A Study on Simulating Convolutional Codes and Turbo Codes

Final Report

By Daniel Chang

July 27, 2001

Advisor: Dr. P. Kinman

Executive Summary

This project includes the design of simulations of several convolutional codes and a turbo code using component convolutional codes. These error correction codes allow a message to be sent with less power over a noisy channel with the same number of bit errors as an uncoded message sent with higher power over the same channel. This is important because it allows for the reduction of power consumption in many communication systems.

Introduction

In most communication systems there is noise in the channel between the transmitter and receiver. As a result, the received signal is different from the transmitted signal. To remedy this problem, error-correction codes, such as convolutional codes and turbo codes, are used. At the channel encoder, these codes add redundancy to a message prior to being transmitted. After Additive White Gaussian Noise (AWGN) is added to the encoded message in the communication channel, the channel decoder uses the known redundancy to determine the correct message. This project includes the design of several simulations of convolutional codes and a turbo code in a channel model.

Methodology

The simulation of the channel model adds zero-mean AWGN with a given variance to a binary message signal that has been converted to an analog signal. The variance of the AWGN in the channel model is given by

$$\sigma^2 = \frac{1}{2 \cdot \frac{E_s}{N_0}}$$

Where the energy per symbol, E_s, is equal to the energy per bit, $E_{b,}$ for an uncoded channel.

The theoretical performance of an uncoded channel with antipodal signaling and AWGN, in terms of Bit Error Rate (BER) and E_b/N_0 is

$$BER = \frac{1}{2} erfc(\sqrt{E_b / N_0})$$

The 8 state RSC encoder is

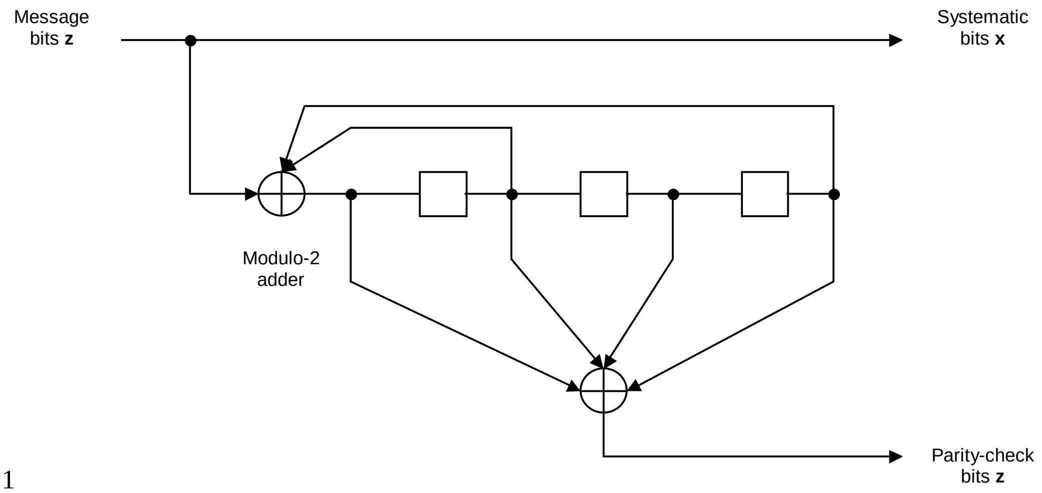

Figure 1

Note that the initial state of the RSC is set to 000.

After the last message bit is sent to the RSC encoder, the RSC changes to the following:

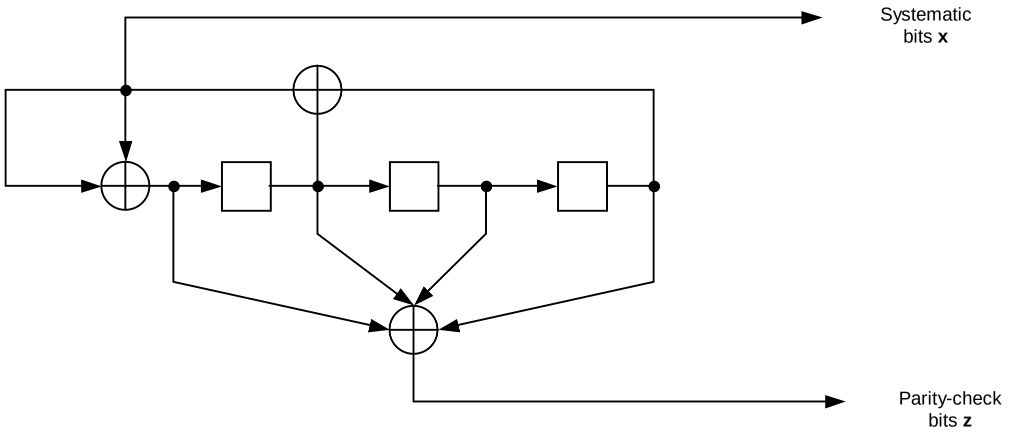

Figure 2.

Then, the RSC cycles three more times. The resulting systematic and parity bits are known as tail bits, and they ensure that the end state of the RSC is 000. The initial and end states of the RSC are both set to 000 because the Viterbi decoding algorithm is more efficient when the initial and end states are known.

For the unpunctured codes, this 8 state RSC is $r = 1/2$. For the punctured code, on odd message bits, the systematic and parity bits are both sent on the channel. However, on even bits only the systematic bit is sent. As a result, the punctured code is rate $r = 2/3$.

In the simulation, $E_b = 2E_s$ for the $r = 1/2$ RSC code. Likewise, $E_b = 3/2 \bullet E_s$ for the $r = 2/3$ RSC code.

The state diagram for the RSC encoder is

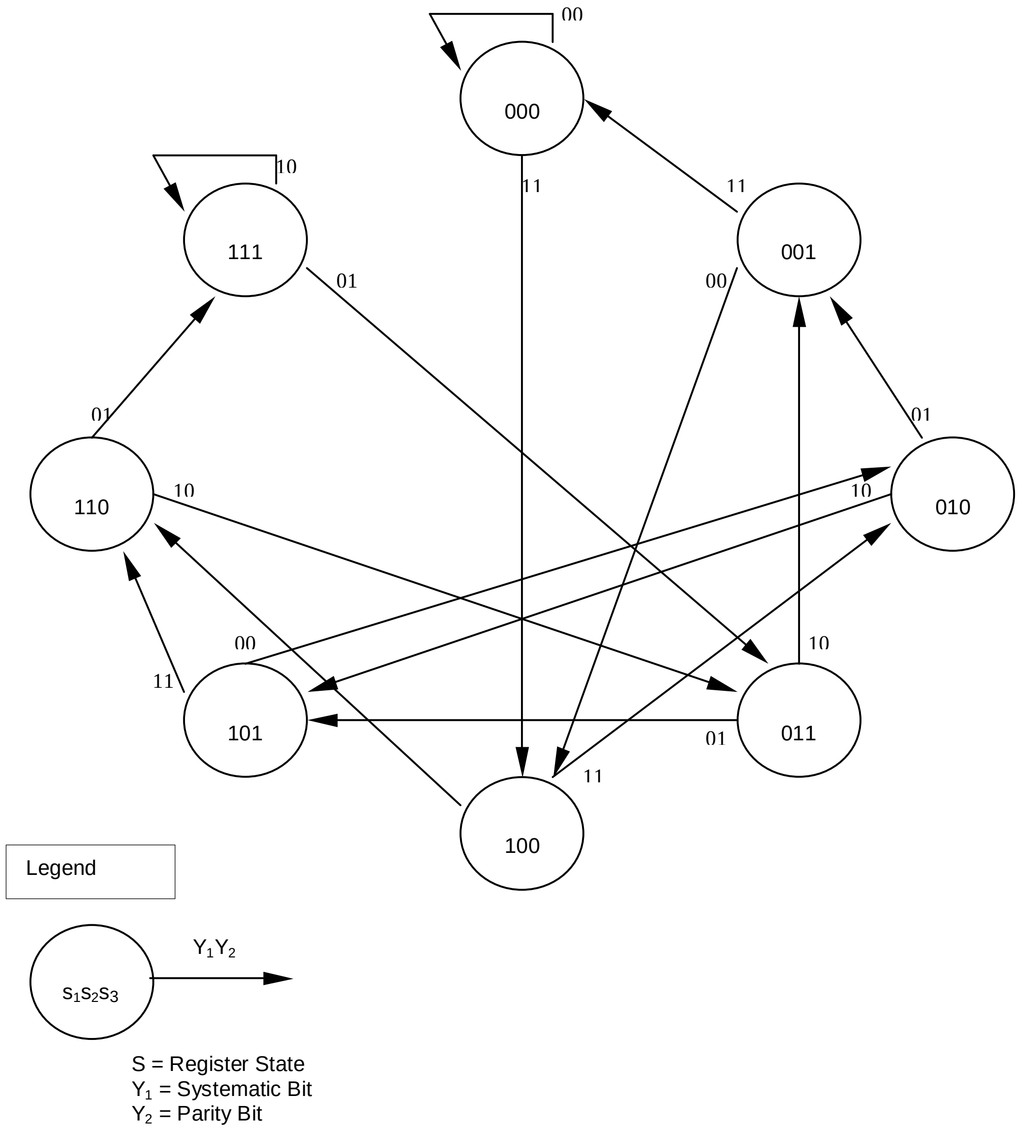

Figure 3.

The Viterbi decoders for the different RSC codes use the same trellis diagram. There are only two possible states that a given state can transition from, and the output symbols from those states are complementary.

The basic trellis is

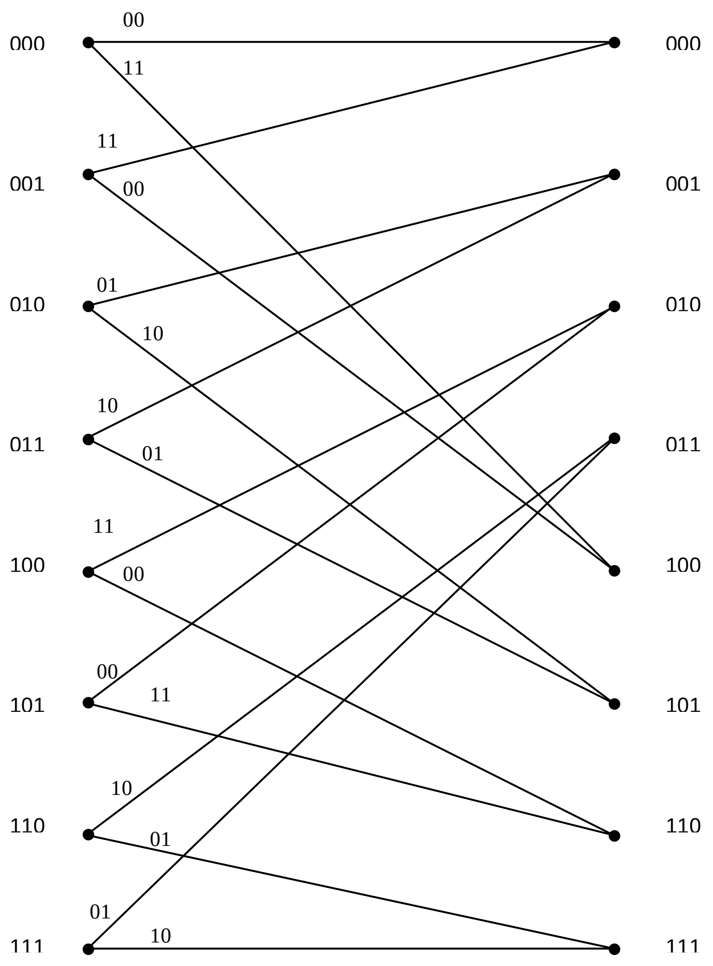

Figure 4.

For the punctured RSC, on even bits, when only the systematic bit is sent, the trellis is the same with the exception that a placeholder replaces the second output symbol, or parity bit.

The Viterbi algorithm is an efficient method of remembering the best path, based on minimum distance, through the trellis into each new column of nodes.

In the Viterbi algorithm with soft decisions, the symbols with added noise are used in the decoding algorithm. Euclidean distance is calculated here. However, with hard decisions, the symbols with added noise are first converted to binary symbols before they are decoded. If the symbol with added noise is < 0, it becomes 0. Otherwise, it becomes 1. Hamming distance is calculated here. Since some information is lost here, we can expect that the Viterbi algorithm with hard decisions is not as efficient as with soft decisions. Likewise, in the punctured case, since only half of the parity bits are sent, we can also expect the punctured code to be less efficient than the unpunctured code. It is important to note, however, that the noise per symbol is less in the punctured code. The result is that we can expect the punctured code to be only slightly less efficient then the unpunctured case.

The turbo code is given by

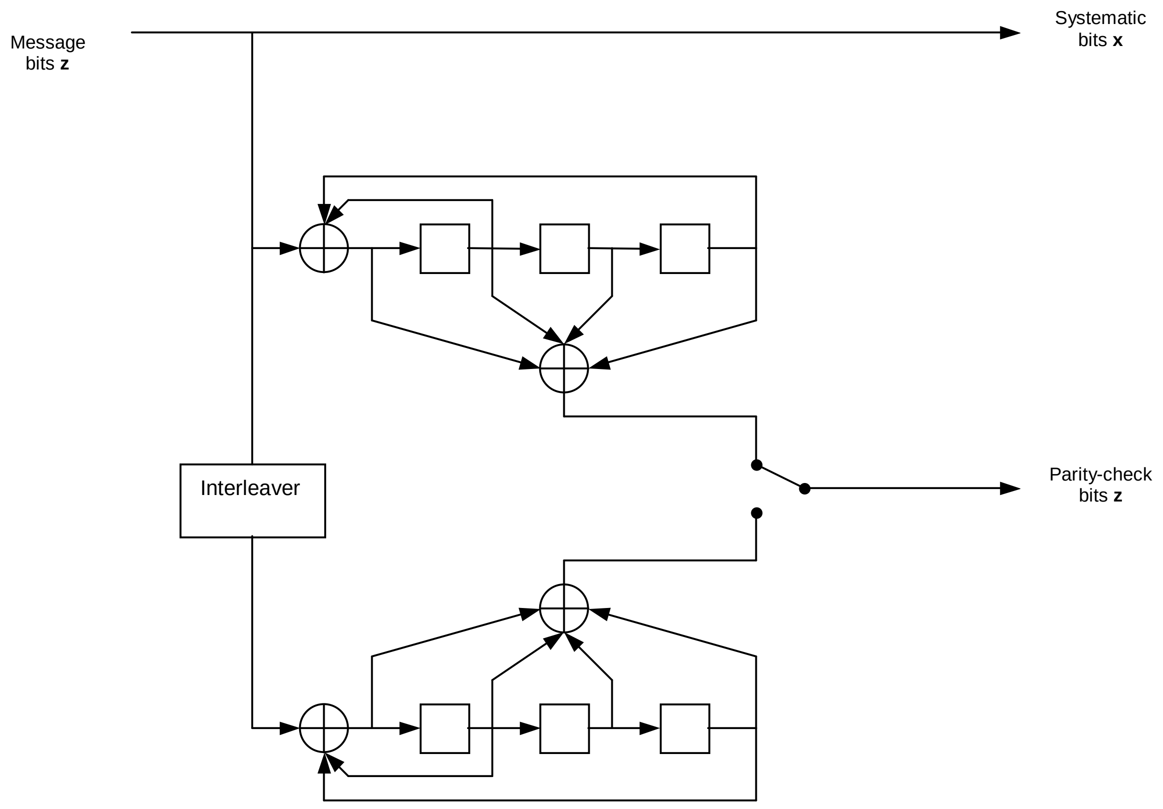

Figure 5.

Note that there are two component 8 state RSC's in this turbo code. The interleaver is a block interleaver with an odd number of rows and an odd number of columns. This design allows for each systematic bit to have a corresponding parity bit that is also sent through the channel. Only one parity bit from one of the component RSC's is sent with each systematic bit. On the next bit, the parity bit from the other RSC is sent. As a result, this code is rate 1/2.

The turbo-code decoder is given by:

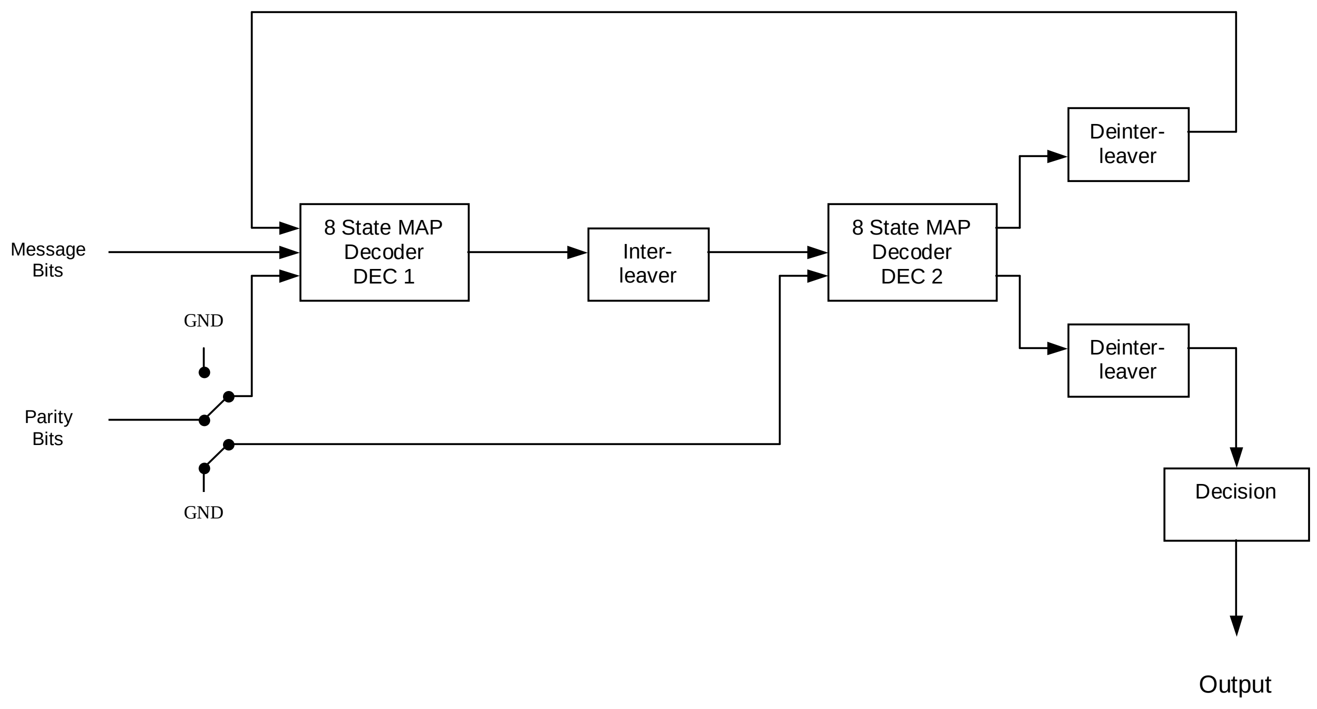

Figure 6.

This decoder does not use component Viterbi algorithm decoders. Although soft-output Viterbi algorithm decoders may be used in turbo codes, symbol-by-symbol maximum a priori (MAP) decoders are used in this design. Note that the interleaver here is the same as the interleaver shown in the encoder. Also, the deinterleavers here correspond to the given interleaver. Note that the message bits are added to the output of decoder #1 and are interleaved and sent to decoder #2. Finally, the turbo-code decoder has an iterative process. With each iteration in the feedback loop, the performance of the code is increased. New information calculated in the previous half-iteration is used in the decoding of the current half-iteration to make better decisions about the original message.

According William R. Ryan in an unpublished paper entitled, “A Turbo Code Tutorial”, the MAP decoder calculates the log likelihood that the encoder is at a certain state at a given time. First,γ’s are calculated for each possible transition from node, s, to node s’ at each time, k, using the following equation

$$\gamma_k(s',s) = \exp[\frac{1}{2}u_k(L^e(u_k)+L_c y^s_k)]\cdot\gamma^e_k(s',s)$$

where

$$\gamma^e(s',s) = \exp[\frac{1}{2}L_c y^p_k x^p_k]$$

Note that there are 16 γ's, or transitions, between two columns of nodes.

Also, note that

$u_k = x_k^s$ = the encoder input word

x_k^p = the parity bit generated by the encoder

$y_k = (y_k^s, y_k^p)$ = the noisy version of (x_k^s, x_k^p)

$L_c = 2\frac{E_b}{N_0}$, for rate ½ code

Next, $\tilde{\alpha}$'s are calculated for each node of the trellis from left to right. They are calculated from the following equation

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}$$

Note that the denominator of this equation represents the normalization for each column of nodes, and the $\tilde{\alpha}$'s represent the probability that the encoder is at a certain state at time k. For both MAP decoders #1 and #2 in the turbo-code decoder, the far left column of $\tilde{\alpha}$'s are set to 1 for s = 0 and 0 for s $\neq$ 0. This is because the encoder is initially set to state 000, corresponding to s = 0.

$\tilde{\beta}$'s are then calculated for each node of the trellis from right to left. They are calculated from the following equation

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_{s'} \tilde{\beta}_k(s)\gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}$$

For the first map decoder, the far right column of $\tilde{\beta}$'s are set to 1 for s = 0 and 0 for s $\neq$ 0. Again, this is because of the added tail bits. However, for the second map decoder, the far right column of $\tilde{\beta}$'s are set to the far right column of $\tilde{\alpha}$'s calculated in the previous step.

Finally, the extrinsic likelihood is calculated using the following equation

$$L(k) = \ln\left(\frac{\sum_{s+} \tilde{\alpha}_{k-1} \cdot \gamma_k^e(s',s) \cdot \tilde{\beta}_k(s)}{\sum_{s-} \tilde{\alpha}_{k-1} \cdot \gamma_k^e(s',s) \cdot \tilde{\beta}_k(s)} \right)$$

Where s+ represents the transitions that are associated with a systematic bit of 1, and where s- represents the transitions that are associated with a systematic bit of 0.

This extrinsic likelihood then interleaved or deinterleaved and is added to the quantity $L_c y_k^s$. This is then passed to the next MAP decoder.

After the final half-iteration, $L_1(u_k)$ is calculated using the following equation

$$L_1(u_k) = L_c y_k^s + L_{12}^e + L_{21}^e$$

where L_{12}^e is the extrinsic log likelihood calculated from decoder #1, and L_{21}^e is the deinterleaved extrinsic log likelihood calculated from decoder #2. If $L_1(u_k) > 0$, the turbo-decoder decides that $u_k = 1$. Otherwise, it decides that $u_k = 0$.

Results

The following figure shows the performance of the simulation of the channel model in comparison with the theoretical uncoded limit.

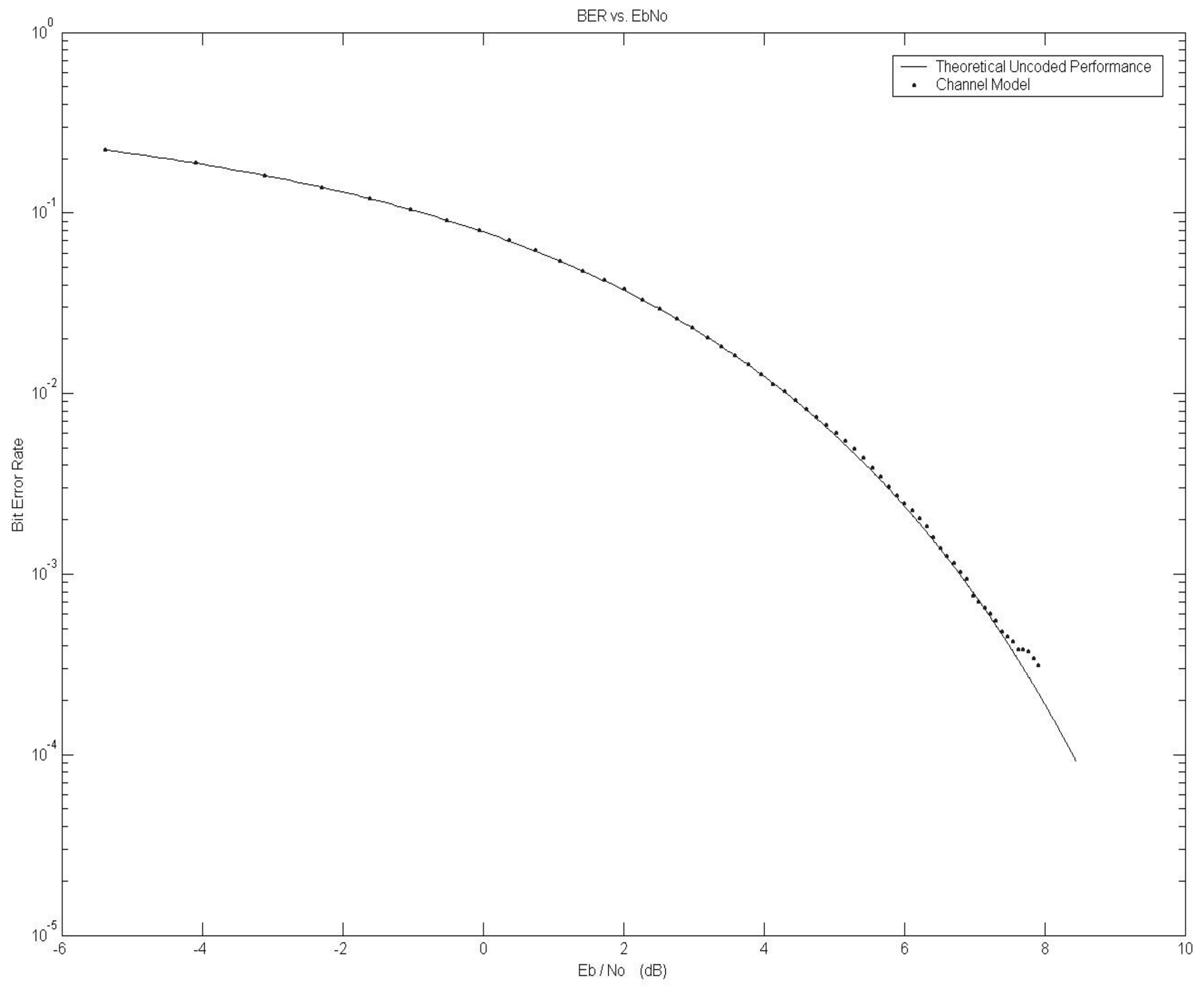

Figure 7.

For the most part, the channel model lies on the theoretical uncoded limit. Figure 7 indicates that the channel model is a good approximation of a channel with zero-mean AWGN.

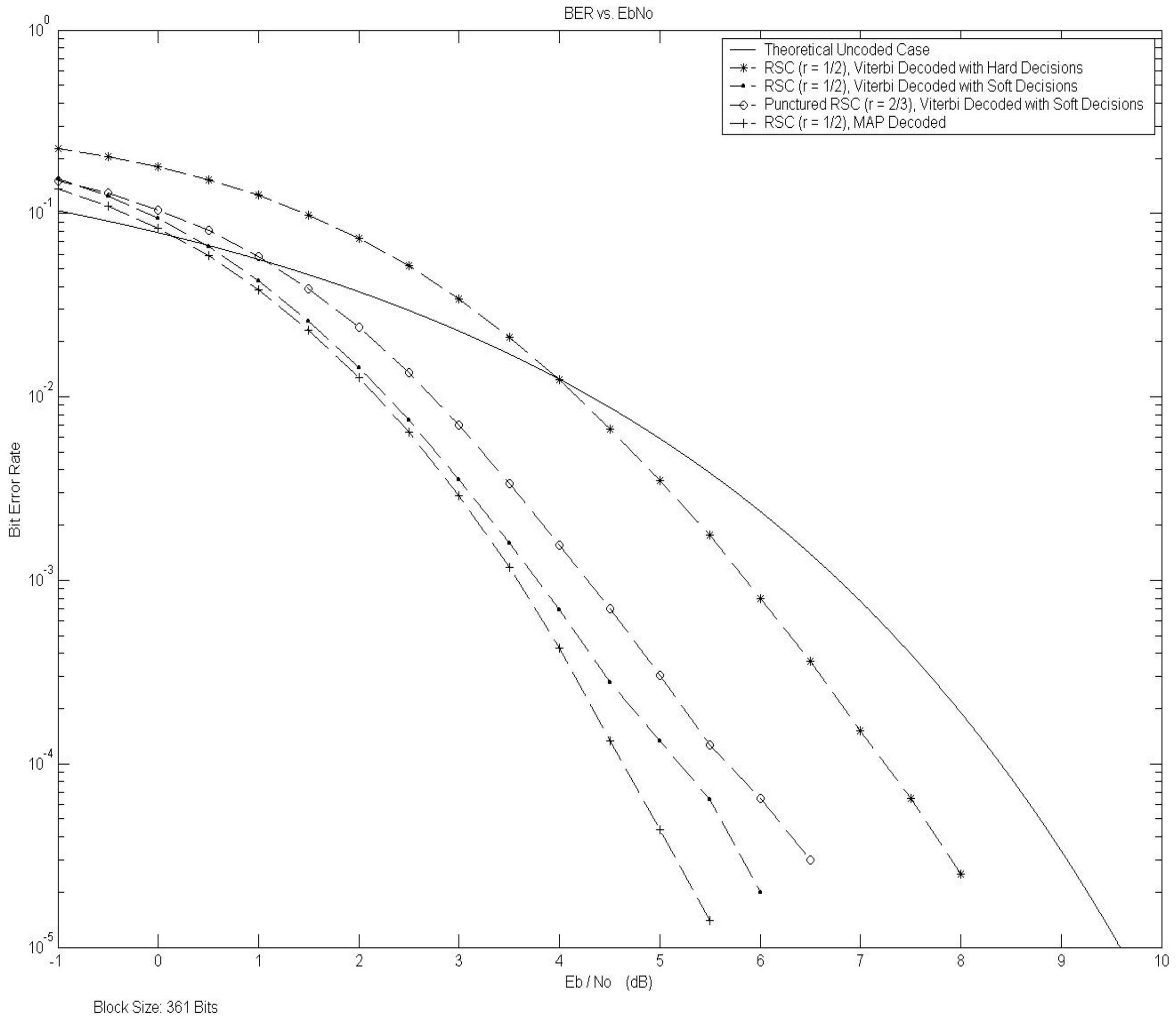

Figure 8.

This figure shows the performance of the RSC codes in comparison with the uncoded case. The block size is 361 bits, and 100,000 blocks were encoded and decoded for each point. For smaller values of E_b/N_0 (dB), the codes perform worse than the uncoded case. However, there is a coding gain for larger values of E_b/N_0 (dB). In particular, for the RSC with hard decisions, there is a coding gain of approximately 2.25 dB for a BER of 10^{-4}. As expected, the RSC with soft decisions performs even better and has a coding gain of approximately 2 dB in comparison to the hard decision case at the same BER. Also, the RSC with puncturing and soft decisions performs better than the hard decision case but slightly worse than the soft decision case without puncturing. Finally, the MAP decoder performs slightly better than the unpunctured soft decision Viterbi algorithm. These curves match those found in literature.

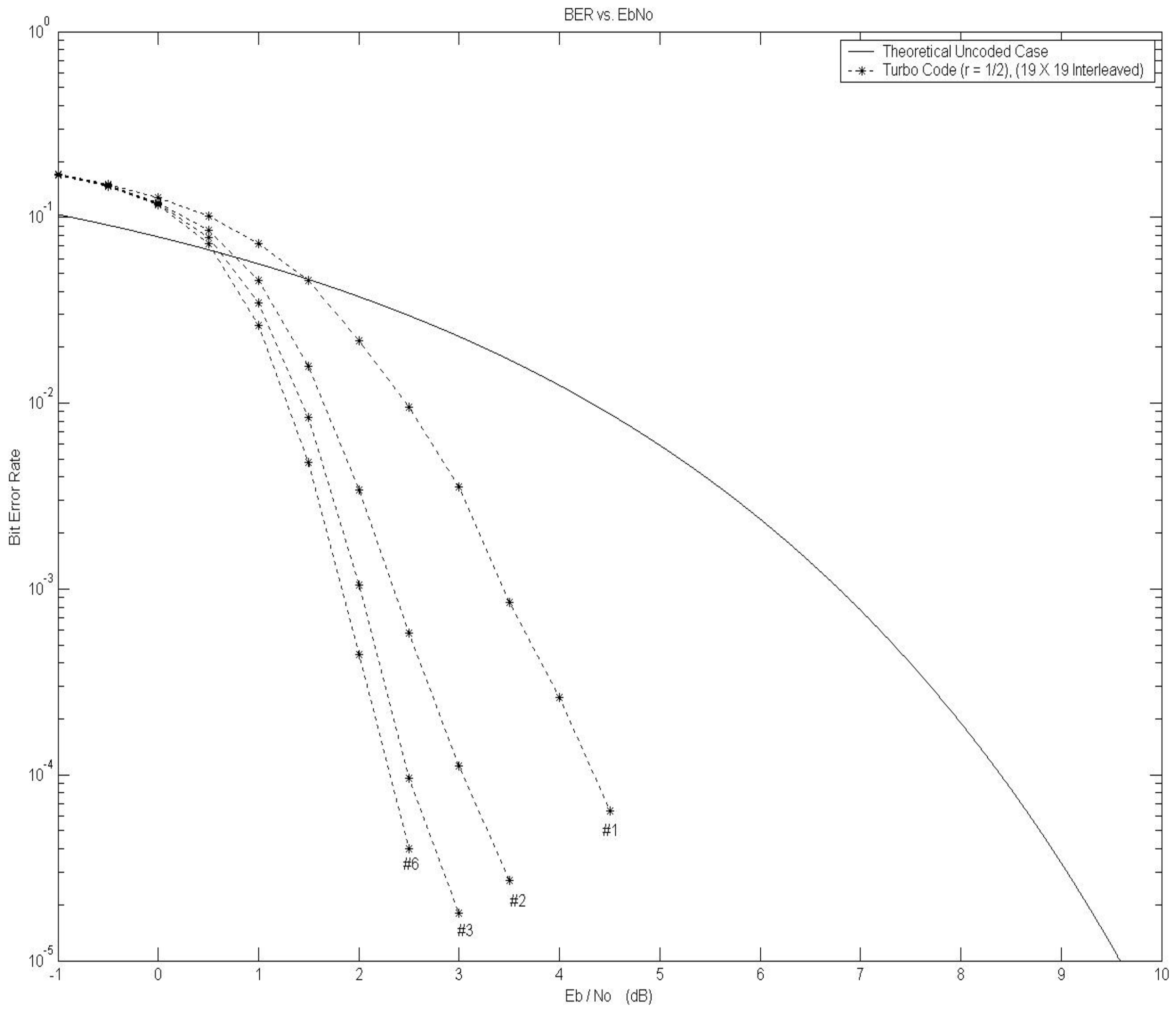

Figure 9.

This figure shows the performance of the turbo code. Iterations #1, #2, #3 and #6 are shown. The block size is 361 bits, and the interleaver is 19 x 19. Although each iteration performs better than the proceeding iteration, the performance increase is less with each iteration.

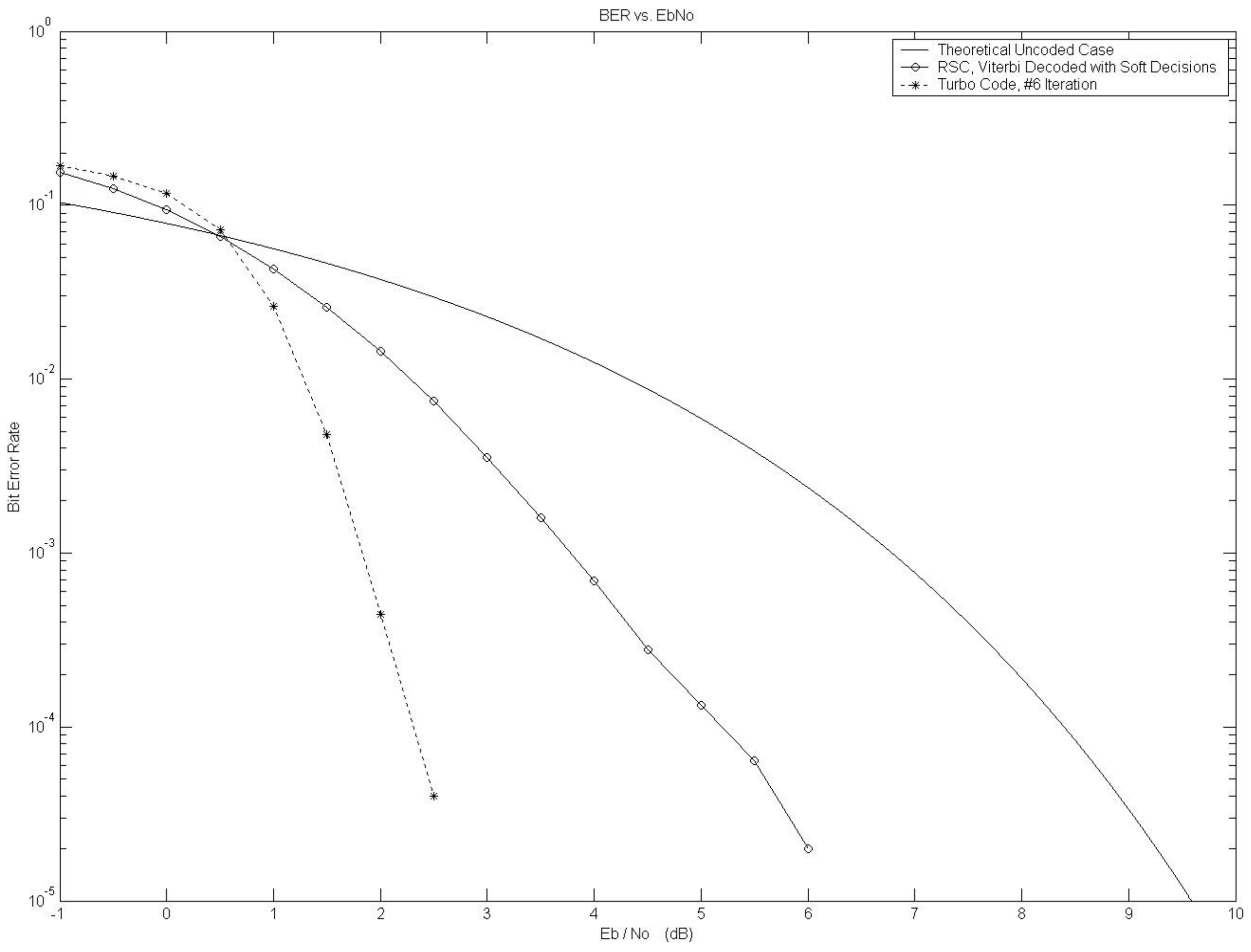

Figure 10.

This figure is a comparison of the turbo code, iteration #6, and the RSC, Viterbi decoded with soft decisions. There is a significant coding gain at low bit error rates.

Conclusions

All of the goals as set forth in the project proposal have been achieved. The verifications have been met, and the requirements have been completed.   The simulations show that the different coding schemes have E_b/N_0 gains for given BER's. The performances of the error correction codes match those found in literature.

Recommendations

Since this project has been designed as a simulation, there are few recommendations for a continuation. However, the hardware that is simulated here can be researched.