# Modulation and Coding

## Laboratory

## Channel Coding – Convolutional code

- **Convolutional codes.**

The main difference between block codes and convolutional codes is the encoding principle. In the block codes, the information bits are followed by the parity bits (e.x. in Hamming code (7,4) the four bits are information bits, and 3 bits are control parity bits. The control bits are distributed in designated positions - they are at position which are the next powers of the number 2)). In convolutional codes the information bits are spread along the sequence. That means that the convolutional codes map information to code bits not block wise, but sequentially convolve the sequence of information bits according to some rule.

A convolutional code is a type of error-correcting code that generates parity symbols via the sliding application of a boolean polynomial function to a data stream.

A convolutional codes are characterized by tree parameters:

$$(n, k, m)$$

where:
$n$ – number of inputs bits,
$k$ – number of output bits,
$m$ - number of memory register.

*code rate* is given by:

$$code\ rate = k / n$$

*Constraint length L* is given by:

$$constraint\ length\ L = k\ (\ m - 1)$$

$L$ are represented number of bits in the encoder memory that affects the generation of $n$ output bits.

## 1.1 Convolutional Encoder

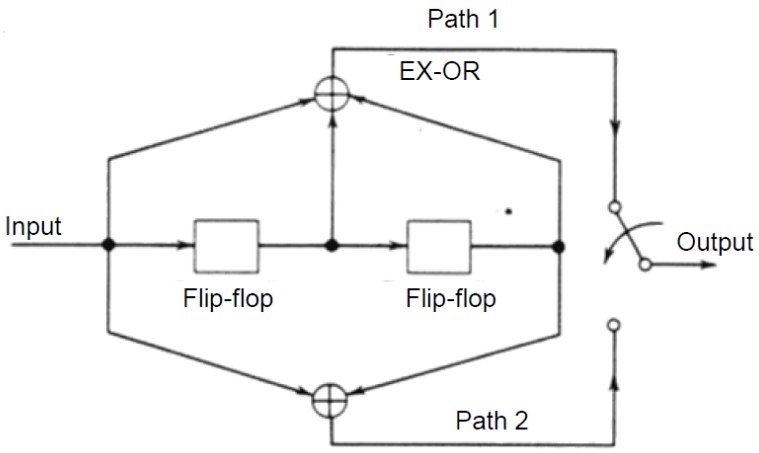


Figure 1: example of convolutional encoder

The implementation of the convolutional coder is shown in Figure 1. The code generated by this code is an unsystematic code. Unlike codes block, usually non-systematic convolutional codes are used.

Example:

Let the message to be encoded is: *10011*. At the beginning of coding, we assume that the encoder is zeroed, i.e. all the latches are set to *0*.

Next bits messages "slip in" on the left, and then "wander" in accordance with the direction of the arrows, after the way is stored in successive sliding registers.

Addition result (according to the diagram) from adders (modulo two), is send to the multiplexer, and thus to the output.

The output are bits pairs.

Note that when the bits from the input string are exhausted, the encoding goes on until resetting the shift register (register have only "*0*").

We assume then that the input of the encoder comes all the time

value *0*, this is the so-called message tail.

1.2 Convolution encoder as a Finite States Machine (FSM)

Convolutional encoder can be also represented as a Finite State Machine. Note that (in binary chase) we can define only four, possibility states: *00, 01, 10, 11* (it is all possible combination of pairs of bits), see fig. 2.

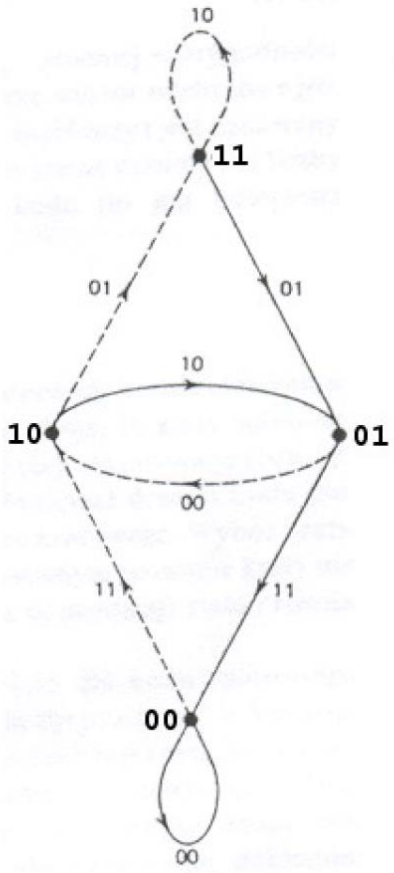


Figure 2 Convolutional Encoder as Finite State Machine

According to fig. 2: solid line is used when the input bits is *0*, and the dotted line is used when input bit is *1*. Sequence of bits, located next to the line, is a output pair of bits.

A trellis diagram is representation of coded message in time domain:

- Every possible codeword is represented by a single unique path through the trellis
- Every codeword starts and stops in $S_0$ the all-zeros state

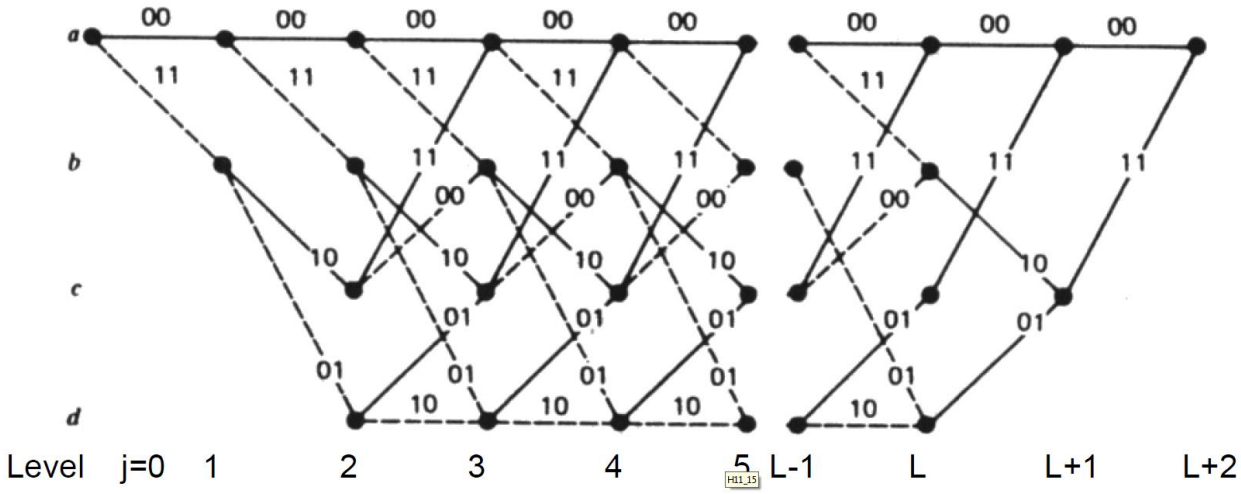


Figure 3: Example of trellis diagram

- Convolutional codes: simulation of data transmission

```
%Set general paramaters:

nbits = 100000;          % number bits to transmit
M = 16;                  % m-state for QAM
k = log2(M);
EbNo = 11;               % Energy/Bit to Noise ratio


%Define a convolutinal coding trellis for a rate 2/3 code.

trellis = poly2trellis([5 4],[23 35 0; 0 5 13]);
codeRate = 2/3;


%Generate random binary data.

dataIn = randi([0 1],nbits,1);

%Convolutinally encode the input data.

codeword = convenc(dataIn,trellis);
```

```
%Reshape the encoded column vector into a matrix having k columns. Then,
convert the binary matrix into an integer column vector.

% Convert encoded bits
codewordMat = reshape(codeword,length(codeword)/k,k);
dataSymbolsInCoding = bi2de(codewordMat);

% Convert originally bits (without channel coding)

dataInMatrix = reshape(dataIn,length(dataIn)/k,k);   % Reshape data into
binary k-tuples, k = log2(M)
dataSymbolsIn = bi2de(dataInMatrix);


%Apply 16-QAM modulation to the encoded symbols.
 0 - initial phase
'gray' - gray (differental) encoding

txSig = qammod(dataSymbolsIn,M,0,'gray');
txSigEC = qammod(dataSymbolsInCoding,M,0,'gray');


%Convert a 10 dB Eb/No to an equivalent signal-to-noise ratio. Pass the signal
through an AWGN channel.
snr = EbNo + 10*log10(k*codeRate);
rxSig = awgn(txSig,snr,'measured');
rxSigEC = awgn(txSigEC,snr,'measured');

%Demodulate the received signal.

demodSig = qamdemod(rxSig,M,0,'gray');
demodSigEC = qamdemod(rxSigEC,M,0,'gray');


%Demodulate the received signal.

demodSig = qamdemod(rxSig,M,0,'gray');
demodSigEC = qamdemod(rxSigEC,M,0,'gray');


%%%%% Path rof encdoded data %%%%%
%Set the traceback depth of the Viterbi decoder.

traceBack = 16;

%Decode the binary demodulated signal by using a Viterbi decoder operating in
a continuous termination mode.

dataOut = vitdec(demodSigBinary,trellis,traceBack,'cont','hard');

%Calculate the delay through the decoder, and compute the bit error
statistics.

decDelay = 2*traceBack;
[numErrorsEC,berEC] = biterr(dataIn(1:end-decDelay),dataOut(decDelay+1:end));
```

```
%%%%%% path for no-encoded data %%%%%

dataOutMatrix = de2bi(demodSig,k);
dataOut = dataOutMatrix(:);                    % Return data in column vector


% Display BER for both cases

[numErrors,ber] = biterr(dataIn,dataOut);
fprintf('\nThe binary coding bit error rate = %5.2e, based on %d errors\n',
...
    ber,numErrors)

fprintf('\n[Convolution error coding]The binary coding bit error rate = %5.2e,
based on %d errors\n', ...
    berEC,numErrorsEC)
```

2.1 Exercises:

- Measure *BER* coefficient for $E_B/N_O$ from 1 to 18 (with step = 1). Draw a chart (*OX* axis for *BER*, *OY* axis for $E_BN_O$)
- Measure BER for QAM4 and QAM64
- Try with other trellis, e.x "*7,[171 133]*" (for a rate 1/2, constraint length 7, convolutional code.)
- Compare with Hamming (7,4) code
- Compare your result with *bertool* (Matlab tool)