20

MULTI-STANDARD DIGITAL FM RECEIVER USING LIMITED IF ARCHITECTURE

by

Grant Y. Smith

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 22, 1998 [Ju[illegible] 1998]

Author ______________________
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by ______________________
J. K. Roberge
Thesis Supervisor (M.I.T.)

Certified by ______________________
Saed Younis
Thesis Supervisor (QUALCOMM, Inc.)

Accepted by ______________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Multi-Standard Digital Fm Receiver Using Limited If Architecture
by
Grant Y. Smith

Submitted to the Department of Electrical Engineering and Computer Science

May 22, 1998

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This document presents a proposal for the development of a digital FM receiver using limited IF architecture and frequency delta sigma demodulation for demodulating signals from both the North American AMPS standard and the Japanese TACS standards. This is a desired goal in order to ease the effort and complexity of FM demodulation, by incorporation into the digital domain. Oversampling delta sigma technologies are well suited to this purpose, and can apply both to the FM detection process and to the decimation necessary to convert to baseband. An exploration of this as it applies to the Japanese JTACS and NTACS standards provides us with a marketable and useful system that can be an effective and useful tool.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

The problem of FM demodulation in modern communications systems presents an environment quite conducive to an examination of many varied communications technologies. In particular, there is a commercial interest in performing frequency-to-digital conversion at the intermediate frequency (IF) stage of the receiver so as to perform channel filtering, gain control, and demodulation in the digital domain. The use of digital filtering and processing techniques is advantageous in a number of ways. Analog IF filters are large and costly, and may also have poor phase performance, while digital filters are cheap to implement, can have precisely linear phase and are far more reliable. Digital circuitry is also far more robust in manufacturing, as well as being more readily altered in future generations as changes are made. However, a digital receiver requires very precise analog to digital performance in traditional demodulation schemes, leading to an examination of oversampling ΔΣ techniques towards these demodulation techniques. ΔΣ techniques for analog-to-digital conversion and FM demodulation can have a tremendous impact by shaping error in such a way that the majority of the error be passed out of the relevant signal band, producing great increases in performance. The additional use of hard limited signals introduces its own benefit with regards to diminishing analog processing, and with it, cost.

FM demodulation schemes must also be regarded in the context of the situation in which they are to be used. FM communications standards have been set in place throughout the globe, carrying with them the expected diversity of specifications. Among these, the Japanese JTACS and NTACS standards are of particular interest to American manufacturers due to the tremendous market and interest in cellular communications. It is therefore of great advantage to develop a system of FM demodulation with functionality across American and Japanese cellular telephony standards, in such a way as to appeal to both markets with the same product.

This thesis, therefore, attempts to merge these two major concepts through the development of a multi-standard digital FM receiver using limited IF architecture. The standards which will be considered include the aforementioned Japanese JTACS and NTACS standards, along with the commonly used American AMPS standard. A key aspect of the work involves developing a demodulator with as much of its processing in the digital domain as possible, and the tradeoff inherent to that will be examined. Also, an exploration of the ΔΣ techniques involved in our limited IF approach will be considered, along with other ΔΣ approaches.

The examination of such a system will take place primarily in a simulation environment using software tools. The C programming language and the mathematical tool Matlab will be the primary tools used in this analysis, as the demodulation scheme, and then the entire receiver, are simulated and examined in this manner.

The work which makes up this MIT Masters of Engineering thesis was performed in part through the MIT VI-A program, working at QUALCOMM, Inc. in San Diego, CA, and thus supervised jointly by a company supervisor and an MIT supervisor. As such, aspects of the work that involve commercial applications and alignment to specific systems are generally based upon their products.

1.1. Thesis Organization

This thesis is organized into eight chapters. This first chapter provides a general overview of the goals and intentions contained herein. The second chapter is an examination of FM demodulation concepts, beginning with a background of FM modulation and demodulation and continuing with some examinations of traditional demodulation schemes. The third chapter is a look at delta sigma modulation, its past, its intentions, its advantages, and its application to frequency demodulation. The fourth chapter is a look at several specific digital demodulation techniques applied to converting a limited IF FM signal to a baseband demodulated signal. The fifth chapter details a specific set of simulations on one of these techniques and its application to our situation. The sixth chapter is a description of the various cellular FM standards we are concerning ourselves with, and the seventh chapter is a detailed description of the complete simulation of the system as it performs for all three standards. The eighth chapter summarizes our conclusions and considers the progress made and future available improvements. Finally, the code used for the simulations can be seen in the appendices.

Chapter 2. FM Demodulation Concepts

As a basis for an understanding of the purpose of this work, an exploration of the basics of frequency modulation and demodulation is necessary. The concept of modulating signals and the methods and benefits of frequency modulation are therefore touched upon here, with particular regard for those aspects which have bearing upon this work. This will also provide a proper forum for characterization of the benefits garnered from the methods proposed in this work.

This chapter, therefore, begins with the concepts of frequency modulation. It continues with a discussion of a commonly used detection technique involving quadrature channel separation, illustrating its basis and benefits. In conclusion, groundwork is set in place for the discussion of limited IF zero crossing approaches later in the thesis.

2.1. Theory

Modulation is a basic principle in communications and a necessary element of all communication schemes, and at its heart is based in the concept of frequency shifting a signal to a new frequency range. This requires two main factors: a source-output signal we'll denote as *m(t)*, and a carrier signal we'll denote as *c(t)*, of the form:

$$c(t) = A_c\left(\cos 2\pi f_c t + \phi_c\right)$$

where A_c is the amplitude, f_c is the carrier frequency, and ϕ_c is the carrier phase. This illustrates the three methods of modulation, where a signal *m(t)* may modulate the carrier *c(t)* in amplitude, frequency, or phase [14].

There are three main benefits of modulation. The first is the translation of a signal from a lowpass situation to a bandpass signal in the frequency of the passband of the channel, allowing for transmission. The second is the introduced ability to accommodate multiple signals in the same transmission medium, through frequency multiplexing and the allocation of signals to separate channels within the medium. The third is the primary benefit of phase and frequency modulation as opposed to amplitude modulation, and is the expanding of the bandwidth of a transmitted signal for the purpose of increasing noise immunity in noisy channels.

Angle modulation, such as frequency modulation (FM) and phase modulation (PM) are based upon the principle of changing the frequency or phase of the carrier based upon the amplitude of the modulating signal. Unlike amplitude modulation (AM), these are inherently non-linear methods, and involve a distinctly more complex problem with regards to modulation and detection. The generation of a PM signal can be accomplished through the following equations [12]:

$$y(t) = A\cos\phi(t)$$

$$\phi(t) = 2\pi f_c t + k_p m(t)$$

while an FM signal behaves according to the following equations:

$$y(t) = A\cos\phi(t)$$

$$\frac{d\phi(t)}{dt} = 2\pi f_c + k_f m(t)$$

It is clear from these that an FM modulation scheme is equivalent to an integration of *m(t)* followed by a PM modulator, and that a PM modulation scheme is equivalent to differentiation of *m(t)* followed by an FM modulator. Indeed it is this first approach that is used in our simulations later on for the purpose of FM modulating a signal. Also to note in these equations are the scaling factors k_p and k_f, which are the phase and frequency deviation constants, respectively. These deviation constants represent the degree to which the phase, or frequency, will vary per unit change on the input signal, and are key components in FM systems.

This PM discussion is brought up primarily as a means of grounding our FM discussion. The formulas for PM are somewhat clearer and more intuitively obvious than those for FM, although FM modulation examples illustrate the concept of frequency modulation quite clearly, such as in Figure 2-1, in which we can see a simple square wave and the correlating FM modulation. Notice how the *frequency* of the modulated wave changes in proportion to the *amplitude* of the input signal. Notice also how the amplitude of the FM wave does not change. Indeed, there is no information contained in the amplitude of an FM signal, a tremendous source of noise immunity.

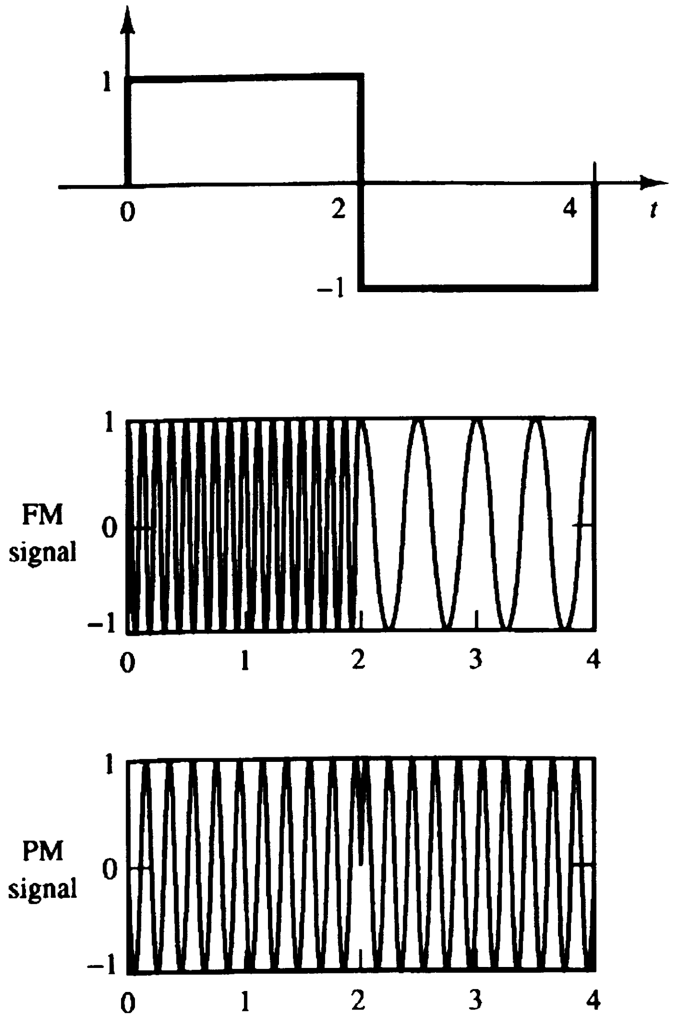


Figure 2-1: FM Modulation Example

2.2. Modulation

FM modulation is generally performed in one of two ways. The first is by directly varying an element of a tuned circuit oscillator, or voltage controlled oscillator (VCO). This is a circuit whose frequency varies directly with input voltage, with a zero input voltage resulting in a sinusoid with frequency f_o. The second is to integrate the input frequency and then to perform PM on the resultant signal. Our discussion and concern lies primarily in the demodulation realm, and as such, our simulation of an FM modulator is simply for the purpose of generating a valid and useful input, and will therefore be done directly from the formulas for FM modulation and through integration of the signal.

2.3. Demodulation

Demodulation of FM is a considerably more interesting problem, with a number of challenges and related solutions presented. There are two fundamental steps involved. The first is the actual detection process, sometimes known as discrimination, which serves to convert frequency deviation into amplitude. The second is essentially AM demodulation, or the conversion to the appropriate frequency range, generally baseband. This general scheme can be seen in Figure 2-2. A few popular approaches to this problem will be discussed.

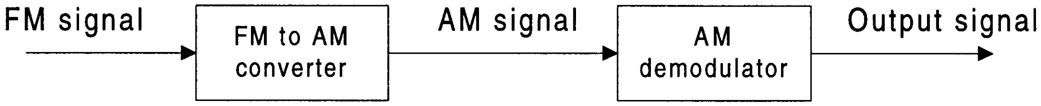


Figure 2-2: General FM Demodulation Scheme

2.3.1. General Demodulation Techniques

The first category of FM to AM conversion schemes we will consider is based upon the desired characteristic of

$$|H(f)| = 2\pi f$$

which will produce an output proportional to frequency, as desired. This can be accomplished with a simple differentiator with such a characteristic, although that may be unwieldy. Another possibility is a tuned circuit with characteristics as shown in Figure 2-3, whereby the desired response is designed to exist across the required band. Examples of these circuits are the "slope detector" parallel LC circuit tuned to the appropriate point, and the Foster-Seely detector, which adds a diode arrangement for increased performance[19].

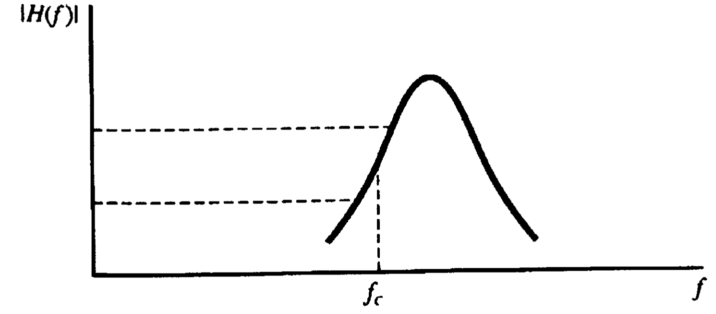


Figure 2-3: Tuned Circuit Used in an FM Demodulator

The second major style of FM to AM conversion schemes involves feedback for the sake of reducing the bandwidth of the FM detector. The first of these is the FM demodulator with feedback (FMFB), as seen in Figure 2-4. This places FM discrimination between a bandpass filter and a lowpass filter, with a voltage controlled oscillator (VCO) on the feedback path. The bandwidth of the discriminator and lowpass filter are designed to match the bandwidth of *m(t)*, and the feedback path performs the AM demodulation as well by eliminating the high frequency modulation [14].

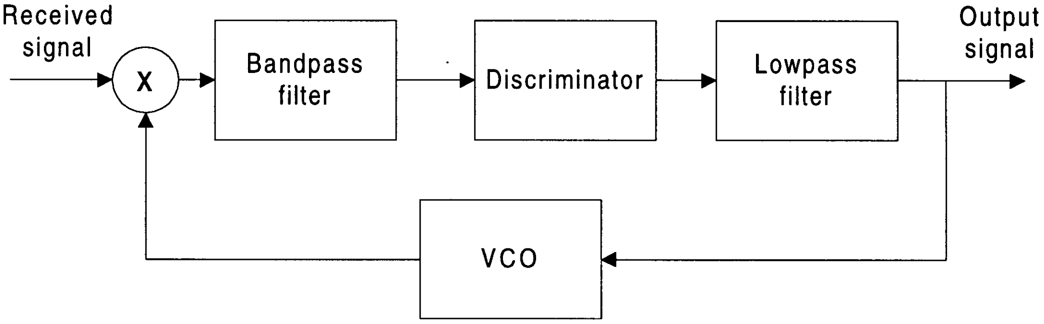


Figure 2-4: FMFB Block Diagram

The second feedback route is the use of a PLL-FM demodulator. A phase locked loop (PLL) is a device that varies the frequency of a voltage-controlled oscillator to match in input frequency. This requires a phase comparator, a loop filter and a VCO, and can be seen in Figure 2-5.

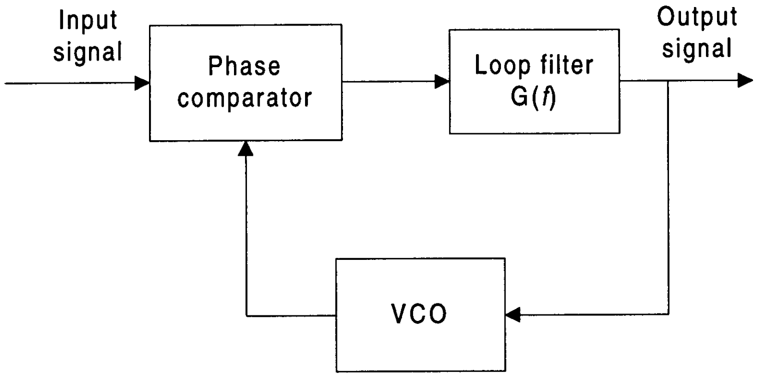


Figure 2-5: PLL-FM Demodulator Block Diagram

Its functionality is as follows: The phase comparator produces an output which is the difference in phase between two input frequencies. That error is filtered, with bandwidth large enough to pass the modulating signal, as necessary in a feedback situation, and used as negative feedback for the VCO, such that the output voltage tracks the frequency of the input, and is therefore a demodulated version of the input signal [19],[14].

2.3.2. I, Q Channel Separation Technique

The FM demodulation technique we'll examine in some more depth here is a quadrature FM detection technique, and the one that can be considered as a base point for our future explorations. The technique will be presented from a general standpoint, and then examined in some more detail.

2.3.2.1. Concept and Implementation

The concept of this FM demodulation technique is the separation of the FM signal into quadrature (or orthogonal) I and Q channels, and then using those channels to determine the phase of the signal, and, from the phase, the frequency signal. At this point we'll begin to consider the costs of the elements required, and the separation between the analog and digital hardware at hand.

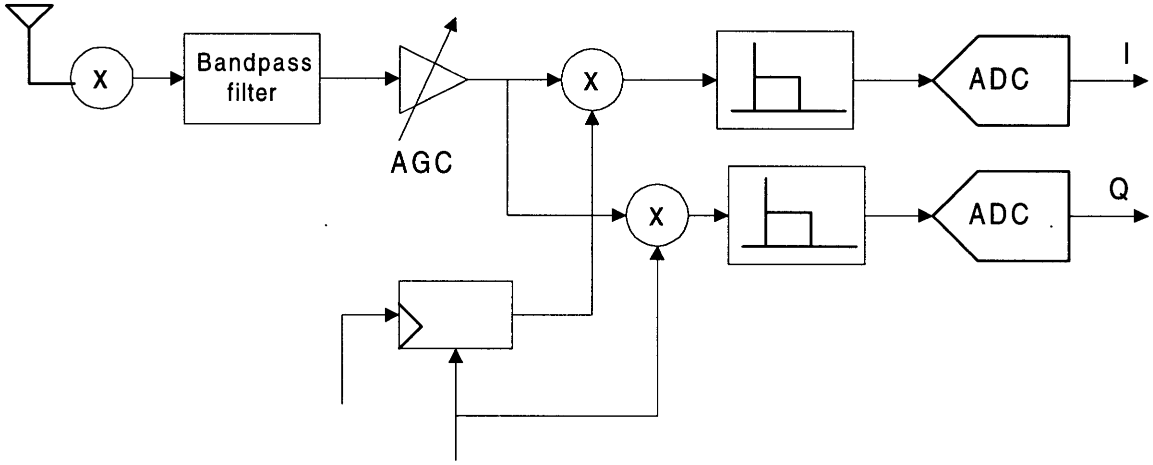


Figure 2-6: I, Q Channel Separation Technique Block Diagram

The general block diagram for this system can be seen in Figure 2-6. At the far left, the start of the signal path, we see an abstraction of the RF system in the antenna, passing the system to our demodulator circuit. Following this is a tunable mixer to bring the relevant channel to IF, such that our frequencies from here on can be independent of the channel selected. Next is a bandpass filter to eliminate other channels, which can be implemented as a fixed bandpass filter rather than a tunable one due to the mixing to a constant IF. The next stage is an automatic gain control (AGC), which is necessary due to the analog to digital conversion later in the system. The analog to digital conversion must be very precise, and therefore requires that our input signal level be precisely tuned to the range of the ADC, without surpassing the range of that ADC. The next stage is I, Q separation. The formulae for I, Q separation are as follows:

$$I(t) = x(t) * \cos\left(2\pi \frac{f_c}{f_s}\right)$$

$$Q(t) = x(t) * \sin\left(2\pi \frac{f_c}{f_s}\right)$$

such that a sampling frequency of $f_c/4$ will simplify I, Q separation to multiplication times a sequence of {0,1,0,-1}, shifted in phase between I and Q. This simplification is a tremendous gain in an actual system, assuming the choice of sampling frequency is acceptable. The I, Q separation in Figure 2-6 is shown in this type, with the multiplicative factor delayed with a register. Following this is a lowpass filter to handle the transients introduced by this separation, and then the ADCs referred to earlier. The remaining processing is in digital hardware, and therefore considerably cheaper to implement. The I and Q signals are then translated into the phase of the inputted signal. This is accomplished by taking the arctangent Q/I, which is clearly the phase when seen in I, Q space. This arctangent is placed into the proper quadrant according to the actual values of I, and Q, and then the derivative is taken in order to recover the frequency from the phase.

Our final result is therefore proportional to the frequency of the transmitted signal, or equivalent to the original signal.

2.3.2.2. Optimizations and Limitations

The primary aspects to be aware of in a system such as this are the relative complexity of the tasks at hand, the power consumption in reasonable implementations of them, and the feasibility of less accurate estimations of parameters. The separation of I and Q using the method looked at above is a clear example of a great gain in simplicity. The IF frequency can be maintained as a fixed parameter, regardless of the chosen FM carrier, and thus an appropriate sample rate can be fixed upon to comply with this parameter. And, the implementation of a simple +/-1 or 0 multiplier is trivial, especially compared to adding in separate local oscillators to perform the same function. Also, the calculation of the arctangent of the quotient of Q and I can be found with the use of a $1/8^{th}$ size lookup table as would be otherwise necessary if the symmetry of the arctangent function is taken advantage of. This is to say, arctan((min $|I|$, $|Q|$) / (max $|I|$, $|Q|$)) can be calculated, and then adjusted to the appropriate quadrant based on the three Boolean tests $|I|>|Q|$, $|I|>0$, and $|Q| > 0$.

Of concern, on the other hand, are those aspects of this design which can be improved upon. The primary one of these is the considerable amount of analog domain processing. Analog filters, analog mixers, and the AGC are all expensive elements to incorporate into a design. The AGC as well, required to keep the signal range within the linear portion of the ADCs, is likely to be very expensive in size, power, and design, and is further seen as an unnecessary element due to the fact that there is no amplitude information in the FM signal, and therefore no particular reason to be so careful maintaining the amplitude. This is therefore to become a key element to eliminate in the following designs.

2.3.3. Limited IF Zero Crossing Counting Approach

A category of demodulation schemes with considerable appeal involves hard limiting the IF signal, and then using zero crossing counting approaches. This is possible, and indeed desirable due to the fact that the FM signal incorporates all its information in its frequency, and there is no information at all in the amplitude. Therefore, by converting a varying signal into a square wave with all the information in the location of the zero crossings, a new set of possibilities open up. A few things are important to keep in mind here. First, that the new signal remains an analog signal, although it does indeed resemble a digital square wave. This is to say, the time of the zero crossing is continuously variable, and distortion of this signal will occur upon sampling because of the introduced quantization of where the zero crossing occurs. The lower the sampling rate, the more distortion that will be introduced. The other important element to remember is the fact that the proper channel must be isolated by this point. There cannot be any significant interference from adjacent or alternate channel information, as our ability to differentiate between that interference and our own signal will disappear when we hard limit. Therefore great care must be taken to ensure that we've properly filtered our signal by this point. Various IF limited approaches will be examined in detail in Chapter 4.

Chapter 3. Delta Sigma Modulation

An examination of the principles of delta sigma modulation will form a solid background for the discussion of the delta sigma based demodulation methods in Chapter 4. Delta sigma principles were originally developed for the purpose of analog to digital conversion, and grew out of delta modulator principles, and these will be presented as a means of understanding these concepts. However, delta sigma modulation is fundamentally a noise shaping mechanism, and can therefore be applied to increase signal-to-noise performance in frequency-to-digital conversion.

Before we begin, a few words about terminology. In the original paper on ΔΣ circuits[15], the term ΔΣ modulator was introduced, with "Δ" referring to Δ from delta modulation, and "Σ" referring to the pre-integration which is the defining difference between the two. The use of the term "modulator" is referring to the pulse-density modulating function of the encoder. However, the term "ΣΔ modulator" became prevalent after a series of papers published by Candy and his co-workers [2]. Also, we find the term "ONS" converter, referring to Oversampled Noise Shaping [16]. The term ΔΣ is the one primarily used in this thesis, in deference to the original terminology.

3.1. Quantization and A/D Conversion

The original problem that ΔΣ modulation was created for was the process of analog to digital conversion. A few of the existing methods for performing A/D conversion will therefore be looked at in contrast. However, ΔΣ converters operate under an entirely different principle. The basic models for a typical A/D converter and for a ΔΣ converter can be seen in Figure 3-1.

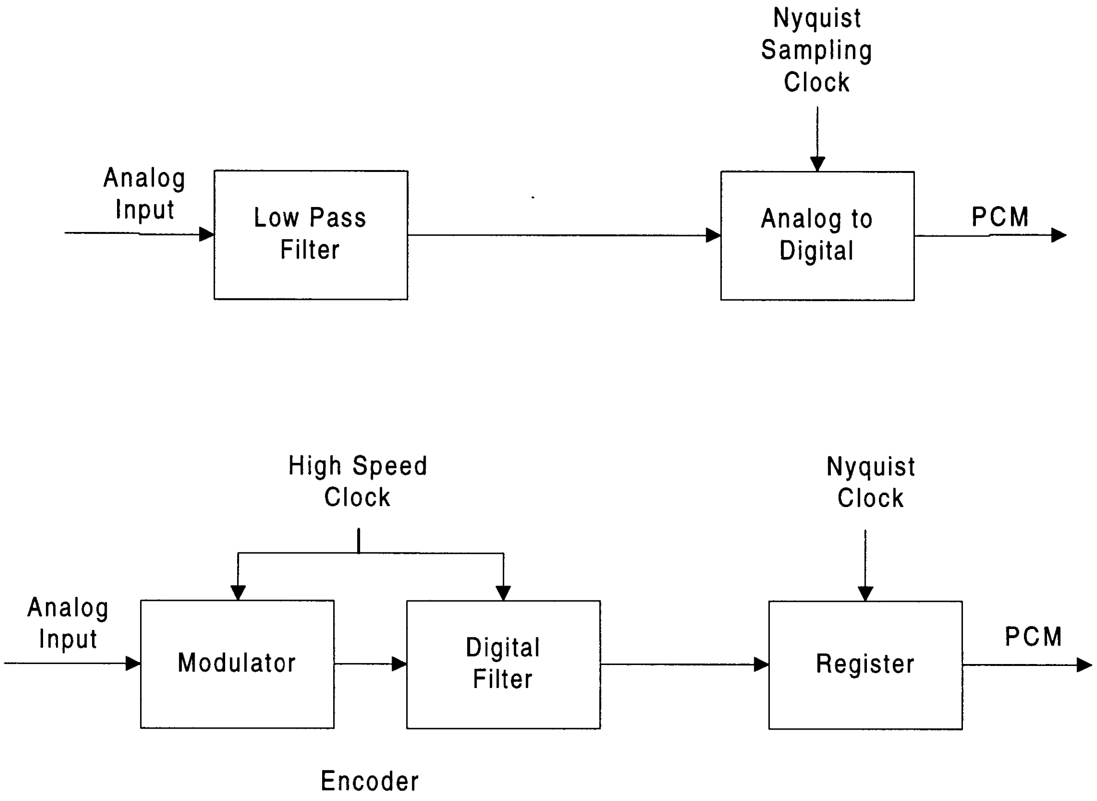


Figure 3-1: Generic models of A/D converter, ΔΣ converter

Generic A/D converters involve 2 steps, the first a lowpass anti-aliasing filter, the second the actual analog to digital conversion, with a sampling clock of at least the nyquist frequency of the signal. One common means of performing this conversion is the use of parallel-encoded or flash A/D converters, the fastest method available. These involve a series of 2^N resistors and a series of 2^N comparators, plus some control logic, and are implemented by creating 2^N equally spaced voltage levels between 0 and V_{ref}, each of which is compared against the input voltage. Effective because of their speed, they also do not require a sample and hold circuit to maintain the analog input, in the way that some ADCs do. The circuitry for a flash converter, however, grows exponentially with N [19], [16].

Another common ADC technique is the successive approximation converter, which achieves N bit resolution with only one comparator. This is done with a comparator, a register, a DAC, and some control logic. The process is as follows: All bits are set to some level, usually 0, and beginning with the MSB, each bit is set to one and the result compared against the input. This provides the knowledge of the proper status of that bit, which can then be held for the next iteration. In this way, each bit can be determined, one at a time, a process requiring only one comparator but N clock cycles. The need for a longer conversion time also makes the use of a sample and hold circuit advisable. [19], [16] As seen in Figure 3-1, ΔΣ converters, on the other hand, involve 3 steps, an encoding, a digital filtering, and a decimation. The encoding and digital filtering are performed at a considerably higher frequency than is desired of the output, with decimation occurring as the final step. It is as a result of this high oversampling that the benefits are gained, as the encoder, which is basically a coarse quantizer, is therefore far less stringent on its requirements than the quantizer used in the standard ADC. Also, once in the digital realm, a digital lowpass anti-aliasing filter can be implemented, considerably cheaper than analog filtering. The anti-aliasing filter for the initial quantization, due to its high sampling rate, can be quite general and fairly easy to develop. However, due to the large degree of oversampling necessary, ΔΣ conversion is best suited for relatively low frequency signals.

3.2. Delta Modulation

Delta Modulation is a logical starting point for our discussion, due to its role as a precursor to ΔΣ modulation. Two equivalent discrete-time models for the delta modulator are seen in Figure 3-2, for the sake of developing our understanding. In these figures, z^{-1} represents a delay element, and Q represents a quantizer, although clearly one with considerably less resolution than the modulator as a whole.

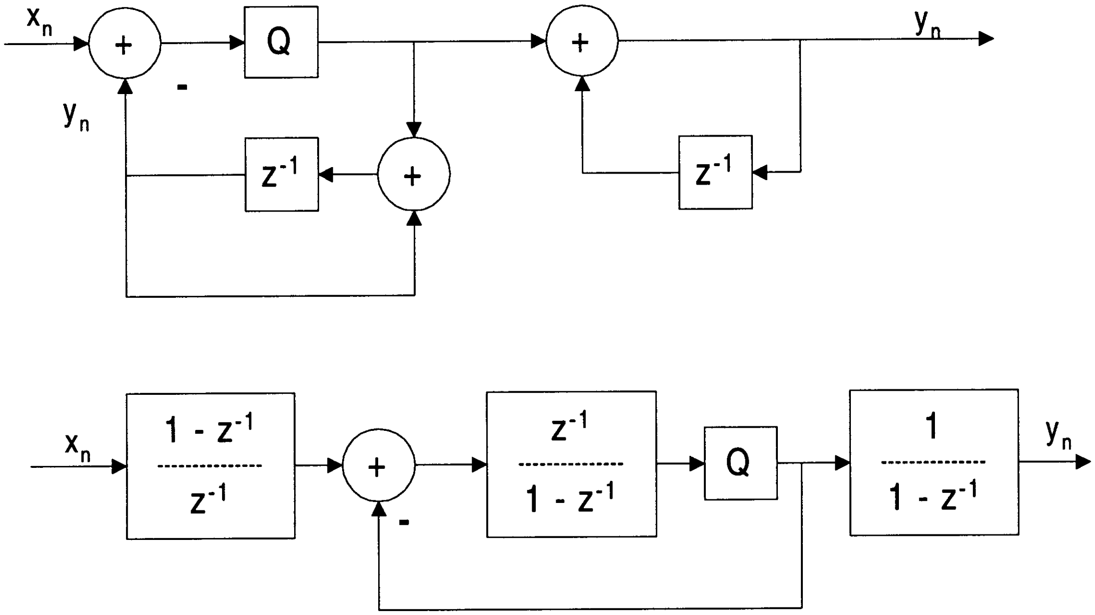


Figure 3-2: Delta Modulators

The first model illustrates the modulator's fundamental performance. The output of the quantizer is the difference between the input x_n and the estimation of that input. Assuming the predictions are close to the actual values of x_n, the error signal out is quite small, and can be quantized with a significantly smaller number of bits than would be necessary to quantize the entire sequence. That output is then integrated in the second half of the modulator to recover the estimated sequence, rather than just our error. The second model is functionally the same modulator, redrawn such that the input is first differentiated, and the output is integrated, immediately prior to the output of the entire system, but the central part is what is to become the concept of a ΔΣ modulator. These surrounding blocks are not necessary to the functionality and can be canceled out, leaving us with Figure 3-3, our basic, single-loop ΔΣ modulator.

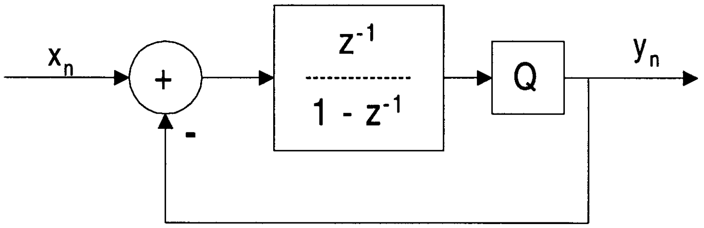


Figure 3-3: Single Loop ΔΣ modulator

3.3. Delta Sigma Modulation

The origination of the ΔΣ modulator was explored in section 3.2., but in this section, the performance of those modulators is explored. This is done primarily through an examination of the first order feedback circuit which is the basic ΔΣ modulator. It is through this that a proper sense of ΔΣ modulation is gained. The first order feedback ΔΣ modulator circuit of Figure 3-3, shown again in

Figure 3-4 in more detail, illustrates the various features of ΔΣ modulation we need to cover in this discussion. These qualities are the ease of implementation, the relative cheapness of the circuitry design and power requirements, and the effective noise-shaping characteristics gained through oversampling and error-tracking.

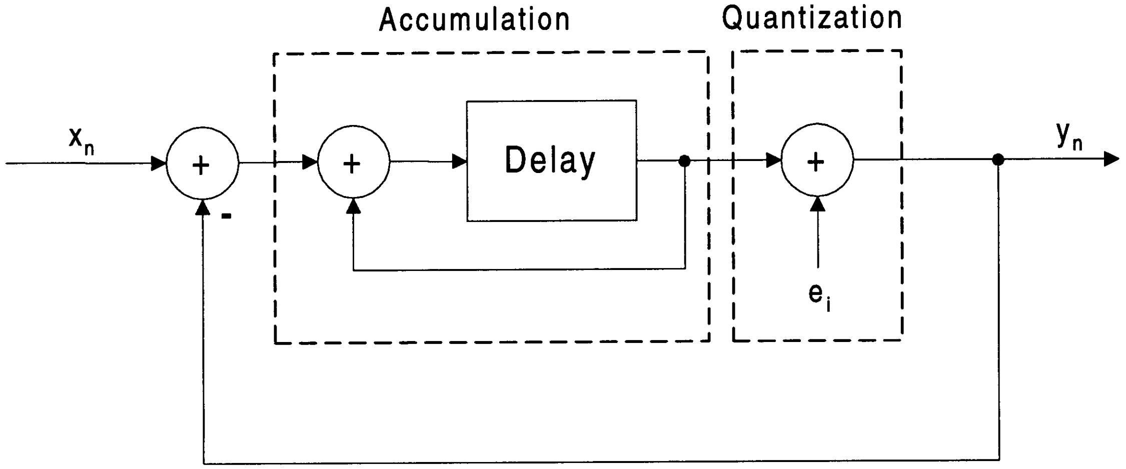
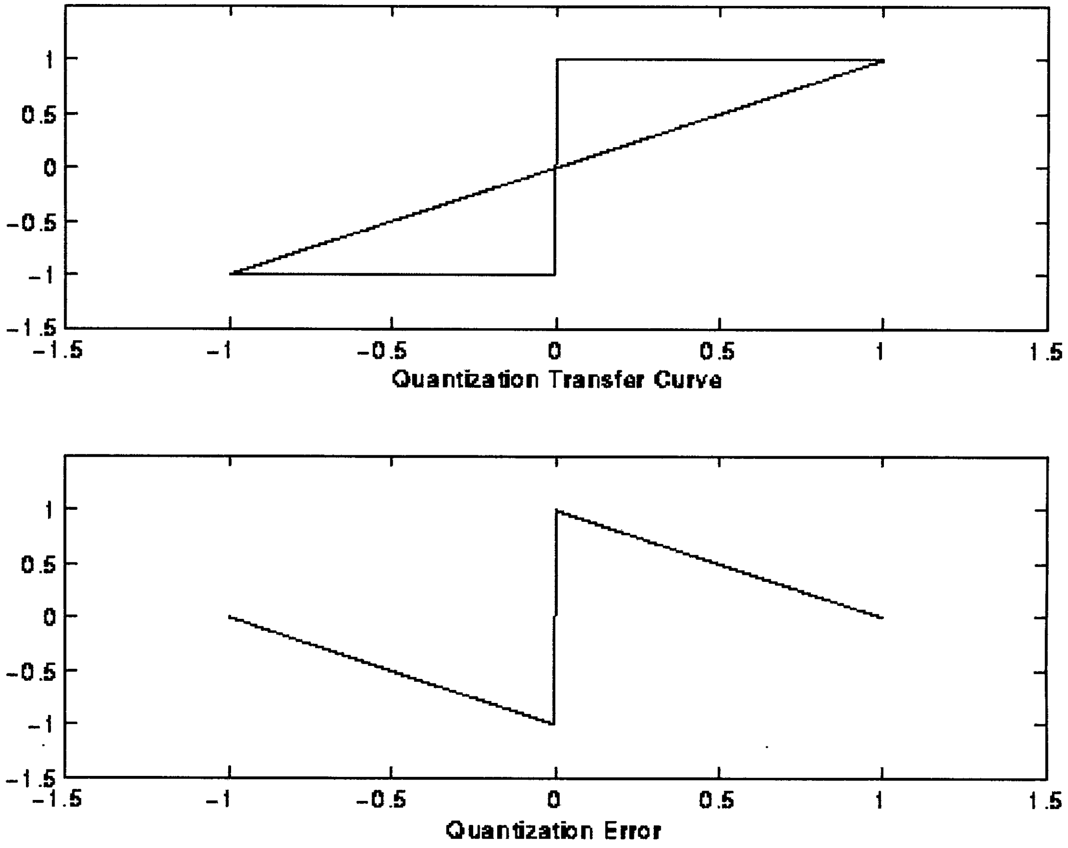


Figure 3-4: Single Loop ΔΣ Modulator (II)

The quantization seen in this circuit is modeled as addition of error, a fundamental characteristic of quantization that can be seen clearly in Figure 3-5, where the input-output characteristic and error characteristic are illustrated for a two-level quantizer. However, as the input is passed to the circuit via an integrator, any error is accumulated and will correct itself. Therefore, the average value of the output signal will track the input signal. This performance can be seen in Figure 3-6, here assuming a multi-level quantization characteristic for the sake of understanding this quantization. Note how the output oscillates between the two levels adjacent to the input such that the average of the output equals the average of the input. This clearly would not be the case in a standard quantizer.



Quantization Transfer Curve

Quantization Error

Figure 3-5: Quantization Characteristic

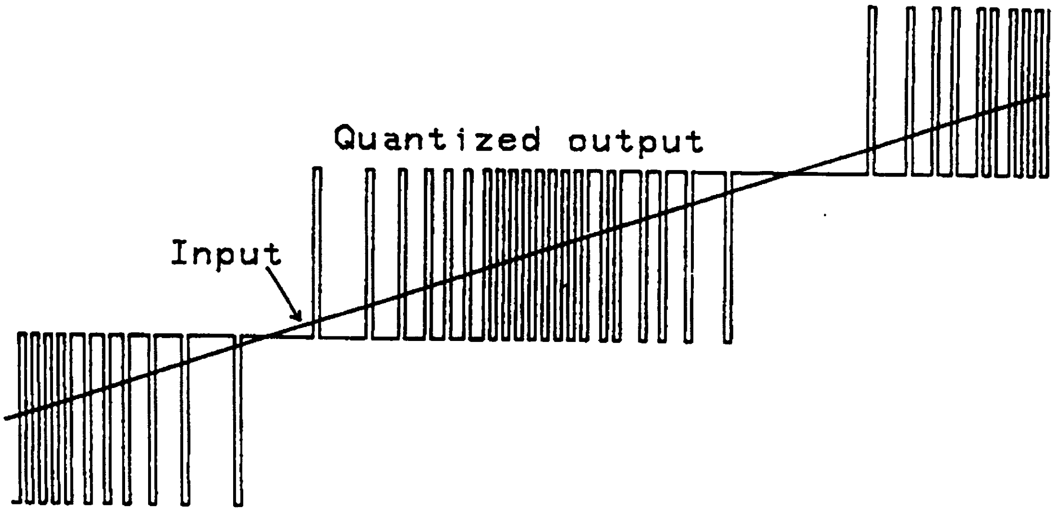


Figure 3-6: Delta Sigma Output to Ramp Input

The noise characteristic of this ΔΣ modulator can be experimentally determined as follows: By determination of the output of the accumulator as

$$w_i = x_{i-1} - e_{i-1}$$

it can be seen that the quantized output is

$$y_i = x_{i-1} + \left(e_i - e_{i-1}\right)$$

This illustrates the differentiation of the quantization error by this circuit while affecting the input only with a delay. The spectral density of the modulation noise

$$n_i = e_i - e_{i-1}$$

can be expressed, assuming uncorrelated error, as

$$N(f) \;=\; E(f)\left|1 - e^{-j\omega\tau}\right| \;=\; 2e_{rms}\sqrt{2\tau}\,\sin\left(\frac{\omega\tau}{2}\right)$$

which is seen in Figure 3-7 as compared to standard quantization noise. Thus, we have successfully shaped the noise spectrum to significantly reduce the noise at low frequencies, but increase it at higher frequencies. Our benefit comes, however, from the use of this system in occasions where our signal band is at that lower frequency segment of the spectrum. Indeed, in that signal band, the noise power is

$$n_o^2 = \int_0^{f_o} \left| N(f) \right|^2 df \approx e_{rms}^2 \frac{\pi^2}{3} (2 f_o \tau)^3, \quad f_s^2 >> f_o^2$$

with an rms value of

$$n_o = e_{rms} \frac{\pi}{\sqrt{3}} (2 f_o \tau)^{3/2}$$

which shows us that each doubling of the oversampling ration reduces the noise by 9 dB, providing 1.5 bits of extra resolution. This gain requires decimation of the modulated signal by a sharply selective digital filter for the sake of maintaining the resolution gain. [2]

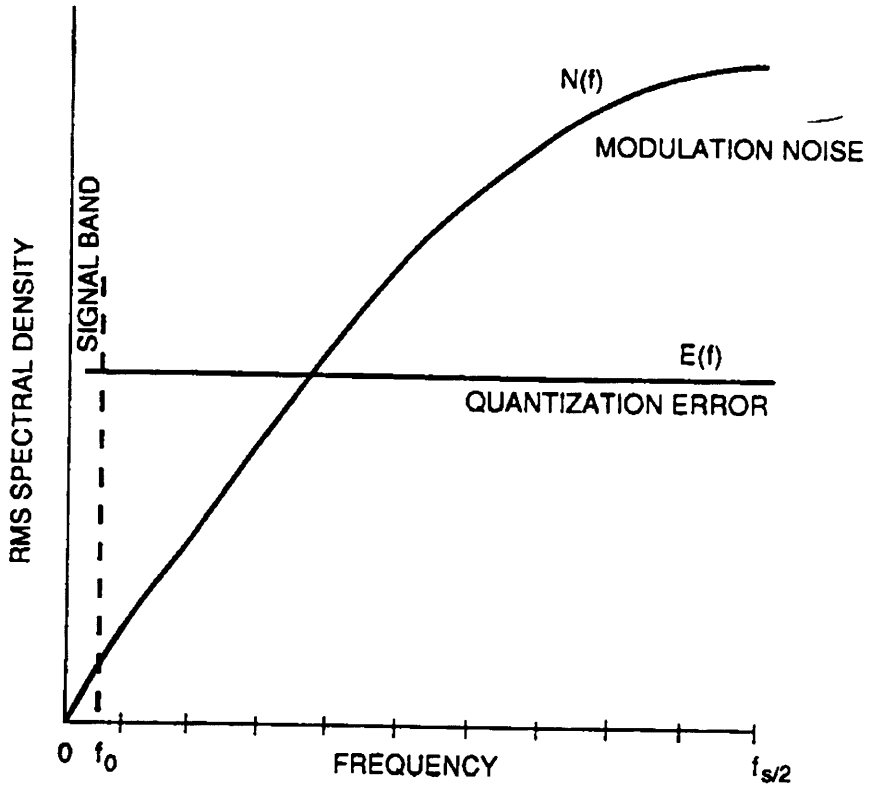


Figure 3-7: Spectral Density of ΔΣ Quantization Compared to Ordinary Quantization

3.4. Frequency Delta Sigma Modulators

Our interest in Delta Sigma is in its ability to perform frequency-to-digital conversion necessary in FM demodulation. This ability is the root of Frequency-ΔΣ-Modulators (FΔΣMs), which will be examined in general in this discussion. General theory and groundwork will be discussed with an eye towards the implementation of a triangularly weighted zero crossing counter implementation in Section 4.3.

The idea behind the FΔΣM is to quantize the angle θ_n by detecting an FM signal's zero crossing positions, therefore expecting a hard limited IF signal input. Detecting the number of zero crossings is done by clocking a counter during a time interval, and reading that number when desired. An error is introduced in this quantization, as our identification of the zero crossing location is imperfect, but this is where the ΔΣ benefit comes into play. This counter is then differentiated to recover frequency information, as seen in Figure 3-8. And, finally, a decimator follows the FΔΣM, compensating for the oversampling, as an inherent part of ΔΣ conversion.

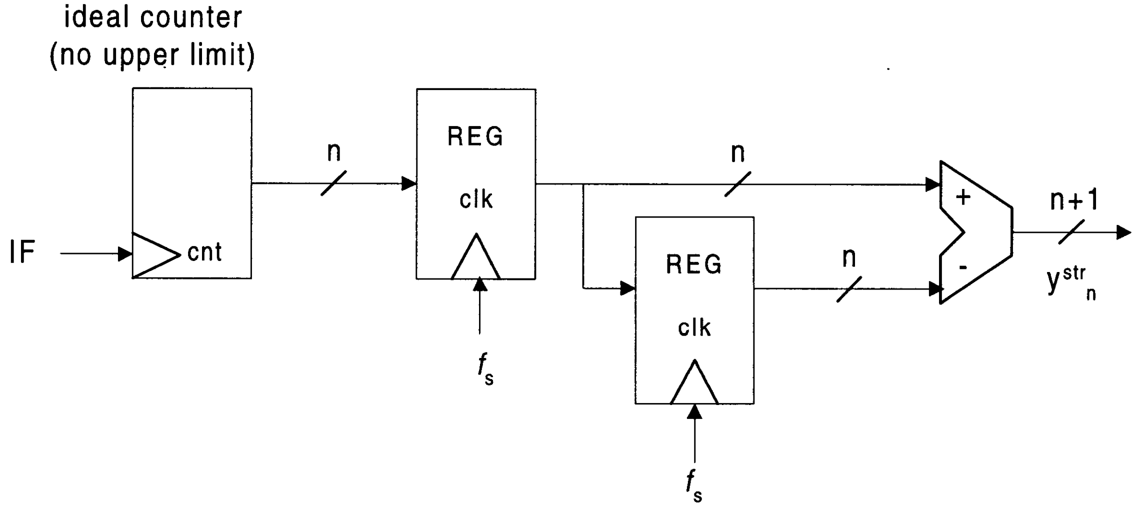


Figure 3-8: Basic FΔΣM Scheme

The output of the circuit in Figure 3-8 can be described by

$$y_n^{str} = \frac{\theta_n}{2\pi} - \frac{\theta_{n-1}}{2\pi} - \left(e_n - e_{n-1}\right)$$

which can in turn be seen, due to the high oversampling ratio as

$$y_n^{str} = \frac{f_c}{f_s} + \frac{k}{f_s} m(t) - \left(e_n - e_{n-1}\right)$$

This is seen to be a the sum of a scaling of *m(t)*, a DC bias proportional to the IF carrier frequency, and a DS noise-shaped quantization error.

From a demodulator point of view, the FM phase detector function is implemented as a an asynchronous counter clocked by the rising edge of the limited FM signal and the phase to frequency conversion performed by the differentiator. This frequency determination is not very precisely done, but accommodated for by the ΔΣ improvement, and the high oversampling compared to the IF frequency produces the error shaping that is one of the great benefits of this approach. Also, the fact that this is all done digitally, albeit at a high sampling frequency, is a tremendous gain in demodulation, as compared to the processing and power costs of analog circuitry, as will be seen in Chapter 4.

A practical implementation of this theoretical construct is seen in Figure 3-9, including an FΔΣM and a sinc^2 decimator. The major change from the theoretical model of Figure 3-8 and this model is the use of modular arithmetic to compensate for the concept of counting all zero crossings that had come before. A tremendously high upper limit would be required on the zero crossing counter in order to accommodate a system for any length of time without modulo arithmetic. However, by being consistent with modular arithmetic and ensuring that the modulus is sufficiently high, we can hereby eliminate that problem. The error introduced when the counter wraps around is also accommodated through the lack of carry in the modulo differentiator. Also worth noting in this implementation is the use of a register and adder for the implementation of the counter. The input signal is used as the clock for the 1st register, such that the input to that register is incremented by one with every positive zero crossing.

Following the system through, the output of the FΔΣM is an oversampled ΔΣ bit stream indicating whether a zero crossing has occurred during that clock cycle. That signal is passed through a sinc^2 comb decimator [8] which integrates the signal twice, performs the decimation to our output frequency, and then takes the derivative twice. At this point, the output is successfully demodulated and has been decimated to our desired frequency.

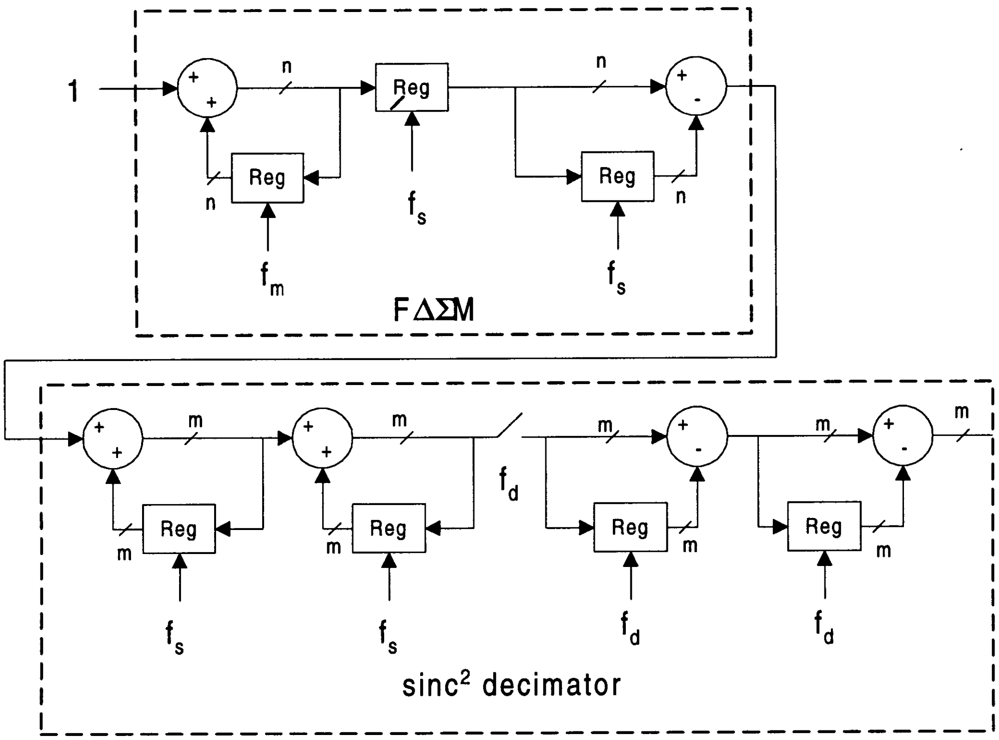


Figure 3-9: Modulo-2^n FΔΣM Followed by Modulo-2^n Decimator

Chapter 4. Digital Demodulation of Limited IF FM to Baseband

This chapter will make use of the background we have now gained in the realms of FM demodulation and $\Delta\Sigma$ modulators to explore the problem of digitally demodulating limited IF FM. These techniques are based upon the elimination of the AGC, as mentioned in section 2.3.2.2., by hard limiting the FM signal, a two step process involving amplification followed by clipping. This signal IF limited FM signal can then be sampled and demodulated entirely using digital hardware, a considerably cheaper method for a number of reasons. From a power standpoint, digital circuitry is considerably cheaper. Also, digital circuitry can be far more easily implemented and then altered.

We shall examine three methods of performing this FM demodulation. The first is based upon the I, Q channel separation technique of section 2. The performance advantages here are slight, but there is the great benefit of being the most open towards backwards compatibility and reuse of old designs. The second is a period counting sample and hold technique, with significant advantages, and some delta sigma effects aiding the processing. The third is the cheapest and most interesting, and is a variant upon the FDSM technique touched upon in section 3.4., optimized as a triangularly weighted solution. This last technique is simulated in depth in Chapter 5.

4.1. Conversion of limited IF FM to I, Q.

The clearest method at this point by which to demodulate the limited IF FM signal is to simply return to what we know and convert the limited IF FM signal to I and Q channels, with subsequent processing as described earlier. This approach can be seen in Figure 4-1.

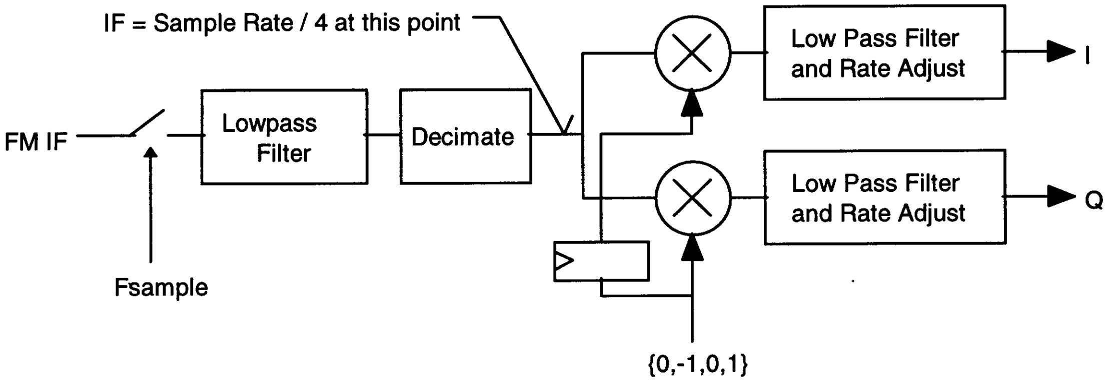

IF = Sample Rate / 4 at this point
FM IF
Lowpass Filter
Decimate
Fsample
Low Pass Filter and Rate Adjust
I
Low Pass Filter and Rate Adjust
Q
{0,-1,0,1}

Figure 4-1: Conversion of Limited IF FM to I, Q

There are several points worth observing in the block diagram. Notice the use of a sampling rate of 4 times the IF for the purpose of facilitating the mixing for I and Q separation, as discussed in section 2.3.2. Notice as well the rate adjust as the final step, in order to accommodate the desired output sampling rate. The decimation/interpolation low pass filter for that purpose can then be combined with the low pass filter required for post-mixing, and the two functionalities can be combined.

This system expects as input a low IF frequency sampled hard limited signal, and produces as outputs the same digital I and Q samples that we see in Figure 2-6. This is clearly a tremendous benefit as a means of adding in functionality without a major system overhaul. The system taking in I and Q channels and recovering phase and then frequency information from them, likely through an arctangent approach, can be maintained as is, and, should there be an inherent distinction at this point of the system, such as a barrier between two separate chips, this is clearly an advantageous approach. This can also be added in for a situation where capability for receiving either an analog IF signal or a hard limited IF signal was necessary, as all the I and Q interpretation circuitry would already be in place.

However, this is certainly not a tremendous gain from a theoretical point of view. We have effectively performed the process of hard limiting followed by lowpass filtering for the sake of emulating an AGC, which is a gain, but not as much of one as we can gain with the next two approaches. However, as this eliminates the AGC and maintains backwards compatibility, this is therefore the clearest first step in a hard limited system.

4.2. Period Counting Sample and Hold Technique.

Our attempts are to gain marked improvement in techniques through the use of a hard limited signal, and one system that provides this is a digital demodulation technique based on counting the periods between zero crossings and thus generating phase information. This attempts to provide for a proper balance between analog and digital techniques, in order to expend little analog current while keeping digital sampling rate down. This is because, given an IF considerably lower than our sampling rate, there is a long and continuous string of positive or negative voltages between zero-crossings, and thus considerable information that is not necessary. Sampling this signal at the high sampling rate would require digital processing far above the necessary rate, while pure analog demodulation would require more current, as described above. This circuit also winds up functioning as a first order frequency delta sigma quantizer circuit, pushing quantization noise out of our lower frequency band of interest.

The technique functions as follows, and can be seen in Figure 4-2. An IF FM signal is hard limited and quantized to one bit, giving us a signal that appears to be a square wave, but with the information contained in the location of the zero crossings. The one-bit stream produced is then exclusive-NOR'd with itself delayed by one sample period, such that a zero is produced at this output only when a zero crossing has been detected. This is our simple zero crossing detector. This signal is passed to the input and the negatively enabled reset of a counter and to the negatively enabled load of a register. Thus, the counter increments at the sampling rate between zero crossings, and then dumps its output to the register and resets at each zero crossing. This is our sample and hold circuit. However, it differs from conventional sample and hold circuits in that the length of time during which each successive output is held is actually non-uniform. This is to say, each output word, which is the number of clock cycles between each zero crossing, is held for an amount of time proportional to the next input word. However, in choosing a sampling clock frequency quite high, such as 19.68 MHz, a common voltage controlled crystal oscillator frequency, we can ensure that the variance between durations of each of those input words is small compared to the large sampling frequency. Therefore, that factor can be dealt with in the next stage. The output of the system described so far is the time period between zero crossings, the inverse of which is the frequency. We have therefore successfully demodulated the signal.

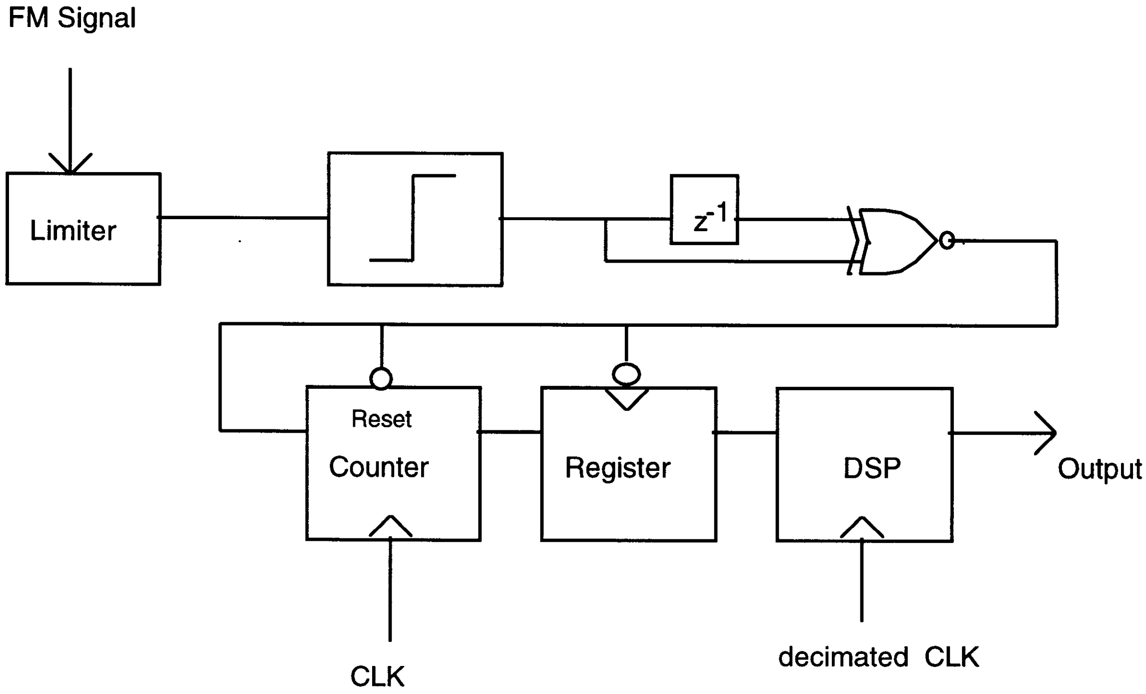


Figure 4-2: Sample and Hold Technique

This signal is then passed to the digital section of the system, which operates at a considerably slower sample rate, effectively decimating the signal. With properly chosen sample rates, this also solves the problem of coupling the non-uniform samples and the uniform digital clock rate. The delta sigma modulation is introduced because the error generated through our quantization is accumulated with each successive sampling, with subsequent rollover, and shifting of noise to higher frequency areas where it does not impact our system. Finally, the signal generated is proportional to the period of the desired audio, rather than the frequency, and this must be corrected by inverting the signal, either directly, or by imitating an inversion curve. For example, as inversion is a relatively complex mathematical function to implement, the curve can be linearized around a point on the curve, as seen in Figure 4-3.

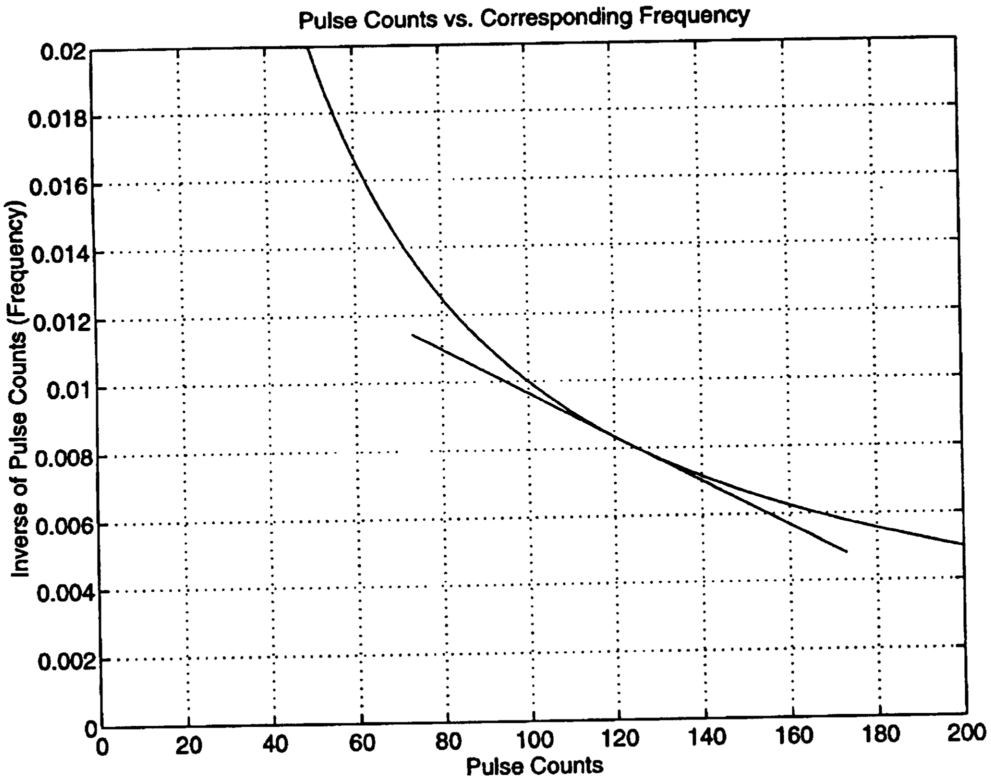


Figure 4-3: Linearization of Period to Frequency Characteristic

Implementation of this system involves careful awareness and manipulation of a number of factors. The frequencies used, from the IF frequency to the initial sampling frequency to the digital sampling frequency and its decimation factor must be observed carefully and their interactions examined. The performance on a variety of modulating signals is also to be carefully considered, with respect to the required ranges in the various standards to which I will be applying this method. Also, as this measures period rather than frequency, that conversion is also to be considered. Also, as always, quantization introduces error, and that error must itself be examined and considered, and the effect of the sigma delta converter set up for maximum benefit.

4.3. Triangularly Weighted ZC Counter

The other main demodulation approach examined in this discussion is a variant upon the FΔΣM approach of section 3.4., known as triangularly weighted zero crossing counting. This takes advantage of some redundancy in the FΔΣM design to pare that design down and thereby save further on processing blocks. This scheme will be examined and the effects of it will be considered, followed by C code simulation in Chapter 5.

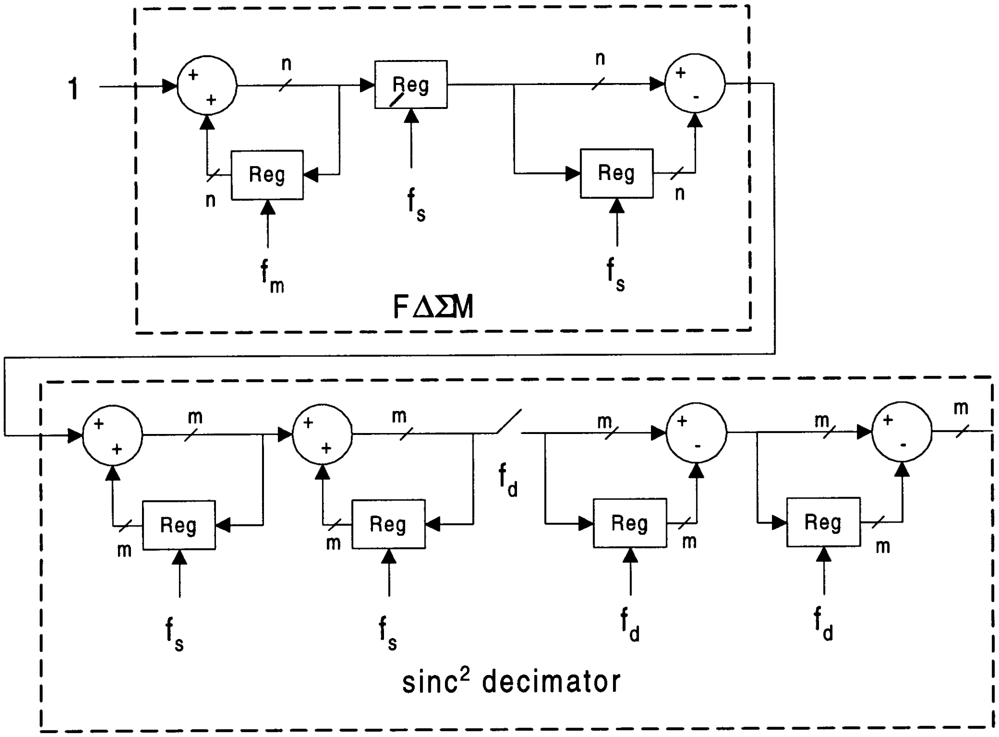


Figure 4-4: Modulo-2^n FΔΣM Followed by Modulo-2^n Decimator

As a starting point in our discussion, let's review the block diagram from section 3.4, shown again here as Figure 4-4. The two major functional blocks are the FΔΣM and the sinc^2 decimator. The minimum word length *m* for the decimator will generally be considerably larger than the minimum word length *n* for the FΔΣM. However, by making n=m, we wind up with the FΔΣM's differentiator as the logical inverse of the first accumulator in the decimator. As both of these blocks were running at the oversampling rate of fs, the total power consumption will be reduced significantly. This new block diagram is displayed in Figure 4-5.

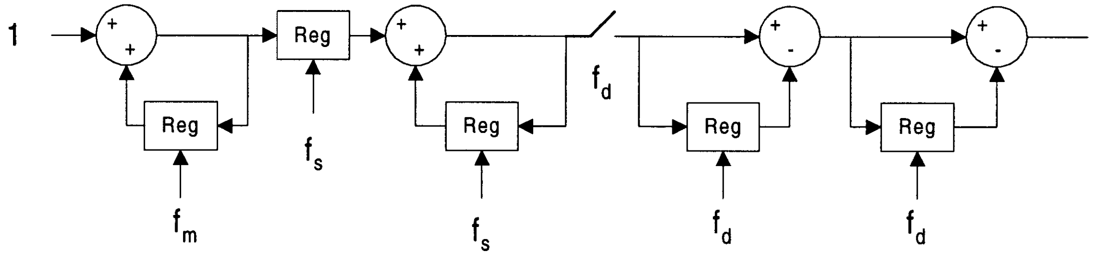


Figure 4-5: Triangularly Weighted ZC Counter Technique

This new circuit is a sinc^2 decimator with the first accumulator clocked not by the sample clock but by the rising edge of the asynchronous FM signal. The behaviour of this counter is similar to the standard FΔΣM, however, the decimated signal has not been differentiated, and thus increments with time. This is then made up for by the additional differentiator at the end. The noise in the circuit is therefore ΔΣ modulated, through the summation of error, while the signal at the output is the frequency of the original input. The SQNR is given by

$$SQNR(dB) = 20\log\left(\frac{2\,\Delta f}{f_s}\right) - 20\log\left(\frac{\sqrt{2}\pi}{3}\left(\frac{f_N}{f_s}\right)^{\frac{3}{2}}\right)$$

with f_N the nyquist frequency of the modulating signal [6]. This illustrates the delta sigma modulation gain, as the first term in this equation is the ordinary positive ZC counting resolution, and the second represents the increased resolution due to ΔΣ noise shaping[1].

The origin of the triangular weighting terminology is the removal of the differentiator, and can be seen in Figure 4-6. The barrier between the FΔΣM and the decimator has been blurred, and it can be considered that the output of the FΔΣM, the output of the counter, is a modulus increasing function, and the result is a stagger-step triangular effect. This is as compared to the differentiated and adjusted signal from a uniformly weighted signal, where the differentiator creates a contained initial system where the output is a fully detected signal.

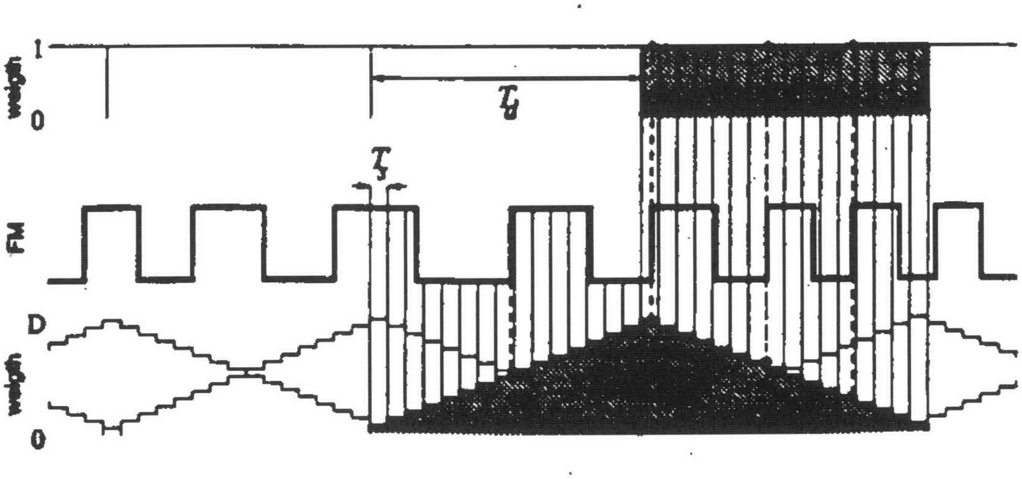


Figure 4-6: Uniformly Weighted Counting (top) vs. Triangularly Weighted Counting (bottom)

The effectiveness of this system will be examined through simulations in Chapter 5. The benefit of the system, however, can be seen through the components involved. A total of 3 registers and 2 adders at the high oversampling frequency, and 2 registers and 2 adders at the lower sampling frequency, are all that is required. These are all fairly cheap to implement, and are all digital logic, making the system as a whole considerably cheaper to implement than any of the systems examined in section 2.3.

Chapter 5. Triangular Weighting Technique Simulation

The simulation based development of this system will take place in two major segments: the development and analysis of the demodulation scheme, as seen in this chapter, and its application to a multi-standard environment in Chapter 7.

The main tools for this work were the C programming language and the mathematical tool Matlab. The C code is attached in Appendix A, along with some Matlab M-files, as relevant. The compiler used was the GNU g++ compiler, on a Sun SPARC-5 platform, and Matlab version 4.2, on the same platform. The work in Matlab was primarily dealing with the general mathematical capability of the tool as well as working with the signal processing toolbox as an aid in the generation of filter coefficients and analytical examination of results.

5.1. Program Development

We'll first examine the development of the demodulation simulation software, the choices which were made, the issues faced, and the means by which the results in section 5.2. were produced.

5.1.1. System Issues

The basis for this system was discussed in section 4.3., however, a few words are in order for the choices made when implementing the system in these simulations. The general idea in this simulation was to observe the behaviour and results obtained as the various aspects of the system are tweaked. The foremost goal was the development of a firm understanding of the capabilities of this demodulation technique parameters such as signal frequency, signal deviation, oversampling frequency, and IF carrier frequency are changed. As such, the system was written in a way such that changes to those parameters would be easily made.

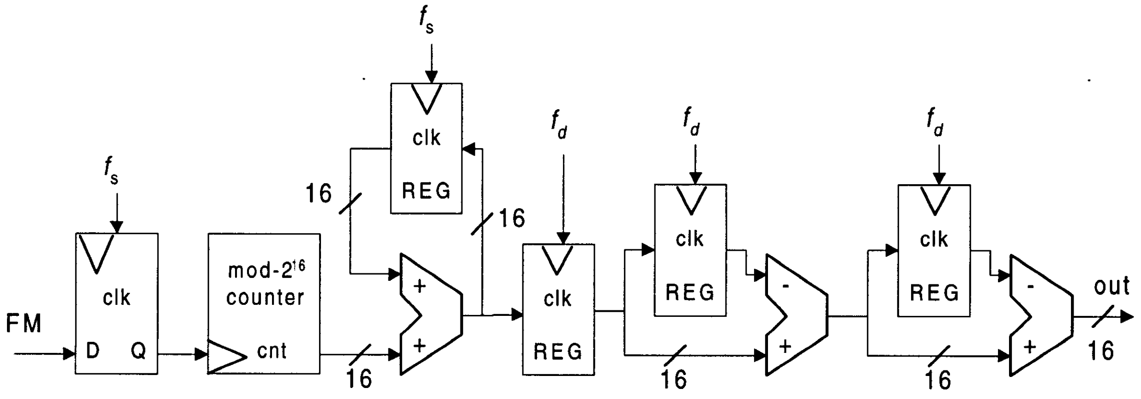


Figure 5-1: Triangularly Weighted FDC Implemented as Sinc^2 Decimator

The simulation of Figure 4-5 is redrawn here as Figure 5-1 in the manner in which it was implemented. Several changes have been made since Figure 4-5. The first is that the synchronization has been pushed back to before the initial counter, in order that that element be removed from concern. Indeed, in a simulation such as this, the synchronizing register is not at all necessary, as synchronization is guaranteed. However, our input sampling frequency is set as the oversampling frequency, which is functionally the purpose of this register. Also, the counter has been returned to a true modular counter, although this is simply an implementation difference. In code, the effect is the same.

A word-length has also been chosen, set to 16. This was chosen as sufficiently wide for our purposes, while not requiring excessive size. The primary concern in word lies at the output, where the operating range must be guaranteed to be within that word length, such that our final output does not illustrate signs of wrapping around, and this was found to be sufficient to guarantee that that be the case.

Also, some baseline frequencies were considered for the sake of initial testing. These initial parameters were chosen as likely candidates for our demodulation system, using frequencies readily available, and parameters correlating to the standards work which will be discussed in the following chapters. A sampling frequency of $f_s = 19.68$ MHz, for example, was chosen due to its use in similar applications, such as in our discussion of the period counting technique, and a carrier frequency of $f_c = 492$ kHz is chosen as an IF due to the relative ease of finding analog filters and elements for operation at that frequency, as a general analog filter is necessary for adjacent and alternate channel rejection. The other basic parameters are based upon the AMPS standard and are as follows: a deviation $\Delta f = 2.9$ kHz, and a modulating signal bandlimited to $f_m = 3$ kHz, leading to a nyquist frequency of $f_N = 6$ kHz.

Finally, the input to system was produced by imitating an FM IF hardlimited signal based upon one or more tone frequencies. This was produced by altering the modulation formulae slightly according to our knowledge that all we are interested in is the limited version, and therefore only interested in the sign of the signal. Our FM formulas:

$$y(t) = A\cos\theta(t)$$

$$\frac{d\theta(t)}{dt} = \omega_c + k_f x(t)$$

are therefore altered by the facts that *x(t)* is a pure tone, or sine wave, and the fact that the cosine function in the first formula is positive for $0 < \theta(t) < 180^\circ$, and negative for $180^\circ < \theta(t) < 360^\circ$. We can therefore simply increment *θ(t)* by 360° times the right side of the second formula, with each passing of 180° corresponding to a reversal of our limited output.

5.1.2. Programming Issues

The implementation of the software designed to perform this functionality was done with repetition of a great many runs with varied parameters foremost in mind. This led to several choices made which can be seen reflected in the code itself.

The first of these is the speed of this system. The need for a large number of points to form the basis for a large enough Fourier transform results in passing on the order of 100,000 points at our oversampling frequency of 19.68 MHz through this system, which takes a considerable amount of time to process. To improve the performance to some degree, the file I/O is performed at the most basic level, writing output in binary form to minimize the file sizes and the time of writing. These can then be read into Matlab and analyzed there.

The second major influence is an attempt to make as many of the parameters of the system inputs as possible. By depending upon an independently created file for inputting all the major parameters of the system, a large sequence of tests can be performed back to back, with a Matlab script setting up each test, running through it, and maintaining the appropriate data.

5.2. Testing and Analysis

5.2.1. Means of Analysis

The fundamental measure of success in a demodulation scheme of this sort is the signal to noise ratio, as seen by observing the power spectral density of the demodulated signal. Our sample runs for the sake of testing the demodulator all involve tones being passed through the system, which allows for fairly straightforward determination of signal to noise.

A sample run, using the base numbers referred to in section 5.1.1., is seen in Figure 5-2. Here, we can very easily see the noise shaping characteristic of the delta sigma modulation, as the noise floor is at its lowest at low frequencies, and builds up to a higher level with increasing frequency. The tone, in this case at 1 kHz, is seen far above that noise floor. The noise measurement was made by integrating the power spectral density of the signal over the audio band of 300 Hz to 3000 Hz, with the exception of those points representing the tone itself, which made up the signal measurement. The SQNR was therefore the log of the ratio of those two signals, and is produced in the file testhovin8.m in Appendix A.2. The SQNR can be seen in this example to be 55.6.

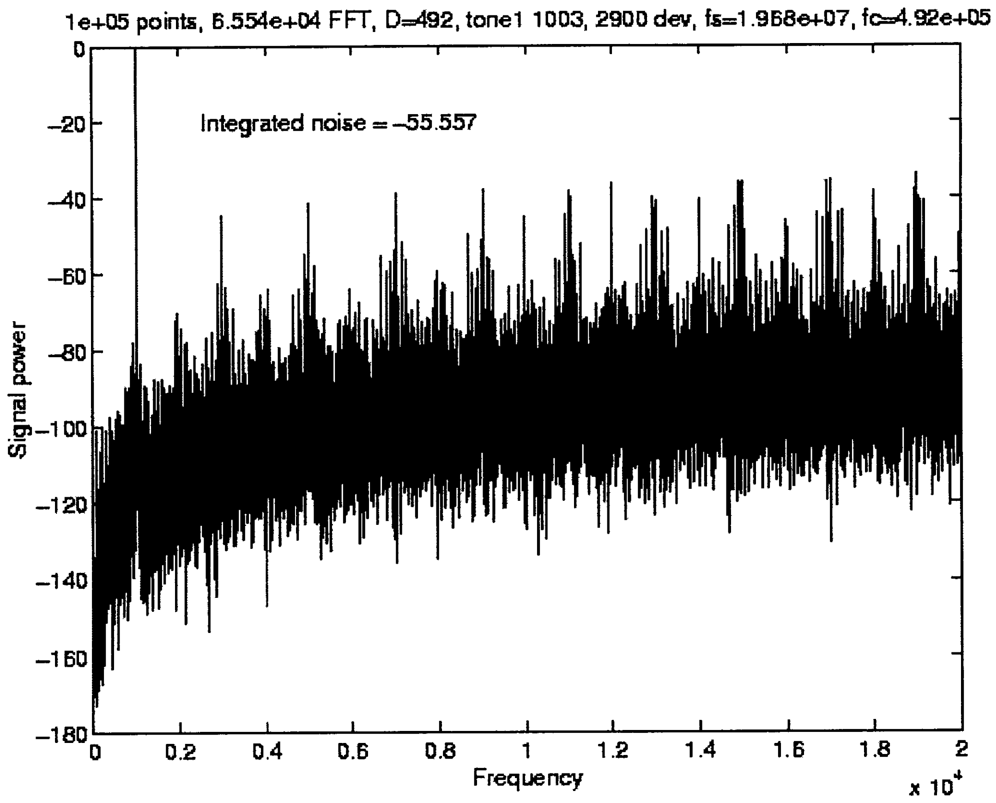


Figure 5-2: Default Specification Demodulation Results

5.2.2. Tone Frequency

The first set of tests were done based upon sweeping the single tone produced across the audio segment of the frequency spectrum, in order to determine how performance varied with tone frequency. This was done with a series of simulations, spaced across the 300 Hz to 3000 Hz range with which we are concerned. The results from this can be seen in Figure 5-3.

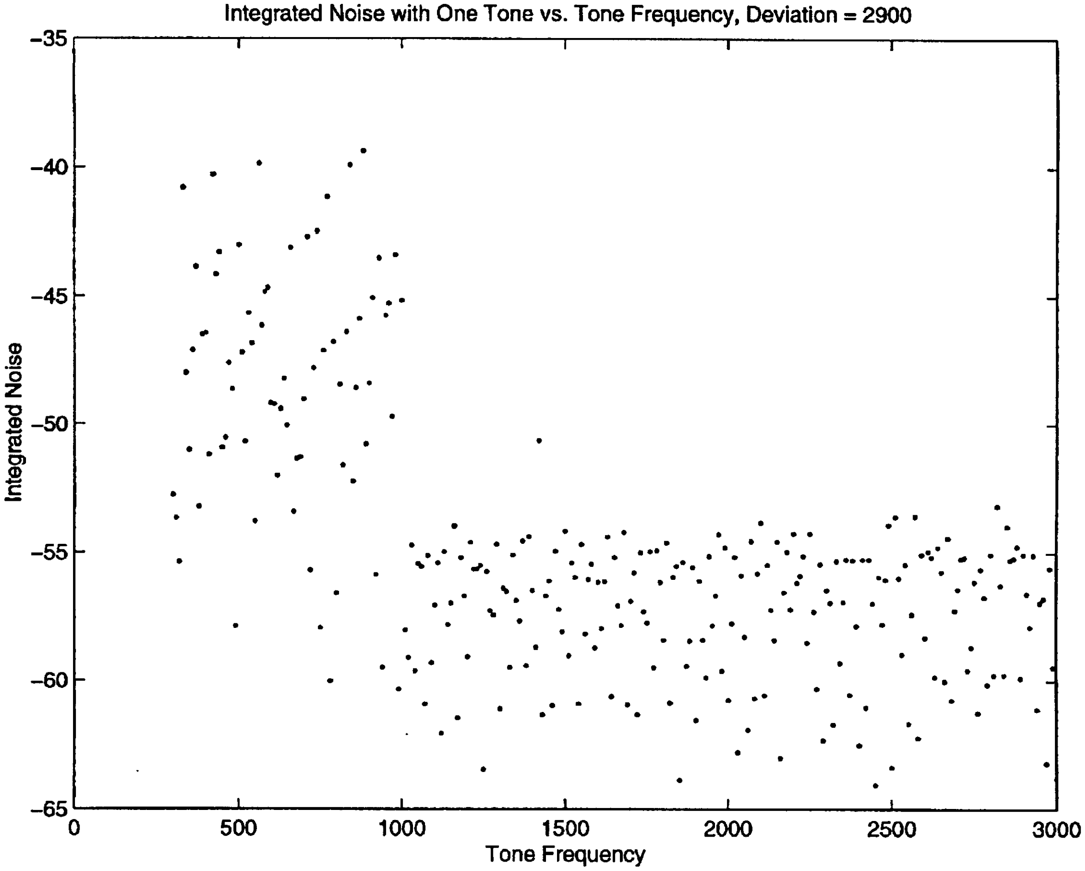


Figure 5-3: Integrated Noise vs. Single Tone Frequency

This plot illustrates an interesting effect. The largest single component to the integrated noise is the presence of reflections of those tones, which therefore manifest themselves at 3, 5, 7, 9, and every further odd multiple of the tone frequency. These reflections are considerably attenuated from the tones themselves, but are far higher than the noise floor. This effect can be clearly seen in Figure 5-3, and best understood working backwards. Above 1 kHz tone frequency, the integrated noise is at around what we might expect to be the noise floor, and varies only a little from frequency to frequency. However, below 1 kHz we see a pattern emerging. This is, in fact, a fabrication of our measurement technique, as the first spur for a 1 kHz tone is at 3 kHz, and is therefore just entering our realm of measurement, from 300 Hz to 3 kHz. Therefore, there is a significant jump at 1 kHz, and slow reduction from there until 600 Hz, when the 5th reflection drops below the 3 kHz barrier, causing the noise to jump once again. This repeats itself as we drop and further spurs enter the measurement realm.

However, despite this, at all frequencies under consideration, there is sufficient SQNR to suit our purposes, as the maximum SQNR can be seen to be 38 dB, existing just before 1 kHz. As such, our base demodulation scheme is seen to be effective at all our relevant range of frequencies.

5.2.3. Tone Deviation

The next parameter examined was the deviation applied to a single tone. The deviation on a 993 Hz tone was varied from 3 Hz to 15 kHz, with the resultant noise of integration plotted in Figure 5-4.

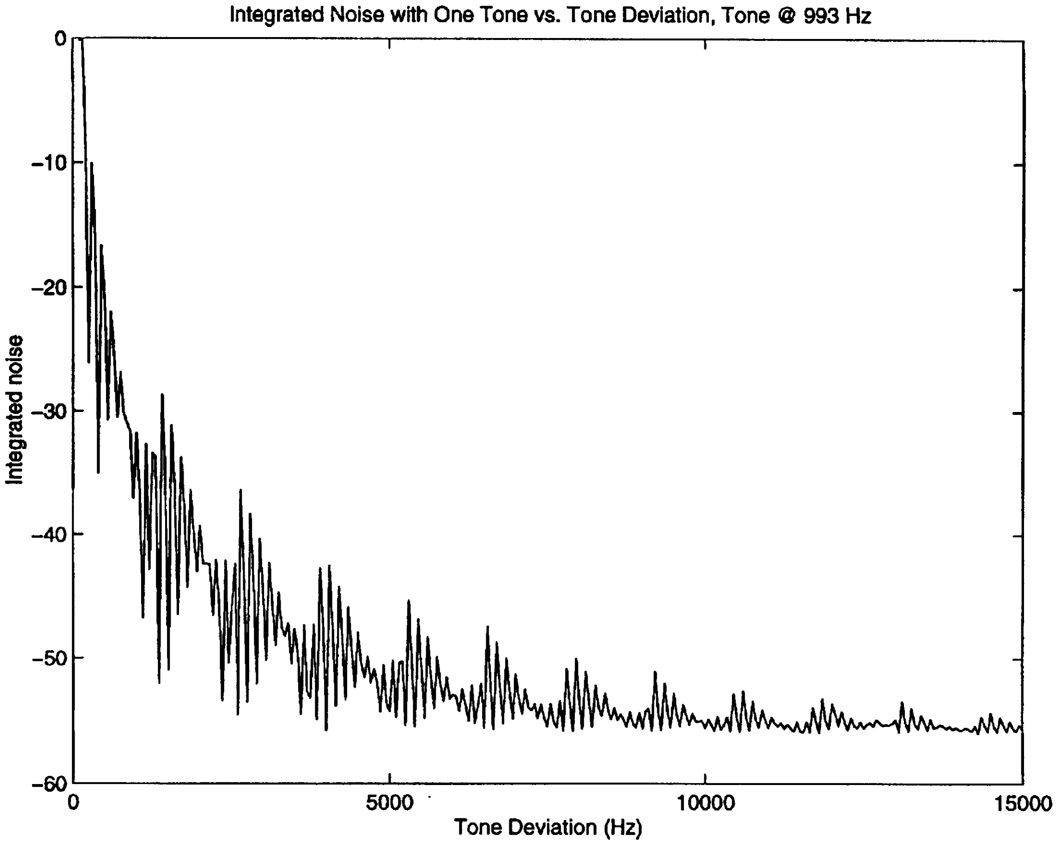


Figure 5-4: Integrated Noise vs. Single Tone Deviation

Here, we see a definite correlation between deviation and performance, as expected. The higher the deviation applied to the tone, the simpler it is to detect and correctly demodulate it. We also see a second order effect present, although this is perhaps a function of the measurement technique, and is not in any means sufficient to alter the fact that the integrated noise when high enough remains well within limits. With those limits firmly set, this can be examined for performance at a relevant deviation, however, with the lowest deviation applied to our signal in these standards at 700 Hz deviation in a JTACS/NTACS SAT tone, we still have an integrated noise of 30 dB, and are thus still quite well off.

5.2.4. Oversampling Frequency

The choice of oversampling frequency was considered for completeness, although our system has no problems with the use of the 19.68 MHz clock. However, the generation of other high-speed oversampling frequencies requires that they be a factor of the 19.68 MHz, and as such, our examination is fairly brief. The results can be seen in Figure 5-5.

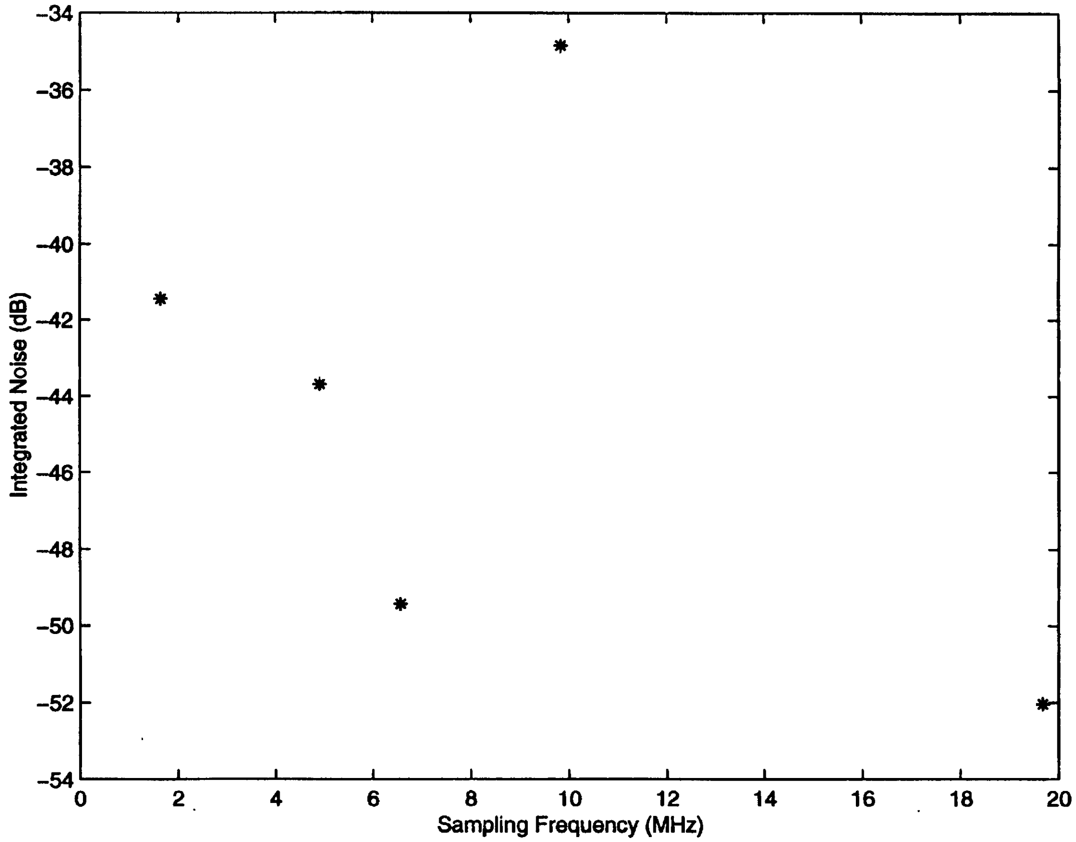


Figure 5-5: Integrated Noise vs. Oversampling Frequency

It can be seen that the results are quite varied, however, we retain the best performance at our full oversampling, and as a result it appears to be an effective choice.

Chapter 6. Cellular FM Standards

The characterization and specification of cellular telephony standards is a key element and necessary task for the agencies responsible for the regulation of frequency spectrum. Spectrum is very limited, and the rights to it are highly competitive, such that when technologies are devised to take advantage of that spectrum, the qualities inherent to those phone calls must be carefully specified so as to enable the design of reliable, functional mobile units. The existence of such standards eliminates misunderstandings between manufacturers and purchasers, facilitates interchangeability and improvement of products and assists purchasers in selecting the appropriate product for their needs. In the United States, this spectrum is specified and licenses are granted by the Federal Communications Commission (FCC), and the standards are governed by the Telecommunications Industry Association (TIA). In Japan, these roles are filled by the Nippon Telegraph and Telephone Corporation (NTT).

This discussion will cover the examination, specification, and simulation of three of these standards, along with a brief discussion of one other. The American AMPS system will be looked into in great detail, and used as a basis for the work. The Japanese JTACS and NTACS systems are the primary new goal, and will be examined, their similarities and differences to the AMPS system characterized, and the applicability of our delta sigma demodulation technique will be examined through computer simulation. Also, the American NAMPS standard will be looked at briefly, as a means of furthering our understanding of the NTACS standard.

The issues we will concern ourselves with here are the processes involved in modulation and demodulation of these signals. Our concern here is the applicability of our delta sigma demodulation technique to these standards, and as such, our goals are simply to recover the signals we send through these channels, and not the specifics of the data streams or the processing involved in handling the phone calls themselves. For this reason, the aspects of the telephony standards that apply to the handling of demodulated signals, especially data and supervisory signals, is not considered here, and is assumed to be dealt with entirely in software at a later stage, upon being provided with the correct data streams.

However, the parameters about the transmitted signals themselves will be looked at in great detail. These include such issues as channel width and location, and distance from adjacent and alternate channels, along with the general sense of the shape of the signals. They also include the interactions between the various forms of data transmitted and the encoding specific to modulation that is applied to these signals, including the audio voice signal, supervisory tones, and data signals.

Each standard will be approached in turn, with every relevant step described in some depth. As we enter the discussions of JTACS and NTACS, however, it will be primarily the similarities and differences that will be focused upon, so as to maintain our mindset of expanding an AMPS system to include those other standards.

6.1. AMPS

The first standards we will observe are the Advanced Mobile Phone Service (AMPS) standards commonly used in North American FM transmission. These include the original AMPS standard, as specified by TIA standards such as IS-95, IS-97, and IS-98, as well as the Narrowband AMPS (NAMPS) standard.

6.1.1. AMPS

The first standard which we will explore, and use as a basis and starting point for this system development is the Advanced Mobile Phone Service, or AMPS, standard. Originally designed as an analog radio-telephone standard by AT&T, it has developed into the standard of choice for US analog cellular mobile communications [17]. With the advent of newer digital standards for cellular communications, the most reliable and current source for AMPS standard information can be found in those documents which describe the functionality of dual-mode wideband spread spectrum cellular systems, such as the Telecommunications Industry Association (TIA) Standards IS-95, IS-97 and IS-98.

6.1.1.1. Channels

The AMPS standard uses channels allocated in the 800 MHz band, a very commonly used cellular telephony frequency band. Each of the base and the mobile dedicated channel spaces are of bandwidth 25 MHz and the spacing between a mobile channel and its base counterpart is 45 MHz. The spacing between adjacent channels is 30 kHz. There is availability for up to 833 channels for transmission by each the base and the mobile unit, with 666 of these channels in basic coverage, but with mandatory capability for the additional 167 channels. These channels are numbered from 1 to 799 and then from 990 to 1023, and their center frequencies specified according to Table 6-1.

Transmitter	Channel Number	Center Frequency (MHz)
Mobile	$1 \leq N \leq 799$ $990 \leq N \leq 1023$	0.030 N + 825.000 0.030 (N - 1023) + 825.000
Base	$1 \leq N \leq 799$ $990 \leq N \leq 1023$	0.030 N + 870.000 0.030 (N - 1023) + 870.000

Table 6-1: AMPS Channels

Desensitization from nearby channels is also specified in IS-98, with the adjacent channel being defined as that channel immediately next to, or 30 kHz above or below, the current channel, and the alternate channel defined as the channel one more spot removed, or 60 kHz away. The minimum adjacent channel selectivity is specified to be 16 dB, and the minimum alternate channel selectivity is specified to be 60 dB.

The purpose of these channels can be further specified as Forward Analog Control Channels (FOCC), Forward Analog Voice Channels (FVC), Reverse Analog Control Channels (RECC) or Reverse Analog Voice Channels (RVC). The control channels are used to communicate purely signaling information and are used primarily for overhead issues such as paging a mobile unit, or access channels for the mobile unit to access a base system. The voice channels, meanwhile, are the traffic channels used by the analog system for the sake of transmitting both voice communication and signaling data.

6.1.1.2. Transmit Voice Processing

The first major portion of the transmitted signal, and the one most obvious to the end user is the voice signal that makes up a significant portion of the channel. All user terminals require a certain degree of processing on that voice signal in order to compensate for various unique qualities of those user terminals, whether those be physical acoustic qualities, the influence of the microphone, or the early voice processing. However, there are a number of modulation related voice-processing steps which are specified as necessary qualities for a user terminal or base station's vocoder, as can be seen in figure. These stages are described in detail in the following sections. Note that the voice signals throughout are bandlimited to 300Hz to 3kHz.

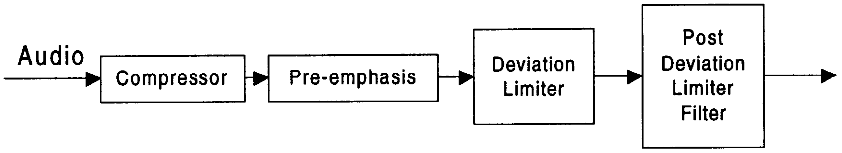


Figure 6-1: Transmit Voice Processing Block Diagram

6.1.1.2.1. Transmit Level Adjustment

The first such specification, and one more general in nature, concerns the fixing of a reference level for the remaining processing. This is set such that a 1004 Hz tone at nominal acoustic speech level will produce a ±2.9 kHz frequency deviation of the transmitted carrier. This 1004 Hz tone will be the generally used reference throughout. The nominal acoustic speech level is defined according to whether measurements are being taken at the base station or the mobile unit. The mobile unit requires a transmit objective loudness rating (TOLR) equal to -46 dB when transmitting to a reference base station, and a nominal receive objective loudness rating (ROLR) equal to 51 dB when receiving from a reference base station. A base station, meanwhile, is required to set that 1004 Hz tone with ± 2.9 kHz deviation to a level of -18 dBm0 at the network interface. The mobile unit's responsibility for achieving this lies in dealing with the physical characteristics of the unit itself.

6.1.1.2.2. Compressor

Speech signals are not uniformly distributed, but rather, have a higher probability for smaller amplitudes and a smaller probability for larger amplitudes. Therefore, it makes sense to design a quantizer with more quantization regions at lower amplitudes and less at higher amplitudes. This is accomplished through the use of a syllabic compandor. The idea behind a compandor is to compress the large amplitudes prior to the system, and then expand those amplitudes on the other end of the system in order to recover the original signal. [14]

The next stage specified is therefore is the compressor portion of a 2:1 syllabic compandor. This serves to compress the signal such that for every 2 dB change in input level, the change in output level is a nominal 1 dB. There is a specified nominal attack time of 3 ms and a nominal recovery time of 13.5 ms, which are the CCITT-recommended nominal values. These are chosen so as to maintain a response time slow enough to not respond to level changes significantly faster than the 20 ms R-C time constant, but to restrict slower syllabic variations. This compressor is to be designed according to the 1000 Hz acoustic tone and nominal 2.9 kHz peak frequency deviation mentioned earlier. [20],[21]

6.1.1.2.3. Pre-Emphasis

A pre-emphasis curve is next applied to the signal. The rationale behind this lies in an examination of the noise power spectrum of FM, and noting the parabolic shape. FM modulation performs better for low-frequency components than for high frequency components, and it is therefore in our interests to design a system that takes advantage of that. We therefore focus our transmission efforts on the more crucial high frequency section of the audio band by applying a linear frequency response filter to the signal, emphasizing the high frequency spectrum. This characteristic must have a nominal +6 dB/octave response between 300 and 3000 Hz. It gains us an improvement factor ρ_{FM} of 15.23 dB, as seen through these calculations [22] :

$$\rho_{FM} = \frac{(f_2 / f_1)^3}{3[f_2 / f_1 - \tan^{-1}(f_2 / f_1)]}$$

with $f_2 >> f_1$ and $f_2 = 3$ Hz and $f_1 = 300$ Hz such that:

$$\rho_{FM} = 10 \log \frac{100}{3} = 15.23 \text{ dB}$$

6.1.1.2.4. Deviation Limiter

The deviation limiter exists for the purpose of limiting the audio portion of the instantaneous frequency deviation to ±12 kHz. This can simply be done by limiting the amplitude of the audio signal.

6.1.1.2.5. Post Deviation-Limiter Filter

The deviation limiter must be followed by a low-pass filter to account for any additional high frequency elements that may have been introduced by that limiting. That filter is carefully specified in Table 6-2:

Frequency Band	Attenuation Relative to 1000 Hz
3000 - 5900 Hz	$\geq$ 40 log (f /3000) dB
5900 - 6100 Hz	$\geq$ 35 dB
6100 - 15000 Hz	$\geq$ 40 log (f /3000) dB
above 15000 Hz	$\geq$ 28 dB

Table 6-2: Post Deviation-Limiter Specification

6.1.1.3. Wideband Data Signals

The second major aspect of the transmitted signals are the wideband data streams . Wideband data streams are sent in both the voice channels and in control channels for the purpose of call processing. These streams are 10 kbps binary streams and can convey a wide variety of messages, including page response messages, origination messages, order confirmation messages, order messages, called-address messages, and other such necessary communication. There are extensive guidelines pertaining to their creation, although, as mentioned before, those will not be examined here, and we will concern ourselves only with ensuring that the data stream specified for transmission be properly encoded, transmitted, and recovered at the other end.

6.1.1.3.1. Encoding

The wideband data stream must be encoded with a Manchester (biphase code), such that each nonreturn-to-zero binary one be transformed into a zero-to-one transition, and each nonreturn-to-zero binary zero be transformed into a one-to-zero transition. This encoding is illustrated in Figure 6-2. This encoding also converts the 10 kbps signal into a

20 kbps signal. It will also be desirable, although not specifically required by the specification, to smooth out such a curve, so as to cut down on its bandwidth and remain within frequency emission specifications. This can be accomplished either through a low pass filtering technique, or a curve matching technique.

The purpose of this encoding is to shift the concentration of energy away from the audio band and up closer to the transmission rate of 10 kHz. This allows a burst of signals transmitted over the same channel as a voice signal to be properly detected and decoded.

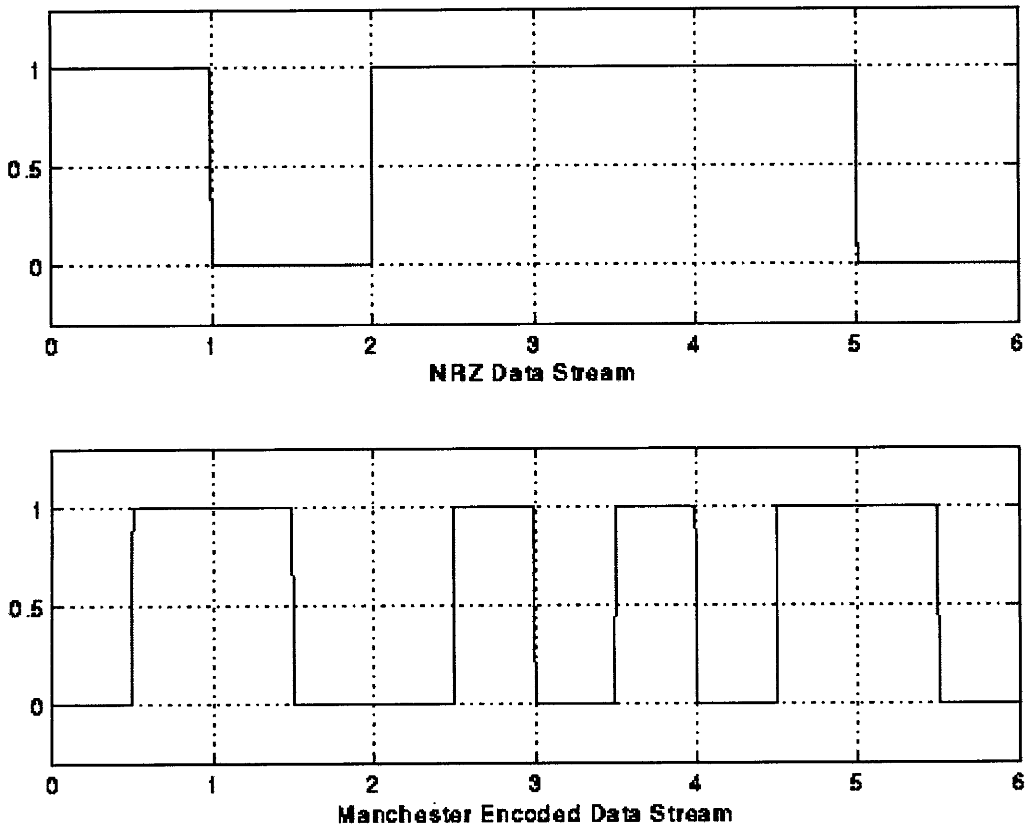


Figure 6-2: Comparison of NRZ Data Stream to Manchester Data Stream

6.1.1.3.2. Modulation

The filtered wideband data stream will then be used to modulate the transmitter carrier with direct binary frequency shift keying (FSK). This is specified such that a binary one will correspond to a nominal peak frequency deviation 8 kHz above the carrier frequency, and a binary zero will correspond to a nominal peak frequency deviation 8 kHz below the carrier frequency.

6.1.1.4. Supervisory Tone Signal Characteristics

There are two supervisory tones that are transmitted, the supervisory audio tone (SAT) and the signaling tone (ST). The supervisory audio tone is a tone at one of three frequencies, 5970, 6000 or 6030 Hz, each with a tolerance of ± 15 Hz. This tone is modulated using narrowband FM, with a deviation of ± 2 kHz. These are set such that each land station is assigned one of those three tones, and adds that tone to each forward voice channel. The mobile station is responsible for detecting, filtering, and modulating the reverse voice channel with that same SAT. The SAT is suspended during transmission of wideband data, but not during transmission of the ST. Detection of the SAT must be performed at least every 250 ms. The ST is a tone within 10 kHz ± 1 Hz, producing a nominal frequency deviation of ± 8 kHz of the carrier frequency, and is clearly also suspended during wideband data transmission.

SAT detection must be made every 250 ms, and must follow the form shown in Table 6-3. This illustrates the 15 Hz tolerance placed upon each SAT tone.

Measured Frequency of Incoming Signal	Measured SAT Determination	Where
$f < f_l$	No valid SAT	$f_l = 5955 \pm 5$ Hz
$f_l \leq f < f_2$	SAT = 5970	$f_2 = 5985 \pm 5$ Hz

$f_2 \leq f < f_3$	SAT = 6000	$f_3 = 6015 \pm 5$ Hz
$f_3 \leq f < f_4$	SAT = 6030	$f_4 = 6045 \pm 5$ Hz
$f_4 \leq f$	No valid SAT	
no SAT Received	No valid SAT	

Table 6-3: SAT Detection Specification

6.1.1.5. Receive Voice Processing

Just as with modulation of a transmitted signal, there are certain specific steps incorporated into the demodulation of a received signal, primarily for the sake of reversal of those alterations made during modulation.

6.1.1.5.1. De-Emphasis

The first of these is a de-emphasis curve, designed to have a nominal -6 dB per octave response between 300 and 3000 Hz. This is simply the reverse characteristic of the pre-emphasis curve.

6.1.1.5.2. Expandor

The second is the expandor portion of a 2:1 syllabic compandor. It is once again referenced to a 1000 Hz from a carrier with a ± 2.9 kHz peak frequency deviation. This undoes the effect of the compressor by converting every 1 dB change in input level to a nominal 2 dB. As per the compressor, there is a nominal attack time specification of 3 ms and a nominal recovery time specification of 13.5 ms. In addition, the expandor succeeds in suppressing receiver noise and interference relative to speech signals. [21]

6.1.1.5.3. Audio Level Adjustment

Finally, the mobile and base stations must both ensure that the receive output levels match the respective 51 dB ROLR and -18 dBm0 levels referred to in 7.1.1.2.1. This once again involves examination of the system as a whole and the physical characteristics of the speaker and enclosure in the mobile unit.

6.1.2. NAMPS

Narrowband AMPS is a Motorola developed system that squeezes three conversations into one cellular channel using analog frequency division multiplexing. Its primary interest in this aspect is as the North American technology analogous to the Japanese NTACS standard, or Narrowband TACS. Channel spacing is only 10 kHz as opposed to the typical AMPS 30 kHz, as can be expected through an understanding of analog frequency division multiplexing.

This channel spacing is accomplished by maintaining the AMPS channels and channel numbers, but adding two additional bits to any communication regarding the channel number, to specify a narrow analog channel centered on the AMPS channel center frequency, or centered on the a frequency 10 kHz above or below the AMPS channel center frequency. This can be seen in Table 6-4.

C 13	C 12	Narrow Analog Channel	Description
0	1	N_L	Channel 10 kHz below N
0	0	N_M	Channel centered on N
1	0	N_U	Channel 10 kHz above N
1	1		Reserved
Transmitter	**Channel Number**	**Narrow Analog Channel Designator**	**Center Frequency (MHz)**
Mobile	$1 \leq N \leq 799$	N_L	0.030 N + 825.000 - 0.010
		N_M	0.030 N + 825.000
		N_U	0.030 N + 825.000 + 0.010
	$990 \leq N \leq 1023$	N_L	0.030 (N - 1023) + 825.000 - 0.010
		N_M	0.030 (N - 1023) + 825.000
		N_U	0.030 (N - 1023) + 825.000 + 0.010
Base	$1 \leq N \leq 799$	N_L	0.030 N + 870.000 - 0.010
		N_M	0.030 N + 870.000
		N_U	0.030 N + 870.000 + 0.010
	$990 \leq N \leq 1023$	N_L	0.030 (N - 1023) + 870.000 - 0.010
		N_M	0.030 (N - 1023) + 870.000
		N_U	0.030 (N - 1023) + 870.000 + 0.010

Table 6-4: NAMPS Channels

Various features of about the NAMPS system, including different deviations, and, most importantly, a new, sub-audio signaling scheme using digital supervisory audio tones and digital signaling tones, are essentially identical to their implementation in NTACS and we'll therefore examine those as they apply to NTACS. NAMPS is brought up here simply in order to examine its correlation to NTACS and to illustrate how the additional channels are incorporated.

6.2. TACS

The other major category we're going to examine are the Total Access Communications System (TACS) standards, JTACS and NTACS. These two standards are used for cellular telephone systems in Japan, and have several subtle differences from the AMPS standard systems. Herein lies the thrust of creating a multi-standard demodulator, as, by adding Japanese TACS capability, we can significantly increase the applicability, and the market, for this product and concept.

6.2.1. JTACS

The logical first of these two is the JTACS standard, which is baseline identical to AMPS, but has a number of differences in specifications and other aspects to be aware of.

6.2.1.1. Channels

The biggest difference between the AMPS standard and the JTACS standard lies in the channel specification and channel widths. JTACS, and NTACS for that matter, use the 800 MHz carrier frequency band, similar to AMPS, but parcels out that band in a different manner. JTACS channels are spaced 25 kHz apart, instead of AMPS' 30 kHz spacing, and that slightly tighter requirement forces many of the modulation specifications to be somewhat altered. However, for JTACS, there are only 399 channels available, those lying within the 10 MHz band, from 860 to 870 MHz for base transmit and from 915 to 925 MHz for mobile transmit. Note that, as opposed to AMPS, the base's transmit channel lies below the mobile's transmit channel. The spacing between a base transmitted channel and a mobile transmitted channel has also grown to 55 MHz as opposed to AMPS' 45 MHz separation. Also, the 399 channels available for JTACS are chosen as the even numbered channels among the possible channel numbers 1 to 799, as seen in Table 6-5.

Transmitter	Channel Number	Center Frequency (MHz)
Mobile	$1 \leq N \leq 799$	0.0125 N + 915.000
Base	$1 \leq N \leq 799$	0.0125 N + 860.000

Table 6-5: JTACS Channels

One further difference worth noting is the change in terminology from AMPS to TACS, wherein channels known in AMPS as adjacent are in TACS known as interleaved, and channels known in AMPS as alternate are known in TACS as adjacent. Therefore, in JTACS, the interleaved channels are those channels with center located 25 kHz above or below one's center frequency, and the adjacent channels are those with center located 50 kHz above or below one's center frequency. The minimum interleaved channel selectivity is specified to be 35 dB, and the minimum adjacent channel selectivity is specified to be 70 dB.

Channels are designated voice or control channels, as in an AMPS environment, and indeed, there are 24 dedicated control channels for JTACS. These are in two groups of twelve channels, and are the even-numbered channels between 418 and 440, and between 520 and 542. These channels used entirely for signaling and data communications, as opposed to the voice traffic that exists on voice channels.

6.2.1.2. Transmit Voice Processing

The modulation process applied to JTACS audio is very similar to that applied to AMPS audio, as mentioned before. The major fundamental difference lies in the 25 kHz channel, as opposed to the 30 kHz channel used in AMPS processing. This has caused the nominal peak carrier frequency deviation to be changed to ±2.3 kHz instead of the ±2.9 kHz used for AMPS. This has a direct impact upon the deviation limiter stage, such that the instantaneous peak frequency deviation must now be limited to ± 9.5 kHz, rather than the 12 kHz for AMPS.

There is additionally now a specified characteristic for the audio bandpass filtering. Whereas bandpass filtering is assumed and necessary for AMPS as well, in order to maintain the 300-3000 Hz banded audio signal, the JTACS standard specifies that the attenuation rate of such a filter be at least 24 dB / octave. This filter must be precede the pre-emphasis highpass filtering, and, indeed, should be the initial step applied to the voice signal.

With the exception of these two differences, however, the remaining voice modulation blocks remain and are the same. Bandpass filtering, compression, pre-emphasis, our newly respecified deviation limiter, and post deviation limiter filtering all are applied to the voice signal prior to combination with signaling and supervision, and modulation.

6.2.1.3. Wideband Data Signals

Wideband data signals are once again used, and dealt with in a similar fashion. They must again be Manchester encoded, such that a non-return-to-zero binary zero be converted to a one to zero transition, and each non-return-to-zero binary one be converted to a zero to one transition. For JTACS, however, there are two differences. The data streams are at 8 kbps instead of the AMPS 10 kbps. Also, the direct binary frequency shift keying modulation is done at 6.4 kHz deviation, as opposed to the 8 kHz binary frequency shift keying of AMPS. This deviation change can be readily dealt with in implementation, although the change in data rate can pose some problems.

6.2.1.4. Supervisory Tone Signal Characteristics

The JTACS SAT and ST are handled in much the same way, with slight changes, once again. The SAT remains one of the three frequencies, 5970, 6000, or 6030 Hz, but it now modulates the carrier with a peak carrier frequency deviation of ± 1.7 kHz. The ST, meanwhile, has been moved to 8 kHz, and is applied with a peak frequency deviation of ± 6.4 kHz. These changes are once again a result of the slightly smaller channel width.

6.2.1.5. Receive Voice Processing

JTACS voice demodulation is functionally identical to AMPS. The performance requirements may be somewhat altered, but there are no differences in the tasks performed, and it can thus be seen as identical to AMPS voice demodulation.

6.2.2. NTACS

The NTACS standard, like the NAMPS standard, uses a narrower channel width, but adds in several other more complex features that must be taken into account. NTACS systems must, by specification, be dual mode, and incorporate JTACS functionality as well, introducing the issue of handling handoff procedure between NTACS and JTACS, and indeed, in a dual-mode system with digital Code Division Multiple Access (CDMA) technology, to and from CDMA as well. Also, as a result of the smaller channels and a 3

MHz range that is added to the valid NTACS band, the channel identification number requires a 12th bit, an issue that should be considered in adjusting existing AMPS systems.

6.2.2.1. Channels

The NTACS channel space builds upon the JTACS channel space described in 6.2.1.1. The two 10 MHz regions from JTACS, from 860 MHz to 870 MHz and from 915 MHz to 925 MHz are set aside as basic channel spaces, for base and mobile transmit, respectively. An additional 17 MHz band is also available, being from 843 to 860 MHz for base transmit and from 898 to 915 MHz for mobile transmit. NTACS capable mobile stations will be required to operate in an undetermined 3 MHz range, or another 239 channels, within that band. These will be the channel numbers 801 to 2160, with the channel numbers 0 and 800 undefined. Furthermore, the entirety of the 799 channels in the basic 10 MHz region are available, as each channel is considered to be 12.5 kHz wide, one half the size of JTACS channels. These channels are shown in Table 6-6.

Transmitter	Channel Number	Center Frequency (MHz)
Mobile	1 ≤ N ≤ 799	0.0125 N + 915.000
	801 ≤ N ≤ 2160	(N-800) (0.0125) + 898.000
Base	1 ≤ N ≤ 799	0.0125 N + 860.000
	801 ≤ N ≤ 2160	(N-800) (0.0125) + 843.000

Table 6-6: NTACS Channels

As mentioned in 6.2.1.1., the TACS terminology for nearby channels differs from the AMPS terminology, and NTACS follows the same format as JTACS. As such, the interleaved channels are those channels with center located 12.5 kHz above or below one's center frequency, and the adjacent channels are those with center located 25 kHz above or below one's center frequency. The minimum interleaved channel selectivity is specified to be 35 dB, and the minimum adjacent channel selectivity is specified to be 70 dB.

The dedicated control channels referred to in JTACS remain dedicated control channels in NTACS.

6.2.2.2. Transmit Voice Processing

NTACS voice modulation is identical to JTACS voice modulation in all ways except the instantaneous peak frequency deviation, which must be limited to ± 5.0 kHz instead of the ± 9.5 kHz for JTACS and the ± 12 kHz for AMPS mobile stations. This is in order to accommodate for the once again smaller channel width of 12.5 kHz. However, that is the only difference, as it is primarily in signaling that NTACS differs from JTACS and AMPS.

6.2.2.3. Wideband Data Signals

NTACS wideband data signals are produced at the same 8 kbps as JTACS wideband data signals, and are encoded with the same Manchester encoding scheme. However, the binary frequency shift keying modulation scheme is dependent in NTACS upon the type of channel. On a control channel, the filtered data stream will be used to modulate the transmitter carrier with a deviation of 6.4 kHz, such that a binary one will correspond to a nominal peak frequency deviation of 6.4 kHz above the carrier frequency, and a binary zero will correspond to a nominal peak frequency deviation of 6.4 kHz below the carrier frequency. On a voice channel, however, the nominal peak frequency deviation will be ± 0.7 kHz.

6.2.2.4. Supervisory Signals

The main difference and addition to NTACS that differentiates it from both AMPS and JTACS is the use of a digital SAT (DSAT), and a digital ST (DST), for the purpose of supervisory communication. These supervision and signaling tones are encoded in such a way in order to be transmitted in the 0 to 300 Hz band below the audio bandwidth of 300 to 3000 Hz, and thus enable these narrowband channels. Their purpose, however, is exactly analogous to the AMPS and JTACS SAT and ST, and their origination lies in the call processing that occurs in the system before the scope of this work.

6.2.2.4.1. DSAT/DST Transmission, detection

The DSAT is one of seven digital sequences added to the voice transmission, as opposed to the JTACS and AMPS choice of one of three individual tones. The DSAT is transmitted at 200 NRZ bits per second, and are added to the overall signal by producing an average peak deviation of 700 Hz ± 70 Hz The DSAT is continually transmitted by both the mobile and base stations except during transmission of the sync words, data words or the DST. When discontinued, DSAT phase must be maintained such that the phase will be correct when continued. The DSAT sequences are listed in Table 7-6.

The DST is one of seven digital sequences as well, consisting of the logical inverses of the seven DSAT sequences, and therefore also at 200 NRZ bits/second. Bit sync and phase between DSAT and DST must also be maintained as they are not both transmitted at once, but when moving from one to the other, the appropriate phase is kept. The DST is also transmitted through modulation of the voice carrier with the sequence at an average peak deviation of 700 Hz ± 70 Hz.

6.2.2.4.2. DSAT/DST Transition

Transitions between transmission of DSAT and DST must also be carefully set up so as not to disturb the phase of the DSAT vector, and not cause false detection. This is done through the use of specified bit masks, corresponding to each of the DSAT/DST pairs of sequences. Inversion, if desired, is then allowed only at a positive mask bit for that sequence. The bit masks, along with their corresponding DSAT vectors and DST vectors, are shown in Table 6-7.

Sequence Number	DSAT Vector	DST Vector	Transition Bit Mask
#0	2556CB	DAA934	FF003E
#1	255B2B	DAA4D4	0BBF82
#2	256A9B	DA9564	BD780F
#3	25AD4D	DA54B2	3FF118
#4	26AB2B	D954D4	0AE6F6
#5	26B2AD	D94D52	8001FF
#6	2969AB	D69654	1C0FCD

Table 6-7: NTACS DSAT/DST Sequences

The interruption of the DSAT by wideband data does not require adherence to the DSAT bit masks as does interruption by the DST.

6.2.2.5. Receive Voice Processing

NTACS voice demodulation is functionally identical to JTACS. The expected peak frequency deviations are different, as described in voice modulation, but apart from that, the stages and expectations are identical.

Chapter 7. System Simulation

Now that we have a firm understanding of the standards involved in this discussion and in the requirements which must be met, an examination of the procedure and results from the process of simulation can be examined. The ultimate goal is the development of a digital IF FM demodulator, using delta sigma techniques, to perform demodulation of signals of AMPS, JTACS and NTACS signals, and the means by which this was accomplished and examined was through C-language code that simulated and illustrated the capability of such a system.

Once again, we shall begin with an examination of the AMPS standard, and the development of a system to perform AMPS modulation and demodulation in order that our starting point be well determined and successful. A full modulation scheme must be simulated in order to arrive at the appropriate input for our system, and it must be approached with an eye to being adjustable for the purpose of the standards which are to follow. However, it is true that our demodulator input is a limited IF signal, and as such, we must only produce a limited IF output from our modulator.

Once a functional AMPS standard system has been developed, we can extend it for the sake of incorporating the additional capability of JTACS and NTACS. Most of this work at this juncture lies in the process of breaking apart the signal into its audio and signaling portions and properly processing and formatting those, as the demodulation itself has already been shown to be effective. However, this is also necessary in order to further our justification in the proper functionality of that demodulation process.

The main tools for this work were once again the C programming language and Matlab, as per section 5. The code involved in this aspect of the development is attached in Appendix B, and the system used was the same as for the simulation of the demodulation scheme.

While the code used for these simulations was grounded in functionality and practicality rather than elegant or streamlined programming aesthetic, there are several points that are worth pointing out. All of the code used in these simulations is quite heavy on overhead, featuring the ability to alter a great many factors and aspects of the code from the 'defines.c' file. This allows access to whichever debugging files are required in a particular run, and implementation only of whatever segments of the code are of concern at any particular run. Also, the channel files, consisting of the fully modulated signal, is stored as a binary file in order to keep it as small as possible. These sorts of aspects to the code are quite important for two main reasons, the first being that considerable alterations and adjustments need to be made throughout development, making it desirable to make changes quickly and easily. The second major reason for some of these factors is the fact that at our highest sampling point we are sampling at 19.68 MHz, an interpolation factor of 2,460 from our audio input sampling frequency of 8 kHz. As such, our interim files and our loop iterations can get quite large, and the simulations can become very slow. Even a small audio input sample set of 500 points will result in a demodulator input of 12,300,000 points. Therefore, it is important to seek out speed benefits wherever possible. The elimination of debug files that will not be used is a tremendous boon, especially with regards to those debug files at higher frequencies. And the storage of 12 million points using ASCII format results in approximately 20 bytes/sample, or, files of around 24 MB in size. Storing those intermediate files as unsigned integers can gain a benefit resulting in that same file being approximately 5 MB in size, a considerable gain. Further benefit could be garnered by encoding data as single bits, as each data point is only one bit of information, and, indeed, would be an excellent next step beyond what was already done, were additional speed benefits necessary.

7.1. AMPS Simulations

Our initial effort, as mentioned, is the development of a purely AMPS system, and as such we will concern ourselves at first only with that aspect of the work. The fundamental system involves the creation of input signals, modulating those signals, adding whatever channel noise inherent to transmission might be applicable, and then the demodulation and recovery, to within desired specifications, of those original signals. This basic top scale block diagram can be seen in Figure 7-1.

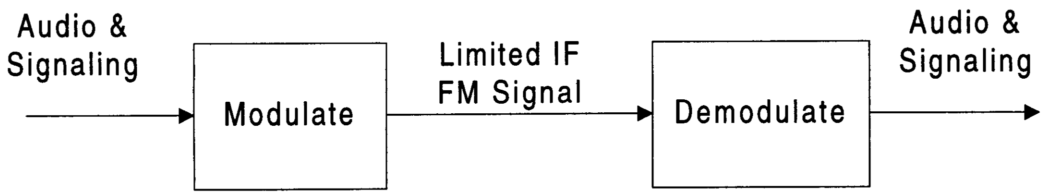


Figure 7-1: Overall Simulation Block Diagram

7.1.1. Transmit Processing

The modulation process exists in these simulations primarily for the sake of providing our demodulation system an input signal, and as such is done with an eye towards keeping the system basic and typical, and very specification oriented. We accept three inputs, with varied amounts of preprocessing done on each. The first is an 8 kHz audio signal, at first only a tone for the purpose of testing, then more complex audio signals in later testing, expected to consist of a stream of double length floating point numbers, ranging from -1 to +1. The second is a randomized digital wideband data stream at the specified 10 kbps, again simply floating point numbers file, with inputs of 1 or 0. And the third is a steady SAT tone of either 5970, 6000 or 6030 Hz, at a sampling frequency of 120 kHz, one of the frequencies in our system. The SAT tone is kept steady as it is not expected to be changing at anything close to the frequencies of our simulations. It is also a floating point input file, and expected to range from -1 to +1. The output of the modulation process is the limited IF FM signal that doubles as an input to the demodulation system. All three of these input files were produced using combinations of Matlab and simple peripheral C-code.

Figure 7-2 illustrates the general block diagram for the modulation processing which will be examined in depth in this section. The main blocks will remain constant for our discussions of JTACS and NTACS, and as such, the vast majority of this block diagram will remain relevant for those discussions as well.

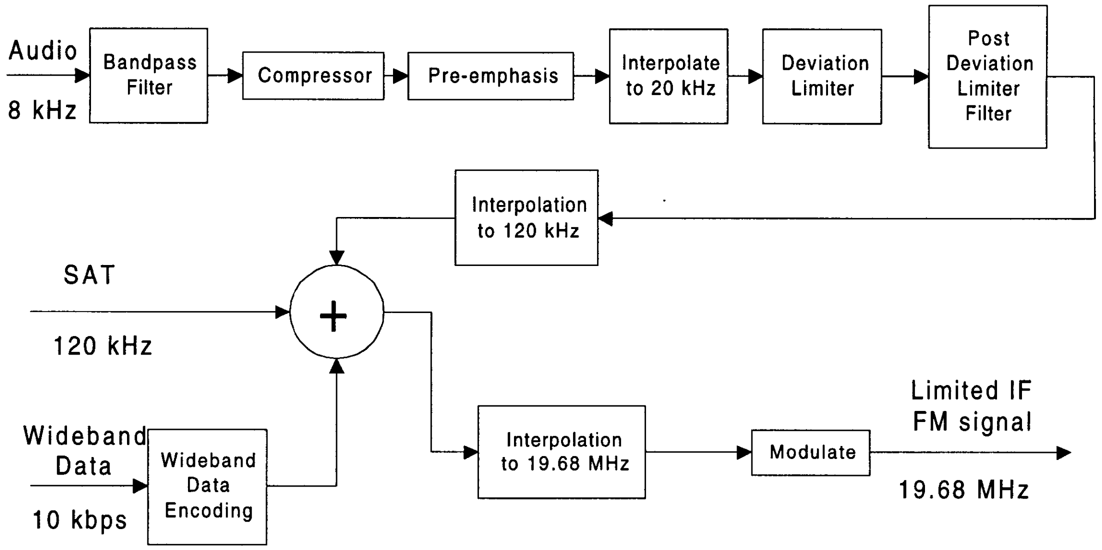


Figure 7-2: Transmit Block Diagram

7.1.1.1. Voice Processing

As is evident from the block diagram, the majority of the processing performed in this code is done on the voice signal, the steps for which were seen in our discussion of the AMPS standard itself. A few wrinkles have been added in the implementation of the system, however.

The first of these is the Bandpass Filter, limiting our audio input to the 300 - 3000 Hz range which was expected. This is not a change from any earlier situation, but simply an explicit implementation of this stage.

The other major one of these is the varied sampling frequencies used in the process. This arose from the realities of the signals we are dealing with. The first is an audio input at 8 kHz, so chosen because of its common nature in such applications and because it allows for effective and simple bandpass filtering. However, a sampling frequency of 8 kHz is below the Nyquist frequency required for the SAT tone of around 6 kHz, and therefore we chose interpolation to 20 kHz for addition of the SAT signal. The deviation limiter functions benefits from the higher sampling frequency, in order to minimize the margin required for peak values which don't necessarily coincide with the discrete sample points, and is thus placed after that interpolation. The SAT tone was then increased to a sampling frequency of 120 kHz, but the 20 kHz frequency was maintained for the purpose of the deviation limiter and post deviation-limiter filter. The 120 kHz frequency is necessary for the inclusion of the wideband data stream. The wideband data, as will be seen, is smoothed out in order to diminish the low frequency components which will interfere with our audio signal, and as such requires a sampling frequency even higher than the 20 kbps created by Manchester encoding the 10 kbps signal. Once the three independent signals have been combined at 120 kHz, they must be interpolated to 19.68 MHz, our IF sampling frequency prior to modulation.

7.1.1.1.1. Bandpass Filter

The bandpass filter is designed to bandlimit the audio input to the nominal audio bandwidth of 300 to 3000 Hz. The choice of implementation was a 6th order Butterworth bandpass filter, with frequency response as seen in Figure 7-3. The bandwidth was chosen to be 200 Hz to 3300 Hz in order to limit the response at 300 Hz and at 3000 Hz to -0.5 dB. The combination of the 3rd order roll off in the sub-300 Hz section of this filter and the 1st order roll off of the pre-emphasis curve will yield a 4th order roll off below 300 Hz, or 24 dB/octave, as is consistent with current commercial designs.

The filter coefficients were found by using the Matlab butter command, which produces the numerator and denominator vectors for our IIR. The implementation of an IIR filter, or indeed an FIR filter, in C code was tested and then used extensively throughout this work, and the same block of code will be seen in a number of locations throughout.

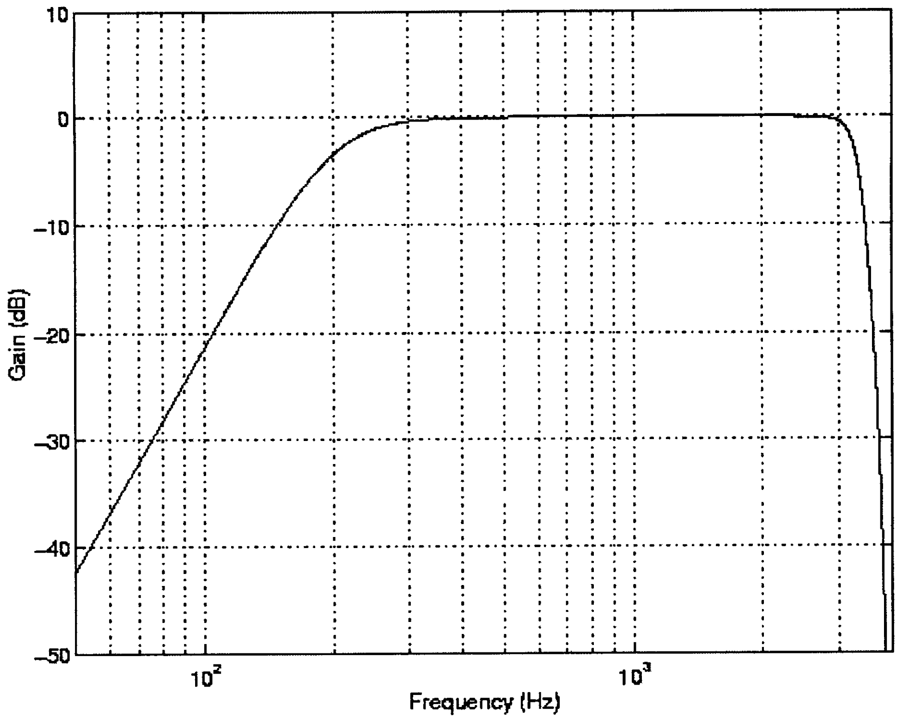


Figure 7-3: Audio TX Bandpass Filter Response

7.1.1.1.2. Compressor

The compressor is implemented with a digital version of a typical analog feedback implementation. This is illustrated in Figure 7-4. The key design issue is the choice of loop filter time constant τ, which is chosen so as to achieve the attack and decay times of 3ms and 13.5 ms, respectively. The remaining constants in the design are K_c, which is set to $\pi/2$ for a normalized 0dB reference of 2 units peak-to-peak, and a, as calculated below:

$$a = \frac{\left(\frac{1}{f_s}\right)}{\tau} = 0.000625$$

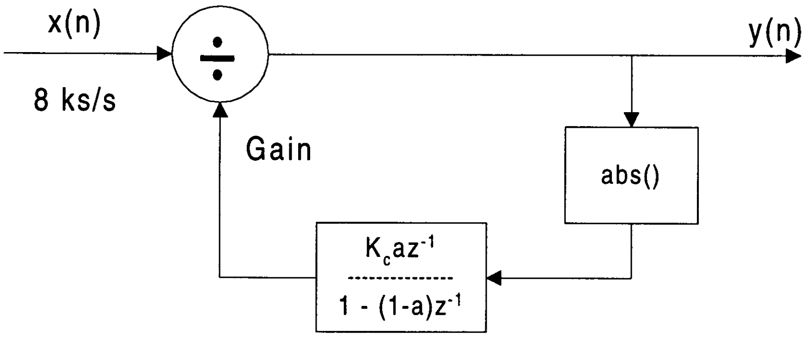


Figure 7-4: Compressor Block Diagram

7.1.1.1.3. Pre-Emphasis

The pre-emphasis characteristic is achieved with a 1st order IIR high-pass filter. This was chosen rather than a 2-tap FIR differentiator in order to avoid the high-frequency droop given the low sample rate relative to the 3000 Hz specification. The transfer function is therefore:

$$H(z) = \frac{k(1-z^{-1})}{(1+a_1 z^{-1})}$$

The constants were set with k chosen to maintain a 0 dB gain at 1 kHz, and a_l chosen for the +6 dB/octave. The frequency response can be seen in Figure 7-5.

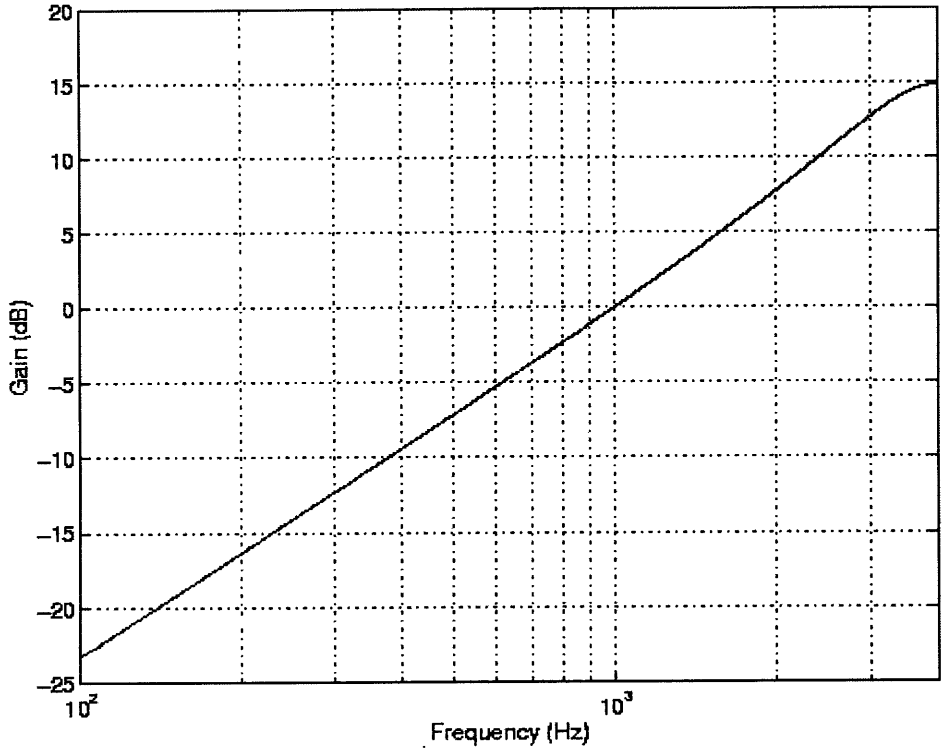


Figure 7-5: Pre-Emphasis Response

7.1.1.1.4. Interpolation to 20 kHz

The interpolation from 8 kHz to 20 kHz is the first sampling frequency change in this system, but carries the fundamentals that influence the design of all our interpolators and decimators. An interpolation from 8 kHz to 20 kHz adds the additional complexity of being a non-integer interpolation, implemented with a 5x interpolator, followed by a 2x decimator.

An interpolator is composed of a change in sampling rate, inserting either zeros between each sample, or by maintaining one sample for multiple interpolated samples (which will carry with it some filtering of its own), and filtering to eliminate aliased copies of the spectrum. This change in sample rate is done in code through a nested for loop for sample rate escalation, and through an if clause for sample rate decreasing. The filtering in an interpolation can then be done as usual.

A 2:5 interpolation ratio, therefore, is accomplished by upsampling the signal, low pass filtering, and then decimating by 2. This is done with a 5x for loop and a clause that skips every other sample, after low pass filtering.

The interpolation filter used for this case is a 30-tap FIR, designed with a sinc function and Kaiser window. The frequency response is shown in Figure 7-6. Note the 3 kHz roll-off of -0.8dB, the > -50 dB stopband attenuation, and the zero at the SAT frequency of 6 kHz.

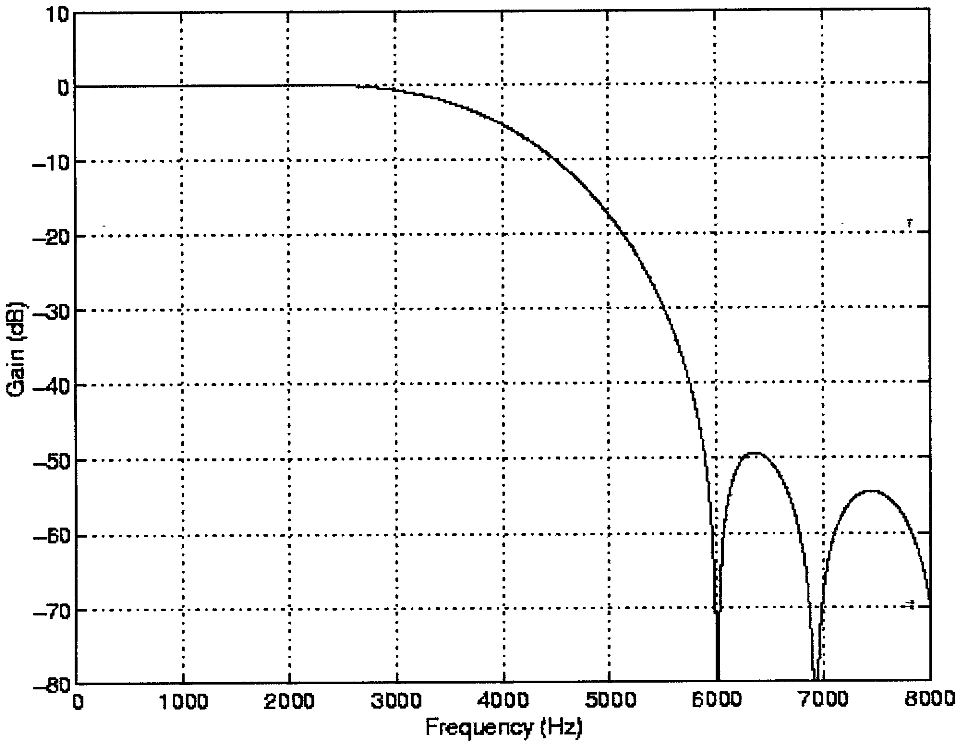


Figure 7-6: Audio TX Interpolation to 20 kHz Response

7.1.1.1.5. Deviation Limiter

The deviation limiter exists for the purpose of limiting the audio modulation to a maximum instantaneous frequency deviation of 12 kHz, as per the standard. This is implemented as a threshold function, whereby any audio value of absolute magnitude greater than a set threshold is clipped to that threshold, thus limiting the eventual frequency after modulation. The threshold to apply is found via:

$$Threshold = \frac{Maximum\,Frequency}{Audio\,Deviation} = \frac{12\,kHz}{2.9\,kHz} = 4.1379$$

7.1.1.1.6. Post Deviation-Limiter Filter

The implementation of the post deviation-limiter filter is well defined in the specifications, and was simply chosen to meet those. A direct lowpass filter was chosen, as a 5th order Butterworth, with cutoff frequency of 3.15 kHz, in order to try and attain close to 0dB attenuation at 3 kHz. The coefficients of this filter were once again found using Matlab. The frequency response of this filter can be seen in Figure 7-7.

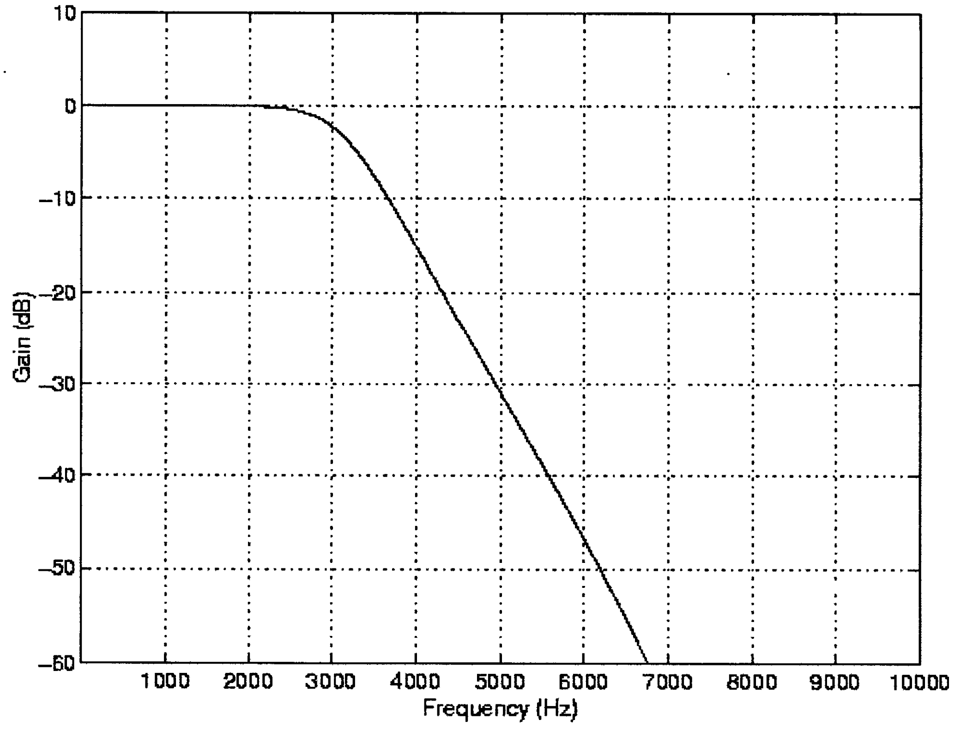


Figure 7-7: Post Deviation-Limiter Filter Response

7.1.1.1.7. Interpolation to 120 kHz

A further interpolation is then introduced to reach the 120 kHz sample rate of the wideband data sequence, as will be seen. This is done using a set of 3 cascaded sinc filters. The first of these is accomplished as described in section 7.1.1.1.4., by using a zero order hold immediately before the 6x upsampling. This is then passes to two other sinc filters of length chosen to place the filter's zeros as appropriate. These were chosen as sinc filters are very easy to implement, both in hardware and in code, and would therefore be a effective choice in an implemented system. The frequency response of this interpolation filter can be seen in Figure 7-8.

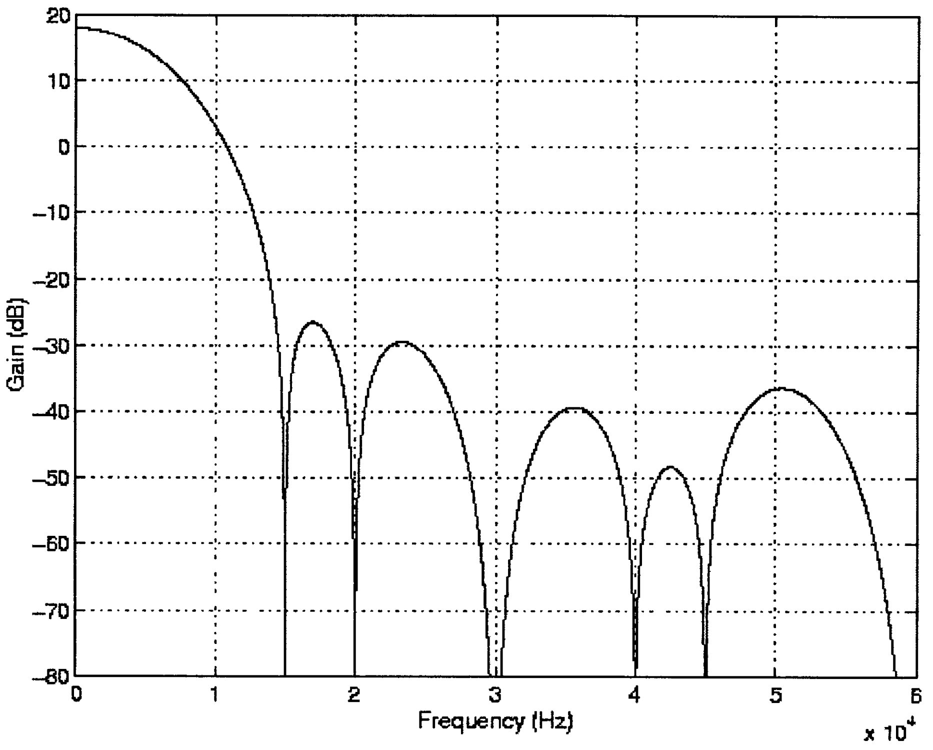


Figure 7-8: Interpolation to 120 kHz Filter Response

7.1.1.2. Supervisory Audio Tone

The second component to this signal is the SAT, read in, as stated above, at 120kHz. This is the frequency at which our wideband stream data is created, as will be seen in section 7.1.1.3., and with the SAT as a simple steady sine wave, it was seen as simplest to incorporate the tone at that sampling frequency. The modulation code therefore has no processing to perform on the SAT other than reading it as input.

7.1.1.3. Wideband Data Signals

The wideband data signals require a considerable amount of processing, from the 10 kbps input stream to a 120 kHz smoothed waveform. The input data stream, an untouched digital stream, must be Manchester encoded, as mentioned in section 7.1.1.3.1., and must also be softened into a sinusoidal waveform so as to eliminate high frequency transients created through step changes. This is done using a waveform generator in order to carefully ensure the appropriate encoding, and eliminate any danger of amplitude variations that a lowpass filter could introduce.

The waveform generator is shown in Figure 7-9, and indicates the Manchester encoding of NRZ ones to zero to one transitions and NRZ zeros to one to zero transitions. Each data symbol is encoded as 12 points generally matching a sine wave. However, a smoothed waveform will require knowledge of the next data point by midway through its waveform. Therefore, input data is read every 12th output of the encoder, and each set of 12 outputs is made up of 6 points from the prior symbol and 6 points from the current symbol. This introduces a half bit delay.

Figure 7-9 indicates the results from each of the four possible sequences of bits, zero to zero in the top left, zero to one in the top right, one to one in the bottom left, and one to zero in the bottom right. Note the dependence of each bit's waveform upon the next bit. Note also the fixed amplitude of -1 to +1.

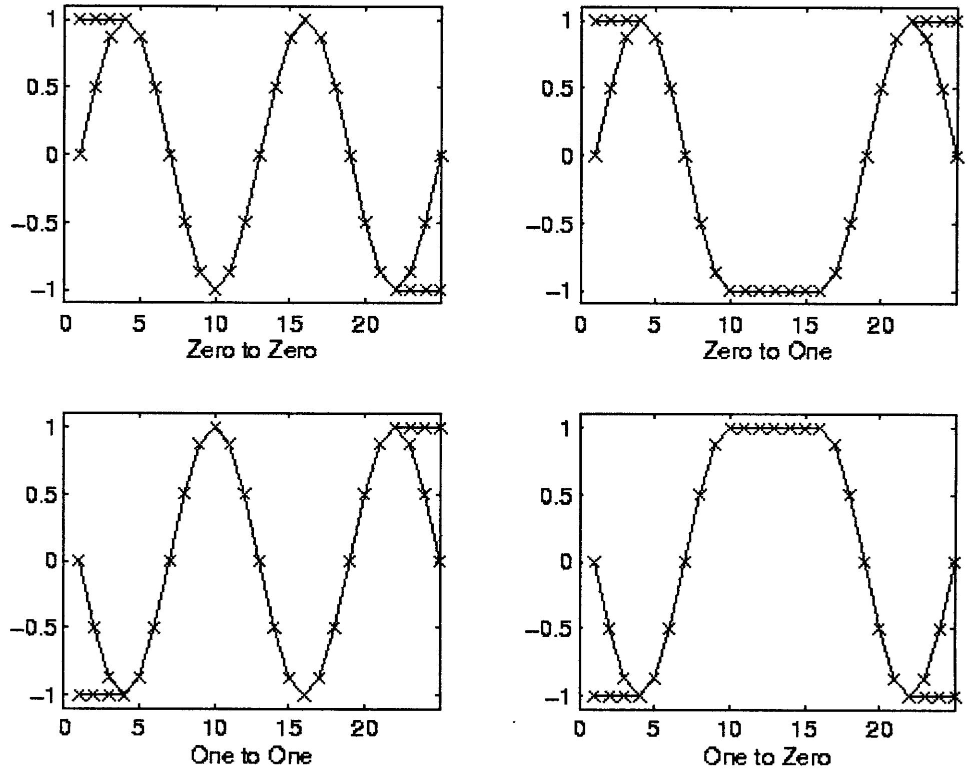


Figure 7-9: AMPS Wideband Data Waveform Generator

An explicit description of this is shown in Table 7-1.

X(n-1)	X(n)	Wide Band Data Waveform											
		Second half of X(n-1) Waveform						First Half of X(n) Waveform					
0	0	0	-.5	-.866	-1	-.866	-.5	0	.5	.866	1	.86	.5
0	1	0	-.5	-.866	-1	-1	-1	-1	-1	-1	-1	-.866	-.5
1	0	0	.5	.866	1	1	1	1	1	1	1	.866	.5
1	1	0	.5	.866	1	.866	.5	0	-.5	-.866	-1	-.866	-.5

Table 7-1: Wideband Data Waveform Generator Function

The implementation of this in C code is done by operating at a frequency of 120 kHz, and reading in a new symbol every 12th run through. The output is simply read out from a set of the four possible transitions, chosen according to the location in the run and the current and previous symbols.

Figure 7-10 is an illustration of the progression of the wideband data stream from 10 kbps digital stream to Manchester encoded data to softened sinusoidal waveform. Although the progression occurs all in one step, the theoretical understanding is better seen in the two stages.

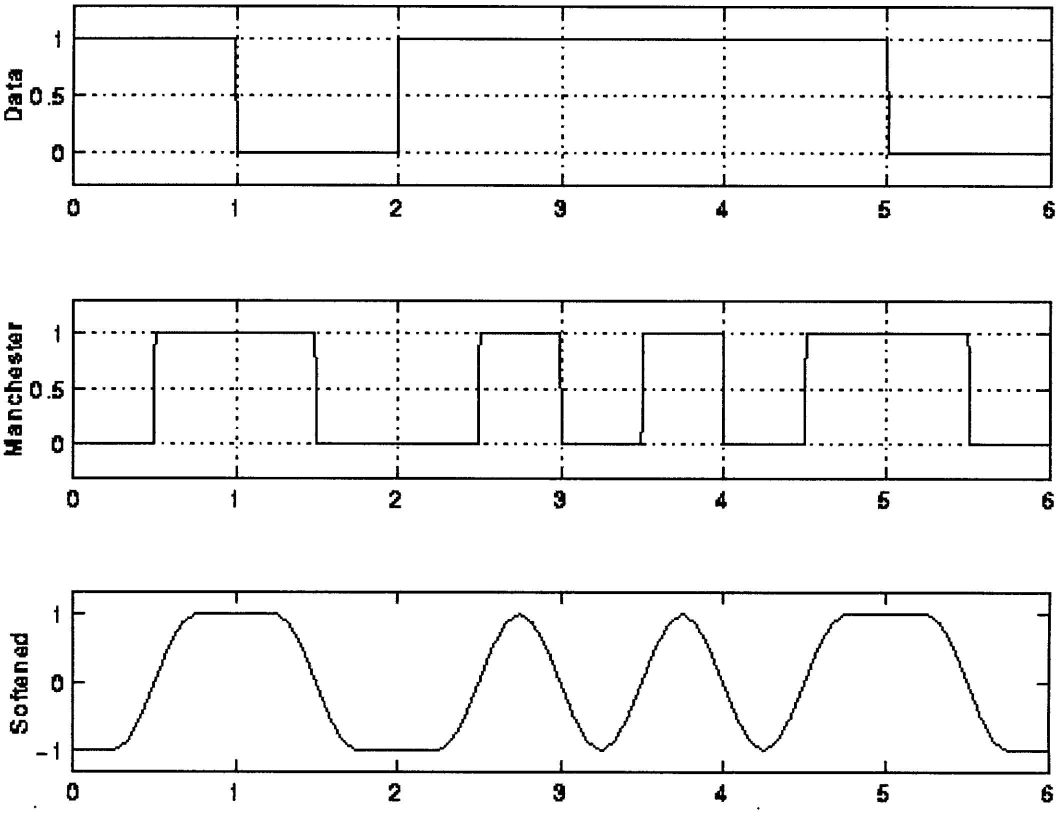


Figure 7-10: AMPS Wideband Data Waveforms

7.1.1.4. Modulation

The final step in the transmit process is the actual modulation, converting these three signals into a unified signal which could be transmitted, or, rather, in our case, will be passed to the demodulation receive process in order to test that segment of our system. This involves interpolation to our oversampling frequency of 19.68 MHz, followed by applying a version of the typical FM modulation scheme that takes advantage of the fact that we only require a limited IF FM signal.

7.1.1.4.1. Signal Combination

The initial step in modulation is the combination of these three signals we have (audio, SAT, wideband data) into one combined signal. Through our efforts in signal processing, we have created three signals, all of which nominally range from -1 to +1. This is guaranteed for the SAT and wideband data, and the audio ranges, but was specified as the nominal frequency deviation. There will clearly be a range, given the variance inherent in standard spoken speech [21], however, the deviation limiter will ensure that those are not excessive. We wish to apply three different nominal peak frequency deviations to these signals, and that is accomplished by scaling each of them by that factor at this point in time, as will be seen more specifically in 7.1.1.4.3. and our discussion of the modulation process. However, once scaled and combined, we must be very careful in our interpolation to ensure that we do not alter these amplitudes in any way.

7.1.1.4.2. Interpolation to 19.68 MHz

Interpolation from 120 kHz to 19.68 MHz is interpolation by a factor of 164, which we break down into two interpolations by 2 and one interpolation by 41. Each has its own interpolation filter, and breaking the interpolation apart into smaller stages in this manner allows us to simplify those filters and succeed without requiring exceptionally complex interpolation filters.

The two interpolations by 2, from 120 kHz to 240 kHz, and from 240 kHz to 480 kHz are identical in all ways. They are both done using 36 tap fir filters, the coefficients of which were found as follows:

```
b = fir1(36, .5);
```

where b contains the coefficients for the FIR filters. The cutoff frequency of .5 * sampling frequency is chosen as the appropriate frequency of $\pi/2$, and the use of a FIR filter is simple to implement.[11] Note also the factor of two lost in amplitude due to the interpolation by two and resultant zero-stuff, which is made up by justifying the signal after filtering. The frequency response of these interpolation filters can be seen in Figure 7-11.

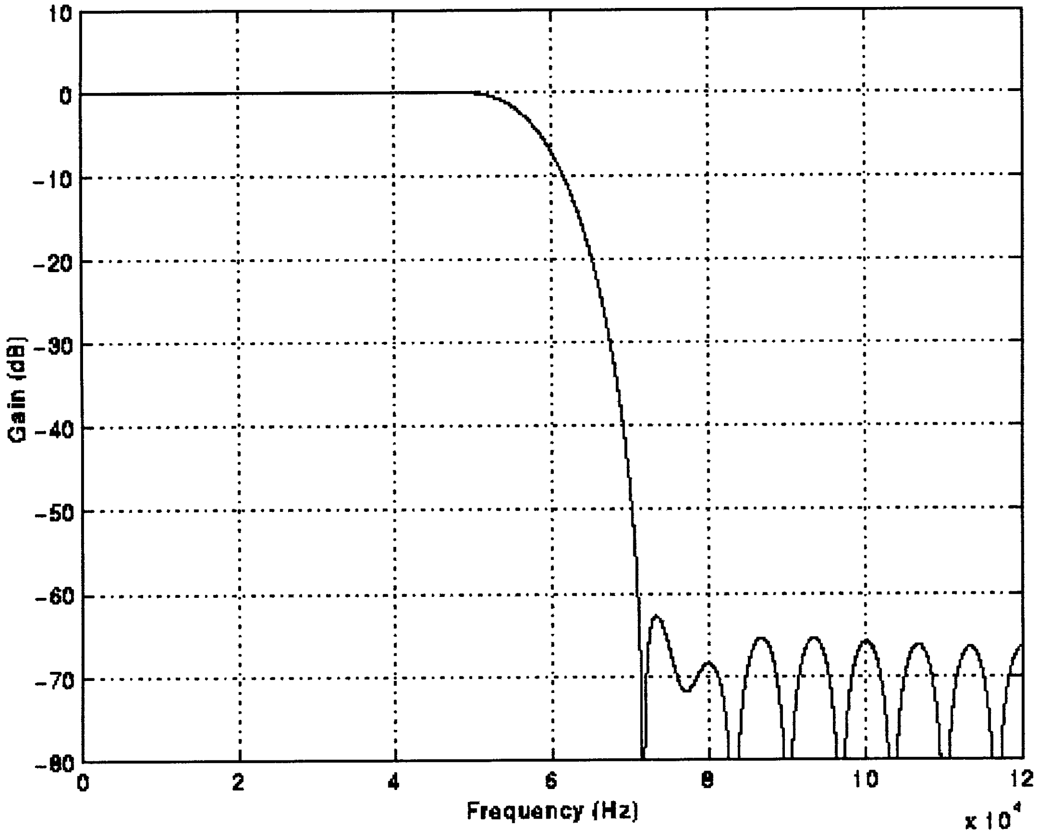


Figure 7-11: TX Interpolation Filter Response

The third interpolation, by 41, involves the use of a sinc^2 filter, using a zero order hold for the first sinc and an explicit sinc filter for the second. This filter is effective in removing aliases, while being simple to implement and maintaining a constant amplitude.

7.1.1.4.3. Modulation to limited IF FM

Finally, our signal is ready for the modulation process. The standard FM modulation scheme as seen in Chapter 2 is represented by these formulas [12]:

$$y(t) = A\cos\theta(t)$$

$$\frac{d\theta(t)}{dt} = \omega_c + k_f x(t)$$

where ω_c is the carrier frequency and k_f the peak nominal frequency deviation. Therefore, through our combining in section 7.1.1.4.1., we have produced k_f * x(t). This is then added to the FM_IF, as the elimination of downconverting from the carrier frequency to IF in the receiver allows us to treat the IF as our carrier frequency. We then take advantage of our knowledge of the cosine function, and the fact that between 0 degrees and 180 degrees, the cosine function is positive, and between 180 degrees and 360 degrees, the cosine function is negative. We therefore create a running sum of 360 * our θ(t), thereby integrating it, and simply flip the sign of our output every time we pass 180 degrees. This produces the sign of our output, or rather, a stream of ones and zeros correlated to the sign of our output, which is all we need in a limited IF FM situation. This then, is our final, processed and modulated transmit output.

7.1.2. Receive Processing

The process of simulating demodulation in this case is an exercise in signal recovery and detection, as our delta sigma demodulation scheme itself was simulated in

Chapter 5. The code performing that aspect was incorporated directly from the code for Chapter 5, and we therefore pick up this discussion starting with a 40 kHz sampling frequency demodulated signal. This signal incorporates the three components, although it is important to remember that the SAT is muted during transmission of wideband data.

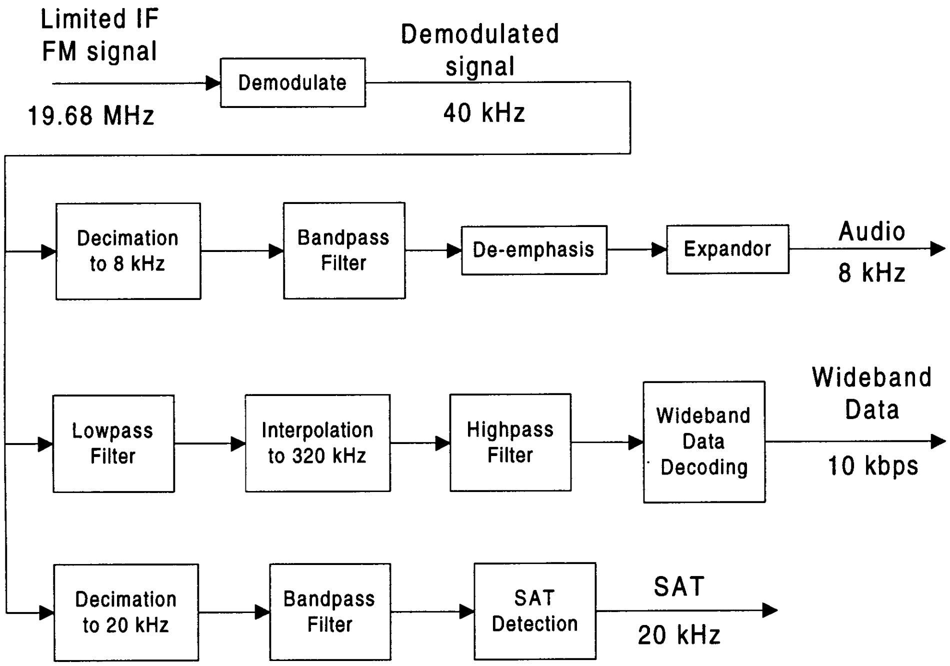


Figure 7-12: RX Block Diagram

7.1.2.1. Voice Processing

7.1.2.1.1. Decimation to 8 kHz

The first step in the processing of the audio signal is to decimate to the 8 kHz at which the majority of the transmit processing was performed. It is also at this frequency that we will perform receive voice processing, namely, those elements which serve to reverse the effects of the transmit processing.

The decimation is by a factor of 5, which is easily arranged, and the decimation filter is akin to the one used for the 8 kHz to 20 kHz interpolation filter in the transmit stage. However, this time we use a 50-tap implementation of a Kaiser windowed sinc function.

The frequency response can be seen in Figure 7-13. Note the 6 kHz zero at the SAT tone of 6 kHz, the stopband attenuation of 50 dB, and the roll off at 3 kHz of -0.7 dB.

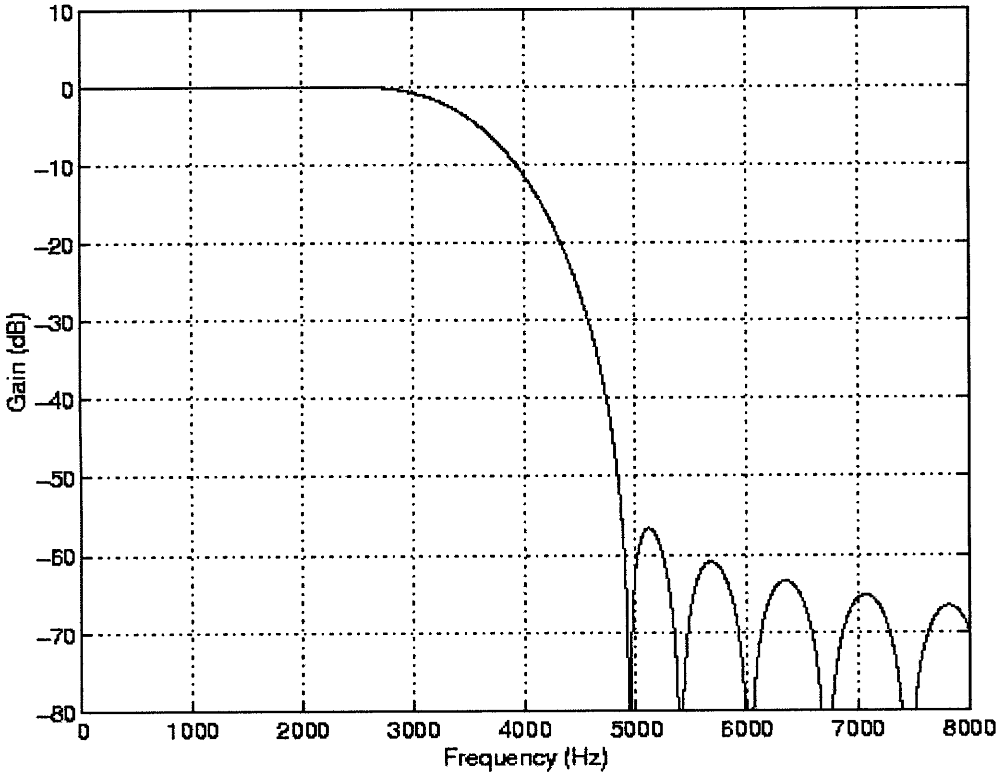


Figure 7-13: Audio RX Decimation to 8 kHz Filter Response

7.1.2.1.2. Band Pass Filter

The next necessary step is the inclusion of a band pass filter to specify only the 300 Hz to 3000 Hz desired for the audio range. A considerable amount of the work here has already been done in the decimation to 8 kHz, however, it is still desirable to attain an audio frequency response equivalent to a 4th order HPF at 300 Hz and a 6th order HPF at 3 kHz. This is done through the convolution of a Butterworth 5th order HPF, with cutoff frequency at 280 Hz, with a 3rd order LPF, with cutoff frequency of 3100 Hz.

Another useful benefit of the bandpass filter is the elimination of the DC offset created by the delta sigma demodulation scheme relating to the carrier frequency. This therefore centers the audio signal at zero. The frequency response can be seen in Figure 7-14.

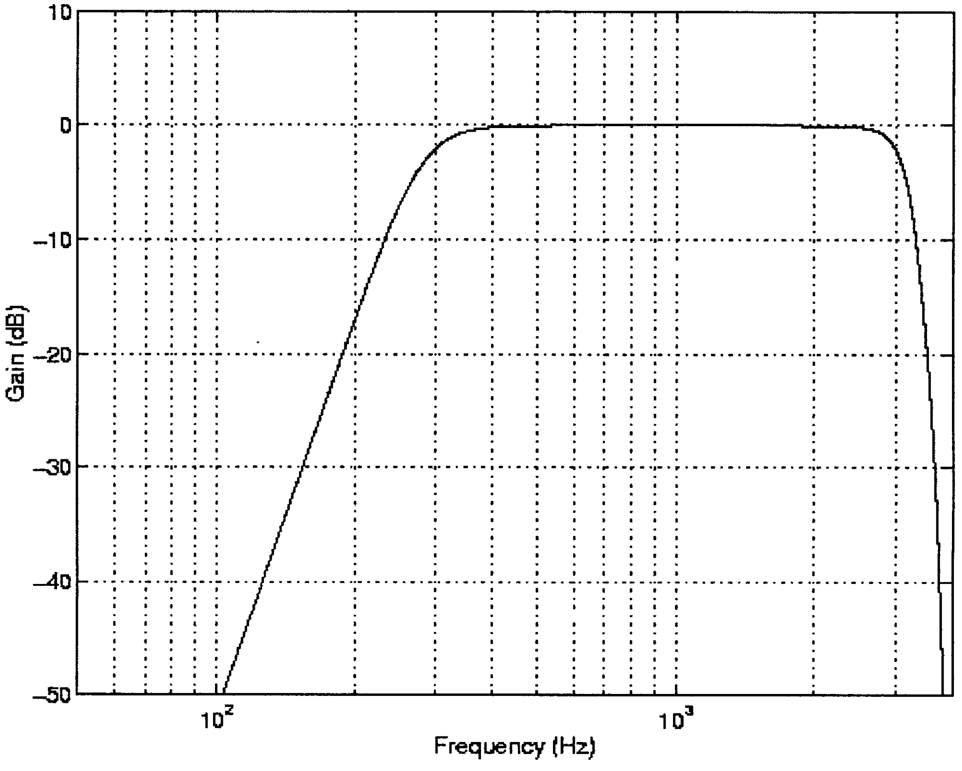


Figure 7-14: Audio Receive Bandpass Filter Response

7.1.2.1.3. De-Emphasis

The de-emphasis characteristic is clearly specified by the standard, and is accomplished using the most direct implementation. The characteristic of -6dB/octave over the audio bandwidth is accomplished with a single pole low pass IIR, of transfer function:

$$H(z) = \frac{k}{\left(1 + a_1 z^{-1}\right)}$$

with k chosen as per the pre-emphasis curve to maintain a 0 dB gain at 1 kHz, and a1 chosen for the -6dB/octave slope. The simple nature of the single pole low pass IIR allows us to implement it in code directly as a function of the input and of the previous output, rather than dealing with our ready made IIR implementation involving extensive array structures. The frequency response of the de-emphasis stage can be seen in Figure 7-15.

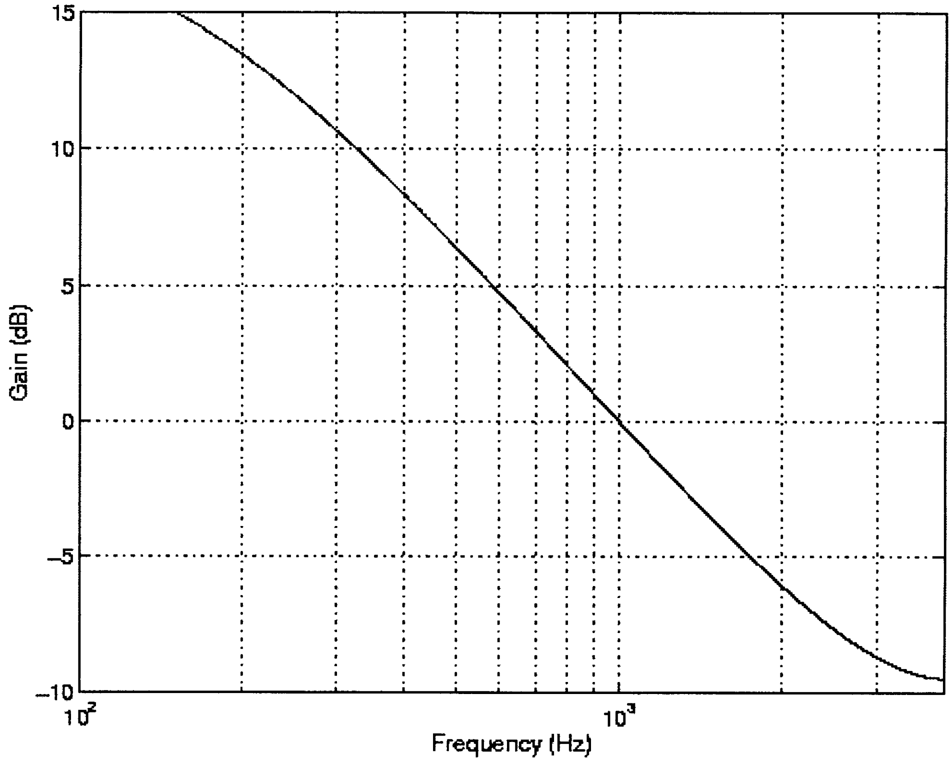


Figure 7-15: De-Emphasis Response

7.1.2.1.4. Expandor

The expandor portion of a 2:1 syllabic compandor is done in much the same way as the compressor, as the digital equivalent of a standard analog feedback implementation. Shown in Figure 7-16, the design once again is built around the attack and delay times of 3 ms and 13.5 ms. K_e is $\pi/2$, as per the compressor, and a is once again 0.00625.

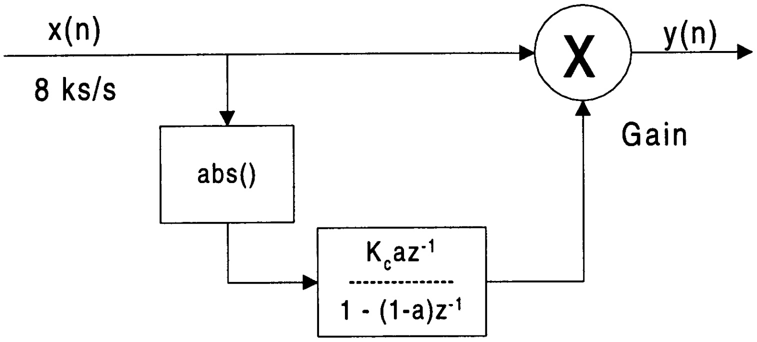


Figure 7-16: Expandor Block Diagram

The output of the expandor is the final audio signal. All the preprocessing done during the transmit stage in order to aid the transmission of the audio signal has been undone, and the bandlimited signal is ready for output. Specifications need to be met with regards to the speaker in a mobile unit, much the same as the transmit specifications on a mobile unit microphone, but those aspects directly dealing with the communications scheme have been dealt with.

7.1.2.1.5. Results

The success of the system can be seen in Figure 7-17 and Figure 7-18, comparing an input audio signal to the output signal, after encoding, combination with a wideband data stream, transmit and receive processing, and decoding. Figure 7-17 is an illustration of the frequency responses of a the input and output based on a signal made up of a set of tones. The matching of the frequency responses can be clearly seen. Figure 7-18 is a time domain representation of an audio signal sent through the system, and the output generated. This signal was in fact a considerably larger exercise, and the system was used for empirical observations of the audio quality of the output, which was quite successful.

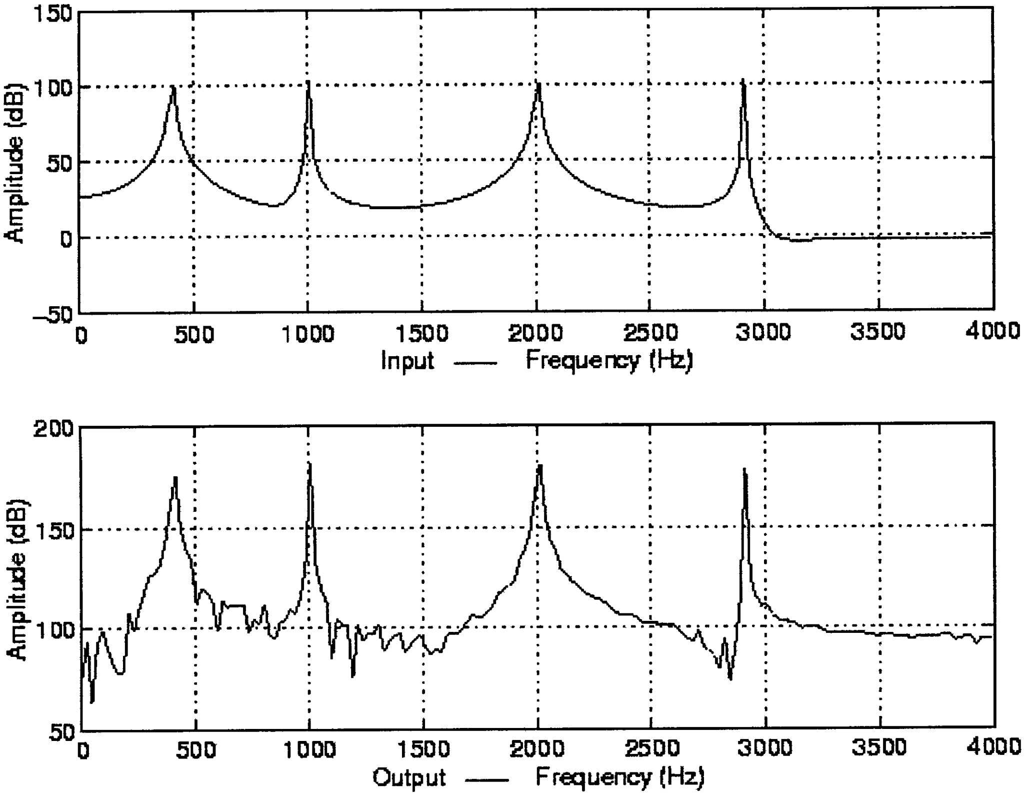


Figure 7-17: AMPS Voice Processing Results, Frequency Domain

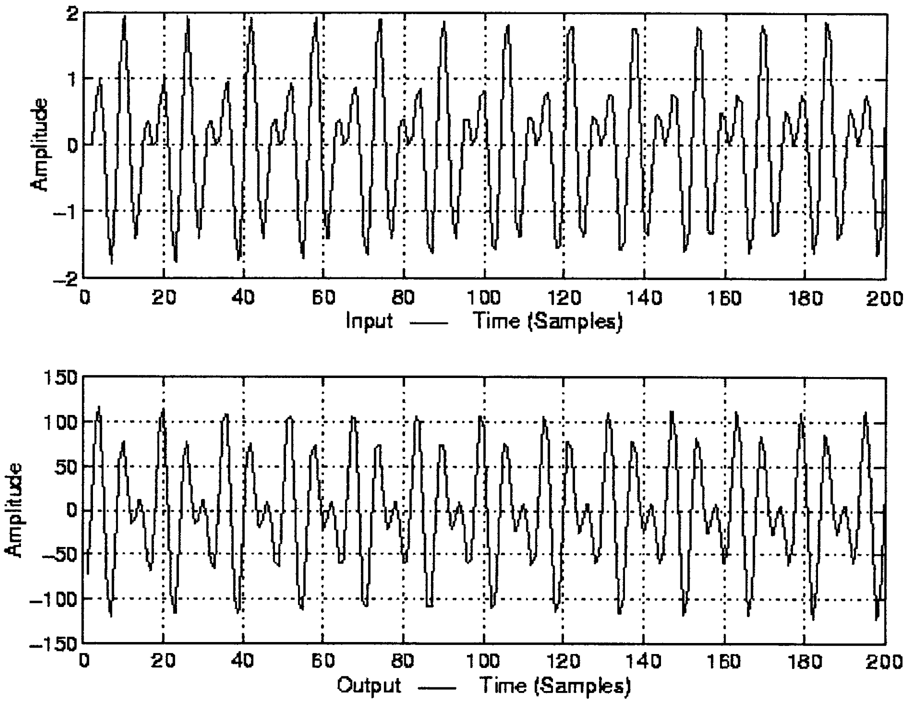


Figure 7-18: AMPS Voice Processing Results, Time Domain

7.1.2.2. Supervisory Audio Tone Recovery

7.1.2.2.1. Decimation to 20 kHz

The first task in SAT recovery is decimation from our 40 kHz sampling frequency to the 20 kHz frequency we had intended to use in the transmit stage for the generation of the SAT. By doing this, we facilitate future development intended for the mobile unit in terms of detecting a SAT tone, and then applying that same tone to its transmitted signal, as is necessary. The decimation to 20 kHz is accomplished using a set of cascaded sinc filters. A 3-tap sinc, a 2-tap sinc, and a 2-tap sinc^2 are applied to the unified signal prior to downsampling, each of small enough size to be easily implemented in C code. The frequency responses of these filters can be seen in Figure 7-19.

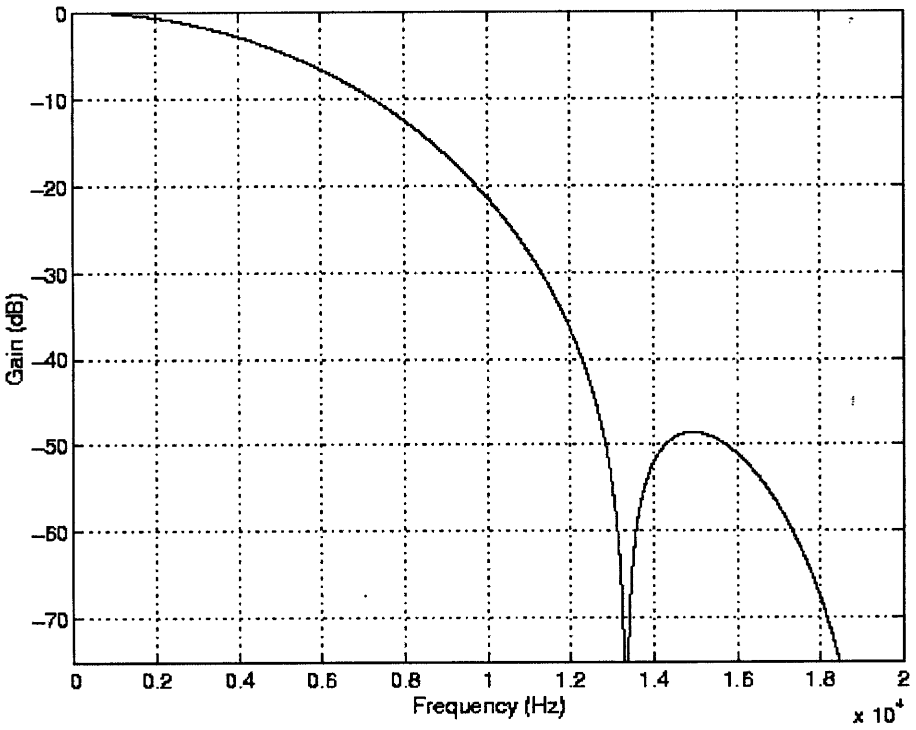


Figure 7-19: SAT Decimation to 20 kHz Filter

7.1.2.2.2. Bandpass Filter

The next crucial step is the isolation of the SAT tones at 5970, 6000 and 6030 Hz. This is accomplished through the use of a bandpass filter centered on 6 kHz with a passband of 120 Hz. This 4th order butterworth IIR was generated via the following Matlab command:

```
b = butter(4, [5.88 6.12]/10);
```

This bandpass filter creates an audio band rejection of at least 50 dB, and should leave us with a simple tone at the SAT of choice. Another benefit of this bandpass filter is the elimination of the DC offset created by the delta-sigma demodulation technique, leaving us with a zero centered signal. The frequency response of this filter can be seen in Figure 7-20.

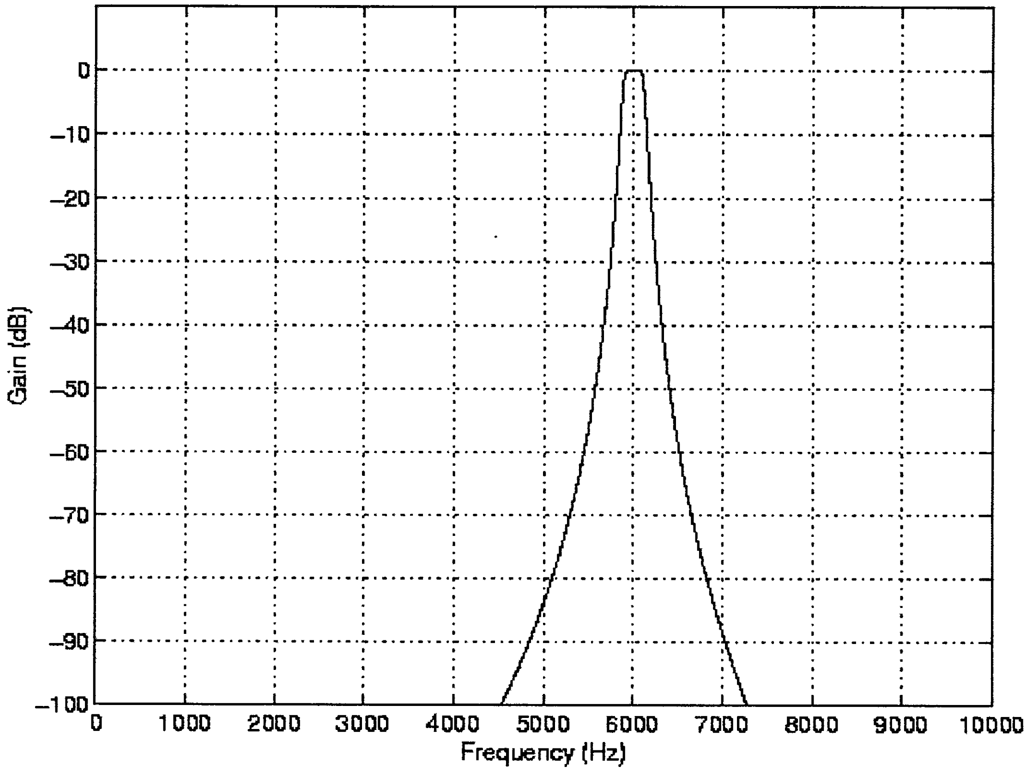


Figure 7-20: SAT RX Bandpass Filter

7.1.2.2.3. Detection

Finally, detection of the specific SAT in question is accomplished through a zero-crossing counting scheme. The SAT tone is limited by taking the sign bit of the SAT, as the bandpass filter has given us a signal with an average of zero. Zero crossings are then counted using a windowing technique, counting the number of zero crossings per 240 ms. This last was chosen in order to meet our specification of detecting the SAT at least every 250 ms. Finally, at the close of every window, the number of zero crossings detected is matched up against detection thresholds, and the appropriate SAT, or an invalid signal, is our final result. The SAT detection thresholds are listed in Table 7-2, and are correlated to the SAT detection specifications of Table 6-3.

Zero-Crossing Count	SAT Detection
N < 2857	SAT tone invalid
2857 ≤ N < 2873	5970
2873 ≤ N < 2888	6000
2888 ≤ N < 2904	6030
2904 ≤ N	SAT tone invalid

Table 7-2: SAT Detection Thresholds

7.1.2.2.4. Results

SAT recovery can be illustrated through an examination of the output file generated after SAT detection. The 240 ms window is clearly evident, as for that period of time the system remains in a SAT tone invalid state (SAT = 4), but after that, the SAT locks into place to match the 6000 Hz input tone (SAT = 2), and the system has proved successful. This is illustrated in Figure 7-21.

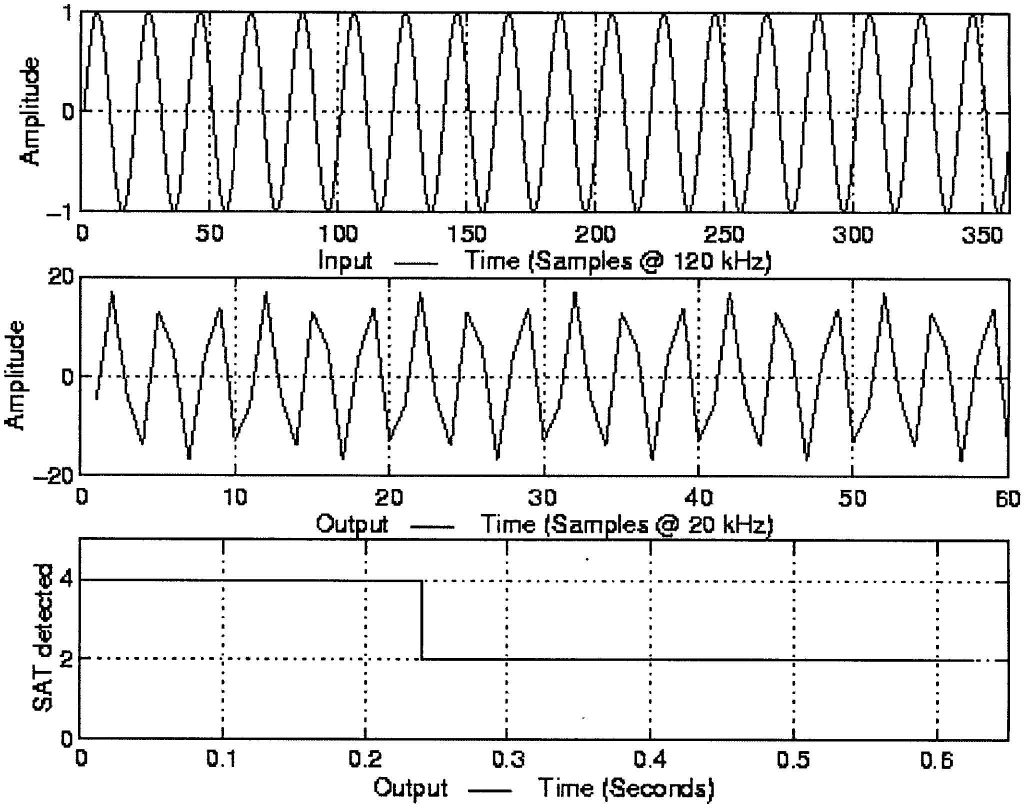


Figure 7-21: AMPS SAT Recovery Results

7.1.2.3. Wideband Data Recovery

Recovery of the wideband data is a somewhat tricky issue in terms of recovering timing information due to our Manchester encoding and detecting a signal that has been overlaid on top of an audio signal. The process involves an RX lowpass filter, interpolation to 320 kHz sampling frequency to gain appropriate time resolution for timing recovery, a highpass filter to attenuate audio band interference, and actual decoding of the data symbols from the interpolated signal. The block diagram for these steps can be seen in Figure 7-22.

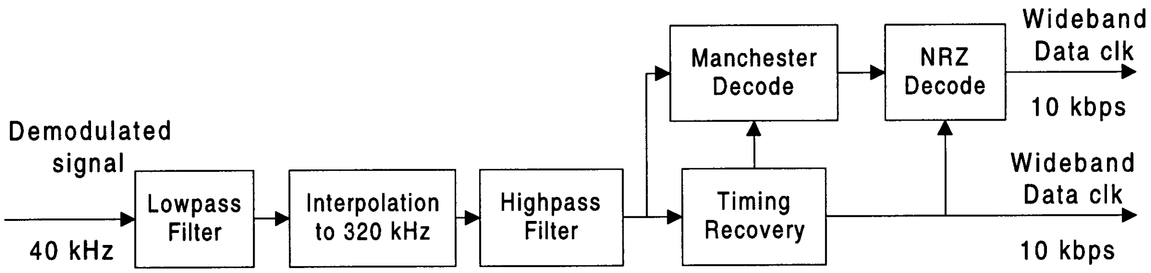


Figure 7-22: Wideband Data Decoder Block Diagram

7.1.2.3.1. Lowpass Filter

A lowpass filter begins the wideband data recovery process by bandlimiting the signal and eliminating frequencies above 13 kHz. This is accomplished through a simple 2-tap sinc filter with transfer function:

$$H(z) = \frac{\left(1+z^{-1}\right)}{2}$$

7.1.2.3.2. Interpolation to 320 kHz

Interpolation to 320 kHz is necessary to process at a fast enough speed to have the time resolution necessary for timing recovery and detection. This 8x interpolation is done using linear interpolation, a process whereby the 7 newly added points are a linear ramp between the current sample and the next. This can be accomplished through the use of a zero-order hold and an 8-tap sinc filter, but can be more easily implemented in code by simply specifying the linear ramp from one 40 kHz sample to the next. This will introduce a delay of one 40 kHz sample, as well.

7.1.2.3.3. Highpass Filter

Simulations indicated the need for an additional highpass filter for the purpose of attenuating the audio band and thereby facilitating concentration on the wideband data stream, which we'll recall, has a power spectrum shifted away from baseband and towards the 10 kHz of its symbol frequency. This was implemented as a 50 tap FIR, with cutoff at the 3 kHz upper end of the audio spectrum. There is no need to attenuate the SAT frequency of 6 kHz as the SAT is disabled during transmission of wideband data. The filter coefficients were found using Matlab as follows:

```
b = fir1(50, 3/160, 'high');
```

and the filter frequency response can be seen in Figure 7-23.

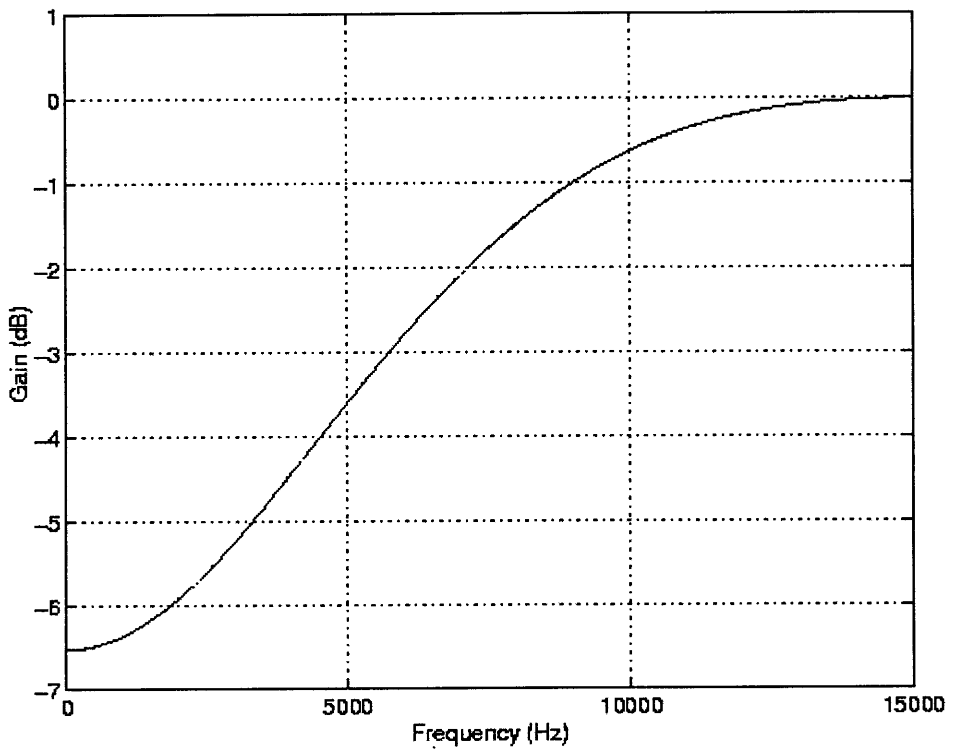


Figure 7-23: Wideband Data Detection Highpass Filter

7.1.2.3.4. Decoding

Now that we can be certain our signal is composed only of the wideband data stream, we can go about the process of decoding that stream. This involves several steps. The first is the limiting and zero crossing detection of our signal. The zero crossings will be the inputs to the next stage. That next stage is the recovery of timing of the

Manchester clock, which is accomplished through the use of a 1st order digital phase locked loop (DPLL). This will enable us to lock to the Manchester symbol rate of 20 ks/s, hopefully producing the zero-to-one sequences of NRZ logic ones and the one-to-zero sequences of NRZ logic zeros. Once timing has been determined, we can apply that to the limited signal and recover our Manchester encoded bit stream. Finally, the NRZ bit stream can be decoded from the Manchester bit stream simply by taking the second bit of every bit pair. Should our timing be off by one sample, we will have the logical inverse of our NRZ bit stream, but we will also be able to correct ourselves based on the property that in a Manchester stream, the XOR of any two bits that are part of one sequence will be one. The block diagram of this process can be seen in Figure 7-24.

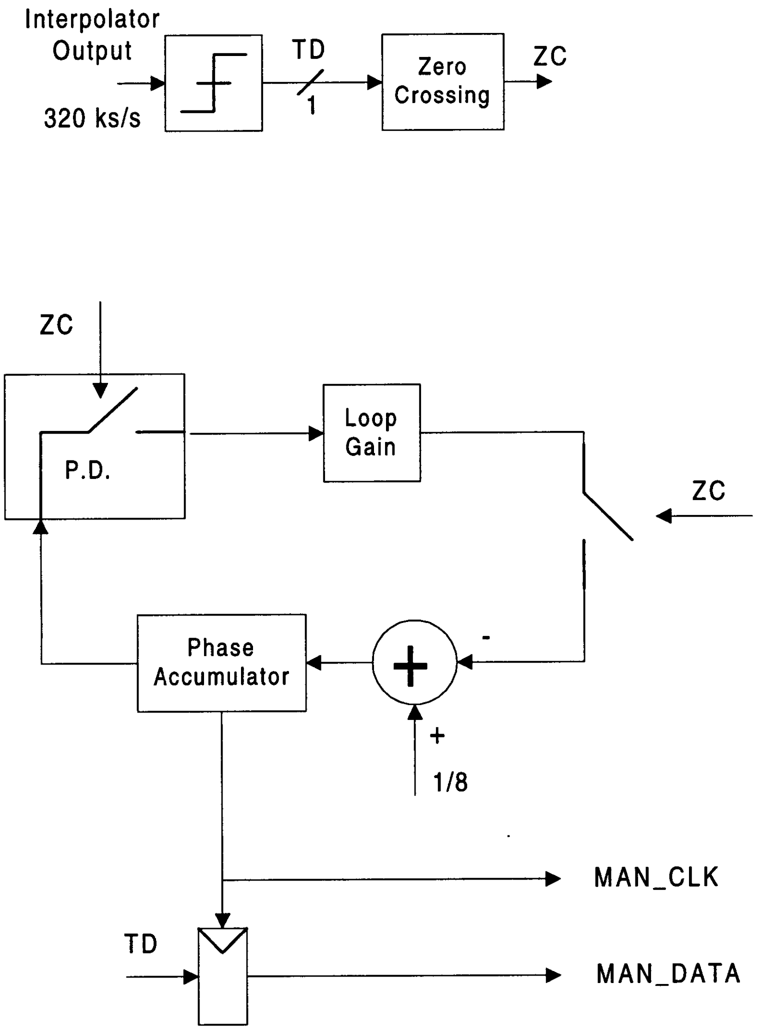


Figure 7-24: Manchester Decoding Block Diagram (DPLL)

The limiting process is a simple one, as our high pass filter will have removed any DC component, and will have left us with a zero centered signal. Finding zero crossings is also trivial. The loop as seen in Figure 7-25 is based on a phase accumulator which will ramp from -1 to 1 and then drop down to 1 again, with a free running phase increment of 1/8. This 16 step accumulation means the phase accumulator, when left to its own devices, will wrap once every 20 kHz Manchester symbol period. The zero crossings cause a factor to be added or subtracted to the phase accumulator, along with the standard 1/8, of size equivalent to a gain factor times the value of the phase accumulator at that point. The basic idea is for the zero crossings to occur when the phase accumulator is at zero, and therefore not affect the loop. Should the phase accumulator not be in phase with the input signal, however, the zero crossings will cause the phase accumulator to be adjusted by the amount by which it is off, and therefore recover the appropriate phase.

Timing is recovered by latching the limited signal on the wrap around of the phase accumulator. This will provide us with the Manchester encoded signal. This process can be viewed in Figure 7-25, wherein we see the phase accumulator's progress and corrections applied, and also see an instance of symbol repetition and the response of the phase accumulator to that.

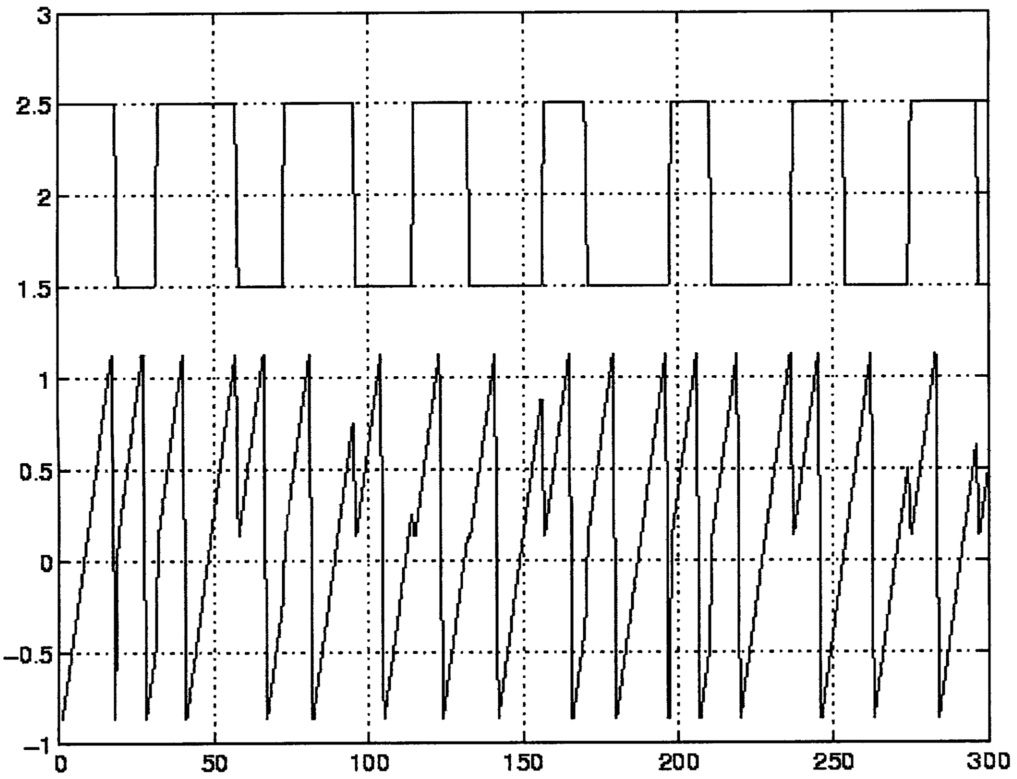


Figure 7-25: DPLL Functionality Plots: TD (top), Phase Accumulator (bottom)

Finally, the Manchester data stream can be converted into a NRZ stream as stated, by simply taking every other symbol. However, in order to ensure against incorrect timing, a constant testing process can be done, such that if at any point in time an unused symbol and the thought-to-be NRZ symbol following it are identical, we can delay our NRZ output process by one and thereby adjust our timing correctly.

7.1.2.3.5. Results

The successful recovery of the input signal can be seen in Figure 7-26. Figure 7-26 is an illustration of an inputted NRZ data stream, and the final output after encode, transmit, receive, and decode. The errors at the start as the filters fill their buffers and the DPLL achieves lock are evident, but the eventual success of the process is evident. The square wave nature of the data stream was chosen in order to facilitate matching timing of the input signal to the output signal.

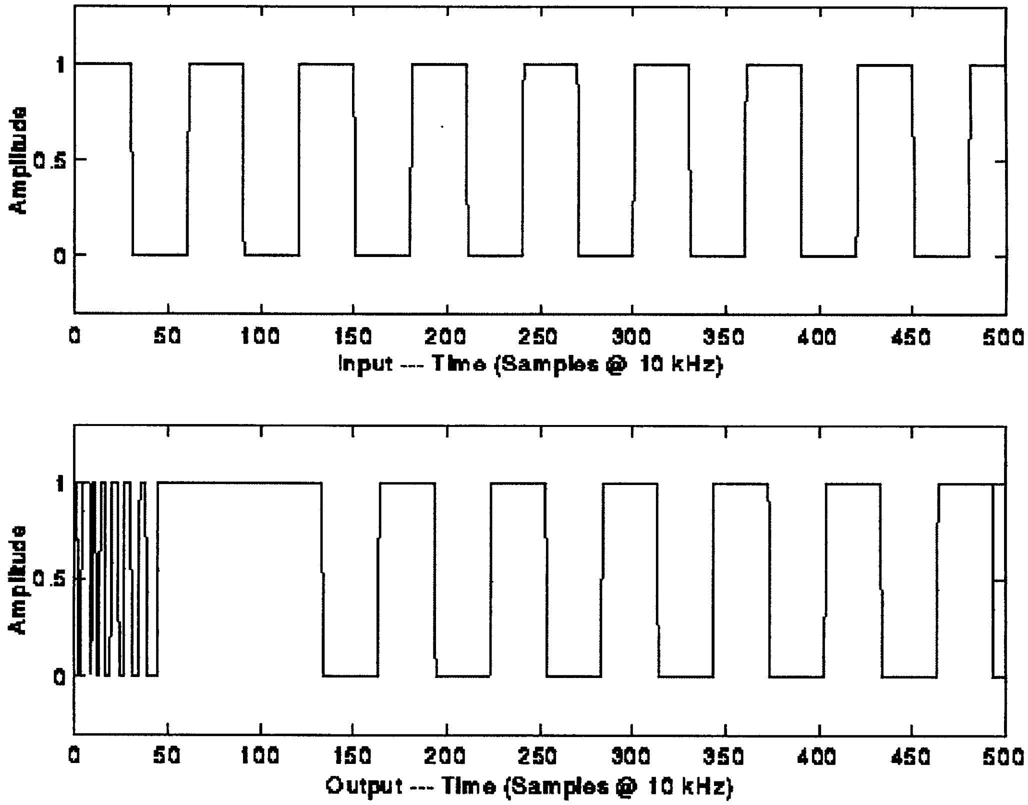


Figure 7-26: AMPS Wideband Data Results, Simple Input

7.2. JTACS Simulations

Beginning with our functional AMPS simulation, we shall now examine the adjustments necessary for appropriate JTACS functionality. As a means by which to proceed, we will focus on the changes necessary to implement, and endeavor to incorporate them into the existing system. We shall also examine the existing simulation code for its relevance to the new challenges which are posed in the JTACS realm.

Our comparative based discussion of the JTACS standard and its functionality in Section 7.2. will therefore aid us here by allowing us to focus on those new issues at hand. Once again, we shall step through each of the three elements involved in the signal, those being the audio signal, the supervisory signals, and the wideband data, and we will progress from transmit through receive.

7.2.1. Transmit Processing

7.2.1.1. Voice Processing

JTACS transmit voice processing is very similar to AMPS voice processing, as mentioned in our discussion of standards, with a few small exceptions. The first of these is the differing nominal peak carrier frequency deviation. This, however, is simply applied as the scaling factor in our code, immediately prior to combination of the signals. Therefore, that is an alteration that can be very easily made.

The second specified difference is the explicit requirement of a 24 dB/octave attenuation rate in the audio bandpass filter. However, in our design of a bandpass filter, we chose this as a useful benchmark, and have therefore already met that requirement.

The third, as mentioned, is the necessity for a different threshold on the deviation limiter, due to the smaller channel width. This is chosen in order to accommodate our maximum instantaneous frequency deviation of 9.5 kHz as follows:

$$Threshold = \frac{Maximum\,Frequency}{Audio\,Deviation} = \frac{9.5\,kHz}{2.3\,kHz} = 4.1304$$

Beyond those three, the additional factors in transmit voice processing are all implemented in the same way, and we are therefore all set in that respect.

7.2.1.2. Wideband Data Signals

Wideband data signals are encoded in much the same fashion in JTACS as in AMPS, however, once again, the nominal peak frequency deviation is now at 6.4 kHz rather than the 8 kHz of AMPS. This can once again be simply implemented.

The one major difference with regards to these signals that has a bearing upon our implementation of the system is the change in wideband NRZ data rate from 10 kbps to 8 kbps. As a result of this, our curve matching must be altered, as we are no longer adjusting a single symbol to a curve at 12 times its sampling rate, but to a curve 15 times its sampling rate. This requires an entirely new wideband data waveform generation function, as seen in Table 7-3, with 15 points scattered along a sinusoidal curve instead of the 12 points from the AMPS waveform generator. A plot of the waveform generation can be seen in Figure 7-27. Once the appropriate curve has been matched, however, we once again have a 120 kHz sampling rate curve that we can proceed with, applying our new deviation to.

X(n-1)		**Wide Band Data Waveform**														
	X(n)	Second half of X(n-1) Waveform							First Half of X(n) Waveform							
0	0	0	-.41	-.74	-.95	-1	-.87	-.59	-.21	.21	.59	.87	1	.95	.74	.41
0	1	0	-.41	-.74	-.95	-1	-1	-1	-1	-1	-1	-1	-1	-.95	-.74	-.41
1	0	0	.41	.74	.95	1	1	1	1	1	1	1	1	.95	.74	.41
1	1	0	.41	.74	.95	1	.87	.59	.21	-.21	-.59	-.87	-1	-.95	-.74	-.41

Table 7-3: Wideband Data Waveform Generator Function

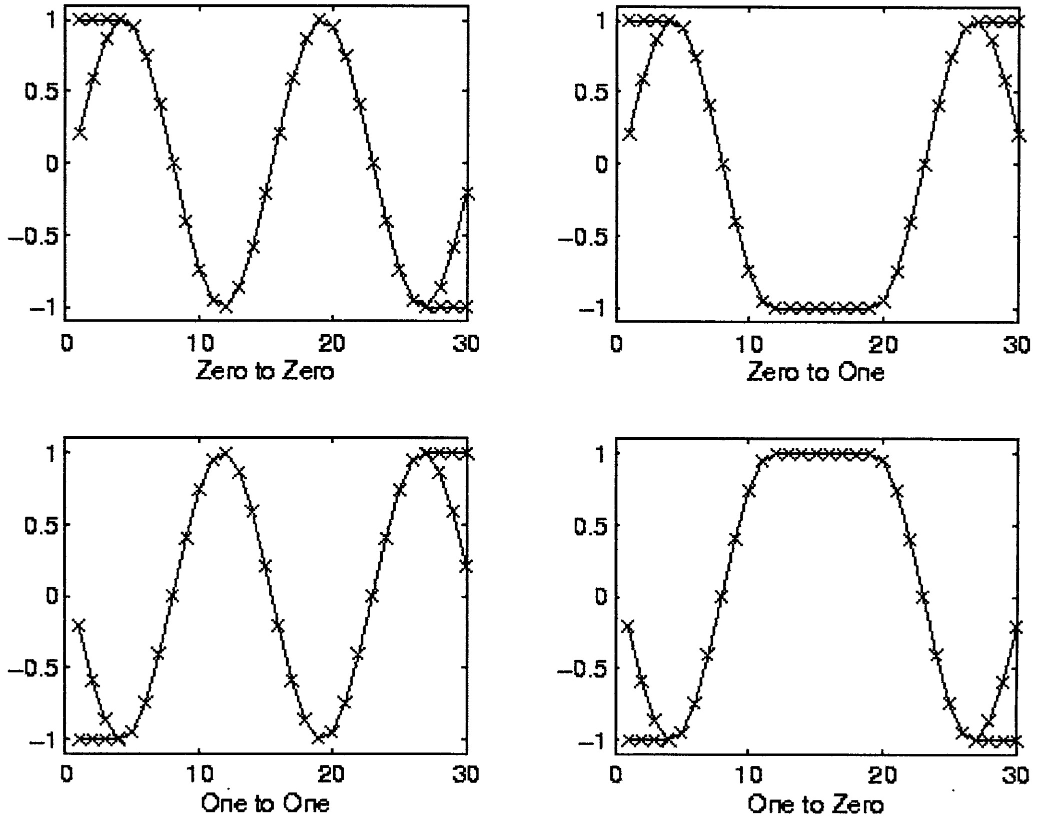


Figure 7-27: Wideband Data Waveform Generator Plot

7.2.1.3. Supervisory Audio Tone Signal Characteristics

As indicated in the discussion on the standards themselves, the only difference between the AMPS SAT and the JTACS SAT is once again the nominal peak frequency deviation. The JTACS deviation is 1.7 kHz, and that will therefore be the scaling factor applied to the SAT upon combination.

7.2.2. Receive Processing

7.2.2.1. Voice Processing

Recovery of the audio signal is accomplished in the same way as AMPS receive voice processing. As the only differences applied to the transmit were in the deviation applied to the signal, we can expect a slightly different response, however, we remain well within margins. This can be seen in Figure 7-28 and Figure 7-29, which illustrate the frequency and time domain responses audio outputs based upon two separate inputs, which are also shown. These examinations are the same ones which were applied to the AMPS system, and can therefore be seen to be as effective in demodulating those signals as the AMPS processing was.

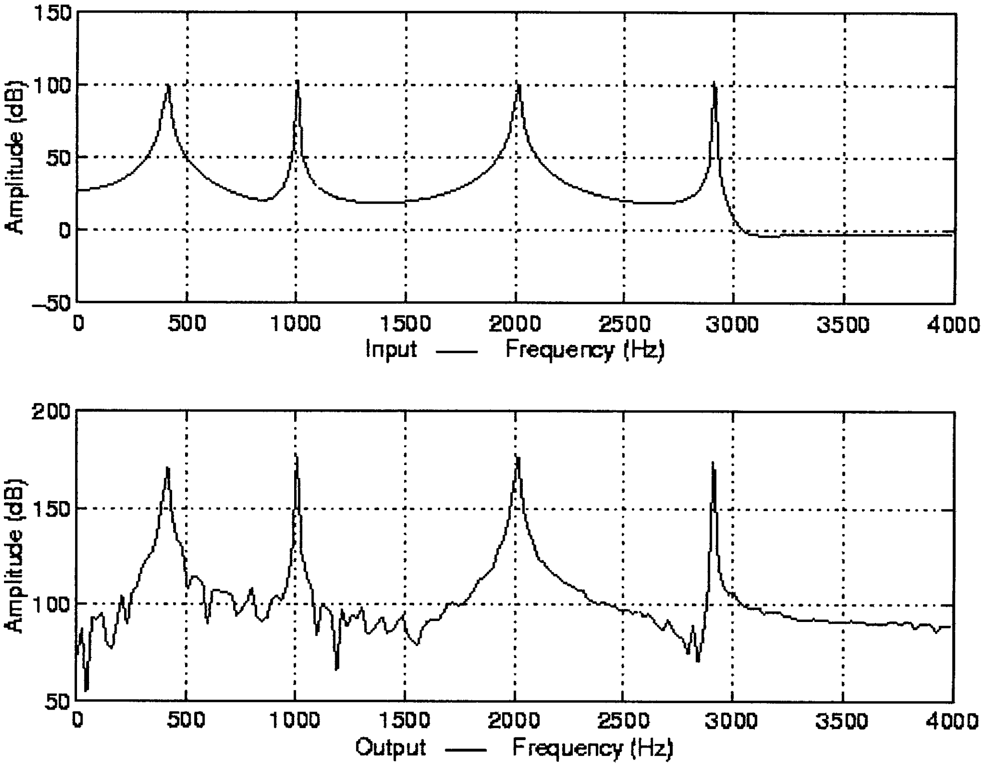


Figure 7-28: JTACS Voice Processing Results, Frequency Domain

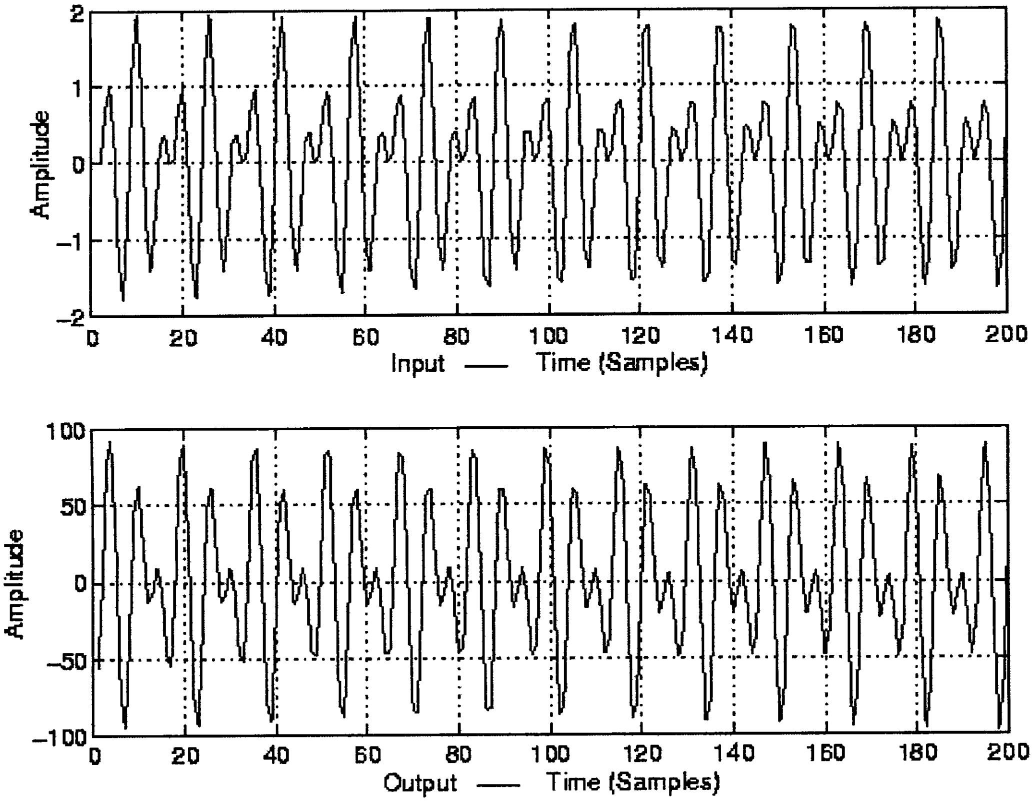


Figure 7-29: JTACS Voice Processing Results, Time Domain

7.2.2.2. Supervisory Audio Tone Recovery

SAT recovery is also done in the same way, and can be verified exactly as AMPS SAT recovery was verified. Figure 7-30 illustrates this, using the same inputs as used in Section 7.1.2.2.4.

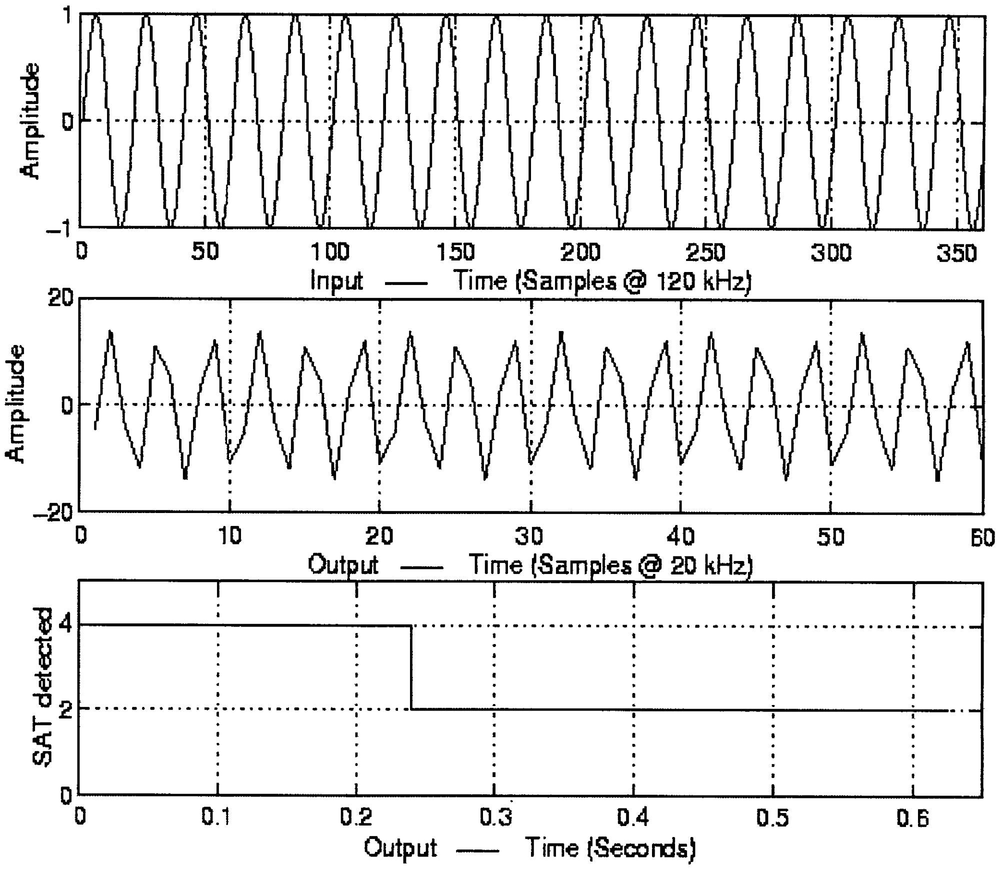


Figure 7-30: JTACS SAT Recovery Results

7.2.2.3. Wideband Data Recovery

Wideband Data Recovery was also done without alterations. An illustration of the error signal of a wideband data stream can be seen in Figure 7-31.

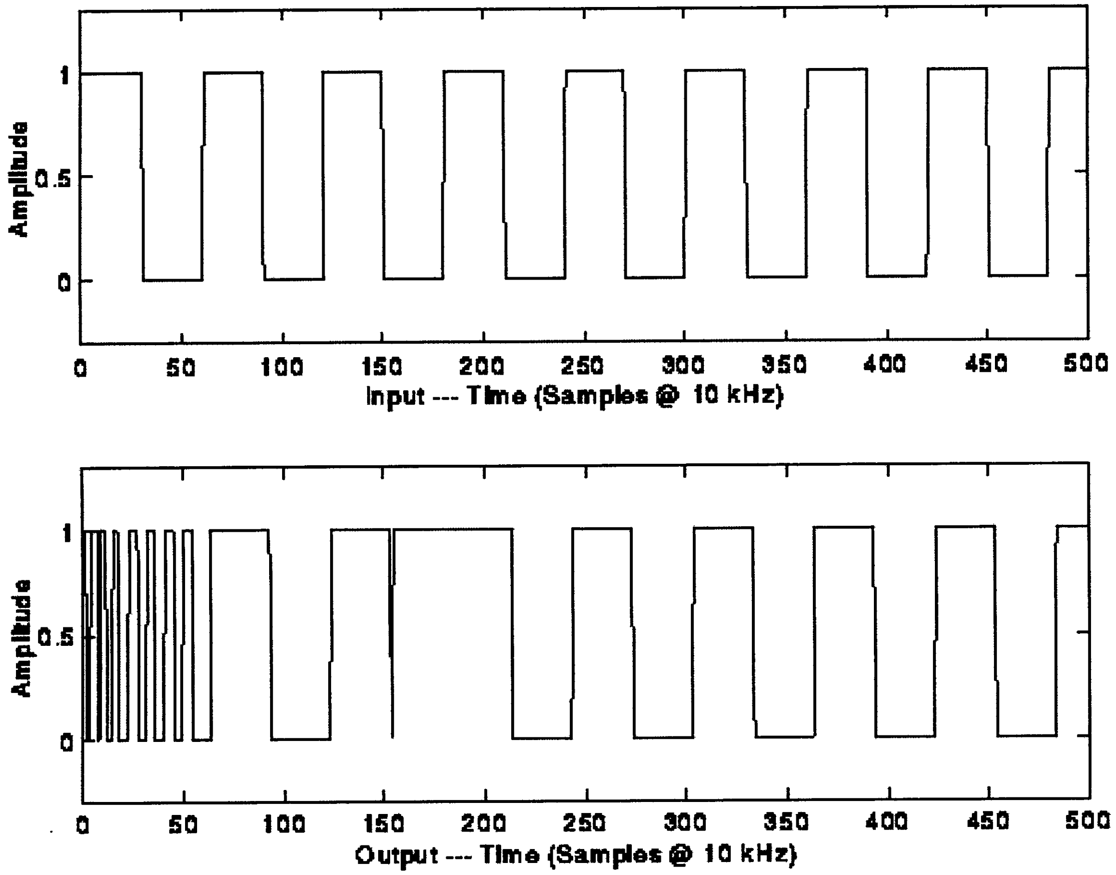


Figure 7-31: JTACS Wideband Data Recovery Results

7.3. NTACS Simulations

Now that we have proven functional AMPS and JTACS systems, we can take them and incorporate those additional changes necessary for the sake of NTACS processing. These changes are once again heavily based upon changing the deviations of the various signals and ensuring we can still recover the appropriate signals, but there is also the major added aspect of the DSAT and DST signal to recover.

7.3.1. Voice Processing

NTACS voice processing is identical to JTACS voice processing with the exception of a smaller nominal peak frequency deviation of 1.5 kHz, and a smaller peak instantaneous frequency deviation of 5 kHz. The former influences the scaling factor applied for the combination of signals, and the latter adjusts the threshold of the deviation limiter as follows:

$$Threshold = \frac{Maximum\,Frequency}{Audio\,Deviation} = \frac{5\,kHz}{1.5\,kHz} = 3.3333$$

The receive path is entirely identical to JTACS audio receive processing, and produced the results in Figure 7-32 and Figure 7-33, which indicate that the system remains functional for NTACS. These are, once again, based upon the same inputs and outputs used for our AMPS and JTACS audio results.

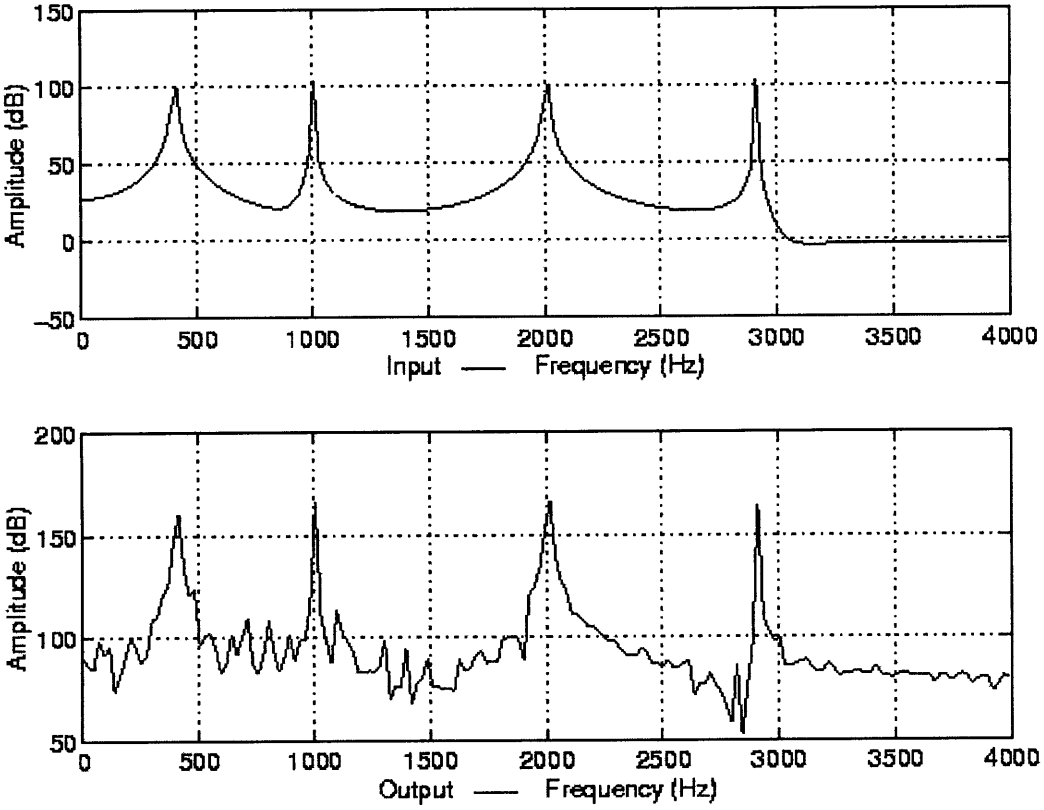


Figure 7-32: NTACS Voice Processing Results, Frequency Domain

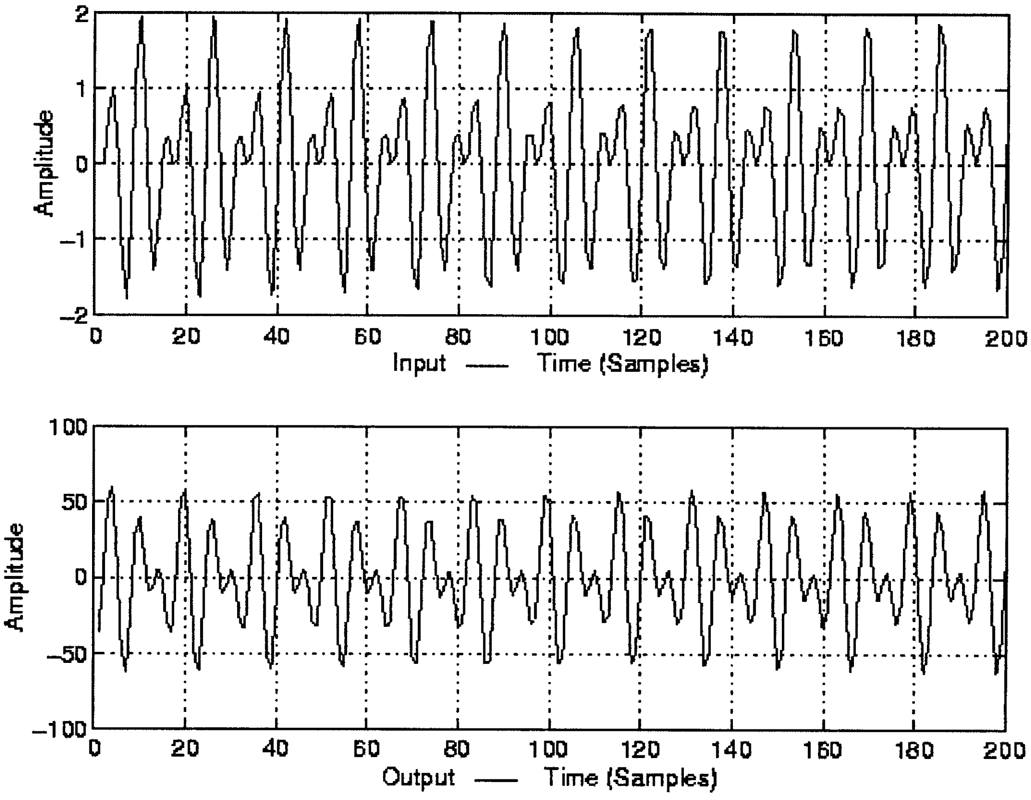


Figure 7-33: NTACS Voice Processing Results, Time Domain

7.3.2. Supervisory Signals

7.3.2.1. DSAT/DST Transmission

The sub-audio DSAT/DST generation described in section 6.2.2.4. is dealt with prior to incorporation into this simulation. This is done using a C program designed specifically to translate the choice of the 14 different signals (7 DSAT, 7 DST) into the relevant 24 bit length bit streams. This file, satntacsgen.c is attached as Appendix 2.7.

The encoding is done by reading in a 200 bps input file, specifying which of the sequences is desired. However, given that the choice can only change on positive bits of the mask, if the current mask bit is not set, the new input is ignored, and the prior input choice is maintained. Whichever sequence is selected, that sequence is passed on for the output, bit by bit. However, this 200 bps sequence is to be passed to the simulation code at the expected SAT input sampling frequency of 120 kHz, and is thus repeated 600 times in order to match the appropriate sampling frequency. This sequence is then incorporated into the simulation and used to modulate the carrier with a nominal peak frequency deviation of 700 Hz.

Once again, it is worth noting that the DSAT/DST is muted during transmission of wideband data.

7.3.3. Wideband Data Signals

NTACS wideband data is also dealt with in the same way as JTACS wideband data, however, with a considerably smaller nominal peak frequency deviation of 0.7 kHz on a voice channel. We will consider only the voice channels at this point, as the more challenging of the two possibilities, as opposed to the control channel's 6.4 kHz deviation, the same as JTACS, but without audio to affect our quality. As with JTACS, the input data stream is at a sampling frequency of 8 kHz, and therefore uses the JTACS created 15 point curve.

Chapter 8. Conclusion

8.1. Overview

Through the simulations performed in this work, it seems clear that this new digital FM demodulator has succeeded in achieving its goals. The performance, both as an independent demodulator, and as a complete multi-standard system have been examined and to all analysis appears to be a successful means of performing this demodulation. The primary goals set forth were achieved, in implementing a delta sigma based demodulation scheme that successfully avoids the problems inherent in analog I/Q demodulation, such as phase and gain mismatch between the two paths, and transfers most of the selectivity processing to the digital domain, allowing for flexibility of programming, elimination of costly and difficult analog filtering, and improved testability and yield. The challenges of adhering to multiple standards have been faced, and the specifics of dealing with those differing standards are easily approachable from a digital domain perspective, where that would not be the case using analog blocks and processes.

8.2. Future Improvements

There remain, however, a number of realms in which future improvements and work can certainly be done. The first and foremost of these is clearly the logical following through of the system into more specific implementations in hardware and in actual experimental situations. The continuation of this work into a hardware specific design, a hardware description language, and actual chip design and testing would illustrate the effectiveness of this approach directly.

Various demodulation schemes were examined throughout this work, of which the one chosen for simulation work was seen to be the cheapest while falling under our required specifications. As such, it seems like a successful and proper approach. However, here as well, further advancements and techniques in delta sigma modulation can be examined for their applicability and use in this situation. There are certainly other useful designs available, and, in exploring those, we may be able to determine further benefits and advancements in a demodulation scheme.

Finally, this applicability to these standards is a gain and benefit, but the basic concept, of expanding into multiple FM environments, can be expanded into other areas as well. Systems such as the Canadian Aurora system, the Nordic NMT network, or other European NMT networks can be looked into, as necessary, and this multi-standard approach applied to these[17]. In addition, some the general concepts and approaches used in this research can apply to the newer digital systems put into place around the globe. For the purposes of this discussion, however, the key element is the success of this specific system, as seen through these simulation, and the performance within our desired goals and specifications.

References

1. Høvin, M., Lande, T.S., Wisland, D.T., and Kiaei, S.: "Triangularly weighted zero-crossing detector providing ΔΣ frequency-to-digital conversion", *IEE Electronics Letters*, 1997, vol. 33, no. 13.
2. Candy, J.C . and Temes, G.C.: *Oversampling delta-sigma data converters*, IEEE Press, New York, 1992.
3. Riley, T.A.D., Copeland, M.A. and Kwasniewski, T.A.: "Delta-sigma modulation in fractional-N frequency synthesis", *IEEE J. Solid State Circuits*, vol. 28, no. 5, pp. 553-559, May 1993.
4. Beards, R.D. and Copeland, M.A.: "An oversampling delta-sigma frequency discriminator", *IEEE Trans. on Circuits and Systems*-II, vol. 41, no. 1, pp. 26-32, Jan. 1994.
5. Galton, I.: "Analog-input digital phase-locked loops for precise frequency and phase demodulation", *IEEE Trans. on Circuits and Systems*-II, vol. 42, no. 10, pp.26-32, Oct. 1995.
6. Høvin, M., Olsen, A., Lande, T.S., and Toumazou, C.: "Novel second-order Δ–Σ modulator/frequency-to-digital converter", *IEE Electronics Letters*, vol. 31, no. 2, pp.81-82, 1995.
7. Høvin, M., Olsen, A. Lande, T.S. and Toumazou, C.: "Delta-sigma modulators using frequency modulated intermediate values", IEEE J. Solid State Circuits, vol. 32, no. 1, Jan. 1997.
8. Chu, S. And Burrus, C.S.: "Multirate filter designs using comb filters", IEEE Trans. on Circuits and Systems, vol. CAS-31, pp. 913-924, Nov. 1984.
9. Hogenauer, E. B.: "An economical class of digital filters for decimation and interpolation", IEEE Trans. Acoust. Speech, Signal Processing, vol. ASSP-29, pp. 155-162, Apr. 1981.
10. Rohde, U.L., Whitaker, J., and Boucher, T.T.N.: *Communications Receivers*, McGraw-Hill, New York, 1997.
11. Oppenheim, A.V. and Schafer, R.W.: *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1989.
12. Oppenheim, A.V. and Willsky, A.S.: *Signals and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1983.
13. Freeman, R.L: *Telecommunication System Engineering*, John Wiley & Sons, New York, 1989.
14. Proakis, John G., and Salehi, Masoud: *Communications System Engineering*, Prentice Hall, Upper Saddle River, NJ, 1994.
15. Inose, H., and Yasuda, Y: "A unity feedback coding method by negative feedback," *Proceedings of the IEEE*, vol. 51, pp. 1524, 1535, Nov. 1963.
16. Hein, S. and Zakhor, A.: *Sigma Delta Modulators : nonlinear decoding algorithms and stability analysis*, Kluwer Academic Publishers, Norwell, MA, 1993.
17. Lee, W. C. Y.: *Mobile Cellular Telecommunications Systems*, McGraw-Hill Book Company, New York, 1989.

18. Tsui, J.: *Digital Techniques for Wideband Receivers*, Artech House, Boston, MA, 1995.
19. Horowitz, P. and Hill, W.: *The Art of Electronics*, Cambridge University Press, New York, NY, 1989.
20. Young, W.R.: "Advanced Mobile Phone Service: Introduction, Background, and Objectives," Bell Systems Technical Journal, vol. 58, no. 1, Jan 1979.
21. Arredondo, G.A., Feggeler, J.C., and Smith, J.I.: "Voice and Data Transmission," Bell Systems Technical Journal, vol. 58, no. 1, Jan 1979.
22. Panter, P.F.: *Modulation, Noise, and Spectral Analysis*, McGraw-Hill Book Company, New York, 1965.
23. EIA Interim Standard, "Recommended Minimum Standards for 800-MHz Cellular Subscriber Units", EIA/IS-19-B, May 1988.
24. EIA/TIA Standard, "Mobile Station - Land Station Compatibility Specification", EIA/TIA-553, September 1989.
25. TIA/EIA Interim Standard, "Recommended Minimum Performance Standards for Dual-Mode Wideband Spread Spectrum Cellular Mobile Stations", TIA/EIA/IS-98-A, July 1996.
26. TIA/EIA Interim Standard, "Mobile Station-Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular System", TIA/EIA/IS-95-A, May 1995.
27. I-3, "Total Access Communications System (TACS) For Japan Mobile Station Compatibility Specification.", RCR STD-36, June 1993.

Appendix 1. Triangularly Weighting Technique Simulation Code

Appendix 1.1. Hovin8.c

```
/* hovin8.c
   written by Grant Smith
   9/30/97

*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define INPUTFILE inputfilename
#define OUTPUTFILE outputfilename
#define OUTPUTINFO outputinfoname

#define OUTPUT1 "testhovin8_1.dat"
#define OUTPUT2 "testhovin8_2.dat"
#define OUTPUT3 "testhovin8_3.dat"
#define OUTPUT4 "testhovin8_4.dat"
#define OUTPUT5 "testhovin8_5.dat"

#define FM_IF                (fmif)   /* fc */

#define IF_SAMPLING_FREQUENCY   (ifsamplingfreq)  /* fs */

#define BASEBAND_SAMPLING        (basebandsampling)   /* fd
*/

#define    DECIMATION_FACTOR    (IF_SAMPLING_FREQUENCY    /
BASEBAND_SAMPLING)


#define TONE_1_FREQUENCY tone1freq     /* fm */
#define TONE_1_DEVIATION tone1dev    /* fm dev */

#define TONE_2_FREQUENCY tone2freq
#define TONE_2_DEVIATION tone2dev
```

Appendix 1. Triangularly Weighting Technique Simulation Code

```
#define TOPVALUE 65536

#define TOTAL_OUTPUT_POINTS outputpoints

#define MASK 0xffff

#define buffersize 10000

void main(int argc, char *argv[])
{
  char filename[30];
  char inputfilename[30];
  char outputfilename[30];
  char outputinfoname[30];

  float fmif;
  float ifsamplingfreq;
  float basebandsampling;
  float outputpoints;
  float tone1freq;
  float tone1dev;
  float tone2freq = 0.0;
  float tone2dev = 0.0;

  FILE *inputFile;
  FILE *outputFile;
  FILE *outputInfo;
  FILE *debug1, *debug2, *debug3, *debug4, *debug5;

  unsigned mainIndex;
  unsigned pointsCount;

  int bufferIndex=0;
  unsigned buffer[buffersize];

  unsigned signal;
  unsigned signalold;
  unsigned count;
  unsigned countold;
  unsigned acc;
  unsigned accold;
  unsigned dec;
  unsigned decold;
  unsigned sub1;
  unsigned sub1old;
  unsigned out;

  double   value;
  double   theta_1_Increment;
  double   theta_2_Increment;
  double   phi = 0.0;

  int      IFLevel = 0;


  /**** command line arguments correct? ****/

  if (argc != 2) {
    printf("Error:  Format:  hovin8 <filename>\n"
         "              where filename contains the input
variables\n"
         "        and filename.dat and filename.info contain
output\n");
    exit(2);
  }

  strcpy(filename,argv[1]);
  strcpy(inputfilename, filename);
  strcpy(outputfilename, strcat(filename, ".dat"));
  strcpy(filename,argv[1]);  /* corrupted by strcat, reload
*/
  strcpy(outputinfoname, strcat(filename, ".info"));


  /**** initializations  ***/

  /* open debugging files */

  if ((inputFile = fopen(INPUTFILE,"r")) == NULL) {
    printf("Could not open file %s for input.\n",INPUTFILE);
    exit(1);
  }

  if ((outputFile = fopen(OUTPUTFILE,"w")) == NULL) {
    printf("Could not open file %s for
output.\n",OUTPUTFILE);
    exit(1);
  }

  if ((outputInfo = fopen(OUTPUTINFO,"w")) == NULL) {
    printf("Could not open file %s for output
info.\n",OUTPUTINFO);
    exit(1);
  }

/*
  if ((debug1 = fopen(OUTPUT1,"w")) == NULL) {
    printf("Could not open file %s for output.\n",OUTPUT1);
    exit(1);
  }
  if ((debug2 = fopen(OUTPUT2,"w")) == NULL) {
    printf("Could not open file %s for output.\n",OUTPUT2);
    exit(1);
  }
  if ((debug3 = fopen(OUTPUT3,"w")) == NULL) {
    printf("Could not open file %s for output.\n",OUTPUT3);
    exit(1);
  }
  if ((debug4 = fopen(OUTPUT4,"w")) == NULL) {
    printf("Could not open file %s for output.\n",OUTPUT4);
    exit(1);
  }
  if ((debug5 = fopen(OUTPUT5,"w")) == NULL) {
    printf("Could not open file %s for output.\n",OUTPUT5);
    exit(1);
  }
*/

  fscanf(inputFile, "%e\n", &ifsamplingfreq);
  fscanf(inputFile, "%e\n", &basebandsampling);
  fscanf(inputFile, "%e\n", &fmif);
  fscanf(inputFile, "%e\n", &outputpoints);
  fscanf(inputFile, "%e\n", &tone1freq);
  fscanf(inputFile, "%e\n", &tone1dev);

  printf("\nRunning   hovin   algorithm   on   the   following
data:\n"
      "ifsamplingfreq = %2.0f\n"
      "basebandsampling freq = %2.0f\n"
      "fm if frequency = %2.0f\n"
      "output points = %2.0f\n"
      "tone1freq = %2.0f\n"
      "tone1deviation = %2.0f\n",
      ifsamplingfreq, basebandsampling,
      fmif, outputpoints, tone1freq, tone1dev);

  fprintf(outputInfo,"%e \n%e \n%e \n%e \n%e \n%e \n%e \n%e
\n",
       IF_SAMPLING_FREQUENCY,
       DECIMATION_FACTOR,
       FM_IF,
       (float) TOTAL_OUTPUT_POINTS,
          TONE_1_FREQUENCY,
       TONE_1_DEVIATION,
          TONE_2_FREQUENCY,
       TONE_2_DEVIATION);

  pointsCount        =        (unsigned)        (((double)
TOTAL_OUTPUT_POINTS)*DECIMATION_FACTOR);
```

```
  /* to create IF */
  theta_1_Increment  =  2.0  *  M_PI  *  TONE_1_FREQUENCY  /
IF_SAMPLING_FREQUENCY;
  theta_2_Increment  =  2.0  *  M_PI  *  TONE_2_FREQUENCY  /
IF_SAMPLING_FREQUENCY;

  signal = 1;
  count = 0;
  acc = 0;
  dec = 0;
  sub1 = 0;
  out = 0;


  /***** MAIN LOOP  *****/

  for (mainIndex=0; mainIndex < pointsCount; mainIndex++) {

    /****  Create Limited IF FM signal from tone1, tone2.
****/

    value    =    TONE_1_DEVIATION    *    cos(mainIndex    *
theta_1_Increment);
/*          value   +=  TONE_2_DEVIATION  *  cos(mainIndex  *
theta_2_Increment);
*/

    phi += 360.0 * (value + FM_IF) / IF_SAMPLING_FREQUENCY;

    if (phi > 180.0) {
      phi = phi - 180.0;
      IFLevel = 1 - IFLevel;
    }

    signalold = signal;
    signal = 1 - (IFLevel);  /* remove DC bias */

/*  fprintf(debug1, "%u\n", signal); */


    /**** Counter  ****/
    countold = count;
    if((signal - signalold) >0)
      count = countold + 1;
    else
      count = countold;
    count = count & MASK;  /* mod-x */
/*  fprintf(debug2, "%u\n", count); */
```

```
    /****  Accumulate ****/
    accold=acc;
    acc = count + accold;
    acc = acc & MASK;
/*  fprintf(debug3, "%u\n", acc); */


    /**** decimate  ****/
    if (( mainIndex % ((unsigned) DECIMATION_FACTOR)) == 0)
{
      if(((mainIndex * 10) % pointsCount) == 0)
     printf(".\n");  /* show progress */

      decold = dec;
      dec = acc;
/*    fprintf(debug4, "%u\n", dec); */


      /****  subtract 1st ****/
      sub1old=sub1;
      sub1 = dec-decold;
      sub1 = sub1 & MASK;
/*    fprintf(debug5, "%u\n", sub1); */


      /****  subtract 2nd ****/
      out = sub1-sub1old;
      out = out & MASK;
/*    fprintf(outputFile, "%u\n", out); */
      buffer[bufferIndex++]=out;


      /**** dump buffer  ****/
      if(bufferIndex==buffersize) {
     if((fwrite(buffer,    sizeof(unsigned),    buffersize,
outputFile) == 0)) {
       printf("error while writing output\n");
       exit(2);
     }
     bufferIndex=0;
      }
    }
  }

  if((fwrite(buffer,   sizeof(unsigned),   (bufferIndex-1),
outputFile) == 0)) {
    printf("error while writing output\n");
    exit(2);
  }
  fclose(outputFile);
  fclose(outputInfo);
/*
  fclose(debug1);
  fclose(debug2);
  fclose(debug3);
  fclose(debug4);
  fclose(debug5);
*/

  printf("Done\n");

}
```

Appendix 1.2. Testhovin8.m

```
clear all;


% input data

load test1.info;
info=test1;

originalSamplingFrequency=info(1);
decimationFactor=info(2);
fmFrequency=info(3);
outputPoints=info(4);
tone1Frequency=info(5);
tone1Deviation=info(6);
tone2Frequency=info(7);
tone2Deviation=info(8);

clear info;


inputFile=fopen('test1.dat', 'r');
f=fread(inputFile, inf, 'uint');

fclose(inputFile);


%prepare data


f=f(10:length(f));
f2=f-(sum(f)/length(f));

samplingFrequency                                       =
originalSamplingFrequency/decimationFactor;

fftLength= 2 ^ floor(log2(outputPoints));


%processing

pxx=psd(f2, fftLength, samplingFrequency);

freq=linspace(0,(samplingFrequency/2.0), length(pxx));
```

```
binIncrement = (samplingFrequency/2.0) / (length(pxx) -
1.0);

startPointer=floor(300.0/binIncrement) +1;
endPointer = ceil(3000.0 / binIncrement) +1;

tone1Pointer = round(tone1Frequency / binIncrement) +1;
tone2Pointer = round(tone2Frequency / binIncrement) +1;

toneWidth = floor(outputPoints/50000 +4);

noise = sum(pxx(startPointer:(tone1Pointer-(toneWidth +
1))));
noise = noise + sum(pxx((tone1Pointer +
(toneWidth+1)):endPointer));

%noise = noise + sum(pxx((tone1Pointer +
(toneWidth+1)):(tone2Pointer-(toneWidth+1))));
%noise = noise + sum(pxx((tone2Pointer +
(toneWidth+1)):endPointer));



signal = sum(pxx((tone1Pointer -
toneWidth):(tone1Pointer+toneWidth)));
%signal = signal + sum(pxx((tone2Pointer -
toneWidth):(tone2Pointer + toneWidth)));
dBnoise = 10*log10(noise/signal);

dbns=10 * log10(pxx);

ndbns=dbns - max(dbns);



%display

plot(freq,ndbns);
xlabel('Frequency');
ylabel('Signal power');
line = sprintf('Integrated noise = %3.3f', dBnoise);
text(2.5e+3, -20, line);

% title([num2str(outputPoints), ' points, ',
num2str(fftLength), ' FFT, D=' , num2str(decimationFactor),
', tone1 ' , num2str(tone1Frequency), ', ' ,
num2str(tone1Deviation), ' dev, tone2 ' ,
num2str(tone2Frequency), ', ' , num2str(tone2Deviation), '
dev, fs=' , num2str(originalSamplingFrequency) , ', fc=' ,
num2str(fmFrequency)]);
```

```
title([num2str(outputPoints),       '       points,       ',
num2str(fftLength), ' FFT, D=' , num2str(decimationFactor),
',   tone1   '   ,   num2str(tone1Frequency),   ',   '   ,
num2str(tone1Deviation),      '      dev,      fs='      ,
num2str(originalSamplingFrequency)    ,    ',    fc='    ,
num2str(fmFrequency)]);
```

Appendix 1.3. Test

```
19.680e+6
40.0e+3
492000.0
100000
1003.0
2.9e+3
```

Appendix 2. Multi-Standard Digital FM Receiver Simulation Code

Appendix 2.1. Main.c

```
/* main.c
   written by Grant Smith
   started:  11/17/97

*/


#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "defines.c"
#include "basetx.c"
#include "mobilerx.c"


void main()
{

  /****  preparation of demodulation signal ****/
  if(!(TX_DIS))
    if(basetx(INPUTFILE, TXFILE)) {
      printf("transmit processing failure.\n");
      exit(2);
    }


  /****  Receive Processing   ****/
  if(!(RX_DIS))
    if(mobilerx(RXFILE, OUTPUTFILE)) {
      printf("receive processing failure.\n");
      exit(4);
    }


  printf("Done.\n");
  return 0;
}
```

Appendix 2.2. Basetx.c

```
/* basetx.c
   written by Grant Smith
   11/17/97

   base transmit processing.

   designed for use within main.c

*/

int basetx(char *input, char *output)
{

  /****  declarations  ****/

  FILE *inputAudioFile;
  FILE *inputSatFile;
  FILE *inputWbdFile;
  FILE *outputFile;
  FILE *debug1;
  FILE *debug2;
  FILE *debug3;
  FILE *debug4;
  FILE *debug5;
  FILE *debug6;
  FILE *debug7;
  FILE *debug8;
  FILE *debug9;
  FILE *debug10;
  FILE *debug11;
  FILE *debug12;
  FILE *debug13;

  char filename[30];

  int mainIndex;
  int index;
  int wbdindex=0;
  int ifIndex;
  int ifIndex2;
  int ifIndex3;
  int interpaindex;
  int interpaflag;
  int interpbindex;

  int bufferIndex=0;
```

```
unsigned buffer[BUFFERSIZE];

/* signal path variables */
double audio;
double fil=0.0;
double com=0.0;
double pe=0.0;
double pe1=0.0;
double threshold;
double d1=0.0;
double txaudio1=0.0;
double interpb1=0.0;
double interpb2=0.0;
double txaudio=0.0;
double sat=0.0;
double txsat=0.0;
double wbd=0.0;
double wbdold=0.0;
double txwbd=0.0;
double phi=0.0;
unsigned txout=0;

double bpf_reg[TX_BPF_LENGTH];
double bpf_taps[TX_BPF_LENGTH] = TX_BPF_TAPS;
double bpf_reg2[TX_BPF_LENGTH];
double bpf_taps2[TX_BPF_LENGTH] = TX_BPF_TAPS2;
double comold=0.0;
double interpa_reg[TX_INTERPA_LENGTH];
double interpa_taps[TX_INTERPA_LENGTH] = TX_INTERPA_TAPS;
double postdev_reg[POSTDEV_LENGTH];
double postdev_taps[POSTDEV_LENGTH] = POSTDEV_TAPS;
double postdev_reg2[POSTDEV_LENGTH];
double postdev_taps2[POSTDEV_LENGTH] = POSTDEV_TAPS2;
double interpb1_reg[TX_INTERPB1_LENGTH];
double interpb2_reg[TX_INTERPB2_LENGTH];
double ampswbd00[12]= AMPSWBD00;
double ampswbd01[12]= AMPSWBD01;
double ampswbd10[12]= AMPSWBD10;
double ampswbd11[12]= AMPSWBD11;
double tacswbd00[15]= TACSWBD00;
double tacswbd01[15]= TACSWBD01;
double tacswbd10[15]= TACSWBD10;
double tacswbd11[15]= TACSWBD11;
double interp1_reg[TX_INTERP_LENGTH];
double interp2_reg[TX_INTERP_LENGTH];
double interp3_reg[TX_INTERP3_LENGTH];
double interp_taps[TX_INTERP_LENGTH] = TX_INTERP_TAPS;

double compGain=1;
double value;
```

```
  double interp1;
  double interp2;
  double interp3=0.0;

  double temp;
  double foo;


  /****  initializations ****/

  /* open files */

  /*strcpy(filename, input ".audio.dat");*/
  strcpy(filename, "data/input.audio.dat");
  if((inputAudioFile=fopen(filename, "r")) == NULL) {
    printf("Could not open file %s for input.\n", filename);
    return(1);
  }

  /*strcpy(filename, input ".sat.dat");*/
  strcpy(filename, "data/input.sat.dat");
  if((inputSatFile=fopen(filename, "r")) == NULL) {
    printf("Could not open file %s for input.\n", filename);
    return(1);
  }

  /*strcpy(filename, input ".wbd.dat");*/
  strcpy(filename, "data/input.wbd.dat");
  if((inputWbdFile=fopen(filename, "r")) == NULL) {
    printf("Could not open file %s for input.\n", filename);
    return(1);
  }

  if ((outputFile=fopen(output, "w")) == NULL) {
    printf("Could not open file %s for output.\n", output);
    return(2);
  }

  if ((debug1=fopen(TXDEBUG1, "w")) == NULL) {
    printf("Could  not  open  file  %s  for  debug  output.\n",
TXDEBUG1);
    return(3);
  }
  if ((debug2=fopen(TXDEBUG2, "w")) == NULL) {
    printf("Could  not  open  file  %s  for  debug  output.\n",
TXDEBUG2);
    return(3);
  }
  if ((debug3=fopen(TXDEBUG3, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG3);
    return(3);
  }
  if ((debug4=fopen(TXDEBUG4, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG4);
    return(3);
  }
  if ((debug5=fopen(TXDEBUG5, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG5);
    return(3);
  }
  if ((debug6=fopen(TXDEBUG6, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG6);
    return(3);
  }
  if ((debug7=fopen(TXDEBUG7, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG7);
    return(3);
  }
  if ((debug8=fopen(TXDEBUG8, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG8);
    return(3);
  }
  if ((debug9=fopen(TXDEBUG9, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG9);
    return(3);
  }
  if ((debug10=fopen(TXDEBUG10, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG10);
    return(3);
  }
  if ((debug11=fopen(TXDEBUG11, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG11);
    return(3);
  }
  if ((debug12=fopen(TXDEBUG12, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG12);
    return(3);
  }
  if ((debug13=fopen(TXDEBUG13, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
TXDEBUG13);
    return(3);
  }


  /* initialize filter registers */

  for (index=0; index<TX_BPF_LENGTH; index++) {
    bpf_reg[index]=0;
    bpf_reg2[index]=0;
  }
  for (index=0; index<TX_INTERPA_LENGTH; index++)
    interpa_reg[index]=0;
  for (index=0; index<POSTDEV_LENGTH; index++) {
    postdev_reg[index]=0;
    postdev_reg2[index]=0;
  }
  for (index=0; index<TX_INTERPB1_LENGTH; index++)
    interpb1_reg[index]=0;
  for (index=0; index<TX_INTERPB2_LENGTH; index++)
    interpb2_reg[index]=0;
  for (index=0; index<TX_INTERP_LENGTH; index++)
    interp1_reg[index]=0;
  for (index=0; index<TX_INTERP_LENGTH; index++)
    interp2_reg[index]=0;
  for (index=0; index<TX_INTERP3_LENGTH; index++)
    interp3_reg[index]=0;


  if (MODE==AMPS)
    threshold=AMPS_DEVIATION_THRESHOLD;
  if (MODE==JTACS)
    threshold=JTACS_DEVIATION_THRESHOLD;
  if (MODE==NTACS)
    threshold=NTACS_DEVIATION_THRESHOLD;


  /************  Processing  **************/


  for (mainIndex=0; mainIndex < TX_NUMSAMPLES; mainIndex++)
{

    /**** 8 kHz ****/

    /*    if(!((mainIndex*30) % TX_NUMSAMPLES))
      printf("%d\n",  (30-(mainIndex*30/TX_NUMSAMPLES)));  /*
to show progress */
```

```
    printf("~~%d~~\n",  (TX_NUMSAMPLES-mainIndex));  /*  to
show progress */



    /**** Audio Processing ****/

    /** Read Audio Input **/
    fscanf(inputAudioFile, "%lf\n", &audio);

    /** Band Pass Filter **/
    /**  IS-95: ???**/
    /* simple IIR filter, 300-3000 */
    /* audio in, fil out */

    temp=0.0;
    for (index=0; index<(TX_BPF_LENGTH-2); index++) {
      bpf_reg[index]=bpf_reg[index+1];
      temp+=bpf_reg[index]*bpf_taps[TX_BPF_LENGTH - index -
1];
      bpf_reg2[index+1]=bpf_reg2[index+2];
      temp-=bpf_reg2[index+1]*bpf_taps2[TX_BPF_LENGTH      -
index - 1];
    }
    bpf_reg[index]=bpf_reg[index+1];
    temp+=bpf_reg[index]*bpf_taps[TX_BPF_LENGTH  -  index  -
1];
    bpf_reg[TX_BPF_LENGTH-1]=audio;
    temp+=bpf_reg[TX_BPF_LENGTH-1] * bpf_taps[0];
    bpf_reg2[TX_BPF_LENGTH-1]=fil;
    temp-=bpf_reg2[TX_BPF_LENGTH-1] * bpf_taps2[1];

    fil=temp;

    if(TX_BPF_DIS) fil=audio;

    if(!(TXDEBUG1DIS))
      fprintf(debug1, "%4.7f\n", fil);

    if(TX_BPF_STOP) continue;

    /** Compressor **/
    /**  IS-95: 3.1.3.1.1**/
    /*fil in, com out */

    com=fil / compGain;
    compGain=((1-COMP_A)  *  compGain)  +  (COMP_KC  *  COMP_A  *
absdbl(com));

    if(TX_COM_DIS) com = fil;
    if(!(TXDEBUG2DIS))
      fprintf(debug2, "%4.7f\n", com);

    if(TX_COM_STOP) continue;

    /** Pre-Emphasis **/
    /**  IS-95: 3.1.3.1.2 **/
    /* H=k(1+bz-1)/(1+az-1) */
    /* com in, pe out */

    if(!(TX_PE_DIS)) {
      pe=PE_K*(com + PE_B * comold) - PE_A * pe;
      comold=com;
    }

    if(TX_PE_DIS) pe=com;

    if(!(TXDEBUG3DIS))
      fprintf(debug3, "%4.7f\n", pe);

    if(TX_PE_STOP) continue;

    /***  upsample to 20 kHz !!!!   ***/
    /* upsampling by 2.5  */
    /* we'll implement that with an upsampling by 5, and a
decimation by 2 */
    /* such that every other sample will be ignored. */


    for (interpaindex=0; interpaindex<5; interpaindex++) {


      /** Interpolation filter, 8->40  **/
      /* fir filter made with kaiser window to interpolate
from 8 to 40 */
      pe1=0.0;
      for (index=0; index<(TX_INTERPA_LENGTH-1); index++) {
     interpa_reg[index]=interpa_reg[index+1];
     pe1+=interpa_reg[index]*interpa_taps[TX_INTERPA_LENGTH
- index - 1];
      }
      if(interpaindex)
     interpa_reg[TX_INTERPA_LENGTH-1]=0;
      else
     interpa_reg[TX_INTERPA_LENGTH-1]=pe;
      pe1+=interpa_reg[TX_INTERPA_LENGTH-1]            *
interpa_taps[0];
```

```
      if (interpaflag) {
     interpaflag=0;
     continue;
      } else
     interpaflag=1;

      /* justify signal for -1 to 1 peak to peak */
      /* 10 for compressor, 5 for interpolation filter 8->40
*/
      pe1 = 50 * pe1;

      if(!(TXDEBUG4DIS))
     fprintf(debug4, "%4.7f\n", pe1);


      /** Deviation Limiter **/
      /**  IS-95: 3.1.3.1.3 **/
      /* simple threshold function.*/
      /* threshold chosen at 4.1379 such that anything with
deviation */
      /*   greater   than   12   kHz   will   be   cut   off
(threshold*deviation=max) */
      /* jtacs:  cutoff at 9.5 kHz, threshold @ 4.1304 */
      /* ntacs:  cutoff at 5 kHz, threshold @ 3.3333  */
      /* pe1 in, dl out */

      if(pe1 > threshold)
     dl=threshold;
      else if(pe1 < (-threshold))
     dl=-threshold;
      else
     dl=pe1;

      if(TX_DL_DIS) dl=pe1;

      if(!(TXDEBUG5DIS))
     fprintf(debug5, "%4.7f\n", dl);

      if(TX_DL_STOP) continue;


      /** Post Deviation-Limiter Filter **/
      /** IS-95: 3.1.3.1.4 **/
      /* simple IIR filter */
      /* dl in, txaudio out */

      temp = 0.0;
      for (index=0; index<(POSTDEV_LENGTH-2); index++) {
     postdev_reg[index]=postdev_reg[index+1];
     temp+=postdev_reg[index]*postdev_taps[POSTDEV_LENGTH  -
index - 1];
     postdev_reg2[index+1]=postdev_reg2[index+2];
     temp-
=postdev_reg2[index+1]*postdev_taps2[POSTDEV_LENGTH  -  index
-1];
      }
      postdev_reg[index]=postdev_reg[index+1];
      temp+=postdev_reg[index]*postdev_taps[POSTDEV_LENGTH -
index - 1];
      postdev_reg[POSTDEV_LENGTH-1]=dl;
      temp+=postdev_reg[POSTDEV_LENGTH-1] * postdev_taps[0];
      postdev_reg2[POSTDEV_LENGTH-1]=txaudio1;
      temp-=postdev_reg2[POSTDEV_LENGTH-1]                  *
postdev_taps2[1];

      txaudio1 = temp;

      if(TX_POSTDEV_DIS) txaudio1=dl;

      if(!(TXDEBUG6DIS))
     fprintf(debug6, "%4.7f\n", txaudio1);

      if(TX_POSTDEV_STOP) continue;

      /*********  upsample to 120 kHz!!!   *****/

      for (interpbindex=0; interpbindex<6; interpbindex++) {

     /** Interpolation filter, 20->120  **/

     /** interpolation filter b1 (sinc) **/
     /*  note  zero-order  hold  rather  than  zero  stuff
interpolation */

     interpb1=interpb1+(txaudio1-interpb1_reg[0])/8;
     for (index=0; index<(TX_INTERPB1_LENGTH-1); index++)
       interpb1_reg[index]=interpb1_reg[index+1];
     interpb1_reg[TX_INTERPB1_LENGTH-1]=txaudio1;

     /** interpolation filter b2 (sinc) **/
     interpb2=interpb2+(interpb1-interpb2_reg[0])/4;
     for (index=0; index<(TX_INTERPB2_LENGTH-1); index++)
       interpb2_reg[index]=interpb2_reg[index+1];
     interpb2_reg[TX_INTERPB2_LENGTH-1]=interpb1;

     txaudio = interpb2;


     if(!(TXDEBUG7DIS))
```

```
       fprintf(debug7, "%4.7f\n", txaudio);

     if(TX_AUDIOSET_STOP) continue;


     /**** SAT Processing****/

     if (!(TX_SAT_DIS)) {

       /** Read SAT Input **/
       fscanf(inputSatFile, "%lf\n", &sat);
       txsat = sat;

     }

     /**** Wideband Data Processing  ****/

     if (!(TX_WBD_DIS)) {  /* 120 kHz */

       if (MODE==AMPS) {

         /* wbd is read in every 12th.  then, depending upon
the previous
             wbd, the last 6 pieces of old data and first 6
of new data are
            produced */
         /* wbd read in, txwbd out */

         if (wbdindex==12) {
           wbdold=wbd;
           fscanf(inputWbdFile, "%lf\n", &wbd); /* 10kbps */
           wbdindex=0;
         }

         if (wbdold)
           if (wbd)   /* 1, 1 */
          txwbd=ampswbd11[wbdindex];
           else       /* 1, 0 */
          txwbd=ampswbd10[wbdindex];
         else
           if (wbd)   /* 0, 1 */
          txwbd=ampswbd01[wbdindex];
           else       /* 0, 0 */
          txwbd=ampswbd00[wbdindex];

         wbdindex++;

       }  else {         /*  JTACS or NTACS  */
```

```
         /* wbd is read in every 15th.  then, depending upon
the previous
             wbd, the last 6 pieces of old data and first 7
of new data are
            produced */
         /* wbd read in, txwbd out */

         if (wbdindex==15) {
           wbdold=wbd;
           fscanf(inputWbdFile, "%lf\n", &wbd);  /* 8kbps */
           wbdindex=0;
         }

         if (wbdold)
           if (wbd)    /* 1, 1 */
          txwbd=tacswbd11[wbdindex];
           else        /* 1, 0 */
          txwbd=tacswbd10[wbdindex];
         else
           if (wbd)    /* 0, 1 */
          txwbd=tacswbd01[wbdindex];
           else        /* 0, 0 */
          txwbd=tacswbd00[wbdindex];

         wbdindex++;

       }

       if(!(TXDEBUG8DIS))
         fprintf(debug8, "%1.4f\n", txwbd);
     }


     /**** Sum Signals, Modulate  ****/
     /** multiply each signal by its deviation, add carrier,
       divide by sampling frequency,
       produce limited tone  **/

     if (MODE==AMPS) {
       value = TX_AMPS_AUDIO_DEVIATION * txaudio;
        if(!(TX_SAT_DIS)) value += TX_AMPS_SAT_DEVIATION *
txsat;
        if(!(TX_WBD_DIS)) value += TX_AMPS_WBD_DEVIATION *
txwbd;
     }

     if (MODE==JTACS) {
       value = TX_JTACS_AUDIO_DEVIATION * txaudio;
        if(!(TX_SAT_DIS)) value += TX_JTACS_SAT_DEVIATION *
txsat;
```

```
        if(!(TX_WBD_DIS)) value += TX_JTACS_WBD_DEVIATION *
txwbd;
     }

     if (MODE==NTACS) {
       value = TX_NTACS_AUDIO_DEVIATION * txaudio;
        if(!(TX_SAT_DIS)) value += TX_NTACS_SAT_DEVIATION *
txsat;
        if(!(TX_WBD_DIS)) value += TX_NTACS_WBD_DEVIATION *
txwbd;
     }



     if(!(TXDEBUG9DIS))
     fprintf(debug9, "%4.7f\n", value);

     if(TX_SIGNAL_STOP) continue;

     /** interpolate to 19.68 MHz, first by 2, then by 2,
then by 41 **/

     /* interpolate by 2 */
     for (ifIndex=0; ifIndex < 2; ifIndex++) {

       /** interpolation filter **/
       /* from matlab:  b=fir1(36, .5) */
       interp1=0.0;
       for (index=0; index<(TX_INTERP_LENGTH-1); index++) {
         interp1_reg[index]=interp1_reg[index+1];

interp1+=interp1_reg[index]*interp_taps[TX_INTERP_LENGTH   -
index - 1];
       }
       if(ifIndex==0)  /* zero stuff */
         interp1_reg[TX_INTERP_LENGTH-1]=value;
       else
         interp1_reg[TX_INTERP_LENGTH-1]=0;
              interp1+=interp1_reg[TX_INTERP_LENGTH-1]     *
interp_taps[0];

       /*justify signal */
       interp1=interp1 * 2;

       if(!(TXDEBUG10DIS))
         fprintf(debug10, "%4.7f\n", interp1);


       if(TX_INTERP1_STOP) continue;
```

```
        /* interpolate by 2 */
        for (ifIndex2=0; ifIndex2 < 2; ifIndex2++) {

          /** interpolation filter **/
          interp2=0.0;
           for (index=0; index<(TX_INTERP_LENGTH-1); index++)
{
            interp2_reg[index]=interp2_reg[index+1];

interp2+=interp2_reg[index]*interp_taps[TX_INTERP_LENGTH   -
index - 1];
          }
          if(ifIndex2==0)
            interp2_reg[TX_INTERP_LENGTH-1]=interp1;
          else
            interp2_reg[TX_INTERP_LENGTH-1]=0;
                 interp2+=interp2_reg[TX_INTERP_LENGTH-1]   *
interp_taps[0];

          /*justify signal */
          interp2=interp2 * 2;

          if(!(TXDEBUG11DIS))
            fprintf(debug11, "%4.7f\n", interp2);


          if(TX_INTERP2_STOP) continue;

          /* interpolate by 41 */
          for (ifIndex3=0; ifIndex3 < 41; ifIndex3++) {

            /** interpolation filter (sinc) **/
            /* note zero order hold for sinc ^2 */
            /* factor of 64 for fir filter bounding*/
            interp3=interp3+(interp2-interp3_reg[0])/41;
                 for  (index=0;  index<(TX_INTERP3_LENGTH-1);
index++)
           interp3_reg[index]=interp3_reg[index+1];
            interp3_reg[TX_INTERP3_LENGTH-1]=interp2;


            if(!(TXDEBUG12DIS))
           fprintf(debug12, "%4.7f\n", interp3);

                   phi+=   360.0   *   (interp3   +   FM_IF)   /
IF_SAMPLING_FREQUENCY;

            if (phi>180.0)  {
           phi = phi-180.0;
          txout = 1-txout;
           }


           /****  Write TX output ****/
           buffer[bufferIndex++]=txout;

           /**** dump buffer  ****/
           if(bufferIndex==BUFFERSIZE) {
          if((fwrite(buffer,  sizeof(unsigned),  BUFFERSIZE,
outputFile) == 0)) {
            printf("error while writing tx output\n");
            exit(2);
          }
          bufferIndex=0;
           }
         }
       }
     }
      }
    }
  }


  /****  Cleanup ****/
  fclose(inputAudioFile);
  fclose(inputSatFile);
  fclose(inputWbdFile);
  fclose(outputFile);
  fclose(debug1);
  fclose(debug2);
  fclose(debug3);
  fclose(debug4);
  fclose(debug5);
  fclose(debug6);
  fclose(debug7);
  fclose(debug8);
  fclose(debug9);
  fclose(debug10);
  fclose(debug11);
  fclose(debug12);
  fclose(debug13);

  printf("AMPS transmit processing completed.\n");
  return 0;

}
```

Appendix 2.3. Mobilerx.c

```
/* mobilerx.c
   written by Grant Smith
   11/17/97

   designed for use within main.c

*/

int mobilerx(char *input, char *output)
{

  /****  declarations  ****/

  FILE *inputFile;
  FILE *outputAudioFile;
  FILE *outputSatFile;
  FILE *outputWbdFile;
  FILE *debug1;
  FILE *debug2;
  FILE *debug3;
  FILE *debug4;
  FILE *debug5;
  FILE *debug6;
  FILE *debug7;
  FILE *debug8;
  FILE *debug9;
  FILE *debug10;
  FILE *debug11;
  FILE *debug12;
  FILE *debug13;
  FILE *debug14;
  FILE *debug15;
  FILE *debug16;
  FILE *debug17;

  char filename[30];

  int mainIndex;
  int index;
  int ifIndex;
  int wbdIndex;

  unsigned int buffer[BUFFERSIZE];
  int bufferIndex=BUFFERSIZE;

  /* signal path variables  */
  unsigned rxin;
  unsigned rxsignal=0;
  unsigned rxsignalold=0;
  unsigned count=0;
  unsigned countold=0;
  unsigned acc=0;
  unsigned accold;
  unsigned dec=0;
  unsigned decold;
  unsigned sub=0;
  unsigned subold=0;
  unsigned sig;
  unsigned sigold=12100;

  int sata1=0, sata2=0, sata3=0, satb1=0, satb2=0, satc1=0,
satc2=0, satc3=0;
  int satdec=0;
  int satsig=0;
  double satsigdbl=0.0;
  int satsig2=0;
  int sat=4;

  int wbdsig=12100;
  int wbdsigold=12100;
  int wbdinterp=0;
  int wbdinterpold=0;
  double wbdhpf=0.0;
  int wbdtd=0;
  int wbdtdold=0;
  int wbdzc=0;
  double wbdphaseacc=0.0;
  double wbdloopgained=0.0;
  int manclk=0;
  int manclkold=0;
  int mandata=0;
  int mandataold=0;
  int manflag=1;
  int wbd;

  int auddec=0;
  int fil=0;
  double fildbl=0;
  int de=0;
  int audio;


  /*filter variables  */
  double wbd_hpf_reg[RX_WBD_HPF_LENGTH];
  double wbd_hpf_taps[RX_WBD_HPF_LENGTH] = RX_WBD_HPF_TAPS1;
  double wbd_hpf_taps2[RX_WBD_HPF_LENGTH] =
RX_WBD_HPF_TAPS2;
  double sat_reg[RX_SAT_BPF_LENGTH];
  double sat_taps[RX_SAT_BPF_LENGTH] = RX_SAT_BPF_TAPS1_1;
  double sat_taps12[RX_SAT_BPF_LENGTH] = RX_SAT_BPF_TAPS1_2;
  double sat_reg2[RX_SAT_BPF_LENGTH];
  double sat_taps2[RX_SAT_BPF_LENGTH] = RX_SAT_BPF_TAPS2_1;
  double sat_taps22[RX_SAT_BPF_LENGTH] = RX_SAT_BPF_TAPS2_2;
  double dec_reg[RX_DEC_LENGTH];
  double dec_taps[RX_DEC_LENGTH] = RX_DEC_TAPS1;
  double dec_taps2[RX_DEC_LENGTH] = RX_DEC_TAPS2;
  double bpf_reg[RX_BPF_LENGTH];
  double bpf_taps[RX_BPF_LENGTH] = RX_BPF_TAPS;
  double bpf_reg2[RX_BPF_LENGTH];
  double bpf_taps2[RX_BPF_LENGTH] = RX_BPF_TAPS2;

  /* sat detection variables */
  int satcount = SATWINDOW;
  int satzcs = 0;
  double satsig2old;

  double expGain=1;

  double temp;

  /****  initializations ****/

  if((inputFile=fopen(input, "r")) == NULL) {
    printf("Could not open file %s for input.\n", input);
    return(1);
  }

  /*strcpy(filename, output ".audio.dat");*/
  strcpy(filename, "data/output.audio.dat");
  if ((outputAudioFile=fopen(filename, "w")) == NULL) {
    printf("Could not open file %s for output.\n",
filename);
    return(2);
  }

  /*strcpy(filename, output ".sat.dat");*/
  strcpy(filename, "data/output.sat.dat");
  if ((outputSatFile=fopen(filename, "w")) == NULL) {
    printf("Could not open file %s for output.\n",
filename);
    return(2);
  }

  /*strcpy(filename, output ".wbd.dat");*/
  strcpy(filename, "data/output.wbd.dat");
  if ((outputWbdFile=fopen(filename, "w")) == NULL) {
    printf("Could not open file %s for output.\n",
filename);
    return(2);
  }

  if ((debug1=fopen(RXDEBUG1, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG1);
    return(3);
  }
  if ((debug2=fopen(RXDEBUG2, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG2);
    return(3);
  }
  if ((debug3=fopen(RXDEBUG3, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG3);
    return(3);
  }
  if ((debug4=fopen(RXDEBUG4, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG4);
    return(3);
  }
  if ((debug5=fopen(RXDEBUG5, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG5);
    return(3);
  }
  if ((debug6=fopen(RXDEBUG6, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG6);
    return(3);
  }
  if ((debug7=fopen(RXDEBUG7, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG7);
    return(3);
  }
  if ((debug8=fopen(RXDEBUG8, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG8);
    return(3);
  }
  if ((debug9=fopen(RXDEBUG9, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG9);
    return(3);
  }
  if ((debug10=fopen(RXDEBUG10, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG10);
    return(3);
  }
  if ((debug11=fopen(RXDEBUG11, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG11);
    return(3);
  }
  if ((debug12=fopen(RXDEBUG12, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG12);
    return(3);
  }
  if ((debug13=fopen(RXDEBUG13, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG13);
    return(3);
  }
  if ((debug14=fopen(RXDEBUG14, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG14);
    return(3);
  }
  if ((debug15=fopen(RXDEBUG15, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG15);
    return(3);
  }
  if ((debug16=fopen(RXDEBUG16, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG16);
    return(3);
  }
  if ((debug17=fopen(RXDEBUG17, "w")) == NULL) {
    printf("Could not open file %s for debug output.\n",
RXDEBUG17);
    return(3);
  }

  /* initialize filter registers */

  for (index=0; index<RX_WBD_HPF_LENGTH; index++)
    wbd_hpf_reg[index]=0;
  for (index=0; index<RX_SAT_BPF_LENGTH; index++) {
    sat_reg[index]=0;
    sat_reg2[index]=0;
  }
  for (index=0; index<RX_DEC_LENGTH; index++)
    dec_reg[index]=0;
  for (index=0; index<RX_BPF_LENGTH; index++) {
    bpf_reg[index]=0;
    bpf_reg2[index]=0;
  }


  /* initialize dec_taps, 2nd half  (50 entries is too long
for
     initialization within declaration */
  for (index=0; index< (RX_DEC_LENGTH/2) ; index++)
    dec_taps[index+RX_DEC_LENGTH/2] = dec_taps2[index];
  /* initialize sat saps */
  for (index=5; index< 9 ; index++)
    sat_taps[index] = sat_taps12[index-5];
  for (index=5; index< 9 ; index++)
    sat_taps2[index] = sat_taps22[index-5];
  /* initialize wbd hpf taps */
  for (index=0; index< 26 ; index++)
    wbd_hpf_taps[index+26]= wbd_hpf_taps2[index];



  /************  Processing  **************/

  for (mainIndex=0; mainIndex < RX_NUMSAMPLES; mainIndex++)
{  /* 40 kHz */

    if(!((mainIndex*50) % RX_NUMSAMPLES))
      printf("**%d**\n", (50-(mainIndex*50/RX_NUMSAMPLES)));
/* to show progress */

    for   (ifIndex=0;   ifIndex   <   RX_DECIMATION_FACTOR;
ifIndex++) {

      /**** Read RX Input ****/
      /* is at IF_SAMPLING_FREQUENCY  (19.68 MHz) */
      /* fscanf(inputFile, "%u\n", &rxin); */

      if (bufferIndex == BUFFERSIZE) {
     if   ((fread(buffer,   sizeof(unsigned),   BUFFERSIZE,
inputFile))==0) {
       printf("error while reading rx input\n");
       exit(2);
     }
     bufferIndex=0;
      }

      rxin=buffer[bufferIndex++];
```

```
   if(!(RXDEBUG1DIS))
  fprintf(debug1, "%u\n", rxin);

   /**** Demodulate ****/
   /** Hovin **/
   /* drops us to 40 kHz */

   rxsignalold = rxsignal;
   rxsignal = 1 - ((int) rxin);  /* remove DC bias */

   if(!(RXDEBUG2DIS))
  fprintf(debug2, "%u\n", rxsignal);

   /* Counter  */
   countold = count;
   if((rxsignal - rxsignalold) >0)
  count = countold + 1;
   else
  count = countold;
   count = count & MASK;  /* mod-x */

   if(!(RXDEBUG3DIS))
  fprintf(debug3, "%u\n", count);

   /*  Accumulate */
   accold=acc;
   acc = count + accold;
   acc = acc & MASK;

   if(!(RXDEBUG4DIS))
  fprintf(debug4, "%u\n", acc);

 }

/* decimate */
decold = dec;
dec = acc;

if(!(RXDEBUG5DIS))
  fprintf(debug5, "%u\n", dec);

/* subtract 1st */
subold=sub;
sub = dec-decold;
sub = sub & MASK;

if(!(RXDEBUG6DIS))
  fprintf(debug6, "%u\n", sub);
```

```
    /*  subtract 2nd */
    sig = sub-subold;
    sig = sig & MASK;

    if(mainIndex==0)
      sig=12106;

    if(!(RXDEBUG7DIS))
      fprintf(debug7, "%u\n", sig);


    /**********************/
    /**** Recover WBD  ****/
    /**********************/

    if (!(RX_WBD_DIS)) {   /* same for all modes?  */

      /** Low Pass Filter **/
      /* sig in, wbdsig out */
      /* 2-tap sinc */

      wbdsigold=wbdsig;
      wbdsig=(sig+sigold)/2;
      sigold=sig;

      if(RX_WBD_BPF_DIS) wbdsig=sig;

      if(!(RXDEBUG8DIS))
     fprintf(debug8, "%d\n", wbdsig);


      for(wbdIndex=0; wbdIndex < 8; wbdIndex++) {    /* 320
kHz */

     /** Interp Filter **/
     /* linear interpolation between wbdsigold and wbdsig */
     /* wbdsig in, wbdinterp out */

     if (wbdIndex==0)
       wbdinterp=wbdsigold;
     else
           wbdinterp=(int)   (wbdinterp+((double)   (wbdsig-
wbdsigold))/8);

     if (RX_WBD_INT_DIS) wbdinterp=wbdsig;

     if(!(RXDEBUG9DIS))
       fprintf(debug9, "%d\n", wbdinterp);

     /** High Pass Filter **/
     /* eliminate noise from audio band */
     /* FIR1(50, 3/160, 'high')  */
     /* wbdinterp in, wbdhpf out */

     wbdhpf=0.0;
     for (index=0; index<(RX_WBD_HPF_LENGTH-1); index++) {
       wbd_hpf_reg[index]=wbd_hpf_reg[index+1];

wbdhpf+=wbd_hpf_reg[index]*wbd_hpf_taps[RX_WBD_HPF_LENGTH  -
index - 1];
     }
     wbd_hpf_reg[RX_WBD_HPF_LENGTH-1]=wbdinterp;
     wbdhpf+=wbd_hpf_reg[RX_WBD_HPF_LENGTH-1]                *
wbd_hpf_taps[0];

     wbdhpf=wbdhpf-5696;   /* set average to zero */

     if (RX_WBD_HPF_DIS) wbdhpf=wbdinterp;

     if(!(RXDEBUG15DIS))
       fprintf(debug15, "%4.7f\n", wbdhpf);


     /** Decode wbd  **/
     /* wbdinterp in, write wbd out */

     wbdtdold=wbdtd;
     wbdtd = (wbdhpf > MAN_THRESHOLD);
     wbdzc = (!(wbdtd == wbdtdold));

     if(!(RXDEBUG16DIS))
       fprintf(debug16, "%d\n", wbdtd);

     wbdloopgained=wbdphaseacc*WBDLOOPGAIN*wbdzc;
     wbdphaseacc=wbdphaseacc+(1.0/8.0)-wbdloopgained;

     if(!(RXDEBUG17DIS))
       fprintf(debug17, "%4.7f\n", wbdphaseacc);

     if (wbdphaseacc>1)
       wbdphaseacc=-1;

     manclkold=manclk;

     if (wbdphaseacc>0)
       manclk=1;
     else
       manclk=0;

     if (manclk<manclkold) {  /*20 kHz (eventually) */
```

```
       mandataold=mandata;
       mandata=wbdtd;

       if (manflag) { /* first of two */

         manflag=0;
       } else {   /* second of two in symbol */
          if (!(mandata==mandataold)) {   /* we're in phase
*/
           manflag=1;
           wbd=mandata;

           /** Write WBD **/
           fprintf(outputWbdFile , "%d\n", wbd);
          }       /* if out of phase, remain at second in
symbol */
       }
     }
      }
    }

    /***********************/
    /**** Recover SAT  ****/
    /***********************/

    if (!(RX_SAT_DIS)) {

      if ((MODE==JTACS) || (MODE==AMPS)) {

     /** Decimation Filters **/
     /* 3 tap sinc, 2 tap sinc, 2 tap sinc ^2, dec by 2  */
     /* sig in, satdec out */

     sata1=sata2;
     sata2=sata3;
     sata3= (int) sig;
     satb1=satb2;
     satb2=(sata1+sata2+sata3)/4;
     satc1=satc2;
     satc2=satc3;
     satc3=(satb1+satb2)/2;

     fprintf(debug9, "%d\n", satc3);

     satdec=(satc1+2*satc2+satc3)/4;

     if (RX_SAT_DEC_DIS) satdec=sig;

     if(!(RXDEBUG10DIS))
```

```
       fprintf(debug10, "%d\n", satdec);


     if (!(mainIndex % 2)) {    /* 20 kHz */

       /** SAT Band Pass Filter **/
       /* where is this filter from?  */
       /* satdec in satsig out*/

       temp=0.0;
       for (index=0; index<(RX_SAT_BPF_LENGTH-2); index++) {
         sat_reg[index]=sat_reg[index+1];
           temp+=sat_reg[index]*sat_taps[RX_SAT_BPF_LENGTH -
index - 1];
         sat_reg2[index+1]=sat_reg2[index+2];
         temp-=sat_reg2[index+1]*sat_taps2[RX_SAT_BPF_LENGTH
- index-1];
       }
       sat_reg[index]=sat_reg[index+1];
          temp+=sat_reg[index]*sat_taps[RX_SAT_BPF_LENGTH  -
index-1];
       sat_reg[RX_SAT_BPF_LENGTH-1]=satdec;
       temp+=sat_reg[RX_SAT_BPF_LENGTH-1] * sat_taps[0];
       sat_reg2[RX_SAT_BPF_LENGTH-1]=satsigdbl;
       temp-=sat_reg2[RX_SAT_BPF_LENGTH-1] * sat_taps2[1];

       satsigdbl=temp;

       satsig= (int) temp;

       if(RX_SAT_BPF_DIS) satsig=satdec;

       if(!(RXDEBUG11DIS))
         fprintf(debug11, "%d\n", satsig);


       /** hard limit **/
       /* satsig in, satsig2 out*/
       if(satsig<SATTHRESHOLD)
         satsig2=0;
       else satsig2=1;

       /** Identify SAT with ZC counter **/
       /* satsig2 in, sat out */

       if(--satcount == 0) {
         /*choose sat*/
         if(satzcs<SAT1)
           sat=0;
         else if(satzcs<SAT2)
```

```
           sat=1;
         else if(satzcs<SAT3)
           sat=2;
         else if(satzcs<SAT4)
           sat=3;
         else sat=0;
         satcount=SATWINDOW;
         satzcs=0;
       }
       else {
         /*count ZCs */
         if(!(satsig2old==satsig2))
           satzcs++;
         satsig2old=satsig2;
       }

       /** Write SAT **/
       fprintf(outputSatFile , "%d\n", sat);
     }
    }
    }


    /**************************/
    /**** Audio Processing ****/
    /**************************/

    /** Decimation Filter **/
    /* 50 tap FIR, f=3.75/20; n=[-24.5:24.5];
       b=f*sinc(f*n); b=b.*kaiser(50, 4.2); */
    /* sig in, auddec out */


    temp=0.0;
    for (index=0; index<(RX_DEC_LENGTH-1); index++) {
      dec_reg[index]=dec_reg[index+1];
      temp+=dec_reg[index]*dec_taps[RX_DEC_LENGTH - index -
1];
    }
    dec_reg[RX_DEC_LENGTH-1]=sig;
    temp+=dec_reg[RX_DEC_LENGTH-1] * dec_taps[0];

    auddec= (int) temp;

    if (RX_AUD_DEC_DIS) auddec=sig;

    if(!(RXDEBUG12DIS))
      fprintf(debug12, "%d\n", auddec);
    if (!(mainIndex % 5)) {    /* 8 kHz */


      /** Band Pass Filter **/
      /**  IS-95: ?? **/
      /* simple IIR filter */
      /* auddec in, fil out*/

      temp=0.0;
      for (index=0; index<(RX_BPF_LENGTH-2); index++) {
     bpf_reg[index]=bpf_reg[index+1];
     temp+=bpf_reg[index]*bpf_taps[RX_BPF_LENGTH  -  index  -
1];
     bpf_reg2[index+1]=bpf_reg2[index+2];
     temp-=bpf_reg2[index+1]*bpf_taps2[RX_BPF_LENGTH - index
- 1];
      }
      bpf_reg[index]=bpf_reg[index+1];
      temp+=bpf_reg[index]*bpf_taps[RX_BPF_LENGTH  -  index  -
1];
      bpf_reg[RX_BPF_LENGTH-1]=auddec;
      temp+=bpf_reg[RX_BPF_LENGTH-1] * bpf_taps[0];
      bpf_reg2[RX_BPF_LENGTH-1]=fildbl;
      temp-=bpf_reg2[RX_BPF_LENGTH-1] * bpf_taps2[1];

      fildbl=temp;

      fil=(int) temp;

      if(RX_BPF_DIS) fil=auddec;

      if(!(RXDEBUG13DIS))
     fprintf(debug13, "%d\n", fil);


      /** De-Emphasis **/
      /**  IS-95: 2.2.2.1.1 **/
      /*   single   pole   low-pass   IIR   H(z)=k/(1+az^-1),
k=.727937, a=-.88  */
      /* fil in, de out */


      if(RX_DE_DIS)
     de=(int) fil;
      else
     de=(int)  (DE_K*fil - DE_A * de);

      if(!(RXDEBUG14DIS))
     fprintf(debug14, "%d\n", de);
```

```
      /** Expandor **/
      /**  IS-95: 2.2.2.1.2 **/
      /* de in, audio out */

      expGain=((1-EXP_A)  *  expGain)  +  (EXP_KC  *  EXP_A  *
((double) abs(de)));
      audio = (int) (((double) de) * expGain);

      if(RX_EXP_DIS) audio=de;

      /** Write Audio output **/
      fprintf(outputAudioFile , "%d\n", audio);
    }
  }



  /****  Cleanup ****/
  fclose(inputFile);
  fclose(outputAudioFile);
  fclose(outputSatFile);
  fclose(outputWbdFile);
  fclose(debug1);
  fclose(debug2);
  fclose(debug3);
  fclose(debug4);
  fclose(debug5);
  fclose(debug6);
  fclose(debug7);
  fclose(debug8);
  fclose(debug9);
  fclose(debug10);
  fclose(debug11);
  fclose(debug12);
  fclose(debug13);
  fclose(debug14);
  fclose(debug15);
  fclose(debug16);
  fclose(debug17);

  printf("AMPS receive processing completed.\n");
  return 0;

}
```

Appendix 2.4. Satntacsgen.c

```
/* satntacsgen.c
   by Grant Smith
   4/22/98

   create SAT/ST from input file
                 choice of 14 different sequences, 7 each in
SAT and ST
           read in at 8 1/3 samples/second.

   intended for use as test input to  main.c
*/


#include <stdio.h>
#include <math.h>

#define AUD_SAMPLES 5000
#define SAMPLES ((AUD_SAMPLES*SAMPLINGFREQ/8000)+1)
#define SAMPLINGFREQ 9.0


#define INPUTFILE "ntacssatin.dat"
#define OUTPUTFILE "ntacssat.dat"
#define OUTPUTFILE2 "ntacssat120.dat"

int main() {

  int index;
  int i;
  int j;
  int k;
  FILE *infile;
  FILE *outfile;
  FILE *outfile2;

  char codes[14][6]= {{2,5,5,6,12,11},
                 {2,5,5,11,2,11},
                 {2,5,6,10,9,11},
                 {2,5,10,13,4,13},
                 {2,6,10,11,2,11},
                 {2,6,11,2,10,13},
                 {2,9,6,9,10,11},
                 {13,10,10,9,3,4},
                 {13,10,10,4,13,3},
              {13,10,9,5,6,4},
              {13,10,5,2,11,2},
              {13,9,5,4,13,4},
              {13,9,4,13,5,2},
              {13,6,9,6,5,4}};

char masks[14][6]= {{15,15,0,0,3,14},
              {0,11,11,15,8,2},
              {11,13,7,8,0,15},
              {3,15,15,1,1,8},
              {0,10,14,6,15,6},
              {8,0,0,1,15,15},
              {1,12,0,15,12,13},
              {15,15,0,0,3,14},
              {0,11,11,15,8,2},
              {11,13,7,8,0,15},
              {3,15,15,1,1,8},
              {0,10,14,6,15,6},
              {8,0,0,1,15,15},
              {1,12,0,15,12,13}};

int hex[16][4]= {{0,0,0,0},
           {0,0,0,1},
           {0,0,1,0},
           {0,0,1,1},
           {0,1,0,0},
           {0,1,0,1},
           {0,1,1,0},
           {0,1,1,1},
           {1,0,0,0},
           {1,0,0,1},
           {1,0,1,0},
           {1,0,1,1},
           {1,1,0,0},
           {1,1,0,1},
           {1,1,1,0},
           {1,1,1,1}};


int input=0;
int inputold=0;
int output=0;


if ((infile = fopen(INPUTFILE, "r")) == NULL) {
  printf("can't open input file\n");
  exit(2);
}
if ((outfile = fopen(OUTPUTFILE, "w")) == NULL) {
  printf("can't open output file\n");
```

```
    exit(2);
  }
  if ((outfile2 = fopen(OUTPUTFILE2, "w")) == NULL) {
    printf("can't open output file\n");
    exit(2);
  }


  for(index=0; index<SAMPLES; index++) {
    for (i=0; i<6; i++) {
      for (j=0; j<4; j++){    /* 200 bps */

     inputold=input;
     fscanf(infile, "%d\n", &input);

     /* we can only change inputs on true bit in mask */
     if (!(hex[masks[input][i]][j]))
       input=inputold;

     output=hex[codes[input][i]][j];

     fprintf(outfile, "%d\n", output);

     for (k=0; k<600; k++){  /* 120000 kHz */
       fprintf(outfile2, "%d\n", output);

     }
      }
    }
  }



  fclose(infile);
  fclose(outfile);
  fclose(outfile2);

  printf("done\n");

  return 0;
}
```

7 - 20