PUNCTURED CONVOLUTIONAL CODING SCHEME FOR MULTI-CARRIER MULTI-ANTENNA WIRELESS SYSTEMS

by

Christopher Lamont Taylor

Memorandum No. UCB/ERL M01/27

27 July 2001

COVER

PUNCTURED CONVOLUTIONAL CODING SCHEME FOR MULTI-CARRIER MULTI-ANTENNA WIRELESS SYSTEMS

by

Christopher Lamont Taylor

Memorandum No. UCB/ERL M01/27

27 July 2001

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Punctured Convolutional Coding Scheme for Multi-Carrier Multi-Antenna Wireless Systems

by Christopher Lamont Taylor

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee:

Robert W. Brodersen

Professor Robert W. Brodersen
Research Advisor

7/27/01

Date

* * * * * * *

Professor Jan M. Rabaey
Second Reader

7/25/01

Date

Acknowledgment

It is amazing how much can be gained in such a short period of time. During my time here I have experienced the true benefit of higher learning and why Berkeley is special. It takes a special place and people to nurture and develop students to absorb the vast amount of knowledge that Berkeley professors, staff, and my fellow students have to offer. This nurturing has come in no small part, from my advisor Professor Bob Brodersen. His vision, patience, guidance, and endless encouragement have helped me when no one could. He also possesses a special quality found only in great teachers, a warmth and kindness that eliminates the intimidation factor between teacher and pupil. I also want to thank Professor Jan Rabaey for his co-development in the Berkeley Wireless Research Center, the best research center the University has to offer. I am grateful of the faculty, staff, and researchers I have had the opportunity to interact with they were invaluable in assisting me in anything that I asked and those things that I should be asking.

Certain organizations and people that I have met along my course here deserve to be mentioned.

Paul Husted, Andy Klein, Ada S.Y. Poon, and Ning Zhang, some of the founding members of the small but diligent communications algorithm group. I have learned more from you than from any class and without you I would be still be sitting at Intel 79.

The BARF team (too many to name), it is ironic how a bunch of analog designers were so beneficial to the knowledge base of a communications guy. Their vast knowledge ranges far beyond the length of this project.

Thanks to the SSHAFT team who provided me, and anyone with a viable algorithm, a way to realize my design. They provided a means for me to get a practical perspective on the effects of my design. Can you dig it...

I would like to thank the staff at the BWRC for the up keep of the center, from a state-of-the-art design environment to lunch on Friday. I would like to thank DARPA for

funding my research. I also thank the company sponsors who invested in the BWRC and sent their leading researchers to talk with us on the ways and wonders of industry.

Contents

Figures

Tables

1

Introduction

1.1 Motivation

The work presented here is to begin to think about ways for future wireless communication systems to thrive with the growing shortage of signal bandwidth. Are there ways for existing wireless systems to share and use a common bandwidth efficiently and without interference among each other? Bandwidth for today's wireless technology is at a premium, and in some cases is the bottleneck in delivering high data rate services to people worldwide. The overall theme of this research is to create a framework that consists of complex wireless systems that demonstrate the gain from Multiple Element Array (MEA) algorithms to provide high data rates at high bandwidth efficiency.

Algorithm implementations in CMOS are expected to carry the brunt of the processing power needed; therefore it is essential that communication systems of the future be developed with implementation issues in mind so that there is a clear understanding of

what is realizable and what is not. It is therefore equally important to create a framework to determine if today's advanced algorithms can be rapidly implemented in CMOS.

1.2 Research Goals

A system level simulation and analysis on the effects of a variable rate error correction technique. The simulation blocks should be modular in design so that they can be used in conjunction with other blocks to create any advanced digital communication system. The wireless systems created here use MEA algorithm techniques [1]. The system is decomposed into key building blocks. Each block is represented as a floating-point model then as a fixed-point model. The floating-point model simulates the optimal functionality of the system. The fixed-point model is functional equivalent to the floating-point model and should model the bit level accuracy of the model. A logic equivalency between the fixed-point model and behavioral VHDL is determined which in turn is used in a rapid design flow SSHAFT [2] process to generate actual chip design.

1.3 Thesis Organization

There are 5 chapters to this report. Chapter 2 presents the framework that the proposed error correction scheme is meant for. The theory behind the specific blocks making up the proposed scheme is discussed in Chapter 3. Chapter 4 discusses the floating-point and fixed-point SIMULINK implementation and issues involving hardware implementation. Also, discussed is the hardware implementation of the Viterbi decoder design in MODULE COMPILER™. A summary of the project and a discussion of possible future work are discussed in Chapter 5.

2

The MCMA Framework

This chapter provides a description of the Multiple-Carrier Multiple-Antenna (MCMA) framework. The framework is meant to build advanced next generation communication algorithms for wireless Local Area Networks (LAN's) applications. The idea is to provide a general framework to allow the creation of (MCMA) systems at a high level of abstraction from a consortium of building blocks. The blocks are designed in SIMULINK and should be modular in design and "flexible" enough so that major parameters can be changed with relative ease. SIMULINK allows for this type of design, it gives systems designers the flexibility to build different algorithm blocks and build a variety of complex systems to meet a vast array of system applications. This section details the specifications for such a system.

2.1 MCMA System Specifications

The ultimate goal is to provide high data rates asynchronously to multiple users in an indoor local wireless environment. The building blocks should be "general" enough to be placed in a variety of different system configurations. If the inputs and outputs of a particular algorithmic block are specified then any system can be arbitrarily designed

with a high level of confidence the block will perform as specified. The material in this section will discuss the basic understanding of the "key" blocks needed for a complete system simulation. Assumptions are made here for the explicit purpose of a general understanding of the blocks. Our current system is based on previous research, "The Hornet Radio Proposal" [3]. Table 2.1 gives a list of the major specifications for the MCMA framework.

Specification	**Value**	
Carrier Frequency	5 GHz	f_c
Sample Rate	15 MHz	f_s
Modulation	QPSK	
Transmission/Multiple Access Scheme	Asynchronous	
Uplink	TDD/FDMA	
Downlink	OFDMA	
Number of carriers	64	N
Carrier Spacing	15MHz/64 = 234.4	
Cyclic prefix length	6	c_p
Uncoded Symbol Rate	214.28 kHz	T_s
Number of transmit antennae	4	A
Number of receive antennae	4	A

Table 2.1 System Specifications

Future wireless systems will likely be placed in the 5GHz spectrum, hence the specification here. An existing analog front end can handle a sampling rate (f_s) of 15MHz, thus the system bandwidth is 15MHz. Each user is synchronous in the downlink and asynchronous in the uplink. For the downlink, the system will utilize Orthogonal Frequency Division Multiple Access (OFDMA) as the multiple access scheme. The uplink has not been determined. One idea is a form of Frequency Division Duplexing (FDD) using polyphase filter banks at the basestation, see [4]. The current system can handle up to 64 subcarriers (N), where $N \geq$ the number of users (M). The number of antennae used is 4 transmit and 4 receive for each user and is based on work by another BWRC researcher [5]. The system will transmit based on a packet-based data protocol and the transmission method will be point to multipoint and provide a guaranteed throughput to each user. The bit error rate performance required is 10^{-5}.

The complexity of the system is greatly reduced, for implementation reasons, by allocating only a small number of subcarriers per user, as described in Section 2.2

2.2 OFDM

OFDM is a powerful modulation technique that increases bandwidth efficiency and eases the removal of inter-symbol interference (ISI) and inter-channel interference ICI. Implementations advances in FFT technology enable OFDM to be implemented in hardware, for a large number of subcarriers. It is a common misconception that orthogonal frequency division multiplexing (OFDM) is a multi-access technique. Rather, it is a technique to combat frequency selective fading [3]. The basic idea of OFDM is to divide spectrum into several subcarriers. By dividing the bandwidth into smaller channels, narrowband channels are created from a wideband environment. These narrowband subcarriers experience flat fading, which is an important characteristic for today's multiple antenna algorithms. To obtain high spectral efficiency the frequency response of the subcarriers are overlapping and orthogonal thus the name OFDM. The orthogonality principle of OFDM is key because it helps to eliminate (ISI) and ICI. This is done by introduction of a cyclic prefix [6]. The cyclic prefix is the last part of the OFDM symbol is pre-pended to the beginning the same OFDM symbol. The cyclic prefix benefit is twofold: it acts as a guard space between subcarriers to eliminate ISI and by the orthogonality principle it eliminates ICI. ISI is eliminated because the channel spread created by the channel is contained in the cyclic prefix, which must be longer than the impulse response of the channel. The sampled output at the receiver ignores samples from [0, T_{cp}], where T_{cp} is the length of the cyclic prefix, therefore, the sampling interval for the receiver correlator is [T_{cp}, T_s], where T_s is the symbol rate. ICI is eliminated by the transformation of the linear convolution in the channel to a cyclic convolution. This is due to the cyclic prefix being longer than the channel impulse response. Fig. 2.1 is a high-level block diagram of an OFDM transceiver. The input symbols are **X**(n), where $n = 0,1,2,\cdots,N$, N being the number of subcarriers. These subcarriers are the complex-valued symbols from the modulator. The IFFT is taken on modulated symbols, the cyclic prefix is added to combat the multipath of the channel. After a serial to parallel

conversion the symbol is sent through the channel. The receiver is essentially the reverse of the transmitter chain; the output Y(n) can be written as $\mathbf{Y}(n) = \mathbf{X}(n) * h(n) + w(n)$, where $w(n)$ is the additive noise term.

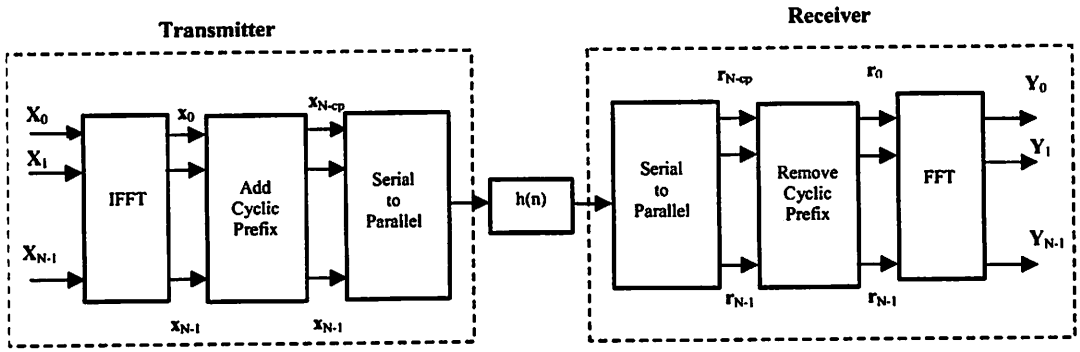

Figure 2.1 OFDM Block Diagram

Orthogonal frequency division multiple access (OFDMA) is chosen to be the multiple access scheme. It permits the use of narrowband MEA algorithms, which are the focus of algorithms studied at the BWRC. Due to the complexity problem as described in [3] at least one frequency channel is allocated per user. This keeps the number of MEA processing algorithms proportional to the number of users. Given $N = 64$ subcarriers and $M = 64$ users, the total number of processing units at the basestation is $NM = 4096$. This number of processing unit is not practical for today's systems. If each user transmits across the entire 15 MHz band, the user must have a MEA unit for each subcarrier frequency, since there are 64 subcarriers and 64 users the total number of MEA processing units is 4096. A simpler approach is to apply at least one subcarrier to a given per user, thus only 1 MEA unit is needed per user. The basestation no longer requires NM units, but needs only M MEA units. This allows for a more realistic overall system development and demonstration of proof of concept. There are other system parameters that must be determined to establish the performance of the system. The length of the cyclic prefix is a function of the excess RMS delay spread, d, [7] and the sampling rate, f_s. To ensure the removal of ISI, the cyclic prefix length should be

$$c_p \geq \lceil f_s \cdot \sigma_\tau \rceil \tag{2.1}$$

The excess RMS delay spread (d) for an indoor channel is around 200 ns, and it is common practice when creating practical systems σ_τ be twice the excess RMS delay, thus $\sigma_\tau = 2d$. σ_τ is 400 ns, so to satisfy (2.1), $c_p \geq 6$. The single-user data rate depends on the number of carriers and the sampling rate. Increasing the number of subcarriers increases the number of users at the expense of lower single-user data rates. The uncoded single-data rate is [3]:

$$\begin{aligned} T_s &\approx \left(\frac{f_s}{N}\right) \cdot (\textit{OFDM symbol efficiency due to cyclic prefix}) \\ &\approx \left(\frac{f_s}{N}\right)\left(\frac{N}{N+c_p}\right) \end{aligned} \tag{2.2}$$

The efficiency of an OFDM symbol due to the cyclic prefix in practice should be no less than 80%. Given $N = 64$, $T_s = 214.28$ kSymbols/sec. This is a relatively small data rate for each user. The value doesn't take into account other factors such as the multiplicative improvement with the increase in the number of transmit and receive antennas and the better bandwidth efficiency due to higher order modulation schemes. Theoretically, with multiple antennae (A), the symbol rate is a multiplicative factor of A larger, which has been shown by simulation in [3]. Higher order modulation schemes have not been implemented because the timing synchronization and frequency offset technique has not yet been finalized. For implementation simplicity the MCMA framework currently sacrifices frequency diversity by on allocating at least only subcarrier to each user. The hope is the gain in spatial diversity by the multiple antennae and higher modulation will compensate.

2.3 QPSK Modulation

Quadrature phase-shift keying (QPSK) is a signaling scheme where data is modulated onto the phase of the carrier signal and is represented by [8]

$$\begin{aligned} s_m(t) &= g(t)\cos\left[\frac{2\pi}{M}(m-1)\right]\cos(2\pi f_c t) - g(t)\sin\left[\frac{2\pi}{M}(m-1)\right]\sin(2\pi f_c t), \\ & m = 1,2,\cdots,M,\ 0 \leq t \leq T \end{aligned} \tag{2.3}$$

where $g(t)$ is the signal pulse, f_c is the carrier frequency, M is the number of waveforms that carry the transmitted information, and T is the symbol period. The mapping of $k = \log_2(M)$ information bits to the $M = 4$ possible phases is done by Gray encoding. Gray encoding maps bits in adjacent positions so that they differ by one bit position. This allows the most likely errors caused by noise will result in a single bit error in the k-bit symbol. A constellation where the initial phase is $\pi/4$ is used here and is discussed in Section [4.2], see Fig. 2.2.

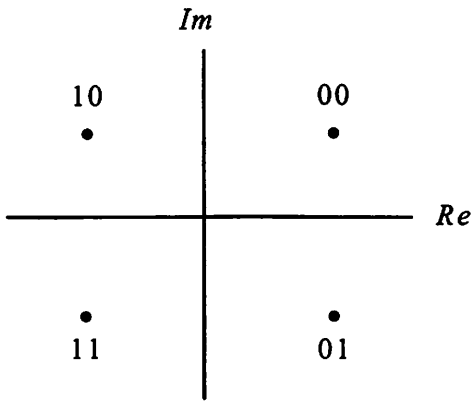

Figure 2.2 QPSK Constellation

QPSK was chosen as the modulation scheme for the following:

- The BWRC analog group has an existing platform built for QPSK
- QPSK has inherent properties of reliability at high noise levels, and a bounded constant envelope waveform signature
- The timing/synchronization scheme has not yet been specified for the framework
- QPSK is relatively less stringent than the higher order modulation levels

It is optimal to use higher order modulation schemes to obtain higher user data rates. The analog front-end model and the system level description on what the modulation scheme is heavily determined by the design of the analog front-end and the synchronization method. The specifications of the analog front end can handle have not been answered yet.

2.4 Timing/Synchronization and Frequency Offset

For simulation purposes perfect synchronization is assumed. The method to handle the synchronization and phase error of the system is still under development. The techniques currently being explored can be classified into two categories: pilot symbol based and cyclic prefix based. In pilot assisted schemes, a sequence of known pilot symbols is sent before any data symbols and the receiver correlates the received signal with a local copy of the known pilot symbols [9]. In cyclic prefix based techniques, the last L (where L is the length of the cyclic prefix) samples of an OFDM symbol are identical to the cyclic prefix (the first L samples). The receiver correlates between two sets of received signals that are N samples apart and since correlation is a similarity measure, the maximum will be achieved at the correct prefix position and synchronization information can be calculated [2]. Pilot based synchronization algorithms generally perform well, but creates overhead at the expense of non-information carrying training symbols. Cyclic prefix based correlation is inherent in OFDM and thus does not require any overhead. However, since the correlation is done only over a set of L samples, which is relatively short compared to the length of an OFDM symbol. Noise will significantly affect performance, especially for small SNR scenarios. A current method under construction at the BWRC is a combination of both techniques.

2.5 MEA Algorithms

Multiple Element Array algorithms are algorithms designed for wireless systems that have multiple transmit or receive antennas [1]. There are a couple of MEA algorithms that are being explored for use in MCMA framework. The Single Value Decomposition (SVD) algorithm is a narrowband MEA algorithm based on singular vale decomposition described exhaustively in [5]. The other is implementable approach to Foschini's QR Decomposition [10] for combining multiple antennas [11]. MEA algorithms are attractive because they assume the channel is narrowband. This assumption is key given in this design given each user's subcarrier frequency is considered narrowband. Given

that most MEA algorithms are been designed for narrowband channels there is a vast array of algorithms to choose from that will fit in the MCMA framework.

2.6 Error Control Scheme

A system level simulation and analysis of a variable-rate coding scheme to optimize performance is discussed. A variable-rate coding scheme is designed to offer optimal error protection levels to users based on the channel condition. As the channel is time varying it is foreseeable that different users (subcarriers) will experience different channel responses. As the code rate increases so does the throughput for a given user. It is optimal to provide better error protection to users who experience a "bad" channel condition at a given time and very little error protection to users who experience a "good" channel condition at a given time. Fig. 2.3 is an overview of the major blocks of a MEA system architecture, synchronization is not included since it is assumed to be perfect.

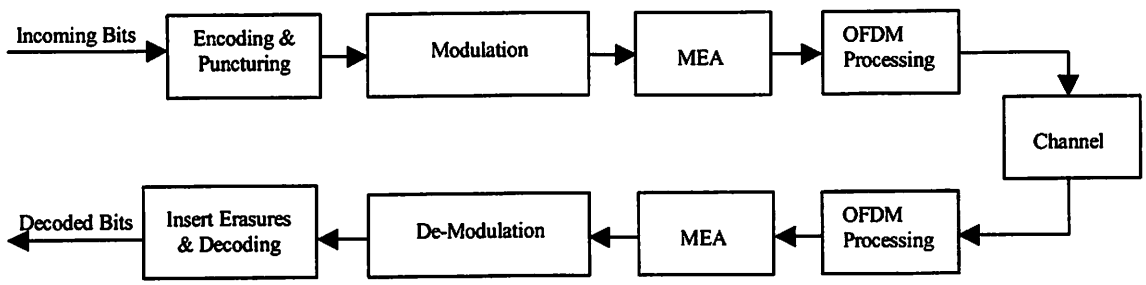

Figure 2.3 MEA System Architecture

2.6.1 Encoding and Puncturing

The encoding process is the first major signal processing that happens and decoding is the last of the signal processing done. Fig. 2.4 are the subsystems that comprise the error correction encoding used for this project.

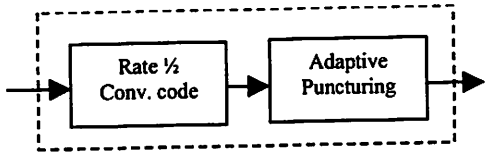

Figure 2.4 Encoding and Puncturing Blocks

A rate 1/2 convolutional encoder with a constraint length of nine and generator polynomial [753 561] is used for the encoding process. Puncturing is used to create the needed variable coding rates to provide various error protection levels to the users of the system. The different rates used are rate 1/2, rate 2/3, rate 3/4, and rate 7/8. The choosing of these rates is determined by increased throughput as described in Section 4.2.2

2.6.2 Erasure Insertion and Quantization

Figure 2.5 comprise the sub-blocks that make up the decoding process.

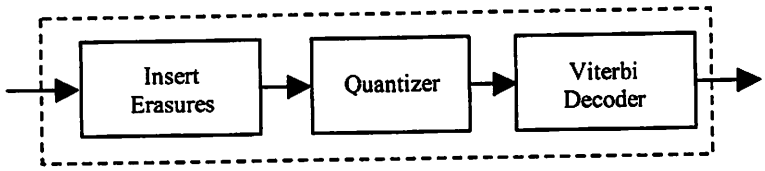

Figure 2.5 Erasure and Decoding Blocks

Erasure bits have to be inserted into the punctured data sequence to return the symbol rate of the system back to rate 1/2. These erasure bits are binary 0's inserted into the bit positions that were deleted by the puncturing process. Four-level soft-decision decoding is for the decoder and is supplied by the Quantizer block. The output values of the block are integer values [0,15].

2.6.3 Viterbi Decoder

Viterbi decoding is ideal for punctured convolutional codes because it doesn't require the exponential complexity that is incurred when trying to use higher rate codes. The Viterbi algorithm is exponential in the constraint length and in the number of input bits into the

decoder. By puncturing and erasure insertion, the Viterbi decoder only operates on the metric on one input bit per encoded symbol instead of the higher numbers needed for the higher rate codes. The traceback length is usually 5-7 times greater than the constraint length. This ensures that the decoder has enough time-steps to properly decode the noisy input bits. Insert erasure bits due to puncturing causes the realization of an accurate decoding sequence to be extended due to the dummy 0's that are added. For punctured systems the traceback length has to be extended to compensate for the addition of these dummy bits. There is no determined metric to find the optimal traceback length for punctured codes; simulations for performance are usually used to determine the length. The major parameters of the Viterbi decoder are:

- 4-bit level quantization
- Constraint length (K) of nine, with generator polynomial [753 561]
- Traceback length of 120

The MCMA framework consists of libraries that contain key digital signal processing blocks to build a multitude of communications systems. The fundamental development of the MCMA framework was created with the SSHAFT design flow in mind. SIMULINK's high level of abstraction allows for multiple systems to be explored. Performance trade-offs can be made among competing systems. Using the SSHAFT design flow implementation issues can quickly be explored and the most feasible of the systems can rapidly be determined and implemented.

Appendix

A2.1 SSHAFT DESIGN FLOW

The SIMULINK to Silicon Hierarchical Automated Flow Tools (SSHAFT) is an automated design flow for single chip radios [2]. The goal is to develop a design methodology that targets the implementation of computational complex, mobile radio systems. The standard design flow for implementing direct mapped architectures onto ASIC's is plagued with the potential of underdetermined design times. The standard flow has three

phases that are handled by different design teams. Systems designers' design using high levels tools such as MATLAB® or simulators like Cadence SPW. This system level abstraction is used to show functional behavior of the algorithm. This design is passed to ASIC designers who map the design to a hardware language such as VHDL or Verilog. The design is finally handed to physical designers who synthesize the hardware code for fabrication. This procedure requires verification at each stage to make sure that the entire system design is encapsulated in each of the three abstraction levels. More to the point, any algorithmic modifications made at the system level alter the performance because the system designer does not have the information from the lower levels about power constraints or delay requirements [12].

The design flow here takes SIMULINK as the initial description and automatically generates a direct mapped architecture see Fig.2.6. In order for the flow to work design decisions are divided into four levels of specification:

- Function level: SIMULINK serves as the functional descriptor for each block in the chip
- Signal level: Physical signal descriptions can also be captured with SIMULINK. SIMULINK's fixed-point blockset allows the defining of wordlengths and characterization of hardware issues such as, quantization errors due to truncation or rounding.
- Circuit level: MODULE COMPILER (MC) is used to generate circuit level abstraction views and gate level descriptions like VHDL and Verilog. The architectural exploration inherent in MC allows the designer to trade off power, area, and delay for different circuit implementations.
- Floorplan level: Physical placement of each block is done by importing the netlist and abstraction views into Cadence's Design Planner™ floorplanning tool. When a block is added in SIMULINK and unplaced block will appear in Design Planner™. The user must manually place the circuit level generated blocks.

From Fig. 2.6 all the input blocks relate to the four levels of specification just described. The outputs from Fig. 2.6 encapsulate everything for power and timing estimates to

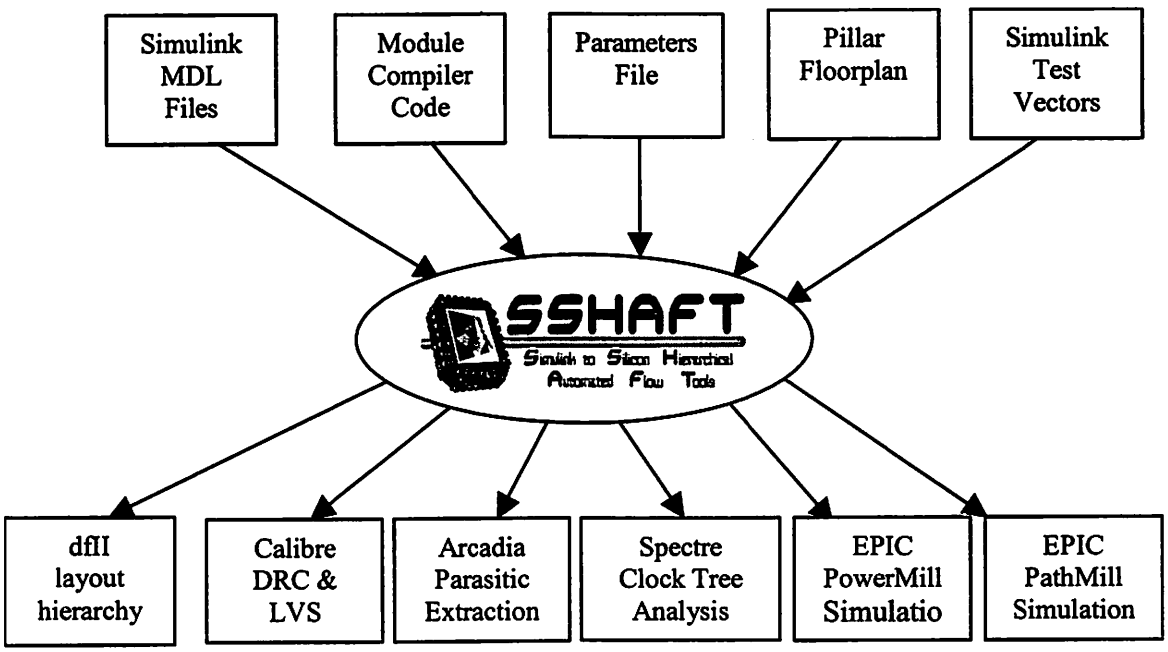

Figure 2.6 SSHAFT Design Flow

layout mask patterns needed for fabrication [2]. The flow allows system designers to create ICs that exhibit acceptable performance and consume minimal power without the need to cross abstraction boundaries.

From a single SIMULINK system description the SSHAFT design flow can automatically perform the subsequent steps needed to obtain a netlist, perform placement and routing, and obtain a layout that can be sent to foundries for fabrication.

3

Error Control Theory

3.1 Introduction

Modern digital communication system requirements are becoming more and more stringent with respect to error-free transmission. Next generation systems would like to offer Quality of Service (QoS) guarantees to users, this cannot be done unless more efficient error correction schemes can be implemented. There is also exponential growth in the Wireless industry for the same demands but that require less power. The research describe here falls in the latter area.

This chapter begins with the introduction of convolutional coding and it's structural properties. The Viterbi algorithm follows, with the metrics to determine the upper bounds on error probability. The final section focus includes punctured convolutional codes and their performance metrics. It is the work by Viterbi that promotes the motivation here to apply his algorithm for the decoding of this error correction scheme.

In 1948 Shannon published a paper describing the fundamental concepts and mathematical theory of information transmission. The paper was entitled "A Mathematical Theory of Communication" in the Bell Systems Technical Journal [13]. This paper is the basis at which the research community thinks about digital communication today. One of the theorems in this paper deals primary with error control coding for noisy channels, called the noisy channel coding theorem, which can be paraphrase as such,

Shannon's Noisy Channel Coding Theorem

> *With every channel we can associate a "channel capacity" C (bits/sec). There exist such error control codes that information can be transmitted at a rate below C (bits/sec) with an arbitrarily low bit error rate.*

Shannon believed that information bits could be transmitted near C with a small probability of error given that proper channel encoding and decoding could be attained. This sparked a great amount of work in the search of "good" error control codes and efficient decoding techniques to achieve these Shannon limits. Unfortunately all these codes fall short of the goals set by the noisy channel coding theorem, but great strides have been made in the search for more reliable data transmission. This chapter is concentrates on the design and performance measures of basic encoding and decoding techniques of convolutional codes.

3.2 Convolutional Coding Theory

The encoding process of convolutional codes is significantly different to that of block encoding. Block codes are developed through the use of algebraic techniques. Block encoders group information bits into length k blocks. These blocks are then mapped into codewords of length n. A convolutional encoder converts the entire input stream into length n codewords independent of the length k. The development of convolutional codes is based mostly on physical construction techniques. The evaluation and the nature

of the design of convolutional codes depends less on an algebraic manipulation and more on construction of the encoder.

Convolutional codes were first introduced by Elias [14] in 1955. He proved that redundancy could be added to an information stream through the use of linear shift registers. In 1961, Wozencraft and Reiffen described the first practical decoding algorithm for convolutional codes [15]. The algorithm was based on sequential decoding, however sub-optimal for decoding convolutional codes. Several other algorithms were developed off of Wozencraft and Reiffen initial work. In 1967, Viterbi proposed a maximum likelihood-decoding scheme for decoding convolutional codes [16]. The importance of the Viterbi algorithm is that it proved to be relatively easy to implement given the encoder has a small number of memory elements. It is the work by Viterbi that promotes the motivation here to apply his algorithm for the decoding of this error correction scheme.

Figure 3.1 is a binary rate 1/2 linear convolutional encoder. The rate of the encoder is determined by the fact that the encoder outputs two bits for every one bit at the input. In general, an encoder with k input bits and n output bits is said to have a rate k/n. The rate k/n is defined as the code rate (R_c) of the system.

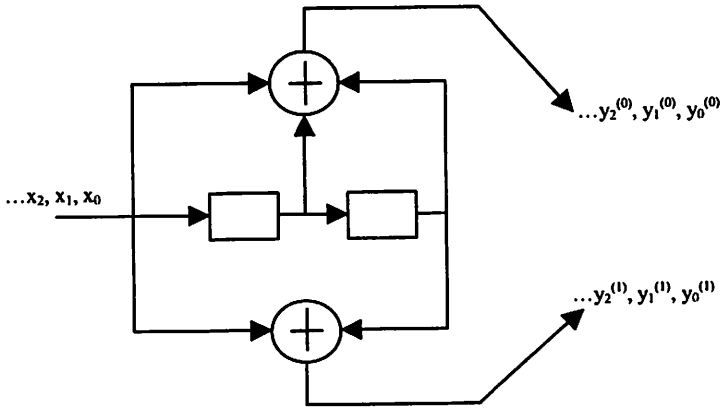

Figure 3.1 Convolutional Encoder

In Fig. 3.1 a binary stream of data $\mathbf{x}=(x_0,x_1,x_2,...)$ is fed into a series of memory elements. The bits travel through the shift register, the values of the individual memory elements are tapped off and added modulo-2 according to a fixed pattern. This creates a pair of output coded data streams $\mathbf{y}^{(0)}=(y_0{}^{(0)},y_1{}^{(0)},y_2{}^{(0)}...)$ and $\mathbf{y}^{(1)}=(y_0{}^{(1)},y_1{}^{(1)},y_2{}^{(1)}...)$. These output streams are multiplexed to create a single encoded data stream $\mathbf{y}=(y_0{}^{(0)}y_0{}^{(1)},y_1{}^{(0)}y_1{}^{(1)},y_2{}^{(0)}y_2{}^{(1)},...)$. The data stream $\mathbf{y}$ is the convolutional code word. Each element in the interleaved output stream y is a linear combination of the elements in the input stream $\mathbf{x}^{(0)},\mathbf{x}^{(1)},\mathbf{x}^{(2)},...\mathbf{x}^{(k-1)}$ assuming that the shift register contents are initialized to zero before the encoding process. The linearity of the codes words shows that if $\mathbf{y}_1$ and $\mathbf{y}_2$ are code words corresponding to inputs $\mathbf{x}_1$ and $\mathbf{x}_2$, then $(\mathbf{y}_1+\mathbf{y}_2)$ is the code word that corresponds to the input of $(\mathbf{x}_1+\mathbf{x}_2)$. The linear structure of these codes allows for use of powerful techniques from linear algebra theory.

There is a way to characterize the encoder structure of convolutional codes called generator sequences. Generator sequences are obtained by applying an impulse response $g_j{}^{(i)}$ where the i^{th} output of the encoder is obtained by applying a Dirac delta function $\delta=(1000...)$ data stream at the j^{th} input. The impulse responses for Fig. 3.3 are

$$\begin{aligned}\mathbf{g}^{(0)}&=(111)\\ \mathbf{g}^{(1)}&=(101)\end{aligned} \tag{3.1}$$

The generators have been terminated at a point where the following output values are all zeros. It should now be evident that the generators sequences can be determined by "counting" the number of taps off the shift register that connect to the j^{th} generator sequence. Since there are two memory elements each incoming bit can affect at most 3 bits, hence the length of the generator sequence. The constraint length K of a convolutional code in its simplest terms can be defined as the maximum number of taps off the shift registers in the encoder.

$$\mathbf{K}=l+1, \tag{3.2}$$

where l is the number of memory elements of the encoder structure. The memory of the encoder has a direct impact to the complexity of the decoder, specifically the Viterbi

algorithm that is used here. In practical implementations of the Viterbi algorithm the complexity is exponential in the constraint length and the number of input bits k.

There are two popular ways to describe a convolutional encoder. One way is graphically like Fig. 3.1; the other is by a generator matrix (3.3). A generator matrix is formed by interleaving the generator sequences $\mathbf{g}^{(0)}$ and $\mathbf{g}^{(1)}$,where m the number of generator sequences.

$$\mathbf{G}=\begin{bmatrix} g_o^{(0)}g_o^{(1)} & g_1^{(0)}g_1^{(1)} & g_2^{(0)}g_2^{(1)} & \cdots & g_m^{(0)}g_m^{(1)} & & \mathbf{0} \\ & g_o^{(0)}g_o^{(1)} & g_1^{(0)}g_1^{(1)} & g_2^{(0)}g_2^{(1)} & \cdots & g_m^{(0)}g_m^{(1)} & \mathbf{0} \\ & & g_o^{(0)}g_o^{(1)} & g_1^{(0)}g_1^{(1)} & g_2^{(0)}g_2^{(1)} & \cdots & g_m^{(0)}g_m^{(1)} \\ \mathbf{0} & & & \ddots & \ddots & \ddots & \cdots \end{bmatrix} \quad (3.3)$$

The action of the convolutional encoder can be described as a discrete convolutional operation, which leads for an appropriate transform that will provide a simpler multiplicative representation for encoding. The D-transform, called the delay transform can be interpreted as a delay operator, with the exponent denoting the number of time delay with respect to the $\mathbf{D}^{(0)}$ term.

$$\mathbf{x}^{(i)}=\left(x_0^{(i)},x_1^{(i)},x_2^{(i)}\ldots\right)\Leftrightarrow \mathbf{X}^{(i)}(D)=x_0^{(i)}+x_1^{(i)}D+x_2^{(i)}D^2+\ldots \quad (3.4)$$

$$\mathbf{y}^{(i)}=\left(y_0^{(i)},y_1^{(i)},y_2^{(i)}\ldots\right)\Leftrightarrow \mathbf{Y}^{(i)}(D)=y_0^{(i)}+y_1^{(i)}D+y_2^{(i)}D^2+\ldots \quad (3.5)$$

$$\mathbf{g}_j^{(0)}=\left(g_0^{(i)},g_1^{(i)},g_2^{(i)}\ldots\right)\Leftrightarrow \mathbf{G}_j^{(i)}(D)=g_{j0}^{(i)}+g_{j1}^{(i)}D+g_{j2}^{(i)}D^2+\ldots \quad (3.6)$$

The encoding operation of a single input encoder can be represented as follows.

$$\mathbf{Y}^{(i)}(D)=\mathbf{X}(D)\mathbf{G}_j^{(i)}(D) \quad (3.7)$$

$$\mathbf{Y}^{(i)}(D)=[\mathbf{X}(D)]\begin{bmatrix} \mathbf{G}_o^{(0)}(D) & \mathbf{G}_o^{(1)}(D) & \cdots & \mathbf{G}_o^{(n-1)}(D) \\ \mathbf{G}_1^{(0)}(D) & \mathbf{G}_o^{(1)}(D) & \cdots & \mathbf{G}_1^{(n-1)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{G}_{k-1}^{(0)}(D) & \mathbf{G}_{k-1}^{(1)}(D) & & \mathbf{G}_{k-1}^{(n-1)}(D) \end{bmatrix} \quad (3.8)$$

The matrix G(D) is called the transfer-function matrix. The number of rows represents the k input streams and the number of columns represents the n output streams. If the input stream to Fig. 3.1 is

$$\mathbf{x} = (101) \tag{3.9}$$

the corresponding D-transform is

$$\mathbf{X} = 1 + D^2 \tag{3.10}$$

The transfer-function matrix for Fig. 3.1 is

$$\mathbf{G}(D) = \left[G^{(0)} \quad G^{(1)}\right] = \left[1 + D + D^2 \quad 1 + D^2\right] \tag{3.11}$$

The D-transform of the output coded bits are

$$\mathbf{Y}(D) = \mathbf{X}(D)\mathbf{G}(D) = \left[1 + D^2\right] \bullet \left[1 + D + D^2 \quad 1 + D^2\right] \tag{3.12}$$

$$\mathbf{Y}(D) = \left[1 + D + D^2 + D^2 + D^3 + D^4 \quad 1 + D^2 + D^2 + D^4\right] \tag{3.13}$$

$$\mathbf{Y}(D) = \left[1 + D + D^3 + D^4 \quad 1 + D^4\right] \tag{3.14}$$

since the arithmetic here is based on Galois fields [17] of size p, the addition and multiplication are modulo p, in this case p is two. Given

$$\mathbf{Y}(D) = \left(\mathbf{Y}^{(0)}(D), \mathbf{Y}^{(1)}(D), \cdots, \mathbf{Y}^{(n-1)}(D)\right) \tag{3.15}$$

Inverting the transform yields

$$\begin{aligned} \mathbf{y}^{(0)} &= (11010) \\ \mathbf{y}^{(1)} &= (10001) \end{aligned} \tag{3.16}$$

Thus, the output code word for $\mathbf{y} = (11, 10, 00, 10, 01)$

3.2.1 Structural Properties of Convolutional Codes

The techniques used to analyze and compare block codes does not work so well when it comes to convolutional codes. A considerable amount of the analysis of block codes resides in obtaining a fixed length codeword to determine the minimum distance. Convolutional codes are somewhat different in that the encoder can generate code words of arbitrary length. There are three popular methods to describing the performance of convolutional codes: the tree diagram, the trellis diagram, and the state diagram. All three methods show the possible evolution of the states of the encoder thru time, for more

information on the other methods see [8]. The state diagram has been chosen here because it shows a more compact structure in describing the possible states and outputs of the encoder. It also leads nicely into the discussion of decoding which is described in Section 3.3.

The convolutional encoder is a state machine. It contains memory elements whose contents determine the mapping between the next set of input and output bits. Fig. 3.2 below is the state diagram for the encoder in Fig. 3.1.

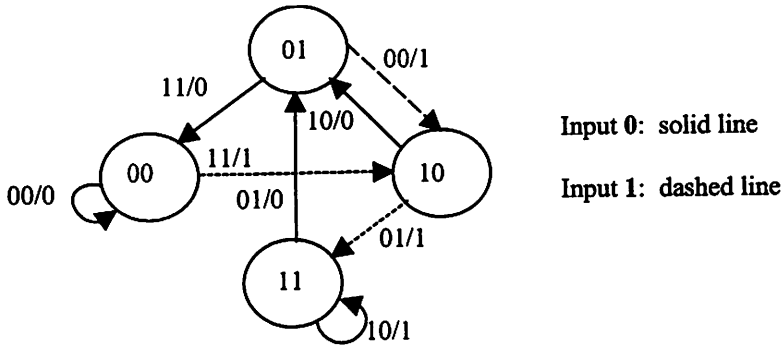

Figure 3.2 State Diagram

As with most finite-state machines, the encoder only can move between states in a limited manner. Each branch in the state diagram has a label of the form XX/Y, where XX is the output pair corresponding to the input bit Y. The distance properties and the error rate performance of a convolutional code can be obtained from its state diagram. Another benefit of the state diagram is that it gives performance measures that are considered key in comparing convolutional codes: the minimum free distance, denoted by d_{free}. Convolutional code words are linear, therefore, there exists a subspace $\mathbf{C} = \{C_i, C_j, \cdots, C_m\}$ in which any two code words C_i and C_j added together produce another code word that exists in the subspace C. The number of places in which two code words differ is referred to as the Hamming distance between the two code words. The minimum free distance, denoted d_{free}, is the minimum Hamming distance between all pairs of code words. In relation to the state diagram, d_{free} is the minimum Hamming distance between any two different paths of any length L, where the paths begin in the same state$^{(i)}$ and end in the same state$^{(j)}$ where i need not equal j. The minimum distance

is an important metric because it inherently gives the error correcting power of the codeword. As stated previously, in order to correct errors within a codeword the received code word must not land on another code word. The value of d_{free} gives a measure of how many bits can be "flipped" in order for the received code word not to be a given code word in the subspace C. It is important to note that the value of d_{free} increases as the constraint length increases. Daut [18] derived a simple upper bound the minimum free distance of a rate 1/n convolutional code. It is given by

$$d_{free} \leq \min_{r \geq 1} \left\lfloor \frac{2^{r-1}}{2^r - 1}(K + r - 1)n \right\rfloor \tag{3.17}$$

where $\lfloor x \rfloor$ denotes the largest integer contained in r, K is the constraint length, r is the number of input bits, and n is the number of encoded output bits. It should be evident by (3.17) that d_{free} increases if either the constraint length increases or the code rate R_c decreases. These factors must be carefully examined on a system level because they affect the overall system performance and the implementation complexity of the Viterbi decoder.

3.3 Viterbi Algorithm

In 1967 Andrew Viterbi proposed an algorithm as an approach to the decoding of convolutional codes [16]. Shortly after, Forney showed that the Viterbi algorithm is a maximum-likelihood (ML) [19] decoding algorithm for convolutional codes. In 1979, Cain, Clark, and Geist showed that the complexity of the Viterbi algorithm could be greatly simplified through puncturing [20], which will be discussed later in this chapter. The Viterbi algorithm makes for an efficient implementation of the maximum likelihood sequence detection algorithm. Fundamentally the algorithm determines the most likely path taken given a received sequence.

A brief description of how the Viterbi algorithm works is needed. For a more detailed description of the algorithm refer to [17], [8]. This example used here is for hard decision decoding because it simplifies the decoding to minimum Hamming distance

decoding, thus simplifying the explanation. Let the length of a given path be $B = \frac{L}{k}$ branches, where L is the length of the information sequence and k the number of input bits into the decoder. Also, let n be the number of coded bits per branch. Define the Hamming distance

$$d_j^{(i)} = \sum_{m=1}^{n} y_{jm} \oplus c_{jm}^{(i)},\ j = 1, 2, \ldots, B,\ m = 1, 2, \ldots, n \tag{3.18}$$

as the metric between the received sequence $\bar{y}$ and a candidate sequence $\bar{C}^{(i)}$, $i = 1, 2, \ldots, 2^L$ on the jth branch. The Hamming path metric between the received sequence $\bar{y}$ and a candidate sequence $\bar{C}^{(i)}$ is

$$d^{(i)} = \sum_{j=1}^{B} d_j^{(i)} = \sum_{j=1}^{M} \sum_{m=1}^{n} y_{jm} \oplus c^{(i)}{}_{jm} \tag{3.19}$$

The value $d^{(i)}$ can be considered as the Hamming distance between the received vector and the candidate sequence.

Consider computing the path metric on a branch-by-branch basis for a candidate path $\bar{C}^{(i)}$:

$$d^{(i)} = \sum_{j=1}^{B} d_j^{(i)} = \sum_{j=1}^{M} \sum_{m=1}^{n} y_{jm} \oplus c^{(i)}{}_{jm} \tag{3.20}$$

The value $d^{(i)}$ can be considered as the Hamming distance between the received vector and the candidate sequence.

Consider computing the path metric on a branch-by-branch basis for a candidate path $\bar{C}^{(i)}$:

$$d^{(i)} = \underbrace{\sum_{j=1}^{B} d_j^{(i)}}_{\text{Total Path Metric}} = \underbrace{\sum_{j=1}^{J} d_j^{(i)}}_{\text{Left of J}} + \underbrace{\sum_{j=J+1}^{B} d_j^{(i)}}_{\text{Right of J}} \tag{3.21}$$

At any arbitrary time J, the path metric can be broken into two equations, the partial path metric to the left of time J and the partial path metric to the right of J. Suppose two

candidate paths $\bar{C}^{(i_1)}$ and $\bar{C}^{(i_2)}$ merge at time J and share a common path for $j \geq J+1$. If $\bar{C}^{(i_2)}$ has a smaller path metric than $\bar{C}^{(i_1)}$ for $j \leq J$ then $\bar{C}^{(i_2)}$ will always have a smaller path metric than $\bar{C}^{(i_1)}$ Thus, $\bar{C}^{(i_2)}$ is called the survivor path and $\bar{C}^{(i_1)}$ is excluded as a candidate path. For each of the $2^{k(K-1)}$ states at time J, store the list of transistions in survivor path and the partial path metric of each survivor path. After a predetermined amount of time, go back through the trellis along the survivor path until the all-zero state is reached. This is the optimal path and the input bit sequence corresponding is the maximum likelihood decoded sequence.

3.3.1 Algorithmic Complexity

The complexity of the decoder greatly depends on two major factors, the number of input bits *k*, and the constraint length K. There are $2^{k(K-1)}$ states at each time instance and each one must store the survivor paths and the partial path metrics. There are $2^{k(K-1)}$ surviving paths at each stage and $2^{k(K-1)}$ computations for each surviving path. The decoding of a code requires the algorithm to keep track of $2^{k(K-1)}$ surviving paths and $2^{k(K-1)}$ computations. At each stage of the trellis, there are 2^k paths that merge at each node. Since each path converges at a common node, there are 2^k computations at each node. The minimum distance path at a given time instance is the surviving path of the 2^k paths that merge at each node. The decoding computations at each stage increases exponentially with *k* and K. This exponential increase in computation makes it prohibitive for large values of K and *k*.

The optimal performance of the Viterbi algorithm requires that for long information sequences the decoding delay should be infinite. For practical applications an infinite delay is not possible. A solution to modify this problem is referred as path memory truncation. The goal is to determine a fixed decoding delay that does not significantly degrade from the optimal performance of the algorithm. The modification is to retain at any time *t* the most recent δ decoded information symbols in each surviving sequence. At each new information bit received force a decision on the bit received δ branches back in the trellis. Equivalently, for each new information bit received a decision is made on

the bit δ time instances later; this decoding delay is called the traceback length. The traceback length δ should be chosen large enough so that with a high probability all surviving sequences at time t stem from the same node $t-\delta$. It is known that a delay $\delta \geq 5K$ results in negligible performance degradation.

3.3.2 Error Bounds

The error rate performance of convolutional codes with Viterbi decoding is based on two metrics. One is hard decision decoding, where the input samples from the matched filter or cross correlator is a 2-level quantized value $\in [0,1]$. For an AWGN channel the minimum SNR per bit for an arbitrary small probability of bit error p *is*

$$\frac{E_b}{N_o} = \frac{1}{2}\pi \ln 2 \quad (0.37\ dB) \tag{3.22}$$

where E_b is the energy required for a bit [8]. The second is soft-decision decoding where the input samples from the matched filter are unquantized or quantized to M levels. The minimum SNR per bit for soft-decisions is

$$\frac{E_b}{N_o} = \ln 2 \quad (-1.6\ dB) \tag{3.23}$$

Thus, there is a 2 dB improvement when soft-decision decoding is implemented. The difference between hard-decision decoding and soft-decision decoding is approximately 2 dB in the range from $10^{-2} > P_M > 10^{-6}$, where P_M is the probability of a codeword error. It is for this reason that soft-decision decoding is implemented for this work. The probability of error for soft-decision decoding via the Viterbi algorithm gives an upper bound metric on how the algorithm will perform so as to benchmark with the simulated values. The computation of the probability of error for soft-decision decoding requires the knowledge of the weight distribution of the code [8]. Weight distributions of the codes used here a given in texts, e.g., [21], [8], and [17]. From [8] the upper bound of the bit error probability is

$$P_b < \frac{1}{k} \sum_{d=d_{free}}^{\infty} \beta_d Q\left(\sqrt{\frac{2E_b}{N_o} R_c d}\right) \tag{3.24}$$

where β_d is the total number of bit errors of all paths of weight d. It is this upper bound that sets the benchmark of performance for this system. Another important metric of error control coding is the coding gain.

$$coding\ gain\ (dB) = 10\log_{10}\left(\frac{\left(\frac{E_b}{N_o}\right)_{uncoded}}{\left(\frac{E_b}{N_o}\right)_{coded}}\right) \tag{3.25}$$

Coding gain is the reduction in $\frac{E_b}{N_o}$ required for a given bit error probability. In simpler terms, it is the improvement in dB of a coded system vs. an uncoded system for a given bit error rate. Since we are using QPSK as the modulation scheme, the equation above reduces to [8].

$$coding\ gain\ (dB) \leq 10\log_{10}\left(R_c d_{free}\right) \tag{3.26}$$

3.4 Punctured Convolutional Codes

Since the complexity of Viterbi decoding is exponential in the number of input symbols as *k* gets large implementation complexity becomes difficult. Classes of codes called punctured convolutional codes were introduced by Cain and Clark [20] in 1979. By periodically deleting bits via a puncturing matrix these codes allow for higher rate codes, which give a higher coding gain while not suffering the implementation penalty from a large value of *k*. If the encoder structure is a lower rate code 1/n, then there are only 2^k computations for each node at the decoding trellis, which is suitable for practical implementations.

The punctured coded approach can be best understood by illustration, for a detailed example see [20]. Assume the mother code, or standard encoder structure, is rate 1/2. For k input bits the encoder output n bits.

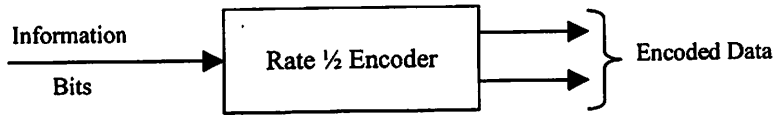

Figure 3.3 Rate ½ Encoder

Assume the information sequence is length L, thus the block represents L transmitted output bit pairs. A rate P/Q punctured convolutional code can be obtained from a rate $1/n$ convolutional code by deleting $n \cdot P - Q$ code symbols from every $n \cdot P$ encoded bit corresponding to the encoding of P information symbols by the rate 1/n code. The rate of the convolutional code is $R_c' = \frac{P}{(n \cdot P - \sigma)}$, where $\sigma = n \cdot P - Q$. The deletion of the coded bit is represented by an n-by-P puncturing matrix $\mathbf{P}_\sigma$.

The elements of the puncturing array are zeros and ones, corresponding to keeping or deleting the encoded bits respectively. The puncturing device deletes symbols from the code sequence according to the puncturing matrix. Figure 3.4 is an illustrative example of how puncturing works:

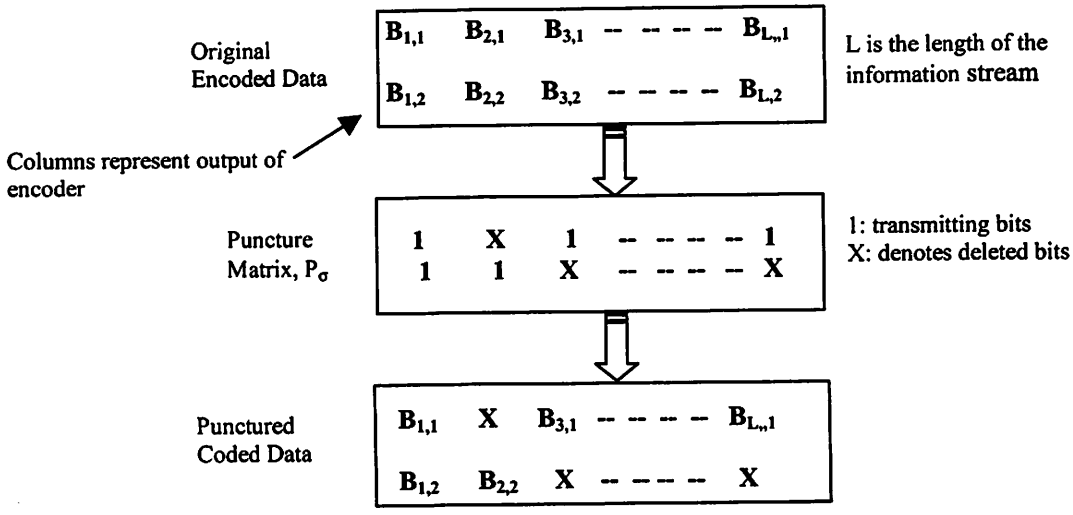

Figure 3.4 Puncture Example

The output sequence of from Fig. 3.4 is

$$\mathbf{y} = (B_{1,1}B_{1,2}, XB_{2,2}, B_{3,1}X, \ldots, B_{L,1}X) \tag{3.27}$$

Remember that the X's denote data that is deleted. Look at the first two output sequences $\mathbf{y}' = (B_{1,1}B_{1,2}, XB_{2,2})$ for every two input sequences (input value for output $B_{1,1}B_{1,2}$ and the input value of the output $B_{1,1}X$) there are three output sequences(the transmitted values of y') Choosing an arbitary generator matrix for a rate 2/3 code, the trellis structure for the rate 2/3 code and y' would be equal.

All punctured convolutional codes can be decoded using the Viterbi decoder. In order to take advantage of the simple 1/n mother code the punctured coded data must be transformed back into a rate 1/n structure for input to the decoder. Erasure bits are added back to the punctured received data to allow the decoder to work on the 1/n mother code and not the P/Q punctured code. From prior knowledge of the puncturing matrix $\mathbf{P}_\sigma$ erasure bits are inserted in the previously deleted positions corresponding to the positions of the punctured encoded bits at the transmitter. Figure. 3.5 shows how erasure bits are added, it is assumed that the erasure matrix is based on Fig.3.4.

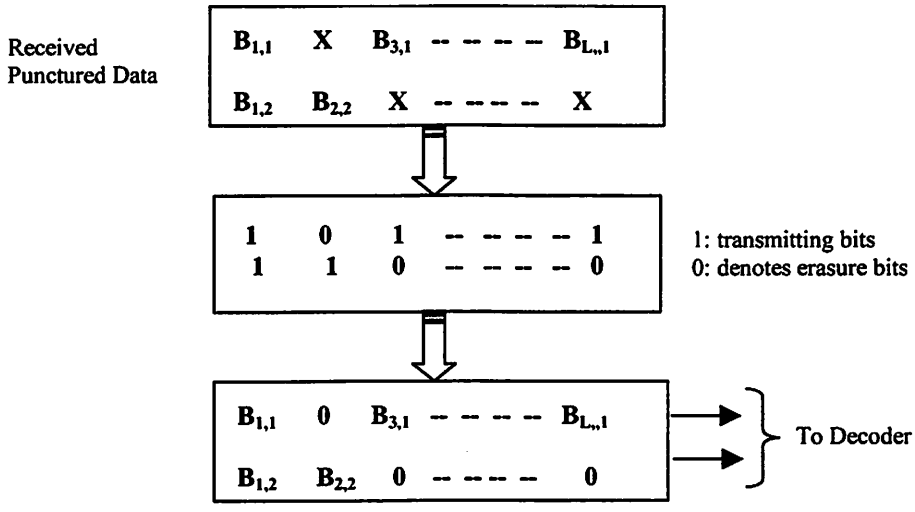

Figure 3.5 Insert Erasure Example

The input sequence to the decoder is $\mathbf{r} = (B_{1,1}B_{1,2}, 0B_{2,2}, B_{3,1}0, \ldots, B_{L,1}0)$, where the zeros represent some constant value denoted for erasure bits. The decoder operates on the 1/n mother code so there is no added complexity to the decoder except at increase in the

traceback length due to puncturing. The performance bounds for punctured convolutional codes are equal to those for convolutional codes except there is a slight performance loss of (0.1 to 0.2 dB) for punctured codes. The performance loss is due in part to the smallere values of d_{free} for some punctured codes. The bit error probability for punctured codes from [22] is

$$P_b \leq \frac{1}{k}\sum_{d_{free}}^{\infty} c_d Q\left(\sqrt{2d\left(\frac{k}{n}\right)\left(\frac{E_b}{N_o}\right)}\right) \tag{3.28}$$

where c_d is the total number of bit errors on all paths of weight $d \geq d_{free}$ that diverge from the correct path and remerge at a some later time. To choose good codes maximize d_{free} and minimize c_d. The search for optimum punctured codes has been done by [20], [23], [24]. The research done here uses the codes generated by these papers.

4

Error Control System Design

4.1 Introduction

This chapter describes the SIMULINK model of the error correction scheme designed for use with the MCMA framework. The focus is to build a modular and flexible simulation model that can be used for a variety of advanced communication systems. A discussion of the floating-point model will be detailed on a block-by-block basis. This is followed by a migration to the fixed-point block equivalent model. The Viterbi decoder is not described as a fixed-point block in SIMULINK, it is described in MODULE COMPILER. MODULE COMPILER (MC) allows a high-level description of a design to be written and generated into a gate-level description (VHDL).

SIMULINK, a software package from Mathworks®, is a graphical user interface (GUI) software package for modeling and analyzing dynamic systems. Models are hierarchical, thus can be built from a top-down or bottom-up approach. The top-down approach is taken here where the viewed level is the top-level. A more detailed view of a block can be contained underneath these top-level views.

4.2 Floating Point

A top-level view is shown is Fig. 4.1. The major components are the convolutional encoder, puncture, insert erasure, quantizer, and Viterbi decoder blocks.

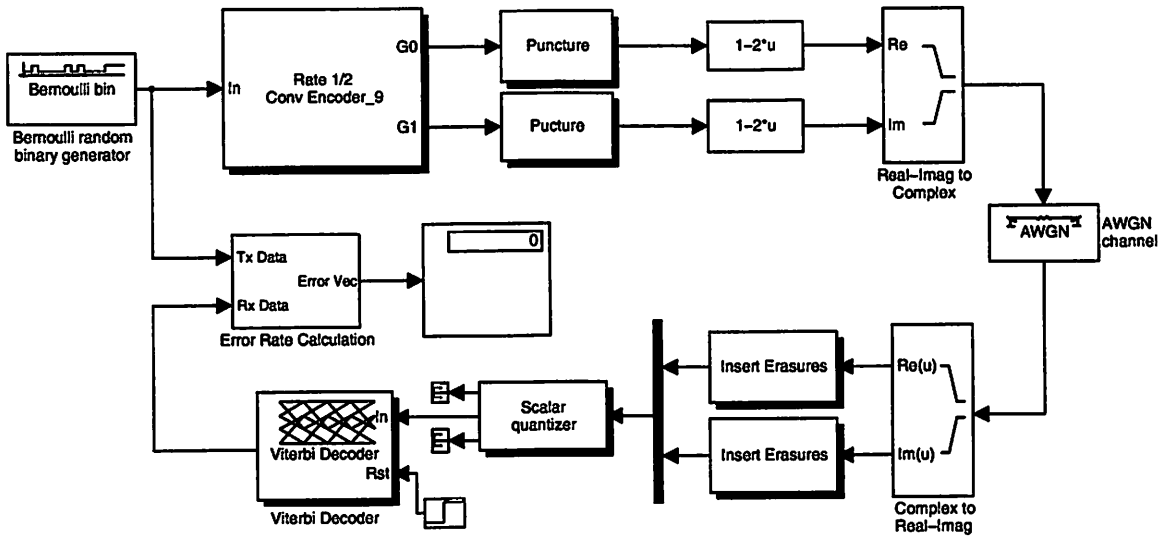

Figure 4.1 Floating Point System Level Model

The minor components of the top-level include the remaining blocks shown. The Bernoulli random generator generates a random binary sequence with a probability of zero equal to 0.5. Gray-coded QPSK is used as the modulation scheme and is created using the f(u)=1-2u function block and the `Real-Imag to Complex` blocks. The mapping of a binary '0' to 1 and binary '1' to –1, is the same as the constellation shown in Fig. 2.2.

4.2.1 Convolutional Encoder

As described in Section 3.2, a rate 1/2 convolutional encoder is used. A rate 1/2 encoder was chosen due to the maximum free distance metric and the simplicity in the decoder complexity. The decoder complexity is proportional to the number of input bits, k, and the constraint length, K of the encoder. The encoder configuration is a series of K-1 single delay registers being added modulo-2 to form the encoded symbols, see Fig. 4.2.

Information bits are shifted in at the left, and for each *k* information bit the output of the modulo-2 adder provides *n=2* encoded symbols, thus a rate 1/2 encoder.

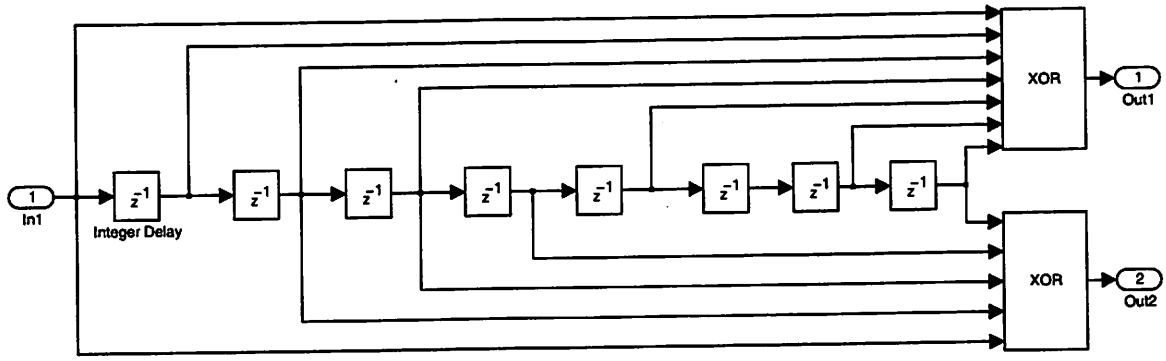

Figure 4.2 Floating Convolutional Encoder, K=9

The maximum d_{free} is increased with an increase in the constraint length or decrease in the code rate. In order to achieve a BER of 10^{-5} for the punctured code rates that are used here an optimal constraint length has to be determined. In practice constraint lengths of 7-9 are used for reasonable decoding performance. Table 4.1 shows the maximum d_{free} and upper bound on the coding gain for a rate 1/2 with various constraint lengths.

	Maximum d_{free}	Coding Gain (dB)
K=5	7	5.44
K=6	8	6.02
K=7	10	6.99
K=8	10	6.99
K=9	12	7.78
K=10	12	7.78

Table 4.1 Maximum free distance and Coding Gain

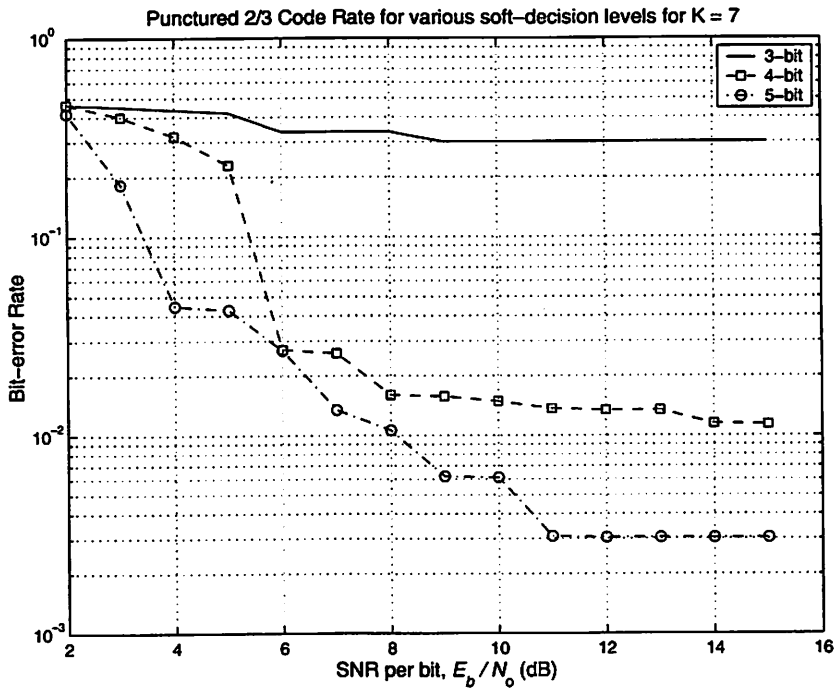

Figure 4.3 BER curve for rate 2/3, K=7

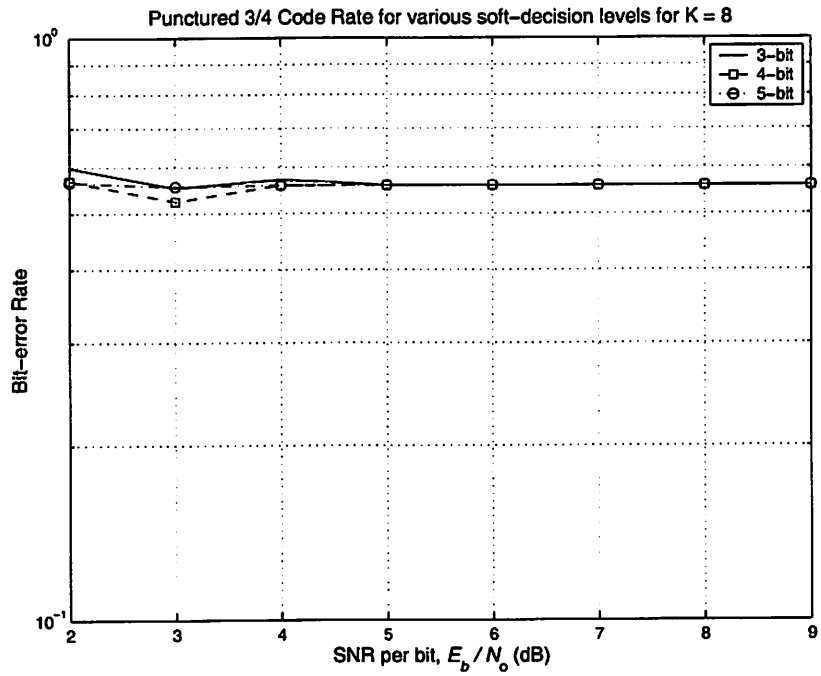

Figure 4.4 BER curve for rate 3/4, K=8

The maximum d_{free} is obtained from [21] and the coding gain comes from (3.26). Table 4.1 shows that there is no performance increase from a K=7 to K=8. The decoder complexity is exponential in the constraint length so it is important to determine the smallest constraint length that provides acceptable performance. Simulations were run for K=7 to K=9 to determine the constraint length that met the BER performance metric of 10^{-5}.

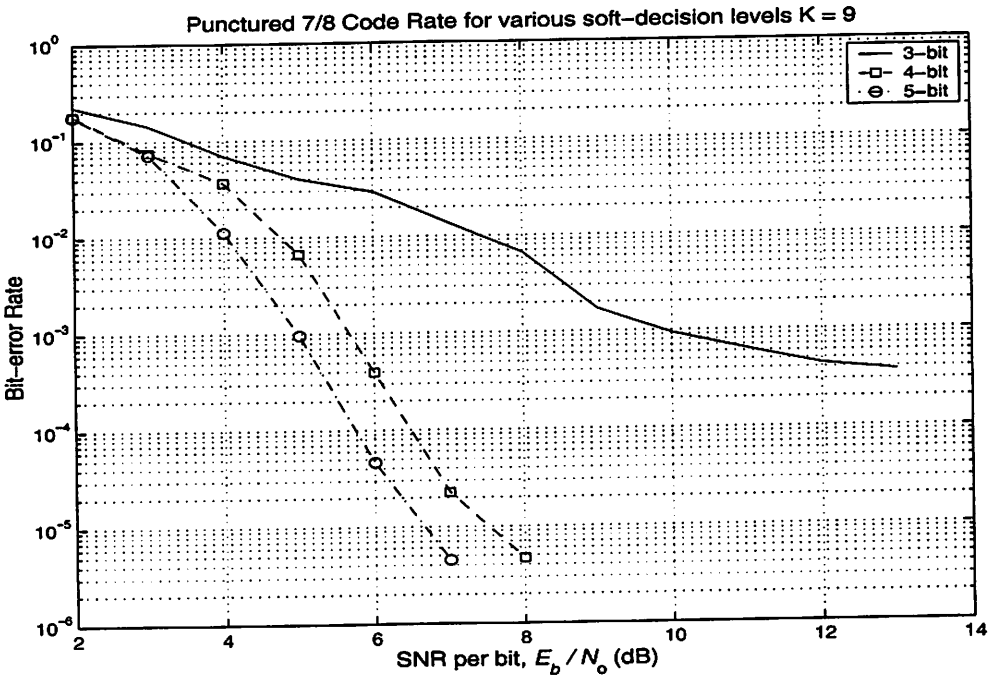

Figure 4.5 BER curve for rate 7/8, K=9

Figures 4.3, 4.4, 4.5 show the BER performance for the three constraints and how they perform for different code rates. It is clear from the figures that an encoder with constraint length of 9 is the only one that contains enough memory to meet the BER requirement for all code rates. The generator polynomial [753 561], see [22], provides the maximum d_{free} for $K=9$.

4.2.2 Puncturing

The theory behind bit puncturing was discussed in Section 3.4. The code rates used are $\frac{P}{Q}, P < Q$, punctured rates. The method of deleting bits requires the expansion of the puncturing matrix to a vector of length 2*Q. Input bits are sent through a selector block that emulates only selecting data values that correspond to a bit position carrying a '1' from the puncturing matrix, see Fig. 4.6.

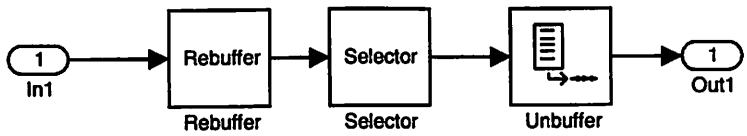

Figure 4.6 Floating Point Puncture Block

For example, if the puncturing matrix is *j=[1 0 1;1 1 0]* it gets expanded to a vector *v=[1 1 0 1 1 0]* ,where the maximum length of *v* is 2*Q. There are 1's in the 1st, 2nd, 4th, and 5th bit positions of *v*. Thus, for any input vector *x*, only the bits in the 1st, 2nd, 4th, and 5th bit positions will be passed, creating an output vector of length *P*.

Four different code rates, 1/2, 2/3, 3/4, and 7/8 where selected based on the increase in both data rate and coding gain. The puncturing matrices used are:

- Rate 2/3: $P_\sigma = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$
- Rate 3/4: $P_\sigma = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$
- Rate 7/8: $P_\sigma = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$

The upper bound on the BER performance is shown in Fig. (4.7). Figure (4.7) also shows that ideally only a dB of signal power is needed to increase from one of the given code rates to the next.

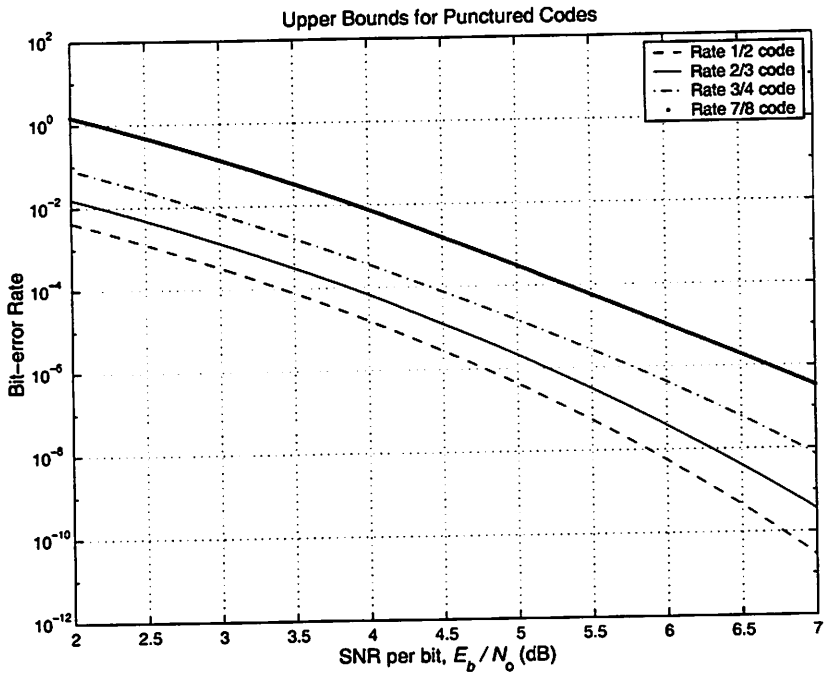

Figure 4.7 BER Upper Bound on Punctured Codes

There is a maximum increase of throughput of 75%, given an increase in code rate from rate 1/2 to rate 7/8. This 1 dB of SNR is ideal and doesn't take into account quantization error. Section 4.2.4 will show simulation results for different quantization levels.

4.2.3 Insert Erasure

The `'Insert Erasures'` block essentially does the opposite of the puncturing block, see Fig. (4.8).

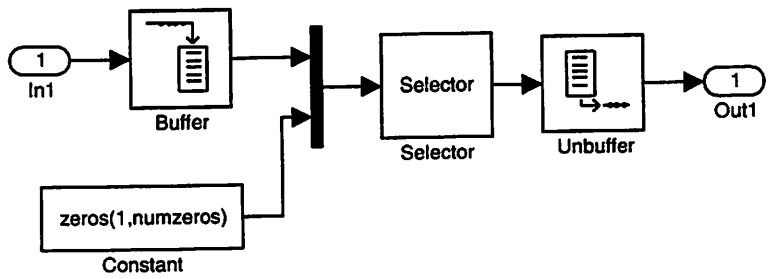

Figure 4.8 Floating Point Insert Erasure Block

The puncture block takes in at a length *2*Q* and outputs symbols at a length 2**P*, creating the punctured rate $\frac{P}{Q}$. It is this length change that creates the variable coding rates. The `Insert Erasure` block appends *z*, where *z* is the number of zeros in the puncturing matrix. The `Insert Erasures` block converts the variable lengths and code rates generated by the puncturing block back to the original rate 1/2 code rate for Viterbi decoding. Using the puncturing matrix, *p=[1 1 0; 1 0 1]*, *z=2*, thus 2 dummy values of zeros are added to the input creating the length *2*Q* at the input to the puncturing block. The '`Selector` block rearranges the new length input vector, *2*Q* so that the zeros are placed in the element positions that were originally deleted by the puncturing block. Puncturing is essentially the deletion of data bits making it more difficult for the decoder to correct errors. To create the original rate 1/2 needed at the input to the Viterbi decoder a buffer is added between the `Insert Erasure` block and the `Quantizer` block.

4.2.4 Quantizer

The Viterbi decoder is designed to take soft decision input values to maximize performance. The quantization levels explored here are 3, 4, and 5-bitsl. Proakis [8] showed that there is no huge gain in improvement for the added complexity for soft-decision level greater that 5 for soft-decision Viterbi decoding. There is no correlation done on the signal after the AWGN channel block. The '`Quantizer`' allows a range of the inputs signals to be specified. Quantization ranges are next specified given the range of the input signal and the quantization levels used. The integer range for a given soft decision level is $[0,2^n-1]$, where *n* is the soft decision level. For example, if we assume from QPSK modulation that the signal range is (-1,1), and using a 3-bit quantization level the integer range is [0,7]. The quantization partitions are:

- 3-bit quantization: [-.75 -.5 -.25 0 .25 .5 .75]
- 4-bit quantization: [-.875 -.75 -.625 -.5 -.375 -.25 -.125 0 .125 .25 .375 .5 .625 .75 .875]
- 5-bit quantization: [-0.9375 -0.8750 -0.8125 -0.7500 -0.6875 -0.6250 -0.5625 -0.5000 -0.4375 -0.3750 -0.3125 -0.2500 -0.1875 -0.1250 -0.0625 0 0.0625 0.1250

0.1875 0.2500 0.3125 0.3750 0.4375 0.5000 0.5625 0.6250 0.6875 0.7500 0.8125 0.8750 0.9375]

Given the constraint length is 9 was found to be optimal for this system Figures 4.10, 4.11, and, 4.12 show the performance of the various soft-decision levels explored.

It is clear that 4-bit quantization is the best tradeoff of complexity vs. performance since it consistently mirrors the performance of the more precise 5-bit level curve, as the code rate increases from rate 1/2 to rate7/8. Given 4-bit quantization seems to be the best tradeoff, Fig. 4.13 shows the simulated performance of all code rates where K=9. The simulated output doesn't mirror the idea BER performance of Fig. 4.7. The code rates are separated by about a dB except from rate ¾ to rate 7/8.

4.3 Fixed-Point

Logic circuits of fixed-point hardware consume less power, are smaller in size, and less complicated than those of floating-point hardware. The fixed-point representation is to provide an equivalent algorithmic system that models fixed-point constraints. It bridges the gap between dynamic system design and hardware implementation. In terms of the research here designing a fixed-point equivalent model provides verification of the high-level design. As described in Section (4.4), MODULE COMPILER is used as the method to determine power, area and, delay estimates of an algorithmic block and generate behavioral VHDL. The behavioral VHDL is verified against the fixed-point model output to ensure to bit level accuracy. The fixed-point blocks developed use word lengths and data types that are precise enough so not to incur a huge degradation in performance. The purpose here is to create a functional fixed-point model not a system to model degradation to meet a size, complexity, or power consumption requirement. Once a functional model is complete further work can be down to manipulate the data types and word lengths to tradeoff power and accuracy.

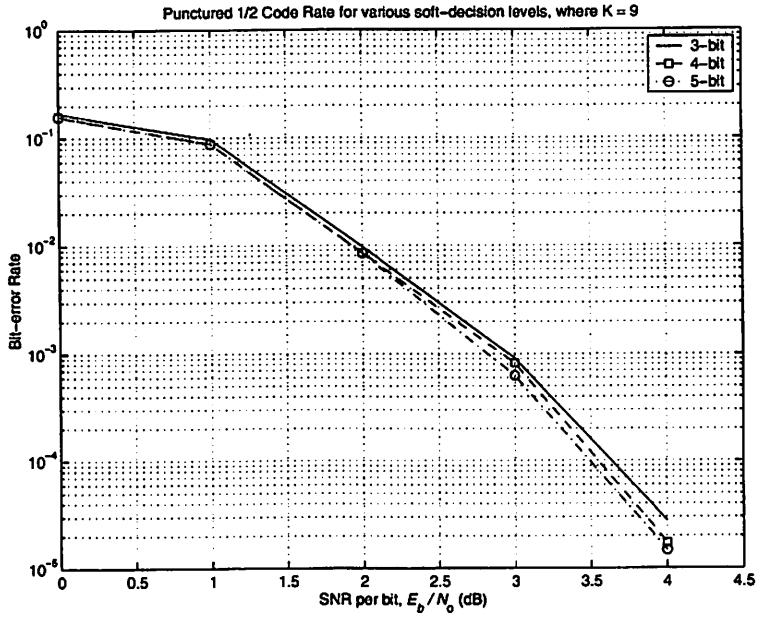

Figure 4.9 Soft Decision Performance for rate ½

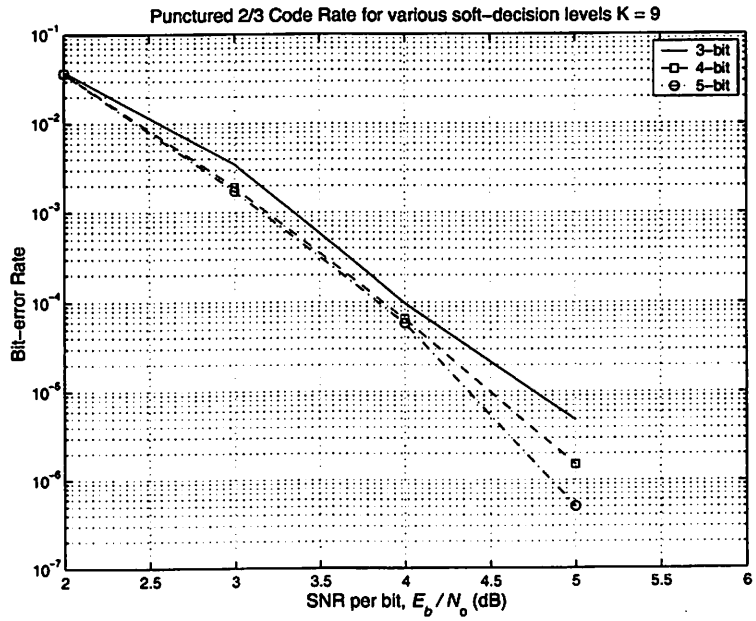

Figure 4.10 Soft Decision Performance for rate 2/3

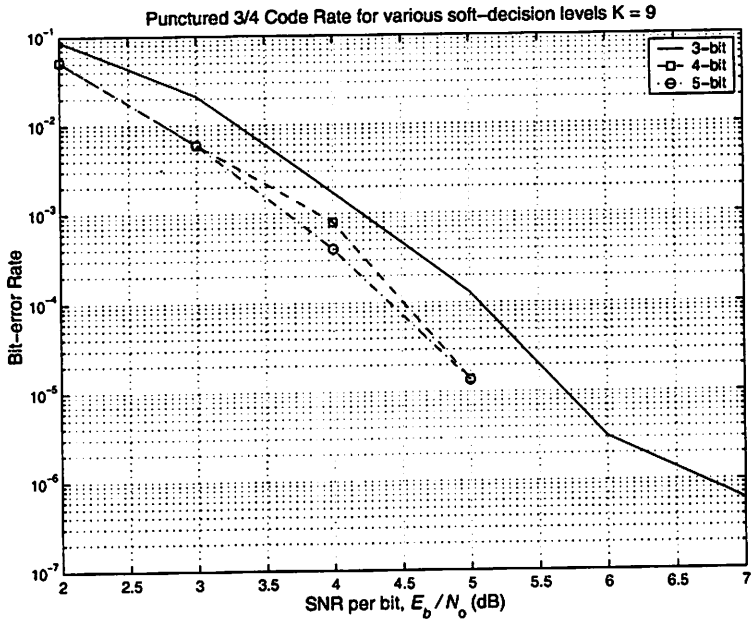

Figure 4.11 Soft Decision Performance for rate ¾

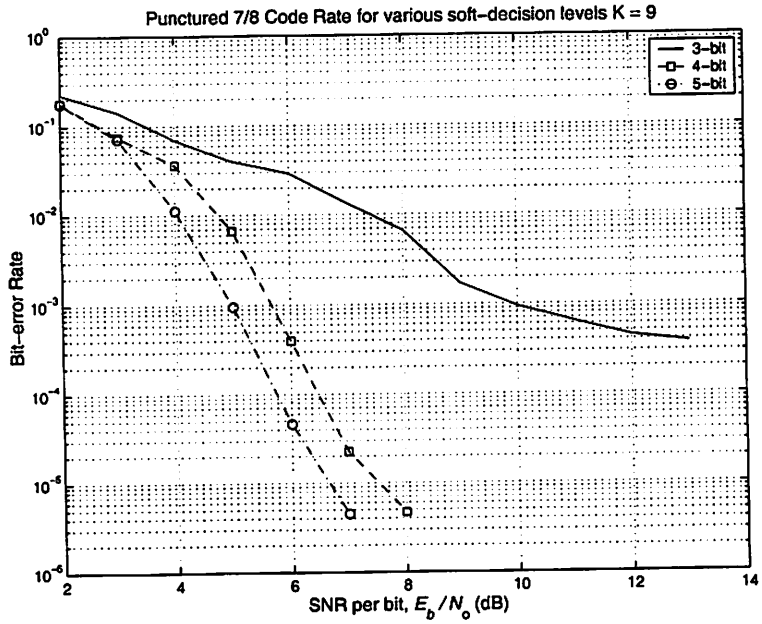

Figure 4.12 Soft Decision Performance for rate 7/8

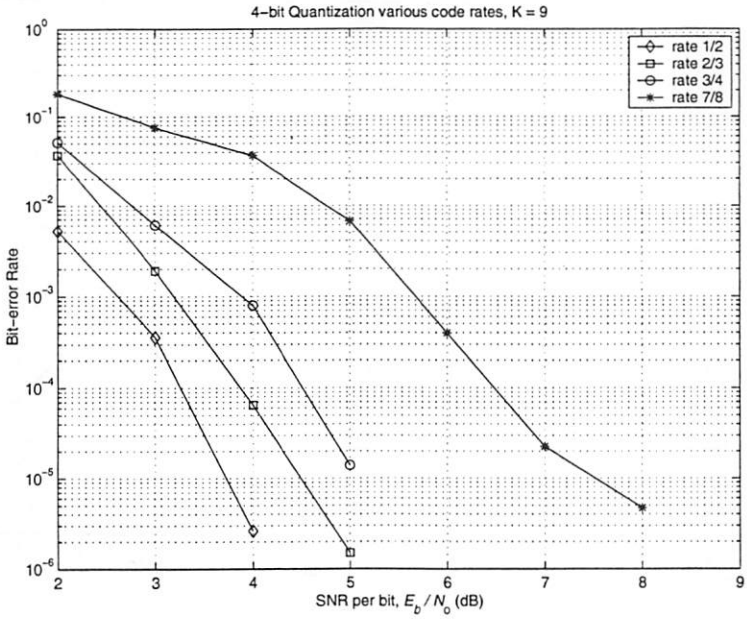

Figure 4.13 4-bit Quantization System Performance

Figure 4.14 is the top-level view of the fixed-point equivalent to the punctured convolutional coding system.

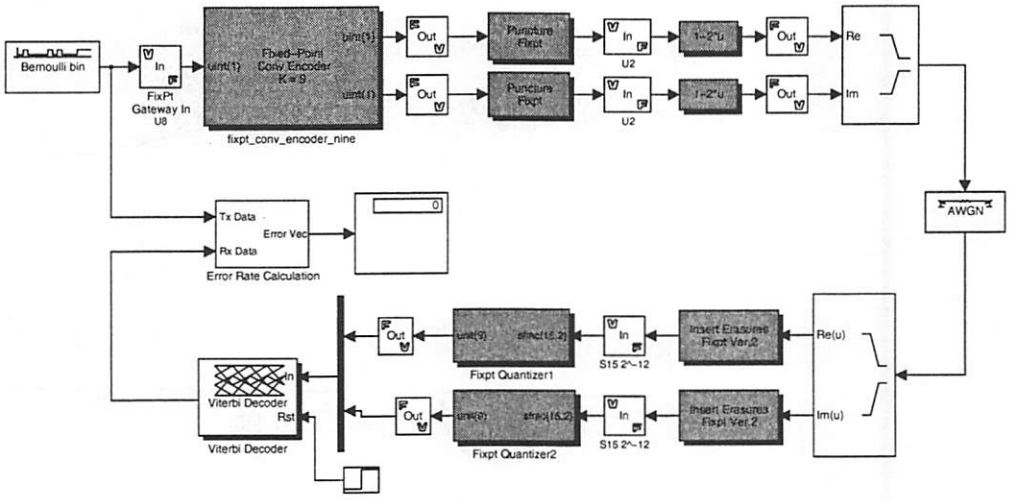

Figure 4.14 Fixed-Point System Model

4.3.1 Fixed-Point Convolutional Encoder

Some blocks exist both in SIMULINK's floating-point blockset as well as it's fixed-point blockset. The blocks that make up the encoder exist both in the floating and fixed-point blockset, so the design is straightforward. From Fig. 4.15 there is a one-to-one architecture mapping between the two systems.

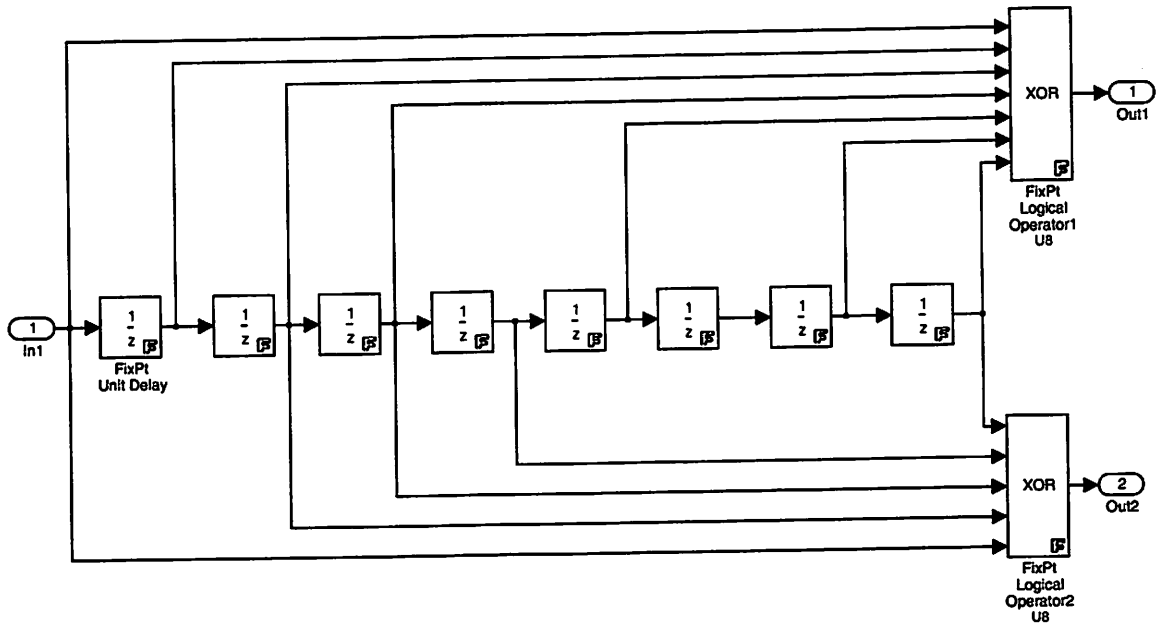

Figure 4.15 Fixed-Point Convolutional Encoder

4.3.2 Fixed-Point Puncturing

The fixed-point model cannot handle the potentially infinite long number of $\frac{P}{Q}$ puncture rates. In order to build implementable hardware the number of various code rates must be fixed. The `Multiport Switch` serves as the `Selector` block in floating point model. The `Multiport Switch`, shown in Fig. 4.16, must be greater than or equal to *2*Q-1* of the highest $\frac{P}{Q}$ punctured code rate used. The method used to periodically delete bits is as follows:

- Given a puncture matrix, the element positions that have a '1' are stored.

- These positions are passed to the select line of the `Multiport Switch`.

The '`Multiport switch`' takes in Q bits at each cycle but only outputs the vector values that are given from the select line, an output of P.

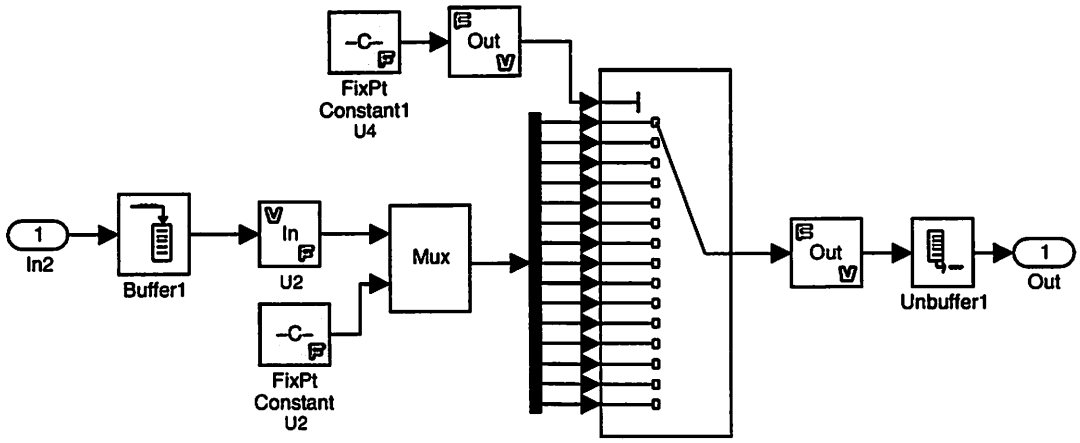

Figure 4.16 Fixed-Point Puncture Block

4.3.3 Fixed-Point Insert Erasure

The erasure fixed-point model was designed to have the same architecture as the fixed-point puncture model, so the models are the same. It made sense because erasure insertion is essentially the opposite of puncturing. It uses a width 15 `Multiport Switch` to accommodate the highest code rate 7/8. A length P input is received and is appended with zeros to create the length Q. The appendix of zeros is added via the `Constant` block connected to the `MUX`. The select line is fed a variable from another `Constant` block to determine the output ordering of the data stream. The output ordering is a vector that denotes the bit positions of the appended zeros back into the punctured data stream.

4.3.4 Fixed-Point Quantizer

The fixed-point equivalent requires that the range of the input signal be known so that a large enough wordlength can be chosen to accurately represent the input signal. The

block can handle 3, 4 and 5-level quantization by shifting the decimal point right for a higher soft-decision level, see Fig. 4.17.

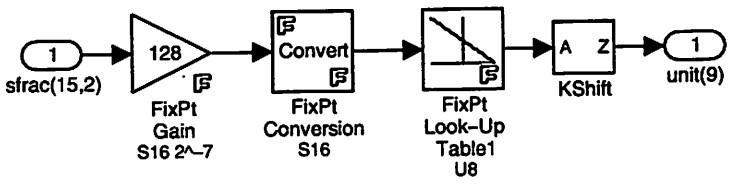

Figure 4.17 Fixed-Point Quantizer Block

4.4 Module Compiler Description

Module Complier™ (MC) is a software tool designed to explore different implementation architectures that have the same functionality. MC can also be used to describe the Power, Area, and Delay (PAD) of the data-path of a design. MC enables optimization of high performance data-path captured at the architectural level of abstraction. This is accomplished by writing a high-level description of the design and MC maps it into a gate-level description (VHDL, Verilog). It also generates an RTL model for fast functional simulation. If the goal is to bridge the gap between system designers and ASIC designers, then a compromise must be made at a level of abstraction everyone can use. MC uses an architectural level of abstraction, which is "high" enough for systems designers to understand since there is a one-to-one correspondence between the input MC language (MCL) and the architecture of the synthesized circuit. It is also "low" enough for ASIC designers because MC can generate a gate-level descriptor, e.g. VHDL.

It is for this reason that MC is used as the circuit level specification of the SSHAFT design flow. Designers can describe the fixed-point SIMULINK model in MC hardware language called MCL and generate behavioral VHDL. The behavioral VHDL can be run through a simulator to determine logical equivalency with the SIMULINK fixed-point model.

4.4.1 Viterbi Decoder

The MCMA framework consists of a floating-point and fixed-point representation in SIMULINK and a hardware description written in MCL code. The Viterbi algorithm when implemented is a highly data intensive algorithm and it made since not to represent it in both in a SIMULINK fixed-point representation and MC. It would be highly redundant and time consuming to do both. SIMULINK has a suitable built-in Viterbi decoder block that is highly parameterable and accurately represents the algorithm. Therefore, the MC description models the behavior of the built-in Viterbi decoder block.

The MCL code generated is based of the same fundamentals of the MCMA framework, flexible, modular, and parameterable. The following attributes of the decoder are:

Traceback length (σ): 120

- Soft input precision: 4 bits
- Constraint length (K): 9
- Generator polynomial: [753 561]

The SIMULINK model supports any of the above parameters and they can be set in the parameter mask of the SIMULINK model. All of the above are parameterable in MC as well, except for the constraint length, which has a unique construction.

The algorithm is explained in detail in Chapter 3 so the implementation will only be discussed here. There are three functional blocks in a Viterbi decoder as depicted in Fig. 4.18

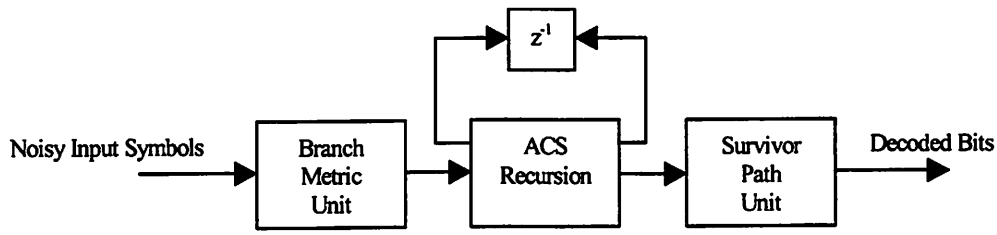

Figure 4.18 Functional Units of Viterbi Decoder

The branch metric unit, which calculates the distances between the noisy inputs and ideal symbols; the Add-Compare-Select (ACS) unit, which computes the state metrics; and the survivor memory unit, which keeps track of decisions made by the ACS units or the path through the trellis.

This is an n-bit soft-decision-input decoder, where *n=4*. The decoder takes as an input integer values from 0 to *a*, where $a = 2^n - 1$. The branch metric calculation is determined by using the Euclidean distance metric that can be simplified to:

$$bm0 = 2 * a - in0 - in1$$
$$bm1 = a - in0 + in1$$
$$bm2 = a + in0 - in1$$
$$bm3 = in0 + in1,$$

bm0 is calculated if the transmitted symbol is *00*, *bm1* is calculated if *01* is transmitted, *bm2* is calculated if *10* is transmitted, and *bm3* is calculated if *11* is transmitted. The input values are *in0* and *in1* range between [0,15]. A transmitted '0' is mapped to integer values [0,7] with integer value 0 being the strongest probability of a transmitted '0'. A transmitted '1' is mapped to integer values [8,15] with integer value 15 being the strongest probability of a transmitted '1'.

The ACS unit uses the maximum Euclidean decision metric to select the correct branch metric, that is, the largest branch metric of the two that appears at each state is chosen as the correct branch. Finally the survivor memory unit saves the selected branch metrics creating the survivor path metric, which gives the correct decoded sequence path. The traceback length determines the survivor memory unit length. The traceback length is usually 5 to7 times the constraint length. Puncturing requires this the traceback be larger since more time is need to merge given the insertion of the dummy bits.

Input Value	Interpretation Value
0000 (0)	0 *(strongest '0')*
0001 (1)	1
0010 (2)	2
0011 (3)	3
0100 (4)	4
0101 (5)	5
0110 (6)	6
0111 (7)	7 *(weakest '0')*
1000 (8)	8 *(weakest '1')*
1001 (9)	9
1010 (10)	10
1011 (11)	11
1100 (12)	12
1101 (13)	13
1110 (14)	14
1111 (15	15 *(strongest '1')*

Table 4.2 Soft Decision Levels

Architecture

A parallel architecture was chosen for implementation. The parallel architecture directly maps into hardware. Parallelism in implementation is faster, smaller, and more efficient. The disadvantage is that for large number of states, N, the global interconnects become a problem since the architecture scales with N, which exponentially grows with the constraint length K. It is also not suitable for implementations that require variable constraint lengths.

The data computation complexity can be described by the algorithm's trellis diagram, which contains $N = 2^{K-1}$ states at any given time.

Constant geometry was chosen because for the paralleled architecture because it's direct mapping into hardware made the design easier. In Fig. 4.19 the nodes of the trellis make up the states the decoder. From Fig. 4.19 it is clear there is a pattern in the implementation mapping. The pattern is that the incoming branch metrics for a given state come from the N and $N+\frac{2^{K-1}}{2}$ branch metric. Each ACS unit uses radix-2 update architecture, where each ACS unit calculates branch metrics for two states. Each ACS

unit calculates metrics for two states thus, there are half as many ACS units as there are states creating a more powerful efficient design. Radix-2 ACS update is used due to its simplicity and the speed gain. Using a radix-4 approach is only a factor of 1.4 faster but has an area overhead factor of 2.7 with respect to the radix-2 approach, if using 0.25 μm technology [25].

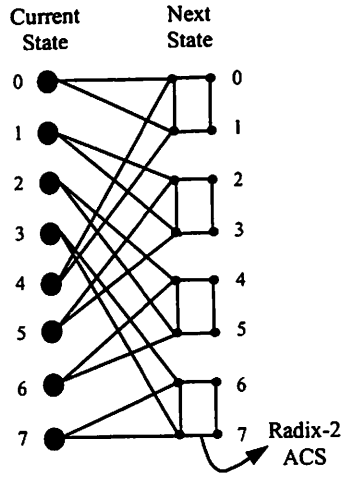

Figure 4.19 8-State Trellis Diagram

In order to keep the word length bounded when calculating the branch metric modulo arithmetic based normalization is used [26].

$$\lceil \log_2((\log_2 N + 1) \cdot \lambda_{max}) \rceil + 1 \qquad \textbf{(4.1)}$$

where $\lambda_{max} = 2 * a$ is the maximum branch metric value. The word length for the ACS given 4-bit input precision is 10.

The ACS units determine 2^{K-1} path decisions every clock cycle. The traceback stores the most recent σ path decisions (the parameter for σ is the traceback length) and uses these decisions to trace the most likely path through the trellis resulting from the last σ symbols. The method used here due to the lack of memory (SRAM) models, is a shift register of length σ. After σ clock cycles the last value in the shift register is read and should be the decoded bit for the time 0 noisy input symbol.

Performance

Tables 4.3, 4.4, and 4.5 summarize the different power, area, critical path delay, and decoding speed for different adder types. All the results are based on MC design reports using 0.25 μm technology with 1.6 V supply. MC allows optimizations to be made based on:

- Speed: Try to generate the fastest circuit possible
- Size: Ignore timing restrictions and minimize area
- Power: Ignore timing restrictions and minimize power

Tables 4.3, 4.4, and 4.5 show estimate PAD numbers for the different optimization settings.

MODULE COMPILER allows micro-architectures to be explored to determine if certain circuits provide a better performance. In this case different adder types where chosen for the ACS units since it is the dominant computation intensive portion of the decoder design. MC allows for 5 different adder types:

- Ripple: Smallest and slowest of all types and poor pipelining
- Carry Look Ahead Select (CLSA): Used when Ripple adder is too slow. Area is on the order of $\log_2$ of the bit width times larger than a Ripple adder.
- Carry Select (CSA): Generally larger and faster then CLA; works moderately well with pipelining
- Carry Look Ahead (CLA): Second fastest adder type and the most flexible. Can provide a structure ranging from Ripple to FASTCLA.
- Fast Carry Look Ahead (FASTCLA): The fastest and largest of all adder types.

The general theme is there is a decrease in area from FASTCLA to Ripple but increase in area from Ripple to FASTCLA.

Adder Types (Power)	Power (W)	Area (mm^2)	Critical Path Delay (ns)	Decoding Speed (Mb/s)
Ripple	0.0430	6.672	16.6	6.02
CLSA	0.0532	8.230	118.5	0.84
CSA	0.0330	6.197	16.9	5.92
CLA	0.0430	6.899	17.2	5.81
FASTCLA	0.0430	7.076	16.5	6.06

Table 4.3 MC estimates optimized for Power

Adder Types (Speed)	Power (W)	Area (mm^2)	Critical Path Delay (ns)	Decoding Speed (Mb/s)
Ripple	0.0430	6.781	12.2	8.20
CLSA	0.0832	10.185	123.6	0.81
CSA	0.0632	9.155	61.6	1.62
CLA	0.0430	7.144	10.8	9.26
FASTCLA	0.0732	10.015	19.5	5.13

Table 4.4 MC estimates optimized for Speed

Adder Types (Size)	Power (W)	Area (mm^2)	Critical Path Delay (ns)	Decoding Speed (Mb/s)
Ripple	0.0430	6.559	17.0	5.88
CLSA	0.0430	6.561	17.0	5.88
CSA	0.0330	6.184	16.9	5.92
CLA	0.0430	6.861	17.2	5.81
FASTCLA	0.0430	6.993	16.6	6.02

Table 4.5 MC estimates optimized for Size

The decoding speed is the inverse of the critical path delay. The power estimate for MC is inaccurate because it estimates the number of standard cells it would take to build the circuit and then derives some power consumption based on the number of standard cells. This is inaccurate because it doesn't take into consideration the switching activity of the circuit, which gives a better estimate of the power consumption of a given circuit. The Area estimate when optimized for size seemed to be the most accurate. The size of the design should increase from Ripple to FASTCLA and this is shown with the most precision when optimizing for size.

Design Exploration

What is now interesting is how changing parameters such as, traceback length, wordlength of ACS recursion, soft bit precision, and feature sizes affect the area estimate of the chip. The design environment at the BWRC has several technology libraries to chose from. The two major feature sizes to chose from are 0.25μm and 0.18μm. Within these two feature sizes there are technology files for different voltages and temperatures. To be consistent, the simulations are over both feature sizes at 85C°. The devices to be built here will most likely operate at room temperature (25C°), 85C°was the closest temperature available, the results between the two should be negligible. There is very little area change for a given technology. There is a factor of two decrease in area when changing to a new technology. The wordlength of the ACS unit is a function of the soft decision bits see Eq. (4.1). There is a one to one correspondence between an increase in the number of soft decision bits and the wordlength. In terms of the design exploration, varying the wordlength is equal to changing the soft decision bit. A wordlength increase from 8-bit to 16-bit for the ACS recursion is *~1.5x* increase in area. A 16-bit wordlength can be considered overkill for the decoder design. Assume there are 256 states (K=9), by using equation (4.1) using the entire bit width would require 10-bit soft decision precision. The tradeoff between complexity and performance after 5-bit precision is not worthwhile. Wordlengths between 8-12 will adequately cover most combinations of constraint lengths from 6 to 9 and soft decision values from 3-6 bits. The traceback depth is also another major area of consumption. The Simulink model could simulate with σ as low as 90. The design here used σ=120 because of the loss of resolution due to fixed-point representation.. The traceback depth is parameterable and it is interesting to know what the area consumption is over an acceptable BER performance. Figure 4.20 shows for a given wordlength the amount of area needed for traceback depth ranging from 90 to120. The tables assume 0.25μm at 1.6 V and a CLA adder type.

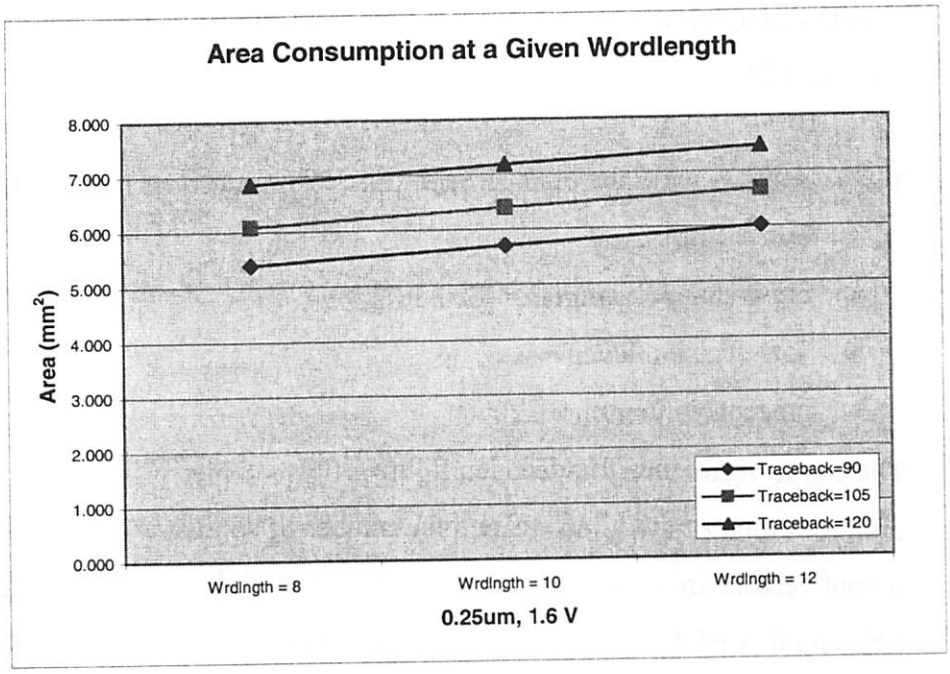

Figure 4.20 Area vs. Wordlength

The conclusion to be made is that there is no fluctuation in area due to change in a given technology. The wordlength of the ACS unit shows an average of 5% increase in area for every two bits. The traceback depth area increases by 2K-1*σ because each ACS unit has a shift register σ wide. As the number of states increases so does the number of shift registers along with a change in length of σ. For a fixed constraint length there is a 12% increase in area for every additional 15 increment increases in σ.

Verification

Finally the MC design and SIMULINK model are verified to be logically equivalent for a certain set of test vectors. MC generates behavioral VHDL, which can be put into Synopsys's VHDL Debugger to simulate the behavioral functionality of the MCL code [12]. In SIMULINK, for a given set of inputs a given set of outputs can be generated. In this case, the input vector is the input to the convolutional encoder and the output vector is the decoded sequence out of the Viterbi decoder. The input and output vector are fed in the Synopsys VHDL Debugger. The Debugger also generates an output based on these same input vectors, if the Debugger generated output vector is the same as the

SIMULINK model output vector the two are logically equivalent and the design is verified, see Fig. 4.21

The parameters set in the built-in SIMULINK Viterbi decoder block are:

- Constraint, K = 9
- Generator Polynomial = [753 561]
- Soft-decision level = 4
- Traceback Length = 120

The input vector into the decoder follows flow of Fig. 4.14. The output vector is the output of the decoder block. The total number of simulated bits is 500. These input and output vectors are entered into the Synopsys VHDL Debugger. Along with the generated behavioral VHDL for MC, the Debugger generates an output sequence based on the SIMULINK input vector. Fig. 4.21 shows the decoded output bits from the behavioral VHDL, B. and the decoded output bits from the SIMULINK decoder, B_DUM. The VHDL generated output vector is equal to the SIMULINK output vector, thus verifying logic equivalency.

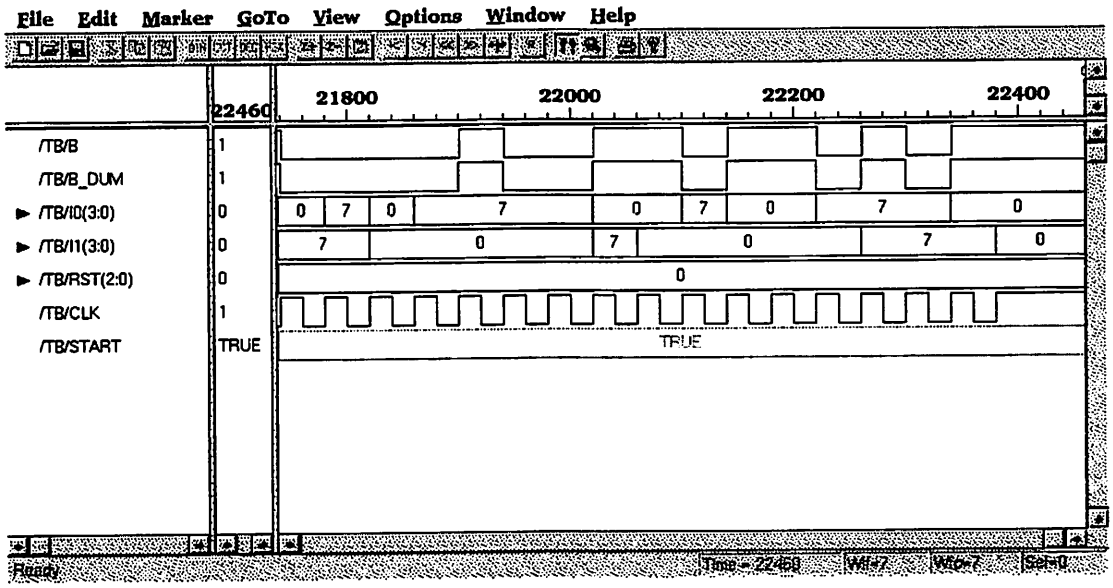

Figure 4.21 VHDL Debugger

Appendix

A4.1 MODULE COMPILER SOFTWARE DESCRIPTION

This appendix explains the MC language (MCL) code for the Viterbi decoder implemented in Section 4.4. As shown in Fig. 4.18 the branch metric unit, ACS recursion, and survivor path unit are what is needed to describe the behavior of the Viterbi algorithm. The code below covers how the branch metric calculation, ACS recursion, and , survivor path unit are implemented.

In order to determine the performance of different adder implementations the following was added:

```
directive (carrysave=cs, fatype=a_type, muxtype=m_type);
```

Using directives in MC allows for a simple parameter change at the main GUI and the five different adders discussed in section 4.4 can be tested.

A4.1.1 Branch Metric Calculation

The branch metric calculation is determined by equation (4.1). Each state in the trellis has a predetermined branch metric that can be based the output values of the encoder. In order to find the predetermined branch metric calculation of a given state, the encoder output sequence must be determined for both a logical '0' input and a logical '1' input. Once the encoder output is known for a given state, the branch metric calculated for that state is also known. For example, state0 has a given state where all the values in the memory elements of the encoder are equal zero. Let Fig. 4.2 be the encoder structure, for an input of '0' the encoder output is 00, for an input of '1' the encoder output is 11. Thus the branch metric calculated for state 0 is bm0 and bm3 as described the code below.

```
wire [soft_bit+1] bm0 = 14 - in0 - in1;  //if (00) is transmitted
wire [soft_bit+1] bm1 = 7 - in0 + in1;   //if (01) is transmitted
wire [soft_bit+1] bm2 = in0 + 7 - in1;   //if (10) is transmitted
wire [soft_bit+1] bm3 = in0 + in1;       //if (11) is transmitted
```

For the 256-state decoder described here, all the input and output characteristics of the states had to be determined in order to calculate the branch metrics corresponding to the state.

A4.1.2 Add Compare Select (ACS) Unit

The ACS unit does the following operations:

1. Calculates the incoming branch metrics for a given state
2. Compares the two competing branch metrics to determine which is the survivor partial path metric for a given state
3. Determines the correct decision bit for that state transition; determines if the survivor partial path metric came from the "upper" of "lower" branch metric

Let's look at the ACS function:

```
      function
ACS(d0,d1,nextstate0,nextstate1,state0,state1,branch0,branch1,branch2,br
anch3,cs,a_type,m_type);
      string cs, a_type, m_type;
      input state0, state1;
      input branch0, branch1, branch2, branch3;  // Branch metrics
      output unsigned [1] d0,d1;
      output nextstate0, nextstate1;
      integer wl = width(state0);


directive (carrysave=cs, fatype=a_type, muxtype=m_type);

// State metric accumulation
      wire unsigned [wl] sum0 = state0 + branch0;
      wire unsigned [wl] sum1 = state0 + branch1;
      wire unsigned [wl] sum2 = state1 + branch2;
      wire unsigned [wl] sum3 = state1 + branch3;

      // Hamming distance calculation
      wire signed [wl] comp0 = sum0 - sum2;
      wire signed [wl] comp1 = sum1 - sum3;

// MAXIMUM weight selection through trellis
      nextstate0 = comp0[wl-1] ? sum0 : sum2;
      nextstate1 = comp1[wl-1] ? sum1 : sum3;

// Decision bit
      d0 = comp0[wl-1];
      d1 = comp1[wl-1];

endfunction
```

MC requires that all function arguments be declared to initialize the variables of the function. As described in Fig. 4.19 the ACS unit calculates state metric for two

consecutive states (i.e., state0&state1; state2&state3...state(N)&state(N+1)). Each state has incoming branch metrics from the N[th] and $N+\frac{2^{K-1}}{2}$ state. The first state (state(N)) calculation of the ACS unit is calculated by `d0`, `state0`, `nextstate0`, `branch0`, and `branch1`. The second state (state(N+1)) calculation of the ACS unit is calculated by `d1`, `state1`, `nextstate1`, `branch2`, and `branch3`. The explanation here will focus on the ACS unit for state0 and state1. All other ACS units operate in the same fashion, so the explanation of one ACS unit is sufficient. The output decision variable for the state0 is `d0`. The `nextstate0` variable stores the surviving state metric for `state0`. The `state0` variable specifies the current state metric and is added to incoming branch metric to determine the survivor path.

The state metric accumulation is handled by:

```
// State metric accumulation
      wire unsigned [wl] sum0 = state0 + branch0;
      wire unsigned [wl] sum1 = state0 + branch1;
      wire unsigned [wl] sum2 = state1 + branch2;
      wire unsigned [wl] sum3 = state1 + branch3;
```

The `sum` variables add the current state value to the incoming branch metric of that state. For example, for `state0`, the `sum0` and `sum2` variables calculate the partial path metrics for the upper and lower branch metrics that enter that state. For `state0` the incoming branch metric is from the N[th] state if the input is a transmitted logical '0' or the branch metric is from the $N+\frac{2^{K-1}}{2}$ state if the input is a transmitted logical '1'.

The Hamming distance calculation is done by:

```
// Hamming distance calculation
wire signed [wl] comp0 = sum0 - sum2;
wire signed [wl] comp1 = sum1 - sum3;
```

For comp0, the input branch metrics are subtracted from one another to determine if the comparator value is positive or negative. From

```
// Path selection through trellis
nextstate0 = comp0[wl-1] ? sum0 : sum2;
nextstate1 = comp1[wl-1] ? sum1 : sum3;
```

if the `comp0` value is positive then the lower branch metric is selected, `sum2`. If comp0 is negative then, sum0, the upper branch metric is selected. Finally, the `d0` variable is the result of the nextstate0 variable and corresponds to the selection of the survivor partial path metric, either the upper (state(N)) branch metric or lower (state(N+1)) branch metric.

Only one ACS unit is described here since they all operate the same way, except for different branch metrics.

```
ACS(d0,d1,nextstate0,nextstate1,state0,state128,bm0,bm3,bm3,bm0,cs
,a_type,m_type);
state0 = reset ? 0 : sreg(nextstate0);
state1 = reset ? 0 : sreg(nextstate1);
nextsp0 = d0 ? cat(sp0[sp1-2:0],0) : cat(sp128[sp1-2:0],0);
nextsp1 = d1 ? cat(sp0[sp1-2:0],1) : cat(sp128[sp1-2:0],1);
sp0 = sreg(nextsp0);
sp1 = sreg(nextsp1);
```

From this example, the state transitions are for `state0 (state(N))` and `state128` (state $N+\frac{2^{K-1}}{2}$, where K = 9), see Fig. 4.19. If we relate this to the `ACS` function then we replace `state0` with `state0` and `state1` with `state128`. The `nextsp0 = d0 ? cat(sp0[sp1-2:0],0) : cat(sp128[sp1-2:0],0);` line selects based on `d0`, what path should be chosen for state0. If d0 is positive then the lower path from the $N+\frac{2^{K-1}}{2}$ branch metric is chosen, `sum128`. If `d0` is negative then the upper path from the N^{th} branch metric is chosen, `sum0`. This also follows for the second state calculation,

nextsp1. The `nextsp0` is stored in a shift register, by the following line: `sp0 = sreg(nextsp0);`. This leads to the discussion of the survivor path unit.

A4.1.3 Survivor Path Unit

The survivor path unit is incorporated in the ACS function:

```
nextsp0 = d0 ? cat(sp0[spl-2:0],0) : cat(sp128[spl-2:0],0);
nextsp1 = d1 ? cat(sp0[spl-2:0],1) : cat(sp128[spl-2:0],1);
sp0 = sreg(nextsp0);
sp1 = sreg(nextsp1);
```

The traceback length is defined as the fixed decoding delay σ. The variable `spl` is the traceback length and is set to a value of 120. The `cat(sp0[spl-2:0],0) : cat(sp128[spl-2:0],0);` declaration in the function appends input values of length spl-2 :0 or 118:0 to create the an output vector length of 119. The 119-length register stores the partial path metrics for the given state. For example, in the command `nextsp0 = d0 ? cat(sp0[spl-2:0],0) : cat(sp128[spl-2:0],0);` the given state is `state0` and a decision is stored on if the incoming branch metric came from the upper path,. `cat(sp0[spl-2:0],0)`, or the lower path, `cat(sp128[spl-2:0],0.` The width of `nextsp0` inherits the width of the value it is equal to, thus the width of `nextsp0` is 118. The `sp0 = sreg(nextsp0);` becomes a 118-length shift register. New partial path metrics are stored for each clock cycle and after 118 clock cycles the oldest partial path metric becomes the decoded bit. The same metric is done for the following `nextsp1` declaration.

At the end of the file is the command:

```
bit = sp0[spl-1];
```

It outputs the stored values in the shift register. At each new input bit received a decision on the bit received σ=118 clock cycles is outputted. The traceback length σ is long enough that all surviving sequences at time t stem from the same node t-σ clock cycles back. As a result, all states should have merged by the 118^{th} clock cycle so the selecting of `sp0` is arbitrary, any of the 256 states can be chosen as outputting the decoded sequence.

5

Conclusion

5.1 Summary

The MCMA framework is to describe sophisticated digital communication algorithms that exploit diversity in three dimensions: time, frequency, and space. "The Hornet proposal:" by [3] was the first attempt to capture the formulation of this framework at a simulation level. The work here was to create a flexible error correction scheme that can be used in a variety of complex system simulations. A floating-point and fixed-point adaptive punctured convolutional coding scheme has been introduced with code rates of 1/2, 2/3, 3/4, and 7/8. A parameterable parallel Viterbi decoder design in MODULE COMPILER has also been designed. The decoder is a design example to show how the MCMA framework integrates with SSHAFT design flow.

5.1.1 Design Flow

The MCMA framework was also created with the SSHAFT design flow in mind. The MCMA framework should contain algorithmic blocks that are modular in design and easy to parameterize. The framework should enclose three types of libraries:

- Floating point – Demonstrates algorithmic functionality using SIMULINK
- Fixed-point – Models degradations due to rounding, truncation and word length limitations. Verifies "logic equivalency" with Module Complier generated behavioral VHDL code
- MODULE COMPILER – Gives rough, Power, Area, and Delay estimates of fixed-point design. Generates behavioral VHDL code to simulate the algorithm system created in SIMULINK. Combined with fixed-point design rapid realization to silicon can be created by using the push button design flow developed

Given the creation of blocks in these libraries, complex systems can easily be designed at a high level of abstraction to verify functionality. Using the SSHAFT design flow entire systems can be developed or analyzed at the low level of abstraction to determine the feasibility of systems.

5.2 Future Work

In terms of the work described here several more steps need to be carried out. The Puncture and Insert Erasure blocks were created using blocks from the fixed-point blockset of SIMULINK. The majority of these blocks exists are operations that exist as standard cells that are native to the design flow. This system should be "pushed" through the design flow, fabricated and tested to show proof-of-concept that the MCMA framework and SSHAFT design flow work together. The Puncture and Insert Erasure blocks need logically equivalent MC generated behavioral VHDL code to create circuit level specification needed for the SSHAFT design flow.

Higher order modulation schemes should be researched and created for the framework. The current system does not boast high data rates due to the efficiency of QPSK at 2

bits/symbol. Much higher-level modulation such as 16 (4 bits/symbol), 32 QAM (5 bits/symbol), and 64 QAM (6 bits/symbol), could alone increase bandwidth efficiency up to 6x. However, multi-level modulation techniques require a robust timing recovery and carrier synchronization architecture that has not yet been finalized.

More complex error coding could be researched. Coffey [27] has developed space-time codes that exploit the temporal component of MEA algorithms. Using Reed-Muller block codes Davis and Jedwab [28] have attempted to control the peak-to mean power control problem inherent in OFDM.

Finally, a specific application for this work needs to be finalized. It is difficult to truly optimize a physical layer description without detailed knowledge of the end application and its requirements (e.g., BER, bit rate, packet length). Once a suitable application has been identified, and the framework is proliferated with the major building blocks of a digital communications truly complete systems can be built.

References

[1] J. Winters, J. Salz, and R. Giltin, "The impact of antenna diversity on the capacity of wireless communication systems", *IEEE Transactions on Communications*, Vol. 42, No. 4, pp. 1740-1751, Apr. 1994

[2] SSHAFT group homepage, http://bwrc.eecs.berkeley.edu/research/ic_design_flow

[3] A. Klein, *The Hornet Proposal: A Multicarrier, Multiuser, Multiantenna System Specification*, Master Project Report, University of California at Berkeley, Fall 2000

[4] C. Shi and K. Camera, *Polyphase Filter Bank Structures For a Space Radar System*, EE225C Class Project, University of California at Berkeley, Fall 2000

[5] S. Poon, D. Tse, and R. Brodersen, "An Adaptive multi-antenna transceiver for slowly flat fading channels", to appear in *IEEE Transactions on Communications*, 2001

[6] A. Peled, A. Ruiz, "Frequency domain data transmission using reduced computational complexity algorithms", *IEEE Int. Conf. Acoustics, Speech, Signal Processing*, pp. 964-967, Denver, Co, 1980

[7] T. Rappaport, *Wireless Communications Principles and Practice*, Prentice-Hall, Englewood Cliffs, N.J, 1996

[8] J. Proakis, *Digital Communications*, McGraw-Hill, 1995

[9] P. Husted *Design and Implementation of Digital Timing Recovery and Carrier Synchronization for High Speed Wireless Communications*, Master Project Report, University of California at Berkeley, Fall 1999

[10] G. Foschini, M. Gans, "On the limits of wireless communications in a fading environment when using multiple antennas", *Wireless Personal Communications*, vol. 6, No.3, pp.311-335, Mar. 1998

[11] N. Zhang, B. Haller, and R. Brodersen, "Systematic architecture exploration for implementing interference suppression techniques in wireless receivers", *IEEE Workshop on Signal Processing Systems*, pp. 218-227, Oct. 2000

[12] W. Davis, N. Zhang, K. Camera, F. Chen, D. Markovic, N. Chan, B. Nikolic, R. Brodersen, "A Design Environment for High Throughput, Low Power Dedicated Signal Processing Systems", *IEEE Custom Integrated Circuits Conference*, May, 2001

[13] C. Shannon, "Mathematical Theory of Communication", *Bell Systems Technical Journal*, Vol.28, pp. 379-423 and pp. 623-656, 1948

[14] P. Elias, "Coding for Noisy Channels", *IRE Conv. Record*, Part 4, pp. 37-47, 1955

[15] J. Wozencraft and B. Reiffen, *Sequential Decoding*, MIT Press, Cambridge, Mass., 1961

[16] A. Viterbi, "Error Bounds for Convolutional and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, Vol. IT-13, pp. 260-269, Apr. 1967

[17] S. Wicker, *Error Control Systems: for digital communication and storage*, Prentice-Hall, Upper Saddle River, NJ, 1995

[18] D. Daut, J. Modestino, and L. Wismer, "New Short Constraint Length Convolutional Code Constructions for Selected Rational Rates", *IEEE Transactions on Information Theory*, Vol. It-28, No. 5, pp. 794-801, Sep. 1982

[19] G. Forney, "The Viterbi Algorithm", *Proceedings of the IEEE*, Vol.61, pp. 268-278, March 1973

[20] J. Cain, G. Clark, and J.Geist, "Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding", *IEEE Transactions on Information Theory*, Vol. IT-25, No.1, pp. 97-100, Jan. 1979

[21] S. Lin and D. Costello, *Error Control Coding: fundamentals and applications*, Prentice-Hall, Englewood Cliffs, N.J., 1983

[22] C. Lee, *Convolutional Coding: fundamental and applications*, Artech House, Norwood, MA, 1997

[23] Y.Yasuda, K. Kashiki, and Y. Hirata, "High-Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding", *IEEE Transactions on Communications*, Vol. No.3, pp. 315-319, Mar. 1984

[24] K. Hole, "New Short Constraint Length Rate (N-1)/N Punctured Convolutional Codes for Soft Decision Viterbi Decoding", *IEEE Transactions on Communications*, vol. No. 9, pp. 1079-1081, Sep. 1988

[25] P. Black, and T. Meng, "A 140-Mb/s, 32-State, Radix-4 Viterbi Decoder", *IEEE Journal of Solid State Circuits*, Vol. 27, No. 12, pp. 1877-1885, Dec. 1992

[26] [Shu90] C. Shung, P. Siegel, G. Ungerboeck, H. Thapar, "VLSI Architectures for Metric Normalization in the Viterbi Algorithm", IEEE International Conference on Communications, pp. 1723-1728 vol. 4, Apr. 1990

[27] G. Raleigh, and J. Cioffi, "Spatio-Temporal Coding for Wireless Communication", *IEEE Transactions on Communications*, Vol. 46, No. 3, pp. 357-366, Mar. 1998

[28] J. Davis, and J. Jedwab, "Peak-to-mean power control in OFDM, Golay complementary sequences, and Reed-Muller codes", *IEEE Transactions on Information Theory*, Vol. 45, No. 7, pp. 2397-2417, Nov. 1999