Final Report
June 1990 – September 1994
NASA Grant NAG-3-1166

Fault Tolerance in Space-Based Digital Signal Processing and Switching Systems

Protecting Up-Link Processing Resources, Demultiplexer, Demodulator, and Decoder

Robert Redinbo
Department of Electrical Engineering
and Computer Science
University of California
Davis, CA 95616

September 1994

TABLE OF CONTENTS

ITEM **PAGE**

LIST OF FIGURES

LIST OF FIGURES (cont.)

LIST OF TABLES

Final Report
June 1990 – September 1994
NASA Grant NAG-3-1166

Fault Tolerance in Space-Based Digital Signal Processing and Switching Systems

PROTECTING UP-LINK PROCESSING RESOURCES, DEMULTIPLEXER, DEMODULATOR, AND DECODER

Robert Redinbo
Department of Electrical Engineering
and Computer Science
University of California, Davis

ABSTRACT

Fault tolerance features in the first three major subsystems appearing in the next generation of communications satellites are described. These satellites will contain extensive but efficient high-speed processing and switching capabilities to support the low signal strengths associated with very small aperture terminals. The terminals' numerous data channels are combined through frequency division multiplexing (FDM) on the up-links and are protected individually by forward error-correcting (FEC) binary convolutional codes. The front-end processing resources, demultiplexer, demodulators and FEC decoders extract all data channels which are then switched individually, multiplexed and remodulated before retransmission to earth terminals through narrow beam spot antennas. Algorithm based fault tolerance (ABFT) techniques, which relate real number parity values with data flows and operations, are used to protect the data processing operations. The additional checking features utilize resources that can be substituted for normal processing elements when resource reconfiguration is required to replace a failed unit.

The FDM demultiplexer is efficiently implemented by a multirate polyphase filter bank where segments of a uniform channel extraction are combined with a fast Fourier transform section to process input samples at a reduced rate. The demultiplexers' operations are protected by a real number convolutional code that produces comparable parity values at an even slower rate in a parallel similar subsystem. Parity values computed directly from the output channel's data provide detection when compared in a totally self-checking checker. The prototype baseband channel separation filters are viewed as finite impulse response (FIR) types, however, it is also shown how infinite impulse response (IIR) types can be employed. It is further demonstrated that the real convolutional code's parity filter can be modified, while still preserving its error-detecting capabilities, to simplify the parity generation processes. In any case, the parallel parity subsystem can

replace any failed subunit in the main demultiplexer. This parity scheme is also effective in protecting against failures in an A/D converter system employing separate converters in a rotating fashion. A similar protection method can be applied also in protecting any discrete Fourier transform realization, detecting failures at the data level in any fast algorithm.

The individual channel demodulators and FEC decoders perform nonlinear operations and several data channels may share the same processing resources. The demodulator, while based on a matched filter principle, contains timing and phase tracking feedback loops that make it very difficult to apply fault-tolerant design techniques. However, the redundancy inherent in the binary convolutional code used to combat transmission noise on individual data channels offers features for protecting FEC decoder realizations. The demodulation operations are protected by the fault tolerance capabilities designed into the demultiplexer and FEC decoder which surround the demodulator. Protection methods for Viterbi type FEC decoders rely upon certain invariant internal and external characteristics of the Viterbi algorithm. The decoder's output data stream is re-encoded and the successor states' metrics recomputed externally. The relative size of this metric and its comparisons with the successor's value as furnished by the decoder indicate whether the decoder has failed or the channel noise has exceed the designed performance level of the code. The effects of an increase in transmission noise will appear in many channels simultaneously, while any failure in a decoder will affect only those channels supported by the failing resource. With a small amount of additional storage, protection levels are enhanced by checking necessary conditions on the decoder's choice for successor path. The FEC decoder can also be protected internally by generating real number parity values related to survivor values in the state space of the decoder. This internal protection scheme focuses on new high-speed, block-processing parallel Viterbi decoder realizations.

Final Report
June 1990 – September 1994
NASA Grant NAG-3-1166

Fault Tolerance in Space-Based Digital Signal Processing and Switching Systems

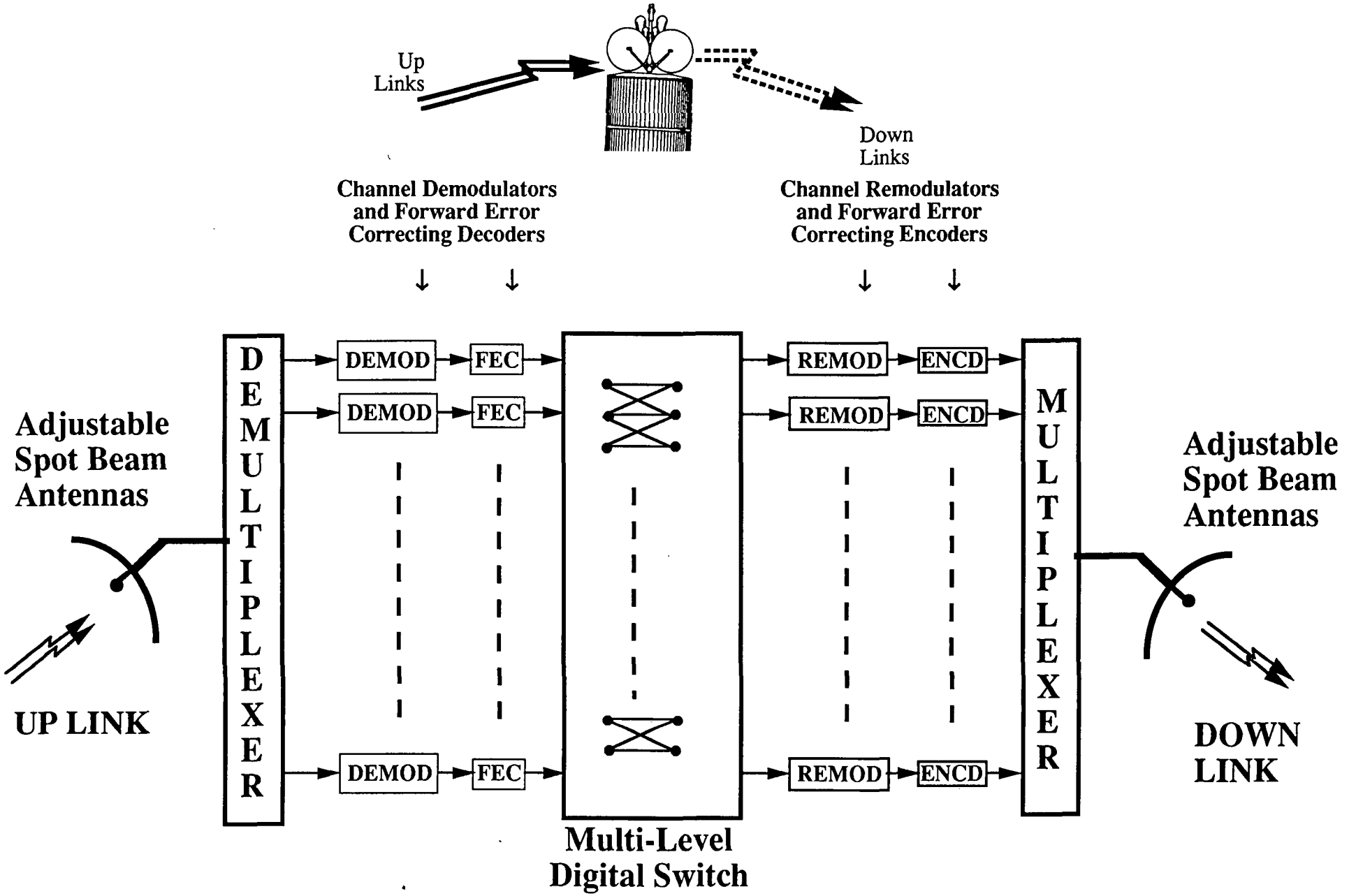
PROTECTING UP-LINK PROCESSING RESOURCES, DEMULTIPLEXER, DEMODULATOR, AND DECODER

Robert Redinbo
Department of Electrical Engineering and Computer Science
University of California, Davis

I. INTRODUCTION

Communication satellites designed to serve numerous small users, generally termed very small aperture terminals (VSAT's), will require extensive sophisticated processing in the space-borne segment. The importance of spot beam switching antenna technology is one driving factor for VSAT's. Small, low-power terminals can receive adequate signal strength from orbiting satellites by having the space transmissions concentrated by antenna radiating patterns of spot beams that dwell on relatively small areas. However, this approach requires complete demodulation and switching of all users at the satellite, not common practice today. Furthermore, demodulation and decoding followed by remodulation and recoding after switching on the satellite separates the communications path into two independent links, basically doubling the overall performance gains possible through modulation and coding.

The application of fault tolerance design principles to these communication satellites is studied using realistic system configurations as outlined in an internal NASA planning document [27] and described in a report on a proof-of-concept implementation by industry [28]. Figure 1 shows the basic subassemblies associated with a future switching communications satellite. As will be discussed later, the demultiplexer shown as a single large block is implemented by a multirate, polyphase filter bank that efficiently extracts



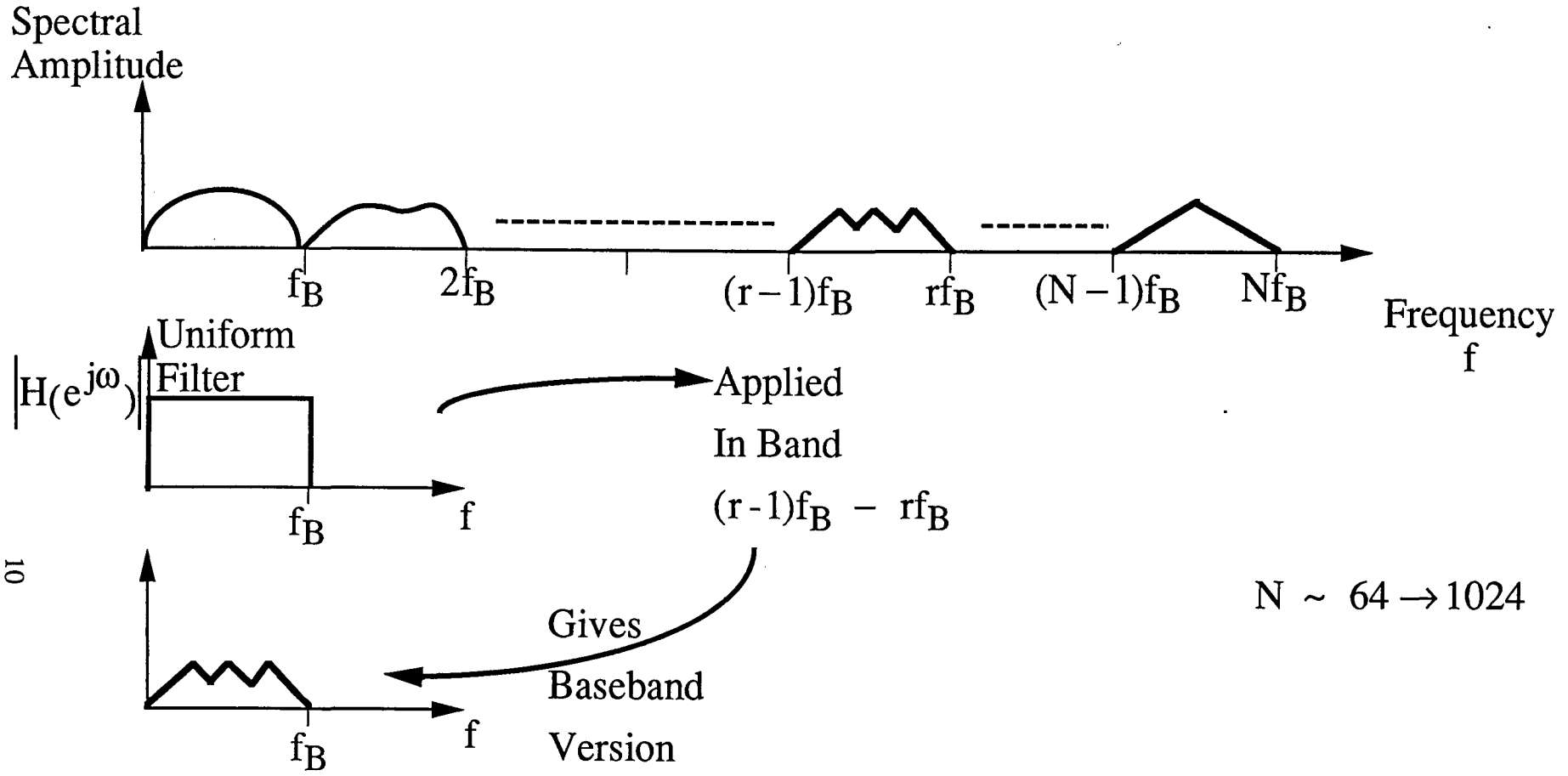
Digital Processing And Switching Subsystems in Communications Satellite
Figure 1

individual users' channels. These channels employ frequency division multiplexing (FDM), thus avoiding the tight timing synchronization requirements of the alternative method, time division multiplexing (TDM). In FDM, each user occupies a preassigned frequency band as depicted in Figure 2, which also outlines how a uniform lowpass filter can be translated to extract individual channels.

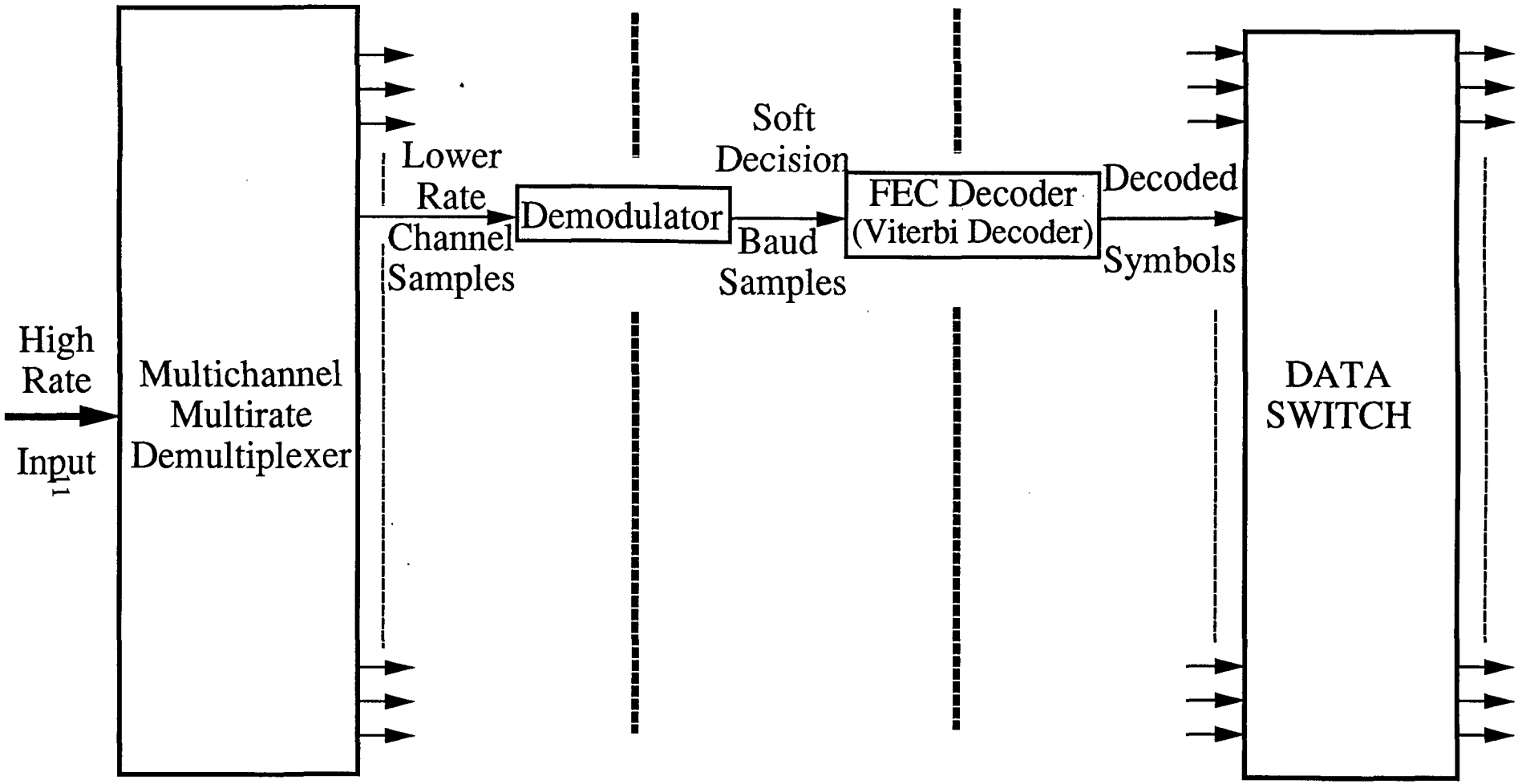
The demodulation and decoding of the forward error-correcting code (FEC) for each channel are shown in Figure 1 as separate subassemblies although their implementations undoubtedly will share hardware resources among several channels. However, there is no current system level subassembly that can perform either of these channel operations in a combined way as with demultiplexing and its efficient filter bank.

The individual channels are switched according to their respective destinations on the down link. This link usually employs TDM because the spot beam antennas and their dwell positions are naturally related to time segments. The necessary timing synchronization requirements are easily met since all users can observe the down link data bursts on each dwell. This report concentrates on fault tolerance in the first three parts of the satellite up-link resources, demultiplexer, demodulator, and FEC decoder because any fault appearing in these parts corrupt the data irreparably. Fault tolerance techniques for the latter subassemblies will be examined in future work. However, the fault-tolerant design techniques explored here have applications in these parts as well. It seems reasonable to study fault tolerance issues considering the satellite system in natural data flow order.

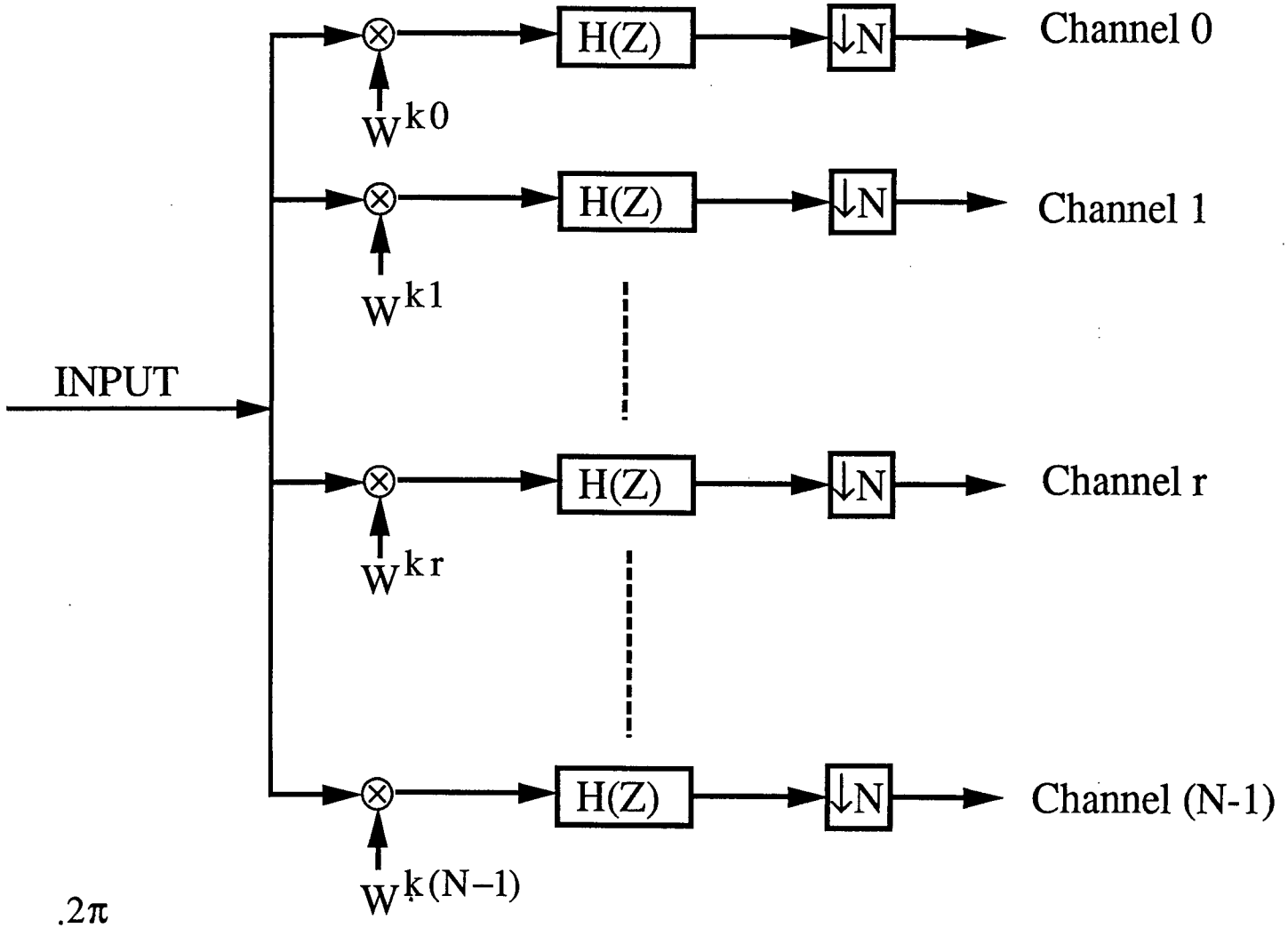
The three up-link resources being examined in this report are repeated in Figure 3. Subsequent figures will outline the function and subassemblies within the three subblocks representing demultiplexer, demodulator and decoder. Theoretically, an FDM demultiplexer separates channels by frequency shifting across the frequency band a prototype uniform baseband filter represented by transfer function H(Z) in Figure 4. Since individual output channels have a much narrower bandwidth due to the constraints of this filter, a much slower sampling rate at the output still adequately represents the data. This



Demultiplexing Frequency Division Multiplexed (FDM) Signals
Figure 2



Up Link Processing Subsystems
Figure 3



$$W = e^{j\frac{2\pi}{N}}$$

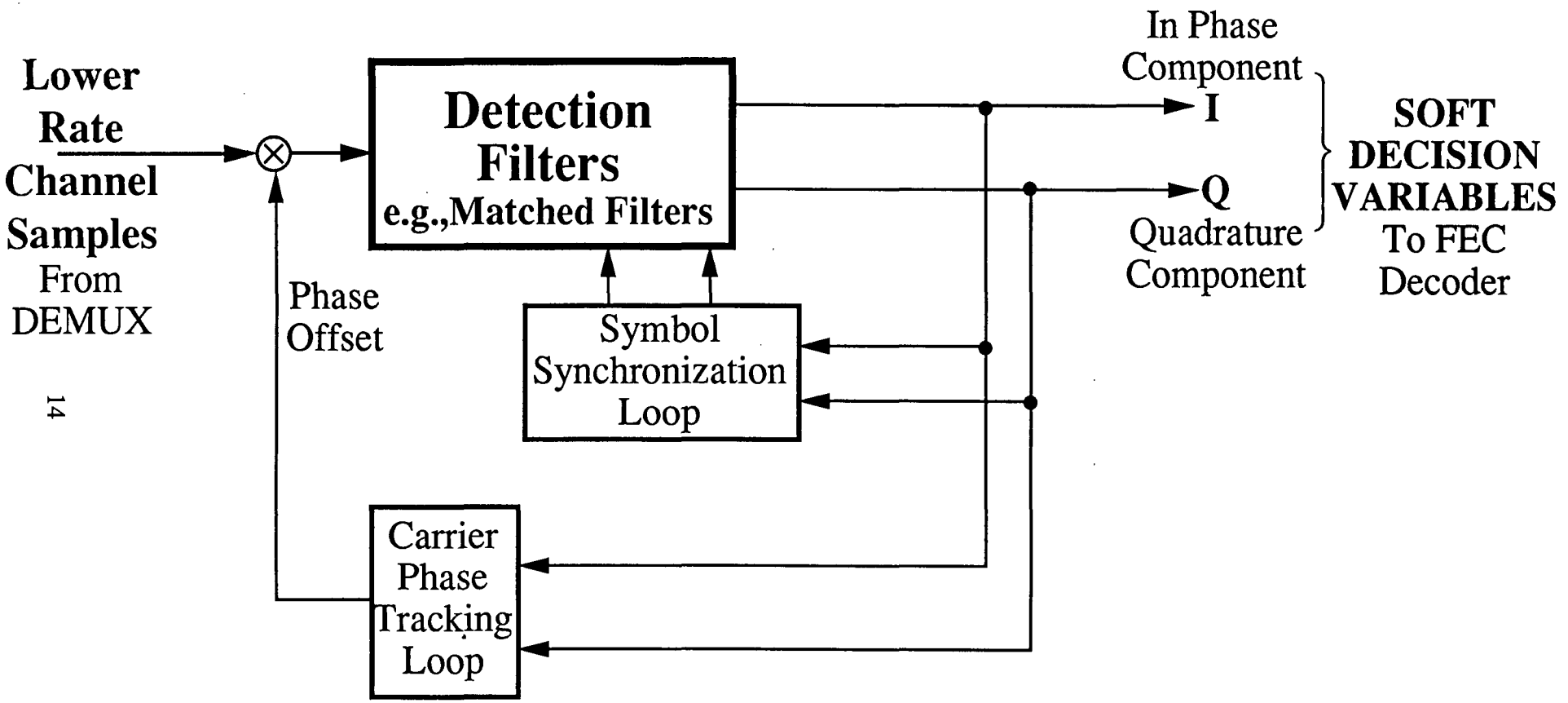
Demultiplexer Theoretically Shifts Uniform Filter
Producing Lower Rate Outputs
Figure 4

allows an efficient multirate filter bank to separate channels, a topic to be discussed more fully in a later section.

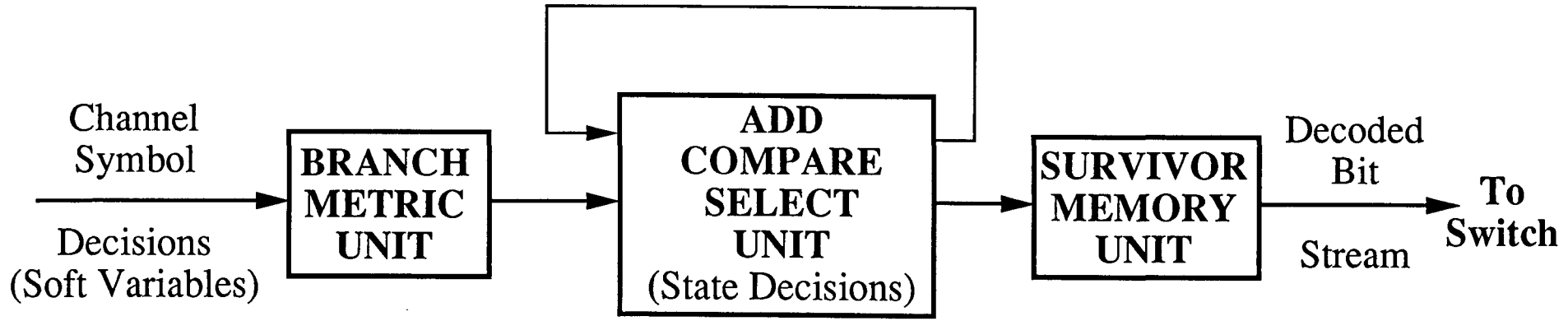
The demodulator translates several channel samples into confidence levels representing baud symbol decisions. Detection theory shows how this is performed by a coherent matched filter [29-31]; however, timing and phase tracking loops are needed to establish proper symbol epochs as well as the proper reference phase for the symbol's baud [28]. This configuration is outlined in Figure 5. The feedback loops greatly complicate the application of any standard fault tolerance design techniques to this subassembly.

Convolutional codes are generally used to combat channel errors, and the usual decoder involves a Viterbi algorithm. The implementation of the FEC decoder contains three parts, as shown in Figure 6 [30,31]. The soft decision variables which convey the demodulator's confidence level in a baud symbol decision are translated into branch metrics, the fundamental updating information needed in the Viterbi algorithm. Maximum path values at internal states are selected in the add-compare-select unit and the optimum sequence estimate is constructed by the survivor memory unit. The decoder then passes the possibly corrected data bits to the switch for routing, which may be based upon information in a header that also passed through the decoder. A major section later in this report presents novel fault tolerance features for protecting this type of decoder.

Fault tolerance is basically a redundancy management problem: in what form to obtain the redundancy and where to put it in the system? The first role for this redundancy is to detect when faulty behavior is occurring, for without adequate failure detection corrupted data can be passed through the satellite. However, a second role for this redundancy is equally important for space-borne systems. After failures have been detected, spare resources must be introduced through system reconfiguration to maintain data flow through the satellite. One challenge of fault tolerance for space-based systems involves transferring redundancy included for fault detection into spare resources that can be employed in reconfiguring the system after failures. Of course, once the redundancy is



Demodulator Subsystem
Figure 5

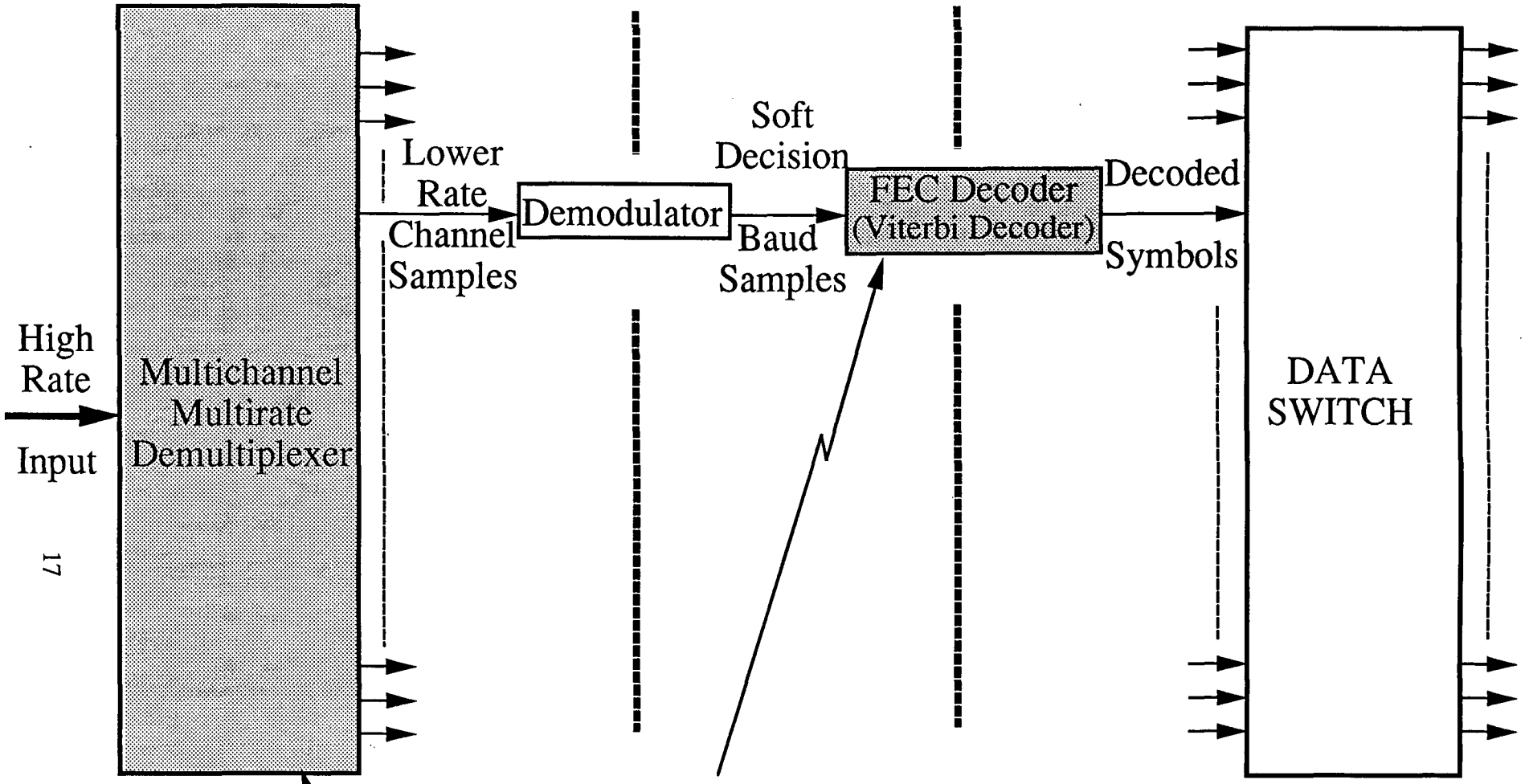


General Form, Viterbi Type FEC Decoder
Figure 6

transformed into data processing resources, the detection capabilities will be reduced. This philosophy for dual use of fault tolerance support resources is critical because of the hardware limitations in orbit. The overhead for fault tolerance must serve dual roles. Many of the methods described in this paper are in accord with this philosophy.

Convolution codes protecting the data streams introduce redundancy throughout the front-end processing operations, particularly when these data are being processed in direct coded form. On the other hand, the demodulator with its two nonlinear tracking loops is very difficult to protect, even with a combination of conventional fault tolerance techniques. One way to protect the demodulator is to move detection requirements further downstream, as outlined in Figure 7. If the demultiplexer and FEC decoders are individually fault-tolerant and if errors appear in the decoded data bits, while there are no indications of failures in the other units, either the channel noise has exceeded the convolutional code's designed performance, or there has been a failure within the demodulator. But communications theory demonstrates that any errors due to channel noise appear simultaneously in all data channels using the same transmission medium, and therefore, adjacent channels will also sense an increase in errors, a situation that is easily determined. Thus, one cause can be distinguished from the other. This type of fault tolerance can be termed the "sandwich" method. The difficult subassembly to protect is placed between two fault-tolerant subsystems, where the coded data pass through all three. When no faulty unit is indicated but the decoded data indicate errors, the intermediate unit becomes suspect. The source causing the decoded errors is easily attributed to common channel noise. If not, a faulty unit is indicated. This reduces the protection of the up-link resources to the major challenges of guaranteeing fault tolerance in the demultiplexer and the FEC decoders.

This report contains two main sections, one describing fault tolerance in the demultiplexer and the other focusing on protection in implementations of the Viterbi decoding algorithm. There are several subsections under each section addressing specific



Protecting Demodulator Using Fault Tolerance in
Preceding and Succeeding Subassemblies
Figure 7

aspects of the respective topics. Two appendices provide extra details about very specialized subproblems.

II. PROTECTING EFFICIENT DEMULTIPLEXER BANKS

1. Motivation

Data channels in communication systems are easily combined according to frequency division multiplexing (FDM). This method is particularly useful because frequency selectivity is all that is required to extract individual channels from the overall signal constellation. Many satellite communication systems employ this method of multiplexing since there is no requirement for common timing synchronization between data channels. This approach is even more appealing from a hardware implementation viewpoint because very efficient demultiplexer realizations, called polyphase multirate filter banks, are available [1-3]. They take advantage of the narrow band nature of the individual demultiplexed channels, permitting them to be sampled at a relatively lower rate as compared to the rate required for the wide band constellation.

The basic demultiplexing philosophy envisions a narrow band filter extracting each channel from the multiplexed signals. Figure 2, shown earlier, visualizes N multiplexed channels, each with relative bandwidth f_B, combined into an FDM signal. It also shows the basic demultiplexing philosophy where an idealized narrow band filter with Z transform transfer function H(Z), is shifted in frequency to separate a band of frequencies corresponding to a channel, in the case shown in the figure from (r-1) f_B to rf_B. This multiplexing format utilizes single sideband forms of each user (analytic signal representation), and therefore, the complete constellation may be reconstructed using a sampling rate of Nf_B [Section 9.2, 1; Chapter 5, 46]. A theoretical view of the demultiplexer appearing previously in Figure 4 shows how the uniform baseband filter is effectively shifted to each respective band by the scaling phasors. The symbol $\downarrow N$ indicates that the output channel only produces samples at a rate $1/N^{th}$ of the input sampling

rate [4]. Generally, the uniform filter represented by H(Z) in the figure has a finite impulse response (FIR) configuration with the attendant advantage of linear phase [5].

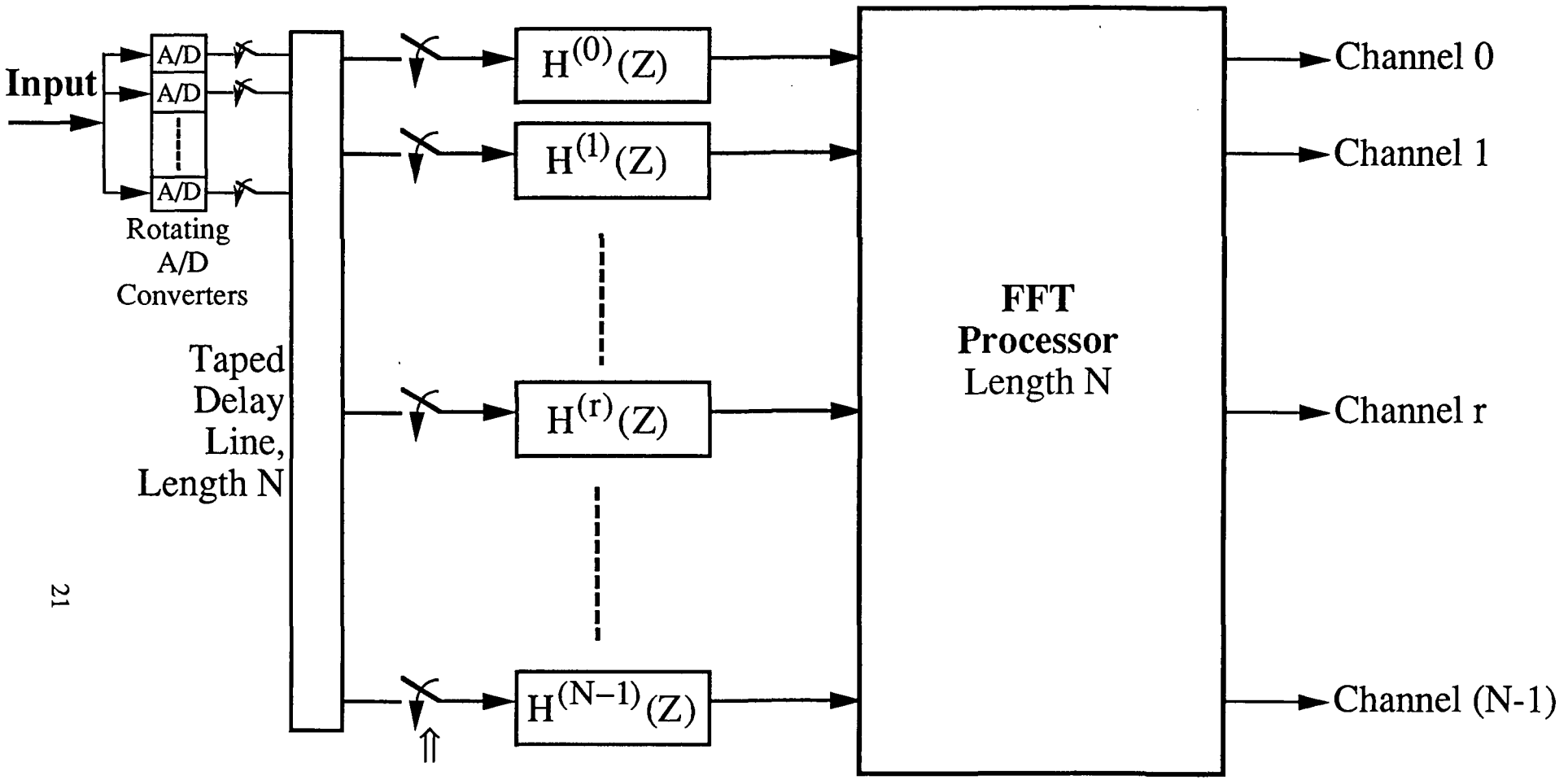
It is well known [1, 4], and will be reviewed later in a subsection of this report, that the uniform filter banks can be realized by defining certain segments of the baseband filter's transfer function and then using the outputs from these shorter filters as simultaneous inputs to a discrete Fourier transform operation. This approach to demultiplexing is outlined in Figure 8, where a fast Fourier transform (FFT) algorithm realizes the discrete Fourier transform. The relationship between the new shorter segmented filters $H^{(r)}(Z)$, r = 0, 1, ..., N-1, and the original baseband filter H(Z), will be summarized later. A rotating A/D subassembly is shown at the input in Figure 8 which translates the analog input into digital samples by using several A/D converters in a round-robin fashion. A method for protecting this very important subassembly will be included later. For the purposes of the intervening development, digital values will be assumed available. This form of polyphase multirate filtering is called critically sampled [1] because the downsampling rate and the number of channels are equal. The most important feature of Figure 8 is its slower sampling rate applied BEFORE the filtering and FFT operations, permitting the digital hardware implementing these functions to operate at a data rate $1/N^{th}$ that of the input data sampling rate. Nevertheless, the input is still sampled at a suitably high rate commensurate with its wider bandwidth. The high rate data input samples are temporarily stored, but no processing is performed at this incoming rate until the data are routed to the N individual segmented filters. The efficiency achieved by this form of demultiplexer is a consequence of the shared processing in the discrete Fourier transform operations.

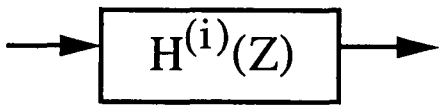
There are situations where this efficient form of demultiplexer must be highly reliable. Yet, the very efficient sharing of processing resources makes this form extremely sensitive to even simple failures which can easily contaminate many data channels simultaneously. There are several aspects of applying fault tolerance to a demultiplexer system as described above. The first important consideration is the detection of failures,

whether they are permanent or temporary and transient. Once inaccurate performance is detected, the failed subunit must be identified and located. Finally, if the failures persist, the system must be reconfigured so that adequate performance is still achieved. This section concentrates on the first aspect, fault detection. For, without an indication of improper operation, the other aspects of fault tolerance cannot be invoked.

The detection of failures in digital systems can be applied at various levels of the implementation from the gate level up through whole digital subsystems [6-8]. Furthermore, there are numerous techniques available at each level. However, in the case of signal processing where different kinds of application specific integrated circuits (ASIC) are interconnected to affect the overall processing operation, it is difficult to incorporate modifications at the digital design level to support fault tolerance. There is an emerging alternate method of fault tolerance, termed by some Algorithm-Based Fault Tolerance (ABFT), that views the algorithmic operations and the data sample flow as the important items to protect regardless of the underlying hardware realization. The first use of this technique was in protecting matrix operations [9], and there have been many other applications investigated [10-17]. Most research has been directed to protecting linear algorithms.

The fundamental approach in ABFT employs real number error-detecting codes to define parity values associated with a group of data samples. This basic philosophy is outlined in Figure 9. These codes can be either block or convolutional codes [18-20]. In either case, the original processing algorithm is combined with the parity generation process, generally leading to a composite, efficient, simplified parity generation algorithm that produces independent parity values which are associated with the output data. Then comparable parity values are computed directly from the original processing algorithm's output data. The respective parity values, one from each set but computed in different ways, should be identical, except possibly for some small round-off error differences since they are evaluated in two dissimilar ways. Errors are detected when the respective parity



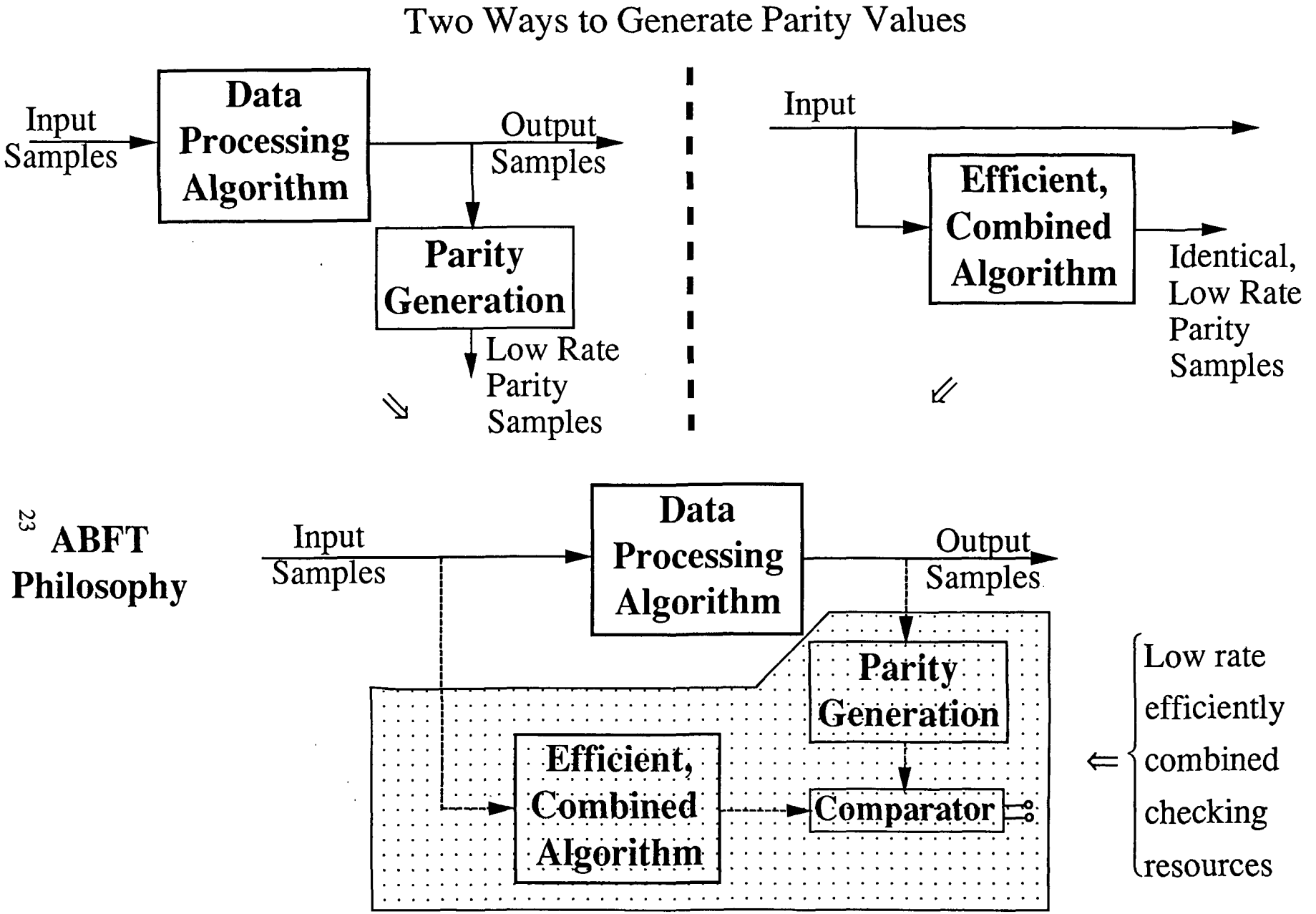


Segment of
Prototype
Filter

Polyphase Multirate Filter Bank
Figure 8

values differ significantly. This type of fault tolerance will be applied to protecting the demultiplexer. The underlying philosophy holds that the processed data samples are the critical items whose integrity is to be guaranteed. Any failures that cause incorrect data are to be first detected with appropriate diagnosis and reconfiguration actions to follow.

The complete details of real convolutional codes will be given later, but for the moment, the form of ABFT to be employed in protecting the demultiplexer will be explained assuming that the real number parity values are generated by a FIR filter, represented by transfer function Q(Z). Furthermore, only every K^{th} output of filter Q(Z) represents a parity value associated with the real convolutional code. This implies that the output of the parity filter Q(Z) is passed through a downsampler at rate K, where K typically is in the range from 5-10. The fundamental error-detecting approach for one channel in the demultiplexer is shown symbolically in Figure 10, where this downsampling operation is denoted as $\downarrow K$. This theoretical view produces the output for channel r by passing a frequency shifted data stream through the uniform baseband filter H(Z). The output is processed by the parity generation filter Q(Z) with only every K^{th} sample preserved. It will be shown later that H(Z) cascaded with Q(Z) and followed by a downsampler represents the simplified parallel forward parity generation algorithm. These two related parity streams are compared and any significant difference between respective values indicate errors, up to the error detecting potential of the code employed.



Principle of Algorithm Based Fault Tolerance
Figure 9

2. *Review of Efficient Implementations of Filter Banks*

An analysis bank of filters will be examined where each of the L transfer functions $H_0(Z), H_1(Z), ..., H_{L-1}(Z)$ bandlimit their respective signal outputs so that each may be sampled at a rate 1/N of the input rate. This general setting is depicted in Figure 11a. The L transfer functions will be assumed FIR types, for the purposes of the exposition. (Similar results are possible for infinite impulse response (IIR) filler forms and these results are developed in an appendix just for completeness.) Each of the L filter paths can be analyzed separately and the generic situation is isolated in Figure 11b, for further development. The Z transform quantities shown in these figures employ the two-side Z transform. Infinite limits in the summations are included in its definition below, even though only a finite number of nonzero terms appear for the FIR filter case

$$\{h_p(n)\} \leftrightarrow H_p(Z) = \sum_{n=-\infty}^{+\infty} h_p(n) Z^{-n} \tag{1}$$

The sequence $\{h_p(n)\}$ is a shifted version of the prototype filter's impulse response:

$$\{h_p(n)\} \leftrightarrow \left\{ h(n) e^{j2p\frac{pf_B}{Nf_B}n} \right\}$$

The sampling period is $T = Nf_B$, as noted earlier when discussing the single sideband nature of the individual data channels.

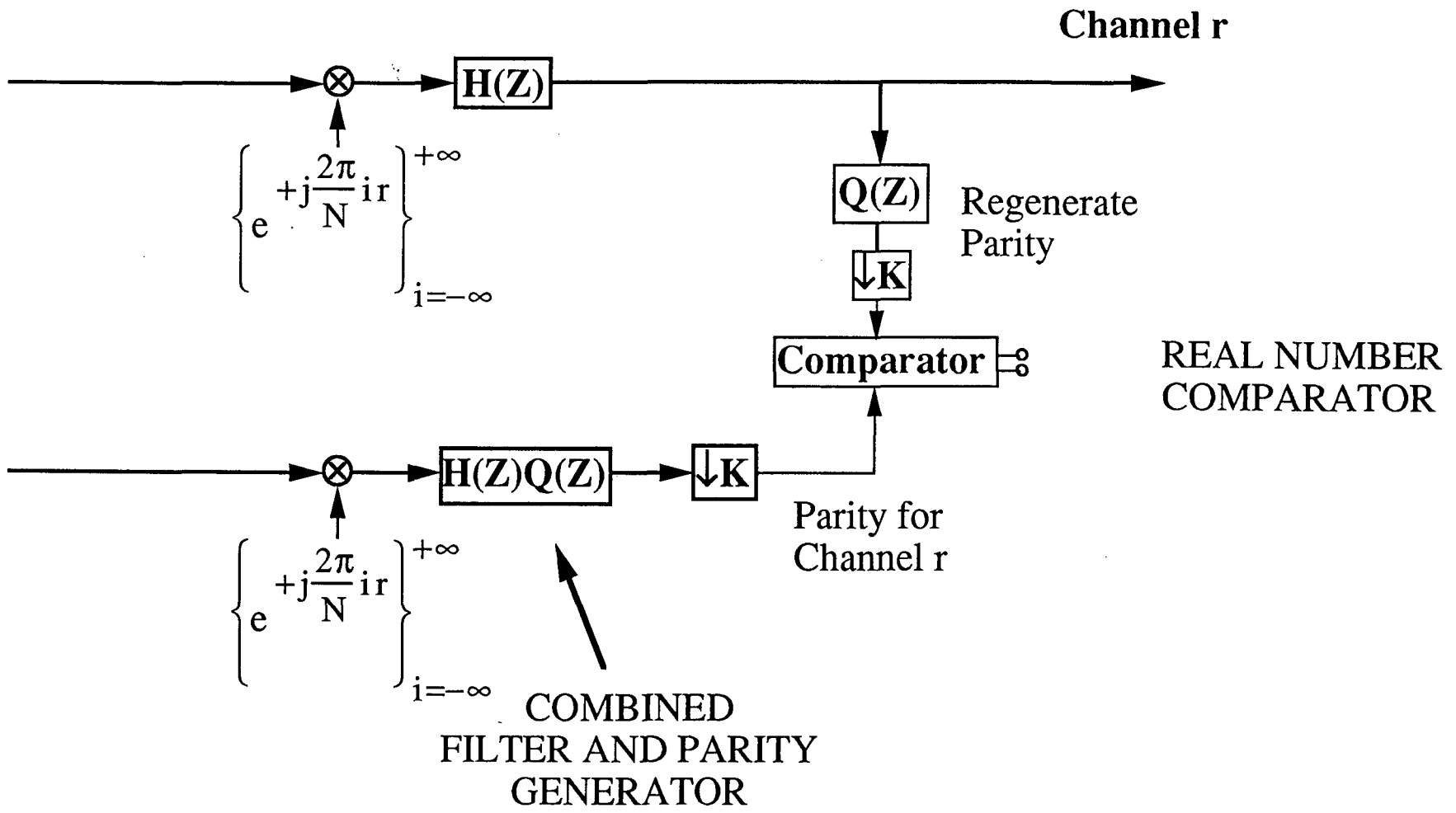
The output sequence from the filter, denoted by the sequence $\{\xi_p(r)\}$ in Figure 11b, may be written in terms of the input samples and the impulse response of the filler

$$\xi_p(m) = \sum_{s=-\infty}^{+\infty} h_p(m-s)\, x(s) = \sum_{s=-\infty}^{+\infty} h_p(s)\, x(m-s) \tag{2}$$

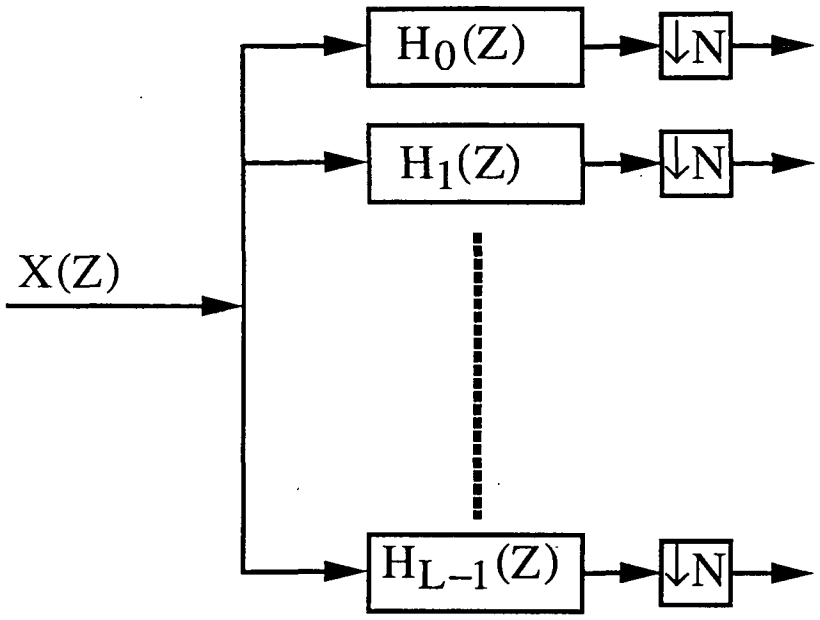
The down sampler $\downarrow N$ basically keeps every N^{th} sample of $\{\xi_p(m)\}$ and its output $y_p(r)$ may be written as:

$$y_p(r) = \xi_p(rN) = \sum_{s=-\infty}^{+\infty} h_p(rN-s)\, x(s) = \sum_{s=-\infty}^{+\infty} h_p(s)\, x(rN-s) \tag{3}$$

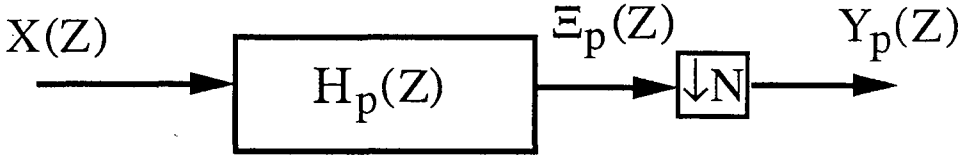
The summation index s may be decomposed using the Euclidean algorithm and the single sum replaced by a double summation with one over only N values.



Protecting Demultiplexer Channel
Figure 10



Bank of L Filters Followed by Downsamplers
Figure 11a



$$\Xi_p(Z) \leftrightarrow \{\xi_p(r)\} \qquad Y_p(Z) \leftrightarrow \{y_p(r)\}$$

Generic Filter Path
Figure 11b

General Analysis Bank
Figure 11

$$s = uN + v \quad : \quad v = 0, 1, ..., N\text{-}1; \quad u = 0, \pm 1, \pm 2, ... \tag{4a}$$

$$y_p(r) = \sum_{v=0}^{N-1} \sum_{u=-\infty}^{+\infty} h_p(rN - uN - v)\, x(uN + v) \tag{4b}$$

$$= \sum_{v=0}^{N-1} \sum_{u=-\infty}^{+\infty} h_p(uN + v)\; x(rN - uN - v) \tag{4c}$$

The impulse response $\{h_p(m)\}$ is segmented into N subsequences and the weighting operation separated into N parallel convolutions. The N parallel convolutions employ segmented impulse responses related to the original p^{th} channel impulse response in the following way.

$$h_p^{(v)}(a) = h_p(aN + v) \; ; \; \begin{matrix} v = 0, 1, ..., N - 1 \\ a = 0, \pm 1, \pm 2, ... \end{matrix} \tag{5a}$$

$$y_p(r) = \sum_{v=0}^{N-1} \left[\sum_{u=-\infty}^{+\infty} h_p^{(-v)}(r - u)\; x(uN + v) \right]$$

$$= \sum_{v=0}^{N-1} \left[\sum_{u=-\infty}^{+\infty} h_p^{(v)}(u)\; x((r - u)N - v) \right] \; ; r = 0, \pm 1, \pm 2, ... \tag{5b}$$

An important interpretation of the decomposition is shown in Figure 12, describing the separation of the filtering action of the original path represented by $H_p(Z)$. The Z transform of the N segmented impulse responses, $\{h_p^{(v)}(a)\}$, equation (5a), is defined in an obvious way incorporating the lower sampling rate caused by the downsampling

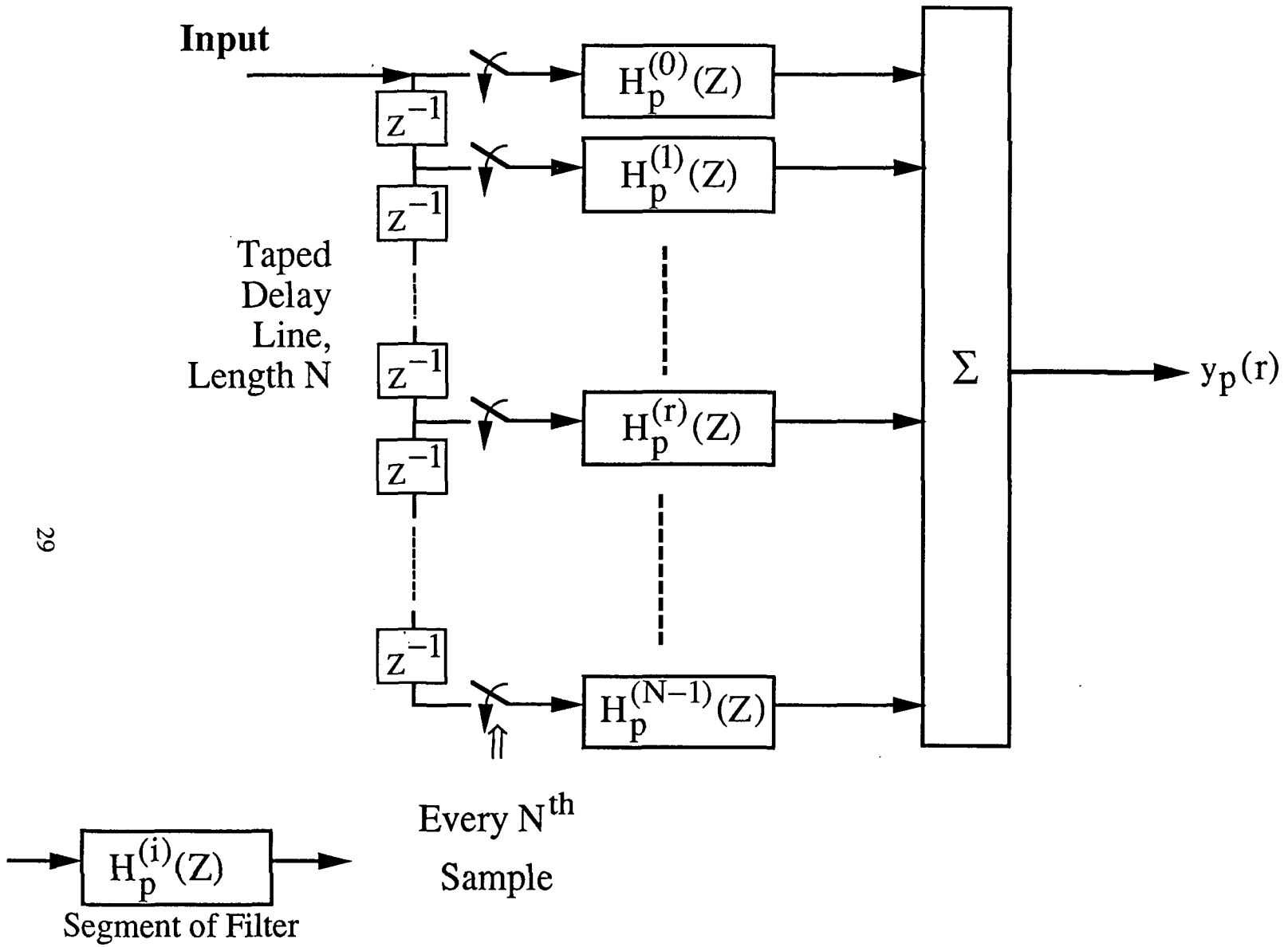
$$H_p^{(v)}(Z) = \sum_{a=-\infty}^{+\infty} h_p(aN + v)\; Z^{-aN} \quad ; v = 0, \; 1, ..., \; N - 1. \tag{6}$$

The sampling reduction by factor N permits the input in equation (5b) to be delayed according to variable v, and each subsequence so formed weighted by a segmented impulse response $\{h^{(v)}(a)\}$. The delay line depicted in Figure 12 separates the input stream into

these respective subsequences. In this way processing is decomposed into N parallel paths.

The same number of operations are performed in this approach, but each parallel self-contained path can operate at a rate reduced by factor N. The ability to use a slower processing rate in each disjoint parallel path provides a serial-to-parallel tradeoff. Digital signal processing is now possible, whereas in the original configuration, the speed requirements because of the input's high bandwidth would have been prohibitive. The high speed operations are now confined to the analog-to-digital conversion unit, which must be performed under all circumstances, and the length N shift register type memory storage. There are dramatic efficiencies possible when L = N and the filters $H_i(Z)$, i = 0, 1, ..., N - 1, are frequency shifted versions of a single uniform baseband transfer function.

Demultiplexers can be viewed as a special form of Figure 11a, wherein L = N and the N transfer functions are constrained to be related to one common baseband transfer function H(Z) as shown earlier in Figure 4. It is easy to demonstrate the effects of the scaling phase sequence $\{e^{j\frac{2\pi}{N}pr}\}_{r=-\infty}^{+\infty}$ is to shift the filter response;



Block Implementation of Filter Path $H_p(Z)$

Figure 12

$$\{e^{j\frac{2\pi}{N}pr}h(r)\}_{r=-\infty}^{+\infty} \leftrightarrow H\left(e^{j2\pi(f-\frac{p}{N})}\right)$$

; f is Frequency Variable.

The impulse response {h(r)} corresponds to the transfer function H(Z). Under these conditions, the output of the p^{th} filter path may be written with the aid of equation (4) as:

$$y_p(r) = \sum_{v=0}^{N-1} \sum_{u=-\infty}^{+\infty} h(uN+v)\, x((r-u)N-v)W^{-pv}$$

$$W = e^{j\frac{2\pi}{N}}$$

$$= \sum_{v=0}^{N-1} \sum_{u=-\infty}^{+\infty} h((r-u)N-v)\, x(uN+v)W^{pv} \qquad (7)$$

One important feature stands out in equation (7). The scaling phasor is not a function of the inner summation on variable u and, therefore, may be moved to the output of each respective filter path. In this regard, the segment impulse response and its corresponding Z transform may be identified.

$$h^{(v)}(a) = h(aN + v)$$

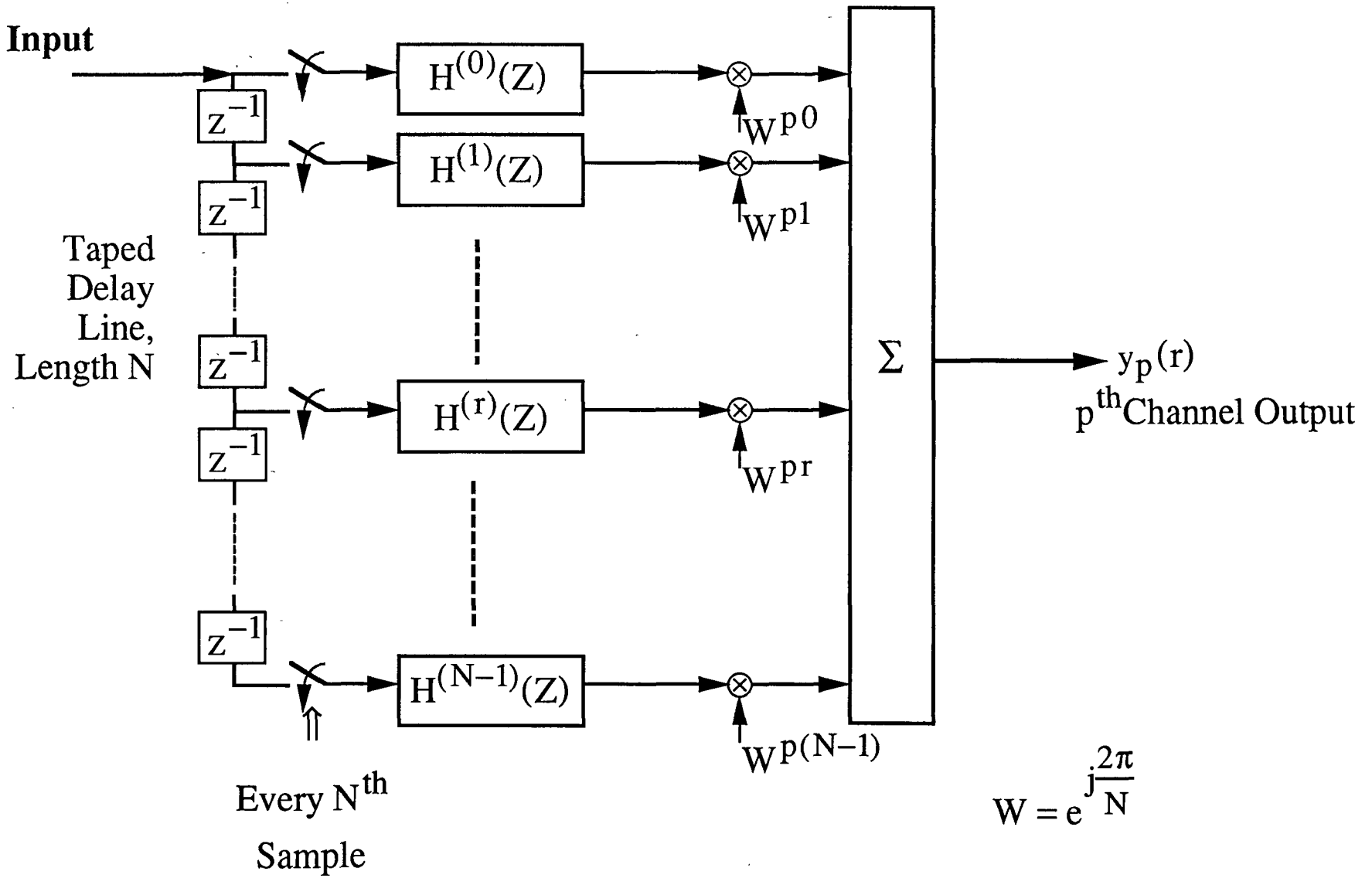
$$; \quad v = 0, 1, \ldots, N-1 \qquad (8a)$$
$$a = 0, \pm 1, \pm 2, \ldots.$$

$$H^{(v)}(Z) = \sum_{a=-\infty}^{+\infty} h^{(v)}(a)\, Z^{-aN} \qquad (8b)$$

The p^{th} channel output is the sum of N scaled versions of filter paths with transfer functions $H^{(0)}(Z), H^{(1)}(Z), \ldots, H^{(N-1)}(Z)$

$$y_p(r) = \sum_{v=0}^{N-1} w^{pv} \left\{ \sum_{u=-\infty}^{+\infty} h^{(-v)}(r-u)\, x(uN+v) \right\} \qquad (9)$$

This is shown in Figure 13.



Output of p^{th} Channel for Uniform Filter

Figure 13

The final summation over index variable v in equation (9) is equivalent to forming the p^{th} discrete Fourier transform coefficient for the N outputs of the segmented impulse response filters $\left\{h^{(-v)}(r)\right\}_{v=0}^{N-1}$. Furthermore, the only change needed to get an output for a different output, say $\{y_m(r)\}$, is to modify the scaling coefficients, $\{e^{j\frac{2\pi}{N}mv}\}_{v=0}^{N-1}$, affecting the outer sum. The same segmented impulse response filters are employed, but the scaling values change. Thus Figure 8 represents the general case where an FFT form of a discrete Fourier transform is applied to the respective outputs of the segmented filter functions. All N channel outputs of the demultiplexer are obtained simultaneously. This is the basis for the great efficiency of polyphase multirate demultiplexer filter banks.

There are new general lattice decompositions of the indices that lead to even more efficient realizations particularly when timing and phase tracking compensations are to be integrated in the demultiplexer [47, 48]. The fault detection schemes developed in succeeding sections also apply to these more general formulations.

3. Real Convolutional Codes and DSP Operations

Convolutional codes have been defined traditionally over finite field alphabets [21, 22], but recent research results show how they may be extended to systems using either integer or real arithmetic [18, 20, 14]. Nevertheless, the basic approach to convolutional codes remains the same, particularly with regard to a matrix description of the encoding and parity checking functions. Only systematic forms of convolutional codes will be considered primarily because the normal filtering operations are not altered and such forms are automatically noncatastrophic [22]. Only the detecting capabilities of such codes are used; any correcting operations could easily exceed the original processing requirements.

The encoding matrix for a systematic convolutional code, G, has a block-type format involving *m* fundamental finite sized matrices whose dimensions are related to the

rate and number of parity check positions in the code. The parameter m determines the constraint length of the code.

$$G = \begin{pmatrix} G_0 & G_1 & - & - & G_m & 0 & - \\ 0 & G_0 & - & - & G_{m-1} & G_m & - \\ 0 & 0 & - & - & - & G_{m-1} & - \\ 0 & 0 & | & & | & - & - \\ 0 & 0 & - & - & - & - & - \\ | & | & | & G_0 & - & - & | \\ - & - & - & | & G_o & G_1 & - \\ | & | & | & - & 0 & G_0 & - \\ - & - & - & - & 0 & 0 & - \\ | & | & | & | & | & | & | \end{pmatrix} \qquad (10)$$

The $k \times m$ submatrices G_j, $j = 0, 1, ..., m$, have distinctive forms and divide into two types.

$$G_0 = (I \mid P_0) \quad ; I,\ k \times k \qquad \text{Identity Matrix}$$

$$P_0,\ k \times (n - k) \qquad \text{Parity-Check Matrix} \qquad (11a)$$

$$G_j = (0 \mid P_j) \quad ; 0, k \times k \qquad \text{Zero Matrix}$$

$$; P_j, k \times (n - k) \qquad \text{Parity-Check Matrix}$$

$$j = 1, 2, \ldots, m. \qquad (11b)$$

The entries in the parity check submatrices P_i may be either 0 or 1 even for the real Marshal code case [18, 20], or in the more general case, real numbers [14, 23].

The parity positions are a function of possibly $M = (m + 1)k$ input samples through the action of the P_j parts of each G_j. The stack of these parity weighting values will be denoted by an $\{M \times (n - k)\}$ matrix Q with respective columns $\{\underline{q}_r\}$

$$Q = \begin{pmatrix} P_m \\ P_{m-1} \\ | \\ | \\ | \\ | \\ P_2 \\ P_1 \\ P_0 \end{pmatrix} = (\underline{q_0}, \underline{q_1}, \underline{q_2}, \ldots, \underline{q_{n-k-1}}) \quad ; \quad M = (m+1)k \tag{12a}$$

$$\underline{q_c} = ((q_c^{(j)})) \quad ; \quad j = 0, 1, 2, \ldots, M - 1$$

$$\underline{q_c}\ M \times 1 \text{ Column Vector} \tag{12b}$$

$$c = 0, 1, 2, \ldots, (n - k - 1).$$

The (n - k) parity position associated with the input values are obtained by the weighing action of columns $\underline{q_r}$. Each parity value may be viewed as the output of an FIR filter, described notationally using the Z transform of column $\underline{q_c}$.

FIR Filter Effect, Column c.

$$Q_c(Z) = \sum_{j=0}^{M-1} q_c^{(M-1-j)}\, Z^{-j} \quad ; \; c = 0, 1, 2, \ldots, (n-k-1) \tag{13}$$

Real convolutional codes can also be imbued with a distance structure similar to the usual one applied to finite field symbol codes. It is possible to define a metric in terms of a real Hamming weight. For illustrative purposes, consider the real code symbols to be defined with infinite precision real or complex numbers and note that these algebraic structures have a unique and easily discernible zero element. Let $\underline{X}$, $\underline{Y}$ and $\underline{Z}$ be $(1 \times L)$ vectors of numbers where L will be fixed by the context below. The real Hamming weight is the number of nonzero components in a vector, say $\underline{Z}$.

$$W(\underline{Z}) = \#\ \text{NONZERO COMPONENTS IN}\ \underline{Z} \tag{14a}$$

A valid distance function (obeying the usual mathematic requirements of nonnegativeness, symmetry and the triangle inequality, [21]) may be defined using the additive operator and the Hamming weight.

$$d(\underline{X}, \underline{Y}) = W(\underline{X} - \underline{Y}) \tag{14b}$$

In practical systems, the precision of the arithmetic implementation will dictate the occurrence of zero components. However, that does not limit the theoretical view being presented here.

A convolutional code produces ever lengthening output vectors as more input digits pass through the encoding process. In order to describe this behavior, variable length vectors will be used and so appropriate notation will be established. The $1 \times [(i+1)k]$ vector $[u]_i^{(k)}$ represents an input of $(i+1)$ subblocks each of length k.

$$[u]_i^{(k)} = \left(u_0^{(0)}, u_0^{(1)}, \ldots, u_0^{(k-1)},\ u_1^{(0)},\ u_1^{(1)},\ \ldots\ u_1^{(k-1)}, \ldots u_i^{(0)},\ u_i^{(1)}, \ldots u_i^{(k-1)}\right) \tag{15a}$$

; $u_j^{(r)}$ r^{th} Digit of Subblock j.

In a similar way $[v]_i^{(n)}$, a $1 \times [(i+1)n]$ vector, corresponds to the output of the encoder when $[u]_i^{(k)}$ is the input.

$$[v]_i^{(n)} = \left(v_0^{(0)},\ v_0^{(1)}, \ldots, v_0^{(n-1)},\ v_1^{(0)} - v_1^{(n-1)}, \ldots, v_i^{(0)}, \ldots v_i^{(n-1)}\right) \tag{15b}$$

; $v_j^{(r)}$ r^{th} Output Digit of Subblock j.

The encoding action is described through a truncated form of the encoding matrix G, equation (10). This truncated submatrix is denoted by $[G]_i$ and is extracted from G as the upper left $\{(i+1)k \times (i+1)n\}$ elements.

$$[u]_i^{(k)}\ [G]_i = [v]_i^{(n)} \qquad \text{; Encoding Action} \tag{15c}$$

The minimum column distance associated with $(i+1)$ input elements is labeled by d_i and is formally defined as:

$$d_i = \min\left\{ d\left([u']_i^{(k)}[G]_i, [u'']_i^{(k)}[G]_i\right)\right\} \tag{16}$$

all $[u']_i^{(k)}$ and $[u'']_i^{(k)}$

$[u']_0^{(k)} \neq [u'']_0^{(k)}$

The infinum is over the distance between encoded output sequence corresponding to input sequences that certainly differ at least somewhere in the first subblock of k input digits. Because the code is linear, this equivalent to an infinum over encoded sequences originating with inputs nonzero in at least the first k digits.

$$d_i = \min\left\{ w\left([u]_i^{(k)}[G]_i\right)\right\} \tag{17}$$

all $[u]_i^{(k)}$

such that $[u]_0^{(k)} \neq \underline{0}$

The column distance d_i is a nondecreasing function of index i. Hence, the minimum free distance is a natural definition as the limit is approached.

$$d_{free} = \lim_{i \to \infty} \min\left\{ w\left([u]_i^{(k)}[G]_i\right)\right\} \tag{18}$$

all $[u]_i^{(k)}$

$[u]_0^{(k)} \neq \underline{0}$

Another aspect of finite field convolutional code theory that carries over directly to real convolutional codes is the concept of a dual code (space). For example, there is a dual matrix H such that

$$G H^T = 0 \qquad ; H^T \text{ denotes Hermitian transpose} \tag{19}$$

where the matrices are infinite dimensions. However, this theory also applies to the submatrices extracted as the upper left corner such as those involved in the definition of the

column distance. (In fact, this is the motivation for calling this value the column distance as will be seen shortly.)

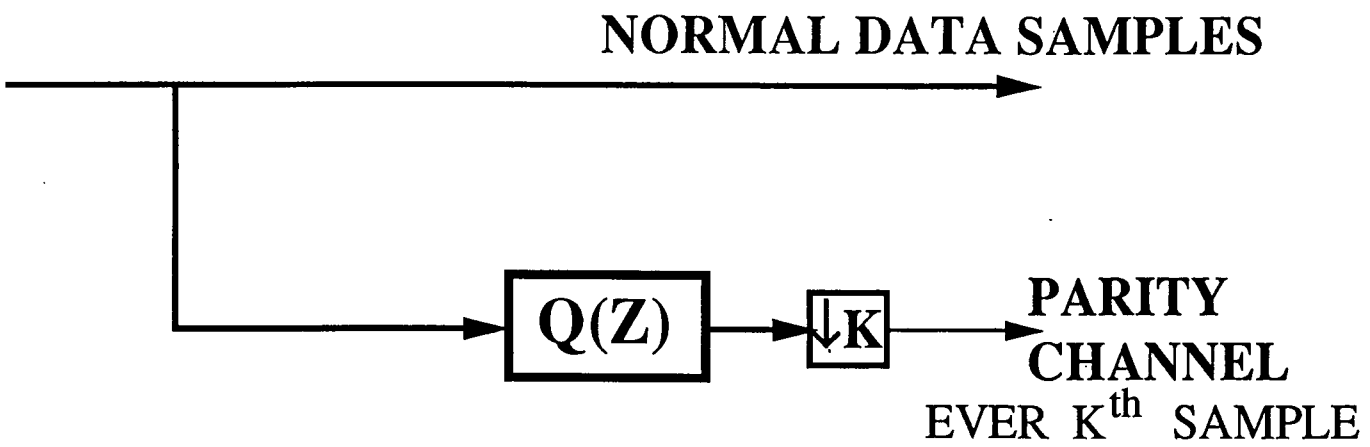
$$[G]_i \ [H]_i^T = 0 \tag{20}$$

As in the usual development for block and convolutional codes over finite fields, for every codeword piece of weight w, there exists a dependency relationship among w columns of $[H]_i$, [21]. Hence, if the minimum column distance for length (i+1) input subblocks is w, every (w-1) columns of $[H]_i$ must be linearly independent [21]. It is also easy to show that any code over finite fields can be regarded as a code over the real numbers translating the field integers into the integer subset of the reals. The distance properties of this real code will be at least as good as those for the finite field code in its algebraic structure [23].

High rate convolutional codes with only one parity channel will be used for protecting output data channels emanating from a demultiplexer. Binary-based codes, for which there exist tables of high performance codes [24], will be chosen. In particular, a rate K/(K + 1) systematic convolutional code is defined by a single parity weight filter, equations (12) and (13). A single parity value for every K input sample is produced by sampling an FIR filter with transfer function denoted by Q(Z), equation (13) without a subscript. A convenient view of the parity production process is shown in Figure 14. The data flow normally and are simultaneously tapped to this FIR parity filter, Q(Z). The downsampling symbol $\downarrow$K indicates that after every K data samples, one parity value is produced. The previously adopted notation may be used to explicitly show the components of the output codeword. The parity values are labeled $v_i^{(k)}$, i = 0, 1, ...

$$[v]_i^{(K+1)} = \left(u_0^{(0)}, \ u_0^{(1)}, \ldots, u_0^{(K-1)}, \ v_0^{(K)}, \ u_1^{(0)}, \ u_1^{(1)}, \ldots \ u_1^{(K-1)}, \ v_1^{(K)} \ldots \right.$$

$$\left. \ldots, u_i^{(0)}, \ldots u_i^{(K-1)} \ v_i^{(K)} \right)$$



Q(Z), Parity Weighting Filter

Parity Generation in a Rate (K/K+1) Systematic Convolutional Code
Figure 14

where the input data that passes directly through to the systematic codeword values are

$$[u]_i^{(K)} = \left(u_0^{(0)}, \ldots, u_0^{(K-1)},\ u_1^{(0)} \ldots,\ u_0^{(K-1)}, \ldots u_i^{(0)}, \ldots u_i^{(0)},\ v_i^{(K-1)} \right).$$

4. Composite Filtering and Parity Generation

This section develops methods for combining the parity generation operations with filter banks, such as shown in Figure 11, forming a cascaded system, depicted in Figure 15. A generic channel with signal value notation overlaid is presented in the middle of this figure. The output of the t^{th} filter, $H_t(Z)$, is denoted by $Y_t(Z) \leftrightarrow \{y_t(r)\}$. The parity output $\{p_t(a)\}$, after downsampling by factor K, may be written in terms of the t^{th} channel signal $\{y_t(r)\}$.

$$p_t(a) = \sum_{d=0}^{K-1} \sum_{c=\infty}^{+\infty} q((a-c)K-d)\, y_t\,(cK+d)$$

$$a = 0, \pm 1, \pm 2, \ldots \qquad t = 0, 1, \ldots, L-1 \qquad (21)$$

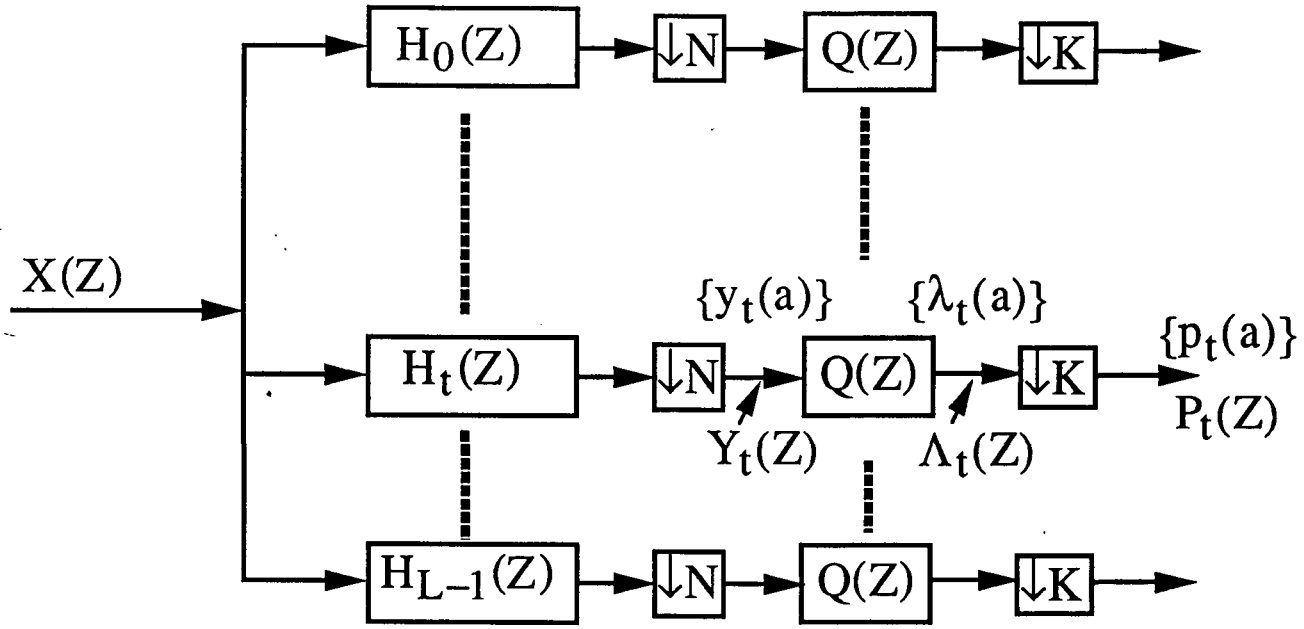
$$= \sum_{d=0}^{K-1} \sum_{c=\infty}^{+\infty} q(cK+d)\ y_t\ \left((a-c)K-d\right)$$

A substitution from a previous result, equation (4), shows how the input is effectively weighted by a composite of the filter and parity weighting functions.

$$p_t(a) = \sum_{v=0}^{N-1} \sum_{u=-\infty}^{+\infty} x(uN+v)\ g_t^{(v)}\ (aK-u) \qquad (22a)$$

The composite weighting functions $\{g_t^{(v)}(r)\}$ contain every N^{th} sample of the filter weighting, properly offset by index v.

$$g_t^{(v)}(s) = \sum_{r=-\infty}^{+\infty} q(r)\, h_t\left((s-r)N-v\right) \qquad v = 0, 1, \ldots, N\text{-}1 \qquad (22b)$$



Parity Generation For Analysis Bank Outputs
Figure 15

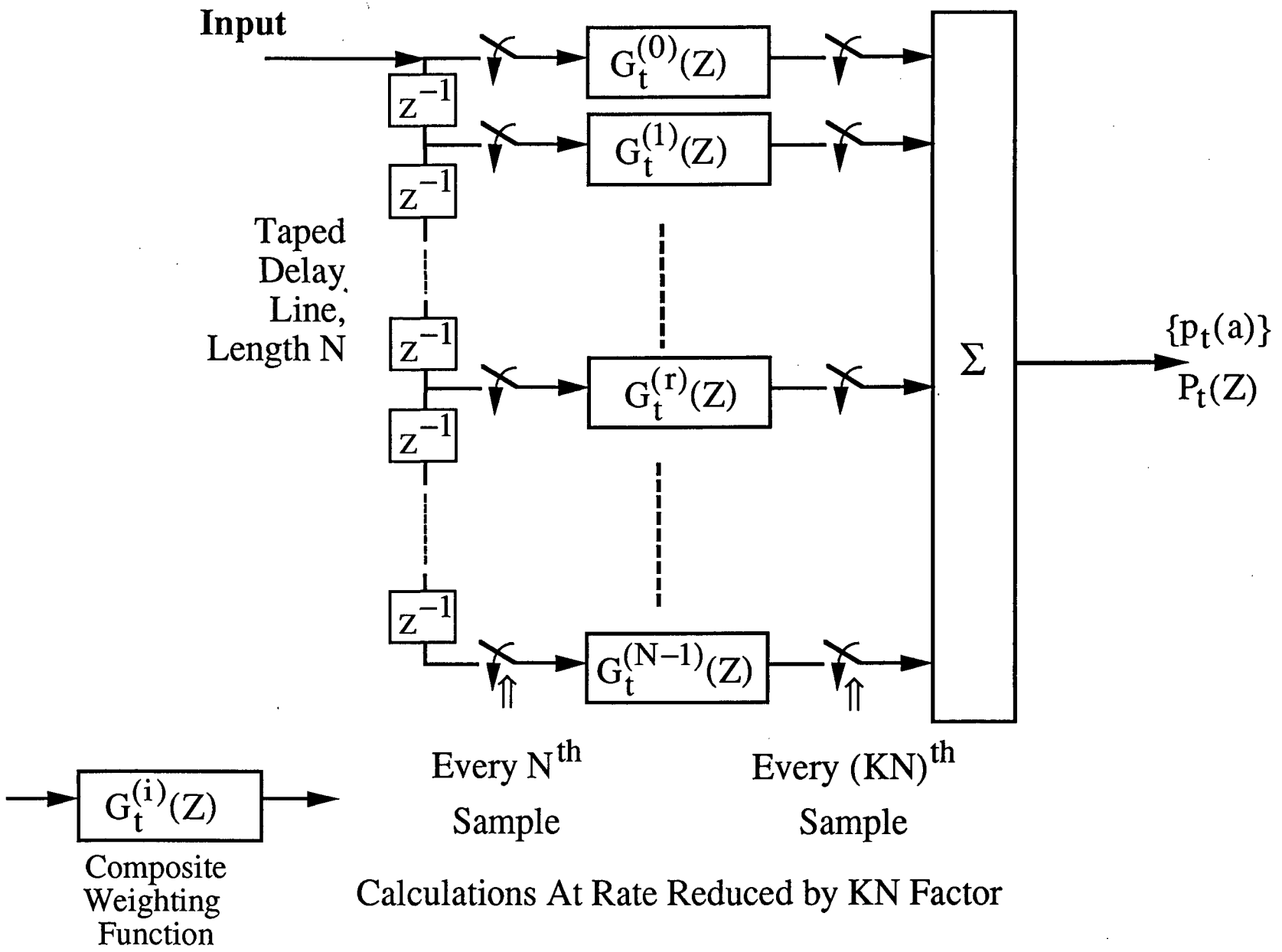
The output sample index a is scaled by K in the argument of $g_t^v(\ )$ inside the definition of $p_t(a)$, equation (22a), while it is further scaled by N in this definition, equation (22b). The net effect has the input data weighted by values every N^{th} point, in steps of KN with respect to the data indices. There are alternate ways of rearranging the above equations to demonstrate this more clearly, however, the overall weighting functions do not reduce as compactly. In this alternate arrangement, the input data are weighted at sample instances in multiples of KN even though the summations employ values at steps of N, for all fixed offset indices v and d. A schematic description of the parity generation associated with filter $H_t(Z)$ is shown in Figure 16, where the Z transform of the composite weighting functions are employed.

$$G_t^{(v)}(Z) = \sum_{s=-\infty}^{+\infty} g_t^{(v)}(s)\, z^{-sN} \qquad \begin{matrix} v = 0, 1, \ldots, N-1 \\ t = 0, 1, \ldots\ L-1 \end{matrix} \tag{23}$$

The real savings in computing the respective channel parities occur for the case of uniform filters at the critically sampled rate, L = N. With the filter bank as in Figure 4, the outputs of each $H_t(Z)$ are scaled by a complex phasor, $\{W^{tv}\}$, as in equation (7). This translates the parity channel output $p_t(a)$ into a modified equation (22).

$$p_t(a) = \sum_{v=0}^{N-1} \sum_{u=-\infty}^{+\infty} \left[x(uN+v)W^{tv}\right] g^{(v)}(ak-u) \tag{24}$$

The uniform filter weighting function $g^{(v)}(s)$ is defined similarly to equation (22b), with index t dropped. The complex roots of unity are functions only of the outer index v, and, when all N channels are considered, the complete set of parity values may be calculated by a DFT operation, as described earlier with regard to the polyphase multirate filter banks. The calculation rate is reduced by a factor KN, even though the individual composite channels accept data at intervals of N. The index v in $g^{(v)}(s)$ determines the offset in the composite weighting function and the data, seen in equations (22b) and (24). The effective



Decomposition of Parity Filter for Channel t
Figure 16

computational rate for the composite parity calculation process is indicated in Figure 17, showing how all parity values are obtained simultaneously through the DFT.

5. Protecting a Polyphase Filter Demultiplexing System

The basic protection philosophy was outlined previously in Figure 10 which depicted the method by showing a generic channel. The parity values are calculated in two ways, one by a parallel composite parity generation process as described in the last section. The second comparable parity values are computed directly from the channel's demultiplexed output. The first set of parities are calculated according to equations (22) employing the composite weighting. The other parity estimates are computed by a formula following the form of equation (21). These two versions of $p_t(a)$, labeled $p_t'(a)$ and $p_t''(a)$ are compared in a totally self-checking comparator. The combined protection system is detailed for generic channels r in Figure 18; identical calculations for each of the N outputs would be made. This figure also includes an A/D subassembly based on the rotating use of a small number of A/D converters as noted earlier with regard to Figure 8. The protection of this important unit will be presented at the end of this section.

The full details of this generalized version of a totally self-checking equality checker [7] are contained in a forthcoming book chapter [25]. A description of this self-checking comparator is presented in Figure 19. The threshold value Δ is selected to allow small differences between the two versions of comparable parity samples, accounting for roundoff noise discrepancies arising because they are computed by different subsystems. The parity weight filters, $G^{(v)}(Z)$ blocks in Figure 18, combine the effects of Q(Z) and H(Z), equations (22) and (23). However, the computational rate is reduced further by a factor of K, making this scheme an efficient protection approach. Since each channel compares a pair of parity values every K^{th} output value, errors are detected with a latency of at most K output samples. The detecting capability of the code is sometimes specified in

terms of the minimum distance for a constraint length (i = m in equation (17)) implying that a group of errors up to the level of d_m in each constraint length can be discerned.

The A/D conversion process which translates analog signals into corresponding digital sample approximations can be a source of a single point failure from which no recovery is possible. However, for practical high-speed applications separate A/D converters are combined and operate in a rotating round-robin fashion. Such a configuration is indicated at the beginning of Figure 18, and, for the sake of discussion, it will be assumed that the subassembly employs s A/D converters accepting successive analog values in a round-robin fashion. The input sampling rate can be s times the maximum capability of a single A/D converter. Typically s can range from 2 to 8.

The output of the A/D subassembly is the sequence $\{x(k)\}$ where $x(k) \leftrightarrow$ digital sample x(t) at t = kT. If a single converter in the round-robin arrangement fails, the values $\{x(k)\}$ have a possibly random sequence added that can have only nonzero values every s^{th} sample. The new output of the subassembly may be modeled as the sequence $\{w(k)\}$ which contains sequence $\{\alpha_p(k)\}$ representing the possibly random error values.

$$w(k) = x(k) + \alpha_p(k) \qquad \text{; A/D Outputs, } p^{th} \text{ Converter Failed}$$

$$p = 0, 1, \ldots, (s\text{-}1).$$

Several realistic assumptions concerning the statistical properties and relationships of the input and error sequence will be made.

The input samples are drawn from a zero-mean, wide-sense stationary sequence with autocorrelation function $R_x(m)$. The additive error sequence however is nonzero only possibly for every s^{th} value depending on the index of the failed converter.

$$\alpha_p(k) = \begin{cases} \varepsilon_d & k = ds + p \\ & (k \equiv p \bmod s) \\ & \qquad ; p = 0,1,\ldots,(s-1) \\ 0 & k \not\equiv p \bmod s \qquad d, \text{ INTEGER} \end{cases}$$

Furthermore, the error values $\{\varepsilon_d\}$, viewed as a decimated sequence, is also wide-sense stationary with mean η and autocorrelation function $r_\varepsilon(b)$. The failure of a specific A/D converter is equally likely and uncorrelated with the input sequence $\{x(k)\}$.

The autocorrelation function of the composite sequence $\{w(k)\}$ may be developed.

$$E[w(k)\, w(k+m)] = R_w(m) = R_x(m) + R_\varepsilon(m)$$

The function $R_\varepsilon(m)$ is time-varying but is not dependent on the A/D failure index because of the averaging effects in the autocorrelation function's definition

$$R_\varepsilon(m) = E[\alpha_p(k)\alpha_p(k+m)] = \begin{cases} E[\varepsilon_a \varepsilon_b] & k = as + p \\ & m = (b-a)s \\ 0 & \text{OTHERWISE} \end{cases}$$

Hence, $R_\varepsilon(m) = r_\varepsilon(b)$ for $m = bs$. The Z transform of the autocorrelation sequence $R_w(m)$ may be separated in two pieces, one associated with the data sequence $\{x(k)\}$, $S_x(Z)$, and the other related to the error autocorrelation function $r_\varepsilon(b)$.

$$S_w(Z) = \sum_{m=-\infty}^{+\infty} R_w(m) Z^{-m} = S_x(Z) + S_\varepsilon(Z^s)$$

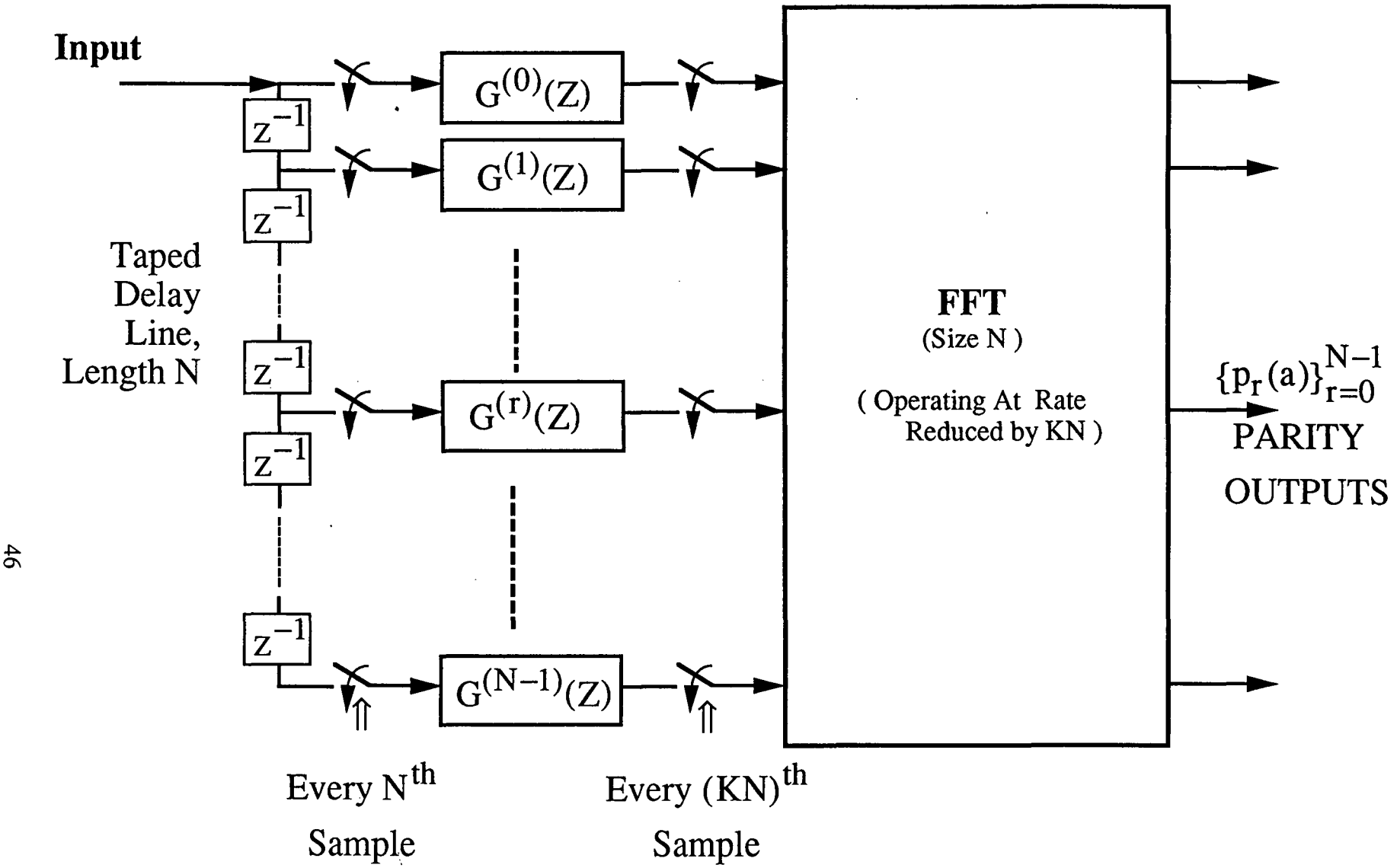
The s exponent in $S_\varepsilon(Z^s)$ is due to the separation of error values by s.

The effects of a single A/D converter are apparent from these developments. The spectrum of the corrupted sequence, $S_w(Z)$, contains s copies of the error spectrum, $S_\varepsilon(z^s)$

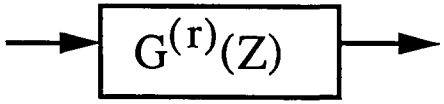
$$S_w(e^{j\omega T}) = S_x(e^{j\omega T}) + S_\varepsilon(e^{j\omega sT}) \qquad ; \text{T SAMPLING PERIOD.}$$

The resulting additive influence due to an A/D converter's failure appearing in a generic demultiplexer channel, say channel q, is determined by the cascade of $H_q(z)$ and $S_\varepsilon(Z^s)$ where $H_q(z)$ represents the prototype filter shifted to channel q. This spectral density is $|H_q(z)|^2 S_\varepsilon(z^2)$.

A protection scheme for the A/D subassembly employing the parity checking approach outlined earlier may be given. Firstly, at least one demultiplexer channel is kept vacant in the multiplexing format. This means that, except for noise, only zero values should be present at its demultiplexer output. However, the parity values associated with

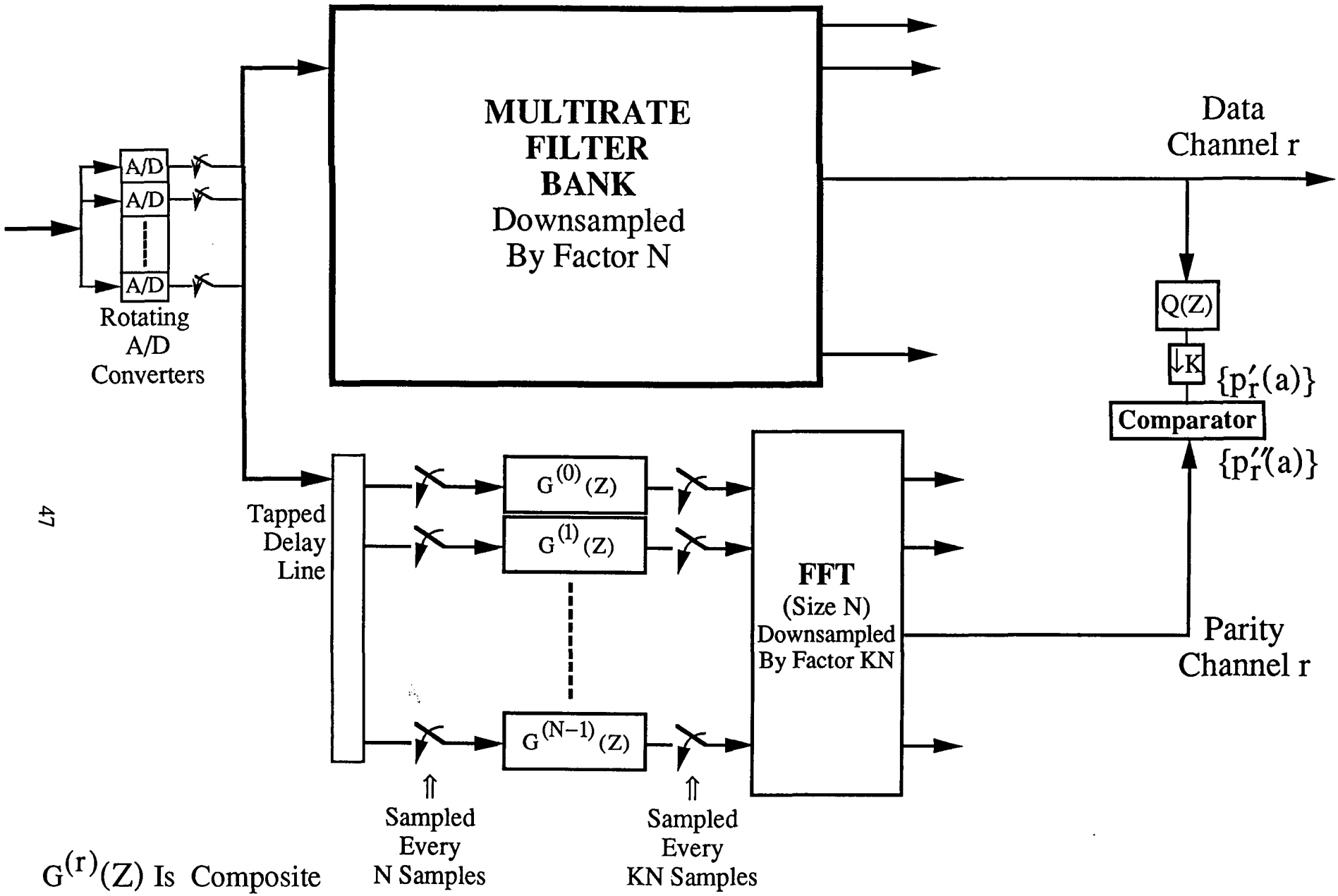


Calculations At Rate Reduced by KN Factor



Composite Weighting Function

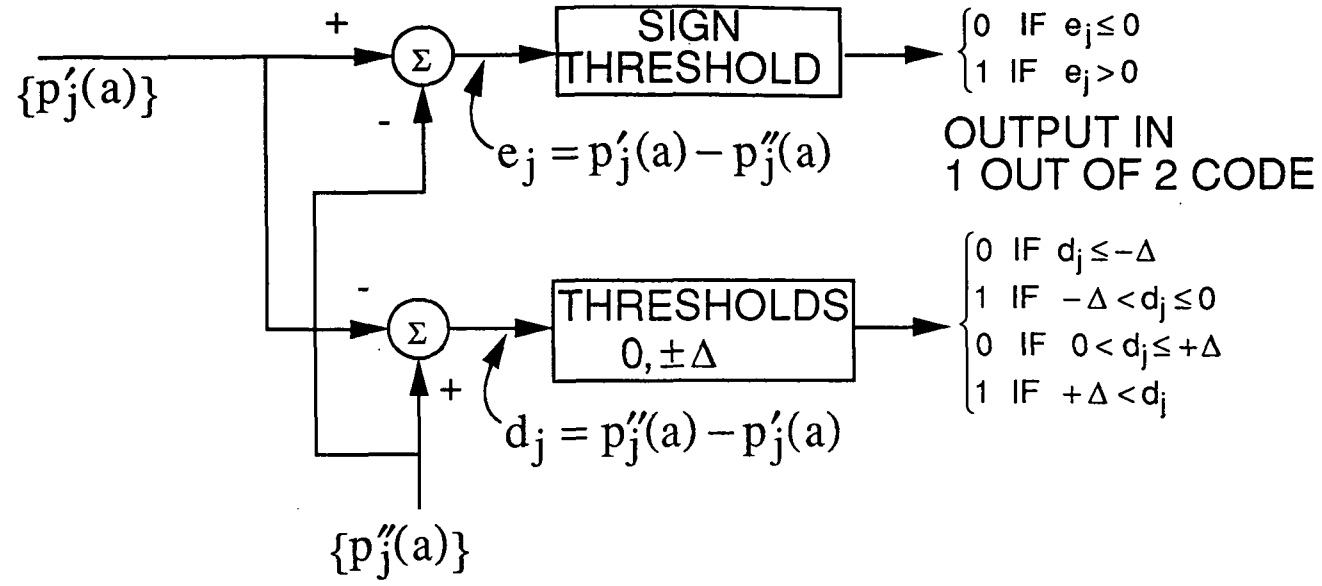
Parity Generation for Uniform, Critically Sampled Case
Figure 17

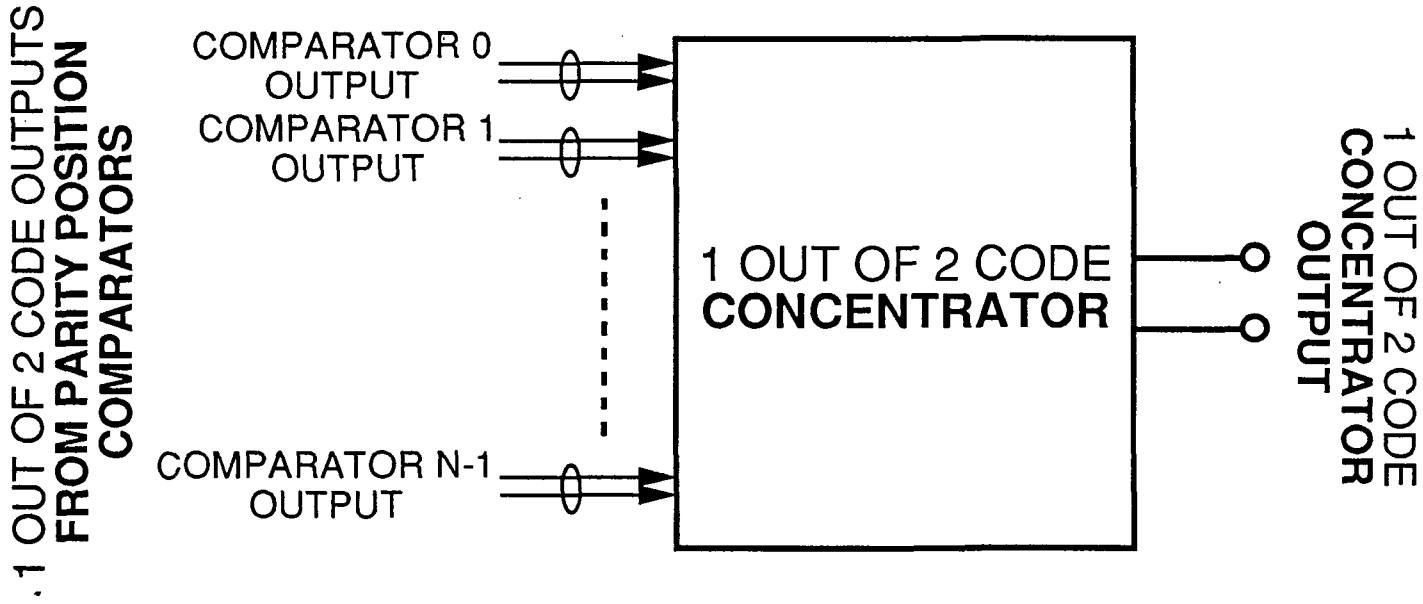


$G^{(r)}(Z)$ Is Composite Of $Q(Z)$ And $H(Z)$

Q(Z): Parity Weighting Filter
H(Z): Prototype Separating Filter

Protection of Demultiplexer Channels
Figure 18





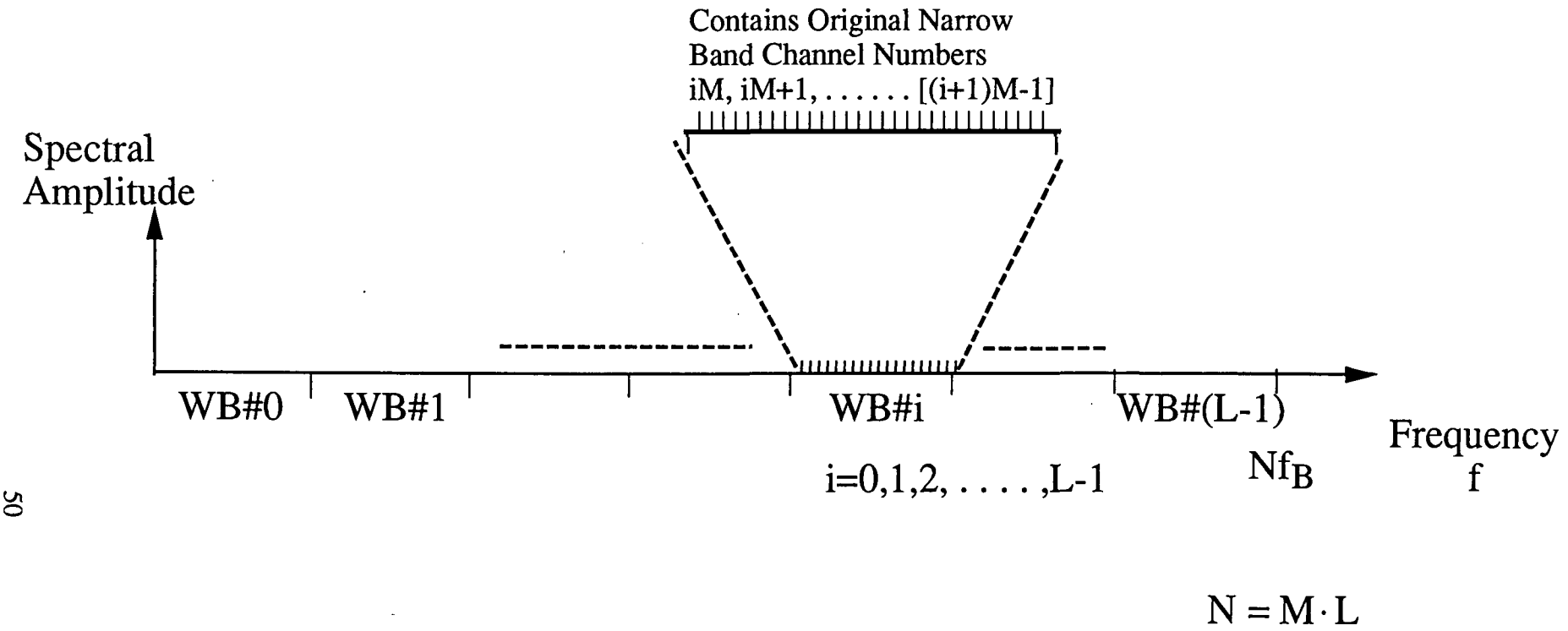
Totally Self-Checking Comparator
Figure 19

this vacant channel, say index q, is represented by $|H_q(Z)Q(Z)|^2 S_\epsilon(Z)^2$. Because the convolutional code is linear, the parity values associated with channel q that are produced in parallel should be zero too. For failure detection purposes, these conditions may be checked by a comparator judging against one zero input. Therefore, protection of the A/D subassembly is accomplished by vacating a multiplexer channel and checking the output parity stream for zero, allowing a small tolerance for channel noise effects. One guide for selecting the channel q to be vacated is to pick a channel where $|H_q(Z)\ Q(Z)|^2 S_\epsilon(Z^s)$ contains large energy.

6. Two-Level Demultiplexing

There are different bandwidth requirements for various data channels in a multiplexed system. A high-speed channel will require the same spectral space as a number of the more common, low-speed channels. Hence in many multiplex schemes, a two-tiered hierarchical approach is adopted. Several wide band (WB) channels, each occupying the bandwidth of a fixed number of narrow band (NB) channels, are extracted by the first level of demultiplexing, and for any WB channel carrying NB channels, a second level of demultiplexing is applied.

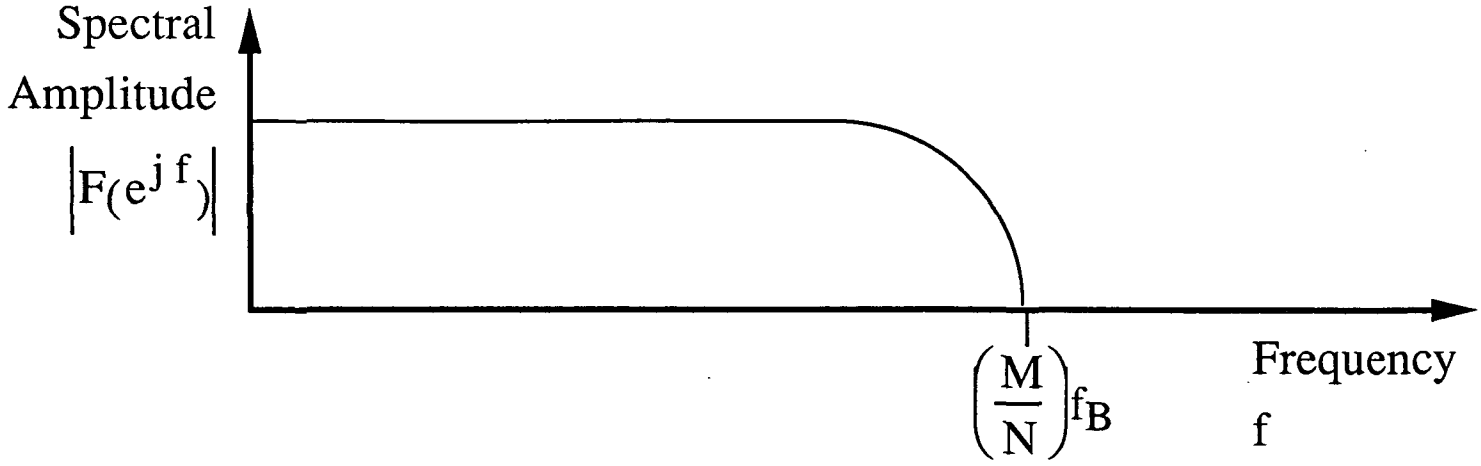
A spectral view of the hierarchical channel configuration is demonstrated in Figure 20, where the possible total number of NB channels is N, but allocated into L WB channels each capable of containing M NB channels. This requires the arithmetic relationship N = ML. In the first level of demultiplexing, the L individual WB channels are removed by filtering with a shifted prototype baseband filter having typical idealized transfer functions F(Z), depicted in Figure 21a. When a WB channel carries M NB channels, a second demultiplexer filter bank employing prototype baseband filter H(Z) is used. Such a typical filter characteristic is contrasted against F(Z) in Figure 21b. The L WB channels are separated by a familiar form, polyphase, multirate filter bank displayed in the upper part of Figure 22, producing outputs at a rate downsampled by factor L. Any WB channel



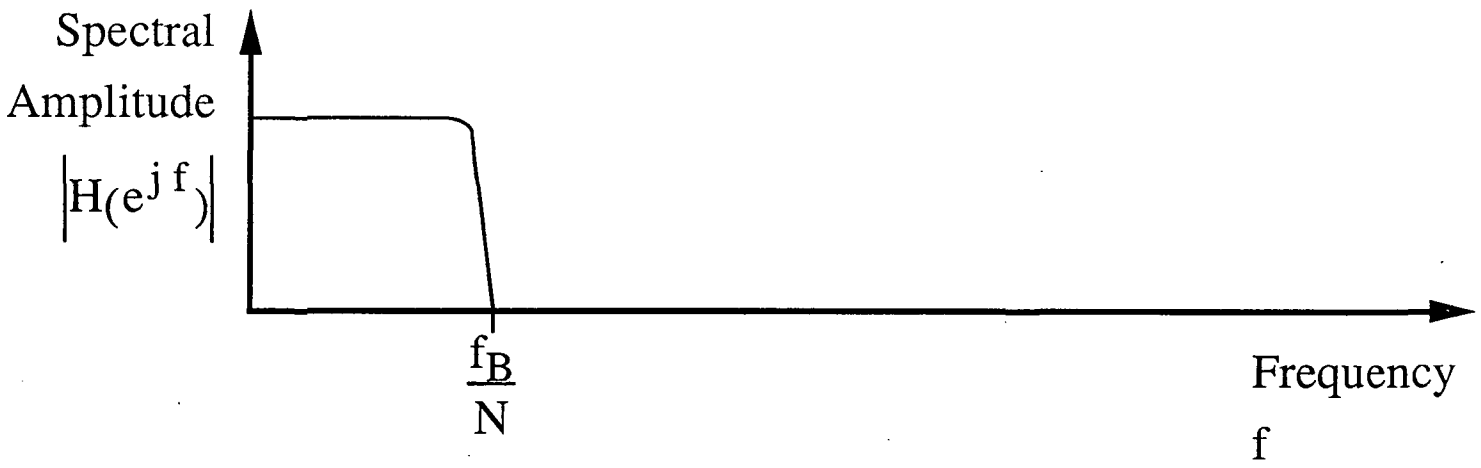
$N = M \cdot L$

L Wide Band (WB) Channels
Each Wide Band Cahnnel Can Contain M Narrow Band (NB) Channels

Two Level Multiplexing Hierarchy
Figure 20

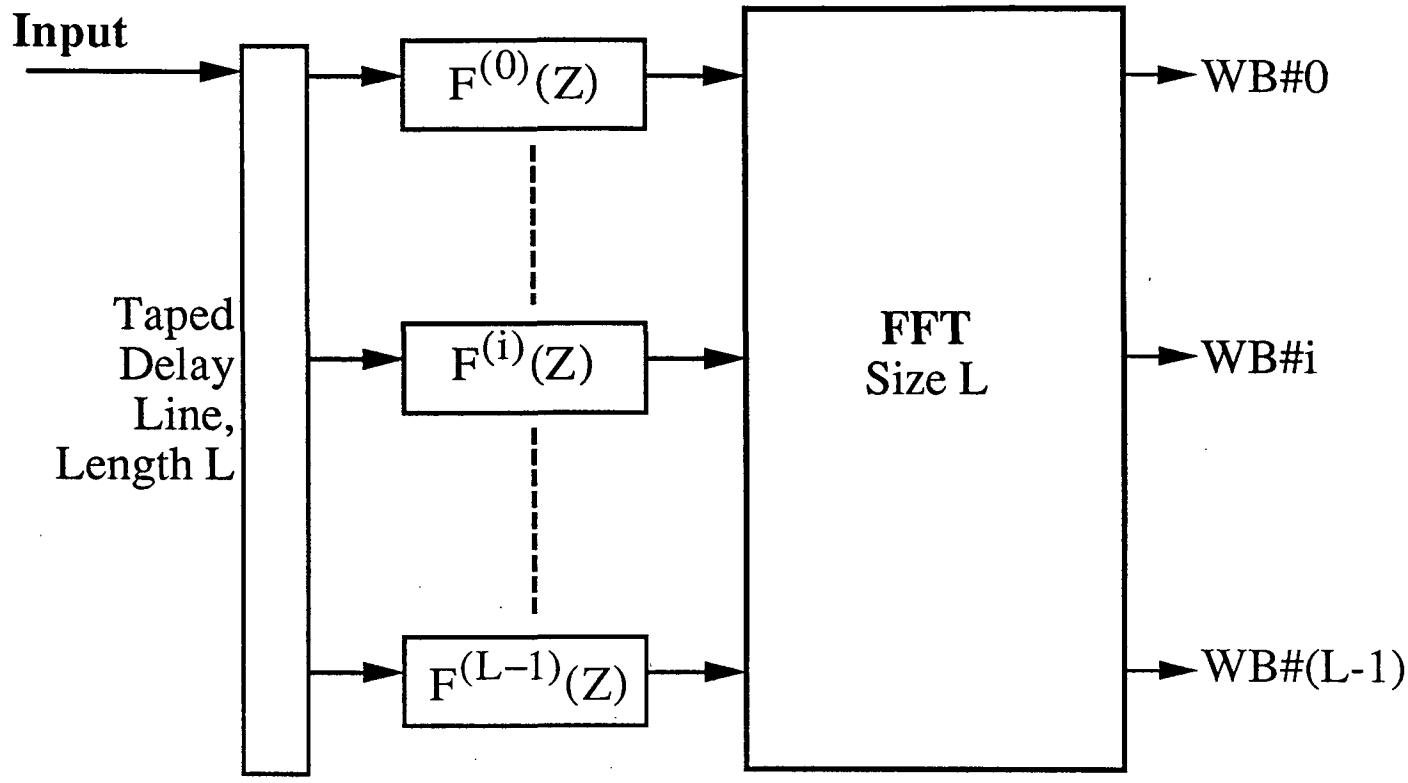


F(Z), Baseband Filter For Separating Wide Band Channels
Figure 21a



H(Z), Baseband Filter For Separating Narrow Band Channels
Figure 21b

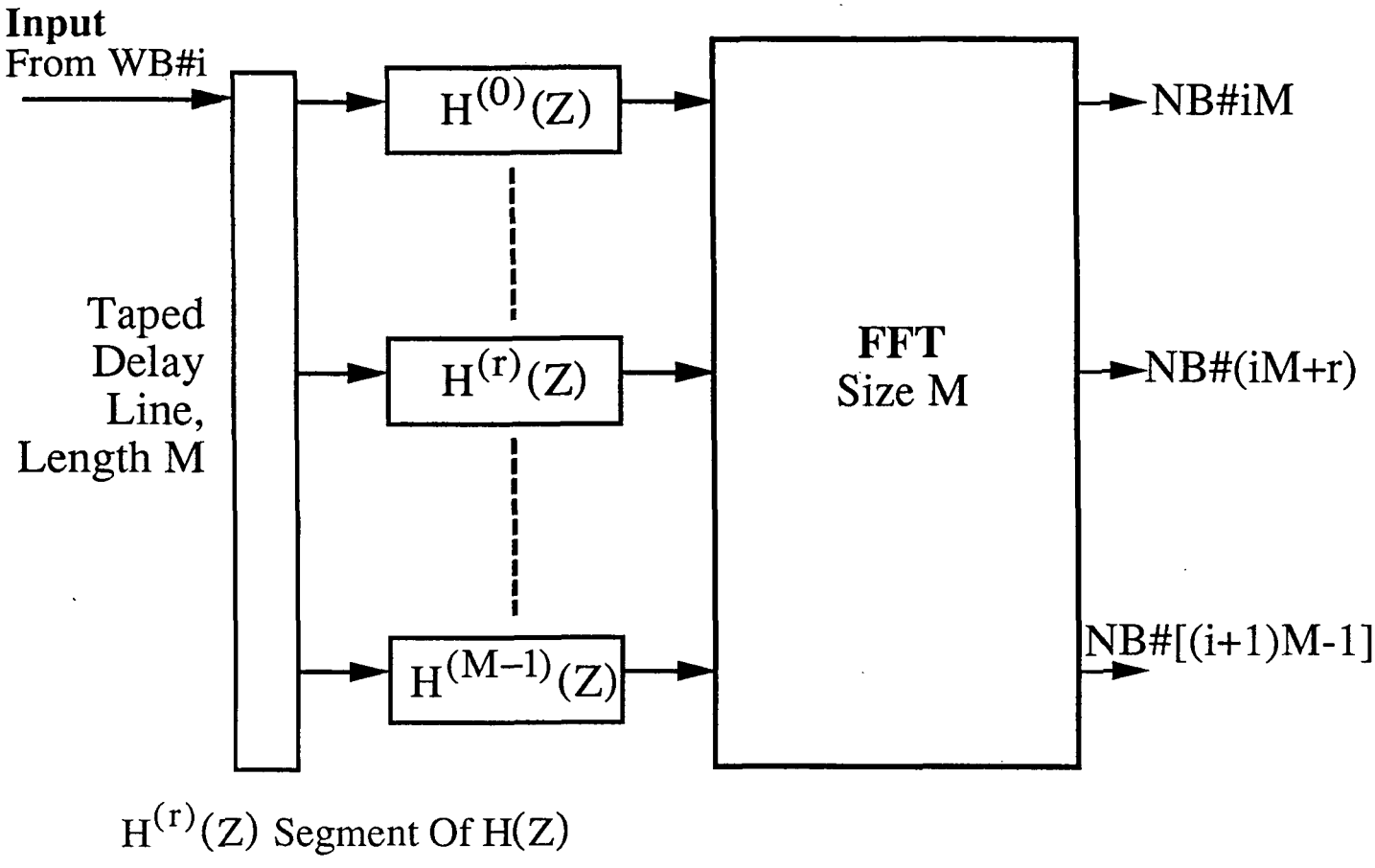
Filter Characteristics For Extracting Both
Wide Band and Narrow Band Channels
Figure 21



$F^{(i)}(Z)$ Segment Of F(Z)

Wide Band Channel Demultiplexing
Figure 22a

Two Tier Demultiplexing Approach
Figure 22



Narrow Band Channel Demultiplexing
Figure 22b

Two Tier Demultiplexing Approach
Figure 22

requiring further demultiplexing passes through a second similar filter bank operating at a rate additionally reduced by factor M. This bank is depicted in the lower part of Figure 22.

At first, it would seem that parities generated for protecting WB channel outputs could be processed further directly obtaining parity values associated with individual NB channels. However, the downsampling operations make this approach inefficient as the development in this section demonstrates.

Let the output of a parity generating filter associated with WB channel s, (s = 0, 1, ..., L-1) be denoted by sequence $\{\alpha^{(s)}(r)\}$. These parities are needed for checking the L outputs of Figure 22a. They may be expressed in terms of the input samples, $\{x(r)\}$, and the baseband filter impulse response $\{f(v)\}$ and convolutional code's parity channel weighting $\{q(v)\}$.

$$\alpha^{(s)}(r) = \sum_{c=0}^{L-1} \sum_{t=-\infty}^{+\infty} \sum_{u=-\infty}^{+\infty} x((rK-t)L-c)W_L^{-sc} \qquad ; \quad W_L = e^{j\frac{2\pi}{L}}$$

$$\{q(u)\, f((t-u)L+c)\} \quad ; \quad s = 0, 1, \ldots, L-1. \qquad (26)$$

The downsampling of the parity values by factor KL is evident in the argument of the inputs. The parity outputs for all L channels can be generated using a discrete Fourier transform (usually an FFT algorithm) as before.

On the other hand, parties associated with the M NB channels possibly occupying WB channel number s will be denoted by the sequence $\left\{\beta_{(m)}^{(s)}(r)\right\}$ where m ranges 0, 1,..., M-1. The NB prototype filter impulse response is labeled as $\{h(r)\}$.

$$\left\{\beta_{(m)}^{(s)}(r)\right\} = \sum_{c=0}^{M-1} \sum_{d=0}^{L-1} \sum_{u=-\infty}^{+\infty} \sum_{t=-\infty}^{+\infty} \sum_{p=-\infty}^{+\infty} x\Big(\big[(rK-t)M-c-p\big]L-d\Big)$$

$$q(u)f(pL+d)h((t-u)M+c)W_M^{-cm}W_L^{-sd} \qquad (27)$$

$$W_M = e^{j\frac{2\pi}{M}}$$

$$; s = 0, 1, \ldots, L-1 \ , \text{ WB Channel Index}$$

$$m = 0, 1, \ldots, M-1 \ , \text{ NB Channel Index}$$

Incidentally, this equation shows how a two-dimensional discrete Fourier transform enters naturally when two-tier demultiplexing is examined in this way. Note that the channel indices s for WB and m for NB, appear separately in the discrete Fourier transform kernels.

There are significant differences between these last two equations. In the NB case, equation (27), the parity filter weighting is interwoven first with the NB prototype impulse response $\{h(r)\}$ and then coupled to the input data downsampled by factor rate ML. However, in the WB situation described by equation (26), the parity filter weighting is applied at a downsampled rate of only factor L. The data enter each composite filter channel at totally different rates, and it does not appear appealing to further process and downsample the $\{\alpha^{(s)}(r)\}$ outputs to obtain the channel parities, $\left\{\beta^{(s)}_{(m)}(r)\right\}$.

It is a natural question to try to find a linear filter that may be downsampled at its output such that processing $\{\alpha^{(s)}(r)\}$, the s^{th} WB channel output through it and downsampling by factor M, yields one of the NB channel parity values $\left\{\beta^{(s)}_{(m)}(r)\right\}$. Let $\{\Phi(r)\}$ denote this desired impulse response and label the downsampled output when processing $\{\alpha^{(s)}(r)\}$ by the sequences $\left\{\psi^{(s)}_{(m)}(r)\right\}$

$$\left\{\psi^{(s)}_{(m)}(r)\right\} = \sum_{c=0}^{M-1} \sum_{d=0}^{L-1} \sum_{u=-\infty}^{+\infty} \sum_{t=-\infty}^{+\infty} \sum_{p=-\infty}^{+\infty} x\left(\left[(rM - tM - c)K - p\right]L - d\right)$$

$$\Phi(tM + c)q(u)f\left((p-u)L + d\right)W_M^{-cm}W_L^{-sd}. \qquad (28)$$

The goal of finding an impulse response $\{\Phi(r)\}$ making $\left\{\psi^{(s)}_{(m)}(r)\right\}$ equal to $\left\{\beta^{(s)}_{(m)}(r)\right\}$ equation (27) is frustrated by different samplings of the data, appearing in their respective arguments. In equation (28) there is a scaling by K which is not present in the earlier equation (27).

7. *Protecting Sequential Discrete Fourier Transforms with Real Convolutional Codes*

The discrete Fourier transform (DFT), generally implemented through some form of fast algorithm, is central to many signal processing systems, including the polyphase filter bank implementation. This section describes how real convolutional codes can be employed to protect any discrete Fourier transform realization. Input data are grouped and then transformed, by weighting with appropriate roots of unity and summing, into another group which represents a spectral decomposition of the original data. A common viewpoint considers the input data as a vector with the resulting transformed output providing the respective weights attached to sinusoidal basis vectors in a spectral reconstruction. In numerous cases, the input data represent a segmentation of the sequential data flow with the DFT continuously operating on input data vectors as they are formed.

The discrete Fourier transform of N data samples, possibly complex-valued, uses an N^{th} root of unity W in the following sum formula

$$Y_p = \sum_{i=0}^{N-1} X_i W^{ip} \quad ; \quad W = e^{j\frac{2\pi}{N}} \quad p = 0, 1, \ldots, N-1. \tag{29}$$

The N data samples $X_0, X_1, \ldots, X_{N-1}$, produce N transform coefficients $Y_0, Y_1, \ldots, Y_{N-1}$ which describe the contribution of powers of the complex phasor W^{-p} in the reconstruction of the data samples. This reconstruction is defined through the inverse transform employing the complex conjugate of W unity and contains a normalizing factor, $\frac{1}{N}$.

$$X_i = \frac{1}{N} \sum_{p=0}^{N-1} Y_p W^{-pi} \quad ; \quad i = 0, 1, \ldots, N-1. \tag{30}$$

Vector and matrix notation provides a compact equivalent representation of the DFT, viewing the input data as the N×1 column vector $\underline{X}$. The resulting transform coefficients appear in N×1 vector $\underline{Y}$ after applying the N×N DFT matrix Ω.

$$\underline{Y} = \Omega\underline{X} \quad ; \quad \Omega = ((W^{ij})) \qquad (31)$$

row index i, column index j,

i, j = 0, 1, ..., N−1

If the data are segmented sequentially into blocks of size N, the input and transform vectors may be indexed with superscripts indicating the sequential progression. The DFT operation matrix Ω is applied to each input vector in succession.

$$\underline{Y}^{(r)} = \Omega\underline{X}^{(r)} \quad ; \quad \text{Sequential index r} \qquad (32)$$

$$r = 0, 1, 2, \ldots$$

A systematic convolutional code determines the parity samples to be associated with a stream of data by finite impulse response (FIR) linear filters.

$$Q(Z) = q_0 + q_1Z^{-1} + q_2Z^{-2} + \ldots + q_{M-1}Z^{-(M-1)} \quad ; \quad M = (m+1)K. \qquad (33)$$

The most recently arrived input data samples are weighted by $q_0, q_1, \ldots, q_{K-1}$, respectively; the next most recent group are weighted by $q_K, q_{K+1}, \ldots, q_{2K-1}$, etc., with the m^{th} most recent block scaled by $q_{mK}, q_{mK+1}, \cdots, q_{(m+1)K-1}$.

The first step towards protecting the outputs of the DFT operation, Figure 23, is to consider the parity generation process using the outputs $\{Y_i^{(r)}\}$, the i^{th} component of the transform vector $\underline{Y}^{(r)}$ with r representing the sequential index. The respective output parity values will be labeled $P_i^{(s)}$, where sequence index s refers to the parity associated with current input group $Y_i^{(sK)}, Y_i^{(sK-1)}, \ldots, Y_i^{(sK-(K-1))}$. The parity value $P_i^{(s)}$, also involves previous samples of the i^{th} transform coefficient due to the FIR filter memory.

$$P_i^{(s)} = \sum_{a=0}^{M-1} q_a \, Y_i^{(sK-a)}. \qquad (34)$$

This parity generation process is depicted in Figure 24a for one stream of output transform coefficients, where the symbol $\downarrow K$ denotes that the output is decimated by a factor K.

The N parity values associated with every new group of K output transform coefficients are denoted by parity vector $\underline{P}^{(s)}$, containing the N elements $P_i^{(s)}$, i=0, 1, ... , N-1.

$$\underline{P}^{(s)} = \sum_{a=0}^{M-1} q_a \underline{I}_N \underline{Y}^{(sK-a)}. \tag{35}$$

This is shown in Figure 24b. The algorithm-based fault tolerance philosophy requires also that comparable parity values be generated directly from the input data $\{\underline{X}^{(r)}\}$, hopefully in an efficient manner. In order to understand how comparable parity estimates $\{\underline{Y}^{(sK-a)}\}$ may be generated directly, substitute the respective values of $\hat{\underline{P}}^{(s)}$ from equation (32).

$$\hat{\underline{P}}^{(s)} = \sum_{a=0}^{M-1} q_a \underline{I}_N \Omega \underline{X}^{(sK-a)}. \tag{36}$$

This may be rewritten because the matrix identity commutes with all matrices, showing how the parity estimate vector is associated with input vector $\underline{X}^{(r)}$.

$$\hat{\underline{P}}^{(s)} = \Omega \left[\sum_{a=0}^{M-1} q_a \underline{I}_N \underline{X}^{(sK-a)} \right]. \tag{37}$$

An algorithmic fault tolerance approach is shown in Figure 25 which contains the parity generation process with the embedded DFT operation according to equation (37). Since there is a new parity estimate vector produced only every K vector inputs, the DFT operation, no matter how it is implemented, is performed at a rate reduced by a factor K with regard to those in the normal processing channel. The two parity vectors $\underline{P}^{(s)}$ and $\hat{\underline{P}}^{(s)}$ are compared in a fault-tolerant totally self-checking comparator (see Figure 19) which permits a small threshold Δ to exist between related components. The comparable parity components are computed by two methods and therefore may incur different roundoff errors. These small errors could be mistaken for system failures without this threshold.

The use of binary convolution codes over the real numbers eliminates the need for multiplications in the parity generation FIR filters. On the other hand, the necessity of storing (mK) values for each parity channel in implementing the FIR filter actions can be mitigated by forming the (m+1) sums associated with every group of K data values as they arrive. Consider the operations for encoding the input data stream according to equation (37), ultimately producing an element of $\hat{\underline{P}}^{(s)}$. Similar approaches apply to the FIR filtering

needed to apply to the transform coefficients $\underline{Y}^{(s)}$ yielding parity values in $\underline{P}^{(s)}$, equation (35). Suppressing the subscripts indicating vector components for the moment, but retaining the superscripts indexing the arrival sequence, the arriving data $X^{(sK)}$, $X^{(sK-1)}$, ..., $X^{((s-1)K+1)}$ are used in forming intermediate sums, $\zeta_j^{(s)}$, that will be used shortly in forming parity values.

$$\zeta_j^{(s)} = \sum_{i=0}^{K-1} X^{(sK-i)} q_{jK+i} \quad ; \quad j = 0, 1, \ldots, m. \tag{38}$$

These segment sums will be used in subsequent parity estimates according to the following correspondence:

$$\zeta_j^{(s)} \quad \leftrightarrow \quad \hat{p}^{(s+j)}$$

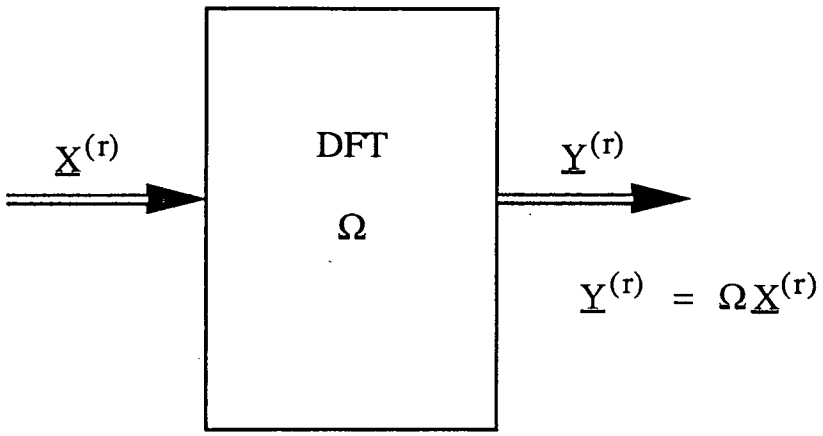
The parity estimate at index s, $\hat{p}^{(s)}$, is formed from (m+1) of these intermediate sums.

$$\hat{p}^{(s)} = \sum_{r=0}^{m} \zeta_r^{(s-r)}. \tag{39}$$

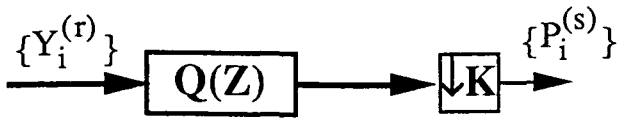
When a new segment of K input values arrives, (m+1) intermediate values are formed and $\zeta_0^{(s)}$ is used currently in $\hat{p}^{(s)}$. However, the remaining m intermediate values, $\zeta_j^{(s)}$, j = 1, 2, ..., m are saved. The array intermediate values needed for each new parity estimate $\hat{p}^{(s)}$ is indicated in Figure 26. They form a triangular array at each moment with the current parity estimate formed by summing on the diagonal as shown by the arrow in Figure 26. The number of intermediate values carried forward to be used when the next group of K values arrives can be computed by the following equation; they are grouped above the dotted right angle in Figure 26.

$$\sum_{i=0}^{m-1} (m-i) = \frac{m(m+1)}{2} \quad ; \quad \begin{array}{l}\text{Number of Intermediate} \\ \text{Values Carried Forward}\end{array} \tag{40}$$

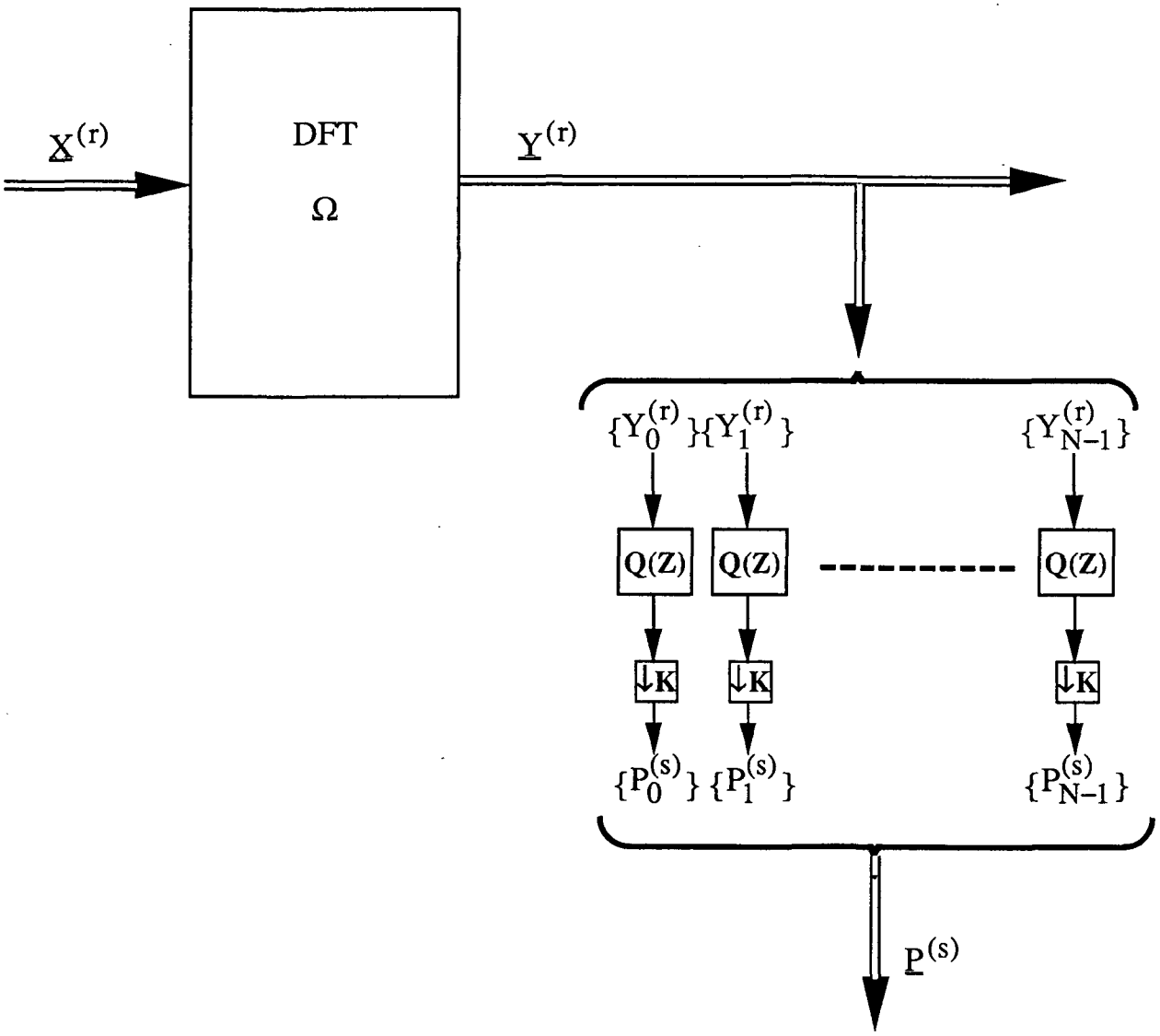
An approximation of the number of operations needed to compute the two comparable sets of parities, $\underline{P}^{(s)}$ and $\underline{\hat{P}}^{(s)}$, may be developed. For each group of K input data vectors, there are at most $2N((m+1)(K-1)+m)$ summations. Assuming that an FFT is used to



Sequential Vector DFT Processor
Figure 23

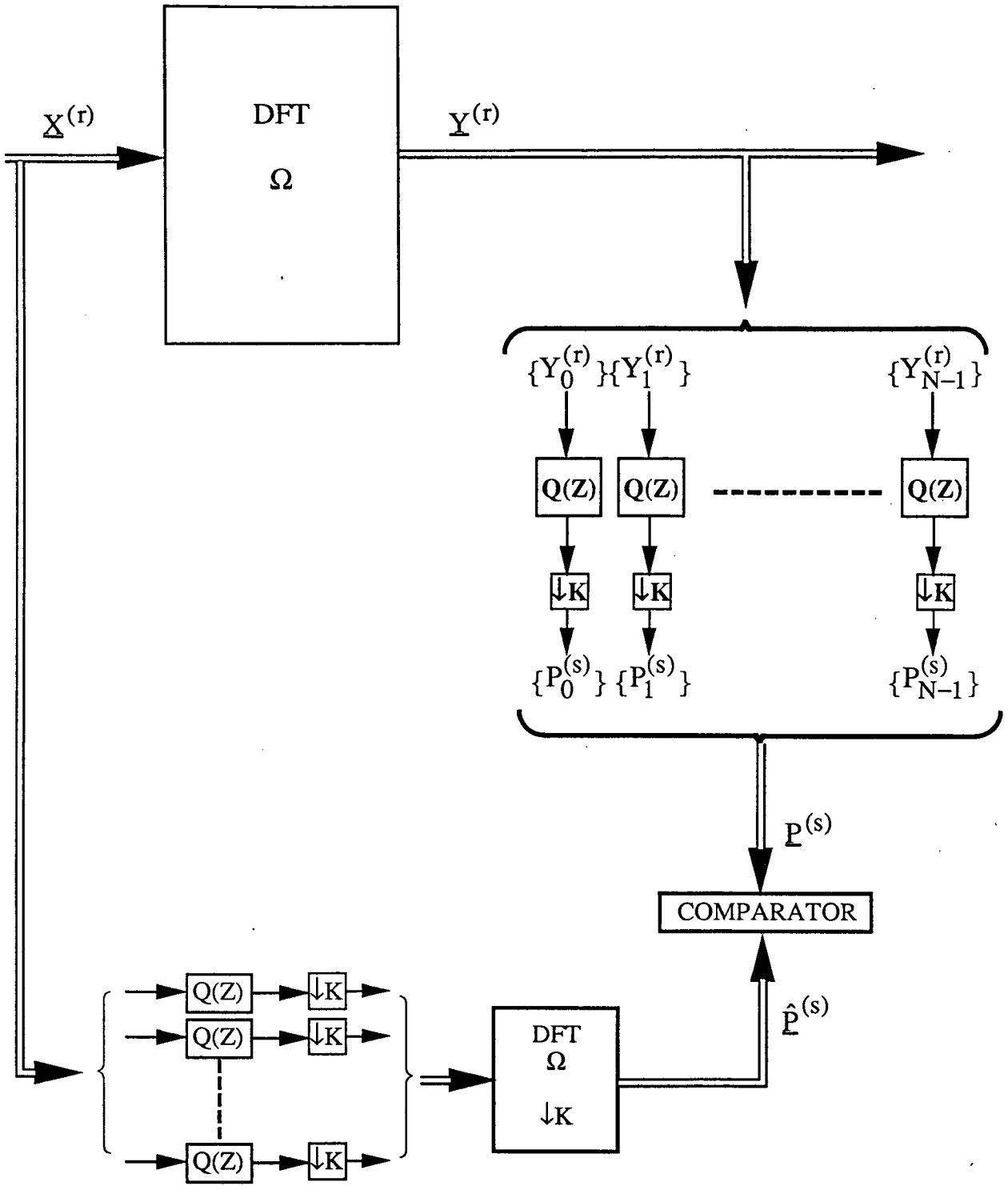


Parity Generation for Transform Coefficient i
Figure 24a

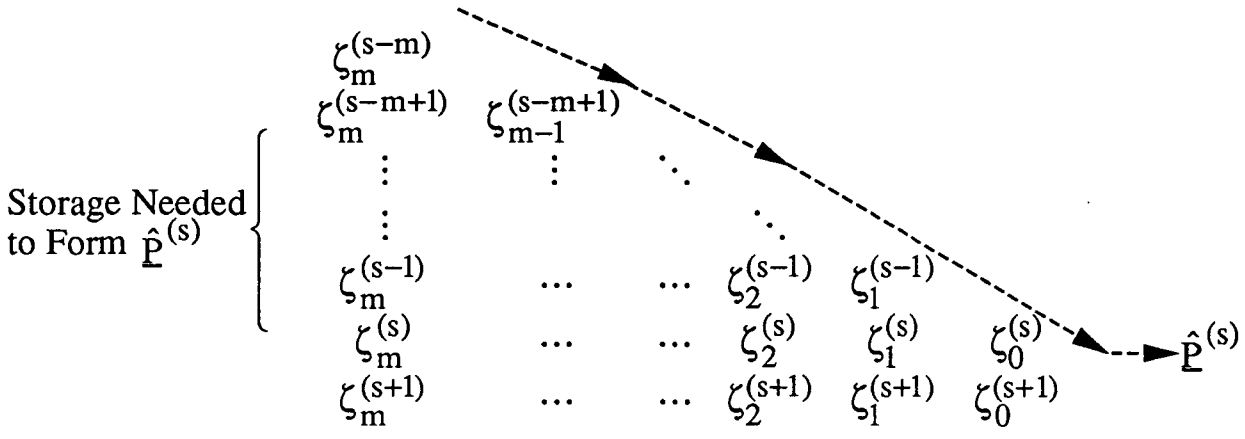


Parity Generation at Transform Output
Figure 24b

Parity at DFT Output
Figure 24



Protection Through Parity Generation and Regeneration
Figure 25



Memory Requirements for Intermediate Values at Index s
Figure 26

implement the DFT operations, giving about $N\log_2 N$ additions and multiplications, a comparison of the parity overhead is possible. Including the 4N differences required in the totally self-checking comparators, Figure 19, the totals, normalized by factor K because of the lower computational rate in the parity channels are at most

$$\frac{2N}{K}\left[(m+1)(K-1)+m+2+\frac{1}{2}\log_2 N\right] \quad \text{SUMMATIONS}$$

and

$$\frac{N}{K}\log_2 N \quad \text{MULTIPLICATIONS.}$$

On the other hand, the DFT implementation employs $N\log_2 N$ summations and multiplications. Hence the scaling factor K plays a significant role in lowering the parity computational overhead on a per-input sample basis. Unfortunately, the storage requirements are roughly $2N\frac{m(m+1)}{2}$, whereas a standard FFT with interstage storage uses about $N\log_2 N$ locations.

8. Single Parity Channel Real Convolutional Codes

The protection methods presented for filter banks and discrete Fourier transform realizations employ single parity channel convolutional codes. This subsection demonstrates that such codes exist in abundance. In particular, an especially useful class of burst correcting convolutional codes are described in detail to exemplify code construction techniques. As noted earlier, the generalized concept of distance is interrelated with the linear independence of columns of the parity-check matrix for these kinds of codes. The easily proved result, stated formally in [23], guarantees that real codes constructed by directly mapping prime finite field elements into the integers in a natural way have minimum distance properties at least equal to the original finite field code.

The real codes are utilized so that a single parity sample is produced for every group of k data samples (see Figure 14). Thus, any combination of failures in every group of k

processed data samples needs to be detectable. The concept of burst errors covers this error situation very nicely. A burst error is a contiguous inclusive group of possible errors always starting and ending with an error. This model handles the onset of errors in a group of processed data without constricting the exact nature of individual intervening errors. This model contrasts with the situation where errors occur randomly in unspecified positions throughout a constraint length of processed data. Since the detection procedure checks parity validity in a continuous block fashion, the onset of a burst of errors up to the full length of (k+1) samples is detected regardless of the failure mechanism.

An (n,k) convolutional code with constraint block length parameter m is defined by the encoding matrix G, equation (10) and subsequent equations. However, this matrix is fully defined by a submatrix containing only the upper left (km × nm) elements because of the repeating subblock structure. This finite size matrix is denoted by $[G]_m$. The corresponding parity-check ((n–k)m × nm) submatrix denoted by $[H]_m$ obeys the annihilating requirements first given in equation (20).

The starting position of any burst may be assumed to be on a boundary related to subblocks of length as dictated by the encoding action. Using the same indexing notation for the encoded digits of equation (15b), a single burst error E of length b may be exemplified by

SINGLE BURST OF LENGTH b

$$E = (0, 0, \ldots, 0, e_j^{(0)}, e_j^{(1)}, \ldots, e_{j+b_o}^{(c_o-1)}, 0, 0, \ldots, 0, \ldots)$$

$$b = b_o n + c_o \quad ; \quad 0 \le c_o < n \quad , \quad b \ge 1$$

$$e_j^{(0)} \neq 0 \quad , \quad e_{j+b_o}^{(c_o-1)} \neq 0$$

There are several classes of well-known burst-correcting convolutional codes such as the Berlekamp-Preparata-Massey codes and the Iwadare codes, well-documented in standard textbooks [21, Chapter 14; 22, Chapter 14]. The code design relies on showing

that the effects of a burst of length b starting in one code subblock can be distinguished from another nonoverlapping burst starting elsewhere, provided there is suitable guard space containing no bursts in between. For fault protection purposes, detection properties are all that are needed, but these classes are very efficient so they represent good choices. However, the possibility of single burst correction capabilities may be useful if temporary errors should need to be removed. Correction procedures are described briefly at the end of this section.

The design for burst convolutional codes starts with the parity-check matrix $[H]_m$, expressed here in one of its systematic forms. The parity submatrices P_i, i = 0, 1, ..., m, appeared earlier in equations (11).

$$[H]_m = \begin{pmatrix} I_{n-k} & -P_0^T & 0 & 0 & \cdots & 0 & 0 \\ 0 & -P_1^T & I_{n-k} & -P_0^T & \cdots & \vdots & \vdots \\ \vdots & \vdots & 0 & -P_1^T & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \cdots & 0 & 0 \\ 0 & -P_m^T & 0 & -P_{m-2}^T & \cdots & I_{n-k} & -P_0^T \end{pmatrix}; \quad [(n-k)m \times nm] \qquad (41)$$

Each P_i^T is $(n-k) \times k$

The stacks representing the parity filter weighting function Q(z) are obvious in this format.

As an example of an efficient burst-correcting high rate, k/k+1, convolutional codes, the Berlekamp-Preparata-Massey class will be designed. This presentation is similar to the one in [21, Chapter 14], where the parity channel weighting positions are slightly different from those indicated in equation (10). Since the parity values are generated in parallel, this is of no consequence. The constraint parameter for this class is m = 2(k+1) and the parity-check matrix's first 2(k+1) columns are given by a special [2(k+1) × (k+1)] submatrix B_0.

$$B_o = \begin{pmatrix} 0 & 0 & I_{k+1} & 0 \\ 0 & & \cdots & \\ \vdots & & T & \\ 0 & & & \end{pmatrix} \qquad 2(k+1) \times (k+1) \qquad (42a)$$

T is a $k \times k$ skewed triangular matrix with the only possible nonzero elements below the skew diagonal, i.e.,

$$t_{pq} = 0 \text{ if } q \le k - p.$$

$$T = \begin{pmatrix} 0 & 0 & \cdots & \cdots & \cdots & 0 & a \\ 0 & 0 & \cdots & \cdots & 0 & c & b \\ 0 & 0 & \cdots & 0 & f & e & d \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad k \times k \qquad (42b)$$

Elements a, b, c, d, e, ... indicate the possible nonzero positions. How they are determined will be discussed shortly.

The complete construction of parity-check matrix $[H]_m$ proceeds by adding new groups of n = (k+1) columns by using shifted versions of B_0. Each new group is derived from the previous (k+1) columns by shifting that submatrix down one row and inserting an all-zero row at the top. The $[H]_m$ that emerges has each group of successive columns represented by $[2(k+1) \times (k+1)]$ submatrix B_{i+1} obtained from the preceding B_i by applying a row shift operator R.

$$[H]_m = (B_0 \;\; B_1 \;\; B_2 \;\; \ldots \;\; B_{2k+1}) \qquad (43a)$$

$$B_{i+1} = RB_i \quad , \quad i = 0, 1, \ldots, 2k+1 \qquad (43b)$$

$$R = \begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 0 \\ & & & & & 0 \\ & & I_k & & & \vdots \\ & & & & & 0 \end{pmatrix} \quad \begin{matrix} (k+1) \times (k+1) \\ \text{ROW SHIFT OPERATOR} \end{matrix} \qquad (43c)$$

The unspecified possibly nonzero entries in T, defining B_0, are determined by the requirements that all (2k+1) submatrices of the form $(B_0\ B_i)$, i = 1, 2, ..., 2k+1 be nonsingular (a square [2(k+1) × 2(k+1)] matrix). The fact that each B_i is a shifted version allows these (2k+1) conditions to be ordered so that the individual possibly nonzero terms in T can be found in succession starting from a, b, c, etc. (See equation (42b).)

There are many choices for the variables in T that satisfy the conditions, but a set that uses binary values is particularly appealing. There are tables of solutions [21], [22] and an example for burst correction up to 6 samples (k = 5) is given below. One possible set of nonzero values in T gives values for a through j with all being 1 except i = 0. The parity submatrices may be listed and the corresponding parity filter weighting terms are given in Table 1.

$$-P_0^T = 000\ 00 \qquad -P_6^T = 000\ 00$$
$$-P_1^T = 100\ 00 \qquad -P_7^T = 000\ 01$$
$$-P_2^T = 010\ 00 \qquad -P_8^T = 000\ 11$$
$$-P_3^T = 001\ 00 \qquad -P_9^T = 001\ 11$$
$$-P_4^T = 000\ 10 \qquad -P_{10}^T = 010\ 11$$
$$-P_5^T = 000\ 01 \qquad -P_{11}^T = 100\ 11$$

These codes also have a simple correcting procedure. First a syndrome S is formed by assembling the data and parity samples for a constraint length of m = 2(k+1) blocks of (k+1); these are denoted by the vector $\underline{r}$.

$$\underline{S} = \underline{r}\,[H]_m^T \qquad \underline{r}\ \ 1 \times m(k+1)\ \text{SAMPLES} \qquad (44)$$

$$[H]_m^T\ \ m(k+1) \times m\ \text{Parity Check Matrix}$$

This 1 × m syndrome vector $\underline{S}$ may be separated into two (k+1) parts.

$$\underline{S} = (\underline{S}', \underline{S}'') \qquad \underline{S}', \underline{S}''\ \ 1 \times (k+1)\ \text{Components of Syndrome} \qquad (45)$$

If the burst is confined to the first subblock of length (k+1), these two groups of syndrome positions must be interrelated, as will be demonstrated next. Let $\underline{E}$ be a $1 \times (k+1)$ vector representing a burst in the first (k+1) positions of the constraint length of symbols being considered in $\underline{r}$. Then the rows of $[H]_m^T$ corresponding to these positions in product (44) are represented by B_0^T. Hence the syndrome for this special case has the form:

$$\underline{S} = \underline{E}\, B_0^T. \tag{46a}$$

Also under these same conditions the two parts of $\underline{S}$, $\underline{S}'$, $\underline{S}''$, take the following form because of the format of B_0, equation (42a).

$$\underline{S}' = \underline{E}\, I_{k+1} \tag{46b}$$

$$\underline{S}'' = E \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \mathbf{T} & \\ 0 & & & \end{pmatrix} \tag{46c}$$

Note that the burst error $\underline{E}$ appears intact in the first part, $\underline{S}'$, which is easily identified in the syndrome $\underline{S}$ computed from the assembled positions in $\underline{r}$. However, combining equations (46b) and (46c) shows how the two parts of $\underline{S}$ must be related.

$$\underline{S}'' = \underline{S}' \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \mathbf{T} & \\ 0 & & & \end{pmatrix} \tag{47}$$

The code design guarantees that if condition (47) is verified using syndrome $\underline{S}$, the first part $\underline{S}'$ gives the burst values which are confined to the (k+1) positions of the first subblock. This can be performed for each subblock as it is processed, so the scheme will catch and correct the onset of errors up to the burst correcting capability of the code, (k+1). However, extra calculations are required to generate the syndrome from the combined data and parity positions.

Table 1: Q(Z) Terms for Rate 5/6 FIR Parity Filter

$q_{24} \rightarrow$	0	$q_{49} \rightarrow$	0		
	0		0		
	0		1		
	1		1		
	0		1		
$q_{19} \rightarrow$	0	$q_{44} \rightarrow$	0		
	0		0		
	1		0		
	0		1		
	0		1		
	0		0		
	1		0		
	0		0		
	0		0		
	0		1		
$q_9 \rightarrow$	1	$q_{34} \rightarrow$	0	$q_{59} \rightarrow$	1
	0		0		0
	0		0		0
	0		0		1
	0		0		1
	0		0		0
	0		0		1
	0		0		0
	0		0		1
$q_0 \rightarrow$	0	$q_{25} \rightarrow$	1	$q_{50} \rightarrow$	1

III. PROTECTING VITERBI TYPE CONVOLUTIONAL DECODERS

1. Role of Convolutional Codes

Each data channel passing through the satellite is encoded with a binary convolutional code for combating noise introduced by the transmission medium. This noise is generally modeled as white Gaussian noise (WGN) added to the outputs of the demultiplexer. The demodulator's operation is predicated on this assumption of WGN affecting the samples from the demultiplexer. A simplified overview of one data stream's convolutional code protection is shown in Figure 27. At the encoder, data are grouped by subblocks of k bits while the n binary digits actually transmitted are a function of the present k bits and the m previous subblocks, the memory in the encoder. Thus, the rate of the code is k/n. Figure 28 explicitly shows the bits in the encoder memory that are used in determining each group of n output bits.

The algebraic description of the encoding operation is almost identical with that explained in Section II, with the exception that the systematic encoding is not enforced here. The information bits are not necessarily separately identifiable. This only constrains the exact forms of the submatrices G_i in equation (10); they are no longer required to have the structures in equations (11). It is the distance structure of these codes that determines performance improvements when employed through a WGN environment with the signal-to-noise ratio at the demultiplexer the determining factor. In particular, if the noise power level changes temporarily, this performance level changes accordingly, and the output of the decoder can produce undetected errors because the correcting capability of the code has been exceeded. These errors are determined in a statistical fashion. The assumption underlying the code design choices directly affect the probability of bit error at the FEC decoder's output.

The demodulator for a coherent system produces decision variables that represent the relative confidence that a transmitted symbol over a baud was sent as a particular binary

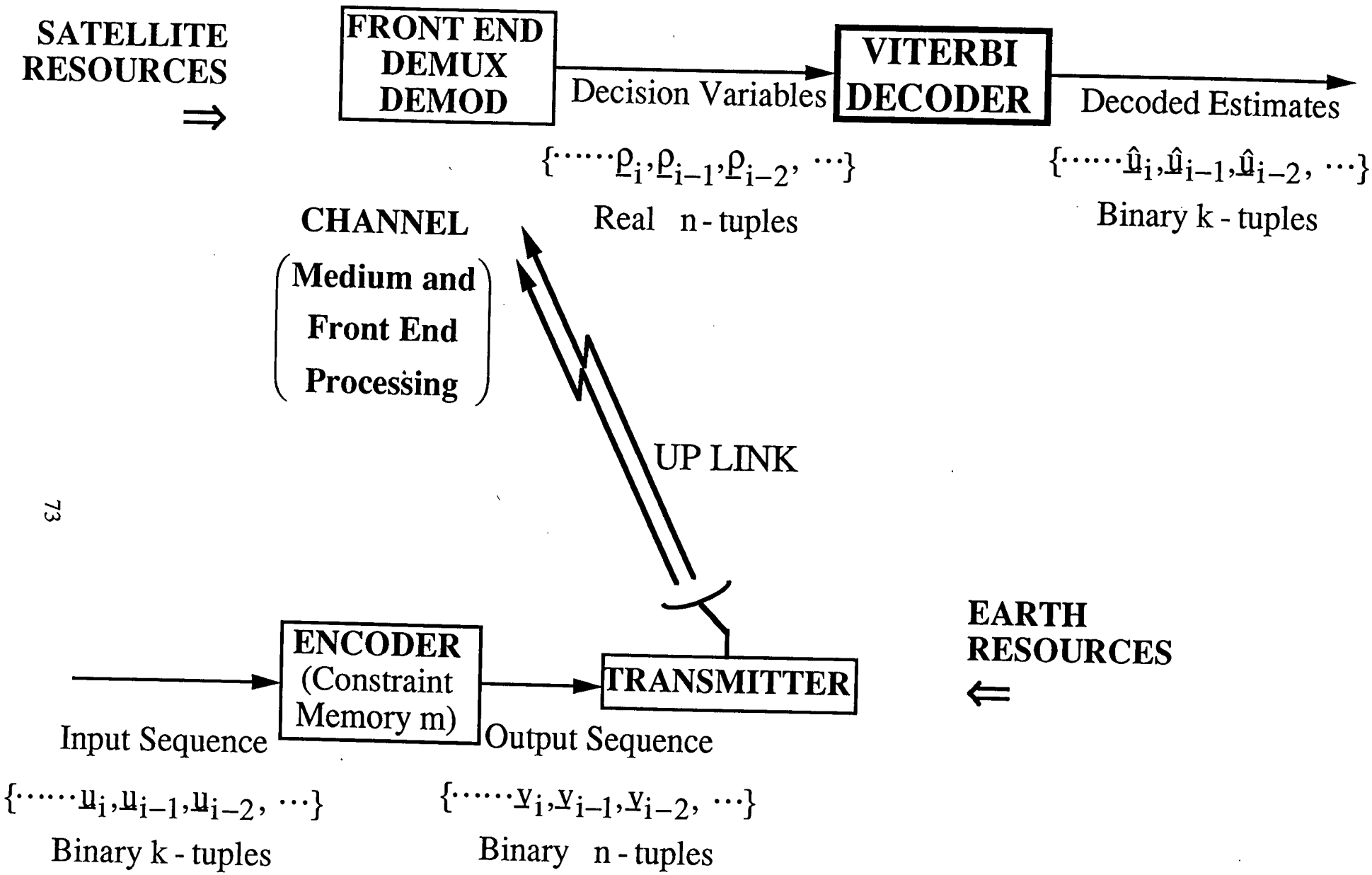
value. The match filter's output is a random variable which may be described in part by a conditional Gaussian density function [31, Chapter 1]. However, these variables are generally quantized so that they are adequately represented by a few bits of precision: three to four bits per sample are an adequate number [31, Section 1.3.5]. Nevertheless, the decision variables corresponding directly to confidence levels of n code symbols may be viewed as an n-tuple of real numbers, as indicated in the upper part of Figure 27. Each n vector $\underline{\rho}_i$ corresponds to the original n bits in vectors $\underline{v}_i$ comprising the encoded stream. The role of the decoder is to decide which related information bits in $\underline{u}_i$ originally entered the encoder by only observing the output confidence levels from the demodulator. The decoding is optimal in some statistical sense, and there is a delay between the original input bits entering the encoder in the earth resources and the associated decoded bits leaving the decoder, besides the long propagation delay in the transmission path for satellites.

2. *Decoding Convolutional Codes*

Optimal decoding involves the *maximum a posteriori* (MAP) estimation of the encoder's state sequence given the observation of the demodulator's soft decision outputs [21-22, 30-32]. This MAP estimator minimizes the uniform cost function wherein all errors are equally costly [29, Section 2.4]. The first step in establishing the decoding operations centers on the concept of an encoder state. As indicated in Figure 28, the encoder determines the n outputs bits, $\underline{v}_i$, based upon the present k input bits in $\underline{u}_i$ and the m groups of previous input bits contained in the k vectors, $\underline{u}_{i-1}, \underline{u}_{i-2}, \ldots, \underline{u}_{i-m}$. This suggests a natural definition of a physical encoder state as determined by the (mk) bits in this latter group of memory bits. The state vector is denoted by $\underline{x}_i$.

$$\underbrace{\underline{v}_i}_{\substack{\text{Present}\\\text{Outputs}}} \leftarrow \underbrace{\underline{u}_i}_{\substack{\text{Present}\\\text{Inputs}}} \underbrace{\underline{u}_{i-1}\ \underline{u}_{i-2}\ \cdots\cdots\ \underline{u}_{i-m}}_{\substack{\text{Encoder}\\\text{Memory}}} \tag{48}$$

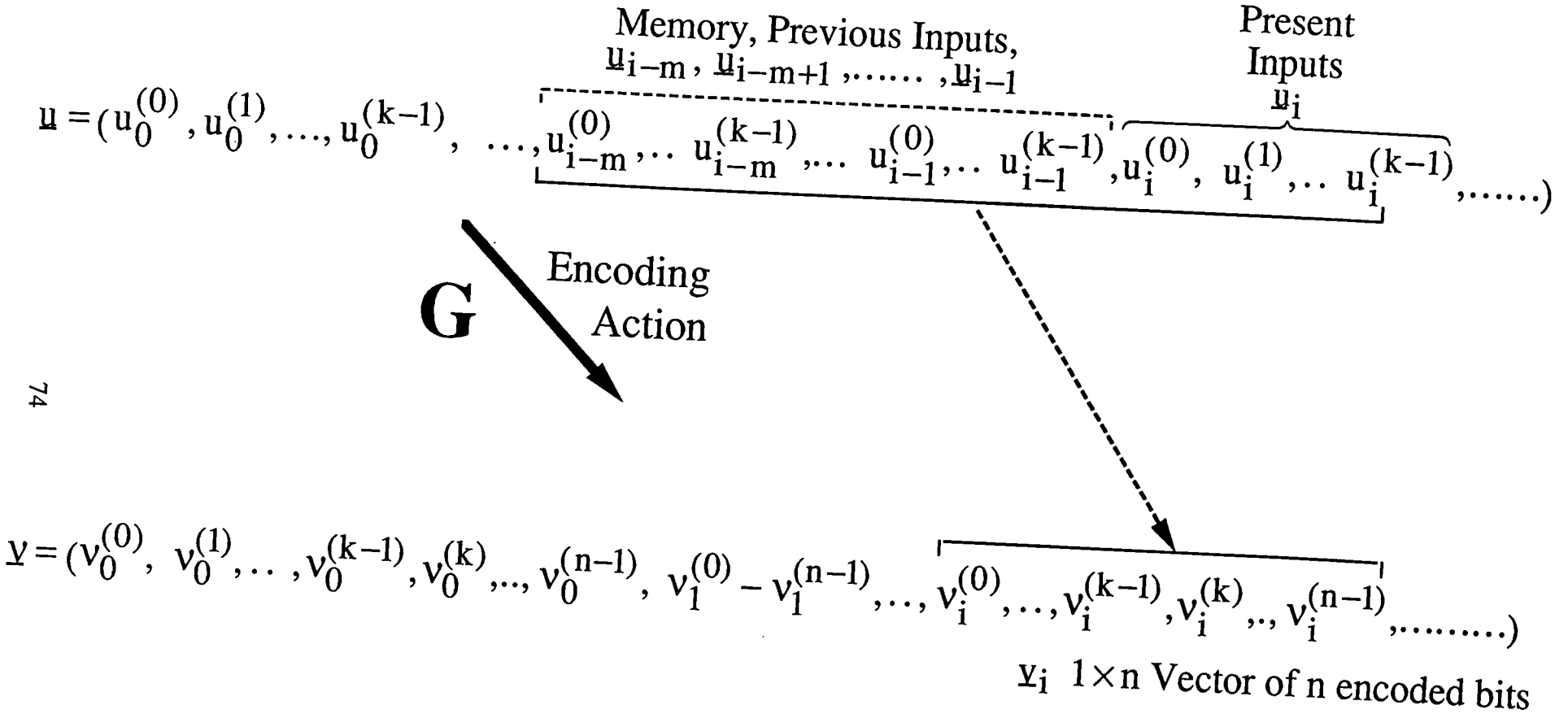
$$\underline{x}_i = (\underline{u}_{i-1}\ \underline{u}_{i-2}\ \cdots\cdots\ \underline{u}_{i-m})$$



Convolutional Code Protection of Data Channels
Figure 27

Rate k/n binary convolutional code uses generator matrix G:

$$\underline{u}\,G = \underline{v}$$



k bits are expanded to n bits by each encoding action which also involves memory of m groups of input bits.

Encoding Action,
Rate k/n Convolutional Code with Memory Constraint Parameter m
Figure 28

The encoder state space at the index i is labeled $\underline{X}_i$; it contains 2^{mk} possible elements.

Two adjacent states $\underline{x}_i$ and $\underline{x}_{i+1}$ overlap in (m–1)k vectors, including the k present input bits at index i, $\underline{u}_i$. Thus, it is reasonable to define a transition as the couple of adjacent states.

$$\underline{\xi}_i = (\underline{x}_{i+1}, \underline{x}_i) \leftrightarrow (\underline{u}_i\ \underline{u}_{i-1} \cdots \cdots \underline{u}_{i-m+1}) \quad ; \quad (m+1)k = M \text{ Bits} \qquad (49)$$

$$\text{where } \underline{x}_{i+1} = (\underline{u}_i\ \underline{u}_{i-1} \cdots \cdots \underline{u}_{i-m+1})$$

Because of the duplication of many k-bit input groups, the encoding operation, denoted here by function f( ), is definable directly on the transition space.

$$\underline{v}_i = f(\underline{\xi}_i) \qquad ; \text{ ENCODING FUNCTION f( )} \qquad (50)$$

$$= f((\underline{x}_{i+1}, \underline{x}_i)) = f(\underline{u}_{i-1}\ \underline{u}_{i-2} \cdots \cdots \underline{u}_{i-m})$$

The transition space, labeled $\underline{\Xi}_i$, contains $2^{(m+1)k}$ vectors because of the overlapping inputs contained in the defining couple, equation (49).

The demodulator outputs preserve the integrity of the bit positions as dictated by the encoder output $\underline{v}_i$. Furthermore, the statistical properties of the channel may be taken as memoryless [30-32], implying that the conditional probabilities associated with related n vectors through the medium may be factored into probability functions describing individual components.

$$P(\underline{\rho}_i/\underline{v}_i) = \prod_{r=0}^{n-1} P(\rho_i^{(r)}/v_i^{(r)}) \qquad (51)$$

The notational conventions established earlier and appearing in Figure 28 are used, particularly with regard to the components of the subblocks.

The decoder examines the soft decision variables from the demodulator over a long sequence of samples in order to estimate the state sequence over the comparable space of state transitions. Generically, a span of L transition vectors in the encoder is considered

$$\underline{\xi}_i^{(L)} = (\underline{\xi}_i \ \underline{\xi}_{i-1} \ \cdots \ \cdots \ \underline{\xi}_{i-L+1}) \tag{52}$$

The related soft decision variables, after the effects of these transitions are transmitted, demultiplexed and demodulated are consolidated in a sequence of n-tuples.

$$\underline{\rho}_i^{(L)} = (\underline{\rho}_i \ \underline{\rho}_{i-1} \ \underline{\rho}_{i-2} \cdots \ \cdots \ \underline{\rho}_{i-L+1}) \tag{53}$$

Individual n-tuples directly correspond to encoded bits which are in turn related to individual transitions.

$$(v_i^{(0)}, v_i^{(1)}, \ldots \ldots, v_i^{(n-1)}) = \underline{v}_i = f(\underline{\xi}_i) \leftrightarrow \underline{\rho}_i = (\rho_i^{(0)}, \rho_i^{(1)}, \ldots \ldots, \rho_i^{(n-1)}) \tag{54}$$

The memoryless property of equation (51) translates into a factorization of the conditional probabilities associated with the decision sequence, $\underline{\rho}_i^{(L)}$, given by the transition sequence, $\underline{\xi}_i^{(L)}$.

$$P(\underline{\rho}_i^{(L)} / \underline{\xi}_i^{(L)}) = \prod_{j=0}^{L-1} P(\underline{\rho}_{i-j} / \underline{\xi}_{i-j}) \tag{55}$$

The MAP estimation goal is to find a good replica of the original transition sequence $\underline{\xi}_i^{(L)}$. It is assumed that the probability of the initial state $\underline{x}_{i-L+1}$, $P(\underline{x}_{i-L+1})$, is known for all values in the state space $\underline{X}_{i-L+1}$. The sequence estimate $\underline{\xi}_i^{(L)}$ is denoted by $\hat{\underline{\xi}}_i^{(L)}$, with all of its vector components carrying a circumflex too.

$$\hat{\underline{\xi}}_i^{(L)} = (\hat{\underline{\xi}}_i, \hat{\underline{\xi}}_{i-1}, \ldots \ldots, \hat{\underline{\xi}}_{i-L+1}) \quad \text{MAP ESTIMATES} \tag{56}$$

The minimum uniform error estimates required in MAP dictates a search over a finite, although large, group of transitions.

$$P(\hat{\underline{\xi}}_i^{(L)}, \underline{\rho}_i^{(L)}) = \max_{\forall \underline{\xi}_i^{(L)}} \{P(\underline{\xi}_i^{(L)}, \underline{\rho}_i^{(L)})\} \quad ; \quad \text{Provided } P(\underline{x}_{i-L+1}) \text{ known .} \tag{57}$$

The state transition sequence depends on the progression of states because of relationships (49).

$$\underline{\xi}_i^{(L)} = (\underline{\xi}_i \ \underline{\xi}_{i-1} \ \cdots \ \cdots \ \underline{\xi}_{i-L+1}) \leftrightarrow (\underline{x}_{i+1} \ \underline{x}_i \ \cdots \cdots \ \underline{x}_{i-L+1}) \tag{58}$$

Furthermore, the encoding operation guarantees a Markov property among the progression of these states:

$$P(\underline{x}_{i+1}/\underline{x}_i, \underline{x}_{i-1}, \cdots \cdots, \underline{x}_{i-L+1}) = P(\underline{x}_{i+1}/\underline{x}_i) \tag{59}$$

The usual simplifications lead to the factorization and separation of probability expressions where the probability of the initial state $P(\underline{x}_{i-L+1})$ is assumed known [30-32].

$$P(\hat{\underline{\xi}}_i^{(L)}, \underline{\rho}_i^{(L)}) = \max_{\forall \underline{\xi}_i^{(L)}} \left\{ P(\underline{x}_{i+1-L}) \bullet \prod_{j=0}^{L-1} P(\underline{x}_{i+1-j}/\underline{x}_{i-j}) \bullet \prod_{r=0}^{L-1} P(\underline{\rho}_{i-r}/\underline{\xi}_{i-r}) \right\} \tag{60}$$

The maximization process required to find the MAP sequence is not altered by mapping the argument of the max{ } operation using any function that is monotonic on the unit interval. A logarithm function has the additional feature of transforming products into summations.

$$\ln\left[P(\hat{\underline{\xi}}_i^{(L)}, \underline{\rho}_i^{(L)}) \right] = \max_{\forall \underline{\xi}_i^{(L)}} \left\{ P(\underline{x}_{i+1-L}) + \sum_{j=0}^{L-1} \ln[P(\underline{x}_{i+1-j}/\underline{x}_{i-j})] + \sum_{r=0}^{L-1} \ln[P(\underline{\rho}_{i-r}/\underline{\xi}_{i-r})] \right\} \tag{61}$$

The individual terms identified with the soft decision variables from the demodulator and the transitions in the encoder may be given symbols called branch metrics, a descriptive name whose purpose will be developed next.

$$\lambda(\underline{\xi}_{i-j}) = \ln[P(\underline{x}_{i+1-j}/\underline{x}_{i-j})] + \ln[P(\underline{\rho}_{i-r}/\underline{\xi}_{i-r})] \quad ; \tag{62}$$

BRANCH METRIC ASSOCIATED WITH TRANSITION $\underline{\xi}_{i-j}$

The maximizing process may be done recursively and incrementally by attaching values to the finite states in each state space $\underline{X}_{i-L+1}, \underline{X}_{i-L+2}, \ldots, \underline{X}_i, \underline{X}_{i+1}$ underlying all possible transition sequences. A useful aid in visualizing this sequential maximization process is the concept of a trellis diagram [30-32]. At each index i, there are 2^{mk} states in space $\underline{X}_i$, each being assigned an individual node and with each connected by a directed arc to those states in state space $\underline{X}_{i+1}$ that constitute elements in the transition space $\underline{\Xi}_i$ dictated by the encoding action. These transitions, labeled generically as $\underline{\xi}_i$, connect states $\underline{x}_i$ and

$\underline{x}_{i+1}$. A description of the spaces involved in a trellis diagram for transitions between states at adjacent indices i and i+1 is given in Figure 29a. A complete trellis diagram starting from states at index (i–L+1) through those at index (i+1) displays all transitions in sequence $\underline{\xi}_i^{(L)}$. The Viterbi algorithm relies on the fact that only paths from the beginning to states at intermediate indices which yield maximum values need to be preserved [30-32].

The cumulative maximum metrics at each intermediate state will be denoted by $\Gamma(\underline{x}_r)$ where the initial values at the states in space $\underline{X}_{i+1-L}$ are given by the known quantities,

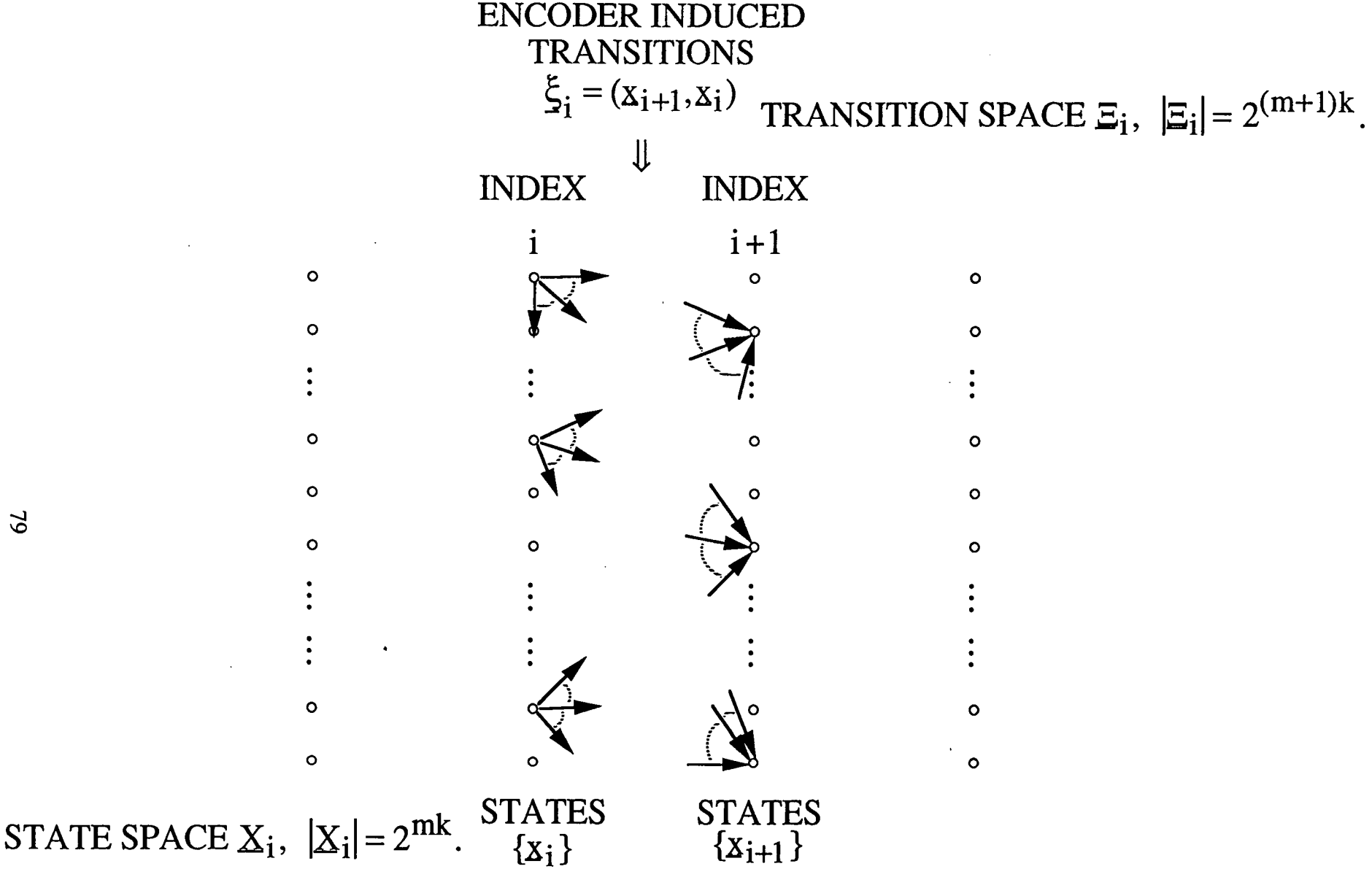
$$\Gamma(\underline{x}_{i+1-L}) = \ln(P(\underline{x}_{i+1-L})).$$

The new path metrics at intermediate states are defined recursively from path metrics at the state level just preceding using the branch metrics, equation (62):

$$\Gamma(\underline{x}_{r+1}) = \max_{\forall \underline{x}_r} [\Gamma(\underline{x}_r) + \lambda(\underline{\xi}_r)] \qquad (63)$$

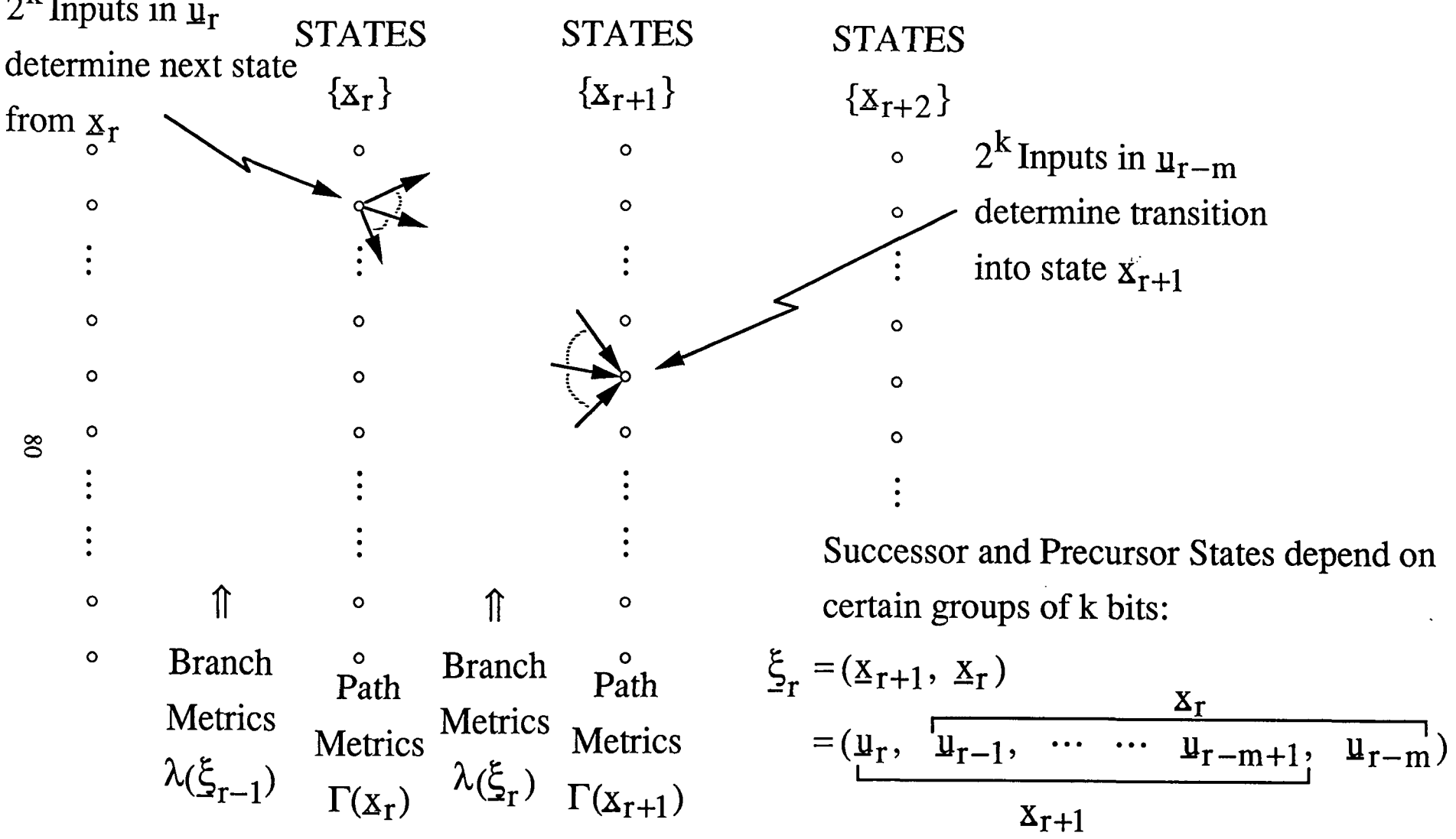
$$\text{Part of } \underline{\xi}_r = (\underline{x}_{r+1}, \underline{x}_r)$$

The path and branch metrics involved in the Viterbi algorithm are shown in Figure 29b where the input patterns that govern each transition are indicated also. As new demodulator soft decision variables are considered, the combination of branches from beginning states at index (i+1–L) define paths forward through the trellis diagram. At each new state index and for each state node at this level, a path called the survivor will be selected. The survivor path at each state is determined by choosing the path that has the highest path metric impinging on this state. The input k subblock that defines the branch back one stage level on the maximum path is appended to the previously selected maximum path up to that stage's path. These paths may have common branches, particularly the further removed from the present state under consideration. The final MAP decision for the sequence of decision variables in $\underline{\rho}_i^{(L)}$ is the path $\hat{\underline{\xi}}_i^{(L)}$ with the largest path metric among those at state index (i+1), $\Gamma_{max}(\underline{x}_{i+1})$.



State and Transition Spaces Underlying Trellis Diagram
Figure 29a

COMPONENTS OF TRELLIS DIAGRAM
Figure 29



Path and Branch Metrics for Decoding with Trellis Diagram
Figure 29b

COMPONENTS OF TRELLIS DIAGRAM
Figure 29

3. *Recursive MAP Estimation—The Viterbi Algorithm*

Normal communication deals with a continuous stream of data symbols. There is no finite sequence that returns to a known state periodically. This finite sequence approach would be a very inefficient coding method, particularly when it is well-known that very good performance is achieved by accepting common survivor branches several paths back from the current processing point in the recursive algorithm [30-32]. There is a very high probability that all paths pass through common branches from three to five constraint lengths back. The depth at which a final branch decision will be accepted as part of the optimum sequence estimation is called the truncation depth of the decoder.

In continuous operation, the Viterbi algorithm keeps a record of survivor paths back Δ branches for each current state in the trellis diagram. Δ is the truncation depth parameter. The decoder survivor selection function is denoted by $SUR^{\Delta}(\underline{x}_{r+1})$. The decoder examines all survivors and presents the optimum decoder output if all survivors have a common branch back Δ branches, or it indicates a decoding failure if there are any uncommon branches. This path selection process is visualized in Figure 30.

Any implementation of the Viterbi algorithm generally has three identifiable subassemblies that parallel the three aspects of the calculations and selection: branch calculations, state maximum selections and survivor path records. These three units are distinguished in Figure 31, which also characterizes their interconnections. The Branch Metric Unit (BMU) computes values relating the likelihood of a particular branch transition to the soft decision variables from the decoder as in equation (62). A transition is determined by present and next state nodes while the demodulator's variables are rough estimates of the symbols observed over the communication medium. The add-compare-select unit (ACSU) performs the recursive update according to equation (63) and passes information about survivor branches into each new state to the survivor memory unit (SMU). The feedback path around this unit indicates the recursive nature of the algorithm.

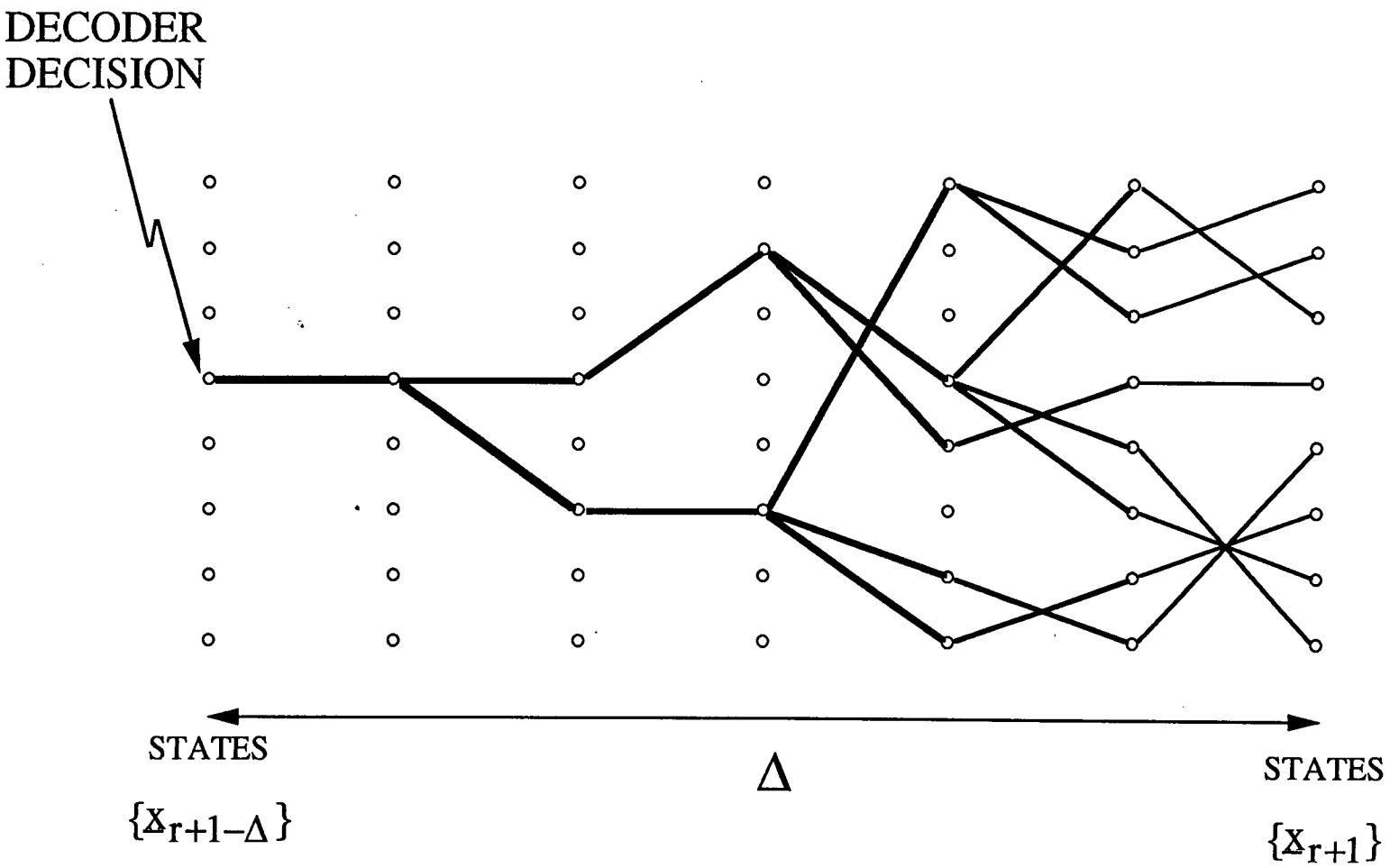
The SMU keeps current survivor paths for each state back as far as the truncation depth Δ and establishes the common branch decoder value for the decoder output.

There are many implementation options and details are contained in textbooks [30,31] and in the literature [33-35]. For example, the path metrics must be renormalized periodically to avoid numerical overflow. However, such details, while important in actual practice, are easily incorporated in the protection techniques discussed in the next sections. Many internally computed values are not observable from outside the decoder. The mapping from soft decision variables to branch metrics is nonlinear and the maximization decisions have a profound thresholding effect. The fault tolerance design challenges can be divided into two categories, depending on whether the features to be checked are observable externally or not.

The fundamental definition of the decoder as a MAP sequence estimator implies certain features of any realization leading naturally to an algorithm-based fault tolerance approach. On the other hand, parts of a decoder's units have internal characteristics that are not easily available externally. Furthermore, high-speed block realization of these types of decoders place special constraints on the internal variables. The next section addresses the external protection methods. The following section deals with decoders operating in a block processing fashion and shows how they can be protected primarily with internal features.

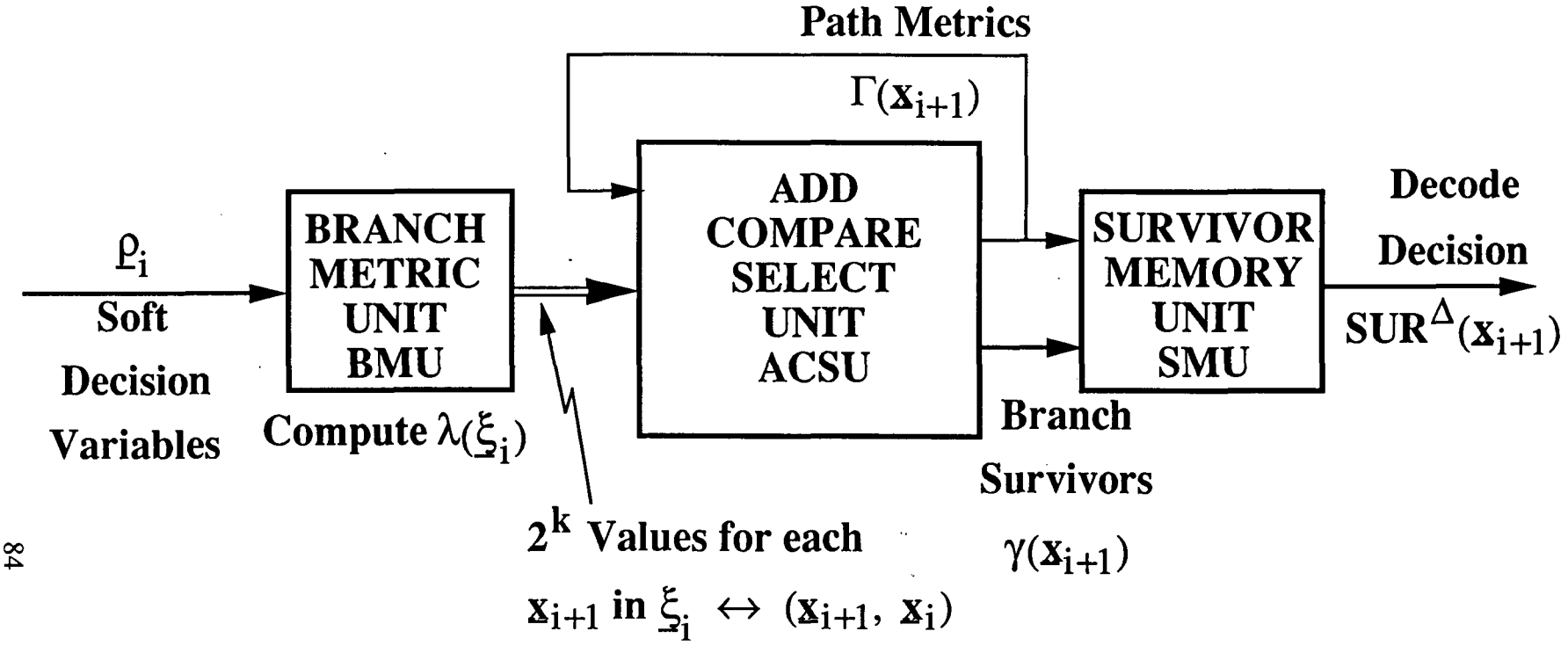
4. External Protection of Decoder Features

The MAP sequence estimation procedure selects one of a finite but large number of choices in producing the maximum value. The decoding algorithm partitions the finite dimensional vector space defined by the soft decision variables in $\underline{\rho}_i^{(L)}$. The region over which one set of transitions represents the proper decoder's choice generates path metric values that exceed metrics computed for all other regions given the subspace of soft



$SUR^{\Delta}(x_{r+1})$ is common state back Δ stages.

Truncation Depth in Trellis Diagram
Figure 30



ACSU records path metric for each state, sometimes normalizing;
SMU records Survivor paths for each state.

Viterbi Decoder Subassemblies
Figure 31

decision variables [29]. There is also a lower bound for each decoder region which is related to the concept of minimum free distance of the code.

The decoder can select an incorrect set of branches because the channel noise has increased statistically causing larger excursions in the symbol demodulator's decision variables so that they fall in different regions with higher probability. While this causes incorrectly decoded outputs, the decoder is still functioning properly and the resulting decoded errors are within the theory used to design such decoders. On the other hand, failures in the hardware implementing the decoding algorithm can produce the same effects. Any temporary change in the level of channel statistics will affect many adjacent channels similarly, producing a large number of incorrect decoder outputs with higher probability. This can be viewed as changing the variance of the variables emanating from the matched filters in all the demodulators, thus increasing the probability that the resulting soft decision variables will favor incorrect symbol levels.

The decoder output values may be re-encoded into an allegedly correct code sequence, corresponding with the information symbols produced by the decoder at truncation depth Δ. So if soft decision variables from the demodulator have been saved back to depth Δ, the successful common path can be used to recompute the path metric at this state. This recomputed path metric should be identical with the one originally computed by the decoder. This checking procedure requires that path metrics for survivor paths in the decoder must be saved back to the truncation depth Δ. However, the survivor paths going back from each state begin to converge to common paths several constraint lengths back, reducing these backward storage requirements drastically.

One method for protecting the decisions of a Viterbi decoder using the principles just described is outlined in Figure 32. There are two comparisons of the recomputed path metric for the selected decoded state at truncation depth Δ. One is with the value preserved in the decoder's survivor path memory unit, while the other is against a common lower bound developed from the statistical properties of the MAP estimation regions. This bound

could be made conservatively larger using the concept of the minimum distance of the code [31,32].

The Viterbi decoder exhibits other useful features at the truncation depth Δ primarily concerning the values of other path metrics impinging on the selected state on the common path. The path metrics on two successively chosen states can be reconstructed using the soft decision variables stored from the demodulator. Figure 33 depicts two necessary conditions occurring at these successive choices. It is known that the next choice on an optimal path is one of the survivor states from the presently chosen one. The path metrics are easily recomputed for all of these 2^k successor states. On the other hand, the selected next state must have the largest path metric coming from this previously decoded state, among all the 2^k precursor states that impinge upon it. The recomputation of these values needs (2^k-1) path metrics from these precursor states at this previous stage level. These values would have to be stored by the Viterbi decoder itself.

Another set of protection checks based on the observations above are shown in Figure 34. However, they represent validation of certain externally observable features, and there are many path metrics at unselected states that could be in error due to internal failures. Yet, these calculations provide checks on internally computed values that are critically important to the chosen path. Some failures in the BMU and ACSU are detected by these checks and in some instances they can indicate which units have failed. These protection procedures are very efficient to implement because they use relatively few calculations and only a small amount of extra storage is needed since this may be compressed as choices are made. Only precursors to states on common paths in the decoder's SMU are required. The simple protection schemes outlined here provide good coverage for maintaining the performance of the decoder, producing decoded output symbols within the design parameters of the code.

5. *Internal Protection, High-Speed Implementations*

The feedback loop around the decoder implementation shown in Figure 31, representing the recursive aspect of the Viterbi algorithm, limits the operating speed for this type of configuration [36]. Very recent work demonstrates new structures for high-speed implementations that ameliorate this limitation [36,37]. However, these new realizations introduce additional fault tolerance challenges. This subsection first outlines these new design approaches and then develops internal protection schemes which, while not as efficient as the simple external detection methods described earlier, fully protect the new structures.

There is an algebraic setting for expressing the Viterbi algorithm that leads to a natural decomposition of the recursive structure. Previous literature in combinatorial optimization and work on networks and graphs has examined the kind of maximization procedures encountered in the Viterbi algorithm [38-40]. The useful algebraic structure is a semiring [41,42] where both the additive and multiplicative semigroups are commutative. The underlying motivation for the two operators concerns addition and maximization. These new operators, multiplication, $\otimes$, and addition, $\oplus$, are defined over the real numbers with the respective identities also chosen for this set.

ADDITION $\oplus$	MULTIPLICATION $\otimes$	
IN REAL NUMBERS	IN REAL NUMBERS	
$\oplus \leftrightarrow \max(\,,\,)$	$\otimes \leftrightarrow +$	(64)
Identity	Identity	
$U \leftrightarrow -\infty$	$E \leftrightarrow 0$	

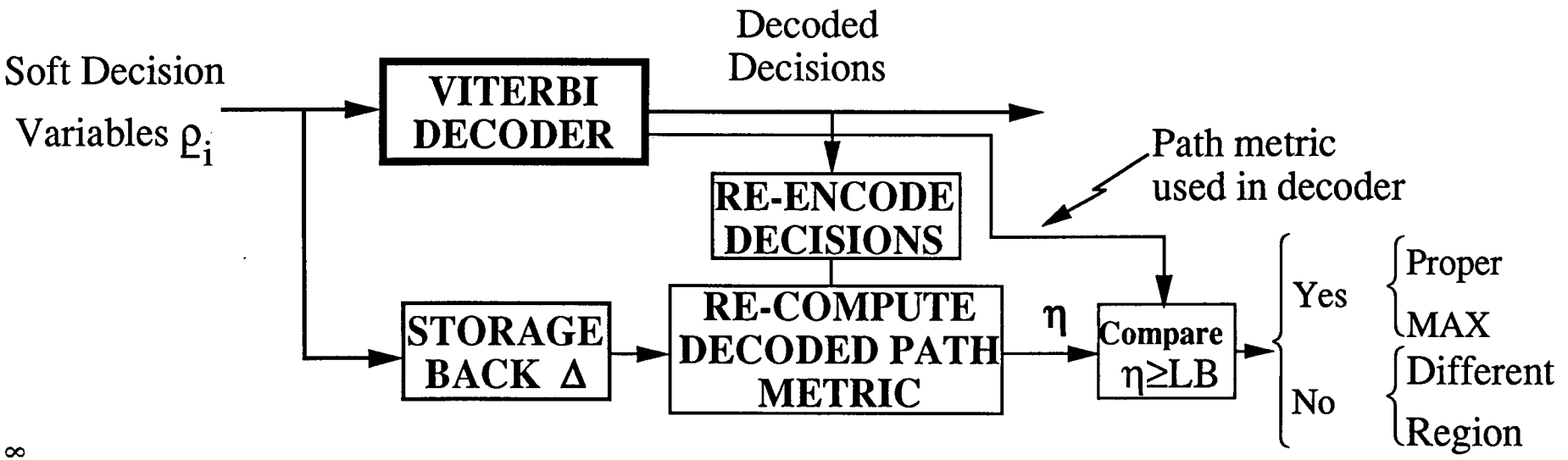
Based on these definitions, it follows that

$$U \otimes a = U \quad ; \quad \forall\, a \in \text{Reals} \tag{65}$$

The usual associative and distributive properties hold even though the existence of inverse elements is not guaranteed because the underlying structures are only semigroups.

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \qquad \text{ASSOCIATIVE} \tag{66a}$$

FAULT TOLERANCE CHECK

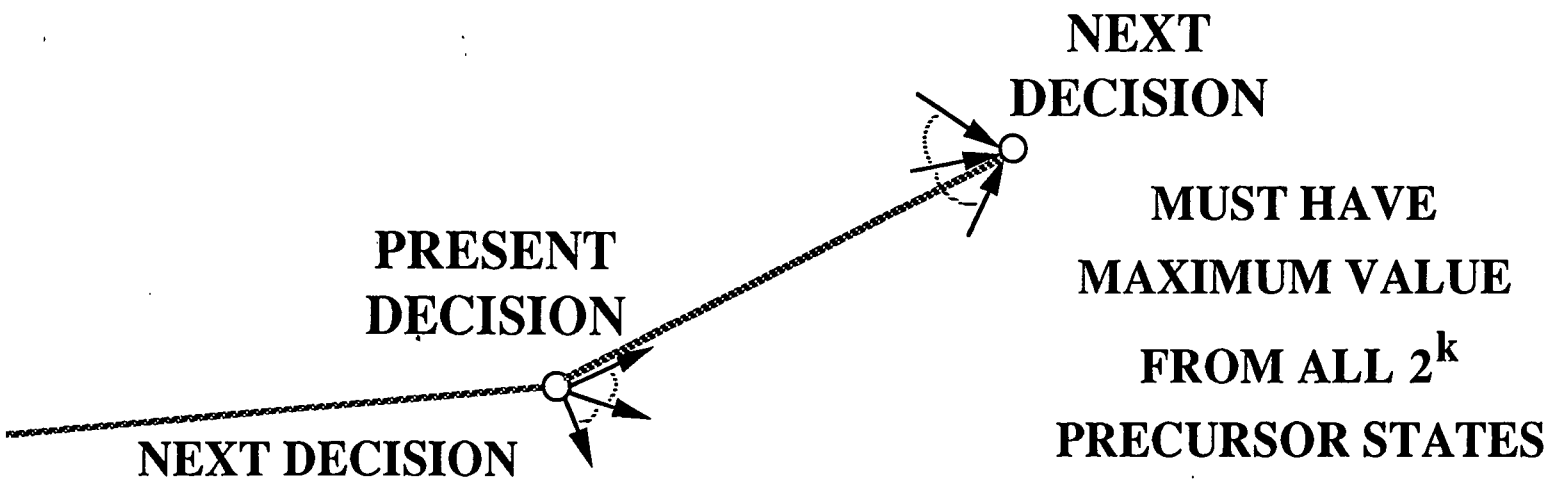


MAP Estimator has region where it is above a lower bound (LB)

A DECISION IN A DIFFERENT REGION INDICATES:

1.) NOISY CHANNEL
OR
2.) FAILED DECODER

Fault Tolerance Using External Features
Figure 32



Successor and Precursor States depend on certain groups of k bits:

$$\underline{\xi}_r = (\underline{x}_{r+1},\ \underline{x}_r)$$

$$= (\underbrace{\underline{u}_r}_{x_{r+1}},\ \overbrace{\underline{u}_{r-1},\ \cdots\ \cdots\ \underline{u}_{r-m+1},\ \underline{u}_{r-m}}^{x_r})$$

Two Necessary Conditions for
Decisions at Truncation Depth
Figure 33

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \text{LEFT DISTRIBUTIVE} \qquad (66b)$$

$$(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a) \qquad \text{RIGHT DISTRIBUTIVE} \qquad (66c)$$

The recursion in equation (63), the heart of the Viterbi algorithm, may be expressed in that algebraic setting. The maximization operation $\oplus$ is used to combine the 2^k precursor states to state $\underline{x}_{r+1}$, part of the legitimate choice for transition $\underline{\xi}_r$. Denote these precursor transitions as $\underline{\xi}_r^{((j))} = (x_{r+1}, x_r^{((j))})$; $j = 0, 1, \ldots, (2^k-1)$.

$$\Gamma(\underline{x}_{r+1}) = [\Gamma(\underline{x}_r^{((0))}) \otimes \lambda(\underline{\xi}_r^{((0))})] \oplus [\Gamma(\underline{x}_r^{((1))}) \otimes \lambda(\underline{\xi}_r^{((1))})]$$

$$\oplus \cdots \oplus \left[\Gamma\left(\underline{x}_r^{((2^k-1))}\right) \otimes \lambda\left(\underline{\xi}_r^{((2^k-1))}\right)\right] \qquad (67)$$

Of course, it is possible to append (or intersperse) the additive identity U a number of times to this expansion without affecting the value of the new path metric $\Gamma(\underline{x}_{r+1})$. This equation is reminiscent of the inner product of one vector containing branch metrics with another vector of path metrics for all previous states wherein U values are inserted for nonexistent transitions.

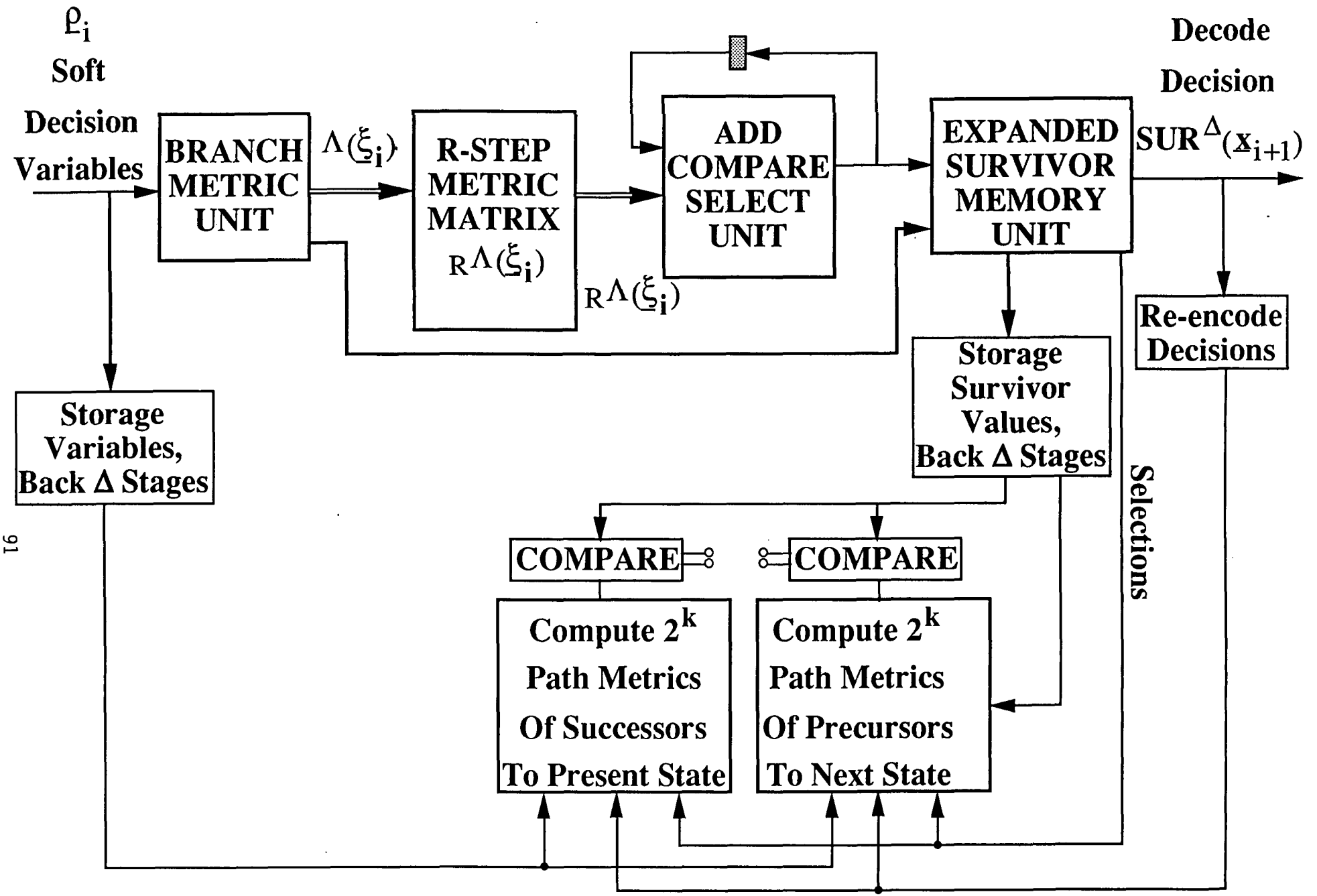
With this motivation in mind, it is possible to define vectors and matrices containing semiring elements along with associated operations based on the fundamental semiring operators $\otimes$ and $\oplus$. Matrix multiplication $\underset{\text{Mtrx}}{\otimes}$ and matrix addition $\underset{\text{Mtrx}}{\oplus}$ are established for appropriately sized rectangular arrays in a similar way to normal matrix operations [39,42].

$\underset{\text{Mtrx}}{\otimes}$ MATRIX MULTIPLICATION INVOLVES $\otimes$ FOLLOWED BY $\oplus$

$$\mathbf{A} \underset{\text{Mtrx}}{\otimes} \mathbf{B} = \mathbf{C} \quad \Leftrightarrow \quad \mathbf{c_{ij}} = \bigoplus_s \left[a_{is} \otimes b_{sj}\right] \quad \forall \mathbf{i, j} \qquad (68)$$

$\underset{\text{Mtrx}}{\oplus}$ MATRIX ADDITION INVOLVES COMPONENTWISE $\oplus$ OPERATION

$$\mathbf{A} \underset{\text{Mtrx}}{\oplus} \mathbf{B} = \mathbf{D} \quad \Leftrightarrow \quad \mathbf{d_{ij}} = a_{ij} \oplus b_{ij} \quad \forall \mathbf{i, j} \qquad (69)$$



Checking Necessary Conditions at Truncation Depth
Figure 34

Analogous operations exist for vectors since they are rectangular arrays of size 1 in one dimension. It will be convenient later when discussing parity generation to employ a componentwise multiplication of vectors. This operation, which has no counterpart in usual matrix theory, is denoted by $\underset{\text{Compnt}}{\otimes}$.

$\underset{\text{Compnt}}{\otimes}$ VECTOR COMPONENTWISE MULTIPLICATION

$$X \underset{\text{Compnt}}{\otimes} Y = Z \quad \Leftrightarrow \quad z_t = x_t \otimes y_t \qquad \forall t \tag{70}$$

The Viterbi algorithm may be expressed using vectors and matrices associated with the path and branch metrics. The basic array dimension is N, the number of states: $N = 2^{mk}$. The path metrics at states index r, $\{\underline{x}_r\}$, are contained in a vector $\underline{\Gamma}(\underline{x}_r)$, while the branch metrics ascribed to transitions are placed in a square array $\Lambda(\underline{\xi}_r)$. Any nonexistent transitions are given the $\oplus$ identity U in this matrix.

$\underline{\Gamma}(\underline{x}_r)$; $N \times 1$ VECTOR OF PATH METRIC VALUES AT STAGE r

$\Lambda(\underline{\xi}_r)$; $N \times N$ MATRIX OF BRANCH METRIC VALUES AT STAGE r
ELEMENT i,j IS $\lambda(\underline{\xi}_r^{(i,j)})$ WHERE $\underline{\xi}_r^{(i,j)} = (\underline{x}_{r+1}^{((i))}, \underline{x}_r^{((j))})$;
$i, j = 0, 1, \ldots, (N-1)$

Each matrix $\Lambda(\underline{\xi}_r)$ is fairly sparse since there are only 2^k nonidentity elements in each row or column because of finite memory span in the encoder. The Viterbi recursion (63) is written compactly using the matrix vector notation.

$$\underline{\Gamma}(\underline{x}_{r+1}) = \Lambda(\underline{\xi}_r) \underset{\text{Mtrx}}{\otimes} \underline{\Gamma}(\underline{x}_r) \tag{71}$$

A block processing form for the Viterbi algorithm can be developed because of the associativity and distributivity of the related semiring operations. As a first step, the next state path metrics may be expressed using path metrics for states two state indices removed.

$$\underline{\Gamma}(\underline{x}_{r+2}) = \Lambda(\underline{\xi}_{r+1}) \underset{\text{Mtrx}}{\otimes} [\Lambda(\underline{\xi}_r) \underset{\text{Mtrx}}{\otimes} \underline{\Gamma}(\underline{x}_r)]$$

$$= [\Lambda(\underline{\xi}_{r+1}) \underset{\text{Mtrx}}{\otimes} \Lambda(\underline{\xi}_r)] \underset{\text{Mtrx}}{\otimes} \underline{\Gamma}(\underline{x}_r) \tag{72}$$

The generalization to updating the path metrics in groups of R is straightforward.

$$\underline{\Gamma}(\underline{x}_{r+R}) = {}_R\Lambda(\underline{\xi}_r) \underset{\text{Mtrx}}{\otimes} \underline{\Gamma}(\underline{x}_r) \quad ; \quad \text{UPDATE STATE IN INCREMENTS OF R} \tag{73}$$

where

$${}_R\Lambda(\underline{\xi}_r) = \underbrace{\Lambda(\underline{\xi}_{r+R-1}) \underset{\text{Mtrx}}{\otimes} \Lambda(\underline{\xi}_{r+R-2}) \underset{\text{Mtrx}}{\otimes} \cdots \cdots \underset{\text{Mtrx}}{\otimes} \Lambda(\underline{\xi}_r)}_{\text{R BRANCH METRIC UPDATES}} \tag{74}$$

The individual branch metrics define each $\Lambda(\underline{\xi}_{r+j})$, j = 0, 1, ..., (R–1), in succession, and the R product represents the limited path metric maximums, starting from each state at state index r proceeding through to the states at state index (r+R). The combined matrix ${}_R\Lambda(\underline{\xi}_r)$ is called the R-step branch metric matrix.

The feedback computational requirements are lengthened in this viewpoint, an appealing feature since it eases the timing constraints imposed by needing the next state path metrics before the next calculation can begin. This block processing approach is shown in Figure 35, where a new subunit is inserted to compute the R-step branch metric matrix. The add-compare-select unit makes decisions for path metrics in steps of R. Since the survivor memory unit needs path metrics for each step to establish the surviving paths at each stage, the individual branch metrics are passed directly to it from the branch metric unit, a new path in the figure. The intervening path metric vectors are constructed directly from these and the R-step path metric vectors. This duplication of computational effort has one major advantage: these calculations are feedforward and can be performed independently in parallel just as the feedback path now permits R more index epochs allowing parallel paths. The potentials for high-speed realizations based on these type of decompositions are examined in a series of articles [34-37,43,44].

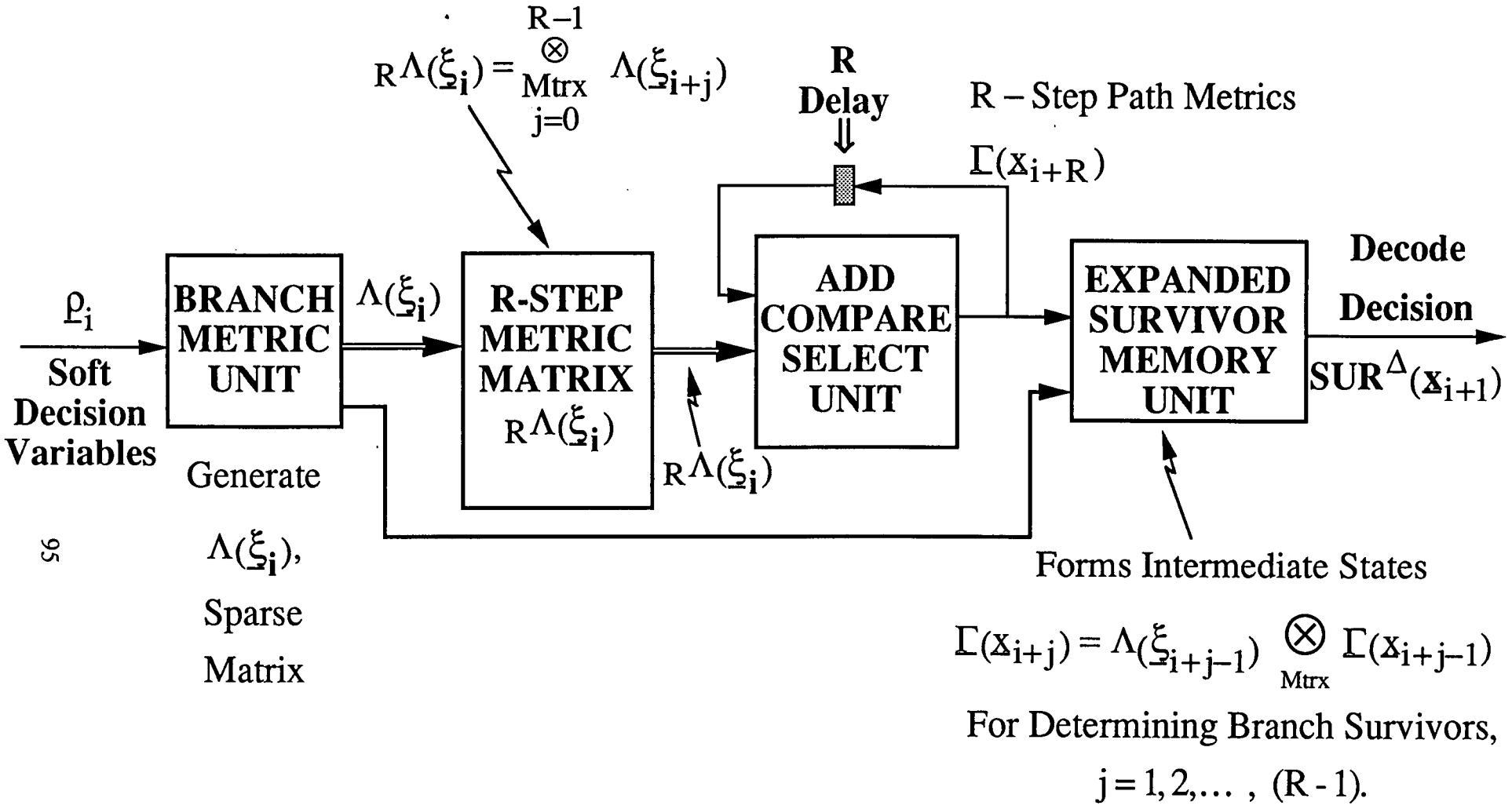
There are opportunities for checking the internal operations in such high-speed realizations of the Viterbi algorithm while at the same time there is a greater need to protect these expanded structures. All the external methods for protection described in earlier sections are assumed to be applied so that there are adequate checks on the survivor memory unit. Hence, the major uncovered failures concern path metric calculations and the maximization choices in the add-compare-select unit.

It should be noted that several fine points of standard implementations [30,31] have not been mentioned. However, they do not change the protection levels afforded by the techniques proposed. One typical example is that of the normalization of path metric values. As the path metric values develop, they grow larger in magnitude, possibly overflowing the finite word size available in computer storage elements. Practical implementations have control facilities for reducing all path metrics when one grows close to the upper limit for overflow [30,31]. However, any normalization actions can be signaled to the checking facilities in a feedforward fashion. If this action is erroneous, the checking system will not be able to produce similar results for comparison purposes, leading to mismatches.

The calculation of the path metric vectors spaced at intervals of R are checked efficiently employing a feature of the survivor memory unit. This unit computes intermediate path metrics independent of the block processing associated with the add-compare-select unit and its recursion. Only one additional matrix calculation is needed to determine a new path metric vector at step interval R from the end of the sequence of intermediate vectors $\underline{\Gamma}(\underline{x}_i), \underline{\Gamma}(\underline{x}_{i+1}), \ldots, \underline{\Gamma}(\underline{x}_{i+R-1})$.

$$\underline{\Gamma}(\underline{x}_{i+R}) = \Lambda(\underline{\xi}_{i+R-1})\, \underline{\Gamma}(\underline{x}_{i+R-1}) \tag{75}$$

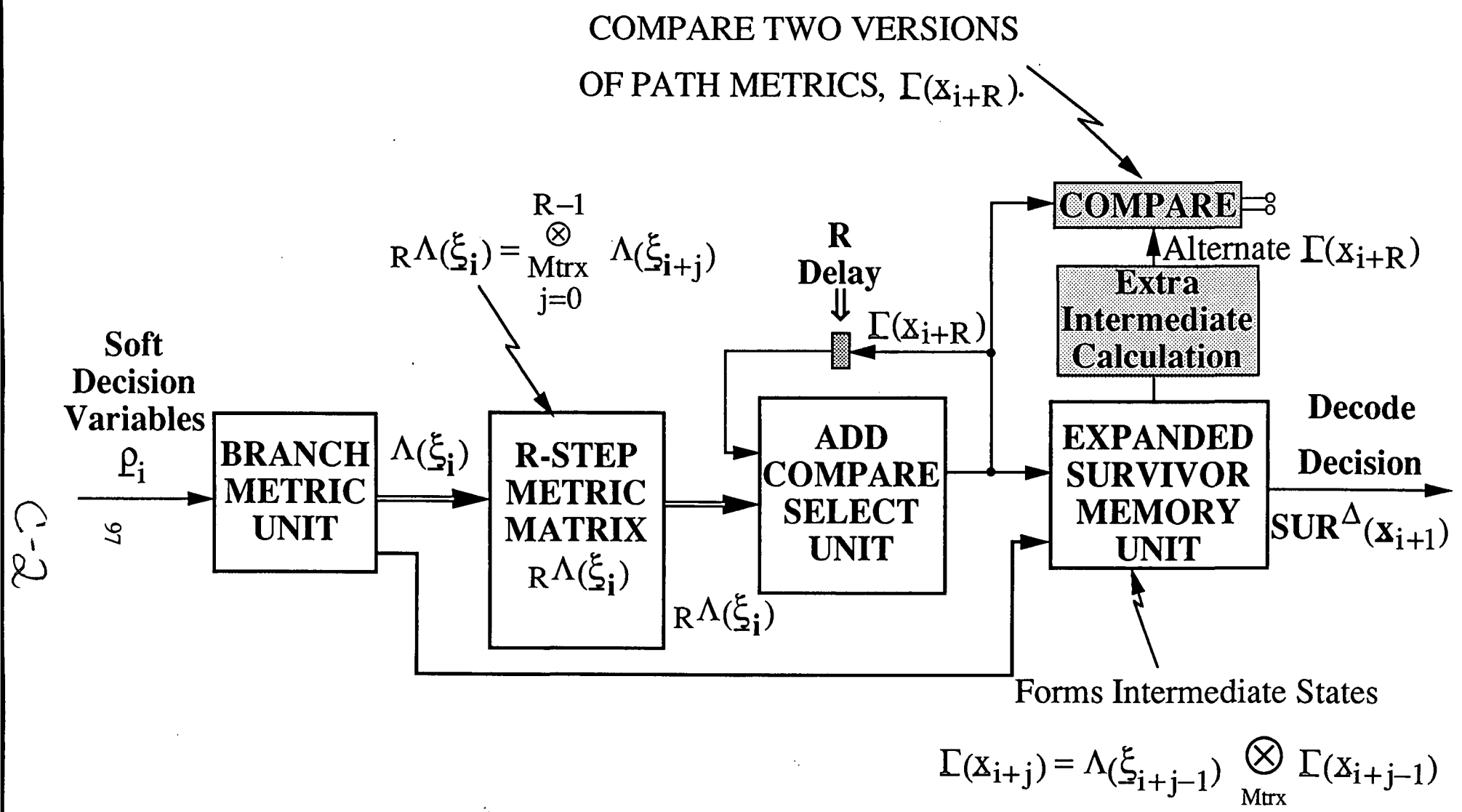
This gives an alternative calculation of $\underline{\Gamma}(\underline{x}_{i+R})$ which may be compared with the similar value emanating from the block processing step around the add-compare-select unit using R-step branch metric matrix ${}_R\Lambda(\underline{\xi}_i)$, equation (74). Such an internal check is shown in the



Block Processing Realization of Viterbi Algorithm
Figure 35

upper right part of Figure 36. Once this protection technique is in place, it is only necessary to check the block processing updates of the path metric vectors at indices pR, i.e., $\{\underline{\Gamma}(\underline{x}_{pR})\}$ multiples of index R. The protection of these vectors is addressed next.

The path metric vectors contain real number components representing parallel number channels, and therefore, may be protected efficiently by real convolutional codes, a familiar approach by now. A rate (k/k+1) binary-based real number code has several appealing features. The parity values related to each component of the vectors are produced infrequently, one parity vector for every k R-step path metric vectors. Since only R-step vectors are considered, each new group of k vectors $\underline{\Gamma}(\underline{x}_{pR}), \underline{\Gamma}(\underline{x}_{(p+1)R}), \ldots, \underline{\Gamma}(\underline{x}_{(p+k-1)R})$ is processed before a parity vector is calculated. Of course, the error-detecting capabilities of the code rely on memory in the parity generation process. The choice of binary-based codes eliminate scaling operations. However, there is a mixing of operations when protecting path metric vectors which are calculated based on semiring operations. The parity vectors are computed by summing path metric vectors with indices determined by the nonzero values in the code's parity filter transfer function Q(Z), equations (12) and (13). These real number summations correspond with the semiring multiplicative operation $\otimes$, equation (64). Therefore, the parity vector at index jR is equivalent to the vector componentwise multiplication of selected vectors as dictated by the nonzero positions in Q(Z). The path metric vectors at index multiples of R that fall in the code's encoding memory span and their respective parity weight locations are given below.



Checking Path Metrics in Block Processing Form of Viterbi Algorithm
Figure 36

Parity Filter Weights		State Vectors
$q_{(m+1)k-1}$	$\rightarrow$	$\underline{\Gamma}(\underline{x}_{(j-(m+1)k+1)R})$
$q_{(m+1)k-2}$	$\rightarrow$	$\underline{\Gamma}(\underline{x}_{(j-(m+1)k+2)R})$
$\vdots$		$\vdots$
q_k	$\rightarrow$	$\underline{\Gamma}(\underline{x}_{(j-k)R})$
q_{k-1}	$\rightarrow$	$\underline{\Gamma}(\underline{x}_{(j-k-1)R})$
$\vdots$		$\vdots$
q_1	$\rightarrow$	$\underline{\Gamma}(\underline{x}_{(j-1)R})$
q_0	$\rightarrow$	$\underline{\Gamma}(\underline{x}_{jR})$

$$\underline{P}_{jR} = \bigotimes_{\substack{\text{Compnt} \\ i \ni q_i = 1 \\ 0 \le i < (m+1)k}} \underline{\Gamma}(\underline{x}_{(j-i)R}) \qquad (76)$$

An algorithm based fault tolerance protection method applied to this situation involves computing comparable parity vectors in two ways. The first calculations use R-step path metric vectors directly from the operating Viterbi algorithm as in equation (76). The other parity vectors are computed in parallel employing branch metric matrices rederived from the soft decision variables from the demodulator. An additional refinement that leads to consolidation later may be introduced. The R-step path metric vectors may be computed using only previous path metric vectors at indices multiples of kR, the code parameter k times the step size R.

$$\underline{\Gamma}(\underline{x}_{(jk-s)R}) = \left[\bigotimes_{\substack{\text{Mtrx} \\ t=1}}^{(k-s)} {}_{R}\Lambda(\underline{\xi}_{[(j-1)k+t-1]R}) \right] \bigotimes_{\text{Mtrx}} \underline{\Gamma}(\underline{x}_{(j-1)kR}) \qquad (77)$$

$$s = 0, 1, \ldots, (k-1)$$

The R-step branch metric matrices are constructed from the individual branch metric matrices, in turn based directly on the soft decision variables from the demodulator.

$${}_{R}\Lambda(\underline{\xi}_{[(j-1)k+t-1]R}) = \bigotimes_{\substack{\text{Mtrx} \\ r=1}}^{(R-1)} \Lambda(\underline{\xi}_{[(j-1)k+t-1]R-r}) \qquad (78)$$

The direct computation of the parity vectors employs R-step branch matrices and R-step path metrics as determined by the decomposition of the indices of nonzero parity weighting coefficients $\{q_j\} \leftrightarrow Q(Z)$.

$$\underline{P}_{jR} = \bigotimes_{\substack{\text{Compnt} \\ s=0,1,K,m; \\ t=0,1,K,(k-1) \\ \text{such that} \\ q_{sk+t}=1}} \left[\left\{ \bigotimes_{\substack{\text{Mtrx} \\ p=1}}^{(k-t)} R\Lambda(\underline{\xi}_{[(j-1-s)k+p-1]R}) \right\} \bigotimes_{\text{Mtrx}} \underline{\Gamma}(\underline{x}_{(j-1-s)kR}) \right] \qquad (79)$$

INDICES s,t BOTH HERE

ONLY INDEX s HERE

However, the inner product of R-step branch matrices, $\bigotimes_{\substack{\text{Mtrx} \\ p=1}}^{(k-t)} R\Lambda(\underline{\xi}_{[(j-1-s)k+p-1]R})$, eventually expands out to include all k such R-step matrices which, in turn, allow the next kR indexed path metric vectors to be computed directly.

$$\underline{\Gamma}(\underline{x}_{(j-s)kR}) = \left[\bigotimes_{\substack{\text{Mtrx} \\ p=1}}^{k} R\Lambda(\underline{\xi}_{[(j-1-s)k+p-1]R}) \right] \bullet \underline{\Gamma}(\underline{x}_{(j-s-1)kR}) \qquad (80)$$

The two ways to generate parity vectors for protecting R-step path metric vectors are shown in Figure 37. The subunits are easily related to respective equations (76) – (80). The upper path to the comparator determines parity vectors in a straightforward way, equation (76), using path metric vectors from the operating Viterbi realization, Figure 36. The larger lower portion shows how the similar parity vectors are generated in parallel starting from soft decision values in $\{\underline{\rho}_i\}$. The ever increasingly larger products are computed by a running block update section that incorporates those R-step matrices that correspond to the s and t indices satisfying $q_{sk+t} = 1$. (See equation (79).) These matrices are used to form the parity vectors in the componentwise vector product unit, and simultaneously, the necessary groups of products are sent to the kR-step vector update unit. This latter subunit generates the kR-step path metric vectors for inclusion in the running componentwise vector product unit that produces the comparable parity vector.

The comparator checks for close agreement between respective components of the two differently generated parity vectors, allowing proper thresholds for tolerating roundoff errors as discussed earlier.

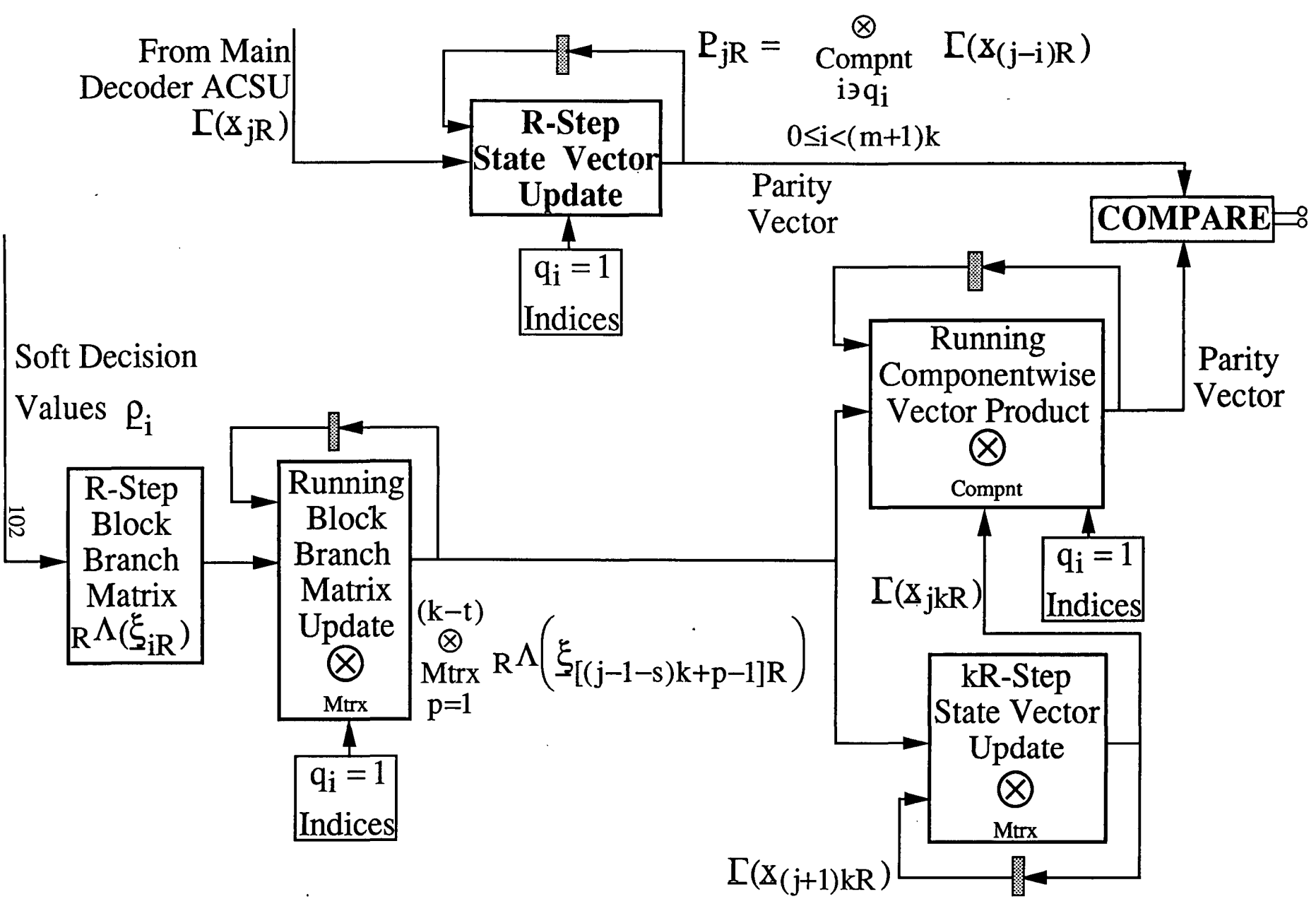
SUMMARY

Communications satellites will contain extensive high-speed data processing capabilities permitting the interconnection of very small aperture terminals (VSAT's). Fault tolerance design features for protecting the uplink processing resources, demultiplexing, demodulating and decoding have been presented. Algorithm-based fault tolerance techniques typify the fundamental protection methodology. However, any additional protection subassemblies must be compatible with functional units, allowing them to serve as replacements during any reconfiguration phase. These new generations of satellites require complete detection and switching of individual data user's channels in space in order to take advantage of spot beam antenna technology, where the limited orbit power is focused into narrow spots for downlink transmission. Complete data switching in space doubles the performance capabilities permitting the use of VSAT's. Individual users employ frequency division multiplexing (FDM) on the uplinks because of its simplified synchronization requirements whereas downlink retransmission uses time division multiplexing since system timing is visible to all VSAT's in the antenna's field of view.

The uplink processing resources offer a significant fault tolerance challenge because undetected failures can overwhelm the data retransmission system. The FDM channels are first separated into individually modulated streams by a highly efficient polyphase multirate demultiplexer when many processing elements are shared including the data passing through a fast Fourier transform (FFT) section. Symbols are extracted from the demultiplexed streams by demodulators, matched filters, which contain nonlinear feedback phase and timing tracking loops. Finally, a forward error-correcting decoder yields the original data for subsequent switching and retransmission. Severally these data are

encoded at the VSAT using a convolutional code and the effects of system noise are combated by Viterbi decoders in the satellite.

Novel fault-tolerant features have been developed within and around these three uplink resources. First, the multiple channels from the polyphase multirate demultiplexer are protected by algorithm-based fault tolerance (ABFT) techniques employing real convolutional codes to produce low rate parity samples in parallel with the normal demultiplexer. These parities are generated in a subsystem virtually identical with the original demultiplexer implementation, except operating at a very much lower rate. Comparable parity values are easily generated from the output channel streams using a finite impulse response filter with 0, 1 weightings, also operating at the same slower rate.



Generating State Parity Vectors in Two Ways
Figure 37

Comparisons of the two differently generated parity streams in threshold checkers provide detection capabilities. The parallel parity subassembly may be substituted into the original demultiplexer when reconfiguration is necessary.

The demodulators are not protected efficiently internally because of the timing and phase tracking loops. However, since the demodulated outputs contain redundancy because of the data convolutional code, if the demultiplexer before and the decoder following these units are protected, any errors appearing at the decoder outputs are due to either excessive uplink noise or failed demodulators. A "sandwich" protection principle prevails: If preceding and succeeding units in a data flow are fault-tolerant, any errors observed in the decoded outputs, not attributable to channel noise, indicate a failed intermediate unit.

The data convolutional code is decoded by a realization of the Viterbi algorithm which also contains internal and external redundancy features due to the code structure. The algorithm performs a maximum a posteriori sequence estimation recursively, selecting outputs corresponding to the highest values internal to the algorithm, unless the channel noise has exceeded designed levels of the code. This attribute is checked by reconstructing the algorithm's internal metrics directly from decoded decisions. Furthermore, the chosen metric should exceed a lower bound, an easily checked condition. The verification of the maximum path metric selections during algorithm operation is accomplished by recalculating these branch metrics from the decoded state sequence. The additional processing resources used for protection are very small and identical with subassemblies normally used in a Viterbi decoder, thus providing extra resources for use in reconfiguration.

REFERENCES

[1] M. Bellanger, *Digital Processing of Signals: Theory and Practice* (2nd Edition). New York: John Wiley & Sons, 1989.

[2] M. Bellanger and J. L. Daguet, "TDM-FDM Transmultiplexer: Digital Polyphase and FFT," *IEEE Transactions on Communications*, Vol. COM-22, pp. 1199-1205, 1974.

[3] M. Bellanger, G. Bonnert and M. Coudreuse, "Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-24, pp. 109-114, 1976.

[4] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*. Englewood Cliffs: Prentice-Hall, 1983.

[5] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs: Prentice-Hall, 1989.

[6] D. K. Pradhan, Editor, *Fault-Tolerant Computing Theory and Techniques,* Vol. 1., Englewood Cliffs: Prentice-Hall, 1986.

[7] J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*. New York: North-Holland, 1978.

[8] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*. Reading, MA: Addison-Wesley Publishing Company, 1989.

[9] K.-H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations." *IEEE Transactions on Computers*, Vol. C-33, pp. 518-528, 1984.

[10] J.-Y. Jou and J. A. Abraham, "Fault-Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent Computing Structures," *Proceedings of the IEEE (Special Issue on Fault Tolerance in VLSI)*, Vol. 74, pp. 732-741, 1986.

[11] J. A. Abraham, "Fault Tolerance Techniques for Highly Parallel Signal Processing Architectures," *SPIE Highly Parallel Signal Processing Architectures*, Vol. 614, pp. 49-65, 1986 (K. Bromley, Editor).

[12] C. J. Anfinson and F. T. Luk, "A Linear Algebraic Model of Algorithm-Based Fault Tolerance," *IEEE Transactions on Computers*, Vol. C-37, pp. 1599-1604, 1988.

[13] F. T. Luk and H. Park, "An Analysis of Algorithm-Based Fault Tolerance Techniques," *SPIE Advanced Algorithms and Architectures for Signal Processing*, Vol. 696, pp. 222-227, 1986.

[14] W. G. Bliss, "Area-Time Efficient and Fault-Tolerant VLSI Arrays for Digital Processing," Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Colorado, 1988.

[15] C. J. Anfinson, R. P. Brent and F. T. Luk, "A Theoretical Foundation for the Weighted Checksum Scheme," *SPIE Advanced Algorithms and Architectures for Signal Processing*, Vol. 975, pp. 10-18, 1988.

[16] R. P. Brent, F. T. Luk and C. J. Anfinson, "Choosing Small Weights for Multiple Error Detection," SPIE High Speed Computing, Vol. 1058, pp. 16-1–16-7, 1989.

[17] F. T. Luk, "Algorithm-Based Fault Tolerance for Parallel Matrix Equation Solvers," *SPIE Real-Time Signal Processing*, Vol. 564, pp. 49-53, 1985 (W. J. Miceli and K. Bromley, Editors).

[18] T. G. Marshall, Jr., "Coding of Real-Number Sequences for Error Correction: A Digital Signal Processing Problem," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-2, pp. 381-392, 1984.

[19] J. K. Wolf, "Redundancy, the Discrete Fourier Transform, and Impulse Noise Cancellation," *IEEE Transactions on Communications*, Vol. COM-31, pp. 458-461, 1983.

[20] T. G. Marshall, Jr., "Real Number Transform and Convolutional Codes," *Proceedings 24th Midwest Symposium on Circuits and Systems*, Albuquerque, NM, pp. 650-653, June 1981.

[21] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes* (Second Edition). Cambridge, MA: The MIT Press, 1972.

[22] S. Lin and D. J. Costello, Jr., *Error Control Coding Fundamentals and Applications*. Englewood Cliffs: Prentice-Hall, 1983.

[23] V. S. S. Nair and J. A. Abraham, "Real Number Codes for Fault-Tolerant Matrix Operations on Processor Arrays," *IEEE Transactions on Computers,* Vol. C-39, pp. 426-435, 1990.

[24] J. Hagenauer, "High Rate Convolutional Codes with Good Distance Profiles," *IEEE Transactions on Information Theory*, Vol. IT-23, pp. 615-618, 1977.

[25] G. R. Redinbo "Real Codes in the Fourier Domain for Fault-Tolerant Signal Processing." Chapter in *Spectral Techniques: Theory and Applications*. C. Springer-Verlag : Berlin, 1991. (Moraga and R. Creutzburg, Editors.)

[26] M. G. Bellanger, G. Bonnerot and M. Coudreuse, "Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. ASSP-24, pp. 109-114, 1976.

[27] W. D. Ivancic and M. J. Shalkhouser, "Destination Directed Packet Switch Architecture for a 30/20 GHz FDMA/TDM Geostationary Communication Satellite Network," NASA Lewis Research Center, Space Electronics Division, Cleveland, OH, August 1991.

[28] TRW, "Multichannel Demultiplexer Demodulator Project, Preliminary Design Review," TRW, One Space Park, Redondo Beach, CA 90278, November, 1990.

[29] H. L. Van Trees, *Detection, Estimation, and Modulation Theory, Part I*, New York: John Wiley & Sons, 1968.

[30] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, New York: McGraw-Hill, 1979.

[31] G. C. Clark, Jr. and J. B. Cain, *Error-Correction Coding for Digital Communications*, New York: Plenum Press, 1981.

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

[33] C. M. Rader, "Memory Management in a Viterbi Decoder," *IEEE Transactions on Communications*, Vol. COM-29, pp. 1399-1401, 1981.

[34] P. G. Gulak and T. Kailath, "Locally Connected VLSI Architectures for the Viterbi Algorithm," *IEEE Journal on Selected Areas in Communications*, Vol. 6, pp. 527-537, 1988.

[35] G. Fettweis and H. Meyr, "High-Rate Viterbi Processor: A Systolic Array Solution," *IEEE Journal on Selected Areas in Communications*, Vol. 8, pp. 1520-1534, 1990.

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

[37] G. Fettweis and H. Meyr, "High-Speed Parallel Viterbi Decoding: Algorithm and VLSI Architecture," *IEEE Communications Magazine*, Vol. 29, No. 5, pp. 46-55, 1991.

[38] B. Carré, *Graphs and Networks*. Oxford: Clarendon Press, 1979.

[39] U. Zimmermann, *Linear and Combinatorial Optimization in Ordered Algebraic Structures*. New York: North-Holland, 1981.

[40] R. A. Cunningham-Green, "Minimax Algebra," In: *Lecture Notes in Economic and Mathematical Systems*. Berlin: Springer-Verlag, Vol. 166, pp. 11-23, 1970.

[41] H. J. Zassenhaus, *The Theory of Groups* (Second Edition). New York: Chelsea Publishing Company, 1958.

[42] M. Yoeli, "A Note on the Generalization of Boolean Matrix Theory," *American Mathematical Monthly*, Vol. 68, pp. 552-557, 1961.

[43] G. Fettweis and H. Meyr, "A Modular Variable Speed Viterbi Decoding Implementation for High Data Rates," In: *Signal Processing IV: Theories and Applications* (Proceedings of Fourth European Signal Processing Conference), Ed. J. L. Lacoume, A. Chehikian, N. Martin and J. Malbos. Amsterdam: North-Holland, pp. 339-342, 1988.

[44] G. Fettweis and H. Meyr, "A Systolic Array Viterbi Processor for High Data Rates," In: *Systolic Array Processors* (International Conference on Systolic Arrays), Ed. J. McCanny, J. McWhirter and E. Swartzlander, Jr. New York: Prentice-Hall, pp. 195-204, 1989.

[45] L. Thiele and G. Fettweis, "Algorithm Transformations for Unlimited Parallelism," *Archiv für Elektronik und Übertragungstechnik (Electronics and Communications)*, Vol. 44, pp. 83-91, 1990.

[46] T.T. Ha, *Digital Satellite Communications* (Second Edition). New York: McGraw-Hill, 1990.

[47] W.H. Yim and F.P. Coakley, "Filter Banks with Rational Decimation and Interpolation Rates," *Electronics Letters*, Vol. 28, No. 8, pp. 726-727, 1992.

[48] W.H. Yim and F.P. Coakley, "Polyphase Matrix and Lattice Decomposition for Multirate Filters and Filter Banks," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 4, pp. 625-627, 1992.

[49] P.E. Beckmann and B.R. Musicus, "Fault-Tolerant Round-Robin A/D Converter System," *IEEE Transactions on Circuits and Systems*, Vol. 38, pp. 1420-1429, 1991.

APPENDIX A

POLYPHASE, MULTIRATE DECOMPOSITIONS OF INFINITE IMPULSE RESPONSE (IIR) FILTER STRUCTURES

This appendix describes how a channel filter with a rational Z transform transfer function may be separated to achieve parallel processing employing segmented filters similar to the popular polyphase multirate filter banks based on finite impulse response (FIR) filter structures. IIR filters can have very sharp bandlimiting characteristics and are therefore attractive candidates for baseband prototype filters. The extension of polyphase multirate filter banks for demultiplexing applications to those with IIR filters was first suggested by Bellanger [26] and is later described in his text [1]. The baseband transfer function, H(Z), will be decomposed in the Z transform domaining producing N parallel filters each operating at a rate reduced by N.

The rational transfer function H(Z) has coefficients related to the different equation description of the filter; the number of zeros, ν is assumed less than the number of poles, δ, to avoid any FIR part in the transfer function.

$$H(Z) = K\frac{\alpha_0+\alpha_1Z^{-1}+\ldots\ \ldots+\alpha_\nu Z^{-\nu}}{b_0+b_1Z^{-1}+\ldots\ \ldots+b_\delta Z^{-\delta}} \qquad ; b_0=1,\ \nu\le\delta \qquad \text{(A-1a)}$$

An alternate equivalent view involves pole and zero roots now considering the polynomial factors expressed in the indeterminant Z, as opposed to Z^{-1} as above. This requires a scaling constant, K′, containing a factor $Z^{s-\nu}$; repeated roots are allowed.

$$H(Z) = K'\frac{\prod_{i=1}^{\nu}(Z-\zeta_i)}{\prod_{j=i}^{\delta}(Z-\rho_j)} \qquad \text{(A-1b)}$$

A key rational function identity, that will be applied to each pole factor is given by

$$\left(Z^N-\rho_j^N\right) = (Z-\rho_j)\left[\sum_{\tau=0}^{N-1} Z^\tau\,\rho_j^{(N-1-\tau)}\right] \qquad \text{(A-2a)}$$

which may be written as a rational function too.

$$(Z - \rho_j) = \frac{\left(Z^N - \rho_j^N\right)}{\sum_{\tau=0}^{N-1} Z^\tau \rho_j^{(N-1-\tau)}} \qquad \text{(A-2b)}$$

This identity may be incorporated in the pole representation of H(Z), equation (A-1b), producing only powers of Z^N in the denominator

$$H(Z) = K' \frac{\prod_{i=1}^{\nu} (Z - \zeta_i)}{\prod_{j=1}^{\delta} (Z^N - \rho_j^N)} \left[\prod_{s-1}^{\delta} \left(\sum_{\tau=0}^{N-1} Z^\tau \rho_s^{N-1-\tau} \right) \right] \qquad \text{(A-3)}$$

The new denominator, being a function of Z^N terms only, will be identified by the polynomial $D(Z^N)$ to emphasize this point

$$D(Z^N) = \prod_{j=1}^{\delta} (Z^N - \rho_j^N) = A \sum_{i=0}^{\delta} d_i Z^{iN} \qquad \text{(A-4)}$$

; A,constant containing $Z^{-\delta N}$.

On the other hand, the numerator of equation (A-3) when expanded, may contain terms of degree up to $[\delta(N-1) + \nu]$, and when scaled by factor $Z^{-\delta N}$ can be written as a polynomial in indeterminant Z^{-1}.

$$N(Z) = Z^{-\delta N} K' \prod_{i=1}^{\nu} (Z - \zeta_i) \left[\prod_{s=1}^{\delta} \left(\sum_{\tau=0}^{N-1} Z^\tau \rho^{N-1-\tau} \right) \right]$$

$$= B \sum_{m=0}^{\mu} c_m Z^{-m}$$

$$; \quad \mu = [\delta(N-1) + \nu] \qquad \text{(A-5)}$$

This numerator polynomial may be separated into N pieces by taking subsequences of the coefficients $\{c_m\}$ using the Euclidean algorithm on index variable m.

$$m = \tau N + s \;;\; \tau = 0, 1, \ldots, \sigma \qquad s = 0, 1, \ldots, (N-1)$$

where $\sigma = \left\lfloor \frac{\mu}{N} \right\rfloor$, the greatest integer part.

$$N(Z) = \sum_{s=0}^{N-1} Z^{-s} \left\{ \sum_{\tau=0}^{\sigma} c_{\tau N+s} Z^{-\tau N} \right\} = \sum_{s=0}^{N-1} Z^{-s} N^{(s)}(Z^N) \qquad \text{(A-6)}$$

where the polynomial factors

$$N^{(s)}\left(Z^N\right) = \sum_{\tau=0}^{\sigma} c_{\tau N+s} Z^{-\tau N}.$$

A parallel decomposition results when equations (A-4) and (A-6) are combined. Each parallel IIR section, identified as $T^{(s)}(Z^N)$ below, operates at a reduced rate.

$$H(Z) = K'' \sum_{s=0}^{N-1} Z^{-s} \left\{ \frac{N^{(s)}(Z^N)}{D(Z^N)} \right\} = K'' \sum_{s=0}^{N-1} Z^{-s} T^{(s)}(Z^N) \qquad \text{(A-7a)}$$

where for convenience individual IIR sections have the notation

$$T^{(s)}(Z^N) = \frac{N^{(s)}(Z^{(N)})}{D(Z^N)}. \qquad \text{(A-7b)}$$

The implication of this decomposition is outlined in Figure A-1. As before, there is no savings in total operations, however, each section operates in parallel at a rate reduced by factor N. The significant savings occur when a uniform filter bank of size N is implemented. A development similar to the one in the text takes advantage of the shifting property coming from multiplying by a phasor. In particular, if the input of each filter with baseband response H(Z) is scaled by a phasor $\left\{ e^{j\frac{f_B}{N}\rho\tau} \right\}_{\tau=-\infty}^{+\infty}$ the Z transform of the overall filtering and scaling operation is $H(Z\, e^{-j\frac{f_B}{N}\rho})$.

$$H(Z) \rightarrow H\left(Z\, e^{-j\frac{f_B}{N}\rho}\right)$$

When incorporating the shifting due to phasor scaling in decomposition (A-7b), only the Z^{-s} terms are affected because of the Z^N functional dependence in all other parts.

$$H\left(Z\, e^{-j\frac{f_B}{N}\rho}\right) = K'' \sum_{s=0}^{N-1} Z^{-s}\, e^{+j\frac{f_B}{N}\rho s}\; T^{(s)}(Z^N) \qquad \text{(A-8)}$$

The resulting implementation resembles Figure 8, with $T^{(s)}(Z^N)$ functions substituted for the respectively indexed $H^{(s)}(Z)$ function there.

The effects of introducing N^{th} powers of the poles, appearing in the denominator polynomials $D\left(Z^N\right)$ is an issue. For poles very close to the unit circle (and inside this circle for stability reasons), using the N^{th} power has a positive impact from an implementation view [Sect. 10.7, 1]. Under these conditions, the effective poles move further from the unit circle producing a favorable influence on roundoff noise [Sect. 7.5, 1]. In order to see this, consider a pole with root ρ that is very close to the unit circle, within ε away for example.

$$\rho = e^{j\theta}(1-\varepsilon) \qquad \varepsilon > 0,\ \varepsilon \text{ very small.} \qquad \text{(A-9)}$$

The Nth power of ρ changes the angle and scales the distance from the circle

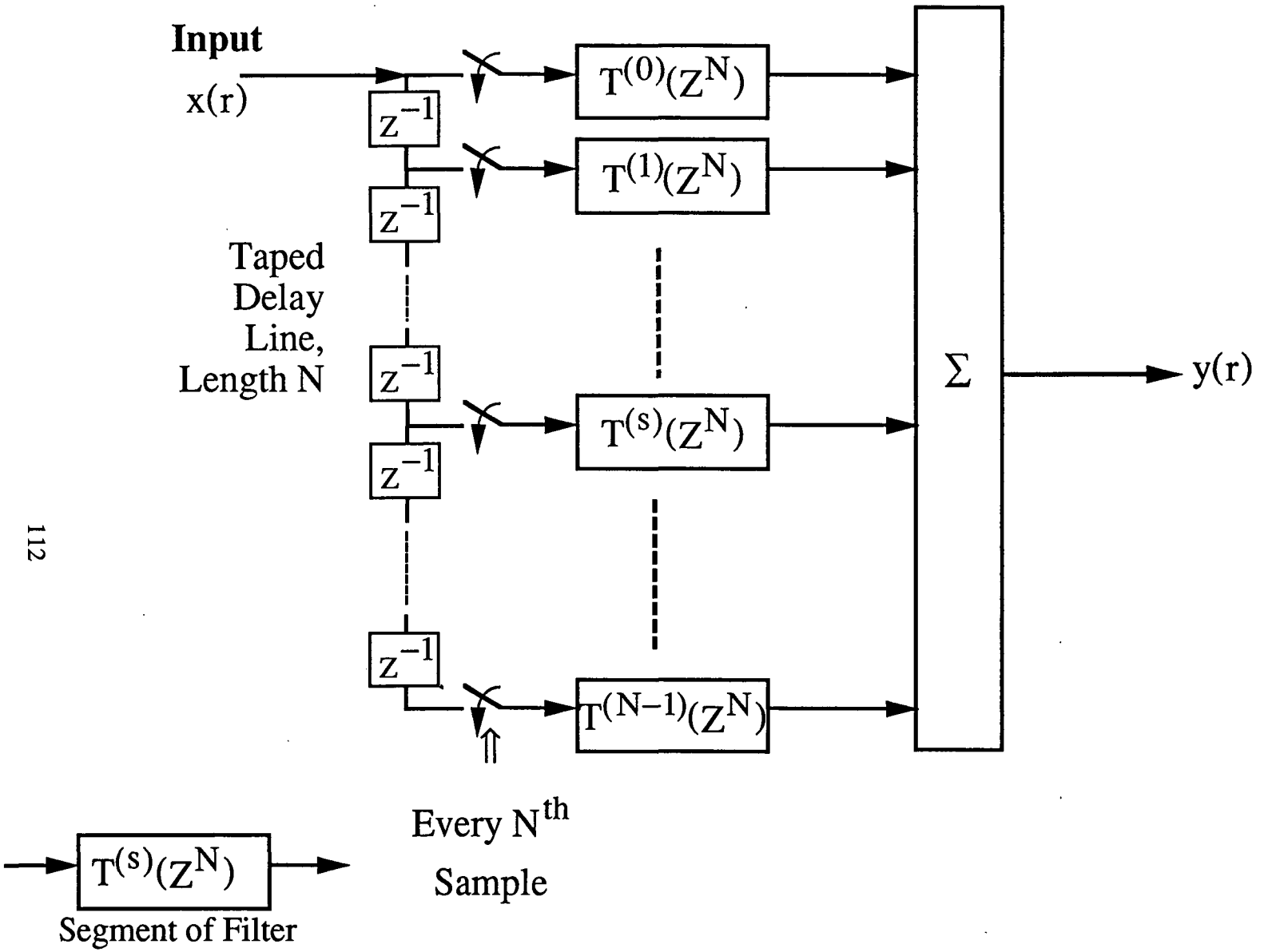
$$\rho^N = e^{jN\theta}(1-\varepsilon)^N \qquad \text{(A-10)}$$

The binomial expansion may be applied to the $(1-\varepsilon)^N$ factor which because of the smallness of ε can be accurately approximated as:

$$(1-\varepsilon)^N = \sum_{\alpha=0}^{N} (-\varepsilon)^\alpha \binom{N}{\alpha} \cong 1 - N\varepsilon \qquad \text{(A-11)}$$

$\binom{R}{\alpha}$ is binomial factor

It is instructive to examine the impulse response associated with the decomposition using the Z^N terms. This will demonstrate how the rational function decomposition produces a time sampling effect. The first step is to locate the roots of $D\ (Z^N)$, which could number up to δ^N. Each factor $\left(Z^N - \rho_j^N\right)$ may be viewed equivalently as



Decomposition of IIR Filter Structure
Figure A-1

$\rho_j^N\left(\left(\frac{Z}{\rho_j}\right)^N - 1\right)$, suggesting these roots resemble N^{th} roots of unity. It is easy to show that the roots are:

$$\xi_{j,q} = \rho_j W_N^q \qquad \begin{matrix} q = 0, 1, \ldots, N-1 \\ j = 1, 2, \ldots, \delta \\ W_N,\ N^{th} \text{ root of unity.} \end{matrix} \tag{A-12}$$

For a fixed index j, the N roots are distinct, as may be demonstrated

$$(\xi_{j,q} - \xi_{j,\rho}) = \rho_j (W_N^q - W_N^\rho) \neq 0 \qquad \text{if } q \neq p \tag{A-13}$$
$$q, \rho = 0, 1, 2, \ldots N-1$$

In order to gain insights without cumbersome notation for considering numerous special cases, assume that all roots are distinct; this restriction implies that the roots are not separated by an N^{th} root of unity.

$$\left(\rho_i / \rho_j\right) \neq W_N^k \qquad ; i \neq j,\ i, j = 0, 1, \ldots, \delta \tag{A-14}$$
$$k = 0, 1, \ldots, N-1$$

The next step is to perform a partial fraction expansion of H(Z) based on the roots of $D(Z^N)$ in the denominator [5]. From equation (A-3), using the distinct roots given in equations (12), this expansion involves determining coefficients $h_{j,t}$.

$$H(Z) = \frac{K'' N(Z)}{\prod_{j=1}^{\delta}\left(\prod_{t=0}^{N-1}(Z - \rho_j W_N^t)\right)} = K'' \sum_{j=1}^{\delta}\sum_{t=0}^{N-1}\left[\frac{h_{j,t}}{\left(Z - \rho_j W_N^t\right)}\right] \tag{A-15a}$$

$$h_{j,t} = \left.\frac{N(Z)}{\prod_{k=1}^{\delta}\prod_{m=0}^{N-1}(Z - \rho_k W_N^m)}\right|_{\substack{k \neq j \text{ and } m \neq t \\ Z = \rho_j W_N^t}} \tag{A-15b}$$

The numerator and denominator terms involved in the evaluation of the $h_{j,t}$ terms, equation (A-15b), will be examined separately.

$$\Lambda_{j,t} = \prod_{\substack{k=1 \\ k \neq j}}^{\delta} \prod_{\substack{m=0 \\ m \neq t}}^{N-1} \left(\rho_j W_N^t - \rho_k W_N^m\right)$$

$$= \rho_j^{\delta N-1} W_N^{-t} \prod_{\substack{k=1 \\ k \neq j}}^{\delta} \prod_{\substack{m=0 \\ m \neq t}}^{N-1} \left(1 - (\rho_k/\rho_j) W_N^{m-t}\right) \qquad \text{(A-16)}$$

$$= \rho_j^{\delta N-1} W_N^{-t} \left\{ \prod_{\substack{k=1 \\ k \neq j}}^{\delta} \prod_{q=0}^{N-1} \left(1 - (\rho_k/\rho_j) W_N^q\right) \right\} \left[\prod_{\tau=1}^{N-1} \left(1 - W_N^{\tau}\right) \right]$$

The last product term in the last line accounts for cases where $k = j$ but $m \neq t$ and is the result of a change of variables to $\tau = m - t$ with $m \neq t$. The only dependency on index t is the scaling factor W_N^{-t} in front. On the other hand, substitutions in numerator, employing its form given in equation (A-6), leads to a constant depending on the indices j and t of $h_{j,t}$.

$$N(Z) \Big|_{Z=\rho_j W_N^t} = \rho_j^{N\delta} \left\{ \sum_{s=0}^{N-1} \rho_j^{-s} W_N^{-st} \left[\sum_{\tau=0}^{\sigma} c_{\tau N+s} \rho_j^{-\tau N} \right] \right\} \qquad \text{(A-17)}$$

The partial fraction expansion may be written consolidating all terms dependent on index t into an inner sum.

$$H(Z) = K'' \sum_{j=1}^{\delta} \left\{ \frac{\sum_{s=0}^{N-1} \rho_j^{-(s-1)} \left[\sum_{m=0}^{\sigma} c_{mN+s} \rho^{-mN} \right] \sum_{t=0}^{N-1} \left(\frac{W^{-(s-1)t}}{Z - \rho_j W_N^t} \right)}{\prod_{\tau=0}^{N-1} (1 - W_N^{\tau}) \left[\prod_{\substack{k=1 \\ k \neq j}}^{\delta} \prod_{q=0}^{N-1} \left(1 - W_N^q (\rho_k/\rho_j)\right) \right]} \right\} \qquad \text{(A-18)}$$

Further grouping of all factors continuing indices s and j into terms labeled $\Gamma_{j,s}$ leads to a more compact form of the expansion.

$$H(Z) = \sum_{j=1}^{\delta} \sum_{s=0}^{N-1} \Gamma_{j,s} \sum_{t=0}^{N-1} \left(\frac{W_N^{-(s-t)t}}{Z - \rho_j W_N^t} \right) \qquad \text{(A-19)}$$

In the sample sequence domain, each transform domain term

$\left(\frac{W_N^{-(s-t)t}}{Z - \rho_j W_N^t} \right)$ corresponds to a causal sequence on index m.

$$\left(\frac{W_N^{-(s-t)t}}{Z - \rho_j W_N^t} \right) \quad \leftrightarrow \quad \left(\rho_j W_N^t\right)^{n-1} u_{-1}(n-1)$$

; $u_{-1}(m)$, unit step function

For fixed indices j and s, combining this inverse transform with the summation over index t introduces a "sifting" operation, modulo N.

$$\sum_{t=0}^{N-1} \left(\frac{W_N^{-(s-t)t}}{Z - \rho_j W_N^t} \right) \quad \leftrightarrow \quad \rho_j^{n-1} u_{-1}(n-1) \sum_{t=0}^{N-1} W_N^{[(n-1)-s+1]t} \qquad \text{(A-20)}$$

where $$\sum_{t=0}^{N-1} W_N^{[n-s]t} = \begin{cases} 1 & n \equiv s \mod N \\ 0 & n \neq s \mod N \end{cases}$$

The index s in equation (A-19) corresponds to the branch s in Figure A-1. Hence, noting equation (A-20), each branch effectively only produces a nonzero sample every N^{th} position viewed as sequence with index n. This becomes more visible by expressing the impulse response corresponding to H(Z) as:

$$H(Z) \leftrightarrow h(n) = \sum_{j=1}^{\delta} \sum_{s=0}^{N-1} \Gamma_{j,s} \begin{cases} N\rho_j^{(n-1)} u_{-1}(n-1) & s \equiv n \mod N \\ 0 & s \neq n \mod N \end{cases}$$

Thus, each branch indexed by s produces samples on every N^{th} index n, offset by the respective value of s.

APPENDIX B

MODIFYING REAL CONVOLUTIONAL CODES FOR POLE CANCELLATION IN IIR FILTER STRUCTURES

When a filter transfer function H(Z) contains poles, the cascade of H(Z) with a single parity weighting function, Q(Z), dictated by a real convolutional code can be simplified by modifying Q(Z) to cancel denominator factors in H(Z). Then the modified composite parity generation system becomes an FIR filter, greatly simpler to realize. This appendix presents the theory of how this can be accomplished by changing the weights represented in Q(Z), without altering the error-detecting capability of the code. For future reference the rational transfer function will be given.

$$H(Z)=\frac{N(Z)}{D(Z)}=\frac{\alpha_0+\alpha_1Z^{-1}+\alpha_2Z^{-2}+.....+\alpha_\nu Z^{-\nu}}{b_0+b_1Z^{-1}+b_2Z^{-2}+.....+b_\delta Z^{-\delta}} \quad ; \qquad b_0=1\ ,\ \nu\le\delta. \qquad \text{(B-1)}$$

Real codes may be scaled by real values in an effort to simplify the cascade of it with function H(Z). The most important characteristic of the code to be preserved is its systematic form. Hence, any row operations on its generator matrix must be confined to simple scaling so that there are no alterations to the data samples in the filtering operation. The semi-infinite generator matrix, equation (10), may be modified by scaling each respective row in the group of k rows associated with input data. The scaling values $\sigma_{k-1}, \sigma_{k-2},\ldots, \sigma_1, \sigma_0$ will be applied to form a new equivalent generator matrix G′, formally defined mathematically as:

$$G' = \Sigma G \qquad \text{where} \qquad \Sigma = \begin{pmatrix} S & 0 & & \\ 0 & S & & \\ & & \ddots & 0 \\ & & 0 & \ddots \\ & & & & S \end{pmatrix} \qquad \text{(B-2a)}$$

$$S = \text{diagonal}\,(\sigma_{k-1},\sigma_{k-2},\ldots,\sigma_1,\sigma_0)\ ;\ k\times k \ \text{ BLOCK DIAGONAL} \qquad \text{(B-2b)}$$

The semi-infinite block diagonal matrix Σ applies the k scalers to each row of G.

The basic coding effects of the generator matrix are completely described by the action of the code segment matrix $G^{(m)}$ that represents mapping $(m + 1)$ groups of k digits in a constraint length segment of input data onto n code digits.

$$G^{(m)} = \begin{pmatrix} 0 & P_m \\ 0 & P_{m-1} \\ 0 & | \\ | & | \\ | & | \\ 0 & P_1 \\ I_k & P_0 \end{pmatrix} \quad \begin{matrix} M \times n \\ \text{CODE SEGMENT MATRIX} \end{matrix} \qquad \text{(B-3)}$$

The rightmost $(n - k)$ columns of this matrix are the columns of Q, equation (12a) in the text, affecting the $(n - k)$ parity values. The modified code has a related code segment matrix $G'^{(m)}$ that contains the effects of scalers $\sigma_{k-1}, \sigma_{k-2}, \ldots, \sigma_1, \sigma_0$ as expressed in matrix S, equation (B-2b).

$$G'^{(m)} = \begin{pmatrix} 0 & SP_m \\ 0 & SP_{m-1} \\ 0 & | \\ | & | \\ | & | \\ 0 & SP_1 \\ SI_k & SP_0 \end{pmatrix} \quad \begin{matrix} M \times n \\ \text{MODIFIED CODE} \\ \text{SEGMENT MATRIX} \end{matrix} \qquad \text{(B-4)}$$

Hence, the new $(n - k)$ FIR parity channel filters are represented by Q'.

$$Q' = \begin{pmatrix} SP_m \\ SP_{m-1} \\ | \\ | \\ | \\ | \\ SP_2 \\ SP_1 \\ SP_0 \end{pmatrix} = (\underline{q'_0}, \underline{q'_1}, \underline{q'_2}, \ldots, \underline{q'_{n-k-1}}) \qquad \text{(B-5a)}$$

$$\underline{q'_c} = ((q'^{(j)}_c)) \quad ; \quad M \times 1 \quad \textit{Column Vector.}$$
$$j = 0, 1, 2, \ldots, M-1$$
$$c = 0, 1, 2, \ldots, (n-k-1) \quad . \qquad \text{(B-5b)}$$

$$Q'_c(Z) = \sum_{j=0}^{M-1} q'^{(M-1-j)}_c Z^{-j} \qquad \text{Z TRANSFORM CONVENTION}$$

The new parity filters represented by the transfer functions $\{Q'_c(Z)\}_{c=0}^{n-k-1}$ are governed by the scaling values in S. The goal is to simplify the parity generation process. If each of the new parity channel's transfer function contains $D(Z)$, the denominator of $H(Z)$, equation (B-1), the poles are effectively removed from the parity generation filters.

$$Q'_c(Z) = D(Z)R_c(Z)$$
$$\textit{Implies}$$
$$H(Z)Q'_c(Z) = R_c(Z)N(Z) \triangleq S_c(Z)$$
$$c = 0, 1, 2, \ldots, (n-k-1) \quad . \qquad \text{(B-6)}$$

The parity channels only need to implement the FIR filter described by the transfer functions $S_c(Z)$. In addition, the decimation operation $\downarrow k$ may be moved back through the FIR structures greatly reducing the parallel parity generation computational rate [1-4]. The scalings of the data samples as exemplified by the SI_k part of $G'^{(m)}$ are not actually performed for the data being processed by the composite filter $H(Z)$. The effects on

individual k samples scaled by the respective σ_i values may be included in the parity regeneration filters, the $Q'_c(Z)$ transfer functions in equation (B-6).

The basic theory and fundamental approach for guiding the desired code modification is described first for a single parity channel filter, a rate $(n-1)/n$ convolutional code. The extension to similar results for a general k/n rate real convolutional code is then discussed.

The new single parity transfer function $Q'(Z) = D(Z)\,R(Z)$ where $D(Z)$ is the denominator for the filter's transfer function and $R(Z)$ is another factor. This may be restated in a matrix equation where polynomial factor $R(Z)$ is represented by vector R and matrix B contains the effects of multiplying by the denominator $D(Z)$.

$$\begin{pmatrix} q'_{M-1} \\ q'_{M-2} \\ | \\ | \\ | \\ q'_1 \\ q'_0 \end{pmatrix} = Q' = BR \tag{B-7a}$$

$$R^T = (R_{M-\delta-1}, R_{M-\delta-2}, \ldots, R_1, R_0) \tag{B-7b}$$

$$B = \begin{pmatrix} b_\delta & 0 & 0 & 0 & 0 \\ b_{\delta-1} & b_\delta & | & | & | \\ | & b_{\delta-1} & — & 0 & 0 \\ b_2 & | & | & b_\delta & 0 \\ b_1 & b_2 & — & b_{\delta-1} & b_\delta \\ 1 & b_1 & — & — & b_{\delta-1} \\ 0 & 1 & — & — & | \\ 0 & 0 & — & — & — \\ — & — & — & b_2 & b_3 \\ | & | & | & b_1 & b_2 \\ — & — & 0 & 1 & b_1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{B-7c}$$

$M \times (M-\delta)$ MATRIX

On the other hand, the new desirable filter weights Q' are related to the original weights through the scaling effects in $(k \times k)$ matrix S.

$$Q' = \Xi Q \tag{B-8a}$$

$$\Xi = \begin{pmatrix} S0 & & 0 \\ 0S & & \\ & \ddots & \\ 0 & & 0 \\ & & 0S \end{pmatrix} \quad \begin{array}{l} (m+1) \text{ Blocks of } S \text{ on diagonal} \\ \Xi \text{ is } M \times M \end{array} \tag{B-8b}$$

The B matrix can be transformed to a particularly simple form by row operations only. These row operations are represented by matrix Γ.

$$\Gamma B = \begin{pmatrix} 0 \\ \cdots\cdots \\ I_{M-\delta} \end{pmatrix} \quad ; \quad \begin{array}{l} 0 \quad \delta \times (M-\delta) \text{ ZERO MATRIX} \\ \Gamma \; M \times M \text{ ROW OPERATIONS} \end{array} \tag{B-9}$$

Applying this row operation matrix to a combination of equations (B-7) and (B-8) exposes the unknown factors in vector R on the right, leaving the unknown scaling factors sandwiched between two known quantities on the left.

$$\Gamma \Xi Q = \begin{pmatrix} 0 \\ \cdots\cdots \\ I_{M-\delta} \end{pmatrix} R \tag{B-10}$$

However, the top δ rows on the right produce homogeneous equations involving the unknown scalers contained in matrix Ξ. The row operations matrix Γ is partitioned so that these homogeneous equations may be explicitly written.

$$\Gamma = \begin{pmatrix} T \\ \cdots \\ L \end{pmatrix} \quad ; \quad \begin{array}{l} T, \; \delta \times M \text{ TOP ROWS} \\ L, \; (M - \delta) \times M \text{ LOWER ROWS} \end{array} \tag{B-11a}$$

$$T \Xi Q = 0 \quad ; \quad 0 \text{ is } \delta \times 1 \text{ ZERO VECTOR} \tag{B-11b}$$

The k unknown scalers $\sigma_{k-1}, \sigma_{k-2}, \ldots, \sigma_1, \sigma_0$, repeated along the diagonal of Ξ are intertwined with the known entities in Γ and the original parity filter weights in Q. This equation may be consolidated by defining a new matrix U in terms of these known

quantities. The elements of T and the new consolidated matrix U are denoted by t_{ij} and u_{ij} respectively, where both indices start from 0 to be compatible with Z transform definitions.

$$\sigma U = 0 \;\; ; \;\; U \;\; k \times \delta \text{ CONSOLIDATED MATRIX} \qquad \text{(B-12a)}$$

HOMOGENEOUS CONSTRAINT

where $\sigma = (\sigma_{k-1}, \sigma_{k-2}, \ldots, \sigma_1, \sigma_0)$

and
$$u_{ij} = \sum_{\ell=0}^{M-1} t_{j\ell} q_{M-1-\ell} \quad ; \quad \begin{array}{l} j = 0, 1, \ldots, (\delta - 1) \\ i = 0, 1, \ldots, k-1 \\ \ell \text{ such that } \ell \equiv i \text{ MOD } k \end{array} \qquad \text{(B-12b)}$$

The solutions to homogeneous equation (B-12a) represent constraints among acceptable choices for the code modification scaling coefficients in vector σ. These components in σ must all be nonzero for proper code modification. Assume that $\delta < k$, i.e., the number of filter poles is less than the number of information positions in the code. Initially the consolidated matrix U will be assumed to have maximum row rank of δ. It will be noted later that if this rank is smaller, a larger number of solutions will be possible. Employing row permutations and column operations it is possible to bring U to the following form:

$$EUF = \begin{pmatrix} A \\ \cdots \\ I_\delta \end{pmatrix} \quad ; \quad \begin{array}{ll} A & (k-\delta) \text{ MATRIX} \\ E & k \times k \text{ ROW PERMUTATIONS} \\ F & \delta \times \delta \text{ COLUMN OPERATIONS} \end{array} \qquad \text{(B-13)}$$

The two operation matrices are nonsingular, and because E represents a permutation, its transpose is its inverse. The homogeneous equation (B-12a) may be recast where the inverse permutation of the components of σ lead to a vector λ.

$$\sigma E^T EUF = 0 \quad \Rightarrow \quad \lambda \begin{pmatrix} A \\ \cdots \\ I_\delta \end{pmatrix} = 0 \qquad \text{(B-14a)}$$

$$\lambda = \sigma E^T \quad ; \quad \text{Permutation of unknown scalers.} \qquad \text{(B-14b)}$$

The components of λ may be separated into independent and dependent parts with equation (B-14a) providing a relationship between these parts.

$$\lambda = (\lambda_I, \lambda_D) \tag{B-15a}$$

$$\lambda_I = (\lambda_0, \lambda_1, \ldots, \lambda_{k-\delta-1}) \;\; ; \;\; \lambda_D = (\lambda_{k-\delta}, \lambda_{k-\delta+1}, \ldots, \lambda_{k-1})$$

$$-\lambda_I A = \lambda_D \tag{B-15b}$$

The scaling choices in σ are only a permutation of the elements of λ.

$$\sigma = \lambda E \tag{B-15c}$$

In the parlance of dual spaces, a setting in which homogeneous equation solutions are often viewed [21, 22], the solution space is the annihilator subspace generated by the $(k - \delta)$ rows associated with the matrix A in the k-dimensional dual space. The additional requirement that σ contain only nonzero entries is not represented directly. (Neither is the practical property that the scalers be neither exceptionally large nor small.) Once the matrix A is determined, acceptable choices are quickly developed. Furthermore it is easy to show that if the row rank of U is less than δ, say ρ, the corresponding annihilator space has increased dimension of $k - \rho$. The solution process still proceeds in the same way with a larger subspace.

Once the scaling choices in σ have been made, all required to be nonzero, the matrix S, equation (B-2b), and its extension Ξ, equation (B-8b), are defined. The new parity channel filter weights are then fixed by equation (B-8a). The remaining unknown quantities in vector R, corresponding to the factor polynomial $R(Z)$ in equation (B-6), may be found from equations (B-10) and (B-11a).

$$LQ' = R \quad ; \quad L, \; (M - \delta) \times M \;\; \text{lower rows of } \Gamma. \tag{B-16}$$

A simple example employing one parity channel filter associated with a rate $\frac{6}{7}$ convolutional code will be outlined. A digital filter design having four poles and zeros with transfer function $H_{ex}(Z)$ is selected.

$$H_{ex}(Z) = \frac{0.001836(1 + z^{-1})^4}{(1 - 1.49237\ Z^{-1} + 0.85011\ Z^{-2})\ (1 - 1.56200\ Z^{-1} + 0.64780\ Z^{-2})} \tag{B-17}$$

Its poles lie within the unit circle and are listed as:

$$Poles = \left\{ \begin{array}{l} 0.78101 \pm j\,0.19457 \\ 0.74618 \pm j\,0.54159 \end{array} \right\}$$

A high-rate binary code was selected from a published list of rate $(n-1)/n$ convolutional codes [24]. This rate $\frac{6}{7}$ code with constraint parameter $m = 3$ has constraint length $M = 24$. The single parity channel has transfer function $Q(Z)$.

$$\begin{aligned} Q(Z) = {} & 1 + Z^{-1} + Z^{-2} + Z^{-3} + Z^{-4} + Z^{-5} + Z^{-7} + Z^{-8} + Z^{-9} + Z^{-10} \\ & + Z^{-12} + Z^{-15} + Z^{-16} + Z^{-17} + Z^{-18} + Z^{-20} + Z^{-22} \end{aligned} \quad \text{(B-18)}$$

A computer program was used to perform the various manipulations to determine the annihilator subspace from which all modification choices may be derived. The condensed matrix U for this case is given:

$$U = \begin{pmatrix} -3.691820 & 13.436189 & -17.729437 & 8.676580 \\ -3.659183 & 12.801716 & -14.161996 & 6.328707 \\ -3.485219 & 10.864257 & -12.434132 & 5.170306 \\ 1.031917 & -3.377532 & 3.748684 & -0.465057 \\ 0.806807 & -2.330170 & 2.232918 & -0.726626 \\ -2.147091 & 6.171027 & -7.224210 & 3.867608 \end{pmatrix} \quad \text{(B-19)}$$

The annihilator subspace is generated by the following rows presented in matrix form.

$$\left(A \vdots I_{k-\delta}\right) E = \begin{pmatrix} -0.021927 & -0.201627 & 1.000000 & 0.788640 & 0.000000 & -0.862873 \\ -0.157595 & 0.152349 & 0.000000 & -0.263590 & 1.000000 & 0.260440 \end{pmatrix} \quad \text{(B-20)}$$

The independent components of σ are σ_3 and σ_1 and their choices of 1.000000 lead to the following scaling coefficients:

$$\sigma = (-0.179522, -0.049279, 1.000000, 0.525091, 1.000000, -0.602433) \quad \text{(B-21)}$$

The modified parity channel filter weights are easily calculated. Note that one of the scaling values is relatively small. The interactive nature of the program allowed all choices to be explored. Changing the independent variables $\sigma_1 = 1.3$ and $\sigma_3 = -1.5$ yields scaling values of a more uniform size.

$$\sigma = (-0.171984, 0.500494, -1.500000, -1.525574, 1.300000, 1.632882) \quad \text{(B-22)}$$

The corresponding components of the other factor $R(Z)$, written in column vector form for this choice is given:

$$R = \begin{pmatrix} 0.000000 \\ 0.908833 \\ 3.787247 \\ 6.691858 \\ 6.591124 \\ 3.254148 \\ -2.343565 \\ -7.077143 \\ -9.836500 \\ -10.685218 \\ -11.125820 \\ -10.805311 \\ -9.068048 \\ -4.051392 \\ 3.718312 \\ 10.218611 \\ 13.079967 \\ 11.429131 \\ 6.287963 \\ 1.632882 \end{pmatrix} \quad \text{(B-23)}$$

The modification procedure can be extended to general rate $\frac{k}{n}$ convolutional codes. A condensed matrix $U^{(n-k)}$, $k \times (n-k)\delta$, can be formed using the original $(n-k)$ parity channel filters and the top δ rows of the row reduction matrix Γ, equation (B-20). In turn, this condensed matrix may be transformed to a convenient form by row permutations and column operations yielding the solution space for selecting the scaling weights in σ.

$$E'U^{n-k)}F' = \left(\frac{A}{I\delta}\right); \begin{matrix} A' & [k-{}^{a}n-k^{o}\delta] \times (n-k)\delta \text{ MATRIX} \\ E' & k \times k \quad \text{ROW PERMUTATION MATRIX} \\ F' & (n-k)\delta \times (n-k)\delta \text{ COLUMN OPERATION MATRIX} \end{matrix} \quad \text{(B-24)}$$

The annihilator subspace corresponding to the solution space is generated by the rows of the following matrix easily constructed from quantities determined above.

$$\left[\left(A' \mid I_{k-(n-k)\delta}\right)E'\right]$$

This solution space exists if $k > (n-k)\delta$.