life.augmented

DT0091 Design tip

Lattice wave digital filter design and automatic C code generation

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 the advantages of lattice wave digital filters (LWDF) and how to use the design tool provided here to generate the corresponding C source code.

- The design tool covers the most common filter types: Butterworth, Chebyshev and Elliptic/Cauer (normal or bi-reciprocal).
- The design tool automatically generates the floating or fixed-point C source code for the filter. The user can choose to optimize for an implementation with minimal memory footprint, or he can choose to optimize for an implementation easily portable to RTL.
- The design tool generates a general implementation with a primary output (low-pass) and a secondary complementary output (high-pass). For the case of bi-reciprocal filters, the tool can generate a specific implementation for interpolation or decimation.

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.

The design tool presented here generates C source code for the lattice implementation, where the upper and lower branches are realized by cascaded 1st and 2nd order all-pass sections, with 1 or 2 two-port adaptors respectively. Each adaptor is made of 3 adders and 1 multiplier. The design tool computes the gamma coefficients, which determine the multipliers' coefficients.

The design is restricted to low-pass filters. The complementary high-pass output is also available. By concatenating two filters, the user can realize a band-pass filter. The frequency response can be mirrored around F/4 by negating even gamma coefficients.

An important subclass of lattice wave digital filters is made of filters with 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 design process

The design tool provided here is in C source code format. This is the command line to generate the executable using GCC: gcc -o WDdesign WDdesign.c

Step 1. Choose the desired filter type:

- Butterworth: no ripples in the pass-band, maximally flat step response, more linear phase and constant group delay in the pass-band. Depending on the chosen specifications, it can be bi-reciprocal or it can have half of the coefficients equal to a power of two, which greatly simplify the implementation of multipliers.
- Chebyshev: type I or direct, ripples in the pass-band, steeper roll-off with respect to Butterworth. (type II, or inverse, has ripples in the stop-band, steeper roll-off but less than type I; it is derived from type I by negating all even gamma coefficients)
- Elliptic/Cauer: ripples in the pass-band and stop-band, steepest roll-off and therefore minimal order but with over/undershoots in the step response. One can choose to design a normal or bi-reciprocal filter before giving the specifications.

Step 2. Enter the specifications: sampling frequency, minimum stop-band attenuation, stop-band attenuation lower edge frequency, and – if the filter is not an Elliptic/Cauer bi-reciprocal – maximum pass-band attenuation spread, pass-band attenuation upper edge frequency.

Step 3. Choose the order: the design tool will estimate the minimum order required to meet the specifications. By choosing an order higher than the minimum, the user has a larger design margin, which can be allocated to improve the pass-band or the stop-band.

Step 4. Choose the auxiliary parameters so that the design margin is allocated to the pass-band or to the stop-band, as desired. The design tool will compute the allowed range.

- Butterworth: the auxiliary parameter is the gamma value of half of the coefficients. Depending on the range, it may be possible to set it to 0 to get a bi-reciprocal filter; or it may be possible to set it to a power of two, to simplify the multipliers.
- Chebyshev: the auxiliary parameter is the pass-band ripple factor. The minimum is associated with a better pass-band, the maximum with a better stop-band.

- Elliptic/Cauer: the auxiliary parameters are the actual value of the stop-band edge frequency and the pass-band ripple factor. The minimum stop-band edge frequency is associated with a better transition band, the maximum with better pass and stop-band. The minimum ripple factor is associated to a better pass-band, the maximum to a better stop-band.

Step 5. The design tool will compute and print on the screen the gamma coefficients of the lattice wave digital filter. The gamma coefficients are also saved to “WDgamma.txt” file (note that the file will be overwritten). In the companion Design tip, there is a MATLAB® script that loads this file and plots the actual frequency response; there are also MATLAB scripts to test the filter implementation.

Step 6. Automatic C source code generation: the user has to choose several implementation details.

- If the filter is bi-reciprocal, the user must choose if the implementation targets a normal filter or an interpolation/decimation filter.
- How many bits are used for the coefficients: 0 means that coefficients are “float”, otherwise coefficients are “int” with the specified number of bits. On most platforms, “int” is 32 bits, if the signal is 16 bits, then coefficients can be 16 bits at most. One bit is for sign: the user can enter 15 bits.
- Choose if temporary variables or registers are re-used: if yes, the number of variables and the memory footprint is minimal; if not, variables and registers are unique, not shared among adaptors, which makes it easier to port the code to RTL.

Design examples

The following design examples are from “*Explicit Formulas for Lattice Wave Digital Filters*” by Lajos Gazsi, 1985 IEEE. Input specifications and auxiliary parameters are reported in the same order as requested by the design tool. The corresponding list of gamma coefficients is reported below, together with examples of generated C source code.

Input specifications and auxiliary parameters

Type	**Butter**	**Butter**	**Cheby**	**Elliptic**	**Elliptic**
Bi-reciprocal	-	-	-	-	Yes
F sampling	16000 Hz	16000 Hz	16000 Hz	16000 Hz	64000 Hz
as stop att.	65 dB	55 dB	40 dB	65 dB	65 dB
fs stop edge	6000 Hz	6000 Hz	5000 Hz	4600 Hz	16300 Hz
ap pass att.	0.5 dB	0.5 dB	1 dB	0.2 dB	-
fp pass edge	3400 Hz	3400 Hz	3000 Hz	3400 Hz	-
N order	9	7	5	7	19
g gamma	0 **bi-reciprocal**	0.0625 = **$1/2^4$**	-	-	-
fs actual	-	-	-	4500 Hz	16300 Hz
ep pass ripple	-	-	0.4	0.18	0.000143

Output gamma coefficients

Gamma 0	0	+0.031281	+0.633810	+0.512898	0
Gamma 1	-0.031091	-0.053071	-0.537177	-0.404406	-0.063978
Gamma 2	0	+0.0625	+0.660461	+0.607707	0
Gamma 3	-0.132474	-0.232840	-0.826042	-0.668724	-0.226119
Gamma 4	0	+0.0625	+0.375455	+0.334236	0
Gamma 5	-0.333333	-0.636546		-0.896134	-0.423068
Gamma 6	0	+0.0625		+0.206694	0
Gamma 7	-0.704088				-0.602422
Gamma 8	0				0
Gamma 9					-0.741327
Gamma 10					0
Gamma 11					-0.839323
Gamma 12					0
Gamma 13					-0.905567
Gamma 14					0
Gamma 15					-0.950847
Gamma 16					0
Gamma 17					-0.984721
Gamma 18					0

Generated C code examples

Butterworth, order 9, bi-reciprocal, to be used for interpolation/decimation, floating point, optimized for software implementation (temporary variables are re-used).

```
// Buttworth order 9, bireciprocal, for interpolation/decimation
// stopband min attenuation as=65 dB at fs=0.75% (100%=F/2)
// passband attenuation spread ap=0.50 dB at fp=0.42% (100%=F/2)
void filter(float i1, float i2, float *o1, float *o2) {
  float t0, x0;
  static float T1, T3, T5, T7;

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

  //**** Upper arm ****
  t0 =i1 -T3 ; x0 =  0.132474*t0    -T3 ; T3 =x0 -t0 ; // adaptor  3: g=-0.132474
  t0 =T7 -x0 ; T7 =  0.295912*t0    -T7 ; *o2=T7 -t0 ; // adaptor  7: g=-0.704088

  //**** Lower arm ****
  t0 =i2 -T1 ; x0 =  0.031091*t0    -T1 ; T1 =x0 -t0 ; // adaptor  1: g=-0.031091
  t0 =x0 -T5 ; *o1=  0.333333*t0    -T5 ; T5 =*o1-t0 ; // adaptor  5: g=-0.333333

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

Butterworth, order 7, half of coefficients are set to $1/2^4$, fixed-point implementation, 15 bits + 1 sign bit per coefficient, optimized for hardware RTL (unique temporary variables).

```
// Buttworth order 7
// stopband min attenuation as=55 dB at fs=0.75% (100%=F/2)
// passband attenuation spread ap=0.50 dB at fp=0.42% (100%=F/2)
void filter(int i1, int i2, int *o1, int *o2) {
  int t0, t1, t2, t3, t4, t5, t6;
  int x0, x1, x2, x4, x6;
  static int T0, T1, T2, T3, T4, T5, T6;
  // filter input: i1=i2=sample(n)

  //**** Upper arm ****
  t0 =T0 -i2 ; x0 =(( 1025*t0 )>>15)+T0 ; T0 =x0 -t0 ; // adaptor  0: g=+0.031281
  t4 =T4 -T3 ; x4 =       (t4  >> 4)+T4 ; T4 =x4 -t4 ; // adaptor  4: g=+0.062500
  t3 =x0 -x4 ; *o2=(( 7629*t3 )>>15)-x4 ; T3 =*o2-t3 ; // adaptor  3: g=-0.232840

  //**** Lower arm ****
  t2 =T2 -T1 ; x2 =       (t2  >> 4)+T2 ; T2 =x2 -t2 ; // adaptor  2: g=+0.062500
  t1 =i1 -x2 ; x1 =(( 1739*t1 )>>15)-x2 ; T1 =x1 -t1 ; // adaptor  1: g=-0.053071
  t6 =T6 -T5 ; x6 =       (t6  >> 4)+T6 ; T6 =x6 -t6 ; // adaptor  6: g=+0.062500
  t5 =x6 -x1 ; T5 =((11909*t5 )>>15)-x6 ; *o1=T5 -t5 ; // adaptor  5: g=-0.636546

  // filter output: sample(n)=(o1+/-o2)>>1 for lowpass/highpass
}
```

Design tool

The code is formatted for compactness, not for readability. Strings are set to avoid double spaces, which may not be copied correctly from the PDF, compromising the formatting of the generated C source code.

```
// Explicit Formulas for Lattice Wave Digital Filters, Lajos Gazsi, 1985 IEEE
#include <stdio.h>
#include <math.h>
#define BUTTW '1' // Butterworth maximally flat
#define CHEB1 '2' // Chebyshev
#define ELLIP '3' // Elliptic/Cauer
#define PI 3.141592653589793L
#define MAXSTR 100 // max string length
#define NMAX 100 // max filter order
#define FNAMECODE "WDfilter.c"
#define FNAMEGAMMA "WDgamma.txt"

void adaptor(int i, int t, double g, char *in1, char *in2, char *out1, char *out2, FILE *f, int n) {
  char pm='+'; int si=0, so=0, Q=1<<n, b, aQ; double a;
  if(g==0.0) { fprintf(f,"%2s%-3s=%-7s;%28s%-3s=%-7s;","",out1,in2,"",out2,in1);
    fprintf(f," // adaptor %2d: g= 0.0\n",i); return; }
  if (g> 0.5) { a=1.0-g; } else if (g> 0.0) { a=g; si=so=1; }
  else if (g>-0.5) { a=-g; so=1; pm='-'; } else { a=1.0+g; si=1; pm='-'; }
  aQ=(int)(a*(double)Q); for(b=0;(b<n)&&(aQ!=(1<<b));b++); b=n-b;
  if(si) fprintf(f," "" t%-2d=%s-%s;",t,in2,in1); else fprintf(f," "" t%-2d=%s-%s;",t,in1,in2);
  if(!n) {  if(so) fprintf(f," %s= "" %f*t%-5d%c%s;",        out1,a,t,pm,in2);
            else   fprintf(f," %s= "" %f*t%-5d%c%s;",        out2,a,t,pm,in2); }
  else { if(b) { if(so) fprintf(f," %s=%7s(t%-3d>>%2d)%c%s;",out1,"",t,b,pm,in2);
                 else   fprintf(f," %s=%7s(t%-3d>>%2d)%c%s;",out2,"",t,b,pm,in2); }
         else  { if(so) fprintf(f," %s=((%5d*t%-2d)>>%2d)%c%s;",out1,aQ,t,n,pm,in2);
                 else   fprintf(f," %s=((%5d*t%-2d)>>%2d)%c%s;",out2,aQ,t,n,pm,in2); } }
  if(so) fprintf(f," %s=%s-t%-2d;",out2,out1,t); else fprintf(f," %s=%s-t%-2d;",out1,out2,t);
  fprintf(f," // adaptor %2d: g=%+f\n",i,g); return;
}

int main(int argc, char *argv[]) {
  FILE *fout; char ftype, *dtype, tline[MAXSTR], in1[10], in2[10],out1[10],out2[10];
  int i, i1, i2, j, bi=0, id=0, rx, Nmin, N, nbits, ccode;
  double F, as, es, fs, phis, ap, ep, fp, phip, t, tt; // filter specs and design
  double k0, k1, k4, c1, c2, c3; // filter order computation
  double ks, kp, g; // Butterworth coefficient computation
  double epmin, w, r, Ai, Bi; // Chebyshev coefficient computation
  double fsmin, q0, q1, q2, q3, q4, m3, m2, m1, m0; // Elliptic/Cauer computation
  double gamma[NMAX]; // filter coefficients

  if(argc<3) { printf("default use: %s [%s [%s]]\n\n",argv[0],FNAMEGAMMA,FNAMECODE); }
  printf("%c) Butterworth    (no ripples, max flat step response)\n",BUTTW);
  printf("%c) Chebyshev I    (passband ripples, steeper transition)\n",CHEB1);
  printf("%c) Elliptic/Cauer (pass&stopband ripples, steepest transition)\n",ELLIP);
  printf("filter type? "); scanf("%c",&ftype);
  if (ftype==ELLIP) { printf("bireciprocal filter (0=no 1=yes)? "); scanf("%d",&bi); }
  printf("F sampling frequency [Hz]? "); scanf("%lf",&F);
  printf("as stopband min attenuation [dB]? "); scanf("%lf",&as);
  es=sqrt(exp(log(10.0)*(as/10.0))-1.0); // as=10.0*log(1.0+es*es)/log(10.0);
  printf("fs stopband lower edge [Hz]? "); scanf("%lf",&fs);
  phis=tan(PI*fs/F); // stopband transformed frequency
  if (bi) { ep=1.0/es; phip=1.0/phis; fp=(F/2.0)-fs; // ap set below after epmin<=ep is chosen
            printf("fp passband upper edge %.0f Hz\n",fp);
  } else {  printf("ap passband attenuation spread [dB]? "); scanf("%lf",&ap);
            ep=sqrt(exp(log(10.0)*(ap/10.0))-1.0); // ap=10.0*log(1.0+ep*ep)/log(10.0);
            printf("fp passband upper edge [Hz]? "); scanf("%lf",&fp);
            phip=tan(PI*fp/F); // passband transformed frequency
  }

  k0=sqrt(phis/phip); k1=k0*k0+sqrt(k0*k0*k0*k0-1.0);
  for(k4=k1,i=0;i<3;i++) k4=k4*k4+sqrt(k4*k4*k4*k4-1.0);
  switch (ftype) {
    case BUTTW: c1=1.0; c2=1.0; c3=k0*k0;   break;
    case CHEB1: c1=1.0; c2=2.0; c3=k1;      break;
    case ELLIP: c1=8.0; c2=4.0; c3=2.0*k4;  break;
```

```
}
Nmin=(int)ceil(c1*log(c2*es/ep)/log(c3)); if ((Nmin%2)==0) Nmin++; // odd min order
printf("order (min=%d, max=%d, must be odd)? ",Nmin,NMAX); scanf("%d",&N);
if (N>NMAX) N=NMAX; if ((N%2)==0) return 0; // if even, code would not be correct

for(i=0;i<NMAX;i++) gamma[i]=0.0;
switch (ftype) {
  case BUTTW:
    ks=(exp(log(es*es)*(1.0/N))-phis*phis) / (exp(log(es*es)*(1.0/N))+phis*phis);
    kp=(exp(log(ep*ep)*(1.0/N))-phip*phip) / (exp(log(ep*ep)*(1.0/N))+phip*phip);
    if ((ks<=0.0)&&(kp>=0.0)) printf("set g = 0 (half coeffs 0, bireciprocal)\n");
    t=(log(fabs(ks))/log(2.0)); if(ks<0.0) t=-t; i1=ceil(t);
    t=(log(fabs(kp))/log(2.0)); if(kp<0.0) t=-t; i2=floor(t);
    for(i=i1;i<=i2;i++) printf("set g = 2^%2d = %f (half coeffs, no mul)\n",i,exp(log(2.0)*(double)i));
    printf("g (%f better passband to %f better stopband)? ",ks,kp); scanf("%lf",&g);
    if (g==0.0) { // Butterworth bireciprocal
      bi=1; gamma[0]=0.0;
      for(i=1;i<=((N-1)/2);i++) { t=tan(PI*(double)i/2.0/(double)N);
                                  gamma[i*2-1]=-(t*t); gamma[i*2]=0.0; }
    } else { // Butterworth
      bi=0; t=sqrt(1.0-g*g); gamma[0]=(1.0+g-t)/(1.0+g+t);
      for(i=1;i<=((N-1)/2);i++) { tt=t*cos(PI*(double)i/(double)N);
                                  gamma[i*2-1]=(tt-1.0)/(tt+1.0); gamma[i*2]=g; } } break;
  case CHEB1:
    for(t=1.0,i=0;i<N;t=t*k1,i++); epmin=(2.0*es)/t;
    printf("ep (%f better passband to %f better stopband)? ",epmin,ep); scanf("%lf",&ep);
    ap=10.0*log(1.0+ep*ep)/log(10.0);
    w=exp(log(((1.0/ep)+sqrt(1.0/(ep*ep)+1.0)))*(1.0/(double)N)); r=(w-1.0/w)*phip;
    gamma[0]=(2.0-r)/(2.0+r);
    for(i=1;i<=((N-1)/2);i++) {
      t=PI*(double)i/(double)N; Ai=r*cos(t); Bi=(w*w+1.0/(w*w)-2.0*cos(2.0*t))*(phip*phip)/4.0;
      gamma[i*2-1]=(Ai-Bi-1.0)/(Ai+Bi+1.0); gamma[i*2]=(1.0-Bi)/(1.0+Bi); } break;
  case ELLIP:
    if (bi) r=es; else r=sqrt(es/ep); r=r*r+sqrt(r*r*r*r-1.0); r=r*r+sqrt(r*r*r*r-1.0);
    t=(0.5)*exp(log(2.0*r)*(4.0/(double)N)); for(i=0;i<4;i++) t=sqrt((0.5)*(t+1/t));
    if (bi) fsmin=(F/PI)*atan(t); else fsmin=(F/PI)*atan(phip*t*t);
    printf("fs (%.0f better transition to %.0f better pass&stopband)? ",fsmin,fs); scanf("%lf",&fs);
    phis=tan(PI*fs/F); if (bi) q0=phis; else q0=sqrt(phis/phip);
    q1=q0*q0+sqrt(q0*q0*q0*q0-1.0); q2=q1*q1+sqrt(q1*q1*q1*q1-1.0);
    q3=q2*q2+sqrt(q2*q2*q2*q2-1.0); q4=q3*q3+sqrt(q3*q3*q3*q3-1.0);
    m3=(0.5)*exp(log(2.0*q4)*((double)N/2.0)); m2=sqrt((0.5)*(m3+1.0/m3));
    m1=sqrt((0.5)*(m2+1.0/m2)); m0=sqrt((0.5)*(m1+1.0/m1));
    if (bi) epmin=1.0/m0; else epmin=es/(m0*m0);
    printf("ep (%f better passband to %f better stopband)? ",epmin,ep); scanf("%lf",&ep);
    ap=10.0*log(1.0+ep*ep)/log(10.0);
    if (bi) es=m0; else es=ep*m0*m0; as=10.0*log(1.0+es*es)/log(10.0);
    g=1.0/ep+sqrt(1.0/(ep*ep)+1.0); g=m1*g+sqrt((m1*g*m1*g)+1.0); g=m2*g+sqrt((m2*g*m2*g)+1.0);
    t=exp(log((m3/g)+sqrt((m3*m3/g/g)+1.0))*(1.0/(double)N));
    t=1.0/(2.0*q4)*(t-1.0/t); t=1.0/(2.0*q3)*(t-1.0/t);
    t=1.0/(2.0*q2)*(t-1.0/t); t=1.0/(2.0*q1)*(t-1.0/t);
    if (bi) w=-1.0; else w=1.0/(2.0*q0)*(t-1.0/t);
    if (bi) gamma[0]=0; else gamma[0]=(1.0+w*q0*phip)/(1.0-w*q0*phip);
    for(i=1;i<=((N-1)/2);i++) {
      t=q4/sin((double)i*PI/(double)N); t=1.0/(2.0*q3)*(t+1.0/t); t=1.0/(2.0*q2)*(t+1.0/t);
      t=1.0/(2.0*q1)*(t+1.0/t); t=1.0/(2.0*q0)*(t+1.0/t); tt=1.0/t;
      if (bi) { Bi=1.0; Ai=2.0/(1.0+tt*tt)*sqrt(1.0-(q0*q0+1.0/(q0*q0)-tt*tt)*tt*tt);
                gamma[i*2-1]=(Ai-2.0)/(Ai+2.0); gamma[i*2]=0.0;
      } else {  t=(1.0+w*w*tt*tt); Bi=(w*w+tt*tt)/t*(q0*phip*q0*phip);
                Ai=(-2.0*w*q0*phip)/t*sqrt(1.0-(q0*q0+1.0/(q0*q0)-tt*tt)*tt*tt);
                gamma[i*2-1]=(Ai-Bi-1.0)/(Ai+Bi+1.0); gamma[i*2]=(1.0-Bi)/(1.0+Bi);  }
  } break; }
printf("flip frequency response around F/4 (0=no 1=yes)? "); scanf("%d",&i);
if(i) for(i=0;i<N;i+=2) gamma[i]=-gamma[i];
if(argc<2) sprintf(tline,"%s",FNAMEGAMMA); else sprintf(tline,"%s",argv[1]);
if(NULL!=(fout=fopen(tline,"wt"))) fprintf(fout,"gamma\n");
for(i=0;i<N;i++) { printf("  gamma[%2d] = ",i);
  if(gamma[i]==0.0) printf(" 0.0\n"); else printf("%+f\n",gamma[i]);
  if(NULL!=fout) fprintf(fout,"%f\n",gamma[i]); }
if(NULL!=fout) { printf("gamma coefficients saved to file %s\n",tline); fclose(fout); }

printf("C code (0=no 1=yes)? "); scanf("%d",&ccode); if (ccode==0) return 0;
if(argc<3) sprintf(tline,"%s",FNAMECODE); else sprintf(tline,"%s",argv[2]);
if (NULL==(fout=fopen(tline,"wt"))) { printf("C code not saved to %s\n",tline); return 0; }
id=0; if (bi) { printf("bireciprocal for decimation/interpolation (0=no, 1=yes)? "); scanf("%d",&id); }
printf("bits (not including sign bit, 0=no quantization)? "); scanf("%d",&nbits);
if(nbits==0) dtype="float\0"; else dtype="int\0";
printf("reuse and minimize temporary variables (0=no, 1=yes)? "); scanf("%d",&rx);

switch (ftype) {
  case BUTTW: fprintf(fout,"// Buttworth order %d",N); break;
  case CHEB1: fprintf(fout,"// Chebyshev I order %d",N); break;
  case ELLIP: fprintf(fout,"// Elliptic/Cauer order %d",N); break; }
if(bi) { fprintf(fout,", bireciprocal"); if(id) fprintf(fout,", for interpolation/decimation"); }
fprintf(fout,"\n// stopband min attenuation as=%.0f dB at fs=%.2f%% (100%%=F/2)\n",as,fs/(F/2.0));
fprintf(fout,"// passband attenuation spread ap=%.2f dB at fp=%.2f%% (100%%=F/2)\n",ap,fp/(F/2.0));
fprintf(fout,"void filter(%s i1, %s i2, %s *o1, %s *o2) {\n",dtype,dtype,dtype,dtype);

if(rx) {
  if((!id)||(N>1)) fprintf(fout," "" %s t0",dtype);
  if(id) { if(N>=7) fprintf(fout,",x0"); }
  else   { if(N>=3) fprintf(fout,",x0"); if(N>=5) fprintf(fout,",x1"); }
  if((!id)||(N>1)) fprintf(fout,";\n");
} else {
    if((!id)||(N>1)) fprintf(fout," "" %s",dtype); j=' ';
    for(i=0;i<N;i++) if(gamma[i]!=0.0) { fprintf(fout,"%ct%d",j,i); j=','; }
    if((!id)||(N>1)) fprintf(fout,";\n");

    if(!id) {
       if(N>1) { fprintf(fout," "" %s",dtype); j=' '; }
       for(i=0;i<=(N-5);i++) { fprintf(fout, "%cx%d",j,i); j=','; }
       if((N-3)>0) { fprintf(fout,"%cx%d",j,N-3); j=','; }
       if((N-1)>0) { fprintf(fout,"%cx%d",j,N-1); j=','; }
       if(N>1) fprintf(fout,";\n");
    } else {
      if(N>5) { fprintf(fout," "" %s",dtype); j=' '; }
      for(i=1;i<=(N-5);i+=2) { fprintf(fout,"%cx%d",j,i); j=','; }
      if(N>5) fprintf(fout,";\n");
    } }

  if((!id)||(N>1)) fprintf(fout," "" static %s ",dtype);
  if (!id) { fprintf(fout,"T0"); for(i=1;i<=(N-1);i++ ) fprintf(fout,",T%d",i); }
  else     { if(N>1) fprintf(fout,"T1"); for(i=3;i< (N-1);i+=2) fprintf(fout,",T%d",i); }
  if((!id)||(N>1)) fprintf(fout,";\n\n");

  if(id) fprintf(fout," "" // interpolator input: i1=i2=sample(n)\n "
                         " // decimator input: i1=sample(n), i2=sample(n+1)\n");
  else   fprintf(fout," "" // filter input: i1=i2=sample(n)\n");

  fprintf(fout,"\n "" //**** Upper arm ****\n");
  if(id&&(N<=3)) fprintf(fout," "" *o2=i1;\n");
  if (!id) { if(N<=3) {       adaptor(0,0,gamma[0],"i2 ","T0 ","*o2","T0 ",fout,nbits); }
             else     { j=rx; adaptor(0,0,gamma[0],"i2 ","T0 ",rx?"x1 ":"x0 ","T0 ",fout,nbits); }; }
  for(i=3;i<N;i+=4) { if(!id) {
      sprintf(in1,"T%-2d",i); sprintf(in2,"T%-2d",i+1);
      sprintf(out1,"x%-2d",rx?(i+1)%2:i+1); sprintf(out2,"T%-2d",i+1);
      adaptor(i+1,rx?0:i+1,gamma[i+1],in1,in2,out1,out2,fout,nbits);
      sprintf(in1 ,"x%-2d",rx?j%2:j); sprintf(in2 ,"x%-2d",rx?(i+1)%2:i+1);
      if ((i+4)>=N) sprintf(out1,"*o2"); else sprintf(out1,"x%-2d",rx?i%2:i); sprintf(out2,"T%-2d",i);
      adaptor(i,rx?0:i,gamma[i],in1,in2,out1,out2,fout,nbits);
    } else {
      if (i==3) sprintf(in1,"i1 "); else sprintf(in1,"x%-2d",rx?0:j); sprintf(in2 ,"T%-2d",i);
      if ((i+4)>=N) sprintf(out1,"*o2"); else sprintf(out1,"x%-2d",rx?0:i); sprintf(out2,"T%-2d",i);
      adaptor(i,rx?0:i,gamma[i],in1,in2,out1,out2,fout,nbits);
    } j=i; }

  fprintf(fout,"\n "" //**** Lower arm ****\n");
  if(N==1) { if(!id) fprintf(fout," "" *o1=i1;\n"); else fprintf(fout," "" *o1=i2;\n"); }
  for(i=1;i<N;i+=4) { if(!id) {
      sprintf(in1,"T%-2d",i); sprintf(in2,"T%-2d",i+1);
      sprintf(out1,"x%-2d",rx?(i+1)%2:i+1); sprintf(out2,"T%-2d",i+1);
      adaptor(i+1,rx?0:i+1,gamma[i+1],in1,in2,out1,out2,fout,nbits);
      if (i==1) sprintf(in1,"i1 "); else sprintf(in1,"x%-2d",rx?j%2:j); sprintf(in2 ,"x%-2d",rx?(i+1)%2:i+1);
      if ((i+4)>=N) sprintf(out1,"*o1"); else sprintf(out1,"x%-2d",rx?i%2:i); sprintf(out2,"T%-2d",i);
      adaptor(i,rx?0:i,gamma[i],in1,in2,out1,out2,fout,nbits);
    } else {
      if (i==1) sprintf(in1,"i2 "); else sprintf(in1,"x%-2d",rx?0:j); sprintf(in2 ,"T%-2d",i);
      if ((i+4)>=N) sprintf(out1,"*o1"); else sprintf(out1,"x%-2d",rx?0:i); sprintf(out2,"T%-2d",i);
      adaptor(i,rx?0:i,gamma[i],in1,in2,out1,out2,fout,nbits);
    } j=i; }

  if(id) fprintf(fout,"\n "" // decimator output: sample(n)=(o1+/-o2)>>1 for lowpass/highpass\n"
                        " "" // interpolator output: sample(n)=o1, sample(n+1)=+/-o2 for lowpass/highpass\n");
  else fprintf(fout,  "\n "" // filter output: sample(n)=(o1+/-o2)>>1 for lowpass/highpass\n");
  fprintf(fout,"}\n\n"); printf("C code saved to file %s\n",tline); fclose(fout); return 0;
}
```

Support material

Related design support material
Wearable sensor unit reference design, STEVAL-WESU1
SensorTile development kit, STEVAL-STLKT01V1
Documentation
Design tip, DT0092, Lattice wave digital filter test and verification

Revision history

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