Matlab Simulation Example 3: PHY abstraction under 11ax OFDM MIMO system

Matlab code part is colored in orange.

Setup:
MCS = 0
num of transmit antenna = 4, num of receive antenna = 2, num of stream = 2
Channel: Model-D, bandwidth = **80MHz**
APEP length = 1000
Channel coding = LDPC

How to change the code in Example 2?
The only part need to be modified is in the “1 full PHY” folder

First, in fullPHY.m

```
mcs = [0]; % Vector of MCS to simulate between 0 and 9
numTxRx = [4 2]; % Matrix of MIMO schemes, each row is [numTx numRx]
chan = "Model-D"; % String array of delay profiles to simulate
maxnumberrors = 40*1e3;  % The maximum number of packet errors at an SNR point
maxNumPackets = 40*1e3; % The maximum number of packets at an SNR point
% maxnumberrors = 1e1;  % The maximum number of packet errors at an SNR point
% maxNumPackets = 1e2; % The maximum number of packets at an SNR point

% Fixed PHY configuration for all simulations
cfgHE = wlanHESUConfig;
cfgHE.ChannelBandwidth = 'CBW80'; % Channel bandwidth
bandwidth = cfgHE.ChannelBandwidth;
cfgHE.APEPLength = 1000;          % Payload length in bytes
cfgHE.ChannelCoding = 'LDPC';     % Channel coding
```

Then, in getBoxSimParams

- Set the SNRs for Model-D, 4x2
- Set sp.Config.NumSpaceTimeStreams = 2;

Please notice the following code in **bold**

```
function simParams =
getBox0SimParams(chans,numTxRx,mcs,cfgHE,maxNumErrors,maxNumPackets)
% getBox0SimParams Example helper function

%   Copyright 2019 The MathWorks, Inc.

% These arrays define the value and order SNRs are defined
channelConfigs = ["Model-B","Model-D"];
anteannaSNRConfigs = [1 1; 4 2; 8 2];

snr = {
```

```
% Model-B
[ ...
{... % 1x1
[-3:4:9,11], ...   % MCS 0
[1:4:13], ...     % MCS 1
[2:4:18], ...     % MCS 2
[7:4:19,21], ...     % MCS 3
[9:4:25], ...     % MCS 4
[13:4:29], ...    % MCS 5
[14:4:30], ...    % MCS 6
[16:4:32,34], ...    % MCS 7
[18:4:34,36] ... % MCS 8
[18:4:38] ...     % MCS 9
}; ...
    {... % 4x2
1:3:17, ...  % MCS 0
5:3:23, ...  % MCS 1
9:4:30, ...  % MCS 2
12:4:36, ... % MCS 3
16:4:40, ... % MCS 4
20:4:43, ... % MCS 5
22:4:46, ... % MCS 6
24:4:48, ... % MCS 7
26:4:50 ...  % MCS 8
29:4:54 ...  % MCS 9
}; ...
    {... % 8x2
1:3:17, ...  % MCS 0
5:3:23, ...  % MCS 1
9:4:30, ...  % MCS 2
12:4:36, ... % MCS 3
16:4:40, ... % MCS 4
20:4:43, ... % MCS 5
22:4:46, ... % MCS 6
24:4:48, ... % MCS 7
26:4:50 ...  % MCS 8
29:4:54 ...  % MCS 9
}; ...
```

```
    ];

% Model-D
    [ ...
    {... % 1x1
    [-3 -2 -1 10], ...     % MCS 0
    0:4:12, ...     % MCS 1
    2:4:18, ...     % MCS 2
    8:4:20, ...     % MCS 3
    [11,15,19,21], ...    % MCS 4
    [14:4:26,28], ...    % MCS 5
    16:4:28, ...    % MCS 6
    18:4:30, ...    % MCS 7
    21.5:4:37.5 ... % MCS 8
    20:4:36 ...     % MCS 9
    }; ...
       {... % 4x2
    [-3:1:-1,-0.5], ...  %[-3:1:-1,-0.5,0,0.5,0.75,1], ...   % MCS 0
    [0,1,1.5,2], ... %[0,1,1.5,2,2.5,2.75,3,3.25], ...   % MCS 1
    [4:1:6,6.5], ... %[4:1:6,6.5,7,7.5,8,8.25], ...   % MCS 2
    [5:1:7,7.5], ... %[5:1:7,7.5,8,8.5,9,9.25], ...  % MCS 3
    [11,11.5,12,12.5], ... %[11,11.5,12,12.5,13,13.5,13.75,14], ...  % MCS 4
    [12:1:15], ... %[12:1:15,15.5,16,16.5,16.75], ...  % MCS 5
    [15:1:17,17.5], ... %[15:1:17,17.5,18,18.5,19,19.25], ...  % MCS 6
    [16:1:18,18.5], ... %[16:1:18,18.5,19,19.5,20,20.25], ...  % MCS 7
    [18.5:1:21.5], ... %[18.5:1:21.5,22,22.5,23,23.5] ...   % MCS 8
    [22,23,23.5,24], ... %[22,23,23.5,24,24.5,25,25.25,25.5] ...   % MCS 9
    }; ...
          {... % 8x2
    1:3:17, ...  % MCS 0
    5:3:23, ...  % MCS 1
    9:4:30, ...  % MCS 2
    12:4:36, ... % MCS 3
    [11:1:14], ... % MCS 4
    20:4:43, ... % MCS 5
    22:4:46, ... % MCS 6
    24:4:48, ... % MCS 7
    26:4:50 ...  % MCS 8
```

```
    29:4:54 ...  % MCS 9
    }; ...
    ] ...
    };

% Create channel configuration
tgaxChannel = wlanTGaxChannel;
tgaxChannel.DelayProfile = 'Model-D';
tgaxChannel.NumTransmitAntennas = cfgHE.NumTransmitAntennas;
tgaxChannel.NumReceiveAntennas = 1;
tgaxChannel.TransmitReceiveDistance = 15; % Distance in meters for NLOS
tgaxChannel.ChannelBandwidth = cfgHE.ChannelBandwidth;
tgaxChannel.LargeScaleFadingEffect = 'None';
fs = wlanSampleRate(cfgHE);
tgaxChannel.SampleRate = fs;
tgaxChannel.PathGainsOutputPort = true;
tgaxChannel.NormalizeChannelOutputs = false;

% Generate a structure array containing the simulation parameters,
% simParams. Each element contains the parameters for a simulation.
simParamsRef = struct('MCS',0,'SNR',0,'RandomSubstream',0,'Config',cfgHE, ...
    'MaxNumPackets',maxNumPackets,'MaxNumErrors',maxNumErrors, ...
    'NumTransmitAntennas',0,'NumReceiveAntennas',0,'DelayProfile',"Model-B",...
    'Channel',tgaxChannel);
simParams = repmat(simParamsRef,0,0);
% There must be a SNR cell for each channel
assert(all(numel(channelConfigs)==numel(snr)))
% There must be a SNR cell element for each MIMO configuration
assert(all(size(anteannaSNRConfigs,1)==cellfun(@(x)size(x,1),snr)))
for ichan = 1:numel(chans)
    channelIdx = chans(ichan)==channelConfigs;
    for itxrx = 1:size(numTxRx,1)
        numTxRxIdx = all(numTxRx(itxrx,:)==anteannaSNRConfigs,2);
        for imcs = 1:numel(mcs)
            snrIdx = mcs(imcs)+1;
            for isnr = 1:numel([snr{channelIdx}{numTxRxIdx,snrIdx}])
                % Set simulation specific parameters
                sp = simParamsRef;
```

```
            sp.MCS = mcs(imcs);
            sp.NumTransmitAntennas = numTxRx(itxrx,1);
            sp.NumReceiveAntennas = numTxRx(itxrx,2);
            sp.DelayProfile = chans(ichan);

            % Set random substream for reproducible results
            sp.RandomSubstream = isnr;

            % Setup PHY configuration
            sp.Config.MCS = mcs(imcs);
            sp.Config.NumTransmitAntennas = numTxRx(itxrx,1);
            sp.Config.NumSpaceTimeStreams = 2;
            sp.Config.SpatialMapping = 'Fourier';

            % Configure channel model
            sp.Channel = clone(tgaxChannel);
            sp.Channel.DelayProfile = chans(ichan);
            sp.Channel.NumTransmitAntennas = numTxRx(itxrx,1);
            sp.Channel.NumReceiveAntennas = numTxRx(itxrx,2);

            % Lookup SNR to simulate
            sp.SNR = snr{channelIdx}{numTxRxIdx,snrIdx}(isnr);

            % Append to other tests
            simParams = [simParams sp]; %#ok<AGROW>
         end
      end
   end
end

end
```

Finally, in box0Simulation.m, enable transmit precoding

```
while numPacketErrors<=maxNumErrors && numPkt<=maxNumPackets
   reset(tgaxChannel); % Reset channel for different realization

   % ----------------------------------------------------------------
   % Comment this to run simulation without beamforming
   % Generate NDP packet - with an empty PSDU as no data
```

```
txNDP = wlanWaveformGenerator([],cfgNDP);
% For each user STA, pass the NDP packet through the channel and calculate
% the feedback channel state matrix by SVD.
% Received waveform at user STA with 50 sample padding. No noise.
rxNDP = tgaxChannel([txNDP; zeros(50,size(txNDP,2))]);

% Get the full-band beamforming feedback for a user
staFeedback = heUserBeamformingFeedback(rxNDP,cfgNDP);
% For each RU, calculate the steering matrix to apply
% Calculate the steering matrix to apply to the RU given the feedback
steeringMatrix = heSUCalculateSteeringMatrix(staFeedback,cfgHE,cfgNDP);

% Apply the steering matrix to each RU
cfgHE.SpatialMapping = 'Custom';
cfgHE.SpatialMappingMatrix = steeringMatrix;
% --------------------------------------------------------------------

% Generate a packet with random PSDU
psduLength = getPSDULength(cfgHE); % PSDU length in bytes
txPSDU = randi([0 1],psduLength*8,1,'int8');
tx = wlanWaveformGenerator(txPSDU,cfgHE);
% Add trailing zeros to allow for channel delay
txPad = [tx; zeros(50,cfgHE.NumTransmitAntennas)];

% Pass through a fading indoor TGax channel

[rx,pathGains] = tgaxChannel(txPad);

% Get perfect timing offset and channel matrix for HE-LTF field
heltfPathGains = pathGains(ind.HELTF(1):ind.HELTF(2),:,:,:,:);
pktOffset = channelDelay(heltfPathGains,pathFilters);
chan =
helperPerfectChannelEstimate(heltfPathGains,pathFilters,ofdmInfo.FFTLength,ofdmInfo.CPLen
gth,ofdmInfo.ActiveFFTIndices,pktOffset);

% Calculate SINR using abstraction
% As multiple symbols returned average over symbols and permute
% for calculations
% Get precoding matrix for abstraction
Wtx = getPrecodingMatrix(cfgHE); % Include cyclic shift applied per STS
Wtx = Wtx/sqrt(cfgHE.NumSpaceTimeStreams);
Htxrx = permute(mean(chan,2),[1 3 4 2]); % Nst-by-Nt-by-Nr
Ptxrx = 1; % Assume transmit power is 0dBW
sinr = calculateSINR(Htxrx,Ptxrx,Wtx,N0);
sinrStore(:,:,numPkt) = sinr;

% Link performance model - estimate PER using abstraction
[perAbs,effSINR] = estimateLinkPerformance(Abstraction,sinr,cfgHE);

% Flip a coin for the abstracted PHY
packetErrorAbs = rand(1)<=perAbs;
```

```
    numPacketErrorsAbs = numPacketErrorsAbs+packetErrorAbs;

    % Store outputs for analysis
    perAbsRawStore(numPkt) = perAbs;
    perAbsStore(numPkt) = packetErrorAbs;
    snreffStore(numPkt) = effSINR;

    % Pass the waveform through AWGN channel
    rx = awgnChannel(rx);

    % Demodulate data symbols
    rxData = rx(pktOffset+(ind.HEData(1):ind.HEData(2)),:);
    demodSym = wlanHEDemodulate(rxData,'HE-Data',cfgHE);

    % Extract data subcarriers from demodulated symbols and channel
    % estimate
    demodDataSym = demodSym(ofdmInfo.DataIndices,:,:);

    % Get channel estimate from channel matrix (include spatial mapping
    % and cyclic shift)
    chanEst = heChannelToChannelEstimate(chan,cfgHE);
    chanEstAv = permute(mean(chanEst,2),[1 3 4 2]); % Average over symbols
    chanEstData = chanEstAv(ofdmInfo.DataIndices,:,:);

    % Calculate single stream pilot estimates per symbol and noise
    % estimate
    chanEstSSPilots = permute(sum(chanEst(ofdmInfo.PilotIndices,:,:,:),3),[1 2 4 5 3]);
    demodPilotSym = demodSym(ofdmInfo.PilotIndices,:,:);
    nVarEst = heNoiseEstimate(demodPilotSym,chanEstSSPilots,cfgHE);

    % Equalization and STBC combining
    [eqDataSym,csi] = heEqualizeCombine(demodDataSym,chanEstData,nVarEst,cfgHE);
    rxPSDU = wlanHEDataBitRecover(eqDataSym,nVarEst,csi,cfgHE);

    % Determine if any bits are in error, i.e. a packet error
    packetError = ~isequal(txPSDU,rxPSDU);
    perStore(numPkt) = packetError;
    numPacketErrors = numPacketErrors+packetError;

    numPkt = numPkt+1;
end
```

Other steps are exactly the same as the Example 2.