----------------------QPSKTXRXSim.m-----------------------

```
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 = 174; % Number of modulated symbols per frame

%Channel coding parameters%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LDPC matrix size, rate must be 1/2
% Warning: encoding - decoding can be very long for large LDPC
matrix!
SimParams.LdpcRow = 148;
SimParams.LdpcColumn = 296;
SimParams.Ldpcmethod = 1;% Method for creating LDPC matrix (0 =
Evencol; 1 = Evenboth)
SimParams.LdpcnoCycle = 1;% Eliminate length-4 cycle
SimParams.LdpcOnePerCol = 3;% Number of 1s per column for LDPC matrix
SimParams.LdpcStrategy = 1;% LDPC matrix reorder strategy (0 = First;
1 = Mincol; 2 = Minprod)
SimParams.LdpcIteration = 1;% Number of iteration;
SimParams.LdpcH = makeLdpc(SimParams.LdpcRow, ...
                           SimParams.LdpcColumn, ...
                           SimParams.Ldpcmethod, ...
                           SimParams.LdpcnoCycle, ...
                           SimParams.LdpcOnePerCol);
[SimParams.LdpcNewH,SimParams.LdpcU,SimParams.LdpcL] =
makeParityChk(SimParams.LdpcH, SimParams.LdpcStrategy);
%Channel coding parameters%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% 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 = 20; % 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.TransmitterFilterCoeffici
 ents, ...
        'DataLength', prmQPSKTxRx.DataLength, ...
        'ScramblerBase', prmQPSKTxRx.ScramblerBase, ...
        'ScramblerPolynomial', prmQPSKTxRx.ScramblerPolynomial, ...
        'ScramblerInitialConditions',
 prmQPSKTxRx.ScramblerInitialConditions,...
        'LdpcNewH',prmQPSKTxRx.LdpcNewH,...
        'LdpcU',prmQPSKTxRx.LdpcU,...
        'LdpcL',prmQPSKTxRx.LdpcL);

    % 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,...
        'LdpcNewH',prmQPSKTxRx.LdpcNewH,...
        'LdpcIteration',prmQPSKTxRx.LdpcIteration,...
        'LdpcU',prmQPSKTxRx.LdpcU,...
        'LdpcL',prmQPSKTxRx.LdpcL,...
        'PrintOption', printData);

    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

end

if useScopes
     releaseQPSKScopes(hScopes);
end

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));
```

------------------------QPSKTransmitterR.m----------------------------

```
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];
        LdpcNewH=zeros(148,296);
        LdpcU=zeros(148,148);
        LdpcL=zeros(148,148);
    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-2*obj.MessageLength, ...
                'ScramblerBase', obj.ScramblerBase, ...
                'ScramblerPolynomial', obj.ScramblerPolynomial, ...
                'ScramblerInitialConditions', obj.ScramblerInitialConditions,...
                'LdpcNewH',obj.LdpcNewH,...
                'LdpcU',obj.LdpcU,...
                'LdpcL',obj.LdpcL);
```

```
            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);

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

---------------------------- QPSKBitsGeneratorR.m----------------------------

```
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];
        LdpcNewH=zeros(148,296);
        LdpcU=zeros(148,148);
        LdpcL=zeros(148,148);
    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/2, 1)];

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

            ldpccode = mod(obj.LdpcU\(obj.LdpcL\mod(obj.LdpcNewH(:, 148
+ 1:end)*scrambledData, 2)), 2);
            CodescrambledData = [ldpccode; scrambledData];


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

            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
```

------------------------------ QPSKReceiverR.m ----------------------------

```
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];
        LdpcNewH=zeros(148,296);
        LdpcU=zeros(148,148);
        LdpcL=zeros(148,148);
        LdpcIteration=1;
        PrintOption = false;
    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, ...
            'LdpcNewH',obj.LdpcNewH,...
            'LdpcIteration',obj.LdpcIteration,...
            'LdpcU',obj.LdpcU,...
            'LdpcL',obj.LdpcL,...
            'PrintOption', obj.PrintOption);
    end

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

            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
```

--------------------------------- QPSKDataDecoderR --------------------------

```
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];
        LdpcNewH=zeros(148,296);
        LdpcU=zeros(148,148);
        LdpcL=zeros(148,148);
        LdpcIteration=1;
        PrintOption = false;
    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-2*obj.MessageLength, ...
        'ScramblerBase', obj.DescramblerBase, ...
        'ScramblerPolynomial', obj.DescramblerPolynomial, ...
        'ScramblerInitialConditions', obj.DescramblerInitialConditions,...
        'LdpcNewH',obj.LdpcNewH,...
        'LdpcU',obj.LdpcU,...
        'LdpcL',obj.LdpcL);

    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);

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

    demodOutMsg=demodOut( ...
        obj.BarkerLength*log2(obj.ModulationOrder)+1 : ...
        obj.FrameSize*log2(obj.ModulationOrder));

    vhat = decodeBitFlip(demodOutMsg',obj.LdpcNewH,obj.LdpcIteration);
    deScrDataMsg=vhat(149:end)';

    % Performs descrambling
    deScrData = step(obj.pDescrambler, deScrDataMsg);

    % 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
```

-------------------------------- makeLdpc----------------------------------

```
function H = makeLdpc(M, N, method, noCycle, onePerCol)
% Create R = 1/2 low density parity check matrix
%
%   M           : Number of row
%   N           : Number of column
%   method      : Method for distributing non-zero element
%                 {0} Evencol : For each column, place 1s uniformly at random
%                 {1} Evenboth: For each column and row, place 1s uniformly at random
%   noCyle      : Length-4 cycle
%                 {0} Ignore (do nothing)
%                 {1} Eliminate
%   onePerCol: Number of ones per column
%
%   H           : Low density parity check matrix
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com


% Number of ones per row (N/M ratio must be 2)
if N/M ~= 2
   fprintf('Code rate must be 1/2\n');
end
onePerRow = (N/M)*onePerCol;

fprintf('Creating LDPC matrix...\n');

switch method
   % Evencol
   case {0}
      % Distribute 1s uniformly at random within column
      for i = 1:N
         onesInCol(:, i) = randperm(M)';
      end

      % Create non zero elements (1s) index
      r = reshape(onesInCol(1:onePerCol, :), N*onePerCol, 1);
      tmp = repmat([1:N], onePerCol, 1);
      c = reshape(tmp, N*onePerCol, 1);

      % Create sparse matrix H
```

```
        H = full(sparse(r, c, 1, M, N));

    % Evenboth
    case {1}
        % Distribute 1s uniformly at random within column
        for i = 1:N
            onesInCol(:, i) = randperm(M)';
        end

        % Create non zero elements (1s) index
        r = reshape(onesInCol(1:onePerCol, :), N*onePerCol, 1);
        tmp = repmat([1:N], onePerCol, 1);
        c = reshape(tmp, N*onePerCol, 1);

        % Make the number of 1s between rows as uniform as possible

        % Order row index
        [r, ix] = sort(r);

        % Order column index based on row index
        for i = 1:N*onePerCol
            cSort(i, :) = c(ix(i));
        end

        % Create new row index with uniform weight
        tmp = repmat([1:M], onePerRow, 1);
        r = reshape(tmp, N*onePerCol, 1);

        % Create sparse matrix H
        % Remove any duplicate non zero elements index using logical AND
        S = and(sparse(r, cSort, 1, M, N), ones(M, N));
        H = full(S);

end % switch

% Check rows that have no 1 or only have one 1
for i = 1:M

    n = randperm(N);
    % Add two 1s if row has no 1
    if length(find(r == i)) == 0
        H(i, n(1)) = 1;
        H(i, n(2)) = 1;
    % Add one 1 if row has only one 1
```

```
    elseif length(find(r == i)) == 1
        H(i, n(1)) = 1;
    end

end % for i

% If desired, eliminate any length-4 cycle
if noCycle == 1

    for i = 1:M
        % Look for pair of row - column
        for j = (i + 1):M
            w = and(H(i, :), H(j, :));
            c1 = find(w);
            lc = length(c1);
            if lc > 1

                % If found, flip one 1 to 0 in the row with less number of 1s
                if length(find(H(i, :))) < length(find(H(j, :)))
                    % Repeat the process until only one column left
                    for cc = 1:lc - 1
                        H(j, c1(cc)) = 0;
                    end
                else
                    for cc = 1:lc - 1
                        H(i, c1(cc)) = 0;
                    end
                end % if

            end % if
        end % for j
    end % for i

end % if

fprintf('LDPC matrix is created.\n');
```

-------------------------------- makeParityChk--------------------------------

```
function [newH,U,L] = makeParityChk( H, strategy)
% Generate parity check vector bases on LDPC matrix H using sparse LU decomposition
%
%   dSource : Binary source (0/1)
%   H         : LDPC matrix
%   strategy: Strategy for finding the next non-zero diagonal elements
%              {0} First    : First non-zero found by column search
%              {1} Mincol : Minimum number of non-zeros in later columns
%              {2} Minprod: Minimum product of:
%                              - Number of non-zeros its column minus 1
%                              - Number of non-zeros its row minus 1
%
%   c          : Check bits
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com

% Get the matric dimension
[M, N] = size(H);
% Set a new matrix F for LU decomposition
F = H;
% LU matrices
L = zeros(M, N - M);
U = zeros(M, N - M);

% Re-order the M x (N - M) submatrix
for i = 1:M

   % strategy {0 = First; 1 = Mincol; 2 = Minprod}
   switch strategy

      % Create diagonally structured matrix using 'First' strategy
      case {0}

         % Find non-zero elements (1s) for the diagonal
         [r, c] = find(F(:, i:end));

         % Find non-zero diagonal element candidates
         rowIndex = find(r == i);

         % Find the first non-zero column
```

```
    chosenCol = c(rowIndex(1)) + (i - 1);

  % Create diagonally structured matrix using 'Mincol' strategy
  case {1}

    % Find non-zero elements (1s) for the diagonal
    [r, c] = find(F(:, i:end));
    colWeight = sum(F(:, i:end), 1);

    % Find non-zero diagonal element candidates
    rowIndex = find(r == i);

    % Find the minimum column weight
    [x, ix] = min(colWeight(c(rowIndex)));
    % Add offset to the chosen row index to match the dimension of the...
    % original matrix F
    chosenCol = c(rowIndex(ix)) + (i - 1);

  % Create diagonally structured matrix using 'Minprod' strategy
  case {2}

    % Find non-zero elements (1s) for the diagonal
    [r, c] = find(F(:, i:end));
    colWeight = sum(F(:, i:end), 1) - 1;
    rowWeight = sum(F(i, :), 2) - 1;

    % Find non-zero diagonal element candidates
    rowIndex = find(r == i);

    % Find the minimum product
    [x, ix] = min(colWeight(c(rowIndex))*rowWeight);
    % Add offset to the chosen row index to match the dimension of the...
    % original matrix F
    chosenCol = c(rowIndex(ix)) + (i - 1);

  otherwise
    fprintf('Please select columns re-ordering strategy!\n');

end % switch

% Re-ordering columns of both H and F
tmp1 = F(:, i);
tmp2 = H(:, i);
F(:, i) = F(:, chosenCol);
```

```
    H(:, i) = H(:, chosenCol);
    F(:, chosenCol) = tmp1;
    H(:, chosenCol) = tmp2;

    % Fill the LU matrices column by column
    L(i:end, i) = F(i:end, i);
    U(1:i, i) = F(1:i, i);

    % There will be no rows operation at the last row
    if i < M

        % Find the later rows with non-zero elements in column i
        [r2, c2] = find(F((i + 1):end, i));
        % Add current row to the later rows which have a 1 in column i
        F((i + r2), :) = mod(F((i + r2), :) + repmat(F(i, :), length(r2), 1), 2);

    end % if

end % for i

% Find B.dsource
%z = mod(H(:, (N - M) + 1:end)*dSource, 2);

% Parity check vector found by solving sparse LU
%c = mod(U\(L\z), 2);

% Return the rearrange H
newH = H;

%fprintf('Message encoded.\n');
```

---------------------------- decodeBitFlip----------------------------------

```
function vHat = decodeBitFlip(ci, H, iteration)
% Hard-decision/bit flipping sum product algorithm LDPC decoder
%
%   rx          : Received signal vector (column vector)
%   H           : LDPC matrix
%   iteration : Number of iteration
%
%   vHat        : Decoded vector (0/1)
%
%
% Copyright Bagawan S. Nugroho, 2007
% http://bsnugroho.googlepages.com

[M,N] = size(H);

% Prior hard-decision
%ci = 0.5*(sign(rx') + 1);

% Initialization
rji = zeros(M, N);

% Asscociate the ci matrix with non-zero elements of H
qij = H.*repmat(ci, M, 1);

% Iteration
for n = 1:iteration

   %fprintf('Iteration : %d\n', n);

   % ----- Horizontal step -----
   for i = 1:M

      % Find non-zeros in the column
      c1 = find(H(i, :));

      % Get the summation of qij\c1(k)
      for k = 1:length(c1)

         rji(i, c1(k)) = mod(sum(qij(i, c1)) + qij(i, c1(k)), 2);

      end % for k
```

```
    end % for i

    % ------ Vertical step ------
    for j = 1:N

        % Find non-zero in the row
        r1 = find(H(:, j));

        % Number of 1s in a row
        numOfOnes = length(find(rji(r1, j)));

        for k = 1:length(r1)

            % Update qij, set '1' for majority of 1s else '0', excluding r1(k)
            if numOfOnes + ci(j) >= length(r1) - numOfOnes + rji(r1(k), j)
                qij(r1(k), j) = 1;
            else
                qij(r1(k), j) = 0;
            end

        end % for k

        % Bit decoding
        if numOfOnes + ci(j) >= length(r1) - numOfOnes
            vHat(j) = 1;
        else
            vHat(j) = 0;
        end

    end % for j

end % for n
```