

Chapter 5. Performance Analysis of Turbo Codes

The turbo code, as described in Chapters 3 and 4, is a very complex channel coding scheme. The turbo code encoder is a parallel concatenation of two recursive systematic convolutional (RSC) codes. The turbo code decoder is an iterative serial concatenation of two soft output Viterbi algorithm (SOVA) decoders. In addition, the presence of interleavers in both the encoder and the decoder further complicates this coding scheme.

The two main methods to evaluate the performance of turbo codes are theoretical analysis and computer simulation. Theoretical analysis of a turbo code is very difficult due to the structure of the coding scheme. A few journal papers have analyzed turbo codes with this theoretical approach; however, their results do not match closely (too optimistic) to computer simulation results. Also, the theoretical analyses presented in these papers are not clearly described and thus are difficult to follow. Furthermore, the theoretical approach often requires tremendous computer run time to find the complete weight distribution of the turbo code codewords.

In this thesis, due to the difficulties presented in the theoretical analysis, the performance of turbo codes is evaluated through computer simulation. MATLAB is used to construct the computer code of a turbo code, and the simulations were carried out on SUN and HP workstations.

5.1 Simulation Setup

The simulation setup is composed of three distinct parts, namely the encoder, the channel, and the decoder. The simulation of the turbo code encoder is based on its description in Chapter 3. The simulated turbo code encoder is composed of two identical RSC component encoders. These two component encoders are separated by a random interleaver. The random interleaver is a random permutation of bit order in a bit stream. This random permutation of bit order is stored so that the interleaved bit stream can be deinterleaved at the decoder.

In the literature, the termination aspects of a turbo code are not very well described. At the turbo code encoder, the RSC encoders need to be properly terminated by returning the code memory, of size m , to the all zero state. To perform this task, the systematic code stream is augmented by the addition of m tail bits. These m tail bits cannot be easily predetermined and depend on the code memory. The strategy to obtain these m tail bits for the systematic code stream is discussed in Chapter 3.

The output of the turbo code encoder is described by three streams, one systematic (uncoded) bit stream and two coded bit streams. The systematic bit stream can only have

one set of m tail bits from one of the two recursive encoders. Figure 5.1 and Figure 5.2 show the two possible scenarios for these m tail bits.

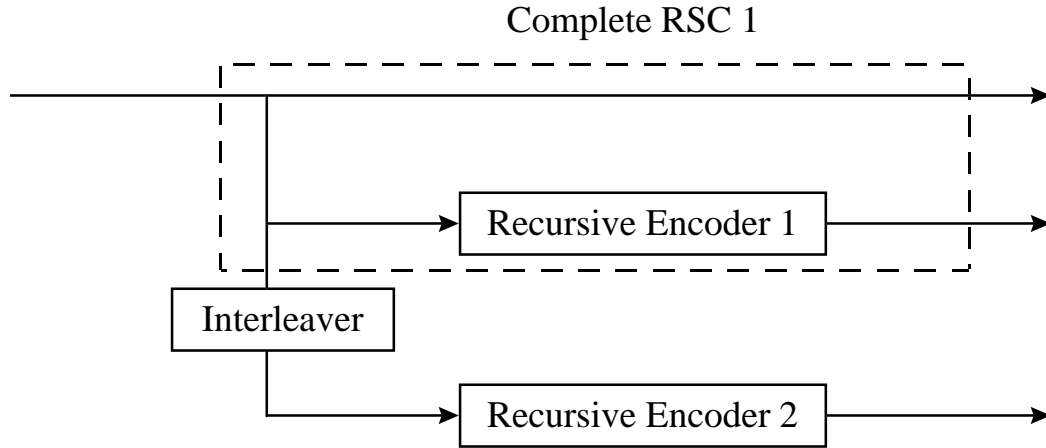


Figure 5.1: Termination of the systematic bit stream is associated with recursive encoder 1.

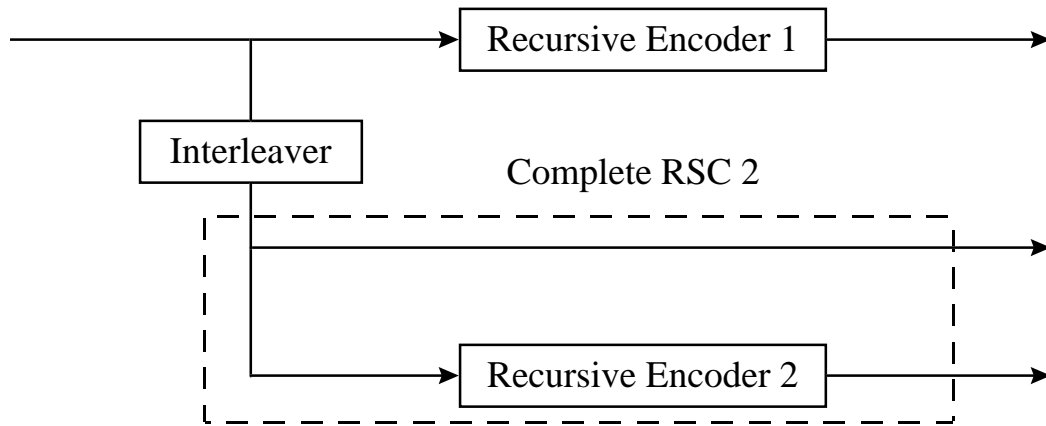


Figure 5.2: Termination of the systematic bit stream is associated with recursive encoder 2.

In the literature, the termination scheme of Figure 5.1 is commonly used. However, the literature also suggests acquisition of the decoded bit decisions at the second component decoder of the turbo code decoder. This is not advisable because the m tail bits for the systematic bit stream are associated with recursive encoder 1. As a result, it is possible that the final path chosen in the second component decoder is not the Maximum Likelihood (ML) path due to the erroneous m tail bits of the systematic bit stream. Thus, if the encoder structure of Figure 5.1 is used, then the decoded bit decisions should be acquired from the first component decoder. Also, if the encoder

structure of Figure 5.2 is used, then the decoded bit decisions should be acquired from the second component decoder. In the simulation, the encoder structure of Figure 5.2 is used.

In its basic form, the turbo code encoder is rate $1/3$. However, in many journal papers, the published computer simulations of turbo codes often use rate $1/2$. This is accomplished by puncturing the coded bit streams of the turbo code. The puncturing pattern is that for one coded bit stream, the odd bits are punctured out, and for the other coded bit stream, the even bits are punctured out.

In the simulation, the Gaussian channel model is used because it is a fairly good model for different transmission mediums. The Gaussian channel (Gaussian noise) is fairly easy to construct from the basic Gaussian distribution with mean of zero and standard deviation of one. In order to use this model, the turbo code encoder output bit streams must be mapped from $\{0,1\}$ to $\{-1,+1\}$ domain.

The simulation of the turbo code decoder is based on its description in Chapter 4. If the encoder structure of Figure 5.2 is used, then the systematic bit stream must be deinterleaved before passing to the first component decoder of the turbo code decoder. Also, the initial (first iteration) a priori values for the first component decoder are set to zero because there are no extrinsic information available from the second component decoder.

It has been found that the turbo code (SOVA) decoder, implemented as described in the literature, did not perform up to the published results. The cause of this performance degradation has been traced to the extrinsic/a priori information that is passed between the component decoders. After many computer simulation runs, it has been determined that the extrinsic/a priori information must be limited (reduced) before passing to the next component decoder. The two main strategies that were devised to reduce the extrinsic/a priori information are:

1. Introduce a limiter on the extrinsic/a priori information so that the values do not go over a predetermined bound.
2. Introduce a “numerical factor” to scale down the extrinsic/a priori information.

After extensive trials and tests, it has been determined that the most effective reduction technique was to introduce a numerical factor E_b/N_0 (signal to noise ratio per bit) to scale down the extrinsic/a priori information. This factor is idealistically reasonable in the sense that for low E_b/N_0 , the extrinsic/a priori values are relatively small so no drastic reduction is required. However, for high E_b/N_0 , the extrinsic/a priori values are relatively large so greater reduction is required. In the simulation, the scale down factor of E_b/N_0 is used on the extrinsic/a priori information.

5.2 Simulation Results

Simulation results for a turbo code are based on bit error rate (BER) performance over a range of E_b/N_0 . The following table shows the rate 1/2 component RSC codes used in the simulation results.

Table 5.1: Rate 1/2 RSC Component Codes Used in Simulation Results

Constraint Length (K)	Feedforward Generator (in Octal)	Feedback Generator (in Octal)
3	5	7
4	15	17
5	23	35
6	53	75

First, the performance of these rate 1/2 RSC codes is shown for soft decision Viterbi decoding. Then, the simulation results for two specific cases of a turbo code using SOVA decoding are compared to the published journal paper results [Jun96]. In addition, other turbo code simulation results are shown for different rate (R), constraint lengths (K), and frame sizes (FS) using the rate 1/2 RSC component codes. These results are first shown in table format and then in graphical format.

Table 5.2: Rate 1/2 RSC Code BER Performance in Soft Decision Viterbi Decoding

E_b/N_0 (dB)	Uncoded	K=3	K=4	K=5	K=6
1	5.7×10^{-2}	5.3×10^{-2}	4.2×10^{-2}	3.5×10^{-2}	2.9×10^{-2}
2	3.9×10^{-2}	2.0×10^{-2}	1.4×10^{-2}	8.9×10^{-3}	5.0×10^{-3}
3	2.4×10^{-2}	6.1×10^{-3}	2.7×10^{-3}	1.1×10^{-3}	8.0×10^{-4}
4	1.3×10^{-2}	1.0×10^{-3}	4.9×10^{-4}	1.8×10^{-4}	3.9×10^{-5}

Table 5.3: Comparison of Turbo Code BER Performance for 10 Decoding Iterations (R=1/2, K=3, FS=192)

E_b/N_0 (dB)	Uncoded	Simulated	Published
1	5.7×10^{-2}	4.5×10^{-2}	5.0×10^{-2}
2	3.9×10^{-2}	4.9×10^{-3}	4.3×10^{-3}
3	2.4×10^{-2}	5.9×10^{-4}	1.5×10^{-4}

Table 5.4: Comparison of Turbo Code BER Performance for 10 Decoding Iterations (R=1/2, K=4, FS=192)

Eb/No (dB)	Uncoded	Simulated	Published
1	5.7×10^{-2}	9.0×10^{-2}	7.0×10^{-2}
2	3.9×10^{-2}	6.1×10^{-3}	4.5×10^{-3}
3	2.4×10^{-2}	1.7×10^{-4}	5.4×10^{-5}

Table 5.5: Turbo Code BER Performance for 10 Decoding Iterations (R=1/2, K=3)

Eb/No (dB)	Uncoded	FS=50	FS=192	FS=1000
1	5.7×10^{-2}	4.9×10^{-2}	4.5×10^{-2}	4.1×10^{-2}
2	3.9×10^{-2}	1.8×10^{-2}	4.9×10^{-3}	3.9×10^{-4}
3	2.4×10^{-2}	3.6×10^{-3}	5.9×10^{-4}	-
4	1.3×10^{-2}	5.9×10^{-4}	6.8×10^{-5}	-

Table 5.6: Turbo Code BER Performance for 10 Decoding Iterations (R=1/3, K=3)

Eb/No (dB)	Uncoded	FS=50	FS=192	FS=1000
1	5.7×10^{-2}	5.8×10^{-2}	1.7×10^{-2}	1.5×10^{-3}
2	3.9×10^{-2}	1.8×10^{-2}	1.8×10^{-3}	1.3×10^{-4}
3	2.4×10^{-2}	2.3×10^{-3}	1.7×10^{-4}	-
4	1.3×10^{-2}	4.0×10^{-4}	3.5×10^{-5}	-

Table 5.7: Turbo Code BER Performance for 10 Decoding Iterations (R=1/2, K=4)

Eb/No (dB)	Uncoded	FS=50	FS=192
1	5.7×10^{-2}	7.8×10^{-2}	9.0×10^{-2}
2	3.9×10^{-2}	2.8×10^{-2}	6.1×10^{-3}
3	2.4×10^{-2}	2.9×10^{-3}	1.7×10^{-4}

Table 5.8: Turbo Code BER Performance for 10 Decoding Iterations (FS=50)

Eb/No (dB)	Uncoded	R=1/2, K=3	R=1/3, K=3	R=1/2, K=4
1	5.7×10^{-2}	4.9×10^{-2}	5.8×10^{-2}	7.8×10^{-2}
2	3.9×10^{-2}	1.8×10^{-2}	1.8×10^{-2}	2.8×10^{-2}
3	2.4×10^{-2}	3.6×10^{-3}	2.3×10^{-3}	2.9×10^{-3}
4	1.3×10^{-2}	5.9×10^{-4}	4.0×10^{-4}	-

Table 5.9: Turbo Code BER Performance for 10 Decoding Iterations (FS=192)

Eb/No (dB)	Uncoded	R=1/2, K=3	R=1/3, K=3	R=1/2, K=4
1	5.7×10^{-2}	4.5×10^{-2}	1.7×10^{-2}	9.0×10^{-2}
2	3.9×10^{-2}	4.9×10^{-3}	1.8×10^{-3}	6.1×10^{-3}
3	2.4×10^{-2}	5.9×10^{-4}	1.7×10^{-4}	1.7×10^{-4}
4	1.3×10^{-2}	6.8×10^{-5}	3.5×10^{-5}	-

Table 5.10: Turbo Code BER Performance for 10 Decoding Iterations (FS=1000)

Eb/No (dB)	Uncoded	R=1/2, K=3	R=1/3, K=3
1	5.7×10^{-2}	4.1×10^{-2}	1.5×10^{-3}
1.5	4.6×10^{-2}	3.5×10^{-3}	3.8×10^{-4}
2	3.9×10^{-2}	3.9×10^{-4}	1.3×10^{-4}

Table 5.11: Turbo Code BER Performance Gain (R=1/2, K=3)

Eb/No (dB)	Uncoded	FS=50 Iter=1	FS=50 Iter=10	FS=192 Iter=1	FS=192 Iter=10	FS=1000 Iter=1	FS=1000 Iter=10
1	5.7×10^{-2}	6.7×10^{-2}	4.9×10^{-2}	8.5×10^{-2}	4.5×10^{-2}	9×10^{-2}	4.1×10^{-3}
2	3.9×10^{-2}	4.0×10^{-2}	1.8×10^{-2}	3.5×10^{-2}	4.9×10^{-3}	2.7×10^{-2}	3.9×10^{-4}
3	2.4×10^{-2}	7.9×10^{-3}	3.6×10^{-3}	7.0×10^{-3}	5.9×10^{-4}		
4	1.3×10^{-2}	1.8×10^{-3}	5.9×10^{-4}	1.2×10^{-3}	6.8×10^{-5}		

Table 5.12: Turbo Code BER Performance Gain (R=1/3, K=3)

Eb/No (dB)	Uncoded	FS=50 Iter=1	FS=50 Iter=10	FS=192 Iter=1	FS=192 Iter=10	FS=1000 Iter=1	FS=1000 Iter=10
1	5.7×10^{-2}	6.1×10^{-2}	5.8×10^{-2}	6.5×10^{-2}	1.7×10^{-2}	7.5×10^{-2}	1.5×10^{-3}
2	3.9×10^{-2}	3.9×10^{-2}	1.8×10^{-2}	2.7×10^{-2}	1.8×10^{-3}	2.1×10^{-2}	1.3×10^{-4}
3	2.4×10^{-2}	7.3×10^{-3}	2.3×10^{-3}	4.4×10^{-3}	1.7×10^{-4}		
4	1.3×10^{-2}	1.0×10^{-3}	4.0×10^{-4}	5.0×10^{-4}	3.5×10^{-5}		

Table 5.13: Turbo Code BER Performance Gain (R=1/2, K=4)

Eb/No (dB)	Uncoded	FS=50 Iter=1	FS=50 Iter=10	FS=192 Iter=1	FS=192 Iter=10
1	5.7×10^{-2}	9.2×10^{-2}	7.8×10^{-2}	1.0×10^{-1}	9.0×10^{-2}
2	3.9×10^{-2}	4.8×10^{-2}	2.8×10^{-2}	3.4×10^{-2}	6.1×10^{-3}
3	2.4×10^{-2}	7.0×10^{-3}	2.9×10^{-3}	5.5×10^{-3}	1.7×10^{-4}

Rate 1/2 RSC Code BER Performance in Soft Decision Viterbi Decoding

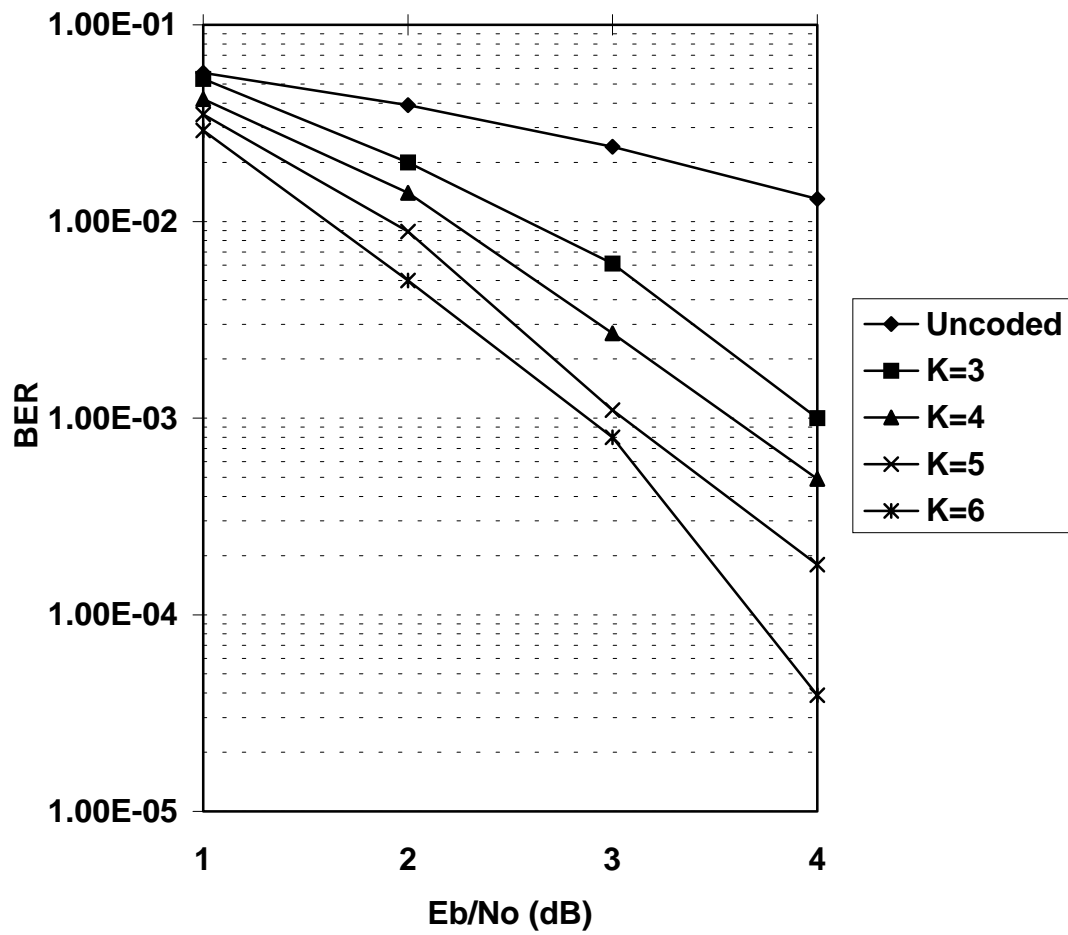


Figure 5.3: Rate 1/2 RSC code BER performance in soft-decision Viterbi decoding.

**Comparison of Turbo Code BER Performance
for 10 Decoding Iterations (R=1/2, K=3, FS=192)**

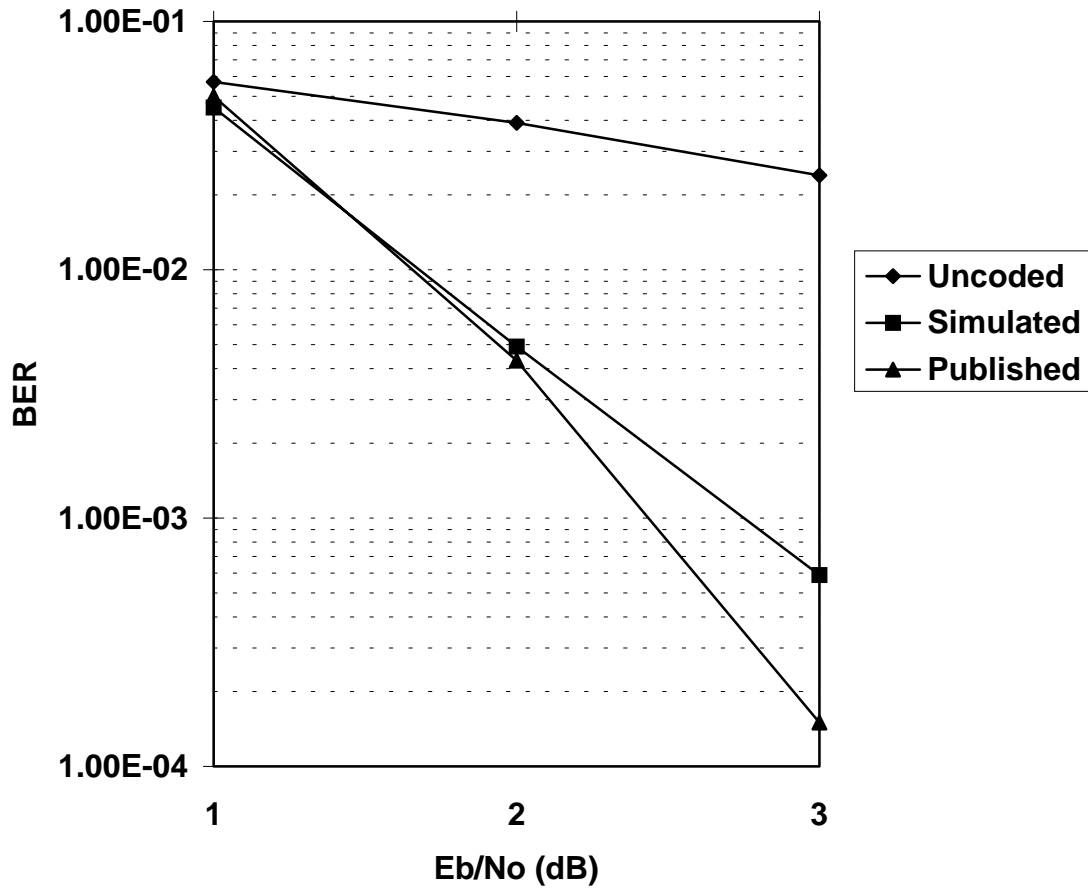


Figure 5.4: Comparison of turbo code BER performance for 10 decoding iterations (R=1/2, K=3, FS=192).

**Comparison of Turbo Code BER Performance
for 10 Decoding Iterations (R=1/2, K=4, FS=192)**

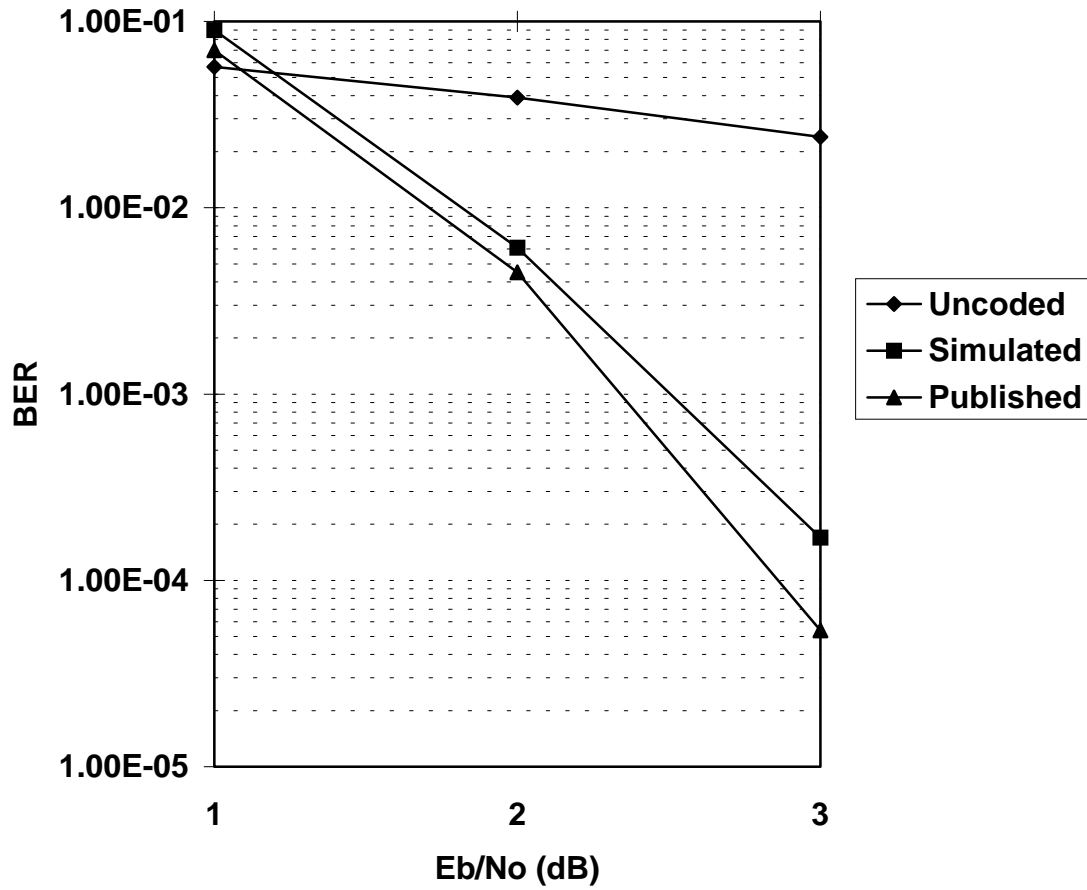


Figure 5.5: Comparison of turbo code BER performance for 10 decoding iterations (R=1/2, K=4, FS=192).

Turbo Code BER Performance for 10 Decoding Iterations
($R=1/2$, $K=3$)

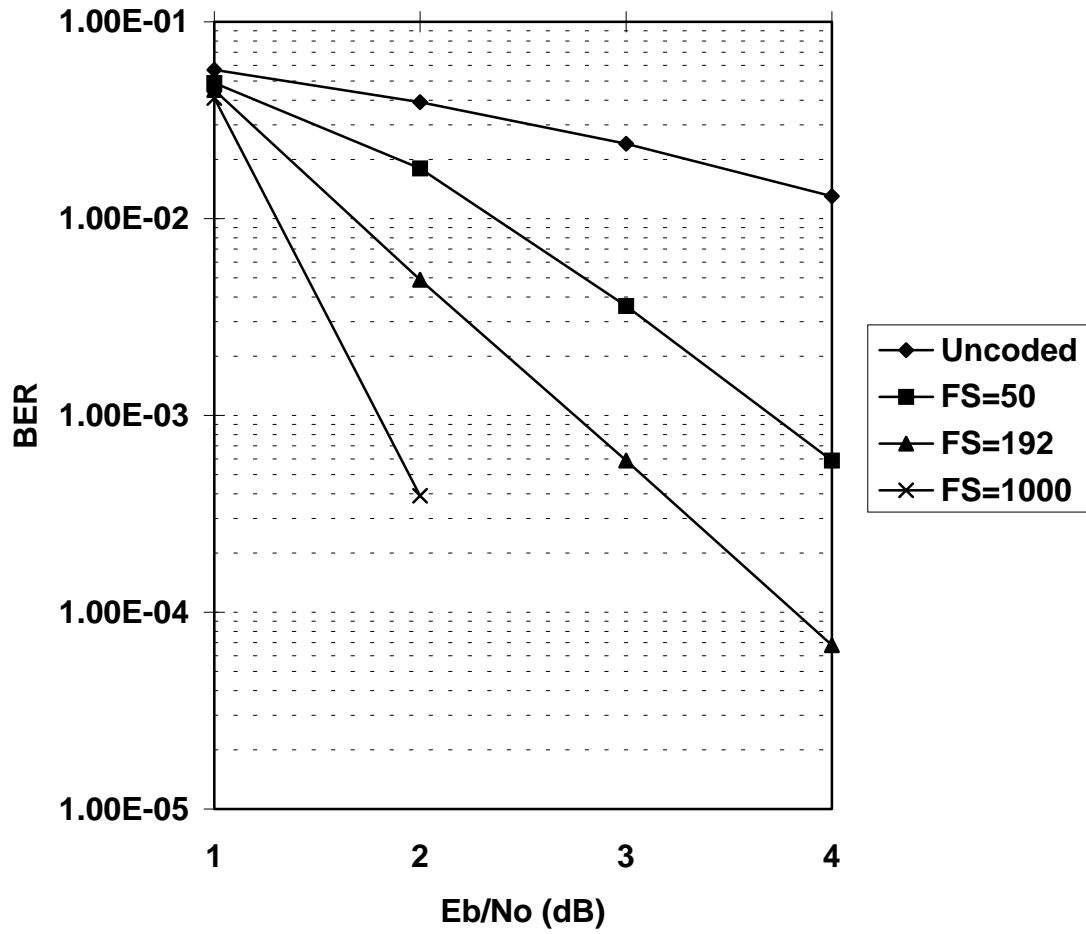


Figure 5.6: Turbo code BER performance for 10 decoding iterations ($R=1/2$, $K=3$).

Turbo Code BER Performance for 10 Decoding Iterations
($R=1/3$, $K=3$)

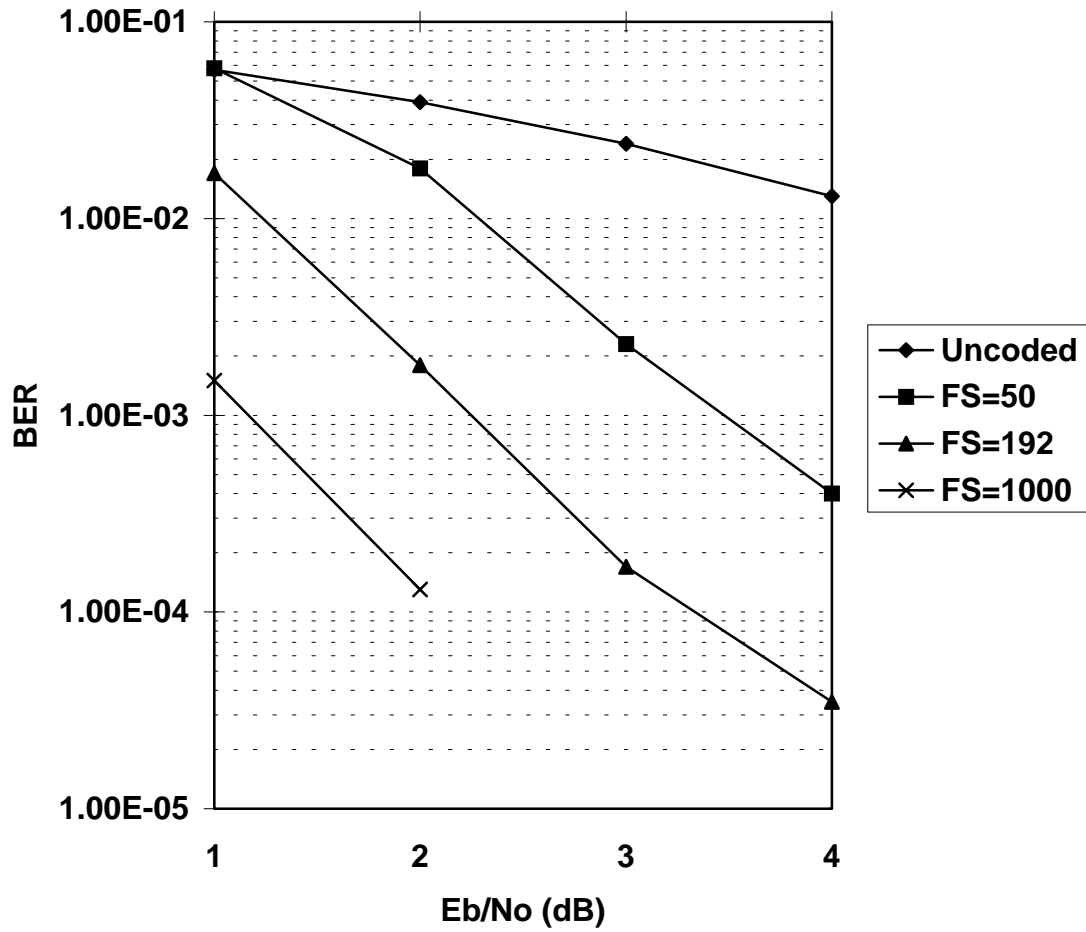


Figure 5.7: Turbo code BER performance for 10 decoding iterations ($R=1/3$, $K=3$).

Turbo Code BER Performance for 10 Decoding Iterations
($R=1/2$, $K=4$)

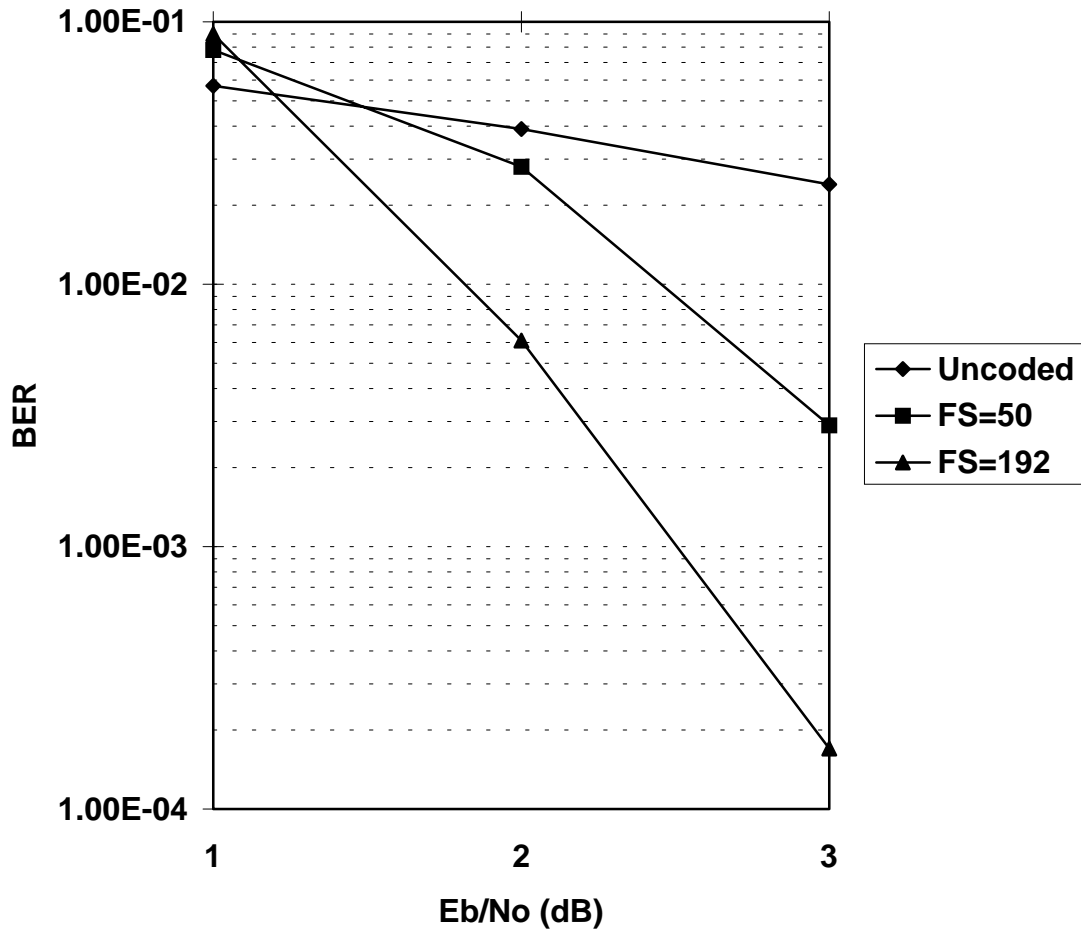


Figure 5.8: Turbo code BER performance for 10 decoding iterations ($R=1/2$, $K=4$).

Turbo Code BER Performance for 10 Decoding Iterations
(FS=50)

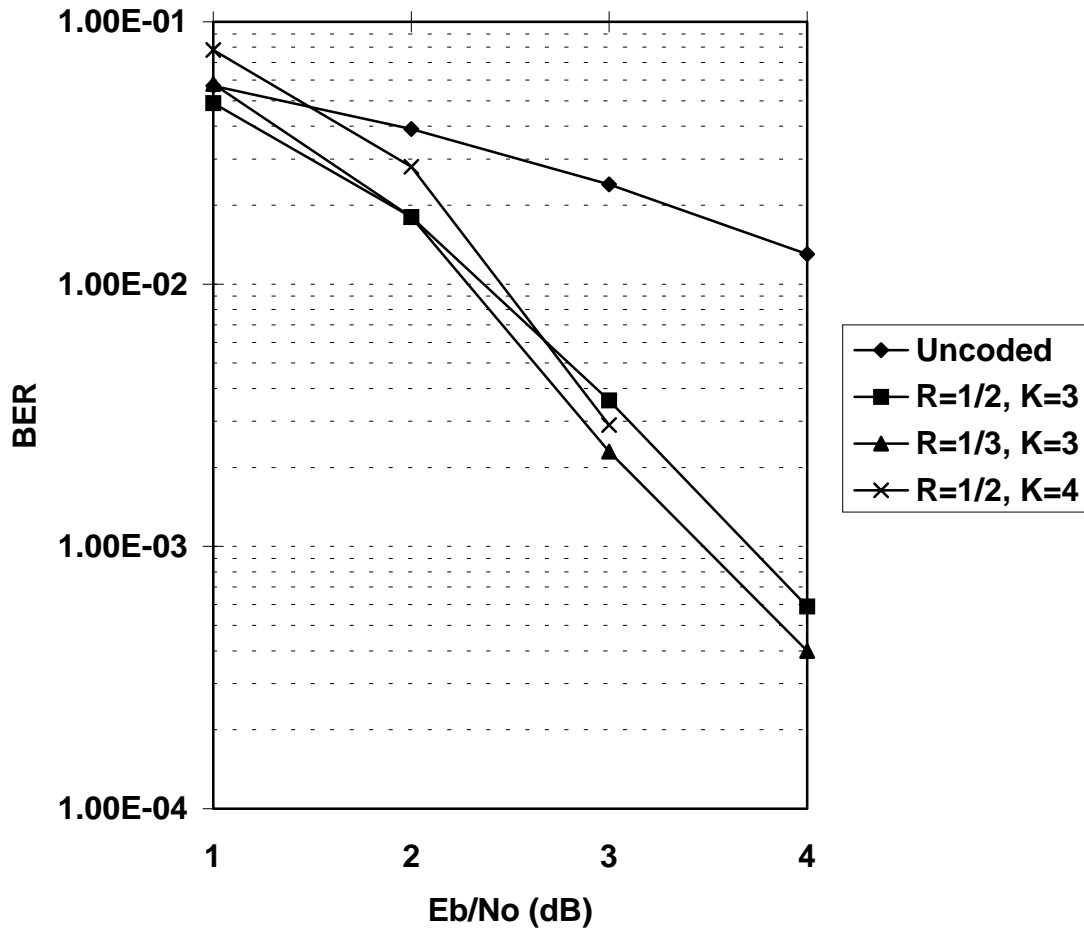


Figure 5.9: Turbo code BER performance for 10 decoding iterations (FS=50).

**Turbo Code BER Performance for 10 Decoding Iterations
(FS=192)**

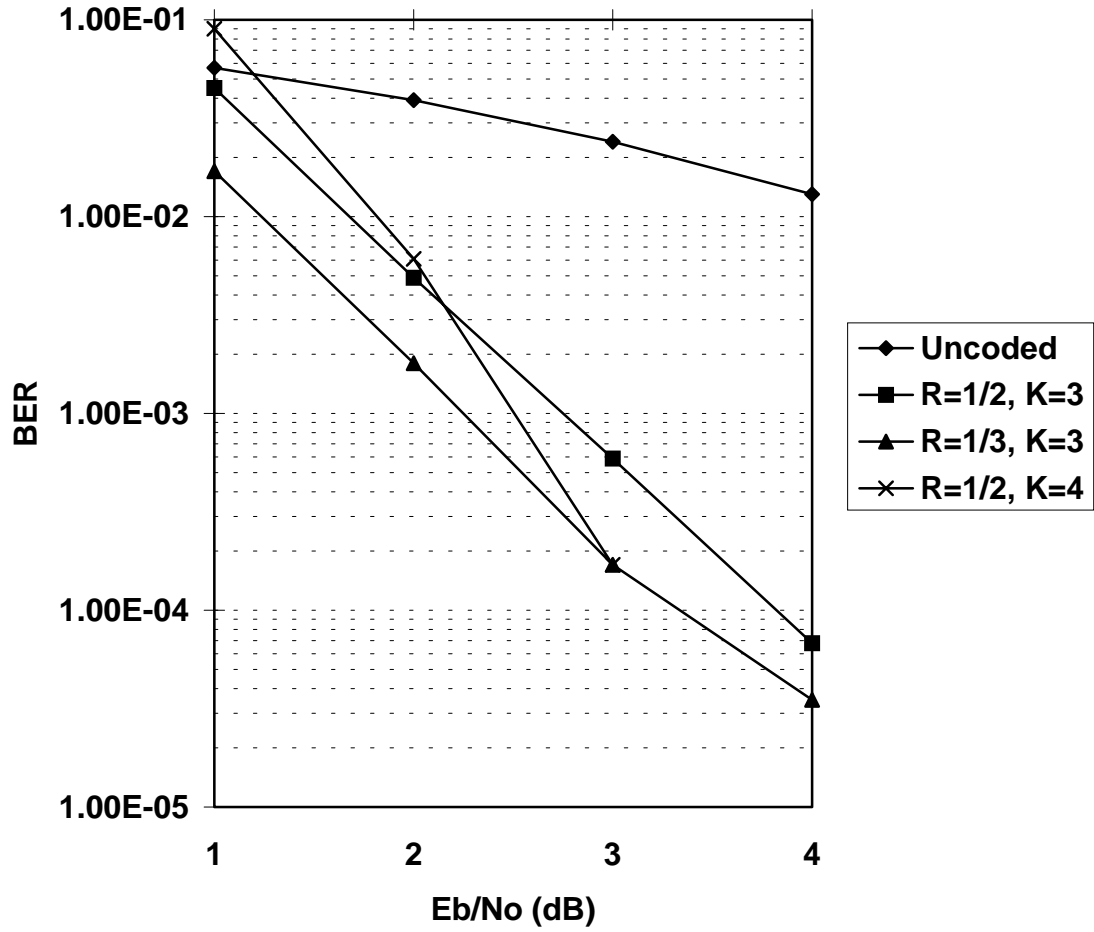


Figure 5.10: Turbo code BER performance for 10 decoding iterations (FS=192).

**Turbo Code BER Performance for 10 Decoding Iterations
(FS=1000)**

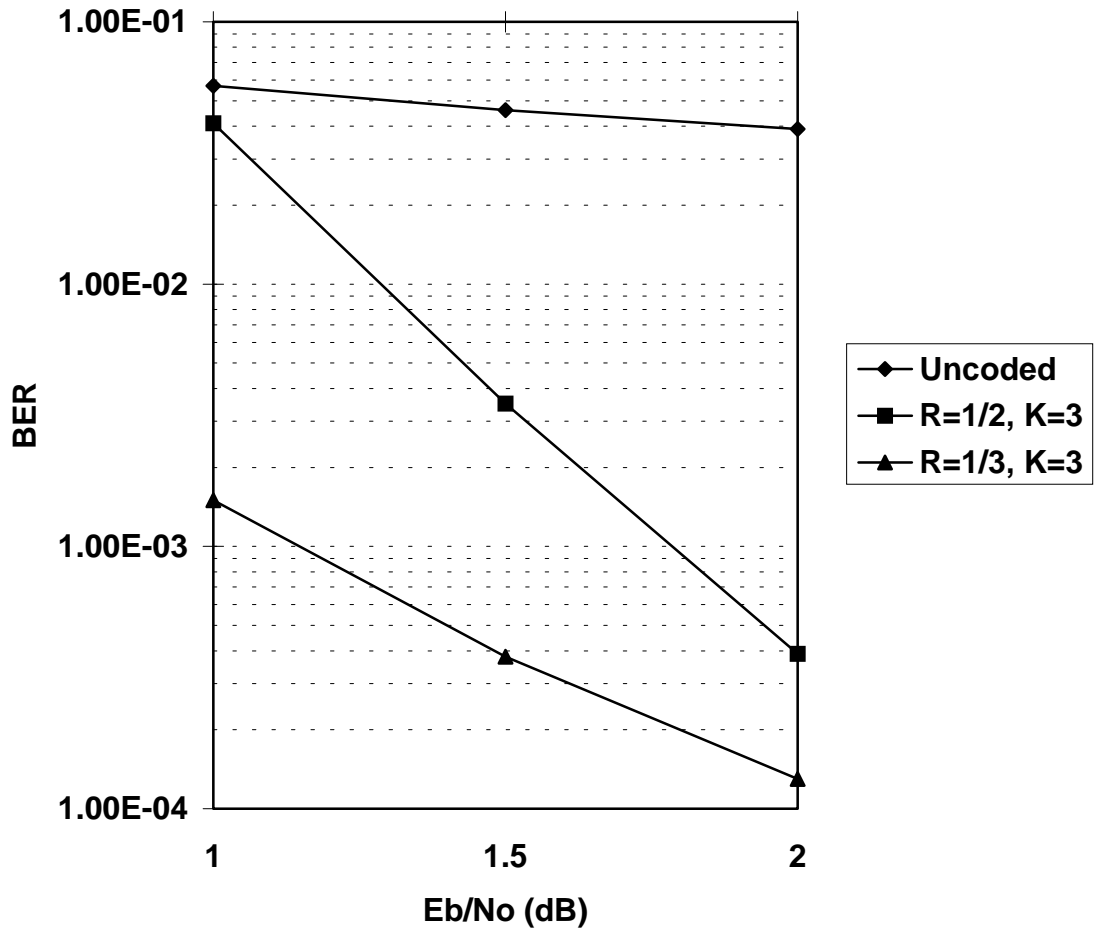


Figure 5.11: Turbo code BER performance for 10 decoding iterations (FS=1000).

Turbo Code BER Performance Gain (R=1/2, K=3)

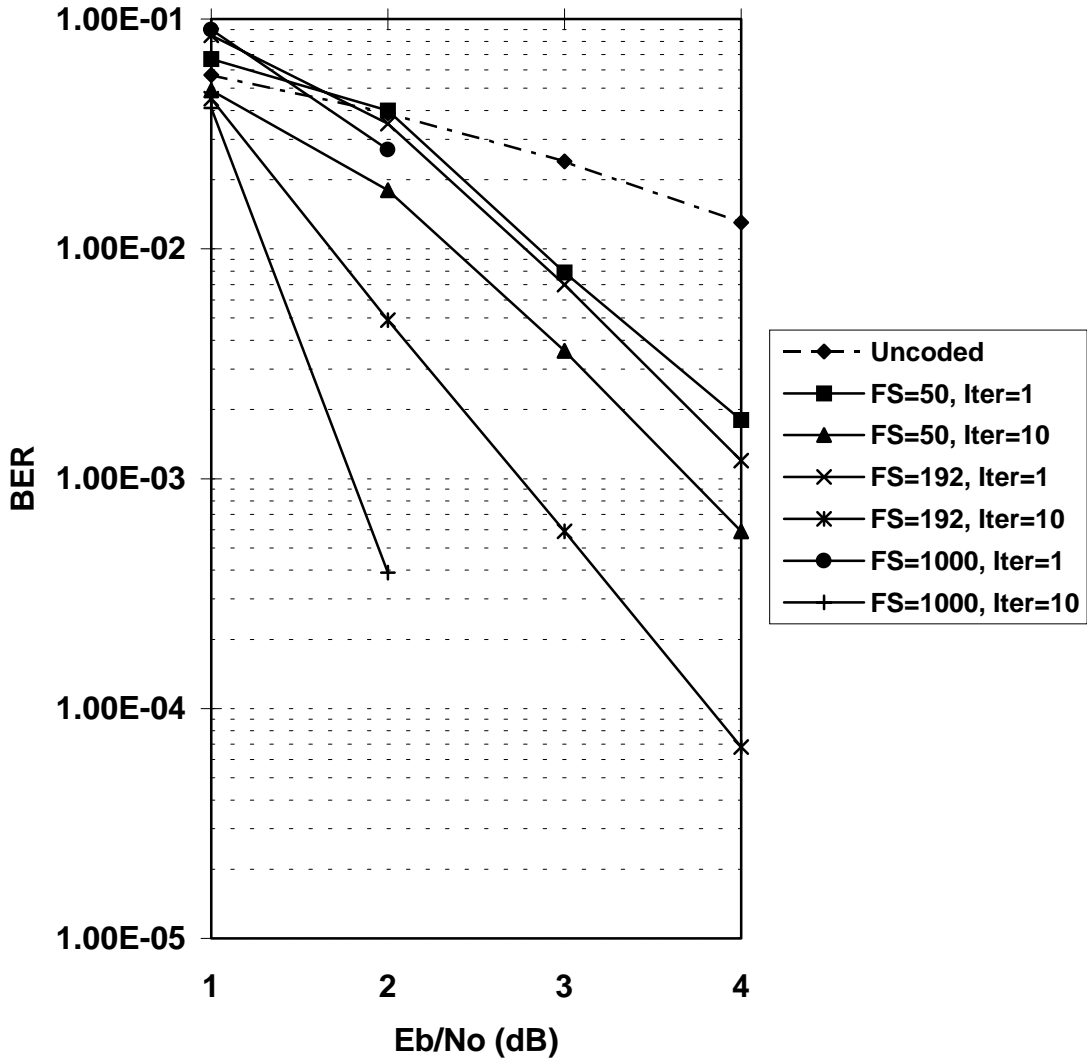


Figure 5.12: Turbo code BER performance gain (R=1/2, K=3).

Turbo Code BER Performance Gain (R=1/3, K=3)

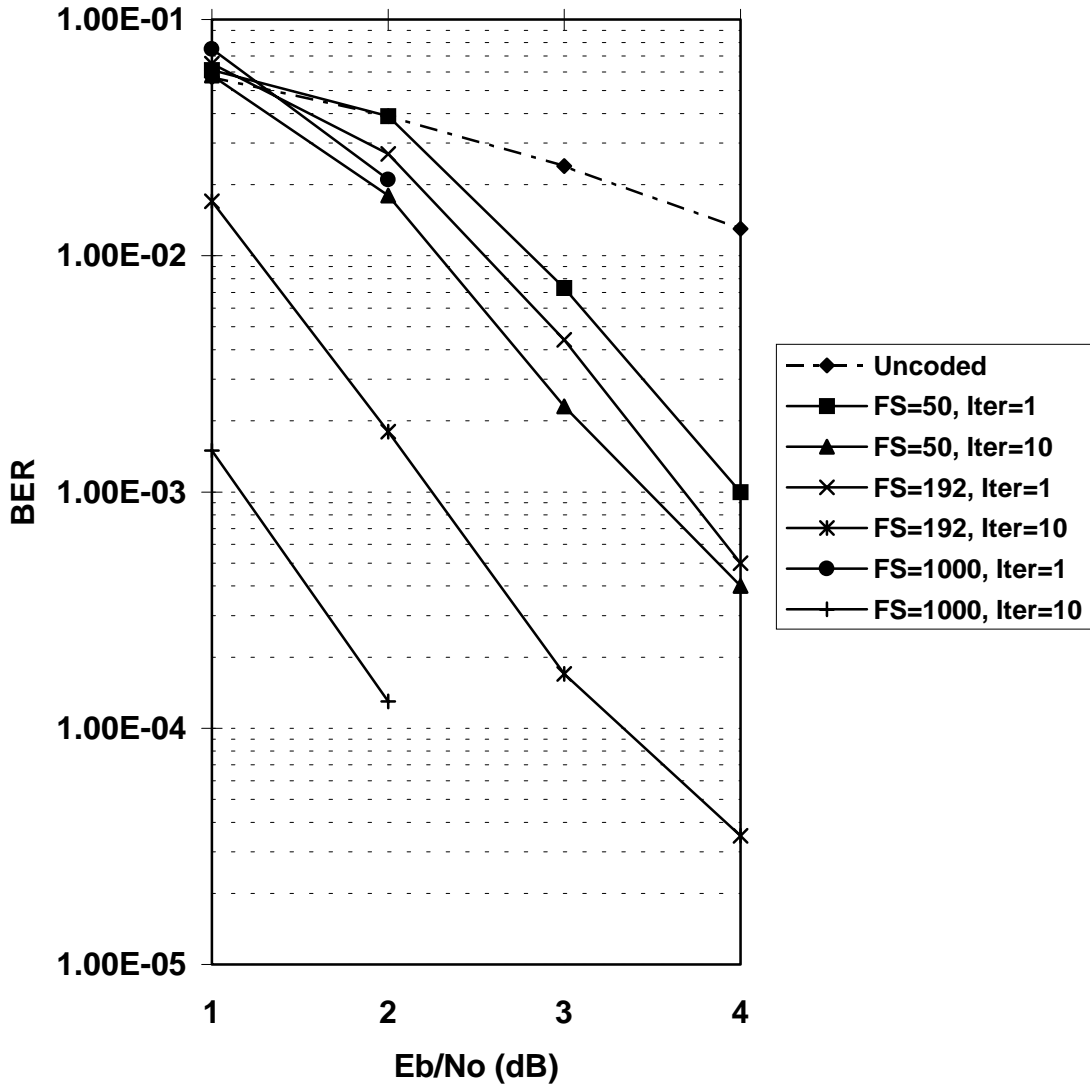


Figure 5.13: Turbo code BER performance gain (R=1/3, K=3).

Turbo Code BER Performance Gain (R=1/2, K=4)

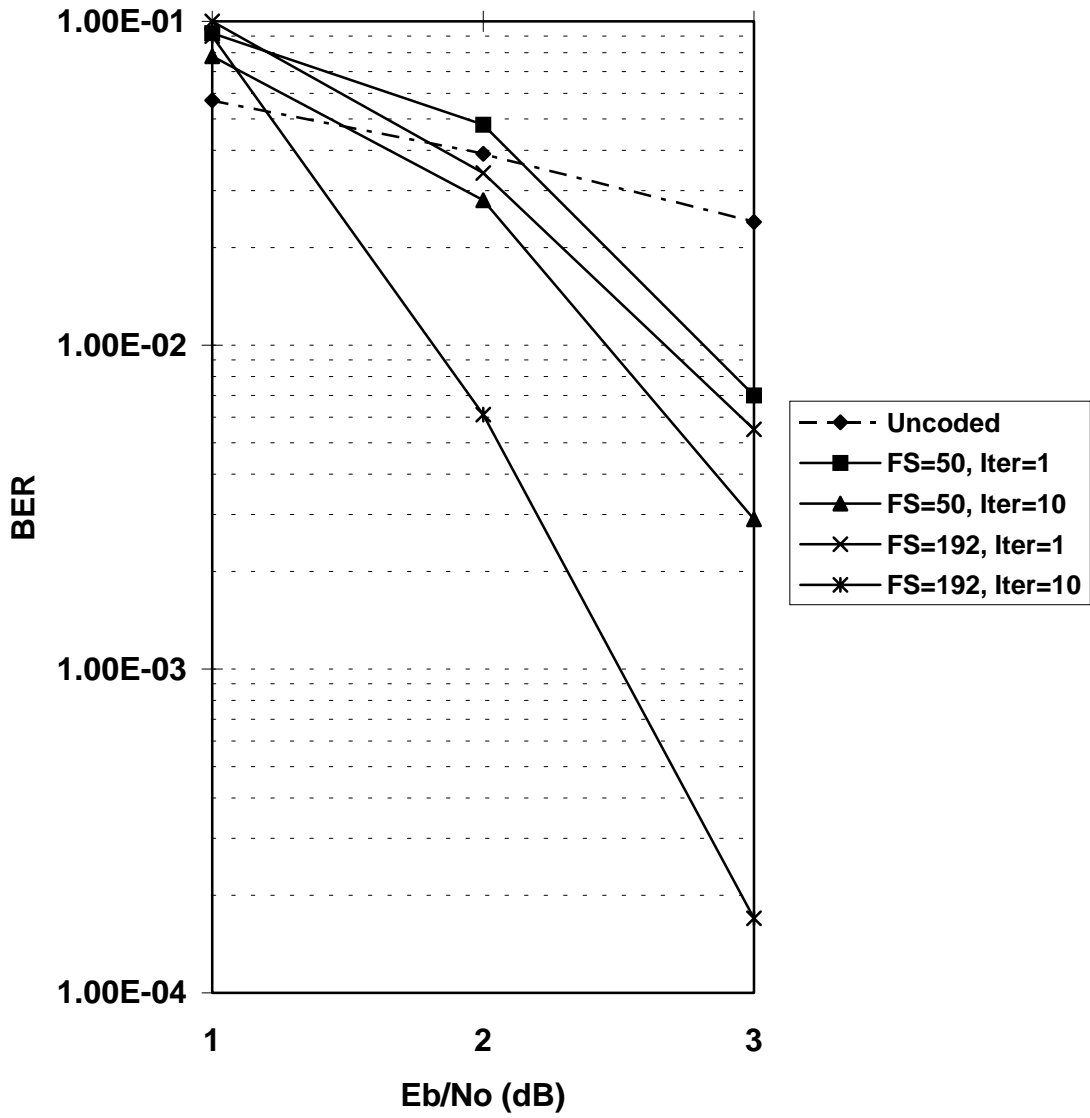


Figure 5.14: Turbo code BER performance gain (R=1/2, K=4).

5.3 Simulation Analysis

Before analyzing the performance of a turbo code, the performance of its rate 1/2 RSC component code is first examined. The performance of the rate 1/2 RSC code in soft decision Viterbi decoding for different constraint lengths is shown in Figure 5.3. In this figure, it can be seen that as the constraint length increases, the performance of the code also increases, resulting in lower BER. This is the typical characteristic of any convolutional code.

To validate the turbo code simulation, Figure 5.4 and Figure 5.5 show the simulated and published performance results for two specific cases. In Figure 5.4, the difference between the simulated and published performance results is negligible for low E_b/N_0 (<2), but for higher E_b/N_0 (>2), this difference becomes greater. This can also be seen in Figure 5.5. However, the simulated turbo code performance results shown in this thesis were independently validated by another researcher, Matthew Valenti, at Virginia Tech (via personal communications).

The difference between the simulated and published performance results are unknown. This problem is believed to be caused by two independent factors. First, it has been suggested that the SOVA decoding algorithm is very sensitive to numerical inaccuracies. The computer may introduce round off error in many of its calculations such that the final output result is not very precise compared to the true value. This may be the case for BER at high E_b/N_0 . As can be seen from Figure 5.4 and 5.5, the relatively significant difference between the simulated and published performance results is located at very low BER. At this low BER, a few bit errors caused by computer inaccuracies will be magnified greatly.

It is speculated that the source of the round off error originates from the iterative decoding process. As small errors are introduced from one decoding iteration, it will be accumulated with other errors from previous decoding iterations. This error may grow as the number of iterations increases. It should be noted that most of the published turbo code simulation results were carried out on supercomputers.

The second factor that may cause the difference between the simulated and published performance results comes from the lack of detail in the SOVA decoding algorithm. Two researchers, J. Hagenauer and P. Jung (listed in the reference), who are well known for their work on turbo codes, were contacted and questioned via personal communications about the details of the SOVA decoding algorithm. However, their help did not resolve the discrepancy between the simulated and published performance results.

The simulated performance results of turbo codes with the same component codes but different frame sizes are shown in Figure 5.6, Figure 5.7, and Figure 5.8. As shown

in these figures, the turbo code performance improves as the frame size increases. This conclusion can be expected because the larger the frame size, the more accurate are the reliability estimations in the SOVA decoding algorithm.

The simulated performance results of turbo codes with fixed frame sizes but different component codes and rates are shown in Figure 5.9, Figure 5.10, and Figure 5.11. From these figures, it can be seen that for a fixed constraint length, a decrease in code rate increases the turbo code performance. It can also be seen that for a fixed code rate, an increase in constraint length improves the turbo code performance. These conclusions are similar to that for convolutional codes.

The overall iterative (10 iterations) decoding gain for a turbo code with the same component codes and rates but different frame sizes are shown in Figure 5.12, Figure 5.13, and Figure 5.14. As shown in these figures, the overall iterative decoding gain increases as the frame size increases. This and the above mentioned characteristics of the simulated turbo code performance results are in the same direction with current published research work.

5.4 Comparison of Turbo Code Decoding and Viterbi Decoding

Turbo code decoding and Viterbi decoding are often compared in terms of bit error rate (BER) performance and computational complexity. In Figure 5.3, BER performance of soft-decision Viterbi decoding for rate 1/2 RSC code is shown. This figure shows that Viterbi decoding is effective for large constraint length codes (e.g. $K=5$ or $K=6$). However, this is not the case for turbo code decoding. From Figures 5.6, 5.7, and 5.8, it can be seen that turbo code decoding is able to provide comparable or better BER performance for small constraint length codes (e.g. $K=3$ or $K=4$) than Viterbi decoding of large constraint length codes (e.g. $K=5$ or $K=6$) under the condition that relatively large frame sizes ($FS \geq 192$ bits) are used.

Fairly high computational complexity is required for turbo code decoding. The SOVA component decoder is approximately two times as complex as the Viterbi decoder (Figure 4.11). Turbo code decoding requires two SOVA component decoders; thus, one iteration of turbo code decoding requires complexity up to four times that of the Viterbi decoder (Figure 4.12). As one can see, the complexity of the turbo code decoder dramatically increases as the number of decoding iterations increases. In this chapter, most of the turbo code simulation results are based on ten decoding iterations, which are approximately equivalent to complexity up to forty times that of Viterbi decoding. Typically, turbo code simulations require up to three days for frame size of 192 bits and over a week for frame size of 1000 bits running MATLAB on the background of SUN SPARC 10 workstations. Due to the decoding complexity of turbo code, most of the

BER performance results are for component code with small constraint lengths, high code rates, and short frame sizes.