Application Note CMX983 Rx Compensation using a Sinc^N filter

Publication: AN/Wireless/983/RxComp_Sinc/2 September 2019

1 Introduction

This document considers a method of compensating for the inherent frequency response of the CMX983 ADC path. It is recommended that readers should familiarise themselves with the CMX983 architecture, as detailed in the device datasheet, prior to considering this Application Note.

2 History

Version	Changes	Date
2.0	Correction to CMX983 sincN response compensation filter script	2019-09-11
1.0	First release	2014-11-27

3 Contents

4 Sinc^N filter compensation

4.1 Sinc filter frequency response

The bit stream at the output of the sigma-delta (∑∆) modulators in the CMX983 receive channel contains quantisation noise which has a characteristic high-pass noise profile. The purpose of the sinc^N receive channel filters is to attenuate this quantisation noise and suitably band limit the signal to prevent aliasing problems in the first downsampler. The length of each sinc filter (1-64) determines the first zero of the sinc^N filter and the number, N, of cascaded stages (3-6) determines the attenuation. Figure 1 shows the frequency response of the four possible user-selectable stages. The frequency axis has been normalised to the (Channel rate (CR1) / sinc Filter Length).

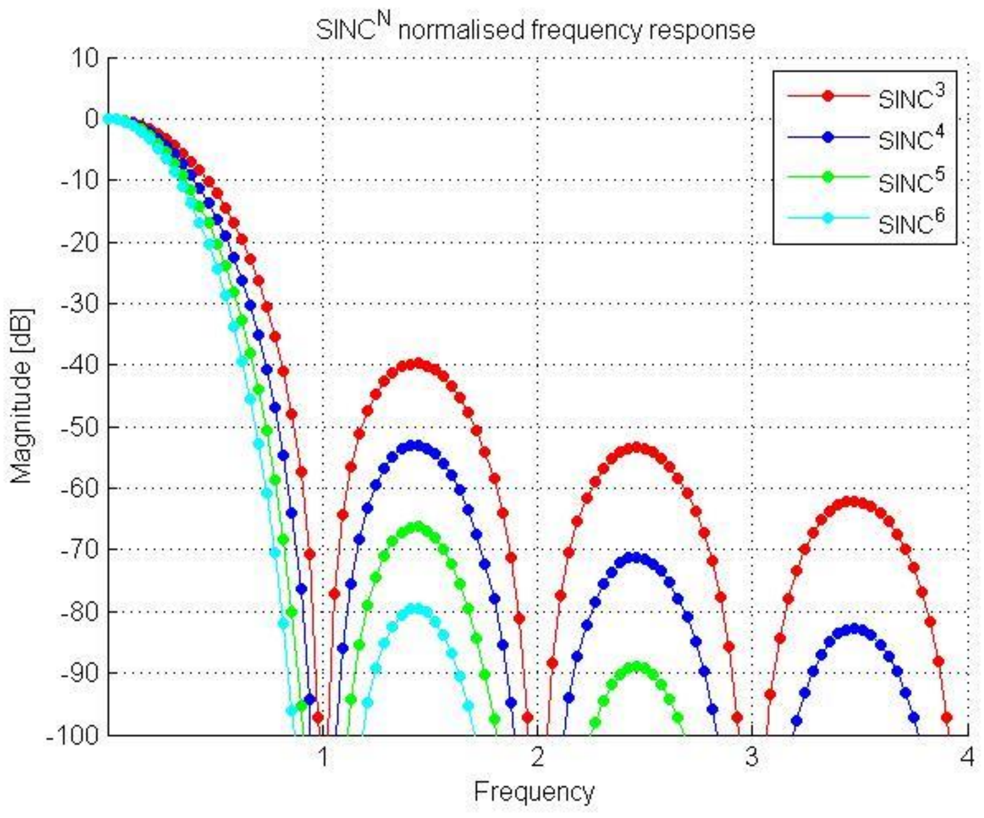

Figure 1 – Normalised frequency response of sinc^N filters

4.2 Inverse sinc filter coefficients

Depending on the ratio of the bandwidth of the wanted signal CR2, the channel rate after the first downsampler, the sinc^N receive channel filters may cause an unwanted droop within the pass-band of the desired signal. This unwanted drop can be compensated by applying a filter with the inverse sinc^N filter using the receive channel FIR filters.

To generate the filter coefficients for a sinc^N compensation filter, CML has provided two scripts in Section 4.4.1 that are compatible with Matlab and GNU Octave respectively.

4.2.1 Using supplied scripts to generate compensation filter coefficients

To use the scripts, copy and paste the text from the end of this application note and save the two scripts in separate files with the following file names: *sinc_N_comp.m*, *output_16Bit_coeffs.m.*

The files are functions designed to be run in the command window within Matlab and GNU Octave.

The first script, *sinc_N_comp.m*, generates FIR filter coefficients to compensate a sinc^N filter response in the CMX983 receive channel. It will return a vector, *b*, containing double precision floating point filter coefficients. The script is run from the command line using the following function call:

```
[b] =  sinc_N_comp( SincLen, NStages, CR2, Fc, M, TwM )

        SincLen - length of the sinc filters
        NStages - Number of sinc stages [3,4,5,6]
        CR2 - Channel Rate 2 [Hz]
        Fc - Pass band edge frequency [Hz]
        M - Filter order,  (must be even)
        TwM - Twiddle factor to increase gain near Fc [1 or greater]
```

The second script, output_16Bit_coeffs.m, is used to convert the filter coefficients to signed integers with 16-bit precision for use with the CMX983 GUI application and as 2's complement filter coefficients in hexadecimal format. These coefficients need to be loaded into one of the four programmable coefficient banks in the A and B channel filters. The script is run from the command line using the following function call:

```
[ b16Bit ] = output_16Bit_coeffs( b, '16Bit_Coeffs.h', 'Hex_Coeffs.txt' );

        b - a vector of floating point filer coefficients
        16Bit_Coeffs.h - optional file name to write 16 Bit signed
                         integer coefficients.
        Hex_Coeffs.txt - optional file name to write 2's complement
                         filter coefficients in hexadecimal format
```

4.3 Test Results

To demonstrate the improvement in received signal quality that can be achieved, 9.6 kbaud baseband 4,16 & 64 -QAM signals with root raised cosine filtering (α = 0.2) were used as inputs to the CMX983 and a compensation filter was applied to these signals. For this test the ADC sampling rate (CR1) was 1.2 MHz and a 6-stage sinc^N filter of length 25 was used to filter the incoming signals. The channel rate after the first downsampler (CR2) was 48 kHz. The filter coefficients were generated using the two provided scripts using the following commands:

```
[b] =  sinc_N_comp( 26, 6, 48000, 4800, 50, 1 );
[ b16Bit ] = output_16Bit_coeffs( b, '16Bit_Coeffs.h', 'Hex_Coeffs.txt' );
```

The resulting filter coefficients were then loaded into the receive channel A and B FIR filters using the CMX983 GUI application.

The following figures demonstrate the benefit of compensating the pass-band droop introduced by the sinc^N filters. In all three cases the received I and Q eye diagrams show an improved eye opening when the sinc^N compensation filter is applied.

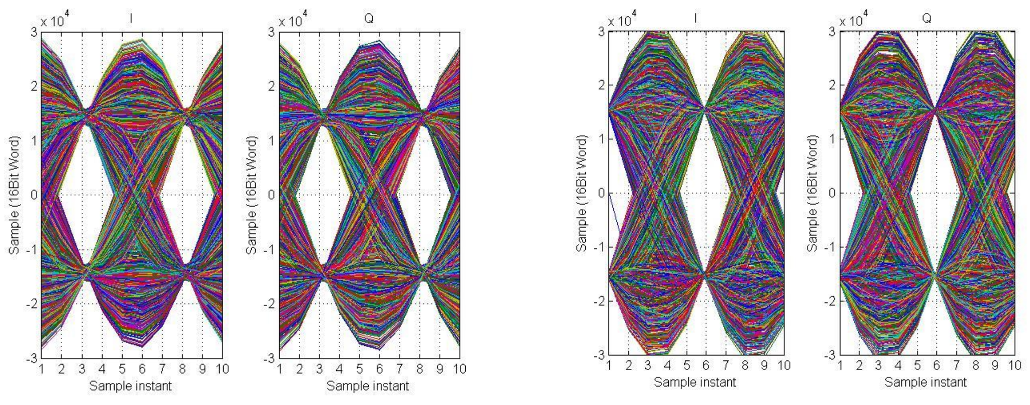

Figure 2- 4-QAM – no sinc^N compensation (left), with sinc^N compensation (right)

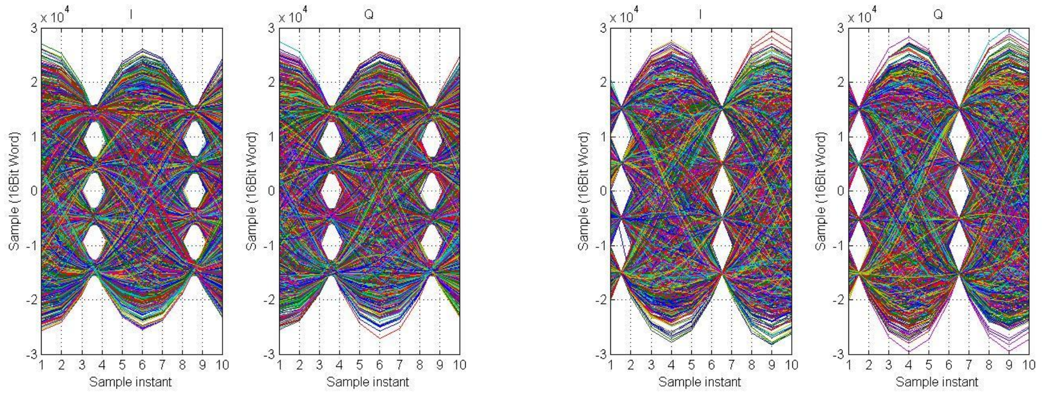

Figure 3 - 16-QAM – no sinc^N compensation (left), with sinc^N compensation (right)

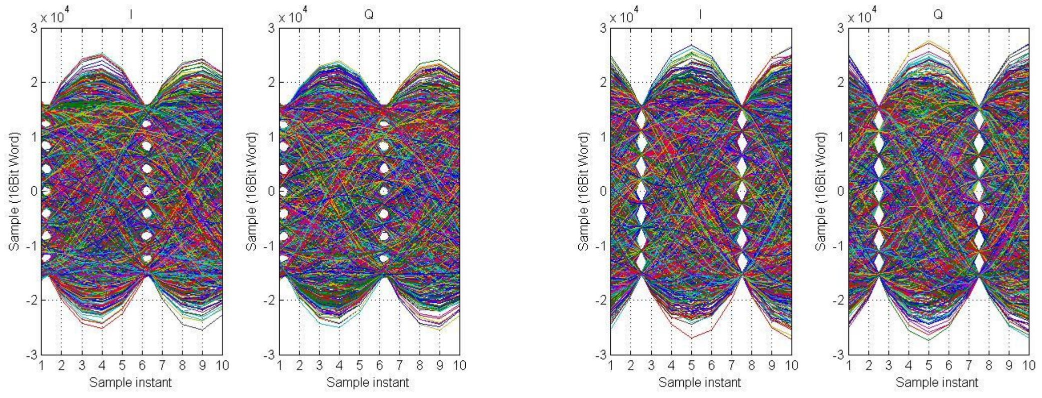

Figure 4 - 64-QAM – no sinc^N compensation (left), with sinc^N compensation (right)

4.4 Matlab and Octave script files

4.4.1 CMX983 sinc^N response compensation filter script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Sinc compensation Filter - Matlab / Octave Script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [b] = sinc_N_comp( SincLen, NStages, CR2, Fc, M, TwM)
% sinc_N_comp   Generates FIR filter coefficients to compensate for
%   sinc^N filter response in the CMX983 receive channel. A truncated
%   frequency response, from DC to the first null (CR1 / Sinc Length) is
%   generated and then inverted to compensate for the droop introduced by
%   the sinc filter. The  M+1 filter coefficients for an FIR Filter
%   are then generated using the frequency sampling method.
%
%   [b] = sinc_N_comp( SincLen, NStages, CR2, Fc, M, TwM )
%
%   RETURNS:
%   Returns the filter coefficients in length M+1 vectors:
%   b - filter coefficients as double precision floating point
%
%   INPUTS:
%   SincLen - length of the sinc filters
%   NStages - Number of sinc stages [3,4,5,6]
%   CR2 - Channel Rate 2 [Hz]
%   Fc - Pass band edge frequency [Hz]
%   M - Filter order,  must be even.
%   TwM - Twiddle factor to increase gain near Fc [1 or greater]

% Error if Fc is greater than Nyquist frequency
if (Fc > CR2/(2))
    error('Passband cut-off frequency greater than Nyquist frequency : (CR2/2)');
end

%% Code
% Number of points in frequency grid.
Npts = 256;

D = SincLen;

% Normalised frequency grid [0 : 1]*(CR1 / SincLen)
% which is used to calculate frequency response of SINC^N.
FF = [0:Npts]./(Npts*D);

% Frequency response of SINC^N filter, 0 Hz -> first null (CR1/Sinc-Length)
Hf = 1/D*abs(sin(pi*FF*D)./sin(pi*FF));
Hf(1) = 1;
Hf = Hf.^NStages;                    % sinc.^N

HfInv = 1./Hf;                       % Inverse Frequency Response

HfInv = HfInv.^TwM;

FN = FF./(max(abs(FF)));             % Normalised frequency grid [0 : 1]*CR2
Fo = Fc/CR2;                         % Normalised passband cut-off frequency

FoInd = find(FN>=Fo);                % Location of Fc on normalised frequency grid

stop = zeros(1,1+Npts/2);
HfInvComp = [HfInv(1:min(FoInd)), stop(min(FoInd)+1:end)];

% Normalised frequency grid for FIR filter, [0 : 0.5]*CR2
FN_FIR = FN(1:1+Npts/2);

% Normalised frequency grid need for FIR2 function, [0:1]*CR2/2
FN_mat = FN_FIR*2;

% Window for the truncated frequency response
win = hamming(M+1);
%   Other available windows, including Boxcar, Hann, Bartlett, Blackman,
%   Kaiser and Chebwin can be specified

% M+1 FIR filter coefficients with 0 dB gain at DC
b = fir2(M, FN_mat, HfInvComp, win);

[hCompF,w] = freqz(b,1, length(FN_mat));

% Normalise so that the sum of the coefficients is 1
b = b./sum(b);

% Plot the frequency response
figure(1);
plot(CR2*w/(2*pi)*1e-3,20*log10(abs(hCompF)));
grid on;
title(['SINC^',num2str(NStages),' compensation filter']);
ylabel('Magnitude [dB]')
xlabel('Frequency [kHz]')
xlim([0 CR2*1e-3/2])

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4.4.2 CMX983 output coefficients script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output coefficients with 16 Bit precision - Matlab / Octave Script
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ b16Bit ] = output_16Bit_coeffs( b, varargin )
%[ b16Bit ] = output_16Bit_coeffs( b, varargin )
%
% Takes a vector, b, of floating point coefficients and returns a
% vector of 16 bit precision coefficients suitable for loading into
% the CMX983 receive channel filters.
%
% This function can take 2 file names as optional arguments.
%
% INPUTS
% b - a vector of floating point filer coefficients
% 16Bit_Coeffs.h - optional file name to write 16 Bit signed integer
%                  coefficients.
% Hex_Coeffs.txt - optional file name to write 2's complement filter
%                  coefficients in hexadecimal format.
%
% RETURNS
% b16Bit - vector of 16 Bit signed integer coefficients
%
% The first of which will be the filename into which the 16 bit
% filter coefficients will be written. For example:
% [ b16Bit ] = output_coeffs( b, '16Bit_Coeffs.h' )
%
% If a second file name is present, hexadecimal filter coefficients will
% be written to the file. For example:
% [ b16Bit ] = output_coeffs( b, '16Bit_Coeffs.h', 'Hex_Coeffs.txt' )


output16Bit = 0;
output16Hex = 0;
if (nargin == 2)
    output16Bit = 1;
    fileName16 = varargin{1};
end

if (nargin == 3)
    output16Hex = 1;
    output16Bit = 1;
    fileName16 = varargin{1};
    fileNameHex = varargin{2};
end

%% Generate fixed point filter coefficients
% Ensure that sum of the coefficients is a power of 2. This will allow integer
% shifts in ADC bit selector(s) to achieve unity gain in passband
bScaled = b./(2.^fix(log2(sum(b)))./sum(b);

while (max(abs(bScaled))<=2^14);
    bScaled = bScaled.*2;
end

b16Bit = fix(bScaled);

%% 2's Complement filter coefficients
b2Comp = b16Bit;
% Index of negative filter coefficients
negInd = find(b16Bit<0);
b2Comp(negInd) = b2Comp(negInd) + 2^16;

%% Coefficients in hexadecimal format
bHex = cell(length(b),1);
for k = 1:length(b2Comp)
    bHex(k,:) = cellstr(['$',dec2hex(b2Comp(k), 4)]);
end

%% Write the coefficients to the output file
if (output16Bit == 1)
```

```
    fid = fopen(fileName16, 'w');
    fprintf(fid, '/*\nFIR Filter\n fixed point precision: 16 bits\n*/\n' );
    fprintf(fid, '#define FILTER_TAP_NUM %d\n', length(b16Bit));
    fprintf(fid, 'static int filter_taps[FILTER_TAP_NUM] = {\n');
    for k = 1:length(b16Bit)-1
        fprintf(fid, '%d,\n', b16Bit(k));
    end
    fprintf(fid, '%d\n};\n', b16Bit(k+1));
    fclose(fid);
end

if (output16Hex == 1)
    fid = fopen(fileNameHex, 'w');
    for k = 1:length(bHex)-1
        fprintf(fid, '%s\n', bHex{k});
    end
    fclose(fid);
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

CML does not assume any responsibility for the use of any algorithms, methods or circuitry described. No IPR or circuit patent licenses are implied. CML reserves the right at any time without notice to change the said algorithms, methods and circuitry and this product specification. CML has a policy of testing every product shipped using calibrated test equipment to ensure compliance with this product specification. Specific testing of all circuit parameters is not necessarily performed.

CML Microcircuits

United Kingdom	p: +44 (0) 1621 875500	e: sales@cmlmicro.com techsupport@cmlmicro.com
Singapore	p: +65 62888129	e: sg.sales@cmlmicro.com sg.techsupport@cmlmicro.com
United States	p: +1 336 744 5050 800 638 5577	e: us.sales@cmlmicro.com us.techsupport@cmlmicro.com

www.cmlmicro.com