"I have an awful foreboding that eventually I'll succumb to you but I feel I owe it to my conscience to put up an awful fight."

EE 311 **March 22 - 25, 2018**
Lecture 16-17

6.4 Classic IIR filters

• Butterworth, Chebyshev, and elliptic filter types tend to be what we think about when we consider a classic IIR filter design.

• The Butterworth filter has a flat pass band, a monotonic stop band and the widest transition band of the three types.
• The elliptic filter has ripple in both the pass and stop bands but has the narrowest transition band of the three.
• The Chebyshev filter falls between the two with some ripple in the pass band, a monotonic stop band, and a transition bandwidth that falls between that of the Butterworth and Elliptic.

The Butterworth Filter

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

• We note that when the frequency is 0, the magnitude is 1 and, at some frequency f_c, the magnitude is at the half-power point = $\sqrt{2}/2$.
• The frequency f_c is called the cutoff frequency.
• The stop band is monotonic which, in this case, means its derivative is always negative. The pass band is said to be maximally flat and never goes above the value of 1.

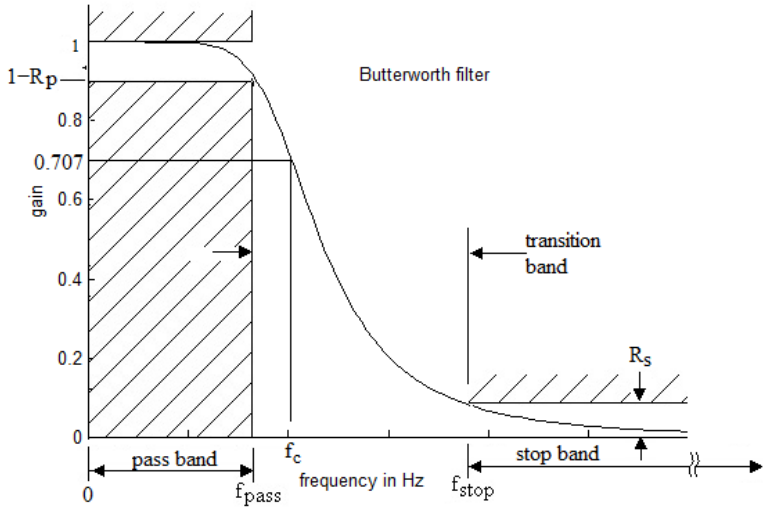


Figure 6.11
A low pass Butterworth filter. The pass band is flat, the stop band is monotonic, and a cutoff frequency is defined as the frequency at which the gain is $\sqrt{2}/2$.

$$H(s)H(-s) = \frac{1}{1+(s/\,j\omega_c)^{2N}}$$

In (6.11) the poles can be found as the roots of the equation given by:

$1+(s/\ j\omega_c)^{2N}=0$

or

$s^{2N}=(-1)(j\omega_c)^{2N}$

and

$s=(-1)^{1/2N}\ j\omega_c$

But $-1=e^{j\pi+2k\pi}$ and $j=e^{j\pi/2}$ so that

$s=\omega_c e^{j\pi(1+N+2k)/(2N)}$ where k = 0,1,2,3...

Since the e^j term has a magnitude of 1 we see that the poles of the Butterworth filter all lie on a circle of radius ω_c. The angle of the pole terms is $\pi(1+N+2k)/2N$, k = 0,1,...(2N-1).

• In deriving the equation for the pole locations the amplitude squared response was used to avoid dealing with the square root function.
• This means that we will have twice as many poles as required for the non-squared function.
• Since the squared amplitude response was found by multiplying $H(s)$ times $H(-s)$, half of the roots belong to $H(s)$ and the other half are those of $H(-s)$ (in the right half plane).

Example 6.7
Design a fourth order Butterworth filter which has a cutoff frequency of 55 Hz.
Solution:
For N = 4 and k going from 0 to 7 we get the angle of the poles at 5π/8, 7π/8, 9π/8,..., 19π/8. The pole locations are shown in Figure 6.12.

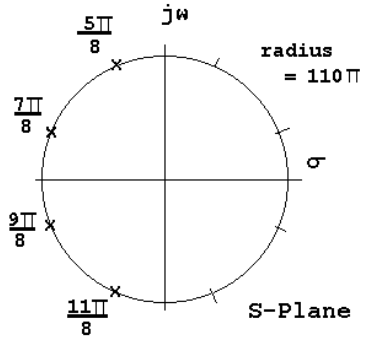


Figure 6.12
Pole locations in the *s*-plane for a fourth order Butterworth filter with a cutoff frequency.

The transfer function for the filter is found by multiplying the pole terms

$$H(s)=\frac{1}{s^4+903.03s^3+4.0773x10^5s^2+1.0784x10^8s+1.4262x10^{10}}$$

• In general an analog Butterworth filter has the form:

$$H(s)=\frac{1}{s^N+K_1s^{N-1}+\cdots+K_N} \qquad (6.12)$$

When we apply the BLT we substitute $S \leftarrow \frac{2}{T}\frac{z-1}{z+1}$. Making this substitution in (6.12) and simplifying gives:

$$H(z) = \frac{(T(z+1))^N}{(2(z-1))^N + \cdots + K_N((T(z+1))^N} \tag{6.13}$$

• Since the BLT maps a stable filter in s to a stable filter in z the poles in (6.13) will fall inside the unit circle.
• The low pass Butterworth filter in *z* now has *N* zeros at the *z* = -1 point.
• a low pass Butterworth filter will have *N* zeros at the *z* = -1 point and will have *N* poles distributed inside the unit circle.
• Since all of the zeros are at *z* = -1, the numerator polynomial is symmetric providing for some efficiency in the calculation of the difference equation.

Example 6.8
Use MATLAB® to design a low pass Butterworth digital filter that has the following specifications.
Sample frequency = 11,025 Hz
Cutoff frequency = 2,500Hz
Order = 6

Solution
The following MATLAB® code produces this filter.

```
fs = 11025;   %sample frequency
fc = 2500;    %cuoff frequency
N = 6;        %order
[num den] = butter(N, fc/(fs/2)); %cutoff frequency is normalized
[H f] = freqz(num, den, 1024, fs);
plot(f, abs(H));
```

Figure 6.13
A sixth order low pass Butterworth filter with a cutoff frequency of 2,500Hz and a sample frequency of 11,025Hz

• To calculate the order of a Butterworth filter from frequency specifications we define the following symbols:

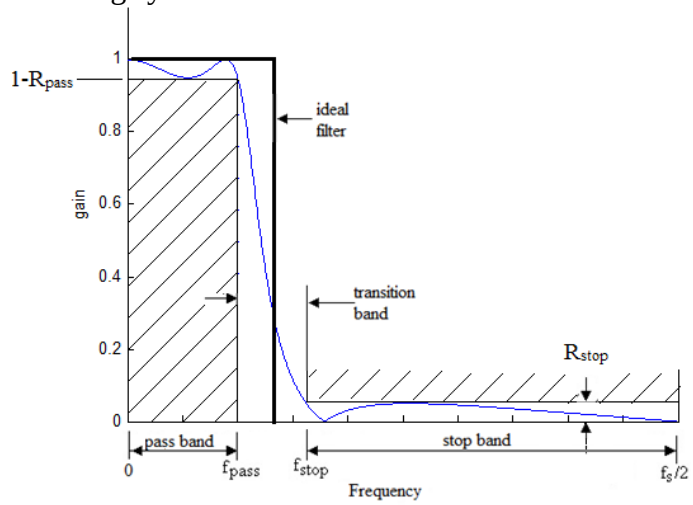


Figure 6.14
IIR filter definitions.

f_{pass} = Pass band edge in Hertz
f_{stop} = Stop band edge in Hertz
R_{pass} = Pass band ripple
R_{stop} = Stop band ripple

R_{pass} and R_{stop} are fractions of the gain and may have no units or they may be converted to decibels.

Using these definitions, we know that the amplitude response has a value of R_{stop} at a frequency of f_{stop}. We also know that the amplitude response has a value of 1-R_{pass} at a frequency of f_{pass}. We can therefore write two equations with two unknowns using equation 6.11.

$$R_{stop}^2 = \frac{1}{1+(f_{stop}/f_c)^{2N}} \tag{6.14}$$

$$(1-R_{pass})^2 = \frac{1}{1+(f_{pass}/f_c)^{2N}} \tag{6.15}$$

These two equations can be solved for N and f_c with about a page of algebra. The solution is

$$N = \frac{\ln(K_s/K_p)}{2\ln(f_{stop}/f_{pass})} \tag{6.16}$$

and

$$\ln(f_c) = \frac{\ln(K_s)\ln(f_{pass}) - \ln(K_p)\ln(f_{stop})}{\ln(K_s/K_p)} \tag{6.17}$$

where

$$K_s = (1/R_{stop}^2) - 1$$

$$K_p = \frac{1}{(1-R_{pass})^2} - 1$$

Note that if this analog filter is to be converted to a digital filter by way of the BLT, the frequencies f_{stop} and f_{pass} should be *prewarped* frequencies.

MATLAB® automates equations 6.16 and 6.17 with the `buttord` function.

Example 6.9
Use MATLAB® to design a low pass Butterworth digital filter that has the following specifications.
Sample frequency = 11,025 Hz
Pass band edge = 2,000Hz
Pass band ripple = 0.01
Stop band edge = 2,700Hz
Stop band ripple = 0.05

Solution
The following MATLAB® code produces this filter.

```
fs = 11025;         %Sample frequency
fpass = 2000;       %Pass band edge
fstop = 2700;       %Stop band edge
%Ripple must be specified in decibels
Rpass = 0.01;RpassDB = -20*log10(1-Rpass);
Rstop = 0.05;RstopDB = -20*log10(Rstop);
[N fn] = buttord(fpass/(fs/2), fstop/(fs/2), RpassDB, RstopDB);
[num den] = butter(N, fn);
[H f] = freqz(num, den, 1024, fs);
plot(f, abs(H));
```

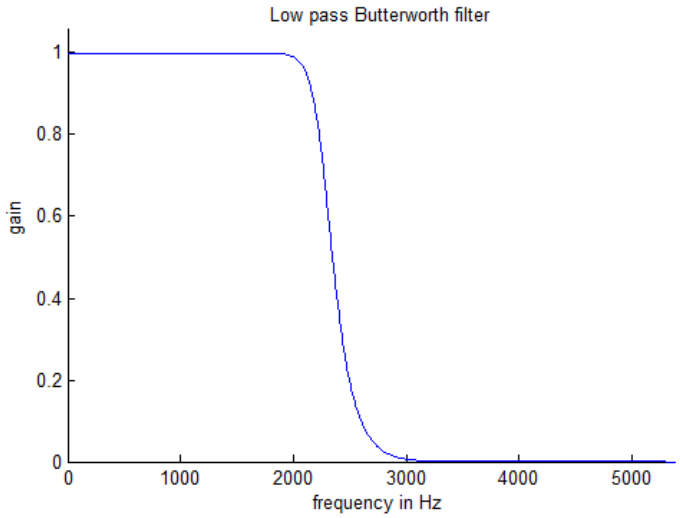


Figure 6.15
A low pass Butterworth filter meeting frequency band and ripple specifications.

EE 311 **March 17, 2017**
Analog Butterworth Filter
SOLUTION

1. Use Matlab to design an analog Butterworth filter which meets the following specifications.

Pass band	0 to 1500Hz
Pass band ripple	0.01
Stop band	2500Hz to ∞
Stop band ripple	0.01

Produce a magnitude plot, a phase plot, and a pole/zero plot for your filter.

```
%ButterAnalog.m
%specifications:
%Pass band  0 to 1500Hz
%Pass band ripple   0.01
%Stop band  2500Hz to ?
%Stop band ripple   0.01
%************* ANALOG FILTER *************
fpass = 1500;Wp = 2*pi*fpass;
fstop = 2500;Ws = 2*pi*fstop;
Rp = .01;RpDB = -20*log10(1-Rp);
Rs = .01;RsDB = -20*log10(Rs);
[N Wn] = buttord(Wp, Ws, RpDB, RsDB, 's');
[num den] = butter(N,Wn,'s');
[H w] = freqs(num, den, 1024);
figure(1);clf;
subplot(2, 1, 1);
plot(w/(2*pi), abs(H));
title('Butterworth Analog Filter');
xlabel('frequency in Hz');
ylabel('Gain');
axis([0 fpass .9 1.1]);
subplot(2, 1, 2);
plot(w/(2*pi), angle(H)*180/(2*pi));
xlabel('frequency in Hz');
ylabel('Phase in degrees');
figure(2);clf;
pzmap(num, den);
title('Butterworth Analog PZ Plot');
```

EE 311 **March 17, 2017**

Digital Butterworth Design

SOLUTION

1. Use Matlab to design a digital Butterworth filter which meets the following specifications:

Sample frequency:	11025
Pass band edge:	fs/10 Hz at 0.9
Stop band edge:	fpass + fs/6 Hz at 0.1

Find the numerator and denominator coefficients for the filter and write the implementation equations in C-code suitable for the ARM processor.

```
%ButterDesign.m
%Specifications: Digital filter
%sample frequency: 11025Hz
%Pass band  0 to fs/10 Hz
%Pass band ripple   0.1
%Stop band  fpass + fs/6 Hz to infinity
%Stop band ripple   0.1
%*** SPECIFICATIONS ***
fs = 11025;
fpass = fs/10;
fstop = fpass + fs/6;
Rp = .1;RpDB = -20*log10(1-Rp);
Rs = .1;RsDB = -20*log10(Rs);
%************* BUTTERWORTH Digital FILTER *************
[NB fn] = buttord(fpass/(fs/2), fstop/(fs/2), RpDB, RsDB);
[num den] = butter(NB, fn);
[H f] = freqz(num, den, 1024, fs);
figure(1);clf;
plot(f, abs(H), 'blue');
xlabel('frequency in Hz');
ylabel('Gain');
%axis([0 fpass .99 1]);   %Pass band blow up
%axis([fstop fs/2 0 .01]); %Stop band blow up
fprintf(1, 'Butterworth order = %2i\n', NB);
fprintf(1, 'Numerator \n');
for i=1:NB+1
    fprintf(1, ' %2.12f \n', num(i));
end
fprintf(1, 'Denominator \n');
for i=1:NB+1
    fprintf(1, ' %2.12f \n', den(i));
end
```

Sample Output

```
>> ButterDesign
Butterworth order =  3
Numerator
 0.050923192467
 0.152769577400
 0.152769577400
 0.050923192467
Denominator
 1.000000000000
 -1.141566654557
 0.683160586547
 -0.134208392256
```

```
//Butter3.c
/*This program implements a filter in floating point
   This is for ARM Cortex M0 processor board.

          0.0509231924Z^3 + 0.1527695774Z^2 + 0.1527695774Z + 0.0509231924
H(Z) =   ------------------------------------------------------------------
           Z^3 - 1.141566654557Z^2 + 0.683160586547Z^2 - 0.134208392256

This filter was designed in MatLab as a 3rd order Butterworth filter with
   fpass = fs/10;
   fstop = fp + fs/6
   Rp = 0.1
   Rs = 0.1
The timer is polled to set the sample frequency to 11025Hz
*/

#define PDRUNCFG (*((volatile unsigned long *) 0x40048238))   //Power-down configuration register
#define SYSAHBCLKCTRL (*((volatile unsigned long *) 0x40048080))   //System AHB clock control
#define IOCON_R_PIO0_11 (*((volatile unsigned long *) 0x40044074))   //Pin control register
#define AD0CR (*((volatile unsigned long *) 0x4001C000))   //A/D Control
#define AD0DR0 (*((volatile unsigned long *) 0x4001C010))   //A/D Ch 0 Data
#define IOCON_PIO1_9 (*((volatile unsigned long *) 0x40044038))   //Pin control register
//Timer 0
#define TMR16B0IR (*((volatile unsigned long *) 0x4000C000))   //Timer 0 Interrupt register
#define TMR16B0TCR (*((volatile unsigned long *) 0x4000C004))   //Timer 0 Timer control register
#define TMR16B0TC (*((volatile unsigned long *) 0x4000C008))   //Timer 0 Timer counter
#define TMR16B0PR (*((volatile unsigned long *) 0x4000C00C))   //Timer 0 Prescale register
#define TMR16B0PC (*((volatile unsigned long *) 0x4000C010))   //Timer 0 Prescale register
#define TMR16B0MCR (*((volatile unsigned long *) 0x4000C014))   //Timer 0 Match Control register
#define TMR16B0MR0 (*((volatile unsigned long *) 0x4000C018))   //Timer 0 Match register 0
#define TMR16B0MR1 (*((volatile unsigned long *) 0x4000C01C))   //Timer 0 Match register 1
#define TMR16B0MR2 (*((volatile unsigned long *) 0x4000C020))   //Timer 0 Match register 2
#define TMR16B0MR3 (*((volatile unsigned long *) 0x4000C024))   //Timer 0 Match register 3
#define TMR16B0CCR (*((volatile unsigned long *) 0x4000C028))   //Timer 0 Capture Control register
#define TMR16B0CR0 (*((volatile unsigned long *) 0x4000C02C))   //Timer 0 Capture register 0
#define TMR16B0CR1 (*((volatile unsigned long *) 0x4000C030))   //Timer 0 Capture register 0
#define TMR16B0EMR (*((volatile unsigned long *) 0x4000C03C))   //Timer 0 External match register
#define TMR16B0CTCR (*((volatile unsigned long *) 0x4000C070))   //Timer 0 Count Control register
#define TMR16B0PWMC (*((volatile unsigned long *) 0x4000C074))   //Timer 0 PWM Control register

//Timer 1
#define TMR16B1IR (*((volatile unsigned long *) 0x40010000))   //Timer 1 Interrupt register
#define TMR16B1TCR (*((volatile unsigned long *) 0x40010004))   //Timer 1 Timer control register
#define TMR16B1PR (*((volatile unsigned long *) 0x4001000C))   //Timer 1 Prescale register
#define TMR16B1MR3 (*((volatile unsigned long *) 0x40010024))   //Timer 1 Match register 3
#define TMR16B1PWMC (*((volatile unsigned long *) 0x40010074))   //Timer 1 PWM Control register
#define TMR16B1MCR (*((volatile unsigned long *) 0x40010014))   //Timer 1 Match Control register
#define TMR16B1MR0 (*((volatile unsigned long *) 0x40010018))   //Timer 1 Match register 0
#define NVIC_ISER (*((volatile unsigned long *) 0xE000E100))  //Int Set Enable Reg
//Numerator
const float b0 = 0.050923192467;
const float b1 =  0.152769577400;
const float b2 =  0.152769577400;
const float b3 = 0.050923192467;
//Denominator
const float a1 = -1.141566654557;
```

```
const float a2 = 0.683160586547;
const float a3 = -0.134208392256;
 //---------------------------------------------------------------------------
int yInt;       //Output variable used by main program and interrupt
int main()
  {unsigned int xInt;
   float x, w, y;
   float x1, x2, x3;
   float y1, y2, y3;

    PDRUNCFG &= ~(1 << 4);      //Power up A/D
    SYSAHBCLKCTRL |= (1 << 13); //ADC Clock enable
    SYSAHBCLKCTRL |= (1 << 8);  //16-bit timer 1 Clock enable
    SYSAHBCLKCTRL |= (1 << 7);  //16-bit timer 0   Clock enable
    IOCON_R_PIO0_11 &= 0xFFFFFF78;//Zero out select bits and
    IOCON_R_PIO0_11 |= (1 << 1);  //  select A/D function P0.11
    //AD Control Reg. B7-B0 = channel 0
    // B15-B8 = ClkDiv.  48 MHz/(ClkDiv + 1) = 4 MHz = AD clock
    AD0CR = 0x0B01;
    //************** Set up PWM using Timer 1 ***************
    NVIC_ISER |= (1 << 17);  //Enable 16-bit timer 1 interrupt
    IOCON_PIO1_9 |= 1;       //Selects match function so Pin1_9 is PWM
    //Timer 1 setup for PWM
    TMR16B1PR = 1;           //Prescale register.  Divide P Clock by 2 = 24 MHz
    //MR3 sets PWM cycle length.  24 MHz/ 1024 = 23.437 KHz with 10 bit res
    TMR16B1MR3 = 1024;       //Match register 3.  TR and Pin1_9 Reset
    TMR16B1MCR |= (1 << 10); //Causes TimerCounter to be reset if match on MR3
    TMR16B1MCR |= (1 << 9);  //Causes interrupt on match to MR3
    TMR16B1PWMC |= 1;        //Enable PWM on Channel 1
    TMR16B1TCR |= 1;         //Enable TimerCounter to run
    //************** Timer 0 setup for sample time ******************
    TMR16B0PR = 1;           //Prescale register.  Divide P Clock by 2 = 24 MHz
    //Sample period = 24 MHz/2176 = 11029Hz.
    TMR16B0MR0 = 2176;       //Match register 0.
    TMR16B0MCR |= 1;         //Causes interrupt bit set on match to MR0
    TMR16B0MCR |= (1 << 1);  //Causes TimerCounter to be reset if match on MR0
    TMR16B0TCR |= 1;         //Enable TimerCounter to run

    while (1)  //Main program loop
     {//Convert A/D input to float from 0 <= x < 1.0
      AD0CR |= (1 << 24);   //Start conversion
      while(AD0DR0 < 0x7FFFFFFF);    //Wait for done bit
      xInt = AD0DR0;
      xInt = ((xInt >> 6) & 0x000003FF);//Shift into position for 10-bit A/D
      x = (float)xInt;
      x = x/1024.0;
      w = b0*x + b1*x1 + b2*x2 + b3*x3;
      y = w - a1*y1 - a2*y2 - a3*y3;
      yInt = (int)(512*y); //Data to PWM

      //Shift values for each section
```

```
      x3 = x2;
      x2 = x1;
      x1 = x;
      y3 = y2;
      y2 = y1;
      y1 = y;
      //Wait for interrupt bit on TMR 0 = sample period
      while((TMR16B0IR & 1) == 0);
      TMR16B0IR |= 1;     //Reset interrupt bit
    }
  }
//PWM is set up to generate an interrupt which is used to load
//  the output at end of PWM cycle.  This avoids the glitch
//  that would otherwise be caused by loading the PWM
//  register in mid-cycle
void TIMER16_1_IRQHandler(void)
  {TMR16B1IR = 8;       //clear interrupt due to MR3
   TMR16B1MR0 = yInt;   //Load the value of y from the main program
  }
```

To do a simulation

Steps

1. Create a new project in µVision using NXP LPC1114 102 processor.
2. Enter the code above as a c-file
3. To simulate a sine wave on the A to D input

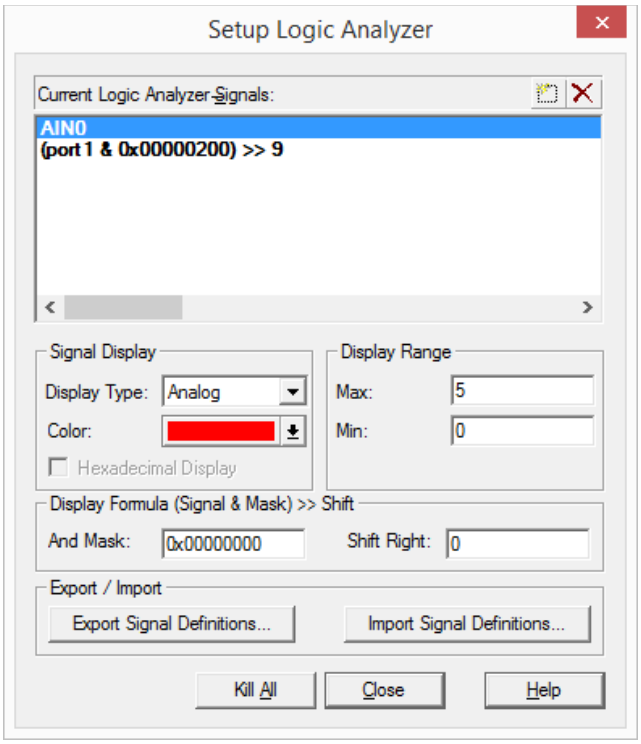
   File → New

   Enter the following simulation code

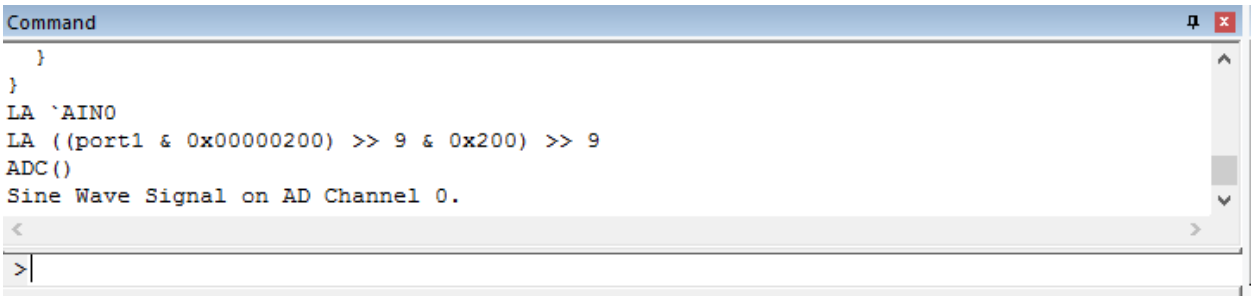
```
//
// Generate Sine Wave Signal on AD Channel 0
//
signal void ADC(void)
 {float volts;        // peak-to-peak voltage
  float frequency;    // output frequency in Hz
  float offset;       // voltage offset
  float duration;     // duration in Seconds
  float val;
  long  i, end;
  volts     = 1.4;
  offset    = 1.6;
  frequency = 1000;  //Change this to whatever is needed
  duration  = 0.1;

  printf ("Sine Wave Signal on AD Channel 0.\n");

  end = (duration * 100000);
  for (i = 0 ; i < end; i++)
   {val = __sin (frequency *
          (((float) STATES)/ CLOCK)*2*3.1415926);
    AD00 = (val * volts) + offset;
    swatch (0.00001);   // in 10 uSec steps
   }
}
```

   File → SaveAs Save this file as Debugger.ini in the project file.

4. Build project
5. *After* successful build run debugger Debug → Start/Stop Debug Session
6. In Debugger: View → Logic analyzer Window
7. In Logic analyzer click on SetUp. Push Insert key and enter `port1.9` as the signal name. To see the analog input signal push the Insert key and enter `ain0.` Your setup window should look like that shown below
8. Close the setup window.
9. At the bottom of the screen there should be an Output window. If not use the View menu to bring it up. The output window will look like that shown below. Type `ADC()` on the command line as shown and push return.
10. Run the simulation for a few seconds and then stop it. Look at your output in the logic analyzer simulation window.



Setup Window for Logic Analyzer



Output window for simulation after entering `ADC()` on the command line and push Enter

Chebyshev Filters

• the Chebyshev filter has an equiripple response in the pass band and improves on the width of the transition band.

• In most cases, if a very flat pass band is not necessary, the Chebyshev filter will provide an implementation that is of lower order and therefore computationally more efficient.

The squared amplitude response for the Chebyshev filter is given by

$$H(s)H(-s) = \frac{1}{1+\varepsilon^2 V_N^2(f/f_{pass})} \tag{6.18}$$

where ε is a parameter related to the pass band ripple, f_{pass} is the edge of the pass band, and $V_N(x)$ is a Chebyshev polynomial defined by

$V_N(x) = \cos(N\cos^{-1}(x))$

To evaluate Chebyschev polynomials we use the following iterative relationships [2] :

$V_0(x) = 1$

$V_1(x) = x$

$V_{k+1}(x) = 2xV_k(x) - V_{k-1}(x)$

From these equations we can make a list of Chebyschev polynomials.

$V_0(x) = 1$

$V_1(x) = x$

$V_2(x) = 2x^2 - 1$

$V_3(x) = 4x^3 - 3x$

$V_4(x) = 8x^4 - 8x^2 + 1$

$V_5(x) = 16x^5 - 20x^3 + 5x$

….

The ripple parameter, ε, is related to the ripple in the pass band, R_p, by the following equation:

$$R_p = 1 - \frac{1}{\sqrt{1+\varepsilon^2}}$$

or

$$\varepsilon = \sqrt{\frac{1}{(1-R_p)^2} - 1}$$

It can be shown that the poles of a Chebyschev filter lie on an ellipse in the *s*-plane [2][3] . The ellipse is defined by two circles as shown in Figure 6.16. The major and minor axes of the ellipse are given by

$$r_{\text{major}} = 2\pi f_{pass}\frac{C^2+1}{2C} \tag{6.19}$$

and

$$r_{\text{minor}} = 2\pi f_{pass}\frac{C^2-1}{2C} \tag{6.20}$$

where

$$C = \left[\frac{1}{\varepsilon} + \sqrt{1 + 1/\varepsilon^2} \right]^{1/N} \tag{6.21}$$

The angle of the poles is the same as the angle for the Butterworth poles: $\pi(1+N+2k)/2N$, k = 0,1,...(2N-1). The angles and pole locations shown in Figure 6.16 are those of a fourth order Chebyschev filter.

From Figure 6.16, we can use geometry to write the pole positions:
$\theta_k = \pi(1+N+2k)/2N$, k = 0,1,...(2N-1)
$x_k = r_{minor}\cos(\theta_k)$
$y_k = r_{major}\sin(\theta_k)$

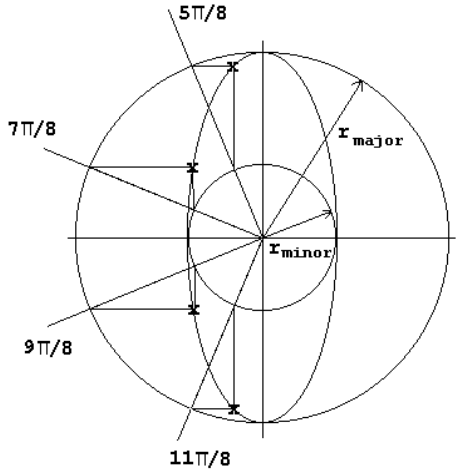


Figure 6.16
The poles of a Chebyschev filter lie on an ellipse in the *s*-plane defined by two circles with radii, r_{major} and r_{minor}.

$$R_{stop}^2 = \frac{1}{1 + \varepsilon^2 V_N^2(f_{stop}/f_{pass})}$$

Solving this equation for V_N^2 gives

$$V_N^2(f_{stop}/f_{pass}) = \frac{1 - R_{stop}^2}{R_{stop}^2 \varepsilon^2} \tag{6.22}$$

The Chebyshev polynomial is defined as

$V_N(x) = \cos(N\cos^{-1}(x))$

In our case x is the quantity f_{stop}/f_{pass} which is greater than 1. The Chebyshev polynomial can also be written as

$V_N(x) = \cosh(N\cosh^{-1}(x))$

where cosh(x) is the hyperbolic cosine defined as

$$\cosh(x) = \frac{e^x + e^{-x}}{2} = \cos(jx)$$

Making this substitution in equation 6.22 and solving for N gives

$$N = \frac{\cosh^{-1}\left(\frac{(1-R_{stop}^2)^{1/2}}{R_{stop}\varepsilon}\right)}{\cosh^{-1}(f_{stop}/f_{pass})} \tag{6.23}$$

Note that if this analog filter is to be converted to a digital filter by way of the BLT, the frequencies f_{stop} and f_{pass} should be *prewarped* frequencies.

Equation (6.23) gives the minimum value for N. In most cases this calculated value will be rounded up to the next highest integer.

MATLAB® has two functions named `cheby1` and `cheb1ord` which automate the algebra. If this is applied to a digital filter, MATLAB® also does the prewarping and an s to z transformation via the BLT.

Example 6.10
Use MATLAB® to design a low pass Chebyshev digital filter that has the following specifications. (These are the same as the Butterworth filter of Example 6.9.)
Sample frequency = 11,025 Hz
Pass band edge = 2,000Hz
Pass band ripple = 0.01
Stop band edge = 2,700Hz
Stop band ripple = 0.05

Solution
The following MATLAB® code produces this filter.

```
fs = 11025;
fpass = 2000;
Rpass = 0.01;RpassDB = -20*log10(1-Rpass);
fstop = 2700;
Rstop = 0.05;RstopDB = -20*log10(Rstop);
[N fp] = cheb1ord(fpass/(fs/2), fstop/(fs/2), RpassDB, RstopDB);
[num den] = cheby1(N, RpassDB, fp);
[H f] = freqz(num, den, 1024, fs);
plot(f, abs(H));
```

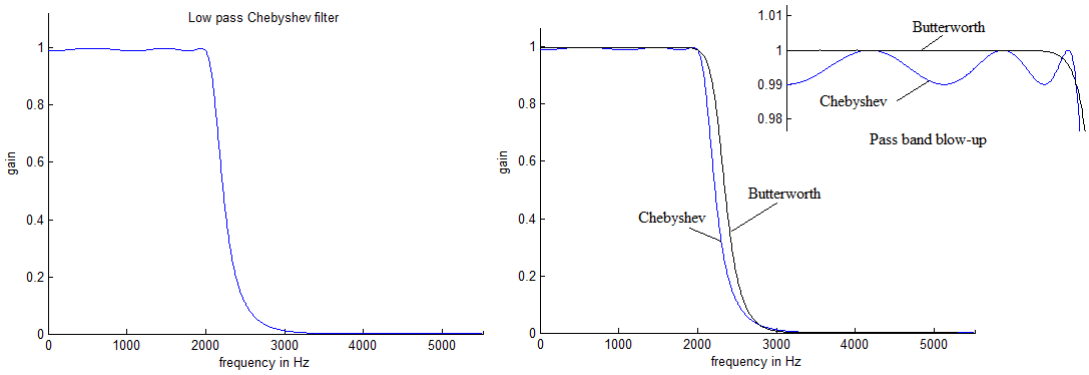
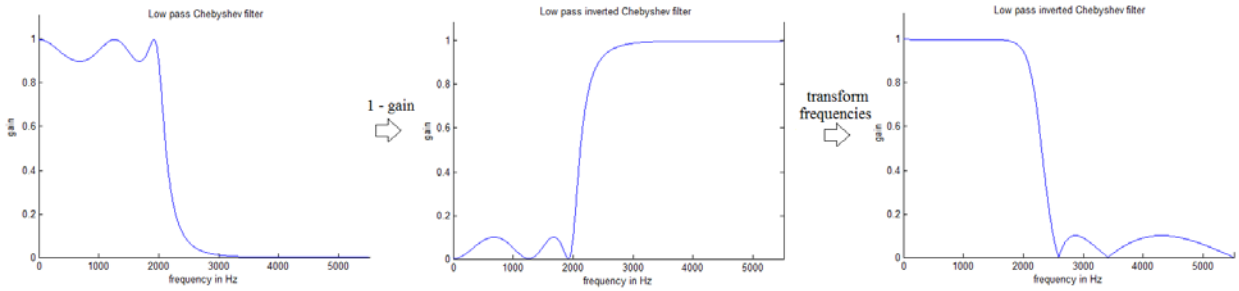


Figure 6.17
Left figure is a low pass Chebyshev filter. Note the ripple in the pass band. Center is the Chebyshev filter compared to a Butterworth filter that meets the same specifications. The Chebyshev filter has a more narrow transition band.

• In designing the Butterworth filter, the cutoff frequency is an important parameter. For a low pass filter the cutoff frequency is normally defined as the point at which the gain falls to $\sqrt{2}/2 = .707$.
• For the Chebyschev filter the important parameter is the frequency point at which the pass band ends (f_{pass}).
• Since we defined the ripple in the pass band as R_{pass}, and we had a gain of unity at 0 Hz, the gain at f_{pass} was 1 - R_{pass}.

• The Chebyschev has noticeable ripple in the pass band but the minimum order necessary to meet the specifications was 6.
• For the Butterworth filter the minimum order was 10. This is a typical result. In most applications, if an absolutely flat pass band is not needed, a Chebyschev filter can be constructed at a lower order.

Inverse Chebyshev filter
• This inverted Chebyshev filter is often referred to as a "type 2" Chebyshev filter where the non-inverted Chebyshev filter is referred to as a "type 1".



(a) **(b)** **(c)**

Figure 6.18
(a) A sixth order non-inverted Chebyshev filter. (b) Subtract the gain in part A from 1. (c) Flip the frequencies so that 0Hz goes to $f_s/2$ and vice-versa.

Example 6.11
Use MATLAB® to design a low pass inverted Chebyshev digital filter that has the following specifications. (These are the same as the Butterworth filter of Example 6.9.)
Sample frequency = 11,025 Hz
Pass band edge = 2,000Hz
Pass band ripple = 0.01
Stop band edge = 2,700Hz
Stop band ripple = 0.05

Solution
The following MATLAB® code produces this filter.

```
fs = 11025;
fpass = 2000;
```

```
Rpass = 0.01;RpassDB = -20*log10(1-Rpass);
fstop = 2700;
Rstop = 0.05;RstopDB = -20*log10(Rstop);
[N fp] = cheb2ord(fpass/(fs/2), fstop/(fs/2), RpassDB, RstopDB);
[num den] = cheby2(N, RpassDB, fp);
[H f] = freqz(num, den, 1024, fs);
plot(f, abs(H));
```

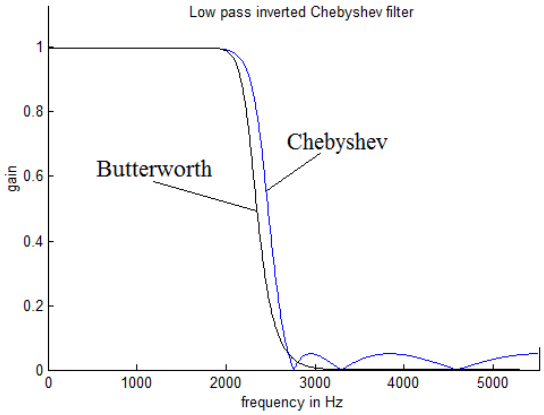


Figure 6.19
A sixth order inverted Chebyshev filter compared to a Butterworth filter which meets the same specifications.

Elliptic Filters

• The Butterworth and Chebyshev analog low pass filters are "all pole" filters (although the inverse Chebyshev filter is not).
• When translated to the *z*-plane by way of the BLT all pole analog filters will have *N*-zeros at the $z = -1$ point which corresponds to $f_s/2$.
• The zero locations produce a monotonic stop band. In the pass band the poles of the Butterworth filter are arranged in such a way as to make the pass band maximally flat. For the Chebyshev filter, some ripple is permitted in the pass band and the pole locations are such that the transition band is sharper than that of the Butterworth counterpart.
• The elliptic filter carries this idea one step further in that it permits ripple in both the pass and stop bands.
• The result is that the elliptic filter has both poles and zeros that can be arranged in close proximity so as to create a filter with a much sharper transition band.

The term "elliptic filter" is used to describe this filter type because it relies on elliptic integrals of the first kind. These have the form [4]

$$f(\varphi,k)=\int_0^{\varphi}\frac{d\theta}{\sqrt{(1-k\cdot\sin^2(\theta)}}$$

In this equation k is called the modulus and φ is called the amplitude. Elliptic integrals cannot be integrated in closed form. They can be written in terms of elliptic functions called Jacobian elliptic functions, and these lead to an infinite series representation for the integral. Thus the design of elliptic filters relies on either an approximation using a truncated infinite series or on tabular look up.

The elliptic filter is said to have an equiripple characteristic in both the pass and stop bands. The term "equiripple" means that the amplitude of the ripple in a band is uniform. As we have seen in the case of FIR filters, this equiripple characteristic is a property of filters that are in some sense optimal. It can be shown that the elliptic filter does provide the lowest order filter to meet a given set of specifications.

The difficulty with the elliptic filter lies in its design. The development and the design of elliptic filters is mathematically complex. The complete development is beyond the scope of this text. We will rely on an intuitive approach based on more rigorous development presented by others.[3]

The general equation for the squared amplitude function of a low pass Butterworth or Chebyschev filter is of the following form:

$$A^2(\omega) = \frac{1}{1+\varepsilon \cdot T^2(\omega)}$$

where ε is a constant and $T(\omega)$ is a rational function.

In this function we see that if $T(\omega) \rightarrow 0$, the amplitude function goes to one. Likewise, if $T(\omega) \rightarrow \infty$, the amplitude function goes to zero.

If we write $T(\omega)$ as the ratio of two polynomials, it is clear that the poles of $T(\omega)$ become the zeros of the amplitude function.

$$A^2(\omega) = \frac{1}{1+C \cdot N^2(\omega)/D^2(\omega)} = \frac{D^2(\omega)}{D^2(\omega)+C \cdot N^2(\omega)} \tag{6.24}$$

Figure 6.20 shows a typical low pass filter which has equiripple in both the pass and stop bands. Since this is the characteristic of the elliptic filter we can relate this figure to the general squared amplitude function (6.24) and thereby derive the transfer function for the elliptic filter.

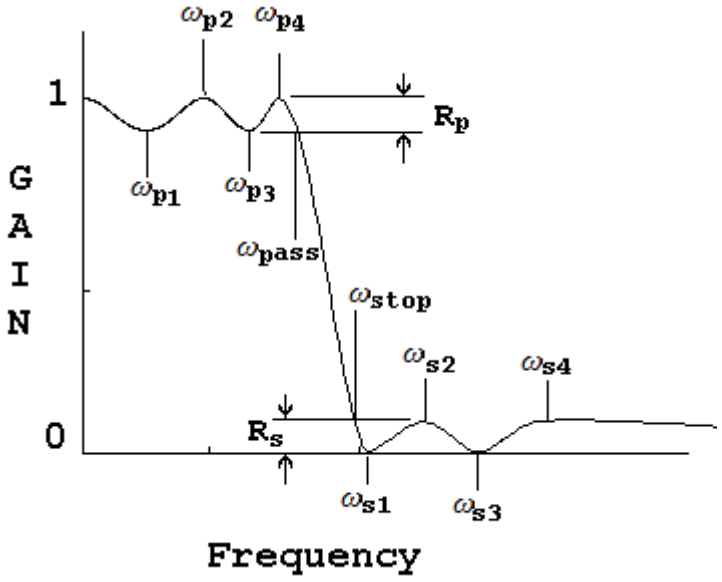


Figure 6.20

A low pass filter with equiripple characteristics.

Thus, from (6.24), we see that the roots of $D(\omega)$ must be at ω_{s1}, ω_{s3}, etc. Likewise, the roots of $N(\omega)$ must be at 0, ω_{p2}, ω_{p4}, etc. We also know the value of the function at ω_{p1} and ω_{p3} is $(1\text{-}R_p)$. The value of the function at ω_{s2}, and ω_{s4} is (R_s). If we take the derivative of the (6.24) we can see from the figure that it will equal zero where the slope of the function equals zero $(0, \omega_{p1}, \omega_{p2}, \omega_{p3}, \omega_{p4}, \omega_{s2}$, and $\omega_{s4})$. Combining this information, it is possible to write a differential equation for the transfer function for the filter. This differential equation, when written in integral form becomes an elliptic integral of the first kind. Such integrals cannot be integrated in closed form so that we must rely on tables or on infinite series approximations. The derivation and approximations necessary for an elliptic filter are beyond the scope of this book[3].

MATLAB® implements the design equations for an elliptic filter in two functions: `ellipord` determines the proper filter order from specifications and `ellip` creates the filter.

Example 6.12
Use MATLAB® to design a digital low pass elliptic filter based on the following specifications. (These are the same as the Butterworth filter of Example 6.9.)
Sample frequency = 11,025 Hz
Pass band edge = 2,000Hz
Pass band ripple = 0.01
Stop band edge = 2,700Hz
Stop band ripple = 0.05

Solution
The following MATLAB® code produces this filter.

```
fs = 11025;
fpass = 2000;
Rpass = 0.01;RpassDB = -20*log10(1-Rpass);
fstop = 2700;
Rstop = 0.05;RstopDB = -20*log10(Rstop);
[N fp] = ellipord(fpass/(fs/2), fstop/(fs/2), RpassDB, RstopDB);
[num den] = ellip(N, RpassDB, RstopDB, fp);
[H f] = freqz(num, den, 1024, fs);
plot(f, abs(H));
```

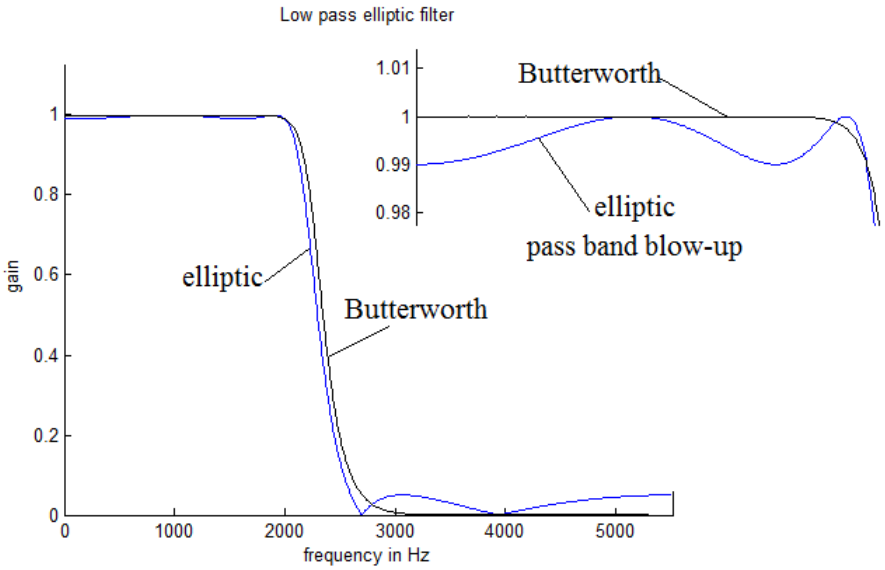


Figure 6.21
A low pass elliptic filter compared to a Butterworth that meets the same specifications.

Summary of Classic IIR Filters

While there are other types of digital IIR filters, the four classic filters (Butterworth, Chebyshev, inverse Chebyshev, and elliptic) presented here are the most popular. A summary of the characteristics of these filters is given in Table 6.4.

Filter Type	Pass Band	Transition Band	Stop Band	Comments
Butterworth	Maximally Flat	wide	Monotonic	Low pass has N zeros at $f_s/2$., simple design procedure, order is relatively high to meet given specifications, stop band has more attenuation than is necessary
Chebyschev	Equiripple	medium	Monotonic	Low pass has N zeros at $f_s/2$, simple design procedure, order is higher than elliptic but lower than Butterworth for given specifications, stop band has more attenuation than is necessary
Inverse Chebyschev	Monotonic	medium	Equiripple	Design is slightly more complicated than Chebyshev, order is the same as Chebyshev
Elliptic	Equiripple	narrow	Equiripple	Design is mathematically complex, order is very low compared to Butterworth or Chebyshev for given specifications

Table 6.4
Characteristics of classic filters.