Implementation of a turbo codes test bed in the Simulink environment

Ibrahim S. Raad
University of Wollongong, ibrahim@uow.edu.au

Mehmet Yakan
University of Wollongong

IMPLEMENTATION OF A TURBO CODES TEST BED IN THE SIMULINK ENVIRONMENT

Ibrahim S. Raad, Mehmet Yakan

School of Electrical, Computer and Telecommunications Engineering
University of Wollongong,
Wollongong, N.S.W Australia
ibrahim@uow.edu.au

ABSTRACT

This paper presents an implementation of turbo codes test bed developed in Matlab's Simulink which will aid researchers in the field of turbo codes.It discusses the design, implementation and presents the validation results.

Key Words-*Turbo Codes, Simulink, Bit Error Rate*

1. INTRODUCTION

The invention of turbo codes [1] not only assisted in attaining a performance approaching the Shannon theoretical limits of channel coding for transmissions over Gaussian channels, but also revitalised channel coding research.

Turbo coding opened a new chapter in the design of iterative detection-assisted communication systems, such as turbo trellis coding schemes and turbo channel equalisers. Since then, a number of researchers continued enhancing and integrating these codes into the communication system.

This paper describes an implementation of a turbo codes test bed in Simulink, which can assist other researchers in this field achieve their set objective. The paper structure is as follows: Section 2 describes the structure of the turbo encoder and decoder. Section 3 introduces the Simulink package and describes the blocks required for the implementation. Section 4 presents and describes the implemented turbo encoder and decoder within an overall system to validate its operation. Finally, the validation results are presented and discussed in Section 5.

2. ARCHITECTURE OF TURBO CODES

2.1. Turbo encoder block diagram

Turbo encoders are designed usually from two or more convolutional encoders connected in parallel with an interleaver in between the two to ensure that the data received by the second encoder is statistically independent [2]. The same information is passed into both the convolutional encoders from the input in bit format. The encoders scramble the information, getting this data ready to be sent across the physical channel. Puncturing is employed to extract the systematic bits and recursive bits from the information. These will be used by the decoder to ensure the data is error free when it arrives at the end users terminal. Figure 1 depicts a schematic diagram of a turbo encoder with two convolutional encoders separated by an interleaver. Multiplexing and puncturing is used before the data is transmitted across the channel.

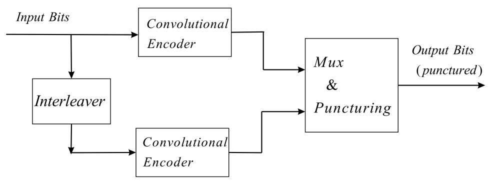

Fig. 1. Turbo encoder schematic [2].

Due to the small number of low weight code words turbo codes can perform well at low signal to noise ratio (SNR). The relatively small minimum distance of the code limits the performance of turbo codes at higher signal to noise ratio. Therefore, it can be said that the goal of turbo codes design is to reduce the multiplicity of low weight code words [3].

2.2. Turbo Decoder Block Diagram

The Schematic for a standard turbo decoder is shown in Figure 2. Two decoders are connected in an iterative manner with interleavers and de-interleavers connecting both, which will allow the first decoder to take advantage of the second decoder's approximation of the probability of the received bits. This process continues until the bit error rate (BER) is zero. At the end of the decoding process a hard decision is carried out on the soft output of the second decoder.

A closer look at the turbo decoder and we find that the systematic channel observation $y^{(s)}= \{y_1^{(1)}, y_2^{(1)}, ..., y_{N_{tc}}^{(1)}\}$ is received by the first decoder. It also receives the obser-

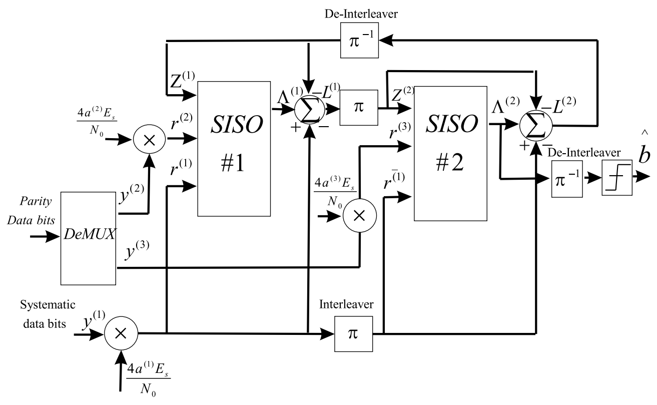

Fig. 2. Turbo Decoder block diagram [4].

vations of the first encoders parity bits $y^{(p)}$=$\{y_1^{(2)}$, $y_2^{(2)}$, ..., $y_{N_{tc}}^{(2)}\}$ and a priori information $Z^{(1)}$ derived from the second decoder's output, where N_{tc} is the interleaver size.

3. THE SIMULINK PACKAGE

3.1. Introduction to Simulink

"Simulink is a software package for modelling, simulating, and analysing dynamic systems. It supports linear and non-linear systems, modelled in continuous time, sampled time, or a hybrid of the two. Systems can also be multi-rate, i.e., have different parts that are sampled or updated at different rates" [3] [5].

With the introduction of communication packages such as Simulink, the programming environment has enabled system designers to take advantage of the package to conduct experiments to validate the functionality and effectiveness of their systems in reduced time [5]. Using Simulink, complete systems can be modelled and directly mapped onto hardware to get accelerated results.

3.2. Simulink testing of signals

Figure 3 shows the manipulation of the original signal as it is passed through puncturing and the convolutional encoder. Simulink uses the $poly2trellis$ function in the convolutional encoder which has the following structure:

$$trellis \quad = \quad ploy2trellis(3, [3; 7], 3) \qquad (1)$$

The first number represents the number of shift registers plus one, which in this case is 3, also known as the constraint length. The next numbers are the generator polynomials, which are represented in octal, 3_8 and 7_8 respectively. This allows the trellis in the turbo codes to be setup. The last number is the feedback polynomial.

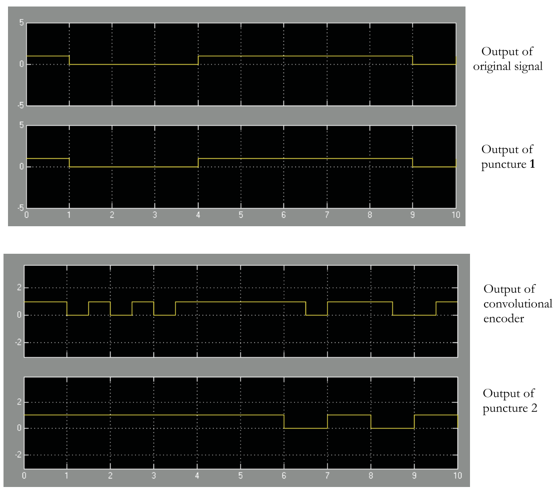

Fig. 3. Comparison of the original signal with signal after puncturing and convolutional encoding.

When a signal is passed through the convolutional encoder, the encoder produces systematic and recursive outputs. This is due to the fact that this system uses recursive systematic convolutional encoders with feedback. Therefore, the output of the encoder has twice the amount of information that is transmitted to the encoder. This can be seen in Figure 3 by comparing the output of the convolutional encoder with the original input signal.

4. DESIGN OF TURBO CODES IN SIMULINK

4.1. Turbo Encoder

The input is generated using the Bernoulli Binary Generator with the frame size and sampling rate setup using the control panel available. This input is entered into the first encoder and into the second encoder after it has gone through an interleaver so to ensure that the data is statistically independent. Two puncture blocks are used after the first encoder. Puncture 1 produces the systematic output of the encoder, which is the same at the original signal, this can be seen in Figure 3. Puncture 2 gives the recursive output of the same encoder. This is a low weight code since it is the output of the first encoder.

A third puncture block is used after the second encoder. Again, a systematic and recursive signal is produced from the second encoder. However only the recursive output is needed. Since only the recursive output (high weight code) is required, a single puncture is placed after the second encoder with puncture rate of $\{0\ 1\}$ to obtain the recursive output of the second convolutional encoder.

The three output signals that are produced become the input of a matrix concatenation and sent into the unipolar to

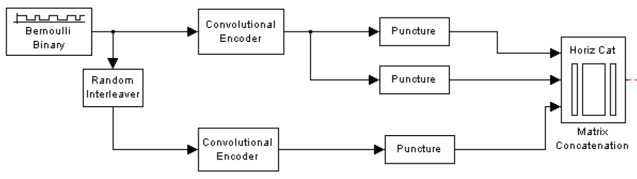

Fig. 4. Turbo Encoder developed in Simulink.

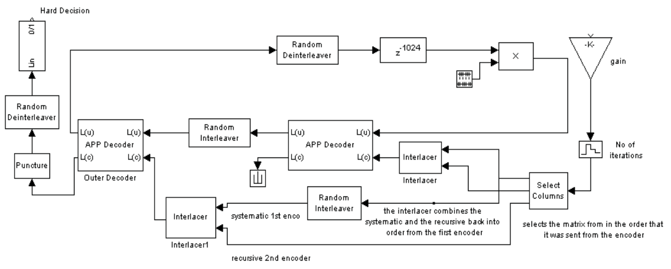

Fig. 5. Turbo Decoder developed in Simulink

bipolar converter. Finally, this signal is transmitted across the AWGN channel. This is shown in Figure 4.

4.2. Turbo decoder

Once the signal is passed through the AWGN channel it is sent through the gain block. This gain is a ratio of the amount of noise that is in the channel and is set by the user before running the simulation. The gain in this design has been set to $2/s$ where s is the variance in the channel. Once through the gain block the signal is then passed through a zero-order hold block. This block adjusts the decoder according to the number of iterations the simulations is going to execute, which is also determined by the user.

The column select is the reverse of the matrix concatenation, which was explained earlier. Three signals are received at the output of the column select. The first signal and second signals are the systematic output and recursive output from the first encoder. The third and final signal is the recursive output from the second encoder.

The first and second signals are sent to an interlacer and then forwarded to the input of the first decoder. These two signals are interlaced together, as one is the systematic output signal and the other is the recursive output signal, respectively. The first signal is passed through a random interleaver before being interlaced with the third signal. This is in order to keep the data statistically independent from each other [6]. This interlaced signal is then passed onto the second decoder. This joining and interlacing can be seen in Figure 5. Also entering the second decoder is the decoded signal from decoder one. An interleaver is placed on the signal that is obtained from decoder 1. With these two signals entering the second decoder, the performance of the second decoder is improved as it has access to more information.

Once the information is received from the second decoder it is then passed through a puncture to remove the redundant data that is produced. The puncture pattern that is used is $\{1\ 0\}$. Once the signal is passed through this puncture the signal is de-interleaved and the result is then passed through the Hard Decision block to convert the final signal to 1's and 0's. Finally the sequence of bits is compared to the original bits using the error rate calculation block. This error rate calculation is called the Bit Error Rate (BER).

As we can see in Figure 5 there is a feedback loop where the signal leaves the second decoder and enters back into the first decoder. This is the iterative part of the decoder and it is done to improve the result of the BER. From the second decoder back to the first decoder a de-interleaver is introduced, and also a delay is inserted which is set to the size of the frame that is being sent through the system. Then the signal is placed back into the first decoders input for further decoding. The input and output of all the decoders are what is known as soft-input / output. This is because the decoders themselves do not make decisions on whether the bit should be a one or a zero. The decoders get as close as possible to achieving the original signal. The final decoding is left to the Hard Decision block.

5. RESULTS

This section presents results obtained from the turbo codes developed in Simulink. They show that this test bed is a useful tool for software implementation and research involving turbo codes.The number of iterations used for the plots listed below is set to five. As shown in Figure 6, as the number of iterations increased the BER decreased. It can also be seen in Figure 7 as the frame size increased the BER decreased. Figure 8 compares the constraint length k. Again the increase of k produced a lower BER. A study of the three decoding algorithms available in Simulink APP decoders shows that the MAX* and the APP decoding algorithm performed better than the MAX algorithm. This is shown in Figure 9.

6. CONCLUSION

This paper presented the implementation of turbo codes test bed in SIMULINK, which can be used by researchers in this field. Covering such topics as the design and validation of

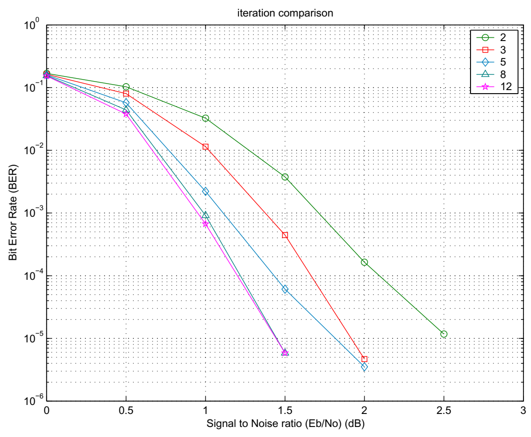

Fig. 6. Plot comparison of varying number of iterations.

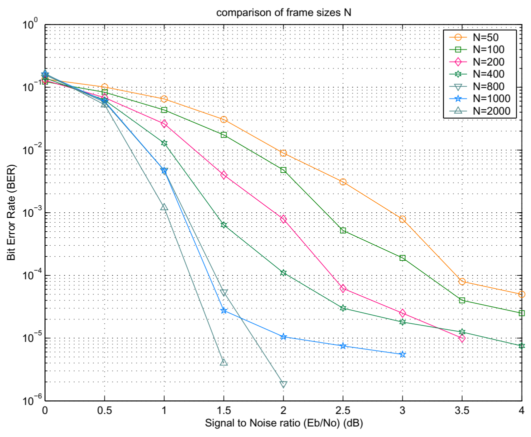

Fig. 7. Plot comparison of varying frame sizes N.

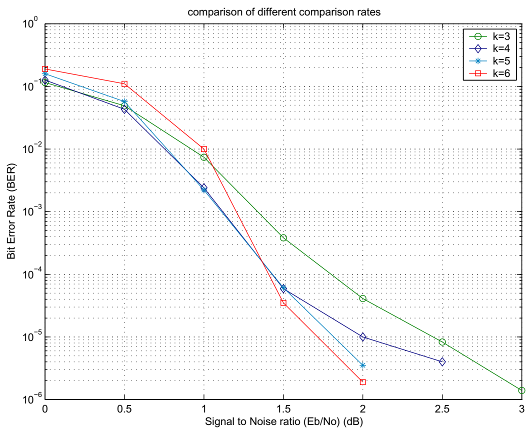

Fig. 8. Plot comparison of constraint lengths k.

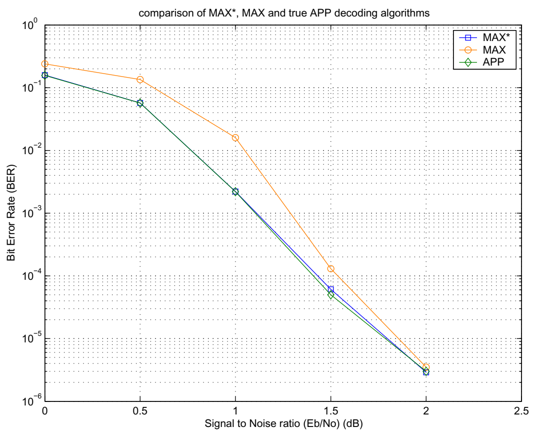

Fig. 9. Plot comparison of three decoding algorithms - MAX*, MAX and True APP.

the test bed for turbo codes, this paper is a useful contribution to researchers looking to begin work on turbo codes. The development is very useful in Simulink as it is modula based. Finally, comparing the validation results with other turbo codes results, this development has been a success.

7. REFERENCES

[1] A. Glavieux C.Berrou and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo codes," in *ICC*. May 1993, pp. 1064–1070, IEEE.

[2] Ibrahim Raad, *Study of Space Time Spreading Across Turbo Codes*, Master of engineering - research, School of Electrical, Computer and Telecommunication Engineering, University of Wollongong, Australia, February 2004.

[3] Mehmet Yakan, *Wireless, Error Correction Codes (Turbo Codes) and Simulink*, Thesis, School of Electrical, Computer and Telecommunications Engieering, Wollongong, N.S.W, October 2004.

[4] Danie Van Wyk Pieter Van Rooyen, Michiel Lotter, *Space-Time Processing for CDMA Mobile Communications*, vol. 1, Kluwer Acemdemic, Boston/Dordrecht/London, 2000.

[5] Mathwork, *The language of Technical Computing V6.5, Simulink Help package. Release 13, June 18*, mathwork, 2002.

[6] E. Villebrun P. Robertson and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," *In IEEE International Conference on Communications*, pp. 1009–1013, 1995.