FIR filter design by sampling, windowing and modulating the sinc() function

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 design an FIR filter by sampling, windowing and modulating the sinc() function.

- A MATLAB® script is provided to design the FIR filter. The script does not use MATLAB functions (except to plot) and does not vectorize computations, so it can be easily translated to every programming language.
- A MATLAB test script is provided to verify the performance of the FIR filter. The complementary filter is also verified.
- Two different C implementations are provided, Direct I form and Direct II Transposed form, together with their corresponding MATLAB test script to verify their performance.

Description

Digital filters can be seen as a weighted average of a certain number of input samples, current and past, plus the weighted average of a certain number of past output samples. The former average is the finite impulse response part (FIR); the latter average is the infinite impulse response part (IIR).

The advantages of FIR filters are the following:

- FIR filters are always stable, while IIR filters can be unstable and are indeed sensitive to quantization of coefficients and computation errors
- FIR filters can have perfect linear phase (symmetric coefficients), while IIR filters can only approximate linear phase

- FIR transients have finite duration, while IIR transients may be infinite: an input that goes to zero does not guarantee that the output returns to zero

The disadvantage is that FIR filters need to have a lot of coefficients to meet the requirements, while equivalent IIR filters can be much shorter.

FIR filter design algorithm

The FIR filter is defined by a set of coefficients. In the time domain, filtering is equivalent to the convolution of the input and the coefficient set. In the frequency domain, filtering is equivalent to multiplying the spectrum of the input with the FFT transform of the coefficients. The ideal low pass filter has a boxcar FFT transform in the frequency domain. This corresponds to the sinc() function, the cardinal sinus, in the time domain.

The sinc() function is defined as sin(pi x)/(pi x). When x goes to 0, the sinc() goes to 1. There is no division by zero.

The coefficients of the FIR filter are computed by **sampling the sinc() function**. The sampling step S determines the cutoff frequency Fc = S * Fs/2, where Fs is the sampling frequency.

The sampling interval determines the approximation of the ideal low pass: the wider the interval, the better the approximation. The interval goes from –L to +L, where L = n * S, a multiple of the sampling step. This ensures that the number of coefficients (N) is odd.

The coefficients are symmetric, the first being equal to the last, etc. As the coefficients are symmetric, the phase is linear and the group delay is constant.

As the filter order N is odd, the group delay is an integer, (N-1)/2. The complementary filter can then be realized by subtracting the output of the filter from the delayed signal. The delay has to match the group delay of the filter.

In order to improve the stopband attenuation, **a window is applied**: each coefficient is multiplied by the corresponding window coefficient. Multiplication in the time domain is equivalent to convolution in the frequency domain. The effects are the following

- The effect of the main lobe of the window transform is to smooth the passband of the FIR filter
- The effect of the side lobes of the window transform is to increase the stopband attenuation of the FIR filter

The **coefficients are then normalized** by dividing by the sum of the coefficients themselves. This is done in order to have a DC gain equal to 1 (0 dB).

At this point the FIR filter is a low pass filter. By negating every other coefficient, the FIR filter becomes a high pass filter. In general, one can move the center of passband at a given frequency by **modulating the coefficients**.

Finally, the **coefficients can be quantized** by multiplying by a given scaling factor and rounding to the nearest integer. In order to keep the DC gain equal to 1, the output of the FIR must be divided by the same scaling factor.

FIR filter design MATLAB® script

The script computes the set of plain coefficients and the set of windowed coefficients (Figure 1). The frequency response is computed to highlight the difference between the two filters (Figure 2). The set of windowed coefficients is saved to a file for later use.

The script is commented to enable changes and the design of a specific FIR filter.

```
% sinc(x) = sin(pi*x)/(pi*x), sinc(0)=1
step=.1; % cutoff is f = step*Fs/2
limit=step*40; % increase to reduce ripples
c=[]; i=1;
for x = -limit : step : +limit,
  if abs(x)>0,c(i)=sin(pi*x)/(pi*x);
  else        c(i)=1;
  end;
  i=i+1;
end;

% windowing, hann window
N=length(c);
win=[]; cw=[];
for n = 0 : N-1,
  win(n+1) = 0.5 * (1-cos(2*pi*n/(N-1)));
  cw(n+1) = c(n+1) * win(n+1);
end;

% normalize to have DC gain equal to 1
c=c/sum(c);
cw=cw/sum(cw);

% move center frequency to w0
w0=0.25; % from 0 to 1, 1 is Fs/2
c0=(1/cos(pi*w0))^2; % to maintain DC gain equal to 1
for n = 1 : N,
  c(n)  = c(n)  * cos(pi*w0*(n-1)) * c0;
  cw(n) = cw(n) * cos(pi*w0*(n-1)) * c0;
end;

% quantization
Q = 4096; % 2^12 bits for the coefficient mantissa
for n = 1 : N
  c(n)  = round(c(n)*Q);
  cw(n) = round(cw(n)*Q);
end;
while c(1)==0 && c(end)==0, c=c(2:end-1); end;
while cw(1)==0 && cw(end)==0, cw=cw(2:end-1); end;
N=length(c);
Nw=length(cw);

% save to file
h=fopen('fir_cw.txt','wt'); fprintf(h,'%d\n',cw); fclose(h);

% freqz(c,1) frequency response
% this computation can be vectorized
Fs=100; % sampling frequency
I=1000; % number of frequency points between 0 and Fs/2
for i = 1 : I,
  f(i)=Fs/2/(I-1)*(i-1); w=2*pi*f(i);
  z1=exp(-1i*w/Fs); z=1;
  H(i)=0; Hw(i)=0;
  for n = 1 : max(N,Nw),
    if n<=N,  H(i)  = H(i)  + c(n)/Q  *z; end;
    if n<=Nw, Hw(i) = Hw(i) + cw(n)/Q *z; end;
    z = z*z1; % z^n
  end;
end;

% plots
```

```
figure; hold on;
plot([-(N-1)/2:(N-1)/2],c,'.-'); plot([-(Nw-1)/2:(Nw-1)/2],cw,'o-');
legend(sprintf('sinc, %d non zero coeffs',N-sum(c==0)), ...
       sprintf('win sinc, %d non zero coeffs',Nw-sum(cw==0)));
title(sprintf('Q=%d (%d bits coefficients)',Q,log2(Q)));
xlabel('coefficient number'); ylabel('coefficient value');
grid on; zoom on; axis tight

figure; hold on;
plot(f,20*log10(abs(H))); plot(f, 20*log10(abs(Hw)));
legend('sinc','windowed sinc');
title(sprintf('frequency response, passband %g+/-%g Hz',w0*Fs/2,step*Fs/2));
xlabel('frequency Hz'); ylabel('magnitude dB');
grid on; zoom on; axis tight;
```

Figure 1. Coefficient set computed by the FIR filter design MATLAB® script

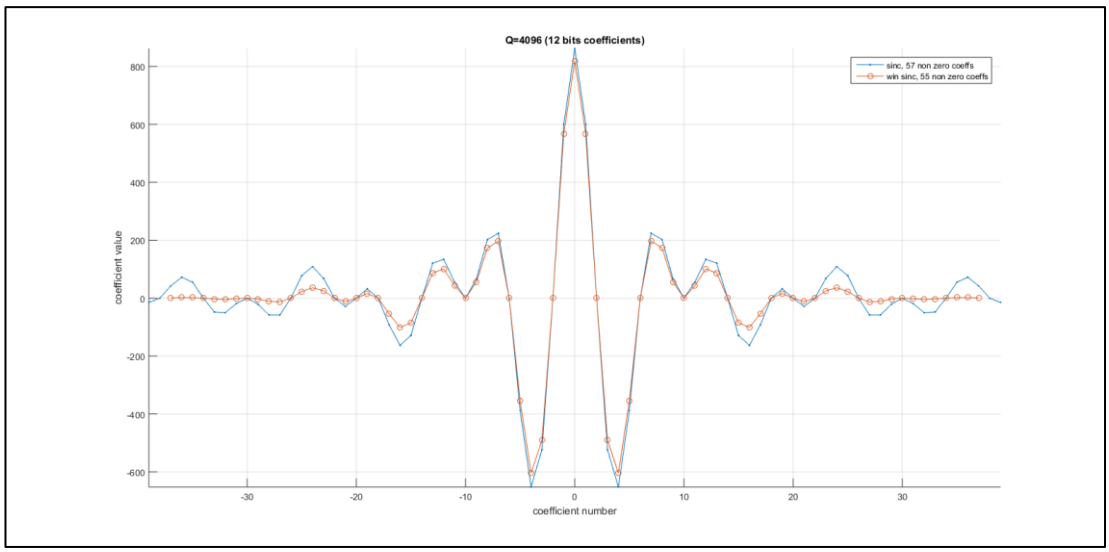
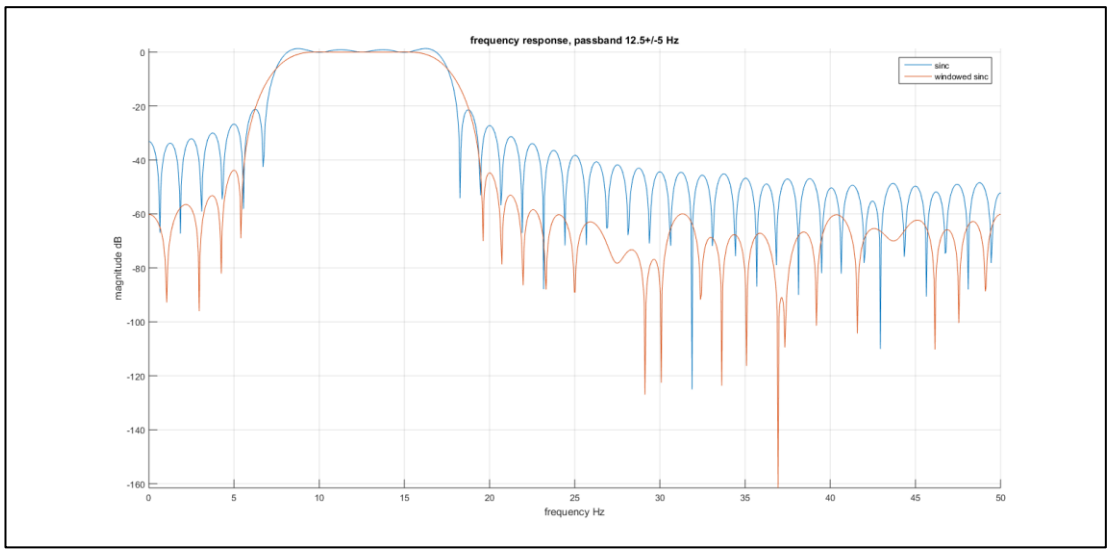


Figure 2. Frequency response computed by the FIR filter design MATLAB® script



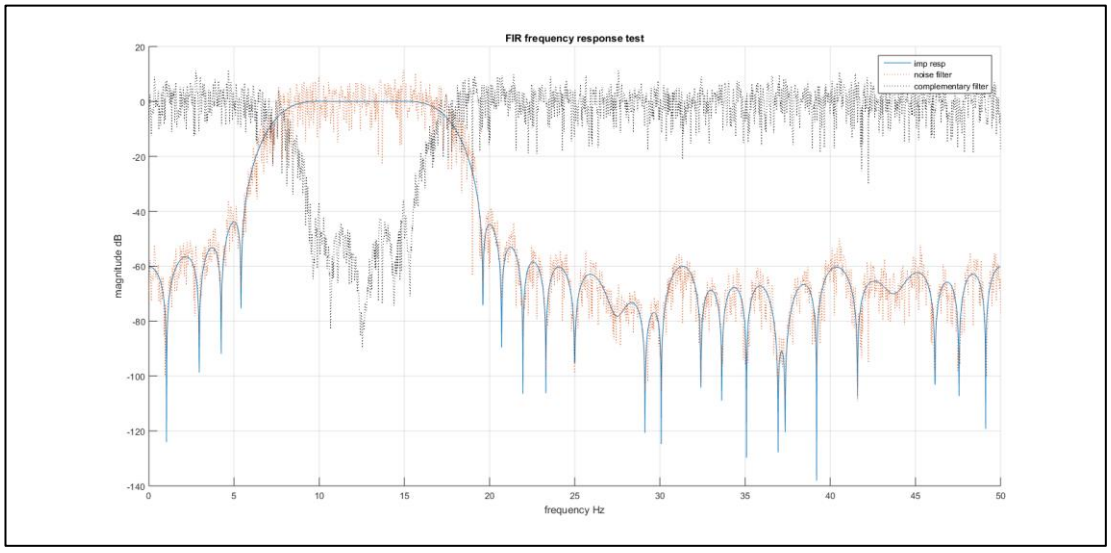
FIR filter test MATLAB® script

This script feeds the filter with an impulse to get the coefficient set on the output. Then it uses the MATLAB function freqz() to compute the frequency response.

The script also feeds the filter with white noise. Then it uses the FFT to verify that the spectrum matches the frequency response. The complementary filter is also implemented by subtracting the filter output from the delayed signal (Figure 3).

```
Fs=100; % sampling frequency
Q = 4096; % 2^12 bits for the coefficient mantissa
NFFT=2^12; % FFT resolution

% impulse response gives the coefficient set
in = zeros(1,NFFT); in(1)=Q;
out = filter(cw/Q,1,in);
[Himp,fimp]=freqz(out/Q,1,NFFT,Fs);

% filter noise and compute power spectrum
in = round((2*rand(1,NFFT)-1)*Q/4);
out = filter(cw/Q,1,in);
NOVL=round(NFFT*0.9); win=hann(NFFT);
[Pn,fn]=pwelch(out/Q,win,NOVL,NFFT,Fs);

% complementary filter by subtraction
delay = (Nw-1)/2;
ind = [zeros(1,delay) in(1:end-delay)];
outc = ind-out;
[Pcn,fcn]=pwelch(outc/Q,win,NOVL,NFFT,Fs);

% plot
figure; hold on;
plot(fimp,20*log10(abs(Himp)));
plot(fn  ,10*log10(abs(Pn  ))+10*log10(Q),':');
plot(fcn ,10*log10(abs(Pcn ))+10*log10(Q),'k:');
legend('imp resp','noise filter','complementary filter');
grid on; zoom on; title('FIR frequency response test');
xlabel('frequency Hz'); ylabel('magnitude dB');
```

Figure 3. Frequency response computed from the impulse response (blue) and from the filtered white noise (red); the complementary filter is also shown (black).



FIR filter C implementation

Implementation as Direct I form (firI.c)

The FIR is implemented as Direct I form: the output is the weighted average of the current and past input samples. Input samples are kept in a circular buffer instead of a FIFO buffer in order to reduce the number of operations.

The utility takes three arguments: the name of the file with the coefficient set, the name of the file with the input samples, and the name of the file for the output samples. The first is saved by the FIR filter design MATLAB® script. The second is generated by the implementation test MATLAB script shown below.

```
#include <stdio.h>
#define MAXCOEFFS 1000

int main(int argc, char *argv[]) {
  FILE *fc=NULL, *fi=NULL, *fo=NULL;
  int coeffs[MAXCOEFFS], ncoeff; // coefficients array
  int cBUF[MAXCOEFFS], iBUF; // circular buffer and its index
  int n, x, y; // temp, in, out

  if(argc<4) { printf("usage: %s cfile infile outfile\n",argv[0]); return 0; }
  for(;;) {
    if(NULL==(fc=fopen(argv[1],"rt"))) { printf("cannot read %s\n", argv[1]);
break; }
    if(NULL==(fi=fopen(argv[2],"rt"))) { printf("cannot read %s\n", argv[2]);
break; }
    if(NULL==(fo=fopen(argv[3],"wt"))) { printf("cannot write %s\n",argv[3]);
break; }
    for(ncoeff=0;(!feof(fc))&&(ncoeff<MAXCOEFFS);ncoeff++) {
      n=fscanf(fc,"%d",&coeffs[ncoeff]); if(n<1) break; }
     //for(n=0;n<ncoeff;n++) printf("coeff[%d]=%d\n",n,coeffs[n]);
     for(n=0;n<(ncoeff-1);n++) cBUF[n]=0; // init status registers
     iBUF=0; // init pointer to circular buffer
     for(;!feof(fi);) {
       n=fscanf(fi,"%d",&x); if(n<1) break; // read new input sample
       // *** filter function ***
       y = coeffs[0]*x; // init output
       for(n=1;n<ncoeff;n++) y += coeffs[n] * cBUF[ (iBUF+ncoeff-n-1)%(ncoeff-1)
]; // MAC
       cBUF[iBUF]=x; iBUF=(iBUF+1)%(ncoeff-1); // store new sample in circular
buffer
       // *** filter function ***
       fprintf(fo,"%d\n",y); // write output sample
     }
     break;
  }
  if(fc!=NULL) fclose(fc); if(fi!=NULL) fclose(fi); if(fo!=NULL) fclose(fo);
  return 0;
}
```

Implementation as Direct II Transposed form (firII.c)

The FIR filter is implemented as Direct II Transposed form: a set of status registers is maintained. The output and the status registers can be updated in parallel. Note that the input sample is multiplied by every coefficient in the set: symmetries can be exploited here to avoid recomputing the same product.

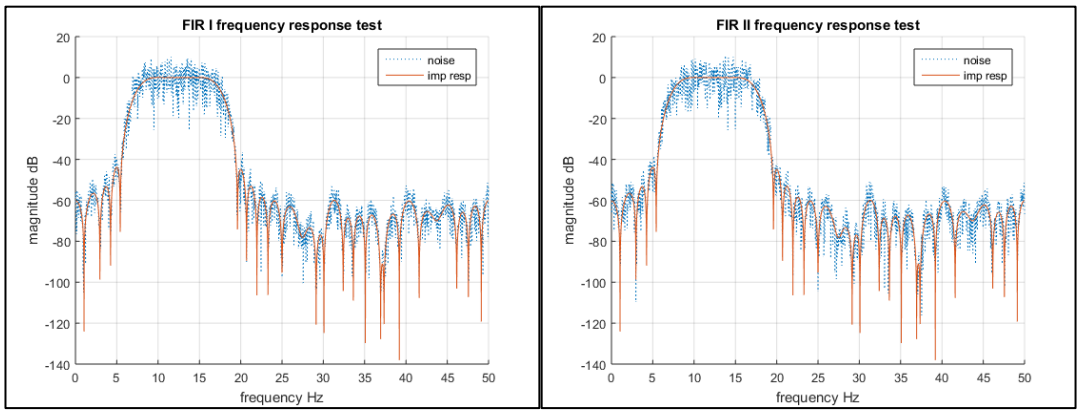
```
#include <stdio.h>
#define MAXCOEFFS 1000

int main(int argc, char *argv[]) {
  FILE *fc=NULL, *fi=NULL, *fo=NULL;
  int coeffs[MAXCOEFFS], ncoeff; // coefficients array
  int BUF[MAXCOEFFS]; // status registers
  int n, x, y; // temp, in, out

  if(argc<4) { printf("usage: %s cfile infile outfile\n",argv[0]); return 0; }
  for(;;) {
```

```
    if(NULL==(fc=fopen(argv[1],"rt"))) { printf("cannot read %s\n", argv[1]);
break; }
    if(NULL==(fi=fopen(argv[2],"rt"))) { printf("cannot read %s\n", argv[2]);
break; }
    if(NULL==(fo=fopen(argv[3],"wt"))) { printf("cannot write %s\n",argv[3]);
break; }
    for(ncoeff=0;(!feof(fc))&&(ncoeff<MAXCOEFFS);ncoeff++) {
      n=fscanf(fc,"%d",&coeffs[ncoeff]); if(n<1) break; }
    //for(n=0;n<ncoeff;n++) printf("coeff[%d]=%d\n",n,coeffs[n]);
    for(n=0;n<(ncoeff-1);n++) BUF[n]=0; // init status registers
    for(;!feof(fi);) {
      n=fscanf(fi,"%d",&x); if(n<1) break; // read input sample
      // *** filter function ***
                                        y = coeffs[0       ]*x + BUF[0]; // compute
output
      for(n=1;n<(ncoeff-1);n++) BUF[n-1] = coeffs[n       ]*x + BUF[n]; // update
status
                              BUF[ncoeff-2] = coeffs[ncoeff-1]*x;
      // *** filter function ***
      fprintf(fo,"%d\n",y); // write output sample
    }
    break;
  }
  if(fc!=NULL) fclose(fc); if(fi!=NULL) fclose(fi); if(fo!=NULL) fclose(fo);
  return 0;
}
```

FIR filter implementation test MATLAB® script

The script performs the same operations as the previous test script, but instead of calling the MATLAB filter() function, it calls the C implementation shown in the previous paragraphs: firI or firII. See Figure 4.

Figure 4. Frequency response of the FIR I (left) and FIR II implementation (right).



Test of firI.c

```
Fs=100; % sampling frequency
Q = 4096; % 2^12 bits for the coefficient mantissa
NFFT=2^12; % FFT resolution

% impulse response gives the coefficient set
in = zeros(1,NFFT); in(1)=Q;
h=fopen('fir_in.txt','wt'); fprintf(h,'%d\n',in); fclose(h);
system('firI fir_cw.txt fir_in.txt fir_out.txt');
h=fopen('fir_out.txt','rt'); out=fscanf(h,'%d\n'); fclose(h);
[Himp,fimp]=freqz(out/Q/Q,1,NFFT,Fs);

% filter noise and compute power spectrum
in = round((2*rand(1,NFFT)-1)*Q/4);
h=fopen('fir_in.txt','wt'); fprintf(h,'%d\n',in); fclose(h);
system('firI fir_cw.txt fir_in.txt fir_out.txt');
h=fopen('fir_out.txt','rt'); out=fscanf(h,'%d\n'); fclose(h);
NOVL=round(NFFT*0.9); win=hann(NFFT);
[Pn,fn]=pwelch(out/Q/Q,win,NOVL,NFFT,Fs);
```

```
% plot
figure; hold on;
plot(fn  ,10*log10(abs(Pn))+10*log10(Q),':');
plot(fimp,20*log10(abs(Himp)));
legend('noise','imp resp');
grid on; zoom on; title('FIR I frequency response test');
xlabel('frequency Hz'); ylabel('magnitude dB');
```

Test of firII.c

```
Fs=100; % sampling frequency
Q = 4096; % 2^12 bits for the coefficient mantissa
NFFT=2^12; % FFT resolution

% impulse response gives the coefficient set
in = zeros(1,NFFT); in(1)=Q;
h=fopen('fir_in.txt','wt'); fprintf(h,'%d\n',in); fclose(h);
system('firII fir_cw.txt fir_in.txt fir_out.txt');
h=fopen('fir_out.txt','rt'); out=fscanf(h,'%d\n'); fclose(h);
[Himp,fimp]=freqz(out/Q/Q,1,NFFT,Fs);

% filter noise and compute power spectrum
in = round((2*rand(1,NFFT)-1)*Q/4);
h=fopen('fir_in.txt','wt'); fprintf(h,'%d\n',in); fclose(h);
system('firII fir_cw.txt fir_in.txt fir_out.txt');
h=fopen('fir_out.txt','rt'); out=fscanf(h,'%d\n'); fclose(h);
NOVL=round(NFFT*0.9); win=hann(NFFT);
[Pn,fn]=pwelch(out/Q/Q,win,NOVL,NFFT,Fs);

% plot
figure; hold on;
plot(fn  ,10*log10(abs(Pn))+10*log10(Q),':');
plot(fimp,20*log10(abs(Himp)));
legend('noise','imp resp');
grid on; zoom on; title('FIR II frequency response test');
xlabel('frequency Hz'); ylabel('magnitude dB');
```

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
Design tip, DT0092, Lattice wave digital filter test and performance verification

Revision history

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

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.