Lecture 3 : Introduction to Binary Convolutional Codes

Binary Convolutional Codes

1. Convolutional codes were first introduced by Elias in 1955 as an alternative to block codes. In contrast with a block code, whose encoder assigns an n-bit codeword to each block of k information bits, a convolutional encoder assigns code bits to an incoming information bit stream continuously.
2. Convolutional codes differ from block codes in that the encoder contains memory and the n encoder outputs at any time unit depend not only on the k inputs but also on m previous input blocks.
3. A binary convolutional code is denoted by a three-tuple (n, k, m) with a k-input, n-output linear sequential circuit with input memory m.
4. The current n outputs are linear combinations of not only the present k input bits, but also the previous $m \times k$ input bits.

5. m denotes the number of previous k-bit input blocks which would be memorized in the encoder.

6. m is called the *memory order* of the convolutional code. Typically, n and k are small integers with $k < n$, but the memory order m must be made large to achieve low error probabilities.

7. Then in 1967, Viterbi proposed a maximum likelihood decoding scheme that was relatively easy to implement for cods with small memory orders.

8. This scheme, called Viterbi decoding, together with improved versions of sequential decoding, led to the application of convolutional codes to deep-space and satellite communication in early 1970s.

9. Code rate $R_c = \frac{k}{n}$

Encoders for the Convolutional Codes

1. A binary convolutional encoder is structured as a mechanism of shift registers and modulo-2 adders, where the output bits are mod-2 additions of selective shift register contents and present input bits.

2. n is the number of output sequences.

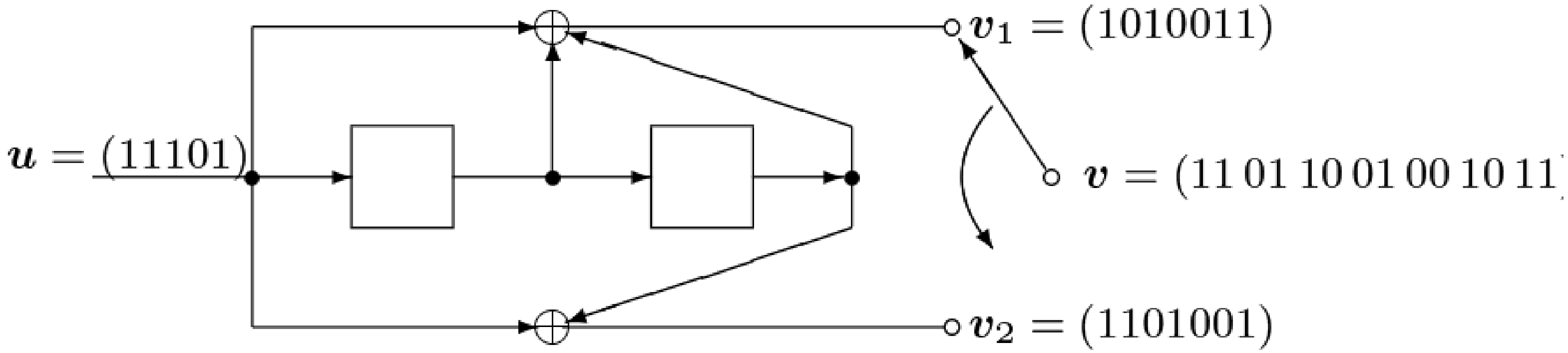
3. k is the number of input sequences ($k = 1$ is usually used).

4. m is the maximum length of the k parallel shift registers. If the number of stages of the j-th shift registers is K_j, $m = \max_{1 \le j \le k} K_j$.

5. $K = \sum_{j=1}^{k} K_j$ is the total memories utilized in the encoder. K is also called the *overall constraint lengths*.

6. The definition of *constraint length* of a convolutional code is defined as $m + 1$.

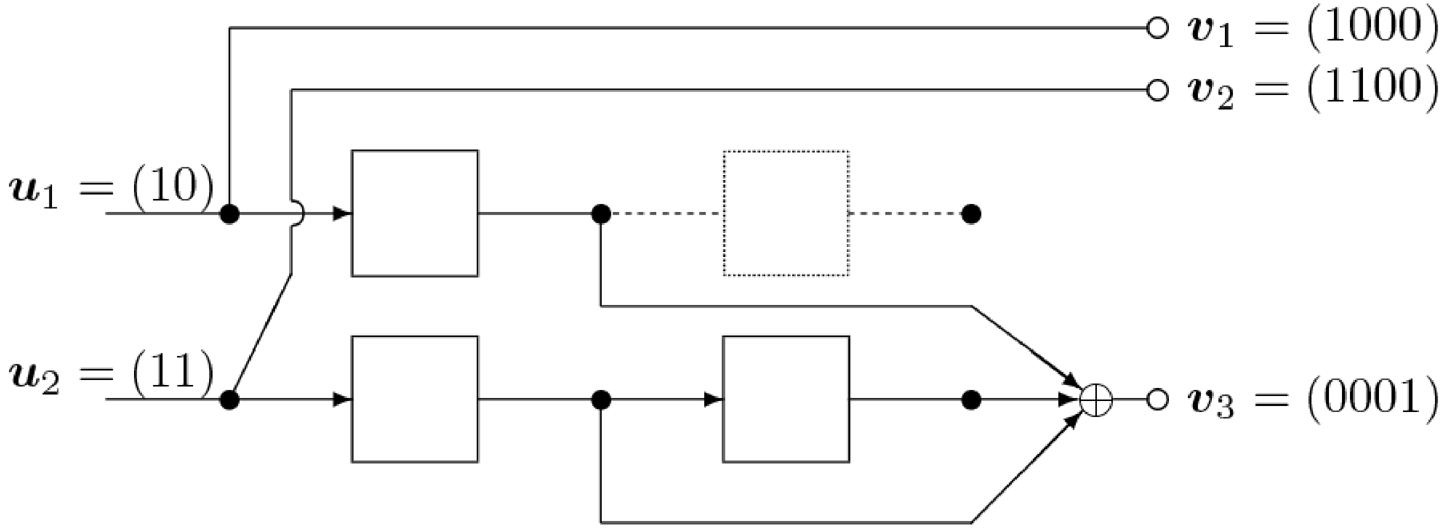
★ Example 1: Encoder for a binary (2,1,2) code



★ $\mathbf{u}$ is the information sequence and $\mathbf{v} = \mathrm{MUX}[\mathbf{v}_1, \mathbf{v}_2]$ is the corresponding codeword.

★ In octal form, the generator is denoted as (7,5).

★ Example 2: Encoder for a binary (3, 2, 2) convolutional code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001)$

Impose Response and Convolution

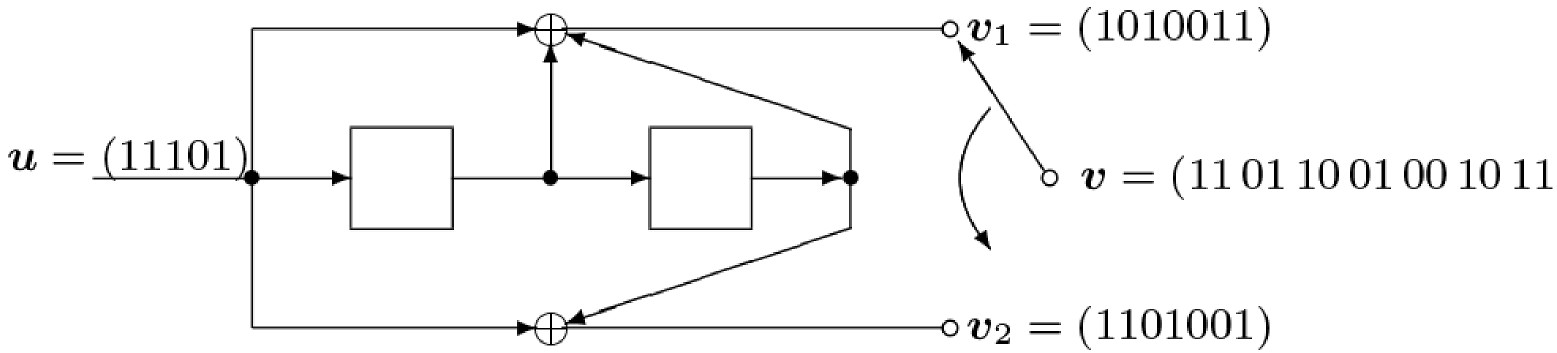
1. The encoders of convolutional codes can be represented by *linear time-invariant* (LTI) systems.

2. $\mathbf{v}_j = \mathbf{u}_1 * \mathbf{g}_1^{(j)} + \mathbf{u}_2 * \mathbf{g}_2^{(j)} + \cdots + \mathbf{u}_k \mathbf{g}_k^{(j)} = \sum_{i=1}^{k} \mathbf{u}_i * \mathbf{g}_j^{(i)}$,
   where $*$ denotes the convolutional operation and $\mathbf{g}_i^{(j)}$ denotes the impulse response of the ith input sequence with the response to the jth output.

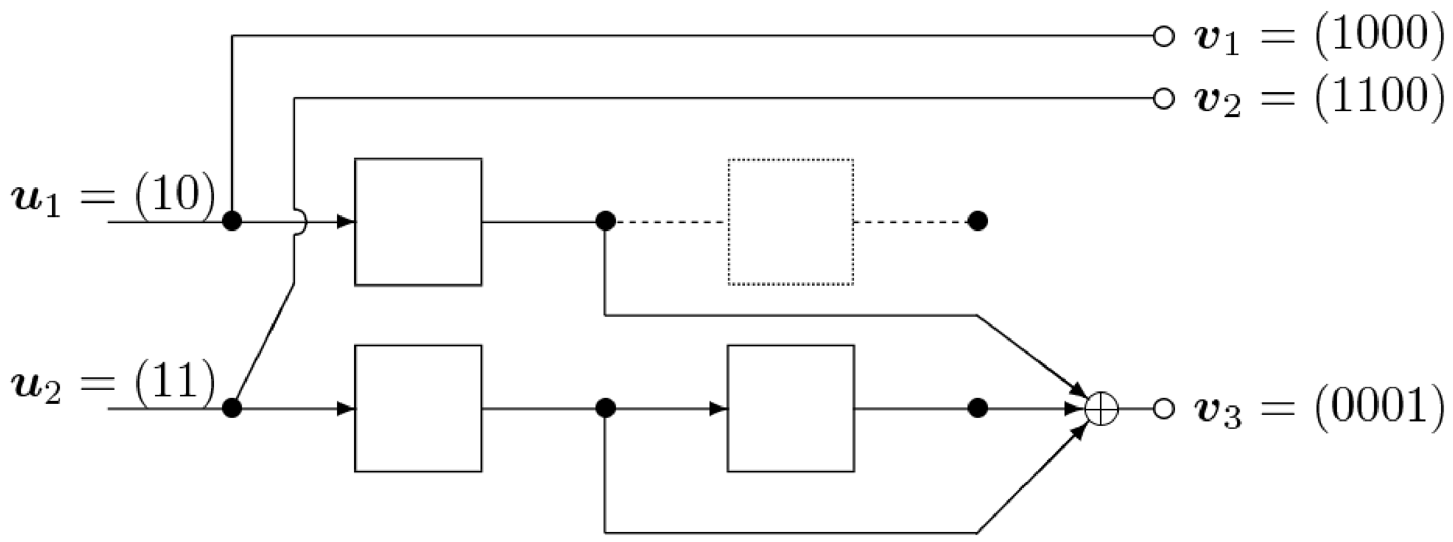
3. $\mathbf{g}_i^{(j)}$ can be obtained by stimulating the discrete impulse $(1, 0, 0, \cdots)$ at the ith input and observing the jth output, when all other inputs are fed the zero sequence $(0, 0, 0, \cdots)$.

4. The impulse responses are also called the *generator sequences* of the encoder.

★ Example 3: Impulse response for a binary (2,1,2) code



★ Example 4: Impulse response for a binary (3, 2, 2) convolutional code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001$

Generator Matrix in the Time Domain

1. The convolutional codes can be generated by a generator matrix multiplied by the information sequences.

2. Let $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k$ are the information sequences and $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n$ the output sequences.

3. Arrange the information sequences as

$$\begin{aligned}\mathbf{u} &= (u_{1,0}, u_{2,0}, \cdots, u_{k,0}, u_{1,1}, u_{2,1}, \cdots, u_{k,1}, \cdots, u_{1,l}, u_{2,l}, \cdots, u_{k,l}, \cdots) \\ &= (\mathbf{w}_0, \mathbf{w}_1, \cdots, \mathbf{w}_l, \cdots),\end{aligned}$$

4. Arrange the output sequences as

$$\begin{aligned}\mathbf{v} &= (v_{1,0}, v_{2,0}, \cdots, v_{n,0}, v_{1,1}, v_{2,1}, \cdots, v_{n,1}, \cdots, v_{1,l}, v_{2,l}, \cdots, v_{n,l}, \cdots) \\ &= (\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_l, \cdots),\end{aligned}$$

5. The relation between $\mathbf{v}$ and $\mathbf{u}$ can characterized as

$$\mathbf{v} = \mathbf{u} \cdot \mathbf{G},$$

where $\mathbf{G}$ is the generator matrix of the code

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_m & & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{m-1} & \mathbf{G}_m & & \\ & & \mathbf{G}_0 & \cdots & \mathbf{G}_{m-2} & \mathbf{G}_{m-1} & \mathbf{G}_m & \\ & & & & \ddots & & & \ddots \end{bmatrix}.$$

with the $k \times n$ submatrices

$$\mathbf{G}_l = \begin{bmatrix} g_{1,l}^{(1)} & g_{1,l}^{(2)} & \cdots & g_{1,l}^{(n)} \\ g_{2,l}^{(1)} & g_{2,l}^{(2)} & \cdots & g_{2,l}^{(n)} \\ \vdots & \vdots & \cdots & \vdots \\ g_{k,l}^{(1)} & g_{k,l}^{(2)} & \cdots & g_{k,l}^{(n)} \end{bmatrix}.$$

6. The element $g_{i,l}^{(j)}$, for $i \in [1, k]$ and $j \in [1, n]$, are the impulse response of the

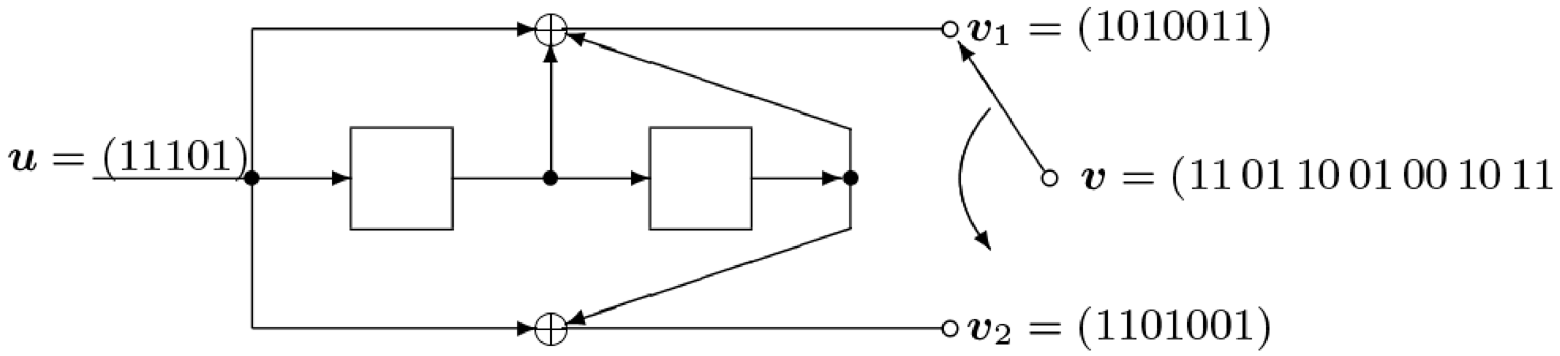
ith input with respect to jth output:

$$\mathbf{g}_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \cdots, g_{i,l}^{(j)}, \cdots, g_{i,m}^{(j)})$$

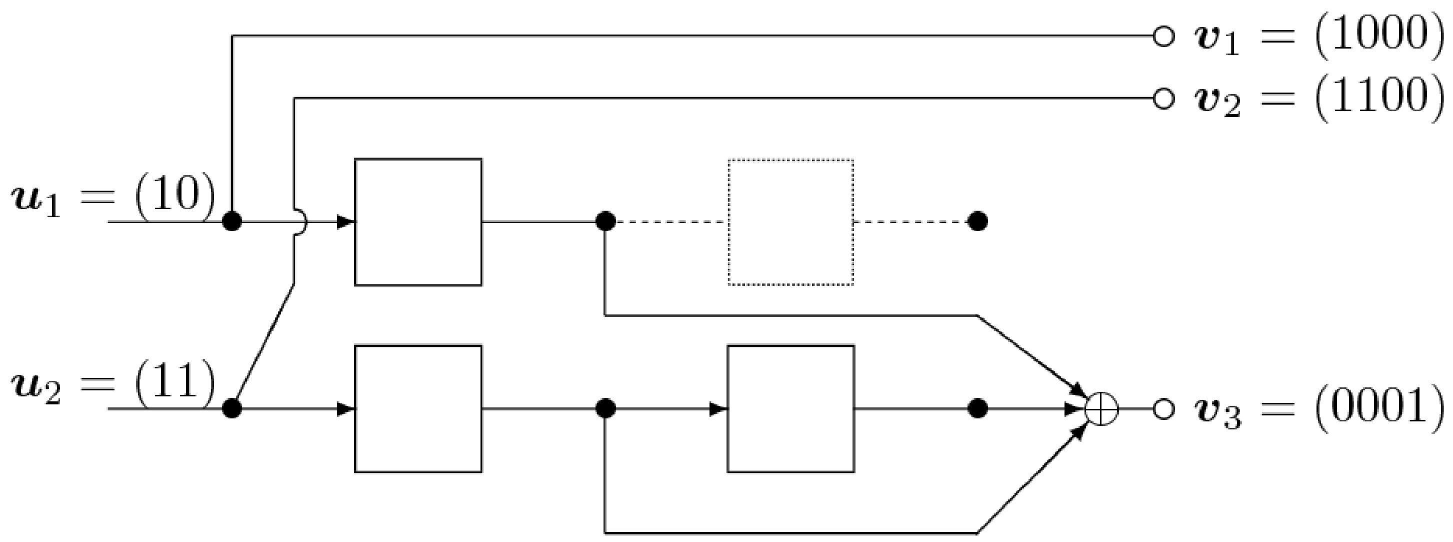
7. Since the code word $\mathbf{v}$ is a linear combination of rows of the generator matrix $\mathbf{G}$, an (n, k, m) convolutional code is a linear code.

8. For an kL finite length information sequence, the corresponding code word has length $n(L + m)$, where the final nm outputs are generated after the last nonzero information block has entered the encoder.

★ Example 5: Generator matrix of a binary (2,1,2) code



★ Example 6: Generator matrix of a binary (3, 2, 2) convolutional code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001$

Generator Matrix in the Z Domain

1. In a linear system, time-domain operations involving convolution can be replaced by more convenient transform-domain operations involving polynomial multiplication.

2. Since a convolutional encoder is a linear system, each sequence in the encoding equations can be replaced by corresponding polynomial, and the convolution operation replaced by polynomial multiplication.

3. According to the Z transform,

$$\mathbf{u}_i \Rightarrow u_i(D) = \sum_{t=0}^{\infty} u_{i,t} D^t$$

$$\mathbf{v}_j \Rightarrow v_j(D) = \sum_{t=0}^{\infty} v_{j,t} D^t$$

$$\mathbf{g}_i^{(j)} \Rightarrow g_i^{(j)}(D) = \sum_{t=0}^{\infty} g_{i,t}^{(j)} D^t$$

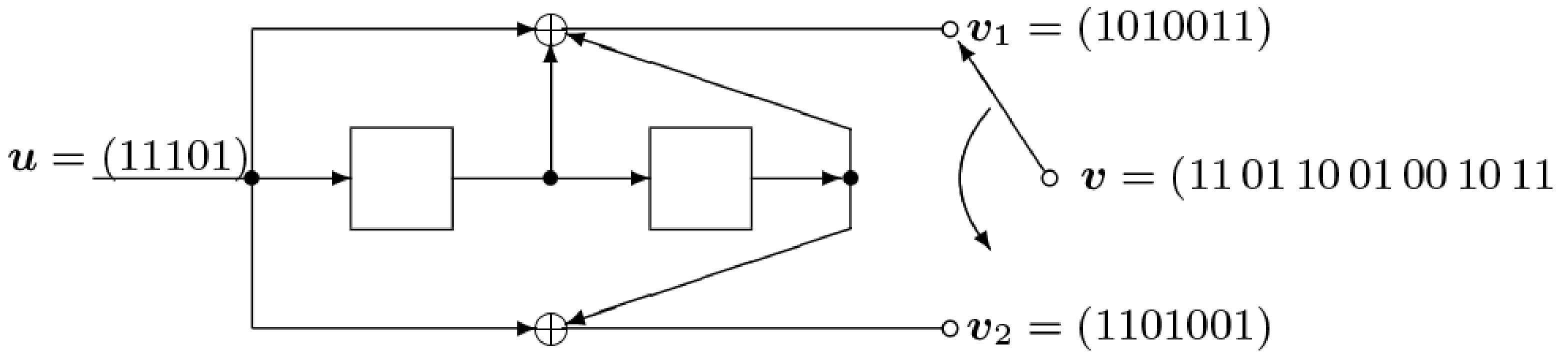
4. The convolutional relation of the Z transform $Z\{u * g\} = u(D)g(D)$ is used to transform the convolution of input sequences and generator sequences to a multiplication in the Z domain.
5. $v_j(D) = \sum_{i=1}^{k} u_i(D) g_i^{(j)}(D)$
6. $g_i^{(j)}(D)$ are called *generator polynomials*.
7. The indeterminate D can be interpreted as a delay operator, and the power of D denoting the number of time units a bit is delayed with respect to the initial bit.
8. We can write the above equations into a matrix multiplication:

$$\mathbf{V}(D) = \mathbf{U}(D) \cdot \mathbf{G}(D)$$

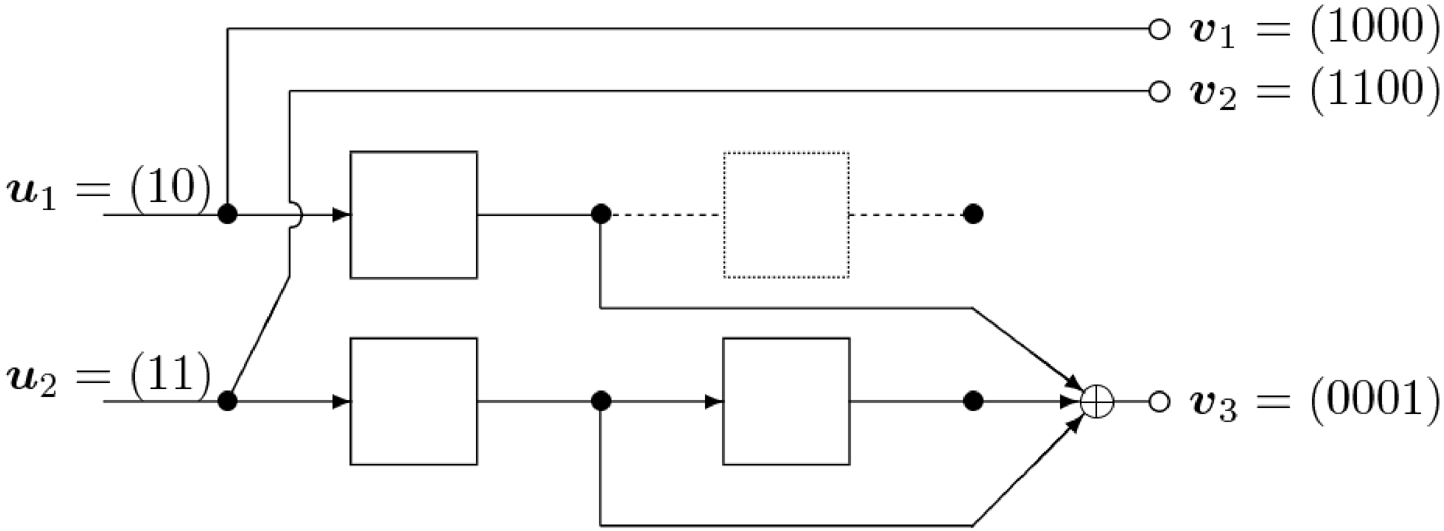
where

$$\mathbf{U}(D) = (u_1(D), u_2(D), \cdots, u_k(D))$$
$$\mathbf{V}(D) = (v_1(D), v_2(D), \cdots, v_n(D))$$
$$\mathbf{G}(D) = \left[g_i^{(j)}(D) \right]$$

★ Example 7: Generator matrix of a binary (2,1,2) code



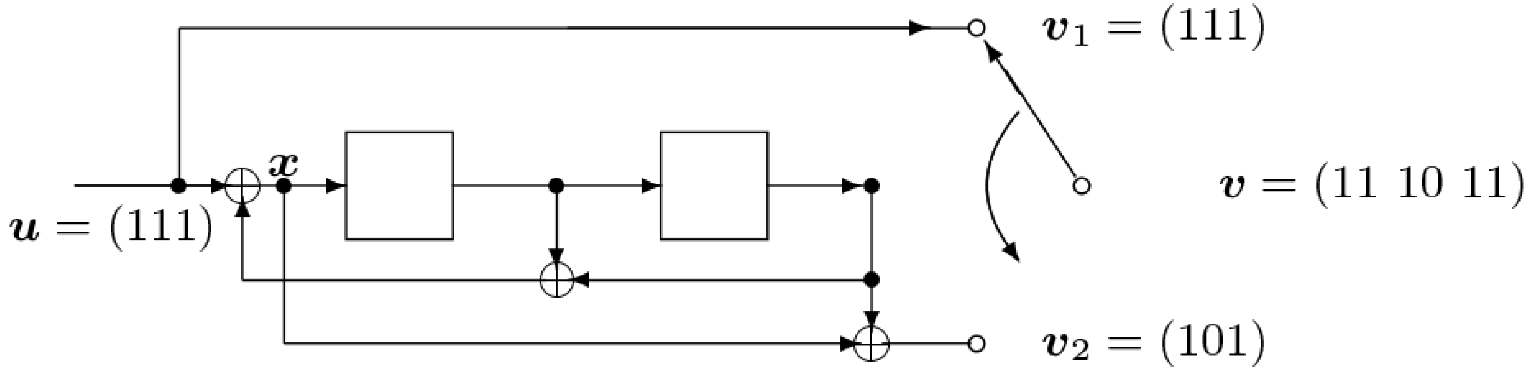
★ Example 8: Generator matrix of a binary (3, 2, 2) convolutional code



$\boldsymbol{u} = (11\ 01)$

$\boldsymbol{v} = (110\ 010\ 000\ 001$

FIR and IIR systems



1. All examples in previous slides are finite impulse response (FIR) systems that are with finite impulse responses.
2. The above example is an infinite impulse response (IIR) system that is with infinite impulse response.
3. The generator sequences of the above example are

$$g_1^{(1)} = (1)$$
$$g_1^{(2)} = (1, 1, 1, 0, 1, 1, 0, 1, 1, 0, \cdots)$$

4. The infinite sequence of $g_1^{(2)}$ is caused by the recursive structure of the encoder.

5. By introducing the variable $\mathbf{x}$, we have

$$
\begin{aligned}
x_t &= u_t + x_{t-1} + x_{t-2} \\
v_{2,t} &= x_t + x_{t-2}
\end{aligned}
$$

Accordingly, we can have the following difference equations:

$$
\begin{aligned}
v_{1,t} &= u_t \\
v_{2,t} + v_{2,t-1} + v_{2,t-2} &= u_t + u_{t-2}
\end{aligned}
$$

6. We then apply the z transform to the second equation:

$$
\sum_{t=0}^{\infty} v_{2,t} D^t + \sum_{t=0}^{\infty} v_{2,t-1} D^t + \sum_{t=0}^{\infty} v_{2,t-2} D^t = \sum_{t=0}^{\infty} u_t D^t + \sum_{t=0}^{\infty} u_{t-2} D^t,
$$

$$
v_2(D) + D v_2(D) + D^2 v_2(D) = u(D) + D^2 u(D).
$$

$\Rightarrow$

$$v_2(D) = \frac{1+D^2}{1+D+D^2}u(D) = g_{12}(D)u(D).$$

7. The generator matrix is obtained:

$$\mathbf{G}(D) = \begin{pmatrix} 1 & \frac{1+D^2}{1+D+D^2} \end{pmatrix}.$$

Polynomial/Systematic Form of the Generator Matrix

Example 4.2. Consider the generator matrix for a rate-2/3 convolutional code given by

$$\mathbf{G}(D) = \begin{bmatrix} \dfrac{1+D}{1+D+D^2} & 0 & 1+D \\ 1 & \dfrac{1}{1+D} & \dfrac{D}{1+D^2} \end{bmatrix}.$$

If we (a) multiply the first row by $(1+D+D^2)/(1+D)$, (b) add the new first row to the second row, and (c) multiply the new second row by $1+D$, with all operations over $\mathbb{F}_2(D)$, the result is the systematic form

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & 0 & 1+D+D^2 \\ 0 & 1 & \dfrac{1+D^3+D^4}{1+D} \end{bmatrix}.$$

Noting that the submatrix $\mathbf{P}(D)$ is the rightmost column of $\mathbf{G}_{\text{sys}}(D)$, we may immediately write

$$\mathbf{H}_{\text{sys}}(D) = \begin{bmatrix} 1+D+D^2 & \dfrac{1+D^3+D^4}{1+D} & 1 \end{bmatrix}$$

because we know that $\mathbf{H}_{\text{sys}}(D) = [\mathbf{P}^{\text{T}}(D)|\mathbf{I}]$. The least common multiple of the denominators of the entries of $\mathbf{G}(D)$ is $(1 + D + D^2)(1 + D^2)$, from which

$$\mathbf{G}_{\text{poly}}(D) = \begin{bmatrix} (1+D)^3 & 0 & (1+D+D^2)(1+D)^3 \\ (1+D+D^2)(1+D^2) & (1+D+D^2)(1+D) & D(1+D+D^2) \end{bmatrix}.$$

Lastly, by multiplying each of the entries of $\mathbf{H}_{\text{sys}}(D)$ by $1 + D$, we obtain

$$\mathbf{H}_{\text{poly}}(D) = \begin{bmatrix} 1+D^3 & 1+D^3+D^4 & 1+D \end{bmatrix}.$$

★ $\mathbf{G}(D)$, $\mathbf{G}_{\text{sys}}(D)$, and $\mathbf{G}(D)_{\text{poly}}$ all generate the same code.

Encoder Realizations and Classifications

1. When $g_{i,j}(D)$ is a rational function, that is, $g_i^j(D) = \frac{a(D)}{b(D)}$, we assume that it has the form
$$g_i^{(j)}(D) = \frac{a_0 + a_1 D + \cdots + a_m D^m}{1 + b_1 D + \cdots + b_m D^m}.$$

2. It implies that
$$v_j(D) = u_i(D)\frac{a(D)}{b(D)}.$$
$$\Rightarrow b_0 v_{j,t} = a_0 u_{i,t} + a_1 u_{i,t-1} + \cdots + a_m u_{i,t-m}$$
$$- b_1 v_{j,t-1} - b_2 v_{j,t-2} - \cdots - b_m v_{j,t-m}$$

3. Fig 4.2 depics the *Type I IIR realization* of the transfer function $g_j^{(j)}(D) = \frac{a(D)}{b(D)}$, while Fig 4.3 depics the *Type II IIR realization*.

4. The FIR case, i.e., without feedback, is the special case in Fig. 4.2 and 4.3 with $b_2 = b_3 = \cdots = b_m = 0$.

5. The Type I form is also called the *controller canonical form* or the *direct canonical form*.
6. The Type II form is also called the *observer canonical form* or the *transposed canonical form*.

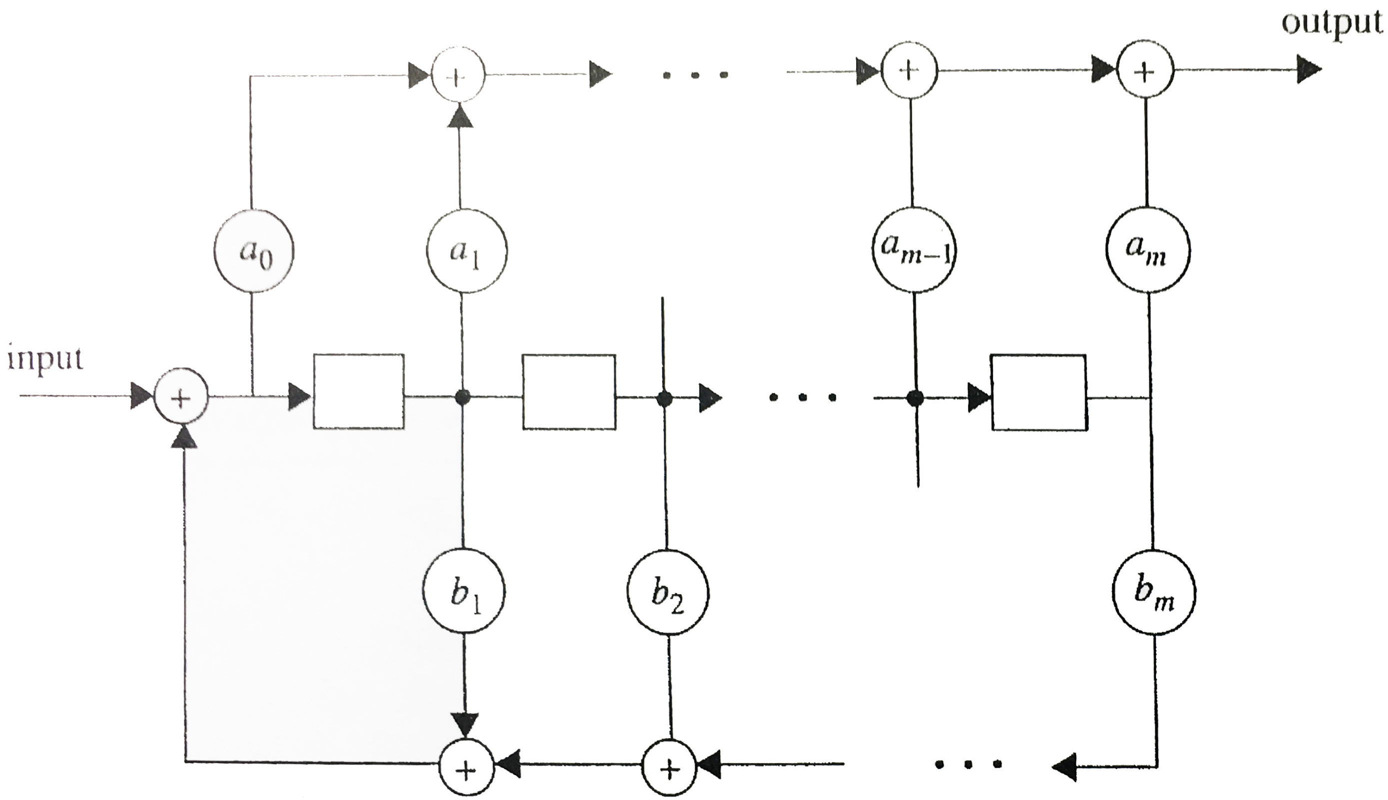


Figure 4.2 Type I realization of the transfer function $g_i^{(j)}(D) = a(D)/b(D)$, with $b_0 = 1$. The input is $u^{(i)}(D)$ and the output is $c^{(j)}(D)$.

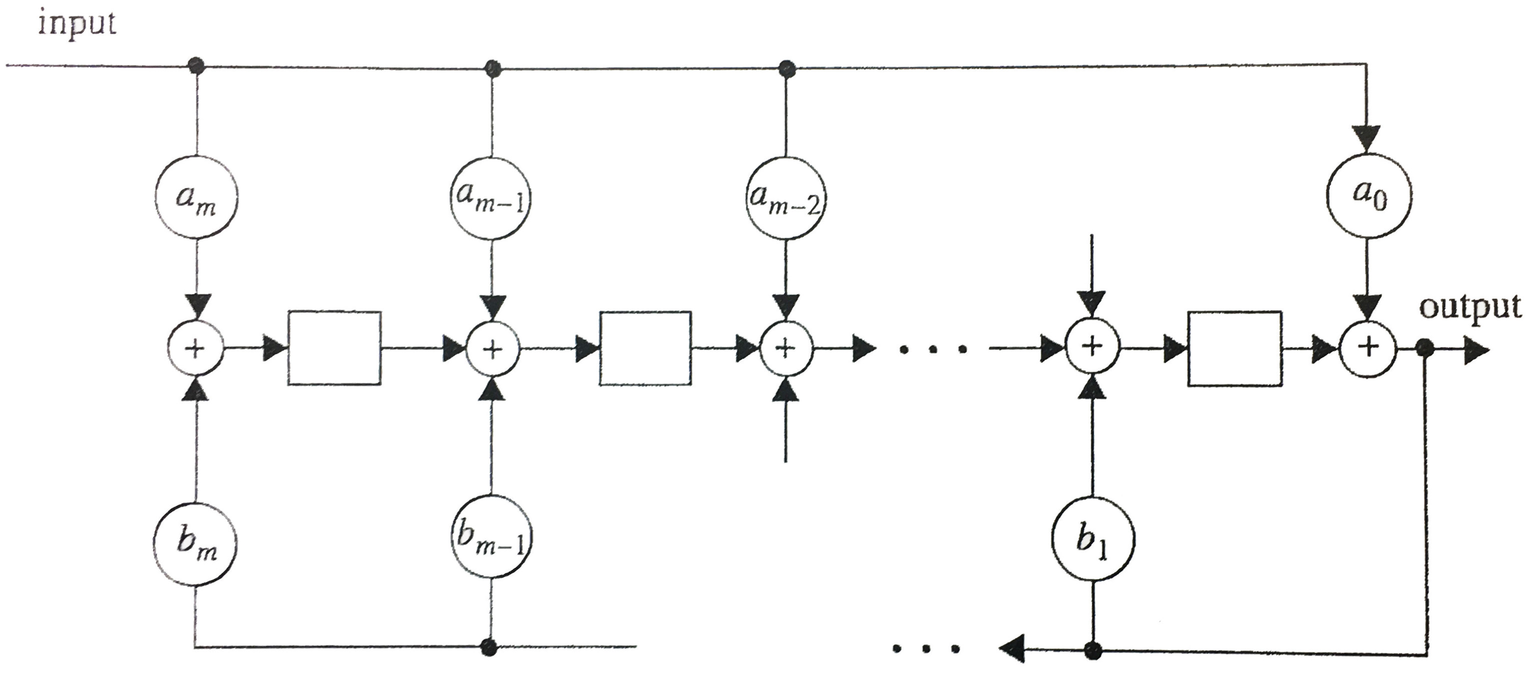


Figure 4.3 Type II realization of the transfer function $g_i^{(j)}(D) = a(D)/b(D)$, with $b_0 = 1$. The input is $u^{(i)}(D)$ and the output is $c^{(j)}(D)$.

Example 4.3. The Type I encoder realization of the rate-1/2 convolutional code with generator matrix $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$, originally depicted in Figure 4.1, is presented in Figure 4.4(a). The Type I encoder for the systematic form of this code, for which the generator matrix is

$$\mathbf{G}_{\mathsf{sys}}(D) = \begin{bmatrix} 1 & \dfrac{1 + D^2}{1 + D + D^2} \end{bmatrix},$$

is presented in Figure 4.4(b). The Type II encoder for $\mathbf{G}_{\mathsf{sys}}(D)$ is presented in Figure 4.4(c).

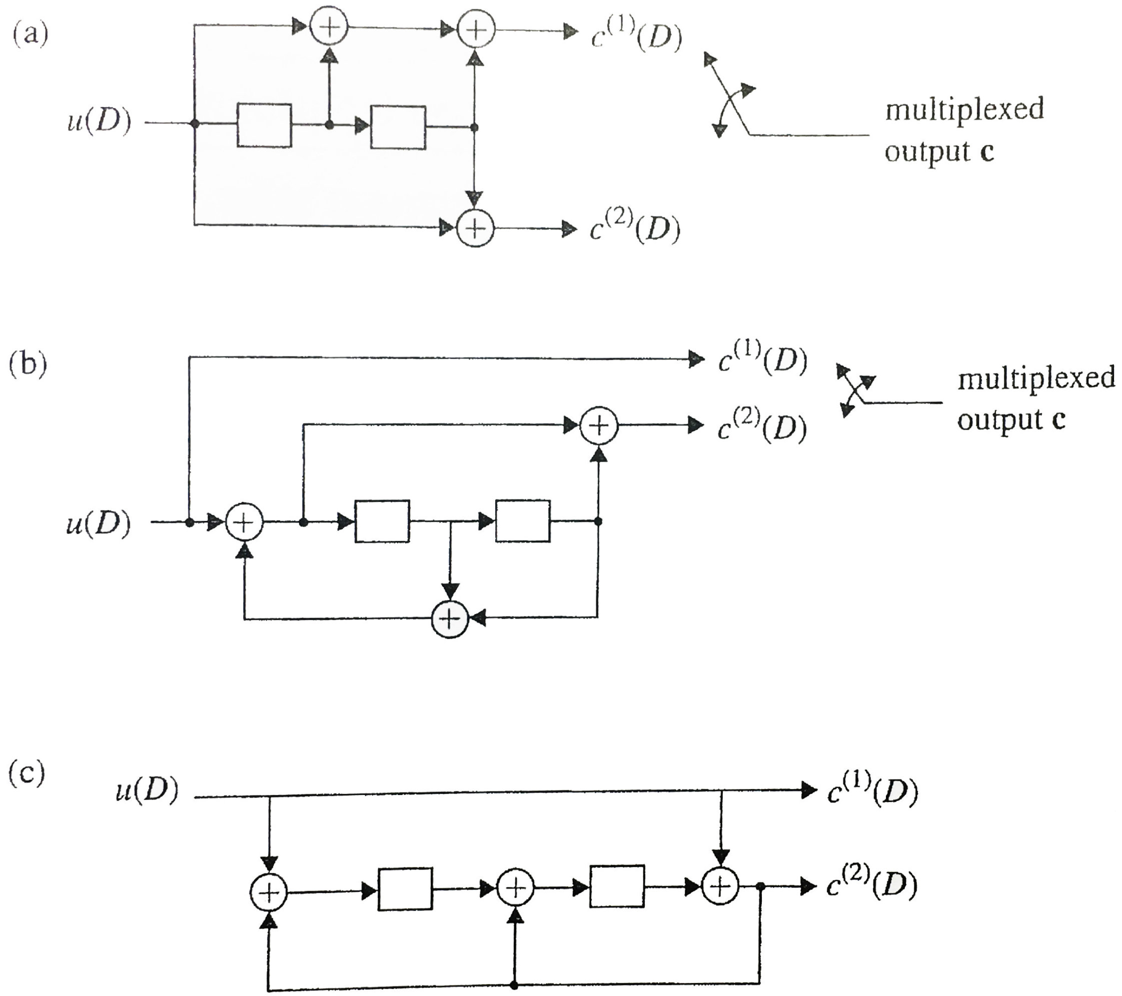


Figure 4.4 Encoder realizations of the non-systematic and systematic versions of the rate-$1/2$ convolutional code given in Example 4.3 with generator matrix $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$. (a) Type I realization of a non-systematic encoder. (b) Type I realization of a systematic encoder. (c) Type II realization of a systematic encoder.

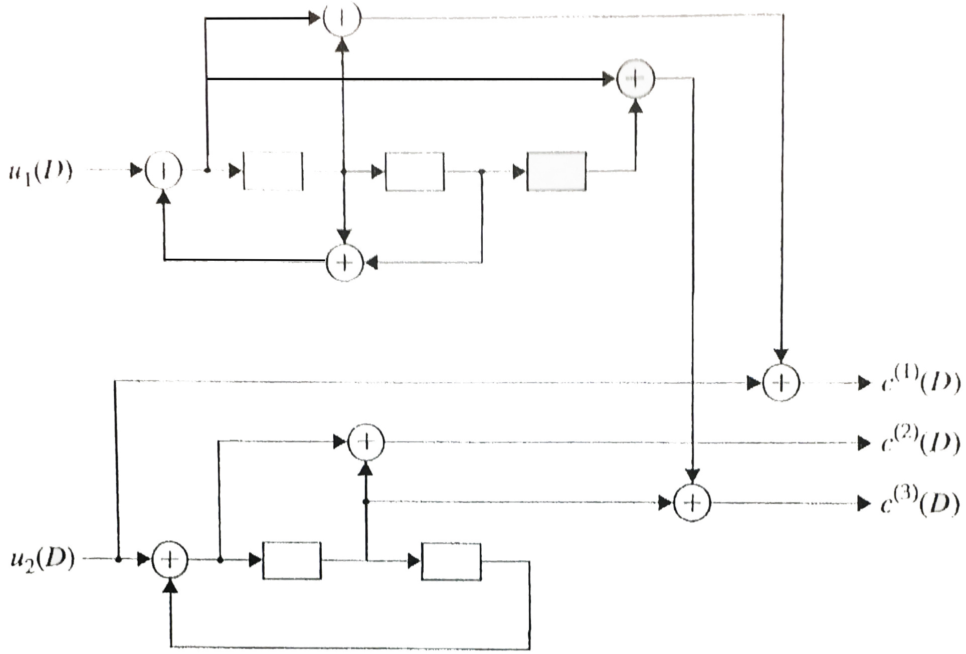


Figure 4.5 Type I encoder realization of $G(D)$ for the rate-2/3 encoder of the convolutional code given in Example 4.2 and discussed further in Example 4.3.

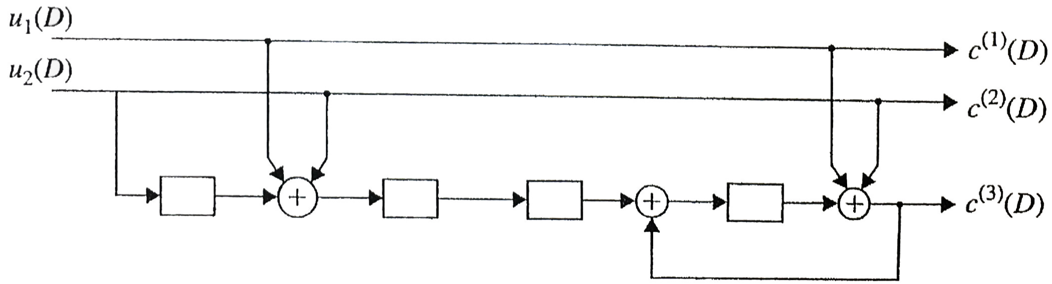


Figure 4.6 Type II encoder realization of $G_{\text{sys}}(D)$ for the rate-2/3 convolutional code given in Examples 4.2 and 4.3.

Four Classes of Convolutional Codes

Table 4.1. Convolutional encoder classes

Convolutional encoder class	Example $\mathbf{G}(D)$
Non-recursive non-systematic ($\bar{\text{R}}\bar{\text{S}}$C)	$\begin{bmatrix}1+D+D^2 & 1+D^2\end{bmatrix}$
Recursive systematic (RSC)	$\begin{bmatrix}1 & \dfrac{1+D^2}{1+D+D^2}\end{bmatrix}$
Recursive non-systematic (R$\bar{\text{S}}$C)	$\begin{bmatrix}\dfrac{1+D}{1+D+D^2} & 0 & 1+D \\ 1 & \dfrac{1}{1+D} & \dfrac{D}{1+D^2}\end{bmatrix}$
Non-recursive systematic ($\bar{\text{R}}$SC)	$\begin{bmatrix}1 & 1+D+D^2\end{bmatrix}$

1. Parallel turbo codes require that the constituent convolutional codes be of the RSC type.

Termination

1. The *effective code rate*, $R_{\text{effective}}$, is defined as the average number of input bits carried by an output bit.
2. In practice, the input sequences are with finite length.
3. In order to terminate a convolutional code, some bits are appended onto the information sequence such that the shift registers return to the zero.
4. Each of the k input sequences of length L bits is padded with m zeros, and these k input sequences jointly induce $n(L+m)$ output bits.
5. The effective rate of the terminated convolutional code is now
$$R_{\text{effective}} = \frac{kL}{n(L+m)} = R\frac{L}{L+m}$$
6. When L is large, $R_{\text{effective}} \approx R$.
7. All examples presented are terminated convolutional codes.

Truncation

1. The second option to terminate a convolutional code is to stop for $t > L$ no matter what contents of shift registers have.
2. The effective code rate is still R.
3. The generator matrix is clipped after the Lth column:

$$\mathbf{G}^c_{[L]} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_m & & & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_m & & & \\ & & \ddots & & & \ddots & & \\ & & & \mathbf{G}_0 & & & & \mathbf{G}_m \\ & & & & & \ddots & & \vdots \\ & & & & & & \mathbf{G}_0 & \mathbf{G}_1 \\ & & & & & & & \mathbf{G}_0 \end{bmatrix}.$$

Tail Biting

1. The drawback of truncation method is that the last few blocks of information sequences are less protected.
2. Tail biting is to start the convolutional encoder in the same contents of all shift registers (state) where it will stop after the input of L information blocks.
3. Equal protection of all information bits of the entire information sequences is possible. The effective rate of the code is still R.
4. The generator matrix has to be clipped after the Lth column, and manipulated

as follows:

$$
\tilde{\mathbf{G}}^{c}_{[L]} = \begin{bmatrix}
\mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_m & & & \\
& \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_m & & \\
& & \ddots & & \ddots & & \\
& & & \mathbf{G}_0 & & & \mathbf{G}_m \\
\mathbf{G}_m & & & & \ddots & & \vdots \\
\vdots & \ddots & & & & \mathbf{G}_0 & \mathbf{G}_1 \\
\mathbf{G}_1 & \cdots & \mathbf{G}_m & & & & \mathbf{G}_0
\end{bmatrix}.
$$

where $\tilde{\mathbf{G}}^{c}_{[L]}$