Polyphase Filter Banks: A Physicist's Attempt

Matthew Cooper

May 17th, 2019

Contents

1 Introduction **2**

2 Fundamentals of Digital Signals **4**

2.1 Analog Vs. Digital 4

2.2 Analog Signal Sampling 5

2.3 Downconversion 6

2.4 Digital Signal Filters 8

2.5 Summary 10

3 Polyphase Implementation **12**

3.1 Theory 12

3.2 2-tap Example 13

3.3 Coded Example 16

3.4 Discussion/Future Plans 23

4 Conclusion **25**

Chapter 1

Introduction

In the modern era, the speed of digital processing components has made it such that the gap between digital and analog is becoming ever smaller, allowing us the advantages that come with conversion of analog signals to digital signals. These advantages are numerous, but, as with anything, there is an equivalent exchange, and there are tradeoffs in using digital signals which, if not taken into account, can turn a scientific data product into something completely unusable.

In my researching of this subject, it became clear early on that my understanding of the fundamentals that go into digital signal processing (DSP hereafter) were not up to par, which resulted in several hours of researching different aspects of DSP in order to complete the mental picture I needed. It is my goal in the pages of this report to recreate, as best I can, this journey, making pitstops at different ideas that helped shape my current, albeit incomplete, understanding.

The first reference I was able to find for polyphase filter bank implementation was a paper from 1973 (Schafer and Rabiner, 1973). In this paper, however, the term polyphase had not been coined yet. Its original implementation was

to increase the channel edge resolution for speech analyzers in order to create a perfect reconstruction filter. Not suprising, this came out of Bell Labs. We'll discuss this paper in greater detail when we come to the theory section.

Although it started as a method of reconstructing speech output, the applications of polyphase filter banks has found use in almost all areas where channelization is required, such as cell phone signal downconversion and up conversion (Harris, 2003), as well as being considered for implementation in the Extended Owens Valley Solar Array.

In this report, we will review some basics of analog-to-digital conversion, methods which have been created to navigate the restrictions that such conversions place on the type of data which can be sampled, and then proceed with how polyphase filter banks are utilized in the optimization of such a process.

Chapter 2

Fundamentals of Digital Signals

2.1 Analog Vs. Digital

The filtering and manipulation of analog signals has been widely used for decades. Why, then, is there the push for analog-to-digital conversion? For systems where the average power of a signal is the only necessity, staying in the analog domain is perfectly acceptable. The issue arises when a system also needs to accurately measure and record phase information. In analog systems, the gain to phase balance of a signal cannot be maintained to better than 1% over a range of temperatures (Harris, 2003).

Figure 1.1 gives an example of the error which can be introduced by such errors. This is particularily crippling for a multi-antennae radio array, which relies on phase-locking between antennae in order to steer the beam. Since no two antennae's analog components can match identically, leaving the signal in the analog domain can produce spurious phase shifts, which are functions of

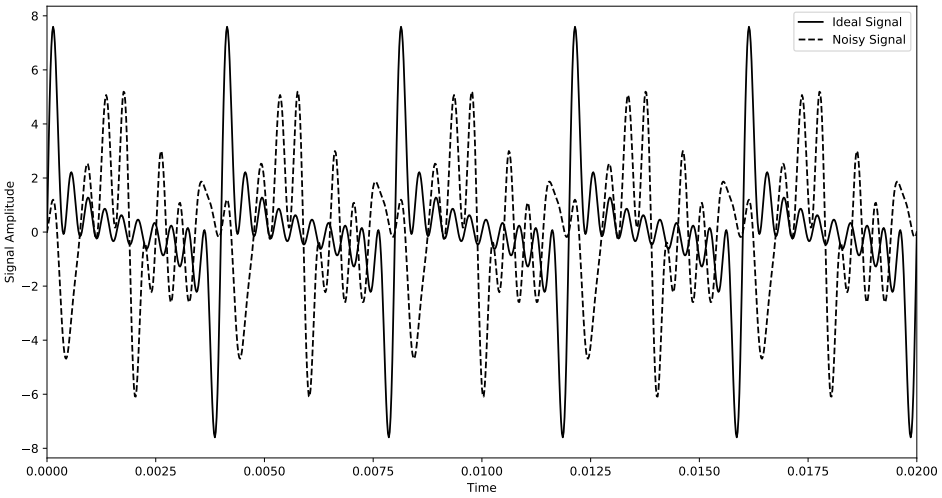

Figure 2.1: Plotted above is an ideal signal which has had uncertainty introduced in its phase and amplitude profile.

many variables which cannot be accounted for simply, if at all. This is where analog-to-digital conversion is helpful, since the phase information stored in a digital signal is not subject to environmental effects.

2.2 Analog Signal Sampling

In analog-to-digital conversion, there are two relations which must be considered: the Nyquist Sampling Theorem, and the Frequency Resolution Relation. These combined determine the sampling rate which must be attained in order to recover a maximum frequency, as well as how much resolution there will be between frequencies. The Nyquist Theorem (Landau, 1967) states that

$$f_{crit} = \frac{f_{sample}}{2} \tag{2.1}$$

where f_{crit} is the maximum frequency which can have power 'definitively' attributed to it (i.e. has no alias), and f_{sample} is the sampling frequency. This relation is troubling for radio transmissions. Consider a 10 GHz signal. In order to recover this signal, it must be sampled at 20 billion samples per second. At 64 bit resolution, this equates to around 160 Gigabytes of data. Currently, FPGA's on the market sit in the MHz processing speed range, although ADC converters currently exist which can handle giga-samples per second.

The second relation, the Frequency Resolution Relation, gives a size requirement for the hardware register which must be fed to the FFT in order to maintain a specific resolution between frequencies.

$$f_{res} = \frac{f_{sample}}{N} \tag{2.2}$$

where f_{res} is the resolvability. So, for higher sample rates, the register size must also increase to maintain frequency resolvability. Clearly, a register which is of the order of 160 Gigabytes is something science fiction would hesitate to create, so how is this issue circumvented? The obvious answer is to sample less, but therein lies the problem, since in the process we lose information either by loss of resolution or by ambiguity in frequency aliasing. The method which was created to address this issue was the concept of downconversion.

2.3 Downconversion

Downconversion (heterodyning) was originally worked on by Nikola Telsa and Reginald Fessenden (Espenschied 1959). Fessenden patented the heterodyne principle in 1902, and that same year founded the National Electrical Signaling Company (NESCO). John Vincent Lawless Hogan, who went to work for Fessenden in 1910, showed how the concept had greatly improved the sensitivity

of radio receivers (Godara, 1999). The introduction of this concept revolutionized electromagnetic signaling, and it is no stretch to say that without it, the subject of this paper, as well as many other DSP related ideas, would never have existed.

What is heterodying? Heterodyning is the process by which a signal is mixed with a higher frequency signal in order to mirror the signal to a lower frequency band. This can be easily shown mathematically. Consider a monochromatic input signal, x_{in}, and a local signal, x_{lo}. Then,

$$x_{in}x_{lo} = sin(f_{in}t)sin(f_{lo}t) \tag{2.3}$$

$$= \frac{1}{2}[cos((f_{lo} - f_{in})t) - cos((f_{lo} + f_{in})t] \tag{2.4}$$

Here, we see that there are two mirrors of the input signal; one which is at higher frequencies, and one which is at lower frequencies, with an example shown in Figure 2.2.

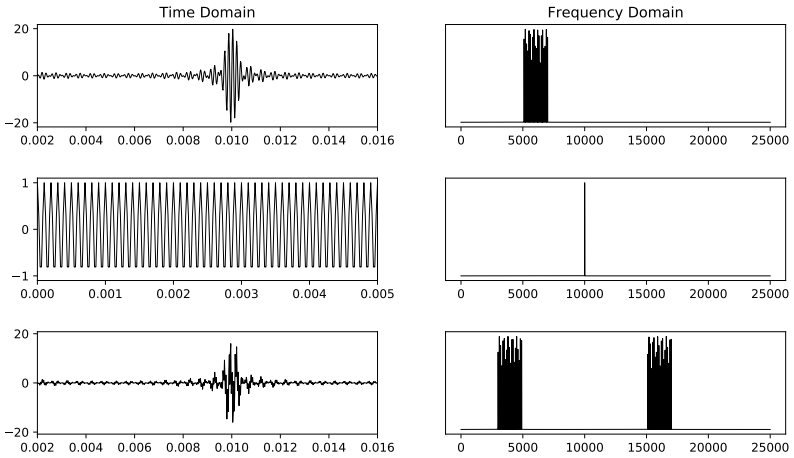

Figure 2.2: (top) Signal x_{in} in the time and frequency domain (middle) Signal x_{lo} in time and frequency domain (bottom) Signal $x_{in}x_{lo}$ in the time and frequency domain

The part we are interested in is, of course, the lower frequencies, since we can lower the sampling rate of these frequencies without losing their information. Therefore, the higher frequency components must be removed. This is accomplished by convolving the input signal with a transfer function known as a filter function.

2.4 Digital Signal Filters

Digital signal filters are arguably one of the biggest fields within the realm of electrical engineering, and for good reason. This process allows one to attenuate certain frequency bands, while leaving others (relatively) unaffected.

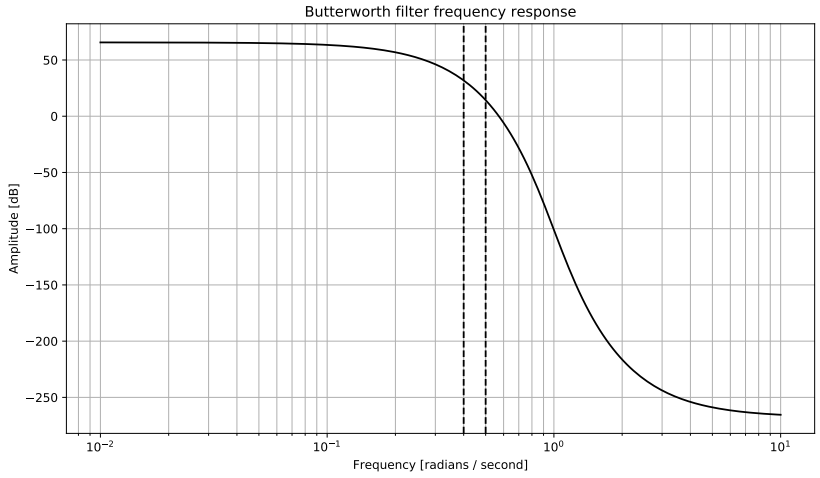

Figure 2.3: Plotted above is the frequency responses for a Butterworth filter of low-pass type. The y-axis indicated the attenuation (in dB) of the different frequency components. The first dashed line indicated the end of the pass-band, and the second is the beginning of the stop band.

Figure 2.3 shows a typical frequency response profile for a filter transfer function. As can be seen, all frequencies are affected, but the stopband frequences are attenuated much more heavily than in the passband. Other filter

types, such as Chebyshev, Bessel, and Elliptic, have other features which would appear in such a plot. The Butterworth filter has no rippling in the attenuation of its passband, whereas an Elliptic or Bessel filter would. Thus, different filters are chosen for different applications. Filters are, of course, not restricted to the digital domain. Specific bandpass filters have been created for radio astronomy in particular, such as the cryogenic S-band filter, which is a hardware bandpass filter which mimics a Chebyshev/Elliptic bandpass filter (Srikanta, 2012). This filter is an example of how analog components still find their place in modern systems. The drawback to such a filter is that once installed, the filter cannot be altered without completely replacing the hardware. A digital filter can be altered simply by changing the coefficient register in the FPGA. The tradeoff is that digital filters with very narrow passbands are subject to artifacts and numerical instability, whereas an analog filter sampling frequency is essentially infinite, so numerical instability is a non-issue.

Digital signal filters are commonly derived in the Laplace domain, due to the simplicity of the mathematics. The task is then to convert it to the Z-domain, via transforms such as a bilinear transform. One then takes the impulse response of a given filter, which for an FIR filter gives the coefficients with which to convolve the input signal, via.

$$y[n] = \sum_{k=0}^{N-1} x[k]h[n-k] \tag{2.5}$$

Example time-domain coefficients for the Butterworth filter above are given in Figure 2.4. The results of such a convolution can be observed in Figure 2.5, where we have taken the down-converted signal from above and implemented a bandpass Butterworth filter. This signal can now be decimated (down-sampled) without any loss of information.

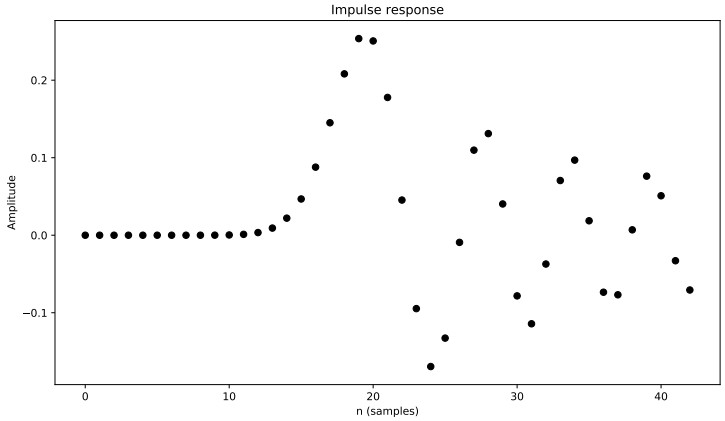

Figure 2.4: Plotted above is the impulse response, h[n], for a Butterworth filter of low-pass type. The y-axis indicated the coefficient value.

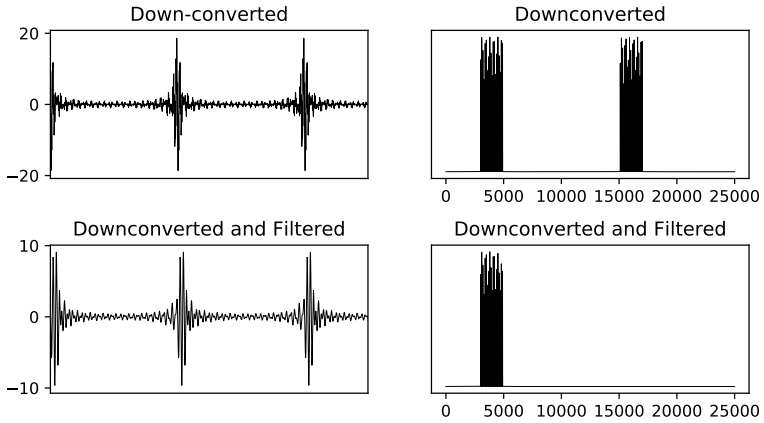

Figure 2.5: (top) Downconverted signal and its Fourier Transform. (bottom) Downconverted and filtered signal and its Fourier Transform.

2.5 Summary

In this section, we have gone through the steps a signal takes after being measured by a front end system and converted to a digital signal. This process involves sampling as a given rate, downconverting to a lower/higher frequency

band, filtering to remove the high frequency component, then decimating the low frequency result. As one can imagine, this process can be very hardware expensive. Are there ways to make this process more efficient? It happens to be the topic of this paper.

Chapter 3

Polyphase Implementation

Up to this point, we have discussed the individual components that make up digital signal processing. What, then, is a polyphase filter bank? A polyphase filter bank (PFB) is a method of deconstructing the input signal and filter in order to more efficiently carry out the convolution-decimation process which was outlined above. In a direct implementation, as will be shown below, several of the multipications and additions are unnecessary, and will be discarded during the decimation process. The PFB allows one to avoid these extra calculations, thus limiting the amount of hardware needed, which is especially important when utilizing finite-space FPGA boards.

3.1 Theory

This section will rely heavily on a paper by (Schafer and Rabiner, 1973), wherein they describe a method for perfect reconstruction across a signal band by a series of bandpass filterings. They begin by defining the series of bandpass filters as

$$h_k(n) = h(n)cos(w_k n) \tag{3.1}$$

where $\omega_k = \triangle\omega * k$, $\triangle\omega$ is the distance between center frequencies of adjacent bands, and k is an integer value. The filter for the entire band is then

$$\widetilde{h}(n) = \sum_{k}^{M} h_k \tag{3.2}$$

which, when substituted in, yields

$$\widetilde{h}(n) = h(n) \sum_{k}^{M} cos(w_k N) = h(n)d(n) \tag{3.3}$$

Schafer and Rabiner realized that if $\triangle\omega = \frac{2\pi}{NT}$, and $M = \frac{N-1}{2}$, where M is the number of channels, then

$$d(n) = \frac{sin(\pi n)}{sin(\frac{\pi n}{N})} \tag{3.4}$$

They go on to show that if one were to introduce a delay in d(n), they can remove phase echos which occur during the filtering process. The relation to the implementation below is reasonable. We split the filter and signal into subfilters and subsignals, as they have done with the channels. We then time delay the filters relative to one another, as they show here, and as (Harris, 2003) shows. In (Harris, 2003), this time delay is given as an unit delay, however. Another unresolved issue is that, for Schafer and Rabiner, they choose the lowpass filter to be zero at integer multiples of the integer N, which is a function of the number of channels. It became unclear as to how this should be implemented in the breakdown of the filter into subfilters.

3.2 2-tap Example

This section provides a mathematical walkthrough of a simple 2-tap polyphase implementation. It suffices to exhibit the time delay which the second filter

must have in order to recover the same output as the direction convolution-decimation. In a direct computation of the convolution of a signal/filter pair, as shown in Figure 3.1, where the signal is $x[n] = \{x_0, x_1, x_2, x_3, x_4, x_5\}$, and the filter is $h[n] = \{h_0, h_1, h_2, h_3\}$, where the lengths are L=6 and M=4, respectively,

h_0x_0	h_0x_1	h_0x_2	h_0x_3	h_0x_4	h_0x_5	0	0	0
0	h_1x_0	h_1x_1	h_1x_2	h_1x_3	h_1x_4	h_1x_5	0	0
0	0	h_2x_0	h_2x_1	h_2x_2	h_2x_3	h_2x_4	h_2x_5	0
0	0	0	h_3x_0	h_3x_1	h_3x_2	h_3x_3	h_3x_4	h_3x_5
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8

Figure 3.1: The above table represents the computations necessary to implement a direction convolution. Each column in the table is summed to create the y_i coefficients at the bottom.

We can easily see that there are 24 multiplications (L*M), and 27 additions (M-1)*(L+M-1). Now, if we change our sample rate, as shown in Figure 3.2,

Before Decimation	After Decimation
$y_0 = h_0x_0$	$y_0 = h_0x_0$
$y_1 = h_0x_1 + h_0x_1$	0
$y_2 = h_0x_2 + h_1x_1 + h_2x_0$	$y_2 = h_0x_2 + h_1x_1 + h_2x_0$
$y_3 = h_0x_3 + h_1x_2 + h_2x_2 + h_3x_1$	0
$y_4 = h_0x_4 + h_1x_3 + h_2x_2 + h_3x_1$	$y_4 = h_0x_4 + h_1x_3 + h_2x_2 + h_3x_1$
$y_5 = h_0x_5 + h_1x_4 + h_2x_3 + h_3x_2$	0
$y_6 = h_1x_5 + h_2x_4 + h_3x_3$	$y_6 = h_1x_5 + h_2x_4 + h_3x_3$
$y_7 = h_2x_5 + h_2x_4$	0
$y_8 = h_3x_5$	$y_8 = h_3x_5$

Figure 3.2: In the above left column are the coefficients produced by the direct convolution, and in the above right the coefficients remaining after a 2-tap decimation (every other sample).

we see that large fraction of the multiplications and additions that were carried out have now been thrown away. Now to show how a polyphase implementation avoids these unnecessary computations. First, the filter is split into N-tap

subfilters, in this case N=2, which gives $h[n]^+ = \{h_0, h_2\}$ and $h[n]^- = \{h_1, h_3\}$. Then, the filter is convolved with a sub-band of the input signal, $x[n]$, which are $x[n]^+ = \{x_0, x_2, x_4\}$ and $x[n]^- = \{x_1, x_3, x_5\}$, respectively. These convolutions are shown in Figure 3.3 and 3.4.

h_0x_0	h_0x_2	h_0x_4	0
0	h_2x_0	h_2x_2	h_2x_4
y_0^+	y_1^+	y_2^+	y_3^+

Figure 3.3: This table represents the convolution of $h[n]^+$ and $x[n]^+$.

The number of multiplications has been reduced to twelve. We have thus reduced the number of multiplications by half, which for a 2-tap decimation seems reasonable. However, we have also lowered the number of summations, from twenty seven to thirteen.

h_1x_1	h_1x_3	h_1x_5	0
0	h_3x_1	h_3x_3	h_3x_5
y_0^-	y_1^-	y_2^-	y_3^-

Figure 3.4: This table represents the convolution of $h[n]^-$ and $x[n]^-$.

First Filter	Second Filter	Convolution and Decimation
$y_0^+ = h_0x_0$	$y_{-1}^- = 0$	$y_0^+ + y_{-1}^- = h_0x_0 = \mathbf{y_0}$
$y_1^+ = h_0x_2 + h_2x_0$	$y_0^- = h_1x_1$	$y_1^+ + y_0^- = h_0x_2 + h_1x_1 + h_2x_0 = \mathbf{y_2}$
$y_2^+ = h_0x_4 + h_2x_2$	$y_1^- = h_1x_3 + h_3x_1$	$y_2^+ + y_1^- = h_0x_4 + h_1x_3 + h_2x_2 + h_3x_1 = \mathbf{y_4}$
$y_3^+ = h_2x_4$	$y_2^- = h_1x_5 + h_3x_3$	$y_3^+ + y_2^- = h_1x_5 + h_2x_4 + h_3x_3 = \mathbf{y_6}$
$y_4^+ = 0$	$y_3^- = h_3x_5$	$y_4^+ + y_3^- = h_3x_5 = \mathbf{y_8}$

Figure 3.5: The table above shows how the coefficients of the separate convolutions are summed.

Now, combining the results of these two separate convolutions as in Figure 3.5, but in the order that the theory suggets, we take the second output to be delayed by a timestep, which effects a spectral folding of sorts, but with the

added benefit of the stopband phase offsets destructively cancelling one another (in theory).

3.3 Coded Example

Herein is presented an example Python code which implements a 5-tap 10, and 20-tap polyphase decomposition of an ideal lowpass filter. For the filter, an ideal Butterworth lowpass filter from the Python scipy library was utilized, with the passband edge set at 5000 Hz and the stopband edge set at 7000 Hz, with maximum attentuation in the passband of .1 dB and minimum attenuation in the stopband of 37 dB. This resulted in a 20th order filter. As a reference for this code, a powerpoint from Binghamton University (Fowler) was consulted. A vital aspect is the sequence with which the signal subbands are multiplied by the filter subbands. It is not direct, and there are zero pads that are necessary in order to recover the proper coefficients. Other than the first signal band, the rest of the bands are multiplied with the filter coefficients in reverse order. The first has a zero appended to the end of the subband, while the rest have zeros prepended to the beginning. Without these two steps, you will get back nonsense, as was discovered.

To demonstrate that the algorithm is in fact behaving appropriately, three separate signals were chosen for all three filters. This allows us to examine frequencies in the three main bands of the filter: the passband, the transition band, and the stopband, as well as to see that as the number of taps increases, the flat channel response does, in fact, begin to manifest.

The first of signal analyzed contained fourty frequency components, separated at 100 Hz, ranging from 0 kHz to 4 kHz, which has been shown in Figure 3.6. From this, we see that the implementation is indeed working, and that there has been a shift in the phase response between the two. Of course, the

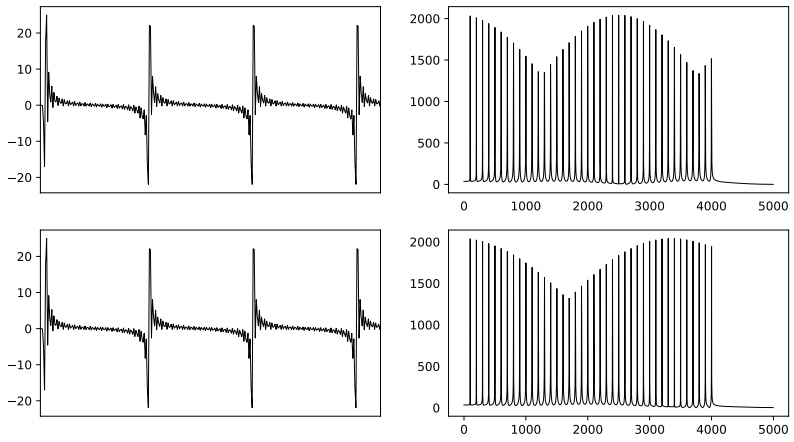

Figure 3.6: (top) is an example of a direct implementation of the Butterworth filter using Scipy lfilter and the Fourier Transform of its output after decimation for the first signal using 5-tap. (bottom) is the polyphase implementation of the same filter on the same signal with 5-tap.

flat channel response that is expected still doesn't occur, though this is likely due to the filter that was chosen not having its zeros located in the correct places, as was described in Schafer and Rabiner. However, if we increase to a 10-tap, the phase pattern does, in fact, change, as is shown in Figure 3.7. It seems that increasing the tap number helps approach this ideal that was discussed. Figure 3.8 shows the maximum that was possible, a 20-tap filter. Notice that the number of taps is directly related to how much decimation the signal undergoes. Thus, the frequency band has decreased with successively larger N-taps.

Indeed, for the last case, the flat channel response compared to a direct implementation seems to be a reasonable outcome, although I'm not sure why there is extra power in certain bands and not in the others. The likely culprit is a bit of aliasing since the 20-tap was the largest decimation.

For the second signal, the number of components was kept at fourty with

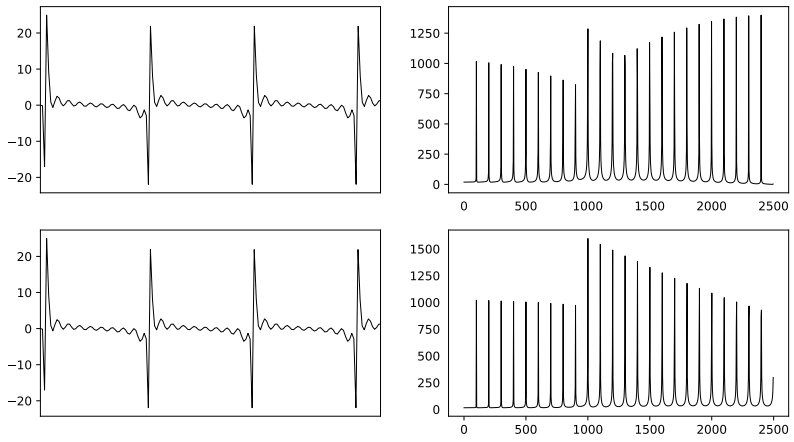

Figure 3.7: Direct (top) vs. Polyphase (bottom) Implementation for first signal using a 10-tap filter.

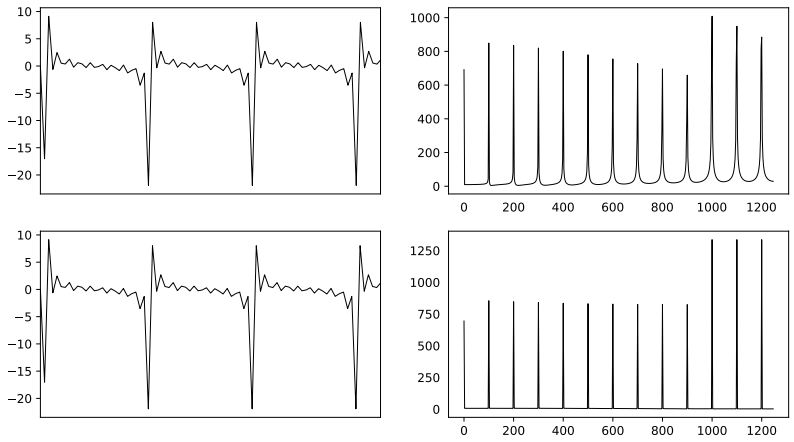

Figure 3.8: Direct (top) vs. Polyphase (bottom) Implementation for first signal using a 20-tap filter.

the same spacing, but was shifted so that the range was 4 kHz to 8 kHz, giving frequency contributions from the high end of the passband, the entire transition

band, and the low end of the stopband. The results of this for the 5-tap are plotted in Figure 3.9.

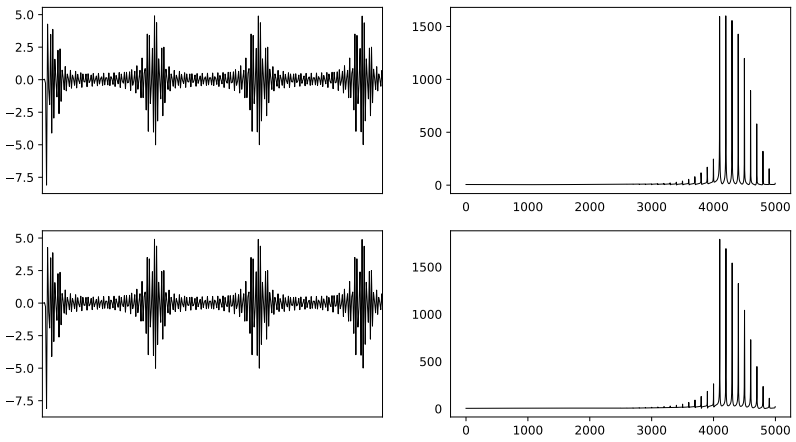

Figure 3.9: Direct (top) vs. Polyphase (bottom) Implementation for second signal using a 5-tap filter.

With much relief, we the that both methods recover the same response. This was the part where I originally discovered that an error was present. Next, the 10-tap case is plotted in Figure 3.10.

Notice the shift in where the frequencies mapped after the higher decimation. A cursory logical nod seems to say this checks out, since these frequencies were at the edge of the passband and have shifted to the lower end from the decimation. This trend seems to hold up in the case of the 20-tap filter, as well, as the frequencies map to the same values, although the scale is less due to the higher decimation rate, as shown in Figure 3.11. Also, notice the sharpness of the peaks in the 20-tap case as compared to the direct filter implementation. The resolution difference here is quite considerable, and will be stress tested at the end of this section.

Continuing this trend in the third signal, the components were again shifted,

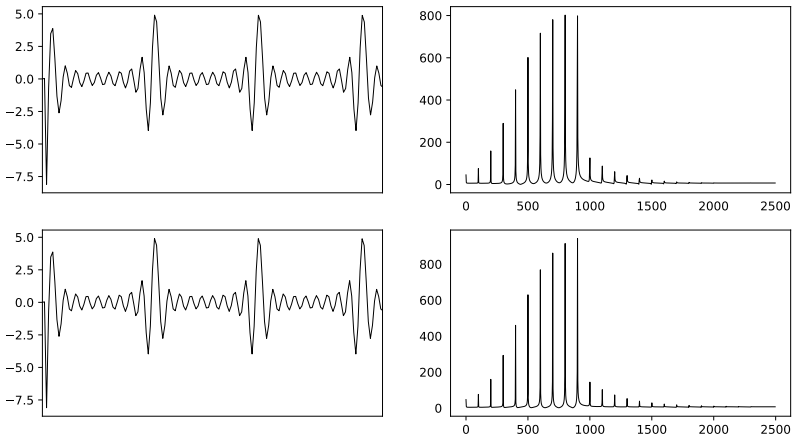

Figure 3.10: Direct (top) vs. Polyphase (bottom) Implementation for second signal using a 10-tap filter.

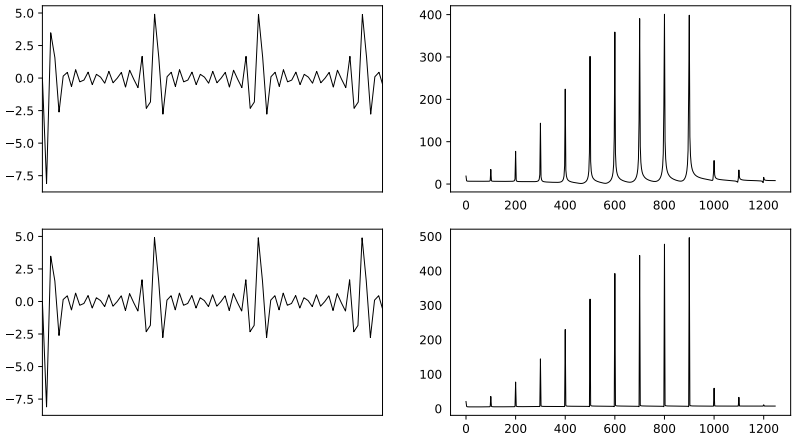

Figure 3.11: Direct (top) vs. Polyphase (bottom) Implementation for second signal using a 20-tap filter.

this time from 6 kHz to 10 kHz. Now, there are no frequencies in the passband. The results of this for the 5-, 10-, and 20-tap are shown in Figure 3.11, 3.12,

and 3.13, respectively.

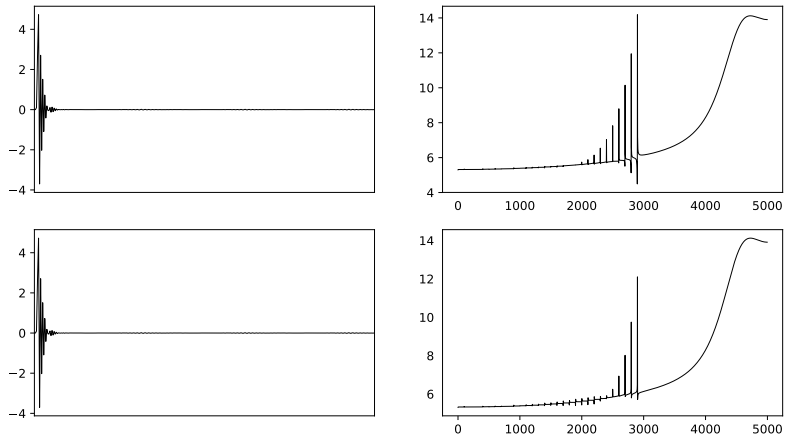

Figure 3.12: Direct (top) vs. Polyphase (bottom) Implementation for third signal using a 5-tap filter.

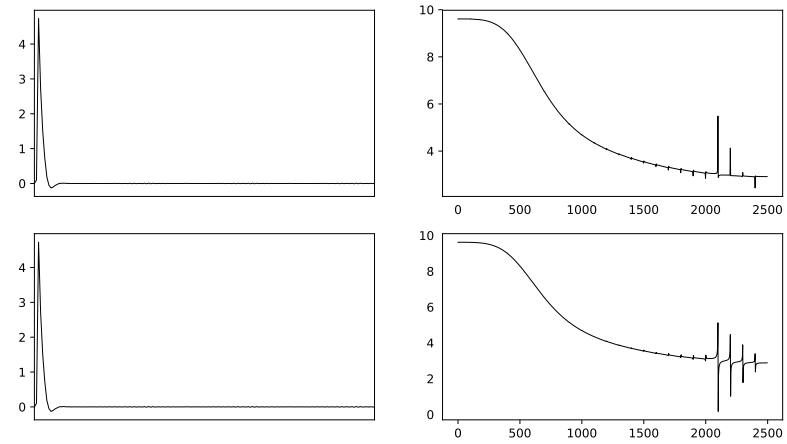

Figure 3.13: Direct (top) vs. Polyphase (bottom) Implementation for third signal using a 10-tap filter.

As is expected, as the decimation number increases, the power attentuation

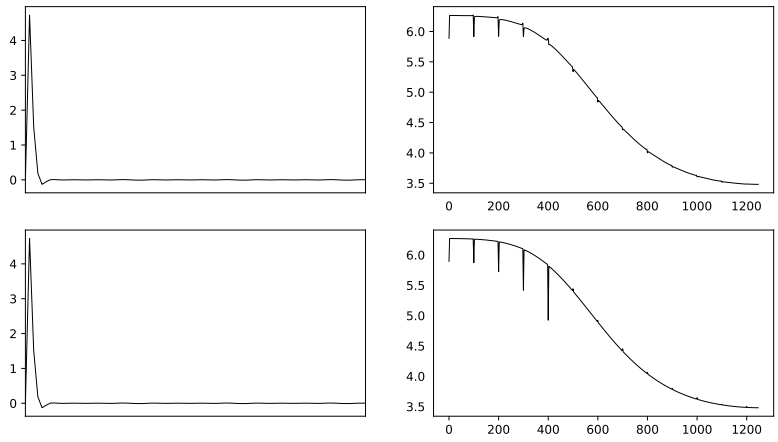

Figure 3.14: Direct (top) vs. Polyphase (bottom) Implementation for third signal using a 20-tap filter.

drops considerably. Interestingly enough, though, there are some odd low power harmonics which become less profound as we increase the tap number. This is likely somehow related to the increase in the frequency resolution as we go to higher N-taps.

```
#-------------------------------------------------------------------
# Implementation of a polyphase downsampler
M = 5
N = 10*4096
signal = FDMSig[500:N+500,0]
sbl = int(signal.shape[0]/M)
sigBands = np.zeros([sbl + 1,M])
sigBands[:sbl,0] = signal[::M]
```

Here is where the time reordering was implemented. I cannot overemphasize how vital this step was to making this algorithm work properly.

```
for i in range(1,M):
```

```
sigBands[1:,M-i] = signal[i::M]
fbl = int(resp.shape[0]/M)
filtBands = np.zeros([fbl, M])
for i in range(0,M):
filtBands[:,i] = resp[i::M]
subConv = np.zeros([sbl + 1, M])
for i in range(0, M):
subConv[:, i] = sig.lfilter(filtBands[:,i], 1, sigBands[:,i])
subConv = subConv.transpose()
polyPhaseOut = subConv.sum(axis=0)
"' Implementation of filtering, then downsampling "'
filtSig = sig.lfilter(b, a, FDMSig[:,0])
decFiltSig = filtSig[::M]
#-------------------------------------------------------------------------
```

Here I have omitted the code used to create the signal.

3.4 Discussion/Future Plans

Hopefully the above is satisfactory to show that the implementation has been successful. Now, lets dive into that interestingly sharp frequency resolution that was seen in the 20-tap case.

It appears that, although it seemed that way in the above plots, this method isn't providing any extra resolution that the original filtering did not. The future plans are to find higher order filters and see if going from GHz to MHz using this method is stable.

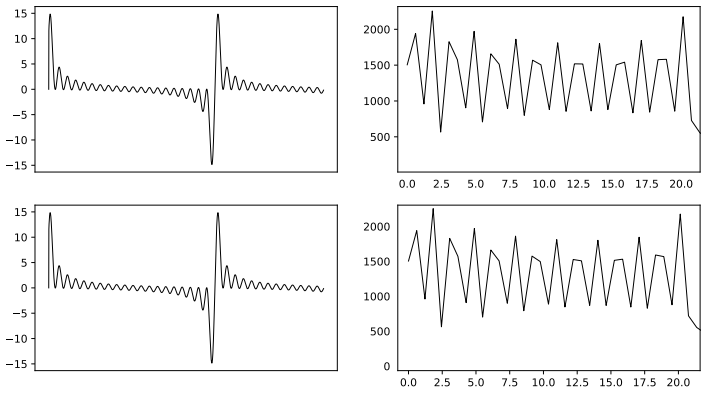

Figure 3.15: Comparison of spectral resolution for the direct and polyphase 20-tap filter process with twenty components separated by 1 Hz.

Chapter 4

Conclusion

From the initial review of DSP basics to the implementation of a divided filter bank, this report has sought to provide a basic insight into the polyphase filter bank implementation of a decimation of a higher frequency signal. Hopefully, this has been successful. In closing...it works!!! This has no doubt been the most trying project I've taken on to date, and the satisfaction of it actually functioning is quite remarkable. Thanks for the fun semester. I've learned quite a lot. And now I'm taking a nap.

Bibliography

Fowler http://www.ws.binghamton.edu/fowler/fowler%20personal%20page/EE521_files/IV-05%20Polyphase%20FIlters%20Revised.pdf

F. J. Harris, C. Dick and M. Rice, "Digital receivers and transmitters using polyphase filter banks for wireless communications," in IEEE Transactions on Microwave Theory and Techniques, vol. 51, no. 4, pp. 1395-1412, April 2003.

H. J. Landau, "Sampling, data transmission, and the Nyquist rate," in Proceedings of the IEEE, vol. 55, no. 10, pp. 1701-1706, Oct. 1967. doi: 10.1109/PROC.1967.5962

J.H., Jr, Hammond, Purington, E.S.. (1957). A History of Some Foundations of Modern Radio-Electronic Technology. Proceedings of the IRE. 45. 1191 - 1208. 10.1109/JRPROC.1957.278525.

L. C. Godara, "Introduction to "The heterodyne receiving system, and notes on the recent Arlington-Salem tests"," in Proceedings of the IEEE, vol. 87, no. 11, pp. 1975-1978, Nov. 1999. doi: 10.1109/JPROC.1999.796359

Pal, Srikanta J. Lancaster, Michael D. Norrod, Roger. (2012). HTS Bandstop Filter for Radio Astronomy. Microwave and Wireless Components Letters, IEEE. 22. 236-238. 10.1109/LMWC.2012.2193122.

R. W. Schafer and L. R. Rabiner, "A digital signal processing approach to interpolation," in Proceedings of the IEEE, vol. 61, no. 6, pp. 692-702, June 1973. doi: 10.1109/PROC.1973.9150