Abu Dhabi University

CEN 464 - Digital Signal Processing

Project II Report
Digital Signal Processing of Audio Signal

Authors:
Muhammad Obaidullah 1030313

Supervisor:
Dr. Mohammed Assad Ghazal

Section 1

June 8, 2014

Contents

Abstract

In this project we designed a digital Finite Impulse Response Filter using AtMega 328. The project's aim was to get familiarized with digital signal processing by using the C language programming. We began by first designing the filter using either MATLAB or online filter design tool.

1 Introduction

Human ear is a very astonishing and complex organ. The brain even makes the system more complex by combining information from two years in a perplexing neural network. Although it might seem very natural and hidden to us but our brain does very complex signal processing everyday, every second, every moment on the sounds heard by both the ears.

For example, On a noisy road, you can easily hear the friend sitting beside you because your brain categorizes the road noise as background and amplifies the frequencies which are coming from your friend's voice. Thus, brain is capable of selective filtering by removing he background noise. This kind of filtering is based on band pass filter which signal processing engineers use nowadays. Several other examples might be given to prove the fact that human brain is an extremely complex and advanced audio signal processor.

Some decades back, before the introduction of transistors, analog devices were quite dominant in the market and all the electronics were using analog filters. But now we live in a digital age where everything is stored, processed and provided in digital form. This transistor driven transition from analog to digital domain has lead scientist and engineers to design the same filters which they used to design using analog devices to be designed now by using digital devices like micro-processors, FPGAs etc.

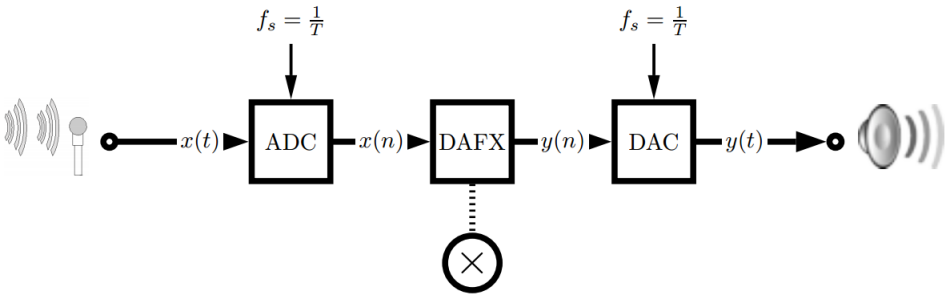

Figure 1: The figure shows how a typical digital signal processing is done. The input signal from microphone is first converted from analog to digital and then fed into the micro-processor which multiplies the input by the filter coefficients and then the DAC converts the digital signal back to analog for the speaker to output. [3]

2 List of equipment used

- A Computer.

- MATLAB.
- MATLAB Filter Design Toolbox.
- AtMega 328.
- Arduino Board.
- USB B Cable.
- Function Generator.
- Oscilloscope.
- Jumper Wires.
- Breadboard.
- Microphone.
- Speaker.
- Internet Connection.
- TFilter Website.

3 Project Circuit

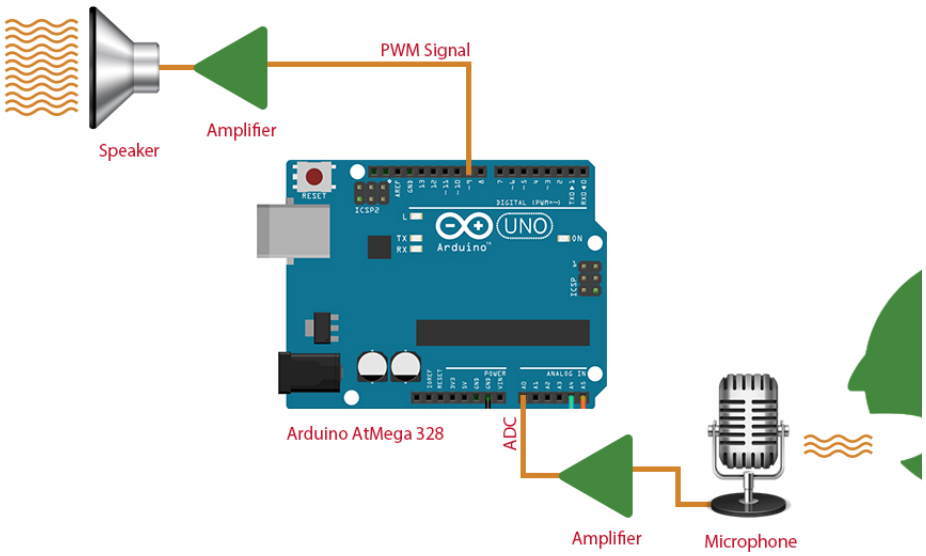

Figure 2: The figure shows how the microphone can be connected to the micro-controller to read the audio signals and provide the output to the speaker using PWM signal.

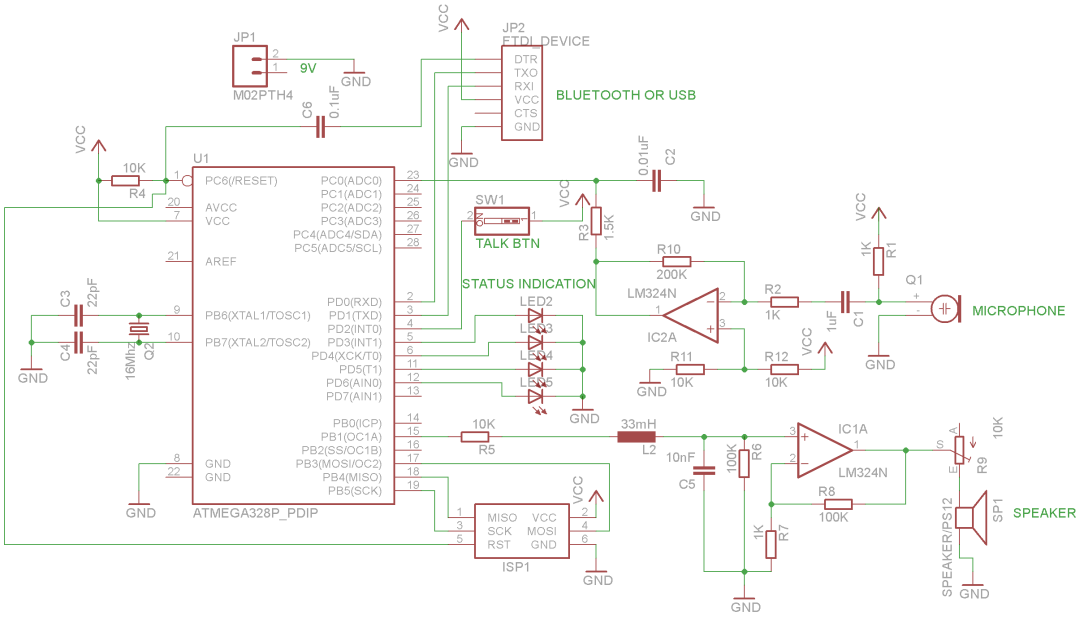

Figure 3: The figure shows the circuit diagram with the amplifier circuit at both, the input microphone and the output speaker.

4 Proposed PCB

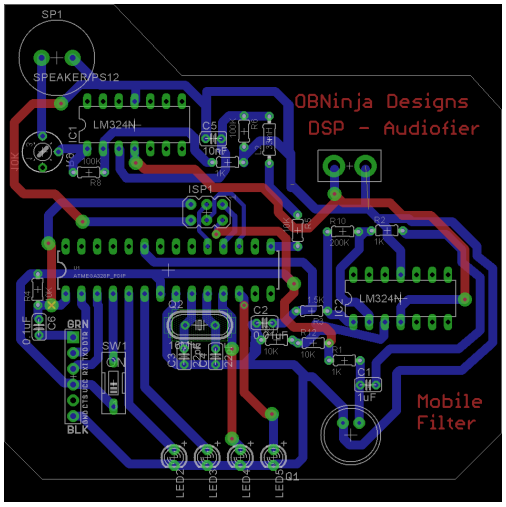

Figure 4: The X-Ray vision of the printed circuit board of the audio processor.

5 Low Pass Filter

A low pass filter allows lower frequency components of the signal to pass while attenuating the higher frequencies. In audio signals, the low frequency sounds, for example a roar will pass unaffected and the higher frequency components of the sound signal for example the cry or shout will pass be completely attenuated.

5.1 Filter Design

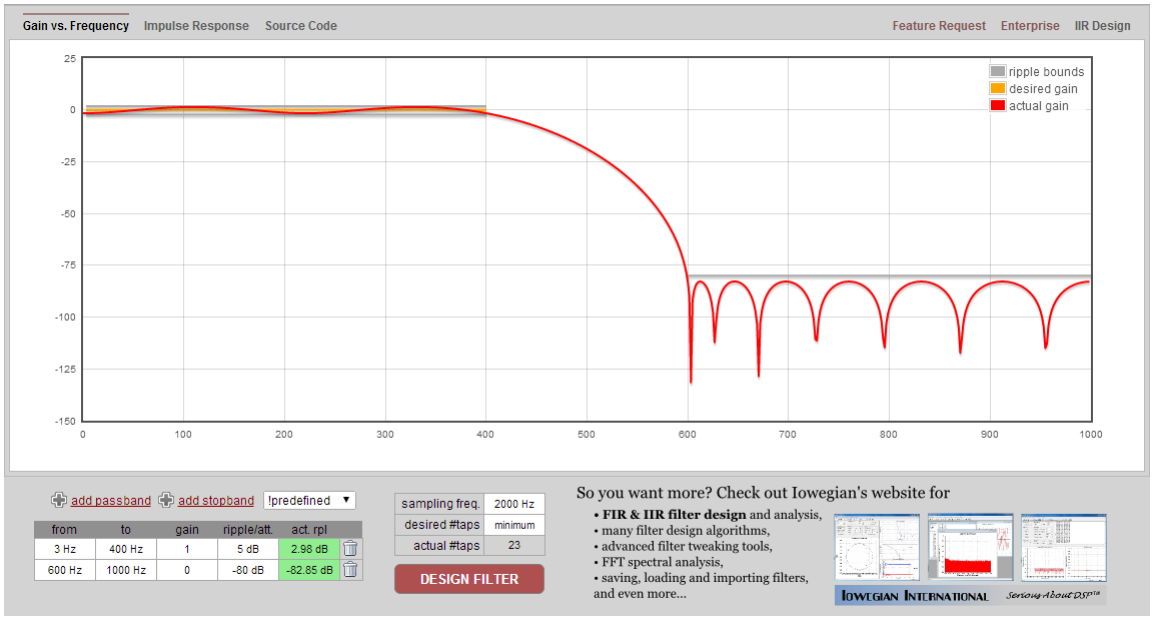

Figure 5: The figure shows the magnitude response of the 39 taps low pass filter to filter the frequencies from 500Hz to 1000Hz. The passband ripple was set to 5dB and stopband attenuation was set to -80dB.

5.2 Results

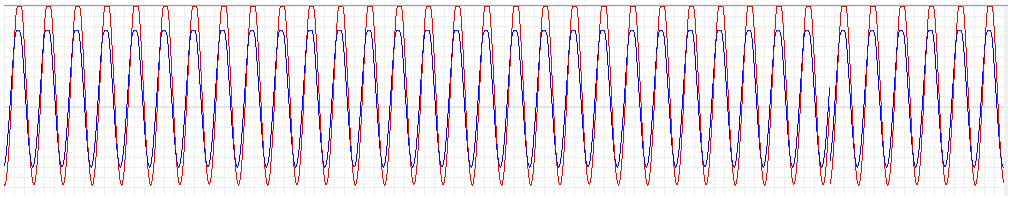

Figure 6: The figure shows what happens when 20Hz is passed through the input. It is allowed to pass since 40Hz is in the pass band. The red signal is the input and the blue is output.

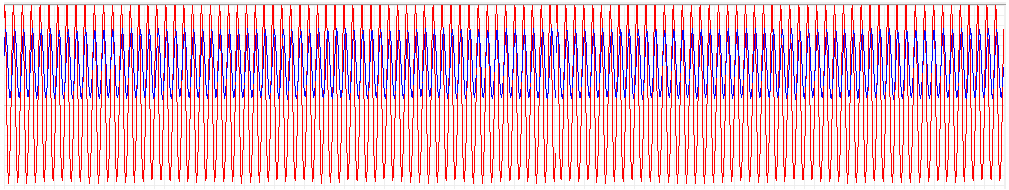

Figure 7: The figure shows what happens when 650Hz is passed through the input. The red signal is the input and the blue is output.

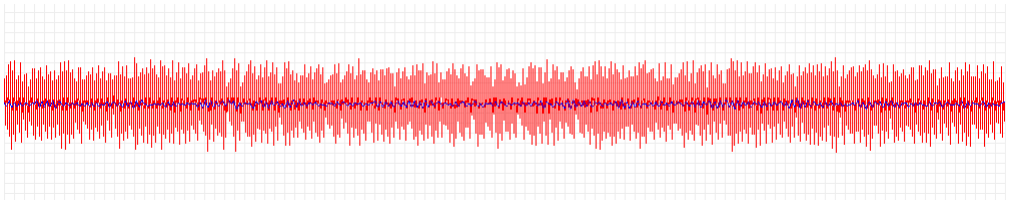

Figure 8: The figure shows what happens when 870Hz is passed through the input. The red signal is the input and the blue is output.

6 High Pass Filter

A high pass filter allows higher frequency components of the signal to pass while attenuating the lower frequencies. In audio signals, the low frequency sounds, for example a roar will be completely attenuated and the higher frequency components of the sound signal for example the cry or shout will pass unaffected.

6.1 Filter Design

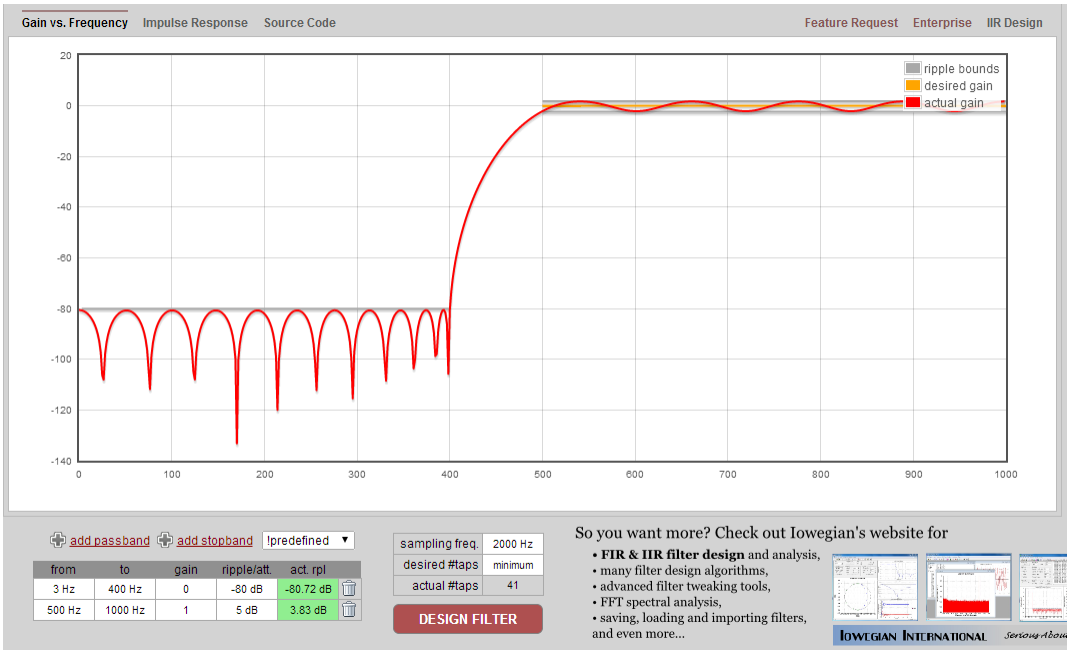

Figure 9: The figure shows the magnitude response of the 41 taps high pass filter to filter the frequencies from 3Hz to 400Hz. The passband ripple was set to 5dB and stopband attenuation was set to -80dB.

6.2 Results

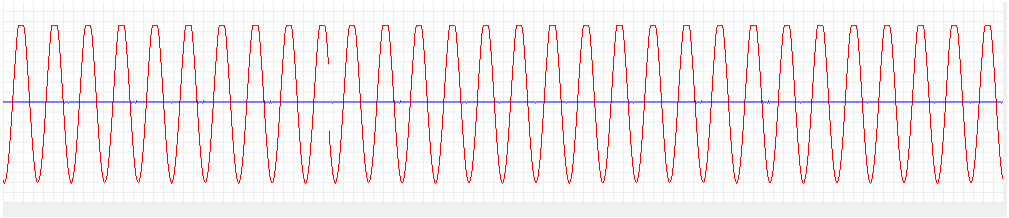

Figure 10: The figure shows what happens when 20Hz is passed through the input. It is allowed to pass since 40Hz is in the pass band. The red signal is the input and the blue is output.

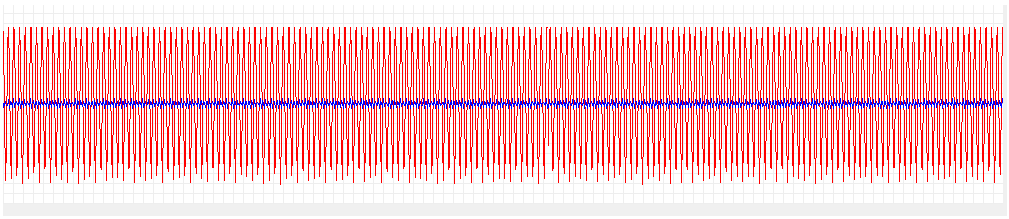

Figure 11: The figure shows what happens when 550Hz is passed through the input. The red signal is the input and the blue is output.

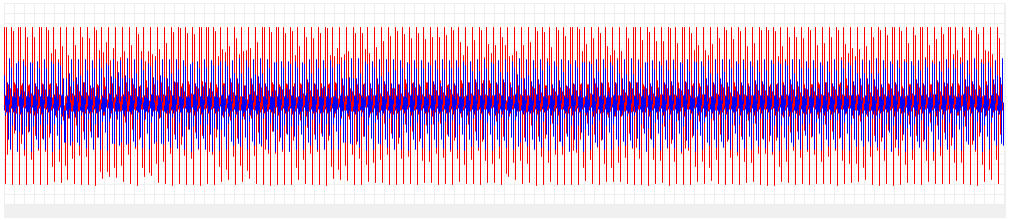

Figure 12: The figure shows what happens when 950Hz is passed through the input. The red signal is the input and the blue is output.

7 Band Pass Filter

A band pass filter allows certain selected frequency range (bandwidth) to pass while attenuating the other frequency components. This filter is extremely useful for removing noise and tuning into a selected frequency band from multiplexed channel signals. This type of filter is often seen to be used right after the antenna. This removes the Gaussian white noise components which have been added frequencies outside the band of operation. Also it selects the band of operation and does not allow other frequency channels to pass in to the output.

7.1 Filter Design

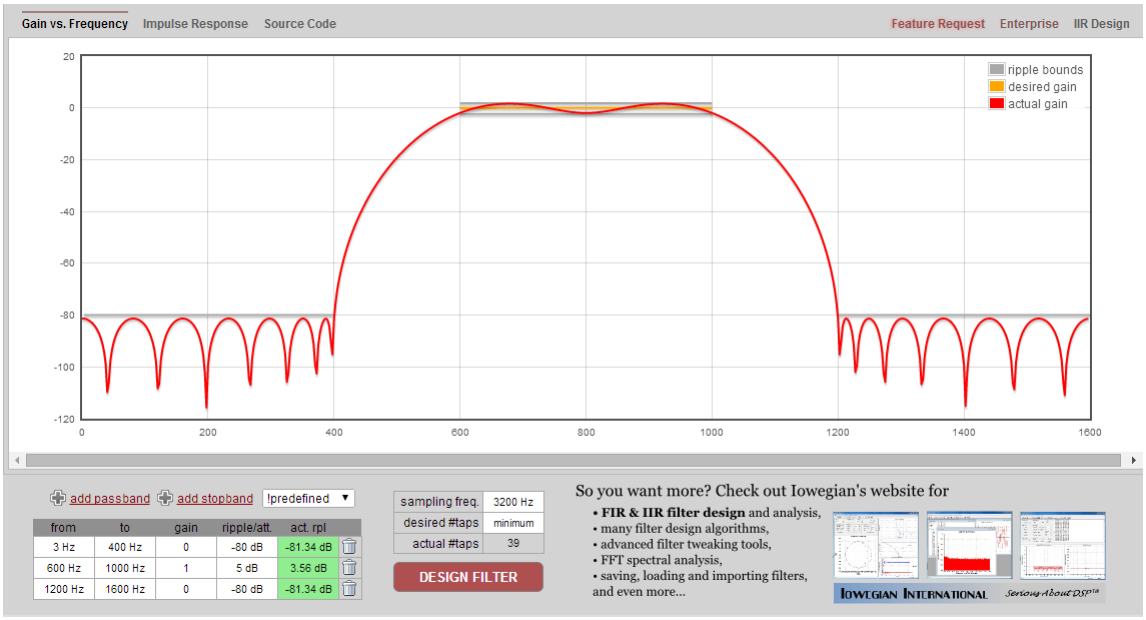

Figure 13: The figure shows the magnitude response of the 39 taps band pass filter to pass the frequencies from 600Hz to 1000Hz. The passband ripple was set to 5dB and stopband attenuation was set to -80dB.

7.2 Results

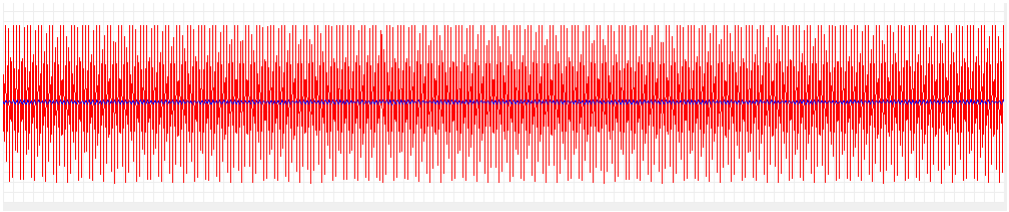

Figure 14: The figure shows what happens when 200Hz is passed through the input. It is allowed to pass since 40Hz is in the pass band. The red signal is the input and the blue is output.

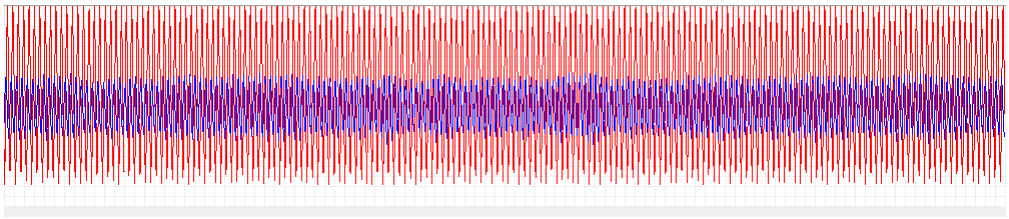

Figure 15: The figure shows what happens when 650Hz is passed through the input. The red signal is the input and the blue is output.

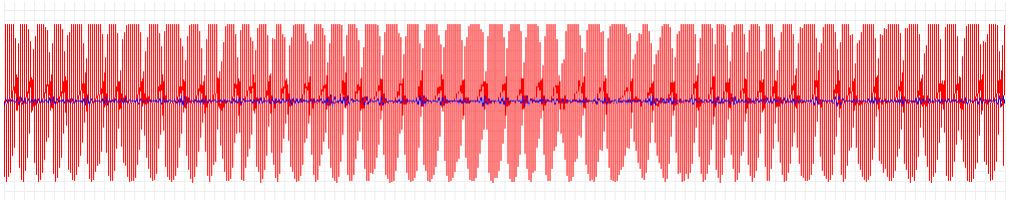

Figure 16: The figure shows what happens when 1600Hz is passed through the input. The red signal is the input and the blue is output.

8 Band Reject Filter

A band reject filter rejects a certain band of frequencies and allows all the other frequency components of signal pass through. This kind of filters are used when the frequency of unwanted signal is precisely known and is to be attenuated for producing excellent output results. For example some very advanced voice amplifiers have band reject filters to reject the signal frequencies which are feed back into the microphone because the microphone was too near the speaker. It is the exact inverse of the Band-pass filter.

8.1 Filter Design

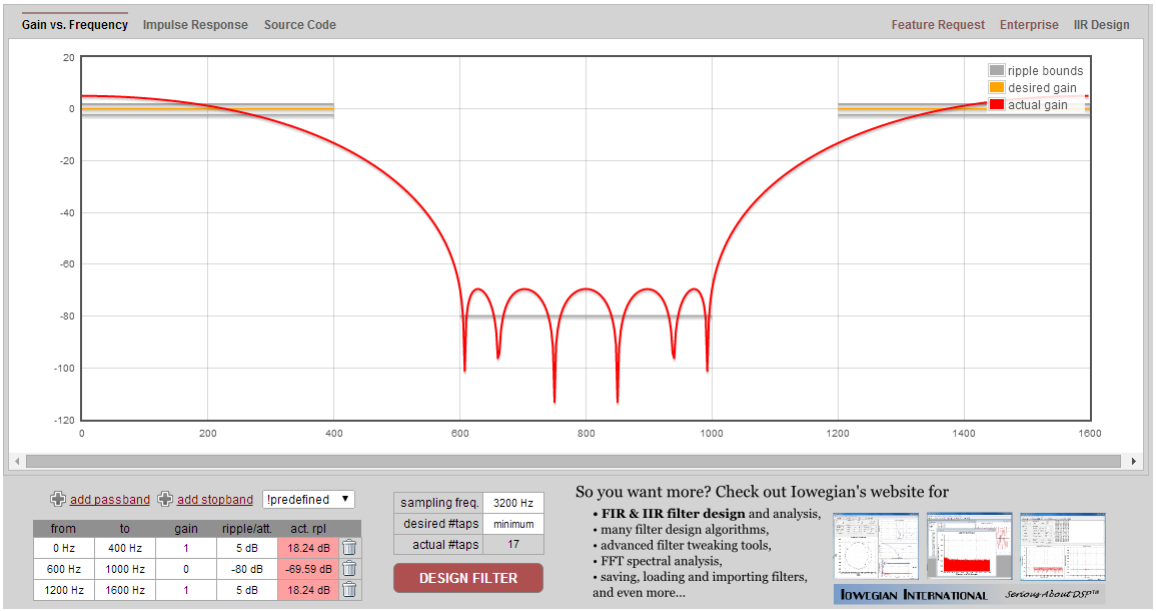

Figure 17: The figure shows the magnitude response of the 17 taps band reject filter to reject the frequencies from 600Hz to 1000Hz. The passband ripple was set to 5dB and stopband attenuation was set to -80dB.

8.2 Results

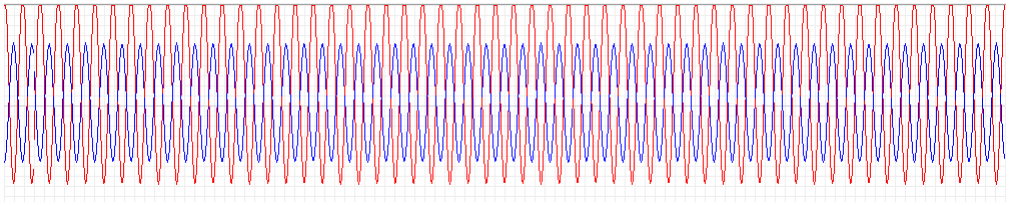

Figure 18: The figure shows what happens when 40Hz is passed through the input. It is allowed to pass since 40Hz is in the pass band. The red signal is the input and the blue is output.

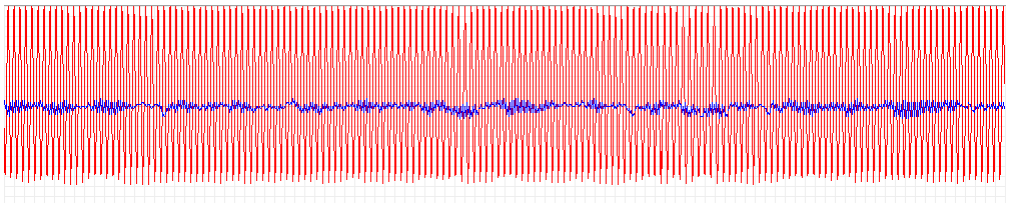

Figure 19: The figure shows what happens when 850Hz is passed through the input. The red signal is the input and the blue is output.

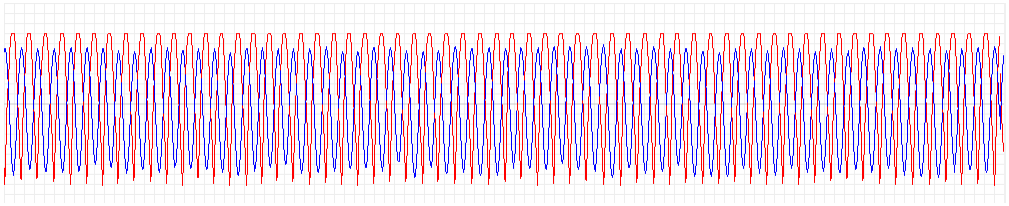

Figure 20: The figure shows what happens when 850Hz is passed through the input. The red signal is the input and the blue is output.

9 Problems & Discussions

9.1 Maximum Sampling Frequency

The ADC clock is derived from the system clock which in the case of AtMega 328 using the Arduino bootloader is about 16 Mhz. The system clock is prescaled by 128 prescaler. The following equation finds out the sampling frequency of the ADC.

$$f_{ADC} = \frac{f_{system}}{128} = \frac{16 \times 10^6}{128} = 125,000 = 125KHz \tag{1}$$

According to NyquistShannon sampling theorem, sampling frequency should be twice the maximum frequency of the input. Theoretically, the micro-controller should be then able to process signals of about $\frac{125Khz}{2} = 62.5Khz$. In our project the input is the human voice which has 20 Hz to 20,000 Hz bandwidth. In order to digitize the human voice, we need a sampler which takes samples twice this bandwidth.

$$f_{nyquist} = 2 \times 20Khz = 40KHz < 125Khz \tag{2}$$

The AtMega 328 bootloaded with Arduino takes samples at 125 Khz which is way above the Nyquist-Shannon criterion. The samples from the ADC pin 0 are taken 125,000 times in a second and stored in the register for the programmer to access. So the maximum rate at which we can sample using this AtMega 328 is 125Khz.

9.2 ADC Value Fetching Frequency

The AtMega 328 we are using is clocked at 16 Mhz and at maximum speed, it processes 16 Million lines of code in a second provided that the line of code does not contain arithmetic operations. In case of performing arithmetic operations, it takes several clock cycles to come up with a result for that particular line of code. All of these complications reduce the rate at which the new input sample is taken from the ADC register.

Let's assume one case and perform some calculations. The low pass filter code we wrote is of 64 lines, so the maximum ADC value fetching frequency will be:

$$f_{ADCfetch} = \frac{16 \times 10^6}{64} = 250,000 = 250KHz \tag{3}$$

Therefore, at maximum rate, without any arithmetic calculations, AtMega 328 is capable of fetching input 250,000 times in a second which is much higher than the rate ADC is sampled. So the actual ADC sampling rate will in fact remain 125Khz.

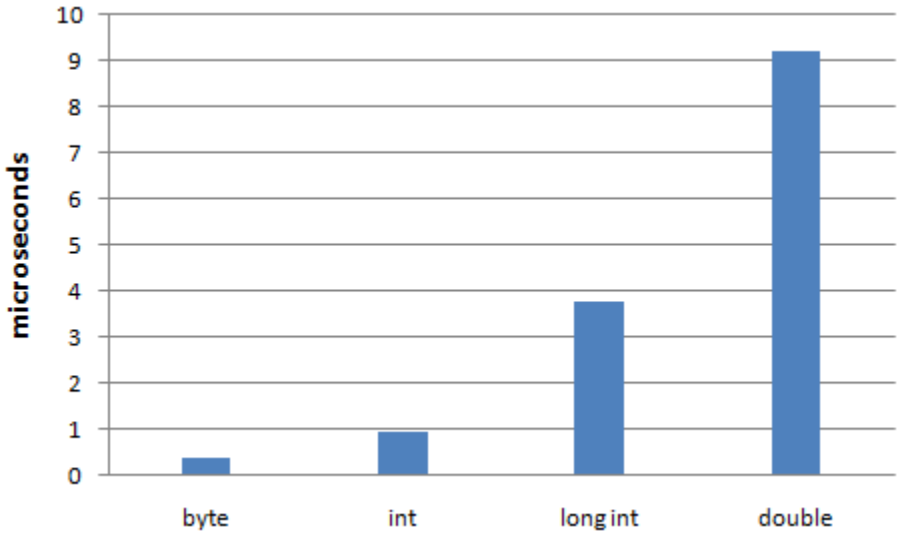

Figure 21: The figure shows how many microseconds an AtMega 328 takes to arithmetically process different data types. The double of size 4 arithmetic operation takes about 9.2 μs, a long integer takes about 3.8 μs, a integer operation about 0.9 μs and a byte less than 0.4 μs. [1]

In our arduino code, we are performing many double operations which is slowing down the processor and the sample fetching rate from ADC. Lets take the example of Low pass filter with 39 taps and calculate how much time does it take to calculate the output and take the next sample.

$$t_{output} = 9.2 \times 10^{-6} \times 39 = 3.588 \times 10^{-4} s \approx 0.36ms \tag{4}$$

If the arduino is taking 0.36 ms to calculate one output, combined this with the time it takes to process all the lines of code, we can come up with the total time it take for the arduino to produce the output and the maximum processing frequency.

$$t_{total} = 0.36 \times 10^{-3} + \frac{1}{125 \times 10^3} = 3.68 \times 10^{-4} \approx 0.37ms \tag{5}$$

$$f_{ADCprocessing} = \frac{1}{0.37 \times 10^{-3}} = 2702.7Hz \approx 2.7Khz \tag{6}$$

If our device is capable of sampling at rate of 2.7 Khz, then the input frequency should be half of this in order to satisfy nyquist rate.

$$f_{processable} = \frac{2.7 \times 10^3}{2} = 1350Hz \approx 1.3KHz \tag{7}$$

Thus, arduino can only process signals from 0-1.3 Khz frequencies. For higher frequencies, a high speed DSP chip is required such as FPGA.

9.3 Proof:

To support the mathematical findings of the digital signal processing speed of the AtMega, I designed a Low pass filter with pass-band from 3Hz - 1.5KHz and stop-band from 2KHz to 3.3Khz. After this I input range of frequencies to the ADC pin. Below are my results.

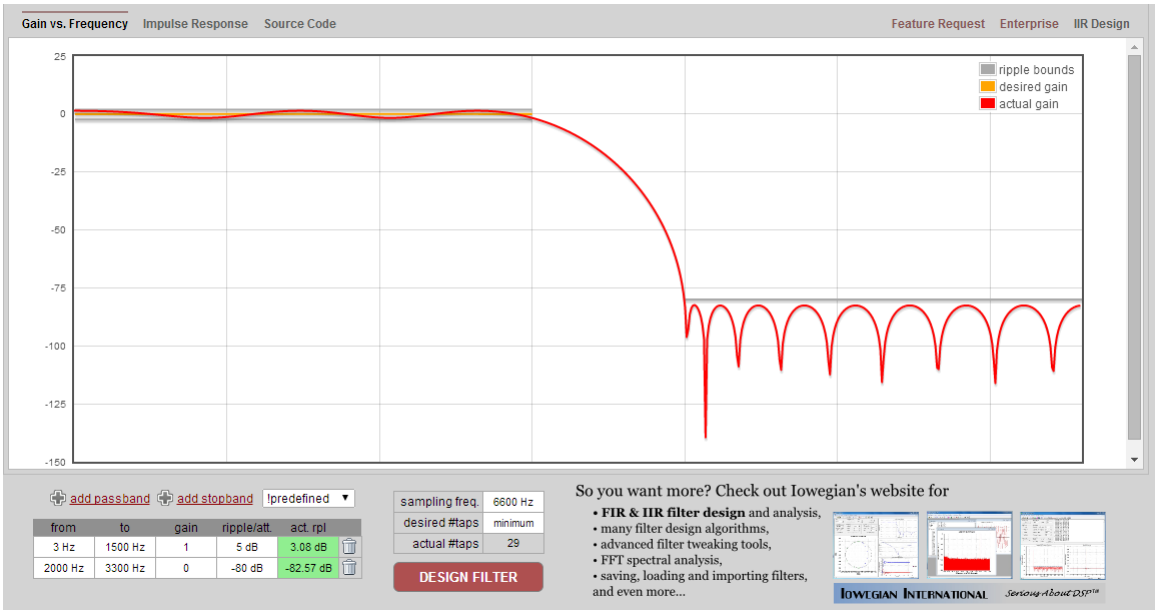

Figure 22: The figure shows a 29 taps low pass filter design with -80 dB attenuation in the stop band. As we can see that the filter is designed to attenuate the signals from 2KHz to 3.3Khz.

9.3.1 20Hz:

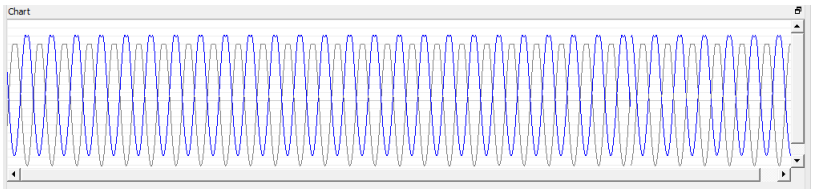

Figure 23: The figure shows what happens when a low frequency 20Hz is passed to the ADC of AtMega. The output is delayed a bit but it is constant throughout the time. It is also not attenuated as expected in the filter design.

9.3.2 at 5KHz:

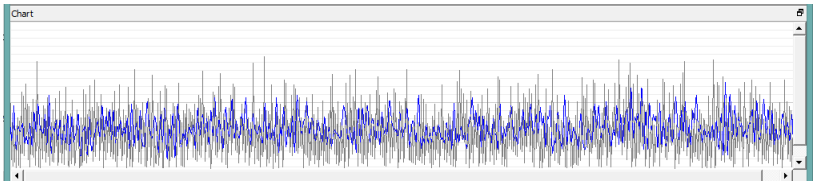

Figure 24: The figure shows what happens when a very high frequency, 5KHz, is passed to the ADC of AtMega. Forget about the output, input even isn't correctly captured because of slow capture rate. Since it is a Low pass filter and 5KHz is very high frequency, it is supposed to attenuate this frequency. But AtMega is allowing it to pass through.

10 Conclusion

All in all, this project successfully implemented the low pass, high pass, band reject, and band pass filter on an AtMega microprocessor clocked at 16 Mhz and boot loaded with Arduino boot loader. Additionally, we found out experimentally that is the input frequency increases higher than about 1200 Hz, the AtMega becomes kind of a all-pass filter and allows all frequencies to pass.

Moreover, after the confusing discovery of unpredictable response of AtMega to high frequencies, a theoretical and mathematical explanation was proposed in the project. Theoretically, we found that the microprocessor we are currently using can process signals of upto 1.3 Khz.

References

[1] Arduino Blog, *Speed of floating point operations – test results*, `http://forum.arduino.cc/index.php/topic,40901.0.html`, 07 April 2008 [1 June 2014].

[2] The Scientist and Engineer's Guide to Digital Signal Processing. *Audio Processing*, `http://www.analog.com/static/imported-files/tech_docs/dsp_book_Ch22.pdf`, [1 June 2014].

[3] Cardiff University. *Basic Digital Audio Signal Processing*, `http://www.cs.cf.ac.uk/Dave/CM0268/PDF/07_CM0268_DSP.pdf`, [2 June 2014].