Application of the Residue Number System to Digital Communication Systems

Todd R. Hunter

1997

Application of the Residue Number System to Digital Communication Systems

A Thesis

Submitted to the College of Graduate Studies and Research

in Partial Fulfilment of the Requirements

for the Degree of Master of Science

in the Department of Electrical Engineering

University of Saskatchewan

by

Todd Ronald Hunter

Saskatoon, Saskatchewan, Canada

December, 1997

COPYRIGHT

ABSTRACT

Digital signal processors (DSP) are beginning to dominate the communications industry. The demand of the industry is to provide smaller, lighter, lower powered communication technology without sacrificing performance. This study investigates the application of an alternative method of implementation for a digital frequency modulation receiver, the cross-correlator. Instead of using a traditional fixed point DSP based platform, residue number systems (RNS) are applied to the application. The main focus of this work is to evaluate the performance of an RNS based cross-correlator receiver and compare it to the same receiver using traditional fixed point DSP techniques.

Theoretical calculations are performed on the receiver and a model is developed. The model is developed to allow for the analysis of both the fixed point DSP based receiver as well as the RNS based receiver. This model is used to analyze the bit error rate (BER) performance of the receiver for various input signal to noise ratios. Comparison of the model's BER results indicates that an RNS based receiver using 6-bits for quantization of all receiver inputs and 6-bits at the output of the receiver performs as well as an 8-bit fixed point DSP based receiver.

The cross-correlator receiver is also simulated to determine the accuracy of the model. Both the RNS and DSP based methods of implementation are simulated. The BER results from the simulation show that the RNS based receiver using 6-bits for quantization and output does not perform as well as the 8-bit DSP based system. However by using 8-bits for the quantization of the low pass filter coefficients, the BER performance of the RNS based system significantly improves. The model and simulation provide similar performance ratings, however the model is not as affected by the characteristics of the low pass filters.

ACKNOWLEDGMENTS

The author thanks Professor Brian L. F. Daku for his guidance and instruction throughout the course of this research.

The author also thanks Mr. Leon Lipoth, Mr. Garth Wells and the rest of the crew of TRLabs for their encouragement, support and guidance.

Telecommunications Research Laboratories (TRLabs) is gratefully acknowledged for providing financial support.

Last, but not least, the author would like to thank some good friends who provided encouragement and humor at all times: Tracey, Robert, Paymaan and Denise.

TO

My Mother and Father

for all their love, encouragement and support.

Contents

List of Figures

List of Tables

ABBREVIATIONS

AC	- alternating current
A/D	- analog to digital
ASIC	- application specific integrated circuit
AWGN	- additive white Gaussian noise
baud	- symbols per second
BER	- bit error rate
CRT	- Chinese remainder theorem
dB	- decibel
DC	- direct current
DSP	- digital signal processing
FFT	- fast fourier transform
FIR	- finite impulse response
FM	- frequency modulation
FSK	- frequency shift keying
GHz	- gigahertz
Hz	- cycles per second
kHz	- kilohertz
LSB	- least significant bit
MMRNS	- multi-moduli residue number system
MSB	- most significant bit

PSK	- phase shift keying
RAM	- random access memory
RNS	- residue number system
ROM	- read only memory
SNR	- signal to noise ratio
SQNR	- signal to quantization noise ratio
VLSI	- very large scale integration

Chapter 1

INTRODUCTION

1.1 Background

Digital signal processing (DSP) is one of the most dominant forces in the communications industry. Digital signal processors are being used in cellular phones, satellite communications, modems, fax machines, and the list goes on. One of the key features of digital signal processing is that the processing is done in software that can be programmed and reprogrammed. This allows a single hardware setup to be very versatile and adaptive to changing requirements. The other advantage of digital signal processing over analog processing is that an entire processing system can be confined to a few integrated circuits. A typical system may require an analog-to-digital (A/D) converter, a digital signal processor, and some external memory. Therefore, the size, weight and power requirements for a DSP-based system tend to be smaller than those for analog processing systems. With the current rate of advances in circuit fabrication, soon an entire system will be implemented on a single integrated circuit, including A/D converter, and enough memory for the entire system.

For reasons that are obvious, designers and manufacturers of communication technology, especially hand-held products, aim to reduce the size, weight and power requirements of their products. They would also prefer to minimize the costs associated with developing these products. Part of the drive behind designing with digital signal processors is the fact that they can perform many functions, while being contained within a small, lightweight package. However, if a design is required that only performs a single function, a DSP system may not be the optimal solution. In these cases, an application specific integrated circuit (ASIC) may provide a more efficient solution, consuming less power or chip real-estate while at a lower cost.

While there seem to be many advantages to digital signal processing, there is room for improvement. One area, that is of concern, is the effect of the finite word lengths inherently present in digital systems. The finite word lengths force the processor to scale or truncate numbers and only retain a certain amount of precision in the numbers being processed. This loss of precision can also be considered as error, or internally generated noise, since it is a deviation from the desired. It is a false assumption that once the signals are in digital format they are then operated on in a noise free environment. For example, when two 8-bit binary numbers are multiplied:

$$\begin{aligned} 11001011 \times 10111001 &= 1001001010110011 \\ &\Rightarrow 10010010, \end{aligned} \tag{1.1}$$

the result is a 16-bit number. In order to store this number in an 8-bit memory location, the 16-bit number must be truncated into an 8-bit number, therefore, accuracy is lost. As the numbers are processed, this error propagates and increases, even in the digital system. It would therefore be desirable to minimize this truncation effect due to finite word lengths.

1.1.1 Motivation

The motivation behind the work described in this thesis is to minimize the effect of the finite word lengths present in digital processing systems. This minimization can be interpreted in two ways. First, if the internally generated noise can be reduced, then the output will have a larger signal to noise ratio and therefore a lower bit error rate (BER). From another perspective, it may be possible to reduce the system's finite word lengths, while keeping the output BER relatively constant (i.e. output BER remains constant with shorter word lengths). This reduction in the number of bits required for the system word length directly affects the size and power requirements for a digital system.

1.1.2 Residue Number Systems

Residue number systems (RNS), which will be defined in Chapter 2, are known to be able to perform simple arithmetic operations at high speeds [1]. Addition, subtraction and multiplication operations are considered simple because their results, when working with integers, are also integers. The objective behind RNS is to break down large numbers into sets of small numbers, called residues, that can be operated upon independently, and simultaneously. The main advantage behind RNS is that they can perform additions and multiplications very easily and quickly, and therefore lend themselves towards digital signal processing, where multiplication and additions are the real backbone. With smaller numbers to operate upon, the multipliers and adders of a RNS can be reduced in size and complexity. If the residues can be made small enough, these operations can be performed using high-speed, low-power ROM look-up tables, or other similar techniques.

The disadvantages of RNS are that they have difficulty performing divisions, scaling and comparisons. Therefore these operations should be avoided if at all

possible. Another disadvantage of RNS is that in order to retain every bit of accuracy, very large dynamic ranges are required to prevent overflow errors. This accumulation of bits has the tendency to increase the system dynamic range which can directly affect its size.

Most of the previous research applying residue number systems involves implementing digital filters and correlators. The main reasons for its use in these applications is because of its high speed operations. The intent of this thesis is to apply residue number system techniques to an entire receiver system, making this research unique.

1.2 Contribution of the Thesis

The purpose for this work is to apply a residue number system to a cross-correlator receiver. A cross-correlator receiver can be used for a variety of modulation techniques including FSK and PSK signalling. The RNS-based cross-correlator's performance will be compared to that of a fixed point DSP-based cross-correlator receiver in order to determine the feasibility of a RNS-based receiver. The receivers will not actually be implemented, but simulated and modelled using Matlab.

The cross-correlator receiver was chosen for the analysis in this thesis because of its modularity and suitability for implementation in DSP systems. Some background and analysis is given in the thesis by Kevin Andrew Farrell [2] entitled "Performance of the Cross-Correlator Receiver for Binary Digital Frequency Modulation".

A theoretical analysis of the RNS-based system is presented along with a similar analysis for a fixed point DSP system to evaluate the bit error rate (BER) at the output of each system for comparison. The focus of this thesis is not just to evaluate the performance of the cross-correlator as a receiver, but rather, to evaluate the

performance of the residue number system in comparison to the common fixed point DSP system. Therefore, the receiver's parametrics, including modulation indicies and frequency deviation constants, have all been kept constant for the entire analysis. Frequency offsets, phase offsets, inter-symbol interference and bit synchronization have also been ignored to simplify analysis.

The performance of the two different receivers have been compared for various A/D converter bit lengths, number of bits used for processing and the number of bits used for the output for various SNR's present at the input of the receivers.

1.3 Thesis Outline

Chapter 2 provides an introduction to residue number systems, including their operations for conversions and arithmetic, as well as methods for implementing these functions. Chapter 3 discusses how an RNS is applied to a system and specifically the application of an RNS to the cross-correlator receiver. Models for the RNS-based receiver and the fixed point based receiver are developed in Chapter 4. These models are used to determine the theoretical performance of the receivers. Chapter 5 shows some of the simulation results obtained for the receivers for various systems at different input SNR's allowing for comparison between the receivers, as well as to the theoretical results. The results are summarised in Chapter 6, along with a conclusion and some suggestions for future work.

Chapter 2

RESIDUE NUMBER SYSTEMS

Residue number systems (RNS) have been investigated for centuries. In the 1950's, these systems were investigated for use in the design of computers because of their ability to perform fast multiplications and additions. The fact that these systems have difficulty with divisions, scaling, sign detection and magnitude comparisons kept these techniques from reaching widespread use [3]. Although these systems still have trouble performing these more complicated operations, their advantages are obviously well suited for digital signal processing (DSP) systems where multiplications and additions dominate.

Published work on Residue Number Systems dates back to as early as the first century A.D. when Sun-Tsu, a Chinese scholar, described a rule to determine a number having the remainders 2, 3 and 2 when divided by the numbers 3, 5 and 7 respectively [1]. Out of interests sake, the answer is 23. Other popular research has been performed by Szabo and Tanaka in 1967, publishing a book on the subject [4], but failing to produce long lasting research because of the difficulty performing the more complicated operations.

Prior to this time, the only real application for RNS was for general purpose

computers. The introduction of digital signal processing aroused the interest of researchers recognizing that this new application required many of the advantages that RNS could provide. That is, most digital signal processing algorithms including correlators, digital filters and fast Fourier transforms which are dominated by large numbers of repetitive addition and multiplication operations. Therefore, residue number systems naturally lend themselves towards this application.

2.1 Introduction to Residue Number Systems

A Residue Number System (RNS) is a system that breaks down numbers into a set of smaller numbers that can be operated upon more quickly and easily. Each number that is converted to the RNS is represented by a set of residues. Since these residues are inherently small, the space, time and power required to process these numbers can be reduced, if the processing of each residue is done simultaneously. RNS operations can be broken down into two groups, those which are simple and those that are very complex. The simple operations which include addition, subtraction and multiplication can be performed easily and quickly. Other operations, including division and scaling, are very difficult and therefore require more time, energy and complexity. Though few fixed point DSP algorithms actually use the division operation, almost all algorithms employ some sort of scaling. That is, most DSP multiplication operations automatically scale the product down, usually by discarding the less significant bits. A system that has no scaling operation requires a large dynamic range so that overflow does not occur.

The other operations that must be considered in an RNS are the conversion processes to and from RNS format. The conversion of numbers from integer or binary to RNS is fairly straightforward while the reverse is somewhat more complicated. The Chinese remainder theorem (CRT) is the technique that is used to

convert the residues back to its appropriate integer or binary number [1]. These conversion processes take time and space as well, and therefore must be included in the analysis of such a system.

2.1.1 Finite Fields and Rings

Digital signal processing systems operate on finite fields because of the finite number of states that a process can enter. Residue number systems make use of finite fields as well, but an RNS can have several finite fields present and operated upon simultaneously.

A set of numbers, $S_P = \{0, 1, 2, \cdots, p-1\}$, where p is a prime number, combined with modulo-p addition and multiplication defines a finite field which is represented as $\{S_P, +, \times\}$ (this is also called a Galois field and is represented as $G(p)$). For every number $s_i \in S_p$; there also exists a multiplicative inverse $s_i^{-1} \in S_p$ such that $(s_i \times s_i^{-1})_{\text{mod p}} = 1$ (except for the number 0). Note that the modulo-p addition and multiplication form closed operations, which means any operation performed on two numbers from the set S_p will produce a number in S_p.

The finite field for $p = 3$ is defined by $\{0, 1, 2, +, \times\}$. A few examples of finite field operations are provided to help explain the finite field concept. The actual theory behind these operations is shown in later sections.

$$1_3 + 2_3 = 3 \Rightarrow 0_3, \qquad 2_3 + 2_3 = 4 \Rightarrow 1_3, \tag{2.1}$$

$$1_3 \times 2_3 = 2 \Rightarrow 2_3, \qquad 2_3 \times 2_3 = 4 \Rightarrow 1_3, \tag{2.2}$$

The addition and multiplication tables for the field S_3 are shown in Figure 2.1, where c' and d' are the modulo 3 sum and product of a and b, respectively.

The inverse of an element can be determined from the multiplication table by

c'		b		
	+	0	1	2
	0	0	1	2
a	1	1	2	0
	2	2	0	1

d'		b		
	x	0	1	2
	0	0	0	0
a	1	0	1	2
	2	0	2	1

Figure 2.1: Addition and Multiplication Tables for Mod 3 Arithmetic

finding the corresponding element that produces a multiplicative result of 1. The actual operations are shown below (note, the element 0 does not have an inverse.)

$$1_3^{-1} = 1_3 \qquad (1_3 \times 1_3 = 1 \Rightarrow 1_3), \tag{2.3}$$

$$2_3^{-1} = 2_3 \qquad (2_3 \times 2_3 = 4 \Rightarrow 1_3). \tag{2.4}$$

Another characteristic of a finite field is the presence of a generator element g. This generator element can generate all of the nonzero elements of that field. If $g \in S_p$ is the generator for the field, then the first $p-1$ powers of g, according to modulo p multiplication, along with zero, constitute the finite field S_p. This generator element can also be used in a technique which implements multiplications using additions, which will be discussed later.

If the modulus is a composite number, rather than prime, then the field is a finite ring, which has a considerably weaker structure than a finite field. The structure is weaker primarily because not all numbers in the field will have an inverse. Finite rings have another disadvantage in that they do not contain a generator element, limiting the number of implementation methods available to perform operations within the field.

The field $F(m)$, which could be either a finite field or finite ring, has a legitimate range $[0, m-1]$. That is, any integer within this range can be exactly represented by this field. Integers outside of this range cannot be uniquely represented. If the

legitimate range is not big enough for the operations being performed, there will be overflow errors. For example, $2_3 + 2_3 = 1_3 \Rightarrow 1 \neq 4$. The number 1 is the result of the field's operation, but the actual desired result is 4. In this field it is not possible to determine if the result 1_3 actually represents 1, or 4, 7, etc. Therefore overflow must not occur, or the results will not be correct.

The legitimate range can be converted from an all positive range into a dynamic range that can represent both positive and negative integers. The legitimate range is divided in half and the upper half is used to represent the negative quantities. Therefore, the dynamic range becomes $[-(m-1)/2, (m-1)/2]$ if m is odd and $[-m/2, (m/2-1)]$ if m is even. The negative region $[-(m-1)/2, -1]$ for odd values of m, maps to the region $[(m+1)/2, m-1]$ while the negative region $[-m/2, -1]$ for even values of m, maps onto $[m/2, m-1]$. This mapping is described pictorially in Figure 2.2 for the even m case.

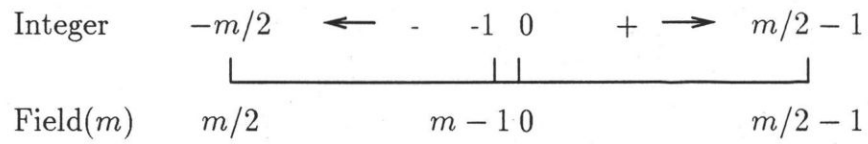

Figure 2.2: Dynamic Range Illustration

2.1.2 Multiple Moduli Residue Number Systems

The main feature of residue number systems is that they can provide a large dynamic range while keeping addition and multiplication operations fast and simple. This is accomplished by breaking a DSP number system into a set of parallel finite fields. This breaks the long word length required by a system with a large dynamic range into many small word length systems with small dynamic ranges. Such a system is correctly called a multiple moduli residue number system (MMRNS), but since most RNS are MMRNS, they are simply referred to as RNS. Figure 2.3

illustrates the basic structuring of an RNS. An L moduli RNS has a legitimate range $[0, M-1]$, where $M = m_1 m_2 \cdots m_L$, and m_i is the modulus for the i*th* parallel system. In an RNS, it is not necessary for every modulus to be prime, as long as they are pairwise prime. That is, no two moduli have any non-unity common factors. If the moduli are not pairwise prime then the residue combinations will no longer be unique.

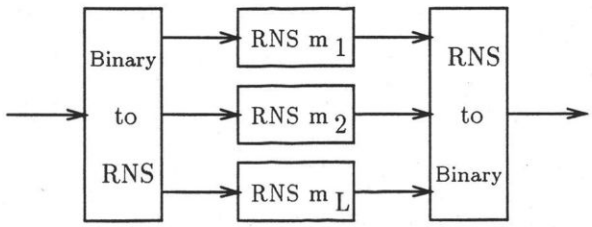

Figure 2.3: Multiple Moduli RNS Block Diagram

2.2 Residue Number System Arithmetic

The first operation that should be discussed is the conversion or encoding process of integer or binary numbers into residue digits. As discussed earlier, any number that is to be represented exactly in residue format, must lie within the dynamic range of the system. Initially, an integer S is converted into modulo m residue format by taking the least positive integer remainder that is left-over when S is divided by m. That is, the residue $\langle S \rangle_m = S - km$, where k is the greatest integer that will produce the remainder $\langle S \rangle_m$ in the range $[0, m-1]$, and it is read S modulo m or S mod m. The notation $\langle S \rangle_m$ will be used throughout this thesis to represent the modulo m residue of S.

For a multiple moduli RNS, the conversion process is simply repeated for each of the individual moduli in the set $\{m_1, m_2, \cdots, m_L\}$ where m_i is a single modulus from the set of L moduli. S is now represented in the multiple moduli RNS as

$\{s_1, s_2, \cdots, s_L\}$ where $s_i = \langle S \rangle_{m_i}$. After conversion to the multiple moduli RNS form, each individual residue can be operated upon separately, and simultaneously, if desired. Following the processing of the individual residues, the multiple moduli residue number can be converted back to integer by using the Chinese remainder theorem or representation thereof, which will be discussed shortly.

2.2.1 Addition and Multiplication

After all the numbers are converted into residue format, operations can be performed on the residues independently. That is, for integers A, B and C where $a_i = \langle A \rangle_{m_i}$, $b_i = \langle B \rangle_{m_i}$ and $c_i = \langle C \rangle_{m_i}$, the operation

$$\{a_1, a_2, \cdots, a_L\} \circ \{b_1, b_2, \cdots, b_L\} = \{c_1, c_2, \cdots, c_L\}, \tag{2.5}$$

where $\circ$ represents either addition, subtraction or multiplication, can be independently implemented using $c_i = \langle a_i \circ b_i \rangle_{m_i}$ and $c_j = \langle a_j \circ b_j \rangle_{m_j}$ for all $i \neq j$. These are the fundamental operations for residue arithmetic.

A simple example can help explain the principles of an RNS system. Let an RNS be defined by the modular set

$$m = \{5, 7\}, \tag{2.6}$$

that is, $m_1 = 5$ and $m_2 = 7$. The legitimate range for this system is $[0, 34]$, since $M - 1 = (5 \times 7) - 1 = 34$. The dynamic range is $[-17, 17]$. The numbers -2 and 6 are represented in this RNS as

$$\begin{aligned} -2 &= \{\langle -2 \rangle_5 = 3, \langle -2 \rangle_7 = 5\} \Rightarrow \{3, 5\} \text{ and} \\ 6 &= \{\langle 6 \rangle_5 = 1, \langle 6 \rangle_7 = 6\} \Rightarrow \{1, 6\}, \end{aligned} \tag{2.7}$$

respectively. Note that -2 maps to 33 within the system range $[0, 34]$, which can be shown by encoding 33,

$$33 = \{\langle 33\rangle_5 = 3, \langle 33\rangle_7 = 5\} = \{3, 5\}. \tag{2.8}$$

The sum $-2 + 6 = 4 = \{4, 4\}$ and the product is $-2 \times 6 = -12 = \{3, 2\}$. These operations are performed in RNS arithmetic as follows:

$$\{3, 5\} + \{1, 6\} = \{\langle 3 + 1\rangle_5 = 4\ ,\ \langle 5 + 6\rangle_7 = 4\} = \{4, 4\}, \tag{2.9}$$

$$\{3, 5\} \times \{1, 6\} = \{\langle 3 \times 1\rangle_5 = 3\ ,\ \langle 5 \times 6\rangle_7 = 2\} = \{3, 2\}. \tag{2.10}$$

Another example can be used to show the overflow effects of an RNS. The numbers 6 and 4 are encoded $\{1, 6\}$ and $\{4, 4\}$ respectively. The product of these numbers, $6 \times 4 = 24$, is outside of the dynamic range $[-17, 17]$. According to residue arithmetic,

$$\{1, 6\} \times \{4, 4\} = \{\langle 1 \times 4\rangle_5 = 4, \langle 6 \times 4\rangle_7 = 3\} = \{4, 3\}, \tag{2.11}$$

similarly,

$$\{\langle -11\rangle_5 = 4, \langle -11\rangle_7 = 3\} = \{4, 3\}. \tag{2.12}$$

Obviously, $24 \neq -11$, therefore care must be taken to ensure that no overflows occur.

As shown in the example above, the sign of numbers does not affect how the operations are performed. The only time the polarity of the numbers matter is when performing the conversion to residue, afterwards, all numbers are treated identically. This makes the operations simple, straightforward and similar.

2.2.2 Division and Scaling

This section to describes the division and scaling operations that are used in traditional DSP algorithms. Again, division, as an algorithm, is not a commonly used operation in DSP techniques, however scaling is used to keep the dynamic range and the word lengths small. It would be useful to include the scaling operation in RNS algorithms, but as described below, it is a very complex operation to incorporate since it does not always provide a simple integer result.

Suppose it is intended to divide a number a by some integer b. In traditional DSP systems, when a number is to be scaled down, it is usually divided by a power of 2 and only the integer portion of the remaining number is retained. This can be expressed numerically by

$$c = \left| \frac{a}{b} \right|, \tag{2.13}$$

where $|\cdot|$ represent the truncation operation. For example, the number 59 would be represented in binary as 0011 1011 (59), where the integer number has been placed in parentheses. If this number is divided by $2^3 = 8$, then the result would be 0000 0111 (7). Note that $\frac{59}{8} = 7.375 \neq 7$. The error resulting from this scaling will be investigated in Chapter 4.

In RNS format, a division by an integer would be numerically illustrated as shown below.

$$c_m = \left\langle \frac{a}{b} \right\rangle_m = \langle ab^{-1} \rangle_m, \tag{2.14}$$

where b^{-1} is the modulo m inverse of b. Note if m is composite, then not all values of b will have an inverse and therefore scaling cannot be performed. If m is prime then the following principles remain true for b and its inverse:

$$b_m^{-1} = \left\langle \frac{1}{b} \right\rangle_m, \tag{2.15}$$

$$\langle bb^{-1}\rangle_m = 1, \tag{2.16}$$

$$b_m^{-1} = 1 \text{ if } b = 1 \text{ and} \tag{2.17}$$

$$b_m^{-1} = \frac{k_b m + 1}{b} \text{ for all } b \neq 0, 1, \tag{2.18}$$

where k_b is some integer constant. First of all, if $b = 1$ then there is no purpose in scaling a since $\frac{a}{1} = a$. Therefore, substituting b^{-1} into Equation 2.14,

$$c_m = \left\langle a\left(\frac{k_b m + 1}{b}\right)\right\rangle_m \tag{2.19}$$

$$= \left\langle \frac{ak_b m + a}{b}\right\rangle_m, \tag{2.20}$$

where $k_b = \frac{bb^{-1}-1}{m}$. Substituting k_b into Equation' 2.19 gives

$$c_m = \left\langle \frac{a}{b}\left(\frac{bb^{-1} - 1}{m}\right)m + \frac{a}{b}\right\rangle_m \tag{2.21}$$

$$= \left\langle \frac{a}{b}(bb^{-1} - 1) + \frac{a}{b}\right\rangle_m. \tag{2.22}$$

The only way that Equation 2.22 can be true is if

$$\frac{a}{b}(bb^{-1} - 1) = 0 \text{ or } k_c m, \tag{2.23}$$

where k_c is some integer constant. Therefore, if Equation 2.23 $= 0$ then

$$\frac{a}{b}(bb^{-1} - 1) = 0 \tag{2.24}$$

$$bb^{-1} = 1 \tag{2.25}$$

$$b = 1, \tag{2.26}$$

which, again, is a meaningless scaling quantity. However, if Equation 2.23 $= k_c m$ then

$$\frac{a}{b}\left(bb^{-1} - 1\right) = k_c m \tag{2.27}$$

$$a\left(b^{-1}-\frac{1}{b}\right) = k_c m \tag{2.28}$$

$$b^{-1}-\frac{1}{b} = \frac{k_c m}{a}. \tag{2.29}$$

Therefore, given a, only some values of b will satisfy Equation 2.29 for any $k_c = 1, 2, 3, \ldots$. This means that the scaling operation is restricted to certain numbers, which is not acceptable since there may be no control over the value of a. This makes it impossible to reliably scale a number down in RNS and is therefore not implemented in this work.

A simple example can illustrate this scaling operation. In a single moduli system, $m = 7$, the number 6 is divided by 2 and 4. Note, $2_7^{-1} = 4$ and $4_7^{-1} = 2$.

$$\begin{aligned}\left\langle\frac{6}{2}\right\rangle_7 &= \left\langle 6\times 2^{-1}\right\rangle_7 \\ &= \langle 6\times 4\rangle_7 \\ &= \langle 24\rangle_7 \\ &= 3_7,\end{aligned} \tag{2.30}$$

which is the correct answer. However,

$$\begin{aligned}\left\langle\frac{6}{4}\right\rangle_7 &= \left\langle 6\times 4^{-1}\right\rangle_7 \\ &= \langle 6\times 2\rangle_7 \\ &= \langle 12\rangle_7 \\ &= 5_7,\end{aligned} \tag{2.31}$$

which is not the correct answer (the correct answer would be 1.5, when truncated would be 1 and rounded would be 2). Equation 2.32 shows that $a = 6$ and $b = 4$

are not compatible for any $k_c = 1, 2, 3, ...$, and therefore cannot be divided.

$$\begin{aligned} b^-1 - \frac{1}{b} &= \frac{k_c m}{a} \\ 2 - \frac{1}{4} &= \frac{k_c \times 7}{6} \\ \frac{7}{4} &= \frac{7}{6} k_c. \end{aligned} \tag{2.32}$$

Therefore, without some interaction between the *independent* residue systems, the division of a number cannot be performed. This limitation could be overcome if interaction is allowed between individual parallel systems, however, the structure becomes complex, and eliminates the independence between systems.

2.2.3 Chinese Remainder Theorem

The final operation to be discussed here is the conversion of the RNS numbers back to integer. This conversion process is known as the Chinese remainder theorem (CRT) [1]. The mathematical formula for the CRT is given by:

$$\langle X \rangle_M = \left\langle \sum_{i=1}^{L} \bar{m}_i \langle \bar{m}_i{}^{-1} \rangle_{m_i} x_i \right\rangle_M, \tag{2.33}$$

where $\bar{m}_i = M/m_i$ and $\bar{m}_i{}^{-1}$ is the modulo m_i inverse of $\bar{m}_i$. An example is presented here to help explain this operation. The residue number $\{3, 2\}$ (integer -12) as defined by the RNS describe in the previous example, $m = \{5, 7\}$, is decoded as follows:

$$\begin{aligned} & M = 5 \times 7 = 35 \\ & \bar{m}_i = M/m_i \qquad \bar{m}_i{}^{-1} = \langle m_i \rangle \\ & \bar{m}_1 = 35/5 = 7 \quad \bar{m}_1{}^{-1} = \langle 7 \rangle_5^{-1} = 2^{-1} = 3, \\ & \bar{m}_2 = 35/7 = 5 \quad \bar{m}_2{}^{-1} = \langle 5 \rangle_7^{-1} = 5^{-1} = 3, \end{aligned} \tag{2.34}$$

$$\bar{m_1}\langle \bar{m_1}^{-1} \rangle_{m_1} x_1 = 7 \times 3 \times 3 = 63, \tag{2.35}$$

$$\bar{m_2}\langle \bar{m_2}^{-1} \rangle_{m_2} x_2 = 5 \times 3 \times 2 = 30, \tag{2.36}$$

$$\begin{aligned} \langle X \rangle_M &= \langle \bar{m_1}\langle \bar{m_1}^{-1} \rangle_{m_1} x_1 + \bar{m_2}\langle \bar{m_2}^{-1} \rangle_{m_2} x_2 \rangle_M \\ \langle X \rangle_{35} &= \langle 63 + 30 \rangle_{35} = \langle 23 \rangle_{35} \Rightarrow -12. \end{aligned} \tag{2.37}$$

2.2.4 Residue Number System Example

The principles of a residue number system can be best illustrated using a complete example, following the process from beginning to end. A digital filter can be used as the system to be implemented because it incorporates multiplications and additions in its structure. A simple second order digital finite impulse response filter is described by

$$y(n) = c_0 x(n) + c_1 x(n-1) \tag{2.38}$$

The coefficients c_0 and c_1 represent the filter coefficients, which depend on the type of filter to be implemented, and the variables $x(n)$ and $x(n-1)$ are two sequential samples from an input waveform. For the purpose of this example, let $c_0 = 54$, $c_1 = -39$, $x(0) = 15$ and $x(-1) = 27$ (note, it may be necessary to scale the coefficients and round off all numbers to make them integers). In order to design a system, it is desired to prevent system overflows. Therefore, a maximum output should be found to determine an appropriate dynamic range. Suppose maximum inputs to the filter are ± 64, then the maximum output would be $54(64) - 39(-64) = 5952$. If such a system is to be broken down into a 4 moduli system then the 4 moduli must provide a dynamic range greater than ± 5952. To get an idea of the size of moduli required for a 4 moduli system, $(2 \times 5952)^{\frac{1}{4}} = 10.4$. Therefore, a system allowing for some expansion could be designed by using the moduli $m = \{11, 13, 15, 16\}$. With this set of moduli, $M = 11 \times 13 \times 15 \times 16 = 34320$, which produces a dynamic

range approximately ± 17160, or 15-bits of word length. The numerical operations for this system are shown below, both in integer format on the left, and in residue format on the right.

The first step is to convert all of the integers to residue format.

$$\begin{array}{ll} c_0 = 54 & c_0 = \{10, 2, 9, 6\} \\ c_0 = -39 & c_1 = \{5, 0, 6, 9\} \\ x(0) = 15 & x(0) = \{4, 2, 0, 15\} \\ x(-1) = 27 & x(-1) = \{5, 1, 12, 11\} \end{array} \tag{2.39}$$

With the residue number system described above, the next step is to perform the filtering operations. The numerical calculations are performed, as shown below in both integer and residue format.

Integer Format

$$y(0) = 54 \times 15 + -39 \times 27 = -243, \tag{2.40}$$

Residue Format

$$c_0 x(0) = \{10, 2, 9, 6\} \times \{4, 2, 0, 15\} = \{7, 4, 0, 10\} \tag{2.41}$$

$$c_1 x(-1) = \{5, 0, 6, 9\} \times \{5, 1, 12, 11\} = \{3, 0, 12, 3\} \tag{2.42}$$

$$y(0) = c_0 x(0) + c_1 x(-1) = \{7, 4, 0, 10\} + \{3, 0, 12, 3\} = \{10, 4, 12, 13\} \tag{2.43}$$

This result can be *checked* by finding the residue's of -243.

$$y(0) = -243 = \{10, 4, 12, 13\} \tag{2.44}$$

Therefore, the RNS result is the same as the integer calculated result, however, the RNS result must still be converted back to a recognizable integer format. The method for converting the residues back to integer format is via the CRT. While this exact method may not be the practical method of implementation for an actual system, it does allow for a mathematical interpretation.

$$\begin{aligned} \bar{m}_1 &= \tfrac{34320}{11} = 3120 \quad \bar{m}_3 = \tfrac{34320}{15} = 2288 \\ \bar{m}_2 &= \tfrac{34320}{13} = 2640 \quad \bar{m}_4 = \tfrac{34320}{16} = 2145 \end{aligned} \tag{2.45}$$

$$\langle \bar{m}_1 \rangle_{11}^{-1} = \langle 7 \rangle_{11}^{-1} = 8 \tag{2.46}$$

$$\langle \bar{m}_2 \rangle_{13}^{-1} = \langle 1 \rangle_{11}^{-1} = 1 \tag{2.47}$$

$$\langle \bar{m}_3 \rangle_{15}^{-1} = \langle 8 \rangle_{11}^{-1} = 2 \tag{2.48}$$

$$\langle \bar{m}_4 \rangle_{16}^{-1} = \langle 1 \rangle_{11}^{-1} = 1 \tag{2.49}$$

$$\begin{aligned} \bar{m}_1 \langle \bar{m}_1 \rangle_{11}^{-1} = 3120 \times 8 \times 10 &= 249600 \\ \bar{m}_2 \langle \bar{m}_2 \rangle_{13}^{-1} = 2640 \times 1 \times 4 &= 10560 \\ \bar{m}_3 \langle \bar{m}_3 \rangle_{15}^{-1} = 2288 \times 2 \times 12 &= 54912 \\ \bar{m}_4 \langle \bar{m}_4 \rangle_{16}^{-1} = 2145 \times 1 \times 13 &= \underline{27885} \\ \textstyle\sum_{i=1}^{4} \bar{m}_i \langle \bar{m}_i \rangle_{m_i}^{-1} &= 342957 \end{aligned} \tag{2.50}$$

$$\langle 342957 \rangle_{34320} = 34077 \tag{2.51}$$

Since this example is based on a *dynamic* range rather that a *legitimate* range,

$$\langle 342957 \rangle_{34320} = -243. \tag{2.52}$$

Thus both methods, the integer calculations and the RNS calculations gave the same results. The RNS procedure seems like a very *roundabout* method of performing a couple of simple integer calculations, however for larger systems with small sized moduli, RNS will become more advantageous. As mentioned earlier, the CRT, itself, would probably not be used as the actual method of conversion from RNS to integer. As shown in the example, the CRT contains many operations on large numbers making this procedure long and computationally intensive and therefore inconvenient to work with. This suggests that alternate methods of conversion should be investigated for simplification in implementation.

2.3 Implementation Concerns with RNS

The first step in analyzing a system that is to be implemented using RNS is to find the dynamic range required for the system. The major influence on the dynamic range is the accuracy required in the numbers being operated on. The accuracy will determine the size/length of the numbers, in terms of bits. For example, if an analog-to-digital (A/D) convertor is used at the input of the system, the word length of the output symbol/number is specified by the A/D convertor (8-bit, 12-bit, 16-bit, etc.). The filter coefficients in digital filters also require a degree of accuracy, which also determines their associated word lengths. The dynamic range of a system describes how large the system is, and provides a qualitative basis to measure system size (chip real-estate) and power requirements. Therefore, the dynamic range of a system should be investigated first to determine the system requirements.

2.3.1 Dynamic Range

The dynamic range can be deduced from the block diagram of the complete system that is to be implemented. The input symbols/numbers word length, from the A/D convertoror other sources, will be multiplied and/or added to other symbols/numbers with their respective word lengths. All of the multiplications and additions throughout the system must be accounted for in order to determine the actual dynamic range. Note that dynamic range implies signed numbers, and therefore, the multiplication of two signed numbers of lengths n and m, produces a number having a word length $n + m - 1$, since only one sign bit is required in the product. The sum of two numbers will have a length that is one bit greater than the length of the larger number. In exponential notation,

$$2^n \times 2^m \Rightarrow 2^{n+m-1} \tag{2.53}$$

$$2^n + 2^m \Rightarrow 2^{p+1} \text{ where } p = \begin{cases} n & \text{if } n > m \\ m & \text{if } m > n \end{cases} . \tag{2.54}$$

After finding the dynamic range required for the system, a final decision can be made to determine whether or not an RNS is suitable for the application. If a system requires a dynamic range of 50 bits or more, an RNS implementation may not provide the best solution. The decision is based upon how many parallel systems can be implemented, the memory, or chip real-estate, required, as well as time and processing power requirements. All these factors can be compared to a fixed point DSP implementation. This decision will also be influenced by the complexity involved in implementing the system operator functions.

2.3.2 Addition and Multiplication

The RNS addition operation is fairly easy to implement. The various methods that will be discussed make use of a 2's-Complement adder or look-up ROM, or a combination of the two. The 2's-Complement adder method simply adds two numbers 2's-Complement style, $a + b = c$. The sum is then compared to the modulus m_i. If the sum is greater than the modulus, $c > m_i$, then the modulus m_i is subtracted from the sum to produce the proper residue sum, $c' = c - m_i$.

A similar method of implementation is to perform regular 2's-Complement addition on the two numbers, $a + b = c$, after which all results are taken to a correction table, where the correct residue sum can be looked up, $c \Rightarrow c'$. Figure 2.4 shows the structure of a 2's-Complement Adder with correction. The correction table can be calculated ahead of time and stored in high speed read only memory (ROM). Therefore, if c' is the output from the correction table, then $c' = c = \langle a + b \rangle_{m_i}$ for $c < m_i$ and $c' = c - m_i = \langle a + b \rangle_{m_i} = c - m_i$ for $c \geq m_i$. These two methods require a few instructions and therefore take time to perform.

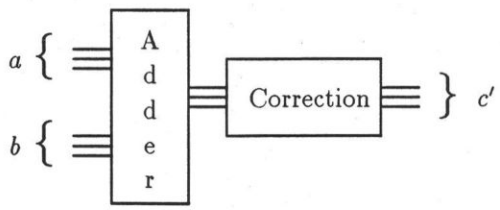

Figure 2.4: 2's-Complement Adder with Look-up Correction Table

Another method of performing the addition is to store the entire addition table in ROM, as shown in Figure 2.5. The numbers being added must be combined such that their combined result addresses the correct element in the look-up table. Obviously, the size of the numbers and therefore, the moduli are restricted by the addressable memory space allocated for the table. The size of the moduli cannot exceed the square-root of the allowed memory space for the table. This method is the simplest and fastest method of addition, requiring only one memory fetch.

The amount of memory required for the table, and its restriction on the moduli are the drawbacks of the look-up table method.

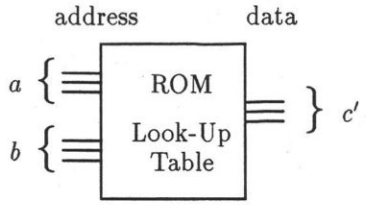

Figure 2.5: Look-Up Table Addition/Multiplication Method

Multiplication can be much more complicated than addition. One method of multiplication is performed by index-addition. As discussed earlier, finite fields contain a generator element. This generator element can generate all of the non-zero elements in the field, multiplicatively. That is, $S_p = \{0, \langle g^{\alpha} \rangle_{m_i}\}$, where g is the generator for the field $F(p) = \{S_p, +, \times\}$ and $\alpha = 0, 1, ..., p-1$ (α is known as the index). Every non-zero element in the field has a corresponding index. The conversion of numbers to their respective indices is similar to taking the *log* of these numbers, where multiplication is performed with an addition after taking the *log* of the numbers of interest. That is, $g^{\alpha_a} g^{\alpha_b} = g^{\alpha_a + \alpha_b} = g^{\alpha_c}$, where α_a is the index for the number a. Therefore, index-addition is performed by converting the numbers of interest to their respective indices, modulo adding these indices and converting the resulting index back to its corresponding residue representation. A block diagram of the Index-Addition Multiplication technique is shown in Figure 2.6.

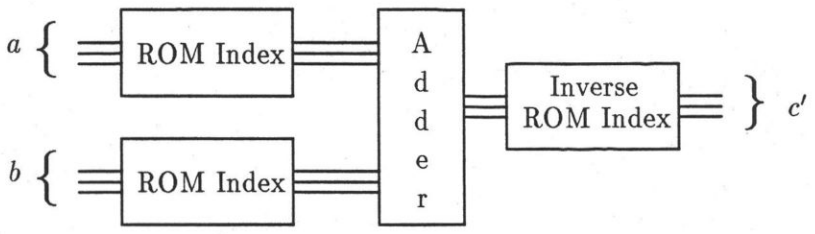

Figure 2.6: Index-Addition Multiplication method

An easier method that may require more power and the use of a 2's-Complement multiplier is similar to the addition-correction method described earlier. A 2's-Complement multiplier multiplies the two numbers as regular 2's-Complement numbers. Following the multiplication, the product is used to address a correction table where the proper mod m_i multiplication results can be stored. Again both of these methods of multiplication require a few instructions and therefore more processing time and power.

Another simple solution that is fast, is to store an entire multiplication look-up table in high-speed ROM, as shown in Figure 2.5, where $\langle a \times b \rangle_{m_i} = d'$. The drawbacks are similar to those of the addition method, where the amount of memory space required for the look-up table is equal to the square of the moduli. Figure 2.1 showed the addition and multiplication look-up tables (shown in integer representation) that would be used for modulo 3 operations. Figure 2.7 shows how the look-up tables could be setup such that a and b can generate the correct address of the result for the modulo 3 addition and multiplication. These tables are displayed in binary format. The least significant bits of the address can be obtained from a, and the most significant bits from b.

c'

	+	00XX	01XX	10XX
	XX00	00	01	10
a	XX01	01	10	00
	XX10	10	00	01

b

d'

	x	00XX	01XX	10XX
	XX00	00	00	00
a	XX01	00	10	10
	XX10	00	10	01

b

X denotes don't care

Figure 2.7: Binary Look-up Tables for Mod 3 Arithmetic

2.3.3 Conversion To and From RNS

Before numbers are processed in residue format, they must be converted from binary to residue. One point that must be noted, the conversion to and from RNS is all overhead. That is, regular DSP implementations do not require any conversion processes, therefore, RNS implementations must compensate for these extra operations required. This conversion can also be implemented in a variety of ways. The fastest and easiest method is to implement a conversion look-up table, where the input binary number can address a memory location which will contain its associated residue number. The number of binary to residue conversion tables required will depend on the number of moduli making up the RNS. This operation simply takes one binary number and converts it into L residue numbers, where L is the number of moduli defining the RNS.

Since a 16-bit A/D converter is usually enough accuracy for most applications, the addressable memory required for this conversion table is easily accessible. The number of bits located at addresses within the table will depend upon the accuracy required as mentioned with the selection of the moduli. More complicated methods can be used that may require less memory, but simply increase the processing time.

The conversion process from RNS back to binary is more difficult to implement, which may be obvious from the demonstration of the CRT in [1]. This process combines the L residues into a single binary number. In small systems, the CRT can be used to generate a look-up table for the conversion of the residues back to integers. In larger systems, with large dynamic ranges, a single table look-up conversion is simply not practical. That is, a system with a N-bit dynamic range would require 2^N addressable locations for the conversion table. For example, a 16-bit dynamic range would require 2^{16} = 65Kbytes/words of ROM, but a system with a 32-bit system would require 2^{32} =4Gbytes/words of ROM. Note, the actual size (bytes/words) of the memories will depend on how much acurracy is desired

at the output. Therefore, a means of simplification is required for this conversion process.

This conversion operation gets more complicated with more moduli. Therefore, if one or more of the residues can be zeroed, then the process can proceed in a reduced output format [5]. Suppose an integer s is encoded into the RNS defined by $m = \{m_1, m_2, m_3\}$. If $s = km_2$, where k is an integer, then $s = \{s_1, 0, s_3\}$. Placing this restraint on s reduces the possible representations of s in the RNS from $m_1 \times m_2 \times m_3$ to $m_1 \times m_3$. In the same manner, if s is any integer in the dynamic range of the RNS, and $s = \{s_1, s_2, s_3\}$, then one of the closest restricted representations can be found by making $s_2 = 0$. This can be accomplished by subtracting s_2 from all the residues. That is, $s' = \{s_1 - s_2, s_2 - s_2, s_3 - s_2\} = \{\bar{s_1}, 0, \bar{s_3}\}$, where $\bar{s_i} = s_i - s_2$ for $i = 1, 3$. Now, the modified integer can be decoded by decoding only the two residues, $\bar{s_1}$ and $\bar{s_3}$. It is possible that this reduced conversion operation can be performed using a table look-up method, since this will require less memory. The CRT can then be used to generate numbers to be stored in the look-up table. The direct output from the 2 residue conversion can be used as the final output as $\{\bar{s_1}, 0, \bar{s_3}\} \Rightarrow s'$, however, if more acurracy is desired, s_2 can be added to s' to provide a more accurate output s. Note that if s_2 is to be added to the output, it must first be converted from RNS in order to distinguish between positive and negative values for s_2.

Some systems may require more than one residue be zeroed. In this case, an integer must be found that can be subtracted from the restricted residues in order that they will all become zero. This number can then be subtracted from the remaining residues, which can then be decoded. A look-up table can be used to find the subtracting integer, as well as for the final conversion process.

Another method of performing the CRT is to follow the exact procedures layed out in the CRT itself, real-time. This method will require finding the inverse of

an integer, two multiplications, L additions (for an L moduli system), and finding the modulo M residue. The multiplication by $\langle \bar{m_1}^{-1} \rangle_{m_i}$ can be integrated into the binary to residue conversion tables, but the multiplication by $\bar{m_i}$, which is a large number with respect to m_i, is difficult. Most of the operations are fairly straightforward, but the modulo M conversion will deal with large numbers, which is better to avoid.

Chapter 3

CROSS-CORRELATOR DESIGN

This chapter explains the design of a cross-correlator receiver used for binary frequency shift keying (FSK) using RNS techniques. The cross-correlator was chosen because of its parallel structure and component modularity. The functional block diagram for the cross-correlator receiver is shown in Figure 3.1. The math behind the function of this receiver will not be too detailed since the aim here is only to provide a brief background of the functionality of the cross-correlator receiver. A more detailed analysis of this specific receiver is given in Farrell's Masters Thesis [2] entitled "Performance of the Cross-Correlator Receiver For Binary Digital Frequency Modulation".

3.1 Receiver Functionality

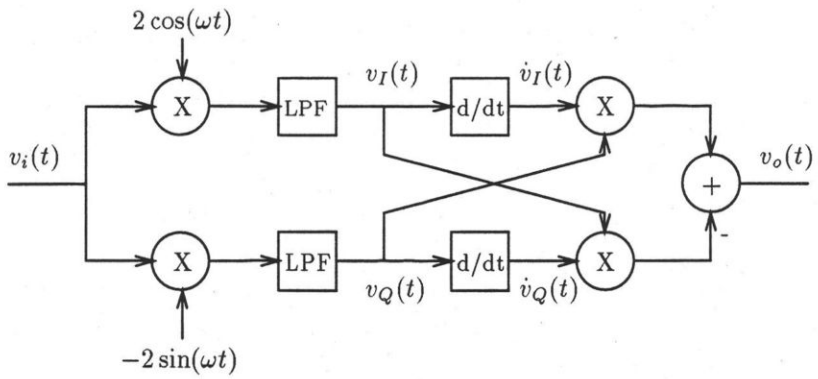

Figure 3.1: Cross-Correlator Functional Block Diagram

The input bandpass signal is mixed to in-phase and quadrature baseband components. The lowpass filters, after the mixers, serve the purpose of removing the sum frequency terms generated by the mixers, while allowing the difference frequency terms to pass. Both of these outputs are then differentiated and multiplied by their opposite undifferentiated signals. The multiplied signals are then summed and the output, as will be shown later, is a signal whose amplitude is proportional to the frequency difference between the input signal and the local oscillator frequency of the mixers. This is the basic functionality of this receiver, the simplified math is shown below to explain these concepts.

Let the input signal be a single sinusoid

$$v_i(t) = V_r \cos(\omega_i t + \theta_i) \tag{3.1}$$

where ω_i and θ_i are the initial frequency and phase of the input signal. (This could be one of the signals in a binary FSK system.) After mixing and lowpass filtering, the in-phase and quadrature signals are

$$v_I(t) = V_r \cos(\omega_c t - \omega_i t - \theta_i) \text{ and} \tag{3.2}$$

$$v_Q(t) = -V_r \sin(\omega_c t - \omega_i t - \theta_i). \tag{3.3}$$

This is assuming that the lowpass filters do not alter the difference frequency terms, but completely remove the sum frequency terms. Letting $\Delta\omega = \omega_c - \omega_i$, the outputs from the differentiators are

$$\dot{v}_I(t) = -V_r \Delta\omega \sin(\Delta\omega t - \theta_i) \text{ and} \tag{3.4}$$

$$\dot{v}_Q(t) = -V_r \Delta\omega \cos(\Delta\omega t - \theta_i). \tag{3.5}$$

Multiplying the corresponding signals provides

$$\dot{v}_I(t) v_Q(t) = V_r^2 \Delta\omega \sin^2(\Delta\omega t - \theta_i) \text{ and} \tag{3.6}$$

$$\dot{v}_Q(t) v_I(t) = -V_r^2 \Delta\omega \cos^2(\Delta\omega t - \theta_i). \tag{3.7}$$

when subtracted from each other, according to Figure 3.1, and using the identity $\cos^2(x) + \sin^2(x) = 1$, the output of the system becomes

$$v_o(t) = V_r^2 \Delta\omega. \tag{3.8}$$

Notice that the initial phase of the input signal has no effect on the receivers' output. This is a characteristic of quadrature receivers.

3.2 The Digital Perspective

Now that the receiver's principles have been presented, the system can be converted into a digital system, and analyzed. The first task is to define where the analog signals can be converted to digital signals. It is important to keep the applications

for this receiver in mind. This receiver could be used for communication systems that transmit through copper, fiber or air. That is, the carrier frequency could be 2KHz or 200GHz, or anywhere in between. The problem with placing an analog-to-digital (A/D) converter at the very input to the receiver is finding an A/D converter with a sampling frequency high enough to sample the signal at its Nyquist frequency [6].

To overcome this requirement for a high-speed A/D converter, some preprocessing circuitry can be added to the receiver's input. This analog processing will contain mixers, filters and an automatic gain control circuit. The mixers are used to bring the input signal at the carrier frequency down to some intermediate frequency that can be sampled at a more convenient sampling frequency. The filtering will eliminate any upper frequency terms generated by the mixers, as well as help to reduce the input noise from the channel. The automatic gain control is used to maintain a relatively constant input signal envelope, in order to allow the desired signal to make use of the full dynamic range of the A/D converter, while not exceeding this range.

While this analog circuitry is functionally important, it is not included when referring to the cross-correlator receiver since only the digital portion of the receiver is of interest.

A complete functional block diagram is shown in Figure 3.2 which includes this analog circuitry. The input signal is brought down to a lower intermediate frequency using a mixer at some frequency f_{hi} ($\omega_{hi} = 2\pi f_{hi}$), allowing an inexpensive (relatively low speed) A/D converter to be placed at the input to the cross-correlator receiver.

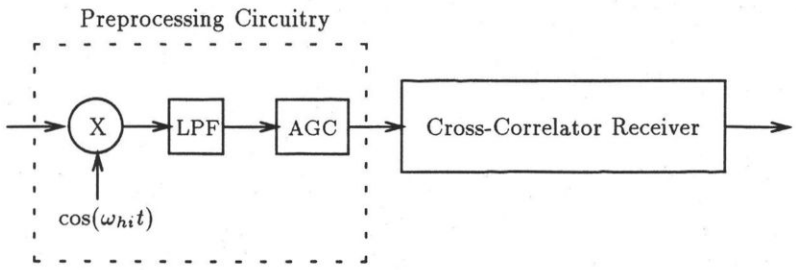

Figure 3.2: Functional Block Diagram of Entire Receiver System

3.3 Residue Number System Design

Before getting into the digital design, the intermediate frequency of the input signal to the A/D converter should be chosen such that the down mixed signal does not interfere with DC, while providing a signal that can be sampled at a reasonable sampling frequency. It is suggested that a sampling frequency be chosen as an integer multiple of the intermediate carrier frequency. This would allow for easier implementation of the numerically generated down mixers.

The quadrature mixers mix the intermediate frequency signal to the in-phase and quadrature baseband components. These mixers can be implemented using a single cycle of each waveform which can be sampled, quantized and stored in memory. This single cycle can then just be cycled through to represent an infinite length waveform. It is therefore desirable to have an A/D converter sampling frequency that is an integer multiple of the intermediate frequency in order that the single cycle of the cos / sin waveform accurately represents its intended waveform.

The differentiators can be implemented using FIR filters. A differentiator determines the slope of an input as a function of time. Since the input is sampled at constant intervals, the time between samples is a known constant. The other parameter that is needed for the slope is the difference in magnitude between adjacent samples. Therefore, the simplest differentiator consists of a subtraction between two adjacent samples and a division by the sampling interval [7]. This will

approximate the derivative of the input signal half way between these two samples. Since this signal is to be multiplied with the original undifferentiated signal, it is desireable to maintain an integer number of sample delays for all signals. This same technique can be spread over three samples rather than two, and the output will approximate the derivative of the signal at the time of the middle sample (divided by twice the sampling interval).

Although the output from the differentiator is not the actual slope of the input until it is divided by the sampling interval, this division can be omitted and performed at the output of the entire system, or just accounted for.

3.3.1 Finding the System Dynamic Range

As described in Chapter 2, the dynamic range must be estimated for this system. Figure 3.3 shows the cross-correlator receiver block diagram, with the A/D converter placed in the circuit. The location of the A/D converter determines where the digital system will begin, and therefore which elements will affect the dynamic range.

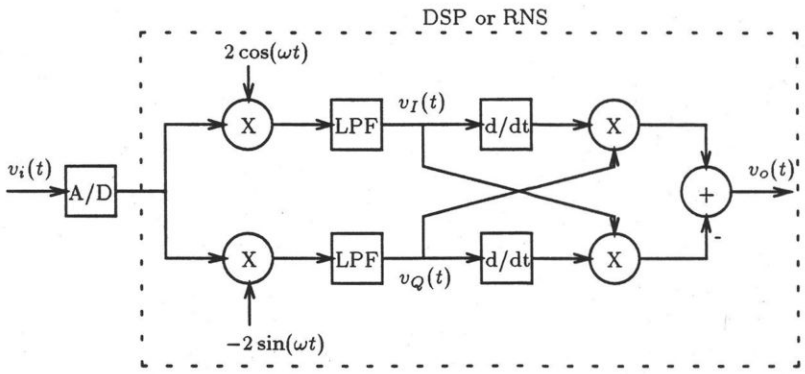

Figure 3.3: Cross-Correlator Analog/Digital Block Diagram

Most of the operations in the system are self-explanatory, except for the low pass filter, which is implemented as a finite impulse response (FIR) filter. Equation

3.9 describes the input-output relationship for an FIR filter.

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k), \tag{3.9}$$

where $x(n)$ and $y(n)$ are the input and output sequences, respectively, and $h(n)$ represents the coefficients for the Nth order filter.

An elementary block diagram can be drawn to show all of the addition and multiplication operations included within the system. Figure 3.4 is the block diagram showing the upper (in-phase) arm of the digital system. This figure shows the system broken down into buffers, multipliers and adders connected by signal flow lines. The signal flow lines in Figure 3.4 are labelled with their corresponding required dynamic ranges, in bits. That is, the A/D converter is a d-bit A/D converter which samples the analog signal and quantizes it into a d-bit binary number. The numbers used in the mixers are scaled and quantized to m-bits and the filter coefficients are scaled and quantized to c-bits. These are the only external numbers that are required to calculate the dynamic range for the system.

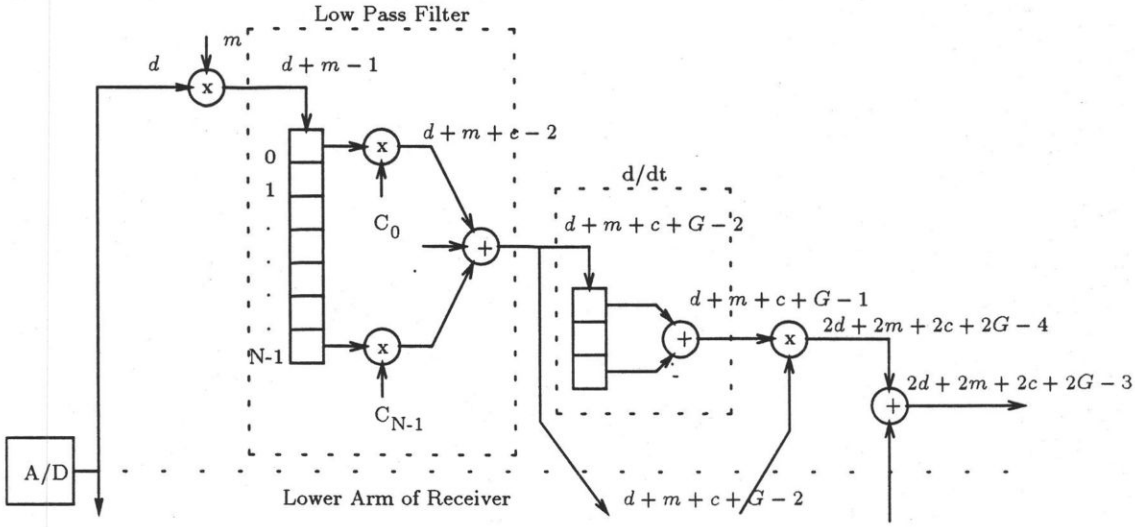

Figure 3.4: Elementary Block Diagram

The addition and multiplication rules, as described in Section 2.3.1, are followed to determine the dynamic range at the output. However, for the lowpass filter,

only the very worst case filter coefficients and inputs signal would justify accounting for all coefficient multiplications and the addition of all N outputs. In order to calculate a more realistic dynamic range, this analysis will consider the scaling of the filter coefficients c and some filter gain constant G used for the filter. Therefore if the pass band gain of the original filter was unity, then the total gain of the filter would be the gain desired from the filter plus the scaling factor required to scale and quantize the coefficients.

This system requires an output dynamic range of $(2d + 2m + 2c + 2G - 3)$-bits as shown in Figure 3.4. Now some calculations can be performed to determine if this system can benefit from RNS techniques. For example, suppose an 8-bit A/D converter is used with mixer and filter coefficients quantized to 8-bits and a filter gain of 1. The systems dynamic range at the output would be

$$2(8 + 8 + 8 + 1) - 3 = 47\text{-bits}. \tag{3.10}$$

This shows the effect of scaling in a regular fixed point DSP processor, where these outputs would usually remain the same order as the input (8-bits).

A 47-bit sytsem is a large system, and this can possibly be decreased by reducing the number of bits used for the A/D converter quantization, mixer coefficient quantization and the filter coefficient quantization. For example if 6-bits were used to represent all numbers the dynamic range at the output would be reduced to 37-bits. These options will be considered in the analysis of the receiver.

3.4 Implementation Considerations

After performing some analysis and simulations, the exact system to be implemented can be chosen. The methods of performing the required operations will depend on the size of the system and its word lengths.

3.4.1 Moduli Selection

To demonstrate the moduli selection process, the 37-bit system will be considered. In order to make up a 37-bit dynamic range system, the RNS moduli must add up to more than 37. If a 4 moduli system is used, $37/4 \approx 10$, whereas if 5 moduli are used, $37/5 = 7.4$, which can be rounded up to 8-bits which is a basic byte and easily adopted for most digital systems. For this reason, a 5 moduli (8-bit), system will be analyzed to provide a 40-bit dynamic range at the output. In this case, there is no advantage to increasing the dynamic range, but if the dynamic range is increased by adding another moduli to the system, the system will have the capability for error detection. Error detection is not of concern right now, so this characteristic will be ignored for the time being.

In order to keep the digital system as an 8-bit system, all of the moduli will be kept equal to, or under 8-bits. The moduli that were chose for this system, ensuring relatively prime moduli, are $m = \{256, 255, 253, 251, 247\}$. Therefore $M = 1023932532480 = 39.90$-bits.

3.4.2 Operations

The simplest method of performing all operations in RNS is to use look-up tables. The numbers being operated upon can be combined to make up an address, to a location in memory which contains the correct RNS result for the operation being performed (addition, multiplication, etc.).

The tables required in this system include conversion tables, addition and multiplication tables. The conversion table from 8-bit 2's-Complement to RNS will require 5 tables (one for each residue), each with 2^8 addresses of 8-bit ROM. The addition and multiplication operations will each require 5 tables with $(2^8)^2 = 2^{16}$ addresses of 8-bit ROM. The conversion table to convert RNS back

to 2's-Complement would require 2^{40} addresses, which is very large, therefore, the output dynamic range is scaled down to $\approx 2^{16}$ by zeroing 3 of the residues, as described in Section 2.3.3. This can be performed using a $(2^8)^3 = 2^{24}$ address 16-bit ROM (8-bits for each residue) look-up table, to find the zeroing residues to subtract from the 2 remaining residues. The remaining modified residues can then be converted back to 2's-Complement using a $(2^8)^2 = 2^{16}$ address 16-bit ROM look-up table. Scaling the output dynamic range from 40-bits to 16-bits may have an effect on the performance of the receiver. Therefore, this scaling should be included in the evaluation and comparison of the receivers' performance.

The addition and multiplication tables must be generated for each of the moduli. Addressing within these tables can be performed by using one of the residues as the lower 8-bits of the address, and the other residue to make up the upper 8-bits of the address. The addition, multiplication and 2's-Complement to RNS conversion tables are all straight forward calculations, while the RNS to 2's-Complement operation is more difficult. However, all tables could be generated using software techniques or applications.

Reducing the dynamic range of the output to 16-bits allows this conversion to be simplified. As noted earlier this conversion process can be performed using 2 moduli look-up tables. The format of the process is shown in Figure 3.5. The first look-up table takes 3 residues and determines what number must be added to each of them in order to make all of them zero. The output of this first look-up will consist of two residues, whose moduli correspond to the two remaining non-zero residues. These two outputs are added to the remaining residues, and their sums are used to look up the final output. Therefore this conversion back to 2's-Complement will require 2^{24} address 16-bit ROM, 2^{16} address 16-bit ROM and the 2 addition tables.

The memory requirements for the entire system will therefore be five 256 $\times$

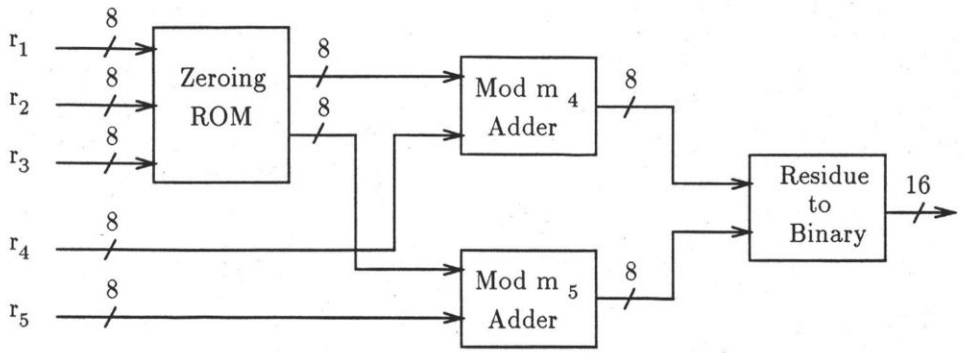

Figure 3.5: RNS to 2's-Complement Conversion Block Diagram

8-bit ROM for the conversion to RNS, ten 64k $\times$ 8-bit ROM for addition and multiplication tables, and one 16M $\times$ 16-bit and one 64k $\times$ 16-bit ROM for conversion back to 2's-Complement, all totalling under 33Mbytes of ROM. Program memory must also be included in the memory requirements. 16k of memory, including program RAM and data RAM should be suitable for most applications where real-time processing is performed.

Chapter 4

THEORETICAL ANALYSIS

It is useful to model communication systems in order to analyze their performance. This section will introduce the mathematical models that are required to analyze the cross-correlator receiver's performance. The objective of this analysis is to determine the performance of an RNS-based cross-correlator receiver, and compare it's performance to a fixed point DSP-based cross-correlator receiver. The main focus of this research is to view the effects of word lengths on the performance of the receiver.

4.1 Noise Injection Models

The key component in this analysis is the noise injection model. Finite word lengths are an inherent part of digital systems and associated with these finite word lengths are rounding and truncation errors. These errors can be also be viewed as noise sources.

4.1.1 Quantization Noise Model

A quantizer is a device that converts an input continuous signal into a finite word length number. In digital systems, the analog-to-digital (A/D) converter is the device that quantizes an input signal voltage into a finite word length binary number. The A/D converters considered in this work convert analog signals into 2's-Complement binary numbers. This is the first component of the digital system and also the first source of noise. All numbers used in a digital system must be quantized to the systems word length, including the input signal to the A/D converter, as well as filter coefficients and any other numbers used for processing. Since all of the other numbers must be quantized prior to processing, only the A/D converter will be discussed in this section.

The A/D converter samples the input waveform and holds that voltage level constant, while the quantizer converts that voltage to an applicable digital form. Figure 4.1 shows a functional block diagram of a typical A/D converter. Since the A/D converter only has a finite number of output levels to represent the continuous range input waveform, error is introduced to the digital approximation. This 'quantization error' is the difference between the analog input waveform and the digital output waveform from the converter. A quantizer input-output characteristic for an eight level uniform quantizer is shown in Figure 4.2.

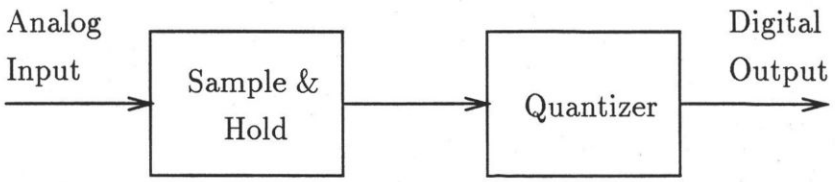

Figure 4.1: A/D Converter Block Diagram

It is important to note that the input analog signal must have prior analog processing in order to ensure that it is within the voltage range of the A/D converter. If an input signal is too small, it should be amplified to make use of the

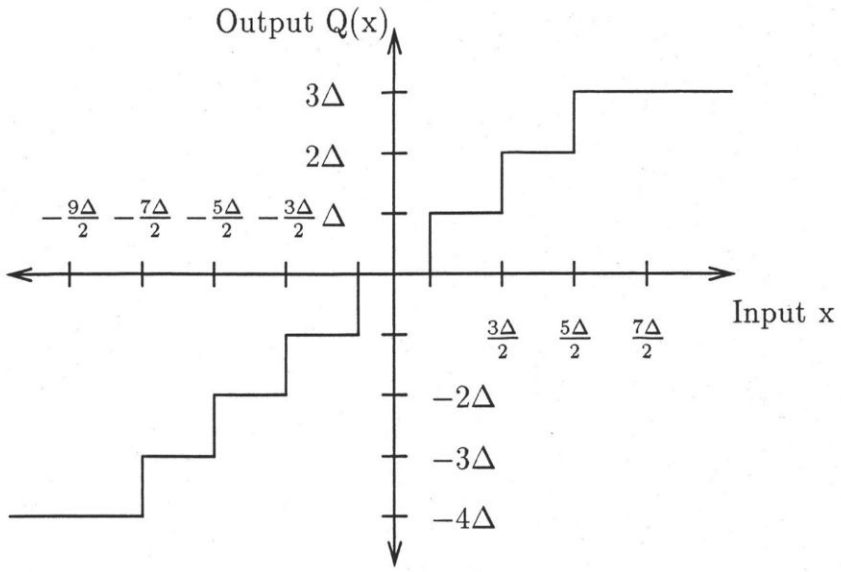

Figure 4.2: An Eight Level Midtread Quantizer

entire quantizer range, while if the input is too large, it must be attenuated or clipped to ensure that the maximum input voltage to the quantizer is not exceeded. A midtread quantizer, as shown in Figure 4.2, is commonly used because of its insensitivity to extremely small fluctuations in the input signal about zero.

The parameters associated with an A/D converter are its step size, which determines its resolution, and its output word length, which along with the step size, determines the input signals dynamic range. The A/D converter's output word length is $b+1$ bits, which designates that the number of quantization levels is 2^{b+1}. (Note: $b+1$ bits are used to represent the output rather than b bits in order to simplify the analysis with the sign bit.) Therefore, 2^b represents the maximum possible positive output from the A/D converter and $-2^b - 1$ is the most negative value output (approximately -2^b). This is only approximate because of the designation of the codewords associated with the A/D converter. That is, there are an even number of codewords to represent the outputs, but the zero codeword is usually used to represent the zero level, leaving an odd number of codewords to be split between the positive and negative regions.

The step size, Δ, and the word length, $b+1$, of the converter are related by the equation

$$\Delta = \frac{R}{2^{b+1}}, \tag{4.1}$$

where R is the total range to be covered by the A/D converter.

It is obvious from the input-output characteristics that the quantization error, $e_q(n)$, at the output of the A/D converter is going to be within the range

$$\frac{\Delta}{2} < e_q(n) \leq \frac{\Delta}{2}, \tag{4.2}$$

assuming that the input signal levels are within the range R.

Figure 4.3 shows a sample input waveform to an A/D converter along with the output from the A/D converter (scaled to portray the voltage level that its output numbers represent). Figure 4.4 shows the discrete distribution of the quantization error resulting from this analog-to-digital conversion. The X-axis shows the quantization error while the Y-axis shows how many times that exact error has been produced.

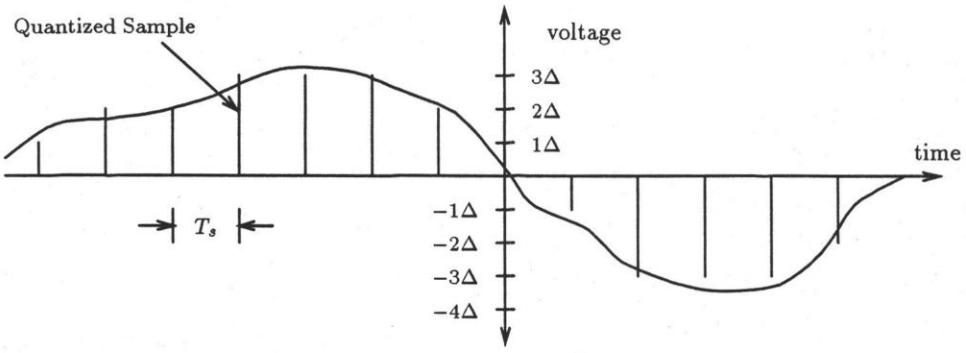

Figure 4.3: Input Waveform and Quantized Digital Output

In a system model, it is convenient to adopt a statistical model for this quantization error. This quantization error can be modelled using a uniformly distributed noise source whose probability density function depends upon the resolution or

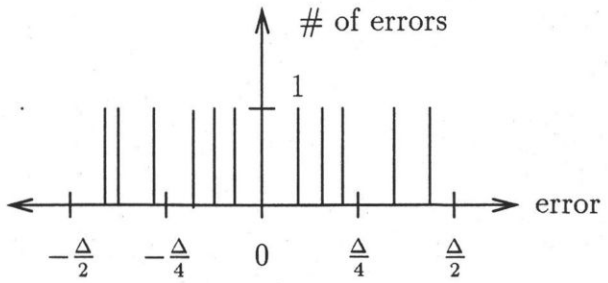

Figure 4.4: Quantization Error

minimum step size, Δ. The probability function for the quantization error, (e), of a uniform quantizer with step size Δ and word length $b+1$ is shown in Figure 4.5. $p(e)$ is the probability that the quantizer will produce an error of e magnitude. The upper and lower limits for the error are at $\pm\frac{\Delta}{2}$, and in order for the total area under the curve to be 1, the density function must have a level of $\frac{1}{\Delta}$.

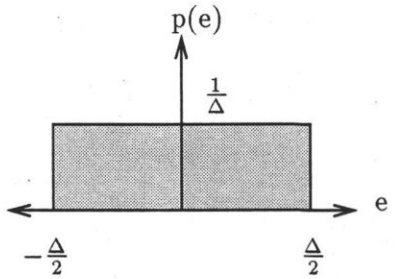

Figure 4.5: Probability Density Function of Quantization Noise

The mean of the quantization noise is intuitively 0, as the distribution is uniform from $-\frac{\Delta}{2}$ to $\frac{\Delta}{2}$. As shown on pages 414-419 in [8], the noise power, P_e, is equal to the variance σ_e^2, of the noise (since the mean value of the error is 0). That is,

$$\begin{aligned} P_e = \sigma_e^2 &= \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} e^2 p(e) de \qquad (4.3) \\ &= \frac{e^3}{3}\frac{1}{\Delta}\Big|_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \\ &= \frac{1}{3\Delta}\left(\frac{\Delta^3}{8} + \frac{\Delta^3}{8}\right) \\ &= \frac{\Delta^2}{12} \end{aligned}$$

The quantization noise power can be calculated in terms of the number of bits used. For a $(b+1)$ bit quantizer (i.e. $\Delta = 2^{-(b+1)}$) for $R = 1$, the quantization noise power is

$$\begin{aligned} \sigma_e^2 &= \frac{2^{-2(b+1)}}{12} \qquad (4.4) \\ &= \frac{2^{-2b}}{48} \\ \sigma_e^2(dB) &= -20b\log(2) - 10log(48) \qquad (4.5) \\ &= (-6.02b - 16.81)dB. \end{aligned}$$

The model for the quantization noise source, as shown in Figure 4.6, consists of a noise source with a uniform probability density function that injects noise to the output of the sample and hold block. This model will not provide the exact results that would be seen in an actual system, however, the statistics will be the same.

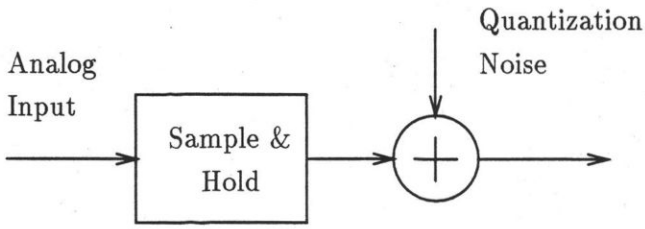

Figure 4.6: Quantization Noise Model

4.1.2 Truncation Noise Model

The other source of error, or noise, with finite word lengths is associated with the truncation or scaling of numbers. In a digital system, as numbers are added or multiplied together, the result tends to be larger than the two original numbers. This problem is associated more with fixed point systems rather than floating point systems. As numbers grow in a floating point system, the exponent of the number

is increased, where fixed point systems do not have this exponent. Therefore, as numbers grow, they must be scaled down to avoid overflows. The common method of scaling in these processors is to simply shift the bits in the output word to the right. In a way, this shifts the decimal point, but for every bit that the decimal point moves, another bit of accuracy is dropped. This is referred to as a truncation, the output word is truncated and the truncated bits are lost. An example can help illustrate the effect of scaling and truncation on integers. Suppose the integer 91, represented in an 8-bit system, is to be scaled down by 4 in order to avoid possible overflow in proceeding calculations. (Note: 91 can be represented as integer binary as 01011011 or as a fraction $91_8 = \frac{91}{128} = 0.1011011$, where $2^{8-1} = 128$.)

$$\begin{aligned} 91 &= 01011011 \\ 91_8 &= 0.1011011 \\ 91_8/4 &= 0.001011011 \\ \|91_8/4\| &= 0.0010110 \quad = 21_8, \end{aligned}$$

where $\|\cdot\|$ represents the truncation operation. Note $91/4 = 22.75$ while $91_8/4_8 = 21_8$, showing the effect of truncation on the accurracy of the scaled integers.

This loss of bits obviously creates some loss of accuracy in the result and a deviation from the desired output. This error can be modelled with a noise injector similar to the quantization noise model, however, the statistics are not identical. It is important to understand the word format of the digital system to identify the parameters or statistics associated with the truncation effects.

Most digital systems use 2's-Complement format for their numbers because of the similarity between the operations for positive and negative numbers. 2's-Complement format uses the most significant bit (MSB) as a sign bit. That is, a MSB of 0 represents a positive number and a 1 represents a negative number. A integer number is converted to 2's-Complement binary by first converting the

magnitude into binary format. For positive numbers, the 2's-Complement format is the same as the binary format, assuming that the magnitude is not conflicting with the sign bit. The negative version of that magnitude is found by taking the complement of every bit in the positive magnitude format and then adding 1. For example, in an 8-bit system, the decimal numbers $+\frac{43}{128}$ and $-\frac{43}{128}$ are converted to 2's-Complement format as shown below.

$$
\begin{aligned}
43 &\Rightarrow 0010\ 1011 \\
43_8 &\Rightarrow 0.010\ 1011 \qquad\qquad\qquad\qquad \Rightarrow 0.010\ 1011 \\
-43_8 &\Rightarrow 0.010\ 1011 \Rightarrow 1.101\ 0100 + 0.000\ 0001 \Rightarrow 1.101\ 0101
\end{aligned}
$$

The conversion from 2's-Complement format to decimal is done by simply performing these steps in reverse. That is, for negative numbers, subtract 1 from the LSB and then take the complement of the result to produce the magnitude representation of that number. Now that the format of these numbers is known, they can be observed as they are operated upon.

Truncation errors are associated with operations on finite length words. The most dramatic effect of this is seen by observing the fixed point multiplication instruction. Suppose the two numbers $+43_8$ and $+58_8$ are multiplied together. The product of these two 8-bit numbers is a 15-bit number as can be seen below.

$$
\begin{aligned}
43_8 &\Rightarrow 0.010\ 1011 \\
\times \underline{58_8} &\Rightarrow \times\ \underline{0.011\ 1010} \\
2494_{15} &\qquad 0.00\ 1001\ 1011\ 1110 \\
\|2494_{15}\| &\qquad 0.00\ 10011 \\
2494_{15} > 2432_{15} &\Leftarrow 000.0\ 1001\ 1000\ 0000
\end{aligned}
$$

Suppose the result was negative,

$$-2494_{15} \quad \Rightarrow \quad 1.11\ 0110\ 0100\ 0010$$

$$\|-2494_{15}\| \quad\quad\quad 1.11\ 0110\ 0$$

$$-2494_{15} \ > \ -2560_{15} \ \Leftarrow \ 1.11\ 0110\ 0000\ 0000$$

Looking at the truncation of these numbers, it is obvious that performing the truncation on 2's-Complement numbers will result in a number less than the original result (closer to $-\infty$), whether positive or negative.

The probability density function for the truncation error is a discrete density function since the error is restricted to only the numbers that can be represented by the longer untruncated number. The truncation error can range from the truncation of all zero bits, or 0, to the truncation of all ones, whose magnitude depends on the length of the truncated output and the number of bits truncated. If b_T bits are truncated from a number to result in a $b+1$ bit number (including the sign bit), then the truncation error can range from 0 to $-(2^{-b+1} - 2^{-(b+b_T+1)})$. The magnitude of the errors occur at discrete intervals of $2^{-(b+b_T+1)}$, which is the decimal equivalent value of the least significant bit of the truncated number. Therefore the probability density function for the truncation error can be plotted as shown in Figure 4.7.

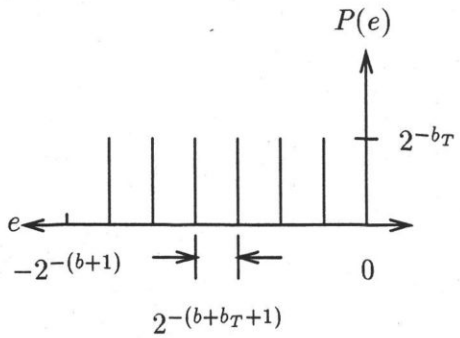

Figure 4.7: Truncation Noise Probability Density Function

The mean of this truncation error, n_{Tx}, is

$$n_{Tx} = -\frac{2^{-(b+1)} - 2^{-(b+b_T+1)}}{2} \tag{4.6}$$

$$\approx -2^{-(b+2)}. \tag{4.7}$$

The variance, σ^2_{Tx}, is

$$\sigma^2_{Tx} = 2^{-b_T} \sum_{n=0}^{2^{b_T}-1} (-2^{-(b+b_T+1)}n)^2 - n^2_{Tx} \tag{4.8}$$

$$\approx \frac{2^{-2(b+1)}}{12}. \tag{4.9}$$

This truncation error can also be modelled similar to the quantization error model. Instead of actually performing the truncations, this error can be modelled by injecting noise at the output of the multiplier or adder, or wherever the truncation would have occurred. The block diagram is shown in Figure 4.8.

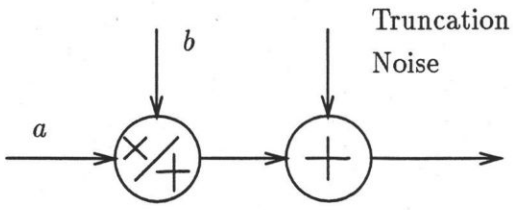

Figure 4.8: Truncation Noise Model

These are the only parts required to model the complete system of interest in this application. These noise injector models can be placed throughout the system model to demonstrate the effect of both quantization and truncation noise.

4.2 Cross-Correlator Receiver System Model and Analysis

This section will derive the statistical models for both a traditional DSP-based receiver as well as an RNS-based receiver. The results of the statistical analysis will then be used to compare the two receiver models.

The cross-correlator receiver model used in this analysis is shown in Figure 4.9. This single model is used to derive the signal and noise components at the output of the receiver for both the traditional DSP-based receiver as well as for the RNS-based receiver. In the analysis for the RNS-based receiver, the truncation noise injectors, denoted $n_{Txx}(n)$, are injecting zero noise, with the exception of the output noise injector, $n_{TO}(n)$. Therefore, the derivation must be kept general, and must retain all parameters in order that the derivation be adaptable to both the DSP and RNS-based systems.

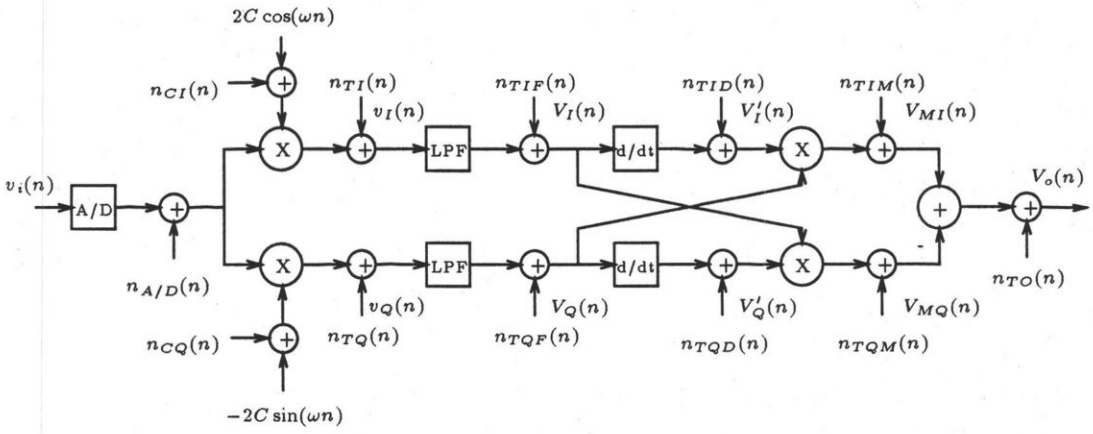

Figure 4.9: System Model for a Traditional DSP-Based Receiver

4.2.1 System Components

In order to fully analyze this system, each individual block must be broken down into a possible implementation format for digital systems. In a logical order, the system in Figure 4.9 will be broken apart from the input to the output.

It is assumed that the system includes some analog processing prior to the A/D converter that performs down conversion to intermediate frequencies, filtering, automatic gain control and clipping functionality. This will provide the appropriate input signal characteristics for the receivers to operate in an optimal state, utilizing the majority of their dynamic ranges. The A/D converter is modelled with an impulse sampler and a quantizer. The analysis will begin after the impulse sampler, since this sampler will be present in both methods of implementation. Therefore, the input to the system is a string of unquantized numbers or samples, $v_i(n)$. The quantizer quantizes the input samples into fixed length numbers that can be processed digitally. The bit lengths of these numbers determines the step size, Δ, for the A/D converter quantizer and the quantization noise, $n_{A/D}(n)$.

The first set of multipliers, on the left side of Figure 4.9 which act like down converters, multiply the input signal by numerically generated cos and sin waveforms. Note, the upper arm of the receiver (cos side) is considered the in-phase side (I) and the lower arm (sin side) is considered the quadrature-phase (Q). The quantization effects are also present when quantizing the sin and cos waveforms, as modelled with $n_{CI}(n)$ and $n_{CQ}(n)$ respectively. A single cycle of each waveform can be sampled, quantized and stored in memory. This single cycle can then be recycled through to represent an infinite length waveform. Therefore, it is desirable to have a sampling frequency that is an integer multiple of the center frequency in order that the cos / sin waveform accurately represent its intended waveform. The mixers are followed by truncations and the associated truncation noise error modelled by $n_{TI}(n)$ and $n_{TQ}(n)$. Note, for an RNS-based receiver, $n_{TI}(n)$ and $n_{TQ}(n)$,

along with all the following truncation noise sources are zero, since RNS-based receivers do not require any truncations.

The next blocks are the low pass filters. These filters are implemented using finite impulse response (FIR) filters as described in Chapter 3. The coefficients for these filters must be quantized, which will result in some error in the filter. This quantization can be regarded as the movement of the poles and zeros of the filter which mildly alters the frequency response of the filter, depending on the degree of quantization [9].

Coefficient quantization has an effect on the magnitude response of the filter. The linear phase response of the FIR filter is not severely affected so it is still assumed linear. In order to compensate for the loss of accuracy in the individual coefficients, the filter order can be increased. Since the overall frequency response remains relatively unchanged, for the purposes of this analysis, the low pass filter will be assumed ideal. That is,

$$H(f) = \begin{cases} 1 & f \leq f_{lpf} \\ 0 & f > f_{lpf} \end{cases} \tag{4.10}$$

It is also assumed that the fixed point DSP has a double length accumulator to allow for double length numbers to be accumulated. This allows for the filtering function to keep double precision throughout its operation until a truncation at its output, modelled by $n_{TIF}(n)$ and $n_{TQF}(n)$.

The differentiators are implemented using the simple difference equation

$$V_{diff}(n) = V(n+1) - V(n-1).$$

This differentiator will approximate the derivative of the input signal $V(n)$ at the midpoint of the two samples used in the difference equation, or at sample n. Although the output from the differentiator is not the actual slope of the input

until it is divided by twice the sampling interval, this division will be omitted and accounted for at the output of the entire system.

The differentiator is simply the subtraction of two samples, but the possibility does arise that the result may exceed the range that can be stored by the system. Therefore, the output will be truncated to ensure that overflows do not occur, modelled with $n_{TID}(n)$ and $n_{TQD}(n)$.

The remaining blocks include two multipliers and an adder. Both of these operations are elementary and consist of the multiplication or addition operation followed by a truncation. The truncations after the multipliers are modelled by $n_{TIM}(n)$ and $n_{TQM}(n)$. The output of the adder is the final output from the receiver, therefore the truncation that is performed here consists of both the regular addition operation truncation as well as the final output truncation to provide the appropriate output word sizes, modelled with $n_{TO}(n)$.

Now that the basic functionality of the receiver has been presented, and the sources of the quantization and truncation noise have been introduced, the statistical analysis of the cross-correlator receiver can be performed.

4.3 Statistical Analysis

The analysis of the receiver begins by looking at the input signal received from the channel. However, since the focus of this work is to analyze the implementation methods of the digital portion of the receiver, the signal present at the input to the A/D converter will be considered as the input to the receiver. As discussed in Chapter 3, the preprocessing circuitry will perform some down conversion and filtering on the input waveform, but it is assumed that this processing will have a negligible effect on the input desired signal. This filtering will reduce the amount of channel noise entering the receiver.

The input signal to the receiver is

$$v_i(t) = A_c \cos(2\pi f_i t + \theta_i) + n_i(t), \tag{4.11}$$

where A_c is the peak amplitude of the desired signal, f_i is the frequency of the input signal, θ_i is the initial phase offset for the signal, and n_i is the channel noise present at the input to the receiver. It is assumed that the power spectral density for the input noise, n_i, to the receiver remains constant over the low pass filtered spectrum from $-\frac{f_s}{2}$ to $+\frac{f_s}{2}$, where f_s is the A/D converter sampling frequency, with a double sided power spectral density of $N_o/2$ W/Hz. Therefore, the input noise power to the receiver is $N_o f_s/2$. The input signal to the A/D converter is assumed to have a relatively constant amplitude with a peak magnitude of A_c Volts. It is desirable to look at the relationship between the input signal power and the input noise power to the receiver, or the signal to noise ratio (SNR).

$$SNR = \frac{A_c^2/2}{N_o f_s/2} \tag{4.12}$$

In order to perform statistical analysis on the receiver, the mean and standard deviation for this input noise is required. It is assumed that the mean for the input noise is 0, however, the standard deviation is determined by the input AC power of the noise. It can be shown that the variance, σ^2 = AC Power (and Total Power since the DC Power = 0), or $\sigma^2 = N_o f_s/2$ as described in [10].

Another method of evaluating the SNR is to look at the energy per bit, E_b, with respect to the power spectral density of the noise, N_o. The energy per bit can be derived from the modulated format as,

$$E_b = \frac{A_c^2}{2}\frac{1}{R_b}, \tag{4.13}$$

where A_c is the input signal's peak amplitude and R_b is the bit rate. Once the input is sampled, E_b must be evaluated with respect to the sampling frequency.

Suppose the sampling frequency is an integer multiple of the bit rate,

$$M = \frac{f_s}{R_b}. \tag{4.14}$$

Then the energy per bit can be evaluated as

$$E_b = \frac{A_c^2}{2}\frac{M}{f_s}. \tag{4.15}$$

Similarly, the input noise power spectral density, N_o, is related to the noise variance, σ^2, as

$$N_o = 2\frac{\sigma^2}{f_s}. \tag{4.16}$$

Therefore, the input SNR per bit, as denoted E_b/N_o is given as

$$SNR_b = \frac{E_b}{N_o} = \frac{A_c^2 M}{4\sigma^2}. \tag{4.17}$$

The objective of this analysis is to determine the output bit error rate (BER) for various input SNR_b's (E_b/N_o) for both the traditional fixed point DSP-based receiver, and the RNS-based receiver designs.

Now that the noise principle has been explained, the derivation of the output signal from the cross-correlator receiver can be performed including all quantization and truncation noise components.

As discussed earlier, the input to the receiver is

$$v_i(t) = A_c \cos(2\pi f_i t + \theta_i) + n_i(t), \tag{4.18}$$

where $n_i(t)$ is the channel noise present at the input to the receiver. After sampling the input signal at a sampling frequency f_s and quantizing the samples, the input signal in discrete representation is

$$v_i(n) = A_c \cos(2\pi \frac{f_i}{f_s} n + \theta_i) + n_i(n) + n_{A/D}(n). \tag{4.19}$$

Note, a continuous time waveform, $x(t)$ sampled at sampling intervals $T_s = \frac{1}{f_s}$ can be represented in discrete representation, $x(n)$. That is, $x(n) = x(t)$ at $t = T_s n (n = 0, 1, 2, ...)$. In discrete domain, everything is represented with respect to f_s, therefore f_s is normalized to 1, and omitted from equations where possible.

The noise components can be combined into a single variable $n_i'(n)$,

$$n_i'(n) = n_i(n) + n_{A/D}(n). \tag{4.20}$$

This signal is then multiplied by the two quadrature mixers, which also contain some quantization noise. After this operation, the output signal could be truncated, and therefore truncation noise is added. To refrain from repetition, only the in-phase component, upper branch in Figure 4.9 is shown here.

$$\begin{aligned} v_I(n) &= \left[A_c \cos(2\pi \frac{f_i}{f_s} n + \theta_i) + n_i'(n)\right]\left[2C \cos(2\pi \frac{f_c}{f_s} n) + n_{cI}(n)\right] \\ &= A_c C \left[\cos(2\pi \frac{f_i - f_c}{f_s} n + \theta_i) + \cos(2\pi \frac{f_i + f_c}{f_s} n + \theta_i)\right] \\ &\quad + A_c \cos(2\pi \frac{f_i}{f_s} n + \theta_i) n_{cI}(n) + 2C \cos(2\pi \frac{f_c}{f_s} n) n_i'(n) \\ &\quad + n_i'(n) n_{cI}(n) + n_{TI}(n). \end{aligned} \tag{4.21}$$

After downmixing, the resulting signals, $v_I(n)$ and $v_Q(n)$, are lowpass filtered using FIR filters. It is understood that FIR filters are not ideal in the least, however, for the means of this analysis, these filters will be assumed ideal. The filters will be considered ideal in the sense that they provide unity gain throughout the passband (DC to f_{clpf}) and zero gain for out-of-band signals. In reality this is impossible, due to the quantization of the filter coefficients and order of filters achievable, however for the purpose of comparing two similar digital implementations of this filter, there should be negligible difference.

These assumptions allow the following conclusions to be made:

1. The difference signal (below f_{clpf}) will be passed unchanged.

2. The sum signal (above f_{clpf}) will be eliminated completely.

3. The noise power exiting the filter will be equal to the noise power entering the filter multiplied by $\frac{f_{clpf}}{f_s}$.

4. The filtering function contains a double length accumulator requiring truncation only at the end of the filtering operation.

Note: the $\overline{\text{overlined}}$ terms are components that have been low pass filtered. The inphase component at the output of the filter is

$$\begin{aligned} V_I(n) &= \overline{A_c C \left\{ \cos(2\pi \frac{(f_i - f_c)}{f_s} n + \theta_i) \right\}} \\ &\quad \overline{+2C \cos(2\pi \frac{f_c}{f_s} n) n_i'(n)} + \overline{n_{cI}(n) A_c \cos(2\pi \frac{f_i}{f_s} n + \theta_i)} \\ &\quad + \overline{n_{cI}(n) n_i'(n)} + \overline{n_{TI}(n)} + n_{TIF}(n). \end{aligned} \tag{4.22}$$

Simplifying, combining and letting $\Delta\omega = 2\pi \frac{(f_i - f_c)}{f_s}$ results in

$$\begin{aligned} V_I(n) &= A_c C \{\cos(\Delta\omega n + \theta_i)\} + \overline{2C \cos(2\pi \frac{f_c}{f_s} n) n_i'(n)} \\ &\quad \overline{+n_{cI}(n) A_c \cos(2\pi \frac{f_i}{f_s} n + \theta_i)} + \overline{n_{cI}(n) n_i'(n)} \\ &\quad + \overline{n_{TI}(n)} + n_{TIF}(n). \end{aligned} \tag{4.23}$$

By letting n_I equal the noise component of the inphase signal, the result is

$$V_I(n) = A_c C \cos(\Delta\omega n + \theta_i) + n_I(n), \tag{4.24}$$

where

$$n_I(n) = \overline{2C \cos(2\pi \frac{f_c}{f_s} n) n_i'(n)} \tag{4.25}$$

$$+n_{cI}(n)A_c\cos(2\pi\frac{f_i}{f_s}n+\theta_i)+\overline{n_{cI}(n)n_i'(n)}$$
$$+\overline{n_{TI}(n)}+n_{TIF}(n).$$

Similarily,

$$V_Q(n)=-A_cC\sin(\Delta\omega n+\theta_i)+n_Q(n), \tag{4.26}$$

where

$$\begin{aligned} n_Q(n) &= \overline{2C\sin(2\pi\frac{f_c}{f_s}n)(n_i(n)+n_{A/D}(n))} \\ &\quad +\overline{n_{cQ}(n)A_c\cos(2\pi\frac{f_i}{f_s}n+\theta_i)}+\overline{n_{cQ}'(n)(n_i(n)+n_{A/D}(n))} \\ &\quad +\overline{n_{TQ}(n)}+n_{TQF}(n). \end{aligned} \tag{4.27}$$

Differentiation of these terms is performed using the simple difference equation,

$$V_I'(n)=V_I(n+1)-V_I(n-1), \tag{4.28}$$

shown here for the in-phase branch. Substituting equation (4.24) into equation 4.28) and adding the truncation noise produces

$$\begin{aligned} V_I'(n) &= \{A_cC\cos(\Delta\omega(n+1)+\theta_i)+n_I(n+1)\} \\ &\quad -\{A_cC\cos(\Delta\omega(n-1)+\theta_i)+n_I(n-1)\}+n_{TID}(n). \end{aligned} \tag{4.29}$$

Using the identities

$$\cos(x-y)-cos(x+y)=2\sin(x)\sin(y) \tag{4.30}$$

and

$$\sin(x-y)-sin(x+y)=-2\cos(x)\sin(y), \tag{4.31}$$

$V_I\prime(n)$ is transformed to

$$\begin{aligned} V_I\prime(n) &= -2A_cC\sin(\Delta\omega n+\theta_i)\sin(\Delta\omega)+n_I(n+1)-n_I(n-1) \\ &\quad +n_{TID}(n). \end{aligned} \tag{4.32}$$

Similarily,

$$\begin{aligned} V_Q\prime(n) &= -2A_cC\cos(\Delta\omega n+\theta_i)\sin(\Delta\omega)+n_Q(n+1)-n_Q(n-1) \\ &\quad +n_{TQD}(n). \end{aligned} \tag{4.33}$$

Substituting G in for the gain of the differentiator, where

$$G = 2\sin(\Delta\omega). \tag{4.34}$$

results in

$$V_I\prime(n) = -A_cCG\sin(\Delta\omega n+\theta_i)+n_I(n+1)-n_I(n-1)+n_{TID}(n) \tag{4.35}$$

and

$$V_Q\prime(n) = -A_cCG\cos(\Delta\omega n+\theta_i)+n_Q(n+1)-n_Q(n-1)+n_{TQD}(n). \tag{4.36}$$

By multiplying the in-phase differentiated signal, equation (4.35) and the quadrature-phase undifferentiated signal, equation (4.26) and then truncating the result gives

$$\begin{aligned} V_{MI}(n) &= V_I\prime(n)V_Q(n) \\ &= \{-A_cCG\sin(\Delta\omega n+\theta_i)+n_I(n+1)-n_I(n-1)+n_{TID}(n)\} \\ &\quad \times\{-A_cC\sin(\Delta\omega n+\theta_i)+n_Q(n)\}+n_{TIM}(n) \\ &= A_c^2C^2G\sin^2(\Delta\omega n+\theta_i)+\{-A_cC\sin(\Delta\omega n+\theta_i)+n_Q(n)\} \end{aligned} \tag{4.37}$$

$$\{n_I(n+1) - n_I(n-1) + n_{TID}(n)\}$$
$$+n_Q(n)\{-A_cCG\sin(\Delta\omega n + \theta_i)\} + n_{TIM}(n).$$

Similarily, for the quadrature-phase branch, using equations (4.36) and (4.24)

$$\begin{aligned} V_{MQ}(n) &= V_Q\prime(n)V_I(n) \qquad (4.38)\\ &= -A_c^2C^2G\cos^2(\Delta\omega n + \theta_i) + \{A_cC\cos(\Delta\omega n + \theta_i) + n_I(n)\} \times \\ &\quad \{n_Q(n+1) - n_Q(n-1) + n_{TQD}(n)\} \\ &\quad +n_I(n)\{-A_cCG\cos(\Delta\omega n + \theta_i)\} + n_{TQM}(n). \end{aligned}$$

Therefore the output of the receiver is just the subtraction

$$\begin{aligned} V_O(n) &= V_{MI}(n) - V_{MQ}(n) \qquad (4.39)\\ &= A_c^2C^2G(\sin^2(\Delta\omega n + \theta_i) + \cos^2(\Delta\omega n + \theta_i)) \\ &\quad + \{-A_cC\sin(\Delta\omega n + \theta_i) + n_Q(n)\} \times \\ &\quad \{n_I(n+1) - n_I(n-1) + n_{TID}(n)\} \\ &\quad - \{A_cC\cos(\Delta\omega n + \theta_i) + n_I(n)\} \times \\ &\quad \{n_Q(n+1) - n_Q(n-1) + n_{TQD}(n)\} \\ &\quad +n_Q(n)\{-A_cCG\sin(\Delta\omega n + \theta_i)\} \\ &\quad -n_I(n)\{-A_cCG\cos(\Delta\omega n + \theta_i)\} \\ &\quad +n_{TIM}(n) - n_{TQM}(n) + n_{TO}(n). \end{aligned}$$

By grouping all the noise components together as n_O and substituting in for G, the output becomes

$$V_O(n) = 2A_c^2C^2\sin(2\pi\frac{(f_i - f_c)}{f_s}) + n_O(n), \qquad (4.40)$$

where

$$
\begin{aligned}
n_O(n) &= \{-A_c C \sin(\Delta\omega n + \theta_i) + n_Q(n)\} \times \\
&\quad \{n_I(n+1) - n_I(n-1) + n_{TID}(n)\} \\
&\quad - \{A_c C \cos(\Delta\omega n + \theta_i) + n_I(n)\} \times \\
&\quad \{n_Q(n+1) - n_Q(n-1) + n_{TQD}(n)\} \\
&\quad + n_Q(n)\{-A_c C G \sin(\Delta\omega n + \theta_i)\} \\
&\quad - n_I(n)\{-A_c C G \cos(\Delta\omega n + \theta_i)\} \\
&\quad + n_{TIM}(n) - n_{TQM}(n) + n_{TO}(n).
\end{aligned} \tag{4.41}
$$

The calculations deriving the output noise power from the output noise component $n_O(n)$ are shown in Appendix A. The output noise power is determined to be

$$
\begin{aligned}
P_n &= 4\sigma_{NQ}^2\sigma_{NI}^2 + 4\sigma_{NQ}^2 n_{NI}^2 + 4n_{NQ}^2\sigma_{NI}^2 \\
&\quad + \sigma_{NQ}^2\sigma_{NTID}^2 + \sigma_{NQ}^2 n_{NTID}^2 + n_{NQ}^2\sigma_{NTID}^2 \\
&\quad + \sigma_{NI}^2\sigma_{NTQD}^2 + \sigma_{NI}^2 n_{NTQD}^2 + n_{NI}^2\sigma_{NTQD}^2 \\
&\quad + \sqrt{2}A_c C\{2\sigma_{NI}^2 + \sigma_{NTID}^2\} \\
&\quad + \sqrt{2}A_c C\{2\sigma_{NQ}^2 + \sigma_{NTQD}^2\} \\
&\quad + \sqrt{2}A_c CG\{\sigma_{NQ}^2 + \sigma_{NI}^2\} + \sigma_{NTIM}^2 + \sigma_{NTQM}^2 \\
&\quad + \sigma_{NTO}^2.
\end{aligned} \tag{4.42}
$$

All of the noise components can now be substituted in for, depending on whether the model is to represent a fixed point DSP-based system or an RNS-based system. All of the noise models and their associated means' and variances' are derived

earlier in this chapter. Note all noise sources with a subscript T, n_{NTxx} and σ^2_{NTxx}, represent truncations and are therefore 0 for the RNS-based systems, with the exception of n_{NTO} and σ^2_{NTO}.

The output signal power P_o is calculated to be

$$\begin{aligned} P_o &= (2A_c^2C^2G)^2 \\ &= 4A_c^4C^4G^2. \end{aligned} \tag{4.43}$$

From these results, the output SNR is found to be

$$\begin{aligned} SNR_{out} &= \frac{P_o}{P_n} \\ &= \{4A_c^4C^4G^2\}/\Big\{4\sigma^2_{NQ}\sigma^2_{NI} + 4\sigma^2_{NQ}n^2_{NI} + 4n^2_{NQ}\sigma^2_{NI} \\ &\quad +\sigma^2_{NQ}\sigma^2_{NTID} + \sigma^2_{NQ}n^2_{NTID} + n^2_{NQ}\sigma^2_{NTID} \\ &\quad +\sigma^2_{NI}\sigma^2_{NTQD} + \sigma^2_{NI}n^2_{NTQD} + n^2_{NI}\sigma^2_{NTQD} \\ &\quad +\sqrt{2}A_cC\{2\sigma^2_{NI} + \sigma^2_{NTID}\} \\ &\quad +\sqrt{2}A_cC\{2\sigma^2_{NQ} + \sigma^2_{NTQD}\} \\ &\quad +\sqrt{2}A_cCG\{\sigma^2_{NQ} + \sigma^2_{NI}\} + \sigma^2_{NTIM} + \sigma^2_{NTQM} \\ &\quad + \sigma^2_{NTO}\Big\}. \end{aligned} \tag{4.44}$$

The output bit error rate (BER) can be determined from SNR_{out} by calculating the probability of the output noise signal being of greater magnitude than the desired output signal (causing an error in the output decision and therefore a bit error). The probability of a bit error, assuming equally likely 1's and 0's is

$$P(error) = \frac{1}{2}P(error/\text{`1'}) + \frac{1}{2}P(error/\text{`0'}), \tag{4.45}$$

where

$$P(error/\text{`1'}) = P\{2A_c^2C^2\sin(2\pi\frac{(f_i-f_c)}{f_s}) - n_O(n_k) < 0\} \tag{4.46}$$
$$= P\{n_O(n_k) > 2A_c^2C^2\sin(2\pi\frac{(f_i-f_c)}{f_s})\}$$

and $n_O(n)$, the output noise signal is a stationary process having the same mean and variance for all $n = n_k$. Similarily,

$$P(error/\text{`0'}) = P\{2A_c^2C^2\sin(2\pi\frac{(f_c-f_i)}{f_s}) - n_O(n_k) > 0\} \tag{4.47}$$
$$= P\{n_O(n_k) < 2A_c^2C^2\sin(2\pi\frac{(f_c-f_i)}{f_s})\}.$$

Therefore,

$$P(error) = \frac{1}{2}P(error/\text{`1'}) + \frac{1}{2}P(error/\text{`0'}) \tag{4.48}$$
$$= P(error/\text{`1'})$$
$$= P\{n_O(n_k) > 2A_c^2C^2\sin(2\pi\frac{(f_i-f_c)}{f_s})\}.$$

The output noise can be considered to have a gaussian distribution according to the central limit theorem, as described on pages 214-221 [11]. Therefore the probability of an error is

$$P(error) = \int_{-\infty}^{-2A_c^2C^2\sin(2\pi\frac{(f_i-f_c)}{f_s})} \frac{1}{\sigma_n\sqrt{2\pi}} e^{-\frac{(x-m_n)^2}{2\sigma_n^2}} dx, \tag{4.49}$$

where m_n is the mean and σ_n is the standard deviation of the ouput noise signal. Therefore, for all input SNR's, E_b/N_o, the output signal and noise parameters can be calculated (as shown in Appendix A). Using these output parameters, the receiver's BER can be calculated by finding the probability of a bit error ($P(error)$ = BER). The performance of the receiver will be based upon the output BER versus the input E_b/N_o.

4.4 Theoretical Results

Using the equations derived in the previous section, the output SNR's, and therefore BER's, can be determined for various systems. A Matlab M-file was used to calculate the output BER for various receivers. A listing of this M-file, BER_OUT.M, is shown in Appendix C.

The model used for the RNS receiver allowed for a different number of bits to be used in the quantization of the system's inputs (including the analog-to-digital conversion, mixer coefficients and low pass filter coefficients) and for the final output. These word lengths are identified as Bits I (Inputs) and O (Output). When discussing the system's word lengths, the inputs word length will precede the output word length. (e.g. 6 8 represents an RNS system whose inputs are quantized to 6 bits and whose output is truncated to 8 bits.)

The model used for the DSP-based receiver allowed only one system word length to be adjusted. This word length was used for all quantizations and truncations, both within the receiver and at the output of the receiver.

For the RNS-based receiver, as there are no truncations within the receiver, the following truncation noise characteristics are used:

$$\begin{aligned} n_{NTxx} &= 0 \text{ and} \\ \sigma^2_{NTxx} &= 0, \text{ except for} \\ n_{NTO} &= -2^{-(b_{out}+2)} \text{ and} \\ \sigma^2_{NTO} &= \frac{2^{2(b_{out}+1}}{12}, \end{aligned}$$

where $b_{out}+1$ is the number of bits used at the RNS receiver output. For DSP-based

receivers, the following truncation noise characteristics are used:

$$\begin{aligned} n_{NTxx} &= -2^{-(b+2)} \text{ and} \\ \sigma^2_{NTxx} &= \frac{2^{-2(b+1)}}{12}, \end{aligned}$$

where $b+1$ is the number of bits used for DSP processing. For both methods of implementation, the following quantization noises characteristics are used:

$$\begin{aligned} n_{A/D} = n_{CI} = n_{CQ} &= 0 \text{ and} \\ \sigma^2_{A/D} = \sigma^2_{CI} = \sigma^2_{CQ} &= \frac{2^{-2(b+1)}}{12}, \end{aligned}$$

where $b+1$ is the number of bits used for quantization.

4.4.1 RNS Model Results

Table 4.1 contains the tabulated BER results for RNS-based systems with varying input E_b/N_o (SNR In) and system word lengths. These results are plotted in Figures 4.10 and 4.11 (split into 2 graphs for clarity) as BER vs. SNR In (in dB).

As shown in Figures 4.10 and 4.11, there is a similar pattern that is followed by all systems, and all systems are very close in performance. The graphs show that as the Input SNR increases, the BER decreases. It appears as though all the RNS systems modeled can be grouped into two categories, corresponding to their performance. The two groups can be considered as *Low*, and *High* BER vs. SNR In curves. These systems can be grouped as follows:

Low - 6 8, 6 10, 8 8, 8 10, 8 16 and 10 10;

High - 6 6 and 8 6;

The first conclusion that can be made from these results is that all receivers

Table 4.1: RNS BER Results from Model ($\times 10^{-3}$)

Bits	**SNR In (dB)**									
I O	1	2	3	4	5	6	7	8	9	10
6 6	76.0	42.1	18.2	5.43	.931	.068	$1.4E^{-3}$	$3.8E^{-6}$	$4.6E^{-10}$	$7.2E^{-12}$
6 8	75.0	41.2	17.5	5.09	.821	.053	$8.7E^{-4}$	$1.6E^{-6}$	$1.0E^{-10}$	$7.2E^{-12}$
6 10	74.9	41.2	17.5	5.06	.814	.052	$8.4E^{-4}$	$1.5E^{-6}$	$8.5E^{-11}$	$7.2E^{-12}$
8 6	75.8	42.0	18.1	5.41	.921	.067	$1.3E^{-3}$	$3.6E^{-6}$	$4.0E^{-10}$	$7.2E^{-12}$
8 8	74.9	41.1	17.5	5.05	.811	.052	$8.3E^{-4}$	$1.5E^{-6}$	$8.8E^{-11}$	$7.2E^{-12}$
8 10	74.8	41.1	17.4	5.04	.804	.051	$8.1E^{-4}$	$1.4E^{-6}$	$7.5E^{-11}$	$7.2E^{-12}$
8 16	74.9	41.1	17.4	5.03	.804	.051	$8.0E^{-4}$	$1.4E^{-6}$	$7.4E^{-11}$	$7.2E^{-12}$
10 10	74.8	41.1	17.4	5.03	.804	.051	$8.0E^{-4}$	$1.4E^{-6}$	$7.4E^{-11}$	$7.2E^{-12}$

Note: I = the number of bits used to quantize the inputs;

O = the number of bits used to represent the output.

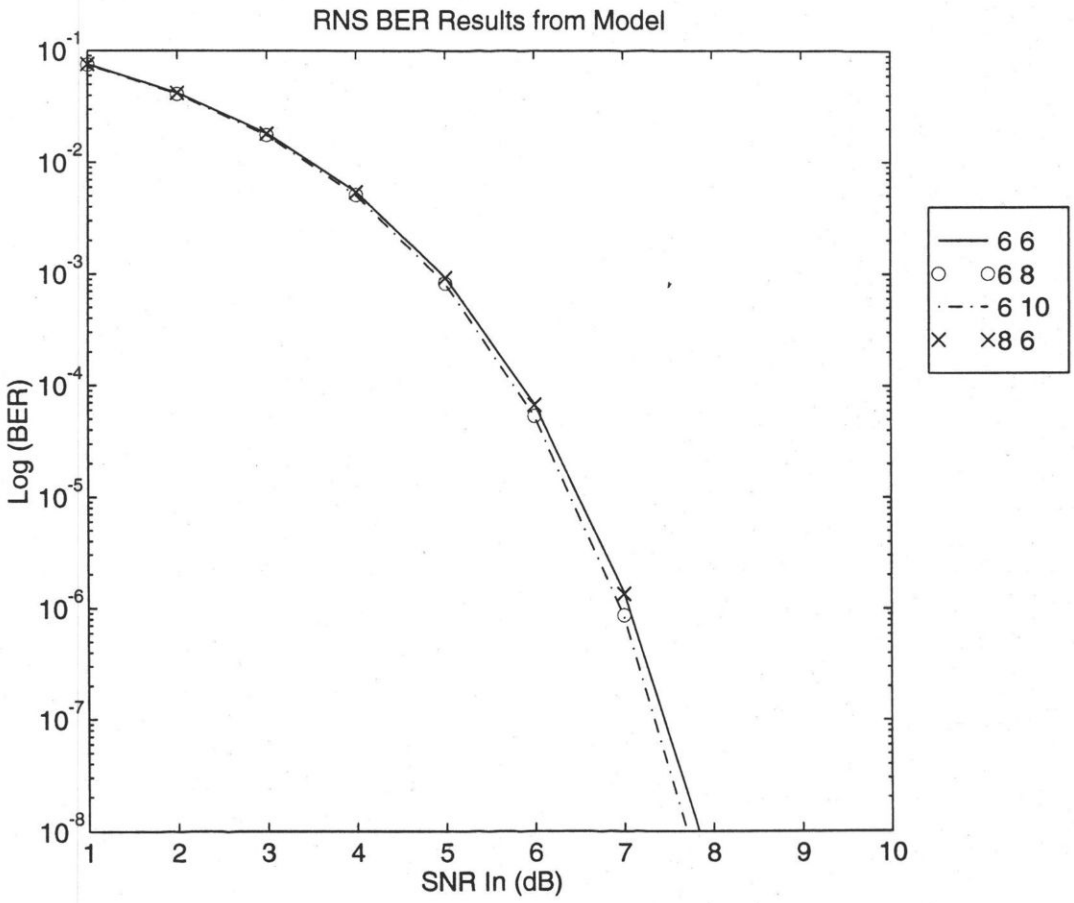

Figure 4.10: RNS BER Results from Model - 6-bit Configurations

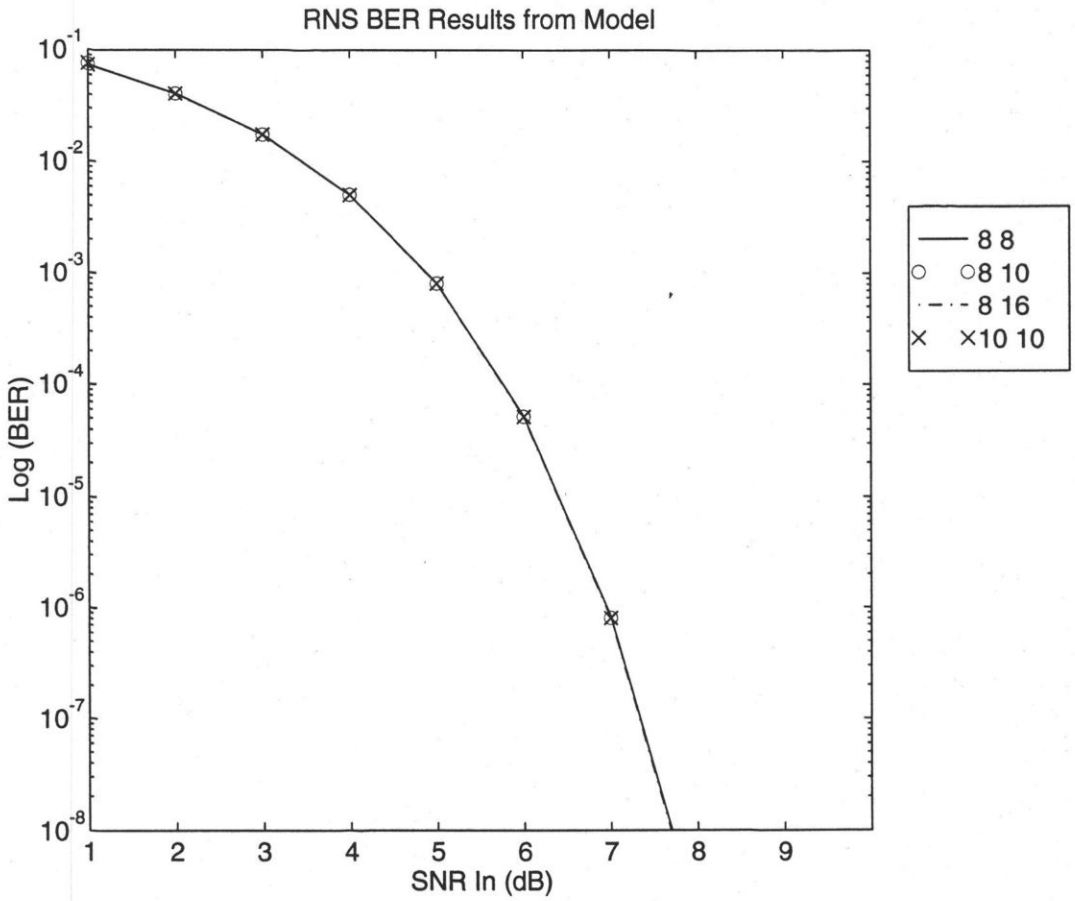

Figure 4.11: RNS BER Results from Model - 8 and 10-bit Configurations

provide similar performance at low Input SNR's, but are differentiated at higher Input SNR's.

These results also show that the number of bits used at the output of the receiver is important. For instance, the 6-bit input systems show increased performance as the number of bits used at the output increase from 6 to 8. The 8-bit input systems show a similar increase in performance as the number of bits used at the output is increased from 6 to 8.

The same increase in performance is not seen, however, when the number of bits used at the output changes from 8 to 10, indicating that there may be a maximum performance for the receiver at some Input SNR.

4.4.2 DSP Model Results

A fixed point DSP-based receiver was also modeled. The number of bits used for the quantization of the input signals, DSP processing and the final output was held constant throughout the receiver model. This means that truncations must be performed within the receiver, therefore requiring truncation noise to be added according to the number of bits used for processing. Table 4.2 contains the tabulated BER results for Traditional Fixed Point DSP-based systems with varying input SNR and system word lengths.

The results for the fixed point DSP-based systems are plotted in Figure 4.12 as BER vs. SNR In (in dB).

These systems can also be seperated into three categories based upon their performance curves. These systems would be seperated as follows:

Low - 8, 9, 10, 12, 14 and 16.

Medium - 7;

Table 4.2: Traditional DSP BER Results from Model ($\times 10^{-3}$)

Bits	**SNR In (dB)**									
	1	2	3	4	5	6	7	8	9	10
6	89.9	52.9	24.9	8.41	1.71	.161	$4.6E^{-3}$	$2.1E^{-5}$	$5.6E^{-9}$	$7.2E^{-12}$
7	81.7	46.3	20.7	6.43	1.15	.088	$1.8E^{-3}$	$4.7E^{-6}$	$5.1E^{-10}$	$7.2E^{-12}$
8	78.2	43.6	19.0	5.66	.956	.067	$1.2E^{-3}$	$2.5E^{-6}$	$1.8E^{-10}$	$7.2E^{-12}$
9	76.4	42.3	18.2	5.33	.876	.059	$9.7E^{-4}$	$1.8E^{-6}$	$1.1E^{-10}$	$7.2E^{-12}$
10	75.6	41.7	17.8	5.18	.838	.055	$8.8E^{-4}$	$1.6E^{-6}$	$9.1E^{-11}$	$7.2E^{-12}$
12	75.0	41.1	17.5	5.06	.812	.052	$8.2E^{-4}$	$1.4E^{-6}$	$7.8E^{-11}$	$7.2E^{-12}$
14	74.9	41.1	17.4	5.03	.804	.052	$8.0E^{-4}$	$1.4E^{-6}$	$7.4E^{-11}$	$7.2E^{-12}$
16	74.9	41.1	17.4	5.03	.804	.052	$8.0E^{-4}$	$1.4E^{-6}$	$7.4E^{-11}$	$7.2E^{-12}$

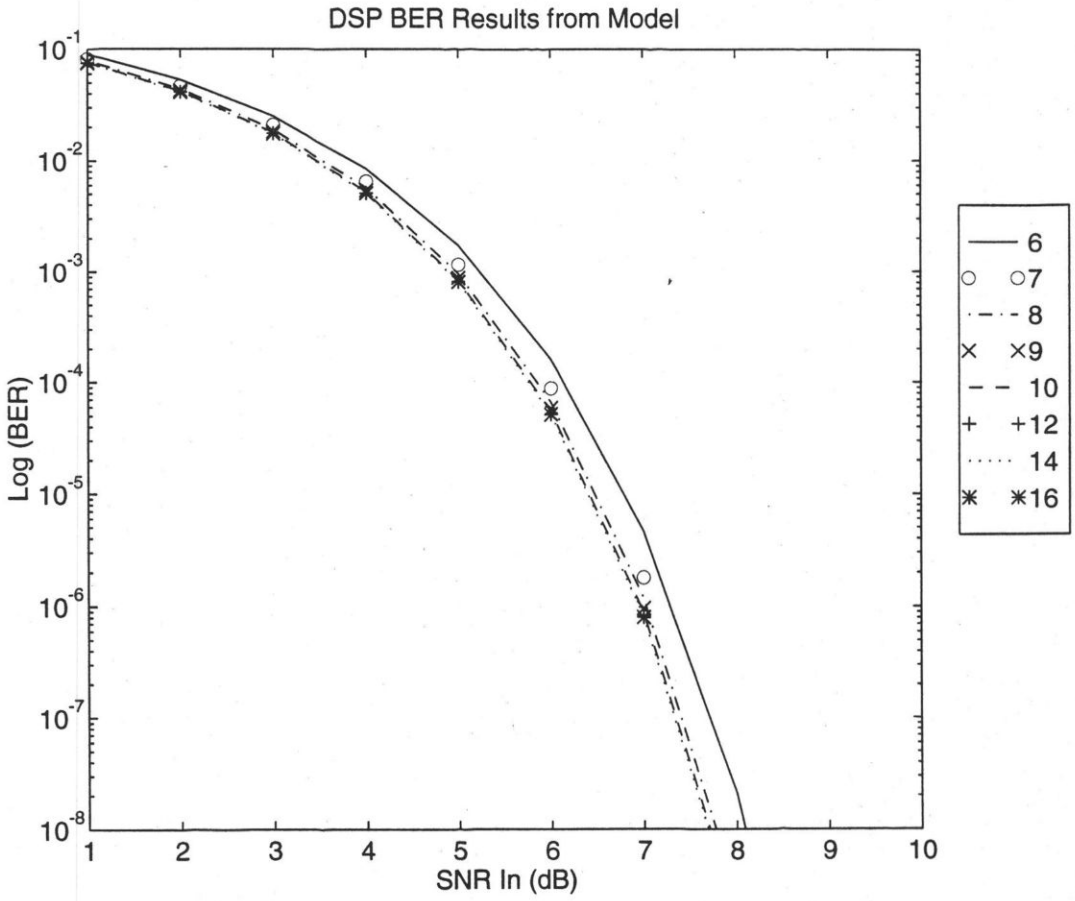

Figure 4.12: Fixed Point DSP BER Results from Model - Various Configurations

High - 6;

The results from the DSP-based model are as expected, where the 6-bit system provides the highest BER and the performance of the receiver improves with the increasing number of bits used. As with the RNS-based receivers, there seems to be a maximum performance for the receiver, since the 8, 9, 10, 12, 14 and 16-bit DSP-based receivers all provide very similar performance.

4.4.3 Comparing RNS and DSP Model Results

Figure 4.13 shows the comparison of the RNS-based receivers and the DSP-based receivers.

The RNS and DSP-based models follow the same general curve, however the RNS-based receiver models consistently maintain a lower BER, for the same Input SNR, than the fixed point DSP-based receiver models, even for systems using fewer bits at the input and output.

In particular, the 6 6 RNS receiver performs better than the 6 and 7-bit DSP-based receivers, and comes very close to the performance of the 8-bit DSP-based receiver. The 6 8 RNS receiver performs slightly better than the 8-bit DSP receiver at high SNR Inputs. The 8 6 and 8 8 RNS receivers perform very similar to the 8 and 10-bit DSP-based receivers.

Therefore, according to this model, a 6 6 RNS receiver could perform as well as an 8-bit DSP-based receiver. In order to verify that this model provides meaningful results, both types of receivers should be simulated and the BER results can be compared again. The results from the simulation should also be compared to those from the model, to determine the accuracy of the model. The simulation of these receivers is described in Chapter 5.

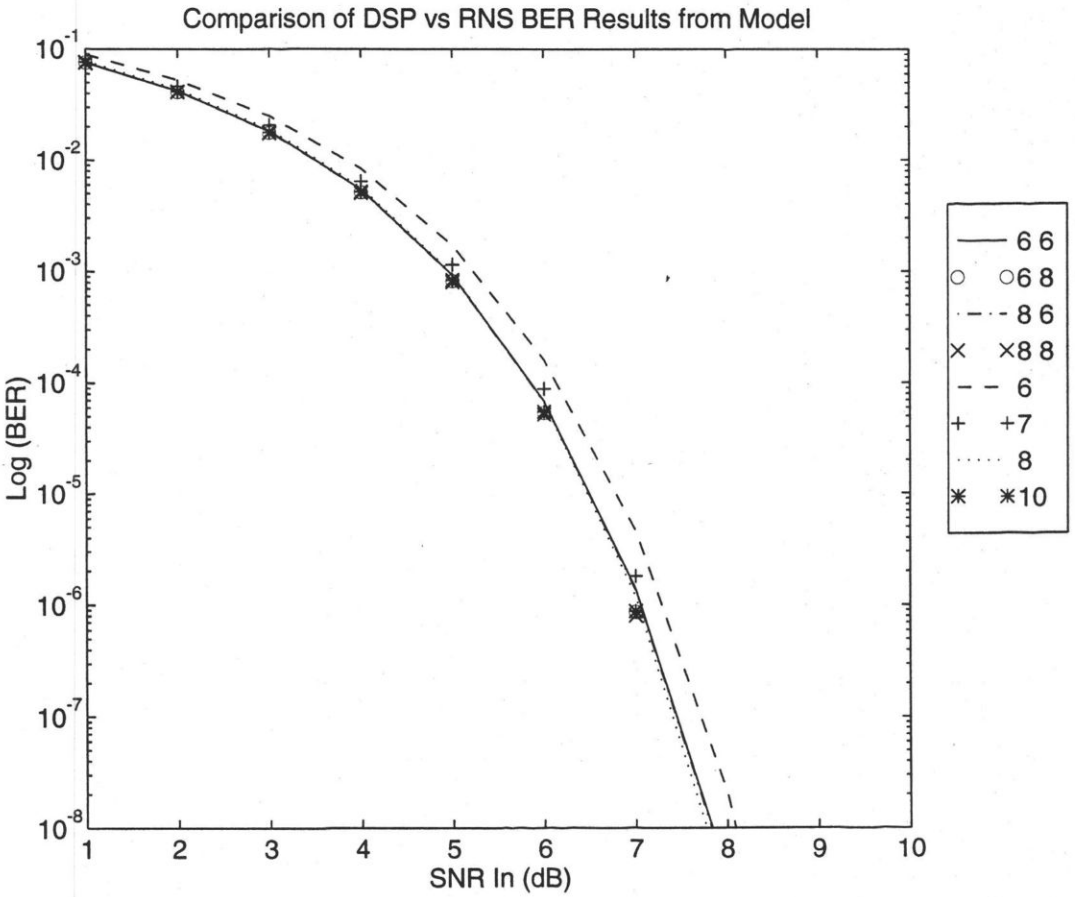

Figure 4.13: Comparison of DSP vs RNS BER Results from Simulation

Chapter 5

SIMULATION OF THE CROSS-CORRELATOR RECEIVER

Matlab was used to simulate the cross-correlator receiver. "M-files" are written as scripts, or series of commands for Matlab to process. The M-files used to simulate the cross-correlator receiver are shown in Appendix C. The main M-file was Rec1.m, which contained all of the parameter's for the receivers (DSP and RNS). These parameters were set and then the simulation was ran to obtain the BER for the specific receiver. All outputs were saved to disk for later analysis.

Some of the key features of the simulations are:

1. f_s = 9600Hz (sampling frequency), f_c = 1700Hz (fsk center frequency), ΔF = 400Hz (fsk peak frequency deviation), R_b = 1200bits/second (bit rate);

2. Automatic gain control kept input power at 80% of maximum;

3. Input was clipped to +/-1;

4. Gaussian noise was generated by using the central limit theorem (averaging a series of normally distributed random numbers);

5. The noise generator seed was set by the time, assuming that a different seed would be utilized for the majority of the simulations;

6. 31st order low pass filter was used with a 700Hz cutoff frequency;

7. Truncation was performed using Matlab's "fix" command by scaling the number, removing the decimal portion, and rescaling down;

8. Gains were added throughout the system to ensure signals were at levels that would utilize the full dynamic range;

9. A simple difference differentiator was utilized for simplicity;

10. Distorted bits, due to filtering, at the beginning and end of the simulation were removed;

11. Each bit at the output was sampled at the middle of it's bit interval;

5.1 Description of Simulation Files

Following is a list of the M-files that were used and a brief description of the function of that M-file. Descriptions are also provided in the M-files themselves, along with an explanation of how to utilize them. Listings of these M-files are given in Appendix C.

Rec1.m

The main file that contains all variables and the high level receiver functions. This file calls all other functions.

Gen.m

Generates the input FSK signal (no noise) as a continuous phase FSK signal at the desired frequency and level.

Gen_noi.m

Generates the white gaussian noise for input to the receiver using a normally distributed random number generator to generate many noise vectors and averages these vectors to obtain one white gaussian noise vector.

Clipper.m

Clips the input signal (including noise) to +/-1. This function assumes that the signal has already been amplified/attenuated to the desired level and simulates a clipper that would be present at the input to an A/D converter.

Down.m

Multiplies the input signal by the inphase and quadrature mixers to provide the down-mixed inphase and quadrature components.

Trunc.m

Scales a signal up by a desired factor, truncates the number and scales the output back down by the same factor. Simulates the truncation of a binary digital number.

Lpf.m

Lowpass filters the input signal according to the input variables.

Diff1.m

Differentiates the input signal using a simple difference equation.

Quant.m

Scales a number up, rounds it off to the nearest integer and then scales it back down by the same factor. Simulates a quantizer.

5.1.1 RNS vs. DSP Simulations

The same batch of M-files are used for the simulation of both the RNS and DSP-based receivers. Rec1.m calls the signal and noise generators to generate an input signal. This input signal is then quantized into two vectors for use in both the RNS and DSP-based receivers, respectively. The number of bits used in the quantization for RNS and DSP is specified individually at the beginning of Rec1.m. The remainder of the receiver's operations are then performed independently on both the RNS and DSP-based implementations.

For practical reasons, the exact method of implemetation for the RNS-based system, as described in Chapter 3, is not used. The RNS-based receiver uses Matlab's normal addition and multiplication techniques, however the number of bits used for quantization of all inputs and the truncation of the outputs is specified as for an RNS-based system. Therefore, the inputs are not converted into residues and the outputs do not have to be converted from residues bach into integers. The performance of the system used for simulation will accurately represent the performance of an actual RNS-based system, without the need of converting numbers to RNS and back.

The difference between the RNS and DSP-based simulations is that the DSP-based receiver requires truncation operations to be performed on its integers after every operation. That is, the truncation algorithm is performed after each multiplication, addition and subtraction, except in the low pass filter algorithm, where a double length accumulator is assumed for the growth of numbers [9]. The DSP system word length is used for the truncation of numbers throughout the receiver. The truncations provide the illusion of finite word lengths, in order to evaluate the performance of the DSP-based receiver.

5.2 Results of the Simulation

Simulations were run on various systems using the described M-files with different sets of input parameters. Input signals and noise were generated and fed into the receiver (at different SNR's). Each receiver was subjected to 1,000,000 bits for input signals with an SNR between 1dB and 6dB and 8,000,000 bits for input signals of 7dB to 10dB SNR. The output was monitored to determine actual output bit errors from which the BER was calculated. If 0 bit errors were obtained after 8,000,000 bits were received, a BER of 0 was recorded (due to computation power and timing requirements).

The receivers were designed so that the word lengths at different locations could be altered independently. The RNS-based receivers allowed for a seperate word length for the input signals (I) (including the analog-to-digital conversion, the mixer coefficients and the filter coefficient quantization) and output (O). The DSP-based receiver only allowed for one system word length to be selected, which was used for input quantization, processing and output.

5.2.1 RNS Simulation Results

Table 5.1 contains the tabulated BER results for RNS-based systems with varying input SNR and system word lengths.

These results are plotted in Figures 5.1 and 5.2 (split into 2 graphs for clarity) as BER vs. SNR In (in dB).

As shown by the graphs, the RNS-based systems used in the simulation appear to be divided into three distinct groups. Each system produces a BER vs. Input SNR curve that follows a one of three patterns. There are *Low, Medium* and *High* BER vs. SNR In curves. Obviously the *Low* curve represents the best receivers

Table 5.1: RNS BER Results from Simulation ($\times 10^{-3}$)

Bits	**SNR In (dB)**									
I O	1	2	3	4	5	6	7	8	9	10
6 6	25.6	12.8	5.51	2.07	.666	.154	, .034	.004	$4.0E^{-4}$	0
6 8	25.4	12.7	5.57	2.05	.618	.162	.033	.003	$4.0E^{-4}$	0
6 10	25.5	12.7	5.49	2.02	.666	.161	.032	.003	$4.0E^{-4}$	0
8 6	12.1	4.74	1.70	.452	.086	.016	.002	$1.0E^{-4}$	0	0
8 8	12.0	4.78	1.65	.420	.087	.018	.002	$1.0E^{-4}$	0	0
8 10	11.7	4.91	1.67	.393	.107	.015	.002	0	0	0
8 16	12.1	4.88	1.68	.412	.104	.019	.002	$2.0E^{-4}$	0	0
10 10	9.99	3.69	1.10	.251	.025	.003	$2.0E^{-4}$	0	0	0

Note: I = the number of bits used to quantize the inputs;

O = the number of bits used to represent the output.

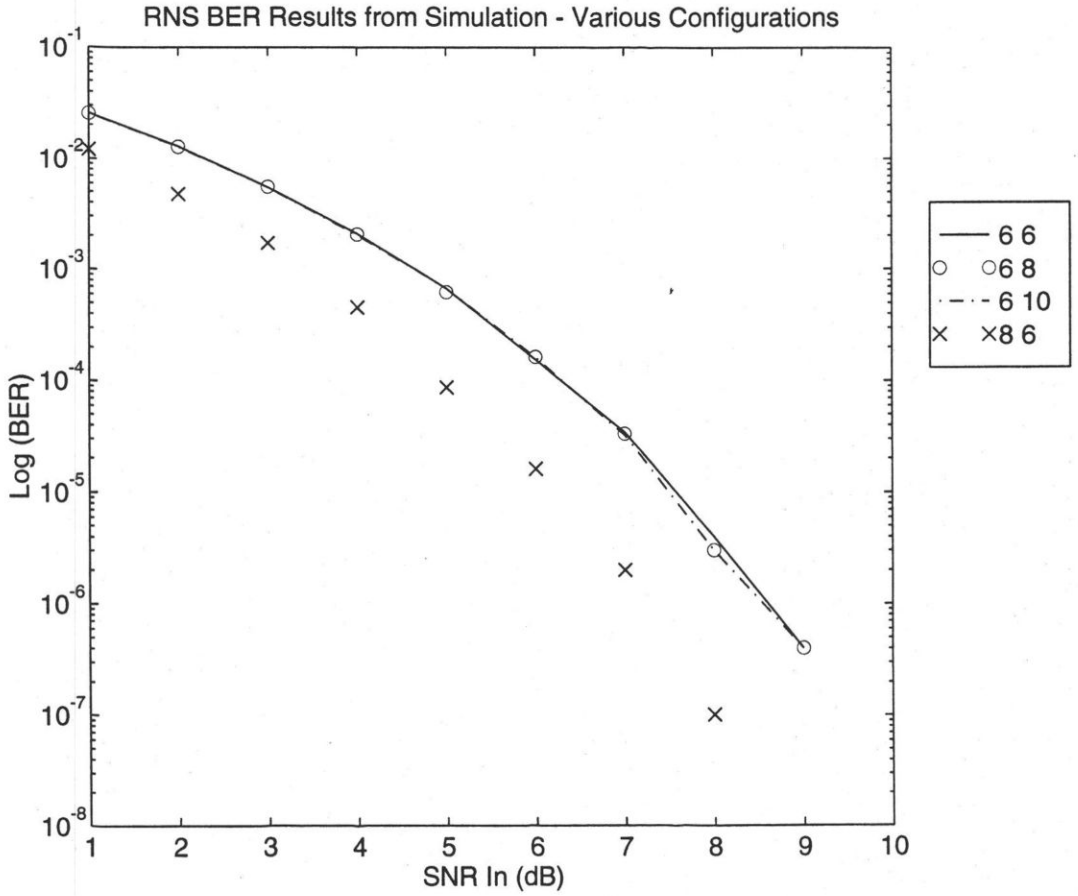

Figure 5.1: RNS BER Results from Simulation - 6-bit Configurations

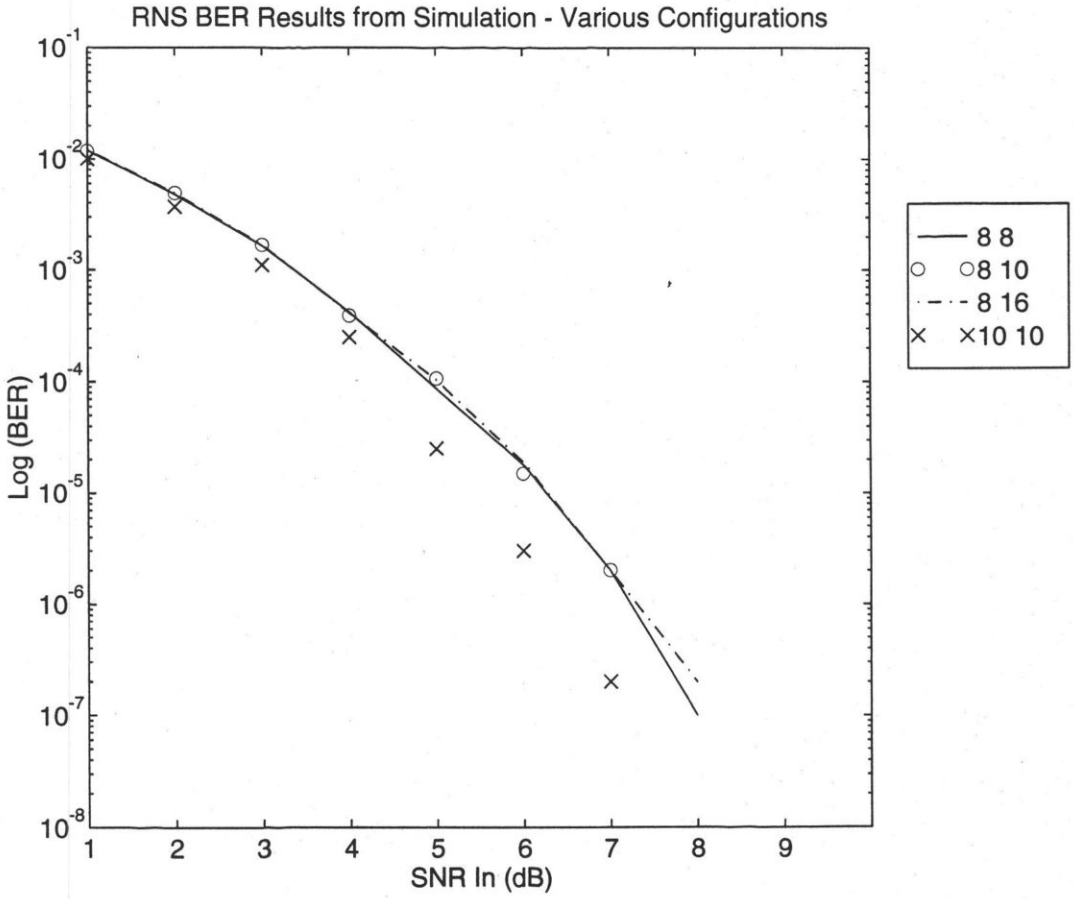

Figure 5.2: RNS BER Results from Simulation - 8 and 10-bit Configurations

since these systems produce the lowest number of bit errors.

The systems can be grouped as follows:

Low - 10 10;

Medium - 8 6, 8 8, 8 10 and 8 16;

High - 6 6, 6 8 and 6 10.

These groupings indicate the importance of the number of bits used in the quantizations. The two lower categories, *Low* and *Medium*, all use 8-bit and 10-bit quantizations. For example, the 8 6 system using 8 bits for the inputs and 6 bits at the output showed lower BER results than the 6 8 and 6 10 systems. The number of bits used at the output does not seem to affect the performance of the receiver.

In order to determine where the quantizing is more important, three more systems were simulated. These receivers altered the number of bits used for the mixer coefficients and the low pass filter coefficients. The results from these systems are shown in Table 5.2, and plotted in Figure 5.3.

The results show that increasing the number of bits used for the mixer coefficient quantization from 6 to 8 does not affect the performance of the receiver. However, increasing the bits used in the lowpass filter coefficients from 6 to 8 definitely improved the receiver's performance.

These results should be compared to those of a Fixed Point DSP-based system. Table 5.3 contains the tabulated BER results for Traditional Fixed Point DSP-based systems with varying input SNR and system word lengths.

Table 5.2: Modified RNS BER Results from Simulation ($\times 10^{-3}$)

Bits	**SNR In (dB)**									
I M F O	1	2	3	4	5	6	7	8	9	10
6 6 6 8	25.4	12.7	5.57	2.05	.618	.162	.033	.003	$4.0E^{-4}$	0
6 8 6 8	25.5	12.7	5.43	2.09	.603	.176	.025	.004	$4.0E^{-4}$	0
6 6 8 8	12.2	4.92	1.56	.441	.124	.015	.006	$5.0E^{-4}$	0	0
6 6 8 6	12.0	5.00	1.69	.456	.111	.011	.003	$3.0E^{-4}$	0	0

Note: I = the number of bits used to quantize the input;

M = the number of bits used to quantize the mixer coefficients;

F = the number of bits used to quantize the filter coefficients;

O = the number of bits used to represent the output.

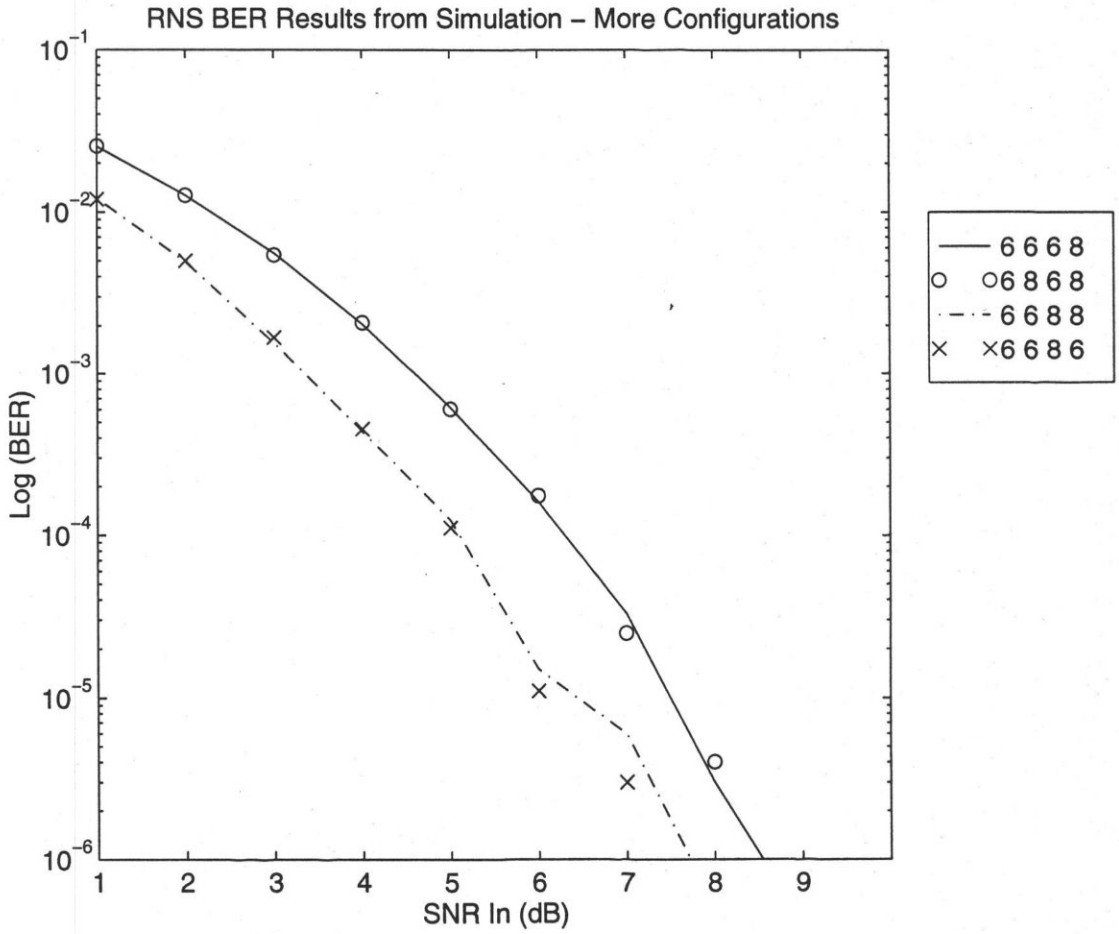

Figure 5.3: RNS BER Results from Simulation - More 6 and 8-bit Configurations

Table 5.3: Traditional DSP BER Results from Simulation ($\times 10^{-3}$)

Bits	**SNR In (dB)**									
	1	2	3	4	5	6	7	8	9	10
6	30.8	16.3	7.47	3.25	1.22	.390	.119	.029	.006	0
7	16.7	7.49	2.92	.884	.234	.053	.007	.001	0	0
8	12.0	5.02	1.79	.433	.095	.021	.002	$1.2E^{-4}$	0	0
9	10.7	3.98	1.27	.267	.040	.009	$4.0E^{-4}$	0	0	0
10	10.2	3.68	1.13	.201	.039	.005	$1.0E^{-4}$	0	0	0
12	9.83	3.73	1.11	.197	.046	.006	$1.0E^{-4}$	0	0	0
14	10.0	3.71	1.14	.236	.033	.003	$1.0E^{-4}$	0	0	0
16	9.91	3.68	1.10	.244	0.25	.004	$1.0E^{-4}$	0	0	0

5.2.2 DSP Simulation Results

These results are plotted in Figure 5.4 as BER vs. SNR In (in dB).

The Fixed Point DSP systems provide similar results to those obtained from the RNS-based systems. These systems can be divided into four groups, *Low, Medium Low, Medium High* and *High* BER vs. SNR In curves.

These systems can be grouped as follows:

Low - 9, 10, 12, 14, 16;

Medium Low - 8;

Medium High - 7;

High - 6.

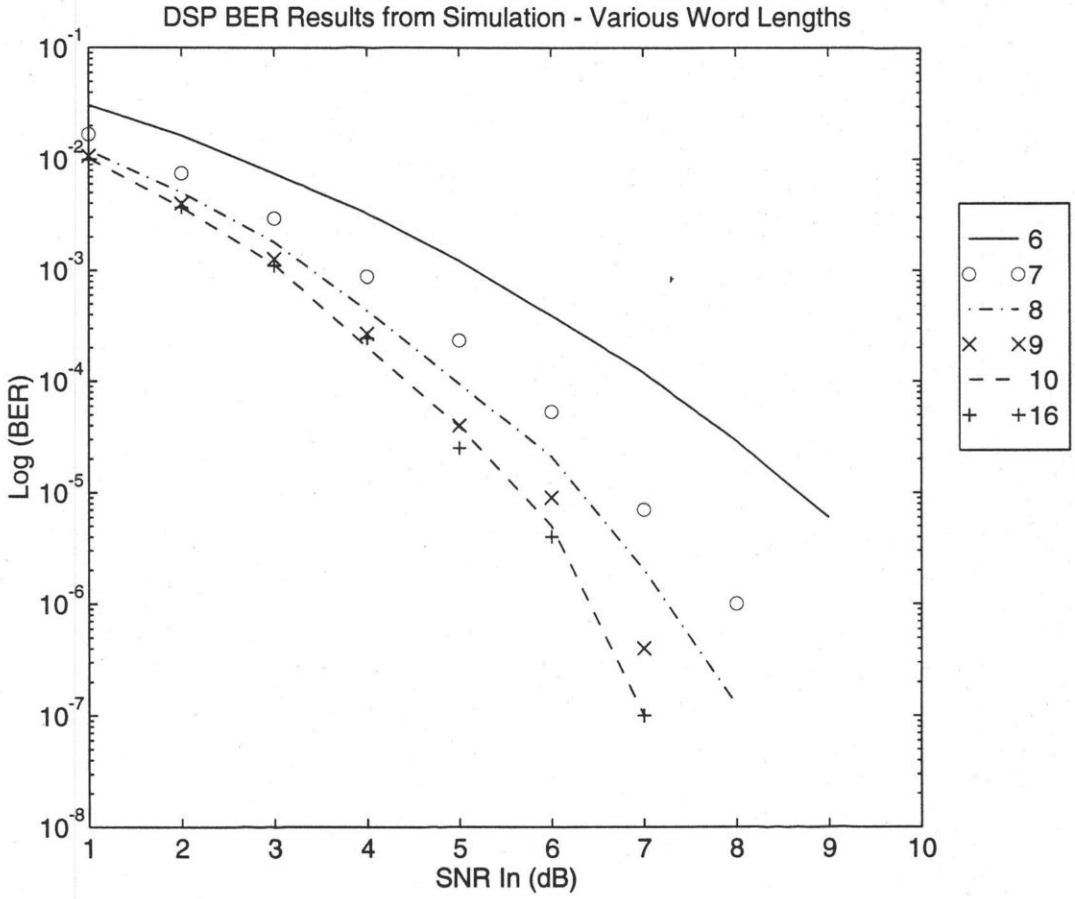

Figure 5.4: DSP BER Results from Simulation - Various Configurations

As with the model's results, the performance of the DSP-based receivers are as expected. The 6-bit receiver has the worst BER with increasing performance as the number of bits used increases. Again, there seems to be a maximum performance curve that the 10 and 16-bit DSP-based receivers approach.

5.2.3 Comparing RNS and DSP Simulation Results

By plotting some of the RNS and DSP-based receivers on the same graph, the performance of these systems can be compared. Figure 5.5 compares some of the RNS systems with the four groups of the DSP-based systems (modelled and simulated systems are distinguished with an M and an S).

As seen in the model's results, all BER vs. SNR In curves follow the same general pattern, however, the simulation's curves are spread out more at both low and high Input SNRs.

The 6 6 and 6 8 RNS systems both perform better than the 6-bit DSP receiver. The 8 6 and 8 8 RNS receivers both out perform the 7-bit DSP system, but are only slightly better that the 8-bit DSP system. As discussed in Section 5.2.1, 6 6 and 6 8 RNS-based receivers using 8-bits for filter coefficient quantization would perform as well as an 8-bit DSP-based receiver.

5.3 Model vs. Simulation

In order to determine the accuracy of the model, its results will be compared to those of the simulation, for similar systems. The performance of the modelled RNS-based receivers and the simulated receivers differ slightly. Figure 5.6 shows the comparison of the modelled and simulated receivers. One of the differences being that the 8 6 receiver performs poorer than the 6 8 and 6 10 receivers in the

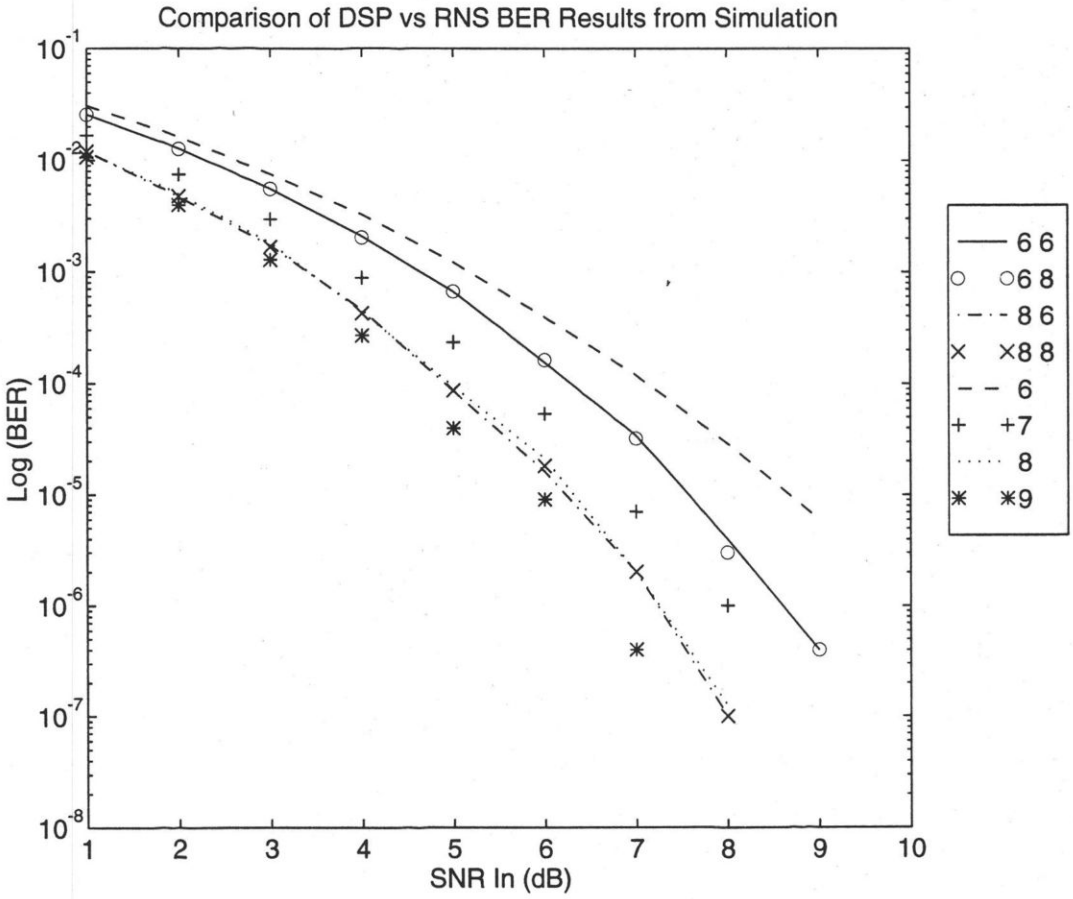

Figure 5.5: Comparison of DSP vs RNS BER Results from Simulation

model, but better in the simulation. This again indicates the importance of the filter coefficient quantization and the difficulties in modelling an FIR filter as an ideal filter with noise injectors.

The BER vs. SNR In curves for the models have a slightly steeper slope that those from the simulation, indicating that the input noise has more affect on the modeled systems than on the simulated systems. However, the relative performance of the receivers within the model and the simulation are consistent which indicates that the truncation noise effects are fairly accurate.

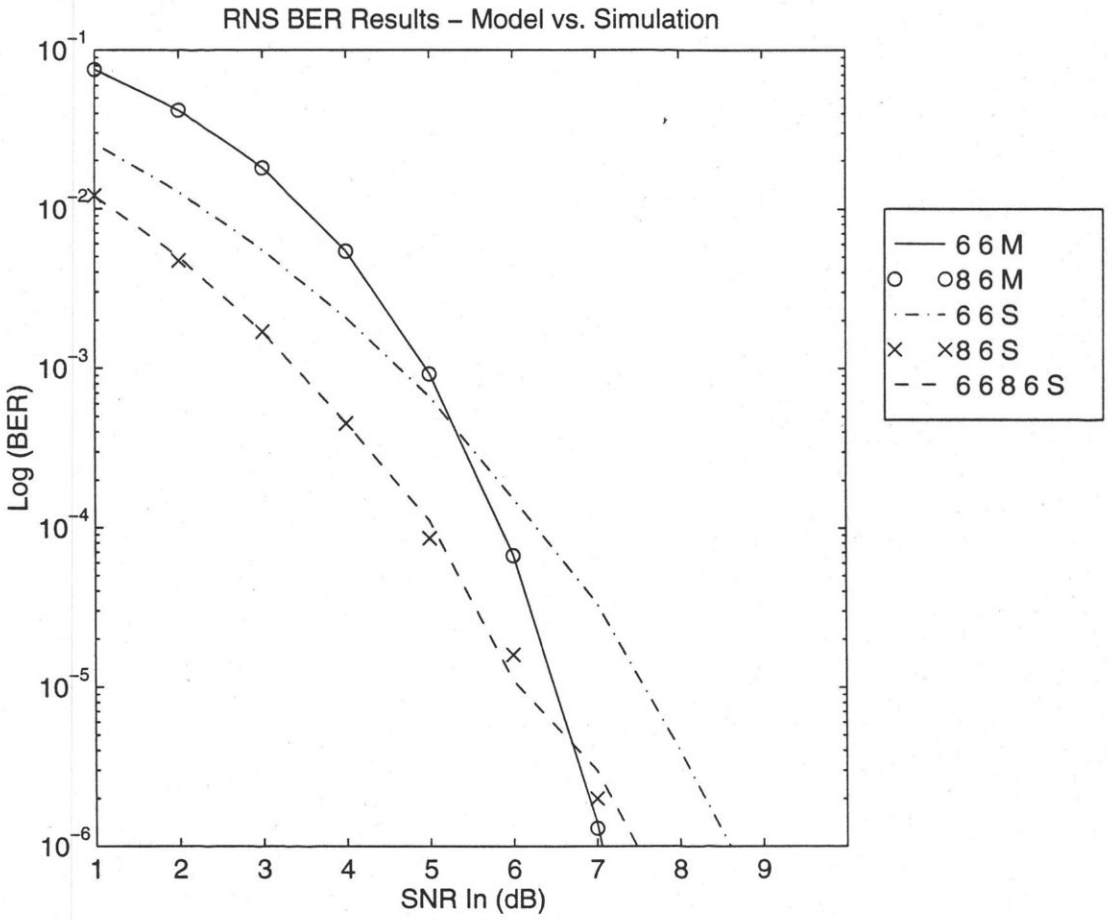

Figure 5.6: Comparison of RNS BER Results from Model vs. Simulation

Chapter 6

CONCLUSION

6.1 Conclusion

This thesis has presented the concept of Residue Number Systems, and applied these techniques to the cross-correlator receiver. Various advantages and disadvantages of this method were discussed briefly, followed by an in depth evaluation of the output BER from a Residue Number System based cross-correlator receiver. Results were obtained using theoretical analysis and computer simulation and compared to the same receiver using a fixed-point DSP-based implementation.

Chapter 4 performed a theoretical analysis of the cross-correlator receiver and used this analysis to develop a model of the system. The model was generated such that it would allow for the performance of both the RNS and DSP-based receivers to be evaluated. The results obtained from modelling of the receivers showed that the RNS-based receiver could perform as well as the DSP-based receiver. In particular, an RNS-based system using 6-bits for quantization and at the output provides a lower bit error rate at all input SNR's than the 8-bit fixed point DSP-

based receiver. Similarily, the RNS systemusing 8-bits at the input and output performed as well as the 10-bit DSP-based system.

Chapter 5 analyzed the performance of the receivers using simulations. The simulations provided different results than the model. The 6-bit input and output RNS-based systems did not perform as well as the 7 or 8-bit DSP-based systems. In order to improve the bit error rate performance of the RNS-based receivers, more bits were used for the quantization of the inputs. The results from this analysis showed that by using 8-bits for the quantization of the filter coefficients, the 6-bit input and output RNS-based system could perform as well as the 8-bit DSP-based receiver. This indicates the significance of the filter coefficients on the performance of the receiver.

In comparing the results obtained from the models and simulations, the first observation made is that the performance of the systems simulated were more spread out. The model showed that the systems should perform better than they did for the simulation at low input SNR's. However, The simulation showed that the systems actually performed better than the model at higher input SNR's. This indicates that the model is more affected by the channel noise present at the input to the receiver, and not as affected by the internally generated noise. Some improvements could be made to closer match the performance of the simulations, however, the results obtained from the model do agree with those from the simulation. That is, a RNS-based system can perform as well a DSP-based system, while using fewer bits at the input and output.

The conclusion of this study is that using a residue number system based cross-correlator receiver can perform as well as a fixed point DSP-based receiver using fewer bits for processing and output. The model of this receiver could be used to give a reasonable analysis of the performance of the receiver, prior to initiating the time consuming efforts of simulating or implementing the receiver. However,

due to the system size and memory requirements associated with the RNS-based receiver, the RNS method of implementation for the cross-correlator receiver may be too complex. There are too many operations that need to be performed, which force the dynamic range requirements to be very large. The advances of floating point processors, and increasing speeds of fixed point processors indicate that these platforms would provide a more suitable solution for this receiver.

6.2 Future Work

The scope of this study was quite limited and therefore there are many areas for futher study.

1. Further investigation into the physical size of the system required for a complete receiver, and whether or not it could be made into a single chip receiver.

2. More work can be performed to increase the RNS's speed, taking advantage of high speed memories, pipelining architectures and simultaneous processors.

3. There has been previous research in developing fast and simple Chinese remainder theorem algorithms for the conversion from RNS to binary (or integer) and this will continue to be an area for further investigation.

4. The addition and multiplication operations can also be researched further to determine the optimum method of implementation (table look-up, shift registers, etc.).

5. Obviously these RNS techniques can be applied to different receiver topologies to determine the advantages in other specific applications.

Bibliography

[1] W. K. Jenkins "Finite arithmetic concepts," in *Handbook for Digital Signal Processing.*, (S. K. Mitra and J. F. Kaiser, eds), John Wiley and Sons Inc., pp. 611-627,667-675, 1993.

[2] K. A. Farrell, "Performance of the Cross-Correlator Receiver for Binary Digital Frequency Modulation," *Master's Thesis.*, Queen's University, January 1993.

[3] W. K. Jenkins and B. J. Leon, "The Use of Residue Number Systems in the Design of Finite Response Digital Filters," *IEEE Trans. Circuits Syst.*, vol. CAS-24, pp. 191-200, April 1977.

[4] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and It's Applications to Computer Technology.* McGraw-Hill Inc., New York, 1967.

[5] F. J. Taylor and A. S. Ramnarayanan, "An Efficient Residue-to-Decimal Converter," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 1164-1168, December 1981.

[6] Leon W. Couch II, *Digital and Analog Communication Systems.* Macmillan Publishing Company, New York, 3rd Edition, 1990.

[7] F. M. Gardner, "Properties of Frequency Difference Detectors," *IEEE Trans. Comm.*, vol. COM-33, pp. 131-138, February 1985.

[8] John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing.* Macmillan Publishing Company, New York, 2nd Edition, 1992.

[9] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-time Signal Procesing.* Prentice-Hall Inc., Toronto, 1989.

[10] John G. Proakis, *Digital Communications.* McGraw-Hill Inc., Toronto, 2nd Edition, 1989.

[11] Athanasios Papoulis, *Probability , Random Variables, and Stochastic Processes.* McGraw-Hill Inc., Toronto, 3rd Edition, 1991.

Appendix A

Output Noise Power Calculation

The output noise signal is

$$
\begin{aligned}
n_O(n) &= \{-A_c C \sin(\omega n + \theta_i) + n_Q(n)\}\{n_I(n+1) - n_I(n-1) + n_{TID}(n)\} \\
&\quad -\{A_c C \cos(\omega n + \theta_i) + n_I(n)\}\{n_Q(n+1) - n_Q(n-1) + n_{TQD}(n)\} \\
&\quad +n_Q(n)\{-A_c C G \sin(\omega n + \theta_i)\} - n_I(n)\{-A_c C G \cos(\omega n + \theta_i)\} \\
&\quad +n_{TIM}(n) - n_{TQM}(n) + n_{TO}(n),
\end{aligned}
\tag{A.1}
$$

where

$$
\omega = 2\pi \frac{(f_i - f_c)}{f_s}, \tag{A.2}
$$

$$
G = 2\sin(\omega), \tag{A.3}
$$

$$
n_I(n) = \overline{2C \cos(2\pi \frac{f_c}{f_s} n)(n_i(n) + n_{A/D}(n))} + \overline{n_c(n) A_c \cos(2\pi \frac{f_i}{f_s} n + \theta_i)} \tag{A.4}
$$

$$+\overline{n_c(n)(n_i(n)+n_{A/D}(n))}+\overline{n_{TI}(n)}+n_{TIF}(n),$$

and

$$n_Q(n) = \overline{2C\sin(2\pi\frac{f_c}{f_s}n)(n_i(n)+n_{A/D}(n))}+\overline{n_s(n)A_c\cos(2\pi\frac{f_i}{f_s}n+\theta_i)} \quad (A.5)$$
$$+\overline{n_s(n)(n_i(n)+n_{A/D}(n))}+\overline{n_{TQ}(n)}+n_{TQF}(n).$$

Note: $\overline{\text{Overlined Terms}}$ indicate low pass filtered.

In order to obtain a reasonable equation that can be used to calculate the noise power at the output, some simplification must be performed. Simplification of this equation will require making some assumptions.

1. All noise components of different sources and times will be considered to be independent (and thereofore uncorrelated). That is, $R\{(n_s(n), n_c(n)\} = 0$ for all n and $R\{(n_{TQ}(n), n_{TQ}(n+1)\} = 0$ for all n

2. Filtering will have no affect on the noise components other than reducing their relative noise power by $\frac{2BW}{f_s}$, where BW is the bandwidth of the filter.

3. Multiplying a noise signal by a sin or cos waveform will not affect the signal other than altering the magnitude of that component.

   If we consider all the noise terms to be uncorrelated and independent then we can calculate the power of the noise output using the addition and multiplication rules described in Appendix B for uncorrelated and independent processes.

$$
\begin{aligned}
n_O(n) &= \{-A_cC\sin(\omega n+\theta_i)+n_Q(n)\} \qquad \text{(A.6)}\\
&\quad \times\{n_I(n+1)-n_I(n-1)+n_{TID}(n)\}\\
&\quad -\{A_cC\cos(\omega n+\theta_i)+n_I(n)\}\\
&\quad \times\{n_Q(n+1)-n_Q(n-1)+n_{TQD}(n)\}\\
&\quad +n_Q(n)\{-A_cCG\sin(\omega n+\theta_i)\}\\
&\quad -n_I(n)\{-A_cCG\cos(\omega n+\theta_i)\}+n_{TIM}(n)-n_{TQM}(n)\\
&\quad +n_{TO}(n)\\
&= \{-\sqrt{2}A_cC+n_Q(n)\}\{n_I(n+1)-n_I(n-1)+n_{TID}(n)\}\\
&\quad -\{\sqrt{2}A_cC+n_I(n)\}\{n_Q(n+1)-n_Q(n-1)+n_{TQD}(n)\}\\
&\quad -\sqrt{2}A_cCGn_Q(n)+\sqrt{2}A_cCGn_I(n)+n_{TIM}(n)-n_{TQM}(n)\\
&\quad +n_{TO}(n)\\
&= -\sqrt{2}A_cC\{n_I(n+1)-n_I(n-1)+n_{TID}(n)\}\\
&\quad +n_Q(n)\{n_I(n+1)-n_I(n-1)+n_{TID}(n)\}\\
&\quad -\sqrt{2}A_cC\{n_Q(n+1)-n_Q(n-1)+n_{TQD}(n)\}\\
&\quad -n_I(n)\{n_Q(n+1)-n_Q(n-1)+n_{TQD}(n)\}\\
&\quad -\sqrt{2}A_cCGn_Q(n)+\sqrt{2}A_cCGn_I(n)+n_{TIM}(n)-n_{TQM}(n)\\
&\quad +n_{TO}(n)\\
&= n_Q(n)\{n_I(n+1)-n_I(n-1)+n_{TID}(n)\}\\
&\quad -n_I(n)\{n_Q(n+1)-n_Q(n-1)+n_{TQD}(n)\}\\
&\quad -\sqrt{2}A_cC\{n_I(n+1)-n_I(n-1)+n_{TID}(n)\}\\
&\quad -\sqrt{2}A_cC\{n_Q(n+1)-n_Q(n-1)+n_{TQD}(n)\}\\
&\quad -\sqrt{2}A_cCG\{n_Q(n)-n_I(n)\}+n_{TIM}(n)-n_{TQM}(n)\\
&\quad +n_{TO}(n),
\end{aligned}
$$

where n_{NX} is the mean of $n_X(n)$.

The noise power can be calculated statistically using the mean and variance of the noise signal in the equation

$$P_{NO} = n_{NO}^2 + \sigma_{NO}^2, \tag{A.7}$$

where n_{NO}^2 represents the DC noise power and σ_{NO}^2 represents the power of the AC noise.

Therefore the mean and variance of the output noise signal must be determined. Using the rules described in Appendix B, the statistics of this noise are found to be

$$\begin{aligned} n_{NO} &= E[n_O(n)] \\ &\quad n_{NQ}\{n_{NI} - n_{NI} + n_{NTID}\} - n_{NI}\{n_{NQ} - n_{NQ} + n_{NTQD}\} \\ &\quad -\sqrt{2}A_cC\{n_{NI} - n_{NI} + n_{NTID}\} \\ &\quad -\sqrt{2}A_cC\{n_{NQ} - n_{NQ} + n_{NTQD}\} \\ &\quad -\sqrt{2}A_cCG\{n_{NQ} - n_{NI}\} + n_{NTIM} - n_{NTQM} + n_{NTO} \\ &= n_{NQ}n_{NTID} - n_{NI}n_{NTQD} - \sqrt{2}A_cC\{n_{NTID} + n_{NTQD}\} \\ &\quad -\sqrt{2}A_cCG\{n_{NQ} - n_{NI}\} + n_{NTIM} - n_{NTQM} + n_{NTO}, \end{aligned} \tag{A.8}$$

$$\begin{aligned} n_{NO}^2 &= \{n_{NQ}n_{NTID} - n_{NI}n_{NTQD} - \sqrt{2}A_cC\{n_{NTID} + n_{NTQD}\} \\ &\quad -\sqrt{2}A_cCG\{n_{NQ} - n_{NI}\} + n_{NTIM} - n_{NTQM} + n_{NTO}\}^2, \end{aligned} \tag{A.9}$$

and

$$\begin{aligned} \sigma_{NO}^2 &= E[n_O(n)^2] - n_{NO}^2 \\ &= \sigma_{NQ}^2\sigma_{NI}^2 + \sigma_{NQ}^2 n_{NI}^2 + n_{NQ}^2\sigma_{NI}^2 \\ &\quad +\sigma_{NQ}^2\sigma_{NI}^2 + \sigma_{NQ}^2 n_{NI}^2 + n_{NQ}^2\sigma_{NI}^2 \end{aligned} \tag{A.10}$$

$$
\begin{aligned}
&+\sigma^2_{NQ}\sigma^2_{NTID} + \sigma^2_{NQ}n^2_{NTID} + n^2_{NQ}\sigma^2_{NTID} \\
&\sigma^2_{NI}\sigma^2_{NQ} + \sigma^2_{NI}n^2_{NQ} + n^2_{NI}\sigma^2_{NQ} \\
&+\sigma^2_{NI}\sigma^2_{NQ} + \sigma^2_{NI}n^2_{NQ} + n^2_{NI}\sigma^2_{NQ} \\
&+\sigma^2_{NI}\sigma^2_{NTQD} + \sigma^2_{NI}n^2_{NTQD} + n^2_{NI}\sigma^2_{NTQD} \\
&+\sqrt{2}A_cC\{\sigma^2_{NI} + \sigma^2_{NI} + \sigma^2_{NTID}\} \\
&+\sqrt{2}A_cC\{\sigma^2_{NQ} + \sigma^2_{NQ} + \sigma^2_{NTQD}\} \\
&+\sqrt{2}A_cCG\{\sigma^2_{NQ} + \sigma^2_{NI}\} + \sigma^2_{NTIM} + \sigma^2_{NTQM} \\
&+\sigma^2_{NTO} \\
= \;& 4\sigma^2_{NQ}\sigma^2_{NI} + 4\sigma^2_{NQ}n^2_{NI} + 4n^2_{NQ}\sigma^2_{NI} \\
&+\sigma^2_{NQ}\sigma^2_{NTID} + \sigma^2_{NQ}n^2_{NTID} + n^2_{NQ}\sigma^2_{NTID} \\
&+\sigma^2_{NI}\sigma^2_{NTQD} + \sigma^2_{NI}n^2_{NTQD} + n^2_{NI}\sigma^2_{NTQD} \\
&+\sqrt{2}A_cC\{2\sigma^2_{NI} + \sigma^2_{NTID}\} \\
&+\sqrt{2}A_cC\{2\sigma^2_{NQ} + \sigma^2_{NTQD}\} \\
&+\sqrt{2}A_cCG\{\sigma^2_{NQ} + \sigma^2_{NI}\} + \sigma^2_{NTIM} + \sigma^2_{NTQM} \\
&+\sigma^2_{NTO}.
\end{aligned}
$$

In order to substitute in for the quadrature noise components, both the means and variances for these noise components must be determined as well.

$$
\begin{aligned}
n_I(n) \;=\; & \overline{2C\cos(2\pi\frac{f_c}{f_s}n)(n_i(n) + n_{A/D}(n))} \\
& \overline{+n_c(n)A_c\cos(2\pi\frac{f_i}{f_s}n + \theta_i)} \\
& +\overline{n_c(n)(n_i(n) + n_{A/D}(n))} + \overline{n_{TI}(n)} + n_{TIF}(n), \\
=\; & \frac{f_{lpf}}{f_s}\{2\sqrt{2}C(n_i(n) + n_{A/D}(n)) + \sqrt{2}A_cn_c(n) \\
& +n_c(n)(n_i(n) + n_{A/D}(n)) + n_{TI}(n)\} + n_{TIF}(n),
\end{aligned}
\tag{A.11}
$$

thus

$$\begin{aligned} n_{NI} &= E[n_I(n)] \\ &= \frac{f_{lpf}}{f_s}\{2\sqrt{2}C(n_{Ni}+n_{NA/D})+\sqrt{2}A_c n_{Nc}+n_{Nc}n_{Ni} \\ &\quad +n_{Nc}n_{NA/D}+n_{NTI}\}+n_{NTIF}, \end{aligned} \tag{A.12}$$

and

$$\begin{aligned} \sigma^2_{NI} &= E[n_I(n)^2]-n_{NI}(n)^2 \\ &= \frac{f_{lpf}}{f_s}\{2\sqrt{2}C(\sigma^2_{Ni}+\sigma^2_{NA/D})+\sqrt{2}A_c\sigma^2_{Nc} \\ &\quad +\sigma^2_{Nc}\sigma^2_{Ni}+\sigma^2_{Nc}n^2_{Ni}+n^2_{Nc}\sigma^2_{Ni} \\ &\quad +\sigma^2_{Nc}\sigma^2_{NA/D}+\sigma^2_{Nc}n^2_{NA/D}+n^2_{Nc}\sigma^2_{NA/D} \\ &\quad +\sigma^2_{NTI}\}+\sigma^2_{NTIF}. \end{aligned} \tag{A.13}$$

Similarily,

$$\begin{aligned} n_Q(n) &= \frac{f_{lpf}}{f_s}\{2\sqrt{2}C(n_i(n)+n_{A/D}(n))+\sqrt{2}A_c n_s(n) \\ &\quad +n_s(n)(n_i(n)+n_{A/D}(n))+n_{TQ}(n)\}+n_{TQF}(n), \end{aligned} \tag{A.14}$$

therefore,

$$\begin{aligned} n_{NQ} &= E[n_Q(n)] \\ &= \frac{f_{lpf}}{f_s}\{2\sqrt{2}C(n_{Ni}+n_{NA/D})+\sqrt{2}A_c n_{Ns}+n_{Ns}n_{Ni} \\ &\quad +n_{Ns}n_{NA/D}+n_{NTQ}\}+n_{NTQF}, \end{aligned} \tag{A.15}$$

and

$$\sigma^2_{NQ} = E[n_Q(n)^2]-n_{NQ}(n)^2 \tag{A.16}$$

$$
\begin{aligned}
= \; & \frac{f_{lpf}}{f_s}\{2\sqrt{2}C(\sigma_{Ni}^2 + \sigma_{NA/D}) + \sqrt{2}A_c\sigma_{Ns}^2 \\
& +\sigma_{Ns}^2\sigma_{Ni}^2 + \sigma_{Ns}^2 n_{Ni}^2 + n_{Ns}^2\sigma_{Ni}^2 \\
& +\sigma_{Ns}^2\sigma_{NA/D}^2 + \sigma_{Ns}^2 n_{NA/D}^2 + n_{Ns}^2\sigma_{NA/D}^2 \\
& +\sigma_{NTQ}^2\} + \sigma_{NTQF}^2.
\end{aligned}
$$

All the noise components can be replaced with their respective substitutes to obtain an equation for the output noise power in terms of the input noise power and other parameters describing the system.

Appendix B

Rules for Uncorrelated, Independent Processes

The following derivations include two processes A and B. These processes are independent (and therefore uncorrelated) and have means n_A, n_B and variances σ_A^2 and σ_B^2.

Multiplication $A \times B$

$$\begin{aligned} n_{AB} &= E[AB] \\ &= E[A]E[B] \\ &= n_A n_B \end{aligned} \tag{B.1}$$

Note: If A and B are independent and uncorrelated then $E[AB] = E[A]E[B]$.

$$\begin{aligned} \sigma_{AB}^2 &= E[(AB)^2] - E[AB]^2 \\ &= E[A^2B^2] - E[AB]^2 \\ &= E[A^2]E[B^2] - E[A]^2E[B]^2 \end{aligned} \tag{B.2}$$

$$
\begin{aligned}
&= (\sigma_A^2 + n_A^2)(\sigma_B^2 + n_B^2) - n_A^2 n_B^2 \\
&= \sigma_A^2\sigma_B^2 + n_A^2\sigma_B^2 + n_B^2\sigma_A^2 + n_A^2 n_B^2 - n_A^2 n_B^2 \\
&= \sigma_A^2\sigma_B^2 + n_A^2\sigma_B^2 + n_B^2\sigma_A^2
\end{aligned}
$$

Note: If A and B are independent, uncorrelated processes then A^2 and B^2 are also independent, uncorrelated processes.

Addition $A + B$

$$
\begin{aligned}
n_{A+B} &= E[A+B] \qquad \text{(B.3)} \\
&= E[A] + E[B] \\
&= n_A + n_B
\end{aligned}
$$

$$
\begin{aligned}
\sigma_{A+B}^2 &= E[(A+B)^2] - E[A+B]^2 \qquad \text{(B.4)} \\
&= E[A^2 + 2AB + B^2] - E[A]^2 - 2E[A]E[B] - E[B]^2 \\
&= E[A^2] + 2E[AB] + E[B^2] - E[A]^2 - 2E[A]E[B] - E[B]^2 \\
&= E[A^2] - E[A]^2 + E[B^2] - E[B]^2 + 2E[AB] - 2E[A]E[B] \\
&= \sigma_A^2 + \sigma_B^2 + 2E[A]E[B] - 2E[A]E[B] \\
&= \sigma_A^2 + \sigma_B^2
\end{aligned}
$$

Subtraction $A - B$

$$
\begin{aligned}
n_{A-B} &= E[A-B] \qquad \text{(B.5)} \\
&= E[A] - E[B] \\
&= n_A - n_B
\end{aligned}
$$

$$
\sigma_{A-B}^2 = E[(A-B)^2] - E[A-B]^2 \qquad \text{(B.6)}
$$

$$
\begin{aligned}
&= E[A^2 - 2AB + B^2] - E[A]^2 + 2E[A]E[B] - E[B]^2 \\
&= E[A^2] - 2E[AB] + E[B^2] - E[A]^2 + 2E[A]E[B] - E[B]^2 \\
&= E[A^2] - E[A]^2 + E[B^2] - E[B]^2 - 2E[AB] + 2E[A]E[B] \\
&= \sigma_A^2 + \sigma_B^2 - 2E[A]E[B] + 2E[A]E[B] \\
&= \sigma_A^2 + \sigma_B^2
\end{aligned}
$$

Appendix C

Matlab M-Files

BER_OUT.M

```
%     Output SNR Calculation Program for
%     RNS based system and traditional DSP
%     based system
%
%              by Todd Hunter
%              on July 15, 1996

%     Setup and Title Screen

clear
disp('Output SNR Calculation')

% Input required parameters for calculation

% SNRIN    =        SNR of input signal (in dB)
```

```
% BITS    =       number of bits used in quantizer
% Fs      =       Relative Sampling frequency
% DELTAF  =       ratio of (Fhi-Flo)/Fs
% C       =       Mixer Amplitudes
% G       =       Gain due to linear differentiators
% PT      =       Total Power
% Pn      =       Total Noise Power
% Ps      =       Total Desired Signal Power
% Po      =       Power level of Output Desired Signal
% Ac      =       Peak level of Desired Signal
% Range   =       + (positive) Range of Quantizers
% Flpf    =       Relative frequency of LowPass Filters

% For all noises

% NNxx    =       Mean of noise xx
% VNxx    =       Variance of noise xx
% xxin    =       Input noise
% xxA/D   =       Quantization Noise from A/D
% xxC/S   =       Quantization Noise S/C mixers
% xxO     =       Output Noise Signal
% xxTx    =       Truncation noise
% xxxYY   =       Location of noise source
% xxxI/Q  =       Noise injected after Mixers
%                         (Inphase/Quadrature)
% xxxIF/QF=       Noise injected after LowPass Filters
% xxxIP/QP=       Noise injected after Differentiators
% xxxIM/QM=       Noise injected after Multipliers
% xxxO    =       Noise injected at Output
```

```
% bits_rns_a2d = input('Number of Bits RNS A/D - ');
% bits_rns = input('Number of Bits RNS - ');
% bits_out = input('Number of Bits RNS Output - ');
% bits_dsp = input('Number of bits DSP - ');



DELTA_F = .04;   %input('Delta F - ');
C = 1;
Range = 1;
SQ2 = 2.^(.5);

% Setup Normal Distribution

y = -20:.001:20;
dist = 1/sqrt(2*pi)*exp((-1*y.^2)/2);

% Input System Parameters

% Param_in = [ bits_rns_a2d1  bits_rns1  bits_out1  bits_dsp1
% bits_rns_a2d2  bits_rns2  bits_out2  bits_dsp2];

Param_in =     [6 6 6 6; 6 6 8 7; 6 8 8 8; 8 6 6 9;
8 6 8 10; 8 8 8 12; 8 8 10 14; 8 8 16 16];

for Param = 1:length(Param_in(:,1))

bits_rns_a2d = Param_in(Param,1); bits_rns = Param_in(Param,2);
```

```
bits_out = Param_in(Param,3); bits_dsp = Param_in(Param,4);


SNR_IN = 1:10;

    G = 1 * sin(2 * pi * DELTA_F);
    % Let PT = 1/2 Range.^2*.8 = Ps + Pn
    Ps = .5*.8.* Range.^2./(1+1./10.^(SNR_IN./10));
    Ac = (2 * Ps) .^ (.5);
    Pn = Ps ./ (10 .^ (SNR_IN ./ 10));
    Flpf = DELTA_F * 1.75;
    Fs = 1;
    Delta_RNS_a2d = Range ./ 2 .^ (bits_rns_a2d - 1);
    Delta_RNS = Range ./ 2 .^ (bits_rns - 1);
    Delta_RNS_out = Range ./ 2 .^ (bits_out - 1);
    Delta_DSP = Range ./ 2 .^ (bits_dsp - 1);
    Po = 4 * Ac .^ 4 * C .^ 4 * G .^ 2;


% Parameters for RNS based Calculation

    Nz = 0;                          %  mean of noises
    NNin = 0;                        %  mean of input noise
    NNAD = Nz; NNc = Nz; NNs = Nz;   %  mean of q noises

    VNin = Pn .^ (1);                %  var of input noise
    VQ_in= Delta_RNS_a2d .^ 2 ./ 12; %  var of q noises
    VQ = Delta_RNS .^ 2 ./ 12;       %  var of q noises
    VNAD = VQ_in; VNC = VQ; VNS = VQ; %  var of q noises
```

```
Tn = 0;
NNTI = Tn; NNTQ = Tn; NNTIF = Tn; NNTQF = Tn;
NNTIP = Tn; NNTQP = Tn;
NNTIM = Tn; NNTQM = Tn;
NNTO = Delta_RNS_out ./ [-2];


Tv = 0;
VNTI = Tv; VNTQ = Tv; VNTIF = Tv; VNTQF = Tv;
VNTIP = Tv; VNTQP = Tv; VNTIM = Tv; VNTQM = Tv;
VNTO = Delta_RNS_out .^ 2 ./ 12;


% Calculate noise components present after LowPass Filters

NNI = Flpf ./ Fs .* (2 .* SQ2 .* C .* (NNin + NNAD)
      + SQ2 .* Ac .* NNc + NNc .* NNin + NNc .* NNAD + NNTI)
      + NNTIF;
NNQ = Flpf ./ Fs .* (2 .* SQ2 .* C .* (NNin + NNAD)
      + SQ2 .* Ac .* NNs + NNs .* NNin + NNs .* NNAD + NNTQ)
      + NNTQF;

VNI = Flpf ./ Fs .* (2 .* SQ2 .* C .* (VNin + VNAD)
      + SQ2 .* Ac .* VNC + VNC .* VNin + VNC .* NNin .^ 2
      + NNc .^ 2 .* VNin + VNC .* VNAD + VNC .* NNAD .^ 2
      + NNc .^ 2 .* VNAD + VNTI) + VNTIF;
VNQ = Flpf ./ Fs .* (2 .* SQ2 .* C .* (VNin + VNAD)
```

```
        + SQ2 .* Ac .* VNS + VNS .* VNin + VNS .* NNin .^ 2
        + NNs .^ 2 .* VNin + VNS .* VNAD + VNS .* NNAD .^ 2
        + NNs .^ 2 .* VNAD + VNTQ) + VNTQF;


% Calculate noise components present at output

    NNO = (NNQ .* NNTIP) - (NNI .* NNTQP)
          - SQ2 .* Ac .* C .* (NNTIP + NNTQP)
          - SQ2 .* Ac .* C .* G .* (NNQ - NNI)
          + NNTIM - NNTQM + NNTO;

    VNO = 4 .* VNQ .* VNI + 4 .* VNQ .* NNI .^ 2
          + 4 .* NNQ .^ 2 .* VNI
          + (VNQ .* VNTIP + VNQ .* NNTIP .^ 2
             + NNQ .^ 2 .* VNTIP)
          + (VNI .* VNTQP) + (VNI .* NNTQP .^ 2)
          + (NNI .^ 2 .* VNTQP);
    VNO = VNO + SQ2 .* Ac .* C .* (2 .* VNI + VNTIP
            + 2 .* VNQ + VNTQP)
          + SQ2 .* Ac .* C .* G .* (VNQ + VNI)
          + VNTIM + VNTQM + VNTO;

    PNO = VNO + NNO .^ 2;


    SNR_OUT_RNS = 10 .* log10(Po ./ PNO);
```

```
% Calculate RNS Probability of Error

    x = (sqrt(Po)-NNO)./VNO;
    for loop = 1:length(x)
        prob_rns(Param, loop) =
          1 - sum(dist(1:(20+x(loop))*(1/.001)))*.001;
    end
    prob_rns;


% Parameters for traditional DSP based Calculation

    Nz = 0;                          %  mean of noises
    NNin = 0;                        %  mean of input noise
    NNAD = Nz; NNc = Nz; NNs = Nz;   %  mean of q. noises

    VNin = Pn .^ (1);                %  var. of input noise
    VQ = Delta_DSP .^ 2 ./ 12;       %  var. of q. noises
    VNAD = VQ; VNC = VQ; VNS = VQ;   %  var. of q. noises


    Tn = Delta_DSP ./ [-2];
    NNTI = Tn; NNTQ = Tn; NNTIF = Tn; NNTQF = Tn;
    NNTIP = Tn; NNTQP = Tn;
    NNTIM = Tn; NNTQM = Tn;
    NNTO = Tn;

    Tv = Delta_DSP .^ 2 ./ 12;
    VNTI = Tv; VNTQ = Tv; VNTIF = Tv; VNTQF = Tv;
```

```
VNTIP = Tv; VNTQP = Tv; VNTIM = Tv; VNTQM = Tv;
VNTO = Tv;


% Calculate noise components present after LowPass Filters

NNI = Flpf ./ Fs .* (2 .* SQ2 .* C .* (NNin + NNAD)
      + SQ2 .* Ac .* NNc + NNc .* NNin + NNc .* NNAD + NNTI)
      + NNTIF;
NNQ = Flpf ./ Fs .* (2 .* SQ2 .* C .* (NNin + NNAD)
      + SQ2 .* Ac .* NNs + NNs .* NNin + NNs .* NNAD + NNTQ)
      + NNTQF;

VNI = Flpf ./ Fs .* (2 .* SQ2 .* C .* (VNin + VNAD)
      + SQ2 .* Ac .* VNC + VNC .* VNin + VNC .* NNin .^ 2
      + NNc .^ 2 .* VNin + VNC .* VNAD + VNC .* NNAD .^ 2
      + NNc .^ 2 .* VNAD + VNTI) + VNTIF;
VNQ = Flpf ./ Fs .* (2 .* SQ2 .* C .* (VNin + VNAD)
      + SQ2 .* Ac .* VNS + VNS .* VNin + VNS .* NNin .^ 2
      + NNs .^ 2 .* VNin + VNS .* VNAD + VNS .* NNAD .^ 2
      + NNs .^ 2 .* VNAD + VNTQ) + VNTQF;


% Calculate noise components present at output

NNO = (NNQ .* NNTIP) - (NNI .* NNTQP)
      - SQ2 .* Ac .* C .* (NNTIP + NNTQP)
      - SQ2 .* Ac .* C .* G .* (NNQ - NNI) + NNTIM - NNTQM
      + NNTO;
```

```
    VNO = 4 .* VNQ .* VNI + 4 .* VNQ .* NNI .^ 2
          + 4 .* NNQ .^ 2 .* VNI
          + (VNQ .* VNTIP + VNQ .* NNTIP .^ 2
             + NNQ .^ 2 .* VNTIP) + (VNI .* VNTQP)
          + (VNI .* NNTQP .^ 2) + (NNI .^ 2 .* VNTQP);
    VNO = VNO + SQ2 .* Ac .* C .* (2 .* VNI + VNTIP + 2 .* VNQ
          + VNTQP) + SQ2 .* Ac .* C .* G .* (VNQ + VNI) + VNTIM
          + VNTQM + VNTO;

    PNO = VNO + NNO .^ 2;


    SNR_OUT_DSP = 10 .* log10(Po ./ PNO);

% Calculate DSP Probability of Error

    x = (sqrt(Po)-NNO)./VNO;
    for loop = 1:length(x)
        prob_dsp(Param, loop) =
            1 - sum(dist(1:(20+x(loop))*(1/.001)))*.001;
    end
    end


figure(1);
semilogy(SNR_IN, prob_dsp);
axis([1 10 .00000001 .100]);
title('DSP BER Results from Model');
```

```
ylabel('Log (BER)');xlabel('SNR In (dB)');
legend('6','7','8','9','10','12','14','16');

figure(2)
semilogy(SNR_IN, prob_rns(1:4,:));
axis([1 10 .00000001 .100]);
title('RNS BER Results from Model');
ylabel('Log (BER)');xlabel('SNR In (dB)');
legend('6 6 6','6 6 8','6 8 8','8 6 6');

figure(3)
semilogy(SNR_IN, prob_rns(5:8,:));
axis([1 10 .00000001 .100]);
title('RNS BER Results from Model');
ylabel('Log (BER)');xlabel('SNR In (dB)');
legend('8 6 8','8 8 8','8 8 10','8 8 16');
```

Rec1.m

```
clear

bits = 11;% DSP quantization/truncation
bits_r = 8;% RNS A2D
bits_r_o = 6;% RNS Output

bits_in = 1000;%        # bits/cycle
fs = 9600;%             Sampling frequency
fc = 1700;%             FSK center frequency
df = 400;%              FSK peak freq. deviation
R = 1200;%              Bit rate
Ac = 1;%                FSK signal's peak amplitude
Gmix = 1;%              Gain of mixers
Glpf = 2;%              Gain of Lowpass Filters
Gdiff = 2;%             Gain of Differentiators
Gmult = 2;%             Gain of Multipliers
fclpf = 700;%           LPF cutoff frequency
order = 31;%            LPF Order

samples = floor(bits_in*fs/R);%   # of samples req'd
Pmax = .8*(1/2);% Max Power at input is 1/2



SNR_set = [1 2 3 4 5 6 7 8 9 10];
```

```
err = zeros(size(SNR_set));%      DSP Error Count
err_r = zeros(size(SNR_set));%    RNS Error Count

for SNR_loop = 1:length(SNR_set)
for loop = 1:1000

SNR_db = SNR_set(SNR_loop)

% Calculate Noise and Signal Powers

SNR = 10^(SNR_db/10);
Ac = sqrt(2*Pmax*SNR/(1+SNR));
No = 2*Pmax/fs/(1+SNR);
Sigma_2 = No*fs/2;
Sigma = sqrt(Sigma_2);


% Generate Input FSK Signal + Noise


[in, actual_bits] = gen(fc, fs, df, R, Ac, bits_in);
noise = gen_noi(samples);
noise = noise*Sigma;

in = in+noise;


% Clip + Quantize Input Signal
```

```
in = clipper(in);

sig_in = quant(in, bits);
sig_in_r = quant(in, bits_r);


% Down Mixing


[sig_i, sig_q] = down(sig_in, fs, fc, Gmix, bits);
sig_i = trunc(sig_i, bits);
sig_q = trunc(sig_q, bits);

[sig_i_r, sig_q_r] = down(sig_in_r, fs, fc, Gmix, bits_r);


% Low Pass Filtering


sig_i = lpf(sig_i, fclpf, fs, order, Glpf, bits);
sig_q = lpf(sig_q, fclpf, fs, order, Glpf, bits);
sig_i = trunc(sig_i, bits);
sig_q = trunc(sig_q, bits);

sig_i_r = lpf(sig_i_r, fclpf, fs, order, Glpf, bits_r);
sig_q_r = lpf(sig_q_r, fclpf, fs, order, Glpf, bits_r);
```

```
% Differentiation


sig_i_d = diff1(sig_i, Gdiff, bits);
sig_q_d = diff1(sig_q, Gdiff, bits);
sig_i_d = sig_i_d * Gdiff;
sig_q_d = sig_q_d * Gdiff;
sig_i_d = trunc(sig_i_d, bits);
sig_q_d = trunc(sig_q_d, bits);

sig_i_d_r = diff1(sig_i_r, Gdiff, bits_r);
sig_q_d_r = diff1(sig_q_r, Gdiff, bits_r);
sig_i_d_r = sig_i_d_r * Gdiff;
sig_q_d_r = sig_q_d_r * Gdiff;


% Multiply and Add


sig_i_m = sig_i_d.*sig_q;
sig_q_m = sig_q_d.*sig_i;
sig_i_m = sig_i_m * Gmult;
sig_q_m = sig_q_m * Gmult;
sig_i = trunc(sig_i_m, bits);
sig_q = trunc(sig_q_m, bits);

sig_i_m_r = sig_i_d_r.*sig_q_r;
sig_q_m_r = sig_q_d_r.*sig_i_r;
sig_i_m_r = sig_i_m_r * Gmult;
```

```
sig_q_m_r = sig_q_m_r * Gmult;


out = sig_i-sig_q;
out = trunc(out, bits);

out_r = sig_i_r-sig_q_r;
out_r = trunc(out_r, bits_r_o);


% Remove distorted end bits


bad_samples = ceil(order/(fs/R)/2)*(fs/R);
out = out(bad_samples+ceil(order/2)+1:samples);
out_r = out_r(bad_samples+ceil(order/2)+1:samples);

actual_bits = actual_bits(bad_samples+1:
                samples-ceil(order/2)-0);



% Sample Output


sig_out = out(round(fs/R/2):round(fs/R):
            samples-(bad_samples+ceil(order/2)+1));
sig_out_r = out_r(round(fs/R/2):round(fs/R):
            samples-(bad_samples+ceil(order/2)+1));
```

```
bit_out = actual_bits(round(fs/R/2):round(fs/R):
            samples-(bad_samples+ceil(order/2)+1));
error = sum(abs((sign(sig_out+.0000001)
            -sign(bit_out+.0000001))/2)) % Cycle Error DSP
error_r = sum(abs((sign(sig_out_r+.0000001)
            -sign(bit_out+.0000001))/2)) % Cycle Error RNS
no_bits_out = bits_in-2*(ceil(order/(fs/R)/2));

err(SNR_loop) = err(SNR_loop)+error;% Total Error DSP
err_r(SNR_loop) = err_r(SNR_loop)+error_r;% Total Error RNS



end
end

total_bits = loop*bits_in
SNR_set
err %        Total Error DSP
err_r %      Total Error RNS

save output2.mat -ascii
```

Gen.m

```
function [out, bits_mod] = gen(fc, fs, df, R, A, b)
%
% This function will generate the carrier frequency signal
% for input to the Cross-Correlator receiver.  The input
% parameters include:
% out = gen(fc, fs, df, R, A, b)
% fc = carrier frequency
% fs = sample frequency
% df = peak change in frequency
% R = bit rate
% A = peak amplitude of signal
% b = # of bits to be generated

M = fs/R;%                    # of bits required
bits = sign(rand(1,b)-.5);
bits_m =ones(M,1)*bits;
bits_mod = reshape(bits_m, 1, (b*M));
phase = cumsum(bits_mod);
phase = 2*pi*(fc/fs*(1:(b*M)) + df/fs*phase);
out = A*cos(phase+pi/9);
```

Gen_noi.m

```
function out = gen_noi(samples)
%
% This function generates the White Gaussian noise by
% generating 40 normally distributed noise patterns and
% applying the Central Limit Theorem.
% The noise generated has a Std dev of 1 and a mean of 0.
% out = gen_noi(samples)
% samples = number of samples required

randn('seed',sum(100*clock));
out = randn(40, samples);
out = mean(out);
out = out/std(out);
```

Clipper.m

```
function out = clipper(in)

% This function simply clips the input signal to ensure that
% no samples are >+1 or <-1.
% out = clipper(in)

clip_samples = fix(in);
clipped = in+clip_samples-(abs(clip_samples).*in);
out = clipped;
```

Down.m

```
function [in_phase, quad_phase] = down(in, fs, fc, G, bits)
%
% This function will perform the downmixing of the input
% signal using cos and sin waves of the frequency fc,
% sampled at fs.  The outputs from the cos and sin mixers are
% sent out as in_phase and quad_phase respectively.
% [in_phase, quad_phase] = down(in, fs, fc, G, bits)
% in = input signal
% fs = sampling frequency
% fc = center frequency of input signal
% G = mixer signal's gain factor
% bits = number of bits to quantize the cos and sin waves to

l = length(in);
in_phase = in.*quant(G*cos(2*pi*fc/fs*(1:l)), bits);
quad_phase = in.*quant(G*sin(2*pi*fc/fs*(1:l)), bits);
```

Trunc.m

```
function out = trunc(in, bits)
%
% This function will scale a number, truncate it and rescale
% it back down.
% trunc(in, bits)
% bits = # of signed bits

out = floor((2^(bits-1))*in)/(2^(bits-1));
```

Lpf.m

```
function out = lpf(in, fclpf, fs, order, G, bits)
%
% This function will lowpass filter the input signal
% out = lpf(in, fclpf, fs, order, G, bits)
% in = input signal
% fclpf = cutof frequency of LPF
% fs = sampling frequency
% order = order of filter used
% G = gain of the filter
% bits = number of bits to quantize the filter coefficients

b = fir1(order,(fclpf/(fs/2)));
b = G*quant(b, bits);
out = filter(b, 1, in);
```

Diff1.m

```
function out = diff1(in, G, bits)
%
% This function differentiates the input signal using a
% FIR filter with the coeficients [.55 .017 -.017 -.55].
% Note: there is no delay introduced into the signals.
% Differentiator has a gain of
% 1/6
% diff1(in, G, bits)
% in = input signal
% G = Gain for filter coefficients
% bits = number of bits to quantize the filter coefficients

b = [-1 0 1];
b = quant(b, bits)

n = [0 0 1]';
Gain = (-1*b)*sin(2*pi*400/9600*n);

out = filter((-1*b), 1, in);
out = [out(2:length(out)) 0];% remove  bit delay
```

Quant.m

```
function out = quant(in, bits)
%
% This function will quantize a number by scaling it by
% the number of bits, round it and rescale
% it back down.
%  out = a2d(in, bits)
% in = input signal]
% bits = # of signed bits used for quantization


out = round((2^(bits-1))*in)/(2^(bits-1));
```