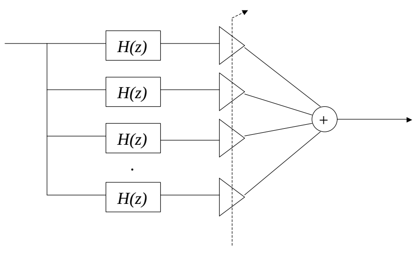
**Applied signal processing**

**Lab 4, May-2013**

In this lab, we will implement a "graphic equalizer" commonly found in stereo equipment using digital signal processing methods. The basic function of a graphic equalizer is to divide the signal into to frequency bands and apply a gain on these frequency bands. Digitally, this can be done using some different approaches. Here, two ways are explained. The "equalizer" can be realized using a filter bank consisting of band pass filters. Applying different gains after these filters the requested functionality can be achieved, as shown in figure 1. The filters can be realized as IIR band-pass filters, they can be designed as butterworth filters. The problem with this approach is that the IIR filters introduce phase shifts, and when adding the signal from the filters the overall response of the system is not flat as expected. In this lab, we however disregard this issue.



The second approach is to construct a FIR filter that conforms to the requested frequency response. This is also called a frequency sampling filter, and the coefficients can be obtained by a inverse DFT of the frequency samples describing the requested frequency response of the filter:

$$h(n) = 1/N \sum_{k=0}^{N-1} H(k) e^{j(2\pi/N)nk} \tag{1}$$

where $H(k)$ are the frequency samples at intervals of $kF_s/N$, $N$ is the number of frequency samples and $h(n)$ are the filter coefficients. $F_s$ is the sampling frequency. For linear phase filters, with positive symmetrical impulse response, we can write

$$h(n) = 1/N \left[ \sum_{k=1}^{N/2-1} 2|H(k)| \cos(2\pi k(n-\alpha)/N) + H(0) \right] \tag{2}$$

where $\alpha = (N-1)/2$). The resulting filter will have a frequency response that is exactly the same as the original response at the sampling instants. However, between the sample instants, the response may be significantly different. To obtain good approximation, we must take a sufficient number of frequency samples.

**Tasks using MATLAB**

Calculate the IIR filter coefficients needed for the equalizer. Divide the the frequency spectrum at 100, 300, 1000 and 3000 Hz, hence the following filters are needed:

1. A low pass filter, cut-off frequency 100 Hz.
2. Three band pass filters 100-300 Hz, 300-1000 Hz, 1000-3000 Hz.

3. A high pass filter, cut off frequency 3000 Hz.

For designing, use MATLAB and the butterworth second order filters (function butter). Note that frequencies entered to function butter are normalized to 0.0-1.0, where 1.0 corresponds to half the sampling frequency ($Fs/2$). Calculate for both 8 kHz and 48 kHz sampling frequencies.

**Tasks at the lab**

1. Implement the IIR filter bank. IIR filters are normally realized as second-order building blocks. This is called a biquadratic section and is characterized by the following equation (direct realization)

$$y(n) = \sum_{k=0}^{2} b_k x(n-k) - \sum_{k=1}^{2} a_k y(n-k) \tag{3}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{4}$$

   Here note that old outputs $y(n-k)$ have to be stored for each filter. Usually higher order filters are realized as a cascade of biquadratic filters. However, when experimenting, the IIR can easily be expanded to higher order:

$$y(n) = \sum_{k=0}^{K} b_k x(n-k) - \sum_{k=1}^{K} a_k y(n-k) \tag{5}$$

   where $K$ is the order of the filter. Implement the filter bank and use the pre-calculated filter coefficients to implement the graphic equalizer.

2. Implement the graphic equalizer using a FIR filter. A FIR filter is calcluated as in previous exercises as a convolution:

$$y(n) = \sum_{k=1}^{K} h(k) x(n-k) \tag{6}$$

   The filter coefficients $h(k)$ kan be calculated using equation 2.

## Notes on TMS 320C671X

### DSP-library

The DSP-library is a set of optimized routines written in assembler to make maximum use of the DSP functionality of the processors. For example, a FIR filtering of length 2000 taps is possible to do at the speed of 48 kHz. For instance, doing FIR filtering is done using the function fir.

```
void DSPF_sp_fir_gen(const float *x, const float *h,
float * restrict r, int nh, int nr)
x Pointer to array holding the input floating-point array.
h Pointer to array holding the coefficient floating-point array.
r Pointer to output array
nh Number of coefficients.
nr Number of output values.
```

It is not required to test the library in the lab, but for those who want to learn more, this is a good extra exercise!

**Dip-switches**

To read the value of the DIP-switches, the following function can be used:

```
int DSK6713_DIP_get(int num) // 6713
where num is the DIP-siwtch number (0-3) and the function returns 1/0
```