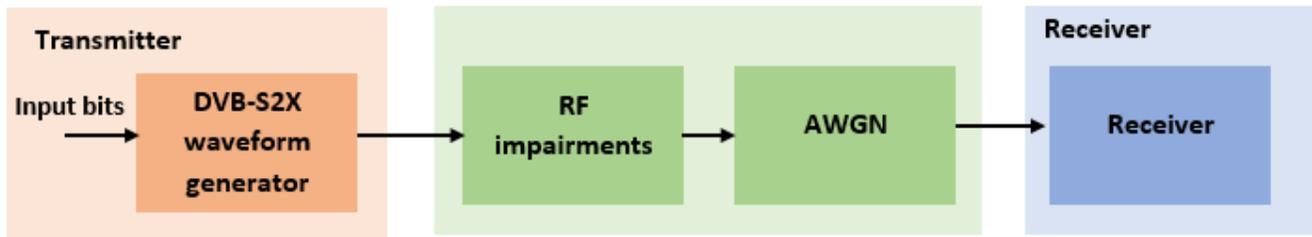


- Regular 36 block pilot symbols
- VL-SNR set 1 pilot symbols
- VL-SNR set 2 pilot symbols

This example designs the synchronization aspects of a DVB-S2X receiver used for VL-SNR applications. The example supports all the nine MODCODs defined by the standard.

ETSI EN 302 307-2 Section 6 Table 20a, Table 20b, and Table 20c [1] on page 4-84 summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. The operating E_s/N_o range for VL-SNR applications is considered from -2 dB to -10 dB. Because the operating E_s/N_o range is low, the carrier, frame, and symbol timing synchronization strategies are challenging design problems.

This diagram summarizes the example workflow.



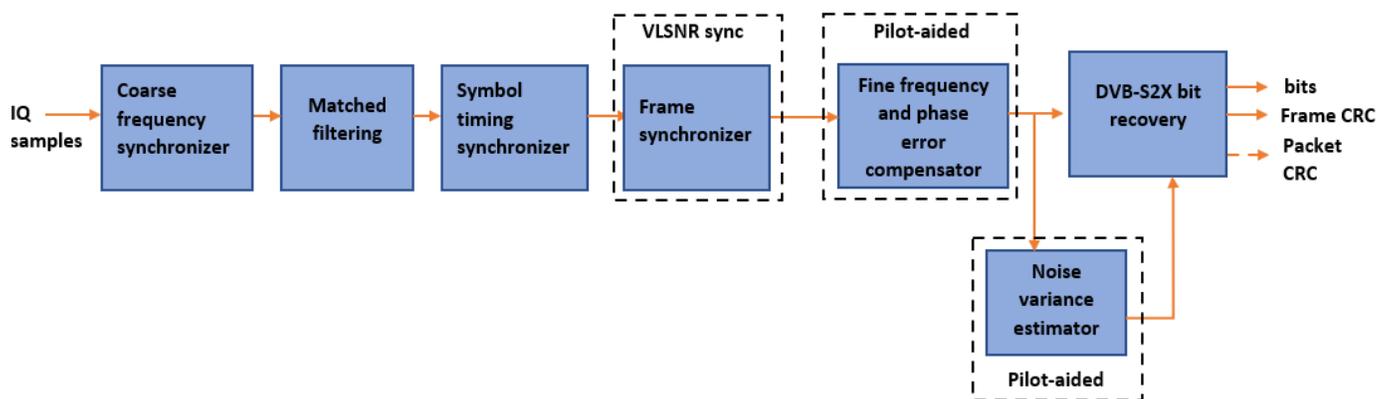
Main Processing Loop

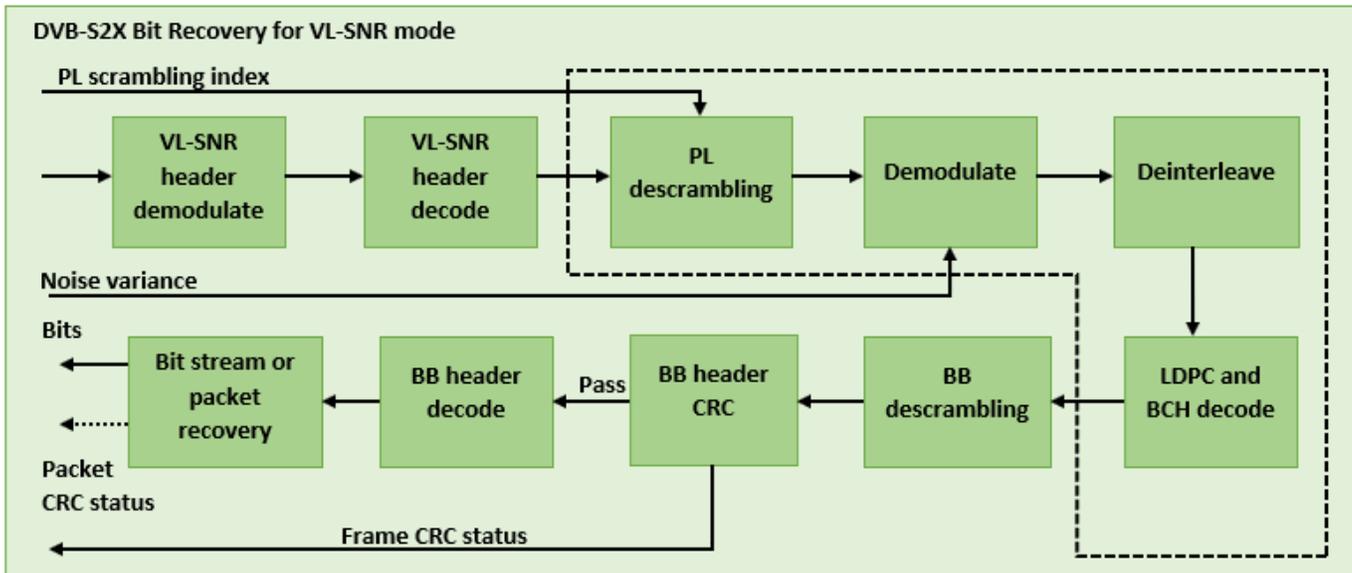
The example processes 20 physical layer (PL) frames of data with the E_s/N_o set to 5 dB, and then computes the BER and PER. Carrier frequency offset, frequency drift, symbol timing offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal. ETSI EN 302 307-2 Section 4.4.4 describes the typical RF impairment ranges used under VL-SNR conditions.

To extract PL frames, the receiver processes the distorted waveform through various timing and carrier recovery strategies. The fine frequency and carrier phase recovery algorithms are pilot-aided. To decode the data frames, the physical layer transmission parameters, such as VL-SNR set type, MODCOD, and FEC frame type are recovered from the VL-SNR header. To regenerate the input bit stream, the baseband (BB) header is decoded.

Because the DVB-S2X standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured.

These block diagrams show the synchronization and input bit recovery workflows.





Download DVB-S2X LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```

if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
    addpath('s2xLDPCParityMatrices');
end
end
  
```

DVB-S2X Configuration

Specify the `cfgDVBS2X` structure to define DVB-S2X transmission configuration parameters. PLSDecimalCode 129 and 131 are the only supported formats because they are used for generating VL-SNR frames.

```

cfgDVBS2X.StreamFormat = "TS";
cfgDVBS2X.PLSDecimalCode = 129;
cfgDVBS2X.CanonicalMODCODName = "QPSK 2/9";
cfgDVBS2X.DFL = 14128;
cfgDVBS2X.RolloffFactor = 0.35;
cfgDVBS2X.SamplesPerSymbol = 2
  
```

```

cfgDVBS2X = struct with fields:
    StreamFormat: "TS"
    PLSDecimalCode: 129
    CanonicalMODCODName: "QPSK 2/9"
    DFL: 14128
    RolloffFactor: 0.3500
    SamplesPerSymbol: 2
  
```

Simulation Parameters

The DVB-S2X standard supports flexible channel bandwidths. Use a typical channel bandwidth such as 36 MHz. The channel bandwidth can be varied from 10MHz to 72MHz. The coarse frequency synchronization algorithm implemented in this example can track carrier frequency offsets up to 20% of the input symbol rate. The symbol rate is calculated as $B/(1+R)$, where B is the channel bandwidth, and R is the transmit filter roll-off factor. The algorithms implemented in this example can correct the sampling clock offset up to 10 ppm.

This table defines the phase noise mask (dBc/Hz) used to generate the phase noise that is applied to the transmitted signal. These noise masks are specified in ETSI TR 102 376-1 Section 4.3 [2] on page 4-84. The peak doppler and frequency drift supported are specified in ETSI TR 102 376-1 Section 4.4.4.

Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz
Level-1	-61.9	-78.7	-88.7	-94.8	-105.7
Level-2	-58.1	-68.5	-78.3	-88.3	-88.3
Level-3	-45	-65	-75	-85	-102

```

simParams.sps = cfgDVBS2X.SamplesPerSymbol; % Samples per symbol
simParams.numFrames = 20; % Number of frames to be processed
simParams.chanBW = 36e6; % Channel bandwidth in Hertz
simParams.cfo = 3e6; % Carrier frequency offset in Hertz due
% to oscillator instabilities
simParams.dopplerRate = 3e3; % Doppler rate in Hertz/sec
simParams.peakDoppler = 20e3; % Peak doppler shift due to receiver motion
simParams.sco = 10; % Sampling clock offset in parts per
% million
simParams.phNoiseLevel = "Level-1"; % Phase noise level
simParams.EsNodB = 5; % Energy per symbol to noise ratio

```

Generate DVB-S2X VL-SNR Waveform Distorted with RF Impairments

To create a DVB-S2X waveform, use the `HelperDVBS2XVLSNRRxInputGenerate` helper function with the `simParams` and `cfgDVBS2X` structures as inputs. The function returns the data signal, transmitted, and received waveforms, physical layer configuration parameters as a structure, and a receiver processing structure. The received waveform is impaired with carrier frequency, frequency drift, symbol timing, sampling clock offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

```

[data,txOut,rxIn,phyConfig,rxParams] = ...
    HelperDVBS2XVLSNRRxInputGenerate(cfgDVBS2X,simParams);
disp(phyConfig)

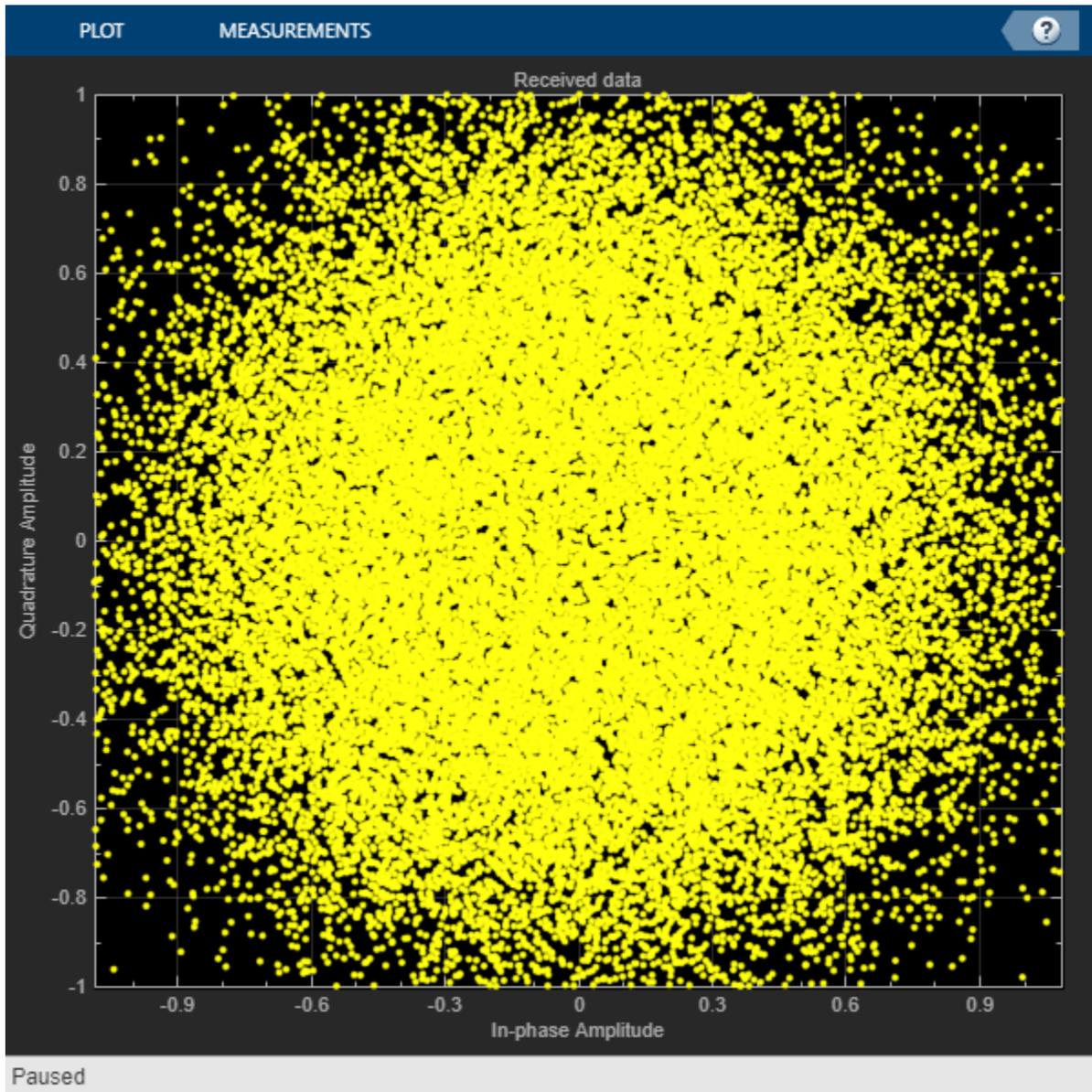
    FECFrame: "normal"
    ModulationScheme: "QPSK"
    LDPCCodeIdentifier: "2/9"

```

```

% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:rxParams.plFrameSize*simParams.sps))

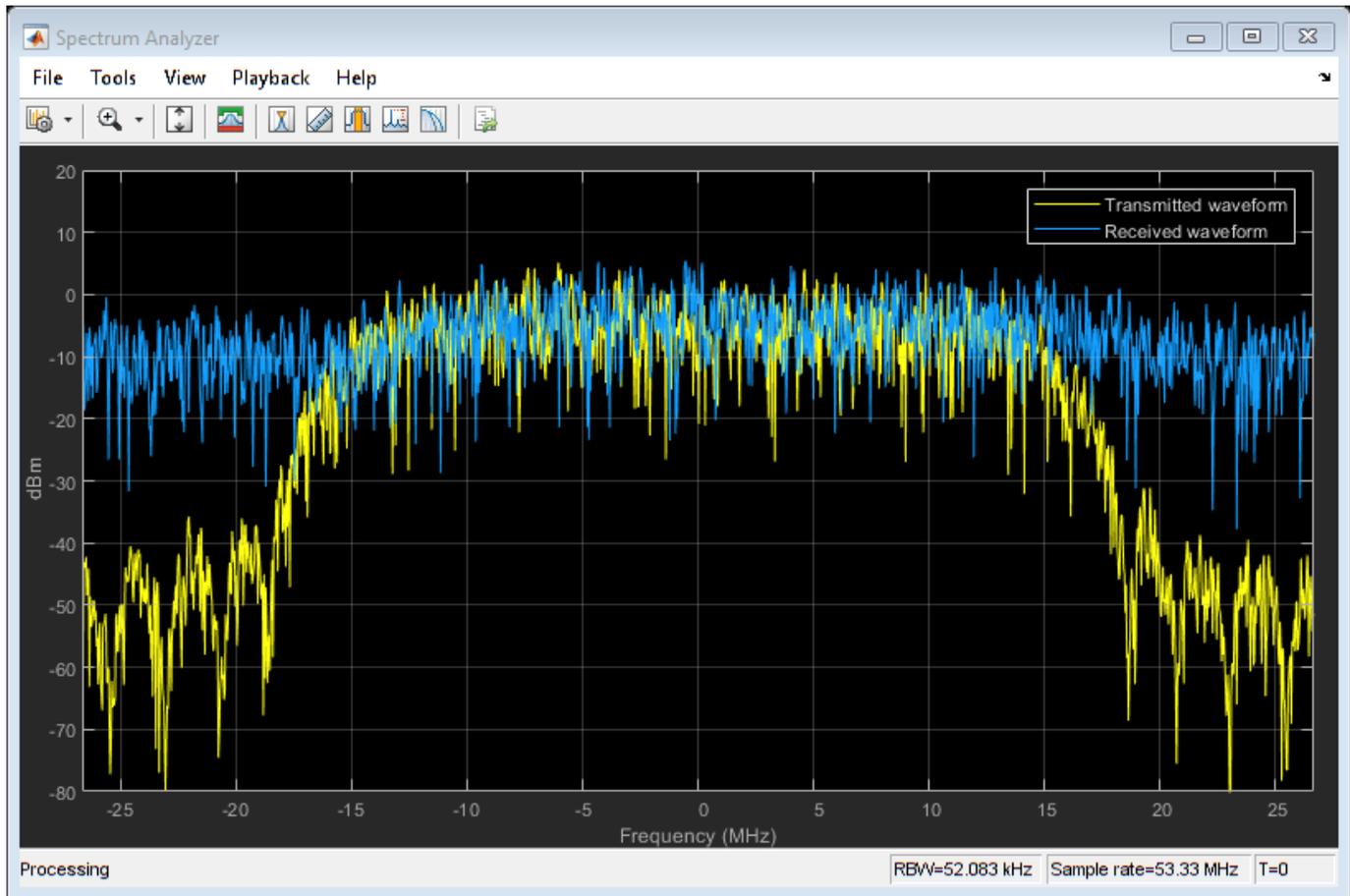
```



```

% Transmitted and received signal spectrum visualization
Rsymb = simParams.chanBW/(1 + cfgDVBS2X.RolloffFactor);
Fsamp = Rsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'ShowLegend',true);
specAn([txOut,rxIn(1:length(txOut))]);

```



Configure Receiver Parameters

At the receiver, coarse frequency synchronization is performed on the received data and is then followed by matched filtering and symbol timing synchronization. The coarse frequency and symbol timing estimators are non-data aided. The coarse frequency estimation algorithm can track carrier frequency offsets up to 20% of the input symbol rate. The coarse frequency estimation, implemented as a first order frequency locked loop (FLL), reduces the frequency offset to a level that the fine frequency estimator can track. The preferred loop bandwidth for symbol timing and coarse frequency compensation depends on the E_s/N_o setting.

When you decrease the E_s/N_o , reduce the loop bandwidth to filter out more noise during acquisition. The number of frames required for the symbol synchronizer and coarse FLL to converge depends on the loop bandwidth setting.

Symbol timing synchronization is followed by frame synchronization and MODCOD format detection. The frame synchronization uses the VL-SNR header. Because the fine frequency and carrier phase synchronization are data-aided, the frame synchronization must detect the start of frame accurately.

The fine frequency estimation can track carrier frequency offsets up to 3.5% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency error less than 0.05% of the input symbol rate.

These settings are assigned in the rxParams structure for synchronization processing. For details on how to set these parameters for low E_s/N_o values, see the Further Exploration on page 4-82 section.

```

rxParams.carrSyncLoopBW = 1e-4;           % Coarse frequency estimator loop bandwidth
                                           % normalized by symbol rate
rxParams.symbSyncLoopBW = 1e-4;         % Symbol timing synchronizer loop bandwidth
                                           % normalized by symbol rate
rxParams.initialTimeFreqSync = 5;       % Number of frames required for coarse frequency
                                           % and symbol timing error convergence
rxParams.fineFreqLock = 4;              % Number of frames required for fine
                                           % frequency estimation
rxParams.NeedSmoothing = false;         % Smoothen the phase estimate

% Total frames used for overall synchronization
rxParams.totalSyncFrames = rxParams.initialTimeFreqSync + rxParams.fineFreqLock;

% Create coarse frequency synchronization System object by using
% HelperDVBS2XVLSNRCoarseFreqSynchronizer helper object
freqSync = HelperDVBS2XVLSNRCoarseFreqSynchronizer('SamplesPerSymbol',simParams.sps, ...
    'NormalizedLoopBandwidth',rxParams.carrSyncLoopBW);

% Create symbol timing synchronization System object by using
% comm.SymbolSynchronizer object
symSync = comm.SymbolSynchronizer('TimingErrorDetector','Gardner (non-data-aided)', ...
    'NormalizedLoopBandwidth',rxParams.symbSyncLoopBW);

% Create matched filter System object by using
% comm.RaisedCosineReceiveFilter object
if simParams.sps == 2
    decFac = 1;
else
    decFac = simParams.sps/(simParams.sps/2);
end
rxFilter = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor',0.35, ...
    'InputSamplesPerSymbol',simParams.sps,'DecimationFactor',decFac);
b = rxFilter.coeffs;
rxFilter.Gain = sum(b.Numerator);

% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

% Initialize data indexing variables
stIdx = 0;
dataSize = rxParams.inputFrameSize;
plFrameSize = rxParams.plFrameSize;
isLastFrame = false;
rxParams.fineFreqCorrVal = zeros(rxParams.fineFreqLock,1);
[formatIdx,formatIdxTemp] = deal(1);
vlsnrSyncStIdx = 93;
payloadStIdx = 899;
vLSNRFrameLen = plFrameSize - vlsnrSyncStIdx + 1;

```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data and recover the input bit stream, process the distorted DVB-S2X waveform samples one frame at a time by following these steps.

- 1 Apply coarse frequency synchronization using a balanced quadrice correlator frequency error detector (BQ-FED) in an FLL [5] on page 4-84.
- 2 Apply matched filtering, outputting at the rate of two samples per symbol.
- 3 Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate.
- 4 Apply frame synchronization to detect the start of frame and MODCOD format to identify the pilot positions.
- 5 Estimate and apply fine frequency offset correction.
- 6 Estimate and compensate for residual carrier frequency and phase noise.
- 7 Demodulate and decode the VL-SNR frames.
- 8 Perform CRC check on the BB header, if the check passes, recover the header parameters.
- 9 Regenerate the input stream of data or packets from BB frames.

```

while stIdx < length(rxIn)

    % Use one DVB-S2X PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are considered.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % After coarse frequency offset loop is converged, the FLL works with
    % previous frequency estimate.
    if rxParams.frameCount < rxParams.initialTimeFreqSync
        coarseFreqLock = false;
    else
        coarseFreqLock = true;
    end

    % Retrieve the last frame samples.
    if isLastFrame
        resSampCnt = plFrameSize*rxParams.sps - length(rxData);
        % Inadequate number of samples to fill last frame
        syncIn = [rxData; zeros(resSampCnt,1)];
    else
        syncIn = rxData;
    end

    % Apply coarse frequency offset compensation.
    [coarseFreqSyncOut,phEst] = freqSync(syncIn,coarseFreqLock);

    % Perform matched filtering and downsample the signal to 2 samples per
    % symbol.
    filtOut = rxFilter(coarseFreqSyncOut);

    % Apply symbol timing synchronization.
    symSyncOut = symSync(filtOut);

    % Apply frame synchronization and identify the MODCOD format. VL-SNR
    % sync frame is detected. PL header preceding the VL-SNR header is
    % ignored.

```

```

if rxParams.frameCount > rxParams.initialTimeFreqSync && ~isLastFrame
    [~,rxParams.syncIndex,formatIdx] = ...
        HelperDVBS2XVLSNRFrameSync(symSyncOut,rxParams.SegLength);
    % MODCOD format identification failure verification
    formatFail = formatIdxTemp ~= rxParams.refFormat;
    if formatFail && ~isLastFrame
        % Update the counters, state variables, and buffers
        stIdx = endIdx;
        rxParams.frameCount = rxParams.frameCount + 1;
        formatIdxTemp = formatIdx;
        rxParams.cfBuffer = symSyncOut(rxParams.syncIndex:end);
        fprintf('%s\n','MODCOD format detection failed')
        continue;
    else
        fprintf('%s\n','MODCOD format detection passed')
        [setNum,phyParams] = getVLSNRParams(formatIdxTemp);
    end
end

% The PL frame start index lies somewhere in the middle of the data
% being processed. From fine frequency estimation onwards, the
% processing happens as a PL frame. A buffer is used to store symbols
% required to fill one PL frame. PL frame is considered from
% the start of VL-SNR header, precisely from the start of the 896 bit
% length Walsh Hadamard (WH) sequence. 90 represents the PL header
% length, and 2 accounts for the two zeros appended before the WH sequence.
fineFreqIn = [rxParams.cfBuffer;...
    symSyncOut(1:vLSNRFrameLen-length(rxParams.cfBuffer))];

% Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
% helper function.
% Add 1 to the conditional check because the buffer used to get one PL
% frame introduces a delay of one to the loop count.
if (rxParams.frameCount > rxParams.initialTimeFreqSync + 1)
    % Extract the payload by removing header
    payload = fineFreqIn(payloadStIdx:end);

    % Get the correlation estimate from the regular 36 length pilot blocks.
    est1 = HelperDVBS2FineFreqEst( ...
        payload(rxParams.regPilotInd),rxParams.regNumPilotBlks, ...
        rxParams.regPilotSeq,rxParams.fineFreqState,36,rxParams.NumLags);

    % Get the correlation estimate from the VL-SNR extra pilot blocks.
    % vLSNRPilotBlk1Params contains the VL-SNR type 1 pilot block
    % length and number of blocks in one frame.
    Lp = rxParams.vLSNRPilotBlk1Params(1);
    est2 = HelperDVBS2FineFreqEst( ...
        payload(rxParams.vLSNRPilotInd1),rxParams.vLSNRPilotBlk1Params(2), ...
        rxParams.vLSNRPilotSeq1,rxParams.fineFreqState,Lp,rxParams.NumLags);

    % vLSNRPilotBlk2Params contains the VL-SNR type 2 pilot block
    % length and number of blocks in one frame.
    Lp = rxParams.vLSNRPilotBlk2Params(1);
    est3 = HelperDVBS2FineFreqEst( ...
        payload(rxParams.vLSNRPilotInd2),rxParams.vLSNRPilotBlk2Params(2), ...
        rxParams.vLSNRPilotSeq2,rxParams.fineFreqState,Lp,rxParams.NumLags);

    estVal = est1 + est2 + est3;

```

```

% Use the correlation values calculated over pilot fields spanning over
% multiple frames to calculate the fine frequency error estimate.
% The estimation uses a sliding window technique.
rxParams.fineFreqCorrVal = [rxParams.fineFreqCorrVal(2:end);estVal];
end

if rxParams.frameCount >= rxParams.totalSyncFrames
    fineFreqLock = true;
else
    fineFreqLock = false;
end

if fineFreqLock
    freqEst = angle(sum(rxParams.fineFreqCorrVal))/(pi*(rxParams.NumLags+1));
    ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
    phErr = exp(-1j*2*pi*freqEst*ind).';
    % 92 accounts for the PL header (90), and 2 for zeros appended before the WH
    % sequence.
    fineFreqOut = fineFreqIn(1:vLSNRFrameLen).*phErr(vlsnrSyncStIdx:end);
    % 898 accounts for the 896 length WH sequence, and 2 zeros padded to the
    % header.
    rxPilots = fineFreqOut(rxParams.pilotInd+payloadStIdx-1);
    phErrEst = HelperDVBS2PhaseEst(rxPilots,rxParams.pilotSeq, ...
        rxParams.phErrState,rxParams.IsVLSNR,setNum,rxParams.Alpha);
    if rxParams.NeedSmoothing
        phErrEst = smoothenEstimate(phErrEst);
    end
    phaseCompOut = HelperDVBS2PhaseCompensate(fineFreqOut(payloadStIdx:end), ...
        phErrEst,rxParams.pilotInd,setNum,rxParams.IsVLSNR);
end

% Recover the input bit stream.
if rxParams.frameCount >= rxParams.totalSyncFrames
    isValid = true;
    syncOut = phaseCompOut;
else
    isValid = false;
    syncOut = [];
end

% Update the buffers and counters.
rxParams.cfBuffer = symSyncOut(rxParams.syncIndex:end);

if isValid % Data valid signal
    % Estimate noise variance by using
    % HelperDVBS2NoiseVarEstimate helper function.
    nVar = HelperDVBS2NoiseVarEstimate(syncOut,rxParams.pilotInd, ...
        rxParams.pilotSeq,false);
    % Recover the BB frame by using HelperDVBS2XBBFrameRecover
    % helper function.
    rxBBFrame = HelperDVBS2XBBFrameRecover(syncOut,phyParams, ...
        rxParams.plScramblingIndex,rxParams.regNumPilotBlks,nVar,true,setNum);
    % Recover the input bit stream by using
    % HelperDVBS2StreamRecover helper function.
    if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
        [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame);
        if ~isFrameLost && length(decBits) ~= dataSize
            isFrameLost = true;
        end
    end
end

```

```

    end
else
    [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame);
    if ~isFrameLost && length(decBits) ~= dataSize
        isFrameLost = true;
        pktCRC = zeros(0,1,'logical');
    end
    % Compute the PER for TS or GS packetized
    % mode.
    pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
    pktsRec = pktsRec + numel(pktCRC);
end
if ~isFrameLost
    ts = sprintf('%s','BB header decoding passed.');
```

```

else
    ts = sprintf('%s','BB header decoding failed.');
```

```

end
% Compute the number of frames lost. CRC failure of the
% baseband header is considered a frame loss.
numFramesLost = isFrameLost + numFramesLost;
fprintf('%s(Number of frames lost = %ld)\n',ts,numFramesLost)
% Compute the bits in error.
if ~isFrameLost
    dataInd = (rxParams.frameCount-2)*dataSize+1:(rxParams.frameCount-1)*dataSize;
    errs = sum(data(dataInd) ~= decBits);
    bitsErr = bitsErr + errs;
end
end

stIdx = endIdx;
rxParams.frameCount = rxParams.frameCount + 1;
formatIdxTemp = formatIdx;
end

MODCOD format detection passed
MODCOD format detection passed
MODCOD format detection passed
MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

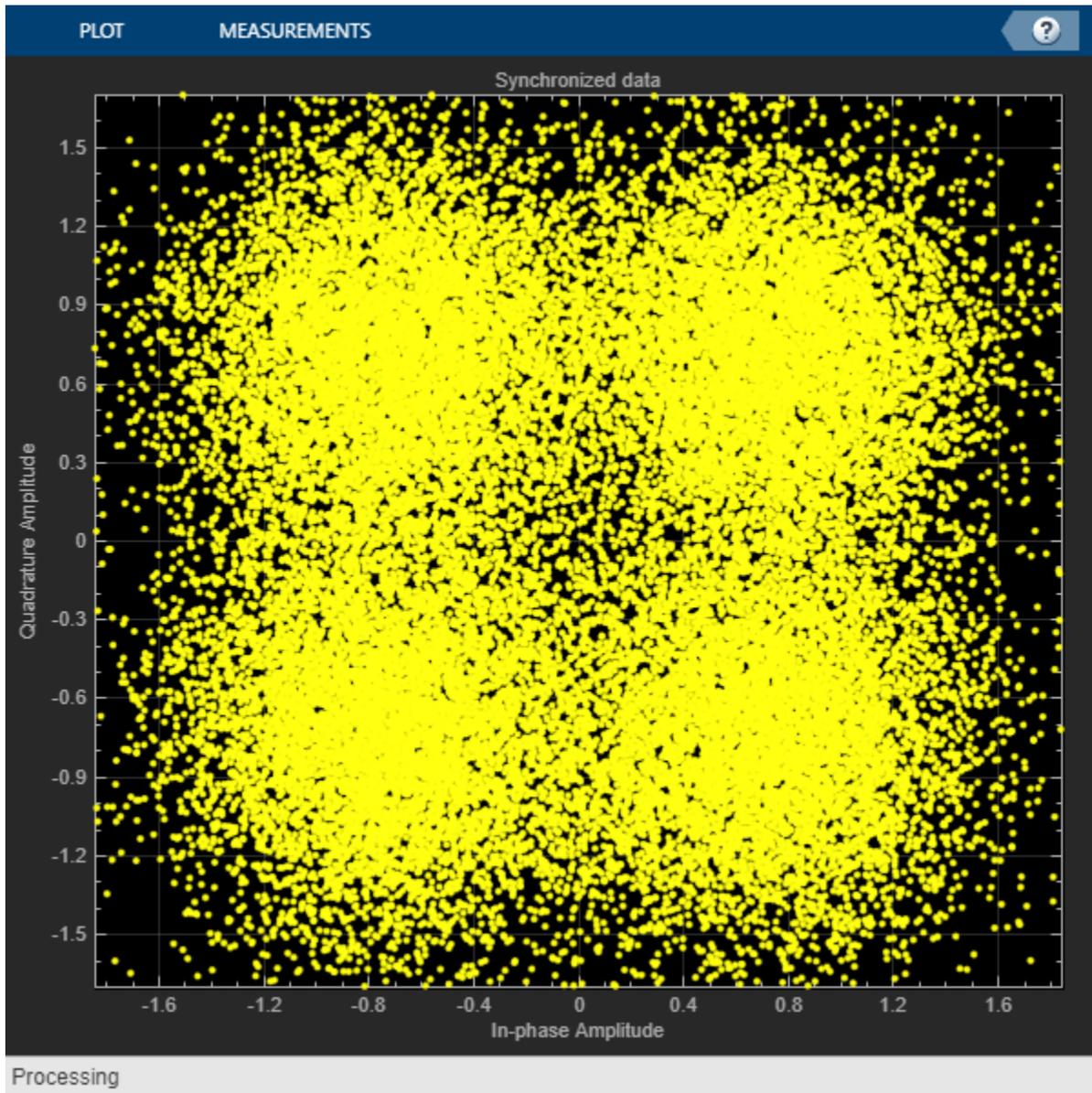
```

```
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
MODCOD format detection passed
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
```

Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```
% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title','Synchronized data', ...
    'XLimits',[-1.7 1.7],'YLimits',[-1.7 1.7], ...
    'ShowReferenceConstellation',false);
syncConst(syncOut)
```



```

pause(0.5)
% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames-rxParams.totalSyncFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        numFramesRec = simParams.numFrames - rxParams.totalSyncFrames - numFramesLost;
        ber = bitsErr/(numFramesRec*dataSize);
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
    end
end

```

```

else
    if strcmpi(cfgDVBS2X.StreamFormat, 'TS')
        pktLen = 1504;
    else
        pktLen = cfgDVBS2X.UPL;           % UP length including sync byte
    end
    ber = bitsErr/(pktsRec*pktLen);
    per = pktsErr/pktsRec;
    fprintf('PER: %1.2e\n',per)
    fprintf('BER: %1.2e\n',ber)
end
end

```

```
PER: 0.00e+00
```

```
BER: 0.00e+00
```

Further Exploration

The operating E_s/N_o range of the VL-SNR mode being very low requires the normalized loop bandwidth of the symbol synchronizer and coarse FLL to be very small for accurate estimation. Set these parameters using the `rxParams.symbSyncLoopBW` and `rxParams.carrSyncLoopBW`.

Configure Coarse Carrier Synchronization Parameters

Initialize `HelperDVBS2XVLSNRCoarseFreqSynchronizer` System object with `rxParams.carrSyncLoopBW` set as $2e-5$ and then run the simulation.

When you set the `PLSDecimalCode` property to 129, set the `rxParams.initialTimeFreqSync` property to 15. When you set the `PLSDecimalCode` property to 131, set the `rxParams.coarseFreqLock` property to 30. Set `simParams.EsNodB` to the lowest E_s/N_o for the chosen modulation scheme from ETSI EN 302 307-1 Section 6 [1] on page 4-84.

Replace the code in the coarse frequency synchronization section with these lines of code. Run the simulation for different carrier frequency offset (CFO) values. After coarse frequency compensation, view the plot and the residual CFO value (`resCoarseCFO`) over each frame to observe the performance of the coarse frequency estimation algorithm. Ideally, the coarse frequency compensation reduces the error to 2% of the symbol rate. If the residual CFO error is not reduced to less than 2% of the symbol rate, try decreasing the loop bandwidth and increasing the `rxParams.initialTimeFreqSync`. The coarse frequency is a first order FLL, and it can only detect the static carrier frequency offset. It cannot track Doppler rate changes.

```

% [coarseFreqSyncOut,phEst] = freqSync(syncIn,false);
% Frequency offset estimate normalized by symbol rate
% freqOffEst = diff(phEst(1:simParams.sps:end))/(2*pi);
% plot(freqOffEst)
% Rsym = simParams.chanBW/(1+cfgDVBS2X.RolloffFactor);
% actFreqOff = (simParams.cfo + simParams.peakDoppler)/Rsym;
% resCoarseCFO = abs(actFreqOff-freqOffEst(end));

```

When the residual carrier frequency offset value (`resCoarseCFO`) is reduced to approximately 0.02, set the `rxParams.frameCount` as the `rxParams.initialTimeFreqSync` value.

Configure Symbol Timing Synchronization Parameters

Try running the simulation using the symbol timing synchronizer configured with a normalized loop bandwidth of $1e-4$. To achieve convergence of the timing loop, the ratio

`rxParams.symbSyncLoopBW/simParams.sps` must be greater than $1e-5$. If the symbol timing loop doesn't converge, try decreasing the `rxParams.symbSyncLoopBW`.

Configure Frame Synchronization Parameters

Proper frame synchronization depends on the segment length, `rxParams.SegLength`, which is used to divide the reference VL-SNR header symbols into smaller segments to perform segment coherent correlation. `rxParams.SegLength` depends on the residual CFO that is present after coarse frequency synchronization and must be less than $\text{round}(3/(8*\text{resCoarseCFO}))$. Prefer using a value that is an integer multiple of 896 (length of VL-SNR WH sequence). If CFO is absent, perform correlation using `rxParams.SegLength` as 896.

Configure Fine Frequency Synchronization Parameters

When you set the `PLSDecimalCode` property to 129, set the `rxParams.fineFreqLock` property to 10. When you set the `PLSDecimalCode` property to 131, set the `rxParams.coarseFreqLock` property to 20. Set `simParams.EsNodB` to the lowest E_s/N_o for the chosen modulation scheme from ETSI EN 302 307-1 Section 6 [1] on page 4-84.

Replace the code in the fine frequency error estimation section with this code. Fine frequency estimator tracks the Doppler rate changes. To estimate the residual CFO error, include the sinusoidal variation of the Doppler shift included in the `actFreqOff` estimate. For an easy workaround, do not introduce Doppler rate in impairment. Instead, add only static CFO and analyze the number of frames required to generate an accurate estimate. Typically, those number of frames are sufficient to estimate CFO changes due to Doppler rate. As the Doppler rate changes are handled by the fine frequency estimator, $2*\text{simParams.peakDoppler}$ must be less than the estimation range of the fine frequency estimator. The estimation range depends on the `rxParams.NumLags` parameter and the normalized CFO that can be estimated is given by $1/(\text{rxParams.NumLags}+1)$. To increase the estimation range, try reducing the `rxParams.NumLags`. You might need more pilot blocks because the estimation accuracy drops.

```
% fineFreqEst = angle(sum(rxParams.fineFreqCorrVal))/(pi*(rxParams.NumLags+1));
% resFineCFO = abs(actFreqOff-freqOffEst(end)-fineFreqEst);
```

Repeat the simulation process and observe the residual CFO value (`resFineCFO`) over each frame. If the fine frequency estimator does not reduce the residual carrier frequency error to approximately 0.03% of the symbol rate, try increasing the `rxParams.fineFreqLock` property value.

When the residual CFO value (`resFineCFO`) is reduced to approximately 0.0003, update `rxParams.totalSyncFrames` based on `rxParams.initialTimeFreqSyncrxParams` and `rxParams.fineFreqLock` values.

Configure Phase Synchronization Parameters

`HelperDVBS2PhaseEstimate` estimates the residual CFO and phase error appropriately up to -8 dB of E_s/N_o . For E_s/N_o less than -8 dB, set `rxParams.Alpha` to less than 0.5 if data that passed through `HelperDVBS2PhaseEstimate` has only phase error. If the data has residual CFO along with phase error, keep `rxParams.Alpha` as 1 and set `rxParams.NeedSmoothing` as true. Smoothing improves the `phaseEst` obtained from `HelperDVBS2PhaseEstimate`.

After refining the synchronization parameters set in the `rxParams` structure, perform the BER simulation for the updated configuration.

Appendix

The example uses these helper functions:

- HelperDVBS2XVLSNRRxInputGenerate.m: Generate DVB-S2X waveform samples distorted with RF impairments and structure of parameters for receiver processing
- HelperDVBS2PhaseNoise.m: Generate phase noise samples for different DVB-S2X phase noise masks and apply it to the input signal
- HelperDopplerShift.m: Generate sinusoidal varying Doppler shift and apply it to input signal
- HelperDVBS2XVLSNRCoarseFreqSynchronizer.m: Perform coarse frequency offset estimation and correction
- HelperDVBS2XVLSNRFrameSync.m: Perform frame synchronization and detect start of VL-SNR header and MODCOD identification
- HelperDVBS2FineFreqEst.m: Estimate fine frequency offset
- HelperDVBS2PhaseEst.m: Estimate carrier phase offset
- HelperDVBS2PhaseCompensate.m: Perform carrier phase compensation
- HelperDVBS2XPLHeaderRecover.m: Demodulate and decode PL header to recover transmission parameters
- HelperDVBS2NoiseVarEstimate.m: Estimate noise variance of received data
- HelperDVBS2XBBFrameRecover.m: Perform PL descrambling, demodulation, decoding and recover BB frame from PL frame
- HelperDVBS2XDemapper.m: Perform soft demodulation for all DVB-S2X based modulation schemes
- HelperDVBS2XLDPCDecode.m: Perform LDPC decoding for all DVB-S2X based LDPC frame formats and code rates
- HelperDVBS2XBCHDecode.m: Perform BCH decoding for all DVB-S2X based frame formats and code rates
- HelperDVBS2StreamRecover.m: Perform CRC check of BB header and recover input stream from BB frame based on header parameters
- HelperDVBS2XBitRecover.m: Perform PL header demodulation and decoding, PL de-scrambling, demodulation, decoding and recover BB frame. Perform CRC check of BB header, and recover input stream from BB frame.

Bibliography

- 1 ETSI Standard EN 302 307-2 V1.1.1(2015-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 extensions (DVB-S2X)*.
- 2 ETSI Standard TR 102 376-2 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: S2 extensions (DVB-S2X)*.
- 3 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

- 4 Mengali, Umberto, and Aldo N.D'Andrea. *Synchronization Techniques for Digital Receivers*. New York: Plenum Press,1997.
- 5 D'Andrea, A. N., and U. Mengali. "Design of Quadricorrelators for Automatic Frequency Control Systems." *IEEE Transactions on Communications*, vol. 41, no. 6, June 1993, pp. 988-97.
- 6 Casini, E., et al. "DVB-S2 Modem Algorithms Design and Performance over Typical Satellite Channels." *International Journal of Satellite Communications and Networking*, vol. 22, no. 3, May 2004, pp. 281-318.
- 7 Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

Local Functions

```
function [setNum,phyParams] = getVLSNRParams(formatIdx)

tableVLSNR = [4 64800 2/9 0; ...
    2 32400 1/5 0; ...
    2 32400 11/45 0; ...
    2 32400 1/3 0; ...
    2 16200 1/5 1; ...
    2 16200 11/45 1; ...
    2 16200 1/5 0; ...
    2 16200 4/15 0; ...
    2 16200 1/3 0];

params = tableVLSNR(formatIdx,:);
phyParams.ModOrder = params(1);
phyParams.FECFrameLen = params(2);
phyParams.CodeRate = params(3);
[n, d] = rat(phyParams.CodeRate);
phyParams.CodeIDF = [sprintf('%0.0f',n) '/' sprintf('%0.0f',d)];
phyParams.HasPilots = true;
phyParams.HasSpread = params(4);

if formatIdx > 6
    setNum = 2;
else
    setNum = 1;
end
end
% Smoothen the phase estimate using moving average filter
function newEst = smoothenEstimate(est)
errDiff = diff(est(2:end));
thres = -2*sign(mean(errDiff));
width = 5;
if thres > 0
    index = find(errDiff > thres);
else
    index = find(errDiff < thres);
end
if ~isempty(index)
    for k = 1:length(index)
        est(index(k)+2:end) = est(index(k)+2:end) - sign(thres)*2*pi;
    end
end
temp = est(2:end);
n = length(temp);
c = filter(ones(width,1)/width,1,temp);
```

```
cbegin = cumsum(temp(1:width-2));
cbegin = cbegin(1:2:end)./(1:2:(width-2))';
cend = cumsum(temp(n:-1:n-width+3));
cend = cend(end:-2:1)./(width-2:-2:1)';
c = [cbegin;c(width:end);cend];
newEst = [est(1);c];
end
```

See Also

Objects

dvbs2xWaveformGenerator | dvbs2WaveformGenerator

Related Examples

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames” on page 4-54
- “End-to-End DVB-S2 Simulation with RF Impairments and Corrections” on page 4-37
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode” on page 4-87

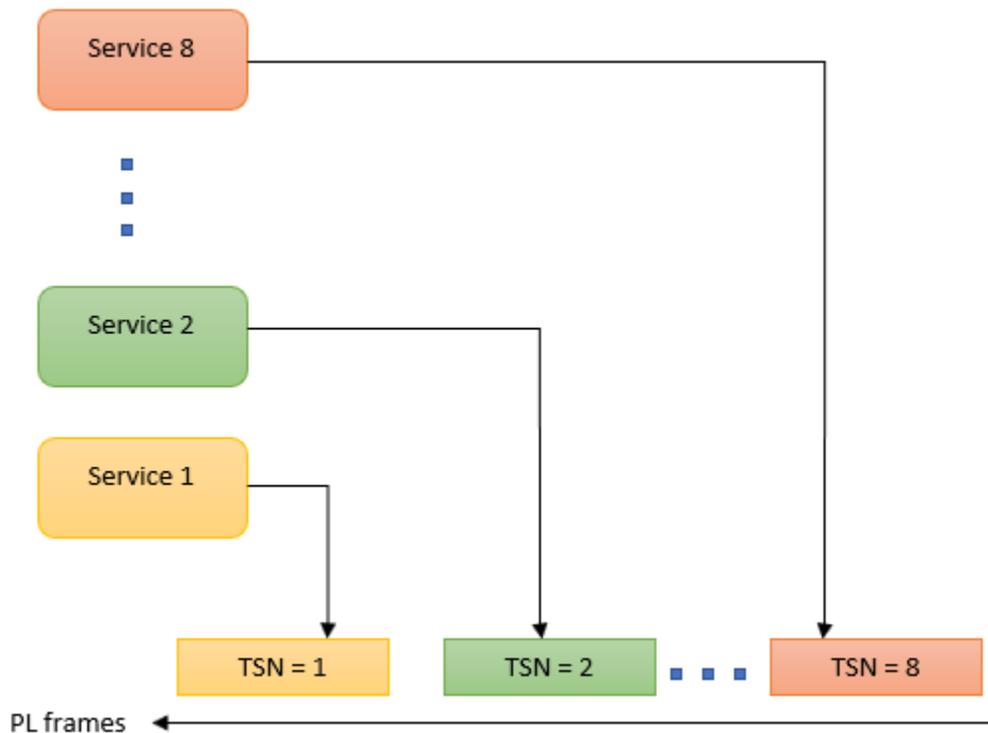
End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode

This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) link that has constant coding and modulation for wideband mode using time slicing. The example describes the symbol timing, frame, and carrier synchronization strategies in detail, emphasizing on how to correct the RF front-end impairments under severe noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

Wideband mode is an optional transmission format in the DVB-S2/S2X standard for high symbol-rate satellite carriers for broadcasting, professional, and interactive services. This format adopts wideband satellite transponder (200–500 MHz) on a single frequency and uses the time slicing concept defined in the DVB-S2/S2X standard (ETSI EN 302 307-2 Annex M) [1 on page 4-100].

Multiple services are time sliced at the transmitter in the form of physical layer (PL) frames, which are transmitted serially in time. Each service is provided as an independent input stream to the `dvbs2xWaveformGenerator` System object® and the maximum number of streams possible is 8. Each service is identified by a time slicing number (TSN) transmitted as part of the PL header.

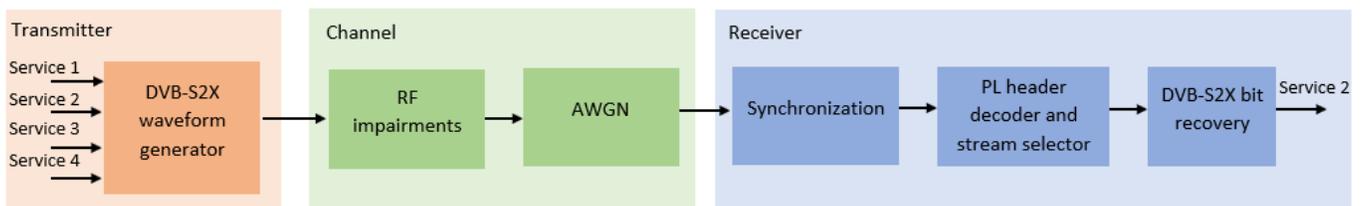


In time slicing receiver, the user selects and decodes a specific stream pertaining to the required service. Based on the TSN information in the header, the receiver decodes certain PL frames and

discards others. Thus, FEC decoding speed is equivalent to that of a regular DVB-S2 application in the order of 100 or 200 Mbps. This approach significantly reduces the complexity of the receiving device and in turn creates the opportunity for new applications like high speed data links over the Ka band (26–40 GHz). The information about the relevant time slicing number is already known to the receiver.

This example designs the synchronization aspects of a DVB-S2X receiver using a time slicing receiver in wideband mode. The example supports all modulation schemes from QPSK to 256APSK. ETSI EN 302 307-2 Section 6 Table 20a and Table 20c [1 on page 4-100] summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. Very low signal to noise ratio (VL-SNR) physical layer signaling (PLS) codes are not supported.

In the example, 4 services are given as independent input streams to the `dvbs2xWaveformGenerator` System object and the receiver recovers input data stream pertaining to service 2 to demonstrate the time slicing feature. This diagram summarizes the example workflow.



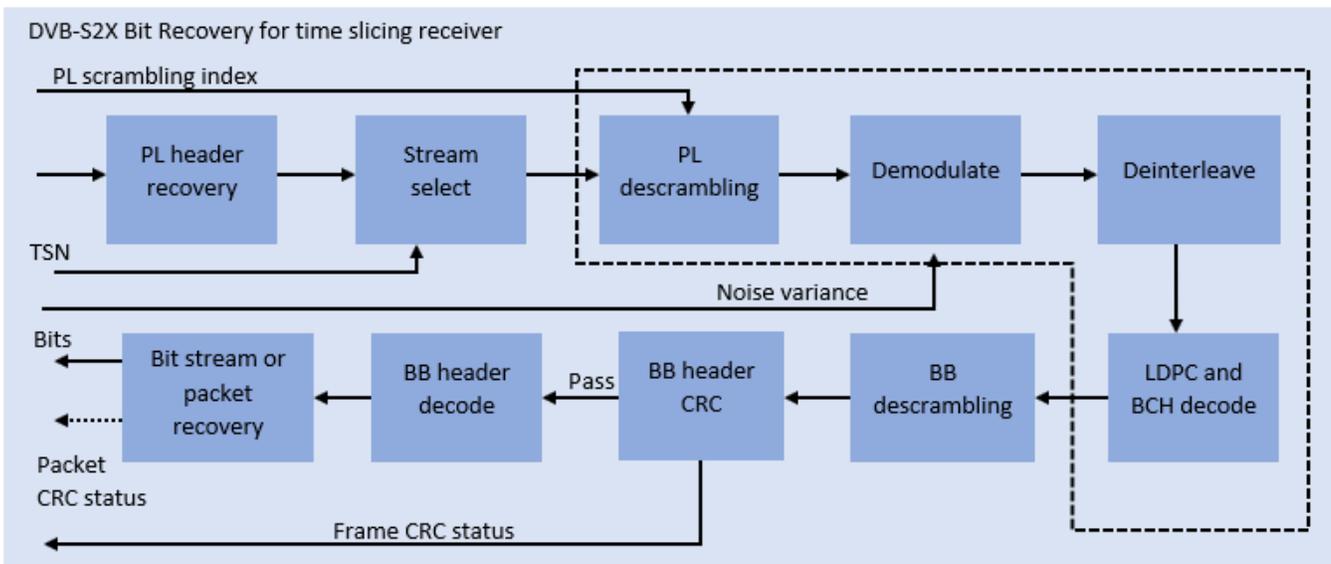
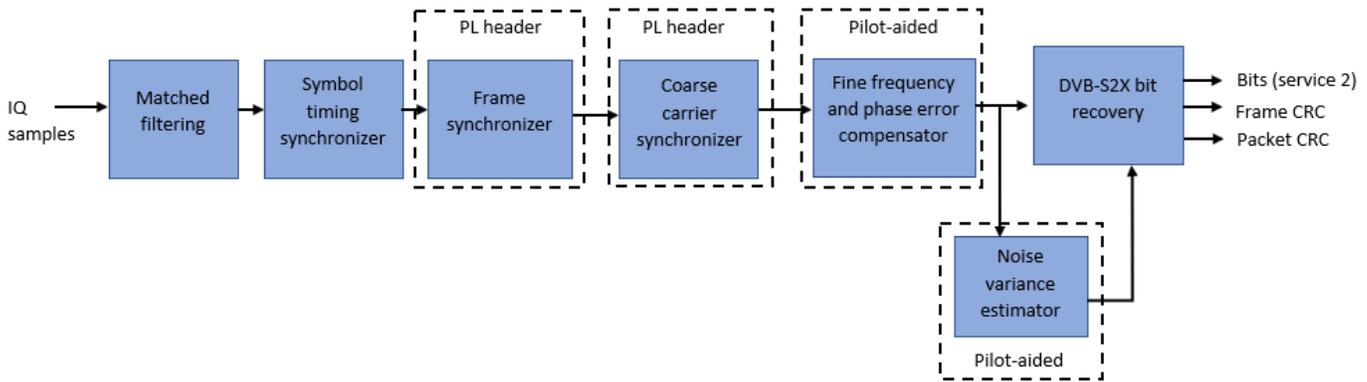
Main Processing Loop

The example processes five PL frames of data per service with the E_s/N_o set to 15 dB, and then computes the BER and PER of the required service PL frames. Carrier frequency offset, frequency drift, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

At the receiver, after matched filtering, perform the timing and carrier recovery operations to recover the transmitted data. To select the PL frames of the required service, the PL header is decoded and depending on the TSN value, appropriate frames are selected for decoding. Along with the TSN, the physical layer transmission parameters, such as the modulation scheme, code rate, and FEC frame type, are recovered from the PL header to enable demodulation and decoding. To regenerate the input bit stream, decode the baseband (BB) header.

Because the DVB-S2X standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured. If the BB frame is a stream of bits, only the decoded bits are returned, and then BER is measured.

These block diagrams show the synchronization and input bit recovery workflows.



Download DVB-S2X LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
    addpath('s2xLDPCParityMatrices');
end
```

DVB-S2X Configuration

Specify the `cfgDVBS2X` structure to define DVB-S2X transmission configuration parameters. `PLSDecimalCode` 129 and 131 are not supported because they are used for generating VL-SNR frames. Only the regular frames are supported. Set `HasTimeSlicing` to true to enable time slicing operation.

```

cfgDVBS2X.StreamFormat = TS ;
cfgDVBS2X.HasTimeSlicing = true;

cfgDVBS2X.NumInputStreams = 4 ;
cfgDVBS2X.PLSDecimalCode = 143;
cfgDVBS2X.DFL = 41128;

cfgDVBS2X.PLScramblingIndex = 0 ;
cfgDVBS2X.RolloffFactor = 0.35;
cfgDVBS2X.SamplesPerSymbol = 2;

```

% 8PSK 23/36
% Data field length in bits for
% code rate 23/36

Simulation Parameters

The DVB-S2X standard for a wideband transponder supports channel bandwidths from 200 to 500 MHz. The symbol rate is calculated as $B/(1+R)$, where B is the channel bandwidth, and R is the transmit filter roll-off factor.

This table defines the phase noise mask (dBc/Hz) used to generate the phase noise that is applied to the transmitted signal. These noise masks are taken from ETSI EN 302 307-1 Annex M.3 [2 on page 4-100].

Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz	10 MHz
Level-1	-25	-50	-73	-93	-103	-114
Level-2	-45	-65	-75	-89	-102	-112
Level-3	-62	-79	-89	-99	-109	-119

The peak Doppler and frequency drift considered are typically expected from a low Earth orbit (LEO) satellite in motion.

```

simParams.serviceNumber = 2;
simParams.sps = cfgDVBS2X.SamplesPerSymbol;
simParams.numFrames = 5;
simParams.chanBW = 200e6;
rolloffFac = cfgDVBS2X.RolloffFactor;
simParams.Fsymb = simParams.chanBW/(1 + rolloffFac);
simParams.cfo = 0.02*simParams.Fsymb;

simParams.dopplerRate = 3e4;
simParams.peakDoppler = 1e6;
simParams.sco = 10;

simParams.phNoiseLevel = Level-1 ;
simParams.EsNodB = 15

simParams = struct with fields:
    serviceNumber: 2
        sps: 2
            numFrames: 5
                chanBW: 200000000

```

% time slicing number to be processed
% Samples per symbol
% Number of frames to be processed
% Channel bandwidth in Hertz

% Symbol rate in symbols/sec
% Carrier frequency offset in Hertz
% due to oscillator instabilities
% Doppler rate in Hertz/sec
% Peak Doppler shift in Hertz
% Sampling clock offset in parts
% per million

% Phase noise level
% Energy per symbol to noise ratio

```

        Fsymb: 1.4815e+08
        cfo: 2.9630e+06
    dopplerRate: 30000
    peakDoppler: 1000000
        sco: 10
    phNoiseLevel: "Level-1"
        EsNodB: 15

```

Generate DVB-S2X Wideband Waveform Distorted with RF Impairments

To create a DVB-S2X waveform, use the `HelperDVBS2XWidebandRxInputGenerate` helper function with the `simParams` and `cfgDVBS2X` structures as inputs. The function returns the transmitted data bits, transmitted, and received waveforms, physical layer configuration parameters as a structure, and a receiver processing structure. The received waveform is impaired with carrier frequency, frequency drift, sampling clock offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

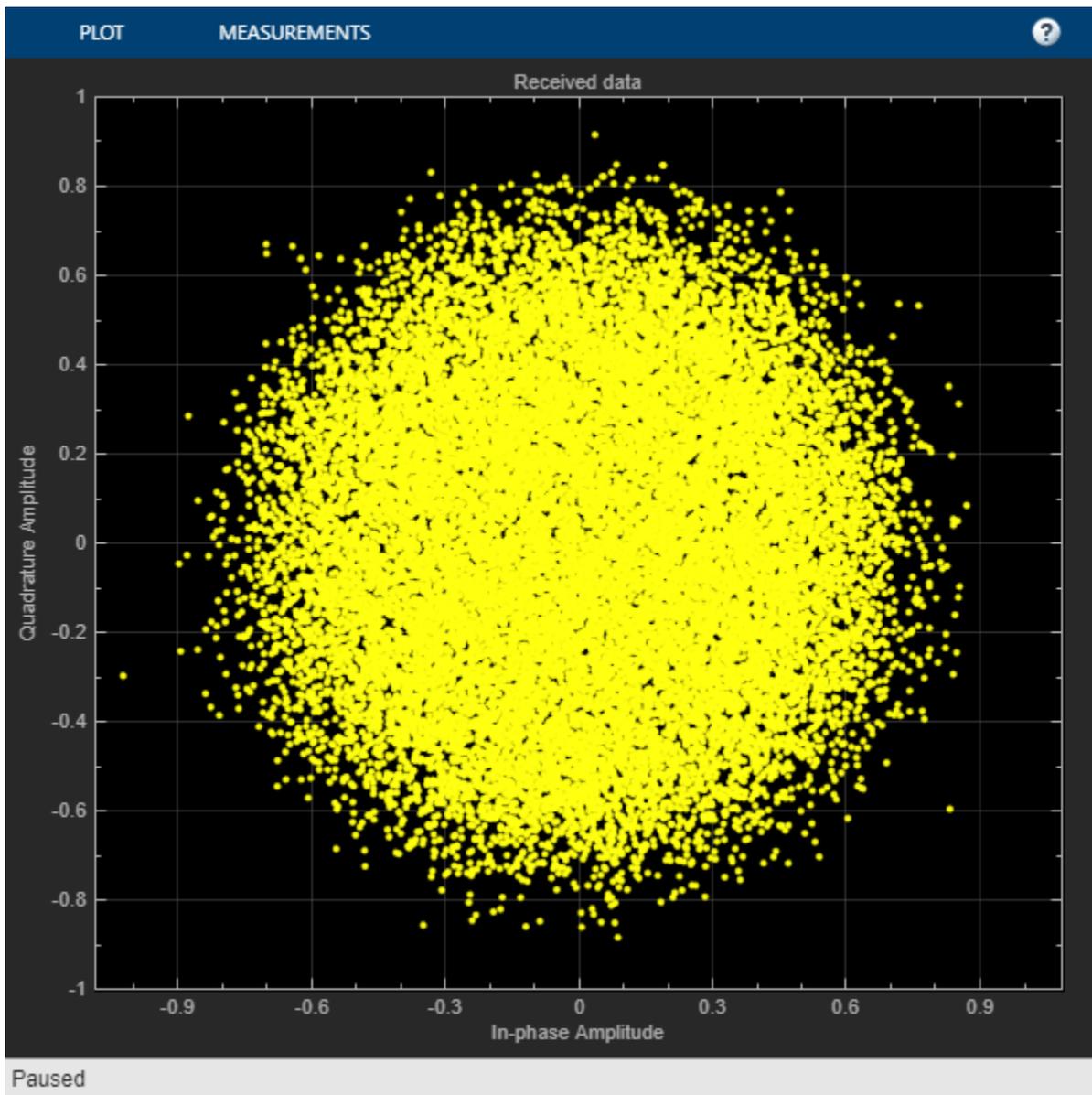
```

[data,txOut,rxIn,phyConfig,rxParams] = HelperDVBS2XWidebandRxInputGenerate(cfgDVBS2X, ...
    simParams);
% Display the physical layer configuration
phyConfig

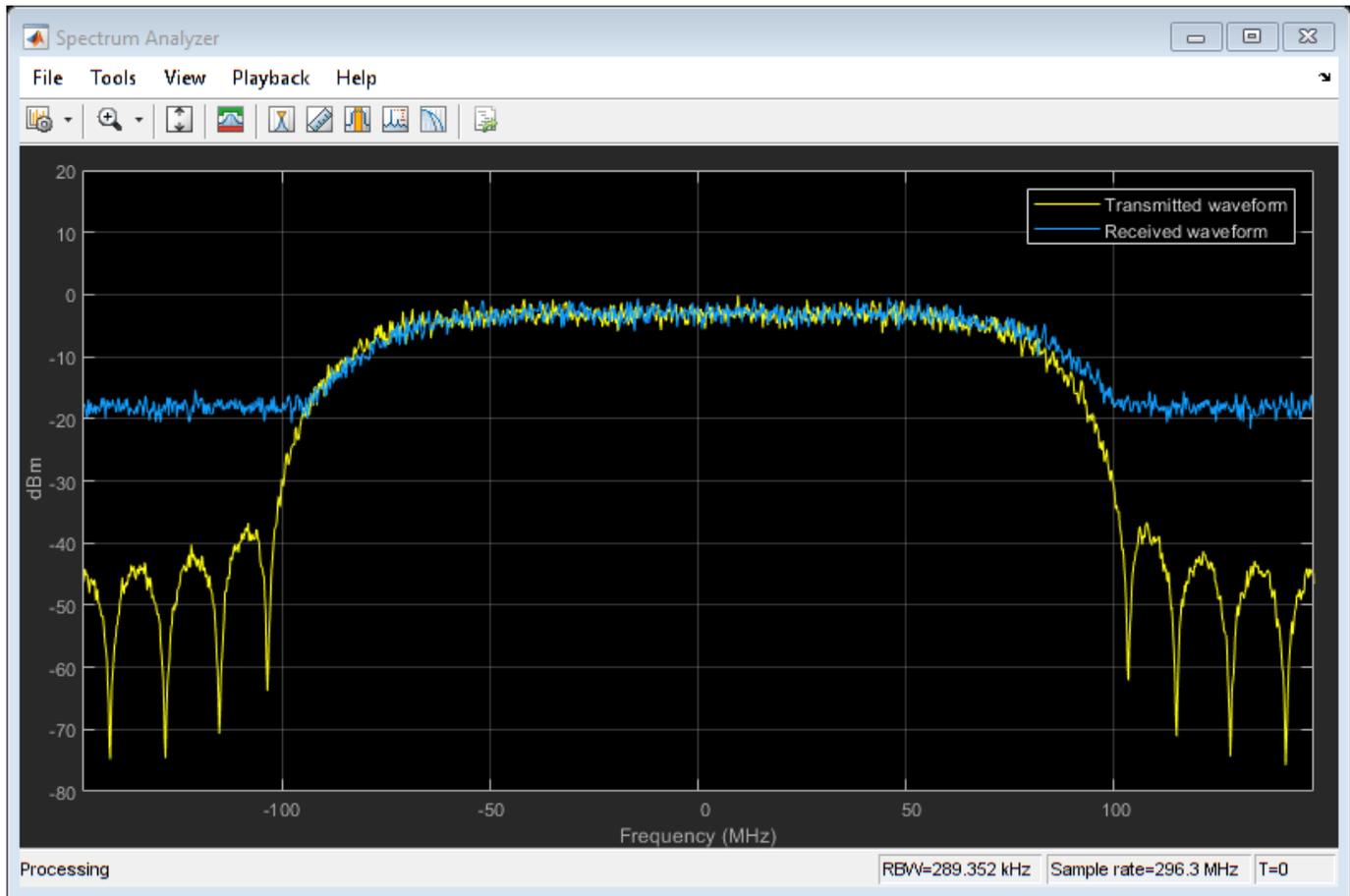
phyConfig = struct with fields:
    FECFrame: {'normal' 'normal' 'normal' 'normal'}
    ModulationScheme: {'8PSK' '8PSK' '8PSK' '8PSK'}
    LDPCCodeIdentifier: {'23/36' '23/36' '23/36' '23/36'}

% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:rxParams.plFrameSize*simParams.sps))

```



```
% Transmitted and received signal spectrum visualization
Fsamp = simParams.Fsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'AveragingMethod','Exponential','ShowLegend',true);
specAn([txOut,rxIn(1:length(txOut))]);
```



Configure Receiver Parameters

At the receiver, matched filtering is performed on the received data and is then followed by symbol timing synchronization, frame synchronization and carrier synchronization. The symbol timing estimator is non-data aided. The preferred loop bandwidth for symbol timing compensation depends on the E_s/N_o setting.

The frame synchronization uses the PL header and two pilot blocks. E_s/N_o plays a crucial role in determining the accuracy of the frame synchronization. The frame synchronization is performed on successive frames for accurate detection.

The coarse frequency error estimation is performed on the PL header. The fine frequency error estimation can estimate carrier frequency offsets up to 3% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency error up to 0.03% of the input symbol rate.

```
rxParams.symbSyncLoopBW = 1e-4;           % Symbol timing synchronizer loop bandwidth
                                           % normalized by symbol rate
rxParams.numFramesForFineFreqLock = 3;    % Number of frames required for fine
                                           % frequency estimation

% Create coarse frequency synchronization System object by using
% HelperDVBS2XWidebandCoarseFrequencySynchronizer helper object
```

```

freqSync = HelperDVBS2XWidebandCoarseFrequencySynchronizer('SampleRate',simParams.Fsymb);

% Create frame synchronization System object by using
% HelperDVBS2XWidebandFrameSynchronizer helper object
frameSync = HelperDVBS2XWidebandFrameSynchronizer(...
    'PLScramblingIndex',cfgDVBS2X.PLScramblingIndex);

% Create symbol timing synchronization System object by using
% comm.SymbolSynchronizer object
symSync = comm.SymbolSynchronizer('TimingErrorDetector','Gardner (non-data-aided)', ...
    'NormalizedLoopBandwidth',rxParams.symbSyncLoopBW, 'DampingFactor',1/sqrt(2));

% Create matched filter System object by using
% comm.RaisedCosineReceiveFilter object
decFac = simParams.sps/2;
rxFilter = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor',0.35, ...
    'InputSamplesPerSymbol',simParams.sps, 'DecimationFactor',decFac);
b = rxFilter.coeffs;
rxFilter.Gain = sum(b.Numerator);

% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

% Initialize data indexing variables
stIdx = 0;
isLastFrame = false;
symBuffer = [];

plFrameSize = rxParams.plFrameSize;
dataSize = rxParams.inputFrameSize;
winLen = plFrameSize;
rxParams.fineFreqCorrVal = zeros(rxParams.numFramesForFineFreqLock,1);
pilotBlkLen = rxParams.pilotBlkLength;

```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data, select the appropriate stream (service) and recover the input data, process the distorted DVB-S2X waveform samples one frame at a time by following these steps.

- 1 Apply matched filtering, outputting at the rate of two samples per symbol.
- 2 Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate.
- 3 Apply frame synchronization to detect the start of frame.
- 4 Estimate and apply coarse frequency offset correction.
- 5 Estimate and apply fine frequency offset correction.
- 6 Estimate and compensate for residual carrier frequency and phase noise.
- 7 Demodulate and decode the PL header.
- 8 Select the PL frames corresponding to the required service based on TSN.
- 9 Demodulate and decode the PL frames.
- 10 Perform CRC check on the BB header, if the check passes, recover the header parameters.
- 11 Regenerate the input stream of data or packets from BB frames.

```

while stIdx < length(rxIn)

    % Use one DVB-S2X PL frame for each iteration.
    endIdx = stIdx + plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are used.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % Retrieve the last frame samples.
    if isLastFrame
        resSampCnt = plFrameSize*rxParams.sps - length(rxData);
        % Inadequate number of samples to fill last frame
        syncIn = [rxData; zeros(resSampCnt,1)];
    else
        syncIn = rxData;
    end

    % Perform matched filtering and downsample the signal to 2 samples per
    % symbol.
    filtOut = rxFilter(syncIn);

    % Apply symbol timing synchronization.
    symSyncOut = symSync(filtOut);

    % Apply frame synchronization
    syncIndex = frameSync(symSyncOut);
    rxParams.syncIndex = [rxParams.syncIndex;syncIndex];

    if length(rxParams.syncIndex) > 1
        % The PL frame start index lies somewhere in the middle of the data
        % being processed. From coarse frequency compensation onwards, the
        % processing happens as a PL frame. A buffer is used to store symbols
        % required to fill one PL frame.
        stIdx1 = rxParams.syncIndex(1);
        endIdx1 = winLen+rxParams.syncIndex(2);

        cfoIn = symBuffer(stIdx1:endIdx1-1);
        % Extract the number of pilots based on frame length
        numPilots = rxParams.possPilotBlks(rxParams.possFrameSizes == length(cfoIn));

        % Apply coarse frequency synchronization.
        [coarseFreqSyncOut,cEstFreq] = freqSync(cfoIn);

        % Extract the pilot indices and PL scrambled pilots based on number
        % of pilots
        pilotInd = rxParams.pilotIndices(1:numPilots(1));
        pilotVal = rxParams.pilotValues(1:numPilots(1));
        numPilotBlks = numPilots(1)/pilotBlkLen;

        % Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
        % helper function.
        corrVal = HelperDVBS2FineFreqEst(coarseFreqSyncOut(pilotInd),numPilotBlks, ...
            pilotVal,0,pilotBlkLen,rxParams.NumLags);

        % Use the correlation values calculated over pilot fields spanning over

```

```

% multiple frames to calculate the fine frequency error estimate.
% The estimation uses a sliding window technique.
fineFreqEst = angle(sum([rxParams.fineFreqCorrVal;corrVal]))/(pi*(rxParams.NumLags));

ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
phErr = exp(-1j*2*pi*fineFreqEst*ind).';
freqCompOut = coarseFreqSyncOut.*phErr(:);

phErrEst = HelperDVBS2PhaseEst(freqCompOut(pilotInd), ...
    pilotVal,0);
phaseCompOut = HelperDVBS2PhaseCompensate(freqCompOut, ...
    phErrEst,pilotInd,0,2);

% Decode the PL header by using the HelperDVBS2PLHeaderRecover
% helper function. Start of frame (SOF) is 26 symbols, which are discarded
% before header decoding. SOF symbols are only required for frame synchronization.
[plsDecCode,timeSliceNum,phyParams] = ...
    HelperDVBS2XWidebandPLHeaderRecover(phaseCompOut(27:180));
% Validate the decoded PL header.
if plsDecCode~=cfgDVBS2X.PLSDecimalCode
    fprintf('%s\n','PL header decoding failed. PLS code is incorrectly decoded')
else
    if timeSliceNum == simParams.serviceNumber
        % Estimate noise variance by using
        % HelperDVBS2NoiseVarEstimate helper function.
        nVar = HelperDVBS2NoiseVarEstimate(phaseCompOut,pilotInd, ...
            pilotVal,rxParams.normFlag);
        % Recover the BB frame by using HelperDVBS2XBBFrameRecover
        % helper function. 180 PL header symbols are discarded.
        rxBBFrame = HelperDVBS2XBBFrameRecover(phaseCompOut(181:end),phyParams, ...
            rxParams.plScramblingIndex,numPilotBlks,nVar,false);
        % Recover the input bit stream by using
        % HelperDVBS2StreamRecover helper function.
        if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
            [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame,true);
            if ~isFrameLost && length(decBits) ~= dataSize(timeSliceNum)
                isFrameLost = true;
            end
        else
            [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame,true);
            if ~isFrameLost && length(decBits) ~= dataSize(timeSliceNum)
                isFrameLost = true;
                pktCRC = zeros(0,1,'logical');
            end
            % Compute the PER for TS or GS packetized mode.
            % Number of packets in error based on packet CRC status
            pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
            % Number of packets recovered from the baseband frame
            pktsRec = pktsRec + numel(pktCRC);
        end
        if ~isFrameLost
            ts = sprintf('%s','BB header decoding passed.');
```

```

frameIdx = ceil((rxParams.frameCount-1)/cfgDVBS2X.NumInputStreams);
fprintf('%s(Number of frames lost = %ld, service %d, frame number = %d)\n',...
    ts,numFramesLost,timeSliceNum,frameIdx)
% Compute the bits in error.
if ~isFrameLost
    dataInd = (frameIdx-1)*dataSize(timeSliceNum)+1:frameIdx*dataSize(timeSliceNum);
    errs = sum(data{timeSliceNum}(dataInd) ~= decBits);
    bitsErr = bitsErr + errs;
end
end
end
symBuffer = symBuffer(endIdx1+1-rxParams.syncIndex(2):end);
winLen = length(symBuffer);
rxParams.fineFreqCorrVal(1:end-1) = rxParams.fineFreqCorrVal(2:end);
rxParams.fineFreqCorrVal(end) = corrVal;
rxParams.syncIndex = rxParams.syncIndex(end);
end
symBuffer = [symBuffer;symSyncOut]; %#ok<AGROW>
stIdx = endIdx;
rxParams.frameCount = rxParams.frameCount + 1;
end

BB header decoding failed.(Number of frames lost = 1, service 2, frame number = 1)
BB header decoding passed.(Number of frames lost = 1, service 2, frame number = 2)
BB header decoding passed.(Number of frames lost = 1, service 2, frame number = 3)
BB header decoding passed.(Number of frames lost = 1, service 2, frame number = 4)
BB header decoding passed.(Number of frames lost = 1, service 2, frame number = 5)

```

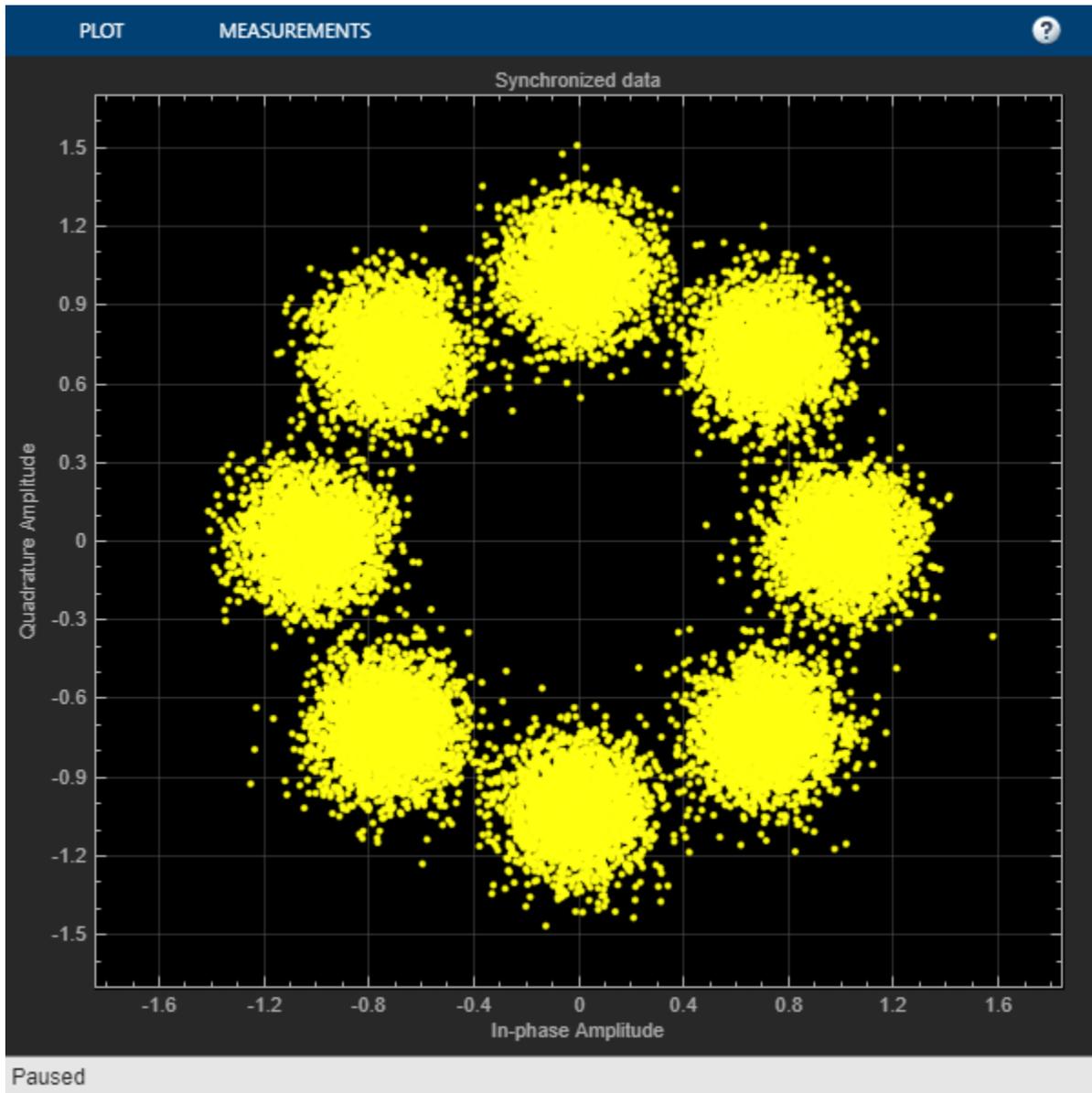
Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```

% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title','Synchronized data', ...
    'XLimits',[-1.7 1.7],'YLimits',[-1.7 1.7], ...
    'ShowReferenceConstellation',false);
syncConst(phaseCompOut)

```



```

% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        numFramesRec = simParams.numFrames - numFramesLost;
        ber = bitsErr/(numFramesRec*dataSize(timeSliceNum));
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
    else

```

```

    if strcmpi(cfgDVBS2X.StreamFormat, 'TS')
        pktLen = 1504;
    else
        pktLen = cfgDVBS2X.UPL;      % UP length including sync byte
    end
    ber = bitsErr/(pktsRec*pktLen);
    per = pktsErr/pktsRec;
    fprintf('PER: %1.2e\n',per)
    fprintf('BER: %1.2e\n',ber)
end
end
PER: 0.00e+00
BER: 0.00e+00

```

Further Exploration

This section explains how to set some of the fields of `rxParams` to make the example work for low SNR conditions.

Configure Symbol Timing Synchronization Parameters

Because the operating E_s/N_o range of the DVB-S2X standard is very low, you must use small value for `rxParams.symbSyncLoopBW` parameter for proper symbol synchronization. If the symbol timing loop does not converge, try decreasing the `rxParams.symbSyncLoopBW` parameter. To achieve convergence of the timing loop, the ratio `rxParams.symbSyncLoopBW/simParams.sps` must be greater than $1e-5$.

Configure Carrier Synchronization Parameters

The `HelperDVBS2PhaseEst` helper function can estimate the phase offset accurately only if the residual carrier frequency offset (CFO) is less than 0.0003 times the symbol rate. Use the second output argument `cEstFreq` from the `step` call of `freqSync` and the fine frequency offset error estimate `fineFreqEst` to calculate the overall CFO estimated.

```

% normCFOEst = cEstFreq/simParams.Fsymb + fineFreqEst;
% actCFO = (simParams.cfo + simParams.peakDoppler)/simParams.Fsymb;
% residualErr = abs(actCFO - normCFOEst);

```

Increase the `rxParams.numFramesForFineFreqLock` value to improve the accuracy of the fine frequency error estimate if `residualErr` is not less than the desired result.

Appendix

The example uses these helper functions:

- `HelperDVBS2XWidebandRxInputGenerate.m`: Generate DVB-S2X waveform samples distorted with RF impairments and structure of parameters for receiver processing
- `HelperDVBS2PhaseNoise.m`: Generate phase noise samples for different DVB-S2X phase noise masks and apply it to the input signal
- `HelperDopplerShift.m`: Generate sinusoidal varying Doppler shift and apply it to input signal
- `HelperDVBS2XWidebandCoarseFrequencySynchronizer.m`: Perform coarse frequency offset estimation and correction
- `HelperDVBS2XWidebandFrameSynchronizer.m`: Perform frame synchronization and detect start of frame

- HelperDVBS2FineFreqEst.m: Estimate fine frequency offset
- HelperDVBS2PhaseEst.m: Estimate carrier phase offset
- HelperDVBS2PhaseCompensate.m: Perform carrier phase compensation
- HelperDVBS2XPLHeaderRecover.m: Demodulate and decode PL header to recover transmission parameters
- HelperDVBS2NoiseVarEstimate.m: Estimate noise variance of received data
- HelperDVBS2XBBFrameRecover.m: Perform PL descrambling, demodulation, decoding and recover BB frame from PL frame
- HelperDVBS2XDemapper.m: Perform soft demodulation for all DVB-S2X based modulation schemes
- HelperDVBS2XLDPCCode.m: Perform LDPC decoding for all DVB-S2X based LDPC frame formats and code rates
- HelperDVBS2XBCHDecode.m: Perform BCH decoding for all DVB-S2X based frame formats and code rates
- HelperDVBS2StreamRecover.m: Perform CRC check of BB header and recover input stream from BB frame based on header parameters
- HelperDVBS2XBitRecover.m: Perform PL header demodulation and decoding, PL de-scrambling, demodulation, decoding and recover BB frame. Perform CRC check of BB header, and recover input stream from BB frame.

Bibliography

- 1 ETSI Standard EN 302 307-2 V1.1.1(2015-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 extensions (DVB-S2X)*.
- 2 ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 3 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 4 P. Kim and D. Oh, "Frame detection for DVB-S2 based time slice," *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, 2014, pp. 1-2, doi: 10.1109/ISCE.2014.6884410.
- 5 Casini, E., et al. "DVB-S2 Modem Algorithms Design and Performance over Typical Satellite Channels." *International Journal of Satellite Communications and Networking*, vol. 22, no. 3, May 2004, pp. 281-318.
- 6 Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

See Also

Objects

dvbs2xWaveformGenerator | dvbs2WaveformGenerator

Related Examples

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames” on page 4-69
- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames” on page 4-54

End-to-End CCSDS SCPPM Simulation Using Deep Space Poisson Channel

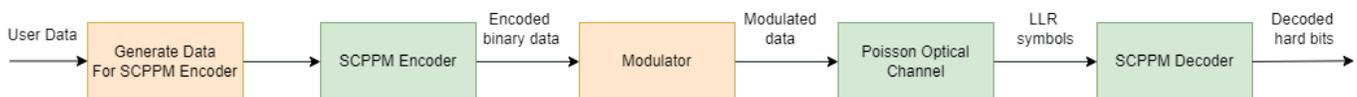
This example shows how to perform bit error rate (BER) analysis for the Consultative Committee for Space Data Systems (CCSDS) serially concatenated pulse position modulation (SCPPM) end-to-end chain using a deep space Poisson channel. The example also describes the information processing in the SCPPM encoder, passing the encoded data through the Poisson optical channel, and then decoding the received soft bits using the SCPPM decoder. This example uses the SCPPM encoder defined in CCSDS 142.0-B-1 section 3.8 [1] on page 4-108 and the SCPPM decoder defined in Coded Modulation for the Deep-Space Optical Channel section III.A [2] on page 4-108. The optical channel used is a binary-input, soft-output, deep space pulse position modulation (PPM) Poisson channel, as defined in Coded Modulation for the Deep-Space Optical Channel section IV [2] on page 4-108.

Processing Chain

To analyze the BER performance for a CCSDS SCPPM end-to-end chain, follow these steps.

- 1 Generate SCPPM encoder input from the user data. Add the cyclic redundancy check (CRC) and termination bits to this user data.
- 2 Encode the generated data using the SCPPM encoder. The output is a sequence of SCPPM codewords where each codeword consists of $15120/m$ PPM symbols. m is the modulation order, specified as an integer in the range [2, 8]. A PPM symbol is an integer value corresponding to each grouping of m binary code symbols.
- 3 Modulate the SCPPM codewords using the M -ary PPM. Each code symbol is mapped to a binary vector of length M (where $M = 2^m$) by placing $M-1$ 0s and a single 1 into one of the possible positions, based on the code symbol value.
- 4 Transmit the modulated binary data over a deep space optical channel that is modeled as a Poisson point process. This channel adds Poisson-distributed noise to the signal, and calculates bit-wise log-likelihood ratios (LLRs) of the input signal.
- 5 Simultaneously demodulate and decode the signal at the receiver, using the SCPPM decoder, to recover the user data.

This block diagram illustrates the optical communications channel.



SCPPM Encoder

To generate data for the SCPPM encoder, first generate random frames of input data. The table shows the supported information block sizes corresponding to the code rate, as described in CCSDS 142.0-B-1 section 3.4.1 [1] on page 4-108. Append 32 CRC binary digits to the end of each information block, as described in CCSDS 142.0-B-1 section 3.6 [1] on page 4-108. These CRC bits help in block error detection and SCPPM decoder termination. Append two 0s to the CRC-attached information block to terminate the outer convolutional encoder of the SCPPM encoder. The resultant encoder input data, u , has 34 additional bits (32 CRC bits + 2 termination bits) as compared to the user data.

Code rate (r)	User data size (in bits)	Resultant data in bits (u) = User data size (in bits) + CRC bits (32) + termination bits (2)
1/3	5006	5040
1/2	7526	7560
2/3	10046	10080

As specified in CCSDS 142.0-B-1 section 3.6 [1] on page 4-108:

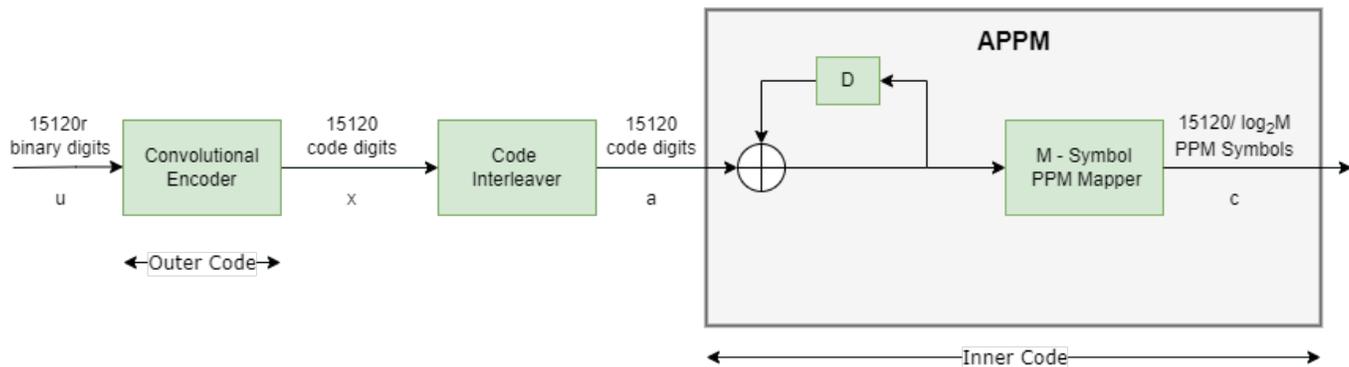
- The generator polynomial for the CRC algorithm is fixed to $x^{32} + x^{29} + x^{18} + x^{14} + x^3 + 1$.
- The initial states of the internal shift register are fixed to 1.

```
crc32Generator = comm.CRCGenerator(...
    Polynomial="x^32+x^29+x^18+x^14+x^3+1", ...
    InitialConditions=1);
```

```
infoSize = 7526;
```

% Information block size without CRC

This figure shows the individual components of the SCPPM encoder.



The SCPPM encoder consists of two constituent codes, referred to as the outer and inner codes, connected by a bit interleaver. The outer code is a short-constraint-three-length convolutional code defined by [5, 7, 7] in octal notation. This table shows the outer convolutional encoder puncturing patterns for the supported code rates, as described in CCSDS 142.0-B-1 section 3.8.2.3.1 [1] on page 4-108.

Code rate (r)	Puncturing Pattern [P ₀ P ₁ P ₂ P ₃ P ₄ P ₅]
1/3	[1 1 1 1 1 1]
1/2	[1 1 0 1 1 0]
2/3	[1 1 0 0 1 0]

The inner code consists of an accumulator ($\frac{1}{1+D}$ filter) and a memoryless PPM modulator, together referred to as an accumulator pulse position modulation (APPM) code. Each SCPPM encoder input block (\mathbf{u}) is of length $15120r$, encoded by the outer convolutional encoder, and yields a code sequence (\mathbf{x}) of length 15120. Interleave this code, bit wise, to produce a sequence (\mathbf{a}) that the accumulator encodes. Map the output sequence from the accumulator to M-ary PPM symbols (\mathbf{c}), an M-length vector, where M is a managed parameter with values from the set {4, 8, 16, 32, 64, 128, 256}, where every $m = \log_2(M)$ binary digits are grouped to form one PPM symbol. Each PPM symbol is an integer in the range [0, M-1].

```
m =  ;
M = 2^m; % M-ary PPM
```

Deep Space Poisson Channel

For this example, consider an optical communications system that uses direct photon detection with a high-order PPM. An M-order PPM uses a time interval that is divided into M possible pulse locations, but places only a single pulse into one of the possible positions. The information to be transmitted determines the position of the pulse. After modulation, the overall length in bits is $\frac{15120}{m} \times 2^m$. Pass this modulated data through the channel.

The receiver, focuses light on the detector. The detector responds to individual photons. Consider these parameters at the detector.

- k — Estimated photon count within each slot
- λ_s — Average number of signal photon incident on the detector per second (including both pulsed and non-pulsed periods)
- λ_b — Average number of noise photon incident per second
- η — Quantum efficiency of the detector
- T_s — Slot time
- $1/M$ — Duty cycle (denotes the average number of pulses transmitted per T_s)

For a PPM channel, the output of a photon-counting detector is modeled as a Poisson process, where the average number of signal photons per pulse is $n_s = \eta\lambda_sMT_s$, and the average number of noise photons per slot is $n_b = \eta\lambda_bT_s$. Assuming the pulse energy is captured in a slot, n_s also denotes the

mean number of signal photons per pulsed slot. For a deep space optical Poisson channel, calculate the LLR of each slot by using these formulas:

The probability of receiving k , given the pulse is not sent in that slot : $p_0(k) = \frac{e^{-n_b} n_b^k}{k!}$.

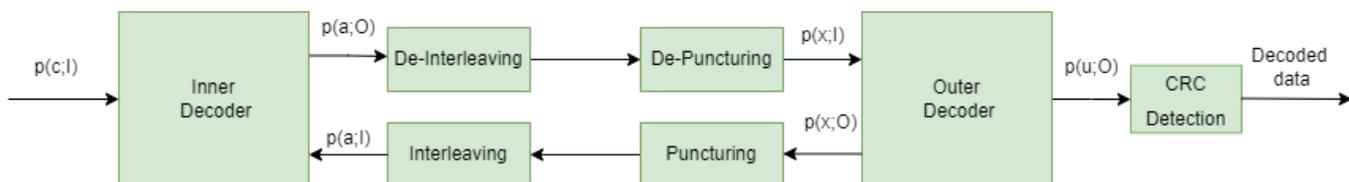
The probability of receiving k , given the pulse is sent in that slot : $p_1(k) = \frac{e^{-(n_s + n_b)} (n_s + n_b)^k}{k!}$.

LLR of each slot : $LLR(k) = \log \frac{p_1(k)}{p_0(k)} = k \log(1 + \frac{n_s}{n_b}) - n_s$.

```
Ts = 32; % PPM slot width in nanoseconds
pps = linspace(-30.25, -29.88, 7)'; % Average number of signal photons per nano second (
ns = power(10, pps/10) * M * Ts; % Average number of signal photons per pulsed slot
nb = 0.2; % Average number of noise photons per slot
```

SCPPM Decoder

This figure shows the SCPPM decoder, as described in Coded Modulation for the Deep-Space Optical Channel section III.A [2] on page 4-108. Each encoder maps an input sequence, either u or a , into an output sequence, either x or c . In an iterative decoding process, each encoder has a corresponding decoder. Conventionally, the modulation and error control coding (ECC) are decoded sequentially, with the demodulator sending its results to the ECC decoder. In SCPPM decoding, consider the modulation and ECC as a single large code, which maps user information bits directly to the symbols transmitted on the channel, and uses an iterative demodulator-decoder to decode this large code. Once the underlying Trellis descriptions of the outer convolutional code, inner accumulator, and modulation are formulated, standard forward-backward algorithms can be used. Numerical results show that this is a powerful technique to obtain near-capacity performance.



The inner decoder takes the input a priori likelihoods of input of the inner encoder, a , and output of the inner encoder, c , and produces an updated likelihood of a . Similarly, the outer decoder takes the input a priori likelihoods of output of outer encoder, x , and produces updated likelihoods of u and x . Iterations between the inner and outer code terminate based on a stopping rule. The SCPPM decoder terminates the iterations if CRC passes, or if it reaches the maximum number of decoding iterations. If CRC passes, the decoder considers it as a valid frame. Otherwise, the decoder considers it as an invalid frame.

```
numFrames = 20;
numErrFrames = zeros(length(ns), 1);
berEst = zeros(length(ns), 1);
errorRate = comm.ErrorRate;
```

```
for itr = 1:length(ns)
    rng("default")
    for frmIdx=1:numFrames
        % Generate input data
```

```

data = randi([0 1],infoSize,1);
crcData = crc32Generator(data);           % Generate CRC
msgIn = [crcData; 0; 0];                  % Add termination bits

% Perform SCPPM encoding
[encSym,info] = ccscdsSCPPMEncode(msgIn,m);
r = info.OuterEncoderCodeRate;

% M-ary PPM Modulation
modOut = zeros(length(encSym)*M,1);
mapIndex = (0:length(encSym)-1)'*M + encSym + 1;
modOut(mapIndex) = 1;

% Pass through Poisson channel
receivedCode = HelperDeepSpacePoissonChannel(...
    modOut,ns(itr),nb);

% SCPPM decoding
[decData,errFrames] = ccscdsSCPPMDecode(...
    receivedCode,r,m);
errorStats = errorRate(int8(msgIn),decData);
numErrFrames(itr) = numErrFrames(itr)+errFrames;
end
berEst(itr) = errorStats(1);
reset(errorRate)
end

```

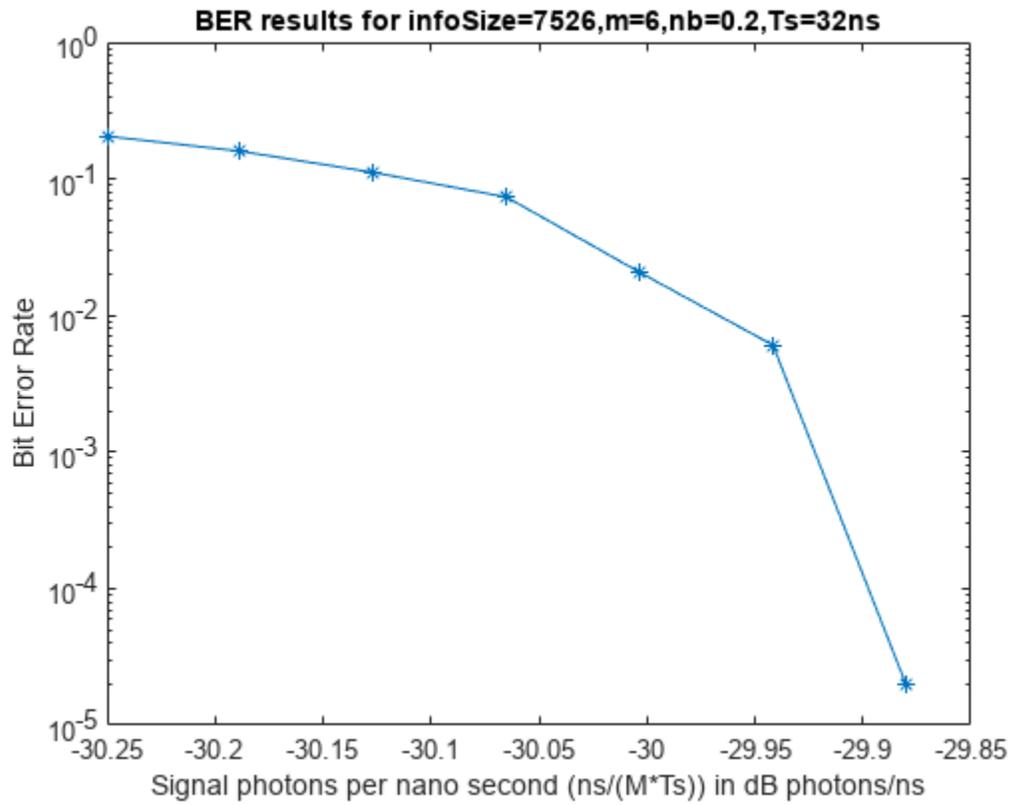
Plot BER

Visualize the BER results. Plot the average number of signal photons per nano second $\frac{ns}{M \times Ts}$ on the x-axis and bit error rate on the y-axis.

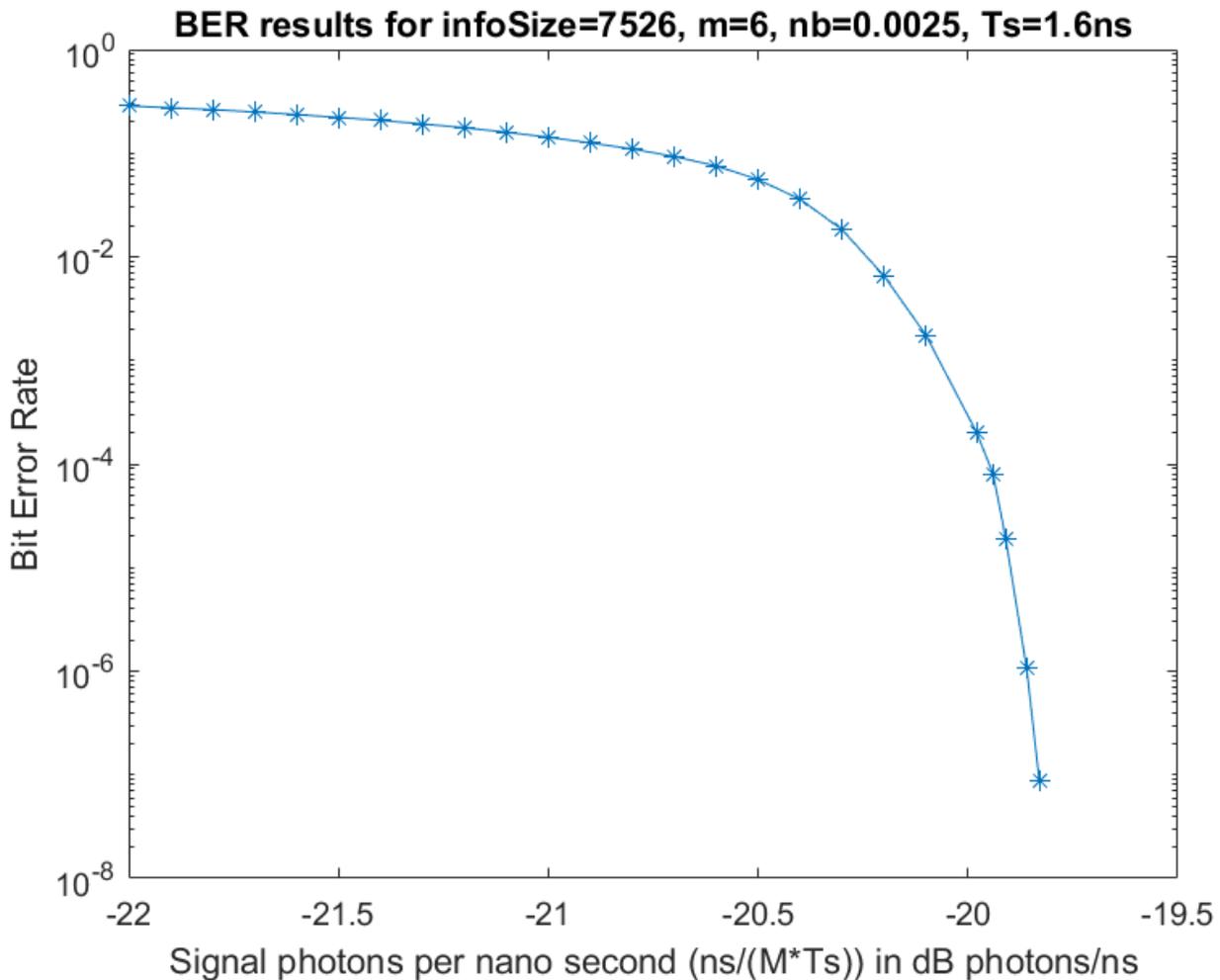
```

semilogy(pps,berEst,'-*')
xlabel("Signal photons per nano second (ns/(M*Ts)) in dB photons/ns")
ylabel("Bit Error Rate")
title(['BER results for infoSize=',num2str(infoSize),' ,m=',num2str(m),' ,nb=',num2str(nb),' ,Ts=' ,

```



This figure shows the BER analysis for an SCPPM decoder over a Poisson channel, for 1500 frames with information size = 7526, code rate (r) = 1/2, average number of noise photons per slot (nb) = 0.0025, slot time (T_s) = 1.6, and modulation order (m) = 6.



Appendix

This example uses the helper file:

HelperDeepSpacePoissonChannel — Add Poisson distributed noise to input signal

References

[1] The Consultative Committee for Space Data Systems. *Optical Communications Coding and Synchronization, Recommended Standard, Issue 1*. CCSDS 142.0-B-1. Washington, D.C.: CCSDS, August 2019. <https://public.ccsds.org/Pubs/142x0b1.pdf>.

[2] Moision, B., and J. Hamkins. "Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation." *Interplanetary Network Progress Report 42-161* (May 1, 2005): 1-25. <https://ui.adsabs.harvard.edu/abs/2005IPNPR.161T...1M>.

[3] Moision, B., and J. Hamkins. "Deep-Space Optical Communications Downlink Budget: Modulation and Coding." Interplanetary Network Progress Report 42-154 (August 1, 2003): 1-28. <https://ui.adsabs.harvard.edu/abs/2003IPNPR.154K...1M>.

[4] Cheng, M. K., M. A. Nakashima, B. E. Moision, and J. Hamkins. "Optimizations of a Turbo-Like Decoder for Deep-Space Optical Communications." Interplanetary Network Progress Report 42-168 (February 1, 2007): 1-31. <https://ui.adsabs.harvard.edu/abs/2007IPNPR.168G...1C>.

See Also

[ccsdsSCPPMEncode](#) | [ccsdsSCPPMDecode](#)

Related Examples

- "CCSDS Optical High Photon Efficiency Telemetry Waveform Generation" on page 2-17

DVB-S2 Bent Pipe Simulation with RF Impairments and Corrections

This model shows a bent pipe satellite link that transmits a Digital Video Broadcasting Satellite Second Generation (DVB-S2) [5] signal from a first ground station to a satellite. The satellite receives the analog signal, amplifies and filters it without demodulation, then retransmits it to a second ground station. That ground station demodulates and decodes the signal, and a testbench calculates the end-to-end packet error rate (PER) and a low density parity check (LDPC) coding bit error rate (BER).

Introduction

The model creates a DVB-S2 signal that includes:

- Bose-Chaudhuri-Hocquenghem (BCH) encoding
- LDPC encoding (normal and short frame) [6], [7]
- Interleaving
- Modulation (QPSK, 8PSK)

The model also optionally applies multiple RF impairments to the signal on the uplink and downlink, and can also optionally correct them. These impairments include:

- Equation-based and table-based memoryless nonlinearities [1]
- Doppler error
- Receiver thermal noise [4]
- Analog filter effects
- Phase noise [2], [3]
- Amplitude and phase imbalances
- DC offset

This example combines the modeling done in the following examples:

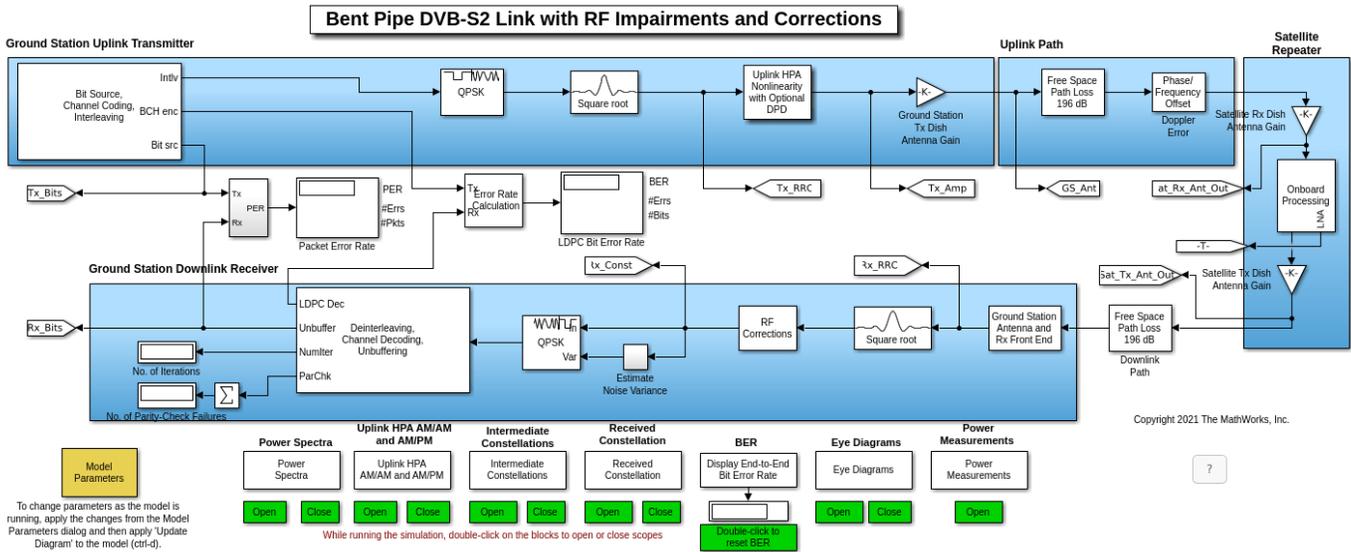
- “RF Satellite Link”
- “DVB-S.2 Link, Including LDPC Coding in Simulink”

Refer to these examples to gain the background necessary to understand this bent pipe example.

Model Overview

This example loads a MAT-file with DVB-S2 LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, then the example downloads them from mathworks.com. Internet connectivity is required to perform this download operation.

The model is shown in the following figure:



The ground station transmitter and uplink path are shown in the top half of the model, and the satellite repeater on the right side of the model. The downlink and ground station receiver are shown in the bottom half of the model. You can change parameters by interacting with the **Model Parameters** block.

The **Model Parameters** block enables you to update uplink, satellite, and downlink parameters separately. In particular, the block enables you to specify the diameters of the ground station and satellite transmit and receive antennas. With the block you can also set the noise figures of the satellite and the receiving ground station analog front ends.

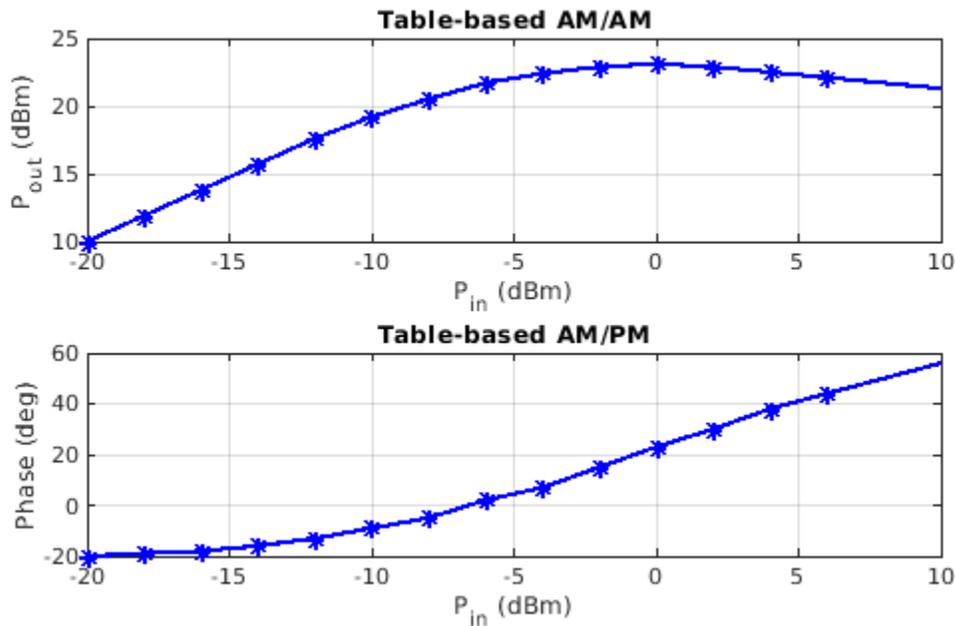
The **Model Parameters** block also enables you to define in-phase/quadrature (I/Q) amplitude imbalance in dB, I/Q phase imbalance in degrees, and an in-phase DC offset as a percentage of the mean received in-phase signal amplitude.

The model also enables multiple visualizations:

- Power spectra
- Constellation diagrams
- Eye diagrams
- AM/AM and AM/PM curves to show nonlinearity effects

The satellite repeater includes several operations not found in the two examples referenced above. First, the repeater models an analog Chebyshev filter to reduce the noise in the signal received by the satellite. You can examine the filter characteristics using the `fvtool` function, using the syntax `fvtool(paramRFSatLink.ChebyNumerator,paramRFSatLink.ChebyDenominator)`.

Also, the satellite repeater employs an amplifier that uses a table-based memoryless nonlinearity. You can use the "Plot Power Characteristics" button of the **Onboard Processing/HPA Nonlinearity** block to generate AM/AM and AM/PM plots for the amplifier. The following figure shows the amplifier AM/AM and AM/PM characteristics.

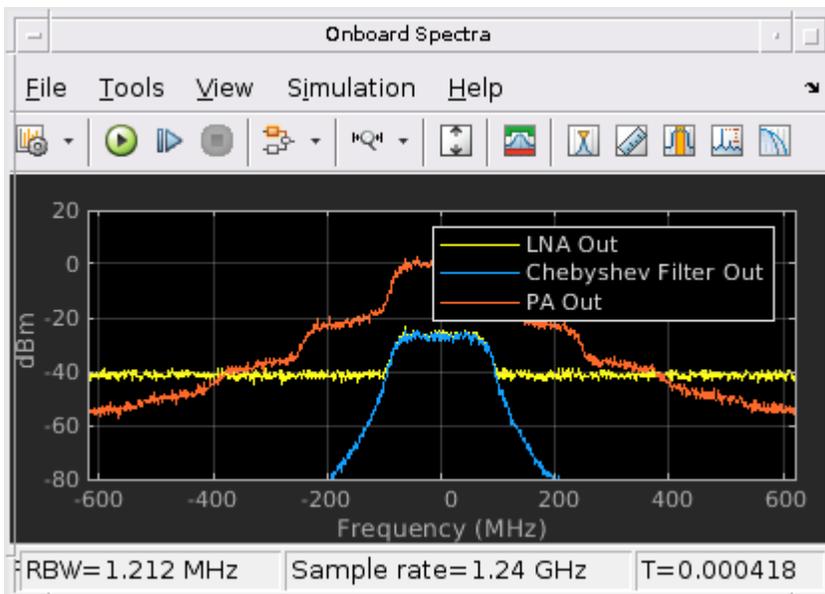
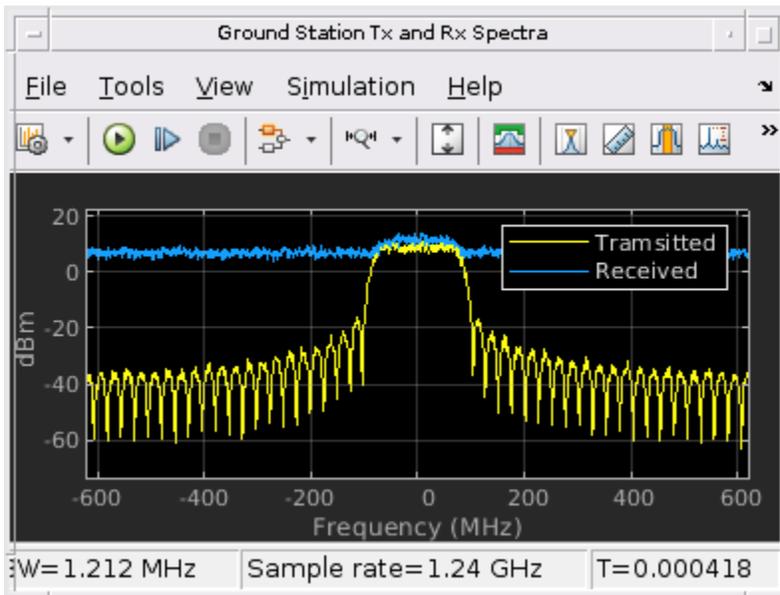


The soft decision QPSK or 8PSK demodulator requires an estimate of the noise variance at its input in order to properly calculate the approximate log-likelihood ratios. The model performs a realistic variance calculation by comparing the received signal against the ideal constellation and calculating error vectors between them. When the noise and other distortions are sufficiently small, the variance calculation is accurate. When the impairments increase such that received constellation points cross over into adjacent, incorrect decision regions, the variance calculation will be overly optimistic.

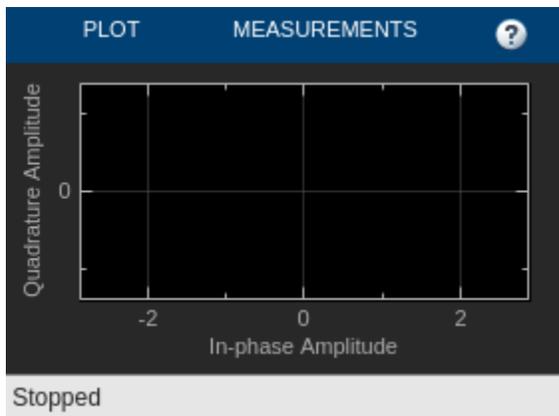
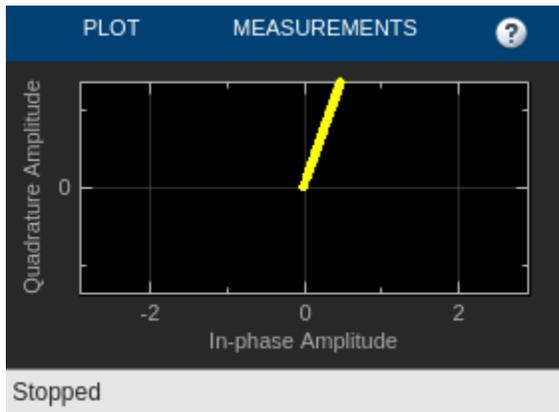
Simulation Results

Run the example to see the following run-time visualizations:

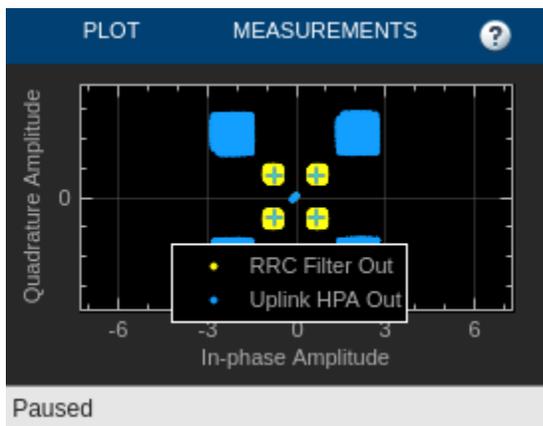
- Power spectra of the transmit and receive ground station signals, and at multiple points during the satellite onboard processing

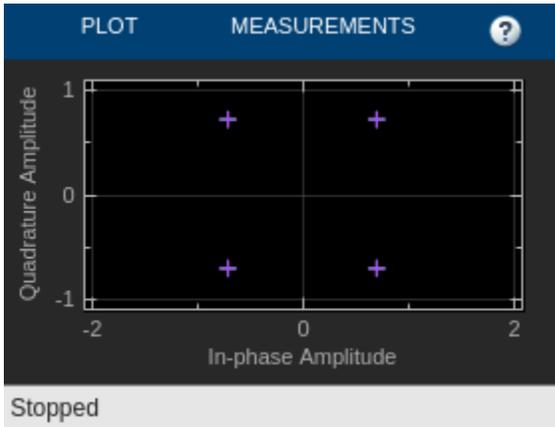


- AM/AM and AM/PM characteristics of the uplink power amplifier

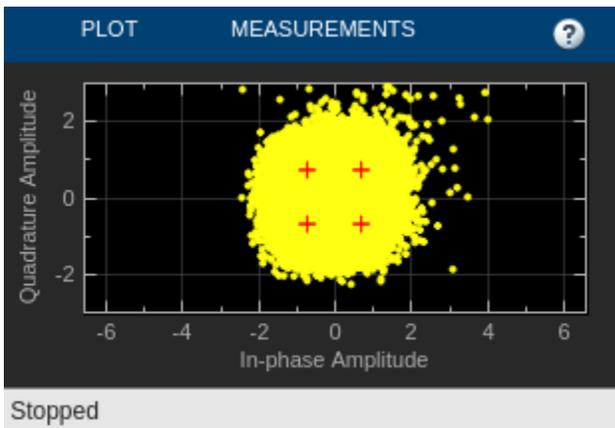


- Constellations before and after the uplink amplifier, and during onboard processing

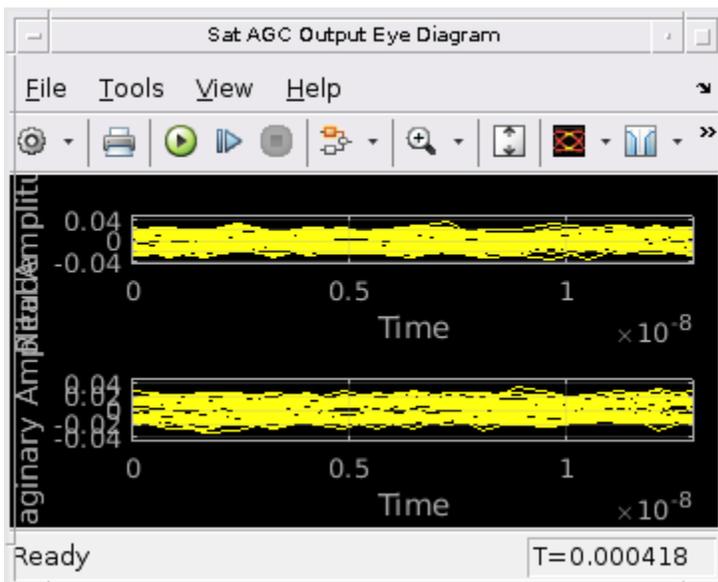


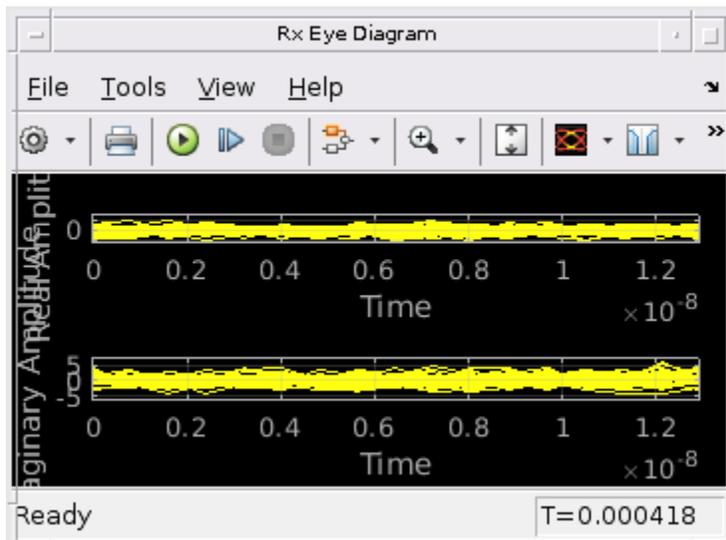
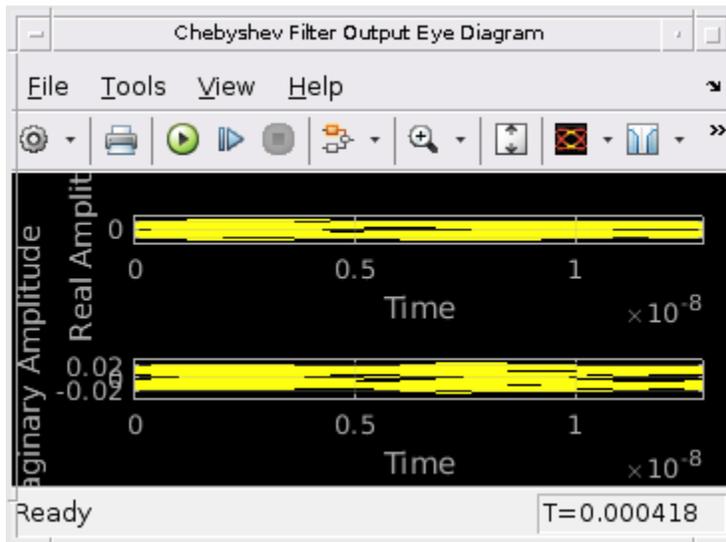


- Received constellation at the ground station input



- Eye diagrams before and after the onboard Chebyshev filter, and at the ground station receiver input





In addition, during run time you can inspect the signal power at the transmitting ground station antenna, the satellite receiver antenna output, the satellite low noise amplifier (LNA) output, and the satellite transmit antenna output.

Further Exploration

You can experiment with the example in the following ways:

- Change modulation and coding formats to determine when the BER unacceptably degrades for a given signal-to-noise ratio (SNR) scenario.
- Turn on single distortions to qualitatively and quantitatively determine their impact on PER and BER.
- Enable RF corrections to ensure that they restore signal quality and PER.
- Reduce the SNR to a level where the RF corrections are no longer effective.
- Navigate to the RF Corrections subsystem and tune the parameter values of the individual blocks in the subsystem, such as the Carrier Synchronizer or the DC Blocker.

- Increase the Chebyshev filter order to determine if the increased group delay distortion affects PER.
- Reduce the satellite amplifier backoff factor to examine its effect on SNR and PER.
- Instead of a geostationary altitude of 35,600 km, change the satellite altitude to a MEO altitude of 20,000 km or a LEO altitude of 2,000 km. Examine how the antenna sizes can then be reduced, or the receiver noise figure can be increased.
- Experiment with different uplink and downlink frequencies.
- Investigate the effect of digital predistortion (DPD) on PER when the uplink amplifier is driven into its saturation region.

Bibliography

[1] Saleh, A. A. M. "Frequency-Independent and Frequency-Dependent Nonlinear Models of TWT Amplifiers." *IEEE Transactions on Communications*, Vol. 29, No. 11, Nov. 1981.

[2] Kasdin, N.J. "Discrete Simulation of Colored Noise and Stochastic Processes and $1/(f^\alpha)$; Power Law Noise Generation." *Proceedings of the IEEE*, Vol. 83, No. 5, May 1995.

[3] Kasdin, N. J., and T. Walter "Discrete Simulation of Power Law Noise ." *Proceedings of the 1992 IEEE Frequency Control Symposium*, IEEE 1992.

[4] Sklar, Bernard, and Fredric J. Harris. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[5] ETSI Standard EN 302 307 V1.1.1(2005-03). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications*.

[6] Gallager, Robert. "Low-Density Parity-Check Codes." *IRE Transactions on Information Theory*, Vol. 8, No. 1, Jan. 1962: 21-28.

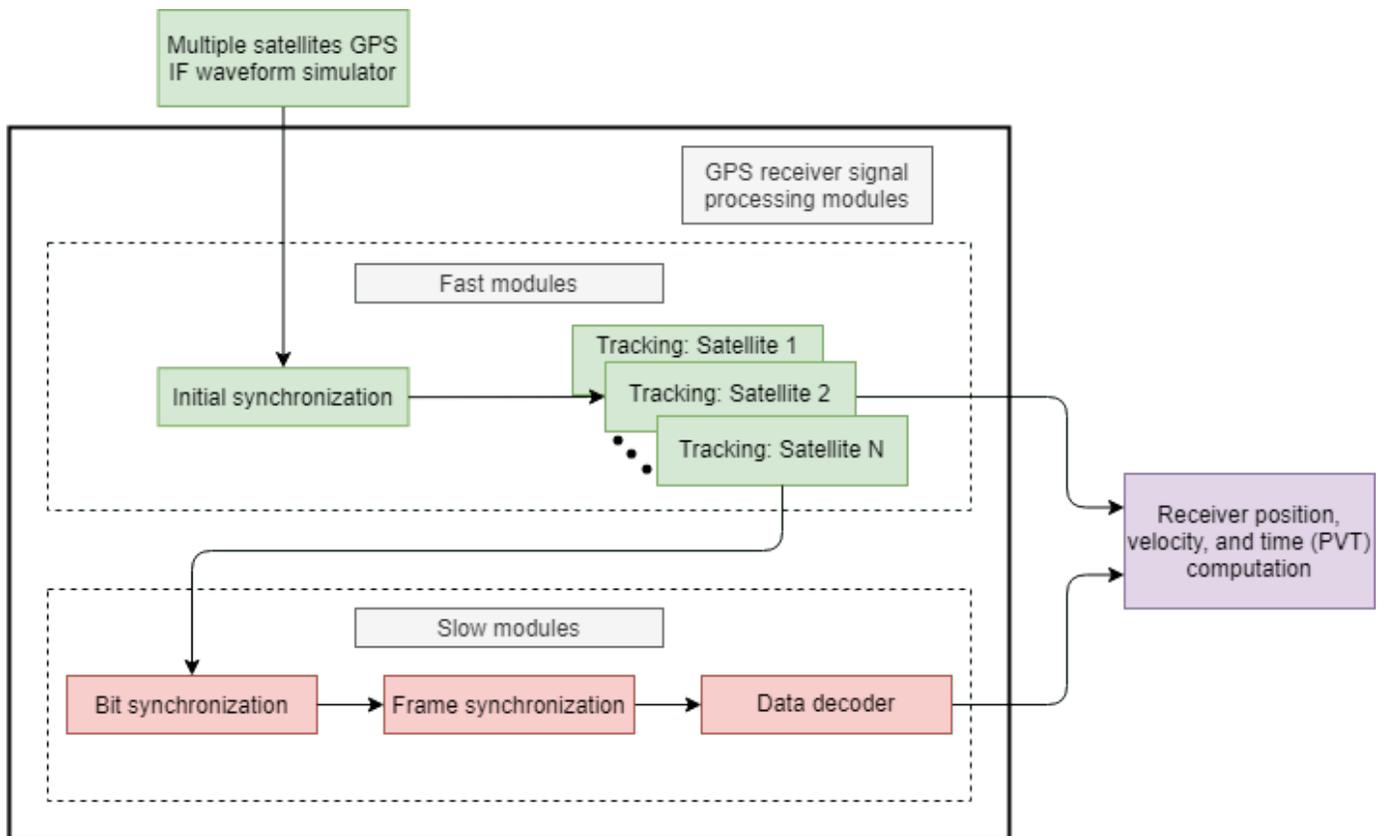
[7] W. E. Ryan, "An Introduction to LDPC Codes." in *Coding and Signal Processing for Magnetic Recoding Systems* (Bane Vasic, ed.). CRC Press, 2004.

GPS Receiver Acquisition and Tracking Using C/A-Code

This example shows how to generate a legacy Global Positioning System (GPS) intermediate frequency (IF) waveform from multiple satellites, add noise to the composite signal, perform initial synchronization, and track the code phase and carrier frequency of the available satellites detected from the initial synchronization operation. This multi-satellite IF waveform simulates the Doppler rate for each satellite. Tracking is done for each satellite independently. The acquisition and tracking shown in this example are on GPS signals that contain coarse acquisition codes (C/A-codes).

Introduction

The IF waveform in this example is generated based on the GPS baseband waveform. For more information on how to set the various parameters to generate a GPS baseband waveform, see “GPS Waveform Generation” on page 2-2 example. In addition to acquisition and tracking, a GPS receiver also performs bit synchronization, frame synchronization, and data decoding. The various blocks used for GPS receiver operations are shown in this figure. The green highlighted blocks are within the scope of this example.



The initial synchronization and tracking modules process data at a high rate (38.192 MHz) and are called fast modules [1] on page 4-138. Because the modules that perform bit synchronization, frame synchronization, and data decoding processes at a low rate (50 bits per second) of data, these modules are called slow modules. To get any meaningful data from these slow modules, significant data must be processed by the fast modules. Because the example focuses on the acquisition and

tracking algorithms in MATLAB™, not much data is processed to get any meaningful results from the slow modules.

This example is divided into three parts.

- 1 Configure Simulation Parameters on page 4-119 — Set various parameters to generate the waveform, configure the channel, and configure the GPS receiver.
- 2 Generate GPS IF Waveform on page 4-121 — Generate GPS IF waveform from multiple satellites, add delay, Doppler, and noise.
- 3 Initial Synchronization and Tracking on page 4-127 — Find the visible satellites and calculate coarse values of Doppler frequency offset and timing offset. Track the changing Doppler offset and code phase of the incoming signal.

Configure Simulation Parameters

Set all the configuration parameters that control the example output.

Set the general configuration parameters. The `ShowVisualizations` flag controls the visualizations in this example. Set it to true to see all the plots in the example. The `WriteWaveformToFile` flag controls writing the generated waveform into a file. If you want to test your receiver with the IF samples generated in this example, you can export the samples into a file by setting `WriteWaveformToFile` to true.

```
ShowVisualizations = true;
```

```
WriteWaveformToFile = true;
```

Transmitter Configuration

Specify the number of GPS satellites in the waveform.

```
numSat = 4;
```

Specify the pseudorandom noise (PRN) identification number (ID) for the satellites that are visible at the receiver. `PRNIDs` length must be equal to or greater than the number of specified satellites. If the length is greater than `numSat` property, then only first `numSat` values are used for simulation.

```
PRNIDs = [7; 11; 20; 28];
```

The entire GPS legacy navigation data consists of 25 frames with each frame having 1500 bits. This navigation data includes entire ephemeris, clock, and almanac information. Because generating the GPS waveform for the entire navigation data can take a lot of time, this example shows generating a waveform for only 20 bits of the navigation data. You can control generating the waveform for a specified number of data bits by using the property `NumNavDataBits`.

```
% Set this value to 1 to generate the waveform from the first bit of the
% navigation data. Set this to any other number to generate waveform from
% the point of interest. Starting from a random point in this example.
NavDataBitStartIndex = 1321;
```

```
% Set this value to control the number of navigation data bits in the
% generated waveform
NumNavDataBits = 20;
```

Define the received signal properties, such as IF center frequency and sampling rate. This example shows the signal at a center frequency of 10 MHz. You can configure this property to generate a

complex baseband signal by setting CenterFrequency property to 0. When you work with baseband data, you can reduce the SampleRate property to 10.23e6 so that example works much faster.

```
% Set the center frequency of the transmission waveform
CenterFrequency = 10e6; % In Hz
```

```
% Set the sample rate of the transmission waveform
SampleRate = 38.192e6; % In samples/sec
```

Channel Configuration

For each satellite, specify the signal-to-noise ratio (SNR). Each element in the vector represents an SNR value of the corresponding satellite PRN ID. The IS-GPS-200 [2] on page 4-138 standard specifies that for C/A-codes, the minimum received signal power is -158.5dBW (see table 3-Va in [2] on page 4-138). From the thermal noise power equation given below, noise power is calculated as -130dBW. Therefore, the minimum SNR at which the receiver must work is -28.5 dB (signal power (dB) - noise power (dB)). This example ignores the degradation due to interference.

Thermal noise power is given by the product of the Boltzmann constant k , receiver temperature in kelvin T , and the signal bandwidth BW . For legacy GPS signals with C/A-code tracking, typical signal bandwidth that is considered is 24MHz.

$$\begin{aligned} \text{Thermal noise power} &= k \times T \times BW \text{ Watts} \\ &= 1.38064852 \times 10^{-23} \times 300 \times 24 \times 10^6 \text{ Watts} \\ &= 9.94 \times 10^{-14} \text{ Watts} \\ &\approx -130 \text{ dBW} \end{aligned}$$

This example works for SNR values as low as -20dB.

```
SNRs = [-18; -18.5; -19.5; -19]; % In dB
```

Because each satellite is at different distance from the GPS receiver, the delay introduced on each signal is different for each satellite. Initialize the number of C/A-code chips delay for each satellite as a column vector. This delay value can be fractional because it is likely that the transmission time is not an integer multiple of the C/A-code chip duration of 0.9775 microseconds.

The delay values considered are much less than the actual values obtained from a live GPS signal. Smaller delay values result in faster simulation time and provide an opportunity to show the acquisition and tracking capabilities in detail.

```
sigdelay = [300.34; 587.21; 425.89; 312.88]; % Number of C/A-code chips delay
```

Because each satellite velocity and position is different, the Doppler shift and Doppler rate introduced for each satellite is different. The “Calculate Latency and Doppler in a Satellite Scenario” on page 1-49 example illustrates how latency and Doppler rate is calculated based on the satellite position and velocity. Because in the “Calculate Latency and Doppler in a Satellite Scenario” on page 1-49 example the change in the Doppler roughly follows a sinusoidal profile, in the current example, Doppler is modeled as a sinusoidally varying frequency offset [3] on page 4-138.

Initialize peak Doppler shift and Doppler rate for each satellite. This example can track Doppler shift from -10KHz to 10KHz.

```
peakDoppler = [3589; 4256; 8596; 9568]; % In Hz
dopplerRate = [1000; 500; 700; 500]; % In Hz/sec
```

Receiver Configuration

In GPS receivers, tracking algorithms tracks frequency, phase, and delay using frequency locked loops (FLLs), phase locked loops (PLLs), and delay locked loops (DLLs) respectively. A wider loop bandwidth enables fast tracking, but can lose lock at low SNRs. A narrower loop bandwidth performs well at low SNRs, but tracks phase, frequency, and delay changes more slowly. The interpretation of these properties is shown in Initial Synchronization and Tracking on page 4-127.

```

PLLNoiseBandwidth = 90; % In Hz
FLLNoiseBandwidth = 4; % In Hz
DLLNoiseBandwidth = 1; % In Hz

```

Configure the PLL integration time. As the PLL integration time increases, the phase tracking algorithm improves. You can set a maximum of 20 milliseconds as integration time because data transitions occur for every 20 milliseconds. Because the location of bit starting index is not known initially, having an integration time of 20 milliseconds causes the bit transition to be somewhere in the middle of the integrated signal. Because this causes the tracking loops to behave unexpectedly, initially, set the PLL integration time to 1 millisecond and increase appropriately when bit synchronization is complete. Because bit synchronization is not in the scope of this example, integration time is fixed to 1 millisecond. While increasing the PLL integration time, decrease the PLL noise bandwidth to make the receiver work at very low SNR values.

```

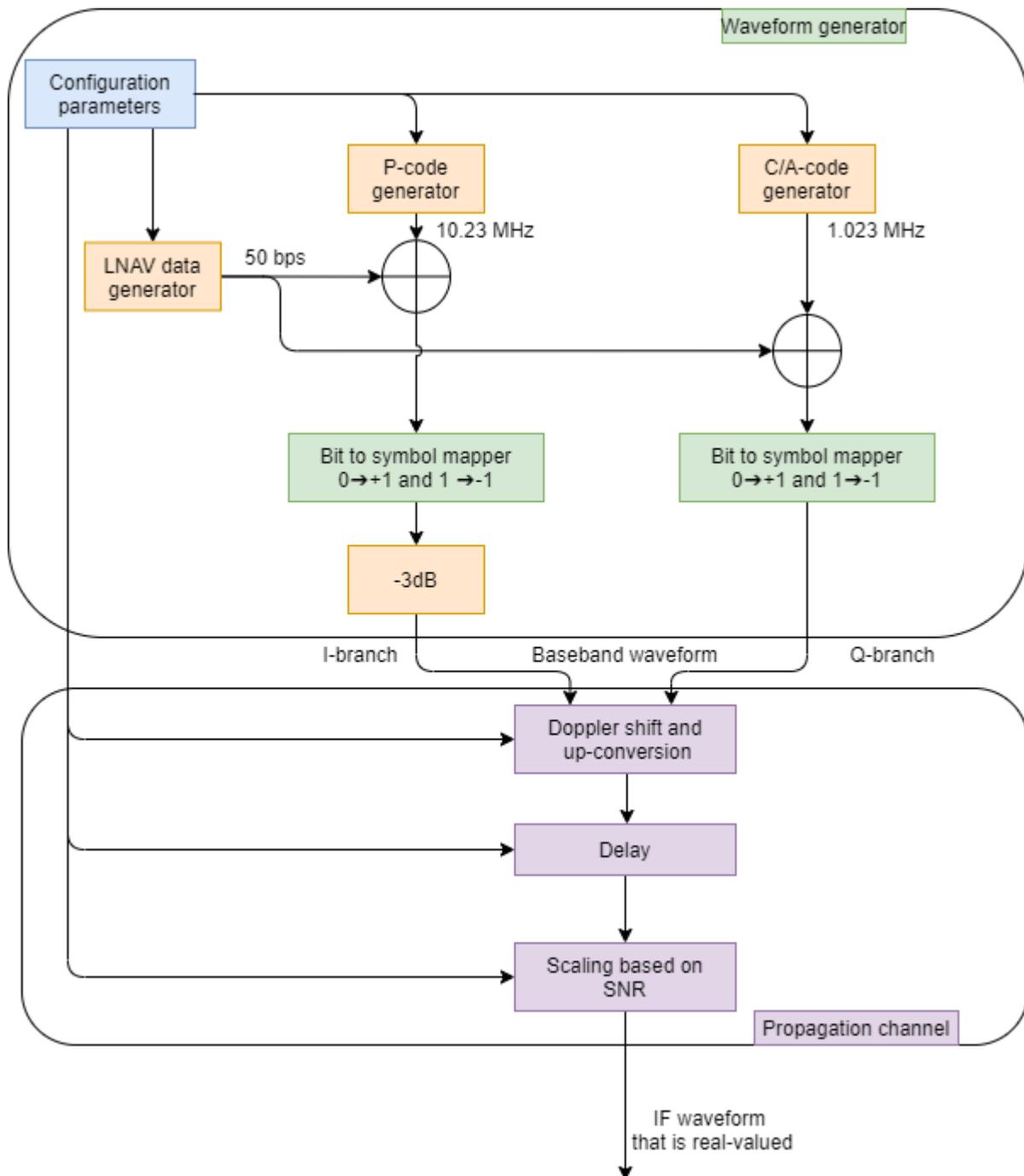
PLLIntegrationTime = 1e-3; % In seconds

```

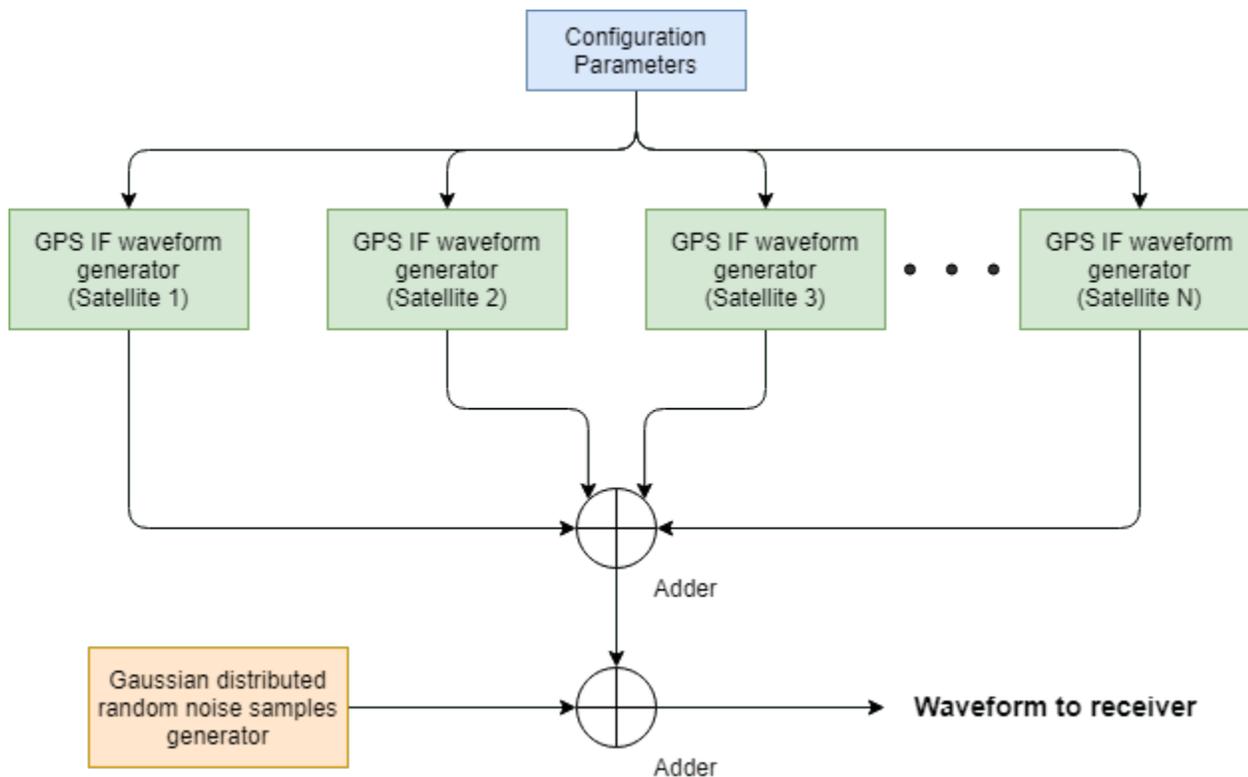
Generate GPS IF Waveform

The IF waveform generated contains precision code (P-code) on the in-phase branch and C/A-code on the quadrature-phase branch, transmitted on L1 frequency (1575.42MHz) from the GPS satellites. The details of the data that is transmitted is described in IS-GPS-200 standard [2] on page 4-138. The “GPS Waveform Generation” on page 2-2 describes how to convert GPS data to a baseband transmission waveform. This baseband waveform is up-converted to an IF waveform. Based on the specified SNR for each satellite, an IF waveform is scaled to match the received signal power. Also, model delays for each satellite signal independently and then add all signals to generate the composite IF waveform. To this composite IF waveform, add fixed amount of the additive white Gaussian noise (AWGN).

Generation of the legacy GPS IF waveform on the L1 frequency (1575.42 MHz) from one satellite involves the steps shown in this figure.



Add the scaled IF waveform from each satellite and then add a constant white Gaussian noise to the composite signal. Because each signal power is scaled as per the SNR, the effective SNR for each satellite signal is different. This figure shows the combining of the waveforms from multiple satellites. Each waveform generator in below figure consists of operations provided in the preceding figure.



Before waveform generation, calculate $k \times T \times BW$ noise power to scale each satellite signal appropriately.

```

k = 1.38064852e-23; % Boltzmann constant in Joules/Kelvin
T = 300;           % Room temperature in Kelvin
B = 24e6;          % Bandwidth in Hz
N = k*T*B;         % Thermal noise power in watts
  
```

Consider one extra navigation data bit for delay modeling. This will enable modeling of a maximum of 20 milliseconds of delay in the signal. Increase `numBitsForDelay` by an integer value to model higher delay values in the signal.

```
numBitsForDelay = 1;
```

To generate the legacy GPS transmission data from each satellite, initialize and use `HelperGPSNavigationConfig` object. This object contains information that is transmitted by a satellite. In this example, all parameters are kept constant for all satellites because the intention of this example is not to decode the data but to illustrate acquisition and tracking for which any random data suffices. For more details on this configuration object, see the “GPS Waveform Generation” on page 2-2 example.

```

resultsig = 0;
% Generate waveform from each satellite, scale and add based on SNR
for isat = 1:numSat
    % Initialize the legacy navigation (LNAV) configuration object
    lnavConfig = HelperGPSNavigationConfig("DataType","LNAV","PRNID",PRNIDs(isat));

    % Generate the navigation data bits from the configuration object
    lnavData = HelperGPSNAVDataEncode(lnavConfig);
  
```

```

% Configure the GPS waveform generation properties
t = lnavConfig.HOWTOW*6; % First get the initial time
% HOWTOW is an indication of next subframe starting point. Because each
% subframe is 300 bits long, 300 bits must be subtracted from the
% initial value to get the first subframe's starting value. This value
% can be negative as well. Each bit is of 20 millisecond duration and to
% get time elapsed for bits, bit index must be multiplied with 20e-3.
bitDuration = 20e-3; % In sec
pCodeRate = 10.23e6; % In Hz
numPChipsPerNavBit = bitDuration*pCodeRate;
navdatalen = length(lnavData);
offsetTime = mod(NavDataBitStartIndex-301,navdatalen)*bitDuration;
inittime = t + offsetTime;

% For modeling delay, get one extra navigation bit from previous bit
navBitIndices = mod(NavDataBitStartIndex+(-1*numBitsForDelay:(NumNavDataBits-1)),navdatalen)
navBitIndices(navBitIndices==0) = navdatalen;
navbits = lnavData(navBitIndices);
navdata = 1-2*navbits;
upSampledNavData = repelem(navdata,numPChipsPerNavBit,1);

% Generate P-code and C/A-code
pgen = gpsPCode("PRNID",PRNIDs(isat),"InitialTime",inittime, ...
    "OutputCodeLength",(NumNavDataBits+numBitsForDelay)*numPChipsPerNavBit);
pcode = 1-2*double(pgen());

% Reduce the power of I-branch signal by 3 dB as per IS-GPS-200 [2].
% See table 3-Va in [2].
isig = pcode/sqrt(2);

cacode = 1-2*double(gnssCACode(PRNIDs(isat),"GPS"));

numCACodeBlocks = (NumNavDataBits+numBitsForDelay)*bitDuration*1e3;
caCodeBlocks = repmat(cacode(:),numCACodeBlocks,1);

% Because C/A-code is 10 times slower than P-code, repeat each sample
% of C/A-code 10 times
qsig = repelem(caCodeBlocks,10,1);

% Generate the baseband waveform
gpsBBWaveform = (isig + 1j*qsig).*upSampledNavData;

% Initialize the number of samples per bit at IF
numSamplesPerBit = SampleRate*bitDuration;

% Introduce Doppler and up-convert the signal to IF
upconvert = HelperGPSUpConvert;
upconvert.PeakDoppler = peakDoppler(isat);
upconvert.DopplerRate = dopplerRate(isat);
upconvert.CenterFrequency = CenterFrequency;
upconvert.IFSampleRate = SampleRate;
gpsIFWaveform = upconvert(gpsBBWaveform);

% Get the number of samples for delay
caCodeRate = 1.023e6;
numDelaySamples = floor(sigdelay(isat)*upconvert.IFSampleRate/caCodeRate);

```

```

% Add delay to the signal by keeping samples of previous bit at the
% beginning of the signal
delayedSig = gpsIFWaveform(numSamplesPerBit-numDelaySamples+1:end);

% Remove the final samples to make all signals of equal length
delayedSig = delayedSig(1:end-numDelaySamples);

% Scale this delayed signal to appropriate power level
currentSNR = 10^(SNRs(isat)/10); % Convert to linear form
signalpower = currentSNR*N;
scaledsig = sqrt(signalpower)*delayedSig/rms(delayedSig);

% Get the composite signal by adding the current satellite signal
resultsig = resultsig + scaledsig;
end

% Add AWGN to the resultant IF waveform
numSamples = length(resultsig);

% For repeatable simulations, set the random number generator to default
rng default;
noisesig = wgn(numSamples,1,10*log10(N));
rxwaveform = resultsig + noisesig;

% Scale the received signal for having unit power
rxwaveform = rxwaveform/rms(rxwaveform);

if WriteWaveformToFile == 1
    bbWriter = comm.BasebandFileWriter('IFWaveform.bb', ...
        upconvert.IFSampleRate,upconvert.CenterFrequency);
    bbWriter(rxwaveform);
end

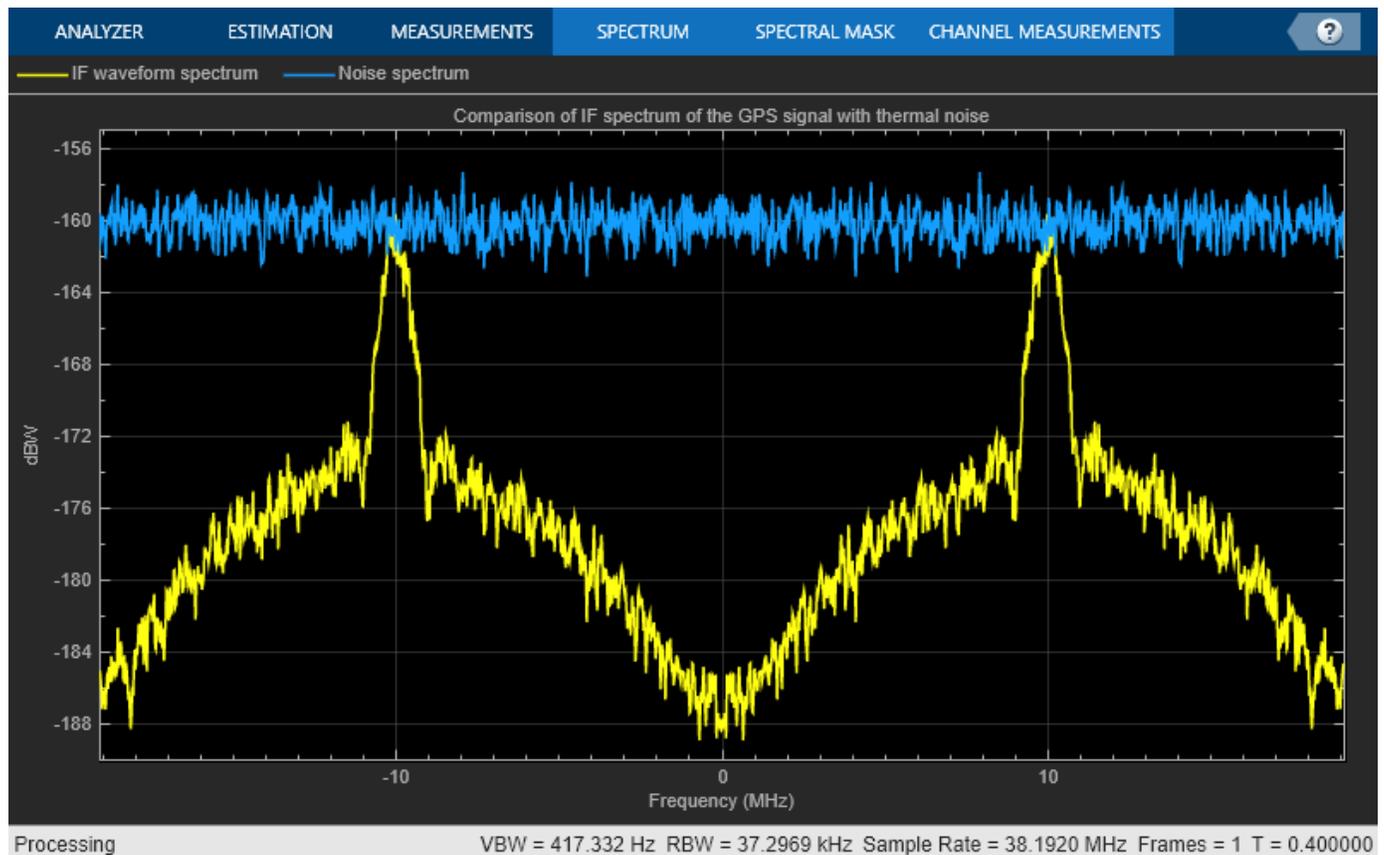
```

Visualize the spectrum of the GPS signal and noise signal. The signal is below the noise power.

```

if ShowVisualizations == 1
    ifscope = dsp.SpectrumAnalyzer('SampleRate',upconvert.IFSampleRate, ...
        'PlotAsTwoSidedSpectrum',true, ...
        'SpectrumType','Power', ...
        'AveragingMethod','Exponential', ...
        'SpectrumUnits','dBW', ...
        'Title', 'Comparison of IF spectrum of the GPS signal with thermal noise', ...
        'ShowLegend',true,'ChannelNames', ...
        {'IF waveform spectrum', 'Noise spectrum'}, ...
        'YLimits',[-190 -155]);
    ifscope([resultsig, noisesig]);
end

```

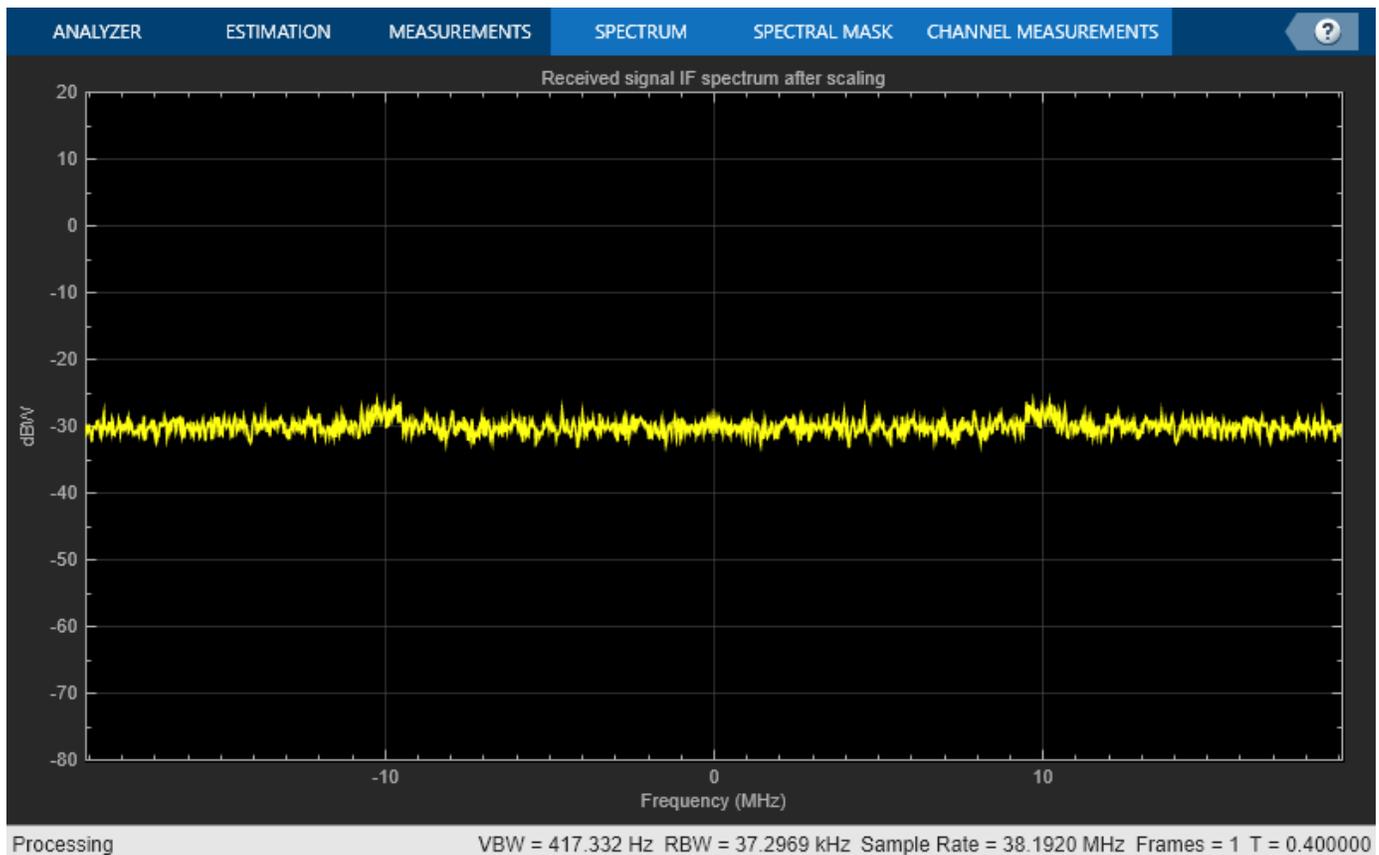


Visualize the received signal spectrum. The spectrum appears as if there is only noise. Because the received signal is scaled to have unit power, the spectrum is at unit power noise signal level.

```

if ShowVisualizations == 1
    rxscope = dsp.SpectrumAnalyzer('SampleRate',upconvert.IFSampleRate, ...
        'PlotAsTwoSidedSpectrum',true, ...
        'SpectrumType','Power', ...
        'AveragingMethod','Exponential', ...
        'SpectrumUnits','dBW', ...
        'Title', 'Received signal IF spectrum after scaling');
    rxscope(rxwaveform);
end

```



Initial Synchronization and Tracking

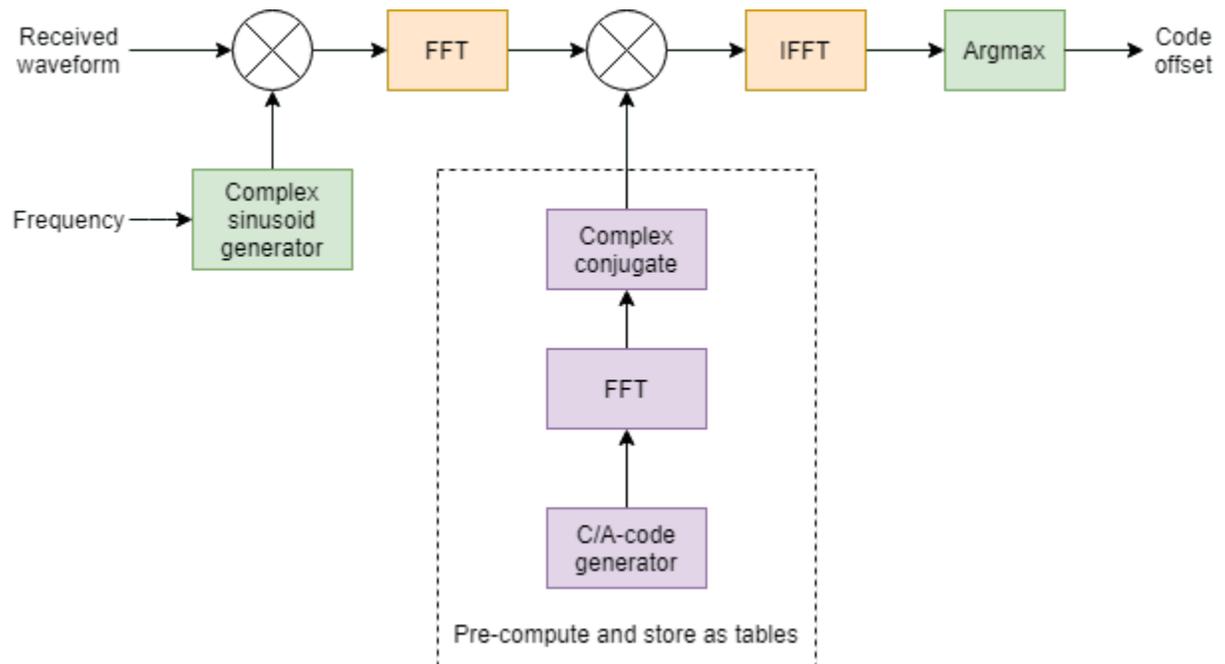
Typically, a receiver starts up in either of these modes [4] on page 4-138:

- Hot start — The receiver has information about its current location and has almanac information of the available satellites. This kind of startup happens when the receiver momentarily loses track of the visible satellite. In this mode, for a given satellite, initial synchronization is skipped and only tracking is performed.
- Warm start — The receiver knows its own approximate location as it did not move much after the receiver is turned off. Also, the receiver has approximate almanac information. This kind of startup typically happens when the receiver is turned off for few seconds to a few hours and the receiver did not move much after the receiver is turned off. In this mode, the search for all the available satellites is skipped. Initial synchronization for known visible satellites is performed and tracking is done on detected satellites.
- Cold start — The receiver does not have any information of its location before it is turned off and receiver might have traveled anywhere throughout the world. Receiver doesn't have any information about the almanac. All the satellites are searched in the initial synchronization and tracking is performed on the detected satellites.

Cold start takes the longest time to locate itself because there is no known information. The time taken by the receiver to locate itself is known as time to first fix (TTFF). This example assumes the receiver to be in cold start mode in the beginning and starts the search for all the possible 32 GPS satellites. Once the visible satellites are detected, the tracking operates on these satellites. This

example does not decode the almanac information and works with the satellites that are detected by initial synchronization.

Initial synchronization module detects all the visible satellites, estimates the coarse Doppler shift, and coarse time offset in the received signal from each satellite [5] on page 4-138. This example transforms the C/A-code into the frequency-domain by using the FFT algorithm and perform multiplication in frequency domain with the signal and then goes back into time-domain by using IFFT. This synchronization is one of the fastest initial synchronization algorithms [5] on page 4-138. The block diagram of this parallel code-phase search algorithm is shown in this figure. Though this figure shows the block diagram for the received IF waveform, the same block works for baseband waveform also (with a difference in the sampling rate and the FFT size).



To use this parallel-code phase algorithm, first initialize the initial synchronization object and configure its properties. The object performs some initial calculations once, then uses them multiple times during the execution time.

```
initialsync = HelperGPSInitialSynchronization;
```

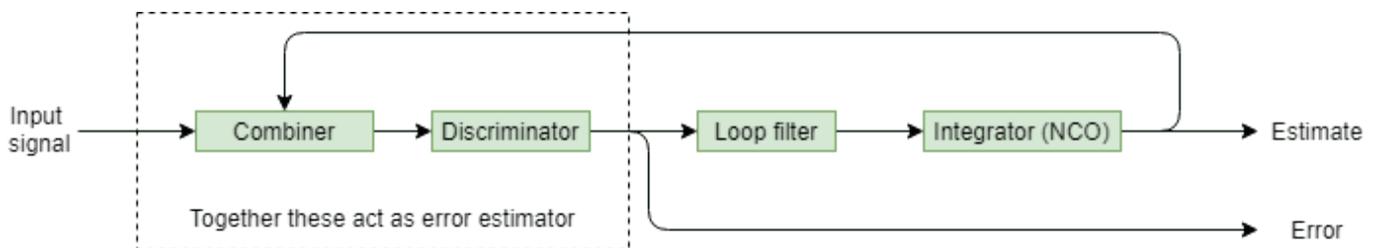
```
% Initially search all 32 satellites. After knowing which satellites are
% visible, update the PRNIDs to search as appropriate
initialsync.PRNIDsToSearch = 1:32;
initialsync.SampleRate = upconvert.IFSampleRate;
initialsync.CenterFrequency = CenterFrequency
```

```
initialsync =
```

```
  HelperGPSInitialSynchronization with properties:
```

```
    PRNIDsToSearch: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
    CenterFrequency: 10000000
    SampleRate: 38192000
    DetectionThresholdFactor: 1.2000
```

The tracking module compensates for the Doppler shift and code phase offset, remaining after initial synchronization. There are three parameters that must be tracked: carrier frequency, carrier phase, and code delay. Each one of these parameters are tracked using the feedback loops. Carrier frequency is tracked using FLL, phase is tracked using PLL, and code phase offset is tracked using DLL. The fundamental structure of any tracking loop (FLL, or PLL, or DLL) is the same and is depicted in the following figure. A combiner combines the incoming signal with that of reference signal generated in the feedback loop. The discriminator finds the error in the incoming signal with that of the reference signal, using the signal generated by the combiner. The tracking loops attempt to reduce this error to zero. This error can be any physical quantity such as frequency in FLL, or phase in PLL, or delay in DLL. This error is processed through a low-pass filter called loop filter. The loop filter reduces the noise in the discriminator output. The loop filter is fundamentally an integrator that accumulates the discriminator output to drive the error (discriminator output) to zero. The other module in the tracking loop is a numerically controlled oscillator (NCO). This is also a simple integrator, which accumulates the output from loop filter [1] on page 4-138.



The main role of the loop filter is to filter out the noise from the output of discriminator [1] on page 4-138. This filtering ensures any sudden changes in errors are reflected only after a sufficient amount of convergence time. This phenomenon can be observed in the results of the tracking module shown at the end of this example, where the discriminator output appears to be noisier than the tracking loop output. The amount of noise that must be allowed to be processed by the tracking loop is controlled by loop noise bandwidth (BW). This parameter differs for each tracking loop because if the loop bandwidth is more, then the loop filter allows more noise to pass through the tracking loop, and the convergence is fast. Similarly, a lesser noise bandwidth filter allows less noise to pass through the filter, and the convergence is slow. This table shows typical values used for noise bandwidth for different tracking loops [1] on page 4-138. An important factor for a tracking loop is the loop order. Higher order loops can tolerate sudden changes in the signal but are less stable. More stable lower order loops cannot operate in dynamic environments. This example uses a first order FLL, a second order PLL, and a first order DLL. The following table shows all these parameters for all the three loops used in this example. To configure these values, see the Receiver Configuration on page 4-121.

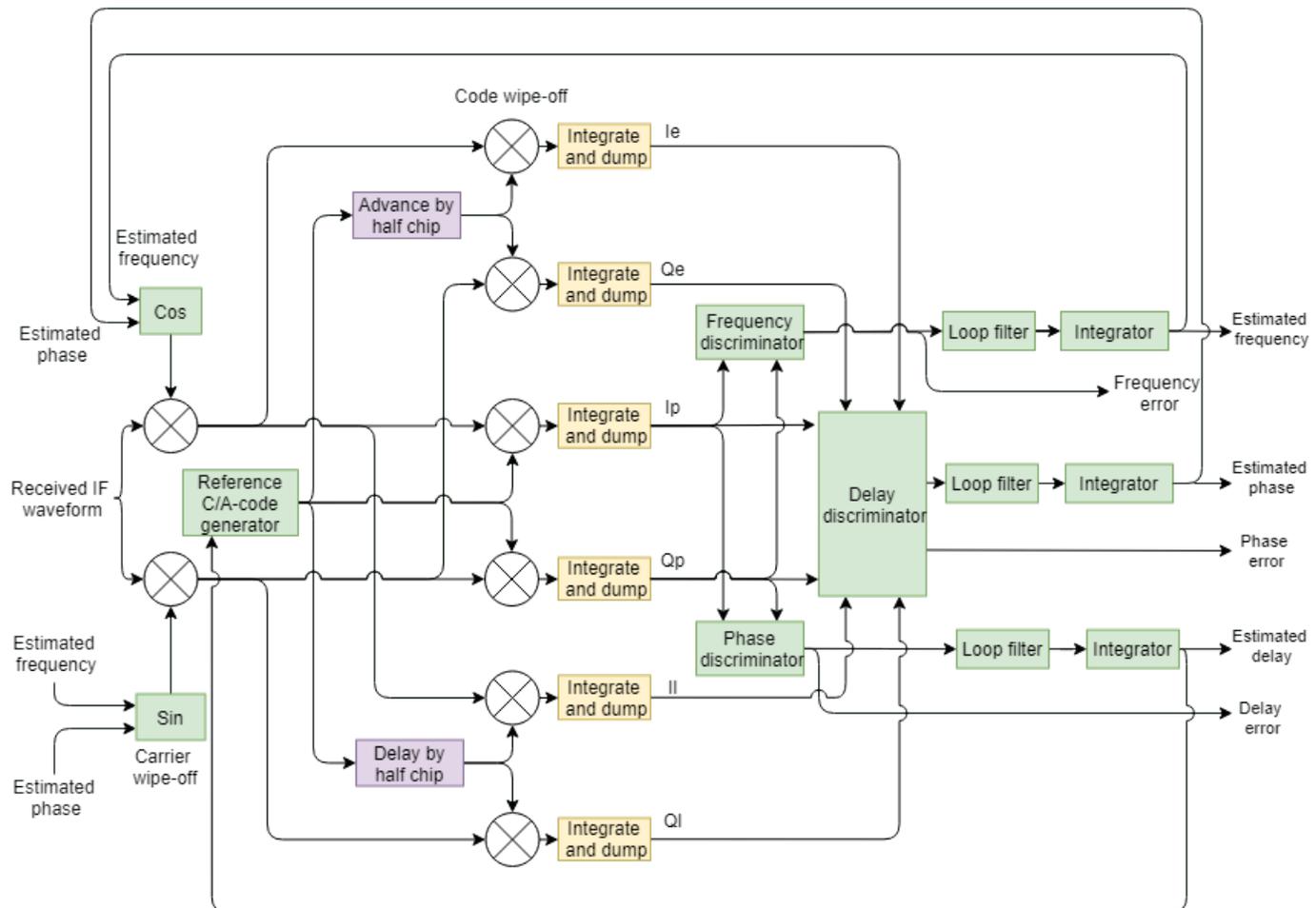
```
table({'4 quadrant atan2'; '2 quadrant atan'; '(E-L)/(2*(E+L))'},[4; 18; 1],[1;2;1], ...
    'VariableNames',{'Discriminator algorithm','Loop filter BW (Hz)','Loop order'}, ...
    'RowNames',{'FLL','PLL','DLL'})
```

```
ans=3x3 table
```

	Discriminator algorithm	Loop filter BW (Hz)	Loop order
FLL	{'4 quadrant atan2'}	4	1
PLL	{'2 quadrant atan' }	18	2
DLL	{'(E-L)/(2*(E+L))' }	1	1

This figure shows the various interconnections of tracking loops of a GPS receiver. In this figure, $I_p + jQ_p$ is the integrated signal from the prompt code, $I_e + jQ_e$ is the integrated signal from the half

chip early code, and $I_l + jQ_l$ is the integrated signal from the half chip late code. Prompt code is the signal that is multiplied by the time synchronized replica of the C/A-code. This time synchronized code is obtained once the lock is established in the loop. Early code is generated by cyclically advancing the prompt code. Late code is generated by cyclically delaying the prompt code by half chip duration. The FLL discriminator uses a four quadrant atan2 discriminator [1] on page 4-138, which calculates the frequency offset by evaluating the angle the phasor rotates in a given amount of integration time. The vector formed by $I_p + jQ_p$ is the phasor. If there is frequency offset, then this phasor rotates with an angular velocity. Use this angular velocity to compute the frequency offset. Because the C/A-code is placed on the quadrature branch, if there is a phase offset, then this phasor will not be at a location near to the y-axis. From the angle made by this phase with the y-axis, calculate the phase offset in the phase discriminator. The delay discriminator works on the principle that the correlation of prompt code must always be highest when compared with early or late codes. If the prompt code correlation is not highest, then the delay discriminator generates an appropriate error that corrects the delay in a way that correlation of prompt code is maximum [1] on page 4-138.



Carrier and code tracking must happen on the visible satellites. For each satellite, separate tracking loops are necessary. Based on the available satellites, the tracking loop objects are created dynamically. This code initializes several parameters required for tracking. Perform initial synchronization to find and track the visible satellites.

```

% Consider the data that is 1 millisecond long.
numSamples = SampleRate*PLLIntegrationTime;
[allRxInput,prevSamples] = buffer(rxwaveform,numSamples);
nFrames = size(allRxInput,2);
numdetectsat = 0;
PRNIDsToSearch = 1:32;

for iBuffer = 1:nFrames
    bufferWave = allRxInput(:,iBuffer);

    if iBuffer == 1
        % This example assumes a hot start for all the satellites once
        % initial synchronization is complete. Hence, perform initial
        % synchronization only once in this example. When decoding the
        % almanac data too, based on the available satellites, initial
        % synchronization can be performed for the visible satellites only.
        numSamplesForInitSync = SampleRate*1e-3; % Corresponding to 1 millisecond
        [y,corrval] = initialsinc(bufferWave(1:numSamplesForInitSync));
        PRNIDsToSearch = y(y(:,4)==1,1).';
        doppleroffsets = y(y(:,4)==1,2).';
        codephoffsets = y(y(:,4)==1,3).';

        % Ideally, this must be updated from the almanac. Because there is
        % no data decoding, this example relies on initial synchronization
        % results.
        initialsinc.PRNIDsToSearch = PRNIDsToSearch;

        numdetectsat = length(PRNIDsToSearch);

        disp(['The detected satellite PRN IDs: ' num2str(PRNIDsToSearch)])
        if ShowVisualizations == 1
            figure;
            mesh(-10e3:500:10e3, 0:size(corrval,1)-1, corrval(:,:,1));
            xlabel('Doppler Offset')
            ylabel('Code Phase Offset')
            zlabel('Correlation')
            msg = "Correlation Plot for PRN ID: " + num2str(PRNIDsToSearch(1));
            title(msg)
        end

        % Initialize all the properties which must be accumulated.
        accuph = zeros(nFrames,numdetectsat); % Each column represents data from a satellite
        accufqy = zeros(nFrames,numdetectsat);
        accufqyerr = zeros(nFrames,numdetectsat);
        accupherr = zeros(nFrames,numdetectsat);
        accuintegwave = zeros(nFrames,numdetectsat);
        accudelay = zeros(nFrames,numdetectsat);
        accudelayerr = zeros(nFrames,numdetectsat);

        % Create a cell array, where each element corresponds to a carrier
        % tracking object.
        carrierCodeTrack = cell(numdetectsat,1);

        % Update properties for each tracking loop
        for isat = 1:numdetectsat
            carrierCodeTrack{isat} = HelperGPSCACodeCarrierTracker;
            carrierCodeTrack{isat}.SampleRate = SampleRate;
            carrierCodeTrack{isat}.CenterFrequency = CenterFrequency;

```

```

        carrierCodeTrack{isat}.PLLNoiseBandwidth = PLLNoiseBandwidth;
        carrierCodeTrack{isat}.FLLNoiseBandwidth = FLLNoiseBandwidth;
        carrierCodeTrack{isat}.DLLNoiseBandwidth = DLLNoiseBandwidth;
        carrierCodeTrack{isat}.PLLIntegrationTime = PLLIntegrationTime*1e3;
        carrierCodeTrack{isat}.PRNID = PRNIDsToSearch(isat);
        carrierCodeTrack{isat}.InitialDopplerShift = doppleroffsets(isat);
        carrierCodeTrack{isat}.InitialCodePhaseOffset = codephoffsets(isat);
    end
end

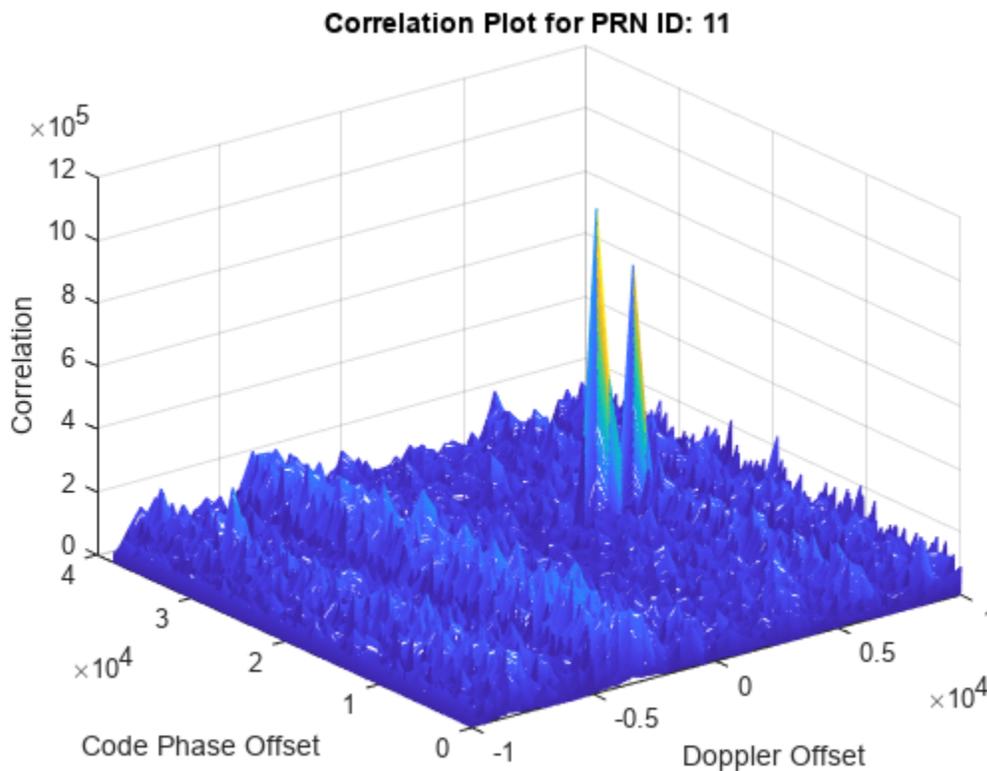
for isat = 1:numdetectsat % Perform tracking for each satellite

    [integwave,fqyerr,fqyoffset,pherr,phoffset,derr,dnco] = ...
        carrierCodeTrack{isat}(bufferWave);

    % Accumulate the values to see the results at the end
    accufqyerr(iBuffer,isat) = fqyerr;
    accufqy(iBuffer,isat) = fqyoffset;
    accupherr(iBuffer,isat) = pherr;
    accuph(iBuffer,isat) = phoffset;
    accuintegwave(iBuffer,isat) = sum(integwave);
    accudelayerr(iBuffer,isat) = derr;
    accudelay(iBuffer,isat) = dnco;
end
end

The detected satellite PRN IDs: 11 7 28 20

```



```
trackedSignal = accuintegwave; % Useful for further GPS receiver processing
```

The following plot shows the tracking loop results. The estimated phase error shows that it (second plot) is not converged until the frequency locked loop is converged (first plot). This behavior is expected because the error in frequency causes an error in phase. Also, the output of PLL and FLL seems to change with time because of the changing Doppler shift with time.

```
if ShowVisualizations == 1
    % for isat = 1:numdetectsat
    for isat = 1 % See tracking results of all the detected satellites by using above line
        groupTitle = ['Tracking Loop Results for Satellite PRN ID:', ...
            num2str(PRNIDsToSearch(isat))];

        figure

        % Plot the frequency discriminator output
        subplot(2,1,1)
        plot(accufqyerr(:,isat))
        xlabel('Milliseconds')
        ylabel('Frequency Error')
        title('Frequency Discriminator Output')

        % Plot the FLL output
        subplot(2,1,2)
        plot(accufqy(:,isat))
        xlabel('Milliseconds')
        ylabel('Estimated Frequency Offset')
        title('FLL Output')
        sgtitle(['FLL ' groupTitle])

        figure

        % Plot the phase discriminator output
        subplot(2,1,1)
        plot(accupherr(:,isat))
        xlabel('Milliseconds')
        ylabel('Phase Error')
        title('Phase Discriminator Output')

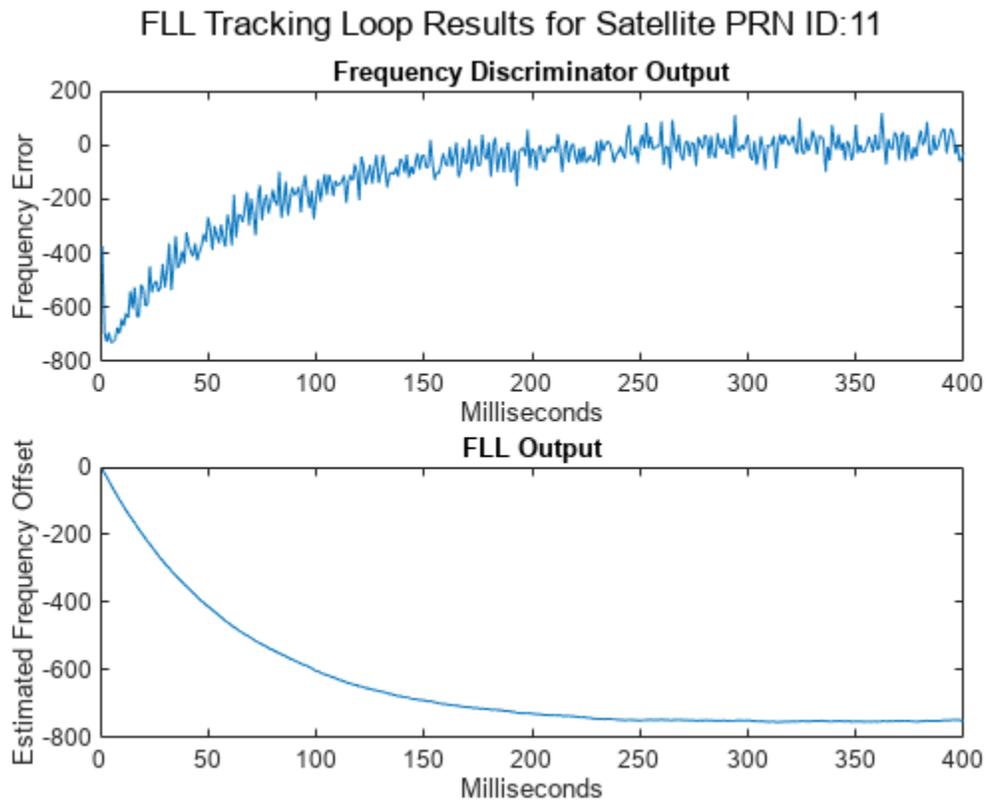
        % Plot the PLL output
        subplot(2,1,2)
        plot(accuph(:,isat))
        xlabel('Milliseconds')
        ylabel('Estimated Phase')
        title('PLL Output')
        sgtitle(['PLL ' groupTitle])

        figure

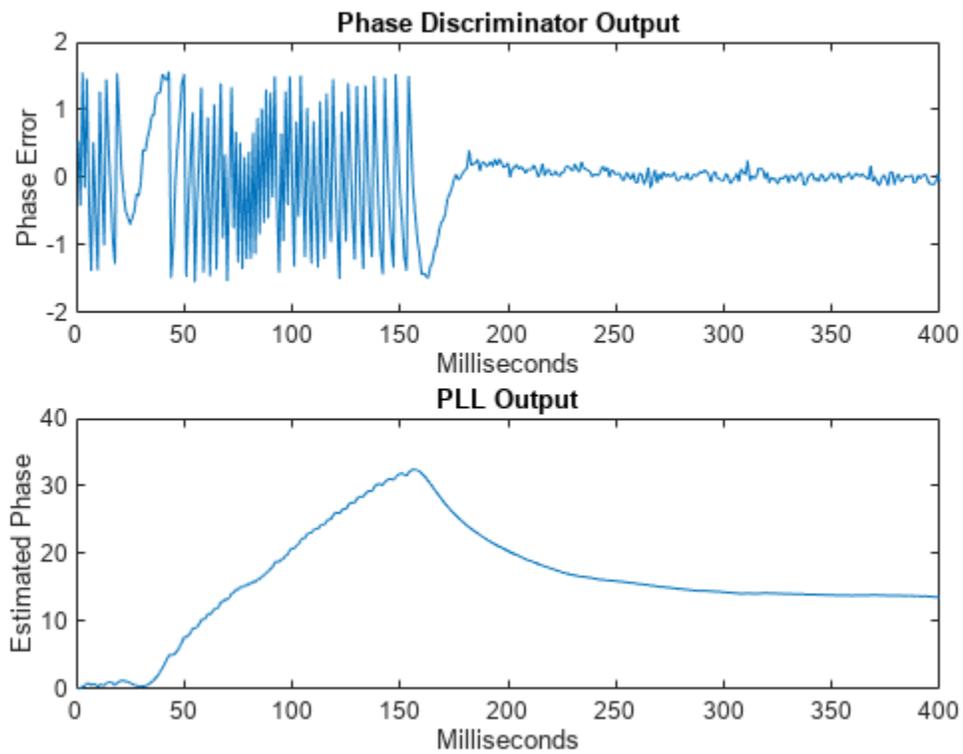
        % Plot the delay discriminator output
        subplot(2,1,1)
        plot(accudelayerr(:,isat))
        xlabel('Milliseconds')
        ylabel('Delay Error')
        title('Delay Discriminator Output')

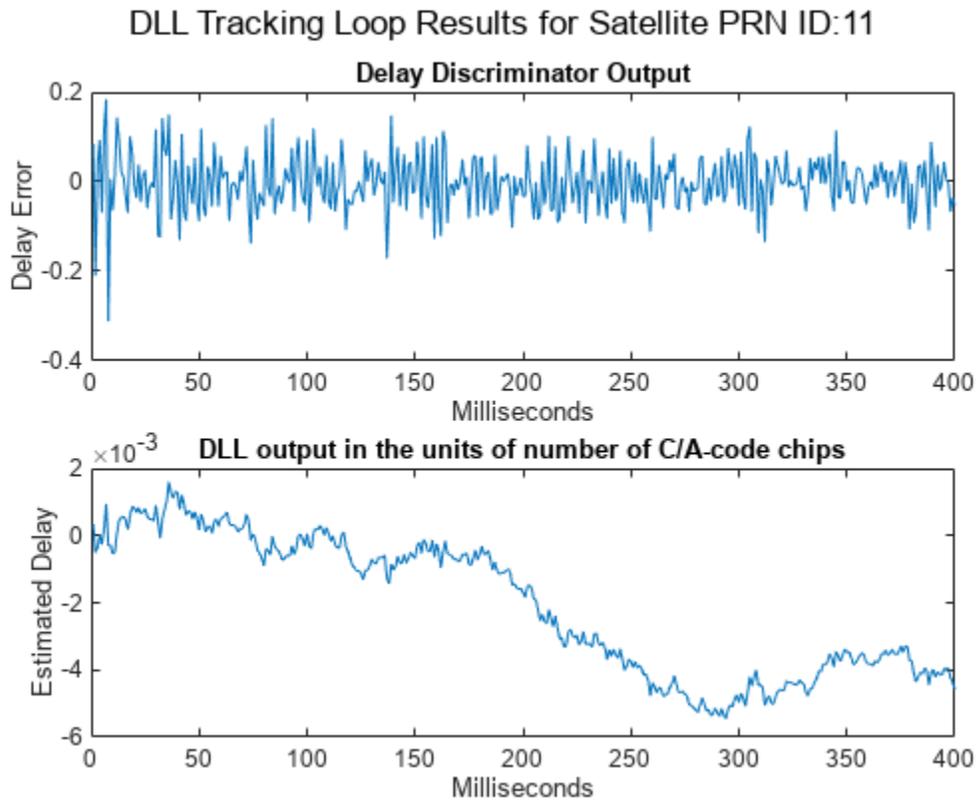
        % Plot the DLL output
        subplot(2,1,2)
```

```
plot(accudelay(:, isat))
xlabel('Milliseconds')
ylabel('Estimated Delay')
title('DLL output in the units of number of C/A-code chips')
sgtitle(['DLL ' groupTitle])
end
end
```



PLL Tracking Loop Results for Satellite PRN ID:11



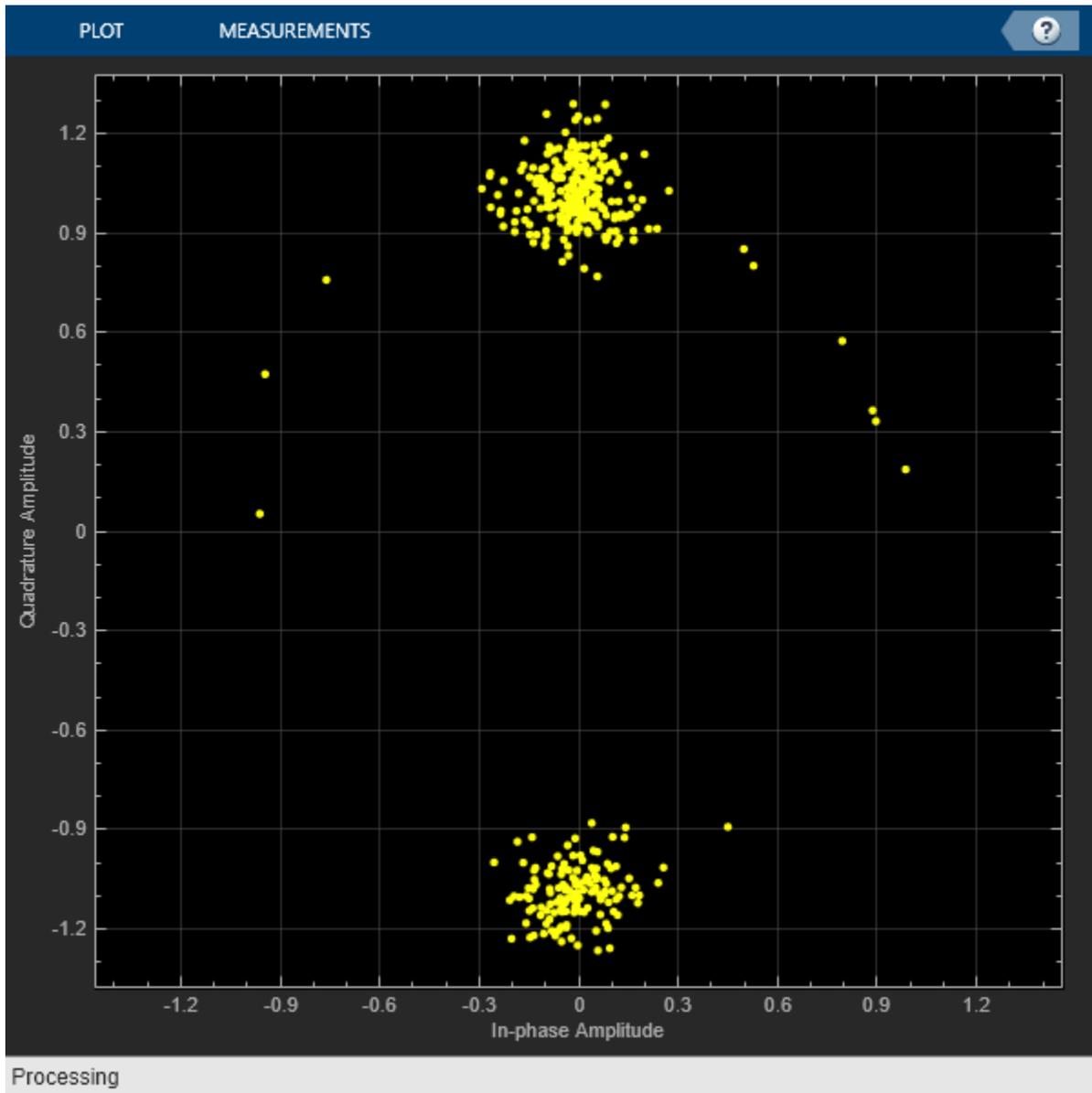


After tracking, plot the constellation of the signal.

```

if ShowVisualizations == 1
    rxconstellation = comm.ConstellationDiagram(1,"ShowReferenceConstellation",false);
    demodsamples = accuintegwave(301:end,:)/rms(accuintegwave(:));
    if ~isempty(demodsamples)
        rxconstellation(demodsamples(:));
    end
end
end

```



Processing

Further Exploration

This example shows the simulation for four GPS satellites. For a GPS-enabled smart phone, typically 6 GPS satellites are visible at any given point of time. You can increase the number of visible GPS satellites and observe the results. Initialize the Doppler, SNR, and delay appropriately.

This example uses hardcoded values for the delay and Doppler in each satellite signal. You can also model these two parameters as a real GPS satellite using “Scenario Generation and Visualization” in Satellite Communications Toolbox. Using these features, calculate the distance between transmitter and receiver and model the delay either using speed of light or any other delay models.

The receiver configuration properties on page 4-121 in this example are set to optimize for the conditions set in this example. You can update these receiver configuration parameters that suit the conditions you want the example to work. For example, for a much lower SNR than what is used in

this example, reduce the PLL noise bandwidth to any appropriate value that the tracking algorithm works properly. PLL noise bandwidth value can be reduced to as low as 18 in high noise conditions.

Appendix

This example uses these data and helper files:

- `gpsAlmanac.txt` — Almanac data file downloaded from Navcen website
- `HelperGPSAlmanac2Struct.m` — Convert text file of almanac to structure
- `HelperGPSCACodCarrierTracker.m` — Carrier frequency and C/A-code phase tracker
- `HelperGPSCEIConfig.m` — Create configuration object for GPS navigation data
- `HelperGPSInitialSynchronization.m` — GPS Initial synchronization
- `HelperGPSNAVDDataEncode.m` — Encode navigation data into bits from data that is in configuration object
- `HelperGPSNavigationConfig.m` — Create configuration object for GPS navigation data
- `HelperGPSUpConvert.m` — Model Doppler shift and up-convert the baseband signal to intermediate frequency

References

[1] Elliott D. Kaplan and C. Hegarty, eds., *Understanding GPS/GNSS: Principles and Applications*, Third edition, GNSS Technology and Applications Series (Boston ; London: Artech House, 2017).

[2] IS-GPS-200, Rev: L. NAVSTAR GPS Space Segment/Navigation User Segment Interfaces. May 14, 2020; Code Ident: 66RP1.

[3] CCSDS 130.11-G-1. Green Book. Issue 1. "SCCC—Summary of Definition and Performance." *Informational Report Concerning Space Data System Standards*. Washington, D.C.: CCSDS, April 2019.

[4] John W. Betz, *Engineering Satellite-Based Navigation and Timing: Global Navigation Satellite Systems, Signals, and Receivers* (John Wiley & Sons, Ltd, 2015).

[5] K. Borre, ed., *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*, Applied and Numerical Harmonic Analysis (Boston, Mass: Birkhäuser, 2007).

See Also

`gnssCACode`

Related Examples

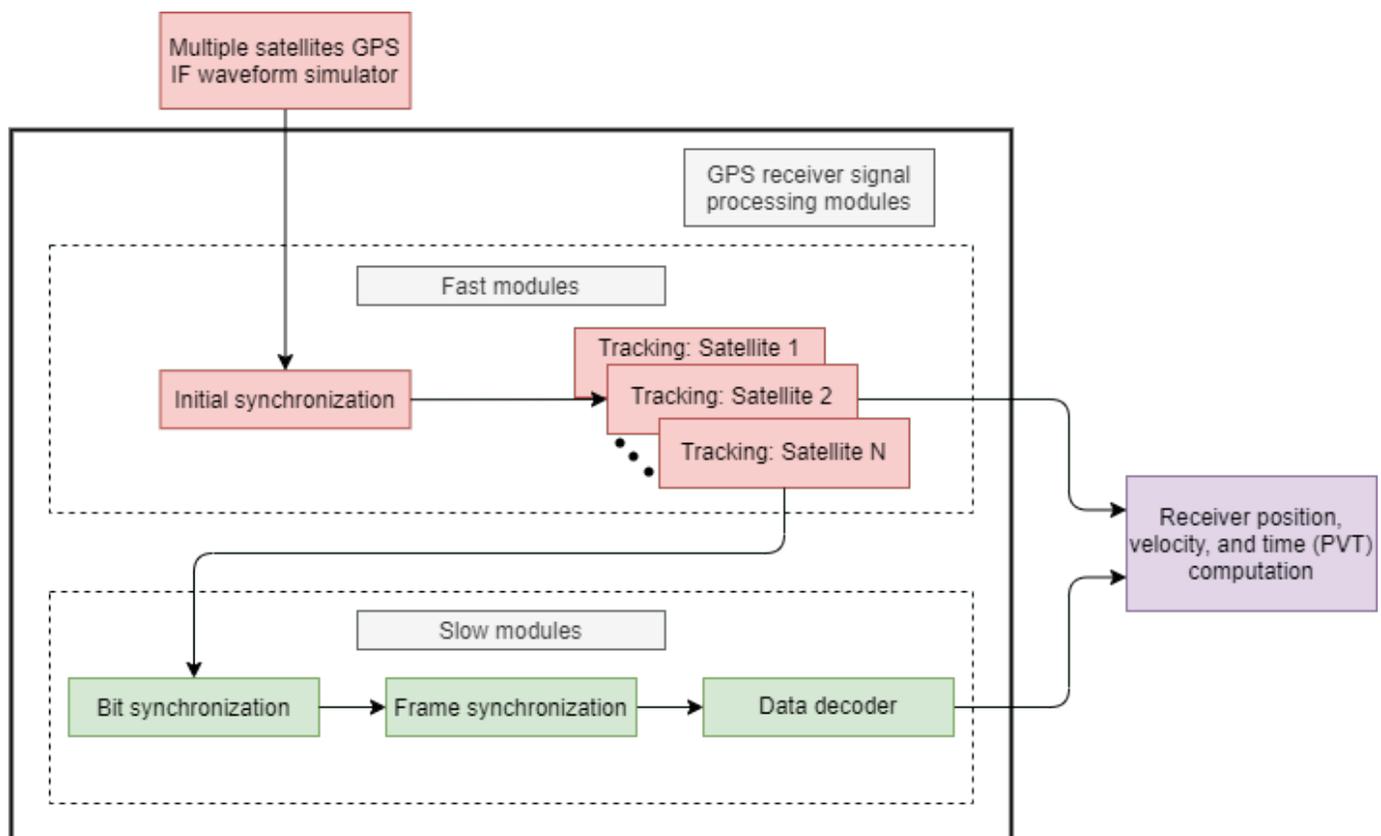
- “GPS Data Decode” on page 4-139
- “GPS Waveform Generation” on page 2-2

GPS Data Decode

This example shows how to perform bit and frame synchronization and decode the legacy GPS navigation (LNAV) data as per IS-GPS-200 [1] on page 4-146.

Introduction

This figure shows the various operations in a typical GPS receiver. The “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118 example describes the operations shown in red blocks in this figure. This example focuses on the green blocks. The GPS LNAV data transmits at a rate of 50 bits per second. In other words, each bit takes 20 milliseconds for transmission. The tracking module in the receiver tracks the phase shift, frequency offset, and delay between the visible GPS satellites and the receiver. For tracking, the received baseband signal is integrated to calculate the tracking metrics [2] on page 4-146. During tracking, if the integration happens over a bit transition, then the effective integrated value decreases because the bit values vary. Hence, the first step in data decoding is to find the exact bit transition boundary. Next, use the frame synchronization module to calculate the frame boundaries, which are required for navigation data decoding. Once the frame boundary is known, perform data decoding to get the received navigation data from a satellite. Data decoding from each tracking output channel must be done independently. This example shows how to decode data for a single satellite.



This example is divided into three parts.

- 1 Bit Synchronization on page 4-140 — Find the bit boundary in the output of the tracking loop.

- 2 Frame Synchronization on page 4-142 — Find the frame boundary in the demodulated bits.
- 3 Decode GPS LNAV Data on page 4-144 — Decode the bits to get the timing, ephemeris, almanac, and other data needed to estimate the receiver position.

To perform any of these steps, tracking results (time and frequency synchronized waveforms) are required. For this example, use 125 subframes of data from satellite PRN ID 7, which are stored in `trackedSignal.mat`, attached to this example as a supporting file. The samples are collected at a signal to noise ratio (SNR) of -23 dB. For more information on acquiring and tracking a GPS signal, see “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118.

Load the samples that the tracking loops output. Each sample in this array has a duration of 1 millisecond. The data type of the tracking loop output is floating-point double precision. To conserve storage, the MAT file stores the data in fixed point format with 1 sign bit, 1 bit for the integer part, and 6 bits for the fractional value. Convert the fixed-point data back into floating-point double format.

```
load trackedSignal.mat
trackedSignal = double(trackedSignal)/(2^6); % Convert fixed-point number to real value
```

Load the transmitted data bits to compare them with the decoded data bits. The “GPS Waveform Generation” on page 2-2 example shows how to generate these bits.

```
load transmittedBits.mat
```

Bit Synchronization

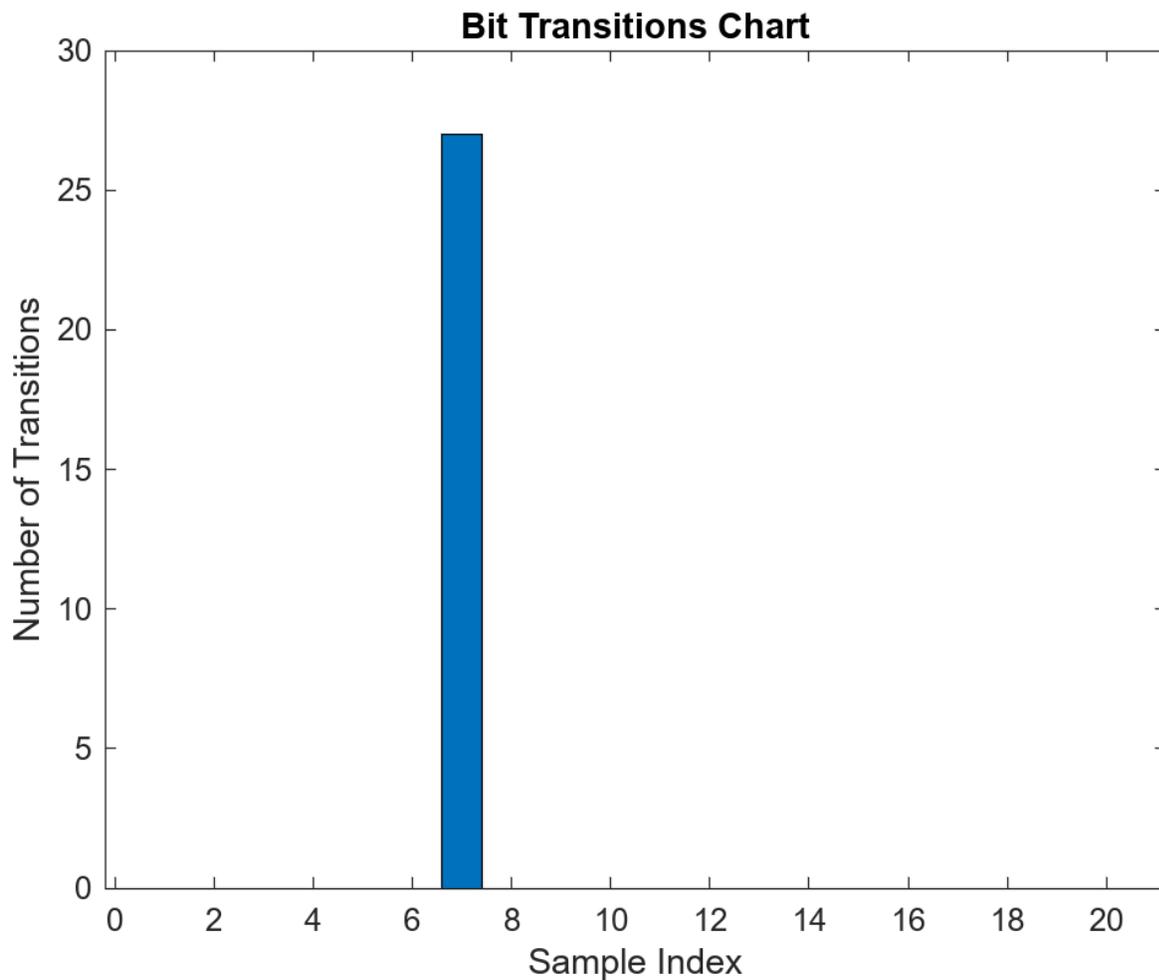
For the LNAV data, once tracking of the C/A-codes is complete, each code block with a C/A-code boundary of 1 millisecond duration is known. This 1 millisecond code block is integrated in the tracking module to get 1 sample. Each data bit consists of 20 such samples corresponding to 20 milliseconds. To get the bit boundary in each block of 20 milliseconds, flag the location (within this 20 millisecond block) that has the maximum number of transitions [2] on page 4-146.

Call the `gnssBitSynchronize` function to find where the bit transition occurs. Because C/A-codes sit on the quadrature branch of the waveform, consider only the imaginary part of the signal. The `gnssBitSynchronize` function estimates the data transition and gives the index of maximum number of transitions to indicate start of bit location within 20 samples. The `numAveragingBits` variable controls the length of the window to search for the data transitions. Because each data bit consists of 20 samples, when you set `numAveragingBits` to 100, a window of $20 \times 100 = 2000$ samples are used to decide the bit transition boundary.

```
numCACodeBlocksPerBit = 20;
numAveragingBits = 100;
numAveragingSamples = numCACodeBlocksPerBit*numAveragingBits;
[maxTransitionLocation, transitionValues] = ...
    gnssBitSynchronize(imag(trackedSignal(1:numAveragingSamples,1)),numCACodeBlocksPerBit);
maxTransitionLocation

maxTransitionLocation = 7

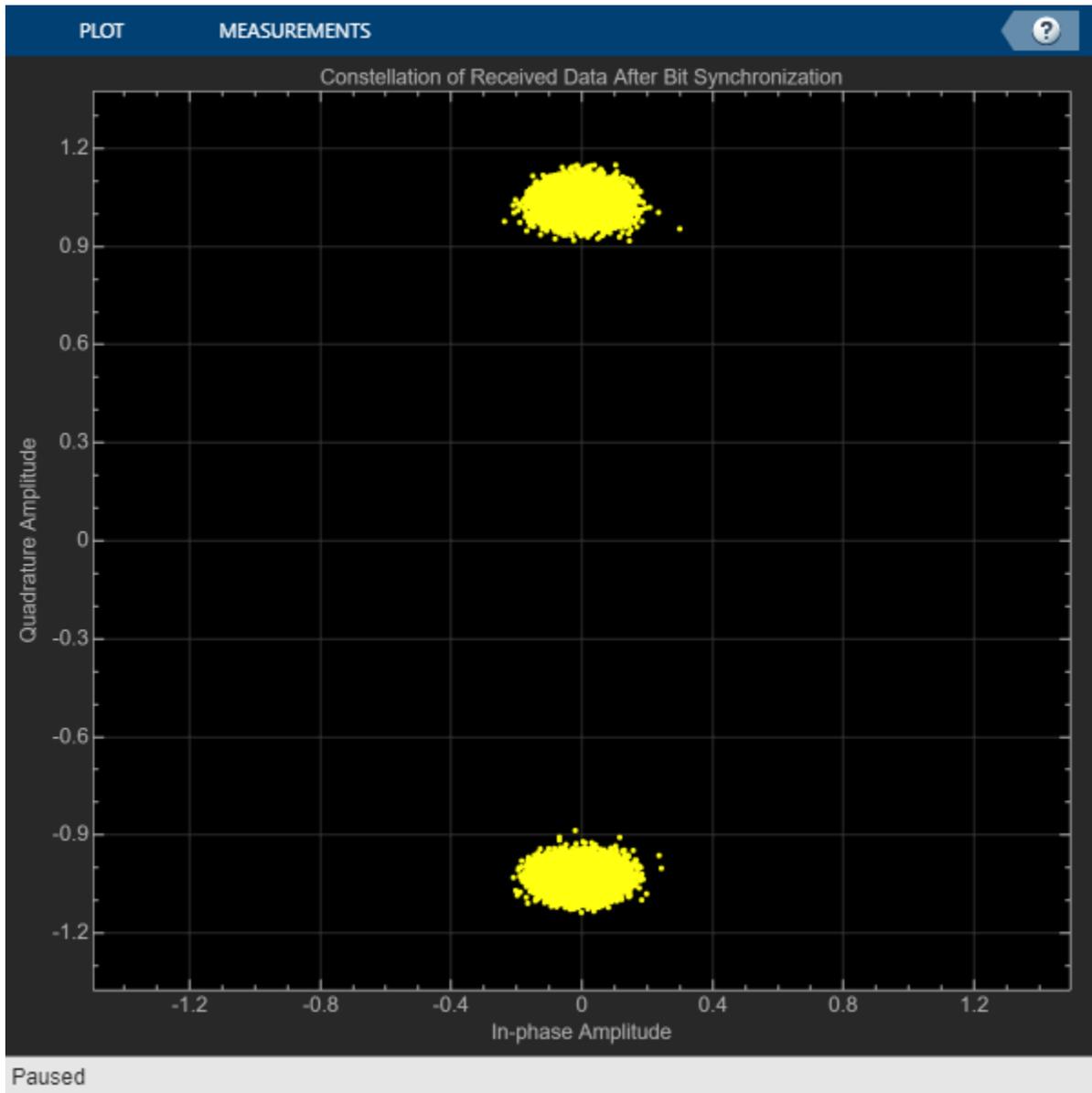
bar(transitionValues)
xlabel('Sample Index')
ylabel('Number of Transitions')
title('Bit Transitions Chart')
```



In the above figure, observe that the highest number of transitions occur at the location of bit transition.

Consider block of 20 samples from the location of the bit transition and integrate every 20 samples to generate soft log-likelihood ratios (LLRs). Get the bit values from the soft LLR values.

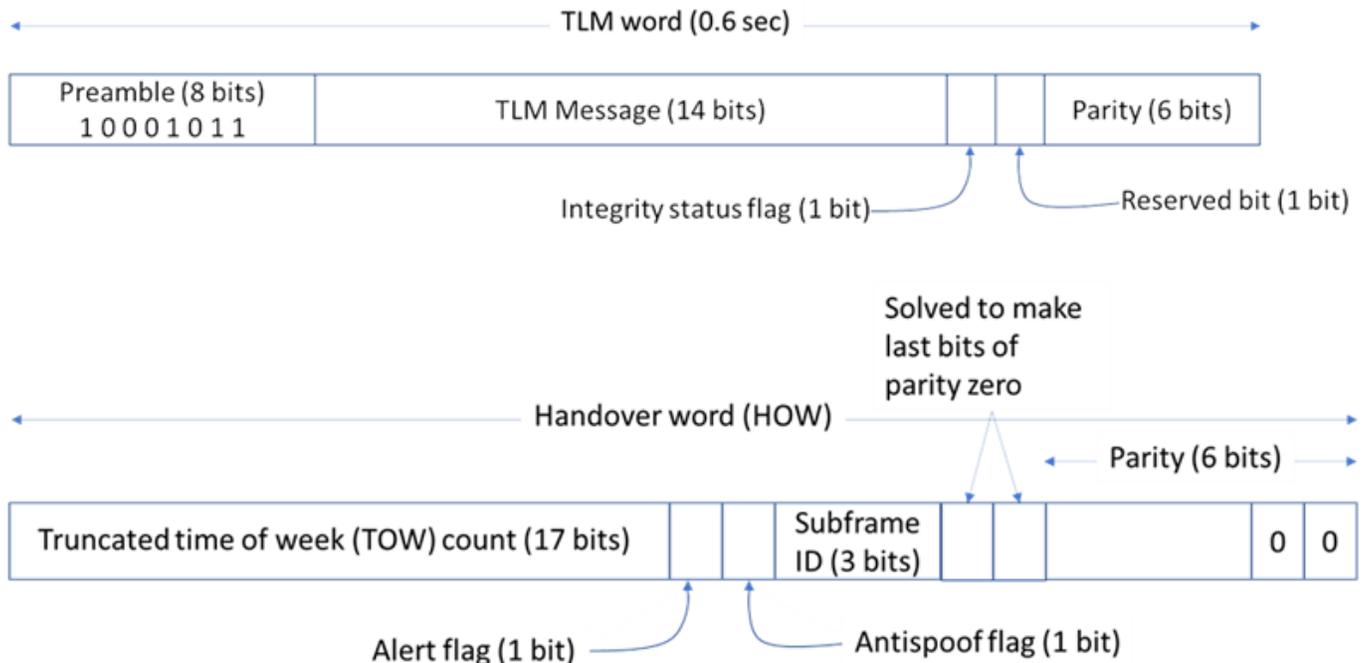
```
[samples, leftout] = buffer(trackedSignal(maxTransitionLocation:end, 1), numCACCodeBlocksPerBit);
softbits = mean(samples).';
bits = imag(softbits) < 0;
rxconstellation = comm.ConstellationDiagram(1, "ShowReferenceConstellation", false);
rxconstellation.Title = "Constellation of Received Data After Bit Synchronization";
rxconstellation(softbits(:))
```



Frame Synchronization

Frame synchronization, which determines the exact starting and ending points of a subframe. This information is necessary for the data decoder to process data.

Each subframe begins with a known 8-bit preamble. The frame synchronization module searches for this 8-bit sequence. Because an 8-bit sequence is small, the same sequence can occur somewhere else within the data. After detecting the 8-bit preamble, decode the first and second words of the subframe. If the parity checks pass, then decode the time of week present in the handover word [1] on page 4-146 and the subframe ID. If the time of week and the subframe ID agree, declare a successful frame boundary detection. Cycle slip in the tracking loops is possible, so continuous processing of the frame synchronization ensures that the decoder always works with an exact subframe [2] on page 4-146. This figure shows the telemetry word and handover word in a GPS LNAV data subframe.



Initialize the frame synchronization System object. This object returns the frame synchronized subframes in each column. Send all of the data through this object. This object also works when each subframe of data is passed through it iteratively.

```
framesync = HelperGPSLNAVFrameSynchronizer;
[syncidx, rxsubframes, subframeIDs] = framesync(bits);
syncidx
```

```
syncidx = 251
```

```
numSubframes = size(rxsubframes,2)
```

```
numSubframes = 123
```

```
subframeIDs
```

```
subframeIDs = 1x123
```

```
2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2
```

```
% Get the index of the synchronized subframes
```

```
allidx = 1:125; % There are 125 sub-frames in 25 frames of data
```

```
offsetidx = 0;
```

```
lostIdx = zeros(size(allidx)); % Pre-allocation
```

```
subframeIDs = [subframeIDs,0]; % To allow searching for all elements
```

```
for idx = allidx
```

```
    if mod(allidx(idx)-1,5)+1~=subframeIDs(idx-offsetidx)
```

```
        offsetidx = offsetidx + 1;
```

```
        lostIdx(offsetidx) = idx;
```

```
    end
```

```
end
```

```
lostIdx = lostIdx(1:offsetidx)
```

```
lostIdx = 1×2
    1    125

availableIdx = allidx(~ismember(allidx,lostIdx));
```

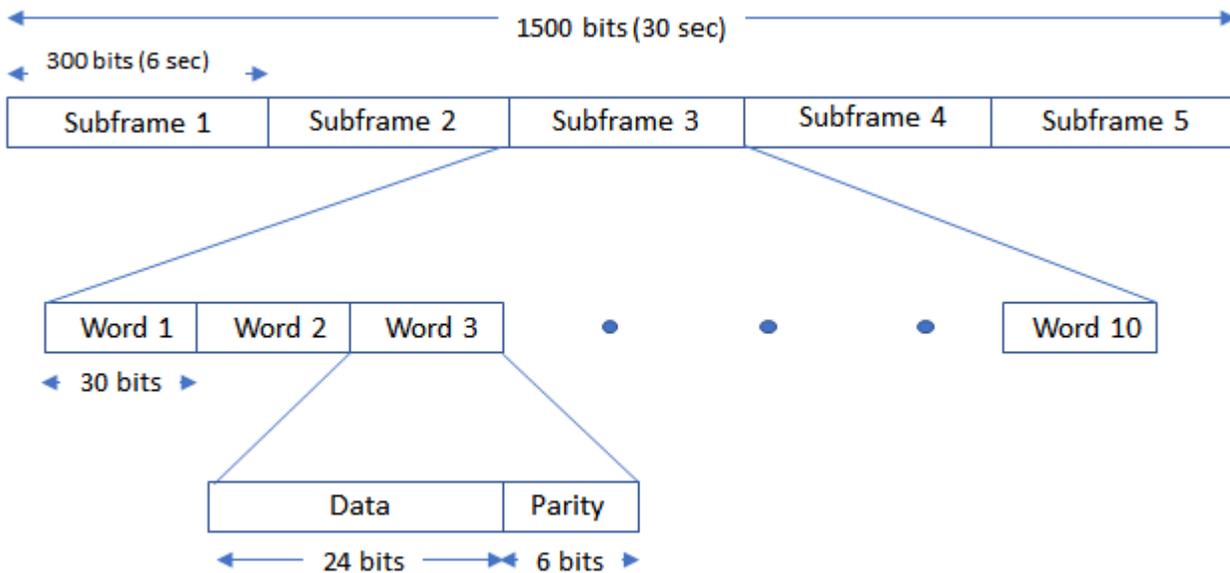
Check how many data bits differ between the frame synchronized data and transmitted data.

```
[transmittedSubframes,remainingBits] = ...
    buffer(transmittedBits,300); % Each subframe is of 300 bits
numBitsInError = nnz(transmittedSubframes(:,availableIdx) - rxsubframes)

numBitsInError = 0
```

Decode GPS LNAV Data

The GPS LNAV data is transmitted in 1500-bit length frames. Each frame consists of five subframes of 300 bits each. Because the data rate is 50 bits per second, transmitting each subframe takes 6 seconds and transmitting each frame takes 30 seconds. Each subframe consists of 10 words with 30 bits (24 data bits and 6 parity bits) in each word. The GPS data contains information regarding the clock and the position of the satellites. This figure shows the frame structure of the LNAV data.



This example processes each subframe independently.

```
cfg = struct;
accuParityChecks = zeros(numSubframes,10);
for isubframe = 1:numSubframes
    [cfg,parityChecks] = HelperGPSLNAVDataDecode(rxsubframes(:,isubframe),cfg);
    accuParityChecks(isubframe,:) = parityChecks;
end
numWordsInError = numel(accuParityChecks) - nnz(accuParityChecks)

numWordsInError = 0
```

The transmitted configuration properties and the decoded configuration properties must have same values. These are the decoded configuration parameters.

cfg

cfg = struct with fields:

```

        DataType: "LNAV"
        Preamble: 139
        TLMMMessage: 0
        HOWTOW: 124
        AntiSpoofFlag: 0
        CodesOnL2: "P-code"
        L2PDataFlag: 0
        CEIDataset: [1x1 struct]
        AgeOfDataOffset: 0
        NMCTAvailabilityIndicator: 0
        NMCTERD: [30x1 double]
        Ionosphere: [1x1 struct]
        UTC: [1x1 struct]
        TextMessage: 'This content is part o'
        SubframeID: 4
        DataID: [2x1 double]
        SVPageID: 63
        Almanac: [1x1 struct]
        ReferenceTimeUTCData: 0
        TimeDataReferenceWeekNumber: 101
        PastLeapSecondCount: 18
        LeapSecondReferenceWeekNumber: 101
        LeapSecondReferenceDayNumber: 1
        FutureLeapSecondCount: 18
        AlmanacSVHealth: [0 0 0 0 0 0 0 0]

```

cfg.CEIDataset

ans = struct with fields:

```

        SVHealth: 0
        IssueOfDataClock: 0
        URAID: 0
        WeekNumber: 101
        GroupDelayDifferential: 0
        SVClockCorrectionCoefficients: [3x1 double]
        ReferenceTimeOfClock: 0
        SemiMajorAxisLength: 2.6560e+04
        MeanMotionDifference: 0
        FitIntervalFlag: 0
        Eccentricity: 0.0200
        MeanAnomaly: 0
        ReferenceTimeOfEphemeris: 0
        HarmonicCorrectionTerms: [6x1 double]
        IssueOfDataEphemeris: 0
        IntegrityStatusFlag: 0
        ArgumentOfPerigee: -0.5200
        RateOfRightAscension: 0
        LongitudeOfAscendingNode: -0.8400
        Inclination: 0.3000
        InclinationRate: 0
        AlertFlag: 0

```

Further Exploration

This example performs frame synchronization, considering all subframes at a time. In practice, each subframe processing occurs in a loop whenever any new data is received. You can perform frame synchronization on every subframe independently. Also, this example does not update the bit synchronization status back to the tracking loop. You can also use the bit synchronization value to increase the phase locked loop (PLL) integration time to enhance the PLL performance at low SNR values.

Appendix

This example uses these helper files:

- `HelperGPSLNAVDataDecode.m` — Decode the LNAV GPS data.
- `HelperGPSLNAVFrameSynchronizer.m` — Perform frame synchronization on the demodulated data.
- `HelperGPSLNAVWordDecode.m` — Decode each word of a subframe.

This example uses these data files:

- `trackedSignal.mat` — Output of the tracking loop in the “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118 example for satellite PRNID 7
- `transmittedBits.mat` — Transmitted bits from the GPS satellite PRNID 7, consisting of 125 subframes of data

References

[1] IS-GPS-200, Rev: L. *NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*. May 14, 2020; Code Ident: 66RP1.

[2] Elliott D. Kaplan and C. Hegarty, eds., *Understanding GPS/GNSS: Principles and Applications*, Third edition, GNSS Technology and Applications Series (Boston ; London: Artech House, 2017).

See Also

`gnssCACode` | `gnssBitSynchronize`

Related Examples

- “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118
- “GPS Waveform Generation” on page 2-2

NR NTN PDSCH Throughput

This example shows how to measure the physical downlink shared channel (PDSCH) throughput of a 5G New Radio (NR) link in a non-terrestrial network (NTN) channel, as defined by the 3GPP NR standard. The example implements the PDSCH and downlink shared channel (DL-SCH). The transmitter model includes PDSCH demodulation reference signals (DM-RS) and PDSCH phase tracking reference signals (PT-RS). The example supports NTN narrowband and NTN tapped delay line (TDL) propagation channels.

Introduction

This example measures the PDSCH throughput of a 5G link, as defined by the 3GPP NR standard [1 on page 4-162], [2 on page 4-162], [3 on page 4-162], [4 on page 4-162].

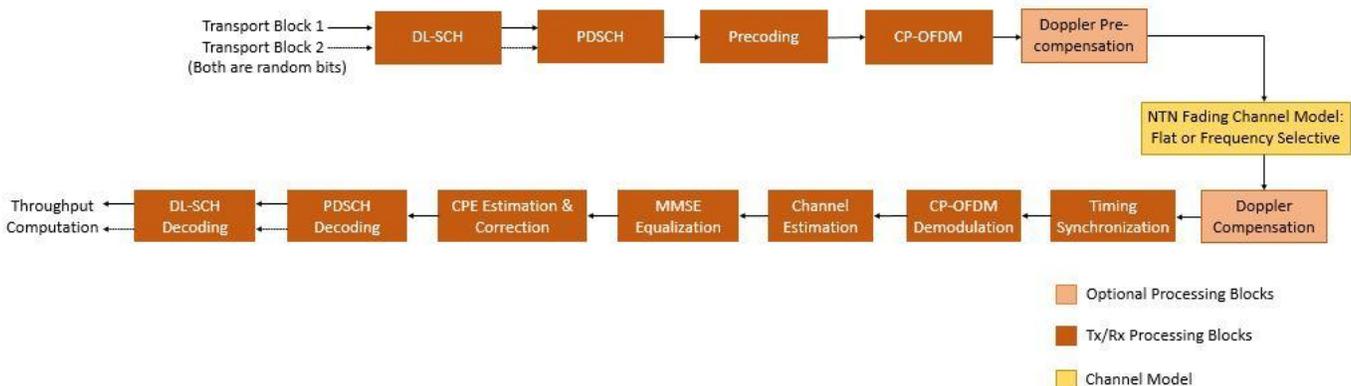
The example models these 5G NR features:

- DL-SCH transport channel coding
- Multiple codewords, dependent on the number of layers
- PDSCH, PDSCH DM-RS, and PDSCH PT-RS generation
- Variable subcarrier spacing and frame numerologies
- Normal and extended cyclic prefix
- NTN narrowband and NTN TDL propagation channel models

Other features of the simulation are:

- PDSCH precoding using singular value decomposition (SVD)
- Cyclic prefix orthogonal frequency division multiplexing (CP-OFDM) modulation
- Slot-wise and non-slot-wise PDSCH and DM-RS mapping
- Timing synchronization and channel estimation
- A single bandwidth part (BWP) across the whole carrier

The figure shows the implemented processing chain. For clarity, DM-RS and PT-RS generation are omitted.



For a more detailed explanation of the steps implemented in this example, see “Model 5G NR Communication Links” (5G Toolbox) and “DL-SCH and PDSCH Transmit and Receive Processing Chain” (5G Toolbox).

This example supports wideband and subband precoding. You determine the precoding matrix by using SVD and averaging the channel estimate across all PDSCH PRBs either in the allocation (for wideband precoding) or in the subband.

To reduce the total simulation time, you can use Parallel Computing Toolbox™ to execute the SNR points of the SNR loop in parallel.

Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10 ms frames. By default, the example uses 2 frames, but, a large number of 10 ms frames is necessary to produce meaningful throughput results. Set the SNR points to simulate. The SNR for each layer is defined per resource element (RE), and it includes the effect of signal and noise across all antennas. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” (5G Toolbox).

```
simParameters = struct;           % Clear simParameters variable to contain
                                  % all key simulation parameters
simParameters.NFrames = 2;       % Number of 10 ms frames
simParameters.SNRIn = 5:2:15;    % SNR range (dB)
```

Set the `displaySimulationInformation` variable to `true` to display information about the throughput simulation at each SNR point.

```
displaySimulationInformation =  ;
```

Doppler Compensation Configuration

The example supports two Doppler compensation configurations: one at transmitter end and the other at receiver end. For compensation at transmitter end, use the `DopplerPreCompensator` field. Set the `DopplerPreCompensator` field to `true` to account for Doppler due to satellite movement by applying Doppler pre-compensation to the transmitted waveform. For compensation at receiver end, use the `RxDopplerCompensator` field. Set the `RxDopplerCompensator` field to `true` to estimate and compensate the Doppler shift for the received waveform.

```
simParameters.DopplerPreCompensator =  ;
simParameters.RxDopplerCompensator =  ;
```

Carrier and PDSCH Configuration

Set the key parameters of the simulation. These parameters include:

- Bandwidth in resource blocks (RBs)
- Subcarrier spacing (SCS) in kHz: 15, 30, 60, or 120
- Cyclic prefix length (CP): normal or extended
- Cell identity
- Number of transmit and receive antennas

Also create a substructure containing the DL-SCH and PDSCH parameters, including:

- Target code rate
- Allocated resource blocks (PRBSet)
- Modulation scheme: 'QPSK', '16QAM', '64QAM', or '256QAM'
- Number of layers
- PDSCH mapping type
- DM-RS configuration parameters
- PT-RS configuration parameters

```

% Set waveform type and PDSCH numerology (SCS and CP type)
simParameters.Carrier = nrCarrierConfig;

simParameters.Carrier.SubcarrierSpacing = 30;

simParameters.Carrier.CyclicPrefix = Normal;
% Bandwidth in number of RBs (11 RBs at 30 kHz SCS for 5 MHz bandwidth)
simParameters.Carrier.NSizeGrid = 11;
% Physical layer cell identity
simParameters.Carrier.NCellID = 1;

% PDSCH/DL-SCH parameters
% This PDSCH definition is the basis for all PDSCH transmissions in the
% throughput simulation
simParameters.PDSCH = nrPDSCHConfig;
% This structure is to hold additional simulation parameters for the DL-SCH
% and PDSCH
simParameters.PDSCHExtension = struct();

% Define PDSCH time-frequency resource allocation per slot to be full grid
% (single full grid BWP)
% PDSCH PRB allocation
simParameters.PDSCH.PRBSet = 0:simParameters.Carrier.NSizeGrid-1;
% Starting symbol and number of symbols of each PDSCH allocation
simParameters.PDSCH.SymbolAllocation = [0,simParameters.Carrier.SymbolsPerSlot];

simParameters.PDSCH.MappingType = A;

% Scrambling identifiers
simParameters.PDSCH.NID = simParameters.Carrier.NCellID;

simParameters.PDSCH.RNTI = 1;

% PDSCH resource block mapping (TS 38.211 Section 7.3.1.6)
simParameters.PDSCH.VRBTtoPRBInterleaving = 0;
simParameters.PDSCH.VRBBundleSize = 4;

% Define the number of transmission layers to be used
simParameters.PDSCH.NumLayers = 1;

% Define codeword modulation and target coding rate
% The number of codewords is directly dependent on the number of layers so
% ensure that layers are set first before getting the codeword number
if simParameters.PDSCH.NumCodewords > 1

```

```

% Multicodeword transmission (when number of layers being > 4)
simParameters.PDSCH.Modulation = { 16QAM , 16QAM };
% Code rate used to calculate transport block sizes
simParameters.PDSCHExtension.TargetCodeRate = [490 490]/1024;
else
simParameters.PDSCH.Modulation = 16QAM ;
% Code rate used to calculate transport block size
simParameters.PDSCHExtension.TargetCodeRate = 490/1024;
end

% DM-RS and antenna port configuration (TS 38.211 Section 7.4.1.1)
simParameters.PDSCH.DMRS.DMRSPortSet = 0:simParameters.PDSCH.NumLayers-1;
simParameters.PDSCH.DMRS.DMRSTypeAPosition = 2 ;
simParameters.PDSCH.DMRS.DMRSLength = 1 ;
simParameters.PDSCH.DMRS.DMRSAdditionalPosition = 2 ;
simParameters.PDSCH.DMRS.DMRSConfigurationType = 2 ;
simParameters.PDSCH.DMRS.NumCDMGroupsWithoutData = 1 ;
simParameters.PDSCH.DMRS.NIDNSCID = 1 ;
simParameters.PDSCH.DMRS.NSCID = 0 ;

% PT-RS configuration (TS 38.211 Section 7.4.1.2)
simParameters.PDSCH.EnablePTRS = 0 ;
simParameters.PDSCH.PTRS.TimeDensity = 1 ;
simParameters.PDSCH.PTRS.FrequencyDensity = 2 ;
simParameters.PDSCH.PTRS.REOffset = 00 ;
% PT-RS antenna port, subset of DM-RS port set. Empty corresponds to lowest
% DM-RS port number
simParameters.PDSCH.PTRS.PTRSPortSet = [];

% Reserved PRB patterns, if required (for CORESETs, forward compatibility etc)
simParameters.PDSCH.ReservedPRB{1}.SymbolSet = []; % Reserved PDSCH symbols
simParameters.PDSCH.ReservedPRB{1}.PRBSet = []; % Reserved PDSCH PRBs
simParameters.PDSCH.ReservedPRB{1}.Period = []; % Periodicity of reserved resources

% Additional simulation and DL-SCH related parameters
% PDSCH PRB bundling (TS 38.214 Section 5.1.2.3)
simParameters.PDSCHExtension.PRGBundleSize = []; % 2, 4, or [] to signify "wideband"
% Rate matching or transport block size (TBS) parameters
% Set PDSCH rate matching overhead for TBS (Xoh) to 6 when PT-RS is enabled, otherwise 0
simParameters.PDSCHExtension.XOverhead = 6*simParameters.PDSCH.EnablePTRS;
% Hybrid automatic repeat request (HARQ) parameters
% Number of parallel HARQ processes to use
simParameters.PDSCHExtension.NHARQProcesses = 1;
% Enable retransmissions for each process, using redundancy version (RV) sequence [0,2,3,1]
simParameters.PDSCHExtension.EnableHARQ = false;
% LDPC decoder parameters

```

```

% Available algorithms: 'Belief propagation', 'Layered belief propagation',
%                       'Normalized min-sum', 'Offset min-sum'
simParameters.PDSCHExtension.LDPCDecodingAlgorithm = ;
simParameters.PDSCHExtension.MaximumLDPCIterationCount = 6;

% Define the overall transmission antenna geometry at end-points
% For NTN narrowband channel, only single-input-single-output (SISO)
% transmission is allowed
% Number of PDSCH transmission antennas (1,2,4,8,16,32,64,128,256,512,1024) >= NumLayers
simParameters.NumTransmitAntennas = 1;
if simParameters.PDSCH.NumCodewords > 1 % Multi-codeword transmission
    % Number of UE receive antennas (even number >= NumLayers)
    simParameters.NumReceiveAntennas = 8;
else
    % Number of UE receive antennas (1 or even number >= NumLayers)
    simParameters.NumReceiveAntennas = 1;
end
end

```

Get information about the baseband waveform after the OFDM modulation step.

```

waveformInfo = nrOFDMInfo(simParameters.Carrier);

```

Propagation Channel Model Construction

Create the channel model object for the simulation. Both the NTN narrowband and NTN TDL channel models are supported [5 on page 4-162], [6 on page 4-162]. For more information on how to model NTN narrowband and NTN TDL channels, see “Model NR NTN Channel” on page 3-8.

```

% Define the general NTN propagation channel parameters
% Set the NTN channel type to Narrowband for an NTN narrowband channel and
% set the NTN channel type to TDL for a NTN TDL channel.
simParameters.NTNChannelType = ;

% Set the parameters common to both NTN narrowband and NTN TDL channels
simParameters.CarrierFrequency = 2e9;           % Carrier frequency (in Hz)
simParameters.ElevationAngle = 50;             % Elevation angle (in degrees)
simParameters.MobileSpeed = 3*1000/3600;      % Speed of mobile terminal (in m/s)
simParameters.SatelliteAltitude = 600000;     % Satellite altitude (in m)
simParameters.SatelliteSpeed = 7562.2;        % Satellite speed (in m/s)
simParameters.SampleRate = waveformInfo.SampleRate;
simParameters.RandomStream = 'mt19937ar with seed';
simParameters.Seed = 73;

% Set the following fields for NTN narrowband channel
if strcmpi(simParameters.NTNChannelType, 'Narrowband')
    simParameters.Environment = 'Urban';
    simParameters.AzimuthOrientation = 0;
end
end

% Set the following fields for NTN TDL channel
if strcmpi(simParameters.NTNChannelType, 'TDL')
    simParameters.DelayProfile = 'NTN-TDL-A';
    simParameters.DelaySpread = 30e-9;
end
end

% Cross-check the PDSCH layering against the channel geometry

```

```

validateNumLayers(simParameters);

% Set up the NTN channel
ntnChan = HelperSetupNTNChannel(simParameters);

% Get the maximum number of delayed samples due to a channel multipath
% component. The maximum number of delayed samples is calculated from the
% channel path with the maximum delay and the implementation delay of the
% channel filter. This number of delay samples is required later to flush
% the channel filter to obtain the received signal.
chInfo = info(ntnChan.BaseChannel);
maxChDelay = ceil(max(chInfo.PathDelays*ntnChan.BaseChannel.SampleRate)) + ...
    chInfo.ChannelFilterDelay;

```

Processing Loop

To determine the throughput at each SNR point, analyze the PDSCH data for each transmission instance using these steps.

- 1 **Generate the transport block** — Get the transport block size for each codeword depending on the PDSCH configuration. Generate new transport block(s) for each transmission depending on the transmission status for given HARQ process.
- 2 **Generate the resource grid** — The `nrDLSCH` (5G Toolbox) System object performs transport channel coding. The object operates on the input transport block. The `nrPDSCH` (5G Toolbox) function modulates the encoded data bits. Apply an implementation-specific multiple-input-multiple-output (MIMO) precoding to the modulated symbols. Map these modulated symbols along with reference signal to the resource grid.
- 3 **Generate the waveform** — The `nrOFDMModulate` (5G Toolbox) function OFDM-modulates the generated grid to get the time-domain waveform.
- 4 **Apply Doppler pre-compensation** — Apply the Doppler shift due to satellite movement to the generated waveform to pre-compensate the channel induced satellite Doppler shift.
- 5 **Model and apply a noisy channel** — Pass the generated waveform through an NTN narrowband or NTN TDL fading channel to get the faded waveform. Add additive white Gaussian noise (AWGN) to the faded waveform. Define the SNR for each layer based on the RE and receive antenna. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” (5G Toolbox).
- 6 **Apply Doppler compensation** — Estimate the Doppler shift in the received waveform and compensate the Doppler shift.
- 7 **Perform synchronization and OFDM demodulation** — For timing synchronization, the received waveform is correlated with the PDSCH DM-RS. The `nrOFDMDemodulate` (5G Toolbox) function then OFDM-demodulates the synchronized signal.
- 8 **Perform channel estimation** — For channel estimation, PDSCH DM-RS is used.
- 9 **Perform equalization and CPE compensation** — The `nrEqualizeMMSE` (5G Toolbox) function equalizes the received PDSCH REs. Use the PT-RS symbols to estimate the common phase error (CPE) and then correct the error in each OFDM symbol within the range of the reference PT-RS OFDM symbols.
- 10 **Calculate precoding matrix** — Use SVD to generate the precoding matrix W for the next transmission.
- 11 **Decode the PDSCH** — Demodulate and descramble the equalized PDSCH symbols, along with a noise estimate using the `nrPDSCHDecode` (5G Toolbox) function to obtain an estimate of the received codewords.

- 12 Decode the DL-SCH** — Pass the decoded soft bits through the `nrDLSCHDecoder` (5G Toolbox) System object. The object decodes the codeword and returns the block cyclic redundancy check (CRC) error. Update the HARQ process with the CRC error. This example determines the throughput of the PDSCH link using the CRC error.

```

% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(simParameters.SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(simParameters.SNRIn),1);

% Set up RV sequence for all HARQ processes
if simParameters.PDSCHExtension.EnableHARQ
    % In the final report of RAN WG1 meeting #91 (R1-1719301), it was
    % observed in R1-1717405 that if performance is the priority, [0 2 3 1]
    % should be used. If self-decodability is the priority, it should be
    % taken into account that the upper limit of the code rate at which
    % each RV is self-decodable is in the following order: 0>3>2>1
    rvSeq = [0 2 3 1];
else
    % In case of HARQ disabled, RV is set to 0
    rvSeq = 0;
end

% Create DL-SCH encoder System object to perform transport channel encoding
encodeDLSCH = nrDLSCH;
encodeDLSCH.MultipleHARQProcesses = true;
encodeDLSCH.TargetCodeRate = simParameters.PDSCHExtension.TargetCodeRate;

% Create DL-SCH decoder System object to perform transport channel decoding
decodeDLSCH = nrDLSCHDecoder;
decodeDLSCH.MultipleHARQProcesses = true;
decodeDLSCH.TargetCodeRate = simParameters.PDSCHExtension.TargetCodeRate;
decodeDLSCH.LDPCDecodingAlgorithm = simParameters.PDSCHExtension.LDPCDecodingAlgorithm;
decodeDLSCH.MaximumLDPCIterationCount = ...
    simParameters.PDSCHExtension.MaximumLDPCIterationCount;

% Set a threshold value to detect the valid OFDM symbol boundary. For a
% SISO case, a threshold of 0.48 can be used to have probability of
% incorrect boundary detection around 0.01. Use 0 to avoid thresholding
% logic.
dtxThresold = 0.48;

% Use an offset to account for the common delay. The example, by default,
% does not introduce any common delay and only passes through the channel.
sampleDelayOffset = 0; % Number of samples

% Set this flag to true, to use the shift value estimated in first slot
% directly for the consecutive slots. When set to false, the shift is
% calculated for each slot, considering the range of shift values to be
% whole cyclic prefix length. This is used in the estimation of integer
% Doppler shift.
usePreviousShift = false;

% Processing loop
for snrIdx = 1:numel(simParameters.SNRIn) % Comment out for parallel computing
% parfor snrIdx = 1:numel(simParameters.SNRIn) % Uncomment for parallel computing
    % To reduce the total simulation time, you can execute this loop in

```

```
% parallel by using Parallel Computing Toolbox features. Comment
% out the for-loop statement and uncomment the parfor-loop statement.
% If Parallel Computing Toolbox is not installed, parfor-loop defaults
% to a for-loop statement. Because the parfor-loop iterations are
% executed in parallel in a nondeterministic order, the simulation
% information displayed for each SNR point can be intertwined. To
% switch off the simulation information display, set the
% displaySimulationInformation variable (defined earlier in this
% example) to false.

% Reset the random number generator so that each SNR point experiences
% the same noise realization
rng('default');

% Make copies of the simulation-level parameter structures so that they
% are not Parallel Computing Toolbox broadcast variables when using parfor
simLocal = simParameters;
waveinfoLocal = waveformInfo;
ntnChanLocal = ntnChan;

% Make copies of channel-level parameters to simplify subsequent
% parameter referencing
carrier = simLocal.Carrier;
pdsch = simLocal.PDSCH;
pdschextra = simLocal.PDSCHExtension;
% Copy of the decoder handle to help Parallel Computing Toolbox
% classification
decodeDLSCHLocal = decodeDLSCH;
decodeDLSCHLocal.reset(); % Reset decoder at the start of each SNR point
pathFilters = [];
thres = dtxThresold;
sampleOffset = sampleDelayOffset;
usePrevShift = usePreviousShift;

% SNR value
SNRdB = simLocal.SNRIn(snrIdx);

% Specify the order in which we cycle through the HARQ process
% identifiers
harqSequence = 0:pdschextra.NHARQProcesses-1;

% Initialize the state of all HARQ processes
harqEntity = HARQEntity(harqSequence, rvSeq, pdsch.NumCodewords);

% Reset the channel so that each SNR point experiences the same channel
% realization
reset(ntnChanLocal.BaseChannel);
reset(ntnChanLocal.ChannelFilter);

% Total number of slots in the simulation period
NSlots = simLocal.NFrames*carrier.SlotsPerFrame;

% Obtain a precoding matrix (wtx) to use in the transmission of the
% first transport block
[estChannelGrid, sampleTimes] = getInitialChannelEstimate(...
    carrier, simLocal.NumTransmitAntennas, ntnChanLocal);
newWtx = getPrecodingMatrix(carrier, pdsch, estChannelGrid, pdschextra.PRGBundleSize);
```

```

% Timing offset, updated in every slot when the correlation is strong
offset = 0;
tDoppler = sampleTimes(end);
shiftOut = 0;

% Loop over the entire waveform length
for nslot = 0:NSlots-1

    % Update carrier slot number to account for new slot transmission
    carrier.NSlot = nslot;

    % Calculate the transport block sizes for the transmission in the slot
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);
    trBlkSizes = nrTBS(pdsch.Modulation,pdsch.NumLayers,numel(pdsch.PRBSets),...
        pdschIndicesInfo.NREPerPRB,pdschextra.TargetCodeRate,pdschextra.XOverhead);

    % Set transport block depending on the HARQ process
    for cwIdx = 1:pdsch.NumCodewords
        % Create a new DL-SCH transport block for new data in the
        % current process
        if harqEntity.NewData(cwIdx)
            trBlk = randi([0 1],trBlkSizes(cwIdx),1);
            setTransportBlock(encodeDLSCH,trBlk,cwIdx-1,harqEntity.HARQProcessID);
            % Flush decoder soft buffer explicitly for any new data
            % because of previous RV sequence time out
            if harqEntity.SequenceTimeout(cwIdx)
                resetSoftBuffer(decodedDLSCHLocal,cwIdx-1,harqEntity.HARQProcessID);
            end
        end
    end
end

% Encode the DL-SCH transport blocks
codedTrBlocks = encodeDLSCH(pdsch.Modulation,pdsch.NumLayers, ...
    pdschIndicesInfo.G,harqEntity.RedundancyVersion,harqEntity.HARQProcessID);

% Get precoding matrix (wtX) calculated in previous slot
wtX = newWtX;

% Perform PDSCH modulation
pdschSymbols = nrPDSCH(carrier,pdsch,codedTrBlocks);

% Create resource grid associated with PDSCH transmission antennas
pdschGrid = nrResourceGrid(carrier,simLocal.NumTransmitAntennas);

% Perform implementation-specific PDSCH MIMO precoding and mapping
[pdschAntSymbols,pdschAntIndices] = hPRGPrecode(size(pdschGrid), ...
    carrier.NStartGrid,pdschSymbols,pdschIndices,wtX);
pdschGrid(pdschAntIndices) = pdschAntSymbols;

% Perform implementation-specific PDSCH DM-RS MIMO precoding and
% mapping
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
[dmrsAntSymbols,dmrsAntIndices] = hPRGPrecode(size(pdschGrid), ...
    carrier.NStartGrid,dmrsSymbols,dmrsIndices,wtX);
pdschGrid(dmrsAntIndices) = dmrsAntSymbols;

% Perform implementation-specific PDSCH PT-RS MIMO precoding and

```

```

% mapping
ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);
ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
[ptrsAntSymbols,ptrsAntIndices] = hPRGPrecode(size(pdschGrid), ...
    carrier.NStartGrid,ptrsSymbols,ptrsIndices,wtX);
pdschGrid(ptrsAntIndices) = ptrsAntSymbols;

% Perform OFDM modulation
txWaveform = nrOFDMModulate(carrier,pdschGrid);

% Pass data through the channel model. Append zeros at the end of
% the transmitted waveform to flush the channel content. These
% zeros take into account any delay introduced in the channel. This
% delay is a combination of the multipath delay and implementation
% delay. This value can change depending on the sampling rate,
% delay profile, and delay spread. Also apply Doppler
% pre-compensation.
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))]; %#ok<AGROW>
[txWaveform,tDoppler] = compensateDopplerShift(...
    txWaveform,ntnChanLocal.BaseChannel.SampleRate, ...
    ntnChanLocal.SatelliteDopplerShift, ...
    simLocal.DopplerPreCompensator,tDoppler);
[rxWaveform,pathGains] = HelperGenerateNTNChannel(...
    ntnChanLocal,txWaveform);

% Add AWGN to the received time-domain waveform. Normalize the
% noise power by the size of the inverse fast Fourier transform
% (IFFT) used in OFDM modulation, because the OFDM modulator
% applies this normalization to the transmitted waveform. Also,
% normalize the noise power by the number of receive antennas,
% because the default behavior of the channel model is to apply
% this normalization to the received waveform.
SNR = 10^(SNRdB/10);
noiseScaling = ...
    1/sqrt(2.0*simLocal.NumReceiveAntennas*double(waveinfoLocal.Nfft)*SNR);
noise = noiseScaling*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

% Perform fractional Doppler frequency shift estimation and
% compensation. Use the cyclic prefix in the OFDM waveform to
% compute the fractional Doppler shift.
[fractionalDopplerShift,detFlag] = estimateFractionalDopplerShift( ...
    rxWaveform,carrier.SubcarrierSpacing,waveinfoLocal.Nfft, ...
    waveinfoLocal.CyclicPrefixLengths(2),thres, ...
    simLocal.RxDopplerCompensator);
rxWaveform = compensateDopplerShift(rxWaveform,waveinfoLocal.SampleRate, ...
    fractionalDopplerShift,simLocal.RxDopplerCompensator);

% Perform integer Doppler frequency shift estimation and
% compensation. Use the demodulation reference signals to compute
% the integer Doppler shift.
[integerDopplerShift,shiftOut] = estimateIntegerDopplerShift(carrier,rxWaveform, ...
    dmrsIndices,dmrsSymbols,sampleOffset,usePrevShift,shiftOut-sampleOffset, ...
    (simLocal.RxDopplerCompensator && detFlag));
rxWaveform = compensateDopplerShift(rxWaveform,waveinfoLocal.SampleRate, ...
    integerDopplerShift,simLocal.RxDopplerCompensator);

% For timing synchronization, correlate the received waveform with

```

```

% the PDSCH DM-RS to give timing offset estimate t and correlation
% magnitude mag. The function hSkipWeakTimingOffset is used to
% update the receiver timing offset. If the correlation peak in mag
% is weak, the current timing estimate t is ignored and the
% previous estimate offset is used.
[t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols);
offset = hSkipWeakTimingOffset(offset,t,mag);
% Display a warning if the estimated timing offset exceeds the
% maximum channel delay
if offset > maxChDelay
    warning(['Estimated timing offset (%d) is greater than the maximum' ...
            ' channel delay (%d). This might result in a decoding failure.' ...
            ' The estimated timing offset might be greater than the maximum' ...
            ' channel delay because of low SNR or not enough DM-RS symbols to' ...
            ' synchronize successfully.'],offset,maxChDelay);
end
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to recreate the
% resource grid. Include zero padding in the event that practical
% synchronization results in an incomplete slot being demodulated.
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end

% Perform least squares channel estimation between the received
% grid and each transmission layer, using the PDSCH DM-RS for each
% layer. This channel estimate includes the effect of transmitter
% precoding.
[estChannelGrid,noiseEst] = nrChannelEstimate(carrier,rxGrid,...
    dmrsIndices,dmrsSymbols,'CDMLengths',pdsch.DMRS.CDMLengths);

% Get PDSCH REs from the received grid and estimated channel grid
[pdschRx,pdschHest] = nrExtractResources(...
    pdschIndices,rxGrid,estChannelGrid);

% Remove precoding from estChannelGrid prior to precoding
% matrix calculation
estChannelGridPorts = precodeChannelEstimate(...
    carrier,estChannelGrid,conj(wtx));

% Get precoding matrix for next slot
newWtx = getPrecodingMatrix(...
    carrier,pdsch,estChannelGridPorts,pdschextra.PRGBundleSize);

% Perform equalization
[pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

% Common phase error (CPE) compensation
if ~isempty(ptrsIndices)
    % Initialize temporary grid to store equalized symbols
    tempGrid = nrResourceGrid(carrier,pdsch.NumLayers);

    % Extract PT-RS symbols from received grid and estimated
    % channel grid
    [ptrsRx,ptrsHest,~,~,~,ptrsLayerIndices] = ...

```

```

        nrExtractResources(ptrsIndices,rxGrid,estChannelGrid,tempGrid);

% Equalize PT-RS symbols and map them to tempGrid
ptrsEq = nrEqualizeMMSE(ptrsRx,ptrsHest,noiseEst);
tempGrid(ptrsLayerIndices) = ptrsEq;

% Estimate the residual channel at the PT-RS locations in
% tempGrid
cpe = nrChannelEstimate(tempGrid,ptrsIndices,ptrsSymbols);

% Sum estimates across subcarriers, receive antennas, and
% layers. Then, get the CPE by taking the angle of the
% resultant sum
cpe = angle(sum(cpe,[1 3 4]));

% Map the equalized PDSCH symbols to tempGrid
tempGrid(pdschIndices) = pdschEq;

% Correct CPE in each OFDM symbol within the range of reference
% PT-RS OFDM symbols
symLoc = ...
    pdschIndicesInfo.PTRSSymbolSet(1)+1:pdschIndicesInfo.PTRSSymbolSet(end)+1;
tempGrid(:,symLoc,:) = tempGrid(:,symLoc,:).*exp(-1i*cpe(symLoc));

% Extract PDSCH symbols
pdschEq = tempGrid(pdschIndices);
end

% Decode PDSCH symbols
[dlschLLRs,rxSymbols] = nrPDSCHDecode(carrier,pdsch,pdschEq,noiseEst);

% Scale the decoded log-likelihood ratios (LLRs) by channel state
% information (CSI)
csi = nrLayerDemap(csi); % CSI layer demapping
for cwIdx = 1:pdsch.NumCodewords
    Qm = length(dlschLLRs{cwIdx})/length(rxSymbols{cwIdx}); % Bits per symbol
    csi{cwIdx} = repmat(csi{cwIdx}.' ,Qm,1); % Expand by each bit
                                           % per symbol
    dlschLLRs{cwIdx} = dlschLLRs{cwIdx} .* csi{cwIdx}(:); % Scale by CSI
end

% Decode the DL-SCH transport channel
decodeDLSCHLocal.TransportBlockLength = trBlkSizes;
[decbits,blkerr] = decodeDLSCHLocal(dlschLLRs,pdsch.Modulation,...
    pdsch.NumLayers,harqEntity.RedundancyVersion,harqEntity.HARQProcessID);

% Store values to calculate throughput
simThroughput(snrIdx) = simThroughput(snrIdx) + sum(~blkerr .* trBlkSizes);
maxThroughput(snrIdx) = maxThroughput(snrIdx) + sum(trBlkSizes);

% Update current process with CRC error and advance to next process
updateAndAdvance(harqEntity,blkerr,trBlkSizes,pdschIndicesInfo.G);
end

% Display the results
if displaySimulationInformation == 1
    fprintf('\nThroughput(Mbps) for %d frame(s) at SNR %d dB: %.4f\n',...
        simLocal.NFrames,SNRdB,1e-6*simThroughput(snrIdx)/(simLocal.NFrames*10e-3));
end

```

```

    fprintf('Throughput(%%) for %d frame(s) at SNR %d dB: %.4f\n',...
           simLocal.NFrames,SNRdB,simThroughput(snrIdx)*100/maxThroughput(snrIdx));
end

```

```
end
```

```

Throughput(Mbps) for 2 frame(s) at SNR 5 dB: 0.0000
Throughput(%) for 2 frame(s) at SNR 5 dB: 0.0000
Throughput(Mbps) for 2 frame(s) at SNR 7 dB: 0.0000
Throughput(%) for 2 frame(s) at SNR 7 dB: 0.0000
Throughput(Mbps) for 2 frame(s) at SNR 9 dB: 6.2308
Throughput(%) for 2 frame(s) at SNR 9 dB: 92.5000
Throughput(Mbps) for 2 frame(s) at SNR 11 dB: 6.7360
Throughput(%) for 2 frame(s) at SNR 11 dB: 100.0000
Throughput(Mbps) for 2 frame(s) at SNR 13 dB: 6.7360
Throughput(%) for 2 frame(s) at SNR 13 dB: 100.0000
Throughput(Mbps) for 2 frame(s) at SNR 15 dB: 6.7360
Throughput(%) for 2 frame(s) at SNR 15 dB: 100.0000

```

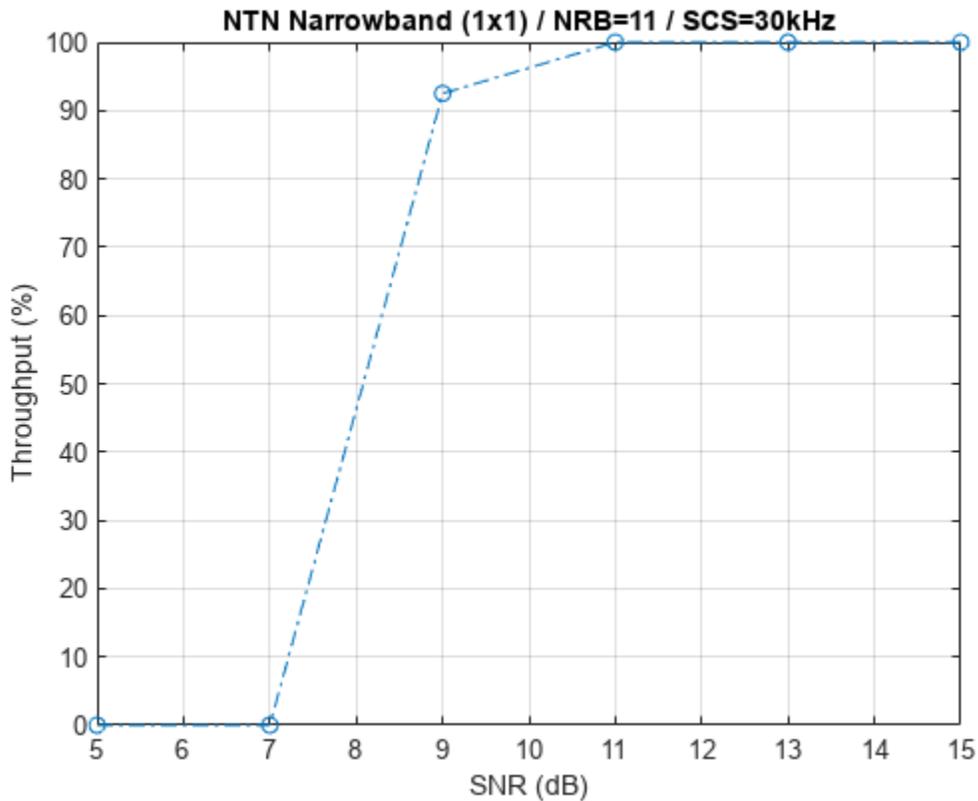
Results

Display the measured throughput, which is the percentage of the maximum possible throughput of the link given the available resources for data transmission.

```

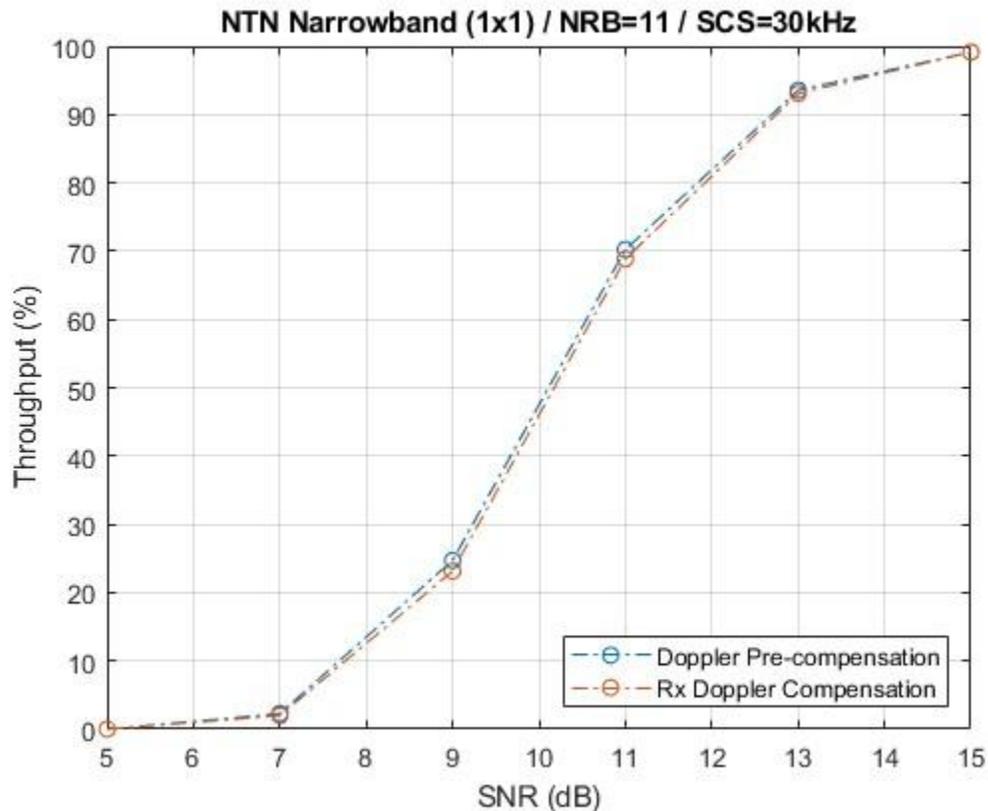
figure;
plot(simParameters.SNRIn,simThroughput*100./maxThroughput,'o-.');
xlabel('SNR (dB)'); ylabel('Throughput (%)'); grid on;
title(sprintf('NTN %s (%dx%d) / NRB=%d / SCS=%dkHz', ...
             simParameters.NTNChannelType,simParameters.NumTransmitAntennas, ...
             simParameters.NumReceiveAntennas, simParameters.Carrier.NSizeGrid,...
             simParameters.Carrier.SubcarrierSpacing));

```



```
% Bundle key parameters and results into a combined structure for recording
simResults.simParameters = simParameters;
simResults.simThroughput = simThroughput;
simResults.maxThroughput = maxThroughput;
```

This next figure shows the throughput results obtained by simulating 1000 frames ($N_{\text{Frames}} = 1000$, $\text{SNR}_{\text{In}} = 5:2:15$) for a carrier with a 30 kHz SCS occupying a 5 MHz transmission bandwidth. The simulation setup includes the default carrier and PDSCH configuration with an NTN narrowband channel. The line corresponding to the Doppler Pre-compensator is achieved by setting the `DopplerPreCompensator` field to true and `RxDopplerCompensator` field to false. The line corresponding to the Rx Doppler Compensator is achieved by setting the `DopplerPreCompensator` field to false and `RxDopplerCompensator` field to true.



Further Exploration

You can use this example to further explore these options:

- To analyze the throughput at each SNR point for a different satellite orbit, vary the satellite altitude and satellite speed.
- To observe the link performance without any Doppler compensation techniques, set `DopplerPreCompensator` and `RxDopplerCompensator` fields to false.
- To observe the link performance in case of no Doppler pre-compensation and using receiver techniques to compensate Doppler shift, set the `DopplerPreCompensator` field to false and `RxDopplerCompensator` field to true.
- To check the throughput performance of different scenarios, change the carrier numerology and the number of transmit and receive antennas, and set the channel model type to TDL.
- To compare the throughput performance in an NTN and terrestrial network, use the `nRTDLChannel` (5G Toolbox) and the `nrcDLChannel` (5G Toolbox) channel objects as shown in “NR PDSCH Throughput” (5G Toolbox).

Appendix

The example uses these helper functions:

- `HARQEntity` — Manage a set of parallel HARQ processes
- `HelperGenerateNTNChannel` — Generate NTN channel

- HelperSetupNTNChannel — Set up NTN channel
- hPRGPrecode — Precoding for PDSCH precoding resource block group (PRG) bundling
- hSkipWeakTimingOffset — Skip timing offset estimates with weak correlation

Selected Bibliography

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[2] 3GPP TS 38.212. "NR; Multiplexing and channel coding." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[3] 3GPP TS 38.213. "NR; Physical layer procedures for control." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[4] 3GPP TS 38.214. "NR; Physical layer procedures for data." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[5] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[6] 3GPP TR 38.811. "Study on new radio (NR) to support non-terrestrial networks." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[7] 3GPP TR 38.821. "Solutions for NR to support non-terrestrial networks (NTN)." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

[8] ITU-R Recommendation P.681-11 (08/2019). "Propagation data required for the design systems in the land mobile-satellite service." P Series; Radio wave propagation.

Local Functions

```
function validateNumLayers(simParameters)
% Validate the number of layers, relative to the antenna geometry

    numLayers = simParameters.PDSCH.NumLayers;
    ntxants = simParameters.NumTransmitAntennas;
    nrxants = simParameters.NumReceiveAntennas;

    if contains(simParameters.NTNChannelType, 'Narrowband', 'IgnoreCase', true)
        if (ntxants ~= 1) || (nrxants ~= 1)
            error(['For NTN narrowband channel, ' ...
                'the number of transmit and receive antennas must be 1.']);
        end
    end

    antennaDescription = sprintf(...
        'min(NumTransmitAntennas,NumReceiveAntennas) = min(%d,%d) = %d', ...
        ntxants,nrxants,min(ntxants,nrxants));
```

```

if numlayers > min(ntxants,nrxants)
    error('The number of layers (%d) must satisfy NumLayers <= %s', ...
        numlayers,antennaDescription);
end

% Display a warning if the maximum possible rank of the channel equals
% the number of layers
if (numlayers > 2) && (numlayers == min(ntxants,nrxants))
    warning(['The maximum possible rank of the channel, given by %s, is equal to' ...
        ' NumLayers (%d). This may result in a decoding failure under some channel' ...
        ' conditions. Try decreasing the number of layers or increasing the channel' ...
        ' rank (use more transmit or receive antennas).'],antennaDescription, ...
        numlayers); %#ok<SPWRN>
end

end

function [estChannelGrid,sampleTimes] = getInitialChannelEstimate(...
    carrier,nTxAnts,ntnChan)
% Obtain channel estimate before first transmission. Use this function to
% obtain a precoding matrix for the first slot.

    ofdmInfo = nrOFDMInfo(carrier);

    chInfo = info(ntnChan.BaseChannel);
    maxChDelay = ceil(max(chInfo.PathDelays*ntnChan.BaseChannel.SampleRate)) ...
        + chInfo.ChannelFilterDelay;

% Temporary waveform (only needed for the sizes)
tmpWaveform = zeros(...
    (ofdmInfo.SampleRate/1000/carrier.SlotsPerSubframe)+maxChDelay,nTxAnts);

% Filter through channel and get the path gains and path filters
[~,pathGains,sampleTimes] = HelperGenerateNTNChannel(...
    ntnChan,tmpWaveform);
chanInfo = info(ntnChan.ChannelFilter);
pathFilters = chanInfo.ChannelFilterCoefficients.';

% Perfect timing synchronization
offset = nrPerfectTimingEstimate(pathGains,pathFilters);

% Perfect channel estimate
estChannelGrid = nrPerfectChannelEstimate(...
    carrier,pathGains,pathFilters,offset,sampleTimes);

end

function wtx = getPrecodingMatrix(carrier,pdsch,hestGrid,prgbundlesize)
% Calculate precoding matrices for all precoding resource block groups
% (PRGs) in the carrier that overlap with the PDSCH allocation

% Maximum common resource block (CRB) addressed by carrier grid
maxCRB = carrier.NStartGrid + carrier.NSizeGrid - 1;

% PRG size
if nargin==4 && ~isempty(prgbundlesize)
    Pd_BWP = prgbundlesize;
else

```

```

    Pd_BWP = maxCRB + 1;
end

% PRG numbers (1-based) for each RB in the carrier grid
NPRG = ceil((maxCRB + 1) / Pd_BWP);
prgset = repmat((1:NPRG),Pd_BWP,1);
prgset = prgset(carrier.NStartGrid + (1:carrier.NSizeGrid).');

[~,~,R,P] = size(hestGrid);
wtx = zeros([pdsch.NumLayers P NPRG]);
for i = 1:NPRG

    % Subcarrier indices within current PRG and within the PDSCH
    % allocation
    thisPRG = find(prgset==i) - 1;
    thisPRG = intersect(thisPRG,pdsch.PRBSets(:) + carrier.NStartGrid,'rows');
    prgSc = (1:12)' + 12*thisPRG';
    prgSc = prgSc(:);

    if (~isempty(prgSc))

        % Average channel estimate in PRG
        estAllocGrid = hestGrid(prgSc,:,:,);
        Hest = permute(mean(reshape(estAllocGrid,[],R,P)),[2 3 1]);

        % SVD decomposition
        [~,~,V] = svd(Hest);
        wtx(:,:,i) = V(:,1:pdsch.NumLayers).';

    end

end

wtx = wtx / sqrt(pdsch.NumLayers); % Normalize by NumLayers
end

function estChannelGrid = precodeChannelEstimate(carrier,estChannelGrid,W)
% Apply precoding matrix W to the last dimension of the channel estimate

[K,L,R,P] = size(estChannelGrid);
estChannelGrid = reshape(estChannelGrid,[K*L R P]);
estChannelGrid = hPRGPrecode([K L R P],carrier.NStartGrid,estChannelGrid, ...
    reshape(1:numel(estChannelGrid),[K*L R P]),W);
estChannelGrid = reshape(estChannelGrid,K,L,R,[]);
end

function [loc,wMovSum,pho,bestAnt] = detectOFDMSymbolBoundary(rxWave,nFFT,cpLen,thres)
% Detect OFDM symbol boundary by calculating correlation of cyclic prefix

% Capture the dimensions of received waveform
[NSamples,R] = size(rxWave);

% Append zeros of length nFFT across each receive antenna to the
% received waveform
waveformZeroPadded = [rxWave;zeros(nFFT,R)];

```

```

% Get the portion of waveform till the last nFFT samples
wavePortion1 = waveformZeroPadded(1:end-nFFT,:);

% Get the portion of waveform delayed by nFFT
wavePortion2 = waveformZeroPadded(1+nFFT:end,:);

% Get the energy of each sample in both the waveform portions
eWavePortion1 = abs(wavePortion1).^2;
eWavePortion2 = abs(wavePortion2).^2;

% Initialize the temporary variables
wMovSum = zeros([NSamples R]);
wEnergyPortion1 = zeros([NSamples+cpLen-1 R]);
wEnergyPortion2 = wEnergyPortion1;

% Perform correlation for each sample with the sample delayed by nFFT
waveCorr = wavePortion1.*conj(wavePortion2);
% Calculate the moving sum value for each cpLen samples, across each
% receive antenna
oneVec = ones(cpLen,1);
for i = 1:R
    wConv = conv(waveCorr(:,i),oneVec);
    wMovSum(:,i) = wConv(cpLen:end);
    wEnergyPortion1(:,i) = conv(eWavePortion1(:,i),oneVec);
    wEnergyPortion2(:,i) = conv(eWavePortion2(:,i),oneVec);
end

% Get the normalized correlation value for the moving sum matrix
pho = abs(wMovSum)./ ...
    (eps+sqrt(wEnergyPortion1(cpLen:end,:).*wEnergyPortion2(cpLen:end,:)));

% Detect the peak locations in each receive antenna based on the
% threshold. These peak locations correspond to the starting location
% of each OFDM symbol in the received waveform.
loc = cell(R,1);
m = zeros(R,1);
phoFactor = ceil(NSamples/nFFT);
phoExt = [pho; -1*ones(phoFactor*nFFT - NSamples,R)];
for col = 1:R
    p1 = reshape(phoExt(:,col),[],phoFactor);
    [pks,locTemp] = max(p1);
    locTemp = locTemp + (0:phoFactor-1).*nFFT;
    indicesToConsider = pks>=thres;
    loc{col} = locTemp(indicesToConsider);
    m(col) = max(pks);
end
bestAnt = find(m == max(m));

end

function [out,detFlag] = estimateFractionalDopplerShift(rxWave,scs, ...
    nFFT,cpLen,thres,flag)
% Estimate the fractional Doppler shift using cyclic prefix

if flag
    % Detect the OFDM boundary locations
    [loc,wMovSum,~,bestAnt] = detectOFDMsymbolBoundary(rxWave, ...
        nFFT,cpLen,thres);

```

```

% Get the average correlation value at the peak locations for the
% first receive antenna having maximum correlation value
wSamples = nan(1,1);
if ~isempty(loc{bestAnt(1)})
    wSamples(1) = mean(wMovSum(loc{bestAnt(1)},bestAnt(1)));
end

% Compute the fractional Doppler shift
if ~all(isnan(wSamples))
    out = -(mean(angle(wSamples), 'omitnan')*scs*1e3)/(2*pi);
    % Flag to indicate that there is at least one OFDM symbol
    % detected
    detFlag = 1;
else
    out = 0;
    detFlag = 0;
end
else
    out = 0;
    detFlag = 0;
end
end

function [out,shiftOut] = estimateIntegerDopplerShift(carrier,rx,refInd, ...
    refSym,sampleOffset,usePrevShift,shiftIn,flag)
% Estimate the integer Doppler shift using demodulation reference signal

if flag
    % Get OFDM information
    ofdmInfo = nrOFDMInfo(carrier);
    K = carrier.NSizeGrid*12; % Number of subcarriers
    L = ofdmInfo.SymbolsPerSlot; % Number of OFDM symbols in slot
    P = ceil(max(double(refInd(:))/(K*L))); % Number of layers
    cpLen = ofdmInfo.CyclicPrefixLengths(1); % Highest cyclic prefix length

    % Range of shift values to be used in integer frequency offset
    % estimation
    shiftValues = sampleOffset + shiftIn;
    if ~(usePrevShift && (shiftIn > 0))
        % Update range of shift values such that whole cyclic prefix
        % length is covered
        shiftValues = sampleOffset + (1:cpLen);
    end

    % Initialize temporary variables
    shiftLen = length(shiftValues);
    maxValue = complex(zeros(shiftLen,1));
    binIndex = zeros(shiftLen,1);
    R = size(rx,2);
    rx1 = [rx; zeros(1*(mod(size(rx,1),2)),R)]; % Append zero, if required
    rxLen = size(rx1,1);
    x_wave = zeros([rxLen P]);

    % Generate reference waveform
    refGrid = complex(zeros([K L P]));
    refGrid(refInd) = refSym;

```

```

refWave = nrOFDMModulate(carrier,refGrid,'Windowing',0);
refWave = [refWave; zeros((rxLen-size(refWave,1)),P)];

% Find the fast Fourier transform (FFT) bin corresponding to
% maximum correlation value for each shift value
for shiftIdx = 1:shiftLen
    % Use the waveform from the shift value and append zeros
    tmp = rx1(shiftValues(shiftIdx):end,:);
    rx = [tmp; zeros(rxLen-size(tmp,1),R)];

    % Compute the correlation of received waveform with reference
    % waveform across different layers and receive antennas
    for rIdx = 1:R
        for p = 1:P
            x_wave(:,rIdx,p) = ...
                rx(:,rIdx).*conj(refWave(1:length(rx(:,rIdx)),p));
        end
    end

    % Aggregate the correlated waveform across multiple ports and
    % compute energy of the resultant for each receive antenna
    x1 = sum(x_wave,3);
    x1P = sum(abs(x1).^2);

    % Find the index of first receive antenna which has maximum
    % correlation energy
    idx = find(x1P == max(x1P),1);

    % Combine the received waveform which have maximum correlation
    % energy
    x_wave_combined = sum(x1(:,idx(1)),2);

    % Compute FFT of the resultant waveform
    x_fft = fftshift(fft(x_wave_combined));

    % Store the value and location of peak
    [maxValue(shiftIdx),binIndex(shiftIdx)] = max(x_fft);
end

% FFT bin values
fftBinValues = (-rxLen/2:(rxLen/2-1))*(ofdmInfo.SampleRate/rxLen);

% Find the shift index that corresponds to the maximum of peak
% value of all the shifted waveforms. Use the FFT bin index
% corresponding to this maximum shift index. The FFT bin value
% corresponding to this bin index is the integer frequency offset.
[~,maxId] = max(maxValue);
loc = binIndex(maxId);
out = fftBinValues(loc);
shiftOut = shiftValues(maxId);
else
    out = 0;
    shiftOut = 1+sampleOffset;
end
end

function [out,t] = compensateDopplerShift(inWave,fs,fdSat,flag,t)

```

```
% Perform Doppler shift correction

t1 = (0:size(inWave,1)-1)'/fs;
if nargin == 5
    t1 = t1 + t; % Add the sample time offset
end
if flag
    out = inWave.*exp(1j*2*pi*(-fdSat)*t1);
else
    out = inWave;
end
t = t1(end);

end
```

See Also

Related Examples

- “Model NR NTN Channel” on page 3-8

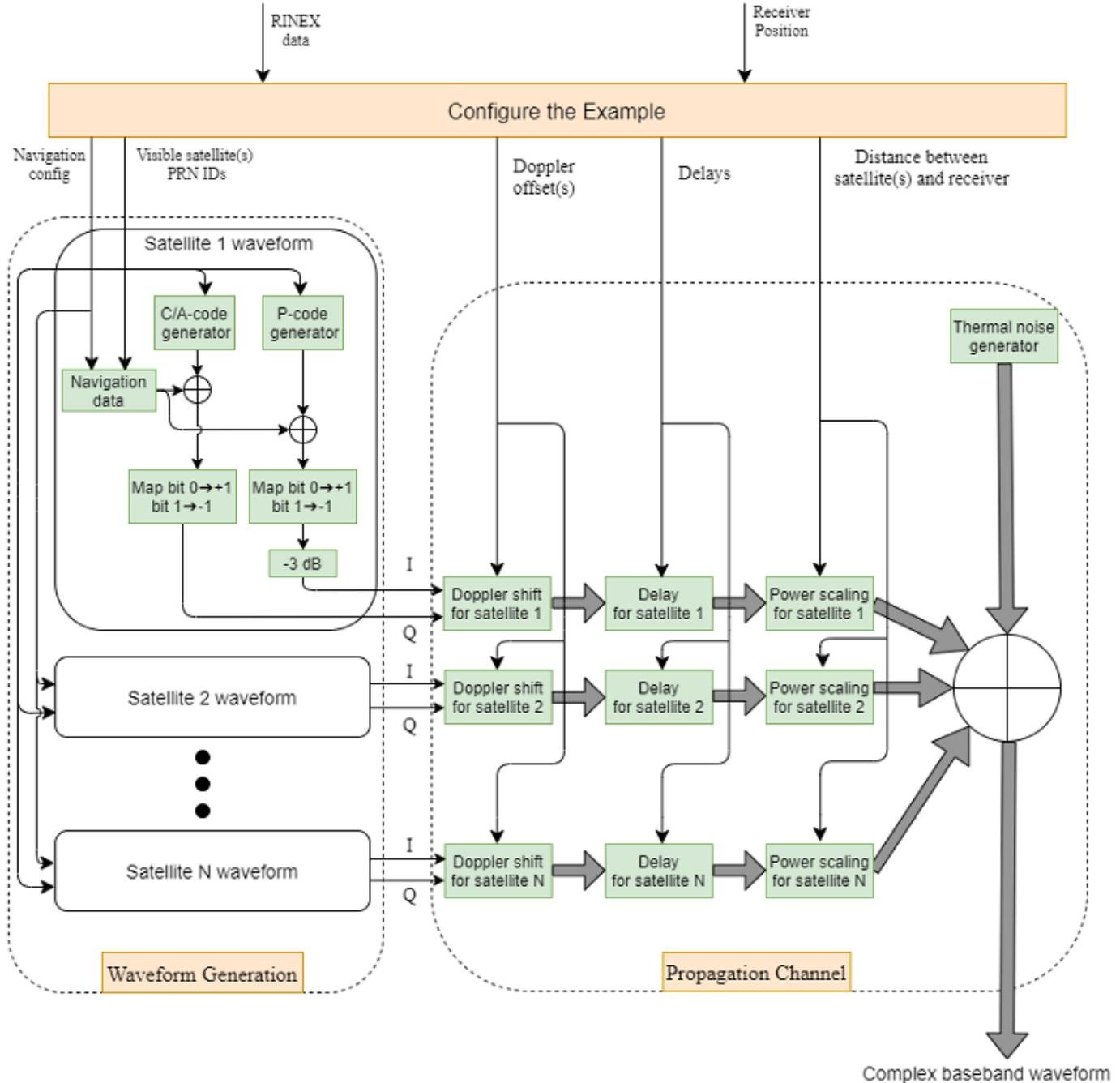
End-to-End GPS Legacy Navigation Receiver Using C/A-Code

This example shows how to estimate the global positioning system (GPS) receiver position using a multi-satellite GPS baseband waveform. You use the receiver independent exchange format (RINEX) and an almanac file to model the GPS constellation and generate a multi-satellite baseband waveform. Simulate the satellite scenario to get relative positions of satellites with respect to the modeled receiver. For this satellite scenario, model the Doppler shift, delay, and received signal power. Based on these calculations, impair the generated baseband signal with Doppler shift, delay, and noise. This example shows how to estimate the simulated receiver position from this impaired GPS baseband signal.

Introduction

In this example, you start with a RINEX file and use `rinexread` (Navigation Toolbox) to read the file and provide input to `satelliteScenario` (Satellite Communications Toolbox) to simulate the GPS constellation. Based on this GPS constellation and a given receiver position, you calculate the Doppler shift, delay, and signal path loss from each visible satellite to the receiver. Based on the ephemeris and clock and almanac data from the RINEX and almanac files, generate data bits as per the IS-GPS-200 [1] on page 4-187 standard. For the satellites that are visible to the receiver, generate coarse acquisition code (C/A-code) and precision code (P-code). P-code is placed on the in-phase (I) branch and C/A-code is placed on the quadrature-phase (Q) branch of the baseband waveform. The I-branch signal is attenuated by 3 dB as given in IS-GPS-200 [1] on page 4-187. Generate the baseband waveform for all the visible satellites and pass this baseband waveform through the propagation channel as shown in the following figure. Propagation channel characteristics for each satellite signal are unique because the position and velocity of each satellite with respect to the receiver is unique. In this example, you model the characteristics of the propagation channel — namely, the Doppler shift, delay, and scaling signal power based on the propagation path loss — and add thermal noise to the composite signal. Provide this noisy signal as input to the GPS receiver. This example supports storing this noisy signal in a file so that you can test your receiver.

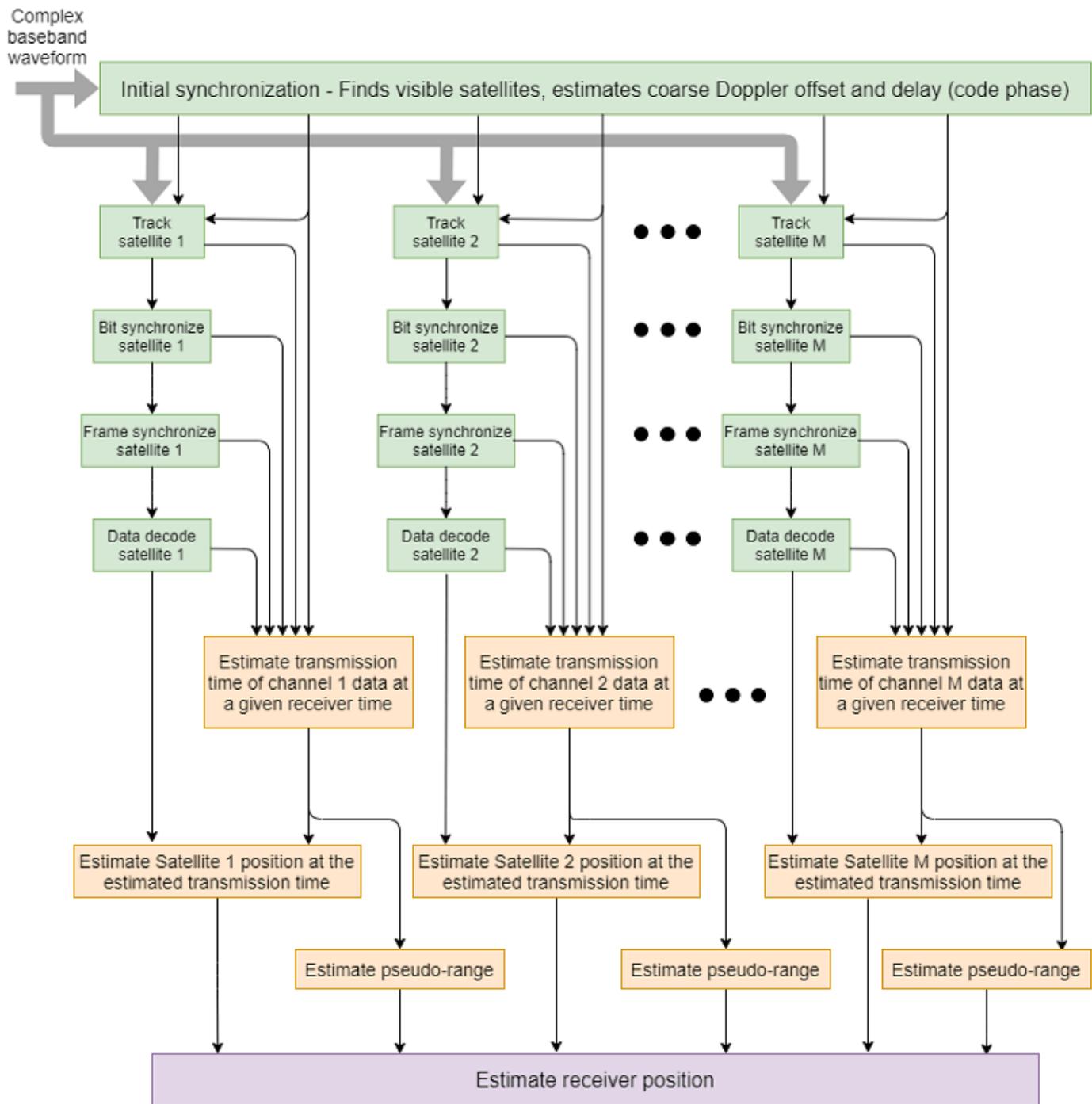
- For a detailed description of properties of GPS waveform generation, see “GPS Waveform Generation” on page 2-2.
- For more details on how to calculate latency and Doppler, see “Calculate Latency and Doppler in a Satellite Scenario” on page 1-49.



The complex baseband waveform generated from the propagation channel is processed through a GPS receiver. The following diagram shows the high-level details of such a GPS receiver. First, the initial synchronization module detects visible satellites. It estimates coarse values of the Doppler offset and delay for the visible satellites. For each of the detected satellites, create separate receiver channels (not to be confused with the propagation channel). Each receiver channel performs tracking, bit synchronization, frame synchronization, data decoding, transmitted time estimation, satellite position estimation, and pseudo-ranges computation. For the estimation of the receiver position, estimate the position of the satellites in the sky and distances from the satellites to the

receiver. Estimate the position of the satellites from the decoded ephemeris data as provided in Table 20-IV in IS-GPS-200 [1] on page 4-187. Estimate the distances from the satellites to the receiver by estimating the propagation time and multiplying it with the speed of light. For computation of this propagation time at the receiver, compute the difference between receiver time and transmission time. Although accurately computing the receiver time is ideal, GPS receiver cannot compute it. It can estimate only the signal transmission time with high accuracy [2] on page 4-187. Accurate receiver time is not needed for GPS receiver position estimation. At a given receiver time, compute the transmission time of each satellite signal and estimate the pseudo-range for each satellite from this value. The pseudo-ranges are not true ranges and have large errors, but because the errors are the same for all the satellites, you can account for the error when solving for the receiver position and get the accurate receiver position [2] on page 4-187. For more details on receiver position estimation, see GPS Receiver Position Estimation on page 4-183 section.

- For a detailed explanation of initial synchronization and tracking, see “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118.
- For more details of decoding GPS data, see “GPS Data Decode” on page 4-139.
- For more details on estimating receiver position, see “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43.



Initialize Parameters

Initialize the parameters that are necessary to configure and run the end-to-end GPS receiver simulation.

Initialize the data duration for which this example must run. Typically, a GPS receiver needs at least 50 seconds of data to estimate the receiver position. To simulate this example for 50 seconds of data would take a lot of wall-clock time based on the available computer resources. For the purposes of

this example, set the data duration to 3 seconds. Also, initialize the sampling rate for baseband waveform generation.

```
simulatedDataDuration = 3; % In seconds
samplingRate = 10.23e6; % In Hz
```

If needed, enable the `WriteWaveformToFile` property so that you can process this waveform through the receiver of your choice.

```
WriteWaveformToFile = true;
```

This example processes data in chunks of 1 millisecond. That is, 1 millisecond of data is generated by the waveform generator and that 1 millisecond of data is processed by the receiver. This example works for a step time of one millisecond only.

```
stepTime = 1e-3; % In seconds
numSteps = (simulatedDataDuration/stepTime) + 1;
```

Initialize the parameters required to set up the channel. This example generates a GPS waveform from a simulated GPS constellation. As a starting point to simulate the GPS constellation, read the RINEX file.

```
rinexFileName = "GODS00USA_R_20211750000_01D_GN.rnx";
almanacFileName = "gpsAlmanac.txt";
rinexdata = rinexread(rinexFileName);
```

Initialize the receiver position to model the propagation channel based on its location. To estimate the receiver position, model the propagation delay accurately. The delay depends on the receiver position. This example models a stationary receiver and does not support a moving receiver.

```
rxlat = 39.021; % Latitude in degrees north
rxlon = -76.827; % Longitude in degrees east; negative longitude is degrees west
rxel = 19; % Elevation in meters
```

Because the GPS data bit rate is 50 bits per second and each C/A-code block is 1 millisecond, each bit consists of 20 C/A-code blocks. Initialize this parameter.

```
numCACodeBlocksPerBit = 20;
```

Initialize the properties required for the receiver. The Receiver must wait for some time before it can start receiving some meaningful data because of large delays modeled in the example. A typical GPS signal has a delay of 60 to 90 milliseconds. So, if receiver waits for 100 milliseconds, then it starts to process signals rather than pure noise.

```
rxWaitTime = 100; % Milliseconds
performInitSync = true; % Initially this must be set to true
```

```
% Initialize maximum number of tracking channels. Minimum of 4 tracking
% channels are needed for proper functioning of the GPS receiver.
maxNumTrackingChannels = 8;
```

```
% Noise bandwidth of each of the tracking loops
PLLNoiseBW = 90; % In Hz
FLLNoiseBW = 4; % In Hz
DLLNoiseBW = 3; % In Hz
```

```
% Bit synchronization parameters
```

```

isBitSyncComplete = zeros(maxNumTrackingChannels,1);
numBitsForBitSync = 100;
numWaitingStepsForBitSync = numCACCodeBlocksPerBit*numBitsForBitSync;
rxcntr = 1;

```

Position estimation requires, at minimum, that subframes 2 and 3 are decoded. The length of a subframe is 6 seconds. For this example, decode 48.5 seconds of data to ensure that subframes 2 and 3 are included.

```

minTimeForPosEst = 48.5; % In seconds
minStepsForPosEst = minTimeForPosEst/stepTime;
subframeDuration = 6; % In seconds
numStepsPerSubframe = subframeDuration/stepTime;

```

Initialize the physical constants required for the simulation.

```

c = physconst("LightSpeed"); % Speed of light in m/sec
fe = 1575.42e6; % GPS L1 frequency in Hz
Dt = 12; % Directivity of the transmit antenna in dBi
DtLin = db2pow(Dt);
Dr = 4; % Directivity of the receive antenna in dBi
DrLin = db2pow(Dr);
Pt = 44.8; % Typical transmission power of a GPS satellite in watts
k = physconst("boltzmann"); % Boltzmann constant in Joules/Kelvin
T = 300; % Room temperature in Kelvin
rxBW = 24e6; % Bandwidth in Hz
Nr = k*T*rxBW; % Thermal noise power in watts
rng default; % Initializing to default random number generation

```

Simulation Configuration

In this section, configure the example based on the parameters that are initialized in the Initialize Parameters on page 4-172 section.

Set up the satellite scenario based on the RINEX file.

```

% Initialize satellite scenario
sc = satelliteScenario;

% Set up the satellites based on the RINEX data
sat = satellite(sc,rinexdata,"OrbitPropagator","gps");
rx = groundStation(sc,rxlat,rxlon); % Set up the receiver
ac = access(sat,rx); % Calculate access between satellites and the receiver

% Get the list of satellites that are considered in satellite scenario from
% the RINEX data
indices = ones(length(sat),1);
for isat = 1:length(sat)
    ele = orbitalElements(sat(isat));

    % Check for match of time of applicability and pseudo-random noise
    % (PRN) IDs so that data from RINEX file is considered for waveform
    % generation
    indices(isat) = find(rinexdata.GPS(:,:).Toe == ele.GPSTimeOfApplicability & ...
        rinexdata.GPS(:,:).SatelliteID == ele.PRN);
end

```

Generate the navigation data to transmit. First generate a navigation configuration object using the helper function `HelperGPSRINEX2Config`. This helper function maps the parameters read from the RINEX file to the configuration parameters needed for navigation data generation.

```
% Generate navigation configuration object
navcfg = HelperGPSRINEX2Config(almanacFileName,rinexdata.GPS(indices,:));

[mintow,locmintow] = min([navcfg(:).HOWTOW]);

% The time of week (TOW) value is such that it must be 1 plus a multiple of
% 5 so that data of subframe 1 is always generated first.
mintow = ceil((mintow-1)/5)*5 + 1;

% HOWTOW is a counter that contains integer values. Incrementing by a value
% of 1 represents 6 seconds of data. The counter increases by a value of 1
% for each new subframe.
[navcfg(:).HOWTOW] = deal(mintow);
% Set the starting of frames based on mintow
firstsubframeID = mod(mintow-1,125) + 1;
frameID = ceil(firstsubframeID/5);
allFrameIDs = [frameID:25,1:(frameID-1)];
[navcfg(:).FrameIndices] = deal(allFrameIDs);

% Generate GPS navigation data
numNavBits = 37500; % Full GPS data length is 37500 bits
navdata = zeros(numNavBits,length(navcfg));
for isat = 1:length(navcfg)
    navdata(:,isat) = HelperGPSNAVDataEncode(navcfg(isat));
end

Get the satellite positions and velocities over time for accurate modeling of the Doppler, delay, and
power. For more information, see "Calculate Latency and Doppler in a Satellite Scenario" on page 1-
49.

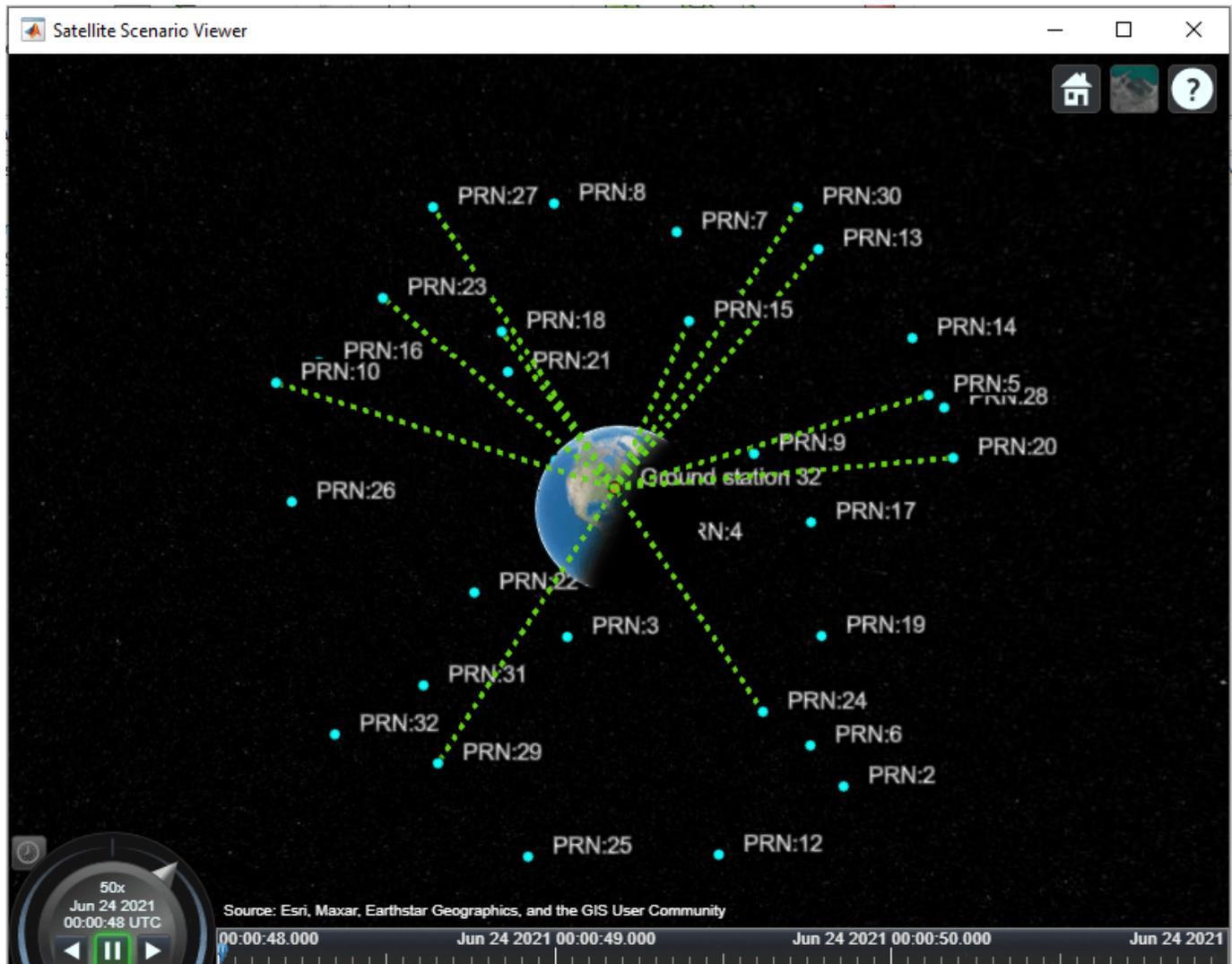
sc.StartTime = HelperGPSConvertTime(navcfg(locmintow).CEIDataset.WeekNumber, ...
    mintow*subframeDuration);
sc.StopTime = sc.StartTime + seconds(simulatedDataDuration);
sc.SampleTime = stepTime;

acstats = accessStatus(ac);
% This example runs for a maximum of 2 minutes of data and in that
% duration, access does not change. Hence, consider only the access status
% of the first sample time.
satindices = find(acstats(:,1));
numsat = length(satindices);

% Get the states of satellites in the sky over time
[initsatpos,satvel] = states(sat,"CoordinateFrame","ecef");

% Shuffle dimensions such that each row corresponds to one satellite data
initsatpos = permute(initsatpos,[3,1,2]);
satvel = permute(satvel,[3,1,2]);
```

This figure shows the satellites in the GPS constellation. The green dotted lines indicate access between the receiver and the GPS satellites. Use `play(sc)` to get this figure.



```

% Calculate Doppler offset over time for all the visible satellites
[az,el] = aer(sat(satindices),rx);
[~,satV] = states(sat(satindices),CoordinateFrame="geographic");
dir = cat(1,permute(cosd(el).*cosd(az),[3 2 1]), ...
    permute(cosd(el).*sind(az),[3 2 1]), ...
    permute(-sind(el),[3 2 1]));
dopV = permute(dot(satV,dir),[3 2 1]);
fo = ((c./(c-dopV))*fe).'; % Observed frequency
fShift = (fo - fe); % Doppler shift over time for all the visible satellites

% Compute the distance between satellites and receiver over time
satdist = zeros(numSteps,numsat);
for istep = 1:numSteps
    satdist(istep,:) = pseudoranges([rxlat,rxlon,rxel], ...
        initsatpos(satindices,:,istep),"RangeAccuracy",0);
end
delays = satdist/c;

% Power at receiver from free space pathloss equation

```

```
SqrtPr = sqrt(Pt*DtLin*DrLin)*(c./(4*pi*fo.*satdist));
timeinweek = mintow*6;
PRNIDs = [navcfg(:).PRNID];
disp("Available satellites - " + num2str(PRNIDs(satindices)))
```

```
Available satellites - 5 10 13 15 18 20 23 24 27 29 30
```

Generate C/A-codes for all the satellites in the GPS constellation. Use the codes for visible satellites while generating the waveform.

```
cacodes = gnssCACode(PRNIDs,"GPS");
caCodeBlockDuration = 1e-3; % Constant value
numcacodeblocks = stepTime/caCodeBlockDuration; % Each C/A-code block is of 1 millisecond duration

% To rate match the C/A-code with P-code, repeat each element 10 times
cacodesig1 = double(repelem(1-2*repmat(cacodes,numcacodeblocks,1),10,1));

% Rate match the C/A-code with the sampling rate
[upfac,downfac] = rat(samplingRate/10.23e6);
upcacodesig = repelem(cacodesig1,upfac,1);
cacodesig = upcacodesig(1:downfac:end,:);
```

Because P-code generation is a time-consuming process, and this example does not use the P-code for any receiver operations, use alternating values of 1 and 0 instead of the actual P-code. Divide the signal value by $\sqrt{2}$ because the I-branch on which P-code sits is attenuated by 3 dB as per IS-GPS-200 [1] on page 4-187.

```
numSamples = stepTime*samplingRate;
pcode = (1-2*repmat([1;0],numSamples/2,1))/sqrt(2);
```

Initialize the object needed to model the Doppler shift.

```
pfo = comm.PhaseFrequencyOffset(FrequencyOffsetSource = "Input port", ...
    SampleRate = samplingRate);
```

Initialize the object needed to model dynamic delays. The dynamic delay on the satellite signal is modeled in two steps. First, a static delay is modeled which does not change with time. Then, model the variable delay using `dsp.VariableFractionalDelay`. Initially while modeling the static delay, do not introduce the entire delay so that dynamic delay modeling handles the remaining delay. This dynamic delay is set to 2000 in this example.

```
% Initialize static delay object
dynamicDelayRange = 20000;
staticdelay = round(delays(1,:)*samplingRate - dynamicDelayRange);
if nnz(staticdelay<0)~=0
    staticdelay = zeros(1,numsat);
end
staticDelayObj = dsp.Delay("Length",staticdelay);
```

```
% Initialize Variable Fractional Delay object for modeling dynamic delay
vfd = dsp.VariableFractionalDelay("InterpolationMethod","Farrow", ...
    "MaximumDelay",65535);
```

Initialize the constellation diagram object.

```
rxconstellation = comm.ConstellationDiagram(1,ShowReferenceConstellation=false, ...
    Title="Constellation diagram of signal at the output of tracking");
```

Initialize the object that performs initial synchronization in the GPS receiver.

```
initialsync = HelperGPSInitialSynchronization;

% Initially search all 32 satellites
initialsync.PRNIDsToSearch = 1:32;
initialsync.SampleRate = samplingRate;
initialsync.CenterFrequency = 0           % Baseband signal

initialsync =
  HelperGPSInitialSynchronization with properties:
           PRNIDsToSearch: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ... ]
      CenterFrequency: 0
           SampleRate: 10230000
DetectionThresholdFactor: 1.2000
```

Initialize the baseband file writer.

```
if WriteWaveformToFile == 1
    bbWriter = comm.BasebandFileWriter("gpsBBWaveform.bb", samplingRate, 0);
end
```

Initialize the properties that are necessary for the end-to-end simulation chain.

```
% Properties that store outputs of initial synchronization
[doppleroffsets, codephoffsets] = deal(zeros(1, maxNumTrackingChannels));

% Properties required for storing outputs from tracking module
[accuPh, accuFq, accuFqErr, accuPhErr, accuIntegWave, accuDelay, accuDelayErr] = ...
    deal(zeros(numSteps, maxNumTrackingChannels));

% Properties to store outputs of bit synchronization and frame
% synchronization
[maxTransitionLocation, sampleCounter] = deal(zeros(maxNumTrackingChannels, 1));
syncidx = zeros(maxNumTrackingChannels, 1);

% Property to store output of data decoder
deccfg = cell(maxNumTrackingChannels, 1);

% Initialize maximum number of steps for which the simulation chain runs
maxSimSteps = 50/stepTime;
```

End-to-End Simulation Chain

This section contains three main steps:

- 1 Waveform generation
- 2 Propagation channel
- 3 Receiver signal processing algorithms

As part of the propagation channel, model the Doppler offset, delay, and noise. As part of the receiver, the processes involved include initial synchronization, tracking, bit synchronization, frame synchronization, and data decoding.

```
tic % Start of simulation
numAx = size(initsatpos, 2);
```

```

for istep = 1:numSteps
    %% Generate waveform
    bitidx = floor((istep-1)/numCACodeBlocksPerBit)+1;

    allpcode = repmat(pcode,1,numsat);
    % Get navigation bit of each satellite at the corresponding step time
    navbitsig = 1-2*navdata(bitidx,satindices);
    iqsig = (allpcode + 1j*cacodesig(:,satindices)).*navbitsig; % Implicit expansion

    %% Propagation channel
    % Model the channel which models Doppler, delay, path loss, and noise on
    % the signal

    % Introduce Doppler to the signal as a frequency offset
    dopsig = pfo(iqsig,fShift(istep,:));

    % Introduce variable fractional delay
    staticDelayedSignal = staticDelayObj(dopsig);
    leftoutDelay = delays(istep,:)*samplingRate - staticdelay; % Value must always be positive
    delayedSig = vfd(staticDelayedSignal,leftoutDelay);

    % Scale the delayed signal as per received power calculated
    rmsPow = rms(delayedSig);
    rmsPow(rmsPow==0) = 1; % To avoid division by zero
    scaledsig = SqrtPr(istep,:).*delayedSig./rmsPow;

    % Add up all the signals at the receiver
    resultsig = sum(scaledsig,2);

    % Generate noise
    noisesig = (wgn(numSamples,1,10*log10(Nr)) + 1j*wgn(numSamples,1,10*log10(Nr)))./sqrt(2);

    % Add constant thermal noise to the composite signal
    rxwaveform = resultsig + noisesig;

    % Scale the received signal for having unit power
    waveform = rxwaveform/rms(rxwaveform);

    if WriteWaveformToFile == 1
        bbWriter(waveform);
    end

    timeinweek = timeinweek + stepTime;

    %% Receiver
    % Because there are large delays (70 to 80 milliseconds) modeled on the
    % signal, start the receiver after some time only so as to process
    % valid signal instead of pure noise.
    if istep > rxWaitTime
        if performInitSync == 1
            performInitSync = 0;
            [y,corrval] = initialsinc(waveform); % Initial synchronization

            PRNIDsToSearch = y(y(:,4)==1,1).';
            doppleroffsets = y(y(:,4)==1,2).';
            codephoffsets = y(y(:,4)==1,3).';

            numdetectsat = length(PRNIDsToSearch);

```

```

if numdetectsat > maxNumTrackingChannels
    % Set this value to limit the number of tracking channels
    numdetectsat = maxNumTrackingChannels;
end

% Perform the required initialization of tracking modules for each
% channel and buffers to store the data.
disp("The detected satellite PRN IDs: " + num2str(PRNIDsToSearch))

% Plot the correlation plot for the first satellite
figure;
mesh(-10e3:500:10e3, 0:size(corrval,1)-1, corrval(:, :, PRNIDsToSearch(1)));
xlabel("Doppler Offset")
ylabel("Code Phase Offset")
zlabel("Correlation")
msg = ["Correlation Plot for PRN ID: " num2str(PRNIDsToSearch(1))];
title(msg)

framesyncbuffer = cell(1,numdetectsat);

% Create a cell array, where each element corresponds to a carrier
% tracking object.
carrierCodeTrack = cell(numdetectsat,1);
framesync = cell(numdetectsat,1);

% Update properties for each tracking loop
for isat = 1:numdetectsat
    carrierCodeTrack{isat} = HelperGPSCACodeCarrierTracker;
    carrierCodeTrack{isat}.SampleRate = samplingRate;
    carrierCodeTrack{isat}.CenterFrequency = 0;
    carrierCodeTrack{isat}.PLLNoiseBandwidth = PLLNoiseBW;
    carrierCodeTrack{isat}.FLLNoiseBandwidth = FLLNoiseBW;
    carrierCodeTrack{isat}.DLLNoiseBandwidth = DLLNoiseBW;
    carrierCodeTrack{isat}.PLLIntegrationTime = 1; % In milliseconds
    carrierCodeTrack{isat}.PRNID = PRNIDsToSearch(isat);
    carrierCodeTrack{isat}.InitialDopplerShift = doppleroffsets(isat);
    carrierCodeTrack{isat}.InitialCodePhaseOffset = codephoffsets(isat);

    % Initialize frame synchronization object
    framesync{isat} = HelperGPSLNAVFrameSynchronizer;
end
end

% Because it would be sufficient to get receiver position after
% running the simulation for 50 seconds of data, stop the loop
% after executing for 50 seconds of data. In the default run of the
% example, only 3 seconds of data is processed and this line is not
% needed. This check is when running the example for large data
% duration only (for at least 50 seconds of data).
if istep > maxSimSteps
    break;
end

for isat = 1:numdetectsat % Perform tracking for each satellite
    [integwvave, fqyerr, fqyoffset, pherr, phoffset, derr, dnco] = ...
        carrierCodeTrack{isat}(waveform);
end

```

```

    % Accumulate the values to see the results at the end
    accuFqyErr(rxcntr,isat) = fqyerr;
    accuFqy(rxcntr,isat) = fqyoffset;
    accuPhErr(rxcntr,isat) = pherr;
    accuPh(rxcntr,isat) = phoffset;
    accuIntegWave(rxcntr,isat) = sum(integwave);
    accuDelayErr(rxcntr,isat) = derr;
    accuDelay(rxcntr,isat) = dnco;
end

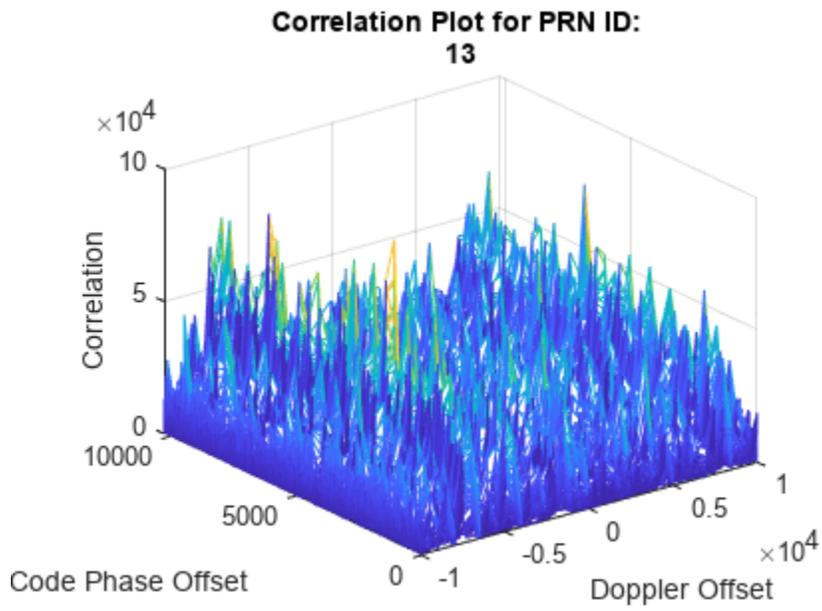
% Perform bit synchronization, frame synchronization, and data
% decoding if numWaitingStepsForBitSync of receiver steps are
% complete.
if rxcntr > numWaitingStepsForBitSync
    % For each detected satellite, perform bit synchronization,
    % frame synchronization, and data decoding
    for isat = 1:numdetectsat
        if ~isBitSyncComplete(isat)
            maxTransitionLocation(isat) = ...
                gnssBitSynchronize( ...
                    imag(accuIntegWave(1:numWaitingStepsForBitSync,isat)), ...
                    numCACodeBlocksPerBit);
            isBitSyncComplete(isat) = 1;
            sampleCounter(isat) = rxcntr - maxTransitionLocation(isat) + 1;
            framesyncbuffer{isat} = accuIntegWave( ...
                maxTransitionLocation(isat):end,isat);
        else % Perform frame synchronization and data decoding
            sampleCounter(isat) = sampleCounter(isat) + 1;
            framesyncbuffer{isat}(sampleCounter(isat)) = accuIntegWave(rxcntr,isat);
            if mod(sampleCounter(isat),numStepsPerSubframe) == 0
                samples = framesyncbuffer{isat}(sampleCounter(isat) - ...
                    numStepsPerSubframe+1:sampleCounter(isat));
                sym = mean(reshape(samples,numCACodeBlocksPerBit,[]));
                bits = imag(sym)<0;
                [syncidx(isat),rxsubframes,subframeIDs] = framesync{isat}(bits(:));
                if ~isempty(rxsubframes) % Then perform data decoding
                    deccfg{isat}.PRNID = PRNIDsToSearch(isat);
                    deccfg{isat} = HelperGPSNAVDataDecode(rxsubframes,deccfg{isat});
                end
            end
        end
    end
end

if mod(rxcntr,1000) == 0
    disp("Processed " + (rxcntr/1000) + " sec of data at the receiver.")
    rxconstellation(accuIntegWave(rxcntr-999:rxcntr,1)/ ...
        rms(accuIntegWave(rxcntr-999:rxcntr,1)))
end

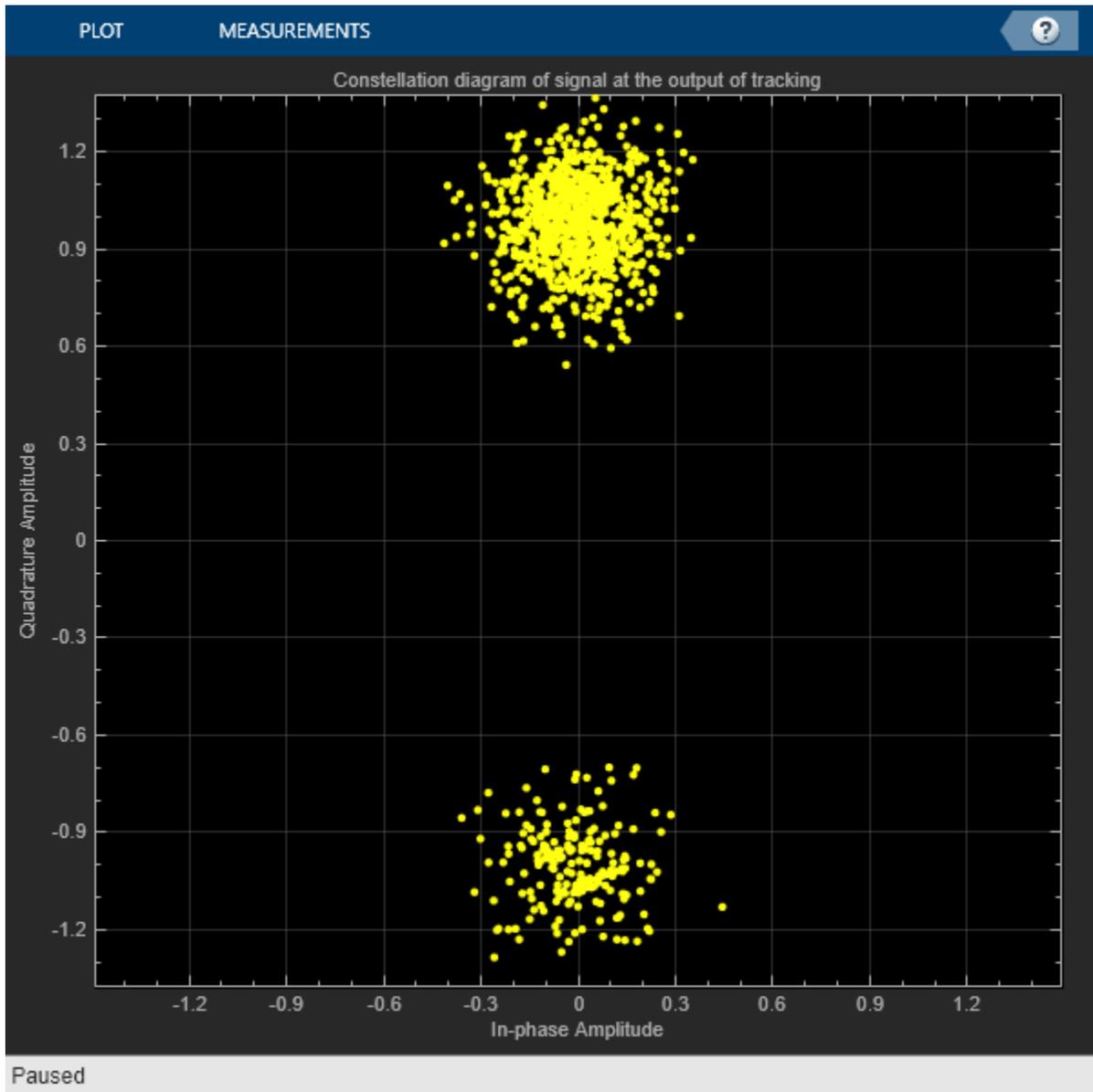
% Update rxcntr
rxcntr = rxcntr + 1;
end
end

The detected satellite PRN IDs: 13 15 24 27 5 29 18 20

```



Processed 1 sec of data at the receiver.
Processed 2 sec of data at the receiver.



```

if WriteWaveformToFile == 1
    release(bbWriter);
end
e2eTime = toc;
disp("End-to-End chain ran for " + e2eTime + " seconds.")

```

End-to-End chain ran for 209.3932 seconds.

GPS Receiver Position Estimation

To estimate the receiver position, you need to know the distances from at least 4 satellites to the receiver and the location of each satellite in the sky at the time of transmission. Receiver position calculation from the output of data decoder involves these steps.

- 1 Estimate transmission time of visible satellites at given receiver time: Transmission time estimates are the natural measurements of a GPS receiver, not the pseudo-ranges. The transmission time of a signal is a culmination of timing computations performed at initial synchronization, tracking, bit synchronization, frame synchronization, and data decoding.
- 2 Compute pseudo-ranges of all satellites: Because the receiver clock is not accurate, you can expect large offsets in the estimated pseudo-ranges. For the same reason that these estimates are not true ranges, they are called pseudo-ranges.
- 3 Estimate visible satellites location in sky at given transmission time: From the decoded data, you get the ephemeris data. This ephemeris data is valid for 2-6 hours [2] on page 4-187 and to calculate the accurate position of a satellite in the sky from this ephemeris data at a given time, use `gnssconstellation` (Navigation Toolbox).
- 4 Estimate receiver position: Estimate the receiver position using `receiverposition` (Navigation Toolbox).

Estimate the transmission time of visible satellites at a given receiver time.

```
caChipRate = 1.023e6; % In Hz
codeOffsetTime = codephoffsets(1:numdetectsat)/caChipRate;
rxcntr(rxcntr<=1) = 2; % So that rxcntr-1 is at least 1
trackingOffsetTime = accuDelay(rxcntr-1,1:numdetectsat)/caChipRate;
bitsyncTime = (maxTransitionLocation(1:numdetectsat) - 1)*caCodeBlockDuration;
framesyncTime = (syncidx(1:numdetectsat)-1)*numCACodeBlocksPerBit*caCodeBlockDuration;

% Calculate transmission time from these parameters
ttl = codeOffsetTime(:) - trackingOffsetTime(:) + bitsyncTime + framesyncTime;
ttl = ttl(syncidx~=0);
```

Estimate the pseudo-ranges.

```
trtemp = max(ttl);
row = (ttl-trtemp)*c;

% Include time of week decoded from the received navigation message
tow = zeros(maxNumTrackingChannels,1);
for isat = 1:maxNumTrackingChannels
    if isfield(deccfg{isat},"HOWTOW")
        tow(isat) = (deccfg{isat}.HOWTOW + 1)*6; % Each subframe has a duration of 6 seconds
    end
end
tow = tow(syncidx~=0);

% Offset the estimated transmission time. This offset is found empirically.
timeOffset = 0.74;
tt = ttl + tow + timeOffset;
```

Estimate the satellite position at the computed transmission time.

```
deccfg1 = deccfg(syncidx~=0);
% Estimate the satellite position only if receiver has processed at least
% minStepsForPosEst of data
isFullDataDecoded = true(length(tt),1);
if rxcntr > minStepsForPosEst
    [timeofweek,SatelliteID,Delta_n,M0,Eccentricity,sqrtA,Toe,Toc,Cis,Cic,Crs,Crc,Cus,Cuc, ...
     OMEGA0,i0,omega,OMEGA_DOT,IDOT,GPSWeek] = deal(zeros(4,1));
    reqCEIFields = ["WeekNumber","ReferenceTimeOfEphemeris","ReferenceTimeOfClock", ...
                    "SemiMajorAxisLength","MeanMotionDifference","MeanAnomaly","Eccentricity", ...
```

```

    "ArgumentOfPerigee", "Inclination", "InclinationRate", "HarmonicCorrectionTerms", ...
    "RateOfRightAscension", "LongitudeOfAscendingNode"];
for isat = 1:length(tt)
    timeofweek(isat) = tt(isat);
    isFullDataDecoded(isat) = (isfield(deccfg1{isat}, "PRNID")) && ...
        (all(isfield(deccfg1{isat}.CEIDataSet, reqCEIFields)));

    if isFullDataDecoded(isat)
        SatelliteID(isat) = deccfg1{isat}.PRNID;
        GPSWeek(isat) = deccfg1{isat}.CEIDataSet.WeekNumber;
        Toe(isat) = deccfg1{isat}.CEIDataSet.ReferenceTimeOfEphemeris;
        Toc(isat) = deccfg1{isat}.CEIDataSet.ReferenceTimeOfClock;
        sqrtA(isat) = sqrt(deccfg1{isat}.CEIDataSet.SemiMajorAxisLength);
        Delta_n(isat) = deccfg1{isat}.CEIDataSet.MeanMotionDifference*pi;
        M0(isat) = deccfg1{isat}.CEIDataSet.MeanAnomaly*pi;
        Eccentricity(isat) = deccfg1{isat}.CEIDataSet.Eccentricity;
        omega(isat) = deccfg1{isat}.CEIDataSet.ArgumentOfPerigee*pi;
        i0(isat) = deccfg1{isat}.CEIDataSet.Inclination*pi;
        IDOT(isat) = deccfg1{isat}.CEIDataSet.InclinationRate*pi;
        hterms = num2cell(deccfg1{isat}.CEIDataSet.HarmonicCorrectionTerms);
        [Cis(isat), Cic(isat), Crs(isat), Crc(isat), Cus(isat), Cuc(isat)] = deal(hterms{:});
        OMEGA_DOT(isat) = deccfg1{isat}.CEIDataSet.RateOfRightAscension*pi;
        OMEGA0(isat) = deccfg1{isat}.CEIDataSet.LongitudeOfAscendingNode*pi;
    end
end

transmissionTime = HelperGPSConvertTime(GPSWeek(isFullDataDecoded), timeofweek(isFullDataDecoded));
refTime = HelperGPSConvertTime(GPSWeek(:), Toc(:));
rx timetable = timetable(refTime, SatelliteID, Delta_n, M0, Eccentricity, sqrtA, ...
    Toe, Cis, Cic, Crs, Crc, Cus, Cuc, OMEGA0, i0, omega, ...
    OMEGA_DOT, IDOT, GPSWeek);
[satpos, satvel] = gnssconstellation(transmissionTime(1), rx timetable);
end

```

Estimate the receiver position, which requires two inputs — pseudo-ranges and the exact satellite position at the transmission time which is estimated at a given receiver time. You can compute these values only if the example runs for at least 48 seconds of data. Because simulating such data would take a long time, the pseudo-ranges and satellite position properties (obtained by running the example for 50 seconds) are stored in the MAT files. If you run the simulation for less than `minStepsForPosEst`, then these MAT files are loaded and the receiver position is computed. If you do not, the required properties are taken directly from the simulation chain.

```

if rxcntr <= minStepsForPosEst
    % The parameters that are loaded here are valid for the default
    % configuration of this example
    load receiverPositionProperties;

    % When loading the parameters for default configuration, update the
    % isFullDataDecoded variable with all ones to compute the receiver
    % position.
    isFullDataDecoded = true(length(row), 1);

    defaultRINEXFileName = "G0DS00USA_R_20211750000_01D_GN.rnx";
    defaultRxPos = [39.021, -76.827, 19];
    exampleRxPos = [rxlat, rxlon, rxel];

    if ~(strcmp(rinexFileName, defaultRINEXFileName) && isequal(defaultRxPos, exampleRxPos))

```

```

        warning("satcom:EndToEndGPSLNAVReceiverExample:InsufficientData", ...
            "Estimated receiver position may be different from what you provided" + ...
            " as the simulation didn't run for entire data." + ...
            " To get accurate receiver position, run the example" + ...
            " for at least 50 seconds of navigation data.");
    end
end

% Compute the receiver position
rxposest = receiverposition(row(isFullDataDecoded),satpos(isFullDataDecoded,:))

rxposest = 1x3

    39.0210  -76.8270  18.5258

estRxPosNED = lla2ned(rxposest,[rxlat,rxlon,rxel],'ellipsoid');
distanceError = vecnorm(estRxPosNED) % In meters

distanceError = 4.0582

```

Further Exploration

This example shows how to perform GPS receiver processing for the simulated satellite constellation for only 3 seconds. Extend the example to 50 seconds and estimate the receiver position to see how the GPS receiver works.

Use your own RINEX file to configure the example and estimate the receiver position from the waveform generated in this example.

Appendix

This example uses these data and helper files:

- `gpsAlmanac.txt` — Almanac data file downloaded from Navcen website
- `HelperGPSAlmanac2Struct.m` — Convert text file of almanac to structure
- `HelperGPSCACodCarrierTracker.m` — Carrier frequency and C/A-code phase tracker
- `HelperGPSCEIConfig.m` — Create configuration object for GPS navigation data
- `HelperGPSConvertTime.m` — Convert GPS week and time of week into datetime object and vice-versa
- `HelperGPSInitialSynchronization.m` — GPS Initial synchronization
- `HelperGPSLNAVDataDecode.m` — Decode the LNAV GPS data.
- `HelperGPSLNAVFrameSynchronizer.m` — Perform frame synchronization on the demodulated data.
- `HelperGPSLNAVWordDecode.m` — Decode each word of a subframe.
- `HelperGPSNAVDDataEncode.m` — Encode navigation data into bits from data that is in configuration object
- `HelperGPSNavigationConfig.m` — Create configuration object for GPS navigation data
- `HelperGPSRINEX2Config.m` — Convert the properties in RINEX file into navigation configuration object properties
- `receiverPositionProperties.mat` — Contains the properties required for calculation of receiver position

References

[1] IS-GPS-200, Rev: M. *NAVSTAR GPS Space Segment/Navigation User Interfaces*. May 21, 2021; Code Ident: 66RP1.

[2] Elliott D. Kaplan and C. Hegarty, eds., *Understanding GPS/GNSS: Principles and Applications*, Third edition, GNSS Technology and Applications Series (Boston ; London: Artech House, 2017).

See Also

Functions

`gnssBitSynchronize` | `gnssCACode` | `gnssconstellation` | `rinexread` | `receiverposition`

Objects

`gpsPCode` | `satelliteScenario`

Related Examples

- “GPS Receiver Acquisition and Tracking Using C/A-Code” on page 4-118
- “GPS Data Decode” on page 4-139
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-43
- “GPS Waveform Generation” on page 2-2
- “Calculate Latency and Doppler in a Satellite Scenario” on page 1-49

Over-the-Air Testing

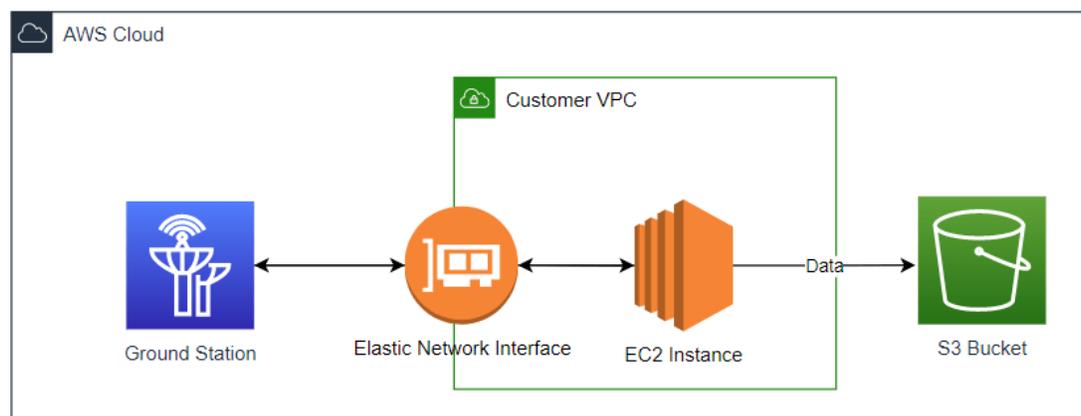
Capture Satellite Data Using AWS Ground Station

This example shows how to use Amazon® Web Services (AWS) Ground Station service from within MATLAB® to receive data from Earth observation satellite AQUA. AQUA (launched in 2002), is an Earth-orbiting National Aeronautics and Space Administration (NASA) scientific research satellite that studies precipitation, evaporation, and cycling of water. You can capture data from other satellites for which you have access permission by changing the satellite information. Using this example, you can capture satellite data as radio frequency (RF) in-phase quadrature (I/Q) samples and as demodulated and decoded data. You can further process and analyze this captured data using the Communications Toolbox™ and Image Processing Toolbox™.

Introduction

AWS Ground Station is a service that enables you to manage satellite communications and process data without the need to build or maintain your own ground station infrastructure. You can get more information about AWS Ground Station service and its capabilities from the AWS Ground Station website.

AWS Ground Station currently supports low earth orbit (LEO) and medium earth orbit (MEO) satellites. These satellites are visible from the ground station for only few minutes during each pass due to their orbital cycles. Communication is possible when the satellites are within the line of sight of a ground station. AWS Ground Station establishes contact with the satellites, and then receives, demodulates, and decodes RF signals. Ground Station then pushes data to the receiver Elastic Compute Cloud (EC2) instance as a VITA 49 stream. This figure shows the end-to-end connection of the data received into Virtual Private Cloud (VPC) from the AWS Ground Station for the scheduled duration and then pushes the data to the AWS Simple Storage Service (S3) bucket.



AWS Services and Costs

This example uses these AWS services, some of which can incur costs on your AWS account. For cost estimates, see the respective pricing page websites for each of the AWS services that this example uses.

- AWS Ground Station pricing
- AWS CloudFormation pricing
- Amazon EC2 pricing (this example uses the m5.4xlarge instance type)
- Amazon VPC pricing
- Amazon CloudWatch pricing
- Amazon SNS pricing
- Amazon S3 pricing

AWS lets you visualize, understand, and manage your AWS costs and usage over time. For more details, see the [AWS Cost Explorer AWS website](#).

All the services created through this example can be released at any point of time. For more details, see [Delete AWS Cloud Resources on page 5-14 section](#).

Set Up Access to AWS

To capture data from satellites using the AWS Ground Station service, you must have an AWS account with permission to AWS services that this example uses. This section describes one-time setup steps to gain access to the AWS Ground Station service. If you have already completed this procedure, skip this section.

Create an IAM User

If you already have an Identity and Access Management (IAM) user, skip this section.

- 1** Sign up for an AWS root account. For details, see the [AWS website](#).
- 2** Open the IAM console at this website: [IAM Console](#).
- 3** Click **Users** in the IAM console, and then click **Add User**. To complete creating an IAM user, follow the AWS add-user steps, which the next five subsections.

Step 1: Set user details

Follow the prompts to set user details. This figure shows an example of setting user details and AWS access type. Because this data capture example uses programmatic calls to AWS, so you must select **Programmatic access** and **AWS Management Console access**.

Add user



Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type*
- Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
 - AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

- Console password*
- Autogenerated password
 - Custom password

Require password reset User must create a new password at next sign-in

* Required

[Cancel](#)

[Next: Permissions](#)

Step 2: Set permissions

Follow the prompts to set permissions. This figure shows an example of setting permissions for your IAM user. Because you must have administration permissions to access AWS services, select the AdministratorAccess policy from the **Attach existing policies directly** policies.

Add user

- 1
- 2
- 3
- 4
- 5

Set permissions

 Add user to group

 Copy permissions from existing user

 Attach existing policies directly

Create policy

↻

Filter policies Showing 698 results

	Policy name	Type	Used as
<input checked="" type="checkbox"/>	 AdministratorAccess	Job function	Permissions policy (2)
<input type="checkbox"/>	 AdministratorAccess-Amplify	AWS managed	None
<input type="checkbox"/>	 AdministratorAccess-AWSElasticBeanstalk	AWS managed	None
<input type="checkbox"/>	 AlexaForBusinessDeviceSetup	AWS managed	None
<input type="checkbox"/>	 AlexaForBusinessFullAccess	AWS managed	None
<input type="checkbox"/>	 AlexaForBusinessGatewayExecution	AWS managed	None
<input type="checkbox"/>	 AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS managed	None
<input type="checkbox"/>	 AlexaForBusinessPolyDelegatedAccessPolicy	AWS managed	None

Cancel

Previous

Next: Tags

Step 3: Add tags

Follow the prompts to add tags. This figure shows an example of adding tags to your IAM user.

Add user

- 1
- 2
- 3
- 4
- 5

Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
<input type="text" value="Add new key"/>		

You can add 50 more tags.

Cancel

Previous

Next: Review

Step 4: Review

Follow the prompts to review the details of your IAM user and Create user. This figure shows an example of IAM user details, permission summary, and attached policies for this example.

Add user



Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	myusername
AWS access type	Programmatic access and AWS Management Console access
Console password type	Autogenerated
Require password reset	Yes
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

Tags

No tags were added.

Cancel

Previous

Create user

Step 5: Complete

This figure shows an example of viewing and downloading your IAM user security credentials. You can get your credentials using any of these options.

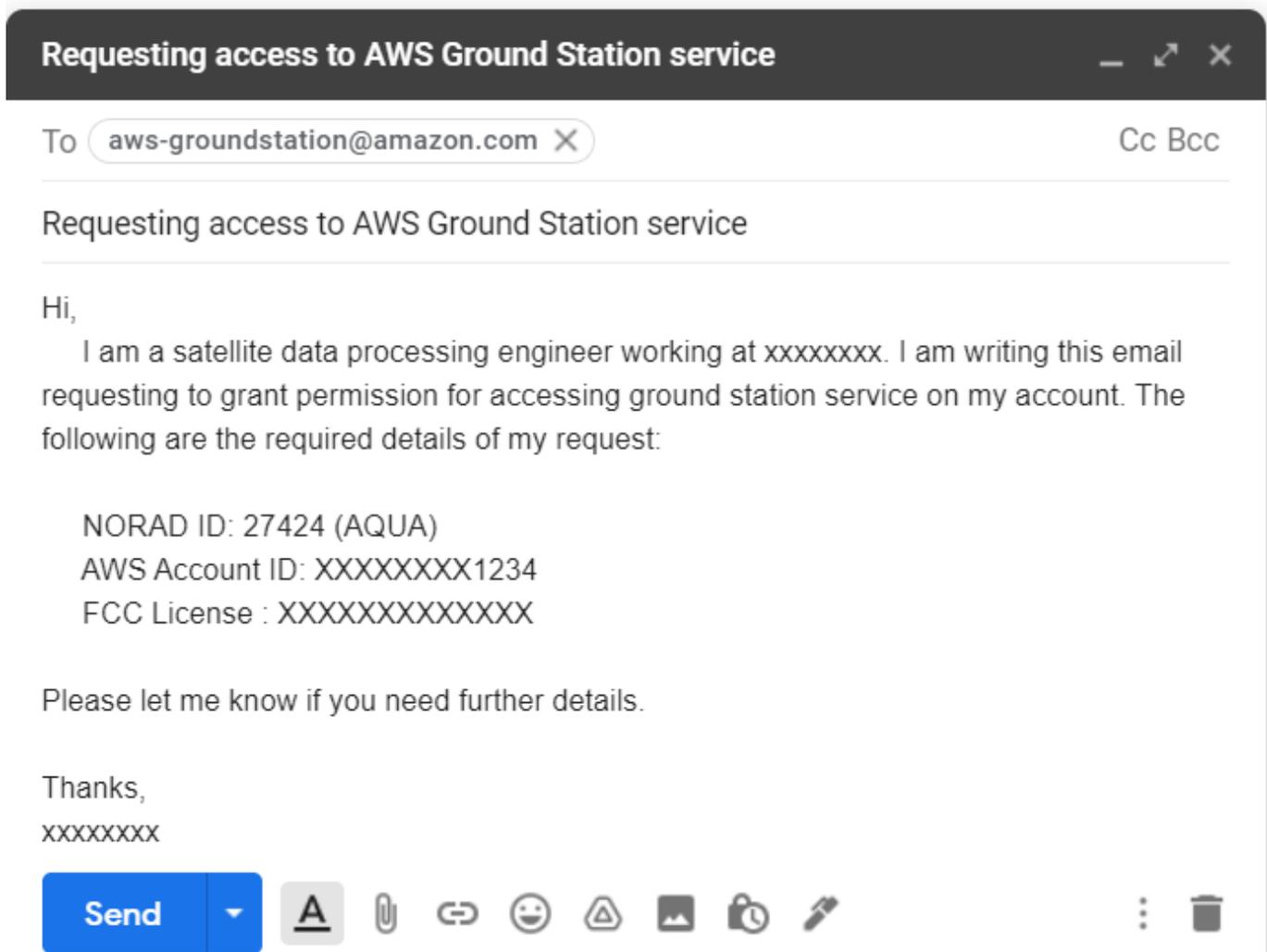
- Click **Download .csv**, to download the credentials file, which includes your IAM username, access key ID, secret access key.
- Click **Send email**, to receive an email with instructions about how to sign into the AWS Management Console.
- Copy the access key ID, secret access key manually.

For more information about IAM users, see the [Creating an IAM user in your AWS account](#) and [Managing access keys for IAM users](#) AWS websites.

Request Access to Ground Station

To get access to ground station services on your account, email aws-groundstation@amazon.com with the NORAD ID, FCC license information (see the FCC licensing), and AWS account ID.

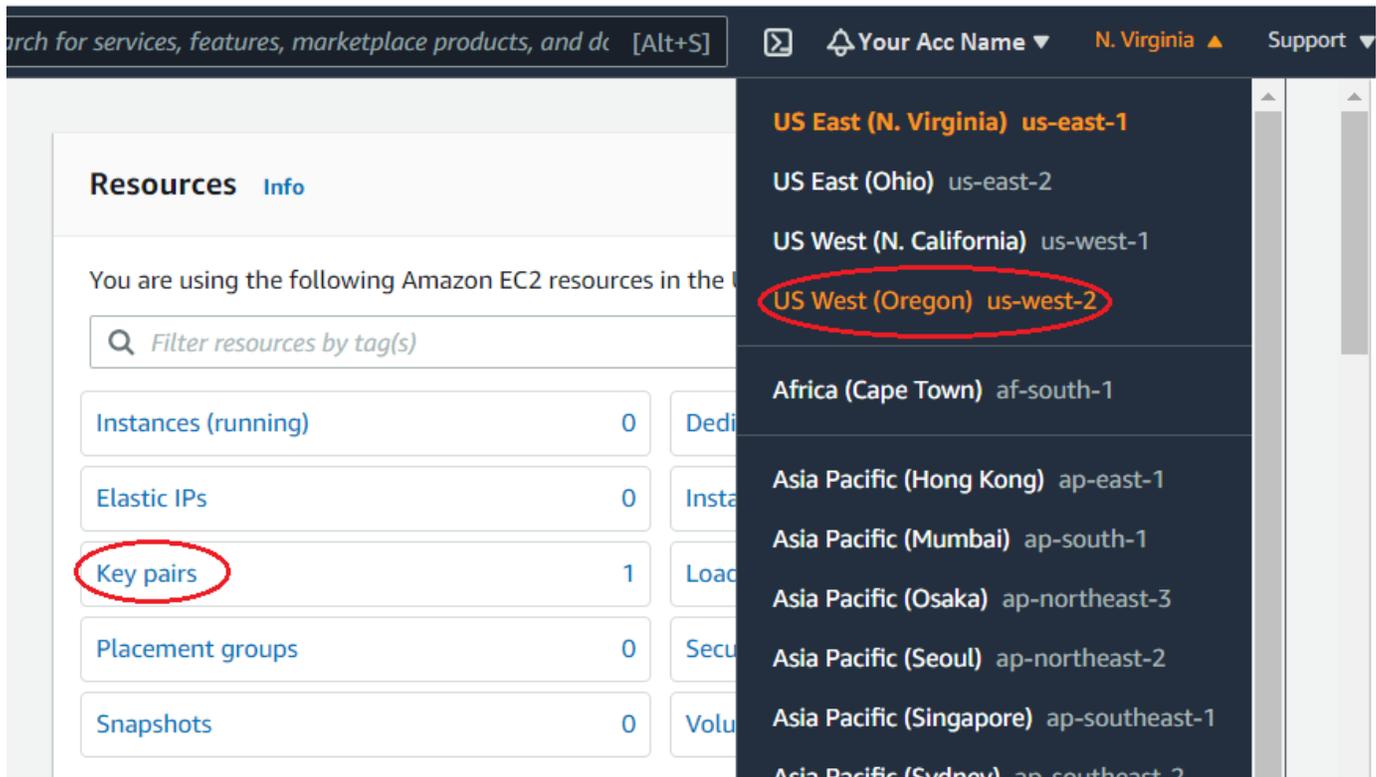
This figure shows a sample email to get access to AWS Ground Station service.



Create EC2 Key Pair

Create an EC2 key pair for the AWS region where you plan to receive data. The EC2 Key pair is used to connect to your EC2 instances for debugging and to get the status of the data capture.

- 1 Open the Amazon EC2 console at this website: [EC2 Console](#).
- 2 Click the region list on the navigation bar. For more information on regions, see the [Regions](#).



- 1 Select **Key pairs**, and then click **Create key pair**. For more information on EC2 key pairs, see the Create a key pair using Amazon EC2 AWS website.
- 2 Specify a unique name for your EC2 key pair and select the file format as `.pem` to save the private key. Click **Create key pair** to create an EC2 key pair.
- 3 The Key pair is created, and the private key file automatically downloads to your host PC. Save the file. This point in the process is the only point at which you have the option to save the private key file. If you lose this file or do not save it, you must repeat the process again to create new key pair.

EC2 > Key pairs > Create key pair

Create key pair

Key pair
A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

File format

pem
For use with OpenSSH

ppk
For use with PuTTY

Tags (Optional)
No tags associated with the resource.

You can add 50 more tags.

Create S3 Bucket

Create an S3 bucket, which stores captured satellite data, by following these steps.

- 1 Open the S3 console at this website: [S3 Console](#).
- 2 Choose **Create bucket**.
- 3 Specify a bucket name, select the corresponding AWS region, and click **Create bucket**.

Amazon S3 > Create bucket

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.

Cancel

For more information, see the [Create your first S3 bucket](#).

Configure AWS CLI

Download and install the latest AWS command line interface (CLI). For details, see the [AWS Command Line Interface](#). You can control multiple AWS services using this AWS CLI.

Configure the AWS CLI using .csv file that you downloaded in [Create IAM user](#) on page 5-3 section. This file contains your IAM username, AWS access key ID, and AWS secret access key.

```
% Specify your IAM username, (for example, IAMUserName = 'yourusername')
IAMUserName =  ;
% Provide your credentials file location
% (for example, C:\Work\new_user_credentials.csv)
csvCredentialsFile = fullfile('C:', 'Work', 'new_user_credentials.csv');
HelperAWSConfig(IAMUserName, "csvFileName", csvCredentialsFile);
```

Alternatively, if you are using single sign on (SSO) or federated login, then the AWS credentials file is created in a default directory. In such cases, run this code instead.

```
HelperAWSConfig(IAMUserName);
```

Get Satellite List

To list the satellites accessible in the given AWS region (that is, the same region where you created the EC2 key pair in the Create EC2 Key Pair on page 5-7 section), run this code.

```
% Provide the region name that you plan to use
region =  ;
% List the satellites available in given region
[satelliteID,groundStationName] = HelperAWSListSatellites(region);
```

Data Capture Setup

Create infrastructure to capture the data (such as leasing, computing resources), and load and run the software on to the leased machines.

- 1 Lease two EC2 instances (the t2.micro and m5.4xlarge instance type).
- 2 Copy the capture script and data transfer scripts to the EC2 instance.
- 3 Configure the Simple Notification Service (SNS) subscription email to get notifications at various stages of the data capture process.

Provide the EC2 key pair full file path, S3 bucket name, and notification email address to the `HelperAWSDataCaptureSetup` helper function. The resources created during this process are tagged with a tag key value pair that can be used to track or control resources. To set up data capture, run this code.

```
% AQUA satellite ID 27424
noradID =  ;
% Provide the full path of your PEM SSH key pair name that you generated in
% the 'Create EC2 Key Pair' section
% (for example, ec2KeyPairFile = 'C:\Work\my-ssh-key-us-west-2.pem')
ec2KeyPairFile = fullfile('C:', 'Work', 'your-ec2-key-pair-us-west-2.pem');
% Provide S3 bucket name to store the satellite data that was created in
% the 'Create S3 Bucket' section, (for example, s3BucketName = 'your-s3-bucket-name')
s3BucketName =  ;
% Provide your email address to receive the notifications
% (for example, youremail@domain.com)
notificationEmail =  ;
% Set up the data capture
[tagKeyName,tagValueName] = HelperAWSDataCaptureSetup(noradID,ec2KeyPairFile, ...
    s3BucketName,notificationEmail,region);
```

During data capture setup, you receive two SNS topic subscription confirmation requests from the AWS Notification service. Confirm the subscription to receive further notifications.

This figure shows a sample email notification from the AWS Notification service. In the email notification, click the "Confirm subscription" link to receive SNS notifications.

AWS Notification - Subscription Confirmation Inbox x



AWS Notifications <no-reply@sns.amazonaws.com>

to me ▾

You have chosen to subscribe to the topic:

arn:aws:sns:us-west-2:11112223333:MyTopic

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

After you complete the data capture set up, you receive a SNS notification. This figure shows an example of this notification.



AWS Notifications <no-reply@sns.amazonaws.com>

to me ▾



Data capture set up has been completed successfully. The details of data capture set up as follows. Ground station stack: GroundStationSetup-2021-Jul-01-21-13, Linux Bastion Stack:DataCaptureSetup-2021-Jul-01-21-02 and Region: us-west-2

Schedule Contact

Check the availability of the selected satellite in the given region and then schedule a contact. To get satellite available start time and available duration, run this code.

```
% Select a ground station (for example, groundStation = 'Ohio 1')
```

```
groundStation = ;
```

```
% Lists the available contacts in the given region and ground station
[contactStartTime,contactDuration] = HelperAWSListContacts(noradID, ...
    groundStation,tagKeyName,tagValueName,region);
```

You can schedule a contact by using the `HelperAWSScheduleContact` helper function with these input parameters.

- Capture start time: Specify your start time of the contact.
- Capture duration: Specify the duration (in seconds) to capture data.

```
% Provide a start time for the contact
```

```
% (for example, captureStartTime ='26-Apr-2021 13:47:12')
```

```
captureStartTime = contactStartTime(1);
```

```
% Provide a contact duration (in seconds), making sure that the contact is available for
% the given duration
```

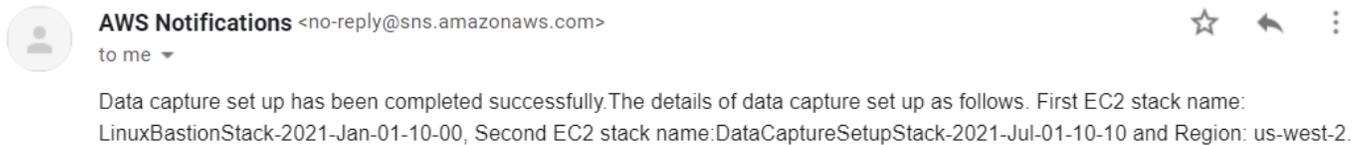
```
captureDuration = ; % in seconds
```

```
% Schedule contact. This function reserves the contact for the specified time
% and captures satellite data for the specified duration.
```

```
contactArn = HelperAWSScheduleContact(noradID,captureDuration, ...
    captureStartTime,groundStation,tagKeyName,tagValueName,region);
```

Email Notifications

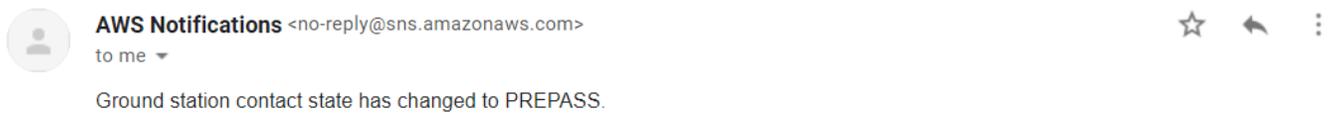
After scheduling a contact, you receive an AWS notification email regarding the status of your contact. This figure shows an example of successful reservation of the contact.



After Ground Station establishes contact with the satellite, you receive notifications for possible contact states like PREPASS, PASS, POSTPASS, COMPLETED and FAILED.

- **PREPASS:** Approximately 2 minutes prior to contact start, you receive an email notification indicating an upcoming pass. This figure shows an example of the ground station contact state changed to PREPASS.

Ground station contact state changed to PREPASS ▸ Inbox x 🖨️ 🔗



- **PASS:** The duration of the contact you scheduled, you receive an SNS event indicating that the capturing satellite data has started. This figure shows an example of the ground station contact state changed to PASS.

Ground station contact state changed to PASS ▸ Inbox x 🖨️ 🔗



- **POSTPASS:** After one minute of pass time, you receive an SNS event indicating the pass has finished. This figure shows an example of the ground station contact state changed to POSTPASS.

Ground station contact state changed to POSTPASS ▸ Inbox x 🖨️ 🔗



- **COMPLETED:** When satellite data captured successfully for the capture duration, you receive an SNS event indicating that contact has been completed successfully. This figure shows an example of the ground station contact state changed to COMPLETED.

Satellite data captured successfully Inbox x

AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾



The captured data files available in '/root/data-receiver/' directory of the ground station EC2 instance and these data files has been uploaded to specified S3 bucket.

- **FAILED:** If any failures occur during contact state change, then you receive an SNS event indicating that the contact has failed. In this case, contact MathWorks® technical support. For details, see contact support on the MathWorks website. This figure shows an example of the ground station contact state changed to FAILED.

Ground Station contact has FAILED Inbox x

AWS Notifications <no-reply@sns.amazonaws.com>
to me ▾



Ground station contact state has FAILED. Contact MathWorks technical support (<https://www.mathworks.com>).

List Scheduled Contacts

List your scheduled contacts by running this code.

```
% List scheduled contacts
[scheduledContactList] = HelperAWSListScheduledContacts(region);
```

Cancel Contact

You can cancel your scheduled contact before the scheduled time. Before doing so, review the terms and conditions for canceling a scheduled contact, as well as the associated charges and pricing. To cancel a scheduled contact, uncomment and run this code.

```
% HelperAWSCancelContact(contactArn, region);
```

Get Captured Data from S3 Bucket

The raw data files (like `downlinkDemodDecode__.bin` and `downlinkData__.bin`) are pushed to the S3 bucket. To download the raw data from the S3 bucket, uncomment and run this code.

```
% % Specify the raw data file in S3 bucket
% satelliteRawDataFileName = ..._MWLCFILEREf_matlab/help/examples/satcom/win64/CaptureSatellite
% HelperAWSGetDataFromS3(s3BucketName, satelliteRawDataFileName);
```

Make another Data Capture

To schedule another data capture, repeat the steps from Data Capture Setup on page 5-11 section.

Delete AWS Cloud Resources

After you have finished capturing data, deleting all resources is recommended to avoid incurring further cost. This action deletes all of resources that are tagged with the tag key value pair. To delete the resources, uncomment and run this code.

```
% HelperAWSCleanResources(region, tagKeyName, tagValueName);
```

In case of further errors, you can delete the resources from the AWS Management Console. For example, to delete the AWS Cloud Formation Stack, follow these steps.

- 1 On the AWS Cloud Formation Console website, and select the stack to delete. The stacks created in this example are named with a prefix `LinuxBastionStack`, `DataCaptureSetupStack`, and are tagged with a prefix of `MW_App` and `SatComToolbox`.
- 2 Click **Actions**, and then click **Delete** from the menu that appears.

The screenshot shows the AWS CloudFormation console interface. At the top, there are buttons for 'Refresh', 'Delete' (circled in red), 'Update', 'Stack actions', and 'Create stack'. Below these is a search bar labeled 'Filter by stack name' and a dropdown menu set to 'Active'. A table lists the stacks:

Stack name	Status	Created time	Description
DataCaptureSetupStack-2021-Jul-07-14-36	CREATE_COMPLETE	2021-07-07 14:36:37 UTC+0530	-
LinuxBastionStack-2021-Jul-07-14-25	CREATE_COMPLETE	2021-07-07 14:25:06 UTC+0530	LinuxBast (qs-1qup6 remove)

Close AWS Account

Only the AWS account root user can close an AWS account. For more information, see the Closing your AWS account AWS website.

Further Exploration

During data capture at the scheduled time, you can connect to the EC2 instance by using this SSH command and observe the data transfer status, such as bandwidth or bytes transferred.

`% Get the SSH command to connect to the EC2 instance`

```
sshCommand = HelperAWSGetSSHCmd(ec2KeyPairFile, tagKeyName, tagValueName, region);
```

Login to the EC2 instance using the `sshCommand` to see the logs from your AWS CLI. You can track the installation and configuration of the software on the EC2 instance by checking the logs at the location `/var/log/user-data.log`.

Other Satellites

You can collect data from any of these satellites by changing the satellite NORAD ID in the data capture setup function.

- NOAA 20 JPSS 1 (NORAD ID 43013): This satellite was launched in 2017 and orbits at an altitude of 825 km. It carries five sensors that are designed to study land and water.
- SUOMI NPP (NORAD ID 37849): This satellite was launched in 2011 and orbits at an altitude of 883 km. It carries four sensors that are designed to provide climate measurements.
- TERRA (NORAD ID 25994): This satellite was launched in 1999 and orbits at an altitude of 705 km. It carries five sensors that are designed to study the surface of the Earth.

Appendix

The example uses these helper functions:

- HelperAWSConfig.m: Sign to AWS CLI through MATLAB by providing IAM username
- HelperAWSListSatellites.m: List available satellites in region and checks if the region has ground station
- HelperAWSDataCaptureSetup.m: Set up Linux bastion and ground station stacks and then launches the EC2 instances respectively for the stacks
- HelperAWSListContacts.m: List contacts present in specific region
- HelperAWSScheduleContact.m: Schedule contact
- HelperAWSListScheduledContacts.m: List scheduled contacts
- HelperAWSCancelContact.m: Cancel scheduled contact
- HelperAWSGetSSHCmd.m: Get secure shell (SSH) command to ssh into ground station Linux machine
- HelperAWSCleanResources.m: Clean all stacks and unused resources in region

See Also

Related Examples

- “VITA 49 File Reader” on page 5-17

VITA 49 File Reader

This example shows how to read signal time data packets and the associated metadata (context packet data) from a VITA 49.2 format file into the MATLAB® workspace.

Introduction

A VITA 49.2 file consists of signal data streams and associated metadata in structured packet format, as defined in ANSI/VITA-49.2-2017. This example demonstrates how to extract context and signal time data packets from a VITA 49.2 file. The VITA Radio Transport (VRT) standard, also known as the VITA 49 protocol, defines a standard format for sending and receiving digitized messages between radio frequency (RF) systems and related equipment.

The VRT is a packet based protocol to convey digitized signal data and metadata (or context data) pertaining to different reference points within a radio receiver. The metadata includes radio front-end parameters, such as RF center frequency, bandwidth, intermediate frequency (IF), center frequency, sampling rate, gain, and location of the satellite.

Packet Formats

A packet in a VITA 49 file starts with a packet prologue consisting of a mandatory packet header followed by a list of fields as determined by the packet type present in the packet header. The packet header includes packet type, packet size and additional information fields interpreted based on packet type. These additional information fields indicate how to interpret the rest of the packet prologue and the packet contents. VITA 49 specification defines eight different packet formats. This example supports only signal time data (packet type 0 to 3) and context data (packet type 4 and 5) packet formats. The packet header is as shown in the following figure.

VRT Packet Header

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fields	Packet Type				C	Indicators				TSI	TSF	Packet Count				Packet Size																

This table shows the eight different packet formats along with the data information they carry.

Packet Type	Data Information
0	Signal Data Packet without Stream ID
1	Signal Data Packet with Stream ID
2	Extension Data Packet without Stream ID
3	Extension Data Packet with Stream ID
4	Context Packet
5	Extension Context Packet
6	Command Packet
7	Extension Command Packet

Signal Data Packet

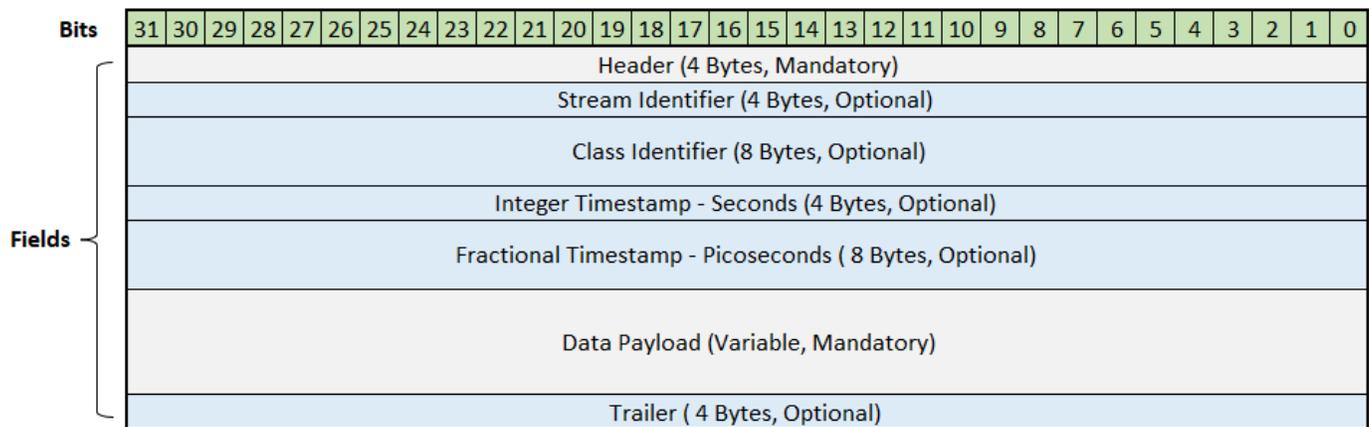
Signal data packets convey digitized IF and RF signals. Signal data packet is further divided into signal time data packet or signal spectral data packet. Signal time data packet represents the signal in time-domain as a sequence of samples sampled at a constant rate. Signal spectral data is a sequence of samples describing the signal frequency or spatial-domain. This example decodes only signal time data packets.

Signal Time Data Packet

Signal time data packets convey digitized IF and RF signals. Signal time data packets encapsulates variable-sized blocks of in-phase quadrature (IQ) data, along with a 32-bit trailer to convey additional information about the state of the receiver at the time the samples were obtained. For example, if the system was being overdriven this would be reported by an indicator in the trailer.

Signal time data may be either real or complex samples and can also be magnitude or power representations of a signal for spectral data. This example only supports complex, cartesian data formats; complex, polar data formats are not supported. The signal data packets in VITA 49.2 are organized as signal data packet streams. A signal data packet stream is identified by a stream identifier (SID) and is a sequence of signal data packets of same signal data packet class. Signal data packet class specifies the type of signal data and the packet content structure. Multiple streams can be formed from the same class, each stream has its own unique SID. The signal data packet structure is as shown in the following figure.

Signal Data Packet Memory Layout



The integer timestamp field (ITF) is a 32-bit number which specifies the reference point time where the data sample is collected with a resolution of 1 second. It may be used to convey UTC time, GPS time, or some user-specified timecode.

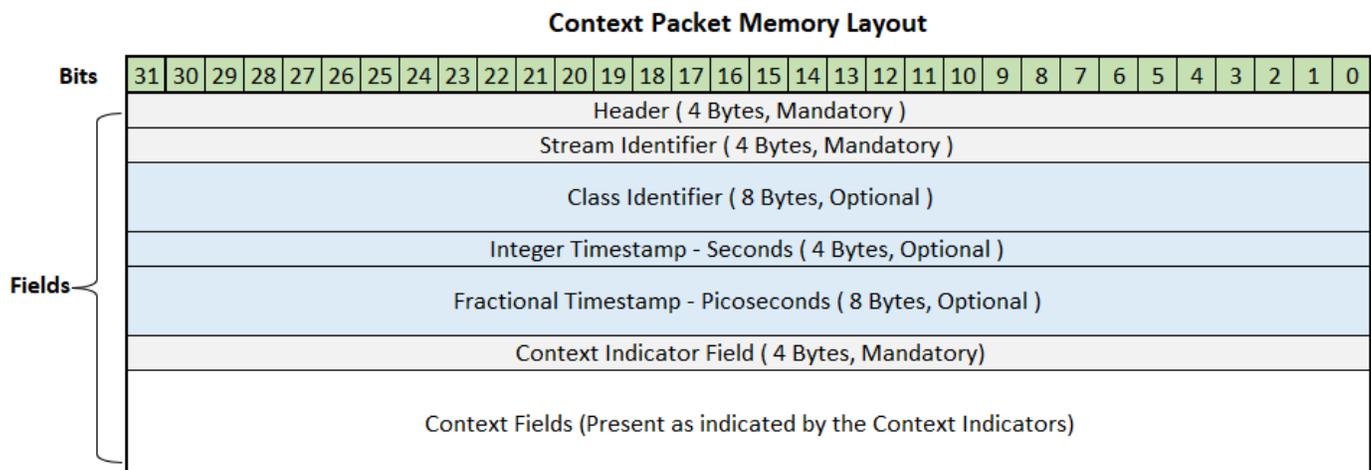
The fractional timestamp field (FTF) is a 64-bit integer that represents the reference point time with a better resolution than ITF. FTFs are classified into three types: sample-count, real-time, and free-running count timestamps.

- The sample-count and real-time timestamps are used to add resolution to the ITF, allowing it to cover a time span of years with precision of sample period or one picosecond.
- The free-running count timestamp provides an incrementing sample count from starting time and is not related to the ITF.

Context Packet

Context packets convey the spatial information and the information of the receiver settings. The information is of variable size, depending on the number of fields used, out of a total of 25 fields. The fields used in a given packet are communicated by the 32-bit context indicator field which precedes the context fields.

Context packets are sent whenever there is a change in receiver settings or spatial information. These packets are typically transmitted at some periodic interval to ensure that a VITA 49.2 receiver can recover from any miscommunication or loss of communication. Such periodic updates typically involve retransmitting all fields required by the application. The context packet structure is as shown in the following figure.



The bits in the context indicator field (CIF) indicates which of the optional context fields are present in the context packet corresponding to each field. The CIF definitions shown in the following figure.

Context Indicator Field Definitions

Bit Position	Context Fields	Used	Number of Words
31	Context Field Change Indicator	Yes	1 bit
30	Reference Point Identifier	Yes	1
29	Bandwidth	Yes	2
28	IF Reference Frequency	Yes	2
27	RF Reference Frequency	Yes	2
26	RF Reference Frequency Offset	Yes	2
25	IF Band Frequency Offset	Yes	2
24	Reference Level	Yes	1
23	Gain	Yes	1
22	Over Range Count	Yes	1
21	Sample Rate	Yes	2
20	Timestamp Adjustment	Yes	2
19	Timestamp Calibration Time	Yes	1
18	Temperature	Yes	1
17	Device Identifier	Yes	2
16	State and Event Indicator	Yes	1
15	Signal Data Packet Payload Format	Yes	2
14 ... 0	Not defined	No	---

Only fields that are marked 'Yes' in the CIF definitions figure will be decoded in this example.

Read VITA 49 Packets

Use the `vita49Reader` object to read files containing VITA 49 formatted data. The object contains information about the VITA 49 file and enables you to read the upcoming signal or context packets from the file.

Specify the VITA 49 file and set the `OutputTimestampFormat` property to `seconds` or `datetime` in the `vita49Reader` object.

Configure a VITA 49 file reader object to read the `VITA49SampleData.bin` data file. The sample data file contains a total of 40 packets. The first packet 10 packets are context packets and the remaining packets are signal data packets.

```
fileName = "VITA49SampleData.bin";
outputTimestampFormat = ;
vita49ReaderObj = vita49Reader(fileName,OutputTimestampFormat=outputTimestampFormat);
```

Read the next upcoming VITA 49 packet in the file by using `read` function on the VITA 49 file reader object. This next packet can be either signal data packet or context packet, depending on the value of the `PacketType` field in the packet structure.

```
firstPacket = read(vita49ReaderObj) % read next packet
```

```
firstPacket = struct with fields:
    PacketType: 4
    StreamID: 0
    ClassID: "7C386C0000"
    IntegerTimestampType: "GPS"
    IntegerTimestampValue: 1625215654
    FractionalTimestampType: "real time"
    FractionalTimestampValue: 0
    RawBytes: [84x1 uint8]
    ContextFieldChangeIndicator: 1
    ReferencePointIdentifier: [1x0 double]
    Bandwidth: 15500025
    IFRferenceFrequency: 0
    RFFrequency: 1.2100e+09
    RFFrequencyOffset: [1x0 double]
    IFBandOffset: [1x0 double]
    ReferenceLevel: -30
    Gain: 36
    OverRangeCount: [1x0 double]
    SampleRate: 17222250
    TimestampAdjustment: [1x1 struct]
    TimestampCalibrationTime: [1x0 double]
    StateAndEventIndicator: [1x1 struct]
    SignalDataPayloadFormat: [1x1 struct]
```

```
nextPacket = read(vita49ReaderObj); % read next packet
```

You can use the `read` function to read a certain number of packets by specifying `NumPackets`.

- When you specify only `NumPackets` and do not specify `PacketType`, the function returns signal data packets and context packets separately.
- When you specify `NumPackets` and `PacketType`, the function returns only the information of the specified packet type (signal data or context). The packet type values from 0 to 3 indicate a signal data packet, a packet type value of 4 and 5 indicates a context packet.

```
n = 39; % Number of packets required
vita49ReaderObj = vita49Reader(fileName);
[signalDataPackets, contextPackets] = read(vita49ReaderObj, NumPackets=n);
anotherPacket = read(vita49ReaderObj)
```

```
anotherPacket = struct with fields:
    PacketType: 1
    StreamID: 0
    ClassID: "7C386C0000"
    PadBitCount: 0
    IntegerTimestampType: "GPS"
    IntegerTimestampValue: 1625215654
    FractionalTimestampType: "real time"
    FractionalTimestampValue: 901216160000
    RawBytes: [1472x1 uint8]
    IQSamples: [361x1 double]
```

```
Trailer: [1x1 struct]
```

Read Signal Data Packets

To read only signal data packets, set `PacketType` to `signal data`.

```
vita49ReaderObj = vita49Reader(fileName);
packetType =  ;
% Skips the context packets until signal data packet arrives
signalDataPacket = read(vita49ReaderObj,PacketType=packetType)

signalDataPacket = struct with fields:
    PacketType: 1
    StreamID: 0
    ClassID: "7C386C0000"
    PadBitCount: 0
    IntegerTimestampType: "GPS"
    IntegerTimestampValue: 1625215654
    FractionalTimestampType: "real time"
    FractionalTimestampValue: 900000344000
    RawBytes: [1472x1 uint8]
    IQSamples: [361x1 double]
    Trailer: [1x1 struct]
```

Read Context Packets

To read only context packets from the VITA 49 formatted file into the MATLAB workspace, set `PacketType` to `context`.

```
vita49ReaderObj = vita49Reader(fileName);
% Seeks to the next context Packet
packetType =  ;
contextPacket = read(vita49ReaderObj,PacketType=packetType)

contextPacket = struct with fields:
    PacketType: 4
    StreamID: 0
    ClassID: "7C386C0000"
    IntegerTimestampType: "GPS"
    IntegerTimestampValue: 1625215654
    FractionalTimestampType: "real time"
    FractionalTimestampValue: 0
    RawBytes: [84x1 uint8]
    ContextFieldChangeIndicator: 1
    ReferencePointIdentifier: [1x0 double]
    Bandwidth: 15500025
    IFRferenceFrequency: 0
    RFFrequency: 1.2100e+09
    RFFrequencyOffset: [1x0 double]
    IFBandOffset: [1x0 double]
    ReferenceLevel: -30
    Gain: 36
    OverRangeCount: [1x0 double]
    SampleRate: 17222250
    TimestampAdjustment: [1x1 struct]
```

```

TimestampCalibrationTime: [1x0 double]
StateAndEventIndicator: [1x1 struct]
SignalDataPayloadFormat: [1x1 struct]

```

Reading Data from RF Capture

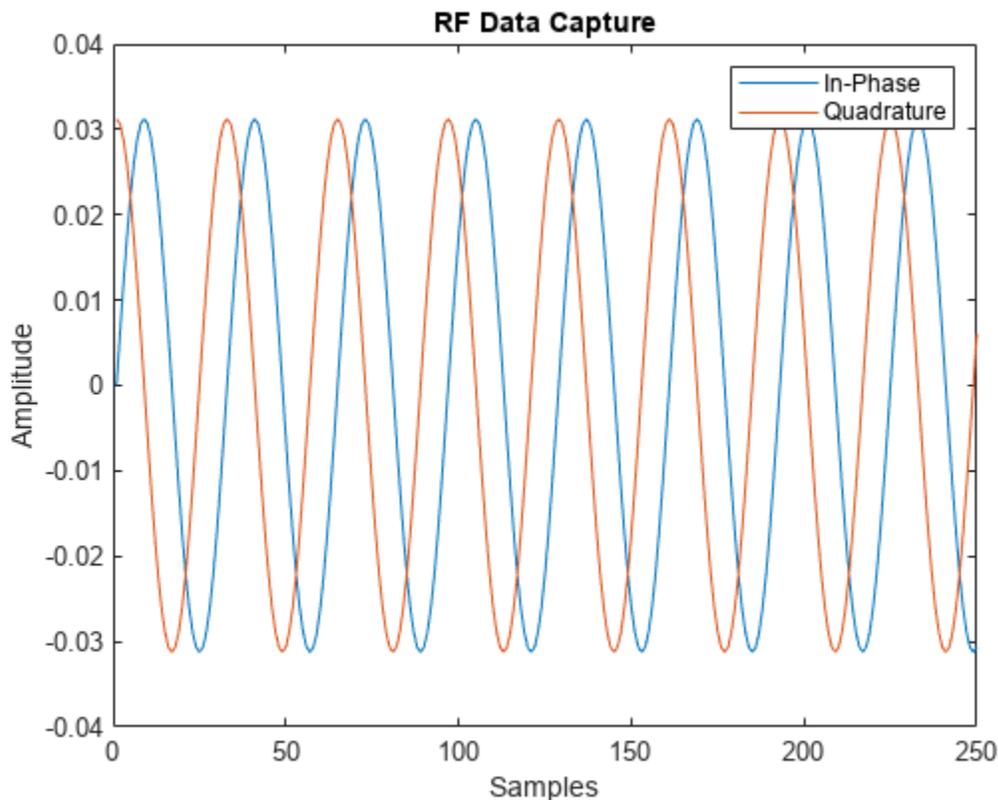
The VITA49SineWaveData.bin data file contains IQ samples of sine wave data captured from RF.

Configure a VITA 49 file reader object to read the VITA49SineWaveData.bin data file. The sample data file consists of a total of nine packets. The first packet is context packet and the remaining are signal data packets.

```

fileName = "VITA49SineWaveData.bin";           % VITA 49 file with sine wave data
vita49ReaderObj = vita49Reader(fileName);
numPackets = 9;                                % First packet is context packet in the file
[dataPackets,~] = read(vita49ReaderObj,NumPackets=numPackets);
iqSamples = dataPackets(1).IQSamples;          % Plot first packet data
plot(real(iqSamples));
hold on;
plot(imag(iqSamples));
xlabel("Samples")
ylabel("Amplitude")
title("RF Data Capture")
legend(["In-Phase","Quadrature"])

```



Reset to First Packet

Reset the position of the VITA 49 file reader to the first packet of the VITA 49 file.

```
reset(vita49ReaderObj);  
readFirstPacket = read(vita49ReaderObj);  
clear vita49ReaderObj
```

References

[1] VITA 49 website: <https://www.vita.com>

See Also

Related Examples

- “Capture Satellite Data Using AWS Ground Station” on page 5-2

Code Generation and Deployment

- “What is C Code Generation from MATLAB?” on page 6-2
- “DVB-S2 HDL Transmitter” on page 6-4
- “DVB-S2 HDL PL Header Recovery” on page 6-16
- “DVB-S2 HDL Receiver” on page 6-36
- “GPS HDL Acquisition and Tracking Using C/A Code” on page 6-46
- “DVB-S.2 System Simulation Using a GPU-Based LDPC Decoder System Object” on page 6-57

What is C Code Generation from MATLAB?

You can use Satellite Communications Toolbox together with MATLAB® Coder™ to:

- Create a MEX file to speed up your MATLAB application.
- Generate ANSI®/ISO® compliant C/C++ source code that implements your MATLAB functions and models.
- Generate a standalone executable that runs independently of MATLAB on your computer or another platform.

In general, the code you generate using the toolbox is portable ANSI C code. In order to use code generation, you need a MATLAB Coder license. For more information, see “Get Started with MATLAB Coder” (MATLAB Coder).

Using MATLAB Coder

Creating a MATLAB Coder MEX file can substantially accelerate your MATLAB code. It is also a convenient first step in a workflow that ultimately leads to completely standalone code. When you create a MEX file, it runs in the MATLAB environment. Its inputs and outputs are available for inspection just like any other MATLAB variable. You can then use MATLAB tools for visualization, verification, and analysis.

The simplest way to generate MEX files from your MATLAB code is by using the `codegen` function at the command line. For example, if you have an existing function, `myfunction.m`, you can type the commands at the command line to compile and run the MEX function. `codegen` adds a platform-specific extension to this name. In this case, the “mex” suffix is added.

```
codegen myfunction.m  
myfunction_mex;
```

Within your code, you can run specific commands either as generated C code or by using the MATLAB engine. In cases where an isolated command does not yet have code generation support, you can use the `coder.extrinsic` command to embed the command in your code. This means that the generated code reenters the MATLAB environment when it needs to run that particular command. This is also useful if you want to embed commands that cannot generate code (such as plotting functions).

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See “Generating Standalone C/C++ Executables from MATLAB Code” (MATLAB Coder) for more information.

C/C++ Compiler Setup

Before using `codegen` to compile your code, you must set up your C/C++ compiler. For 32-bit Windows platforms, MathWorks® supplies a default compiler with MATLAB. If your installation does not include a default compiler, you can supply your own compiler. For the current list of supported compilers, see Supported and Compatible Compilers on the MathWorks website. Install a compiler that is suitable for your platform, then read “Setting Up the C or C++ Compiler” (MATLAB Coder).

After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

Functions and System Objects That Support Code Generation

For an alphabetized list of features supporting C/C++ code generation, see [Satellite Communications Toolbox - Functions and Objects Filtered by C/C++ Code Generation](#).

See Also

Functions

`codegen` | `mex`

More About

- [“Code Generation Workflow” \(MATLAB Coder\)](#)
- [Generate C Code from MATLAB Code Video](#)

DVB-S2 HDL Transmitter

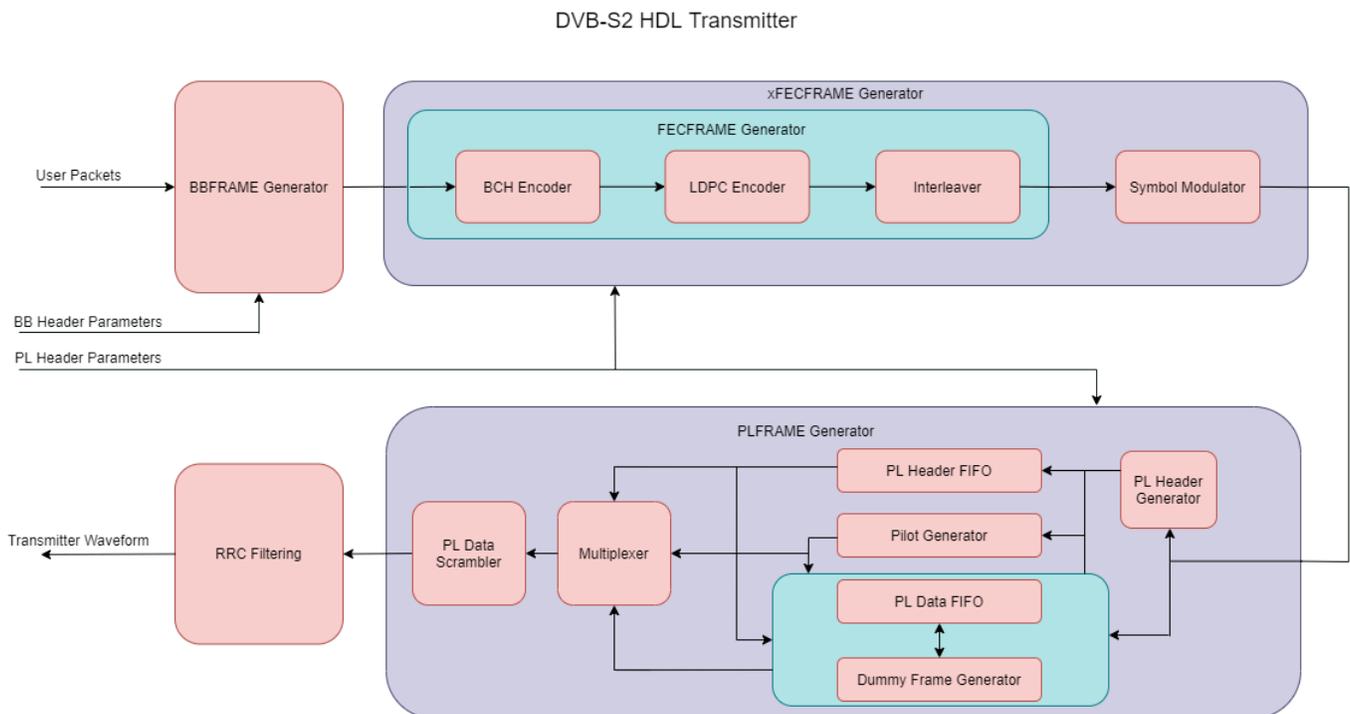
This example shows how to implement a digital video broadcast satellite second generation (DVB-S2) transmitter using Simulink® blocks that are optimized for HDL code generation and hardware implementation.

From this example, you can generate a DVB-S2 transmitter waveform using these steps:

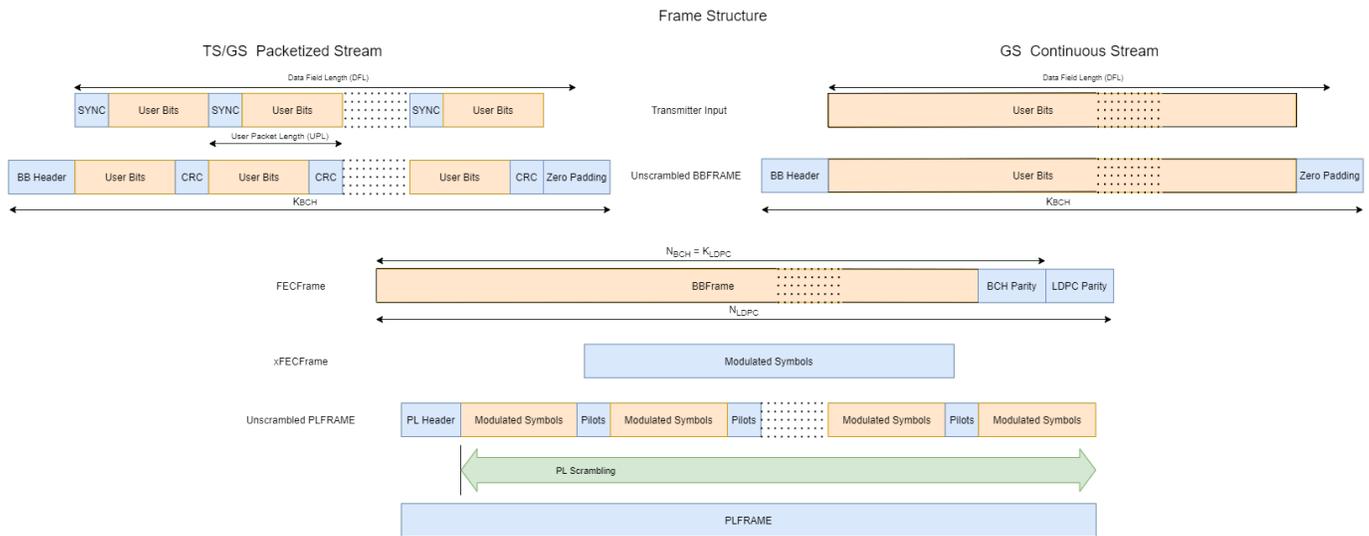
- 1 Generate a baseband frame (BBFRAME).
- 2 Encode using Bose-Chaudhuri-Hocquenghem (BCH) and low-density parity-check (LDPC) codes.
- 3 Interleave, modulate, and generate a physical layer frame (PLFRAME).
- 4 Insert dummy frames.
- 5 Pulse shape the symbols in frames using a root raised cosine (RRC) filter.

Model Architecture

The BBFRAME Generator block generates BBFRAME from the user packets. The FECFRAME Generator block encodes the BBFRAME using BCH and LDPC coding, and interleaves the encoded frame to generate an FECFRAME. The Symbol Modulator block generates the xFECFRAME by mapping the FECFRAME bits to modulation symbols. The PLFRAME Generator block generates the PLHEADER, pilot symbols, and dummy frames, and multiplexes them with the xFECFRAME bits. The PL Data Scrambler block scrambles the multiplexer output to generate a PLFRAME. The RRC Filtering block pulse shapes the PLFRAME symbols to generate the transmitter waveform.



This figure shows the transmitter frame structure.



File Structure

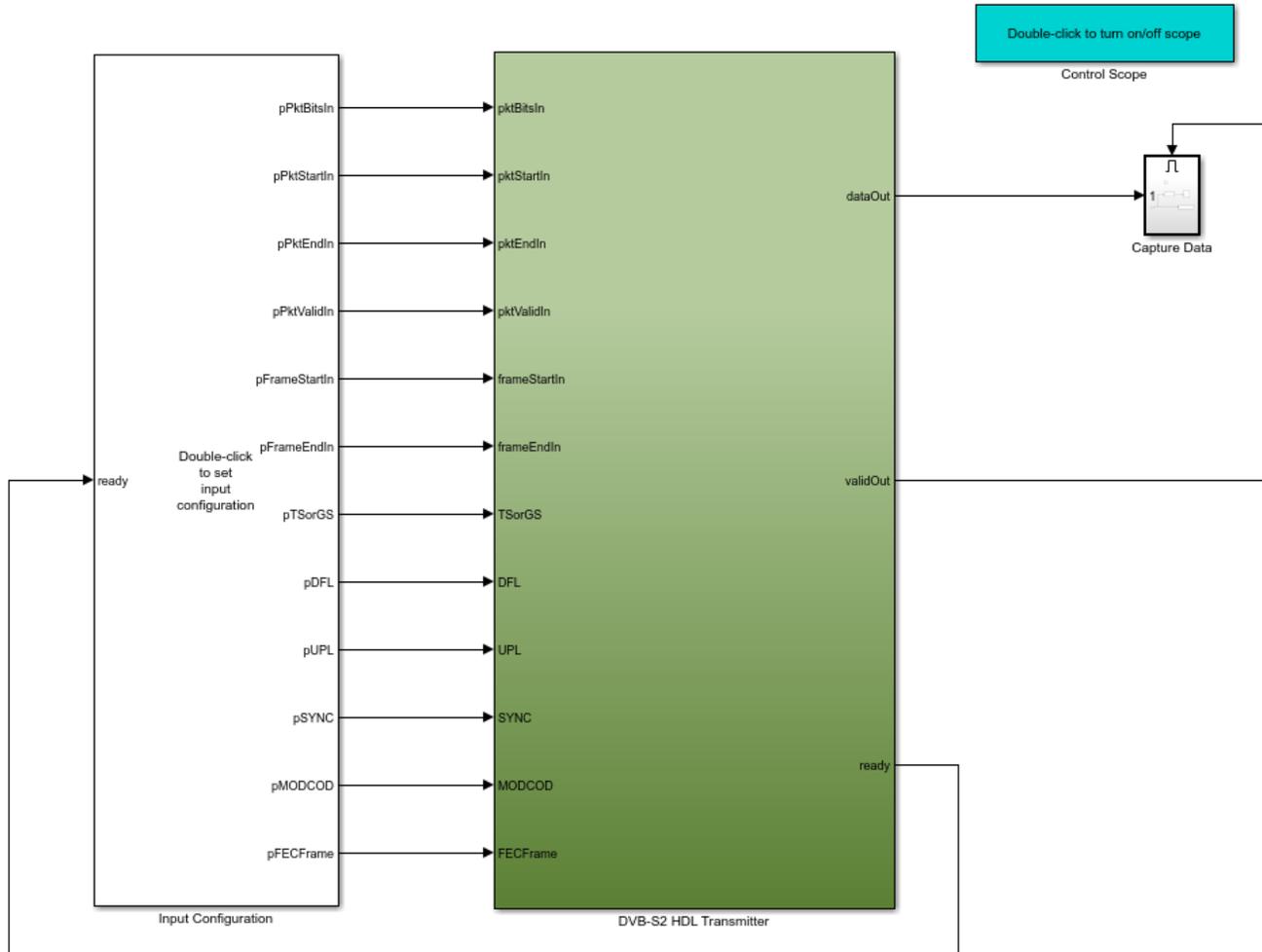
This example uses supporting files.

- `dvbs2hdlTransmitter` — Model for DVB-S2 HDL transmitter.
- `dvbs2hdlTransmitterCore` — Model reference for the transmitter design.
- `dvbs2hdlTxParameters` — Functions that generates parameters for the `dvbs2hdlTransmitterCore` model.
- `dvbs2hdlTxInit` — Script that initializes the `dvbs2hdlTransmitter` model.
- `dvbs2hdlTransmitterVerify` — Script that generates reference transmitter waveform using `dvbs2WaveformGenerator` function and compares the reference waveform with the simulated transmitter output.

System Interface

The figure shows the top-level overview of the `dvbs2hdlTransmitter` model.

DVB-S2 HDL Transmitter



Copyright 2022 The Mathworks, Inc.

Model Inputs

- **pktBitsIn** — Input packet bits, specified as a Boolean scalar.
- **pktStartIn** — Control signal indicating the start of each packet, specified as a Boolean scalar.
- **pktEndIn** — Control signal indicating the end of each packet, specified as a Boolean scalar.
- **pktValidIn** — Control signal indicating whether the **pktBitsIn** is valid, specified as a Boolean scalar.
- **frameStartIn** — Control signal indicating the start of each frame, specified as a Boolean scalar.
- **frameEndIn** — Control signal indicating the end of each frame, specified as a Boolean scalar.
- **TSorGS** — Input stream format, specified as a 2 bit unsigned real integer.

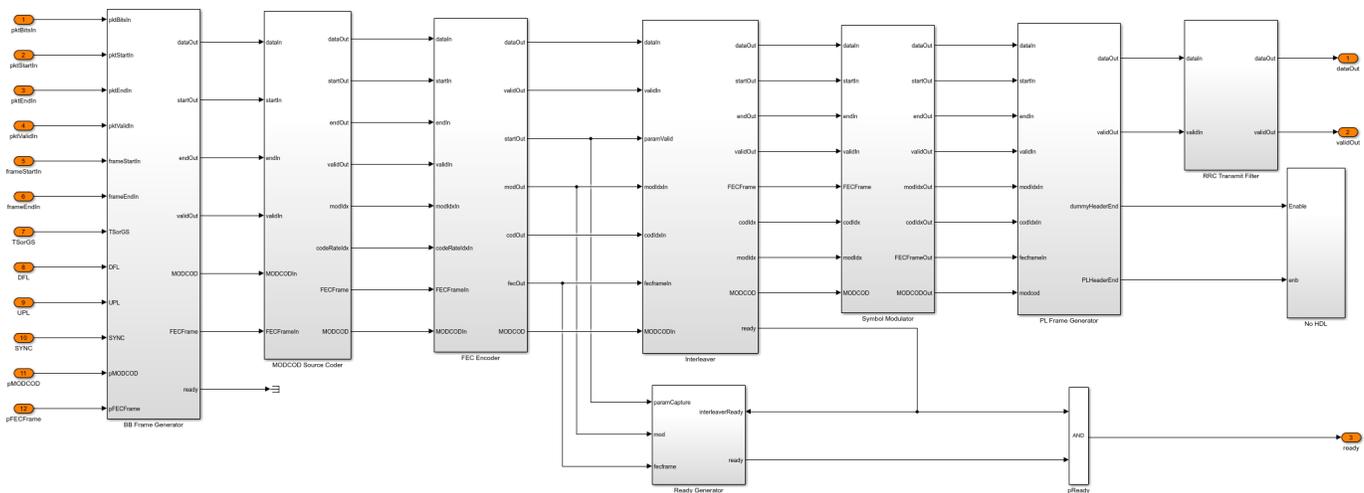
- **DFL** — Data field length (DFL), specified as a 16 bit unsigned real integer.
- **UPL** — User packet length (UPL), specified as a 16 bit unsigned real integer.
- **SYNC** — SYNC word, specified as a 8 bit unsigned real integer.
- **MODCOD** — MODCOD, specified as a 5 bit unsigned real integer.
- **FECFrame** — FECFrame type, specified as a Boolean scalar.

Model Outputs:

- **dataOut** — Transmitter output, returned as a 18 bit complex scalar.
- **validOut** — Control signal indicating whether the **dataOut** is valid, returned as a Boolean scalar.
- **ready** — Control signal indicating whether the transmitter is ready for the input, returned as a Boolean scalar.

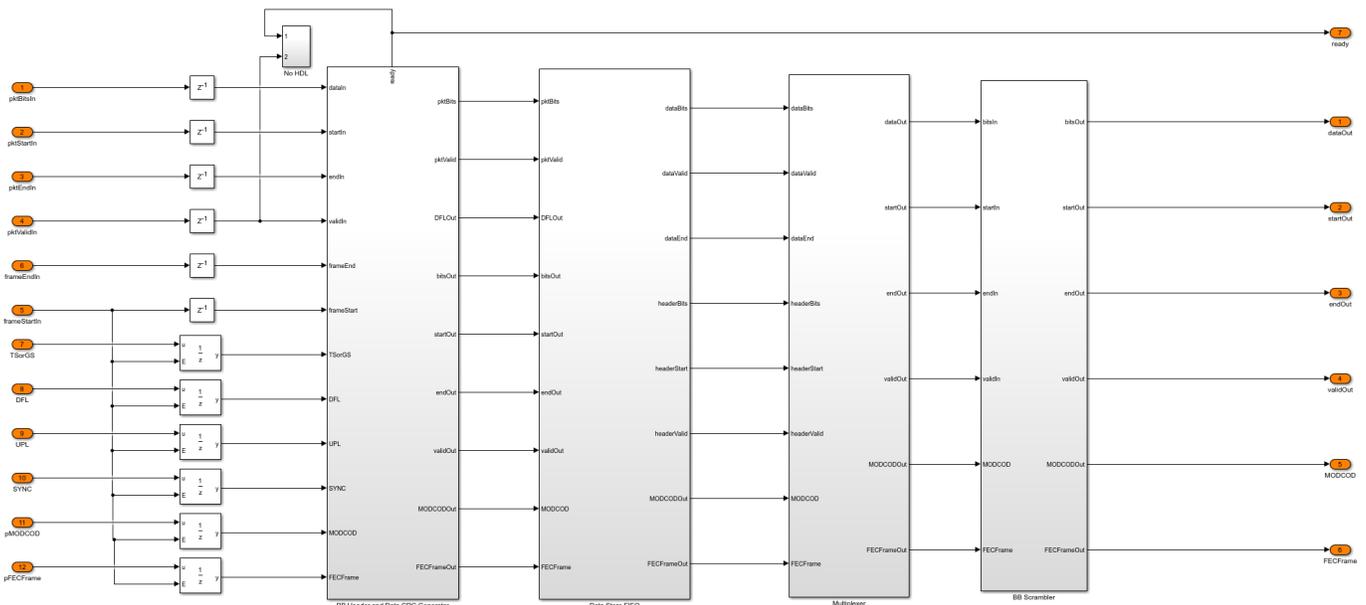
Model Structure

This figure shows the structure of the DVB-S2 HDL transmitter subsystem. The subsystem comprises BB Frame Generator, FEC Encoder, DVB-S2 HDL Interleaver, Symbol Modulator, PL Frame Generator, PL Data Scrambler, and RRC Transmit Filter subsystems.



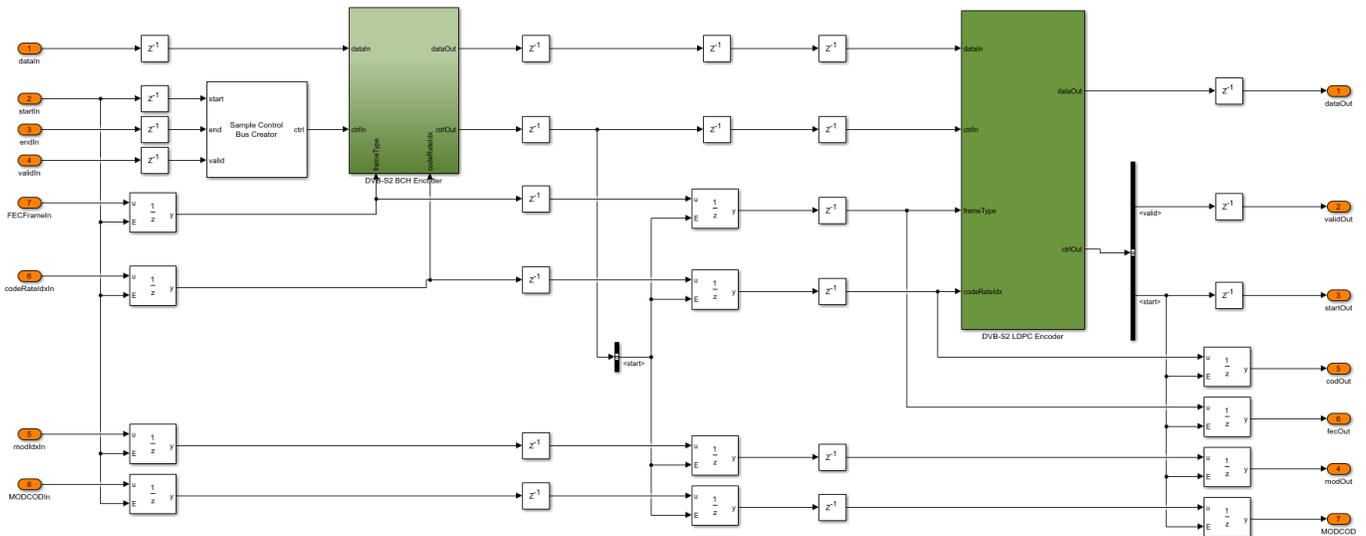
BB Frame Generator

The BB Frame Generator subsystem comprises BB Header and Data CRC Generator, Data Store FIFO, Multiplexer, and BB Scrambler subsystems. The BB Header and Data CRC Generator subsystem generates BB header, discards the SYNC bits, appends CRC bits to each packet using the General CRC Generator HDL Optimized block, and appends padding bits to the data field for each frame. The Data Store FIFO subsystem stores the data field of each frame in a RAM buffer and reads it out after the BB header. The Multiplexer subsystem multiplexes the BB header and data field. The BB Scrambler subsystem scrambles the BB header and data field to generate a BBFRAME.



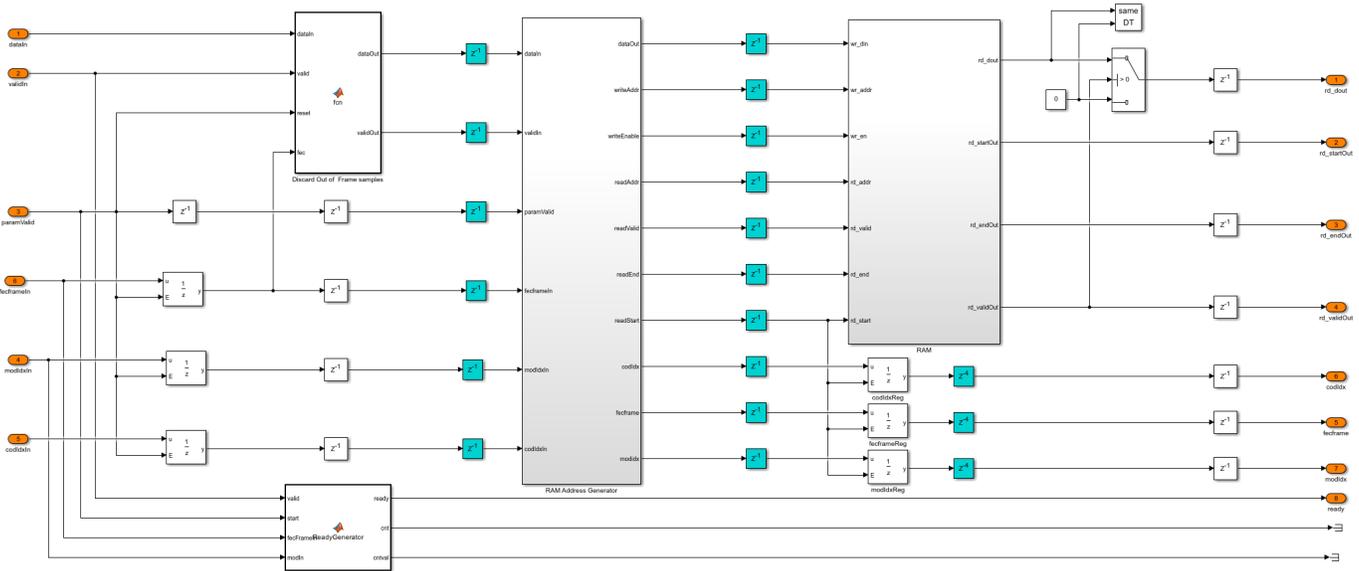
FEC Encoder

The FEC Encoder subsystem encodes the input bits with BCH Encoder subsystem followed by LDPC Encoder subsystem. For more information, see “DVB-S2 HDL BCH Encoder” (Wireless HDL Toolbox) and “DVB-S2 HDL LDPC Encoder” (Wireless HDL Toolbox) examples.

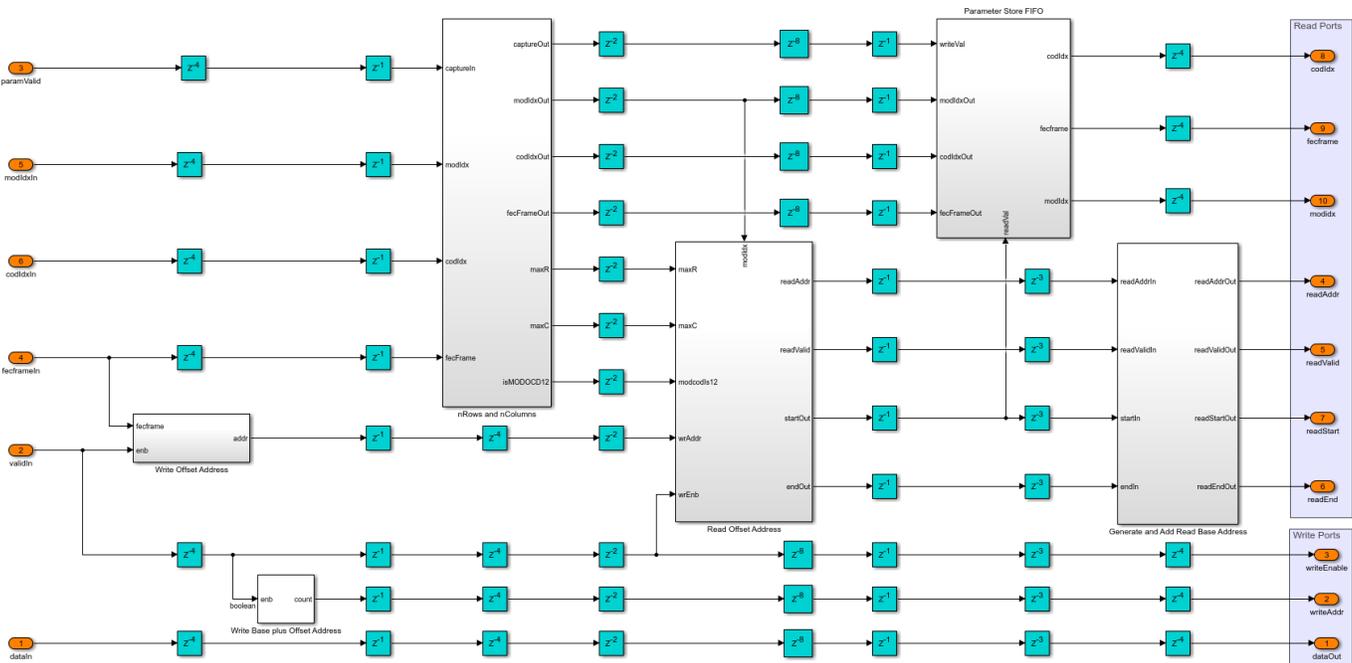


DVB-S2 HDL Interleaver

The DVB-S2 HDL Interleaver subsystem stores encoded bits from the LDPC Encoder subsystem inside the RAM subsystem. The RAM Address Generator subsystem generates read and write addresses for the RAM for interleaving.



In the RAM Address Generator subsystem, the nRows and nColumns subsystem stores the number of rows and columns of each possible configuration in lookup tables (LUT). Based on the **MODCOD** and **FECFrame** parameters, the subsystem determines the number of rows and columns for interleaving. The Read Offset Address subsystem generates the interleaver indices of each frame as an offset address. The subsystem reads only two bits are read for QPSK, three bits for 8-PSK, four bits for 16-APSK, and five bits for 32-APSK every eight time steps. This process ensures that each symbol covers eight time steps of the interleaver, which is equal to one symbol duration. The Generate and Add Read Base Address subsystem adds the offset address with a base read address to get the address of the bits that are stored in the RAM. The Parameter Store FIFO subsystem stores the parameters and reads them at the start of each frame.

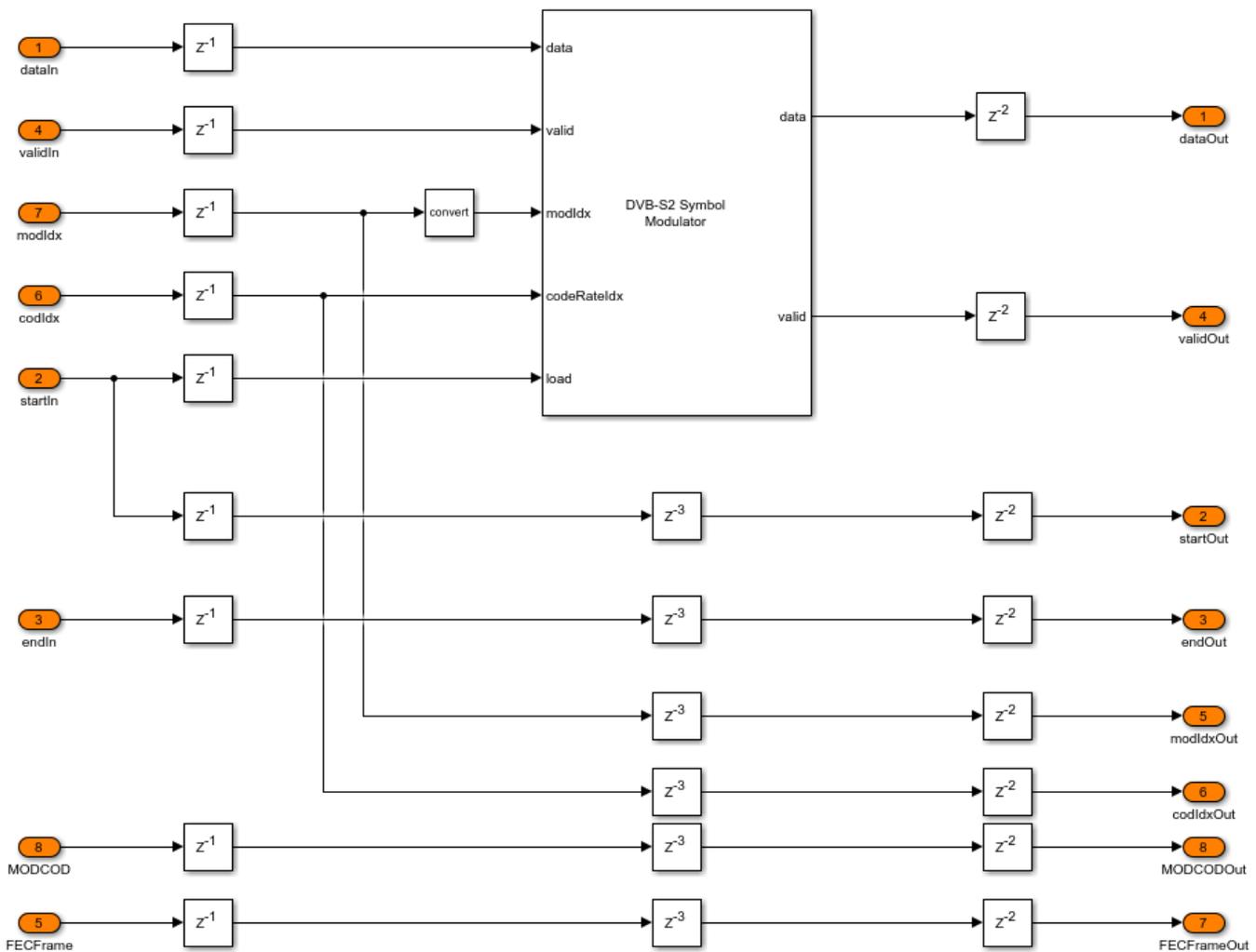


This table shows the rows and columns for deinterleaving in each of the configurations.

Modulation	Rows (Normal)	Rows (Short)	Columns
QPSK	64800	16200	1
8-PSK	21600	5400	3
16-APSK	16200	4050	4
32-APSK	12960	3240	5

Symbol Modulator

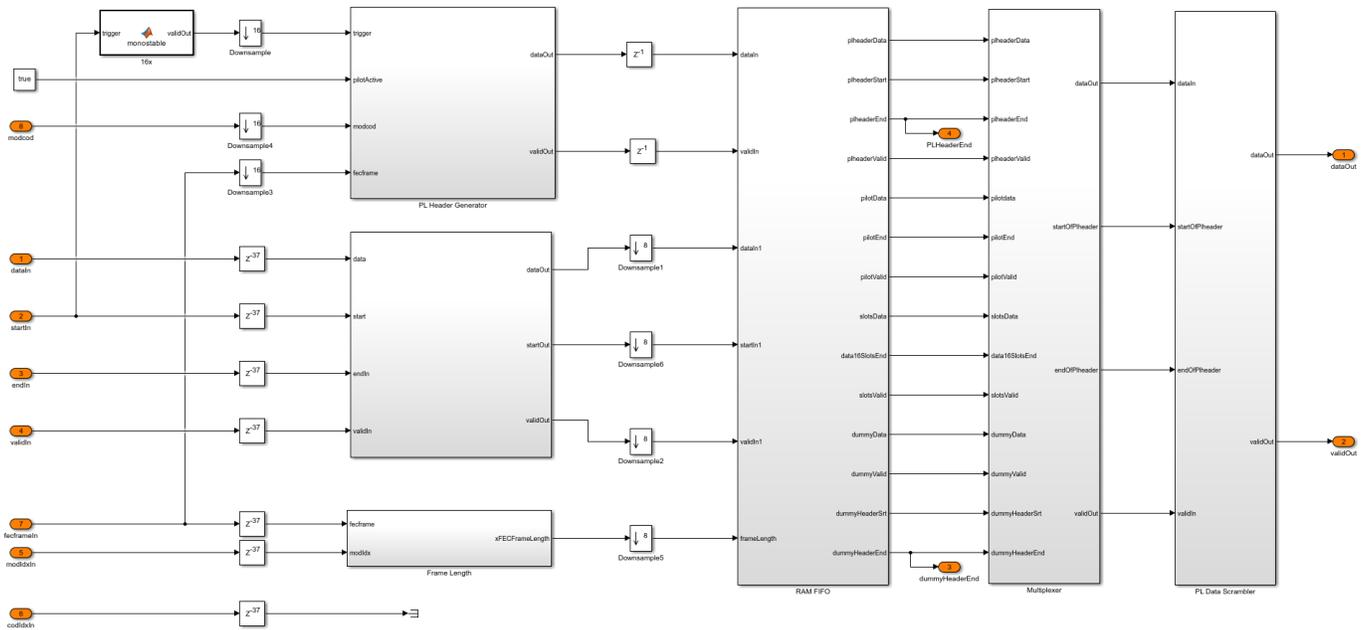
The Symbol Modulator subsystem comprises DVB-S2 Symbol Modulator (Wireless HDL Toolbox) block that maps the input bits to the corresponding modulation symbols.



PL Frame Generator

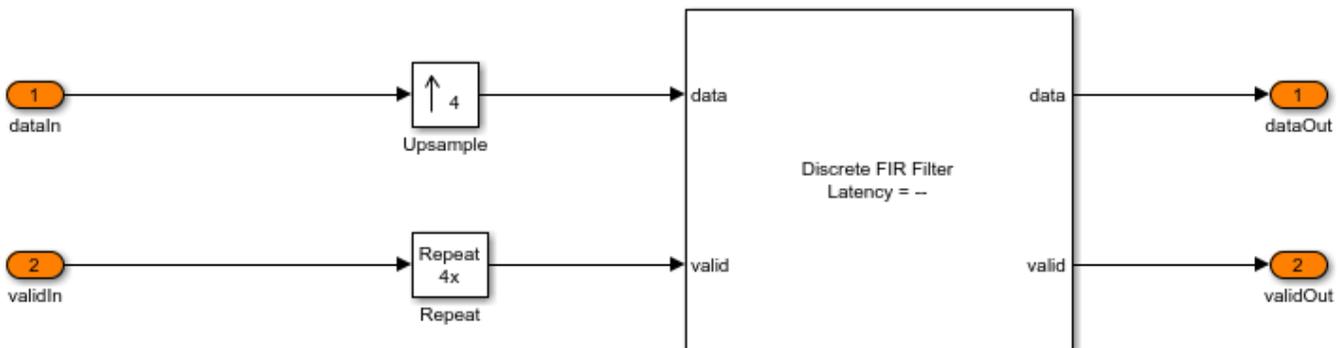
The PL Frame Generator subsystem stores the modulated symbols in the RAM FIFO subsystem. The PL Header Generator subsystem generates the PLHEADER for each frame and stores it in the

RAM FIFO subsystem. When symbols corresponding to a frame are ready in the RAM, the PL Frame Generator subsystem outputs the PLHEADER, pilot symbols, and data symbols according to the frame structure specified in [1]. When the symbols corresponding to a frame are not ready, the Dummy Frame Generator subsystem inside the RAM FIFO subsystem outputs dummy frames. The Multiplexer subsystem multiplexes the PLHEADER, pilot symbols, data symbols, and dummy frames to generate a unscrambled PLFRAME. The PL Data Scrambler subsystem scrambles the data field excluding the PLHEADER. The scrambling starts after the PLHEADER of each frame and continues until the end of the frame.



RRC Transmit Filter

The RRC Transmit Filter subsystem upsamples the input by a factor of four and uses the Discrete FIR Filter (DSP HDL Toolbox) block with an RRC impulse response to pulse shape the PLFRAME symbols.



Run Model

Set the symbol rate, **MODCOD**, **FECFrame** type, input stream format, and user packet length on the mask of the Input Configuration subsystem and run the dvbs2hdlTransmitter model. Set the

same symbol rate for the Input Configuration subsystem in the `dvbs2hdlTransmitter` model, the DVB-S2 Tx subsystem of the `dvbs2hdlTransmitterCore` model reference. Alternatively, execute this command at the MATLAB Command Window to run the model.

```
sim dvbs2hdlTransmitter
```

The **MODCOD** and **FECFrame** must be row vectors. Each element of the row vectors corresponds to the values of **MODCOD** and **FECFrame** in a frame.

Verification and Results

Run the model to display the transmitter spectrum, error plot between the reference waveform and simulation output, and relative mean squared error of the simulation output.

```
### Starting serial model reference simulation build.  
### Successfully updated the model reference simulation target for: dvbs2hdlTransmitterCore
```

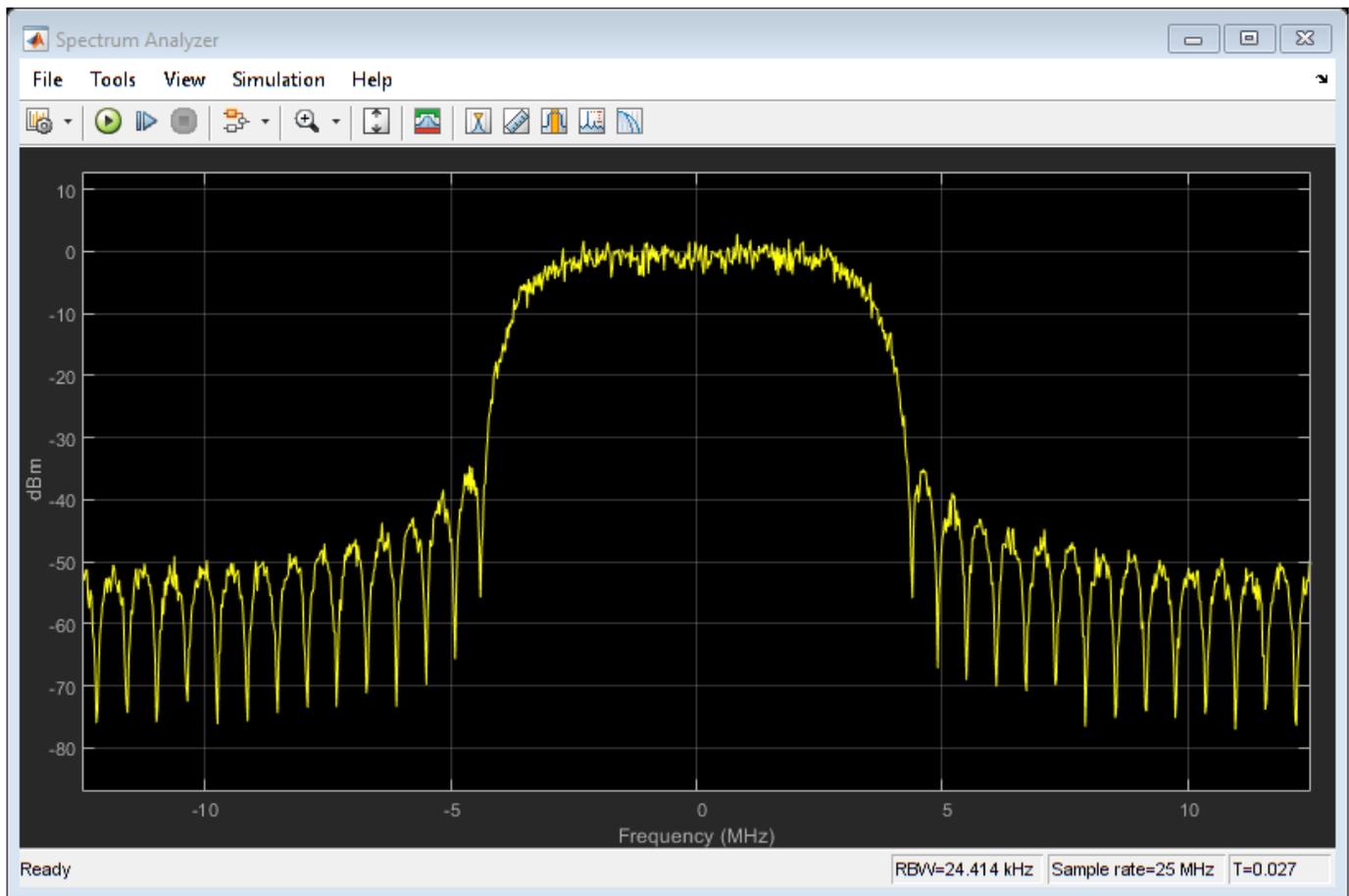
```
Build Summary
```

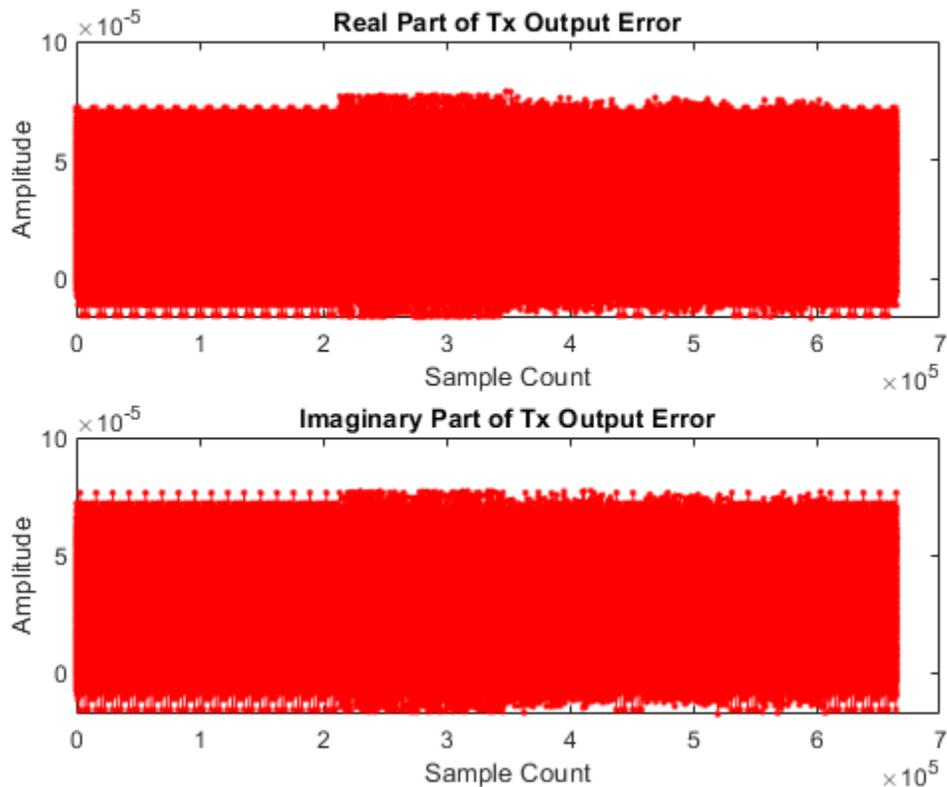
```
Simulation targets built:
```

Model	Action	Rebuild Reason
dvbs2hdlTransmitterCore	Code generated and compiled.	dvbs2hdlTransmitterCore_msf.mexw64 does not exist

```
1 of 1 models built (0 models already up to date)  
Build duration: 0h 2m 38.59s
```

```
Simulation Completed. Running verification script...  
Relative mean squared error (dB) between the simulink output and reference = -79.6677
```





HDL Code Generation

To generate the HDL code for this example, you must have HDL Coder™. Use `makehdl` and `makehdltb` functions to generate HDL code and HDL test bench for the DVB-S2 HDL transmitter subsystem. The test bench generation time depends on the simulation time.

The resulting HDL code is synthesized for a Xilinx® Zynq® UltraScale+ RFSoc ZCU111 board. This table shows the post place and route resource utilization. The maximum frequency of operation is 202 MHz.

Resources	Usage
CLB LUT	14717
CLB Registers	8348
RAMB36	151
RAMB18	1
DSP48	42

References

- 1 ETSI EN 302 307-1. *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

- 2** ETSI TR 102 376-1. *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2).*

DVB-S2 HDL PL Header Recovery

This example shows how to implement DVB-S2 time, frequency, and phase synchronization and PL header recovery using Simulink® blocks that are optimized for HDL code generation and hardware implementation.

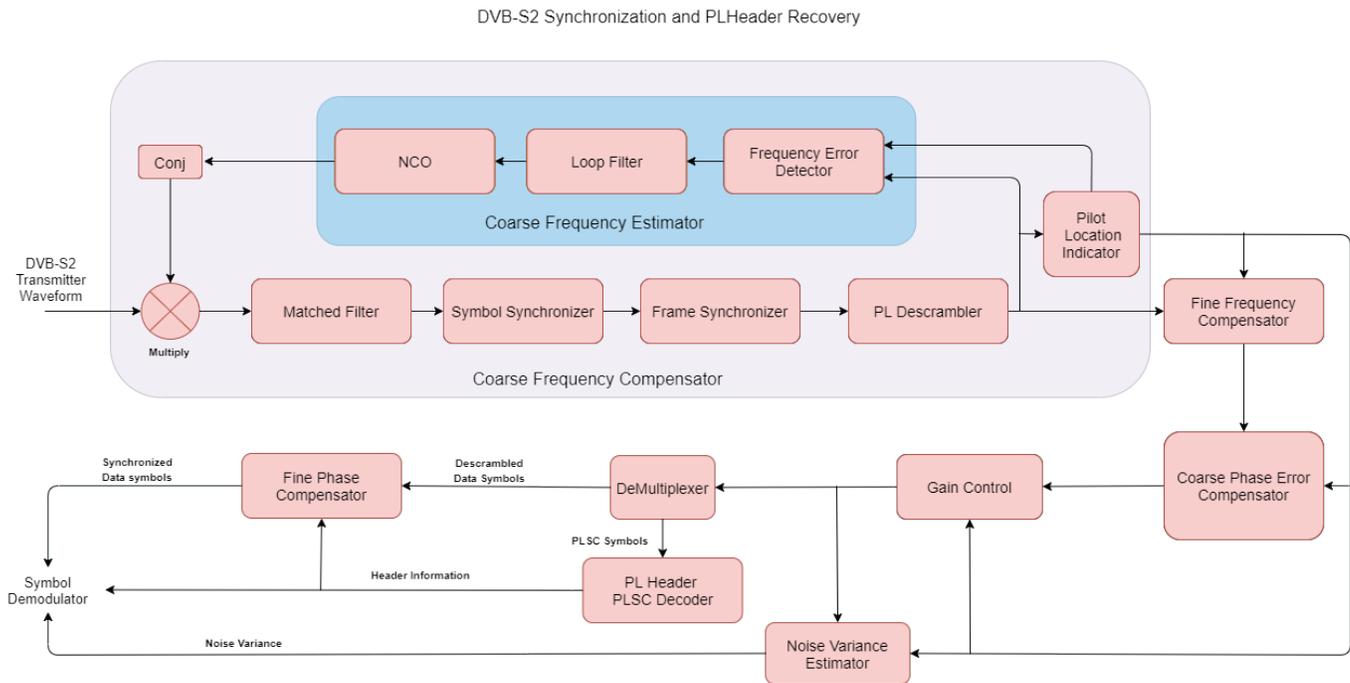
Digital Video Broadcasting Satellite Second Generation (DVB-S2) modems operate in C (4-8 GHz), Ku (12-18 GHz) and Ka (26-40 GHz) frequency bands. According to the DVB-S2 standard, the satellite transponder bandwidth ranges from 1 MHz to 72 MHz. The model in this example operates at a symbol rate of 25 Mbaud with a root raised cosine (RRC) filter roll-off factor of 0.35. For a MATLAB® implementation of end-to-end DVB-S2 receiver, see the “End-to-End DVB-S2 Simulation with RF Impairments and Corrections” on page 4-37 example.

This example shows how to design a DVB-S2 HDL receiver synchronization and physical layer (PL) header recovery system that can handle radio frequency (RF) impairments. The model in this example performs symbol timing synchronization, frame synchronization, coarse and fine frequency synchronization, phase offset estimation and correction, gain correction, and noise variance estimation. Then the model decodes the PL header information followed by fine phase synchronization.

Model Architecture

This section explains the high-level architecture of the model. The model receives the DVB-S2 transmitter waveform sequence that streams into the Coarse Frequency Compensator block. The Symbol Synchronizer block extracts the modulated symbol sequence from the Matched Filter block output and the Frame Synchronizer block locates the start of each frame in the modulated symbol sequence. The PL Descrambler block descrambles the scrambled data symbols and the Pilot Generator block indicates the pilot locations in the frame synchronized sequence. The Coarse Frequency Estimator block estimates the frequency offset, which is used to correct the frequency offset in the transmitter waveform sequence at the model input by conjugate multiplication of the estimate. The Fine Frequency Compensator block corrects the residual frequency left in the PL descrambled sequence. The Coarse Phase Error Compensator block corrects the coarse phase deviation in the Fine Frequency Compensator block output sequence. The phase error compensated sequence is magnitude corrected in the Gain Control block and the gain-corrected sequence is used to estimate noise variance. The Demultiplexer divides the gain-corrected sequence into physical layer signaling code (PLSC) symbols and descrambled data symbols in each frame. The PL Header PLSC Decoder block decodes header parameters **MODCOD** and **FECFrame**. The Fine Phase Compensator block uses the **MODCOD** parameter and corrects the residual phase in the descrambled data symbols that stream into the symbol demodulator.

This block diagram shows the high-level architecture of the model.



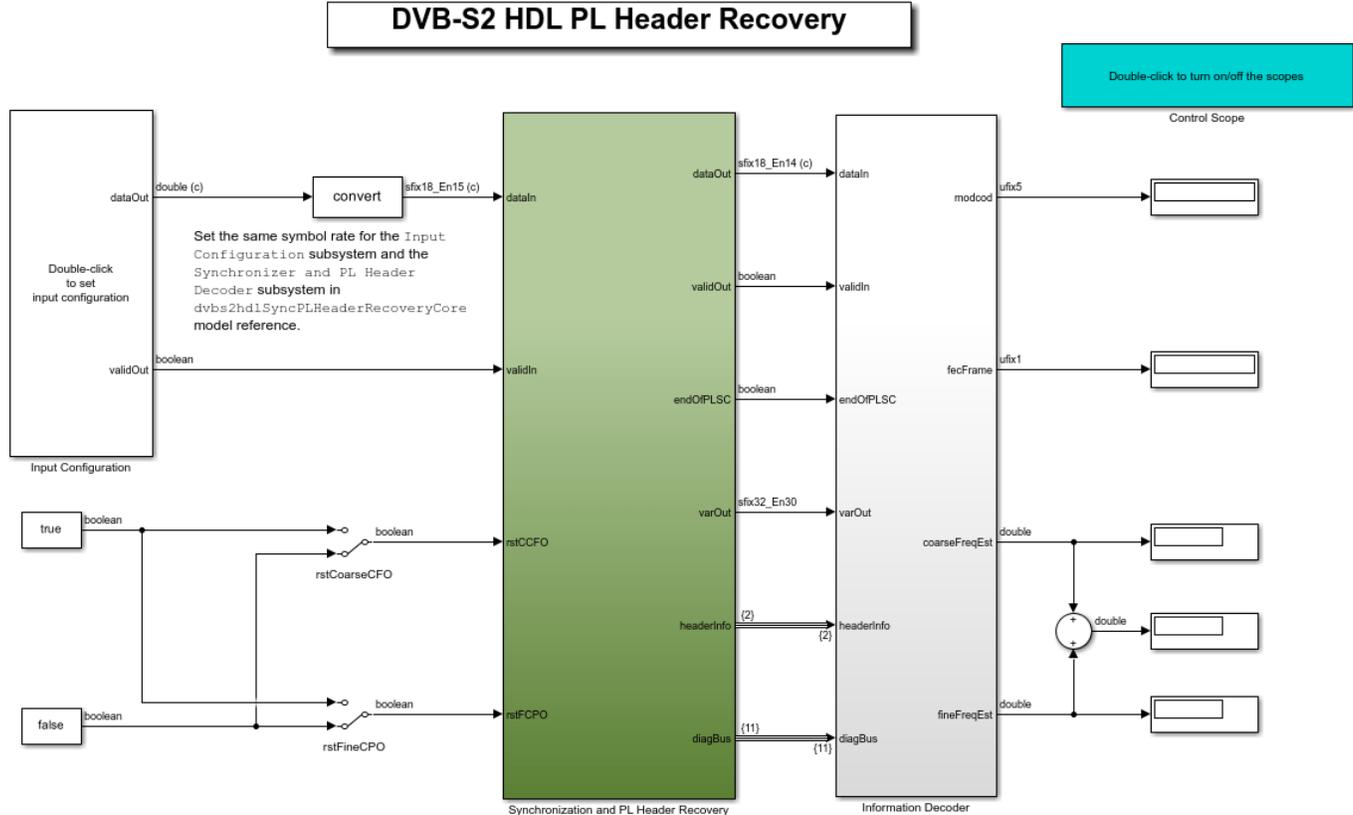
File Structure

This example uses two Simulink models, six MATLAB files, and one Simulink data dictionary.

- `dvbs2hdlPLHeaderRecovery.slx` — Top-level Simulink model.
- `dvbs2hdlSyncPLHeaderRecoveryCore.slx` — Model reference that synchronizes time, frequency, and phase, and decode PL Header.
- `getdvbs2LDPCParityMatrices.m` — Download the LDPC matrices .mat file.
- `dvbs2hdlRxParameters.m` — Generate parameters for the `dvbs2hdlSyncPLHeaderRecoveryCore.slx` model reference.
- `dvbs2hdlPhaseNoise.m` — Introduce phase noise to the input sequence.
- `dvbs2hdlRxInit.m` — Generate the transmitter waveform and initialize the `dvbs2hdlSyncPLHeaderRecoveryCore.slx` model reference.
- `dvbs2hdlStreamRecovery.m` — Decode baseband frame (BBFRAME).
- `dvbs2hdlPLHeaderRecoveryVerify.m` — Gather PL header parameters and XFECFRAME symbols, demodulate symbols, decode FEC (LDPC and BCH), recover baseband frame, and compute bit errors using Satellite Communications Toolbox functions.
- `dvbs2hdlReceiverData.sldd` — Store bus signal configurations that come out of the model reference.

System Interface

This figure shows the top-level overview of the `dvbs2hdlPLHeaderRecovery.slx` model.



Model Inputs

- **dataIn** — Input data, specified as an 18 bit complex data with a sample rate that is four times the symbol rate.
- **validIn** — Control signal to validate the **dataIn** input port, specified as a Boolean scalar.
- **rstCCFO** — Control signal to reset the coarse frequency compensation loops, specified as a Boolean scalar.
- **rstFCPO** — Control signal to reset the fine phase compensation loops, specified as a Boolean scalar.

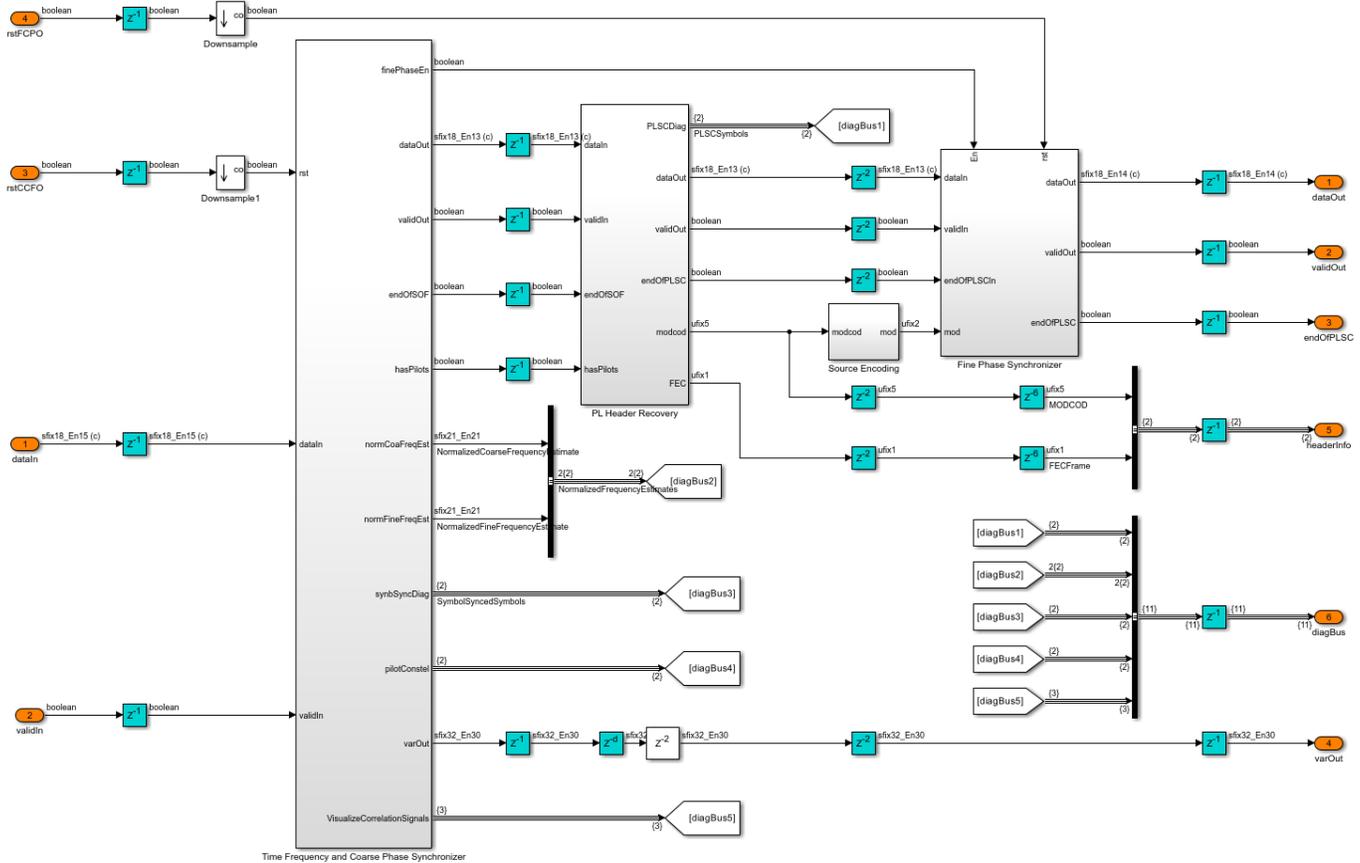
Model Outputs:

- **dataOut** — Decoded output symbols, returned as an 18 bit complex scalar.
- **validOut** — Control signal to validate the **dataOut** output port, specified as a Boolean scalar.
- **endOfPLSC** — Control signal to indicate the end of PLSC symbols in each synchronized frame, specified as a Boolean scalar.
- **nVar** — Estimated noise variance, returned as a 32 bit complex scalar.
- **headerInfo** — Bus signal to provide the parameters **MODCOD** and **FECFrame** of the PL header in each synchronized frame.

- **diagBus** — Bus signal to provide the coarse frequency normalized with the sample rate, the fine frequency normalized with the symbol rate, the symbol synchronized output, the PLSC symbols, and the pilot symbols.

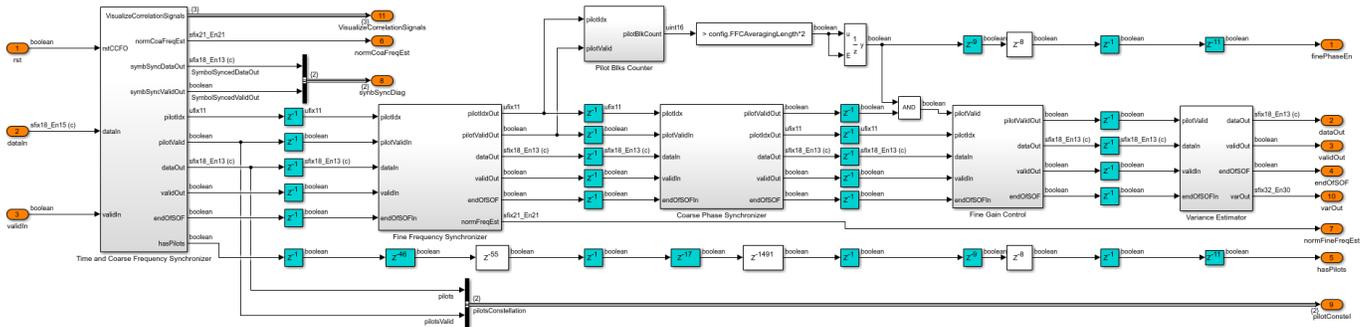
Model Structure

This figure shows the top-level model of the Synchronization and PL Header Recovery subsystem. It comprises Time Frequency and Coarse Phase Synchronizer, PL Header Recovery, and Fine Phase Synchronizer subsystems.



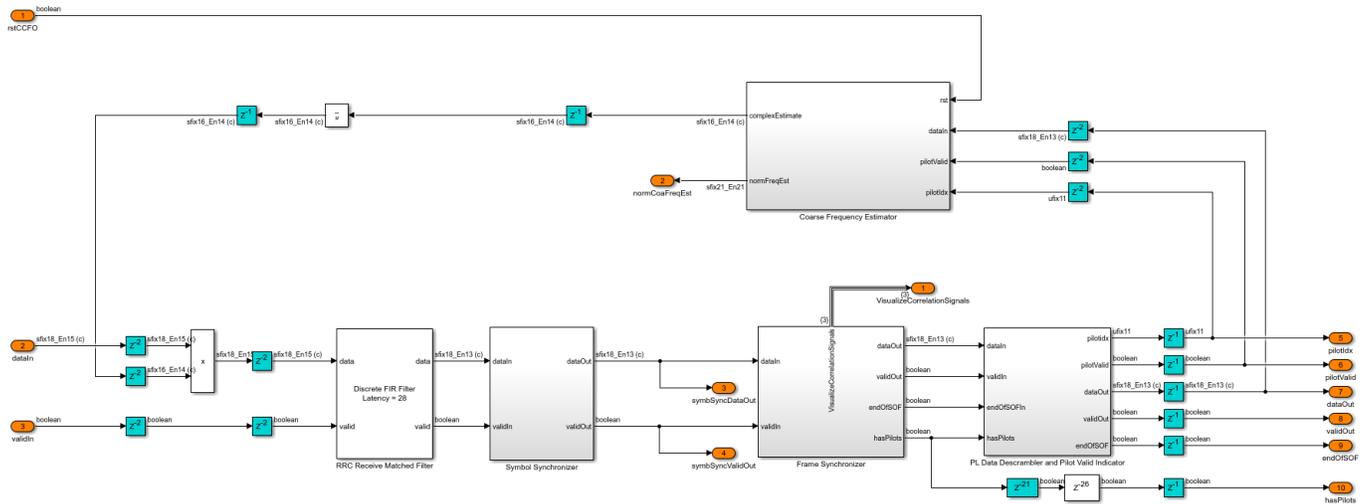
Time Frequency and Coarse Phase Synchronizer

The Time Frequency and Coarse Phase Synchronizer subsystem comprises Time and Coarse Frequency Synchronizer, Fine Frequency Synchronizer, and Coarse Phase Synchronizer subsystems.



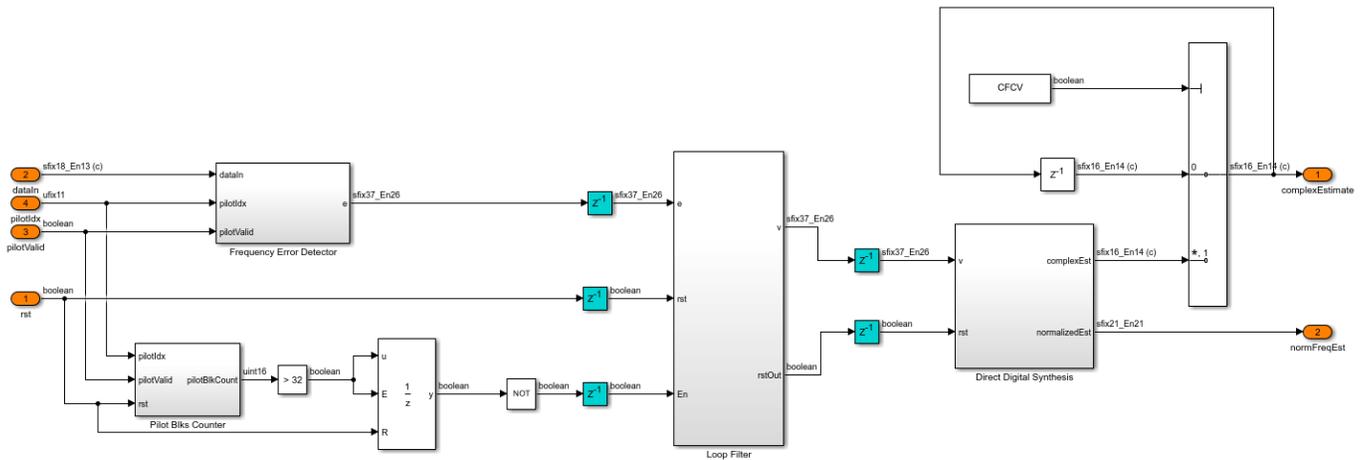
Time and Coarse Frequency Synchronizer

The Time and Coarse Frequency Synchronizer subsystem compensates coarse frequency in a frequency-locked loop (FLL) system. The normalized loop bandwidth of the FLL system is set to $1e-4$. The loop involves RRC matched filtering, symbol synchronization, frame synchronization, PL descrambling, pilot extraction, and coarse frequency estimation.



Coarse Frequency Estimator

The Coarse Frequency Estimator subsystem performs frequency error detection, loop filtering, and direct digital synthesis. The frequency error detection is described with equation C.2 in the Annex C.4 of [2]. The coarse frequency estimator is a pilot-aided frequency estimator. The Frequency Error Detector subsystem outputs frequency error at the pilot locations. The frequency error is passed through the loop filter and the output of the loop filter drives the NCO (DSP HDL Toolbox) block to generate the complex exponential sinusoidal samples. These samples are conjugated and multiplied by the input sequence to correct the frequency offset. The loop filter is disabled for frequency error filtering after 32 pilot blocks so that the estimated frequency remains stable. A reset signal **rstCCFO** resets the loop filter and restarts the estimation process.

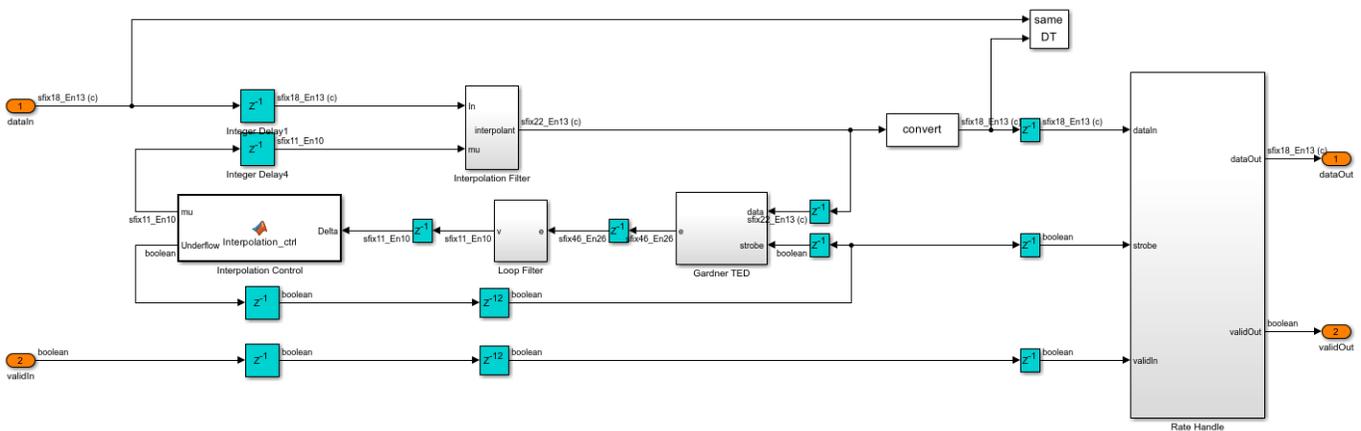


RRC Receive Matched Filter

The RRC Receive Matched Filter is a Discrete FIR Filter (DSP HDL Toolbox) block with matched filter coefficients with four samples per symbol, and a roll-off factor of 0.35. The RRC matched filtered output is an RC pulse-shaped waveform that has zero inter symbol interference (ISI) characteristics at the maximum eye opening in the eye diagram of the waveform. Also, the matched filtering process maximizes the signal-to-noise power ratio (SNR) of the filter output.

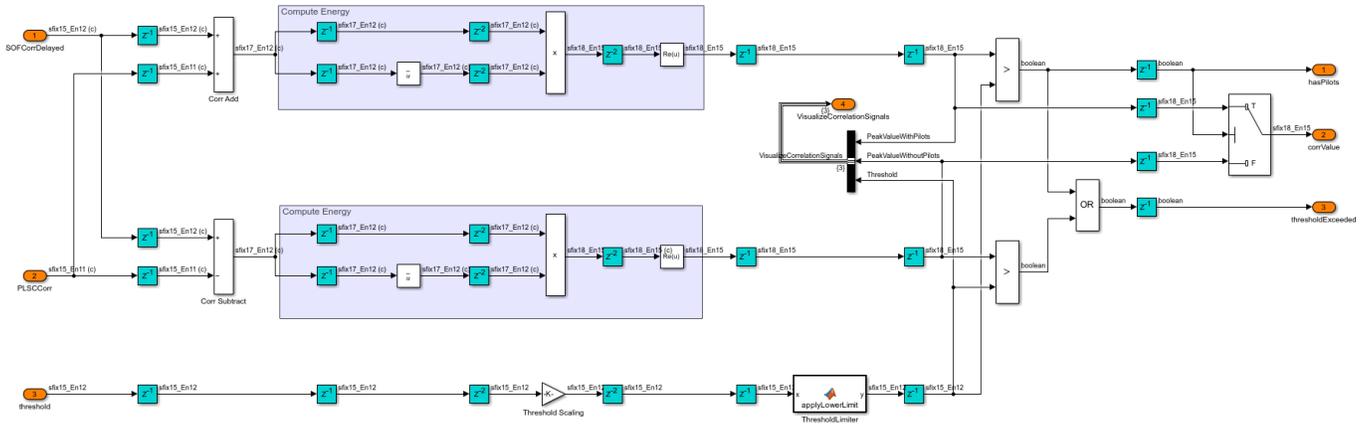
Symbol Synchronizer

The Symbol Synchronizer subsystem is a phase locked loop (PLL) based implementation as described in the chapter 8.4 of [4]. The subsystem generates one output sample for every four input samples. The PLL loop is set with a normalized loop bandwidth of 8e-3. The Interpolation Filter subsystem implements a piecewise parabolic interpolator with a hardware resource efficient farrow structure. This filter introduces fractional delays in the input waveform. As specified in Annex C.2 of [2], the Gardner TED subsystem implements a Gardner timing error detector. The loop filter filters the timing error and pass it on to the Interpolation Control MATLAB function block. This block implements a mod-1 decrementing counter to calculate fractional delays based on the loop-filtered timing error to generate interpolants at optimum sampling instants. The Rate Handle subsystem selects the required interpolant indicated by the strobe.



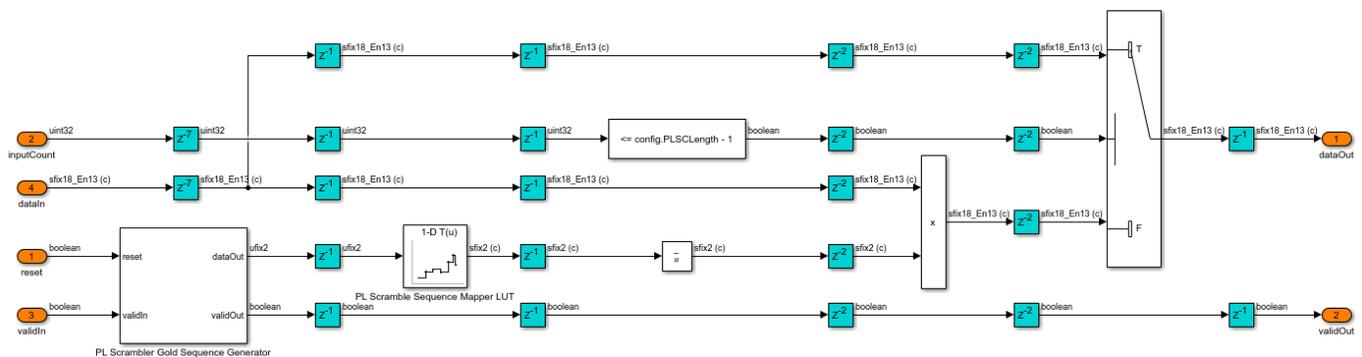
Frame Synchronizer

The Pilot and Sync Pulse Indicator subsystem adds and subtracts the SOF and PLSC correlation values and computes energy at each time step to generate two correlation metrics. The threshold is downscaled with a precomputed value in the `dvbs2hdlRxParameters.m` file, and a lower limit is applied to saturate the threshold with a lower bound. Both of the correlation metrics are compared with the downscaled threshold value. The existence of pilots in the current frame is confirmed if the correlation metric obtained by adding SOF and PLSC correlation values exceeds the downscaled threshold. For a given frame, only one of the two correlation metrics exceeds the threshold based on the existence of pilots.



PL Data Descrambler

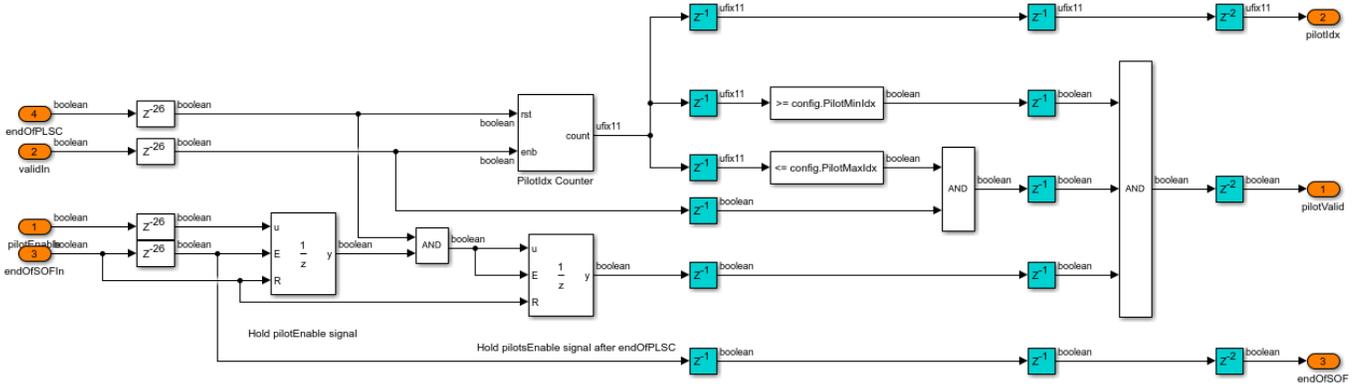
The PL Data Descrambler subsystem uses PL Scrambler Gold sequence Generator subsystem, which is described in section 5.5.4 of [1]. The PL Scrambler Gold Sequence Generator subsystem resets for every frame. The gold sequence is used as an address to the PL Scramble Sequence Mapper LUT block to generate the PL scrambling sequence. The scrambling sequence is conjugated to generate the PL descrambling sequence, and the descrambling is performed by multiplying PL descrambling sequence with the input sequence. A switch is used to multiplex the PLSC symbols and the descrambled data symbols.



Pilot Valid Indicator

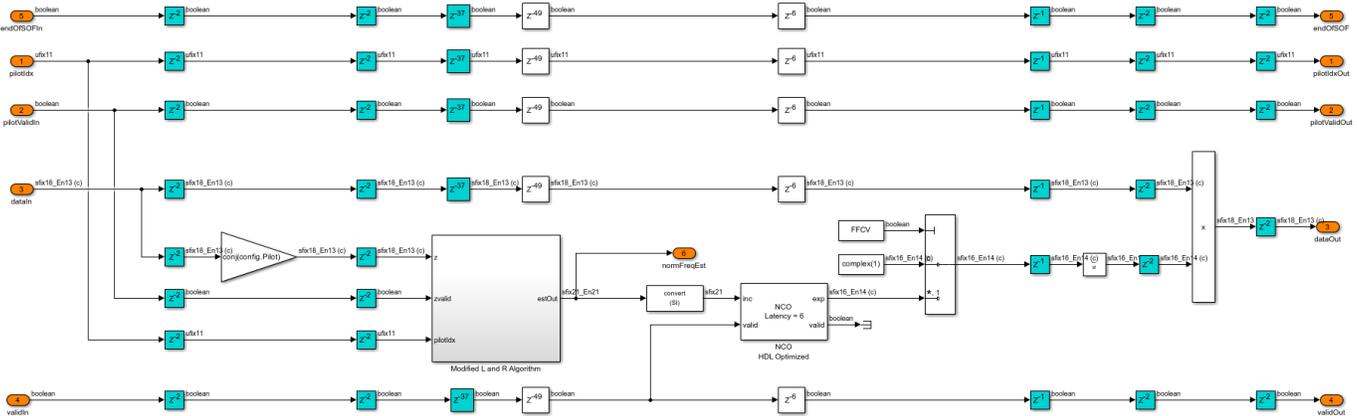
The Pilot Valid Indicator subsystem counts the input sequence and assigns a pilot index for each symbol. As specified in section 5.5.3 of [1], the pilots of length 36 symbols exist after 16 slots (1440 symbols) in a pilot-active XFECFRAME (pilot activeness is confirmed in the frame synchronizer). The subsystem generates a pilot valid signal for 36 symbols to indicate the location of

the pilot block. The counter resets after the pilot block. This process continues for the rest of the XFECFRAME. The signal, which indicates the end of the PLSC symbols of the next XFECFRAME, determines the end of the current XFECFRAME. The pilot valid signal is not generated after 16 slots if the next XFECFRAME is detected, as the PLSC symbols take the place of pilots.



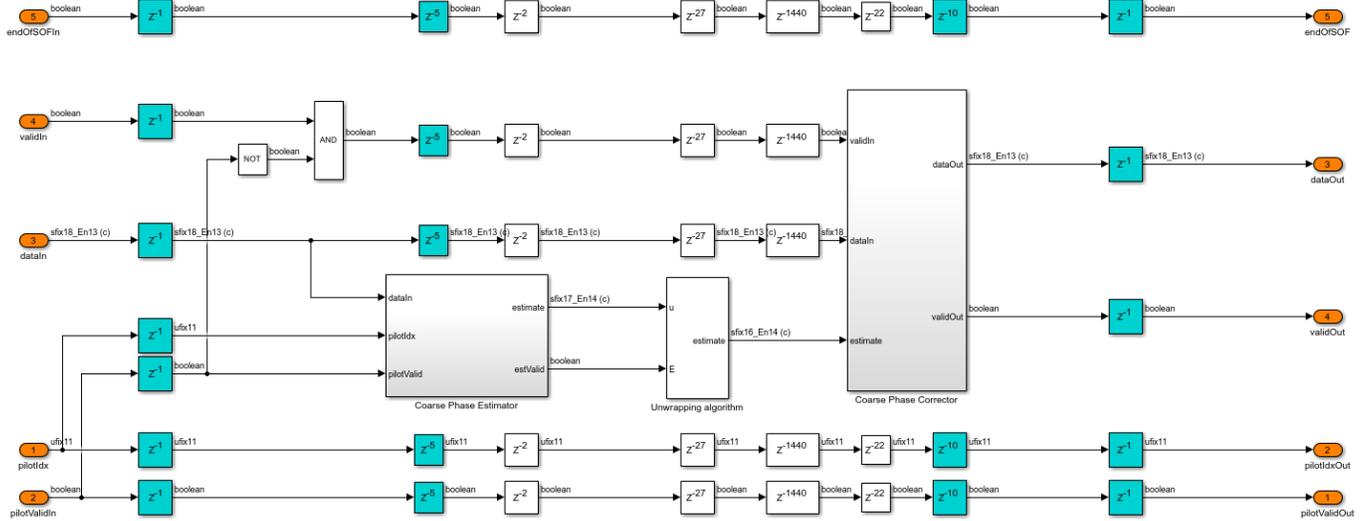
Fine Frequency Synchronizer

The Fine Frequency Synchronizer subsystem uses the Modified L and R algorithm, as described by the equation C.3 in Annex C.4 of [2]. The L and R algorithm is described in [3]. The subsystem implements an 18 point autocorrelation function of the L and R algorithm followed by a 512 length moving average filter. The frequency estimate from the Modified L and R Algorithm subsystem drives the NCO (DSP HDL Toolbox) block to generate the complex exponential sinusoidal samples, which are conjugated and used to correct the frequency offset in the input.



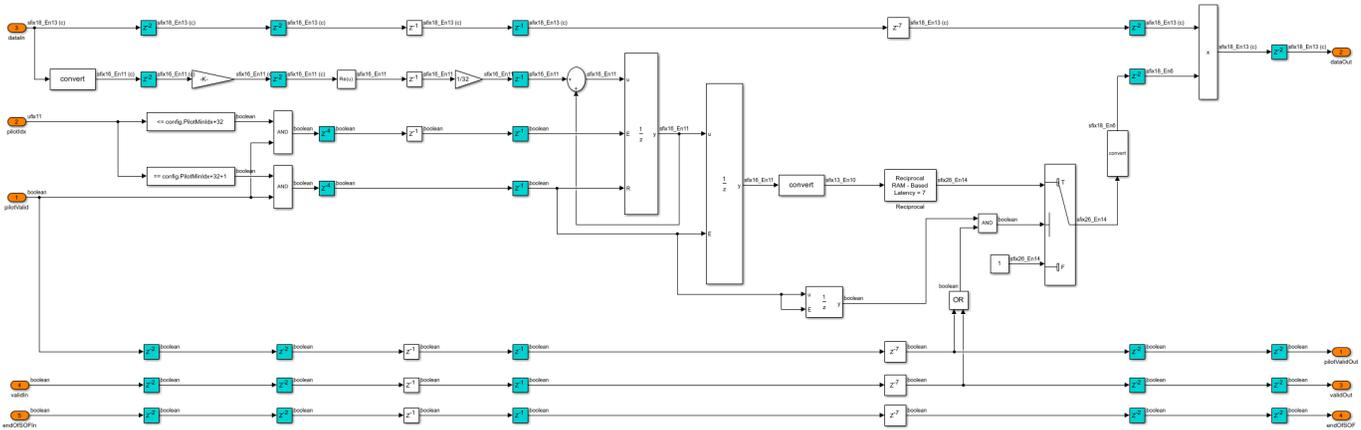
Coarse Phase Synchronizer

The Coarse Phase Synchronizer subsystem uses the pilot aided linear interpolation technique. The Coarse Phase Estimator subsystem estimates the complex phase from each of the 36 pilot symbols and performs averaging, which results in one estimate from each pilot block. The Unwrapping Algorithm subsystem implements the equation C.7 in section C.6.1 of [2] and interpolates the complex phase estimate from two consecutive pilot blocks. This interpolated estimate is used to compensate the phase of the symbols in between these two consecutive pilot blocks.



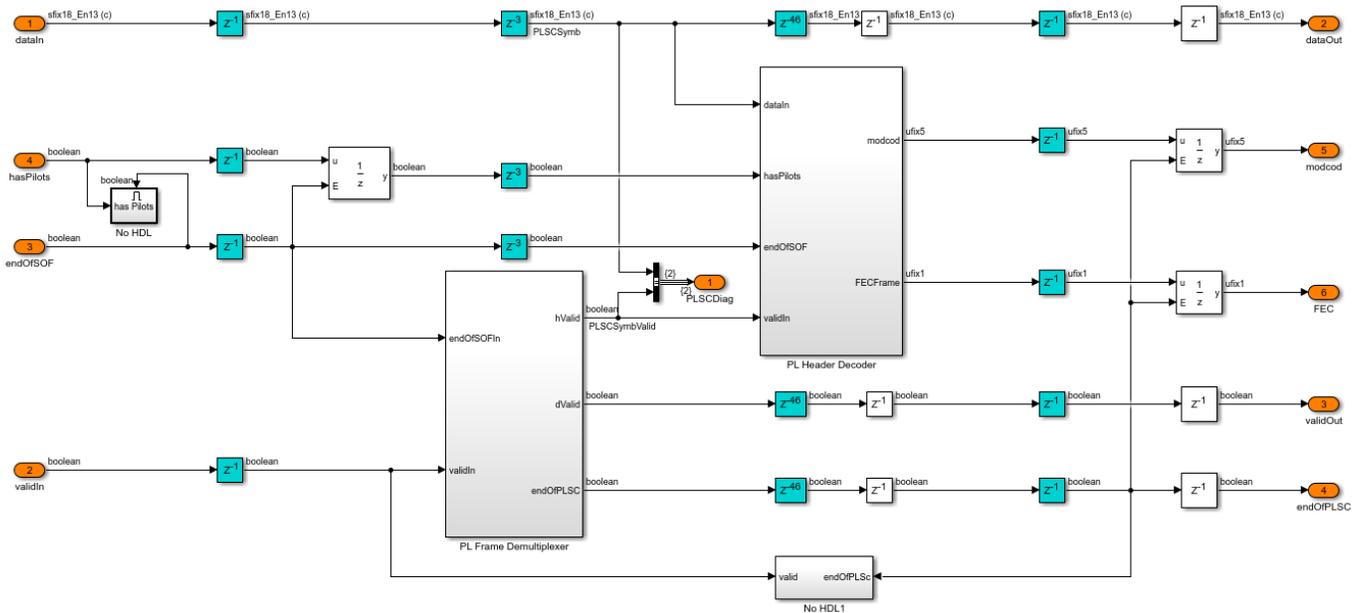
Fine Gain Control

The Fine Gain Control subsystem performs magnitude correction using the estimates derived from the pilot symbols. The input sequence is time and frequency synchronized before gain control. Each estimate is derived by multiplying the pilot symbol in the input sequence with reference pilot. The divide (1/32 gain block) and the integrator averages 32 estimates and stores in a register. The input is divided with the averaged estimate to correct the input magnitude.



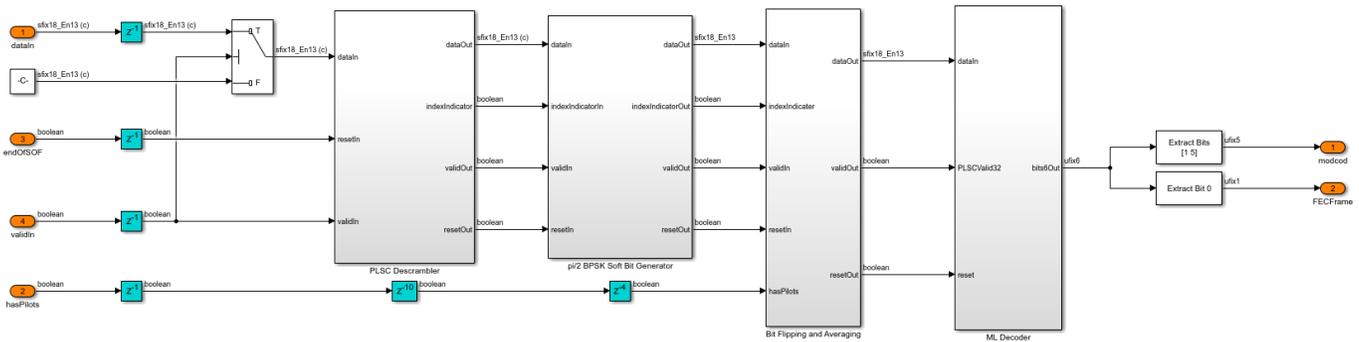
Variance Estimator

The Variance Estimator subsystem computes the noise variance of the input signal. The input sequence is time and frequency synchronized and gain-corrected before estimating the variance. The reference pilot symbols are subtracted from the noise affected pilot symbols in the input sequence to generate zero-mean noisy symbols. The variance is computed by absolute squaring these symbols and averaging previous 2048 symbols using a moving average filter of length 2048.



PL Header Decoder

The PLSC Descrambler subsystem in the PL Header Decoder subsystem descrambles the PLSC symbols. The signal from the **indexIndicator** port of the PLSC Descrambler subsystem distinguishes the even and odd locations of the PLSC symbols. The **pi/2 BSPK Soft Bit Demodulator** subsystem demodulates the PLSC symbols. If the pilots exist in the current PLFRAME (which is decided in the frame synchronization), the **Bit Flipping and Averaging** subsystem multiplies the odd soft bits by -1 in the PLSC symbols. A bit flip for a hard bit is same as multiplying by -1 for a soft bit. The subsystem averages the soft bits in even and odd locations to get one estimate. Likewise, 32 soft bits are generated from 64 soft bits. A maximum likelihood (ML) decoder is used to decode the (32,6) bi-orthogonal encoded bits. The 6 decoded bits are used to construct the **MODCOD** and **FECFrame** type.

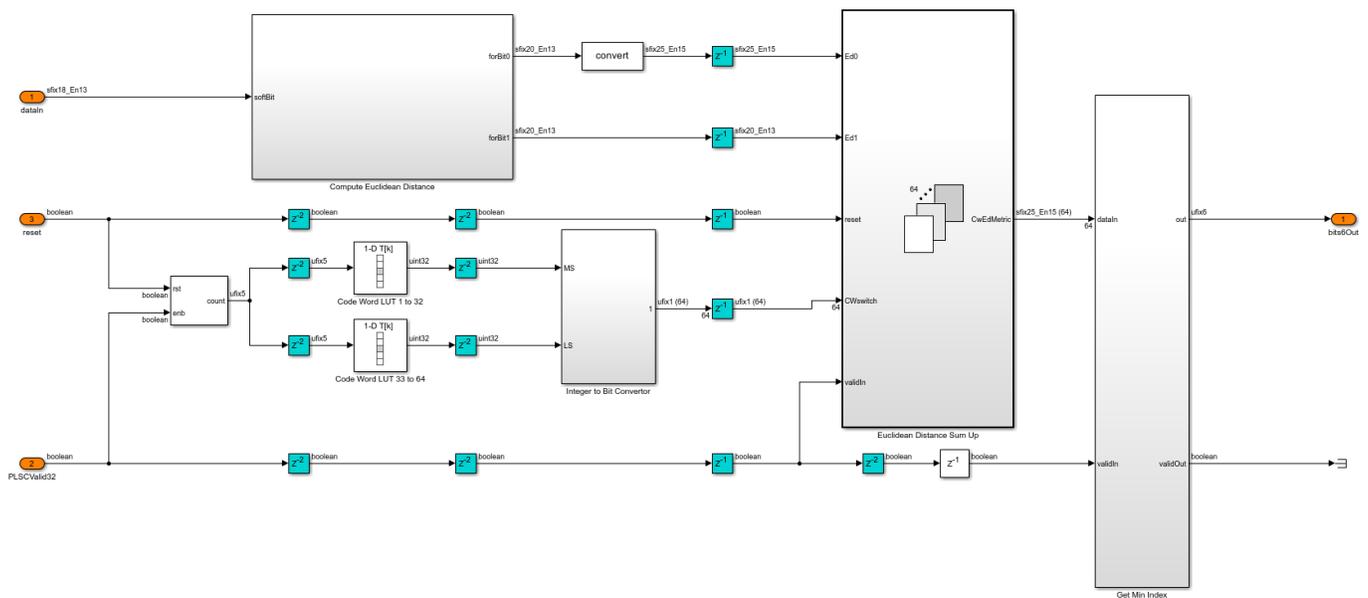


ML Decoder

The ML Decoder subsystem decodes the (32,6) bi-orthogonal code by choosing the maximum likelihood codeword. A total of $2^6 = 64$ codeword combinations, each 32 bits wide, are precomputed in the `dvbs2hdlRxParameters.m` file. The codewords are stored as integers in the `uint32` format, with the first 32 codewords in one look-up table (LUT) and the next 32 codewords in another LUT. The

LUT storing is such that the most significant bit of all of the codewords is called first, followed by the next significant bits, and so on. A bit level Euclidean distance is computed in the **Compute Euclidean Distance** subsystem with -1 and 1 as reference values for bit 0 and bit 1, respectively. The **Euclidean Distance Sum Up** subsystem adds all of the 32 bit level Euclidean metrics over time and generates a codeword Euclidean metric for each codeword. This subsystem uses a *for each* iterator to repeat the execution for all of the codewords and generates 64 codeword Euclidean metrics. The minimum Euclidean metric of 64 combinations maps to the maximum likelihood codeword. The maximum likelihood code word is used to construct the 6 bit input, and the **MODCOD** and **FECFrame** type values.

For a hardware-friendly implementation, the Euclidean metric is computed (computing involves multipliers) outside the **ML Decoder** subsystem as it uses a *for each* iterator.

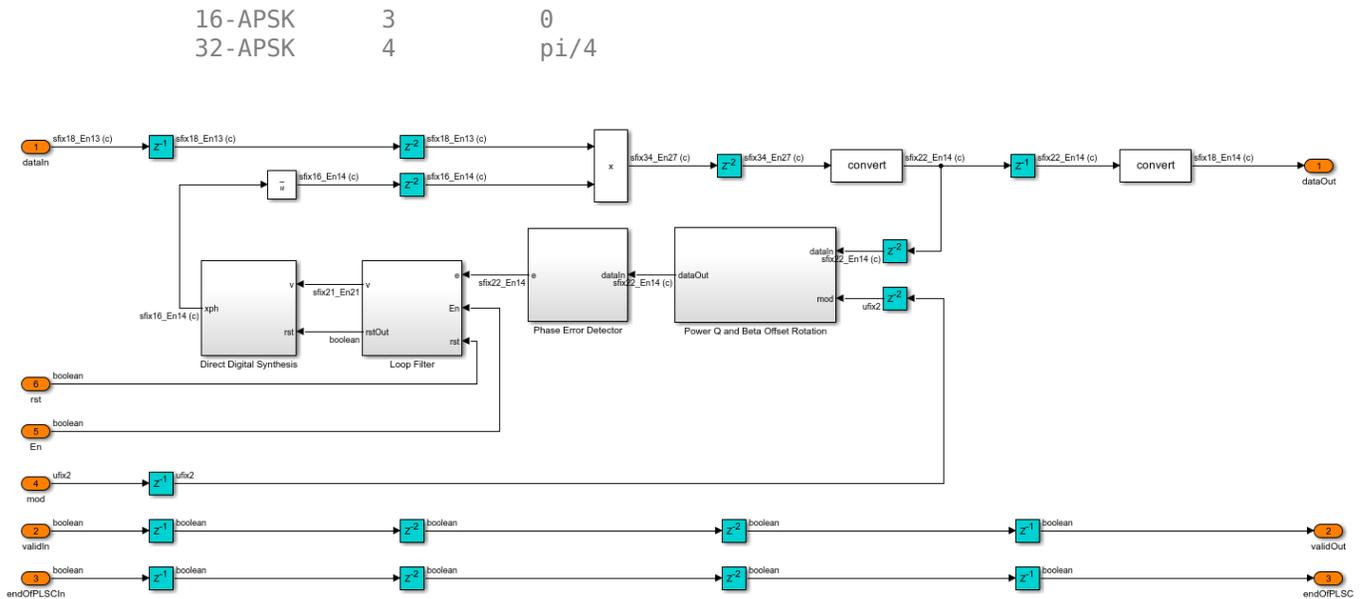


Fine Phase Synchronizer

The **Fine Phase Synchronizer** subsystem is a PLL implementation. Its normalized loop bandwidth is set to $20e-6$. The **MODCOD** value decoded from the PL header specifies the modulation type of the symbols in the frame. The **Power Q** and **Beta Offset Rotation** subsystem raises the QPSK, 8-PSK, 16-APSK, and 32-APSK symbols to a power of Q and rotates the constellation by an angle of β . The **Phase Error Detector** subsystem computes the phase error from the output of the **Power Q** and **Beta Offset Rotation** subsystem, as described by equation C.10 in annex C.6.2 of [2]. The phase error is filtered by the loop filter. The filtered output drives the **NCO (DSP HDL Toolbox)** block in the **Direct Digital Synthesis** subsystem to generate the complex exponential sinusoidal samples, which are conjugated and used to correct the phase of the input samples. A reset signal **rstCCFO** resets the loop filter and restarts the estimation process.

This table shows the Q and β values for the modulated symbols

Modulation	Q	Beta (in radians)
QPSK	1	0
8-PSK	2	$\pi/4$



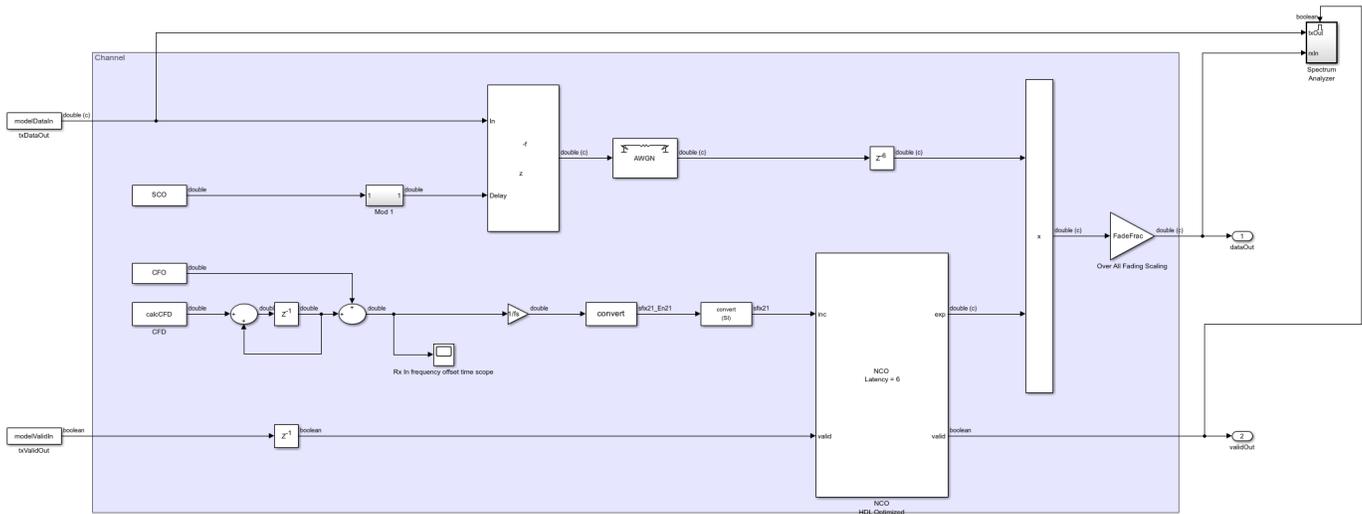
Channel

The Channel subsystem introduces the impairments in this table.

Impairment	Description
Fading Factor	Specified in the interval [0.9,1.1]
Additive white Gaussian noise (AWGN)	Specified in Es/N0 in dB
Carrier frequency offset (CFO)	Specified in Hz
Carrier frequency drift (CFD)	Specified in Hz/second
Carrier phase offset (CPO)	Specified in degrees
Sampling clock offset (SCO)	Specified in the interval [0,1)
Phase noise	Specified as Low, Medium, High

This table defines the phase noise mask level in dBc/Hz that the phase noise generator in the `dvbs2hdlPhaseNoise.m` file uses to generate the phase noise and introduce in the transmitter output signal.

Frequency	Low	Medium	High
100 Hz	-73	-59	-25
1 KHz	-83	-77	-50
10 KHz	-93	-88	-73
100 KHz	-112	-94	-85
1 MHz	-128	-104	-103



Run the Model

Set the symbol rate, **MODCOD**, **FECFrame** type values, input stream format, user packet length and channel impairments on the Input Configuration subsystem mask and run the `dvbs2hdlPLHeaderRecovery.slx` model. Alternatively, to run the model, execute this command at the MATLAB command prompt.

```
sim dvbs2hdlPLHeaderRecovery
```

The **MODCOD** and **FECFrame** type values must be row vectors. Each element of the row vector corresponds to a frame.

Verification and Results

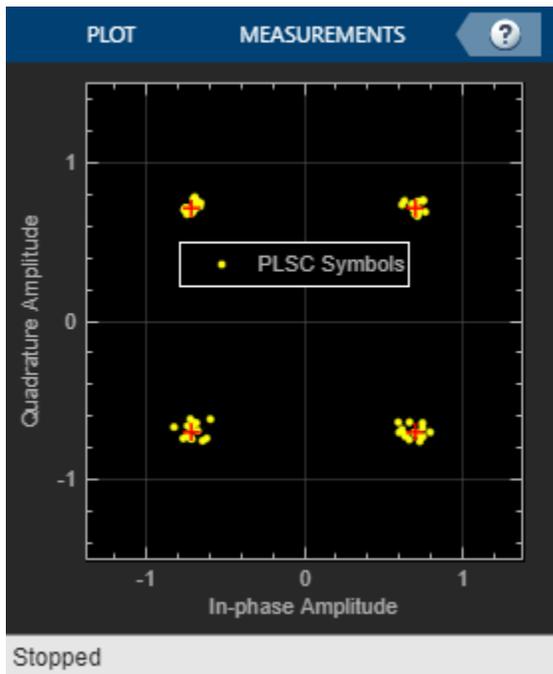
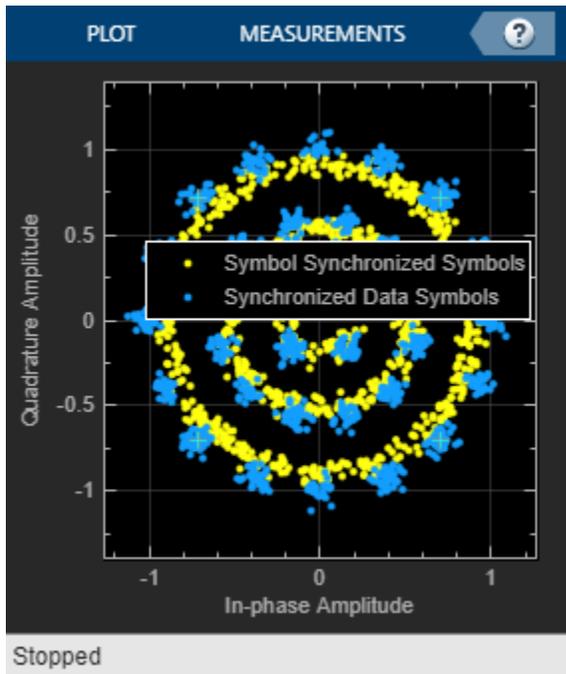
Run the `dvbs2hdlPLHeaderRecovery.slx` model.

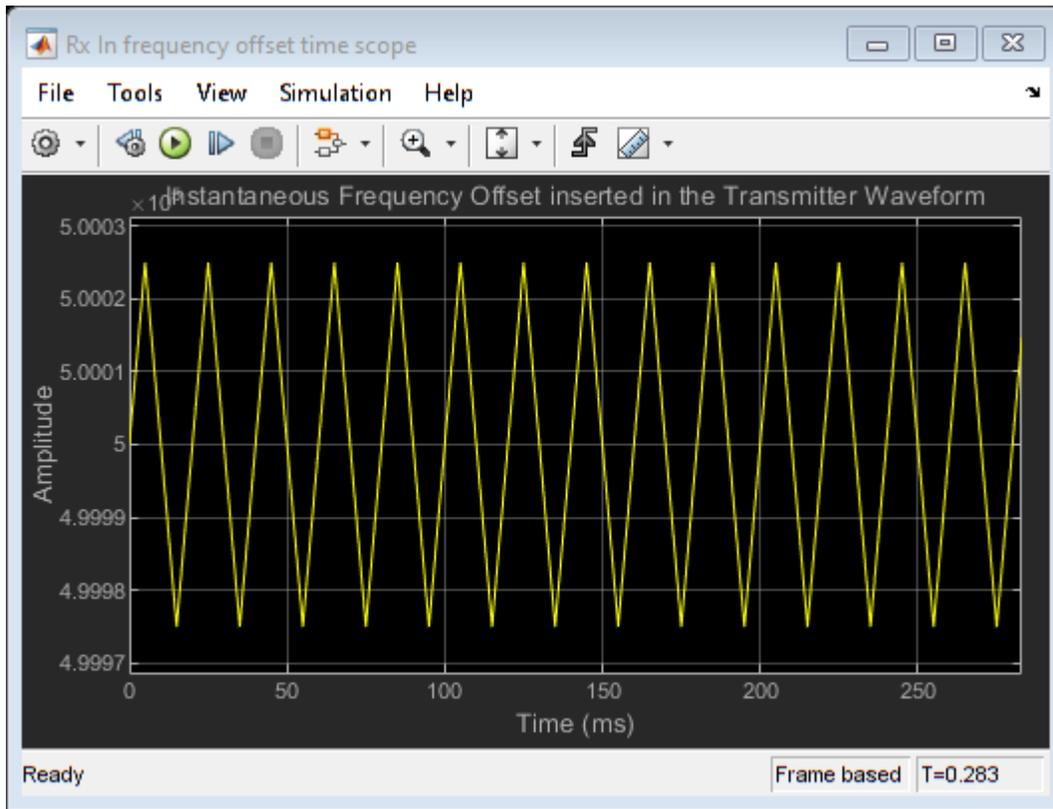
```
### Starting serial model reference simulation build.
### Model reference simulation target for dvbs2hdlSyncPLHeaderRecoveryCore is up to date.
```

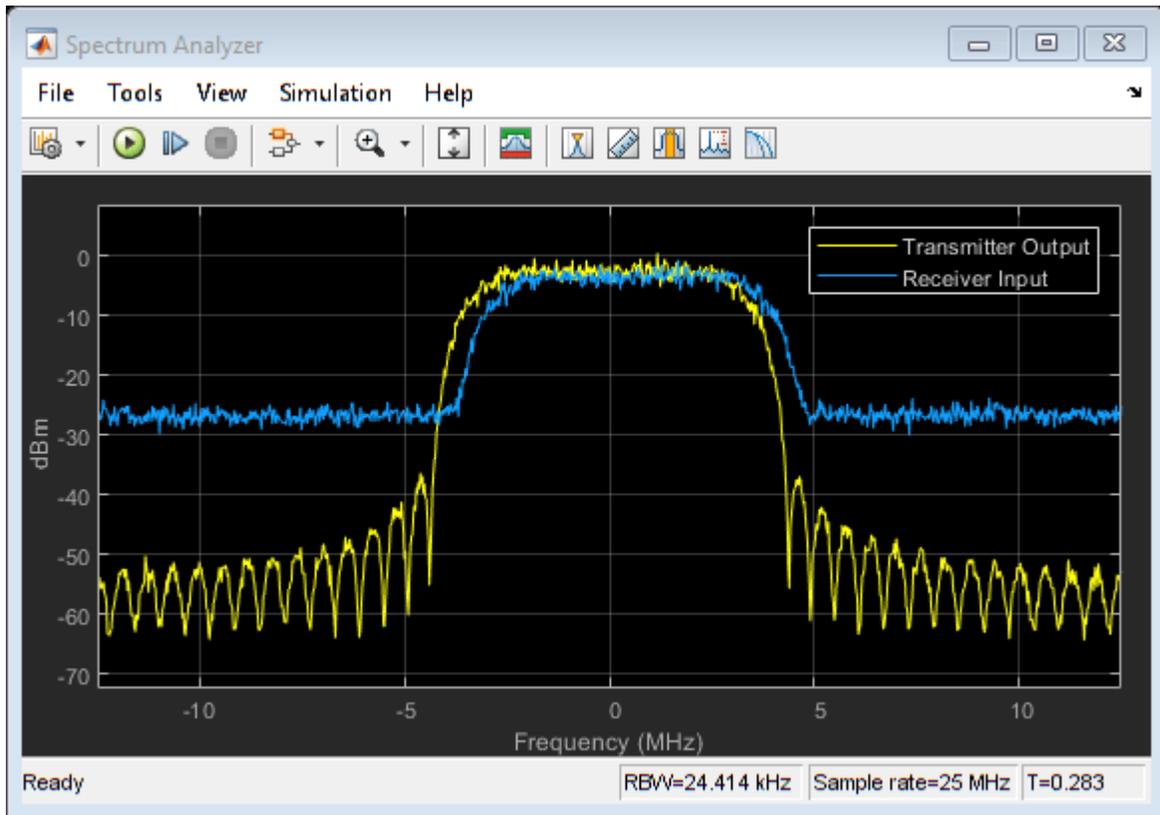
Build Summary

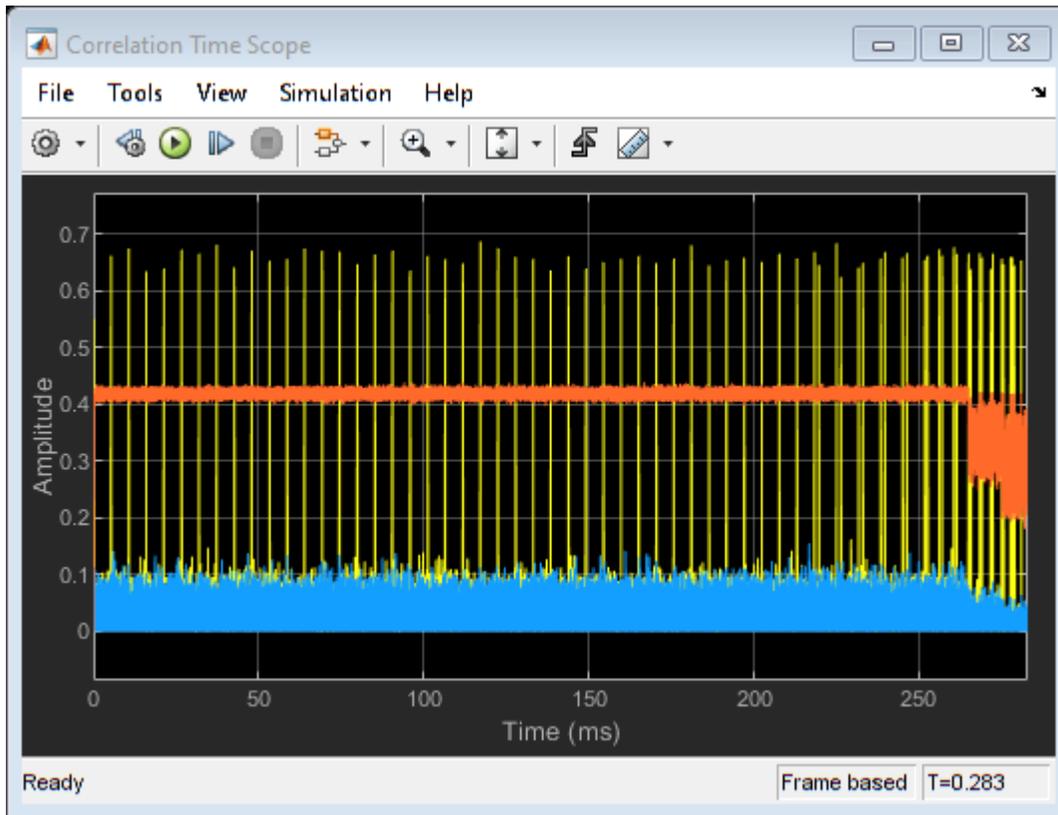
```
0 of 1 models built (1 models already up to date)
Build duration: 0h 1m 15.237s
```

```
Number of frames synced = 68 out of 68
Initial frames not compared = 35
Number of frames lost due to PL header mismatch = 0 out of 33
Number of frames lost due to BB header CRC failure = 0 out of 33
Number of packets errored = 0 out of 641
Number of bits errored = 0 out of 964064
```









HDL Code Generation

To generate the HDL code for this example, you must have HDL Coder™. Use `makehdl` and `makehdltb` commands to generate HDL code and HDL testbench for the Synchronization and PL Header Recovery subsystem. The testbench generation time depends on the simulation time.

The resulting HDL code is synthesized for a Xilinx® Zynq®-7000 ZC706 evaluation board. The post place and route resource utilization are shown in this table. The maximum frequency of operation is 205 MHz.

Resources	Usage
Slice LUT	42134
Slice Registers	63436
RAMB36	20
RAMB18	1
DSP48	248

References

- 1 ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

- 2 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 3 Marco Luise and Ruggero Reggiannini, *Carrier Frequency Recovery in All-Digital Modems for Burst-Mode Transmissions*.
- 4 Michael Rice, *Digital Communications - A Discrete-Time Approach*, Prentice Hall, April 2008.

DVB-S2 HDL Receiver

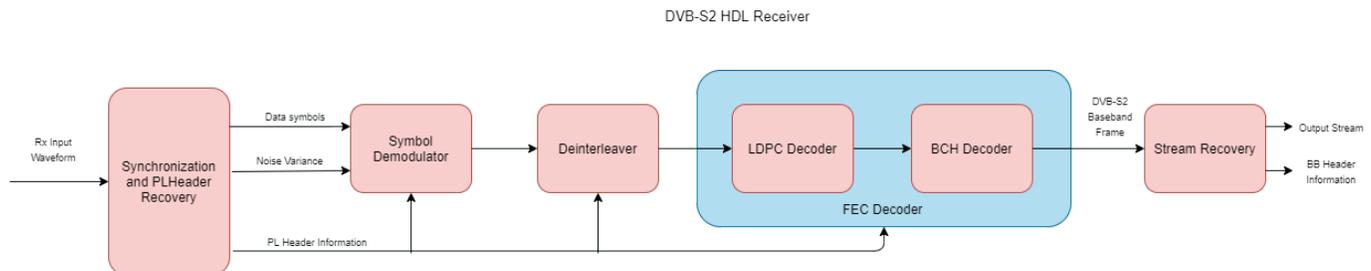
This example shows how to implement DVB-S2 receiver using Simulink® blocks that are optimized for HDL code generation and hardware implementation.

This example shows how to model a digital video broadcast satellite second generation (DVB-S2) HDL receiver system by using the “DVB-S2 HDL PL Header Recovery” (Wireless HDL Toolbox) example to demodulate, deinterleave, decode using low density parity check (LDPC) and Bose-Chaudhuri-Hocquenghem (BCH) codes, and recover the stream bits.

Model Architecture

This section explains the high-level architecture of the DVB-S2 receiver model. The Synchronization and PLHeader Recovery block extracts the data symbols, estimates noise variance, and decodes physical layer (PL) header information from the *Rx Input Waveform* signal. The Symbol Demodulator block demodulates the data symbols and computes soft bits. The Deinterleaver block deinterleaves and FEC Decoder block decodes the soft bits to extract a *DVB-S2 baseband frame* signal. The Stream Recovery block extracts the BB header information and the output stream bits from the baseband frame.

This block diagram shows the high-level architecture of the model.



File Structure

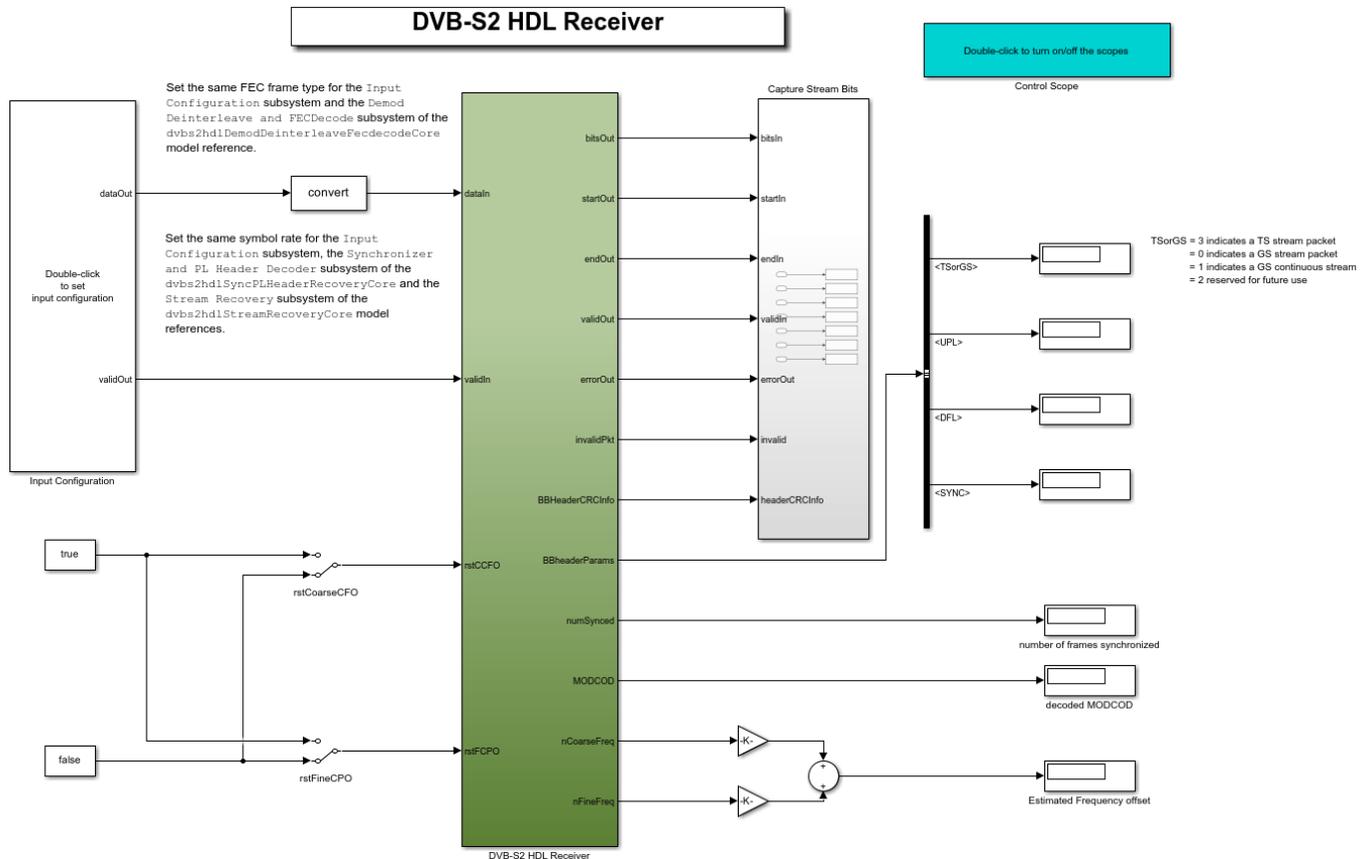
This example uses four Simulink models, five MATLAB files, and one Simulink data dictionary.

- `dvbs2hdlReceiver.slx` — Top-level Simulink model.
- `dvbs2hdlSyncPLHeaderRecoveryCore.slx` — Model reference that synchronizes time, frequency, and phase, and decode PL header.
- `dvbs2hdlDemodDeinterleaveFecdecodeCore.slx` — Model reference that demodulates the symbols, deinterleaves the demodulated soft bits, and decodes the deinterleaved soft bits using forward error correction (FEC). It discards the frames that does not support the configuration according to [1].
- `dvbs2hdlStreamRecoveryCore.slx` — Model reference that recovers the stream of data bits.
- `getdvbs2LDPCParityMatrices.m` — Download the MAT file that stores LDPC parity check matrices that are used to generate the receiver input waveform.
- `dvbs2hdlRxParameters.m` — Generate parameters for the `dvbs2hdlSyncPLHeaderRecoveryCore.slx` model reference.
- `dvbs2hdlPhaseNoise.m` — Introduce phase noise to the input sequence.

- `dvbs2hdlRxInit.m` — Generate the transmitter waveform and initialize the `dvbs2hdlSyncPLHeaderRecoveryCore.slx` model reference.
- `dvbs2hdlReceiverVerify.m` — Gather PL header parameters and stream recovered bits.
- `dvbs2hdlReceiverData.sldd` — Simulink data dictionary to store bus signal configurations.

System Interface

This figure shows the top-level overview of the `dvbs2hdlReceiver.slx` model.



Copyright 2021 The MathWorks, Inc.

Model Inputs

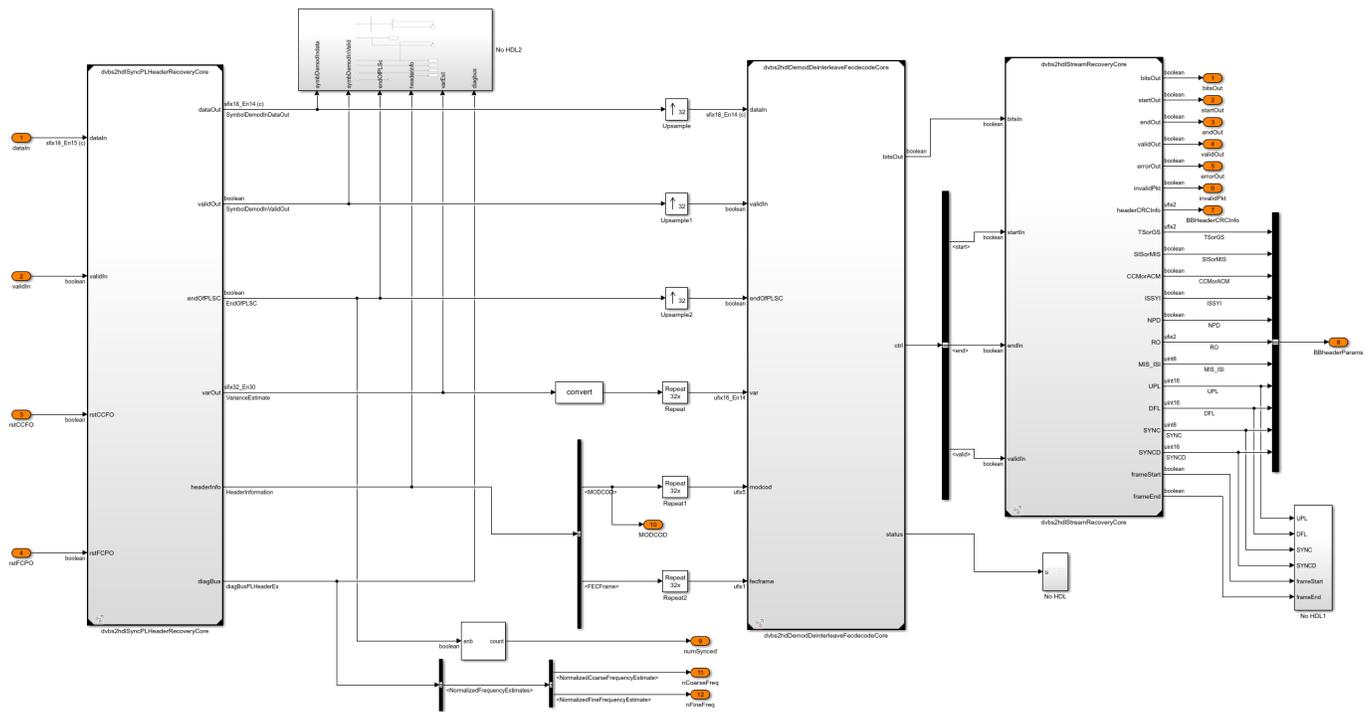
- **dataIn** — Input data, specified as an 18 bit complex data with a sample rate that is four times the symbol rate.
- **validIn** — Control signal to validate the **dataIn**, specified as a Boolean scalar.
- **rstCCFO** — Control signal to reset the coarse frequency compensation loops, specified as a Boolean scalar.
- **rstFCPO** — Control signal to reset the fine phase compensation loops, specified as a Boolean scalar.

Model Outputs:

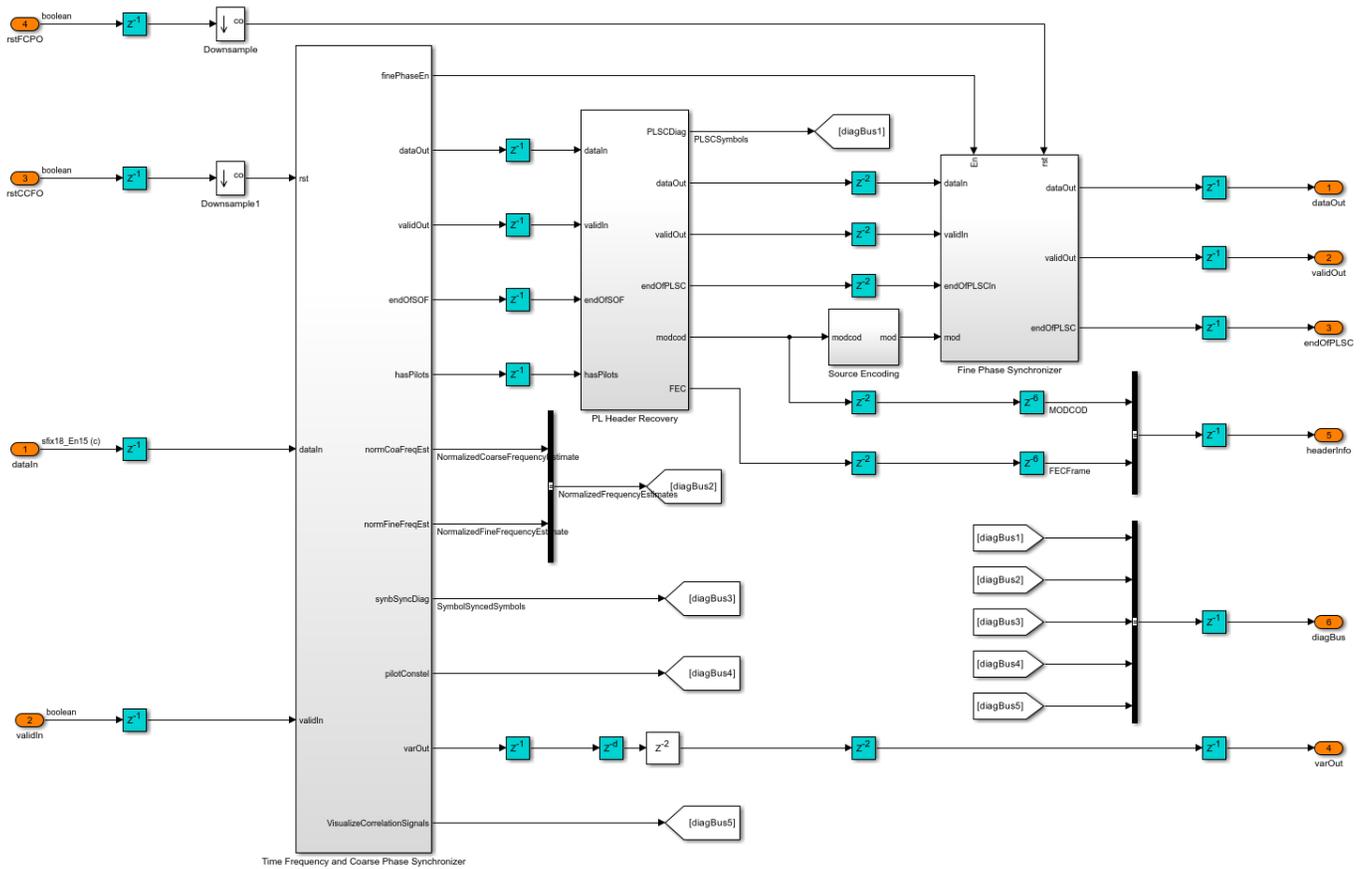
- **diagBus** — Bus signal with diagnosis information.
- **bitsOut** — Decoded stream bits, returned as a Boolean scalar.
- **startOut** — Control signal for start of **bitsOut** stream bits, returned as a Boolean scalar.
- **endOut** — Control signal for end of **bitsOut** stream bits, returned as a Boolean scalar.
- **validOut** — Control signal to validate the **bitsOut**, returned as a Boolean scalar.
- **errorOut** — Control signal to indicate packet CRC failures. It can be ignored for non-packetized continuous streams.
- **invalidPkt** — Control signal to indicate invalid packets that can be discarded. It can be ignored for non-packetized continuous streams.
- **headerCRCInfo** — Header CRC status, returned as a 2 bit real data. MSB bit high indicates a CRC error, and LSB high indicates when the CRC is considered.
- **numSynced** — Number of frames synchronized, returned as a 32 bit scalar integer
- **MODCOD** — Decoded **MODCOD**, returned as a 5 bit scalar integer.
- **nCoarseFreq** — Estimated normalized (with sample rate) coarse frequency offset, returned as a 21 bit scalar.
- **nFineFreq** — Estimated normalized (with symbol rate) fine frequency offset, returned as a 21 bit scalar.
- **BBHeaderParams** — The following are the list of BB header parameters:
- **TSorGS** — Input stream format, returned as a 2 bit real data.
- **SISorMIS** — Single or multiple input stream input, returned as a Boolean scalar.
- **CCMorACM** — Constant coding modulation (CCM), or adaptive coding modulation (ACM) or variable coding modulation (VCM), returned as a Boolean scalar.
- **RO** — Roll-off factor, returned as a 2 bit real data.
- **UPL** — User packet length (UPL), returned as a 16 bit real data.
- **DFL** — Data field length (DFL), returned as a 16 bit real data.
- **SYNC** — SYNC word, returned as an 8 bit real data.
- **ISSYI** — Input stream synchronization indicator (ISSYI), returned as a Boolean scalar.
- **NPD** — Null packet detection (NPD), returned as a Boolean scalar.
- **MIS_ISI** — Input stream identifier (ISI) for multiple input stream input, returned as an 8 bit real data. For single stream, this is reserved by the standard.
- **SYNCD** — Start location of SYNC word in number of bits from start of data field, returned as a 16 bit real data.

Model Structure

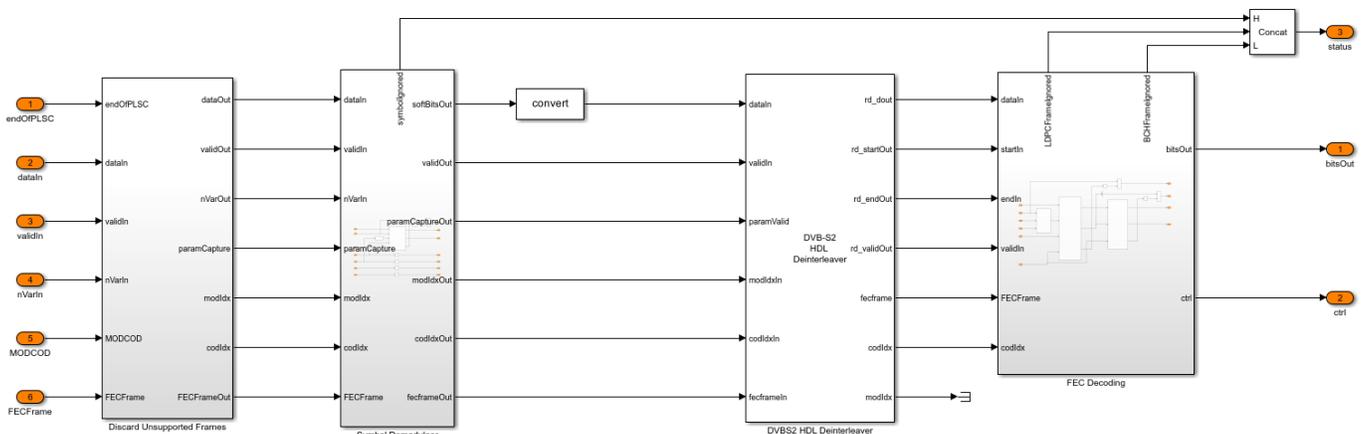
This figure shows the top-level model of the DVB-S2 HDL Receiver subsystem. The subsystem comprises three model references, `dvbs2hdlSyncPLHeaderRecoveryCore`, `dvbs2hdlDemodDeinterleaveFecdecodeCore`, and `dvbs2hdlStreamRecoveryCore`.



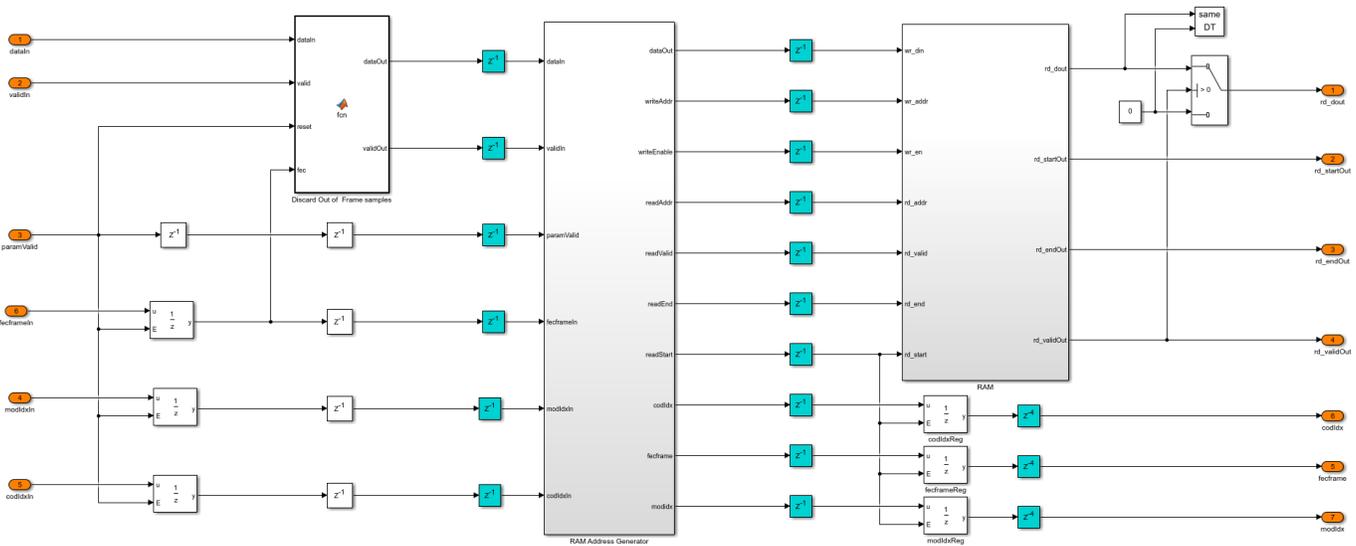
dvbs2hdlSyncPLHeaderRecoveryCore — Synchronizes the input receiver waveform, estimates noise variance, and decodes PL header information. For more information, see “DVB-S2 HDL PL Header Recovery” (Wireless HDL Toolbox) example.



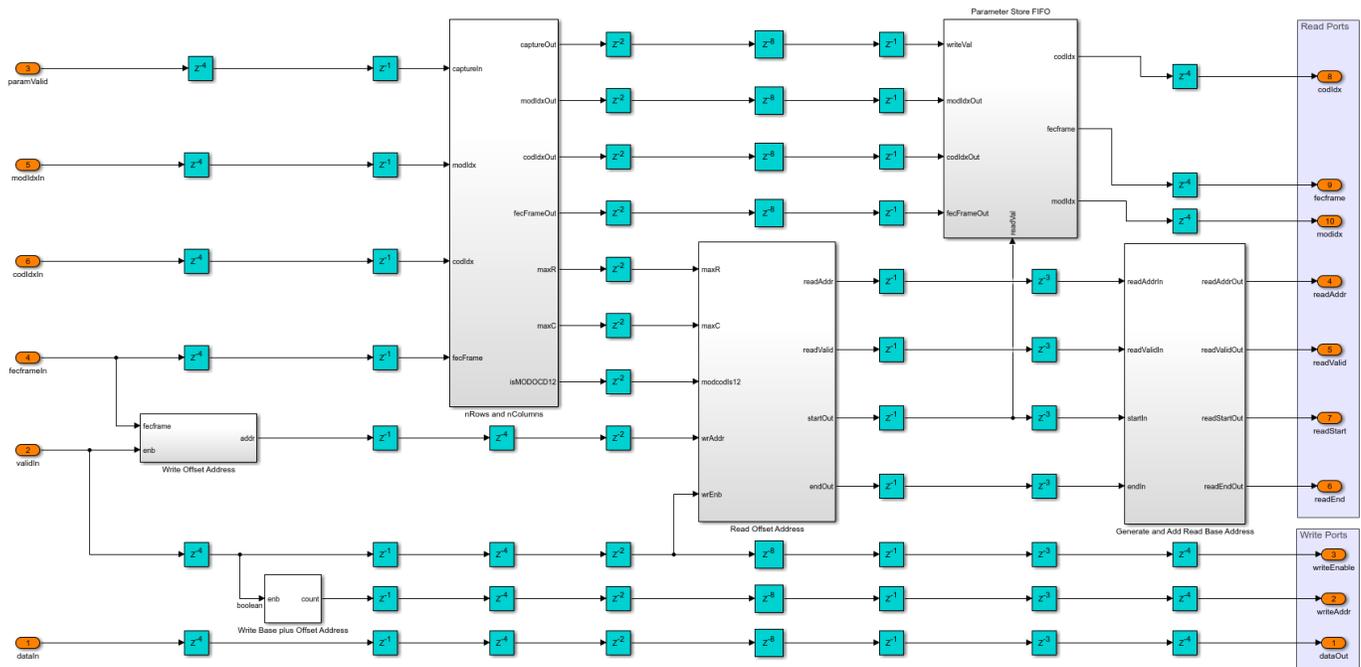
dvbs2hdlDemodDeinterleaveFecdecodeCore – Demodulates the input symbols using DVB-S2 Symbol Demodulator (Wireless HDL Toolbox) block to extract soft bits, deinterleave the soft bits using DVB-S2 Deinterleaver subsystem, and FEC decodes (LDPC and BCH decodes) the soft bits using the DVB-S2 LDPC Decoder (Wireless HDL Toolbox) block and the DVB-S2 BCH Decoder (Wireless HDL Toolbox) block to extract baseband frame (BBFrame). The baseband frame streams into the dvbs2hdlStreamRecoveryCore model reference.



The DVB-S2 Deinterleaver subsystem in the dvbs2hdlDemodDeinterleaveFecdecodeCore model reference continuously stores the soft bits received from the DVB-S2 Symbol Demodulator (Wireless HDL Toolbox) inside the RAM subsystem. The RAM Address Generator subsystem generates the read and write logic to the RAM for deinterleaving.



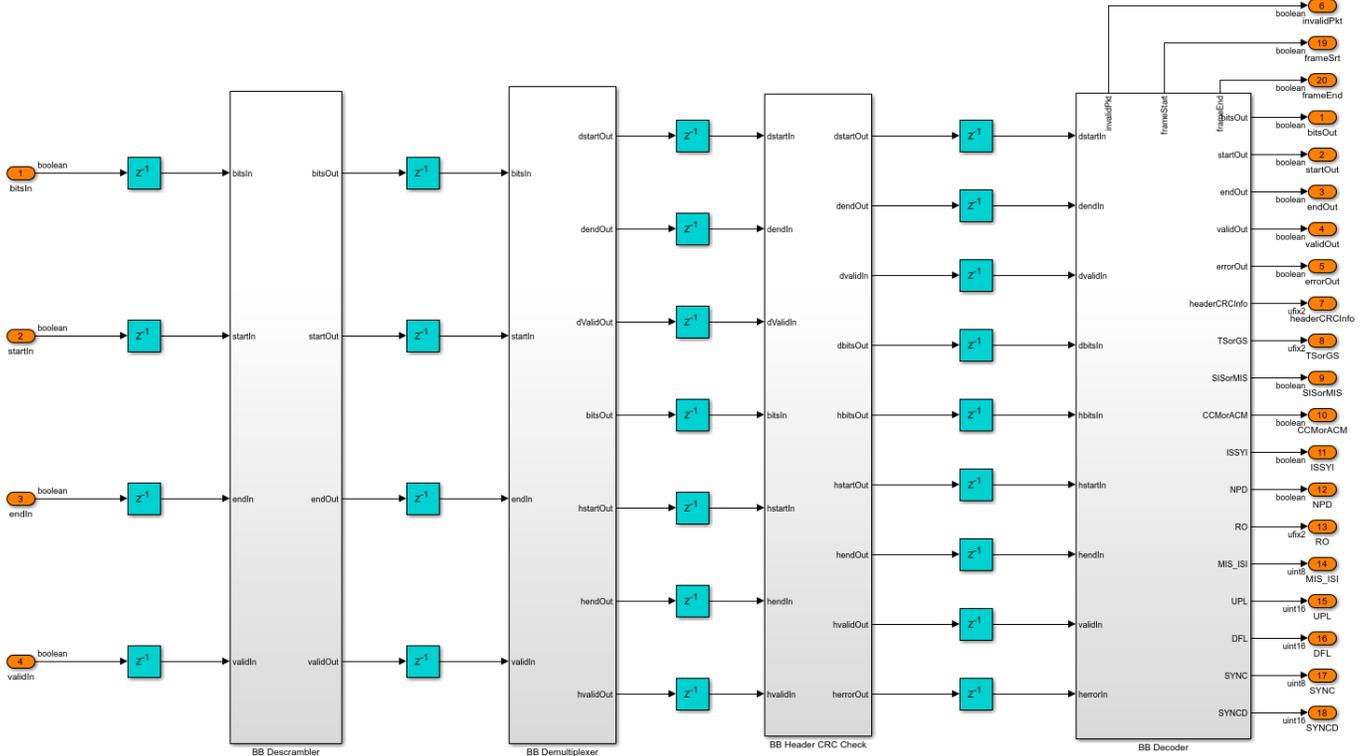
In the RAM Address Generator subsystem, the `nRows` and `nColumns` subsystem stores the number of rows and columns of each of the possible configuration in look-up tables (LUT). Based on the PL header parameters, the number of rows and columns is decided for deinterleaving. The `Read Offset Address` subsystem generates the deinterleaver indices of each frame as an offset address. The `Generate and Add Read Base Address` subsystem adds the offset address with a base read address to get the actual address of the soft-bit stored in the RAM. The `Parameter Store FIFO` subsystem stores the PL header parameters and reads these parameters in synchronous with start of each frame.



This table shows the rows and columns considered for deinterleaving for each of the configurations.

Modulation	Rows (Normal)	Rows (Short)	Columns
QPSK	64800	16200	1
8-PSK	21600	5400	3
16-APSK	16200	4050	4
32-APSK	12960	3240	5

dvbs2hdlStreamRecoveryCore — Decodes the BB header and recovers the stream bits. The BB Descrambler subsystem descrambles the baseband frame. The BB Demultiplexer subsystem demultiplexes the descrambled frame into BB header and data bits. The BB Header CRC Check subsystem uses the General CRC Syndrome Detector HDL Optimized block to check the CRC status and discards the baseband frames that fails CRC check. The BB Decoder subsystem extracts the BB header information and the data field. For packetized stream of bits, the control signals are generated to indicate the start, end, and validity of bits for each packet. The General CRC Syndrome Detector HDL Optimized block checks the CRC status of each packet. For continuous stream of bits, the data field is passed on to the output.



Run the Model

Set the symbol rate, MODCOD, FECFrame type values, input stream format, user packet length and channel impairments on the mask of the Input Configuration subsystem and run the dvbs2hdlReceiver model. Alternatively, to run the model, execute this command at the MATLAB command window.

```
sim dvbs2hdlReceiver
```

The MODCOD value must be a row vector. Each element of the row vector corresponds to a frame.

Note: Use QPSK modulated frames initially to achieve time frequency and phase synchronization.

Verification and Results

Run the dvbs2hdlReceiver.slx model. The model utilizes 120 short QPSK frames for synchronization.

```
### Starting serial model reference simulation build.
### Successfully updated the model reference simulation target for: dvbs2hdlDemodDeinterleaveFec
### Successfully updated the model reference simulation target for: dvbs2hdlStreamRecoveryCore
### Successfully updated the model reference simulation target for: dvbs2hdlSyncPLHeaderRecovery
```

Build Summary

Simulation targets built:

Model	Action	Rebuild Reason
-------	--------	----------------

```
dvbs2hdlDemodDeinterleaveFecdecodeCore Code generated and compiled. dvbs2hdlDemodDeinterleaveF
dvbs2hdlStreamRecoveryCore Code generated and compiled. dvbs2hdlStreamRecoveryCore
dvbs2hdlSyncPLHeaderRecoveryCore Code generated and compiled. dvbs2hdlSyncPLHeaderRecoveryCore
```

3 of 3 models built (0 models already up to date)

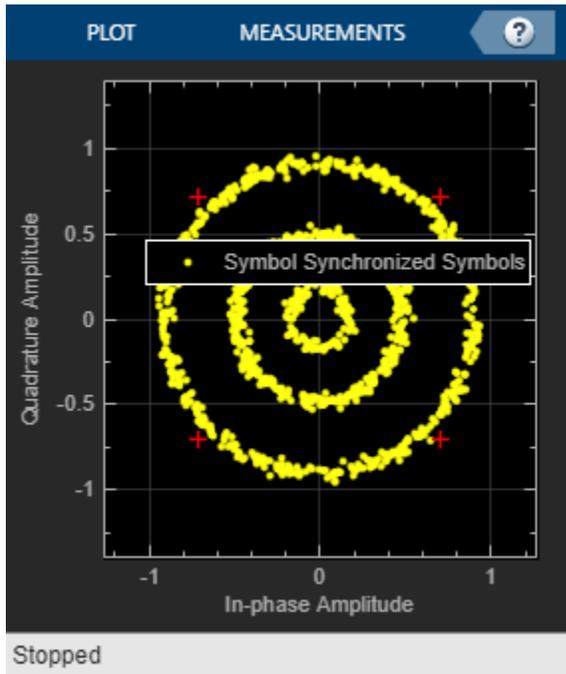
Build duration: 0h 7m 7.8594s

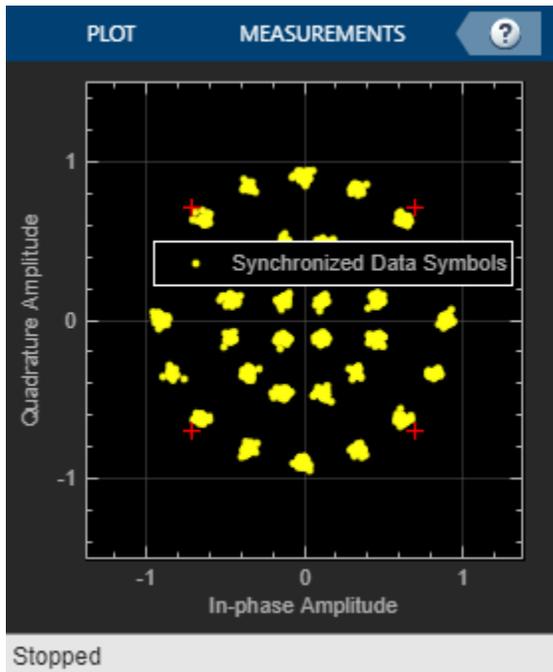
Number of frames synced = 124 out of 124

Initial frames not compared = 120

Number of frames lost due to BB Header CRC failure = 0 out of 4

Number of packets lost due to packet CRC failure = 0 out of 27





HDL Code Generation

To generate the HDL code for this example, you must have HDL Coder™. Use `makehdl` and `makehdl tb` commands to generate HDL code and HDL testbench for the DVB-S2 HDL Receiver subsystem. The testbench generation time depends on the simulation time.

The resulting HDL code is synthesized for a Xilinx® Zynq® UltraScale+ RFSoc ZCU111 board. The post place and route resource utilization are shown in this table. The maximum frequency of operation is 181 MHz.

Resources	Usage
CLB LUT	84262
CLB Registers	98936
RAMB36	821
RAMB18	1
DSP48	302

References

- 1 ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
- 2 ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

GPS HDL Acquisition and Tracking Using C/A Code

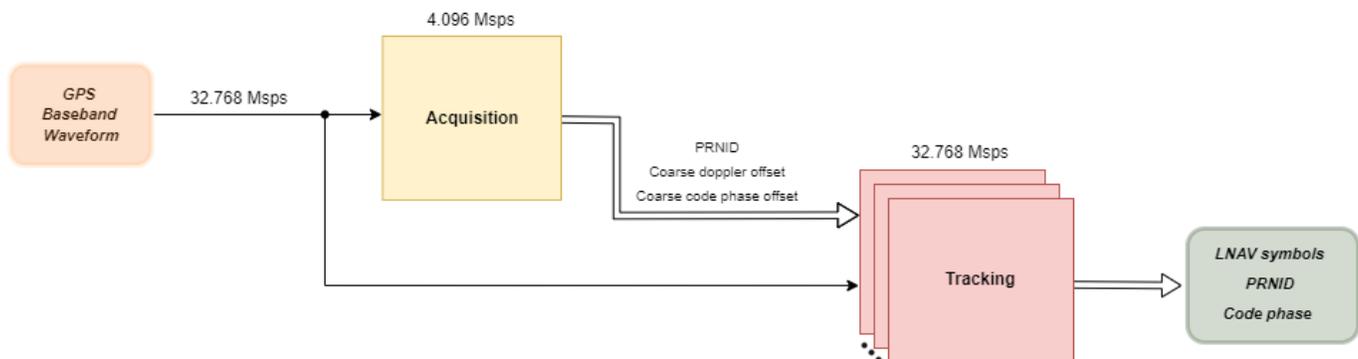
This example shows how to acquire and track multiple Global Positioning System (GPS) satellite signals from a GPS baseband waveform using Simulink® blocks that are optimized for HDL code generation and hardware implementation. You use the L1 Coarse/Acquisition (C/A) code in the input waveform to perform signal acquisition and track the satellites. In acquisition, you detect the satellite signals in the waveform and estimate the coarse Doppler and C/A code phase offsets for the detected satellites. In tracking, you fine-tune the estimates and correct them to recover the legacy navigation (LNAV) symbols. The LNAV symbols can be used for GPS position estimation.

Model Overview

The model that you use in this example mainly consists of acquisition and tracking modules. The input to the model is a GPS baseband waveform with a sample rate of 32.768 Msps. This signal contains multiple satellite waveforms with Doppler offsets, code phase offsets, and additive white gaussian noise (AWGN) noise in it. The model upsamples the GPS waveform by a factor of six and increases the clock from 32.768 MHz to 196.6 MHz to achieve faster acquisition.

The Acquisition block decimates the GPS waveform from 32.768 Msps to 4.096 Msps and operates at this lower sampling rate. This block detects the satellites and generates coarse estimates for them.

The Tracking block uses the GPS waveform at 32.768 Msps, the pseudorandom noise identifier (PRNID) of the detected satellites, and the coarse estimates to track the satellites. At least four satellites are required for GPS position estimation, so tracking does not start until the acquisition process detects four satellites. The Tracking block contains eight instances of tracking logic to simultaneously track eight satellites. The Tracking block uses phase-locked loop, frequency-locked loop, and delay-locked loop to recover the pi/2-BPSK modulated LNAV symbols of each detected satellite. This figure shows an overview of the example model.



File Structure

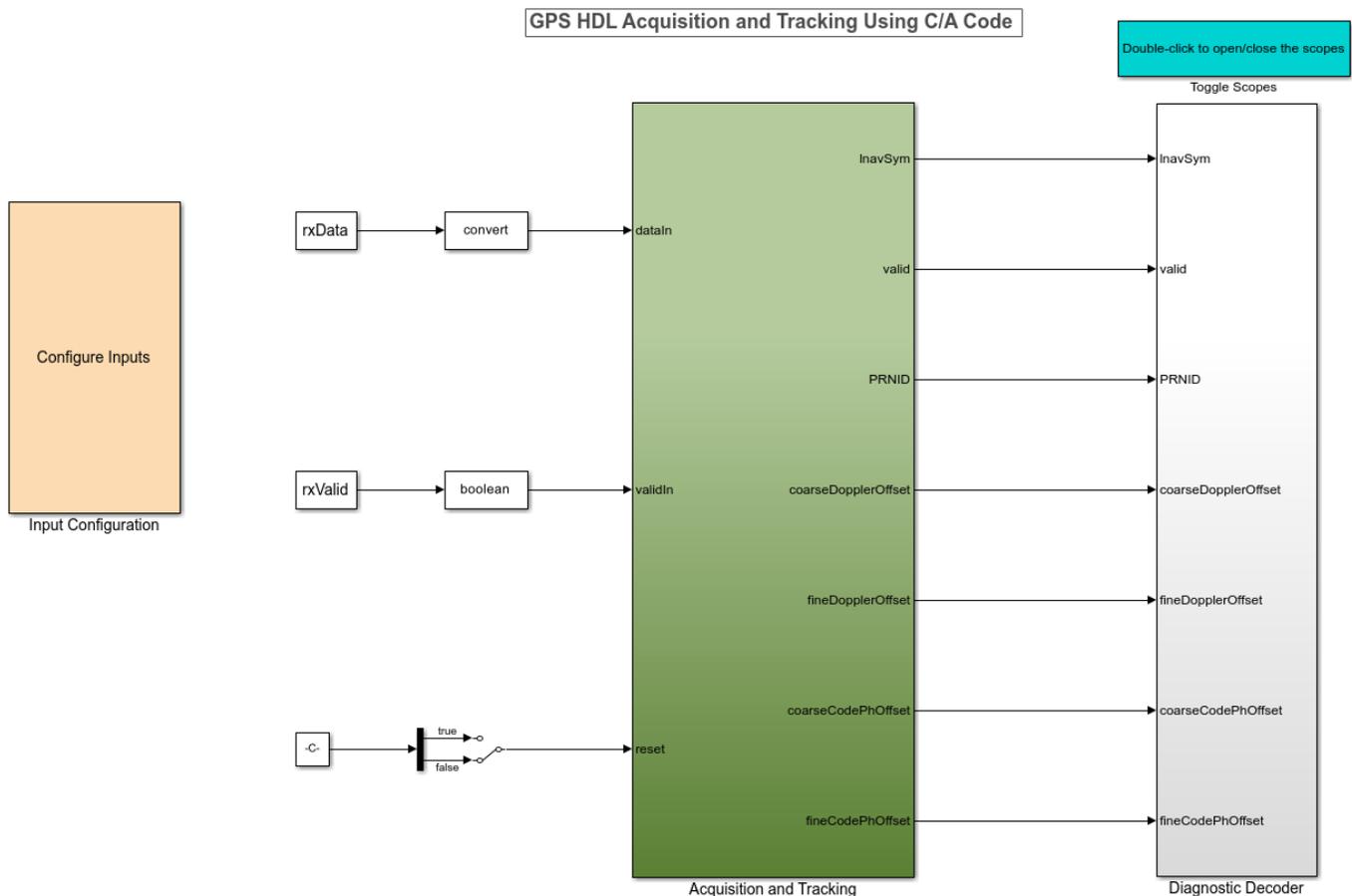
This example uses one Simulink model, two functions and two scripts.

- `gpshdlAcquisitionTracking` — Model to acquire signals and track satellites.
- `gpshdlAcquisitionAndTrackingUsingCACodeInit` — Generates all the parameters and inputs required to run the `gpshdlAcquisitionTracking` model. The model calls this script in the `InitFcn` callback.

- `gpshdlAcquisitionAndTrackingUsingCACodeParameters` — Generates the parameters required to run the model. The model calls this function in the `gpshdlAcquisitionAndTrackingUsingCACodeInit` script.
- `gpshdlGenerateRxInput` — Generates the input GPS waveform for the receiver. The model calls this function in the `gpshdlAcquisitionAndTrackingUsingCACodeInit` script.
- `gpshdlAcquisitionTrackingValidate` — Validates the outputs of the model and plots them for visualization. The model calls this function in the `StopFcn` callback.

Model Interface

This figure shows the structure of the `gpshdlAcquisitionTracking` model.



Copyright 2022 The MathWorks, Inc.

Model Inputs

- **dataIn** — Input data, specified as 18-bit complex data.
- **validIn** — Control signal to validate the **dataIn** signal, specified as a Boolean scalar.
- **reset** — Control signal to reset the receiver. The receiver restarts from acquisition.

Model Outputs:

The output ports are vectors of length 8 because the example tracks eight satellites simultaneously.

- **lnavSym** — Legacy navigation symbols in the input waveform, specified as 16-bit complex data.
- **validOut** — Control signal to validate all the output ports, specified as a Boolean signal.
- **PRNID** — PRNIDs of detected satellites, specified as a 6-bit unsigned integer.
- **coarseDopplerOffset** — Estimated coarse Doppler offset of the detected satellites, specified as a 16-bit integer.
- **fineDopplerOffset** — Estimated fine Doppler offset of the detected satellites, specified as 25-bit real data.
- **coarseCodePhOffset** — Estimated coarse C/A code phase offset of the detected satellites, specified as 20-bit real data.
- **fineCodePhOffset** — Estimated fine C/A code phase offset of the detected satellites, specified as 20-bit real data.

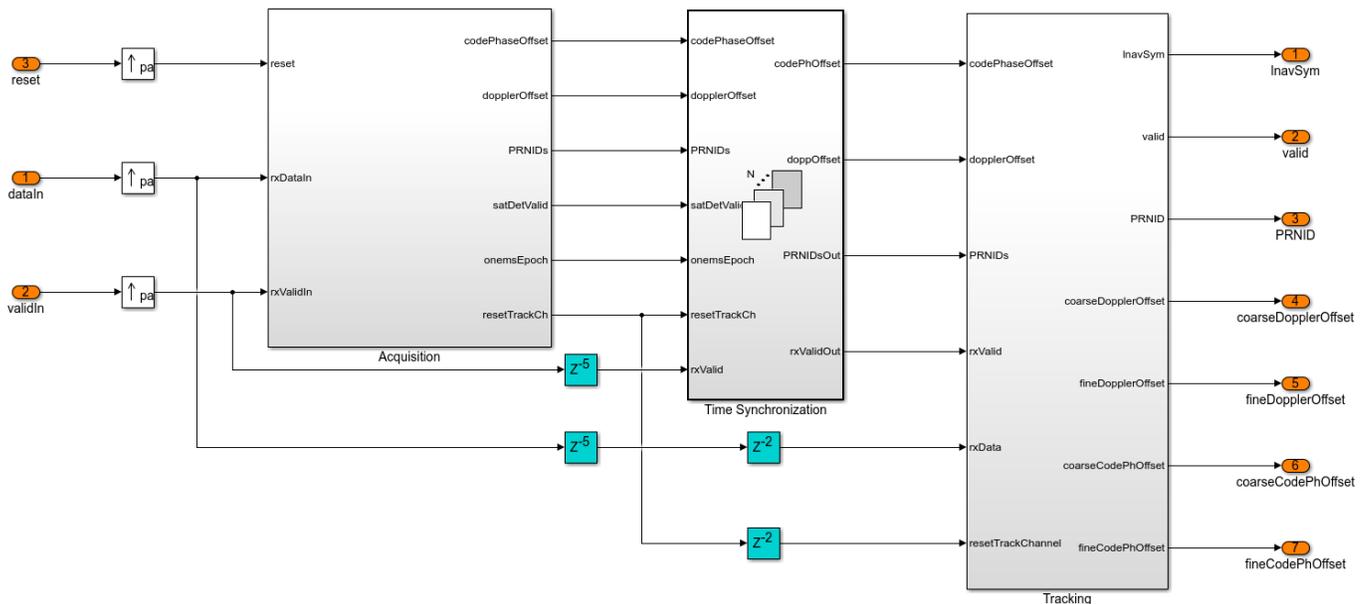
Input Configuration

Double-click the Input Configuration subsystem to configure the transmitter parameters and channel impairments.

- **Number of LNAV data bits** — Number of LNAV data bits to generate transmitter waveform. Use a minimum of eight bits to ensure successful tracking convergence.
- **Number of satellites** — Number of satellites to include in the transmitter waveform, specified as an integer in the range [1,8].
- **Satellite PRNIDs** — PRNIDs of satellites. This value must be a column vector of size equal to number of satellites. Each PRNID must be an integer in the range [1,32].
- **SNR (in dB)** — Signal-to-noise ratio (SNR) of individual satellites in the transmitter waveform, in dB, specified as a column vector of size equal to the number of satellites. The minimum SNR value must be -20 dB.
- **Peak Doppler offset** — Maximum Doppler offset to introduce in the satellite waveform, in Hz, specified as a column vector of size equal to the number of satellites. Each entry of this vector must be in the range [-10000, 10000].
- **Doppler rate** — Rate of change of the Doppler offset, specified in Hz/sec, specified as a column vector of size equal to the number of satellites. Each entry of this vector must be no greater than 1000.
- **C/A code phase offset** — C/A code delay to introduce in the waveform, specified as a column vector of size equal to the number of satellites. Each entry of this vector must be in the range (-1023, 1023).

Model Structure

This figure shows the Acquisition and Tracking subsystem. This subsystem consists of the Acquisition, Time Synchronization, and Tracking subsystems.



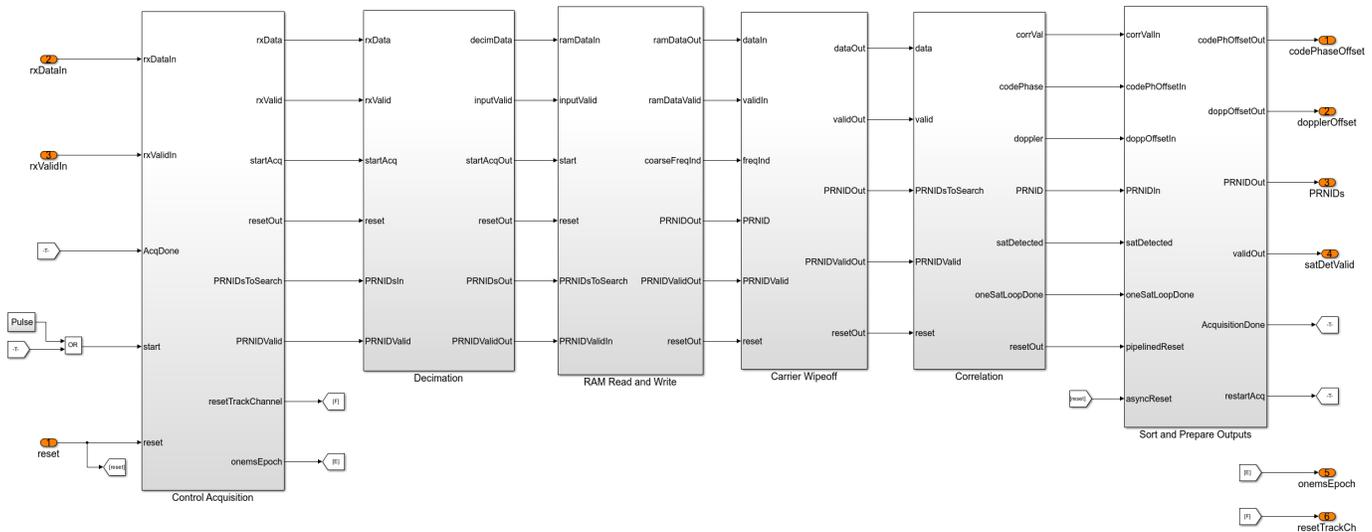
Acquisition Subsystem

The Acquisition subsystem accepts the GPS waveform at a sampling rate of 32.768 Msps, decimates it to 4.096 Msps, and stores one millisecond duration of the decimated waveform. The subsystem selects coarse Doppler frequencies sequentially from -10 kHz to 10 kHz in steps of 1 kHz, generates local carrier waveforms at these frequencies, compensates for these frequencies in the decimated waveform, and outputs carrier-wiped-off waveform. The subsystem converts the carrier-wiped-off waveform into the frequency domain using a 4096-point fast Fourier transform (FFT). The subsystem fetches the frequency-domain C/A code from a look-up table (LUT) and correlates it with the waveform to find the correlation peak. When the peak is greater than a dynamic threshold, the subsystem detects the satellite with the C/A code is detected and the corresponding Doppler frequency and C/A code phase are the coarse estimates of the satellite. The subsystem performs this frequency-domain correlation for four satellites in parallel and for eight sequential searches to finish searching all 32 GPS satellites. The subsystem then sorts and selects the eight detected satellites with the strongest correlation peaks. The subsystem also generates a 1 ms epoch signal, which asserts, for every 1 ms, to use that signal for time synchronization.

The Acquisition subsystem contains these main subsystems:

- **Control Acquisition** — Generates control signals to start acquisition and selects the satellite PRNIDs to search.
- **Decimation** Decimates the input waveform from 32.768 Msps to 4.096 Msps using Cascaded Integrator Comb (CIC) and Finite Impulse Response (FIR) decimation.
- **RAM Read and Write** — Writes one millisecond of decimated waveform to the RAM and reads the waveform multiple times from the RAM until acquisition finishes.
- **Carrier Wipeoff** — Generates the local waveform at coarse Doppler frequencies using a numerically controlled oscillator (NCO) and compensates for the Doppler frequencies in the decimated waveform to output carrier-wiped-off waveform.
- **Correlation** — Performs frequency-domain correlation of the carrier-wiped-off waveform with four satellite C/A codes simultaneously. This subsystem also finds correlation peaks, generates a threshold, and compares the peaks with the threshold to detect satellites.

- **Sort and Prepare Outputs** — Sorts the PRNIDs, coarse Doppler frequencies, and coarse code phase offset values of the detected satellites in decreasing order of their correlation peaks and outputs the eight satellites with the strongest peaks .



Time Synchronization Subsystem

The Time Synchronization subsystem accepts the GPS waveform, the detected satellite PRNIDs, coarse Doppler frequencies, coarse code phase offsets, and the 1 ms epoch signal from the Acquisition subsystem. The Time Synchronization subsystem synchronizes the input GPS waveform after the Acquisition subsystem finishes detecting the satellites and when the incoming 1 ms epoch signal asserts.

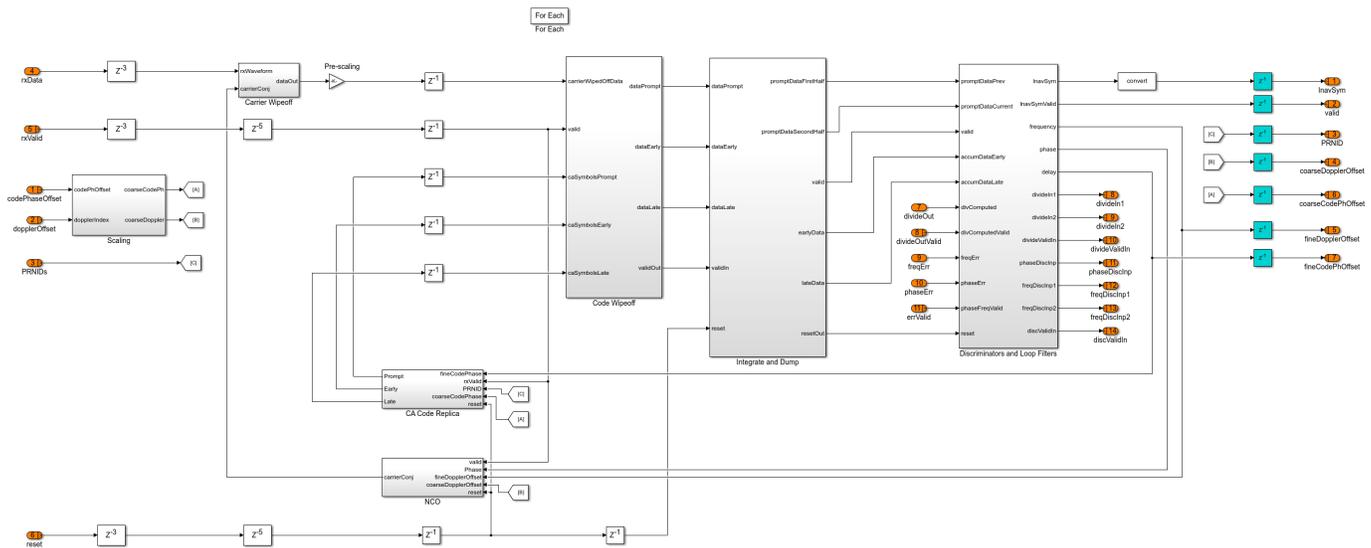
Tracking Subsystem

The Tracking subsystem tracks the satellites detected through acquisition. The subsystem accepts the time-synchronized waveform at 32.768 Msp/s, the detected satellite PRNIDs, coarse Doppler offsets, and code phase offsets. The subsystem uses the coarse Doppler estimate and phase estimate to generate a local carrier using an NCO, removes the Doppler offset from the waveform, and returns a carrier-wiped-off waveform. The subsystem uses the detected PRNID and the coarse code phase offset to fetch replica C/A code from an LUT. The subsystem generates early, prompt, and late versions of this C/A code to correlate with the carrier-wiped-off waveform. The subsystem integrates these correlated outputs for every 1 millisecond (predetection integration time) and returns integrated samples after every millisecond. The integrated prompt outputs are the LNAV symbols of the receiver. The subsystem uses the integrated early and late outputs to estimate delay error and the integrated prompt output to estimate frequency and phase errors. The subsystem filters these errors using loop filters and feeds the filtered values to the NCO and C/A code LUT to aid local carrier generation and replica C/A code generation. This tracking logic applies to a single satellite. Similarly, the Tracking subsystem generates eight instances of this tracking logic and tracks eight satellites simultaneously.

The Tracking subsystem contains the Tracking Core subsystem that carries out the tracking logic. The Tracking Core subsystem contains these main subsystems:

- **NCO** — Accepts the coarse Doppler offset and filtered fine Doppler offset. The subsystem generates a local carrier to compensate for the Doppler offset in the input waveform.

- **CA Code Replica** — Accepts the detected satellite PRNID, coarse code phase offset, and filtered fine code phase offset to generate replica C/A code. This subsystem generates early, prompt, and late versions of the C/A code, each separated by a half C/A chip duration.
- **Code Wipeoff** — Multiplies the carrier-wiped-off waveform with the generated early, prompt, and late C/A codes to give out code-wiped-off waveforms.
- **Integrate and Dump** — Integrates the early, prompt, and late code-wiped-off waveforms every 1 millisecond and outputs the integrated values.
- **Discriminators and Loop Filters** — Uses the integrated prompt value to estimate the frequency and phase errors and uses the integrated early and late values to estimate the delay error. The first- and second-order loop filters filter the generated errors to provide fine estimates.



Run Model

Open the `gpshdlAcquisitionTracking` model and double-click the **Input Configuration** subsystem to change transmitter configuration and channel impairments. Run the model.

Note: It may take around 25 minutes to complete the simulation.

```
### Building the rapid accelerator target for model: gpshdlAcquisitionTracking
### Successfully built the rapid accelerator target for model: gpshdlAcquisitionTracking
```

Build Summary

Top model rapid accelerator targets built:

Model	Action	Rebuild Reason
<code>gpshdlAcquisitionTracking</code>	Code generated and compiled.	Code generation information file does not exist

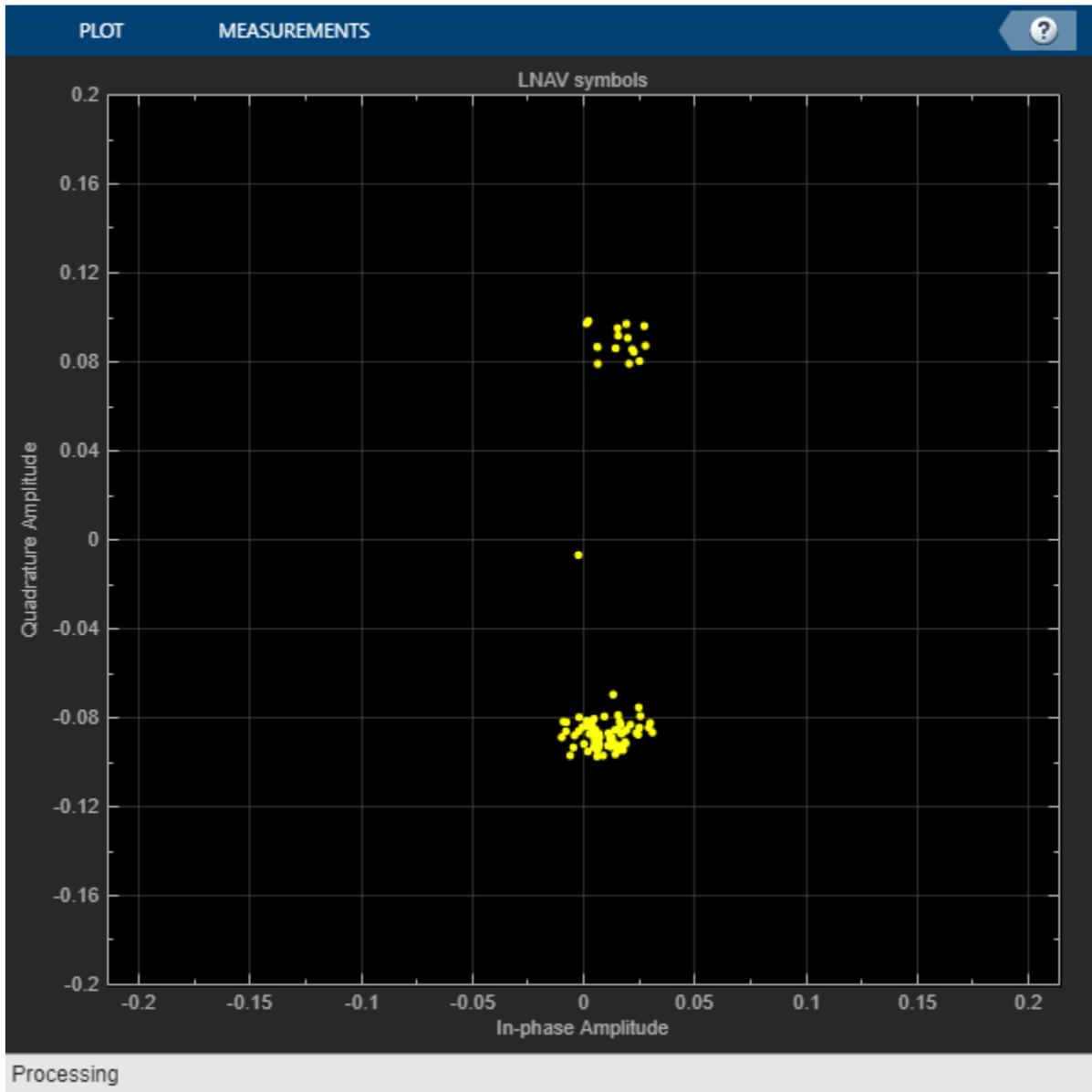
1 of 1 models built (0 models already up to date)

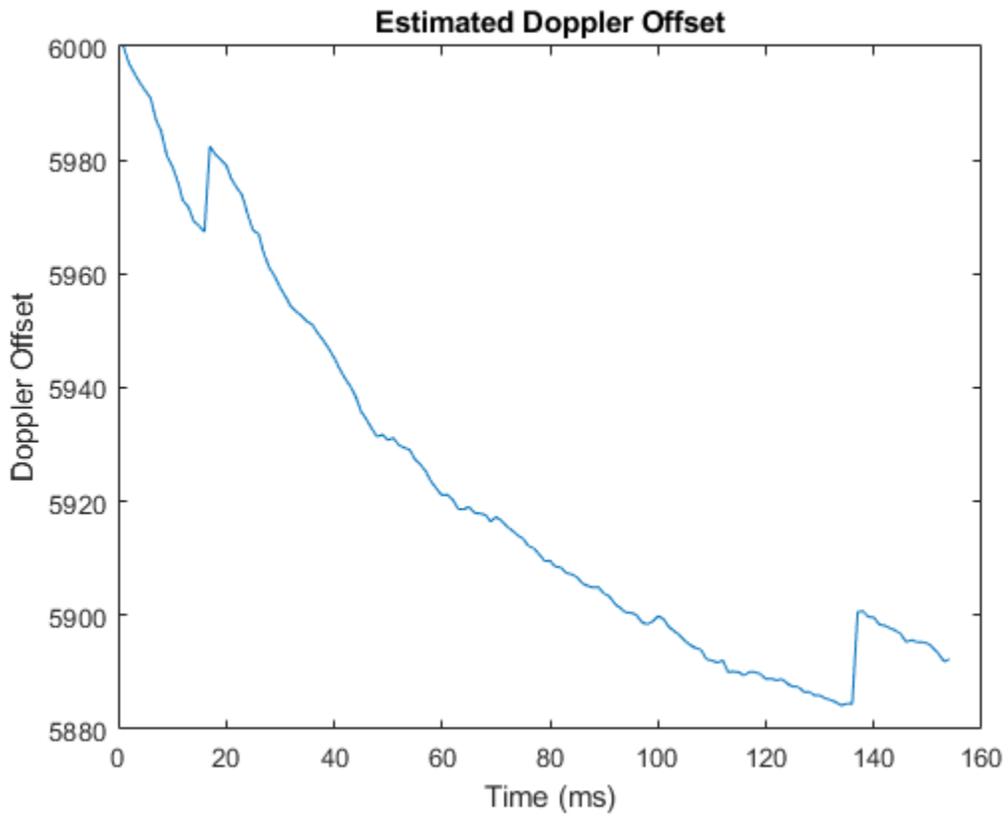
Build duration: 0h 4m 34.936s

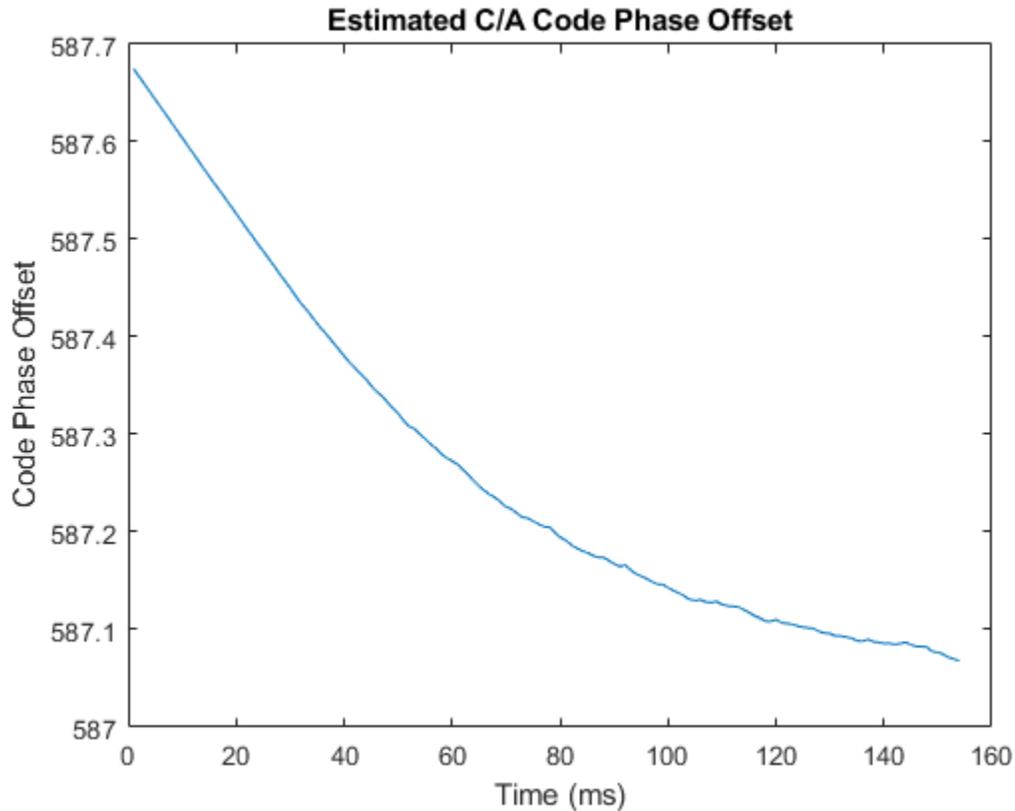
Warning: In rapid accelerator mode, when the simulation is started from the command line, visualization blocks are not updated. If the simulation is started from the toolstrip, visualization blocks are updated.

Input PRNID	Detected PRNID

11	11
18	18
22	22
23	23
Input code phase offset	Estimated code phase offset
300.34	300.19
312.88	312.72
587.21	587.07
425.89	425.72
Input doppler offset	Estimated doppler offset
3289	3253.8
1568	1534.7
5856	5892.2
7796	7836.1







Generate HDL Code

To generate the HDL code, you must have an HDL Coder™ license. Use `makehdl` and `makehdltb` functions to generate HDL code and a HDL test bench for the Acquisition and Tracking subsystem.

The resulting HDL code is synthesized for a Xilinx® Zynq®-7000 ZC706 evaluation board. This table shows the post place and route resource utilization. The maximum frequency of operation is 200 MHz.

Resources	Usage
Slice LUT	65695
Slice Registers	71376
RAMB36	176
RAMB18	52
DSP48	269

Appendix

This example uses these helper files:

- `HelperGPSAlmanac2Struct.m` — Convert text file of almanac to structure
- `HelperGPSCEIConfig.m` — Create configuration object for GPS navigation data

- HelperGPSNAVDDataEncode.m — Encode navigation data from configuration object into bits
- HelperGPSNavigationConfig.m — Create configuration object for GPS navigation data
- HelperGPSNum2BitsVector.m — Scale and convert the value to bits using left MSB

References

- [1]. Kaplan, Elliot D., and C. Hegarty, eds., *Understanding GPS/GNSS: Principles and Applications*. Third edition. GNSS Technology and Applications Series. Boston; London; Artech House, 2017.
- [2]. IS-GPS-200, Rev: L. "NAVSTAR GPS Space Segment/Navigation User Segment Interfaces." GPS Enterprise Space & Missile Systems Center (SMC) - LAAFB. <https://www.gps.gov/technical/icwg/IS-GPS-200L.pdf>
- [3]. Ward, P.W. "GPS Receiver Search Techniques." In *Proceedings of Position, Location and Navigation Symposium - PLANS '96*, 604 - 11. Atlanta, GA, USA: IEEE, 1996. <https://doi.org/10.1109/PLANS.1996.509134>.

DVB-S.2 System Simulation Using a GPU-Based LDPC Decoder System Object

This example shows how to use a GPU-based LDPC Decoder System object™ to increase the speed of a communications system simulation. The performance improvement is illustrated by modeling part of the ETSI (European Telecommunications Standards Institute) EN 302 307 standard for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S.2) [1 on page 6-62]. For further information on using System objects to simulate the DVB-S.2 system see “DVB-S.2 Link, Including LDPC Coding in Simulink”. You must have a Parallel Computing Toolbox™ user license to use the GPU-based LDPC Decoder.

Introduction

The LDPC Decoding algorithm is computationally expensive and constitutes the vast majority of the time spent in a DVB-S.2 simulation. Using the `comm.gpu.LDPCDecoder` System object to execute the decoding algorithm on a GPU dramatically improves simulation run time. The example simulates the DVB-S.2 system, obtaining a benchmark for speed (run time), once with a CPU-based LDPC decoder function (`ldpcDecode`) and once with a GPU-based LDPC Decoder (`comm.gpu.LDPCDecoder`). The example captures the bit error rate for both versions, to show there is no loss in decoding performance using the GPU.

```
fprintf(...
    'DVB-S.2 Digital Video Broadcast Standard Bit Error Rate Simulation\n\n');
```

```
DVB-S.2 Digital Video Broadcast Standard Bit Error Rate Simulation
```

```
fprintf(...
    'Performance comparison of CPU- and GPU- accelerated decoders.\n');
```

```
Performance comparison of CPU- and GPU- accelerated decoders.
```

GPU Presence Detection

The example attempts to query the GPU to detect a Parallel Computing Toolbox user license and the presence of a supported GPU. If the GPU or the Parallel Computing Toolbox is unavailable, a CPU-only simulation can be performed.

```
try
    % Query the GPU
    dev = parallel.gpu.GPUDevice.current;

    % Print out information about the GPU that was found
    fprintf(...
        'GPU detected (%s, %d multiprocessors, Compute Capability %s)\n',...
        dev.Name,dev.MultiprocessorCount,dev.ComputeCapability);

    % Include a GPU-based simulation.
    doGPU = true;

catch % #ok<CTCH>

    % The GPU is not supported or not present, or the Parallel Computing
    %Toolbox was not present and licensed. Consider a CPU-only simulation.
```

```

inp = input(['***NOTE: GPU not detected. ', ...
           'Continue with CPU-only simulation? [Y]/N '], 's');
if strcmpi(inp, 'y') || isempty(inp)
    doGPU = false;
else
    return;
end
end
end

```

GPU detected (Tesla V100-PCIE-32GB, 80 multiprocessors, Compute Capability 7.0)

Initialization

The `getParamsDVBS2Demo.m` function generates a structure, `dvb`, which holds the configuration information for the DVB-S.2 system given the parameters below. Subsequently, the example includes creating and configuring System objects, based on the `dvb` structure.

The `createSimObjDVBS2Demo.m` script constructs most of the System objects used in DVB-S.2 and configures them based on the `dvb` structure.

Then an LDPC decoder configuration object and a GPU-based LDPC Decoder System object are created. The LDPC decoder configuration object is passed to the CPU-based `ldpcDecode` function which uses options equivalent to those used by the GPU-based LDPC Decoder System object.

```

% DVB-S.2 System Parameters
subsystemType = 'QPSK 1/2'; % Constellation and LDPC code rate
EsNodB = 0.75; % Energy per symbol to noise PSD ratio in dB
numFrames = 10; % Number of frames to simulate
maxNumLDPCIterations = 50; % LDPC Decoder iterations

dvb = getParamsDVBS2Demo(subsystemType, EsNodB, maxNumLDPCIterations);

% Create and configure the BCH Encoder and Decoder, Modulator, Demodulator,
% AWGN Channel.

createSimObjDVBS2Demo;

% Construct an LDPC Encoder configuration object
encoderCfg = ldpcEncoderConfig(dvb.LDPCParityCheckMatrix);

% LDPC Decoder Configuration
ldpcPropertyValuePairs = { ...
    'MaximumIterationCount', dvb.LDPCNumIterations, ...
    'ParityCheckMatrix', dvb.LDPCParityCheckMatrix, ...
    'DecisionMethod', 'Hard Decision', ...
    'IterationTerminationCondition', 'Maximum iteration count', ...
    'OutputValue', 'Information part'};

% Construct an LDPC Decoder configuration object
decoderCfg = ldpcDecoderConfig(dvb.LDPCParityCheckMatrix);
if doGPU
    % Construct a GPU-based LDPC Decoder System object
    gpuLDPCDecoder = comm.gpu.LDPCDecoder(ldpcPropertyValuePairs{:});
end

% Create an ErrorRate object to analyze the differences in bit error rate
% between the CPU and GPU.

```

```
BER = comm.ErrorRate;
```

CPU and GPU Performance Comparison

This example simulates the DVB-S.2 system using the CPU-based LDPC Decoder function first, and then the GPU-based LDPC Decoder System object. The example obtains system benchmarks for each LDPC Decoder by passing several frames of data through the system and measuring the total system simulation time. The first frame of data incurs a large simulation initialization time, and so, it is excluded from the benchmark calculations. The per frame and average system simulation times are printed to the Command Window. The bit error rate (BER) of the system is also printed to the Command Window to illustrate that both CPU-based and GPU-based LDPC Decoders achieve the same BER.

```
if doGPU
    architectures = 2;
else
    architectures = 1;
end

% Initialize run time results vectors
runtime = zeros(architectures,numFrames);
avgtime = zeros(1,architectures);

% Seed the random number generator used for the channel and message
% creation. This will allow a fair BER comparison between CPU and GPU.
% Cache the original random stream to restore later.

original_rs = RandStream.getGlobalStream;
rs = RandStream.create('mrg32k3a','seed',25);
RandStream.setGlobalStream(rs);

% Loop for each processing unit - CPU and GPU
for ii = 1:architectures

    % Do some initial setup for the execution loop
    if (ii == 1)
        arch = 'CPU'; % Use CPU LDPC Decoder
    else
        arch = 'GPU';
        decoder = gpuLDPCDecoder;% Use GPU LDPC Decoder
    end

    % Reset the Error Rate object
    reset(BER);

    % Reset the random stream
    reset(rs);

    % Notice to the user that DVB-S.2 simulation is beginning.
    fprintf(['\nUsing ' arch '-based LDPC Decoder:\n']);
    dels = repmat('\b',1,fprintf(' Initializing ...'));

    % Main simulation loop. Run numFrames+1 times and ignore the first
    % frame (which has initialization overhead) for the run time
    % calculation. Use the first run for the BER calculation.
```

```

for rr = 1:(numFrames+1)

    % Start timer
    ts = tic;

    % ***Create an input Message*** %
    msg = zeros(encbch.MessageLength, 1);
    msg(1:dvb.NumInfoBitsPerCodeword) = ...
        logical(randi([0 1],dvb.NumInfoBitsPerCodeword,1));

    % ***Transmit*** %
    bchencOut = encbch(msg);
    ldpcencOut = ldpcEncode(bchencOut,encoderCfg);
    xlvOut = intrlv(ldpcencOut,dvb.InterleaveOrder);
    modOut = pskModulator(xlvOut);

    % ***Corrupt with noise*** %
    chanOut = chan(modOut);

    % ***Receive*** %y
    demodOut = pskDemodulator(chanOut);
    dexlvOut = deintrlv(demodOut,dvb.InterleaveOrder);

    % Use the appropriate LDPC Decoder.
    if strcmp(arch,'CPU')
        ldpcdecOut = logical(ldpcDecode(dexlvOut,decoderCfg,dvb.LDPCNumIterations,'Decision'))
    else
        ldpcdecOut = decoder(dexlvOut);
    end

    bchdecOut = decbch(ldpcdecOut);

    % ***Compute BER *** % Calculate BER at output of LDPC, not BCH.
    ber = BER(logical(bchencOut),ldpcdecOut);

    % Stop timer
    runtime(ii, rr) = toc(ts);

    % Don't report the first frame with the initialization overhead.
    if (rr > 1)
        fprintf(dels);
        newCharsToDelete = fprintf(' Frame %d decode : %.2f sec', ...
            rr-1, runtime(ii,rr));
        dels = repmat('\b',1,newCharsToDelete);
    end
end % end of running a frame through the DVB-S.2 system.

% Report the run time results to the Command Window.
fprintf(dels); % Delete the last line printed out.

% Calculate the average run time. Don't include frame 1 because it
% includes some System object initialization time.
avgtime(ii) = mean(runtime(ii,2:end));

fprintf(' %d frames decoded, %.2f sec/frame\n',numFrames,avgtime(ii));
fprintf(' Bit error rate: %g \n',ber(1) );

```

```
end % architecture loop
```

```
Using CPU-based LDPC Decoder:
```

```
  Initializing ...
```

```
  Frame 1 decode : 0.29 sec  Frame 2 decode : 0.30 sec  Frame 3 decode : 0.32 sec  Frame 4 decode
```

```
  10 frames decoded, 0.28 sec/frame
```

```
  Bit error rate: 0.00785634
```

```
Using GPU-based LDPC Decoder:
```

```
  Initializing ...
```

```
  Frame 1 decode : 0.12 sec  Frame 2 decode : 0.12 sec  Frame 3 decode : 0.12 sec  Frame 4 decode
```

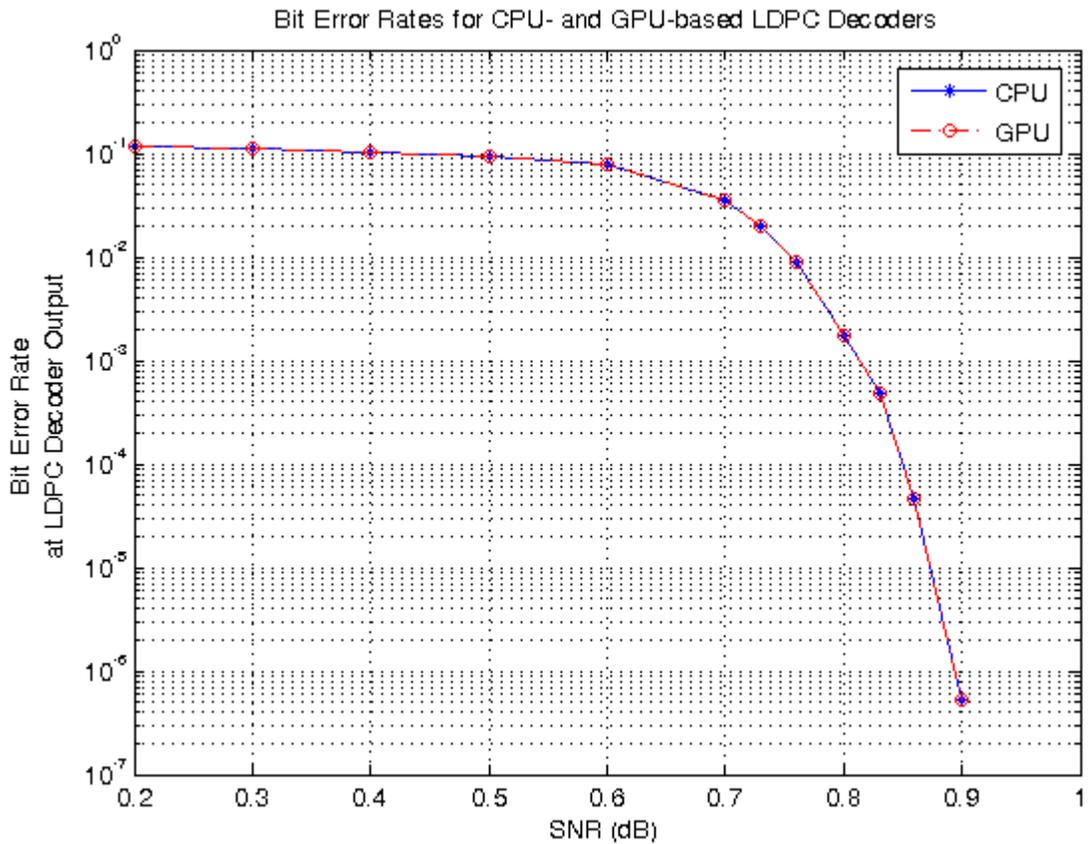
```
  10 frames decoded, 0.11 sec/frame
```

```
  Bit error rate: 0.00785634
```

```
% Reset the random stream to the cached object
```

```
RandStream.setGlobalStream(original_rs);
```

Using code similar to what is shown above, a bit error rate measurement was made offline. The bit error rate performance of the GPU- and CPU-based LDPC Decoders are identical as seen in this plot.



Summary

If a GPU was used, show the speedup based on the average run time of a DVB-S.2 system using a GPU LDPC Decoder vs a CPU LDPC Decoder.

```
if ~doGPU
    fprintf('\n*** GPU not present ***\n\n');
else
    %Calculate system-wide speedup
    fprintf(['\nFull system simulation runs %.2f times faster using ' ...
            'the GPU-based LDPC Decoder.\n\n'],avgtime(1) / avgtime(2));
end
```

Full system simulation runs 2.60 times faster using the GPU-based LDPC Decoder.

Appendix

This example uses the createSimObjDVBS2Demo.m script and getParamsDVBS2Demo.m helper function.

Selected Bibliography

- 1 ETSI Standard EN 302 307 V1.1.1: Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, New Gathering and other broadband satellite applications (DVB-S.2), European Telecommunications Standards Institute, Valbonne, France, 2005-03.