In the previous lecture we looked at the difference equations for Butterworth and Chebyshev filters both of which had feedback terms in the difference equation. In other words, the present output was a function of not only the present and past inputs but also of past outputs. Such filters are called infinite impulse response (IIR) filters.

It is also possible to build digital filters which do not have feedback terms so that the present output is a function only of the present and past inputs. Such filters are inherently more stable and are called finite impulse response (FIR) filters. They also can be designed so as to not distort the phase of the incoming signal and are therefore useful for many communications applications.

The general difference equation for an FIR filter looks like this:

\[ y_k = b_0 u_k + b_1 u_{k-1} + \ldots + b_m u_{k-M} \]

In this equation \( y_k \) is the output at time \( k \), \( u_k \ldots u_{k-M} \) are the present and past inputs, and \( b_0 \ldots b_m \) are constants.

A Matlab function to generate the constants is called fir1. The format is

\[ b = \text{fir1}(N, \text{wn}); \]

where \( b \) is the coefficients \( b_0 \ldots b_m \), \( N \) is the filter order, and \( \text{wn} \) is the cutoff frequency which is normalized to half the sample frequency.

The Matlab program below generates a 30\textsuperscript{th} order filter along with its coefficients and frequency response curve.

\begin{verbatim}
%FIRTest.m
fs = 11025;  %Sample Freq
N = 30;      %Order
wn = 2000;   %Cutoff Freq
b = fir1(N, wn/(fs/2));
[H f] = freqz(b, 1, 512, fs);
figure(1);clf;
plot(f, abs(H));
disp(b);
\end{verbatim}

\begin{verbatim}
Coefficients
-0.00167174045066 -0.00050583813711  0.00228246839124  0.00399544890271
-0.00019167145685 -0.00909004550306 -0.01042916738156  0.00590940521640
 0.02656639333079  0.01911286327072 -0.02706668109083 -0.06675672784088
-0.02657626185902  0.11615520260173  0.28664084854393  0.36325100692489
-0.28664084854393 -0.0115520260173 -0.02657626185902 -0.06675672784088
-0.02706668109083  0.01911286327072  0.28664084854393  0.36325100692489
-0.01042916738156 -0.00909004550306 -0.0019167145685  0.00399544890271
 0.00228246839124 -0.00050583813711 -0.00167174045066
\end{verbatim}
Note that all of the coefficients are numerator terms since the feedback terms are all zero the denominator is 1. Since this filter has linear phase it is symmetric with $b_0 = b_{30}$, $b_1 = b_{29}$, etc.

The program below implements this filter by storing half of the coefficients in an array and taking advantage of symmetry for the other half.

```c
#include<1pc213x.h>

void main()
{
    float b[16] = {-0.00167174045066, -0.00050583813711, 0.00228246839124,
                   0.00399544890271, -0.00019167145685, -0.00909004550306,
                   -0.01042916738156, 0.00590940521640, 0.02656639333079,
                   0.01911286327072, -0.02706668109083, -0.06675672784088,
                   -0.02657626185902, 0.11615520260173, 0.28664084854393,
                   0.36325100692489};

    float u[31] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    float y;

    unsigned int uInt, dtoaOut;
    int i;

    // AD0CR  = 0x00200000;  //Setup A/D: 10-bit AIN0 @ 4.28MHz
    AD0CR  |= 0x01000000;   // Start A/D Conversion
    PINSEL1 = 0x00480000;   // P0.25 set to DA Out, P0.27 set to input AD0.0
    dtoaOut = 0;
    while(1)
    {
        uInt = AD0DR;         // Read A/D Data Register
        while ((uInt & 0x80000000) == 0)  //Wait for the conversion to complete
            uInt = (uInt >> 6) & 0x000003FF; //Shift into position for 10-bit A/D
        u[0] = ((float)uInt)/1024.0;   // Divide by 1024.0;
        y = 0;
        for(i=0;i<15;i++)
            { y += b[i]*(u[i] + u[30-i]); }
        y += b[15]*u[15];             // u[15] is the center term
        // y will be in the range -1 to +1 so adjust for D to A
        DACR = ((int)((y+1)*512.0)) << 6;
        for(i=30;i>0;i--)
            u[i] = u[i-1];
        AD0CR  |= 0x01000000;  // Restart A/D Conversion
    }
}
```
This version of the program uses timer 0 and an interrupt to set the sample frequency to 11025Hz.
//FIRTest.c
/* This program implements an FIR filter designed by the following Matlab m file. The order is 30 and the length is 31.

This program uses an interrupt to set the sample rate to 11025Hz
*/

#include<lpc213x.h>

int flag = 0;  //used for interrupt
void Timer0ISR(void) __irq;

void main()
{
    float b[16] = {-0.00167174045066, -0.00050583813711, 0.00228246839124,
                   0.00399544890271, -0.00019167145685, -0.00909004550306,
                   -0.01042916738156, 0.00590940521640, 0.02656639333079,
                   0.01911286327072, -0.02706668109083, -0.06675672784088,
                   -0.02657626185902, 0.11615520260173, 0.28664084854393,
                   0.36325100692489};
    float u[31] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                   0};
    float y;
    unsigned int uInt, dtoaOut;
    int i;
    VPBDIV = 0x00000002;  //Set up peripheral clock for 30MHz
    TOPR = 0x00000002;  //Set prescaler to 3 -> 10 MHz timer clock
    TOTCCR = 0x00000002;  //Reset Timer 0 & prescaler; disable prescale cntr
    TOMCR = 0x00000003;  //Generate interrupt and reset Timer on match MR0
    TOMR0 = 0x00000038B;  //Match value = 907 or 38Bh -> 11025 Hz
    TOTCR = 0x00000001;  //Start counter 0
    //
    VICVectAddr4 = (unsigned)Timer0ISR;  //Assign addr of ISR to int addr reg
    VICVectCntl4 = 0x00000024;  //bit 5 enables interrupt, bits 0-4 set ch.
    VICIntEnable |= 0x00000010;  //Enable the interrupts
    //
    AD0CR = 0x000200601;  // Setup A/D: 10-bit AIN0 @ 4.28MHz
    AD0CR |= 0x001000000;  // Start A/D Conversion
    PINSEL1 = 0x00480000;  // P0.25 set to DA Out, P0.27 set to input AD0.0
    dtoaOut = 0;
    while(1)
    {
        uInt = AD0DR;  // Read A/D Data Register
        while ((uInt & 0xB0000000) == 0)  //Wait for the conversion to complete
            uInt = AD0DR;
        // Filter implementation
        for (i = 0; i < 16; i++)
            y += b[i] * u[(i + uInt) % 31];
        // Output processing
        dtoaOut = (unsigned int) y;
        // Interrupt handling
        if (flag)
        {
            // Interrupt service routine
            Timer0ISR();
            flag = 0;
        }
        // Main program
        //...
uInt = ((uInt >> 6) & 0x000003FF); //Shift into position for 10-bit A/D
u[0] = ((float)uInt)/1024.0;   // Divide by 1024.0;
y = 0;
for(i=0;i<15;i++)
{y += b[i]*(u[i] + u[30-i]);}
y += b[15]*u[15];
DACR = ((int)((y+1)*512.0)) << 6;
for(i=30;i>0;i--)
u[i] = u[i-1];
while(flag == 0);            // Wait for interrupt
flag = 0;                    // Set flag for next iteration
ADOCR |= 0x01000000;        // Restart A/D Conversion
}

void Timer0ISR(void) __irq
{flag = 1;                 //reset interrupt flag
T0IR |= 0x00000001;       //Clear match 0 interrupt
VICVectAddr = 0x00000000; //Dummy write to signal end of interrupt
}

---

**Implementing difference equations using integer arithmetic**

On many processors integer arithmetic is implemented considerably faster than is floating point arithmetic. In addition, many DSP chips are available in both integer and floating point versions and the integer versions, in addition to being slightly faster, are often significantly cheaper.

Doing a digital filter in integer arithmetic is approximately the same as doing it in floating point arithmetic except for the process of scaling. Floating point numbers have a very large range and overflow is not normally a problem. But if integer arithmetic is used, filters must often be carefully scaled to avoid overflow. This is more of a problem on smaller processors and often double registers or long integers must be used.

Since the ARM processor uses a 32 bit integer format, overflow is a minor problem at best. The program below shows how to implement a 3rd order Butterworth filter using integer arithmetic on the ARM processor.

/*ButterI3.c                                  August 30, 2005
This program implements a 3rd order Butterworth filter using
integer arithmetic.

\( \frac{0.0509232z^3 + 0.1527696z^2 + 0.1527696z + 0.0509232}{z^3 - 1.1415667z^2 + 0.6831606z^2 - 0.1342084} \)

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
*/
#include <LPC213X.H>

/*Multiplier for fractional constants is 2^14
const float b0 = 0.0509232;  //0.0509232*16384 = 834
const float b1 = 0.1527696;  //0.1527696*16384 = 2503
const float b2 = 0.1527696;  //0.1527696*16384 = 2503
const float b3 = 0.0509232;  //0.0509232*16384 = 834
const float a1 = -1.1415667; //-1.1415667*16384 = -18703
const float a2 = 0.6831606;  //0.6831606*16384 = 11193
const float a3 = -0.1342084; //-0.1342084*16384 = -2199
*/

const int b0 = 834;
const int b1 = 2503;
const int b2 = 2503;
const int b3 = 834;
const int a1 = -18703;
const int a2 = 11193;
const int a3 = -2199;

void main(void)
{
int uInt, u, y;
int u1, u2, u3;
int y1, y2, y3;
int dtoaOut;
VPBDIV = 0x02;          //Set the Pclk to 30 Mhz
ADOCR = 0x00200601;    // Setup A/D: 10-bit AIN0 @ 4.28MHz
ADOCR |= 0x01000000;   // Start A/D Conversion
PINSEL1 = 0x00480000;   // P0.25 set to DA Out, P0.27 set to AD0.0
dtoaOut = 0;
while(1)
{
uInt = AD0DR;                // Read A/D Data Register
while ((uInt & 0x80000000) == 0)  //Wait for the conversion
  uInt = AD0DR;
  u = ((uInt >> 6) & 0x000003FF); //Shift for 10-bit A/D
  y = b0*(u + u3) - a1*y1 + b1*(u1 + u2) - a2*y2 - a3*y3;
  //Divide by 16384 = 2^14 to remove constant plus shift left 6
  //   places for D to A Register gives divide by 2^8
  y = ((y >> 14) & 0x000003FF);
  DACR = y << 6;
  ADOCR |= 0x01000000;             //Restart A/D Converter

  y3 = y2;
y2 = y1;
y1 = y;
u3 = u2;
u2 = u1;
u1 = u;
}
}