TSG-RAN Working Group 1(Radio) meeting #7 *TSGR1#7(99) b89*
30 August - 3 September 1999
Hannover, Germany

Source : LGIC and SAMSUNG Electronics Co.

Title : Simulation Results of Puncturing Algorithms for Turbo Code

1 Introduction

In WG1 meeting #6, Samsung and LGIC proposed a joint contribution on the rate matching algorithm for turbo code[1], which became working assumption. But a counter proposal from Nortel followed, which uses the internal PIL interleaver of turbo encoder to determine the puncturing position for the 2nd parity bit stream. Nortel also proposed that convolutional code and turbo code should be treated in a unified manner[2].
In this contribution, we will present some simulation results which compare Samsung & LGIC's method with Nortel's proposed method. Simulation results show that our method, for 1/2 rate turbo code, provides superior performance compared with the interleaver controlled puncturing scheme proposed by Nortel. It is also shown that for convolutional code, Nortel's proposed method provides no significant performance gain compared to the conventional scheme.

2 Simulation Results for Turbo Code Puncturing

The environments of simulation are as follows.

- Depth of PIL interleaver : 320, 324, 640, 964
- Constituent Decoder : Optimum MAP decoder
- Number of Iteration for Decoding : 8
- Channel : static AWGN channel
- Frame Error : more than 100 frame error were counted.
- (a,b) parameter for each RMB : (2,1) for RMB 2, (3,1) for RMB3

2-1. 1/2 rate turbo code generation

Nortel insisted that by using Nortel's PIL interleaver controlled puncturing method, significant performance gain can be achieved for 1/2 rate turbo code. This section will show some simulation results, which compare nortel's method with Samsung & LGIC's method for the rate 1/2 turbo code. The size of PIL interleaver used in simulation is 320, 324, 640, 964 respectively.
Figure 1 shows the simulation results when the size of PIL interleaver is 320 and 320 bit puncturing is applied. In this case, the amount of puncturing imposed on both RMB2 and RMB3 is 160. In fact, when 33.3 % puncturing is imposed to make 1/2 rate turbo code by rate matching puncturing, the amount of puncturing becomes 324 for rate 1/2 turbo code. But, in terms of 1/2 turbo code pattern, there is no puncturing on the tail parts as you can see in the spec. Also, to our knowledge, Nortel's proposed method can not impose puncturing on the tail parts, because the depth of PIL interleaver is the same as the length of information part.
The simulation results shows that Samsung and LGIC's method provides better BER and FER performance. The performance gain exceeds 0.2dB at FER below 10^{-3} .
In the process of e-mail discussion, Nortel also provided some files which contain 1/2 turbo code puncturing pattern.
But, we can't understand in which way Nortel obtained those patterns. We observed that the provided puncturing pattern imposed puncturing over tail part. Anyway, simulation has been performed using the pattern provided by Nortel, and the results are shown in figure 2. The size of PIL interleaver is 320, and the amount of puncturing is 324. Therefore, the amount of puncturing for RMB2 and RMB3 is 162 for Samsung & LGIC's method. The (a,b) parameter for Samsung & LGIC's method is (2,1) for RMB2 and (3,1) for RMB3.
We can also obtain the Berrou's 1/2 puncturing pattern using the proposed puncturing method. For this pattern, the parameter (a,b) was set to (1,1) for RMB2 and (2,1) for RMB3.

As you can see in the figure, our method gives better performance than Nortel's 1/2 puncturing pattern provided by Nortel via e-mail.
Figure 3 and figure 4 show the simulation results for 324 PIL interleaver size and 964 PIL interleaver size. As can be seen in the figure, the difference between our method and Nortel's method becomes more explicit as the length of PIL interleaver increases.

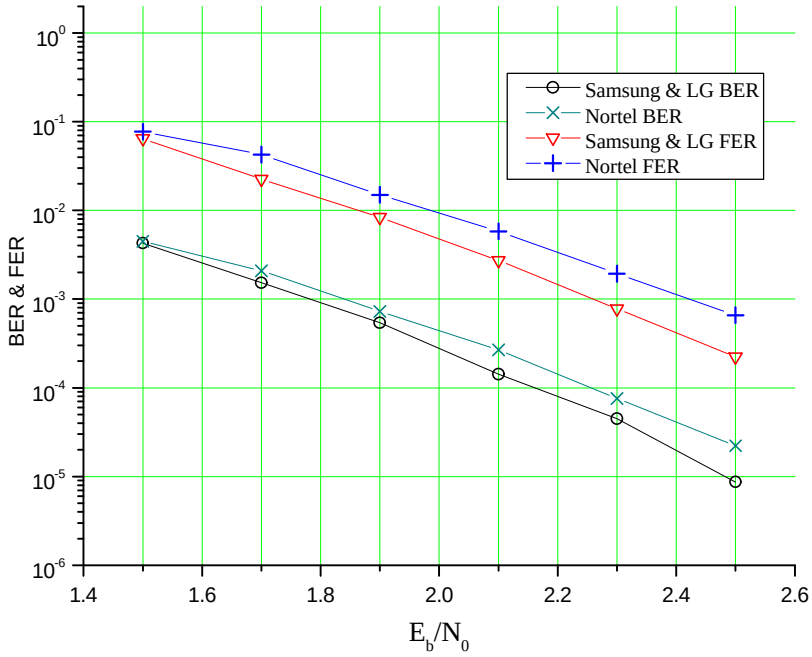

Figure 1. PIL interleaver size : 320, 320 bit puncturing is imposed.

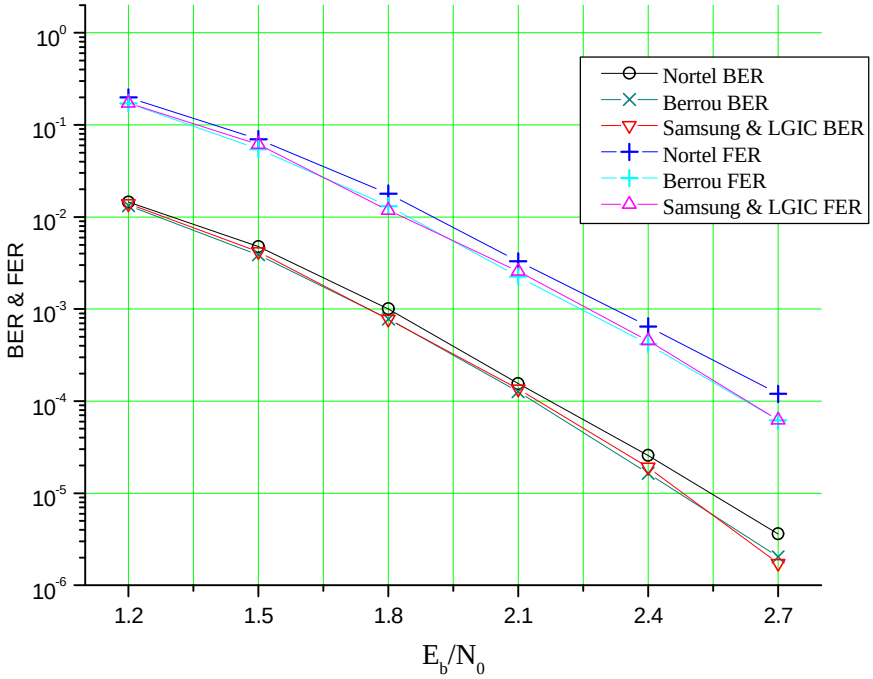

Figure 2. PIL interleaver size : 320, 324 bit puncturing is imposed.

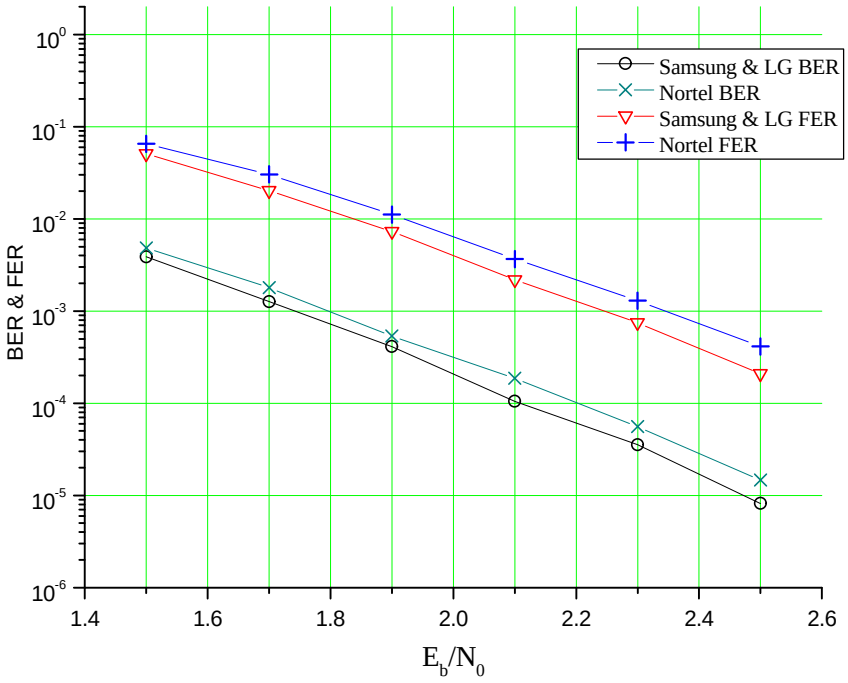

Figure 3. PIL interleaver size : 324, 324 bit puncturing is imposed.

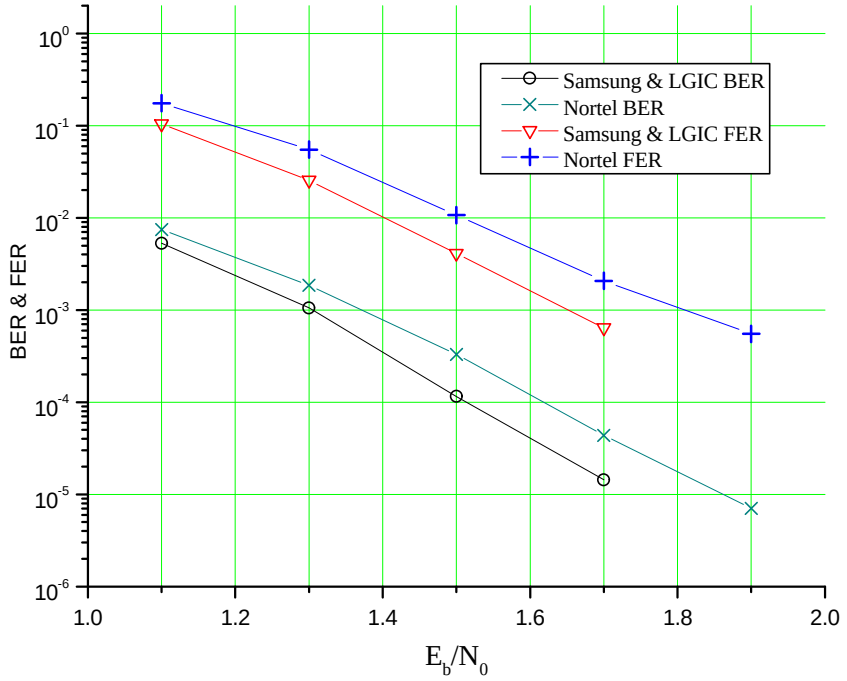

Figure 4. PIL interleaver size : 964, 964 bit puncturing is imposed.

These results suggest that uniform puncturing for the 2nd RSC's parity is equally important requirement as the requirement for the uniform puncturing for 1st RSC's parity. Nortel's method imposes consecutive puncturing on 2nd RSC's parity and, it is our thought that the 2nd constituent decoder can't display the full decoding capability because there is consecutive puncturing for Nortel's method.

2-2 Simulation results for coding rates > 1/2 turbo code

It is hard to find cases which use coding rates higher than 1/2 for turbo code. However, we also performed some simulations to examine the performance of our scheme and Nortel's scheme for the case of coding rates higher than 1/2.

Figure 5 shows the simulation results when the depth of the PIL interleaver is 320. Let's assume that the size of target block after rate matching is 592.

For 1/3 turbo code with 320 PIL interleaver depth, the number of coded bits from turbo encoder is 972, including 12 tail parts. Therefore, to reach the target rate using the proposed method, 380 bit puncturing is imposed. Finally, for the proposed method, both RMB2 and RMB3 shall impose 190 bit puncturing.

If 1/2 turbo code is used as starting point, the number of coded bits from turbo encoder is 652, including 12 tail parts. Therefore, to reach the target rate using the proposed method, 60 bit puncturing shall be imposed. Finally, each RMB for parity stream shall impose 30 bit puncturing to reach the target rate.

As can be seen in the figure, the proposed method gives the best performance while Nortel's pattern provides the worst. The difference between the performance of proposed method(from 1/3 turbo code) and that of 1/2 Berrou pattern is not that significant but can be noticed.

Figure 6 shows the simulation results when the depth of the PIL interleaver is 640. It is assumed that the size of target block after rate matching is 1164.

The simulation results show the same trend as above.

Figure 7 shows the simulation results when the depth of the PIL interleaver is 640 and the size of target block after rate matching is 1076. The results have the same tendency as above.

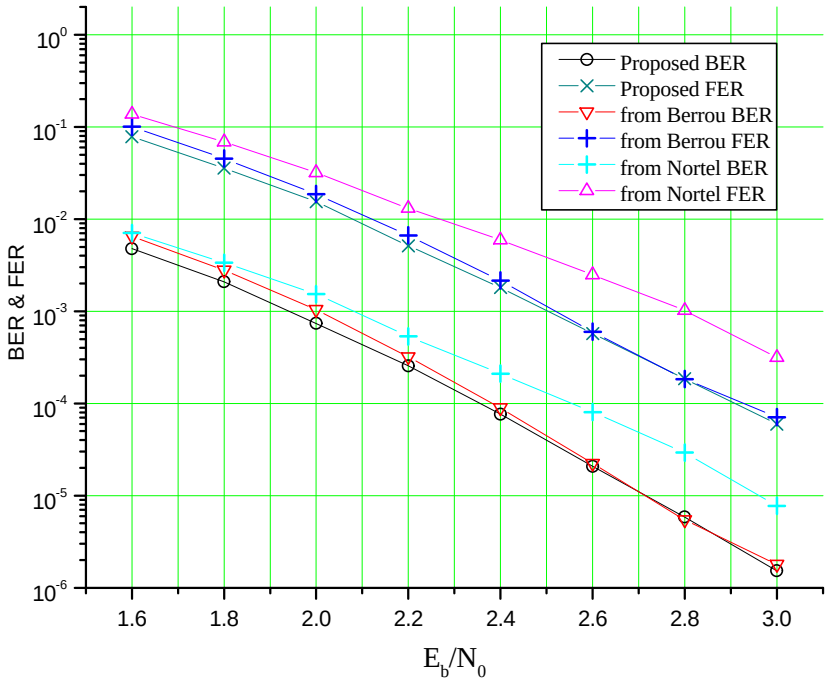

Figure 5. Depth of the PIL interleaver is 320 and 380 bit puncturing is imposed from 1/3 turbo code to obtain the target rate and 60 bit puncturing is imposed from 1/2 Berrou pattern and Nortel pattern

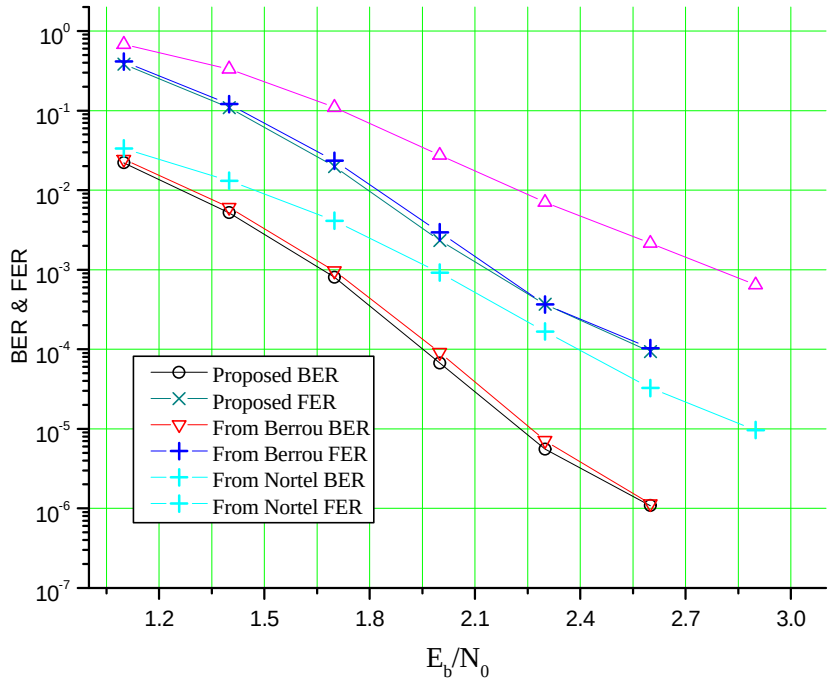

Figure 6. Depth of the PIL interleaver is 640 and 768 bit puncturing is imposed from 1/3 turbo code to obtain the target rate and 128 bit puncturing is imposed from 1/2 Berrou's pattern and Nortel's pattern

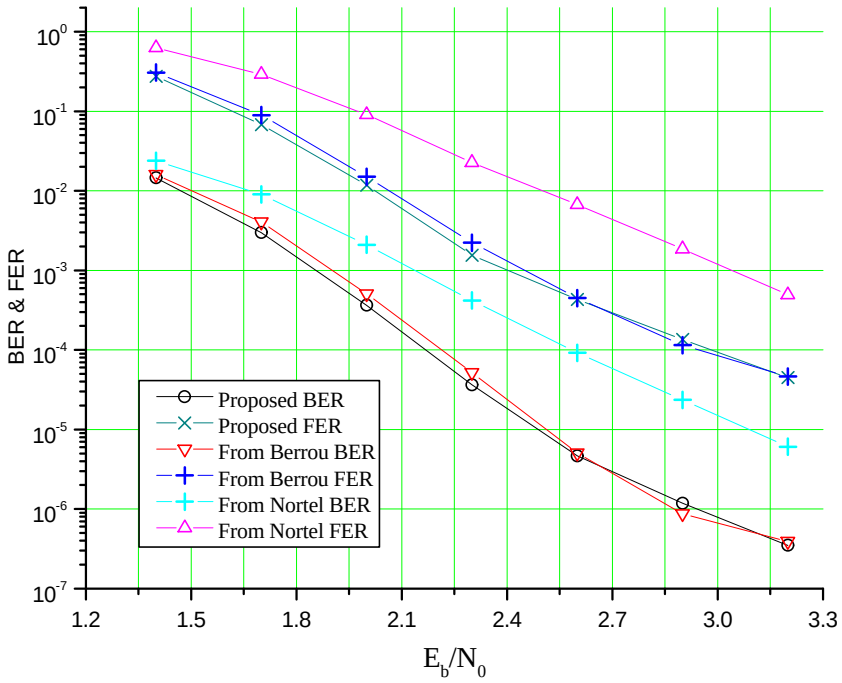

Figure 7. Depth of the PIL interleaver is 640 and 856 bit puncturing is imposed from 1/3 turbo code to obtain the target rate and 216 bit puncturing is imposed from 1/2 Berrou's pattern and Nortel's pattern

3 Simulation Results for Convolutional Code

Nortel proposed that convolutional code and turbo code should be treated in a unified manner. The rationale of the proposal is that by avoiding the puncturing of the bit stream from the generator polynomial '711' for 1/3 convolutional code, better performance can be achieved. LGIC has also studied on this. In Tdoc 907 and 908, we provided some simulation results on this matter[3][4]. In this section, some simulation results will be shown on this issue.
The environments of simulation are as follows

- Block sizes: 160 for downlink
- Puncturing rates: 42 bit puncturing, 50 bit puncturing, 84 bit puncturing and 100 bit puncturing
- trellis termination by '8' zero tail
- Decoding algorithm: Viterbi Algorithm with full path memory
- Number of frame errors: greater than 100
- Channel model: AWGN
- 'only x' represents the method of applying puncturing only to bit stream from polynomial '557' using demultiplexing scheme for turbo code
- 'x, y' represents the method of applying puncturing to bit stream from polynomial '557' and bit stream from '663' using current demultiplexing scheme for turbo code.

To incorporate Nortel's proposal into Samsung & LGIC's method, we define the followings.

- Bit stream from polynomial '711' : input to RMB1
- Bit stream from polynomial '557' : input to RMB2
- Bit stream from polynomial '663' : input to RMB3

And parameter (a,b) for RMB2 is set to (1,1) and (a,b) for RMB3 is set to (2,1), considering the overall uniformity for convolutional code. This setting provides the uniformity over all coded symbol branch to some extent. That is when the puncturing distance over all the coded bits is an exact even number, then overall uniformity is satisfied with parameters (1,1) and (2,1), because the initial puncturing position of '663' bit stream occurs exactly 2 times earlier than the initial position of '557' bit stream. But if the puncturing distance over all coded bit stream is not exactly even, the uniformity over all coded symbol branches will be somewhat broken even though not much. But for the uplink, it is a different problem. In the uplink, a general form of (a,b) can't be found to satisfy the uniformity over all the coded bits. This is one problem of incorporating the Nortel's proposal into the current puncturing scheme for turbo code.
Figure 8 shows the results when 42 bit puncturing is imposed. In figure 8, the legend 'only x' represents the performance when we impose all the puncturing on the bit stream from '557'. That is, 42 bits are punctured from one RMB, ie RMB2.
The legend (x,y) represents the performance when we impose 21 bit puncturing evenly to the bit stream from polynomial '557' and that from polynomial '663'. That is, 21 bits are punctured from RMB2 and RMB3 respectively.
If the current rate matching puncturing for convolutional code is used, then all of the 42 bits are punctured from bit stream '711', which is the worst case.
As you can see in the figure, 'only x' provided the best performance even though the difference is slight. The performance of '(x,y)' method and conventional method is almost the same.
Figure 9 shows the results when 50 bit puncturing is imposed. The tendencies in figure 8 still remains. Figure 10 shows the results when 100 bit puncturing is imposed. The tendencies in figure 8 and 9 still remains.
One problem of puncturing '557' and '663' for convolution code in a unified manner with turbo code is that overall uniformity isn't satisfied in case of uplink. A second problem is that for convolutional code, there are distinct code polynomial for 1/2 rate unlike turbo code. Then, for 1/2 convolutional code, demultiplexing scheme should be changed. These problems make the rationale of Nortel's proposal for convolutional code not suitable.
Of course, we agree that there are some cases when only the bits from polynomial '711' are punctured for conventional scheme, which is the worst case for the conventional puncturing.
This problem can be easily solved without changing the conventional puncturing scheme and multiplexing scheme for convolutional code by setting the initial offset value as follows.

$e_{offset} = (a * s(k) * y + 1) \bmod a * N_c$ for 1/3 convolutional code.

By using the above initial offset, we can prevent the worst puncturing pattern for the conventional scheme, and moreover the

worst pattern can be changed into a good pattern which provides the same performance as in the case of applying all the puncturing to the bit stream from polynomial '557'.
This method of changing the initial offset value for convolutional code can be easily incorporated into the current working assumption for rate matching algorithm.
First, let's define the initial offset value for rate matching algorithm as follows.

$$e_{offset} = (a * s(k) * y + b) \bmod a * N_c$$

And then, we can set parameter (a, b) for RMB as $(2, N_c)$ or $(3, N_c)$ for turbo code and (2,1) for convolutional code. Here N_c means the input size for each RMB.

Figure 11 contains an example which shows the effect of changing the initial offset value as above. When 84 bits are punctured from 504 coded bit stream using conventional puncturing algorithm with the current initial value, only the bits from polynomial '711' are punctured. This is the extremely worst case. When the size of information block for convolutional code is 160 and about 20% puncturing is imposed, 100 bits shall be punctured from the 504 coded bits. This is a puncturing limit. When 84 bits are punctured from 504 coded bits, the puncturing distance is 6, which is a multiple number of 3. '84' is the number which is a multiple of '6' and in the same time closest to the puncturing limit. Therefore, in this case the conventional rate matching puncturing with the current initial offset value will suffer from the worst puncturing loss. But changing the initial offset as above, only the bits from polynomial '557' are punctured, providing better performance

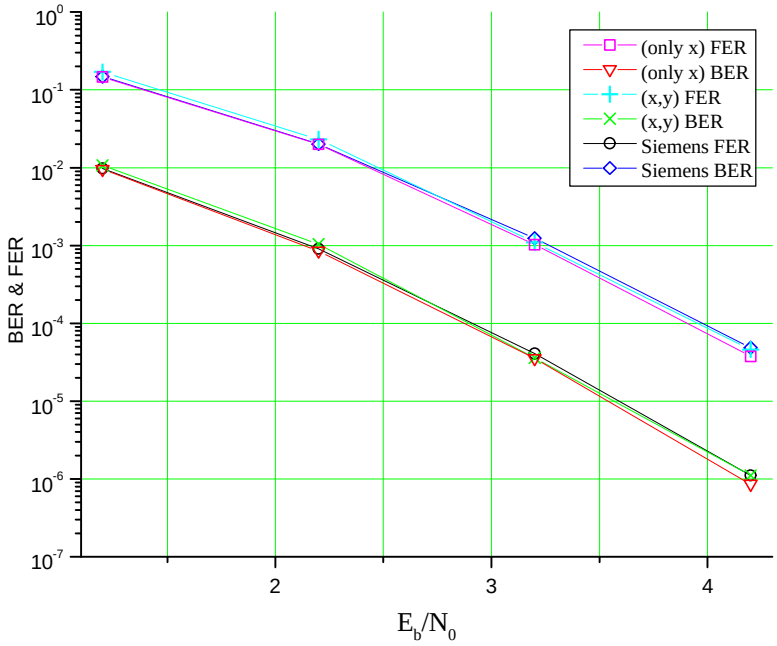

Figure 8. Performance curve of convolutional code when 42 bit puncturing is applied.

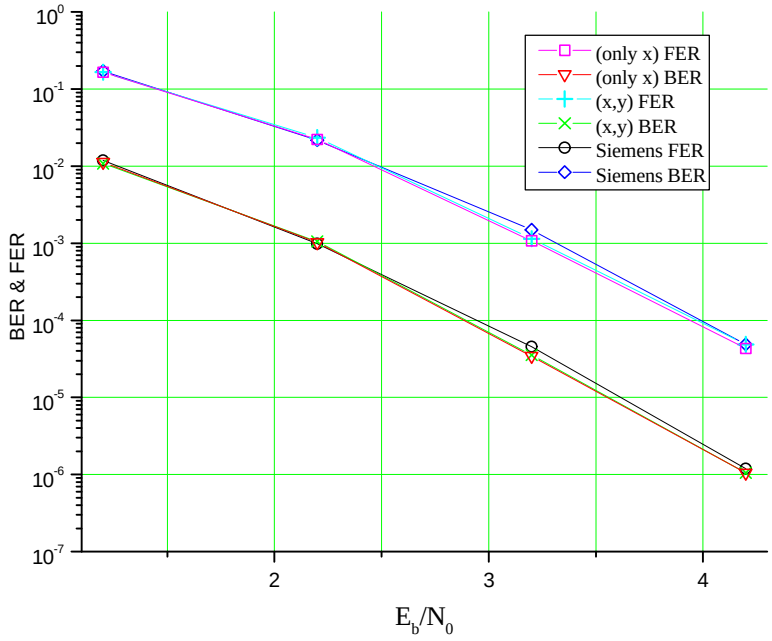

Figure 9. Performance curve of convolutional code when 50 bit puncturing is applied.

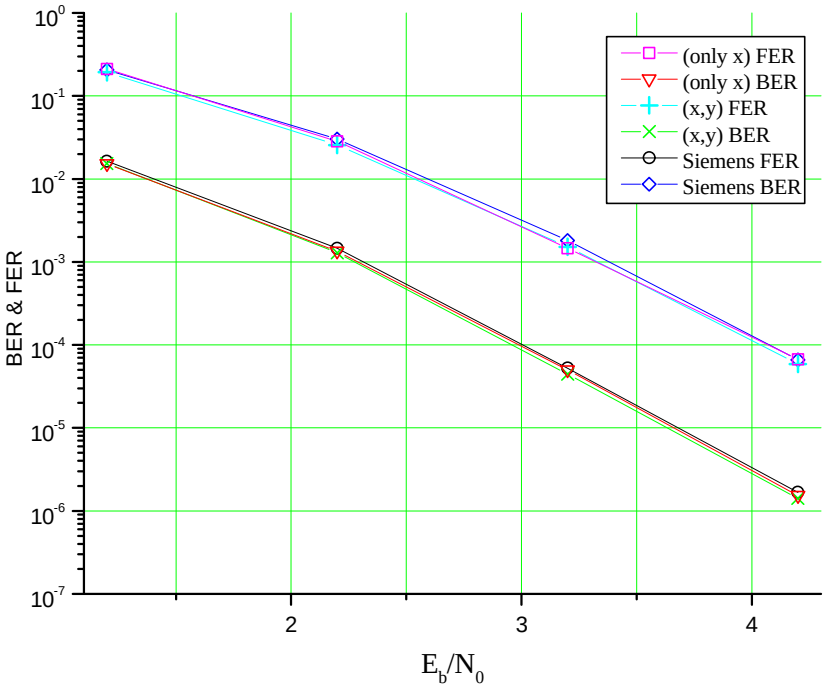

Figure 10. Performance curve of convolutional code when 100 bit puncturing is applied.

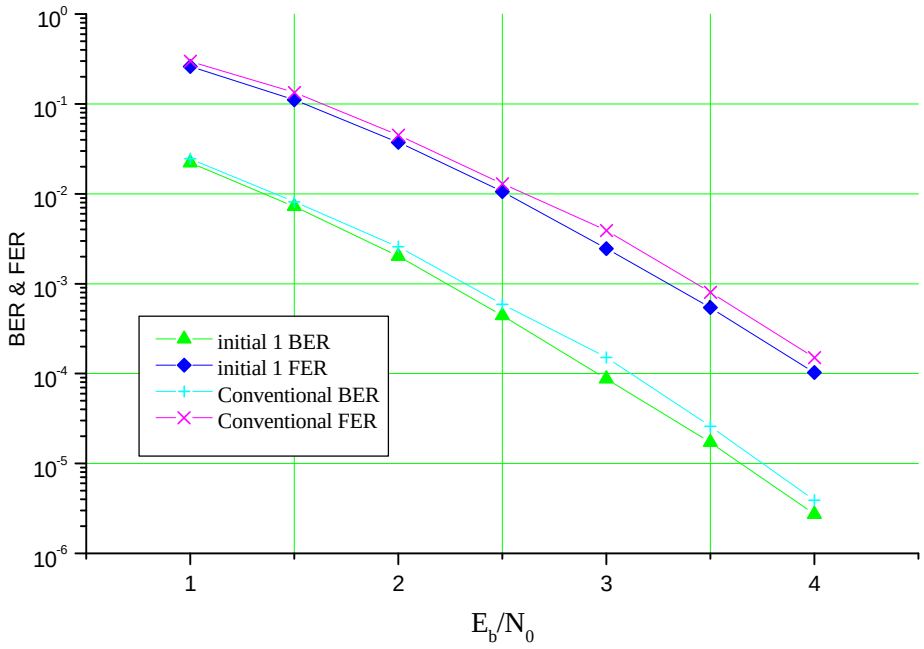

Figure 11. Performance curve of convolutional code when 84 bit Puncturing is applied.

4 Conclusion

In this contribution, we provided some simulation results on the issues suggested by Nortel in the WG1 meeting #6. From our simulation results, we can't find any benefit using the methods proposed by Nortel for both convolutional code and turbo code. And for convolutional code, we propose that we can change the initial offset value to prevent the worst case for the conventional rate matching puncturing algorithm.

5 Reference

[1] 3GPP TSG RAN WG1 R1-99A30 "Unified rate matching scheme for turbo code in both uplink and downlink", Samsung & LGIC.

[2] 3GPP TSG RAN WG1 R1-99950 "Rate Matching Puncturing for 8PCC and Convolutional Code", Nortel,

[3] 3GPP TSG RAN WG1 R1-99907 "Simulation Results of Downlink Puncturing Algorithms", LGIC

[4] 3GPP TSG RAN WG1 R1-99908 "Code Symbol Based Uplink Puncturing Algorithm", LGIC