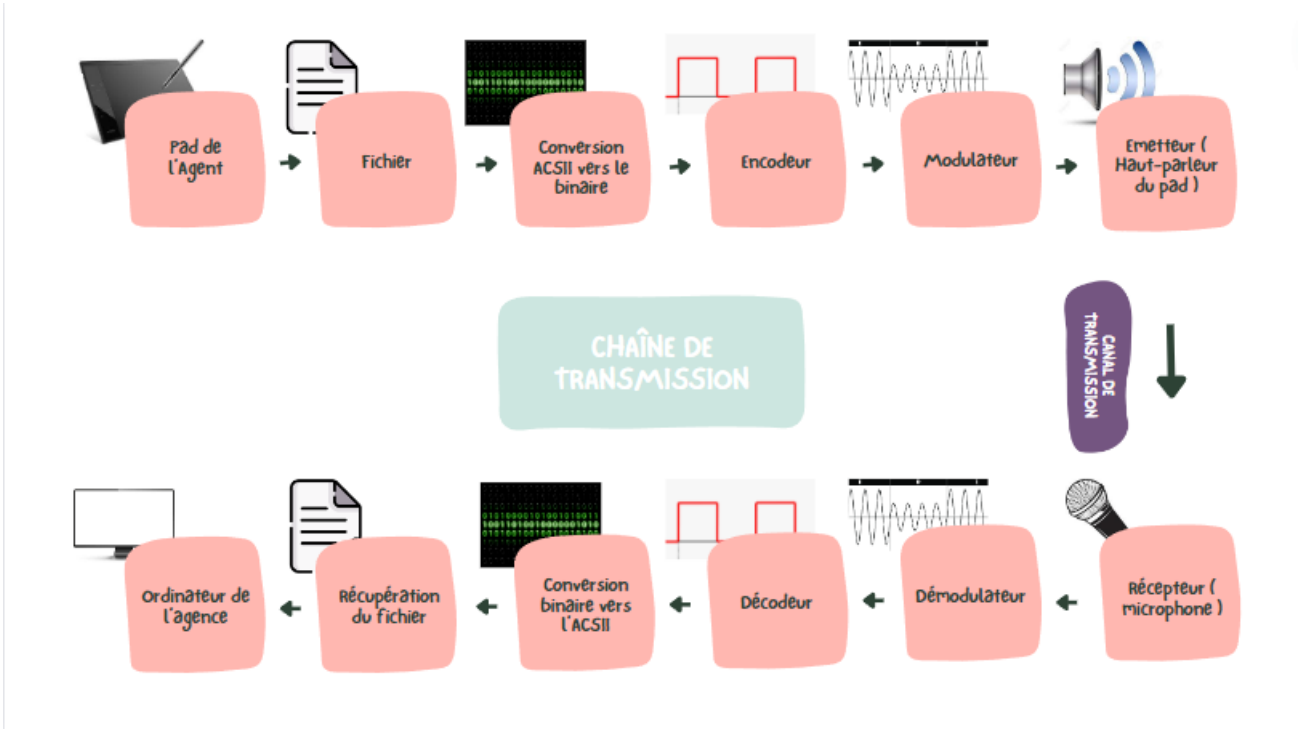
Livrable 4 : Modélisation numérique

Equipe 5: Glairot Valentin, Ledoux Bastien, Petit-Colboc Julien, Robin Arthur, Briant Lucas, Diagne Mame Thierno Birahim.



Le code que nous allons présenter dans ce livrable représente la chaine de transmission ci-dessus. Il prend en entrée un fichier CSV et renvoie en sortie le message envoyé.

In [1]:
```
import numpy as np
import soundfile
import pathlib

emplacementFichier="Message.csv"
Fe = 40000
baud = 300
fp1 = 17000
fp2 = 19000
```

On commence par importer les différentes bibliothèques nécessaires pour la suite de notre programme. On déclare ensuite les différentes valeurs nécessaires. La fréquence d'échantillonnage, la vitesse de transmission ainsi que les différentes fréquences des porteuses.

Conversion Decimal-Binaire

In [2]:
```
# Conversion Decimal-Binaire
def DecimalVersBinaire(n):
    r=''
    while n>0:
        r+=(str(n%2))
        n=n//2
    while len(r)<8:
        r+='0'
    return r[::-1]
```

Dans cette fonction, on convertit de l'ASCII en binaire, en effet, la fonction prend en paramètre "n" et renvoie le code binaire. On ajoute le reste de la division de n par 2 (représentation binaire du bit le moins significatif) converti en chaîne de caractères 'r'. Puis on divise n par 2 (décalage des bits vers la droite pour obtenir le prochain bit binaire). Une fois la conversion effectuée, on ajoute des zéros à gauche pour obtenir une représentation binaire de 8 bits. Enfin, on inverse pour avoir les bits de gauche à droite. Cela retourne la représentation binaire sur 8 bits du nombre décimal donné en paramètre.

Encodage Manchester

In [3]:
```
# Encodage Manchester
def EncodageManchester(r):
    messageCode=[]
    for nb in r:
        if nb==1:
            messageCode.append(0)
            messageCode.append(1)
        else:
            messageCode.append(1)
            messageCode.append(0)
    return messageCode
```

Cette fonction reprend en paramètre le résultat de la fonction précédente. On parcourt la chaîne de caractère "r", quand nous rencontrons un "1", le programme ajoute à "messsageCode" les valeurs 0 et 1. Si la valeur rencontré e est un 0, on ajoute 1 et 0. Le message renvoyé est donc codé en Manchester.

Modulation ASK

In [4]:
```
# Modulation ASK
def ModulationASK(signal):
    Ns =int(Fe/baud)
    N =Ns*len(signal)

    M_duplique=np.repeat(signal,Ns)
    t=np.linspace(0,N/Fe,N)

    Porteuse =np.sin(2*np.pi*fp1*t)

    ASK = Porteuse*M_duplique
    return ASK,Porteuse,N,Ns
```

On duplique chaque échantillon du signal pour correspondre au nombre d'échantillons par symbole Ns, on crée un vecteur de temps t. Ensuite, on génère un signal de porteuse sinusoïdale à la fréquence fp1 en multipliant par 2π pour obtenir la phase en radians. Enfin, on multiplie le signal de porteuse par le signal dupliqué, ce qui modifie l'amplitude de la porteuse en fonction des échantillons du signal d'entrée et nous permet d'obtenir le signal modulé.

Modulation FSK

In [5]:
```
# Modulation FSK
def ModulationFSK(signal):
    Ns =int(Fe/baud)
    N =Ns*len(signal)

    M_duplique=np.repeat(signal,Ns)
    t=np.linspace(0,N/Fe,N)

    P1 = np.sin(2*np.pi*fp1*t)
    P2 = np.sin(2*np.pi*fp2*t)

    FSK = []

    for i in range (0,len(t)):

        if M_duplique[i] > 0:
            FSK.append(P1[i])

        if M_duplique[i] <= 0:

            FSK.append(P2[i])
    return FSK,P1,P2,N,Ns
```

Dans la fonction ci-dessus, on duplique chaque symbole du signal d'entrée pour obtenir un signal étendu. On génère ensuite un signal sinusoïdal modulé en amplitude par le temps. Par la suite, nous vérifions si le signal dupliqué est supérieur ou égal à 0 à l'instant i.

- Si oui, le signal est modulé à la fréquence fp1, et la valeur correspondante de P1 est ajoutée à la liste FSK.
- Si non, cela signifie que le signal est modulé à la fréquence fp2, et la valeur correspondante de P2 est ajoutée à la liste FSK. On retourne ensuite FSK, les deux porteuses, le nombre d'échantillon aisni que le nombre de d'échantillons nécessaires.

Démodulation ASK

In [6]:
```
# Démodulation ASK
def DemodulationASK(signal,Porteuse,Ns):
    Produit=signal*Porteuse
    Res=[]
    for i in range(0,N,Ns):
        if np.trapz(Produit[i:i+Ns])>0:
            Res.append(1)
        else:
            Res.append(0)
    return Res
```

On commence par faire le produit du signal par la porteuse pour commencer à démoduler puis on itère à travers le signal avec un pas de "Ns". Par la suite, "np.trapz" est utilisé pour effectuer une intégration numérique (trapézoïdal) sur la fenêtre actuelle du produit signal * porteuse. Si la somme des valeurs de la fenêtre est positive, le bit démodulé est considéré comme 1, sinon, il est considéré comme 0. Le résultat est ajouté à la liste "Res".

Démodulation FSK

In [7]:
```
# Démodulation FSK
def DemodulationFSK(signal,P1,P2,Ns):
```

```
    P1DM=signal*P1
    P2DM=signal*P2
    Res1=[]
    Res2=[]

    for i in range(0,N,Ns):
        Res1.append(np.trapz(P1DM[i:i+Ns]))
        Res2.append(np.trapz(P2DM[i:i+Ns]))
        Res=[]
        for i in range(len(Res1)):
            if abs(Res1[i])-abs(Res2[i])>0:
                Res.append(1)
            else:
                Res.append(0)
    return Res
```

On définit la fonction "DemodulationFSK" qui prend en paramètre "signal" qui représente le signal encodé et modulé précédemment, "P1" et "P2" qui sont les deux porteuses, "N" le nombre total de bits dans le signal et enfin "Ns" le nombre de bits par échantillon. On établit la première porteuse démodulée en multipliant le signal par la première porteuse, puis nous faisons de même pour la seconde porteuse. Les deux listes "Res1" et "Res2" sont des listes vides qui acceuillerons les résultats Ensuite, la boucle for itère sur une plage de valeurs allant de 0 à N (exclue) avec un pas de Ns. À chaque itération, les fonctions "np.trapz()" sont utilisées pour calculer l'intégrale numérique des tranches de données "P1DM" et "P2DM", qui sont ensuite ajoutées aux listes "Res1" et "Res2". Une deuxième boucle "for" itère sur les indices de la liste Res1 (ou Res2, car elles ont la même longueur). Pour chaque élément de Res1, la différence absolue entre les éléments correspondants de Res1 et Res2 est calculée. Si cette différence est supérieure à 0, alors un 1 est ajouté à la liste Res ; sinon, un 0 est ajouté.

Décodage Manchester

In [8]:
```
# Décodage Manchester
def DecodageManchester(signal):
    messageDecode=[]
    for i in range(0,len(signal),2):
        if signal[i]==1 and signal[i+1]==0:
            messageDecode.append(0)
        else:
            messageDecode.append(1)
    return messageDecode
```

La boucle "for" itère sur les indices du signal en sautant de 2 à chaque itération (0, 2, 4, ...). Cela divise le signal en paires d'éléments consécutifs. À chaque itération, la condition "if" signal[i]==1 and signal[i+1]==0" est vérifiée. Si cette condition est vraie, alors un 0 est ajouté à la liste "messageDecode", sinon un 1 est ajouté.

Conversion Binaire-ASCII

In [9]:
```
#Conversion Binaire-ASCII
def BinaireVersASCII(message):
    bin_data = ""
    for elem in message:
        bin_data += str(elem)
    data_reçu =' '
    def BinaryToDecimal(binary):
        decimal, i, n = 0, 0, 0
        while(binary != 0):
            dec = binary % 10
            decimal = decimal + dec * pow(2, i)
            binary = binary//10
            i += 1
        return (decimal)
    for i in range(0, len(bin_data), 8):
        temp_data = int(bin_data[i+1:i+8])
        decimal_data = BinaryToDecimal(temp_data)
        data_reçu = data_reçu + chr(decimal_data)
    return data_reçu
```

Dans ce programme, la boucle "for elem in message" : itère sur chaque élément de la séquence binaire "message" et les ajoute à la chaîne "bin_data". On déclare ensuite une fonction interne pour convertir un nombre binaire en décimal. On initialise "décimal", "i" et "n". La boucle "while" itère tant que le nombre binaire n'est pas égal à zéro. À chaque itération, le dernier chiffre du nombre binaire est extrait "dec" et ajouté à la valeur décimale "decimal" en utilisant la formule "decimal = decimal + dec * pow(2, i)". Le nombre binaire est ensuite divisé par 10 et le compteur i est incrémenté. La boucle "for i in range(0, len(bin_data), 8):" itère sur la chaîne binaire par groupes de 8 bits. Ensuite "temp_data" extrait un groupe de 7 bits à partir de la position i+1 et le convertit en entier. On fait ensuite un appel de fonction qu'il nous permet de convertir les bits en décimales. Pour la dernière étape, on convertit le nombre décimal en caractère ASCII à l'aide de la fonction "chr" et l'ajoute à la chaîne data_reçu. Pour finir, la fonction renvoie "data_reçu" qui est une chaîne de caractère.

Vérification d'erreurs potentielles

In [10]:
```
# Vérification d'erreur
def VerificationErreurs(message):
    matrice=[]
    for mot in message:
```

```
        for caractere in mot :
            matrice.append((DecimalVersBinaire(ord(caractere))))
    verificationErreurs=[]
    for caractere in matrice:
        resultat=0
        for bit in caractere:
            resultat+=int(bit)
        verificationErreurs.append(resultat%2)
    for i in range(8):
        resultat=0
        for caractere in matrice:
            resultat+=int(caractere[i])
        verificationErreurs.append(resultat%2)
    return verificationErreurs
```

Cette fonction parcours l'entièreté du message puis chaque caractère pour chaque mot. On ajoute ensuite à la matrice l'équivalent binaire, en faisant un appel de fonction. La boucle "for" parcourt chaque caractère binaire dans la matrice. Pour chaque bit du caractère, la somme est calculée. La parité (somme cumulée modulo 2) est ajoutée à la liste verificationErreurs. On calcule ensuite la parité pour chaque bit, une deuxième boucle parcourt chaque position de bit. Pour chaque position de bit, la somme cumulée des bits à cette position pour tous les caractères est calculée. La parité est ajoutée à la liste "verificationErreurs".

Génération de la trame

In [11]:
```
# Génération de la trame
def GenerationTrame(expediteur,destinataire):
    extensionFichier=np.array([pathlib.Path(emplacementFichier).suffix])
    message=np.genfromtxt(emplacementFichier,dtype=str)
    verificationErreurs=VerificationErreurs(message)
    trame=np.concatenate([["DEBUT|"],[len(verificationErreurs)-8],["|"],extensionFichier,["|"],[expediteur],["|"],[destinataire],["|"],message,["|"],verification
    return trame
```

On construit la trame en concaténant plusieurs éléments tels que "DEBUT|", la longueur du message, l'extension du fichier, l'expéditeur, le destinataire, le message, les erreurs éventuelles et "|FIN" dans un seul tableau. Puis on renvoie la trame construite.

Récupération des composants

In [12]:
```
# Récupération des composants
def RecuperationComposant(trame,n):
    composant=''
    nbComposant=1
    for caractere in trame:
        if caractere=='|':
            nbComposant+=1
        elif nbComposant==n:
            composant+=caractere
    return composant
```

On commence par initialiser "composant" et "nbComposant". Une boucle parcourt chaque caractère de la trame. Si le caractère est un '|', le compteur "nbComposant" est incrémenté de 1. Si le compteur "nbComposant" est égal à la valeur de "n" spécifiée en paramètre, alors le caractère est ajouté au composant.

Choix de l'expéditeur et du destinataire

In [13]:
```
# Choix de l'expéditeur et du destinataire
expediteur=input("Entrer le nom de l'expéditeur")
destinataire=input("Entrer le nom du destinataire")

trame=GenerationTrame(expediteur,destinataire)
```

```
Entrer le nom de l'expéditeura
Entrer le nom du destinataireb
```

On commence par demander à l'utilisateur l'expéditeur et le destinataire.

Conversion en binaire de chaque caractère du message

In [14]:
```
# Conversion en binaire de chaque caractère du message
trameBinaire=[]
for composant in trame:
    for caractere in composant:
        caractereDecimal=ord(caractere)
        caractereBinaire=DecimalVersBinaire(caractereDecimal)
        for n in caractereBinaire:
            trameBinaire.append(int(n))
```

On génère ensuite la trame en faisant un appel de fonction. Une fois cela fait, on convertit les caractères de la trame en binaire. Une boucle parcourt chaque composant de la trame. À l'intérieur de cette boucle, une autre boucle parcourt chaque caractère du composant. Chaque caractère est converti en son équivalent décimal à l'aide de la fonction ord(caractère). Ensuite, le caractère décimal est converti en binaire à l'aide de la fonction "DecimalVersBinaire". Enfin, chaque bit de la séquence binaire est ajouté à la liste "trameBinaire" sous forme d'entiers.

Encodage du message binaire

In [15]:
```
# Encodage du message binaire
trameCode=EncodageManchester(trameBinaire)
```

Choix de la modulation

In [16]:
```
while True:
    try:
        nbModulation = int(input("Veuillez entrer un choix valide : 1=ASK 2=FSK "))
        if nbModulation == 1 or nbModulation == 2:
            break  # Sort de la boucle si l'entrée est valide
        else:
            print("Choix non valide. Veuillez entrer 1 pour ASK ou 2 pour FSK.")
    except ValueError:
        print("Veuillez entrer un nombre entier.")

if nbModulation==1:
    # Modulation ASK du message code
    signal,Porteuse,N,Ns=ModulationASK(trameCode)
if nbModulation==2:
    # Modulation FSK du message code
    signal,P1,P2,N,Ns=ModulationFSK(trameCode)
```

```
Veuillez entrer un choix valide : 1=ASK 2=FSK 1
```

Nous demandons à l'utilisateur le type de modulation qu'il souhaite. Si l'utilisateur choisit 1 (ASK), la fonction "ModulationASK" est appelée avec "messageCode" en tant que paramètre, et les résultats sont stockés dans les variables "signal", "Porteuse", "N", et "Ns". Si l'utilisateur choisit 2 (FSK), la fonction "ModulationFSK" est appelée avec "messageCode" en tant que paramètre, et les résultats sont stockés dans les variables "signal", "P1", "P2", "N", et "Ns". Par contre, si l'utilisateur ne rentre pas 1 ou 2, le programme renverra un message d'erreur.

Émission et réception

In [17]:
```
# Emission
soundfile.write("Signal.wav",signal,Fe)

# Réception
fichier_audio=soundfile.read("Signal.wav")[0]
```

Démodulation du fichier reçu

In [18]:
```
# Démodulation du fichier reçu
if nbModulation==1:
    trameDemodule=DemodulationASK(fichier_audio,Porteuse,Ns)
if nbModulation==2:
    trameDemodule=DemodulationFSK(fichier_audio,P1,P2,Ns)
```

Dans la première partie, on définit la démodulation en fonction du choix de l'utilisateur, si l'utilisateur a choisi la modulation ASK (1), la fonction "DemodulationASK" est appelée avec le fichier audio reçu, la porteuse, et Ns. Si l'utilisateur a choisi la modulation FSK (2), la fonction "DemodulationFSK" est appelée avec les paramètres appropriés.

Decodage du signal démodulé

In [19]:
```
# Decodage du signal démodulé
trameDecode=DecodageManchester(trameDemodule)
```

Par la suite, on appelle la fonction "DecodageManchester" et on stocke dans "trameDecode".

Conversion ASCII du message décodé

In [20]:
```
# Conversion ASCII du message décodé
trameASCII=BinaireVersASCII(trameDecode)
```

Nous faisons la même chose pour convertir le message décodé de ASCII à binaire.

Affichage du message ASCII

In [21]:
```
# Affichage du message ASCII
print(trameASCII)
print("Le message reçu est :", RecuperationComposant(trameASCII,6))
```

```
DEBUT|22|.csv|a|b|SOS_J'ai_Besoin_D'aide|0100101000010010001010001000011|FIN
Le message reçu est : SOS_J'ai_Besoin_D'aide
```

On affiche ensuite le message reçu.

Vérification d'erreurs

In [22]:
```
# Vérification d'erreurs
verificationErreurs=VerificationErreurs(RecuperationComposant(trameASCII,6))
nombreErreurs=0
for i in range(len(verificationErreurs)):
    if verificationErreurs[i]!=int(RecuperationComposant(trameASCII,7)[i]):
        nombreErreurs+=1
print("Le système de vérification d'erreurs a détécté",nombreErreurs,"erreur(s).")
```

```
Le système de vérification d'erreurs a détécté 0 erreur(s).
```

La fonction "VerificationErreurs" est appelée avec le composant du message correspondant pour détecter les erreurs. Les erreurs sont comptées en comparant les bits de parité avec les bits du septième composant du message. Le nombre d'erreurs détectées est stocké dans la variable "nombreErreurs" Pour finir, on affiche à l'écran le nombre d'erreurs détectées.

Envoi d'un accusé de réception

In [23]:
```
# Envoi d'un accusé de réception
AR=["RECEPTION|",destinataire,"|",expediteur,"|REUSSIE"]
ARBinaire=[]
for composant in AR:
    for caractere in composant:
        caractereDecimal=ord(caractere)
        caractereBinaire=DecimalVersBinaire(caractereDecimal)
        for n in caractereBinaire:
            ARBinaire.append(int(n))
ARCode=EncodageManchester(ARBinaire)
if nbModulation==1:
    signal,Porteuse,N,Ns=ModulationASK(ARCode)
if nbModulation==2:
    signal,P1,P2,N,Ns=ModulationFSK(ARCode)
soundfile.write("AR.wav",signal,Fe)
AR_audio=soundfile.read("AR.wav")[0]
if nbModulation==1:
    ARDemodule=DemodulationASK(AR_audio,Porteuse,Ns)
if nbModulation==2:
    ARDemodule=DemodulationFSK(AR_audio,P1,P2,Ns)
ARDecode=DecodageManchester(ARDemodule)
ARASCII=BinaireVersASCII(ARDecode)
print(ARASCII)
```

```
RECEPTION|b|a|REUSSIE
```

Pour chaque composant de la liste AR, chaque caractère est converti en binaire à l'aide de la fonction DecimalVersBinaire puis les chiffres binaires de chaque caractère sont ajoutés à la liste ARBinaire. Cette liste est ensuite convertie en un signal encodé à l'aide de la fonction EncodageManchester puis en fonction de la valeur de nbModulation on utilisera soit la modulation ASK ou FSK. Après ça le signal modulé est écrit dans un fichier audio "AR.wav" en utilisant soundfile.write et lu grâce à soundfile.read. Le signal est ensuite démodulé et décodé pour retrouver la séquence binaire originale. Cette même séquence est convertie en ASCII à l'aide de la fonction BinaireVersASCII et enfin est affichée en tant que chaîne de caractères ASCII.

Résultat final

In [24]:
```
print("Le message reçu est :", RecuperationComposant(trameASCII,6))
print("Le système de vérification d'erreurs a détécté",nombreErreurs,"erreur(s).")
print("Accusé de réception :", ARASCII)
```

```
Le message reçu est : SOS_J'ai_Besoin_D'aide
Le système de vérification d'erreurs a détécté 0 erreur(s).
Accusé de réception :  RECEPTION|b|a|REUSSIE
```

In [ ]: