ScienceDirect

# Convolutional Coding

Related terms:

Channel Coding, Constraint Length, Convolutional Code, Shift Register, Subframes

View all Topics

## Introduction to information theory and coding

Alan Bensky, in Short-range Wireless Communication(Third Edition), 2019

### 9.4.3 Convolutional coding

Convolutional coding is a widely used coding method which is not based on blocks of bits but rather the output code bits are determined by logic operations on the present bit in a stream and a small number of previous bits. In the encoder, data bits are input to a shift register of length *K*, called the constraint length. As each bit enters at the left of the register, the previous bits are shifted to the right while the oldest bit in the register is removed. Two or more binary summing operations, let's say *r*, create code bits which are output during one data flow period. Therefore, the code bit rate is 1/*r* times the data rate and the encoder is called a rate 1/*r* convolutional encoder of constraint length *K*. Also needed to completely define the encoder are the connections from stages in the shift register to the *r* summing blocks. These are generator vectors each of which may be simply expressed as a row of K binary digits. The *r* binary adders create even parity bits at their outputs; that is, connections to an odd number of logic "ones" result in an output of "one," otherwise the output is "zero."

Fig. 9.7 shows an example with $K = 3$, $r = 2$, and the generator vectors are chosen as [1 1 1] and [1 0 1]. Discrete sampling times are labeled *n*. The data stream enters on the left and the present bit at time *n*, the most recent bit $n - 1$ and the next earliest bit at $n - 2$ occupy the shift register. Two parity bits are switched out in the interval between *n* and $n - 1$ from the upper adder and then the lower one. When the next data bit arrives, the shift register moves its contents to the right. The $K - 1$ earlier bits, in this case two, determine the state of the encoder. They are shown in gray in Fig. 9.7. There are $2K - 1$ states. For each encoder state there are two possibilities of output code bits, depending on whether the input bit is “zero” or “one.” The progression of states in time, then, are a function of the data stream. Fig. 9.8 is a *state diagram* of our example. Each state is shown inside a circle and the change from one state to another is shown by an arrow, identified by the input bit, slash, output code bits. You can see that encoding can be done by relatively simple hardware.

this case two, determine the state of the encoder. They are shown in gray in Fig. 9.7. There are $2K_{-1}$ states. For each encoder state there are two possibilities of output code bits, depending on whether the input bit is “zero” or “one.” The progression of states in time, then, are a function of the data stream. Fig. 9.8 is a *state diagram* of our example. Each state is shown inside a circle and the change from one state to another is shown by an arrow, identified by the input bit, slash, output code bits. You can see that encoding can be done by relatively simple hardware.

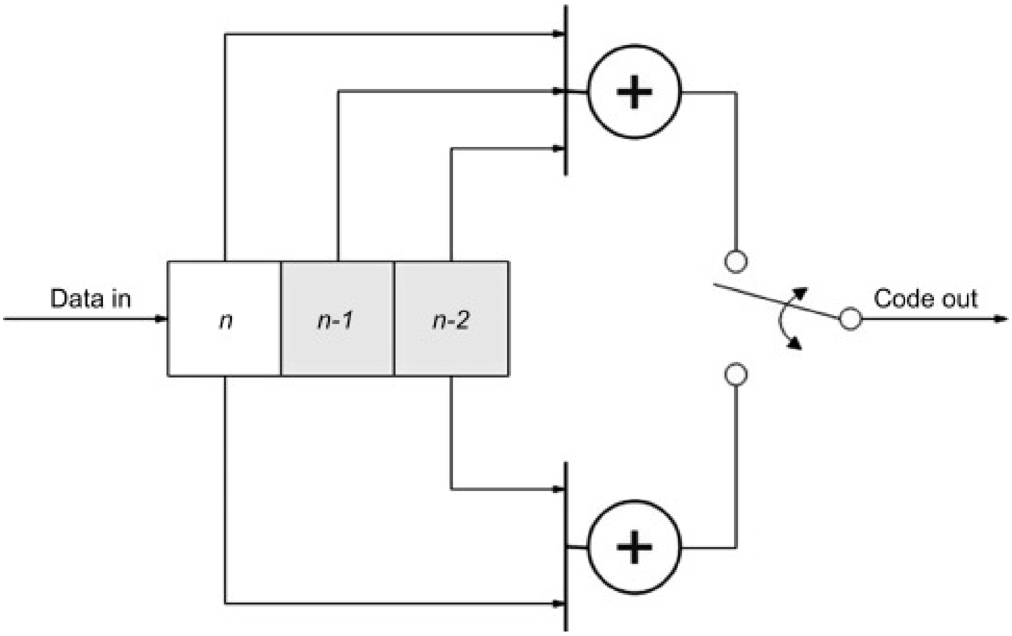


Fig. 9.7. Convolutional encoder.

Fig. 9.7. Convolutional encoder.

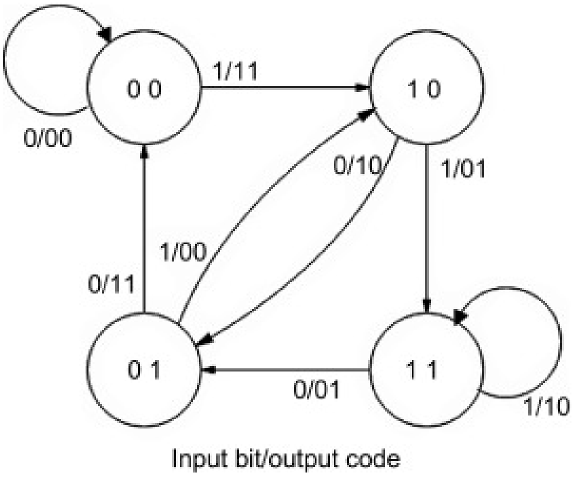
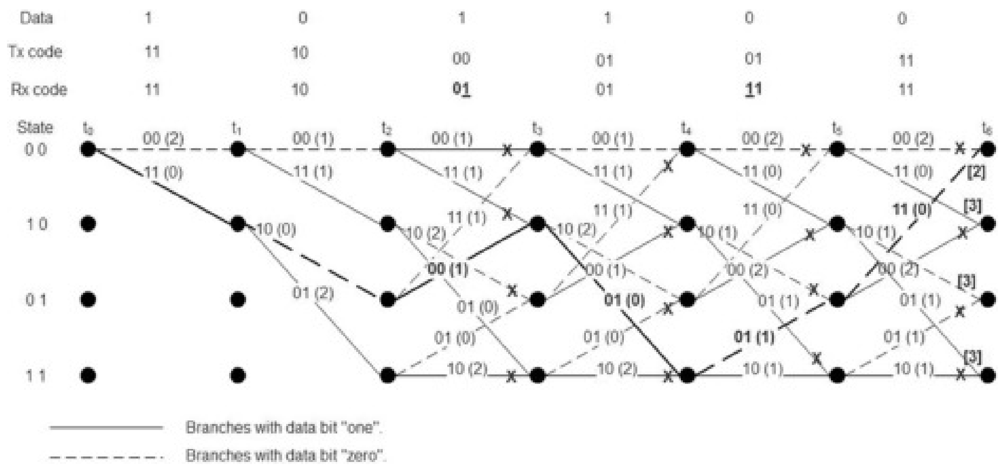


Fig. 9.8. Convolutional encoder state diagram.

Fig. 9.8. Convolutional encoder state diagram.

The decoder estimates the data stream on the basis of the received code bit sequence and knowledge of the encoder state diagram, exemplified by Fig. 9.8. The progression of states in time for all message streams can be shown by a trellis diagram, like Fig. 9.9 which is a continuation of our previous example. We use this diagram to describe the decoding process.

The decoder estimates the data stream on the basis of the received code bit sequence and knowledge of the encoder state diagram, exemplified by Fig. 9.8. The progression of states in time for all message streams can be shown by a trellis diagram, like Fig. 9.9 which is a continuation of our previous example. We use this diagram to describe the decoding process.



Data 1 0 1 1 0 0
Tx code 11 10 00 01 01 11
Rx code 11 10 01 01 11 11
State
0 0
1 0
0 1
1 1
t0
t1
t2
t3
t4
t5
t6
00 (2)
11 (0)
00 (1)
11 (1)
10 (0)
01 (2)
00 (1)
11 (1)
11 (1)
10 (2)
00 (1)
01 (0)
01 (0)
10 (2)
00 (1)
11 (1)
11 (1)
10 (2)
00 (1)
01 (0)
01 (0)
10 (2)
00 (2)
11 (0)
11 (0)
10 (1)
00 (2)
01 (1)
01 (1)
10 (1)
00 (2)
11 (0)
11 (0)
10 (1)
00 (2)
01 (1)
01 (1)
10 (1)
X
[2]
[3]
[3]
[3]
Branches with data bit "one".
Branches with data bit "zero".

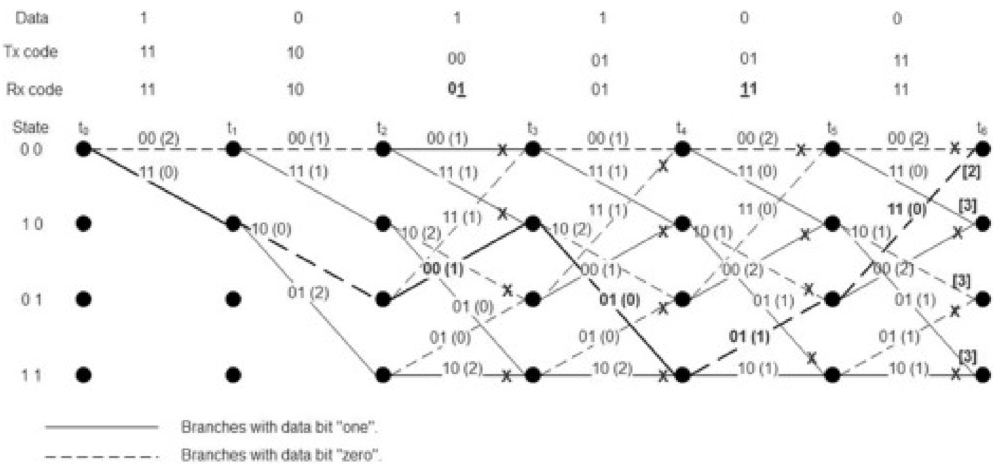

Data
Tx code
Rx code
State
1 0 1 1 0 0
11 10 00 01 01 11
11 10 01 01 11 11
00
10
01
11
$t_0$ $t_1$ $t_2$ $t_3$ $t_4$ $t_5$ $t_6$
Branches with data bit "one".
Branches with data bit "zero".

Fig. 9.9. Convolutional decoder trellis diagram.

Convolutional coding is based on the fact that every possible coded message must traverse through a definite progression of states, and consequently, of $r$-tuple code words, in our case with $r = 2$, bit pairs. Noise on the communication channel may cause some bits to be in error. The trellis diagram shows *all possible transmitted messages*. The number of code bits in the received data stream (message) that differ from any one of the messages in the trellis is called the Hamming distance. The task of the decoder is to find a coded bit sequence in the trellis that has the lowest Hamming distance. That sequence gives the estimated transmitted message.

Fig. 9.9 shows a data stream on the top row and the transmitted and received codes in the two rows below it. Note that the data ends with two consecutive zero's ($K$ – 1 in the general case) which are necessary to insure complete decoding and to have a flushed shift register when the first bit of the next data stream arrives. Each branch in the trellis is labeled with its code bit pair followed in parenthesis with the number of code bits that differ from the corresponding bits in the received code. Solid branch lines are ones and dashed lines are zeros. The transmitted message path which matches the data stream in the first row on the top is shown with bold lines. If there were no errors in the received message, the Hamming distance for this path would be zero. However, due to bit errors at $t_2$ and $t_4$, underlined in the “Rx code” row, its Hamming distance is 2. Does any other path have an equal or shorter Hamming distance? The receiver can check the sum of the bit divergences from the received code for all of the data streams in the trellis and deduce that the true message corresponds to the path with the lowest Hamming distance. The problem is that the number of messages in the trellis increases exponentially with the number of sample times, or length of the data stream. One commonly used solution is *Viterbi decoding*. It eases the computing burden by deleting one of the two paths that enter each state node, thereby preventing the doubling of messages at each sample time. The selection criterion is to choose as the remaining path the one with the lowest accumulated code differences with the received code. Fig. 9.9 shows an **X** on the deleted path branches. The sum of the code divergences, the Hamming distance, of each of the four remaining paths is shown in brackets at the $t_6$ nodes. The bold line path has the lowest Hamming distance, so the message was decoded successfully.

an X on the deleted path branches. The sum of the code divergences, the Hamming distance, of each of the four remaining paths is shown in brackets at the $t_6$ nodes. The bold line path has the lowest Hamming distance, so the message was decoded successfully.

The maximum number of errors that can be corrected is a function of the constraint length and the selected generator vectors. A path length of at least several constraint lengths is necessary to realize that number. For our example with $K = 3$, two errors over a length of around 20 code bits should be correctable, but that also depends on how the errors are distributed [5]. In this example, the coding worked with a shorter message.

The maximum number of errors that can be corrected is a function of the constraint length and the selected generator vectors. A path length of at least several constraint lengths is necessary to realize that number. For our example with $K = 3$, two errors over a length of around 20 code bits should be correctable, but that also depends on how the errors are distributed [5]. In this example, the coding worked with a shorter message.

The above explanation of convolution decoding dealt with hard decisions, that is, code differences were between the discrete logic signal levels. However, the receiver detector may output multiple level digital words for each symbol that reflect a degree of confidence whether the bit is a “zero” or “one.” In this case the Hamming distance can't be used and a Euclidean distance between a noisy point (the received symbol) and the trellis point has to be calculated. This soft-decision Viterbi decoding is otherwise the same as the hard-decision method we described and should give better results since more information about the signal is available.

The above explanation of convolution decoding dealt with hard decisions, that is, code differences were between the discrete logic signal levels. However, the receiver detector may output multiple level digital words for each symbol that reflect a degree of confidence whether the bit is a “zero” or “one.” In this case the Hamming distance can't be used and a Euclidean distance between a noisy point (the received symbol) and the trellis point has to be calculated. This soft-decision Viterbi decoding is otherwise the same as the hard-decision method we described and should give better results since more information about the signal is available.

As we have seen, the basic code rate of a convolutional encoder with $r = 2$ is ½ and it gives good error correcting performance in many situations, particularly when there is a high signal-to-noise ratio. A higher data rate is desirable even at the expense of error correction. To increase the code rate, *puncturing* is used which is a procedure for omitting some of the code bits to get a higher data rate while maintaining proportionally less error correction capability. In the decoder, dummy bits are inserted in place of the omitted bits and the Viterbi algorithm is carried out as described. For example, OFDM IEEE 802.11a $K = 7$ convolutional encoder can produce code rates of ½ (basic), 2/3 or ¾, plus 5/6 for the high throughput and very high throughput physical layers (see Chapter 11).

As we have seen, the basic code rate of a convolutional encoder with $r = 2$ is ½ and it gives good error correcting performance in many situations, particularly when there is a high signal-to-noise ratio. A higher data rate is desirable even at the expense of error correction. To increase the code rate, *puncturing* is used which is a procedure for omitting some of the code bits to get a higher data rate while maintaining proportionally less error correction capability. In the decoder, dummy bits are inserted in place of the omitted bits and the Viterbi algorithm is carried out as described. For example, OFDM IEEE 802.11a $K = 7$ convolutional encoder can produce code rates of ½ (basic), 2/3 or ¾, plus 5/6 for the high throughput and very high throughput physical layers (see Chapter 11).

> Read full chapter

> Read full chapter

# VLSI Signal Processing

VLSI Signal Processing

Surin Kittitornkun, Yu-Hen Hu, in The Electrical Engineering Handbook, 2005

Surin Kittitornkun, Yu-Hen Hu, in The Electrical Engineering Handbook, 2005

## Viterbi Algorithm

Viterbi Algorithm

The convolutional coding has been one of the most widely used error corrections in digital wireless communication. Therefore, the **Viterbi decoding algorithm** must be implemented efficiently in a pipelined/systolic fashion. *A* (*K*, *R*) convolutional coding scheme can be described by an FSM, where *K* is the constraint length and *R* is the code rate. Similar to a mealy FSM, there are a total of 2*K* coding states where output bits and the next state depend on the current state and an input bit.

linear shift register relationships. The first register connection to the XOR gate is indicated by the “1” in the equations, the second by “X,” the third by $X_2$ and so forth. Most convolution codes have a constraint length less than 10.

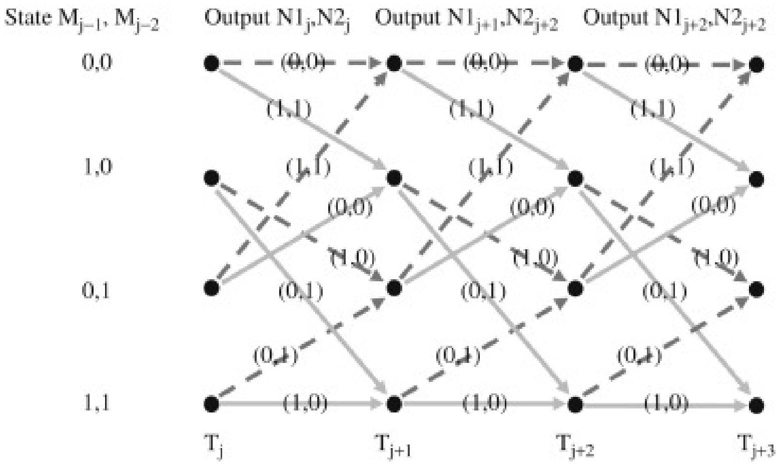
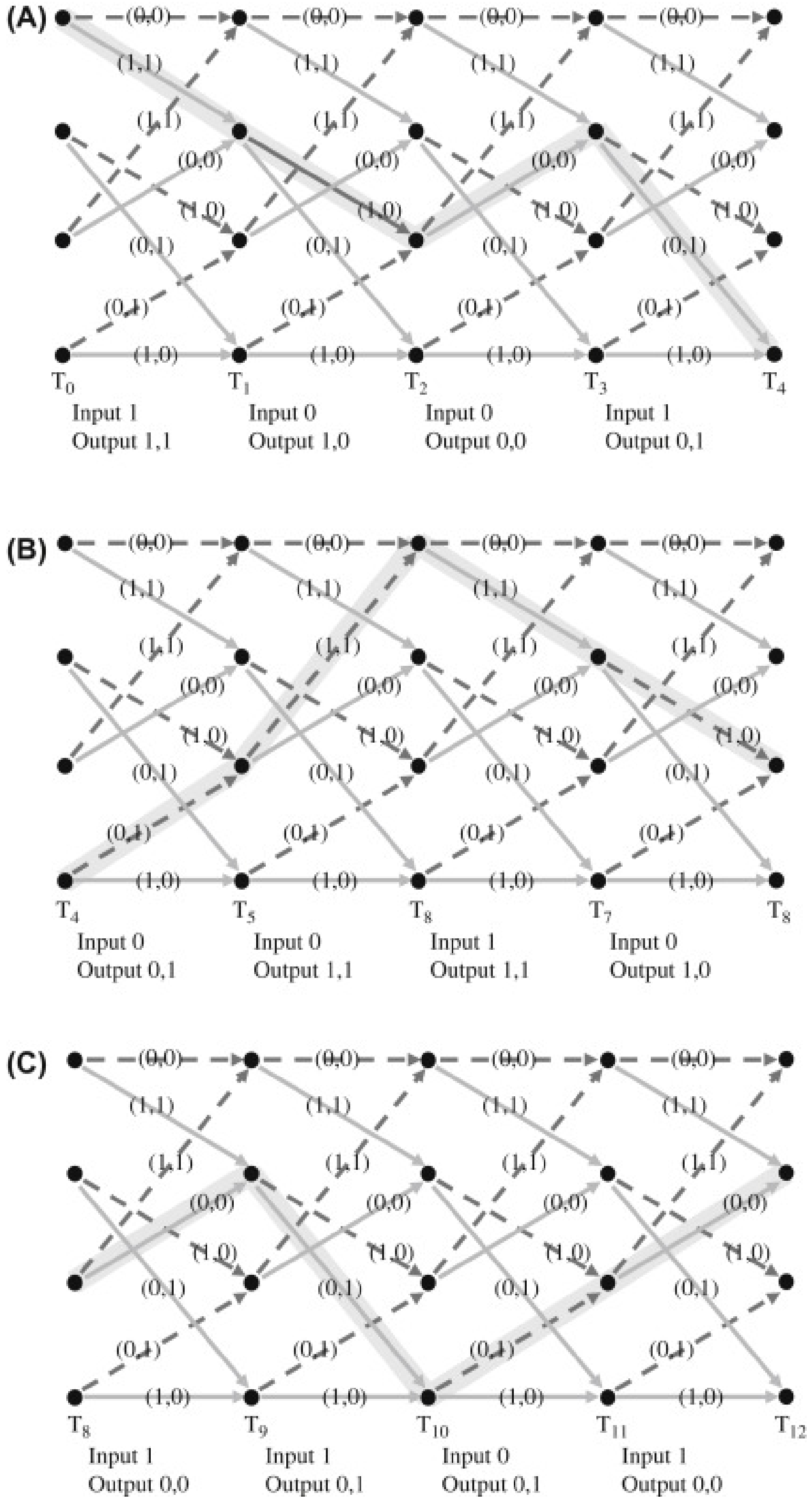


Figure 12.2. Trellis diagram.

Figure 12.2. Trellis diagram.

Now, let us trace the path of the input sequence through the trellis using Fig. 12.3, and the resulting output sequence. This will help us gain the insight that the trellis is representative of the encoder circuit, as use of the trellis will be a key in Viterbi decoding. The highlighted lines show the path of input sequence. The trellis diagram covers from states 0 through 12. It is broken into three sections (A, B, C) in order to fit on the page.

Now, let us trace the path of the input sequence through the trellis using Fig. 12.3, and the resulting output sequence. This will help us gain the insight that the trellis is representative of the encoder circuit, as use of the trellis will be a key in Viterbi decoding. The highlighted lines show the path of input sequence. The trellis diagram covers from states 0 through 12. It is broken into three sections (A, B, C) in order to fit on the page.

(A)
(0,0)
(1,1)
(1,1)
(0,0)
(1,0)
(0,1)
(0,1)
(1,0)
T$_0$
Input 1
Output 1,1
T$_1$
Input 0
Output 1,0
T$_2$
Input 0
Output 0,0
T$_3$
Input 1
Output 0,1
T$_4$
(B)
T$_4$
Input 0
Output 0,1
T$_5$
Input 0
Output 1,1
T$_8$
Input 1
Output 1,1
T$_7$
Input 0
Output 1,0
T$_8$
(C)
T$_8$
Input 1
Output 0,0
T$_9$
Input 1
Output 0,1
T$_{10}$
Input 0
Output 0,1
T$_{11}$
Input 1
Output 0,0
T$_{12}$

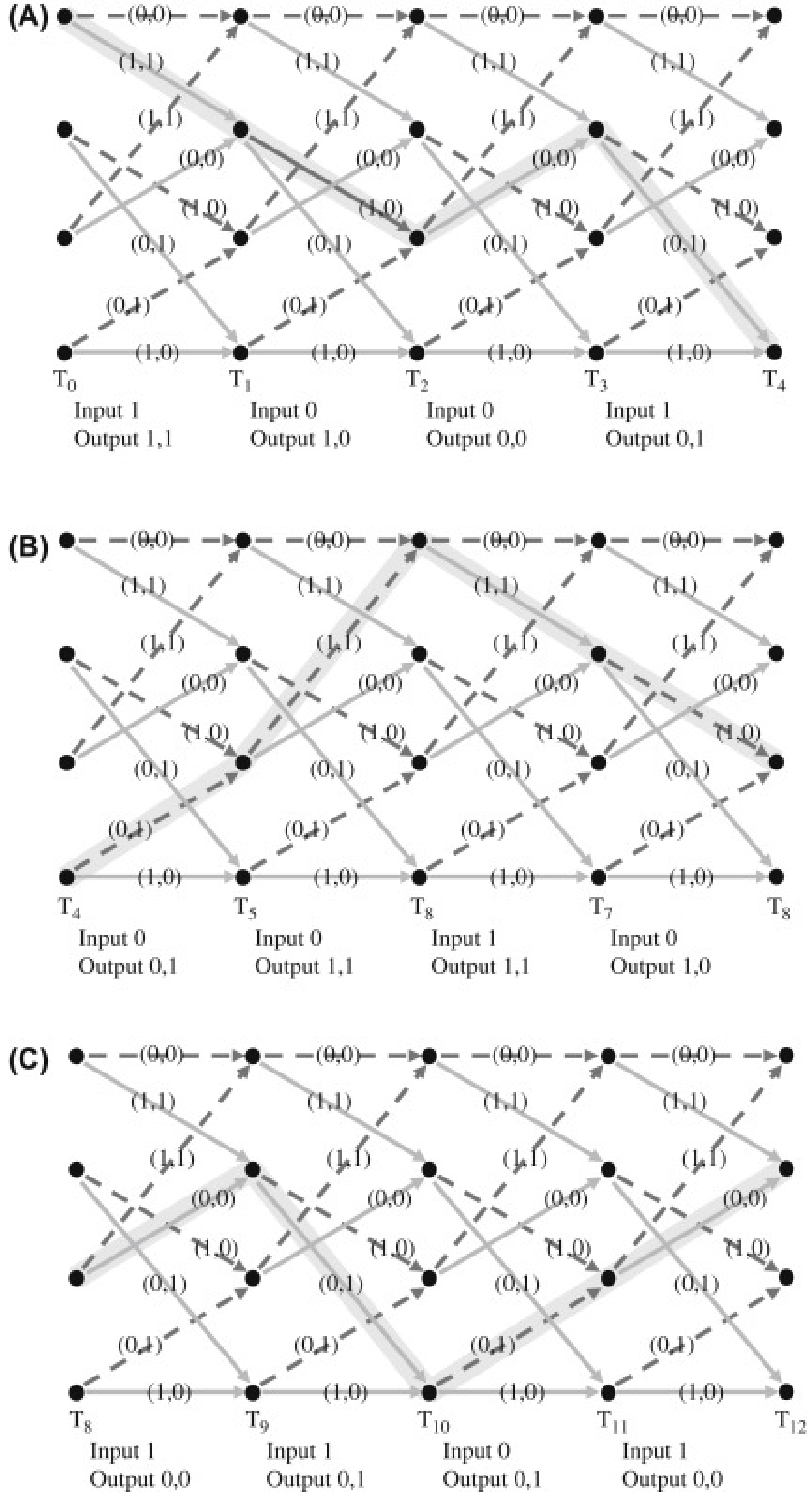


Figure 12.3. Sequence of data through trellis.

Figure 12.3. Sequence of data through trellis.

By tracing the highlighted path through the trellis, you can see that the output sequence is the same as our results when computing using the shift register circuit. For constraint length K, we will have (K–1)2 states in our trellis diagram. Therefore, with K = 3 in our design example, we have four possible states. For a more typical K = 6 or K = 7 constraint length, there would be 32 or 64 states respectively; however, this is too tedious to try to diagram.

By tracing the highlighted path through the trellis, you can see that the output sequence is the same as our results when computing using the shift register circuit. For constraint length K, we will have (K–1)2 states in our trellis diagram. Therefore, with K = 3 in our design example, we have four possible states. For a more typical K = 6 or K = 7 constraint length, there would be 32 or 64 states respectively; however, this is too tedious to try to diagram.

> Read full chapter

> Read full chapter

## Coding

DrM D Macleod MA PhD MIEEE, in Telecommunications Engineer's Reference Book, 1993

### 14.2.1 Types of ECC

There are two main types of ECC: block coding and convolutional coding. In block coding, the input is divided into blocks of k digits. The coder then produces a block of n digits for transmission, and the code is described as an '(n,k) code'. Each block is coded and decoded entirely separately from all other blocks. In convolutional coding, the coder input and output are continuous streams of digits. The coder outputs n output digits for every k digits input, and the code is described as 'a rate k/n code'.

If the input digits are included unmodified in the coder output the code is described as systematic. The additional digits introduced by the coder are then known as parity or check digits. As well as the conceptual attractiveness of systematic codes, they have the advantage that a range of decoder complexities is made possible. The simplest decoder can simply extract the unmodified input digits from the coded digit stream, ignoring the parity digits. A more sophisticated decoder may use the parity digits for error detection, and a full decoder for error correction. Unsystematic codes also exist, but are less commonly used.

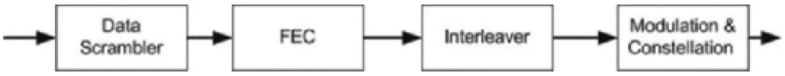
> Read full chapter

## Cognitive radio for broadband wireless access in TV bands: The IEEE 802.22 standards

Carlos Cordeiro, ... Sai Shankar Nandagopalan, in Cognitive Radio Communications and Networks, 2010

### 14.3.3 Channel Coding and Modulation Schemes

Figure 14.6 describes the channel coding process used in 802.22. Channel coding includes data scrambling, convolutional coding or advanced coding, puncturing, bit interleaving, and constellation mapping.


Data Scrambler
FEC
Interleaver
Modulation & Constellation

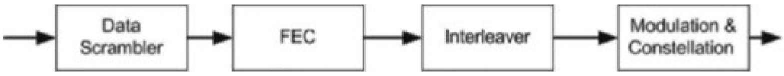


Figure 14.6. Channel coding in 802.22.

The frame payload data are first processed by the data scrambler using a pseudo-random binary sequence generator with the generator polynomial $1 + X^{14} + X^{15}$. The preamble and the control header fields of the frame are not scrambled. The forward error correction (FEC) scheme follows the data scrambler. The mandatory coding scheme in 802.22 is convolutional coding. The data bits are encoded using a rate-1/2 binary convolutional encoder. Duo-binary convolutional turbo code, low-density parity check (LDPC) codes, and shortened block turbo codes (SBTCs) are optional advanced coding schemes. For the interleaving stage, the interleaver used for subcarrier interleaving is employed to interleave encoded bits at the output of the encoder. The interleaving algorithm used in 802.22 is described by the block size *K* and three integer parameters {*p*, *q*, *j*}. The global equation of the algorithm depends on the interleaving pattern of the previous iteration (*j* − 1), the position index of samples (*k*), and two integer parameters (*p*, *q*). The parameter *P* gives the interleaving partition size multiple of the interleaving block size *K*.

Finally, the output bits of the bit interleaver is entered serially into the modulation and constellation mapper. The input data to the mapper are first divided into groups of $N_{cbpc}$ (two for quadrature phase-shift keying (QPSK), four for 16-QAM, and six for 64-QAM) bits and converted into complex numbers representing QPSK, 16-QAM or 64-QAM, constellation points. The mapping for QPSK, 16-QAM, and 64-QAM is performed according to gray-coding constellation mapping. The pilot subcarriers are modulated according to the BPSK modulation using a modulation-dependent normalization factor equal to 1.

> Read full chapter

# Error Correction Coding

Michael Parker, in Digital Signal Processing 101, 2010

## 12.4 Convolutional Encoding

A second major class of channel codes is known as convolutional codes. Convolutional codes can operate on a continuous stream of data, whereas block codes operate on words. Convolutional codes also have memory; the behavior of the code depends on previous data.

Convolutional coding is implemented using shift registers with feedback paths. There is a ratio of “k” input bits to “n” output bits, as well as a constraint length “K.” The code rate is k/n. The constraint length K corresponds to the length of the shift register and also determines the length of time or memory that the current behavior depends on past inputs.

Convolutional coding is implemented using shift registers with feedback paths. There is a ratio of “k” input bits to “n” output bits, as well as a constraint length “K.” The code rate is k/n. The constraint length K corresponds to the length of the shift register and also determines the length of time or memory that the current behavior depends on past inputs.

Next, let's go through a very simple convolutional coding and Viterbi decoding example. We use an example with k = 1, n = 2, and K = 3. The encoder is described by generator equations, using polynomial expressions to describe the linear shift register relationships. The first register connection to the XOR gate is indicated by the 1 in the equations, the second by X, the third by $X_2$, and so forth. Most convolution codes have a constraint length less than 10.

Next, let's go through a very simple convolutional coding and Viterbi decoding example. We use an example with k = 1, n = 2, and K = 3. The encoder is described by generator equations, using polynomial expressions to describe the linear shift register relationships. The first register connection to the XOR gate is indicated by the 1 in the equations, the second by X, the third by $X_2$, and so forth. Most convolution codes have a constraint length less than 10.

The rate is defined as the ratio of input bits to output bits. The rate here is ½ as there are every input bit M, there are two output bits N.

The rate is defined as the ratio of input bits to output bits. The rate here is ½ as there are every input bit M, there are two output bits N.

The output is usually interleaved in a bitstream $N1_j$, $N2_j$, $N1_{j+1}$, $N2_{j+1}$, $N1_{j+3}$, $N2_{j+3}$, … (See Figure 12.1.)

The output is usually interleaved in a bitstream $N1_j$, $N2_j$, $N1_{j+1}$, $N2_{j+1}$, $N1_{j+3}$, $N2_{j+3}$, … (See Figure 12.1.)

Figure 12.1.

Figure 12.1.

The following table shows encoder operation with the input sequence 1,0,1,1,0,0,1,0,1,1,0,0, for each clock cycle. The register is initialized to zero.

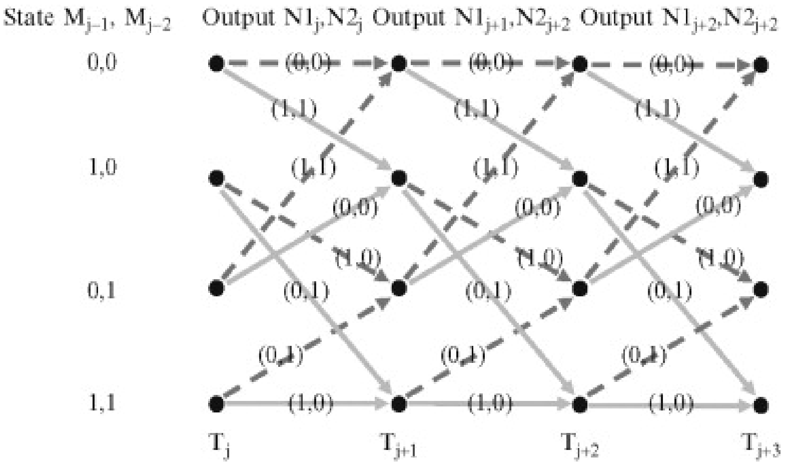
The following table shows encoder operation with the input sequence 1,0,1,1,0,0,1,0,1,1,0,0, for each clock cycle. The register is initialized to zero.

The resulting output sequence is as follows:

The resulting output sequence is as follows:

Notice that the output $N1_j$ and $N2_j$ are both a function of the input bit $M_j$ and the two previous input bits $M_{j-1}$ and $M_{j-2}$. The previous bits, $M_{j-1}$ and $M_{j-2}$, form the state of the encoder. As shown in the state diagram in Figure 12.2, the encoder is at a given state. The input bit $M_j$ causes a transition to another state at each clock edge, or $T_j$. Each state transition results in an output bit pair $N1_j$ and $N2_j$. Only certain state transitions are possible. Transitions due to a 0-bit input are shown in dashed lines, and transitions due to a 1-bit input are shown in solid lines. The output bits shown at each transition are labeled on each transition arrow in the figure.

Notice that the output $N1_j$ and $N2_j$ are both a function of the input bit $M_j$ and the two previous input bits $M_{j-1}$ and $M_{j-2}$. The previous bits, $M_{j-1}$ and $M_{j-2}$, form the state of the encoder. As shown in the state diagram in Figure 12.2, the encoder is at a given state. The input bit $M_j$ causes a transition to another state at each clock edge, or $T_j$. Each state transition results in an output bit pair $N1_j$ and $N2_j$. Only certain state transitions are possible. Transitions due to a 0-bit input are shown in dashed lines, and transitions due to a 1-bit input are shown in solid lines. The output bits shown at each transition are labeled on each transition arrow in the figure.


State M_{j-1}, M_{j-2}
Output N1_j,N2_j
Output N1_{j+1},N2_{j+2}
Output N1_{j+2},N2_{j+2}
0,0
1,0
0,1
1,1
(0,0)
(1,1)
(1,1)
(0,0)
(1,0)
(0,1)
(0,1)
(1,0)
T_j
T_{j+1}
T_{j+2}
T_{j+3}

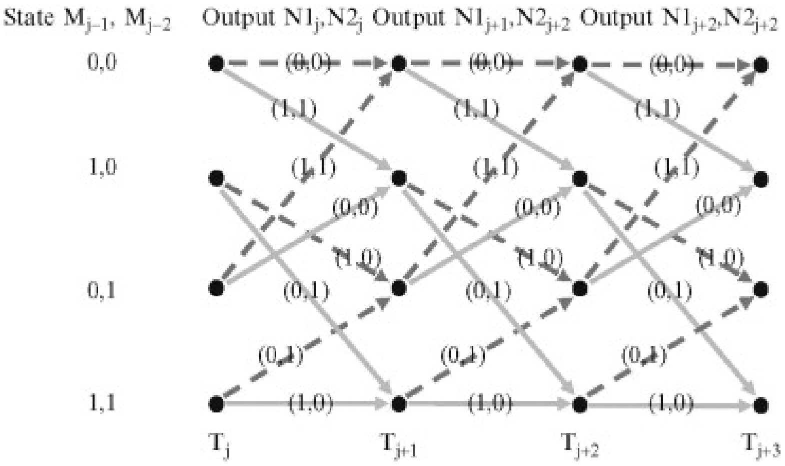


Figure 12.2.

Figure 12.2.

Table 12.1. Encoder state and outputs

Table 12.1. Encoder state and outputs

| Register Value | N1 Value | N2 Value | Timer Clock Value |
|---|---|---|---|
| 1 0 0 | 1 | 1 | T1 |
| 0 1 0 | 1 | 0 | T2 |
| 1 0 1 | 0 | 0 | T3 |
| 1 1 0 | 0 | 1 | T4 |
| 0 1 1 | 0 | 1 | T5 |
| 0 0 1 | 1 | 1 | T6 |
| 1 0 0 | 1 | 1 | T7 |
| 0 1 0 | 1 | 0 | T8 |
| 1 0 1 | 0 | 0 | T9 |
| 1 1 0 | 0 | 1 | T10 |
| 0 1 1 | 0 | 1 | T11 |
| 1 0 1 | 0 | 0 | T12 |

Now, let us trace the path of the input sequence through the trellis using Figure 12.3 and the resulting output sequence. This task helps us gain the insight that the trellis is representative of the encoder circuit because the trellis will be key in Viterbi decoding. The highlighted lines show the path of input sequence.

Now, let us trace the path of the input sequence through the trellis using Figure 12.3 and the resulting output sequence. This task helps us gain the insight that the trellis is representative of the encoder circuit because the trellis will be key in Viterbi decoding. The highlighted lines show the path of input sequence.

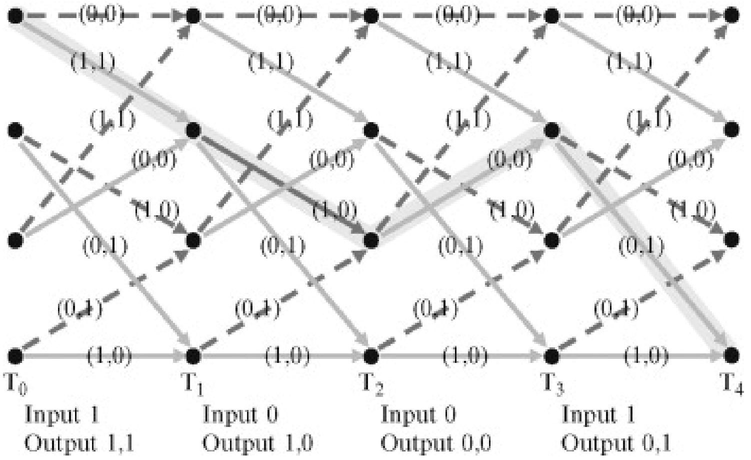


Figure 12.3-1.

Figure 12.3-1.

By tracing the highlighted path through the trellis, you can see that the output sequence is the same as our results when computing using the shift register circuit. For constraint length K, we have $(K - 1)_2$ states in our trellis diagram. Therefore, with K = 3 in our design example, we have 4 possible states. For a more typical K = 6 or K = 7 constraint length, there would be 32 or 64 states, respectively, although this is too tedious to try to diagram.

By tracing the highlighted path through the trellis, you can see that the output sequence is the same as our results when computing using the shift register circuit. For constraint length K, we have $(K-1)_2$ states in our trellis diagram. Therefore, with K = 3 in our design example, we have 4 possible states. For a more typical K = 6 or K = 7 constraint length, there would be 32 or 64 states, respectively, although this is too tedious to try to diagram.

> Read full chapter

> Read full chapter

# Application to Communication Systems

Application to Communication Systems

Yasuo Hirata, Osamu Yamada, in Essentials of Error-Control Coding Techniques, 1990

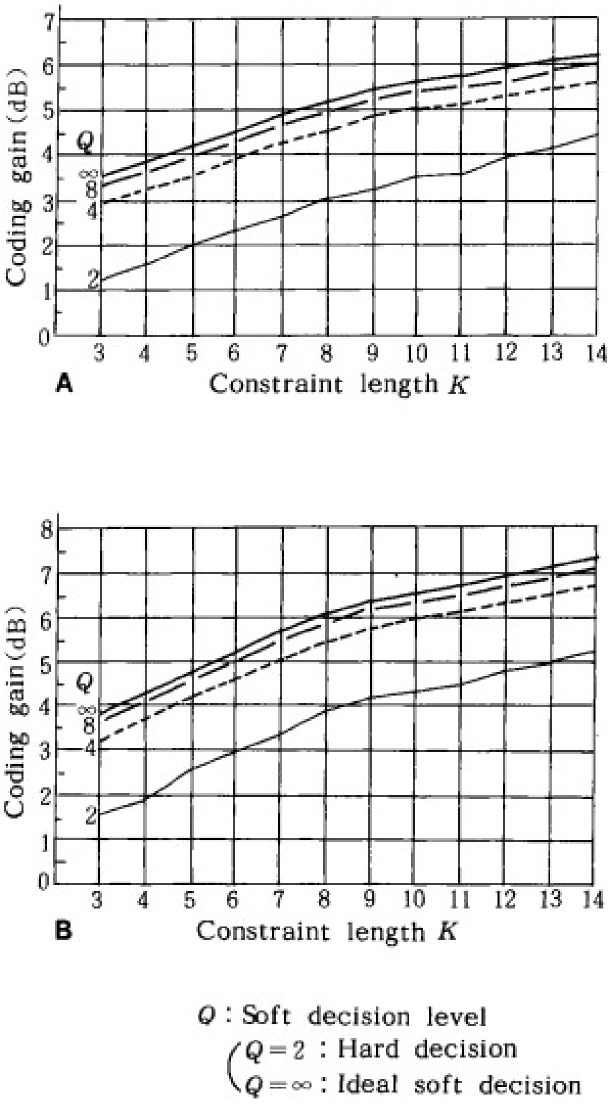
Yasuo Hirata, Osamu Yamada, in Essentials of Error-Control Coding Techniques, 1990

## FEC with Large Coding Gain

FEC with Large Coding Gain

To meet the requirement for FEC with a large coding gain, the soft decision Viterbi decoding will continue to play an important role in the future. Figure 6.9 shows the relationship between the attainable coding gain and the constraint length for the rate 1/2 convolutional coding (Yasuda *et al.*, 1981). Although the coding gain increases according to the increase of the constraint length as is seen in the figure, the hardware complexity of the Viterbi decoder also increases exponentially for the longer constraint length code. For this reason, the code with the constraint length of 7 is most widely used as of the late 1980s. Taking account of future progress, however, it seems that the Viterbi decoder for the code with the constraint length of up to 10 will be available, in which 1 dB higher coding gain compared with the constraint length of 7 can be achieved.

To meet the requirement for FEC with a large coding gain, the soft decision Viterbi decoding will continue to play an important role in the future. Figure 6.9 shows the relationship between the attainable coding gain and the constraint length for the rate 1/2 convolutional coding (Yasuda *et al.*, 1981). Although the coding gain increases according to the increase of the constraint length as is seen in the figure, the hardware complexity of the Viterbi decoder also increases exponentially for the longer constraint length code. For this reason, the code with the constraint length of 7 is most widely used as of the late 1980s. Taking account of future progress, however, it seems that the Viterbi decoder for the code with the constraint length of up to 10 will be available, in which 1 dB higher coding gain compared with the constraint length of 7 can be achieved.


A
Coding gain (dB)
0
1
2
3
4
5
6
7
Q
∞
8
4
2
3
4
5
6
7
8
9
10
11
12
13
14
Constraint length K
B
Coding gain (dB)
0
1
2
3
4
5
6
7
8
Q
∞
8
4
2
3
4
5
6
7
8
9
10
11
12
13
14
Constraint length K
Q : Soft decision level
Q = 2 : Hard decision
Q = ∞ : Ideal soft decision

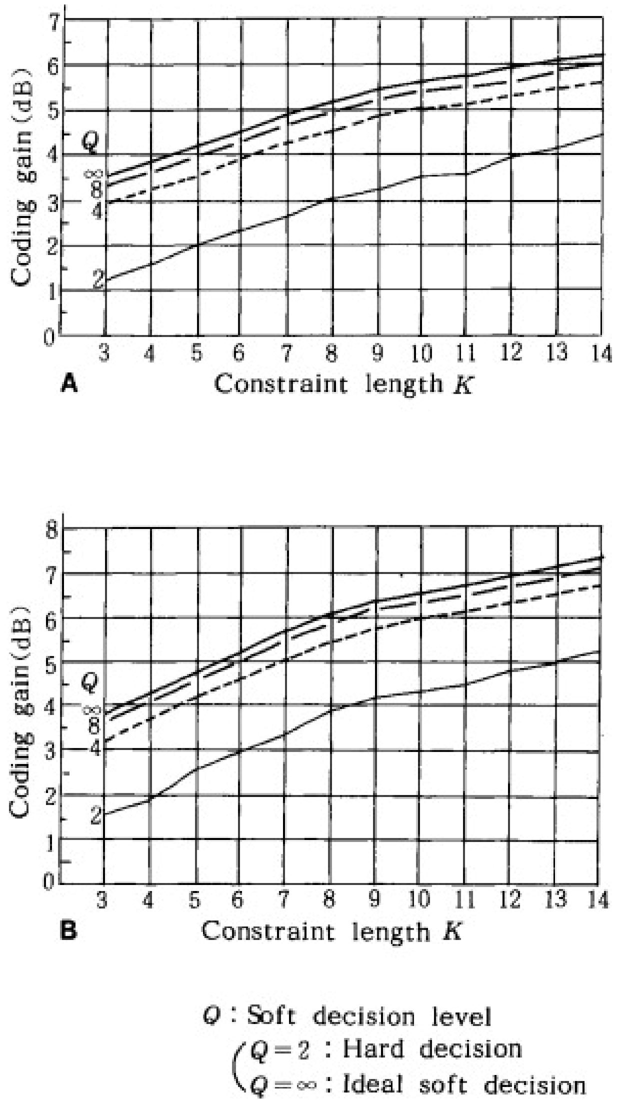
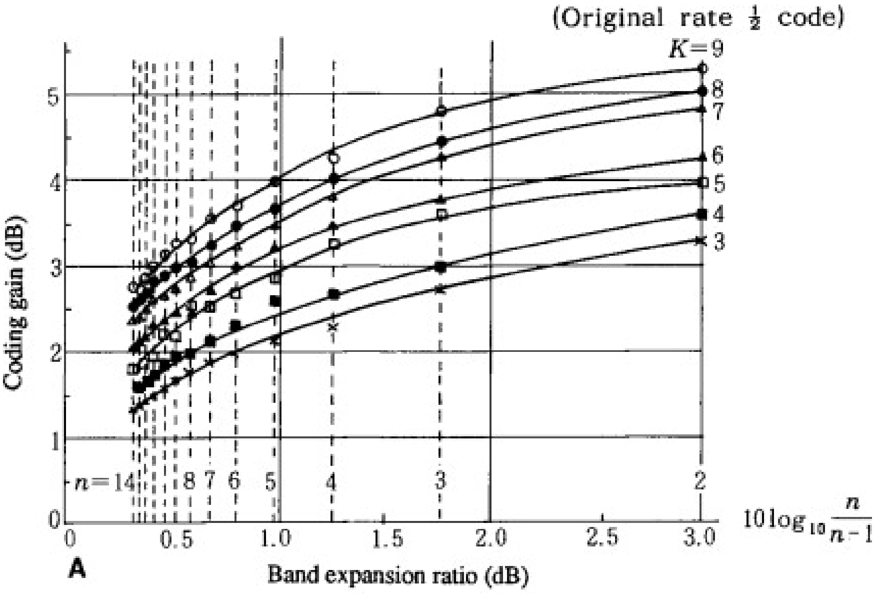


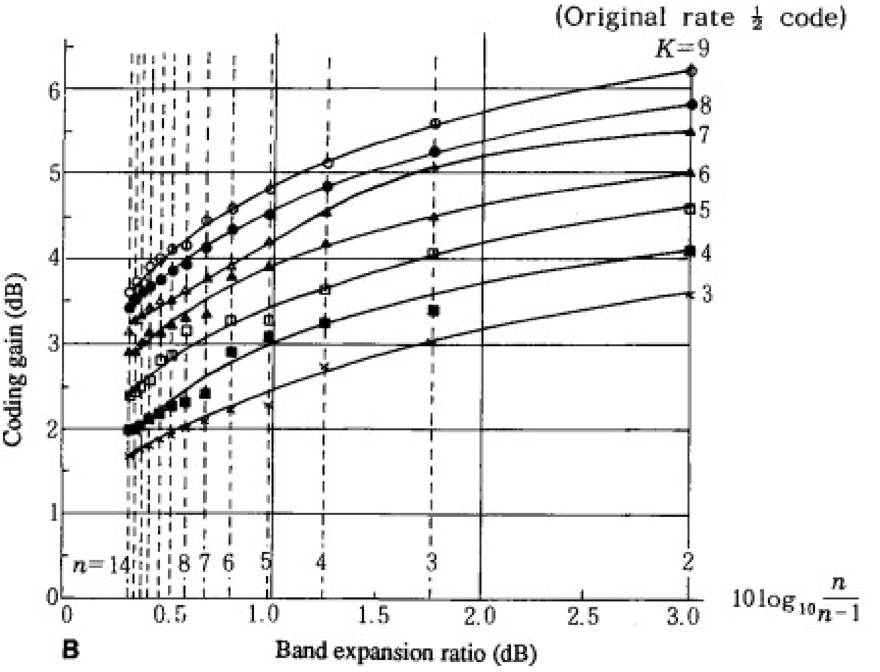
Fig. 6.9. Coding gain versus constraint length K of convolutional codes used for Viterbi decoding. (A) BER = $10^{-4}$; (B) BER = $10^{-6}$.

Fig. 6.9. Coding gain versus constraint length K of convolutional codes used for Viterbi decoding. (A) BER = $10^{-4}$; (B) BER = $10^{-6}$.

As for the code rates, the codes with higher code rate based on the punctured coding are expected to be widely utilized. Figure 6.10 shows the coding gain of the (*n* – 1)/*n* punctured code derived from the rate 1/2 convolutional code with the constraint length of 3 to 9 as a function of the bandwidth expansion ratio in dB (Yasuda *et al.*, 1984).

As for the code rates, the codes with higher code rate based on the punctured coding are expected to be widely utilized. Figure 6.10 shows the coding gain of the (*n* – 1)/*n* punctured code derived from the rate 1/2 convolutional code with the constraint length of 3 to 9 as a function of the bandwidth expansion ratio in dB (Yasuda *et al.*, 1984).


(Original rate $\frac{1}{2}$ code)
K=9
8
7
6
5
4
3
Coding gain (dB)
0
1
2
3
4
5
n=14
8
7
6
5
4
3
2
0
0.5
1.0
1.5
2.0
2.5
3.0
$10\log_{10}\frac{n}{n-1}$
A
Band expansion ratio (dB)


(Original rate $\frac{1}{2}$ code)
K=9
8
7
6
5
4
3
Coding gain (dB)
0
1
2
3
4
5
6
n=14
8
7
6
5
4
3
2
0
0.5
1.0
1.5
2.0
2.5
3.0
$10\log_{10}\frac{n}{n-1}$
B
Band expansion ratio (dB)

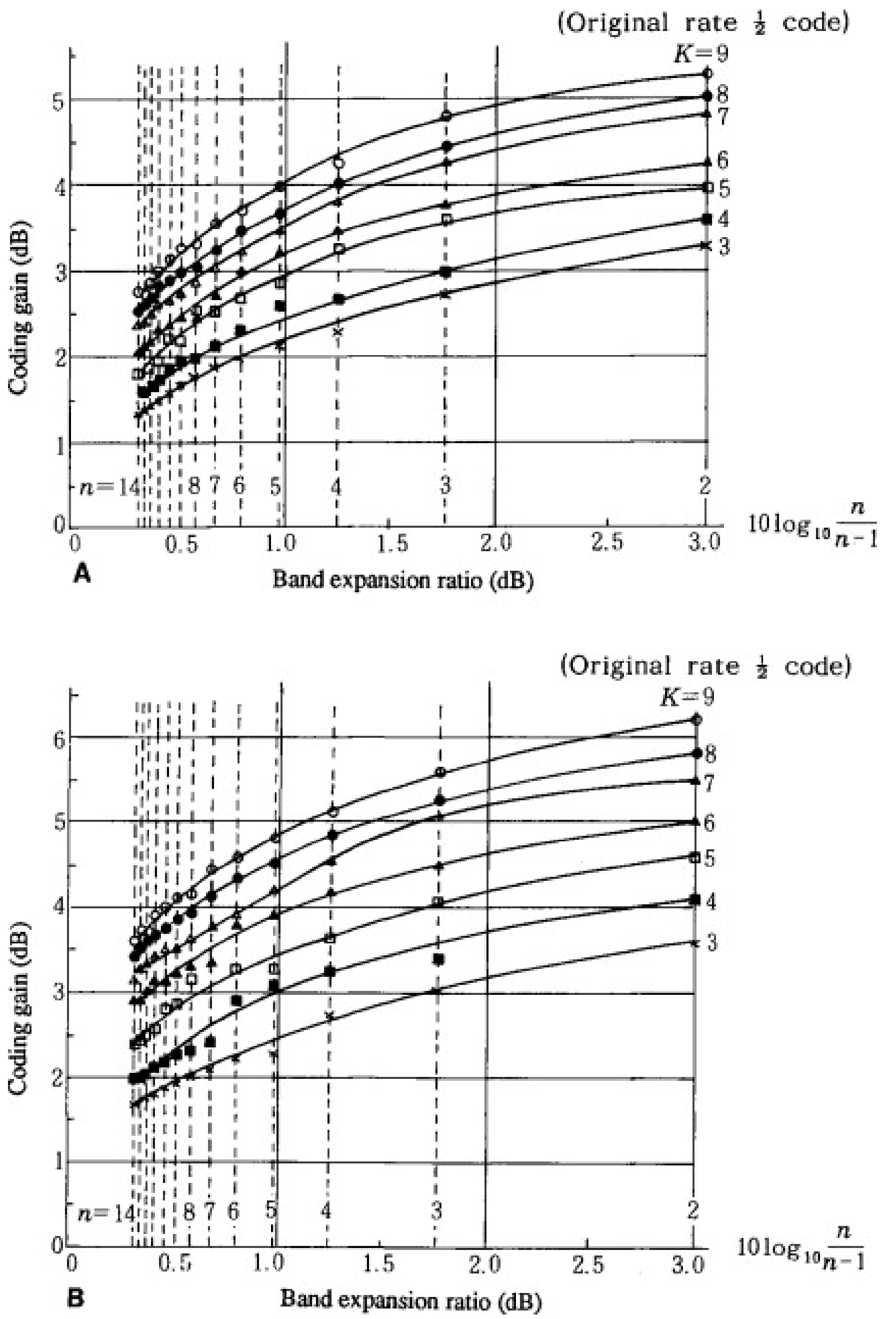


Fig. 6.10. Coding gain of rate $(n-1)/n$ punctured codes for soft decision Viterbi decoding (original rate 1/2 code with constraint length K = 3 to 9). (A) BER = $10^{-4}$; (B) BER = $10^{-6}$.

Fig. 6.10. Coding gain of rate $(n-1)/n$ punctured codes for soft decision Viterbi decoding (original rate 1/2 code with constraint length K = 3 to 9). (A) BER = $10^{-4}$; (B) BER = $10^{-6}$.

> Read full chapter

> Read full chapter

## Uplink Physical-Layer Processing

Uplink Physical-Layer Processing

Erik Dahlman, ... Johan Sköld, in 4G LTE-Advanced Pro and The Road to 5G (Third Edition), 2016

Erik Dahlman, ... Johan Sköld, in 4G LTE-Advanced Pro and The Road to 5G (Third Edition), 2016

### 7.4.1.4 PUCCH Format 4

7.4.1.4 PUCCH Format 4

With the extension of carrier aggregation to handle up to 32 component carriers, the payload capacity of PUCCH format 3 is not sufficient to handle the resulting number of hybrid-ARQ acknowledgments. PUCCH formats 4 and 5 were introduced in release 13 to address this problem.

With the extension of carrier aggregation to handle up to 32 component carriers, the payload capacity of PUCCH format 3 is not sufficient to handle the resulting number of hybrid-ARQ acknowledgments. PUCCH formats 4 and 5 were introduced in release 13 to address this problem.

PUCCH format 4, illustrated in Figure 7.26, is to a large extent modeled after the PUSCH processing with a single DFT precoder covering multiple resource-block pairs. An 8-bit CRC is added to the payload, followed by tail-biting convolutional coding and rate matching to match the number of coded bits to the number of available resource elements. Scrambling, QPSK modulation, DFT-precoding, and mapping to resource elements follow the same structure as the PUSCH—that is each DFT-spread OFDM symbol carries separate sets of coded bits. Multiple resource blocks in the frequency domain, 1, 2, 3, 4, 5, 6, or 8, can be used for PUCCH format 4, allowing for a very large payload. Inter-slot frequency hopping is used, similar to the other PUCCH formats and both normal and extended cyclic prefix is supported. There is also a possibility for a shortened format, leaving the last OFDM symbol in the subframe unused, for the case when sounding is configured in the subframe.

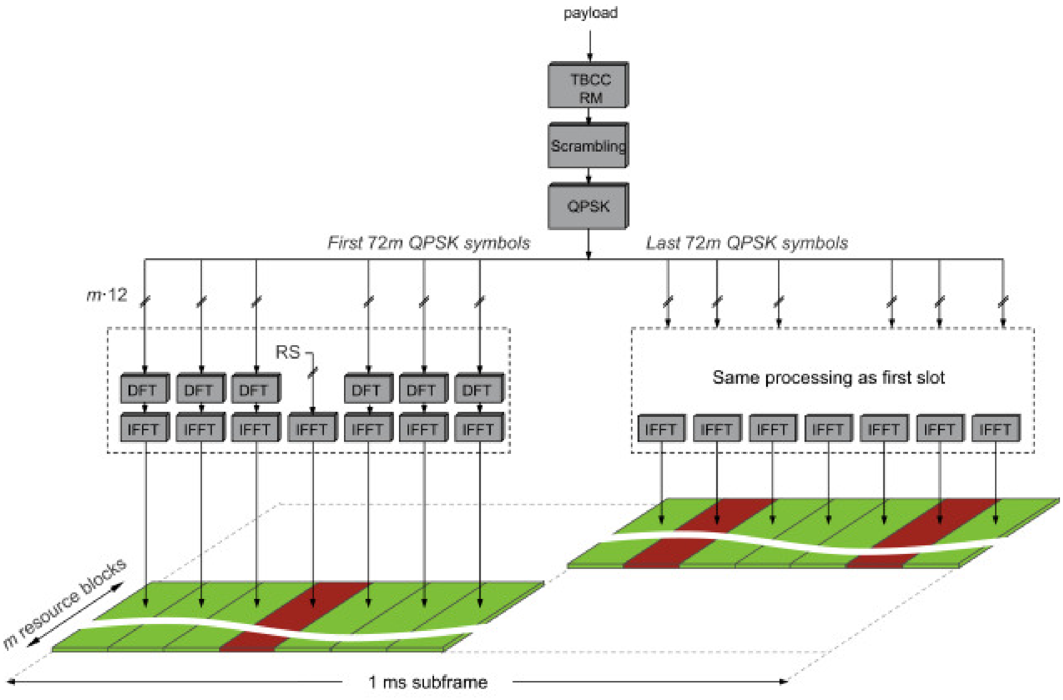


Figure 7.26. PUCCH format 4 (normal cyclic prefix).

> Read full chapter

ELSEVIER ScienceDirect is Elsevier's leading information solution for researchers.

Copyright © 2018 Elsevier B.V. or its licensors or contributors. ScienceDirect ® is a registered trademark of Elsevier B.V. Terms and conditions apply

RELX Group™