Algorithm Mapping

**DSP Implementation & Elementary Functions**
- Improvement by successive approximation

**First guess and successive improvements**

**Example, approximating Reciprocal: Division**

RCPS and RCPDP instructions in C67X return the correct exponent part + 8 bit accurate mantissa part (mantissa error is less than \(2^{-12}\))

RCPS (DP).unit src, dst ; .unit = .S1, .S2 ; single cycle (double cycle DP)

**Newton-Raphson Algorithm** to solve \(f(x) = 0\) for \(x\)

\[
f'(x) = 0 \rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}
\]

**Calculating reciprocal of \(b\) → \(b^{-1}/x = 0\)** See code example enclosed … division.rar

\[
x_{n+1} = x_n - (b - 1/x_n)/(1/x_n^2) = x_n(2 - b/x_n)
\]

By Each iteration the number of accurate bits doubles → 1 iteration 16 bit, 2 iterations full 23 bits in single precision

\[\Delta_{n+1} = -\Delta_n^2 \leftarrow \Delta_n = bx_n - 1\]

→ 3 iterations (16...32...) 52 bits in RCPDP

**Algorithm Mapping**

**DSP Implementation & Elementary Functions**
- Improvement by successive approximation...

**Classical division using 'subc' conditional subtraction**
- subc .unit src1, src2, dst
  - unit: .L1, .L2
- ABS is needed if dividend or divisor are negative
- sign of quotient = sign multiplication
  - if (src1 - src2 ≥ 0)
  - sign of remainder = sign of dividend

Integer division if \(x > y\) → repeat subc 16 times → 16 bit results

Fractional division if \(x < y\) → repeat subc 15 times → Q15 bit results

Example: \(33 / 5\)

\[
\begin{array}{c|c}
\text{HIGH register} & \text{LOW register} \\
\hline
\text{0000000000000001} & \text{0000000010000001} \\
-101 & -101 \\
110 & 11
\end{array}
\]

**Algorithm Mapping**

**DSP Implementation & Elementary Functions**
- Improvement by successive approximation...

**For fixed point it must be done in Q13 because of the value 'two' and addition in the formula ...**

\[\text{mvk} \quad 16384, \text{two} \quad \text{See code example enclosed ... division.rar}\]

\[\text{shr} \quad b, 2, b\]

\[\text{sub} \quad \text{two, b, br} \quad \text{1st iteration initial seed = 1}\]

\[\text{mpy} \quad b, \text{br, brb}\]

\[\text{shr} \quad \text{brb, 13, br}\]

\[\text{mpy} \quad \text{br, br, brb}\]

\[\text{shr} \quad \text{brb, 13, br}\]

\[\text{mpy} \quad \text{br, br, brb}\]

\[\text{shr} \quad \text{brb, 13, br}\]

**Example 2, Square root reciprocal of \(a\):**

RSQRSP (DP) instruction in C67X returns the correct exponent part + 8 bit accurate mantissa part (mantissa error is less than \(2^{-12}\))

\[x_{n+1} = x_n(3 - a/x_n^2)/2\]

**Algorithm Mapping**

**DSP Implementation & Elementary Functions**
- Improvement by successive approximation...

<table>
<thead>
<tr>
<th>HIGH register</th>
<th>LOW register</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000000000</td>
<td>0000000010000001</td>
<td>(1) Dividend is loaded into register. This divisor is left-shifted 15 and subtracted from register. The subtraction is negative, so discard the result and shift the register left one bit.</td>
</tr>
<tr>
<td>-10</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>0011111111111111</td>
<td>0000000000000000</td>
<td>(2) 2nd subtract produces negative answer, so discard result and shift register (dividend) left.</td>
</tr>
<tr>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>0011111111111111</td>
<td>0000000000000000</td>
<td>(14) 14th SUBC command. The result is positive. Shift result left and replace LSB with 1.</td>
</tr>
<tr>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>0011111111111111</td>
<td>0000000000000000</td>
<td>(15) Result is again positive. Shift result left and replace LSB with 1.</td>
</tr>
<tr>
<td>0000000000000000</td>
<td>0000000000000000</td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>0011111111111111</td>
<td>0000000000000000</td>
<td>(16) Last subtract, negative answer, so discard result and shift register left.</td>
</tr>
</tbody>
</table>

\[\text{Answer reached after 16 SUBC instructions.}\]
### Algorithm Mapping...

#### DSP Implementation & Elementary Functions...

**Polynomial Approximation**
- **Least Square Approximation** → MATLAB polyfit → \( p = \text{polyfit}(x,y,N) \)
  \[
  f(x) - p(x) = \int_a^b w(x)[f(x) - p(x)]^2 \, dx
  \]
  Choose a set of orthogonal degree \( m \) polynomials \( T_i \), such that
  \[
  w(x) = 1/\sqrt{1 - x^2}, \quad [a,b] = [-1,1],
  \]
  \[
  T_i(x) = 1, \quad a_i = 1/2, \quad \text{otherwise}.
  \]

**Legendre, Chebyshev, Jacobi, Laguerre, ...**

Chebyshev Theorem: For \( p_n \) being minimax polynomial approximation of \( f \) over \( [a,b] \), there exist at least \( n+2 \) numbers, \( x_i \), such that:
  \[
  a \leq x_0 \leq x_1 \leq ... \leq x_{n+1} \leq b
  \]
  \[
  p_n(x_i) - f(x_i) = (-1)^i \| p_n(x_i) - f(x_i) \| = \| f - p_n \| \quad i = 0, ..., n + 1
  \]

- **Least Maximum Approximation** (Minimax Approximation)
  \[
  f(x) - p_n(x) = \| f(x) - p_n(x) \| = \max |w(x)| f(x) - p_n(x)
  \]

Weierestrauss Theorem: For any continuous \( f \) over \( [a,b] \), there exist at least \( n+2 \) numbers, \( x_i \), such that:
  \[
  a \leq x_0 \leq x_1 \leq ... \leq x_{n+1} \leq b
  \]
  \[
  p_n(x_i) - f(x_i) = (-1)^i \| p_n(x_i) - f(x_i) \| = \| f - p_n \| \quad i = 0, ..., n + 1
  \]

**Example:** \( \exp(-x^2) \) approximated by degree 3 minimax polynomial, \([-0.3, 0.3]\]

### Algorithm Mapping...

#### DSP Implementation & Elementary Functions...

**Polynomial Approximation... minimax**

Example: \( \exp(-x^2) \) approximated by degree 3 minimax polynomial, \([-0.3, 0.3]\]

- Minimax:
  \[
  p^*(x) = a_0 + a_1x + a_2x^2
  \]
  \[
  a_0 = 0.98903973, \quad a_1 = 1.13018381, \quad a_2 = 0.55404091
  \]
  \[
  x_0 = -0.43695806, \quad x_1 = 0.56005776, \quad \varepsilon = 0.04501739
  \]

- Chebyshev: use recursive formula and 'polyval' function in MATLAB

\[
T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2 - 1
\]

\[
a_0 = 3.977463261/\pi, \quad a_1 = 1.775499689/\pi, \quad a_2 = 0.426463882/\pi
\]

\[
p_2(x) = 0.5429906776x^2 + 1.130318208x + 0.9945705392
\]
Algorithm Mapping...

DSP Implementation & Elementary Functions...

- **Polynomial Approximation...** Compare... Chebychev...

```matlab
clc; close all
f=@(x) exp(x)./(1 - x.^2).^(1/2);
a0 = quad(f, -1,1)/pi;
f=@(x) x.*exp(x)./(1 - x.^2).^(1/2);
a1 = quad(f, -1,1)/pi*2;
f=@(x) (2.*x.^2-1).*exp(x)./(1 - x.^2).^(1/2);
a2 = quad(f, -1,1)/pi*2;
syms x
p = a0 + a1*x + a2*(2*x^2-1);
sym2poly(p)
f_handle = ezplot(exp(x),[-2, 2]); hold set(f_handle,'color','r'); ezplot(p, [-2,2]); grid
figure f_handle2 = ezplot(p-exp(x), [-1,1]); set(f_handle2,'color','m'); grid
ans = 
0.5430    1.1303    0.9946
```

Sharif University of Technology, EE Department, Digital Signal Processors Course Notes, imangh@sharif.edu

Algorithm Mapping...

DSP Implementation & Elementary Functions...

- **Polynomial Approximation...** Compare... Chebychev and Taylor

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Taylor</th>
<th>Legendre</th>
<th>Chebychev</th>
<th>Minimax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. Error</td>
<td>0.218</td>
<td>0.081</td>
<td>0.050</td>
<td>0.045</td>
</tr>
</tbody>
</table>
```

Taylor is the worst
Minimax and Chebychev are close

Taylor expansions only give local (i.e., around one value) approximations, should not be used for global...

Algorithm Mapping...

DSP Implementation & Elementary Functions...

- **Look-up Table**

Sample the function with enough resolution in a minimum length interval
Trade off between length and accuracy \(\rightarrow\) improvement by interpolation

Example: \(f(x) = \arccos(x)\), 1024 point table \(\rightarrow\) only for \(0 \leq f \leq \pi/2\)

\(\arccos(-x) = \pi - \arccos(x)\); Linear assembly code without interpolation

```assembly
shr x, 5, index
ldh *+p_arccos[index], arccos
```

; Linear assembly code with interpolation

```assembly
shr x, 5, index
ldh *+p_arccos[index], arccos1
add index, 1, index
ldh *+p_arccos[index], arccos2
shr index, 5, index
sub x, index, dx
sub arccos2, arccos1, df
mpy df, dx, a
```

Exercise-6: Lookup table for arctan + asm code
Best accuracy / cost?

Exercise-5: Remez and Cheby In Matlab, 3-term polys for sine and cosine, compare with Taylor

1) Start with a set of \(x_i\)s in \([a,b]\). Good general choice is:
\[ x_i = \frac{a + b}{2} + \frac{(b - a)}{2} \cos \left( \frac{i\pi}{n + 1} \right), \quad 0 \leq i \leq n + 1 \]

2) Solve the following linear system of equations for \(p_s\) and \(\varepsilon\)

\[
\begin{align*}
(p_0 + p_1 x_0 + p_2 x_0^2 + \cdots + p_n x_0^n - f(x_0)) &= +\varepsilon \\
(p_0 + p_1 x_1 + p_2 x_1^2 + \cdots + p_n x_1^n - f(x_1)) &= -\varepsilon \\
(p_0 + p_1 x_2 + p_2 x_2^2 + \cdots + p_n x_2^n - f(x_2)) &= +\varepsilon \\
&\vdots \\
(p_0 + p_1 x_{n+1} + p_2 x_{n+1}^2 + \cdots + p_n x_{n+1}^n - f(x_{n+1})) &= (-1)^{n+1}\varepsilon
\end{align*}
\]

3) Calculate new extreme points of \(p(x) - f\), replace with \(x_i\)s and repeat

- Minimax: Remez’s Algorithm
  - Remez: Iterative solution to minimax approximation
  - Start with a set of \(x_i\)s in \([a,b] \). Good general choice is:
  - \( x_i = \frac{a + b}{2} + \frac{(b - a)}{2} \cos \left( \frac{i\pi}{n + 1} \right), \quad 0 \leq i \leq n + 1 \)
  - Solve the following linear system of equations for \(p_s\) and \(\varepsilon\)
  - \[
  \begin{align*}
  (p_0 + p_1 x_0 + p_2 x_0^2 + \cdots + p_n x_0^n - f(x_0)) &= +\varepsilon \\
  (p_0 + p_1 x_1 + p_2 x_1^2 + \cdots + p_n x_1^n - f(x_1)) &= -\varepsilon \\
  (p_0 + p_1 x_2 + p_2 x_2^2 + \cdots + p_n x_2^n - f(x_2)) &= +\varepsilon \\
  &\vdots \\
  (p_0 + p_1 x_{n+1} + p_2 x_{n+1}^2 + \cdots + p_n x_{n+1}^n - f(x_{n+1})) &= (-1)^{n+1}\varepsilon
  \end{align*}
  \]
  - Calculate new extreme points of \(p(x) - f\), replace with \(x_i\)s and repeat
Algorithm Mapping...

**DSP Implementation & Elementary Functions**

- **Miscellaneous Techniques**
  - Rewriting the expressions, suitable forms

\[
\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \text{higher order}
\]

\[
\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} + \text{higher order}
\]

When signal generation is needed, sine and cosine cogeneration example:

\[
\sin nx = 2\cos x \cdot \sin(n-1)x - \sin(n-2)x
\]

\[
\cos nx = 2\cos x \cdot \cos(n-1)x - \cos(n-2)x
\]

Exercise -7: Why? Write code and compare

---

**Digital Filters**

- **Teaching ROM chapter 14, 15**
- **Some Practical Stuff**
  1) FIR Filter order prediction:

  - Harris' Formula: \( N \approx \frac{A}{2\pi (\omega_p - \omega_s)/2\pi} \)
  - Kaiser's Formula: \( N \approx \frac{A}{2\pi (\delta_p + \delta_s)/2\pi} \)

  Where \( A \) is attenuation in dB, \( \delta_p \) & \( \delta_s \) are peak ripples in pass and stop bands

Both show inversely proportional relation to transition band width

2) Circular Addressing Settings in C6000 (see enclosed example addressing.rar)

- Address Mode Register (AMR)
  - Only 8 registers can be used: A4, A5, A6, A7, B4, B5, B6, B7
  - And Two sizes only
  - Based on BK0, BK1

---

**FIR Filters, Some Practical Stuff**

- Circular Addressing Settings in C6000 (see example addressing.rar)

Use .align N to align data to size of the buffer in Assembly

Use #pragma DATA_ALIGN (symbol, constant) for global vars in C

Use .space N to reserve area in assembly for buffer

Follow the code in the Lab textbook see also opt workshop chapter 17
Algorithm Mapping...

**FIR Filters, Some Practical Stuff**

3) MATLAB filter design toolbox, Process flow diagram

- Select a Response
- Select a Specification
- Select an Algorithm
- Customize the Algorithm
- Design the Filter
- Design Analysis
- Device or Apply the Filter to Input Data

### Specifiers Object

- Implementation Object

### MATLAB Filter Design Toolbox

- Creating a filter design Object, using response, selecting a specification: Default. Set a new one

```matlab
[b, a] = fdesign.lowpass('FilterOrder', 50, 'CutoffFrequency', 2000, 'StopbandRipple', 100);
```

### Setting Design Parameters

- Change spec (another way)

```matlab
specs = [50, 2000, 100];
[b, a] = fdesign.lowpass(specs, 'FilterOrder', 50, 'CutoffFrequency', 2000, 'StopbandRipple', 100);
```

- Can specify Fs after other parameters

```matlab
Fs = 40;
[b, a] = fdesign.lowpass(specs, 'FilterOrder', 50, 'CutoffFrequency', 2000, 'StopbandRipple', 100, 'Fs', Fs);
```
3) MATLAB filter design toolbox

Setting Design Parameters...
Amplitude specs can be entered in dB, linear or squared values
But always converted and saved in dB

Normalized Frequency
After changing any frequency parameters
Or at the same time

Example
Hd = design(f, 'butter', 'filterstructure', 'df2sos')

Complete List:

butter
design
cheby1
cheby2
equiripple
equiripple_short
ellip
bessel
neutral
iircomb
multistage
Algorithm Mapping...

Digital Filters, Some Practical Stuff...

3) MATLAB filter design toolbox...

Design Filter...

Complete List with more explanation:

```matlab
% Design Methods for class design.lowpass; Fp,Fz,Np,Ap,et;

Butterworth
Chebyshev type 1
Chebyshev type II
Elliptic
Equiripple
Sinc
Kaiser window
Multistage equiripple
```

Some methods might have design time additional spec...

Can get them in structures and ...

```matlab
% Get the default design time options
do = designopts('ellip');
% Match the stopband exactly.
{do,MatchExactly} = 'stopband';

% Show all FIR designs
design(b, 'ellip', do);
set(gcf,'Color','white');
```

Cost Analysis Example:

```matlab
>> help dfilt/analysis
```

Can compare Designs:

```matlab
% Show all FIR designs
design(b,'ellip');
set(gcf,'Color','white');
```

Algorithm Mapping...

Digital Filters, Some Practical Stuff...

>> help dfilt/analysis  ... Design Analysis

Cost Analysis Example:
Algorithm Mapping...

Digital Filters, Some Practical Stuff ...

3) MATLAB filter design toolbox... Design Analysis...

Second-order sections (Filter Design Toolbox Required)
- Scale second-order sections.
- Create options object for use scaling.
- Vector of cumulative second-order section weights.
- Create second-order sections.
- Cascade filter design toolbox.
- Create options object for scale.
- Create options object for filter.
- Add a stage to a cascade or parallel filter.
- Set a stage in a cascade or parallel filter.

Realization and Test
>> y = filter(FilterObj, x)
>> help dfilt/filter

>> help dfilt/filter
FILTER Execute ("run") discrete-time filter.
T = FILTER(HLX) filters the data X using the discrete-time filter
object HLX to generate the filtered data T. The final conditions are
stored in HLX.States.

IF HLX.PersistentMemory is false (default), initial conditions
are set to zero before filtering.

Algorithm Mapping...

Digital Filters, Some Practical Stuff ...

4) Filter Theory and Implementation

LTI Digital Filters: FIR (all \(a_k=0\)) or IIR (at least one \(a_k\neq0\))

\[
y[n] = h[n] * x[n] = \sum_{k=0}^{N-1} b_k x[n-k] - \sum_{k=0}^{M-1} a_k y[n-k]
\]

\[
H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N-1} b_k z^{-k}}{1 + \sum_{k=1}^{M-1} a_k z^{-k}} : \text{Transfer Function} \quad H(e^{j\omega}) : \text{Frequency Response}
\]

A causal FIR filter whose impulse response is symmetrical is guaranteed to have a linear
phase response → Odd (Type I) or Even (Type II) Symmetry \(h[n]=h[n-1]\)

DF I, Twice as many delays as necessary

DF II and Transposed DF II

TDF II is the best, computationally
Update can be done "in sequence"
\(F\), all zeros on/above main diagonal

\[
y(n) = CS(n)
S(n) = FS(n) + F_S(n-1) + Bx(n)
\]
Algorithm Mapping...

Digital Filters, Some Practical Stuff...
4) Filter Theory and Implementation...

Cascade IIR Structure ➔ MATLAB: tf2sos(b,a), dfilt.df2sos(sos) functions

\[ H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \]

* The cascaded biquads can be reordered to reduce the round-off noise. N possible options for N biquads

Algorithm Mapping...

Digital Filters, Some Practical Stuff...
4) Filter Theory and Implementation...

Parallel IIR Structure ➔ Partial Fraction Expansion: MATLAB residuez

\[ H(z) = C + H_1(z) + H_2(z) + \cdots + H_N(z) \]

If using DFI, only 2 registers at input will do

Algorithm Mapping...

Digital Filters, Some Practical Stuff...
4) Filter Theory and Implementation...

Lattice FIR Structure: Less sensitive to small changes in coefficients

\[ H(z) = \sum_{k=0}^{N} h_k z^{-k} = h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots + h(N-1)z^{-(N-1)} \]

\[ h(0)z^{-N} + h(1)z^{-1} + h(2)z^{-2} + \cdots + h(N-1)z^{-N} \]

Image Polynomials

We need a way to calculate \( k \) coefficients to use the structure:

Backward recursive relation exists:

\[ E_{r}(z) - \frac{1}{k_{r}} E_{r}(z) \]

Applications in Adaptive filtering and LPC

\[ y_i(n) = x(n) + k_i x(n-1) \]

\[ e_i(n) = k_i \cdot \epsilon_i(n) \]

\[ y_i(n) = y_i(n-1) + k_i \epsilon_i(n-1) \]

\[ e_i(n) = k_i \epsilon_i(n) + \epsilon_i(n-1) \]

\[ y_i(n) = x(n) + k_i \epsilon_i(n) \]

\[ e_i(n) = k_i \epsilon_i(n) \]

\[ y_i(n) = x(n) + k_i \epsilon_i(n) \]

\[ e_i(n) = k_i \epsilon_i(n) \]

\[ y_i(n) = x(n) + k_i \epsilon_i(n) \]

\[ e_i(n) = k_i \epsilon_i(n) \]
Algorithm Mapping...

Digital Filters, Some Practical Stuff...

4) Filter Theory and Implementation

Lattice Structure for all pole case

Building block:

\[ y_{i+1}(n) = y_i(n) - k_i e_{i+1}(n-1) \]

\[ e_i(n) = k_i y_{i-1}(n) + e_{i-1}(n-1) \]

Example

Stability \(|k_i| < 1\)

Algorithm Mapping...

Digital Filters, Some Practical Stuff...

4) Filter Theory and Implementation

FIR Filter Design: Using Fourier Series

Magnitude response Fourier series representation

\[ H_d(z) = \sum_{n=-\infty}^{\infty} C_n z^{-n} \]

\[ v = fF_N \quad F_N = F/2 \]

\[ H_d\left(\frac{v}{F_N}\right) = \sum_{n=-\infty}^{\infty} C_n e^{j2\pi n v / F} \]

Periodic, period = 2

\[ C_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d\left(\frac{v}{F_N}\right) e^{-j2\pi n v / F} dv \]

\[ H_d\left(\frac{v}{F_N}\right) = \sum_{n=\infty}^{\infty} C_n e^{j2\pi n v / F} \quad C_n = \int \frac{H_d(v)}{2\pi} e^{-j2\pi n v / F} dv \]

\[ H(z) = \sum_{n=\infty}^{\infty} C_n z^{-n} \quad H(z) = z^{-Q} H(z) = \sum_{n=\infty}^{\infty} C_n z^{-n-Q} \]

For non-rectangular windowing: \( C'_n = C_n w(n) \)

Algorithm Mapping...

Digital Filters, Some Practical Stuff...

4) Filter Theory and Implementation

FIR Filter Design: Windowing Method

1) Having a time domain impulse response

Sample → Truncate and window → time shift for causality

2) Having a frequency domain magnitude response

Sample → Use IFFT to have \( h[n] \) → Window → Time shift for causality

\[ H(z) = \sum_{n=\infty}^{\infty} h[n] z^{-n} \quad H(e^{j\omega_T}) = \sum_{n=\infty}^{\infty} h[n] e^{-j\omega_T n} \]

\[ H(e^{j\omega_T}) = e^{-j\omega_T} \sum_{n=-\infty}^{\infty} h[k] e^{-j\omega_T n} = e^{-j\omega_T} H_1(e^{j\omega_T}) \]

\[ H_1(e^{j\omega_T}) = \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_T k} = A(f) \]

\[ h[n] = T \int_{-\infty}^{\infty} A(f) e^{j\omega_T n} df \]

not causal, but easy to design and needs a phase shift to become \( h[n] \)
Algorithm Mapping...

Digital Filters, Some Practical Stuff...
4) Filter Theory and Implementation...
FIR Filter Design: Windowing Method...
Choosing the right window based on Filter specification...

<table>
<thead>
<tr>
<th>Window Type</th>
<th>Normalized Transition Width ((\Delta f / f_s))</th>
<th>Passband Ripple (dB)</th>
<th>Stopband Attenuation (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>(\frac{2}{N})</td>
<td>0.7416</td>
<td>21</td>
</tr>
<tr>
<td>Hanning</td>
<td>(\frac{1.5}{N})</td>
<td>0.0546</td>
<td>44</td>
</tr>
<tr>
<td>Hamming</td>
<td>(\frac{1.5}{N})</td>
<td>0.0194</td>
<td>53</td>
</tr>
<tr>
<td>Blackman</td>
<td>(\frac{1.5}{N})</td>
<td>0.0017</td>
<td>74</td>
</tr>
<tr>
<td>Kaiser</td>
<td>(\frac{1.5}{N} + \beta = 4.5)</td>
<td>0.0274</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>(\frac{1.5}{N} + \beta = 8.96)</td>
<td>0.000275</td>
<td>90</td>
</tr>
</tbody>
</table>

Example: \(f_s = 8\text{kHz}, 1.2\text{kHz to 1.4 kHz Transition, Using Hamming Window}

\[
N = \frac{3.3}{\Delta f} = \frac{3.3}{(1.4-1.2)\text{kHz}} = 8\text{kHz} = 132
\]

Algorithm Mapping...

Digital Filters, Some Practical Stuff...
4) Filter Theory and Implementation...
FIR Filter Design: Windowing Method...

Choosing the right window based on Filter specification...

\[
A(f_s) = \frac{1}{\sqrt{1 + (fs / f_s)^2}}
\]

Frequency sampling
Use ifft to derive \(h[n]\)
High 'n' close to brick-wall filter

Algorithm Mapping...

Digital Filters, Some Practical Stuff...
4) Filter Theory and Implementation...
FIR Filter Design: Windowing Method...

Choosing the right window based on Filter specification...

\[
h(nT_s) = \begin{cases} 
1; & n = 0 \\
0; & n \neq 0
\end{cases}
\]

Nyquist Criterion

\[
\frac{1}{T_s} \sum_{k=-\infty}^{\infty} H \left( f - k \frac{f_s}{T_s} \right) = 1
\]

C) Example: Raised Cosine Pulse shape with pulse duration \(T\)
Designing from impulse response → Application to pulse shaping

\[
d(t) = \sum_k d_k \delta(t - kT) \xrightarrow{\text{p}(t)} x(t) = \sum_k p(t - kT)
\]

\[
p(t) = \frac{\sin(\pi t / T)}{\pi t} \cos(\pi \beta t / T) \quad 1 - 4\beta^2 t^2 / T^2
\]

For causality, introduce a delay, then sampling

\[
t - t_d = nT_s - mT_s,
\]

\[
T_s = T / k \Rightarrow t = \frac{n}{k} - m
\]

Delay \(nT_s\), Truncate \(2mT_s\) => \(2m\) samples
Zero at multiples of \(T=1\), zero ISI
SQRRC
4) Filter Theory and Implementation

Digital Filter Design based on Analog Filters

**Butterworth Filters**

$$H(j\omega) = \frac{1}{\sqrt{1 + (\omega/\omega_c)^{2n}}}$$

$$10 \log_{10} |H(j\Omega)|^2 = 10 \log_{10} \frac{1}{1 + \frac{\omega_c^2}{\omega^2}} = -A_p$$

Normalized to have pass-band at $\Omega=1$

$$10 \log_{10} H(j\Omega)|^2 = 10 \log_{10} \frac{1}{1 + \frac{\omega_c^2}{\omega^2}} = -A_s$$

-3dB $\Omega_{3dB} = 1$

$$p_1 = e^{-\pi \omega_c} \exp(j \frac{\pi}{2} (2n-1))$$

Butterworth polynomials Transfer function needs an scaling factor.

**Chebyshev Filters I, II**

$I$- $H(j\Omega)|^2 = \frac{1 + C_c (\Omega)}{1 + \epsilon C_c (\Omega)}$, $\Omega = \omega/\omega_c$

$$10 \log_{10} |H(j\Omega)|^2 = 10 \log_{10} \frac{1}{1 + \epsilon C_c (\Omega)} = -A_p$$

**Butterworth Design**

$$\epsilon = \frac{1}{A_p}$$

**Chebyshev Design**

$$\epsilon = \frac{A_p}{10}$$

**Elliptic Design**

$$\epsilon = \frac{A_p}{10}$$

**Design Parameters**

- $\omega_c$: Critical frequency
- $\epsilon$: Passband ripple
- $A_p$: Passband attenuation
- $A_s$: Stopband attenuation

**Magnitude, Phase, Group Delay**

- Specified magnitude, some properties on phase or group delay
- Example: Rational Transfer Functions

$$H(s) = \frac{s+1}{s^2 + 2s + 2} \quad \text{and} \quad H(s)H(-s) \frac{1}{s^2 + \omega^2}$$

**Transform to High Pass, Band Pass, Band Stop**

- LP to HP: $s \rightarrow \omega_c / s$, $(\omega_c$ passband freq)
- LP to BP: $s \rightarrow (s^2 + \omega_c^2)/(Bs)$, $B = \omega_c - \omega, \omega_c = \sqrt{\omega_1 \omega_2}$

Digital Filter Design based on Analog Filters

Magnitude, Phase, Group Delay

- Specified magnitude, some properties on phase or group delay

**Example: Rational Transfer Functions**

$$H(s) = \frac{s+1}{s^2 + 2s + 2} \quad \text{and} \quad H(s)H(-s) \frac{1}{s^2 + \omega^2}$$

**Poles on the left (stability)**

- Zeros on the left (Minimum Phase)

**Ideal Low-pass Filters Approximations**

$$|H(j\omega)|^2 = \frac{C_0 + C_2 \omega^2 + ... + C_{2n} \omega^{2n}}{D_0 + D_2 \omega^2 + ... + D_{2n} \omega^{2n}}$$

**Butterworth, Chebyshev I, II and Elliptic**

- Maximally Flat versus equiripple in different regions

**Transform to High Pass, Band Pass, Band Stop**

- LP to HP: $s \rightarrow \omega_c / s$, $(\omega_c$ passband freq)
- LP to BP: $s \rightarrow (s^2 + \omega_c^2)/(Bs)$, $B = \omega_c - \omega, \omega_c = \sqrt{\omega_1 \omega_2}$

**Most Popular: Parks McClellan Method**

- Optimum Equi-ripple FIR filter based on Minimax polynomials
- Important to make pulse shapes and special frequency responses

**remez function in old Matlab (obsolete but still works)**

**firpm function in new Matlab**

$$\text{B=FIRPM(N,F,A)}$$

- $N$: Required order

**Order, a vector for freq, a vector for amplitude**

- Linear phase results
- Distributes $n$(=order) extremes in passband and stop band

**Order , a vector for freq, a vector for amplitude**

- Linear phase results
- Distributes $n$(=order) extremes in passband and stop band

**FIR Filter Implementation:**

- Linear phase Direct form

$$H(z) = \sum_{k=0}^{N-1} b_k z^{-k}$$

**a) Even**

$$H(z) = \sum_{k=0}^{N/2-1} b_k (z^{-k} + z^{-(N-k)})$$

**b) Odd**

$$H(z) = \sum_{k=0}^{(N-1)/2} b_k (z^{-k} + z^{-(N-k)}) + b_{(N-1)/2} z^{-(N-1)/2}$$

**Algorithm Mapping...**

**Digital Filters , Some Practical Stuff...**

**FIR Filter Design: Computer Aided Design for FIR filters**

- Optimum Equi-ripple FIR filter based on Minimax polynomials
- Important to make pulse shapes and special frequency responses

**remez function in old Matlab (obsolete but still works)**

**firpm function in new Matlab**

$$\text{B=FIRPM(N,F,A)}$$

- $N$: Required order
Algorithm Mapping...

Digital Filters, Some Practical Stuff...

4) Filter Theory and Implementation...

Digital Filter Design based on Analog Filters...

Impulse Invariant, Step Invariance, Bilinear Transform

Impulse Invariant

\[ H(z) = Z\{L^{-1} H_s(s)\}_{s \to nT} \rightarrow z = e^{nT} \]

\[ H_s(s) = \sum_{k} \frac{A_k}{s - s_k} \rightarrow H(z) = \sum_{k} \frac{A_k}{z - z_k} \]

\( h(t) \) must be band limited... no HP and some BP usage

Suffers from aliasing error , zero frequency gain, ...

Good to match low frequency if sampling rate chosen carefully

Step Invariance

\[ H(z) = (1 - z^{-1}) Z\{L^{-1} \frac{H_s(s)}{s} \}_{s \to nT} \rightarrow z = e^{nT} \]

\[ H_s(s) = \sum_{k} \frac{A_k}{s - s_k} \rightarrow H(z) = \sum_{k} \frac{A_k (1 - e^{-nT}) z^{-1}}{s_k (1 - e^{-nT} z^{-1})} \]

Algorithm Mapping...

Digital Filters, Some Practical Stuff...

4) Filter Theory and Implementation...

Digital Filter Design based on Analog Filters...

Bilinear Transform

\[ H(z) = H_s(s) \rightarrow z = C \frac{1 - z^{-1}}{1 + z^{-1}} \]

\[ \omega_n = C \tan(\frac{\omega_f T}{2}) \]

\( C \) : Prewarping

Sampling rate high enough and \( f_c \) chosen to be almost \( f_s \)

Reasonable match over a wide frequency range

Does not keep the constant group delay

Matlab Functions:

buttord, chebord, cheb2ord, ellipord

butter, cheby1, cheby2, ellip

yulewalk

Algorithm Mapping...

Digital Filters, Some Practical Stuff...

4) Filter Theory and Implementation...

Digital Filter Design based on Analog Filters...

Yule Walker approximation: \( D \) : desired, \( W \) : weighting, \( H \) : Rational to be found

\[
J(\omega) = \int \left| W(e^{j\omega}) \cdot H(e^{j\omega}) - D(e^{j\omega}) \right|^p d\omega
\]

Least \( p \)th approximation. \( p=2 \) least square approximation

[\text{num,den}] = yulewalk(N,F,D)

Arbitrary Transfer function

Avoid sharp transitions when using Matlab yulewalk

Increasing \( N \) improves the approximation

Deczky's Method

\[
J(\omega) = \alpha E_A + (1 - \alpha) E_G
\]

Minimizes the errors in amplitude response and group delay

Many other criteria, different methods
Exercise 8: Use fdesign Toolbox and filtool in MATLAB
Design a Low pass filter with the following Spec:
Fs = 8KHz, Fpass = 3400 Fstop = 3600 Hz, Astop = 80dB
1) As an FIR linear phase  2) As an IIR
A) Find the best method by trying all the options, conduct some simulations
B) Write optimized Code for the best cases in C67X floating point
C) Convert the best cases into fixed point with optimal wordlength/accuracy
D) Write optimized fixed point code
E) Find the cycle count and HW( HDL) cost for each case
F) Find the SoS design of the best cases, repeat (E)