

# Microcontrollers LAB 4

## FIR filtering using ASSEMBLY

Coordinator: L. Geurts

Cooperator: T. Stas



# FIR filters in Assembly

## Introduction

A filtering operation is very common in many digital applications we know today. It is e.g. used as part of the decoding of a voice signal received through GSM, or to compress or decompress an audio or video signal in real time. In order to process all the samples in time, efficient algorithms are needed, and implementations in Assembly are still the most efficient. Today you will implement a so called FIR filter.

## FIR filters

The formula below is that of an eight-order Finite Impulse Response (FIR) filter.  $y(n)$  is the current output sample to be calculated,  $x(n)$  is the current input sample,  $x(n-1)$  is the previous input sample, and so on.  $b(0)$  up to  $b(8)$  are called the filter coefficients. According to the formula the output sample is a weighted sum of a set of input samples, in which the filter coefficients are the weights. The order of the filter reflects how many input samples are weighed, and in this case the order is 8.

$$y(n) = b(0).x(n) + b(1).x(n-1) + b(2).x(n-2) + \dots + b(7).x(n-7) + b(8).x(n-8)$$

One can prove (see course on Digital Signal Processing) that this filter has a finite impulse response, hence the name FIR filter. The filter you will implement in the lab is a low-pass filter, with a frequency response (~Bode plot) as shown in Figure 1.

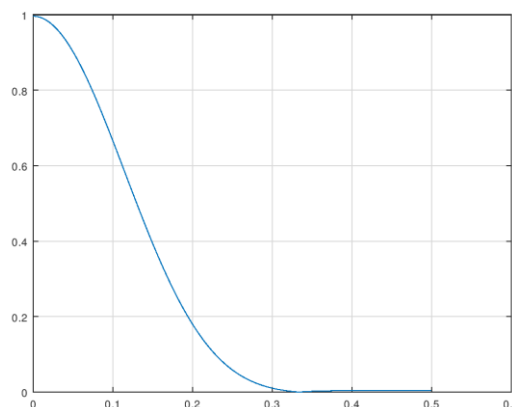


Figure 1 Frequency response of the first FIR filter. The horizontal axis corresponds with the normalized frequency (i.e. the frequency of the signal divided by the sampling frequency). The vertical axis corresponds to the gain. Since higher frequencies are strongly attenuated, this characteristic is that of a low-pass filter.

## Filter coefficients

In order to obtain the filter coefficients, software tools like Matlab or Octave can be used. For an eighth order low-pass filter that cuts off at 0.1 times the sampling frequency, one obtains the following values (from  $b(0)$  to  $b(8)$ ):

0.0051719, 0.0295410, 0.1108839, 0.2191328, 0.2705408,  
0.2191328, 0.1108839, 0.0295410, 0.0051719

These cannot be used as such on our processor, since this is not a fixed-point processor. The PIC18F contains an 8x8 multiplier that can multiply two unsigned chars. A trick to deal with this is to multiply the coefficients with 256, and then one obtains:

1      8      28      56      69      56      28      8      1

As a consequence, the output will also be 256 larger than before.

### Buffers and indirect addressing

The **filter coefficients** will be stored in a buffer starting at address 0x30:

0x30	b(0)
0x31	b(1)
0x32	b(2)
0x33	b(3)
0x34	b(4)
0x35	b(5)
0x36	b(6)
0x37	b(7)
0x38	b(8)

For the **input signal**, **sinusoids** will be used with **frequencies** up to **half the sampling rate**. In order to ensure only positive input values for the A/D converter, a **DC off-set around 2.5 V** has to be added. The **signal generator** in the lab allows the addition of this off-set. Also the input samples will be stored in a buffer, more specifically in a circular buffer:

0x20	x(n-3)	
0x21	x(n-2)	
0x22	x(n-1)	<<< previous sample
0x23	x(n)	<<< current sample
0x24	x(n-8)	<<< oldest sample
0x25	x(n-7)	
0x26	x(n-6)	
0x27	x(n-5)	
0x28	x(n-4)	

When a new sample arrives, the oldest sample will be overwritten. **Indirect addressing** will be used for both buffers (coefficients and samples). The **File Selection Registers FSRx** will point to the appropriate element in the buffer. Use **FSR0** for the input samples, and **FSR1** for the coefficients.

### Implementation issues

Since both **input samples** and **filter coefficients** are positive, **unsigned multiplication** can be used (see instruction **MULWF**), which makes the implementation easier. Note that the multiplication of two bytes, yields a 16-bit value, stored in registers **PRODH:PRODL**.

For each incoming sample, the above formula has to be calculated. Apply the **multiply-and-accumulate method** to implement the filter. Use two registers to store the filter output, a low byte and a high byte, and **clear them before each filter cycle**. Perform the multiplication of each coefficient with the appropriate input sample, and add it to the value in the output register. Don't forget the **carry obtained after adding the two low bytes**! The final result should be divided by 256, but this division does not need to be carried out, since one can **simply take the high byte as the final result**.

### Assignment:

The template program on Toledo "FIRtemplate.asm" already includes many components you need for this lab. An A/D converter is used to read the input sample (pin AN0), and the **sampling frequency** is determined by the **settings of Timer0**. Your lab coach will assign a sampling frequency to you, and you have to determine the right settings.

The result of the A/D converter is sent directly (so without FIR filter) to the PWM module. The **duty ratio** is set equal to **the last input sample**. Design a passive RC filter (low-pass) with a cut-off at half the sample rate to smooth the PWM signal.

Try to upload the altered template program. Test the functionality on your microcontroller board. Prove that the sampling frequency is correct (be creative here).



### Assignment "LOW PASS FIR FILTER":

#### 1. FIR filter implementation

Write the code for the FIR filter according to the above guidelines. Before testing it on the board, simulate the code in MPLAB, without A/D conversion, timers or interrupts (so just the filter code). Use fake numbers for the input sample, e.g. all equal to "1". While simulating, watch the registers relevant for indirect addressing (like FSR0 and FSR1), and the filter output. Only this way you will be sure that your code works properly.

#### 2. Test on PIC board

Now integrate your filter code in the template. Take care of the circular buffer for the input samples. Make sure the latest sample is stored in the right place. Apply a test signal at the input and monitor the output. Your test signal should be a sinusoid with varying frequency and a DC off-set halfway the supply voltage. Make sure your signal stays in the range of the A/D converter. Observe the output. It should show low-pass behaviour.

#### 3. Bode plot

Measure the frequency response of your system. Compare your results with the theoretical values (see excel file on Toledo).

#### 4. Music

Apply a music signal to the input, using a sampling frequency of 10 kHz. Create a DC off-set with a simple voltage divider, and apply the signal with a coupling capacitor. Use a decoupling capacitor at the output. Listen to the effect.

### ★ Assignment "HIGH PASS FIR FILTER":

Repeat the assignment above, but now with other filter coefficients. The result should be a high pass filter. Here below you find some extra guidelines.

The coefficients for an 8<sup>th</sup> order high-pass FIR filter are:

-1 -6 -21 -41 206 -41 -21 -6 -1

Since there are negative coefficients, signed multiplication should be used. One way to implement this is to write code that makes use of unsigned multiplication (see lecture on data types and arithmetic).

An alternative approach is this one:

$$y(n) = b(0).x(n) + b(1).x(n-1) + \dots + b(8).x(n-8)$$

Now, add the absolute value of the largest negative value (-41) to all coefficients, so

$$b'(i) = b(i) + 41$$

All these new coefficients are positive!

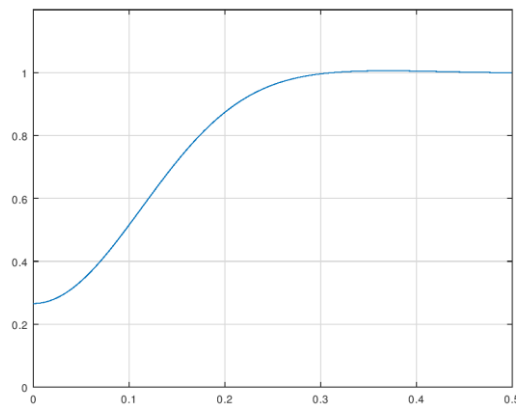
Then the formula for the filter output becomes:

$$y(n) = [b'(0)-41].x(n) + [b'(1)-41].x(n-1) + \dots + [b'(8)-41].x(n-8)$$

$$y(n) = [b'(0).x(n) + b'(1).x(n-1) + \dots + b'(8).x(n-8)] - 41.[x(n) + x(n-1) + \dots + x(n-8)]$$

The first term uses only unsigned multiplication, and so, the result will be positive. The same code as for the low-pass filter can be used. To obtain the correct filter output, the sum of the last 9 samples, multiplied with 41, has to be subtracted from the first term. Be careful however, that the first term does not overflow. If this occurs, an upper byte should be used, besides the high byte and the low byte.

The filter characteristic should look like this in theory (see Figure 2).



*Figure 2 Frequency response of the second FIR filter. Since now the lower frequencies are strongly attenuated, this characteristic is that of a high-pass filter.*

The DC gain = 0,265625, so that means that the DC off-set we used at the input will be attenuated. In order to compensate for that, the value 94 should be added to the PWM duty ratio ( $94 = 128 - 128 \cdot 0.265625$ ).