Lecture 11 objectives are to:

- The Discrete Fourier Transform
- Windowing
- Use DFT to design a FIR digital filter

Discrete Fourier Transform, DFT

Input: N time samples

$\{a_n\} = \{a_0, a_1, a_2, \ldots, a_{N-1}\}$

Output: a set of N frequency bins

$\{A_k\} = \{A_0, A_1, A_2, \ldots, A_{N-1}\}$

$$A_k = \sum_{n=0}^{N-1} a_n W_N^{kn} \quad \text{where} \quad W_N = e^{-j2\pi/N}$$

$k=0,1,2,\ldots,N-1$

Inverse DFT

Input N frequency bins

$\{A_k\}=\{A_0, A_1, A2, \ldots, A_{N-1}\}$

Output time domain samples

$\{a_n\}=\{a_0, a_1, a_2, \ldots, a_{N-1}\}$

$$a_n = \frac{1}{N} \sum_{k=0}^{N-1} A_k W_N^{-kn} \quad \text{where} \quad W_N = e^{-j2\pi/N}$$

$n=0,1,2,\ldots,N-1$

- While the DFT deals only with samples and bins, with no concept of seconds and Hz, when looking at ADC samples spaced at intervals T (in sec)
- Frequency bin m represents components at $k*f_S/N$ (in Hz)
- The DFT resolution in Hz/bin is the reciprocal of the total time spent gathering time samples; i.e., 1/(NT)

```
  for(n = 0; n < 64; n++){
    data = OS_Fifo_Get();
    if(filterFlag) data = FIR(data);
    if(voltFlag) PlotData(data);
    x[n] = data&0xFFFF;
// real is 0 to 1023, imaginary part is 0
    }
    cr4_fft_64_stm32(y,x,64);
// y(k) has same units as x(n)

    for(k = 0; k < 32; k++){
    real = y[k]&0xFFFF;   // bottom 16 bits
    imag = y[k]>>16;      // top 16 bits
    mag = sqrt(real*real+imag*imag);
    if(FFTflag) LCD_Plot(mag);
```

if V is the DFT output magnitude in volts

$dB_{FS} = 20 \log_{10}(V/3)$; // full scale is 3.0 volts

This is code from Lab2 consumer showing how to run the FFT

```
    for(t = 0; t < 64; t++){ // collect 64 ADC samples
      data = OS_Fifo_Get();  // get from producer
      x[t] = data;           // real 0 to 1023, imaginary 0
    }
    cr4_fft_64_stm32(y,x,64);// complex FFT of ADC values
```

If you want to calculate the magnitude from this FFT, the top 16 bits of y has the imaginary part and the bottom half is real.

```
    for(t = 0; t < 32; t++){   // first half
      real = y[t]&0xFFFF;       // bottom 16 bits
      imag = y[t]>>16;          // top 16 bits
      mag[t] = sqrt(real*real+imag*imag);
    }
 http://users.ece.utexas.edu/~valvano/EE345M/sqrt.c
```

This code takes 1024 points, and plots 4 pixels per tick. This means there are 512/4=128 lines across the screen (like the cover of the book). It also uses the db full scale feature of the plotter

```
    for(t = 0; t < 1024; t++){ // collect 1024 ADC samples
      data = OS_Fifo_Get();  // get from producer
      x[t] = data;           // real 0 to 1023, imaginary 0
    }
    cr4_fft_1024_stm32(y,x,1024);  // complex FFT
    for(t = 0; t < 512; t++){   // first half
      real = y[t]&0xFFFF;       // bottom 16 bits
      imag = y[t]>>16;          // top 16 bits
      data = sqrt(real*real+imag*imag);
      ST7735_PlotdBfs(data);
      if((t%4)==3){
        RIT128x96x4PlotNext(); // 4 pixel per tick
      }
    }
    ST7735_PlotNext();   //128 ticks across screen
```

Applications

- Measure S/N ratio
- Identify noise
- DF design

Four or Five approximations

- Finite min
- Finite max (range = max-min)
- Precision (resolution is range/precision)
- Sampling rate
- Finite number of samples -> Spectral leakage

Inherent in the application of FFT or cross correlation in computer based systems is the need to operate on finite sequences. Even virtual memory has finite size, and most customers are not willing to wait for infinite time to get the results. The process of choosing a finite subsequence on which to operate is called windowing. It is critical to capture a reasonable window, because the data is actually considered as an infinite periodic signal. In other words, if we process the finite sequence

x(0), x(1), x(2), … x(N-1)

then the FFT or cross correlation will effectively be determined for the infinite sequence

…, x(0), x(1), x(2), … x(N-1), x(0), x(1), x(2), … x(N-1), x(0), x(1), x(2), … x(N-1),…

Figures 8.4a and 8.4b show an improperly chosen window. Notice that the infinite periodic signal does not accurately represent the original data.

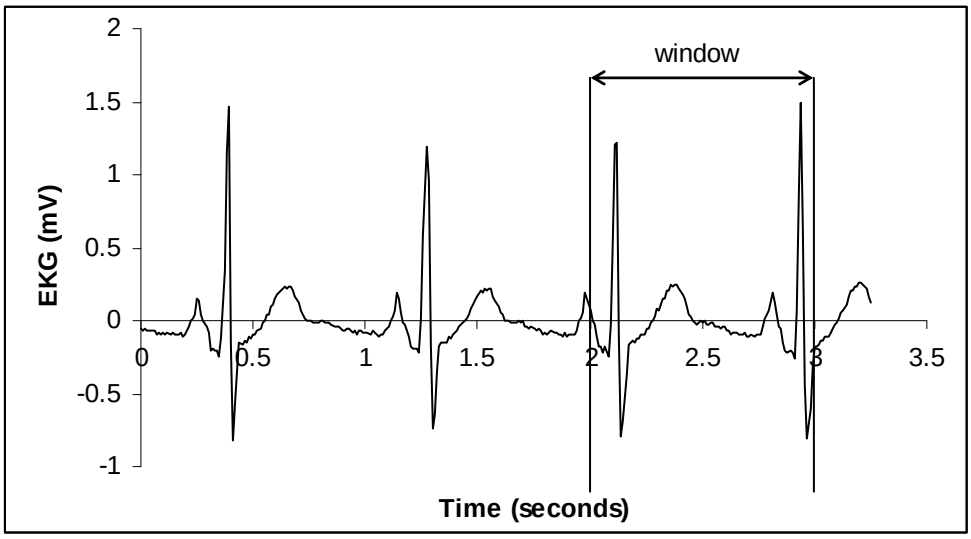


Figure 8.4a. Original data with window from 2 to 3 seconds

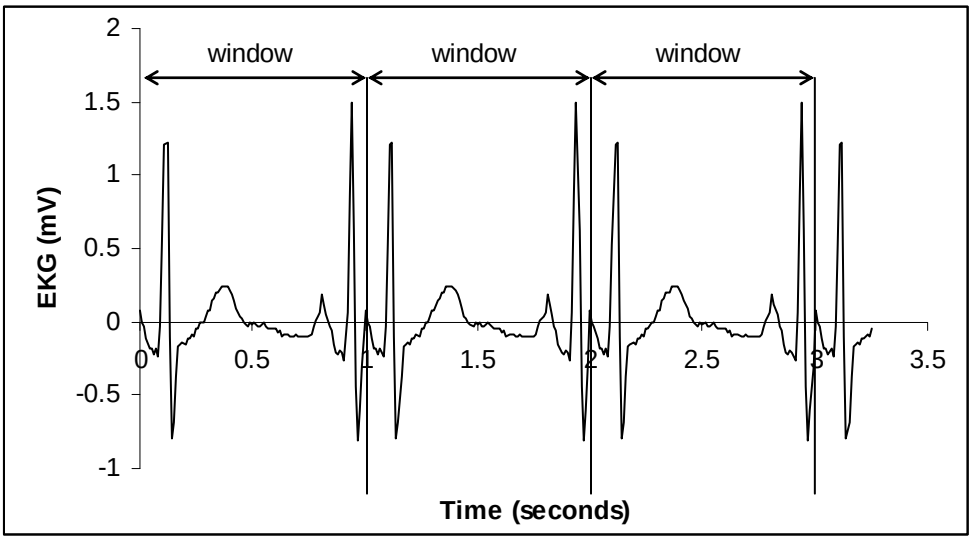


Figure 8.4b. Equivalent infinite periodic signal resulting from the window shown in Figure 2a.

Figures 8.5a and 8.5b show a properly chosen window. Notice that the infinite periodic signal accurately represents the shape of the original data, but the information about heart rate is lost.

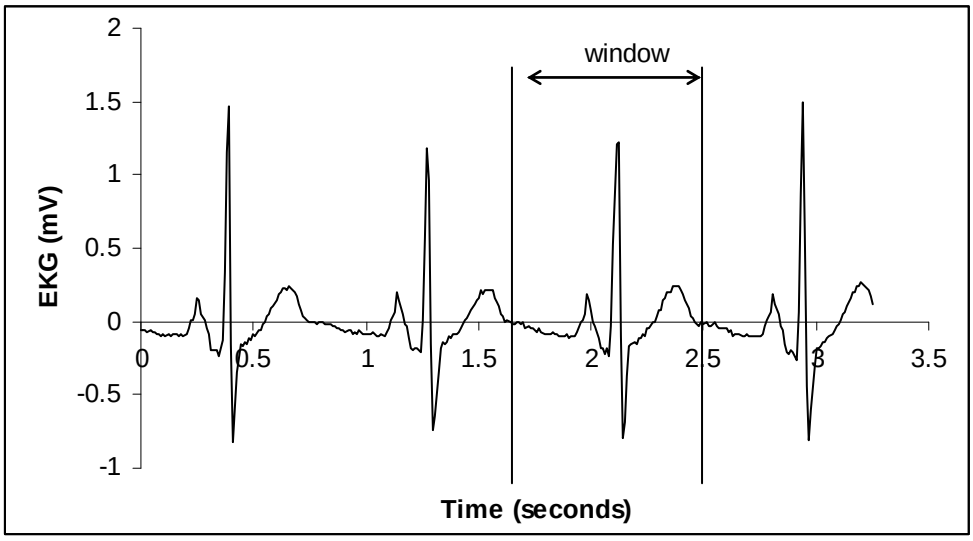


Figure 8.5a. Original data with window from 1.7 to 2.5 seconds

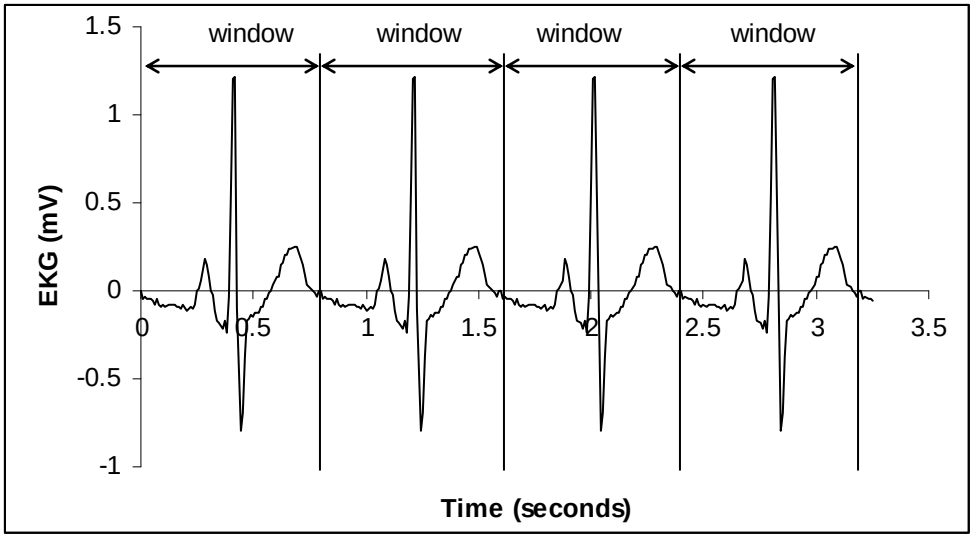


Figure 8.5b. Equivalent infinite periodic signal resulting from the window shown in Figure 8.5a.

If the window has multiple cycles, then there will be multiple correlations. To prevent multiple matches, we can choose a window with one cycle (like Figure 8.5), or we can a mask to the data, as shown in Figure 8.6. This is a trapezoidal mask, but other mask shapes can be used (rectangle, sine-wave, exponential). This method allows us to select a window without having to specify the sequence length. Notice that the window shown in Figure 8.6 could be used to study the shape of just the QRS wave, without including the P and T waves.

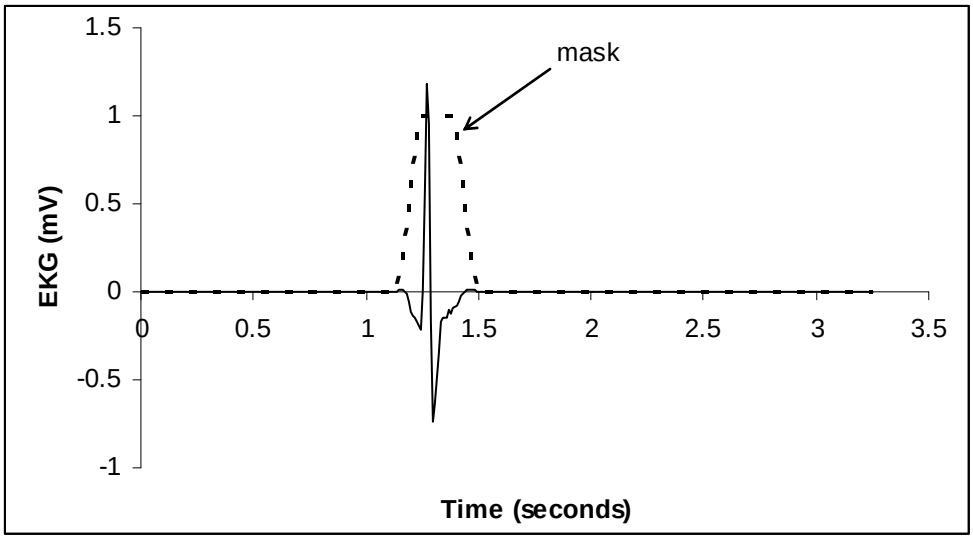
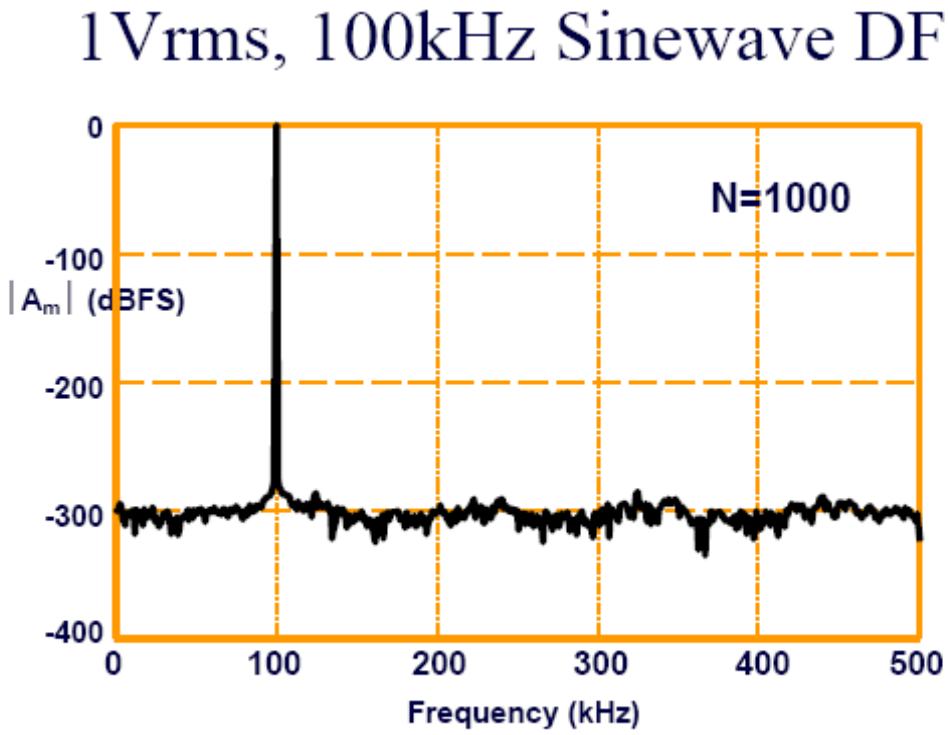


Figure 8.6. A window is created by multiplying the original data with a mask.

Windowing

- Spectral leakage can be virtually eliminated by "windowing" time samples prior to the DFT
  - Windows taper smoothly down to zero at the beginning and the end of the observation window
  - Time samples are multiplied by window coefficients on a sample-by-sample basis
- Windowing sinewaves places the window spectrum at the sinewave frequency
  - Convolution in frequency



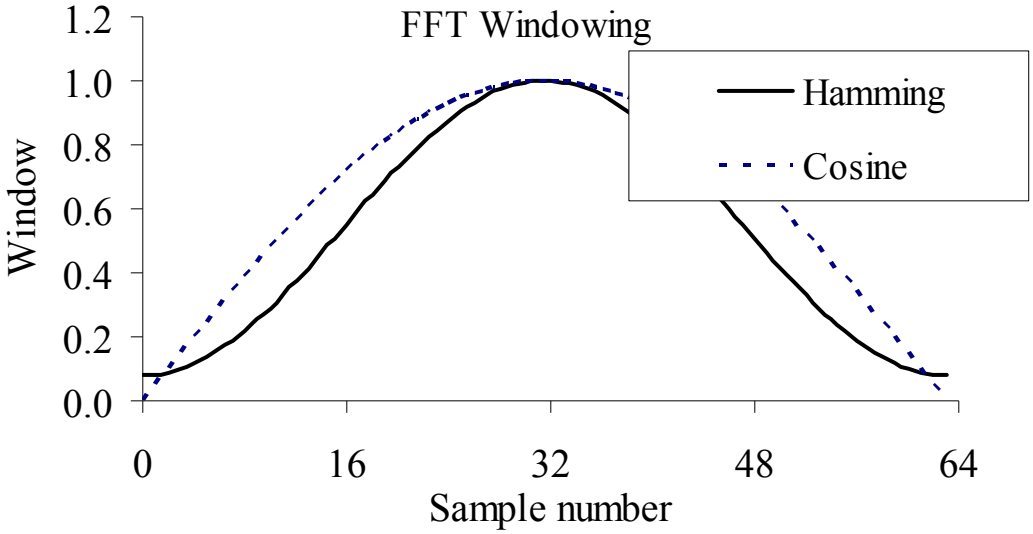
Reference ©2002 Eric Swanson, Mixed Signal Class

$$\frac{1}{N}\sum_{n=0}^{N-1}|w(k)|^2 = 1$$

Window coefficients w(k) will be normalized so that the rms value of the time samples is the same before and after windowing; that is,

Hamming	$w(k) = 0.54-0.46*\cos(2\pi k/(N-1))$
Hann	$w(k) = (\sin(\pi k/(N-1)))^2$
Cosine	$w(k) = \sin(\pi k/(N-1))$
Triangle	$w(k) = (2/N)(N/2 - \lvert k - (N-1)/2\rvert)$

Reference ©2002 Eric Swanson, Mixed Signal Class



FIR digital filter design

You will process the real-time data in the foreground by implementing a FIR digital filter. After you have chosen the sampling rate (e.g., 44 kHz) you next will **choose a FIR filter length** (e.g., N=64). The ratio f_s/N (e.g., 44 kHz/64 = 687 Hz) will determine the frequency resolution of the FIR filter design. Let H(z) be the desired filter

gain transfer function. Table 1 gives an example desired frequency response. The magnitude of H(k) is selected to implement the **desired gain versus frequency response**. In order to preserve the shape of the audio signals, we will implement linear phase. For frequencies above ½ fs, we make H(k) be the complex conjugate of the N-k term. This will guarantee that the inverse DFT of H(z) will yield real results. The desired filter response, plotted as blue dots in Figure 1.

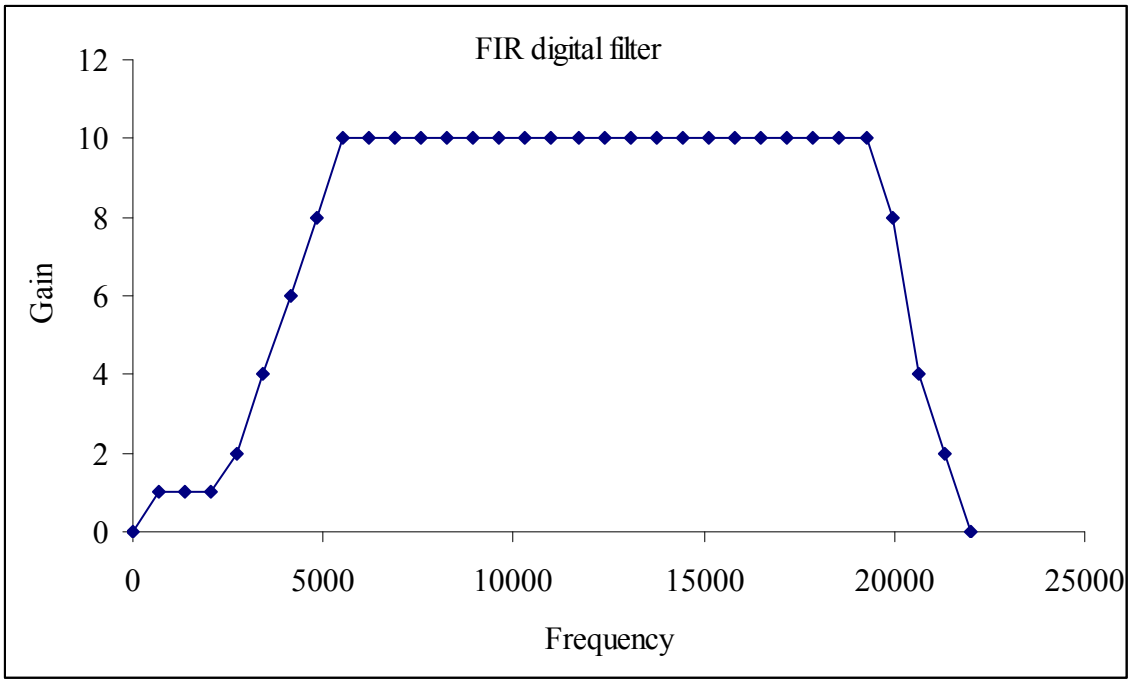


Figure 1. Desired filter response. This is H.

k	f (Hz)	Mag(H(k))	Angle(H(k))
0	0.0	0	0.00
1	687.5	1	-2.45
2	1375.0	1	-4.91
3	2062.5	1	-7.36
4	2750.0	2	-9.82
5	3437.5	4	-12.27
6	4125.0	6	-14.73
7	4812.5	8	-17.18
8	5500.0	10	-19.63
9	6187.5	10	-22.09
10	6875.0	10	-24.54
11	7562.5	10	-27.00
12	8250.0	10	-29.45
13	8937.5	10	-31.91
14	9625.0	10	-34.36
15	10312.5	10	-36.82
16	11000.0	10	-39.27
17	11687.5	10	-41.72
18	12375.0	10	-44.18
19	13062.5	10	-46.63
20	13750.0	10	-49.09
21	14437.5	10	-51.54

k	f (Hz)	Mag(H(k))	Angle(H(k))
32	22000.0	0	-78.54
33	-21312.5	2	76.09
34	-20625.0	4	73.63
35	-19937.5	8	71.18
36	-19250.0	10	68.72
37	-18562.5	10	66.27
38	-17875.0	10	63.81
39	-17187.5	10	61.36
40	-16500.0	10	58.90
41	-15812.5	10	56.45
42	-15125.0	10	54.00
43	-14437.5	10	51.54
44	-13750.0	10	49.09
45	-13062.5	10	46.63
46	-12375.0	10	44.18
47	-11687.5	10	41.72
48	-11000.0	10	39.27
49	-10312.5	10	36.82
50	-9625.0	10	34.36
51	-8937.5	10	31.91
52	-8250.0	10	29.45
53	-7562.5	10	27.00

22	15125.0	10	-54.00	54	-6875.0	10	24.54
23	15812.5	10	-56.45	55	-6187.5	10	22.09
24	16500.0	10	-58.90	56	-5500.0	10	19.63
25	17187.5	10	-61.36	57	-4812.5	8	17.18
26	17875.0	10	-63.81	58	-4125.0	6	14.73
27	18562.5	10	-66.27	59	-3437.5	4	12.27
28	19250.0	10	-68.72	60	-2750.0	2	9.82
29	19937.5	8	-71.18	61	-2062.5	1	7.36
30	20625.0	4	-73.63	62	-1375.0	1	4.91
31	21312.5	2	-76.09	63	-687.5	1	2.45

Table 1. Desired filter response for one patient that compensates for hearing loss. This is H.

Let x(n) be the input (read from the ADC) and X(z) be the input in the frequency domain. Let y(n) be the FIR filter output, and let Y(z) be the FIR filter output in the frequency domain.

$Y(z) = H(z)\, X(z)$

$y(n) = \text{IFFT}\ \{ H(z)\ \text{FFT}\{x(t)\}\ \}$

Take Inverse FFT of the desired gain to get N=64 FIR filter coefficients. Because the negative frequencies in Table 1 are complex conjugates of the positive frequencies, h(n) will be real.

h(n) = IFFT{H(z)} =

1.9113, -2.7895, 0.6680, -4.0131, 0.5350, -3.0053, 1.7441, -2.0000, 1.7441, -3.0053, 0.5350, -4.0131, 0.6680,
-2.7895, 1.9113, -1.1716, 0.9273, -1.6178, -2.6059, -2.2019, -4.5219, -0.0708, -2.9944, 2.0000, -1.4968, -1.0009,
-0.8678, -6.6854, 5.8260, -7.5759, 22.0888, -6.8284, 34.2008, -22.7620, 20.6036, -67.0996, -19.6370, -121.1778,
-56.3812, 494.0000, -56.3812, -121.1778, -19.6370, -67.0996, 20.6036, -22.7620, 34.2008, -6.8284, 22.0888,
-7.5759, 5.8260, -6.6854, -0.8678, -1.0009, -1.4968, 2.0000, -2.9944, -0.0708, -4.5219, -2.2019, -2.6059, -1.6178, 0.9273, -1.1716

Scale to make fixed point coefficients h0 to h63, e.g., 1.9113 ≈ 984/489

```
const long h[64]={489,-714,171,-1027,137,-769,446,-512,446,-769,137,-1027,171,
    -714,489,-300,237,-414,-667,-564,-1158,-18,-767,512,-383,-256,-222,
    -1711,1491,-1939,5655,-1748,8755,-5827,5275,-17177,-5027,-31022,-14434,126464,-14434,
    -31022,-5027,-17177,5275,-5827,8755,-1748,5655,-1939,1491,-1711,-222,-256,-383,
    512,-767,-18,-1158,-564,-667,-414,237,-300};
```

Multiplication in the frequency domain is equivalent to convolution in the time domain. The FIR filter is the convolution of the data with the inverse transform of the desired filter.

$y(n) = h(n) * x(n) = x(n) * h(n)$ (* means convolution here)

$y(n) = \text{sum}\ [h(i) \cdot x(n-i)]$ as i goes from $-\infty$ to $+\infty$. (· means multiplication here)

Because there are a finite number of h(n) terms, the convolution is a finite sum

y[i]= (h[0]*x[i]+h[1]*x[i-1]+h[2]*x[i-2]+…+h[63]*x[i-63])/256; // * means multiplication here

3.6.1 MUL, MLA, and MLS

Multiply, multiply with accumulate, and multiply with subtract, using 32-bit operands, and producing a 32-bit result.

Syntax

```
MUL{S}{cond} {Rd,} Rn, Rm ; Multiply

MLA{cond} Rd, Rn, Rm, Ra ; Multiply with accumulate

MLS{cond} Rd, Rn, Rm, Ra ; Multiply with subtract
```

where:

- '*cond*' is an optional condition code (see *Conditional execution on page 56*)
- '*S*' is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation (see *Conditional execution on page 56*).
- '*Rd*' is the destination register. If *Rd* is omitted, the destination register is *Rn*
- '*Rn*', '*Rm*' are registers holding the values to be multiplied
- '*Ra*' is a register holding the value to be added to or subtracted from

What is the effect caused by sampling jitter?