NASA-CR-191374

p. 214

HIGH-SPEED ARCHITECTURE FOR THE DECODING OF TRELLIS-CODED MODULATION

Semi-Annual Status Report

Center for Space Telemetering and Telecommunications Systems Grant

Period Covered: January 1992-July 1992

NASA Grant No. NAG 5-1491

Principle Investigator: Dr. William P. Osborne

New Mexico State University
Electrical And Computer Engineering
Box 30001 - Dept. 3-0
Las Cruces, New Mexico 88003

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose and Scope

Since 1971, when the Viterbi Algorithm [1] was introduced as the optimal method of decoding convolutional codes, improvements in circuit technology, especially VLSI, have steadily increased its speed and practicality. Trellis-Coded Modulation (TCM), pioneered by Ungerboeck [2,3,4] since 1982, combines convolutional coding with higher level modulation (non-binary source alphabet) to provide forward error correction and spectral efficiency. For binary codes, the current state-of-the-art is a 64-state Viterbi decoder on a single CMOS chip, operating at a data rate of 25 Mbps [5,6]. Recently, there has been an interest in increasing the speed of the Viterbi Algorithm by improving the decoder architecture, or by reducing the algorithm itself. Designs employing new architectural techniques are now in existence, however these techniques are currently applied to simpler binary codes, not to TCM. The purpose of this report is to discuss TCM architectural considerations in general, and to present the design, at the logic gate level, of a specific TCM decoder which applies these considerations to achieve high-speed decoding.

The goal of TCM architecture research is to improve the performance TCM decoders with a minimum of hardware expansion. The emphasis is on 8-PSK and 16-PSK signalling, which provide spectral efficiency and constant amplitude, desirable attributes for satellite communications. Issues of interest include speed of operation, error correction capability (coding gain) and multimode operation, that is, the ability to process multiple modulation formats using a single device with a minimum of total hardware.

A number of approaches to the design of a high-speed TCM decoder are considered: 1) algorithmic reductions, which reduce processing time and hardware, 2) hardware expansion, or parallelism, increasing the throughput at the cost of additional hardware, 3) approximations: modifications to the algorithm which reduce hardware and processing time at the cost of compromise in performance (coding gain), 4) reductions in hardware which reduce total circuit area and allow implementation in a technology faster than CMOS. The Viterbi Algorithm consists of three distinct parts: metric calculation, the add-compare-select function, and path memory updating. Other parts of the process, which are not considered central to the Viterbi algorithm itself but are nevertheless necessary to the complete decoding system, are the quantization of the received signal vectors, and external circuitry to perform

various other functions. Examples of these other functions include outboard decision making, and the generation of soft decisions used to adopt an existing binary decoder to a non-binary channel in a pragmatic [7] TCM system. As shall be shown each of the aforementioned parts of the system has the potential to impact tne speed of the algorithm, and the volume of the required hardware. Also, each part of the system has the potential for reduction.

The format used to quantize the received signal vector directly determines the number of bits needed to represent the numerical quantities used in the algorithm. This ultimately affects the size of the device. Also, the choice of quantization will affect the coding gain [8,9,10]. Metric calculation must be directly matched to the quantization format. Metrics for any kind of coding system can be obtained by using the quantized decoder input as an address in a ROM, however there are advantages to be realized by designing special circuitry to calculate the metrics. The design presented in this paper obtains the metrics from combinational logic, avoiding the bulk and access time of a ROM, and allowing extensive pipelining.

The add-compare-select function includes a feedback loop that precludes pipelining. Fettweis and Meyer [11,12] consider this to be the principle bottleneck in the Viterbi Algorithm, and propose to speed up this part of the process

by combining multiple trellis stages into a super trellis stage with greater connectivity. To date, however, this technique has been applied only to simpler binary codes, and not to a TCM code.

The path memory consists of memory cells and switches interconnected in a way that reflects the trellis structure of convolutional codes. The memory is not complex, but is a significant user of chip area, a factor which is affected by the choice of coding standard. The external logic has less impact on speed of operation than do the other parts of the system but is necessary to the functioning of a complete system, especially if pragmatic TCM is used.

The essential parts of a TCM system have been briefly surveyed in the preceding paragraphs and will be discussed in greater detail later. One remaining issue to be mentioned briefly at this point is the selection of the code to be used. In general, the more powerful TCM codes require larger decoding machinery. The decoding performance of TCM codes has been well researched throughout the eighties; however, less is known about the effect of the choice of code on the architecture. In 1989 Viterbi [7] published the invention of pragmatic TCM, giving a number of very strong reasons why pragmatic TCM is likely to become the TCM coding standard of the future. Based on what has been learned in the preceding years, it

is unlikely that pragmatic TCM will be significantly improved upon, except at great expense. For example, rate 2/3 8-PSK TCM using the best known 64-state convolutional code achieves a coding gain of approximately 3.6 dB over uncoded QPSK, while pragmatic TCM using a simpler 64-state code achieves about 3db. Rate 2/3 8-PSK TCM, using a 1024-state code provides an approximately 1 dB improvement over the best 64-state code [13]. So efforts to improve on pragmatic TCM are probably unwarranted at this time. However, the performance of pragmatic TCM can be matched by using the best known 16-state Ungerboeck code, which can be implemented by a smaller machine. In terms of hardware volume, the two codes are close since the pragmatic code is simpler than the 16-state code in ways that make up for the greater number of states. However, the 16-state code was decided upon, for reasons that will be discussed throughout the remainder of this work.

1.2 Convolutional Codes and Viterbi Decoding

A simple convolutional encoder is shown in Figure 1.1. The device consists of a three-stage shift register with two binary (modulo-2) adders connected to the stages of the shift register as shown. Each binary adder functions as a

parity check, or an exclusive or. This encoder, although simpler than encoders used in practice, generates a reasonably powerful code. Data to be encoded is shifted into the shift register one bit at a time, then the code bits, c_0 and c_1, form the output sequence. This encoder generates two code bits for every bit to be encoded, and thus is said to have a code rate of 1/2. The fact that the number of code bits exceeds the number of input bits makes it possible to reconstruct the correct sequence, even if some of the codebits are received in error.

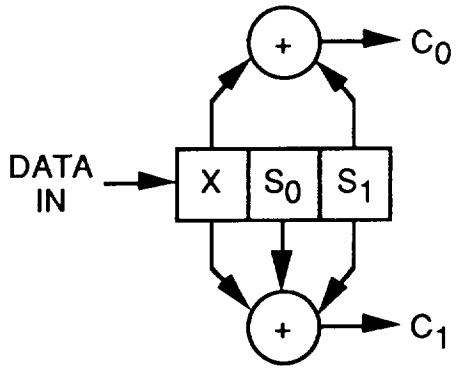

Figure 1.1. 4-state convolutional encoder.

The Viterbi algorithm for decoding a convolutional code sequence is based on the finite-state behavior of the convolutional encoder. The shift register in the convolutional encoder of Figure 1.1, has two bits of

memory. The first of the three stages is the current input, and so is not considered as memory. This encoder then is a 4-state machine, or 4-state convolutional encoder, and the code which it generates is referred to as a 4-state convolutional code. The contents of the memory stages defines the state, with s_0 being the least

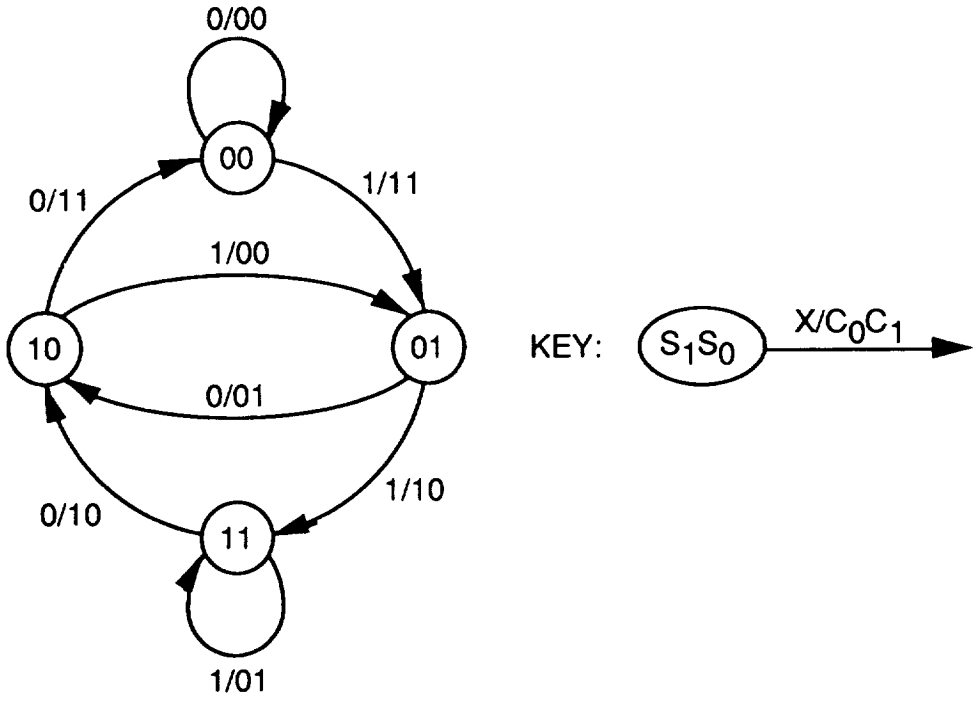

Figure 1.2. State diagram for 4-state convolutional encoder.

significant state bit, and s_1 being the most significant state bit. The relationship between current state, current input, current output, and next state is illustrated by the state diagram of Figure 1.2. A state j which can make a

transition to some state i is referred to as a predecessor to state i, and state i is referred to as a successor to state j. An output is associated with each allowed transition between two states. To represent all or any number of possible state transition histories over some period of time, the four states are arranged vertically, and then repeated horizontally to an arbitrary number of stages, resulting in the trellis diagram of Figure 1.3.

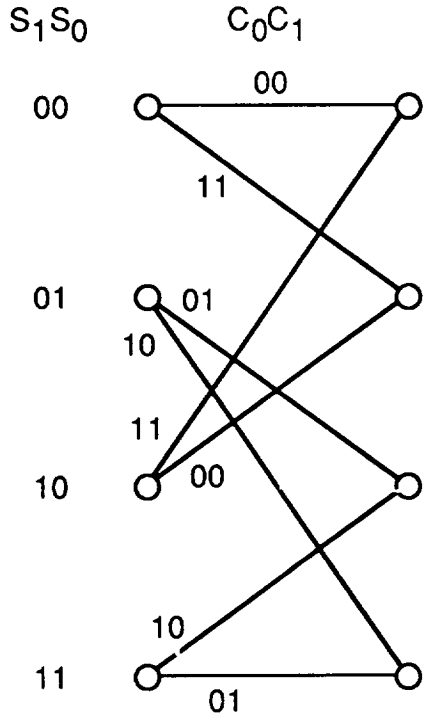

Figure 1.3. 4-state trellis diagram.

The trellis diagram shows the same state transitions as the state diagram, the difference being that the state diagram

is static, whereas the trellis diagram illustrates the behavior of the encoder over a number of periods of time.

The branches of the trellis diagram, representing state transitions, are labeled with the appropriate outputs. Any output sequence which the encoder will generate is made of the outputs associated with the branches of some continuous path through the trellis. If a receiver error occurs, the received sequence will most likely not be a legitimate code sequence, in which case the receiver must find the legitimate code sequence which most closely matches the received sequence. When binary signalling is used, the sequence is selected on the basis of Hamming distance, the number of corresponding bits in which a possible code sequence differs from the received sequence. Depending on the method of signalling, a measure other than Hamming distance could be used as a basis of selection. The measure to be used is referred to as the metric, and the decoder is said to find the minimum metric path. It is impractical to accomplish this by comparing the received sequence with every possible path, since the number of possible paths doubles with each stage of the trellis.

The Viterbi algorithm avoids this massive number of comparisons by taking advantage of the finite-state property of the convolutional encoder. At any given time,

regardless of which state the encoder is actually in, there exists a minimum metric path to each state. The minimum metric path to some state s_j at some time k must include, as a subpath, the minimum metric path to a predecessor to state s_j at time k-1. The number of subpaths which must be considered at any time is therefore limited to the number of states.

Dynamic programming [14] is a well established algorithm for solving problems in which the solution at some stage of operation is a subset of the solution at the next stage. It was Viterbi's insight that Dynamic Programming can be used to decode a convolutionally encoded sequence, therefore the use of dynamic programming in this way is referred to as the Viterbi algorithm. The Viterbi algorithm works as follows: associated with each node of the trellis, which represents a state of the encoder, is a minimum metric path to that node, and a metric for that path. The metric of the minimum metric path to a node is also referred to as the state metric or cumulative metric. Initially, when no part of the code sequence has been received, all of the node metrics are zero. Each time a symbol (a pair of codebits) is received, the received symbol is compared to each branch symbol and a metric is associated with each branch of the current stage of the trellis. Each branch metric is added to the cumulative

metric at its origin node to form a new path metric. At each node of the current stage, the converging path with the least metric is selected, and the metric associated with this path is selected to be the new cumulative metric.

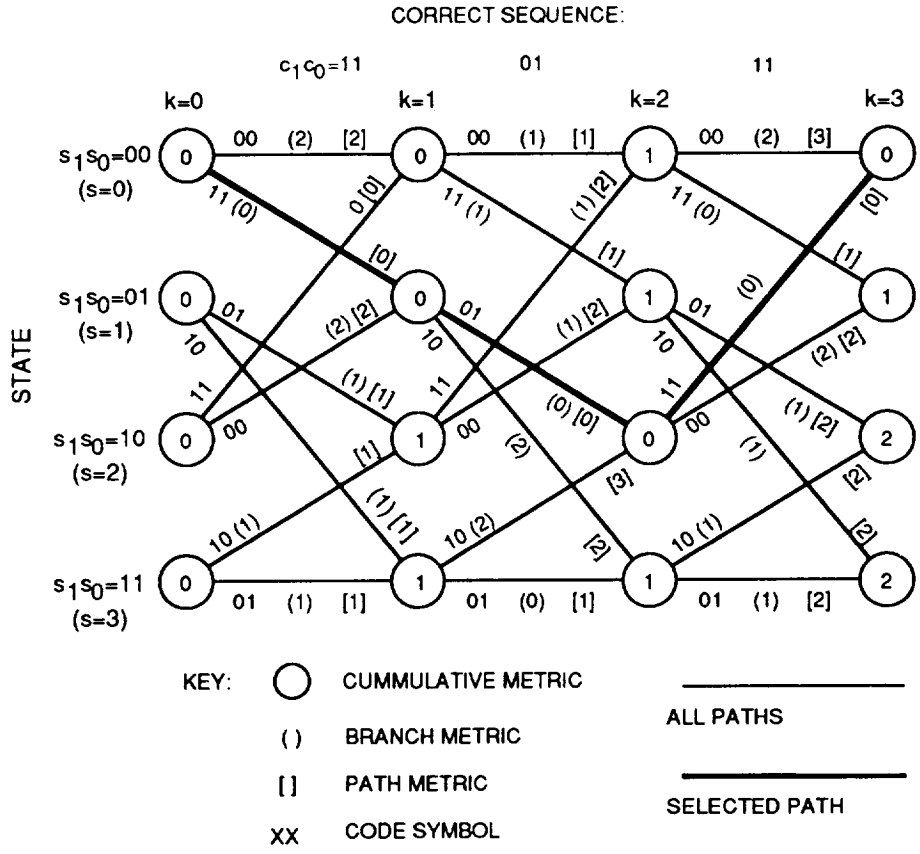

Figure 1.4. Viterbi decoding example.

As an example, suppose a single "1" is shifted through the shift register of the encoder. The resulting output sequence is 110111. The application of the Viterbi algorithm to this sequence is illustrated in Figure 1.4. At k=0, with no source code having been received, all state

nodes are set initially to zero. When the first symbol, 11, is received, branch metrics are computed for all branches, i.e., 2,1,1,0 for branches having symbols 00, 01, 10, 11 respectively. Because the initial cumulative metrics are all zero, the converging path metrics are the same as the branch metrics. At node (k,s) = (1,0), the lower branch is selected, having a path metric of 0, which becomes the new cumulative metric for state 0. At state 1, the upper branch is selected. At states 2 and 3, the upper and lower branches have equal metrics of 1, so the upper branch is selected arbitrarily.

At stage k=2, the same process is repeated, except that now there are non-zero previous cumulative metrics to be added to the branch metrics. At node (k,s) = (2,0), the upper branch has a previous cumulative metric of 0 and a branch metric of 1, resulting in a path metric of 1. The lower branch has a previous cumulative metric of 1 and a branch metric of 1, resulting in a path metric of 2. The upper branch, having the least path metric, is selected, resulting in a state metric of 1. Likewise, the upper branch is selected at nodes (k,s) = (2,1), and (2,2), the lower branch is selected at node (2,3), resulting in state metrics of 1, 0, and 1, respectively. At the third stage the process is repeated again. The correct path is identified by tracing backwards through the trellis. State 0, having the least state metric at the third stage, in

this case 0, is the starting point for the trace back. The lower branch has the lesser path metric, and leads back to node (2,2). At this node, the upper branch is selected, leading back to node (1,1). Here, the upper branch is selected, leading back to node (0,0), correctly identifying the sequence.

In the example of Figure 1.4, the encoder started in state 0 and finished in state 0. The decoder starts with all zero state metrics at stage 0, reflecting the fact that when no sequence has been received the decoder has no knowledge of the state of the encoder. After receiving 3 branches of correct code sequence, only the correct state has a metric of 0, the state metrics being 0, 1, 2, and 2, respectively. This reflects the fact that the decoder now has some information as to the current state of the encoder. The state metrics are updated each time a new code symbol is received, and the degree of certainty as to the state of the encoder depends on the metrics and the probability of error in transmitting a code bit, a characteristic of the channel. At all times, except during the brief start-up period, the decoder is operating with state metrics calculated from the previously received sequence, so it is the decoder's behavior in this condition which is of primary interest. Once the decoder has

received six stages of error-free symbols as is shown in Figure 1.5, the cumulative metrics reach steady state values.

With the decoder having reached this equilibrium, suppose that the encoder is made to transmit the same sequence as before, but this time two of the transmitted bits are received in error, so that the sequence 110111 is received as 100011. Figure 1.6 illustrates the operation of the decoder given this sequence, and is labeled with branch metrics, path metrics and state metrics, as is Figure 1.4. As can be seen, the Viterbi algorithm selects the correct sequence, although three further stages of operation are necessary for it to do so. If we receive the sequence with three errors, shown in Figure 1.7, the decoder selects an incorrect path. Thus we can see that the decoder has a positive but not unlimited capacity to correct errors.

The convolutional encoder is linear, i.e., the output due to the sum of two sequences is the sum of the outputs due to the individual sequences. Because of this, the encoder can be analyzed from the point of view that the all zeroes code sequence is correct, and the conclusions drawn will be applicable to all sequences in general (see Lin and Costello [15], Clarke and Cain [16], or Forney [17]). The examples of Figures 1.5, 1.6, and 1.7, show the Viterbi

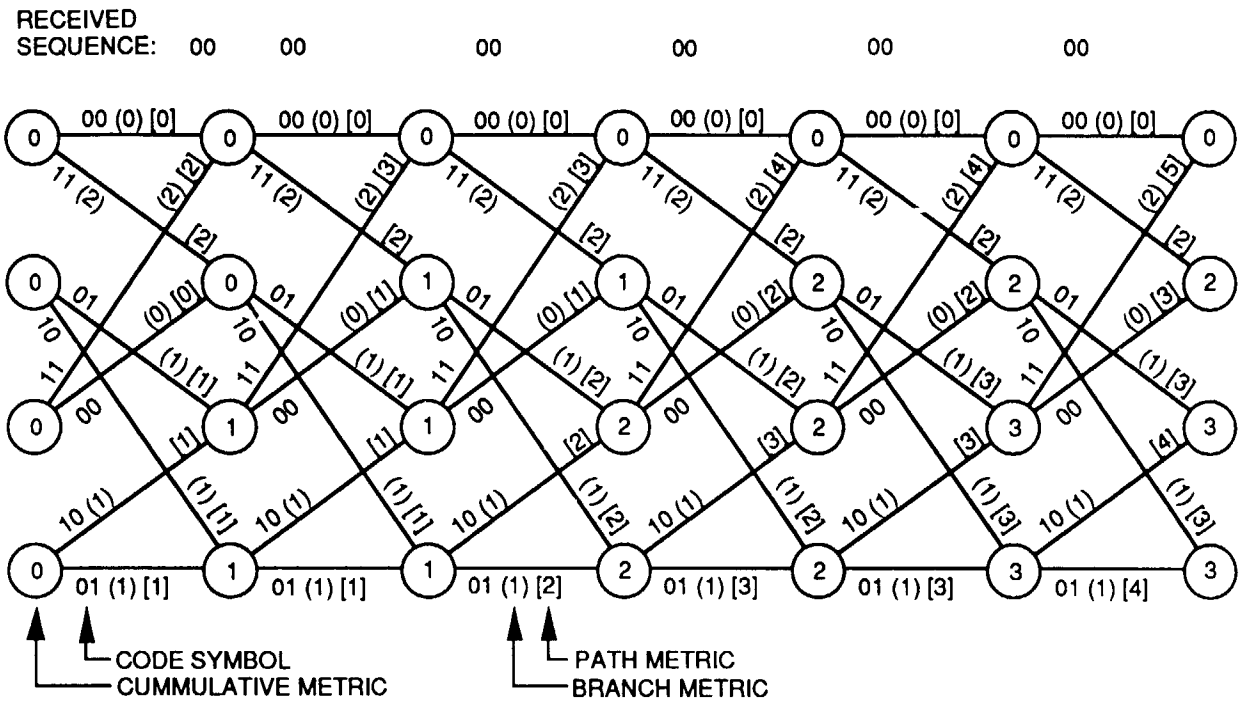

Figure 1.5. Trellis showing equilibrium state metrics.

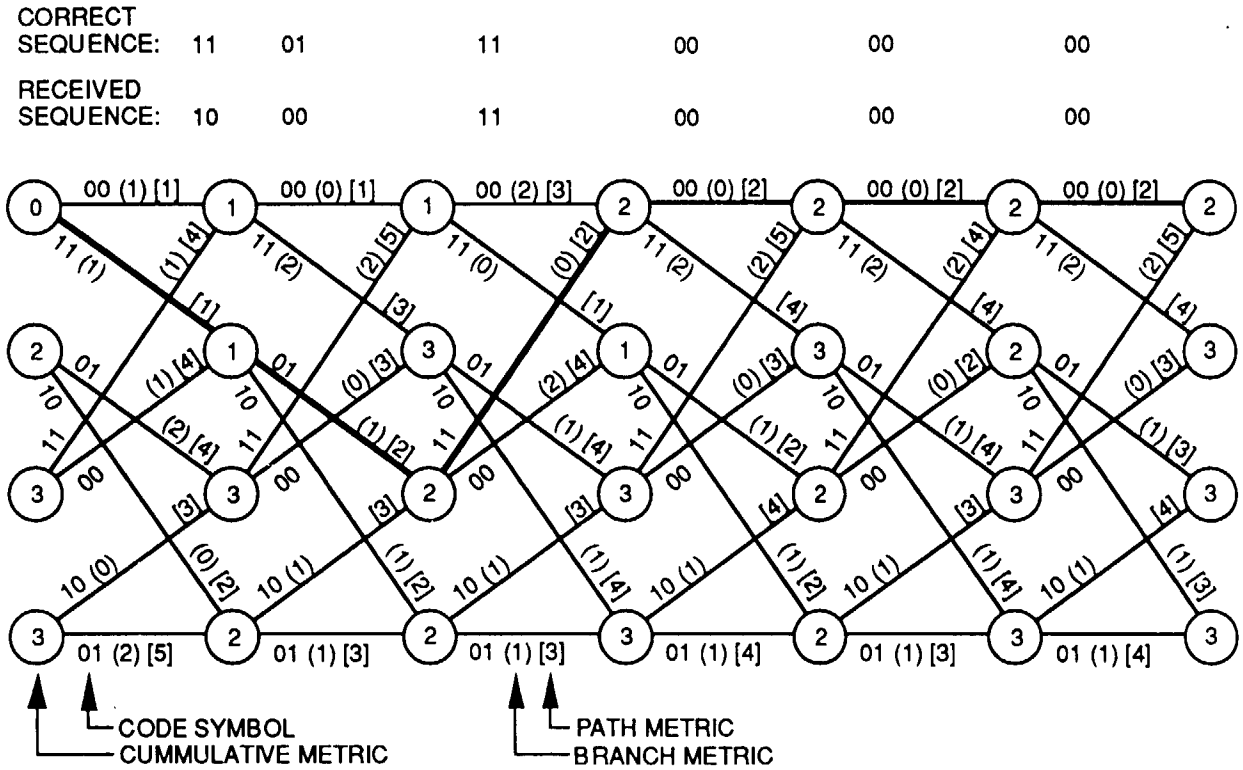

Figure 1.6. Corrected Trellis Error.

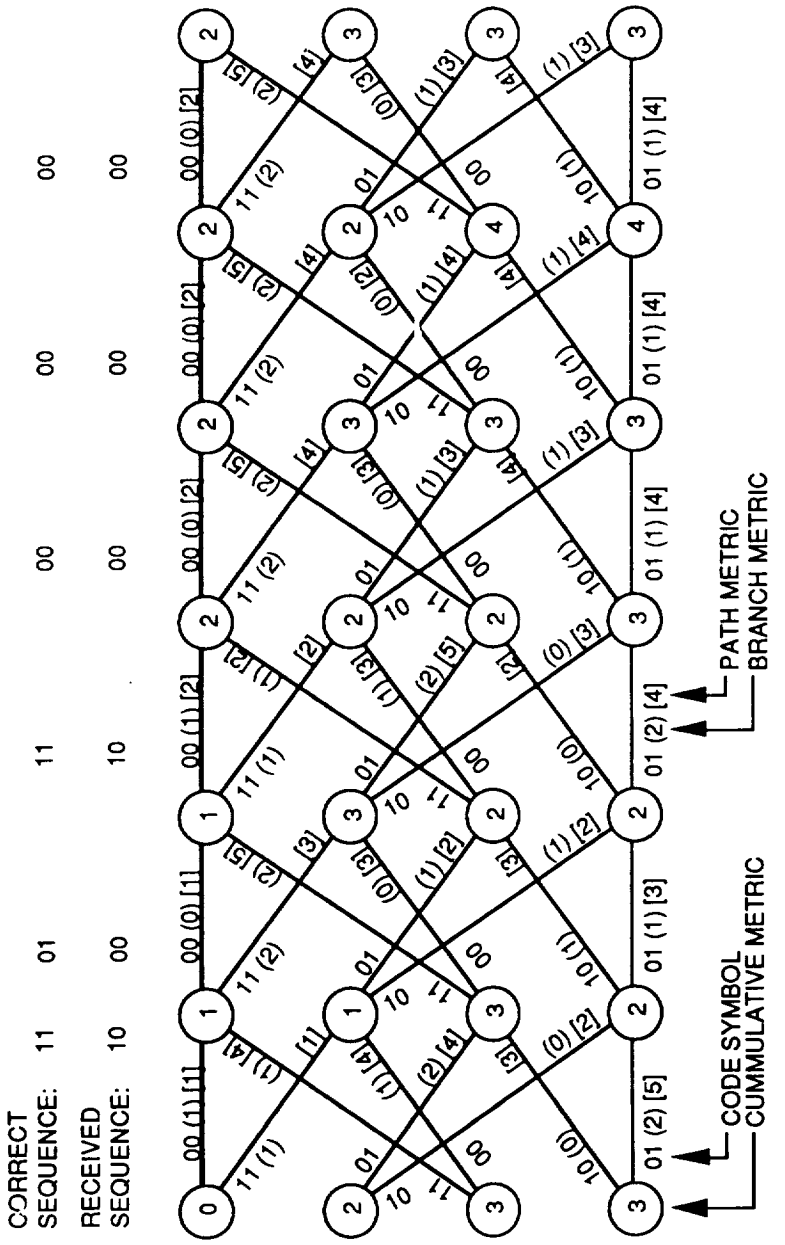

Figure 1.7. Uncorrected Trellis Error.

decoder tracing backwards from the minimum metric node. In fact, the path formed by tracing backwards from *any* node will tend to merge back to the maximum likelihood path, given enough time. The time required depends on the properties of the code and the channel, as well as the specific interfering noise. This means that the Viterbi decoder's memory does not need to retain the likely paths for all time, but only back to the point at which there is a high probability that all paths will be merged. The decoder operates the path memory in a pipeline fashion, such that old information is shifted out as new information is shifted in. If the path memory is made long enough, there is a high probability that the information being shifted out will be correct. There is still a nonzero probability of error, because it is possible that the transmission errors will be such that an error sequence more closely matches the received sequence than does the correct sequence. If this happens, the error-correcting capacity of the code is exceeded.

The error path illustrated in Figure 1.7 diverges from and then reconverges with the correct path. This is typical, because the metrics of non-converged error paths grow to the point that it is overwhelmingly likely that the Viterbi algorithm will eliminate them. Therefore, it is

the reconverged error paths which are of concern in predicting the performance of the code. Typical error sequences are shown in Figure 1.8. The decoder will select

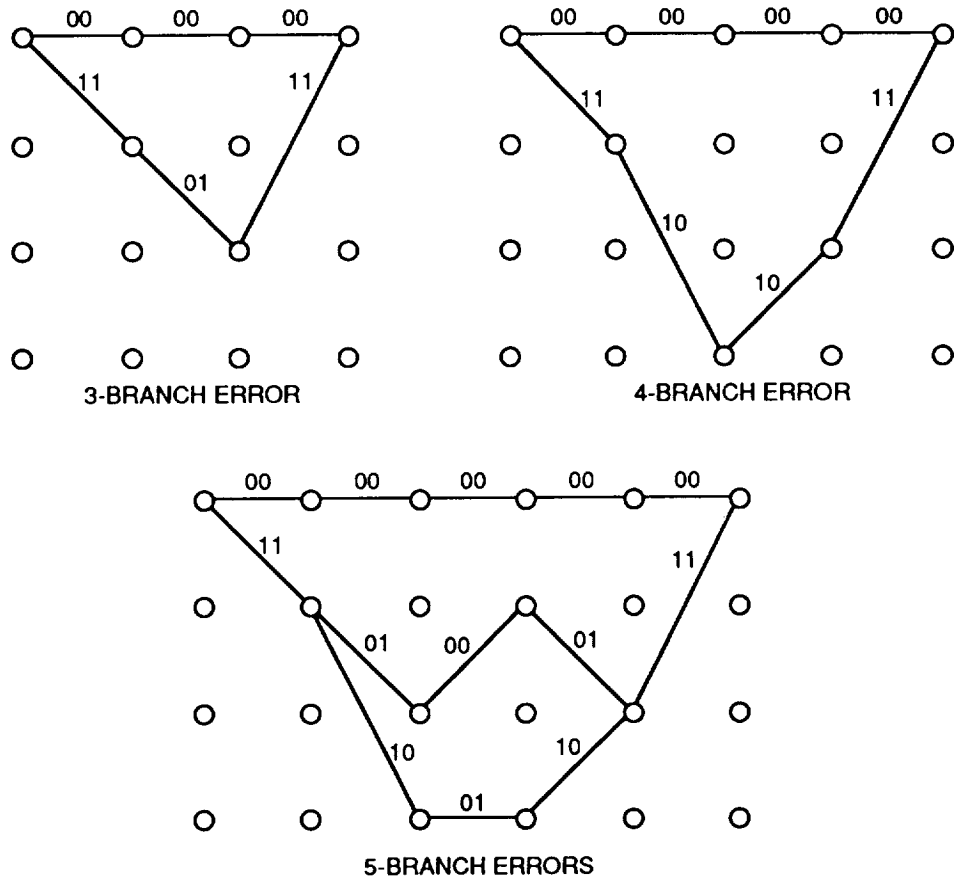

Figure 1.8. Typical error paths for 4-state code.

an error sequence if at any point the received sequence more closely matches an error sequence than the correct sequence. The probability that this will happen depends on the Hamming distance between the error sequence and the correct sequence. Thus we can see that the three-branch

error sequence is of the most concern as it differs from the correct sequence in only 5 bit positions. If three or more of these five bits are received in error, the three-branch error path will be selected. The longer error paths are less likely, as they differ from the correct path in a greater number of bits, yet they still make a non-negligible contribution to the total probability of error.

More powerful codes can be generated by using a convolutional encoder with more than a three-stage shift register, which will increase the number of states and the constraint length. The constraint length, K, is the minimum number of branches in which two paths can diverge and then reconverge. The constraint length for the code used in the previous examples is 3. In general, increasing the constraint length makes it possible to achieve greater Hamming distances for the error paths, and hence reduces the probability of error. This also increases the number of states, so the Viterbi decoder must then be built correspondingly larger. It is also possible to use more than two shift registers, generating more than two codebits for every data bit shifted in, or to design encoders which shift in more than one data bit for each codebit, generating codes of various code rates, i.e., 1/3, 1/4, 2/3 etc.

Obtaining the potential power of a code of given constraint length and code rate requires finding the optimal tap settings, that is, the best connections of the shift register to the adders. There is no analytical way to do this; however, the rule that the metric of the minimum metric error path should be maximized has proven effective. The codes in use today were found by exhaustive searches, comparing the minimum distance error events of the various codes. For the rate 1/2 4-state code, the minimum metric error path is also a constraint length path, but this is not necessarily the case for the more complex codes. Therefore, finding the most powerful codes is no straightforward task.

1.3 Trellis-Coded Modulation

Trellis-Coded Modulation (TCM), the invention of Ungerboeck [2,3,4], is the application of convolutional encoding and Viterbi decoding to non-binary channels to obtain the advantage of bandwidth efficiency. The Viterbi algorithm for TCM is essentially the same as it is for binary codes, the important differences being that the binary code symbols have been replaced by signal vectors, and that the metric is the square of the Euclidean distance in the signal set space, rather than the Hamming distance in the binary space. As an example of a TCM system,

consider the arrangement depicted in Figure 1.9. The binary data to be encoded is divided into two streams, one of which is fed into a rate 1/2 4-state convolutional encoder as discussed in the previous section, the other of which goes directly to the signal set mapper. The two bits

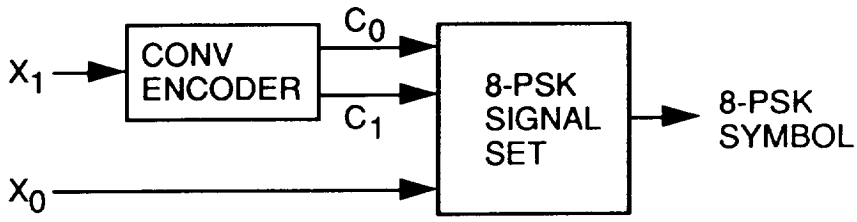

Figure 1.9. Rate 2/3 8-PSK TCM encoder.

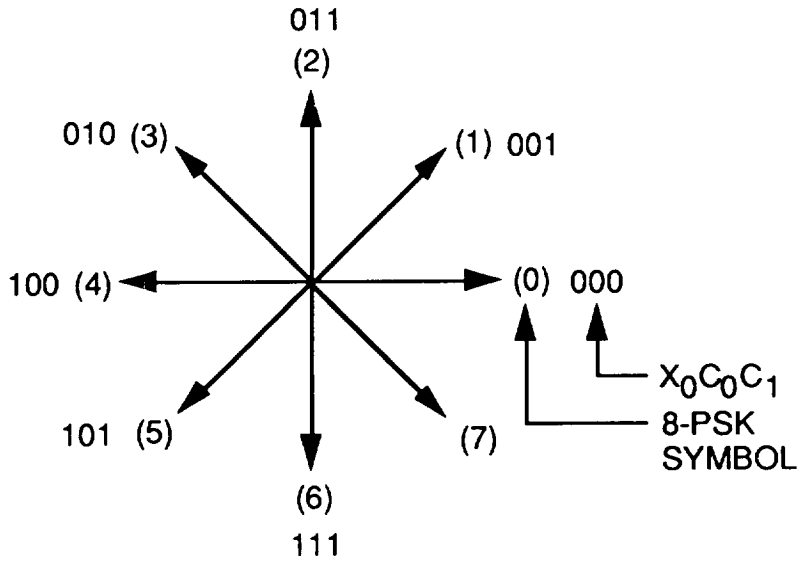

Figure 1.10. 8-PSK TCM signal set.

from the encoder, and the data bit which bypasses the encoder are mapped onto an 8-PSK signal set as shown in Figure 1.10. This arrangement is referred to as rate 2/3

encoding, because the 8-PSK symbol carries 3 code bits representing two encoded bits. The trellis diagram for this system is shown in Figure 1.11. Here, there are two branch symbols associated with each state transition, because only one of the two data bits determines the next state of the encoder, thus there are two ways to make any

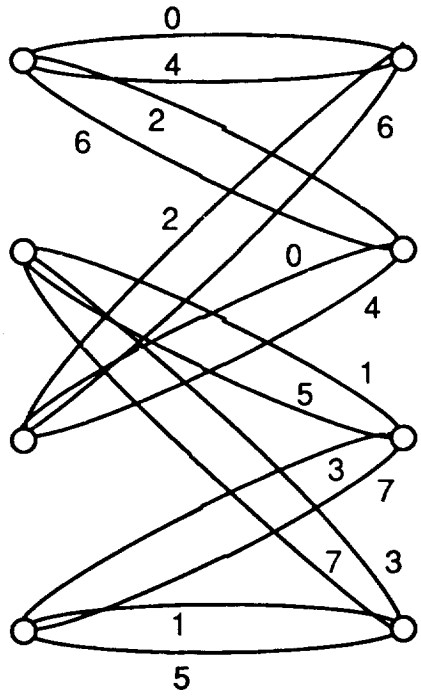

Figure 1.11. Trellis diagram for 4-state rate 2/3 8-PSK TCM.

given state transition. The Viterbi algorithm operates as in the first example, except that the squares of Euclidean distances between received signal and signal set vector are used in place of the Hamming distances.

As in the binary case, error events are paths which diverge from the correct path and then reconverge. The code sequence can be thought of as a vector whose dimension depends on the length of the sequence, i.e., a sequence of N two dimensional vectors can be treated as a vector of dimension 2N. The probability of error depends on the Euclidean distance between the sequence associated with the correct path, and that associated with an error event. The minimum distance error event for the system of Figure 1.9 is shown in Figure 1.12. The minimum distance error event is the most important error event, but longer error events also contribute significantly to the probability of error.

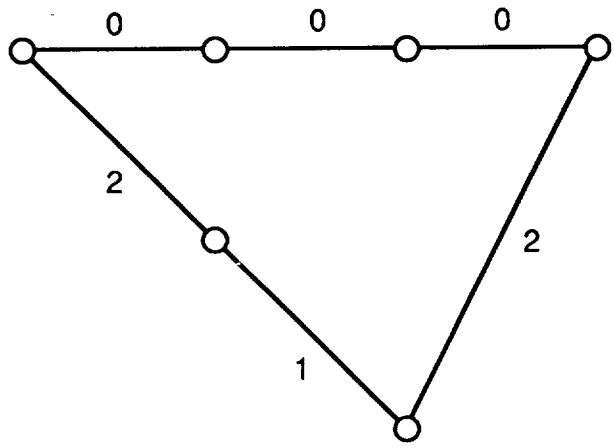

Figure 1.12. Minimum distance error event for 4-state rate 2/3 8-PSK TCM.

As in the binary case, it is possible to generate more powerful codes by using encoders with greater number of

states. For rate 2/3 8-PSK, encoders of 4,8,16, and 64 states are illustrated in Figure 1.13. The 16-state and 64-state encoders shift two data bits into the register each time a code symbol is generated, while the 4-state

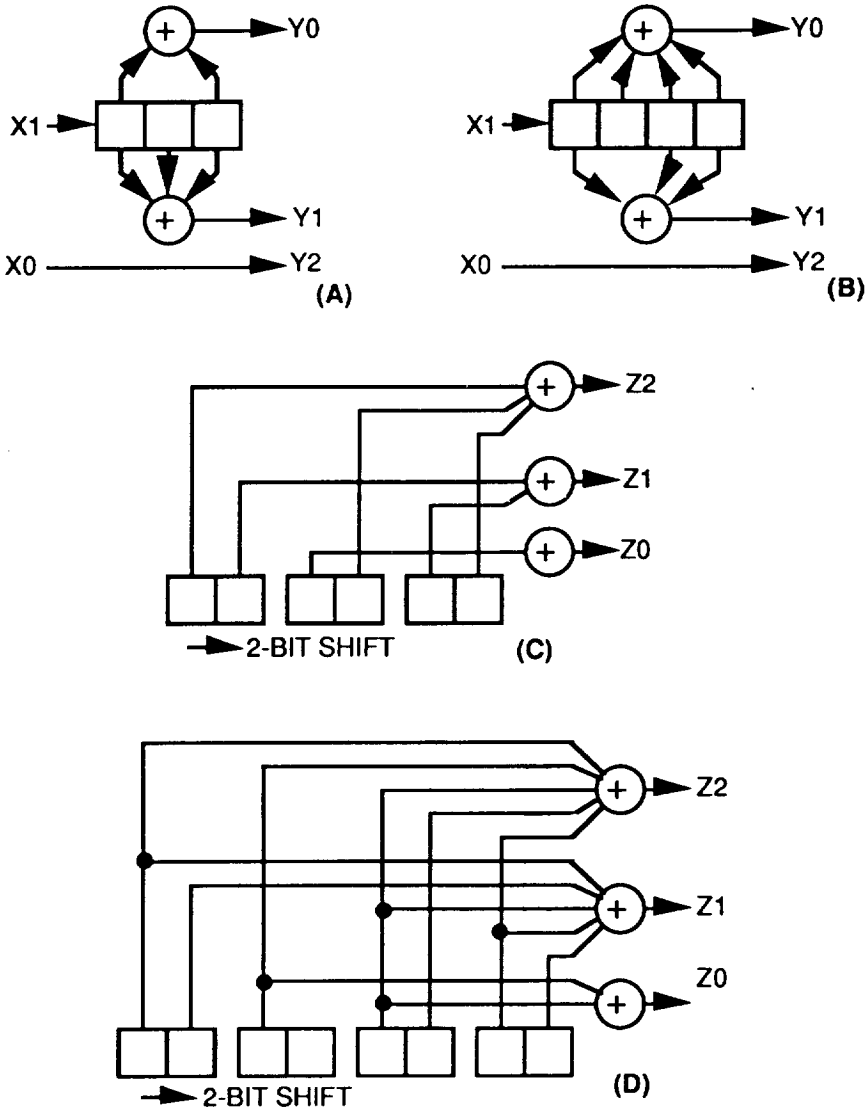

Figure 1.13. Rate 2/3 8-PSK TCM encoders: a) 4-state, b) 8-state, c) 16-state, d) 64-state.

and 8-state encoders shift in only one data bit, the other going directly to the signal set mapper. Given an encoder of a specific number of states, there is no simple analytical method to determine which of many possible tap settings will generate the best code; however, Ungerboeck has established principles for finding the better codes. First of these is the minimum metric criterion, that the code having the greatest metric for the minimum metric error event is expected to be the more powerful code (this is analogous to the minimum Hamming distance rule for binary codes). Next Ungerboeck established the set partitioning principles, which aid in maximizing the minimum distance: 1) all symbols are used with equal frequency, 2) symbols which have the greatest distance are assigned to parallel branches (branches which connect the same pair of states), and 3) symbols with the next greatest distance are assigned to branches which either diverge from the same state and reconverge to the same state. Using these principles, Ungerboeck conducted exhaustive searches to find the most powerful codes for 8-PSK, 16-PSK and a variety of QAM constellations using codes of varying number of states.

1.4 Coding Standards

The coding standard is the complete specification of the method to be used to represent the original data on the communications channel. This includes the type of code (such as convolutional or block) the code rate, the block length for a block code or the constraint length for a convolutional code, the specific code to be used (tap settings or generator polynomials), and the specific signal set (binary, QAM, PSK, etc.). The rate 1/2, constraint

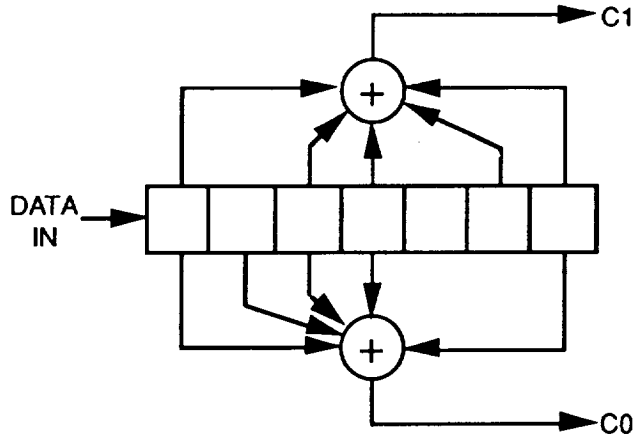

Figure 1.14. Industry standard rate 1/2 K=7 convolutional encoder.

length 7 convolutional encoder shown in Figure 1.14 is in common use today, and is referred to as the "defacto industry standard" [6,7]. Satellite links use this encoder in combination with a block code and convolutional code, with BPSK or QPSK signalling.

For PSK or QAM signalling, the required bandwidth is essentially proportional to the rate at which symbols are transmitted, and depends very little on the number of distinct symbols used in the system. Increasing the number of symbols increases the amount of information that can be transmitted in a given bandwidth (the spectral efficiency), but also increases the probability that one symbol will be mistaken for another (the probability of error), given the same average energy. While current terrestrial links may use signal sets of 256 symbols or more, current satellite links are almost entirely BPSK or QPSK. Future increases in demands for space communication are expected to require an increase in the spectral efficiency of satellite links, which will in turn require a shift from QPSK to a higher level of signalling (a signal set of more than 4 symbols). For a number of reasons, constant amplitude signalling is preferred for use in satellite communications. To increase spectral efficiency while preserving the property of constant amplitude signalling, the logical next step is a move from QPSK to 8-PSK, and possibly later to 16-PSK. However, due to the fact that satellite links are power-limited as well as bandwidth-limited, the use of error correction coding is also necessary. Therefore, the emphasis of this work is on rate 2/3 8-PSK, although rate

3/4 16-PSK is also covered in the section on the multimode codec, Section 3.3.

The power or energy saved by using an error correcting code is referred to as the coding gain. This is the difference in required signal to noise ratio for coded and uncoded systems maintaining the same bit error rate. In order for the comparison to be meaningful, the systems compared must have the same spectral efficiency. Thus the coding gain of a rate 2/3 8-PSK system is determined by comparison with an uncoded QPSK system, both of which carry two data bits per symbol. From the searches of Ungerboeck, it was found that for rate 2/3 8-PSK, most of the available coding gain is realized by the 4-state code, with diminishing marginal returns being obtained through 128 states. Indeed, it appears that most of the worthwhile coding gain is obtained at 64 states, although the construction of larger encoders might be worth the expense in certain specialized applications. As an example, a decoder for a 1024-state code, 16 times the size of a decoder for a 64-state code will produce a coding gain of approximately 1dB beyond that of the best 64-state code known. At a bit error rate of 10^{-5}, the 64-state Ungerboeck code achieves a coding gain of approximately 3.6 dB over uncoded QPSK. This is disappointingly less than

the coding gain predicted by considering only the most likely (minimum distance) error event.

Pragmatic TCM, the invention of Viterbi [7], uses the defacto industry standard convolutional encoder of Figure 1.14, in the TCM configuration of Figure 1.9. This arrangement, applicable to a variety of signal constellations, produces a potential coding gain of 3 dB when used for rate 2/3 8-PSK. Viterbi sets forth several strong arguments for the use of pragmatic TCM: pragmatic TCM is straightforward to implement, uses a currently available industry standard decoder, and uses the same decoder for a variety of modulation schemes, while sacrificing very little in coding gain compared to the optimal code. One of the advantages of the pragmatic standard is the possibility of multimode codec design, a TCM system which handles a variety of modulation formats with a single Viterbi decoder and a minimum of additional hardware. Design considerations for such a device are discussed in [18]. For these reasons, pragmatic TCM is expected to become the primary coding standard of the next decade.

As pointed out, pragmatic TCM has many practical advantages, however, in terms of coding gain, pragmatic TCM is not the optimal code for 64-state TCM. Indeed, pragmatic TCM is asymptotically limited to a coding gain of

3.2 dB, whereas the optimal 64-state code, achieving a coding gain of 3.6 dB at a bit error rate of 10^{-5} achieves continually improving coding gains at error rates less than 10^{-5}. The argument in favor of pragmatic TCM is that it is worthwhile to sacrifice 0.4 dB of coding gain in exchange for certain practical advantages. However, in an application in which 3 dB of coding gain is satisfactory, one might also consider the use of the 16-state Ungerboeck code, which also achieves a coding gain of 3 dB at a bit error rate of 10^{-5} and allows the use of a smaller Viterbi decoder. Also, the 16-state Ungerboeck code achieves a coding gain better than 3 dB at bit error rates less than 10^{-5}.

The choice of coding standard directly effects the architecture of the decoder. The size of the decision-making circuits and the path memory circuits is dictated by the structure of the trellis representing the code. The use of a smaller decoder is advantageous in consideration of high-speed architecture. It should be pointed out that the trellis for the pragmatic standard has only two branches converging into each node (that is from the point of view of the Viterbi decoder, the decision between parallel branches is accomplished external to the Viterbi decoder) whereas the 16-state Ungerboeck code has a trellis with four branches converging into each node. The

consequence of this is that the 16-state code requires approximately half as much hardware as the pragmatic code, rather than one fourth, as would first be thought by looking only at the number of states. Also, the need to make four-way decisions at each node adds additional complications. Therefore, the decision between the pragmatic code and the Ungerboeck code turns out to be rather close. Also, the techniques presented here could have been applied to the pragmatic code, or almost any other useful code. However, based on the consideration of all factors involved, the design presented here uses the 16-state Ungerboeck code.

1.5 Basic Implementation Considerations

From the preceding description of the Viterbi algorithm, one can begin to form an idea of what is required to implement the Viterbi algorithm in hardware. Three distinct operations are involved: 1) calculation of the branch metrics, 2) calculation of the path metrics and selection of the minimal metric path to each node (the "add-compare-select" function), and the path memory.

Metric calculation depends directly on the type of signalling used. In the case of binary signalling with binary channel outputs, logic is needed to calculate Hamming distances, whereas slightly more sophisticated

logic would be required if the use of a soft decision metric is desired. For TCM, the metric is the square of the Euclidean distance, and depends on the geometry of the signal set. In principle, the metric for TCM is a real number. Floating point calculation of metrics may be implemented, but there is no actual advantage in doing so, since the same performance can be obtained by using sufficiently fine quantization of the receiver signal space and the metrics, and using integer arithmetic. Incidentally, the required precision for numbers used to represent the received vectors and the associated metrics is an issue that would have to be faced regardless of whether integer or floating point arithmetic is used, because even floating point arithmetic units would have to be designed to accommodate a decided number of decimal digits. In fact, in designing a decoder for maximum speed, all arithmetic circuits should be custom designed for each specific calculation, so floating point arithmetic is not even considered, and all involved quantities are quantized to an appropriate integer scale. The issue which ultimately drives the entire design is the number of bits actually needed to represent the given quantity, which can be anywhere between 3 to 12, depending on the particular calculation, the coding standard, and the performance requirement. For this design, simulations were performed

to determine how the performance of the decoder would be affected by quantization of the received signal vectors and the metrics. This had to be done *after* the coding standard was decided upon.

It is possible to obtain the metrics from read only memory (ROM) lookup tables or from in-line arithmetic. This design shows that all of the metrics required for the 8-PSK circuit can be obtained by combinational logic using an equal or lesser number of logic gates than would be required for a ROM providing exactly the same metrics. Also, the arithmetic circuits offer the advantage of improved speed through pipelining. Included in the metric calculation of this decoder is a circuit which calculates the eight-bit square of a four bit number, and adding units fit especially to the application.

The add-compare-select circuit must include a register for the cumulative metric, compare path metrics and select the minimum, and have a means for handling metric overflow. At the binary level, the comparison operation is very similar to the addition operation, and can be pipelined. In the decoder discussed here, the problem of metric overflow was avoided by using the modulo-arithmetic method of Hekstra [19]. The add compare and select function will be more complicated if four paths converge into each node, as opposed to only two, and it turns out that a four-way

decision unit requires roughly twice the hardware of a two-way decision circuit.

The path memory circuit retains the paths selected by the add-compare-select circuit. The information which must be retained is the path selected at each node of the trellis, for all states of the code, and for the number of stages which must be retained to insure that all selected paths will merge. The number of stages retained is referred to as the decoder depth. If two paths are merged into each node, one bit of information is required per node; However, if four paths are merged, two bits are required. Thus the total capacity of the path memory circuit is number of states times decoder depth times the base two logarithm of the number of paths converging to a node. This means that the 16-state Ungerboeck code requires about half the memory of the pragmatic code, or one fourth the memory of the 64-state Ungerboeck code. In general, a longer constraint length code requires a longer decoder depth, although a greater number of branches converging into a node requires a longer decoder depth relative to the constraint length, another factor to be considered in selecting the code to minimize the size of the hardware. It is possible to design the circuit so that the information in the memory is the sequence of data bits associated with the various paths. In this approach,

decoded data is clocked out of the path memory at the same rate that received (and quantized) signal vectors are clocked into the metric calculation unit, although with a delay imposed by the decoding circuitry.

1.6 High-Speed Architecture Considerations

The throughput of the Viterbi algorithm, or nearly any other operation, can almost always be increased by building identical units side by side to perform the same operation. This approach, referred to as parallelism, increases the throughput rate by the same factor as the volume of the hardware is increased. Therefore, an increase in speed which is linearly proportional to an expansion in hardware is seen as a technical baseline; a technical achievement would be an increase in speed with a less than proportional increase in hardware. The design presented here will accomplish this. If a way to reduce the hardware volume were found, several parallel units could be built in the same area previously used for only one, accomplishing the desired improvement in speed-to-hardware ratio. Therefore, the problem of increasing speed, and that of reducing hardware are in many respects the same problem.

The timing associated with on-chip operations is a small factor compared to that required for chip to chip connections. Therefore, high-speed design ideally focuses

on single chip architecture, although Fetweiss and Meyr [11,12] work around this obstacle by building parallelism in very large blocks. The choice of VLSI technology offers tradeoffs between speed and chip area. Gallium Arsenide technology offers higher speed but lesser chip area than CMOS. State-of-the-art technology allows a 64-state binary Viterbi decoder on a single CMOS chip [5,6], and Qualcomm plans to offer pragmatic TCM on a single chip in the near future [20]. One possibility for increase in speed would be reduction of the algorithm to a scale that would allow the use of the faster technology, another way in which hardware reduction is closely related to speed improvement. Much of the current research in high-speed Viterbi decoding involves hybrid technologies, i.e., using the faster technology for the speed critical parts of the operation, and slower technology for the rest [21]. To date, a variety of novel techniques for high-speed Viterbi decoding are being applied to binary codes of less than 16 states, but not to more complex codes or TCM.

As discussed in the preceding paragraphs, the objective of high-speed architecture is to achieve an improvement in the ratio of speed-to-hardware volume. In absolute terms speed and hardware volume depend on the specific family of hardware chosen for the construction of the chip, however relative comparisons of various logic

designs can be made in terms of gate counts and gate delays. Thus the logic design can be optimized before the physical design problems are undertaken. For example: a CMOS inverter consists of two MOS transistors, while a NAND gate or a NOR gate consists of four transistors. Other basic logic elements, such as exclusive ORs and latches can be rendered as combinations of inverters, NAND gates and NOR gates. In this way, the overall circuit can be looked at in terms of volume and timing.

The design to be presented here uses extensive pipelining, using a fixed number of gate delay between pipeline stages. The design is totally synchronous, so that a single clock will drive the entire decoder system from beginning to end. The code used is the rate 2/3 8-PSK 16-state Ungerboeck code. Throughout the discussion, where possible, consideration will be given to the results which might have been obtained by applying similar design strategies to the pragmatic code. Throughout the remainder of this work, the design of the decoder will be discussed in terms of gate volume and gate delays, and the reasoning behind all design decisions will be explained.

2. QUANTIZATION

2.1 General Considerations

Ideally, Viterbi decoding of TCM would use floating point numbers for the received signal vectors, as well as the Euclidean metrics. However, due to the fact that, regardless of the technology used, it is impossible for floating point calculations to match the speed of integer calculations, some type of quantization will be employed, representing the involved quantities with a finite number of bits, and allowing metrics to be obtained from lookup tables, or by integer arithmetic. Quantization will always result in some degradation of error-correcting performance, but given an appropriate quantization scheme, performance can be made arbitrarily close to unquantized performance, by making the quantization sufficiently fine.

Quantization of the received signal vector may take a number of forms, the most prevalent being phase-only quantization, phase radius quantization, and rectangular coordinate (I and Q) quantization. This is because the problem of designing quantizers of these forms is at least approachable, whereas quantizers designed to suit generalized decision regions can be excessively complex. Regardless of the form of quantization chosen for the received signal vector, there is the additional issue of

quantization of the branch metrics. Metric quantization is closely related to, and directly affected by signal set quantization, but is an additional design consideration in its own right.

The required resolution of the received signal vectors and the metrics is also affected by the choice of coding standard. As an example of this, consider the following. Research done as part of the NMSU multimode codec study [18], which used the pragmatic standard, found that in the rate 2/3 8-PSK mode, 8-bit quantized I&Q with 4-bit metrics performed essentially as well as unquantized I,Q and metrics, whereas 4-bit I,Q and 4-bit metrics lost about 0.2 dB. Once it was decided that the high-speed design would use the 16-state Ungerboeck code rather than the pragmatic code, it was necessary to determine the necessary resolution of I,Q, and metrics. It was found that unlike the pragmatic code, the 16-state code required 5-bit metrics for satisfactory performance, using 4-bit I&Q. The 16-state code benefitted significantly from the use of 5-bit I&Q but then, only if 6-bit metrics were used. This was quite contrary to the experience with the pragmatic code.

It is reasonable to ask why the 16-state code should be more sensitive to quantization, especially of the metrics, than the pragmatic codes. The performance of any practical decoder is the combined result of the inherent

error-correcting power of the code and the quality of the information given to the decoder's decision unit in the form of metrics calculated from the received signal vector. For unquantized 8-PSK, at $\frac{E_S}{N_0}$ = 10dB (bit error rate between 10^{-5} and 10^{-6}), the performance of the 16-state code is essentially equal to that of the pragmatic code. It is therefore reasonable to ask if the same degree of metric quantization represents a different quality of information to the 16-state decoder than to the pragmatic decoder. This can be seen to be the case, because the pragmatic decoder selects the four signal vectors nearest the received vector (the outboard decision) before proceeding with the Viterbi algorithm. Therefore the pragmatic decoder compares four signal vectors on the basis of the metrics, whereas the 16-state code must use the metrics to distinguish between all eight vectors. The outboard decision does in fact represent an additional bit of information. The choice of quantization scheme for the high-speed decoder was based on simulation results, not on speculation, but the foregoing argument was advanced to show that the observed results are reasonable. It would be interesting to perform further experiments to verify that the effect of the outboard decision on the signal constellation is in fact the reason for the difference in sensitivity to metric quantization.

2.2 Information Theory Considerations

In nearly all of the TCM research done at NMSU, the performance of various quantization schemes has been determined experimentally, through simulation. It is also of interest to look at quantization from the point of view of information theory. According to classical information theory, especially the developments of Shannon [22], the probability distribution of the outputs of any channel with respect to the inputs establishes fundamental limits on the rate at which useful information can be transmitted through the channel. Two parameters of interest in this respect are the channel capacity, C, and the random coding bound, R_0, to be discussed later. In general, practical technology does not achieve the limits indicated by these parameters; however, they are of interest because all reasonably designed codes, at whatever complexity, will show similar, relative gains and losses in response to changes in these quantities. For TCM systems, the source has a discrete signal set, but due to the presence of noise (which is usually assumed to be additive white Gaussian), the received signal is a continuously distributed vector. The quantizer converts the received vector into a discrete output, causing the source, transmission medium, and quantizer to form a discrete channel. Let the set of source symbols be denoted $\{s_i\}$ for i=0,...,M-1 and the set of

output symbols be denoted $\{z_j\}$ for $j=0,\ldots,N$. The discrete channel is characterized by the matrix of transitional probabilities $p_{ij} = P(z_j|s_i)$, the probability that the quantizer selects output z_j given that signal vector s_i was transmitted.

The signal vectors received from the transmission medium convey a degree of information about the transmitted vector, depending on the physical characteristics of the medium, most importantly, the signal-to-noise ratio. The quantizer is included as a practical necessity but does not enhance the information from the channel, and in fact loses information. Clearly, the finer the quantization, the less loss of information. It has been well-argued, that two important parameters which affect the performance of any code using the outputs from the channel are the channel capacity, C, and the random coding bound R_0 [23,24]. Both of these quantities reflect the amount of information available to the decoder. The channel capacity is a concept invented by Shannon [22] and is an absolute limit on the rate at which information can be sent through the channel. The random coding bound is an information rate, derived from the probability of error averaged over all codes which can be supported on the channel [25]. It is impossible that any communication system could ever exceed the channel capacity, and it is generally not practical to

build a system which even meets the channel capacity. The random coding bound, being a statistically expected performance rate, is a more practical parameter than channel capacity. It has been shown [23,25] that the expected attainable error probability of codes on a channel is related exponentially to the block length of the code, and R_0 as follows:

$$P_e \leq C_R 2^{-NR_0} \tag{2.1}$$

provided that $R_0 > R$. Here, N is the block length of the code, R is the number of data bits per symbol, and C_R is an empirically determined constant. Performance at the rate indicated by R_0 has never been attained, since to do so would require large block lengths, and the use of soft decisions. To date, block codes use large block lengths but not soft decisions, whereas convolutional codes use soft decisions but have short block lengths.

For a continuous channel, the channel capacity and the random coding bound are defined in terms of the probability density functions of the channel. For the discrete channel, with a finite set of inputs $\{s_i\}$, and a finite set of outputs $\{z_j\}$, R_0 and C are calculated from the source probabilities $P(s_i)$ and the transitional probabilities $P(z_j|s_i)$ as follows:

$$C = - \sum_j P(z_j) \log_2 [P(z_j)]$$

$$+ \sum_i P(s_i) \sum_j P(z_j|s_i) \log_2 [P(z_j|s_i)] \qquad (2.2)$$

$$R_0 = - \log_2 \left\{ \sum_j \left[\sum_i P(s_i) \sqrt{P(z_j|s_i)} \right]^2 \right\} \qquad (2.3)$$

where the source probabilities $P(s_i)$ are chosen to maximize C and R_0. The derivation of R_0 is due to Gallager [25]. Nearly all channels of practical interest possess symmetry such that C and R_0 are maximized when the source symbols all have equal probabilities, that is $P(s_i) = \frac{1}{M}$, for all i, where M is the number of source symbols. In this case:

$$C = - \sum_j \log_2 [P(z_j)]$$

$$+ \frac{1}{M} \sum_i \sum_j P(z_j|s_i) \log_2 [P(z_j|s_i)] \qquad (2.4)$$

$$R_0 = - \log_2 \left\{ \frac{1}{M^2} \sum_j \left[\sum_i \sqrt{P(z_j|s_i)} \right]^2 \right\} \qquad (2.5)$$

If the channel has symmetry with respect to the relationships between inputs and outputs, that is, if the sets of transitional probabilities $\{P(z_j|s_a)\}$ and

$\{P(z_j|s_b)\}$ are different permutations of the same set for any inputs s_a and s_b, as is the case with the phase-only and I,Q quantization schemes discussed here, then

$\sum_j P(z_j|s_i) \log_2[P(z_j|s_i)]$ does not depend on i, in which case the calculation for the channel capacity further simplifies to:

$$C = -\sum_j P(z_j)\log_2[P(z_j)] + \sum_j P(z_j|s_0)\log_2[P(z_j|s_0)] \qquad (2.6)$$

The channel capacity and the random coding bound are measures of the information available to the decoder after quantizing. This will inevitably be less than before quantizing, however, as the quantizer is a practical necessity, quantizers are included in the system and designed to optimize these parameters.

2.3 Phase-Only Quantization

A TCM system can be made to work reasonably well with phase information only. While phase and magnitude both contribute to maximum likelihood decisions, phase-only quantization may be of interest in the case of non-linear

channels, or the case of relaxed requirements of automatic gain control. Also, phase-only quantization is an effective way to use a relatively small number of quantization regions, compared to I&Q quantization. Studies of phase-only quantization have been performed at NMSU since 1988 [8,9,10]. In these studies, the quantizing operation was modelled by defining a finite set of quantization points analogous to quantization levels in one-dimensional quantization. The receiver then selects the quantization point nearest the received signal vector, and the decoder calculates Euclidean metrics with respect to the quantization point, continuing the decoding process just as though the quantization point were the received vector. In this model, phase-only quantization is represented by locating the quantization points at even intervals on a circle of radius $\sqrt{E_s}$, as illustrated in Figures 2.1a and 2.1b, for the 24-sector phase-only quantization. In Figure 2.1a, 8 of the 24 quantization points coincide with the 8 signal vectors, whereas in Figure 2.1b, the quantization points are offset from the signal vectors. Because the quantization points lie on a circle, the term circular quantization was used. The rule of selecting the nearest quantization point generates the decision regions shown in Figure 2.1.

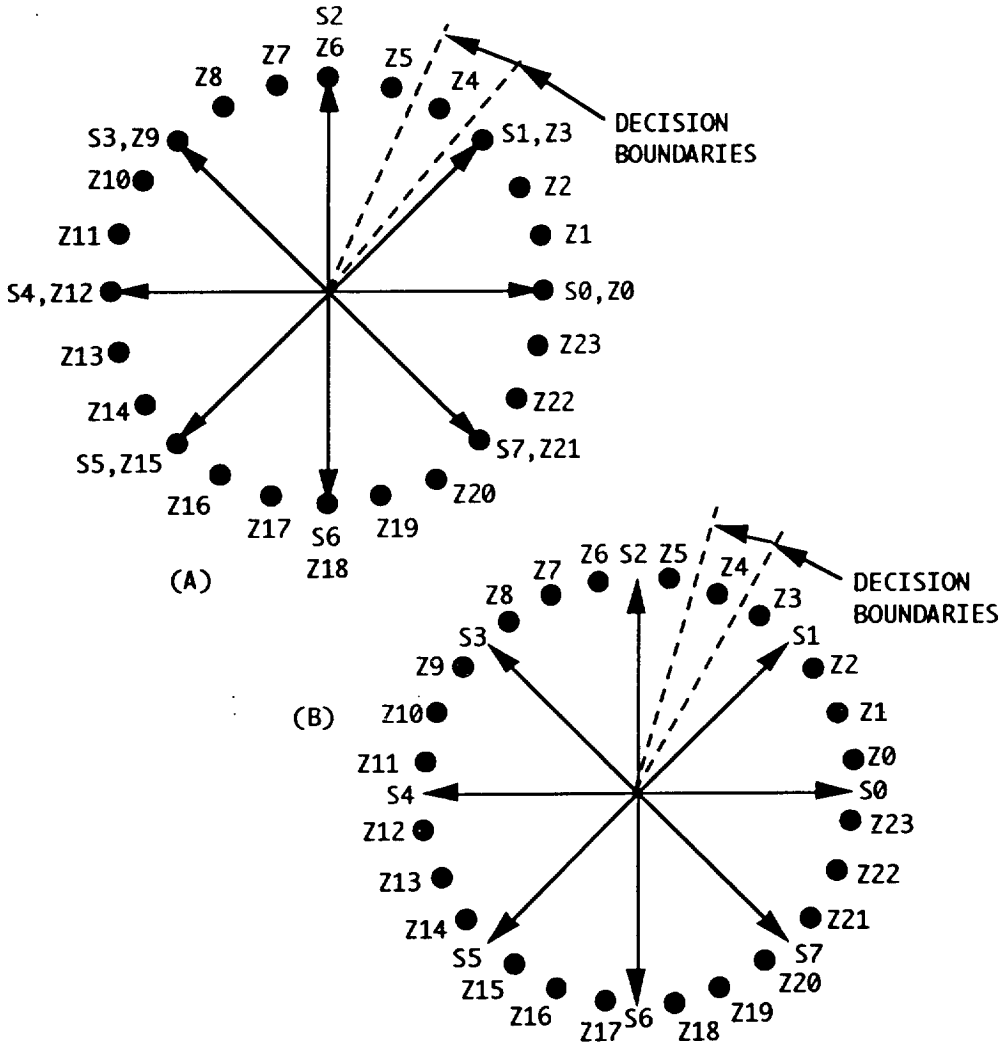

Figure 2.1. 8-PSK 24-sector phase quantization.

The quantization point model is extremely practical (simulations at NMSU demonstrated the performance of TCM systems using this model) but does treat the selection of optimal decision regions, and the optimization of metrics in great detail. Before further discussion, it should be pointed out that the gains to be obtained by fine-tuning

of decision regions and metrics are extremely small (which accounts for the success of the early NMSU simulations), and will most likely be eliminated when the metrics are quantized in a discrete decoder. The theoretically correct metric to use, for any set of decision regions, is the log-likelihood metric, to be discussed in Section 2.6. Once, metrics other than Euclidean distance are used, the location of the quantization points becomes less meaningful, and the quantizer is modelled merely as a set of decision regions. It then remains to discuss the optimal configuration of the decision regions. The notion of quantization points retains its utility, as it treats quantization as a question of precision of numerical quantities used in the algorithm, an issue which must be faced in hardware design anyway.

In 1990, Parsons and Wilson [26], using the term polar quantization, published a paper discussing the design of phase-only quantizers, for M-ary PSK with M=4,8, and 16, using quantizers of M, 2M and 4M zones. Their paper presents the design of phase-only quantizers which optimize R_0 by satisfying Lee's optimality criterion [27], and concludes that this condition is met (for the cases discussed) by a quantizer in which the signal vectors lie on boundaries of decision regions as shown in Figure 2.2b,

as opposed to one in which the signal vectors bisect decision regions as shown in Figure 2.2a. Parsons and Wilson derive their conclusion for 16-sector 8-PSK and then suggest that the same should also be true for 32-sector 8-PSK.

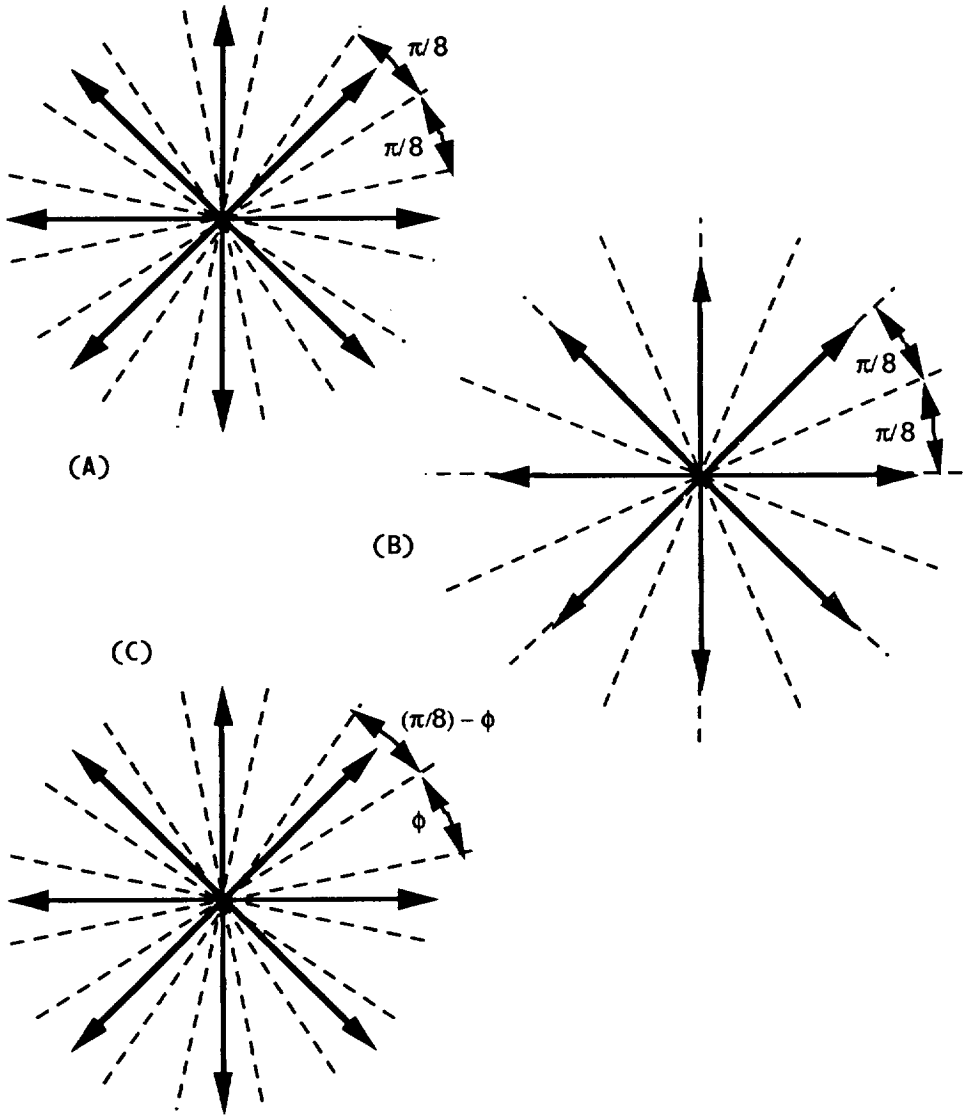

Figure 2.2. 8-PSK 16-sector phase quantization.

The early quantization studies at NMSU did not approach the optimization of quantization zones but rather looked at decoder bit error rate performance as a function of fineness of quantization (16-sector, 24-sector, 32-sector). Because these studies used the configurations of Figures 2.1a and 2.2a, it is in our interest to numerically evaluate R_0 for the various configurations, using equations (2.24), and (2.5). The results are shown in Figure 2.3. Although only three curves are apparent at first glance, there are actually six curves: R_0 for 16, 24 and 32-sector quantization, for both the case where signal vectors lie on the decision boundaries and the case in which they bisect the decision regions. As can be seen, whether the signal vectors lie on the boundaries of decision regions or in the centers of decision regions makes very little difference for 16-sector quantized 8-PSK, and essentially no difference for 24-sector and 32-sector quantized 8 PSK. To gain further insight into this issue, we shall look more closely at Parsons and Wilson's work [26], and look closely at what it means to satisfy Lee's optimization criterion.

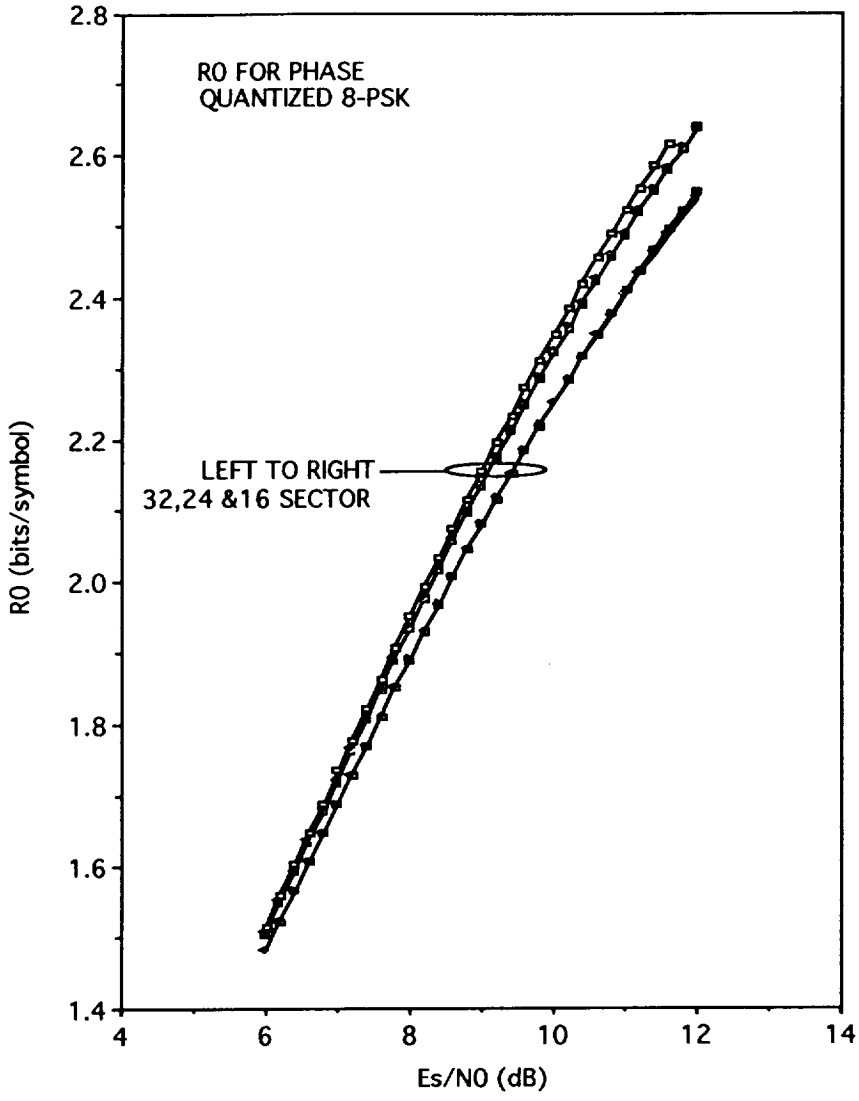

Figure 2.3. Random coding bound for phase quantized 8-PSK.

Lee's optimization criterion states that if ρ is a point on the boundary between two decision regions D_a and D_b of an optimal quantizer then:

$$\sum_{m=0}^{M-1}\left[\frac{1}{\sqrt{P(b|m)}} \sum_{i=0}^{M-1} \sqrt{P(b|i)} - \frac{1}{\sqrt{P(a|m)}} \sum_{i=0}^{M-1} \sqrt{P(a|i)} \right] f(\rho|m) = 0 \qquad (2.7)$$

where $f(x|m)$ is the probability density function of the received vector given that signal m was transmitted, $P(a|m)$ is the probability that the quantizer will select D_a, given that m was transmitted, and $P(b|m)$ is the probability that the quantizer will select D_b, given that m was transmitted. This meaning of Lee's criterion becomes more apparent when the equation is rewritten as

$$\sum_{m=0}^{M-1}\left[\frac{1}{\sqrt{P(a|m)}} \sum_{i=0}^{M-1} \sqrt{P(a|i)} \right] f(\rho|m) = \sum_{m=0}^{M-1}\left[\frac{1}{\sqrt{P(b|m)}} \sum_{i=0}^{M-1} \sqrt{P(b|i)} \right] f(\rho|m) \qquad (2.8)$$

The term on the left hand side represents the incremental contribution of the point ρ to R_0 if ρ is included in D_a, the term on the right represents the incremental

contribution if ρ is included in D_b. If the two are equal (as stated by Lee's criterion) then it is clear that ρ belongs on the boundary between D_a and D_b. If the term on the left side were greater (not fulfilling Lee's criterion) it would mean that R_0 could be increased by adjusting the boundary to include ρ within D_a, and likewise, if the term on the right were greater, it would mean that R_0 could be increased by including ρ in D_b. We can see then, that Lee's criterion is analogous to the condition that a single variable function is maximized at a point of zero derivative, and therefore constitutes a local, not a global maximizer of the R_0 function, a fact which Parson's and Wilson acknowledge. Thus we may interpret Lee's criterion as follows: If a set of decision boundaries is drawn, and Lee's Criterion is satisfied, then incrementally adjusting the boundaries will not increase R_0, but if Lee's criterion is not satisfied, then R_0 can be increased (or decreased) by incrementally adjusting the boundaries. Lee's criterion does not guarantee that R_0 could not be higher for some completely different configuration of quantizer decision regions.

Parsons and Wilson [26] acknowledge that their work proves the configuration of Figure 2.2b to be a local maximizer of R_0, not necessarily a global maximizer. In fact they state,

> "While a proof of global optimality seems difficult, we conjecture that the stated design is optimal, arguing as Lee did for the optimality of the J = M [J is the number of decision regions] design. Demonstration that no other design with J=2M satisfies [Lee's condition] would confirm this of course [p1513]."

Furthermore, Parsons and Wilson [26] do not attempt to show that Lee's condition is met for any configuration of 32-sector 8-PSK, (in fact, they state that phase-only quantization for J > 2M does not meet Lee's condition) and they do not discuss 24-sector 8-PSK. For 16-sector 8-PSK, however, Parsons and Wilson [26] have stated that the configuration of Figure 2.2b satisfies Lee's criterion, whereas the configuration of 2a does not, and thus conclude that it is better that the signal vectors lie on boundaries of decision regions, rather than in centers of decision regions.

We shall now examine 16-sector 8-PSK more closely. The configuration of Figure 2.2b satisfies Lee's criterion, therefore adjusting the decision boundaries will not improve R_0. The configuration of Figure 2.2a does not satisfy Lee's criterion, and therefore its value of R_0,

which is already close to that of Figure 2.2a, *can* be improved by adjusting the decision boundaries, specifically by varying the value of ϕ, as shown in Figure 2.2c. In this configuration the 8 sectors which encompass a signal vector have span of ϕ, the 8 which do not, have span of $\frac{\pi}{8} - \phi$. The optimal value of ϕ depends on the signal-to-noise ratio; however, by selecting an appropriate value of ϕ, it is possible to make R_0 for the configuration of Figure 2.2c exceed R_0 for the configuration of Figure 2.2b. Note that for $\phi = 0$, the configuration degenerates to hard decision 8-PSK, whereas for $\phi = \frac{\pi}{8}$, the configuration of Figure 2.2c is identical to that of Figure 2.2a. For the case of $\phi = \frac{\pi}{4}$ the configuration degenerates to a configuration of little practical value, 8 decision regions, with the decision boundaries coincident with the 8 signal vectors. With $\phi = \frac{\pi}{4}$, the channel capacity (and likewise, the random coding bound) of the configuration can never be more than 2 bits per symbol, at any signal-to-noise ratio. For hard decision 8-PSK, or reasonable values of ϕ, the capacity can approach 3 bits per symbol at sufficiently high SNR's. Figure 2.4 shows R_0 for the configuration of Figure 2.2c as a function of ϕ for $\frac{E_s}{N_0} = 9$, 10, and 11dB. As can be seen, ϕ can be selected to optimize R_0 at the expected signal-to-noise ratio. Figure 2.5 shows R_0 for the three

configurations of 16-sector 8 PSK shown in Figure 2.2. Here, ϕ is chosen to optimize R_0 at $\frac{E_S}{N_0}$ = 10dB. Note that if Figure 2.2c is optimized, the difference between 2c and either 2a or 2b, is greater than the difference between 2a and 2b.

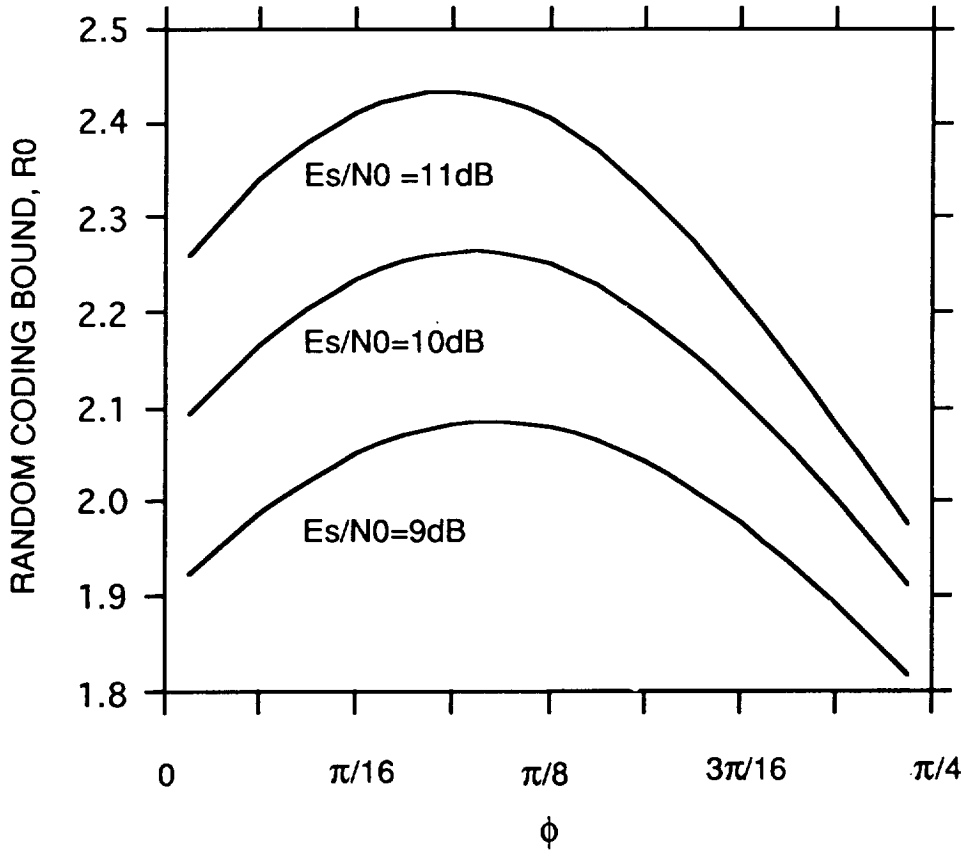

Figure 2.4. Random coding bound of Figure 2.2c.

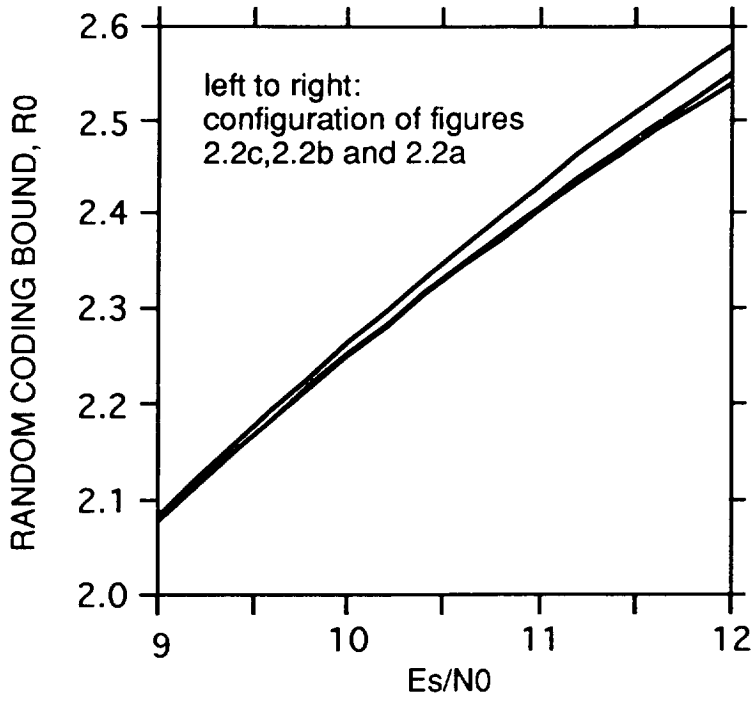

Figure 2.5. Random coding bound of 16-sector 8-PSK.

For 16-sector 8-PSK, the numerical differences in R_0 involved in the previous arguments are in fact very small. For channel capacity, the results are similar, as shown in Figures 2.6 and 2.7. Note, however that the channel capacity and the random coding bound are not necessarily optimized at the same value of ϕ. As the number of quantization regions is increased, the exact placement of the quantization zones becomes less critical in its effect on the performance of actual systems. For 24-sector 8-PSK, the performance of a 4-state TCM Ungerboeck code using the

decision region configurations of Figures 2.1a and 2.1b, is compared in Figure 2.8. This data was obtained from early simulations using the quantization point model, and Euclidean Metrics.

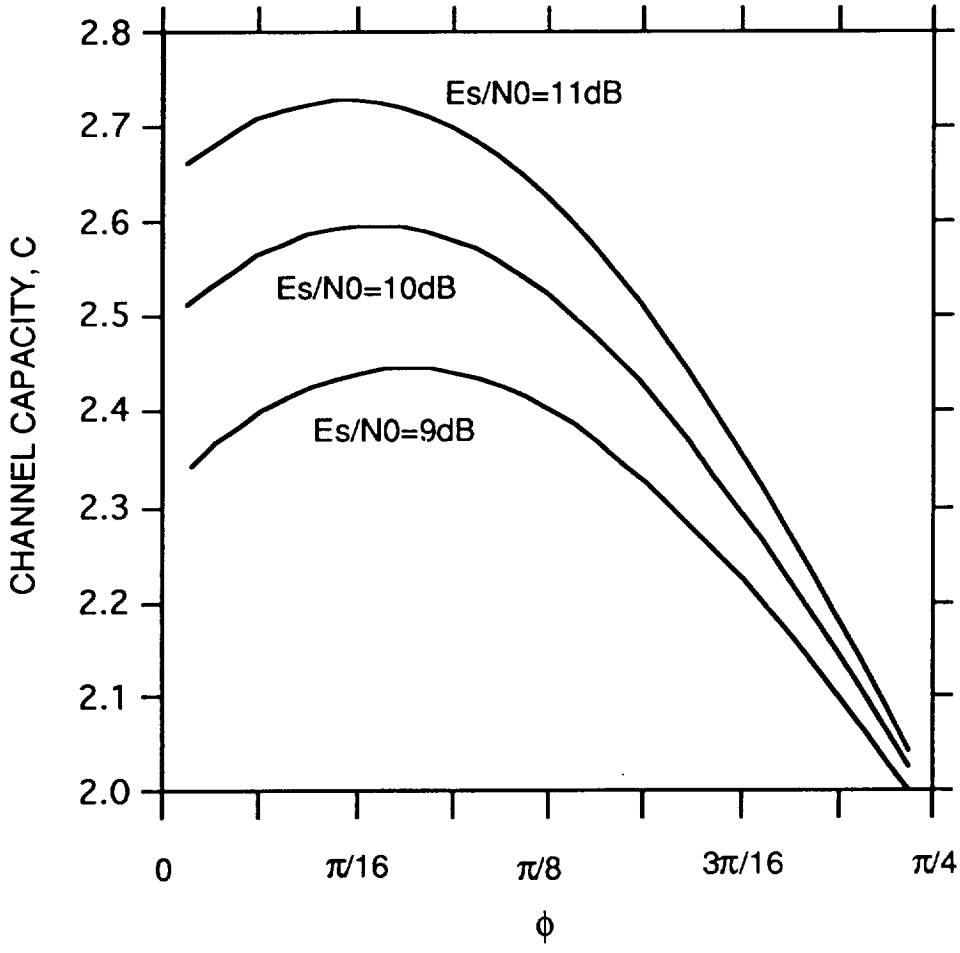

Figure 2.6. Channel capacity of constellation of Figure 2.2c.

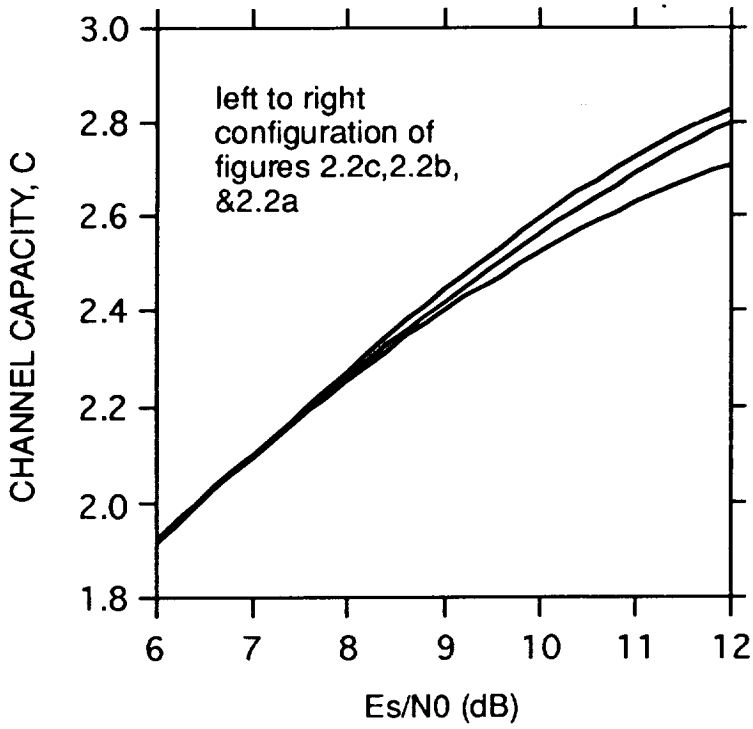

Figure 2.7. Channel capacity for 16-sector 8-PSK.

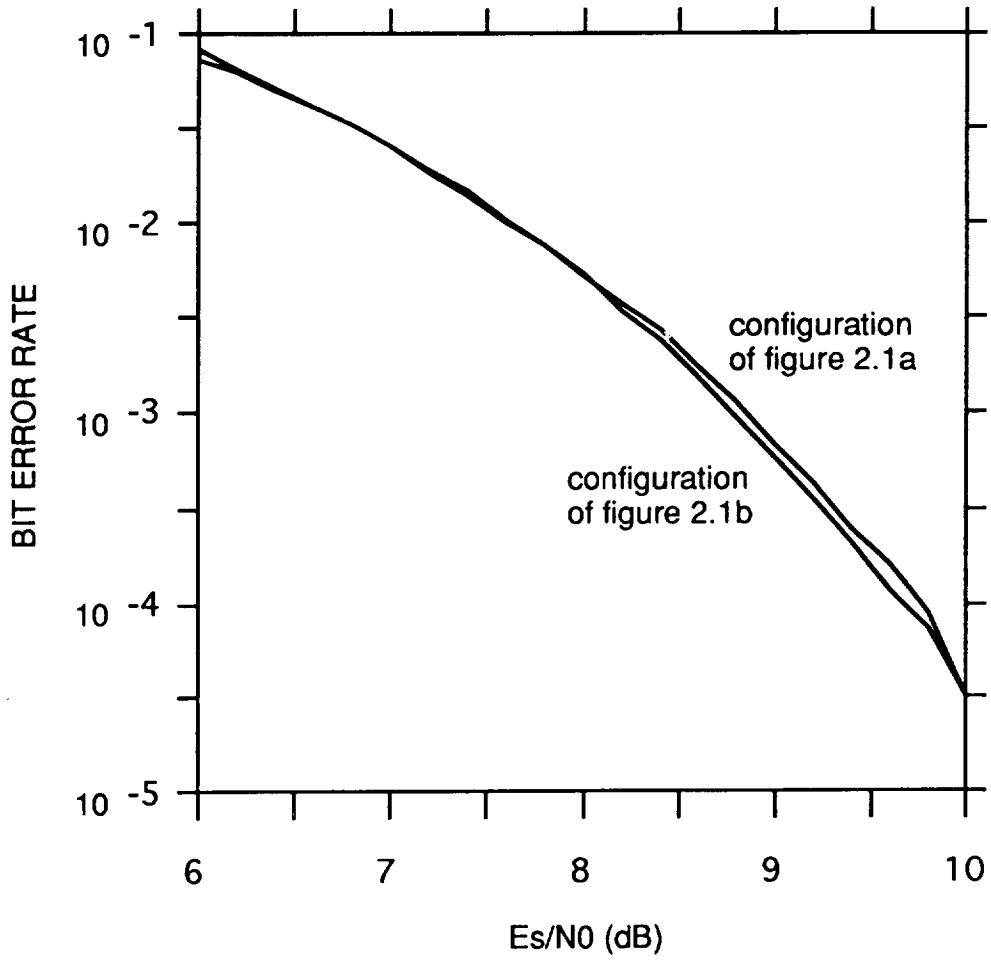

Figure 2.8. Performance of 24-sector 8-PSK, 4-state TCM.

The performance of 16-, 24-, and 32-sector 8-PSK, with a 4-state Ungerboeck code (previously published data [10]) is shown here in Figure 2.9. Also shown in Figure 2.9 is the performance of 8-PSK with unquantized phase and radius hardlimited to $\sqrt{E_S}$. These simulations also used the Euclidean metric. For comparison, the performance of unquantized 8-PSK is also shown. The unquantized phase-

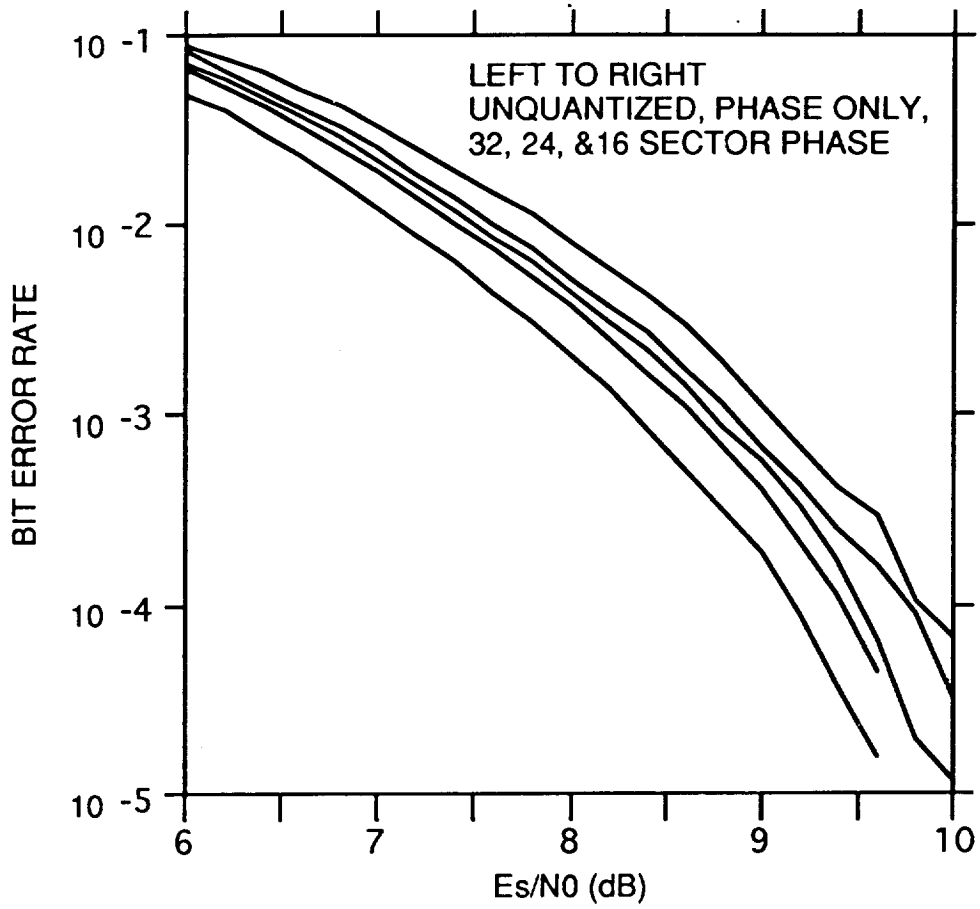

Figure 2.9. Performance of phase quantized 8PSK 4-state TCM.

only curve represents the limit on the performance of phase-only quantization, although a very slight improvement could be obtained by using an optimal metric. This reflects the fact that phase-only quantization, however fine, is limited by the loss of magnitude information. This limitation led to the decision to use I&Q quantization in the high-speed architecture study, as well as the

multimode study [18]. However, phase-only quantization turned out to be extremely useful in the NMSU implementation of pragmatic TCM [28,29], using an existing (off the shelf) binary Viterbi decoder. Pragmatic TCM is discussed in Section 3.2.

2.4 I&Q Quantization

I&Q quantization, that is, individual quantization of the in-phase and quadrature components of the received vector has the important advantage of approaching unquantized performance for sufficiently fine quantization, which is not the case for phase-only quantization. However, a disadvantage is that a much larger number of quantization points must be used, which complicates metric calculation. Also, in order for the magnitude information to be meaningful, the receiver must maintain good automatic gain control. Finally, I&Q quantization limits the range of the received signal vector, so the quantizer must be designed with respect to the expected magnitude of the received vector.

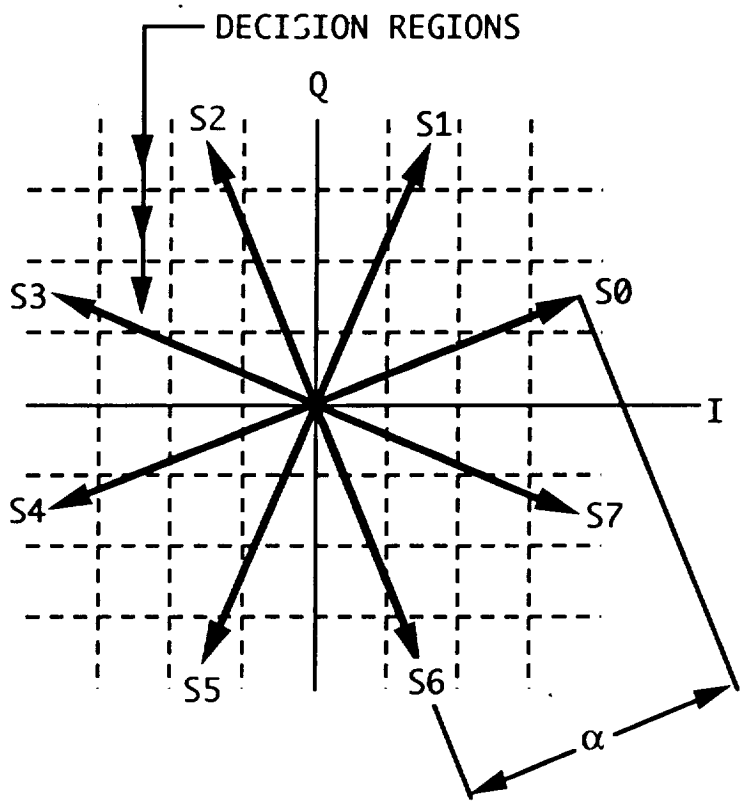

Figure 2.10. I&Q quantization.

To model I&Q quantization, we first assume an 8-PSK constellation as shown in Figure 2.10. This constellation is rotated by 22.5 degrees from that of Figure 1.10. The rotation does not affect the algebraic or analytical properties of the code, but has certain advantages in hardware implementation. We let the I and Q components range from -1 to 1, and then let the signal vectors have length α. We then quantize rectangularly, and symmetrically, so that an equal number of quantization points lie in each quadrant. Because the I and Q

components will be represented as binary numbers in hardware, it is desirable to let the number of quantization values (for I or Q) be a power of 2. From simulations at NMSU [28,29] it is known that 8-level (3-bit) I and Q quantization seriously degrades the performance of pragmatic TCM, whereas the performance of a system using 8-bit I&Q is close to that of an unquantized system. Therefore, for the TCM decoder architecture, we expect to represent the I and Q components of the received vector using now fewer than 4 bits, but no more than eight bits.

For I and Q quantization, an important parameter is the length of the received vector, relative to the boundaries of the rectangular quantization region in the receiver space, denoted as α in Figure 2.10. As is the case for phase-only quantizers, I&Q quantizers should be designed to maximize the random coding bound, R_0. For 4-bit I and Q, R_0 as a function of α is shown in Figure 2.11. R_0 as a function of signal-to-noise ratio is shown in Figure 2.12. At $\frac{E_S}{N_0}$ = 10 dB, R_0 appears to be maximized at approximately α=1.0 and is not very sensitive to α. The insensitivity to α may be due to the fact that at this operating point, most of the probability density of the received signal vector is concentrated within the small number of decision regions adjacent to the source signal vectors, while the remaining decision regions are very

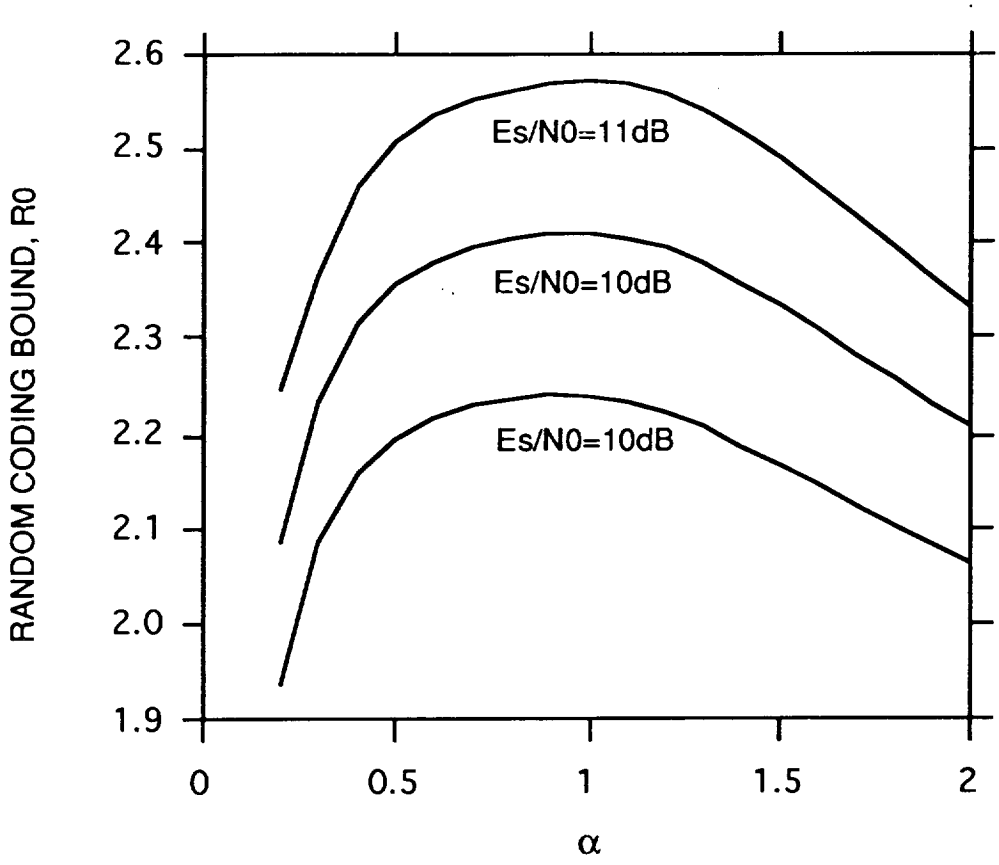

Figure 2.11. Random coding bound for 4-bit I&Q quantization.

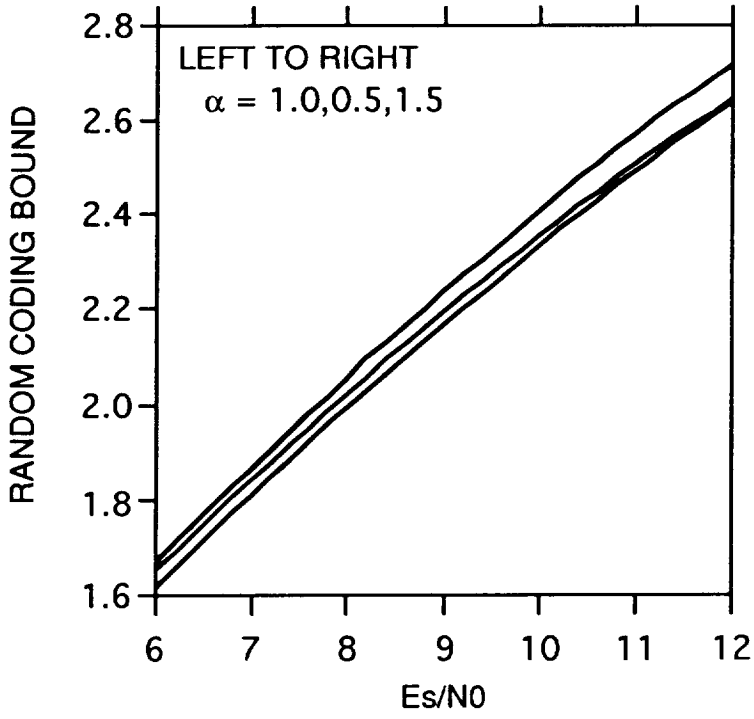

Figure 2.12. Random coding bound for 4-bit I&Q quantization.

under-utilized. This implies that some improvement in performance might be obtainable by the use of non-uniform quantization, with a greater number of decision regions concentrated near the source signal vectors. This, however, is similar to the issue of fine-tuning the decision regions for phase-only quantization, in that the gains to be obtained are probably not worth the added hardware complexity. Fine-tuning of quantization regions can do nothing more than close the gap between quantized and unquantized performance, which for 4-bit quantized I and Q is approximately 0.2dB. Furthermore, uniform

quantization has the advantage of allowing a standard analog-to-digital converter to be used in the demodulator.

2.5 The Log-Likelihood Function

Another issue raised by the quantization of the received signal vector is the calculation of the metrics to be used by the Viterbi decoder. The objective of a good decoder is to select the sequence of encoder output signal vectors which is most likely to be correct, given the sequence of received noisy vectors, that is, to select the encoder output sequence S_m which maximizes $P(S_m|Z)$. If all of the sequences have equal a priori probabilities, and the channel is continuous, then it is equivalent to select the maximum likelihood sequence, that is the sequence S_m which maximizes $p(Z|S_m)$. Here, $P(S_m|Z)$ denotes the conditioned probability of S_m given Z, while $p(Z|S_m)$ denotes the conditioned probability density function of Z given S_m. If the channel is memoryless (that is no signalling interval is affected by any other signalling interval) then

$$p(Z|S_m) = \prod_{i=1}^{L} p(z_i|s_{mi}) \qquad (2.9)$$

where L is the length of the sequence and z_i and s_{mi} are the individual elements of the sequences Z and S_m

respectively. It is equivalent, and computationally more efficient to use log-likelihood functions which may be added, rather than probability density functions, which must be multiplied. Then the decoder would select S_m to maximize

$$-\ln[p(Z|S_m)] = \sum_{i=1}^{L} -\ln[p(z_i|s_{mi})]. \qquad (2.10)$$

If the noise is additive white Gaussian then

$$p(z_i|s_{mi}) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{1}{2\sigma^2} |z_i - s_{mi}|^2\right) \qquad (2.11)$$

where $|z_i - s_{mi}|$ is the Euclidean distance between z_i and s_{mi}. Taking the log-likelihood function leads to the use of Euclidean distance squared as the metric in Viterbi decoding of TCM on the memoryless additive white Gaussian channel.

If the channel is discrete, as it becomes when the quantizer is added to the system, and all S_m have equal a priori probabilities, then maximizing $P(S_m|Z)$ is equivalent to maximizing $P(Z|S_m)$. Here Z denotes the sequence of discrete quantizer outputs, rather than the sequence of continuous signal vectors. The decoder would then select S_m to maximize

$$P(Z|S_m) = \prod_{i=1}^{L} P(z_i|s_{mi}) \qquad (2.12)$$

where the probabilities $P(z_i|s_{mi})$ are the transitional probabilities of the channel. As in the case of the continuous channel, it is preferable to use log-likelihood functions, which may be added, rather than probabilities, which must be multiplied, so the decoder is built to select the sequence S_m which maximizes

$$-\ln[P(Z|S_m)] = \sum_{i=1}^{L} -\ln[P(z_i|s_{mi})]. \qquad (2.13)$$

This condition is equivalently fulfilled by using metrics of the form $a + b \ln[P(z_i|s_{mi})]$ where a and b are arbitrary constants which may be selected to allow the range of metrics to best be represented by a particular hardware design.

2.6 Calculation of Probabilities and Related Parameters

For the discrete channel formed when any form of quantizer is incorporated into a TCM system, the channel capacity, the random coding bound, and the optimal set of metrics must be calculated from the transitional

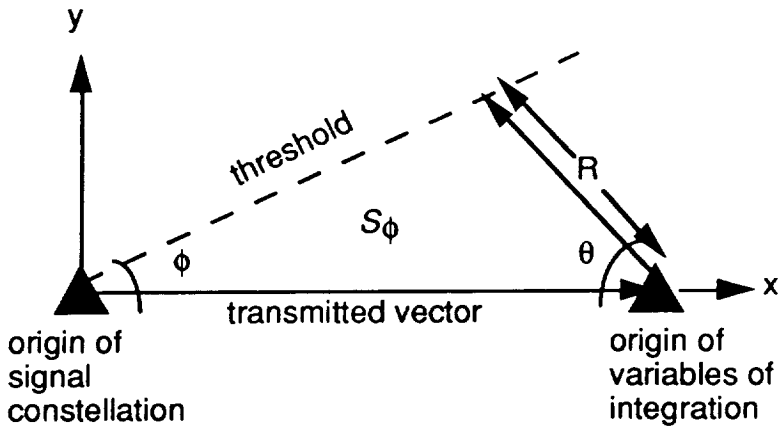

Figure 2.13. Region of integration for sector probability.

probabilities. The transitional probabilities are found by integrating the probability density function of the received vector, given the transmitted vector, over each decision region. For phase-only quantization, the decision regions are angular regions as shown in Figures 2.1a and 2.1b, and also in figures 2.2a, 2.2b, and 2.2c. To find the transitional probabilities for phase-only quantization we first consider the problem of finding the probability, P_ϕ, that the phase of the received signal vector will be removed from the phase of the transmitted signal vector by no more than the angle ϕ, as shown in Figure 2.13. This may be found by integrating the two dimensional Gaussian distribution function over the region S_ϕ giving:

$$P_\phi = \iint_{S_\phi} \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{1}{2\sigma^2}\left[(x-1)^2+y^2\right]\right\} dx\, dy \qquad (2.14)$$

$$\text{where } \sigma^2 = \frac{N_0}{2E_s} \qquad (2.15)$$

The classical approach to this problem is to convert from rectangular to polar coordinates giving:

$$P_\phi = \int_0^\phi \int_0^\infty \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{1}{2\sigma^2}\left[R^2 - 2R\cos\theta + 1\right]\right\} R\, dR\, d\theta$$

$$= \int_0^\phi f(\theta|0)\, d\theta \qquad (2.16)$$

where $f(\theta|0)$ denotes the phase density function, given that a phase of 0 was transmitted. Integrating over R gives:

$$f(\theta) = \frac{1}{2\pi} \exp\left\{-\frac{1}{2\sigma^2}\right\}$$

$$+ \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\sin^2\theta}{2\sigma^2}\right\} \frac{\cos\theta}{\sigma} Q\left(-\frac{\cos\theta}{\sigma}\right) \qquad (2.17)$$

$$\text{where } Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left\{-\frac{y^2}{2\sigma^2}\right\} dy \qquad (2.18)$$

Finding the phase sector probability by this method requires that a double integration be performed

numerically, since no closed form solution for the Q() integral exists.

An alternative method for calculating P_ϕ, which requires only that single integrals be performed numerically is obtained by applying the change of variables:

$$R = \left[(x-1)^2 + y^2 \right]^{\frac{1}{2}} \tag{2.19}$$

$$\theta = \arctan\left(\frac{y}{1 - x}\right) \tag{2.20}$$

This gives the following integral:

$$P_\phi = \iint_{S_\phi} \frac{1}{2\pi\sigma^2} R \exp\left(- \frac{1}{2\sigma^2} R^2\right) dR \, d\theta \tag{2.21}$$

In this expression, the integral with respect to R can be solved in closed form. The limits on R are found as a function of θ:

$$0 \le R \le \left[\frac{\sin \theta}{\tan \phi} + \cos \theta\right]^{-1} \qquad \text{for } 0 \le \theta \le \pi - \phi$$

$$0 \le R \le \infty \qquad \text{for } \pi - \phi \le \theta \le \pi$$

The integral is then broken into two parts and solved giving:

$$P_{\phi} = \frac{1}{2} - \frac{1}{2\pi} \int_{0}^{\pi-\phi} \exp \left\{ - \frac{1}{2\sigma^2} \left[\frac{\sin\theta}{\tan\phi} + \cos\theta \right]^{-2} \right\} d\theta \quad (2.22)$$

where the remaining single integral is then solved numerically. The probability that the received signal vector will have phase between ϕ_0 and ϕ_1 with respect to the transmitted vector may be found from

$$P_{\phi 0,\phi 1} = P[\phi_0 < \phi < \phi_1] = P_{\phi 1} - P_{\phi 0}. \quad (2.23)$$

One problem with this form is that precision problems can arise due to the fact that the difference $P_{\phi 1} - P_{\phi 0}$ can be quite small relative to $P_{\phi 1}$ and $P_{\phi 0}$. This problem may be aleviated by rewriting equation (2.23) in the form:

$$P_{\phi 0,\phi 1} = \frac{1}{2\pi} \int_{0}^{\pi-\phi 1} \exp \left\{ - \frac{1}{2\sigma^2} \left[\frac{\sin\theta}{\tan\phi 1} + \cos\theta \right]^{-2} \right\}$$

$$- \exp \left\{ - \frac{1}{2\sigma^2} \left[\frac{\sin\theta}{\tan\phi 0} + \cos\theta \right]^{-2} \right\} d\theta$$

$$+ \frac{1}{2\pi} \int_{\phi_0}^{\phi_1} \exp\left\{- \frac{1}{2\sigma^2}\left[\frac{\sin\theta}{\tan\phi 0}+\cos\theta\right]^{-2}\right\} d\theta .$$

(2.24)

This form requires more computational time, but yields greater precision when numerical integration methods are applied. A side benefit of equation (2.22) is that it leads directly to an alternative form for the Q() function as follows:

$$Q\left(\frac{1}{\sigma}\right) = 1 - 2P_{\pi/2} = \frac{1}{\pi}\int_0^{\pi/2} \exp\left\{- \frac{1}{2\sigma^2\cos^2\theta}\right\} d\theta \quad (2.25)$$

Substituting x for $\frac{1}{\sigma}$ gives:

$$Q(x) = \frac{1}{\pi}\int_0^{\pi/2} \exp\left\{- \frac{1}{2}\left(\frac{x}{\cos\theta}\right)^2\right\} d\theta \quad (2.26)$$

This form of the integral has finite limits, unlike the standard form.

Because the system is symmetrical, that is, because the probability density function for the received phase given any transmited phase, $f(\theta_j|\theta_i)$, is equal to $f(\theta_j-\theta_i|0)$, formula (2.24) may be used to calculate all of the transitional probabilities required in the analysis of any

phase-only quantization scheme. These may then be used to calculate R_0, C, or log likelihood metrics. The R_0 and C values used in the previous section were found by writing a "C" computer program to calculate the phase transition probabilities by numerical integration of equation (2.24). These were stored in a table and used to calculate C and R_0 from equations (2.6) and (2.5) respectively.

The calculation of the transitional probabilities for I and Q quantization is easier than that for phase-only quantization due to the fact that the I and Q components are independent. That is, the probability that the received sigal vector (I_r,Q_r) will fall within the rectangular region bounded by I_0, I_1, Q_0, Q_1 is given by the product of the probabilities $P[I_0 < I_r < I_1]$ and $P[Q_0 < Q_r < Q_1]$, both of which are found from single integrals:

$$P(z_j|s_i) = \frac{1}{2\pi\sigma^2} \int_{I0}^{I1} \exp\left(- \frac{x^2}{2\sigma^2}\right) dx \int_{Q0}^{Q1} \exp\left(-\frac{y^2}{2\sigma^2}\right) dy \qquad (2.27)$$

For the case of 8-PSK with four bit I&Q quantization, due to the symmetry of the constellation, the calculation of the transitional probabilities requires 32 integrals to be evaluated numerically.

2.7 I and Q Quantization for the TCM Decoder

The preceding analysis using the channel capacity and the random coding bound show that once a sufficiently fine quantization scheme is specified, the exact placement of the decision regions (within reason) can be expected to have little impact on the actual performance of the overall system. For metric quantization, there is no analytical tool which is what channel capacity and random coding bound are for signal space quantization. For the problem at hand, that is, building a machine to decode the 16-state Ungerboeck code for rate 2/3 8-PSK TCM, the desired precision of I, Q, and metrics was determined by computer simulations using BOSS. It was decided that the decoder design presented here should perform at least as well as the pragmatic standard, at a bit error rate of 10^{-5}. The simulations showed that 4-bit quantization of I and Q would not accomplish this, even for unquantizad metrics. It was determined that 5-bit quantization of I and Q, with 7-bit metrics would be essentially equivalent to the performance of unquantized pragmatic TCM. For that reason, those parameters were used for the design.

3. PREVIOUS TCM STUDIES

The high-speed codec design presented in this paper is grounded in experience gained through prior research projects in Trellis-Coded Modulation, including simulations, analytical studies, and hardware projects. These include BOSS simulations of Ungerboeck codes of 4, 8, 16, 64, and 1024 states; BOSS simulations and hardware construction of pragmatic TCM decoders; an analytical study of a multimode codec; and work in bit error spectrum generation, an analytical technique for estimating the performance of various trellis codes.

Since the time that Ungerboeck pioneered TCM in 1982, the performance of trellis codes has been predicted on the basis of the asymptotic coding gain, the probability of the most likely (minimum distance) error event, as described in Chapter 1. In searching for the best codes possible at various constraint lengths, and various signal constellations, Ungerboeck used the minimum distance error event as the criterion of selection. The asymptotic coding gain, ACG is the increase in the minimum distance of a coded system, as compared to an uncoded system carrying the same amount of information per symbol. In the case of rate 2/3 8-PSK, the baseline for comparison is uncoded QPSK, as both carry two bits per symbol. From the QPSK signal

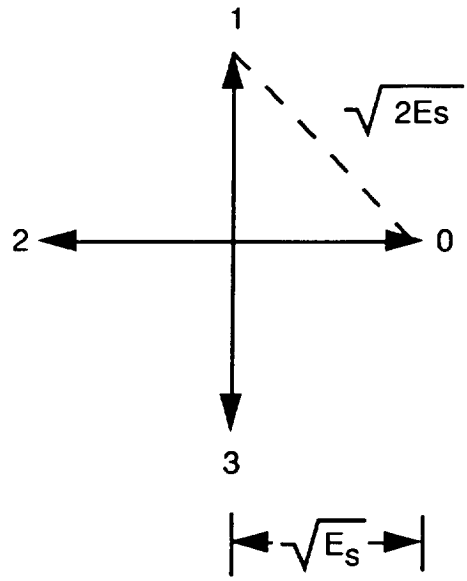

Figure 3.1. QPSK signal constellation.

constellation, Figure 3.1, it can be seen that the minimum distance between signal vectors is $\sqrt{2E_S}$. For the simple 4-state Ungerboeck code, the minimum distance error event is the distance between symbols associated with parallel branches of the trellis, $2\sqrt{E_S}$ or $\sqrt{4E_S}$. Thus the minimum distance between error events for coded 8-PSK represents twice the energy of the minimum distance between uncoded QPSK vectors, and coded 8-PSK is said to have a minimum distance coding gain of 3 dB. The asymptotic coding gain is based not only on the minimum distance but the number of error events at that distance. For 4-state rate 2/3 8-PSK, this turns out to be approximately 3.2 dB.

Asymptotic coding gain is not realized in the actual performance of the decoder, because error events other than

the minimum distance error event contribute significantly to the probability of error. As an example, the true coding gain of the 4-state code, measured at useful threshold bit error rates, falls short of the 3.2 dB asymptotic coding gain, being closer to 1.5 dB, at a bit error rate of 10^{-5}. Non-minimum distance error events usually have very small probabilities but very large numbers, a fact which causes the true coding gain of convolutional codes to be less than the asymptotic gain, and makes analytical calculation of error probabilities of convolutional codes very difficult.

One approach to calculating the probability of error is the union bound. Union bounds, as they apply to binary codes are discussed by Clark and Cain [16], and essentially the same principles apply to binary codes. The union bound approximates the total probability of error as the sum of the probabilities of the individual error events. The union bound will generally overestimate the probability of error, because the probabilities used are the probabilities of pairwise error events, which are not necessarily disjoint. Also, the union bound is not strictly practical, due to the fact that a trellis code possesses an infinite number of error events. For this reason, a union bound calculation is usually based only on the error events which contribute significantly to the overall probability of error. However, the number of error events will still be

quite large, and the problem of finding them is non-trivial.

3.1 BOSS Simulations At NMSU

Due to the inadequacy of the asymptotic error rate prediction, and the difficulty of analytically calculating the error probabilities of trellis codes, simulations are employed as a means of evaluating the performance of trellis codes. Simulations at NMSU have been performed to determine the performance of phase quantized TCM, as discussed in Chapter 2, to determine the performance of codes ranging from 4 to 1024 states for rate 2/3 8-PSK, and to evaluate the performance of pragmatic TCM, using phase quantization as well as quantized I and Q. For rate 2/3 8-PSK Ungerboeck codes of 4, 8, 16 and 64 states were simulated. A 1024-state code was found, using the bit error spectrum technique, then simulated using BOSS.

BOSS stands for "Block Oriented Systems Simulator". BOSS is a commercially available software package which allows simulations of systems to be constructed from previously defined modules, which may be supplied with the system or created by the user. The modules are implemented as FORTRAN subroutines, and the inputs and outputs of the modules correspond to variable types in the FORTRAN

language. Included are vector signals, analogous to arrays, which allow multiple signals of the same type to be "one-lined" on the block diagram, which greatly clarifies the diagram for a system which requires many signals. Generally, modules to perform simpler functions are independently tested and verified, and then used to build up more complex systems. Modules which are defined purely in FORTRAN code, and not constructed out of lower level modules are referred to as primitives. The authors of the BOSS software prefer that users not create their own primitives, but allow for the fact that it may sometimes be necessary. Also, because every BOSS module is effectively a call to a FORTRAN subroutine, which has an overhead in CPU time, the use of specially defined primitives can result in faster simulations. A simulation of a 64-state Ungerboeck decoder built entirely out of basic blocks required nearly a week to run one million symbols, while the equivalent version using in-house primitives required less than 24 hours.

The earlier BOSS simulations were designed to implement specific codes. Later a more general approach was used, implementing the metric calculator, add-compare-select function, and path memory function as in-house BOSS primitives. This means that modules representing these functions appear on the top level block diagram of the BOSS

simulation, but the functions are implemented in FORTRAN code. In these later simulations, a flexible approach was adopted in which the code is defined in terms of two tables: the next state table, which gives the next state of the convolutional encoder as a function of current state and current input, and the next symbol table, which gives an output code symbol to correspond to every state transition represented by in the next state table. The information given by these two tables is sufficient to uniquely define the code. Because the decision unit of a Viterbi decoder looks backwards through the trellis, it is often convenient, and not difficult to convert the next state and next symbol tables into previous state and previous symbol tables.

As an example of a typical Boss simulation for TCM, the top level block diagram for the 1024-state simulation is shown in Figure 3.2. The module 8PSK 1024 STATE DATA generates the test data for the simulation. This module employes a 1024-state convolutional encoder of the kind shown in Figure 3.19, to select 8PSK signal vectors. Gaussian vectors are added to the signal vectors to simulate the effect of noise. The module INTEGER METRICS 8PSK generates metrics for all 8 of the 8-PSK signal vectors. In order to reduce the computing time required

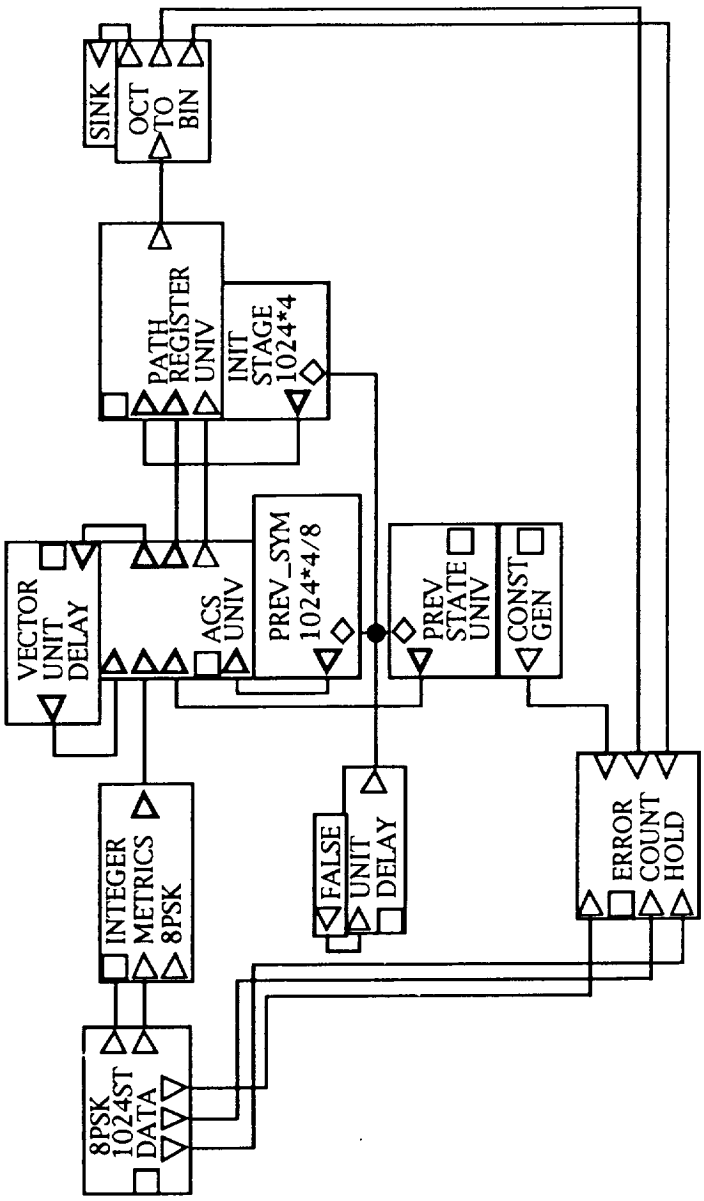

Figure 3.2. BOSS simulation for 1024-state TCM.

for the simulation, integer metrics are used rather than floating point metrics. However, the integer metrics are scaled in such a way that the resolution of the metrics should not be a performance issue. Specifically, the metrics derived from the geometry of the signal set, which range from 0 to 4 (relative to E_S) are multiplied by 255, then the nearest integer is taken.

The ACS UNIV module performs the add-compare-select function, and is implemented as a primitive. This produces lower simulation times than would be obtained by constructing the ACS unit out of smaller modules. The modules PREV_SYM 1024*4/8 and PREV STATE UNIV produce previous state and previous symbol tables for the 1024-state code. These modules can be substituted by other modules to allow the use of different TCM codes. These modules use FORTRAN code to generate the tables, which is done only once, at the beginning of the simulation.

The module PATH REGISTER UNIV is composed of repetitions of a primitive module representing a path stage. The number of repetitions, referred to as replications, gives the decoder its trace-back depth, and is a selectable parameter of the simulation. The module INIT STAGE 1024*4 provides the data to be fed into the first stage of the path memory. This must correspond to the data which drives the encoder to each state, in this case, the two least

significant bits of the binary representation of the state. The data in the path register is represented as integers, the module OCT TO BIN converts the integers to bits. Finally, the data error counter compares the decoded data to the original data, and compiles an error count.

The simulation results for rate 2/3 8-PSK codes are shown in Figure 3.3. This shows the increase in coding gain to be obtained by increasing the complexity of the code. The 64-state Ungerboeck code achieves a coding gain of 3.6 dB over uncoded, as compared with 3.2 dB for pragmatic TCM, which is discussed in the next section. The results of these simulations are presented at the 1991 NAECON conference [30].

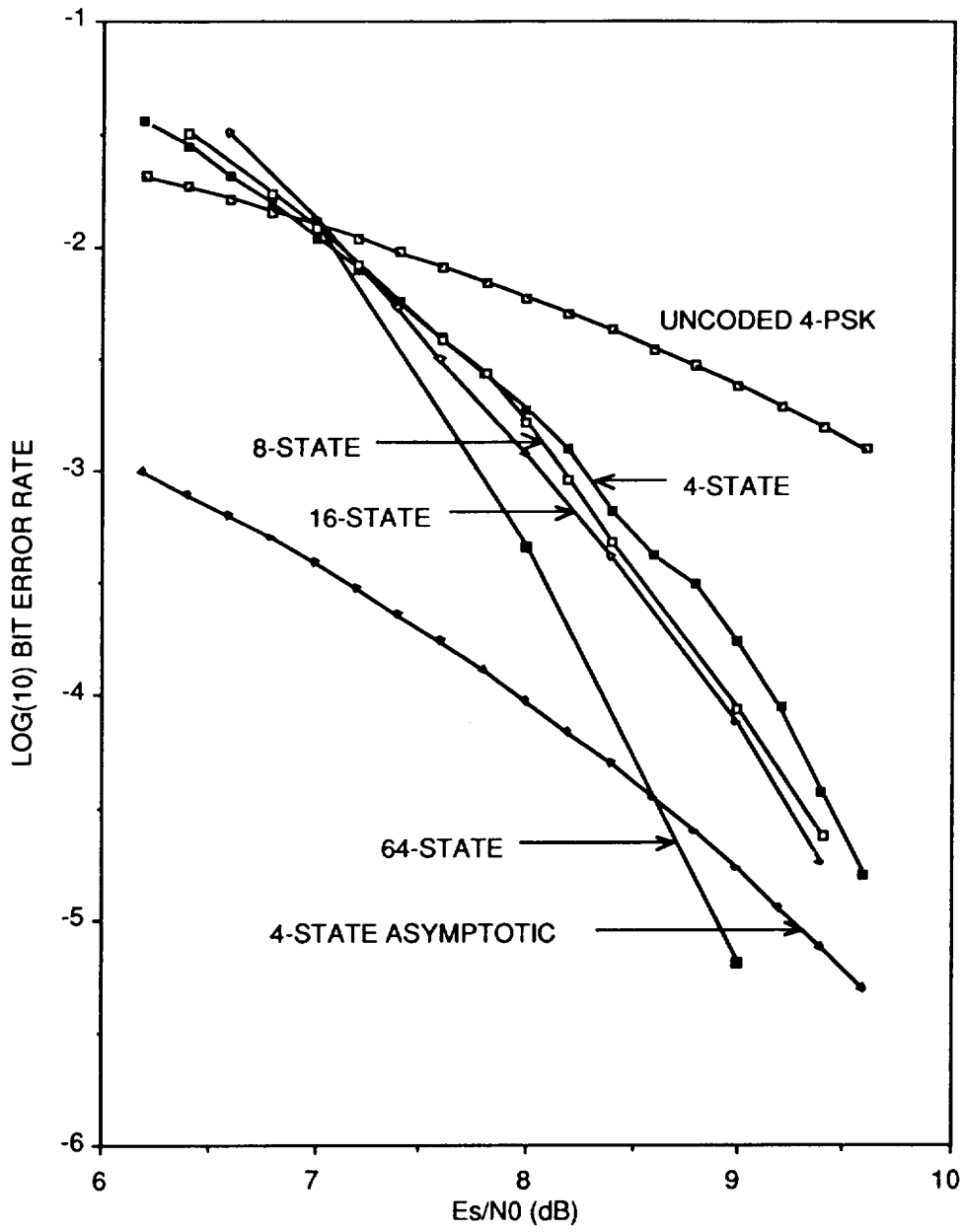

Figure 3.3. Simulation results for rate 2/3 8-PSK codes.

3.2 Pragmatic TCM

In 1988, Viterbi [7] introduced pragmatic TCM, briefly discussed in Chapter 1. Pragmatic TCM is so called because it achieves a considerable simplification in hardware, while suffering only a moderate loss in performance. Pragmatic TCM uses the industry standard 64-state binary convolutional encoder of Figure 1.14 in the TCM system of Figure 1.9. The advantage of doing this lies in the simplicity of the design, and the fact that the same decoder can be used for a variety of modulation formats. Because a reasonably powerful Viterbi decoder is a complex piece of hardware, making one decoder work for a variety of modulation formats is a considerable advantage. One of the possibilities opened by pragmatic TCM is the implementation of non-binary TCM, using a currently marketed Viterbi decoder designed for a binary channel.

After the publication of the concept of pragmatic TCM, the NMSU telemetry lab began work on the design of systems to implement pragmatic TCM for rate 2/3 8-PSK. This was accomplished using a currently available Viterbi decoder, with surrounding circuitry to adapt the binary device to a non-binary channel, as shown in Figure 3.4. While the Viterbi decoder itself represents the most significant investment in hardware, additional parts of the system, are

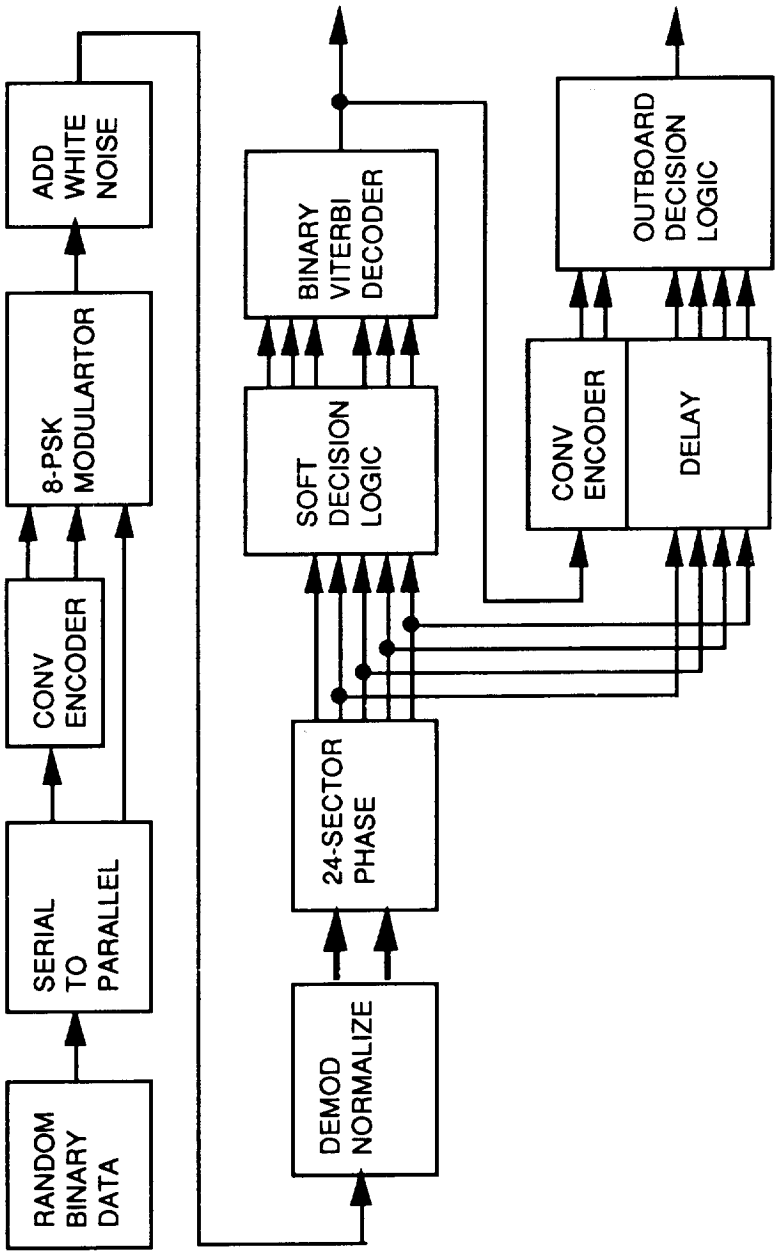

Figure 3.4. Pragmatic TCM system.

also essential to 8-PSK operation. These are the received signal quantizer, the soft decision logic, and the outboard decision logic. The first NMSU experiment in pragmatic TCM was used 24-sector phase quantization, as discussed in Chapter 2. Earlier simulations in TCM established the feasibility of phase-only quantization for use with 8-PSK [8, 9, 10]. From these simulations, it was learned that the performance of 16-sector quantization would be inadequate, that the performance of 24-sector quantization would be acceptable, and that 32-sector quantization would result in only a slight improvement over 24-sector quantization. For this system, the functioning of the outboard decision is the same as it is in the 4-state Ungerboeck code. The use of the decoder's soft decision inputs in a manner appropriate to the phase quantized 8-PSK signal constellation is crucial to the operation of the system.

The first NMSU experiment in pragmatic TCM is shown in Figure 3.4. In this experiment, a computer was used to generate test vectors for the system. Random data is encoded onto a sequence of 8PSK signal vectors in accordance with the pragmatic coding standard. A Gaussian noise vector is added to each signal vector, and then the resulting vector is normalized and represented as a pair of eight-bit numbers. The eight-bit numbers, representing the I and Q components of the noisy vectors, leave the computer

and go to the phase encoder, which generates a five-bit code representing one of 24 phase sectors as shown in Figure 3.6. The five-bit phase code is fed to the soft decision logic, which is explained in Section 3.2.2. The Viterbi decoder recovers only the convolutionally encoded data. Additional logic is necessary to recover the outboard bit, the bit which bypassed the convolutional encoder when the data was encoded. The selection of the outboard bit is effectively a threshold decision between two vectors. The ideal threshold to use depends on the codebits which were modulated onto the signal in the first place. For this reason, the decoded sequence must be reencoded to obtain a maximum likelihood estimate of the codebits. Because the Viterbi decoder introduces a delay into the data, phase information required by the soft decision logic must be delayed to match the decoding delay, as shown in the drawing. The 24-sector phase encoder, the soft decisions, and the outboard decisions are discussed in the following sections.

3.2.1 The 24-sector Phase Quantizer

The 24-sector phase quantizer is illustrated in Figure 3.5. This circuit generates a 5-bit phase code indicating which of 24 phase quantization points is nearest the

received signal vector. The design of the 24-sector phase quantizer is based on three principles:

1) The received vector will be normalized prior to phase sector determination.

2) When the received vector has constant magnitude and varying phase, the component (I or Q) which has the least magnitude is also the component which changes the most in response to a phase change. This component is selected and used to make the phase determination.

3) The use of the absolute value (or magnitude) function on the I and Q components cuts down on the number of comparators necessary to make a phase determination.

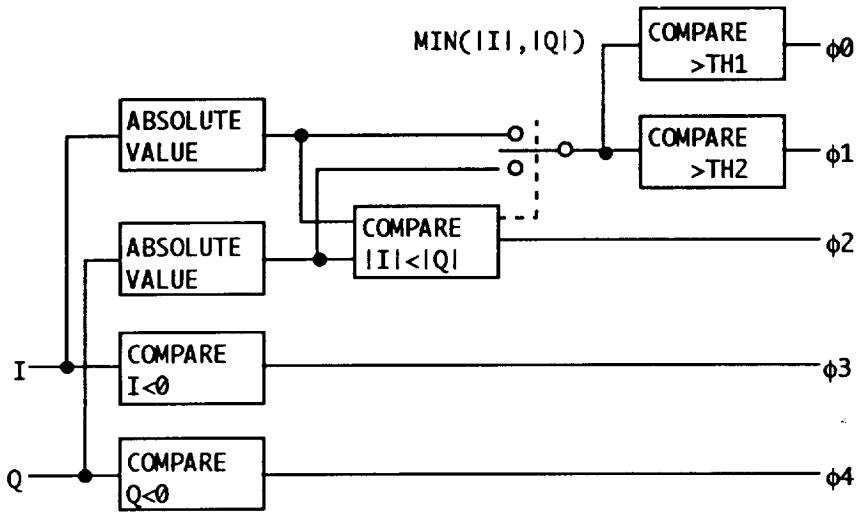

Figure 3.5. 24-sector phase quantizer.

Figure 3.6. Each bit in the code has a specific meaning with respect to the location of the vector, as indicated on the diagram. Note that $\phi 4$ and $\phi 3$ specify the quadrant, while the remaining 3 bits specify the location within the quadrant. Using combinational logic, the phase bits are used to generate the soft decisions for the Viterbi decoder.

3.2.2 Soft Decision Adaptation

The standard Viterbi decoder chip will accept inputs in either of two modes: hard decision, in which the receiver makes a binary determination that the received codebit is either a "0" or a "1" (with no consideration of the relative likelihoods), or soft decision, in which the receiver indicates, on some specified scale, the relative likelihood that the received codebit is a zero or a one. When Viterbi decoding is used with binary signaling, the use of soft decisions can improve performance by as much as 2 dB over hard decisions [16]. Typically, the soft decision is generated by the quantization of an antipodal signal received in the presence of additive white Gaussian noise, as shown in Figure 3.7. Usually, a scale of 0 through 7 (3-bit soft decision) would be used, although decoders which use a scale of 0 through 15 (4-bit soft

decision) are currently available. The decoder uses the soft decisions to calculate a branch metric to associate with each combination of codebits resulting from a state transition of the convolutional encoder. The branch metrics are then used to determine the maximum likelihood sequence.

Ideally, the weight associated with the event that the codebit is a 1, given the received signal R_X, denoted

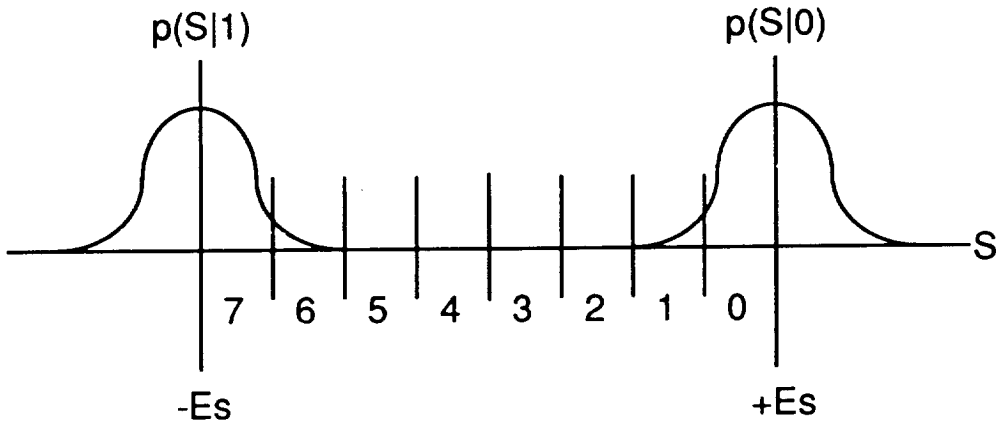

Figure 3.7. Soft decisions for binary channel

$w(c=1|R_X)$, should be proportional to the negative of the log of the probability that the codebit is a 1, $\log[P(c=1|R_X)]$. Likewise, $w(c=0|R_X)$ should be proportional to $\log[P(c=0|R_X)]$. For 3-bit soft decisions, this would lead to:

$$w(c=0|R_X) =$$

$$\text{n.i.}\left[-7\ \frac{\log[P(c=0|R_X)]-\log[P(c=0|R_X=7)]}{\log[P(c=0|R_X=0)-\log[P(c=0|R_X=7)]}\right] \quad (3.1)$$

$$w(c=1|R_X) =$$

$$\text{n.i.}\left[-7\ \frac{\log[P(c=1|R_X)]-\log[P(c=1|R_X=7)]}{\log[P(c=1|R_X=0)-\log[P(c=1|R_X=7)]}\right] \quad (3.2)$$

Here n.i. denotes the nearest integer to the quantity in brackets. Both of these conditions could be satisfied simultaneously by a decoder which accepts two weights for each codebit, one representing the strength of a 1, the other representing the strength of a zero. Soft decision decoders commonly in use do not allow this, as they accept one input representing the strength of a 1, that is $w(c=1|R_X)$, while the weight attached to a zero is implicitly $w(c=0|R_X) = 7 - w(c=1|R_X)$. While this additional constraint precludes the exact simultaneous solution of (3.1) and (3.2), it is known that the Viterbi algorithm is robust, and relatively insensitive to the exact selection of weights [16]. Therefore, the manufacturers of Viterbi decoders resort to the simple expedient of letting the soft decision represent the coordinate of the received signal vector on an integer scale of 0 to 7, that is $w(c=1|R_X)$ is simply R_X. This

technique is effective in that it achieves the expected coding gain over hard decisions.

The preceding discussion pertains to soft decision Viterbi decoders as they are used currently, that is on a binary or quadrature channel. Assuming that the channel is memoryless, codebits transmitted by binary signaling are independent. When quadrature signaling is used, two codebits are transmitted per signal, with each orthogonal component of the two dimensional signal representing a single codebit, so all codebits in quadrature signaling are likewise independent. This means that the probabilities of symbols, each consisting of a pair of codebits, are given by $P(c_1c_0)=P(c_1)P(c_0)$ and $\log[p(c_1c_0)] = \log[P(c_1)] + \log[P(c_0)]$. Since the weights are based on logarithms of probabilities, it is appropriate to let the weight associated with a symbol be the sum of the weights associated with the individual codebits.

Unlike binary or quadrature signaling, in 8-PSK signaling it is not the case that the codebits are independent. Therefore the optimal weight to assign to a pair of codebits is not simply the sum of the codebit weights. However, a decoder designed for use on a binary channel will take the symbol weight to be the sum of the weights given for a pair of code-bits. Therefore, in adapting a binary decoder for use on an 8-PSK channel it is

necessary to assign the soft decision codebit weights not only so that each individual codebit weight reflects the likelihood of that particular codebit, but also so that the sum of the weights assigned to a pair of codebits sums to

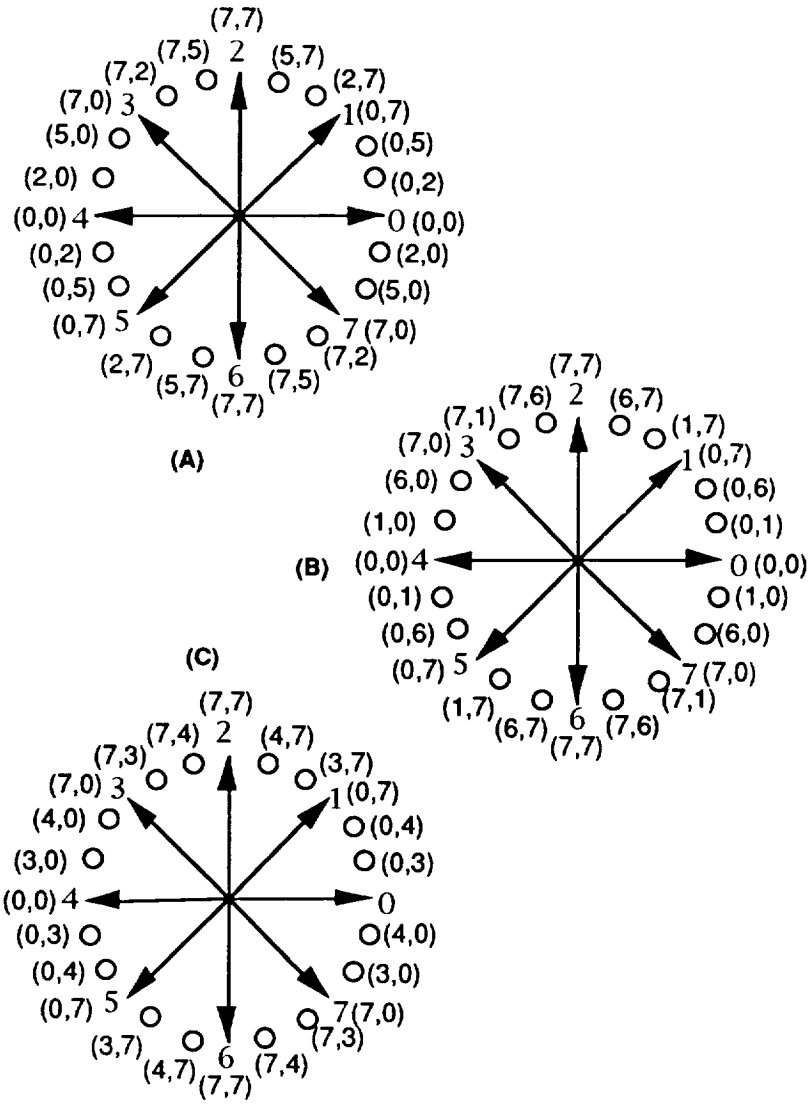

Figure 3.8. Soft decision assignments for 24-sector pragmatic TCM.

an appropriate weight for the associated symbol, or as nearly so as possible.

For the 24-sector 8-PSK pragmatic TCM system, soft decision weights were assigned according to the following principles:

1. As required by the decoder, the soft decision weight indicates the relative likelihood of a zero or a one, with a weight of zero indicating the greatest likelihood of a binary zero, and a weight of seven indicating the greatest likelihood of a binary one.

2. The soft decision assignments are made in a way which reflects the symmetry of the signal constellation.

The constellations of Figure 3.8 all conform to these principles, however, configuration (a) was empirically found to be the best.

The soft decision assignments of Figure 3.8a result from a least square solution to the problem of generating log-likelihood symbol metrics from the soft decision inputs. For brevity, let $w(c_0=0)$ be denoted w_0, then $w(c_0=1)$ may be written as $7 - w_0$. Likewise, let $w(c_1=0)$ be written w_1, and $w(c_1=1)$ be written $7 - w_1$. In this case, w_0 and w_1 correspond to the pair of weights given the soft

decision decoder. Furthermore, let the four codebits be denoted w_{00}, w_{01}, w_{10}, w_{11}, for $w(c_0=0,c_1=0)$, $w(c_0=0,c_1=1)$, $w(c_0=1,c_1=0)$, and $w(c_0=1,c_1=1)$, respectively. The decoder then assumes that the correct symbol weights are given by:

$$w_{00} = w_0 + w_1 \tag{3.3}$$

$$w_{01} = w_0 + (7 - w_1) \tag{3.4}$$

$$w_{10} = (7-w_0) + w_1 \tag{3.5}$$

$$w_{11} = (7-w_0) + (7-w_1) = 14 - w_0 - w_1 \tag{3.6}$$

subject to $0 \leq w_0 \leq 7$, $0 \leq w_1 \leq 7$.

Clearly, it is not possible to generate weights which are optimal, in the sense that they represent log-likelihoods, and which also satisfy the constraint of the above system of equations. The objective is to obtain a set of weights which fit as closely as possible, in the least squared error sense. Let w_{00}', w_{01}', w_{10}', and w_{11}' be the optimal weights, as opposed to the weights calculated by the decoder from the soft decisions, using equations (3.3) through (3.6). The optimal symbol metrics are proportional to the logarithms of the probabilities and also extend over the maximum range made possible by the decoders soft decision mechanism. Clearly, the maximum symbol metric is 14, obtained when both soft decision

inputs are equal to 7. Therefore, the weight of 14 should correspond to the log of the smallest probability of code symbol over all code symbols c1c0 and quantizer outputs z. The minimum soft decision is 0. This gives us:

$$w_{c1c0}' = 14 \frac{-\log[p(c1c0)|z]-\min[-\log[p(c1c0)|z]]}{\max[-\log[p(c1c0)|z]]-\min[-\log[p(c1c0)|z]]} \tag{3.7}$$

where max and min are for all possible values of c1, c0 and z.

The system (3.3) through (3.6) may be optimized separately for each quantizer output z. Because there are four equations and four unknowns a solution such that the implemented metric is equal to the optimal metric, i.e., $w_{c1c0} = w_{c1c0}'$ for all c1 and c0 is not possible. However a least squared error fit can be found to minimize

$$W = \sum_{c1c0} (w_{c1c0}'-w_{c1c0})^2 \tag{3.8}$$

where $0 < w_0 < 7$ and $0 < w_1 < 7$. Since (3.8) is a quadratic equation, W may be minimized by setting:

$$\frac{\delta \bar{w}}{\delta w_1} = \frac{\delta w}{\delta w_0} = 0 \qquad (3.9)$$

which results in:

$$w_1 = \frac{1}{4}(w_{00}' + w_{01}' - w_{10}' - w_{11}' + 14) \qquad (3.10)$$

$$w_0 = \frac{1}{4}(w_{00}' - w_{01}' + w_{10}' - w_{11}' + 14) \qquad (3.11)$$

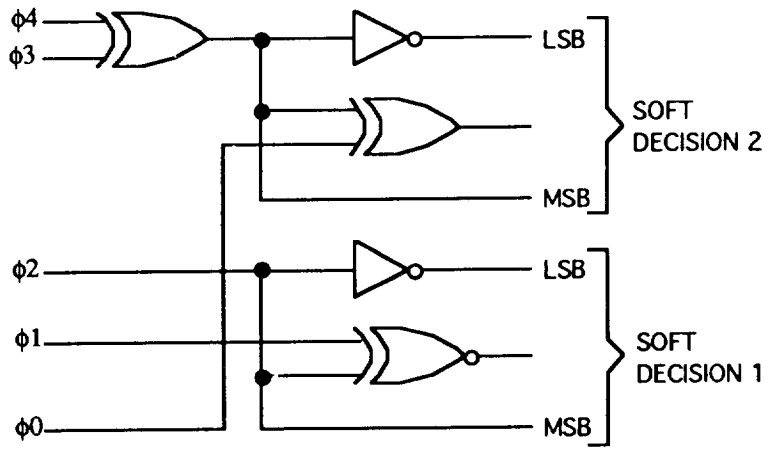

Figure 3.9. Soft decision logic.

If (3.10) or (3.11), give a value of w_0 or w_1 outside the range 0 through 7, then the soft decision to the decoder is hard limited to this range, otherwise the soft decision inputs are taken to be the nearest integers to the solution of (3.10) and (3.11). The values of w00' through w11' are calculated from (3.7), where the symbol

probabilities are calculated using the sector probabilities and Baye's rule, and the sector probabilities are calculated using the procedure described in Chapter 2. This procedure yields the same weights for E_S/N_0 ranging from 5dB to 12dB, i.e., the weights appear to be insensitive to signal-to-noise ratio. This result pertains to the use of 3-bit weights. Of course, if sufficiently fine resolution were used for the weights, there is no doubt small differences would appear over the range of useful SNR's. The soft decisions yielded are the ones of Figure 3.8a, which were also empirically found to be the best. Figure 3.9 illustrates the soft decision logic, a circuit which generates the soft decisions of Figure 3.8a, from the phase code of Figure 3.6.

3.2.3 Outboard Decision Logic

The outboard decision logic makes the outboard bit determination using the information bits from the 24-sector phase quantizer. This is an alternative to building another threshold detector for this purpose. The design of the outboard decision logic, shown in Figure 3.10, is based on two principles:

1) The optimal outboard decision threshold to use depends on the original codebits, g_1 and g_0. For example, if g_1g_0=00, the decision is between vectors 000 and 001, and the optimal threshold is the line formed by the vectors 110 and 111 (see Figure 3.11). Likewise, if g_1g_0 = 01, then the optimal threshold is the line formed by the vectors 100 and 101.

2) When the information from the 24-sector phase detector is used, the combination of ϕ_4 through ϕ_1 which determines the outboard bit depends on the optimal threshold (as determined by g_1 and g_0) and on the position

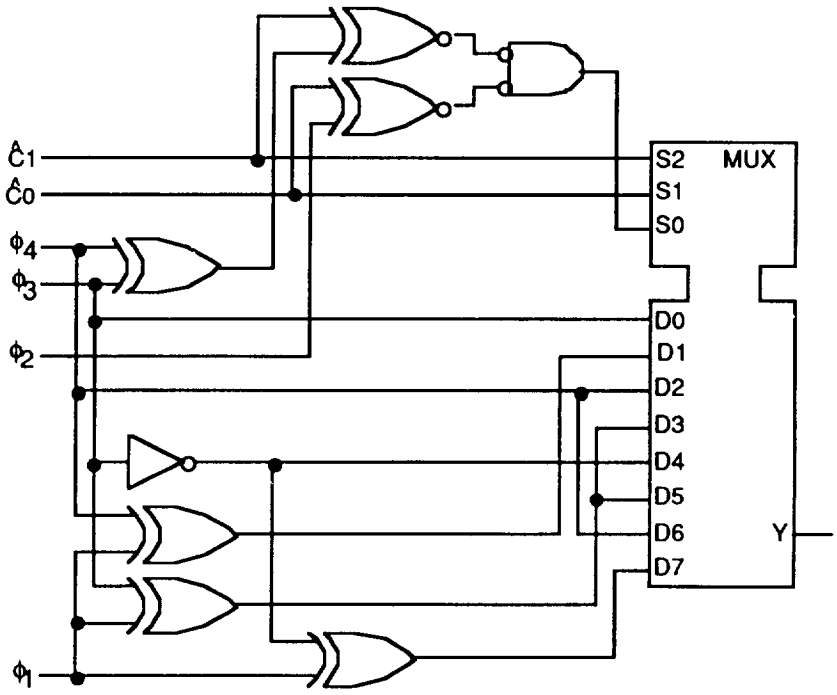

Figure 3.10. Outboard decision logic.

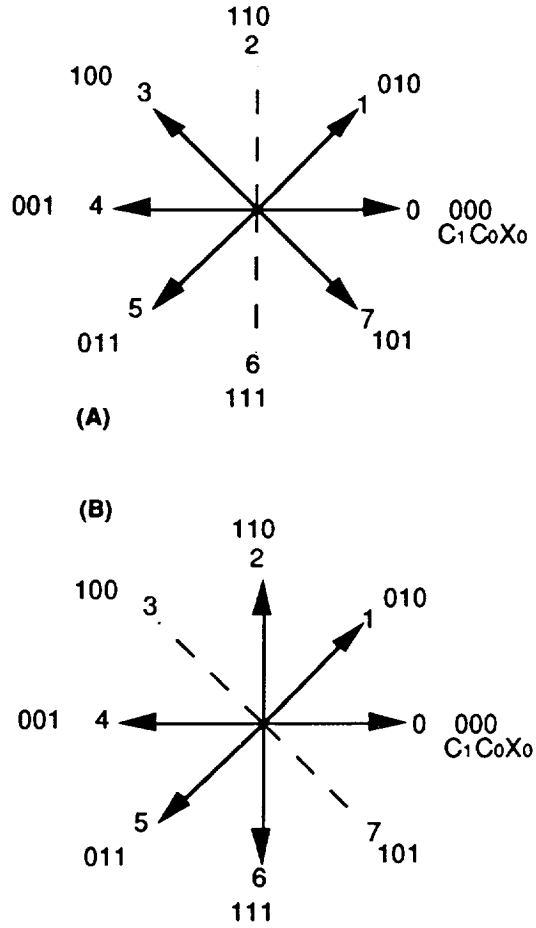

Figure 3.11. Threshold for outboard decision a) c1c0=00, b) c1c0=01.

of the received vector with respect to the threshold. For example, if g_1g_0 = 00, and the received vector is within 4 quantization points of the vector 000 or 001, the decision is between the right half plane and the left half plane and the outboard bit is equivalent to ϕ_3. If the received vector is within one quantization point of the threshold

(110 or 111), then the left plane, right plane decision will not work and the combination $\phi_1 \oplus \phi_4$ is used instead.

It turns out that for all four values of g_1g_0, there is a combination which will work for the case where the received vector is removed from the threshold by more than one quantization point, and another which works for the case where the received vector is within one point of the threshold. The purpose of the 8x1 multiplexer (MUX) is to select the appropriate combination for the given case.

To accomplish this, $\hat{x}_1$, the output from the Viterbi decoder is re-encoded to generate $\hat{g}_1$ and $\hat{g}_0$, maximum likelihood of the original codebits, g_1 and g_0, based on the results of maximum likelihood decoding. The bits $\hat{y}_1$ and $\hat{y}_0$ are estimates of g1 and g0 based on the location of the received vector. In making the outboard decision, $\hat{g}_1$ and $\hat{g}_0$ are compared to $\hat{y}_1$ and $\hat{y}_0$, respectively using the exclusive or gates at the top of the diagram. If $\hat{y}_1\hat{y}_0$ differs from $\hat{g}_1\hat{g}_0$ in *both* bits, it means that the received vector is one or fewer quantization points away from the threshold, and this is indicated by r = 1. The bits $\hat{g}_1$, $\hat{g}_0$, and r cause the MUX to select the logical combination of phase code bits which yields the correct decision. In each of the eight cases, the logical combination to use was determined by inspection.

The bits $\hat{y}_1$ and $\hat{y}_0$ are determined from the phase information bits. Recall that ϕ_4=1 means that I < 0, whereas ϕ_3 = 1 means Q < 0. Therefore, ϕ_4 and ϕ_3 together specify the quadrant of the received vector space. In the upper right and lower left quadrant, g1 = 0, otherwise, g1 = 1. Therefore, $\hat{y}_1 = \phi_4 \oplus \phi_3$. The bit ϕ_2 changes whenever a 45 line is crossed, therefore $\hat{y}_0 = \phi_2$.

3.2.4 Performance of Pragmatic TCM

The system shown in Figure 3.4 was constructed in hardware as well as simulated in BOSS. The performance of the hardware and of the simulation are shown in Figure 3.12. For comparison, the asymptotic error rate for 8-PSK and the theoretical error rate for the 64-state Ungerboeck code are also included. The asymptotic error rate for pragmatic TCM is calculated as $Q\left(\sqrt{\frac{2E_S}{N_0}}\right)$. The error rate for the 64-state Ungerboeck code was calculated from the bit error spectrum technique. At a bit error rate of 10^{-5}, the coding gain of this system is approximately 2.6dB, demonstrating the practicality of pragmatic TCM for 8-PSK. As was discussed in Section 3.2.2, the soft decision assignments of Figure 3.8a were found to be superior to those of Figures 3.8b and 3.8c. The comparison is shown in Figure 3.11. The results of the simulation were presented

at the International Phoenix Conference on Computers [28] in Communications, and the results of the hardware implementation were presented at ICC/Supercomm 92 [29].

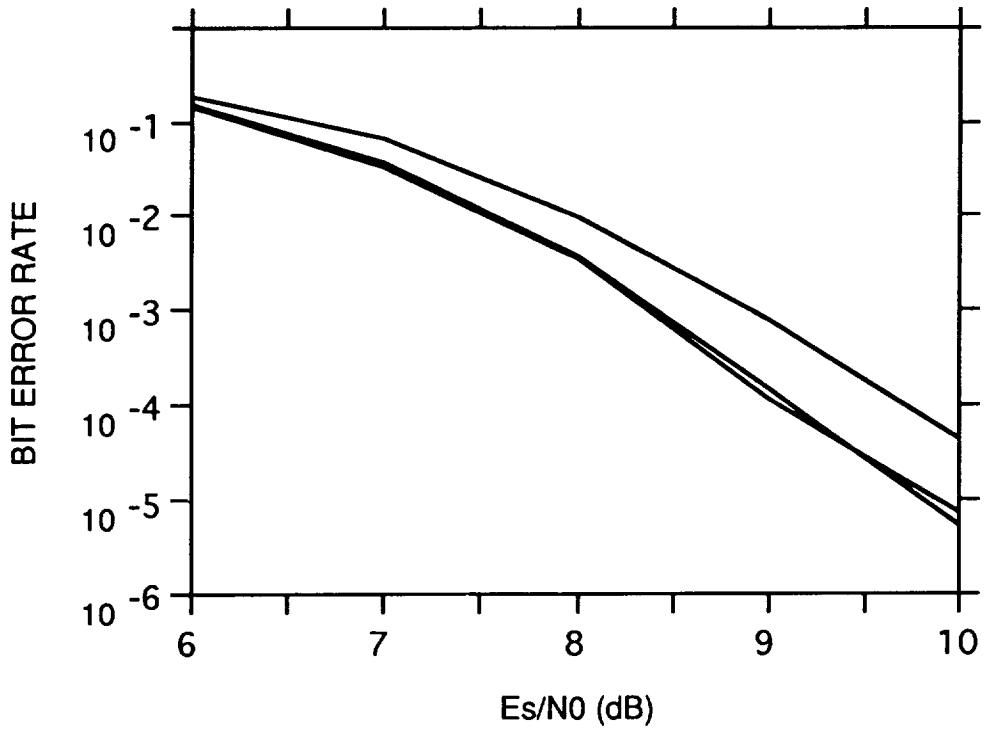

Figure 3.12. Performance of 24-sector 8-PSK pragmatic TCM using different weights.

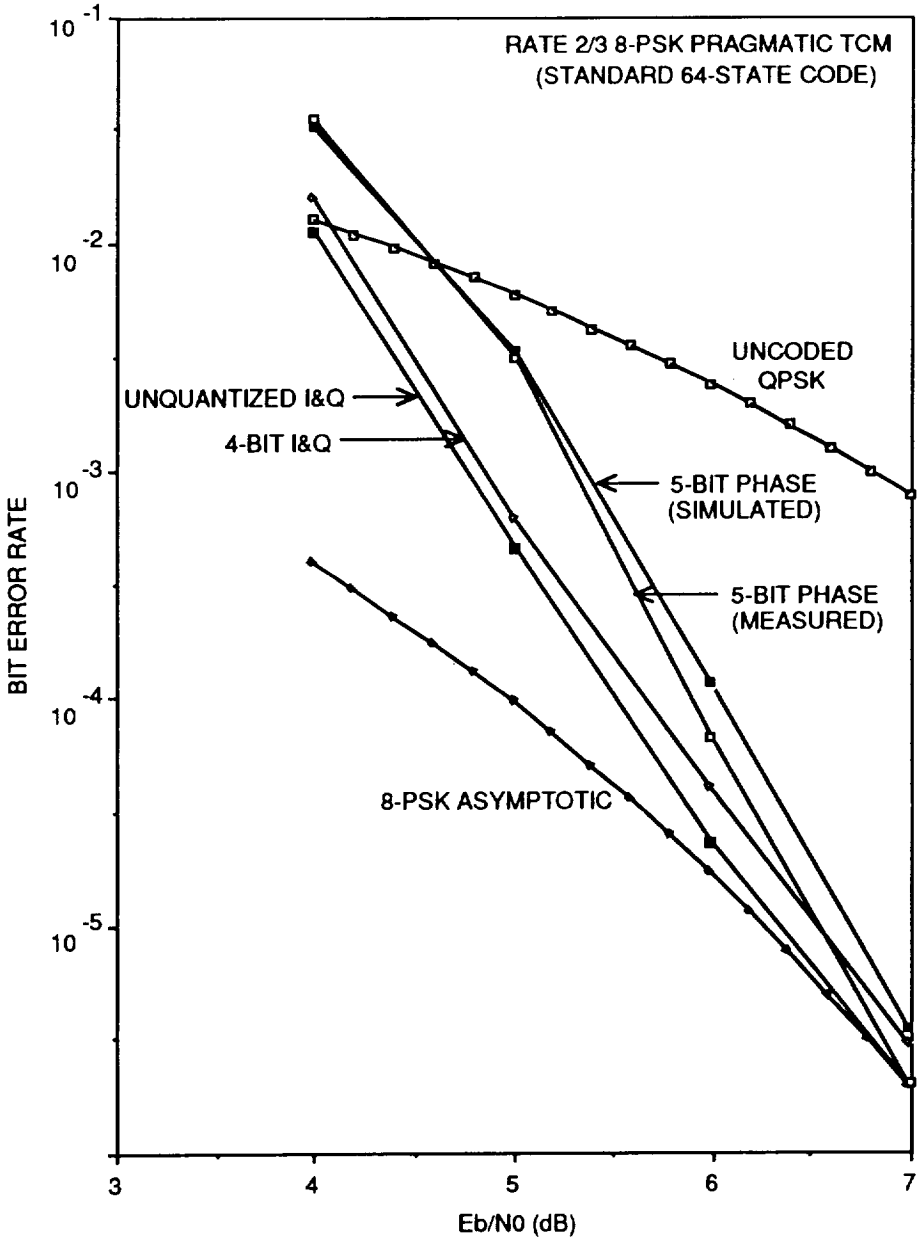

Figure 3.13. Performance of pragmatic TCM.

3.3. Multimode TCM

As mentioned by Viterbi [7], one of the advantages of pragmatic TCM is that it allows the same Viterbi decoder to be used for a variety of modulation formats. Given the interest in constant envelope signaling for satellite communications, the NMSU telemetry lab investigated the design considerations for a modem/codec to operate for BPSK, QPSK 8-PSK, or 16-PSK [18]. This paper addressed symbol synchronizer and phase locked loop considerations, as well as the codec considerations. As part of the design considerations for the codec, the performance of I and Q quantization for pragmatic TCM was investigated. This design assumed the availability of a Viterbi decoder with 4-bit branch metric inputs. At the time, the only commercially available decoder with this feature was the STEL-2020 by Stanford Telecommunications. Unfortunately, this decoder has since been discontinued. However, the approach of finding an adaptation of the soft decision inputs, as was done for phase quantized pragmatic TCM in Section 3.2, is still feasible. The system described in the multimode study used 4-bit quantization of the I and Q components, and then using a read only memory, assigned a 4-bit metric to each decision region.

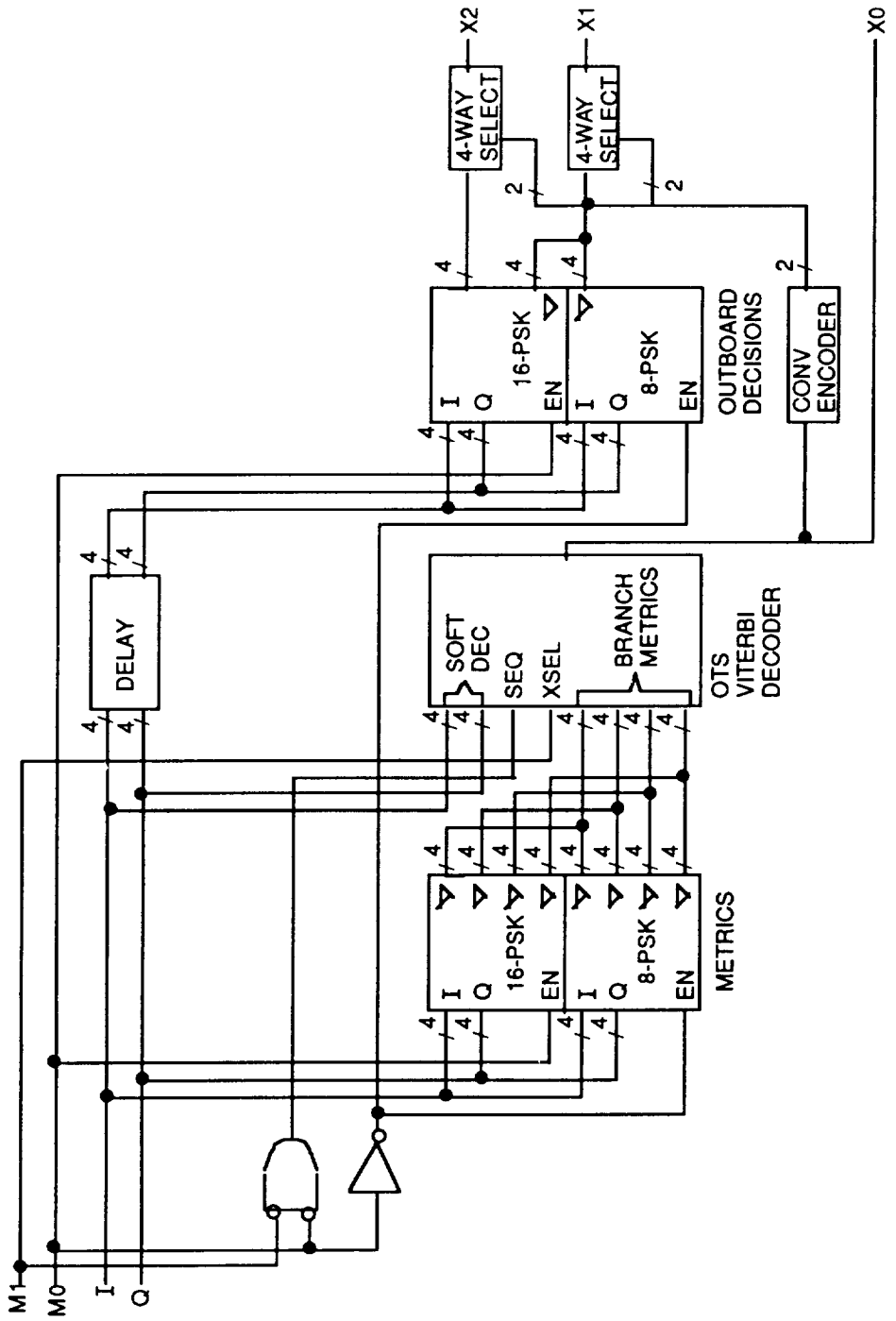

Figure 3.14. Multimode Decoder.

The multimode decoder is shown in Figure 3.14. Pragmatic TCM in BPSK, QPSK, 8-PSK, or 16-PSK is transmitted over an additive white Gaussian noise channel, and received by aquantizer with 16-level quantized outputs for the I and Q components. For BPSK and QPSK operation, which the Viterbi decoder chip was initially designed for, the I and Q components are fed directly to the soft decision inputs. In the 8-PSK and 16-PSK modes, the I and Q components are used to address a ROM, which provides branch metric inputs. Additional ROM's are used to provide the outboard decisions for 8-PSK and 16-PSK. The inputs M1 and M0 select the mode of operation: 00=BPSK, 01=QPSK, 10=8-PSK, and 11=16-PSK. The mode select units select the soft decision or branch metric mode of the Viterbi decoder, and also enable the ROM's which provide metrics and soft decisions for 8-PSK and 16-PSK. If the BPSK or QPSK mode is selected, XSEL, the external branch metric select on the Viterbi decoder is non-asserted, meaning that the decoder will use soft decisions. If the BPSK mode is selected, SEQ (sequence) is asserted, meaning that the two code bits are received in series, but in all other modes, SEQ is non-asserted, and the inputs to the decoder are accepted in parallel. As in the case of the phase quantized pragmatic system, the outboard decision requires the decoded sequence, as well as information of the location of the

received vector, and the location of the vector must be delayed to match the delay introduced by the Viterbi decoder.

The branch metrics and outboard decision metrics are obtained from ROM's. Each ROM has 256 addresses, resulting from the use of four bits of I and four bits of Q. The ROM giving the metric must be 16 bits wide, to provide four 4-bit metrics. Separate metrics must be provided for the 8-PSK and 16-PSK modes of operation, since the optimal metrics are not the same for both cases. The outboard decision table must have a width of 8 bits for 16 PSK and four bits for 8-PSK. This is because an outboard decision consists of one bit for 8-PSK and two bits for 16-PSK, and in each mode, four outboard decisions are made, for the four possible combinations of code bits. When the codebits are determined, by reencoding the decoded sequence, the system selects the appropriate outboard decision.

The bit error rate performance for the multimode system in 8-PSK and 16-PSK modes is shown in Figure 3.15. The performance of BPSK and QPSK is already known from the manufacturers data sheet. The performance results shown in the Figure reflect the effect of using 4-bit numbers for the I and Q components, as well as for the metrics. The multimode system, consisting of a standard Viterbi decoder,

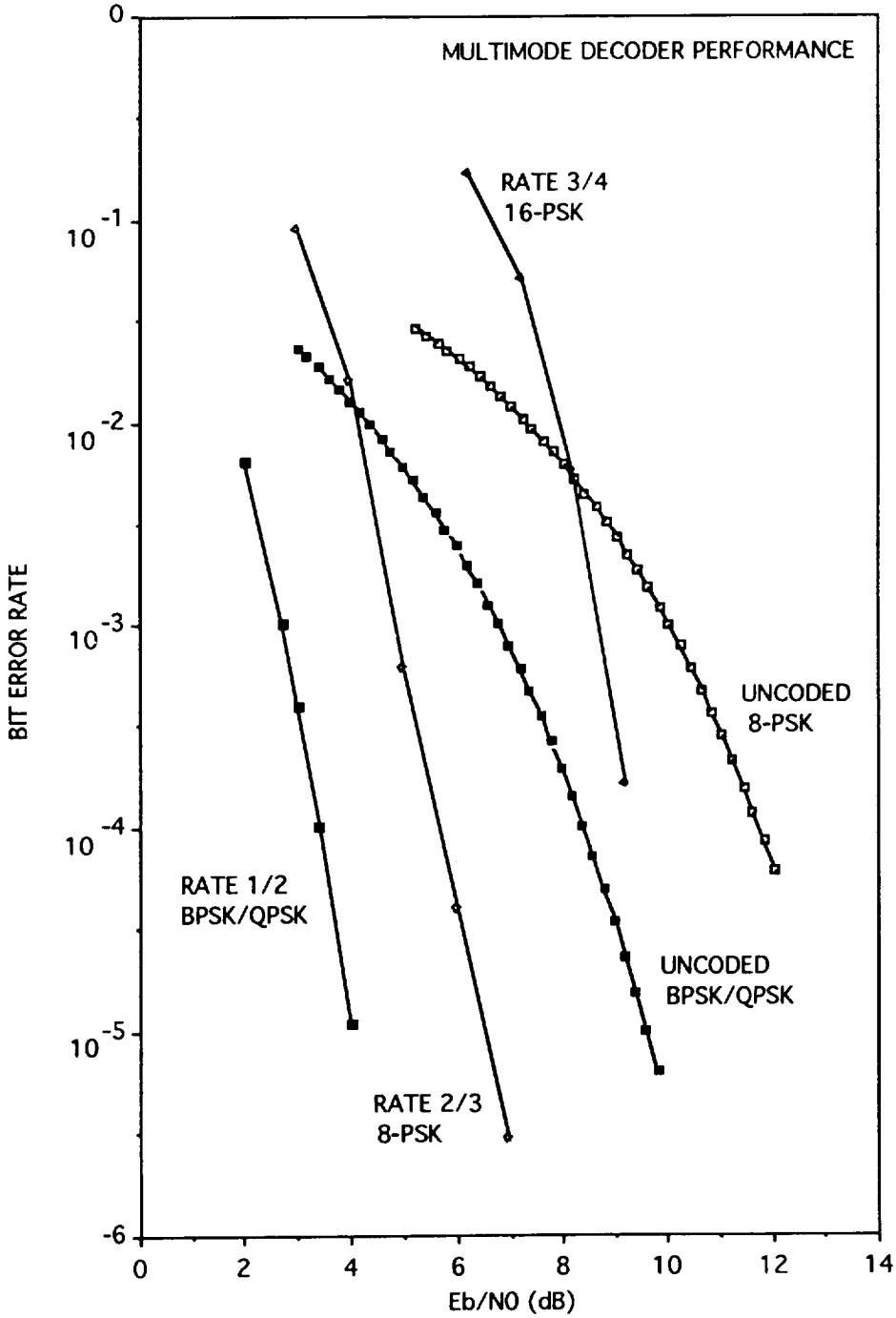

Figure 3.15. Multimode performance.

and a small amount of additional hardware provides meaningful coding gain in all modes of operation. At a bit error rate of 10^{-4}, coded 16-PSK gains about 2.2dB over uncoded 8-PSK. At a bit error rate of 10^{-5}, coded 8-PSK gains essentially 3dB over uncoded QPSK. Thus it can be seen that the pragmatic standard allows the design of a decoder which is effective both in terms of hardware minimization and performance.

3.4 Bit Error Spectrum

The bit error spectrum technique is an analytical method for predicting the error rates of trellis codes, motivated by the long run times required for simulation of the more complex trellis codes. Bit error spectrum methods have also been developed by Rouanne and Costello [13], and also by Zehavi and Wolf [31]. In this work the predominant emphasis is on 8-PSK trellis codes; however, the technique is also applicable to trellis codes of other signal constellations, such as Multi-h. In fact, the first step in the algorithm is to define a table which lists metrics for all of the symbols of the signal set, with respect to the zero symbol. In this way, the bit error spectrum algorithm is made as independent as possible of the geometry of the signal set. To the bit error spectrum

program, the signal set is simply a set of integers, each of which is associated with a floating point metric, or in some cases, as will be explained, more than one metric. To apply the technique to arbitrary constellations, the signal set must be reduced to a vector representation using a technique such as the Gram-Schmidt procedure, so that metrics between the symbols can be calculated. This can in fact be done for any set of M signal vectors for which an M by M table of inter-symbol correlations can be calculated.

The bit error spectrum technique is based on the important algebraeic properties of convolutional codes. The encoder is a finite-state machine, with outputs assigned to the transitions between states. The purpose of the decoder is to find the maximum likelihood state history of the decoder, based on the received sequence of code symbols. An error event is defined as the selection of an incorrect path which diverges from the correct path and then reconverges. The probability of an error event is directly dependent on the vector distance between the correct path and the error path.

A common error rate estimate is the asymptotic error rate, the probability of the most likely error event. The asymptotic error rate is not an accurate estimate because the most likely error event is, of course, not the only error event, and the numbers of less likely error events

can be very large, even as their numbers are very small. At high signal-to-noise ratios, the probabilities of the less likely error events diminish, and the true error rate approaches the asymptotic error rate in the limit. For more complex codes, the minimum distance error path is not necessarily a path of the minimum number of branches, which complicates the problem of finding the most significant error events. Typically simulations are accurate at low signal-to-noise ratios, since shorter run times are sufficient to generate a statistically representative number of errors. The bit error spectrum technique is intended to bridge the gap between low signal-to-noise ratios, where simulations are accurate, and high signal-to-noise ratios, where the asymptotic curve is accurate.

The bit error spectrum technique is a means of calculating higher grade asymptotic error rates. That is, instead of calculating an error rate based on the single most significant error event, an error rate can be calculated from the sum of the N most significant error events:

$$P_b \leq \sum_{i=1}^{N} B(E_i)P(E_i) \qquad (3.12)$$

where P_b is the probability of bit error

E_i is the ith error event

$B(E_i)$ is the bit error weight associated with error event E_i

$P(E_i)$ is the probability of error event E_i.

The bit error weight of an error event is the number of data bits which will be missed if the error event occurs, divided by the number of data bits associated with each stage of the trellis. This is the error event's contribution to the overall bit error rate. Because path selection occurs at each stage of the trellis, each stage of the trellis is regarded as an opportunity for an error event to occur. Because the probability of an error event depends only on the metric of the error path, the previously given summation can be regrouped and written as:

$$Pb \leq \sum_{j=1}^{J} P_j \sum_{k=1}^{K} B(E_{jk}) \qquad (3.13)$$

where P_j is the probability of an error path of a specific metric, which may occur for more than one path

E_{jk} is the kth error event with probability P_j

B is the bit error weight.

Assuming the noise to be additive white Gaussian, the probability of an error event is calculated from the Q() function giving:

$$Pb \leq \sum_{j=1}^{J} Q\left(\frac{m_j}{2\sigma}\right) \sum_{k=1}^{K} B(E_{jk}) \qquad (3.14)$$

where m_j is the path metric, and $\sigma = \frac{N_0}{2}$.

This form of the equation is the most efficient form for calculating the bit error probability, since the summation in k is a function of the code itself, the total bit error weight associated with a particular metric j. It is these weights which are generated by the bit error spectrum technique.

3.4.1 The Generating Function

The bit error spectrum technique is structurally similar to the generating function, a classical approach to the analysis of trellis codes. Because the concepts involved in the generating function are helpful in understanding the bit error spectrum technique, a brief discussion of the generating function will be presented

before resuming the discussion of the bit error spectrum. The generating function yields a sum of products expression which represents all of the paths leading to a specific node of the trellis as follows:

$$X_{node} = a_1 W^{m1} + a_2 W^{m2} + \ldots \qquad (3.15)$$

where m_i is a metric with respect to the all zeroes path, a_i is the number of paths of metric m_i, and W is simply a base of the exponent. This is the simplest form, typically generating functions also include weighting terms for the number of branches associated with a path, and the number of non-zero data bits associated with a path. The use of generating functions dates back to the development of binary convolutional codes, with m_i representing Hamming distances [1]. Zehavi and Wolf [31] had the insight that the generating function can also be applied to Euclidean Distance codes with m_i being a real number rather than strictly an integer. Due to the fact that there is an infinite number of paths to each node, the node equation, X_{node} is an infinite series, but as with other infinite summations, it may be possible to find a closed form expression.

The generating function is derived from the node equations, which are obtained from the state diagram of the

encoder. The state diagram for a simple 4-state code is shown in Figure 3.16. An auxiliary fifth state is added, to provide separate starting and finishing states for error paths, all of which diverge from and rejoin the all zeroes path. Binary convolutional codes are linear, which means

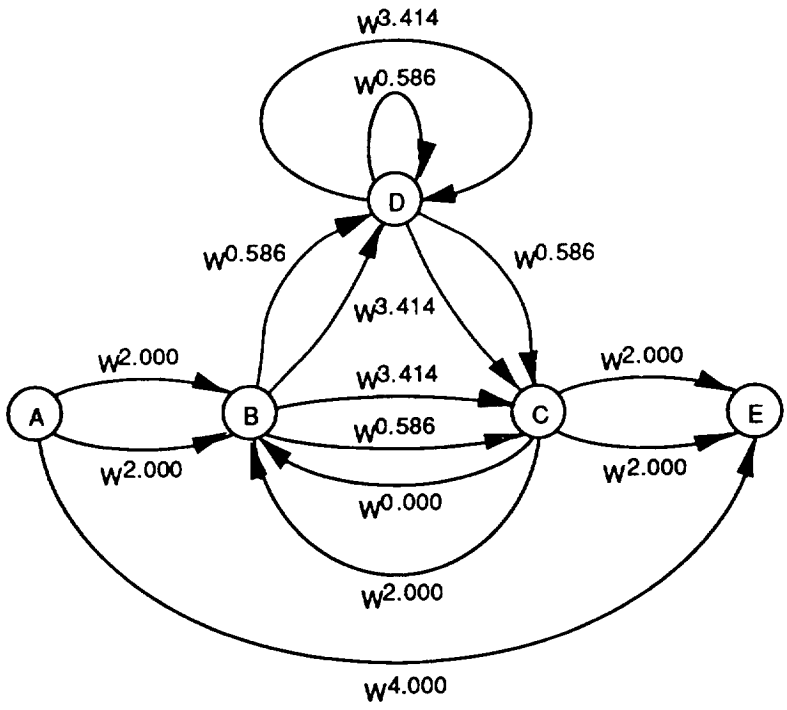

Figure 3.16. Modified state diagram for convolutional encoder.

that performance with respect to the all zeroes path being the correct path, is equivalent to the performance of the code in general. The signal set mapping of TCM codes, is not strictly linear, however, the property of quasi-linearity, a term coined by Rouanne and Costello [13],

allows TCM codes to be analyzed with only slightly more difficulty than linear codes, as will be discussed later. The node equation for each state is written in terms of the node equations for the predecessor state. The "+" operation denotes the convergence of paths and the coefficient indicates the number of paths. Because the metric is represented as an exponent, the addition of a metric due to an added branch is represented by multiplication. Thus, the node equations for the 4-state rate 2/3 8-PSK code are:

$$X_b = 2W^{2.000}X_a + 2W^{2.000}X_c \tag{3.16}$$

$$X_c = (W^{3.414} + W^{0.586})X_d + (W^{3.414+W^{0.586}})X_b \tag{3.17}$$

$$X_d = (W^{0.586+W^{3.414}})X_d + (W^{3.414} + W^{0.586})X_b \tag{3.18}$$

$$X_e = 2W^{2.000}X_c + W^{4.000}X_a \tag{3.19}$$

Error events are caused by paths which converge to node "e", but the error path includes metrics accumulated only after the error path has diverged from node "a", therefore, the generating function is found by solving the system of equations for $\frac{Xe}{Xa}$ as follows:

$$\frac{Xe}{Xa} = W^{4.000} + \frac{W^{2.000}(W^{0.586} + W^{3.414})}{1-(1-2W^{2.000})(W^{0.586+W^{3.414}})} \tag{3.20}$$

Thus we can see that the infinite number of error paths for the 4-state trellis codes is representable by a closed form expression.

3.4.2 Bit Error Spectrum Algorithm

The bit error spectrum technique is similar the generating function in the sense that the paths to any node are defined in terms of the paths to its predecessor nodes, and clearly defined operations exist to depict what happens when a path picks up an additional branch to a successor node and there merges with other paths. The bit error spectrum is in fact a programmatic method for finding the terms of the generating function. Like the generating function, the bit error spectrum technique finds error paths with respect to the all zero path, and represents state zero as two states, a starting state from which error paths diverge, and a finishing state, to which error paths converge. Each entry in the bit error spectrum includes three items of information: the metric, the number of paths, and the average bit error weight per path. Each of these three numbers is a floating point value, the reason for non-integer number of paths and non-integer bit error rates will be explained subsequently.

The program described here is iterative. The first iteration is started by recording one path at the starting state, with a bit error weight of zero and a metric of zero. The first iteration yields all paths of only one branch, the N^{th} iteration generates all paths of N or fewer branches. At each iteration, the algorithm derives a revised spectrum from the spectrum created by the previous iteration. After a sufficient number of iterations, all of the entries which significantly impact the bit error rate of the code should be obtained, although there is straightforward way to predict how many iterations will be required.

The bit error spectrum is stored in the computer in the form of two tables, one that contains the spectrum generated by the previous iteration, and one that holds the spectrum being generated by the current iteration. The table contains a row for each state, including an auxiliary row for the finisher state. Thus for an S state code, there are S+1 rows, numbered 0 through S. Each row has room for a predetermined number of spectral entries, which are stored in order of increasing metric.

The procedure for generating a new spectrum from the previous spectrum is as follows. The starting state, state 0, is never updated, since all of the paths which converge to state 0 of the code are written to the finisher state,

row S of the bit error spectrum table. Therefore the update operation is performed for rows 1 through S of the next spectrum table. The operation of updating the spectrum must reflect what happens to the paths when they pick up an additional branch in going from the predecessor state to the current state, and then merge with other paths. Each entry in the previous spectrum of each predecessor state generates a new entry in the updated spectrum of the current state. The metric of the new entry is equal to the metric of the previous entry plus the transitional metric associated with the branch from the previous state to the new state, while the bit error weight of the new entry is found by adding the bit error weight of the branch to the bit error weight of the previous entry. The bit error weight of a branch is the fraction of nonzero bits associated with the input which causes the encoder to take that branch. If more than one entry of the same metric results, the entries are combined by taking the sum of the numbers of paths and the weighted average of the bit error weights. In practice, memory is conserved by looking to see if an entry for the resulting metric already exists, and if so, performing the combine operation before the new entry is written. At all times, the entries are kept in order of increasing metric. The iteration is completed by generating a new spectrum for every state of the next

spectrum table, each new spectrum being generated from its predecessor states. Once a sufficient number of iterations has been performed, the bit error rate is estimated from the spectrum of the finisher state, row S, using:

$$Pb = \sum_{i} N_i A_i \zeta\left(\frac{m_i}{\sigma}\right) \qquad (3.21)$$

where: m_i is the metric of the i^{th} entry

N_i is the number of paths of metric i

A_i is the average bit error weight of paths of metric i

σ is $\frac{N_0}{2}$

To illustrate this operation consider the example shown in Figure 3.17. Predecessor states P1 and P2 of the previous spectrum are to be combined into the current state C of the next spectrum. The two predecessor states are combined in turn, P1 first. Since P1 is the first predecessor to be combined, there is initially no information at state C. The existing entries at state P1 pick up the additional bit weight and metric of the branch from state P1 to C, thus entries with metrics 4.000 and 6.000 at P1 generate entries with metrics 6.000 and

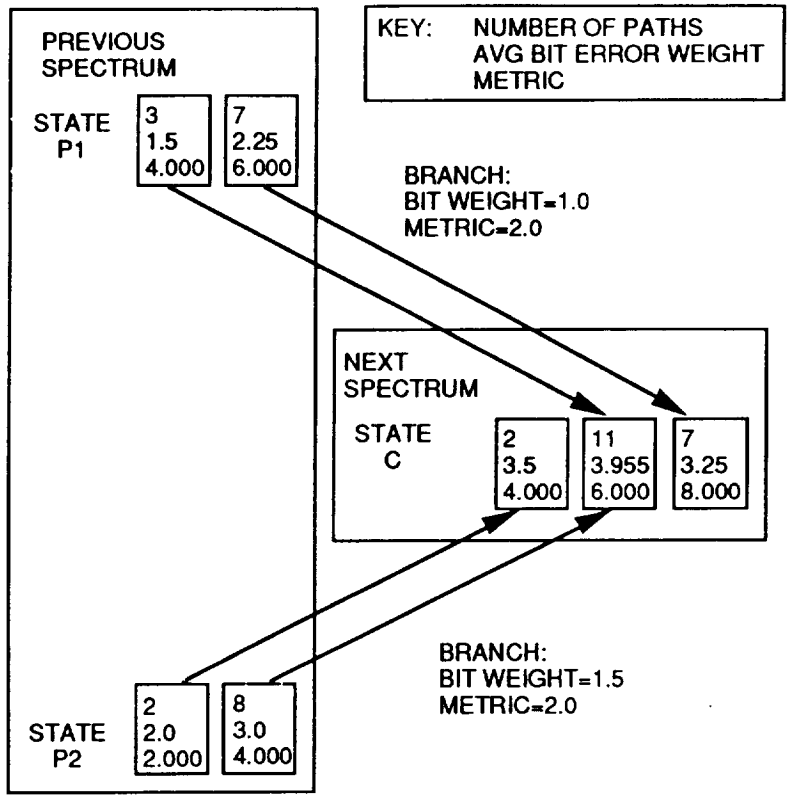

Figure 3.17. Bit error spectrum operation.

8.000 at C. When P2 is combined in C, the entry of metric 2.000 generates an entry with metric 4.000, and no entry with this metric already exists. Therefore the number of paths is the same, but the bit weight and metric are increased by the values associated with the branch from P2 to C. The entry with metric 4.000 at P2 generates a metric of 6.000 at C. An entry with metric 6.000 already exists at C because it was generated by P1, previously. Therefore the resulting number of paths is 3 from P1, plus 8 from P2, for a total of 11. The new average bit weight is the

weighted average of bit weights for paths from P1 and from P2. This is equal to $\frac{3(2.5) + 8(3.0+1.5)}{11} = 3.9555$. The previous spectrum at P2 generates no entry with metric 8.000, so the entry generated by P1 remains the same as it was. Note that in this example the branch metrics are the same, but this is not necessarily always the case. Also, the numbers of paths are shown here to be integers, but due to the non-linearity of the signal set mapping, it is necessary to use non-integer numbers of paths for TCM codes.

To make the bit error spectrum work for arbitrary codes and arbitrary signal sets, the code and signal sets must be defined in a way understood by the machine. This is accomplished by creating a set of tables: the next state table, the next symbol table, the metric table, and the bit weight table. The next state table gives the next state as a function of current state and current input. The next symbol table gives the output symbol associated with each transition depicted in the next state table. Strictly speaking, the bit error spectrum algorithm should have a previous state table and a previous symbol table, since from the previous example, it can be seen that the algorithm merges paths from predecessor states. This, however, is unnecessary, because interchanging the roles of predecessor and successor states generates a "dual" code,

with exactly the same error properties as the original code. The bit error spectrum technique starts with these tables, the tables themselves can be generated by another program (as a function of tap settings or encoder impulse response), or even written manually.

3.4.3 Signal Set Mapping

The metric table is the means of defining the signal set for the bit error spectrum algorithm. To the program, the signal set is simply a set of integers, 0 through M-1, with which a set of metrics is associated. The specific geometry of the signal set is not important to the program. What is important is that the metrics be defined in a meaningful way, ideally as log-likelihoods. Thus the technique could be used for multi-h or FSK codes, as well as for PSK or QAM. It is assumed, however, that the TCM code is generated by mapping an underlying linear code onto the modulation signal set, and that the metrics are defined with respect to symbol zero. For example, the rate 2/3 8PSK encoder of Figure 3.16a accepts 2 data bits, X_1 and X_0, which are used to generate 3 codebits, Y_2, Y_1 and Y_0. The codebits are then mapped onto the 8-PSK signal set. Here, natural mapping is chosen as illustrated in Figure 3.16b.

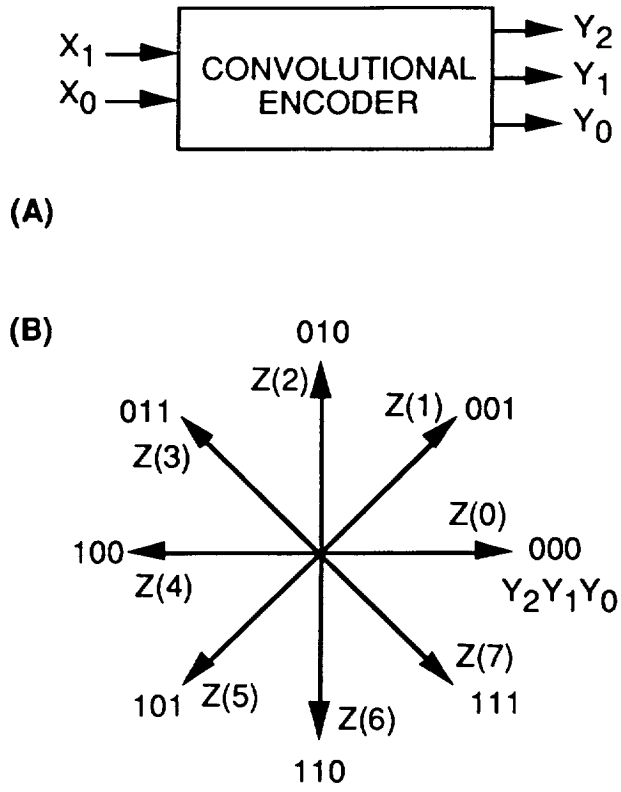

Figure 3.18. a)Rate 2/3 convolutional encoder, b) natural 8-PSK mapping.

The problem to be faced here, is that the mapping is not strictly linear, thus we are not justified in assuming that the performance of the code with respect to the all zeroes sequence is equivalent to the performance of the code for all sequences. If we let Y be a binary number (or the equivalent integer) which indexes the modulation signal vector, and Z(Y) be the actual vector selected by Y, then a strictly linear signal set mapping would give the result:

$$m(Y2) = |Z(Y1+Y2)-Z(Y1)|^2 = |Z(Y2)-Z(0)|^2$$

for any choice of Y1 and Y2. Here the "+" operation is the bitwise exclusive or, and $|Z1-Z0|^2$ denotes the square of the Euclidean distance between two vectors. Also, Y denotes a triple of bits, Y_2, Y_1 and Y_0 (with subscripts), whereas Y1 and Y2 (no subscripts) denote two such triples. This expression shows how the vector space is affected by a change in the underlying codebit space. For a linear mapping, the Euclidean distance between the vectors corresponding to the indexes Y1 and Y1+Y2, depends only on Y2, and is thus denoted m(Y2). For the 8-PSK signal set mapping, linearity applies to some but not all values of Y2. For the non-linear cases, the metric distance m(Y2) depends on Y1 as well as Y2, however it is usually the case that there are fewer possible values of m(Y2) than there are values of Y1. The fact that the non-linearity of the signal set mapping is of a limited extent is the basis of Rouanne and Costello's concept of quasi-linearity [13]. To illustrate this, Table 3.1 shows Y1+Y2 and m(Y2) for all values of Y1 and Y2. As can be seen, m(000), m(001), m(010), m(100), m(101), and m(110) do not depend on Y1, and have values 0.000000, 0.585786, 2.000000, 4.000000, 3.414214, and 2.000000 (scaled to E_S=1), respectively. The

values of m(011) and m(111) can be 0.585786 or 3.414214 depending on Y1. The effect of this on the bit error spectrum program is that the metric table for 8-PSK must have dimensionality 8 by 2, as opposed to 8 by 1, for a strictly linear 8-ary mapping. The symbols 0, 1, 2, 4, 5, and 6 each have only one metric. The symbols 3 and 7 are split between two alternative metrics. When the bit error spectrum program encounters a symbol 3 or 7, two entries with number of paths equal to 0.5 are generated to give the symbol each of its possible metric values. To save computational time, the bit error spectrum employs a symbol split table, to give the number of possible metrics for each symbol. For 8-PSK, the symbol split table is [1, 1, 1, 2, 1, 1, 1, 2]. Thus the algorithm generates fractional paths only when necessary. For other signal sets, a similar procedure is followed. A table similar to Table 3.1, is constructed to determine which symbols have multiple metrics, then the algorithm generates fractional paths for these symbols.

The bit error weight table associates a weight with each encoder input. The bit error weight is the number of nonzero bits in an input divided by the total number of bits in an input. Thus, for a decoder which accepts two bits per symbol, the inputs are 00, 01, 10, and 11, and the bit error weight table is [0.0, 0.5, 0.5, 1.0].

Y1 \ Y2	000	001	010	011	100	101	110	111
000	0.000000	0.585786	2.000000	3.414214	4.000000	3.414214	2.000000	0.585786
001	0.000000	0.585786	2.000000	0.585786	4.000000	3.414214	2.000000	3.414214
010	0.000000	0.585786	2.000000	0.585786	4.000000	3.414214	2.000000	3.414214
011	0.000000	0.585786	2.000000	3.414214	4.000000	3.414214	2.000000	0.585786
100	0.000000	0.585786	2.000000	3.414214	4.000000	3.414214	2.000000	0.585786
101	0.000000	0.585786	2.000000	0.585786	4.000000	3.414214	2.000000	3.414214
110	0.000000	0.585786	2.000000	0.585786	4.000000	3.414214	2.000000	3.414214
111	0.000000	0.585786	2.000000	3.414214	4.000000	3.414214	2.000000	0.585786

$$m(Y2)=\left|Z(Y1+Y2)-Z(Y1)\right|^2$$

Table 3.1. Effect of binary addition on naturally mapped 8-PSK.

3.4.4 Applications and Results

The bit error spectrum technique is an alternative to simulation in comparing the performance of various trellis codes. The bit error spectrum technique is also useful as a part of a code search procedure. The time needed to complete a bit error spectrum depends on the number of iterations. This is a nonlinear relationship, since, as the bit error spectrum tables accumulate more entries, it takes longer and longer to complete each iteration. Thus it is possible to obtain the first five or ten spectral lines in considerably less time than it takes to obtain twenty or thirty. Thus it is possible to quickly eliminate a large number of inferior codes on the basis of short spectrums, and then use longer spectrums to evaluate the few that remain.

A convolutional code can be completely defined in terms of its tap settings, the connections of the shift register to the parity checks, or equivalently in terms of its impulse response. Note however that an encoder has as many impulse responses as there are bits per symbol. For example, if an encoder accepts two bits per symbol, the response to the input 01, and to the input 10 are both needed to completely define the code. For example, the 64-state Ungerboeck encoder of Figure 1.13d has impulse

responses 6-5-7-6 and 2-0-4-2. From these impulse responses, the response to any input can be generated. A routine which allows the computer to generate the code from these sequences makes it convenient to experiment with various codes, entering the impulse responses at the console. Interestingly enough, it is not difficult to find reasonably good codes by trial and error, selecting the impulse responses with regard to Ungerboeck's set partitioning principles. By this strategy, a 1024-state code with a coding-gain of 1dB over the 64-state code was found after only twelve codes were tried. The encoder for this code is shown in Figure 3.19.

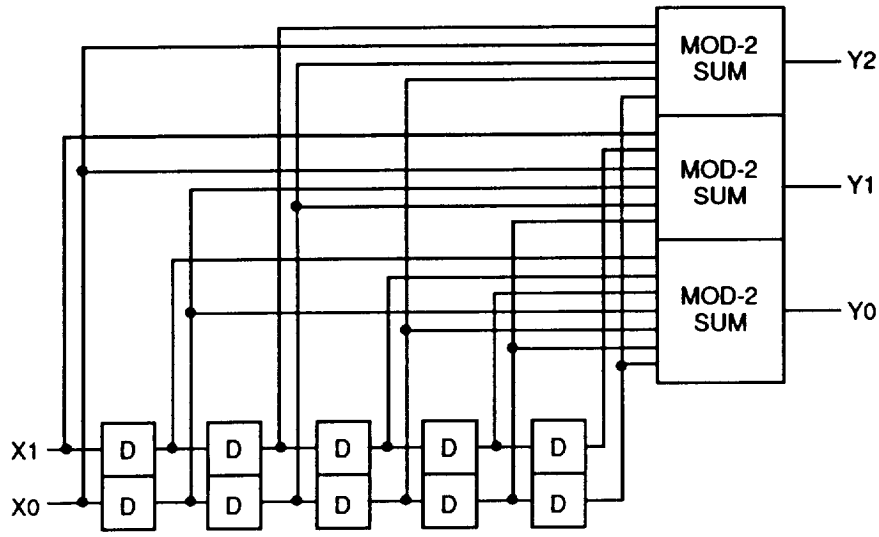

Figure 3.19. 1024-state convolutional encoder.

When the NMSU pragmatic TCM project was in its early stages, the bit error spectrum technique was used to determine if the technique of code puncturing [32], could also be used to incorporate a binary Viterbi decoder into a TCM system, as pragmatic TCM does. The concept was that two bits would be clocked into the industry standard, rate onehalf, constraint length 7 convolutional encoder, generating four code-bits. The four codebits would be linearly mapped to three symbol selection bits, by means of a 4 by 3 binary matrix. This matrix would represent all possible combinations of puncturing and mapping, as well as a large class of mappings that do not directly involve puncturing. The combination of encoder and mapping would generate a TCM code, which could then be decoded with the Viterbi decoder, using quantization and soft decisions, as was done to implement pragmatic TCM. The soft decision adaptation is a compromise which, like an implementation loss, can be expected to effect all codes more or less evenly. Therefore, the use of the bit error spectrum technique to select the puncturing scheme which generates the best code as predicted by Euclidean distance is reasonable.

Since all mapping and puncturing schemes are represented by a 4 by 3 binary matrix, a brute force approach would require $2^{12} = 4096$. By making a judgement

on the basis of the first five spectral lines, an ordinary PC could evaluate all possible combinations in a few days. However, the results of this experiment showed that, at least for the time being, it would be more productive to pursue pragmatic TCM than punctured TCM.

Figure 3.20 compares the results of bit error spectrum analysis to the results of simulations. As can be seen, the bit error spectrum results upper-bound the actual performance, and this effect is very pronounced at low signal-to-noise ratios. At low signal-to-noise ratios, the bit error spectrum will even yield error probabilities greater than 1, a consequence of the overlapping probabilities in the terms of the union bound. The bit error spectrum technique remains useful as a means of comparing the relative performance of different codes. Figure 3.21 compares the bit error spectrum results of the 16-state Ungerboeck code to the asymptotic performance of the pragmatic code. Since the pragmatic code is lower bounded by the asymptotic curve, and the 16-state code is upper bounded by the bit error spectrum calculation, we can expect the performance of the two codes to be essentially equivalent between bit error rates of 10^{-5} and 10^{-6}. At bit error rates of less than 10^{-6}, the 16-state code gains superiority. Essentially the same conclusion can be drawn from Figure 3.22, which shows bit error spectrum results

for both the pragmatic code and the 16-state code. The source listing for the bit error spectrum program is given in appendix A.

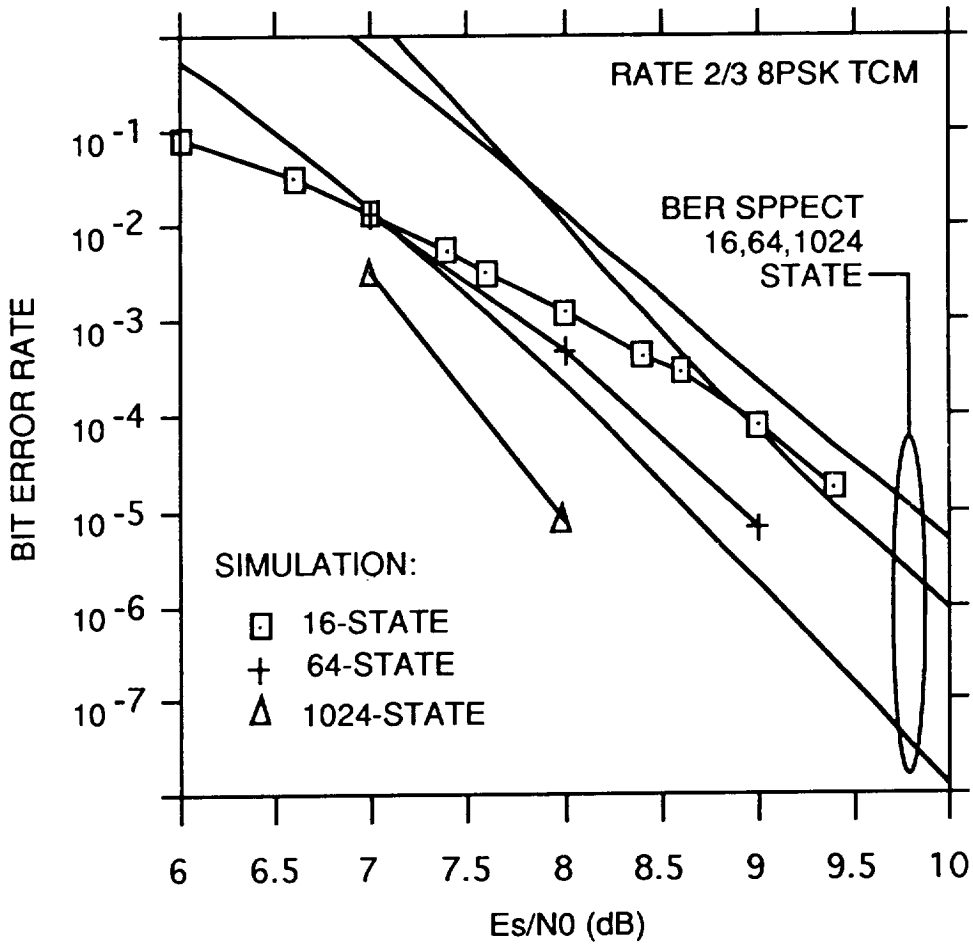

Figure 3.20 Bit error spectrum calculations compared with simulation results.

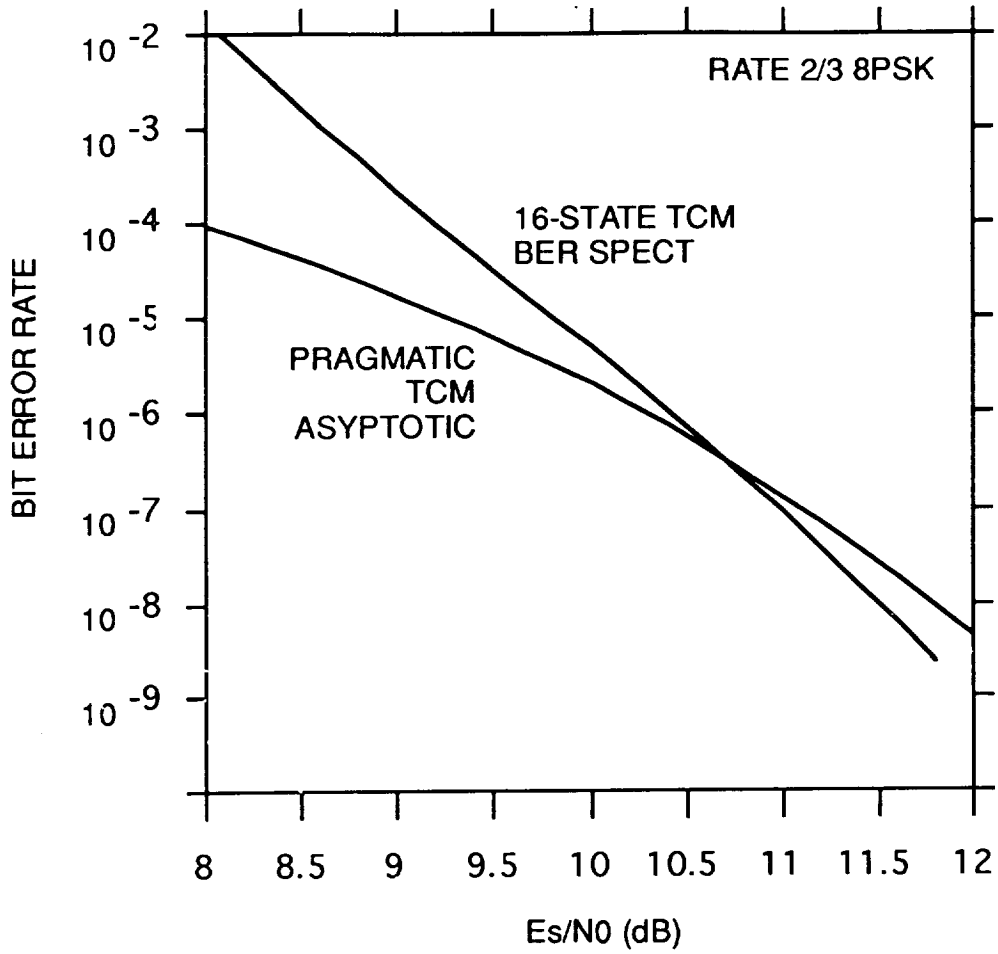

Figure 3.21. Bit error rate spectrum result for 16-state code compared with asymptotic error rate for pragmatic code.

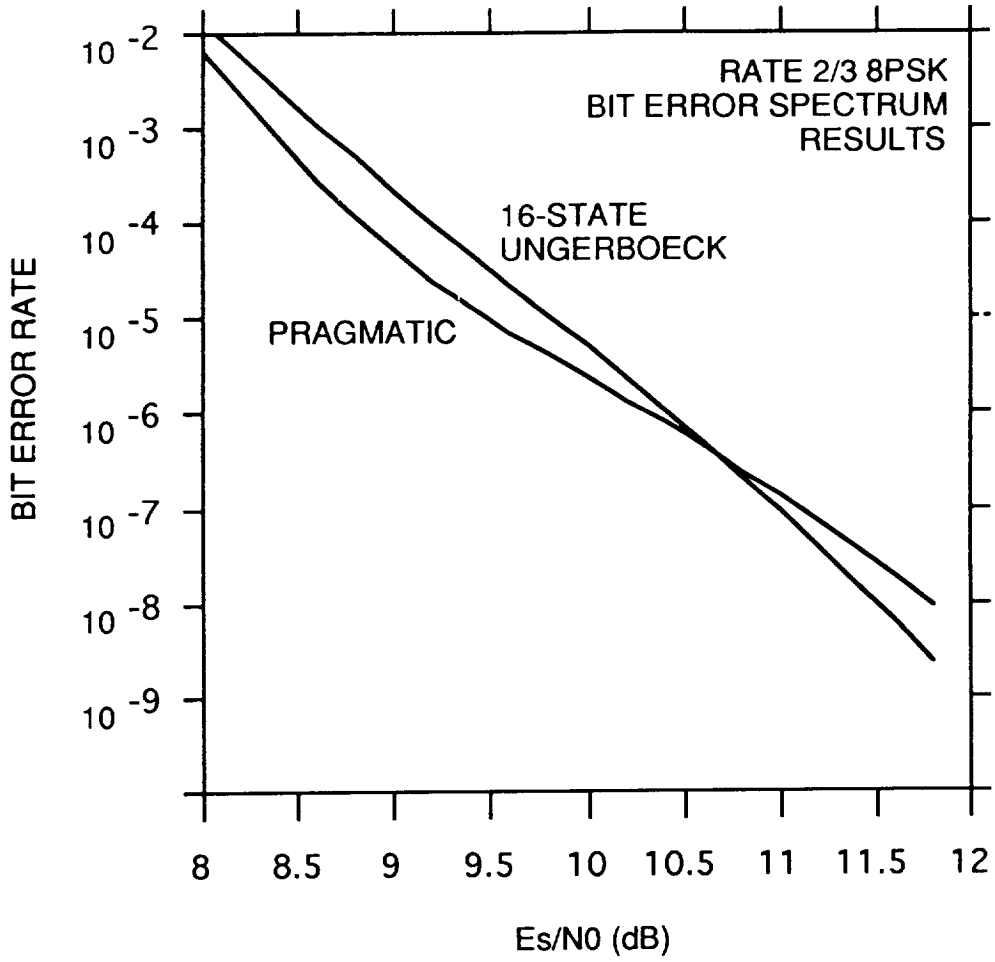

Figure 3.22 Bit error rate spectrum comparison of pragmatic TCM and 16-state Ungerboeck TCM.

3.5 Conclusion

This chapter has presented experience gained in TCM codes before undertaking the high-speed TCM architecture project. The simulations confirmed the expected performance of various TCM codes, while the bit error spectrum technique supplies additional theoretical input.

The high-speed design is to be presented in the next chapter. As this is done, it will become apparent how issues such as quantization and coding standard affect the overall complexity of the design. On the basis of the research described in this chapter, the 16-state Ungerboeck code seemed to be the most favorable coding standard, although the decision was rather close. The techniques presented in Chapter 4, which are applied to the 16-state decoder, are also directly applicable to the design of a pragmatic decoder. However, it appears that the 16-state Ungerboeck code requires slightly less hardware, and is therefore the code used in the high-speed design.

4. HIGH-SPEED DESIGN

The design of the high-speed decoder is hierarchical with the top level consisting of three major units: The metric calculator, the decision-making (ACS) unit, and the path memory unit, as shown in Figure 4.1. The decoder is designed to decode the rate $\frac{2}{3}$ 8-PSK 16-state Ungerboeck code having the decoder shown in Figure 4.2a; however, in order to achieve the high-speed design, the encoder is modified as shown in Figure 4.2b, for reasons discussed in Section 4.3.2. The encoder outputs 3 bits, Y2, Y1, and Y0 which specify an 8-PSK vector according to natural mapping. That is, the three bits specify a binary integer k, and the phase of the transmitted vector is $\phi=(k+\frac{1}{2})\frac{\pi}{4}$, as shown in Figure 4.3.

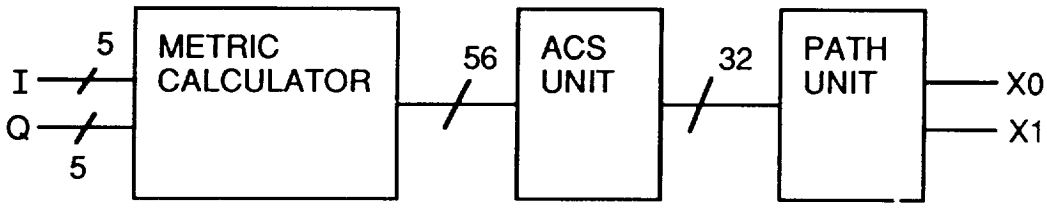

Figure 4.1. Top level diagram of high-speed decoder.

The decoder expects to receive a pair of 5-bit numbers representing the I and Q components of the received signal vector. The representation of I and Q is naturally mapped, uniformly quantized binary numbering, with 0 representing

$-\frac{31}{32}\sqrt{E_S}$ and 31 representing $+\frac{31}{32}\sqrt{E_S}$ as illustrated in Figure 4.4. In fact, the decoder uses natural numbering to represent all numbers used in the algorithm. The design is illustrated down to the logic gate level, and all of the logic has been verified using BOSS. Every operation in the algorithm has been pipelined. Also, the design is fully parallel and fully synchronous, so that every component of the system will run on the symbol clock.

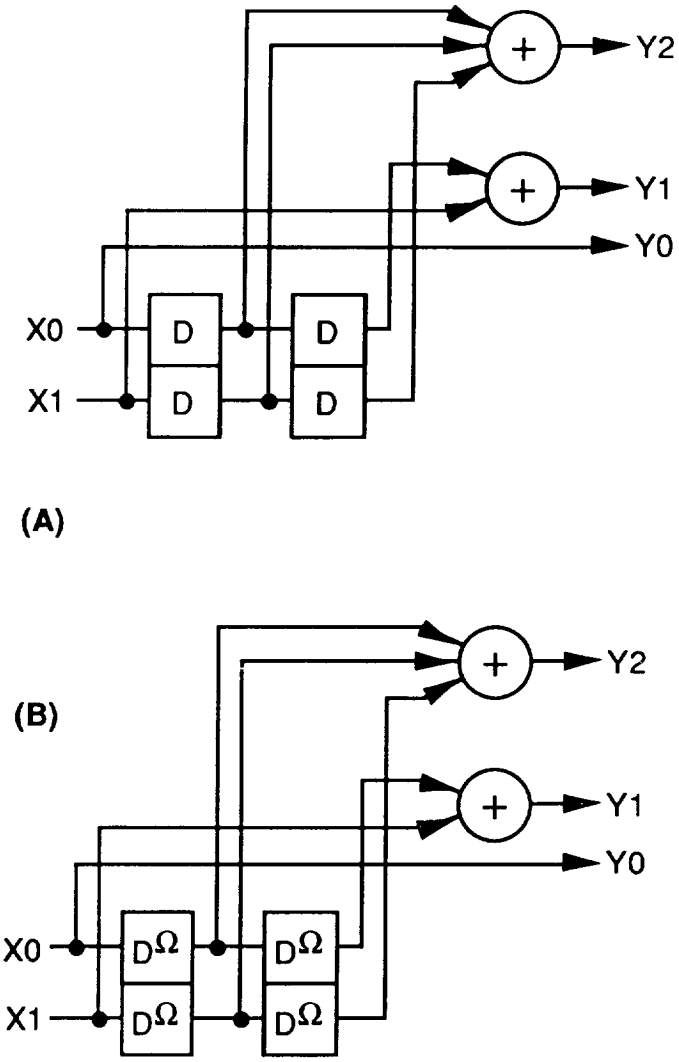

Figure 4.2. a) 16-state Ungerboeck encoder, b) modified encoder.

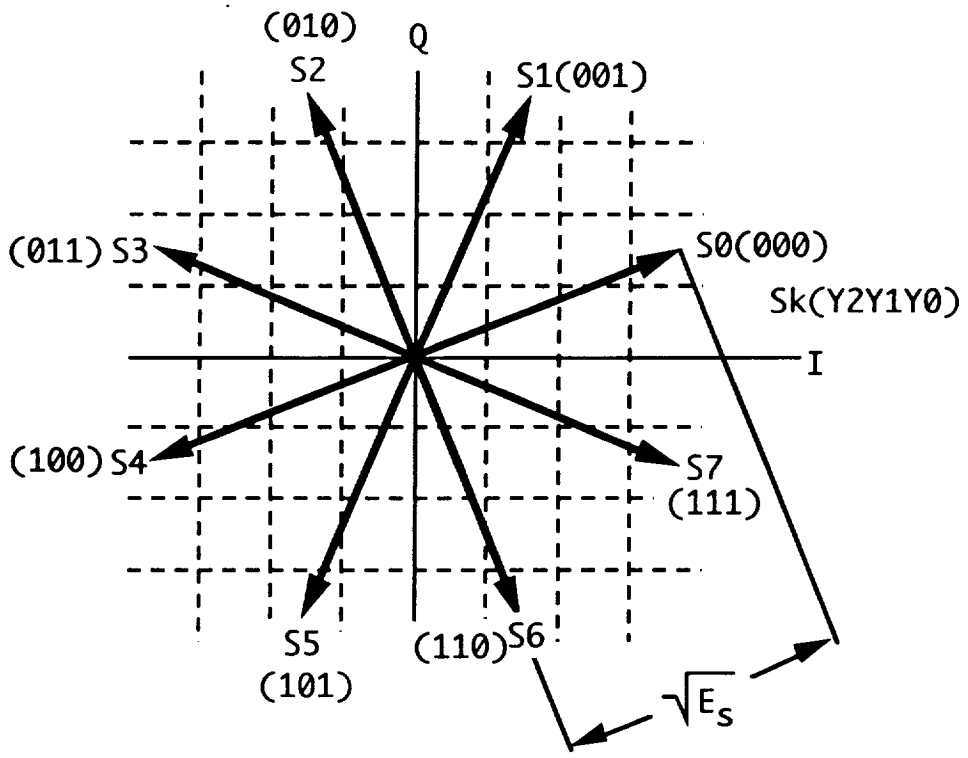

Figure 4.3. 8-PSK signal constellation.

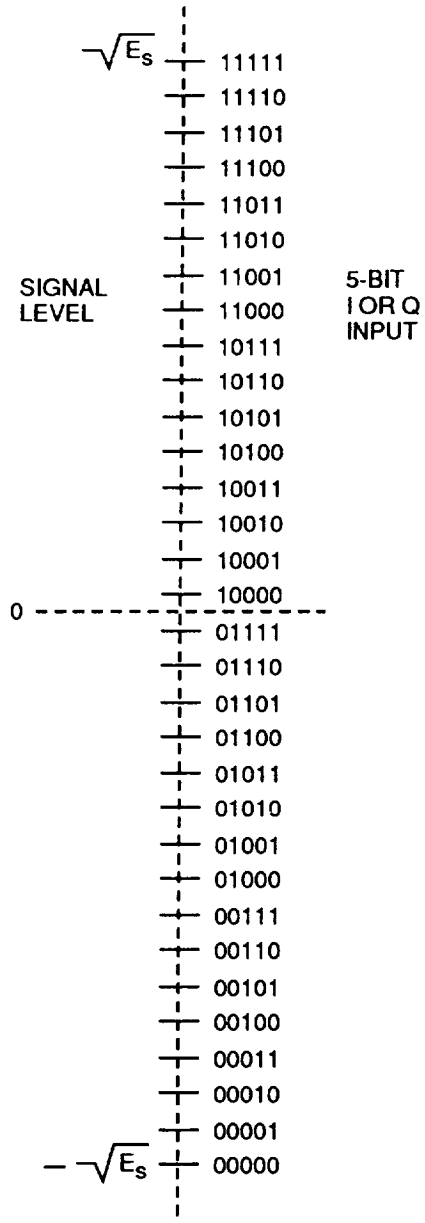

Figure 4.4. 5-bit I and Q inputs for high-speed codec.

4.1 Pipelining

Pipelining is a well known technique in the design of digital systems. The basic idea is as follows. Suppose a

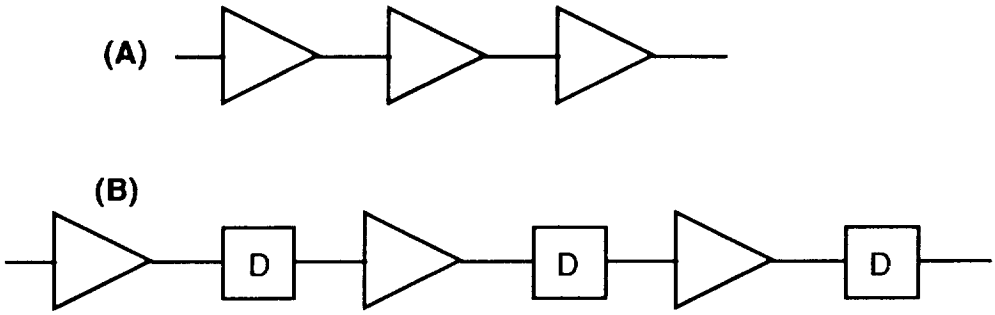

Figure 4.5. a) multi-stage logic. b) pipelining.

given logical operation requires N layers of gates, as shown in Figure 4.5a. The speed of the operation is limited by the propagation delay through these gates. To pipeline the operation is to add a latch after each gate as shown in Figure 4.5b. The time required to complete the operation is still N gate delays; however, a second execution of the operation can begin as soon as the result of the first stage of the first execution of the operation is clocked into the first latch. Thus, although the time delay between the input and the output is unchanged, the throughput of the circuit is increased. The rate at which the operation can be repeated is limited by 1 gate delay as opposed to N gate delays. Of course the designer will not

necessarily place a latch after every single gate, but will choose the tradeoff between speed and hardware which is best suited to the application. The codec presented here employs no more than the equivalent of 3 NAND gates between any pair of latches in the system.

The Viterbi Algorithm consists extensively of arithmetic and logical operations which can yield increased throughput when pipelining is applied. As an example of this, consider the operation of adding two bits. The sum bit is given by the exclusive OR operation, and the carry bit is given by the AND operation. In N-bit addition, the well known carry ripple effect occurs, due to the fact that the sum bit in any position depends on the carry bit of the previous position, and in turn on the results of the operation in every less significant position. Thus if an N-bit adder is designed in the most simplistic way, the most significant bit will not be available until after the time required to perform N+1 single bit additions. One established strategy for dealing with this problem is carry save arithmetic. In carry save arithmetic, the carry bit is considered to be part of the representation of the number. Circuitry which uses the result of the operation may then be designed to accept the carry save representation, or the carry save result can be converted back to natural representation using pipelined circuitry.

The high-speed codec employs a 5-bit adder in the metric calculation unit. The 5-bit adder is built out of 1-bit adders as shown in Figure 4.6. The 1-bit adder is shown in Figure 4.7. Latches are included for both the sum bit (Z), and the carry bit (C), resulting in a pipelining

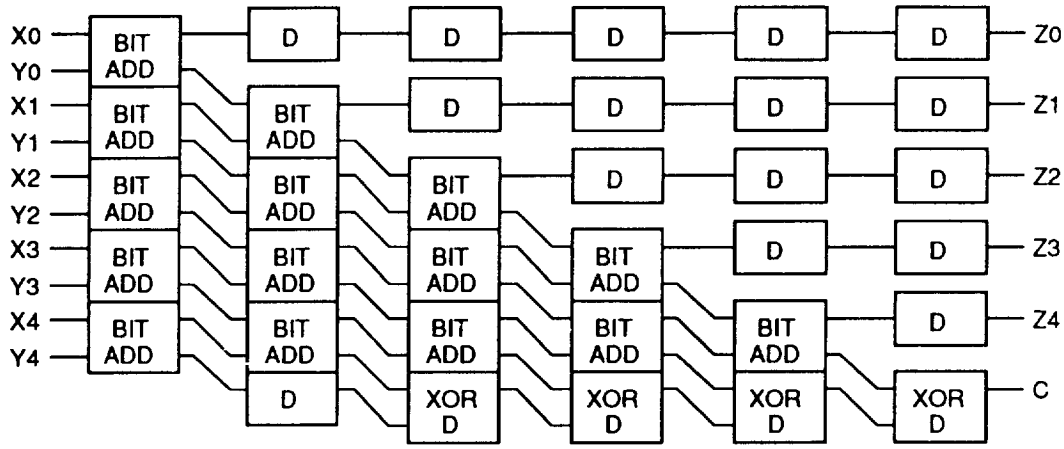

Figure 4.6. 5-bit adder.

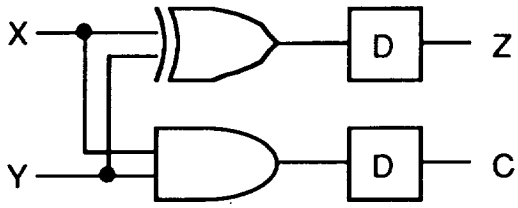

Figure 4.7. 1-bit adder.

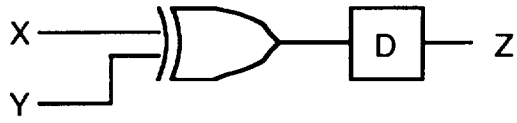

Figure 4.8. XOR/D module.

effect when the 1-bit adder is used as part of a larger circuit. The 5-bit adder includes a block labeled "XOR D" which is simply a latched exclusive OR, as shown in Figure 4.8. The 5-bit adder performs the operation of adding two 5-bit numbers in six stages. In effect, the carry ripple effect has been pipelined. The outputs of the bit adders at each stage of the 5-bit adder form a carry save representation of the result. After the sixth stage, the result is rendered as a 6-bit binary number. The metric adder also employs a 5-bit subtracter, shown in Figure 4.9, which is identical to the 5-bit adder, except that the single bit adders are replaced with single bit subtracters. The single bit subtracter, shown in Figure 4.10 has latched outputs for (Z), the difference bit and (B), the borrow bit.

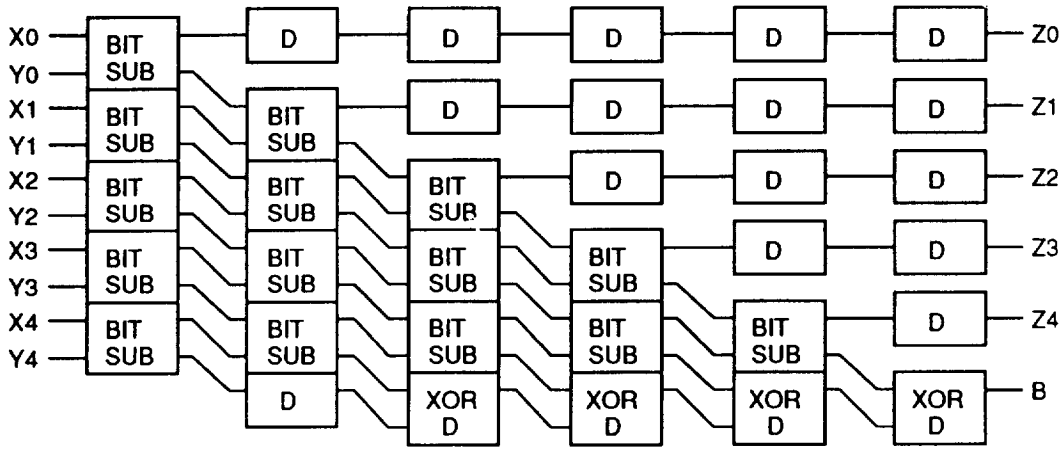

Figure 4.9. 5-bit subtracter.

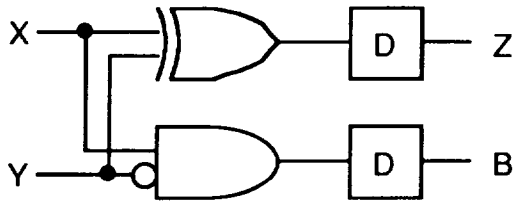

Figure 4.10. 1-bit subtracter.

4.2 Metric Calculation

Each time a signal vector is received, the Viterbi decoder associates a metric with each symbol of the source alphabet. In the case of TCM transmitted on a two dimensional additive white Gaussian channel, the metric is the square of the Euclidean distance between the source symbol vector and the received vector, that is, $m_i = (I_i - I_R)^2 + (Q_i - Q_R)^2$, where I_i, Q_i, I_R and Q_R are the I and Q components of the i^{th} symbol vector, and the received vector, respectively. In general, the quantity $(I_i - I_R)^2$ can potentially use twice as many bits as are used to represent I_R. For the 8-PSK constellation of Figure 4.3, there are 8 symbol vectors, with four possible values for I_i, and the same four possible values for Q_i.

The high-speed decoder is designed to accept the I and Q components of the received vector quantized on a linear integer scale of 0 to 31 (5-bit quantization) with 0 representing $-\sqrt{Es}$ and 31 representing $+\sqrt{Es}$. Using this

scale, the four possible values for the I and Q components of the signal vectors quantize to 30, 20, 10, and 1. The metric calculator is shown in Figure 4.11. The "metric comps" (metric components) unit, shown in Figure 4.12, accepts a 5-bit integer X_R (which can be either the I

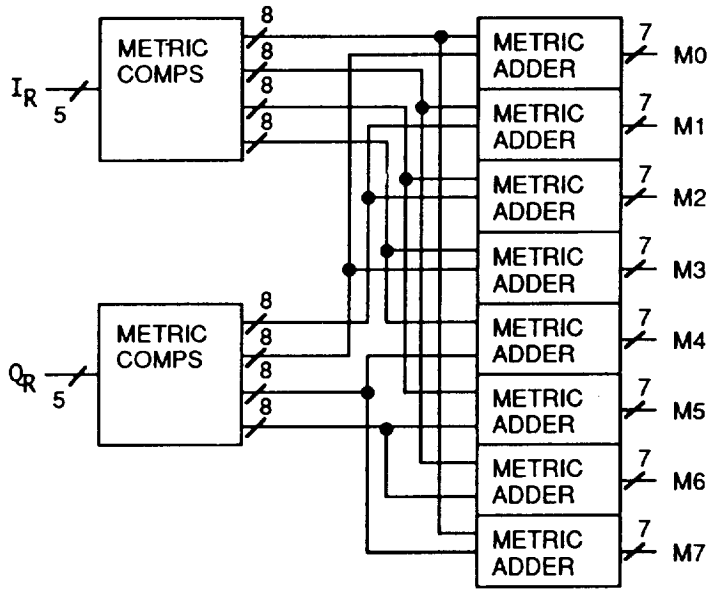

Figure 4.11. Metric calculator.

component or the Q component of the quantized received vector) and calculates $(X-X_R)^2$ for X = 31, 20, 10, and 1. Because X_R will be a 5-bit number, $(X-X_R)^2$ will be a 10-bit number. However, because the decoder will use only 7-bit metrics, and because the first order bit of the square of a binary number is always zero, the two least significant (zeroth and first order) bits of $(X-X_R)^2$ are ommitted by the square law circuit. Therefore, the "metric

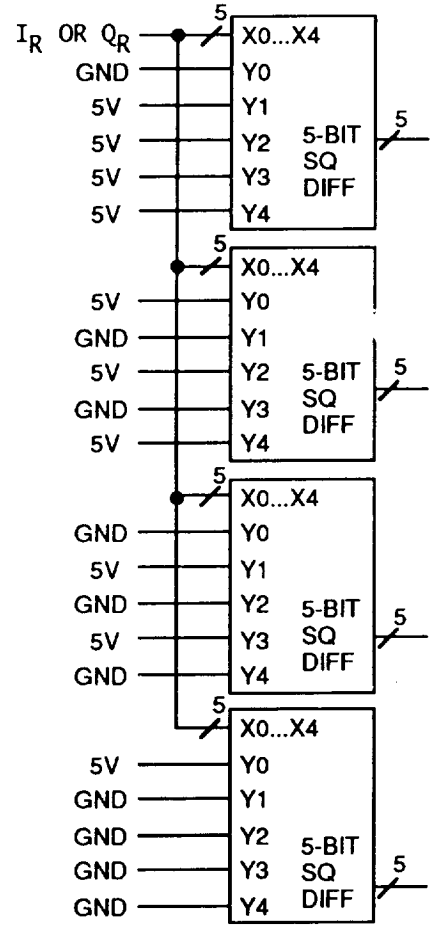

Figure 4.12. Metric comps.

comps" unit provides four 8-bit outputs. Two identical metric comps units are employed, one to provide $(X-I_R)^2$ for the for values of X, and the other to calculate $(X-Q_R)^2$ for the four values of X.

Each of the eight "metric adder" units accepts two "metric comps" outputs, one from the I unit and the other from the Q unit, and sums them to calculate the metric for one of the eight 8-PSK symbols. The metric adder is

discussed in Section 4.2.2. Summing the two 8-bit numbers, produces a 9-bit number of which two bits are discarded to form a 7-bit metric. If the most significant discarded bit (the first order bit of the total) is 1, the metric is rounded up. In binary arithmetic, this amounts to adding 1 to the retained 7-bit number.

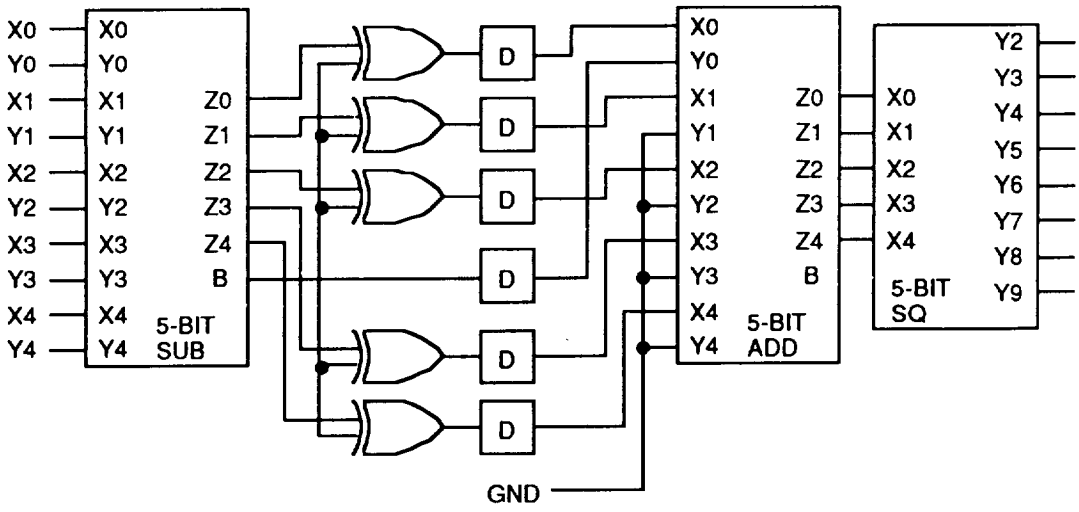

Figure 4.13. 5-bit square difference.

The "square difference" circuit is illustrated in Figure 4.13. This circuit begins with a 5-bit subtracter circuit of the type discussed previously. If the borrow bit (B) is asserted, it means that a larger number was subtracted from a smaller, and the result is incorrect. If this occurs, the result is corrected by inverting each bit of the difference (this is the function of the five exclusive OR gates) and adding 1. If the carry bit is not

asserted, then the exclusive OR gates and the addition operation have no effect. Either way, the 5-bit square law circuit receives as input the absolute value of the difference. The square law circuit is discussed in Section 4.2.1.

4.2.1 Square Law Circuit

The square of an N-bit number is at most a 2N-bit number. By compiling a truth table for the 5-bit square operation, one can derive the Boolean expression for each bit of the result. These are as follows:

$$y_0 = x_0$$

$$y_1 = 0$$

$$y_2 = x_1 \cdot \overline{x_0}$$

$$y_3 = (\overline{x_2} \cdot x_1 \cdot x_0) + (x_2 \cdot \overline{x_1} \cdot x_0)$$

$$y_4 = (x_2 \cdot \overline{x_1} \cdot \overline{x_0}) + (\overline{x_3} \cdot x_2 \cdot x_0) + (x_3 \cdot \overline{x_2} \cdot x_0)$$

$$y_5 = [(\overline{x_4} \cdot x_1) \cdot (x_3 \oplus x_2)] + [x_0 \cdot (x_4 \oplus x_3) \cdot (\overline{x_3 \oplus x_2})] + [x_4 \cdot (x_3 \oplus x_2) \cdot (x_1 \oplus x_0)]$$

$$y_6 = (\overline{x_4} \cdot x_3 \cdot \overline{x_2}) + (\overline{x_4} \cdot x_3 \cdot x_2 \cdot x_1) + (x_4 \cdot \overline{x_3} \cdot \overline{x_2} \cdot x_1) + (x_4 \cdot \overline{x_3} \cdot x_1 \cdot \overline{x_0}) + (x_4 \cdot x_3 \cdot x_0) + (x_3 \cdot \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0})$$

$$y_7 = (\overline{x_4} \cdot x_3 \cdot x_2) + (x_4 \cdot \overline{x_3} \cdot x_2 \cdot \overline{x_1}) + (x_4 \cdot \overline{x_3} \cdot x_2 \cdot \overline{x_0}) + (x_4 \cdot x_3 \cdot x_1)$$

$$y_8 = (x_4 \cdot \overline{x_3} \cdot \overline{x_2}) + (x_4 \cdot x_3 \cdot x_2) + (x_4 \cdot \overline{x_3} \cdot \overline{x_1}) + (x_4 \cdot \overline{x_3} \cdot \overline{x_0})$$

$$y_9 = (x_4 \cdot x_3) + (x_4 \cdot x_2 \cdot x_1 \cdot x_0)$$

The 5-bit square circuit show in Figures 4.14, 4.15, and 4.16, generates bits y_2 through y_9 according to these expressions. The circuit is pipelined, as is the entire decoder. The 5-bit square circuit is employed because it was determined that the 16-state Ungerboeck code would require 5-bit I and Q.

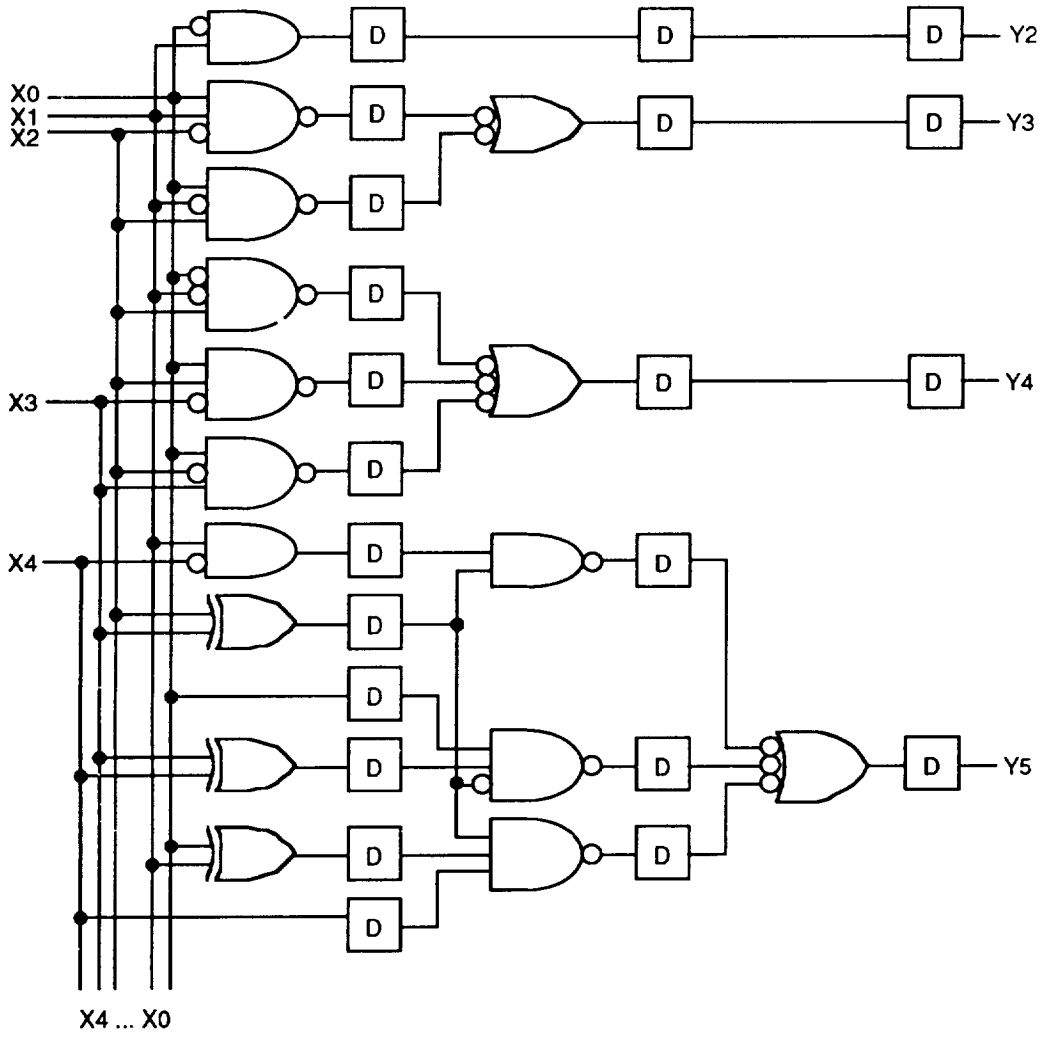

Figure 4.14. 5-bit square, part 1 of 3.

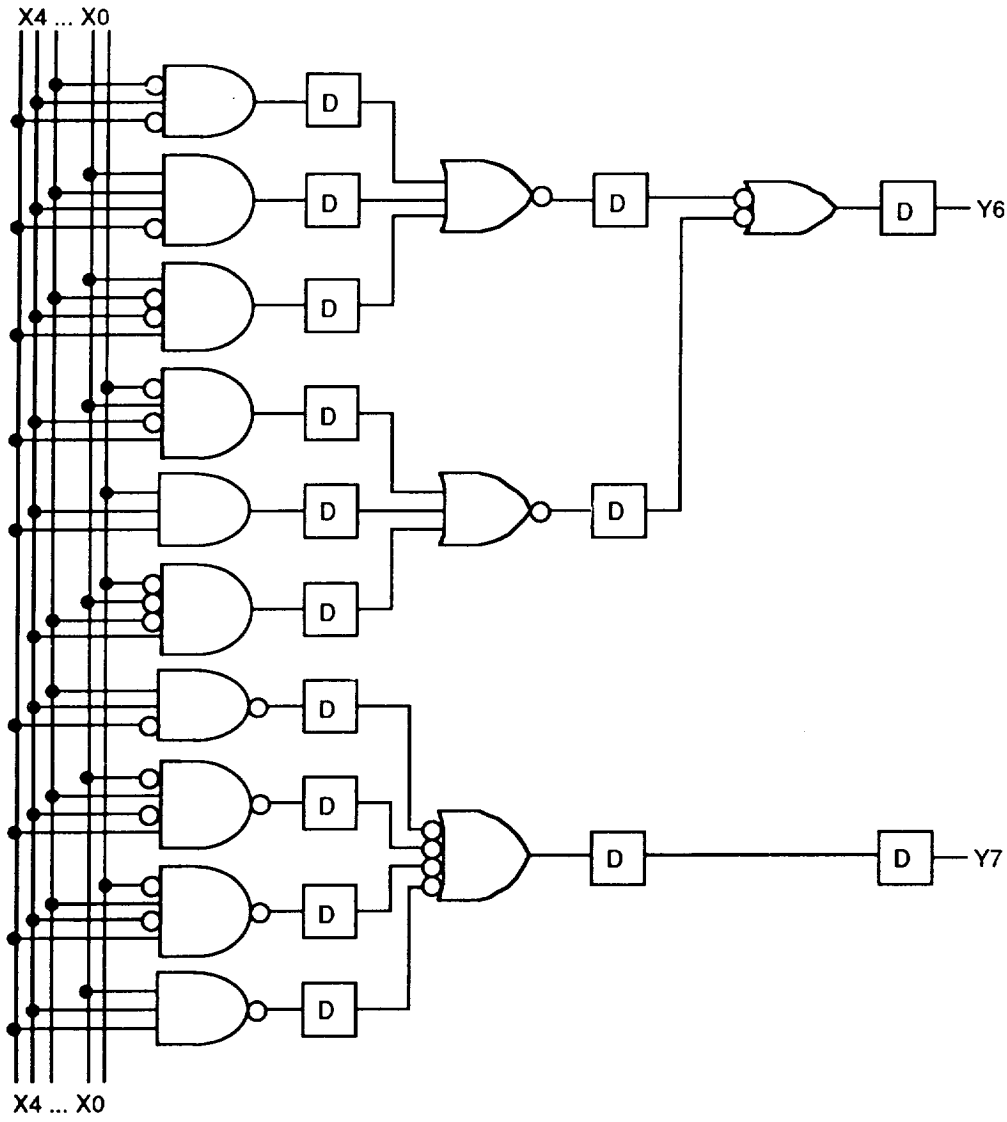

Figure 4.15. 5-bit square, part 2 of 3.

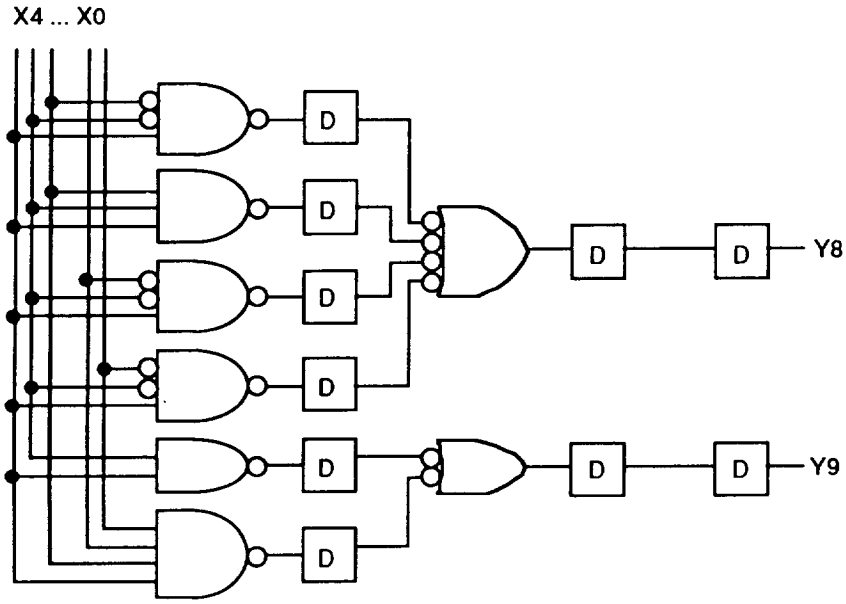

Figure 4.16. 5-bit square, part 3 of 3.

Admittedly, some hardware savings could be obtained by using a code that can use 4-bit I and Q, and therefore a 4-bit square circuit. The logical expressions for the 4-bit square law circuit are as follows:

$$y_0 = x_0$$

$$y_1 = 0$$

$$y_2 = x_1 \cdot \overline{x_0}$$

$$y_3 = (\overline{x_2} \cdot x_1 \cdot x_0) + (x_2 \cdot \overline{x_1} \cdot x_0)$$

$$y_4 = (x_2 \cdot \overline{x_1} \cdot \overline{x_0}) + (\overline{x_3} \cdot x_2 \cdot x_0) + (x_3 \cdot \overline{x_2} \cdot x_0)$$

$$y5 = (\overline{x_3} \cdot x_2 \cdot x_1) + (x_3 \cdot \overline{x_2} \cdot x_1) + (x_3 \cdot x_2 \cdot x_0)$$

$$y6 = (x_3 \cdot \overline{x_2}) + (x_3 \cdot x_1)$$

$$y7 = x_3 \cdot x_2$$

The 4-bit square circuit, as shown in Figure 4.17 can be seen to be much smaller than the 5-bit square circuit.

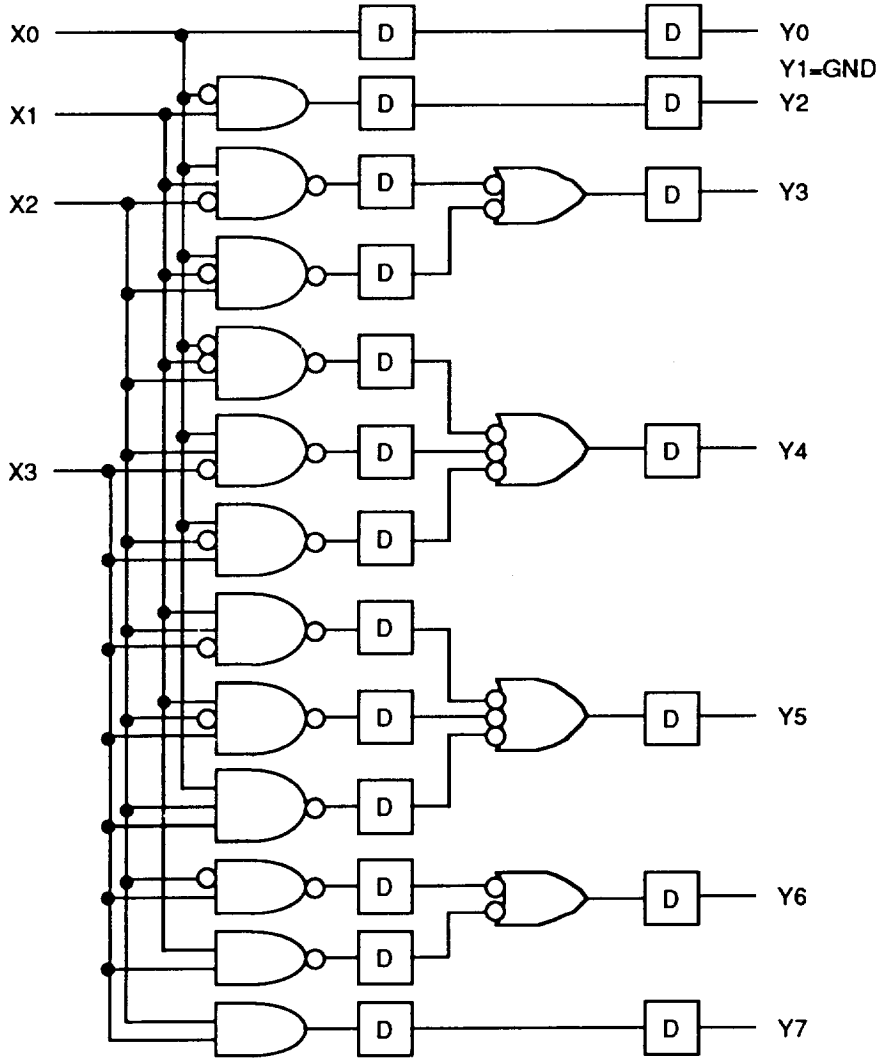

Figure 4.17. 4-bit square.

4.2.2 The Metric Adder

The metric adder performs the last step in the calculation of the metrics. Eight metric adder circuits are employed, each of which outputs the metric for one of the eight 8-PSK symbols. The metric for any given symbol k is given by

$$M_k = (I_k - I_R)^2 + (Q_k - Q_R)^2$$

where Ik and Qk are the I and Q components of the signal vector, and IR and QR are the I and Q components. The square difference terms are provided by the metric components units discussed previously. Each metric adder is given the two square difference terms which will result in the metric for which it is responsible. The metric adder circuit is shown in Figure 4.18. This circuit operates by the same principle as the 5-bit adder discussed previously, however there is an important difference. The metric adder circuit adds two 8-bit numbers, producing a 9-bit number, of which only the seven most significant bits are used. In discarding the two least significant bits, the most significant discarded bit must be carried into the retained bits in order to produce correct binary rounding.

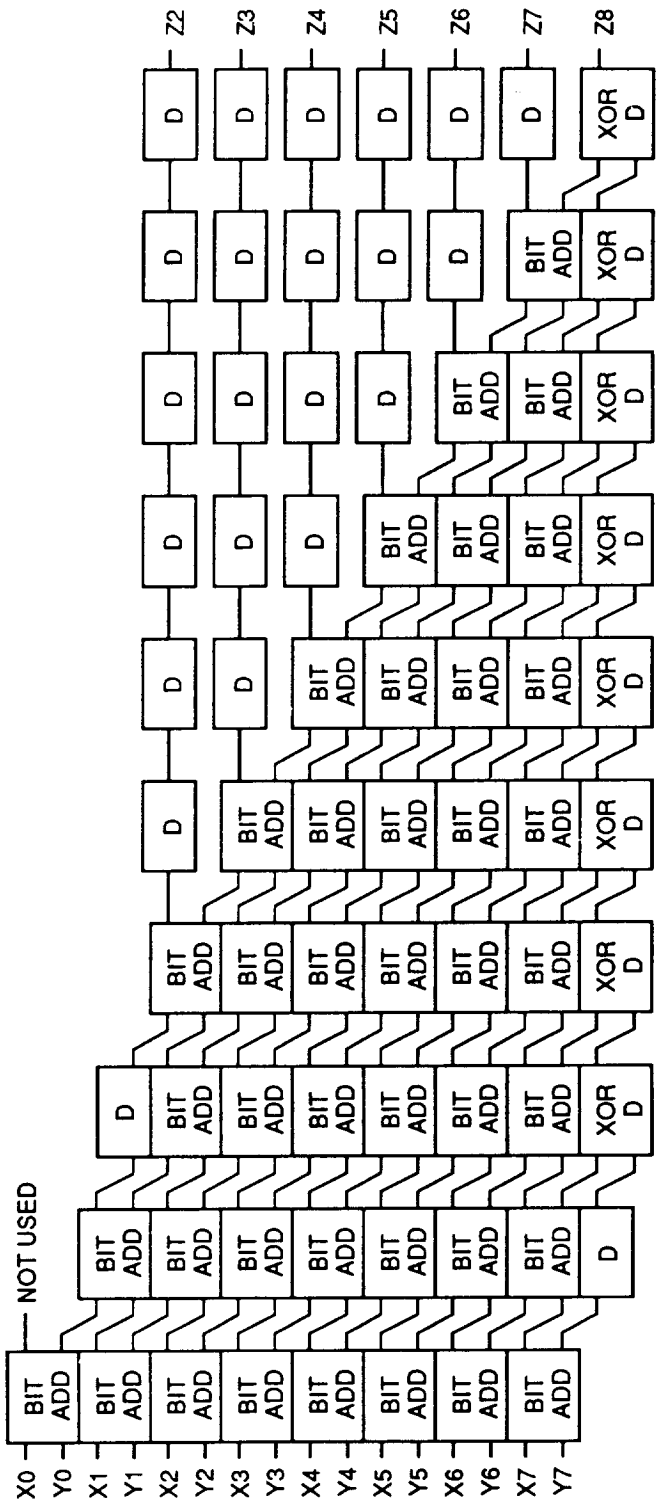

Figure 4.18. Metric Adder.

This is analogous to the rule that in decimal rounding, discarding a digit of 5 requires the result to be rounded up. The experiments with BOSS have shown that failure to perform this step is almost as bad as losing one bit of accuracy. How the rounding is accomplished can be seen in Figure 4.18. In the first stage of the addition, the least significant result bit (top row) cannot effect the final rounded result, so it is simply thrown away. At the second stage of the addition, the least significant result bit is passed to a latch at the third stage, and then to one of the inputs of the top row bit adder at the fourth stage. After the fourth stage, the addition operation continues normally, producing the required 7-bit metric.

4.3 The Add-Compare-Select Circuit

The add-compare-select (ACS) circuit consists of sixteen identical add-compare select cells, each of which stores the cumulative metric and selects the appropriate branch for one of the sixteen nodes of the trellis. The ACS unit is illustrated in Figures 4.19 through 4.22. Each of the ACS cells receives four cumulative metrics from other ACS cells, and four symbol metrics from the metric calculation unit, as dictated by the trellis code.

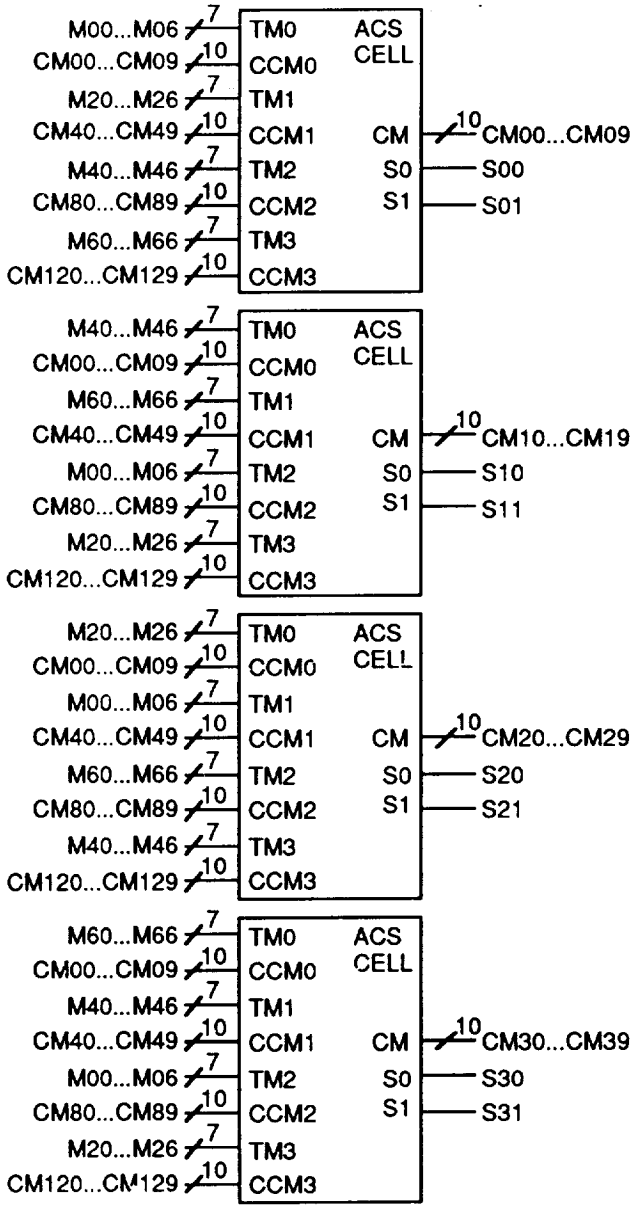

Figure 4.19. ACS unit, part 1 of 4.

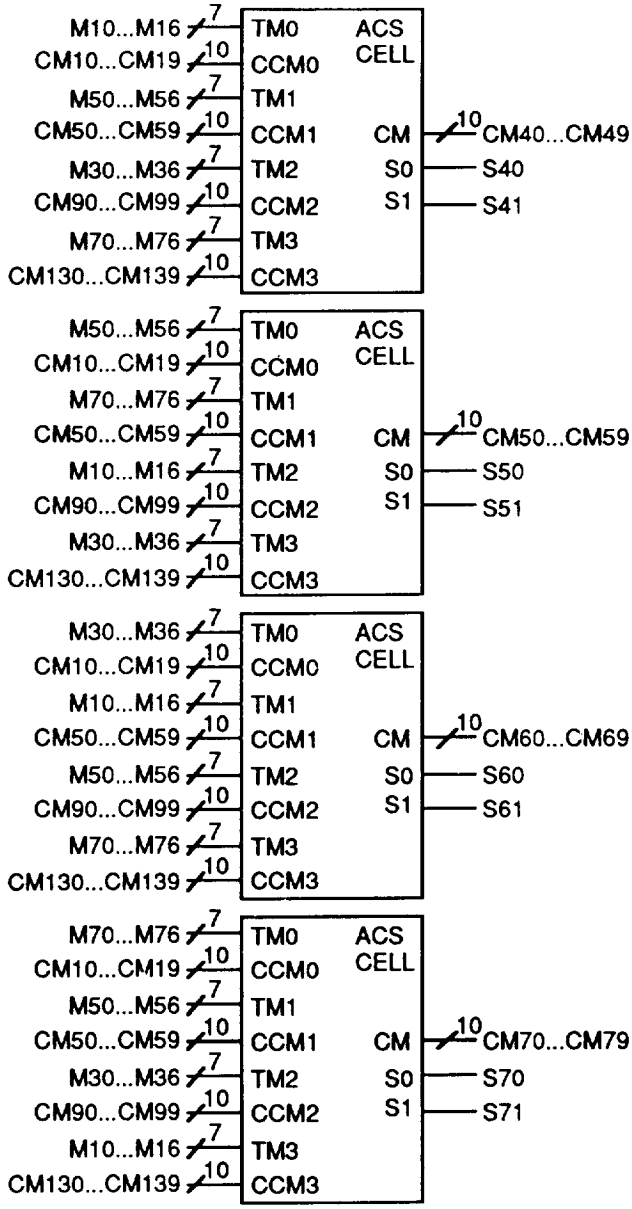

Figure. 4.20. ACS unit, part 2 of 4.

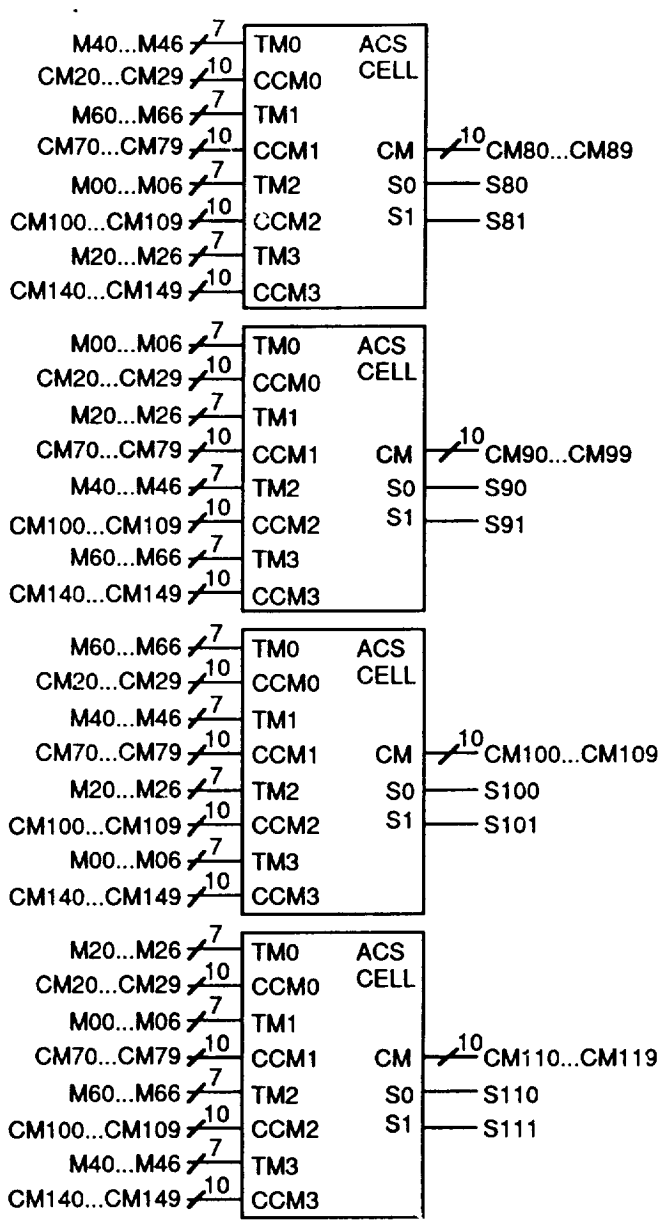

Figure 4.21. ACS unit, part 3 of 4.

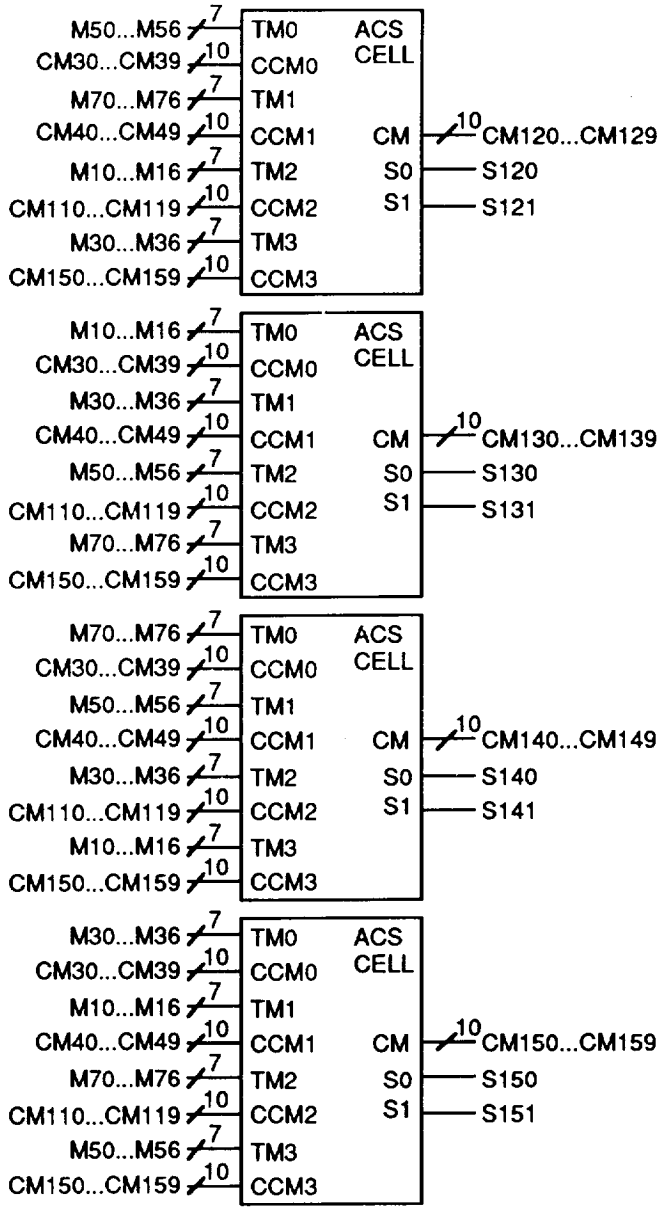

Figure 4.22. ACS unit, part 4 of 4.

The decoder avoids the need to reset metrics by using the modulo arithmetic method of Hekstra [19]. This method requires that the register for the cumulative metric be able to hold a number which is at least twice as large as

the largest difference which can occur between two path metrics. This number is the largest branch metric used in the system, multiplied by the constraint length of the code. The constraint length of the 16-state Ungerboeck code is 3, so the cumulative metric register must be able to hold a number which is 6 times the largest branch metric used in the system. By this rule, the use of 7-bit branch metrics leads to the use of 10-bit cumulative metrics.

The output of the ACS cell consists of a 10-bit number giving the updated cumulative metric for the node, and two bits identifying the converging path to be selected at the node in accordance with the Viterbi algorithm. The new cumulative metrics go back to the appropriate ACS cells, the select variables go to the path memory unit.

The add-compare-select cell is shown in Figure 4.23. This circuit is designed to perform the add-compare-select function for a node to which four branches converge. This is admittedly one of the places at which the pragmatic code, in which only two branches converge to a node, would be considerably simpler. The progressive adder (PROG ADDER) circuit adds a transitional metric (TM) to a previous cumulative metric to form a new path metric (Z an ZZ). The difference between Z and ZZ will be explained in the discussion of the progressive adder.

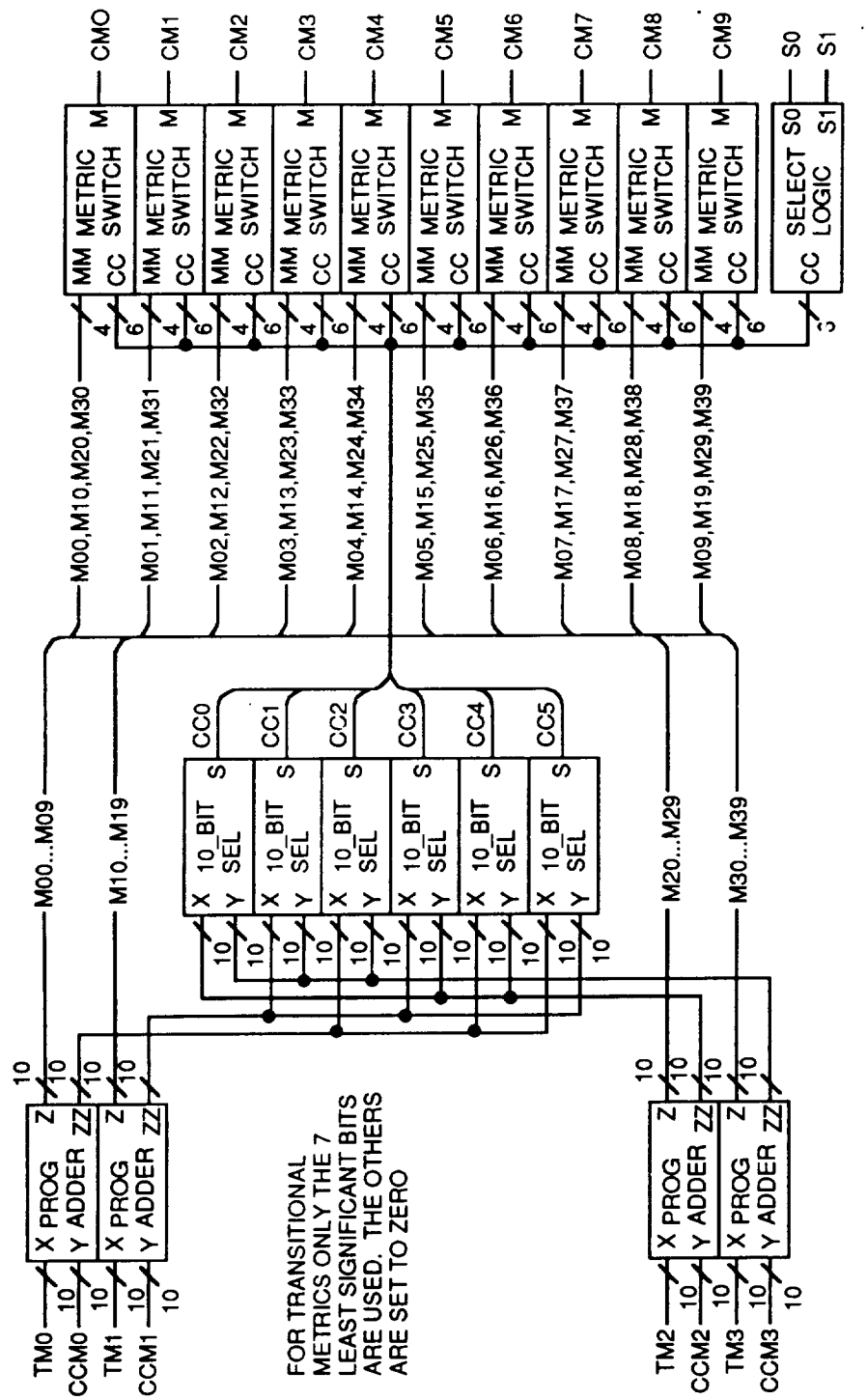

Figure 4.23. ACS Cell.

The four progressive adders generate path metrics for four contending paths. Each 10-bit select (10_BIT SEL) unit compares two of the metrics and generates a bit (CC) indicating which of the two metrics is least. Six comparisons, all six combinations of two of the four metrics are compared tc identify the least of the four metrics. This allows the four-way comparison to be made in the number of cycles required for a two-way comparison, since all six comparisons are performed in parallel. The more conventional approach of performing two two-way comparisons (in parallel) and then a final comparison would require twice as many cycles. As will be shown later, the strategy of the design of this decoder makes it extremely desirable that the add-compare select loop be kept as tight as possible, which is why the six-way comparison strategy was used.

The metric switch is really a unique form of a multiplexer. The ACS cell must select one of four 10-bit path metrics (from the progressive adders) to be the next node metric. Each of the metric switches performs a four-way switching operation for one of the ten bits of the metric. The logic of the metric switch, shown in Figure 4.24, is designed to make the correct selection based on the results of the six comparisons, CC0 through CC5. The select logic, shown in Figure 4.25, is designed to convert

the six comparison results into two bits which identify which of the four bits was selected.

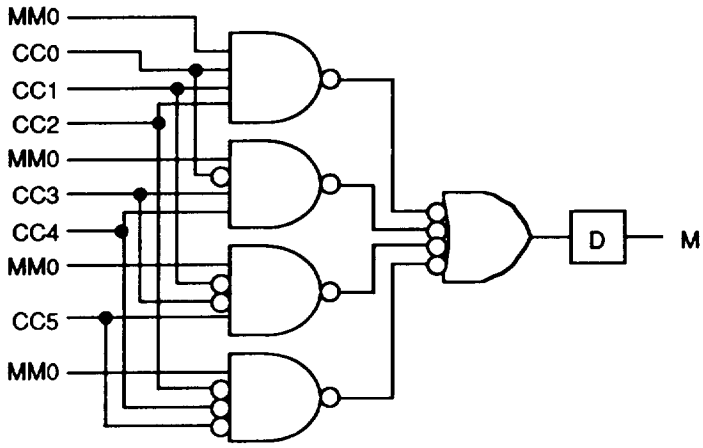

Figure 4.24. Metric switch.

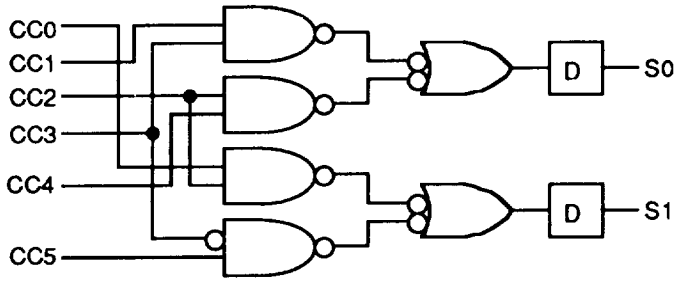

Figure 4.25. Select logic.

4.3.1 The Progressive Adder and the 10_bit Select

The purpose of the progressive adder is to add a transitional metric, provided by the metric calculation unit, to a previous cumulative metric, to generate a new

path metric. The purpose of the 10-bit selector is to compare two path metrics. The progressive adder and the 10-bit selector were designed to work together.

The progressive adder, shown in Figure 4.26, is built on the same pipelining strategy as the 5-bit adder, except that each stage performs the addition of 2-bits, rather than 1. The 2-bit adder is shown in Figure 4.27. The use of the 2-bit adder requires significantly more hardware than the 1-bit adder, but helps to minimize the number of clock cycles in the critical add-compare-select loop. In the pipelined addition method, the less significant bits become available before the most significant bits. In most cases, the least significant bits are simply held until the complete result is available, however, in this case, there is an advantage to allowing the 10-bit selector to receive the lower order bits as soon as they are available. This will also help minimize the number of cycles in the ACS loop. The output Z provides all of the bits of a sum at the same time. The outputs ZZ provide the bits of the sum as they become available. The latches on the outputs Z are to cause the new metric to become available at the same time the output as the corresponding output of the selector.

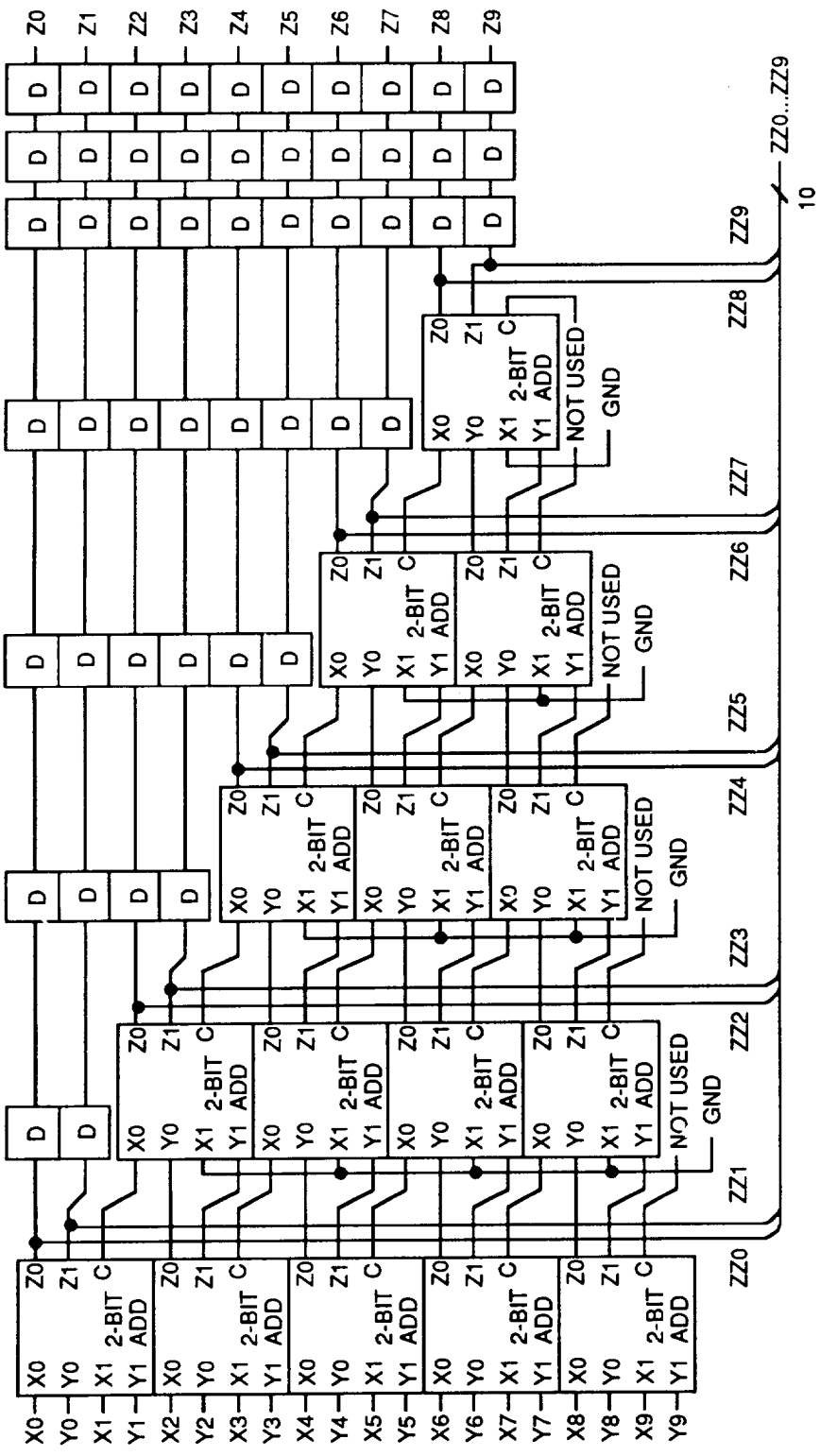

Figure 4.26. Progressive Adder.

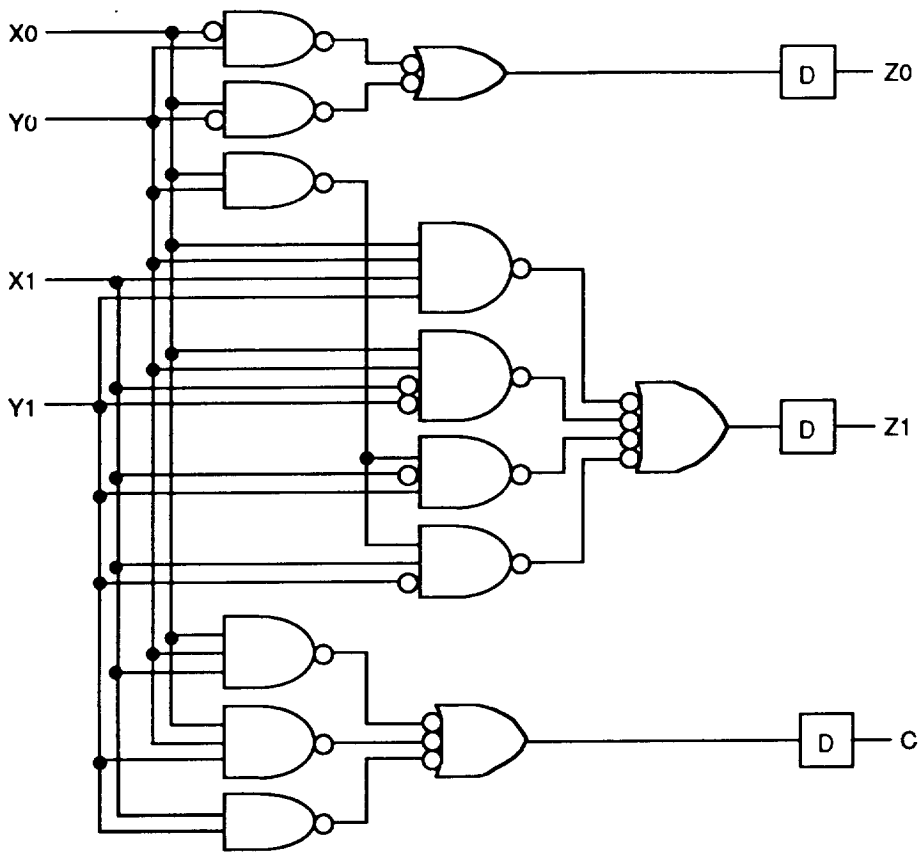

Figure 4.27. 2-bit adder.

The 10-bit comparator, shown in Figure 4.28, is also pipelined. The principle used is that the comparator makes a comparison on the basis of the pair of bits which it has most recently received from the progressive adders. Because the less significant bits are received sooner, this decision will be changed if the more significant bits, received later, indicate a different decision. If the two numbers are equal in the most recently received pair of

bits, then the comparator retains the previously arrived at decision.

Each stage of the 10-bit comparator is a 2-bit selector (2_BIT SEL) as shown in Figure 4.29. The inputs X0, Y0, X1 and Y1, are pairs of bits from X and Y, the two numbers to be compared. The output SY means that Y should be selected on the basis of the X and Y inputs to the current stage. The output SSY means that Y should be selected based the basis of the information received at all previous stages. The output EQ means that the 2-bit inputs to the current stage are equal, i.e., X0 = Y0 and X1 = Y1. As can be seen, the inputs PSY, PEQ, and PSSY are simply the corresponding signals, SY, EQ and SSY from the previous stage.

When the 10-bit comparator compares two metrics, the previously described process is applied to the nine least significant bits of the two numbers. If the two numbers differ in the most significant bit, the decision is reversed by the 3 input exclusive OR gate, following the last 2-bit selector stage. The reason for this is that the decoder uses the idea of Hekstra [19] for avoiding metric overflow. This allows that the arithmetic of the cumulative metrics can be modulo-N, where N is a number which is at least twice as large as the largest difference possible between any two metrics. If it is known that it

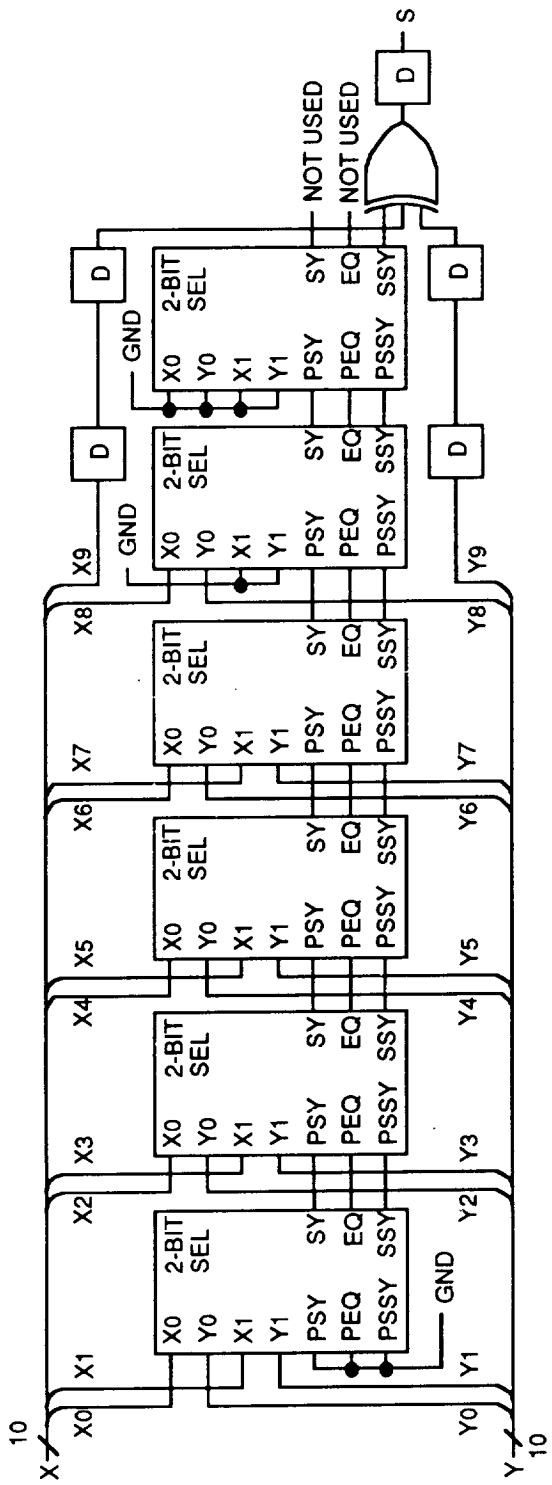

Figure 4.28. 10-bit Comparator.

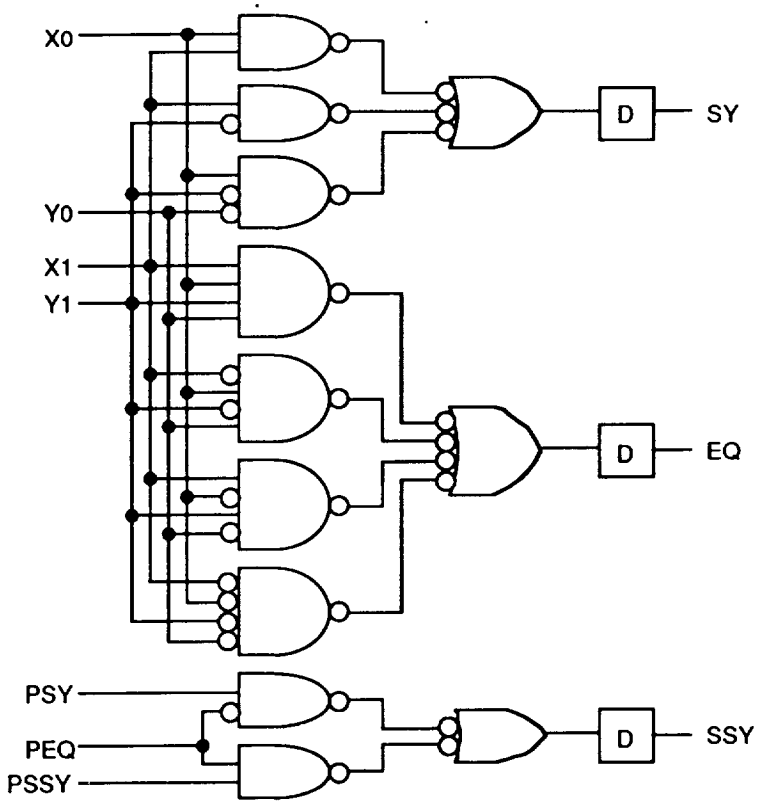

Figure 4.29. 2-bit selector.

is impossible for two metrics to be more than half a cycle apart on the modulo-N circle, then there is no ambiguity as to which is the greater. To illustrate the principle, suppose we are comparing two running totals which we know can never differ by more than 10. We could then store the numbers in modulo-20 registers, but compare the numbers in modulo-10. If both the numbers are greater than 10, or both are less than 10, then the comparison is correct. If only one of the two numbers is less than ten, then the decision must be reversed, thus a non-zero digit in the

ten's column is a signal that the comparison is opposite. The add-compare-select unit applies this principle to the cumulative metrics, except that the storage register is modulo-2^{10}, and the comparison is modulo-2^{9}.

4.3.2 The ACS Feedback Loop

The add-compare-select loop introduces feedback into the Viterbi algorithm, and in this important respect is different from the metric calculation circuit. The add-compare-select loop has been seen to limit the extent to which the Viterbi algorithm can be sped up by pipelining [11]. The important difference between the add-compare-select operation and the metric calculation operation is that the metric operation depends only on current data. Every time a signal vector is received, the metric calculator calculates a set of M metrics, where M is the number of signal vectors, in this case 8. Because the current metric calculation depends in no way on the result of previous calculations, there is no reason why a current metric calculation cannot begin its progression into the pipelined metric calculator as soon as the previous metric calculation has been clocked into a subsequent stage of the pipeline. In this way, the sets of metrics are generated at the rate at which new symbols are clocked into the

decoder, a rate which is limited only by the propagation time between latches in the pipelined circuitry.

In the add-compare-select operation, the cumulative metric, by definition, depends on the result of the previous calculation. A subsequent symbol from a given convolutionally encoded sequence cannot be processed until the calculation of the cumulative metrics associated with the previous symbol is complete. This implies that the rate at which symbols can be processed is limited by the speed at which the ACS operation can be completed.

The high-speed codec design circumvents the limitation imposed by the ACS loop by making a small modification to the coding standard. This works for the following reason. Although the add-compare-select operation cannot process a symbol from a convolutional code sequence until the processing of the previous symbol from the same sequence is complete, the pipelined hardware can begin the processing of a symbol from other independent code sequences on the immediately following clock cycles. Thus if there are Ω pipeline stages in the ACS calculator, the decoder will process Ω independent code sequences concurrently. Each pipeline stage of the ACS unit will hold a calculation in progress associated with a symbol from a different sequence. The parameter Ω will be referred to as the overlap factor. Figure 4.2a shows the standard

convolutional encoder for the 16-state Ungerboeck code. Figure 4.2b shows the modified convolutional encoder. The only difference is that the modification replaces the single delay units with multiple delay units, which delay the input by Ω clock cycles, as opposed to only 1 clock cycle. The effect of this is that the modified encoder is actually encoding the data onto Ω independent code sequences. The independent sequences follow each other in rotation, while symbols from the same sequence follow each other by Ω clock cycles. Metrics from the metric calculation unit arrive at the ACS unit according to the same pattern, which is exactly what is needed to make the decoder function properly. The metrics associated with a symbol arrive at the ACS unit just as the ACS calculation associated with the previous symbol of the same sequence is complete. Meanwhile, the same hardware is being used to process the other independent sequences.

The path memory unit must also be modified to accommodate the modified coding standard. Note that the basic cell of a generic path memory consists of a multiplexer followed by a latch as shown in Figure 4.30a. The modification required is exactly the same as the modification introduced to the convolutional encoder. The single latch is replaced by an Ω stage delay as shown in

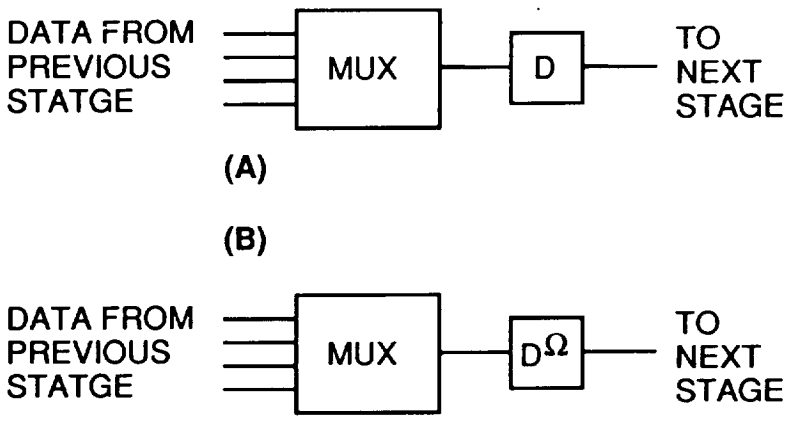

Figure 4.30. a) path memory cell. b) modified path memory cell.

Figure 4.30b. The reason this works is as follows. The purpose of the multiplexer is to select the data to be loaded into the memory. The multiplexer selects the data from the previous stage of the path selected in accordance with the decision made by the ACS unit. Since there are Ω independent sequences, only one out of every Ω decisions pertains to a given code sequence. Thus the additional stages of memory cause the data associated with a given sequence to arrive at the switches of the next stage of the path memory at the same time as the decisions associated with the particular sequence are generated by the ACS unit.

The use of multiple independent coding sequences allows a speedup in operation with a less than proportional expansion in hardware. By allowing only a single code sequence, pipelining the ACS operation does not change the

fact that symbols can only be processed at the rate at which the ACS operation can be performed. Although the exact speeds involved depend on the technology employed and the specific structure of the ACS circuitry, it stands to reason that if latches can be installed at the approximate half-way points in all of the critical paths of the ACS circuitry (Ω=2), the data rate of the overall system can be approximately doubled with only a slight increase in the hardware of the ACS unit. Certainly, a twofold increase in speed has been obtained without a twofold increase in ACS hardware. The effect of this strategy on the memory hardware is that where there was formerly a latch and a MUX, there are now two latches and a MUX. Since a latch consists of two logic gates, and a (two-way) MUX consists of three, the hardware in the path memory expands by approximately 7/5, while increasing the speed of the system by a factor of two. In the case of a trellis with four branches expanding into a node, the benefits of this design approach are comparable. For Ω other than 2, it is a matter of simple arithmetic that the expansion in hardware is less than the increase in speed. It is, however, desirable to minimize the length of the ACS path, since this ultimately drives the size of the memory. In the codec presented here, with the rule of no more than three logic gates between any pair of latches, the ACS operation

came out to require 9 clock cycles, therefore an overlap factor of Ω=9 was employed.

An approach to Viterbi decoder architecture that has received some attention in recent literature, is the combined trellis stage approach of Fettweis and Meyr [11, 12]. This approach is termed as a linear scale solution, because it offers an M-fold increase in speed in return for an M-fold increase in the volume of the hardware of the ACS unit. It is explained later that adopting the combined trellis architecture in the place of the simple trellis architecture multiplies the volume of the ACS unit by the number of states of the trellis code, and the linear scale solution is obtained thereafter. Fettweis and Meyr [11, 12] have applied their architecture to a 4-state binary code. The high-speed TCM decoder does not use the combined trellis architecture, because this approach introduces considerable complexity, which is compounded for codes of greater numbers of states.

The combined trellis stage approach consists of forming a super trellis stage, which shows branches for all of the state transitions which the encoder can make in M steps of operation, unlike a standard trellis stage, which shows only the transitions which the encoder can make in one step. The authors of the combined trellis architecture use the terminology, 1-step trellis to apply to the

standard trellis and M-step trellis to apply to the super trellis. Presumably, if larger hardware can be built to perform the ACS operation for the super trellis stage, the data rate could be increased, since an M-step trellis represents an M-fold increase in data, while the ACS operation for the super trellis should require only slightly longer than the ACS operation for the simple trellis. To apply this approach, metrics must be calculated for the branches of the combined trellis stage, each of which now consists of M symbols. Also, the operation of combining the trellis stages increases the number of branches which connect into each node, and leads to the formation of parallel branches, multiple branches which connect the same pair of states. If the parallel branches are eliminated prior to the super trellis ACS operation, the number of branches converging into a single node is limited to the number of states. Therefore, the difficulty of applying the super trellis approach grows substantially with the number of states of the code.

The combined trellis architecture uses conventional 1-step ACS units to calculate the metrics for the branches of the super trellis. To obtain the desired increase in the data rate, the 1-step ACS units must be paralleled by a factor of M, and the incoming data (symbol metrics) must be blocked to drive the parallel units. Fettweis and

Meyr [11,12] recommend that the resultant increase in hardware be minimized by the interleaving of pipelined architecture, that is an ACS unit which is pipelined in P segments can be responsible for P ACS calculations. Furthermore, the 1-step ACS units are combined with 1-step path memory of length M, so that the complete structure serves to preselect parallel branches, prior to the M-step ACS operation. The net result of all this is that the ACS architecture for the super trellis consists of an M-1 by S (S is the number of states) array of 1-step Viterbi decoders. Thus, for a code with a larger number of states, the additional hardware can be extensive. Furthermore, the M-step ACS unit and the M-step path memory unit must be designed to handle up to S converging branches.

For the 4-state binary code, the complications of the super trellis approach are constrained within reasonable limits. For the 16-state Ungerboeck approach, a less complicated approach was needed, therefore the previously discussed, independent code sequence method was adopted. The independent code sequence multiplies the size of the path memory, while the expansion of the ACS hardware is limited to the introduction of latches needed to implement pipelining. The super trellis approach introduces an M-fold increase in ACS hardware, and the additional memory necessary to implement the array of 1-step Viterbi

decoders. For both approaches, the exact degree of hardware expansion (taking into account both the ACS unit and the path memory) is highly dependent on the code adopted, however, in the case of TCM with 16-states or greater, I believe that the independent code sequence approach will require less total hardware.

4.4 The Path Memory Circuit

The path memory circuit consists of a number of identical stages, as shown in Figure 4.31. The number of stages corresponds to the number of branches which the decoder stores in its memory of the maximum likelihood path to each state. This design parameter is referred to as the decoder depth or the trace-back depth. The performance of the decoder improves significantly with decoder depth up to a point that depends on the individual code. At this point, very little improvement will result from further increasing the decoder depth. There is no known analytical means for determining the required decoder depth, so this parameter is usually found empirically. A commonly used rule of thumb is that the decoder depth should be five times the constraint length of the code. This rule applies to codes in which two branches converge into a node. For codes in which more than two branches converge, and for

punctured codes, a longer decoder depth is usually required. In fact, manufacturers offer decoders which operate in either a short trace-back mode or a long trace-back mode, recommending that the long trace-back for punctured operation. The decoder depth for the high-speed decoder was found by experimentation with BOSS. Figure 4.31 shows that the flow of data in the path memory follows the trellis structure of the code. The connections are shown in more detail in subsequent illustrations. Each stage of the path memory, shown in Figures 4.32 and 4.33, consists of 16 identical path cells, each of which is responsible for one node of the trellis. Each path cell consists of a dual 4 to 1 MUX followed by 9 latches as shown in Figure 4.34. The select inputs, S0 and S1 are generated by the add-compare-select circuit. There is a different pair of select inputs for each state; however, the same set of select signals is used at each stage of the path memory. Since four branches converge into each node of the trellis, each branch represents two bits of originally encoded data. The inputs D00 through D31 represent the data associated with the converging paths, two bits from each of four previous nodes. The outputs Q0 and Q1 represent the data from the selected path. Figures 4.32 and 4.33, show how the path cells of a given stage are connected to the previous stage. Here N denotes the an individual stage of the memory, N-1 denotes a previous

stage. Data lines are indicated by Q, select lines are indicated by S.

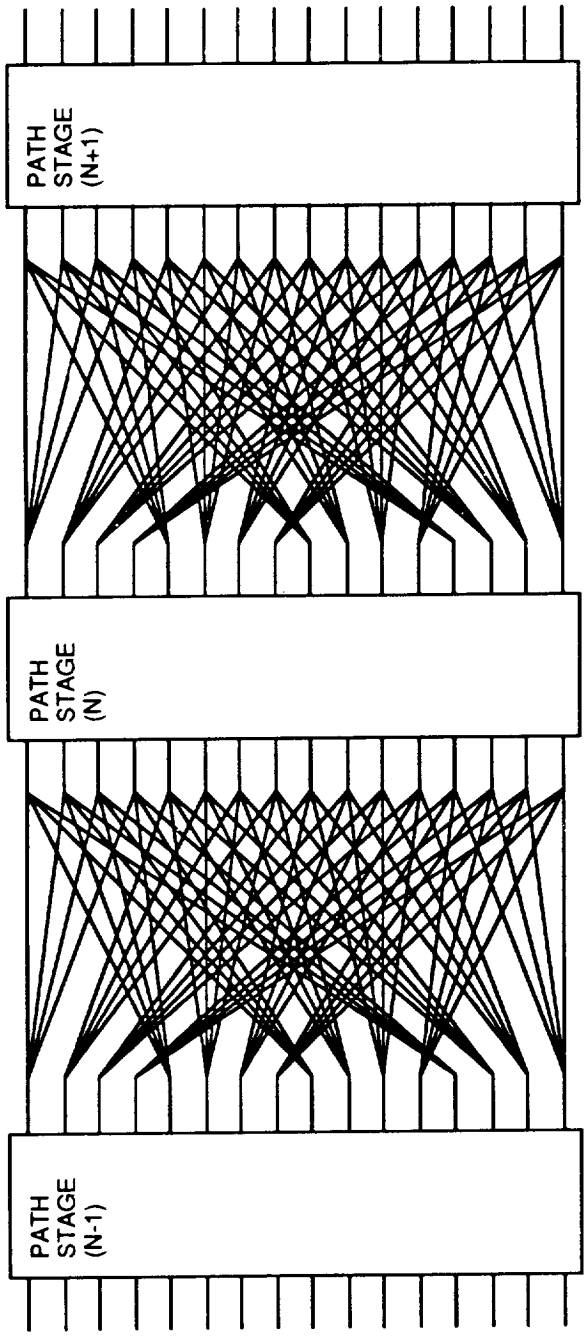

Figure 4.31. Path Memory Circuit.

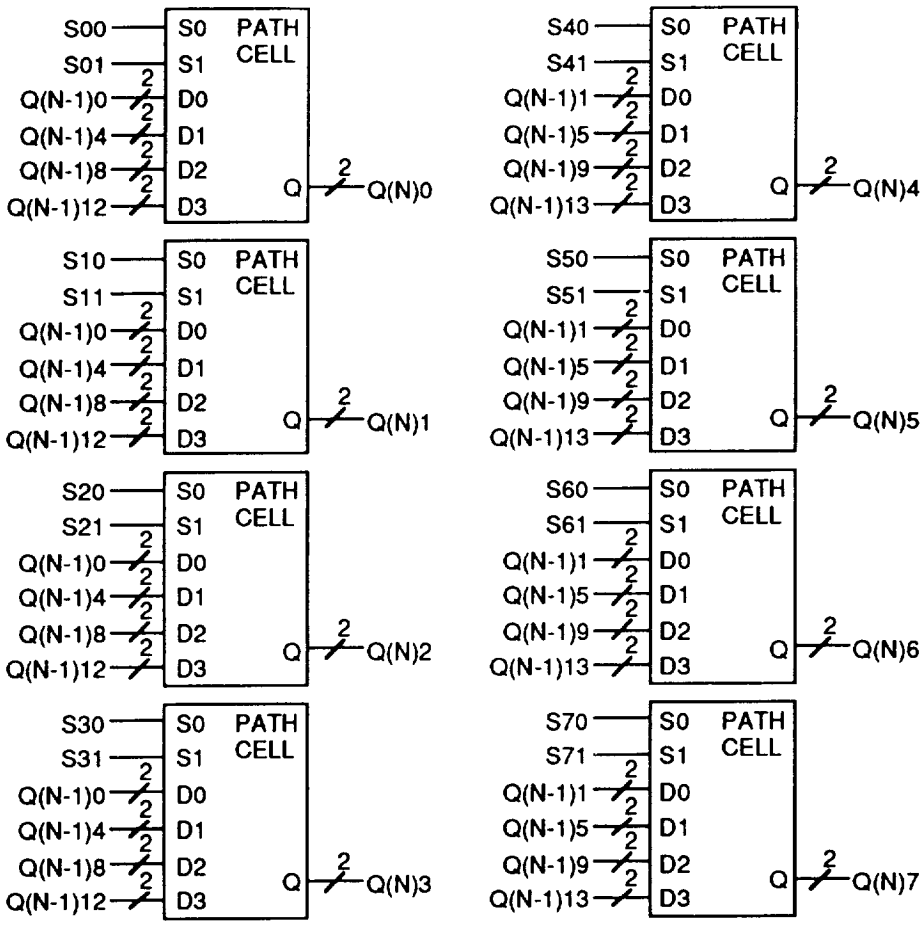

Figure 4.32. Path memory stage, part 1 of 2.

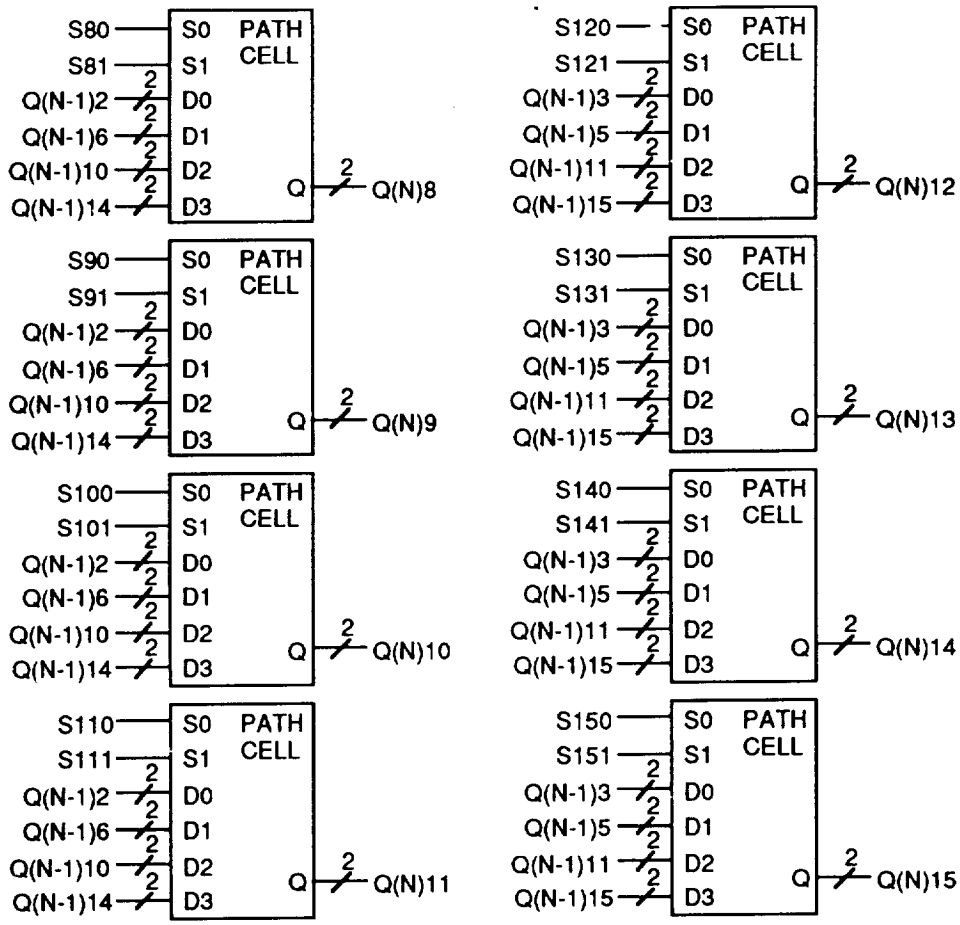

Figure 4.33. Path Memory Stage, part 2 of 2.

C-3

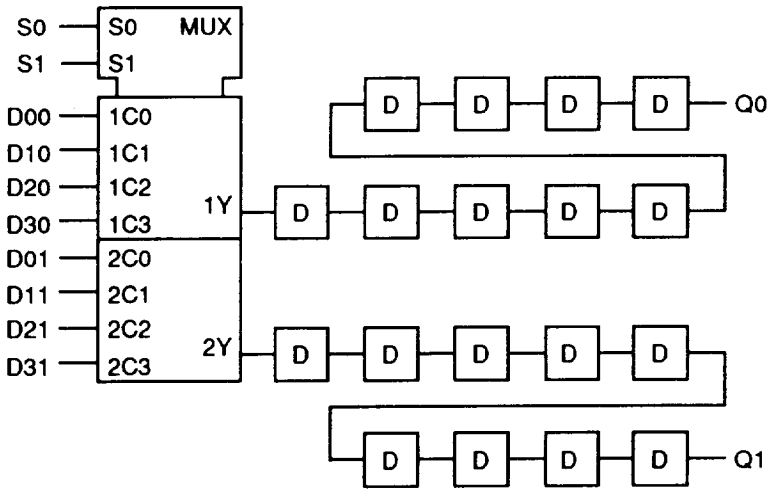

Figure 4.34. Path cell.

4.5 Testing The High-Speed Codec

The block-oriented systems simulator (BOSS) was used to select the design parameters for the high-speed decoder, to test the bit error rate performance, and to verify the final logic. After deciding upon the coding standard, the next consideration was the resolution of the I and Q inputs, and the resolution of the metrics. Table 4.1 shows the bit error rates of various the resolutions of signal vectors and metrics, obtained at by simulating at Es/N0 = 10dB. As can be seen, the performance of any particular combination cannot be easily predicted by studying the effect of I and Q quantization and metric quantization independently. Since the pragmatic standard stands a good

chance of becoming the defacto coding standard of the future, it was considered necessary that the high-speed decoder should achieve performance comparable to the pragmatic decoder at a bit error rate of 10^{-5}. To do this, a bit error rate of less than 3×10^{-6} at $E_S/N_0 = 10dB$ was necessary. As can be seen from the table, the 16-state Ungerboeck code accomplishes this with 5-bit I and Q and 7-bit metrics. Unfortunately, it was difficult to obtain reliable results, since a trial of 5 million symbols is barely sufficient to measure a bit error rate of 10^{-6}, and this was taxing the computer time available for the project. For 8-bit I and Q, the simulation detected no errors in a trial of one million symbols, showing that it is not unreasonable to expect performance which is slightly better than that of pragmatic TCM.

4.5.1 Selection of Quantization Parameters

Signal vector quantization and metric quantization are not interchangeable. Usually the requirement for metric quantization is driven by the degree of signal set quantization. For example, the use of N-bit I and Q components results in 2N-bit square difference terms, two of which are added to produce a 2N+1 bit metric. Therefore, if 4-bit I and Q quantization were decided on, a

9-bit metric represents no further compromise of performance, that is nine bits is the maximum useful metric resolution for 4-bit I and Q, whereas 11-bit I and Q is the maximum useful metric resolution for 5-bit I and Q.

METRIC RESOLUTION \ I AND Q RESOLUTION	4-BIT	5-BIT
4-BIT	4.15E-5	
5-BIT	6.5E-6	1.25E-5
6-BIT	3.7E-6	4.8E-6
7-BIT	3.4E-6	2.1E-6
8-BIT	2.8E-6	1.8E-6
9-BIT	3.1E-6	1.2E-6
10-BIT		
11-BIT		1.2E-6

Table 4.1. Decoder bit error rate at Es/N0=10dB.

Table 4.1 shows the bit error rate as a function of metric resolution and I and Q resolution, at E_S/N_0=10dB. The results of Table 4.1 show that if the metrics are quantized to a low level of resolution, an increase in I and Q resolution will not necessarily result in an improvement in performance unless also accompanied by an increase in metric resolution. As can be seen, with 5-bit metrics the performance of 5-bit I and Q is worse than the performance of 4-bit I and Q. Also, we can see from the chart that with 4-bit I and Q, the performance with maximum metrics is 3×10^{-6}, which is comparable to the performance

of the multimode codec, which used 4-bit I and Q and 4-bit metrics. By using 5-bit I & Q, the 16-state Ungerboeck code improves its performance by approximately a factor of two, achieving performance comparable to unquantized pragmatic TCM. These results were based on trials of 5 million symbols, except for the three results presented for 4- and 5-bit metrics, which were based on 1 million symbols. One of the problems encountered is that 5 million symbols may not have been a sufficient simulation length to obtain confident results. In running the final performance tests for the decoder, a different random sequence was used and a bit error rate of 3.8×10^{-6} was obtained. The variance for the final performance trial, which also used 5 million symbols was calculated at 1.2×10^{-6}. In light of this, a decision to use 5-bit I and Q and 7-bit metrics probably represents a worst case scenario. However, since the logic has been worked out for these parameters, designing a simplified version of the circuit, if desired, should not be a problem.

Quantization significantly affects the size of the overall machine. For example, if 4-bit I and Q are used, then the 4-bit square circuit of Figure 4.17 rather than the 5-bit square circuit of Figures 4.14 through 4.16, and as can be seen the 4-bit square circuit is considerably smaller. With 6-bit I and Q, the design of a specialized

metric calculation would be even more difficult, and it is at this point that a metric RAM would be considered. Metric quantization affects the size of the add-compare-select unit, while the number of cycles in the add-compare-select unit dictates the size of the memory. Therefore, it is extremely worthwhile to let the metric resolution be the minimum required to achieve the desired performance. From the chart we see that 4-bit I and Q with 4-bit metrics is not an acceptable option for this project, since the resulting performance is not even within the order of magnitude of the desired performance. The use of 4-bit I and Q with 6-bit metrics could be an acceptable option, although the performance falls slightly short of pragmatic TCM. The use of 5-bit I and Q with 7-bit metrics achieves performance comparable to pragmatic TCM.

4.5.2 Simulation Design

Several models of the high-speed decoder were built in BOSS, the two most important of which are the logic level simulation and the high level simulation. The logic level version was built solely out of logic gates, to verify the logic as presented in the illustrations in this chapter. The higher level version was constructed out of higher level modules, some of which were written in FORTRAN code.

This approach was necessary because, due to the way BOSS works, the time required to complete simulations of the logic gate model would have made performance testing infeasible. The higher level model requires shorter simulation times and allows the degree of quantization to be easily changed, since it is controlled by a numerical parameter. Changing the degree of quantization requires a complete change in structure of the logic model. Once performance results were obtained for the high level model, the design parameters were decided upon and the logic level model was built. That the logic level model is functionally identical to the higher level model was verified through shorter simulation runs, specifically by showing that identical random input sequences produce identical error counts.

4.5.3 High Level Simulation

The high level simulation is shown in Figure 4.35. The module ARCH DATA generates signal vectors to which Gaussian random vectors are added to simulate the effect of noise. The module IQ CONVERT quantizes the I and Q components of the received signal vector on a scale of 0 to L-1, where L is the number of quantization levels, a

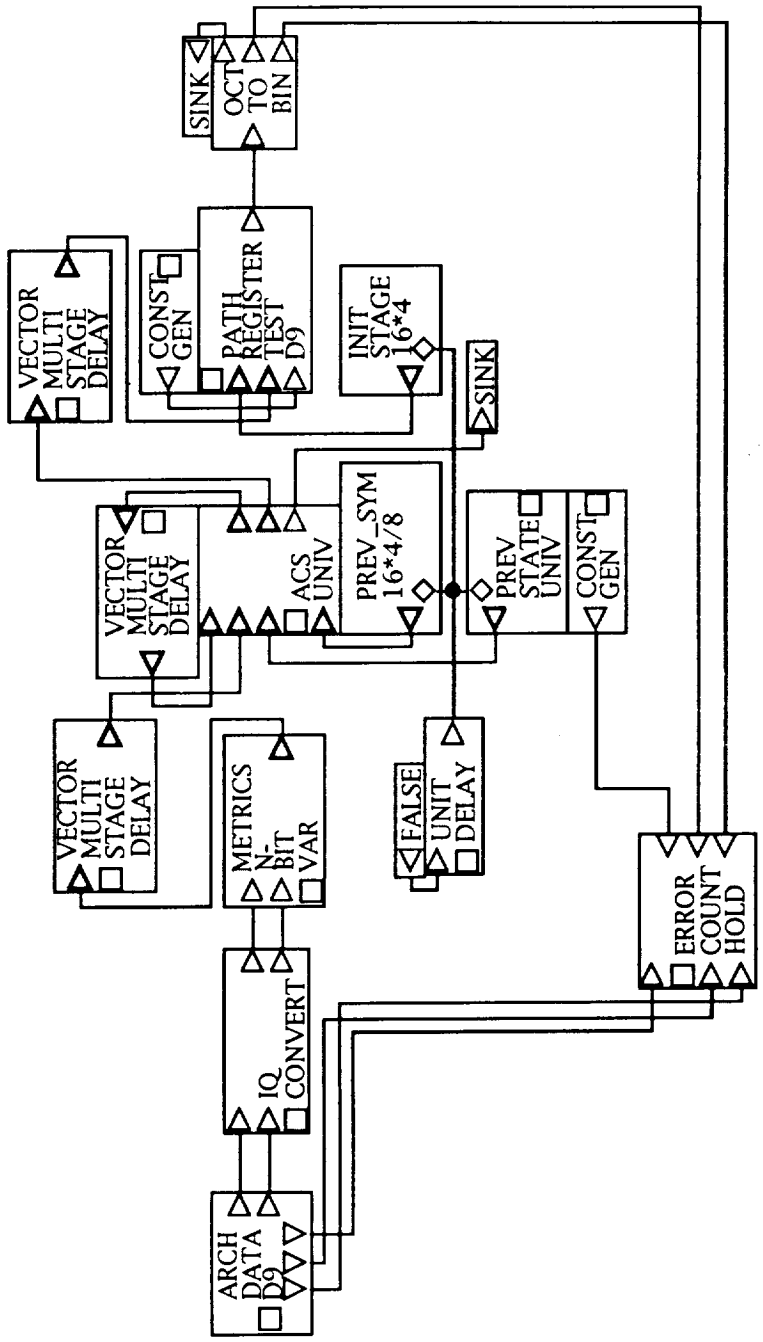

Figure 4.35. High Level BOSS Simulation.

controllable parameter. The metrics are calculated using strictly integer arithmetic; however, before being sent to the ACS module, they are divided by a reduction factor and then rounded to another integer. The reduction factor controls the precision of the metrics used by the ACS unit. If a reduction factor of 1 is used, the precision of the metrics is the maximum useful precision given the degree of I and Q quantization. If a different reduction factor is used, the precision of the metrics (in bits) is 2N+1-log2(R), where N is the number of bits used for I and Q and R is the reduction factor.

The module ACS UNIV performs the add-compare-select function for the Viterbi decoder, and is implemented as a BOSS primitive, i.e., the module is defined in FORTRAN code. This module is written to work for any trellis code defined by the previous symbol table and previous state table, which in this case are supplied by the modules PREV_SYM 16*4/8 and PREV STATE UNIV, respectively. These modules are also implemented as primitives. The previous symbol table is specifically for the code being used here, the previous state module is written to work for any shift register convolutional code, given the number of states and the number of input bits. The path register module is also designed to work for a variety of codes. Once the data is clocked out of the path register module, it is converted to

binary form, by the module OCT TO BIN. The data is then compared to the original data to obtain an error count. In the high level simulation, delays are introduced to correspond with the delays introduced by pipelining in the logic level simulation. This is necessary to assure that at any time, every part of the high-level simulation is handling exactly the same data as the corresponding part of the logic level simulation.

4.5.4 Logic Level Simulation

The top level diagram of the logic level simulation is shown in Figure 4.36. The logic level simulation uses exactly the same data and error counter as the high-level simulation. The 5-bit receiver quantizes the received I and Q components to 32 levels and gives the output in binary form. Here, the modules 7_BIT METRIC GENERATOR, 10_BIT ACS UNIT, and PATH UNIT D9, correspond to the three blocks of the top level diagram of the decoder itself. They are implemented in basic logic which corresponds to that illustrated in the diagrams of this chapter. Short runs, using controlled pseudo-random sequences verified that the logic level simulation functions exactly the same as the high-level simulation.

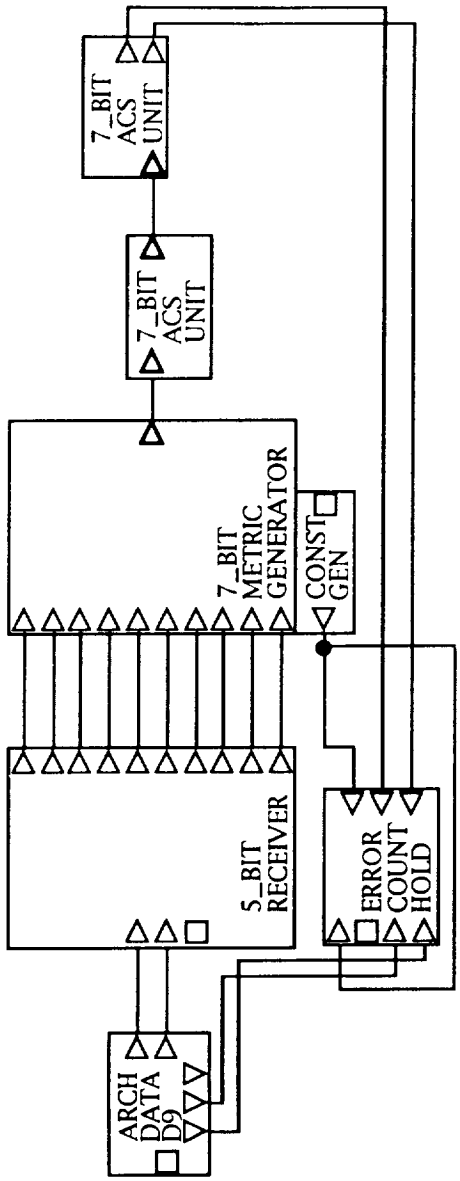

Figure 4.36. Logic Level BOSS Simulation.

4.6 Conclusion

A complete logic design has been presented for a Viterbi decoder to decode the rate 2/3 8-PSK 16-state Ungerboeck TCM code. To achieve high-speed operation, the design has been pipelined throughout, with a maximum of three logic gates between any pair of latches. Higher speeds with slightly greater hardware volume could be obtained by using fewer than 3-gates between latches. Simulations were employed to determine that the design should use 5-bit I and Q components and 7-bit branch metrics. Special circuitry was designed to calculate the branch metrics using Boolean Algebra. A simple approach for circumventing the ACS feedback loop was presented.

The performance of the high-speed decoder is shown in Figure 4.37. The variance of the result was calculated by dividing the simulation time into ten equal intervals, and calculating the sample variance as $\sigma_S = \sqrt{\frac{1}{9}\sum_{i=1}^{10}(x_i-\bar{x})^2}$. The variance of the mean was calculated as $\sigma=\frac{\sigma_S}{\sqrt{10}}$. At E_S/N_0 = 10dB, it can be seen that the decoder has nearly approached the asymptotic error rate for pragmatic TCM, and achieves performance equivalent to quantized pragmatic TCM.

Although the high-speed TCM decoder uses the 16-state Ungerboeck code, the architectural approach could also have been applied to other coding standards, such as pragmatic TCM. The performance of the high-speed design was simulated using BOSS. The results of the simulation are shown in Figure 4.37. At a bit error rate of 10^{-5}, the performance of the high-speed TCM decoder is comparable to the performance of pragmatic TCM. The logic of the complete system has been verified using BOSS. The high-speed decoder design presented here is ready for VLSI development.

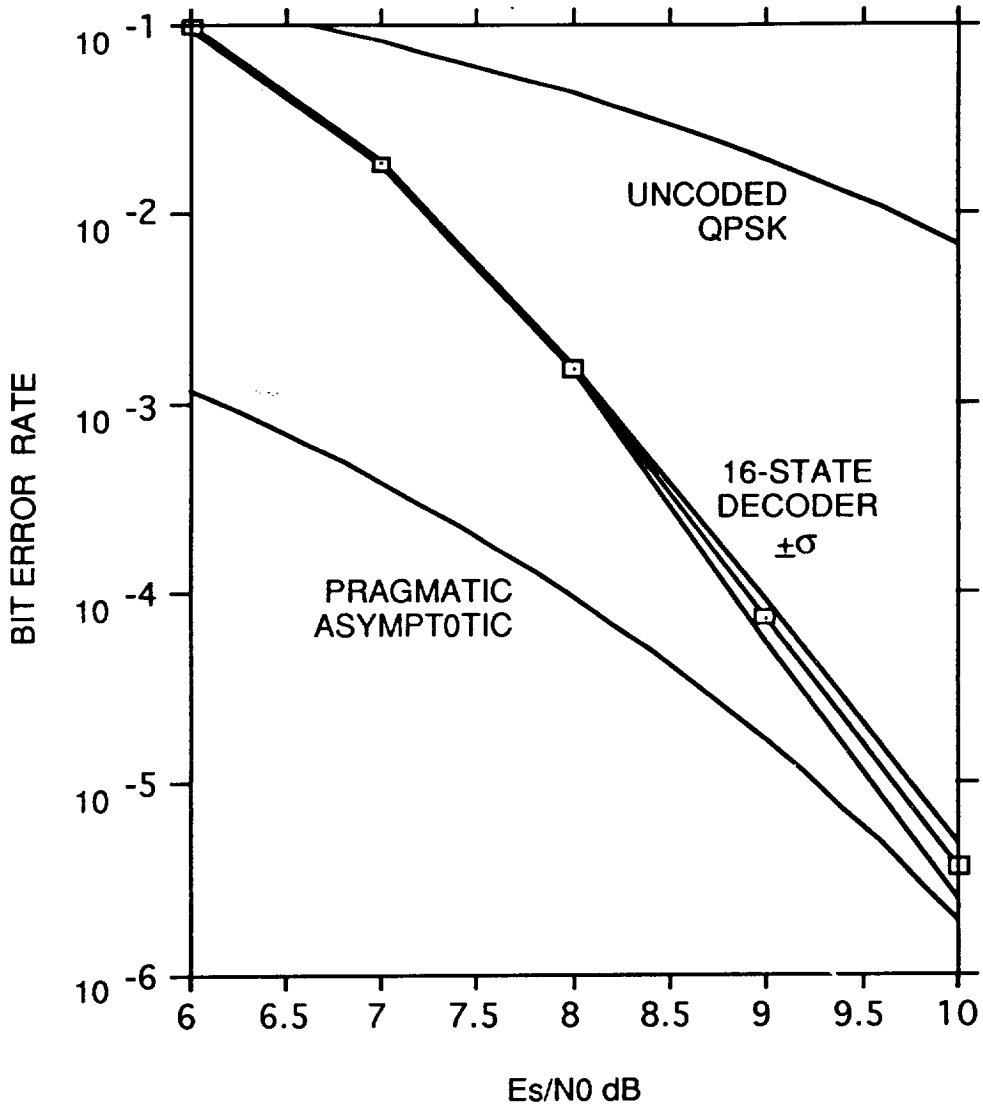

Figure 4.37. Performance of the high-speed decoder.

5. CONCLUSION

5.1 Summary

The design for the high-speed decoder is ready for VLSI development. Based on data obtained from simulations, it is recommended that the high-speed decoder be built to process the rate 2/3 8-PSK 16-state Ungerboeck [2, 3] code, receive signal vectors in the form of 5-bit I and 5-bit Q, generate 7-bit branch metrics for the decision unit, which will retain 10-bit metrics, and use the alternative to cumulative metric rescaling suggested by Hekstra [19]. It is also recommended that the decoder should have a survivor memory of 40, which can be reduced to 30 if additional circuitry is added to select the output data from the minimum metric path in the path memory.

It is by no means suggested that the decoder would not be successful if alternative design parameters were used. For example, the design strategies presented here could have been applied to a pragmatic TCM decoder, or even a multimode decoder. The motivation behind the use of the 16-state Ungerboeck code is that it would allow error correcting performance equivalent to that of pragmatic TCM, with less hardware volume. Another reason for choosing the 16-state Ungerboeck code for this project is to gain additional knowledge. Due to the wide acceptance of pragmatic TCM, the coming decade should see ample data to

document the performance of this code. Based on the bit error spectrum calculation, the 16-state Ungerboeck code should out perform pragmatic TCM at bit error rates < 10-6, where computer simulation data is difficult to obtain. Therefore, construction of a chip to implement this standard would allow the acquisition of data which might not be attainable otherwise. The pragmatic standard has the advantage of wide acceptance, and relatively easy integration into existing systems. Other changes in the design parameters might result in only a slight compromise in performance, such as using 4-bit I and Q, rather than 5-bit I and Q.

The development of the high-speed codec proceeded as follows:

1) As proof of concept, a logic gate BOSS model was built, using the strategies presented here, but with 4-bit I and Q and four bit branch metrics. This model receives no attention in this report.

2) Higher level BOSS simulations were constructed to determine performance, at Es/N0=10dB, of the decoder as a function of I, Q and metric resolution, using a decoder depth of 80. It was determined that the final design would use 5-bit I and Q and 7-bit branch metrics.

3) The logic gate model was upgraded to the new design parameters.

4) Additional tests were conducted to determine the necessary decoder depth.

5) Final performance tests were conducted to determine performance from E_S/N_0 from 6dB through 10dB.

5.2 Suggestions for Further Research

It is almost certain that the Telemetry Center will develop a VLSI implementation based on the logic design presented here. Additional research will be done to attain the maximum attainable clock speed, and to select a substrate technology. CMOS is the most likely candidate for substrate technology.

The bit error spectrum technique has potential for a much wider variety of codes than are presented here. Other code rates and modulation formats, or more powerful codes could be investigated. Also, the C language code could be ported to a workstation more powerful than a PC. Some additional theoretical work is needed to determine the conditions under which the union bound summation will or will not converge. This could be based on the fact that the number of paths grows exponentially while the Q() function, which is used to calculate the probabilities of individual error events, can also be bounded by exponential expressions. Then the standard conditions for convergence

of infinite series could be applied. Research in convolutional codes will also lead to research in concatenated codes and the effect of interleaving on convolutional codes.

REFERENCES

[1] Viterbi, A.J., "Convolutional Codes and their Performance in Communication Systems," IEEE Transactions on Communication Technology, Vol. CT-19, pp. 751-771, October 1971.

[2] Ungerboeck, Gottfried, "Trellis-Coded Modulation with Redundant Signal Sets, Part I: Introduction," IEEE Communications Magazine, Vol. 25, No. 2, pp. 5-11, February 1987.

[3] Ungerboeck, Gottfried, "Trellis-Coded Modulation with Redundant Signal Sets, Part II: State of the Art," IEEE Communications Magazine, Vol. 25, No. 2, pp. 12-21, February 1987.

[4] Ungerboeck, Gottfried, "Channel Coding with Multilevel/Phase Signals," IEEE Transactions on Information Theory, Vol. IT-28, No. 1, pp. 55-67, January 1982.

[5] QUALCOMM Inc., Viterbi Decoder on a Single Chip, K=7, Rate 1/2, San Diego, California, October, 1988.

[6] QUALCOMM Inc., Multi-Code Rate Viterbi Decoder, K=7, San Diego, California, June, 1990.

[7] Viterbi, Andrew J., Jack K. Wolf, Ephraim Zehavi, Roberto Padovani, "A Pragmatic Approach to Trellis-Coded Modulation," IEEE Communications Magazine, Vol. 27, No. 7, pp. 11-19, July 1989.

[8] Carden, Frank, "A Quantized Euclidean Soft-Decision Maximum Likelihood Sequence Decoder: A Concept for Spectrally Efficient TM Systems," Proceedings of the International Telemetering Conference, Vol. XXIV, pp. 375-384, October 1988.

[9] Carden, Frank, and Brian Kopp, "A Quantized Euclidean Soft Decision Maximum Likelihood Sequence Decoder of TCM," IEEE Military Communications Conference, Vol. 2, pp. 279-682, October 1988.

[10] Carden, Frank, and Michael Ross, "A Spectrally Efficient Communication System Utilizing a Quantized Euclidean Decoder," Proceedings of the International Telemetering Conference, Vol. XXV, pp. 575-582, October 1989.

[11] Gerhard Fettweis and Heinrich Meyr, "High-Speed Parallel Viterbi Decoding: Algorithm and VLSI-Architecture", IEEE Communications Magazine, May 1991.

[12] G. Fettweis and H. Meyr, "Parallel Viterbi Algorithm Implementation: Breaking the ACS-Bottleneck," IEEE Transactions on Communication, Vol. COM-37, pp. 785-790, Aug 1989.

[13] Rouanne, Marc, and Daniel J. Costello, Jr., "An Algorithm for Computing the Distance Spectrum of Trellis Codes," IEEE Journal on Selected Areas in Communication, Vol. 7, No. 6, August 1989.

[14] Bellman, R., and S. Dreyfus, Applied Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1962.

[15] Lin, S. and Daniel J. Costello, Jr., Error Control Coding: Fundamentals and Applications, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[16] G.C. Clark and J.B. Cain, Error Correcting Coding for Digital Communications, Plenum Press, New York, 1981.

[17] Forney, G. David, Jr., "Convolutional Codes I: Algebraic Structure," IEEE Transactions on Information Theory, Vol. IT-16, No. 6, pp. 720-728, November 1970.

[18] William Osborne, Frank Carden, Brian Kopp, and Mike Ross, "Multi-Mode Modem/Codec Designs", AIAA Communication Satellite System Conference, 1991

[19] Hekstra, A.P. "An Alternative to Metric Rescaling in Viterbi Decoders," IEEE Transactions on Communications, vol. 37, pp. 1220-1222, Nov. 1989.

[20] QUALCOMM Inc., Pragmatic Trellis Decoder, San Diego, California, May 1992.

[21] Fang, "A Coded 8 MHz System for 140 Mbps Information Rate Transmission Over 80 MHz Nonlinear Transponders," ICDSC, 305-313, 1986.

[22] Shannon, C.E., "A Mathmatical Theory of Communication", Bell System Technical Journal, vol. 27, pp 379-423,623-656, 1948.

[23] Wozencraft, J.M., and R.S. Kennedy, "Modulation and Demodulation for Probabalistic Coding", IEEE Transactions on Information Theory, vol. IT-12, no. 3, pp. 291-297, July 1966.

[24] Massey, J.L., "Coding and Modulation in Digital Communication," Proceedings of the International Zurich Seminar on digital Communications," 1974, pp. E2(1)-E2(4).

[25] Gallager, R.G., "A Simple Derivation of the Coding Theorem and Some Applications," IEEE Transactions on Information Theory, vol. IT-11., pp. 3-18, January 1965.

[26] Parsons, R.D., and S.G. Wilson, "Polar Quantizing for Coded PSK Transmission," IEEE Transactions on Communications, vol. 38, no 9., pp. 1511-1519, September 1990.

[27] Lee, L.N., "On Optimal Soft-Decision Demodulation," IEEE Transactions on Information Theory, vol. IT-22, pp. 437-444, July 1976.

[28] Ross, Michael, Frank Carden and William P. Osborne, "Pragmatic Trellis-Coded Modulation: Using 24-sector Quantized 8-PSK." International Phoenix Conference on Computers and Communications, 1991.

[29] Ross, Michael, William P. Osborne, Frank Carden and Jerry L. Stolarczyk, "Pragmatic Trellis-Coded Modulation: A Hardware Simulation Using 24-sector 8-PSK," Supercomm/ICC, 1992.

[30] Carden, Frank, and Michael Ross, "64-State TCM for Spectrally Efficient Space Communications," NAECON-91.

[31] Zehavi, Ephraim, and Jack K. Wolf, On the Performance Evaluation of Trellis Codes," IEEE Transactions on Information Theory, vol. IT-33, no. 2, March 1987.

[32] Cain, C.G. Clark Jr., & J.M. Geist, "Punctured Codes of rate (n-1)/n and Simplified Maximum Likelihood Decoding," IEEE Transactions on Information Theory, vol. IT-25, pp 97-100, Jan 1979.