----------------------------**1-QPSKTrans.m**----------------------------

```
clc;
clear;

% General simulation parameters
TSimParams.Upsampling = 4; % Upsampling factor
TSimParams.Fs = 2e5; % Sample rate
TSimParams.Ts = 1/TSimParams.Fs; % Sample time
TSimParams.FrameSize = 100; % Number of modulated symbols per frame

% Tx parameters
TSimParams.BarkerLength = 13; % Number of Barker code symbols
TSimParams.DataLength = (TSimParams.FrameSize - TSimParams.BarkerLength)*4; % Number of data payload bits per frame
TSimParams.MessageLength = 112; % Number of message bits per frame, 7 ASCII characters
TSimParams.FrameCount = 100;

TSimParams.RxBufferedFrames = 10; % Received buffer length (in frames)
TSimParams.RaisedCosineGroupDelay = 5; % Group delay of Raised Cosine Tx Rx filters (in symbols)
TSimParams.ScramblerBase = 2;
TSimParams.ScramblerPolynomial = [1 1 1 0 1];
TSimParams.ScramblerInitialConditions = [0 0 0 0];

% Generate square root raised cosine filter coefficients (required only for MATLAB example)
TSimParams.SquareRootRaisedCosineFilterOrder = 2*TSimParams.Upsampling*TSimParams.RaisedCosineGroupDelay;
TSimParams.RollOff = 0.5;

% Square root raised cosine transmit filter
ThTxFilt = fdesign.interpolator(TSimParams.Upsampling, ...
                    'Square Root Raised Cosine', TSimParams.Upsampling, ...
                    'N,Beta', TSimParams.SquareRootRaisedCosineFilterOrder, TSimParams.RollOff);
ThDTxFilt = design(ThTxFilt);
TSimParams.TransmitterFilterCoefficients = ThDTxFilt.Numerator/2;


%SDRu transmitter parameters
TSimParams.USRPCenterFrequency = 900e6;
TSimParams.USRPGain = 25;
TSimParams.USRPInterpolation = 1e8/TSimParams.Fs;
TSimParams.USRPFrameLength = TSimParams.Upsampling*TSimParams.FrameSize*TSimParams.RxBufferedFrames;
```

```
%Simulation Parameters
TSimParams.FrameTime = TSimParams.USRPFrameLength/TSimParams.Fs;
TSimParams.StopTime = 1000;

%%________________________________________________________________
prmQPSKTransmitter = TSimParams ;
%%________________________________________________________________

    % Initialize the components
    % Create and configure the transmitter System object
    hTx = QPSKTransmitterR(...
        'UpsamplingFactor', prmQPSKTransmitter.Upsampling, ...
        'MessageLength', prmQPSKTransmitter.MessageLength, ...
        'TransmitterFilterCoefficients',prmQPSKTransmitter.TransmitterFilterCoefficients, ...
        'DataLength', prmQPSKTransmitter.DataLength, ...
        'ScramblerBase', prmQPSKTransmitter.ScramblerBase, ...
        'ScramblerPolynomial', prmQPSKTransmitter.ScramblerPolynomial, ...
        'ScramblerInitialConditions', prmQPSKTransmitter.ScramblerInitialConditions);


    % Create and configure the SDRu


    ThSDRu = comm.SDRuTransmitter('192.168.10.2', ...
        'CenterFrequency',        prmQPSKTransmitter.USRPCenterFrequency, ...
        'Gain',                   prmQPSKTransmitter.USRPGain, ...
        'InterpolationFactor',    prmQPSKTransmitter.USRPInterpolation);

currentTime = 0;

%Transmission Process
while currentTime < prmQPSKTransmitter.StopTime
    % Bit generation, modulation and transmission filtering
    data = step(hTx);
    % Data transmission
    step(ThSDRu, data);
    % Update simulation time
    currentTime=currentTime+prmQPSKTransmitter.FrameTime
end

release(hTx);
release(ThSDRu);
```

----------------------------**2-QPSKRece.m**----------------------------

```
clc;
clear;
%   4-QAM 下修改频偏纠正和符号同步文件
SimParams.MasterClockRate = 100e6; %Hz
SimParams.Fs = 200e3; % Sample rate

% General simulation parameters
SimParams.M = 16; % M-PSK alphabet size
SimParams.Upsampling = 4; % Upsampling factor
SimParams.Downsampling = 2; % Downsampling factor
SimParams.Ts = 1/SimParams.Fs; % Sample time
SimParams.FrameSize = 100; % Number of modulated symbols per frame

% Rx parameters
SimParams.BarkerLength = 13; % Number of Barker code symbols
SimParams.DataLength = (SimParams.FrameSize - SimParams.BarkerLength)*4; % Number of
data payload bits per frame
SimParams.MessageLength = 112; % Number of message bits per frame, 7 ASCII characters
SimParams.FrameCount = 100;
SimParams.ScramblerBase = 2;
SimParams.ScramblerPolynomial = [1 1 1 0 1];
SimParams.ScramblerInitialConditions = [0 0 0 0];

SimParams.RxBufferedFrames = 10; % Received buffer length (in frames)
SimParams.RCFiltSpan = 10; % Filter span of Raised Cosine Tx Rx filters (in symbols)

% Generate square root raised cosine filter coefficients (required only for MATLAB example)
SimParams.SquareRootRaisedCosineFilterOrder =
SimParams.Upsampling*SimParams.RCFiltSpan;
SimParams.RollOff = 0.5;

% Square root raised cosine receive filter
hRxFilt = fdesign.decimator(SimParams.Upsampling/SimParams.Downsampling, ...
                    'Square Root Raised Cosine', SimParams.Upsampling, ...
                    'N,Beta',          SimParams.SquareRootRaisedCosineFilterOrder,
SimParams.RollOff);
hDRxFilt = design(hRxFilt, 'SystemObject', true);
SimParams.ReceiverFilterCoefficients = hDRxFilt.Numerator;

% Rx parameters
K = 1;
A = 1/sqrt(2);
% 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.
SimParams.PhaseErrorDetectorGain = 2*K*A^2+2*K*A^2; % K_p for Fine Frequency Compensation PLL, determined by 2KA^2 (for binary PAM), QPSK could be treated as two individual binary PAM
SimParams.PhaseRecoveryGain = 1; % K_0 for Fine Frequency Compensation PLL
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
SimParams.TimingRecoveryGain = -1; % K_0 for Timing Recovery PLL, fixed due to modulo-1 counter structure
SimParams.CoarseCompFrequencyResolution = 50; % Frequency resolution for coarse frequency compensation
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

%SDRu receiver parameters
SimParams.USRPCenterFrequency = 900e6;
SimParams.USRPGain = 31;
SimParams.USRPDecimationFactor = SimParams.MasterClockRate/SimParams.Fs;
SimParams.USRPFrontEndSampleRate = 1/SimParams.Fs;
SimParams.USRPFrameLength = SimParams.Upsampling*SimParams.FrameSize*SimParams.RxBufferedFrames;

%Simulation parameters
SimParams.FrameTime = SimParams.USRPFrameLength/SimParams.Fs;
SimParams.StopTime = 100;

prmQPSKReceiver=SimParams ;
prmQPSKReceiver.Platform = 'N200/N210/USRP2';
prmQPSKReceiver.Address = '192.168.10.2';

hRx = sdruQPSKRxR(...
'DesiredAmplitude', 1, ...
'ModulationOrder', prmQPSKReceiver.M, ...
'DownsamplingFactor', prmQPSKReceiver.Downsampling, ...
'CoarseCompFrequencyResolution', prmQPSKReceiver.CoarseCompFrequencyResolution, ...
'PhaseRecoveryLoopBandwidth', prmQPSKReceiver.PhaseRecoveryLoopBandwidth, ...

```
            'PhaseRecoveryDampingFactor',
prmQPSKReceiver.PhaseRecoveryDampingFactor, ...
            'TimingRecoveryLoopBandwidth',
prmQPSKReceiver.TimingRecoveryLoopBandwidth, ...
            'TimingRecoveryDampingFactor',
prmQPSKReceiver.PhaseRecoveryDampingFactor, ...
            'PostFilterOversampling',
prmQPSKReceiver.Upsampling/prmQPSKReceiver.Downsampling, ...
            'PhaseErrorDetectorGain',          prmQPSKReceiver.PhaseErrorDetectorGain, ...
            'PhaseRecoveryGain',               prmQPSKReceiver.PhaseRecoveryGain, ...
            'TimingErrorDetectorGain',         prmQPSKReceiver.TimingErrorDetectorGain, ...
            'TimingRecoveryGain',              prmQPSKReceiver.TimingRecoveryGain, ...
            'FrameSize',                       prmQPSKReceiver.FrameSize, ...
            'BarkerLength',                    prmQPSKReceiver.BarkerLength, ...
            'MessageLength',                   prmQPSKReceiver.MessageLength, ...
            'SampleRate',                      prmQPSKReceiver.Fs, ...
            'DataLength',                      prmQPSKReceiver.DataLength, ...
            'ReceiverFilterCoefficients',      prmQPSKReceiver.ReceiverFilterCoefficients, ...
            'DescramblerBase',                 prmQPSKReceiver.ScramblerBase, ...
            'DescramblerPolynomial',           prmQPSKReceiver.ScramblerPolynomial, ...
            'DescramblerInitialConditions',    prmQPSKReceiver.ScramblerInitialConditions,...
            'PrintOption',                     true);

radio = comm.SDRuReceiver(...
            'IPAddress',             prmQPSKReceiver.Address, ...
            'CenterFrequency',       prmQPSKReceiver.USRPCenterFrequency, ...
            'Gain',                  prmQPSKReceiver.USRPGain, ...
            'DecimationFactor',      prmQPSKReceiver.USRPDecimationFactor, ...
            'FrameLength',           prmQPSKReceiver.USRPFrameLength, ...
            'OutputDataType',        'double');

hSpectrum = dsp.SpectrumAnalyzer(...
      'Name',                  'Actual Frequency Offset',...
      'Title',                 'Actual Frequency Offset', ...
      'SpectrumType',          'Power density',...
      'FrequencySpan',         'Full', ...
      'SampleRate',            200e3, ...
      'YLimits',               [-130,0],...
      'SpectralAverages', 50, ...
      'FrequencySpan',         'Start and stop frequencies', ...
      'StartFrequency',        -100e3, ...
      'StopFrequency',         100e3,...
      'Position',              figposition([50 30 30 40]));
```

```
% Initialize variables
errorIndex=0;
%m=1;
%load ReceSignal.mat
while (true)
   %1. 从 USRP 读取 IQ 信号
   [corruptSignal, len] = step(radio);

 %   len=8000;
   %2. 能否成功读取数据长度
   if len < prmQPSKReceiver.USRPFrameLength
      errorIndex = errorIndex+1;
      disp ( 'Not enough samples returned!' ) ;
      disp(errorIndex)
   else

%    corruptSignal=ReceSignal(:,m);
%    m=m+1;
%    if m>100
%        break;
%    end

%     %3. 如果成功读取，画出接收信号的频谱图
%      corruptSignal = corruptSignal - mean(corruptSignal);   % remove DC component
%      step(hSpectrum, corruptSignal);

%     %4. 如果成功读取，画出接收信号的星座图
%      figure(1)
%      plot(corruptSignal(1:SimParams.Upsampling:end),'ro')
%      axis([-1 1 -1 1])
%      drawnow
%
   %5. AGC（自动增益控制）、匹配滤波后，得到 RCRxSignal，频偏纠正得到 coarseCompSignal，误码率计算得到 BER

    [RCRxSignal,coarseCompSignal,BER]= step(hRx, corruptSignal);

%%      %6. 画出匹配滤波后信号的星座图
%      figure(2)
%      plot(RCRxSignal(1:SimParams.Upsampling:end),'go')
%      axis([-1 1 -1 1])
%      drawnow
%%
%%      %7. 画出匹配滤波后信号的星座图
```

```
%        figure(3)
%        plot(coarseCompSignal(1:SimParams.Upsampling:end),'bo')
%        axis([-1 1 -1 1])
%        drawnow
%
    %8. 输出 BER
    fprintf('Error rate is = %f.\n',BER(1))
    end
end

release(hRx);
release(radio);
```

----------------------------3- QPSKTransmitter.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];
    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.pQPSKModulator         =      comm.RectangularQAMModulator(16,
'BitInput',true,...
                 'NormalizationMethod','Average power',...
                 'SymbolMapping', 'Custom', ...
```

```
                'CustomSymbolMapping', [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7]);


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

        function transmittedSignal = 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
```

---------------------------- **4-sdruQPSKRx** ----------------------------

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

%   Copyright 2012-2014 The MathWorks, Inc.

    properties (Nontunable)
        DesiredAmplitude
        ModulationOrder
        DownsamplingFactor
        CoarseCompFrequencyResolution
        PhaseRecoveryLoopBandwidth
        PhaseRecoveryDampingFactor
        TimingRecoveryLoopBandwidth
        TimingRecoveryDampingFactor
        PostFilterOversampling
        PhaseErrorDetectorGain
        PhaseRecoveryGain
        TimingErrorDetectorGain
        TimingRecoveryGain
        FrameSize
        BarkerLength
        MessageLength
        SampleRate
        DataLength
        ReceiverFilterCoefficients
        DescramblerBase
        DescramblerPolynomial
        DescramblerInitialConditions
        PrintOption
    end

    properties (Access=private)
        pAGC
        pRxFilter
        pCoarseFreqCompensator
        pFineFreqCompensator
        pTimingRec
        pDataDecod
        pOldOutput % Stores the previous output of fine frequency compensation which is used by the same System object for phase error detection
    end
```

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

methods (Access=protected)
    function setupImpl(obj, ~)
        obj.pAGC = comm.AGC;
        obj.pRxFilter = dsp.FIRDecimator(...
            obj.DownsamplingFactor,obj.ReceiverFilterCoefficients);
        obj.pCoarseFreqCompensator = QPSKCoarseFrequencyCompensatorR(...
            'ModulationOrder', obj.ModulationOrder, ...
            'CoarseCompFrequencyResolution',
obj.CoarseCompFrequencyResolution, ...
            'SampleRate', obj.SampleRate, ...
            'DownsamplingFactor', obj.DownsamplingFactor);

        % Refer C.57 to C.61 in Michael Rice's "Digital Communications
        % - A Discrete-Time Approach" for K1 and K2
        theta = obj.PhaseRecoveryLoopBandwidth/...
            (obj.PhaseRecoveryDampingFactor + ...
            0.25/obj.PhaseRecoveryDampingFactor)/obj.PostFilterOversampling;
        d = 1 + 2*obj.PhaseRecoveryDampingFactor*theta + theta*theta;
        K1 = (4*obj.PhaseRecoveryDampingFactor*theta/d)/...
            (obj.PhaseErrorDetectorGain*obj.PhaseRecoveryGain);
        K2 = (4*theta*theta/d)/...
            (obj.PhaseErrorDetectorGain*obj.PhaseRecoveryGain);
        obj.pOldOutput = complex(0); % used to store past value
        obj.pFineFreqCompensator = QPSKFineFrequencyCompensator( ...
            'ProportionalGain', K1, ...
            'IntegratorGain', K2, ...
            'DigitalSynthesizerGain', -1*obj.PhaseRecoveryGain);

        % Refer C.57 to C.61 in Michael Rice's "Digital Communications
        % - A Discrete-Time Approach" for K1 and K2
        theta = obj.TimingRecoveryLoopBandwidth/...
            (obj.TimingRecoveryDampingFactor + ...
            0.25/obj.TimingRecoveryDampingFactor)/obj.PostFilterOversampling;
        d = 1 + 2*obj.TimingRecoveryDampingFactor*theta + theta*theta;
        K1 = (4*obj.TimingRecoveryDampingFactor*theta/d)/...
            (obj.TimingErrorDetectorGain*obj.TimingRecoveryGain);
        K2 = (4*theta*theta/d)/...
            (obj.TimingErrorDetectorGain*obj.TimingRecoveryGain);
```

```
        obj.pTimingRec = QPSKTimingRecovery('ProportionalGain', K1,...
            'IntegratorGain', K2, ...
            'PostFilterOversampling', obj.PostFilterOversampling, ...
            'BufferSize', obj.FrameSize);
        obj.pDataDecod = sdruQPSKDataDecoderR('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);
    end


    function [RCRxSignal,coarseCompSignal,BER] =    stepImpl(obj, bufferSignal)

        % Apply automatic gain control to the signal
        AGCSignal = obj.DesiredAmplitude*step(obj.pAGC, bufferSignal);

        % Pass the signal through square root raised cosine received
        % filter
        RCRxSignal = step(obj.pRxFilter,AGCSignal);

        % Coarsely compensate for the frequency offset
        coarseCompSignal = step(obj.pCoarseFreqCompensator, RCRxSignal);

        % Buffers to store values required for plotting
        coarseCompBuffer = ...
            coder.nullcopy(complex(zeros(size(coarseCompSignal))));
        timingRecBuffer = coder.nullcopy(zeros(size(coarseCompSignal)));

        % Scalar processing for fine frequency compensation and timing
        % recovery
        BER = zeros(3,1);
        for i=1:length(coarseCompSignal)

            % Fine frequency compensation
            fineCompSignal = step(obj.pFineFreqCompensator, ...
                [obj.pOldOutput coarseCompSignal(i)]);
            coarseCompBuffer(i) = fineCompSignal;
            obj.pOldOutput = fineCompSignal;
```

```
                % Timing recovery of the received data
                [dataOut, isDataValid, timingRecBuffer(i)] = ...
                    step(obj.pTimingRec, fineCompSignal);

                if isDataValid
                    % Decoding the received data
                    BER = step(obj.pDataDecod, dataOut);
                end
            end

        end

    end
end
```

---------------------------- **5-sdruQPSKDataDecoder**--------------------------

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

%    Copyright 2012 The MathWorks, Inc.

    properties (Nontunable)
        FrameSize
        BarkerLength
        ModulationOrder
        DataLength
        MessageLength
        DescramblerBase
        DescramblerPolynomial
        DescramblerInitialConditions
        PrintOption
    end

    properties (Access=private)
        pCount
        pDelay
        pPhase
        pBuffer
        pModulator
        pModulatedHeader
        pCorrelator
        pQPSKDemodulator
        pDescrambler
        pBitGenerator
        pBitGeneratorSync
        pBER
        pSyncFlag
        pSyncIndex
        pFrameIndex
    end


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

```
    methods (Access=protected)
        function setupImpl(obj, ~)
            [obj.pCount, obj.pDelay, obj.pPhase] = deal(0);
            obj.pFrameIndex=1;
            obj.pSyncIndex=0;
            obj.pSyncFlag=true;
            obj.pBuffer=dsp.Buffer(obj.FrameSize*2, obj.FrameSize);
            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
            header = (repmat(ubc,1,2))';
            header = header(:);

%             obj.pModulator = comm.QPSKModulator('BitInput', true, ...
%                 'PhaseOffset', pi/4);

            obj.pModulator = comm.RectangularQAMModulator(16, 'BitInput',true,...
                'NormalizationMethod','Average power',...
                'SymbolMapping', 'Custom', ...
                'CustomSymbolMapping', [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7]);

            obj.pModulatedHeader = step(obj.pModulator, header); % Modulate the
header
            obj.pCorrelator = dsp.Crosscorrelator;
            %obj.pQPSKDemodulator = comm.QPSKDemodulator('PhaseOffset',pi/4, ...
            %    'BitOutput', true);
            obj.pQPSKDemodulator = comm.RectangularQAMDemodulator(...
                'ModulationOrder', 16,    ...
                'BitOutput', true, ...
                'NormalizationMethod', 'Average power', 'SymbolMapping', 'Custom', ...
                'CustomSymbolMapping', [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7]);

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

        function    BER = stepImpl(obj, DataIn)

            % Buffer one frame in case that contiguous data scatter across
            % two adjacent frames
            rxData = step(obj.pBuffer,DataIn);

            % Get a frame of data aligned on the frame boundary
```

```
Data = rxData(obj.pDelay+1:obj.pDelay+length(rxData)/2);

% Phase estimation
y = mean(conj(obj.pModulatedHeader) .* Data(1:obj.BarkerLength));

% Compensating for the phase offset
if Data(1)~=0
    phShiftedData = Data .* exp(-1j*obj.pPhase);
else
    phShiftedData = complex(zeros(size(Data)));
end

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

% Perform 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);

% Finding the delay to achieve frame synchronization
z=abs(step(obj.pCorrelator,obj.pModulatedHeader,DataIn));
[~, ind] = max(z);
obj.pDelay = mod(length(DataIn)-ind,(length(DataIn)-1));

% Phase ambiguity correction
obj.pPhase = round(angle(y)*2/pi)/2*pi;

% Print received frame and estimate the received frame index
[estimatedFrameIndex,syncIndex]=bits2ASCII(obj,Received);
obj.pSyncIndex = syncIndex;
% Once it is possible to decode the frame index four times,
% frame synchronization is achieved
if ((obj.pSyncFlag) && (estimatedFrameIndex~=100) && (obj.pSyncIndex>=4))
    obj.pFrameIndex=estimatedFrameIndex;
    obj.pSyncFlag=false;
end
% With the estimated frame index, estimate the transmitted
% message
transmittedMessage=messEstimator(obj.pFrameIndex, obj);
```

```
            % Calculate the BER
            BER = step(obj.pBER,transmittedMessage,Received);
            obj.pCount = obj.pCount + 1;
            obj.pFrameIndex = obj.pFrameIndex + 1;

        end

        function resetImpl(obj)
            reset(obj.pBuffer);
        end

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

    end

    methods (Access=private)
        function [estimatedFrameIndex,syncIndex]=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);
            % Obtain ASCII values of received frame
            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
            % Retrieve last 2 ASCII values
            decodedNumber=y(Ny-1:end);
            % Create lookup table of ASCII values and corresponding integer numbers
            look_tab=zeros(2,10);
            look_tab(1,:)=0:9;
            look_tab(2,:)=48:57;
            % Initialize variables
            estimatedFrameIndex=100;
            syncIndex=0;
```

```
        onesPlace=0;
        tensPlace=0;
        dec_found=false;
        unity_found=false;

        % Index lookup table with decoded ASCII values
        % There are more efficient ways to perform vector indexing
        % using MATLAB functions like find(). However, to meet codegen
        % requirements, the usage of the four loop was necessary.

        for ii=1:10
            % Find the ones place in the lookup table
            if ( decodedNumber(1) == look_tab(2,ii) )
                onesPlace=10*look_tab(1,ii);
                dec_found=true;
            end
            % Find the tens place in the lookup table
            if ( decodedNumber(2) == look_tab(2,ii) )
                tensPlace=look_tab(1,ii);
                unity_found=true;
            end
        end
        % Estimate the frame index
        if(dec_found && unity_found && obj.pSyncFlag)
            estimatedFrameIndex=onesPlace+tensPlace;
            syncIndex=obj.pSyncIndex+1;
        end


    end

    function msg = messEstimator(ind, obj)

        MsgStrSet = ['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'];
        cycle = mod(ind,100);
        msgStr = MsgStrSet(cycle+1,:);
        msgBin = de2bi(int8(msgStr),7,'left-msb');
        msg = reshape(double(msgBin).',obj.MessageLength,1);

    end


  end
end
```

--------------------------**6-QPSKCoarseFrequencyCompensator**-----------------

```
classdef QPSKCoarseFrequencyCompensatorR < matlab.System
%#codegen
%     This object is used only in supporting packages.
%
%    Copyright 2012-2014 The MathWorks, Inc.

    properties (Nontunable)
        ModulationOrder = 4;
        CoarseCompFrequencyResolution = 50;
        SampleRate = 200000;
        DownsamplingFactor = 2;
    end

    properties (Access=private)
        pPhaseFreqOffset
        pCoarseFreqEst
    end

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

    methods (Access=protected)
        function setupImpl(obj, ~)
            currentSampleRate = obj.SampleRate/obj.DownsamplingFactor;
            obj.pPhaseFreqOffset = comm.PhaseFrequencyOffset(...
                'PhaseOffset', 0, ...
                'FrequencyOffsetSource', 'Input port' , ...
                'SampleRate', currentSampleRate);
            obj.pCoarseFreqEst = comm.QAMCoarseFrequencyEstimator( ...
                'FrequencyResolution', obj.CoarseCompFrequencyResolution, ...
                'SampleRate', currentSampleRate);
        end

        function compensatedSignal = stepImpl(obj, filteredSignal)

            % Find the frequency used for correction (the negative of the
            % actual offset)
            FreqOffset = -step(obj.pCoarseFreqEst, filteredSignal);
```

```
            % Remove the frequency offset
            compensatedSignal = ...
                step(obj.pPhaseFreqOffset,filteredSignal,FreqOffset);

        end

        function resetImpl(obj)
            reset(obj.pPhaseFreqOffset);
            reset(obj.pCoarseFreqEst);
        end

        function releaseImpl(obj)
            release(obj.pPhaseFreqOffset);
            release(obj.pCoarseFreqEst);
        end
    end
end
```

---------------------------- **7-QPSKBitsGenerator**----------------------------

```
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
```