Digital Filters

- Digital signal sampled from continuous analog signal $x_c(t)
  - $x(n) = x_c(nT)$ with $-\infty < n < +\infty$
  - finite precision, finite sampling frequency & frequency range
- Causal digital filter
  - calculates $y(n)$ from $y(n-1)$, $y(n-2)$,... and $x(n)$, $x(n-1)$, $x(n-2)$,...
  - not future data (e.g., $y(n+1)$, $x(n+1)$ etc.)
- Linear filter is constructed from a linear equation
- Nonlinear filter is constructed from a nonlinear equation
  - E.g. median filter
- Finite impulse response filter (FIR)
  - relates $y(n)$ only in terms of $x(n)$, $x(n-1)$, $x(n-2)$,...
    - $y(n) = (x(n) + x(n-3))/2$
- Infinite impulse response filter (IIR)
  - relates $y(n)$ in terms of both $x(n)$, $x(n-1)$,..., and $y(n-1)$, $y(n-2)$,...
    - $y(n) = (113 \times x(n) + 113 \times x(n-2) - 98 \times y(n-2))/128$
**Transforms**

- **Time vs. frequency domain**
  - Z-Transform
    \[ X(z) = Z[x(n)] = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \]
  - Laplace Transform

![Diagram showing the relationship between time and frequency domains with Z-Transform and Laplace Transform](image)

**Gain and Phase Response**

- **Analog system**
  - Gain \( |H(s)| \) at \( s = j 2\pi f \), for all frequencies, \( f \)
  - Phase \( \angle(H(s)) \) at \( s = j 2\pi f \)

- **Digital system**
  - Transform, \( H(z) = Y(z)/X(z) \) from DC to \( 1/2 f_s \)
    - One can show that: \( Z[x(n-m)] = z^{-m} Z[x(n)] = z^{-m} X(z) \)
      - E.g., if \( X(z) = Z[x(n)] \), \( Z[x(n-2)] = z^{-2}X(z) \)
  - Let
    - \( z(f) = e^{j2\pi f f_s} = \cos(2\pi f f_s) + j \sin(2\pi f f_s) \) for \( 0 \leq f < 1/2 f_s \)
    - \( H(f) = H(z(f)) = a + bj \), where \( a \) and \( b \) are real numbers
    - Gain \( |H(f)| = \sqrt{a^2+b^2} \) as \( f \) varies from 0 to \( 1/2 f_s \)
    - Phase \( \angle(H(f)) = \tan^{-1}(a/b) \), \( f \) from 0 to \( 1/2 f_s \)
Filter Example (1)

- Low-Q 60 Hz notch filter
  - \( y(n) = (x(n)+x(n-3))/2 \)
- Z-Transform
  - \( Y(z) = (X(z) + z^{-3}X(z))/2 \)
- Rewrite in the form \( H(z) = Y(z)/X(z) \)
  - \( H(z) \equiv Y(z)/X(z) = \frac{1}{2} (1 + z^{-3}) \)
- Determine gain and phase response
  - \( H(f) = \frac{1}{2} (1 + e^{-j6\pi f/f_s}) = \frac{1}{2} (1 + \cos(6\pi f/f_s) - j \sin(6\pi f/f_s)) \)
  - Gain \( \equiv |H(f)| = \frac{1}{2} \sqrt{(1 + \cos(6\pi f/f_s))^2 + \sin(6\pi f/f_s)^2} \)
  - Phase \( \equiv \angle(H(f)) = \tan^{-1}(-\sin(6\pi f/f_s)/(1 + \cos(6\pi f/f_s))) \)

Filter Example (2)

- Gain vs. frequency response
Filter Example (3)

- Multiple Access Circular Queue (MACQ)

```c
short x[4]; // MACQ
void ADC3_Handler(void){
    short y;
    ADC_ISC_R = ADC_ISC_IN3; // ack ADC sequence 3 completion
    x[3] = x[2]; // shift data
    x[2] = x[1]; // units, ADC sample 0 to 4095
    x[1] = x[0];
    x[0] = ADC_SSFIFO3_R & ADC_SSFIFO3_DATA_M; // 0 to 4095
    y = (x[0]+x[3])/2; // filter output
    Fifo_Put(y); // pass to foreground
}
```

### Pointer-Based MACQ

```c
unsigned short x[32]; // two copies
unsigned short *Pt; // pointer to current
unsigned short Sum; // sum of last 16 samples
void LPF_Init(void){
    Pt = &x[0]; Sum = 0;
}
// calculate one filter output
// average previous 16 samples
// called at sampling rate
// Input: new ADC data
// Output: filter output, DAC data
unsigned short LPF_Calc(unsigned short newdata){
    Sum = Sum - *(Pt+16); // sub 16 samples ago
    if(Pt == &x[0]){ // wrap
        Pt = &x[16];
    } else{
        Pt--; // make room for data
    }
    *Pt = *(Pt+16) = newdata; // two copies
    return Sum/16;
}
```
Filter Design

<table>
<thead>
<tr>
<th>Analog condition</th>
<th>Digital condition</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero near ( s = 2j\pi f_s )</td>
<td>pole near ( z = e^{j2\pi f_s} )</td>
<td>low gain near the zero</td>
</tr>
<tr>
<td>pole near ( s = 2j\pi f_s )</td>
<td>zero near ( z = e^{j2\pi f_s} )</td>
<td>high gain near the pole</td>
</tr>
<tr>
<td>zeros in conjugate pairs</td>
<td>poles in conjugate pairs</td>
<td>the output ( y(t) ) is real</td>
</tr>
<tr>
<td>poles in left half plane</td>
<td>poles inside unit circle</td>
<td>stable system</td>
</tr>
<tr>
<td>poles in right half plane</td>
<td>poles outside unit circle</td>
<td>unstable system</td>
</tr>
<tr>
<td>pole near a zero</td>
<td>pole near a zero</td>
<td>high Q response</td>
</tr>
</tbody>
</table>

60Hz digital notch filter, \( f_s = 480 \) Hz

\[ \theta = \pm 2\pi \cdot \frac{60}{f_s} = \pm \pi/4 \]

Zeros on unit circle (gain=0 at 60 Hz):

\[ z_1 = \cos(\theta) + j \sin(\theta), \quad z_2 = \cos(\theta) - j \sin(\theta) \]

Poles next to the zeros, just inside the unit circle (flat pass band away from 60 Hz):

\[ p_1 = \alpha z_1, \quad p_2 = \alpha z_2 \quad \text{where} \quad 0 < \alpha < 1 \]

IIR Filter (1)

- **Transfer function**

\[
H(z) = \frac{k}{\prod_{i=1}^{2} (z-p_i)} = \frac{(z-z_1)(z-z_2)}{(z-p_1)(z-p_2)} = \frac{1 - \cos(\theta)z^{-1} + z^{-2}}{1 - 2\cos(\theta)z^{-1} + z^{-2}}
\]

- **With \( f_s = 480 \)Hz and \( \alpha = 7/8 \)**

\[
H(z) = \frac{1 + z^{-2}}{1 + \frac{49}{64} z^{-2}} \quad y(n) = x(n) + x(n-2) - \left(49y(n-2)\right)/64
\]

- **At \( z = 1 \), this reduces to**

\[
\text{DC Gain} = \frac{2}{1 + \frac{49}{64}} = \frac{128}{64 + 49} = \frac{128}{113}
\]

For DC Gain of 1: \( y(n) = (113x(n) + 113x(n-2)) - 98y(n-2))/128 \)
IIR Filter (2)

long x[3]; // MACQ for the ADC input data
long y[3]; // MACQ for the digital filter output
void ADC3_Handler(void)
{
  ADC ISC R = ADC ISC IN3; // ack ADC completion
  x[2] = x[1]; x[1] = x[0]; // shift data
  y[2] = y[1]; y[1] = y[0];
  x[0] = ADC SS FIFO3 R & ADC SS FIFO3_DATA M;
  y[0] = (113 * (x[0] + x[2]) - 98 * y[2])/128; // filter output
  Fifo_PUT((short)y[0]);
}

Notch filter “Q”:

\[ Q = \frac{f_c}{\Delta f} \]

\(\Delta f\): frequency range where gain is below 0.707 of the DC gain

Lab4 Spectrum Analyzer
Discrete Fourier Transform (DFT)

- Convert time to frequency domain
  
  **Input:** N time samples  
  **Output:** a set of N frequency bins
  
  \[
  \{a_n\} = \{a_0, a_1, a_2, \ldots, a_{N-1}\} \quad \text{and} \quad \{A_k\} = \{A_0, A_1, A_2, \ldots, A_{N-1}\}
  \]
  
  \[
  A_k = \sum_{n=0}^{N-1} a_n W_N^{-kn}, \quad \text{where} \quad W_N = e^{-j2\pi N}, \quad k=0,1,2,\ldots,N-1
  \]

- **Inverse DFT**
  
  **Input:** a set of N frequency bins  
  **Output:** N time samples
  
  \[
  \{A_k\} = \{A_0, A_1, A_2, \ldots, A_{N-1}\} \quad \text{and} \quad \{a_n\} = \{a_0, a_1, a_2, \ldots, a_{N-1}\}
  \]
  
  \[
  a_n = \frac{1}{N} \sum_{k=0}^{N-1} A_k W_N^{-kn}, \quad \text{where} \quad W_N = e^{-j2\pi N}, \quad n=0,1,2,\ldots,N-1
  \]

DFT Properties

- **Parameters**
  
  - While the DFT deals only with samples and bins, assume data is ADC samples spaced at intervals T=1/fs (in sec)
  
  - Frequency bin k 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)
DFT Applications

- Applications
  - Measure S/N ratio
  - Identify noise
  - Filter design

- Four or five approximations
  - Finite min, max, range (max-min)
  - Finite precision & resolution (range/precision)
  - Sampling rate
  - Finite number of samples (spectral leakage)

Spectral Leakage

- Finite sequences, assumed to be periodic
Windowing (1)

- 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
- Window coefficients \( w(k) \)
  - Normalized so that the RMS value of the time samples is the same before and after windowing
  \[
  \frac{1}{N} \sum_{n=0}^{N-1} |w(k)|^2 = 1
  \]

Windowing (2)

- Various windowing functions
  - Hamming \( w(k) = 0.54-0.46\cos\left(\frac{2\pi k}{N-1}\right) \)
  - Hann \( w(k) = \left(\sin\left(\frac{\pi k}{N-1}\right)\right)^2 \)
  - Cosine \( w(k) = \sin\left(\frac{\pi k}{N-1}\right) \)
  - Triangle \( w(k) = \frac{2}{N}\left(\frac{N}{2} - |k - (N-1)/2|\right) \)
Fast Fourier Transform (FFT)

- Faster version of the Discrete Fourier Transform (DFT)
  - FFT spectrum of a cosine with a frequency of \(0.1f_s\)
    - \(f_s = 10\text{kHz}\)
    - Cosine freq = 1kHz
  - Interested region from 0 to \(f_s/2\)
    - Symmetric around \(f_s/2\)

FFT Library (1)

```
for(t = 0; t < 64; t++) {
  // collect 64 ADC samples
  data = OS_Fifo_Get();  // get from producer
  x[t] = data & 0xFFFF;  // real 0 to 1023, imaginary 0
}

cr4_fft_64_stm32(y,x,64);  // complex 64-point FFT

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);
  LCD_Plot(mag);
}
```

http://www.ece.utexas.edu/~valvano/EE345M/sqrt.c
FFT Library (2)

```c
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){  // 4 pixel per tick
        ST7735_PlotNext();
    }
}
ST7735_PlotNext();  // 128 ticks across screen
```

if \( V \) is the FFT output magnitude in volts
\[ \text{dBFS} = 20 \log_{10}(V/3); \]
// full scale is 3.0 volts

Alternative FIR Filter Design (1)

• Specify gain versus frequency
• Specify phase versus frequency

![FIR digital filter diagram]
Alternative FIR Filter Design (2)

- \( Y(z) = H(z) \times X(z) \)
- \( h(n) = \text{IFFT} \{ H(z) \} \)
- Convolution
  - \( y(n) = h(n) \ast x(n) \)
- Constants \( h_0, h_1, \ldots, h_{N-1} \)
- \( y(n) = h_0 \cdot x(n) + h_1 \cdot x(n-1) + \ldots + h_{N-1} \cdot x(n-(N-1)) \)
- \( N \) multiplies, \( N-1 \) additions per sample

How to Choose Sampling Rate

- Nyquist Rate
- Limitation of display
- Limitation of processor
- Limitation of RAM
- Limitation of human eyes and ears
- Limitation of communication channel
How to Choose Number of Samples

- Frequency resolution = $f_s/N$
  - Increase in $N$ results in better frequency resolution
  - However, increase in $N$ leads to a bigger MACQ buffer and more multiplies and additions
- Does not need to be a power of 2
  - DFT calculated once, off line

Design Process (1)

- Specify desired gain and phase, 0 to $\frac{1}{2} f_s$
  - $k$ goes from 0 to $N/2$ ($f = k f_s/N$)
  - $H(k)$ is complex
  - $|H(k)|$ is gain
  - angle($H(k)$) is phase
- For $\frac{1}{2} f_s$ to $f_s$
  - $H(N-k)$ is complex conjugate of $H(k)$
  - Poles and zeros are in complex conjugate pairs
Design Process (2)

- Take IDFT of $H(k)$ to yield $h(n)$
  - $n$ goes from 0 to $N-1$
  - $h(n)$ will be real, because $H(k)$ symmetric
- The digital filter is
  - $y(n) = h_0 \cdot x(n) + h_1 \cdot x(n-1) + \ldots + h_{N-1} \cdot x(n-(N-1))$

Binary Fixed Point Notation

- Binary fixed-point is faster than decimal fixed-point
- $Q^n$ number (16 bit)
  - $n$: specifies the resolution = $2^{-n}$
  - $16-n$: specifies range
- Eg: 10.450 (unsigned number) with $n=11$?
- How is this number stored as an integer?
  - Value = Integer/2048
  - $10.450 = 01010.1110011010$
Example

- Open FIRdesign51.xls
- Change sampling rate to 10,000 Hz
- Adjust red desired gain to make BPF
  - Pass 2 to 4 kHz
  - Look at sharp corner versus round corner
- Notice linear phase
- Copy 51 coefficients into software

2kHz to 4kHz BPF
FIR Filter SW Design

```c
const long h[51]={-3,-9,4,5,0,17,5,-20,-5,-7,-22, 24,41,-8,2,1,-74,-31,71,20,33,125,-19,-350,67, 462,67,-350,-119,125,33,20,71,-31,-74,1,2,-8,41, 24,-22,-7,-5,-20,5,17,0,5,4,-9,-3};
static unsigned int n=50;   // 51,52,... 101
short Filter(short data){unsigned int k;
static long x[102];   // this MACQ needs twice
long y;
n++;
if(n==102) n=51;
x[n] = x[n-51] = data;   // two copies of new data
y = 0;
for(k=0;k<51;k++){
   y = y + h[k]*x[n-k];   // convolution
}
y = y/256;   // fixed point
return y;
}
```

Circular Buffering

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Filter Coefficient Array h[]</th>
<th>Circular Buffer Array xcirc[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>h[0]</td>
<td>x[n - newest]</td>
</tr>
<tr>
<td>1</td>
<td>h[1]</td>
<td>x[n - newest + 1]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>newest</td>
<td>x[n]</td>
<td>x[n - N + 1]</td>
</tr>
<tr>
<td>oldest</td>
<td>x[n - N + 2]</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N - 2</td>
<td>h[N - 2]</td>
<td>x[n - newest - 2]</td>
</tr>
<tr>
<td>N - 1</td>
<td>h[N - 1]</td>
<td>x[n - newest - 1]</td>
</tr>
</tbody>
</table>

"Communication system design using DSP algorithms" by Steven A. Tretter (Chapter 3, page 73)
Optimization

- Pointer implementations of MACQ faster
- Do not try and shift the data
- Convolution $x[n] \ast h[n]$ takes $N$ multiplies, $N-1$ additions per sample
  - Can be optimized to $N/2$ multiplies
  - Coefficients are symmetric
- Assembly optimization with MLA
  - Multiply with accumulate