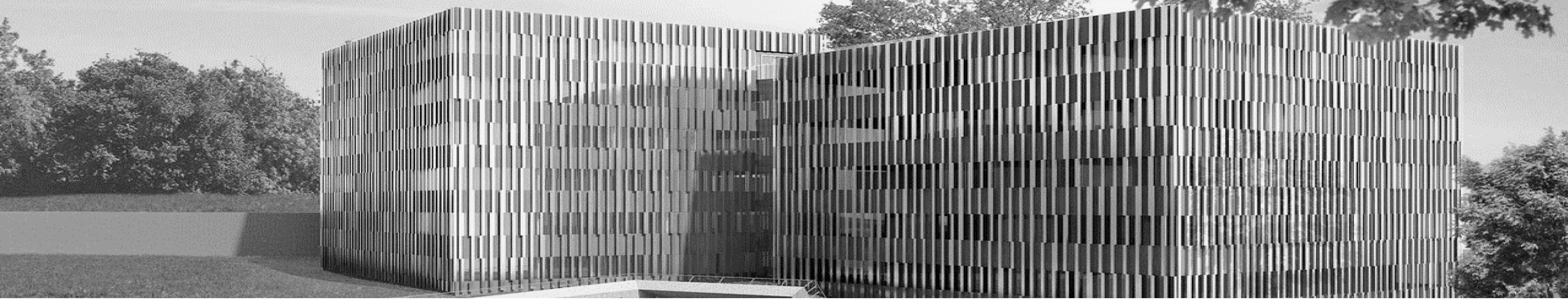


Signal Processing for Mobile Communications – Extended Transceiver Design

Programming Exercise 3: **FEC and Interleaving in OFDM**

About the programming exercise

- The general simulator structure (`L3_FEC_ofdm.c` file) is provided.

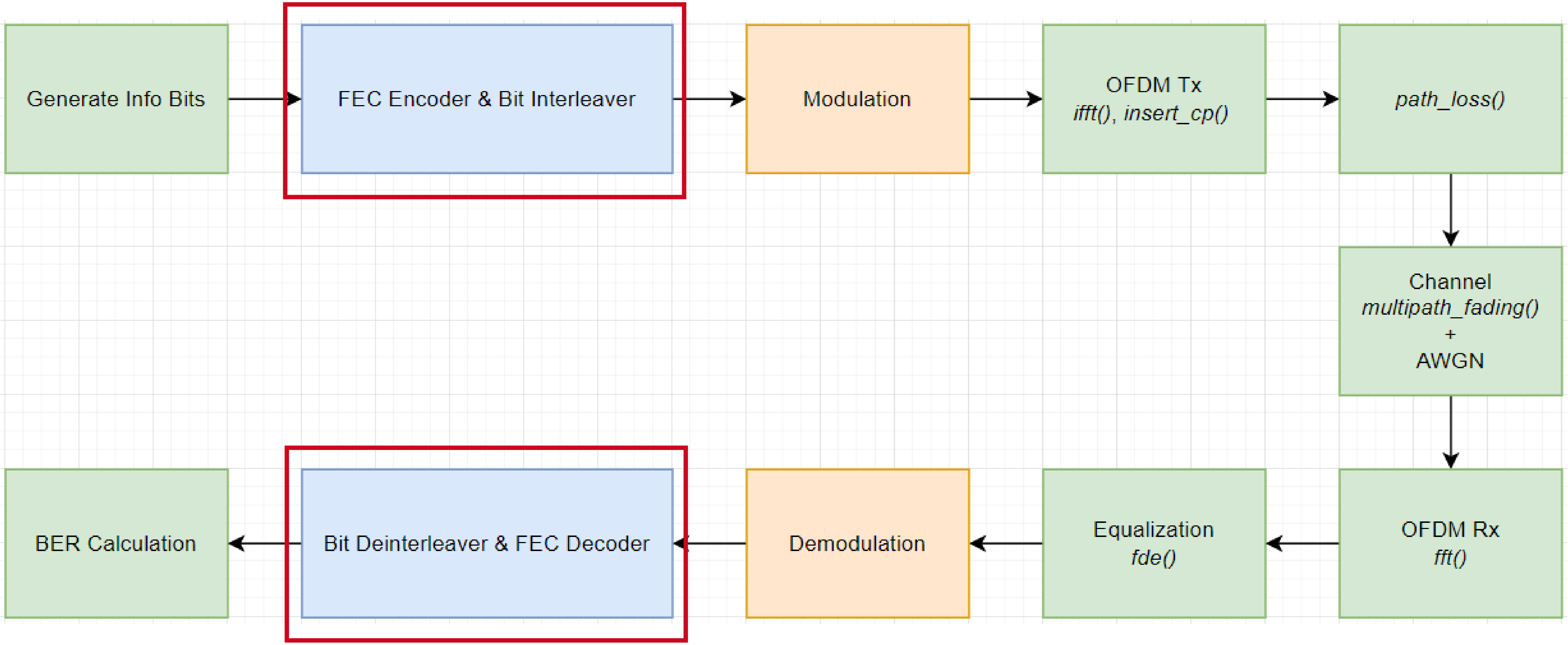
- We will discuss new functionalities you have to implement
  - Interfaces of new functions to be implemented are provided in `OFDM.h`.

- Implementations of new functions go to `L3_Student.c`

- You should analyze the provided simulator structure and understand the signal processing chains at the transmitter and the receiver.

Code flow

- Simulator structure (`main` file):



Tasks for Lab 3

1. Read `README.txt` first

2. Implement **Hamming (8,4) code**:
   – Encoding 4-bit information blocks into 8-bit codewords;
   – Decoding 8-bit codewords corrupted by fading and noise into 4-bit information blocks.

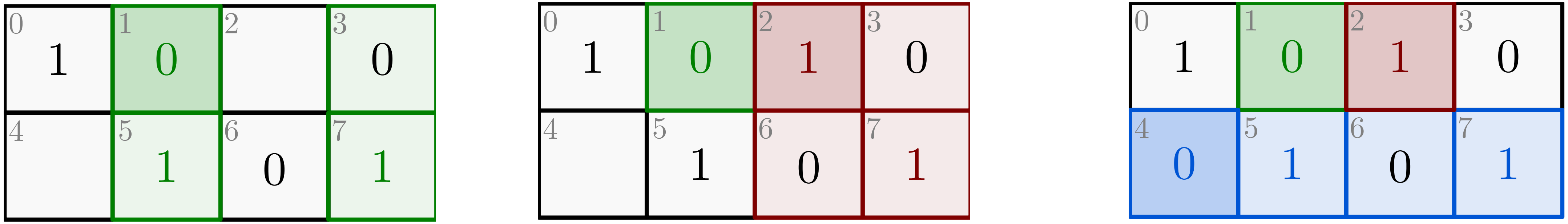
3. Implement **Interleaving** and **Deinterleaving**:
   – Shuffling the encoded bit sequence so each codeword is mapped on several OFDM subcarriers;
   – Reversing the operation at the receiver.

Tasks for Lab 3

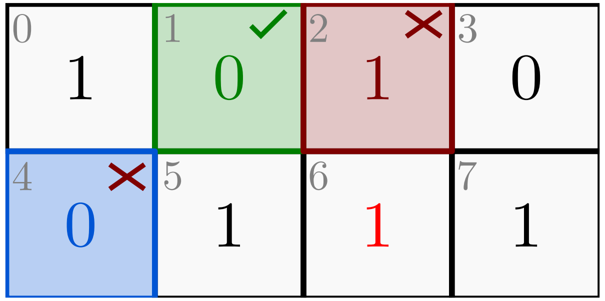
4. Evaluate system performance for both approaches
   - Plot bit error rate versus energy per bit E_b/N_0;
   - For 16QAM and MMSE equalizer, assuming perfect channel knowledge;
   - With and without coding (FEC + Interleaving);
   - For flat *(1 channel tap)* and frequency-selective fading (e.g., *8 channel taps)*;
   - Comment on the achieved system performance improvements the two types of fading:
     - Is there any difference and why?

Hamming code: Parity check calculations

- Position of a bit in a block can be isolated based on a few simple YES/NO questions, i.e. parity bits calculated over certain portions of the block.



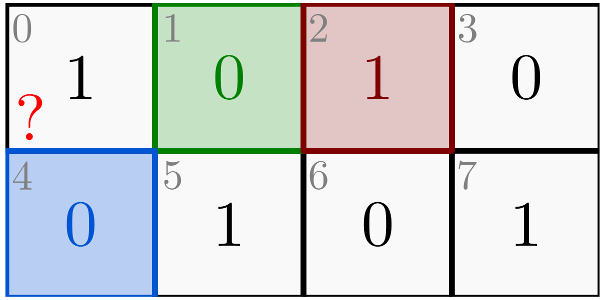
- Example:

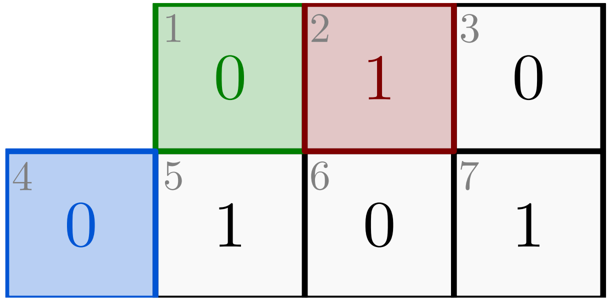


Error Position = 2 + 4 = 6 !!!

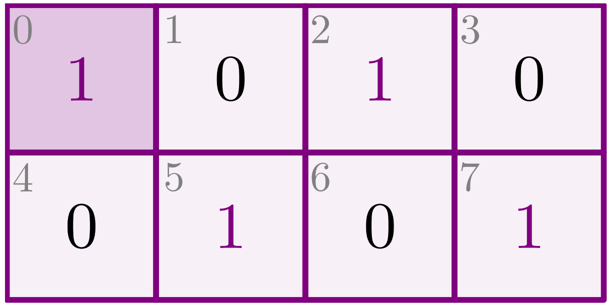
Hamming code: (7,4) and (8,4) variants

- To cope with the ambiguity, the bit at position 0 is either removed, i.e. Hamming (7,4) code, or set as an additional parity over the whole block, i.e. Hamming (8,4) code.





Hamming (7,4)



Hamming (8,4)

Hamming code: A more familiar presentation

- This is a non-systematic version, allowing for a better understanding of the concept.

p_0	p_1	p_2	x_2	p_4	x_5	x_6	x_7
0	1	2	3	4	5	6	7
1	0	1	0	0	1	0	1

$$p_0 = p_1 + p_2 + x_3 + p_4 + x_5 + x_6 + x_7$$

$$p_1 = \overset{0}{x_3} + \overset{1}{x_5} + \overset{3}{x_7}$$

$$p_2 = \overset{0}{x_3} + \overset{2}{x_6} + \overset{3}{x_7}$$

$$p_4 = \overset{1}{x_5} + \overset{2}{x_6} + \overset{3}{x_7}$$

Syndrome calculation in the decoder

$$s_0 = p_0 + p_1 + p_2 + x_3 + p_4 + x_5 + x_6 + x_7$$

$$s_1 = \overset{4}{p_1} + \overset{0}{x_3} + \overset{1}{x_5} + \overset{3}{x_7}$$

$$s_2 = \overset{5}{p_2} + \overset{0}{x_3} + \overset{2}{x_6} + \overset{3}{x_7}$$

$$s_4 = \overset{6}{p_4} + \overset{1}{x_5} + \overset{2}{x_6} + \overset{3}{x_7}$$

Hamming (8,4) block encoder (`L3_Student.c`)

- Implement a function that maps an input 4-bit information block to an output 8-bit codeword, according to the systematic Hamming (8,4) FEC code.

```
void hamming84_encoder(unsigned char infoBits[], unsigned char encBits[]);
```

4-bit input block

8-bit codeword (output)

- Implement a function that encodes a sequence of bits block by block.

```
void fec_encoder(unsigned char infoBits[], unsigned char encBits[], int infoBitsLen);
```

input bit sequence

encoded sequence (output)

input sequence length

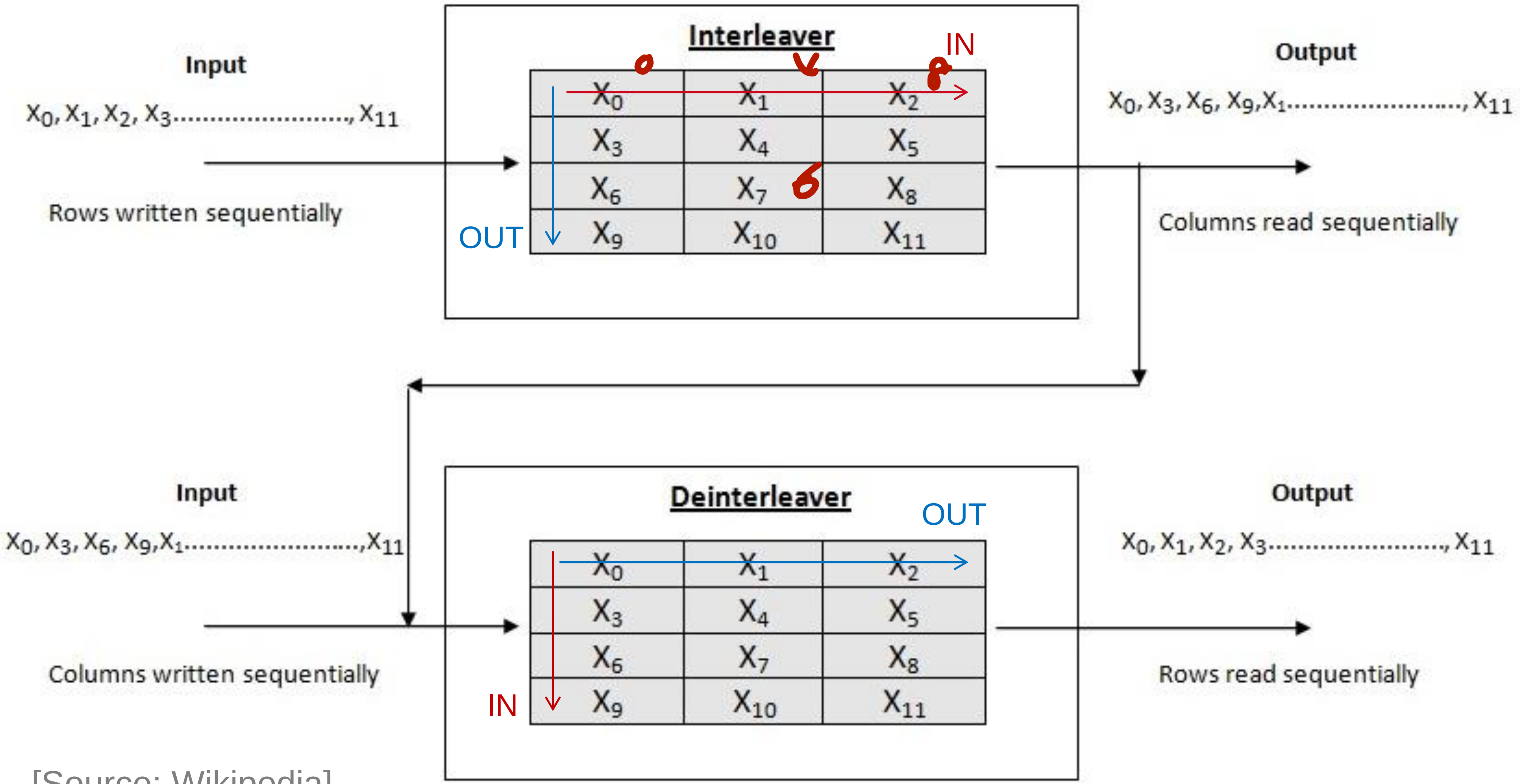
Hamming (8,4) block decoder (`L3_Student.c`)

- Implement the corresponding decoding functions employed at the receiver.
  - Note: The block decoder function should return an integer:
    - 0 – no errors detected
    - 1 – single (corrected) error
    - 2 – double (uncorrectable) error

```
int hamming84_decoder(unsigned char encBits[], unsigned char infoBits[]);

void fec_decoder(unsigned char encBits[], unsigned char infoBits[], int infoBitsLen);
```

Bit Interleaving



[Source: Wikipedia]

Interleaver and deinterleaver

- Implement functions that perform interleaving and deinterleaving **in-place**
  – You might find **`memcpy`** function useful for writing in the (de)interleaving matrix;

```
void bit_interleaver(unsigned char bitSeq[], unsigned char mat[], int len, int cwLen);
```

```
void bit_deinterleaver(unsigned char bitSeq[], unsigned char mat[], int len, int cwLen);
```

Input/Output
bit sequence

(de)interleaving matrix

Bit sequence
length

Codeword length
(no. columns in the matrix)

Submission

- Submit the report before 14:00 on 23.05.2022.

- Attach your code.

- Write a few meaningful sentences about obtained results.
  – How does the BER change with coding compared to the uncoded case?
  – Is there a difference between flat (1 channel tap) and frequency selective channels?
    - What is going on there?