BLIND FRONT-END PROCESSING OF DYNAMIC MULTI-CHANNEL WIDEBAND SIGNALS

by

Kevin Jackson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Jacob Gunther Major Professor	Dr. Todd Moon Committee Member
Dr. Don Cripps Committee Member	Mark McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2016

Abstract

Blind Front-End Processing of Dynamic Multi-Channel Wideband Signals

by

Kevin Jackson, Master of Science

Utah State University, 2016

Major Professor: Dr. Jacob Gunther
Department: Electrical and Computer Engineering

In wireless digital communications, the sender and receiver typically know the modulation scheme with which they will be communicating. Automatic modulation identification is the ability to identify the modulation in a communication system with little to no prior knowledge of the modulation scheme. Many techniques for modulation identification operate on many assumptions including that the input signal is base-banded, the carrier frequency is known and that the signal is narrow-band (i.e. neighboring signals in the wide-band are excluded). This work provides the blind processing of an arbitrary wide-band signal to allow such assumptions. The challenges of such a front-end or pre-processor include detecting signals which can appear at any frequency, with any band-width at any given time and for any arbitrary duration. This work takes as its input a wide-band signal with a random number of sub-signals, each turning on and off at random times and each at random locations in the frequency domain. The output of the system is a collection of signals corresponding to each sub-signal brought down to base-band, isolated in the frequency and time domains, nominally sampled and with estimates of key parameters.

(61 pages)

Public Abstract

Blind Front-End Processing of Dynamic Multi-Channel Wideband Signals

by

Kevin Jackson, Master of Science

Utah State University, 2016

Major Professor: Dr. Jacob Gunther
Department: Electrical and Computer Engineering

Society today relies heavily on the exchange of information via wireless communication. Such communication provides the means of transferring information from point A to point B over the air. In order to accomplish this, a minimum of two devices is required: a transmitter and a receiver. Just as people must speak the same language in order to communicate, the receiver must know how the transmitter communicates in order to make sense of the data being transmitted. For multiple wireless communication system to coexist, they must each speak a unique language, which in the communications world is referred to as modulation. Modulation techniques have a broad spectrum of complexity. This work studies a set of techniques that allow a receiver to identify and adapt to a variety of modulation schemes so as to allow the receiver to eventually communicate with a wide variety of transmitters without know beforehand the languages they speak.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Wireless Communication

Wireless communication is a key asset in modern society. Being in the age of information, we rely heavily on dependable means of sending and receiving information. Wireless communication has many advantages as well as disadvantages.

Many modern devices including satellites, GPS and cell phones could not exist without it. Modern conveniences including Wi-Fi, Bluetooth and AM/FM radio could not exist without it. Wireless communication impacts several spheres. In the military, coordinated operations need wireless communication. The current infrastructure of phone lines and other hard-wire communications are relieved due to cell phones, satellites, radios, etc. Because of the many devices capable of wireless communication, the frequency spectrum is overly congested. The allocation of frequency bands (regions of the frequency spectrum) is managed by government agencies, but users continually run into issues involved with channel sharing. Challenges include, interference, noise drown-out, timing skew, frequency spectrum congestion, etc. A lot of these challenges are overcome via signal processors right after the receiver antenna. Others are eliminated via modulation.

1.1.1 Modulation

Fundamental modulation involves combining the signal from the transmitter with a pure sinusoidal at a designated frequency (called the carrier frequency). This moves all the signal activity previously present at low (or zero) frequency from the transmitter to the carrier frequency. The bandwidth does not change as a result of this operation. It is the transmitter's responsibility to ensure the bandwidth is constrained within its allocation.

The frequency spectrum is allocated and managed by governments to help protect society and also manage resources. For efficiency, many transmitters will use other “modulation” techniques on top of the fundamental so as to use their channel efficiently. More often than not, the term *modulation* is referring to this process rather than the fundamental process of shifting the signal up to the carrier. Common techniques for minimizing bandwidth usage include frequency modulation (FM) and amplitude modulation (AM). The receiver must know the modulation scheme being used in order to correctly demodulate the incoming signal.

1.2 Automatic Modulation Identification

There has been extensive research in automatic modulation identification or classification (AMC), which enables a receiver to automatically identify the modulation scheme of an incoming signal. Much of the existing research assumes most of the front end processing has already been preformed [2, 3]. Or in the case of [4, 5] there is a human in the loop assisting the front-end processor. Furthermore, as in [6] and many others, the modulation schemes considered are only linear ones. This work seeks to relax a lot of the assumptions including supporting any modulation type (linear and non-linear) that yields symmetry in the frequency domain and to provide a very generic front-end processor.

1.3 Problem Statement

Many challenges exist for blind front-end processing of a wide-band signal like that in figure 1.1 in which signals turn on and off spontaneously at arbitrary frequencies, each with different bandwidths and time durations. These challenges include detecting each sub-signal which in figure 1.1 appear as shaded rectangles in contrast with the background representing the random noise. Once detected, each sub-signal’s key features need to be accurately estimated, which include the start and stop times, the center frequencies (carrier frequencies) as well as the bandwidths. This area of research further requires approximations of the sampling rate so as to provide downstream modules with a set of signals that are of the same sampling rate. In the future, because this system also estimates the signal-to-noise ratio (SNR), blind equalization would be more than helpful in order to provide output signals of equal power level to eliminate clipping and other adverse effects from input signals of varying energy magnitudes. To ensure a more accurate sampling rate as its output, the system must remove any carrier frequency offsets that are intrinsic to all received digital signals. This signal conditioning is referred to as front-end processing or pre-processing.

An example input signal on the wide-band receiver may have a spectrogram (frequency vs. time) that looks something like that in figure 1.1. The sub-signals within a single horizontal slice (i.e. channel) are called bursts or a bursty signal when referring to the entire channel.

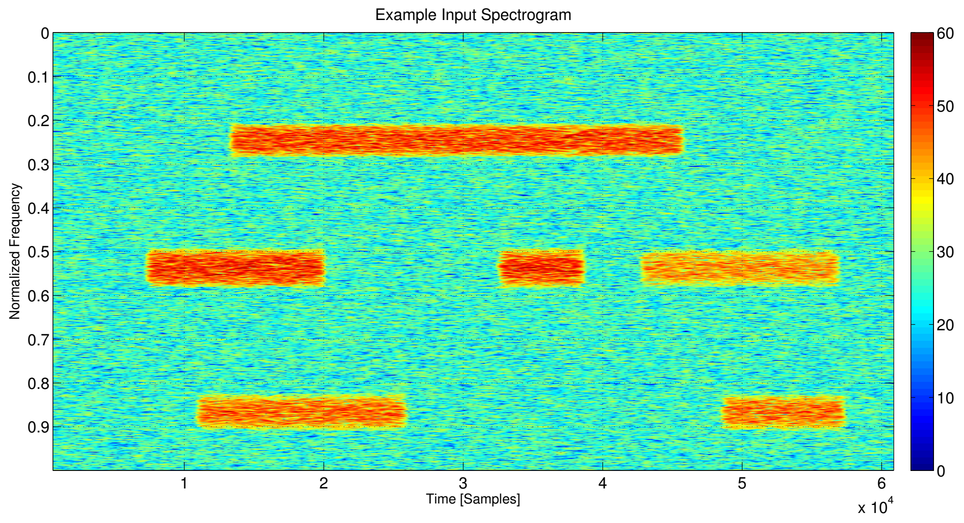

Fig. 1.1: Spectrogram of Example Input Signal

The goal of this front-end or pre-processor is to isolate each burst, clean it up and pass it on to an AMC unit (being developed by a third-party) and to do it in real-time for an arbitrarily long wide-band signal with very little prior knowledge of the incoming signal.

1.4 Solution Design Overview

The overall goal of this project is to take a wide-band signal stream and extract spectral regions that contain signal and clean them up as if the front-end processor were specifically designed for that frequency region, sampling rate and expected SNR and to do all this in real-time for an infinite data stream. Isolating narrow-band signals in both time and frequency requires two separate but similar threshold detection algorithms. No other pre-processing can be done on each burst until the isolation is completed as seen in figure 1.2. Because there is no prior knowledge of the frequencies at which any given signal may appear nor is there any prior knowledge of the bandwidths, the threshold detection algorithm merely provides a coarse estimate of the center or carrier frequency and coarse estimates of the frequency edges heretofore referred to as start and stop frequencies or high (f_h) and low (f_l) frequencies. Given these coarse estimates, the system can bring the spectral region of interest (ROI) down to approximately base-band and lowpass filter most of the spectral regions outside of the ROI and subsequently coarsely down-sample to magnify the ROI in the frequency domain. With rough estimates of the times in which each signal turns on and off, collected data can subsequently be truncated to almost exclusively include only samples that have the burst. Therefore, the result of the first block in figure 1.2 is a collection of finite sub-signals (e.g. the stark rectangles in figure 1.1) that are carved out in both time and frequency—each of which are coarsely base-banded (shifted in frequency to DC) and down-sampled. To more finely correct each signal, this set of signals are each analyzed for offset from DC (base-band) and more finely centered at the zero frequency. This step is indicated by the second block in the processing chain in figure 1.2. After the carrier frequency correction, the system estimates the symbol rate for each detected signal and subsequently re-samples to some nominal sampling rate. With each detected signal being more closely centered at DC, lowpass filtered for isolation in frequency, truncated for isolation in time and sampled at a known sample rate, the signals are deemed sufficient for further processing such as automatic modulation classification (AMC). An added feature for this system is the estimation of the SNR as seen in figure 1.2.

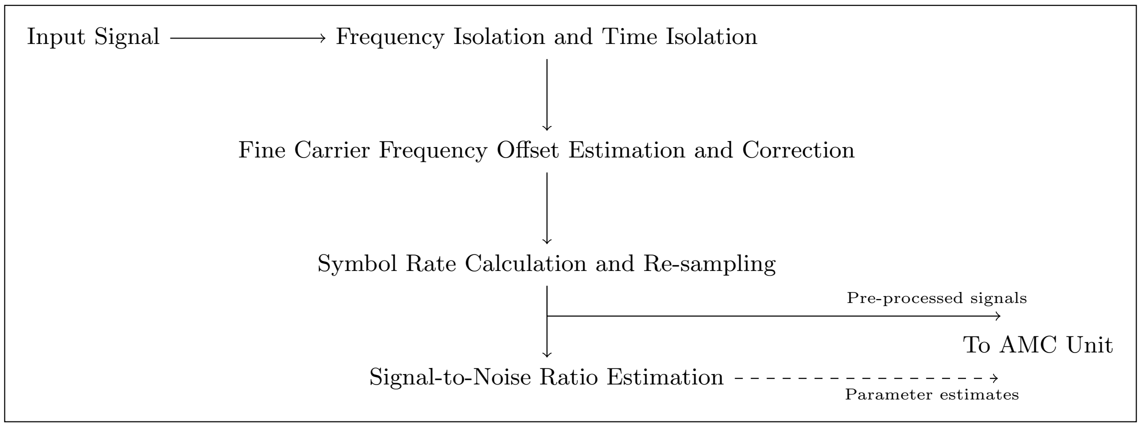
Input Signal
Frequency Isolation and Time Isolation
Fine Carrier Frequency Offset Estimation and Correction
Symbol Rate Calculation and Re-sampling
Pre-processed signals
To AMC Unit
Signal-to-Noise Ratio Estimation
Parameter estimates

Fig. 1.2: Basic System Overview

1.5 Document Overview

The rest of this document is outlined as follows. The signal processing required to achieve the goal of this work is described in chapter 2. The system details including algorithms and theory are outlined in chapter 3; in fact, section 3.1 goes over each sub-system in great detail and explains decisions made throughout the design process and explicitly points out areas of optimization. The other sections in the chapter discusses tools that were created for interfacing with the system for code development, verification and analysis. Some results and observations are portrayed in chapter 4. Finally, chapter 5 concludes this thesis and discusses areas for expansion and future work.

Chapter 2
Signal Processing Solution

This chapter discusses in detail the solution design from a signal processing perspective. It provides a bridge between the system details (i.e. the software solution) and the high level point of view. The first section describes the methods involved in identifying signal locations in time and frequency. The second section describes the techniques used in honing in on each signal.

2.1 Signal Detection

Given the spectrogram in figure 1.1, it is easy for a human to identify the distinct rectangles and estimate their start and stop times and the frequency spectra which they span. This system does not have the luxury of human processing. In order to eliminate data outside of the region of interest for a particular signal, the system is equipped with a method for honing in on each signal. A wideband energy scan is made at regular intervals and with each update of the wideband signal, the signal is analyzed in the frequency domain. The energy in the frequency domain is estimated via the Fast Fourier Transform (FFT) on the data stream. For regions where energy exceeds a threshold, the frequency edges are noted and used to estimate the center frequency (assuming symmetry about the center frequency).

2.2 Signal Isolation

After estimating each signal's center frequency in the wideband data stream, the system makes a copy of the wideband signal and brings it down to baseband by mixing it with a complex exponential at the carrier frequency. After bringing each signal to baseband, the signals are passed through an appropriate low-pass filter (specific to each isolated burst) to remove unnecessary signals from the wide-band and avoid aliasing. Because the signals may be bursty (i.e. they turn on and off at random times), the system only keeps samples between the times in which the bursts are perceived to be present. Thus, the output of the system is a collection of basebanded signals that are of length of the duration of the respective burst (although, down-sampling reduces redundant samples).

2.3 Signal Parameter Estimation and Pre-Processing

Automatic modulation identification often requires *a priori* knowledge of the signal to some degree or another. This work prepares signals for automatic modulation identification by estimating key aspects of signals and preparing them for further analysis. Parameters that are essential to automatic modulation identification are the signal-to-noise ratio (SNR), the carrier frequency and the sample rate. When the signals are isolated in frequency (and in time) as described in section 2.2, the carrier is assumed to be zero or also referred to as DC or baseband. When each signal is centered at DC, their symbol rates are estimated by employing a non-linear operator called the Teager Energy Operator [7, 8] which exposes periodicities in the signal. With symbol rate estimates, the system re-samples to a nominal sampling rate. With the known sample rate after re-sampling, the system estimates the SNR by averaging the energy level for each burst at two spectral locations: the signal plus noise portion centered about DC and the just noise part located at high frequency.

2.4 Solution Summary

This section summarizes the work's main contributions to the problem outlined in section 1.3. This work's overall contribution is the signal conditioning of wideband, dynamic, multi-channel signals with few assumptions. The signal conditioning provided includes

- Burst detection
- Burst isolation in time and frequency
- Coarsely bringing bursts to baseband and down sampling
- Fine correction of residual carrier frequency offsets

- Symbol rate estimation for each burst
- Re-sampling of each burst to a nominal sample rate
- Accurate estimation of each burst's signal-to-noise ratio

2.5 Prior Work

There exists a framework called the Rapid Radio framework that is used for AMC [9]. The techniques used rely on the assumption that a human is in the loop for detecting a sub-signal for processing. Also, the techniques require linear modulation and pulse shaping with a root-raised-cosine. Based on that assumption, the system uses an iterative method to estimate parameters that influence signal features in the frequency domain. These techniques were explored and implemented in MATLAB. The techniques used in this work require less computational complexity and less assumptions on the incoming signal. This work also does not require human in the loop.

Chapter 3

Software Solution

This chapter discusses in detail the algorithms used for the signal processing described in chapter 2. The processing is divided into two main systems. The first is the signal detector which is comprised of a few different software modules, each of which are described in sections 3.1.2 and 3.1.3. The second division in the signal processing flow is the series of signal conditioning algorithms and processes which group is instantiable per detected frequency region of interest (ROI). These are described in sections 3.1.4 through 3.1.10. Section 3.3 discusses tools developed for visualizing the system in progress and for the assessment of its results. Throughout this chapter, there are C code modules and routines that are referenced for various purposes. These references in this document appear in different font to distinguish them, for example `while(i++ < j)` is a C code method for incrementing the variable `i` until it surpasses the value `j`.

3.1 System and Sub-Systems Details

For the remaining of this document, the following notation will be used. Let $x(t)$ be the continuous-time full-band signal that is received. Let $x[n] = x(n\mathrm{T})$ be the quantized, sampled version of the raw, continuous-time full-band signal that is received, where T is the sample period. In other words, $x[n]$ is the output of the analog-to-digital converter (ADC) in the receiver. Let $\mathbf{x}_i$ be the i^{th} data chunk consisting of M samples such that,

$$\begin{aligned}\mathbf{x}_i &= \Big[x\,[iM] \quad x\,[iM+1] \quad x\,[iM+2] \quad \cdots \quad x\,[iM+M-1]\Big]^{\mathrm{T}} \\ &= \Big[x\big(iMT\big) \quad x\big((iM+1)T\big) \quad x\big((iM+2)T\big) \quad \cdots \quad x\big((iM+M-1)T\big)\Big]^{\mathrm{T}}\end{aligned}$$

Further note that when referring to the current data chunk, or the data chunk being used at the present time, the arbitrary subscript may be dropped, i.e. $\mathbf{x} = \mathbf{x}_i\big|_{i=\text{now}}$.

The system, as depicted in figure 3.1, from a top level consists of modules, most of which contain sub-modules. All the modules and sub-modules vary in sophistication and complexity. Each subsystem is described in detail in this chapter. The theory of operation and decisions made are also outlined here. This project is to be implemented on a micro-controller and so it was developed in C so as to be easily portable to the target platform. There were several design choices that were made as a result of using C. These optimizations are also discussed in this chapter.

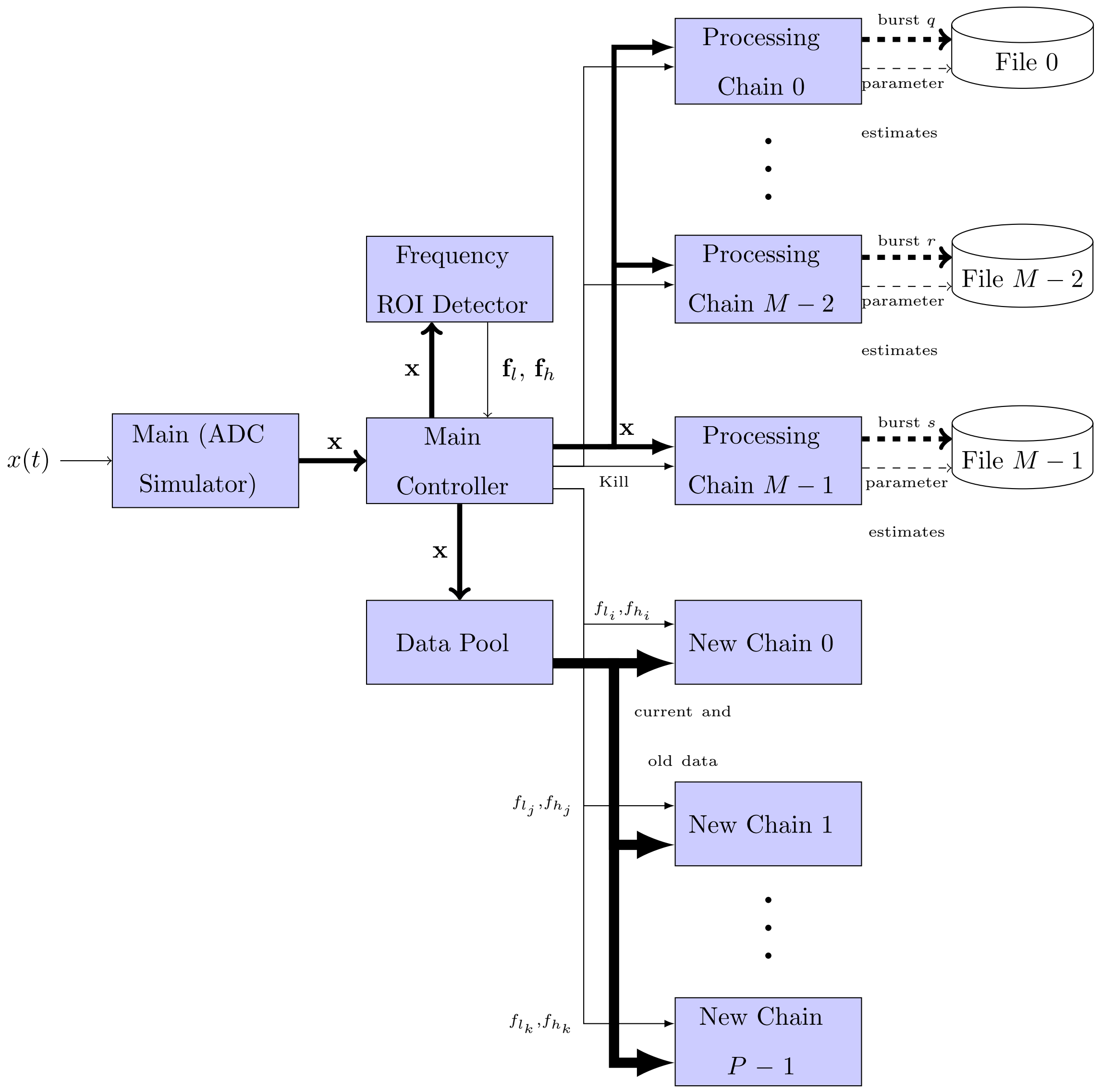

Fig. 3.1: System Top View

3.1.1 Power Spectral Density Estimator

Several modules in the system require an object that can compute and update a signal's power spectral density (PSD) estimate. These modules include the Frequency ROI Detector (i.e. the signal detector or just "the detector") as well as the following which are sub-modules the processing chains in figure 3.1 as seen in figure 3.6: the carrier frequency offset estimator, the symbol rate estimator and the signal-to-noise estimator. The power spectral density estimator uses the Fast Fourier Transform (FFT) to transform the current data chunk from the time-domain to the frequency-domain. The power is then estimated by taking the absolute value (or magnitude of the complex result) and squaring the result. The PSD Estimator is also equipped with smoothing algorithms. Some modules require a smooth PSD estimate. A simple lowpass filter is implemented to reduce the effects of noise on the estimate. Let $Y_i[n]$ be the n^{th} sample from the PSD estimate at time i. The lowpass filtered PSD estimate at i is then represented as,

$$Y_i[n] = \alpha Y_i[n] + (1-\alpha)Y_{i-1}[n], \qquad \alpha \,\epsilon\, [0,1]$$

where $Y_0[\cdot]$ is initialized with zeros. This basic technique is also depicted in figure 3.2. For greater smoothing, the user should use a value closer to one for α, and for scenarios where smoothing is less important, the α value can be closer to zero. The trade off to a low cut-off frequency (α closer to one) in this algorithm is that more data are required (i.e. more calls to `update_psd()` to get an accurate PSD estimate.

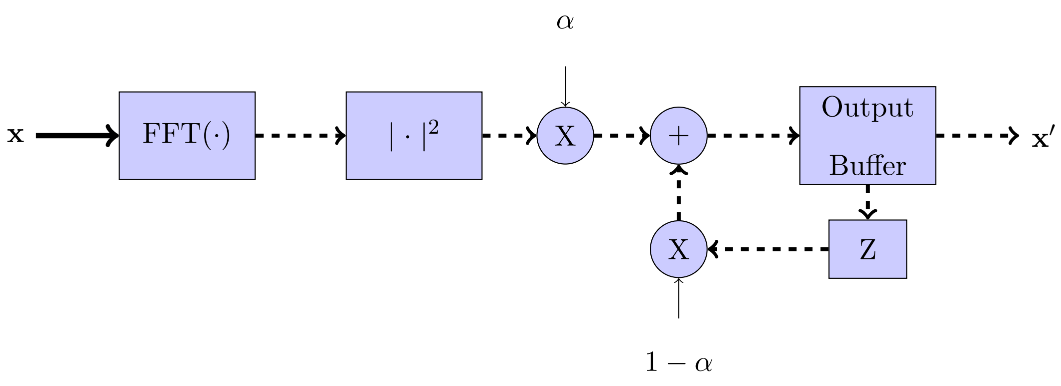

Fig. 3.2: PSD Estimate Calculation with Lowpass Filter Mechanism

For an even more smooth PSD estimate, the array containing the lowpass filtered estimate is run through a nearest neighbor averaging algorithm which is works as follows,

$$Y_{i^+}[n] = \left(Y_i[n-M]+Y_i[n-M+1]+\cdots+Y_i[n-1]+Y_i[n]+Y_i[n+1]+\cdots+Y_i[n+M]\right)/\,(2M+1)$$

Because the incoming signal is complex, a lot of the data that is dealt with by individual modules is complex. The C implementation uses a highly optimized linked library called FFTW (fastest Fourier transform in the west) [10]. The FFTW library, if configured properly, uses the native data type for complex numbers. The built in C library `<complex.h>` provides functions for computing the magnitude of the complex data type. For the power spectral density estimator, it is understood to be more efficient to compute the magnitude-square of each complex sample as follows. Let $x = a + bj$. By definition, $|x| = \sqrt{a^2 + b^2}$. Using the C's complex library, this in code would normally look something like, `y = cabs(x)`. Furthermore, the magnitude square would typically be computed in C with `z = pow(cabs(x),2)` with the `<math.h>` built in library or alternatively as `z = cabs(x) * cabs(x)`. Note that the magnitude-square is simply $|x|^2 = a^2 + b^2$. Therefore, rather than calling `cabs` and squaring, this implementation uses the optimized calculation: `z = creal(x)*creal(x) + cimag(x)*cimag(x)`. This method eliminates at least one multiply and a square-root per sample each time `update_psd()` is called. Later on it will be seen that the PSD estimator's update PSD estimate routine is called at least once (if not more) by more than 3 modules in this system for every input data chunk. Further, for every input data chunk, a few of the modules that use the PSD estimator are instantiated for each detected burst. In other words, for each data chunk, the number of calls to `update_psd()` are at least, $1 + 3 * N_{\text{active bursts}}$. Thus, this optimization saves significant computation. Figure 3.3 provides good comparison. The leftmost plot is the raw FFT of an arbitrary data chunk with signal in it at 0.25 normalized frequency. The center plot is the lowpass filtered version of this data chunk with the lowpass filtering algorithm just described. To even further smooth the spectral view of this signal, the M-nearest neighbor smoothed version is shown in the rightmost plot of figure 3.3. This figure was generated using a 2048 point FFT, 0.0125 α value in the lowpass filter and 10 nearest neighbors on each side of each sample of the lowpass filtered FFT output for the smoothing.

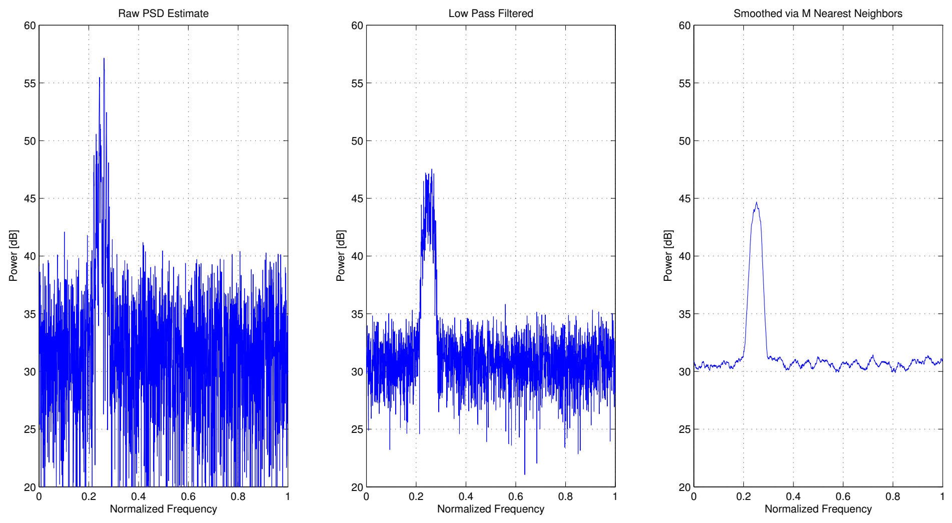

Fig. 3.3: Example comparison of the three potential outputs of the PSD Estimator

3.1.2 Detector

The detector is the first step in the series of processing abstracted in figure 1.2. In figure 3.1, it is found just above the main controller. This module is in charge of maintaining a PSD estimate of the current data chunk using the PSD estimator and identifying regions of interest in the frequency domain which likely contain signal. For each data chunk received, the detector updates its PSD estimate of the incoming signal and employs a simple threshold detection algorithm to identify the ROIs in the frequency domain that correspond to signal activity. In the frequency domain, signals of high signal to noise ratio (SNR) rise above the noise floor and typically (depending on the modulation scheme) look like swellings as seen in figure 3.4. This system is configured with parameters including the noise floor average energy level and the threshold level and thus it assumes the user has prior knowledge of the noise floor as well as the appropriate threshold value to use. The detector returns a list of frequency pairs that described the detected frequency swellings. In the code, these frequency pairs are notated as f_{l_i} and f_{h_i} for the lower and upper edge frequencies, respectively, for the i^{th} detected burst. An example of the threshold detection in action is shown in figure 3.4. In this case, the detector would return $\mathbf{f}_l = \begin{bmatrix} 0.204 & 0.464 \end{bmatrix}^{\mathrm{T}}$ and $\mathbf{f}_h = \begin{bmatrix} 0.29 & 0.53 \end{bmatrix}^{\mathrm{T}}$.

This module works well when the threshold value is appropriate and the SNR values for each burst are reasonable.

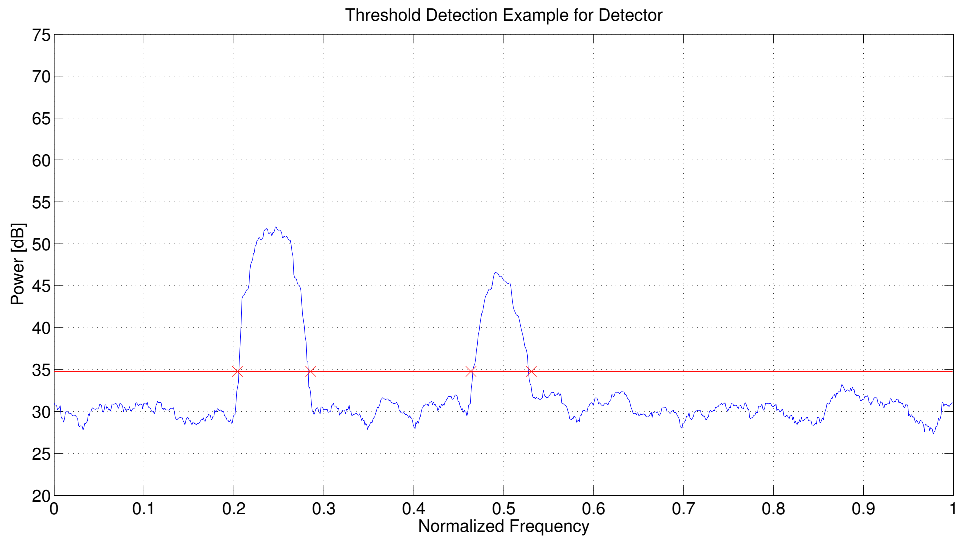

Fig. 3.4: For this example the red horizontal line is the threshold value and the red Xs reflect the detected frequency edges. Note that what is compared against the threshold is the updated, lowpass filtered, smoothed PSD estimate.

3.1.3 Main Controller

The main controller, as seen in figure 3.1 is the central manager. For each burst in the wide band signal, there is a dedicated and customized series of sub-processors responsible for all the front-end processing of the sub-signal. Each series is referred to as a processing chain (see figure 3.1 for a processing chain's place in the top view and figure 3.6 for its subsystems). When the main controller receives a new data chunk from `main` (acting like an ADC output buffer), this top level module forwards it onto the detector and subsequently receives a list of frequency pairs indicating frequency regions that likely contain a signal. The main controller decides from that list, based on the current state of the program, whether or not to spawn a new processing chain for any one of the frequency pairs. The main controller manages several lists in a step called `process_data_chunk()`. If a processing chain already exists for a frequency pair in the newly detected list, the main controller simply forwards the raw input data chunk to the chain. All such chains are kept for the next go-around. For each processing chain whose associated frequency pair is not in the newly detected list, the kill signal is activated indicating to the chain's chain controller to update itself one last time with the current data chunk and then terminate itself. If at the end of the frequency pair "matching" processes there are frequency pairs where no chain was created, a new chain is created and initialized with data stored in a data pool of configurable length. Figure 3.5 provides an abstract for the matching processes.

Detected Frequencies	Old Chains	Matched Chains
f_{l_i}, f_{h_i}	f_{l_g}, f_{h_g}	(empty)
f_{l_j}, f_{h_j}	f_{l_h}, f_{h_h}	
f_{l_k}, f_{h_k}	f_{l_i}, f_{h_i}	
	f_{l_j}, f_{h_j}	

(a) Before Matching

Detected Frequencies	Old Chains	Matched Chains
f_{l_k}, f_{h_k}	f_{l_g}, f_{h_g}	f_{l_i}, f_{h_i}
	f_{l_h}, f_{h_h}	f_{l_j}, f_{h_j}

(b) After Matching, Before Terminating

Detected Frequencies	Old Chains	Matched Chains
(empty)	f_{l_i}, f_{h_i}	(empty)
	f_{l_j}, f_{h_j}	
	f_{l_k}, f_{h_k}	

(c) After Matching and Terminating, Ready for Next Round of Matching

Fig. 3.5: Hypothetical Matching Example

3.1.4 Chain Controller

The processing chain is an instantiable series of sub-modules responsible for concatenating the raw input data chunks for the duration of its assigned burst, bringing the raw data to basebanded according to the roughly estimated carrier frequency, finely correcting carrier frequency offsets, timing offsets and re-sampling to a nominal sample rate. Figure 3.6 illustrates this cascaded process. The chain controller also contains a module for estimating its designated signal's SNR (located near the end of the processing chain in figure 3.6). When the chain controller receives the kill signal from the main controller, it stores its results and estimated parameters to a unique file. This interface can easily be modified to feed directly into its intended cascaded program, the automatic modulation classifier.

In order to support bursty signals, the processing chain has a data pool that is filled as data chunks come in. When the processing chain receives notice from the processing chain controller that no more data for that frequency band of interest is available, the processing chain has the data from its data pool fed into the time isolator which subsequently feeds only the data from the data pool between the estimate start and stop times into the rest of the processing chain (beginning at the CFO Handler sub-module).

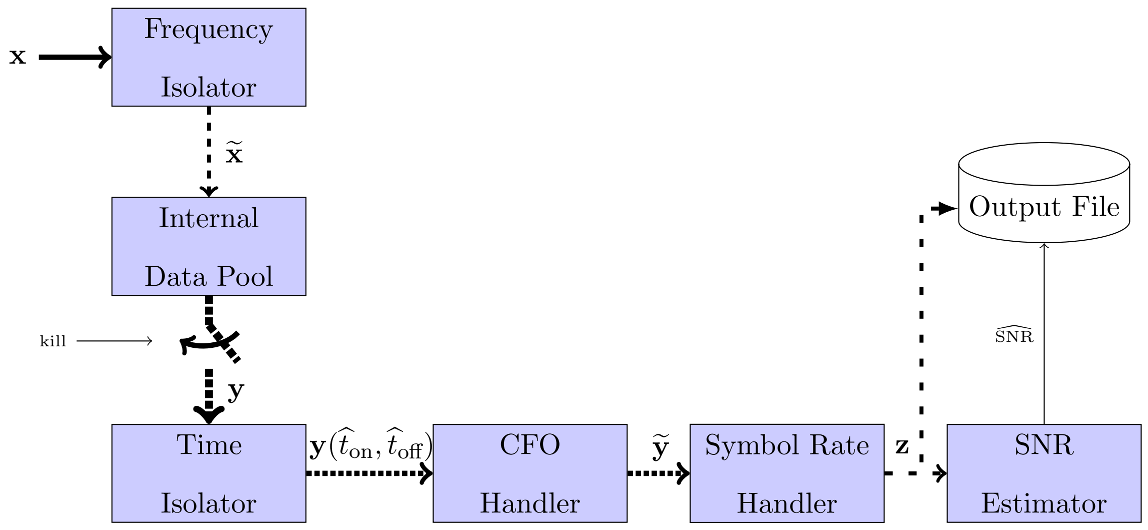

Fig. 3.6: Processing Chain

It is worth noting the benefits of having a dedicated processing chain instance for each burst. The incoming wide-band signal is sampled at a very high rate in order to get a wide spectrum. The bursts in the signal are narrow band compared to the full spectrum in which they reside. Each chain controller operates at a lower sampling rate than the raw incoming data and therefore the DSP operations per sample is actually less in total of all active chains than a single chain on the fully sampled raw signal would be. Therefore, even though a single raw data chunk is copied for each chain, each chain controller's rough down sampler eliminates a lot of complexity.

3.1.5 Frequency Isolator

Each processing chain instantiation has its own designated frequency area on which to operate. This area is bounded between $\widehat{f_l}$ and $\widehat{f_h}$. The frequency isolator ensures that all modules downstream truly operator within the designated frequency area by supplying only the data within the specified region. The frequency isolator takes each incoming data chunk and mixes it with a complex exponential at the estimated carrier which centers the region of interest about DC (i.e. baseband). This module then lowpass filters the result in order to "isolate" the region of interest. (see figure 3.7). The lowpass filter removes higher (and lower) frequency activity that is not needed. For optimization, the lowpass filter is automatically generated at chain initialization and is constructed from the Kaiser window that was created before compile time. To be specific, the following is equation from [11] shows how the finite impulse response (FIR) lowpass filter's (LPF) coefficients are computed.

Let $h[n]$ be the n^{th} coefficient from the FIR LPF, where $n \in [0, M-1]$ for a filter of order M.

$$h[n] = \frac{\sin\big(\omega_c(n-\alpha)\big)}{\pi(n-\alpha)} w[n] \tag{3.1}$$

where $\omega_c = 2\pi(\widehat{f_h} - \widehat{f_c})$ is the desired cutoff frequency and α is a constant derived from the parameters used to generate the Kaiser window coefficients notated as $w[\cdot]$. According to [11], the computation of each Kaiser window coefficient $w[\cdot]$ requires a few multiplies, a couple of adds and also the calculation of a *zero*th*-order modified Bessel function of the* *first kind*. It is apparent that even for a low order FIR filter, pre-computing the Kaiser window saves a lot of computations. For even further optimization on each filter coefficient generated, two look-up tables (LUT), populated prior to compilation, are used. One LUT contains the values for $2\pi(n-\alpha)$ while the other contains the values for $\frac{w[n]}{\pi(n-\alpha)}$ where $n = 0, 1, 2, \ldots, M-1$. Thus, only two multiplies, and a sinusoidal computation are required for each filter coefficient computation at initialization of each processing chain.

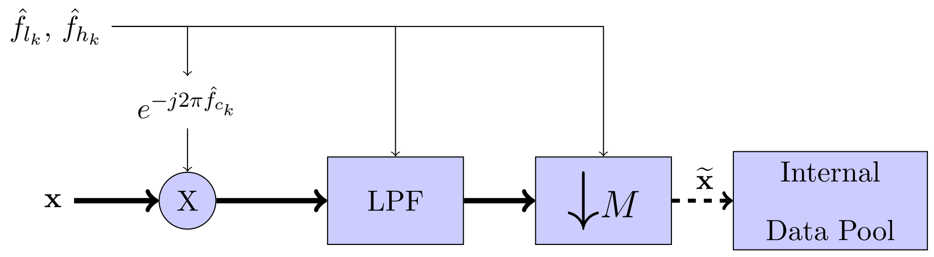

Fig. 3.7: Frequency Isolator for k^{th} Processing Chain

3.1.6 Time Isolator

Because of PSD estimate smoothing and the nature of threshold detection methods, there is a time delay in detecting when a bursty signal turns on and off. As mentioned earlier, the main controller has a data pool to store old data so that when a new signal arises, the processing chain will have a clear view of the start-up edge despite the time delay in detection. Data is continuously added to each processing chain until the detector says the bursty signal has turned off which again is a delayed detection. Because of this, there are signal-absent samples both before and after the samples containing signal. To ensure accurate analysis of each burst, it is important that the signal-absent samples are excluded. The time isolator marches through the coarsely truncated signal, searching for the actual time in which the signal energy level is above and then below a certain threshold which times correspond to the on/off times of the burst. The instantaneous energy for any particular sample is estimated by squaring its amplitude value in the time-domain. Figure 3.8 is an illustration of the time isolation process and figure 3.9 shows an example of the estimation of the start and stop times using the threshold technique.

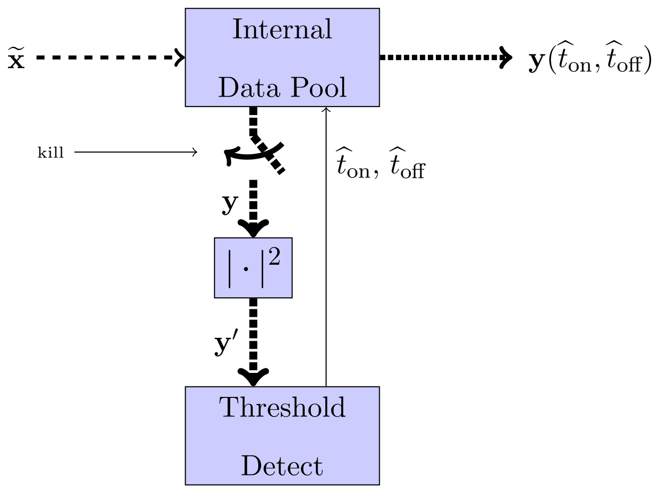
$\widetilde{\mathbf{x}}$
Internal Data Pool
$\mathbf{y}(\widehat{t}_{\text{on}}, \widehat{t}_{\text{off}})$
kill
$\widehat{t}_{\text{on}}, \widehat{t}_{\text{off}}$
$\mathbf{y}$
$|\cdot|^2$
$\mathbf{y}'$
Threshold Detect

Fig. 3.8: Time Isolator for k^{th} Processing Chain

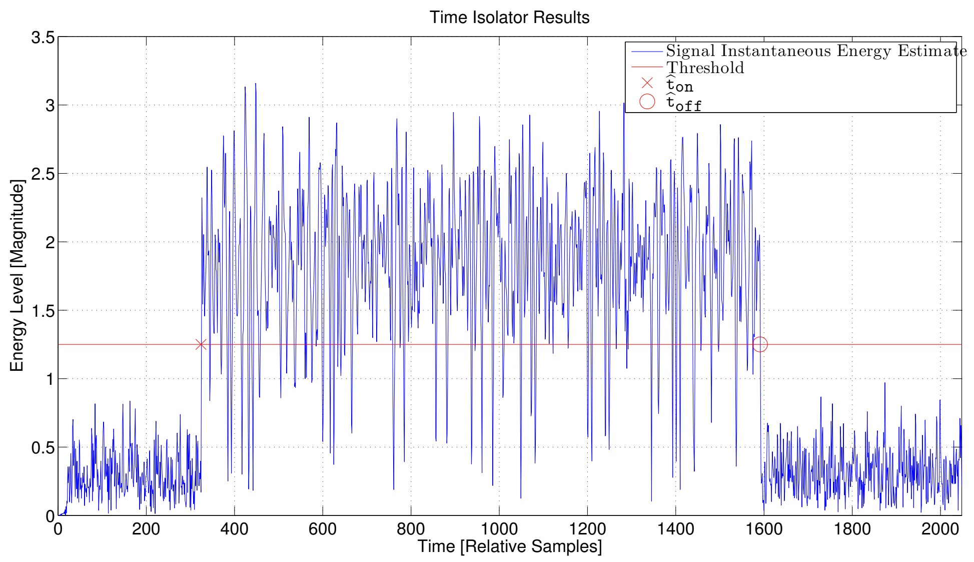

Fig. 3.9: Example Output for the Time Isolator

3.1.7 Carrier Frequency Offset Estimator

Once the chain controller has honed in on the signal part of the data from the internal data pool, notated as $\mathbf{y}(\widehat{t}_{\text{on}}, \widehat{t}_{\text{off}})$ in figure 3.6), the chain controller is ready to more finely correct the adverse effects of the noisy transmission channel. This module estimates the carrier frequency offset (CFO) and then mixes the isolated burst with a complex exponential based on the CFO estimate to undo this offset. The technique used to estimate the CFO relies heavily on that in [12]. The idea is that a when pair of identical filters, each with an equal positive and negative offset respectively from the carrier frequency are applied to the PSD estimate, the power output of each filter theoretically should be equal. Therefore, taking the difference of the power output of such filters will reflect the offset from the carrier if each filter is shifted either up or down in frequency by the same quantity. In [12], [4], such filters are called "band-edge" filters and are used in [4] to detect carrier offsets. Taking this technique a step further, this implementation employs 2 additional copies of the band-edge filter pair. The additional filter pairs are each shifted by some value up and down, respectively, in frequency. Figure 3.10 shows an example of the 3 pairs of band-edge filters, and figure 3.11 shows the 3 pairs on top of each other. The pair's difference in power output of each filter is computed for each pair. Let $p_{i,j}$ represent the power output of filter i of pair j, where $i \in [0, 1]$ and $j \in [0, 2]$. The difference in power output from each filter in pair j is notated as d_j. Thus,

$$\mathbf{d} = \begin{bmatrix} p_{1,0} - p_{0,0} \\ p_{1,1} - p_{0,1} \\ p_{1,2} - p_{0,2} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

The power difference for filter pair j is alternatively expressed as $d_j = |F(z)H(z)_{1,j}|^2 - |F(z)H(z)_{0,j}|^2$ where $H(z)_{i,j}$ is the frequency response of the i^{th} filter of filter pair j. Let the frequency at which the j^{th} pair of band-edge filters are centered be represented by f_j. The goal is to find a value f_{cfo} such that the corresponding d_{cfo} value is zero by fitting a best-fit line to the measured d_j values given the known f_j values. Such an f_{cfo} is the carrier frequency offset. With three pairs of band-edge filters and subsequently 3 power differences among the filters, a linear least-squares solver can be set up as follows.

Let,

$$\mathrm{F} = \begin{bmatrix} f_0 & 1 \\ f_1 & 1 \\ f_2 & 1 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} m & b \end{bmatrix}$$

where m is the slope and b is the y-intercept at $f = 0$. Therefore, the equation for the line is given by $\mathbf{d} = \mathrm{F}\,\mathbf{a}$, from which $\mathbf{a}$ is optimally solved for via the linear least-squares solution,

$$\widehat{\mathbf{a}} = \left(\mathrm{F}^{\mathrm{T}}\mathrm{F}\right)^{-1}\mathrm{F}^{\mathrm{T}}\,\mathbf{d} \tag{3.2}$$

For only 3 points, this solution is simple to solve. However, easy reductions can still be applied to the computation to even further simplify the calculation and reduce the complexity. Because one pair of filters is centered at DC, the f value is zero. Let that pair be the $j = 1$ pair. Also, in this setup, the other two pairs are equally offset from DC, but just in opposite directions. In other words, one pair is offset by positive ϵ and the other offset pair is shifted by negative ϵ in the frequency domain. Therefore, we can set $f_0 = -f_2$. Thus, equation (3.2) becomes,

$$\widehat{\mathbf{a}} = \begin{bmatrix} -\frac{1}{2\epsilon} & 0 & \frac{1}{2\epsilon} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \mathbf{d} \tag{3.3}$$

Therefore,

$$\widehat{m} = \frac{1}{2\epsilon}(d_2 - d_0) \tag{3.4}$$

and,

$$\widehat{b} = \frac{d_0 + d_1 + d_2}{3} \tag{3.5}$$

and thus we can solve,

$$\boxed{0 = m f_{\mathrm{cfo}} + b \quad \Rightarrow \quad \widehat{f}_{\mathrm{cfo}} = -\frac{\widehat{b}}{\widehat{m}} = \frac{2\epsilon}{3}\left(\frac{d_0 + d_1 + d_2}{d_0 - d_2}\right)} \tag{3.6}$$

For a very small offset, $d_0 \approx -d_2$ and $d_1 \approx 0$ making (3.6) yield $\widehat{f_{cfo}} = 0$. If $\frac{2}{3}\epsilon$ is pre-computed, then the linear interpolation of the CFO value is only two multiplies and three adds. Furthermore, rather than convolving the time-domain signal with 6 filters (2 for each of the 3 pairs) this system computes the PSD estimate of the coarsely basebanded signal between the estimated start and stop times given by the time isolator. The CFO estimator then averages samples from the PSD estimate for each filter according to its bandwidth and frequency location.

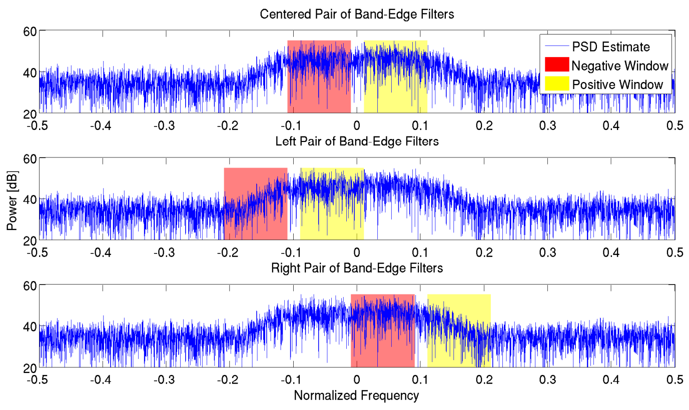

Fig. 3.10: Example of Band Edge Filters

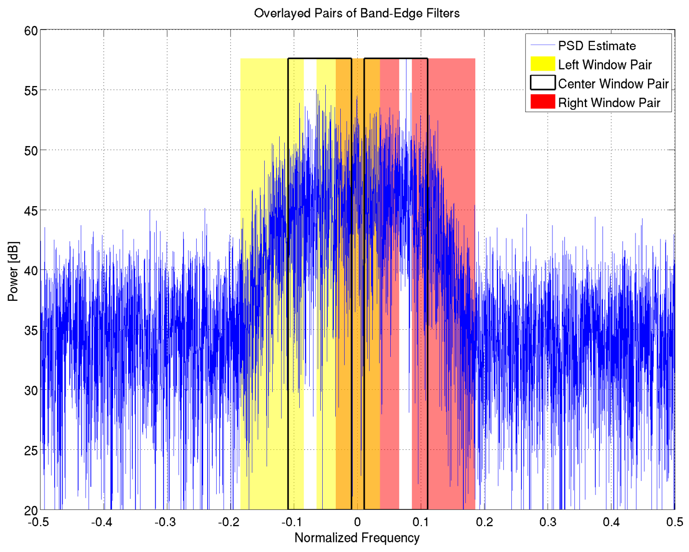

Fig. 3.11: Example of Band Edge Filters Overlaid

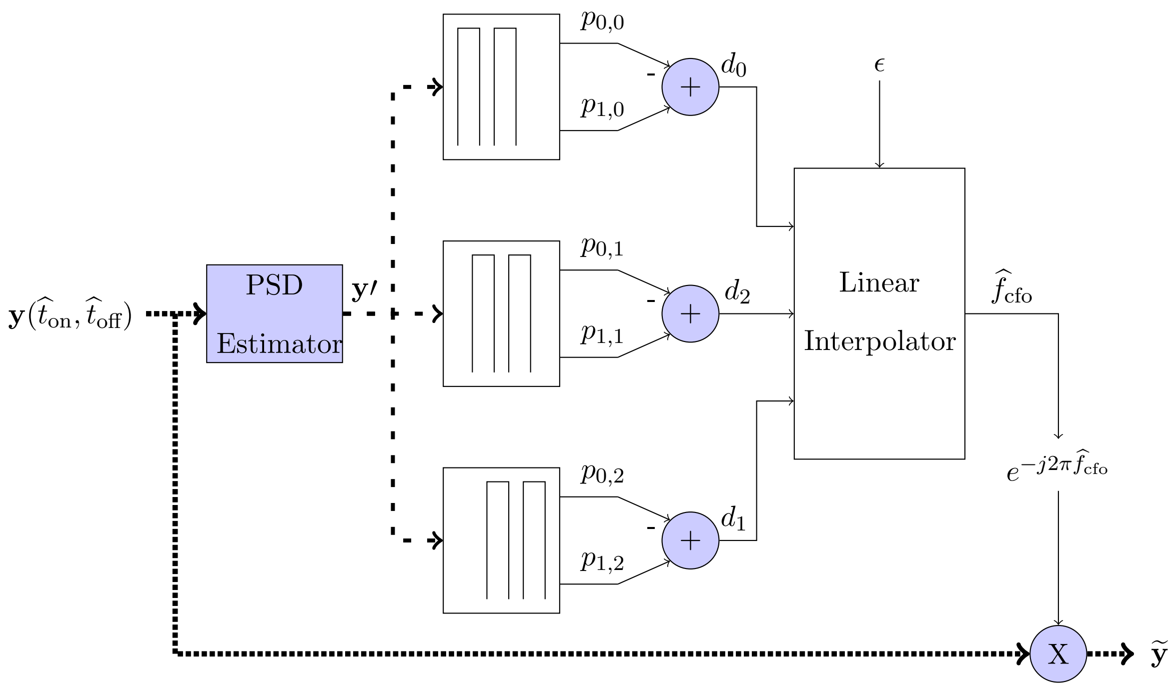

Fig. 3.12: CFO Handler for k^{th} Processing Chain

A good analytic for designing the CFO Estimator's band-edge filters is a visual of how the window pair power difference is affected by the frequency about which they are centered. This is done by taking a single pair of frequency windows, and sweeping their center offset across the entire spectrum and plotting their power difference against their offset. This, intuitively, generates an s-curve and is seen in an example in figure 3.13. This visual gives the designer a good idea of the quality of the $\pm\epsilon$ chosen as well as the gap between the positive and negative windows. Note also in figure 3.13 that the 3 power difference values are included in the plot (notated as red O's) and the interpolated zero-crossing is also included (notated as a red X) as well as the interpolated, best-fit line (notated as a red line). Figure 3.14 is included to show greater detail.

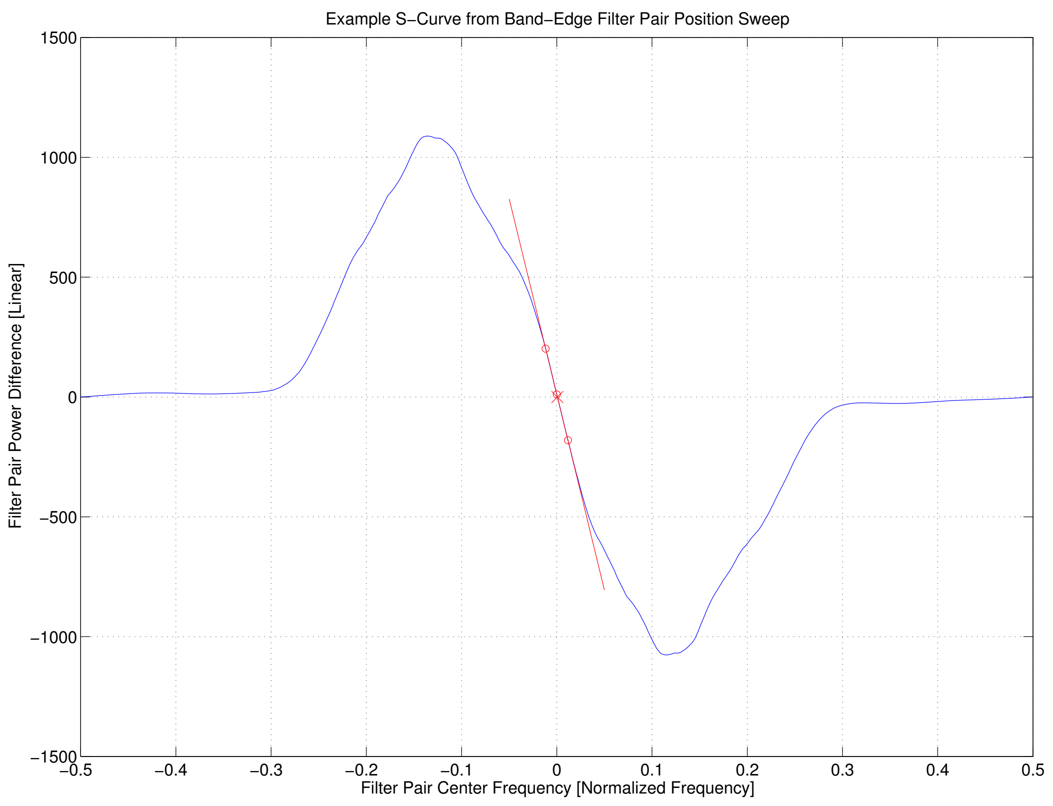

Fig. 3.13: Example S-Curve for a particular CFO Estimator Window Pair

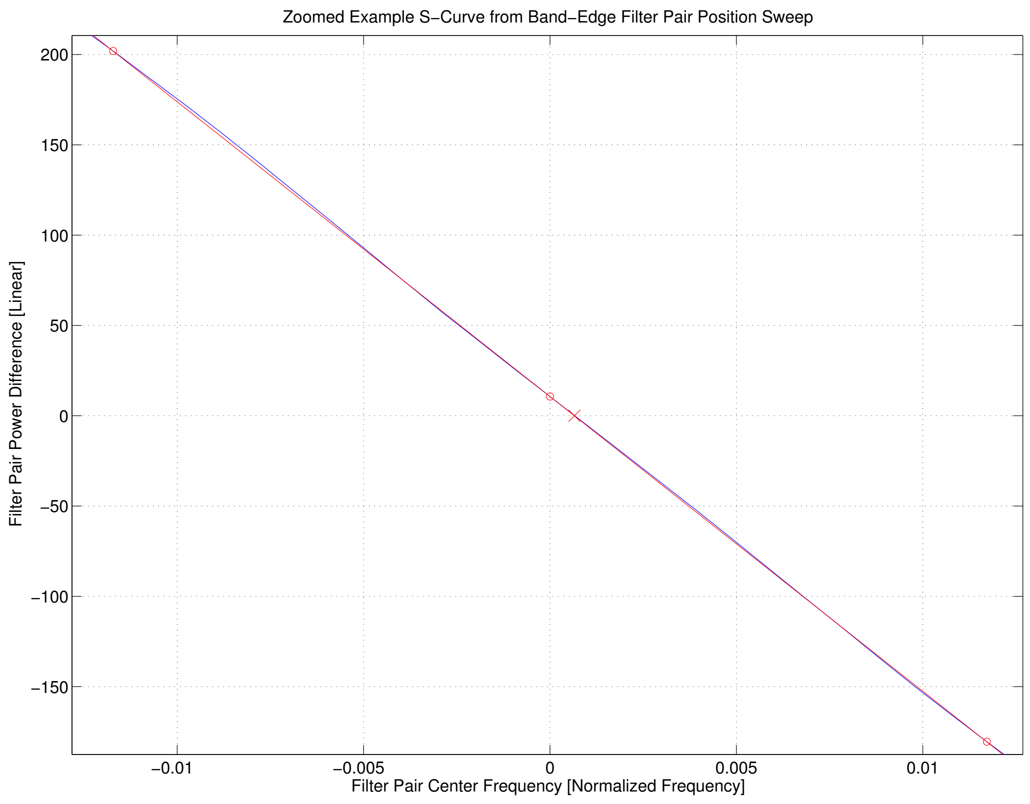

Fig. 3.14: Zoom in of figure 3.13

3.1.8 Symbol Rate Estimator

The symbol rate estimator relies heavily on work done in [7]. The theory of operation is that the received signal (after being isolated in both the frequency and time domains) is periodic according to the symbol rate. Therefore, the Teager energy operator [7, 8, 13] is a non-linear operator that, when applied, exposes periodicities in the signal. In the frequency domain these periodicities are very apparent and therefore, a simple peak finder is sufficient to find the peak location which is reflective of the sampling rate. An example is shown in figure 3.15.

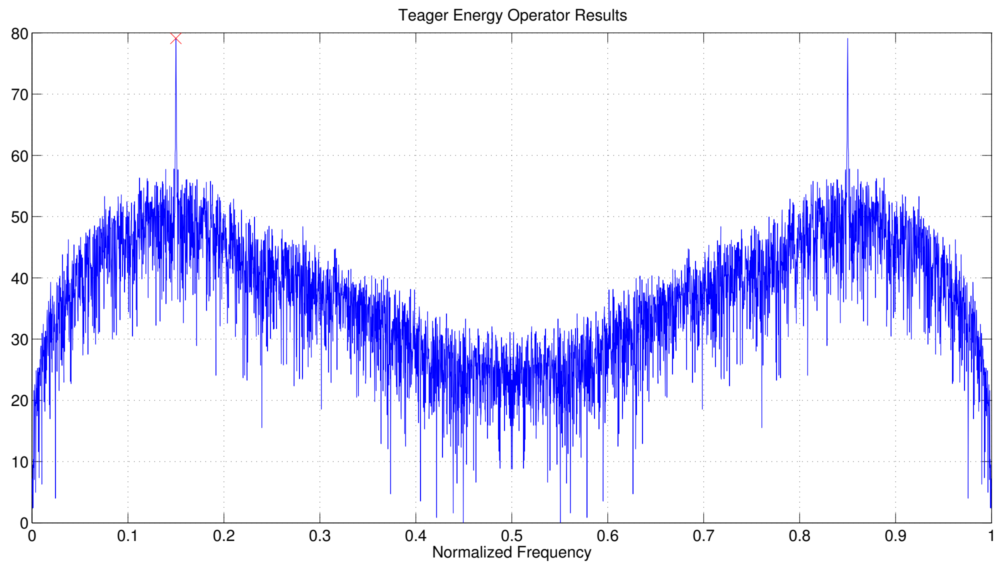

Fig. 3.15: Teager Energy Operator Results in the spectral domain

According to [7], the energy operator is given by

$$\psi_c(x) = \dot{x}\dot{x}^* - \frac{1}{2}\left(x^*\ddot{x} + x\ddot{x}^*\right) \tag{3.7}$$

which is further reduced to

$$\psi_c(x) = \psi_c(x_r) + \psi_c(x_i) \tag{3.8}$$

where x_r and x_i are the real and imaginary parts of the continuous complex signal x. This simplification converts complex algebra to real addition at the cost of an extra addition. Furthermore, according to [7], the discrete form is derived via differentiation approximation,

$$\psi_d[x] = x^2(n) - x(n-1)x(n+1) \tag{3.9}$$

This system uses a modified version to even more simplify the computation and gets similar results.

$$\widetilde{\psi}_d[x] = -x(n-1)x(n+1) \tag{3.10}$$

The first step in this algorithm takes the coarsely basebanded signal which was more finely centered at DC by the CFO handler (see section 3.1.7), and applies the modified version of equation (3.9) found in (3.10). Then the algorithm applies equation (3.8). For example, the n^{th} result in the time-domain of this algorithm is

$$\boxed{\psi(x[n]) = -\Big(\mathscr{R}\Big|x[n-1]x[n+1]\Big| + \mathscr{I}\Big|x[n-1]x[n+1]\Big|\Big)} \tag{3.11}$$

where $\mathscr{R}|\cdot|$ and $\mathscr{I}|\cdot|$ return the real and imaginary parts, respectively.

3.1.9 Resampler

The module responsible for making use of the estimated sample rate, $\widehat{f_s} = \frac{1}{N}$ where N is the number of samples per symbol period, is the resampler. As its name implies, the resampler takes as its inputs the $\widehat{f_s}$, the desired nominal sampling rate $\frac{1}{N_{\text{target}}}$ and the basebanded, time isolated, CFO corrected signal and re-samples the signal so that its resulting sampling rate is the target nominal sampling rate. The resampler computes a sample rate conversion ratio $\frac{U}{D}$ with the current estimated sample rate and the desired sample rate. According to [1], in order to re-sample by an arbitrary ratio, the up-sampling rate is set to some integer and the corresponding down-sampling factor is then computed from the desired target sampling rate. For example, let $U = 5$ and the desired sample rate be $R_{\text{desired}} = \frac{1}{\mathrm{N}_{\text{desired}}}$ where $\mathrm{N}_{\text{desired}} = 4$ samples per symbol period. Let also $\hat{R}$ be the estimated symbol rate which we wish to convert to R_{desired}. We compute at run time the decimation factor D as follows,

$$D = \mathrm{N}_{\text{desired}}\hat{R}U \tag{3.12}$$

It is okay if D is irrational because in the multi-rate poly-phase filter we can interpolate the commutator stride. U must be an integer because it is the also the number of sub-filters in the filter bank. If the number of samples in the filter is not evenly divisible with U, then the remaining coefficients are zeros until each sub-filter is of even length. According to [11], the poly-phase decomposition is as follows: Let $h[n]$ be the original pre and post filter response. The response coefficients are dished out to the sub-filters via

$$h_k[n] = \begin{cases} h[n+k], & n = \text{integer multiple of the interpolation factor M} \\ 0, & \text{otherwise} \end{cases} \tag{3.13}$$

where $k = 0, 1, \ldots, M-1$ and $h_0, h_1, \ldots, h_{M-1}$ are the same length.

$e_k[n] = h[nM+k]$ where M is the number of sub-filters, k is the sub-filter index selecting the k^{th} sub-filter. With a fixed interpolation factor P and a fixed length filter H as depicted in figure 3.16, the sub-filters' coefficients in figure 3.17 are easily distributed prior to run-time thereby making the re-sampling process faster. Furthermore, the polyphase rate converter does not compute the sub-filter output until the commutator is ready for output which may be 1-2 strides depending on the U and D values, thereby halving the number of inner product computations involved in the rate conversion process.

$$\begin{tikzcd} x(nT) \arrow[r] & \boxed{\uparrow P} \arrow[r] & \boxed{H} \arrow[r] & \boxed{\downarrow Q} \arrow[r] & x\left(m\frac{Q}{P}T\right) \end{tikzcd}$$

Fig. 3.16: Resampler Top-View

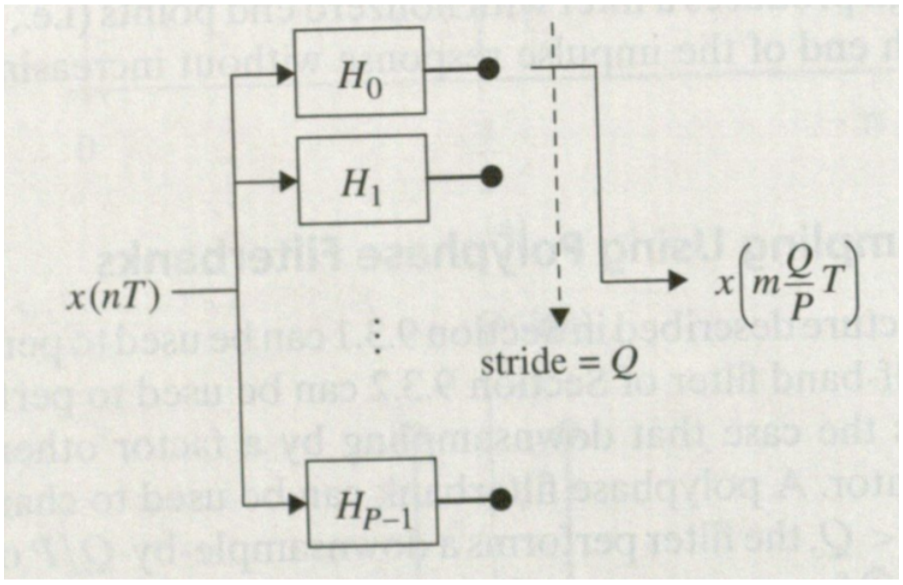

Fig. 3.17: Polyphase rate changer taken from [1] (P is the interpolation factor, Q is the decimation factor)

3.1.10 Signal to Noise Ratio Estimator

At this stage the signal, notated as $\mathbf{z}$ in figure 3.6, is nominally sampled at a known sample rate. Thus it is known how much of the frequency domain the corrected signal occupies. For example, in figure 3.18, the signal is known to be 4 times over-sampled (i.e. 4 samples per symbol period). With this knowledge, the system can expect the signal to take up roughly one half of the frequency spectrum, or two times the reciprocal of the samples per symbol period (in this case the sampling rate). Because it is also a baseband signal, the location of the signal in the spectrum is also known—centered at DC. Thus, the energy of the signal plus noise portion in the frequency domain can be estimated by summing the squared, absolute value of the signal area of the spectrum. Similarly for the noise since we know where only noise resides. More precisely, bandwidth $= \frac{1+\alpha}{N_{\text{nominal}}}$ where, $0 \leq \alpha \leq 1$. Using the lower limit for the bandwidth, (or, perhaps slightly less than), the solely noise part of the spectrum is avoided when estimating its energy level. Similar to the CFO band-edge filtering technique, windows are applied to the PSD estimate of the now nominally sampled and isolated signal burst. The average value within the two windows is taken as an estimate of the average power level of that region. With an estimate of signal + noise energy and just noise energy, the signal energy is simply the difference of the two. Thus, the SNR is estimated by taking the difference of the two and dividing the result by the noise energy level estimate. The C code returns the SNR estimate in linear form and not dB so as to not require using logarithm calculations. Assuming the C code functions as it should, the input to the SNR estimator will always be geometrically the same in the frequency domain, thus the same windows can be used. Therefore, at system initialization, the windows are pre-generated because they only need to be created once.

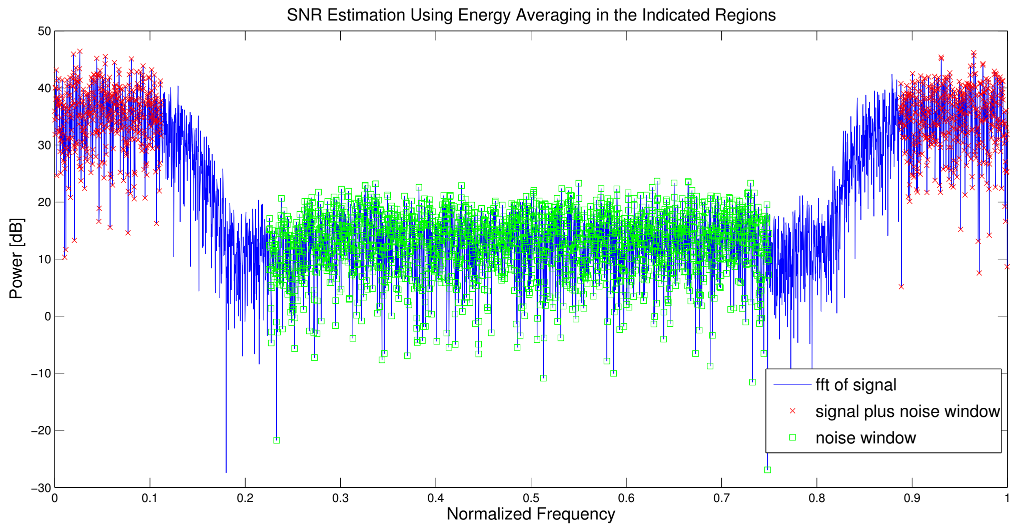

Fig. 3.18: Frequency Domain of Input Signal to SNR Estimator and its Energy Averaging Regions

3.2 Computational Complexity

Table 3.1 summarizes the number of multiplies per sample required in each sub-module. In the table N represents the FFT size for the system, M represents the coarse decimation factor in the frequency isolator and is at least 1, finally, P is the length of the filters used in the CFO calculation, lowpass filtering and rate changing.

Table 3.1: Table of Module Computational Complexity Per Sample (N is the FFT size, M is the decimation factor and P is the size of each filter)

Module	**O**
PSD Estimator	$\log N + 4$
Detector	$\log N + 4$
Frequency Isolator	2
Time Isolator	$\frac{2}{M}$
CFO Handler	$\left(\log N + 4 + 6P + \frac{2}{N}\right)\frac{1}{M}$
Symbol Rate Estimator	$\frac{1}{M}(\log N + 7)$
Resampler	$\frac{1}{M}\left(\frac{2}{N} + P\right)$
SNR Estimator	$\frac{1}{M}\left(\log N + 4 + \frac{3}{N}\right)$

The parameter M has a wide range of values and depends on signal features including the carrier frequency and bandwidth. Parameters N and P are implementation dependent.

Therefore, it is difficult to determine in advance which processing module is the most computationally demanding. Furthermore, the number of operations that must be performed on a per-second basis depends on the base sampling rate of the system, the total analog bandwidth of the analog-to-digital converter, and the expected average number of signals that are likely to be present at any given instant of time. Thus, the computational complexity must be determined on a case-by-case basis using the parameters of the implemented system.

3.3 Human Interface

As mentioned earlier, the interface with the C program is fairly simple. Many parameters that alter the system's behavior appropriate for its input signal characteristics are assigned at run time so the need for re-compiling is minimized. Input signals are pre-generated, Kaiser windows for LPFs are pre-generated and SNR estimation windows are pre-generated prior to compile time. The output of the program is simply processed data for each burst along with an associated file containing the list of estimated parameters which include at least the SNR, carrier frequency, band-width and start and stop times. For program evaluation, this file can also include things like the CFO estimate, the symbol rates before and after re-sampling, etc. The output can easily be modified to interface with it's intended extension program for automatic modulation classification which would take as its input, a signal for when a new burst is available for classification, its SNR, a buffer to receive the data and perhaps an interrupt signal indicating when data is available. The program requires complex floating point format for the input signal. The main function simulates the output of the ADC mixed with the complex exponential demodulator whose output buffer size is the same as the system's input buffer size. For user information, debugging and analysis, the input signal file must have the true signal parameters listed at the start of the input file, followed immediately by the input samples. The system reads N samples at a time, starting from the beginning, for an input buffer size of N, until the end of file is reached.

3.3.1 Virtual Oscilloscope: MATLAB Companion Program

To view the performance of the C program, a companion program was created in MATLAB called the Virtual Oscilloscope. This program takes in special output from the C program which output can be disabled for normal operation as opposed to debug or analysis operative modes. Figures 3.19 through 3.23 are example outputs of the Virtual Oscilloscope. Figure 3.19 is the main figure and is updated with every newly received data chunk. For each detected burst, a figure illustrating the status of the isolators (frequency and time) is created as in figures 3.20, 3.23, 3.26 and 3.27. Note that in figures 3.26 and 3.27, the time isolation results are not displayed. That is because at the particular instance in time, as seen in figure 3.19, the falling edge in time of bursts 4 and 5 have not yet been detected. When the burst has been fully captured and isolated in time, 2 new figures are created the burst. The first, as in figures 3.21 and 3.24, shows some key visuals for each processing sub-module. These visuals include:

- The PSD estimate as well as the 3 frequency window pairs used in the CFO calculation.
- The S-curve from the same set of window pairs and the CFO's 3 data points and interpolation result from the linear curve fit (depicted as 3 red X's, a red circle and red line, respectively).
- The Teager Energy Operator results in the frequency domain as well as the detect peak indicated as a red X.
- A before and after of the re-sampling in the frequency domain.
- The samples used from the frequency domain for the SNR estimation. Signal plus noise samples used are indicated as red circles and the just noise samples are indicated as black squares.

The second figure created after the burst has been isolated in time and frequency is a collection of visuals on the signal status after all the processing (i.e. the final output that is ready to be sent to the AMC unit). These visuals are the final result's eye pattern, IQ plot, scatter plot and symbols on the complex plane. These are seen in figures 3.22 and 3.25.

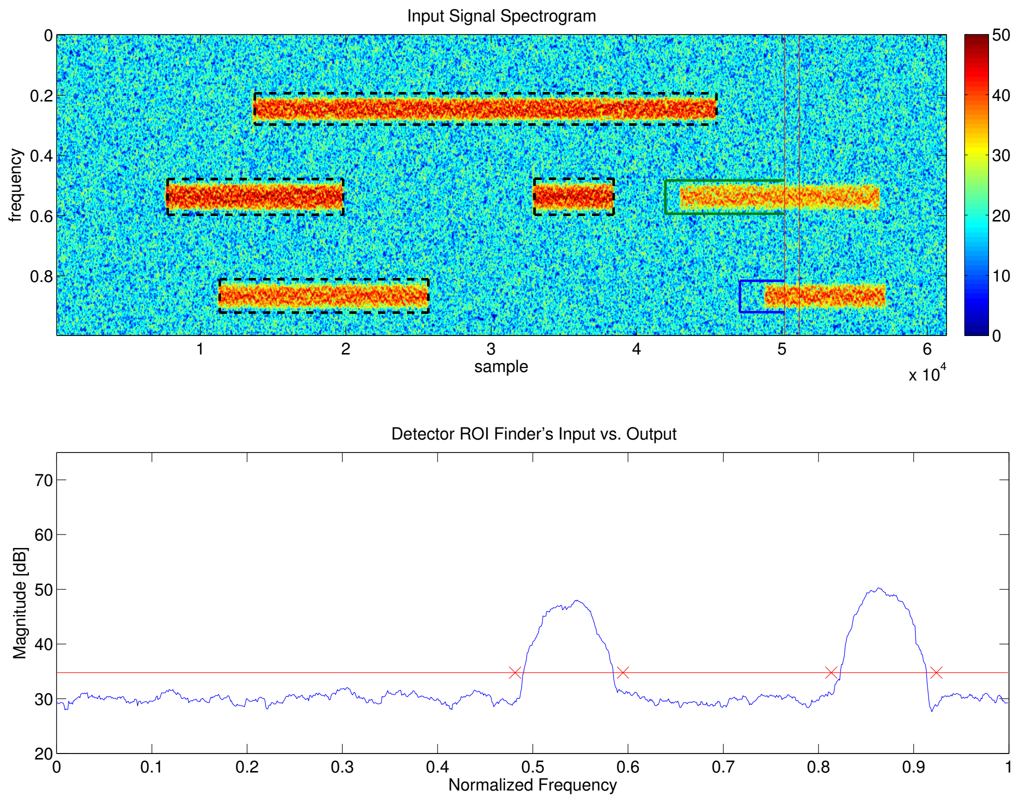

Fig. 3.19: Main Figure

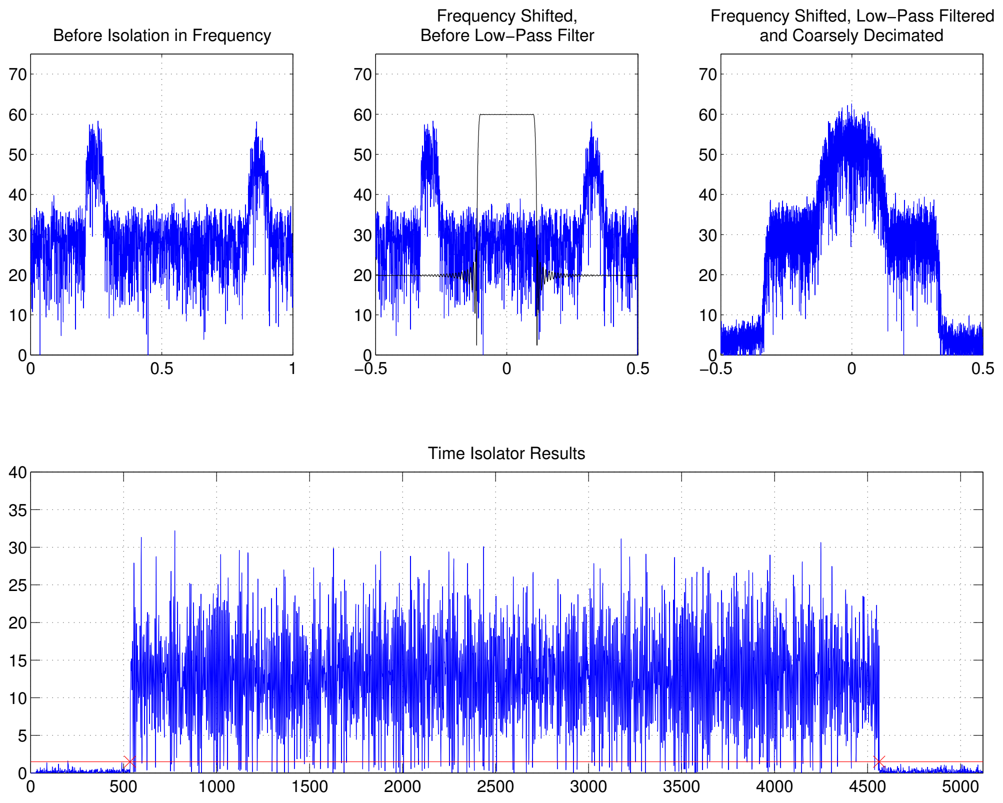

Fig. 3.20: Processing Chain 0 Isolators

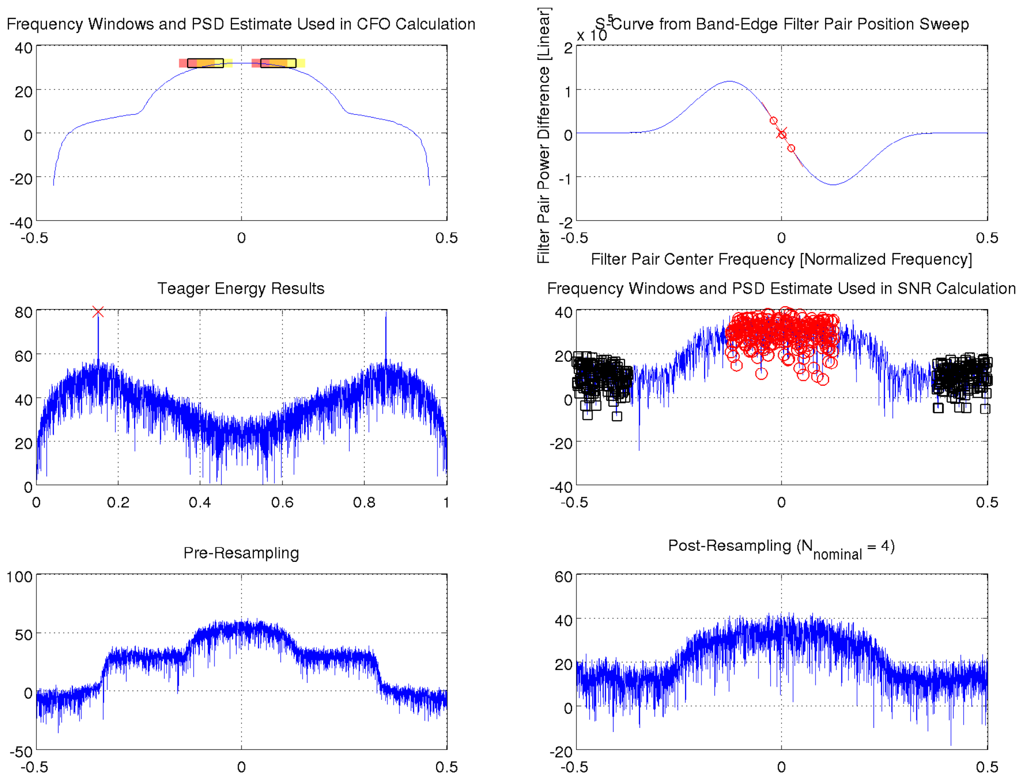

Fig. 3.21: Processing Chain 0 Estimators

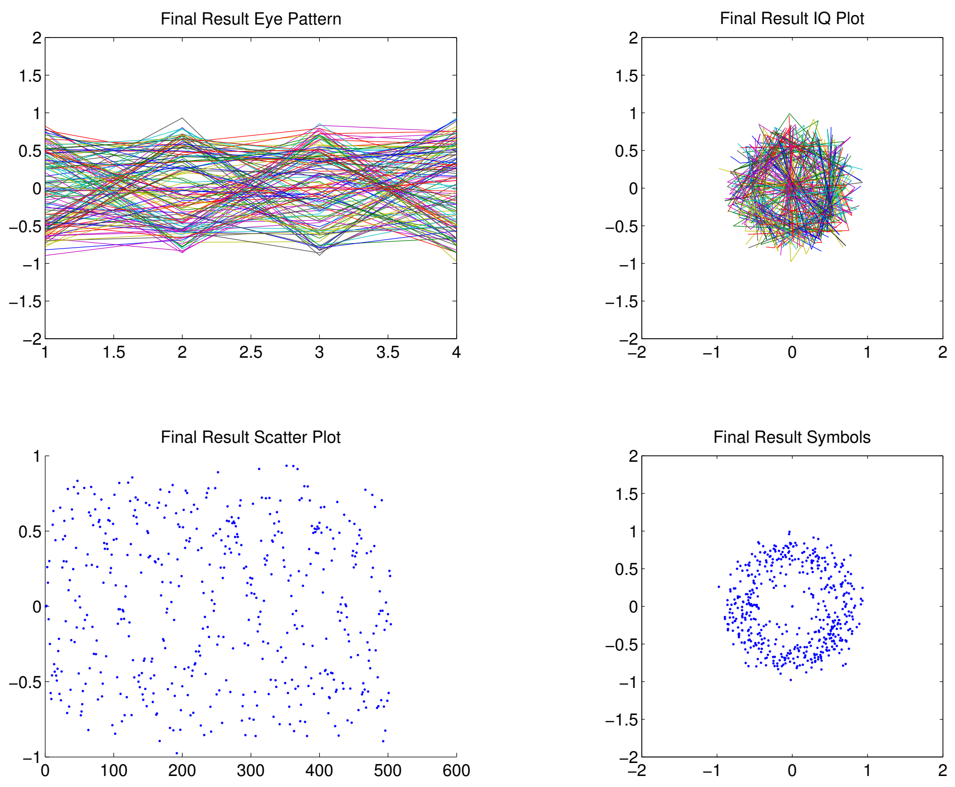

Fig. 3.22: Processing Chain 0 Final Results

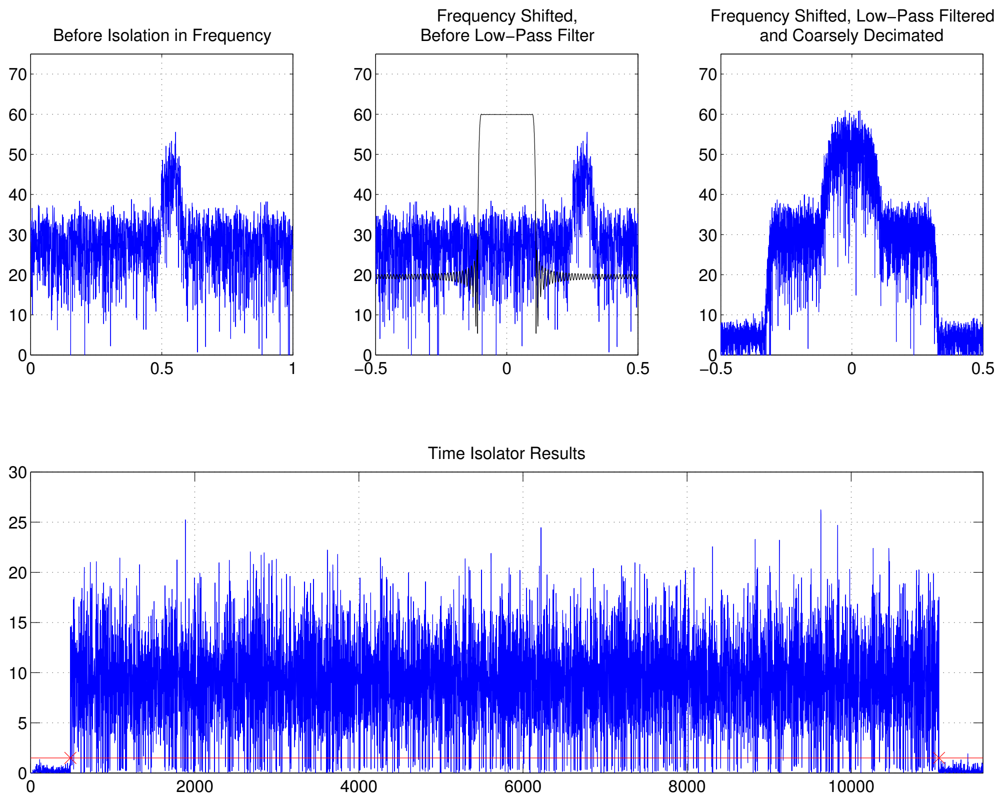

Fig. 3.23: Processing Chain 2 Isolators

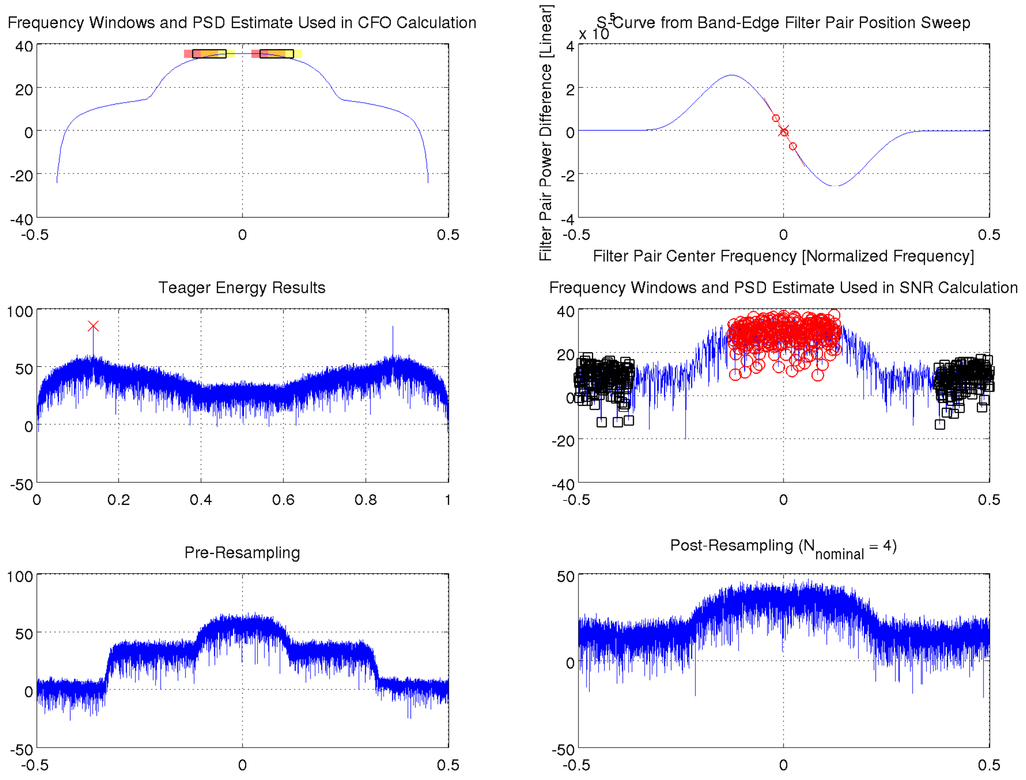

Fig. 3.24: Processing Chain 2 Estimators

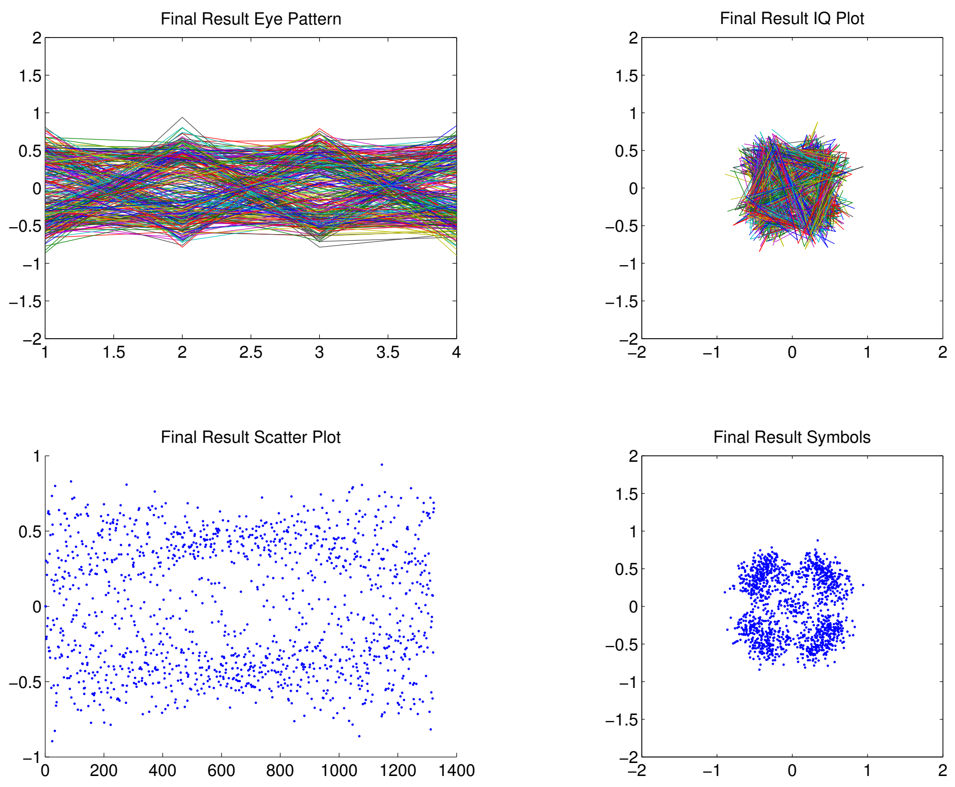

Fig. 3.25: Processing Chain 2 Final Results

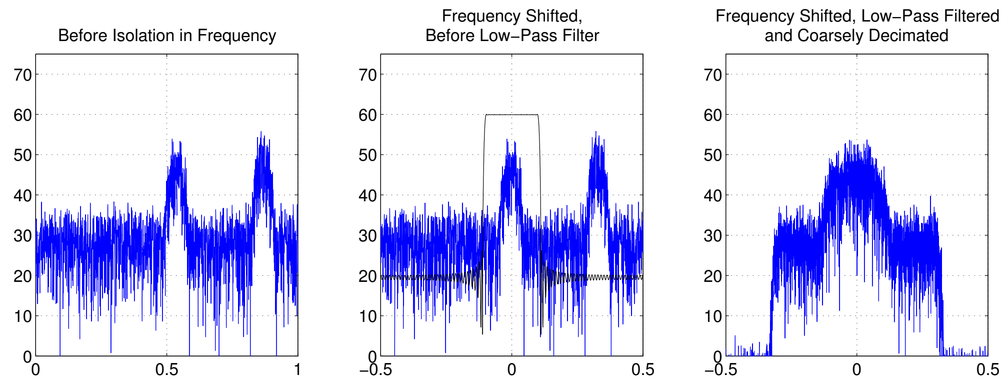

Fig. 3.26: Processing Chain 4 Isolators

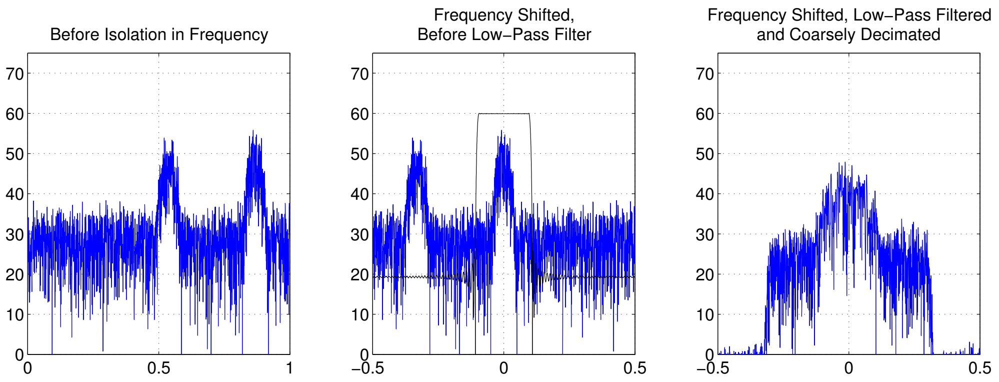

Fig. 3.27: Processing Chain 5 Isolators

The companion program was designed to be like unto an oscilloscope in software form. In the C code, there are code blocks that can be excluded at compile time that are at strategic locations that dump various signals at various stages in the processing chain. The output of these special code blocks are fed into the companion program which parses it and shows a play-by-play animation of the C code at different stages. Some key plots from the companion program are the main plot which displays the input signal's spectrogram, the current data chunk in view on the spectrogram and the detector's view of the current data chunk along with its threshold and currently detected frequency ROIs. This provides the user feed back as to whether or not the threshold value used on the input signal is good or not and whether or not the detector is operating as desired. The detector's sensitivity is adjusted through input parameters at run time which affect its PSD estimator's lowpass filter and smoother's degree of blur. The smaller the cutoff of the PSD estimator's LPF, the slower the detector will be at detecting burst start and stop times (increasing risk of merging in time). The greater the degree of blur in the smoothing of the PSD, the more smearing in frequency occurs (increasing risk of merging in frequency). These parameters are configurable at run time and can be inputs to the executable for convenience.

3.4 Performance Analysis

Results from the MATLAB companion program such as that in section 3.3.1 provide a good method for assessing the quality of the output for a particular run of the system. In order to get an idea for how the system works on average under certain conditions, a MATLAB script was created to repeatedly generate a signal, run it through the system and compute the error of the output and average the errors. The results of such an analysis are included in chapter 4. This MATLAB script can easily be changed to view system behavior with varying parameters such as SNR as in chapter 4, bandwidth, burst length, etc.

Chapter 4

Results

This chapter provides some results of a few tests to verify the functionality as well as assess the quality of the system under simplified conditions. The simplified conditions are such that the wide-band signal contains only one signal with only one burst as seen in figure 4.1. This type of input is what was used for section 4.2. The method of generating the root mean squared (RMS) errors is described in that same section. The system parameters and configuration settings for all the results are outlined in section 4.1.

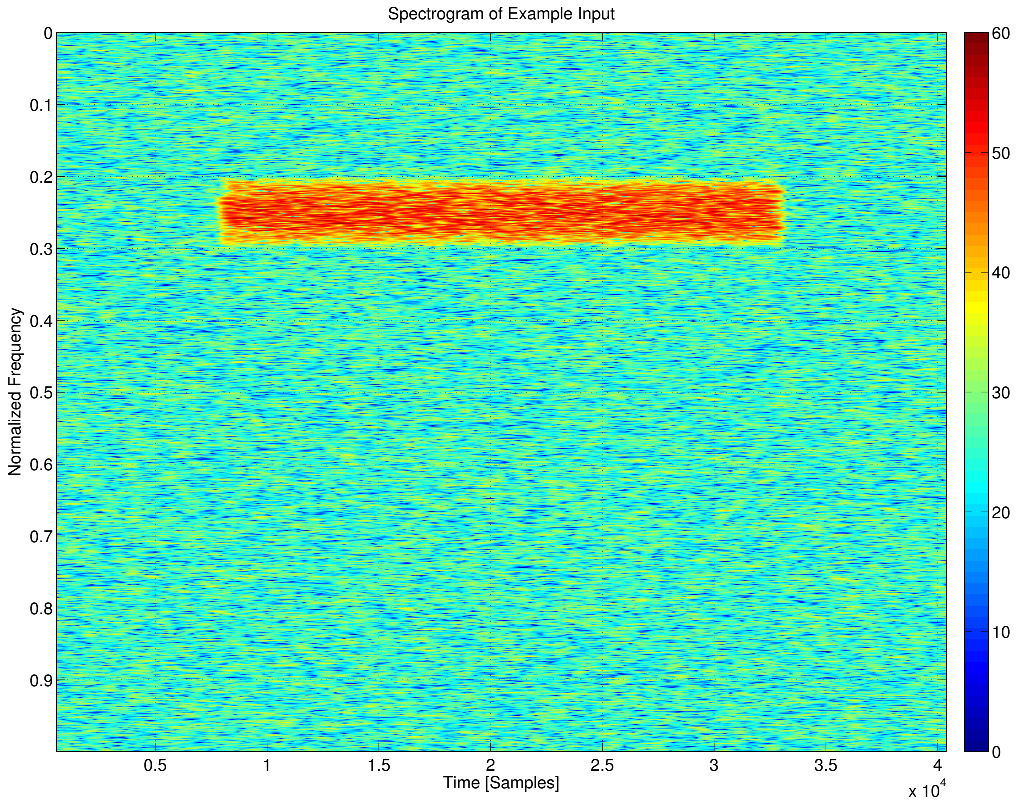

Fig. 4.1: Example input for generating RMS error plots under simplified conditions

4.1 System Configuration

The system parameters used in generating the results in this chapter are summarized in table 4.1. Because of the way the test signals (such as figure 4.1), there is a scale factor on some of the parameter estimates including the SNR and the high and low frequency edges. The test signal has a unit energy noise floor and the signal energy is scaled based on the desired SNR value. Because the threshold is held constant regardless of the SNR (which is unknown to the system) as SNR increases, the scale factor on the estimates become apparent. See figures 4.2 through 4.6 for example. There is also a bias on the SNR estimates. The scale factor and bias are compensated for by subtracting out the bias from the resulting estimate and dividing out the scale factor from the resulting estimate. To improve the estimates from the frequency edge estimators, small biases and scale factors were applied to every estimate. The bias for both was 1e-5 and a 1.0201 scale factor was for the low frequency edge and a 0.9835 scale factor was for the high frequency edge estimate.

4.2 Root Mean Square Error under Simplified Conditions

To generate the root mean square errors in figures 4.2 through 4.6, a MATLAB script generated a fixed length signal at a fixed carrier frequency and band-width for each modulation scheme. The signal power was scaled in increments and then run through the system which outputted estimated parameters including (starting with the top row and moving to the right and continuing with the lower left and onward towards the right again) the carrier frequency estimate ($\widehat{f_c}$), the low frequency edge estimate ($\widehat{f_l}$), the high frequency edge estimate ($\widehat{f_h}$), the SNR ($\widehat{SNR}$), the estimated time in which the burst turns on in units of samples ($\widehat{t}_{\text{on}}$) and the estimated time in which the burst turns off in units of samples ($\widehat{t}_{\text{off}}$). To get a good average of the error at each SNR value, a new signal was generated 100 times so these RMS errors are for 100 iterations.

Table 4.1: Table of Parameters Used in Simulation

Parameter Name	Value	Units
Input Buffer Size	1024	Samples
Old Input Buffer Size	1024	Samples
Max Burst Length	10240	Samples
Detector's PSD Estimator's FFT Size	1024	Samples
Detector's PSD Estimator's α	0.0125	(unitless)
Detector's PSD Estimator's Smoothing Enable	True	Boolean
Detector's PSD Estimator's Smoothing α	0	(unitless)
Detector's PSD Estimator's Smoothing No. Neighbors	10	Samples
Detector's ROI Finder's Min Hump Width	20	Samples
Detector's ROI Finder's Max Trough Width	20	Samples
Detector's ROI Finder's Frequency Padding	0.01	Normalized Frequency
Detector's ROI Finder's Threshold	3000	Power [Linear]
Main Controller's Margin or ϵ	0.05	Normalized Frequency
Frequency Isolator's N_{target}	10	Samples per Symbol Period
Time Isolator's ROI Finder's Min Hump Width	20	Samples
Time Isolator's ROI Finder's Max Trough Width	20	Samples
Time Isolator's ROI Finder's Frequency Padding	N/A	N/A
Time Isolator's ROI Finder's Threshold	1.5	Instantaneous Energy [Linear]
CFO's PSD Estimator's FFT Size	1024	Samples
CFO's PSD Estimator's α	0.99	(unitless)
CFO's PSD Estimator's Smoothing Enable	True	Boolean
CFO's PSD Estimator's Smoothing α	0	(unitless)
CFO's PSD Estimator's Smoothing No. Neighbors	128	Samples
Resampler's N_{target}	4	Samples per Symbol Period
SNR Estimator's PSD Estimator's FFT Size	1024	Samples
SNR Estimator's PSD Estimator's α	0	(unitless)
SNR Estimator's PSD Estimator's Smoothing Enable	False	Boolean
SNR Estimator's PSD Estimator's Smoothing α	N/A	N/A
SNR Estimator's PSD Estimator's Smoothing No. Neighbors	N/A	N/A

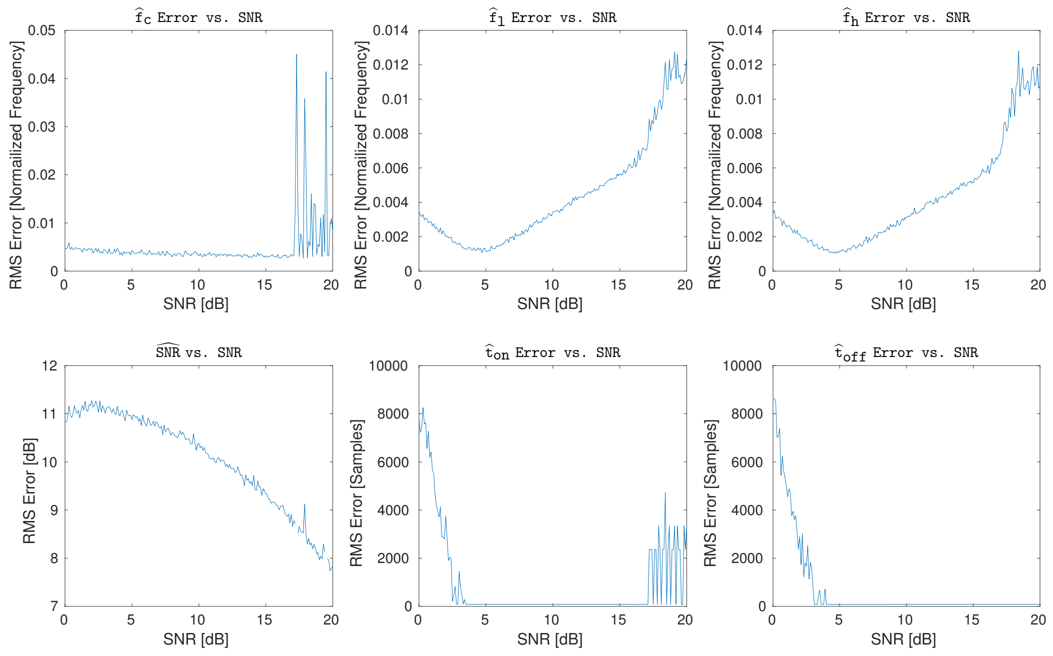

Fig. 4.2: RMS Error for Binary PAM

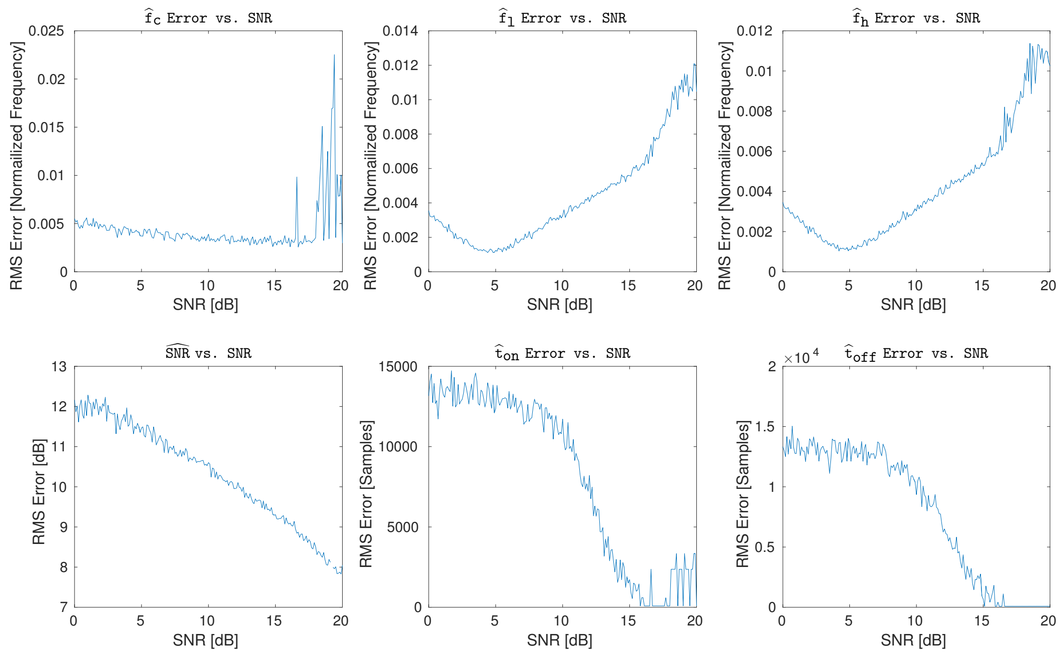

Fig. 4.3: RMS Error for Four PAM

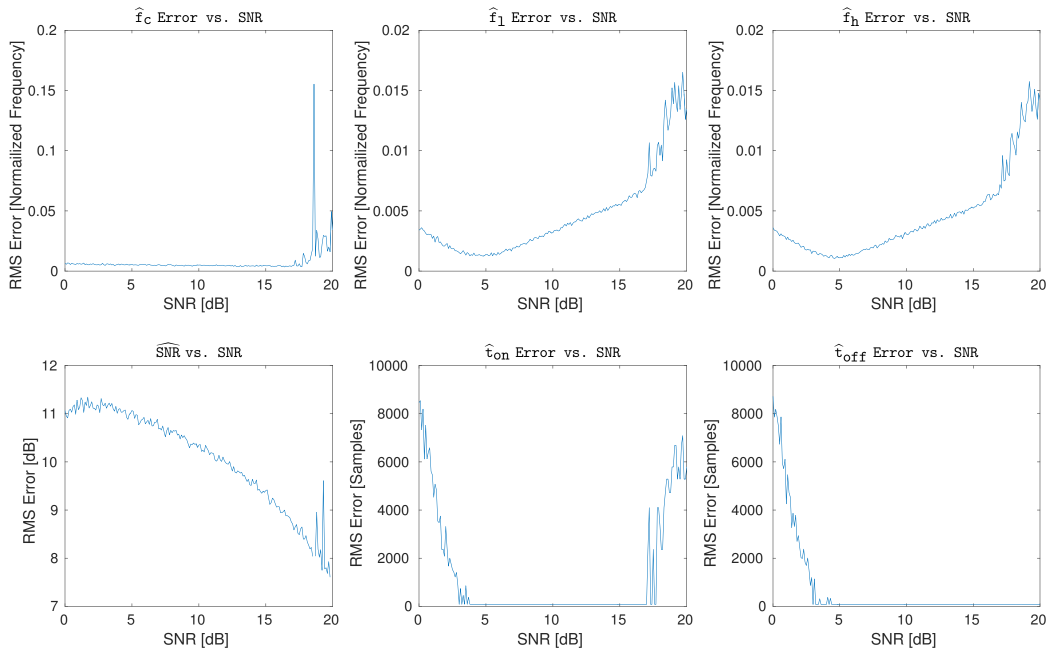

Fig. 4.4: RMS Error for Four QAM

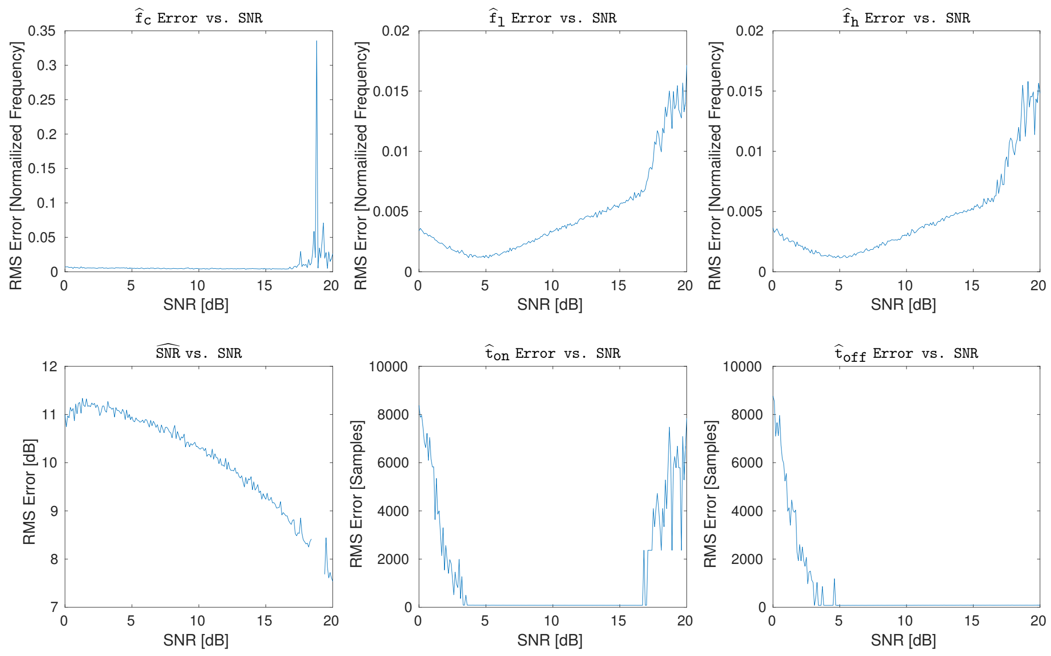

Fig. 4.5: RMS Error for Eight PSK

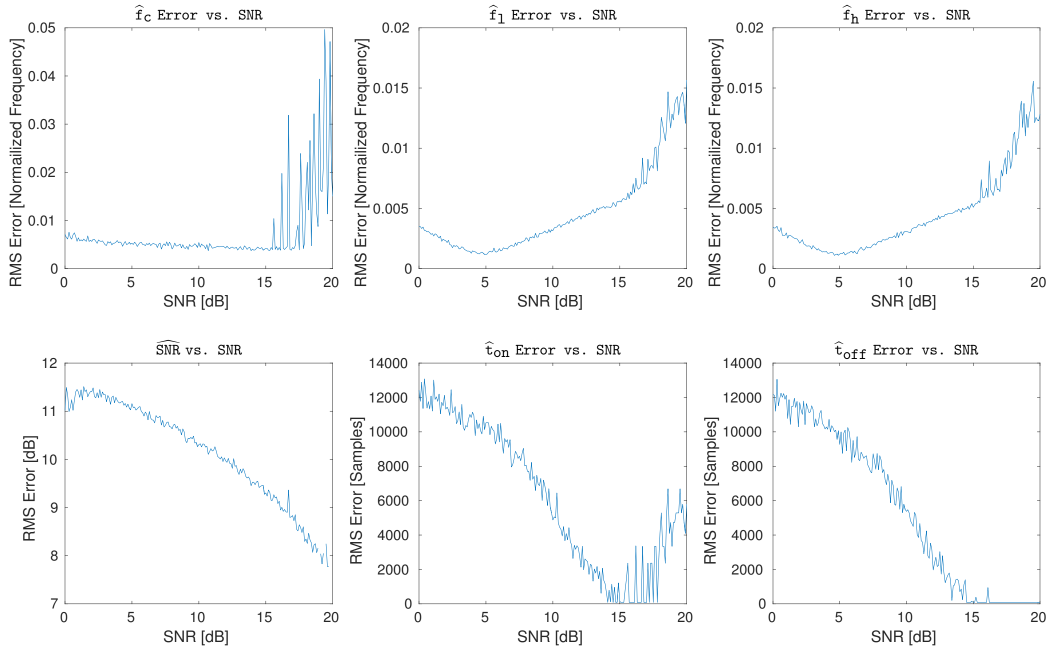

Fig. 4.6: RMS Error for Sixteen QAM

Chapter 5
Conclusion

As seen in the results section, the system provides data fit for automatic modulation classification because the pre-processing and signal conditioning is done. The system outputs signals that are centered at DC within roughly one-hundredth of normalized frequency. Occasionally the constellation of the system output gives a good clue as to what the modulation scheme is as seen in figure 3.25 which appears to be 4-QAM. As explained in the results section the reason for the increase in error with an increase in SNR is because of the way the test signals were generated. All in all, it can be concluded that the system is capable of blindly pre-processing a wide-band signal, outputting a collection of base-banded, time-isolated and nominally sampled signals for each detected burst in real-time.

5.1 Future Work

Although this system accomplishes its minimum requirements, there is much work that needs to be done. Features that were intended but not permitted due to time include supporting stream signals (i.e. long lasting bursts or continuous bursts). Also, after the SNR Estimator, a blind equalizer would be useful to the AMC unit as a pre-filter to ensure all inputs to the AMC device have the same SNR for consistency. It would be nice if this system had the capability of deriving the thresholds used in the detector and the time isolator. This work assumed a priori knowledge of the noise floor. Also, to reduce the number of sine calculations, a complex exponential look up table would be a quick and easy performance improvement for the frequency isolator. Sophisticated error handling would be the up-most priority on the next iteration of the C code implementation. Having the ability to detect faults and discard erroneous signals and avoid crashes would be very handy. Things that would need to be handled are false detections, smearing in frequency and in time, etc. Further support for non-linear modulation schemes such as FSK also need to be incorporated.

References

[1] M. Rice, *Digital communications: A Discrete-Time Approach.* Pearson Education India, 2009.

[2] O. Dobre, A. Abdi, Y. Bar-Ness, and W. Su, "Survey of automatic modulation classification techniques: Classical approaches and new trends," *Communications, IET*, vol. 1, no. 2, pp. 137–156, April 2007.

[3] A. Hazza, M. Shoaib, S. Alshebeili, and A. Fahad, "An overview of feature-based methods for digital modulation classification," in *Communications, Signal Processing, and their Applications (ICCSPA), 2013 1st International Conference on.* IEEE, 2013, pp. 1–6.

[4] J. A. Suris Pietri, "Rapid Radio: Analysis-based receiver deployment," Ph.D. dissertation, Virginia Tech, 2009.

[5] A. Recio, J. Surís, and P. Athanas, "Blind signal parameter estimation for the Rapid Radio framework," in *Military Communications Conference, 2009. MILCOM 2009. IEEE.* IEEE, 2009, pp. 1–7.

[6] A. Recio, J. Suris, and P. Athanas, "Automatic modulation classification for rapid radio deployment," in *Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on*, June 2010, pp. 1–7.

[7] S. Z. Khan, A. Mostayed, and M. E. Kabir, "A frequency blind method for symbol rate estimation," in *Future Intelligent Information Systems.* Springer, 2011, pp. 607–614.

[8] R. Hamila, J. Astola, F. Alaya Cheikh, M. Gabbouj, and M. Renfors, "Teager energy and the ambiguity function," *Signal Processing, IEEE Transactions on*, vol. 47, no. 1, pp. 260–262, Jan 1999.

[9] J. A. Suris Pietri, "Rapid Radio: Analysis-based receiver deployment," 2009.

[10] M. Frigo and S. G. Johnson, "FFTW: An adaptive software architecture for the FFT," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3. IEEE, 1998, pp. 1381–1384.

[11] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Prentice-hall Englewood Cliffs, 1989, vol. 3.

[12] F. Harris, E. Venosa, X. Chen, and C. Dick, "Band edge filters perform non data-aided carrier and timing synchronization of software defined radio QAM receivers," in *Wireless Personal Multimedia Communications (WPMC), 2012 15th International Symposium on*, Sept 2012, pp. 271–275.

[13] J. Kaiser, “Some useful properties of Teager’s energy operators,” in *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, vol. 3, April 1993, pp. 149–152 vol.3.