```
%function [BER_1]=QPSKTXRXSim(EbNo)
clc
clear

load commqpsktxrx_sbits_100.mat; % length 174
% General simulation parameters
SimParams.M = 4; % M-PSK alphabet size
SimParams.Upsampling = 4; % Upsampling factor
SimParams.Downsampling = 2; % Downsampling factor
SimParams.Fs = 2e5; % Sample rate in Hertz
SimParams.Ts = 1/SimParams.Fs; % Sample time in sec
SimParams.FrameSize = 100; % Number of modulated
symbols per frame

% hAlamoutiEnc = comm.OSTBCEncoder;
% hAlamoutiDec = comm.OSTBCCombiner;
Hray = zeros(100, 2, 2);
% Create the Rayleigh distributed channel response
matrix
 %   for two transmit and two receive antennas
Hray(1:2:end, :, :) = (randn(50, 2, 2) + 1i*randn(50,
2, 2))/sqrt(2);
 %   assume held constant for 2 symbol periods
Hray(2:2:end, :, :) = Hray(1:2:end, :, :);

SimParams.H21 = Hray(:,:,1)/sqrt(2); %

% Tx parameters
SimParams.BarkerLength = 26; % Number of Barker code
symbols
SimParams.DataLength = (SimParams.FrameSize -
SimParams.BarkerLength)*2; % Number of data payload
bits per frame
SimParams.ScramblerBase = 2;
SimParams.ScramblerPolynomial = [1 1 1 0 1];
SimParams.ScramblerInitialConditions = [0 0 0 0];

SimParams.sBit = sBit; % Payload bits
SimParams.RxBufferedFrames = 10; % Received buffer
length (in frames)

SimParams.RaisedCosineFilterSpan = 10; % Filter span of
Raised Cosine Tx Rx filters (in symbols)
SimParams.MessageLength = 112;
```

```
SimParams.FrameCount = 100; % Number of frames
transmitted

% Channel parameters
SimParams.PhaseOffset = 0; % in degrees
SimParams.EbNo = 10; % in dB
SimParams.FrequencyOffset = 0; % Frequency offset
introduced by channel impairments in Hertz
SimParams.DelayType = 'Triangle'; % select the type of
delay for channel distortion

% Rx parameters
SimParams.CoarseCompFrequencyResolution = 25; %
Frequency resolution for coarse frequency compensation

% Look into model for details for details of PLL
parameter choice. Refer equation 7.30 of "Digital
Communications - A Discrete-Time Approach" by Michael
Rice.
K = 1;
A = 1/sqrt(2);
SimParams.PhaseRecoveryLoopBandwidth = 0.01; %
Normalized loop bandwidth for fine frequency
compensation
SimParams.PhaseRecoveryDampingFactor = 1; % Damping
Factor for fine frequency compensation
SimParams.TimingRecoveryLoopBandwidth = 0.01; %
Normalized loop bandwidth for timing recovery
SimParams.TimingRecoveryDampingFactor = 1; % Damping
Factor for timing recovery
SimParams.TimingErrorDetectorGain =
2.7*2*K*A^2+2.7*2*K*A^2; % K_p for Timing Recovery PLL,
determined by 2KA^2*2.7 (for binary PAM), QPSK could be
treated as two individual binary PAM, 2.7 is for raised
cosine filter with roll-off factor 0.5

%QPSK modulated Barker code header
BarkerCode = [+1; +1; +1; +1; +1; -1; -1; +1; +1; -1;
+1; -1; +1; +1; +1; +1; +1; +1; -1; -1; +1; +1; -1; +1;
-1; +1]; % Bipolar Barker Code
SimParams.ModulatedHeader = sqrt(2)/2 * (-1-1i) *
BarkerCode;

% Generate square root raised cosine filter
```

```
coefficients (required only for MATLAB example)
SimParams.Rolloff = 0.5;

% Square root raised cosine transmit filter
SimParams.TransmitterFilterCoefficients = ...
  rcosdesign(SimParams.Rolloff,
SimParams.RaisedCosineFilterSpan, ...
  SimParams.Upsampling);

% Square root raised cosine receive filter
SimParams.ReceiverFilterCoefficients = ...
  rcosdesign(SimParams.Rolloff,
SimParams.RaisedCosineFilterSpan, ...
  SimParams.Upsampling);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
prmQPSKTxRx = SimParams; % QPSK system parameters
printData = true; %true if the received data is to be
printed
useScopes = true; % true if scopes are to be used

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

    % Initialize the components
    % Create and configure the transmitter System object
    hTx = QPSKTransmitterR(...
        'UpsamplingFactor', prmQPSKTxRx.Upsampling, ...
        'MessageLength', prmQPSKTxRx.MessageLength, ...

'TransmitterFilterCoefficients',prmQPSKTxRx.Transmitter
FilterCoefficients, ...
        'DataLength', prmQPSKTxRx.DataLength, ...
        'ScramblerBase', prmQPSKTxRx.ScramblerBase, ...
        'ScramblerPolynomial',
prmQPSKTxRx.ScramblerPolynomial, ...
        'ScramblerInitialConditions',
prmQPSKTxRx.ScramblerInitialConditions,...
        'H21',prmQPSKTxRx.H21);

    % Create and configure the AWGN channel System
object
    hChan = QPSKChannelR('DelayType',
```

```
prmQPSKTxRx.DelayType, ...
       'RaisedCosineFilterSpan',
prmQPSKTxRx.RaisedCosineFilterSpan, ...
       'PhaseOffset', prmQPSKTxRx.PhaseOffset, ...
       'SignalPower', 1/prmQPSKTxRx.Upsampling, ...
       'FrameSize', prmQPSKTxRx.FrameSize, ...
       'UpsamplingFactor', prmQPSKTxRx.Upsampling, ...
       'EbNo', prmQPSKTxRx.EbNo, ...
       'BitsPerSymbol',
prmQPSKTxRx.Upsampling/prmQPSKTxRx.Downsampling, ...
       'FrequencyOffset',
prmQPSKTxRx.FrequencyOffset, ...
       'SampleRate', prmQPSKTxRx.Fs);

   % Create and configure the receiver System object
   hRx = QPSKReceiverR('DesiredAmplitude',
1/sqrt(prmQPSKTxRx.Upsampling), ...
       'ModulationOrder', prmQPSKTxRx.M, ...
       'DownsamplingFactor',
prmQPSKTxRx.Downsampling, ...
       'CoarseCompFrequencyResolution',
prmQPSKTxRx.CoarseCompFrequencyResolution, ...
       'PhaseRecoveryDampingFactor',
prmQPSKTxRx.PhaseRecoveryDampingFactor, ...
       'PhaseRecoveryLoopBandwidth',
prmQPSKTxRx.PhaseRecoveryLoopBandwidth, ...
       'TimingRecoveryDampingFactor',
prmQPSKTxRx.TimingRecoveryDampingFactor, ...
       'TimingRecoveryLoopBandwidth',
prmQPSKTxRx.TimingRecoveryLoopBandwidth, ...
       'TimingErrorDetectorGain',
prmQPSKTxRx.TimingErrorDetectorGain, ...
       'PostFilterOversampling',
prmQPSKTxRx.Upsampling/prmQPSKTxRx.Downsampling, ...
       'FrameSize', prmQPSKTxRx.FrameSize, ...
       'BarkerLength', prmQPSKTxRx.BarkerLength, ...
       'MessageLength', prmQPSKTxRx.MessageLength, ...
       'SampleRate', prmQPSKTxRx.Fs, ...
       'DataLength', prmQPSKTxRx.DataLength, ...
       'ReceiverFilterCoefficients',
prmQPSKTxRx.ReceiverFilterCoefficients, ...
       'DescramblerBase',
prmQPSKTxRx.ScramblerBase, ...
       'DescramblerPolynomial',
```

```
prmQPSKTxRx.ScramblerPolynomial, ...
       'DescramblerInitialConditions',
prmQPSKTxRx.ScramblerInitialConditions,...
       'PrintOption', printData,...
       'H21',prmQPSKTxRx.H21);

    if useScopes
       % Create the System object for plotting all the
scopes
       hScopes = QPSKScopes;
    end

hRx.PrintOption = printData;

for count = 1:prmQPSKTxRx.FrameCount
    [transmittedSignal] = step(hTx); % Transmitter
    corruptSignal = step(hChan, transmittedSignal, 0); %
AWGN Channel
    [RCRxSignal,coarseCompBuffer, timingRecBuffer,BER] =
step(hRx,corruptSignal); % Receiver
%     figure(1)
%     plot(real(transmittedSignal))
%     drawnow

%     figure(2)
%     plot(real(modulatedData))
%     drawnow
%
%     figure(3)
%     plot(transmittedData)
%     drawnow
%
%    pause(1)
%     if useScopes
%
stepQPSKScopes(hScopes,RCRxSignal,coarseCompBuffer,
timingRecBuffer); % Plots all the scopes
%     end
end
% if isempty(coder.target)
%     release(hTx);
%     release(hChan);
%     release(hRx);
% end
```

```
% if useScopes
%     releaseQPSKScopes(hScopes);
% end
BER_1=BER(1);
fprintf('Error rate = %f.\n',BER(1));
fprintf('Number of detected errors = %d.\n',BER(2));
fprintf('Total number of compared samples
= %d.\n',BER(3));
```

```
classdef QPSKTransmitterR < matlab.System
%#codegen
% Generates the QPSK signal to be transmitted

%   Copyright 2012 The MathWorks, Inc.

    properties (Nontunable)
        UpsamplingFactor = 4;
        MessageLength = 105;
        DataLength = 174;
        TransmitterFilterCoefficients = 1;
        ScramblerBase = 2;
        ScramblerPolynomial = [1 1 1 0 1];
        ScramblerInitialConditions = [0 0 0 0];
        H21=zeros(100,2);
    end

     properties (Access=private)
        pBitGenerator
        pQPSKModulator
        pTransmitterFilter
    end

    methods
        function obj = QPSKTransmitterR(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access=protected)
        function setupImpl(obj)
            obj.pBitGenerator = QPSKBitsGeneratorR(...
                'MessageLength', obj.MessageLength, ...
                'BernoulliLength', obj.DataLength-
obj.MessageLength, ...
                'ScramblerBase', obj.ScramblerBase, ...
                'ScramblerPolynomial',
obj.ScramblerPolynomial, ...
                'ScramblerInitialConditions',
obj.ScramblerInitialConditions);
           obj.pQPSKModulator  =
comm.QPSKModulator('BitInput',true, ...
                'PhaseOffset', pi/4);
```

```
            obj.pTransmitterFilter =
dsp.FIRInterpolator(obj.UpsamplingFactor, ...
                obj.TransmitterFilterCoefficients);
        end

        function
[transmittedSignal,transmittedData,modulatedData]=
stepImpl(obj)
            % Generates the data to be transmitted
            [transmittedData, ~] =
step(obj.pBitGenerator);

            % Modulates the bits into QPSK symbols
            modulatedData = step(obj.pQPSKModulator,
transmittedData);

            hAlamoutiEnc = comm.OSTBCEncoder;

            % Alamouti Space-Time Block Encoder
            encData = step(hAlamoutiEnc, modulatedData);


            chanOut21 = sum(obj.H21.* encData, 2);

            transmittedSignal=chanOut21;
            % Square root Raised Cosine Transmit Filter
            %transmittedSignal =
step(obj.pTransmitterFilter, chanOut21);
        end

        function resetImpl(obj)
            reset(obj.pBitGenerator);
            reset(obj.pQPSKModulator );
            reset(obj.pTransmitterFilter);
        end

        function releaseImpl(obj)
            release(obj.pBitGenerator);
            release(obj.pQPSKModulator );
            release(obj.pTransmitterFilter);
        end

        function N = getNumInputsImpl(~)
            N = 0;
```

```
        end
    end
end
```

```
classdef QPSKReceiverR < matlab.System

% Copyright 2012-2015 The MathWorks, Inc.

    properties (Nontunable)
        DesiredAmplitude = 1/sqrt(2);
        ModulationOrder = 4;
        DownsamplingFactor = 2;
        CoarseCompFrequencyResolution = 50;
        PhaseRecoveryLoopBandwidth = 0.01;
        PhaseRecoveryDampingFactor = 1;
        TimingRecoveryDampingFactor = 1;
        TimingRecoveryLoopBandwidth = 0.01;
        TimingErrorDetectorGain = 5.4;
        PostFilterOversampling = 2;
        FrameSize = 100;
        BarkerLength = 26;
        MessageLength = 105;
        SampleRate = 200000;
        DataLength = 148;
        ReceiverFilterCoefficients = 1;
        DescramblerBase = 2;
        DescramblerPolynomial = [1 1 1 0 1];
        DescramblerInitialConditions = [0 0 0 0];
        PrintOption = false;
        H21=zeros(100,2);
    end

    properties (Access = private)
        pAGC
        pRxFilter
        pCoarseFreqEstimator
        pCoarseFreqCompensator
        pFineFreqCompensator
        pTimingRec
        pFrameSync
        pDataDecod
        pBER
     end

    properties (Access = private, Constant)
        pUpdatePeriod = 4 % Defines the size of vector
that will be processed in AGC system object
        pBarkerCode = [+1; +1; +1; +1; +1; -1; -1; +1;
```

```
+1; -1; +1; -1; +1 ; +1; +1; +1; +1; +1; -1; -1; +1;
+1; -1; +1; -1; +1]; % Bipolar Barker Code
      pModulatedHeader = sqrt(2)/2 * (-1-1i) *
QPSKReceiverR.pBarkerCode;
   end

   methods
      function obj = QPSKReceiverR(varargin)
         setProperties(obj,nargin,varargin{:});
      end
   end

   methods (Access = protected)
      function setupImpl(obj, ~)
         obj.pAGC = comm.AGC;

         obj.pRxFilter = dsp.FIRDecimator( ...
            'Numerator',
obj.ReceiverFilterCoefficients, ...
            'DecimationFactor',
obj.DownsamplingFactor);

         obj.pCoarseFreqEstimator =
comm.PSKCoarseFrequencyEstimator( ...
            'ModulationOrder',
obj.ModulationOrder, ...
            'Algorithm',         'FFT-based', ...
            'FrequencyResolution',
obj.CoarseCompFrequencyResolution, ...
            'SampleRate',        obj.SampleRate);

         obj.pCoarseFreqCompensator =
comm.PhaseFrequencyOffset( ...
            'PhaseOffset',         0, ...
            'FrequencyOffsetSource', 'Input port', ...
            'SampleRate',          obj.SampleRate);

         obj.pFineFreqCompensator =
comm.CarrierSynchronizer( ...
            'Modulation',             'QPSK', ...
            'ModulationPhaseOffset',   'Auto', ...
            'SamplesPerSymbol',
obj.PostFilterOversampling, ...
            'DampingFactor',
```

```
obj.PhaseRecoveryDampingFactor, ...
                'NormalizedLoopBandwidth',
obj.PhaseRecoveryLoopBandwidth);

            obj.pTimingRec = comm.SymbolSynchronizer( ...
                'TimingErrorDetector',    'Zero-Crossing
(decision-directed)', ...
                'SamplesPerSymbol',
obj.PostFilterOversampling, ...
                'DampingFactor',
obj.TimingRecoveryDampingFactor, ...
                'NormalizedLoopBandwidth',
obj.TimingRecoveryLoopBandwidth, ...
                'DetectorGain',
obj.TimingErrorDetectorGain);

            obj.pFrameSync = FrameFormation( ...
                'OutputFrameLength',
obj.FrameSize, ...
                'PerformSynchronization', true, ...
                'FrameHeader',
obj.pModulatedHeader);

            obj.pDataDecod =
QPSKDataDecoderR('FrameSize', obj.FrameSize, ...
                'BarkerLength', obj.BarkerLength, ...
                'ModulationOrder',
obj.ModulationOrder, ...
                'DataLength', obj.DataLength, ...
                'MessageLength', obj.MessageLength, ...
                'DescramblerBase',
obj.DescramblerBase, ...
                'DescramblerPolynomial',
obj.DescramblerPolynomial, ...
                'DescramblerInitialConditions',
obj.DescramblerInitialConditions, ...
                'PrintOption', obj.PrintOption,...
                'H21',obj.H21);
        end

        function [RCRxSignal, fineCompSignal,
timingRecBuffer,BER] = stepImpl(obj, bufferSignal)
            % AGC control
            RCRxSignal=0;
```

```
            fineCompSignal=0;
            timingRecBuffer=0;
%             AGCSignal =
obj.DesiredAmplitude*step(obj.pAGC, bufferSignal);
%
%             % Pass the signal through Square-Root
Raised Cosine Received Filter
%             RCRxSignal = step(obj.pRxFilter,
AGCSignal);
%
%             % Coarse frequency offset estimation
%             freqOffsetEst =
step(obj.pCoarseFreqEstimator, RCRxSignal);
%
%             % Coarse frequency compensation
%             coarseCompSignal =
step(obj.pCoarseFreqCompensator, RCRxSignal, -
freqOffsetEst);
%
%             % Fine frequency compensation
%             fineCompSignal =
step(obj.pFineFreqCompensator, coarseCompSignal);
%
%             % Symbol timing recovery
%             [timingRecSignal, timingRecBuffer] =
step(obj.pTimingRec, fineCompSignal);

%             % Frame synchronization
%             [symFrame, isFrameValid] =
step(obj.pFrameSync, timingRecSignal);
               isFrameValid=1;
               symFrame=bufferSignal;
            if isFrameValid % Decode frame of symbols
                obj.pBER = step(obj.pDataDecod, symFrame);
            end

            BER = obj.pBER;
        end

        function resetImpl(obj)
            obj.pBER = zeros(3, 1);
            reset(obj.pAGC);
            reset(obj.pRxFilter);
            reset(obj.pCoarseFreqEstimator);
```

```
            reset(obj.pCoarseFreqCompensator);
            reset(obj.pFineFreqCompensator);
            reset(obj.pTimingRec);
            reset(obj.pFrameSync);
            reset(obj.pDataDecod);
        end

        function releaseImpl(obj)
            release(obj.pAGC);
            release(obj.pRxFilter);
            release(obj.pCoarseFreqEstimator);
            release(obj.pCoarseFreqCompensator);
            release(obj.pFineFreqCompensator);
            release(obj.pTimingRec);
            release(obj.pFrameSync);
            release(obj.pDataDecod);
        end

        function N = getNumOutputsImpl(~)
            N = 4;
        end
    end
end
```

```
classdef QPSKDataDecoderR < matlab.System

% Copyright 2012-2014 The MathWorks, Inc.

    properties (Nontunable)
        FrameSize = 100;
        BarkerLength = 13;
        ModulationOrder = 4;
        DataLength = 174;
        MessageLength = 105;
        DescramblerBase = 2;
        DescramblerPolynomial = [1 1 1 0 1];
        DescramblerInitialConditions = [0 0 0 0];
        PrintOption = false;
        H21=zeros(100,2);
    end

    properties (Access = private)
        pCorrelator
        pQPSKDemodulator
        pDescrambler
        pBitGenerator
        pErrorRateCalc
    end

    properties (Constant, Access = private)
        pBarkerCode = [+1; +1; +1; +1; +1; -1; -1; +1;
+1; -1; +1; -1; +1; +1; +1; +1; +1; +1; -1; -1; +1; +1;
-1; +1; -1; +1]; % Bipolar Barker Code
        pModulatedHeader = sqrt(2)/2 * (-1-1i) *
QPSKDataDecoderR.pBarkerCode;

    end

    methods
        function obj = QPSKDataDecoderR(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access = protected)
        function setupImpl(obj, ~)
            obj.pCorrelator = dsp.Crosscorrelator;
```

```
            obj.pQPSKDemodulator =
comm.QPSKDemodulator('PhaseOffset',pi/4, ...
                'BitOutput', true);

            obj.pDescrambler =
comm.Descrambler(obj.DescramblerBase, ...
                obj.DescramblerPolynomial,
obj.DescramblerInitialConditions);

            obj.pBitGenerator =
QPSKBitsGeneratorR('MessageLength',
obj.MessageLength, ...
                'BernoulliLength', obj.DataLength-
obj.MessageLength, ...
                'ScramblerBase', obj.DescramblerBase, ...
                'ScramblerPolynomial',
obj.DescramblerPolynomial, ...
                'ScramblerInitialConditions',
obj.DescramblerInitialConditions);

            obj.pErrorRateCalc = comm.ErrorRate;
        end

        function  BER = stepImpl(obj, data)
            % Phase offset estimation
            %phaseEst =
round(angle(mean(conj(obj.pModulatedHeader) .*
data(1:obj.BarkerLength)))*2/pi)/2*pi;

            % Compensating for the phase offset
            %phShiftedData = data .* exp(-1i*phaseEst);

            hAlamoutiDec = comm.OSTBCCombiner;

            % Alamouti Space-Time Block Combiner
            decData = step(hAlamoutiDec, data, obj.H21);

            % Demodulating the phase recovered data
            demodOut = step(obj.pQPSKDemodulator,
decData);

            % Performs descrambling
            deScrData = step(obj.pDescrambler, ...
                demodOut( ...
```

```
obj.BarkerLength*log2(obj.ModulationOrder)+1 : ...
             obj.FrameSize*log2(obj.ModulationOrder)));

          % Recovering the message from the data
          Received = deScrData(1:obj.MessageLength);
          bits2ASCII(obj, Received);

          [~, transmittedMessage] =
step(obj.pBitGenerator);

          BER = step(obj.pErrorRateCalc,
transmittedMessage, Received);
       end

       function resetImpl(obj)
          reset(obj.pCorrelator);
          reset(obj.pQPSKDemodulator);
          reset(obj.pDescrambler);
          reset(obj.pBitGenerator);
          reset(obj.pErrorRateCalc);
       end

       function releaseImpl(obj)
          release(obj.pCorrelator);
          release(obj.pQPSKDemodulator);
          release(obj.pDescrambler);
          release(obj.pBitGenerator);
          release(obj.pErrorRateCalc);
       end
    end

    methods (Access=private)
       function bits2ASCII(obj,u)
          coder.extrinsic('disp')

          % Convert binary-valued column vector to 7-
bit decimal values.
          w = [64 32 16 8 4 2 1]; % binary digit
weighting
          Nbits = numel(u);
          Ny = Nbits/7;
          y = zeros(1,Ny);
          for i = 0:Ny-1
```

```
            y(i+1) = w*u(7*i+(1:7));
        end

        % Display ASCII message to command window
        if(obj.PrintOption)
            disp(char(y));
        end
    end
end
end
```

```
classdef QPSKChannelR < matlab.System
%#codegen

%   Copyright 2012-2013 The MathWorks, Inc.


    properties (Nontunable)
        DelayType = 'Triangle';
        RaisedCosineFilterSpan = 10;
        PhaseOffset = 47;
        SignalPower = 0.25;
        FrameSize = 100;
        UpsamplingFactor = 4;
        EbNo = 7;
        BitsPerSymbol = 2;
        FrequencyOffset = 5000;
        SampleRate = 200000;
    end

    properties (Access=private)
        pPhaseFreqOffset
        pVariableTimeDelay
        pAWGNChannel
    end

    properties (Constant, Access=private)
        pDelayStepSize = 0.05;
        pDelayMaximum = 8;
        pDelayMinimum = 0.1;
    end

    methods
        function obj = QPSKChannelR(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access=protected)
        function setupImpl(obj, ~, ~)
            obj.pPhaseFreqOffset =
comm.PhaseFrequencyOffset(...
                'PhaseOffset', obj.PhaseOffset, ...
                'FrequencyOffset',
obj.FrequencyOffset, ...
```

```
                'SampleRate',obj.SampleRate);
            obj.pVariableTimeDelay =
dsp.VariableFractionalDelay(...
                'MaximumDelay',
obj.FrameSize*obj.UpsamplingFactor);
            obj.pAWGNChannel = comm.AWGNChannel('EbNo',
obj.EbNo, ...
                'BitsPerSymbol', obj.BitsPerSymbol, ...
                'SignalPower', obj.SignalPower, ...
                'SamplesPerSymbol', obj.UpsamplingFactor);
        end


        function corruptSignal = stepImpl(obj, TxSignal,
count)

            % Calculates the delay
            if strcmp(obj.DelayType,'Ramp')
                delay = ...
                    min(((count - 1) * obj.pDelayStepSize
+ obj.pDelayMinimum), ...
                    (obj.FrameSize-
obj.RaisedCosineFilterSpan) ...
                    *obj.UpsamplingFactor); % Variable
delay taking the form of a ramp
            else
                % Variable delay taking the shape of a
triangle
                index = mod(count-
1,2*obj.pDelayMaximum/obj.pDelayStepSize);
                if index <=
obj.pDelayMaximum/obj.pDelayStepSize
                    delay = index * obj.pDelayStepSize;
                else
                    delay = 2*obj.pDelayMaximum - index *
obj.pDelayStepSize;
                end
            end

            % Signal undergoes phase/frequency offset
            rotatedSignal =
step(obj.pPhaseFreqOffset,TxSignal);

            % Delayed signal
```

```
            delayedSignal = step(obj.pVariableTimeDelay,
rotatedSignal, 0);

            % Signal passing through AWGN channel
            corruptSignal = step(obj.pAWGNChannel,
delayedSignal);

        end

        function resetImpl(obj)
            reset(obj.pPhaseFreqOffset);
            reset(obj.pVariableTimeDelay);
            reset(obj.pAWGNChannel);
        end

        function releaseImpl(obj)
            release(obj.pPhaseFreqOffset);
            release(obj.pVariableTimeDelay);
            release(obj.pAWGNChannel);
        end

        function N = getNumInputsImpl(~)
            N = 2;
        end
    end
end
```

```
classdef QPSKBitsGeneratorR < matlab.System
%#codegen
% Generates the bits for each frame

%   Copyright 2012 The MathWorks, Inc.

    properties (Nontunable)
        MessageLength = 105;
        BernoulliLength = 69;
        ScramblerBase = 2;
        ScramblerPolynomial = [1 1 1 0 1];
        ScramblerInitialConditions = [0 0 0 0];
    end

    properties (Access=private)
        pHeader
        pScrambler
        pMsgStrSet
        pCount
    end

    methods
        function obj = QPSKBitsGeneratorR(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access=protected)
        function setupImpl(obj, ~)
            bbc = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1
+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1]; % Bipolar
Barker Code
            ubc = ((bbc + 1) / 2)'; % Unipolar Barker
Code
            temp = (repmat(ubc,1,2))';
            obj.pHeader = temp(:);
            obj.pCount = 0;
            obj.pScrambler =
comm.Scrambler(obj.ScramblerBase, ...
                obj.ScramblerPolynomial,
obj.ScramblerInitialConditions);
            obj.pMsgStrSet = ['Hello world 1000';...
              'Hello world 1001';...
              'Hello world 1002';...
```

```
'Hello world 1003';...
'Hello world 1004';...
'Hello world 1005';...
'Hello world 1006';...
'Hello world 1007';...
'Hello world 1008';...
'Hello world 1009';...
'Hello world 1010';...
'Hello world 1011';...
'Hello world 1012';...
'Hello world 1013';...
'Hello world 1014';...
'Hello world 1015';...
'Hello world 1016';...
'Hello world 1017';...
'Hello world 1018';...
'Hello world 1019';...
'Hello world 1020';...
'Hello world 1021';...
'Hello world 1022';...
'Hello world 1023';...
'Hello world 1024';...
'Hello world 1025';...
'Hello world 1026';...
'Hello world 1027';...
'Hello world 1028';...
'Hello world 1029';...
'Hello world 1030';...
'Hello world 1031';...
'Hello world 1032';...
'Hello world 1033';...
'Hello world 1034';...
'Hello world 1035';...
'Hello world 1036';...
'Hello world 1037';...
'Hello world 1038';...
'Hello world 1039';...
'Hello world 1040';...
'Hello world 1041';...
'Hello world 1042';...
'Hello world 1043';...
'Hello world 1044';...
'Hello world 1045';...
'Hello world 1046';...
```

```
'Hello world 1047';...
'Hello world 1048';...
'Hello world 1049';...
'Hello world 1050';...
'Hello world 1051';...
'Hello world 1052';...
'Hello world 1053';...
'Hello world 1054';...
'Hello world 1055';...
'Hello world 1056';...
'Hello world 1057';...
'Hello world 1058';...
'Hello world 1059';...
'Hello world 1060';...
'Hello world 1061';...
'Hello world 1062';...
'Hello world 1063';...
'Hello world 1064';...
'Hello world 1065';...
'Hello world 1066';...
'Hello world 1067';...
'Hello world 1068';...
'Hello world 1069';...
'Hello world 1070';...
'Hello world 1071';...
'Hello world 1072';...
'Hello world 1073';...
'Hello world 1074';...
'Hello world 1075';...
'Hello world 1076';...
'Hello world 1077';...
'Hello world 1078';...
'Hello world 1079';...
'Hello world 1080';...
'Hello world 1081';...
'Hello world 1082';...
'Hello world 1083';...
'Hello world 1084';...
'Hello world 1085';...
'Hello world 1086';...
'Hello world 1087';...
'Hello world 1088';...
'Hello world 1089';...
'Hello world 1090';...
```

```
            'Hello world 1091';...
            'Hello world 1092';...
            'Hello world 1093';...
            'Hello world 1094';...
            'Hello world 1095';...
            'Hello world 1096';...
            'Hello world 1097';...
            'Hello world 1098';...
            'Hello world 1099'];
        end

        function [y,msg] = stepImpl(obj)

            % Converts the message string to bit format
            cycle = mod(obj.pCount,100);
            msgStr = obj.pMsgStrSet(cycle+1,:);
            msgBin = de2bi(int8(msgStr),7,'left-msb');
            msg =
reshape(double(msgBin).',obj.MessageLength,1);
            data = [msg ; randi([0 1],
obj.BernoulliLength, 1)];

            % Scramble the data
            scrambledData = step(obj.pScrambler, data);

            % Append the scrambled bit sequence to the
header
            y = [obj.pHeader ; scrambledData];

            obj.pCount = obj.pCount+1;
        end

        function resetImpl(obj)
            obj.pCount = 0;
            reset(obj.pScrambler);
        end

        function releaseImpl(obj)
            release(obj.pScrambler);
        end

        function N = getNumInputsImpl(~)
            N = 0;
        end
```

```
        function N = getNumOutputsImpl(~)
            N = 2;
        end
    end
end
```

```
%   Copyright 2012 The MathWorks, Inc.

classdef QPSKScopes < matlab.System

    properties (Access=private)
        pRxScope % Spectrum analyzer System object to
plot received signal after filtering
        pRxConstellation % Constellation scope System
object to plot received signal after filtering
        pFreqRecConstellation % Constellation scope
System object to plot received signal after filtering
        pTimingError % Time scope System object to plot
normalized timing error
    end

    methods
        function obj = QPSKScopes(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access=protected)
        function setupImpl(obj, ~, ~, ~)
            obj.pRxScope =
dsp.SpectrumAnalyzer('SpectralAverages', 2, ...
                 'PowerUnits', 'dBW', 'YLimits', [-130
-15], ...
                 'Title', 'After Raised Cosine Rx
Filter', ...
                 'SpectralAverages', 1, ...
                 'YLabel', 'PSD', ...
                 'SpectrumType', 'Power density', ...
                 'Position', figposition([1.5 37.2 24
26]));
            obj.pRxConstellation =
comm.ConstellationDiagram( ...
                 'ShowGrid', true, ...
                 'Position', figposition([1.5 72 17
20]), ...
                 'SamplesPerSymbol', 2, ...
                 'YLimits', [-1 1], ...
                 'XLimits', [-1 1], ...
                 'Title', 'After Raised Cosine Rx
Filter');
```

```
          obj.pFreqRecConstellation =
comm.ConstellationDiagram( ...
                 'ShowGrid', true, ...
                 'Position', figposition([19 72 17
20]), ...
                 'YLimits', [-1 1], ...
                 'XLimits', [-1 1], ...
                 'SamplesPerSymbol', 2, ...
                 'Title', 'After Fine Frequency
Compensation');
          obj.pTimingError = dsp.TimeScope( ...
                 'Title', 'Normalized Timing
Error', ...
                 'YLabel', 'mu (half symbols)',
'TimeSpan', 1000, ...
                 'YLimits', [-0.1 1.1], 'ShowGrid',
true, ...
                 'Position', figposition([26 45 11.8
17.8]));
       end


       function stepImpl(obj, RCRxSignal,
coarseCompBuffer, timingRecBuffer)

             % Plots the constellation of the filtered
signal
             step(obj.pRxConstellation,RCRxSignal);

             % Plots the spectrum scope of the filtered
signal
             step(obj.pRxScope,RCRxSignal);

             % Plots the constellation of the phase
recovered signal

step(obj.pFreqRecConstellation,coarseCompBuffer);

             % Plots the time scope of normalized
timing error
             step(obj.pTimingError,
timingRecBuffer(1:10:end));
       end
```

```
        function resetImpl(obj)
            reset(obj.pRxConstellation);
            reset(obj.pFreqRecConstellation);
            reset(obj.pRxScope);
            reset(obj.pTimingError);
        end

        function releaseImpl(obj)
            release(obj.pRxConstellation);
            release(obj.pFreqRecConstellation);
            release(obj.pRxScope);
            release(obj.pTimingError);
        end

        function N = getNumInputsImpl(~)
            N = 3;
        end

        function N = getNumOutputsImpl(~)
            N = 0;
        end
    end
end
```