©gopalax -International Journal of Technology And Engineering System(IJTES):
Jan –March 2011- Vol.2.No.2.

Implementation of encoder and decoder for Turbo codes

[1]Rutuja D. Deshmukh and [2]Prof. D.M.Meshram

Department of Electronics, Priyadarshini college of engineering, Nagpur, India.
[1]*rutuja1502@yahoo.com*., [2]*divyameshram@gmail.com*

Abstract— **Error correcting coding (ECC) is a critical part of modern communication systems, where it is used to detect and correct errors introduced during a transmission over a channel. It relies on transmitting the data in encoded form, such that the redundancy introduced by the coding allows a decoding device at the receiver to detect and correct errors. In information theory, turbo codes are a class of high-performance forward error correction (FEC) codes developed in 1993, which were the first practical codes to closely approach the channel capacity, a theoretical maximum for the channel noise at which reliable communication is still possible. Turbo codes are finding use in (deep space) satellite communications and other applications where designers seek to achieve reliable information transfer over bandwidth or latency constrained communication links. Turbo codes are nowadays competing with LDPC codes, which provide similar performance. It is theoretically possible to approach the Shannon limit by using a block code with large block length or a convolutional code with a large constraint length. The processing power required to decode such long codes makes this approach impractical. Turbo codes overcome this limitation by using recursive coders and iterative soft decoders.The recursive coder makes convolutional codes with short constraint length appear to be block codes with a large block length, and the iterative soft decoder progressively improves the estimate of the received message. The turbo encoder and decoder is based on convolution constituent codes which will outperform all other Forward Error Correction techniques. The turbo decoder can be modified easily to fit any size for advanced communication system-on-chip product. Therefore it has been adopted by important broadband communication applications and standards such as DVB-RSC (Digital Video Broadcast Return Channel Satellite) and the 3G wireless communication.**

***Keywords*-Turbo Codes, Error Correcting Code, Encoder, Decoder**

I. INTRODUCTION

Coding theorists have traditionally attacked the problem of designing good codes by developing codes with a lot of structure, which lends itself to feasible decoders, although coding theory suggests that codes chosen at random should perform well if their block sizes are large enough. The challenge to find practical decoders for almost random, large codes has not been seriously considered until recently. Perhaps the most exciting and potentially important development in coding theory in recent years has been the dramatic announcement of turbo codes by Berrou et al. in 1993. The announced performance of these codes was so good that the initial reaction of the coding establishment was deep skepticism, but recently researchers around the world have been able to reproduce those results. The introduction of turbo codes has opened a whole new way of looking at the problem of constructing good codes and decoding them with low complexity . Turbo codes achieved near-Shannon-limit error correction performance with relatively simple component codes and large interleavers. A required $Eb=N0$ of 0.7 dB was reported for a bit-error rate (BER) of 10.5 for a rate 1/2 turbo code . Multiple turbo codes (parallel concatenation of $q > 2$ convolutional codes) and a suitable decoder structure derived from an approximation to the maximum a posteriori (MAP).

II. PARALLEL CONCATINATION OF CONVOLUTION CODES

The codes considered in this article consist of the parallel concatenation of multiple (q ¸ 2) convolutional codes with random interleavers (permutations) at the input of each encoder. This extends the original results on turbo codes reported which considered turbo codes formed from just two constituent codes and an overall rate of 1/2. Figure 1 provides an example of parallel concatenation of three convolutional codes. The encoder contains three recursive binary convolutional encoders with $m1$, $m2$, and $m3$ memory cells, respectively. In general, the three component encoders may be di®erent and may even have di®erent rates. The first component encoder operates directly on the information bit sequence **u** of length N, producing the two output sequences **x**0 and **x**1. The second component encoder operates on a reordered sequence of information bits, **u**2, produced by a permuter (interleaver), ¼2, of length N, and outputs the sequence **x**2. Similarly, subsequent component encoders operate on a reordered sequence of information bits. The interleaver is a pseudorandom block scrambler defined by a permutation of N elements without repetitions: A complete block was read into the the interleaver and read out in a specified (fixed) random order. The same interleaver was used repeatedly for all subsequent blocks. Figure 1 shows an example where a rate r = 1=n = 1=4 code is generated by three component codes with memory m1 = m2 = m3 = m = 2, producing the outputs **x**0 , **x**1, *g*1,*g*0, **x**2, **x**3 , where the generator polynomials *g*0 and *g*1 have octal representation . Note that various code rates can be obtained by proper puncturing of **x**1, **x**2, **x**3, and even **x**0. We used the encoder in Fig. 1 to generate an (n(N + m);N) block code, where the m tail bits of code 2 and code 3 are not transmitted. Since the component encoders are recursive, it is not sufficient to set the last m information bits to zero in order to drive the encoder to the all-zero state, i.e., to terminate the trellis. The termination (tail) sequence depends on the state of each component encoder after *N* bits, which makes it impossible to terminate all component encoders with m predetermined tail bits. This issue, which had not been resolved in the original turbo code implementation, can be dealt with by applying a simple method described that is valid for any number of component codes. A design for constituent convolutional codes, which are not necessarily optimum convolutional codes, was originally reported in for rate 1=*n* codes.

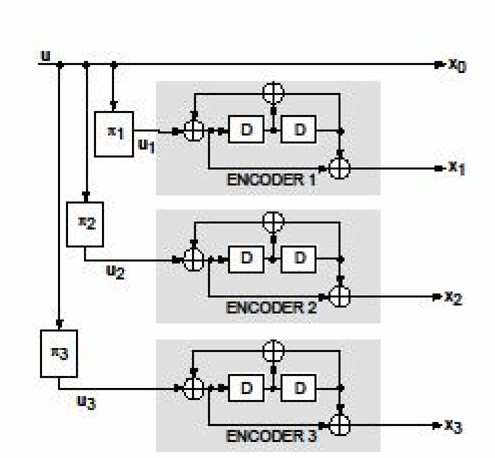


Figure 1: Example of encoder with three codes.

II. DESIGN OF CONSTITUENT ENCODERS

Maximizing the weight of output codewords corresponding to weight-2 data sequences gives the best BER performance for a moderate bit signal-to-noise ratio (SNR) as the random interleaver size *N* gets large. In this region, the dominant term in the expression for bit error probability of a turbo code with *q* constituent encoders is

$$P_b \approx \frac{\beta}{N^{q-1}} Q\left(\sqrt{2r\frac{E_b}{N_0}\left(\sum_{j=1}^{q} d^p_{j,2} + 2\right)}\right)$$

where dp j;2 is the minimum parity-weight (weight due to parity checks only) of the codewords at the output of the *j*th constituent code due to weight-2 data sequences, and ¯ is a constant independent of *N*. Define *dj;*2 = *dpj;*2 +2 as the minimum output weight including parity and information bits, if the *j*th constituent code transmits the information (systematic) bits. Usually one constituent code transmits the information bits (*j* = 1), and the information bits of others are punctured. Define *def* = $\sum^{q}_{j=1}$ *dp j;*2 +2 as the effective free distance of the turbo code and 1=*Nq*¡1 as the interleaver's gain. We have the following bound on *dp* 2 for any constituent code.

III. INTERLEVER DESIGN

The random interleaver uses a fixed random permutation and maps the input sequence according to the permutation order. The length of the input sequence is assumed to be L. Figure 2 shows a random interleaver with L=8.

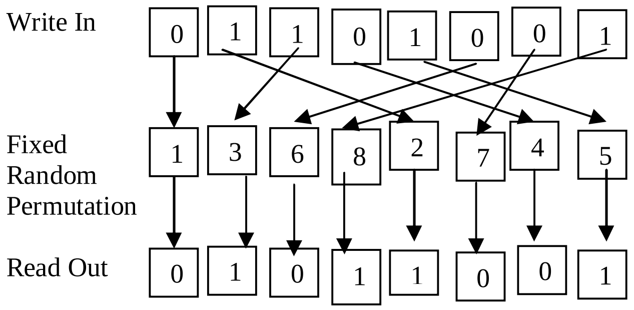


Figure 2: A Pseudo random Interlever with N=8

From Figure 2, the interleaver writes in [0 1 1 0 1 0 0 1] and reads out [0 1 0 1 1 0 0 1].

IV VITERBI DECODER

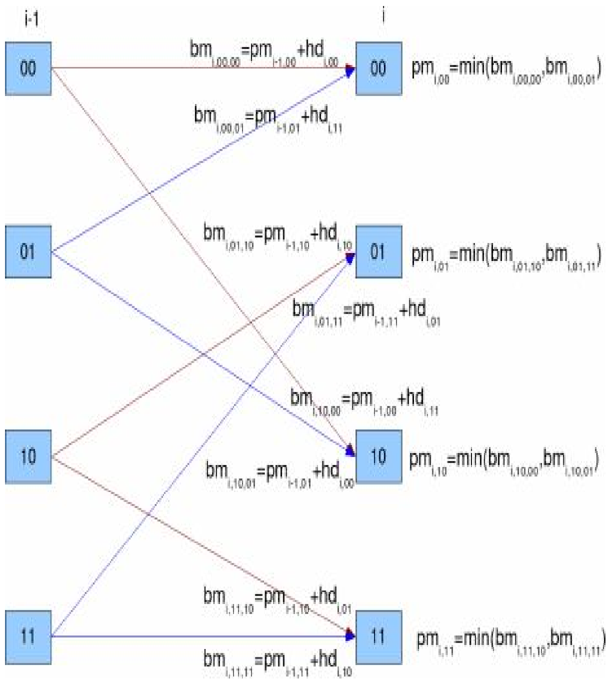
This Viterbi decoding algorithm was for a simple Binary Convolutional Code with rate 1/2, constraint length K=3. For optimal decoding for a modulation scheme with memory (as is the case here), if there are N coded bits, we need to search from 2^N possible combinations. This became prohibitively complex as N becomes large.

(1) Though we reached each state from 2 possible states, only one of the transition in valid.

(2) We found the transition which is more likely (based on the received coded bits) and ignore the other transition.

(3) The errors in the received coded sequence are randomly distributed and the probability of error is small.

Based on the above assumptions, the decoding scheme proceed as follows: Assume that there are N coded bits. Take two coded bits at a time for processing and compute Hamming distance**,** Branch Metric**,** Path Metric and Survivor Path index for (N/2)+K-1 times. Let i be the index varying from 1 till (N/2)+K-1.



i-1
i
00
01
10
11
bm_{i,00,00}=pm_{i-1,00}+hd_{i,00}
bm_{i,00,01}=pm_{i-1,01}+hd_{i,11}
pm_{i,00}=min(bm_{i,00,00},bm_{i,00,01})
bm_{i,01,10}=pm_{i-1,10}+hd_{i,10}
pm_{i,01}=min(bm_{i,01,10},bm_{i,01,11})
bm_{i,01,11}=pm_{i-1,11}+hd_{i,01}
bm_{i,10,00}=pm_{i-1,00}+hd_{i,11}
bm_{i,10,01}=pm_{i-1,01}+hd_{i,00}
pm_{i,10}=min(bm_{i,10,00},bm_{i,10,01})
bm_{i,11,10}=pm_{i-1,10}+hd_{i,01}
bm_{i,11,11}=pm_{i-1,11}+hd_{i,10}
pm_{i,11}=min(bm_{i,11,10},bm_{i,11,11})

Figure 3: **Branch Metric and Path Metric computation for Viterbi decoder**

A. Traceback Unit

Once the survivor path is computed (N/2)+K-1 times, the decoding algorithm can start trying to estimate the input sequence. Thanks to presence of tail bits (additional K-1 zeros) , it is known that the final state following convolution codes is State 00.

	i/p if previous state			
current state	**00**	**01**	**10**	**11**
00	0	0	x	x
01	x	x	0	0
10	1	1	x	x
11	x	x	1	1

Table 1: Input given current state and previous state

V. OBSERVATIONS

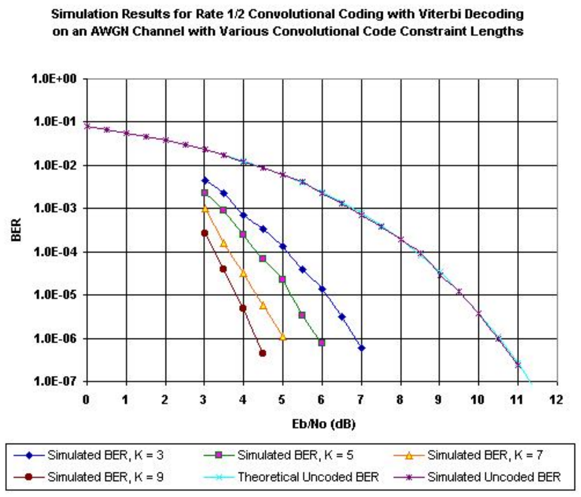
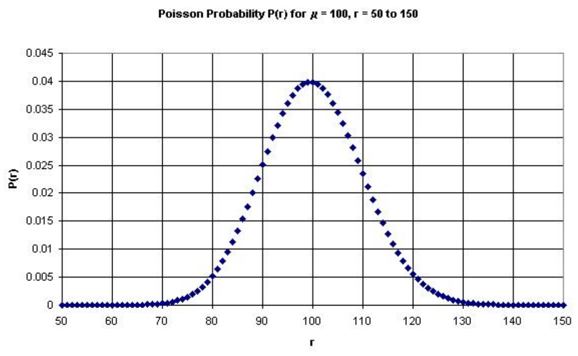


Figure 4: BER plot for BPSK modulation in AWGN channel with Binary Convolutional code and hard decision Viterbi decoding

(1) I obtained the results shown in this chart using the example simulation code, with the trellis depth set to Kx 5, using the adaptive quantizer with three-bit channel symbol quantization. For each data point, I ran the simulation until 100 errors (or possibly more) occurred. With this number of errors, I have 95% confidence that the true number of errors for the number of data bits through the simulation lies between 80 and 120.

(2) Notice how the simulation results for BER on an uncoded channel closely track the theoretical BER for an uncoded channel, which is given by the equation P(e) = 0.5 * rfc(sqrt(Eb/N0)) = Q(sqrt(2Eb /N0). This validates the uncoded BER algorithm and the Gaussian noise generator. The coded BER results appear to agree well with those obtained by others.

(3) Error events occur as a Poisson process, a random sequence of events in time. The Poisson process has a mean rate _ equal to n/t, where n is the number of events (the number of errors, in this case) and t is the time interval of the measurement. For the purposes of the simulation, let's let t = the total number of bits in the simulation. Let's say we measure 100 errors in 100,000 bits. The rate is thus 100/100,000, or 1 x 10-3. If we set up the simulation to run for 100,000 bits, then the mean μ of the Poisson distribution is t, or 100 errors. The formula for the probability of an expected number r of errors, given a mean of μ errors, is μ So the Poisson distribution for 50 to 150 errors,given a mean of 100 errors, is illustrated in the chart below:



.The cumulative probability of the above distribution for the range of 80 to 120 errors is actually 95.99%(approximately).

REFERENCES

[1] Clark, G. C. Jr. and J. Bibb Cain., Error-Correction Coding for Digital Communications, New York, Plenum Press, 1981.

[2] Gitlin, Richard D., Jeremiah F. Hayes, and Stephen B. Weinstein, Data Communications Principles, New York, Plenum, 1992.

[3] Heller, J. A. and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communication," IEEE Transactions on Communication Technology, Vol. COM-19, October 1971, pp 835–848.

[4] Yasuda, Y., et. al., "High rate punctured convolutional codes for soft decision Viterbi decoding," IEEE Transactions on Communications, vol. COM-32, No. 3, pp 315–319, Mar. 1984.

[5] Haccoun, D., and G. Begin, "High-rate punctured convolutional codes for Viterbi and sequential decoding," IEEE Transactions on Communications, vol. 37, No. 11, pp 1113–1125, Nov. 1989.

[6] G. Begin, et.al., "Further results on high-rate punctured convolutional codes for Viterbi and sequential decoding," IEEE Transactions on Communications, vol. 38, No. 11, pp 1922–1928, Nov. 1990.