VITURBO: A Reconfigurable Architecture for Ubiquitous Wireless Networks

Mani Bhadra Vaya

Thesis: Master of Science
Electrical and Computer Engineering
Rice University, Houston, Texas (August 2002)

RICE UNIVERSITY

VITURBO: A Reconfigurable Architecture for Ubiquitous Wireless Networks

by

Mani Bhadra Vaya

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Houston, Texas

August, 2002

ABSTRACT

VITURBO: A Reconfigurable Architecture for Ubiquitous Wireless Networks

by

Mani Bhadra Vaya

A run-time reconfigurable architecture for ubiquitous wireless networks has been designed and implemented. Reconfigurable architectures have the ability to change themselves dynamically thus presenting a viable proposition for handset design for ubiquitous networks, where a key requirement is the flexibility to switch across different standards in different environments, examples of which are Orthogonal Frequency Division Multiplexing based IEEE802.11a Wireless Local Area Networks (WLAN) and Code Division Multiple Access based 3^{rd} Generation (3G) cellular networks. Channel encoding and decoding are essential components of these communication systems and different forms of convolutional encoders and decoders are used. We present the design and implementation of a novel reconfigurable architecture that can decode a range of convolutionally coded data (constraint lengths 3-9); and Turbo coded data (constraint length 4). Our architecture can support channel decoding for most of the current communication systems like WLAN, 3G, and Global System for Mobile Communications.

To my dearest Minnu

Acknowledgments

This work is a dedication to my dearest friend Minnu who is no more between us, but whose memories will always live on. Her loss is immeasurable, but the strength I derive from her love keeps me going.

No words would be enough to thank my parents and siblings who have always stood by me. I am also very grateful to Dr. Lindley Doran at Rice Counselling Center who helped me cope with the toughest of times when I was simultaneously dealing with the loss of my loved one and the tight schedules for this work.

This work would not be complete without thanking my advisor Dr. Joe Cavallaro, whose support and guidance made it possible. Dr. Aazhang's constant encouragement was very valuable and I thank him for that. My sincere thanks go out to Dr. Varman for his help and advice.

Contents

Illustrations

Tables

Chapter 1

Introduction

1.1 Motivation

Ubiquitous wireless networks, nascent as they are, would in the future be able to provide seamless wireless communications, with no strings attached [5]. Users would be able to access the network at almost any place and at any time. Different communication systems would be needed to support communications in outdoor and indoor environments, and these would have to be integrated strongly, in order to provide a seamless network. OFDM based systems like WLAN provide high speed local area network access up to 54 Mbps. 3^{rd} generation cellular communication systems, based on CDMA provide outdoor network access at data rates up to 2 Mbps. A receiver architecture with the ability to seamlessly transfer between such high speed local area networks and outdoor cellular networks would provide the backbone for the realization of such ubiquitous networks. Numerous issues on different layers of networks would have to be dealt with, in order to realize such an architecture. This thesis deals with issues in design of physical layer architectures for receiver structures of such ubiquitous networks. A simple solution would be to have architectures for all possible standards, and to switch between different architectures, as and when required. Though simple in conception, such a receiver would be extremely bulky, and would hardly be classified as handset. A more elegant solution, as proposed in this thesis, is the design of reconfigurable architectures, which can provide the flexibility to switch across standards with the same piece of hardware, hence saving area and cost.

We have narrowed down the problem to design of reconfigurable architectures for baseband processing, and more specifically, forward error correction techniques in various standards. Forward error correction techniques, though computationally some of the most intensive, are a necessary component of wireless communication systems as seen today. 3G Cellular systems, WLAN systems, GSM, General Packet Radio Service (GPRS), and many other systems have accepted various forms of convolutional coding as key components in their respective error correction schemes. Such a study would give a greater insight into the operation of different standards, and how reconfigurable architectures could prove beneficial.

1.2 Contributions

A high speed reconfigurable channel decoding architecture for multiple standards is the core contribution of this work. The architecture is capable of Turbo and Viterbi decoding for a wide range of code parameters, varying from constraint lengths 3 to 9 and rate 1/2 and 1/3, and any generator polynomial. The architecture has been designed and implemented on a Field Programmable Gate array (FPGA).

Earlier work as proposed by Bickerstaff et. al. [8], was limited to unified Viterbi and Turbo decoding for 3^{rd} generation cellular systems (3G), where data rates of the order of 2 Mbps are required. In the work proposed by Chadha et. al. [13], hard input reconfigurable Viterbi decoding using Hamming distances for WLAN and 3G was proposed, with 26 Mbps as the achievable data rates. However, the proposed architecture can achieve and beat the data rates recommended in WLAN and 3G standards.

Our proposed architecture can achieve data rates up to 60 Mbps for soft input, Euclidean distance based Viterbi decoding for both WLAN and 3G, while up to 3.5 Mbps is achievable for Turbo decoding for 3G. The area consumed for this architecture is much smaller than the combined area of the three different decoders which have been proposed in WLAN and 3G. This implies a significant saving in area as well as cost.

The proposed architecture uses a completely parallel decoding for all the possible constraint lengths. Smaller constraint length (< 9) decoders use parts of the constraint length 9 decoder, and the data routing in the entrails is realized through multiplexer banks.

Architectural power saving schemes have been employed in order to shut down parts of the circuit that may not be needed for a particular decoding type. This ensures that while maintaining its flexibility, the architecture does not compromise on the power consumption, unlike some earlier works [13].

This unique blend of flexibility achieved, area and power consumed, and throughput delivered make this architecture a powerful proposition for future communication devices.

1.3 Organization

Following is a brief overview and organization of the thesis. Chapter 2 details the physical layers of OFDM and CDMA systems and the different channel encoding and decoding algorithms used in these systems. Chapter 3 has been used to elucidate the architectures for Viterbi and Turbo decoding. The complete decoder architectures are broken down into smaller sub units and analyzed. The crux of our contribution is expounded in Chapter 4. In chapter 5, we present the design methodology used to realize this architecture. We also present the results of simulations, and analyze them critically. The thesis is concluded in Chapter 6 and future work forms chapter 7 .

Chapter 2

Communication Systems and Error Correction Techniques

A wireless communication system, like many other communication systems comprises of various layers of network. Physical layer computations for wireless communication systems are extremely expensive, and architectures for such computations have been the subject of intensive research over the past few decades. In this chapter we shall study the physical layer processing of OFDM and CDMA systems. We shall also gain an understanding of the different channel encoding and decoding algorithms used in these systems. This shall enable us to understand the need for flexible processors.

2.1 Code Division Multiple Access for Cellular Systems

The precipitous growth of cellular communication systems can be traced back to early 1980's, when the first generation of cellular systems, using analog transmissions were brought to life. In the late 1980's, the second generation cellular systems started taking form. These digital systems offered higher spectrum efficiency and better data services than their predecessor system. Some examples of these systems are GSM, a Time Division Multiple Access (TDMA) based system, and IS-95, a CDMA based system.

CDMA is the protocol for the emerging 3^{rd} generation cellular systems, and promises to deliver high data rates, multimedia communications, multi-rate services, and Quality of Service in the wireless framework. In CDMA, access is provided to different users by assigning a signature waveform to each user. A spreading sequence consists of a combination of +1/-1, which is specific to the given user and has some unique correlation properties such that the spreading sequence of different users are orthogonal to each other.

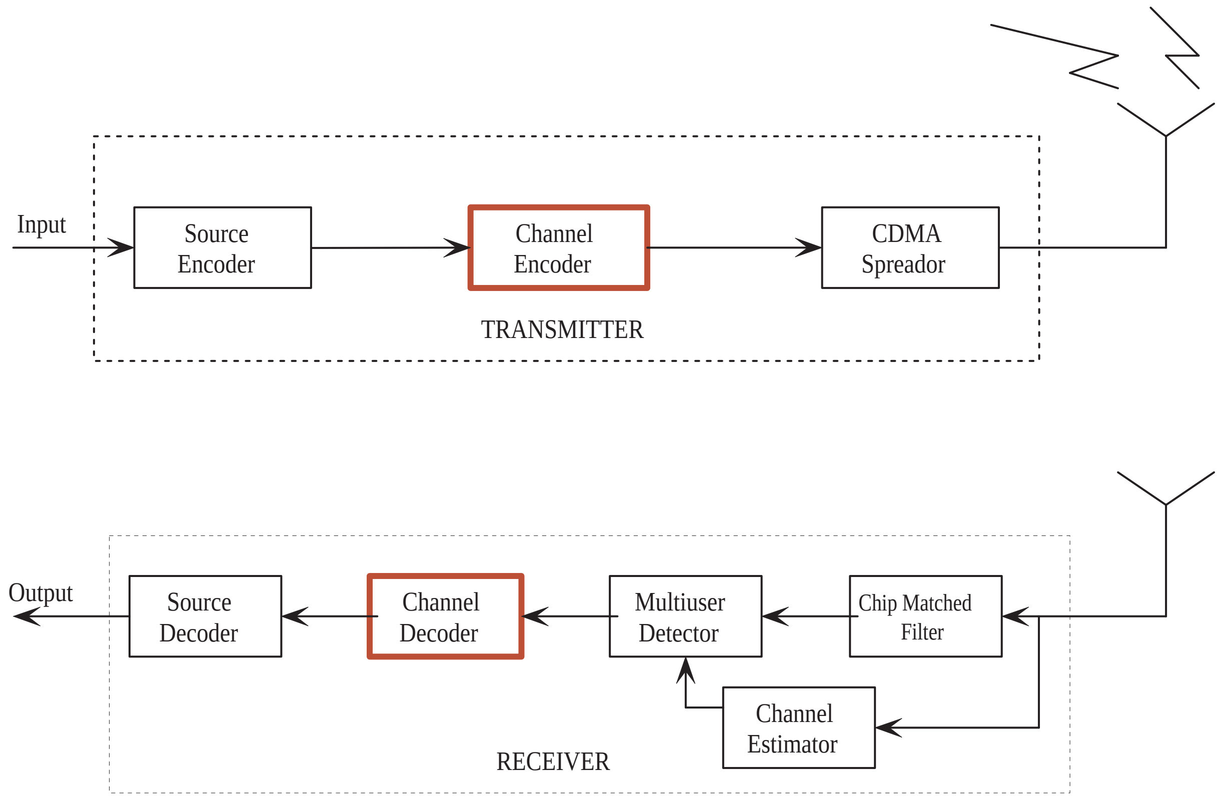


Figure 2.1 : CDMA System physical layer

A typical CDMA transmitter and receiver system is shown in Figure 2.1. At the transmission end, the data is first source encoded in order to compress it, and remove redundancies. In the next step the data is *channel encoded*, where deliberate redundancy is introduced in order to provide robustness again channel. Voice is encoded using convolutional codes, because of the streaming nature of these codes (section 2.3.1), while data is encoded using Turbo codes. The channel encoded data is then passed through a spreading mechanism as described earlier. Upon spreading, the chips so obtained from the information bits are modulated by RF carrier using a digital modulation technique, and transmitted across the channel.

The channel adds noise to the signal, and introduces various effects like attenuation, multi-path, and fading. This degraded signal is processed by the receiver, which endeavors to restore the original data bits. The baseband signal is extracted from the radio signal using the demodulator, which is sent to the chip matched filter. The continuous time signal from the channel is converted to discrete time signal by sampling the output of a matched filter to the chip waveform. Channel estimation and detection make use of the pilot bits in order to estimate the channel and detect the data bits. Once detected, the data is passed through a *channel decoder*, which could either be a Viterbi decoder for convolutionally encoded data or Turbo decoder for Turbo encoded data. Finally source decoding un-compresses the bits, and outputs the original bit stream.

2.2 Orthogonal Frequency Division Multiplexing for WLAN

The very basic principle of OFDM system is to split a high rate datastream into a number of lower rate datastreams that are transmitted simultaneously over a number of subcarriers, each of a different frequency. The null of the frequency of one sub carrier is at the peak of an adjacent frequency, hence giving the prefix *orthogonal*. Even though this results in overlap of carriers, it also ensures that the signals at different frequencies are received at the receiver without mutual interference.

IFFT/FFT is used to digitally modulate /demodulate the data over a large number of carriers. Figure 2.2 shows the various components of an OFDM transmitter and receiver system. At the transmitter side, the data is channel encoded to introduce

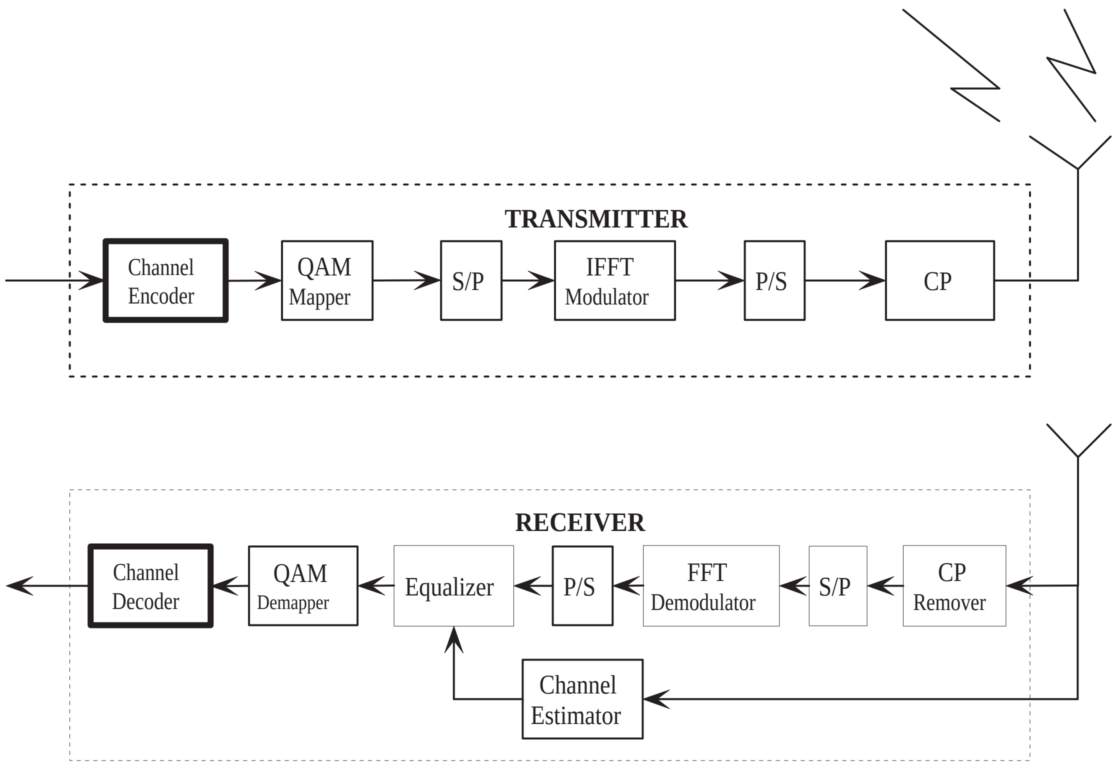


Figure 2.2 : OFDM System physical layer

redundancies and increase robustness. The data is then passed through an interleaver, where it is decorrelated, and spread over many tones. PSK (Phase Shift Keyed) or QAM (Quadrature Amplitude Modulation) based modulators are used to modulate the data which is then converted from Serial to Parallel form to be fed into IFFT. In this way, complex frequency domain data is transformed into time domain. To reduce intersymbol interference, the time varying data is cyclically extended with cyclic prefix. This data is ready for transmission by the R/F transmitter.

At the receiver, the cyclic prefix is removed and data is converted from serial to parallel in order to be processed through a FFT demodulator. In an OFDM symbol, training symbols are embedded in order to help in the estimation of the channel transfer function. The channel transfer function is obtained by interpolation between coefficient values obtained upon the use of training symbols. For the frequencies where channel transfer function was relatively small, noise may be amplified upon division by a small value, and hence render the equalization scheme useless. However, since the data was channel encoded we can still retrieve the symbols that experienced a bad channel with the help of those that encountered a relatively better channel. The equalizer is followed by channel decoder, which is the Viterbi decoder for convolutional codes. The output of channel decoder is the original data stream.

As seen from the discussion above, channel encoding and decoding form an important part of both the communication systems. While Viterbi and Turbo decoder are needed for 3rd generation CDMA systems, only Viterbi decoding is needed for OFDM based WLAN systems. In the following sections we shall give a detailed exposition of the channel encoders and decoders mentioned above.

2.3 Channel Encoding Algorithms

In this section we shall give a brief overview of the channel encoding schemes used in the systems discussed above: simple convolutional coding, which was proposed by Elias in 1955 [10] and parallely concatenated convolutional coding (Turbo coding), proposed recently by Berrou [1] in 1993.

2.3.1 Convolutional Encoding

Convolutional codes are generated when the information sequence to be encoded is transmitted through a linear finite-state shift register. $K, k - bit$ stages and n linear algebraic function generators constitute such a linear shift register [14]. The input data is shifted along the registers at a rate of k bits at a time, and the number of output bits for each $k - bit$ input sequence is n bits. This predicates a code rate of k/n, where code rate is defined as: *number of input bits/ number of output bits.* The constraint length of such an encoder is defined as the number of memory elements in the shift register plus one, which is K.

The state of a given encoder is defined by the values of the binary data in the shift registers. The number of states possible for a given convolutional encoder is contingent upon the number of permutations possible for the binary values in the shift registers, which is obviously dependent upon the number of shift registers. As discussed earlier, for the case of constraint length K, there are $K - 1$ shift registers, and hence the number of possible permutations is 2^{K-1}, which gives the number of possible states. For the case of K=3, the number of possible states is hence $2^{3-1} = 4$. It may be noted here that not only do the number of states increase exponentially with the constraint length K, the complexity of the decoder also increases exponentially with K. The error protection capability of convolutional codes increases with constraint length and the inverse of code rate.

The generator polynomials correspond to the shift register connections to the function generator, where a 1 corresponds to the output of the shift register being connected, and 0 corresponds to the opposite. For the case shown in the Figure 2.3, the output sequence 1 corresponds to generator polynomial $g1$, and the output sequence 2 corresponds to generator polynomial $g2$. Since, for the first function generator all the three stages are connected, the generator polynomial is :

$$g_1 = [111] \tag{2.1}$$

For the second function generator, stages 1 and 3 are connected and hence,

$$g_2 = [101]. \tag{2.2}$$

These generators are usually expressed in octal form, and for this case are $G = [7, 5]$.

The three parameters discussed above: Rate (k/n), constraint length (K), and generator polynomial (G) form a particular type of convolutional codeword.

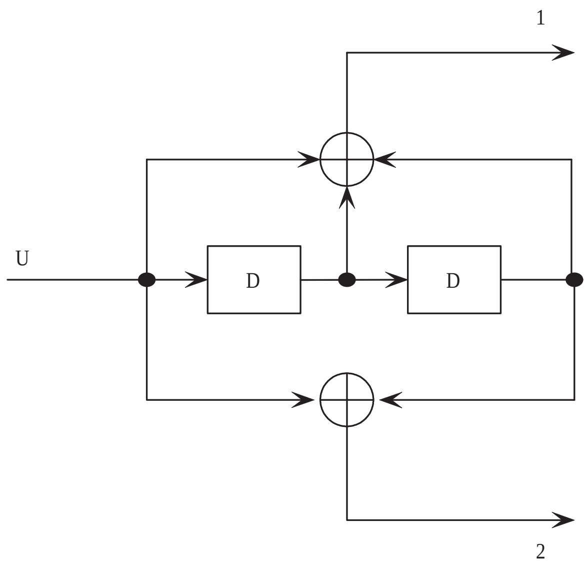


Figure 2.3 : K=3 [7,5] Convolutional Encoder

Trellis representation of convolutional coding is a key to understanding the behavior of convolutional coding. A brief discussion follows. The encoder $(K = 3, [7, 5])$, as discussed earlier, has 4 possible states. Let's give the left shift register(Figure 2.3) a binary weight of 2^1, and the right shift register a binary weight of 2^0. It is usually assumed that the encoder starts in an all zero state. If the first input bit is a zero, zero is shifted across the shift registers , and the encoder stays in the all zeroes state at the next clock edge. However, when the input is one, the encoder transitions to the 10_2 state at the next clock edge. If the next input bit is one, the encoder transitions to the 11_2 state, otherwise, it transitions to the 01_2 state. The following trellis structure in Figure 2.4 shows all the possible transitions for the given encoder. While the horizontal axis represents the stages (advancement in time), the vertical axis represents the possible states. In the figure, the dotted lines represent the transitions when 0 is the input. Also, worth noticing is the fact that the set of bold lines , and feeble lines form a butterfly structure each, as shown in the adjacent boxes in Figure 2.4. The numbers on top of each transition arrow give the output data for that transition.

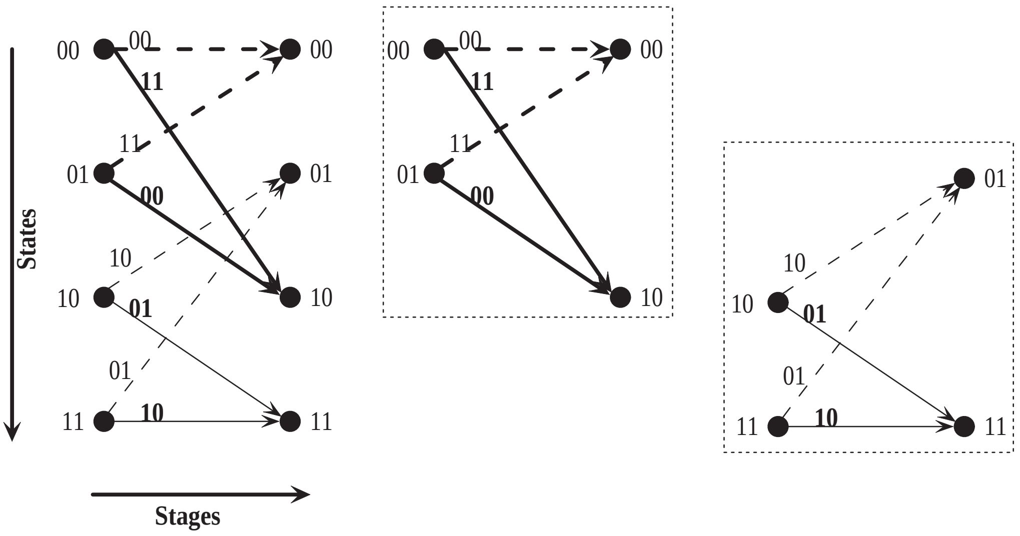


Figure 2.4 : K=3 [7,5] State transitions and the two possible butterflies

2.3.2 Turbo Encoding

In 1993, Berrou et. al [1] introduced a new coding technique referred to as Turbo coding, which represents a significant advance in information theory, by approaching very close to Shannon's channel capacity limit. Turbo coding [12], [15] is a relatively new invention in the world of information theory, but due to its superlative and unmatched performance, much research has gone into its analysis [21]. As shown in

Figure 2.5, a Turbo encoder consists of two parallel binary, rate 1/2 recursive systematic convolutional (RSC) encoders separated by an N-bit interleaver or permuter. A coding rate of 1/3 is obvious from the figure, as for every bit going in, there are 3 bits being outputted.

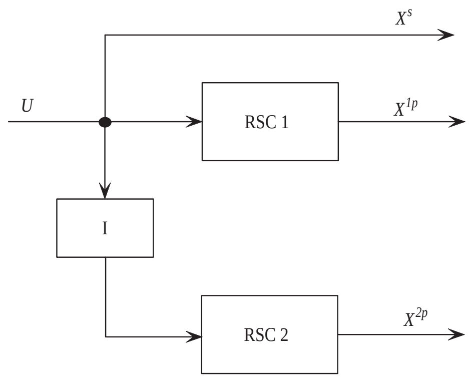


Figure 2.5 : Turbo Encoder

Recursive Systematic Convolutional(RSC) Encoder

Standard feed-forward convolutional encoders are not used for Turbo coding. In any non-recursive convolutional codes, an input string of all zeros except a one at one position results in an encoded stream with a very low weight. The output of the interleaver would also contain a one in a single position, leading to a low weight output from the other encoder. However to improve the performance of Turbo codes, the number of low weight code words needs to be kept to the minimum. To this end, recursive systematic convolutional codes are employed. Here, systematic refers to the fact that one of the output streams is a replica of the input stream. As can be seen in the Figure 2.6 this code is recursive because a set of outputs is fed back to the input. Due to the feedback nature of RSC, an input containing a single one would result in a semi-infinite weight code word. The interleavers thus have a high probability of causing a high weight at the output of the second encoder.

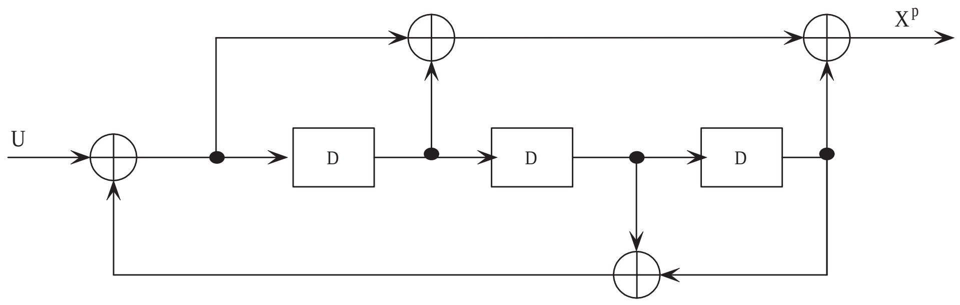


Figure 2.6 : Recursive Systematic Convolutional (RSC) Encoder

Interleaver

Interleaver design has been focus of much attention in the past, as it is an important issue in design of a Turbo coding system. The interleavers are required to perform in such a way that if the output of one encoder is a low-weight codeword, the second encoder should produce a high weight codeword. Random interleavers have proven to be a good choice for interleavers, in terms of performance, but their implementation is costly compared to other interleaver forms [16]. For a random interleaver, the information bits are stored in a vector and read out in a random fashion. This random order should however be known at the receiver in order to decode the data. A simpler interleaver is a block interleaver in which data is written to a matrix of n rows and m columns, and read out by columns.

Rate, constraint length, and generator polynomials are the three parameters that define a convolutional code. 3G systems use convolutional codes with rate 1/2, constraint length 9 and generator polynomial {561, 753} for Data channel, rate 1/3, K=9, generator polynomial {557, 663, 711} codes are used for Control channel. Turbo coding with rate 1/3, and constituent encoders of constraint length 4 as shown in Figures 2.5 and 2.6 is used in 3G. WLAN systems use rate 1/2 constraint length 7 convolutional codes with generator polynomials as {133,171}. The difference in the types of codes being used signifies the differences in the environments to which the signals are subjected. Ideally, an outdoor cellular environment is harsher than WLAN and hence the coding used is more robust.

2.4 Channel Decoding Algorithms

Channel encoding is an extremely simple task when compared to channel decoding. In the following subsections is a description of the different channel decoding algorithms that are used in the different standards. While only Viterbi decoding is used in WLAN, both Viterbi and Turbo decoding are required for 3G. In section 2.4.1, the basic Viterbi algorithm is discussed, which is used for decoding convolutionaly coded data. Two different algorithms for Turbo decoding, namely the Maximum a posteriori probability(MAP) algorithm and the Soft Output Viterbi Algorithm(SOVA) are discussed in later sub-sections. One of the key differences between Turbo decoding (SOVA/MAP) and Viterbi decoding is that while for Viterbi decoding, only hard decisions(+/- 1) are needed to be calculated, for Turbo decoding, reliability of the decisions (also known as soft decisions) are also needed to be computed, and these soft decisions are iterated over many cycles in order to obtain the best results. While MAP algorithm gives a better bit error rate, SOVA offers simpler implementation, implying lesser area and power. Also, SOVA shares many common features with the Viterbi algorithm and hence is a better candidate for our proposed reconfigurable architecture.

2.4.1 The Viterbi Algorithm

In 1967 A. J. Viterbi proposed an algorithm for optimal decoding of convolutionally coded data. The Viterbi algorithm (henceforth referred to as VA), as it is known, finds the most likely sequence of state transitions through a finite state trellis.

The Viterbi decoder, upon reception of a received vector r, generates the estimate z of the transmitted code word, that maximizes the probability $p(r|z)$. The well known Baye's rule for probability is:

$$p(r|z)p(z) = p(z|r)p(r) \tag{2.3}$$

The maximum a posteriori probability(MAP) selects the estimate that maximizes $p(z|r)$, and is equivalent to the Viterbi decoder if the distribution of source word is uniform. The input sequence x, that was presented to the encoder may be represented as: $x = (x_0^0, x_0^1, ...x_0^{k-1}, x_1^0, x_1^1, ...x_1^{k-1}, ...x_L^0, x_L^1, ...x_L^{k-1})$, where L is the number of k bit blocks that together form the information sequence x. As this is a rate k/n code, this input sequence generates an output sequence y, of L n-bit blocks and additional m blocks, where m is the length of longest shift register in the encoder. The output sequence y is given as $y = (y_0^0, y_0^1, ...y_0^{k-1}, y_1^0, y_1^1, ...y_1^{k-1}, ...y_{L+m-1}^0, y_{L+m-1}^1, ...y_{L+m-1}^{k-1})$. However, upon transmission through the channel, noise is added to the data and the received data r can be represented as:

$r = (r_0^0, r_0^1, ...r_0^{k-1}, r_1^0, r_1^1, ...r_1^{k-1}, ...r_{L+m-1}^0, r_{L+m-1}^1, ...r_{L+m-1}^{k-1})$.

From the basic rules of probability, we know that

$$p(r|z) = \prod_{i=0}^{L+m-1} \prod_{j=0}^{n-1} p(r_i^j|z_i^j), \tag{2.4}$$

where the subscript corresponds to the block number and the superscript corresponds to the bit number inside the block. Equation 2.4 is also referred to as the likelihood function of z. By taking the log of both sides we have the log likelihood equation:

$$log(p(r|z)) = \sum_{i=0}^{L+m-1} \sum_{j=0}^{n-1} log(p(r_i^j|z_i^j)). \tag{2.5}$$

The term $log(p(r_i^j|z_i^j))$ is converted to an easier form called the bit metrics which can be denoted as: $M(r_i^j|z_i^j)$. The term path metric refers to the sum of these terms and is given by:

$$M(r|z) = \sum_{i=0}^{L+m-1} \sum_{j=0}^{n-1} M(r_i^j|z_i^j). \tag{2.6}$$

Another way to express the path metrics is using the branch metrics, where the k^{th} branch metric is defined as the sum of the bit metrics for the k^{th} block of r given z,

$$M(r_k|z_k) = \sum_{i=0}^{n-1} M(r_k^j|z_k^j). \tag{2.7}$$

The k^{th} partial path metric is obtained by summing the branch metrics for the first k branches traversed by the path.

Following is a brief description of how the algorithm proceeds to evaluate the output data. In the first step, Euclidean distances are used to evaluate branch metrics for all possible state transitions. Subsequently, from the two paths that end in the same state (for rate $1/n$ code), the path associated with the best path metric is selected as most likely, and a decision corresponding to this is calculated. A trellis structure is built along the way, where each state transition is noted with all the relevant information like path metric and the decision bit. When all the data bits have been received, and the trellis has been completed, the last stage of trellis is used as the starting point for tracing back. The traceback operation outputs the decoded data, as it traces back along the maximum likelihood path.

2.4.2 Maximum A posteriori Probability (MAP) decoding for Turbo codes

The MAP algorithm was proposed by Berrou et. al in 1972 [2]. A detailed and lucid exposition of MAP and SOVA algorithms for Turbo decoding was published by Hagenauer [4], and is a useful reference. The trellis structure of a binary feedback convolutional encoder has the structure shown in Figure 2.7. S_k is the encoder state at time k, u_k is bit associated with the transition from time $k-1$ to k, and the trellis states at level $k-1$ and at level k, have been indexed by the integers s' and s respectively. In this algorithm, the decoder decides $u_k = +1$ if $P(u_k = +1|y) > P(u_k = -1|y)$, and it decides u_k =-1, otherwise, where y is the received noisy sequence. In the logarithm domain, the decision u_k is given by:

$$u_k = sign[L(u_k)], \tag{2.8}$$

where $L(u_k)$ is log a posteriori probability ratio defined as:

$$L(u_k) = log[\frac{P(u_k = +1|y)}{P(u_k = -1|y)}] = log[\frac{\sum\limits_{\substack{(s',s)\\u_k=+1}} p(s', s, y)}{\sum\limits_{\substack{(s',s)\\u_k=-1}} p(s', s, y)}]. \tag{2.9}$$

Using Bayes rule, Eqn 2.9 may be expressed as :

$$L(u_k) = log[\frac{P(y|u_k = +1)}{P(y|u_k = -1)}] + log[\frac{P(u_k = +1)}{P(u_k = -1)}], \tag{2.10}$$

where the second term gives the *apriori* information of bit u_k. The pair s' and s determines the information bit u_k and the coded bits $x_{k,\nu}$, for $\nu = 2, \ldots.., n$. The sum of the probabilities $p(s', s, y)$ in the denominator or numerator of equation 2.9 is taken over all existing transitions from state s' to state s labeled with the information bit $u_k = -1$ or with $u_k = +1$, respectively. The joint probability $p(s', s, y)$ can be

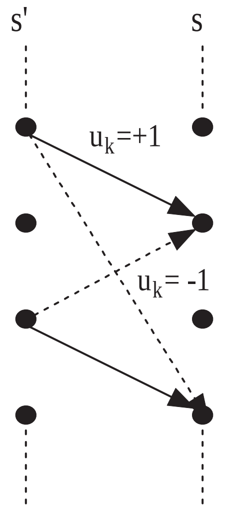


Figure 2.7 : Trellis structure for systematic feedback convolutional encoders

expressed as a product of three independent probabilities:

$$p(s', s, y) = p(s', y_{j<k}).p(s, y_k|s').p(y_{j>k}|s), \tag{2.11}$$

or

$$p(s', s, y) = p(s', y_{j<k}).P(s|s').p(y_k|s'.s).p(y_{j>k}|s), \tag{2.12}$$

which can be expressed as:

$$p(s', s, y) = \alpha_{k-1}(s').\gamma_k(s', s).\beta_k(s). \tag{2.13}$$

Here $y_{j<k}$ denotes the sequence of received symbols y_j from the beginning of the trellis up to time $k-1$ and $y_{j>k}$ is the corresponding sequence from time $k+1$ up to the end of trellis. The forward recursion of the MAP algorithm yields,

$$\alpha_k(s) = \sum_{s'} \gamma_k(s', s).\alpha_{k-1}(s'). \tag{2.14}$$

The backward recursion yields,

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s', s).\beta_{k-1}(s). \tag{2.15}$$

In order to perform the optimum 'symbol-by-symbol' MAP rule, the trellis has to be finite. Its assumed that at the start and at the end of the observed sequence all paths merge at the zero state. Thus, the forward and backward recursion are initialized with $\alpha_{start}(0) = 1$ and $\beta_{end}(0) = 1$. Upon a transition between s' and s, the branch transition probabilities are given by :

$$\gamma_k(s', s) = p(y_k|u_k).P(u_k). \tag{2.16}$$

Using the log-likelihood ratios, the a priori probability $P(u_k)$ can be expressed as:

$$P(u_k = +/-1) = \frac{e^{+/-L(u_k)}}{1 + e^{+/-L(u_k)}} = (\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)/2}}).e^{L(u_k)u_k/2} = A_k.e^{L(u_k)u_k/2}. \tag{2.17}$$

Similarly $p(y_k|u_k)$ for systematic convolutional codes can be written as:

$$p(y_k|u_k) = B_k exp(\frac{1}{2}L_c y_{k,1} u_k + \frac{1}{2}\sum_{\nu=2}^{n} L_c y_{k,\nu} x_{k,\nu}). \tag{2.18}$$

The terms A_k and B_k are equal for all transitions from $k-1$ to k and will cancel out in the ratio in equation 2.9. Hence, the branch transition operation to be used in equations 2.14 and 2.15 reduces to :

$$exp(\frac{1}{2}u_k(L_c y_{k,1} + L(u_k))).\gamma_k^{(e)}(s', s), \tag{2.19}$$

where

$$\gamma_k^{(e)}(s', s) = exp(\frac{1}{2}\sum_{\nu=2}^{n} L_c y_{k,\nu} x_{k,\nu}). \tag{2.20}$$

The first exponential in equation 2.19 is common in all terms in the sums of equation 2.9, and hence we can divide all terms by these and obtain the soft output of MAP algorithm as:

$$L(u_k) = L_c u_{k,1} + L(u_k) + log(\frac{\sum\limits_{\substack{(s',s)\\ u_k=+1}} \gamma_k^e(s', s).\alpha_{k-1}(s').\beta_k(s)}{\sum\limits_{\substack{(s',s)\\ u_k=-1}} \gamma_k^e(s', s).\alpha_{k-1}(s').\beta_k(s)}). \tag{2.21}$$

2.4.3 Soft Output Viterbi Algorithm

The Viterbi algorithm as proposed by Andrew J. Viterbi is of the soft input hard output type (SIHO). For decoding turbo codes, soft outputs are required for reliability information that is passed between the decoders. Therefore the Viterbi algorithm was modified [3], to give soft outputs, as in the MAP algorithm. Forney described the VA in its MAP format [17]. It searches for the i^{th} state sequence $S^{(i)}$ and thus the desired information sequence $u^{(i)}$ by maximizing over i, the a-posteriori probability:

$$P(S^{(i)}|y) = p(y|S^{(i)})\frac{P(S^{(i)})}{p(y)}. \tag{2.22}$$

Given that y is fixed, we can equivalently maximize:

$$p(y|S^{(i)})P(S^{(i)}). \tag{2.23}$$

This maximization is obtained when for each state s and each time k the path with the largest probability $p(S^{(i)}_{j\leq k}, y_{j\leq k})$ is selected. This probability can be calculated by multiplying the branch transition probabilities associated with path i. They are defined in equation 2.16. Since logarithm is monotonically increasing, the maximum is not altered upon a log operation and hence we perform the same metric computation as in the forward recursion of log-MAP in previous section. In equations 2.17 & 2.18 $logA_k$ and $logB_k$ are the same for all paths and hence irrelevant. For the i^{th} path at time k we obtain the path metric as

$$M_k(s^{(i)}) = M_{k-1}(s^{'(i)}) + (1/2)L(u_k)u_k^{(i)} + (1/2)\sum_{v=1}^{n} L_c y_{k,v}.x_{k,v}^{(i)} \tag{2.24}$$

Here $s^{(i)}$ denotes the state of the path i at time k, $u_k^{(i)}$ is the information bit, and $x_{k,v}^{(i)}$ are the coded bits of path i at time k. For systematic codes we have

$$M_k(s^{(i)}) = M_{k-1}(s^{'(i)}) + (1/2)L_c y_{k,1}u_k^{(i)} + (1/2)\sum_{v=2}^{n} L_c y_{k,v}x_{k,v}^{(i)} + (1/2)L(u_k)u_k^{(i)}. \tag{2.25}$$

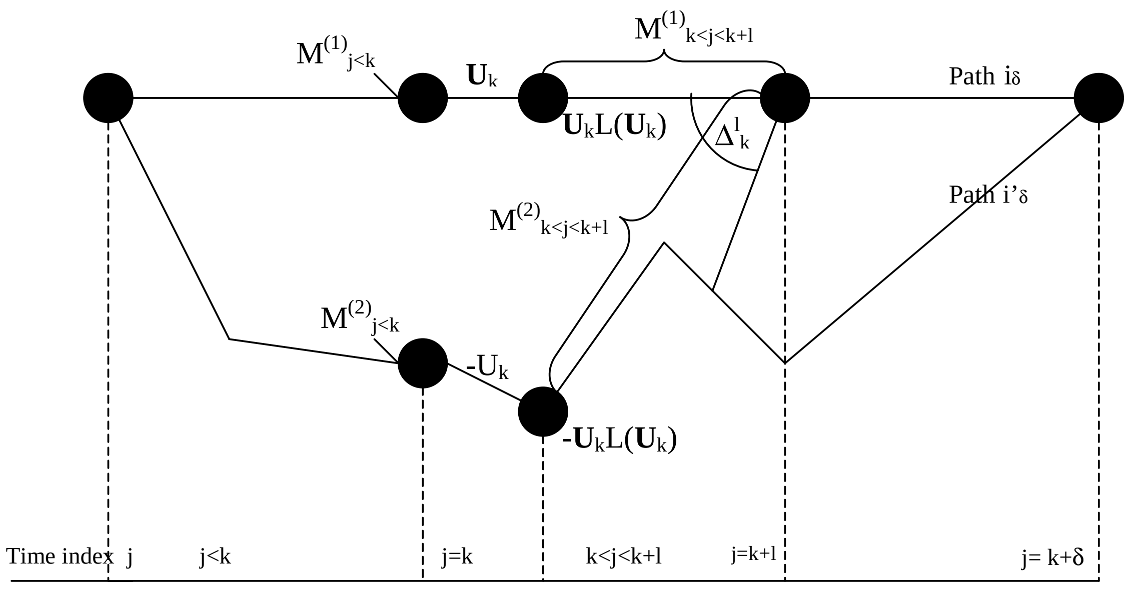


Figure 2.8 : Derivation of the metric difference Δ^l_k

This modification of the metric for the Viterbi algorithm (VA) incorporates the *apriori* information about the probability of the information bits. If we have a good channel, $|L_cy|$ will be larger than $|L(u)|$ and the decoding relies on the received channel values. If the channel is bad, decoding relies on the *apriori* information $L(u)$. In iterative decoding this is the extrinsic value from the previous decoding step.

The joint probability of the path i and of the received sequence $y_{j\leq k}$ at time k, and the metric in equation 2.25 are related by:

$$p(path, i, y_{j\leq k}) = p(S^i_{j\leq k}, y_{j\leq k}) = (\prod_{j=1}^{k} A_j.B_j).exp(M_k(s^{(i)}). \tag{2.26}$$

We wish to obtain the soft output for bit $\hat{u_k}$, which the VA decides after a delay δ. Figure 2.8 shows the computation of this soft ouput.

The VA proceeds in the usual way be calculating the metrics for path i using equation 2.25. For each state it selects the path with the largest metric $M_k(s^{(i)})$. At time $k+\delta$ the VA has selected the maximum likelihood (ML) path with index i_δ and discarded the other path. Along the ML path i_δ which decides the bit $\hat{u_k}$, $\delta+1$ nonsurviving paths with indices $l = 0...\delta$ have been discarded. The metric difference is defined as

$$\Delta_k^l = M_{k+l}^{s^{(i_l)}} - M_{k+l}^{s^{(i_l^{'})}} \geq 0. \tag{2.27}$$

Then, it can be shown that the probability $P(correct)$ that the path decision of the survivor was correct at time $k+l$ given $y_{j\leq k+l}$ is, using equation 2.26

$$P(correct) = \frac{p(pathi_l, y_{j\leq k+l})}{p(pathi_l, y_{j\leq k+l}) + p(pathi_l', y_{j\leq k+l})}, \tag{2.28}$$

or

$$P(correct) = \frac{exp(M_{k+l}(s^{(i_l)}))}{exp(M_{k+l}(s^{(i_l)})) + exp(M_{k+l}(s^{(i_l')}))}, \tag{2.29}$$

which implies,

$$P(correct) = \frac{exp(\Delta_k^l)}{1+exp(\Delta_k^l)}. \tag{2.30}$$

Hence, the likelihood ratio of this path decision is Δ_k^l, because

$$\log\frac{P(correct)}{1-P(correct)} = \Delta_k^l. \tag{2.31}$$

The soft output of the VA is the decision $\hat{u_k}$ times the Δ_k^l values of the errors and can be approximated as

$$L(\hat{u_k}) \approx \hat{u_k}\sum_{l=0}^{\delta}\Delta_k^l \approx \hat{u_k}.\min_{l=0..\delta}\Delta_k^l \tag{2.32}$$

The sum and the minimum is only over those nonsurviving paths which would have led to a different decision $\hat{u_k}$. Thus we have the same hard decisions as the classical VA, and the reliability of the decisions is found by taking the minimum of the relevant metric differences along the ML path.

The soft output in its approximate version of eqn 2.32, using equations 2.25 and 2.27 has the following format

$$L_{SOVA}(\hat{u_k}) = Lcy_{k,1} + L(u_k) + \hat{u_k}.(first\, 3\, terms\ in\ eqn. 2.24). \tag{2.33}$$

However, in the proposed architecture, a slightly different approach [7] is used. No hard decisions are made until $k =$ frame size. Once k reaches frame size, the path ending at the state with the highest metric is selected as the survivor path. The ML path is traced back to obtain the hard decisions $\hat{u}_k$, and the corresponding Δ_k. We define another parameter called the reliability depth as δ. The reliability value Δ_k^* is obtained by taking the minimum over the nonsurviving path within the time window $[k, k+\delta]$, that would have led to a different decision. The soft output for bit u_k is approximated by $L_{sova} = (\hat{u}_k).\Delta_k^*$. A safe estimate of reliability depth is nearly $3 * K$ (where K is constraint length).

Chapter 3

Viterbi and Turbo Decoding Architectures

While the last chapter detailed the various decoding algorithms, this chapter shall delve into the architectures for these algorithms. We shall break up the Viterbi and Turbo decoding architectures into smaller sub units and analyze the workings of each unit.

Four major components of a Viterbi decoder as shown in Figure 3.1 are: Branch Metric Unit, Add Compare Select Unit, Survivor Management Unit, and Control Unit. A complete Turbo decoder, as shown in Figure 3.2 comprises of two main processing elements: the Soft Input Soft Output block, implementing the SOVA or MAP algorithm and the Interleavers/Deinterleavers which scramble/descramble the data according to the interleaving algorithm used on the encoding side. For the case of SOVA based Turbo decoding system, the four units that form a Soft Input Soft Output block (shown in the blowup of Figure 3.2) are similar to the constituents of a Viterbi decoder, but the complexity of these units is higher for Turbo decoding. Especially, the Survivor Management unit for Turbo decoding is extremely complex compared to Viterbi decoding, as it also generates the soft outputs.

3.1 Branch Metric Unit (BMU)

An integral component of Viterbi and Turbo decoding, the BMU computes the branch metrics for each state and stage of the trellis. As discussed in the previous chapter,

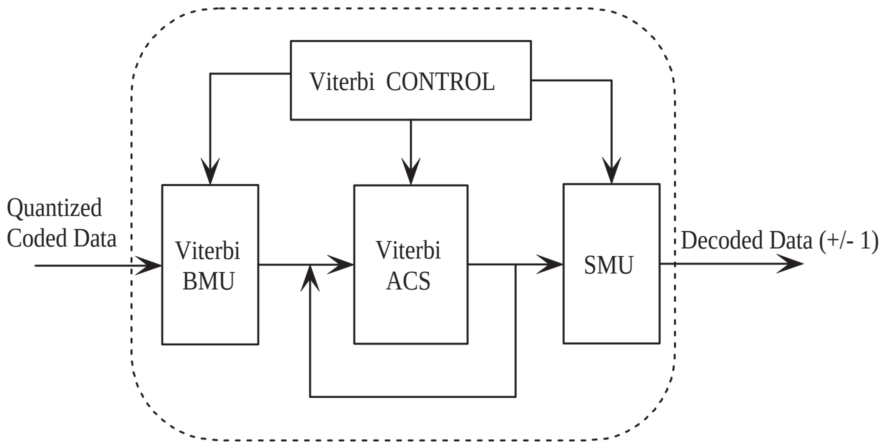


Figure 3.1 : Viterbi decoder

there are 2^{K-1} states and 2^k branches that go into each state at each stage of the trellis, and hence $(2^{K-1}) * (2^k)$ branch metrics are required to be calculated for each stage. However, for an n bit codeword, there are 2^n *unique* codewords and hence as many unique branch metrics need to be computed. Branch metric computations are contingent upon the rate of the code (factor n) used, constraint length, and the generator polynomial used to derive the code.

Various different metrics are used for computation of the branch Metrics. For the case of SOVA based Turbo decoding, the branch metric is computed as shown in Equation 2.25. This computation involves the use of multipliers and adders, as is obvious from the equation. However, various implementations use look up tables instead of multipliers to evaluate the metrics in order to save computational complexity and power consumption.

For the case of Viterbi decoding, Hamming distance and Euclidean distance are two very popular metrics. While Hamming distance computations are fairly simple, in the fact that only single bit Exclusive Or operations are required, they do not provide

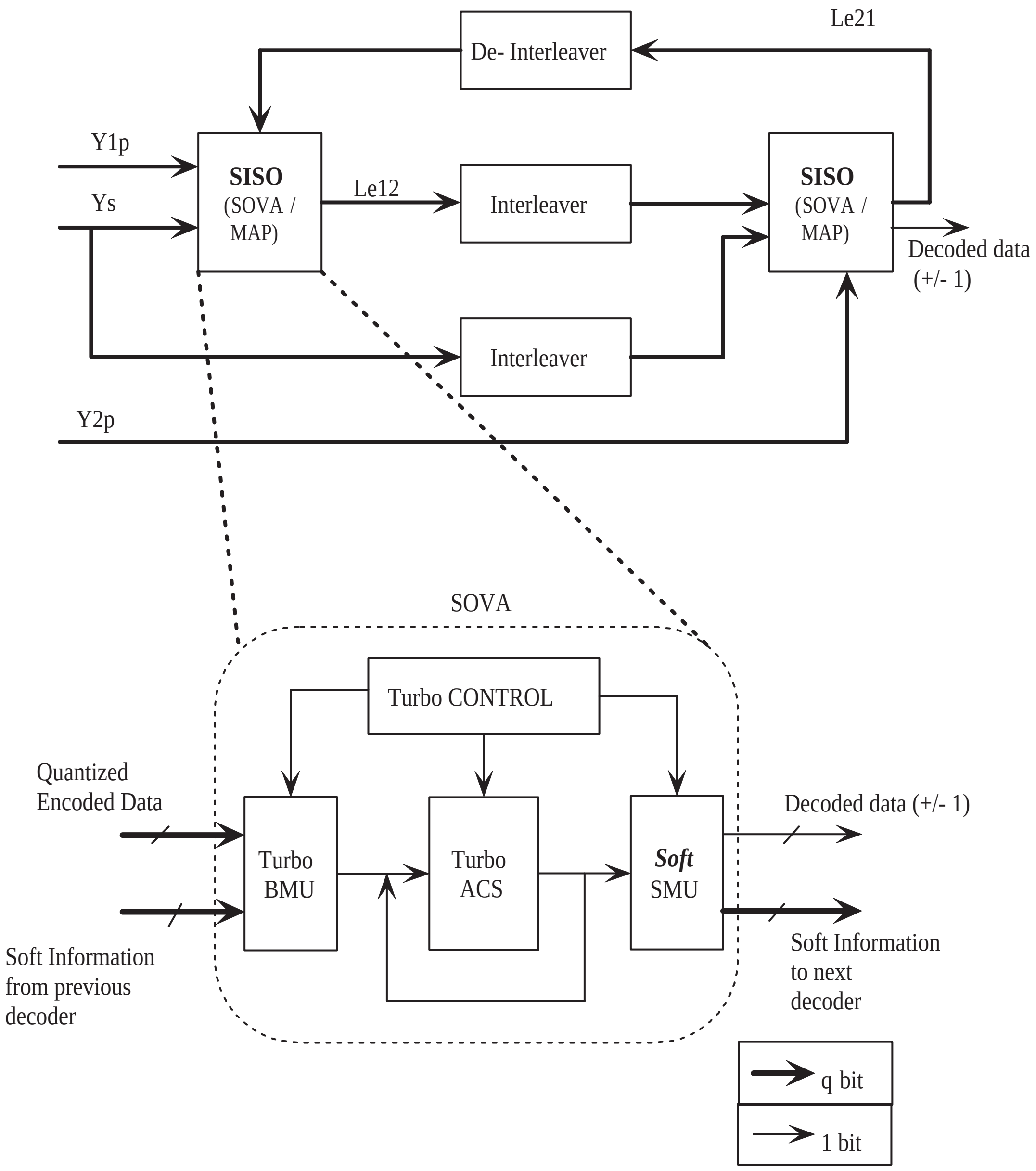


Figure 3.2 : A complete Turbo Decoder

the computational accuracy of Euclidean distances metrics. In the case of Euclidean distances, multipliers are required for evaluating the squares of distances, and adders are required for subsequent processing. Also, in various architectures, LUTs are used for computation of branch metrics, as they offer a compromise between the very precise Euclidean and the very simple Hamming distances metrics. As discussed in the previous chapter, the branch metrics contribute to the path metric computations. The branch metrics generated by the BMUs are passed on to the ACS units according to the code configuration, for the subsequent computations.

3.2 Add Compare Select (ACS) Unit

Computationally, one of the most intensive, this unit uses the accumulated path metrics and the current branch metrics to compute the survivor path metrics at every state and stage of the trellis. These survivor path metrics are again fed back to the (same/other) ACS units for further processing. Therefore, as we advance in time and build up the trellis, the same set of ACS units is used for processing the ever changing path metrics.

As shown in Figure 3.3, along with the new surviving path metrics, the decision bits required for survivor management are also evaluated in case of Viterbi decoding. For Turbo decoding, the difference between the path metrics of the competing paths is also required, as it is used for the computation of soft information, which is passed between different iterations. It may be noted here that the computation of survivor path metric and decision bits is dependent upon the type of metrics used for branch metric computation. Therefore, while the maximum of two path metrics is selected as survivor in case of Turbo decoding as described earlier, the minimum of two competing path metrics may be used as survivor in Euclidean distance based Viterbi decoding.

It is worth noting here, how the butterfly structures in the trellis are exploited for ACS computation. As shown in the Figure 4.5, a certain butterfly has a fixed pair of starting and ending stages, for a given constraint length K. 2^{K-2} butterfly structures make up a single stage of the trellis. For a rate k/n code there are 2^k competing paths entering each node of the trellis.

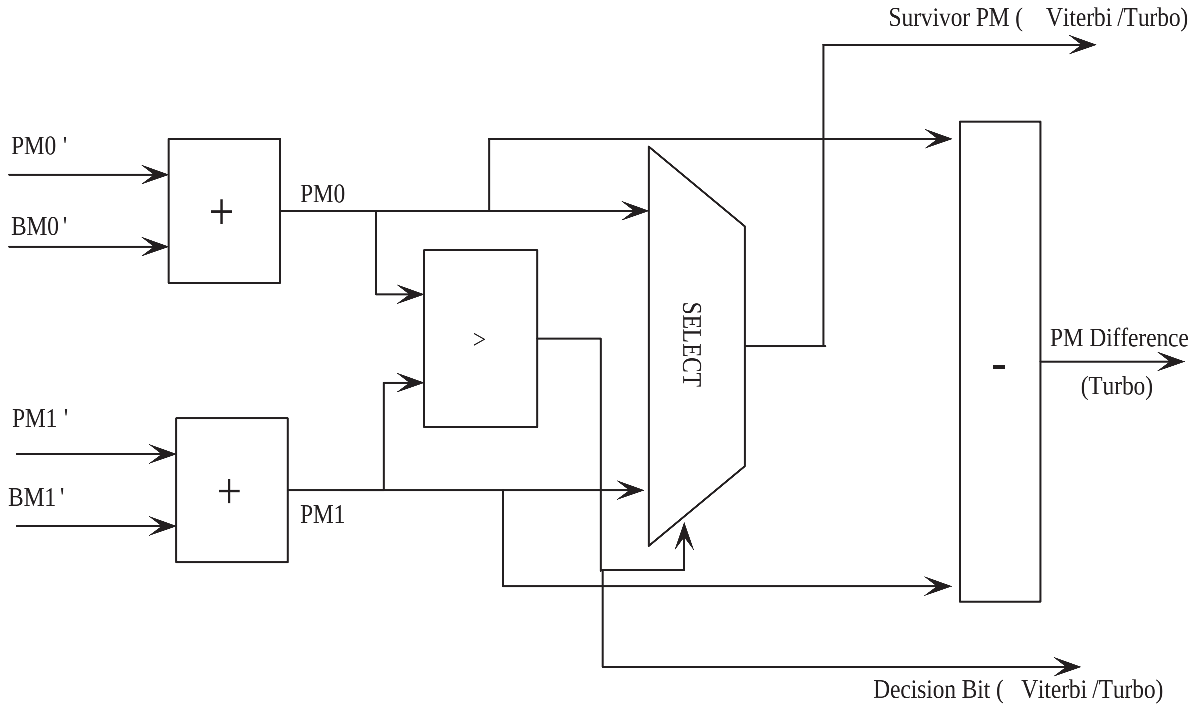


Figure 3.3 : Add Compare Select Unit for Turbo Decoding

3.2.1 Metric Normalization

As finite number of bits are used for storing path metrics and these path metrics accumulate over time, there is a need to normalize the path metrics in order to avoid overflow. Every path metric computed is compared against a threshold at every stage of the computation. If, in any case any path metric exceeds this threshold, a flag is set, and a certain constant value is subtracted from all path metrics, hence providing a mechanism against overflow.

3.3 Survivor Management Unit

This unit uses the decision bits obtained from the ACS unit to compute the decoded data bits. There are two competing algorithms for survivor management, namely register exchange and traceback. The register exchange implementations are superior in terms of regularity of design and decoding latency, but traceback implementations achieve higher implementation efficiency and lower power consumption.

3.3.1 Register Exchange

The register exchange algorithm computes the symbol sequence of survivor paths associated with all trellis states based on the decisions on the paths that merge in a specific state. A shift register is associated with every node in the decoding operation, and contains the information regarding the surviving partial path that terminates at that node. Such registers must be able to send and receive data to and from registers, and all of this information exchange must take place simultaneously, for high speed. As the decoding operation proceeds, information is updated and exchanged, contingent upon the survivor branches. Due to the high access bandwidth, power consumption becomes a very critical issue in this implementation. Computationally expensive but conceptually lucid, this operation is pitted against the traceback operation as described below.

3.3.2 Traceback

A history of surviving branches at each node is stored for traceback operation. Noteworthy is the fact that the contents of registers associated with each state are not exchanged with each other, but only updated by one bit with each successive stage of decoding. Such an implementation is significantly faster than register exchange, and is usually preferred when hardware implementation is the ultimate goal. An important property of the traceback operation is that if every state from a current time is followed backwards through its ML path, all the paths converge at a point somewhere previous in time. The number of stages required to be traced back, for convergence of paths is known as the decoding depth and is a linear function of constraint length. Five times the constraint length has been proven to a be a good estimate for decoding depth. The traceback for VA is relatively simple in the sense that only one ML path needs to be traced back for the computation of decoded data. However for computation of soft decisions, traceback becomes very intensive, details of which follow.

Soft Information for Turbo Decoding

The Soft Output Viterbi Algorithm uses the difference of incoming path metrics to a state, as a measure of the reliability of a particular decision at that state. While, the Viterbi algorithm traces back over one path, the SOVA traces back over the Maximum Likelihood(ML) path and its next competitor, in order to establish the reliability of a decision. This is based on the idea that all the decision bits along the traceback path are based on correctly choosing the ML path, and if the competitor path has a different decision, it can only be as confident as the decision to choose the competitor over the ML path. In the original form, as many tracebacks as the number of states were performed, because at that point the ML path was unknown. A more elegant approach was suggested by Meyr et. al. [6], named as Two Step SOVA-SMU.

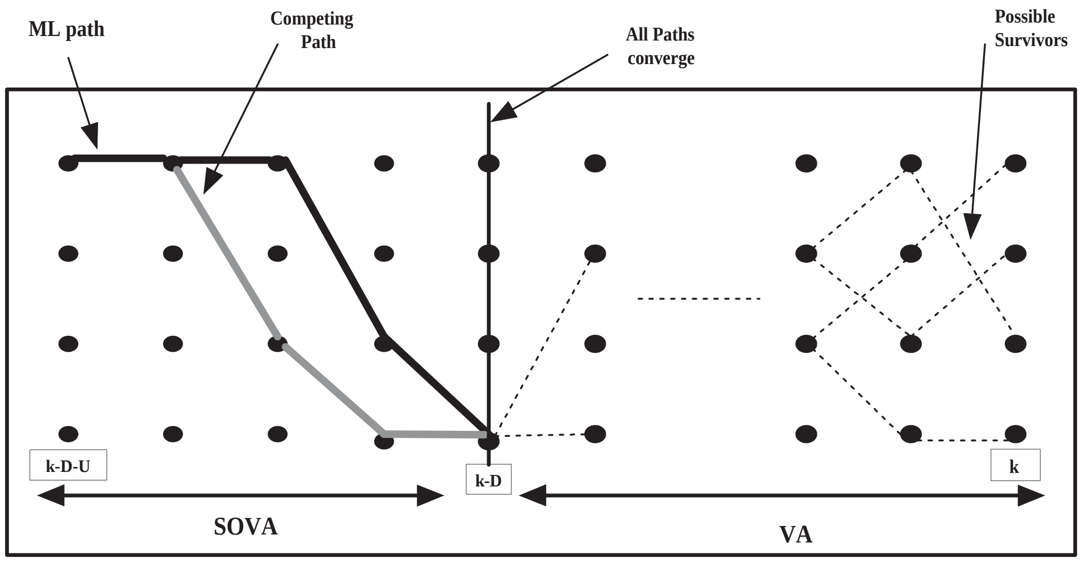


Figure 3.4 : Two Step SOVA decoding

Two Step SOVA-SMU Architecture

One of the most popular techniques for traceback for Soft Output Viterbi Algorithm was proposed [6] before the advent of Turbo Codes. The proposed algorithm split the required traceback operations for soft decision computation into two steps. As a first step, hard deciding Traceback Unit is employed to compute the final survivor sequence while the update operation is postponed for the next step. Once the final survivor is found, only the likelihood values associated with this survivor sequence are updates with respect to the competing paths that are traced back starting from the known survivor states at the survivor depth D. In Figure 3.4 the right hand side shows the hard decision traceback, where the final survivor is determined. The left hand side shows that in each decoding step, a comparison between the final survivor path and a competing path is mandatory. The starting point for the comparison is the final survivor state at depth D, and a depth of U is traced back in order to compute the soft decisions. The number U is known as the reliability depth and is about 3 times the constraint length.

A high level block diagram of the update processing unit is shown in Figure 3.5. In each decoding step, one new path comparison is started, which finishes the soft decision computation after U cycles. Here dec_s corresponds to the decision bit, s_{k-D} corresponds to the starting state of the path comparison, $Del'_{s,k}$ $(\Delta'_{s,k})$ corresponds to the difference between the path metrics at the states s and time k, and L_{k-D-U} is the soft decision or the likelihood.

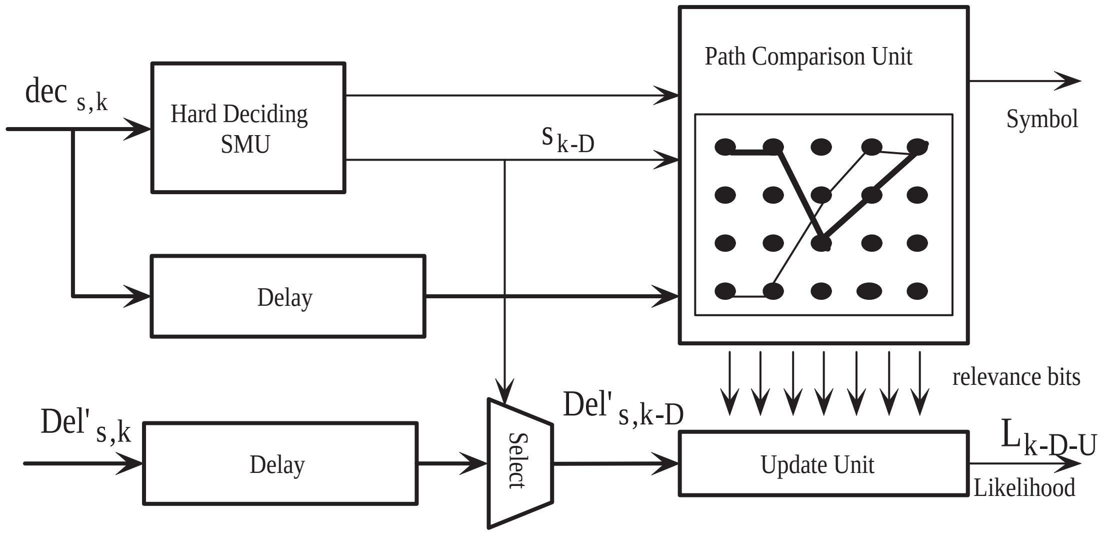


Figure 3.5 : Two Step SOVA Soft Decision Traceback Unit

3.4 Interleaver

Interleavers, are in effect address generation units, where the address is generated according to a certain mathematical formula. For example in the case of block interleavers, data is written row-wise, using an address generation unit, in n rows and m columns. When reading out, the data is read out column-wise, by another address generation unit, hence interleaving the data.

By its very nature, decoding Turbo coded data is an iterative process. However this iterative decoding can be realized with a single processor that performs all the computations or with a pipelined architecture that allows data to flow through a series of computational elements. The choice of a particular architecture is contingent upon the requirements of a given system. Various implementations of Turbo decoders on Digital Signal Processors [18], Field Programmable Gate Arrays [22], and Application Specific Integrated Circuits [11] have been proposed so far.

Chapter 4

VITURBO : The Reconfigurable Architecture

In this work, we have presented a reconfigurable architecture for future wireless systems. The crux of the contribution comes in terms of reconfigurable architecture for Viterbi and Turbo decoding for varying code configurations. Flexibility is a key to this work, and has been realized in hardware using various control and routing structures. Figure 4.1 shows a complete block diagram of the proposed architecture, with the components critical to reconfigurability, shaded. As we have seen and shall see, the different decoding algorithms share various common units, but the number of units used and the connections of these units are highly variable and change upon any change in the coding parameters. For instance, the path metrics computed from one ACS unit might need to be routed back to any of a number of ACS units that need the updated path metrics for their computation. With the change of constraint length or coding rate or decoding type, the data routes are altered by the multiplexers, and the operations being performed inside a unit are changed accordingly. Considering the fact that we can decode constraint length 3 to 9 and rate 1/2 or 1/3 convolutional codes, we have 14 different configurations that can be handled by this system. Adding to this number is the Turbo decoder, bringing the total possible configurations to 15. On top of this, our system can decode codes generated with any generator polynomial possible. Though all the different possible generator polynomials are not used in practice, the number of possible generator polynomials for K=9 and rate 1/2 alone is a whopping $2 * 2^9 = 1024$.

4.1 Related Work

Reconfigurable/Flexible decoding architectures have been actively researched in the last decade. While some work has been done on flexible Viterbi decoding architectures, the idea of reconfiguration between Turbo and Viterbi decoding is nascent. Most related works have been limited to certain specific standards or applications, VITURBO has taken a step away from it, as the ultimate motivation for our system is ubiquitous communications, where any code configuration may be supported, depending upon the environment.

A unified Viterbi and Turbo decoder for 3GPP [8] has been proposed recently. In this work, log-MAP algorithm is used for Turbo decoding, and the maximum data rate for Turbo decoding is 2 Mbps. Also, the Viterbi decoder is mainly aimed at low data rate(12 Kbps) voice channels. The ACS processing is done by 4 Viterbi/log-MAP butterfly units, which is sufficient for 3G data rates, but can not support the extremely high data rates demanded by systems like WLAN. Such a system might be useful for a certain specific standard like 3G, but is hardly useful when multiple standards/systems are in question.

Texas Instruments' flexible Viterbi decoder [9], which operates within a programmable Digital Signal Processor (DSP) is another such system. Though a flexible system with variable constraint lengths and code rates, the throughput is limited to 2.5 Mbps. The coprocessor is extremely dependent on the DSP, and can not work as a standalone processor. Such a decoder is again limited to only certain applications, and is dependent on the processor type being used with it.

Another interesting reconfigurable Viterbi decoding architecture was proposed by Chadha et. al. [13]. This system uses single bit Hamming distance metrics which is hardly usable in a practical scenario. Also, the decoder can only perform Viterbi decoding, hence limiting flexibility. Though aggressively designed with a fully parallel ACS architecture, a major drawback is that even when the decoder is decoding a constraint length 3 code, it is consuming as much power as required for constraint length 9 decoding. It may be noted here that the number of ACS units required for constraint length 9, fully parallel decoding is 64 times that required for constraint length 3, fully parallel decoding.

A foldable state scheduling scheme for processing a wide range of constraint lengths, with a fixed set of hardware was another flavor of reconfigurable/flexible Viterbi decoding. According to the proposed work [19], configuration data needs to loaded into the hardware every time there is a need to switch between different configuration. This loading of configuration data is a slow process compared to the '*click of a button*' reconfigurability offered by our system. Also, Turbo decoding is not available with this decoder. However, the proposed architecture has a utilization of 100% for almost all configurations, which means that area used has been minimal.

4.2 VITURBO : Salient Features

In this thesis, three important issues have been dealt with. First and foremost is the issue of support for Viterbi as well as Turbo decoding. Using the SOVA algorithm for Turbo decoding and Viterbi algorithm for Viterbi decoding was one strategy that paid rich dividends . Even though SOVA is much more complicated than the simple Viterbi algorithm, there are many similarities which have been exploited. For example, the ACS operation is inherently similar, even though the branch metrics feeding the ACS are computed differently, and the trellis structures are different. Similarities in Traceback operations of the two decoding techniques have also been exploited.

Another important issue is the reconfiguration between different constraint lengths and rates, on top of the two different decoding techniques. A fully parallel architecture (Figure 4.4), for the highest constraint length is employed in order to provide the high data rates as made mandatory by the different standards (802.11a WLAN systems support up to 54Mbps, 3G systems support up to 2 Mbps). The fully parallel architecture also provides flexibility for decoding not only constraint length 9 codes, but also constraint length 3 codes, where the latter uses parts of the former. Multiplexers have been employed for routing the path metric, and the path metric differences to and from the ACS units in order to provide this reconfigurability. Also, the decision bits are passed through multiplexers, so as to reorder the data going to the memory.

Limited power consumption is another feature of this reconfigurable architecture. As mentioned earlier, a fully parallel architecture has been employed for this decoder. However, not all the units are used for all the different decoding techniques, that are possible with this architecture. For example, while the constraint length 9 decoder uses all the ACS units in the decoder, constraint length 7 decoder uses a quarter of the ACS units available. Also, various parts of circuit are not used for Viterbi decoding, but are needed for Turbo decoding. Hence, schemes have been devised to power down parts of the circuit that are not being used for the decoding type in progress.

The complete architecture has some very important computational units, which play a significant role in the reconfiguration of the system. These are the Branch Metric Unit, Add Compare Select Unit , Multiplexer Bank, and Survivor Management Unit, as shown in Figure 4.4. As mentioned earlier, the architecture is completely parallel, and hence for the case of constraint length 3-9, we have as many as 128 ($2^{K_{max}-2}$) ACS units, each of which caters to a single butterfly structure, as shown in Figure 4.5. A detailed explanation follows.

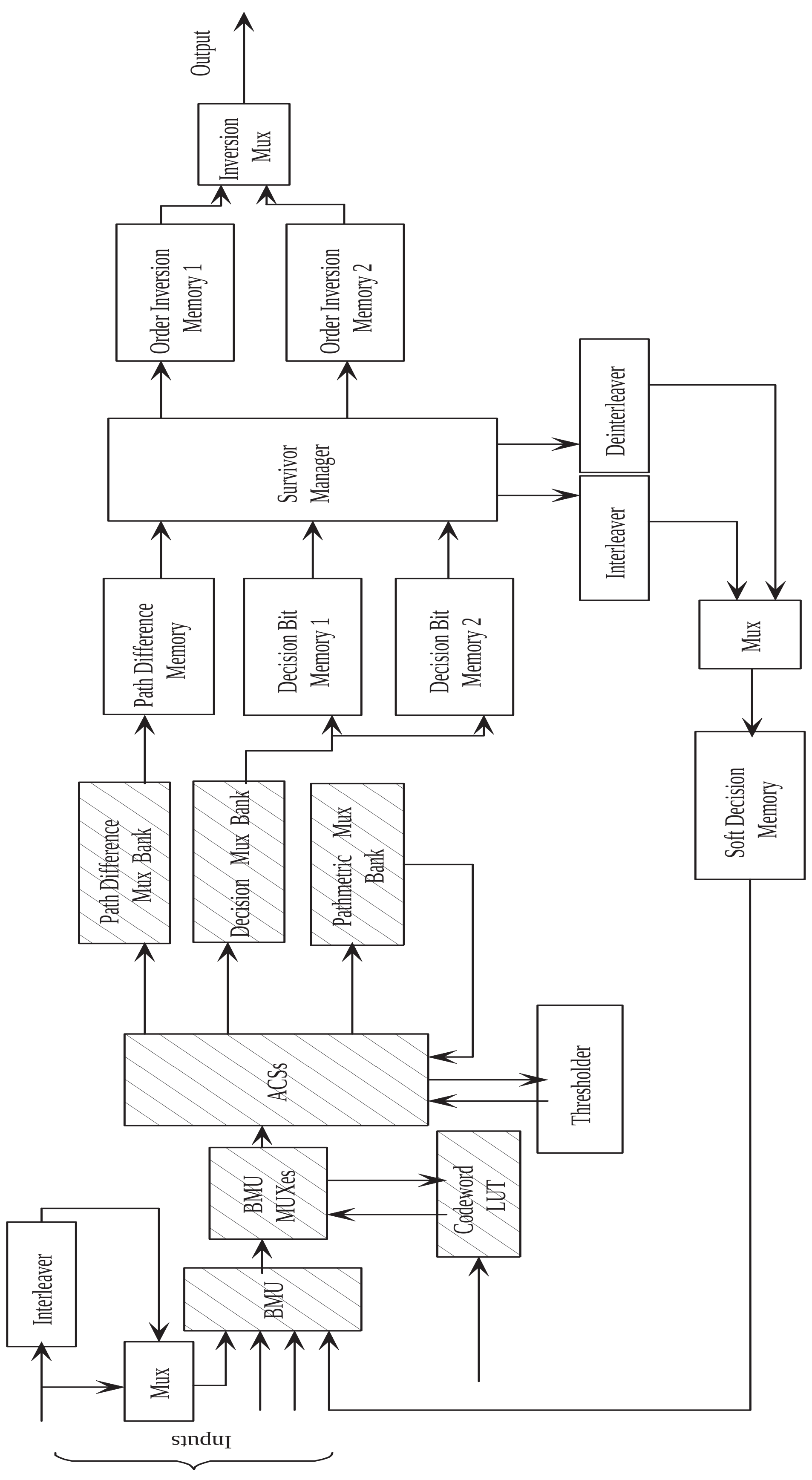


Figure 4.1 : VITURBO: The complete architecture

4.3 Branch Metric Unit (BMU)

In order to provide enhanced flexibility, the Branch Metric Unit has been divided into three different computational units: $BMcompute$ unit, a $Codeword\ Look-Up\ Table$, and $2^{K_{max}-2}$ $BMmuxes$, as shown in Figure 4.2

4.3.1 BMcompute

The $'BMcompute'$ unit computes all the possible branch metrics for a given decoder type with inputs being, the received data, the decoder type(Turbo/Viterbi), and the code rate. It must be noted here that the number of possible branch metrics is equal to 2^n, where n is the denominator in the code rate formula (k/n). Depending on the decoder type, branch metrics for either Viterbi or Turbo decoding are computed. Multipliers and adders form this unit, and the process has been pipelined, with multiplication and addition/subtraction being processed in two different steps. For Viterbi decoding , Euclidean distances are calculated(which are reduced to dot products). The branch metrics for Turbo decoding are obtained as in equation 2.24. These branch metrics are fed to the ACS units using $BMmuxes$, as described below.

4.3.2 BMmux and Codeword Look-Up Table

Each ACS unit needs a specific pair of branch metrics, which is obtained using a multiplexer, named $BMmux$. This multiplexer plays a critical role in the reconfiguration, as it has to select the pair of branch metrics depending upon the constraint length, rate, coding type and the index of the ACS unit in question. The $BMmux$ uses the $Codeword\ Look-Up\ \ Table$ to implement this.

The $Codeword\ Look-Up\ \ Table$ provides the codeword for each possible transition in the trellis for all the possible generator polynomials used in 802.11a and

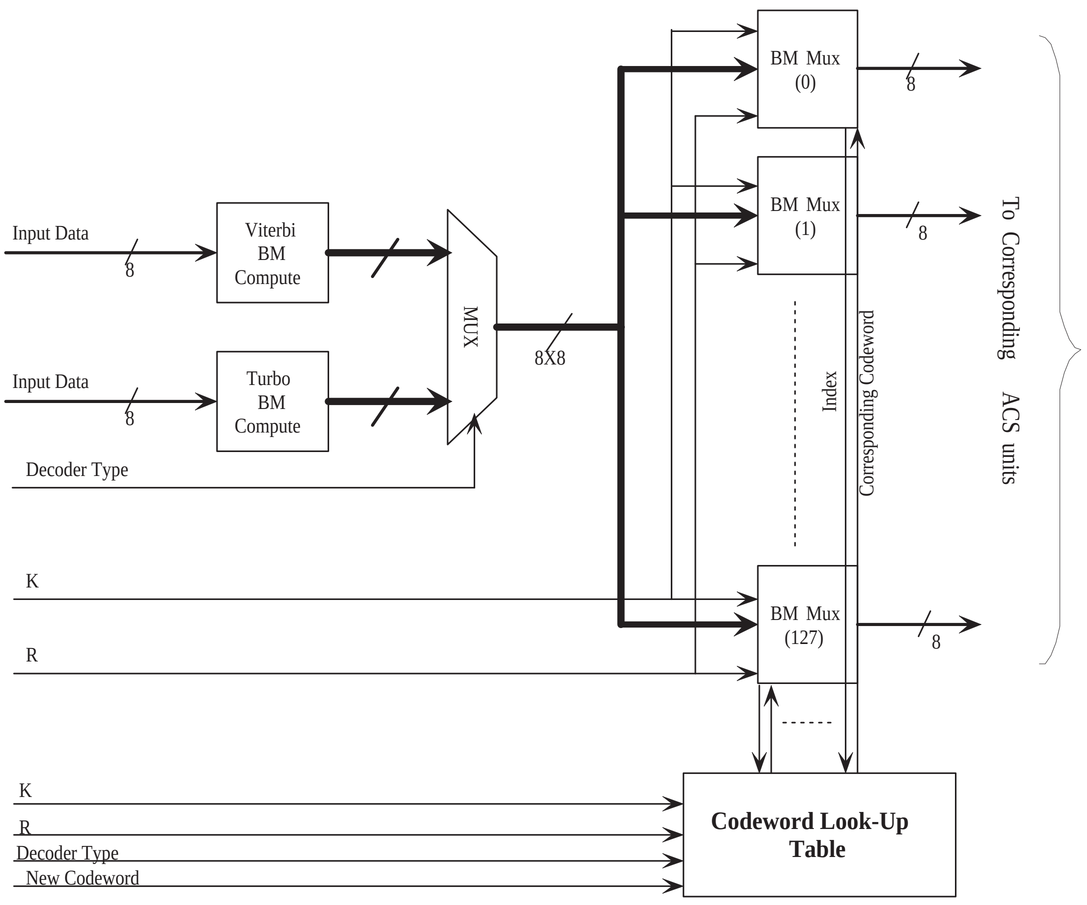
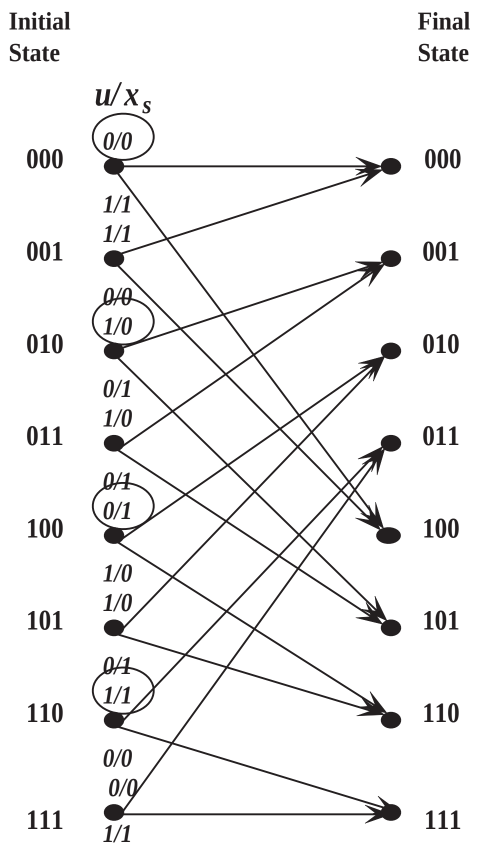


Figure 4.2 : Branch Metric Unit (BMU)

3G. Additional generator polynomials for constraint lengths 3-9 not mentioned in the two standards above have also been provided. Herein lies a very critical idea for reconfiguration of the system. A unique feature of this $Codeword\ Look-Up\ \ Table$ is that new codewords can be written to it, hence providing support for almost any generator polynomial in the constraint length range of 3 to 9. This feature could be used for enhanced security, though is not applicable to the scenario under review.

In order to better understand the workings of the BMU, we shall now delve deeper into the workings of the $Codeword\ Look-Up\ \ Table$. An example $Codeword\ Look-$



Index	LUT Data
0	00
1	10
2	01
3	11

Codeword Look-Up Table

Figure 4.3 : 3G Turbo state transitions and the correponding codeword LUT

$Up\ \ Table$ for 3G Turbo decoder (discussed earlier in Section 2.3.2). is shown in Figure 4.3. The number of elements in the LUT is contingent upon the constraint length and rate of coding. Equation 2.25 shows the computation of path metrics using the previous path metrics and branch metrics. The last three terms in the Eqn. 2.25, which form the branch metrics are dependent upon the values of u and x_s which are in turn dependent upon the constraint length and generator polynomial as shown in the Figure 4.3. The encircled numbers are critical to each butterfly and the knowledge of these allows us to compute the branch metrics for all the transitions in a particular butterfly. The $Codeword\ Look-Up\ \ Table$ is organized such that a certain $BMmux$

can obtain the relevant data from it using its index, as shown in Figures 4.3 and

4.4 Add Compare Select Unit

Once the requisite path metrics and branch metrics have been obtained, the *ACS* unit does the addition, comparision and selection of path metrics. The *ACS* unit takes in as inputs, the concerned path metrics, the concerned branch metrics, and outputs the survivor path metrics for Viterbi and Turbo decoding, and the decision bits for Viterbi and Turbo decoding. For the case of Turbo decoding, the difference between the path metrics as in Eqn. 2.27 is also outputed.

It is worth noting that for the case of traditional Viterbi decoding, the minimum of two path metrics is the survivor path metric. However, for the case of *SOVA* based Turbo decoding (Section 2.4.3), the maximum of the two competing path metrics forms the survivor path metric. Simply enough, if a maximum of two numbers is computed, the minimum is also available. The decision bit is evaluated according to the index of the state from which the survivor path metric originated.

It may be noted here that since K=4 for constituent Turbo encoders, four (2^{K-2}) *ACS* Units, specifically with indices from 0 to 3 have been programmed to do the ACS computations for both Viterbi and Turbo decoding, while the remaining ACS units(4 -127) are only used for computing Path Metrics and decision bits for Viterbi decoding, as shown in Figure 4.4.

4.4.1 Thresholder

The Thresholder is used to limit the ever accumulating path metrics that are increasing over time. This unit gets the path metrics outputted from the ACS unit, and compares all of them to a given threshold. Whenever, any path metric reaches a

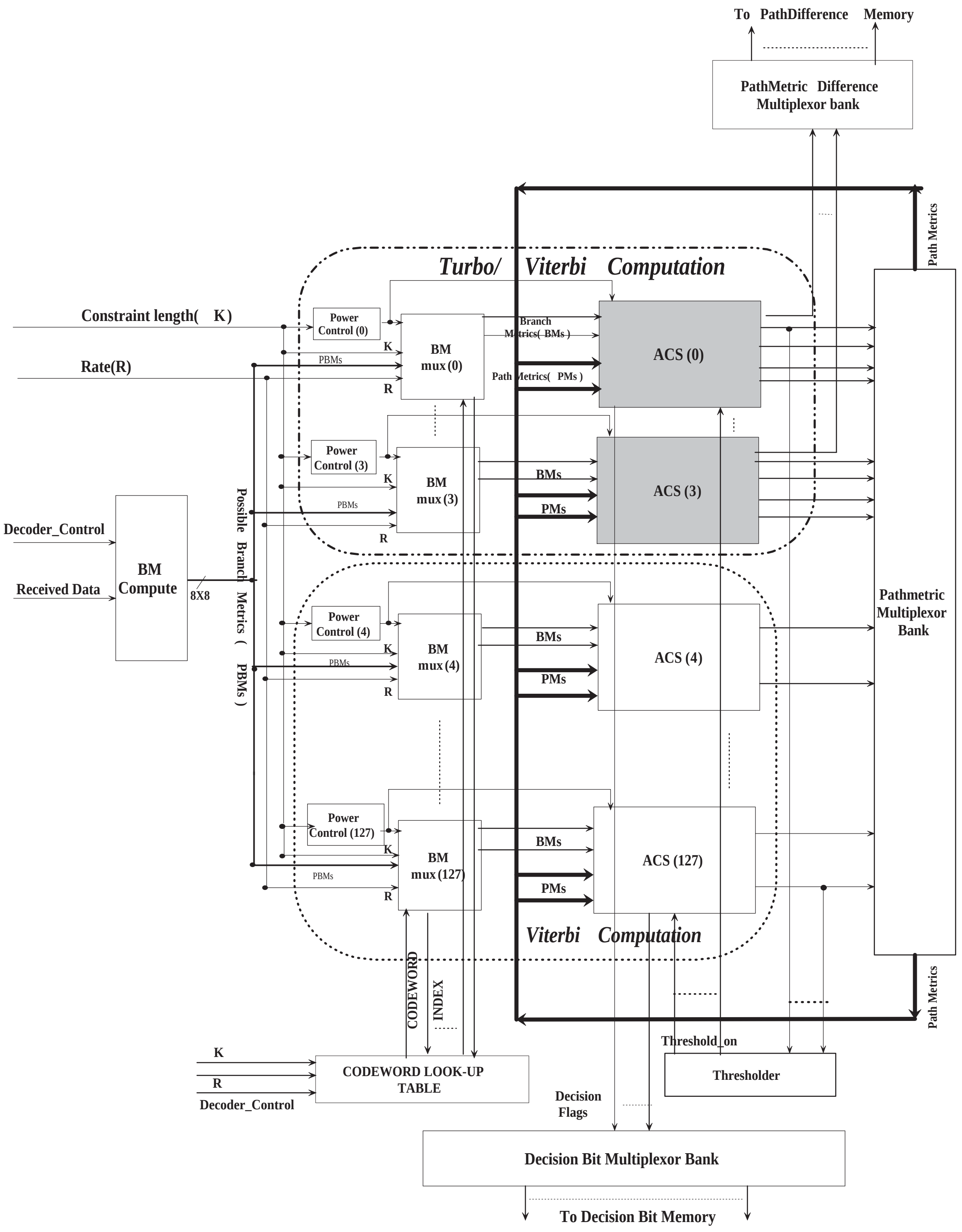


Figure 4.4 : The ACS, BMU and Multiplexer bank

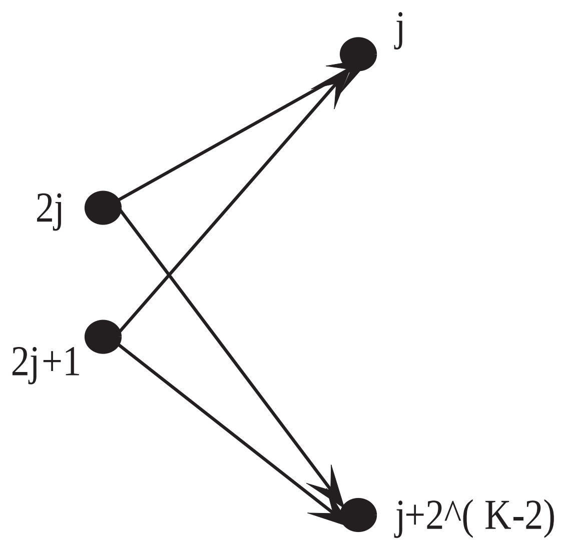


Figure 4.5 : j^{th} Butterfly for constraint length K

threshold, this unit sends a control signal to the ACS unit to truncate all the path metrics, hence providing protection against overflow.

4.5 Multiplexer Banks

4.5.1 Muxpathmetric

Figure 4.5 shows the butterfly structure for any given constraint length K code. Here, j corresponds to the index of butterfly unit. The j^{th} ACS unit operates on the jth butterfly structure. As is evident from the figure, the survivor path metrics computed from the j^{th} ACS unit, are fed back to a range of ACS Units (as shown by the termination states of the Butterflies) depending upon the constraint length in question. Interestingly, this problem is solved by using a bank of multiplexers, each multiplexer getting inputs from the outputs of different ACS units, and feeding an input of a particular ACS unit. It may be remembered that two path metrics are fed to ACS

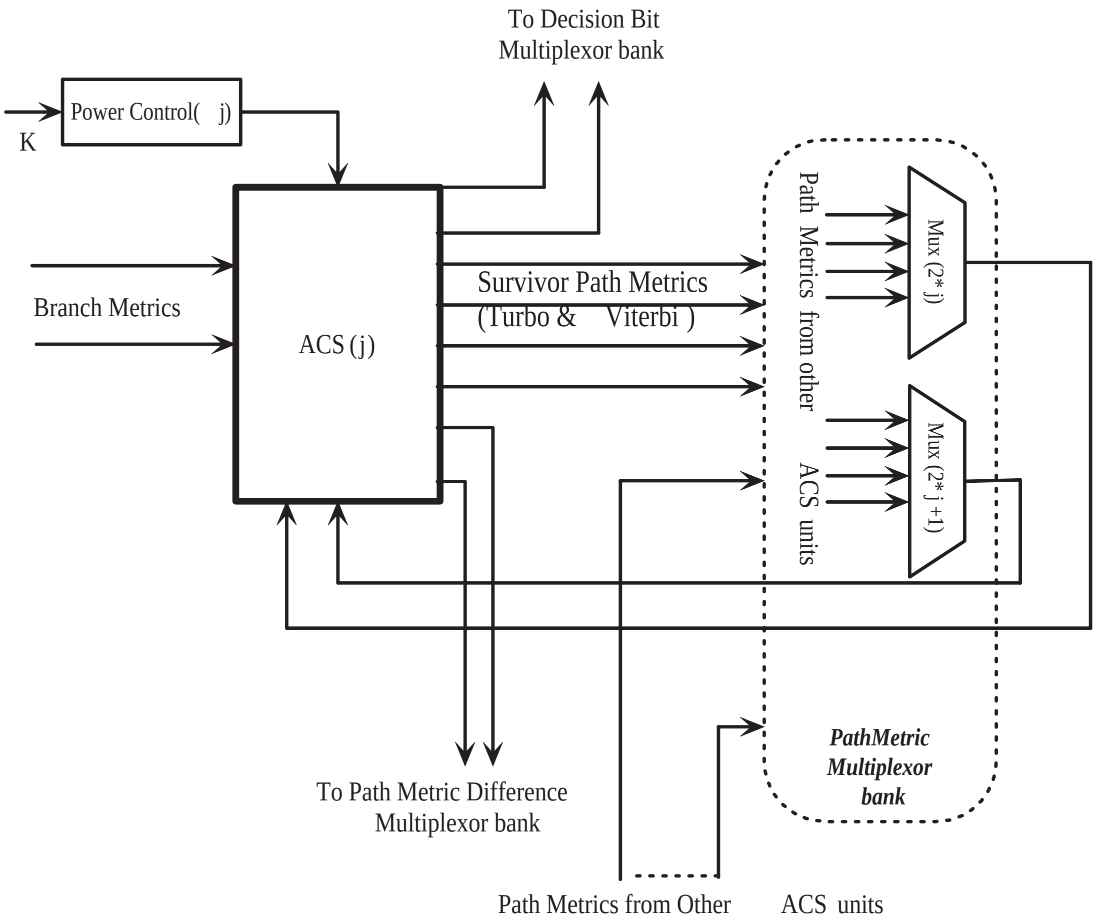

To Decision Bit Multiplexor bank
Power Control(j)
K
Branch Metrics
ACS (j)
Survivor Path Metrics (Turbo & Viterbi)
Path Metrics from other ACS units
Mux (2* j)
Mux (2* j +1)
PathMetric Multiplexor bank
To Path Metric Difference Multiplexor bank
Path Metrics from Other ACS units

Figure 4.6 : j^{th} ACS unit and Multiplexer bank interconnections

unit and hence, there are 2^{K-1} multiplexers, also referred to as Muxpathmetric. The multiplexers have inbuilt logic to route the path metrics according to their indices, decoding type, and constraint length being decoded. While 2 input multiplexers are sufficient for Viterbi decoding, 4 input multiplexers are used for Viterbi/Turbo decoding. This is another unit that is critical to the reconfigurability of the architecture, as it provides the flexibility to migrate from one constraint length to another.

4.5.2 Memory Mux

These multiplexers are similar to the Multiplexers for path metrics as defined earlier, but instead of feeding data to ACS units, they are used to reorder the decision bits that are being written to the memory.

4.6 Survivor Management

4.6.1 Flexible Traceback Unit for Viturbo

A block based approach is used for both Viterbi and Turbo decoding. In the case of Viterbi decoding, it is assumed that the trellis has been terminated at state zero, and hence traceback starts from the zeroth state, once the whole data block has been received. The flexible Traceback unit is another instance of the inherent flexibility in the proposed architecture. A very simple approach is used for this purpose. Consider the case of constraint length K code, where 2^{K-1} is the maximum number of states possible. Let us assume that the current state is C, the decision bit stored at this state is *decbit*, the state achieved on tracing back is P, and the decoded bit on Viterbi decoding is xv. Decision-LUT is a Look-Up table for various possible decisions for Turbo Decoding (for the case of 3G Turbo encoder), and xt is the Turbo decoded bit. Then

```
If $2*L \geq 2^{K-1}$ then
    $P$= $2*C$ + $decbit$ - $2^{K-1}$
    $xv$=1
else
    $P$ = $2*C$ + $decbit$
    $xv$=0
```

```
end if
    $xt$ = Decision-LUT($2 * C + decbit$)
```

4.6.2 Soft Decision Computation for Turbo Decoding

As mentioned earlier, traceback does not start until the whole block of data has been received. In the case of Turbo decoding, once the data has been received, the maximum path metric among all ending states is evaluated, and traceback starts from the state with the maximum path metric, and the path followed is the maximum likelihood path. This approach simplifies the memory management issues, at the cost of increased number of traceback logic units. Once the traceback starts, one soft decision per clock cycle is outputted.

4.6.3 Interleaving

A block based interleaver was implemented for Turbo decoding. Data is written in a matrix format, and the transpose of the matrix is the outputted as the interleaved data.

4.7 Architectural Power Control

Power consumption in an FPGA is given by the sum of the Quiescent power and the power consumed by all the switching elements. The power consumed by each switching element of the design is given by: $P = C * V^2 * E * F$
where

P = Power Consumption in mW,

C= Capacitance in Farads,

V= Volts,

E= Switching activity(number of transitions per clock cycle), and

F= Frequency in Hz.

In the above formula, the product $E * F$ is the most variable element and is under the designer's control up to a certain extent. In our design, most of the computations are pipelined, and triggered by the rising edge of the clock. When the clock input to a certain unit is reduced to zero frequency, that particular unit is effectively shut down. Also, if a signal is not allowed to toggle, it is effectively consuming no power.

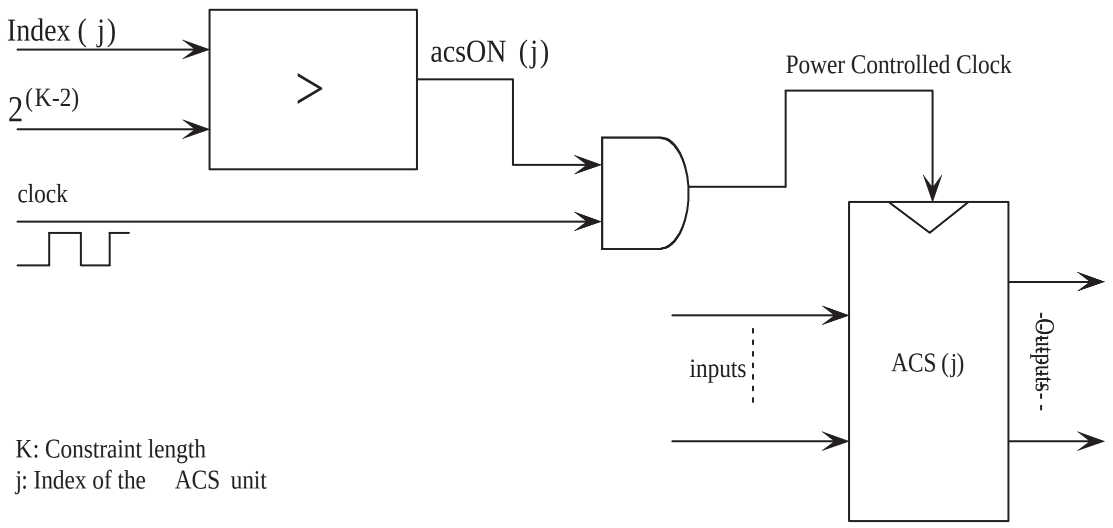


Figure 4.7 : Power control architecture for j^{th} ACS unit

As discussed earlier, we have a fully parallel architecture for constraint length 9. For the case of constraint length 7, not all the units of the circuit are being utilized. For example only 1/4 of the ACS units are being actively used for constraint length

7 computation. In the proposed architecture, a power control mechanism has been devised in order to limit power consumption to only those units that are in operation. The ACS units' power control mechanism is shown in the Figure 4.7. The mechanism works as follows:

```
    IF j $\leq 2^{K-2}$ then
acsON(j) =1
else
acsON(j) =1
end IF
```

The acsON(j) signal ANDs with the global clock, and as it is obvious, if the acsON(j) is equal to 0, the ACS is turned off. Similarly, the BMmuxes also are power controlled, and hence the input to a certain ACS unit that is not in use is not allowed to toggle. Thus, neither the inputs, nor the outputs of the unit can toggle, and hence cant consume power. Similarly, several other units, for example the Survivor management unit for Turbo decoding, are powered down when not in use.

Chapter 5

Implementation and Results

Field Programmable Gate Arrays(FPGA) provide a very comprehensive and flexible prototyping environment for hardware design. For this reason, VITURBO was implemented on a Xilinx Virtex-II 2000K gate FPGA. Very High Speed Integrated Circuits Hardware Description Language (VHDL) was used to describe the architecture.

5.1 FPGA Architecture

A typical FPGA consists of three major components: Configurable Logic, Programmable Interconnect, and Input/Output Blocks (IOBs). In the case of Xilinx Virtex II FPGAs, the internal Configurable logic includes four major elements: Configurable Logic Blocks, Block SelectRAM, Multiplier blocks, and Digital Clock Manager. The Configurable Logic Blocks provide functional elements for combinatorial and synchronous logic, including storage elements. Block SelectRAM memory modules provide large 18 Kbit dual-port Random Access Memories. 18x18 bit dedicated multipliers form the multiplier blocks. A self calibrating, fully digital solution for clock distribution delay compensation, clock multiplication and division is integrated in the Digital Clock Managers. Programmable Routing resources interconnect all these elements. Every programmable element is tied to a switch matrix, which allows multiple connections to the general routing matrix, where the General Routing Matrix is an array of routing switches. All the programmable elements including the Programmable Interconnect, are controlled by values stored in Static Memory cells. These values are loaded into the memory cells during configuration and can be reloaded to change the configuration of programmable elements as many times as required.

The XC2V2000 FPGA has an array of 56X48 CLBs, (where each CLB is made up of 4 slices) 56 eighteen bit multiplier blocks, and 56 X 18000 bits of SelectRAM memory. Up to 624 I/O pads are possible for XC2V2000 FPGA, which on the whole is made up of 2 million system gates.

5.2 Development Environment

Xilinx's Integrated Software Environment (ISE) was chosen for the design and implementation of the proposed architecture. The first step in the design was the use of VHDL for the description of the architecture. Synopsis' Synplicity was used to compile and synthesize the design. RTL level schematics were analyzed after compilation, using the graphical interface provided by Synplicity. Behavioral level simulations were then performed using Modelsim, provided by Mentor Graphics. Upon completion, the design goes through the different steps of implementation which include Translation, Map, Place and Route, and finally generation of Programming file, which can be downloaded to the FPGA. Power simulations and analysis were done using Xilinx's Xpower tool, which uses the Value Change Dump (*.vcd) files generated by ModelSim simulator during the Post-Place& Route VHDL model simulation.

5.3 VITURBO Implementation Issues

The implemented architecture can decode constraint length 3-9 convolutional codes, and also constraint length 4 Turbo codes. VHDL was used to describe the architecture, and it was implemented on Xilinx Virtex-2 FPGAs. It must be noted here that the gate count and maximum clocking frequency of a circuit implemented on an FPGA, are inferior, when compared to an Application Specific Integrated Circuit (ASIC) implementation, and hence care must be taken to evaluate the given data, in comparison with ASICs.

5.3.1 Area-Time Analysis

Table 5.1 shows the various area- time- decoder type tradeoffs involved in the proposed reconfigurable design. The gate counts for Logic and Memory have been separated in order to give an in-depth analysis of the architecture. Each of the architectures listed in the table was implemented independently on an FPGA. As expected, the last architecture with the maximum decoding capability is the largest in area. One thing to note is that the area does not increase as an exponential function of the constraint length. The reason is that even though the number of ACS units(and hence the area associated with them) increases exponentially (as a power of two) with the constraint length, the area required by other units like BMU, SMU and Control Unit are not dramatically affected by the constraint length. Also, the number of Input/Output pads used by the architecture does not change.

The maximum operating frequency (Figure 5.1) of the circuit decreases with increasing constraint lengths for Viterbi decoding. The ACS computations and routing of path metrics form the critical path and as the constraint length increases, the path metrics have to be routed across more complicated and longer paths. For constraint length 3, data transfer takes place between only 2 ACS units. However for constraint length 9, internal data transfer takes place between 128 ACS units.

Another interesting observation that can be made from Table 5.1 is that the maximum operating frequencies for standalone constraint length 9 Viterbi decoder and constraint length 4 Turbo decoder are higher, compared to the reconfigurable VITURBO decoder comprising of constraint length 3-9 Viterbi and constraint length 4 Turbo decoder. This difference in the maximum operating frequency is mainly due to the addition multiplexer banks to the critical path of ACS for VITURBO. These multiplexers make the reconfiguration possible, at the cost of increased design complexity.

Decoder Type	Gates(Logic)	Gates(Memory) 128 bit frame	Max. Frequency
Viterbi (7)	62,987	65536	71.3 MHz
Viterbi (9)	166,348	262,146	68.4MHz
Turbo	37,812	65536	67.7 MHz
Viterbi (3-5)	33,487	16,384	64.6 MHz
Viterbi (3-5)+ Turbo(4)	42,385	81,812	63.3 MHz
Viterbi (3-7)	67,042	65,536	62.8 MHz
Viterbi (3-7)+ Turbo(4)	76,037	131,072	62.1 MHz
Viterbi (3-9)	181,560	262,146	61.9 MHz
VITURBO: Viterbi (3-9)+Turbo(4)	190,288	327,680	60.5 MHz

Table 5.1 : Architectural Tradeoffs (Area/ Frequency)

As seen in Table 5.1, the logic area requirement for a reconfigurable VITURBO decoder (constraint length 3 -9 Viterbi decoding and Turbo decoding) is nearly 23,940 gates (14% gates) more than a standalone constraint length 9 Viterbi decoder. Comparing a reconfigurable constraint length 3-9 Viterbi decoder with a reconfigurable decoder with constraint length 3-9 Viterbi decoding and constraint length 4 Turbo decoding capabilities, we see that the area overhead of the latter from the former is only 8,728 gates (5% gates). This demonstrates how the common aspects of the the various decoders have been exploited to design the proposed reconfigurable architecture.

The memory area requirements, as seen in the Table 5.1 are directly proportional to frame length being used, which in our case is 128. Also, the requirements increase exponentially with the constraint length (as a power of two). This trend is visible from the memory requirements for different Viterbi decoder realizations. However, for a given constraint length, Turbo decoding memory requirements are larger than Viterbi decoding, because of the soft decision computation requirements.

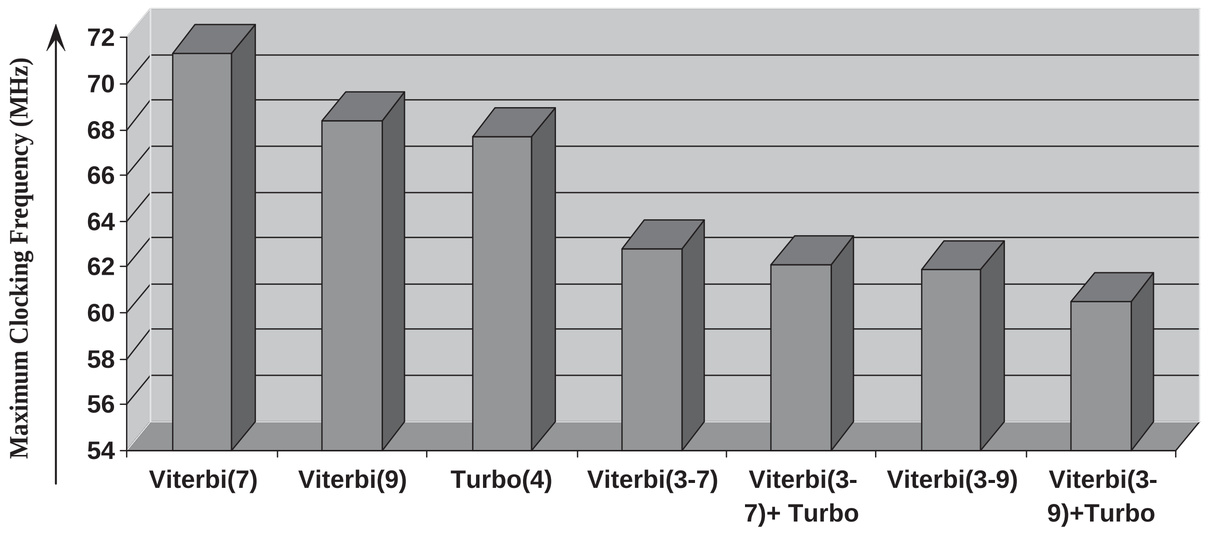


Table 5.2 : Maximum clocking frequencies for different decoder realizations

5.3.2 Throughput Analysis

With the architecture described above, the throughput of Viterbi decoding is limited by the fact that we have used a block based approach. However, a way past this bottleneck is a pipelined decoding architecture as shown in Figure 7. In this case, two memory banks are used for Survivor memory and two memory banks are used for Order Inversion memory. It may be recalled that output is initially produced in the inverted order when using the block based approach, and hence Order Inversion memory is required to buffer the output data, and output in correct order. The two additional memory banks increase the throughput of the circuit, as data is written in alternate memory , on every new input block. Throughput up to 1bit/clock cycle is obtained, with an initial latency of $2 * N + 3$ clock cycles, where N is the block length.

However, in the case of Turbo decoding, 4 iterations are usually performed for reliable performance. In the given architecture, only one SOVA decoder has been implemented, and hence the throughput is limited by the fact that the decoder will be used for up to 8 times in order to do 4 complete iterations of Turbo decoding, as each iteration involves the usage of the decoder twice. For each iteration, the given SOVA based decoder has a latency of $2 * N + 8$ clock cycles, where N is the block size. Hence, a hefty $8 * (2 * N + 8)$ cycles are required for complete Turbo decoding. For a frame of length 100 or greater, this data rate is nearly 3.5 Mbps (Table 5.3), well above the current 3G standards.

5.3.3 Power Consumption Analysis

Table 5.3 shows the power consumption of the different configurations of the VI-TURBO architecture. It is clear from the data that while reconfigurable architectures provide enhanced flexibility, the power consumption of the circuit is not compromised. Different configurations of the circuit are capable of powering down units which are not used for those configurations. We have introduced a new metric for power consumption, Joules/bit which gives the power consumed for each bit processed. This

Decoder Type	Clock Frequency	Data Rate	Power Consumption (sans Quiescent Power)	Energy/Bit (Joules/bit)	Quiescent Power
Viterbi (K=5)	54 MHz	54 Mbps	138.06 mw	2.55 nJ	225 mW
Viterbi (K=5)	2 MHz	2 Mbps	5.11 mw	2.55 nJ	225 mW
Viterbi (K=7)	54 MHz	54 Mbps	501.3 mw	9.27 nJ	225 mW
Viterbi (K=7)	2 MHz	2 Mbps	18.5 mw	9.27 nJ	225 mW
Viterbi (K=9)	54 MHz	54 Mbps	1.42 w	26.96 nJ	225 mW
Viterbi (K=9)	2 MHz	2 Mbps	59.54 mw	26.96 nJ	225 mW
Turbo(K=4)	54 MHz	3.15 Mbps	165.02 mw	52.38	225 mW
Turbo(K=4)	34.3 MHz	2 Mbps	104.76 mw	52.38	225 mW

Table 5.3 : Power Consumption for different configurations of VITURBO

enables us to get a more accurate look at the relative computational complexity of each decoding algorithm. Turbo decoding is computationally very intensive, as is obvious from the table.

Chapter 6

Conclusions

Reconfigurable architectures have the potential to provide the speed and versatility required to realize receiver structures for future ubiquitous wireless networks. Versatility is a key word in the design of such receivers, as different wireless communication standards would require completely different receivers, if reconfiguration was not possible. This work demonstrates the idea of reconfigurable architectures with reference to channel decoding algorithms in various wireless communication standards.

The proposed reconfigurable architecture has the flexibility to decode a very wide range of convolutionally coded data, with the capability to reconfigure at run-time. Viterbi decoding and SOVA based Turbo decoding can be realized with this architecture, which has been aptly named 'Viturbo'. Constraint length 3-9, rate 1/2 -1/3 convolutionally coded data and constraint length 4, rate 1/3 Turbo coded data can be decoded with this architecture. The generator polynomials used are the ones that maximize the free distance in each case. However, this architecture provides a unique feature for decoding data coded using *any* other generator polynomials, via the use of a programmable Codeword Look Up Table.

We have employed a completely parallel approach which has led to a high speed architecture that can provide throughputs in the range of 60 Mbps for constraint length 3-9 Viterbi decoding and 3.5 Mbps for SOVA based Turbo decoding (4 iterations). It has been demonstrated in this architecture that with a 5% overhead in area (excluding memory), a constraint length 3-9 Viterbi decoder can support Turbo decoding.

Thus, even while providing extreme flexibility, this architecture does not compromise on the area consumption. The system has various architectural power control units that contribute to huge power savings by shutting down parts of the circuit that may not be useful for a certain decoding type in progress. Overall, this architecture provides a completely flexible decoding architecture with very high throughputs and hence can be used in most contemporary wireless communication systems.

Chapter 7

Future Work

Viturbo has immense applications in the field of ubiquitous wireless networks, which are yet to take shape. An improvement over this system would be the use of log-MAP decoding algorithm for Turbo decoding, which has been shown to be superior to SOVA decoding algorithm. In order to lower the power consumption, termination algorithms for Turbo decoding can be used. The proposed architecture with its immense parallelism has the potential to support higher data rate Turbo decoding, by providing more Survivor Management units. Only 4 out of 128 ACS units have been employed for Turbo decoding, but potentially all of them could be used. This would mean an immensely parallel architecture, with throughputs up to one output per clock cycle (nearly 60 Mbps with current technology) Such a decoder could find use in some recently proposed modifications of WLAN, which support data rates of up to 54 Mbps using Turbo coding as the channel coding algorithm. Potentially, the proposed architecture could also be modified to support both data and voice communications for 3G system simultaneously, by multiplexing the processing in time. Also, an ASIC implementation of the proposed architecture would give us more specific area, time, and power data for a given fabrication technology.

While this reconfigurable architecture has been proposed for channel decoding algorithms, commonalities in other baseband algorithms need to be exploited in order to design a complete reconfigurable architecture for wireless networks.

Bibliography

[1] C. Berrou, A. Glavieux, and P. Thitimasjshima, "Near Shannon Limit error correcting coding and decoding: Turbo-Codes(1)," *IEEE International Conference on Communications,* Geneva, Switzerland, May 1993, pp. 1064-1070.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate" *Intl Symposium on Information Theory*Asilomar, CA, Jan 1972, p. 90.

[3] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft decision outputs and its applications," *IEEE Globecom Conference* Dallas, TX, USA, Nov. 1989, pp. 1680-1686.

[4] J. Hagenauer; E. Offer, L. Papke, P. Hoeher, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory,* Mar. 1996, Vol. 42, No. 2, pp. 429-445.

[5] "Seamless Multitier Wireless Networks for Multimedia Applications, " Rice University, http://cmc.rice.edu/research/RENE'.

[6] O. Joeressen, M. Vaupel, and H. Meyr, "High-Speed VLSI Architectures for Soft-Output Viterbi Decoding," *International Conf. on Application Specific Array Processors,*1992, pp. 373-384.

[7] A. Ghrayeb, W.E. Ryan, "Performance of High Rate Turbo Codes Employing

the Soft-Output Viterbi Algorithm(SOVA)" *Thirty Third Asilomar Conference on Signals, Systems, and Computers,*1999, vol 2, pp. 1665 -1669.

[8] M. Bickerstaff, D. Garrett et. al, "A unified Turbo/Viterbi Channel Decoder for 3 GPP Mobile Wireless in 0.18 μm CMOS" *IEEE Solid State Circuits Conference,* 2002, pp. 125-441.

[9] D. Hocevar, A. Gatherer, "Architecture selection for a low power flexible Viterbi Decoder," *IEEE International Conference on 3rd Generation Wireless Communications,*2000, vol. 5, pp. 2257-2264.

[10] P. Elias, "Coding for Noisy Channels," *IRE Conv. Record,*1955, vol 4, pp. 43-47.

[11] M. Jezequel and P. Penard, "Turbo4: A high bit-rate chip for Turbo code encoding and decoding," *IEE Colloquium on Turbo Codes in Digital Broadcasting - Could It Double Capacity?* 1999, pp. 4/1 -4/5.

[12] B. Sklar, "A Primer on Turbo Code Concepts," *IEEE Communications Magazine,* Dec. 1997, vol. 35, pp. 94-102.

[13] K. Chadha and J. Cavallaro, "A reconfigurable Viterbi Decoder Architecture," *Thirty Fifth Asilomar Conference on Signals, Systems and Computers,*2001, vol. 1, pp. 66-71.

[14] J. G. Proakis, "Digital Communications", McGraw Hill , New York 1995.

[15] C. Heegard and S.B. Wicker, "Turbo Coding," Kluwer Academic Publishers, MA 1999.

[16] Ushirokawa, Okamura, Kamiya, and Vucetic, "Principles of Turbo Codes and their Application to Mobile Communications," *IEICE Transactions,* July 1998.

[17] G.D. Forney, "The Viterbi Algorithm," *Proc. of IEEE,*Mar. 1973, vol 61, pp. 268-278.

[18] M. Vaya and N. Ahmed, "Turbo Decoding on TMS320C6211 Digital Signal Processor," *DSP Laboratory Project Report, Rice University,*2001.

[19] P. H. Kelly and P.M. Chau, "A flexible constraint length, foldable Viterbi Decoder," *IEEE Global Telecommunications Conference GLOBECOM,*1993, pp. 631-635.

[20] J. Hagenauer, "Iterative decoding of Binary block and convolutional codes," *IEEE Transactions on Information Theory,*Mar. 1996, pp. 429-445.

[21] Zhongfeng Wang, H. Suzuki, and K. K. Parhi, "VLSI implementation issues of Turbo decoder design for wirless applications," *IEEE Workshop on Signal Processing Systems,*1999, pp 503-512.

[22] J. Steensma and C. Dick, "FPGA implementation of a 3GPP Turbo deocder," *Thirty-Fifth IEEE Asilomar Conference on Signals, Systems and Computers,* 2001, vol.1, pp. 61 -65