# Simulation of Various Channelizer Structures Directed by Cyclostationary Detector

Brian H. Hulette

Project Report submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Engineering
in
Electrical Engineering

Amir I. Zaghloul, Co-Chair
Jeffrey H. Reed, Co-Chair
T. Charles Clancy

February 18, 2015
Falls Church, Virginia

Keywords: SDR, Cyclostationarity, Detection, Channelizer
Copyright 2015, Brian H. Hulette

# Simulation of Various Channelizer Structures Directed by Cyclostationary Detector

Brian H. Hulette

(ABSTRACT)

One common problem in Software-Defined Radio (SDR) systems is that of detecting and then isolating narrow signals of interest in wideband sampled data. This involves using some means to detect the frequency of one or more signals of interest and then tuning, filtering, and decimating them. This problem is particularly challenging due to the high sample rate of this wideband data, so efficient algorithms are very desirable. Solutions to this have applications in many areas, including both Cognitive Radio and Electronic Warfare.

A MATLAB simulation of an SDR framework for detecting and sub-band tuning channelized signals is presented, with a focus on finding an efficient means to combine the two algorithms. Detection is performed using the Spectral Correlation Density (SCD) function which exploits the cyclostationary property of digital signals [**?**]. These detections are then used to direct a channelizer which will tune, filter, and decimate all of the detected signals. Two different channelizer structures are examined: a polyphase analysis/synthesis channelizer [**?**] and an overlap-save filter bank [**?**].

A summary of relevant background information is provided. This includes a discussion of cyclostationarity and the SCD as well as background on both channelizer structures. For the overlap-save filter bank it will be shown that in many cases computation can be saved by using the same FFT computation for both the cyclostationary detector and the channelizer.

In this simulation, simple QPSK signals are used to model the signals of interest, but the framework is applicable to any modulation or signal type that has cyclostationary features. While other means of signal detection may be more reliable for specific signals, a cyclostationary detector's primary strength is its ability to function for a wide variety of digital signals.

# Acknowledgments

First I must acknowledge the work of both Craig Carlson and Ruth Stoehr. The core of my testbed for this project is a robust QPSK Transmitter/Receiver pair we designed together for an ECE5654 project in the Spring of 2013.

I would also like to thank the members of my committee, particularly Dr. Zaghloul, for their patience and support.

Finally, I would like to thank my friends and family, especially my girlfriend Meredith, for helping me through this years long process, and letting me use "My Masters" as an excuse for nearly anything.

# Contents

**1 Introduction** **1**

**2 Background Information** **2**

2.1 Cyclostationary Detection . . . . . 2

2.1.1 Cyclostationarity . . . . . 2

2.1.2 Estimating the SCD . . . . . 3

2.2 Polyphase Analysis/Synthesis Channelizer . . . . . 4

2.2.1 Combining Analysis and Synthesis Channelizers . . . . . 6

2.2.2 Limitations . . . . . 8

2.2.3 Advantages . . . . . 8

2.3 Overlap-save Filter Bank . . . . . 8

2.3.1 Limitations . . . . . 11

2.3.2 Advantages . . . . . 11

**3 Simulation** **12**

3.1 Testbed . . . . . 12

3.2 Cyclostationary Detector . . . . . 14

3.3 Polyphase Analysis/Synthesis Channelizer . . . . . 16

3.3.1 Adding Cyclostationary Detection . . . . . 18

3.4 Overlap-Save Filter Bank . . . . . 19

3.4.1 Adding Cyclostationary Detection . . . . . 19

**4 Conclusion** **21**

4.1 Future Work . . . . . 21

**A Project Source** **23**

A.1 Cyclostationary Detection . . . . . 23

A.1.1 `cyclostationary_detect.m` . . . . . 23

A.1.2 `cyclic_spectrum.m` . . . . . 24

A.2 Polyphase Channelizer . . . . . 25

A.2.1 `cyclostationary_and_polyphase.m` . . . . . 25

A.2.2 `polyphase_channelizer.m` . . . . . 26

A.3 Overlap-Save Channelizer . . . . . 27

A.3.1 `cyclostationary_and_overlap_save.m` . . . . . 27

A.3.2 `overlap_save_channelizer.m` . . . . . 27

A.3.3 `os_fft.m` . . . . . 28

A.3.4 `os_filter.m` . . . . . 29

# List of Figures


2.1 Creating a single channel of a polyphase analysis channelizer . . . . . 5

2.2 Full Polyphase Analysis/Synthesis Channelizer Structures . . . . . 6

2.3 Using Polyphase Analysis/Synthesis Channelizer Structures in concert. Step (1) is an Analysis Channelizer that breaks the signal into 16 channels, Step (2) is a set of Synthesis Channelizers for each signal of interest, and Step (3) is a complex phasor that corrects the frequency offset. Note that this figure depicts the signal in the frequency domain, simply for ease of illustration - all inputs and outputs are time-domain signals. . . . . 7

2.4 Overlap-Save Filter Bank Structures . . . . . 10

3.1 Frequency spectrum of the 10 MHz test signal used for each module . . . . . 13

3.2 Estimates of the SCD at various cyclic frequencies. FFT size = 1024, Averaged over 10 FFTs. . . . . 15

3.3 Output of a 16 channel polyphase analysis channelizer operating on the test signal. X Axes are frequency in Hz, Y Axes are magnitude in dB. . . . . 17

3.4 Output of the combined cyclostationary detector and polyphase channelizer. All three test signals sampled at four samples per symbol. . . . . 18

3.5 Output of the combined cyclostationary detector and overlap-save filter bank. All three test signals sampled at four samples per symbol. . . . . 20

# Chapter 1

# Introduction

A common problem when designing a Software-Defined Radio (SDR) system is the problem of detecting and tuning signals of interest. A typical SDR will sample at a relatively high rate to acquire a wideband signal. Then it will attempt to detect the channels within the acquired frequency range that contain signals of interest, and then tune, filter, decimate, and demodulate those signals.

Many popular SDR applications perform this detection step with a "man in the loop". A falling raster of the wideband data is presented and the man in the loop will select the frequencies that he would like to process ([**?**] is a web-based implementation of this). A common example of this is to acquire a wideband slice of the HAM radio spectrum and then select different frequencies to demodulate new FM push-to-talk signals and listen to them.

This paper investigates a structure that automates this process - but rather than processing a single analog signal, this framework is intended to process an arbitrary number of digital signals at once. Detection is performed by a cyclostationary detector rather than by a man in the loop. This exploits the cyclostationary properties exhibited by most digital signals with a fixed symbol rate. Tuning, filtering, and decimating is performed by a channelizer, so that many signals can be processed at once. Two different channelizer structures are examined: a polyphase channelizer composed of an analysis channelizer followed by several synthesis channelizer, and an overlap-save filter bank. These structures are evaluated on two important criteria: 1) Their ability to accurately reproduce every detected signal, and 2) Their computational efficiency when combined with a cyclostationary detector.

Chapter 2 provides background information on cyclostationary properties and the SCD upon which the detector relies, as well as both of the channelizer structures that are simulated. Chapter 3 discusses the MATLAB simulation that was created and the results of it. Finally, Appendix A provides annotated source code for key parts of the simulation, as well as locations where the entire source code can be found.

# Chapter 2

# Background Information

## 2.1 Cyclostationary Detection

### 2.1.1 Cyclostationarity

Often signals are modeled as stochastic random processes with properties such as mean and variance, but many man-made signals, particularly digital communications, can be modeled with another statistical property called *cyclostationarity*, which implies the signal has some parameter which varies periodically with time. The frequency of this variation is called the cyclic frequency, $\alpha$. [**?**].

Of particular interest for this project is second-order cyclostationarity, where a signal has a periodic auto-correlation, $R_{xx}(t, t+\tau)$:

$$R_{xx}(t, t+\tau) = E\{x(t)x^*(t+\tau)\} = \sum_{\alpha} R_{xx}^{\alpha}(\tau)e^{j2\pi\alpha t} \tag{2.1}$$

The function $R_{xx}^{\alpha}(\tau)$ is the cyclic auto-correlation function (CAF), given by:

$$R_{xx}^{\alpha}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} R_{xx}(t, t+\tau)e^{-j2\pi\alpha t}dt \tag{2.2}$$

The CAF is a function of two parameters, the cyclic frequency, $\alpha$ and the delay, $\tau$. This function can be used for signal detection in the time domain. An estimate of the two-dimensional function versus $\alpha$ and $\tau$ is computed, then features within this plane are used for detection (e.g. [**?**, **?**]). However, what is of interest for this project is detection in the

frequency domain. This can be performed using the Fourier transform of the CAF, called the Spectral Correlation Density (SCD), given by:

$$S_{xx}(\alpha, f) = \int_{-\infty}^{\infty} R_{xx}^{\alpha}(\tau) e^{-j2\pi f\tau} d\tau \tag{2.3}$$

The SCD is a generalization of the conventional Power-Spectral Density (PSD), which it reduces to at $\alpha = 0$ [**?**]. Like the CAF, the SCD also contains unique features based on the modulation type, symbol rate, and carrier frequency of the transmitted signal(s), which we can exploit for detection. In [**?**], Gardner defines this SCD for various basic modulation types, including QPSK, which is used in the presented simulation.

### 2.1.2 Estimating the SCD

We can estimate a slice of the SCD at cyclic frequency $\alpha$ as the cross spectral density of the two frequency-shifted time sequences $x_L(t) = x(t)e^{j\pi\alpha t}$ and $x_U(t) = x(t)e^{-j\pi\alpha t}$ [**?**].

$$S_{xx}(\alpha, f) = X_L(f)X_U^*(f) \tag{2.4}$$

Note that $x_L(t)$ has been shifted down in frequency by $\alpha/2$, and $x_U(t)$ has been shifted up by the same amount. In order to estimate the SCD in this way we must perform two FFTs: one for both $X_L(f)$ and $X_U(f)$. However, it is possible to estimate both of these spectra using a single FFT, $X(f)$, in some cases:

$$\begin{aligned} S_{xx}(\alpha, f) &= X_L(f)X_U^*(f) && (2.5)\\ &= X(f - \alpha/2)X^*(f + \alpha/2) && (2.6) \end{aligned}$$

Using the second relation we can compute $S_{xx}(\alpha, f)$ using $X(f)$, by shifting that FFT in either direction by $\alpha/2$, and then conjugate multiplying the two shifted spectra together. This is significant, since a simple forward FFT of the acquired data $X(f)$ is useful for other operations, such as channelization, as we will see in Section 2.3.

In order to perform this shift accurately using a single FFT we need to circular shift the FFT by an integer number of bins. This means we can only accurately perform this frequency shift when $\alpha/2$ is a multiple of the FFT resolution. Thus $\alpha$ must satisfy the following relationship:

$$\alpha = \frac{2kf_s}{N_{fft}},\ k \in \mathbb{Z} \tag{2.7}$$

If an estimate of the SCD at a cyclic frequency that does not satisfy this relation is required, then we must take the original approach (perform the frequency shift in the time domain) to be perfectly accurate. Alternatively, we can still use a single forward FFT and approximate the shift by interpolating between bins, but this is only an approximation.

## 2.2 Polyphase Analysis/Synthesis Channelizer

The first channelizer structure we will examine relies on the Analysis and Synthesis Polyphase Channelizers described by Fred Harris [**?**]. A Polyphase Analysis channelizer can be used to break up a wideband signal into $D$ distinct channels, each decimated by a factor of $D$ from the wideband sample rate. While a Polyphase Synthesis Channelizer performs the reverse operation, combining $D$ channels into one wideband signal. The channel frequencies used in both structures are integer multiples of the output sample rate:

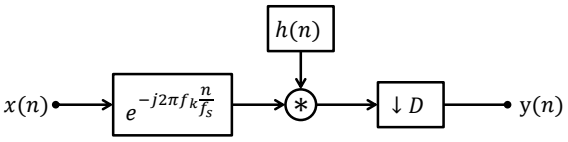
$$f_k = k\frac{f_s}{D} \tag{2.8}$$

where $f_k$ is the center frequency of channel $k$ and $f_s$ is the input sample rate. A Polyphase Synthesis Channelizer performs the reverse operation - it combines $D$ distinct channels with equal sample rate into a single wideband signal.

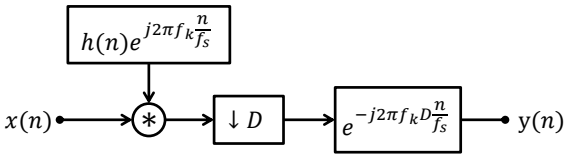
These two structures can be combined to create a very flexible channelizer that can tune channels with arbitrary center frequencies and various bandwidths [**?**] - a simplified version of this structure is discussed in Section 2.2.1.

In [**?**], Harris shows how the Polyphase Analysis Channelizer works by starting with a basic filter/decimator and making incremental modifications until it becomes an Analysis Channelizer as shown in Figure 2.2a. A brief summary of this description is provided here, which follows Figure 2.1.

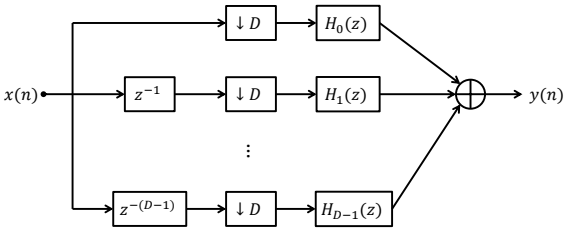
We start with a single channel of a basic channelizer in Figure 2.1a. This channel consists of a multiplication with a complex phasor to tune the desired center frequency down to baseband, followed by a low-pass filter, represented by $h(n)$, and finally a decimation by $D$. The first modification we can make takes advantage of the Equivalency Theorem, which says that we can switch the order of the tuner and the filter, **if** the filter is changed to a bandpass filter centered at $f_k$. This modification is shown in Figure 2.1b. In this figure the tuner has also been moved after the decimator. Note that if the channel frequency, $f_k$, is an integer multiple of the output sample rate, $\frac{f_s}{D}$, then the tuner simplifies to $e^{j2\pi n} = 1$, so it can be dropped completely. Thus we will restrict the polyphase analysis channelizer to center frequencies which are integer multiples of the output sample rate.



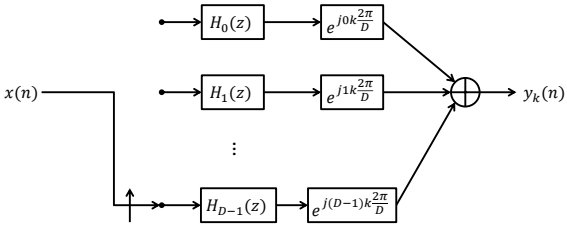

(a) Single channel of a simple tuning, filtering, and decimating channelizer. $h(n)$ is a baseband low-pass filter.




(b) Equivalency theorem allows the tuning step to be moved after the decimation. The baseband filter must be shifted up to $f_k$. If $f_k$ is a multiple of the output sample rate, $\frac{f_s}{D}$, the tuning step can be dropped.




(c) The Noble Identity allows the filtering and decimation to be combined to remove excess computations.




(d) The series of delays and decimation steps can be replaced with an input commutator and complex phasors can be added after each filter to select channel $k$.

Figure 2.1: Creating a single channel of a polyphase analysis channelizer

Next we can use the Noble Identity to switch the place of the decimation and the filter, as shown in Figure 2.1c. In order to do so we need to define a set of new filters, $H_r(z)$, such that:

$$H(z) = H_0(z^D) + z^{-1}H_1(z^D) + \ldots + z^{-(D-1)}H_{D-1}(z^D) \tag{2.9}$$

This means that each filter, $H_r(z)$, has has an impulse response which is $h(n)$ shifted by $r$ samples and decimated by $D$. Using the Noble Identity in this way saves us from performing computations for samples that will just be dropped by the decimator. Note that in Figure 2.1c the tuner has been dropped as well, since we are restricting the channelizer to frequencies which are multiples of the output sample rate.

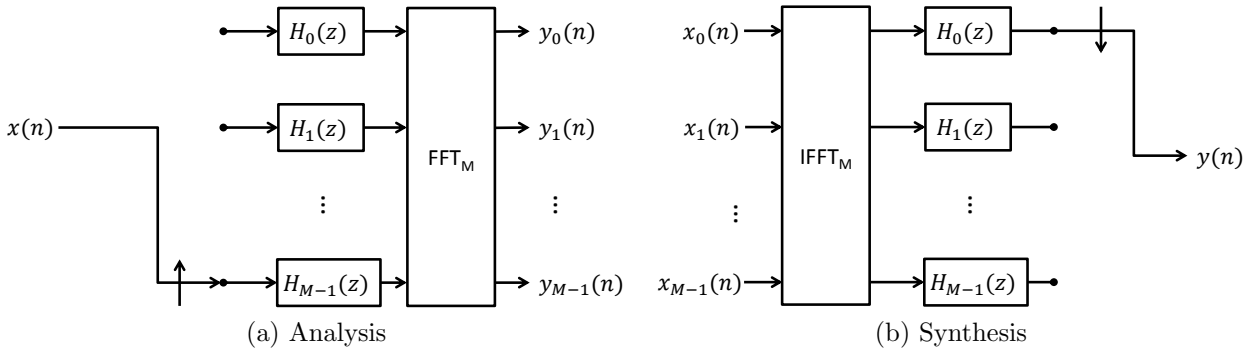
Finally, we complete the structure for a single channel of a polyphase analysis channelizer in Figure 2.1d. The combination of delays and decimators at the front of the previous structure is actually just a commutator. In [**?**] Harris explains this by thinking about the decimators as a switch that closes every $D$ samples. So in Figure 2.1c, when all of the switches close the filter at the bottom gets the oldest sample, and each filter as you go up gets one newer sample until you get to the most recent sample on the top. The next sample is not processed until

the switches close again, and it is passed to the filter at the bottom. With this explanation it is easy to see the whole structure can be replaced with a commutator.

The final structure has one more modification. The outputs are multiplied by complex phasors which select the individual channel, $k$, centered at $f_k$. A detailed description of this can be found in [**?**]. When all of the channels are combined to form the full channelizer these complex phasors all represent a DFT. The set of phasors that are multiplied and then summed together for channel $k$ correspond to the $k$th output of a DFT:

$$y_k(n) = \sum_{r=0}^{D-1} y_r(n) e^{j(2\pi/D)rk} \tag{2.10}$$

Where $y_r(n)$ represents the output of filter $H_r(z)$, and $y_k(n)$ is the output of channel $k$. Thus we can compute all $D$ of the complex phasors with a $D$ point FFT, as shown in Figure 2.2a. Figure 2.2b shows the complementary structure: the Polyphase Synthesis Channelizer. It is essentially a mirror image of the Analysis case - an IFFT followed by filters and a commutator. It combines $D$ channels at an equal sample rate, $f_s$, into a single wideband signal at sample rate $Df_s$. The following section shows how these two structures can be used together to produce a very flexible channelizer.




(a) Analysis (b) Synthesis

Figure 2.2: Full Polyphase Analysis/Synthesis Channelizer Structures

### 2.2.1 Combining Analysis and Synthesis Channelizers

The primary limitations of an Analysis Channelizer alone are that it is limited to a single output sample rate, and the channel center frequencies must be a multiple of that output sample rate. These limitations are perfectly acceptable when every signal being processed is at the same bandwidth and symbol rate, and they are channelized with a known spacing, but in many SDR applications this is not the case. One solution to this problem is to use analysis and synthesis structures *together* to create one very flexible structure.

Figure 2.3 illustrates one simple way that this might be implemented. The first step of this process uses an Analysis Channelizer to break up the wideband into small equal parts (1). Then, Synthesis Channelizers are used to re-combine the parts of the signal that correspond to signals of interest (2). Finally, complex phasors are used to remove the frequency offset created by the Synthesis Channelizers, as well as any residual offset based on the desired center frequency (3).

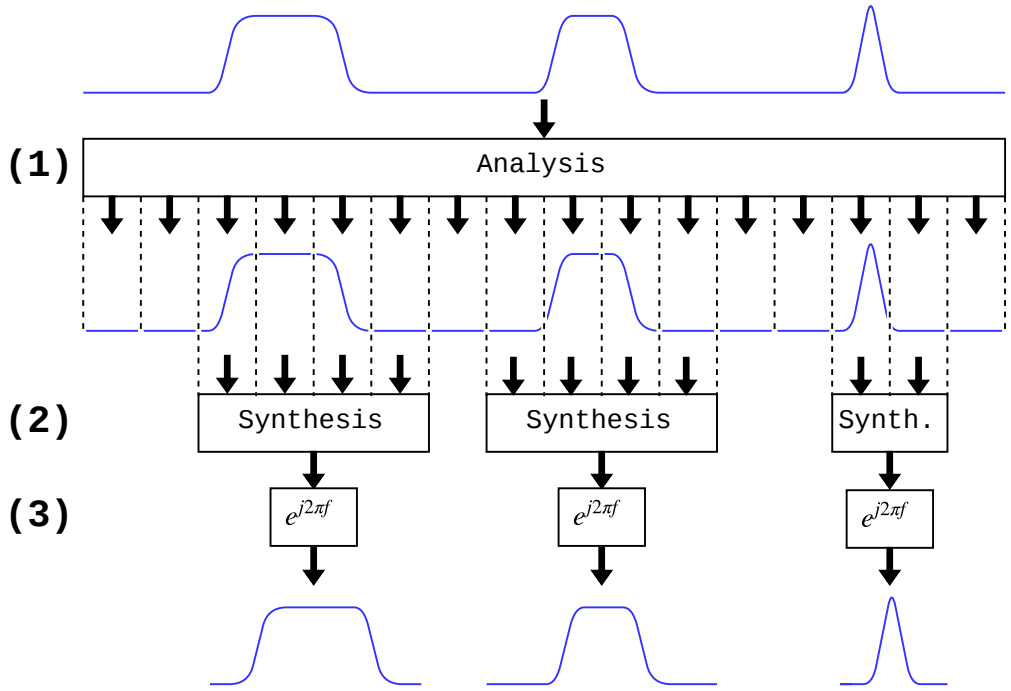



Figure 2.3: Using Polyphase Analysis/Synthesis Channelizer Structures in concert. Step (1) is an Analysis Channelizer that breaks the signal into 16 channels, Step (2) is a set of Synthesis Channelizers for each signal of interest, and Step (3) is a complex phasor that corrects the frequency offset. Note that this figure depicts the signal in the frequency domain, simply for ease of illustration - all inputs and outputs are time-domain signals.

The analysis channelizer can be running all the time, while the synthesis channelizers and phasors in steps (2) and (3) can be dynamically allocated as signals of interest are detected (by some external detector).

[?] describes a more complex take on this same concept, but this simplified version is sufficient for this project.

### 2.2.2 Limitations

We have already examined the limitations of a Polyphase Analysis Channelizer used alone: every channel must be at the same output sample rate, $f_s/D$, and they must have center frequencies which are multiples of that sample rate.

If the combination analysis/synthesis structure is used then this limitation can be avoided, however there are still other issues. First, the output sample rate must still be a decimation of the input sample rate - arbitrary rates are not allowed. Second, while the output signals can be reconstructed quite well after they have been split by the analysis channelizer, the reconstruction is not perfect and could introduce errors. Harris discusses the concept of a "perfect reconstruction" filter [**?**] which may circumvent this limitation, but it is beyond the scope of this project.

Finally, there is no obvious way that this structure can be efficiently combined with SCD estimation for detection. Nothing about an analysis channelizer can be re-used for the detection approach used in this project.

### 2.2.3 Advantages

The primary advantage of a lone Polyphase Analysis Channelizer is its simplicity and efficiency, but its limitations make it an untenable solution for this project.

The combination Polyphase Analysis/Synthesis Channelizer adds some complexity, but it is still quite efficient for the amount of flexibility that it provides. Another major benefit of both Analysis and Synthesis Channelizers is that they can be implemented in hardware relatively easily. That means the Combination Analysis/Synthesis structure could be implemented in hardware. One can imagine a hardware implementation of this structure where an Analysis Channelizer is always running, and a bank of Synthesis Channelizers are dynamically connected and disconnected as signals are detected.

## 2.3 Overlap-save Filter Bank

The Overlap-Save Filter Bank structure used in this simulation is based entirely on a description by Mark Borgerding from March 2006 [**?**]. Borgerding's concept is based on the well known Overlap-Save (OS) fast convolution technique.

OS fast convolution can be used to speed up convolution with a filter that has a long impulse response. The concept is that rather than convolving in the time-domain, $O(N^2)$, it is faster to first perform an FFT of both the signal and the filter and conjugate multiply in the frequency domain, then IFFT to go back to the time domain, $O(N \log_2 N)$. There is nothing new about this idea, but Borgerding's innovation is that he shows how to extend this concept

to tune, filter and decimate any number of channels with arbitrary center frequencies and bandwidths.

We will now see how the OS Filter Bank performs Tuning, Filtering, and Decimation. The same terms that Borgerding defines will be used in this description:

| | |
|---|---|
| $x(n)$ | Input data |
| $h(n)$ | Baseband filter response |
| $y(n)$ | Tuned, filtered and decimated output data |
| $P$ | Length of $h(n)$ |
| $N$ | FFT size |
| $D$ | Decimation factor |
| $V = N/(P-1)$ | Ratio of FFT size to filter order |

*Tuning:* If the frequency of the signal of interest corresponds to one of the FFT bins then we can simply circular shift the FFT output to place that bin at the center. Then filtering can be performed at baseband. If all channels satisfy this criterion then we can re-use a single forward FFT for every channel, simply by circular shifting it to the appropriate bin in every case. This approach is shown in Figure **??**.

However, if this condition is not satisfied then we will need to perform the frequency shift in the time domain by multiplying by a complex phasor. Fortunately, there is a still a way to re-use the forward FFT in this case. After taking an FFT of the non frequency-shifted signal we can shift the baseband filter response up to the desired frequency, then after the filtered signal is returned to the time domain with an IFFT, we perform the frequency shift with a complex phasor. In addition to allowing us to re-use the forward FFT for each channel, this is more efficient than frequency shifting before the forward FFT since the multiplication is performed after decimation. This approach is shown in Figure 2.4b.

*Filtering:* As discussed earlier, this structure relies on OS fast convolution for filtering. The low-pass filter impulse response for each channel is Fourier Transformed and then conjugate multiplied with the FFT of the input data after the signal of interest has been tuned to baseband. Alternatively, if tuning is being performed in the time domain, the filter impulse response must be tuned up to the signal frequency before being Fourier transformed.

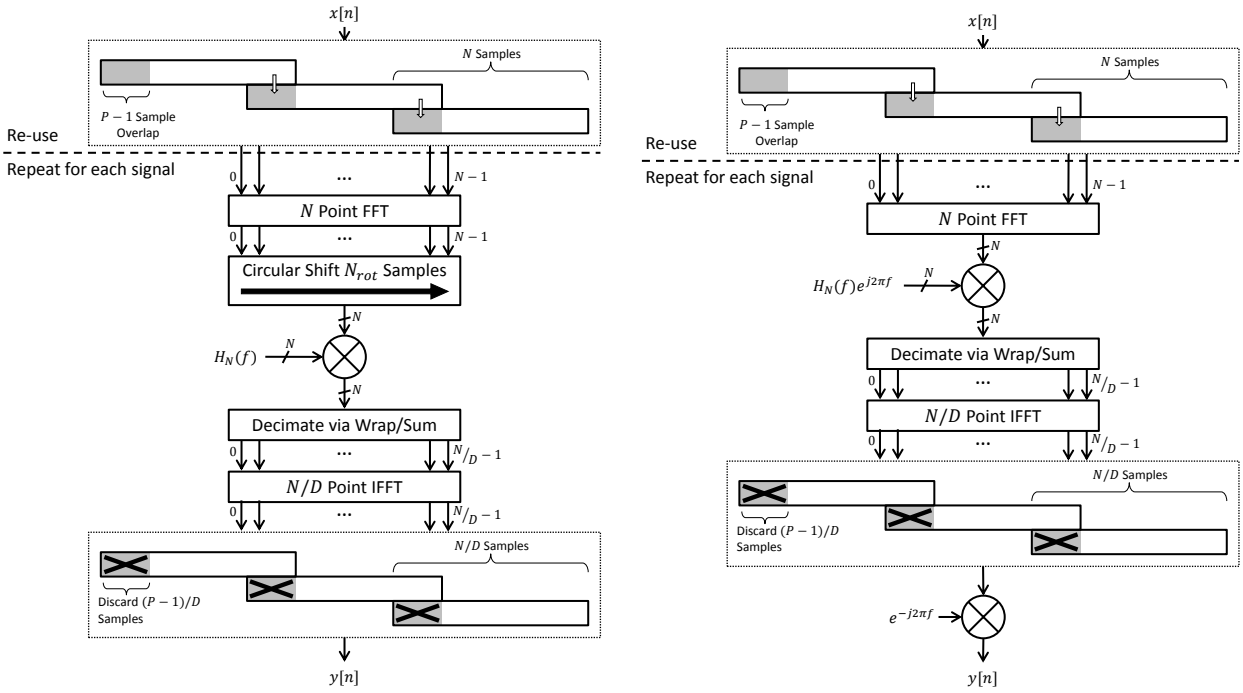
*Decimation:* Following the frequency domain filtering we have two different options for decimation. The first option is to perform a full size inverse FFT of the output, and then decimate in the time domain. The issue with this approach is that we are computing a larger inverse FFT than we really need to. Borgerding suggests decimating in the frequency domain as a solution to this. The approach (called "Wrap/Sum" in Figure 2.4) is simple: coherently add together the components of the frequency spectrum that would be aliased upon time domain decimation. For example, if the FFT size is 1024 and we need to decimate by a factor of 4, then coherently sum FFT samples 0 to 255, 256 to 511, 512 to 767 and 768 to 1024. The result is a 256 sample frequency spectrum which we can now inverse FFT to produce the

decimated output. The only trick with this approach is that we must discard just $(P-1)/D$ samples rather than $P-1$ to account for the overlap. This reveals one restriction of this structure: the filter order $P-1$ must be an integer multiple of the decimation, $D$.

### 2.3.1 Limitations

There are a few limitations of this structure that are worth mentioning. First, as mentioned earlier, the filter order $P-1$ must be an integer multiple of the decimation factor $D$. This problem is pretty easy to solve though - simply zero-pad the filter to achieve an appropriate length. Another limitation that is potentially challenging is that the FFT size, $N$, must be an integer multiple of the decimation rate, $D$. So decimation rates whose prime factors are larger than 2 or 5 (or others, depending on the FFT implementation) could lead to FFT inefficiency for this structure.

One more limitation occurs when attempting to rotate the FFT to frequency shift. As previously mentioned, the precision of the frequency shift is limited by the resolution of the FFT. However, the precision is also limited further by $V$, the ratio of FFT size to filter order. This is because we must restrict mixing to the frequencies whose period completes




(a) Performing frequency shift by circular shifting the FFT output

(b) Performing precise frequency shift in the time-domain

Figure 2.4: Overlap-Save Filter Bank Structures

in $L = N - (P - 1)$ samples. Borgerding provides the following equation for computing the number of FFT bins to rotate to shift to frequency $f$ ([**?**] Equation (1)):

$$N_{rot} = \text{round}\left(\frac{Nf}{Vf_s}\right)V \tag{2.11}$$

Where $f_s$ is the wideband sample rate. This equation simply adjusts $N_{rot}$ to the nearest multiple of $V$ bins. It is worth noting that the solution presented for making $P - 1$ an integer multiple of $D$ is only making this problem worse by making $V$ larger - but there's nothing to be done about that other than opting for a shorter filter.

### 2.3.2 Advantages

The greatest benefit of the Overlap-Save Filter Bank for this project is the ease with which it can be combined with SCD Estimation for cyclostationary detection. Since the first step for both algorithms is a forward FFT of the wideband input, if we were to design a joint cyclostationary detector/Overlap-Save Filter Bank we could re-use the same forward FFT for both algorithms.

The only design challenge is finding a combination of FFT size, sample rate, and decimation factor that will work for both algorithms.

Additionally, averaging the overlapped FFT frames together may have an effect on the accuracy of the SCD estimation, but analyzing this effect is beyond the scope of this project.

# Chapter 3

# Simulation

A MATLAB simulation of these structures has been written. Directions to find all of the source code can be found in Appendix A. Each module - testbed, cyclostationary detector, polyphase analysis/synthesis channelizer, and overlap-save filter bank - was developed and tested separately. The detector was then combined with each channelizer and their performance was evaluated. The same logical flow is followed in this Chapter as the modules are presented.

## 3.1 Testbed

Each Module is tested with the same wideband signal, shown in Figure 3.1. It is a waveform sampled at 10 MHz with three QPSK signals at different symbol rates and center frequencies:

- 312.5 ksymbols/s at -2.5 MHz
- 156.25 ksymbols/s at 0 MHz
- 625 ksymbols/s at 2.5 MHz

Note that the symbol rates were all chosen as fractions of the wideband sample rate by design, as this allows the approximation discussed in Section 2.1.2 to be used.

A simulation of a QPSK transmitter/receiver pair is used to generate the signals used in this signal, and to attempt to demodulate the output of the channelizers. This transmitter channel encodes the data with a convolutional code, packetizes the encoded data, and appends a synchronization sequence to each packet. The receiver uses a squared spectrum to correct any major frequency offsets, then uses the synchronization sequence to correct timing and

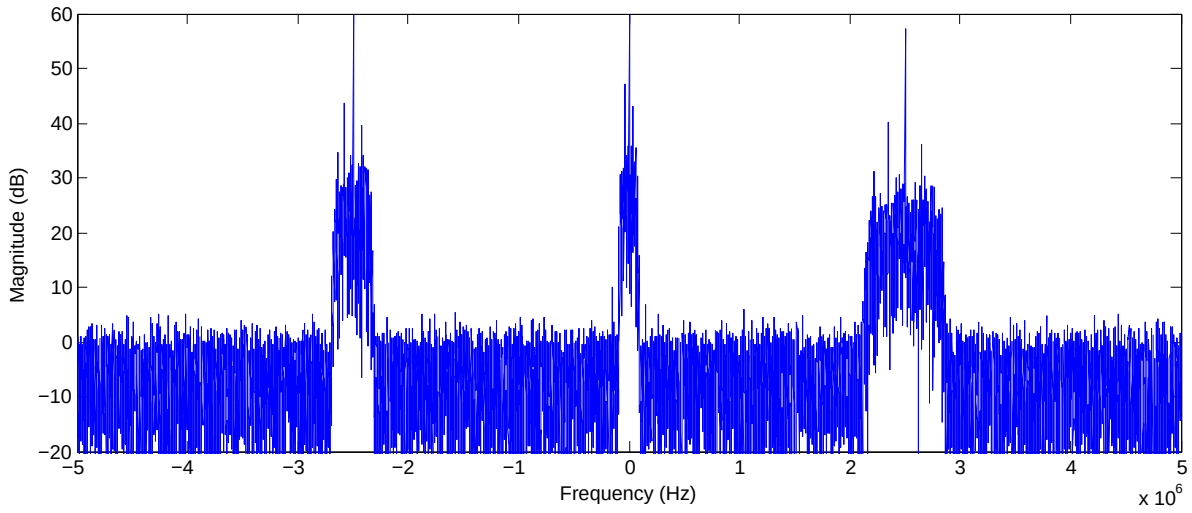



Figure 3.1: Frequency spectrum of the 10 MHz test signal used for each module

phase offsets. on a per packet basis. Finally, it performs channel decoding and outputs the demodulated bits[1].

[1]This transmitter/receiver pair is based on a project the author completed for ECE5654 along with Craig Carlson and Ruth Stoehr. The original report for this project is available at https://github.com/TheNeuralBit/cyclo_channelizer/raw/master/reference/ECE5654_Project_Report.pdf.

## 3.2 Cyclostationary Detector

The next module is the cyclostationary detector. It works by computing estimates of the SCD at particular cyclic frequencies and searching for features in those estimates. The first step, estimating the SCD, can be performed by the MATLAB function `cyclic_spectrum(...)`, which will compute an estimate of the given data at a specific cyclic frequency, $\alpha$. It accepts a few additional parameters:

**Mode:** Set to either of the techniques described in Section 2.1.2: Exact time-domain frequency shifts, or a single FFT with frequency-domain shifts. The latter is what we are primarily concerned with, for its efficiency.

**FFT Size:** Size of the FFT used to go to the frequency domain.

**Averaging:** Will average $N$ estimates together to produce a more accurate result.

According to [**?**], QPSK signals generate a large peak in the SCD at the signal's center frequency when $\alpha$ is equal to the signal baud rate. So for this application estimates are generated at cyclic frequencies corresponding to baud rates of interest. Figure 3.2 shows examples of these estimates for our test signal.

In each of these estimates it easy to detect the large, narrow peak at the center frequency of the signal with the corresponding baud rate. However, there are also other features, which agree with the theoretical SCD from [**?**]. The higher baud rate signals generate wide peaks in the lower cyclic frequency estimates, like the peak at 2.5 MHz in Figure 3.2b. These are easy to distinguish from the main peak with the human eye since they are much wider, but distinguishing them computationally is more challenging. The lower baud rate signals also generate features in the higher cyclic frequency estimates, in the form of two, much smaller peaks, straddling the actual center frequency. Examples of this are visible at 0 MHz in Figure 3.2c and 3.2d. These features are easily filtered out by either a human or a computer with a simple threshold.

The second portion of the cyclostationary detection must use these SCD estimates to generate a list of detected frequencies, and the corresponding signal's baud rate. In order to do so the following algorithm has been devised. It accepts the input wideband data, and a list of potential baud rates that are of interest.

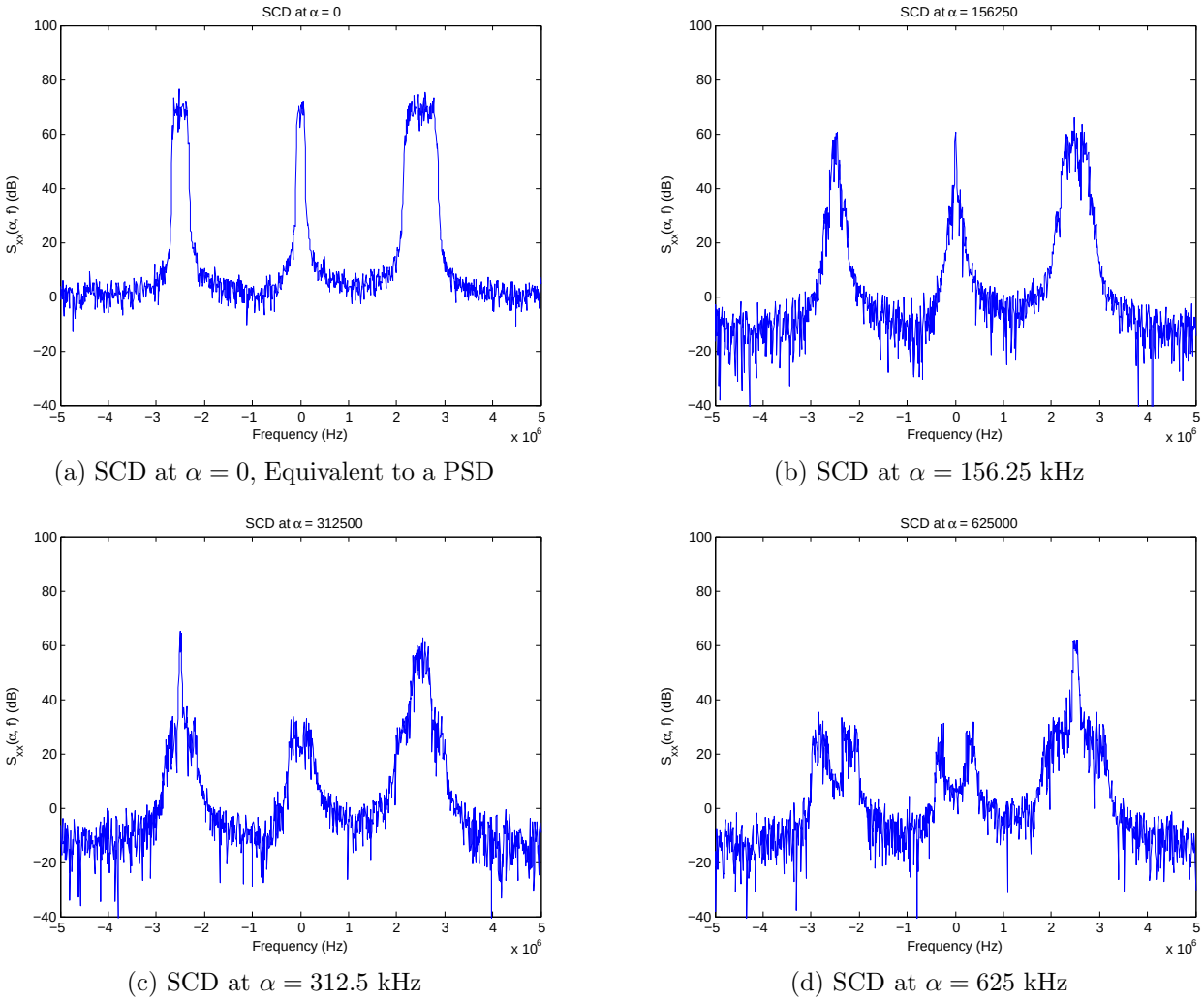
1. Initialize two empty lists `f[]` (center frequencies) and `b[]` (corresponding baud rate for each frequency)
2. Sort list of potential baud rates from lowest to highest
3. Estimate SCD at $\alpha$ equal to the first (lowest) baud rate

4. Detect peaks in the SCD. Use a threshold and a minimum peak separation to filter out false alarms.

5. For each detected peak, check if an entry already exists in `f[]` near that frequency. If so, replace it with this peak's frequency and replace the corresponding entry in `b[]` with $\alpha$. Otherwise, append a new entry to `f[]` and `b[]` with the same values.

6. Repeat from 3 using the next baud rate in the list

This is a very simple algorithm, but it works well for this test. It is fully implemented in the `cyclostationary_detect(...)` function. This approach will essentially select the highest symbol rate that appears to be active at a given center frequency.




(a) SCD at $\alpha = 0$, Equivalent to a PSD

(b) SCD at $\alpha = 156.25$ kHz

(c) SCD at $\alpha = 312.5$ kHz

(d) SCD at $\alpha = 625$ kHz

Figure 3.2: Estimates of the SCD at various cyclic frequencies. FFT size = 1024, Averaged over 10 FFTs.

In an actual application a more robust approach should be used. One possibility which could be investigated but is beyond the scope of this project would be to use a morphological filter to isolate only very narrow peaks. Between that and the thresholding, other features would be easily ignored. For this project the simple approach was taken, because the primary focus is on efficiently estimating the SCD.

There is a major limitation to the implementation of this approach which is important to note. The SCD estimates are generated only based on the beginning of the input signal. This means that signals are detected based on only the first $N_{FFT}N_{Avg}$ samples. This works well in this application since the test signals are on at the beginning of time and continue until there is no more data to transmit, but in reality signals are going up and down all the time. In an actual application the cyclostationary detector should be run continuously to detect when signals come up and go down.

## 3.3 Polyphase Analysis/Synthesis Channelizer

The simulation of the polyphase analysis channelizer is broken up into three major parts: `analysis_channelizer(...)`, `synthesis_channelizer(...)`, and `polyphase_channelizer(...)`. The latter of these combines the first two to implement the flexible channelizer described in Section 2.2.1.

The implementation of `analysis_channelizer(...)` accepts time domain input data and a number of channels to produce, and outputs that number of channels in the time domain. No additional configuration is necessary. A low-pass filter with cut-off at the output's nyquist frequency is designed and used. The output is indexed such that the the frequency of the channel at index $k$ is given by:

$$f_k = -\frac{f_s}{2} + k\frac{f_s}{D},\ k = 1 \dots D \tag{3.1}$$

Where k follows the standard indexing for MATLAB arrays.

`synthesis_channelizer(...)` is designed to perfectly complement `analysis_channelizer(...)`, so that if it were passed the list of channelized data produced by the synthesis channelizer as an argument, it would produce a replica of the original input. Internally, it designs and uses a low-pass filter with cutoff at the input's nyquist frequency, so that an analysis and synthesis channelizer of the same size will use the same filter.

Finally, `polyphase_channelizer(...)` utilizes both of these functions to implement a flexible channelizer structure. It accepts the input data, its sample rate, and two additional configuration parameters:

**Frequencies:** A list of the center frequencies of the signals of interest

**Decimations:** A list of the decimation factors to be used for each corresponding center frequency. Every decimation factor must be a factor of the highest decimation factor.

The simulation will first create a synthesis channelizer with decimation equal to the highest requested decimation (Step 1 from Figure 2.3), an example of the output of this step for our test signal is shown in Figure 3.3. For all requested outputs with this decimation factor, the nearest output of the analysis channelizer will be selected and frequency shifted so that the desired center frequency is at baseband, and passed through. For every remaining output, a synthesis channelizer will be produced to combine the channels surrounding that frequency (Step 2) and a complex phasor will shift the desired center frequency down to baseband (Step 3).

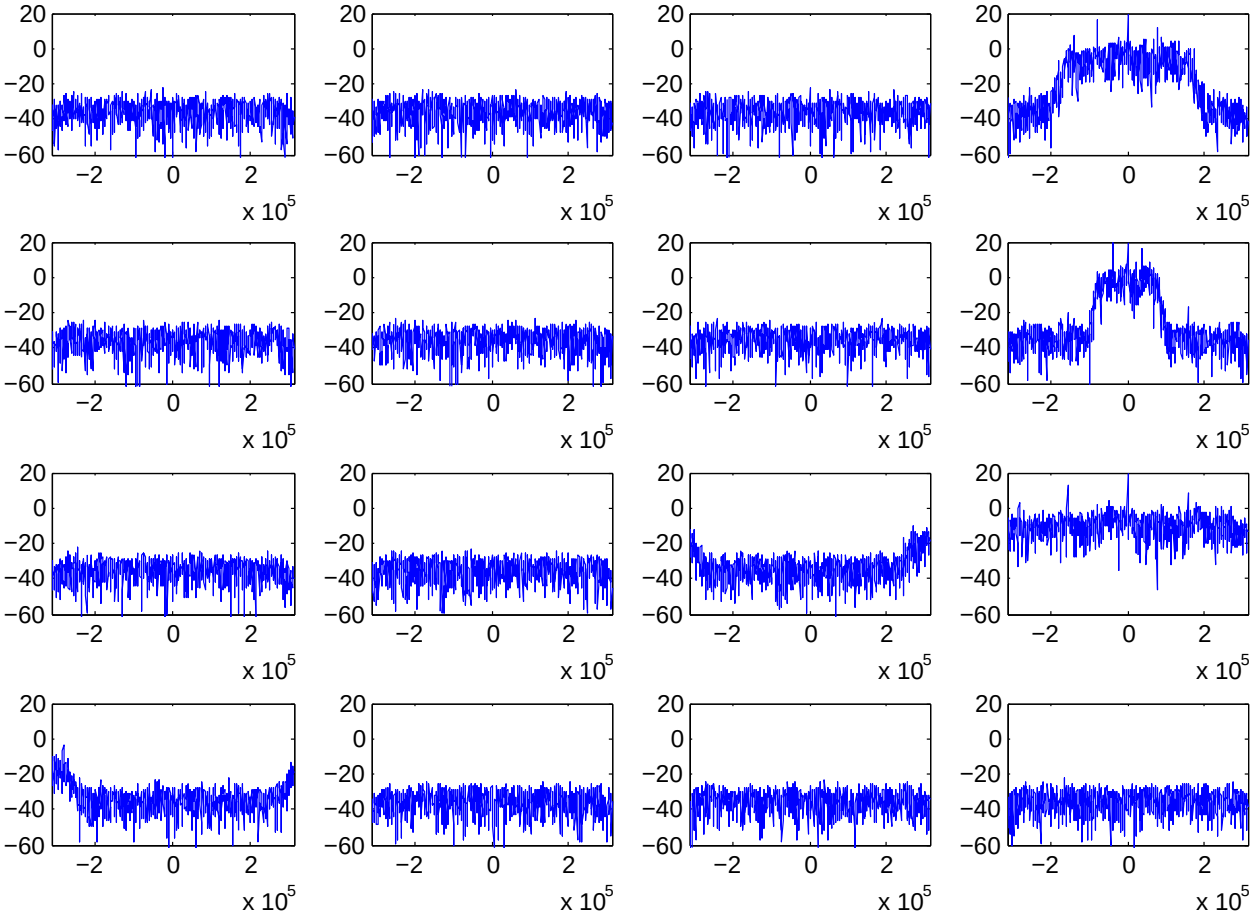



Figure 3.3: Output of a 16 channel polyphase analysis channelizer operating on the test signal. X Axes are frequency in Hz, Y Axes are magnitude in dB.

The major limitation of this implementation is that certain frequencies at the highest decimation factor *cannot* be produced. Any signal at the lowest output sample rate with center frequency at the boundary between two outputs of the analysis channelizer will not be chan-

nelized correctly. A solution to this would be to force the analysis channelizer to use a higher decimation factor than any of the requested outputs. However there are certain trade-offs here: it increases the computational complexity, since a narrower filter is required, and every output is separated into more parts before being combined which could harm the signal integrity.

### 3.3.1 Adding Cyclostationary Detection

As discussed in Section 2.2.2, there is no way to efficiently combine this polyphase channelizer structure with wideband SCD estimation. Thus, this simulation simply uses an unmodified version of the cyclostationary detector discussed previously to direct an unmodified polyphase channelizer.

This function, `cyclostationary_and_polyphase(...)` accepts the input data and performs detection and channelization. The detector can be configured with all the same parameters discussed in Section 3.2, and outputs a list of center frequencies and corresponding symbol rates. The symbol rates are converted to decimation factors based on the desired number of samples per symbol, specified by the user. The list of frequencies and corresponding decimation factors are then passed to the channelizer.

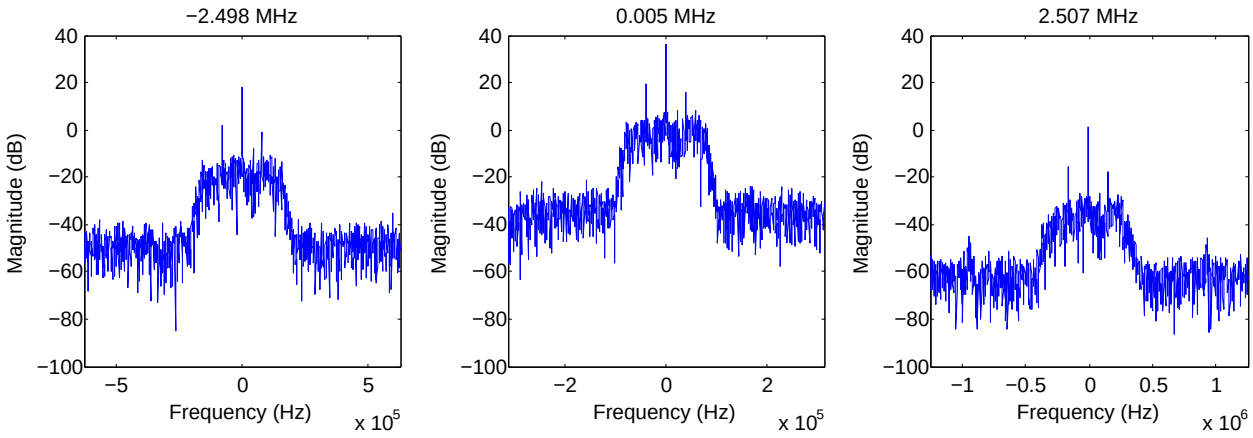



Figure 3.4: Output of the combined cyclostationary detector and polyphase channelizer. All three test signals sampled at four samples per symbol.

Figure 3.4 shows the output of this simulation when run on our test signal. In this case the channelizer was configured to output each signal at four samples per symbol. Note the small peaks in the 2.5 MHz signal around $\pm 1$ MHz - these are not present in the original signal, they are produced from aliasing in the analysis channelizer, which can be seen in the corresponding outputs in Figure 3.3. Regardless, the high SNR of these outputs allowed every channel to be demodulated with a 0% BER.

## 3.4 Overlap-Save Filter Bank

The final module is the Overlap-Save Filter Bank. The Filter Bank is implemented with tuning performed in the frequency domain by circular shifting the FFT, as shown in Figure 2.4a. The simulation cannot be configured to tune in the time domain at this time.

The channelizer is fully implemented in `overlap_save_channelizer(...)`. The function accepts a list of center frequencies and a list of corresponding decimation factors, just like the polyphase channelizer, but it also allows the user to specify the FFT size to be used internally.

The first step performed in this function is determining the value of the variable $P$. This corresponds to the length of the filter used, and determines the amount of overlap in each FFT. $P$ must be greater than or equal to the length of the longest filter and $P-1$ must be a factor of the FFT size. A different filter must be generated for each unique decimation factor[2], and the function assumes that the longest filter response will be for the narrowest filter - the filter for the largest decimation factor. So in order to determine $P$, the function first determines the length of the filter for the largest decimation factor, then rounds up until $P-1$ is the nearest factor of the FFT size. Every other filter will be zero-padded to this length.

Next, the FFTs are computed (overlapped by $P-1$ samples), and they are re-used to tune, filter and decimate each channel, as described in Section 2.3

### 3.4.1 Adding Cyclostationary Detection

As discussed in Section 2.3.2 a major advantage to the Overlap-Save Filter Bank is that the Forward FFTs can be re-used to estimate the SCD. This is implemented in the simulation function `cyclostationary_and_overlap_save(...)`. This function uses two parts of the overlap-save channelizer separately. First it uses `os_fft(...)` to determine $P$ and compute overlapped FFTs, then it uses these FFTs to perform cyclostationary detection, and finally it re-uses the overlapped FFTs, along with the detection information in `os_filter(...)` to tune, filter, and decimate each detected channel.

Note that this process uses the same `cyclostationary_detect(...)` function described in Section 3.2, but it had to be modified to accept either time-domain sampled data or frequency domain data. If it receives time-domain data it performs an FFT to enter the frequency domain, but if it receives frequency domain data it skips that step.

Decimation factors are determined to output a user configured number of samples per symbol, just as in the Polyphase Channelizer.

[2]The filters used by the overlap-save filter bank are designed in the same manner as the polyphase channelizer - with a cutoff at the nyquist frequency of the output sample rate.

Figure 3.5 shows the output of this filter bank. The high SNR of these outputs meant that every channel was demodulated with a 0% BER.

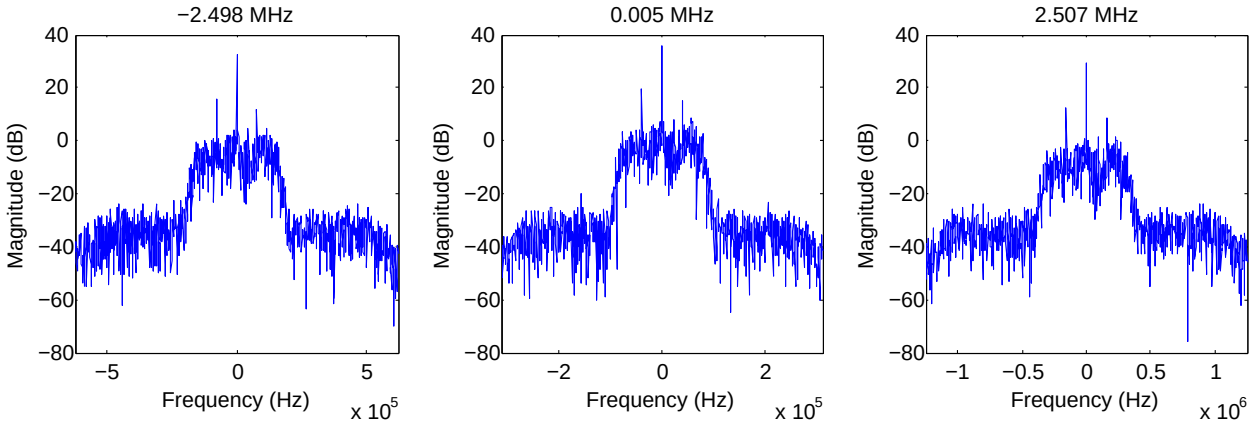



Figure 3.5: Output of the combined cyclostationary detector and overlap-save filter bank. All three test signals sampled at four samples per symbol.

# Chapter 4

# Conclusion

Simulations of a cyclostationary detector and two separate channelizer structures, a combination polyphase analysis/synthesis channelizer and an overlap-save filter bank, have been presented. Both channelizer structures were combined with the cyclostationary detector in order to evaluate their performance when detecting and isolating QPSK signals at various symbol rates in a wideband test signal. Both structures proved effective at accurately detecting and isolating these signals, but only the Overlap-Save Filter Bank was able to be combined with detection to improve computational efficiency.

Currently these simulations only uses the beginning of the test signal for detection, but in an actual implementation the detector would be running continuously and dynamically directing the channelizer to begin processing new signals. It should be noted that in this described system, both channelizers would need to have knowledge of the required signal bandwidths and output sample rates before processing begins, so that appropriate filters can be designed, and in the case of the polyphase analysis/synthesis channelizer, so that the size of the analysis channelizer can be chosen.

## 4.1 Future Work

This work could be extended in many different ways. In this project, the limitations of the various structures were taken as a given, however many of these limitations can actually be avoided. For example, Harris shows that it is in fact possible to adjust the output sample rate of the analysis channelizer by adjusting the input commutator [**?**]. That topic was left out of this simulation since it is difficult to devise a flexible version of it that will work for any sample rate. Future work could focus on what output sample rates are possible and devise an approach for a general purpose solution.

Another limitation taken as a given is that SCD estimation with a single FFT is impossible

for cyclic frequencies that do not satisfy Equation 2.1.2 - but this could potentially be overcome as well. For cyclic frequencies, where $\alpha/2$ does not correspond to a specific FFT bin the FFT could be interpolated before shifting. This would not be perfectly accurate, but it would be worth evaluating how close of an approximation this yields. Additionally, one could evaluate how complex of an interpolation could be used while still maintaining the computational advantage of using a single FFT.

The only presented test case was very low noise - it would be beneficial to evaluate how well the detector and channelizers work as a function of noise power. A Monte Carlo test could be performed to evaluate the detector's probability of detection, $P_d$, vs. noise power at a given threshold. Both channelizers should also be evaluated by plotting the Bit Error Rate (BER) of every output channel as a function of noise power. Perhaps we will find that one of the channelizers harms the signal integrity.

Finally, it would be useful to do an in-depth analysis of the computational complexity of both combined detector/channelizers. The theoretical computational complexity could be determined in addition to an analysis of simulation runtime.

# Appendix A

# Project Source

The full source is hosted on GitHub (https://github.com/TheNeuralBit/cyclo_channelizer). Some of the critical files are included in this appendix.

## A.1 Cyclostationary Detection

### A.1.1 cyclostationary_detect.m

```
function [output_freqs, bauds] = cyclo_detect(data, bauds_to_check, threshold, peak_distanc
% cyclo_detect - Perform cyclostationary detection on input data vector
%       data           - 1D vector of time domain data. Wideband signal to
%                        perform detection on.
%       bauds_to_check - 1D vector of floats. List of baud rates to search for
%       threshold      - Threshold, in dB, for peak search
%       peak_distance  - Minimum peak separation, in Hz
%       nfft           - FFT size
%       f_s            - sample rate of data

    configuration;

    output_freqs = [];
    bauds = [];

    if size(data, 1) == nfft
        cyc_fft = data;
    else
        cyc_fft = compute_cyclo_fft(data, nfft);
    end

    % Truncate to a multiple of the averaging factor
```

```
    num_ffts = floor(size(cyc_fft, 2)/CYCLO_AVERAGING)*CYCLO_AVERAGING;
    cyc_fft = cyc_fft(:, 1:num_ffts);

    % Perform averaging
    cyc_fft = squeeze(mean(reshape(cyc_fft, nfft, CYCLO_AVERAGING, []), 2));

    freqs = linspace(-f_s/2, f_s/2, nfft);
    spec = zeros(length(bauds), 1);
    for idx = 1:length(bauds_to_check)
        baud = bauds_to_check(idx);
        spec = single_fft_cyclo(cyc_fft, baud, f_s);
        % Only used the first averaged SCD estimate for detection
        spec = spec(:,1);
        if DEBUG_FIGURES
            figure;
            plot(linspace(-f_s/2, f_s/2, nfft), 10*log(abs(spec)));
            title(sprintf('SCD at \\alpha = %d', baud));
        end
        [pks, locs] = findpeaks(abs(spec), 'MinPeakHeight', threshold, ...
                                           'MinPeakDistance', round(peak_distance*baud/(f_s
        % Iterate through the peaks
        % If one of them is close to a peak we already found, overwrite that one
        % with the new frequency and baud rate
        % Otherwise add the new frequency and baud rate
        for loc = locs'
            freq = freqs(loc);
            [val, nearest_idx] = min(abs(output_freqs - freq));
            if length(output_freqs) > 0 && val < peak_distance
                output_freqs(nearest_idx) = freq;
                bauds(nearest_idx) = bauds_to_check(idx);
            else
                output_freqs = [output_freqs, freq];
                bauds = [bauds bauds_to_check(idx)];
            end
        end
    end
end
```

### A.1.2 cyclic_spectrum.m

```
function [cyc_spec] = cyclic_spectrum(data, alpha, fft_size, F_S, method, averaging)
% cyclic_spectrum - Estimate SCD at a given alpha using one of two methods
%       data            - 2D vector of frequency data. Result will be averaged
%                         across the time dimension
%       alpha           - cyclic frequency to compute
%       fft_size        - FFT size
%       F_S             - sample rate of data that f_data was generated from
%       method          - either 'one_fft' (frequency shift in frequency domain
%                         using circular shift of the FFT), or 'freq_shift'
```

```
%                         (frequency shift in the time domain, requires two
%                         FFTs)
%       averaging       - Number of FFT frames to average together

    if nargin < 6
        averaging = 1;
    end

    if strcmp(method, 'freq_shift')
        data_reshape = reshape(data(1:fft_size*averaging), fft_size, averaging);
        t = 0:1/F_S:(fft_size*averaging - 1)/F_S;
        t = reshape(t, fft_size, averaging);
        d_right = data_reshape.*exp( j*2*pi*alpha/2*t);
        d_left  = data_reshape.*exp(-j*2*pi*alpha/2*t);
        cyc_spec = mean(fftshift(fft(d_left, [], 1), 1).*...
                        conj(fftshift(fft(d_right, [], 1), 1)), 2);
    elseif strcmp(method, 'one_fft')
        cyc_fft = compute_cyclo_fft(data, fft_size);

        % Truncate to a multiple of the averaging factor
        num_ffts = floor(size(cyc_fft, 2)/averaging)*averaging;
        cyc_fft = cyc_fft(:, 1:num_ffts);

        % Perform averaging
        cyc_fft = squeeze(mean(reshape(cyc_fft, fft_size, averaging, []), 2));

        cyc_spec = single_fft_cyclo(cyc_fft, alpha, F_S);
    else
        cyc_spec = 0;
    end
end
```

## A.2 Polyphase Channelizer

### A.2.1 cyclostationary_and_polyphase.m

```
function [channels output_f_s freqs] = cyclo_and_polyphase(data, bauds_to_check, samps_per
% cyclo_and_polyphase - Combined cyclostationary detector and Polyphase
%                       Analysis/Synthesis Channelizer
% Input: data           - 1D vector of time domain data. Wideband signal to
%                         perform detection on and filter.
%        bauds_to_check - 1D vector of floats. List of baud rates to search for.
%        samps_per_sym  - Desired number of samples per sybol in each output
%                         channel
% Output: output        - 1xD cell array of time domain data for each channel.
%                         item at index k has center frequency -fs/2 + kfs/D
%         output_f_S    - array of sample rates for each output
```

```
%         freqs         - list of center frequencies for each output
    configuration;
    threshold = CYCLO_PEAK_THRESH;
    min_spacing = CYCLO_PEAK_MIN_SPACING;
    fft_size = 1024*2;

    [freqs, bauds] = cyclo_detect(data, bauds_to_check, threshold, min_spacing, fft_size, I

    % TODO: Warning if decimations not actually close to integer
    decimations = round(F_S./bauds./samps_per_sym);
    channels = polyphase_channelizer(data, freqs, decimations, F_S);
    output_f_s = F_S./decimations;
end
```

### A.2.2 polyphase_channelizer.m

```
function [output] = polyphase_channelizer(data, freqs, decimations, F_S)
% polyphase_channelizer - Use combined polyphase analysis/synthesis channelizer
%                         to output channels at each desired center frequency
% Input:  data        - 1D vector of time domain data. Wideband signal to
%                       be filtered.
%         freqs       - List of output center frequencies
%         decimations - List of corresponding decimation factors
%         F_S         - input sample rate
% Output: output      - 1xD cell array of time domain data for each channel.
%                       item at index k has center frequency -fs/2 + kfs/D
    configuration;

    output = cell(length(freqs), 1);

    D = max(decimations)*2;
    split_channels = analysis_channelizer(data, D);
    split_f_s = F_S/D;
    if DEBUG_FIGURES
        plot_channels(split_channels, repmat(split_f_s, D, 1), ...
                      'AxisLabels', 0, 'YMin', -60, 'YMax', 20);
    end

    for i=1:length(freqs)
        freq = freqs(i);
        dec = decimations(i);
        out_f_s = F_S/dec;
        if dec == D
            output{i} = split_channels{round(D*(freq/F_S + 0.5))};
        else
            nearest_bin = round(D*(freq/F_S + 1.0/2 + 1.0/2/D));
            num_bins = D/dec;
            min_bin = nearest_bin - num_bins/2;
            max_bin = nearest_bin + num_bins/2 - 1;
```

```
            output{i} = synthesis_channelizer(split_channels(min_bin:max_bin));

            % Compute offset
            nearest_freq = nearest_bin*split_f_s - F_S/2 - split_f_s/2;
            f_off = freq - nearest_freq;

            % Perform frequency shift
            t = 0:(1/out_f_s):((length(output{i})-1)/out_f_s);
            output{i} = output{i}.*exp(1i.*2.*pi.*(-f_off - split_f_s/2).*t);
        end
    end
end
```

## A.3 Overlap-Save Channelizer

### A.3.1 cyclostationary_and_overlap_save.m

```
function [channels output_f_s freqs] = cyclo_and_overlap_save(data, bauds_to_check, samps_p
% cyclo_and_overlap_save - Combined cyclostationary detector and OS Filter Bank
% Input: data           - 1D vector of time domain data. Wideband signal to
%                         perform detection on and filter.
%        bauds_to_check - 1D vector of floats. List of baud rates to search for.
%        samps_per_sym  - Desired number of samples per sybol in each output
%                         channel
% Output: output        - 1xD cell array of time domain data for each channel.
%                         item at index k has center frequency -fs/2 + kfs/D
%         output_f_S    - array of sample rates for each output
%         freqs         - list of center frequencies for each output
    configuration;
    threshold = CYCLO_PEAK_THRESH;
    min_spacing = CYCLO_PEAK_MIN_SPACING;
    fft_size = 1024*2;


    potential_decimations = F_S./bauds_to_check./samps_per_sym;
    [data_fft, P, V] = os_fft(data, potential_decimations, fft_size);
    [freqs, bauds] = cyclo_detect(fftshift(data_fft, 1), bauds_to_check, threshold, min_spa
    decimations = F_S./bauds./samps_per_sym;
    channels = os_filter(data_fft, freqs, decimations, F_S, fft_size, P, V);
    output_f_s = F_S./decimations;
end
```

### A.3.2 overlap_save_channelizer.m

```
function [output] = overlap_save_channelizer(data, freqs, decimations, F_S, fft_size)
```

```
% overlap_save_channelizer - Use overlap-save filter bank to output channels at
%                             each desired center frequency
% Input:  data        - 1D vector of time domain data. Wideband signal to
%                       be filtered.
%         freqs       - List of output center frequencies
%         decimations - List of corresponding decimation factors
%         F_S         - input sample rate
%         fft_size    - FFT size
% Output: output      - 1xD cell array of time domain data for each channel.
%                       item at index k has center frequency -fs/2 + kfs/D
    [data_fft, P, V] = os_fft(data, decimations, fft_size);
    output = os_filter(data_fft, freqs, decimations, F_S, fft_size, P, V);
end
```

### A.3.3 os_fft.m

```
function [data_fft, P, V] = os_fft(data, decimations, fft_size)
% os_fft - Compute overlapped FFTs for use in Overlap-Save Filter Bank
% Input:  data        - 1D vector of time domain data. Wideband signal to
%                       be filtered.
%         decimations - List of the decimation factors that will be used in
%                       follow-on processing.
%         fft_size    - FFT size
%
% Output: data_fft    - Overlapped FFT data
%         P           - Filter length. Equal to the length of the filter for
%                       the largest decimation factor, rounded up to the
%                       closest divisor of FFT size
%         V           - Ratio of FFT size to P-1
    dec_lcm = lcms(decimations);
    %dec_lcm = 32;

    output = cell(length(decimations), 1);

    longest_filt = design_filter(max(decimations));
    P = length(longest_filt);

    % TODO: allow frequency shifting in EITHER time or frequency
    % If we are frequency shifting in the time domain then
    % we only need to ensure that P is a multiple of the decimation
    %P = P + dec_lcm - (mod((P - 1) - 1, dec_lcm) + 1);

    % If shifting in the frequency domain we need to be a litle more
    % ensure that V is an integer
    P = ldf(P-1, fft_size) + 1;

    V = round(fft_size/(P - 1));

    % Compute FFT
```

```
    data_fft = fft(buffer(data, fft_size, (P - 1)), [], 1);
end

function [rtrn] = ldf(k, n)
    % ldf returns the lowest divisor of n that is greater than k
    if mod(n, k) == 0
        rtrn = k;
    elseif 2*k > n
        rtrn = n;
    else
        rtrn = ldf(k+1, n);
    end
end
```

### A.3.4 os_filter.m

```
function [output] = os_filter(data_fft, freqs, decimations, F_S, fft_size, P, V)
% os_filter - Second stage of Overlap-Save Filter Bank
% Input:  data_fft    - Overlapped FFT data. From os_fft.
%         freqs       - List of output center frequencies
%         decimations - List of corresponding decimation factors
%         fft_size    - FFT size
%         P           - Filter length. Equal to the length of the filter for
%                       the largest decimation factor, rounded up to the
%                       closest divisor of FFT size. From os_fft.
%         V           - Ratio of FFT size to P-1. From os_fft.
% Output: output      - 1xD cell array of time domain data for each channel.
%                       item at index k has center frequency -fs/2 + kfs/D
    for idx = 1:length(freqs)
        freq = freqs(idx);
        decimation = decimations(idx);

        %% DESIGN THE FILTERS
        disp('Designing the filter...')
        filt_time = design_filter(decimation);
        % Zero pad so that filter is length P-1
        if length(filt_time) < P
            filt_time = [filt_time zeros(1, P - length(filt_time))];
        end
        filt_fft = fft(filt_time, fft_size);

        % Rotate FFT to desired center freq
        disp('Rotating FFT...')
        num_bins = -round(fft_size*freq/F_S/V)*V;
        rot_fft = circshift(data_fft, num_bins, 1);

        % Filter FFT
        disp('Filtering FFT...')
        filt_data_fft = zeros(size(rot_fft));
```

```
        for i = 1:size(rot_fft, 2)
            filt_data_fft(:,i) = filt_fft'.*rot_fft(:,i);
        end
        %rot_fft.*repmat(filt_fft', size(rot_fft, 1));

        disp('Decimating and Returning to Time Domain...')
        inv_fft = ifft(filt_data_fft, [], 1);
        tmp = buffer(inv_fft(:), fft_size - (P - 1), -(P - 1));
        tmp = tmp(:).';
        output{idx} = tmp(1:decimation:end);
    end
end
```