Daltech, Dalhousie University
Department of Electrical and Computer Engineering

ECED 4502 – Digital Signal Processing

Lab 3 – FIR Filter Design using GNU Radio

Objectives

The objectives of this lab are to observe the characteristics of the frequency response of finite impulse response (FIR) digital filters and to understand the main elements involved in their design by using GNURadio.

Background

The background material for this Lab is presented in Appendix A and relevant code is in Appendix B.

Procedure

1. Use the previously posted instruction on Brightspace to install GNURadio. Once Installation is finished. Open "GNURadio Companion". It will open an initial screen of GNURadio as illustrated in Fig. 1.
2. When the initial splash screen appears, choose "**File→Open**". Navigate to the downloaded file directory and choose "**FIR_filter_design.grc**", it should bring the screen shown in the Fig. 2 and run the program by going into "**Run→Execute**" or press **F6**. The screen showed in Fig.4 should appear with the default configuration.
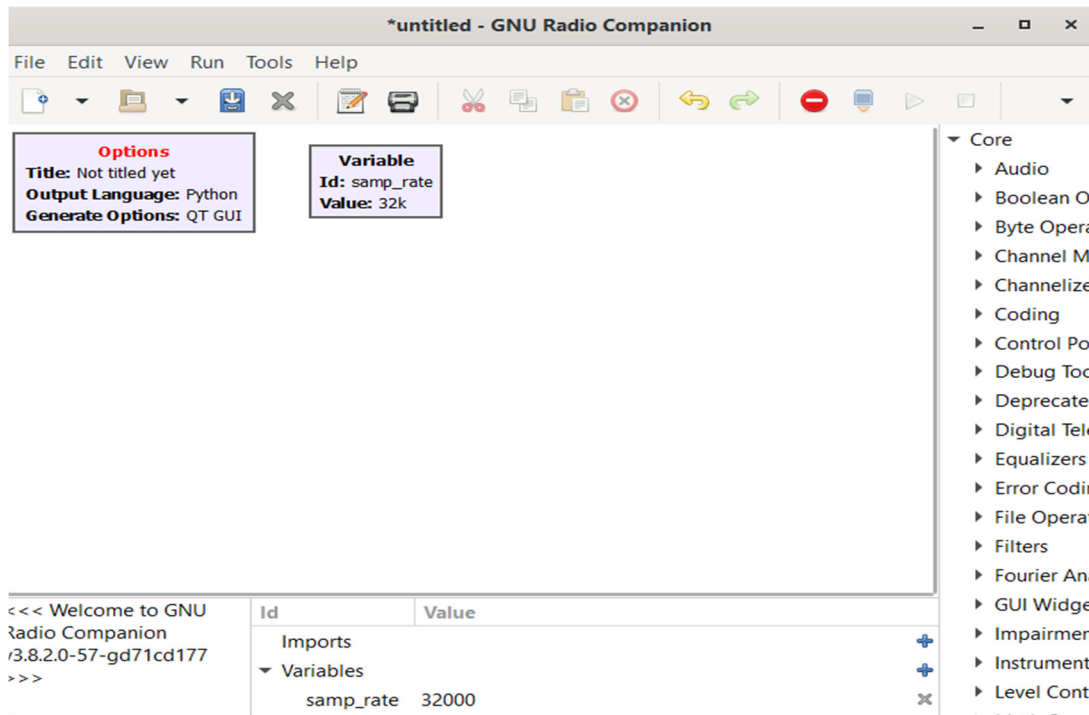3. Refer Fig. 3 which illustrate the different blocks of the Flow Chart.
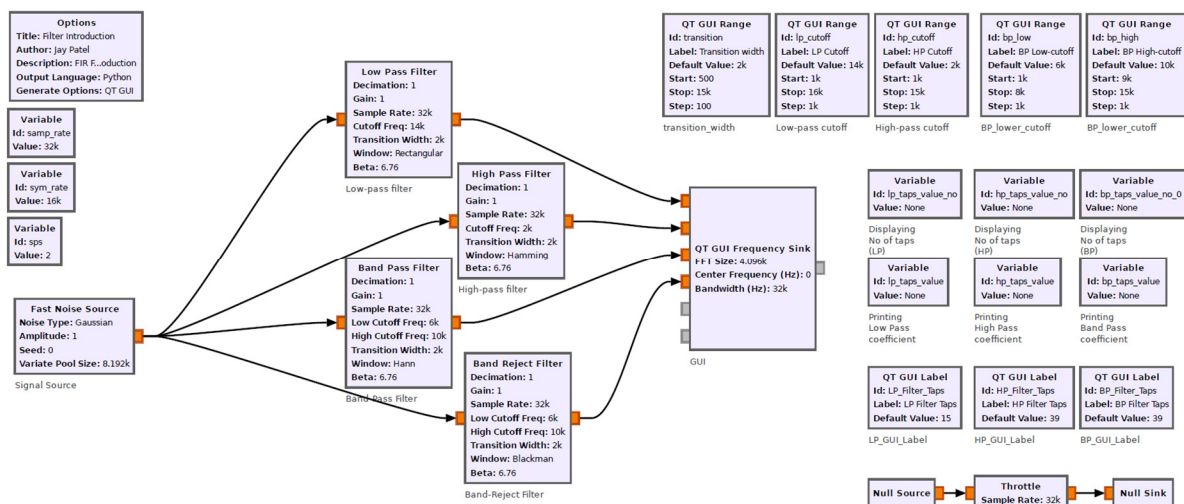
Fig.1. Initial GNURadio welcome screen

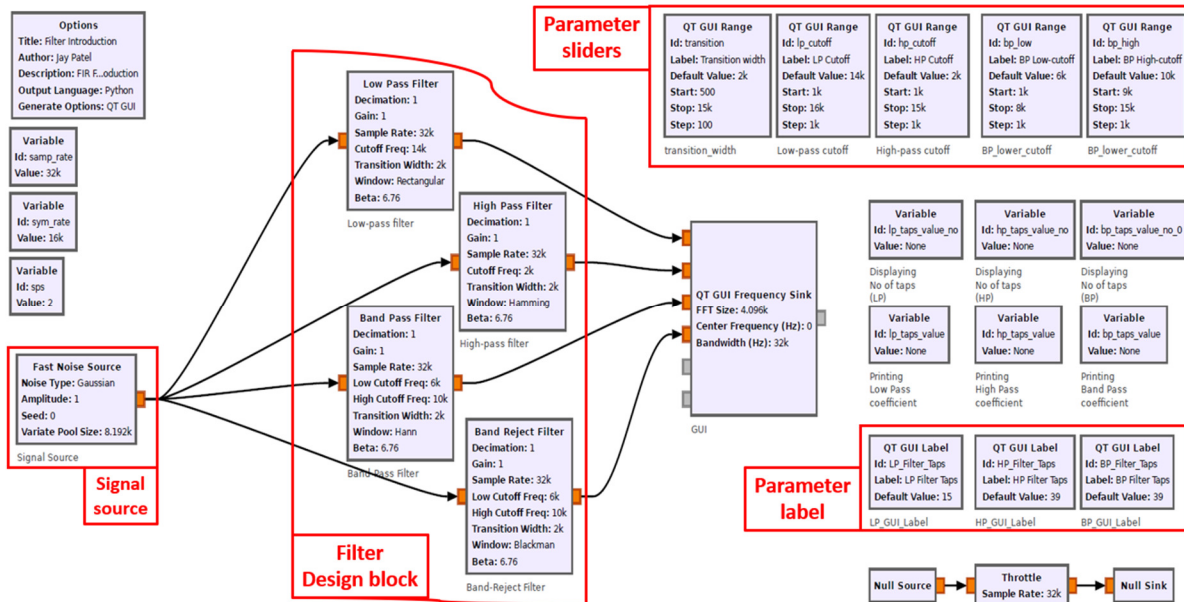Fig.2. GNURadio Screen with the FIR filter design experiment setup
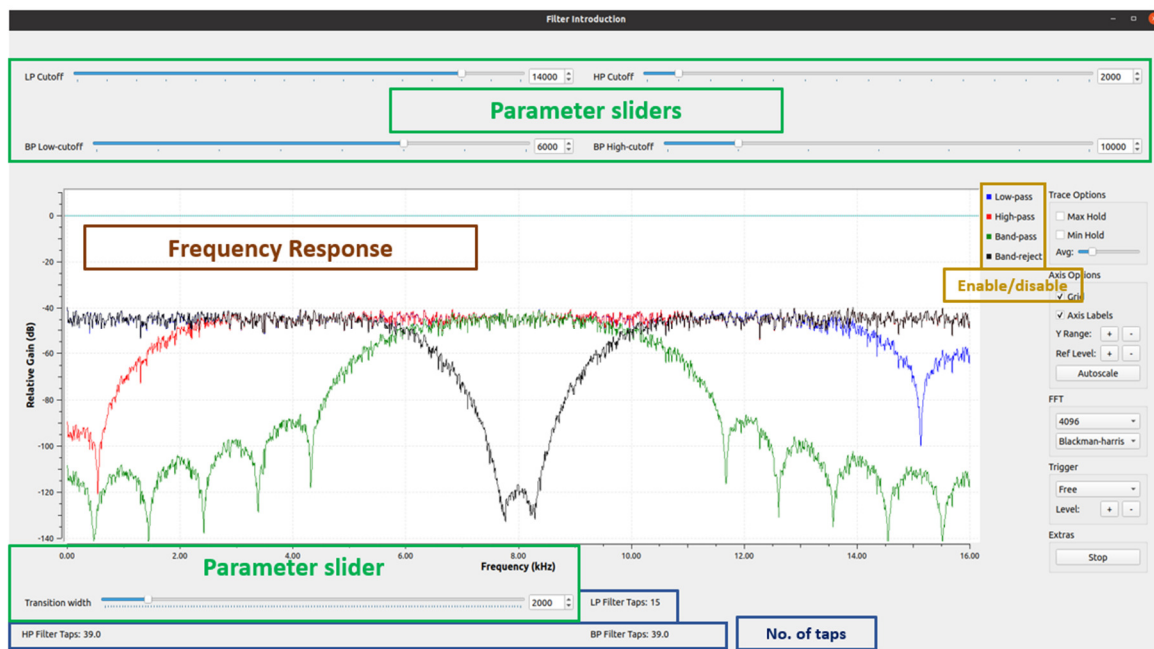
Fig.3. Introduction to Flow Graph
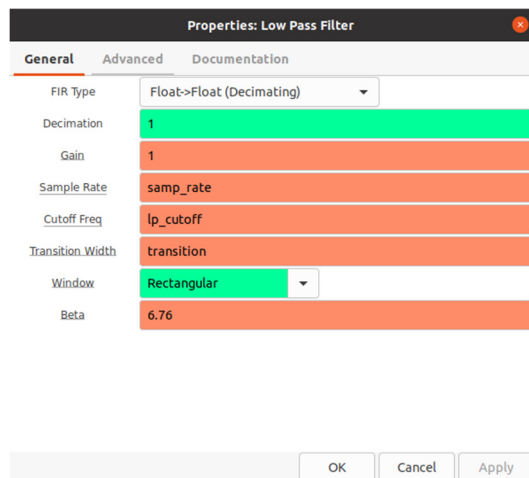
Fig.4. GNURadio Output QT GUI

a. Low pass filter block configuration   b. High pass filter block configuration

c. Band pass filter block configuration   d. Band reject filter block configuration

Fig.5. Filter design block configuration

4. Figure 3 shows the filter design blocks with the parameter tabs needed to design an FIR filter. Figure 4 shows the frequency response plots for all four filters: low-pass, high-pass, band-pass and band-stop. The filter parameters are given in "**FIR Filter Settings**" which you can open by double clicking on the respective filter, see figure 5. As a part of the screen (figure 4) where you can select (i) the filter type – you should try first the low-pass filter; (ii) the number of coefficients (taps) in your filter (the more taps, the better approximation of the ideal filter you will get) and (iii) the cut-off frequency of your filter – the cut-off frequency should be always smaller than *fs/2* with *fs* selected either through sliders or counter shown in the upper part of the GUI. You could try to design your filter using different windows, number of taps, and by using different configuration setting, you could observe better distinction from the QT GUI Label shown in the lower part of the GUI which displays the length of the window in number of taps, a.k.a the number of filter coefficients.

The coefficients in your design are printed in the "**command prompt of GNURadio**" see figure 6 below. The number of filter coefficients, obtained by adjusting various parameters in the GUI such as transition band width, LP and HP cut off (passband corner) frequencies, can be seen at any time at the bottom of the GUI.

- Enable/Disable the filter response by clicking on the name of GUI ;
- Change the transition bandwidth and observe the filter responses and number of taps needed for filter design and also compare the effect of different windows on the number of taps and relative gain ;
- Change the number of taps (length of your filer) from 77 to 39 to 9 and 5 taps and observe different filter responses.
- Record your filter frequency responses for the given values of the taps.

No. of Taps	Transition band width	Cut-off frequency
77		
51		
39		
25		
19		
11		
09		
05		

- Record filter coefficients as you can implement the same filter using MatLab or Python (refer to Appendix B for more details).

- 3 -

- Comment on your observations and compare the results to the frequency response of the filter displayed in GNURadio.
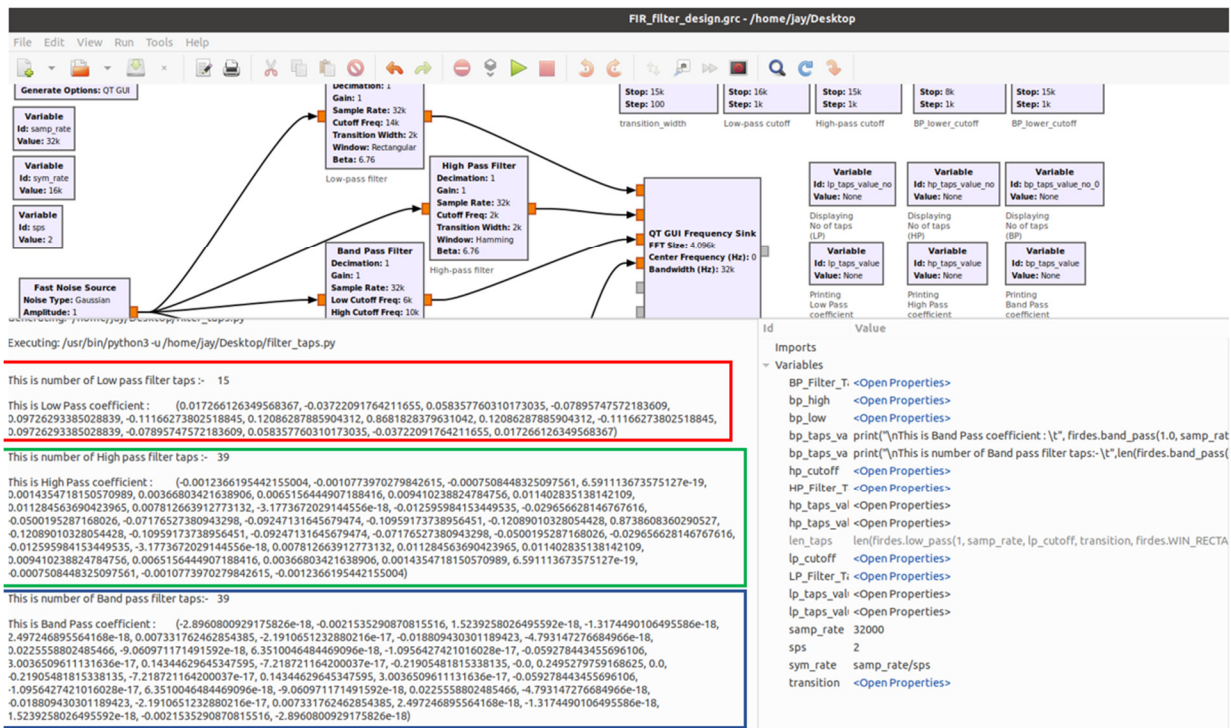
Fig.6. Filter taps (coefficients) and number of taps displayed in GNURadio command prompt

5. Now change the window type in the Filter design block configuration (see figure 5) to Hamming, Rectangular and Blackman to each filter and compare the frequency response of all four filters.

6. In your lab report you should comment on the following:
    - How does the number of taps affect the frequency response of your FIR filters?
    - What is the separation between the passband and stopband in your designs?
    - How far does the transition zone extend?

7. Comment on your observations.

Appendix A   FIR Digital Filters

Introduction

A digital filter is a basic building block in any Digital Signal Processing (DSP) system. The frequency response of the filter depends on the value of its coefficients, or taps. Many software programs can compute the values of the coefficients based on the desired frequency response. In our course, the students are using Labview programs to obtain the filter coefficients. These values are typically floating point numbers and they are represented with a fairly high degree of precision. However, when a digital filter is implemented in hardware, such as the TI TMS320C26 boards as it is a case in the second part of this laboratory experiment, the designer wants to represent the coefficients (and also the data) with the smallest number of bits that still gives acceptable resolution for the numbers. This is because representing a number with excess bits increases the size of the registers, buses, adders, multipliers and other hardware used to process that signal. The bigger sizes result in a chip with a larger die size, which translates into increased power consumption and a higher chip price. Thus, the bit precisions used to represent numbers are important in the performance of real-world signal processing designs.

Filtering with FIR systems

A Finite Impulse Response (FIR) digital filter is one whose impulse response is of finite duration. This can be stated mathematically as

$$h[n] = 0 \Rightarrow n < 0 \text{ or } n > N$$

where h[n] denotes the impulse response of the digital filter, n is the
discrete time index, and N is the length of the filter. A difference equation is the discrete time equivalent of a continuous time differential equation. The general difference equation for a FIR digital filter is

$$y[n] = \sum_{k=0}^{N} h[k] \cdot x[n-k]$$

where y(n) is the filter output at discrete time instance n, h[k] is the k-th feedforward tap, or filter coefficient, and x[n-k] is the filter input delayed by k samples. The $\sum_{k=0}^{N}$ denotes summation from $k = 0$ to $k = N$ where N is the number of feedforward taps in the FIR filter. The calculations of the output sample in terms of the input samples and the h[k] coefficients are represented in the figure below with the z^{-1} representing the delays (of one sampling period):
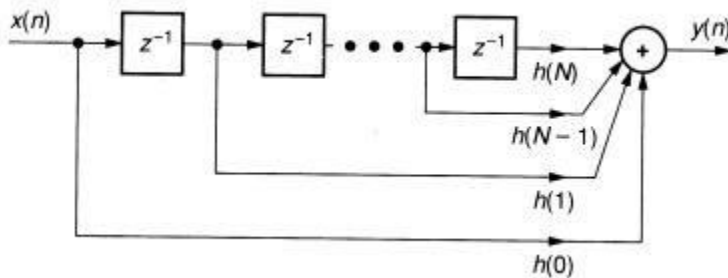

Figure 3: FIR structure

 Note that the FIR filter output depends only on the previous N inputs. This feature is why the impulse response for a FIR filter is finite.

When the input to a FIR filter is the unit impulse $o[n]$, the output of the filter is the impulse response h[n].

Advantages of FIR filters

FIR filters are simple to design and they are guaranteed to be bounded input-bounded output (BIBO) stable. By designing the filter taps to be sym-metrical about the center tap position, a FIR filter can be guaranteed to have linear phase. This is a desirable property for many applications such as music and video processing. FIR filters also have a low sensitivity to filter coefficient unitization errors. This is an important property to have when implementing a filter on a DSP processor or on an integrated circuit.

FIR Filter Design by Windowing

The most commonly used techniques for design of infinite impulse response (IIR) filters are based on transformations of continuous-time systems into discrete-time IIR systems. This is partly because continuous-time filter design was a highly advanced art before discrete-time filters were of interest and partly because of the difficulty of implementing a noniterative direct design method for IIR filters.

In contrast, FIR filters are almost entirely restricted to discrete-time implementations. Consequently the design techniques for FIR filters are based on directly approximating the desired frequency response of the discrete-time ssystem. Furthermore, most techniques for approximating the magnitude response of an FIR system assume a linear phase constraint, thereby avoiding the problem of

spectrum factorization that comp licates the direct design of IIR filters.

The simplest method of FIR filter design is called the window method. This method generally begins with an ideal desired frequency response that can be represented as

$$H_d(e_{j0}) = \sum_{n=-\infty} h_d[n] \, e_{j0} \tag{0}$$

where $h_d[n]$ is the corresponding impulse response sequence, which can be expressed in terms of $H_d(e^{j0})$ as

$$h_d[n] = \frac{1}{2 \cdot TX} \cdot \int_{-IT}^{TX} H_d(e^{j0}) \cdot e^{j0} \, d0) \tag{1}$$

Many idealized systems are defined by piecewise-constant or piecewise-functional frequency responses with discontinuities at the boundaries between bands. As a result, they have impulse responses that are noncausal and infinitely long. The most straightforward approach to obtaining a causal FIR approximation to such systems is to truncate the ideal response. Equation (1) can be thought of as a Fourier series

representation of the periodic frequency response $H_d(e^{j0})$, with the sequence $h_d[n]$ playing the role of the Fourier coefficients. Thus, the approximation of an ideal filter by truncation of the ideal impulse response is identical to the issue of the convergence of Fourier series, a subject that has received a great deal of study. A particularly important concept from this theory is the Gibbs phenomenon, which will be covered in the example with this Appendix.

The simplest way to obtain a causal FIR filter from $h_d[n]$ is to define a new system with impulse response h[n] given by

$$h[n] = \left\{ \begin{array}{l} h_d[n], .\, 0 \le n \le M \\ 0, .\, otherwise \end{array} \right. \tag{2}$$

More generally, we can represent h[n] as the product of the desired impulse response and a finite-duration "window" w[n]; i.e.,

$$h[n] = h_d[n] \cdot w[n] \tag{3}$$

where for simple truncation as in Eq. (2), the window is the rectangular window

$$w[n] = \left\{ \begin{array}{l} 1, .\, 0 \leq n \leq M \\ 0, .\, otherwise \end{array} \right. \tag{4}$$

It follows from the windowing theorem that

$$H(e^{j0}) = \frac{1}{2 \cdot TX} \cdot \int_{-TX}^{TX} H_d(e^{j8}) \cdot W(e^{j(0)-8})d0) \tag{5}$$

That is, $H(e^{j0})$ is the periodic convolution of the desired ideal frequency response with the Fourier transform of the window. Thus, the frequency response $H(e^{j0})$ will be a "smeared" version of the desired response $H_d(e^{j0})$. Figure 2(a) depicts typical functions $H_d(e^{j0})$ and $W(e^{j0})$ as required in Eq.(5).

If w[n] = 1 for all n (i.e., we do not truncate at all), $W(e^{j0})$ is a periodic impulse train with period $2 \cdot TX$, and therefore $H_d(e^{j0}) = H(e^{j0})$.

This interpretation suggests that if w[n] is chosen so that $W(e^{j0})$ is concentrated in a narrow band of frequencies around $0) = 0$, then $H(e^{j0})$ will "look like $H_d(e^{j0})$ except where $H_d(e^{j0})$ changes very abruptly. Thus, the choice of window is governed by the desire to have w[n] as short as possible in duration so as to minimize computation in the implementation of the filter while having $W(e^{j0})$ approximate an impulse; that is, we want $W(e^{j0})$ to be highly concentrated in frequency so that the convolution of Eq. (5) faithfully reproduces the desired frequency response. These are conflicting requirements, as can be seen in the case of the rectangular window of Eq. (4), where

$$W(e^{j0}) = \sum_{n=0}^{M} h_d e^{-j0) \cdot n} = \frac{1 - e^{-j0) \cdot (M+1)}}{1 - e^{-j0)}} = e^{-j0) \cdot M / 2} \cdot \frac{\sin(0) \cdot (M+1)/2)}{\sin(0)/2)} \tag{6}$$

The magnitude of $W(e^{j0})$ is plotted in Fig. 8 for the case M = 7. Note that $W(e^{j0})$ for the rectangular window has generalized linear phase. As M increases, the width of the "main lobe" decreases. The main lobe is usually defined as the region between the first zero crossings on either side of the origin. For the rectangular window, the mainlobe width is $\Delta 0)_m = \frac{4 \cdot TX}{M + 1}$. However, for the rectangular window, the sidelobes are significantly large and in fact, as M increases, the peak amplitudes of the mainlobe and the sidelobes grow in a manner such that the area under each lobe is a constant while the width of each lobe decreases with M. Consequently, as $W(e^{j(0)-8})$ "slides by" a discontinuity of $H_d(e^{j0})$ with increasing $0)$, the integral of $W(e^{j(0)-8}) \cdot H_d(e^{j0})$ will oscillate as each sidelobe of $W(e^{j(0)-8})$ moves past the discontinuity. This result is depicted in Fig.7(b). Since the area under each lobe remains constant with increasing M, the oscillations occur more rapidly but do not decrease in amplitude as M increases.

In the theory of Fourier series, it is well known that this nonuniform convergence, the Gibbs phenomenon, can be moderated through the use of a less abrupt truncation of the Fourier series by tapering the window smoothly to zero.

Figure 7 (a) Convolution process implied by truncation of the ideal impulse response. (b) Typical approximation resulting from windowing the ideal impulse response.
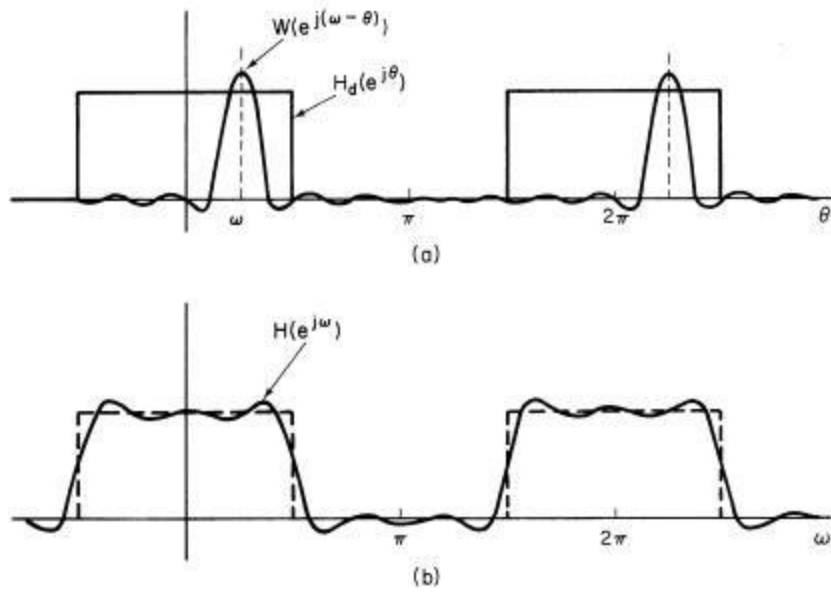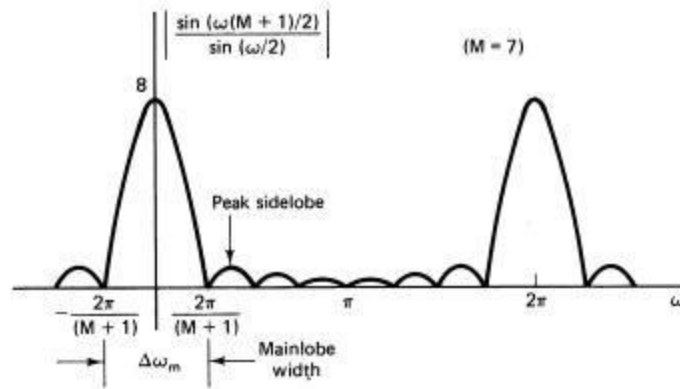
Figure 8 Magnitude of the Fourier transform of a rectangular window (M = 7).

Design Example

The desired frequency response is defined as

$$H_{lp}(e^{j\omega}) = \begin{cases} e^{-j\omega \cdot M/2}, & |\omega| \le |\omega_c| \\ 0, & |\omega_c| < |\omega| \le |\pi| \end{cases} \tag{86}$$

where the generalized linear phase factor has been incorporated into the definition of the ideal lowpass filter. The corresponding ideal impulse response is

$$h_{lp}[n] = \frac{1}{2 \cdot \pi} \int_{-\pi}^{\pi} e^{-j\omega \cdot M/2} \cdot e^{j\omega \cdot n} d\omega = \frac{\sin(\omega_c \cdot (n - M/2))}{\pi \cdot (n - M/2)} \tag{87}$$

- 8 -

for $-\infty < n < +\infty$. It is easily shown that $h_{lp}[M - n] = h_{lp}[n]$, so
if we use a symmetric window in the equation

$$h[n] = \frac{\sin(\omega_c \cdot (n - M/2))}{\pi \cdot (n - M/2)} \cdot w[n] \tag{88}$$

then a linear phase system will result.

The upper part of Fig. 31 depicts the character of the amplitude response that would result for the typical windows used in your design. Figure 31 displays the important properties of window method approximations to desired frequency responses that have step discontinuities. It applies accurately when ω_c is not close to zero or to π and when the width of the mainlobe is smaller than $2 \cdot \omega_c$. At the bottom of the figure is a typical Fourier transform for a symmetric window. This function should be visualized in different positions as an aid in understanding the shape of the approximation $H(e^{j\omega})$ in the vicinity of ω_c .
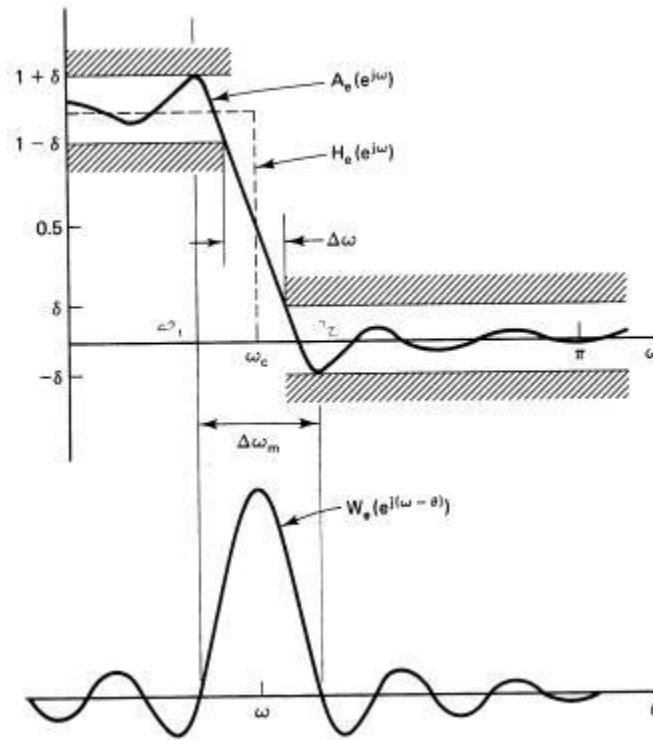

Figure 31 Illustration of type of approximation obtained at a discontinuity of the ideal frequency response.

Appendix B   FIR Digital Filters design using Matlab / Python

Using MATLAB :

After designing a FIR filter with N=5 taps in GNURadio with $f_s = 10\ kHz$ and cut off
frequency of your choice, you will verify your design in Matlab using the following steps:
- If your coefficients were [0.15 0.19 0.20 0.19 0.15], define in Matlab the
  impulse response vector h as:

```
1.  h=[0.15 0.19 0.20 0.19 0.15];and get the frequency response of your filter as
2.  hf=fft(h);
3.  plot(abs(hf));
```

- Now try to do zero padding on your impulse response as to get

```
1.  h1=[h zeros(1,100)];
2.  hf1=fft(h1);
3.  plot(abs(hf1));
4.  plot(20*log10(abs(hf1)));
```

- Comment on your observations in these two steps and compare the last results
  to the frequency response of the filter displayed in GNURadio.
- In the Matlab code provided, describe what each line does and how it affects
  the graphs.
- Design your own low-pass FIR filter with impulse response **myh** in Matlab by
  using the following sample statements:

```
1.  myt=-4:4;
2.  myh=sinc(0.2*t);
```

- Filter sinusoidal signals $x[n] = \sin(w_O n)$ with different w_O using a filter
  having the impulse response $h[n]$ from your design using Matlab command
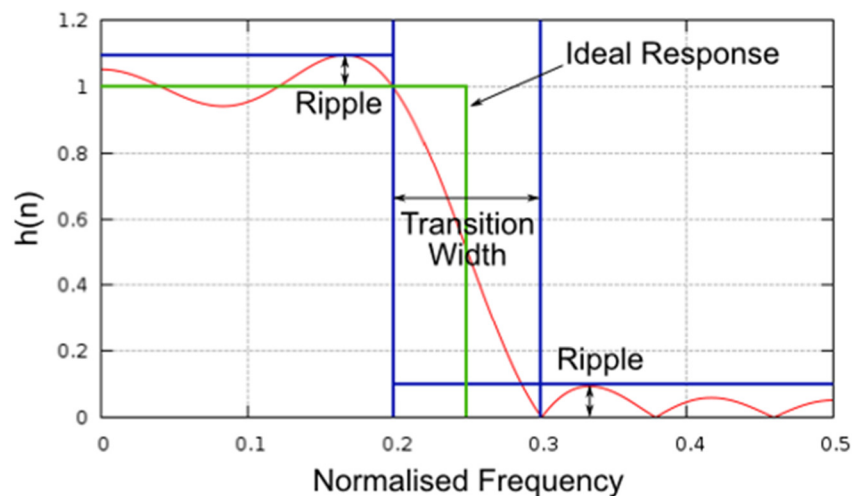
  filter:

```
1.  t=1:200;
2.  x=sin(0.25*t);
3.  y=filter(h,1,x);
4.  plot(y);
5.  plot(abs(fft(y)));
```

Using Python

For most filters we will see (known as FIR, or Finite Impulse Response, type filters), we can represent the filter itself with a single array of floats. For filters symmetrical in the frequency domain, these floats will be real (versus complex), and there tends to be an odd number of them. We call this array of floats "**filter taps**". We often use **h** as the symbol for filter taps. Here is an example of a set of filter taps, which define one filter:

```
h =  [ 9.92977939e-04  1.08410297e-03  8.51595307e-04  1.64604862e-04
 -1.01714338e-03 -2.46268845e-03 -3.58236429e-03 -3.55412543e-03
 -1.68583512e-03  2.10562324e-03  6.93100252e-03  1.09302641e-02
  1.17766532e-02  7.60955496e-03 -1.90555639e-03 -1.48306750e-02
 -2.69313236e-02 -3.25659606e-02 -2.63400086e-02 -5.04184562e-03
  3.08099470e-02  7.64264738e-02  1.23536693e-01  1.62377258e-01
  1.84320776e-01  1.84320776e-01  1.62377258e-01  1.23536693e-01
  7.64264738e-02  3.08099470e-02 -5.04184562e-03 -2.63400086e-02
 -3.25659606e-02 -2.69313236e-02 -1.48306750e-02 -1.90555639e-03
  7.60955496e-03  1.17766532e-02  1.09302641e-02  6.93100252e-03
  2.10562324e-03 -1.68583512e-03 -3.55412543e-03 -3.58236429e-03
 -2.46268845e-03 -1.01714338e-03  1.64604862e-04  8.51595307e-04
  1.08410297e-03  9.92977939e-04]
```

Let's visualize transition width. In the diagram below, the **green** line represents the ideal response for transitioning between a passband and stopband, which essentially has a transition width of zero. The **Red** line demonstrates the result of a realistic filter, which has some ripple and a certain transition width.


You might be wondering why we would not just set the transition width as small as possible. The reason is mainly that a smaller transition width results in more taps, and more taps means more computations–we will see why shortly. A 50-tap filter can run all day long using 1% of the CPU on a Raspberry Pi. Meanwhile, a 50,000-tap filter will cause your CPU to explode! Typically, we use a filter designer tool, then see how many taps it

- 11

outputs, and if it's way too many (e.g., more than 100) we increase the transition width. It all depends on the application and hardware running the filter, of course.

**Let's design a Low Pass filter using 51 taps and 3 kHz cut off.**

```python
1.  import numpy as np
2.  from scipy import signal
3.  import matplotlib.pyplot as plt
4.
5.  num_taps = 51 # it helps to use an odd number of taps
6.  cut_off = 3000 # Hz
7.  sample_rate = 32000 # Hz
8.
9.  # create our low pass filter
10. h = signal.firwin(num_taps, cut_off, nyq=sample_rate/2)
11.
12. # plot the impulse response
13. plt.figure(figsize=(16,6))
14. plt.plot(h, 'b.-')
15. plt.title("Time Domain")
16. plt.grid()
17. plt.show()
```

Simply plotting this array of floats gives us the filter's impulse response:
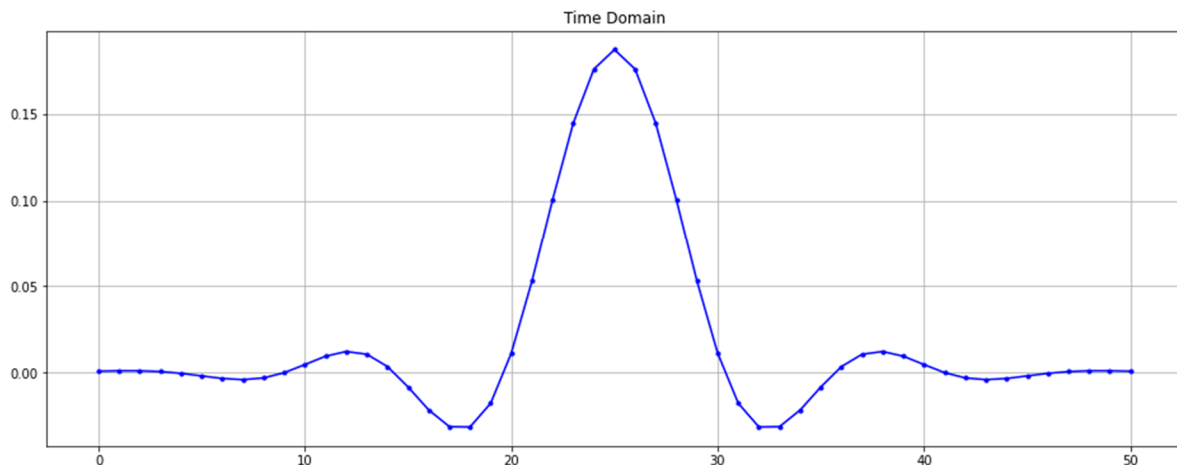
And here is the code that was used to produce the frequency response, shown earlier. It's a little more complicated because we have to create the x-axis array of frequencies.

```python
1.  plt.figure(figsize=(16,6))
2.  # plot the frequency response
3.  H = np.abs(np.fft.fft(h, 1024)) # take the 1024-point FFT and magnitude
4.  H = np.fft.fftshift(H) # make 0 Hz in the center
5.  w = np.linspace(-sample_rate/2, sample_rate/2, len(H)) # x axis
6.  plt.plot(w, H, 'b.-')
7.  plt.title("frequency response")
8.  plt.grid()
9.  plt.show()
```

- 12

frequency response
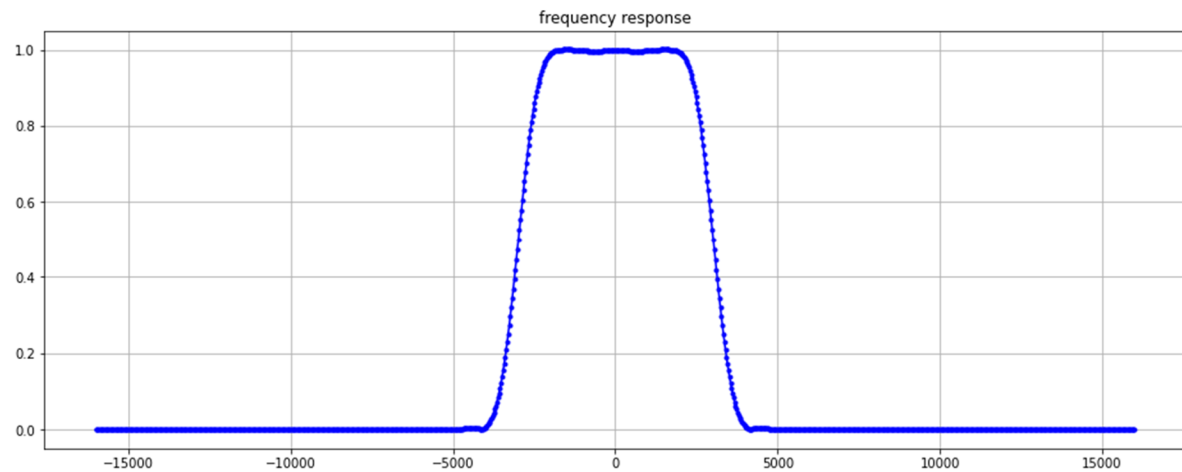
Now let's generate a binary signal and apply filter on the given signal x.

```
1.  x =  np.array([-1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,
2.          0.,   0.,   0.,  -1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,
3.          0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,
4.          0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,
5.          0.,   0.,   0.,   0.,  -1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,
6.          0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,
7.          0.,   0.])
8.  num_symbols = 10
9.  sps = 8
10. # Filter our signal, in order to apply the pulse shaping
11. x_shaped = np.convolve(x, h)
12. plt.figure(2)
13. plt.figure(figsize=(16,6))
14. plt.plot(x_shaped, 'b.-')
15. for i in range(num_symbols):
16.     plt.plot([i*sps+num_taps//2+1,i*sps+num_taps//2+1], [min(x_shaped), max(x_shap
    ed)])
17. plt.grid(True)
18. plt.title("filter signal")
19. plt.show()
```


filter signal