Chapter 3. Turbo Code Encoder

This chapter describes the turbo code encoder and its components in detail. The fundamental turbo code encoder is built using two identical recursive systematic convolutional (RSC) codes with parallel concatenation [Ber93]. An RSC encoder is typically $r = \frac{1}{2}$ and is termed a component encoder. The two component encoders are separated by an interleaver. Only one of the systematic outputs from the two component encoders is used, because the systematic output from the other component encoder is just a permuted version of the chosen systematic output. Figure 3.1 shows the fundamental turbo code encoder.

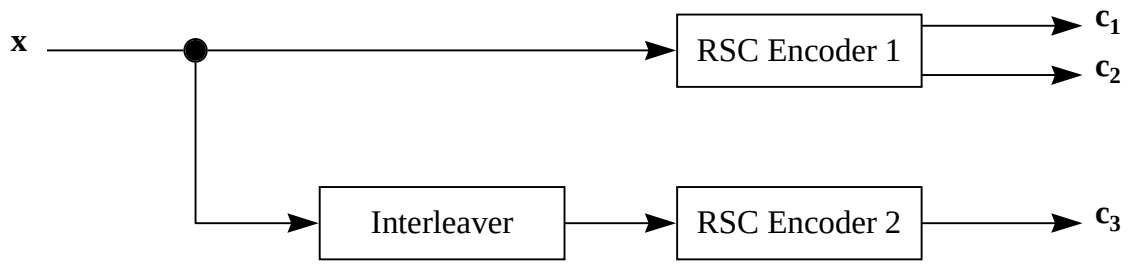

Figure 3.1: Fundamental turbo code encoder.

Figure 3.1 shows an $r = \frac{1}{3}$ turbo code encoder. The first RSC encoder outputs the systematic $\mathbf{c_1}$ and recursive convolutional $\mathbf{c_2}$ sequences while the second RSC encoder discards its systematic sequence and only outputs the recursive convolutional $\mathbf{c_3}$ sequence.

3.1 Recursive Systematic Convolutional (RSC) Encoder

The recursive systematic convolutional (RSC) encoder is obtained from the nonrecursive nonsystematic (conventional) convolutional encoder by feeding back one of its encoded outputs to its input. Figure 3.2 shows a conventional convolutional encoder.

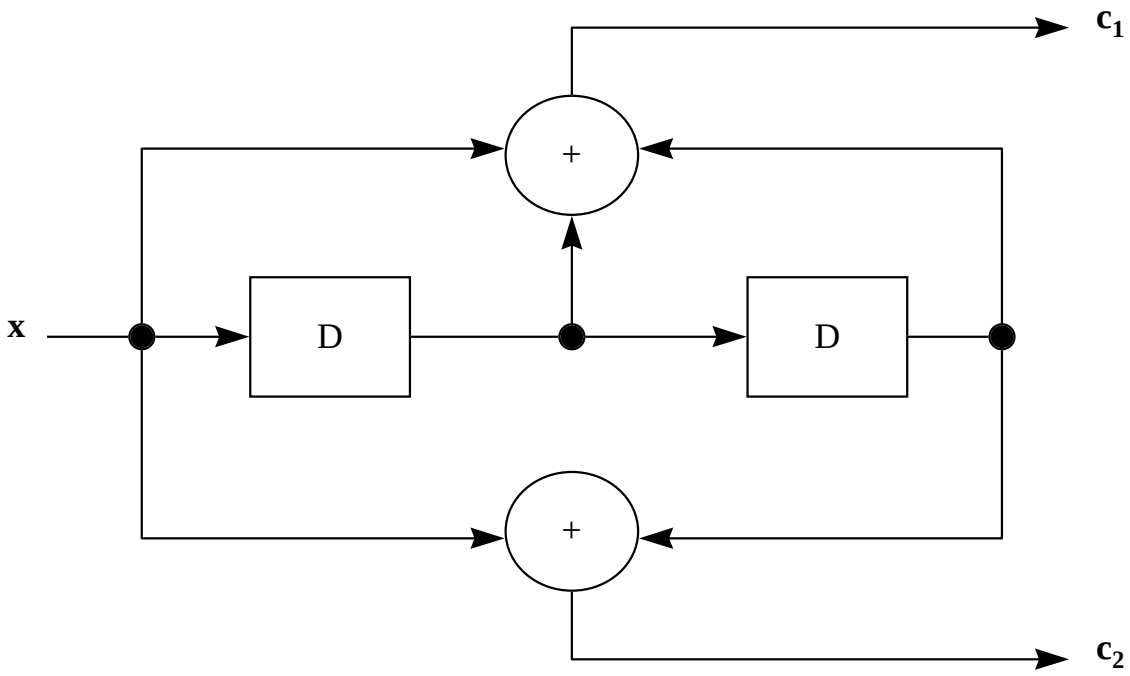

Figure 3.2: Conventional convolutional encoder with r=1/2 and K=3.

The conventional convolutional encoder is represented by the generator sequences $\mathbf{g_1}$ =[111] and $\mathbf{g_2}$ =[101] and can be equivalently represented in a more compact form as $\mathbf{G}$=[$\mathbf{g_1}$, $\mathbf{g_2}$]. The RSC encoder of this conventional convolutional encoder is represented as $\mathbf{G}$=[1, $\mathbf{g_2}$ / $\mathbf{g_1}$] where the first output (represented by $\mathbf{g_1}$) is fed back to the input. In the above representation, 1 denotes the systematic output, $\mathbf{g_2}$ denotes the feedforward output, and $\mathbf{g_1}$ is the feedback to the input of the RSC encoder. Figure 3.3 shows the resulting RSC encoder.

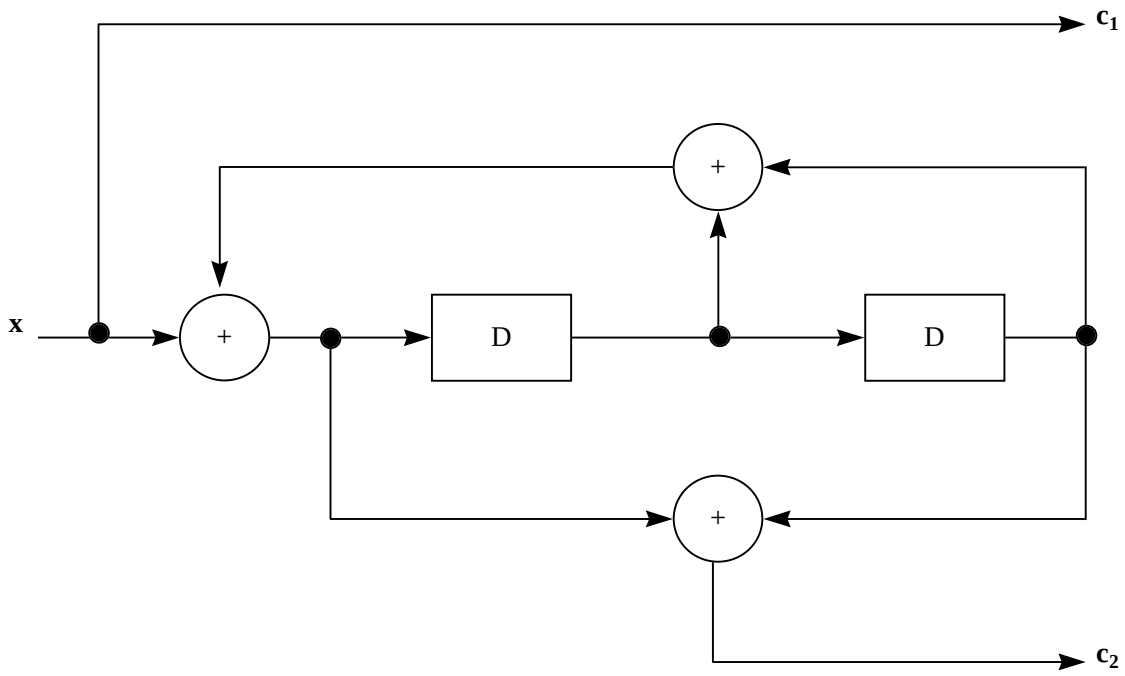

Figure 3.3: The RSC encoder obtained from Figure 3.2 with r=1/2 and K=3.

It was suggested in [Bat93] that good codes can be obtained by setting the feedback of the RSC encoder to a primitive polynomial, because the primitive polynomial generates maximum-length sequences which adds randomness to the turbo code.

3.1.1 Trellis Termination

For the conventional convolutional encoder, the trellis is terminated by inserting m = K-1 additional zero bits after the input sequence. These additional bits drive the conventional convolutional encoder to the all-zero state (trellis termination). However, this strategy is not possible for the RSC encoder due to the feedback. The additional termination bits for the RSC encoder depend on the state of the encoder and are very difficult to predict [Div95a]. Furthermore, even if the termination bits for one of the component encoders are found, the other component encoder may not be driven to the all-zero state with the same m termination bits due to the presence of the interleaver between the component encoders. Figure 3.4 shows a simple strategy that has been developed in [Div95a] which overcomes this problem.

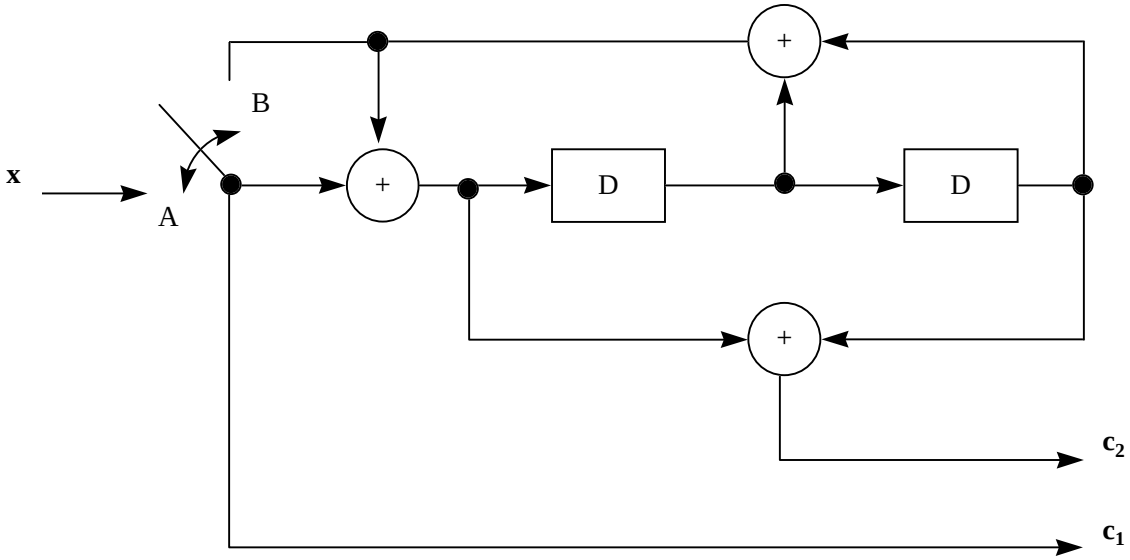

Figure 3.4: Trellis termination strategy for RSC encoder.

For encoding the input sequence, the switch is turned on to position A and for terminating the trellis, the switch is turned on to position B.

3.2 Recursive and Nonrecursive Convolutional Encoders

The nature of recursive and nonrecursive convolutional encoders is best examined by an example. Figure 3.5 shows a simple nonrecursive convolutional encoder with generator sequences $\mathbf{g}_1$=[11] and $\mathbf{g}_2$=[10].

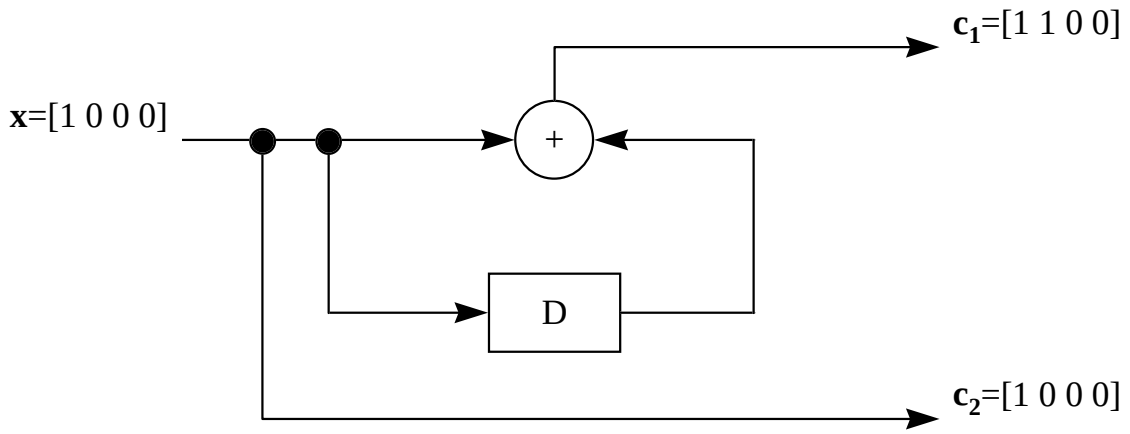

Figure 3.5: Nonrecursive r=1/2 and K=2 convolutional encoder with input and output sequences.

Figure 3.6 shows the equivalent recursive convolutional encoder of Figure 3.5 with $\mathbf{G}=[1, \mathbf{g}_2 / \mathbf{g}_1]$.

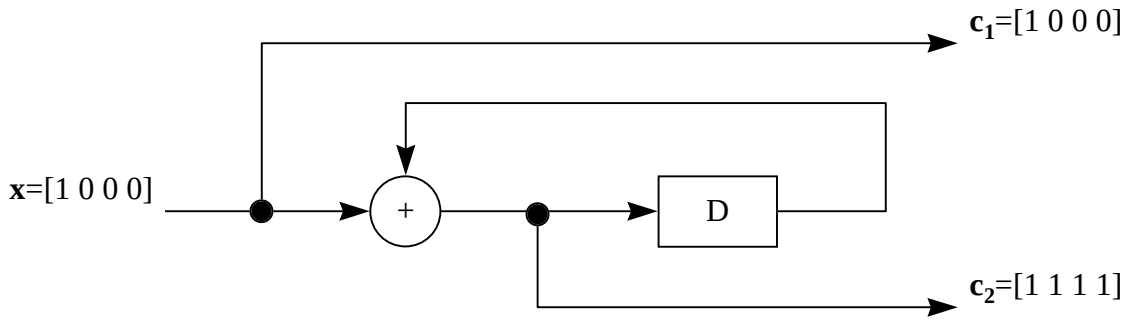

Figure 3.6: Recursive r=1/2 and K=2 convolutional encoder of Figure 3.5 with input and output sequences.

From Figure 3.5 and Figure 3.6, given the same input sequence, the nonrecursive encoder produces an output codeword with weight of 3 and the recursive encoder produces an output codeword with weight of 5. Thus, a recursive convolutional encoder tends to produce codewords with increased weight relative to a nonrecursive encoder. This results in fewer codewords with lower weights and this leads to better error performance. For turbo codes, the main purpose of implementing RSC encoders as component encoders is to utilize the recursive nature of the encoders and not the fact that the encoders are systematic [Div95b].

Figure 3.7 shows the state diagram of the nonrecursive encoder.

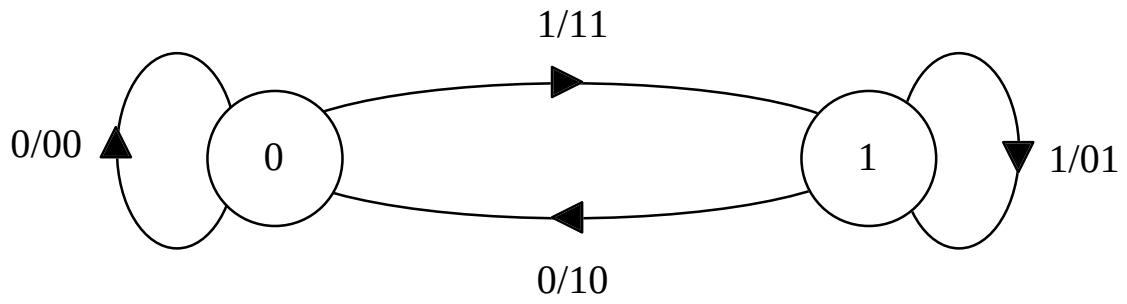

Figure 3.7: State diagram of the nonrecursive encoder in Figure 3.5.

Figure 3.8 shows the state diagrams of the recursive encoder.

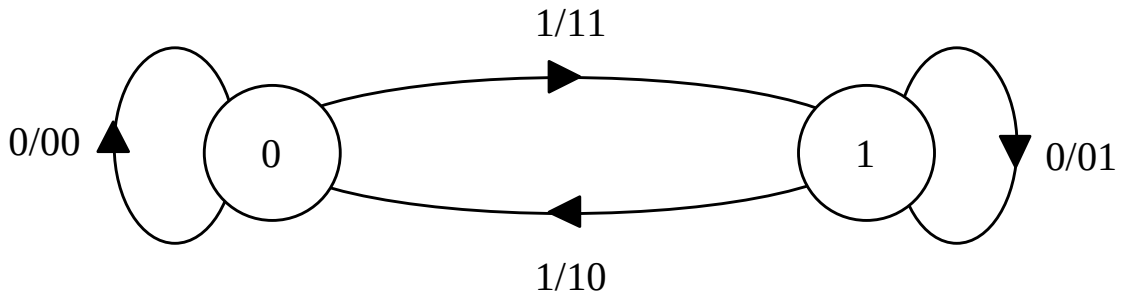

Figure 3.8: State diagram of recursive encoder in Figure 3.6.

Clearly, the state diagrams of the encoders are very similar. The transfer function for both encoders are identical [Ben96] and is found to be $T(D) = \frac{D^3}{1-D}$ where N and J are neglected. Furthermore, the two codes have the same minimum free distance and can be described by the same trellis structure [Ber93]. Thus, the codes have the same first event error probability [Ben96]. However, these two codes have different bit error rates (BERs). This is due to the fact that BER depends on the input-output correspondence of the encoders [Ben96]. It has been shown that the BER for a recursive convolutional code is lower than that of the corresponding nonrecursive convolutional code at low signal-to-noise ratios E_b/N_o [Ber93], [Ben96].

3.3 Concatenation of Codes

A concatenated code is composed of two separate codes that are combined to form a larger code [Pro95]. There are two types of concatenation, namely serial and parallel concatenations. Figure 3.9 shows the serial concatenation scheme for transmission.

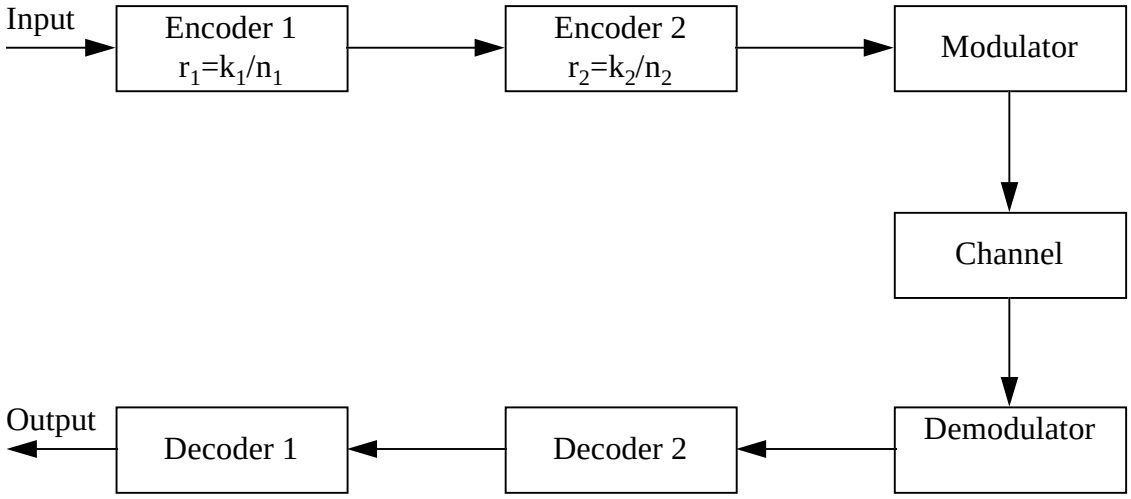

Figure 3.9: Serial concatenated code.

The total code rate for serial concatenation is

$$r_{tot} = \frac{k_1 k_2}{n_1 n_2} \tag{3.1}$$

which is equal to the product of the two code rates [Pro95]. Figure 3.10 shows the parallel concatenation scheme for transmission.

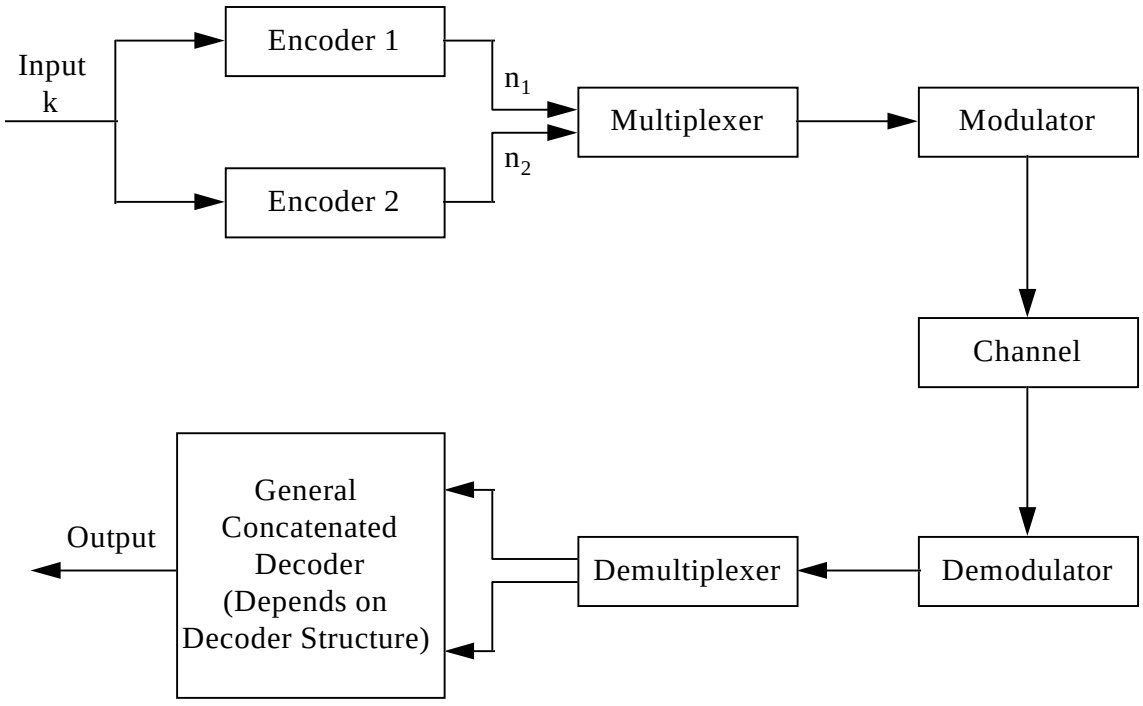

Figure 3.10: Parallel concatenated code.

The total code rate for parallel concatenation is

$$r_{tot} = \frac{k}{n_1 + n_2} \tag{3.2}$$

For both serial and parallel concatenation schemes, an interleaver is often used between the encoders to improve burst error correction capacity or to increase the randomness of the code.

Turbo codes use the parallel concatenated encoding scheme. However, the turbo code decoder is based on the serial concatenated decoding scheme. The serial concatenated decoders are used because they perform better than the parallel concatenated decoding scheme due to the fact that the serial concatenation scheme has the ability to share information between the concatenated decoders whereas the decoders for the parallel concatenation scheme are primarily decoding independently. In Chapter 4, it will be shown how the serial concatenated decoding scheme is implemented for a turbo code.

3.4 Interleaver Design

For turbo codes, an interleaver is used between the two component encoders. The interleaver is used to provide randomness to the input sequences. Also, it is used to increase the weights of the codewords as shown in Figure 3.11.

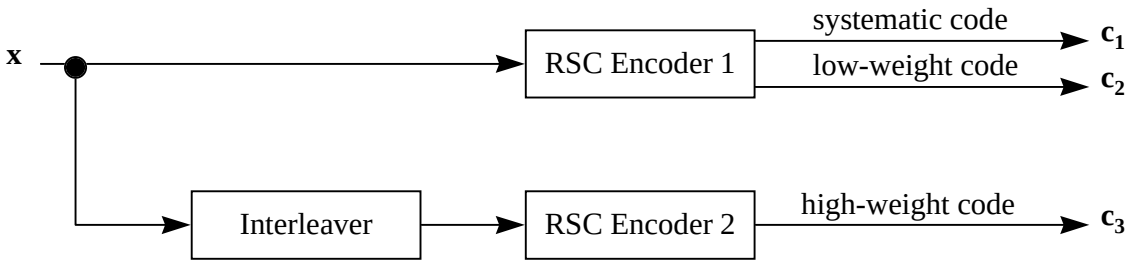

Figure 3.11: The interleaver increases the code weight for RSC Encoder 2 as compared to RSC Encoder 1.

From Figure 3.11, the input sequence **x** produces a low-weight recursive convolutional code sequence $\mathbf{c_2}$ for RSC Encoder 1. To avoid having RSC Encoder 2 produce another low-weight recursive output sequence, the interleaver permutes the input sequence **x** to obtain a different sequence that hopefully produces a high-weight recursive convolutional code sequence $\mathbf{c_3}$. Thus, the turbo code's code weight is moderate, combined from Encoder 1's low-weight code and Encoder 2's high-weight code. Figure 3.12 shows an illustrative example.

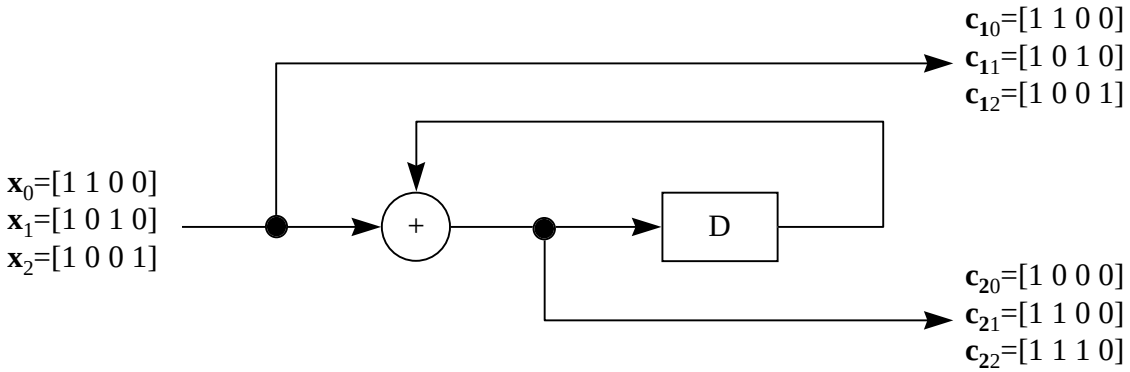

Figure 3.12: An illustrative example of an interleaver's capability.

From Figure 3.12, the input sequence $\mathbf{x}_i$ produces output sequences $\mathbf{c}_{1i}$ and $\mathbf{c}_{2i}$ respectively. The input sequences $\mathbf{x}_1$ and $\mathbf{x}_2$ are different permuted sequences of $\mathbf{x}_0$. Table 3.1 shows the resulting codewords and weights.

Table 3.1: Input and Output Sequences for Encoder in Figure 3.12

	Input Sequence $\mathbf{x}_i$	Output Sequence $\mathbf{c}_{1i}$	Output Sequence $\mathbf{c}_{2i}$	Codeword Weight i
i = 0	1 1 0 0	1 1 0 0	1 0 0 0	3
i = 1	1 0 1 0	1 0 1 0	1 1 0 0	4
i = 2	1 0 0 1	1 0 0 1	1 1 1 0	5

As it can be seen from Table 3.1, the codeword weight can be increased by utilizing an interleaver.

The interleaver affects the performance of turbo codes because it directly affects the distance properties of the code [Jun94]. By avoiding low-weight codewords, the BER of a turbo code can improve significantly. Thus, much research has been done on interleaver design. The following subsections show representative interleavers commonly used in turbo code design.

3.4.1 Block Interleaver

The block interleaver is the most commonly used interleaver in communication systems. It writes in column wise from top to bottom and left to right and reads out row wise from left to right and top to bottom. Figure 3.13 shows a block interleaver.

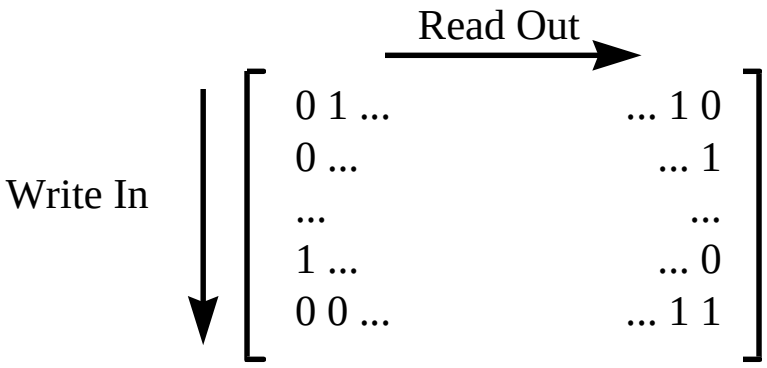

Figure 3.13: Block interleaver.

From Figure 3.13, the interleaver writes in [0 0 ... 1 0 1 ... 0 ... 1 ... 1 0 1 ... 0 1] and reads out [0 1 ... 1 0 0 ... 1 ... 1 ... 0 0 0 ... 1 1].

3.4.2 Random (Pseudo-Random) Interleaver

The random interleaver uses a fixed random permutation and maps the input sequence according to the permutation order. The length of the input sequence is assumed to be L. Figure 3.14 shows a random interleaver with L=8.

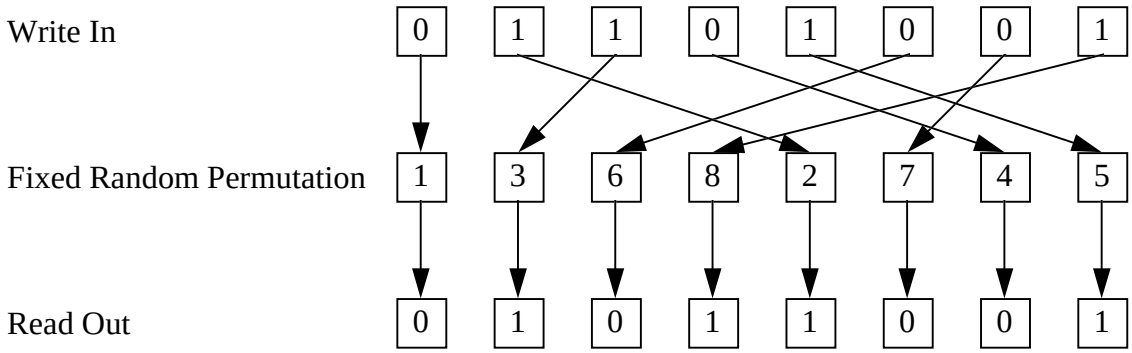

Figure 3.14: A random (pseudo-random) interleaver with L=8.

From Figure 3.14, the interleaver writes in [0 1 1 0 1 0 0 1] and reads out [0 1 0 1 1 0 0 1].

3.4.3 Circular-Shifting Interleaver

The permutation **p** of the circular-shifting interleaver is defined by

$$p(i) = (ai + s) \bmod L \qquad (3.3)$$

satisfying $a < L$, a is relatively prime to L, and $s < L$ where i is the index, a is the step size, and s is the offset [Dol95]. Figure 3.15 shows a circular-shifting interleaver with L=8, a=3, and s=0.

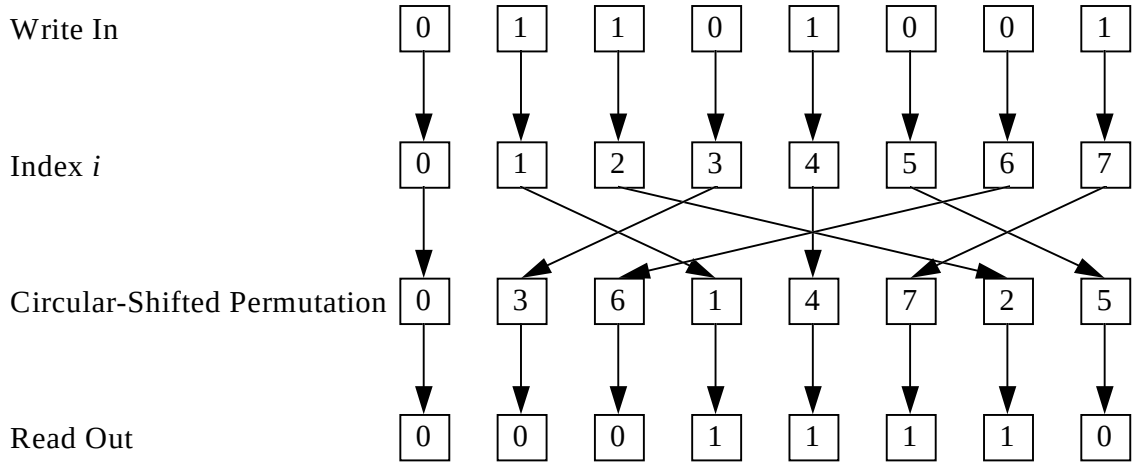

Figure 3.15: A circular-shifting interleaver with L=8, a=3, and s=0.

From Figure 3.15, the interleaver writes in [0 1 1 0 1 0 0 1] and reads out [0 0 0 1 1 1 1 0]. Also, it can be seen that the adjacent bit separation is either 3 or 5. This type of interleaver has been shown to do a very good job of permuting weight-2 input sequences with low codeword weights into weight-2 input sequences with high codeword weights. However, because of the regularity (3 or 5 adjacent bit separation for Figure 3.15) inherent in this type of interleaver, it may be difficult to permute higher weight (weight>2) input sequences with low codeword weights into other input sequences with high codeword weights [Dol95].

3.4.4 Semirandom Interleaver

The semirandom interleaver is a compromise between a random interleaver and a "designed" interleaver such as the block and circular-shifting interleavers [Dol95]. The permutation algorithm for the semirandom interleaver is described below [Dol95].

Step 1. Select a random index $i \in [0, L\text{-}1]$.

Step 2. Select a positive integer $S < \sqrt{\frac{L}{2}}$.

Step 3. Compare i to previous S integers. For each of the S integers, compare i to see if it lies within $\pm S$. If i does lie within the range, then go back to Step 1. Otherwise, keep i.

Step 4. Go back to Step 1 until all L positions have been filled.

Figure 3.16 shows a semirandom interleaver with L=16 and S=2.

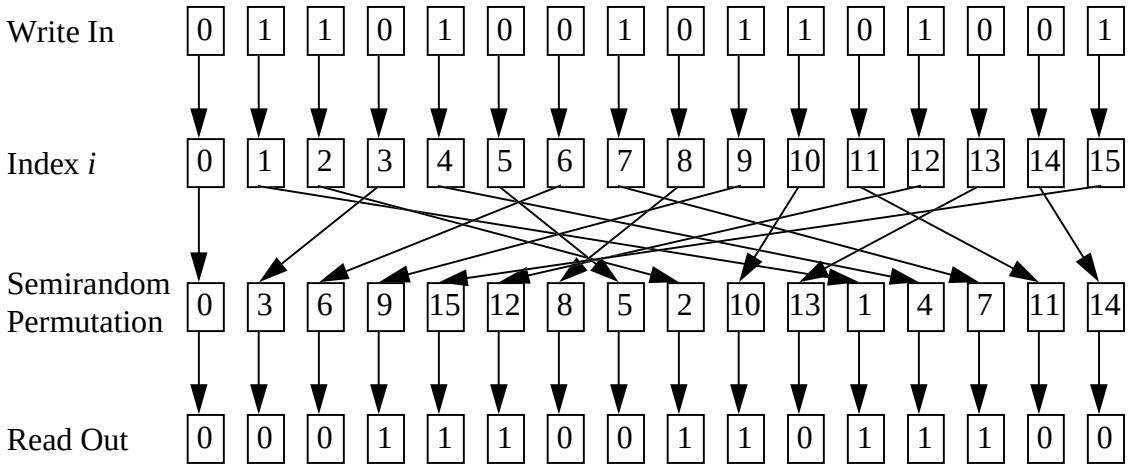

Figure 3.16: A semirandom interleaver with *L*=16 and *S*=2.

From Figure 3.16, the interleaver writes in [0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1] and reads out [0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0]. The semirandom interleaver tries to introduce some randomness to overcome the permutation regularity; however, the algorithm does not guarantee to finish successfully [Dol95].

3.4.5 Odd-Even Interleaver Design

The odd-even interleaver design is specifically for the $r = \frac{1}{2}$ turbo code. A $r = \frac{1}{2}$ turbo code is obtained by puncturing the two coded (nonsystematic) output sequences of a $r = \frac{1}{3}$ turbo code. However, by puncturing these two coded output sequences, it is possible that an information (systematic) bit may not have any of its coded bits (both of the associated coded bits may be punctured out). Likewise, it is also possible for an information bit to have one or both of its coded bits. Thus, if an error occurs for an unprotected information bit (without any of its coded bits), the turbo code decoder may degrade on its performance.

The odd-even interleaver design overcomes this problem by allowing each information bit to have exactly one of its coded bits. As a result of this interleaver, the error correction capability of the code is uniformly distributed over all information bits [Bar94]. The following is an example which illustrates this type of interleaver design. The labels used are from the fundamental turbo code encoder shown in Figure 3.1.

The information (systematic) sequence $\mathbf{x}=\mathbf{c_1}$ of L=9 produces a coded sequence $\mathbf{c_2}$ for RSC Encoder 1. From sequence $\mathbf{c_2}$, only the odd positioned coded bits are stored as

shown in Figure 3.17. Note that the plain subscript denotes the bit position within a bit sequence.

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$	$\mathbf{x}_7$	$\mathbf{x}_8$	$\mathbf{x}_9$
$\mathbf{c}_{21}$	-	$\mathbf{c}_{23}$	-	$\mathbf{c}_{25}$	-	$\mathbf{c}_{27}$	-	$\mathbf{c}_{29}$

Figure 3.17: Odd coded bits of sequence $\mathbf{c}_2$ are stored for information (systematic) sequence x.

A 3 x 3 block interleaver is used to permute the information sequence **x** for RSC Encoder 2 as shown in Figure 3.18.

$\mathbf{x}_1$	$\mathbf{x}_4$	$\mathbf{x}_7$
$\mathbf{x}_2$	$\mathbf{x}_5$	$\mathbf{x}_8$
$\mathbf{x}_3$	$\mathbf{x}_6$	$\mathbf{x}_9$

Figure 3.18: 3 x 3 block interleaver.

The information sequence **x** is wrote in column wise and read out row wise. The permuted information sequence **x** produces a coded sequence $\mathbf{c}_3$. From sequence $\mathbf{c}_3$, only the even positioned coded bits are stored as shown in Figure 3.19.

$\mathbf{x}_1$	$\mathbf{x}_4$	$\mathbf{x}_7$	$\mathbf{x}_2$	$\mathbf{x}_5$	$\mathbf{x}_8$	$\mathbf{x}_3$	$\mathbf{x}_6$	$\mathbf{x}_9$
-	$\mathbf{c}_{34}$	-	$\mathbf{c}_{32}$	-	$\mathbf{c}_{38}$	-	$\mathbf{c}_{36}$	-

Figure 3.19: Even coded bits of sequence $\mathbf{c}_3$ are stored for permuted information sequence x.

For the $r = \frac{1}{2}$ turbo code, the coded bit sequences must then be multiplexed together as shown in Figure 3.20.

$\mathbf{x}_1$	$\mathbf{x}_4$	$\mathbf{x}_7$	$\mathbf{x}_2$	$\mathbf{x}_5$	$\mathbf{x}_8$	$\mathbf{x}_3$	$\mathbf{x}_6$	$\mathbf{x}_9$
$\mathbf{c}_{21}$	$\mathbf{c}_{34}$	$\mathbf{c}_{27}$	$\mathbf{c}_{32}$	$\mathbf{c}_{25}$	$\mathbf{c}_{38}$	$\mathbf{c}_{23}$	$\mathbf{c}_{36}$	$\mathbf{c}_{29}$

Figure 3.20: Information sequence x and multiplexed coded sequence.

From Figure 3.20, it can be seen that each information bit has its own coded bit.

3.4.6 Optimal (Near-Optimal) Interleaver

The optimal interleaver can be described as the interleaver that produces the fewest output coded sequences with low weights [Bar94]. This interleaver design is both tedious and exhaustive. The following algorithm describes the interleaver design concept.

1. Generate a random interleaver.
2. Generate all possible input information sequences.
3. For all possible input information sequences, encode each of the input information sequences and determine the resulting codeword weight. This gives the weight distribution of the code.
4. Determine the minimum codeword weight and the number of codewords with that weight.

This algorithm is repeated for a "reasonable number of times" and keeps the interleaver that produces the largest minimum codeword weight with the lowest number of codewords of that weight.