ECE212A: Project

Kenny Chan
UID: 004769092
kennych418@ucla.edu

December 11, 2020

Abstract

The goal of this project is to design an optimum filter that meets a set of specifications while achieving the lowest possible hardware cost. My approach was to design a MATLAB script that generates the optimum filter for any set of lowpass filter specifications through the Parks McClellan optimization technique.

The MATLAB script first generates an estimated FIR filter order that can meet the specifications with a lot of margin. It will then generate the respective filter, estimate the lowest number of bits, verify ripples in the pass and stop band, and verify the SNR using a -6dB full-scale sinusoidal signal. If the filter can meet all of the speicifcations, the script will decrement the filter order, repeat the process, and log all results until it can't generate a filter that meets the specifications. At the end, the script will return the filter with the lowest hardware cost.

My architecture takes advantage of the symmetry in the filter coefficients to reduce the number of multipliers by a factor of 2. This helps reduce the hardware cost and quantization noise of my system.

For the filter specifications provided by the project, my optimum filter order is 104 with 15+1 bits. The system can meet the ripple specifications and has an SNR of 74.5. The total hardware cost is 41865.

1 Filter Design Algorithm

1.1 Motivation

A filter's cost is driven by two properties: its length and number of bits. Reducing the filter length cuts down on the amount of registers, multipliers, and adders need, whereas reducing the number of bits increases the size of each hardware module. However, a smaller filter length also reduces the ripple margin in the pass and stop band. This implies that the number of bits would have to increase to reduce the magnitude of the roundoff noise that could cause the system to violate the ripple specification. This tradeoff motivated me to design a script that can scan through a wide range of filter lengths, calculate the number of bits needed to stay within specifications, and choose the set with minimal hardware cost. Furthermore, this provides the flexibility to generate a minimal cost filter on any set of lowpass filter specifications.

Info: After plotting the results, I learned that the minimum number of bits to maintain an SNR just above the specification was constant for the range of filter lengths I scanned, so the cost was only dependant on the filter length. Nevertheless, it was pretty helpful to verify that the filter I received was the lowest cost I could achieve. Graphs for number of bits, SNR, and cost relative to filter lengths shown in Appendix Figure 1.

1.2 Algorithm

The algorithm starts by calculating a Parks McClellan estimate of the filter length from the given passband edge, stopband edge, ripple, and sampling frequency specifications. I also add a constant number with the estimate to increase the range of filter lengths evaluated. **To avoid a very long run time, the user of the script can set the constant number, EXTENDEDRANGE on line 18, to 6**.

Afterwards, the algorithm uses the new filter length along with an error and weighing function generated but the input specifications to a symmetric FIR filter. It will then verify the pass and stop band ripple before any coefficients are quantized by parsing through every sample in the bands returned by MATLAB's freqz() function. At the same time, it will measure the amount of margin available in both bands. This sets the boundary for the roundoff noise's magnitude. I used the equation (1) to calculate the minimum number of bits. The goal is to make sure the roundoff noise does not cause the ripples in either band to exceed the specifications.

$$|Q(e^{j\omega})| \leq \frac{\Delta}{2\sqrt{2}}\sqrt{N} \tag{1}$$

$$\text{where } \Delta = \tfrac{1}{2^n}$$

Once the number of bits is determined, the algorithm will quantize the filter coefficients and verify the pass and stop band ripple again. Once passed, it will generate a -0.6dB sinusoid signal to check the SNR through the formula below. To simulate the roundoff noise, I wrote a function to calculate each value of y[n] the same way my filter architecture would and applied MATLAB's quant() function after every multiplication.

$$y[n] = \Sigma_0^N h[m]x[n-m] \tag{2}$$

After verifying that the SNR is above the specification, the algorithm will decrement the number of bits by 1 and repeat the process to try and see if it can get away with a lower cost. This will also doublecheck that the number of bits chosen by equation (1) is actually the minimal number of bits.

The algorithm will record the designed filter's length, number of bits, SNR, and cost. If the filter was able to pass all the specifications at least once before decrementing the number of bits, it will decrement the filter length and repeat the entire process.

Algorithm 1: `Optimal Filter Scan`

Input: Lowpass filter specifications
Result: Filter length, number of bits, and SNR of the lowest cost filter

1. Calculate PM filter length estimate and add a constant;
2. Generate the filter with the set filter length;
3. Verify pass and stop band ripple;
4. Calculate number of bits from maximum allowable roundoff noise;
5. Quantize filter coefficients;
6. Verify pass and stop band ripple again;
7. Simulate sinusoid and apply filter to measure SNR;
8. If all specs were met, decrement number of bits and repeat from 5.;
9. Record filter length, number of bits, and SNR;
10. If all specs were met at least once, decrement filter length and repeat from 2.;

return Filter Length, Number of Bits, and SNR of lowest cost filter ;

2 Filter Architecture and Calculating Cost

2.1 Exploiting Symmetry

I took advantage of the filter's symmetry to reduce the number of multipliers in my architecture. The output of an FIR is shown in equation (2), but can be expanded to:

$$y[n] = h[0]x[n] + h[1]x[n-1] + ... + h[N-1]x[N-1] \tag{3}$$

Since we know h[i] = h[N-1-i], equation (3) can be changed to these two equations:

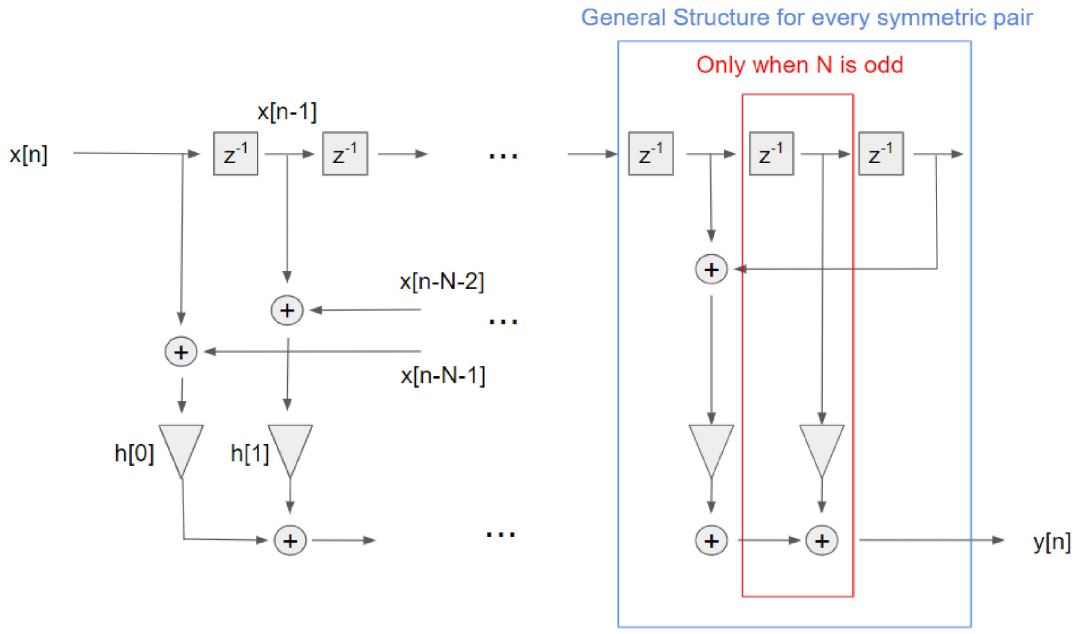
$$y[n] = \Sigma_{i=0}^{\frac{N}{2}} h[i](x[n-i] + x[n-N-1-i]) \tag{4}$$

$$y[n] = [\Sigma_{i=0}^{\frac{N-1}{2}} h[i](x[n-i] + x[n-N-1-i])] + h[\frac{N-1}{2}+1]x[\frac{N-1}{2}+1] \tag{5}$$

Equation (4) represents an even filter length and equation (5) represents an odd filter length.

2.2 Block Diagram

The block diagram of these equations is shown below. The box highlighted in blue shows the general pattern of the filter structure for every two samples of x[n]. The box in red shows the hardware that is only present when the filter length is odd and only appears once in the middle of the filter.



Since $|x[n]| \leq ||X_1|| \leq 0.5$, there will not be any overflow issues after the adders that sum x[n-i] and x[n-N-i]. Since the gain of the filter is less than one, $\Sigma_{i=0}^{N-1} h[n](x[n-i] + x[n-N-i])$ stays within the bound of y[n]. Therefore, the trail of adders leading to the y[n] output will not need any scaling. In summary, the filter architecture does not need any extra scaling.

2.3 Modeling Cost

I calculated the hardware cost from the general pattern. Every two samples of x[n] has 2 registers, 2 adders, 1 multiplier, and another register to hold the filter coefficient for the multiplier. The special case for odd length adds 1 register, 1 multiplier+register, and 1 adder. Furthermore, the first calculation with x[n], x[n-N-1], and h[0] has 1 less register and adder since it is at the very front. In summary, the total

cost of an even length filter is modeled with equation (6) and an odd length filter is modeled with equation (7).

$$Cost = \frac{N}{2}(2n^2 + 23n) - 9n \tag{6}$$

$$Cost = \frac{N-1}{2}[2n^2 + 23n] + 2n^2 + 5n - 9n \tag{7}$$

3 Resultant Filter

3.1 Performance

After running my optimal filter scanning MATLAB script, I received a filter with the following performance:

Property	**Result**
Filter Order	104
Number of Bits	15+1
Passband Edge	Between 0.1992π and 0.2031π
Stopband Edge	Between 0.2480π and 0.2500π
Stopband Attenuation	60.2203
Passband Ripple	0.0953
Passband Group Delay Variation	0
SNR for -6dB full-scale sinusoid	74.5dB
Total Cost	41865

3.2 Verification Methodology

To measure the **pass and stop band edges** empirically, I ran my filter through MATLAB's freqz() function to receive the filter's frequency response as an array. I parsed through array to find the first point where the magnitude falls below the passband ripple and recorded the two frequencies where the intersect occurs between. I did the same for the stop band ripple. A graph of the frequency response is shown in Appendix Figures 2 and 3.

Quantitatively, it makes sense that the pass and stop bands stay at the original specification's frequency. From equation (8), this filter was designed through the optimization technique with an error function and weighing function that are bounded by the original specification. Since the desired filter, $H_D(e^{j\omega})$ is an ideal lowpass that also meets specifications, the resultant filter, $H(e^{j\omega})$, has the same pass and stop bands.

$$H(e^{j\omega}) = H_D(e^{j\omega}) + \frac{\epsilon(e^{j\omega})}{W(e^{j\omega})} \tag{8}$$

To measure the **passband ripple and stopband attenuation** empirically, I used a similar methodology to measuring the pass and stop band edges. I passed through the frequency response and returned the largest difference from 1 in the passband or 0 in the stopband. I converted them back to dB and attenuation and they matched the original specification.

Quantitatively, it makes sense because of the chosen error and weighing functions used to generate the filter. From equation (8), $H_D(e^{j\omega})$ is ideal, so the ripple magnitude is only defined by $\epsilon(e^{j\omega})$ and $W(e^{j\omega})$. Within MATLAB, I set $|\epsilon(e^{j\omega})| \leq \delta_p$ and $|W(e^{j\omega})| = 1$ in passband and $\frac{\delta_s}{\delta_p}$ in the stopband.

To measure the **SNR** empirically, I generated a -6dB sinusoid and passed it through the filter. Then, I used MATLAB's SNR() functin on the output, which constructs a periodogram from the result and compares the magnitude of the largest spike (representing the signal power) against the sum of the surrounding bins (representing the noise power).

Quantitatively, we can calculate the SNR from the ratio for signal power to noise power. For a filter of length 105, there are 53 multipliers, so $\sigma_e^2 = 53\frac{\Delta^2}{12}$. For a -6dB sinusoid, signal power is 0.125. Equation

(9) shows how the SNR is greater than the original specifications.

$$10log_{10}SNR = 10log_{10}|\frac{SignalPower}{NoisePower}| = 10log_{10}|\frac{0.125}{53\frac{\Delta^2}{12}}| = 74.827 \tag{9}$$

To measure group delay empirically, I used MATLAB's grpdelay() function. This returned a constant 52.5 in the passband. The linear phase response is shown in the frequency response graphs in Appendix Figures 2 and 3.

Quantitatively, it makes sense that the group delay is 0 since it is a symmetric FIR filter. This has been proven in class and in homework assignments.

4 Appendix

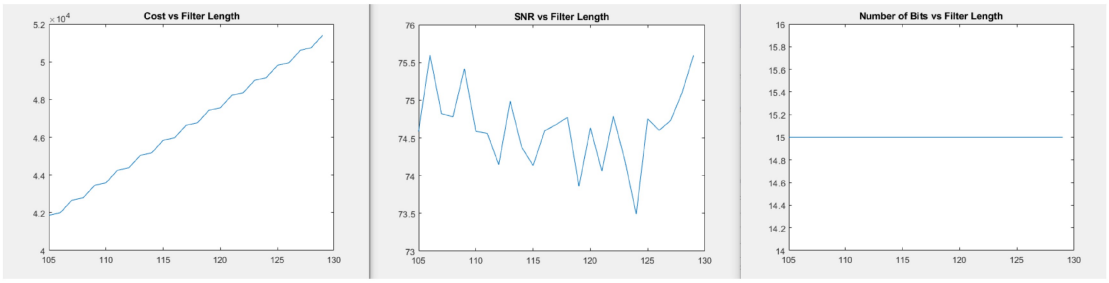


Figure 1. Graphs of the number of bits, SNR, and cost relative to filter lengths scanned by my algorithm.

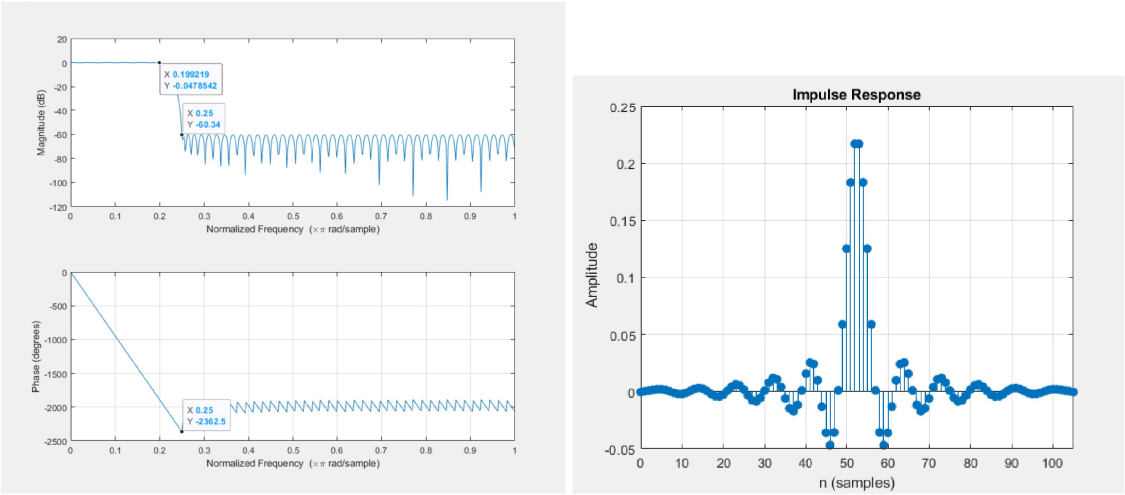
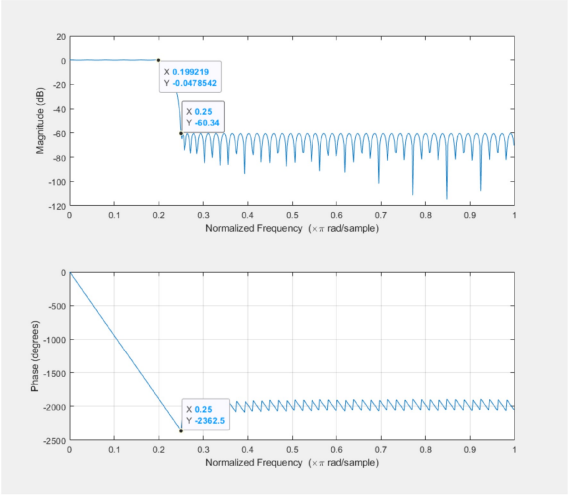


Figure 2. Unquantized filter's frequency response (left) and impulse response (right). Passband and stopband are labeled in the frequency response, and the phase response is linear in the passband.



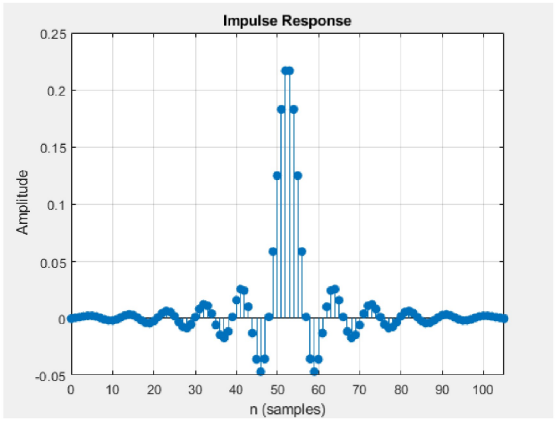


Figure 3. Quantized filter's frequency response (left) and impulse response (right). Passband and stopband are labeled in the frequency response, and the phase response is linear in the passband.

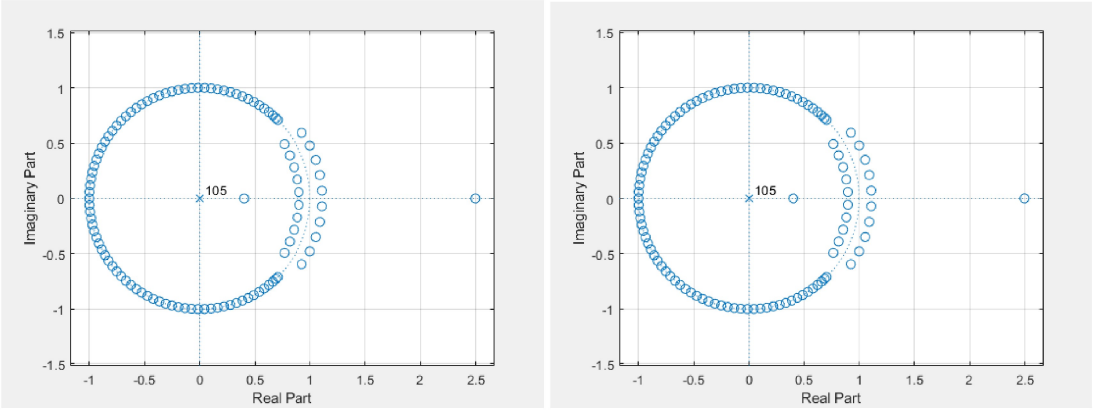


Figure 4. Unquantized filter's pole-zero plot (left) and quantized filter's pole-zero plot (right). There is no major difference between the two and stability is guaranteed since the filter has no poles.