VHDL IMPLEMENTATION OF TURBO ENCODER AND DECODER
USING LOG-MAP BASED ITERATIVE DECODING

Rajesh Akula, Assoc. Prof.,
Department of ECE,
TKR College of Engineering & Technology, Hyderabad.
akula_ap@yahoo.co.in

Abstract:— In order to have reliable communication, channel coding is often employed. Turbo code as a powerful coding technique has been widely studied and used in communication systems. Turbo coding is an advanced forward error correction algorithm. A standard component in third generation (3G) wireless communication systems. Third generation (3G) mobile communication systems aim to provide a variety of different services including multimedia communication. This requires digital data transmission with low bit error rates. Due to the limitation of the battery life of wireless devices, transmitted power should remain as low as possible, which makes the system more susceptible to noise and interference. Error control coding is thus used to increase the noise immunity of the communication systems. A New coding scheme called “turbo coding”, which stems from convolution coding, has been adopted in the 3G mobile communication system due to its high coding gain and reasonable computation complexity. Turbo coding was initially introduced by Berrou, Glavieux and Thitimajshima in 1993. Ultimate Performance that approaches the Shannon limit requires a new approach using iteratively run soft in/soft out (SISO) decoders called turbo decoders. Two key innovations in turbo coding: Parallel concatenated encoding and iterative decoding.

Parallel concatenated encoders consist of two or more component encoders for convolution codes. Decoding is performed iteratively. The output of the first decoder is permuted and fed to the second decoder to form one cycle of the iteration. Each systematic code is decoded using a soft-in soft-out (SISO) decoder. A SISO decoder can be implemented using the various decoding algorithm Turbo code decoder algorithm is studied in detail in this paper. The performance of Turbo code used in Code Division Multiple Access (CDMA) under Additive White Gaussian Noise (AWGN) channel is evaluated. The bit error rates (BER) of Turbo code at low signal-to-noise ratio (SNR) are obtained by simulations.

Tools used: Xilinx 9.2i, Altera, Synopsys, Matlab, Libero IDE
Language: VHDL

Index Terms—Iterative decoding, MAP algorithm, soft-in/soft-out (SISO) decoder, turbo code, very large scale integration (VLSI), VHDL.

I. INTRODUCTION

Error correcting coding is used to enhance the efficiency and accuracy of information transmitted. In a communication transmission system, data is transferred from a transmitter to a receiver across a physical medium of transmission or channel. The channel is generally affected by noise or fading which introduces errors in the data being transferred. Error-correcting code is a signal processing technique used for correcting errors introduced in the channel.

It is done by encoding the data to be transmitted and introducing redundancy in it such that the decoder can later reconstruct the data transmitted using the redundant information.

The basic goal of communications systems is to transmit information from its source to the destination either in analog or digital form with as few errors as possible. Digital communications has better error correction capability compared to analog communications. Data transmission in discrete messages provides for greater signal processing capability. The ability to process a communications signal means that the errors caused due to noise or other impairments in the course of transmission can be detected and corrected. In addition, digital systems offer faster data processing. They are not only more cost effective but are less subject to distortion and interference which makes them more reliable than the analog systems.

In the modern communication systems, FEC (Forward Error Correction) is a very important module to protect data transmitted without error. Once the data encoded by the FEC encoding module, even the data is interfered by the noise in the transmitting channel, we can still correctly get the original data after we decode the received data by the FEC decoding module. Because of the rapid development of communication systems, the system requirement BER (Bit Error Rate) has became much higher.
Forward Error Correction

Forward error correction (FEC) is a technique for error control during data transmission, whereby redundant information is added to the original data, which allows the receiver to detect and correct errors without the need to resend the data. The main advantage of FEC is to avoid the retransmission at the cost of higher bandwidth requirements on average and therefore is employed in the situations where retransmission is relatively costly or impossible. The original information may or may not appear in the encoded output. A typical communications system utilizing forward error-correction (FEC) is shown in Figure 1.

![Fig 1 FEC in a Communications System](image)

There are different types of FEC such as block coding and convolutional coding. Block codes work on fixed size blocks of bits or symbols or predefined size while convolutional codes work on a stream of bits of any arbitrary length. A block encoder treats each block of data independently and is a memory less device.

A convolutional encoder accepts a stream of message symbols and produces a stream of code symbols as outputs. The main feature of convolutional codes is that encoding of any single bit is strongly influenced by the preceding bits i.e. the memory and this memory is utilized when the convolutional decoder tries to estimate the most likely output.

Maximum A Posteriori (MAP) decoding algorithm for convolution codes which was proposed in 1974 by Bahl et al., but initially received very little attention because of its increased complexity over alternative convolution decoder for a minimal advantage in bit error rate (BER) performance. However, an iterative decoder developed by Berrou et al. in 1993 has enjoyed renewed and greatly increased attention. They considered the iterative decoding of two Recursive Systematic Convolution (RSC) codes concatenated in parallel through a non-uniform interleaver. For decoding the component codes they used a Soft Input/Soft Output (SISO) decoder based on the MAP algorithm. In iterative decoding, several decoding algorithms have been used, including the optimal MAP symbol estimation and its simplification called the max-log-MAP algorithm (Additive MAP Algorithm). A further simplification of log MAP is offered by the modified soft-output Viterbi algorithm (SOVA) which works in a sliding-window SISO decoding algorithm.

INTRODUCTION TO TURBO CODE SYSTEM

The Turbo codes exhibit better performance at much lower power levels. Implementation of Turbo codes consists in designing the Turbo Encoder and Decoder. Turbo coding technique consists essentially of a parallel concatenation of two binary convolution codes, decoded by an iterative decoding algorithm. These codes obtain an excellent bit error rate (BER) performance by making use of three main components. They are constructed using two systematic convolution encoders that are IIR FSSMs, usually known as recursive systematic convolution (RSC) encoders, which are concatenated in parallel. In this parallel concatenation, a random interleaver plays a very important role as the randomizing constituent part of the coding technique. The first encoder operates on an original copy of data, whereas the second encoder operates on an “interleaved” version of data. Interleaving is the method in which bits are rearranged according to a predefined algorithm. The outputs of two RSC encoders and interleaver are appended to get a bit stream of 3 times the original bit stream. It will be transmitted to the channel. This coding scheme is decoded by means of an iterative decoder that makes the resulting BER performance be close to the Shannon limit.

In the original structure of a turbo code, two recursive convolution encoders are arranged in parallel concatenation, so that each input element is encoded twice, but the input to the second encoder passes first through a random interleaver. This interleaving procedure is designed to make the encoder output sequences be statistically independent from each other. The
systematic encoders are binary FSSMs of IIR type, usually have code rate $R=1/2$. In order to improve the rate, another useful technique normally included in a turbo coding scheme is puncturing of the convolutional encoder outputs.

The Decoder is based on a BCJR Algorithm that incorporates soft-input and soft-output values along with the channel reliability values to improve decoding performance, which is called Log MAP Iterative decoding.

The decoding algorithm for the turbo coding scheme involves the corresponding decoders of the two convolutional codes iteratively exchanging soft-decision information, so that the information can be passed from one decoder to the other. The decoders operate in a soft-input—soft-output mode; that is, both the input applied to each decoder, and the resulting output generated by the decoder, should be soft decisions or estimates. Both decoders operate by utilizing what is called $a$ priori information, and together with the channel information provided by the samples of the received sequence, and information about the structure of the code, they produce an estimate of the message bits. They are also able to produce an estimate called the extrinsic information, which is passed to the other decoder, information that in the following iteration will be used as the $a$ priori information of the other decoder. Thus the first decoder generates extrinsic information that is taken by the second decoder as its $a$ priori information. This procedure is repeated in the second decoder, which by using the $a$ priori information, the channel information and the code information generates again an estimation of the message information, and also extrinsic information that is now passed to the first decoder. The first decoder then takes the received information as its $a$ priori information for the new iteration, and operates in the same way as described above, and so on.

The iterative passing of information between the first and the second decoders continues until a given number of iterations is reached. With each iteration the estimates of the message bits improve, and they usually converge to a correct estimate of the message. The number of errors corrected increases as the number of iterations increases. However, the improvement of the estimates does not increase linearly, and in practice, it is enough to utilize a reasonable small number of iterations to achieve acceptable performance.

One of the most suitable decoding algorithms that performs soft-input—soft-output decisions is a maximum $a$ posteriori (MAP) algorithm known as the BCJR (Bahl, Cocke, Jelinek, Raviv, 1974) algorithm. Further optimizations of this algorithm lead to lower complexity algorithms, like SOVA (soft-output Viterbi algorithm), and the LOG MAP algorithm, which is basically the BCJR algorithm with logarithmic computation.

This paper evaluates the BER performance of Turbo codes used CDMA under Additive White Gaussian Noise (AWGN) channels. In addition, this presents more accurate approximation in the modified Turbo code decoding algorithm and corrects the error in the decoding algorithm under AWGN channel. Deep space, satellite, Mobile and Cellular communication belong to such a class, where receivers operate at a very low SNR due to limitations in antenna size, transmission power and long transmission range. The Turbo codes are implemented using MATLAB & Active HDL. MATLAB code for Turbo Coder & Decoder is written, simulated & verified. Performance is analyzed introducing errors in the decoder input and also with puncturing. Active HDL code for Turbo Coder & Decoder is written.

**Encoding of Turbo Code**

As conventional convolutional code, the encoder for a Turbo code accepts $k$-bit blocks of the information sequence $u$ and produces an encoded sequence (code word) $v$ of $n$-symbol blocks. Moreover, each encoded block depends not only on the corresponding $k$-bit message block at the same time unit, but also on $m$ previous message blocks.

Hence, the encoder has a memory order of $m$. The set of encoded sequences produced by a $k$-input, $n$-output encoder of memory order $m$ is called an $(n,k,m)$ Turbo code. The ratio $R=k/n$ is called the code rate. Since the encoder contains memory, it must be implemented with a sequential logic circuit.

![Fig. 2 Encoder of turbo code](image)

**RSC Encoder**

A rate-$1/2$ recursive systematic encoder outputs the input $u$ unchanged in parallel with an encoded bit, $p$. The encoded bit $p$ is defined by two generator polynomials: $g_1(D)$ and $g_2(D)$. The generator polynomials define the encoder and are represented in octal format.

For example a $(5,7)$ encoder has generator polynomials of $g_1(D)=1+D^2$ and $g_2(D)=1+D+D^2$, where $1+D^2$ and
1+D+D2 are represented as 101 and 111 in octal, respectively.

**Interleaver**

The purpose of the interleaver is to reorder a group of K input bits before they are encoded by the second RSC encoder. Typically, a turbo encoder interleaver is implemented by some type of pseudo-random algorithm. The interleaving algorithm that is used in the encoder can have significant impact on the performance of the decoder [6]. For the purposes of this study, only one interleaver type was used, a random K-bit interleaver.

**Encoder for a (5,7) RSC Code**

The (5,7) encoder is a memory-two encoder. The two memory elements can take on four possible states. A hardware realization for a (5,7) encoder is shown in Figure 3.

![Encoder Block Diagram for a (5,7) RSC Code](image)

The value of the two memory elements in the encoder, R0 and R1, define the “state” of the encoder. A state diagram is created from the possible state changes of R0 and R1 (as all possible input sequences are generated) and is used to create a trellis diagram utilized in the decoder operation. The state diagram for the (5,7) encoder is shown in Figure 4.

![State diagram for the (5,7) encoder](image)

**Decoding of Turbo Code**

A turbo decoder is an iterative decoder, in which multiple decoders share probability information with each other in an iterative fashion. The turbo decoder receives as its input a soft decision value from the demodulator. This soft decision value will represent the probability that the transmitted bit was a 1 or 0.

**PCCC Iterative Decoder**

A PCCC decoder is composed of two identical decoders, two identical interleavers, one deinterleaver, and a hard-decision output logic block, as shown in Fig 5.

Maximum a posteriori (MAP) soft-in, soft-out (SISO) decoder is used for the two decoders. The interleaver used matches the interleaver used in the encoder, and the deinterleaver performs the opposite operation of the interleaver. The three outputs from the encoder: u, p, and q, are received into the decoder as u’, p’, and q’. The received bits: u’, p’, and q’ correspond to u, p, and q after having been affected by the transmission channel.
The decoder operates on a 3K-bit block of received sequences u’, p’, and q’. Each 3K-bit block of received data passes through a number of iterations in the two decoder system. The number of iterations depends on the performance requirements and system-level considerations. Once a sufficient number of decoder iterations have been performed, the extrinsic information from both decoders and the log-likelihood ratio of the received systematic symbols are used to compute the decoded message.

![PCCC Iterative Decoder Block Diagram](image)

Fig 5  PCCC Iterative Decoder Block Diagram

Part of the power of turbo codes comes from the fact that the two encoded data sequences are decoded separately, but the probability information from each decoder (extrinsic information) is shared. Since both decoders operate on the same set of information bits, each decoder has the capability of helping the other. Therefore, each iteration through the decoder provides an increase (to a point) in the probability of producing the correct codeword at the decoder output.

**Deinterleaver/Interleaver**

The interleaver used in the decoder must match the interleaver used in the encoder. The interleaver will be different in terms of hardware implementation, based on the fact that soft values (or multi-bit values) are being passed between the decoders. Therefore, the size and complexity of the interleaver implementation is related to the number of bits (or quantization) selected as the output of each decoder. The deinterleaver performs the opposite function of the interleaver; where the interleaver permutes the data, the deinterleaver de-permutates the data back to the original order.

Two component decoders are linked by interleavers in a structure similar to that of the encoder. As seen in the figure, each decoder takes three inputs:
1) The systematically encoded channel output bits;
2) The parity bits transmitted from the associated component encoder;
3) The information from the other component decoder about the likely values of the bits concerned.

This information from the other decoder is referred to as *a-priori* information. The component decoders have to exploit both the inputs from the channel and this *a-priori* information. They must also provide what are known as soft outputs for the decoded bits. This means that as well as providing the decoded output bit sequence, the component decoders must also give the associated probabilities for each bit that it has been correctly decoded.

The soft outputs from the component decoders are typically represented in terms of the so-called *Log Likelihood Ratios (LLRs)*, the magnitude of which gives the sign of the bit, and the amplitude the probability of a correct decision. The LLRs are simply, as their name implies, the logarithm of the ratio of two probabilities.
Notice that the two possible values of the bit $u$ are taken to be +1 and -1, rather than 1 and 0, as this simplifies the derivations that follow.

The decoder operates iteratively, and in the first iteration the first component decoder takes channel output values only, and produces a soft output as its estimate of the data bits. The soft output from the first encoder is then used as additional information for the second decoder, which uses this information along with the channel outputs to calculate its estimate of the data bits. Now the second iteration can begin, and the first decoder decodes the channel outputs again, but now with additional information about the value of the input bits provided by the output of the second decoder in the first iteration. This additional information allows the first decoder to obtain a more accurate set of soft outputs, which are then used by the second decoder as a-priori information. This cycle is repeated, and with every iteration the BER of the decoded bits tends to fall.

However, the improvement in performance obtained with increasing numbers of iterations decreases as the number of iterations increases. If we decode with more iterations, we can achieve lower Bit Error Rate (BER) when the channel Signal to Noise Ratio (SNR) is fixed. Hence, for complexity reasons, usually only about eight iterations are used.

**Log Likelihood Ratios**

Information is passed from one component decoder to another in the form of Log Likelihood Ratios (LLR) [5]. As its name suggests, the LLR of a certain bit is the log of the ratio of the probability that a certain bit takes the value 1 to the probability that this same bit takes the value -1. The LLR of a bit $u_k$ is denoted as $L(u_k)$ and is defined as

$$L(u_k) = \ln \left( \frac{P(u_k = +1)}{P(u_k = -1)} \right)$$

Figure 7 shows how the LLR values vary with $P(u_k = +1)$. A positive LLR indicates that the bit $u_k$ is more likely to be +1, and a negative LLR indicates that it is more likely to be -1. The magnitude of the LLR reflects how sure we are about the decision of the decoded bit.

$$P(u_k = \pm 1) = \frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)/2}}. e^{\pm L(u_k)/2}$$

![Figure 7: $L(u_k)$ versus $P(u_k=+1)$.

The larger the magnitude, the more it is likely that the bit is decoded correctly. A LLR value of zero indicates that it is equally likely that the decoded bit is +1 or -1. The probabilities $P(u_k=+1)$ or $P(u_k=-1)$ can be recovered from the LLR. In fact, knowing that $P(u_k = -1) = 1 - P(u_k = +1)$, and taking the exponent of both sides of above equation , we get:
Notice that above equation is the product of a constant term (the bracketed term) with another term that depends on whether the bit is +1 or -1 [5].

In channel coding, we are interested in \( P(u_k=+1) \) given a certain vector \( y \) of received values \( y_k \)'s. This is translated into a conditional LLR of a bit \( u_k \) conditioned on a received sequence by below equation also known as the a-posteriori probability of the decoded bit \( u_k \).

\[
L(u_k|y) = \frac{P(u_k = +1|y)}{P(u_k = -1|y)}
\]

On the other hand, in the case of BPSK modulation over an AWGN fading channel with fading coefficient \( a \) and noise variance \( \sigma^2 \), we define:

\[
P(y_k|x_k = \pm 1) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{E_b}{2\sigma^2}(y_k \pm a)^2}
\]

Similarly to the first equation, we also define:

\[
L(y_k|x_k) = l_{n1} \frac{P(y_k|x_k = +1)}{P(y_k|x_k = -1)}
= l_{n1} \left( e^{-\frac{E_b}{2\sigma^2}(y_k - a)^2} \right)
= L_{c} y_k
\]

\[
L_{c} = A \frac{E_b}{2\sigma^2},
\]

The MAP Algorithm

A brief overview of the Maximum A-Posteriori (MAP) algorithm introduced in 1974 by Bahl, Cocke, Jelinek and Raviv is discussed. The MAP algorithm is optimal for Convolutional codes and it minimizes the number of bits decoded incorrectly, while the Viterbi algorithm minimizes the probability of an incorrect path in the trellis. The MAP algorithm provides the decoded bit sequence along with the probability that each bit is decoded correctly. In this section, we shall briefly introduce MAP decoding, and we will omit the derivations which are presented in [5].

First we need to define the following:

- \( \alpha_{k-1}(s) \) is the probability that the trellis is in state \( s \) at time \( k - 1 \) given the sequence of bits previous to bit \( k \);
- \( \beta_k(s) \) is the probability that the trellis is in state \( s \) at time \( k \) given the received channel value \( y_k \) at time \( k \).

The values \( \gamma_k(s, s) \) are probability that the trellis moves from \( s' \) to \( s \) given the received channel value \( y_k \) at time \( k \).

The values \( \gamma_k(s, s) \) are derived from the channel values \( (L_c \text{ and } y_k) \) and the A-Priori information \( L(uk) \), which are reliability values passed from previous decoders, and are obtained as follows [5]:

\[
\gamma_k(s, s) = e^{ukL(u_k)/2} e^{\frac{E_b}{2\sigma^2} \sum_{l=1}^{n} y_{kl} x_{kl}}
\]

where \( x_k \) is the codeword associated with the transition from state \( s \) to states \( 1 \) is the bit, and \( n \) is the number of bits per codeword. The values \( \alpha_{k-1}(s') \) are calculated recursively and in a forward manner as described in the following:
with the following initial conditions:

\[ \alpha_0(S_0 = 0) = 1 \]
\[ \alpha_0(S_0 = 1) = 0 \text{ for all } s \neq 0 \]

The \( \beta_k(s) \) are also calculated recursively but in a backward manner following the Equation

\[
\beta_{k-1}(s) = \sum_{s'} \gamma_k(s', s) \beta_k(s')
\]

with the initial (or terminating) conditions:

\[ \beta_N(0) = 1 \]
\[ \beta_N(s) = 0 \text{ for all } s \neq 0 \]

Thus, after performing the above calculations, the MAP decoder outputs the following LLR [5], \( L(u_k|y) \):

\[
L(u_k|y) = \ln \left( \frac{\sum_{s' : u_k = 1} \alpha_{k-1}(s', s) \gamma_k(s', s) \beta_k(s')} {\sum_{s' : u_k = -1} \alpha_{k-1}(s', s) \gamma_k(s', s) \beta_k(s')} \right)
\]

Max-Log-MAP approximates the extremely complex summation of Equation for \( L(u_k|y) \) by a maximum operation. Consequently, it is a sub-optimal decoding algorithm.

---

**Fig 8 Alpha, Beta, Gamma Calculations**

**RESULTS & CONCLUSIONS**

In this paper, I first investigated the performances and complexities of the various SISO algorithms. The SISO algorithm providing the best compromise between the performance and complexity is selected in our turbo decoder implementation. Next, the critical issues regarding turbo decoder implementations are discussed to keep the decoder complexity as low as possible while avoiding significant performance degradation. Based on these investigations, a turbo decoder with the selected SISO algorithm is designed and implemented using VHDL as design entry and simulation language. Finally, the VHDL design is verified by comparing the VHDL simulation results with performance obtained from Matlab simulations.

The simulation results show that Turbo code is a powerful error correcting coding technique under SNR environments.
It has achieved near Shannon capacity. However, there are many factors need to be considered in the Turbo code design. First, a trade-off between the BER and the number of iterations need to be made, e.g., more iterations will get lower BER, but the decoding delay is also longer. Secondly, the effect of the frame size on the BER also needs to be considered. Although the Turbo code with larger frame size has better performance, the output delay is also longer. Thirdly, the code rate is another factor that needs to be considered. The higher coding rate needs more bandwidth.

From the simulation results, it is observed that the behavior of the Turbo decoder is quite different under different channel environments. Another drawback of the Turbo code is its complexity and also the decoding time.

(1). Effects of the number of iterations on BER.

Increasing the number of iterations is not much help in the low regions of SNR. In the middle to high regions of SNR, when the number of iterations increases from 1 to 3, the performance of the Turbo decoder improves dramatically. In other words, BER decreases dramatically. This is due to the decoder 1 and decoder 2 share the information and make more accurate decisions. As the number of iterations increases, the performance of the Turbo decoder improves. However, after the number of iterations reaches a certain value, the improvement is not significant. Fig 10 shows the convergence of the decoding iterations.

The AWGN channel corresponds to a SNR_b of 1.2 dB and code rate of 1/2.

Fig 9 Effects of frame size on BER

The conclusion is that 3 iterations are good enough to get reasonable results in the middle region of SNR_b, and 2 iterations, for high region of SNR_b. High region is defined as SNR_b>1.4 dB, middle range 1.4 dB >SNR_b>0.3 dB in AWGN channel. In the other words, further iterations do not gain much improvement of the BER.

(2). Larger frame size gets better performance

The larger the frame size therefore, it will produce larger distance by using an interleaver. The correlation between the two adjacent bits will become smaller. Hence the decoder gives better performance. The simulation results verified this conclusion. However, since Turbo code is a block code, it causes a time delay before getting the complete decoding output. Increasing the frame size also increases the delay time. Fig 9 shows the BERs of Turbo code under AWGN channel with the code rate=1/2, iteration=3, frame size L =384 bits. From the figure we can see that the Turbo code with larger frame size has better performance.
Fig 10. Iteration convergence and BER

A. Performance Comparison

Performances of the turbo decoders using different SISO algorithms were evaluated by computer simulations fig 12 and 13. The turbo encoder in the simulations uses two identical rate-1/2 RSCCs with the generator polynomial of [5, 7]. It is observed that turbo codes are capable of achieving very low bit error rates (BER) in regions of low SNR values.

In this paper, the same S-random interleaver is used in all the simulations. This guarantees that the performance difference comes solely from the SISO decoding algorithms. It is observed that turbo decoding with MAP algorithm provides the best performance out of all the simulated decoding algorithms. Performance of Log-MAP algorithm at BER of $10^{-4}$ is very close from that obtained using the MAP algorithm and is approximately 0.6 dB better than that obtained using SOVA.

Fig 12. Smart Power (power usage)

B. Complexity Comparison

In this section, we perform a decoding complexity analysis and comparison over the different SISO algorithms.

C. Implementation Verification and Discussions

The turbo decoder was designed using VHDL as the design description and simulation tool using Xilinx 9.2i simulator. A register-transfer-level (RTL) turbo decoder model was generated. VHDL simulation was performed to verify the designed turbo decoder.

This guarantees that Log-MAP decoder implemented in Matlab-simulation has the same dynamic range limitation as that of the VHDL implementation. Therefore, any performance difference is only due to the limited precision of the VHDL implementation. Results of both simulations show a very good match and therefore verify the turbo decoder implementation.
References:
[8] UWB paper