DSSS Receiver for Search and Rescue Tracking System

Since R2024a

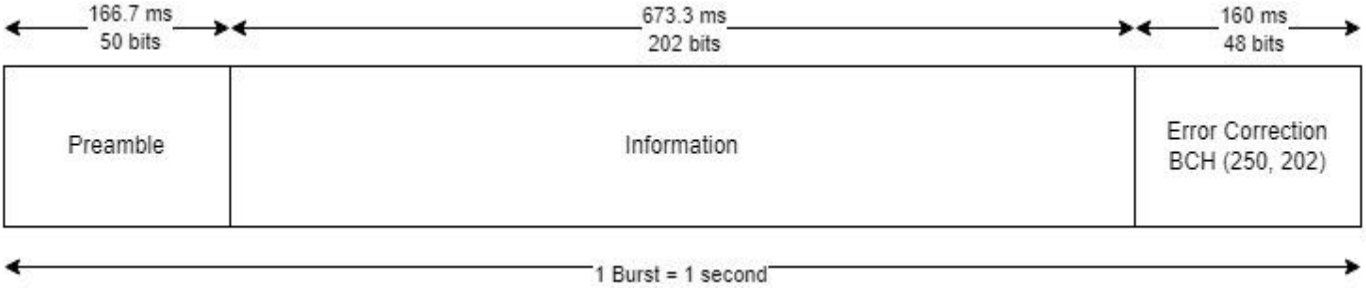
This example shows how to implement a direct sequence spread spectrum (DSSS) receiver to track a rescue beacon. In this example, the system generates an OQPSK-modulated DSSS burst that mimics the signal of an emergency locator transmitter (ELT) defined in the second-generation COSPAS-SARSAT (Search And Rescue Satellite-Aided Tracking) specification [1]. The system passes the burst through a channel and simulates a receiver that detects, synchronizes, despreads, and decodes the burst.

With this example, you learn how to use MATLAB System objects and functions to accelerate the system design and implementation by:

- Calculating error correction bits
- Generating the scrambling sequence
- Modulating with OQPSK
- Impairing the channel with frequency and phase offset
- Implementing automatic gain control
- Detecting the preamble
- Estimating coarse and fine frequency offset
- Recovering symbol timing
- Performing error detection and correction

COSPAS-SARSAT

The COSPAS-SARSAT system is a second-generation satellite system for ELTs that serve as distress beacons in a life-threatening situation to summon help from government authorities [2]. The transmission modulation uses DSSS to spread data bits modulated with OQPSK. DSSS adds signal gain in the form of a spreading gain and is ideal for long-distance transmissions. OQPSK is a modulation that resists amplitude variations. Its constant-envelope and continuous-phase signal properties are ideal for efficient amplification using power amplifiers. A one-second packet burst contains the following information:



- **Preamble** (50 bits, 166.7 ms): All preamble bits are zero.
- **Information Message** (202 bits, 673.3 ms): The information contains location data, identification codes, and receiver status.
- **Error Correction (Parity) (**48 bits, 160 ms): The 202 information bits are encoded using an error correction code.

The burst consists of a preamble, followed by the information payload, and ending with parity bits for error detection and correction of the information payload. The preamble assists the receiver in establishing synchronization quickly with high probability of detection and low probability of false alarm. To transmit the emergency signal as long as possible, the system preserves battery life by transmitting the burst once every 30 seconds with a random back-off time to prevent another emergency beacon from accidentally transmitting at the same exact time as another beacon. The transmission is treated as a random-access burst. Consequently, the signal start time, frequency offset, phase offset, timing offset, signal strength, and channel are all unknown at the receiver.

The total size of the packet is 300 bits. The system spreads each bit using a spreading factor of 256, with the in-phase (I) and quadrature (Q) branches of the OQPSK modulation spread using unique orthogonal spreading codes. Since the modulation order of OQPSK is two bits per symbol, the system sends two bits for every 256 OQPSK symbols. With a specified chip rate of 38400 chips per second, the packet duration is one second (150 bits transmitted per I and Q branch with each bit spread over 256 OQPSK symbols).

Example Overview

This example models an ELT transmitter and ELT receiver. The transmitter generates one burst after a user-selectable number of seconds. The receiver can travel in an aircraft or satellite. This example can generate impairments such as:

- Large-scale fading from path loss
- Additive White Gaussian Noise (AWGN) from thermal noise
- Frequency and phase offset from transmitter-receiver oscillator mismatch and Doppler effects

Settings

Set the simulation and receiver parameters. Distance affects received signal power due to path loss. Velocity introduces Doppler effects such as from receivers in a search aircraft traveling at 200 meters per second, or receivers in low earth orbit (LEO) traveling at 7800 meters per second that can add up to approximately 10 kHz of frequency offset.

```
settings.distance       =          300000; % distance between transmitter and receiver in
meters

settings.velocity       =          -7000; % relative velocity between transmitter and
receiver in m/s

settings.oversampling   =          4; % oversampling factor for tx and rx (must be >= 2)

settings.txPower        =          33; % transmit power in dBm (33 dBm max)

settings.simTime        =          5; % length of simulation in seconds

settings.burstDelay     =          3.5; % number of seconds until burst is received

settings.maxDoppler     =          12000; % maximum detected Doppler in Hz
```

Impairments

The COSPAS-SARSAT transmission frequency is 406.05 MHz, with a tolerance of +/- 1200 Hz or approximately 3 ppm. Specify the frequency offset between the transmitter and receiver due to oscillator mismatch.

```
impairments.rxFreqOffset =          1200; % receiver frequency offset Hz
```

Set the parameters for the transmission protocol. The parameters for the COSPAS-SARSAT system are detailed in [1].

```
system.fc              = 406.05;   % carrier frequency in MHz
system.preambleLength  = 50;       % preamble length in bits
system.payloadLength   = 202;      % payload length in bits
system.parityLength    = 48;       % parity length in bits
system.chipRate        = 38400;    % chips per second
system.spreadingFactor = 256;      % DSSS spreading factor
```

Compute parameters derived from the settings.

```
sps = 2 * settings.oversampling; % define samples per symbol
fs = system.chipRate * sps; % define the sampling frequency

spreadingFactor_IQ = system.spreadingFactor / 2;

packetLength = system.preambleLength + ...
               system.payloadLength + ...
               system.parityLength;    % packet length in bits
numChips = spreadingFactor_IQ * packetLength; % number of chips in the packet
numPreambleChips = system.preambleLength * spreadingFactor_IQ;
numBurstSamples = (packetLength/2) * system.spreadingFactor * sps;

dopplerShift = (system.fc*1e6) * settings.velocity / physconst('LightSpeed');

checkSettings(settings); % check restrictions on settings

rng(1973); % set a random seed for repeatability
```

DSSS Transmitter

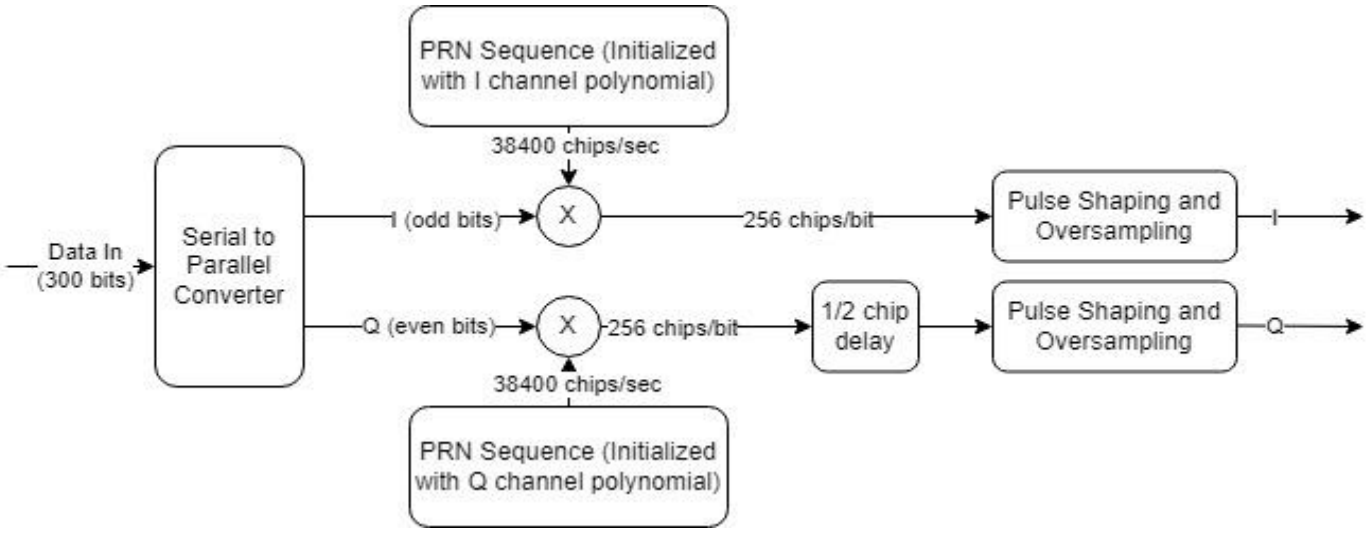
Direct sequence spread spectrum communications is a method for encoding a stream of bits with a long sequence of bits that have the property of zero autocorrelation. This sequence of bits is known as a pseudo-random noise (PRN) sequence. This method spreads the incoming bits such that the spectrum bandwidth increases, and the power spectrum of the signal resembles bandlimited white noise due to the zero-autocorrelation property. This characteristic makes DSSS resistant to narrowband interference and jamming. The spreading also adds information to the signal that acts as a spreading gain that depends on the spreading factor, or the number of spreading bits per input bit.

As a generic example of DSSS spreading, consider the first 24 bits of the PRN sequence below. Choose a spreading factor of 4, such that every encoded bit is spread with four spreading bits. Encode six bits of information such that a logical 1 inverts the PRN sequence for four bits and uses those as the spreading bits, while a logical 0 preserves the PRN sequence for the four bits (the mapped sequence below, where underlined bits denote the inverse operation).

- `PRN sequence: 1000 0010 0001 1000 1010 0111`
- `Information bits: 1 0 0 1 1 0`
- `Mapped sequence: 1000 0010 0001 1000 1010 0111`

- DSSS output bits: 0111 0010 0001 0111 0101 0111

The DSSS transmitter takes an incoming serial bitstream and splits it into two streams: the in-phase (I) stream takes the odd bits, and the quadrature (Q) stream takes the even bits. Two PRN sequence generators output 38400 bits for each stream to use as the spreading sequence. The spreading codes are based on an $x_{23}+x_{18}+1$ generator polynomial. The I and Q streams use a unique sequence by initializing the PRN generator with predefined polynomials. The Q stream is offset by one half of a chip to produce the offset QPSK constellation.



Generate the PRN sequences using `comm.PNSequence`. The transmitter and receiver use this sequence to spread and despread the data, respectively.

```
% Configure and initialize the PRN generators for the I and Q streams
DSSS.normalI = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]; % COSPAS-SARSAT Normal I mode
DSSS.normalQ = [0 0 1 1 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0]; % COSPAS-SARSAT Normal Q mode
DSSS.hLFSRI = comm.PNSequence( ...
    Polynomial =        'X23+X18+1',  ...
    InitialConditions = DSSS.normalI, ...
    SamplesPerFrame =   numChips);

DSSS.hLFSRQ = comm.PNSequence( ...
    Polynomial =        'X23+X18+1',  ...
    InitialConditions = DSSS.normalQ, ...
    SamplesPerFrame =   numChips);

% Generate the PRN sequences
DSSS.PRN_I = DSSS.hLFSRI();
DSSS.PRN_Q = DSSS.hLFSRQ();
```

The COSPAS-SARSAT specification lists the first 64 chips of the normal I and Q sequences to verify correct implementation. Use the `dec2hex` function to output the hexadecimal values for verification against the standard.

```
dec2hex(bit2int(DSSS.PRN_I(1:64), 16))

ans = 4×4 char array
    '8000'
    '0108'
    '4212'
    '84A1'


dec2hex(bit2int(DSSS.PRN_Q(1:64), 16))

ans = 4×4 char array
```