

Informatique, Multimedia & Réseaux 2<sup>nd</sup> year - IMR-2  
2022 - 2023

# Wireless Project

## Practical Lab



**Robin Gerzaguët**

robin.gerzaguët@enssat.fr

•  
[https://gitlab.enssat.fr/rgerzagu/2a\\_imr\\_wirelessnetwork\\_project](https://gitlab.enssat.fr/rgerzagu/2a_imr_wirelessnetwork_project)  
•

**ENSSAT**  
LANNION



**Université  
de Rennes**

École Nationale Supérieure de Sciences Appliquées et de Technologie

Université de Rennes

Technopole Anticipa • Lannion

# Contents

<b>1</b>	<b>Description of the norm and main parameters</b>	<b>2</b>
1.1	Transmitter description . . . . .	2
1.1.1	Time Frequency Grid . . . . .	2
1.1.2	Main RF parameters . . . . .	3
1.1.3	Grid positioning description . . . . .	5
1.2	Forward Error Corrector (FEC) and Modulation and Coding Scheme (MCS) de- scription . . . . .	5
1.2.1	FEC encoder/decoder description . . . . .	6
1.2.2	Quadrature Amplitude Modulation mapping . . . . .	11
1.3	Channels description . . . . .	13
1.3.1	Synchronisation channels . . . . .	13
1.3.2	PBCH . . . . .	14
1.3.3	PDCCH . . . . .	16
<b>2</b>	<b>Wireless Lab</b>	<b>19</b>
2.1	Extraction of the time frequency matrix . . . . .	20
2.2	PBCH decoding . . . . .	21
2.2.1	BPSK decoding . . . . .	21
2.2.2	Hamming748 decoder . . . . .	22
2.2.3	PBCH decoding . . . . .	22
2.3	PDCCH decoding . . . . .	23
2.3.1	MCS decoding . . . . .	23
2.3.2	PDCCHU decoding . . . . .	23
2.4	PDSCH decoding . . . . .	24
2.4.1	FEC and MCS overall functions . . . . .	24
2.4.2	PDSCH decoding . . . . .	24

# Chapter 1


## Description of the norm and main parameters

### 1.1 Transmitter description

#### 1.1.1 Time Frequency Grid

As described in the main lecture, all the users share the same time-frequency (T/F) resources and are mapped in the same T/F container. In this container, different physical channels exist and have different purpose. In the project, we will consider the following channels

- The synchronisation channels, which will be used for synchronisation purpose (both in time and frequency). We will use 2 synchronisation channels. The primary channel (also called PSCH for Primary Synchronisation Channel) and the secondary channel (also called SSCH for Secondary Synchronisation Channel)
- The Broadcast channel (also called Physical Broadcast Channel or PBCH) that is addressed to all users and contains all useful information related to the base station, and some necessary information to decode the other channels.
- The Control Channel (or Physical downlink control channel (PDCCH)) which contains all the useful information to decode the payload (time frequency location, Modulation used, code rate and Forward Error Corrector (FEC) type used)
- The payload channel (Physical downlink shared channel or PDSCH) that contains all the payload to decode.

 The assumption that three physical channels entirely define the 5G-NR grid is only done for simplicity. You can find in the lecture slide a real 5G-NR time frequency grid with all the considered channels. In addition to the one presented and exploited in the project, many control channels exist (here merged into the PDCCH channel) in addition to broadcast channel.

Please note carefully the different channels names as we will use them intensively in the rest of the project. In our project, we will consider the following simpler time frequency grid, on which the different channels are represented. Please note that here we have allocated 4 users, and that this grid is automatically adapted to the number of current users.

The position of the channel may thus depend on the number of users and the size of each channels, but note that some channels position are fixed in the T/F grid

- The PSCH always occupies the first OFDM symbol and occupies all the frequency resource.
- The SSCH always occupies the second OFDM symbol and occupies all the frequency resource.
- The BCH always start at the beginning of the third symbol. Its size depends on the number of users (see the associated channel section for its description)
- The PDCCH and the PDSCHs have floating position and size, depending on the number of users, and current allocation.

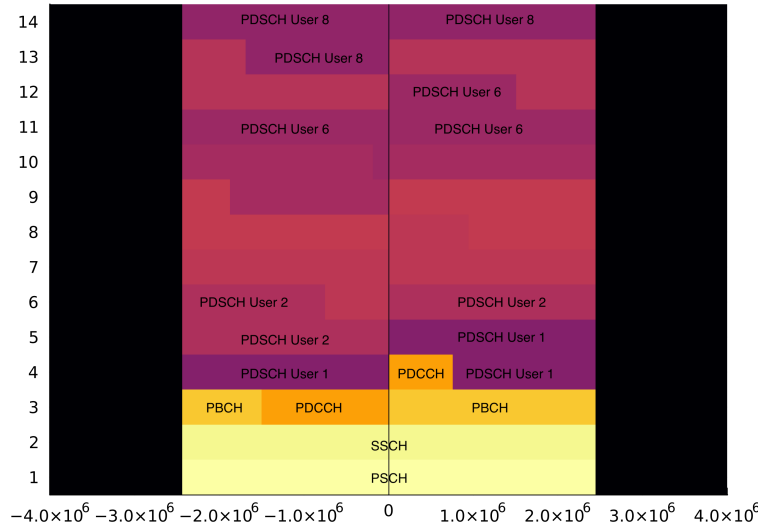


Figure 1.1: The proposed simplified time frequency grid.

### 1.1.2 Main RF parameters

We emulate a transmission between a base station (the transmitter) and the eNobeB (i.e mobile terminal as the receiver). To transmit the signal we use the **Orthogonal Frequency Division Multiplexing** (OFDM), which is adapted to multiplex different channel (the aforementioned T/F grid), and ease the channel compensation (one tap per subcarrier equalization scheme). A time and frequency representation of the transmitted signal in time domain is presented on Figure 1.2. Note the 2 area on the signal: the left part with constant amplitude corresponds to the synchronisation channel whereas the right part (with strong variations) corresponds to the data part. We can also have a look on the signal frequency domain on 1.3. We can see the classic OFDM spectrum with low side lobe decay, associated to the waveform. Each bin in frequency domain correspond to a subcarrier, that carry a different information.

In the project we will use the following parameter for the transmission, adapted to the 5G-NR standard (5G-NR-10MHz) [ZM07, 3GP09]

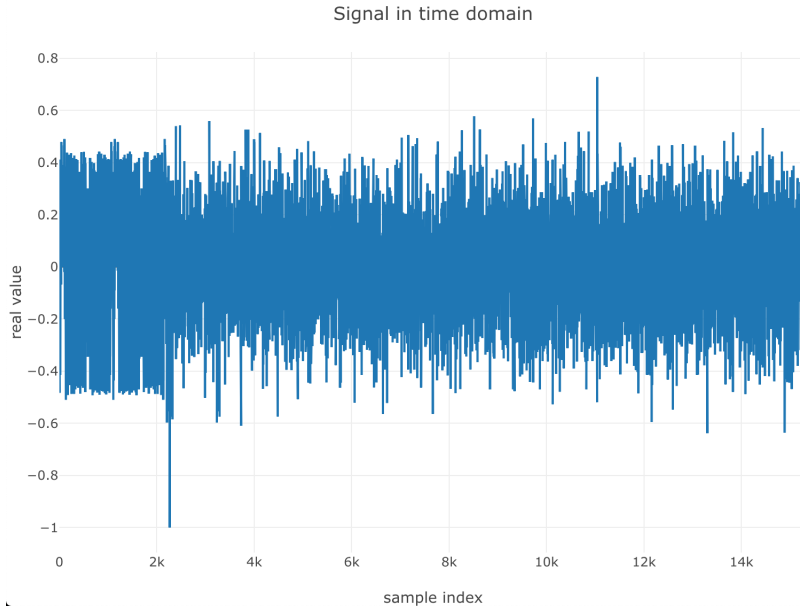


Figure 1.2: OFDM signal in time domain

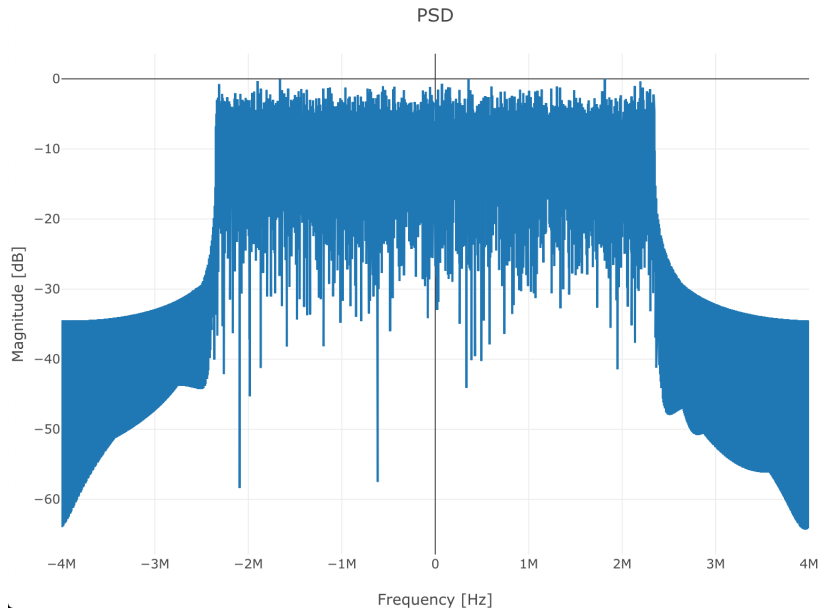


Figure 1.3: OFDM power spectral density

The FFT used in the project is of size 1024, but only 624 subcarriers are allocated. The allocated subcarriers are defined as follows:

$$S_a = U \left\{ \left( \left( 1 : \frac{N_{\text{Re}}}{2} \right) \right); \left( N - \frac{N_{\text{Re}}}{2} + 1 : N \right) \right\} \quad (1.1)$$

- $S_a$ : Vector of allocated subcarriers
- $N$  : FFT size (i.e 1024)

Table 1.1: Numerology

Parameter	Value	Difference with the norm ?
FFT size	1024	✓
CP size	72	✓
Sampling frequency	8 MHz	✗(hardware)
RB size (subcarriers per RB)	12	✓
Allocated subcarriers	624	✓
Waveform	OFDM	✓
Modulation	BPSK to 256-QAM	✓
Symbols per frame	14	✓

- $N_{\text{Re}}$ : Number of allocated subcarriers (i.e 624)

It means that the 312 first subcarriers and the 312 last subcarriers only are allocated. This is done as the signal belongs to  $\mathbf{C}$ , which lies to a frequency shift description of the spectrum: in other words, by defining  $S_a$  as it is done in (1.1), we ensure that the subcarriers around the DC are allocated (as it is shown in Figure 1.3).

### 1.1.3 Grid positioning description

Each channel belongs to the T/F grid, and to ease the positioning, we will consider a grid rule to position each of our channel.

- We have 14 symbols per frame, and symbol index is numbered from 1 to 14
- We have 624 subcarriers, grouped in 52 RB of 12 subcarriers. It means that we will only address subcarriers with multiple of 12. The RB index starts from 1 to 52. RB-1 corresponds to subcarriers from 1 to 12, and RB-52 corresponds to subcarriers from 612 to 624.
- We will position the channels by a grids coordinate (symbol index, RB index). Some examples are depicted on the Figure 1.4. Following how channels are described in Section 1.1.1, we can point out the position of the 3 fixed channel
  - PSCH starts at (1,1) and stops at (1,52)
  - SSCH starts at (2,1) and stops at (2,52)
  - PBCH starts at (3,1)

## 1.2 Forward Error Corrector (FEC) and Modulation and Coding Scheme (MCS) description

The channels use different FEC systems and different modulation scheme. In this section we describe how the FEC coder work and how they suppose to be used for proper decoding. These step are necessary to be applied before extracting the information bits contained in the different channels and subchannels.

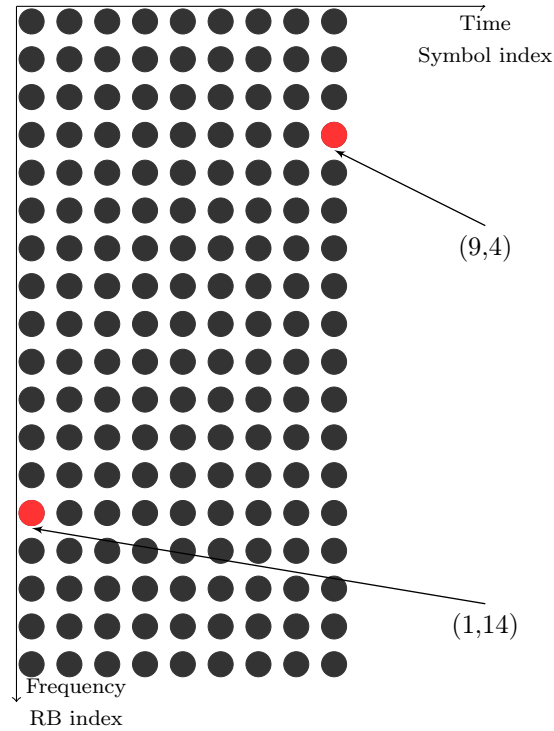


Figure 1.4: Grid position system description

### 1.2.1 FEC encoder/decoder description

In this section, we focus on the FEC systems. We consider 3 different systems

- The Hamming encoder/decoder, a simple and robust error corrector scheme
- The Convolutional encoder/decoder, a more sophisticated approach as it can use soft bit decoder (based on log likelihood ratio metric to be as closer to the best achievable performance)
- The polar code, a new scheme used in 5G.

#### 1.2.1.1 Hamming 748 coding scheme

Hamming codes are a family of linear error-correcting codes. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are perfect codes, that is, they achieve the highest possible rate for codes with their block length and minimum distance of three[Ham50].

In his original paper, Hamming elaborated his general idea, but specifically focused on the Hamming(7,4) code which adds three parity bits to four bits of data. It means that for each group of 4 input bits, the output of the FEC encoder will be 7 bits (the four original bits, and 3 redundancy bits). For instance, for an incoming sequence of 24 bits, the output bit length will be 42 bits.

This code is also called the **Hamming74** FEC.

In the project, we will focus on a slightly modified Hamming74 coding scheme on which we will add another parity bit: this code will be called the **Hamming748**. There is two purpose to this modification

- Add the possibility to detect one more error
- To have an output sequence that is exactly the double of the input sequence (as a consequence, 24 information bits will lead to 48 encoded bits).

### Encoder

To encode the data, a classic matrix formulation can be used. We denote

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

the four bits to be encoded. The output of the classic Hamming74 is denoted as

$$y_{74} = x^T \times A \quad (1.2)$$

where  $^T$  denotes the transpose operator,  $\times$  denotes a bitwise multiplication (the result is either 0 or 1 depending on the mod result) and  $A$  is called the encoding matrix of size  $4 \times 7$  and defined as follows

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (1.3)$$

This matrix is the matrix used in the Hamming74 coding scheme. The result of the Hamming748 is based on the output  $y$  on which a parity bit is added

$$y_{748} = B \times y \quad (1.4)$$

with  $\times$  denotes a bitwise multiplication (the result is either 0 or 1 depending on the mod result) and  $B$  is called encoding matrix of size  $8 \times 7$  and defined as follows

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (1.5)$$



It is to note that the Hamming748 can be done in a one step operation by defining the matrix  $D$  as  $D = A \times B$  and  $y_{748} = x^T \times D$

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (1.6)$$

In practice, to perform the FEC encoder, we should feed the encoder with a number of bits  $M = 4 \times N$  and it will return  $2M$  bits. A synoptic of the encoding scheme (system view) is depicted on Figure 1.5.

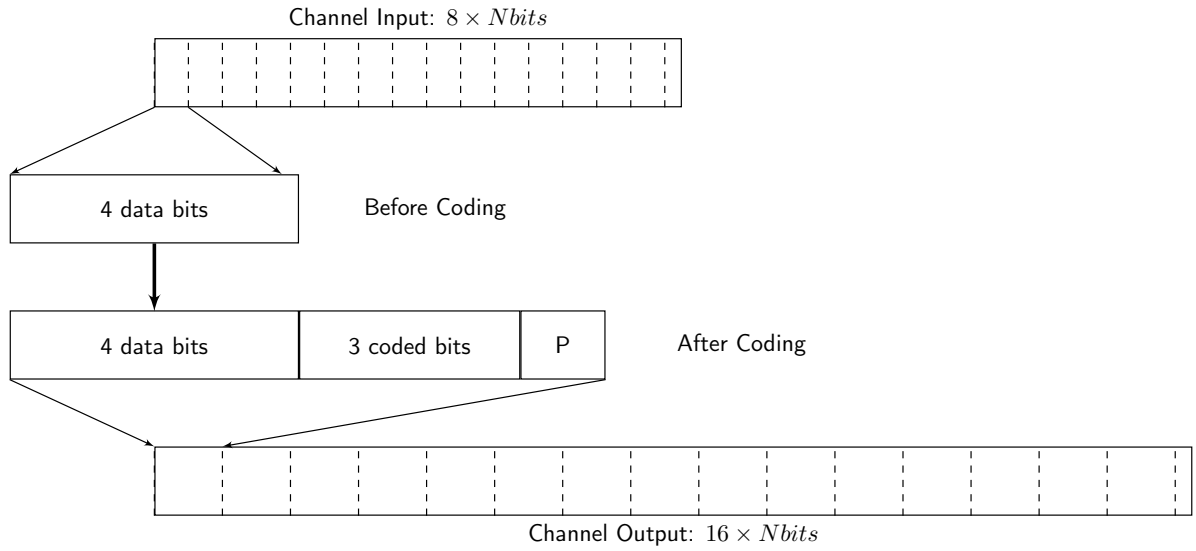


Figure 1.5: Hamming encoding scheme

### Decoder

To decode the data, we have to control that the parity equations are correct. If not, depending on which equation is false, the position of the erroneous bit can be recovered.

The first operation consist to apply a matrix decoding, through the matrix syndrome  $H$  defined as

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad (1.7)$$

This matrix is of size  $3 \times 7$  as it returns the 3 redundancy equations used for error correction. This syndrome is applied on the first 7 bits of the encoded word (i.e the output of a Hamming74

encoder). We denote this syndrome  $\epsilon$ .

$$\epsilon = H \times y_{74} = H \times \begin{bmatrix} y_{748}[0] \\ y_{748}[1] \\ y_{748}[2] \\ y_{748}[3] \\ y_{748}[4] \\ y_{748}[5] \\ y_{748}[6] \end{bmatrix} \quad (1.8)$$

The decoding as to be done as follows

- If there is no error,  $\epsilon$  should be  $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
- If there is an error in the first seven bits, the value of the syndrome gives the position of the error. For instance, if we have  $\epsilon = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$  the first bit is erroneous (decimal value associated to 001 is 1), whereas if we have  $\epsilon = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$  it is the third bit that contains an error (011 is 3 in decimal). The bit has to be corrected through a bit flip. We can be sure that the number of error is even as the last parity equation should be valid.
- If 2 error occurs, the syndrome cannot be equal to 0, but does not give the good position. However, we can ensure that there is a even number of error by using the last bit. Indeed, the summation of the seven first bits (with  $\text{mod } 2$ ) would be equal to the last bit value. In this case, The packet cannot be corrected and have to be dropped (a retransmission would be necessary).
- If the last bit is wrong (i.e, summation of the seven first bits (with  $\text{mod } 2$ ) is not equal to the last bit), but that the syndrome is equal to 0, we can ensure that the transmission went OK and that only this bit is wrong.

### 1.2.1.2 Convolutional code v27

As described on its Wikipedia page, convolutional code (CC) is a type of error-correcting code that generates parity symbols via the sliding application of a boolean polynomial function to a data stream. The sliding application represents the *convolution* of the encoder over the data, which gives rise to the term 'convolutional coding'. The sliding nature of the convolutional codes facilitates trellis decoding using a time-invariant trellis. Time invariant trellis decoding allows convolutional codes to be maximum-likelihood soft-decision decoded with reasonable complexity.

#### Encoder

There are many CC architectures and implementations [BM96], and in this project we will focus on the so-called v27 convolutional code. It has a rate of  $1/2$  and use a depth of 7 samples. It means that the output of the coded stream is based on (at max) 7 incoming samples (so a maximal delay of 6). The CC encoder is described in Figure 1.6.

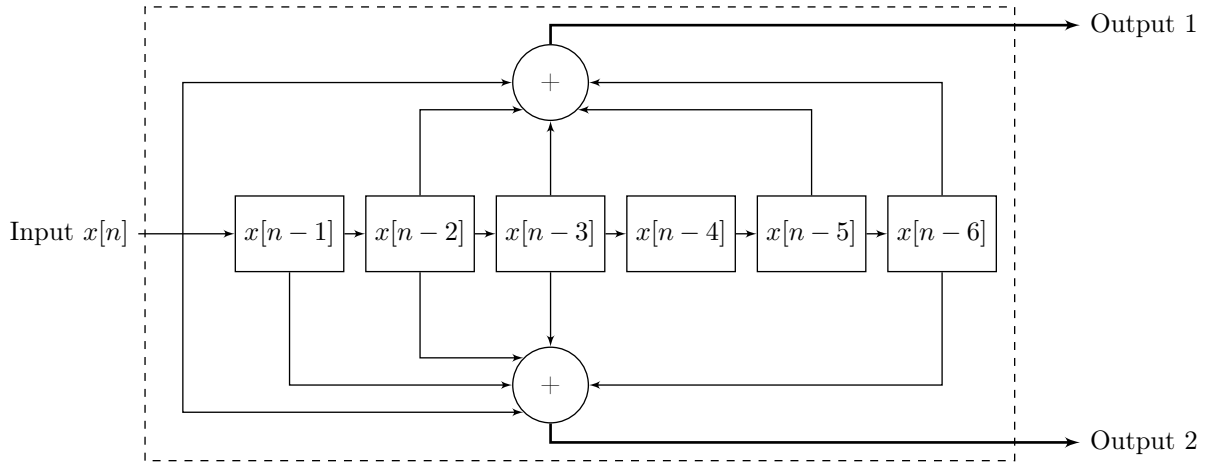


Figure 1.6: V27 Encoder with (133,171) polynomial for generation [Gol05].

A CC generates two outputs denoted as stream 1 (at top in Figure 1.6) and stream 2 (bottom in Figure 1.6). The two streams are obtained with modulo 2 adders based on the incoming signal with different delay.

- Stream 1 uses  $x[n]$ ,  $x[n-2]$ ,  $x[n-3]$ ,  $x[n-5]$  and  $x[n-6]$ . This sequence can be defined as a binary sequence which is 1011011, or more conveniently in octal as 133. (To convert a binary sequence, to octal, the binary sequence is separated in 3 bits groups [1;011;011] which gives [1;3;3] or 133 in octal.)
- Stream 2 uses  $x[n]$ ,  $x[n-1]$ ,  $x[n-2]$ ,  $x[n-3]$  and  $x[n-6]$ . This sequence can be defined as a binary sequence which is 1111001, or more conveniently in octal as 171. (To convert a binary sequence, to octal, the binary sequence is separated in 3 bits groups [1;111;001] which gives [1;7;1] or 171 in octal.)

This is the reason why the proposed code is called a v27 CC, or a CC with  $K = 7$  and generation polynomials of [133,171] or equivalently [1011011,1111001].

If an incoming sequence  $x[n]$  of size  $N$  is proposed to the encoder, it results into 2 sequences. The first one  $y_1$  is obtained from output 1 is of size  $N+K+1$  and the second one  $y_2[n]$  is obtained from output 2 and is of size  $N+K+1$ . The complete coded sequence is denoted  $y$  is of size  $N+2(K+1)$  and is obtained as the multiplexing of the 2 sequences.

$$y = [y_1[1]; y_2[1]; y_1[2]; y_2[2] \dots y_1[end]; y_2[end]] \quad (1.9)$$

For instance, if the incoming sequence is  $x = [1; 1; 0; 1]$ , the encoded sequence is

$$y = [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0].$$

## Decoder

Several decoders can be applied to recover the encoded sequence. Among them, the best performance can be obtained with the use of a Viterbi Decoder that is a Maximum Likelihood decoder and thus minimizes the error probability [For73].

The decoding process is not in the scope of this lecture. The interested readers can refer to some noticeable works of the literature, for instance [HOP96, Vit98].

In the project context all the functions required for proper decoding are provided. We use the Scikit module of Python to be able to decode. As the documentation stated, we will need the generator polynomial and the depth of the Viterbi algorithm, set to 6.

---

```
1 # Define a Convolutional decoder as a v27 with 6 depth for decoding
2 cc1 = fec.FECConv(('1011011', '1111001'), 6)
```

---

## 1.2.2 Quadrature Amplitude Modulation mapping

In this part, we focus on the mapping operation, i.e the operation that transforms the binary stream into a symbol stream. Depending on the modulation format used, several bits can be transformed into one symbol: we call the number of bits per symbol  $m$  which is typically between 1 (BPSK) and 8 (256-QAM)[PS01].

In this project, we will use different modulation format (i.e different values of  $m$ ). It is important to have in mind that having an higher  $m$  leads to a better bitrate but also leads to higher sensitivity to noise. As a consequence, in a real system, the value of  $m$  is *adapted* to the encountered signal quality. Furthermore, the control channel bears very important data, as if these channels cannot be detected all the information is lost. As a consequence, these channels often have low value of  $m$  to limit the risk to lost all the packet in the current frame.

### 1.2.2.1 BPSK (also called 2-QAM)

The first possibility is to have a BPSK format. In such a case, each bit is transformed into a symbol with the following rule

- If the bit is 0, the generated symbol is  $-1$
- If the bit is 1, the generated symbol is  $1$

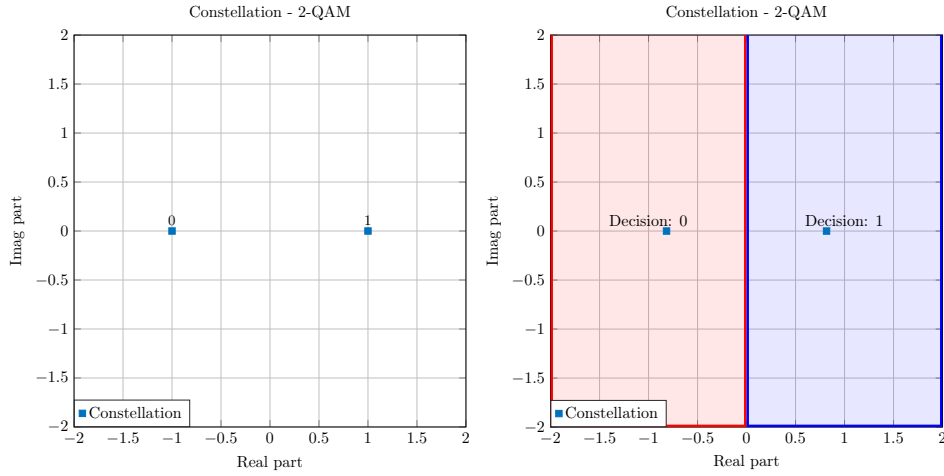


Figure 1.7: BPSK mapping and Voronoi regions

For instance, the symbol sequence associated to  $[1;0;0;1;1;0]$  is  $[1;-1;-1;1;1;-1]$ . We can represent the obtained constellation and this is done on Figure 1.7.

To demodulate such a sequence, one has to be sure that the noise will not affect the decision. Thus, we define two area: if the received signal has a negative real part, the associated bit should be 0. On the contrary, any symbol with a positive real part should be transformed into a 1. A way to automatize this partition is to define a threshold, associated to 0. This kind of space partitionning is called Voronoi region [CS82]. The display of the two decision region is shown on right of Figure 1.7.

#### 1.2.2.2 QPSK (also called 4-QAM)

We can use higher order modulation and use the fact that the modulation can be done on two independent paths, represented here by the complex formulation. In this case, one can use the 4-QAM which associates one symbol to a tuple of 2 bits. The mapping operation is represented on Figure 1.8.

Note the position of the tuples. Each closest element only differs from 1 bit. It means that we can not use any arbitrary position association: this is called the gray coding. For instance, the symbol sequence associated to  $[1;0;0;1;1;0]$  contains 3 symbols  $[1-1i; -1+1i; 1-1i]$ . To apply the decision to the incoming QPSK stream, the same approach as the one done in BPSK can be applied. We have now four different regions, based on the real and imaginary parts. The regions are depicted on the right of Figure 1.8.

#### 1.2.2.3 Higher modulation order

In the project, we will also consider 16-QAM and 64-QAM and the mapping between bits and sequences is described on Figure 1.9. The voronoi regions can be defined as threshold that are at each even value.

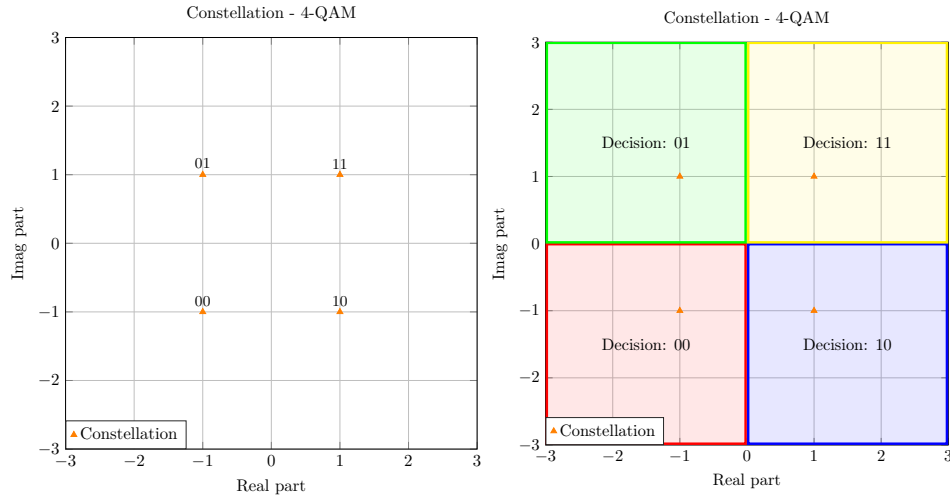


Figure 1.8: QPSK mapping and Voronoi regions

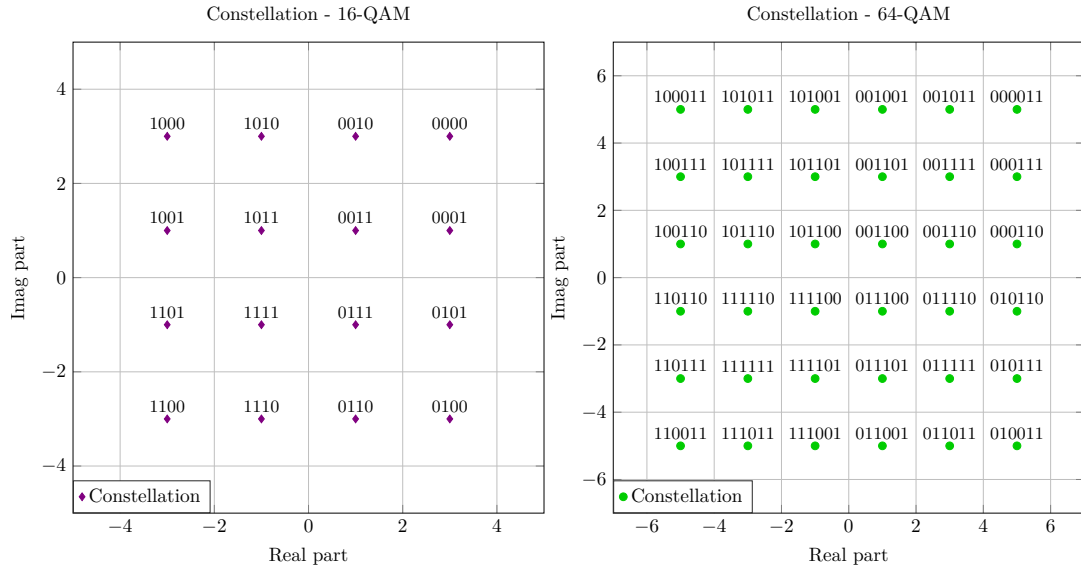


Figure 1.9: 16-QAM and 64-QAM mapping association

## 1.3 Channels description

In this section, we describe the characteristics of the different channel encountered in the project.

### 1.3.1 Synchronisation channels

The 2 first OFDM symbols are occupied by synchronisation symbols. Their purpose is to guarantee that the eNobeB will be able to be synchronized with the base station and be able to detect and decode the information of the other channels.

The PSCH and the SSCH are special sequence, defined in frequency domain (i.e in the T/F grid) such that the signal in time domain offers good performance for autocorrelation: these sequence are called Zadoff-Chu (ZC) Sequence[GML14], or CAZAC sequence (Constant Amplitude with Zero

Autocorrelation) sequence. Assuming that these sequences are known at the receiver, performing a correlation between the received signal and the CAZAC sequence will create a strong peak at the perfect synchronisation sample index [ZM07].

The expression of the CAZAC sequence for the primary synchronisation channel is described in equation (1.10)

$$S_{\text{psch}}[n] = 10^{\frac{p_b}{20}} \times e^{\frac{j\pi\mu_{\text{pss}} \times n \times (n + \text{mod}(N_{\text{Re}}, 2))}{N_{\text{Re}}}} \quad (1.10)$$

- $S_{\text{psch}}[n]$ : PSCH signal in frequency domain
- $n$  is the subcarrier index [between 1 and  $N_{\text{Re}}$ ]
- $P_b$  is the boost applied on the sequence in dB: It is equal to 3dB
- $\mu_{\text{pss}}$  : Kernel of the ZC sequence: It is equal to 2
- mod is the modulo operator
- $N_{\text{Re}}$  is the number of total subcarriers (i.e.  $52 \times 12 = 624$ )

The expression of the secondary synchronisation channel is as follows:

$$S_{\text{ssch}}[n] = 10^{\frac{p_b}{20}} \times e^{-\frac{j\pi\mu_{\text{pss}} \times n \times (n + \text{mod}(N_{\text{Re}}, 2))}{N_{\text{Re}}}} \quad (1.11)$$

These sequences will be used for synchronisation, by converting them in time domain after mapping the  $N_{\text{Re}}$  elements on the previously specified  $S_a$  subcarriers

$$\begin{cases} s_{\text{psch}}[m] = \text{FFT}(S_{\text{psch}}) = \sum_{k=0}^{N_{\text{FFT}}} S_{\text{psch}}[k] e^{-2j\pi \frac{km}{N_{\text{FFT}}}} \\ s_{\text{ssch}}[m] = \text{FFT}(S_{\text{ssch}}) = \sum_{k=0}^{N_{\text{FFT}}} S_{\text{ssch}}[k] e^{-2j\pi \frac{km}{N_{\text{FFT}}}} \end{cases} \quad (1.12)$$

and be used in frequency domain for channel compensation (Zero-Forcing (ZF) equalizer)

### 1.3.2 PBCH

The PBCH (Physical Broadcast Channel) contains useful information for all users that share the same T/F container. It means that all the users have to decode it. The modulation format for the PBCH is fixed as always the same: the channels follows a **BPSK scheme** with **Hamming748 FEC** (see Section 1.2.1.1). After correctly demodulate and apply the FEC decoder, it contains the following important informations:

- The cell identifier and number of users
- And all the users PBCHU, which each of them contains the following information
  - The user identifier
  - The modulation and coding scheme (MCS) of user PDCCH
  - The position of the PDCCH for the current user
    - \* With the symbol index (see grid description in Section 1.1.3)
    - \* And the RB position

- The HARQ policy for user (not used, filler bits here) [And12].

The detail of the frame is described in Figure 1.10. The size is described with the inclusion of the FEC encoding. As the FEC encoder has (in the PBCH case) a rate of  $1 \div 2$ , the number of bits after FEC is multiplied by 2 (see Section 1.2.1.1 for the FEC description). The detail of the information bit is described in the top and bottom boxes. They assume that the FEC decoder has been applied (see section 1.2.1.1:).

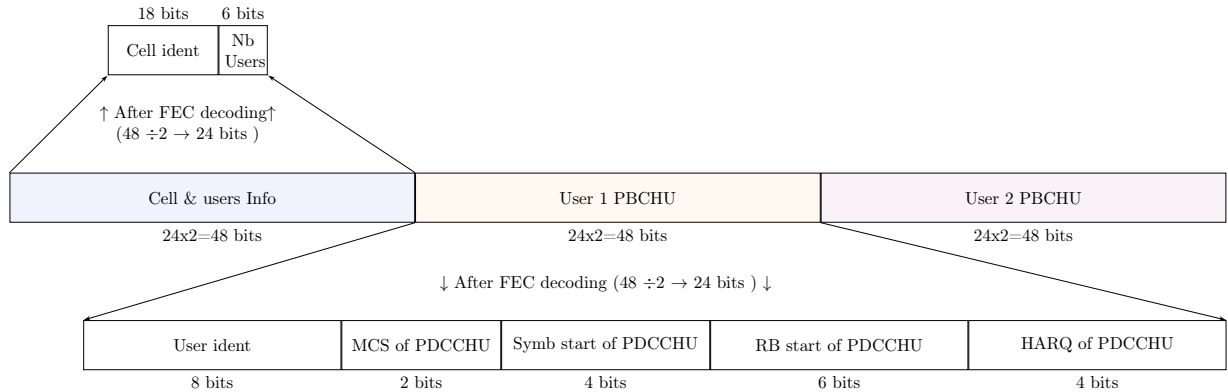


Figure 1.10: BPCH frame

#### Remark

For all binary information, the conversion to an integer value is done with a big-endian formalism.

To obtain the usefull information, 3 steps have to be done

- Recover the subchannels, as shown by the box color on Figure 1.10. In this case, each sub-channel as a size of 48 bits.
- Apply the FEC decoder to obtain data subchannels of 24 bits (see Section 1.2.1.1 for encoder/decoder description)
- Segment and recover the different information flags as depicted on the top and the bottom of Figure 1.10.

As a user the purpose is to detect which PBCHU is of interest. This can be done by decoding all the content and then extract the information related to the PBCHU that matches the associated user ident. After this steps, ther user have extracted 4 usefull information

- The MCS of the next channel to decode (the PDCCH)
- The position of the PDCCHU (part of the PDCCH associated to the current user)
- The HARQ policy (not used in this project).





Figure 1.11: PDCCH and PDCCHU allocations

### 1.3.3 PDCCH

The PDCCH (Physical Downlink Control Channel) regroups all the PDCCHU (Physical Downlink Control Channel for User). The scheme of the channel is depicted on Figure 1.11.

- The position of the PDDCH of interest is given by decoding the previous channel (PBCH) for user of interest. There is no need to decode all the PDCCH and the PBCH gives the position of the PDCCHU !
- The modulation format and the FEC used in the PDCCHU is given by the MCS flag decoded in the PDCCH. This flag is on 2 bits and thus has a value between 0 and 3. The mapping between this value and the PDCCHU parameter is given in Table 1.2. In this project, we only consider the Hamming cases, it means that the MCS flag (from the PBCH) should be 0 or 2.

Table 1.2: MCS for PDCCH		
MCS value	Modulation	FEC used
0	BPSK	Hamming748
1	BPSK	Polar 1/3 • <b>Not in the project</b>
2	QPSK	Hamming748
3	QPSK	Polar 1/3 • <b>Not in the project</b>

The PDCCHU have also several information flags that can be obtained after FEC decoding. The informations are regrouped on Figure 1.12. The information will address the PDSCH, i.e the payload content.

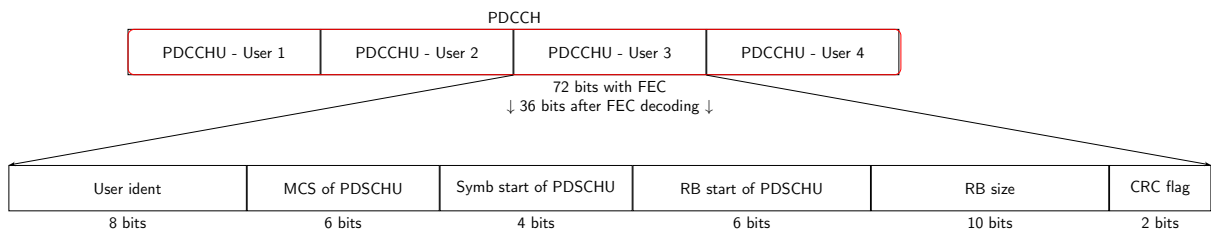


Figure 1.12: PDCCHU informations

- The user ident is the same as the one of the PBCH. It is there to be sure we decode the correct PDCCH
- the MCS of the PDSCH is on 6 bits. It has various values, associated to different constellation format and different FEC format. The different supported configurations are depicted on Table 1.3 et 1.3.3

Rate   Constellation	BPSK	QPSK	16-QAM	64-QAM	256-QAM
1/3	0	1	2	3	4
<b>1/2</b>	5	6	7	8	9
2/3	10	11	12	13	14
3/4	15	16	17	18	19

Table 1.3: Rate and Constellation for different values of MCS

- Using Convolutional Code (CC)
- Using Hamming

Rate   Constellation	BPSK	QPSK	16-QAM	64-QAM	256-QAM
1/3 (Hamming124)	20	21	22	23	24
<b>1/2 (Hamming 748)</b>	25	26	27	28	29
2/3 (Hamming 128)	30	31	32	33	34
3/4 (Hamming 2416)	35	36	37	38	39

Table 1.4: Rate and Constellation for different values of MCS

In the project, the user will be mainly with Hamming748 and CC1/2.

- The position of the PDSCH, with the symb start and the RB start, as described in Section 1.1.3.
- The RB size: this is the allocation in terms of RB of the PDSCH. The payload will occupy all the RB between the starting position and the ending position (start position + RB size).
- The Cyclic Redundancy Check (CRC) flag. A CRC is added on the PDSCH to be sure that the payload is correctly decoded. Several CRC can be added and the different polynoms are described in Table 1.3.3.

CRC flag	CRC size	Polynom used
0	8	$1 + X + X^2 + X^8$
1	16	$1 + X^2 + X^{15} + X^{16}$
2	24	$X^{24} + X^{22} + X^{20} + X^{19} + X^{18} + X^{16} + X^{14} + X^{13} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^3 + X + 1$
3	32	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

### 1.3.3.1 PDSCH

Finally, here we are: the PDSCH is the channel that bears the usefull information. All symbols here are following the mapping defined in the PDCCHU and the information is protected by a CRC

- The symbol sequence is extracted based on the information from the PDCCHU:
  - With MCS (constellation size  $q$ ) FEC used with the coding ratio ( $r$ ) and crc size  $c$  defined in PDCCHU.
  - Binary size (coded) is  $N_{Fc} = nbRB \times 12 \times \log_2(q)$ .

- Payload bitsize (before coding)  $N$  is obtained from binary size  $\times r$ .
- Sometimes, Padding bits have to be added (if  $N_c$  cannot be produced from  $N$  bits by the FEC):
  - \* If it is possible, we have  $N_c = N_{Fc}$ .
  - \* Otherwise,  $Z$  padding bits are added and  $N_c = N - Z$ .
  - \* If so,  $Z$  is calculated to be minimized.
  - \* ! For Convolutional code, tail of code (Output is  $N_c = \frac{1}{r} \times N + 16$ ).
- Coded payload is obtained as follow:
  - \* Content is created with size  $N - c$ .
  - \* CRC is appended to this data.
  - \* Data is fed to the FEC to produce  $N_c$  bits.
  - \* Eventually, potential padding bits are added.
- Finally a bitstream is obtained
  - This bitstream in the project has to be converted into a ASCII sequence using the appropriate Python functions

## Chapter 2

# Wireless Lab

The project will be done under Python. Most of the functions will have to be coded and some functions will be provided. The transmitter and receiver description are described in the previous chapter. When necessary, the reader should have a deep look to the appropriate parts.

We emulate in this section a small network where the base station provided a broadband service to several users. Each group of student is associated to a device (i.e a mobile terminal) and has a unique identifier. This identifier will be used to extract the dedicated payload content from the multiplexed stream. The *ultimate* purpose is to decode the sent text and discover the secret key that has been hidden for each user in the frame. In the proposed lab, we will focus on the L2 part: we will suppose that we will have a time frequency grid from the low layer (L1) and that this matrix has been obtained from an over the air transmission. The purpose is to recover the payload associated to your group identifier and discover the secret key hidden in the payload content.

To do so, we will have to decode all the channels that are multiplexed in the received LTE frame. Each channel has a specific information that will be used to sequentially obtain all the necessary flags and parameters for the final payload decoding. We will first have to build the necessary functions to recover all the parameters, and then apply the successive decoding functions.

### Code warning

The LTE channels are not static, their size and their parameters evolve by time (associated to network load and link quality). So your code should be !

- Clean your code and use functions when needed
- Always use variables and parameters: nothing should be hard-coded
- Use structures or classes to have a clean nested code, especially for function calls.
- Use Jupyter if you want to put all the code in a clean netbook.

### Python dependencies

Most of the code will be hand crafted but we will need some Python packages to speed up the code.

- related to numpy to manage Time / Frequency matrixes
- related to matplotlib to display images
- related to unit testing
- Package math for special functions
- Package scikit for FEC convolutional decoder

To load all the dependencies (after being sure there are installed through pip), you can use the following Python code

```
1 import numpy as np
2 import math
3 # Module to display T/F matrix
4 import matplotlib.pyplot as plt
5 from matplotlib import ticker, cm
6 # Module for Unit testing
7 import pytest
8 # Module for convolutionnal decoder
9 import sk_dsp_comm.fec_conv as fec
```

## 2.1 Extraction of the time frequency matrix

1. Load the CSV datafile that embed the ideal time frequency matrix. This time frequency is a matrix of complex symbols that bears all the channels for all the different users. The matrix is stored as a CSV file and can be loaded with the following command

```
1 my_data = np.genfromtxt('./tfMatrix.csv', delimiter=';')
2 mat_complex = my_data[:,0::2] +1j*my_data[:,1::2]
```

Note the second line that merges the real and the imaginary parts of the signal.

- What is the size of the matrix ? Why such dimensions ?
- We need to recover only the useful part, as some of the subcarriers is not allocated in OFDM. From the subcarrier mapping described in Section 1.1.2, obtain the submatrix with only the allocated subcarriers. Explain what are the new dimensions of this matrix.

```
1 # --- Getting allocated subcarriers
2 tfMatrix_short = tfMatrix[?,?];
```

- Have a look on the matrix power distribution. This can be done with the use of contourf matplotlib function. Use the following code and complete the missing parts related to the Figure informations.

```
1 def powerDistributionGraph(Z):
2     """
```

```

3     Draw the power distribution graph
4     """
5     fig, ax = plt.subplots()
6     cs = ax.contourf(np.linspace(0, len(Z[0]), len(Z[0])), np.linspace(0,
7         len(Z), len(Z)), Z)
8     cbar = fig.colorbar(cs)
9     ax.set_title('??')
10    ax.set_xlabel('??')
10    ax.set_ylabel('??')

```

Remark the 4 main area. The yellow and orange corresponds to the synchronisation channels, the green to the control channels and the black area corresponds to the payload content. Note that we use here the classic signal processing view for the T/F and thus the image is transposed with respect to the size of the matrix we have.

### T/F matrix and transposes

Note that in classic signal theory, time frequency matrices are often displayed with the transpose as we conversely set the x axis as the time domain and the y as frequency (subcarriers). As Python is row based indexing, it is however simpler here to compute each row as symbols and thus represent the T/F matrix with symbols on y axis.

The synchronisation channels (see Section 1.3.1) are used for synchronisation and also for channel estimation. They will be not be used in the first part of the lab, and can be (for the moment) dropped. As stated in the norm description, they occupy the two first OFDM symbols. Create the shorter matrix with the useful channels (namely the PBCH, PDCCH and PDSCH) with the following command

```

1  # --- Removing PSCH and SSCH channels
2  qamMatrix = tfMatrix_short[:,?]

```

Now, we need to decode the first channel that bears information: the PBCH.

### Conclusion

At this stage, you should have the following elements

- A working Python environment
- 3 defined Matrix with different sizes

## 2.2 PBCH decoding

Carefully read the section related to the PBCH description in Section 1.3.2. As stated in this section, the PBCH has a fixed format: the symbol are BPSK and the FEC used is a Hamming748

### 2.2.1 BPSK decoding

1. Check that the first symbols associated to the PBCH belongs to a BSPK (or 2-QAM).

2. Create a Python function that demodulates a BPSK stream into a binary stream. The function should have the following template

---

```
1 bitSeq = bpsk_demod(qamSeq)
```

---

Check that the function gives the correct output. This can be done by defining a unit test. For BPSK, this is done by defining the `test_bpsk` function defined in `tests_modulation.py`. Open the function and have a look on the source code. Extract the function and call it in your workspace.

3. If you have a look on Section 1.3.2, the PBCH have a variable size, depending on the number of allocated users. We need first to decode the 48 bits to extract the cell and users information. From the `qamMatrix` variable, create the symbol stream associated to the first 48 bits, and decode the bitstream.

### 2.2.2 Hamming748 decoder

We now have a bitstream of 48 elements that contains the cell and user information. This information is protected with the use of a Hamming748 coding system, as explained in Section 1.3.2. We need to create a Hamming decoding method.

1. Propose a Python function `hamming748_decode` that returns a decoded binary sequence that have been coded with a Hamming coder. The function should follow the following template:

---

```
1 # --- Calling Hamming decoding function
2 bitDec = hamming748_decode(bitSeq)
```

---

2. Check that your method is OK with its associated unitary test defined in `tests_hamming.py`

### 2.2.3 PBCH decoding

We can now recover the useful information.

1. Create a dictionary associated to the first slot of the PBCH decoding that contains the useful information (cell size and number of users). What is the cell ident ? How many users are allocated in our network ? To convert a binary information into a value readable for humans, you can use the given function `bin2dec`

---

```
1 def bin2dec(nb):
2     """
3     Transform a binary list to an integer
4     """
5     n = "0b"
6     for b in nb:
7         n = n + str(b)
8     return int(n, 2)
```

---

2. Deduce the size of the full PBCH channel

3. Proceed to decode all the PBCHU. Keep only the one associated to your user ident.
4. Create a dictionary associated to the PBCHU of interest (i.e your PBCHU).

### Conclusion

At this stage you should know the following elements:

- A functional BPSK demodulator and Hamming748 decoder
- The cell ident
- The number of users that are allocated
- If you are allocated in the proposed frame
- Where is located your PDCCHU
- What MCS your PDCCHU uses

## 2.3 PDCCH decoding

We want now to recover the information contains in the PDCCH. Read carefully the section related to the PDCCH channel. The PBCH contains the location of each PDDCHU and we will decode only the PDCCHU of interest. This channel uses a specific channel coder and a specific MCS. You can recover the link between the MCS flag (in your PBCHU) and the chosen FEC/MCS in Table 1.2.

### 2.3.1 MCS decoding

In this lab we will only use the Hamming748 system for our FEC. As a consequence, there is no need to build a specific function as the FEC is exactly the same as the one used in the PBCH. On the contrary, if the PBCH only uses BPSK, the PDCCHUs can use both BPSK and QPSK. We thus need to define a QPSK demodulation function.

1. Create a template function `qpsk_demod`
2. Form the Voronoi region defined in Section 1.2.2, propose and code a QPSK demodulation function.
3. Check that your method works with the function `test_qpsk`.
4. Built a function that can both decode a PDCCHU stream depending on the MCS flag value. This function should call the 2 decoding function, related to BPSK and QPSK decoding.

### 2.3.2 PDCCHU decoding

We can now recover the stream.

1. Extract the QAM sequence of the appropriate length to recover the PDCCHU at the correct position in the TF grid



2. Decode the sequence using the correct format
3. Extract the information and create a Python dictionary that gather all the useful information.

### Conclusion

At this stage you should have the following element

- The position of the user PDSCH
- The RB allocation of the PDSCH
- The MCS used in the PDSCH
- the CRC type used in the PDSCH

and we are now ready to decode the payload content.

## 2.4 PDSCH decoding

We can now decode the payload content. We first have to proposed some additional function to be able to decode higher MCS, other FEC types and be able to use the CRC to decode the data.

### 2.4.1 FEC and MCS overall functions

We use a specific MCS for the PDSCH. This MCS leads to the support of

- Higher QAM format. In the project we only support BPSK, QPSK, and 16QAM. Support of 64QAM and 256QAM will not be implemented.
  - Use of CC FEC in addition to the Hamming748 FEC.
1. We provide the 16-QAM decoding function. Check that the test `test_qam16.m` passes.
  2. build a function `PBCHU_demod(qamSeq,mcs)` which apply the appropriate constellation demodulation based on the MCS flag obtained from the PDCCHU.
  3. Built a function `PBCHU_fec(qamSeq,mcs)` which apply the appropriate FEC decoder based on the associated MCS flag
  4. Built a function `PBCHU_crc(qamSeq,crcSize)` that uses the provided CRC functions and that apply the appropriate CRC check to the PDSCH data. The CRC decoding functions are provided and you will have to use `crcDecode` and `getCRCPoly`. Test with the unitary test function `test_crcDecode`.

### 2.4.2 PDSCH decoding

We are now ready to decode our PDSCH channel and recover the payload data.

1. Extract the symbol sequence related to the PDSCH

2. Apply the Symbol demodulation to recover the bit sequence
3. Apply the appropriate FEC decoder scheme. For the Convolutional decoder you can use the proposed decoding function from the python module (see the convolutional description to create c1)

---

```
1 dec = cc1.viterbi_decoder(np.array(qamSeq).astype(int), 'hard')
```

---

4. You Now have the payload ! Check its integrity with the use of the CRC check. The CRC has then to be removed from the payload content.
5. Recover the ASCII sequence. The message is protected by a encrypted system. As this is out of scope of the project, the decoding method is given in the file `binary_transformation.py`. The decoding be done with the use of the provided functions (`bitToByte` then `cesarDecode` and finally `toASCII`). For `cesarDecode`, you will need your user identifier.

---

```
1 # Assuming the message decoding from QPSK or QAM16 is qamSeq
2 # Convert the binary sequence into bytes
3 mess = bitToByte(seq)
4 # Bytes are "encrypted", uncrypt them
5 real_mess = cesarDecode(USER,mess); # USER is your user group
6 final_mess = toASCII(real_mess)
```

---

## Conclusion

The project is now terminated. From the T/F matrix you should now have

- Decoded the payload associated to your user
- Print the payload text into the Python terminal
- Discover the secret key !

Some last things before the end:

- Redo the decoding steps with the use of `tf_matrix_2.mat`. What is the new key ?
- Redo the decoding steps with the use of `tf_matrix_3.mat`. What happens ? What can we do to address this issue ?

# Appendix

## Glossary

5G-NR	5G New Radio
ADC	Analog to Digital Converter
BPSK	Binary Phase Shift Keying
BS	Base station i.e transmitter
CAZAC	Constant Amplitude Zero Autocorrelation: sequence used in synchronisation
CRC	Cyclic Redundancy Check
eNobeB	Mobile receiver
CP	Cyclic prefix
DAC	Digital to Analog Converter
FEC	Forward Error Corrector
FFT	Fast Fourier Transform
LTE	Long Term Evolution: current studied standard
MCS	Modulation and coding scheme
OFDM	Orthogonal Frequency Division Multiplexing
PBCH	Physical Broadcast Channel: Channel that have base station information
PDCCH	Physical Downlink Control Channel, channel with user information
PDSCH	Physical Downlink Shared Channel, channel with user payloads.
QAM	Quadrature Amplitude Modulation: modulation format used in channels
RB	Resource Block: group of subcarriers which corresponds to minimal allocation
Rx	Receiver
T/F	Time Frequency Grid
Tx	Transmitter
ZC	Zadoff-Chu sequence, special sequence used in synchronisation
ZF	Zero Forcing, strategy used for channel equalisation

# Bibliography

- [3GP09] 3GPP. Physical channels and modulation, 3GPP TS 36.211. V10, 2, 2009.
- [And12] Iryna Andriyanova. Cours 7: Protocoles ARQ/HARQ - communications sans fil. Lesson M2 ISIM, 2012.
- [BM96] Sergio Benedetto and Guido Montorsi. Design of parallel concatenated convolutional codes. *IEEE Transactions on Communications*, 44(5):591–600, 1996.
- [CS82] J Conway and N Sloane. Voronoi regions of lattices, second moments of polytopes, and quantization. *IEEE Transactions on Information Theory*, 28(2):211–226, 1982.
- [For73] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.
- [GML14] Malik Muhammad Usman Gul, Xiaoli Ma, and Sungeun Lee. Timing and frequency synchronization for ofdm downlink transmissions using zadoff-chu sequences. *IEEE Transactions on Wireless Communications*, 14(3):1716–1729, 2014.
- [Gol05] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.
- [Ham50] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.
- [HOP96] Joachim Hagenauer, Elke Offer, and Lutz Papke. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on information theory*, 42(2):429–445, 1996.
- [PS01] John G Proakis and Masoud Salehi. *Digital communications*, volume 4. McGraw-hill New York, 2001.
- [Vit98] Andrew J. Viterbi. An intuitive justification and a simplified implementation of the map decoder for convolutional codes. *IEEE Journal on Selected Areas in Communications*, 16(2):260–264, 1998.
- [ZM07] Jim Zyren and Wes McCoy. Overview of the 3GPP long term evolution physical layer. *Freescale Semiconductor, Inc., white paper*, 7:2–22, 2007.