Design and Testing of a Real Time Simulation for Trellis Coded Modulation

by

Allon Benzakein

B.Sc(Eng) *Cape Town*

Submitted to the Department of Electrical Engineering
in partial fulfillment of the requirements for the degree of

MSc(Eng)

at the

UNIVERSITY OF CAPE TOWN

February 1995

signature removed

Signature of Author .....
Department of Electrical Engineering
January 30, 1995

Certified by .....
Dr. Robin M. Braun
Director of the Communications Research Group
Thesis Supervisor

Accepted by .....
Prof. Barry J. Downing
Head of Department

Declaration

I declare that this thesis is my own unaided work. It is being submitted for the degree of Master of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination.

signature removed

.....

(Signature of Candidate)

30 January 1995

Acknowledgements

I would like to thank my parents Mercia and Vivian for their constant encouragement, support and unfailing confidence in me.

Thanks are also extended to supervisor Robin Braun for his help and guidance.

I would like to acknowledge two of my fellow students. Michael Horwitz for his thorough proof reading and Benno Bohm for his advice and help.

Synopsis

The aim of this project is to build and test a real time simulation for Trellis Coded Modulation(TCM). The tests to be performed are a comparison between Ungerboeck and pragmatic codes and varying and observing different Viterbi decoder parameters for coded 8PSK.

TCM is coded modulation which means the choice of modulation scheme is linked with the encoding technique. Convolutional codes are important for an understanding of TCM. They are described by the rate (which is the number of inputs over the number of outputs) and the number of memory elements, ν. For TCM schemes, soft decision decoding, based on euclidean distance, rather than hard decision decoding, based on hamming distance, is used. Ungerboeck developed a mapping of encoder bits to channel signals on a constellation diagram. The mapping is called mapping by set partitioning and aims to find the smallest free euclidean distance for a given code. The free euclidean distance is the minimum euclidean distance between output sequences in a code. The asymptotic coding gain is a measure of coding gain based on the free euclidean distance.

The Viterbi algorithm is a maximum likelihood decoder for convolutional codes. For practical systems the truncated Viterbi algorithm is used. The truncated Viterbi algorithm provides suboptimal performance depending on the value of the truncation depth W. The Viterbi algorithm for TCM involves the computing branch metrics, based on

the Euclidean distance between the received signal and all the expected constellation points and adding these branch metrics to path metrics to obtain survivor path metrics for each state. A symbol is then decoded by choosing the state with the lowest survivor metric. Hardware implementations consist of an add compare select unit and a storage-decision module. For the implementation in this thesis, register exchange was used for the storage-decision module.

Pragmatic codes allow designers to use the best 1/2 rate convolutional codes with a specific mapping in a TCM system. The same code is used for MPSK with M=4,8 or 16. The resulting loss of performance is balanced out by the fact that chips for implementing encoding and Viterbi decoding are readily available. Tests on pragmatic codes in this project aim to estimate the extent of this degradation by comparing the pragmatic code with best Ungerboeck for $\nu = 3$. In order to perform the comparison the best $nu = 3$ 1/2 rate convolutional code was found.

The real time simulation was a combination of hardware and software. In order to simplify the simulation only base band I and Q signals were generated and no modulation or demodulation was performed. The 8PSK level shifter provides the correct I and Q levels using a digital to analogue converter. The encoder can implement either the Ungerboeck or the pragmatic code. As input a pseudo random bit sequence(PRBS) is generated. The analogue to digital card is a PC plug in card which sample the I and Q signals. Noise is generated using a filtered PRBS and added to both I and Q channels. The software performs the Viterbi algorithm for both Ungerboeck and pragmatic codes. Other functions to get average values for the constellation points, check for errors and synchronize the decoded symbols with the input PRBS, have been implemented. A separate programme was also written to perform the 4PSK decoding.

Tests on uncoded 4PSK closely matched the theoretical prediction. Error events were used to measure coding gains for the Ungerboeck code and to compare the Ungerboeck with the pragmatic codes with a truncation depth of 20. The codes had similar coding gains with the pragmatic code performing better in terms of probability of symbol

error while the Ungerboeck code performed better in terms of error event probability. Error event length histograms were obtained for both Ungerboeck and pragmatic codes. Important characteristics of each code can be seen from these histograms. Tests were also performed with varying truncation depths. These tests revealed that Ungerboeck codes require a truncation depth, W, greater than 20 while increasing W above 20 for pragmatic codes yielded little performance improvement. Event length histograms for different truncation depths yielded interesting results that enabled estimation of the number of unmerged paths for a given value of W. This information is important in deciding on optimum values for truncation depths. Tests on the number of quantization bits showed that for the optimum trade off between performance and complexity 6 bits in each the I and Q channels should be used. Observation of survivor path metrics showed that the slope increased for decreasing signal to noise ratios, a factor the can be used in synchronization for carrier recovery schemes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

TCM is a bandwidth and power efficient coded modulation technique. It provides coding gains which exceed traditional convolutional coding methods. It is for this reason that the Digital Radio Communication group at the University of Cape Town has an interest in developing knowledge and experience in this field.

The first student in this group to work with TCM was Theo Lindebaum. The work done was in creating a software simulation environment for simulating TCM. The thesis described in this document follows on from the work done by Theo Lindebaum and puts into practice a real time TCM simulation with a combination of hardware and software. The objectives of this thesis are as follows:

- Build a real time 8PSK TCM simulation with available hardware and software.
- Test the system with both the optimum Ungerboeck codes and pragmatic codes and compare the two in terms of error performance and arrive at estimates of their coding gains.
- Vary and observe various decoder parameters, plot the results and draw conclu-

sions as to the trade offs between decoder complexity and performance.

- Pass on experience gained in TCM and Viterbi decoding with a view towards future members of the group implementing a high speed, low cost full TCM system.

This report provides a theoretical background to TCM necessary for an understanding of the implementation, details the design and construction of the system and finally provides the results of the tests and the interpretation of these results. Specifically: chapter 2 deals with the Principles of TCM, explaining both the construction of basic Ungerboeck codes and their evaluation. In chapter 3 Viterbi Decoding is explained in detail as it is an important limiting factor in any TCM implementation. Being one of the focuses of the test setup, pragmatic codes are dealt with in chapter 4. Chapters 5 deals with the implementation of the simulation describing the degree of abstraction and detailing both the hardware and sofware aspects of the simulation. In chapter 6 the results are presented and interpreted. Finally in chapter 7 conclusions are drawn and recommendations made.

Appendix A shows how to calculate the standard deviation for probability of error tests and shows samples of the results for the tests performed on Ungerboeck and pragmatic codes.

Chapter 2

Principles of TCM

The block diagram of a General Digital Communication system is shown in Figure 2-1. At the transmitter side, the data source is the block producing the data that needs to be sent, the source encoder aims to eliminate redundancy of the source by using, for example, Huffman coding. The channel encoder introduces redundancy for the purposes of error checking or error correction. Finally the coded information modulates a carrier in the modulator and the signal T(t) is transmitted over the channel. At the receiver the reverse is done with the channel decoder deciding on the most likely information originating from the source based on the output of the demodulator which in turn depends on R(t) a noisy version of the transmitted signal, T(t).

One of the most important features of TCM is fact that it is *Coded Modulation.* What this means in terms of Figure 2-1 is that in the transmitter, the channel encoder and the modulator are combined and at the receiver, the demodulator and the channel decoder are combined. Combining the channel encoder and the modulator implies that the choice of modulation scheme and channel encoder is not made independently, in addition there is a very specific mapping of encoder output to modulator symbol. This

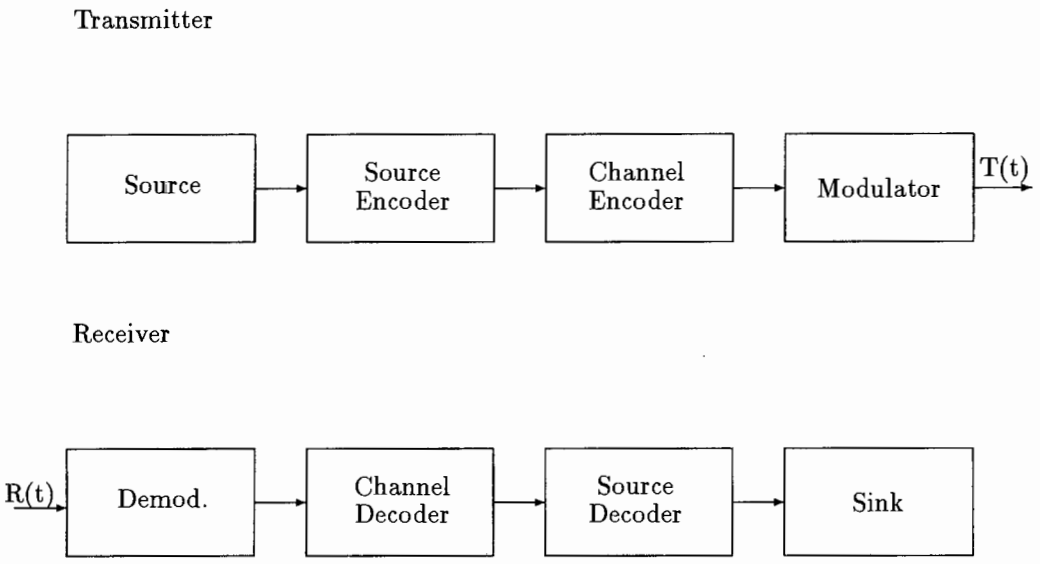

Figure 2-1: Block Diagram of a General Communication System

is in contrast to traditional Forward Error Correction (FEC) techniques where the encoder is optimized independently and can be used with any modulation scheme.

2.1 Motivation for TCM

Channel Capacity provides a fundamental limit to the amount of information, in bits per channel use that can be transmitted over a noisy channel. Shannon's second theorem states that given unlimited channel coding, error free transmission is possible provided transmission is below the channel capacity. Conversely error free transmission is not possible at a rate, in bits per channel use, above the channel capacity. Figure 2-2 shows this capacity for MPSK modulation schemes as well as the Shannon limit, $log_2(1+SNR)$, which cannot be improved upon by any modulation scheme. The graphs are plotted for channels with discrete valued input and continuous valued output. Continuous valued output is important because at the receive side *soft decision* decoding is assumed [1]. Soft decision decoding will be explained in detail in the next chapter.

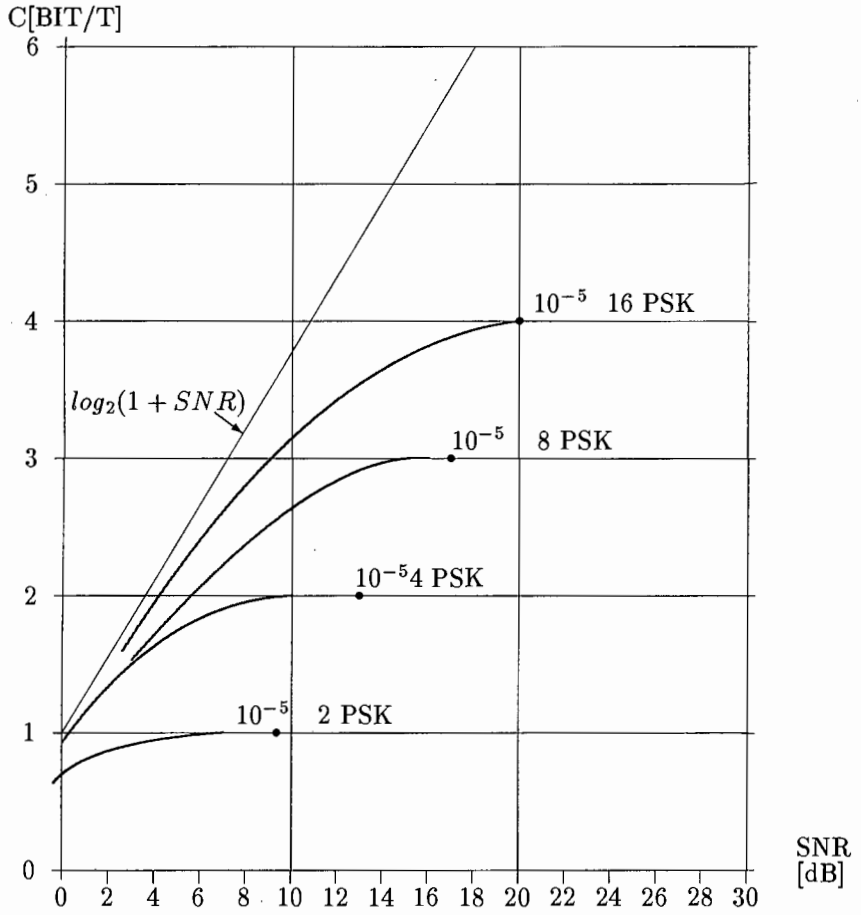

Figure 2-2: Capacity of MPSK Modulation Schemes

Figure 2-2 provides a visual way of appreciating the improvements that can result from a TCM scheme. Assume a communication system is required to transmit 2 information bits per channel use. Of the modulation schemes shown in the graph the simplest and most obvious choice would be 4PSK. Using this scheme, a probability of error of 10^{-5} occurs for SNR = 12.9dB. If, however, redundancy is added through channel coding and the expanded signal set of 8PSK is used to transmit 2 information bits, looking at the graph, error free transmission is already theoretically possible at 5.9dB. This can be verified by checking where the graph for the capacity of 8PSK intersects the required 2 bits per symbol line. This improvement is achieved without a bandwidth penalty ie the bandwidth for the uncoded 4PSK is identical to that of the coded 8PSK. Using signal constellations of higher order (16PSK or higher) schemes can only at best result in a 1.2dB lowering of SNR for the same capacity compared with coded 8PSK. The idea of adding redundancy and mapping this redundancy to a higher order modulation

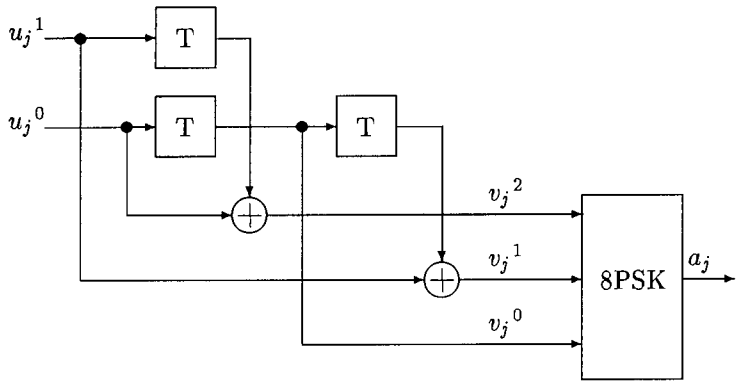

Figure 2-3: Convolutional Encoder

with double the number of signal points is central to TCM schemes.

The emulation described in this document implements the same scheme described above. 2 bits of information are transmitted using coded 8PSK and the resulting gains are compared to the 4PSK uncoded case.

2.2 Convolutional Codes

Convolutional Codes are a form of channel coding where the code words are generated by linear operations on the source bits. An example of the best 8 state TCM convolutional encoder for 8PSK is shown in Figure 2-3. Convolutional codes are traditionally not linked to a modulation scheme or mapping, but in this case, the code was specifically designed for use with 8PSK and with a specific mapping which will be described later in this chapter. This encoder implements a (3,2,2) code, the 2 input bits $u_j{}^0, u_j{}^1$ and 3 outputs $v_j{}^0, v_j{}^1, v_j{}^2$ are shown at time unit j. In general a convolutional encoder for an (n,k,m) code has n outputs, k inputs and memory order m and is referred to as a rate k/n code. Note all $\oplus$ operators represent modulo-2 addition.

In researching the notation for convolutional codes, the author found various anomalies in the literature with respect to encoder memory and code constraint lengths. In order

to avoid confusion the different notation will be explained and a notation developed that will be used throughout this document.

The encoder shown in Figure 2-3 has *memory order m=2.* This means the length of the *longest shift register* is 2. Although there are 3 memory elements in the encoder, only 2 of them are connected with the output of one connected to the input of the next. Stated more formally, if K_i is the length of the ith shift register, then

$$m \equiv \max_{1 \leq i \leq k} K_i \tag{2.1}$$

The *constraint length* is then defined as

$$n_A \equiv n(m+1) \tag{2.2}$$

The interpretation of this definition is that the constraint length is *the maximum number of output bits that can be affected by a single input bit* [2]. A more useful definition for TCM is the maximum number of time units over which an input bit affects an output bit. This is simply m+1 and will be referred to as the time constraint length n_t. In his seminal paper on TCM [1] Ungerboeck defines the constraint length of a convolutional code, ν, to be the number of memory elements in the encoder. This is contrary to both of the above definitions. In the comparison of equivalent pragmatic and Ungerboeck codes, the important factor is not constraint length but decoder complexity. This depends solely on the number of states in the encoder which in turn depends on the number of memory elements in the encoder. In this document ν will be used to refer to the number of memory elements in a convolutional encoder. For the encoder of Figure 2-3 $\nu = 3$.

Convolutional codes get their name from the fact that the output sequences are discrete

convolutions of the input sequence with the impulse response of the encoder. These impulse responses can be represented as polynomials in D, the delay operator, with the power of D representing the number of time units a bit is delayed. In this transform domain, the convolution of an input sequence with an impulse response to get an output, can be replaced by polynomial multiplication. For any (n,k,m) code there are k·n generator polynomials with the highest order polynomial having order m. The generator polynomials are of the form $g_i^{(j)}$ which relates the ith input to the jth output. This can be represented in matrix form. For the encoder example given the generator matrix is

$$\mathbf{G}(D) = \begin{bmatrix} D & 1 & 0 \\ 1 & D^2 & D \end{bmatrix} \tag{2.3}$$

For (1,2,m) codes which are the type used for pragmatic codes, the code is often represented by two octal numbers. These are the impulse responses, in octal, of the two outputs.

2.3 Hard vs. Soft Decision Decoding

In Figure 2-1 at the receiver, the channel decoder needs some measure/metric of how close the received signal is to any of the expected symbols. The difference between soft decision and hard decision decoding is the metric that is passed from the demodulator to the channel decoder. For the 8PSK constellations of Figure 2-4, symbols are at phase offsets of $\theta = 2\pi i/8 + \pi/8 \quad i = 0, 1, ..., 7$ from the carrier and correspond to the three bit binary numbers shown. In a scheme employing hard decision decoding, decision boundaries would be at $\theta_D = 2\pi i/8 \quad i = 0, 1, ..., 7$. Examples of these decision boundaries are shown for i = 0,1,2 in the left constellation of Figure 2-4. If a symbol at point a is detected at the demodulator, the information passed on to the

Figure 2-4: Hard vs. Soft Decision for an 8PSK Constellation

channel decoder is that symbol 001 was received because a falls within the decision boundary of the 001 symbol. The channel decoder therefore uses Hamming Distances in its algorithm in order to decode the sequence.

In contrast, soft decision decoding does not make any decision about which constellation point is received before decoding is complete. Instead, the Euclidean distances between the received point and the all 8 constellation points are computed and passed on as metrics to the decoding algorithm. In the right constellation of Figure 2-4, the Euclidean distance between the same point a, and the constellation point represented by 2, is shown. Note in this constellation the decimal equivalent numbers are given because the binary numbers have no relevance in working out metrics. These decimal equivalent numbers are also referred to later in the trellis representation.

Convolutional coding techniques are often referred to as Forward error correction(FEC). This is not accurate when applied to soft decision decoding because no decision is made as to the received symbol until decoding is complete. There are therefore no errors to be corrected. Hard decision decoding however can, in the presence of noise, pass on incorrect estimates of the received channel signal to the channel decoder and it is the purpose of the decoder to correct these errors. Thus it is error correction.

2.4 Mapping By Set Partitioning

While section 2.1 showed the existence of codes which, using expanded signal sets, improve the channel capacity, they give no way of formulating these codes. The key to achieving this gain is in the mapping of channel encoder outputs to channel constellation points. In his seminal paper [1] Ungerboeck developed a method of working out an optimal mapping for various constellations. He called this mapping, Mapping by Set Partitioning. It will be presented here for an 8 state 8PSK code.

An important quantity when evaluating TCM codes is the free Euclidean distance, d_{free}. The free Euclidean distance of a convolutional code is defined as

$$d_{free} = \min_{\{a_j\} \neq \{a_j'\}} \left[\sum_n d^2(a_j, a_j') \right]^{1/2} \tag{2.4}$$

Where $d^2(a_j, a_j')$ is the squared Euclidean distance between channel signals a_j and a_j' and the minimum is over all valid channel *sequences* $\{a_j\}$ and $\{a_j'\}$. It is the aim of Ungerboeck's mapping by set partitioning to produce codes with maximum d_{free} for a given constellation and number of memory elements ν.

Figure 2-5 shows the trellis diagram that results from the encoder of Figure 2-3 and the two sequences that account for the free distance. Trellis diagrams are essential for an understanding of Mapping by Set partitioning and Viterbi Decoding which is dealt with in the next chapter. The main trellis diagram is located between the first two sets of eight nodes and shows the state transitions from time $j-1$ to time j. Each node in the trellis diagram represents a state in the encoder corresponding to a 3 bit binary number which is the contents of the three memory elements in the encoder. The states, m_j, at time j are numbered 0 to 7 from top to bottom. Each state has 4 branches leaving it, these branches denote the four channel symbols and corresponding four state transitions that result when the two bit input symbol, $({u_{j-1}}^1, {u_{j-1}}^2)$ is applied to the

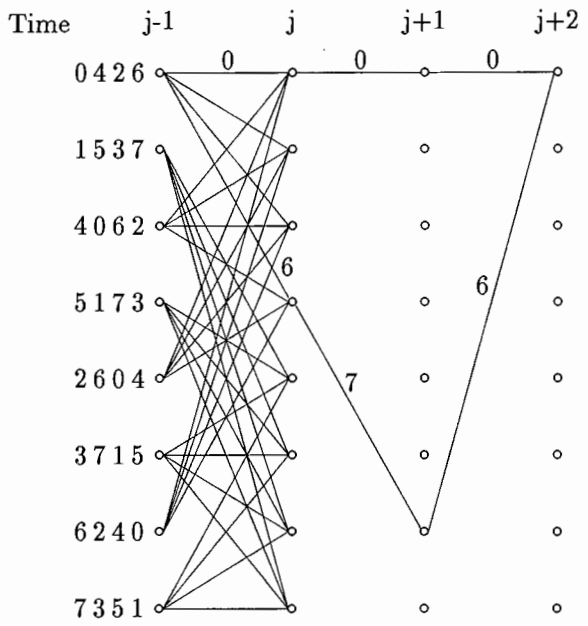

Figure 2-5: Ungerboeck 8 State 8PSK Trellis Diagram

encoder. The branches are those resulting from inputs 0 to 3, from top to bottom, for each state. To the left of the first set of nodes are numbers which show the four channel outputs, a_{j-1}, for the four state transitions, they too are labelled from top to bottom. These numbers are mapped to constellation points with the same mapping as Figure 2-4. So, for example, if the encoder is in state 6 at time $j-1$ and the encoder input is 10(binary) or 2(decimal), the encoder would change to state 2 at time j with an output, a_{j-1}, of 4.

Figure 2-6 shows Ungerboeck's Mapping by Set Partitioning. Starting with the original constellation, A0, the constellation is successively partitioned into two with the filled points showing the members of each set. The minimum Euclidean distance, Δ, between signals in a set increases as the sets are partitioned. Distances are calculated assuming an average signal energy of 1. Bearing in mind the purpose of the mapping is to obtain a code with the maximum possible d_{free} the following rules [1] were formulated:

1. Channel signals should be transmitted equally often.

2. Branches originating from the same state should be assigned signals from subset

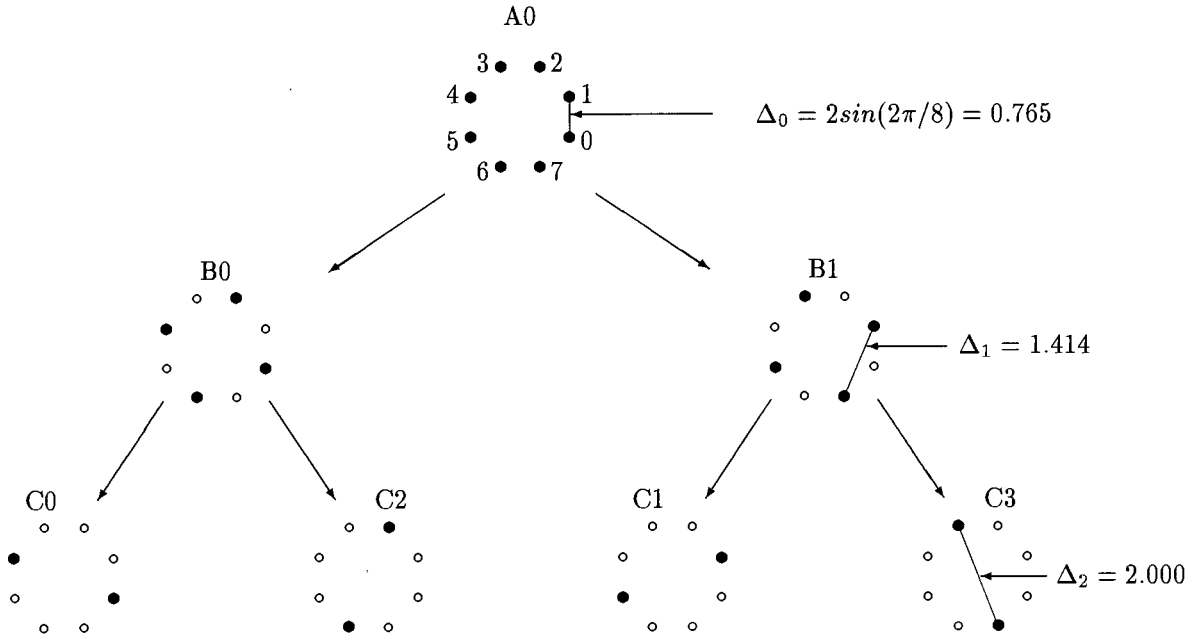

Figure 2-6: Mapping by Set Partitioning

B0 or B1.

3. Branches joining at the same state should be assigned signals from subset B0 or B1.

4. Parallel transitions (transitions where two branches join the same two states) should be assigned signals from subset C0 or C1 or C2 or C3.

Rule 1 reflects a general guideline for good codes that channel signals should be well scrambled with no signal being favoured. Rules 2 and 3 ensure that the free distance of the code is at least twice the minimum distance Δ_1 of subsets B0 and B1. Rule 4, while not appropriate for the code shown in Figure 2-5 because it has no parallel transitions, is relevant for pragmatic codes, and is used in formulating pragmatic codes.

The trellis in Figure 2-5 was created using the above rules. Looking at the numbers on the left representing channel outputs, rule 2 is apparent: all the sets of 4 are signals from either B0 or B1, never from both. These are the channel signals originating

from the same state. Likewise, for rule 3, looking at the second set of nodes, the four branches joining at any state are either from B0 or B1. For example, the four channel signals joining at state 5 are (from top to bottom) 5,1,7,3 which are all from subset B1.

The proof of the success of the code is in the calculation of d_{free}. Out of all possible pairs of channel sequences, $\{a_j\} \neq \{a'_j\}$, two of the sequences that account for the free distance are shown with darker branches and labelled with the appropriate channel output signals. The free distance is

$$\begin{aligned} d_{free} &= \sqrt{d^2(0,6) + d^2(0,7) + d^2(0,6)} \\ &= \sqrt{\Delta_1^2 + \Delta_0^2 + \Delta_1^2} \\ &= 2.141 \end{aligned}$$

This pair of sequences is not the only pair that differ, in Euclidean distance, by the free distance. Figure 2-7 shows another pair of sequences that differ by the free distance. For this pair, the Euclidean distance between the sequences is

$$\begin{aligned} d &= \sqrt{d^2(1,7) + d^2(0,7) + d^2(4,2)} \\ &= \sqrt{\Delta_1^2 + \Delta_0^2 + \Delta_1^2} \\ &= 2.141 \\ &= d_{free} \end{aligned}$$

In section 2.1 it was mentioned that the measure of a TCM code is always as a gain relative to the uncoded case. A convenient and often quoted gain is the *Asymptotic Coding Gain*(ACG). This is dependant solely on the free distance of a code relative to the minimum distance, d_{min}, of the uncoded constellation.

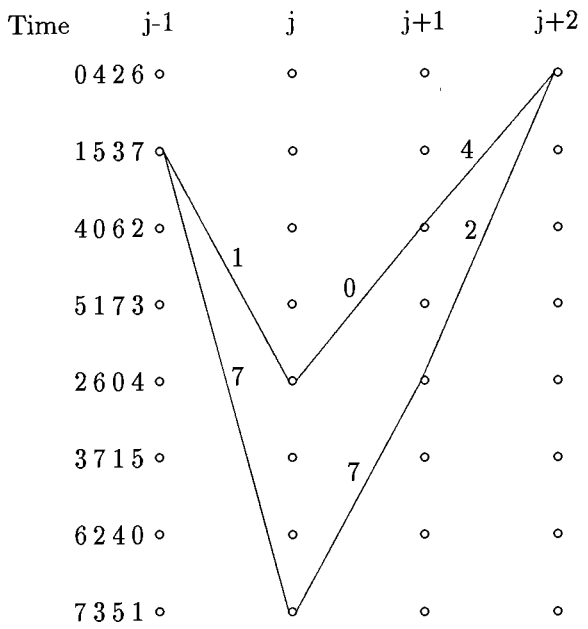

Figure 2-7: Alternative Free Distance

$$ACG = \gamma = \frac{d^2_{free}/E}{d^2_{min}/E'} \tag{2.5}$$

Where E and E' are the average signal energies for the coded and uncoded case respectively. For the code above, the uncoded case is 4PSK for which d_{min} is the same as Δ_1 in the set partitioning diagram. Since MPSK has a constant signal energy for all values of M, E and E' are the same. The Asymptotic Coding gain for the best $\nu = 3$ Ungerboeck code is therefore 3.6dB. For $\nu = 2, 4$ the equivalent gains are 1.608dB and 4.1dB respectively.

Chapter 3

Viterbi Decoding

The Viterbi algorithm for decoding convolutional codes was first proposed in 1967 by Viterbi. It was later found to be a maximum likelihood decoding algorithm for convolutional codes[2].

The Viterbi Algorithm is an essential part of any TCM system. At higher speeds, it is what limits the throughput of a TCM system. One of the aims of this project is to find the effects of varying various parameters in the Viterbi decoder on the performance of a TCM system. This enables designers of hardware decoders to establish trade offs between performance and hardware complexity which corresponds to cost.

The Viterbi algorithm and its various implementations are, though fairly simple, notoriously difficult to explain. The procedure followed in this chapter is to give the method for each step of the algorithm followed by an illustrated example of the step. Some effort was made to provide clear and consistent notation.

3.1 Fundamental Operation

The Viterbi Decoder implemented in this project uses a soft decision Viterbi Algorithm and in both the case of pragmatic and Ungerboeck codes it is implemented for an 8 state code. It is however a cumbersome example to use to explain the operation of the Viterbi Algorithm. For this chapter the Viterbi algorithm will be explained for the simplified (1,2,2) code with trellis diagram as in Figure 3-1. Hard decision decoding is assumed. The principles are easily extended to the 8 state soft decision case.

The trellis in Figure 3-1 extends from time j = 1 to 10. The two output bits $v_j{}^0, v_j{}^1$ for each branch are shown along the appropriate branch. The upper branch leaving any state represents an input, u_j, of 0 and the lower branch an input, u_j, of 1. The states at any time j are $m_j = 0$ to $m_j = 3$ from top to bottom.

In order to follow the working of the Viterbi Algorithm a few definitions are necessary. Assume that the input to a channel is the sequence $\mathbf{v} = (v_0, v_1, ..., v_L)$ and that a noisy version of this sequence $\mathbf{r} = (r_0, r_1, ..., r_L)$ is received. The branch metric from state m_{j-1} to state m_j is

$$d(m_{j-1}, m_j) = HD(r_{j-1}, v_{j-1}) \tag{3.1}$$

where HD denotes the Hamming distance between the received symbol, r_{j-1}, and the symbol v_{j-1}. The symbol v_{j-1} is the symbol corresponding to the state transition from m_{j-1} to m_j. Branch metrics are then added to form partial path metrics:

$$M_j(m_j) = P_{j-1}(m_{j-1}) + d(m_{j-1}, m_j) \tag{3.2}$$

where $P_j(m_j)$ is set to 0 for m = 0, j = 0 and set to ∞ (or a suitably large number)

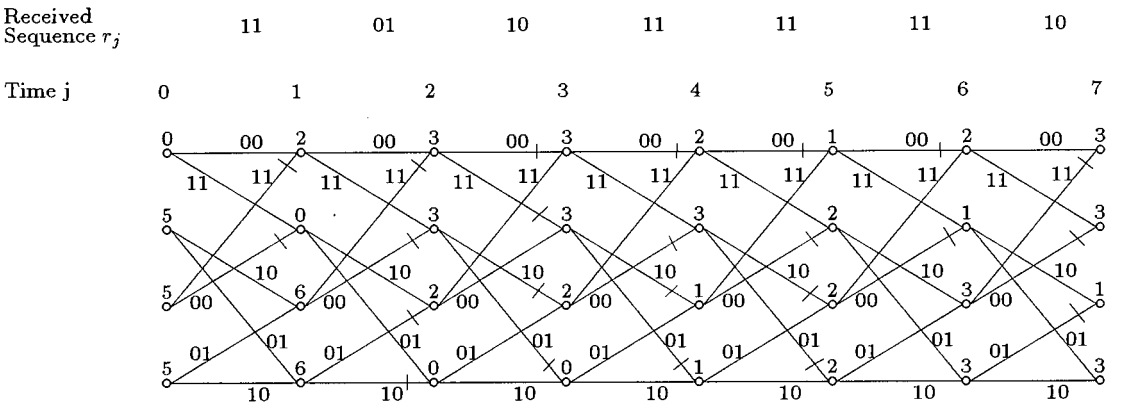

Figure 3-1: Viterbi Algorithm

for $m \neq 0$, j = 0 at the beginning of the algorithm. For $j > 0$, $P_j(m_j)$ is defined by

$$P_j(m_j) = \min_{S_j(m_j)} M_j(m_j) \tag{3.3}$$

The minimum is over the partial path metrics of all the paths, $S_j(m_j)$, entering state m_j. $P_j(m_j)$ is the metric of what is called the *survivor path.*

The following steps constitute the Viterbi algorithm:

1. Initialize survivor path metrics for each state for j = 0:

$$m_j = \begin{cases} 0 & \text{for } m = 0 \\ \infty & \text{for } m \neq 0 \end{cases}$$

   This ensures that all paths originate from state 0 as in the encoder.

2. Increase j by 1. Calculate the branch metrics for all expected channel symbols v_j. Compute the partial path metrics for each branch entering each state by adding the branch metric to the previous survivor metric along that branch. Compare the partial path metrics entering each state and select the smallest as the survivor metric. If there are equal partial metrics entering the same state some random method of choosing between them must be found. Store the survivor paths.

3. If $j < L$ repeat step 2 where L is the length of the received sequence. If $j = L$ choose the state with the lowest survivor metric and trace back along the stored survivor paths to find the decoded sequence, $\{\hat{u}_j\}$.

The above steps are implemented in the example shown in Figure 3-1 for j = 0 to 7. In the example, the input binary sequence, u_j, is the 7 bit pn sequence {1 1 1 0 0 1 0}. The encoder output v_j is therefore {11 01 10 01 11 11 10} assuming the encoder starts in state 0. Now assume that due to noise in the channel, the demodulator passed on the following sequence, r_j = {11 01 10 11 11 11 10 }, to the Viterbi decoder. Note, the only difference is that the 1st bit of the 4th symbol has changed. Above every node, the survivor path metric for that state is shown. To initialize, state m = 0 is set to 0 and all the others are set to 5. This is large enough to prevent any paths from these states surviving past time j = 1. At nodes where $j > 1$, branches with larger partial path metrics have a line through them indicating that the other branch was chosen as the survivor. Where the partial path metrics are the same, no line appears through either branch. At time $j = 7$ the state with the lowest survivor metric is state 2 and the decoded sequence is the same as the input sequence and is shown in dark lines. This shows the error correcting ability of the Viterbi algorithm.

As an example of step 2: entering state 1 at time j=3 the expected symbols, v_j, are 11 and 00 from top to bottom respectively. Their Hamming distances to the received symbol, 10, are both 1. Adding these to the previous survivor metrics along their paths, the partial path metrics for branches 11 and 00 are 4 and 3 respectively. The lower branch with $v_j = 00$ is therefore selected as the survivor path and the survivor path metric is 3. The upper branch has a line through it indicating it was discarded.

The above algorithm is the full implementation of the Viterbi Algorithm. In practice this algorithm is rarely implemented in its full form for two reasons. Firstly, in most applications it is not possible or convenient for the decoder to receive the entire sequence before any bits are decoded. Secondly the storage necessary for storing the survivor paths for each state and at each time interval, is proportional to the length of the

sequence. The decoder therefore has a constraint on the length of sequences it can receive. The solution to both these limitations is provided with a slightly modified version of the Viterbi algorithm called the *Truncated Viterbi algorithm.* Depending on the *Truncation Depth, W,* the truncated Viterbi algorithm can be a near optimal decoding algorithm.

In the truncated Viterbi algorithm only a window of data with length W is stored with a decision being made about the first input symbol, u, after W channel output signals, r_j, have been received. Thereafter one input symbol is decoded for each channel output, r_j received. Steps 1 and 2 are identical for both the full and truncated versions of the Viterbi algorithm. Step 3 for the truncated algorithm states:

If $j < W$ repeat step 2. For $j \geq W$ choose the state with the smallest survivor path metric and trace back along this path to the stored survivor path from time $j - W$ to $j - W + 1$. The input causing this transition is the decoded symbol $\hat{u}_j$. Discard the stored survivor paths between $j - W$ and $j - W + 1$. If $j = L + W$ stop otherwise repeat step 2.

Figure 3-2 shows the same example implemented with the truncated Viterbi algorithm. The truncation depth for this example is 4. It should be noted that the choice of 4 is only for ease of the explanations that follow, in practice for a 1/2 rate code the truncation depth should be about 5ν[3]. At each time unit, j, the survivors are shown for the transition j-1,j and the previous 3 transitions. This represents all the information that needs to be stored in a truncated Viterbi decoder. The survivor path metrics at time j are also shown. At each iteration a new set of survivors is calculated and stored and the survivors furthest back in time are discarded. The branches corresponding to the decoded bit, $\hat{u}_j$, are shown as dark lines. So, for example, at time $j = 6$ the state with the lowest survivor path metric is state 1 with a metric of 1. Tracing back from this state along the survivor paths yields the stored survivor path between state 3 at time $j = 2$ and state 3 at time $j = 3$ shown as a dark line. This corresponds to a decoded bit of $\hat{u}_j = 1$.

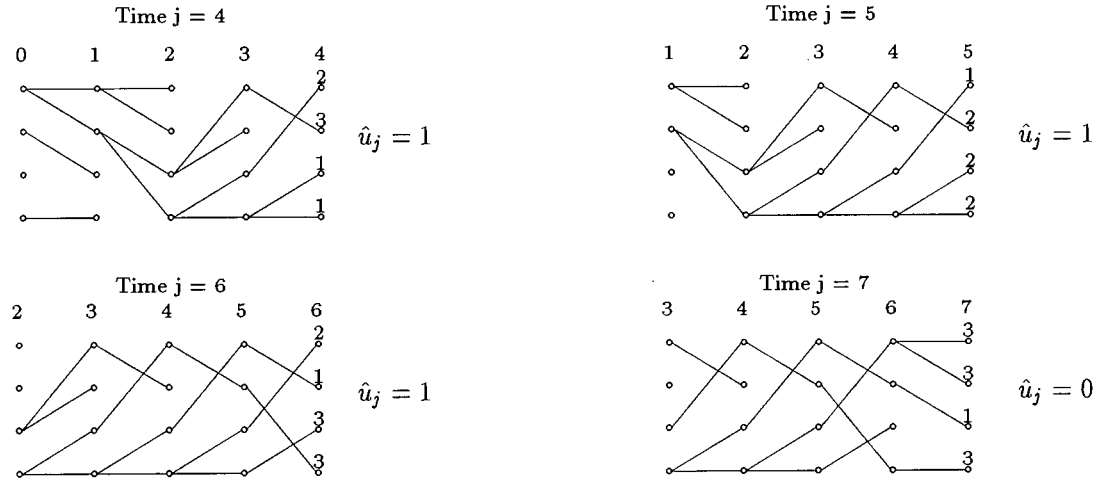

Figure 3-2: Truncated Viterbi Algorithm

One of the shortcomings of the truncated Viterbi algorithm is that there is a possibility that the decoded sequence, $\{\hat{u}_j\}$, is not a valid sequence that can be produced by the encoder. The degree of optimality of the truncated Viterbi algorithm and the trade off between performance and hardware complexity is dealt with in chapter 6 where the results for varying truncation depths are presented.

3.2 Hardware Implementation

Research done in this project looked at hardware efficient implementations of the Viterbi algorithm. Although the final implementation was in software aspects of efficiency and hardware suitable methods were used in coding the algorithm. There are two modules necessary for a hardware implementation, the add compare select(ACS) module and the storage/decision module.

The ACS module is shown in Figure 3-3 and performs step 2 of the algorithm excluding the storage. There are 4 blocks (1 for each state, m_j,) that calculate the survivor path metrics. Each block has as inputs the 2 previous survivor path metrics from the 2 states, m_{j-1}, connected to state m_j and the received symbol, r_j. Each block shown has

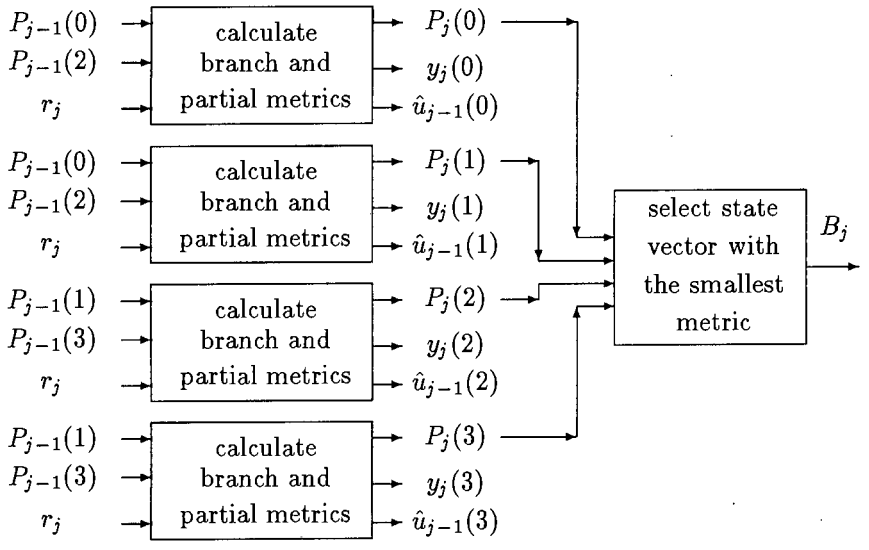

Figure 3-3: ACS unit

3 outputs for the state m_j: the survivor path metric, $P_j(m_j)$, the decoded symbol for the survivor path entering state m_j, denoted by $\hat{u}_{j-1}(m_j)$ and the trace back variable, $y_j(m_j)$. The trace back variable indicates which of the two branches *entering a state* has been chosen as the survivor branch [4] [5]. Note $\hat{u}_{j-1}(m_j)$ is the input bit in the encoder at time unit j-1 that causes the survivor state transition to state m_j. For this example, the trace back variable is defined as

$$y_j(m_j) = \begin{cases} 0 & \text{if upper branch selected as survivor} \\ 1 & \text{if lower branch selected as survivor} \end{cases}$$

The importance of having both $\hat{u}_{j-1}(m_j)$ and $y_j(m_j)$ will become apparent when the register exchange method is explained. The survivor path metrics are then compared and the state, B_j, with the smallest metric is selected.

The outputs of the ACS module are fed to the storage-decision module. This module performs step 3 of the truncated Viterbi algorithm. There are 2 main methods for implementing this module: trace back and register exchange[3], each has its own advantages. In the trace back method the $y_j(m_j)$ are stored and recalled sequentially

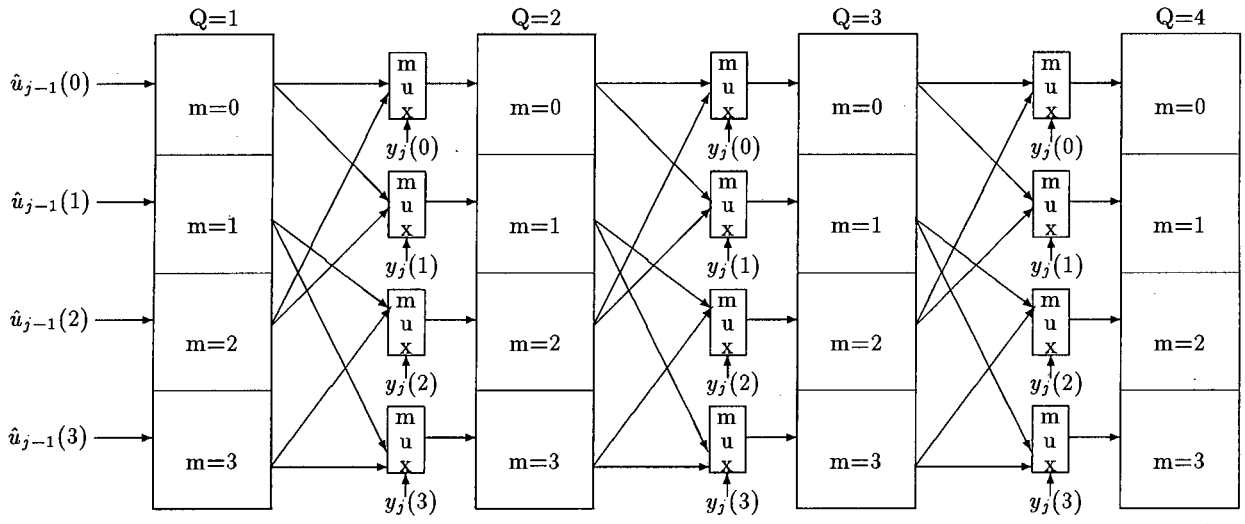

Figure 3-4: Hardware for Register Exchange

once the state, B_j, with the lowest metric has been determined. There are efficient, fully parallel methods of implementing this method which avoid the need for a clock at several times the symbol rate[5]. These methods however require double the amount of storage. In this project the method used is register exchange. For a fully sequential, software implementation as is the case for this project, register exchange is the most efficient.

For the register exchange method, the complete decoded sequence, $\{\hat{u}_j(m_j)\}$, for each state, is stored. This results in a total of W sets of 2^ν registers. For the trellis in this chapter with $W = 4$ this corresponds to 4 sets of 4 registers with each register containing a single bit.

Figure 3-4 shows the hardware required to implement the register exchange method for the example trellis and W=4. Each register is labelled according to which register set, Q, it belongs to and which state, m, it represents. As with the trellis diagram, the registers shown are those corresponding to states 0 to 3 from top to bottom respectively. At every clock edge, registers are exchanged according to the connection of the trellis. Each register, $Q_m \quad Q > 1$, has 2 possible registers, from the set of registers labelled $(Q-1)_m$, from which it can get its input. Which of these 2 possible registers is passed

4.1 Origin of Pragmatic Codes

8PSK Ungerboeck codes for various values of ν are optimum in the sense of having the maximum possible Euclidean d_{free} or equivalently the maximum possible Asymptotic Coding Gain(ACG). One of the difficulties of implementing these codes is that the codes are different for each constellation or order of constellation. Thus encoders and decoders cannot easily be modified to run under different circumstances. These modifications might be, for example, to increase the bandwidth efficiency of a system by changing it from a 2/3 code with 8PSK to a 3/4 code with 16PSK. For Ungerboeck codes, this means redesigning both encoder and more importantly the Viterbi decoder because the codes in each case are far different. In addition Viterbi decoders for high constraint lengths require costly VLSI design.

Viterbi proposed a solution to these problems through pragmatic codes [6]. Pragmatic codes involve adapting convolutional codes, that have been optimized for straight convolutional encoders ie in terms of free hamming distance, for use in a TCM type structure. The Pragmatism refers to 2 facts that make implementing these codes convenient and cost effective. Firstly chips are available that implement encoders/decoders for these codes. Secondly a single code can be used for all constellations. Upgrading to a higher bandwidth efficiency by increasing the constellation size requires only minor changes with the identical Viterbi decoder.

4.2 Formulation of Pragmatic codes

A generalized MPSK pragmatic encoder/modulator is shown in Figure 4-1 for M = 4,8,16. The ability of this encoder/modulator to be modified for different bandwidth efficiencies is immediately apparent. This generalized encoder/modulator first appeared in a paper by Viterbi [6]. In this paper it is proposed that the industry standard time

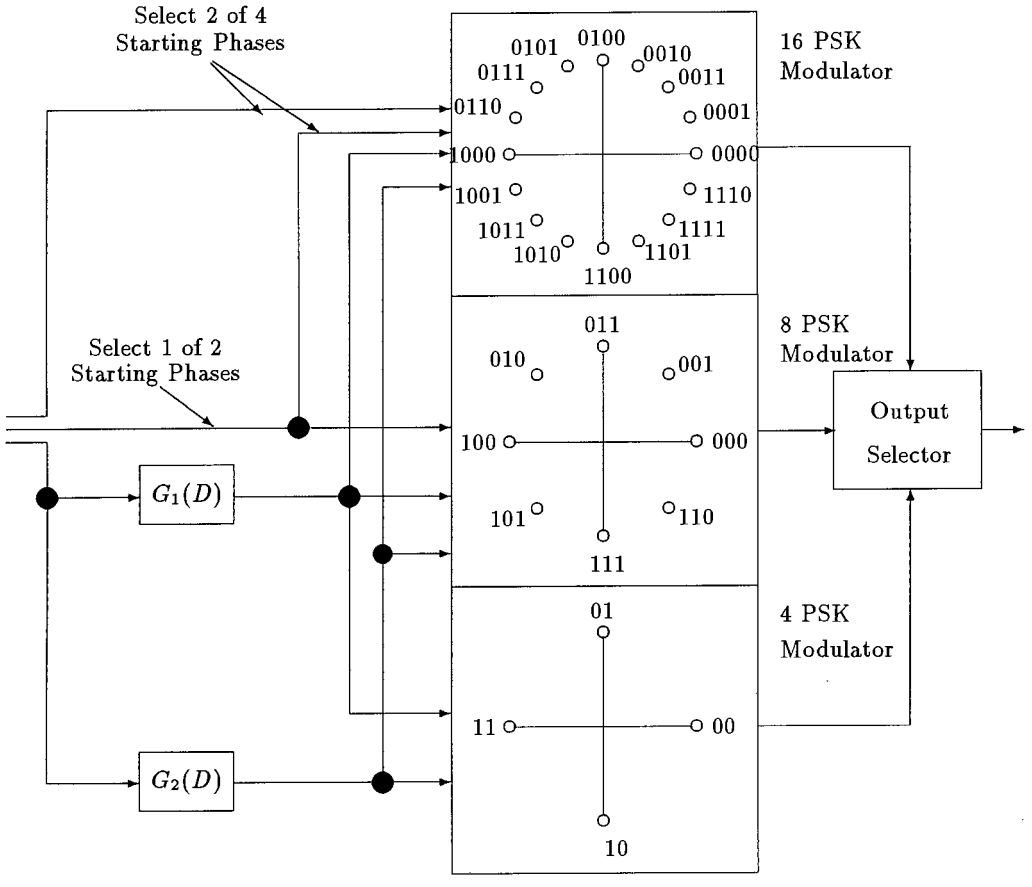

Figure 4-1: Generalized MPSK Pragmatic Encoder/Modulator

constraint length $n_t = 7$ rate 1/2 code with generator polynomials G_1, G_2 is used combined with the mapping as shown in Figure 4-1. It is claimed in this paper that the best $n_t = 7$ rate 1/2 code (with maximum free hamming distance) and hence the code used for the pragmatic code has generator polynomials

$$\begin{aligned} G_1(D) &= 1 + D^2 + D^3 + D^4 + D^5 \\ G_2(D) &= 1 + D + D^2 + D^3 + D^6 \end{aligned} \tag{4.1}$$

This in contradiction to the technical data sheet for the Q1875 Pragmatic Trellis decoder[7] which is manufactured by Qualcomm specifically for pragmatic codes. In this data sheet the best $n_t = 7$ rate 1/2 code is given as that produced by the encoder

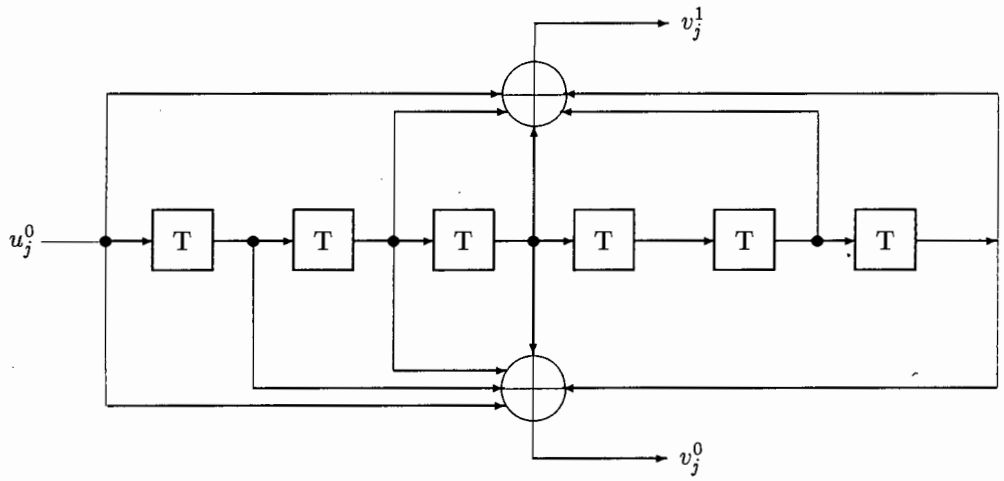

Figure 4-2: Qualcomm $\nu = 6$ 1/2 Encoder

shown in Figure 4-2. This corresponds to a code with impulse responses (in octal) of {171,133} which is equivalent to generator polynomials:

$$\begin{aligned} G_1(D) &= 1 + D^2 + D^3 + D^5 + D^6 \\ G_2(D) &= 1 + D + D^2 + D^3 + D^6 \end{aligned} \tag{4.2}$$

The above generator polynomials are the correct ones for the best $n_t = 7$ rate 1/2 code which is confirmed in[3] where convolutional codes with the largest $dfree$ hamming are tabulated for various values of n_t. These codes were found using exhaustive searches. The free hamming distance for this code is 10. The trellis produced by the encoder/modulator in Figure 4-1 is shown in Figure 4-3 for 8PSK. For the $n_t = 7$ code, X is a 5 bit binary number representing the value of 5 bits of the state register in the encoder. The trellis structure results from the fact that the encoder contains a shift register of length 6. State vectors at time j are simply shifted versions of connected state vectors at time j-1 with the least significant bit replaced by the input at time j-1.

Specific rules are given in [6] for mapping the encoder output bits to channel phase signals a_n. For M = 4,8,16 and $l = \log_2 M$ the encoder/modulator of Figure 4-1

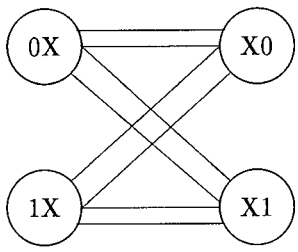

Figure 4-3: General form of Pragmatic Trellis

operates as follows. $l-1$ bits are input in parallel. The least significant bit is input into the convolutional encoder with the remaining $l-2$ bits selecting one of 2^{l-2} starting phases. The convolutional encoder has 2 output bits which are mapped in gray code as offsets from the starting phases as shown below

$$
\begin{aligned}
00 &\rightarrow \quad 0rad \\
01 &\rightarrow \quad \pi/2^{l-1}rad \\
11 &\rightarrow \quad 2\pi/2^{l-1}rad \\
10 &\rightarrow \quad 3\pi/2^{l-1}rad
\end{aligned}
$$

The $l-2$ uncoded bits select 1 of 2^{l-2} starting phases according to:

$$\pi i/(l-2) \quad where \quad i = 0 \ \ to \ \ (2^{l-2}-1)$$

i is the decimal equivalent of the binary number formed by the $l-2$ uncoded bits. This mapping translates to starting phases of $0, \pi$ for 8PSK and $0, \pi/2, \pi, 3\pi/2$ for 16PSK.

The mapping is based on 2 principles. The first ensures signal points on the constellation that have the same coded bits are as far apart as possible in terms of euclidean distance. Looking at the 8PSK constellation in Figure 4-1 all points that have the

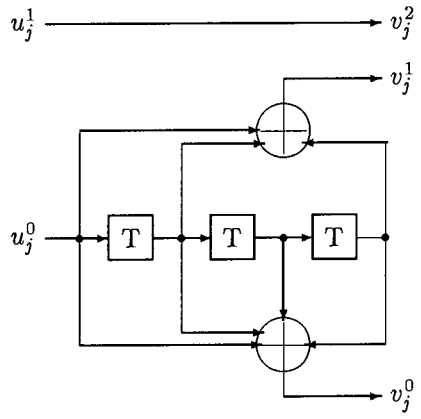

Figure 4-4: Pragmatic $\nu = 3$ Encoder

maximum Euclidean separation of 2 (assuming an average signal energy of 1) have been assigned to binary numbers that only differ in the most significant (uncoded) bit. This corresponds to obtaining the maximum possible Euclidean distance between points assigned to parallel transitions in the trellis. The second more subtle principle exploits the fact that the code used has been optimized for the maximum free hamming distance. This is exploited by ensuring that the 2 encoded bits are mapped so that increasing hamming distance between points maps to increasing euclidean distance. This is achieved by Gray coding the 2 coded bits as phase offsets. Thus the good hamming distance properties of the code are translated to good Euclidean distance properties.

4.3 Equivalent $\nu = 3$ comparison

For the purposes of the comparison in this project $\nu = 3$ codes were used for both pragmatic and Ungerboeck. Implementing the above $n_t = 7(\nu = 6)$ code would mean implementing a Viterbi decoder for a 64 state code. This slows down the simulation to a large extent and makes performance tests too slow.

The modification to $\nu = 3$ involved finding the best 1/2 rate $\nu = 3$ convolutional code. This was found in [3] and [2] to be the code with impulse responses {17,15}. This code

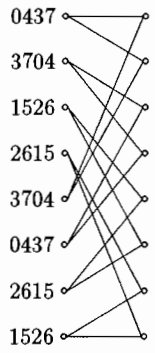

Figure 4-5: Pragmatic $\nu = 3$ Trellis

has a free hamming distance of 6. The above rules were followed for mapping encoder output symbols to 8PSK constellation points. The resulting encoder, including the uncoded bit, is shown in Figure 4-4. The resultant trellis is shown in Figure 4-5 where each branch represents a parallel transition. This trellis follows the same structure as the trellis in Figure 4-3 with X in this case being a 2 bit binary number.

In a simulation similar to that in this project [8], a $\nu = 3$ comparison between Ungerboeck and pragmatic codes is also made. The pragmatic code used in [8] is that with impulse responses {13,15}. This code is a member of a class of codes called complementary codes [9]. Complementary codes allow easy computation of the free hamming distance(d_{free} hamming) of the code. This {13,15} code has been optimized only for maximum d_{free} hamming which is 6. While the code used in this project has the same d_{free} hamming of 6, it has also been optimized for the weight multiplier, w_j, corresponding to the free distance. In general w_j refers to the total number of bit errors that result for all sequences whose output symbols differ in hamming weight by j. Thus w_6 is the weight multiplier corresponding to $d_{free} = 6$. For the pragmatic code used in this project $w_j = 2$ while for the {17,15} code used in [8] $w_j = 3$. The best judge of a good code, particularly at high signal to noise ratios, is one which has the maximum free distance and the minimum w_j corresponding to d_{free}. Thus the code used in this project is a better code than that used in [8] and is more suitable for use as the best rate $1/2$ $\nu = 3$ code which is what is required for pragmatic codes.

4.4 ACG of Pragmatic Codes

The d_{free} Euclidean for pragmatic codes is determined by the Euclidean distance between constellation points associated with parallel transitions. These parallel transitions are what limit the ACG achievable with pragmatic codes. For the M=8 case used in this project parallel transitions consist of 2 branches each assigned to opposite ends of the constellation. ie antipodal signals. For a normalized signal energy of 1 the Euclidean distance between antipodal signals is $\Delta_2 = 2$ as shown in Figure 2-6. Thus with the minimum distance for the uncoded case being $\Delta_1 = 1.414$ the asymptotic coding gain for the M=8 pragmatic code is

$$ACG = 20 \log \frac{2.0}{1.414} = 3.01 dB \tag{4.3}$$

The ACG performance loss compared with the equivalent Ungerboeck code is evident.

Chapter 5

Design of the Simulation

The simulation designed and implemented in this project is a mixture of hardware and real time software. The input sequence generation, encoder, level shifter, noise generation and analog to digital conversion was built in hardware. The Viterbi decoding was implemented in software and written using the C programming language. This hybrid form of simulation has advantages over pure software simulation. It is far quicker, with the limiting factor being the software implementation of the Viterbi algorithm. The hardware encoder designed can also be re-used to test a full hardware implementation of a Viterbi algorithm. The hardware can also be upgraded to be able to test for other effects such as imperfect synchronization or inter symbol interference(ISI) without slowing down the simulation.

The purpose of this chapter is both to detail the design of the simulation and to explain any assumptions made. This chapter puts into context the results of the error rate tests presented in the next chapter by explaining the conditions under which they were obtained.

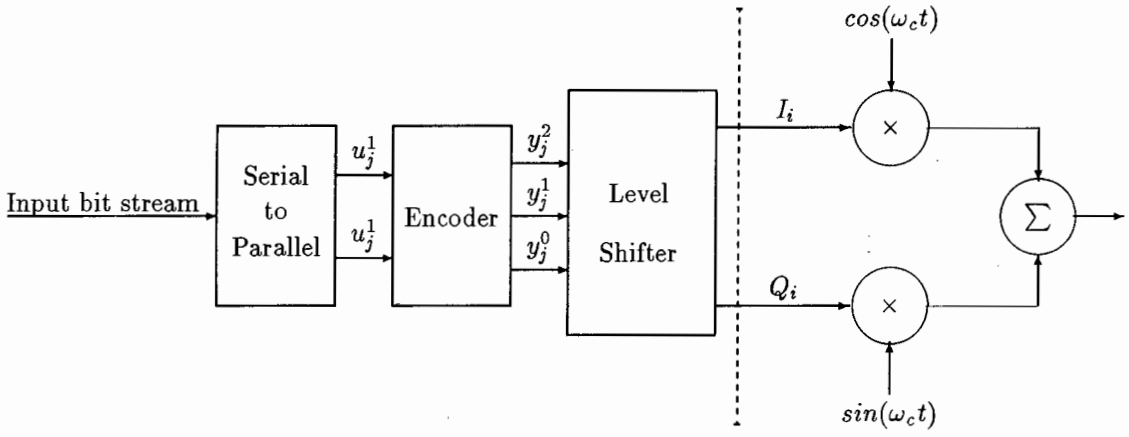

Figure 5-1: TCM Encoder/Modulator

5.1 Degree of Abstraction

Figures 5-1 and 5-2 respectively show a full TCM MPSK Encoder/Modulator and Decoder/Demodulator. For the purposes of the simulations performed in this thesis it was considered unnecessary to build the full TCM system shown in these diagrams. In Ungerboeck's seminal paper on Trellis coded modulation [1], simulations are done assuming interference free signalling over a band limited additive white Gaussian noise channel. Perfect carrier and symbol synchronization is also assumed. The same methods are used in the design of the software simulations for the V32 TCM scheme in [10]. These assumptions were also made in the design of the simulation for this project. In terms of hardware the above assumptions allow the exclusion of all components to the right of the dashed line in Figure 5-1 and all components to the left of the dashed line in Figure 5-2.

The main aim of the tests described in the next chapter is to estimate the coding gain of pragmatic and Ungerboeck codes and the effect on error performance of varying Viterbi decoder parameters. The above assumptions do not detract from achieving these aims. The simplifications made prevent the results being obscured by ISI or synchronization problems. While these problems are important factors for a full TCM system they are not the focuses of this thesis.

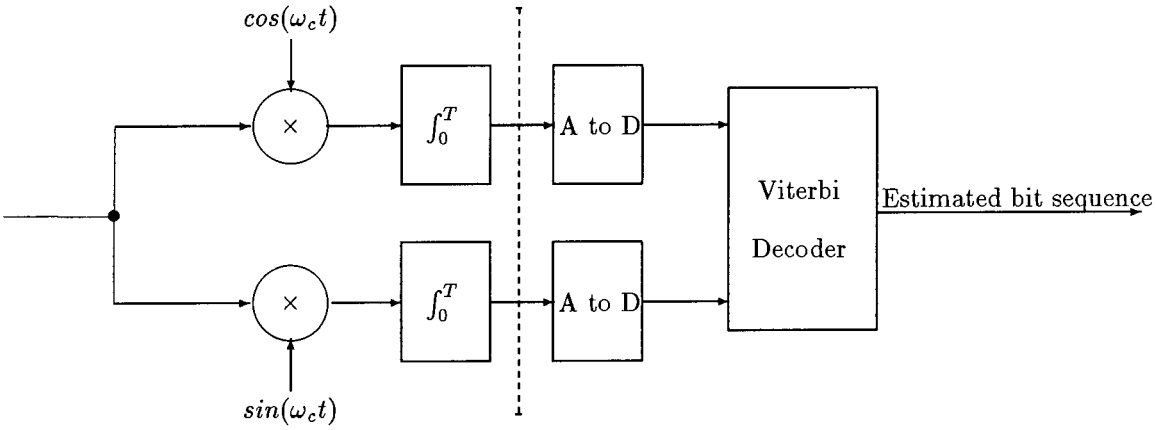

Figure 5-2: TCM Decoder/Demodulator

5.2 Hardware Emulation

The hardware was designed to be as simple and reliable as possible. Components available in the local university laboratory were used. Although the system was designed for a bit rate of 2400bps, the hardware can operate at up to 1Mbps and is thus suitable for testing higher speed decoders.

5.2.1 Level Shifter

This module is dealt with first because it is what sets the mapping of encoder output bits to constellation points. This mapping needs to be dealt with before the encoder is explained.

The level shifter has a 3 bit wide digital input and as outputs the level shifted I and Q channel outputs. The design uses two 12 bit DAC1221 Digital to Analog Converters(DACs) to generate the required I and Q levels. The mapping used is the same as the Ungerboeck mapping shown in Figure 2-6 with the decimal equivalent of the 3 bit binary input shown. The required I and Q signal levels for constellation point i with amplitude A are

$$I_i = A\cos\left(\frac{\pi i}{4} + \frac{\pi}{8}\right) \tag{5.1}$$

$$Q_i = A\sin\left(\frac{\pi i}{4} + \frac{\pi}{8}\right) \tag{5.2}$$

The constellation shown for the generalized pragmatic encoder in Figure 4-1 is phase shifted by $\pi/8$ from the constellation described in the above equations. The constellation in Figure 2-6 is simpler to implement as it only has 4 distinct levels in I and Q whereas the constellation in Figure 4-1 has 5 distinct levels.

The approach was to use as big a range of the DAC as possible to ensure minimum deviation from the ideal levels. The DAC was used in polar mode. In this mode the binary input with decimal equivalent 2048 results in 0V at the output. The closest values under these conditions were 8 1203 2893 and 4088. Comparing these with the exact values the average rms error is 0.03% of the LSB of the DAC. Figure 5-3 shows the constellation with the decimal equivalent of the required binary input into each DAC.

To provide the required 12 bits into each DAC, GALs were programmed as combinational circuits. Both GALs have as inputs the three bit output from the encoder and each GAL provides the correct 12 bit number to the DACs. The level shifter was built on printed circuit board. Figure 5-4 shows the resulting constellation which was obtained from a storage oscilloscope operating in XY mode. The I signal was applied to the X deflection and the Q signal to the Y deflection. [1]

An additional circuit after the level shifter scales and shifts the bipolar output of the level shifter to voltages in the range 0 to 5v which is what is required for the analog to digital converter.

[1] It should be noted that while the X and Y axes represent equal voltages per division, the X axis divisions in the Figure are larger by a factor of approximately 4/3.

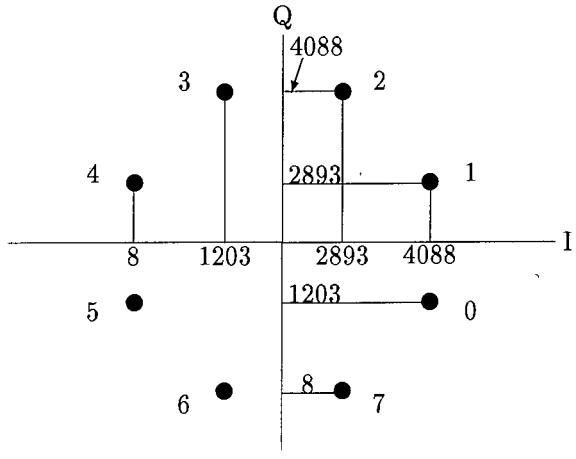

Figure 5-3: I and Q DAC levels

5.2.2 Encoder and Input Sequence Generation

Performing error rate tests requires the generation of input bit sequences. The circuits to generate an input pseudo random bit sequence(PRBS), perform the 2 bit serial to parallel conversion and implement the encoders were built on the same PCB.

PRBSs can be generated using a shift register and performing an exclusive-or between the final bit and a specified bit and feeding this back to the 1st bit. For an m bit shift register the maximum length is $2^m - 1$ as the state with every bit 0 cannot progress. Various maximum length PRBSs are listed in [11]. On the PCB there are 2 PRBSs implemented, a 3 bit and a 5 bit shift register sequence. It was later realized that these would not be long enough for reliable error rate tests. Thus a 23 bit shift register was built. This was built using 3 CD4015 dual 4 bit shift registers with the 18th bit as the feedback tap. The CD4015 can only be initialized to the all 0 state thus to prevent the shift register from getting stuck in the 0 state, an exclusive-nor is performed between the 18th and the final bit. The PRBS can be reset by pressing a pushbutton. The autocorrelation properties of PRBSs [12], [11] make them suitable input for error rate tests. The random properties of the sequence ensure that the first of Ungerboecks mapping rules, that channel signals should be transmitted equally often, is adhered to.

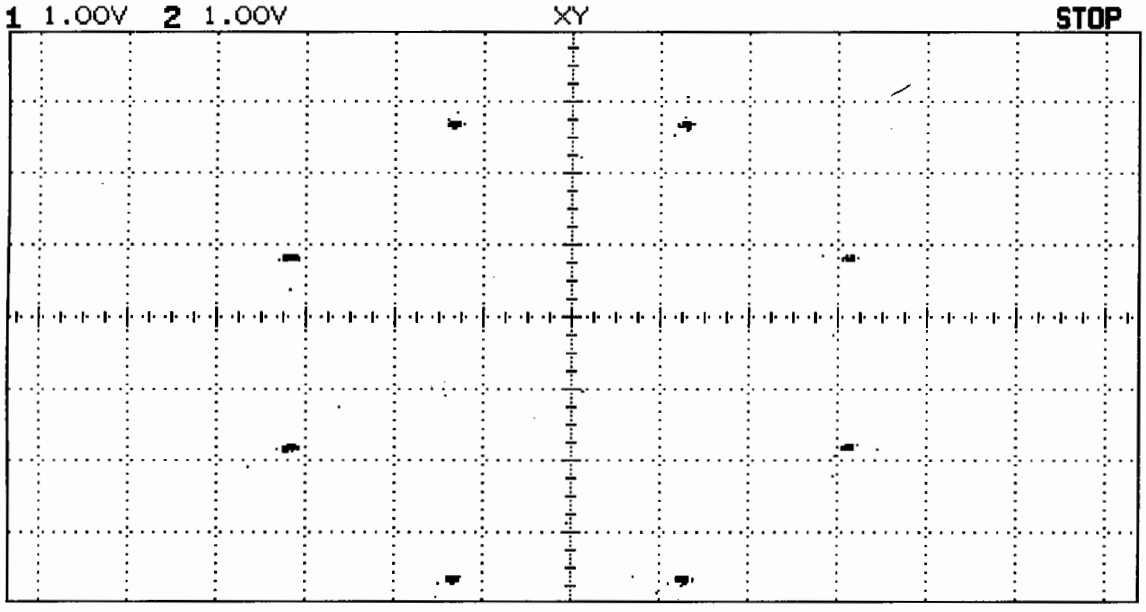

Figure 5-4: Oscilloscope Image of Constellation

The output of the PRBSs are converted into 2 bit parallel data in a serial to parallel converter and the output fed into the encoders. There are 2 different rate 2/3 codes on the PCB; optimal Ungerboeck and pragmatic codes for $\nu = 3$ shown in Figures 2-3 and 4-4 respectively. Codes are selected by changing jumpers on the board which enable tri-state buffers for the appropriate code.

The mapping of encoder output bits to I and Q levels is set by the level shifter. The level shifter has been designed specifically for Ungerboeck codes. The mapping for pragmatic codes is as shown in Figure 4-1 and that for Ungerboeck shown with the appropriate binary numbers in Figure 2-4. The difference between the 2 mappings, ignoring the $\pi/8$ phase offset, is that the 2 least significant bits are mapped differently. For Ungerboeck codes they follow a binary sequence while for pragmatic codes they follow a Gray sequence. To convert to the required mapping for pragmatic codes using the existing level shifter, the Gray to binary conversion is performed as shown in Figure 5-5.

Tests were also performed on uncoded 4PSK. In order to generate 4PSK, the 2 output bits from the serial to the parallel converter are input directly to the 2 most significant

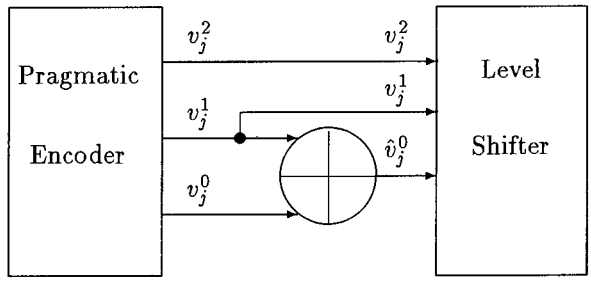

Figure 5-5: Binary to Gray conversion for Pragmatic codes

inputs of the level shifter. The least significant bit of the level shifter is kept at +5v.

5.2.3 A to D card

An A to D card was built for sampling the I and Q channels. It was built as a PCB plug in card for an IBM PC. Although A to D cards for a PC are commercially available the cheapest solution for achieving high speed accurate A to D conversion was to build the card. The ADCs used were the 8 bit ADC0820's, one for each of the I and Q channels. These chips were used because of their fast conversion time ($2.5\mu S$) and features which allow them to interface easily with a microprocessor. The inputs to the card were the I and Q channel, +5v power and ground from the external power supply and the clock from the PRBS generator. The clock ensured that samples were taken in the middle of each symbol interval. The external power supply was used as the reference for the ADC's as it is far more stable than the noisy PC power supply.

The card also has a 2 bit digital output line. This was originally to be used to output the decoded bits, but outputting the decoded bits was later found to be unnecessary. Two different modes of conversion were tested with the original board having to be modified to allow the ADCs to operate in the more reliable read mode. The procedure for an A to D conversion is for the clock to interrupt the processor. The interrupt service routine(ISR) then starts the conversion of the I channel ADC using one of the digital output lines. The result of the conversion is then read by the ISR and the same

is done for the Q channel.

Address decoding was designed to allow for an adjustable base address with the 3 ports (2 ADC ports and 1 output port) being at offsets from the base address.

5.2.4 Noise Generation

Noise in a full TCM system is added to the modulated signal. This is equivalent, for the unmodulated system used in this project, to adding *uncorrelated* noise in both I and Q channels. ie. the noise added to the I channel must be uncorrelated with the noise added to the Q channel. Considering the constellation diagram the reason why the 2 noise sources need to be uncorrelated is intuitively evident. In a full modulated TCM scheme AWGN would result in the constellation points blurring with circular symmetry around each point. Correlated noise sources added to I and Q in an unmodulated scheme, would result in straight lines at 45 degrees from the I axis with the centre of the line at each constellation point.

The noise is generated by filtering a polar PRBS. The power spectral density of a PRBS is flat up to 1% of the frequency at which it is clocked. Filtering a PRBS at 1% of the clock frequency therefore results in band limited white Gaussian noise.

For this project 2 identical 31 bit shift registers are used. The PRBS in each is converted to a polar signal and the resulting signal is filtered using a simple RC low pass filter. Each PRBS is clocked by a separate crystal generated clock at 1Mhz. The signals are filtered using a simple RC low pass filter with $f_{3dB} = 8842Hz$ resulting in an equivalent noise bandwidth, B, of $\pi/2 f_{3dB} = 13888Hz$. The output of the filter is then fed into an amplifier the gain of which is adjustable to allow the noise source to be set to different power levels. Thus for error rate tests, the signal level is kept constant and only the noise power is varied.

Samples of the signal plus noise spaced apart by $1/f_s$, where f_s is the symbol frequency, are taken at the A to Ds. Ideally the noise component of any sample should be uncorrelated with any of the previous samples. The noise samples should be independent samples of additive white Gaussian noise. The noise is however bandlimited to BHz and this introduces correlation or dependence between noise samples. The extent of this correlation can be estimated by computing the autocorrelation function for bandlimited white Gaussian noise. The ideal power spectral density is shown in Figure 5-6. The resulting autocorrelation function, R_{NN}, is

$$R_{NN}(\tau) = N \cdot \frac{\sin(2\pi B\tau)}{2\pi B\tau} \tag{5.3}$$

with N being the noise power[13]. The autocorrelation function is plotted in Figure 5-7. The graph has been normalized for a bandwidth B of 1Hz and noise power of 1W. Setting $f_s = 1/\tau$ to indicate the correlation of samples spaced $1/f_s$ apart, the x axis represents the ratio of the bandwidth B to the sampling frequency, f_s. In order to ensure independent noise samples, the ratio of f_s to B must be decided on carefully. Independent samples can be obtained by sampling at frequencies where the autocorrelation function has nulls, this occurs at ratios of $2B/f_s = k$ where k is any integer. A better method is to make the ratio $2B/f_s$ large enough so that the noise samples are sufficiently independent.

For the noise filters used in this project, the equivalent noise bandwidth was set at B = 13888Hz. The symbol frequency is 1200Hz. For this ratio, the autocorrelation function yields $N \cdot 0.0061$. This ratio is marked in Figure 5-7 with the vertical dashed line. The fact that the autocorrelation function at this point is such a small fraction of the total noise power indicates that the noise samples are sufficiently independent. As a verification a number of the error rate tests performed at f_s = 1200Hz were repeated at $f_s/2$. This made little difference to the results indicating that f_s= 1200Hz is low enough to ensure sufficiently independent samples.

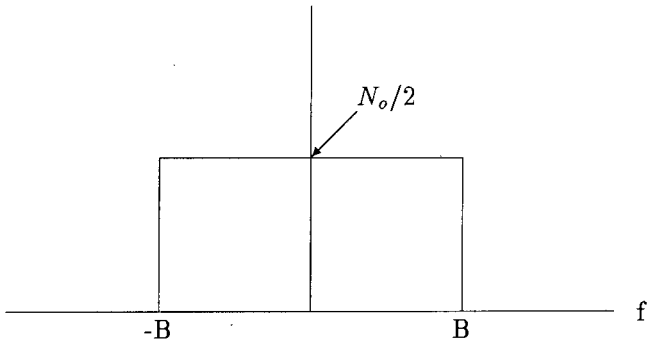

Figure 5-6: PSD of Ideal Band Limited White Noise

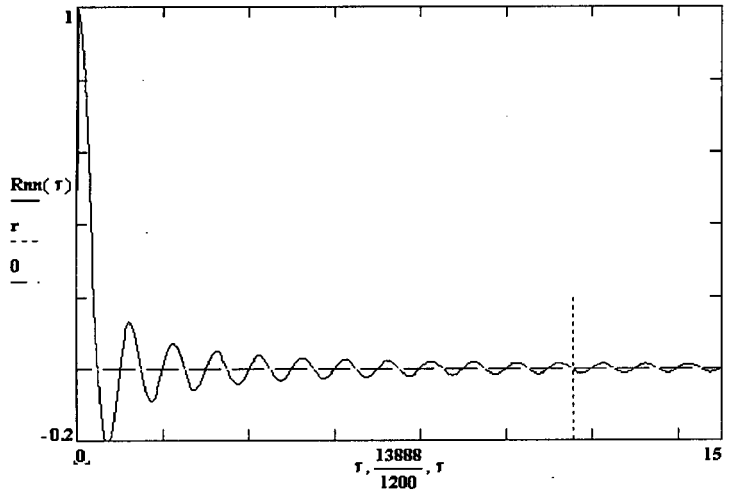

Figure 5-7: Autocorrelation Function for Band Limited Noise

The above argument can also be applied to the requirement that the I and Q noise sources need to be uncorrelated with each other. If the PRBS for each noise source is manually reset at separate times with the time difference far exceeding $1/f_s$ the resulting correlation will be far less than that calculated above and shown in Figure 5-7.

Figure 5-8 shows the I channel at $f_s = 1200Hz$ with 0.87Vrms noise added. This is equivalent to an E_s/N_o ratio of 9.0dB. Figure 5-9 shows the constellation diagram with 0.77Vrms noise added in each channel which is equivalent to an E_s/N_o of 10.0dB. The method for calculating E_s/N_o , where E_s is the average symbol energy and N_o is the noise power spectral density, is explained in section 6.1.

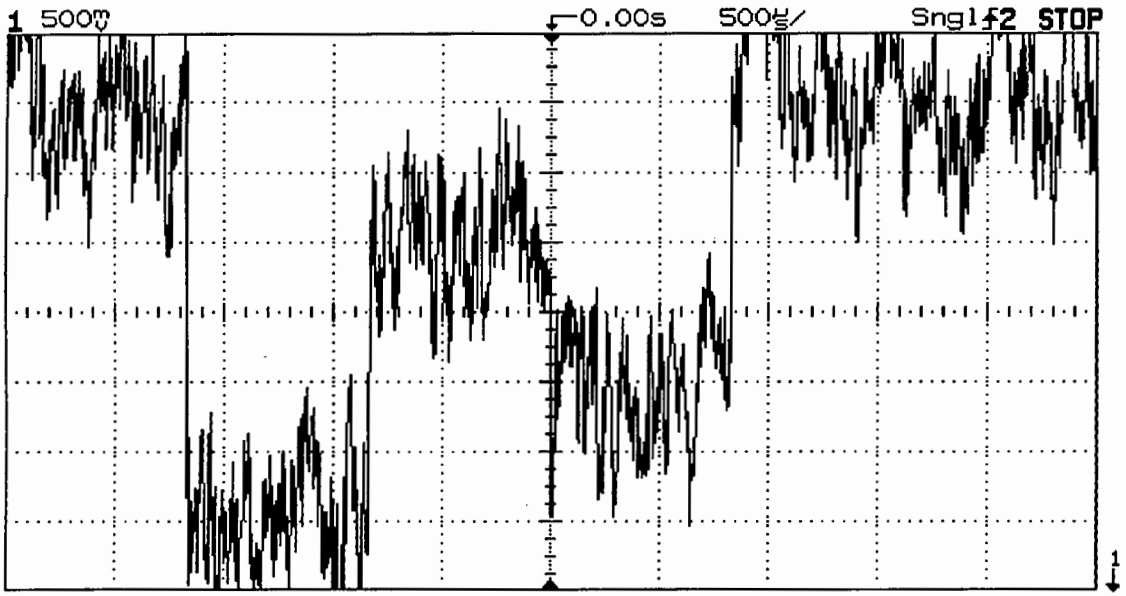

Figure 5-8: I channel with $E_s/N_o = 9.0dB$

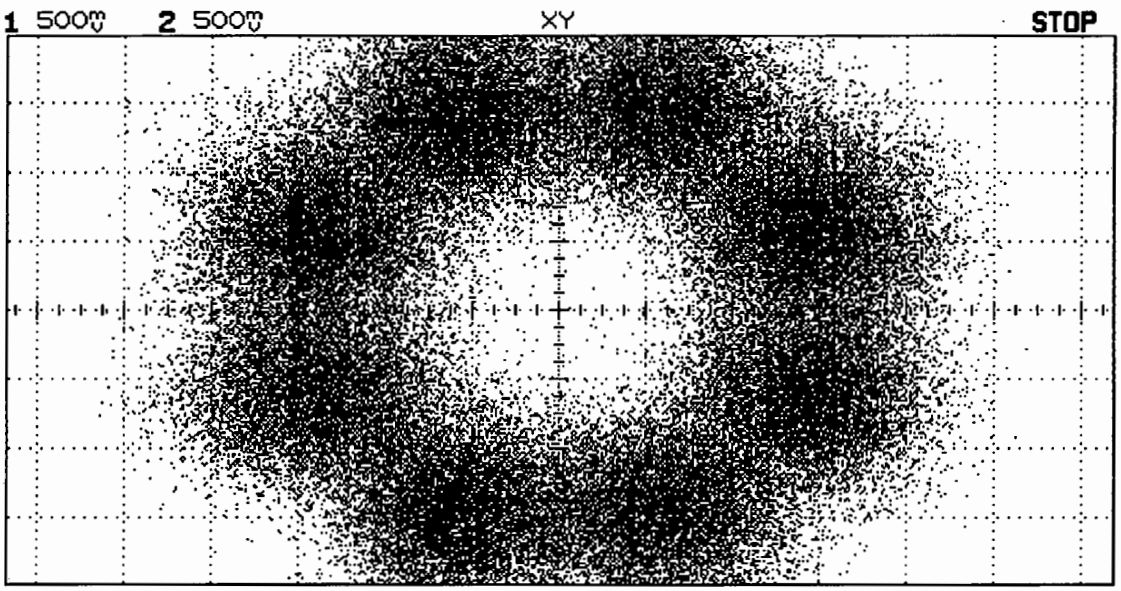

Figure 5-9: Constellation with $E_s/N_o = 10.0dB$

5.3 Software Decoding and Logging

Software written for this simulation centred around the implementation of a real time Viterbi algorithm. Other peripheral functions were needed to aid with error counting, processing of the results, determining average values for the constellation points and interrupt initializing and servicing. The programme will be explained by first detailing the implementation of the Viterbi algorithm followed by explanations of the purpose of the more important peripheral functions. Finally an overall flow chart of the programme will be presented and explained. This follows the way in which the programme was designed. While it was desirable to get the programme to run reasonably quickly, speed was not a major requirement. Where possible code was made as efficient as possible without resorting to lower level programming.

5.3.1 Viterbi Algorithm Implementation

The implementation consists of 3 functions: the function to perform the Branch metric calculation, the add compare select function and the register exchange function.

Branch metric Calculation is performed in a function which has as inputs the sampled I and Q values and the average I and Q values for each of the 8 constellation points. The average values are calculated in a separate function which will be explained later. The Euclidean distance, ED, between 2 points $\{I_1, Q_1\}$ and $\{I_2, Q_2\}$ on a constellation diagram is defined as

$$ED = \sqrt{(I_1 - I_2)^2 + (Q_1 - Q_2)^2} \tag{5.4}$$

Calculating squares and square roots is time consuming in a real time programme thus absolute rather than square values were computed. Thus the ED is calculated as

$$ED = |I_1 - I_2| + |Q_1 - Q_2| \tag{5.5}$$

This is valid for the comparisons required in the algorithm.

The branch metrics are passed on to the function performing the **Add Compare Select**. In this function branch metrics are added on to the appropriate survivor path metrics according to the trellis connections. Comparisons of path metrics are then performed and the $y_j(m_j)$ variables and corresponding survivor path metrics for each state are chosen. In the example in section 3.2, the $y_j(m_j)$ variables are single bit variables. For the rate 2/3 codes implemented in this decoder these variables are 2 bit variables indicating which of the 4 branches entering each state is the survivor path. The survivor path metrics are stored as *long int* variables in the C programme. The long int variable is a 32 bit variable and it was found that for all the tests implemented, these variables did not overflow. Thus, as opposed to most practical implementations of the Viterbi algorithm, metric normalization is not necessary for this implementation.

In section 3.2 an example was given where the single bit $\hat{u}_{j-1}$ variable, for each state at the output of the ACS unit, did not change in time. The same is true for the 2 bit $\hat{u}_{j-1}$ value in Ungerboeck codes. This can be seen from the Ungerboeck 8 state 8PSK trellis in Figure 5-10. Looking at state 0 at time j for example, all the branches entering the state are those resulting from an input, u_{j-1}, of 0. Likewise for any state m_j the value of the input u_{j-1} resulting in any transition to that state is (m_j mod 4) where mod indicates the remainder when m_j is divided by 4. This is illustrated in the Figure: to the left of each state at time j-1 the value of the state m_{j-1} is given. To the right of each state at time j the value of u_{j-1} that results in the all the transitions to that state, is given.

Thus knowing the $y_j(m_j)$ value for state m_j provides no extra information about the decoded value $\hat{u}_{j-1}$ entering that state. However knowing the $y_j(m_j)$ value for state m_j uniquely defines a state m_{j-1} which in turn defines a decoded symbol $\hat{u}_{j-2}$. In

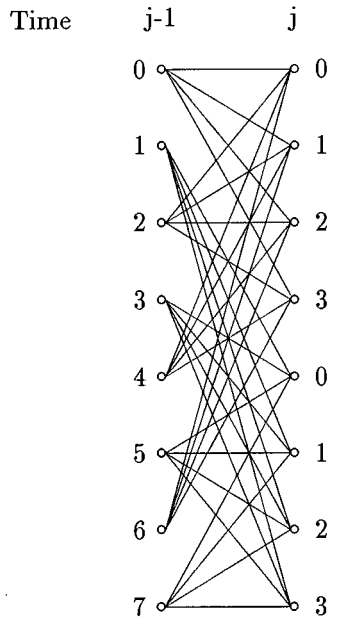

Figure 5-10: Ungerboeck 8PSK trellis showing u_{j-1} variables.

the ACS function in the programme for Ungerboeck codes, the $y_j(m_j)$ values are first decided upon for each state m_j and from them the $\hat{u}_{j-2}$ values are worked out. Using this feature of the trellis amounts to increasing the truncation depth by 1 for the same amount of shift registers. The principle is the same as that developed for the trace back implementation developed in [4].

The above argument does not hold for pragmatic codes or any code where the trellis diagram contains parallel transitions. Parallel transitions imply that there are branches entering a state at time j that result from input values u_{j-1} which are not the same.

The **Register Exchange** function then performs the required shifting of registers according to the $y_j(m_j)$ and $\hat{u}_{j-2}$ results from the ACS. Registers are exchanged as explained in section 3.2. If W or more symbols have been received, the decoded symbol is worked out.

Except for the branch metric calculation, the above functions needed different implementations for Ungerboeck and pragmatic codes. The code for each is written in each function and the choice between them is made by a command line variable passed to

the programme.

5.3.2 Averaging Constellation Points

In order to compensate for drift or offsets caused by the addition of noise, average values are calculated for each I and Q level. Before decoding begins, the jumpers are configured to have the 3 bit shift register PRBS as the input sequence. The I and Q values are sampled and averaged over suitably long periods to obtain average I and Q levels for each constellation point. These average values are then used to calculate branch metrics for each of the 8 constellation points.

5.3.3 Checking for Error

In order to perform error rate tests, the decoded symbol $\hat{u_j}$ needs to be compared with the original input symbol u_j. This is done in the software by having the same 23 bit shift register PRBS running in the programme. Only symbol error rather than bit error is counted as this is a more relevant gauge. For every symbol error counted, the position of the symbol error is also recorded to enable error event counting which will be explained in the next chapter.

5.3.4 PRBS Synchronization

As there are no digital input lines, there is no way of signalling the programme precisely when the input PRBS begins. Thus a method was devised to synchronize the PRBS running in the software with that in the external circuitry. To begin an error rate test, the user needs to hold the input PRBS in reset mode by holding the pushbutton

down. Decoding is then begun by pressing a button on the computer keyboard while still keeping the PRBS pushbutton pressed. The pushbutton is then released shortly afterwards and the encoding begins. The result of this is that the PRBS running in the software always starts off ahead of the hardware PRBS. 30 symbols at a time are checked and if more than 10 errors are found, the 2 PRBSs are judged to be unsychronized and the software PRBS is shifted back by one. This is done until the 2 PRBSs are synchronized and the symbol counter and error counter are reset.

As the maximum error rate tested for was 10^{-2}, 10 errors in 30 symbols is strong indication that the PRBSs are unsychronized. While rather crude this method was found to be extremely effective.

5.3.5 Overall Operation

The overall operation of the software is shown in flow chart form in Figure 5-11. The programme first obtains the average values for the constellation points. After being prompted by the user, the deciding loop begins. If 30 symbols have been decoded , they are checked for synchronization using the criterion explained above. If they are not synchronized the software PRBS is shifted, symbol and error counters are reset and the Viterbi algorithm is performed. If the PRBSs are synchronized, the shifting is skipped and the programme moves to Viterbi algorithm. The outputs of a truncated Viterbi algorithm are not valid until W symbols have been received where W is the truncation depth. Thus only after W symbols have been received is the decoded bit compared with the expected PRBS symbol. At this stage the programme waits to be interrupted by the external clock. A loop variable is incremented while waiting to gauge how much time is left between the end of required processing and the interrupt. the Interrupt service routine then samples the A to D and the process is repeated with the new sampled I and Q values. After the last symbol has been decoded, the results are displayed and processed.

5.3.6 4PSK Decoding

A separate programme was written for performing error rate tests on uncoded 4PSK. The functions for checking errors, synchronizing the PRBSs and calculating average signal points are reused in this programme. The function for calculating average signal points was modified to calculate average values for the signal angles and from them decision boundaries for each point. The decoding is done by performing the equivalent of an arctangent on the sampled I and Q channels to calculate the angle of the received signal. The angle is then compared to the decision boundaries to obtain the decoded symbol.

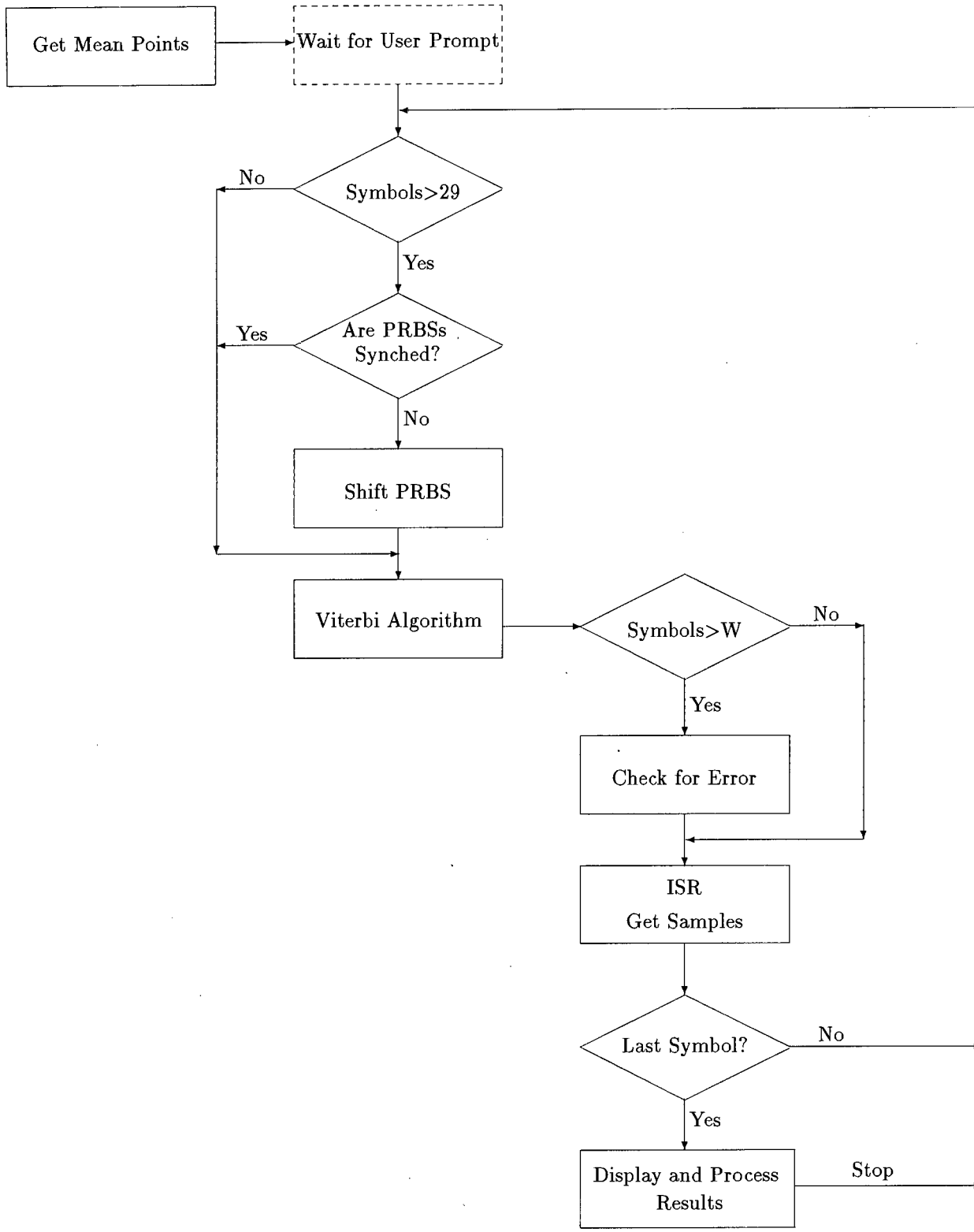

Figure 5-11: Flow Chart of Software

Chapter 6

Simulation Results and Interpretation

This chapter presents the results of the many tests performed on the system. Most of the results are presented in graphical form and the relevant interpretations are made. In order to interpret some of the results it is necessary to explain some aspects of the theory and implementation details. Although some of these explanations might be more appropriate in previous chapters, it was thought easier to explain them in this chapter together with the relevant results.

6.1 E_s/N_o Calculation

The 2 quantities measured directly in determining signal to noise ratio were the average signal power, C, and the average noise power, N. The average signal power was measured at the output of the level shifting DACs and calculated using

$$C = E\{|a_j^2|\} = \frac{1}{M} \cdot \sum_{i=0}^{M} (I_i^2 + Q_i^2) \tag{6.1}$$

$E\{|a_j^2|\}$ is the expected value or mean of the channel signal a_j. I_i and Q_i are the respective I and Q components of the ith constellation phase point. M is 4 or 8 for 4PSK or 8PSK. The average noise power was measured using an rms voltmeter. The noise power, equivalent to the noise variance, is V_{rms}^2 or σ^2.

To convert from the power ratio C/N to the required ratio of E_s/N_o the following conversion is needed[14].

$$\frac{E_s}{N_o} = \frac{C}{N} \cdot \frac{B}{f_s} \tag{6.2}$$

E_s is the average symbol energy, f_s is the symbol rate and B is the equivalent noise bandwidth. In the simulation designed for this project the noise is bandlimited to BHz. Assume that the symbol rate is $2B$ and that the I and Q signals are fed into ideal single sided Nyquist low pass filters (ie ideal "brick wall " filters with cut off frequency of $2f_s$.) This assumption is valid for this simulation because the received I and Q samples are samples from unfiltered signals and are thus equivalent to those sampled, at the correct instant, from an ideal Nyquist filter. The conversion from C/N to E_s/N_o then becomes

$$\frac{E_s}{N_o} = \frac{C}{N} \cdot \frac{B}{2B} = \frac{C}{2N} \tag{6.3}$$

In the project simulation the equation above can be extended to any symbol rate, f_s, as the values of C and N are independent of the symbol rate. This is true provided the symbol rate is low enough to ensure uncorrelated noise samples as explained in section 5.2.4.

In the original simulations reported by Ungerboeck[1] the signal to noise ratio (referred to as SNR in the paper) is defined as

$$SNR = \frac{E\{|a_j^2|\}}{2\sigma^2} \tag{6.4}$$

This is entirely equivalent to equation 6.3. In a later paper reporting the same simulations[15], the same ratio is more conventionally referred to as E_s/N_o. For direct comparison with Ungerboeck's results E_s/N_o will be used. Other results [8], [16], [6] use the more widely used E_b/N_o where E_b is the average energy per bit. Conversion to E_b/N_o is done using $E_b = E_s/2$ for 4PSK and coded 8PSK.

6.2 Uncoded 4PSK

In measuring coding gains relative to the uncoded case, gains could be measured relative to the predicted theoretical performance of the uncoded case. This is making the assumption that the simulation performs exactly as is theoretically predicted. In order to prevent the need for this assumption, tests were done on uncoded 4PSK. The resulting gains for coded 8PSK are shown relative to the measured, uncoded case.

In order to show the simulation is performing close to what is predicted by the theory, the uncoded case was compared directly with the theoretical prediction.

The probability of symbol error for 4PSK in the presence of AWGN [17] is given by

$$Pe = erfc\left(\sqrt{\frac{E_s}{2N_o}}\right) - \frac{1}{4}erfc^2\left(\sqrt{\frac{E_s}{2N_o}}\right) \tag{6.5}$$

For the E_s/N_o values over which measurements were taken in this project, the second term in equation 6.5 was found to be negligible and was thus ignored.

The C code for the decoding of 4PSK has no Viterbi algorithm and is thus is far simpler and quicker to run. The 4PSK tests were thus originally performed at a far higher symbol rate of $f_s = 4000$Hz. The results of these tests are plotted along with the theoretical prediction from equation 6.5, in Figure 6-1.

It was later realized that a symbol frequency of 4000Hz might result in noise samples that have a high degree of correlation for the reasons explained in section 5.2.4. The tests were thus repeated at $f_s = 1200$Hz which is the same frequency as tests performed for the 8PSK coded schemes. The results of these tests are shown in Figure 6-2. It can be seen that the simulation run at $f_s = 1200$Hz agrees far more closely with the theory than that for $f_s = 4000$Hz indicating the effect of correlation of noise samples.

Although the agreement between the theory and the measured results are extremely close, both of the measured graphs shown indicate a slightly lower error rate than that predicted by theory. This apparent anomaly was not considered a major flaw. The deviations from theory are small and the main thrust of the thesis is to measure relative performance ie. coded vs. uncoded and results for varying decoder parameters. Provided all measurements are performed under the same conditions these relative measurements are valid. Possible reasons for this anomaly are the fact that the noise sources have finite crest factors and the fact that at $f_s = 1200$Hz there is some residual correlation between noise samples.

6.3 Error Events

As expected for a Viterbi decoder, errors observed in the simulation appeared in clusters. An important quantity for comparing TCM codes is the error event probability

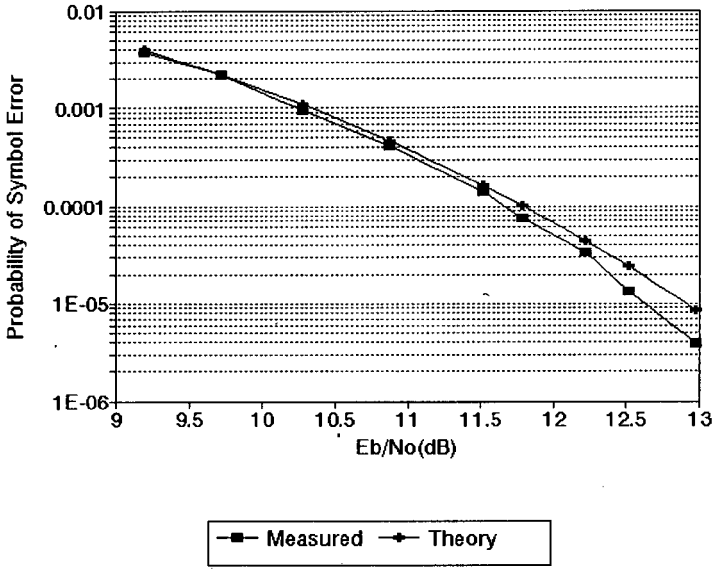

Figure 6-1: Uncoded 4PSK P(e) vs Es/No - Theory vs. measured for $f_s = 4000$Hz

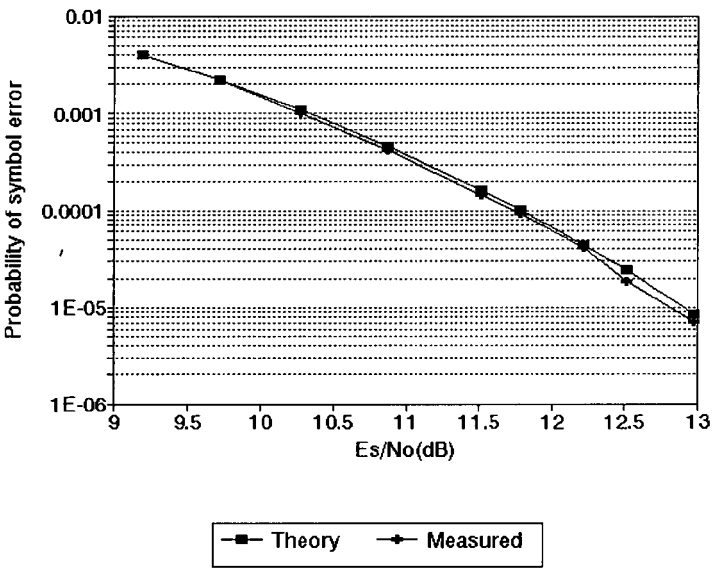

Figure 6-2: Uncoded 4PSK P(e) vs Es/No - Theory vs Measured for $f_s = 1200$Hz

rather than the probability of symbol or bit error. An error event occurs when the Viterbi decoder chooses an incorrect sequence following a correct choice of state. This is best illustrated in the trellis of Figure 2-5 showing the paths that result in d_{free}. An error event would be counted if the correct path is the all 0 path and at time j+2 the decoder chooses the lower path. This amounts to the decoder choosing the incorrect survivor.

In the software implemented in this simulation, there is a symbol counter that counts the number of symbols that have been received. When an error occurs the symbol position, as measured by the symbol counter, is stored. After the last symbol is decoded, processing is performed on the array of error positions and estimates are made of the number and length of error events. The method of counting event lengths will be discussed in section 6.5.

For counting the number of events, symbol errors are considered part of the same error event if their symbol position is close enough to the previous error. It is possible for incorrect decoded branches to yield correct decoded symbols resulting in no error being counted. There is a limit to how many times different branches can have the same decoded symbols without merging. If in the Ungerboeck encoder the same input symbol is received twice in a row, the encoder will always end up in the same state irrespective of which state it started in. For the Pragmatic code used the same is true of 3 symbols received in a row. In fact for Pragmatic codes for 3 of either 00 or 01 input symbol (ie the upper branch) the encoder will always end up in state 0. For 3 of either 10 or 11 input symbols (ie the lower branch) the encoder will always end up in state 7. Thus symbol errors closer than 3 symbol positions apart can be considered to be part of the same error event, errors greater than 3 symbols apart the errors must be from different events.

Pe	Coding Gain
1E-3	2.0dB
1E-4	2.4dB
1E-5	2.6dB

Table 6.1: Measured Ungerboeck Coding gains with W = 20

6.4 Pragmatic and Ungerboeck coding Gains

When optimizing codes, Ungerboeck in each case attempted to find the code with the maximum d_{free}. In terms of performance this will result in the best error event performance rather than the best symbol or bit error performance. Making the d_{free} as large as possible ensures that the most likely error is as unlikely as possible. However once that error has occurred, for Ungerboeck codes, it results in more than 1 symbol error. Looking at the $dfree$ of the Ungerboeck code used, the event length is 3, for the pragmatic code it is 1. Error event lengths will be dealt with in detail in the next section. The following comparison is made on the basis of measured error event performance with error events being counted as described above.

Figure 6-3 shows the error event probability, Pe, vs E_s/N_o for the 8PSK Ungerboeck code with a truncation depth of 20. Plotted also are the error performance of the measured uncoded 4PSK and the 3.6dB ACG for the Ungerboeck code. The coding gain of the measured result can be estimated from the graph. The appropriate signal to noise ratios for the estimate of the coding gain at $Pe = 10^{-5}$ are shown. Coding gains for 3 different values of Pe are shown in table 6.1. It can be seen that the measured performance moves closer to the ACG for increasing signal to noise ratio as expected.

Figure 6-4 shows the measured performance of pragmatic and Ungerboeck codes. It can be seen that for all values of signal to noise ratio the Ungerboeck code performed better than the pragmatic code in terms of error event probability.

The difference between Ungerboeck and pragmatic code performance under these circumstances is very slight. The fact that a truncation depth of 20 is used tends to flatter

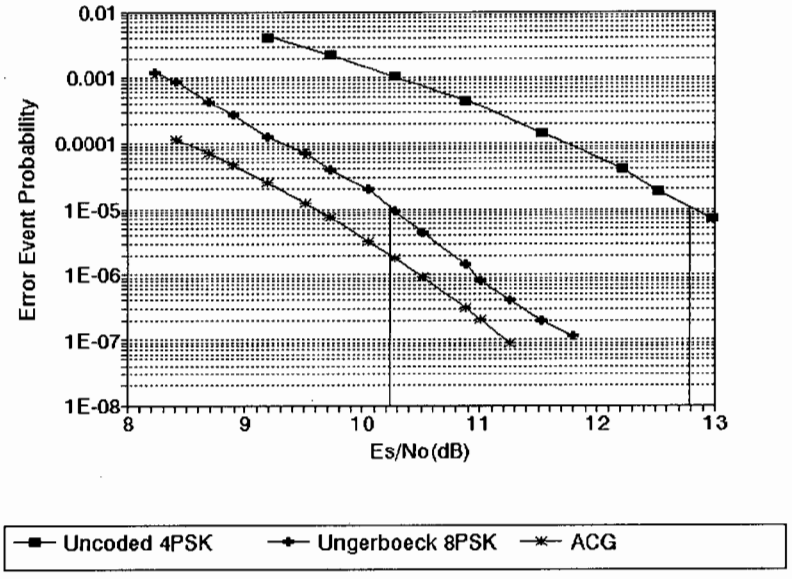

Figure 6-3: Ungerboeck Coded 8PSK

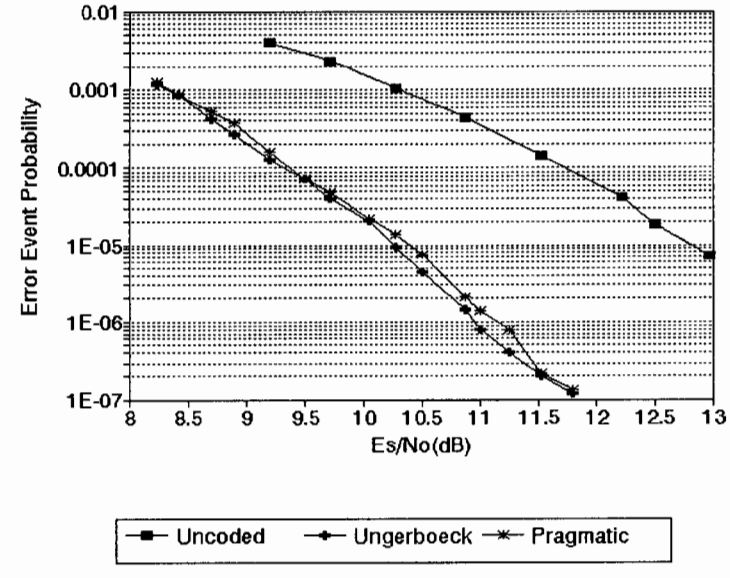

Figure 6-4: Pragmatic and Ungerboeck codes

the pragmatic code. A full explanation of why this is the case will follow in the sections on event length histograms and the effect of varying truncation depths. In terms of probability of symbol error Pragmatic codes performed better for almost all signal to noise ratios. The fact that pragmatic codes performed better than Ungerboeck codes in terms of probability of symbol error might appear to be contrary to what is expected. In the following section however it is possible to see why this occurs and why it is in line with the structure of the codes.

6.5 Error Event Length Histograms

The processing performed in the software on the position of symbol errors determined both the number and length of error events. The distribution of error event lengths for the 2 codes used shows up important differences in the structure of these codes. It is also important for determining the trade off between truncation depth and error performance.

Error event lengths were initially counted by determining the number of symbol errors in each error event where an error event was determined using the criteria explained in section 6.3. This ignores branches that while on the incorrect path, result in the correct decoded symbol. This method measures errors per error event rather than error event length. An improvement was made to this by calculating the difference between the symbol position of the last and first symbol error in an event. All the histograms plotted in this chapter use the improved method. An example of an error event length histogram for the Ungerboeck code with W=20 and at an E_s/N_o of 9.5dB, is shown in Figure 6-5.

The histograms in this chapter are all plotted in terms of absolute error events rather than rates of events. The reason for this is to make various comparisons simpler and more intuitive. They were all plotted for equal number of received symbols.

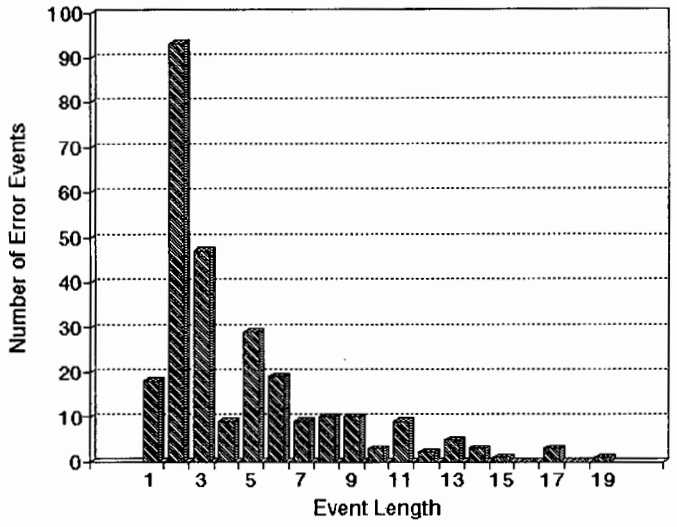

Figure 6-5: Histogram for the Ungerboeck Code

There are a number of important points that enable correct interpretation of this histogram:

- As explained in section 5.3.1, for the Ungerboeck trellis, all the branches entering a state are those resulting from the same input at the previous time unit. When counting merged error events therefore, the last branch of the incorrect sequence will have the same decoded symbol as that for the correct sequence and will not be counted as part of the error event. In terms of the histogram this implies that for all merged paths the event length is one longer than the event length shown in in histogram.

- With the above observation in mind, it is clear that the event with length 3 (shown in the length 2 column in Figure 6-5) is the most frequent event. This is the event corresponding to d_{free}. This observation confirms the fact that the error event corresponding to d_{free} is the most likely error event.

- There are 2 ways in which errors can be made in a Viterbi decoder: Errors as a result of merged paths and errors as a result of unmerged paths. The former result from an incorrect choice of survivor. The latter result from an incorrect choice of

the best state B_j. Adding one to the length of each event is only always correct for merged paths. Unmerged path errors result from events longer than the truncation depth, W. Therefore for W = 20, the number of errors due to unmerged paths will be far less than those due to merged paths. Stated differently, unmerged error events of length W + q will appear in the q column of the histogram but because they have such a long length they are unlikely to be a significant proportion of the events in column q. The histogram does not distinguish between errors due to merged and unmerged paths.

A histogram for the pragmatic code under the same conditions is shown in Figure 6-6. For pragmatic codes the simplification that the last branch of every merged error event will not be counted is not true. The fact that there are parallel branches entering each state prevent this simplification from applying to pragmatic codes.

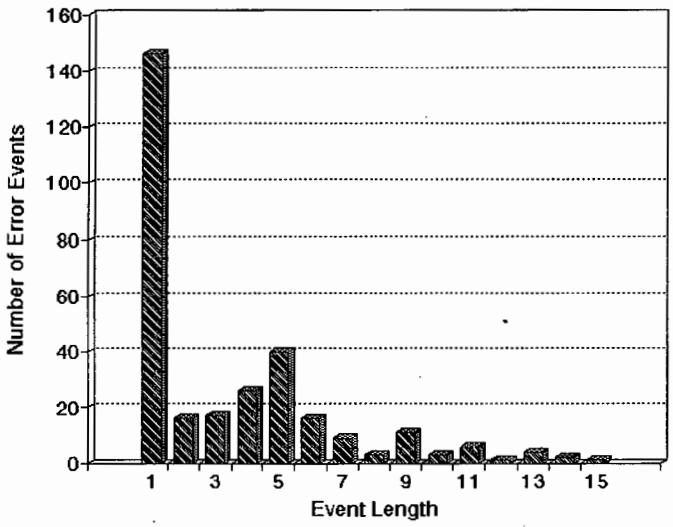

Figure 6-6: Histogram for the Pragmatic Code

It is apparent from the histogram that the most frequent event is that corresponding to the free distance which for the pragmatic code has length 1. Close inspection of the pragmatic trellis diagram shows that a merged error event of length 2 is not possible. ie. it is not possible for paths in a pragmatic trellis originating in a state at time j to merge at any other state at time j+2. Therefore events appearing in the column of

length 2 are either merged events with length greater than 2 or unmerged errors. In the following section this fact proves useful in estimating the number of unmerged paths at W+2 and from it the total number of unmerged paths. Another interesting observation from the histogram is the increased frequency of the event with length 5. This was observable for all values of signal to noise ratio and for truncation depth greater than about 18 indicating sequences differing by small Euclidean distances at this length.

Comparing the 2 histograms it is possible to see why Ungerboeck codes result in higher symbol error rates than pragmatic codes even though the latter has a higher ACG. For Ungerboeck codes the most frequent error event as shown on the histogram results in 2 symbol errors while the most frequent for pragmatic results in only 1 symbol. Thus although Ungerboeck codes make these errors less frequently than pragmatic codes, every Ungerboeck error event adds 2 symbol errors as opposed to 1 symbol error for pragmatic.

6.6 Results for varying Truncation depths

Truncation depths determine how much storage is necessary in a Viterbi decoder. It is in the interest of a designer of a hardware Viterbi decoder to have as small a truncation depth as possible for a required performance. Another possibility for saving resources in a hardware Viterbi decoder is in the module that decides which state B_j is the state with the lowest survivor metric. This requires comparisons of survivor metrics for all the states. Depending on whether the implementation is serial or fully parallel, these comparisons can be either time consuming or hardware intensive respectively. The sole purpose of choosing the best state is to prevent errors due to unmerged paths. Decisions about merged paths are made when a survivor is chosen from each state. If the truncation depth is made large enough, the number of unmerged paths becomes negligible and the choice of B_j becomes irrelevant. Thus to save on the survivor metric comparisons, the same state can be used as the best state with little loss of performance

provided the truncation depth is long enough. The loss in performance and the trade offs involved can be estimated using the plots in this section.

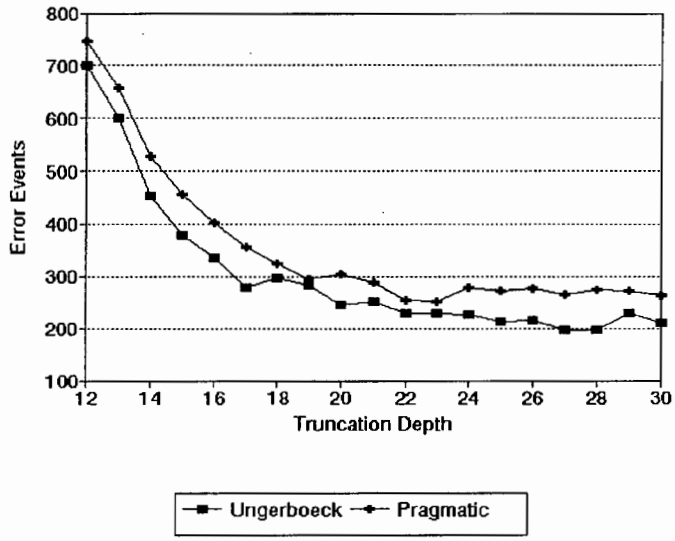

Figure 6-7: Results for Varying Truncation Depth

Figure 6-7 shows the result of varying truncation depth for both the Ungerboeck and pragmatic code at $E_s/N_o = 9.5dB$. An important aspect to notice is that for truncation depths greater than 20 the difference between Ungerboeck errors and pragmatic errors tends to increase. As a numerical indication of this, the largest difference in the number of error events between Ungerboeck and pragmatic codes occurs at W=28. This tends to indicate that for truncation depths larger than 20 the gain of the Ungerboeck code would increase relative to the pragmatic code. Thus Ungerboeck codes need a larger truncation depth than pragmatic codes indicating longer average event lengths. This agrees with the rule of thumb developed in [3] which states that for a 1/2 code structure (which is the case for the pragmatic code) the truncation depth should be about 5ν while for a 2/3 code (for the Ungerboeck case) the truncation depth should be about 8ν.

The histograms in Figures 6-8 and 6-9 are event length histograms for Ungerboeck and pragmatic codes respectively. In each case histograms are compared for $W_1 = 12$ and $W_2 = 30$. For W=30 there will be very few errors made as a result of unmerged paths. The important point to deduce is that for each event length L, the difference

between the height of the bars will be almost exactly equal to the number of unmerged errors that have length L for W=12. In general for any $W_2 > 30$ the number of unmerged paths at length L for truncation depth $W_1 < W_2$ can be approximated by the difference in the total error events between tests at $W = W_2$ and tests at $W = W_1$. The approximation becomes closer as W_2 is increased.

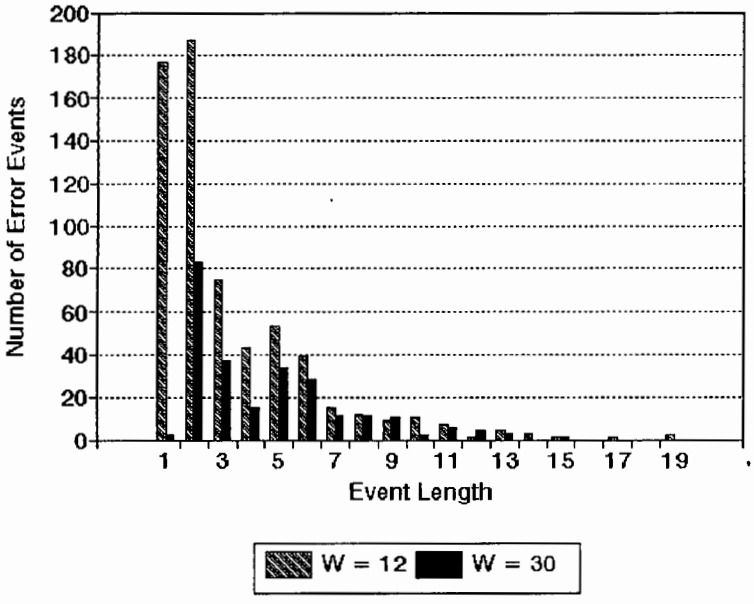

Figure 6-8: Ungerboeck Histogram for W=12 and W=30

For the Ungerboeck histogram as before to get true event lengths the lengths must be incremented by 1. The events shown at length 1 on the graph decrease to almost 0 for W=30 indicating that almost all errors at W=12 are unmerged errors. Thus the number of error events at length 1 for a given W, can be used to determine how many unmerged paths there are resulting in an error event of length 1.

For longer error events it can be seen that the difference between the height of the bars becomes less indicating that there are far fewer unmerged error events for longer events. For the pragmatic code, the error events at length 2 decrease to almost 0 which is in line with the fact that there cannot be any error events 2 symbols long for pragmatic codes.

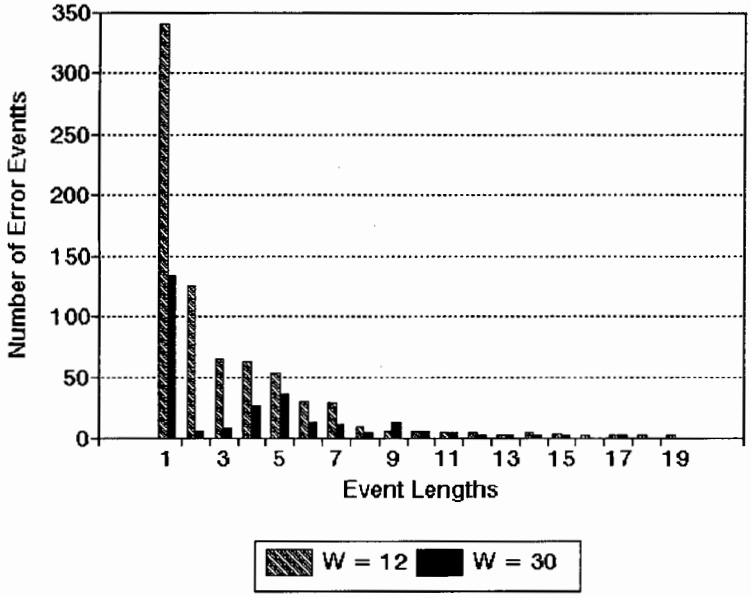

Figure 6-9: Pragmatic Histogram for W=12 and W=30

6.7 Results for varying Soft Decision Levels

The quantization levels in I and Q are an important factor in the design of a high speed Viterbi decoder for TCM. It determines the size of metric storage registers and the complexity of branch metric calculations. It is in the interest of the designer to use as few soft decision levels as possible for a given performance requirement. In the simulation for this project the ADCs used were 8 bits wide resulting in a maximum of $2^8 = 256$ soft decision levels. The signal range however did not cover the entire input range of the ADCs. The outputs from the software function that calculates the average constellation point values gives a signal range of 149 instead of 256. Thus when the received bytes are shifted by 1 the range is effectively halved every time. Figure 6-10 shows the result of halving the soft decision levels the results were taken with $E_s/N_o = 9.5$dB and a truncation depth of 20. In the graph, 8 bits represents the signal range of 149 and the range is halved as one less bit is used.

The results shows a minimal decrease in performance between 8,7 and 6 bits. For

the number of quantization bits less than 6 the performance degrades markedly. It is stated in [18] that for BPSK modulation there is only a 0.2dB loss of performance between 8 level soft decision and infinite level. Less than 8 bits results in a sharp drop in performance. 8 levels in BPSK is equivalent in the 8PSK case to having 8 quantization levels between each I or Q channel level or 24 quantization levels between the maximum and minimum I/Q levels. In the simulation for this project the ranges for 8,7,6,5,4 quantization bits are 149,74,37,18,9 levels respectively. The sharp drop in performance measured in this simulation thus occurs when decreasing the quantization levels from 37 to 18. The results in this simulation for varying soft decision levels thus agree with the result in [18].

6.8 Survivor Path Metrics

In equation 3.3 the survivor path metric, $P_j(m_j)$, was defined. Figure 6-11 shows the survivor path metric, measured during a simulation, corresponding to state 1 for a 7 bits long input PRBS. ie. generated using a 3 bit shift register. It can be seen that the metric dips below the surrounding values every 7 symbols. This occurs when the correct path moves through state 1 resulting in state 1 being selected as the state, B_j, with the smallest survivor path metric.

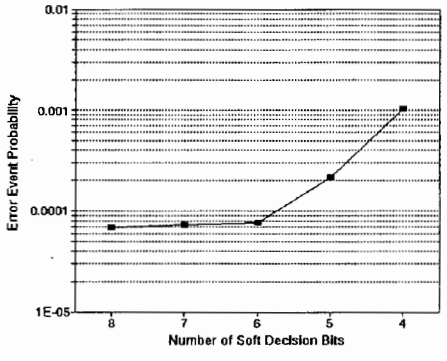

Figure 6-10: Performance for Varying Soft Decision Bits

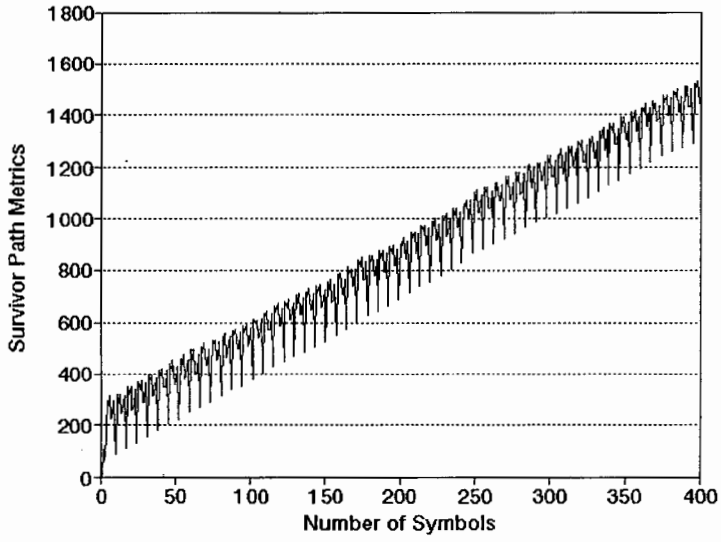

Figure 6-11: Survivor path metric - 3 bit input PRBS

Figure 6-11 shows the survivor path metrics for state 1 with the full 23 bit shift register input PRBS. Path metrics are shown for 4 different signal to noise ratios. Both Figures 6-11 and 6-12 have been plotted for Ungerboeck code simulations. As in the previous graph, the symbols for which state 1 becomes the state with the lowest metric, B_j, can be seen. In this case it does not occur periodically which is as a result of the larger PRBS.

Metrics for different signal to noise ratios can be distinguished by 2 factors. Firstly the slope increases for decreasing signal to noise ratios. Secondly, the difference between the metric when the state corresponds to B_j and the metric when it doesn't increases for increasing signal to noise ratios. The latter property provides an intuitive feel for how the decoder makes errors in the choice of B_j. It can also be seen from the graph that state 1 never corresponds to B_j for 2 consecutive symbols. This is in line with the fact that there are no branches in the Ungerboeck trellis connecting state 1 at time j to state 1 at time j+1. The same is not true for state 0 and 7 in the Pragmatic trellis and for states 0 2 5 and 7 in the Ungerboeck trellis. Plots of metrics for these cases showed that the state corresponds to B_j for consecutive symbols.

Plotting survivor path metrics provides verification that the branch metric calculation and the ACS function are operating as expected. A plot of survivor path metrics for different signal to noise ratios also has important implications for high speed hardware implementations of Viterbi decoders. In practical systems metrics cannot be allowed to grow without bound. Metric normalization, which involves subtracting a fixed amount from each survivor path metric, needs to be performed. Choices of how often to perform normalization and how big to make registers storing the survivor path metrics can be made with the aid of plots like that in Figure 6-12.

In practical TCM Viterbi decoders path metrics are also used to aid in carrier synchronization required for PSK [7], [6]. Synchronization of the carrier is monitored by monitoring the slope of the survivor path metrics. Although Figure 6-11 shows an increasing slope for increasing signal to noise ratio, a similar increase in slope can be observed for an increasing error in the phase of the recovered carrier. Thus synchronization is achieved by adjusting the phase of the recovered carrier based on the slope of the survivor path metrics.

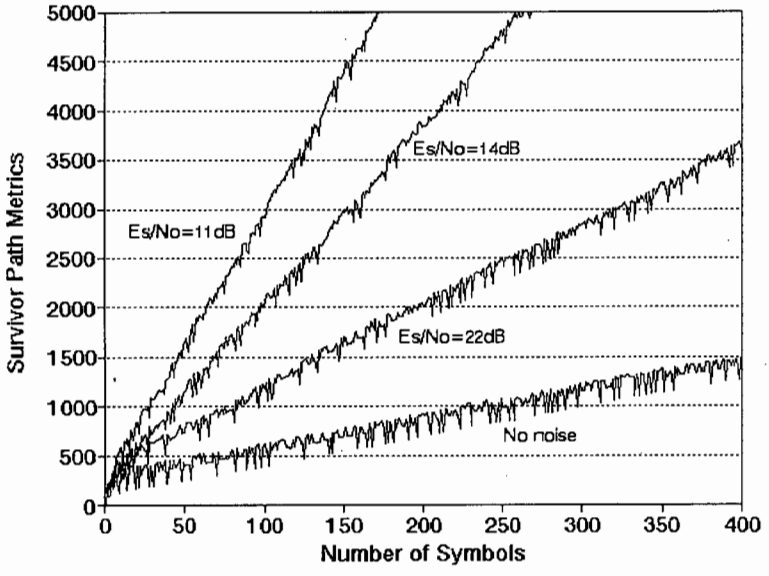

Figure 6-12: Survivor path metrics for different Es/No - 23 bit input PRBS

Chapter 7

Conclusions and Recommendations

Overall the thesis was successful in terms of achieving the original aims. The TCM simulation worked correctly, results were obtained which were interpreted and understood. More specifically the following can be concluded from the work done.

- Uncoded 4PSK performs extremely close to what is expected from the theory. This indicates that the simulation conditions are close to the ideal.
- For the purposes of comparing codes that have been optimized for the smallest d_{free}, the most reliable measure of performance is error event probability rather than bit or symbol error probability.
- The Ungerboeck code using a truncation depth of 20 provided substantial coding gains over the uncoded scheme. The coding gain fell well short of the ACG indicating that the ACG is an over optimistic estimate of performance. The measured performance did improve for increasing E_s/N_o.
- For the conditions of the simulation with a truncation depth of 20 the Ungerboeck $\nu = 3$ code performed better than the equivalent pragmatic code in terms of

error event performance. In terms of symbol error performance the pragmatic code performed better. Based on these results it can be concluded that minimal performance loss is incurred in moving from Ungerboeck to pragmatic coding. Thus for most applications pragmatic codes are recommended as the best solution because of their ease of implementation.

- Monitoring and collating results about the error event lengths reveal interesting and potentially useful features that change for different codes as well as for different truncation depths. Event lengths enable estimation of the number of unmerged error events for a given truncation depth which is useful in determining the optimum truncation depth and decoder simplifications

- For the $\nu = 3$ Ungerboeck code, a truncation depth of 20 is not sufficient to ensure the best performance. For the pragmatic code, little more improvement can be obtained by increasing the truncation depth beyond 20.

- There is little performance degradation in moving between 8,7 or 6 soft decision bits in each of the I and Q channels. Thereafter there is a sharp drop in performance. Increasing soft decision bits from 6 to 8 results in a considerable increase in decoder complexity for little increase in performance. Thus the best choice for soft decision levels to ensure near otpimal performance for the least decoder complexity is 6 bits in each of I and Q.

- Survivor path metrics are useful for monitoring synchronization. The survivor path metrics monitired in this project can also be used to decide optimal methods of metric normalization.

Finally, the author gained a thorough understanding of TCM and in particular Viterbi decoding. The fact that the simulation was real time and partly hardware, enabled the author to gain insight into some of the practical complexities involved, in particular noise generation and signal to noise ratio measurements.

Appendix A

Statistics of the Results

Throughout this thesis results have been given in terms of error probability. The measured results are however error rates rather than error probability. The reason for naming them error probabilities is that the values given are error rates that serve as estimates for the error probability. As they are estimates they have an associated variance and standard deviation. Assuming the errors counted are observations corresponding to a Poisson distribution, the standard deviation for an observation of x errors (events or symbols) out of n received symbols is

$$\sigma = \frac{\sqrt{x}}{n} \tag{A.1}$$

The tables below shows samples of the results of the error event rate tests for Ungerboeck and Pragmatic codes along with the values of x and n for each observation. In the table Pe represents the error event probability.

It can be seen that the results for high signal to noise ratios are far less reliable than

E_s/N_o	Symbols	Events	Pe
(dB)			
11.8	60E6	7	1.2E-6
11.5	55E6	11	2.0E-7
11.3	49E6	20	0.4E-6
11.0	49E6	39	0.8E-6
10.9	49E6	67	1.4E-6
10.5	35E6	155	0.4E-5
10.28	21E6	196	0.9E-5
10.1	21E6	427	2.0E-5
9.7	7E6	276	0.4E-4
9.5	3.5E6	250	0.7E-4
9.2	1.5E6	192	1.3E-4

Table A.1: Ungerboeck Error Event Results for W=20

E_s/N_o	Symbols	Events	Pe
(dB)			
11.8	60E6	8	1.3E-6
11.5	55E6	12	2.2E-7
11.3	49E6	40	0.8E-6
11.0	49E6	69	1.4E-6
10.9	49E6	102	2.1E-6
10.5	35E6	270	0.8E-5
10.28	21E6	285	1.4E-5
10.1	21E6	485	2.3E-5
9.7	7E6	348	0.5E-4
9.5	3.5E6	257	0.7E-4
9.2	1.5E6	242	1.6E-4

Table A.2: Pragmatic Error Event Results for W=20

those for lower signal to noise ratio. It was impractical to run the simulation longer than the longest run shown above which is 60×10^6. For a symbol rate of $f_s = 1200Hz$ this translates to approximately 14 hours.

Bibliography

[1] G. Ungerboeck. Channel coding with multilevel/phase signals. *IEEE Transactions on Information Theory*, IT-28(1):55–67, January 1982.

[2] S. Lin and D.J. Costello. *Error Control Coding : Fundamentals and Applications.* Prentice-Hall, January 1983.

[3] George C. Clark and J. Bibb Cain. *Error-Correction Coding for Digital Communications.* Plenum Press, New York and London, August 82.

[4] Peter J. Golda, Allon Benzakein, J.G. van de Groenendaal, and R.M. Braun. Vlsi appropriate design of a trace-back viterbi decoder. *South Africa Section IEEE*, pages 76–80, October 94.

[5] T.K. Truong, Irving S. Reed, Ming-Tang Shih, and E.H. Satorius. Vlsi design for a trace-back viterbi decoder. *IEEE Communications Magazine*, 40(3):616–624, March 92.

[6] A.J. Viterbi, J.K. Wolf, E. Zehavi, and R. Padovani. A pragmatic approach to trellis-coded modulation. *IEEE Communications Magazine*, 27(7):11 – 19, July 1989.

[7] Q1875 pragmatic trellis decoder. *QUALCOMM Technical Data Sheet*, 1992.

[8] V.K. Dubey, N.K. Lim, and E. Gunawan. Performance evaluation of pragmatic tcm codes through band limited nonlinear satellite channel. *IEE Proceedings Part I, Commun., Speech & Vision*, 139(1):15–23, February 92.

[9] Lalit R. bahl and Frederick Jelinek. Rate 1/2 convolutional codes with complementary generators. *IEEE Transactions on Information Theory*, IT-17(6):718–727, November 1971.

[10] Theo Lindebaum. Design of a simulation for trellis coded modulation. *Masters Thesis, Dept. Elec. Eng, University of Cape Town*, October 92.

[11] P. Horowitz and W. Hill. *The Art of Electronics.* Cambridge Univeristy Press, Cambridge, 89.

[12] F.G. Stremler. *Introduction to Communication Systems (Third Edition).* Addison-Wesley Publishing Company, 1990.

[13] M.C. Jeruchim, P. Balaban, and K. Shanmugan. *Simulation of Communication Systems.* Plenum Press, New York and London, 1992.

[14] K. Feher. *Digital Communications Microwave Applications.* Prentice-Hall, 1981.

[15] G. Ungerboeck. Trellis-coded modulation with redundant signal sets: Part i: Introduction. *IEEE Communications Magazine*, 25(2):5–11, February 1987.

[16] V.K. Dubey, N.K. Lim, and E. Gunawan. Pragmatic vs tcm codes: A performance comparison for severely band-limited non-linear satellite channel. *GLOBECOM*, pages 502.2.1–502.2.5, December 1990.

[17] S Haykin. *Digital Communications.* John Wiley and Sons, January 1988.

[18] Setting soft-decision thresholds for viterbi decoder code words from psk modems. *QUALCOMM Application Note AN1650-2*, April 1991.