International Journal of Electronics and Communication Engineering.
ISSN 0974-2166 Volume 15, Number 1 (2022), pp. 1-15

http://www.irphouse.com

Design and Implementation of a Turbo Codes Using a DSP Processor

Budani Jeoffrey, Ibo Ngebani, Sajid M. Sheikh and Ishmael Zibani

Electrical Engineering Department,
University of Botswana, Gaborone, Botswana

Abstract

Turbo codes are classified as forward error correction codes, and their performance is known to be close to Shannon's limits. Error correction coding is an important part of wireless communication systems and this paper focuses on the design and implementation of a turbo encoder and decoder using a DSP processor. Most implementations of Forward Error Correction codes turbo are based on FPGA and ASIC implementations. This involves a large number of logic gates, registers and routing resources. The manufacturing cycle is also much costly, length and involves a lot of manpower. Advances in VLSI technology accompanied with the ever increasing speed of processors makes it highly feasible to implement these complex FEC codes using DSP processors. To validate our designs, randomly generated bits are modulated using BPSK, then send over an Additive White Gaussian Noise (AWGN) channel. The received noisy signal is then demodulated and the original sent bits restored via an iterative decoding algorithm which runs in the turbo decoder. Our design and implementation is based on the Texas Instruments TMS 320C6713 DSP processor.

Keywords: DSP processor, Code Composer Studio, Turbo codes.

Introduction

The evolution of wireless communication generations has imposed stringent requirements on communication systems. Upcoming technologies shall require higher user capacity, higher data rates, high reliability, availability, bandwidth efficiency and security. As a result, better channel coding schemes are needed as a means of satisfying the above requirements. Studies has shown that coding schemes need to meet Shannon's capacity theorem to meet current communication demands are often very complex as

they rely upon iterative decoding algorithms. Shannon capacity theorem states that there is an encoding method that allows information to be transmitted over a noisy channel with an arbitrarily low error probability for any communication system with a channel capacity C with a data rate R of at least less than C[2]. However, this theory means that more redundant bits need to be added to achieve a low error probability. This causes delays and high computational requirements. However, the theorem does not describe how to build such a capacitive approximation code. Until the discovery of turbo codes, much research was done on channel coding that could satisfy Shannon's capacitive theorem[3] .Turbo codes were discovered by Berrou and Glavieux in 1993[4]. Turbo codes design are based on concatenated coding, randomness, and iterative decoding[1]. The turbo encoder consists of two recursive systematic convolutional (RSC) encoders linked together by an interleaver[1]. The input bits that enter the first encoder are re-ordered with the aid of an interleaver before getting into the second encoder. The codewords of the turbo encoder are comprised of input bits followed by the parity test bits of each encoder. The code rate of the turbo codes can be varied by the process of puncturing. The turbo codes are decoded via an iterative deciphering algorithm[1]. Current implementations of Turbo codes are reliant on FPGA and ASIC technologies. FPGAs have disadvntages of being slower, complex and are often more expensive than custom silicon. Moreover mistakes not detected at design time have a large impact on the development time and cost. The same also applies to ASIC design which are faster than FPGAs. However some large ASICs can take a year or more time to design. ASIC design tools are very much expensive and costly. In ASIC the designer has to take care of DFM issues, signal integrity issues and many more huddles. All of these problems can be avoided if a DSP processor is used. Advances in VLSI technology have made the speed of processors to be much higher to be able to handle complex algorithms with virtually no latency. In this paper, a turbo encoder and decoder are implemented by using the Texas instruments TMS 320C6713 processor.

RELATED WORKS

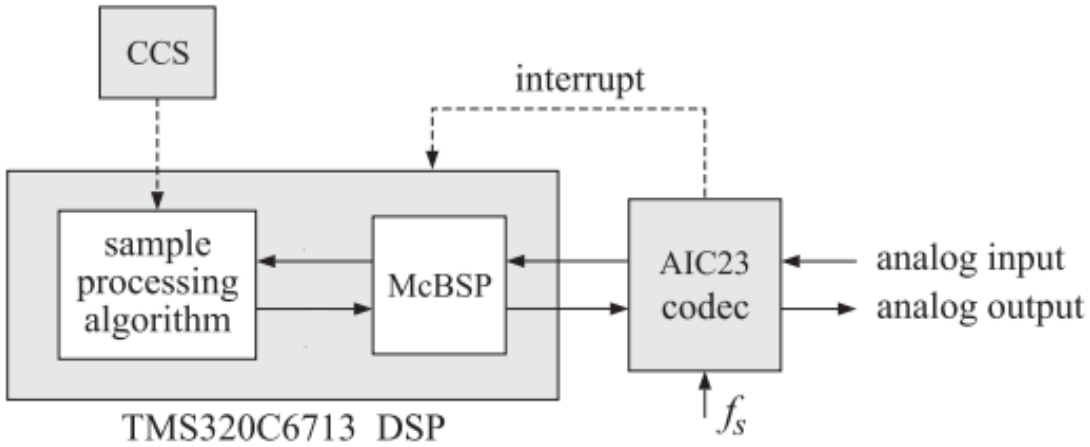
Related studies have also been conducted on the implementation of turbo codes using other technologies. Fleah M.A and Al-Doori Q.F[5] presented the Design and Implementation of Turbo encoder/ decoder using an FPGA. In this task, focus was on designing turbo code using a MATLAB program via two programming methods, Simulink and M.file. Simulink design is used to build turbo code and actually implement it in FPGA devices using VHDL code. The M.files are used to measure turbo code performance in terms of bit error rate (BER). BER is tested by changing turbo code parameters such as code length, number of iterations, different rates, and different decoding algorithms. Abdel-Azim E.M.A[6] presented the Design And Implementation For A Multi-Standard Turbo Decoder Using A Reconfigurable ASIP. This task demonstrates an efficient architecture for implementing turbo decoders with a scalable, low power, application-specific instruction set processor (ASIP). Parallel processing of the ASIP architecture has been proposed to meet the high throughput requirements of turbo decoders, one of the most important requirements of 4th generation (4G) wireless communication systems.

TMS320C6713 DSK PROCESSOR

The hardware implementation is performed with the Texas Instruments TMS320C6713 DSP Starter Kit (DSK), which is based on the TMS320C6713 floating point DSP running at 225MHz[7]. Basic clock cycle command time is 1 / (225MHz) = 4.44 nanoseconds. Up to 8 instructions can be executed in parallel during each clock cycle, achieving up to 8 x 225 = 1800 million instructions per second (MIPS)[7]. The C6713 processor composed of 256KB of internal memory and is capable of addressing 4GB memory externally. The DSK board consists of 16MB SDRAM memory and a 512KB Flash ROM[7]. It features an on-board 16-bit stereo audio codec (Texas Instruments AIC23) that acts as both an ADC and DAC converter[7]. The board also consists of input and output audio jacks namely microphone, stereo line, speaker and head-phone respectively. The AIC23 codec is designed such that its sampling frequency are as follows: fs = 8, 16, 24, 32, 44.1, 48, 96 kHz[7].The Analog-to-Digital Converter (ADC) of the codec is employed as a multi-bit, third-order, noise-shaping delta-sigma converter, allowing for the various oversampling ratios that can be used to achieve the sampling frequency selection described above[7]. The Digital-to-Analog Converter part is also designed as a multi-bit secondary noise shaping delta-sigma converter. The converter's oversampling interpolation filter acts as a near-ideal reconstruction filter with the Nyquist interval as the passband[7].

The DSK also features four user programmable DIP switches and each associated with LED which regulates and monitor the programs running on the DSP[8]. All DSK functionality is managed by Code Composer Studio (CCS), a fully integrated development environment (IDE) consists of an optimized C / C ++ compiler, assembler, linker, debugger, and program loader[8]. The CCS aids the programming aspects of the DSP by read signals stored in the DSP's memory and display them in time or frequency domains[8]. The CCS is linked to DSK through a USB connection using a computer.

The following block diagram depicts the overalloperations involved in all of the hardware experiments. Processing is interrupt-driven at the sampling rate fs[7], [8].



The AIC23 codec is configured (through CCS) to operate at one of the above sampling rates fs. Each collected sample is converted to a 16-bit two's complement integer (a short data type in C)[8]. The codec actually samples the audio input in stereo, that is, it collects two samples for the left and right channels[8]. At each sampling instant, the

codec combines the two 16-bit left/right samples into a single 32-bit unsigned integer word (an unsigned int, or Uint32 data type in C), and ships it over to a 32-bit receive register of the multichannel buffered serial port (McBSP) of the C6713 processor, and then issues an interrupt to the processor[8]. Upon receiving the interrupt, the processor executes an interrupt service routine (ISR) that implements a desired sample processing algorithm programmed with the CCS (e.g. filtering, audio effects, etc.)[7]. During the ISR, the following actions take place: the 32-bit input sample is read from the McBSP, and sent into the sample processing algorithm that computes the corresponding 32-bit output word, which is then written back into a 32-bit transmit-register of the McBSP, from where it is transferred to the codec and reconstructed into analog format, and finally the ISR returns from interrupt, and the processor begins waiting for the next interrupt, which will come at the next sampling instant[8]. All processing operations during the execution of the ISR must be completed in the time interval between samples, that is, T = 1/fs[7].

TURBO CODE ENCODER

The basic turbo encoder is composed of two Recursive Systematic Convolutional (RSC) encoders as component codes linked together by an interleaver[1]. The input bits fed to the first convolutional encoder is scrambled by the interleaver before being fed to the second convolutional encoder. The interleaver play a significant role in introducing the randomness hence creating codewords from each component encoder statistically independent[1]. The turbo code is generated by the input bits followed by the parity bits from both encoders. Alternatively, it can be generated by the puncturing process. In this process, the input bit is followed by one of the encoder's parity bits on an odd or even time base. This process also change the code rate of the turbo encoder. Figure 1 below depicted a basic turbo code encoder which is consists of two identical convolutional encoders parallel concatenated. The possible codewords as per figure 1, without puncturing and with puncturing respectively are as follows:

i) C1,C2,C3,C1,C2,C3,C1,C2,C3…C1,C2,C3

ii) C1,C2,C1,C3,C1,C2,C1,C3,…,C1,C2,C1,C3

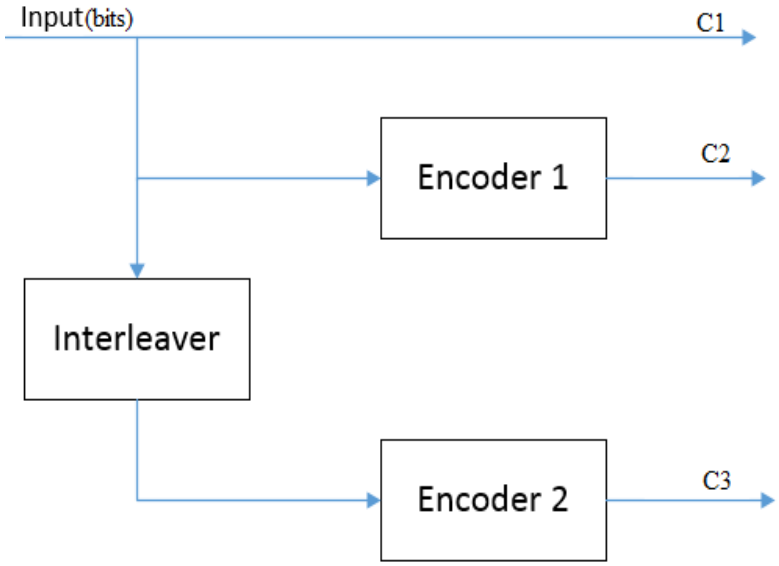


Figure 1: basic block diagram of turbo code encoder

Furthermore figure 1 can be explored as shown in figure 2 below.

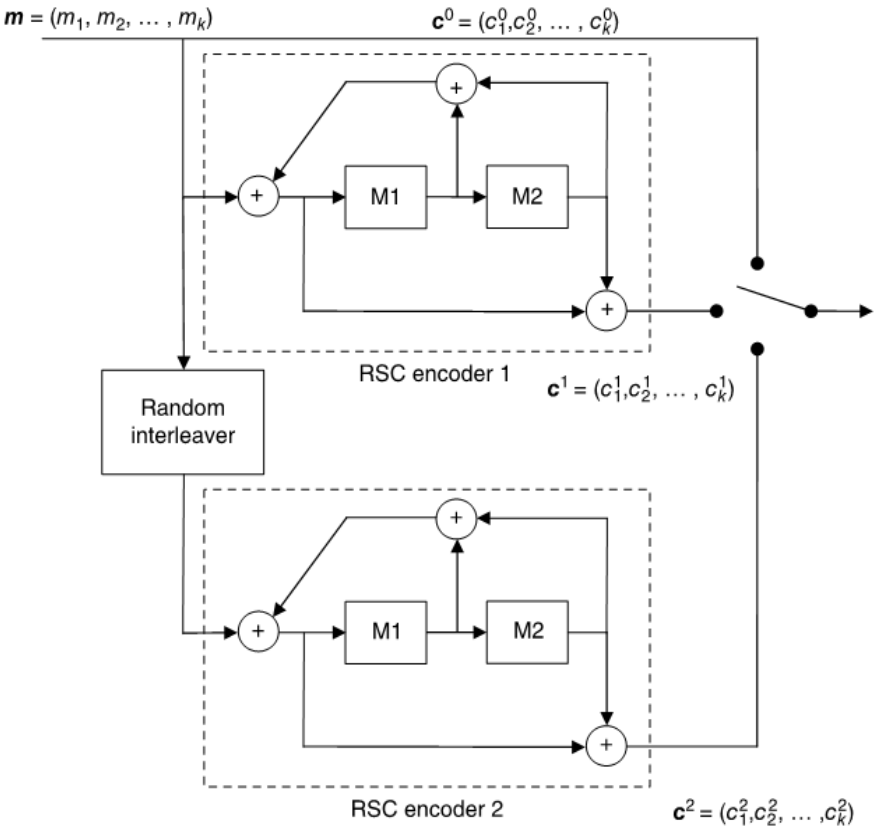


Figure 2: Explored basic block diagram of turbo code encoder[1].

INTERLEAVER

Interleaver are devices that reorder bit sequences in a one to one pseudo random format[9]. They are usually used for improving error correction capabilities of coding schemes over busty channels[9]. They achieve this by to increase the minimum distance in the code distance spectrum therefore the design of turbo codes need a diligent design of its constituent interleaver[9]. The opposite process to interleaving is known as de-interleaving, dedicated to reinstate the received sequence into its original order. For this paper we going to consider a random interleaver of the turbo encoder. In this algorithm, the data is randomized as depicted in figure 3 below.

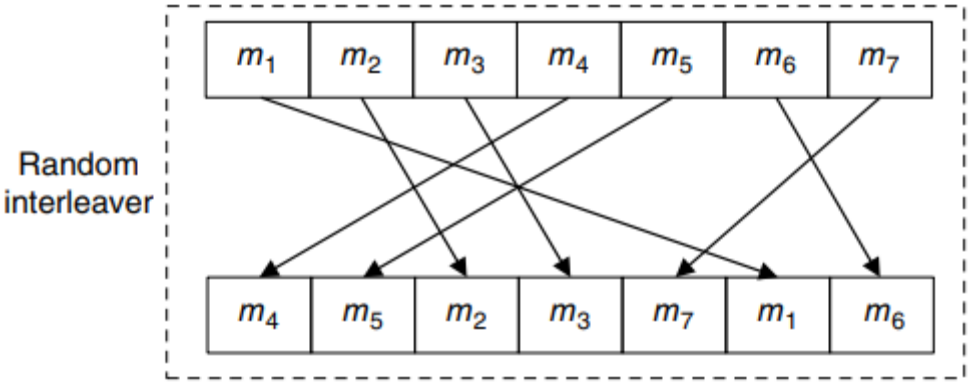


Figure 3: Random interleaver of the turbo encoder[1].

TRELLIS DIAGRAM

The turbo encoding process can be represented by the use of trellis diagram. As depicted from figure 2 above, we can observe that each RSC encoder consists of two shift registers hence the register states and corresponding output to the input bits are summarized in table 1 below. Furthermore the table can be transformed into a trellis diagram as shown in figure 4.

Table 1

Time (i)	Input mi	Memory State		Output (Ci0 Ci1)
		M1	M2	
1	1	0	0	1 1
2	0	1	0	0 1
3	0	1	1	0 1
4	1	0	1	1 1
5	0	0	0	0 0
6	0	1	0	0 0
7	0	0	1	0 0
8	0	1	0	0 0

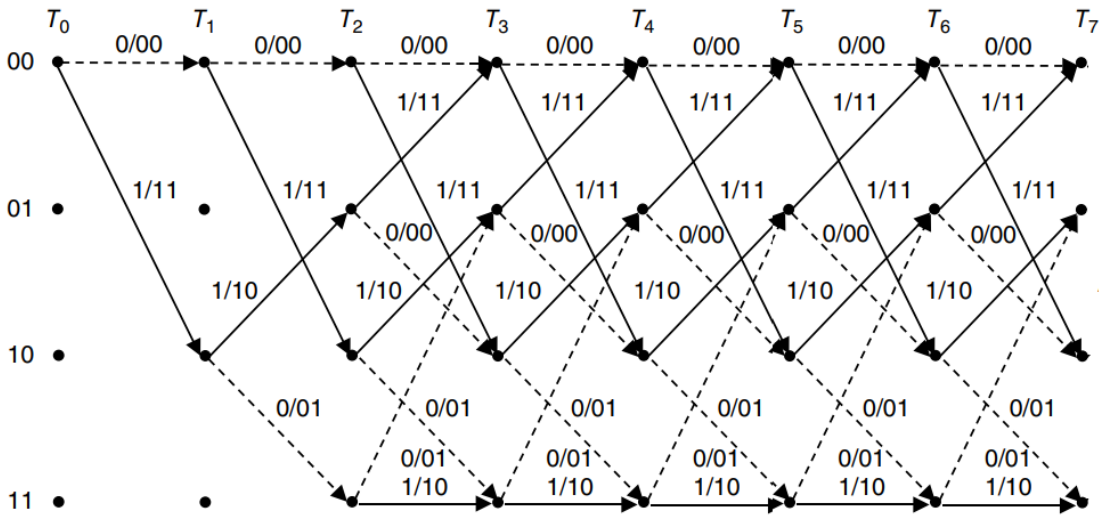


Figure 4: RSC encoder trellis diagram

TURBO CODE DECODER

Turbo Decoder is based on an iterative scheme and consists of two soft-input-soft-output (SISO) concatenated component decoders. As a result, the input to the decoder is the channel value actually received and not quantized to 0 or 1.The two configuration decoders use the same trellis structure and the same decoding algorithm. The iterative decoding scheme is based on a posterior probability. The two decoders provides and accepts the output as a priori information hence minimizing the probability of error in the original information bit. Figure 5 below depicts the block diagram of turbo code decoder.

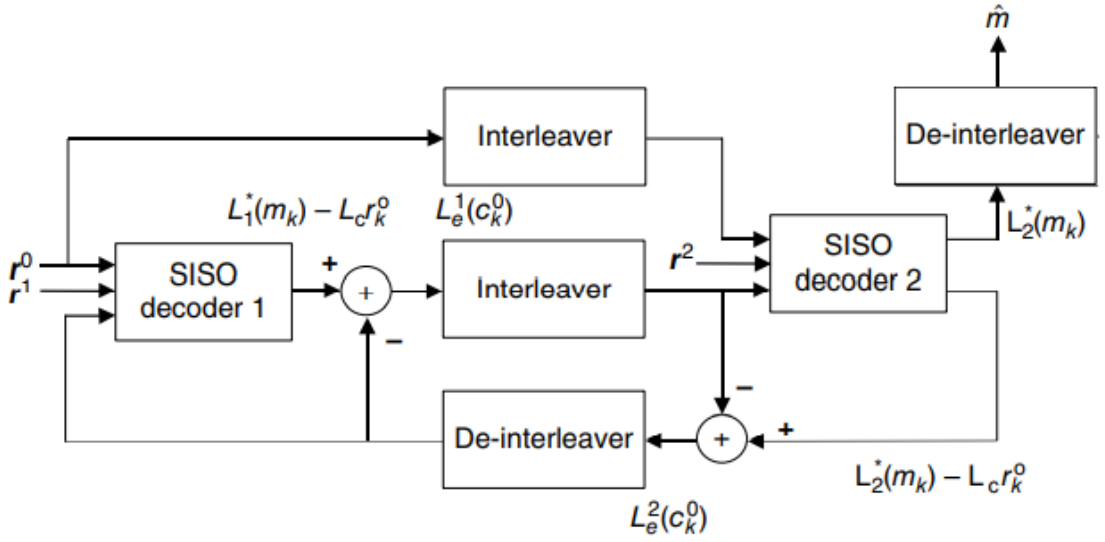


Figure 5: Turbo code decoder

OPERATION OF TURBO CODE DECODER

The turbo encoder RSC encoders 1 and 2 correspond to the turbo decoder SISO decoders 1 and 2, respectively. As observed in figure 5, there are three inputs (r0, r1, r2) to the turbo code decoders. The SISO decoder 1 uses the received information sequence (r0), the received parity sequence from RSC encoder 1 (r1), and the a priori information which is the de-interleaved extrinsic information from SISO decoder 2[1]. The inputs to the SISO decoder 2 are the interleaved received information sequence (r0), the received parity sequence by the RSC encoder 2 (r2), and the a priori information that is the interleaved extrinsic information from the SISO decoder 1[1]. On the first iteration, SISO Decoder 1 decode without a priori information and makes the output available to SISO Decoder 2 as a priori information. The SISO decoder 2 uses this a priori information, the interleaved received information r0, and the received parity r2 to initiate the decoding[1]. In the iterative process, SISO decoder 1 receives a priori information from SISO decoder 2 and decodes it again with a priori information, received information r0 and received parity r1[1]. This iteration allows the decoder to reduce the error probability of the information bits, but it increases the latency. Moreover, the higher the number of iterations, the faster the coding gain will drop[1].

The SISO decoder is based on the Maximum a Posteriori (MAP) algorithm (also known as Bahl-Cocke). Jelinek–Raviv (BCJR) algorithm)[1]. The algorithm uses the probability that the information bit u will be decoded correctly[10]. This probability is maximized by this algorithm, and the probability is called the maximum a posteriori probability[10]. The MAP decoder uses the log-likelihood ratio (LLR) to formulate the bit probabilities as soft outputs[1]. The soft output of the first MAP decoder is defined as:

$$L_1(m_k) = \log\left[\frac{P\left(m_k = +1 \middle| \left(r_k^0, r_k^1\right)\right)}{P\left(m_k = -1 \middle| \left(r_k^0, r_k^1\right)\right)}\right].$$

After observing the two inputs (rk0 and rk1), the LLR calculates and compares the probabilities when the information mk is +1 and -1[1]. The calculated soft output value of each information bit is decoded to be +1 if the value is greater zero otherwise its -1 if less than zero[10].

Using Bayes theorem, the probability that information mk is +1 is calculated as:

$$P\left(m_k = +1 \middle| \left(r_k^0, r_k^1\right)\right) = \frac{P\left(m_k = +1, \left(r_k^0, r_k^1\right)\right)}{P\left(r_k^0, r_k^1\right)}$$

$$= \frac{\sum_{(s',s)\in S^p} P\left(s_{k-1} = s', s_k = s \middle| \left(r_k^0, r_k^1\right)\right)}{P\left(r_k^0, r_k^1\right)}$$

$$= \frac{\sum_{(s',s)\in S^p} P\left(s_{k-1} = s', s_k = s, \left(r_k^0, r_k^1\right)\right) / P\left(r_k^0, r_k^1\right)}{P\left(r_k^0, r_k^1\right)}$$

Similarly, when mk = -1 its probability is calculated as:

$$P\left(m_k = -1 \middle| \left(r_k^0, r_k^1\right)\right) = \frac{P\left(m_k = -1, \left(r_k^0, r_k^1\right)\right)}{P\left(r_k^0, r_k^1\right)}$$

$$= \frac{\sum_{(s',s)\in S^n} P\left(s_{k-1} = s', s_k = s \middle| \left(r_k^0, r_k^1\right)\right)}{P\left(r_k^0, r_k^1\right)}$$

$$= \frac{\sum_{(s',s)\in S^n} P\left(s_{k-1} = s', s_k = s, \left(r_k^0, r_k^1\right)\right) / P\left(r_k^0, r_k^1\right)}{P\left(r_k^0, r_k^1\right)}$$

Hence, LLR can further be expressed as follows in terms of the trellis diagram:

$$L_1(m_k) = \log\left[\frac{\sum_{(s',s)\in S^p} P\left(s_{k-1} = s', s_k = s, \left(r_k^0, r_k^1\right)\right)}{\sum_{(s',s)\in S^n} P\left(s_{k-1} = s', s_k = s, \left(r_k^0, r_k^1\right)\right)}\right]$$

Where
Sk is state at time k in the trellis diagram,
S′ and S are a state of the trellis diagram, and
Sp and Sn are transition sets of the trellis diagram when input bit is positive and negative respectively as shown in figure 6 below.
The probability terms for the numerator and denominator of the LLR can be rewritten as:

$$P\left(s_{k-1} = s', s_k = s, \left(r_k^0, r_k^1\right)\right) = P\left(s', s, \left(r_k^0, r_k^1\right)\right)$$

$$= P\left(s', s, \left(r_{k-1}^0, r_{k-1}^1\right), \left(r_k^0, r_k^1\right), \left(r_{k+1}^0, r_{k+1}^1\right)\right)$$

$$= P\left(\left(r_{k+1}^0, r_{k+1}^1\right) \middle| s', s, \left(r_{k-1}^0, r_{k-1}^1\right), \left(r_k^0, r_k^1\right)\right) P\left(s', s, \left(r_{k-1}^0, r_{k-1}^1\right), \left(r_k^0, r_k^1\right)\right)$$

$$= P\left(\left(r_{k+1}^0, r_{k+1}^1\right) \middle| s', s, \left(r_{k-1}^0, r_{k-1}^1\right), \left(r_k^0, r_k^1\right)\right) P\left(s, \left(r_k^0, r_k^1\right) \middle| s', \left(r_{k-1}^0, r_{k-1}^1\right)\right) P\left(s', \left(r_{k-1}^0, r_{k-1}^1\right)\right)$$

$$= P\left(\left(r_{k+1}^0, r_{k+1}^1\right) \middle| s\right) P\left(s, \left(r_k^0, r_k^1\right) \middle| s'\right) P\left(s', \left(r_{k-1}^0, r_{k-1}^1\right)\right)$$

$$= P\left(\left(r_{k+1}^0, r_{k+1}^1\right) \middle| s_k = s\right) P\left(s_k = s, \left(r_k^0, r_k^1\right) \middle| s_{k-1} = s'\right) P\left(s_{k-1} = s', \left(r_{k-1}^0, r_{k-1}^1\right)\right)$$

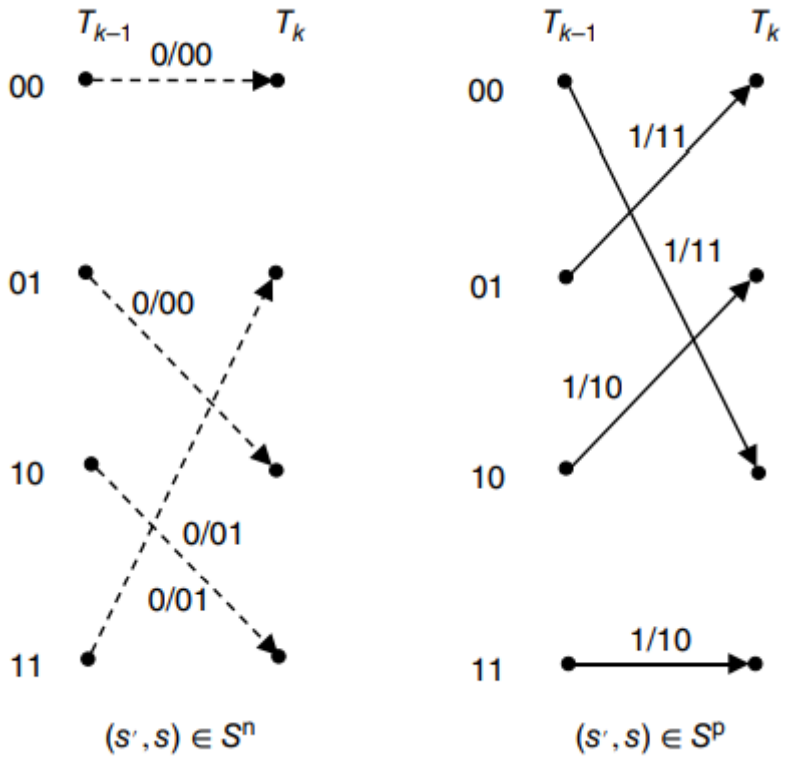


Figure 6: Transition sets of the trellis diagram of figure 4

Next, we express the following metrics, the forward path metric αk-1(sk-1 = s'), the backwards path metric (βk(sk=s)) and the branch metric (γk(sk-1=s',sk=s)) as follows:

$$\alpha_{k-1}(s_{k-1} = s') = P\left(s_{k-1} = s', \left(r_{k-1}^0, r_{k-1}^1\right)\right)$$

$$\beta_k(s_k = s) = P\left(\left(r_{k+1}^0, r_{k+1}^1\right) \middle| s_k = s\right)$$

$$\gamma_k\left(s_{k-1} = s', s_k = s\right) = P\left(s_k = s, \left(r_k^0, r_k^1\right) \middle| s_{k-1} = s'\right)$$

The probability P(sk-1= s',sk=s,(rk0,rk1)) can be expressed in terms of the metrics as follows

$$P\left(s_{k-1} = s', s_k = s, \left(r_k^0, r_k^1\right)\right) = \beta_k(s_k = s)\gamma_k(s_{k-1} = s', s_k = s)\alpha_{k-1}(s_{k-1} = s')$$

Furthermore, the LLR is expressed in terms in terms of metrics as follows:

$$L_1(m_k) = \log\left[\frac{\sum_{(s',s)\in S^p} \beta_k(s_k = s)\gamma_k(s_{k-1} = s', s_k = s)\alpha_{k-1}(s_{k-1} = s')}{\sum_{(s',s)\in S^n} \beta_k(s_k = s)\gamma_k(s_{k-1} = s', s_k = s)\alpha_{k-1}(s_{k-1} = s')}\right]$$

FORWARD METRIC CALCULATION

The time index for forward path metric calculation is considered from k-1 to k as follows

$$\alpha_k(s_k = s) = P\left(s_k = s, \left(r_k^0, r_k^1\right)\right)$$

$$= \sum_{s_{k-1}} P\left(s_{k-1} = s', \left(r_{k-1}^0, r_{k-1}^1\right)\right) P\left(s_k = s, \left(r_k^0, r_k^1\right) \middle| s_{k-1} = s'\right)$$

$$= \sum_{s_{k-1}} \alpha_{k-1}(s_{k-1} = s') \gamma_k(s_{k-1} = s', s_k = s).$$

Given that the recursive systematic convolution encoder starts at state 0 (00), the initial conditions for the forward path metric are given as follows:

$$\alpha_0(s_k = s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases}$$

BACKWARD PATH METRIC CALCULATION

The time index for backward path forward metric calculation is considered from k+1 to k as follows

$$\beta_k(s_k = s') = P\left(\left(r_{k+1}^0, r_{k+1}^1\right) \middle| s_k = s'\right)$$

$$= \sum_{s_{k+1}} P\left(s_{k+1} = s, \left(r_{k+1}^0, r_{k+1}^1\right) \middle| s_k = s'\right) P\left(\left(r_{k+2}^0, r_{k+2}^1\right) \middle| s_{k+1} = s\right)$$

$$= \sum_{s_{k+1}} \gamma_{k+1}(s_k = s', s_{k+1} = s) \beta_{k+1}(s_{k+1} = s).$$

Given that the recursive systematic convolution encoder terminates at state 0 (00), the initial conditions for the backward path metric are given as follows:

$$\beta_K(s_k = s') = \begin{cases} 1, & s' = 0 \\ 0, & s' \neq 0 \end{cases}$$

BRANCH METRIC CALCULATION

The branch metric is rewritten as follows:

$$\gamma_k\left(s_{k-1} = s', s_k = s\right) = P\left(s_k = s, \left(r_k^0, r_k^1\right) \middle| s_{k-1} = s'\right)$$

$$= P\left(\left(r_k^0, r_k^1\right) \middle| s_{k-1} = s', s_k = s\right) P\left(s_k = s \middle| s_{k-1} = s'\right)$$

$$= P\left(\left(r_k^0, r_k^1\right) \middle| \left(c_k^0, c_k^1\right)\right) P\left(c_k^0\right)$$

Where p((rk0,rk1) | (ck0,ck1)) is the likelihood probability and
P(ck0) is the a priori probability
The likelihood probability is given as follows:

$$P\left(\left(r_k^0, r_k^1\right)\middle|\left(c_k^0, c_k^1\right)\right) = P\left(r_k^0 \middle| c_k^0\right) P\left(r_k^1 \middle| c_k^1\right)$$

$$= \frac{\omega_0}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(r_k^0 - c_k^0\right)^2}{2\sigma^2}} \frac{\omega_1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(r_k^1 - c_k^1\right)^2}{2\sigma^2}}$$

$$= A_k e^{\left(r_k^0 c_k^0 - r_k^1 c_k^1\right)/\sigma^2}$$

And considering that ω0 and ω1 are extremely small values hence these terms are ignored and resulting in the priori probability is expressed as follows:

$$P\left(c_k^0\right) = \frac{e^{L_a\left(c_k^0\right)/2}}{1 + e^{L_a\left(c_k^0\right)}} e^{c_k^0 L_a\left(c_k^0\right)/2}$$

$$= B_k \, e^{c_k^0 L_a\left(c_k^0\right)/2}$$

Where

$$L_a\left(c_k^0\right) = \log\left[\frac{P\left(c_k^0 = +1\right)}{P\left(c_k^0 = -1\right)}\right].$$

Hence the branch metric is given as

$$\gamma_k\left(s_{k-1} = s', s_k = s\right) = A_k B_k e^{\left(c_k^0 L_a\left(c_k^0\right) + c_k^0 L_c r_k^0 + c_k^1 L_c r_k^1\right)/2}$$

Where

$$L_c = \frac{2}{\sigma^2}.$$

With that, we can further express LLR as follows:

$$L_1(m_k) = \log\left[\frac{\sum_{(s',s)\in S^p}\beta_k(s_k=s)\gamma_k(s_{k-1}=s',s_k=s)\alpha_{k-1}(s_{k-1}=s')}{\sum_{(s',s)\in S^n}\beta_k(s_k=s)\gamma_k(s_{k-1}=s',s_k=s)\alpha_{k-1}(s_{k-1}=s')}\right]$$

$$= \log\left[\frac{\sum_{(s',s)\in S^p}\beta_k(s_k=s)\left(e^{\left(c_k^0 L_a\left(c_k^0\right)+c_k^0 L_c r_k^0+c_k^1 L_c r_k^1\right)/2}\right)\alpha_{k-1}(s_{k-1}=s')}{\sum_{(s',s)\in S^n}\beta_k(s_k=s)\left(e^{\left(-c_k^0 L_a\left(c_k^0\right)-c_k^0 L_c r_k^0+c_k^1 L_c r_k^1\right)/2}\right)\alpha_{k-1}(s_{k-1}=s')}\right]$$

$$= \log\left[\frac{\sum_{(s',s)\in S^p} e^{c_k^0 L_a\left(c_k^0\right)/2} e^{c_k^0 L_c r_k^0/2}\beta_k\left(s_k=s\right)e^{c_k^1 L_c r_k^1/2}\alpha_{k-1}(s_{k-1}=s')}{\sum_{(s',s)\in S^n} e^{-c_k^0 L_a\left(c_k^0\right)/2} e^{-c_k^0 L_c r_k^0/2}\beta_k\left(s_k=s\right)e^{c_k^1 L_c r_k^1/2}\alpha_{k-1}(s_{k-1}=s')}\right]$$

And since $e^{\pm c_k^0 L_a\left(c_k^0\right)/2}$ and $e^{\pm c_k^0 L_c r_k^0/2}$ are independent of the state transition, LLR is further simplified as

$$L_1(m_k) = L_a\left(c_k^0\right) + L_c r_k^0 + \log\left[\frac{\sum_{(s',s)\in S^p}\beta_k(s_k=s)e^{c_k^1 L_c r_k^1/2}\alpha_{k-1}(s_{k-1}=s')}{\sum_{(s',s)\in S^n}\beta_k(s_k=s)e^{c_k^1 L_c r_k^1/2}\alpha_{k-1}(s_{k-1}=s')}\right]$$

$$= L_a\left(c_k^0\right) + L_c r_k^0 + L_e\left(c_k^0\right)$$

Where La(Ck0) is the a priori information of the MAP algorithm.
Lc(rk0) is the channel value.
Le(Ck0) is the extrinsic information that is going to be fed to the second MAP decoder.

EXAMPLE OF TURBO ENCODING AND DECODING

Referencing RSC encoder trellis diagram in figure 4. Given that input sequence is 1 0 0 1 0 and terminated with the last two bits 0 0 and the memory initial state being 0 0. The output of the first encoder is as follows: 1 1 0 1 0 1 1 1 0 0 0 0 0 0. Using the interleaver shown in figure 3 and its resultant is as follows: 0 1 1 0 0 0 0, therefore the output of the second encoder is as follows: 0 0 1 1 1 0 0 0 0 1 0 1 0 0. The output of the turbo code encoder is then given as follows: C0: 1 0 0 1 0 0 0, C1:1 1 1 1 0 0 0 and C2: 0 1 0 0 1 1 0. The transmitted symbols after BPSK modulated is sequentially given as follows: [C01 C11 C21, C02, C12, C22…C07, C17, C27] = [1 1 0 0 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0] = [+1 +1 -1 -1 +1 +1 -1 +1 -1 +1 +1 -1 -1 -1 +1 -1 -1 +1 -1 -1 -1].
The transmitted symbols are sent through a noisy channel hence a Gaussian noise is added to them as shown in figure .. below

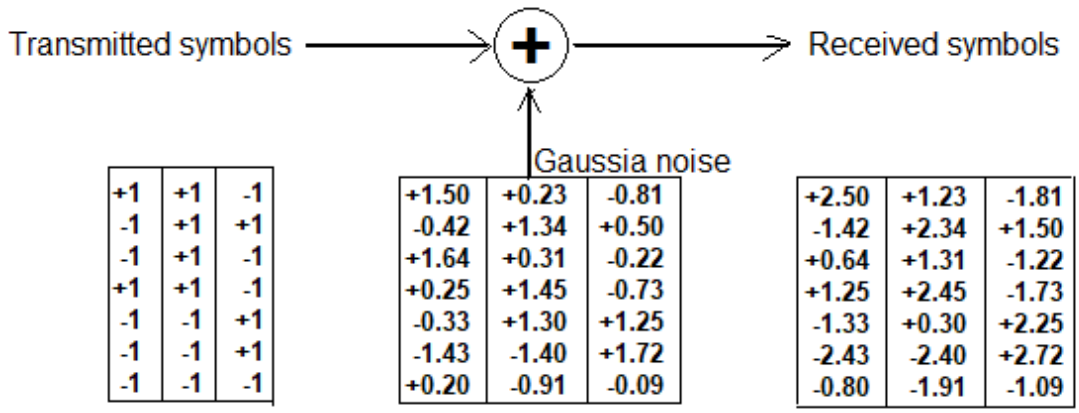
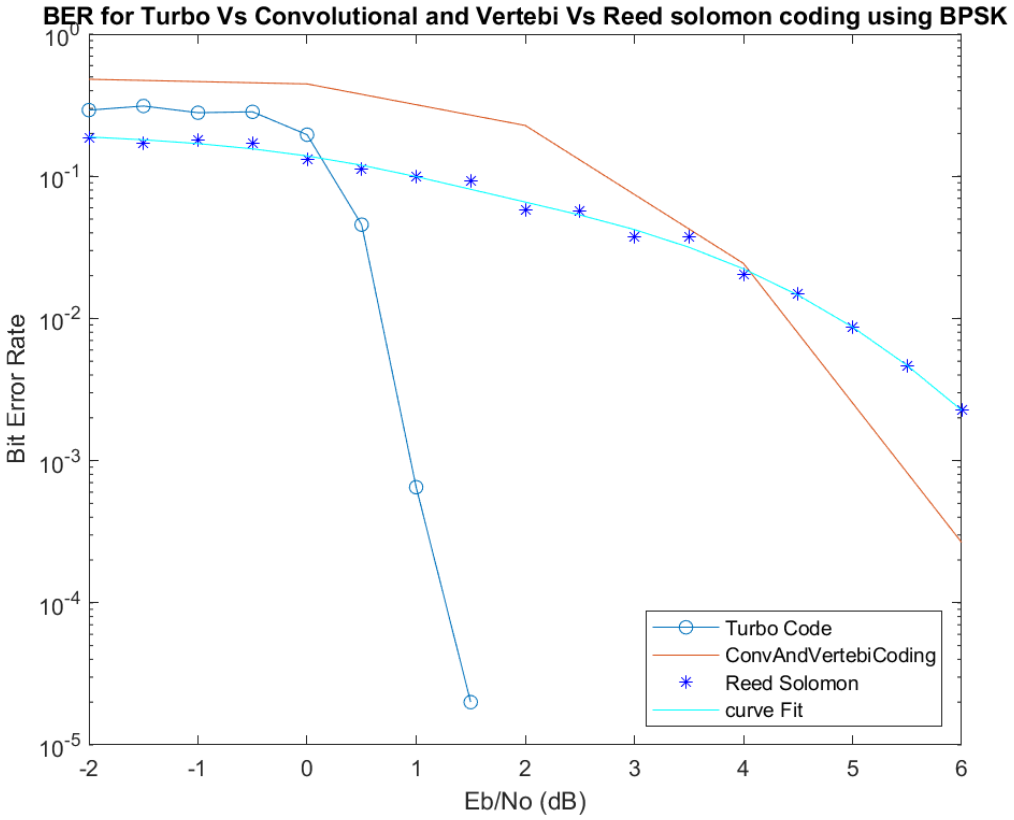


Figure: Received noisy symbol example

RESULTS AND ANALYSIS

Three channel coding were simulated, and their bit error rates (BER) were compared and their performance are depicted in figure 29 below. The channel coding under considerations are Turbo codes with soft decoding, convolutional encoding with vertebi decoding and Reed Solomon code using BPSK modulation.



As shown above, turbo codes has an outstanding performance among the three codes because it has low bit error rate at low signal to noise ratio (SNR). Reed Solomon code at low SNR has low bit error rate compared to convolutional codes with Vertebi decoding algorithm but as the SNR increases the convolutional coding gives low bit error rate than Reed Solomon code. It can be observed that as the SNR increases the performance of turbo codes does not change hence it is most ideal channel code since it gives low bit error rate at lower powers. This can be justified by recording the BER of all codes at SNR =1.5dB which records BER for turbo code in the range of 10-4 , convolutional code in the range of 10-1 and reed Solomon code in the range of 100.

CONCLUSION

Based on the matlab simulation, Turbo codes are the ideal channel code as it showed that it achieve low bit error rate al lower SNR. It also has better bandwidth efficiency and provide better communication security. It will recommend that it should be incorporated in future generation implementation of communication systems and further be improved. The implementation of turbo codes using DSP has further improved as delays are being minimised.

REFERENCES

[1] H. Kim, Wireless Communications Systems Design, 1st ed. John Wiley & Sons, Ltd, 2015.

[2] D. D. P. Agrawal and D. Q.-A. Zeng, “Channel Coding,” 2012. https://plato.csie.ncku.edu.tw/2012Fall_WIRELESS/Chapt-04.pdf (accessed Feb. 18, 2021).

[3] H. Chen, “Turbo Codes.” http://web.ee.nchu.edu.tw/~code/course/course1/turbo codes.pdf (accessed Feb. 18, 2021).

[4] R. Muthammal, “Turbo codes,” 2015. https://www.researchgate.net/figure/Turbo-codeencoder-32-Turbo-decoder-The-figure-shows-how-a-turbo-code-is-decoded-when_fig2_321669464.

[5] M. A. Fleah and Q. F. Al-Doori, “Design and Implementation of Turbo encoder/ decoder using FPGA,” 2019, doi: 10.1109/CAS47993.2019.9075589.

[6] E. M. A. Abdel-azim, “Design and implementation for a multi-standard turbo decoder using a reconfigurable asip,” CAIRO UNIVERSITY, 2013.

[7] S. DIGITAL, TMS320C6713 DSK Technical Reference. Stafford: SPECTRUM DIGITAL INC, 2003.

[8] R. Chassaing and D. S. Reay, Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK, 2nd ed. Wiley-IEEE Press, 2011.

[9] J. Vucetic, B., Yuan, Turbo Codes. Springer, Boston, MA, 2000.

[10] A. S. Babu and M. A. Ambroze, “From Convolutional Codes to Turbo Codes,” Plymouth University, 2015.