Introduction to DFEEYE

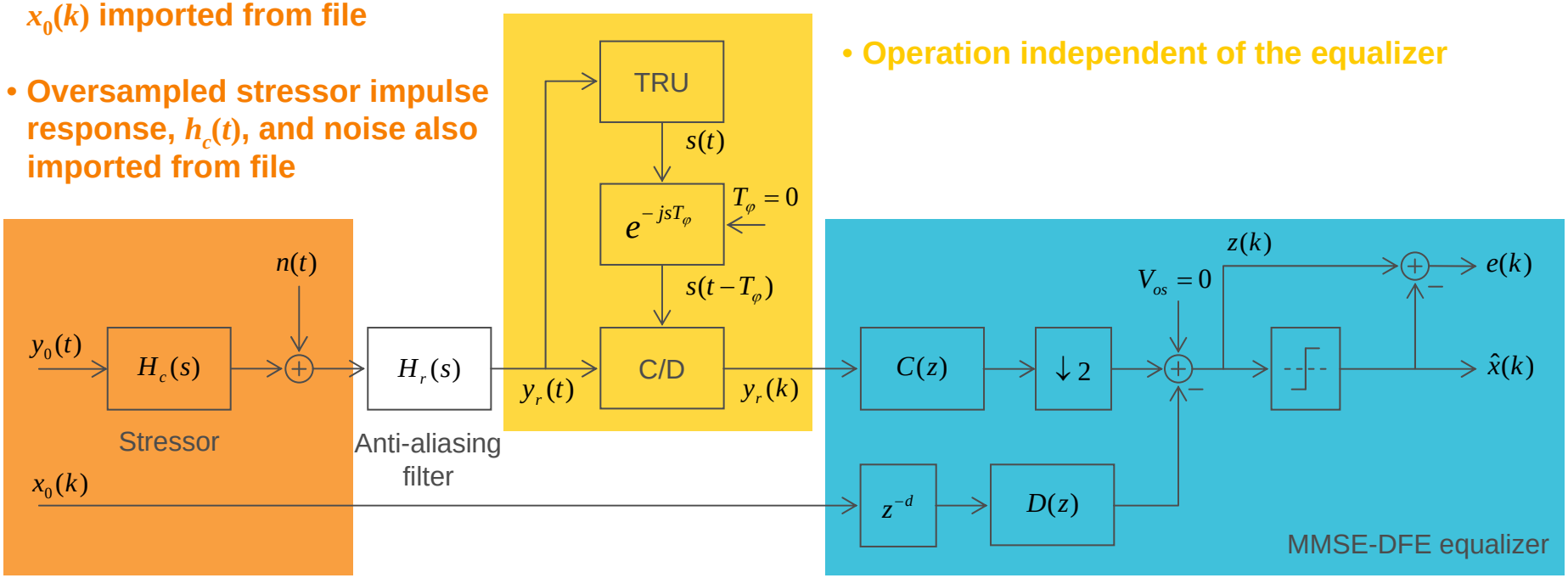
Adam Healey, Mark Marlett
LSI Corporation
September 27, 2007
T11/07-550v0

Overview

- DFEEYE derives “traditional” eye diagram figures of merit from waveforms with no discernable eye opening
  - Includes a reference equalizer of finite complexity
  - Output is a measure of horizontal and vertical eye opening
- DFEEYE includes a timing recovery function that emulates the prescribed jitter timing reference
- DFEEYE supports the inclusion of compliance transfer functions (stressors) for transmitter testing
- DFEEYE can be used to characterize signals for receiver stress testing

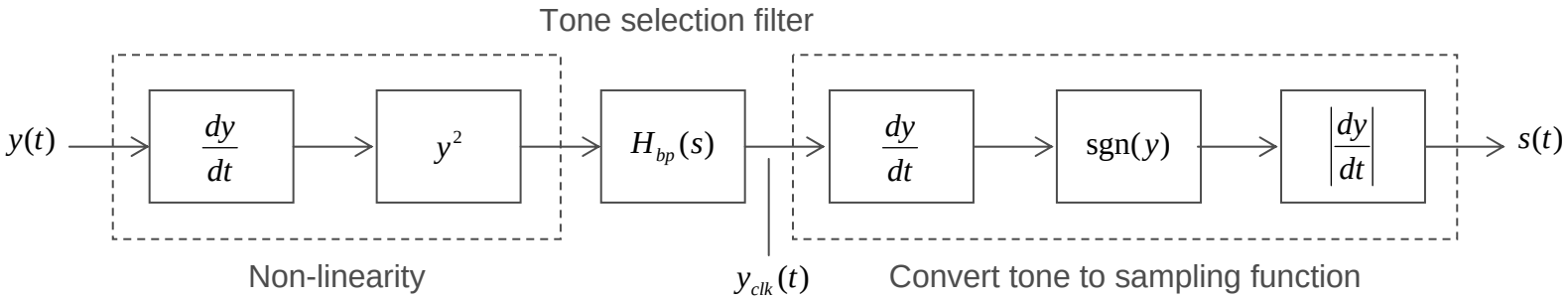
DFEEYE functional diagram

- Oversampled waveform, $y_0(t)$, and corresponding symbols $x_0(k)$ imported from file
- Oversampled stressor impulse response, $h_c(t)$, and noise also imported from file
- Waveform and symbol files use the format required by TWDP to support re-use of data and comparison of tools



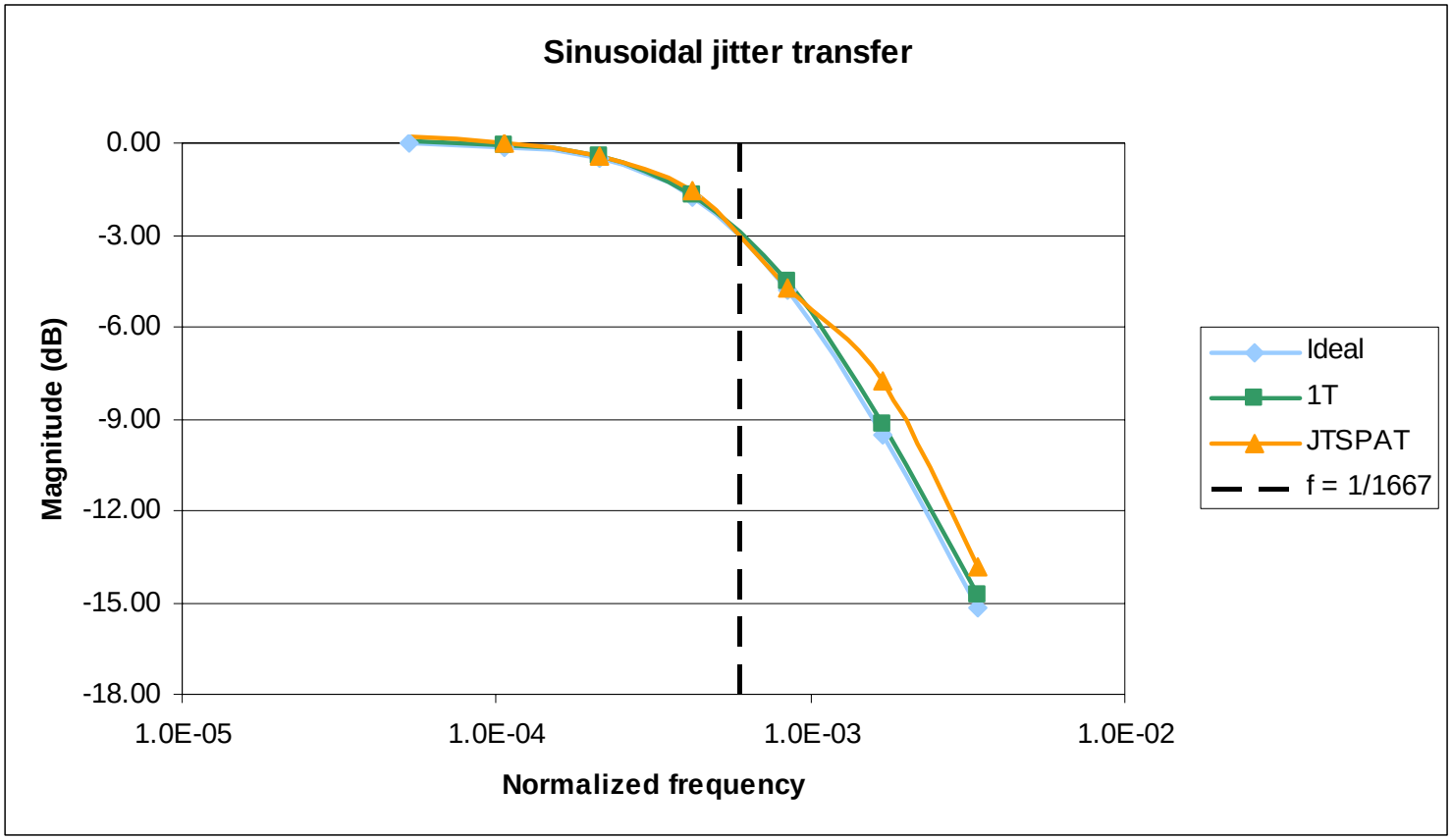
- TRU uses non-linear spectral line technique to emulate prescribed jitter timing reference
- Operation independent of the equalizer
- Textbook MMSE solution is used to calculate feed-forward and feedback filter coefficients
- Driven by symbols imported from file (e.g. all decisions are correct) with optimal delay, d

Timing recovery unit (TRU)



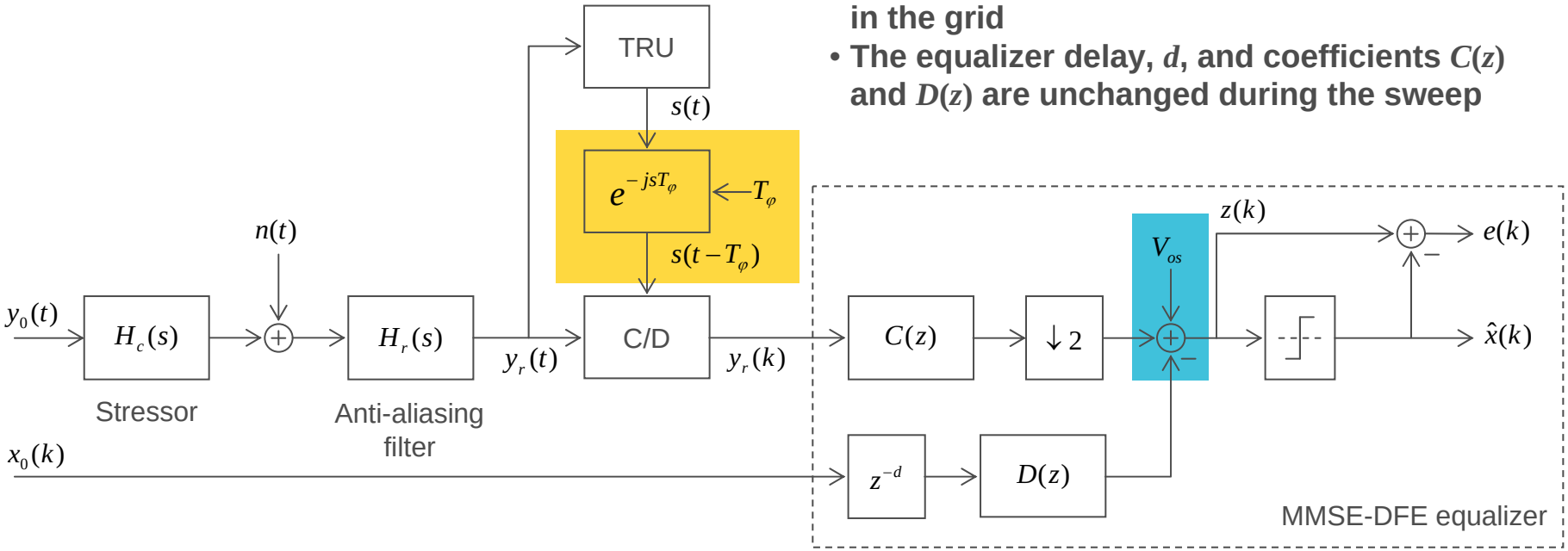
- Utilize textbook non-linear spectral line technique
- Bandwidth and order of band-pass filter set to emulate prescribed reference timing recovery unit (per MJSQ)
  - First-order low-pass response with respect to jitter
  - Corner frequency set to 1/1667 of the signaling speed
- Straight-forward, one-shot calculation

TRU performance



DFEEYE eye diagram construction

- Time offset, T_φ, and voltage offset, V_{os}, are swept over a grid
- The bit error ratio is computed at each point in the grid
- The equalizer delay, d, and coefficients $C(z)$ and $D(z)$ are unchanged during the sweep

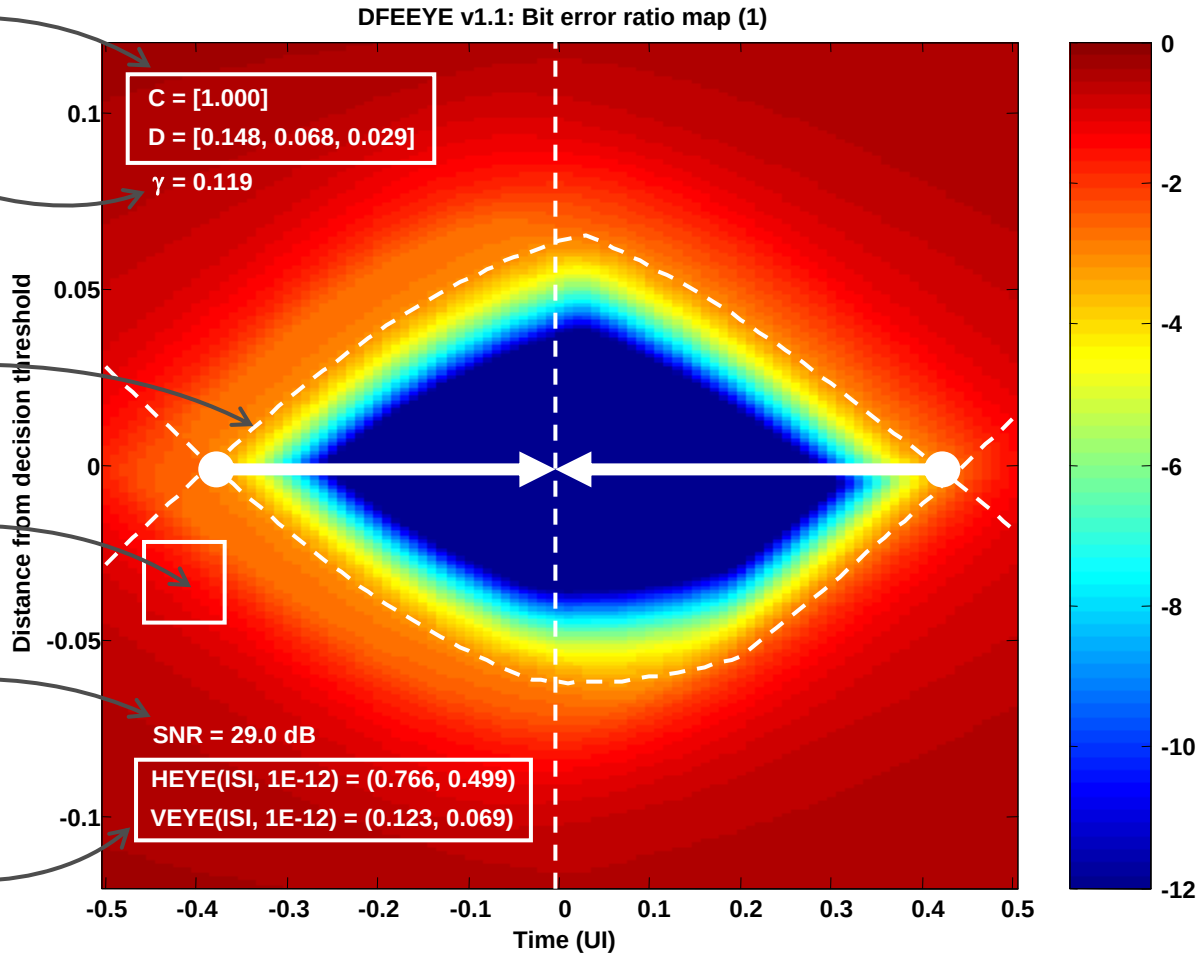


Tour of DFEEYE output

- Feed-forward and feedback coefficients
- Equalizer target amplitude (e.g. the mean distance to threshold for a “0” or a “1”)
- Eye contour indicates eye opening in the absence of noise (e.g. due to ISI only)
- Color gradient proportional to log(BER)

$$SNR = 20\log\left(\frac{VEYE(ISI)}{\sigma}\right)$$

- Figure of merit: eye opening defined to be twice the min. distance to nominal phase or threshold



System requirements

- DFEEYE is a MATLAB® script
  - Runs on a basic installation with no add-ons required
  - Tested on MATLAB R12 and R14 installations
  - Tested on Windows and UNIX installations

Operation

- Configure program inputs and constants in the script header
  - sampleFile – (path, file) containing oversampled waveform
  - symbolFile – (path, file) containing corresponding transmitted symbols
  - stressorFile – (path, file) containing oversampled impulse response and noise amplitude of the stressor (may contain multiple stressors)
  - ber0 – target bit error ratio
  - eqNc – number of T/2-spaced feed-forward taps
  - eqNd – number of feedback taps
- Refer to script comments for file formats and acceptable parameter ranges
- Invoke script
  - Graphical output displayed when execution is complete
  - Outputs also available from the command line (snr, heyeISI, heyeBER, veyeISI, veyeBER)

Script

```
%% MATLAB (R) script to compute the MMSE-DFE slicer input eye diagram %%%%%%%%%

%% Version: 1.1
%% Date: September 26, 2007
%% Author: Adam Healey, LSI Corporation

clear variables
verStr = 'v1.1';    % Version string
warning off         % Suppress "divide-by-0" and "log of zero" warnings

%% Define inputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% sampleFile : Contains exactly "oversampling" samples per symbol. The samples
%% must be circularly shifted to align with the symbols in "symbolFile." The
%% file format is a single column of chronological numerical samples, in ASCII
%% format, with no header or footer
%%
%% symbolFile : The file format is a single column of chronological symbols, in
%% ASCII format, with no header or footer
sampleFile      = 'tx_samples.txt'; % Example to be used with *_TX_TCTF.txt
symbolFile      = 'tx_symbols.txt';
%% sampleFile      = 'rx_samples.txt'; % Example to be used with *_RX.txt
%% symbolFile      = 'rx_symbols.txt';
oversampling    = 16;           % samples/symbol, must be an even integer

%% Define constants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Note that T is the symbol period (normalized to 1) and "eqNc" must be 1 or
%% an even integer
ber0            = 1E-12;        % target bit error ratio
maxEqDelay      = 16;           % maximum equalizer delay (symbol periods)
eqNc            = 1;            % number of feed-forward taps (T/2-spaced)
eqNd            = 3;            % number of feedback taps (T-spaced)
dphi            = 1/100;        % eye diagram phase step (symbol periods)
dvee            = 1/200;        % eye diagram amplitude step

graphOut        = 1;            % control graphical outputs
                                % 0 : disable graphical outputs
                                % 1 : bit error ratio map

%% Define stressors %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% stressorFile : File format is a column of values, in ASCII format, for each
%% stressor, with no header or footer. Each column is to be separated by white
%% space. The first row of each column contains the RMS noise amplitude and is
%% followed by chronological numerical samples of impulse response sampled at
%% T/"oversampling" intervals.
stressorFile    = 'BETA_EPSILON_TX_TCTF.txt';   % Beta/Epsilon TCTF set
%% stressorFile    = 'BETA_EPSILON_RX.txt';        % All-pass stressor

h0 = load( stressorFile );
rmsNoise = h0(1, :);
h0 = h0(2:end, :);

%% Load input waveform and symbol sequence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y0 = load( sampleFile );
x0 = load( symbolFile );

numSymbols  = length( x0 );
numPoints   = oversampling*numSymbols;

y0 = y0(1:numPoints);           % Ensure that y0 has the correct length
y0 = y0(:).';                   % Ensure that y0 and x0 are row vectors
x0 = x0(:).';

y0 = y0-mean( y0 );             % Remove any DC offset from the samples
x0 = x0/(max( x0 )-min( x0 ));  % Ensure that the symbols are bipolar and that
x0 = x0-1/2;                    % the minimum distance is 1

e1 = eye( eqNd+1, 1 );          % Define the identity vector, to be used later

%% Compute the frequency response of the anti aliasing filter %%%%%%%%%%%%%%%%%
%% The anti-aliasing filter is a fourth-order Butterworth filter with the -3 dB
%% frequency set to 3/4 of the signaling speed.
freq = (-numPoints/2:numPoints/2-1)/numSymbols;

%% Denominator and numerator polynomials for the frequency response
ar = [1, 12.3141, 75.8181, 273.4537, 493.1335];
```

```
 76 br = 493.1335;
 77
 78 Hr = br./polyval( ar, j*2*pi*freq );
 79
 80 %% Compute the frequency response for the spectral line bandpass filter %%%%%%%%
 81 w1 = 2*pi*(1-1/1667);                       % Define the pass band
 82 w2 = 2*pi*(1+1/1667);
 83
 84 %% Denominator and numerator polynomials for the bandpass filter
 85 ap = [1, w2-w1, w1*w2];
 86 bp = [0, w2-w1, 0];
 87
 88 Hp = polyval( bp, j*2*pi*freq )./polyval( ap, j*2*pi*freq );
 89
 90 %% Compute the equalized eye at the slicer input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 91 if any( graphOut )
 92     figNum = gcf;
 93 end
 94
 95 for ii = 1:size( h0, 2 )        % stressor index
 96     %% Compute the stressor transfer function
 97     Hc = fftshift( fft( h0(:, ii), numPoints ) );
 98     Hc = Hc(:).';               % Ensure Hc is a row vector
 99
100     %% Process the waveform through the stressor and the anti-aliasing filter
101     yr = real( ifft( fft( y0 ).*fftshift( Hc.*Hr ) ) );
102
103     %% Compute the noise auto-correlation sequence %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104     %% The noise at the anti-aliasing filter input is assumed to be white with
105     %% power spectral density that yields the prescribed noise amplitude at the
106     %% filter output. The output is then sampled at T/2 intervals.
107     Rnn = real( ifft( abs( fftshift( Hr ) ).^2 ) );
108     Rnn = (rmsNoise(ii).^2)*Rnn/Rnn(1);
109
110     sampledRnn = Rnn(1:oversampling/2:end);
111     sampledRnn = toeplitz( sampledRnn(1:eqNc) );
112
113     %% Compute the sampling function and sample the processed waveform %%%%%%%%%
114     km1 = mod( (0:numPoints-1)-1, numPoints )+1;
115     kp1 = mod( (0:numPoints-1)+1, numPoints )+1;
116
117     yclk = real( ifft( fft( (yr(kp1)-yr(km1)).^2 ).*fftshift( Hp ) ) );
118     yclk = yclk(kp1)-yclk(km1);
119
120     time = (0:numPoints)/oversampling;  % Wrap waveforms to ensure all edges are
121     yr = [yr, yr(1)];                   % detected
122     yclk = [yclk, yclk(1)];
123
124     kr = find( diff( yclk > 0 ) > 0 );      % Eye center index
125     kf = find( diff( yclk > 0 ) < 0 );      % Eye crossing index
126     k = sort( [kr, kf] );
127     index1 = double( kr(1) > kf(1) )+1;     % Index of the first eye center
128
129     tk = time(k)-(1/oversampling)*yclk(k)./(yclk(k+1)-yclk(k));
130
131     %% Compute the MMSE-DFE coefficients %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
132     yk = interp1( time, yr, mod( tk, time(end) ) );
133     Y = toeplitz( yk, [yk(1), yk(end:-1:end-eqNc+2)] );
134     Y = Y(index1:2:end, :);
135     Ryy = Y'*Y+numSymbols*sampledRnn;
136
137     %% Minimize MSE over the specified equalizer delay range %%%%%%%%%%%%%%%%%%%
138     eqDelayList = (0:maxEqDelay-1);
139
140     for jj = 1:length( eqDelayList )
141         eqDelay = eqDelayList(jj);
142
143         xd = x0(mod( (0:numSymbols-1)-eqDelay, numSymbols )+1);
144         X = toeplitz( xd, [xd(1), xd(end:-1:end-eqNd+1)] );
145         Rxx = X'*X;
146         Ryx = Y'*X;
147
148         lambda(jj) = 1/(e1'*inv( Rxx-Ryx'*inv( Ryy )*Ryx )*e1);
149         D(:, jj) = lambda(jj)*inv( Rxx-Ryx'*inv( Ryy )*Ryx )*e1;
150         C(:, jj) = inv( Ryy )*Ryx*D(:, jj);
```

```
151     end     % for jj = length( eqDelayList )
152
153     %% Install the best equalizer delay %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
154     [bestMSE, bestIndex] = min( lambda );
155     bestEqDelay = eqDelayList(bestIndex);
156
157     xd = x0(mod( (0:numSymbols-1)-bestEqDelay, numSymbols )+1);
158     X = toeplitz( xd, [xd(1), xd(end:-1:end-eqNd+1)] );
159
160     bestC = C(:, bestIndex);
161     bestD = D(2:end, bestIndex);
162
163     noiseVar = bestC'*sampledRnn*bestC;
164
165     %% Generate slicer input eye diagram %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
166     phiList = linspace( -0.5, 0.5, round( 1/dphi )+1 );
167     veeList = linspace( -0.5, 0.5, round( 1/dvee )+1 );
168
169     for jj = 1:length( phiList )
170         phi = phiList(jj);
171
172         yk = interp1( time, yr, mod( tk+phi, time(end) ) );
173         Y = toeplitz( yk, [yk(1), yk(end:-1:end-eqNc+2)] );
174         Y = Y(index1:2:end, :);
175
176         zk = Y*bestC-X*[0; bestD];
177
178         %% Compute the minimum distance from the noiseless, equalized samples
179         %% to the decision threshold
180         eyeLid0(jj) = max( zk(find( xd < 0 )) );
181         eyeLid1(jj) = min( zk(find( xd > 0 )) );
182
183         %% Compute the bit error ratio as a function of offset from the nominal
184         %% sampling time and decision threshold
185         dk = ones( length( veeList ), 1 )*zk.'-veeList(:)*ones( 1, numSymbols );
186         dk(:, find( xd < 0 )) = -dk(:, find( xd < 0 ));
187
188         berMap(:, jj) = mean( erfc( dk/sqrt( 2*noiseVar ) )/2, 2 );
189     end     % for jj = 1:length( phiList )
190
191     %% Normalize the coefficients so that the DC gain of the equalizer is 1 and
192     %% compute the effective target amplitude, gamma
193     gamma2(ii) = 1/sum( bestC );
194     Cn(:, ii) = gamma2(ii)*bestC;
195     Dn(:, ii) = gamma2(ii)*bestD;
196
197     %% Compute the horizontal and vertical eye opening at the slicer input in
198     %% the absense of noise
199     eyeLid = min( [-eyeLid0; eyeLid1] );
200
201     kISI = find( abs( diff( eyeLid > 0 ) ) > 0 );
202     phiISI = phiList(kISI)-dphi*eyeLid(kISI)./(eyeLid(kISI+1)-eyeLid(kISI));
203
204     if length( phiISI ) == 0
205         phiISI = [0, 0];
206     end
207
208     if length( phiISI ) == 1
209         phiISI = sort( [phiISI, -sign( phiISI )/2] );
210     end
211
212     heyeISI(ii) = 2*max( min( [-phiISI(1), phiISI(2)] ), 0 );
213     veyeISI(ii) = 2*gamma2(ii)*max( eyeLid(find( phiList == 0 )), 0 );
214
215     %% Also compute the signal-to-noise ratio for reference
216     snr(ii) = 10*log10( (veyeISI(ii)/gamma2(ii))^2/noiseVar );
217
218     %% Compute the horizontal and vertical eye opening at the slicer input at
219     %% the specified target bit error ratio
220     phiTub = berMap(find( veeList == 0 ), :);
221     veeTub = berMap(:, find( phiList == 0 )).';
222
223     kBER = find( abs( diff( phiTub > ber0 ) ) > 0 );
224     nBER = find( abs( diff( veeTub > ber0 ) ) > 0 );
225
```

```
    phiBER = phiList(kBER)-dphi*phiTub(kBER)./(phiTub(kBER+1)-phiTub(kBER));
    veeBER = veeList(nBER)-dvee*veeTub(nBER)./(veeTub(nBER+1)-veeTub(nBER));

    if (length( phiBER ) == 0) | (length( veeBER ) == 0)
        phiBER = [0, 0];
        veeBER = [0, 0];
    end

    if length( phiBER ) == 1
        phiBER = sort( [phiBER, -sign( phiBER )/2] );
    end

    heyeBER(ii) = 2*max( min( [-phiBER(1), phiBER(2)] ), 0 );
    veyeBER(ii) = 2*gamma2(ii)*max( min( [-veeBER(1), veeBER(2)] ), 0 );

    %% Present graphical output(s) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if any( graphOut == 1 )     % MMSE-DFE slicer input eye diagram
        figure( figNum );
        clf;

        imagesc( phiList, gamma2(ii)*veeList, log10( berMap ) );

        hold on
        plot( phiList, gamma2(ii)*eyeLid0, '--', 'Color', 'white' );
        plot( phiList, gamma2(ii)*eyeLid1, '--', 'Color', 'white' );
        hold off

        jetColors = jet;
        colormap( jet );
        caxis( [round( log10( ber0 ) ), 0] );
        colorbar;
        set( gca, 'YDir', 'normal' );
        set( gca, 'Color', jetColors(end, :) );

        tapStr = [];
        tapStr = [tapStr, sprintf( '\nC = [%.3f', Cn(1, ii) )];
        for jj = 2:eqNc
            tapStr = [tapStr, sprintf( ', %.3f', Cn(jj, ii) )];
        end
        tapStr = [tapStr, ']'];
        if eqNd > 0
            tapStr = [tapStr, sprintf( '\nD = [%.3f', Dn(1, ii) )];
            for jj = 2:eqNd
                tapStr = [tapStr, sprintf( ', %.3f', Dn(jj, ii) )];
            end
            tapStr = [tapStr, ']'];
        else
            tapStr = [tapStr, sprintf( '\nD = []' )];
        end
        tapStr = [tapStr, sprintf( '\n\\gamma = ' )];
        tapStr = [tapStr, sprintf( '%.3f', gamma2(ii)/2 )];

        eyeStr = [];
        eyeStr = [eyeStr, sprintf( 'SNR = %.1f dB\n', snr(ii) )];
        eyeStr = [eyeStr, sprintf( 'HEYE(ISI, 1E%.0f) = ', log10( ber0 ) )];
        eyeStr = [eyeStr, sprintf( '(%.3f, ', heyeISI(ii) )];
        eyeStr = [eyeStr, sprintf( '%.3f)\n', heyeBER(ii) )];
        eyeStr = [eyeStr, sprintf( 'VEYE(ISI, 1E%.0f) = ', log10( ber0 ) )];
        eyeStr = [eyeStr, sprintf( '(%.3f, ', veyeISI(ii) )];
        eyeStr = [eyeStr, sprintf( '%.3f)\n', veyeBER(ii) )];

        titleStr = [];
        titleStr = [titleStr, sprintf( 'DFEEYE %s: ', verStr )];
        titleStr = [titleStr, sprintf( 'Bit error ratio map (%d)', ii )];

        text( -0.45,  gamma2(ii)*0.40, tapStr, 'Color', 'white' );
        text( -0.45, -gamma2(ii)*0.40, eyeStr, 'Color', 'white' );
        title( titleStr );
        ylabel( 'Distance from decision threshold' );
        xlabel( 'Time (UI)' );

        figNum = figNum+1;
    end

end         % for ii = 1:size( h0, 2 )
```

```

warning on              % Restore warnings

%% End of script %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```