

Digital Filtering:

TD 02: FIR filters (Python and SigmaDSP)

Antonin NOVAK

Laboratoire d'Acoustique de l'Université du Maine, Le Mans, France



Goal:

FIR (finite impulse response) **filters**

- theory refresh and concept behind

work:

- FIR filters in Python (`scipy.signal.lfilter`)
- implement FIR filters in Sigma Studio (different ways)
- circular buffer implementation in Python



1 FIR filters

Work

two types of linear digital filters:

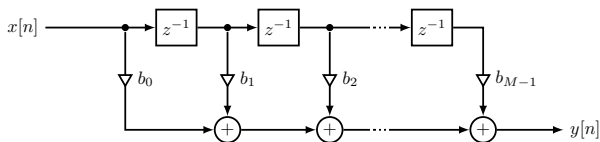
FIR finite impulse response

impulse response settles to zero in finite time

IIR infinite impulse response

impulse response is of infinite length (due to feedbacks)

FIR filters



filter equation:
$$y[n] = \sum_{m=0}^{M-1} b_m x[n - m]$$

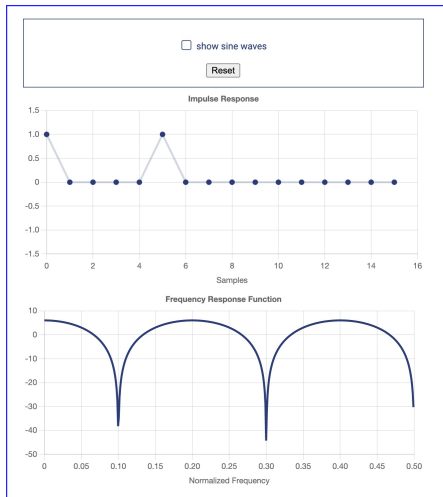
convolution equation:
$$y[n] = \sum_{m=0}^{M-1} h_m x[n - m]$$

b_m ... filter coefficients

h_m ... impulse response coefficients

FIR concept (web application)

<https://imdea.ant-novak.com/DF/FIR>



FIR filters: Frequency Respose Function

$$H_{2\pi}(\Omega) = \sum_{m=0}^{M-1} b_m e^{-j\Omega m}$$

$$H[k] = \sum_{m=0}^{M-1} b_m e^{-j\frac{2\pi}{M}km}$$

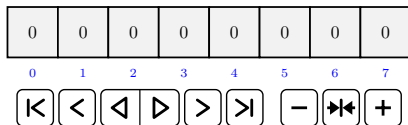
$\Omega = 2\pi F$... normalized angular frequency $\langle 0, 2\pi \rangle$

F ... normalized frequency $\langle 0, 1 \rangle$

$F = 1$ and $\Omega = 2\pi$... sampling frequency

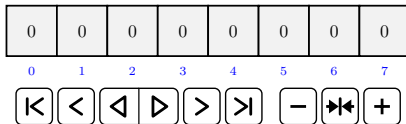
FIR implementation using circular buffer

Circular buffer (memory space) is a single fixed-size buffer as if it were connected end-to-end

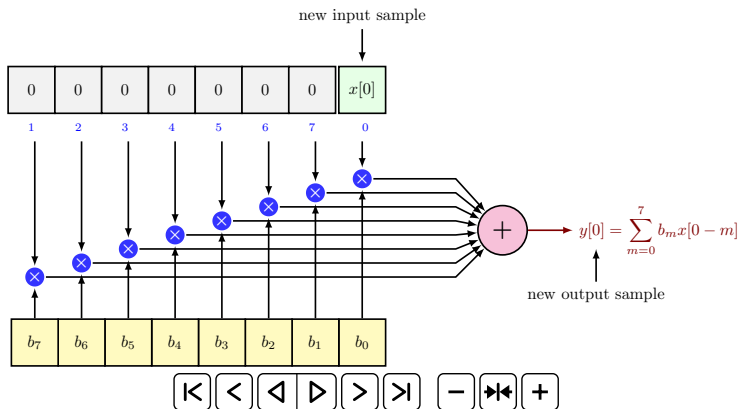


FIR implementation using circular buffer

Circular buffer (different point of view)



FIR implementation using circular buffer





2

FIR filters

Work

Work 1: FIR in Python (scipy.signal.lfilter)

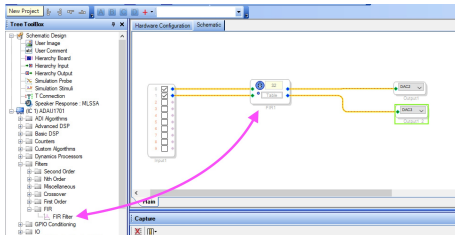
Use FIR filter in Python and export its coefficients.

1. Create a numpy array x representing a Dirac signal (length = 1000).
2. Create a numpy array b representing FIR filter coefficients such that $y[n] = 0.5 x[n] + 0.5 x[n - 5]$.
3. Use `scipy.signal.lfilter()` function to get the output y of the filter.
4. Use `scipy.fft.fft()` to plot the spectra of y and compare it with the FRF one obtained from `scipy.signal.freqz()`
5. Change the filter coefficients to a low pass filter generated using `scipy.signal.firwin(32, .2)` and run the code to verify the `fft` and `freqz` give same results.
6. Export the filter coefficients b to a text file using `np.savetxt('coeffs.txt', b.T)`.

Work 2: FIR in Sigma Studio (block)

Use FIR filter (with coefficients generated in Python) on SigmaDSP.

1. Open Sigma Studio, create a new project in which you place the ADA1701.
2. Place an **FIR filter** as depicted bellow, make it 2-channel (grow algorithm), and set the length of the filter to 32 (the same length of the exported filter from Python).

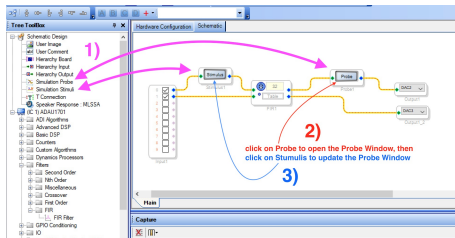


3. Click on "Table" on the FIR block to open the table with filter coefficients and click on "Load table values" to load the values saved in `coeffs.txt` (File exported from python).
4. Compile and load your program to DSP and test connecting your music and phones.

Work 3: Stimuli / Probe in Sigma Studio

Learn using Stimuli / Probe for FRF simulated measurement.

1. Place **Probe** and **Stimulus** blocks as depicted bellow.

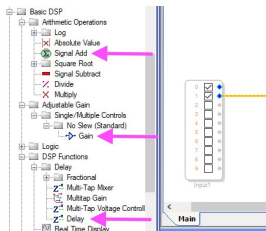


2. Then click on **Probe** to open the graphical window depicting FRF and then click on **Stimulus** to update the window. The FRF of the FIR filter should appear.

Work 4: FIR in Sigma Studio (add, gain, delay)

Build a FIR filter in Sigma Studio for which $y[n] = b_0x[n] + b_{20}x[n - 20]$.

1. Create a FIR filter using blocks **Add**, **Gain**, and **Delay** (see Figure bellow to find them).

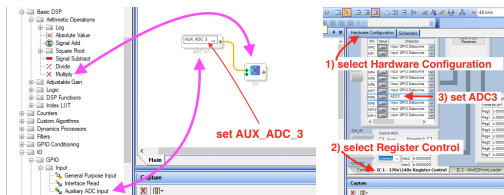


2. Coefficients b_0 and b_{20} are implemented as **Gain**. Start with $b_0 = 1$ and $b_{20} = 1$. Put also another **Gain** at the end of the filter (gain = 0.5) to avoid clipping.
3. Use **Probe** and **Stimulus** to verify the FRF.
4. Observe both the FRF depicted by **Probe** and the audio output and how they are influenced by the value of b_{20} .

Work 5: Coefficient modification in real-time

Make the coefficient b_{20} changeable in real time using the potentiometer placed on the evaluation board.

1. Replace the block **Gain** (representing b_{20}) by **Multiply** block and multiply the signal by **Auxiliary ADC input** (see the left Figure below)



2. To make the potentiometer available, you need to set the ADC3 as multipurpose pin MP8 in registers (see right Figure above).
3. Use the block **DSP Readback** available in **Basic DSP** → **DSP Functions** to read the value of the ADC3. Place it after the **Auxiliary ADC input** and press Read to update.
4. Listen to the output while changing the value of the potentiometer.
5. Note that the blocks **Probe** and **Stimulus** do not work with the potentiometer.

Work 6: Circular buffer in Python, sample-by-sample processing

- ▶ Start with the code bellow generating a sine signal as the input signal.

```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

fs = 48000          # sample rate [Hz]
T = 5e-3           # signal duration [s]
f0 = 1e3           # frequency [Hz]
N = 5              # circular buffer size

t = np.arange(0, T, 1/fs) # time vector
x = np.sin(2*np.pi*f0*t)  # input sine signal

circular_buffer = np.zeros(N) # circular buffer initialization
pointer = 0                # pointer initialization
```

- ▶ Add a for loop to process the input signal sample-by-sample:
 - ▶ Inside the loop, implement the circular buffer using a numpy array and a pointer to the current position in the buffer.
 - ▶ Use modulo operation % to reset the pointer when it reaches the end of the buffer.
 - ▶ Print the buffer during each iteration to see how it changes using:

```
print(f'Buffer: {circular_buffer}')
input('Press Enter to continue...')
```

Work 7: FIR Implementation in Python

- ▶ Continue with the previous code to implement the FIR filter.
- ▶ Use the impulse response `h` and do the filtering using `lfilter()` as shown bellow.

```
h = signal.firwin(15, 100/fs) # FIR filter coefficients
N = len(h)                   # circular buffer size

y0 = signal.lfilter(h, 1, x) # output signal using lfilter
y = np.zeros(len(x))         # output signal initialization
```

- ▶ Implement the convolution inside the loop, sample by sample.
- ▶ Use the algorithm from page 10.

Project

Start working on the project.

The project assignment is available on
<https://imdea.ant-novak.com/DF>.