Projet : ESCAPE NO GAME

Livrable 4

Présentation de l'équipe :

- De Barros Barbosa Flavio
- Kadiri Shahineze
- Goutier Galaad
- Bohic Evan
- Rabaux Malo
- Letellier Baptiste

Rappel : Chaine de transmission

In [ ]:

```
# librairies
import numpy as np                  #On importe les bibliothèques permettant respectivement : faire des opérations impossible en python nati
import matplotlib.pyplot as plt                                  # faire des graphiques
import binascii                                                  # convertir du binaire en ascii et inversement
from scipy.signal import butter, filtfilt                        # permet de faire le filtre passe bas

def filtre_passe_bas(signal, frequence_coupure=300, frequence_echantillonnage=Fe, ordre=5):  #filtre passe bas (pour la démodulation A
    nyquist = 0.5 * frequence_echantillonnage
    normalise_coupure = frequence_coupure / nyquist
    b, a = butter(ordre, normalise_coupure, btype='low', analog=False)
    return filtfilt(b, a, signal)
```

Phase 1 :

Conversion ASCII - Binaire :

In [116]:

```
# ----------------Conversion a ASCII Binaire--------------------


#Initialisation du message textuel + conversion en binaire
Texte = "Mission accomplie, envoyez moi du renfort!" #message envoyé
tableau_bits = Texte.encode()
entier_binaire = int.from_bytes(tableau_bits,"big") #transforme le texte en entier
binaire = bin(entier_binaire)[2:]  #transforme l'entier en binaire (sous forme de string) tout en retirant les 2 première caractère (qui sont les

print("Le message initiale est :", Texte)
print("_____________________________________________________")          #on affiche le message initial et le message converti en binai
print("Le message initial en binaire est :", binaire)

#-------------------------------------------------------------------------
```

```
Le message initiale est : Mission accomplie, envoyez moi du renfort!
_____________________________________________________
Le message initial en binaire est :
100110101101001011100110111001101101001011011110110111000100000011000010110001101100011011011110110110101110000011011000110100101100101
110000100000011001010110111001110110011011110111110010110010101111010001000000110110101101111011010010010000001100100011101010010000001
001100101011011100110011001100110111101110010011101000010000
```

Encodeur

In [118]:

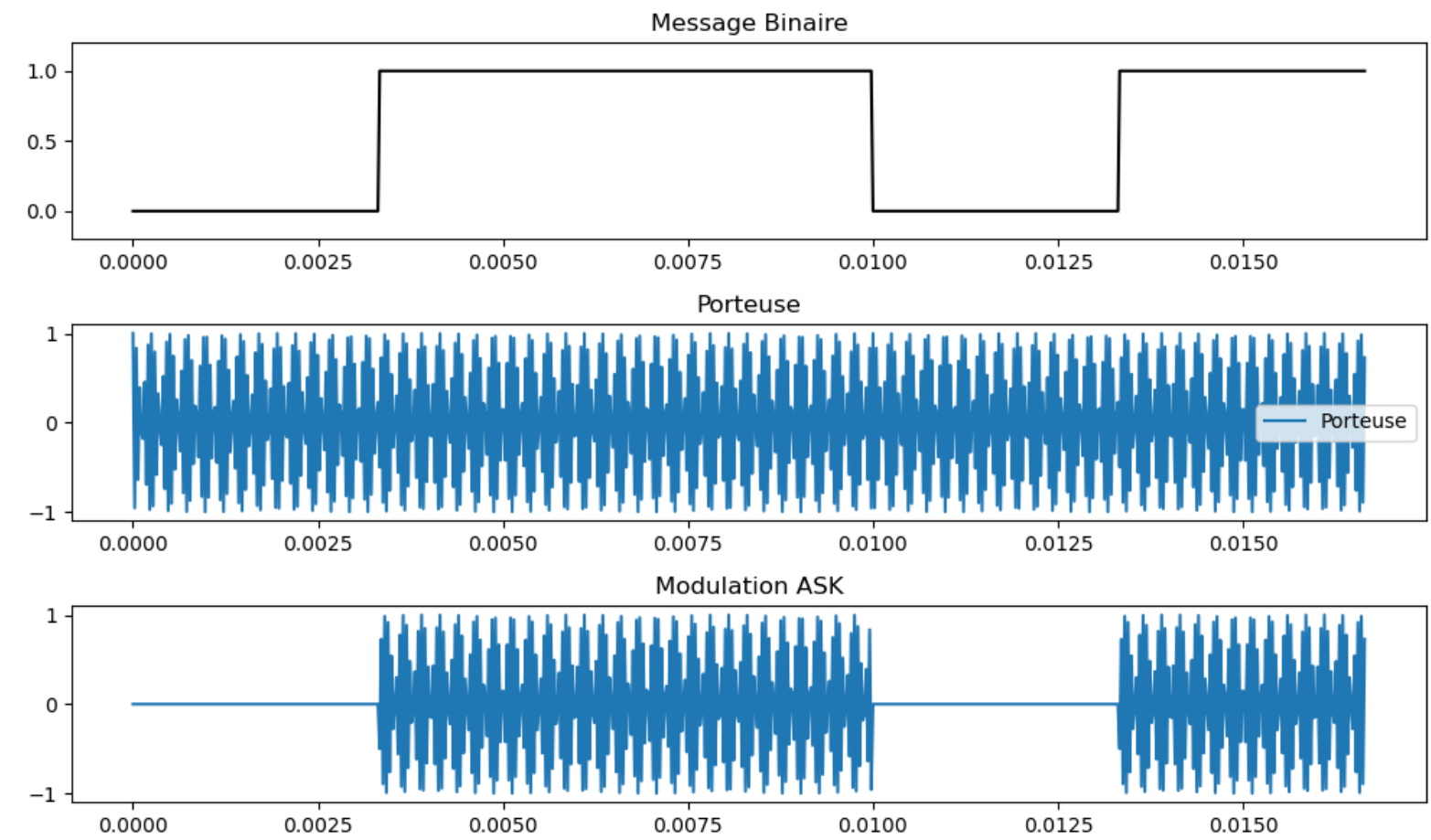
```
# Encodeur :  #on utilise l'encodage Manchester et prendrons comme norme 1=01 et 0=10

binaire_encoder=""   #création de la variable qui va stocker le binaire encodé

for element in binaire:

    if element=="1":
        binaire_encoder+="01"       #on transforme les "1" en "01" et les "0" en "10"

    else:
        binaire_encoder+="10"

print("___________________________________________________________")
print("Le message en binaire encodé:",binaire_encoder)     #on affiche le message binaire une fois encodé
print("___________________________________________________________")
```

```
________________________________________________
Le message en binaire encodé:
01101001011001100101100110100110010101101001011001010110100101100101100110100110010110010101011001011001010110101001101010101010010110101
0011001011010100101100101101010010110010110010101011001011001011001100101011010101001011001011010100101100110100110010110100110011010
010110101010011010101010100101101001100110010110010101101001010110010110100101100101010110010101011010011001011010011001100101010110011
0011010101010100101100101100110010110010101011001011001101001101001101010101001011010011010100101011001100110100110101010101001010110
101001011010011001100101100101011010010110100101101001011001010101100101011010011010010101100110101010011010101001
________________________________________________
```

Modulation ASK

In [125]:

```
#Modulation
M = []  # Message binaire encodé M
for element in binaire_encoder:
    M.append(int(element))

Fe = 44100  # Fréquence d'échantillonnage
Fp = 20000  # Fréquence de l'onde porteuse
baud = 300  # Débit souhaité en bit/s


Nbits = len(M)  # Nombre de bits
Ns = int(Fe / baud)  # Nombre d'échantillons par bit
N = Ns * Nbits  # Nombre total d'échantillons

# Génération du message binaire dupliqué
M_duplique = np.repeat(M, Ns)

# Génération du vecteur temps
t = np.arange(N) / Fe  # Assure une correspondance exacte avec M_duplique

# Génération de la porteuse
P = np.cos(2 * np.pi * Fp * t)  # Porteuse unique

# Modulation ASK
ASK = M_duplique * P

# Affichage des résultats
plt.figure(figsize=(10, 6))  #on définit la taille des graphiques

# Affichage du message binaire
plt.subplot(3, 1, 1)
plt.plot(t[:Ns*5], M_duplique[:Ns*5], 'k')        #ici nous n'affichons que les 5 premiers bits sur les 3 graphiques pour plus de lisibilité
plt.title("Message Binaire")
plt.ylim(-0.2, 1.2)

# Affichage de la porteuse
plt.subplot(3, 1, 2)
plt.plot(t[:Ns*5], P[:Ns*5], label="Porteuse")
plt.title("Porteuse")
plt.legend()

# Affichage du signal ASK
plt.subplot(3, 1, 3)
plt.plot(t[:Ns*5], ASK[:Ns*5])
plt.title("Modulation ASK")

plt.tight_layout()
plt.show()
```

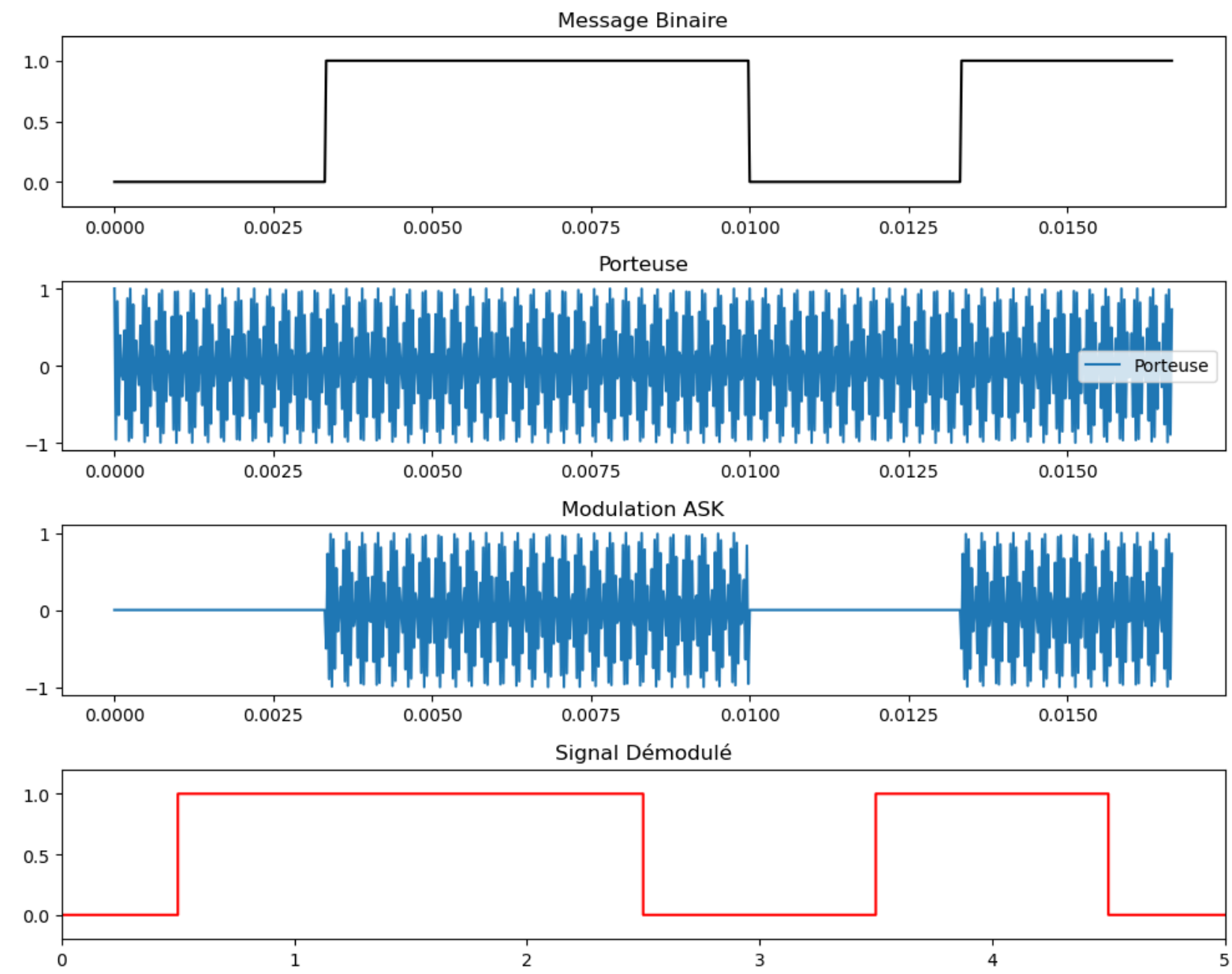


Phase 2 :

Démodulation ASK :

In [104]:

```
#Démodulation
ASK_demod = ASK * P  # Multiplication par la porteuse



signal_filtre = filtre_passe_bas(ASK_demod) #  filtre passe-bas pour supprimer le bruit haute fréquence

# Seuil dynamique à partir de la moyenne du signal filtré
threshold = np.mean(signal_filtre)
bits_recuperes = signal_filtre[Ns//2::Ns] > threshold

# Correction de la taille des bits récupérés
bits_recuperes = bits_recuperes[:Nbits]

# Affichage des résultats
plt.figure(figsize=(10, 8)) #on définit la taille des graphiques

# Affichage du message binaire

plt.subplot(4, 1, 1)
plt.plot(t[:Ns*5], M_duplique[:Ns*5], 'k')   #ici nous n'affichons que les 5 premiers bits sur les 3 graphiques pour plus de lisibilité
plt.title("Message Binaire")
plt.ylim(-0.2, 1.2)

# Affichage de la porteuse
plt.subplot(4, 1, 2)
plt.plot(t[:Ns*5], P[:Ns*5], label="Porteuse")
plt.title("Porteuse")
plt.legend()

# Affichage du signal ASK
plt.subplot(4, 1, 3)
plt.plot(t[:Ns*5], ASK[:Ns*5])
plt.title("Modulation ASK")



# Affichage du signal Démodulé
plt.subplot(4, 1, 4)
plt.step(range(len(bits_recuperes)), bits_recuperes, where='mid', color='r')  #Ce graphique ne sera pas aligné avec les autres car n'utilise p
plt.title("Signal Démodulé")                                                  # unité pour l'abscisse
plt.xlim(0, 5)
plt.ylim(-0.2, 1.2)
plt.tight_layout()
plt.show()

# Affichage des bits récupérés
print("Bits récupérés :", bits_recuperes.astype(int))
```



Bits récupérés : [0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0
1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0
1 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1
0 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1
1 0 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0
1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 0 1 0
1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0
1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0
1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1
0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1
0 1 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0
1 1 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 0
0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1
0 0 1 1 0 1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 0 1
1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0
1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0
0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 1 0
1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0 1 0
1 0 0 1]

Décodeur :

In [106]:

```
# Décodeur :
binaire_decoder=""
binaire_recup=""
for element in bits_recuperes:      #on transforme notre liste de bits récupéré en string
    if element==False:
        binaire_recup+="0"
    else:
        binaire_recup+="1"




for i in range(0,len(binaire_recup)-1,2):
    if binaire_recup[i]+binaire_recup[i+1]=="01":      #on décode la codage Manchester
        binaire_decoder+="1"
    else:
        binaire_decoder+="0"

print("Le message binaire reçu une fois décodé donne:",binaire_decoder)  #On affiche le message binaire décoder

print("_____________________________________________________")

print("Le message reçu est le bon:",binaire_decoder==binaire) #On affiche si le signal message binaire reçu est le même que celui envoyé

print("_____________________________________________________")
```

```
Le message binaire reçu une fois décodé donne:
1001101011010010111001101110011011010010110111101101110001000000110000101100011011000110110111101101101011100000110110001101001011001010
1100001000000110010101101110011101100110111101111001011001010111101000100000011011010110111101101001001000000110010001110101001000000110
0011001010110111001100110011011110111001001110100001000001
_____________________________________________________
Le message reçu est le bon: True
_____________________________________________________
```

Conversion Binaire - ASCII :

In [108]:

```
#-----------------Conversion Binaire ASCII---------------------------------------

#on transforme le binaire en ascii (en texte)

binaire_entier = int(binaire_decoder, 2)
nombre_bits = binaire_entier.bit_length() + 7 // 8
tableau_binaire = binaire_entier.to_bytes(nombre_bits, "big")
texte_ascii = tableau_binaire.decode()

print("Le message binaire reçu est :", binaire)   #on affiche le message binaire reçu
print("    ")
print("Le message décodé est :", texte_ascii)     #on affiche le message reçu sous forme de texte
print("_____________________________________________________")
#-------------------------------------------------------------------------------
```

```
Le message binaire reçu est :
1001101011010010111001101110011011010010110111101101110001000000110000101100011011000110110111101101101011100000110110001101001011001010
1100001000000110010101101110011101100110111101111001011001010111101000100000011011010110111101101001001000000110010001110101001000000110
0011001010110111001100110011011110111001001110100001000001

Le message décodé est : Mission accomplie, envoyez moi du renfort!

_____________________________________________________
```

In [ ]: