ITU-Telecommunication Standardization Sector Temporary Document RN-028

STUDY GROUP 15 Original: English

Red Bank, NJ 21-25 May 2001

Question: 4/15

SOURCE[1]: *AT&T.*

TITLE: G.gen: G.dmt.bis: G.lite.bis: : Decoder Structure of Multi-Tone Turbo Trellis Coded Modulation Proposal

---

Abstract

This contribution proposes to use Multi-Tone Turbo Trellis Coded Modulation (MTTCM) technique as an option for the forward error correction in ADSL modems. The contribution describes the decoder structure and includes a recommendation for the decoding of the proposed Turbo code.

---

* Contact: Hamid R. Sadjadpour, Tel:(973)236-6807, Email:sadjadpour@research.att.com;

I. Description of Decoder Structure

The original Turbo code utilized Maximum A Posteriori (MAP) algorithm also known as BCJR [1] algorithm. In this proposal, we describe the MAP algorithm together with the logarithmic versions of the MAP algorithm that are computationally less complex. A new hardware implementation of MAP-based decoding algorithms is also presented here.

A. MAP Algorithm

The MAP decoding algorithm is a recursive technique that computes the Log-Likelihood Ratio (LLR) of each bit based on the entire observed data block of length N.

$$\Lambda_1(d_k) = \log \frac{\Pr(d_k = 1|R_1^N)}{\Pr(d_k = 0|R_1^N)} \tag{1}$$

$\Pr(d_k = 1|R_1^N)$ is the a posteriori probability (APP) of the information input data at time k (d_k) when it is equal to 1 given the entire received data. The observation block data sequence is $R_1^N = \{R_1, \ldots, R_k, \ldots, R_N\}$ where $R_k = \{d_k^r, y_k^{r_i}\}$. The state of the encoder S_k is represented by a v-tuple

$$S_k = (a_k, a_{k-1}, \ldots, a_{k-v+1}) \tag{2}$$

where a_k is the output of the first shift register in the RSC encoder. The conditional joint probability $\Gamma_k^j(s)$ is defined as

$$\Gamma_k^j(s) = \Pr(d_k = j, S_k = s|R_1^N) \tag{3}$$

The APP of d_k is thus equal to

$$\Pr(d_k = j|R_1^N) = \sum_s \Gamma_k^j(s), \quad \text{j=0,1} \tag{4}$$

The LLR can be rewritten by substituting (4) into (1).

$$\Lambda_1(d_k) = \log \frac{\sum_s \Gamma_k^1(s)}{\sum_s \Gamma_k^0(s)} \tag{5}$$

The numerator and denominator of (5) can be multiplied by $\Pr(R_1^N)$ and these values will become joint probabilities instead of conditional joint probabilities. If the system at time $k-1$ is at state $s^{'}$, then (5) can be written as

$$\Lambda_1(d_k) = \log \frac{\sum_s \sum_{s'} \Pr(d_k = 1, S_k = s, S_{k-1} = s^{'}, R_1^N)}{\sum_s \sum_{s'} \Pr(d_k = 0, S_k = s, S_{k-1} = s', R_1^N)} \tag{6}$$

The BCJR algorithm [1] defines these joint probabilities in terms of three parameters.

$$\alpha_k(s) = \Pr(S_k = s|R_1^k) \tag{7}$$

$$\beta_k(s) = \frac{\Pr(R_{k+1}^N | S_k = s)}{\Pr(R_{k+1}^N | R_1^k)} \tag{8}$$

and

$$\gamma_j(R_k, s^{'}, s) = \Pr(d_k = j, S_k = s, R_k | S_{k-1} = s^{'}) \tag{9}$$

The LLR in (6) can now be described in terms of (7), (8), and (9).

$$\Lambda_1(d_k) = \log \frac{\sum_s \sum_{s'} \gamma_1(R_k, s^{'}, s)\alpha_{k-1}(s^{'})\beta_k(s)}{\sum_s \sum_{s'} \gamma_0(R_k, s^{'}, s)\alpha_{k-1}(s^{'})\beta_k(s)} \tag{10}$$

$\alpha_k(s)$ and $\beta_k(s)$ can be computed by forward and backward recursions respectively based on $\gamma_j(R_k, s^{'}, s)$.

$$\alpha_k(s) = h_\alpha \sum_{s'} \sum_{j=0}^{1} \gamma_j(R_k, s^{'}, s)\alpha_{k-1}(s^{'}) \tag{11}$$

$$\beta_k(s) = h_\beta \sum_{s'} \sum_{j=0}^{1} \gamma_j(R_{k+1}, s, s^{'})\beta_{k+1}(s^{'}) \tag{12}$$

where h_α and h_β are the normalization factors. $\gamma_j(R_k, s^{'}, s)$ consists of the transition probability of the descrete Gaussian memoryless channel and transition probabilities of the encoder trellis. From (9), $\gamma_j(R_k, s^{'}, s)$ is given by

$$\gamma_j(R_k, s^{'}, s) = \Pr(R_k | d_k = j, S_k = s, S_{k-1} = s^{'}) \times \Pr(d_k = j | S_k = s, S_{k-1} = s^{'}) \times \Pr(S_k = s | S_{k-1} = s^{'}) \tag{13}$$

The second term in (13) is the transition probability of the discrete channel, the third term is equal to 1 or 0 depending on whether it is possible for $d_k = j$ when the sytem transition is from state $s^{'}$ to state s, and the fourth term is the transition state probabilities and for equiprobable binary data, it is equal to $\frac{1}{2}$. In the first iteration, decoder 1 does not have any additional a priori information. The second decoder however, will utilize the output information from the first decoder as a priori information. After the first iteration, each decoder utilizes output information from the other decoder as a priori information. This output information of each decoder corresponds to the parity information of that decoder. d_k^r and y_k^{ri} are two uncorrelated Gaussian variables in R_k based on the conditions expressed in (13). Therefore, the second term in (13) can be divided into two terms.

$$\Pr(R_k | d_k = j, S_k = s, S_{k-1} = s^{'}) = \Pr(d_k^r | d_k = j, S_k = s, S_{k-1} = s^{'}) \times \Pr(y_k^{ri} | d_k = j, S_k = s, S_{k-1} = s^{'}) \tag{14}$$

The received signals are utilized to compute $\gamma_j(R_k, s^{'}, s)$ and consequently, accurate computation of this variable is very important in computation of other variables of the MAP decoding algorithm.

Suppose a QAM signal consists of m bits and $d_k = j$ is represented by the real (I_{k_1}) part of the QAM signal. Then the second term in (14) can be written as

$$\Pr(d_k^r | d_k = j, S_k = s, S_{k-1} = s^{'}) = \sum_{k_1=0}^{\frac{m}{2}} \Pr(I_{k_1}^r | I_{k_1} = i, S_k = s, S_{k-1} = s^{'})$$

Here m is assumed to be even number. If the bit (d_k) was represented by the imaginary (Q_{k_1}) part of the QAM signal, then I_{k_1} will be replaced by Q_{k_1} in the above equation. $\Pr(y_k^{ri} | \ldots)$ can be computed in a similar way.

The MAP decoding algorithm consists of the following steps:

1. Initialize $\alpha_o(s)$ and $\beta_N(s)$ as follows:

$$\alpha_o(s) = \begin{cases} 1 & \text{if } s = 0 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$$\beta_N(s) = \frac{1}{M} \quad \text{for all s} \tag{16}$$

   where M is the total number of states. The above initialization is based on trellis termination of the Turbo block into arbitrary state. Turbo block can be terminated to all zero state and in this case, $\beta_N(s)$ function should be initialized accordingly [2].

2. Upon receiving each d_k^r and its corresponding $y_k^{r_i}$, the decoder computes $\gamma_j(R_k, s^{'}, s)$ for j=0 and 1, then computes $\alpha_k(s)$ for all values of s according to (11). The computed values of $\gamma_j(R_k, s^{'}, s)$ and $\alpha_k(s)$ are stored for $1 \leq k \leq N$.
3. The backward recursion for $\beta_k(s)$ is performed after all the N data sequence and their corresponding parity bits are received based on (12) for $1 \leq k \leq N-1$.
4. The soft output decoded bits, $\Lambda_1(d_k)$, are computed according to (10) for $1 \leq k \leq N$.

It can be shown [1] that the soft output decoded bits, $\Lambda_1(d_k)$ or $\Lambda_2(d_k)$, can be divided into three terms. Figure 1 shows this iterative decoding scheme. The inputs to each decoder are the received input data sequence, d_k^r, the received parity bits $y_k^{r_1}$ or $y_k^{r_2}$, and the logarithm of the likelihood ratio (LLR) associated with the parity bits from the other decoder (W_k^1 or W_k^2), which is used as *a priori* information. The decoder utilizes all these inputs to create three outputs corresponding to the weighted version of these inputs. In Figure 1, $\hat{d}_k$ represents the weighted version of the received input data sequence, d_k^r. Also d_n^r in the same figure demonstrates the fact that the input data sequence is fed into the second decoder after interleaving. The input to each decoder from the other decoder is used as *a priori* information in the next decoding step and corresponds to the weighted version of the parity bits.

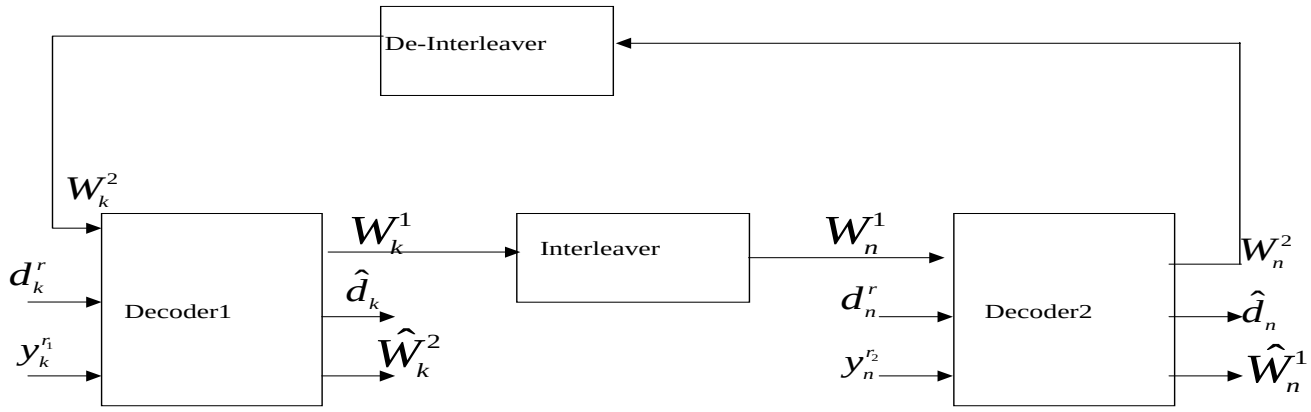

Fig. 1. Turbo Decoder.

In order to utilize this algorithm, $\alpha_k(s)$ variables are computed for the entire data block of length N, then $\beta_k(s)$ variables are computed. This approach requires all these variables to be stored for the entire of Turbo block. For the cases that may utilize a large value of N, the memory requirement for MAP-based decoding algorithm becomes extremely large. If all the $\alpha_k(s)$ and $\beta_k(s)$ variables are computed within one clock cycle for each k, then this approach requires 2N clock cycles to compute all the variables of the MAP decoding algorithm. We propose another hardware implementation that requires storage of only half of the $\alpha_k(s)$ and $\beta_k(s)$ variables and the computations of these variables will be carried with the minimum number of clock cycles (N clock cycles).

B. MAX-Log-MAP Algorithm

As described earlier, MAP algorithm is computationally very intensive for most applications and it may not be suitable for chip design. There are two major problems with MAP decoding algorithm. First, MAP requires accurate estimation of the noise variance and its performance is very sensitive to SNR mismatch. Second, the fixed-point representation of the MAP decoding variables requires between 16 to 24 bits. These requirements are not suitable for VLSI chip design with current technology.

To avoid these problems, we can compute the Natural logarithm of all these variables, i.e., $\gamma_j(R_k, s^{'}, s), \alpha_k(s)$, and $\beta_k(s)$. Since $\gamma_j(R_k, s^{'}, s)$ is the result of multiplication of three factors in (13), the logarithm of $\gamma_j(R_k, s^{'}, s)$, $\tilde{\gamma}_j(R_k, s^{'}, s)$, is the addition of the logarithm of these three factors.

$$\tilde{\gamma}_j(R_k, s^{'}, s) \;=\; \ln \gamma_j(R_k, s^{'}, s) = \ln(\Pr(R_k | d_k = j, S_k = s, S_{k-1} = s^{'})) + \ln(\Pr(d_k = j | S_k = s, S_{k-1} = s^{'}))$$

$$+ \quad \ln(\Pr(S_k = s | S_{k-1} = s^{'})) \tag{17}$$

In an additive white Gaussian noise (AWGN) environment, the first term in the right side of (17) is an exponent and by taking the Natural logarithm of this value, we eliminate the non-linear exponent operation, e.g., $\ln(\exp(A)) = A$.

For $\alpha_k(s)$ we have

$$\tilde{\alpha}_k(s) = \ln(\alpha_k(s)) = \ln(h_\alpha \sum_{s^{'}} \sum_{j=0}^{1} \gamma_j(R_k, s^{'}, s)\alpha_{k-1}(s^{'})). \tag{18}$$

A simple approximation to the logarithm of the sum of numbers is the logarithm of the maximum number.

$$\ln(A + B + C + \ldots) \simeq \ln(\max(A, B, C, \ldots)) \tag{19}$$

Therefore, $\tilde{\alpha}_k(s)$ can be approximated as

$$\tilde{\alpha}_k(s) = \ln(\alpha_k(s)) \simeq \max_{alls^{'}}(\tilde{\gamma}_j(R_k, s^{'}, s) + \tilde{\alpha}_{k-1}(s^{'})) + \ln h_\alpha \tag{20}$$

with the following initialization condition:

$$\tilde{\alpha}_o(s) = \begin{cases} 0 & \text{if } s = 0 \\ -\infty & \text{otherwise} \end{cases} \tag{21}$$

Similar approximation can be used to compute $\tilde{\beta}_k(s)$.

$$\tilde{\beta}_k(s) = \ln(\beta_k(s)) \simeq \max_{alls^{'}}(\tilde{\gamma}_j(R_{k+1}, s, s^{'}) + \tilde{\beta}_{k+1}(s^{'})) + \ln h_\beta \tag{22}$$

The initialization of $\tilde{\beta}_k(s)$ based on (16) is

$$\tilde{\beta}_N(s) = \ln(\frac{1}{M}) \quad \text{for all s} \tag{23}$$

The soft output of the decoded data for this approach is

$$\tilde{\Lambda}_1(d_k) \simeq \max_{alls,s^{'}}(\tilde{\gamma}_1(R_k, s^{'}, s) + \tilde{\alpha}_{k-1}(s^{'}) + \tilde{\beta}_k(s)) - \max_{alls,s^{'}}(\tilde{\gamma}_0(R_k, s^{'}, s) + \tilde{\alpha}_{k-1}(s^{'}) + \tilde{\beta}_k(s)) \tag{24}$$

The $\alpha_k(s)$ and $\beta_k(s)$ parameters in the MAP algorithm are approximated in the MAX-Log-MAP algorithm by maximization operation. Therefore, there is an approximation error in the computation of these two variables. Since these two variables are computed recursively, this approximation error is propagated throughout the entire block of data. If the SNR requirement for a given BER performance is very high, then this approximation error is comparable to the noise and it will have a significant effect on the performance of the system. On the other hand, if the SNR requirement is not high, then this approximation error is much less than the noise power and this will not be a significant factor in performance degradation. We will show this in the simulation results section. The BER performance of the MAX-Log-MAP is always worse than that of the MAP algorithm.

C. Log-MAP Algorithm

The Log-MAP algorithm computes the MAP parameters by utilizing a correction function to compute the logarithm of sum of numbers. More precisely for $A_1 = A + B$, then

$$\tilde{A}_1 = \ln(A + B) = \max(\tilde{A}, \tilde{B}) + f_c(|\tilde{A} - \tilde{B}|) \tag{25}$$

where $f_c(|\tilde{A} - \tilde{B}|)$ is the correction function. $f_c(|\tilde{A} - \tilde{B}|)$ can be computed using either a look-up table [2] or simply a threshold detector [3] that performs similar to look-up table. The simple equation for threshold detector is

$$f_c(|\tilde{A} - \tilde{B}|) = \begin{cases} 0.375 & \text{if } |\tilde{A} - \tilde{B}| \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

(25) can be extended recursively. If $A_2 = A + B + C$, then

$$\tilde{A}_2 = \ln(A_1 + C) = \max(\tilde{A}_1, \tilde{C}) + f_c(|\tilde{A}_1 - \tilde{C}|) \tag{26}$$

This recursive operation is specially needed for computation of the soft output decoded bits.

At each step, the logarithm of addition of two values by maximization operation is accommodated for by a correction value which is provided by a look-up table or a threshold detector in the Log-MAP algorithm. The Log-MAP parameters are very close approximations of the MAP parameters and therefore, the Log-MAP BER performance is close to that of the MAP algorithm.

D. Simplified-Log-MAP Algorithm

Study of the MAP algorithm shows that accurate computation of $\gamma_j(R_k, s', s)$ function is very important since it contains all the received information data. $\alpha_k(s)$ and $\beta_k(s)$ parameters are computed recursively, therefore any error in the computation of these parameters can propagate and results in poor estimation of these parameters. For example, if $\alpha_k(s)$ is computed with approximation using (19), then this error will result in inaccurate computation of the values of $\alpha_{k+1}(s)$. Any additional error in computation of $\alpha_{k+1}(s)$ will create larger errors in computation of $\alpha_{k+2}(s)$ and this can continue until the end of the Turbo block. This is the main reason that the MAX-Log-MAP algorithm performs worse than the MAP algorithm. Similar discussion is true for the backward recursive computation of $\beta_k(s)$ parameters. $\tilde{\alpha}_k(s)$ and $\tilde{\beta}_k(s)$ can then be computed as

$$\tilde{\alpha}_k(s) = \max_{alls'}(\tilde{\gamma}_j(R_k, s', s) + \tilde{\alpha}_{k-1}(s')) + f_c(|\triangle_\alpha|) \tag{27}$$

$$\tilde{\beta}_k(s) = \max_{alls'}(\tilde{\gamma}_j(R_{k+1}, s, s') + \tilde{\beta}_{k+1}(s')) + f_c(|\triangle_\beta|) \tag{28}$$

where $\triangle_\alpha$ is defined as the difference between the maximum and minimum values of $(\tilde{\gamma}_j(R_k, s', s) + \tilde{\alpha}_{k-1}(s'))$. $\tilde{\alpha}_o(s)$ and $\tilde{\beta}_N(s)$ initializations are similar to (21) and (23).

However, after computing the logarithm of these parameters using the correction function, it is not necessary to compute $\Lambda_1(d_k)$ with high accuracy. As a matter of fact, a lot of times at moderate to high SNR only one value in the numerator or denominator of (10) is the dominant factor. Therefore, using max operation similar to (19) to compute $\Lambda_1(d_k)$ will not have any significant effect on the BER performance of the Simplified-Log-MAP algorithm, while it reduces the computational complexity of the Simplified-Log-MAP algorithm compared to that of the Log-MAP algorithm. Besides any error due to this approximation in computation of $\Lambda_1(d_k)$ will not propagate through the entire data sequence. The soft output of the decoded bits are approximated as:

$$\tilde{\Lambda}_1(d_k) \simeq \max_{alls,s^{'}}(\tilde{\gamma}_1(R_k, s^{'}, s) + \tilde{\alpha}_{k-1}(s^{'}) + \tilde{\beta}_k(s)) - \max_{alls,s^{'}}(\tilde{\gamma}_0(R_k, s^{'}, s) + \tilde{\alpha}_{k-1}(s^{'}) + \tilde{\beta}_k(s)) \qquad (29)$$

Table 1 compares the computational complexity of all the MAP-based decoding algorithms for a T1 data rate. The total number of operations are for only one iteration with $v = 3$ and $M = 8$. From this Table, we can conclude that the total number of operations per bit for the Simplified Log-MAP is 16 and 13 percent less than the MAP and Log-MAP algorithms respectively.

Operation	**MAP**	**Max-Log-MAP**	**Simplified-Log-MAP**	**Log-MAP**
Maximization	2M - 1	5M - 2	3M - 2	4M - 4
Addition	4M	10M - 2	12M - 2	14M - 4
Multiplication	10M	0	0	0
Table Look-up	0	0	2M	4M - 2
Total Operation	14M	10M - 2	12M - 2	14M - 4
Total Ops. for T1 in Mops.	345.86	240.86	290.27	333.5

TABLE I

COMPLEXITY COMPARISON BETWEEN DIFFERENT MAP ALGORITHMS.

E. A new hardware architecture for MAP-based decoding algorithm

The new hardware implementation of the MAP-based decoding algorithms is based on the assumption that the entire N block of data is available at the receiver. For instance, the digital subscriber loop (DSL) modems that use the discrete multi-tone (DMT) technology allocate the information bits to the tones in a transmission frame. The number of transmitted bits in each frame can be equal to the Turbo block size (N). In other applications in which the information and the parity bits are transmitted sequentially, in order to take full advantage of the coding gain associated to Turbo codes, we need to perform many iterations in the decoding operation. Therefore, the above assumption is reasonable

after the first iteration. The new approach presented here is for the MAP decoding algorithm. A generalization of these steps to logarithmic versions of the MAP algorithm is straightforward.

1. Initialize $\alpha_o(s)$ and $\beta_N(s)$ based on (15) and (16). Set $k = 1$ and Compute $\gamma_j(R_k, s^{'}, s)$ and $\gamma_j(R_{N-k+1}, s^{'}, s)$ for all j.
2. Compute $\alpha_k(s)$, $\beta_{N-k}(s)$, $\gamma_j(R_{k+1}, s^{'}, s)$, and $\gamma_j(R_{N-(k+1)+1}, s^{'}, s)$ for all values of $s, s^{'}$, and j in a parallel structure simultaneously in one clock cycle. Store all these values in memory.
3. Set $k = k + 1$ and go back to step 2 for $1 \leq k \leq \frac{N}{2}$. When $k > \frac{N}{2}$, then go to step 4.
4. For $\frac{N}{2} + 1 \leq k \leq N$, compute $\alpha_k(s)$ and $\beta_{N-k}(s)$ simultaneously. Use these values together with the stored $\alpha_k(s)$ and $\beta_k(s)$ parameters calculated in step 2 to compute $\Lambda_1(d_k)$ and $\Lambda_1(d_{N-k})$. Do not store $\alpha_k(s)$ and $\beta_{N-k}(s)$ parameters for $\frac{N}{2}+1 \leq k$. Remember that all the values of $\gamma_j(R_k, s^{'}, s)$ are already computed in step 2.
5. Set $k = k + 1$ and go back to step 4 and continue the algorithm for the entire Turbo block.

   This approach can be used for both decoders.

This approach has many advantages over the previous proposed technique. If there are a total of $M = 2^v$ states, the total memory requirement for $\alpha_k(s)$ or $\beta_k(s)$ parameters for this approach is $2^{v-1} \times N$ as compared to $2^v \times N$ in the previous technique. In many applications with Turbo code, N is a large number and this approach reduces the memory requirements.

The computation of parameters is carried much faster in this approach with the minimum hardware requirements. As a matter of fact, the soft output decoding for each iteration is available immediately after finishing computation of $\alpha_k(s)$ and $\beta_k(s)$ parameters. The total number of clock cycles to perform the operations for the entire Turbo block of length N is only N clock cycles as compared to 2N cycles for the previous approach.

To show the feasibility of these techniques, we have computed the gate count requirement for implementation of Simplified-Log-MAP algorithm with this new hardware architecture for a Turbo code with three iterations. The operation is divided into two parts: Computing the gamma (γ) values for each bit based on the received M-QAM constellation and Simplified-Log-MAP decoding algorithm. The first operation for a Turbo block requires approximately 35k gate and memory size of 1k×11 bits of ROM. For Simplified-Log-MAP decoding, it also requires 35k gates and memory size of 1k×11 bits ROM and 22k×8 bits of RAM. The total processing time for both operations require 6500 clock cycles for 3 iterations. For a clock with rate of 14 nsec, this is equivalent of 92 μsec which is smaller than one frame size (250 μsec). This shows that processing delay will not be an issue for this design.

For a block size of 1088 bits and 0.35 μm technology (corresponding to 70 MHz clock), we can have approximately 22 iterations which is more than enough.

II. Simulation Results

The BER performance of the Simplified-Log-MAP algorithm is compared to that of the MAP, Log-MAP, and Max-Log-MAP algorithms. The simulation results are for a Turbo code with 1/2 code rate, with only one random interleaver, $N = 1088$, $v = 3$, and the feedback and feed-forward generator polynomials equal to 15_{oct} and 17_{oct} respectively. Three iterations were used for the simulations and the results in figure 2 and 3 are for QPSK and 64-QAM constellation sizes. As we can see from the results, the Log-MAP decoding algorithm has similar performance to the MAP algorithm (For QPSK, it is exactly the same, so we did not plot it). The performance loss for the MAX-Log-MAP compared to the MAP algorithm is from 0.2 dB for QPSK, up to 0.35 dB for 64-QAM. The SNR requirement for a given BER is higher at larger constellation sizes; therefore, the approximation of the logarithm has more significant effect on the BER performance of the MAX-Log-MAP algorithm. The Simplified Log-MAP has a negligible performance degradation compared to the MAP algorithm for QPSK constellation, while the performance loss is approximately 0.1 dB. for 64-QAM. It can be concluded from the above results that Simplified Log-MAP algorithm together with the new hardware implementation are an appropriate combination for implementing Turbo decoders in practice without any significant loss in performance.

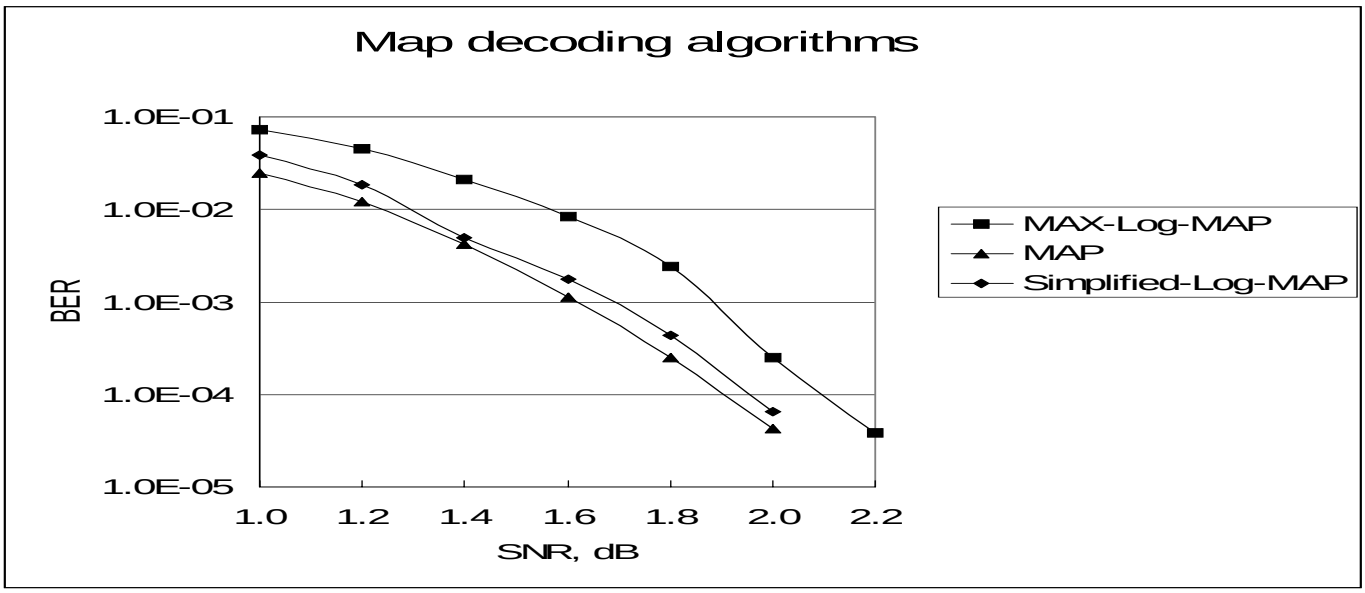

Fig. 2. BER performance of MAP-based decoding algorithms for QPSK constellation.

III. Summary

This contribution should be presented under the activity in G.gen.

We propose to include Multi-Tone Turbo Trellis Coded Modulation to G.lite.bis and G.dmt.bis.

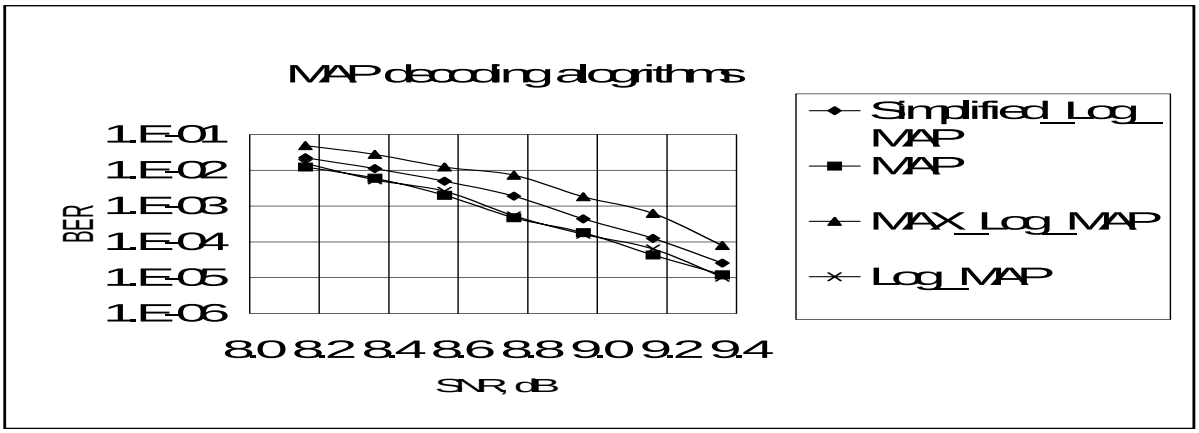

Fig. 3. BER performance of MAP-based decoding algorithms for 64QAM constellation.

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, *"Optimum decoding of linear codes for minimizing symbol error rate,"* IEEE Trans. on Inf. Theory, vol. IT-20, pp. 284-287, Mar. 1974.

[2] S. Benedetto, G. Montorsi, D. Divsalr, and F. Pollara *"Soft-Output Decoding Algorithm in Iterative Decoding of Turbo Codes,"* TDA Progress Report 42-124, pp. 63-87, February 15, 1996.

[3] W.J. Gross and P.G. Gulak, *"Simplified MAP algorithm suitables for implementation of turbo decoders,"* Electronics Letters, vol. 34, no. 16, August 6, 1998.