Réception-Totale-Épuré-BOUVIER

December 12, 2025

1 Pramétrage de l'ADALM PLUTO pour la réception

[29]:
```
# Importation des bibliothèques à compléter ici
import adi                                  # communication avec Adalm Pluto (AP)
import numpy as np                          # gestion des tableaux en Python
import matplotlib.pyplot as plt             # affichage de graphiques

sdr = adi.Pluto("ip:192.168.3.1")      # adresse relevée dans le fichier config
print(sdr) # affichage des paramètres par défaut de réception (rx) et␣
 ↪d'émission (tx) de l'A-P

# CHoix des paramètres
Freq_osc_local = 433e6              # la fréquence porteuse théorique du␣
 ↪signal à observer
Freq_PasseBas  = 5e6                # la largeur du filtre de réception

Freq_ech       = 10e6               # la fréquence d'échantillonnage du signal␣
 ↪reçu
print(" ")

# choix de la fréquence d'échantillonnage : conséquence sur la taille mémoire␣
 ↪utilisée
Duree_Ech  = 0.3          # Choix de la durée de temps pendant lequel on veut␣
 ↪échantillonner, théoriquement en seconde

Tech = 1/Freq_ech          # CALCULER la période d'échantillonnage
print("La période d'échantillonnage est de      : ",   round(Tech*1000000,3)    ␣
 ↪     , "µs")

Taille_Buffer = Duree_Ech * Freq_ech       # CALCULER la taille mémoire␣
 ↪nécessaire à cet échantillonnage
print("La taille de l'échantillon capté sera de : " , Taille_Buffer/1000, "k␣
 ↪Octets")
print(" ")

# Affectation des paramètres à l'A-P
sdr.rx_lo           = int(Freq_osc_local)
```

```
sdr.rx_rf_bandwidth = int(Freq_PasseBas)

sdr.sample_rate     = int(Freq_ech)

sdr.rx_destroy_buffer()                              # (voir https://pysdr.org/fr/
 ↪content-fr/pluto.html)
sdr.rx_buffer_size  =  int(Taille_Buffer)

sdr.gain_control_mode_chan0 = "slow_attack"
gain = 0
# ou bien
#sdr.gain_control_mode_chan0 = "manual"   # désactivation du controle␣
 ↪automatique de gain (AGC); Ainsi on pourra, à la ligne suivante, fixer le␣
 ↪gain.
#gain = 70.0  # dB                          la gamme permise est de 0 à 74.5 dB
#sdr.rx_hardwaregain_chan0 = gain           # puisque l'AGC est désactiver il␣
 ↪faut affecter une valeur au gain de l'ampli de réception
```

```
Pluto(uri="ip:192.168.3.1") object "PlutoSDR" with following key properties:

rx_lo:                   432.999998   MHz, Carrier frequency of RX path
rx_hardwaregain_chan0    73           dB, Gain applied to RX path. Only
applicable when gain_control_mode is set to 'manual'
rx_rf_bandwidth:         5.0          MHz, Bandwidth of front-end analog filter
of RX path
gain_control_mode_chan0: slow_attack  Receive path AGC Options: slow_attack,
fast_attack, manual

tx_lo:                   2449.999998  MHz, Carrier frequency of TX path
tx_hardwaregain_chan0:   -10          dB, Attenuation applied to TX path
tx_rf_bandwidth:         18.0         MHz, Bandwidth of front-end analog filter
of TX path
tx_cyclic_buffer:        0            Toggles cyclic buffer

filter:                  [128.0, 4.0, 128.0, 4.0] FIR filter file
sample_rate:             10.0         MSPS, Sample rate RX and TX paths
loopback:                0            0=Disabled, 1=Digital, 2=RF



La période d'échantillonnage est de      :  0.1 µs
La taille de l'échantillon capté sera de :  3000.0 k Octets
```

2 Acquisitions

2.1 Série d'acquisitions

[18]:
```
import time
# la précision du temps d'exécution de la boucle :
Duree_Obs = 20        # choix en secondes
nbr_acq   = Duree_Obs/Duree_Ech         # combien faire d'itération à la boucle
print(nbr_acq)

print("ATTENDRE que le processus soit terminé.")

# La boucle et ses initialisations

debut=time.time()

PuissancesdBm = []
PrdBm = 0

for i in range (int(nbr_acq)) :
    Donnees = sdr.rx()
    Signal_Recu = abs(Donnees/(2**12))
    PrW   = np.average((Signal_Recu**2)/1)
    PrdBm = 10*np.log10(PrW/0.001)
    PuissancesdBm.append(PrdBm)

    print(".", end='')  # met 1 point(.) par itération afin de voir la␣
 ↪progression. "end='"' permet de mettre en ligne

fin = time.time()
print("C'est fini !")

print ("La durée totale de la boucle était finalement de : ", fin - debut ,␣
 ↪"secondes")
Duree_Calc_reel = (fin - debut)/len(PuissancesdBm)
#print(Duree_Calc_reel)
```

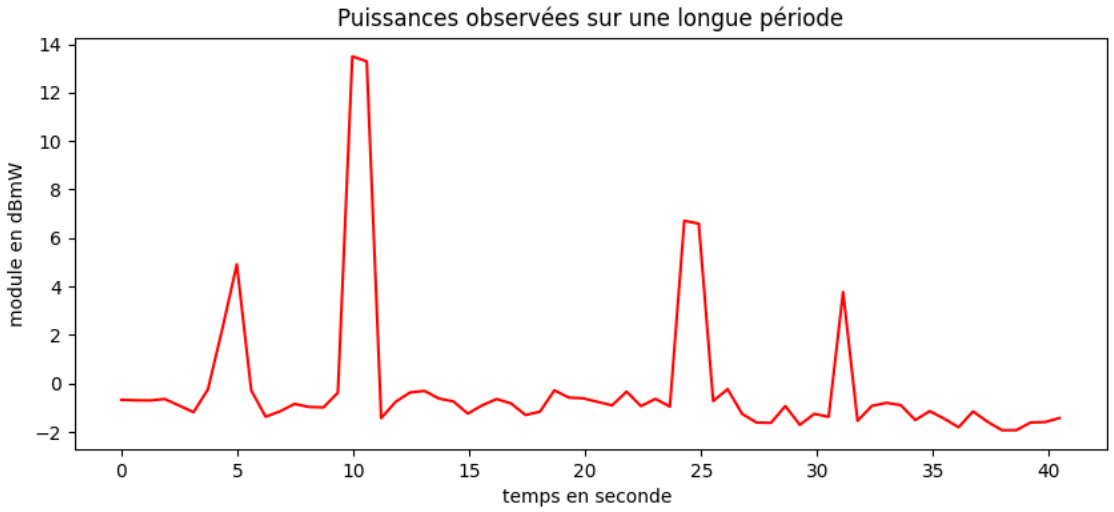
```
66.66666666666667
ATTENDRE que le processus soit terminé.
…C'est fini !
La durée totale de la boucle était finalement de :  40.456583976745605 secondes
```

2.2 Affichage de la série d'acquisition

[19]:
```
ech_temps = np.linspace(0 , len(PuissancesdBm)*Duree_Calc_reel ,␣
 ↪len(PuissancesdBm))
deffig = plt.figure(figsize=(10,4)) # dimensionne le tracé
```

```
plt.plot( ech_temps, PuissancesdBm , color='red')
plt.title("Puissances observées sur une longue période")
plt.xlabel("temps en seconde ")
plt.ylabel("module en dBmW")
```

[19]: Text(0, 0.5, 'module en dBmW')



2.3 Exploitation des acquisitions

[20]:
```
# Ce relévé peut se faire automatiquement pour la valeur maximale de la puissance :
Pmax = np.max(PuissancesdBm)
print("La puissance maximale observée est de :" , round(Pmax, 3) , "dBm")

Pmoyenne = np.average(PuissancesdBm)
print("La puissance moyenne (proche du bruit) est de :" , round(Pmoyenne, 3) , "dBm")

Pcomp = (Pmax + Pmoyenne)/2
print("Le niveau de détection peut etre choisi à :" , round(Pcomp, 3) , "dBm")
```

```
La puissance maximale observée est de : 13.496 dBm
La puissance moyenne (proche du bruit) est de : -0.128 dBm
Le niveau de détection peut etre choisi à : 6.684 dBm
```

2.4 Acquisitions d'une température

[90]:
```
import time
#Freq_ech       = 10e9

Pcomp = 0
PrdBm = Pmoyenne - 50      # initialisation avec une valeur que l'on est␣
 ↪certain de ne jamais atteindre !
compteur = 0
Duree_Ech  = 0.3
niveau_comp = Pcomp   # niveau intermédiaire entre le bruit et le signal de la␣
 ↪figure précédente
print("ATTENDRE que le processus soit terminé.")

Duree_Obs = 20        # choix en secondes
nbr_acq   = Duree_Obs/Duree_Ech         # combien faire d'itération à la boucle
print(nbr_acq)

PuissancesdBm = [] #tableau avec les puissances
Sig_normalisé = [] #tableau avec le signal normalisé
condition_passee = False
debut=time.time()

for i in range (int(nbr_acq)) :
    Donnees = sdr.rx()
    Signal_Recu = abs(Donnees/(2**12)) #on garde uniquement la partie absolue
    Signal_Recu_complexe = Donnees/(2**12) #on garde le signal complexe
    PrW   = np.mean(Signal_Recu**2)/1
    PrdBm = 10*np.log10(PrW/0.001)
    #print(PrdBm)
    if PrdBm > niveau_comp :
        PuissancesdBm.append(Signal_Recu) #on ajoute la puissance au tableau␣
 ↪des puissances
        Sig_normalisé.append(Signal_Recu_complexe) #on ajoute le signal␣
 ↪complexe au tableau pour la représentation fréquentielle
        print("!", end='')
        compteur += 1
        fin = time.time()
        break
    else:
        print(".", end='')
        compteur += 1
        #Sig_normalisé.append(Signal_Recu_complexe)

print("Fin : Capture réalisée en : " , compteur , "itérations")
```

```
PuissancesdBm_list = [item for array in PuissancesdBm for item in array] #on␣
 ↪ajoute toutes les valeurs contenu dans les np.array dans une unique liste␣
 ↪python
print(PuissancesdBm_list[:10])

print ("La durée totale de la boucle était finalement de : ", fin - debut ,␣
 ↪"secondes")
Duree_Calc_reel = (fin - debut)/compteur
print(Duree_Calc_reel)
```

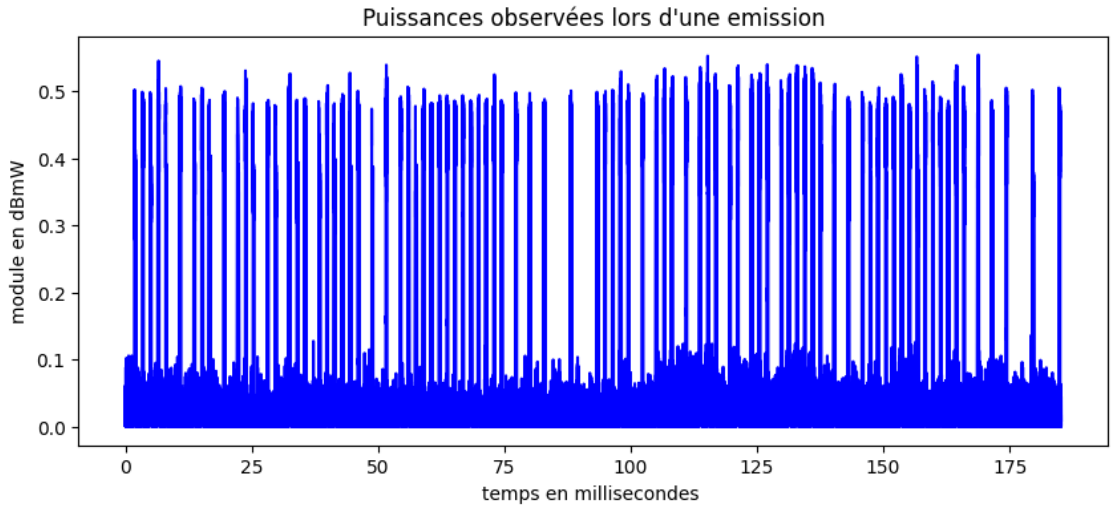
```
ATTENDRE que le processus soit terminé.
66.66666666666667
!Fin : Capture réalisée en :  1 itérations
[0.4243971700593758, 0.43940598976817186, 0.46096084018386413,
0.4440187768531311, 0.4121510272197332, 0.4163932698071452, 0.43698205261126205,
0.463452862866266, 0.409056506131222, 0.4359246911795716]
La durée totale de la boucle était finalement de :  0.7641487121582031 secondes
0.7641487121582031
```

2.5 Sa représentation temporelle

[39]:
```
# Echelle verticale :
Signal_Utile  = PuissancesdBm_list

# Echelle horizontale : fabrication de cette échelle
print(len(PuissancesdBm_list))
ech_temps =  np.linspace(0 , len(PuissancesdBm_list)*Duree_Calc_reel/10000 ,␣
 ↪len(PuissancesdBm_list))

# Le tracé
deffig = plt.figure(figsize=(10,4)) # dimensionne le tracé

plt.plot( ech_temps, PuissancesdBm_list , color='blue')
plt.title("Puissances observées lors d'une emission")
plt.xlabel("temps en millisecondes ")
plt.ylabel("module en dBmW")
```

```
3000000
```

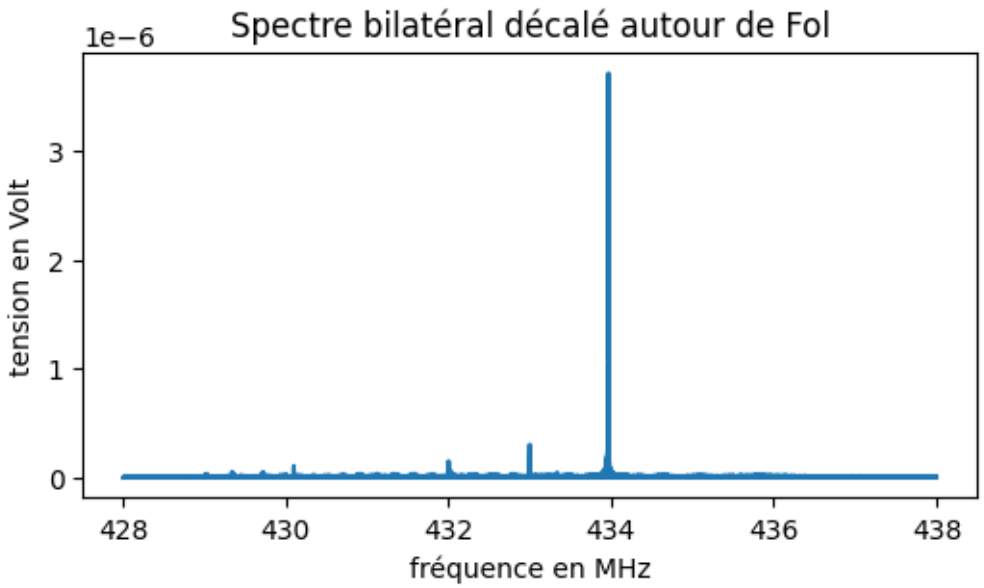
[39]:
```
Text(0, 0.5, 'module en dBmW')
```



2.6 Représentation

[40]:
```
#On met tous les np.array dans un seul np.array
Sig_normalisé_simple = np.concatenate(Sig_normalisé)

# Echelle verticale : les tensions reçues
spectre1 = np.fft.fft(Sig_normalisé_simple/2**12)              # la fonction F.F.
 ↪T. sous numpy
spectre2 = np.fft.fftshift(spectre1)             # pour réarranger les␣
 ↪échantillones dans l'ordre
spectre  = np.abs(spectre2)/len(Sig_normalisé_simple)          # le module (pour␣
 ↪ne pas prendre la phase)

# Echelle horizontale : fabrication de cette échelle
DebutSpectre = Freq_osc_local - Freq_ech/2           # choix du début du␣
 ↪spectre à regarder
FinSpectre   = Freq_osc_local + Freq_ech/2           # choix de la fin du␣
 ↪spectre à regarder

ech_freq = np.linspace( DebutSpectre , FinSpectre , len(Sig_normalisé_simple))

# Le tracé
deffig = plt.figure(figsize=(6,3)) # dimensionne le tracé
plt.plot(ech_freq/1000000 , spectre)
plt.title("Spectre bilatéral décalé autour de Fol")
plt.xlabel("fréquence en MHz")
plt.ylabel("tension en Volt")
```

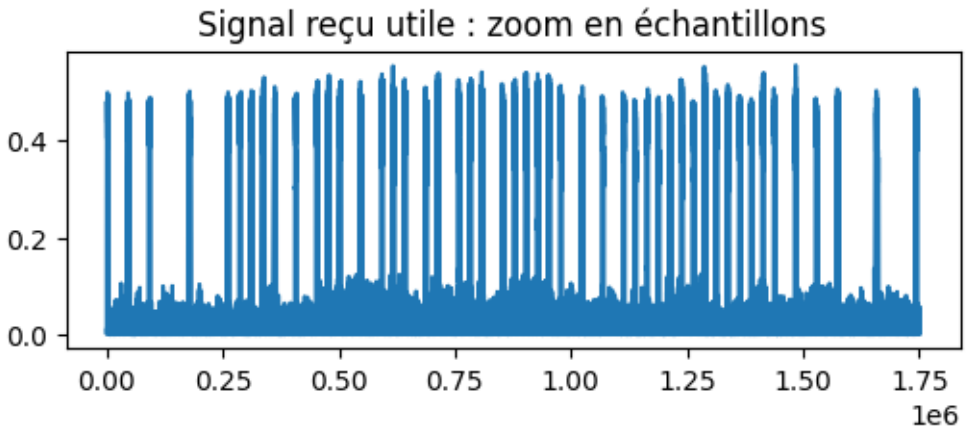
[40]:
```
Text(0, 0.5, 'tension en Volt')
```



3 Zoom sur une trame

[64]:
```
# réalisation d'un zoom en échantillons
t_debut = 1250000      # choix en fonction de la figure précédente
t_fin   = 3000000    # choix en fonction de la figure précédente

visualisation = PuissancesdBm_list[int(t_debut) : int(t_fin)]

deffig = plt.figure(figsize=(6,2)) # dimensionne le tracé
plt.plot(visualisation)
plt.title("Signal reçu utile : zoom en échantillons")
```

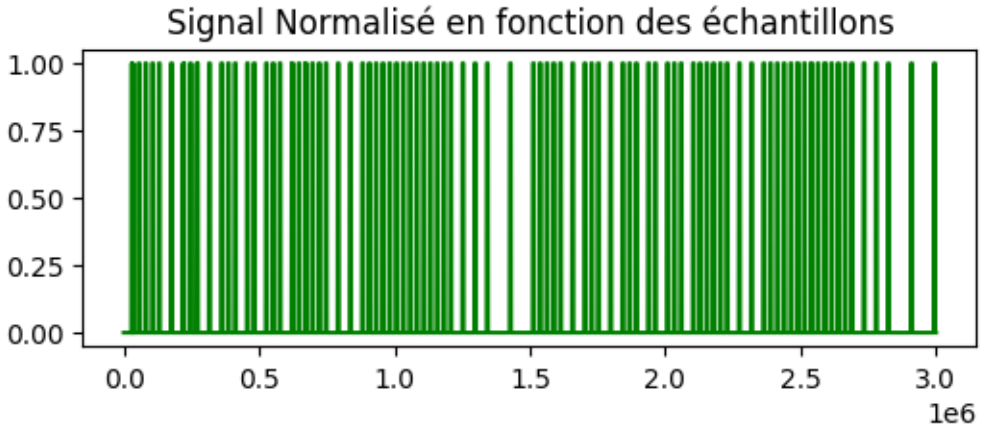
[64]:
```
Text(0.5, 1.0, 'Signal reçu utile : zoom en échantillons')
```



4 Mise en forme du signal

[66]:
```
#  Sur la totalité des données

Signal_Norma = []

for x in PuissancesdBm_list:
    if x > 0.3:
        Signal_Norma.append(1)
    else:
        Signal_Norma.append(0)

print(Signal_Norma [ 500 : 550])  # observation de quelques résultats

deffig = plt.figure(figsize=(6,2)) # dimensionne le tracé
plt.plot( Signal_Norma , color='green')
#plt.plot( Signal , color='blue')

plt.title("Signal Normalisé en fonction des échantillons")
print("")
```

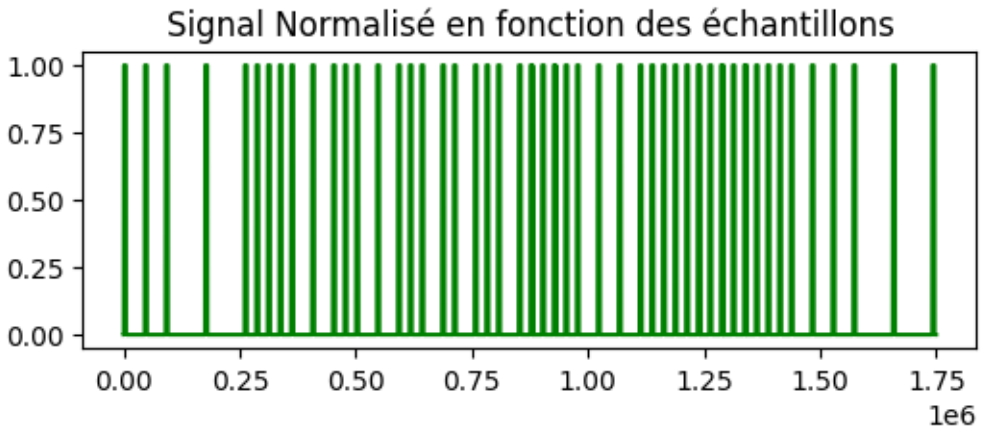
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```



[67]:
```
#  Sur une partie des données : données réduites à une trame large

Signal_Norma_reduit = []
```

```
for x in visualisation:
    if x > 0.3:
        Signal_Norma_reduit.append(1)
    else:
        Signal_Norma_reduit.append(0)

print(Signal_Norma_reduit [ 500 : 550])  # observation de quelques résultats

deffig = plt.figure(figsize=(6,2)) # dimensionne le tracé
plt.plot( Signal_Norma_reduit , color='green')
#plt.plot( visualisation , color='blue')

plt.title("Signal Normalisé en fonction des échantillons")
print("")
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```



4.1 Comptage des états

[68]:
```
A_Compter  =  Signal_Norma_reduit

compteur_0 = 0
compteur_1 = 0
bit_precedent = 2     # une valeur qui ne soit ni 0 ni 1

tableau_longueurs = []

for i in A_Compter:
```

```
    if i == 0 and bit_precedent == 0:
        compteur_0 += 1
    elif i == 0 and bit_precedent != 0:
        compteur_0 += 1
        tableau_longueurs.append(compteur_1)
        bit_precedent = 0
        compteur_0 = 0
    elif i == 1 and bit_precedent == 1:
        compteur_1 += 1
    elif i == 1 and bit_precedent != 1:
        compteur_1 += 1
        tableau_longueurs.append(compteur_0)
        bit_precedent = 1
        compteur_1 = 0


print(tableau_longueurs)
```

```
[0, 215, 4717, 40329, 4708, 40340, 0, 0, 4672, 80333, 4740, 80352, 0, 0, 4742,
20326, 0, 0, 4684, 20324, 4696, 20306, 4713, 20300, 4717, 40352, 4686, 40320,
4717, 20299, 1, 0, 4449, 0, 19, 0, 75, 0, 6, 0, 9, 0, 148, 20323, 4696, 40318,
4716, 40331, 0, 0, 4707, 20319, 4699, 20310, 4709, 40338, 0, 0, 4702, 20317,
4696, 40317, 0, 3, 0, 0, 4585, 0, 127, 20293, 4726, 20307, 4709, 40353, 4, 0,
4681, 20308, 4089, 0, 107, 0, 5, 0, 9, 0, 14, 0, 18, 0, 1, 0, 23, 0, 4, 0, 67,
0, 11, 0, 95, 1, 55, 0, 47, 0, 78, 0, 29, 0, 9, 0, 14, 20303, 0, 0, 4719, 20304,
0, 0, 3942, 0, 84, 0, 54, 0, 11, 0, 122, 0, 42, 0, 6, 0, 238, 0, 16, 0, 183,
20286, 1, 2, 4722, 20298, 4351, 0, 175, 0, 78, 0, 108, 40332, 4706, 40315, 2, 0,
4724, 40370, 4660, 20310, 4716, 20305, 4708, 20304, 1, 0, 4711, 20326, 4694,
20299, 3690, 0, 63, 0, 98, 0, 27, 0, 13, 1, 132, 0, 21, 0, 72, 0, 128, 0, 60, 0,
109, 0, 6, 0, 24, 0, 42, 0, 42, 0, 33, 0, 127, 20294, 4733, 20285, 0, 2, 3678,
0, 39, 0, 6, 0, 127, 0, 73, 0, 160, 0, 9, 0, 33, 0, 24, 0, 12, 0, 16, 0, 15, 0,
15, 0, 492, 20326, 4691, 20304, 3607, 0, 590, 0, 513, 20338, 4683, 20299, 4719,
20315, 0, 0, 4705, 20312, 4609, 0, 96, 40310, 4722, 40318, 4718, 40348, 4669,
80366, 4707, 80342, 4753]
```

5 Binarisation des données

[69]:
```
Donnees_Binaires = []     # initialisation

for i in tableau_longueurs:
    if i < 23000 and i > 19000: #état bas court
        Donnees_Binaires.append(0)
    elif i < 41000 and i > 38000: #état bas long
        Donnees_Binaires.append(1)
    elif i < 81000 and i > 79000: #état bas très long (synchro)
        Donnees_Binaires.append("Synchro")
```

```
print(Donnees_Binaires)
```

```
[1, 1, 'Synchro', 'Synchro', 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
'Synchro', 'Synchro']
```

5.1 Sélection d'une trame uniquement

[74]:
```
Une_Trame_Bin = []
compteur_synchro = 0

for i in range (0 , len(Donnees_Binaires)) :
    if Donnees_Binaires[i] == "Synchro" :
        compteur_synchro = compteur_synchro + 1

    if compteur_synchro == 2 :
        Une_Trame_Bin.append(Donnees_Binaires[i])

print(Une_Trame_Bin)

# en supprimant l'identifiant de synchro
La_Trame_Bin = Une_Trame_Bin[1:len(Une_Trame_Bin)]                    # pour ne␣
 ↪pas prendre le "synchro"

print("La trame avec laquelle on va travailler est: ", La_Trame_Bin)
print("La longueur d'une trame est de : ", len(La_Trame_Bin), "Bits")

y = 0

for i in range (0 , len(La_Trame_Bin)) :
    bit_i = La_Trame_Bin[i]              # juste un élément d'écriture
    bit_position = bit_i << (41-i)       # place le premier bit extrait de la␣
 ↪trame en position 41 (début de chiffre)
                                         # et ainsi de suite
    y = y | bit_position                 # réalisation d'un OU entre la␣
 ↪précédente valeur de ma données et la nouvelle
                                         # ceci permet de la rajouter !
    #print(i,41-i,La_Trame[i],bin(y))

LA_TRAME = y
print("La trame étudiée s'écrit : ", LA_TRAME, hex(LA_TRAME), bin(LA_TRAME))
```

```
['Synchro', 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1]
La trame avec laquelle on va travailler est:  [0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0,
0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1]
```

```
La longueur d'une trame est de :  42 Bits
La trame étudiée s'écrit :  219731656711 0x3329070007
0b110011001010010000011100000000000000111
```

6 CRC

6.1 Récupération du CRC et passage

[77]:
```
# Récupération du deuxième 1/2 octet du dernier octet le code CRC
print(hex(0b1111))
Dernier_OctB = LA_TRAME & 0b1111
print("Le code CRC est :", bin(Dernier_OctB))
CRC = Dernier_OctB

# Remplacement du dernier octet de la trame par ce code CRC.
# Remarque : à partir de ce point la trame utilisée fera 42-8+4=38 bits

print("Trame initiale           : " , bin(LA_TRAME ))

TrameBinaire34 = LA_TRAME >> 8     # on supprime le dernier octet
print("Trame sans dernier octet : " , bin(TrameBinaire34))

TrameBinaire38 = TrameBinaire34 << 4   # on rajoute 4 bits pour préparer␣
 ↪l'arrivée du code  CRC
print("Trame 34 + 4 bits        : ", bin(TrameBinaire38))

TrameBinaire38crc = TrameBinaire38 | CRC   # on rajoute le CRC calculé
print("Trame avec CRC           : " ,bin(TrameBinaire38crc))
```

```
0xf
Le code CRC est : 0b111
Trame initiale           :  0b110011001010010000011100000000000000111
Trame sans dernier octet :  0b11001100101001000011100000000
Trame 34 + 4 bits        :  0b1100110010100100000111000000000000
Trame avec CRC           :  0b110011001010010000011100000000111
```

6.2 Vérification de l'acquisition

[78]:
```
dividende = TrameBinaire38crc     # pour avoir une boucle généraliste

LongTrame = 38                    # le nombre de bits de la séquence à diviser
LongDiv   = LongTrame - 4         # pour les 4 derniers bits la division est␣
 ↪différente (reste sur 4 bits)

polynome  = 0b10011               # le nombre de bits du polynome générateur
LongPoly  = 5
```

```
TestBPF = 0                          # initialisation d'une variable qui permet de␣
 ↪tester la valeur du bit de poids fort
TestBit = 0

for i in range (0 , LongDiv) :
       TestBit = 1 << (LongDiv - i - 1)                                          #␣
 ↪place un "1" logique au meme niveau de que le bit de poids fort du dividende
       TestBPF = dividende & TestBit                                             #␣
 ↪identification du bit de poid fort
       if TestBPF != 0:                                                          # si␣
 ↪le bit de poids fort est à 1
           polynome_decale = polynome << (LongDiv - i - LongPoly)                # on␣
 ↪décale le  polynome à gauche
           dividende = dividende ^ polynome_decale                               #␣
 ↪réalisation de la division/OUEX si bit de poid fort à 1
       else :                                                                    #␣
 ↪réalisation de la division/OUEX si bit de poid fort à 0 = aucune action
           dividende = dividende ^ 0b0                                           #␣
 ↪inutile mais permet de structurer !


Reste = (dividende)
if Reste == 0 :
    print("Le reste de cette division est = ", bin(Reste) , "Donc la␣
 ↪transmission est bonne !")

else :
    print("Le reste de cette division est = ", bin(Reste) , " ce qui n'est pas␣
 ↪égal à 0. Donc la transmission n'est PAS bonne ...")
```

```
Le reste de cette division est =  0b0 Donc la transmission est bonne !
```

7 Expression de la température

[89]:
```
#octet 1 = identifiant
id = hex(LA_TRAME )[2:4]
print("L'identifiant de l'emetteur est : ", id)

#octet 2H = canal d'émission
canal = hex(LA_TRAME)[4]
print("L'acquisition est faites sur le canal : ", int(canal) + 1) #+1 pour␣
 ↪avoir le vrai canal

#récupération des données en hexa (octets 2B à 3 inclus)
octet_2 = int(hex(LA_TRAME)[5], 16)
octet_3H = hex(LA_TRAME)[6]
```

```
octet_3B = hex(LA_TRAME)[7]

print("Octet 2 bas : ", octet_2, "\nOctet 3 haut :", octet_3H, "\nOctet 3 bas:␣
 ↪", octet_3B)

Temp_F = (int(octet_3B) * 256 + int(octet_3H) * 16 + int(octet_2) - 900) / 10
print("Température en °F : ", Temp_F)

#On transforme la température en °C

Temp_C = (Temp_F - 32) / 1.8
print("Température en °C : ", round(Temp_C, 1))
```

```
L'identifiant de l'emetteur est :  33
L'acquisition est faites sur le canal :  3
Octet 2 bas :  9
Octet 3 haut : 0
Octet 3 bas:  7
Température en °F :  90.1
Température en °C :  32.3
```

[ ]: