DT0092
Design tip

Lattice wave digital filter test and performance verification

By Andrea Vitali

Main components	
STM32L476xx STM32L486xx	Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100 DMIPS, up to 1 Mbyte Flash, 128 Kbytes SRAM, USB OTG FS, LCD, analog, audio
STM32F411xx	Arm® Cortex®-M4 32-bit MCU+FPU, 125 DMIPS, 512 Kbytes Flash,128 Kbytes RAM, USB OTG FS, 11 TIMs, 1 ADC, 13 comm. interfaces

Purpose and benefits

This design tip explains how to test and verify the performance of lattice wave digital filters (LWDF) when the C source code implementation is available. In particular, the following procedures will be described:

- How to compute the frequency response from the gamma coefficients
- How to compute the frequency response from the impulse response
- How to verify the frequency response by filtering white noise
- How to verify the performance and stability of the filter implementation
- How to verify the performance of the interpolation/decimation implementation

Description

Wave digital filters have some notable advantages: excellent stability under non-linear operating conditions due to overflows and round-off errors, low coefficient word-length and good dynamic range.

An important subclass of lattice wave digital filters is made of filters with a bi-reciprocal transfer function: H(f) = 1-H(F/2-f), where F is the sampling frequency and f goes from 0 to F/2. Because of the symmetry, H(F/4)=0.5 (-3dB), the pass-band attenuation is dependent on the stop-band attenuation and cannot be specified.

Bi-reciprocal filters have half as many adaptors as the usual lattice wave digital filter of the same order. Also, interpolation or decimation with a factor of two is very economical: the upper and lower branches work at the lowest frequency. For decimation, the input is distributed in a round-robin fashion to halve the frequency. For interpolation, the output is concatenated to double the frequency.

The companion design tip, DT0091, provides the design tool needed to generate the list of gamma coefficient and the C source code implementation.

How to compute the frequency response from the gamma coefficients

The following MATLAB® script imports the file with the gamma coefficients ("WDgamma.txt") and computes the frequency response of the filter, magnitude and phase. The group delay is also plotted.

```
NFFT=2^13; % number of points for frequency response
Fs=input('sampling frequency Hz? '); % T=1/Fs sampling interval
data=importdata('WDgamma.txt'); % filename as specified in WDdesign.c
gamma=data.data;  N=length(gamma);
if mod(N,2)==0, fprintf('order must be odd\n'); return; end;

%---- frequency response for the upper and lower branch
gamma0=gamma(1); gamma1=gamma(2:end); i=1:(N-1)/2;
B0=gamma0; Ai=gamma1(2*i-1); Bi=gamma1(2*i).*(1-gamma1(2*i-1));
f=linspace(0,Fs/2,NFFT); z=exp(1i*2*pi*f/Fs); % w=2pif, s=jw, z=exp(jwT)
z1=z.^-1; z2=z.^-2; Hup=(-B0+z1)./(1-B0*z1); Hlow=ones(1,NFFT);
for i=1:(N-1)/2,
  H=(-Ai(i)-Bi(i)*z1+z2)./(1-Bi(i)*z1-Ai(i)*z2);
  if mod(i,2)==0, Hup=Hup.*H; else Hlow=Hlow.*H; end;
end;
hLP=(Hlow+Hup)/2; hHP=(Hlow-Hup)/2; % complementary output
gLP=-diff(unwrap(angle(hLP)))./(pi/NFFT); % group delay as derivative of phase
gHP=-diff(unwrap(angle(hHP)))./(pi/NFFT); % with respect to dw, w=2pif

%---- plot
figure; subplot(2,1,1); hold on;
plot(f,20*log10(abs(hLP)),'b'); plot(f,20*log10(abs(hHP)),'k');
xlabel('frequency Hz'); ylabel('dB'); title('magnitude');
legend('lowpass','highpass'); grid on;
subplot(2,1,2); hold on; plot(f,angle(hLP),'b'); plot(f,angle(hHP),'k');
xlabel('frequency Hz'); ylabel('rad'); title('phase');
legend('lowpass','highpass'); grid on;

figure; subplot(2,1,1);
plot(f(1:end-1),gLP,'b'); axis([0 max(f) -100 +100]); grid on;
xlabel('frequency Hz'); ylabel('delay T=1/Fs'); title('lowpass group delay');
subplot(2,1,2); plot(f(1:end-1),gHP,'k'); axis([0 max(f) -100 +100]); grid on;
xlabel('frequency Hz'); ylabel('delay T=1/Fs'); title('highpass group delay');
```

How to compute and verify the frequency response using an impulse and noise

The following MATLAB® script probes the filter with an impulse. The output of the filter is then used to compute the frequency response. The script also probes the filter with white noise and the power spectrum of the output is plotted over the frequency response magnitude to verify that they match.

```
fprintf('LWDF test to find frequency response\n');
NFFT=2^13; % FFT resolution
fflag=input('test type (0=normal, +/-1=interp/decim)? ');
fflag=abs(fflag); % test decimator as if it is an interpolator
iflag=input('I/O type (0=float, 1=integer)? ');
if(iflag==0) A=input('test signal max amplitude? ');
else A=input('test signal bits (max: half of integer bits - 1)? '); A=2^A;
end;
Fs=input('sampling frequency Hz? ');

%---- impulse response to frequency response and group delay
in=zeros(1,NFFT); in(1)=A;
[outLP,outHP]=WDtest(in,iflag,fflag);
[hLP,hf]=freqz(outLP,1,NFFT,Fs); [gLP,hf]=grpdelay(outLP,1,NFFT,Fs);
[hHP,hf]=freqz(outHP,1,NFFT,Fs); [gHP,hf]=grpdelay(outHP,1,NFFT,Fs);

%---- white noise to power spectrum (frequency response)
NOVL=round(NFFT*0.5); win=hann(NFFT);
in=((2*rand(1,NFFT)-1)*A); if(iflag) in=round(in); end;
[outLP,outHP]=WDtest(in,iflag,fflag);
[pLP,pf]=pwelch(outLP,win,NOVL,NFFT,Fs);
[pHP,pf]=pwelch(outHP,win,NOVL,NFFT,Fs);

%---- plot
figure; subplot(2,1,1); hold on;
plot(hf,20*log10(abs(hLP))-20*log10(A)-fflag*6,'b');
plot(hf,20*log10(abs(hHP))-20*log10(A)-fflag*6,'k');
plot(pf,10*log10(abs(pLP))-20*log10(A)+10*log10(Fs),'b:');
plot(pf,10*log10(abs(pHP))-20*log10(A)+10*log10(Fs),'k:');
```

```
if (fflag) xlabel('frequency Hz (before decim or after interp)');
else xlabel('frequency Hz'); end;
ylabel('dB'); title('magnitude');
legend('lowpass','highpass','lowpass noise','highpass noise'); grid on;
subplot(2,1,2); hold on; plot(hf,angle(hLP),'b'); plot(hf,angle(hHP),'k');
xlabel('frequency Hz'); ylabel('rad'); title('phase');
legend('lowpass','highpass'); grid on;

figure; subplot(2,1,1); plot(hf,gLP,'b'); axis([0 max(hf) -100 +100]); grid on;
xlabel('frequency Hz'); ylabel('delay T=1/Fs'); title('lowpass group delay');
subplot(2,1,2); plot(hf,gHP,'k'); axis([0 max(hf) -100 +100]); grid on;
xlabel('frequency Hz'); ylabel('delay T=1/Fs'); title('highpass group delay');
```

The script uses the following MATLAB® function which in turn calls the C utility shown below.

```
function [outLP,outHP] = WDtest(in,iflag,fflag)
  if(iflag), fmt='%d\n'; cmd='WDtest_int';   in=round(in); % integer input and test
  else       fmt='%f\n'; cmd='WDtest_float';                % float   input and test
  end;
  if(fflag==-1), in1=in(1:2:end); in2=in(2:2:end); % decimator
  else           in1=in;          in2=in;          % normal or interpolator
  end;
  h=fopen('WDin1.txt','w'); fprintf(h,fmt,in1); fclose(h);
  h=fopen('WDin2.txt','w'); fprintf(h,fmt,in2); fclose(h);
  system(sprintf('%s WDin1.txt WDin2.txt WDout1.txt WDout2.txt',cmd));
  out1=importdata('WDout1.txt'); L1=length(out1);
  out2=importdata('WDout2.txt'); L2=length(out2);
  if(L1~=L2), fprintf('error!\n'); return; end;
  if(fflag==1), % interpolator
    outLP(1:2:L1+L2)=out1; outLP(2:2:L1+L2)=+out2;
    outHP(1:2:L1+L2)=out1; outHP(2:2:L1+L2)=-out2;
  else % normal or decimator
    outLP=(out1+out2)/2; outHP=(out1-out2)/2;
  end;
end
```

The C utility includes the source code of the filter implementation generated by the design tool (“WDfilter.c”). The utility reads two input files, which are then fed to the inputs of the lattice wave digital filter, and writes two output files taken from the outputs of the filter.

If the filter has a floating point implementation, this is the utility to be compiled:

```
#include <stdio.h>
#include <stdlib.h>
#include "WDfilter.c"
#define CNTMAX 1048576 // safety
int main(int argc, char *argv[]) {
  FILE *fin1, *fin2, *fout1, *fout2;
  float gain, ofs, in1, in2, out1, out2, cnt;
  for(fin1=fin2=fout1=fout2=NULL;;) {
    if (argc<=4) { printf("usage: %s infile1 infile2 outfile1 outfile2 [offset [gain]]\n\n",argv[0]); break; }
    if (argc>5)  ofs=atof(argv[5]); else ofs=0.0;   printf("offset subtracted (before gain) = %f\n",ofs);
    if (argc>6) gain=atof(argv[6]); else gain=1.0;  printf("gain (mul if >0, div if <0) = %f\n",gain);
    if (NULL==( fin1=fopen(argv[1],"r"))) { printf("cannot read  %s\n\n",argv[1]); break; }
    else printf("input file1: %s\n",argv[1]);
    if (NULL==( fin2=fopen(argv[2],"r"))) { printf("cannot read  %s\n\n",argv[2]); break; }
    else printf("input file2: %s\n",argv[2]);
    if (NULL==(fout1=fopen(argv[3],"w"))) { printf("cannot write %s\n\n",argv[3]); break; }
    else printf("output file: %s\n",argv[3]);
    if (NULL==(fout2=fopen(argv[4],"w"))) { printf("cannot write %s\n\n",argv[4]); break; }
    else printf("output file: %s\n",argv[4]);
    printf("float input and output\n");
    for(cnt=0;(!feof(fin1))&&(!feof(fin2))&&(cnt<CNTMAX);cnt++) {
      fscanf(fin1,"%f",&in1); fscanf(fin2,"%f",&in2);
      in1=in1-ofs; if (gain>=0.0) in1=in1*gain; else in1=-in1/gain;
      in2=in2-ofs; if (gain>=0.0) in2=in2*gain; else in2=-in2/gain;
      filter(in1,in2,&out1,&out2);
      fprintf(fout1,"%f\n",out1); fprintf(fout2,"%f\n",out2);
    } break; }
  if (fin1 !=NULL) fclose(fin1 ); if (fin2 !=NULL) fclose(fin2 );
  if (fout1!=NULL) fclose(fout1); if (fout2!=NULL) fclose(fout2);
  return 0;
}
```

If the filter has a fixed-point implementation, this is the utility to be compiled:

```
#include <stdio.h>
#include <stdlib.h>
#include "WDfilter.c"
#define CNTMAX 1048576 // safety
int main(int argc, char *argv[]) {
  FILE *fin1, *fin2, *fout1, *fout2;
  int gain, ofs, in1, in2, out1, out2, cnt;
  for(fin1=fin2=fout1=fout2=NULL;;) {
```

```
    if (argc<=4) { printf("usage: %s infile1 infile2 outfile1 outfile2 [offset [gain]]\n\n",argv[0]); break; }
    if (argc>5)  ofs=atoi(argv[5]); else ofs=0;   printf("offset subtracted (before gain) = %d\n",ofs);
    if (argc>6) gain=atoi(argv[6]); else gain=1;  printf("gain (mul if >0, div if <0) = %d\n",gain);
    if (NULL==( fin1=fopen(argv[1],"r"))) { printf("cannot read  %s\n\n",argv[1]); break; }
    else printf("input file1: %s\n",argv[1]);
    if (NULL==( fin2=fopen(argv[2],"r"))) { printf("cannot read  %s\n\n",argv[2]); break; }
    else printf("input file2: %s\n",argv[2]);
    if (NULL==(fout1=fopen(argv[3],"w"))) { printf("cannot write %s\n\n",argv[3]); break; }
    else printf("output file: %s\n",argv[3]);
    if (NULL==(fout2=fopen(argv[4],"w"))) { printf("cannot write %s\n\n",argv[4]); break; }
    else printf("output file: %s\n",argv[4]);
    printf("integer input and output\n");
    for(cnt=0;(!feof(fin1))&&(!feof(fin2))&&(cnt<CNTMAX);cnt++) {
      fscanf(fin1,"%d",&in1); fscanf(fin2,"%d",&in2);
      in1=in1-ofs; if (gain>=0) in1=in1*gain; else in1=-in1/gain;
      in2=in2-ofs; if (gain>=0) in2=in2*gain; else in2=-in2/gain;
      filter(in1,in2,&out1,&out2);
      fprintf(fout1,"%d\n",out1); fprintf(fout2,"%d\n",out2);
    } break; }
  if (fin1 !=NULL) fclose(fin1 ); if (fin2 !=NULL) fclose(fin2 );
  if (fout1!=NULL) fclose(fout1); if (fout2!=NULL) fclose(fout2);
  return 0;
}
```

How to verify the performance and stability of the filter implementation

The following MATLAB® script uses sinusoids in the pass-band and in the stop-band to verify the pass-band ripples, the stop-band attenuation, the recovery from hard discontinuities, the magnitude of over/under-shoots following a step discontinuity, and the return to zero-output when the input goes to zero.

```
fprintf('LWDF test stability\n');
fflag=input('test type (0=normal, +/-1=interp/decim)? ');
iflag=input('I/O type (0=float, 1=integer)? ');
if(iflag==0) A=input('test signal max amplitude? ');
else A=input('test signal bits (max: half of integer bits - 1)? '); A=2^A;
end;
Fs=input('sampling frequency Hz? ');
f1=input(sprintf('test frequency in passband (0 to %.0f Hz)? ',Fs/2));
f2=input(sprintf('test frequency in stopband (0 to %.0f Hz)? ',Fs/2));
f3=min([0.02*Fs/2,f1,f2])/2; % low frequency for step test
L=4*fix(Fs/f3); % test length is 4 cycles of lowest frequency

%---- test
inf1=A*sin(2*pi*f1*[0:L-1]/Fs);       % passband ripples
inf1(L/4:L/2)=-A;                     % recovery from hard discontinuity
inf2=A*sin(2*pi*f2*[0:L-1]/Fs);       % stopband attenuation
ins =A*sign(sin(2*pi*f3*[0:L-1]/Fs)); % step response over/undershoots
inz =zeros(size(ins));                % zero-in to zero-out
in=[inf1 inf2 ins inz];
[outLP,outHP]=WDtest(in,iflag,fflag);

%---- plot
figure;
subplot(3,1,1); plot(in); axis tight; grid on;
xlabel('sample'); ylabel('NADC'); title('input signal');
subplot(3,1,2); plot(outLP); axis tight; grid on;
xlabel('sample'); ylabel('NADC'); title('output lowpass');
subplot(3,1,3); plot(outHP); axis tight; grid on;
xlabel('sample'); ylabel('NADC'); title('ouput highpass');
```

How to verify the performance of interpolation/decimation

The following MATLAB® script uses a sinusoid to verify the interpolation by a factor of two. Interpolation is usually done by inserting one zero after every other sample and then filtering. In the frequency domain, both the low and high frequency interpolated signals are plotted. In the time domain, the low frequency interpolated signal is plotted over the original signal to verify that they match.

```
fprintf('LWDF test for 1:2 interpolation (must be designed for this)\n');
NFFT=2^13; % FFT resolution
fflag=+1; % interpolator
iflag=input('I/O type (0=float, 1=integer)? ');
if(iflag==0) A=input('test signal max amplitude? ');
else A=input('test signal bits (max: half of integer bits - 1)? '); A=2^A;
end;
Fs=input('sampling frequency Hz? ');
f=input(sprintf('test frequency (0 to %.0f Hz)? ',Fs/2));
```

```
%---- test
in=A*sin(2*pi*f*[0:NFFT-1]/Fs);
[outLP,outHP]=WDtest(in,iflag,fflag);
NOVL=round(NFFT*0.5); win=hann(NFFT);
[pLP,pf]=pwelch(outLP,win,NOVL,NFFT,Fs*2);
[pHP,pf]=pwelch(outHP,win,NOVL,NFFT,Fs*2);

%---- plot
figure; subplot(2,1,1); hold on;
plot(pf,10*log10(abs(pLP))-20*log10(A),'b');
plot(pf,10*log10(abs(pHP))-20*log10(A),'k');
xlabel('frequency Hz'); ylabel('dB'); title('frequency domain');
legend('lowpass interp','highpass interp'); axis tight; grid on;
subplot(2,1,2); hold on; L=fix(Fs/f)*4; L=min(L,NFFT);
plot([0:L-1],in(1:L),'b.-'); plot([0:L*2-1]/2,outLP(1:L*2),'k.-');
xlabel('sample'); ylabel('NADC'); title('time domain');
legend('input','lowpass interp'); axis tight; grid on;
```

The following MATLAB® script uses a sinusoid to verify the decimation by a factor of two. Decimation is usually done by filtering and then dropping every other sample. In the frequency domain, both the low and high frequency folded decimated signals are plotted. In the time domain, the low frequency decimated signal is plotted over the original signal to verify that they match.

```
fprintf('LWDF test for 2:1 decimation (must be designed for this)\n');
NFFT=2^13; % FFT resolution
fflag=-1; % decimator
iflag=input('I/O type (0=float, 1=integer)? ');
if(iflag==0) A=input('test signal max amplitude? ');
else A=input('test signal bits (max: half of integer bits - 1)? '); A=2^A;
end;
Fs=input('sampling frequency Hz? ');
f1=input(sprintf('test frequency (0 to %.0f Hz, will not be folded)? ',Fs/2/2));
f2=input(sprintf('test frequency (%.0f to %.0f Hz, will be folded)? ',Fs/2/2,Fs/2));

%---- test
in=A*(sin(2*pi*f1*[0:2*NFFT-1]/Fs)+sin(2*pi*f2*[0:2*NFFT-1]/Fs))/2;
[outLP,outHP]=WDtest(in,iflag,fflag);
NOVL=round(NFFT*0.5); win=hann(NFFT);
[pLP,pf]=pwelch(outLP,win,NOVL,NFFT,Fs/2);
[pHP,pf]=pwelch(outHP,win,NOVL,NFFT,Fs/2);

%---- plot
figure; subplot(2,1,1); hold on;
plot(pf,10*log10(abs(pLP))-20*log10(A),'b');
plot(pf,10*log10(abs(pHP))-20*log10(A),'k');
xlabel('frequency Hz'); ylabel('dB'); title('frequency domain');
legend('lowpass decim','highpass decim'); axis tight; grid on;
subplot(2,1,2); hold on; L=fix(Fs/f1)*4; L=min(L,NFFT);
plot([0:L-1],in(1:L),'b.-'); plot([0:L/2-1]*2,outLP(1:L/2),'k.-');
xlabel('sample'); ylabel('NADC'); title('time domain');
legend('input','lowpass decim'); axis tight; grid on;
```

Examples

This is the floating-point implementation for the Elliptic/Cauer filter, order 7:

```
// Elliptic/Cauer order 7
// stopband min attenuation as=76 dB at fs=0.56% (100%=F/2)
// passband attenuation spread ap=0.14 dB at fp=0.42% (100%=F/2)
void filter(float i1, float i2, float *o1, float *o2) {
  float t0, x0, x1;
  static float T0, T1, T2, T3, T4, T5, T6;

  // filter input: i1=i2=sample(n)

  //**** Upper arm ****
  t0 =i2 -T0 ; T0 =  0.487102*t0    +T0 ; x1 =T0 -t0 ; // adaptor  0: g=+0.512898
  t0 =T4 -T3 ; x0 =  0.334236*t0    +T4 ; T4 =x0 -t0 ; // adaptor  4: g=+0.334236
  t0 =x0 -x1 ; T3 =  0.331276*t0    -x0 ; *o2=T3 -t0 ; // adaptor  3: g=-0.668724

  //**** Lower arm ****
  t0 =T1 -T2 ; T2 =  0.392293*t0    +T2 ; x0 =T2 -t0 ; // adaptor  2: g=+0.607707
  t0 =i1 -x0 ; x1 =  0.404406*t0    -x0 ; T1 =x1 -t0 ; // adaptor  1: g=-0.404406
  t0 =T6 -T5 ; x0 =  0.206694*t0    +T6 ; T6 =x0 -t0 ; // adaptor  6: g=+0.206694
  t0 =x0 -x1 ; T5 =  0.103866*t0    -x0 ; *o1=T5 -t0 ; // adaptor  5: g=-0.896134

  // filter output: sample(n)=(o1+/-o2)/2 for lowpass/highpass
}
```

The output of the tests are in Figures 1-3: the recovery after a hard discontinuity is verified, over/undershoots after each step discontinuity are clearly visible, the output goes to zero after the input goes to zero.

Figure 1. Frequency response

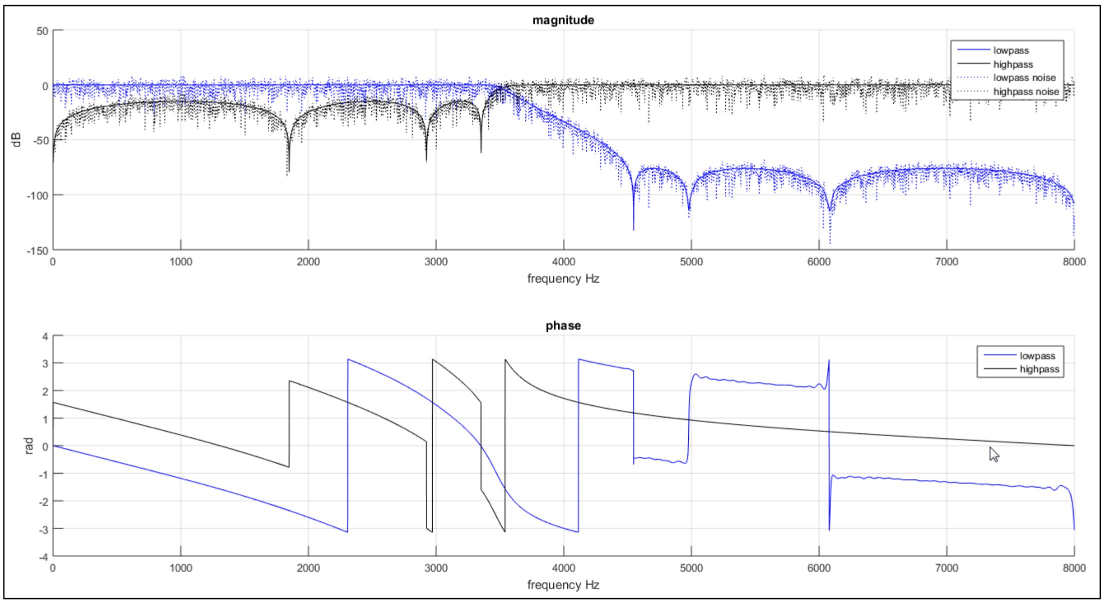


Figure 2. Group delay

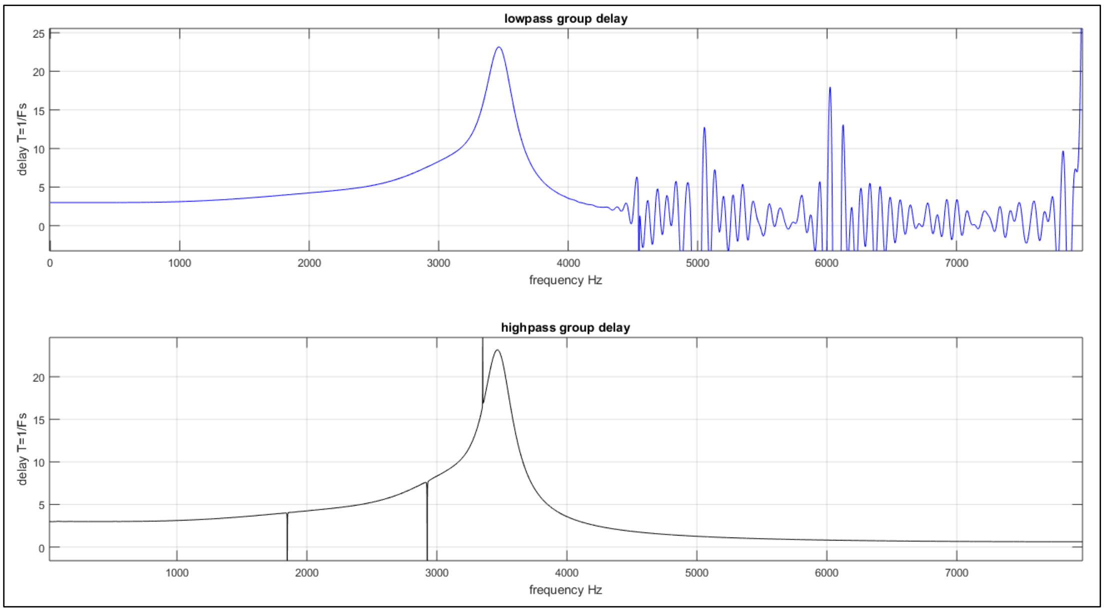
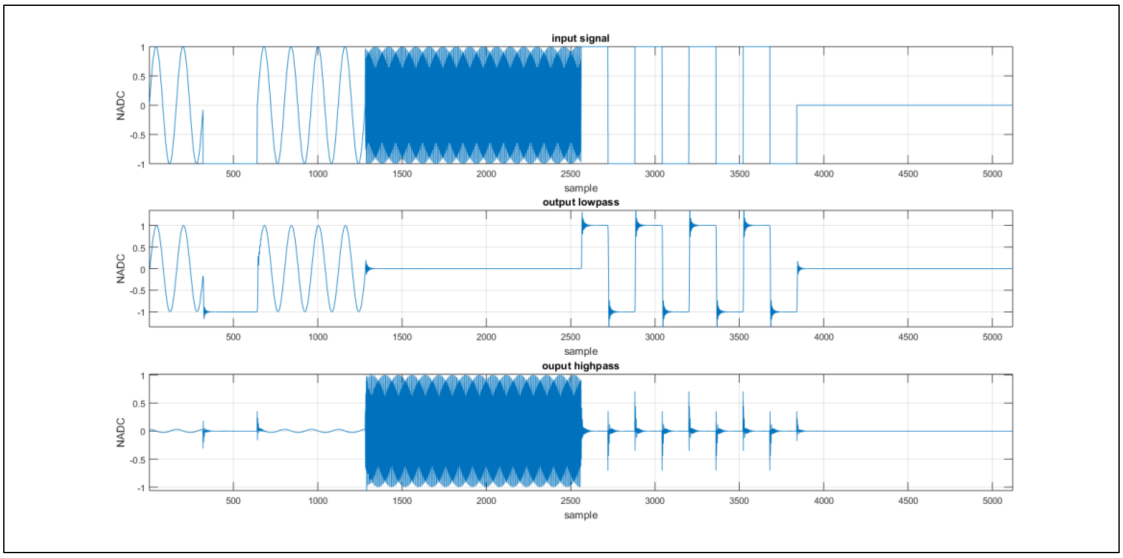


Figure 3. Stability check for the Elliptic/Cauer filter, order 7.



This is the floating-point implementation for the bi-reciprocal Elliptic/Cauer filter, order 19, designed for interpolation/decimation:

```
// Elliptic/Cauer order 19, bireciprocal, for interpolation/decimation
// stopband min attenuation as=77 dB at fs=0.51% (100%=F/2)
// passband attenuation spread ap=0.00 dB at fp=0.49% (100%=F/2)
void filter(float i1, float i2, float *o1, float *o2) {
  float t0, x0;
  static float T1, T3, T5, T7, T9, T11, T13, T15, T17;

  // interpolator input: i1=i2=sample(n)
  // decimator input: i1=sample(n), i2=sample(n+1)

  //**** Upper arm ****
  t0 =i1 -T3 ; x0 =  0.226119*t0    -T3 ; T3 =x0 -t0 ; // adaptor  3: g=-0.226119
  t0 =T7 -x0 ; T7 =  0.397578*t0    -T7 ; x0 =T7 -t0 ; // adaptor  7: g=-0.602422
  t0 =T11-x0 ; T11=  0.160677*t0    -T11; x0 =T11-t0 ; // adaptor 11: g=-0.839323
  t0 =T15-x0 ; T15=  0.049153*t0    -T15; *o2=T15-t0 ; // adaptor 15: g=-0.950847

  //**** Lower arm ****
  t0 =i2 -T1 ; x0 =  0.063978*t0    -T1 ; T1 =x0 -t0 ; // adaptor  1: g=-0.063978
  t0 =x0 -T5 ; x0 =  0.423068*t0    -T5 ; T5 =x0 -t0 ; // adaptor  5: g=-0.423068
  t0 =T9 -x0 ; T9 =  0.258673*t0    -T9 ; x0 =T9 -t0 ; // adaptor  9: g=-0.741327
  t0 =T13-x0 ; T13=  0.094433*t0    -T13; x0 =T13-t0 ; // adaptor 13: g=-0.905567
  t0 =T17-x0 ; T17=  0.015279*t0    -T17; *o1=T17-t0 ; // adaptor 17: g=-0.984721

  // decimator output: sample(n)=(o1+/-o2)/2 for lowpass/highpass
  // interpolator output: sample(n)=o1, sample(n+1)=+/-o2 for lowpass/highpass
}
```

The output of the tests are in Figures 4-5.

The interpolator has been tested with a 1 kHz sinusoid, the input sampling frequency is Fi=64 kHz, the output is Fo=128 kHz: the low-pass interpolated signal is at 1 kHz, the high-pass interpolated signal is at 63 kHz. In general, an input frequency 0<f<Fi/2 will be seen at f on the low-pass output and Fi-f on the high-pass output.

The decimator has been tested with a 1 kHz and an 18 kHz signal, the input sampling frequency is Fi=64kHz, the output is Fo=32 kHz: the low-pass decimated signal is at 1 kHz, the high-pass folded decimated signal is at 14 kHz. In general, an input frequency 0<f<Fi/4 will be kept by the low-pass and seen at f in the output, an input frequency Fi/4<f<Fi/2 will be kept by the high-pass and folded at Fi/2-f.

Figure 4. Interpolator test for the bi-reciprocal Elliptic/Cauer filter, order 19, designed for interpolation/decimation

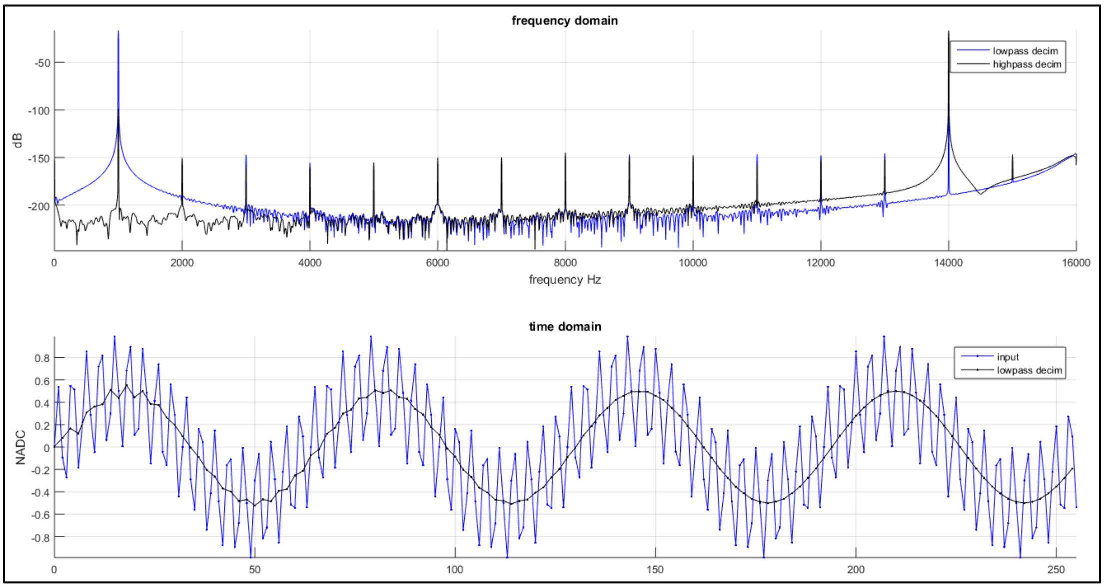
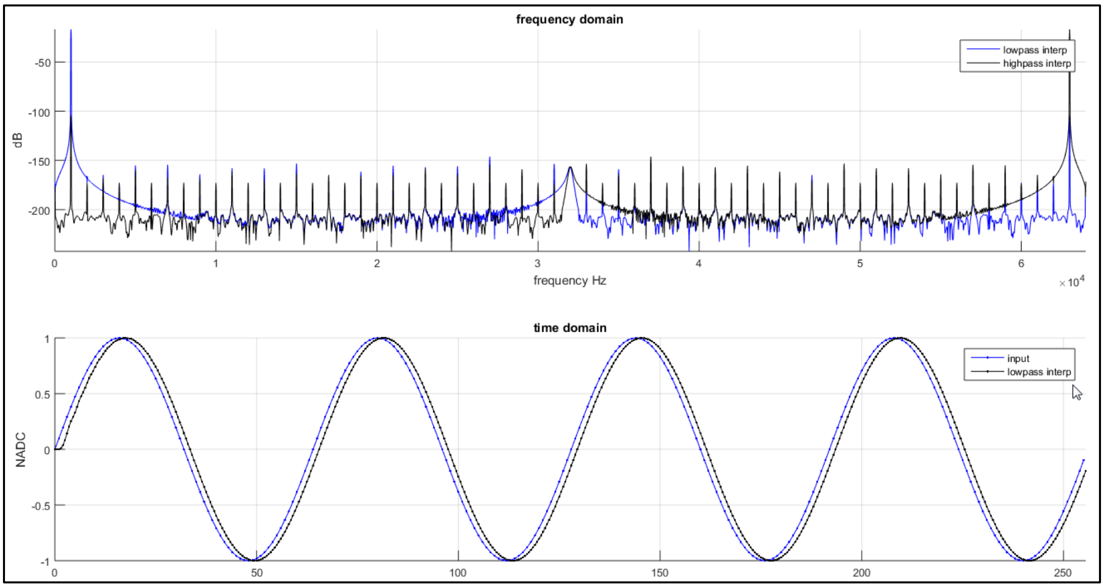


Figure 5. Decimator test for the bi-reciprocal Elliptic/Cauer filter, order 19, designed for interpolation/decimation



Support material

Related design support material
Wearable sensor unit reference design, STEVAL-WESU1
SensorTile development kit, STEVAL-STLKT01V1
Documentation
Design tip, DT0091, Lattice wave digital filter design and automatic C code generation

Revision history

Date	Version	Changes
16-Nov-2017	1	Initial release.